Database Extensions, Page Fragments and Plugins

Presenter: Roger Sprik Updated September 2014 for European PSUG Malta Conference

Released in PowerSchool 7.9

Key Documents on PowerSource:

- Database Extensions Visual Walkthrough - ID: 70768
- Database Extensions Advanced User Guide for PowerSchool 8.x - ID: 72513
- Database Extensions Best Practices - ID: 71545
- Database Extension and Custom Field Migration Frequently Asked Questions for PowerSchool 8.x - ID: 72504
- Data Dictionary Tables for PowerSchool 8.x - ID: 72521

Overview

- Database Extensions is the name for the new method for custom data entry points in PowerSchool.

- It will currently run side-by-side with the old way of doing custom data, but eventually will completely replace it. It is not required as of 8.0.

- **Before**:
    - Users go to System - Custom Fields/Screens
    - Only Student, Staff (teachers table), Course and Section fields were available to be created this way
    - Only text fields could be created
    - All data went into one giant custom field that had to be parsed
    - The only way to do one-to-many custom fields was to use virtual tables which is little known and very difficult to work with

- **After:**
    - Users go to System - Page and Data Management - Manage Database Extensions
    - Users create REAL tables and fields of various data types.
    - Migration options for legacy custom fields
    - Many, many more tables can be extended, users can create one-to-many tables and even independent tables.
    - Inherent grouping of fields for logical organization.
    - No more 999 limit!
    - All tables get some tracking fields: whocreated, whencreated, whomodified, whenmodified. It won't tell you what changed, but who and when.
    - Any extended table can be exported from and imported into with the Data Export and Import Managers.

The Old Way. *With custom fields, all custom data is stored in a single field in the core table. They are all "Text" types.*

Students Table	
Last_Name	Adair
First_Name	Brandon
DCID	2
ID	5
CUSTOM	Medical Notes: Allergic to peanuts contact_1_email: contact1@email.com contact_2_email: contact2@email.com contact_1_employer: Wal Mart contact_2_employer: Lanham Law Offices parent_notes: Brandon is only allowed to be picked up from school on Wednesdays by his father.

Database Extensions - The New Way. *Faster, real tables and fields, multiple data types.*

Students	*Core Table "Parent"*		**U_Def_Ext_Students**	*New extended table "Child"*
Last_Name	Adair			
First_Name	Brandon		*This table "relates" to the Students table*	
DCID (*Primary Key*)	2	<->	StudentsDCID (*Foreign Key*)	2
ID	5		Contact1_Email	contact1@email.com
			Contact2_Email	contact2@email.com
			Contact1_Employer	Wal Mart
			Contact2_Employer	Lanham Law Offices
			Medical_Notes	Allergic to Peanuts

Custom Field Migration

- **Core Custom Fields.** The core PowerSchool product has many legacy fields that were never "real" fields, but were created by the developers as custom fields. These are known as "Core Fields".
- **Activities** have traditionally been stored as custom fields.
- **SuccessNet** integration fields were also custom fields.
- **User Custom Fields** are the fields created by customers

- History of options to migrate core custom fields and user custom fields.
    - 7.9 First set of core custom fields can be migrated
    - 7.10 Second set of core fields can be migrated. User fields can be migrated one at a time. Activities and SuccessNet fields migrated automatically
    - 7.11 User custom fields can be migrated all at once (*but you might not want to*)

- Core Fields Migration (*See Data Dictionary for complete list*)
    - 1st 59 student core custom fields are moved to a new table called StudentCoreFields
        - ACT_Composite/Date/English/Math/Reading/Science, SAT
        - Emerg_1/2/3_Ptype/Rel
        - Guardian, Guardian_LN, Guardian_FN, Guardian_MN
        - Mo/Father_Home_Phone, Mo/Father_Day_Phone, Mo/Father_Employer
        - PrimaryLanguage, SecondaryLanguage
        - PrevStudentID, Family_Rep, Area, Dentist_Name, etc
    - 2 teacher/user core custom fields are moved to a new table called UsersCoreFields
        - DOB
        - Gender
- Core Fields 2 Migration (*See Data Dictionary for complete list*)
    - 52 more student custom fields are migrated to the StudentCoreFields table
        - Autosend fields
        - CRT_ fields
        - EC_ fields
        - IPT_ fields
        - a few others.
    - 1 course field is moved to the School_Course table
        - Alt_Course_Number
- User Custom Fields migration - Two ways:
    - **One-at-a-time**. You can migrate an existing custom field from the screen where you create a new extended field.
    - **All at once**. From System - Page and Data Management - Custom Field Data Migration.
    - Which way is best? All at once is faster, but one at a time gives you the opportunity to organize your fields into logical groups and a chance to standardize your naming convention.
- Migrated fields continue to operate the “old” way without changing any existing custom pages, exports, reports, etc. PowerSchool is programmed to act like migrated fields still exist in the “old” location, but there is a performance penalty. You will eventually want to use them with the “new” rules.
- New fields created AFTER migration must use the new naming and querying rules
- Please note the issues with migration of fields that contain more than 4000 characters.

Custom Field Migration is managed from:

System > Page and Data Management > Custom Field Data Migration

Organizational Concepts

It's important to understand the concept of an Extension Group vs an Extension table vs an Extension Field. A "Group" is a high level organization structure. It's a way to keep your tables and fields organized by function, such one group for managing Families, and another group for managing College Applications. You can also choose to have one group for ALL your custom fields. If you plan on sharing database extensions, you should consider multiple groups. You should choose your names for groups carefully, keeping in mind that the culture of sharing in the PowerSchool community may lead to potential naming conflicts.

Remember the Flow

Core Table > Extension Group(s) > Extension Table(s) > Extension Field(s)

Cautionary Notes:

- You cannot delete, change or rename extensions once created. So plan carefully and think about your users and customizations when naming them.
- For each extension group's core table, you can only have one 1:1 extended table, but you can have multiple 1:many tables.
- When you install someone else's plugin and it contains extensions, they will be added to your system and the tables and fields cannot be removed, only abandoned.
- When a plugin containing a database extension is installed or modified, before using you must restart PowerSchool/PowerTeacher and ReportWorks services.
- When you disable a plugin, it won't be served from CPM, but there is no way to tell in CPM if a custom page is associated with a plugin.
- ReportWorks will show extended fields, but must be restarted first.
- It's not obvious, but you can have more than one core table per extension group… just select the core table and then pick an existing extension group in Step 2.
- The Database Extension wizard suggests group names like U_Students_Extension and table names like U_DEF_EXT_STUDENTS --- you DO NOT have to use those names, you might want to use names that make sense for your application.

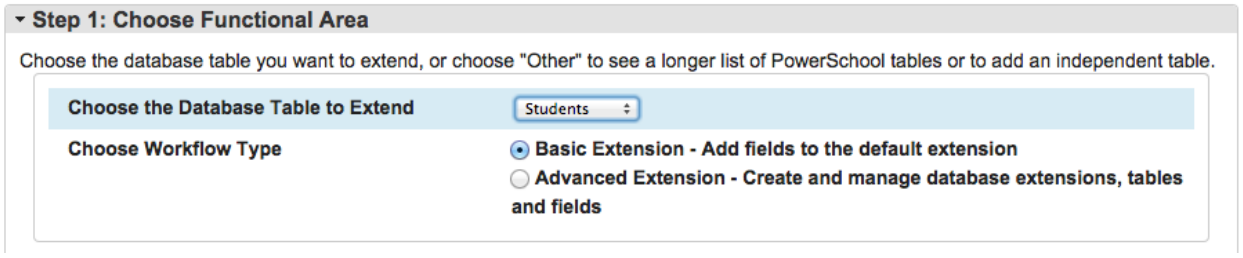
Creating Database Extensions

There are more complete instructions in the PowerSource documents, these are the basics.

System > Page and Data Management > Manage Database Extensions

There are 4 steps to creating a database extension

Step 1 - <u>Choose the core table that will be extended.</u> You can choose Basic or Advanced Extension. The Basic option will automatically select default options for steps 2 and 3, bringing the user directly to step 4.

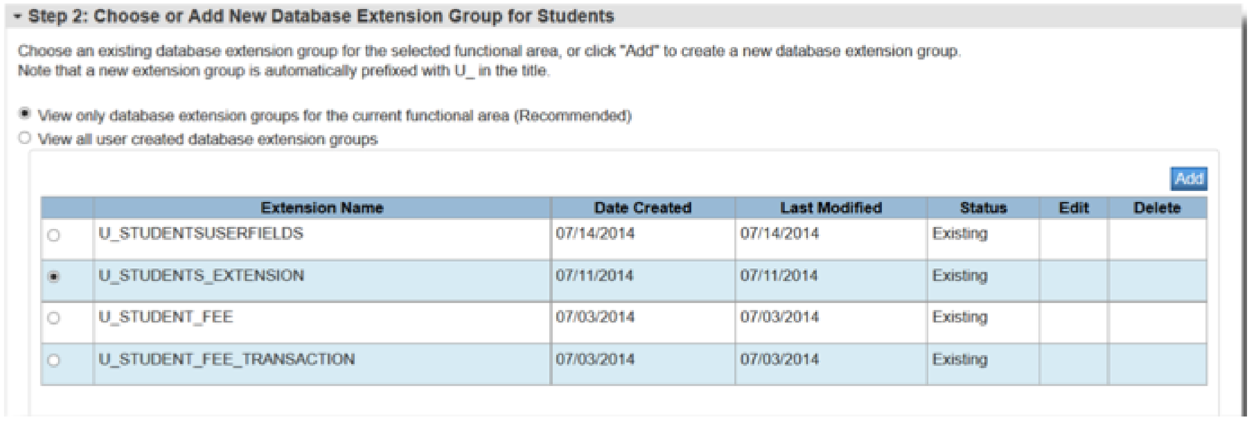

▾ Step 1: Choose Functional Area

Choose the database table you want to extend, or choose "Other" to see a longer list of PowerSchool tables or to add an independent table.

Choose the Database Table to Extend	Students
Choose Workflow Type	[✓ Basic Extension - Add fields to the default extension] Advanced Extension - Create and manage database extensions, tables and fields

Step 2 - Choose or add a database extension group. The database extension group is simply a name given to a group of database extended tables. Many users choose to use a single extension group for all custom fields. Others choose to create different groups to compartmentalize data (i.e. demographic data and contact data).

U_Students_Extension is the default extension group when extending the Students table. This is the option that will be used if the Basic Extension type is selected in step 1.



▾ Step 2: Choose or Add New Database Extension Group for Students

Choose an existing database extension group for the selected functional area, or click "Add" to create a new database extension group.
Note that a new extension group is automatically prefixed with U_ in the title.

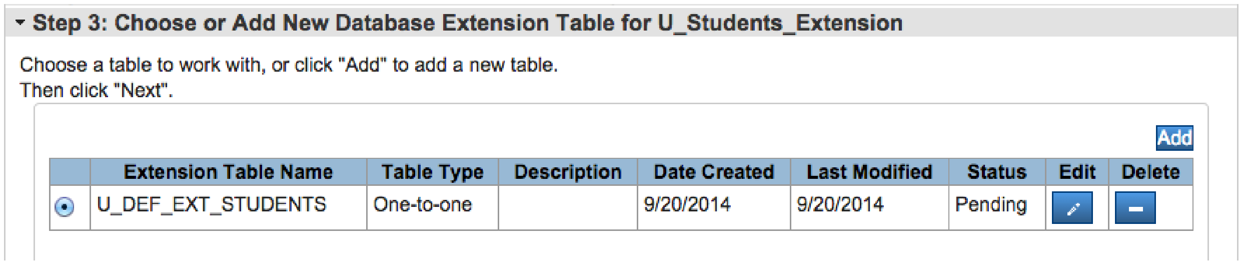
[✓ View only database extension groups for the current functional area (Recommended)]
View all user created database extension groups

	Extension Name	Date Created	Last Modified	Status	Edit	Delete
○	U_STUDENTSUSERFIELDS	07/14/2014	07/14/2014	Existing		
●	U_STUDENTS_EXTENSION	07/11/2014	07/11/2014	Existing		
○	U_STUDENT_FEE	07/03/2014	07/03/2014	Existing		
○	U_STUDENT_FEE_TRANSACTION	07/03/2014	07/03/2014	Existing		

Step 3 - Choose or add a database table. Each extension group can contain multiple tables. There are two types of tables.

- **One-to-One tables** hold data that can be defined only once for a student. These are the tables that will hold new or migrated custom fields.
- **One-to-Many tables** can have multiple records for a single student. Examples of these types of tables in core PowerSchool include CC, StoredGrades, and Log

U_Def_Ext_Students is the default extension table when extending the Students table. This is the option that will be used if the Basic Extension type is selected in step 1.

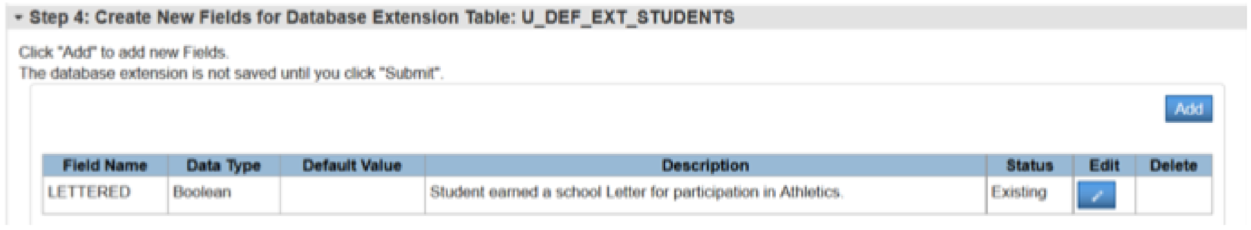


▾ Step 3: Choose or Add New Database Extension Table for U_Students_Extension

Choose a table to work with, or click "Add" to add a new table.
Then click "Next".

	Extension Table Name	Table Type	Description	Date Created	Last Modified	Status	Edit	Delete
●	U_DEF_EXT_STUDENTS	One-to-one		9/20/2014	9/20/2014	Pending	✎	−

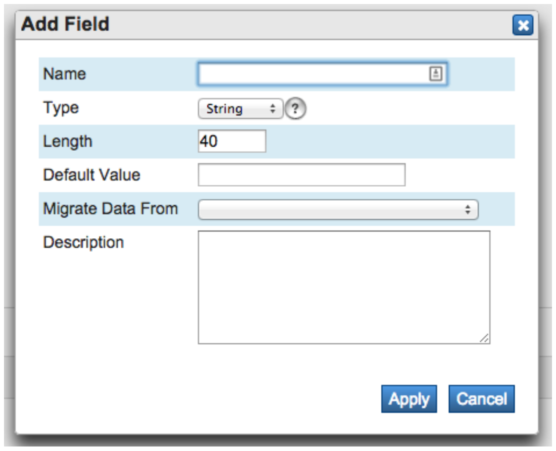
Step 4 - Create new field for the selected table. Here the user can create new fields. When the Add button is clicked, the Add Field dialog will appear.



▾ Step 4: Create New Fields for Database Extension Table: U_DEF_EXT_STUDENTS

Click "Add" to add new Fields.
The database extension is not saved until you click "Submit".

Field Name	Data Type	Default Value	Description	Status	Edit	Delete
LETTERED	Boolean		Student earned a school Letter for participation in Athletics.	Existing	✎	



Name	
Type	String
Length	40
Default Value	
Migrate Data From	
Description	

Field	Description
Name	Enter the field name as it will appear in PowerSchool.
Type	Select the filed data type from the pop-up menu: • **String**: Fixed-length character string. Strings in excess of 4000 characters will be truncated. • **Integer**: A number that can be written without a fractional or decimal component. • **Date**: A point-in-time value. • **Double**: An approximate representation of a decimal value. • **Boolean**: Data has two values (1=true and 0=false). • **CLOB**: (Character Large Object) data stored in a separate location referenced by the table. • **BLOB**: (Binary Large Object) collection of binary data stored as a single entity.
Length	Enter the maximum length of data that can be entered in the field.
Default Value	Enter the default value for the field.
Migrate Data From	Choose a legacy custom field in order to migrate data from the selected field to your database extension field. The default Type is set to String and the default Length is set to 4000 automatically. This option is unavailable if there are no existing legacy custom fields.
Description	Enter a brief description of the field.

Important notes about field creation

- Once a field has been created, it cannot be modified. Take time to consider names, data-types, and lengths carefully before submission.
- Leave the 'Migrate Data From' dropdown empty to create a new field.
- If migrating a new extended field will be created and all data from the custom field will be migrated into the newly-defined extended field.
- Fields can be renamed when migrating. PowerSchool maps previous names to the new extended names to ensure that no custom pages or queries will break.

Using One-to-One Extensions

- Searching, exports, list students (wherever the PS application itself asks you for a "fieldname")
  - ExtensionGroupName.Field_Name.
  - Example: U_Students_Extension.DistrictID
- Reports that use DATs (wherever you normally put ~(fieldname))
  - ~(ExtensionGroupName.Field_Name)
  - Example:~(U_Students_Extension.DistrictID)
- HTML pages
  - [PrimaryTable.ExtensionsGroupName]FieldName
  - Example: [Students.U_Students_Extension]DistrictID
- SQL Queries.
  - The extended database tables link on {CORETABLE}DCID

```
SELECT * FROM
CoreTable
LEFT JOIN ExtendedTable  ON CoreTable.DCID = ExtendedTable.StudentsDCID
```

  - EXAMPLE:

```
SELECT
s.lastfirst, s.grade_level, s2.districtid

FROM
students s
LEFT JOIN U_DEF_EXT_STUDENTS s2 ON s.dcid = s2.studentsdcid

WHERE
s.enroll_status=0 AND s.grade_level=12 AND s.schoolid = 100
```

One-to-Many Extension Tables

A one-to-many extended table allows you to have multiple records that are tied back to a single parent record. For example, multiple college applications for a single student.

- To work with one-to-many tables you use the tlist_child tag. This tag will create a table, so use it outside of any other tables, but still within a standard type page that has a form tag with the usual submit button and associated hidden inputs.

  ```
  ~[tlist_child:<CoreTableName>.<ExtensionGroup>.<ExtensionTable>;displaycols:< List of Fields>;fieldNames:<List of Column Headers>;type:<FormatName>]
  ```

  Example:

  ```
  ~[tlist_child:Students.U_CollegeApp.U_Applications;displaycols:Institution,Request_Date,Status;fieldNames:Institution,Request Date,Status;type:html]
  ```

- SQL query example for a one-to-many table.

  ```
  SELECT
          s.lastfirst,
          app.institution,
          app.request_date,
          app.status
  FROM
          students s
          LEFT JOIN U_Applications app ON s.dcid = app.studentsdcid
  WHERE
          s.id = 2
  ```

Independent (Standalone) Extension Tables

Creates a table that is not associated with any existing PowerSchool table. An example might be a list of college institutions.

- To work with independent tables use the tlist_standalone tag. It works much like the tlist_child tag.

  ```
  ~[tlist_standalone:<ExtensionGroup>.<ExtensionTable>;displaycols:<List of Fields>;fieldNames:<List of Column Headers>;type:<FormatName>]
  ```

Example:

```
~[tlist_standalone:U_CollegeApp.U_Institutions;displaycols:Institution_Name,Phone,URL;fieldNames:Instit
ution Name,Phone Number,Web Address;type:html]
```

- Example SQL Query for Independent table:

```
SELECT * FROM U_Institutions
```

Even more advanced?

- Refer to the Advanced User Guide for Database Extensions for special formatting of tlist_child and tlist_standalone tags.
- The forums are starting to discuss even more advanced applications of extended tables for when tlist_child and tlist_standalone aren’t sufficient.

Page Fragments

Page fragments are ways to customize any existing PowerSchool page or even new pages without customizing the page itself. Rather, you create a snippet of code that is inserted dynamically into the existing page via an “insertion point”, a special location in the source code of a page.

- With insertion points, ***the original source page does not have to be customized*** in order to add new content to that page. This can help dramatically cut down on the number of custom pages that need to be created and subsequently updated when a new version of PowerSchool is released.
- You can physically move fragments around on the page using client-side DOM manipulation via standardized metadata.
- You can insert a page fragment into a wildcard, thus being able to customize multiple pages at once - wherever that wildcard is used.

A page fragment is simply a snippet of content to be added to a target page. It could be something simple like the following example:

```
<p>Hello world! I'm an auto-inserted page fragment.</p>
```

Or, a page fragment could be a complex combination of HTML code and jQuery scripts. Because page fragments will be inserted in to existing PowerSchool HTML pages they do not require any of the standards HTML <head>, <body>, or other tags. The main page already contains those tags.

The naming of your page fragment is the key. Always save your page fragments to the same directory where the source page or wildcard exists.

- Name_of_file (without the “.html”) +
- .name_for_page_fragment (whatever you want to name it) +
- .insertion point (a common insertion point is “content.footer”) +
- .txt

Example: **home.Emergency_Numbers.content.footer.txt**

Example of jQuery page fragment for the modifydata DistrictID

```
<script>
$j(function() {
  $j("table tr").eq(0).after('<tr>\
    <td class="bold">District ID</td>\
    <td><input type="text" name="[Students.U_Students_Extension]DistrictID" value=""></td>\
    </tr>\
    <tr>');
});
</script>
```

Note: If a field has validation, PowerSchool can add additional characters beyond your control that can break the method above. Here's another example using the following pattern:

- Build the content you want to add in a hidden element (such as a table).
- Use jQuery to move the element to your desired location
- Delete the hidden table

Example that adds the legacy "Family_Ident" field to the "Modify Data" screen.
(For a more complete example that also includes a function to automatically select the next available FamilyID see the "Family Management" page at psugcal.org)

Filename: modifydata.familyid.content.footer.txt

```
<!-- create a hidden table with added rows -->
<table id="familyidhiddentable" style="display: none;">
<tr id="familyidrow">
    <td class="bold">Family ID</td>
    <td><input type="text" id="familyidfield" name="[Students]Family_Ident" value=""></td>
</tr>
</table>

<!-- use jQuery to move the inserted rows to target table and remove the hidden row -->
<script>
$j(function() {
    /* place the family id row at the beginning of the table */
    $j("table:first").prepend($j("#familyidrow"));

    /* remove the hidden table */
    $j("#familyidhiddentable").remove();

});
</script>
```

Database Extension Plugins

PowerSchool provides an exporting function that builds a complete plugin package containing database extensions and custom pages/fragments in one file. This plugin can then be easily imported into other PowerSchool servers in one simple step.

You work with Plugins from **System - System Settings - Plugin Management Dashboard**

Plugin advantages:

- An improvement over CPM "Import" ability
- Easy to share
- Can import files that CPM cannot import, like image and PDF files.
- Complete "package" of database extensions plus pages/fragments. Files will show up in CPM.
- Can be **disabled** with one click. All the files will still be there and even show in CPM, but will not be served.
- The plugin can be **deleted** which deletes the files/pages. However, extended tables and fields CANNOT be deleted.

Many of the popular customizations in the community have already been re-written as plugins and are now very easy to install.

A plugin does not have to contain database extensions, many just have custom pages and those won't add any extended tables to your server.

Creating Plugins

For more on how to create/export plugins and other information, please refer to the "Advanced User Guide for Database Extensions". The unpublished link to create a plugin is:

```
https://<server>/admin/customization/CreatePackagePage.action
```

Note: The database extensions exist in the "user_schema_root" folder and end in .xml.

Appendix

Page Fragment Examples

Learning to use jQuery/Javascript to move content around on a web page is important for creating good page fragments. Here are some examples.

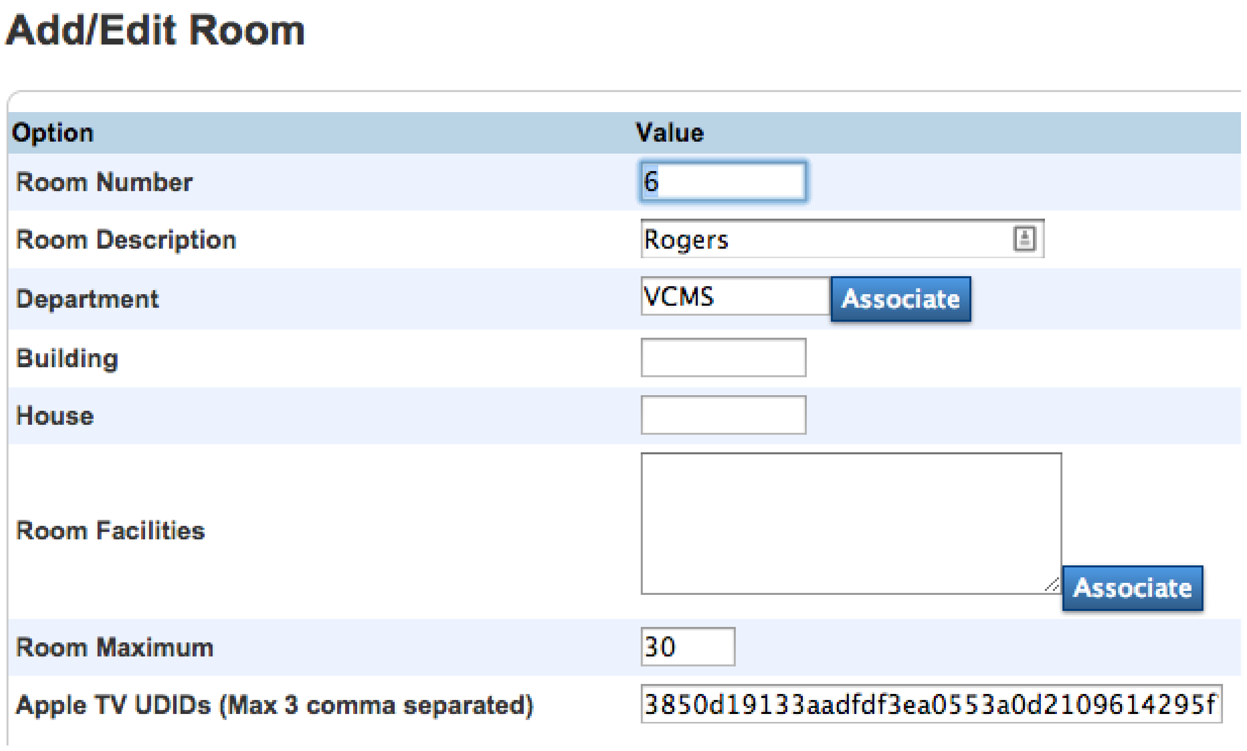
roomedit.appletvudid.content.footer.txt

This example is used at Valley. We use a Mobile Device Managment solution that include AppleTV management. Our exports from PowerSchool include the UDIDs for our AppleTVs. We created an Extension Group named “U_JAMF” that extended the Rooms table. This example uses “append” to add another row to the bottom of the table.

```
<!-- Add Apple TV UDID to room edit. Note escapes for line breaks -->
<script>
$j(function() {

$j('table').append('\
   <tr>\
   <td class="bold">Apple TV UDIDs (Max 3 comma separated)</td>\
   <td>\
   <input type="text" name="[Room.U_JAMF]apple_tv_udids" value="" size=45 maxlength=128>\
   </td>\
   </tr>\
');

});
</script>
```



Add/Edit Room

Option	Value
Room Number	6
Room Description	Rogers
Department	VCMS
Building	
House	
Room Facilities	
Room Maximum	30
Apple TV UDIDs (Max 3 comma separated)	3850d19133aadfdf3ea0553a0d2109614295f

functions.lockers_resetclasscounts.content.footer.txt

Source: Michael Moore on the PowerSource forums. This example uses a method of building up the html content into a variable, then copies the contents of that variable “html” into the row “closest” to the “Incidents” link in the Special Functions screen. What’s of note here is not the actual html content, but the method used.

```
<script>
    var html = '~[if.is.a.school]';
    html += '<tr class="~[evenoddrow]">';
    html += '<td><a href="/admin/lockers/home.html">Locker Management</a></td>';
    html += '<td>Mass assign and other related functions.</td>';
    html += '</tr>';
    html += '[/if]';
    html += '<tr class="~[evenoddrow]">';
    html += '<td><a href="/admin/resetclasscount.html">Reset Class Counts</a></td>';
    html += '<td>Updates section student enrollment counts.</td>';
    html += '</tr>';

    $j("[href^='/admin/incidents/home.html']").closest("tr").after(html);
</script>
```

modifydata.districtid.content.footer.txt

In this example we're adding an extension DistrictID field to the “Modify Info” screen and inserting it after the 4th row (remember numbering starts at 0).

```
<!-- create a hidden table with added rows, use jQuery to move the inserted rows to
target table -->
<table id="districtidhiddentable" style="display: none;">
<tr id="districtidrow">
   <td class="bold">District ID</td>
   <td>
   <input type="text" name="[Students.U_Students_Extension]DistrictID" value="">
   </td>
</tr>
</table>
<script>
$j(function() {

    /* place the district id row at after the 4th row */
    $j("table:first tr").eq(3).after($j("#districtidrow"));

    /* remove the hidden table */
    $j("#districtidhiddentable").remove();

});
</script>
```

more2.additional_links.leftnav.footer.txt

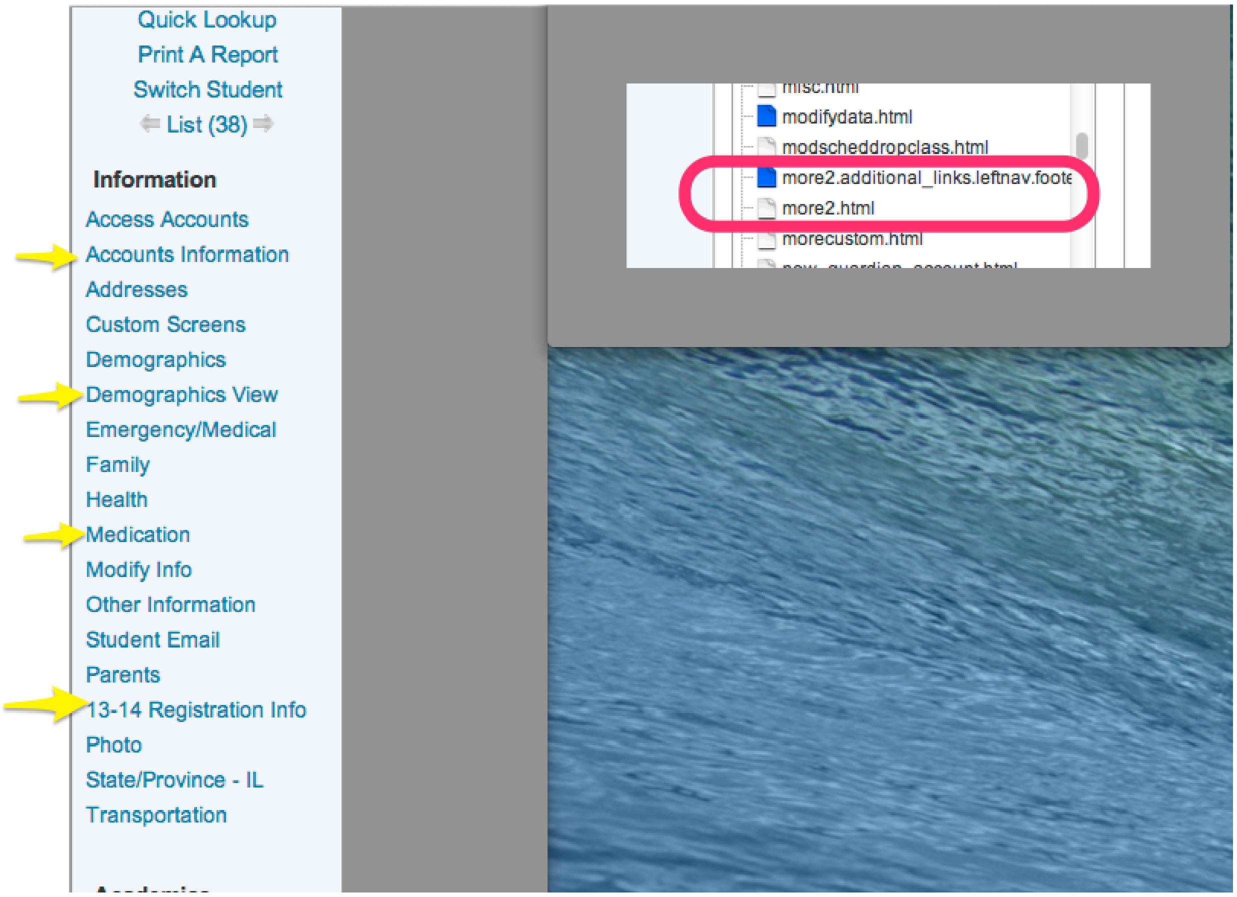
John Dunleavy on the forums posted this solution he used to add more links to the left nav area for a student (the more2.html page)

```
<!-- create new links and insert in more2 menu, use jQuery to place the inserted
rows in target row.  Many thanks to Roger Sprik & Matt Freund. -->

<script>
$j(function() {

    /* place the div id row at respective row */
    $j("table:first
br").eq(0).after('~[if.~(studentscreenaccess;generaldemographics.html)=1]<a
href="Accounts.html?frn=~(studentfrn)">Accounts Information</a><br>[/if]');
    $j("table:first
br").eq(4).after('~[if.~(studentscreenaccess;generaldemographics.html)=1]<a
href="demo.html?frn=~(studentfrn)">Demographics View</a><br>[/if]');
    $j("table:first
br").eq(8).after('~[if.~(studentscreenaccess;health/home_health.html)=1]<a
href="studentpages/MedConsiderations.html?frn=~(studentfrn)">Medication</a><br>[/i
f]');
    $j("table:first br").eq(13).after('~[if.~(studentscreenaccess;state.html)=1]<a
href="13_14Registration.html?frn=~(studentfrn)">13-14 Registration</a><br>[/if]');

});
</script>
```



Simpler Method to add links to the left nav (more2.html)

Jason Spring on the forums suggests this may be the simplest method to add more links to the left navigation:

```
<script>
$j("[href^='cumulative.html']").before('<a
href="MyPage.html?frn=~(studentfrn)">MyPage Text Label</a><br>');
</script>
```

reporttabs.fvsd_tab.report.tabs.txt

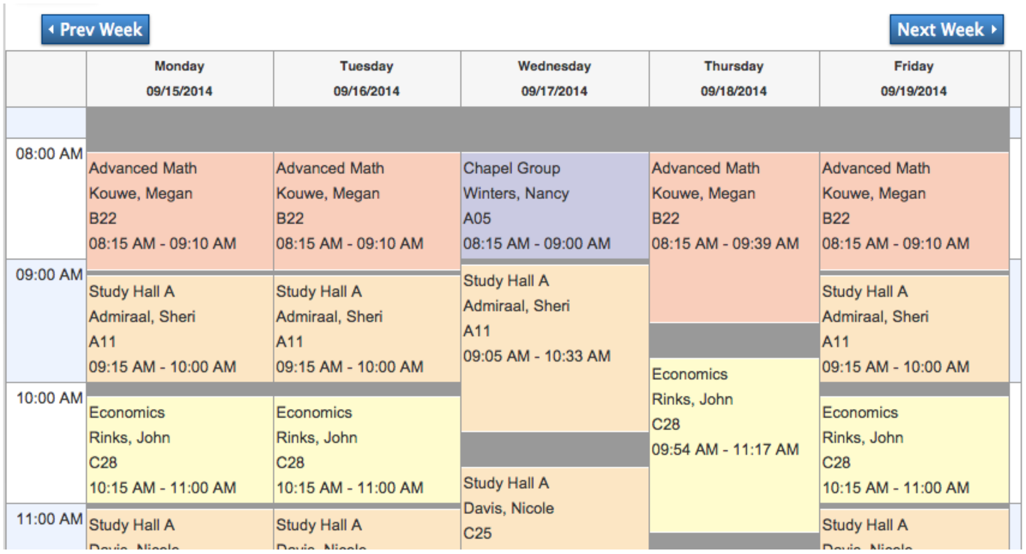
Nathan Jones suggested this method of building an “HTML Template” into a non-valid type=”text/template” script so that it’s ignored by the browser. Then use jQuery to copy it into place. Here we are adding the sqlReports 4 tab as the LEFT most tab on the System Reports page. (the sqlReports4 plugin already adds it to the right side, so you probably won’t actually use this specific example, but it illustrates a good method and the fact that reporttabs has an insertion point).

```
<script type="text/template" id="sqlreports-tab-template">
    <!-- sqlReports 4 begin -->
    <li class="~[if.~[gpv:repType]=sqlReports4]selected[/if]">
        <a
href="~[if.~[gpv:repType]=sqlReports4]#[else]/admin/sqlReports4/home.html?frn=~(us
erfrn)[/if]">ICSDReports</a>
    </li>
    <!-- sqlReports 4 end -->
</script>

<script>
var template = $j('#sqlreports-tab-template').html();
var select = $j('.tabs');
select.prepend(template);
</script>
```

Advanced Example: Add Prev Week and Next Week Buttons to the Bell Schedule screen

This advanced example was presented by Brian Andle at PSUG Vegas 2013. The presentation and finished fragment can be found on PowerSource at this link: https://powersource.pearsonschoolsystems.com/exchange/view.action?download.id=637



	Monday 09/15/2014	Tuesday 09/16/2014	Wednesday 09/17/2014	Thursday 09/18/2014	Friday 09/19/2014
08:00 AM	Advanced Math Kouwe, Megan B22 08:15 AM - 09:10 AM	Advanced Math Kouwe, Megan B22 08:15 AM - 09:10 AM	Chapel Group Winters, Nancy A05 08:15 AM - 09:00 AM	Advanced Math Kouwe, Megan B22 08:15 AM - 09:39 AM	Advanced Math Kouwe, Megan B22 08:15 AM - 09:10 AM
09:00 AM	Study Hall A Admiraal, Sheri A11 09:15 AM - 10:00 AM	Study Hall A Admiraal, Sheri A11 09:15 AM - 10:00 AM	Study Hall A Admiraal, Sheri A11 09:05 AM - 10:33 AM	Economics Rinks, John C28 09:54 AM - 11:17 AM	Study Hall A Admiraal, Sheri A11 09:15 AM - 10:00 AM
10:00 AM	Economics Rinks, John C28 10:15 AM - 11:00 AM	Economics Rinks, John C28 10:15 AM - 11:00 AM	Study Hall A Davis, Nicole C25		Economics Rinks, John C28 10:15 AM - 11:00 AM
11:00 AM	Study Hall A	Study Hall A			Study Hall A