Dataface API Essentials

Steve Hannah
Web Lite Solutions Corp.
Steve@weblite.ca

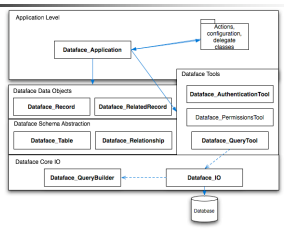
What is the Dataface API?

- API = Application Programming Interface
- It is the foundation of Dataface.
- A group of classes and functions to interact with Dataface and work with Data at a higher level of abstraction

What can I do with the API?

- Add, edit, delete, find records.
- Create forms and lists of results.
- Customize your application to suit your needs exactly.

Framework Architecture



Dataface_Application: Central class that handles all requests and allows different parts of application to communicate with each other.

Dataface_Table : Objects of this class represent tables from the database. Uses information from the database and the fields.ini file.

Dataface_Relationship: Objects of this class represent relationships between records. Uses information from the relationships.ini file.

Dataface_QueryTool : Can be used to obtain results from the database based on user defined query parameters.

Dataface_QueryBuilder : Converts Dataface query descriptions to SQL queries that can be fed directly to the Database.

Dataface_IO : Performs saving, reading, and deleting operations.

Dataface_PermissionsTool : Abstract interface for dealing with permissions. Works closely with the permissions.ini file.

Dataface_AuthenticationTool : Tool to handle user authentication.

Dataface_Record : Data object to represent a single record from the database. Can be used to manipulate, save, edit record.

Dataface_RelatedRecord : Data object to represent a single related record in the database. A related record is different than a Record in that it knowns which parent record owns it and may contain fields from multiple tables.

The Public API

- dataface-public-api.php contains convenience wrapper functions to access many common functions of the framework.
- Functions to: get records, build forms, display templates, and more.

Initializing the API

- You can use the API after the script has called df_init() once.
- The API is automatically initialized if you are writing code for an action or delegate class.
- If writing separate script, you need to initialize:

```
require_once '/path.to.dataface/dataface-public-api.php';
df_init(__FILE__, '/url.to.dataface');
```


Loading records

- Many ways to load records:
  1. Dataface_Application::getRecord()
  2. df_get_record()
  3. df_query()
  4. mysql_query()

Dataface_Application:: getRecord()

```
<?php
$app =& Dataface_Application::getInstance();
        // Obtains a reference to the Dataface_Application object

$record =& $app->getRecord();
        // A Dataface_Record object containing the record that was
        // found based on
        // the GET query parameters.
```

E.g. If the URL entered was *index.php?-table=People&FirstName=Bob* then $record would contain the first result matching the query.

df_get_record()

- Supply your own custom query to this function to obtain the desired record.

```
$query = array('FirstName'=>'Bob');
$record =& df_get_record('People', $query);
```

- The above would obtain the first record in the People table matching FirstName=Bob or Bobby, etc..

df_query()

- Wrapper for mysql_query with some perks!
- If you are using the multilingual extension (or ever will), this will automatically translate the query to return results in the proper language.

```
$query = array('FirstName'=>'Bob');
$record =& df_get_record('People', $query);

$res = df_query("select * from People where FirstName='Bob'");
if ( !$res ) trigger_error(mysql_error(df_db()), E_USER_ERROR);

if ( mysql_num_rows($res) > 0 ) $record = new Dataface_Record('People', mysql_fetch_assoc($res));
else trigger_error("No records matched your query", E_USER_ERROR);
```


mysql_query()

- Old faithful… performs a query directly against the database - not part of Dataface API - just core PHP.

```
$query = array('FirstName'=>'Bob');
$record =& df_get_record('People', $query);

$res = mysql_query("select * from People where FirstName='Bob'", df_db());
if ( !$res ) trigger_error(mysql_error(df_db()), E_USER_ERROR);

if ( mysql_num_rows($res) > 0 ) $record = new Dataface_Record('People', mysql_fetch_assoc($res));
else trigger_error("No records matched your query", E_USER_ERROR);
```


Dataface_Record

- The df_get_record and getRecord() methods return Dataface_Record objects.
- How do you work with them? Full API docs for class at
  http://dataface.weblite.ca/Dataface_Record

Useful methods of Dataface_Record include:

getValue() : Returns the value of a particular column in the record.

val() : alias of getValue()

strval() : Gets the value of a column in the record as a string (useful if the column is a non-string type (e.g. date) and you need to display it as a string).

display() : Gets value of a column prepared to display to the screen. This honours permissions so if the current user does not have permission to see a column, this will return 'NO ACCESS'

q() : Alias of display()

qq(): Similar to q() except this encodes all html entities in output to make it more friendly to browser output.

htmlValue() : Formats the value as HTML. This is handy for Blob fields as it will output the <a> tag or <img> tag to display or download the data stored.

preview() : A truncated value for a field. Good for list view so that long values don't ruin display.

Exercise I

- Create a stand-alone script that provides the user with a search form to search on 2 or 3 columns of a database, and output the details of the first record found.
- Make 4 versions of this script, 1 for each method discussed in previous slides.

Record Iterators

- *Iterators* allow you to step through lists.
- **2 methods:**
  hasNext() - true or false - whether there is another record in the list.
  next() - obtains the next record in the list
- Some Dataface functions return iterators (e.g. *df_get_records()).*
- Alternative methods exist to return arrays (e.g. *df_get_records_array()).*

df_get_records()

- Very similar to df_get_record() except this returns an iterator to go through all results of the find.

```
$query = array('FirstName'=>'Bob');
$iterator =& df_get_records('People', $query);
while ( $iterator->hasNext() ){
        $record =& $iterator->next();
        echo "<p>Found {$record->qq('FirstName')} {$record->qq('LastName')}</p>";
        unset($record);
}
```


df_get_records_array()

- Same as df_get_records() except this returns an array instead of an iterator.

```
$query = array('FirstName'=>'Bob');
$records = df_get_records_array('People', $query);
foreach ($records as $record){
        echo "<p>Found {$record->qq('FirstName')} {$record->qq('LastName')}</p>";
}
```

Exercise 2

- Add onto your script from exercise 1 so that when the user searches he gets a list of all of the matching results. He can then click on any of the results to see details of that record.
- Create 2 versions of your script: 1 using df_get_records and the other using df_get_records_array().

Related Records

- Dataface makes it easy to work with relational data partly because of its relationship abstraction via the relationships.ini file.
- Accessing the related records of a given record programmatically is easy: Dataface_Record::getRelatedRecords()

getRelatedRecords()

- Dataface_Record allows you to find the records that are related.
- Assume the People table has a relationship called 'courses' showing the courses that the person is enrolled in.

```
$record = df_get_record('People', $query); // load our record
$courses =& $record->getRelatedRecords('courses');
foreach ($courses as $course){
        echo "Enrolled in: {$course['Name']} which starts on {$course['StartDate']}.";
}
```

Notice that getRelatedRecords() returns an array of associative arrays representing the records. I.e. each record returned is just an associative array - and not an object.

The getRelatedRecordObjects() actually wraps each record in a Dataface_RelatedRecord object so that it can be manipulated more easily.

The getRelationshipIterator() is similar to the getRelatedRecordObjects() method except that it returns an iterator instead of an array.

Filtering & Sorting Related Records

- What if you only want a subset of the related records?
- What if you want the related records sorted on a particular field?
- getRelatedRecordObjects(
  'courses', // name of the relationship
  0, // start position
  10, // Number of records to get
  "CourseNumber > 200", // search params
  "CourseNumber asc" // sort ascending by course number
  );

Setting start and limit parameters to null forces them to use default values.

Default values for start and limit are 0 and 30 respectively.

A value of 'all' for start indicates that the limit of 30 records should be ignored - I.e. all records should be returned.

Setting the search params to 0 causes it to be ignored.

```
$courses = $record->getRelatedRecords('courses');

$courseObjects = $record->getRelatedRecordObjects('courses');

foreach ($courses as $course ){

        echo "Name: {$course['Name']}";
}

foreach ($courseObjects as $course ){
        echo "Name: {$course->val('Name')}";
}
```

Exercise 3

- Expand your script from exercise 2 so that the details page now shows a list of the related records for the current record. And make it so that if the user clicks on the related records, it will take them to the details page for that record.

Modifying records

- Dataface_Record objects can be modified via the setValue() method, and saved using the save() method.

```
$record = df_get_record('People', $query); // load our record
$record->setValue('FirstName', 'John');
$record->setValue('BirthDate', '1980-12-24');
$record->save();
```


Forms

Edit Record Forms
New Record Forms
Related Record Forms

PEAR Class Libraries

- Dataface uses the PEAR class libraries extensively. (http://pear.php.net)
- All forms are based on HTML_QuickForm - a class that makes it easy to generate robust web forms.

HTML_Quickform example:

```
// Import the QuickForm class
require_once 'HTML/QuickForm.php';

// Create the QuickForm object
$form = new HTML_QuickForm('myform','POST');

// Add elements to the form
$form->addElement('text','fname','First Name');
$form->addElement('textarea','desc', 'Description');
$form->addElement('submit');

// Try to validate the form (make sure the form has been submitted etc...)
if ( $form->validate() ){
        // Process the form (calls our custom save function
        $form->process('save');

        // Forward to the success page
        header('Location: success.php');

        // Important to exit to stop execution.
        exit;
}

// Finally display the form if we get here.
$form->display();
```

The addElement() method allows you to add elements to the form (e.g. text fields, select lists, text areas, etc…).

The process() method calls our specified function (in this case 'save()') to handle what to do to process the form.

The display() method displays the form.

There is a good HTML_QuickForm tutorial at http://www.midnighthax.com/quickform.php

Form Processing Guidelines

1. In any *single* request you may *process* the form or *display* the form, but **not both.**

```
if ( $form->validate() ){
        // The form validated OK, so we process it!
        $form->process('save');

        // Now we forward to a success page to display the results.
        // We don't display anything on the current request, because
        // we have processed the form
        header('Location: success.php');

        exit;       // IMPORTANT!
}

//If we got this far, then we didn't process the form this time around.
// So we display the form
$form->display();
```

Why is this rule so important?

Example: Your form adds new records to the database. If the user presses refresh in his browser to see the page again, your script will add another new record to the database each time. This is not what we want!

Exceptions to this rule: Forms that do not modify any persistent data or state may display on the same request. E.g. a search form.

HTML_QuickForm processing

- Our example included the line:
  $form->process('save');
  which calls our custom save() function to save the form input. Our save function might look like:

```
/**
 * A custom save function to save the form's input.  This is called by
 * $form->process() after the form is validated
 */
function save($values){
        $record = new Dataface_Record('People', array());
        $record->setValue('FirstName', $values['fname']);
        $record->setValue('Description', $values['desc']);
        $record->save();

}
```

The save() function receives one parameter '$values' which is an associative array of the array values that were submitted to the form.

Default Values

HTML_QuickForm will allow you to set default values for fields.

This is handy for editing existing data.

```
$form->setDefaults( array(
        'fname'=>'Bob',
        'lname'=>'Jones',
        'desc' => 'Nothing special'
        )
);
```

Exercise 4

- Create a form using HTML_QuickForm to add new records into a particular table (at least 4 fields). Try to use the most appropriate widget for each field.
- Use the dataface API for actually saving the record. (I.e. Dataface_Record->setValue() / save()).

Exercise 5

- Modify the form in Exercise 4 so that it can be used to edit existing records also.

Dataface_QuickForm

- Dataface class that extends HTML_QuickForm
- Can generate forms based on DB schema
- Form for single table only.

Using Dataface_QuickForm

```
$form = df_create_new_record_form('People');
if ( $form->validate() ){
        // The form validated OK, so we process it!
        $form->process(array(&$form,'save'), true);

        // Now we forward to a success page to display the results.
        // We don't display anything on the current request, because
        // we have processed the form
        header('Location: success.php');

        exit;       // IMPORTANT!
}

//If we got this far, then we didn't process the form this time around.
// So we display the form
$form->display();
```

Notice that $form->process() takes an array as a parameter. This is a PHP callback.

http://us2.php.net/manual/en/function.call-user-func.php

It tells Dataface to calll the save() method on the $form Dataface_QuickForm object.

Dataface_QuickForm results

- What does that code produce?
- An HTML form to create a new record on the People table.
- It obeys Dataface configuration rules (e.g. from fields.ini file).

Edit record forms

- Similarly you can call df_create_edit_record_form() to produce a form to edit a record:

```
$record = df_get_record('People', array('UserID'=>10));
$form = df_create_edit_record_form($record);
        // $form to edit the $record with UserID 10
```


Partial forms (specifying fields)

- Both df_create_new_record_form() and df_create_edit_record_form() take an optional 2nd parameter.
- Pass an array of the field names to include in the form.

```
$form = df_create_edit_record_form(
        $record, // The record to edit
        array('UserID','FirstName','LastName')
        );
        // This form only includes the UserID,
        // FirstName, and LastName fields
```


Related Record Forms

- Adding related records can be handled nicely by the Dataface_ShortRelatedRecord form.

```
$form = df_create_new_related_record_form(
        $record, // the parent record of the relationship
        'courses', // the name of the relationship
        );

// Or specifying only certain fields
$form = df_create_new_related_record_form(
        $record,
        'courses',
        array('CourseName','CourseNumber')
        );
```

Add Existing Related Record Form

- df_create_existing_related_record_form () can be used instead to create an existing relationship.

Alternative way to create related record

- If relationship is one-to-many (I.e. there is no join table - we just have to add to a single table), we can just use *df_create_new_record_form()* and set default values.

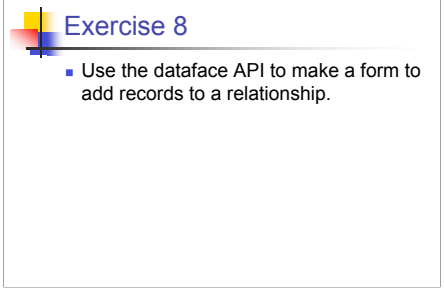
```
$form = df_create_new_record_form('courses');
$form->setDefaults(
        array('OwnerID'=>$record->val('PersonID')
        );
```


Exercise 6

- Alter your form from exercise 5 to use df_create_new_record_form() and df_create_edit_record_form()

Exercise 7

- Change your form in exercise 6 so that it only shows 2 fields from the table (say FirstName and LastName).


Exercise 8
Use the dataface API to make a form to add records to a relationship.