Jig Mapper

The Jig Object-Document-Mapper is an implementation of the abstract Active Record Cursor class (cursor). Have a look into it for additional method descriptions.

Namespace: `\DB\Jig`
File location: `lib/db/jig/mapper.php`

Instantiation

To use the Jig ODM, create a valid Jig DB Connection (jig#constructor) and follow this example:

```
$mapper = new \DB\Jig\Mapper(\DB\Jig $db, string $file);
```

If you'd like to create a model class, you might like to wrap it up:

```
$f3->set('DB',new DB\Jig('data/'));

class User extends \DB\Jig\Mapper {
    public function __construct() {
        parent::__construct( \Base::instance()->get('DB'), 'users.json' );
    }
}

$user = new User();
$user->load(array('@_id = ?','515c570f28de6'));
// etc.
```

Definition: The primary key of Jig documents is named `_id` .

Syntax

$filter

The `$filter` argument for Jig accepts the following structure:

```
// array value for parameterized queries
array( string $expr [, string $bindValue1 [, string $bindValue2 [, ...]]] )
```

The `$expr` part must contain a valid code expression, where all mapper fields are prefixed by a `@`-char. You can bind values to them with positional or named tokens. Here is an example:

```
// positional tokens
array('@username = ? and @password = ?','John','acbd18db4cc2f85cedef654fccc4a4d8')
// named tokens
array('@username = :user and @password = :pw',':user'=>'John',':pw'=>'acbd18db4cc2f85
cedef654fccc4a4d8')
```

Important: Jig is a schema-less document mapper, so the fields of a document may vary from one record to another. To create a valid `$expr` string, keep in mind to add additional field existence checks to prevent running into weird undefined variable errors. Adding some checks for that can be achieved easiely by adding some `isset` conditions:

```
array(
    '(isset(@username) && @username == ?) && (isset(@password) && @password = ?)',
    'John','acbd18db4cc2f85cedef654fccc4a4d8'
    )
```

Info: You can use all common comparison operators in your condition and a single `=` works too.

Search

The best way to search in Jig is to use some `preg_match` conditions:

```
$userList = $user->find(array('(isset(@email) && preg_match(?,@email))','/gmail/'));
// returns all users with an email address that contains GMAIL

// ends with gmail.com => /gmail\.com$/
// starts with john  => /^john/
```

The equivalent of a SQL `IN` operator goes like this:

```
$user->find(array('in_array('_id',array(1,2,3))'));
```

If your document uses an array field, i.e. tags, you can find all posts by tag with just switching the in_array parameters:

```
$post->find(array('isset(@tags) && in_array("fat-free",@tags)'));
```

$option

The `$option` argument for Jig accepts the following structure:

```
array(
    'order' => string $orderClause,
    'limit' => integer $limit,
    'offset' => integer $offset
)
```

i.e:

```
array(
    'order' => 'score SORT_DESC, team_name SORT_ASC',
    'limit' => 20,
    'offset' => 0
)
```

Methods

exists

Return TRUE if the given field is defined

```
bool exists( string $key )
```

set

Assign a value to a field

```
scalar|FALSE set( string $key, scalar $val )
```

This class takes advantage of the Magic class (magic) and ArrayAccess interface. It means you can set and get variables with direct access like this:

```
$mapper->foo = 'bar';
$mapper['foo'] = 'bar';
```

get

Retrieve value of field

```
scalar|FALSE get( string $key )
```

clear

Clear value of field

```
NULL clear( string $key )
```

fields

Return field names of the mapper object

```
array fields( )
```

cast

Return fields of the mapper object as an associative array

```
array cast( [ object $obj = NULL ] )
```

token

Convert tokens in string expression to variable names

```
string token( string $str )
```

find

Return records that match a given criteria

```
array|FALSE find( [ array $filter = NULL [, array $options = NULL [, int $ttl = 0 [,
bool $log = TRUE ]]]] )
```

count

Count records that match a given criteria

```
int count( [ array $filter = NULL [, $ttl = 0 ]] )
```

insert

Insert a new record

```
array insert( )
```

update

Update the current record

```
array update( )
```

erase

Delete the current record

```
bool erase( [ array $filter = NULL ] )
```

copyfrom

Hydrate the mapper object using an array

```
NULL copyfrom( array | string $var [, callback $func = NULL ] )
```

This function allows you to hydrate the mapper using an array (or the name of a hive variable containing an array).

`$func` is the callback function to apply to the hive array variable:

```
if ($func)  $var = $func($var);
```

copyto

Populate hive array variable with mapper fields

```
NULL copyto( string $key )
```

factory

Convert an array to a mapper object

```
protected object factory( string $id, array $row )
```

This *protected* method is used internally by the `select` method.