# The Sky is the Limit

**If you need more than a page and post site structure, or of you need to convert a data-intensive legacy site that requires significant database connectivity — WordPress can still be your tool of choice... but with a twist.**

**Nick Batik, Web Developer, Pleiades WebCenter**

## An introduction to Pods

PODS is a WordPress plugin that functions as a CMS framework for creating and managing your own content types.

Before I get too far along, I would like to take a moment and speak to the new WordPress users, designers, and non-technical developers. Much of what I am going to cover here may seem overwhelming or frightening. Don't panic. You're not required to do any of this. I mostly want you to understand that if you have a need within your organization, or you encounter a client with complex, inter-related data, that WordPress is still a good, viable solution, but you will need to find a PODS developer such as myself or local Austin PODS guru Jeff Bernier.

If you are a technical developer, but new to WordPress and plugins, PODS is probably not the best place to start, either.

PODS is similar to Custom Post Types in as much as it lets you create special-purpose tabs in your admin sidebar with custom content fields. Unlike CPTs, though, the PODS UI makes it easy to add and customize content types with a simple, intuitive drag-and-drop interface.

Additionally, each custom content field can have input helpers, data validation, and display helpers. Each custom data type (called a POD) can be linked to other fields in other PODS to form complex data relationships (called PICK fields).

You can specify the data type of each POD field. Your options include a single line of text, paragraph text (which uses the TinyMCE editor similar to posts and pages), code text (which lets you paste in anything), date format, number format, and even file upload.

Relationships can be bi-directional, so any change made to the specified PICK field in one POD will appear in the corresponding PICK field of the related POD.

Like WordPress, PODS is infinitely expandable.

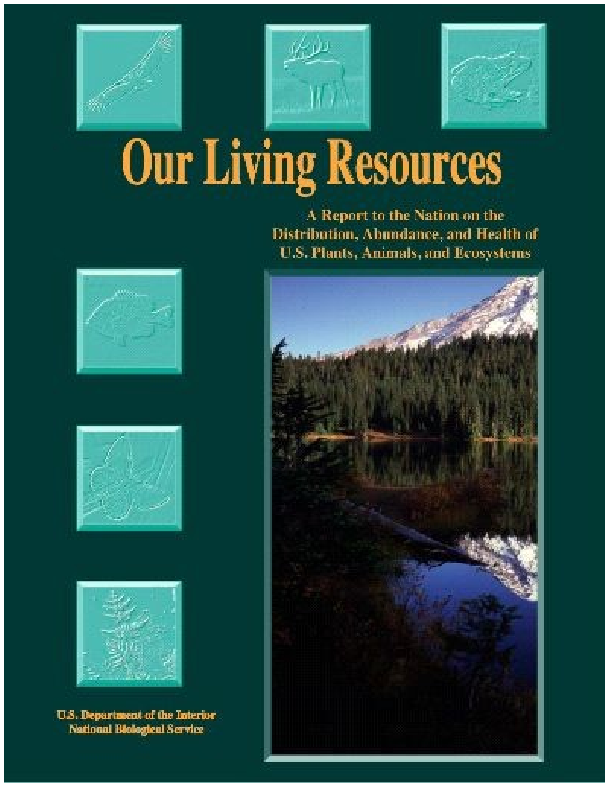
PODS is designed for complex content types with multiple relationships. With PODS developers can successfully deal with very complex ways of relating data, formatting, saving and manipulating it, and showing the information that cannot be done in native WordPress.

Enough theory, let's look at a real-life example:

**What Happens when an *old* client wants a *new* web site?**

**U.S. Bureau of Land Management**

The original BLM site was spun-off from my primary design project, the award winning print publications, “Our Living Resources” which was the first in a series of related books.


Our Living Resources
A Report to the Nation on the Distribution, Abundance, and Health of U.S. Plants, Animals, and Ecosystems
U.S. Department of the Interior
National Biological Service

My challenge was the very magnitude of the task. The client agency required the site to contain the full text of the publication. We decided to start with Our Living Resources, both because it was the first in the series, and because it was the smallest – 530 pages, consisting of 196 separate articles, 402 authors and contributors, 617 charts, graphs, photos, and tables, 2,037 bibliographic references, and a professionally indexed cross-reference of latin and common species names consisting of 4,139 entries that had to be linked to the articles.

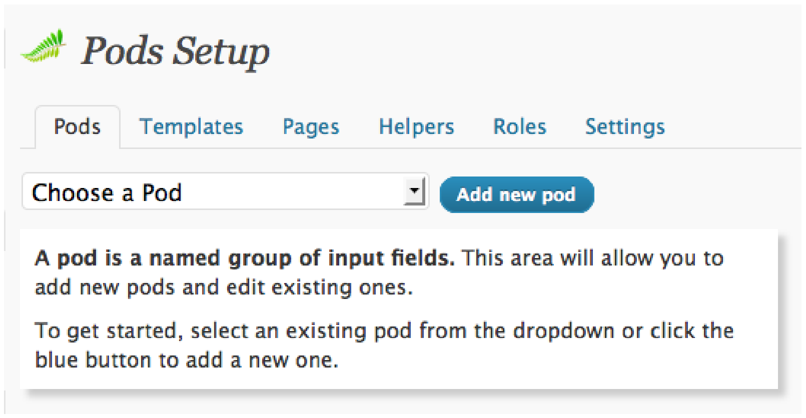
I felt this would be a perfect demonstration project for WordPress and PODS.

**Where to Begin?**

PODS can import data as a .csv (comma separated values), or if there are multiple PICK references, the data must be formatted as PHP array statements.

The book was formatted in QuarkXpress, and the content exported from Quark using the BeyondPress plugin. Because data within a book is not generally organized for direct-access linking in a database, in preparing the data to import, I decided to include as many relational fields as possible in the hope that the various content of the book could be imported into the PODS and be related through enough different PICK fields, that on any given page all the necessary bits and pieces could be assembled into a complete article. The related fields included the article title, article accession number, author name(s), page number within the book, and individual line numbers within the article text files, among others. The steps needed to prepare the data for import go well beyond the scope of this presentation.

## Creating PODS



**A pod is a named group of input fields.** This area will allow you to add new pods and edit existing ones.

To get started, select an existing pod from the dropdown or click the blue button to add a new one.

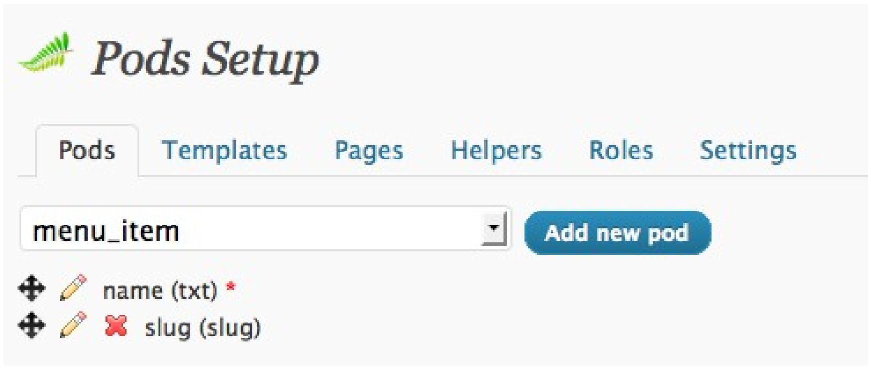
The PODS setup interface is really quite simple. You begin by adding a new POD.

My first issue was creating a menu navigation that was as close as possible to the Table of Contents in the book. To do this, I created three PODS:

| | |
|---|---|
| Menu Item | - Created the high-order categories. |
| Section | - Created the major categories within each Menu Item |
| Articles | - For the name and relationship data of each article within each Section. |

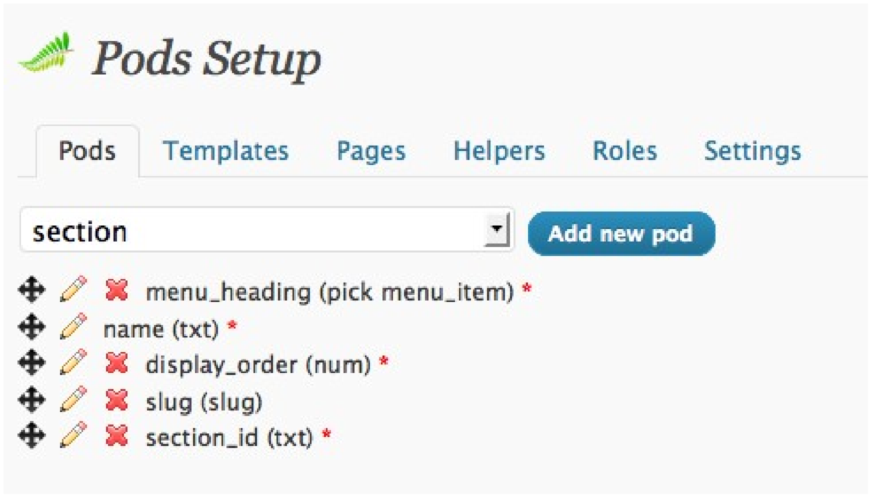
The menu items were each associated with a page, and the page inserted into a WordPress Custom Menu. I could have created all of the navigation through PODS, but because I wanted to include general information pages, and links to the future sites of the next publications to convert, I opted to use WordPress for my top navigation and PODS for everything else.

The first three PODS I created looked like this:



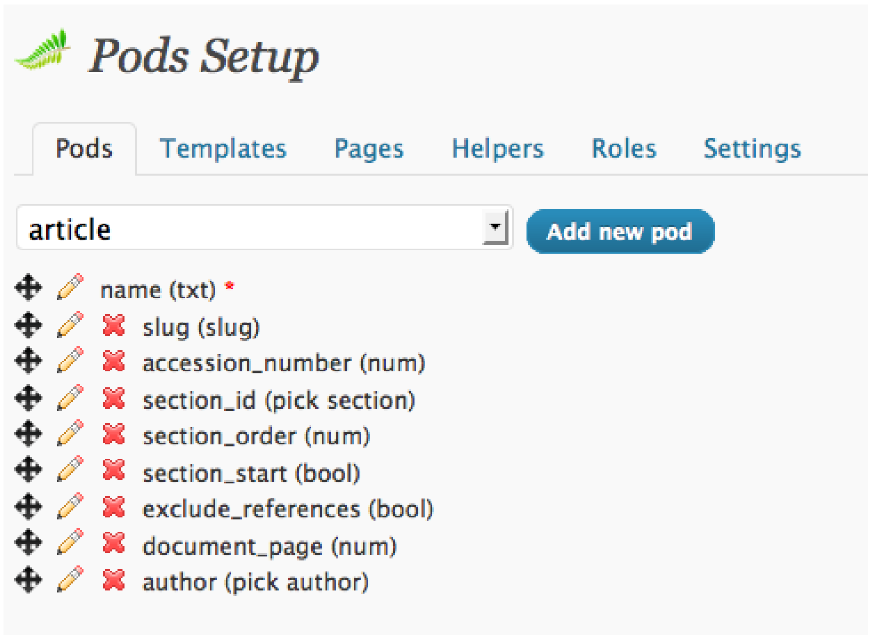
menu_item

name (txt) *
slug (slug)



section

menu_heading (pick menu_item) *
name (txt) *
display_order (num) *
slug (slug)
section_id (txt) *



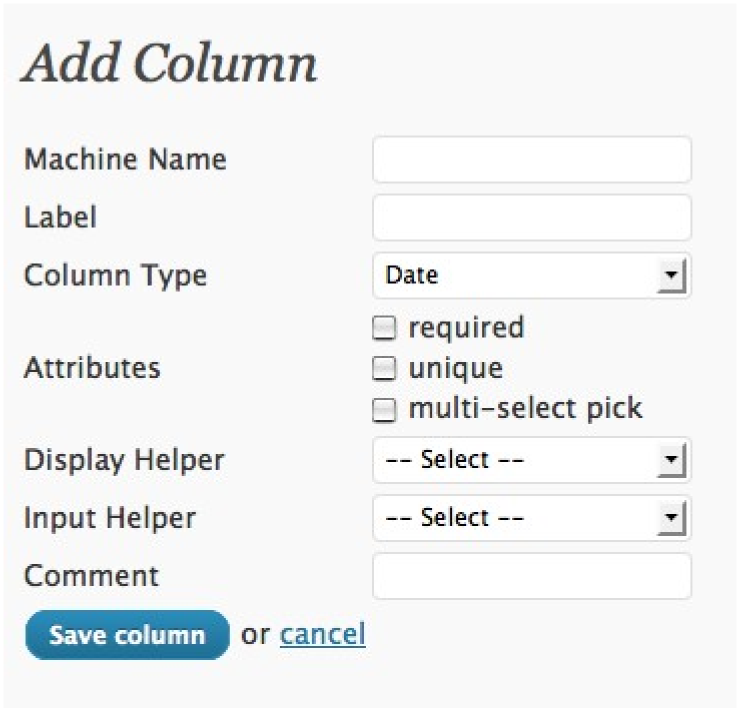
article

name (txt) *
slug (slug)
accession_number (num)
section_id (pick section)
section_order (num)
section_start (bool)
exclude_references (bool)
document_page (num)
author (pick author)

You will note that section has “menu heading” which is a PICK field relationship to “menu_item”, and article has “section_id” which is a PICK field relationship to “section”.

I will show how that works in a few minutes when we get into the code.

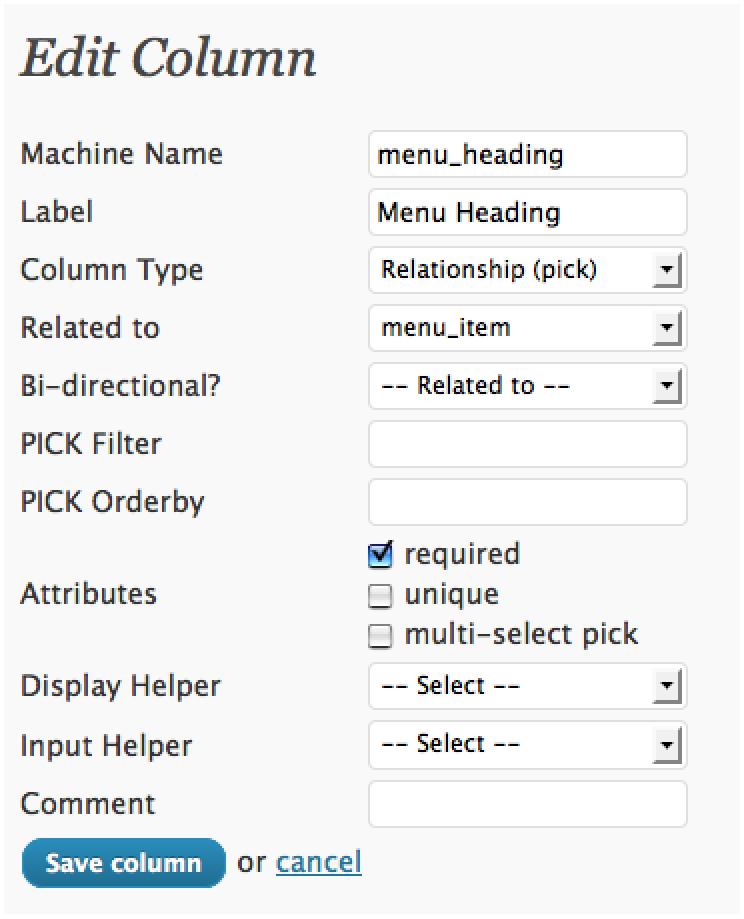
Creating fields in PODS is very easy using the PODS UI. When you create or edit a POD it displays an edit box on the page that looks like this:



## Add Column

|  |  |
|---|---|
| Machine Name |  |
| Label |  |
| Column Type | Date |
| Attributes | ☐ required<br>☐ unique<br>☐ multi-select pick |
| Display Helper | -- Select -- |
| Input Helper | -- Select -- |
| Comment |  |

Each field is entered and edited with these options. Choosing a Column Type of PICK offers a few more options that let you relate the field to a WordPress page, post, taxonomy, user, or another POD. Here we see that menu_heading in the Sections POD is related to the Menu Items POD:



## Edit Column

| | |
|---|---|
| Machine Name | menu_heading |
| Label | Menu Heading |
| Column Type | Relationship (pick) |
| Related to | menu_item |
| Bi-directional? | -- Related to -- |
| PICK Filter | |
| PICK Orderby | |
| Attributes | [✓ required] unique multi-select pick |
| Display Helper | -- Select -- |
| Input Helper | -- Select -- |
| Comment | |

I then created PODS for the article text, authors, images, and references. The cleaned and organized content was then imported into each POD.

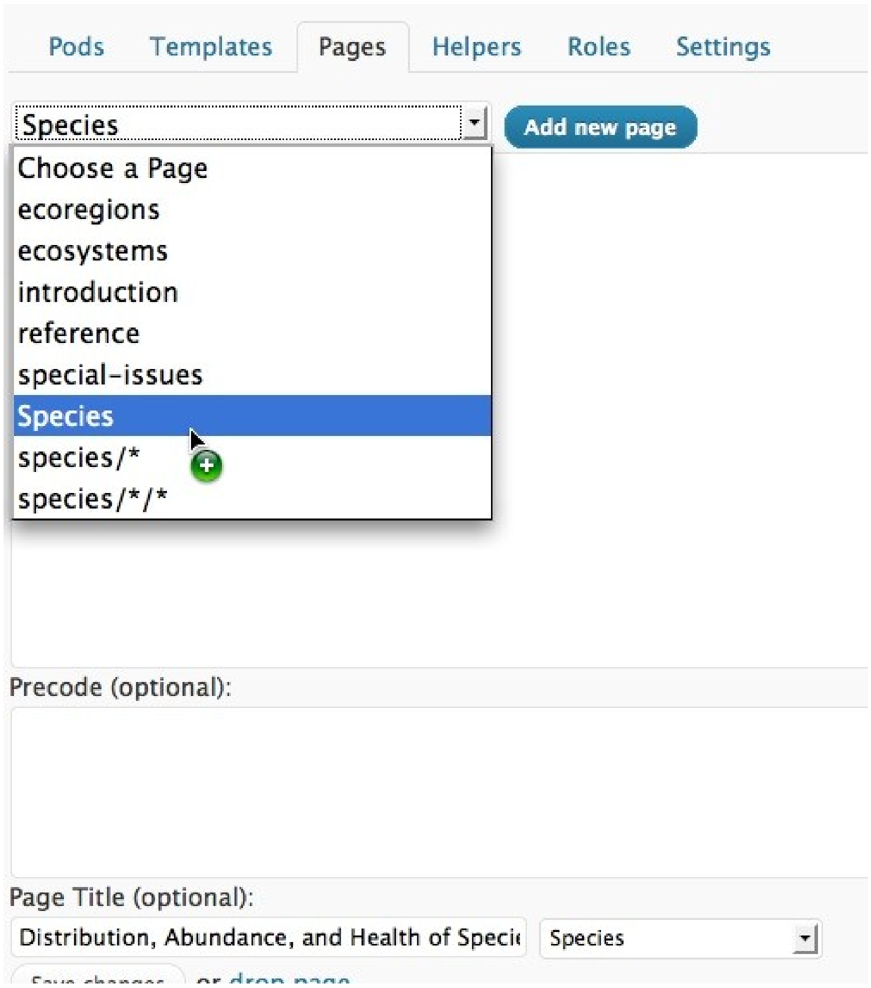
The next step is getting content to display.

PODS has a built-in pages and templates facility, but I prefer to work within my theme framework. PODS handles that equally well.

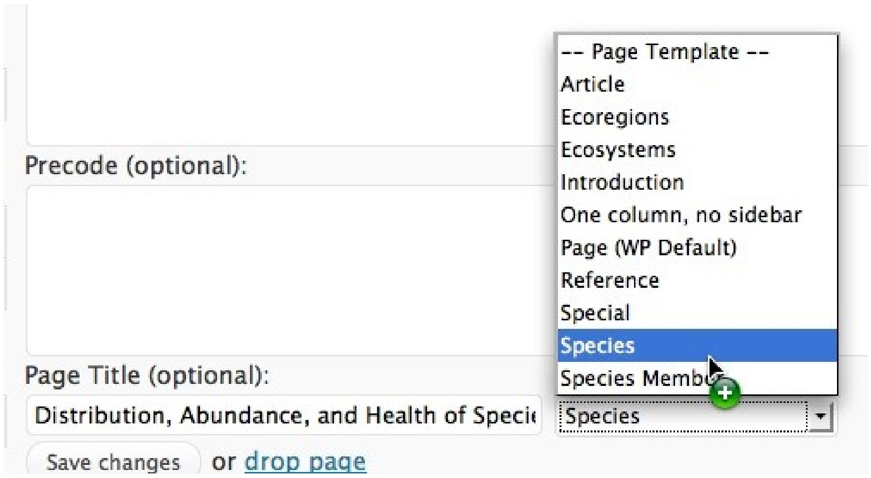
Before we proceed much further, it is helpful to understand that PODS page display is trigger by the URL. Because I have different content I want to display from the menu structure, I needed corresponding pages. The final result needed to look like this:

| | |
|---|---|
| domainname.com/**Species**/ | - displays the list of sections within species |
| domainname.com/Species/* | - when a section is selected, displays the list of articles |
| domainname.com/Species/*/* | - when an article in section is selected, displays the text. |

Not surprisingly, the setup in PODS looks just like this:



Then it was just a matter of associating each page with a template page I had added to my theme folder:



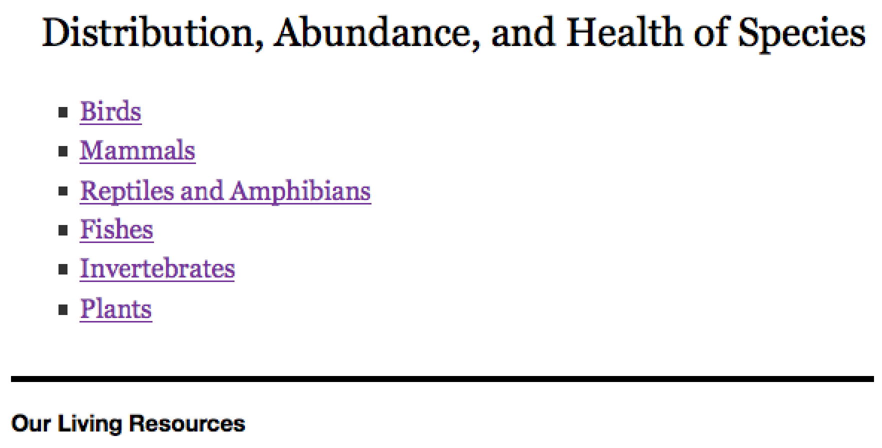
Now let's take a look at the code within each template page. The first page, Species, just needs to display the list of sections within the Species menu item.

```
<?php
```

```
          $species = new Pod('section');

          $params = array( 'select'=>'*','orderby'=>'t.display_order',
'where'=>'menu_heading.name="Species"' );
          $species->findRecords($params);

          $total_entries = $species->getTotalRows();
        ?>

        <?php if( $total_entries > 0 ) : ?>
          <ul>
            <?php while ( $species->fetchRecord() ) : ?>

              <?php
                // set our variables
                $member_name      = $species->get_field('name');
                $member_slug      = $species->get_field('slug');
              ?>

              <li>
                <a href="<?php echo get_permalink(); ?>
                  <?php echo $member_slug; ?>/">
                  <?php echo $member_name; ?>
                </a>
              </li>

            <?php endwhile ?>
          </ul>
        <?php endif ?>
```

Here we see the code to get the content from PODS, and to display it on the page. I could have put the selection parameters inside of the “findRecords” function call, but I broke them out separately so they would be easier to understand. The parameters specify selecting all of the record content (you can select only specific fields if you are expecting lots of data), sorting the records in the order set by the “display_order” field. Because there is a PICK field relationship with menu_heading, the “where” clause can limit the search to just those records whose related record in menu_heading has a name field containing “Species”.

Because PODS does not save its information in the the WordPress pages and posts database table, the normal WordPress Loop will not work, so you have to create your own. To do this, I first test to see if the search returned any records, then using a PHP while loop, retrieve each POD record with the “fetchRecord” function. I copy the content out of two fields in the array – name and slug. Slug is a required field in PODS, and its content is the auto-generated URL. Next I format the the name and slug with list tags. We now have a page display of the list of species generated entirely from PODS.



## Distribution, Abundance, and Health of Species

- Birds
- Mammals
- Reptiles and Amphibians
- Fishes
- Invertebrates
- Plants

**Our Living Resources**

Now let's look at what happens when we click on one of those links:

```
        <?php
            $found_member = false;

            global $pods;
            $member_slug  = pods_url_variable(-1);
            $member       = new Pod('section', $member_slug);

                if( !empty( $member->data ) )
                {
                $found_member = true;

                $member_slug        = $member->get_field('slug');
                $member_name        = $member->get_field('name');
                $member_section     = $member->get_field('section_id');
                }
        ?>

            <h3><?php echo $member_name; ?></h3>

        <?php
            $articles = new Pod('article');

                $params = array();

                $params['select'] = '*';
                $params['where'] = 'section_id.section_id='.'"'.$member_section.
'"';
                $params['orderby'] = 't.section_order';
                $params['limit'] = -1;

                $articles->findRecords($params);

                $total_entries = $articles->getTotalRows();
          ?>

        <?php if( $total_entries > 0 ) : ?>
          <ul>
            <?php while ( $articles->fetchRecord() ) : ?>
```

```
              <?php
                // set our variables
                $articles_name      = $articles->get_field('name');
                $articles_slug      = $articles->get_field('slug');
              ?>

              <li>
                <a href="<?php echo get_permalink(); ?><?php echo $articles_slug; ?
>/">
                  <?php echo $articles_name; ?>
                </a>
              </li>

            <?php endwhile ?>
          </ul>
        <?php endif ?>
```

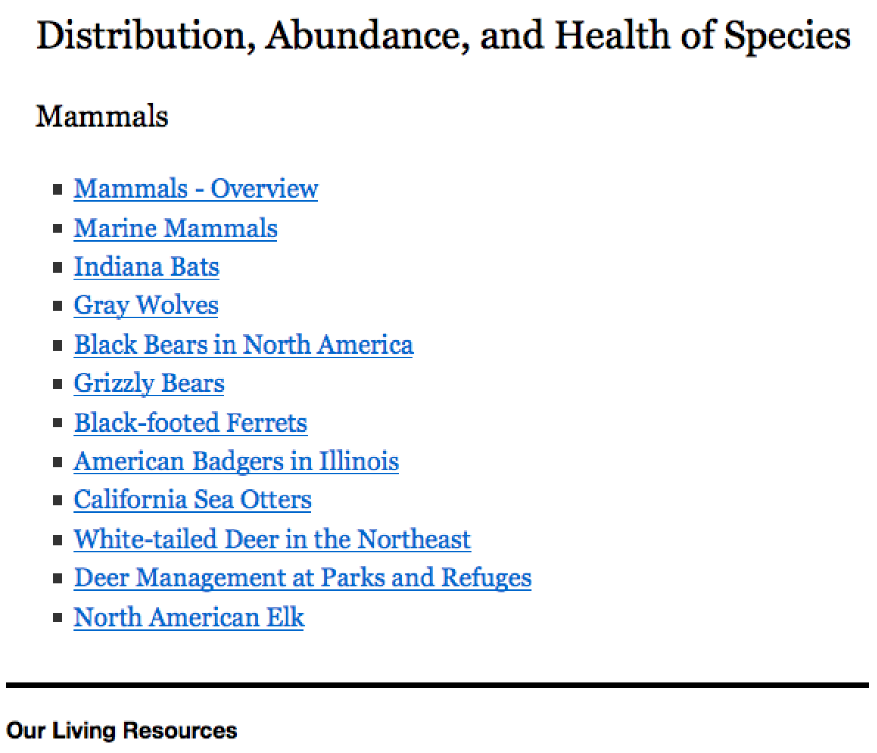
The first little odd bit of code...

```
      $member_slug  = pods_url_variable(-1);
      $member       = new Pod('section', $member_slug);
```

...tells PODS to return the information about my current location. Because there are no static pages, and no fixed links between them, I don't know which of the available links the user clicked on – Birds, Mammals, Reptiles, etc.

First I construct a search to find the records in the “articles” POD where the “section_id” field in the related “section” POD contains whatever content is in the “member_section” field. In other words, If the user clicked on “Mammals”, the pods_url_variable(-1) returns the slug for Mammals, then the search in articles looks for every article with a Section ID of “Mammals”.

The next code, similar to the previous page, loops through the found list and displays every article found in that section, sorted by the “section_order” field.



## Distribution, Abundance, and Health of Species

### Mammals

- Mammals - Overview
- Marine Mammals
- Indiana Bats
- Gray Wolves
- Black Bears in North America
- Grizzly Bears
- Black-footed Ferrets
- American Badgers in Illinois
- California Sea Otters
- White-tailed Deer in the Northeast
- Deer Management at Parks and Refuges
- North American Elk

**Our Living Resources**

This final link takes the user to the page content:

```
<?php
     $found_article = false;

     global $pods;
     $article_slug  = pods_url_variable(-1);
     $article       = new Pod('article', $article_slug);

          if( !empty( $article->data ) )
          {
          $found_article = true;

          $article_slug       = $article->get_field('slug');
          $article_name       = $article->get_field('name');
          $article_number     = $article->get_field('accession_number');
          }
  ?>

<?php
     $text = new Pod('article_text');

     $params = array();
     $params['select'] = '*';
     $params['where'] = 't.name=' . '"' . intval($article_number) . '"';
     $params['orderby'] = 't.line_number';
     $params['limit'] = -1;

     $text->findRecords($params);

     $total_entries = $text->getTotalRows();
?>
```

```
<?php if( $total_entries > 0 ) : ?>

      <?php while ( $text->fetchRecord() ) : ?>

        <?php
          // set our variables
          $text_reference   = $text->get_field('paragraph_reference');
          $text_content     = $text->get_field('paragraph_text');
?>

        <p><?php echo $text_content; ?></p>

      <?php endwhile ?>

  <?php endif ?>
```

Again, I use the pods_url_variable to tell me what was clicked for me to be here. From that I get the article accession number. Most of the display content (text, authors, references, images) are keyed off of this number.

With that, I find all records in the “article_text” POD with a matching accession number, sorted by line number.

The fetchRecord loop then tags and displays each paragraph.

This is a much-simplified version of the actual final page. There I include a corresponding lookup in the “authors”, “images”, and “references” PODS. The list of authors is displayed at the top of each page, with links to author detail pages; the images are positioned next to the paragraph where they are first referenced, along with an AJAXed lightbox display of enlarged versions; the bibliographic references have a bi-directional link between the reference in text and the reference footnote.

Hopefully this little demonstration gave you a taste for the possibilities of WordPress PODS. You saw, here, how to use PODS to get information about your location within a dynamically-generated hierarchy; how to use data from one POD field to find related records in other PODS; and how to combine data from different fields in different PODS to display your content on a page.

There are far more capabilities than we can cover here. You can connect PODS with WordPress pages, posts, authors and taxonomies; assign roles for access to PODS and POD options; and use both WordPress and Theme functions for manipulating content.

If you have ever wondered if there are any limits to what can be done with WordPress, when it is combined with PODS, the sky is the limit!

**Contact Information:**

Nick Batik

Pleiades WebCenter

nbatik@pleiadeswebcenter.com

512.879.8658