



# Wacs Programming Guide

Second Edition

for WACS 0.8.1

B "Beaky" King

Published 22nd August 2008

---

# Wacs Programming Guide

by B "Beaky" King

for WACS 0.8.1

Published 22nd August 2008

Copyright © 2006, 2007, 2008 B King

## Abstract

WACS is a tool for building Adult Web Sites; it is equally suitable for managing a private collection or building a commercial web site. It has many best of breed features including dynamic filtering, model catalogs, automatic download and powerful search engine. It comes with a powerful API (application programming interface) implemented in both Perl and PHP5 languages to allow web developers to leverage it's facilities from their own programs.

This book describes the application programming interface provided by WACS, and how to utilise it from perl and Php languages. It provides an extensive introductory tutorial with a large number of worked example programs as well as a complete API reference manual. Additionally it provides a schema reference for the WACS database tables as understanding the fields available to you is central to writing programs that utilitise it. The intended audience is web developers and WACS site managers who wish to tailor an existing WACS installation to meet their precise requirements; people merely wishing to use or manage an existing WACS installation may well find the default configurations provided suffice.

---

---

# Table of Contents

I. WACS API Programming Tutorial .....	1
1. Introduction .....	2
Overview .....	2
About This Book .....	2
About The Examples .....	2
2. Basics: Getting Started .....	3
Outline .....	3
A First WACS Program .....	3
Modules: Importing .....	3
Configuration And Security .....	4
Initialising Database Connection .....	4
Fetching Some Records .....	6
Showing The Results .....	7
Finishing Off .....	9
Putting It All Together .....	9
Running MySimple .....	12
Reviewing The First Program .....	12
3. Using More Database Fields .....	13
Adding Model Icons .....	13
More Model Information .....	14
Using HTML tables .....	15
Adding The Model Details .....	17
Adding Other Icons .....	19
Improving Error Reporting .....	21
4. Set Display Routines .....	24
About Set Display .....	24
Sets: The Basic Bones .....	24
Adding Icons .....	27
Making The Text More Readable .....	28
Connecting Sets And Models .....	30
Understanding The Data Architecture .....	30
Using Relationships With Assoc .....	30
An Example Using Assoc .....	31
5. The User Interface Toolkit .....	35
Introducing WacsUI .....	35
Including WacsUI support .....	35
WacsUI: DescribeHer .....	35
The addkeyicons function .....	36
WacsUI: Other Functions .....	36
Conclusions .....	36
II. WACS API Programming Reference .....	38
6. WACS API: Core Module .....	39
Core Module: Summary .....	39
Core Module: Reference .....	39
7. WACS API: User Interface Module .....	62
User Interface Module: Summary .....	62
User Interface Module: Reference .....	62
8. WACS API: Standard Components Module .....	74
Standard Components Module: Summary .....	74
Standard Components Module: Reference .....	74
9. WACS API: Identification Module .....	84

Identification Module: Summary .....	84
III. WACS Database Schema .....	86
10. Schema Reference: Sets .....	87
Sets: Schema SQL .....	87
Sets: Defined Values .....	88
11. Schema Reference: Assoc .....	92
Assoc: Schema SQL .....	92
Assoc: Defined Values .....	92
12. Schema Reference: Idmap .....	93
Idmap: Schema SQL .....	93
Idmap: Defined Values .....	93
13. Schema Reference: Models .....	95
Models: Schema SQL .....	95
Models: Defined Values .....	96
14. Schema Reference: Download .....	99
Download: Schema SQL .....	99
Download: Defined Values .....	99
15. Schema Reference: Photographer .....	101
Photographer: Schema SQL .....	101
Photographer: Defined Values .....	101
16. Schema Reference: Tag .....	104
Tag: Schema SQL .....	104
Tag: Defined Values .....	104
17. Schema Reference: Vendor .....	105
Vendor: Schema SQL .....	105
Vendor: Defined Values .....	106
18. Schema Reference: Conn .....	107
Conn: Schema SQL .....	107
Conn: Defined Values .....	107
19. Schema Reference: Keyword .....	108
Keyword: Schema SQL .....	108
Keyword: Defined Values .....	108
Index .....	109

---

## List of Tables

1. The Key WACS Modules .....	38
6.1. Function Summary: Core Module .....	39
7.1. Function Summary: User Interface Module .....	62
8.1. Function Summary: Standard Components Module .....	74
9.1. Function Summary: Identification Module .....	85
10.1. stype: Type of Set: defined values .....	88
10.2. sstatus: Status of Set: defined values .....	88
10.3. sauto: Automatic Update of Set Allowed?: defined values .....	89
10.4. srating: Overall Rating For The Set: defined values .....	89
10.5. stechqual: Technical Quality Rating For The Set: defined values .....	89
10.6. svary: Unusualness Rating For The Set: defined values .....	89
10.7. sformat: Format of the File(s) In The Set: defined values .....	90
10.8. sidlogo: Presence of Burnt-in Logo: defined values .....	90
10.9. serrors: Presence of Known Errors: defined values .....	90
10.10. scatflag: Generalised type of the set: defined values .....	90
10.11. slocation: generalised description of locations: recommended values .....	91
10.12. suscattr: how to generate the 18 USC 2257 declaration: defined values .....	91
11.1. astatus: association status: defined values .....	92
12.1. istatus: idmap status: defined values .....	93
12.2. iactive: model activity status as this identity: defined values .....	93
12.3. isite: Some recommended site abbreviations: recommended values .....	94
13.1. mstatus: model record status: defined values .....	96
13.2. mrating: model rating: defined values .....	96
13.3. mpus: model's normal pubic hair style: defined values .....	96
13.4. mflag: special marking flag for models: defined values .....	97
13.5. model activites flags: defined values .....	97
13.6. mcstatus: accuracy of home country field: defined values .....	97
13.7. mrace: race of the model: defined values .....	98
13.8. mbuild: body type of the model: defined values .....	98
13.9. vital statistics: meanings .....	98
14.1. dstatus: download status: defined values .....	99
14.2. dtype: download set type: defined values .....	100
14.3. dsetflag: Suggested value for scatflag based on parsing result .....	100
15.1. pgender: gender of the photographer: defined values .....	102
15.2. pregon: geographical location of the photographer: defined values .....	102
15.3. prating: overall rating of photographer: defined values .....	102
15.4. phardness: rating of how explicit this photographer can be: defined values .....	102
15.5. photographer activites covered flags: defined values .....	103
15.6. photographer technologies used flags: defined values .....	103
16.1. tstatus: tag entry status: defined values .....	104
16.2. tflag: tag content type status: defined values .....	104
17.1. vcurrent: vendor existance status: defined values .....	106
17.2. vshow: vendor index inclusion status: defined values .....	106
17.3. vmdiruse et al: vendor URL auto-usuability status: defined values .....	106
18.1. cflag: connection type: defined values .....	107
18.2. cstatus: connection entry status: defined values .....	107
19.1. kflag: active entry status: defined values .....	108

---

## List of Examples

2.1. WACS Module Import .....	3
2.2. Config and Security .....	4
2.3. Database Connection Initialisation .....	5
2.4. Database Query .....	6
2.5. Outputting The List .....	8
2.6. Php: Complete Simple Program .....	10
2.7. Perl: Complete Simple Program .....	11
3.1. Modified Output Loop with Icon Code .....	13
3.2. Modified SQL command for more Model Info .....	14
3.3. New version of the loop using tables .....	16
3.4. Adding Model Information .....	18
3.5. Adding A Rating Icon .....	20
3.6. Calling <code>dberror</code> for better error reporting .....	22
4.1. The Basic <code>SetDisp</code> Program .....	25
4.2. Adding A Set Icon .....	27
4.3. Making Camel-Style Text Readable .....	29
4.4. Modified Icon Cell .....	31
4.5. <code>getmodel</code> Subroutine .....	32
4.6. Calling The <code>getmodel</code> Function .....	33
5.1. WacsUI initialisation .....	35
5.2. Using WacsUI: <code>describeher</code> .....	35
5.3. Using <code>AddKeyIcons</code> .....	36

---

# Part I. WACS API Programming Tutorial

This part of the WACS Programming Guide is designed to introduce you to programming using the WACS API - examples will be given in both Perl and PHP5 dialects so you can choose to work in either language.

Chapter 1, *Introduction*

Chapter 2, *Basics: Getting Started*

Chapter 3, *Using More Database Fields*

Chapter 4, *Set Display Routines*

Chapter 5, *The User Interface Toolkit*

---

# Chapter 1. Introduction

## Overview

Welcome to WACS, Web-based Adult Content Server, a free software package for the management of material of an "Adult Nature" (or basically whatever euphemism for porn you prefer). It is web-based and can be used for the management of an existing collection, as a download manager, or as a back-end system for running a commercial adult web site. It is dramatically different from most other image gallery systems in that it understands photo sets and video clips as basic concepts, instead of single photographs. It also includes far more specialised tagging, source, relationship and attribute marking concepts than other more generalised systems. WACS' abilities in the areas of searching and dynamic filtering are really industry-leading in their power and flexibility.

## About This Book

This electronic book, the WACS Programming Guide, is designed to act both as an introduction to programming with the WACS API in either perl or PHP, and as a reference volume for both the API itself and the database schema. This book assumes you already have a basic knowledge of programming in your chosen language (PHP5 or perl5) and have some understanding of databases and in particular SQL (Structure Query Language). Some familiarity with WACS at a user level would also be a distinct advantage, and I'd strongly recommend working through the companion user guide first - who knows it might give you some ideas about neat extra features you can add to your own site. All documentation for WACS is available both within the distribution and from the WACS Web Site at Sourceforge.net [<http://wacsip.sourceforge.net/>].

It is important to stress that *ALL* of the collection management tools are implemented in Perl and the PHP interface is an optional addition to, not an alternative to, the core Wacs system which is perl based. Given the relative youth of the WACS system, php5 has been selected for the implementation to save future porting efforts as it is expected that php5 or later will be the minimum common standard by the time Wacs reaches 1.0. There is no intention to support older dialects of php at this point.

As the WACS software package is Open Source, we're always looking for contributions; if you create a site design (or prototype for one) which you don't end up using, maybe you would consider donating it to the repository of sample *WACS Skins*. We can always substitute our own artwork into already written web application code.

## About The Examples

For copyright/licensing reasons, the example images feature sets from photoshoots by the main developer of WACS (Beaky) and a friend of his. These sets will be available on our demonstration site when that goes live. Please understand that due to the bandwidth and storage costs in running such a server on the internet, and the need to verify (as best we can) that the applicant is an adult, there is a small charge for access to the site.



---

# Chapter 2. Basics: Getting Started

## Outline

In this chapter we're going to talk about the basic first steps in making use of the WACS API from your own programs. We're going to assume that you've got a WACS server you can use up and running; that you know where things are on it and that you have appropriate write access to the web document tree (if you're working in PHP) or the cgi-bin directory (if you're working in Perl). Hopefully you'll have both some models and a few image sets known in the WACS system to work with. For these first code examples, you could merely load the sample model profiles we've provided in the `samples` directory of the WACS distribution.

While the finished code of the sample programs featured here is available in the `samples` directory of the WACS Core distribution (for the Perl version) or the WACS-php distribution (for the PHP5 version), you may wish to type it in as you go along as an aid to learning how to use the interface. If you do, we'd recommend calling this file `mysimple` for perl, or `mysimple.php` for PHP. For consistency, we're going to put the PHP dialect first and then the Perl dialect in each of the examples.

The basic structure of your first WACS application will consist of five steps; these are:

1. import the WACS API modules
2. read configuration and check access rights
3. initialise the database connection
4. run an appropriate database query
5. retrieve records and display them

## A First WACS Program

### Modules: Importing

The very first step is to import the WACS API modules into your program file along with those standard modules needed to access the database. These files should be in the right location already and should just be found without any additional specification of where they are.

#### Example 2.1. WACS Module Import

```
require_once "wacs.php";
require_once "DB.php";

$wacs = new Wacs;
```

The same code segment implemented in perl looks like:

```
use Wacs;  
use DBI;
```



### Note

The PHP interface requires an *Object Handle* to use when accessing the WACS module which we're simply calling `$wacs`. Perl doesn't need such a construct - there is simply the one instance.

## Configuration And Security

The second step is to read the standard WACS configuration file to find out where everything is, and then check that this user is allowed to access the WACS system. This is a two step process, and the reading of the configuration file must be done first; otherwise WACS doesn't know where to look for the security files it needs to determine whether this user should be given access or not.

### Example 2.2. Config and Security

```
// read the Wacs configuration files  
$wacs->read_conf();  
  
// check the auth(entication and authorisation) of this user  
$wacs->check_auth( $_SERVER['REMOTE_ADDR'], 1 );
```

and here is the same thing again in the perl dialect:

```
# read the Wacs configuration files  
read_conf;  
  
# check the auth(entication and authorisation) of this user  
check_auth( $ENV{"REMOTE_ADDR"}, 1 );
```

## Initialising Database Connection

The third step is to initialise the database connection. Since some databases require an environment variable to determine where their configuration files have been stored, this needs to be set first. Wacs provides for this and this code will create that environment variable, if needed, and then proceed to establish the database connection itself.

### Example 2.3. Database Connection Initialisation

```
// database initialisation
// - establish environment variable
$dbienv = $wacs->conf_get_attr("database","dbienvvar");
if( ! empty( $dbienv ) )
{
    putenv($dbienv."=".$wacs->conf_get_attr("database","dbienvvalue"));
}
// - connect to the database
$dbhandle= DB::connect( $wacs->conf_get_attr("database","phpdbconnect") );
if( DB::iserror($dbhandle))
{
    die("Can't connect to database\nReason:".$dbhandle->getMessage."\n");
}
$dbhandle->setFetchMode(DB_FETCHMODE_ORDERED);
```

and here's how we do it in perl:

```
# database initialisation
# - establish environment variable
$dbienv = conf_get_attr( "database","dbienvvar" );
if( $dbienv ne "" )
{
    $ENV{$dbienv}= conf_get_attr( "database","dbienvvalue" );
}
# - connect to the database
$dbhandle=DBI->connect( conf_get_attr("database","dbiconnect"),
                      conf_get_attr("database","dbuser"),
                      conf_get_attr("database","dbpass") ) ||
die("Can't connect to database\nReason given was $DBI::errstr\n");
```

OK, let's just study this code for a moment. It first calls the WACS API function **conf\_get\_attr** with the section parameter of *database* as it wants database related configuration information, and an argument of *dbienvvar*. The WACS API function **conf\_get\_attr** is short for *configuration get attribute* and returns the value of the configuration file parameter of that name or it's default value. The *dbienvvar* means *database interface environment variable*. A typical value for this might be something like `ORACLE_HOME` which is the environment variable that Oracle 10g and 11i requires to be set in order to find it's current configuration.

The next line of the code checks to see if we got back an actual variable name (eg `ORACLE_HOME`) or an empty string (ie nothing). If we were given a valid variable name, then we're going to need to set it the value it should be, which again we can get from the configuration file, this time called *dbienvvalue* which is short for *database interface environment value* (as distinct from the *variable* name we just looked up). A likely value for this might be `/usr/local/oracle`. Obviously if we're given no variable name to set, there's no point looking for a value for it! Conversely we are assuming that having bothered to name the variable in the configuration file, also put in a valid value for it - this code could break if the variable name is specified but not it's value.

The second section of these code segments is to do with the establishment of a connection to the database and is a little different between the two versions. Both systems use a handle for the database connection,

which we call `$dbhhandle` - imaginative name huh? In both cases, the respective database APIs provide a **connect** function which takes an argument of how to connect to the database. The Php version takes a single argument, which is stored in our configuration files as `phpdbconnect` and includes the whole username, password and database specification in a single lump. The Perl version asks for three: the database specification, the username and finally the password. The configuration file knows these as `dbconnect`, `dbuser` and `dbpass` respectively.

The final bit copes with putting out some kind of error message, at least showing the point of failure, if we are unable to establish a connection to the database. The methods are very slightly different, but the effect is very much the same between the two versions. We then just tell the PHP DB interface how we wish it to organise the returned data; the perl DBI default is pre-determined and is what we want.



### Tip

Note that you might wish to have completed the output of the HTML header section and started the body by this point so that should the database connection fail, the error message will be visible.

## Fetching Some Records

The next step in the process is to use the database connection we've established to actually make a request of the database. For now don't worry about what that request is or how we've written it - we'll come back to that topic in detail later in this chapter. Look at the mechanics of how we're issuing the request and getting back the results. What we're going to ask the database for is a list of those girls who are marked as *Favourite Solo* models. We chose this because both the models in our current samples directory are marked as this and so even if you only have our sample records loaded, you should find some matches.

### Example 2.4. Database Query

```
// do db select
//           0         1         2         3
$query = "select mname, modelno, mbigimage, mimage from ".
        $wacs->conf_get_attr("tables", "models").
        " where mflag = 'S' order by mname";
$cursor = $dbhhandle->query( $query );
```

The method is a little different in perl in that it is separated into two steps; as a result it looks like this...

```
# do db select
#           0         1         2         3
$query = "select mname, modelno, mbigimage, mimage from ".
        conf_get_attr("tables", "models").
        " where mflag = 'S' order by mname";
$cursor = $dbhhandle->prepare( $query );
$cursor->execute;
```



### Note

The query structure is very similar between Php and perl apart for the two step process of validating and then separately executing the query in perl. This is mostly down to different

traditions that exist for database accesses in each language. The net result is similar in technical terms and identical in output terms

In both cases we're putting together an SQL query that reads:

```
select mname, modelno, mbigimage, mimage
from models
where mflag = 'S'
order by mname
```

This query asks the database to fetch the four named items: `mname`, `modelno`, `mbigimage`, and `mimage` from the database table called `models` where the field `mflag` has a value of the capital letter `S` and to sort the results it returns to us by the value in the field called `mname`. It may not surprise you to learn that `mname` is the model's name, `modelno` is our reference number for her, `mbigimage` is the (location of the) large size headshot of her and `mimage` is the (location of the) smaller size headshot of her.

You may have noticed that the only part of this that wasn't copied verbatim from the code is the `from models` bit and that there we've used the WACS API call `conf_get_attr` to get the actual name of the database table concerned from the main WACS configuration file. This is actually important and it's strongly recommended that you do use this form when creating SQL queries. If you really insist on knowing why, take a look at the section on the tables part of the `wacs.cfg` configuration file in the WACS configuration guide.

Once we've created the SQL query, we feed it to the database routines. The first step is to pass in the SQL query and have the database perform that search on the database. Once the query has been executed, we want to pull back the matching records (or rows in database parlance) for each model. In both Php and Perl we're calling a routine that returns to us a single row from the database (a single model's record in this case) each time it's called. When we run out of records, a null return is given and our while loop ends. In Php, the function to do this is called using `fetchRow` which returns the next row as an array of values, which we assign into the variable `$results` each time. In Perl, the function we're using is called `fetchrow_array` because perl offers us a choice in the type of data we are returned and in this case we want a numerically indexed array.



### Note

There are other approaches to getting back the data, including having it returned in one big lump (such as with the Php call `getAll()`) - this has been avoided as some WACS installations might have tens of thousands of matching records for some queries.

## Showing The Results

The final step is to actually generate some output from the data we've fetched from the database. We're going to do this as an unordered list in HTML, so we're going to be adding a little formatting to the output as we retrieve each record.

## Example 2.5. Outputting The List

```
print "<ul>\n";
while( $results = $cursor->fetchRow() )
{
    print "<li>";
    print "<a href=\"". $wacs->conf_get_attr("server", "cgiurl");
    print "wacsmphumbs/" . $results[1]. "\">";
    print $results[0]. "</a></li>\n";
}
print "</ul>\n";
```

and here's the perl version...

```
print "<ul>\n";
while( @results = $cursor->fetchrow_array )
{
    print "<li>";
    print "<a href=\"". conf_get_attr("server", "cgiurl");
    print "wacsmphumbs/" . $results[1]. "\">";
    print $results[0]. "</a></li>\n";
}
print "<ul>\n";
```

We start off by printing out the HTML instruction to start an unordered list (<ul>) in a line on its own. We then start a while loop which goes through each entry until it's done them all. Both versions use the database cursor object (`$cursor`) to fetch the next record (aka row) from the database using the **fetchRow** or **fetchrow\_array** method and assigning it into the array `$results` (or in perl `@results`). The act of the assignment fails when there are no more records to fetch and the while loop will terminate. The construct here is based upon the fact that both languages have separate operators for assignment (=) and comparison (== and eq) and so the code is unambiguous (at least to the php and perl interpreters it is!).

Once inside the body of the while loop we print out the start of list entry tag (<li>) and start in on making use of the data. In the quest to make this example a little bit more satisfying, we've tried to make sure this application does something vaguely useful. A simple list of names is all well and good, but we wanted it to actually *do* something! So what we've done here is to create a link around each models name that points to her model page as displayed by the standard WACS tools. The raw HTML to achieve this would look like:

```
<a href="http://www.mywacsserver.com/cgi-bin/wacsmphumbs/123">
Sarah</a>
```

So we're left with a slight problem here in that we don't know in advance (trust me on this) what the WACS server is called, we don't know what the models are called and we don't know what their numbers are. We have no idea if we have a model number 123 or not and whether she's called Sarah; but the WACS system should be able to fill in all the blanks for us.

The first part of the code merely prints out the start of the HTML `<a href=">` and then we ask the WACS configuration system what it's externally visible URL for cgi-bin programs is. We do this using the **conf\_get\_attr** call again, telling it we want an answer in the section `server` of the URL for cgi scripts aka `cgiurl`. On the next line of the example we put the name of the WACS application we want to link to, in this case **wacsmphumbs**. Since the way we tell **wacsmphumbs** what we want it to look up is to add a slash and then the model number to the URL, we add a slash (/) on the end and then the number.



## Tip

You may have noticed that we added a comment on the line above the SQL select statement with 0,1,2,3 with each number above the field name in the query. This was a shorthand to ourselves to remind us what the index number in the array is for each of those database fields.

Since the order of the fields we asked for was `mname`, `modelno`, `mbigimage` and then `mimage`, the results in the array will be the same - element 0 will be the `mname`, element 1 will be the model number, and so on. In both cases we're dealing with a single-dimensional array. The first field we want to go into the URL for **wacsmodelthumbs** is the model number, so that will be element 1 (not zero) therefore we write `$results[1]`. We then finish off the URL reference by closing the quotes (") and the `>` tag.

We then want to print the model's name which will be element 0 in our arrays, put out the closing anchor tag (`</a>`) and then finish off the unordered line entry with the end line tag (`</li>`). We then print out a new line so the generated page is easier to read. The moving on to the next record will be done as a by-product of the test for the next iteration around the while loop. Once we exit the loop, we finish off the HTML unordered list.

## Finishing Off

To just finally finish it off, we need to add a few more pieces just to make it work. For the PHP version, we need to declare it as being a PHP program with `<?php` at the very start of the file, with a matching `>` at the very end. For Perl, we need to declare it as a Perl script with the very first line being just `#!/usr/bin/perl`. Additionally for Perl, we need to output the mime content type declaration so that the web browser knows what kind of object it's being passed - this is done simply with:

```
print "Content-Type: text/html\n";
print "\n";
```

Next we need a couple of lines of HTML preamble near the beginning (as mentioned before, just before the database connection code so we could see any error message that appears):

```
<html>
<head>
<title>MySimple: Index Of Favourites</title>
</head>
<body>
```

Similarly at the end, we just need to finish the page off with the HTML tail piece:

```
</body>
</html>
```

## Putting It All Together

With all the components in place, let's review the new MySimple WACS program in its entirety. We include the modules, initialise the configuration system, check the authorisation, connect to the database, draft the query, submit it and then loop through the results. Not really that complex now we know what each part does. Anyway here's the finished code....

**Example 2.6. Php: Complete Simple Program**

```
<?php
// MySimple - sample WACS API program (PHP5)
require_once "wacs.php";
require_once "DB.php";
$wacs = new Wacs;
// read the Wacs configuration files
$wacs->read_conf();
// check the authentication and authorisation) of this user
$wacs->check_auth( $_SERVER['REMOTE_ADDR'], 1 );
// start the HTML document
print "<html>\n";
print "<head>\n";
print "<title>MySimple: Index Of Favourites</title>\n";
print "</head>\n";
print "<body>\n";
// database initialisation
// - establish environment variable
$dbienv = $wacs->conf_get_attr("database","dbienvvar");
if( ! empty( $dbienv ) )
{
    putenv($dbienv."=".$wacs->conf_get_attr("database","dbienvvalue"));
}
// - connect to the database
$dbhandle= DB::connect( $wacs->conf_get_attr("database","phpdbconnect") );
if( DB::iserror($dbhandle))
{
    die("Can't connect to database\nReason:".$dbhandle->getMessage()."\n");
}
$dbhandle->setFetchMode(DB_FETCHMODE_ORDERED);
// do db select
//          0          1          2          3
$query = "select mname, modelno, mbigimage, mimage from ".
        $wacs->conf_get_attr("tables","models").
        " where mflag = 'S' order by mname";
$cursor = $dbhandle->query( $query );
// output the results
print "<ul>\n";
while( $results = $cursor->fetchRow() )
{
    print "<li>";
    print "<a href=\"".$wacs->conf_get_attr("server","cgiurl");
    print "wacsmphthumbs/".$results[1].\">";
    print $results[0]."</a></li>\n";
}
print "</ul>\n";
// finish off
print "</body>\n";
print "</html>\n";
?>
```



## Example 2.7. Perl: Complete Simple Program

```
#!/usr/bin/perl
#
# MySimple - Sample WACS Program (Perl)
#
use Wacs;
use DBI;
# read the Wacs configuration files
read_conf;
# check the auth(entication and authorisation) of this user
check_auth( $ENV{"REMOTE_ADDR"}, 1 );
# output the HTML headers
print "Content-Type: text/html\n";
print "\n";
print "<html>\n";
print "<head>\n";
print "<title>MySimple: Index Of Favourites</title>\n";
print "</head>\n";
print "<body>\n";
# database initialisation
# - establish environment variable
$dbienv = conf_get_attr( "database","dbienvvar" );
if( $dbienv ne "" )
{
    $ENV{$dbienv}= conf_get_attr( "database","dbienvvalue" );
}
# - connect to the database
$dbhandle=DBI->connect( conf_get_attr("database","dbiconnect"),
                        conf_get_attr("database","dbuser"),
                        conf_get_attr("database","dbpass") ) ||
die("Can't connect to database\nReason given was $DBI::errstr\n");
# do db select
#           0           1           2           3
$query = "select mname, modelno, mbigimage, mimage from ".
        conf_get_attr("tables","models").
        " where mflag = 'S' order by mname";
$cursor = $dbhandle->prepare( $query );
$cursor->execute;
print "<ul>\n";
while( @results = $cursor->fetchrow_array )
{
    print "<li>";
    print "<a href=\"".conf_get_attr("server","cgiurl").";
    print "wacsmphumbs/". $results[1]. "\">";
    print $results[0]. "</a></li>\n";
}
print "<ul>\n";
# finish off
print "</body>\n";
print "</html>\n";
```

## Running MySimple

Our first WACS application is now complete, so copy the file into either the web server document tree (for Php) or the web server cgi-bin directory (for perl). When you call up the URL, you should see something like this....



Granted it's fairly plain, but the names are in alphabetical order and there are links on each name to that girl's model page. If you didn't see any output, or got an error, you need to check the error log for the server you're using. With Apache on linux, the usual location of this is `/var/log/httpd/www.mywacserver.com-errorlog` or something similar to that.

## Reviewing The First Program

This has been a fairly long and intense chapter, but we obviously had a lot of ground to cover and we really wanted to achieve a usable program before the end of it. This hopefully we've done. We've seen how to include the WACS module and the Database interface module. We've seen how to use **read\_conf** and **check\_auth** to read the configuration files and check the user's credentials. We've then made multiple uses of **conf\_get\_attr** to get all of the information together we need to make a connection to the database.

After all that setup procedure, which will become a very familiar template as you program with the WACS API, we looked at creating and sending a query to the database, retrieving the results and formatting those results as a simple web page. In the next chapter, we'll look at how to make use of other information stored within the database.

---

# Chapter 3. Using More Database Fields

## Adding Model Icons

In the simple example in the last chapter, we saw how to create a list of model's names with hypertext links on each name to that model's standard WACS model page. Obviously that's not a particularly presentable page by itself, so the next step is to add a head shot for each model to the links.

We actually already paved the way for doing this by including the two headshot image fields in the results we asked for from the SQL query - if you remember, we put:

```
select mname, modelno, mbigimage, mimage
```

Since we have the data already, all we need to do now is to add a few extra statements to the output section to output an appropriate image tag and we'll have included the model's headshot too. We have a configuration attribute in the **server** section of the configuration file called **wacsurl** that tells us where the WACS area can be found on the WACS server. Standard size model headshots are conventionally found in the **icons/** directory directly below the top level. So all we need to do is add in a call to **conf\_get\_attr** to get it and build the appropriate HTML **img** tag. In PHP we'd write:

```
print "<img src=\"".$wacs->conf_get_attr("server","wacsurl");  
print "icons/".$results[3]."\\" alt=\"[".$results[0]."]\">";
```

and in perl we'd write:

```
print "<img src=\"".$conf_get_attr("server","wacsurl");  
print "icons/".$results[3]."\\" alt=\"[".$results[0]."]\">";
```

this needs to be done just below the line that establishes the link to the model's WACS model page, but before her name (you could put it after if you prefer) and closing **</a>**.

### Example 3.1. Modified Output Loop with Icon Code

```
while( $results = $cursor->fetchRow() )  
{  
    print "<li>";  
    print "<a href=\"".$wacs->conf_get_attr("server","cgiurl");  
    print "wacsmphumbs/".$results[1]."\\">";  
    print "<img src=\"".$wacs->conf_get_attr("server","wacsurl");  
    print "icons/".$results[3]."\\" alt=\"[".$results[0]."]\">";  
    print $results[0]."</a></li>\n";  
}
```

and in perl this now looks like:

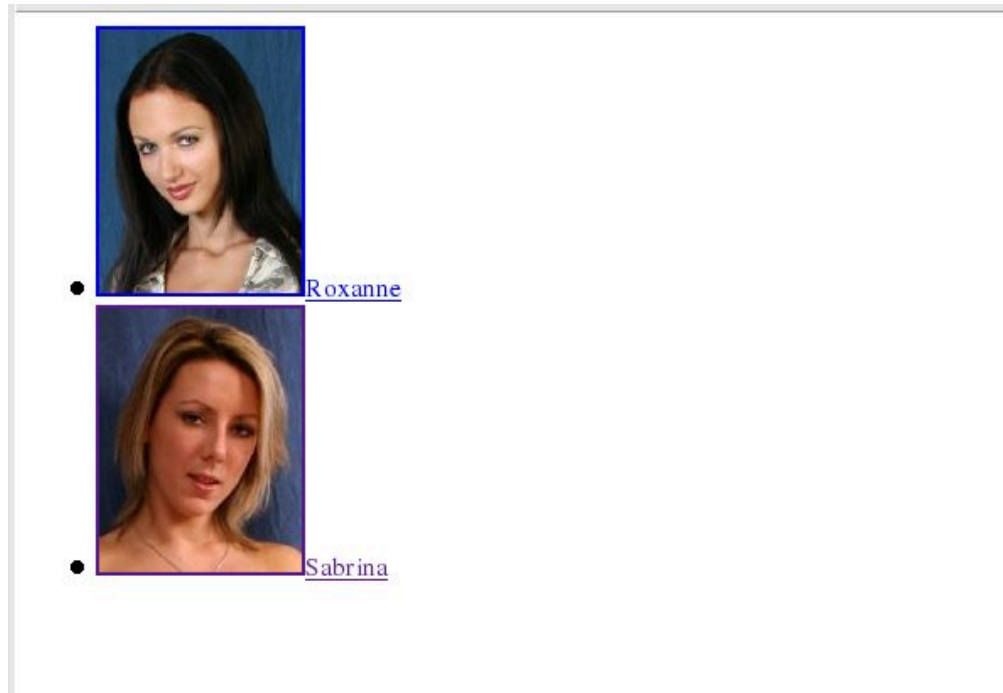
```
while( @results = $cursor->fetchrow_array )  
{
```

```

print "<li>";
print "<a href=\"".conf_get_attr("server","cgiurl");
print "wacsmphumbs/". $results[1]. "\">";
print "<img src=\"".conf_get_attr("server","wacsurl");
print "icons/". $results[3]. "\" alt=\"[". $results[0]. "]"\">";
print $results[0]. "</a></li>\n";
}

```

We then copy up the modified version of the program and run it and we should see something like this:



## More Model Information

The WACS database does of course carry far more information about the model than just her name and icons, so for the next step we're going to look at adding a few basic pieces of information about her to each entry. The first step is to add some additional fields to the list of what we want returned by the SQL query. Initially we're going to add another five fields: they are `mhair`, `mlength`, `mtitsize`, `mnsets` and `mnvideos`. These database fields give us her hair colour, length, the size of her breasts and the number of images sets and videos we have by her respectively. The modified version of the query looks like:

### Example 3.2. Modified SQL command for more Model Info

```

// do db select
//           0      1      2      3      4
$query = "select mname, modelno, mbigimage, mimage, mhair, "
//           5      6      7      8
//           "      mlength, mtitsize, mnsets, mnvideos from ".
//           $wacs->conf_get_attr("tables","models").
//           " where mflag = 'S' order by mname");
$cursor = $dbhandle->query( $query );

```

in php.



## Note

We've added a second line of comments with the element numbers within the array that the returned database field will appear in; mlength will be index 5 for instance.

The same code in perl will look like:

```
# do db select
#           0         1         2         3         4
$query = "select mname, modelno, mbigimage, mimage, mhair, ".
#           5         6         7         8
#           "           mlength, mtitsize, mnsets, mnvideos from ".
#           "           conf_get_attr("tables","models").
#           " where mflag = 'S' order by mname";
$cursor = $dbh->prepare( $query );
$cursor->execute;
```

## Using HTML tables

The next step is to modify the display loop to include the extra details and in this case it probably makes sense to switch to using an HTML table cell to contain and manage the entry. We'll start off by simply re-writing the existing display loop to build the results into an HTML table instead - once we have that working, we'll restyle the table to include the extra fields we just added to the query. There is no actual requirement to make use of all the fields we've requested.

Lets have a look at the structure of the HTML document we're outputting here: First we need to open the new table, then each model will have her own row as we go through with the headshot image on the left and her name on the right, and finally we'll finish off the table. The HTML (minus the links) to do this will look something like:

```
<table>
  <tr>
    <td></td>
    <th>Roxanne</th>
  </tr>
  <tr>
    <td></td>
    <th>Sabrina</th>
  </tr>
</table>
```

Of course the next step is to re-write the code to actually recreate the necessary HTML; the start and end of the table simply replace the unordered list (<ul> and </ul> ) tags outside the loop that iterates through the list of models returned by the database. The list element (<li> and </li>) tags get replaced by the row start and end tags (<tr> and </tr>). Since we're putting the headshot icon and the name in separate elements and want a link to the appropriate model page on both of them, we need to double up the code that creates the hypertext link to wacsmphumbs. We then include the icon (with alignment attributes) in a standard table tag ( <td> and the name in a heading (<th>) table tag so it comes out in bold and is centred.

The mysimple example thus re-written will look like:

**Example 3.3. New version of the loop using tables**

```
// output the results
print "<table>\n";
while( $results = $cursor->fetchRow() )
{
    // start the HTML table row
    print "<tr><td valign=top align=center>\n";
    // link around the headshot image
    print "<a href=\"". $wacs->conf_get_attr("server","cgiurl")";
    print "wacsmphumbs/". $results[1]. "\">";
    // head shot image
    print "<img src=\"". $wacs->conf_get_attr("server","wacsurl")";
    print "icons/". $results[3]. "\"[".$results[0]."]\"></a>\n";
    // end this cell and start the next
    print "</td><th>\n";
    // link around name
    print "<a href=\"". $wacs->conf_get_attr("server","cgiurl")";
    print "wacsmphumbs/". $results[1]. "\">";
    // the name
    print $results[0]. "</a>\n";
    // end the HTML table row
    print "</th></tr>\n";
}
print "</table>\n";
// finish off
```

and re-writing the same function in perl gives us something like:

```
# output the results
print "<table>\n";
while( @results = $cursor->fetchrow_array )
{
    # start the HTML table row
    print "<tr><td valign=top align=center>\n";
    # link around the headshot image
    print "<a href=\"". conf_get_attr("server","cgiurl")";
    print "wacsmphumbs/". $results[1]. "\">";
    # head shot image
    print "<img src=\"". conf_get_attr("server","wacsurl")";
    print "icons/". $results[3]. "\"[".$results[0]."]\"></a>\n";
    # end this cell and start the next
    print "</td><th>\n";
    # link around name
    print "<a href=\"". conf_get_attr("server","cgiurl")";
    print "wacsmphumbs/". $results[1]. "\">";
    # the name
    print $results[0]. "</a>\n";
    # end the HTML table row
    print "</th></tr>\n";
}
print "</table>\n";
```

```
# finish off
```

When run, this modified version of the script should produce the following:



As you can see, this has improved the layout somewhat over the previous version using just unordered list elements. Now to add those extra fields....

## Adding The Model Details

To display some more details about the model, we're going to span the headshot on the left hand side over several rows, and add the model details themselves as additional table rows on the right hand side. Our first change therefore is to add `rowspan=4` to the options on the image container `<td>` tag. The resulting php code is:

```
// start the HTML table row
print "<tr><td rowspan=4 valign=top align=center>\n";
// link around the headshot image
```

and in perl reads:

```
# start the HTML table row
print "<tr><td rowspan=4 valign=top align=center>\n";
# link around the headshot image
```

Next we add the second row which will include her hair colour and length, then a third row which will describe her breast size and the fourth row that gives the number of image sets and the number of videos we have for her.

### Example 3.4. Adding Model Information



```
// end the HTML table row
print "</th></tr>\n";
// do the second row (her hair)
print "<tr><td>hair: ";
print $results[5]." ".$results[4];
print "</td></tr>\n";
// do the third row (her breasts)
print "<tr><td>breasts: ";
print $results[6]."\n";
print "</td></tr>\n";
// do the fourth row (her sets)
print "<tr><td>sets: ";
print $results[7];
if( $results[8] > 0 )
{
    print " videos: ".$results[8];
}
print "</td></tr>\n";
```

and the same implemented in perl would look like:

```
# end the HTML table row
print "</th></tr>\n";
# do the second row (her hair)
print "<tr><td>hair: ";
print $results[5]." ".$results[4];
print "</td></tr>\n";
# do the third row (her breasts)
print "<tr><td>breasts: ";
print $results[6]."\n";
print "</td></tr>\n";
# do the fourth row (her sets)
print "<tr><td>sets: ";
print $results[7];
if( $results[8] > 0 )
{
    print " videos: ".$results[8];
}
print "</td></tr>\n";
}
```

With these changes made, if you now run this version of the program, which is called **mysimple4** in the `samples/programming` directory, you should see something like this:



	<p><u><a href="#">Roxanne</a></u></p> <p>hair: Long Dark Hair</p> <p>breasts: Small</p> <p>sets: 8 videos: 1</p>
	<p><u><a href="#">Sabrina</a></u></p> <p>hair: Shoulder B londe</p> <p>breasts: Small</p> <p>sets: 5</p>

There's obviously a lot more room for using many more of the fields within the model schema for further improvement of our model index, and we'll return to this subject in a later chapter (Chapter 5, *The User Interface Toolkit*). Before we leave the topic of models and move on to sets, we will cover just one more topic, that of adding rating icons.

## Adding Other Icons

One of the significant features of WACS is its ability to include various attribute icons within pages to make specific aspects and attributes easier to recognise. While many of them need some additional logic to handle their display, a few of them like the model's rating and country of origin are actually fairly simple to use. We're going to take a quick look at how we'd use the WACS API to include the rating icons before moving on to look at how we handle sets. We will return to the more complex cases later when we look at the User Interface toolkit API.

For the model's rating, we need the field called `mrating` so the first step is to add this to the list of fields that we select from the database:

```
// do db select
//           0           1           2           3           4
$query = "select mname, modelno, mbigimage, mimage, mhair, ".
//           5           6           7           8           9
//           "          mlength, mtitsize, mnsets, mnvideos, mrating ".
//           "from ".$wacs->conf_get_attr("tables","models").
//           " where mflag = 'S' order by mname";
$cursor = $dbh->query( $query );
```

and in perl the change makes this section read:

```
# do db select
#           0           1           2           3           4
```

```

$query = "select mname, modelno, mbigimage, mimage, mhair, ".
#           5           6           7           8           9
#           "           mlength, mtitsize, mnsets, mnvideos, mrating ".
#           "from ".conf_get_attr("tables","models").
#           " where mflag = 'S' order by mname";
$cursor = $dbh->prepare( $query );
$cursor->execute;

```

With the rating field now in the data returned to us by the database, we can move down and update the display section to make use of it. The first step needed is to change the `rowspan` setting from 4 to 5 to accomodate the extra line of output.

```

// start the HTML table row
print "<tr><td rowspan=5 valign=top align=center>\n";
// link around the headshot image

```

and in perl...

```

# start the HTML table row
print "<tr><td rowspan=5 valign=top align=center>\n";
# link around the headshot image

```

The final step is to add the processing of the `mrating` field. All WACS icons are typically stored in the `glyphs/` directory which is within the web server document tree. To find its exact URL, you use the `conf_get_attr` function to retrieve the value `iconurl` in the section server. Within this directory, you will find five files called `rating-1.png` through `rating-5.png` which look like this:



To make use of this we need to first test our data to see if we have a valid ratings value at all, then merely concatenate a string to create the necessary icon reference. In php, this will look like this:

### Example 3.5. Adding A Rating Icon

```

print "</td></tr>\n";
// add the rating icon (if we have a value)
print "<tr><td align=center valign=top>";
if( $results[9] > 0 )
{
    print "<img src=\"";
    print $wacs->conf_get_attr("server","iconurl");
    print "rating-".$results[9].".png\">";
    print " alt=\"[".$results[9]." out of 5]\">";
}
else
{
    print "no rating";
}
print "</td></tr>\n";

```

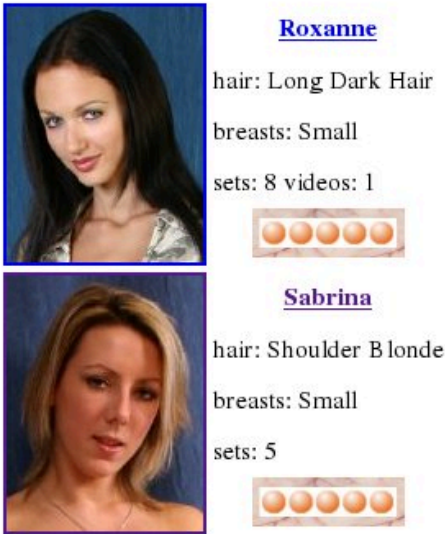
while the same example in perl, would look like this:

```

print "</td></tr>\n";
# add the rating icon (if we have a value)
print "<tr><td align=center valign=top>";
if( $results[9] > 0 )
{
    print "<img src=\"".conf_get_attr("server","iconurl");
    print "rating-".$results[9].".png\"";
    print " alt=\"[".$results[9]." out of 5]\>";
}
else
{
    print "no rating";
}
print "</td></tr>\n";
}

```

Once you've put in these three changes, you can run the resulting script and expect to get an output something like this:



The screenshot displays two model profiles in a list. Each profile consists of a small portrait photo on the left and text on the right. The first profile is for 'Roxanne', with long dark hair, small breasts, 8 sets, 1 video, and a rating of 4 out of 5 (represented by 4 orange circles). The second profile is for 'Sabrina', with shoulder-length blonde hair, small breasts, 5 sets, and a rating of 4 out of 5 (represented by 4 orange circles).

At this point we're hopefully beginning to get a rather more satisfying display of model details. Obviously there are many other tweaks we might like to add, and we'll return to some of those later on when we look at the User Interface Toolkit and the routines that provides. There is however one more thing we really should cover now - what happens when something goes wrong.

## Improving Error Reporting

One of the most important things in good website engineering is ensuring that when things fail, it's handled gracefully with some kind of reasonable error message returned to the user, and that the event is logged properly in the system error logs. There are basically four ways in which a WACS application is likely to fail - authentication, failure to parse the configuration files, and failure to connect to the database, and failure to find the content.

The authentication failure is pretty conclusively covered by the core WACS `check_auth` function and its partners. The parser is rather more tricky to cope with, and the XML parse routines tend to just abort - it's also very all or nothing; the file parses or it doesn't. Additionally once a configuration file is in place, it's unlikely to become corrupted; if it's merely disappeared the defaults will be used and the system will most likely have problems at the next stage of connecting to the database. The third is connecting to the database, which we'll deal with in a moment. The fourth, failure to find content, doesn't result in completely blank screens and should get reported to you quite quickly. Additionally there are so many places it could be (raid partition, lvm volume, remote fileserver) that we can't really do much in a general way.

Where we can get some traction is with decent reporting of database connection problems, and this where the `dberror` function comes into play. Previously, if we failed to connect to the database we did the following in php:

```
if( DB::iserror($dbhandle))
{
    die("Can't connect to database\nReason:".
        $dbhandle->getMessage()."\n");
}
```

and the similar steps in perl were:

```
$dbhandle=DBI->connect( conf_get_attr("database","dbiconnect"),
                      conf_get_attr("database","dbuser"),
                      conf_get_attr("database","dbpass") ) ||
die("Can't connect to database\nReason given was $DBI::errstr\n");
```

To improve this, we're going to change this (called **mysimple6** in the example code) to use the `dberror` function instead. This is a routine that uses named parameters, a technique we'll see a lot more of later as we use the WacsUI programming library. Basically we pass it up to five arguments or parameters, but we tell it what each one is, thus the order doesn't matter and if any of them are missing, it doesn't affect the values of the others. The `dberror` routine expects parameters called: **header**, **message**, **error**, **dbuser** and **dbhost**.

The **header** is to tell the routine how early in the proceedings we are and whether we still need to start the HTML of the web page. Setting **header** to `y` says we do want a header added, setting it to `n` says we don't. The next one, **message** is the message that the end user will see. The next three are the error message returned by the database routines, the username it was trying to use, and the database connect string it was trying to use. Here is the code for doing this in PHP5:

### Example 3.6. Calling `dberror` for better error reporting

```
if( DB::iserror($dbhandle))
{
    $wacs->dberror( array(
        "header"=>"y" ,
        "message"=>"MySimple6: Can't connect to database",
        "error"=>$dbhandle->getMessage() ,
        "dbuser"=>$wacs->conf_get_attr("database","dbuser") ,
        "dbhost"=>$wacs->conf_get_attr("database","phpdbconnect")
    ));
}
```

while the same basic code in perl looks a little simpler because the parameter names don't need to be *packaged up* into an array before they're passed:

```
$dbhhandle=DBI->connect( conf_get_attr("database","dbiconnect"),
                        conf_get_attr("database","dbuser"),
                        conf_get_attr("database","dbpass") ) ||
dberror( header=>'n',
         message=>"Can't connect to database",
         error=>$DBI::errstr,
         dbuser=>conf_get_attr("database","dbuser"),
         dbhost=>conf_get_attr("database","dbiconnect") );
```

With the error reporting improved, we'll move on to other things. We'll continue to use the short form version of the error message for brevity in the later examples, but you'll know that you probably want to actually use `dberror` in most cases. Next up, we'll take a look at displaying set details rather than those of models....

---

# Chapter 4. Set Display Routines

## About Set Display

So far we've looked at displaying the information in the models table in the database, but of course there is also the small matter of sets without which whole thing wouldn't have much point. In this chapter we're going to look at displaying details of the sets, and then towards the end of the chapter, how to tie models and sets together.

In most of these examples, we're going to use the standard WACS tools to actually display the details of the sets themselves, but you can of course write your own web apps to do this should you wish to. In most cases we'll throttle the examples to only show a first few sets from the databases and assume you'll develop your own strategies for paginating and sub-dividing the sets in real world applications.

## Sets: The Basic Bones

Since we're starting a new application, we'll start from scratch with the basic bones which we'll call **setdisp**. Much of the basic structure of this program should be getting quite familiar by now. The same five basic steps are to be found here - bring in the modules, initialise them, set up the database connection, submit the query and loop through the results outputting them.

What we're setting out to do in this script is to display a list of the latest additions of image sets marked as being of category flag type T which means they're solo sets involving toy usage. This we achieve by requesting only sets of type I which means image sets and of category flag type T.



### Tip

The full lists of recommended values for the type and category flag can be found in the schema reference section at the back of this book in Chapter 10, *Schema Reference: Sets*.

The basic format is that we once again create an HTML table with a row for each record. There's a link on the name of the set that leads to the standard WACS page display program **wacsindex**. This takes a number of URL arguments but the one we're using here is to prefix the set number with `page` which puts it into paged display mode and appended with a `.html` so that it saves correctly and in some cases will get cached. We're shrinking the font in which it's displayed as it can be quite a long line of text in it's stored form (but more on that topic later).



### Note

The SQL query itself looks after the ordering of the output; the `order by sadded desc` retrieves the entries in the reverse order in which they were added - the database field `sadded` being the date the set was added to the database, and the `desc` (meaning descending) puts the biggest value first. In this case that is the most recent date...

**Example 4.1. The Basic SetDisp Program**

```

<?php
// setdisp - set display program
require_once "wacs.php";
require_once "DB.php";
$wacs = new Wacs;
$wacs->read_conf();
$wacs->check_auth( $_SERVER['REMOTE_ADDR'],1 );
// start the document
print "<html>\n";
print "<head>\n";
print "<title>SetDisp - List of Sets</title>\n";
print "</head>\n";
print "<body>\n";
// connect to the database
$dbienv = $wacs->conf_get_attr("database","dbienvvar");
if( ! empty( $dbienv ) )
{
    putenv($dbienv."=".$wacs->conf_get_attr("database","dbienvvalue"));
}
$dbhandle = DB::connect( $wacs->conf_get_attr("database","phpdbconnect"));
if( DB::iserror($dbhandle) )
{
    die("Can't connect to database\nReason:".$dbhandle->getMessage()."\n");
}
$dbhandle->setFetchMode(DB_FETCHMODE_ORDERED);
//          0          1          2          3          4          5
$query = "select setno, stitle, stype, scatflag, simages, scodec ".
        "from ".$wacs->conf_get_attr("tables","sets")." ".
        "where stype = 'I' and scatflag = 'T' ".
        "order by sadded desc ";
$cursor = $dbhandle->query( $query );
print "<table>\n";
$setcount=0;
while( (($results = $cursor->fetchRow()) &&
        ($setcount < 25 ) ) )
{
    // start the row
    print "<tr><td align=center>\n";
    // create the link
    print "<a href=\"".$wacs->conf_get_attr("server","cgiurl");
    print "wacsindex/page".$results[0].".html\">";
    // print out the set name
    print "<font size=-2 face=\"arial,helv,Helvetica,sans\">";
    print $results[1]."</font></a>\n";
    // end the row
    print "</td></tr>\n";
    $setcount++;
}
print "</table>\n";
print "</body>\n";
print "</html>\n";
?>

```

and implementing the same code in perl gives us:

```
#!/usr/bin/perl
# setdisp - set display program
use Wacs;
use DBI;
read_conf();
check_auth( $ENV{'REMOTE_ADDR'},1 );
# output the HTML headers
print "Content-Type: text/html\n";
print "\n";
print "<html>\n";
print "<head>\n";
print "<title>SetDisp - List of Sets</title>\n";
print "</head>\n";
print "<body>\n";
# connect to the database
$dbienv = conf_get_attr("database","dbienvvar");
if( $dbienv ne "" )
{
    $ENV{$dbienv}= conf_get_attr( "database","dbienvvalue" );
}
$dbhandle=DBI->connect( conf_get_attr("database","dbconnect"),
                        conf_get_attr("database","dbuser"),
                        conf_get_attr("database","dbpass") ) ||
die("Can't connect to database\nReason given was $DBI::errstr\n");
#           0         1         2         3         4         5
$query = "select setno, stitle, stype, scatflag, simages, scodec ".
        ".from ".conf_get_attr("tables","sets")." ".
        "where stype = 'I' and scatflag = 'T' ".
        "order by sadded desc ";
$cursor = $dbhandle->prepare( $query );
$cursor->execute;
print "<table>\n";
$setcount=0;
while( (($results = $cursor->fetchrow_array ) &&
        ($setcount < 25 )) )
{
    # start the row
    print "<tr><td align=center>\n";
    # create the link
    print "<a href=\"".conf_get_attr("server","cgiurl");
    print "wacsindex/page".$results[0].".html\">";
    # print out the set name
    print "<font size=-2 face=\"arial,helv,Helvetica,sans\">";
    print $results[1]."</font></a>\n";
    # end the row
    print "</td></tr>\n";
    $setcount++;
}
print "</table>\n";
print "</body>\n";
print "</html>\n";
```



When we run this set against our demonstration web server, we get the following output which is a list of the sets containing dildo use in most-recent first order.

```
Sabrina BlackTopGreySkirtPinkBraPanties WhiteBedDildoPussyClip
Roxanne RedWhiteTuftsSeeThruBabyDollDressMatchingPanties WhiteSofaRopeLightsChristmasTreeDildo
Sabrina CyanSeeThruLingerieTopWhiteStockingsNoPanties WhiteSofaDildo
Roxanne BrownLeopardPrintBraMatchingPanties WhiteDoubleBedDildo
```

## Adding Icons

While it works and is usable, it's not exactly the greatest web page ever, so let's try and brighten it up a little. It'd be quite nice to be able to include an icon, and of course wacs has the infrastructure to do this for us. In fact, it offers us three different options of what size of icons we'd like: `set`, `std` and `mini`. In this case since we're trying to get a fair number of entries shown, we'll opt for the `mini` version. We get this by calling the `wacsimg` command and specifying that we'd like the `mini` version.

To make this happen we need to add another cell to the table with the HTML `img` tag pointing at `wacsimg`. As before we'll specify both `align` and `valign` properties for this table cell. So if we modify the code, much as we did before for the model icons, we get the following in php:

### Example 4.2. Adding A Set Icon

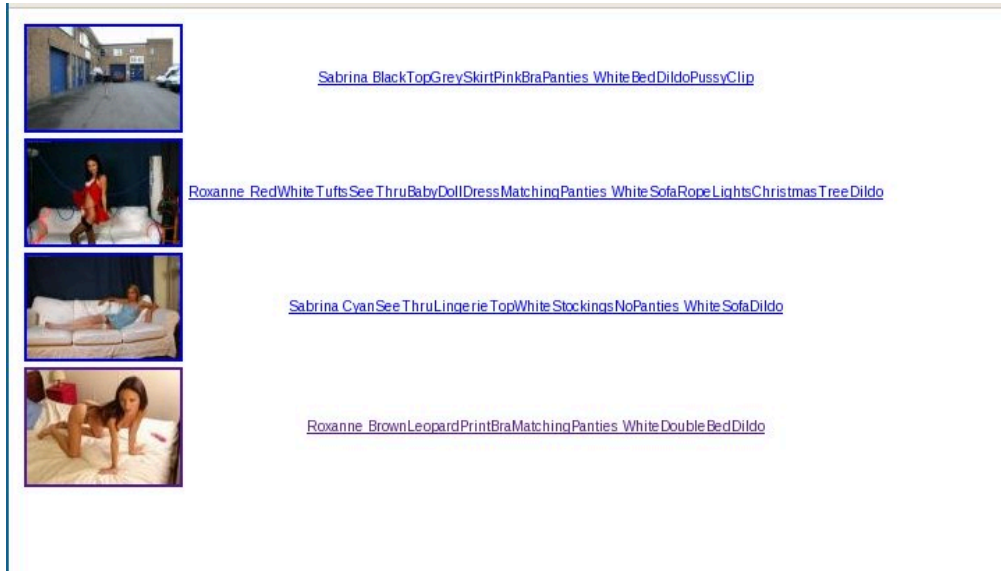
```
// start the row
print "<tr><td valign=top align=center>\n";
// create the link for the icon
print "<a href=\"".$wacs->conf_get_attr("server","cgiurl");
print "wacsindex/page".$results[0].".html\">";
// add the icon itself
print "<img src=\"".$wacs->conf_get_attr("server","cgiurl");
print "wacsimg/mini".$results[0].".jpg\" alt=\"[icon for ";
print $results[0]."]\">";
// end cell, next cell
print "</td><td align=center>\n";
// create the link
```

and of course the same example in perl looks like:

```
# start the row
print "<tr><td valign=top align=center>\n";
# create the link for the icon
print "<a href=\"".$conf_get_attr("server","cgiurl");
print "wacsindex/page".$results[0].".html\">";
# add the icon itself
print "<img src=\"".$conf_get_attr("server","cgiurl");
```

```
print "wacsimg/mini".$results[0].".jpg\" alt=\"[icon for ";
print $results[0]."]\">";
# end cell, next cell
print "</td><td align=center>\n";
# create the link
```

and if we run the resulting program, we get something like this:



## Making The Text More Readable

One of the design decisions taken when designing WACS was to encourage directory names to be the same as the set names, and to make those more usable outside of the WACS system, to make them not include spaces. Instead the so-called *Camel Technique*, so named because of all the humps in it, where an upper case letter signifies the start of each new word. This is used along with a technique where underscores ( \_ ) act as the transitions between the three sections of the set name: these are:

1. Model or Models name(s)
2. Her Clothing
3. Location and Action

However the underscore aspect is only used in the directory name and not in the set title (field `stitle`) as stored in the database which has spaces instead. Amongst our tasks, we will need to replace the spaces with the appropriate HTML table tags.

Fortunately we can use a regular expression to convert the *Camel-Style* text back into something a little bit more readable. This next group of changes to the code are to do exactly that. We're going to take a slightly different approach from before as we're not going to make the split off parts into separate HTML table cells. This is because that makes both the font setting and HTML link creation much more complex - we're merely going to insert a forced line break `<br>` tag into the places where we want a new line to start. Then we're going to break up the Camel-Style text into separate words. We do this with:

Our first substitution is going to be to replace the spaces (the section dividers in the `stitle` field) with the appropriate HTML directives. The second and third ones actually break up the words at the points the case changes:


**Example 4.3. Making Camel-Style Text Readable**

```
// print out the set name
print "<font size=-2 face=\"arial,helv,Helvetica,sans\">";
$prettytext = $results[1];
$prettytext = preg_replace('/\s/','<br>', $prettytext );
$prettytext = preg_replace('/(\w)([A-Z][a-z])/',' $1 $2', $prettytext );
$prettytext = preg_replace('/([a-z])([A-Z])',' $1 $2', $prettytext );
print $prettytext."</font></a>\n";
// end the row
```

To implement the same functionality in perl actually uses exactly the same regular expressions (aka *regex*) but looks very different as it's all done in assignment operations without any explicit function call. There's no `preg_replace` used here. Anyway here is exactly the same functionality in perl:

```
# print out the set name
print "<font size=-2 face=\"arial,helv,Helvetica,sans\">";
$prettytext = $results[1];
$prettytext =~ s/\s/<br>/g;
$prettytext =~ s,(\w)([A-Z][a-z]), $1 $2,g;
$prettytext =~ s,([a-z])([A-Z]), $1 $2,g;
print $prettytext."</font></a>\n";
# end the row
```

With these changes in place, we can once again copy over the code and we have a much more presentable output from the program; here's an example:

	<p><a href="#">Sabrina</a>  <a href="#">Black Top Grey Skirt Pink Bra Panties</a>  <a href="#">White Bed Dildo Pussy Clip</a></p>
	<p><a href="#">Roxanne</a>  <a href="#">Red White Tufts See Thru Baby Doll Dress Matching Panties</a>  <a href="#">White Sofa Rope Lights Christmas Tree Dildo</a></p>
	<p><a href="#">Sabrina</a>  <a href="#">Cyan See Thru Lingerie Top White Stockings No Panties</a>  <a href="#">White Sofa Dildo</a></p>
	<p><a href="#">Roxanne</a>  <a href="#">Brown Leopard Print Bra Matching Panties</a>  <a href="#">White Double Bed Dildo</a></p>

Hopefully with this we've got the output presentation of the sets list looking a whole lot better than it was in the first example. There are of course many more fields within the set database that we could also make use of in our pages. We will return to them when we look at the WACS User Interface Toolkit in Chapter 5, *The User Interface Toolkit*. For now, before we finish our look at sets, we're just going to look at how we find the model or models featured in a given set.

# Connecting Sets And Models

## Understanding The Data Architecture

One of the things that often confuses people about true relational databases is that they are unable to do a one-to-many or many-to-many relationship directly. While many so called *easy-to-use* databases do offer field types that purport to offer such linking, they are problematic and do not fit into any sensible logical model for how things should be structured. Worse, each vendor's implementation (those who do implement it at all) is different and incompatible. However with a sensible schema design, this limitation really isn't a problem at all.

One such instance of this need to link one-to-many is the concept of linking a set with a model within WACS. In the easy case, you'd have thought that you'd simply put the model number into one of the fields in the set schema and the job would be done. But what do you then do when you have two models featuring in a set; easy you might say - one is the main model, the other is a secondary model, so just add a second field for the additional model and put the second number there. Of course that then makes the SQL query more complex each time as you've got to check both fields before you know if a model is in a set or not. It still might work, but it's already getting cumbersome. You might discover a set first by virtue of the additional model and only afterwards identify the *official* primary model.

Just about every adult site we've encountered does feature at least a few sets with three models, so suddenly we're looking at a second additional model field and having to check that as well. And believe me, there are a few sites of which Sapphic Erotica comes to mind in particular where sets with three, four, five or even six models in a single set are relatively common. Simply put, adding models to the sets table just doesn't scale. So we take the proper relational database approach and add an additional schema called **assoc** for associations which gives us these relationships. It's a very simple schema, basically containing a primary key, a model number and a set number.

## Using Relationships With Assoc

The process of finding out who is in a set becomes very simple and straight forward - you simply search the assoc table for the set number you're looking at. If we're looking for who is in set no 123, we simply use the following SQL query:

```
select amodelno from assoc
where asetno = 123
```

We then merely loop through the results of the above query and each record we find is another model involved in this set. If we don't get any results returned, then there aren't any models associated with this particular set. Of course we probably want more than just the model number(s), but that too is relatively simple. Consider the following query:

```
select modelno, mname, mimage, mbigimage
from models, assoc
where modelno = amodelno
and asetno = 123
```

This query simply retrieves the model details for each model who is involved with this particular set, one record at a time. Due to the way relational databases are engineered, this is actually a very quick and

efficient process. The first line of the `where` clause does what is known as a *relational join* and establishes the necessary connection between the `assoc` and `models` tables necessary for what we're trying to do. Additionally it's a very logical and elegant solution that will cope with none, one, two, three, four or as many models as you like within a single simple action.



### Note

Although we make use of the `assoc` table, we don't actually use any results from it - we don't need to - it has silently taken care of handling the connection we needed to make.

## An Example Using Assoc

If we go back to our example program displaying sets, we can modify it to include this activity as a sub-routine. What we're going to do is to divide the right hand side of the output into the two cells, one with the title, and the other with the model(s) featuring in the set. The icon will remain on the left. First step is to add the `rowspan` attribute to the left hand side cell so the icon spans it.

### Example 4.4. Modified Icon Cell

```
// start the row
print "<tr><td rowspan=2 valign=top align=center>\n";
// create the link for the icon
```

and in perl, it'll look very similar:

```
# start the row
print "<tr><td rowspan=2 valign=top align=center>\n";
# create the link for the icon
```

The next step is to create a new function to handle the query to look up the entries in the `assoc` table. We're going to call this function simply `getmodel` and it'll take just one argument, the set number for which we want the model(s) details. It will return to us a potentially quite long string variable containing all the model names that matched surrounded by a link to each model's WACS model page.



### Note

So long as we use a different cursor variable to the database routines we can quite happily run another query and loop through it's results while inside an outer loop looking at the results of a completely different query. This is where the whole concept of a cursor becomes really useful.

**Example 4.5. getmodel Subroutine**

```

function getmodel ( $setno ) {
    global $dbhhandle;
    global $wacs;
    $gmresult='';
    //           0           1           2           3
    $modelquery="select modelno, mname, mimage, mbigimage ".
        "from ".$wacs->conf_get_attr("tables","models").
        ", ".$wacs->conf_get_attr("tables","assoc")." ".
        "where modelno = amodelno ".
        " and asetno = ".$setno." ".
        "order by mname ";
    $modelcursor=$dbhhandle->query( $modelquery );
    // loop through the results
    while( $modelresults = $modelcursor->fetchRow() )
    {
        // do we need a divider?
        if( ! empty( $gmresult ) )
        {
            $gmresult.="<br>";
        }
        // add the model link
        $gmresult.="<a href=\"".$wacs->conf_get_attr(
            "server","cgiurl")."wacsmphumbs/" .
            $modelresults[0]."\>";
        // add her name and close link
        $gmresult.=$modelresults[1]."</a>";
    }
    // return the complete string
    return( $gmresult );
}

```

and the same code implemented in perl looks like this:

```

sub getmodel( $ )
{
    my( $setno )=@_;
    my( $gmresult, $modelquery, $modelcursor, @modelresults );
    $gmresult='';
    #
    $modelquery="select modelno, mname, mimage, mbigimage ".
        "from ".conf_get_attr("tables","models").
        ", ".conf_get_attr("tables","assoc")." ".
        "where modelno = amodelno ".
        " and asetno = ".$setno." ".
        "order by mname ";
    $modelcursor=$dbhhandle->prepare( $modelquery );
    $modelcursor->execute;
    # loop through the results
    while( @modelresults = $modelcursor->fetchrow_array )
    {

```

```
    # do we need a divider
    if( $gmresult ne "" )
    {
        $gmresult.="<br>";
    }
    # add the model link
    $gmresult.="<a href=\"".conf_get_attr("server","cgiurl").
        "wacsmphumbs/".$modelresults[0].\">";
    # add her name and close link
    $gmresult.=$modelresults[1].\"</a>";
}
# return the complete string
return( $gmresult );
}
```

The final step of this process is to add into our main loop going through the retrieved set records a call to the `getmodel` function. This looks like:

#### Example 4.6. Calling The `getmodel` Function

```
// next right hand cell
print "<tr><td align=center><font size=-1>\n";
print getmodel( $results[0] );
print "</font></td></tr>\n";
// increment set count
```

and in perl this looks like

```
# next right hand cell
print "<tr><td align=center><font size=-1>\n";
print getmodel( $results[0] );
print "</font></td></tr>\n";
# increment set count
```

With these changes incorporated into the code, we now have the finished version of the `setdisp` program (**`setdisp4.php`** or **`setdisp4`** in the `samples` directory. If we now copy this script up to the web server and run it, we should see something like this:

	<p><u>Sabrina</u> <u>Black Top Grey Skirt Pink Bra Panties</u> <u>White Bed Dildo Pussy Clip</u></p>
	<p><u>Sabrina</u> <u>Roxanne</u> <u>Red White Tufts See Thru Baby Doll Dress Matching Panties</u> <u>White Sofa Rope Lights Christmas Tree Dildo</u></p>
	<p><u>Roxanne</u> <u>Sabrina</u> <u>Cyan See Thru Lingerie Top White Stockings No Panties</u> <u>White Sofa Dildo</u></p>
	<p><u>Sabrina</u> <u>Roxanne</u> <u>Brown Leopard Print Bra Matching Panties</u> <u>White Double Bed Dildo</u> <u>Roxanne</u></p>

Once again we've gradually developed a program up to the point where it is now offering quite reasonable functionality and layout making use of the WACS programmers toolkit API. Hopefully this has given you an insight into what WACS is capable of and the basics of how to make use of it's API. In due course, we hope to have a respository of WACS skins, or mini-site scripts, which you can download and tailor to your own needs. If in the course of learning the WACS API you write some programs you'd be happy to share with others, please send them to us and we'll include them in the respository.



---

# Chapter 5. The User Interface Toolkit

## Introducing WacsUI

In this chapter, we're going to take a slightly different tack, we're going to just look at code segments you could choose to include within your application, primarily user interface components taken from the User Interface toolkit, WacsUI. This is not going to be an exhaustive review of what is available as that is covered in detail in the reference section, Chapter 7, *WACS API: User Interface Module*. Instead this is just a quick taster of just a few of the calls provided by the **WacsUI** toolkit.

So far we've been dealing with the various routines that are provided by the Core Wacs module - and these relate primarily to configuration parameters and security. There is a second module available for you to use called WacsUI, the WACS User Interface Toolkit. This concerns itself primarily with providing utility functions to ease the tasks of formatting and preparing data from the database into a form more suitable for use in web pages.

## Including WacsUI support

To include support for the WACS User Interface (WacsUI) toolkit within your application, you need to add the following extra lines to your code, ideally just after the Wacs core module.

### Example 5.1. WacsUI initialisation

```
require_once "wacsui.php";

$wacsui = new WacsUI;

and here's the perl dialect of the same activity...

use WacsUI;
```

## WacsUI: DescribeHer

### Example 5.2. Using WacsUI: describeher

```
print $wacsui->describeher(
    array( 'hair'=>$results[4],
          'length'=>>results[5],
          'titsize'=>>results[6],
          'pussy'=>>results[7],
          'race'=>>results[8],
          'build'=>>results[9],
          'height'=>>results[10],
          'weight'=>>results[11],
          'occupation'=>>results[12])) . "\n";
```



## Note

We have to *package up* our parameter list as an array in order to pass it in Php; perl is somewhat simpler with a simple sequence of named parameters.

```
print describeher(  
    hair=>$results[4],  
    length=>$results[5],  
    titsize=>$results[6],  
    pussy=>$results[7],  
    race=>$results[8],  
    build=>$results[9],  
    height=>$results[10],  
    weight=>$results[11],  
    occupation=>$results[12] )."\n";
```

The above example is based upon modifying the MySimple example program from in the second chapter to add the following extra fields into the query: `mhair`, `mlength`, `mtitsize`, `mpussy`, `mrace`, `mbuild`, `mheight`, `mweight`, `moccupation` after the `mimage` (with a comma of course) and before the `from` clause.

## The `addkeyicons` function

Both the `models` and `sets` schemas feature fields that contain a space seperated list of keywords that mark certain attributes found within that set. These can be quickly turned into a small HTML table of icons using the routine `addkeyicons`. The fields suitable for use with this are `scatinfo` from the `sets` table and `mattributes` from the `models` table. These are passed as the first attribute; the second being the displayed size of the icons which for the default icons would be a maximum of 48 x 48 pixels. The function is called simply with:

### Example 5.3. Using `AddKeyIcons`

```
addkeyicons( $results[16], 24 );
```

## WacsUI: Other Functions

Another example of using the `wacsui` module can be found in the `newssets.php` application in the samples directory. This is a more "real world" worked example showing a new releases index page; it makes use of both the `iconlink` and `addkeyicons` functions.

Detailed documentation on each call available and how it works can be found in the API reference section Chapter 7, *WACS API: User Interface Module*.

## Conclusions

We've now come to the end of the basic WACS API tutorial, at least for this edition of the WACS Programmers Guide. It is our intention to expand this section in future editions. Still, it has hopefully introduced you to the key concepts in making use of the WACS Programming API and given you some useful simple programs to build on when creating your own applications. The rest of this book consists of the WACS API reference manual and the WACS Database Schema Reference. If these do not provide

sufficient information, please contact us via the methods listed on the WACS web site at SourceForge [<http://wacsiip.sourceforge.net>].

---

# Part II. WACS API

## Programming Reference

This is the API (Application Programming Interface) reference manual for the WACS environment. It documents the main API calls in both Perl and PHP dialects. There are now six operational modules available as part of the WACS system, plus a utility module used by the installers.

**Table 1. The Key WACS Modules**

WACS Module List		
<i>name</i>	<i>part of</i>	<i>description</i>
<b>Wacs.pm</b>	Core	the main Wacs module
<b>WacsUI.pm</b>	Core	the Wacs User Interface module
<b>WacsStd.pm</b>	Core	the Wacs Standardised Components module
<b>WacsID.pm</b>	Core	the Wacs Identification module
<b>wacs.php</b>	wacs-php	the main Wacs module, Php dialect
<b>wacsui.php</b>	wacs-php	the Wacs user interface module, Php dialect

Chapter 6, *WACS API: Core Module*

Chapter 7, *WACS API: User Interface Module*

Chapter 8, *WACS API: Standard Components Module*

Chapter 9, *WACS API: Identification Module*

---

---

# Chapter 6. WACS API: Core Module

## Core Module: Summary

**Table 6.1. Function Summary: Core Module**

<b>function</b>	<b>description</b>
read_conf	locate and read the XML based configuration file
check_auth	check that this is an authorised access
auth_error	report an authentication error and suggest remedies
auth_user	return the registered username for this IP
add_auth	add a new authentication token to access control system
find_config_location	try to locate the specified XML config file
conf_get_attr	get the requested configuration attribute
auth_get_attr	get the requested access control list attribute
dberror	produce a more helpful error page when db connections fail
gettoday	get today's date as a string suitable for the current DB
timecomps	break a date down into component parts
vendlink	provide a link to the vendors site
getvaluname	takes a single character flag and converts to string
geticonlist	gets the icon array for the specified object type
gettypecolour	get the prevailing colour scheme for the set type
divideup	make a directory name more readable
alsofeaturing	find and list any other models featured in this set
checkexclude	check for this file name being one to ignore/hide
checkindex	check for what might be an index file
makedbSAFE	try to make the returned string safe for use in the database
addheadercss	add standard preamble to enable javascript menus
setgroupperms	set the appropriate group permissions for co-operative updating

## Core Module: Reference

## Core Module: Reference

The following pages contain the \*nix style reference pages for each function call in the WACS core module. These detail what the function does, what parameters it takes, what it returns and which versions of the core library it is available in.

## Name

`read_conf` — read Wacs core config modules

## Synopsis

```
use Wacs;
```

```
read_conf
```

## Summary

The `read_conf` causes the standard WACS XML configuration file, `wacs.cfg` to be parsed and the contents read into internal memory structures in the WACS module for later use by other WACS routines. The main interface to accessing this information is the call `conf_get_attr`.

`read_conf` is sensitive to the environment variable `WACS_CONFIG` which specifies a directory containing an alternative `wacs.cfg` configuration file.

## Availability

`read_conf` is available in both perl and php.

## Name

`check_auth` — check if this IP address is authorised for access

## Synopsis

```
use Wacs;
```

```
check_auth(ip_address, vocal_error);
```

```
scalar ip_address;
```

```
scalar vocal_error;
```

## Summary

`check_auth` checks whether the passed IP address is authorised for access to this Wacs server at this time. This authorisation may be by either permanent or lease permission based upon the calling IP address. This IP address is specified by the first parameter to the function. The second parameter controls what will be done about it: if the value is 0 (zero), the call will merely terminate the session by exiting the program; if the value is 1 (one), an authorisation error HTML page will be displayed offering the user the option to log in. In the Perl version, an additional option of 2 (two) is available which outputs a failure icon in the case of an expired lease and a request for an image file - this is not possible in PHP as the content type of text/html has already been determined.

## Availability

`check_auth` is available in both perl and php.

## Name

`auth_error` — create a reasonable HTML error page with reason and link to remedy

## Synopsis

```
use Wacs;
```

```
auth_error(message);
```

```
scalar message;
```

## Summary

`auth_error` creates a reasonable HTML error page with reason and link to remedy if applicable (ie login page). The message parameter will be placed in a bordered box near the bottom of the message and can be used to convey additional information. `check_auth` sets this to `Sorry, your lease has expired.` when that is the case.

## Availability

`auth_error` is available in both perl and php.



## Name

`auth_user` — return the account name of the user associated with IP address

## Synopsis

```
use Wacs;

scalar auth_user(ip_address);

scalar ip_address;
```

## Summary

`auth_user` returns the account name of the user associated with the specified IP address.

## Availability

`auth_user` is available in both perl and php.

## Name

`add_auth` — add a new authentication token to the access control list

## Synopsis

```
use Wacs;
```

```
add_auth(...);
```

## Parameters

parameter	description
<code>ipaddr</code>	The IP Address of the host being authorised.
<code>user</code>	account name of the user being registered
<code>type</code>	type of registration being undertaken - currently <code>lease</code>
<code>role</code>	level of access granted currently: <code>viewer</code> , <code>power</code> or <code>admin</code>
<code>date</code>	date at which this lease should expire
<code>prefexcl</code>	preference exclusions: the <code>scatflag</code> values not to be shown by default
<code>usedirect</code>	whether to use the <code>usedirect</code> function if supported by the server - can be <code>yes</code> or <code>no</code>
<code>imagepage</code>	whether to create links to framed page or raw ones - should be <code>frame</code> or <code>raw</code>
<code>scaling</code>	when to use image scaling - can be <code>none</code> , <code>slide</code> , <code>slide+page</code> and <code>all</code>
<code>size</code>	size of scaled images when applicable in the format <code>1024x768</code>
<code>quality</code>	jpeg quality setting used when scaling images
<code>delay</code>	desired delay before next image in slideshow

## Summary

`add_auth` adds a new authentication token to the access control list, ie the leases file. This is the action taken by the `wacslogin` command after it has authenticated the user. It can also be used to update the user preferences - it is used by `wacslogin`, `wacspref` and `wacslogout`.

## Availability

`add_auth` is currently available only in perl. A php implementation is possible in a future release if required.

## Name

`find_config_location` — return the location of the requested config file

## Synopsis

```
use Wacs;
```

```
scalar find_config_location(configuration_filename);
```

```
scalar configuration_filename;
```

## Summary

`find_config_location` returns the location of the requested config file. It first checks the directory specified by the `WACS_CONFIG` environment variable, and then tries the built-in list of possible WACS configuration file locations. This list is normally: `/etc/wacs.d`, then `/usr/local/etc/wacs.d` and finally `/opt/wacs/etc/wacs.d`. If the specified file is not found in any of these locations, a null string is returned.



### Note

The location specified by the environment variable `WACS_CONFIG` takes precedence, if and *only if* the requested file is present there. The normal directories are searched afterwards if the file is not found in the directory specified.

## Availability

`find_config_location` is available in both perl and php.

## Name

`conf_get_attr` — get the specified attribute from the config file values

## Synopsis

```
use Wacs;
```

```
scalar conf_get_attr(configuration_section, configuration_attribute);
```

```
scalar configuration_section;
```

```
scalar configuration_attribute;
```

## Summary

`conf_get_attr` returns the specified attribute from the config file or its default value if not specified there. The WACS configuration files are divided into a number of logical sections; the first parameter specifies which of these is required: amongst those defined are `database`, `tables`, `fsloc`, `server`, `security`, `download`, `colours`, `layout`, `precedence` and `debug`. Please see the WACS configuration guide and sample `wacs.cfg` files for more information on what information is available.

## Availability

`conf_get_attr` is available in both perl and php.

## Name

`auth_get_attr` — get the specified attribute from the authorisation file values

## Synopsis

```
use Wacs;
```

```
scalar auth_get_attr(ip_address, authorisation_attribute);
```

```
scalar ip_address;
```

```
scalar authorisation_attribute;
```

## Summary

`auth_get_attr` returns the specified attribute from the authorisation file or it's default value if not specified there. These look ups are based on the IP address of the host - typical attributes include the user name, the preference exclusions, the role, and the various preference settings - see `add_auth` for more info.

## Availability

`auth_get_attr` is available in both perl and php.

## Name

`dberror` — produce a more helpful error page when db connections fail

## Synopsis

```
use Wacs;
```

```
dberror(...);
```

## Parameters

parameter	description
header	Whether to add an HTML preamble or not: n for no, y for yes.
message	The message the end-user should receive
error	The error message returned from the database routines; logged in the web server error log
dbuser	The database user account with which the access was being attempted, from the config file's <code>dbuser</code> entry.
dbhost	The host specification of the database that it was trying to access, from the config file's <code>dbconnect</code> entry when using perl, and the <code>phpdbconnect</code> entry when using PHP

## Summary

The `dberror` function provides a detailed and hopefully helpful error message when the WACS subsystem cannot connect to the database server. It also logs details of the failure to the web server error log.

## Availability

`dberror` is available in both perl and php. It was introduced in Wacs 0.8.1.

## Name

gettoday — get todays date and various relations thereof

## Synopsis

```
use Wacs;
```

```
scalar gettoday(...);
```

## Parameters

parameter	description
format	which format to return date in (DD-MON-YYYYY or YYYY-MM-DD) - default is native format for the current database
epoch	the actual date to convert in Unix seconds since 1970 format.
offset	number of days different from today - assumed to be historical if postive, future if negative - thus yesterday will be 1, a week ago will be 7, tomorrow will be -1.

## Summary

The `gettoday` function returns either todays date or various deviations thereform - yesterday, a week ago, two weeks ago, etc.

## Availability

`gettoday` is available in both perl and php.

## Name

`timecomps` — return seperated time components

## Synopsis

```
use Wacs;
```

```
array timecomps(date_in_db_format);(format);
```

```
scalar date_in_db_format;
```

## Summary

The `timecomps` breaks a database format date up into year, month and day components. The optional **format** parameter can specify a non-native date format for conversion purposes.

## Availability

`timecomps` is available in both perl and php.



## Name

`vendlink` — provide (if possible) a link to the vendor's site for this model

## Synopsis

use `Wacs`;

```
scalar vendlink(...);
```

## Parameters

parameter	description
vendor	the vendor's reference (ie their vsite id)
page	which page to get: valid options are <code>directory</code> , <code>modelpage</code> , <code>bio</code> , <code>vidindex</code> , <code>vidpage</code> , <code>imgpage</code> , <code>altpage</code> , or <code>signup</code> .
name	the model's name
key	the model's key for this site
altkey	the model's alternative key for this site
setkey	the setkey if this request needs it (depends on the value of page above)
sessionkey	the session key (if required and known).
modelno	the WACS model number for this request (believe me we occasionally need this)
setno	the WACS set number for this request (see above - this too)
dbhandle	current handle to the database connection

## Summary

The `vendlink` provides (if possible) a link to a page on the vendor's site for this model or set. Specify the page you require using the `page` parameter - can link to any one of the many pages the vendor database knows about.

## Availability

`vendlink` is available in only in perl at present. If you need it in PHP, please put in a request for it on the sourceforge tracker.

## Name

`getvaluename` — provide the long name for the specified value of specified type

## Synopsis

```
use Wacs;
```

```
scalar getvaluename(...);
```

## Parameters

parameter	description
object	The object you want the mapping for - see <code>geticonlist</code> below
value	The value you want mapped to it's long format (often a single character).

## Summary

The `getvaluename` function returns the long (readable) name for the specified short value of specified fixed values attribute type. For instance, if you want to get the long name for type "M", you call `getvaluename` with `object=>"types"` and `value=>M` and `getvaluename` will return `Masturbation`.

## Availability

`getvaluename` is available in both perl and php.

## Name

`geticonlist` — return the array of attributes to filename/long name mappings.

## Synopsis

```
use Wacs;

hashref geticonlist(requested_object);

scalar requested_object;
```

## Summary

The `geticonlist` function returns an array/ hashref of the legal values for the requested type object. In some cases this will be the filenames of the icon for the attributes; in other cases it'll be the single character legal values and their long form names. Valid requests include: `models`, `sets`, `types`, `media`, `dstatus`, `regions`, `flags` and `pussy`.

## Availability

`geticonlist` is available in both perl and php.

## Name

`gettypecolour` — return the background colour for this type of set

## Synopsis

```
use Wacs;
```

```
scalar gettypecolour(set_type);
```

```
scalar set_type;
```

## Summary

The `gettypecolour` returns the HTML colour specification for the background of the current set type. Pass it the set stype value `I`, `V`, etc.

## Availability

`gettypecolour` is available in both perl and php.

## Name

divideup — make Camel-style text more readable and add HTML markup

## Synopsis

use Wacs;

```
scalar divideup(original_text, divider, already_small_font);
```

```
scalar original_text;
```

```
scalar divider;
```

```
scalar already_small_font;
```

## Summary

The `divideup` function returns a more readable version of the so-called Camel Style wording used in creating WACS directories. It also embeds HTML directives to try and ensure that even long entries don't take up too much space. The first argument is the original text (typically the field stitle from the sets database), the second (`divider`) is typically the HTML break tag `<br>` but could be other things like a table divider sequence `</td><td>` . The third parameter signifies whether the font in use is already small - if set to 0 (zero), HTML tags to reduce the font size be based on using size is -1 for long lines; if it's set to 1 (one) it'll be assumed they were already using size is -2, and will therefore use size = -3.

## Availability

`divideup` is available in both perl and php.

## Name

`alsofeaturing` — look for any other models also featuring in this set

## Synopsis

```
use Wacs;
```

```
scalar alsofeaturing(...);
```

## Parameters

parameter	description
setno	The set number of this set
primary	The model number we already know about for this set; exclude this model from the results. Leave blank if you want all models listed.
staysmall	stay in a small font - if this is set to Y font change specifications will not cause a size change.
linkto	which wacs application to link to (assumed to be in cgi-bin)
dbhandle	current handle to the database connection

## Summary

The `alsofeaturing` function returns a list of models featured in this set along with links to an appropriate WACS application.

## Availability

`alsofeaturing` is currently available only in perl. It is likely to be moved to WacsUI and also be implemented in php in the near future.

## Name

checkexclude — test for being a directory file or other reserved purpose name

## Synopsis

```
use Wacs;  
  
scalar checkexclude(filename);  
  
scalar filename;
```

## Summary

The checkexclude returns 1 if the file is one of those that should be excluded from consideration (ie . or .. or one of ours like .info or .unpack). If the file looks genuine, returns 0.

## Availability

checkexclude is available only in perl as it is just used for collection management tasks.

## Name

checkindex — try to guess if this is an index image file

## Synopsis

```
use Wacs;
```

```
scalar checkindex(filename);
```

```
scalar filename;
```

## Summary

The `checkindex` tries to guess if a given file name is likely to be an index file or a regular image file based upon its name. If it's a name associated with index files, it returns 1; if it isn't `checkindex` returns 0.

## Availability

`checkindex` is available in only perl as it is really only appropriate to collection management tools.



## Name

`makedbSAFE` — try to make the returned string safe for use in the database

## Synopsis

```
use Wacs;
```

```
scalar makedbSAFE(...);
```

## Parameters

parameter	description
string	the string of text to be considered
allow	characters to allow which are not normally acceptable: at present only forward slash (/) is recognised
deny	characters to deny which are normal acceptable: at present any space character (space, tab, newline) given here will cause any whitespace characters to be stripped out.

## Summary

The `makedbSAFE` function is designed to remove characters which are unsuitable for feeding to the database. It normally works with a default set of rules, which implicitly disallows forward slash (but this can be explicitly allowed with `allow=>'/'`). Similarly white space can be removed from a file name when required using the `deny` option.

## Availability

`makedbSAFE` is available in both `php` and `perl`. This function was added in `Wacs 0.8.1`.

## Name

`addheadercss` — prints out the header cascading style sheet preamble

## Synopsis

```
use Wacs;
```

```
addheadercss(css_preamble_type);
```

```
scalar css_preamble_type;
```

## Summary

The `addheadercss` prints out the required css preamble to support the appropriate pull down menu system. At present only one type, "csshoriz" is recognised, but additional options can be added.

## Availability

`addheadercss` is available in both perl and php.

## Name

setgroupperms — set group permissions to allow both command line and web management of sets.

## Synopsis

```
use Wacs;
```

```
setgroupperms( . . . );
```

## Parameters

parameter	description
target	pathname of the file or directory to update
group	the unix group to set permissions to (usually wacs, can be obtained with conf_get_attr on security -> admingroup.
mode	access mode that should be set - typically ug+rwX.

## Summary

The setgroupperms function sets the group permissions on the specified file to allow updating by both command line tools and the web interface. This is typically done by making all files group-writable to the wacs group of which both apache and the approved WACS administrative users should be members.

## Availability

setgroupperms is available only in perl as it is used only for collection management tasks.

---

# Chapter 7. WACS API: User Interface Module

## User Interface Module: Summary

**Table 7.1. Function Summary: User Interface Module**

<b>function</b>	<b>description</b>
describeher	tries to make a sensible sentence out of model data
whatshedoes	describes the kind of sets this model appears in
addkeyicons	makes a little HTML table with the attribute icons
addratings	makes a little HTML table with the set ratings
iconlink	build a link around the icon for this set
addlinks	add standard top-of-the-page menus
read_menu	read the XML menu files and create menu record structure
menu_get_default	get the default link for the menu title
menu_get_title	get the menu title itself
menu_get_body	get the body of the menu
menu_get_entry	get a single entry from the menu

## User Interface Module: Reference

## Name

describeher — tries to make a sensible sentence out of model data

## Synopsis

```
use Wacs;
```

```
use WacsUI;
```

```
scalar describeher(...);
```

## Parameters

parameter	description
hair	The colour of her hair
length	The length of her hair
titsize	The size of her breasts
cupsize	The cupsize of her breasts if known
pussy	The usual style of her pubic hair
race	Her race (in broad terms)
eyes	The colour of her eyes
distmarks	distinguishing marks - easy ways to recognise her
build	her physical build/body type
height	her height in centimetres (NB: field not suitable for imperial values)
weight	her weight in kilograms (NB: field not suitable for imperial values)
vitbust	her bust measurement in centimetres
vitwaist	her waist measurement in centimetres
vithips	her hips measurement in centimetres
occupation	her occupation (if stated)
aliases	other names she's known by

## Summary

The `describeher` tries to make a readable biography entry out of the various model attribute parameters in the `model` table. The result is returned as a string.

## Availability

`describeher` is available in both perl and php.

## Name

whatshedoes — describes the kind of sets this model appears in

## Synopsis

```
use Wacs;
```

```
use WacsUI;
```

```
scalar whatshedoes(...);
```

## Parameters

parameter	description
solo	does she feature in solo sets (Y, N)
straight	does she feature in straight sets (Y, N)
lesbian	does she feature in lesbian sets (Y, N)
fetish	does she feature in any sets flagged as fetish
toys	does she use toys in any of her sets
masturbation	does she masturbate in any of her sets
other	does she do any activites marked as other

## Summary

The `whatshedoes` function takes the truth values for doing certain kinds of activities and makes it into a descriptive sentence which is returned as a string.

## Availability

`whatshedoes` is available in both perl and php.

## Name

addkeyicons — makes a little HTML table with the attribute icons in

## Synopsis

```
use Wacs;
```

```
use WacsUI;
```

```
addkeyicons(list_of_attribute_keywords, icon_size);
```

```
scalar list_of_attribute_keywords;
```

```
scalar icon_size;
```

## Summary

The `addkeyicons` function takes a space separated list of attribute keywords such as the sets table `scatinfo` field or the models table `mattributes` field and prints out the associated icons in a small HTML table. It scales the icons to the specified size in doing so.

## Availability

`addkeyicons` is available in both perl and php.

## Name

`addratings` — makes a little HTML table with the ratings icons in

## Synopsis

```
use Wacs;
```

```
use WacsUI;
```

```
addratings(...);
```

## Parameters

parameter	description
overall	The overall rating for the set (1 to 5)
variety	How unusual the content or action of the set is
techqual	The technical quality of the photography, lighting and set
size	How big the icons should be: normal or small
orientation	whether the table should be vertical or horizontal
title	display title on table: y for yes, n for no.

## Summary

The `addratings` function is similar to `addkeyicons` in that it outputs an HTML table with icons in. In this case, it's the ratings icons for each of the three main set ratings: overall, variety and techqual. It can display the table in two sizes, with or without a title and in a horizontal or vertical orientation.

## Availability

`addratings` is available in both perl and php. This function was introduced in Wacs 0.8.1



## Name

`iconlink` — build a link around the icon for this set

## Synopsis

```
use Wacs;
```

```
use WacsUI;
```

```
iconlink(...);
```

## Parameters

parameter	description
<code>type</code>	set type value (I, V, etc)
<code>setno</code>	The set number
<code>sarea</code>	The toplevel area of the set
<code>scategory</code>	The middle level area of the set
<code>sdirectory</code>	The lower level area of the set
<code>model</code>	The model's name - used in the alt tag in the images
<code>resize</code>	Whether to resize or not - 0 is actual size, 1 is rescaled to standard size, 2 is rescaled to mini size
<code>destloc</code>	Which configuration variable to use for location of link destination application - typically <code>cgiurl</code> for perl scripts and <code>wacsurl</code> for php scripts
<code>destapp</code>	The stem of the URL to link to around the icon, something like <code>wacsindex/page</code> , needs to include any parameter introducers like <code>page</code> or <code>setid=</code>
<code>destext</code>	The extension of the URL to link to, or null, ie <code>.html</code> or <code>.php</code>

## Summary

The `iconlink` function displays the icon for a set at the requested size surrounded by an appropriate link to the set concerned.

## Availability

`iconlink` is available in both perl and php. The `destloc`, `destapp` and `destext` options are only available in 0.8.1 or later.

## Name

addlinks — add standard top-of-the-page menus

## Synopsis

```
use Wacs;
```

```
use WacsUI;
```

```
addlinks(...);
```

## Parameters

parameter	description
myname	name of the calling program
context	general area of the current page: possible values are <code>modelindex</code> , <code>models</code> , <code>search</code> , <code>tags</code> , <code>newimage</code> , <code>newvideo</code> or <code>admin</code>
title	Title of the menu (not currently used)
exclude	name of link to exclude (normally this apps name so it doesn't link to itself)
mode	menu mode: either <code>normal</code> for old-style simple top line menu or <code>csshoriz</code> to use javascript pull down menus
options	optional parameter list (array)
optdesc	matching descriptions for the above

## Summary

The `addlinks` function is a generalised interface to adding a top of the page menu - you specify a general category into which the page you're writing falls, and it adds an appropriate selection of the standard menus.

## Availability

`addlinks` is currently only available in perl. It is assumed php programmers will want more fine-grained control over their menus and so it has not been implemented.

## Name

`read_menu` — read the XML menu files and create menu record structure

## Synopsis

```
use Wacs;  
  
use WacsUI;  
  
read_menu(menu_filename);  
  
scalar menu_filename;
```

## Summary

The `read_menu` reads the specified menu XML file into the internal data structures of the `wacsui` object. It should be called before using any of the other menu routines. For the standard system menus, the collection management tools use the file `menu.cfg` in the `wacs` config directory (usually `/etc/wacs.d`). You can edit the standard menu file to add your own additional menu definitions for use in specific applications. If your application wishes to use an alternate namespace, you could specify an alternate menu config name, something like `mysite.cfg` and also place it in the `wacs` config directory.

## Availability

`read_menu` is available in both perl and php.

## Name

`menu_get_default` — get the default link for the menu title

## Synopsis

```
use Wacs;
```

```
use WacsUI;
```

```
scalar menu_get_default(...);
```

## Parameters

parameter	description
<code>name</code>	the menus name; typically in lower case (eg <code>navigation</code> )
<code>caller</code>	name of the calling application
<code>exclude</code>	applications to exclude from menus; typically the calling application itself
<code>options</code>	an array of options to be substituted.
<code>optdesc</code>	a matching array of descriptions

## Summary

The `menu_get_default` returns the default link for the top-of-the-page menu title before the menu pull-down is activated. Normal substitutions are applied to this option if specified.

## Availability

`menu_get_default` is available in both perl and php.

## Name

`menu_get_title` — get the menu title itself

## Synopsis

```
use Wacs;
```

```
use WacsUI;
```

```
scalar menu_get_title(...);
```

## Parameters

parameter	description
name	Name of the menu whose title you want

## Summary

The `menu_get_title` function returns the readable title for the specified menu. This is typically what the link address returned by `menu_get_default` will surround.

## Availability

`menu_get_title` is available in both perl and php.

## Name

`menu_get_body` — get the body of the menu

## Synopsis

```

use Wacs;

use WacsUI;

scalar menu_get_body(...);

```

## Parameters

parameter	description
name	name of the menu concerned
caller	name of the program calling it
exclude	name of program to exclude from menus
options	array of options to use
optdesc	array of matching descriptions for the options above
isarea	hashref/array of image-based sarea values
vsarea	hashref/array of video-based sarea values
mflags	hashref/array of model flags
vsites	hashref/array of vendor codes and names
pre	prefix for generated entries (eg <code>&lt;li&gt;&lt;a href=\"</code> )
intra	middle section for generated entries (eg <code>\ "&gt;</code> )
post	postfix for generated entries (eg <code>&lt;/a&gt;&lt;/li&gt;</code> )

## Summary

The `menu_get_body` function returns a big string containing the HTML formatted body of the requested menu. Using the `pre`, `intra` and `post` parameters you can include the correct entry pre-amble, mid-section and tail-section for your desired menu layout.

## Availability

`menu_get_body` is available in both perl and php.

## Name

`menu_get_entry` — get a single entry from the menu

## Synopsis

```

use Wacs;

use WacsUI;

scalar menu_get_entry(...);

```

## Parameters

parameter	description
name	name of the menu concerned
caller	name of the program calling it
entry	hashref/array of the current entry object from menu tree
options	array of options to use
optdesc	array of matching descriptions for the options above
isarea	hashref/array of image-based sarea values
vsarea	hashref/array of video-based sarea values
mflags	hashref/array of model flags
vsites	hashref/array of vendor codes and names
pre	prefix for generated entries (eg <code>&lt;li&gt;&lt;a href=\"</code> )
intra	middle section for generated entries (eg <code>\ "&gt;</code> )
post	postfix for generated entries (eg <code>&lt;/a&gt;&lt;/li&gt;</code> )

## Summary

The `menu_get_entry` takes an individual menu entry (which may result in multiple menu entry lines) and processes it into a string that is returned. It is available separately as it can be called with custom parameters via options and optdesc to do specific non-standard parameters.

## Availability

`menu_get_entry` is available in both perl and php.

---

# Chapter 8. WACS API: Standard Components Module

## Standard Components Module: Summary

**Table 8.1. Function Summary: Standard Components Module**

<b>function</b>	<b>description</b>
masthead	creates a top-of-the-page summary for any page handling set
modelheads	adds the icons with links for model(s) specified
findmodel	creates a table and choice box for models with a given name
kwscore_reset	resets the keyword scoring system back to defaults
kwscore_process	process the provided string looking for keywords
kwscore_get	get the specified result from the processing of the strings provided previously
removedups	remove duplicates from an attribute string
removeconflicts	remove items that contradict the set attributes from the model attributes

## Standard Components Module: Reference

The `WacsStd` module contains standard components for building the standard WACS collection management tool interface. Since all these tools are written in perl, this module is only implemented in perl.



## Name

masthead — top of page banner for set-based apps

## Synopsis

```
use WacsStd;  
  
masthead( . . . );
```

## Parameters

parameter	description
setno	The set number
stype	The set type (single letter database format)
scatinfo	The attributes for the set
scatflag	The set type flag (single letter database format)
stitle	The assigned set title, aka standard description
sofftitle	The official title (usually from original site)
sarea	Toplevel directory entry
scategory	Middle level directory entry
sdirectory	lowest level - actual holding directory (filename for videos)
simages	Number of images in the set
sindexes	Number of index images for the set
saspect	aspect ratio (mainly for videos)
sformat	file format for this set (.jpg, .png, .mov, .wmv etc)
sdurhrs	video or DVD scene duration - hours value
sdurmin	video or DVD scene duration - minutes value
sdursec	video or DVD scene duration - seconds value
sphotog	photographer reference code (references pref in photographer)
sfoundry	organisation where the set came from
modelno	associated model number
downloadno	associated download record number
useicon	when working with a set number 0, attempt to get an icon by asking for a thumbnail of the first image
addlinks	add set browsing links to the masthead centre section
width	make the masthead table the specified width only
dbhandle	the current database handle object

## Summary

`masthead` generates a standard top-of-the-page banner heading for any page that is intended to document or amend a standard set record. It does a *best efforts* with whatever fields it has passed to it.

## Availability

`masthead` is only available in Perl.

## Name

`modelheads` — adds the icons with links for model(s) specified

## Synopsis

```
use WacsStd;
```

```
modelheads(lookup_method, set_number, dbhandle);
```

```
scalar lookup_method;
```

```
scalar set_number;
```

```
scalar dbhandle;
```

## Summary

The `modelheads` function was originally written as part of the implementation of `masthead` but has broader uses. It provides a table of a model (or group of models) headshots with ratings and name. The *lookup\_method* can be one of `byset` (where it's the models featured in the specified set number) or `byno` (where the second argument is the *model number* rather than the set number). The default option in other cases is any models who've been added today - it is recommended you specify `bydate` and pass the date for this option.

## Availability

`modelheads` is currently only available in perl.

## Name

`findmodel` — creates a table and choice box for models with a given name

## Synopsis

```
use WacsStd;  
  
use WacsUI;  
  
findmodel(...);
```

## Parameters

parameter	description
<code>mname</code>	the model name or beginning of the name to look for
<code>offeralt</code>	Whether to offer an alternative choice or not: y or n
<code>offervalue</code>	What the value returned for the alternative should be, eg <code>next</code>
<code>offercapt</code>	What the caption for the alternative value should be
<code>incsubmit</code>	Whether to include a submit button or not: y or n
<code>dbhandle</code>	pointer to the currently active database handle
<code>cgihandle</code>	pointer to the currently active CGI object

## Summary

The `findmodel` function takes the name of a model and searches the database for who it might conceivably be. It checks the model's name, her aliases and the name from each of her ID map entries. It presents a headshot, description, and a radio button to allow her to be chosen. It can optionally offer an additional radio button for another purpose. The chosen model's number or `next` will be returned in a CGI variable called `modelno`.

## Availability

`findmodel` is only available in perl at this time

## Name

`kwscore_reset` — resets the keyword scoring system back to defaults

## Synopsis

```
use WacsStd;  
  
kwscore_reset(scope);  
  
scalar scope;
```

## Summary

The `kwscore_reset` function resets the currently built attributes table. It is possible to run the `kwscore_process` function several times with different fields from the database and so it does not naturally reset the internal table of results - this call provides that facility and should always be called before each new set to consider. The `scope` parameter is currently ignored but may in future modify the behaviour.

## Availability

As keyword scoring is a collection administration activity, it is currently only implemented in perl.

## Name

`kwscore_process` — process the provided string looking for keywords

## Synopsis

```
use WacsStd;
```

```
kwscore_process(...);
```

## Parameters

parameter	description
string	the string to be processed against the keyword database
dbhandle	the database session object pointer

## Summary

The `kwscore_process` function allows you to submit a string to the keyword scoring system for consideration. Its scores will be stored allowing both retrieval of results and modification of those results by subsequent invocation of the `kwscore_process` with alternative strings. It is perfectly possible to consider both the title (field `stitle`) and the official title (field `soffttitle`) if that is appropriate. It could also be run on the description of the set if that is present.

## Availability

As a collection administration function, `kwscore_process` is currently only available in perl.

## Name

`kwscore_get` — get the specified result from the processing of the strings provided previously

## Synopsis

```
use WacsStd;
```

```
kwscore_get(...);
```

## Parameters

parameter	description
what	which result you are requesting: valid ones are: <code>cat</code> , <code>loc</code> , <code>det</code> , <code>attr</code> or <code>other</code> .
default	a default value you want returned if nothing is found for this request

## Summary

The `kwscore_get` function retrieves the results from any `kwscore_process` calls made since the last `kwscore_reset`. The `what` argument specifies what to return:- `cat` returns a category flag (`scatflag` etc.), `loc` returns a location (`slocation`), `det` returns a detailed location (`slocdetail`), `attr` returns the attributes (`scatinfo` and `other` is available for future expansion).

## Availability

As a collection administration function, `kwscore_get` is currently only available in perl.

## Name

removedups — remove duplicates from an attribute string

## Synopsis

```
use WacsStd;  
  
scalar removedups(raw_attribute_list);  
  
scalar raw_attribute_list;
```

## Summary

The `removedups` function removes any duplicate entries from a space-separated list of attributes - this is typically necessary when merging more than one source of attribute information like that from the `kwscore_get` function and the result of fetching model attributes. Please also see `removeconflicts` function below.

## Availability

As a collection administration function, `removedups` is currently only available in perl.



## Name

`removeconflicts` — remove items that contradict the set attributes from the model attributes

## Synopsis

```
use WacsStd;  
  
scalar removeconflicts(...);
```

## Parameters

parameter	description
<code>model</code>	The model's attributes ( <code>mattributes</code> field)
<code>existing</code>	The existing combined attributes (ie those taken from the set <code>scatinfo</code> field)

## Summary

The `removeconflicts` is designed to stop contradictory overwriting of mutually exclusive model attributes - typically those relating to pubic hair trimming, as these can often vary between sets of the same model. It is provided with the model's attributes plus the existing set attributes - if the existing set attributes do not include a contradictory value, then the model's attributes are included. If there's a conflict, the model's pubic hair attribute is dropped in favour of that in the set. This is usually the correct behaviour. This if a model is normally considered to have a shaven pussy, but appears in a set before she's shaven it (or even as she does so), then the set may be marked with the hairy attribute. If that is there, the model's default of shaven will be removed and only her other attributes (tattoos, piercings, etc) will be imported.

## Availability

As a collection administration function, `removeconflicts` is currently only available in perl.

---

# Chapter 9. WACS API: Identification Module

## Identification Module: Summary



### Warning

The existing identification module has many flaws and it is intended to massively overhaul it in the near future. We would not recommend utilising functions from this module at the present time. If you need a particular routine listed here, please contact us and we'll consider moving it to one of the more stable modules if appropriate

**Table 9.1. Function Summary: Identification Module**

<b>function</b>	<b>description</b>
<b>ident_img</b>	Identify characteristics of an image set from download info
<b>ident_vid</b>	Identify characteristics of a video clip from download info
<b>reset_attr</b>	reset the global attribute table
<b>id_get_flag</b>	get previously determined flag (run ident_* first)
<b>id_get_info</b>	get previously determined catinfo (run ident_* first)
<b>id_get_photog</b>	get previously determined photographer (run ident_* first)
<b>id_get_dnldno</b>	get download record number
<b>id_get_modelno</b>	get the model number
<b>id_get_modelname</b>	get the model's name
<b>id_get_vendor</b>	get the vendor reference
<b>id_get_dbhandle</b>	get the current DB handle
<b>id_get_key</b>	get the current models id at the current vendor
<b>id_get_setkey</b>	get the set key at the current vendor
<b>id_get_setname</b>	get the name of the most recent set
<b>id_get_status</b>	get the status of the most recent set
<b>id_get_notes</b>	get the current value of the notes field
<b>id_get_setno</b>	get the current value of the setno field
<b>addassoc</b>	Add a new model/set association
<b>dnld_img</b>	retrieve a download record based upon the namestem
<b>dnld_markdone</b>	mark the download as having been done
<b>dnld_checkadd</b>	take a download specification and if not existing, add it
<b>dnld_update</b>	update an attribute in the download record
<b>vend_dnld</b>	return the download location for this vendor
<b>alloc_nextkey</b>	allocate the next free primary key for named table
<b>find_cookies</b>	find the current web browser cookies file (firefox only)
<b>vid_getsize</b>	get the size of a video clip
<b>extractphotog</b>	find the photographer name (only works on ATK at present)
<b>find_namestem</b>	find the common stem of a file name
<b>id_mpage</b>	process a modelpage looking for links to suitable sets
<b>chk_vid_type</b>	check to see if this url is a video file type

---

# Part III. WACS Database Schema

This is the Database Schema Reference Manual, or data dictionary, for the WACS environment. This documents the database tables in use, their contents, structure, relationships and assigned values.

The WACS database schemas are built with the convention that the first letter of the schema name is prefixed to all fields within that schema. Thus a field from the sets schema will start with the letter *s*, a field from the assoc schema will start with the letter *a* and so on. Generally related fields will have fundamentally the same name, such that the set number is *setno* in the sets schema, *asetno* in the assoc schema, *tsetno* in the tags schema, *dsetno* in the download schema, and so on. This makes performing relational joins much easier and more portable since one can do the likes of `where amodelno = modelno` without any ambiguity and without having to specify the table name explicitly.

Where possible fields with a limited set of possible values will be single character fields with a reasonably neumatic value for each possible value. Thus the media type (*stype*, *dtype*, etc) is **V** for Video Clip, **I** for Image Set, **D** for DVD scene, and so on. A lookup hash of the legal values will typically be available for programmers to use from the core *Wacs* module (see the Part II, “WACS API Programming Reference” for more details).

- Chapter 10, *Schema Reference: Sets*
- Chapter 11, *Schema Reference: Assoc*
- Chapter 12, *Schema Reference: Idmap*
- Chapter 13, *Schema Reference: Models*
- Chapter 14, *Schema Reference: Download*
- Chapter 15, *Schema Reference: Photographer*
- Chapter 16, *Schema Reference: Tag*
- Chapter 17, *Schema Reference: Vendor*
- Chapter 18, *Schema Reference: Conn*
- Chapter 19, *Schema Reference: Keyword*

---

# Chapter 10. Schema Reference: Sets

## Sets: Schema SQL

```
create table sets
( setno          number(9) primary key,
  stype          char(1) not null,
  sstatus       char(1) not null,
  sauto         char(1),
  srating       char(1),
  sflag         char(1),
  stechqual     number(2),
  svariety      number(2),
  svisits       number(2),
  sformat       varchar2(10),
  scodec        varchar2(40),
  stitle        varchar2(240),
  sofftitle     varchar2(240),
  sofficon      varchar2(160),
  saddicon      varchar2(160),
  sname         varchar2(80),
  shair         varchar2(80),
  smodelno      varchar2(40),
  slocation     varchar2(20),
  slocdetail    varchar2(40),
  sattire       varchar2(20),
  sphotog       varchar2(6) references photographer,
  ssource       varchar2(80),
  sfoundry      varchar2(80),
  sproddate     date,
  sreldate      date,
  suscattr      char(1),
  snotes        varchar2(240),
  sdesc         varchar2(2048),
  sindexes      number(6),
  simages       number(6),
  sdurhrs       number(2),
  sdurmin       number(2),
  sdursec       number(2),
  slandx        number(6),
  slandy        number(6),
  sportx        number(6),
  sporty        number(6),
  saspect       varchar(10),
  sbytes        number(12),
  sdvdno        number(6),
  sdvddisc      number(2),
  sdvdttitle    number(3),
  sdvdstartch   number(3),
  sdvdendch     number(3),
```

```

sidlogo          char(1),
serrors          char(1),
sduplicates      number(9),
scatinfo         varchar2(160),
scatflag         char(1),
snamestem        varchar2(80),
sdownload        varchar2(160),
sarea            varchar2(160),
scategory        varchar2(160),
sdirectory       varchar2(240),
scomments        varchar2(240),
sadded           date,
samended         date
);

```



**Note**

sattire is a new field introduced into the schema in release 0.8.1; it will not be used until the 0.8.2 release cycle (or later). It is intended to hold a general category of the model's clothing, derived from keywords; expected to include values like Casual, Sports, Lingerie, Elegant, etc.

## Sets: Defined Values

**Table 10.1. stype: Type of Set: defined values**

stype	
<b>I</b>	Image Set
<b>V</b>	Video Clip
<b>A</b>	Audio File
<b>S</b>	DVD Scene

**Table 10.2. sstatus: Status of Set: defined values**

sstatus	
<b>M</b>	Manually Added, Details Not Checked
<b>A</b>	Automatically Added, Details Not Checked
<b>N</b>	Normal - Checked
<b>G</b>	Good - Thoroughly Checked
<b>U</b>	Unknown

**Table 10.3. sauto: Automatic Update of Set Allowed?: defined values**

sauto	
<b>N</b>	None (no auto updates)
<b>L</b>	(on-disk) Location only - all attributes manual
<b>A</b>	Append only - all existing entries stay
<b>F</b>	Fully auto-generated - all values can change

**Table 10.4. srating: Overall Rating For The Set: defined values**

srating	
<b>5</b>	Finest
<b>4</b>	Very Good
<b>3</b>	Good
<b>2</b>	Reasonable
<b>1</b>	Mediocre
<b>0</b>	None Specified

**Table 10.5. stechqual: Technical Quality Rating For The Set: defined values**

srating	
<b>5</b>	Finest - HD Video done well, Multi-megapixel stills
<b>4</b>	Very Good - Well lit SD or good HD Video, good megapixel + stills
<b>3</b>	Good - Well done low-res SD, good sub-megapixel stills; not quite so good but higher res
<b>2</b>	Reasonable - either very small, or bad equipment (flash on camera) used moderately well
<b>1</b>	Mediocre - lack of skill, bad equipment, poor composition
<b>0</b>	None Specified

**Table 10.6. svariety: Unusualness Rating For The Set: defined values**

svariety	
<b>5</b>	Very Unusual - look at the set scenario and think "What the F***!"
<b>4</b>	Unusual - unusual and very interesting - "Wow"
<b>3</b>	Neat - interesting and impressive but not quite "Wow"
<b>2</b>	Cute Twist - a slightly unusual twist, unusual pose etc
<b>1</b>	Ordinary - can still score very highly in overall and tech
<b>0</b>	None Specified

**Table 10.7. sformat: Format of the File(s) In The Set: defined values**

<b>sformat</b>	
<b>JPEG</b>	JPEG image
<b>GIF</b>	GIF image
<b>PNG</b>	PNG image
<b>PNM</b>	PNM,PBM,PGM,PPM image
<b>WMV</b>	Windows Media Player Video
<b>AVI</b>	AVI Video (codec specified separately)
<b>QT</b>	QuickTime .mov Video (codec specified separately)
<b>MPEG</b>	MPEG Video (MPEG-1 or 2)

**Table 10.8. sidlogo: Presence of Burnt-in Logo: defined values**

<b>sidlogo</b>	
<b>U</b>	Unknown
<b>Y</b>	Yes - image/video has burnt-in logo
<b>N</b>	No - image/video is clean of bugs

**Table 10.9. serrors: Presence of Known Errors: defined values**


<b>serrors</b>	
<b>N</b>	None detected
<b>F</b>	Fixed - faulty images/video have been fixed - Quality may have been compromised - sizes/signatures no indicative of original
<b>E</b>	Encoding Only - causes message but renders OK
<b>C</b>	Some Corrupt Images/Segments of video

**Table 10.10. scatflag: Generalised type of the set: defined values**

<b>scatflag</b>	
<b>F</b>	Fuck - straight sex
<b>L</b>	Lesbian - lesbian sex
<b>G</b>	Group - more than two people having sex, mixed-gender
<b>T</b>	Toy - Solo but uses toys such as dildo, vibrator, etc
<b>S</b>	Solo - Model on her own (possibly with a non-participatory audience)
<b>M</b>	Masturbation - Solo but includes masturbation activities
<b>N</b>	None - not determined yet
<b>B</b>	Backstage - Behind The Scenes set featuring this model
<b>C</b>	Clothed - non-nude set featuring this model
<b>D</b>	Duplicate - duplicate set - maybe from a different site



**Table 10.11. slocation: generalised description of locations: recommended values**

slocation (recommended values)	
	<b>Note</b> This is a <i>Recommended Values</i> list only; additional values can be added as appropriate
<b>Balcony</b>	Balcony or Terrace; outdoors but not part of Garden
<b>Bathroom</b>	Bathroom, Toilet or Shower Cubicle
<b>Bedroom</b>	Bedroom or other sleeping area
<b>Country</b>	Country - including Beach, Forest, and Fields
<b>Dining Room</b>	Dining Room or Eating Area
<b>Garden</b>	Garden or other private outdoor area
<b>Hallway</b>	Hallway, Staircase or Entrance
<b>Kitchen</b>	Kitchen or Kitchen area of apartment
<b>Laundry</b>	Laundry, Cleaning or Utility Area
<b>Lounge</b>	Lounge, Sitting Room or Other Seating Area
<b>Office</b>	Office, including Home PC desk
<b>Other Room</b>	Any other room - (Domestic) Library, Junk Room, Garage, etc
<b>Specialised</b>	Specialised Location: Swimming Pool, Shop, Recording or TV Studio, Factory, Railway Station, etc; additional details can be placed in slocdetail.
<b>Sports</b>	Location associated with Sports and Exercise: Gym, Locker Room, etc.
<b>Studio</b>	White or other plain background Photographic Studio - but NOT Television or Audio recording studios as a feature of the set theme

**Table 10.12. suscattr: how to generate the 18 USC 2257 declaration: defined values**

suscattr	
<b>V</b>	Vendor based - use vendor's USC declaration address
<b>P</b>	Photographer based - use photographer's address for USC declaration
<b>N</b>	Suppress declaration - <i>NOT RECOMMENDED FOR US RESIDENTS</i>
<b>G</b>	Generic - include generic text with all vendor addresses

---

# Chapter 11. Schema Reference: Assoc

## Assoc: Schema SQL

```
create table assoc
( assocno                number(9) primary key,
  amodelno              number(6) references models,
  asetno                number(9) references sets,
  astatus               char(1),
  aadded                date,
  aamended              date
);
```

## Assoc: Defined Values

**Table 11.1. astatus: association status: defined values**

astatus	
<b>M</b>	Manually Added
<b>G</b>	Generated Automatically
<b>R</b>	Relationship entry - not the primary model for this set.

---

# Chapter 12. Schema Reference: Idmap

## Idmap: Schema SQL



### Note

A possible future direction is for this table to be relationally linked to the vendors table such that `idmap.isite = vendor.vsite`

```
create table idmap
( identryno          number(7) primary key,
  imodelno           number(6) references models,
  istatus            char(1),
  isite              varchar2(20) not null,
  ikey               varchar2(30),
  ialtkey            varchar2(30),
  iname              varchar2(30),
  inotes             varchar2(80),
  iactive            char(1),
  ichanged           date,
  ichecked           date,
  iadded             date,
  iamended           date
);
```

## Idmap: Defined Values


**Table 12.1. istatus: idmap status: defined values**

istatus	
<b>M</b>	Manually Added
<b>A</b>	Generated Automatically
<b>I</b>	Imported From Another WACS site

**Table 12.2. iactive: model activity status as this identity: defined values**

iactive	
<b>Y</b>	Yes - active model (refresh list with auto tools)
<b>D</b>	Dormant - no new sets for a while (don't bother checking)
<b>N</b>	No - inactive (id not known)
<b>O</b>	Obsolete - old reference (no longer there)

**Table 12.3. isite: Some recommended site abbreviations: recommended values**

isite (recommended values)	
 <b>Note</b> This is a <i>Recommended Values</i> list only; additional values can be added as appropriate	
<b>ALS</b>	ALSScan.com
<b>AMK</b>	AMKingdom.com (aka ATK Galeria)
<b>ATE</b>	ATKExotics.com
<b>ATKP</b>	ATKPremium.com
<b>AW</b>	AbbyWinters.com
<b>FJ</b>	FemJoy.com
<b>IFG</b>	infocusgirls.com
<b>JAFN</b>	jennyandfriends.net
<b>KPC</b>	karupspc.com (aka Karup's Private Collection)
<b>KHA</b>	karupsha.com (aka Karup's Hometown Amateurs)
<b>SE</b>	sapphicerotica.com
<b>TF</b>	teenflood.com
<b>WACSD</b>	WACS Demo site (coming soon)

---

# Chapter 13. Schema Reference: Models

## Models: Schema SQL



### Note

Please notice that the use of metric in the vital statistics is not intended to be a dig at the imperial measurements, merely that it reliably and consistently conveys the necessary information as sensible, manageable integers. Utility functions are planned to make it easier to convert and update in a future release of WACS. You try writing an SQL query to find models between 5ft 3ins and 5ft 6ins in height, as compared to between 160 and 168 cms in height. See what I mean?

```
create table models
( modelno          number(6) primary key,
  mname            varchar2(40),
  mhair            varchar2(15),
  mlength          varchar2(20),
  mtitsize         varchar2(10),
  mcupsize         char(1),
  meyes            varchar2(15),
  mrace            varchar2(15),
  mattributes      varchar2(60),
  malias           varchar2(60),
  mdisting         varchar2(80),
  musual           varchar2(60),
  mimage           varchar2(80),
  mbigimage        varchar2(80),
  mstatus          char(1),
  mrating          char(1),
  mpussy           char(1),
  mflag            char(1),
  mvideos          char(1),
  msolo            char(1),
  mstraight        char(1),
  mlesbian         char(1),
  mfetish          char(1),
  mmast            char(1),
  mtoys            char(1),
  mother           char(1),
  mnsets           number(4),
  mnimages         number(7),
  mnvideos         number(4),
  mcountry         varchar2(30),
  mhometown        varchar2(80),
  mage             number(3),
  mageyear         number(4),
  mcstatus         char(1),
  mvitbust         number(4),
  mvitwaist        number(4),
```

```

mvithips      number(4),
mbuild       char(1),
mheight      number(3),
mweight      number(3),
moccupation  varchar2(30),
mcontact     varchar2(80),
mnotes       varchar2(240),
mbio         varchar2(240),
madded       date,
mamended     date
);

```

## Models: Defined Values

**Table 13.1. mstatus: model record status: defined values**

mstatus	
<b>A</b>	Automatically Added, Details Not Checked
<b>M</b>	Manually Added, Details Not Checked
<b>N</b>	Normal - Checked
<b>G</b>	Good - Thoroughly Checked
<b>P</b>	Placeholder - Not Real Person

**Table 13.2. mrating: model rating: defined values**

mrating	
<b>5</b>	Finest (included in Q= searches and front page)
<b>4</b>	Very Good (included in Q= searches and front page)
<b>3</b>	Good (not included in Q= searches, included in front page)
<b>2</b>	Reasonable (not included in Q= searches or front page)
<b>1</b>	Mediocre (not included in Q= searches or front page)
<b>0</b>	None Specified (listed in U= searches)


**Table 13.3. mpusy: model's normal pubic hair style: defined values**

mpusy	
<b>H</b>	Hairy
<b>T</b>	Trimmed
<b>B</b>	Brazilian style shaved - very little hair above clit area
<b>S</b>	Shaven - completely
<b>V</b>	Varies (best avoided, try and pick one of above - her <i>usual style</i> )
<b>N</b>	Not Specified

**Table 13.4. mflag: special marking flag for models: defined values**

mflag	
<b>S</b>	Favourite Solo
<b>L</b>	Favourite Lesbian
<b>C</b>	Favourite Cutie
<b>F</b>	Favourite Straight
<b>M</b>	Current Featured Model
<b>P</b>	Placeholder (not a real person)

**Table 13.5. model activities flags: defined values**

model activities flags	
fieldname	possible values
 <b>Note</b> Automatically updated by <b>updatestats</b>	
<b>mvideos</b>	<b>Y</b> - Yes, does this; <b>N</b> - No, doesn't do this
<b>msolo</b>	
<b>mstraight</b>	
<b>mlesbian</b>	
<b>mfetish</b>	
<b>mmast</b>	
<b>mtoys</b>	
<b>mother</b>	

**Table 13.6. mcstatus: accuracy of home country field: defined values**

mcstatus	
<b>C</b>	Certain - country of origin stated in bio
<b>I</b>	Inferred - from location or other models seen with
<b>G</b>	Guess - based on photographer or building style
<b>N</b>	None

**Table 13.7. mrace: race of the model: defined values**

<b>mrace</b>	
<b>Caucasian</b>	Caucasian - European Descent aka White
<b>Oriental</b>	Oriental - Chinese, Japanese, SE Asian
<b>Asian</b>	Indian Sub-Continent - India, Pakistan, etc
<b>Negroid</b>	Negroid - of African Descent aka Black
<b>Aboriginal</b>	Aboriginal - indigenous peoples - First Nation, Polynesian, etc
<b>Latina</b>	Latin American - aka Hispanic
<b>Mixed</b>	Mixed race and others

**Table 13.8. mbuild: body type of the model: defined values**

<b>mbuild</b>	
<b>V</b>	Very Slim
<b>S</b>	Slim
<b>M</b>	Medium
<b>H</b>	Heavy

**Table 13.9. vital statistics: meanings**

<b>vital statistics</b>	
<b>mweight</b>	Weight in Kilos
<b>mheight</b>	Height in centimetres
<b>mvitbust</b>	Bust measurement in centimetres (vital stats part 1)
<b>mvitwaist</b>	Waist measurement in centimetres (vital stats part 2)
<b>mvithips</b>	Hips measurement in centimetres (vital stats part 3)



---

# Chapter 14. Schema Reference: Download

## Download: Schema SQL

```
create table download
(
  downloadno          number(7) primary key,
  dmodelno           number(6) references models,
  dsetno             number(9) references sets,
  dstatus            char(1),
  dtype              char(1),
  dsite              varchar2(20) not null,
  dkey               varchar2(30),
  dsetkey            varchar2(40),
  dsetname           varchar2(240),
  dsetflag           char(1),
  dnotes             varchar2(240),
  durl               varchar2(240),
  darchive           varchar2(240),
  dsignature         varchar2(82),
  dsize              number(9),
  dpulled            date,
  dadded             date,
  damended           date
);
```

## Download: Defined Values


**Table 14.1. dstatus: download status: defined values**

dstatus	
<b>U</b>	Not Yet Attempted
<b>F</b>	Failed - Retry when possible
<b>S</b>	Successful - set registered in database, available
<b>P</b>	Pending - downloaded, awaiting unpacking
<b>A</b>	Aborted - don't download for some reason
<b>D</b>	Deferred - held back from being downloaded
<b>R</b>	Relationship Entry - a second model for a set
<b>L</b>	Liasion - a proto-Relationship Entry not yet linked
<b>E</b>	Error - not the right model, etc
<b>I</b>	In Progress - download currently in progress
<b>X</b>	Incomplete - record of it's existance but too little info to download it

**Table 14.2. dtype: download set type: defined values**

dtype	
<b>I</b>	Image Set
<b>V</b>	Video Clip
<b>A</b>	Audio File

**Table 14.3. dsetflag: Suggested value for scatflag based on parsing result**

dsetflag	
	<p><b>Note</b></p> <p>Any valid value for scatflag from the sets table. This is a hint on the set type based upon the parsing process picking out keywords</p>

---

# Chapter 15. Schema Reference: Photographer

## Photographer: Schema SQL

```
create table photographer
( pref          varchar2(6) primary key,
  pname         varchar2(40),
  aliases      varchar2(80),
  pgender      char(1),
  paddress     varchar2(120),
  pemail       varchar2(80),
  pwebsite     varchar2(80),
  pusual       varchar2(40),
  pregion      varchar2(20),
  pcountry     varchar2(50),
  plocation    varchar2(50),
  pstyledesc   varchar2(80),
  prating      number(2),
  phardness    number(2),
  psolo        char(1),
  ptoys        char(1),
  plesbian     char(1),
  pstraight    char(1),
  pgroup       char(1),
  pfetish      char(1),
  pdigital     char(1),
  pfilm        char(1),
  pvideo       char(1),
  phdvideo     char(1),
  pcamera      varchar2(40),
  pcamnotes    varchar2(80),
  pcomments    varchar2(240),
  pnotes       varchar2(240),
  pbiography   varchar2(1024),
  padded       date,
  pamended     date
);
```

## Photographer: Defined Values

**Table 15.1. pgender: gender of the photographer: defined values**

pgender	
<b>M</b>	Male
<b>F</b>	Female
<b>U</b>	Unknown

**Table 15.2. pregion: geographical location of the photographer: defined values**

pregion	
<b>Europe</b>	Europe
<b>North America</b>	USA and Canada
<b>South America</b>	South and Central America
<b>Middle East</b>	Middle East (brave photographer!)
<b>Asia</b>	Asia (India and the Indian Sub-continent ONLY)
<b>Orient</b>	Orient (Asia excluding Indian Sub-continent)
<b>Australasia</b>	Australia and New Zealand
<b>Africa</b>	Africa
<b>Other</b>	Other

**Table 15.3. prating: overall rating of photographer: defined values**

prating	
<b>0</b>	None
<b>1</b>	Awful - poor equipment and technique
<b>2</b>	Poor - uninteresting and badly composed/exposed work
<b>3</b>	Reasonable - technically OK, but very unenterprising
<b>4</b>	Good - good technique, interesting compositions and direction
<b>5</b>	Excellent - Excellent technique, interesting and challenging compositions and direction

**Table 15.4. phardness: rating of how explicit this photographer can be: defined values**

phardness	
<b>0</b>	None - Not Rated
<b>1</b>	Soft-focus (very arty)
<b>2</b>	Glamour - sharp but no open leg, genital detail, etc
<b>3</b>	Normal - wide range of shots but not particularly strong
<b>4</b>	Hard (close-ups)
<b>5</b>	Fetish - pretty extreme, gaping, etc

**Table 15.5. photographer activities covered flags: defined values**

photographer activities covered flags	
fieldname	possible values
psolo	Y - Yes, does this; N - No, doesn't do this; O - Occasionally does this
ptoy	
plesbian	
pstraight	
pgroup	
pfetish	

**Table 15.6. photographer technologies used flags: defined values**

photographer technologies used flags	
fieldname	possible values
pdigital	Y - Yes, uses this technology; N - No, doesn't use this technology.
pfilm	
pvideo	
phdvideo	

---

# Chapter 16. Schema Reference: Tag

## Tag: Schema SQL

```
create table tag
( tagno                number(9) primary key,
  tmodelno            number(6) references models,
  tsetno              number(9) references sets,
  tstatus             char(1),
  tflag               char(1),
  tgroup              number(6),
  tdesc               varchar2(40),
  towner              varchar2(20),
  texpiry             date,
  tadded              date,
  tamended            date
);
```

## Tag: Defined Values

**Table 16.1. tstatus: tag entry status: defined values**

tstatus	
<b>T</b>	Temporary - expire as per expiry rules
<b>V</b>	Viewed, Temporary - expire as per expiry rules, hide from index
<b>P</b>	Permanent - don't expire, show in index
<b>A</b>	Archived - don't expire, don't show in normal indexes

**Table 16.2. tflag: tag content type status: defined values**

tflag	
<b>M</b>	Model-based tag entry
<b>S</b>	Set-based tag entry

---

# Chapter 17. Schema Reference: Vendor

## Vendor: Schema SQL


```
create table vendor
( vsite          varchar2(20) primary key,
  vname          varchar2(45),
  vshortname     varchar2(20) not null,
  vregion        varchar2(20),
  vcountry       varchar2(50),
  vweburl        varchar2(120),
  vsignup        varchar2(120),
  vrating        number(2),
  vtechrates     number(2),
  vuscdecl       varchar2(240),
  vcurrent       char(1),
  vshow          char(1),
  vsubscribed    char(1),
  vuntil         date,
  vusername      varchar2(80),
  vpassword      varchar2(30),
  vidting        number(2),
  vidtvid        number(2),
  vcomexcl       varchar2(240),
  vmdirectory    varchar2(240),
  vmdiruse       char(1),
  vmdirpages     number(3),
  vmpage         varchar2(240),
  vmpaguse       char(1),
  vmbio          varchar2(240),
  vmbiouse       char(1),
  vmvideos       varchar2(240),
  vmviduse       char(1),
  vvidpage       varchar2(240),
  vviduse        char(1),
  vimgpage       varchar2(240),
  vimguse        char(1),
  valtpage       varchar2(240),
  valtuse        char(1),
  vsrvimg        varchar2(240),
  vsrvvid        varchar2(240),
  vmultimg       char(1),
  vmultvid       char(1),
  vnotes         varchar2(240),
  vadded         date,
  vamended       date
);
```

# Vendor: Defined Values

**Table 17.1. vcurrent: vendor existance status: defined values**

<b>vcurrent</b>	
<b>Y</b>	Yes - still an active site
<b>N</b>	No - no longer trading at that web address

**Table 17.2. vshow: vendor index inclusion status: defined values**

<b>vshow</b>	
	<p><b>Note</b></p> <p>This option only really affects vendormode and vendor-based lists of models; if you don't use vendor mode, it's not likely to be relevant.</p>
<b>Y</b>	Yes - show in indices
<b>N</b>	No - hide from indices

**Table 17.3. vmdiruse et al: vendor URL auto-usuability status: defined values**

<b>vmdiruse et al</b>								
<i>fieldname</i>	<i>page purpose</i>	<i>possible values</i>						
<b>vmdiruse</b>	Model Directory	<table border="1"> <tr> <td><b>Y</b></td> <td>link is (auto)usable</td> </tr> <tr> <td><b>N</b></td> <td>link is not (auto)usable</td> </tr> <tr> <td><b>S</b></td> <td>link usable only with session key</td> </tr> </table>	<b>Y</b>	link is (auto)usable	<b>N</b>	link is not (auto)usable	<b>S</b>	link usable only with session key
<b>Y</b>	link is (auto)usable							
<b>N</b>	link is not (auto)usable							
<b>S</b>	link usable only with session key							
<b>vmpaguse</b>	Model Page							
<b>vmbiouse</b>	Model Biography							
<b>vmviduse</b>	Model's Videos Page							
<b>vviduse</b>	Video Set Page							
<b>vimguse</b>	Image Set Page							
<b>valtuse</b>	Alternate Image Set Page							



---

# Chapter 18. Schema Reference: Conn

## Conn: Schema SQL

```
create table conn
( centryno          number(9) primary key,
  cgroup           number(6),
  corder           number(3),
  cflag            char(1),
  cstatus          char(1),
  cmodelno        number(6) references models,
  csetno           number(9) references sets,
  cphotog         varchar2(6) references photographer,
  ctype           varchar2(20) not null,
  cdesc           varchar2(80),
  ccomments       varchar2(240),
  cpath           varchar2(160)
  cadded          date,
  camended        date
);
```

## Conn: Defined Values



### Warning

Conn (connections) is a recent addition and not all parts of the toolchain are in place yet. As the management tools are added, it is expected that at least the legal values for fields will change and be expanded.

**Table 18.1. cflag: connection type: defined values**

cflag	
A	Ad-Hoc - A casual index of some random theme
G	Gallery - A slightly more focused collection with a specific concept behind it.

**Table 18.2. cstatus: connection entry status: defined values**

cstatus	
M	Manually Added
T	Imported from a Tag set

---

# Chapter 19. Schema Reference: Keyword

## Keyword: Schema SQL

```
create table keyword
( kentryno          number(9) primary key,
  kflag             char(1),
  kword             varchar(30) not null,
  kexclusions       varchar(120),
  kiloc             varchar(20),
  kiscore            number(1),
  kicat             char(1),
  kicscore           number(1),
  kidet             varchar(40),
  kidscore           number(1),
  kiattr            varchar(30),
  kiascore           number(1),
  kiother            varchar(40),
  kioscore           number(1),
  knotes            varchar(80),
  kadded            date,
  kamended           date
);
```

## Keyword: Defined Values

**Table 19.1. kflag: active entry status: defined values**

<b>kflag</b>	
<b>A</b>	Applies to All Added
<b>N</b>	Not Active (Ignore)

---

# Index

## A

- addheadercss, 59
- addkeyicons, 64
  - Using ..., 36
- addlinks, 67
- addratings, 65
- add\_auth, 43
- alsofeaturing, 55
- assoc
  - astatus values, 92
  - Field Listing, 92
  - making connections, 30
- astatus, 92
- auth\_error, 41
- auth\_get\_attr, 46
- auth\_user, 42

## C

- cflag, 107
- checkexclude, 56
- checkindex, 57
- check\_auth, 40
- Configuration
  - Reading The..., 4
- Configuration Values
  - Getting..., 5
- conf\_get\_attr, 5, 45
- conn
  - cflag values, 107
  - cstatus values, 107
  - Field Listing, 107
- Connection
  - Database, Initialising..., 4
- cstatus, 107

## D

- Data Architecture, 30
- Database
  - Environment Variables, 5
  - Fetching Records..., 7
  - Initialising Connection To..., 4
- dberror, 47
- describeher, 62
  - WacsUI: Introducing, 35
- divideup, 54
- download
  - dsetflag values, 100
  - dstatus values, 99
  - dtype values, 100
  - Field Listing, 99

- dsetflag, 100
- dstatus, 99
- dtype, 100

## F

- findmodel, 77
- find\_config\_location, 44

## G

- geticonlist, 52
- gettoday, 48
- gettypecolour, 53
- getvaluename, 51

## I

- iactive, 93
- iconlink, 66
- icons
  - adding set ..., 27
- idmap
  - Field Listing, 93
  - iactive values, 93
  - isite recommended values, 93
  - istatus values, 93
- isite, 93
- istatus, 93

## K

- keyword
  - Field Listing, 108
  - kflag values, 108
- kflag, 108
- kwscore\_get, 80
- kwscore\_process, 79
- kwscore\_reset, 78

## M

- makedbSAFE, 58
- masthead, 74
- mbuild, 98
- mcstatus, 97
- menu\_get\_body, 71
- menu\_get\_default, 69
- menu\_get\_entry, 72
- menu\_get\_title, 70
- mfetish, 97
- mflag, 97
- mheight, 98
- mlesbian, 97
- mmast, 97
- modelheads, 76
- models
  - activities values, 97

- connection to sets, 30
  - Field Listing, 95
  - mbuild values, 98
  - mcstatus values, 97
  - mflag values, 97
  - mpussy values, 96
  - mrace values, 97
  - mrating values, 96
  - mstatus values, 96
  - vital statistics fields, 98
  - Modules
    - Importing WACS API, 3
  - mother, 97
  - mpussy, 96
  - mrace, 97
  - mrating, 96
  - msolo, 97
  - mstatus, 96
  - mstraight, 97
  - mtoys, 97
  - mvideos, 97
  - mvitbust, 98
  - mvithips, 98
  - mvitwaist, 98
  - mweight, 98
  - MySimple (Sample Program)
    - Perl Version Source Code, 11
    - Php Version Source Code, 10
    - Sample Run Output, 12
  - MySimple2 (Sample Program)
    - Sample Run Output, 14
  - MySimple3 (Sample Program)
    - Sample Run Output, 17
  - MySimple4 (Sample Program)
    - Sample Run Output, 18
  - MySimple5 (Sample Program)
    - Sample Run Output, 21
- P**
- pdigital, 103
  - pfetish, 103
  - pfilm, 103
  - pgender, 101
  - pgroup, 103
  - phardness, 102
  - phdvideo, 103
  - photographer
    - activities covered values, 103
    - Field Listing, 101
    - pgender values, 101
    - phardness values, 102
    - prating values, 102
    - pregion values, 102
    - technologies used values, 103
  - plesbian, 103
  - prating, 102
  - pregion, 102
  - psolo, 103
  - pstraight, 103
  - ptoys, 103
  - pvideo, 103
- R**
- readable
    - making Camel-Style ..., 28
  - read\_conf, 39
  - read\_menu, 68
  - Relational Database Model, 30
  - removeconflicts, 82
  - removedups, 81
- S**
- sauto, 88
  - scatflag, 90
  - serrors, 90
  - SetDisp (Sample Program)
    - Sample Run Output, 27
  - setdisp program, 24
  - SetDisp2 (Sample Program)
    - Sample Run Output, 28
  - SetDisp3 (Sample Program)
    - Sample Run Output, 29
  - SetDisp4 (Sample Program)
    - Sample Run Output, 33
  - setgroupperms, 60
  - sets
    - connecting to models, 30
    - Field Listing, 87
    - introduction to displaying, 24
    - sauto values, 88
    - scatflag values, 90
    - serrors values, 90
    - sformat values, 89
    - sidlogo values, 90
    - slocation recommended values, 91
    - srating values, 89
    - sstatus values, 88
    - stechqual values, 89
    - styp values, 88
    - suscattr values, 91
    - svariety values, 89
  - sformat, 89
  - sidlogo, 90
  - slocation, 91
  - SQL
    - Simple Example, 6

srating, 89  
sstatus, 88  
stechqual, 89  
Structure of a WACS app, 3  
stype, 88  
suscattr, 91  
svariety, 89

## T

tag  
    Field Listing, 104  
    tflag values, 104  
    tstatus values, 104  
text  
    Camel-Style, 28  
tflag, 104  
timecomps, 49  
tstatus, 104

## U

Using relationships, 30

## V

vcurrent, 106  
vendlink, 50  
vendor  
    Field Listing, 105  
    vcurrent values, 106  
    vmdiruse values, 106  
    vshow values, 106  
vmdiruse, 106  
vshow, 106

## W

WACS Core  
    addheadercss, 59  
    add\_auth, 43  
    alsofeaturing, 55  
    auth\_error, 41  
    auth\_get\_attr, 46  
    auth\_user, 42  
    checkexclude, 56  
    checkindex, 57  
    check\_auth, 40  
    conf\_get\_attr, 45  
    dberror, 47  
    divideup, 54  
    find\_config\_location, 44  
    geticonlist, 52  
    gettoday, 48  
    gettypecolour, 53  
    getvaluename, 51  
    makedbSAFE, 58

    read\_conf, 39  
    setgroupperms, 60  
    timecomps, 49  
    vendlink, 50  
WACS Std  
    findmodel, 77  
    kwscore\_get, 80  
    kwscore\_process, 79  
    kwscore\_reset, 78  
    masthead, 74  
    modelheads, 76  
    removeconflicts, 82  
    removedups, 81  
WACS UI  
    addkeyicons, 64  
    addlinks, 67  
    addratings, 65  
    describeher, 62  
    iconlink, 66  
    menu\_get\_body, 71  
    menu\_get\_default, 69  
    menu\_get\_entry, 72  
    menu\_get\_title, 70  
    read\_menu, 68  
    whatshedoos, 63  
WacsUI  
    addkeyicons, 36  
    describeher, 35  
    Including Support For..., 35  
    Introduction To..., 35  
    whatshedoos, 63