PHP OBJECT GENERATOR
RELATIONS

INTENDED AUDIENCE

The intended audience for this document is medium / advanced PHP developers who already have some experience generating simple objects using POG. By simple, we mean objects that are not 'connected' to each other. Essentially, this document will not make sense to you if you aren't familiar with POG already. If you are new to POG, please take a look at the following 'essential' articles before proceeding:

- Introduction to PHP Object Generator
- Database wrappers and POG
- POG PART I, ESSENTIALS

RELATIONS

This article is a follow up to POG PART I, ESSENTIALS. In addition to the 5 essential CRUD methods, POG can also generate *additional* methods which facilitate object relations in PHP4 and PHP5. POG can generate 3 types of connected objects:

- ***PARENT***
- ***CHILD***
- ***SIBLING***

When an object is identified as type 'Parent' or 'Child' or 'Sibling', POG generates the usual 5 CRUD methods as well as *additional* methods (referred to as *relations methods*) that accelerate application development even more. What follows is a description of the relations methods as well as code samples that demonstrate how they can be used.

Please note that these additional methods are only generated if a developer specifies that an object is either a Parent, Child or Sibling. Otherwise the object will be generated with only the usual 5 CRUD methods.

PARENT OBJECTS

A Parent object is POG's implementation of one-to-one and one-to-many relations. If your application requires this type of object relations, you can generate objects using this feature. All parent objects in POG require a corresponding Child object.

CHILD OBJECTS

A Child object is POG's implementation of one-to-one and one-to-many relations. If your application requires this type of object relations, you can generate objects using this feature. All parent objects in POG require a corresponding Parent object.

SIBLING OBJECTS

A Sibling object is POG's implementation of many-to-many relations. If your application requires this type of object relations, you can generate objects using this feature. All sibling objects in POG require a corresponding sibling object.

CHILD -> SET {PARENT}

The Set {Parent} Relations method allows you to associate a parent object to a child object. When a child object is generated, POG automatically adds a ParentId attribute to the child object. When Set {parent} is called, POG essentially associates the object id of the parent to the child

CODE EXAMPLES:

To set an Author object as the parent of a child object:

```
$author = new Author(); //create a parent object
$author -> name = 'Michael Critchton';
$author -> Save();


$book = new Book();  //create a child object
$book -> title = "State of Fear";
$book -> SetAuthor($author); //this associates the parent to the child
$book -> Save();
```

Note:

child -> Set {parent} is almost equivalent to **parent -> Add {Child}** (see below). The main difference is that Set {parent} requires that the parent object be previously saved before associating a child to it. Depending on the code context, the developer can choose which method is most convenient to him/her.

CHILD -> GET {PARENT}

The Get {Parent} Relations method allows you to get the parent object associated with a child object.

CODE EXAMPLES:

To get the parent object of a child object :

```
$author = new Author();                 //create a parent object
$author -> name = 'Michael Critchton';

$book = new Book();                     //create a child object
$book -> title = "State of Fear";
$author -> AddBook($book);              //(see below).
$author -> Save(true);


$author = $book -> GetAuthor();         // retrieves the parent of the book object
echo $author -> name;                   // prints "Michael Critchton
```

PARENT -> ADD {CHILD}

The Add {Child} Relations method allows you to ***add a reference of a child*** *object to a parent object. The child object* ***reference*** *is added to a private children Array inside the Parent object. The child is not saved to the database until you save the parent or until you save the child to the database manually. As of POG 2.5, the same child will not be added more than once to the parent if the child already has a childId.*

CODE EXAMPLES:

To Add a book object to an Author object, simply do the following:

```
$book = new Book();                    //create a child object
$book -> title = "State of Fear";
$author = new Author();                //create a parent object
$author -> name = 'Michael Critchton';
$author -> AddBook($book);             //associate child to parent
```

Note: in PHP 4 (and PHP 4 only), since object references are not handled intuitively, adding children **through iteration** will not work as expected. For example, the following does not work in PHP 4

```
Foreach (childList as child)
{
      $parent->AddChild($child)
}
```

Instead, do the following:

```
Foreach ( array_keys($childList ) as $key )
{
      $child = childList[$key];
      $parent->AddChild($child)
}
```

The above code forces PHP to pass a reference of the child object to the AddChild method.

PARENT -> GET {CHILDREN} LIST

The Get {Children} List Relations method allows you to retrieve an array of child objects associated with the parent object. The list of child objects includes children that have already been saved to the database as well as child objects that haven't

Get {Children} List returns an array of child objects.

Get {Children} List can take an optional conditions array (similar to GetList. See POG Essentials)

CODE EXAMPLES:

To retrieve a list of books associated with an author, simply do the following:

```
$author = new Author();                    //create a parent object
$author -> name = 'Michael Critchton';
$author -> Save();

$book = new Book();                        //create a child object
$book -> title = "Jurassic Park";
$book -> SetAuthor($author);               //associate child object to parent and save
$book -> Save();

$book = new Book();                        //create a second child object
$book -> title = "State of Fear";
$author -> AddBook($book);                 //associate 2nd child to parent object

$bookList = $author -> GetBookList();      //returns a list of children

foreach($bookList as $book)
{
      echo $book -> title;
}
```

The code sample above will print

Jurassic Park

State of Fear

PARENT -> SET {CHILDREN} LIST

The Set {Children} List() Relations method allows you to Set the array of child objects associated with the parent object.

CODE EXAMPLES:

To set a list of books associated with an author , simply do the following:

```
$author = new Author();                    //create a parent object
$author -> name = 'Michael Critchton';
$author -> Save();

$book1 = new Book();                       //create a child object
$book1 -> title = "Jurassic Park";
$book1 -> Save();

$book2 = new Book();                       //create a second child object
$book2 -> title = "State of Fear";

$bookList = array();                  //add book1 and book2 to booklist
$bookList[] = $book1;
$bookList[] = $book2;

$author -> SetBookList($bookList);
```

Note: Set {Children} List essentially allows you to remove any relationship between a parent and all children objects without having to delete either the parent or the children. Previous children will become orphans.

SIBLING -> ADD {SIBLING}

The Add {Sibling} Relations method allows you to associate a sibling object to another sibling object. The 2nd sibling object is added to a private sibling Array inside the 1st sibling object. The sibling object is not saved to the database until Save() is called.

CODE EXAMPLES:

Let's consider the example of Magazines and Subscribers. A Magazine has many subscribers and each subscriber can subscribe to 1 or more magazines. To prevent circular dependencies, A siblings cannot add each other successively within the same code block.

For example:

```
$silbing1 -> AddSibling2($sibling2)
```

```
$sibling2 -> AddSibling1($sibling1) (NOT ALLOWED IF sibling 1 already references sibling 2)
```

To Add a Subscriber object to a Magazine object, simply do the following:

```
$subscriber = new Subscriber();
$subscriber -> name = "John";
$magazine = new Magazine();
$magazine -> name = 'Newsweek';
$magazine -> AddSubscriber($subscriber);
```

SIBLING -> SET {SIBLING} LIST

The Set {Sibling} List() Relations method allows you to Set the array of sibling objects.

CODE EXAMPLES:

To set a list of subscribers associated with a magazine, simply do the following:

```
$magazine = new Magazine();
$magazine -> name = 'Popular Mechanics';
$magazine -> Save();

$subscriber1 = new Subscriber();
$subscriber1 -> name = "John";
$subscriber1 -> Save();

$subscriber2 = new Book();
$subscriber2 -> name = "Mary";

$subscriberList = array();
$subscriberList [] = $subscriber1;
$subscriberList [] = $subscriber2;

$magazine -> SetSubscriberList($subscriberList);
```

Note: Set {SIBLING} List essentially allows you to remove any relationship between 2 sibling objects without having to either one of them.

SIBLING -> GET {SIBLING} LIST

The Get {Sibling} List Relations method allows you to retrieve an array of sibling objects associated with the current object. The list of sibling objects includes sibling objects that have already been saved to the database as well as those that haven't

Get {Sibling} List returns an array of sibling objects.

Get {Sib.ing}List can take an optional conditions array (similar to GetList. See POG Essentials)

CODE EXAMPLES:

To retrieve a list of magazines associated with a subscriber , simply do the following:

```
$magazine = new Magazine();
$magazine -> title = "Newsweek";

$subscriber = new Subscriber();
$subscriber -> name = 'John';
$subscriber -> AddMagazine($magazine);
$subscriber -> Save();



$magazine2 = new Magazine();
$magazine2 -> title = "Popular Mechanics";
$subscriber -> AddMagazine($book);

$magazineList = $subscriber -> GetMagazineList();

foreach($magazineList as $magazine)
{
      echo $magazine -> title;
}
```

The code sample above will print

Newsweek

Popular Mechanics

OBJECT -> SAVE($DEEP)

When an object is identified as "Parent" or "Sibling", POG generates the object with a slightly different Save CRUD method. The modified Save method is known as Save($DEEP). The Save($DEEP) method extends the functionality of the traditional SAVE method by allowing you to recursively Save all other objects (children or siblings) associated with the current object. By default, all objects are saved deep, i.e. when the $deep parameter is not specified.

Save($deep) returns the object Id of the inserted/updated object.

CODE EXAMPLES:

To Save an object 'deep', simply do the following:

```
$book = new Book();                 //create a child object
$book -> title = "State of Fear";
$author = new Author();             //create a parent object
$author -> name = 'Michael Critchton';
$author -> AddBook($book);

$author -> Save (true);             //Saves both Author and Book objects
```

Note: Save(true) is the same as Save():

To Save an object 'shallow', simply do the following:

```
$book = new Book();                 //create a child object
$book -> title = "State of Fear";
$author = new Author();             //create a parent object
$author -> name = 'Michael Critchton';
$author -> AddBook($book);

$author -> Save (false);            //Saves only the Author object
```

Note: Save(false) is different from Save():

OBJECT -> DELETE($DEEP)

When an object is identified as "Parent" or "Sibling", POG generates the object with a slightly different Delete CRUD method. The modified Delete method is known as Delete($DEEP). The Delete($DEEP) method extends the functionality of the traditional DELETE method by allowing you to recursively Delete all objects (children or siblings) associated with the current object. By default, objects are ***NOT*** *deleted deep if the $deep* parameter is not specified.

CODE EXAMPLES:

To Delete an object 'deep', simply do the following:

```
$book = new Book();                 //create a child object
$book -> title = "State of Fear";
$author = new Author();             //create a parent object
$author -> name = 'Michael Critchton';
$author -> AddBook($book);

$author -> Delete (true);           //deletes both Author and Book
```

Note: Delete(true) different from Delete():

To Delete an object 'shallow', simply do the following:

```
$book = new Book();                 //create a child object
$book -> title = "State of Fear";
$author = new Author();             //create a parent object
$author -> name = 'Michael Critchton';
$author -> AddBook($book);

$author -> Delete (false);          //Deletes only Author object
```

Note: Delete(false) is the same as Delete():

OBJECT -> DELETELIST($DEEP)

When an object is identified as "Parent" or "Sibling", POG generates the object with a slightly different DeleteList CRUD method. The modified DeleteList method is known as DeleteList($DEEP). The DeleteList($DEEP) method extends the functionality of the traditional DELETELIST method by allowing you to recursively Delete all objects (children or siblings) associated with the list of objects being deleted. By default, objects are ***NOT*** *deleted deep if the $deep* parameter is not specified.

DeleteList also takes a conditions array(similar to GetList. See POG Essentials)

CODE EXAMPLES:

To Delete a list of objects 'deep', simply do the following:

```
$author = new Author();
$author->DeleteList(array(array("name", "=", "Michael Critchton")), true);
```

The above statement will delete all authors named "Michael Critchton" and all books written by the author.

WE LOVE FEEDBACK

Don't be shy, come and talk to us on the POG Google Group:
http://groups.google.com/group/Php-Object-Generator

Subscribe to our RSS feed to obtain the latest news:
http://www.phpobjectgenerator.com/plog/rss/

Send us a hello through email:
pogguys@phpobjectgenerator.com