Document Template Customization (Advanced)

Chapter 7

Document Owner Security

Document templates can be configured to allow only specified owners of a document to view that document, although administrative security groups can still be authorized to view documents regardless of owner. To configure this form of security, follow the steps below:

1. Enable the "Allow Document Owner Security" option in the template properties of the document template. Note that once this option is enabled, there will be an addition document template security right labeled "View if Owner". Users with this right can view documents of this type only if they are the owner. However, users with the standard "View" right will be able to view documents of this type whether or not they are the owner.

2. Configure one or more staff profile reference fields within the document template and assign the "Document Owner" field property. The staff referenced by these fields will be the "owners" of the documents. You must configure how these fields will be populated. An administrator could create the document and fill the field thereby assigning the document to the specified owners. Alternatively, document or sections actions could be used to populate these fields in order to assign ownership. For example, to set ownership of a document to the creator of the document, you can configure an action triggered by document creation and that sets the "document owner" field to the current user.

Note that document owner security is enforced in all areas of PowerSchool Special Programs including standard reports and advanced reports.

Document owner security, when implemented in staff documents, can provide a full-featured alternative to self-service documents. If you enable the "Allow Document Owner Security" in a staff document template, you can also enable a secondary option labeled "Staff Profile Is Document Owner". This secondary option causes the staff for whom a document is created to automatically become the owner. Follow the steps below to configure a security group that allows it members to create/edit/view staff documents for their own staff profile (while excluding other staff from accessing those same documents):

1. Under staff privileges, make sure the basic view staff privilege is enabled. Also enable "View Documents" and "Create Edit Template Documents". Note that this will not allow access to any template-based documents unless template rights are assigned. However, if you are using this approach, do not use file-based documents for staff because file-based documents do not support document owner security and therefore other staff will be able to see them. If you need the equivalent of file-based documents for staff, use file-attachment only document templates instead.

2. For each relevant staff document template, grant the "View If Owner" right and any additional editing rights to the document template. The bottom line is that any member of this group will only be able to view/access those staff documents for which the user is the owner. If you grant the standard "View" right, that will enable members to access other staff users' document.

3. Setting up the security like above will cause a "My Staff Documents" link to appear on the home page of members that give them quick access to their documents. However, make sure the "Access Self Service Documents" privilege under "Special Access Privileges" is turned off because otherwise a link to "My Self-Service Documents" will appear instead of the link to "My Staff Documents".

There is a related "Initially Restrict Document to Owners" document template property option that can be enabled in conjunction with the document owner security model. The option initially restricts access to new individual documents to the owners only, and prevents even staff users that have general view access from accessing them (restriction does not apply to ADMIN/CONSULTANT users). In the profile document list, documents with this restriction will have a padlock icon next to the document name. A document action modification of "Set Document Owner Restricted Status" is used to set or remove the restricted status at the appropriate time. After removal of the restricted status from a document, staff users with general view access will be able to access it.

Document Workflow

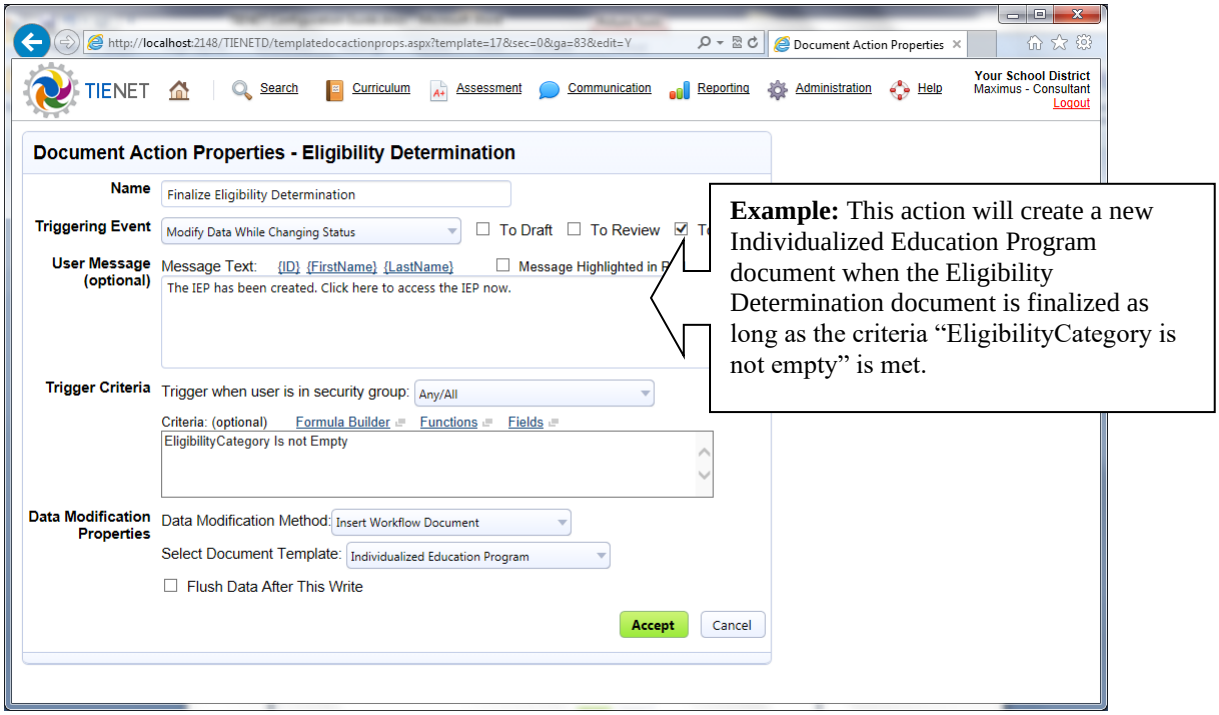
Document workflow refers to the ability of PowerSchool Special Programs to support a process that is larger than one document template through sequencing of documents and potential data flow from one document directly to the next. A simple typical document workflow for Special Education might be:

- Pre-Referral or Interventions
- Referral
- Assessment
- Eligibility Determination
- Individualized Education Program

The document-to-document sequencing is established by creating a document or section action in each relevant document template that triggers creation of a new document in the next document template. The document/section action should be designed to trigger at the right time, and have the right criteria for moving forward in the workflow process. The new document that is created has an explicit workflow link to the document that was used to create it, and this relationship is explicitly shown in the document list.

There are three types of document/section actions that can be used to create new workflow documents (i.e. documents that have a successor relationship to a previous document in a flow). The three approaches differ with respect to the degree of automation with which the succeeding workflow document is created.

- **Insert Workflow Document:** This is a data modification option associated with document action triggers that modify data, such as "Modify Data While Changing Status". Section actions do not support this. Such document actions create a new document of the specified target template such that the new document has a successor workflow relationship to the current document. The new document is created regardless of whether the user actually has access to the new document. If a document from the same target document template already exists with a workflow relationship to the current document, the action does not execute to avoid creating multiple successor documents. This type of action can optionally have a user message, but the user message will only appear to the end user if the end user has view rights to the next document created. Such messages appear also a hyperlink to the new document created. When using this type of document action, consider enabling the "Create Via Workflow Only" option in the template properties of the target document template to prevent documents of that type from being created with no workflow relationship.

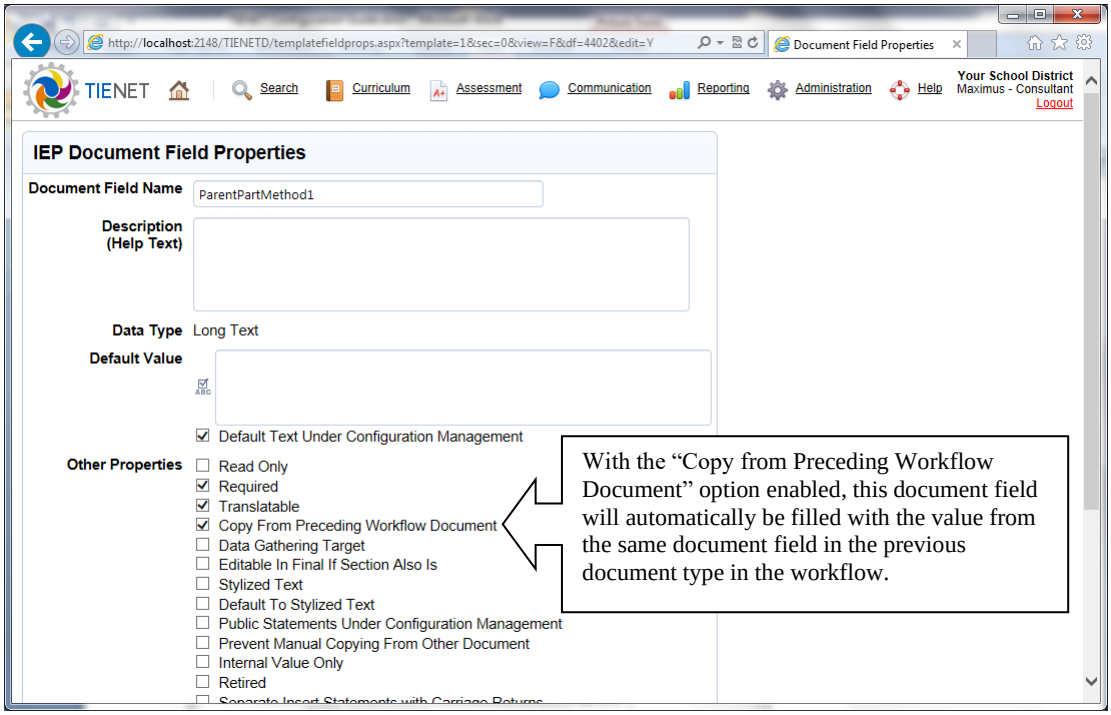


- **Enable Succeeding Workflow Document:** This is a data modification option associated with triggers that modify data, such as "Modify Data When Section Completed". This enables the creation of a new document of the specified target template such that the new document has a successor workflow relationship to the current document. If a document from the same target document template already exists with a workflow relationship to the user's current document, the action does not execute. While this is similar to 'Insert Workflow Document', it does not actually create the succeeding workflow document. Rather once the creation of a succeeding workflow document is "enabled", a special link to create the new document subsequently appears on the main documents list screen. Additionally, if the user attempts to create the next document directly from the new document menu, PowerSchool Special Programs offers the user the option of attaching the

new document to the pending workflow process. Also, note that the action will enable creation of the succeeding document even if the current user does not have access to the succeeding document.

- **Prompt Create Workflow Document:** A number of different document and section actions support a "Prompt Create Workflow Document" follow-up action where one can specify the document template for the document that the action will create. In actual use, the action will prompt the user to create a new document that will have a workflow relationship to the user's current document. If a document from the same target document template already exists with a workflow relationship to the user's current document, the action does not execute. Note that this document action only executes for users who have any draft edit rights to the target document template. Note that this type of guided action does not guarantee creation of the new document, but rather offers the user the option of creating it. Since this approach is purely a prompt and does not enforce the workflow, this could be used in conjunction with one of the previous approaches.

FYI – Copying data from preceding workflow document: When configuring document workflow, you can configure document fields to flow from Document A to Document B by marking the fields in Document B with the "Copy from Preceding Workflow Document" option as show below. To successfully flow, fields marked this way must also exist in Document A (with the same name and data type). A similar property can be set in child template properties within Document B that enables child template data to flow from Document A to Document B. To successfully flow, the child template must also exist in Document A with the same child template ID. Then any fields that exist in both child templates (with the same name and data type) will flow.



A possible approach is to have a few documents on the new document menu that are starting points into the Special Ed Process, and then to completely automate creation of the remaining documents via a workflow process. To do this, you would:

- Identify the document templates that are entry points into the Special Ed process like "Referral". Make sure these have workflow actions that create the subsequent documents according to logic of the business requirements.

- For the subsequent documents, enable the 'Create via workflow only' template option. Documents with this option enabled do not appear on the new document menu.

Note that the data gathering mechanism described in the next section is not enabled when the document workflow mechanism described here is being used.

Data Gathering

PowerSchool Special Programs has a previously described feature called “data flow” which allows document fields to be linked to profile fields such that any field values in the profile automatically flow into the corresponding document fields, either when the document is created or when it is later updated from the profile. You may have a situation where it would also be beneficial for values entered into previous documents (from the same template or even from a different template) to flow directly into a new document. The “Data Gathering” feature allows you to do this, but keep in mind that “data gathering” will only work if the document fields involved have the same name and data type in the source and target templates. Additionally, in the “document workflow” scenario described previously, the data gathering mechanism will not be used (i.e. document workflow overrides data gathering).

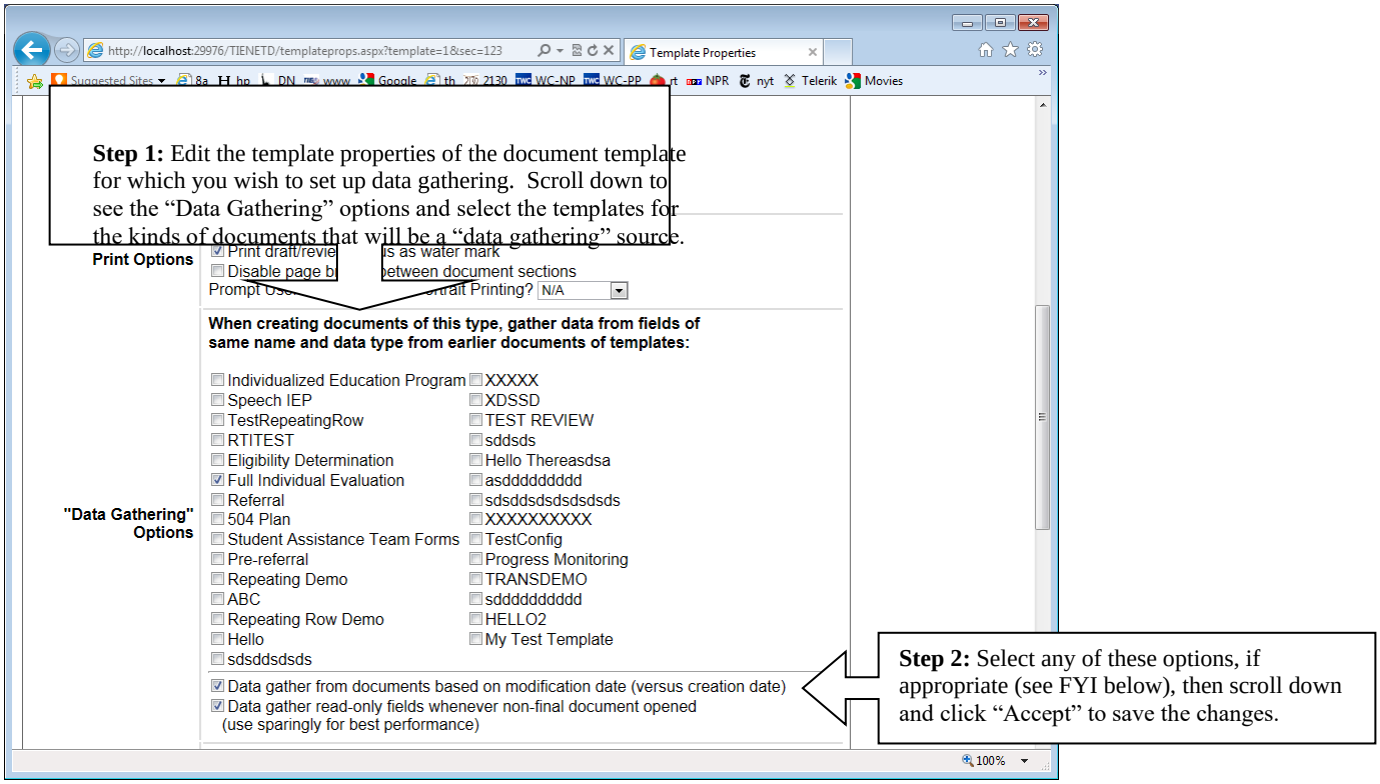
The material in this section will refer to data gathering sources and targets. A source is a template, document or field that data will flow FROM. A target is a template, document or field that data will flow TO. When a new document is a “data gathering” target, data can flow into it from more than one source document, but at most one source document per source template. If there is more than one potential source document for a source template, PowerSchool Special Programs selects the latest document based on the document’s creation date (or alternatively the modification date if so configured).

Configuring data gathering for a target template has two major steps:

In the target template, you identify which templates can act as source templates (via template properties).

In the target template, you mark the document fields that are to be target fields (via field properties). Note that a document field that is linked to a profile field cannot be a target for data gathering.

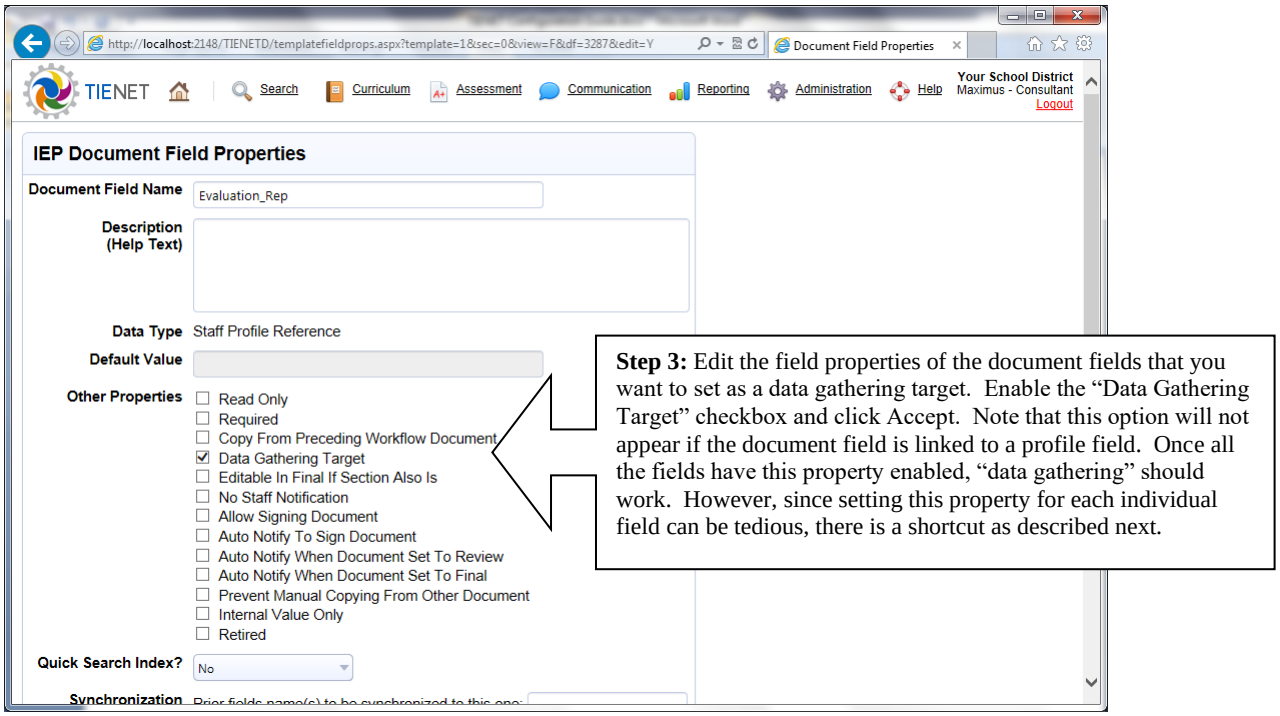
To set up “data gathering”, follow the steps below:

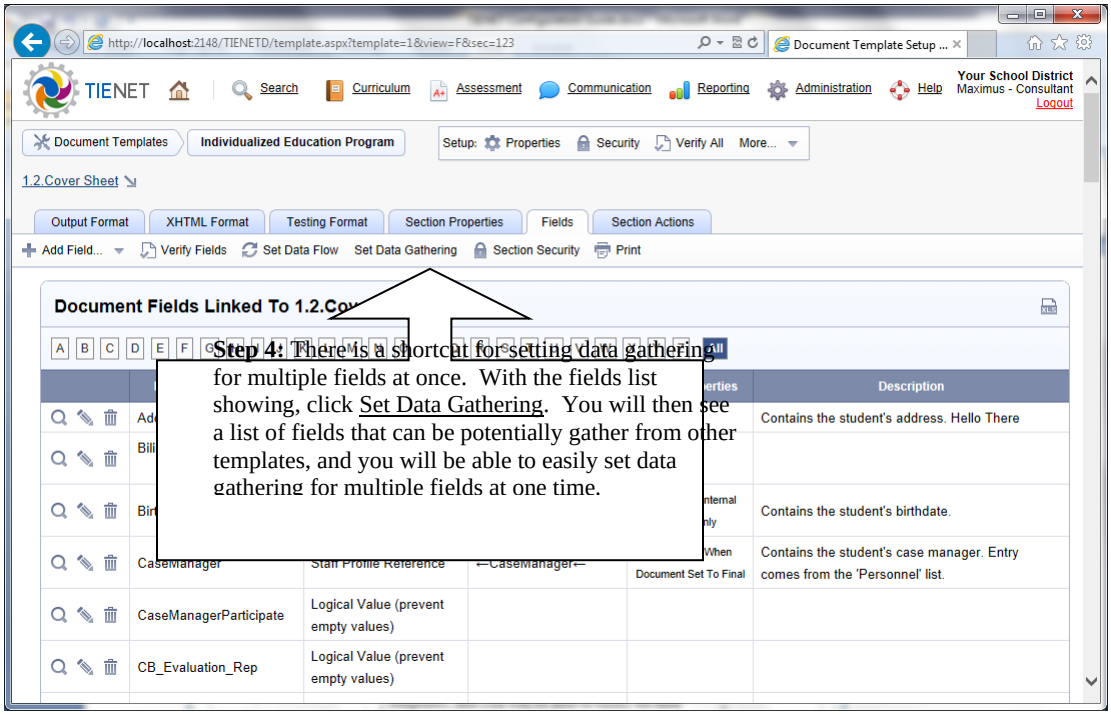


FYI – About "Data Gathering" options: There are several yes/no options in template properties that modify the behavior of "data gathering", as follows:

Data gather from documents based on modification date (versus creation date): When PowerSchool Special Programs performs data gathering, it will gather from only one source document per template. Normally, if there are several potential source documents, PowerSchool Special Programs selects the latest based on creation date by default. If this option is enabled, it selects based on modification date instead.

Data gather read-only fields whenever non-final document opened: Normally, data gathering occurs only when a target document is created. The yes/no option described here can be useful in situations where source documents may be modified after the target document is created. If this option is enabled, data gathering occurs whenever a non-final target document is opened, but this is limited to fields that are read-only in the target document.



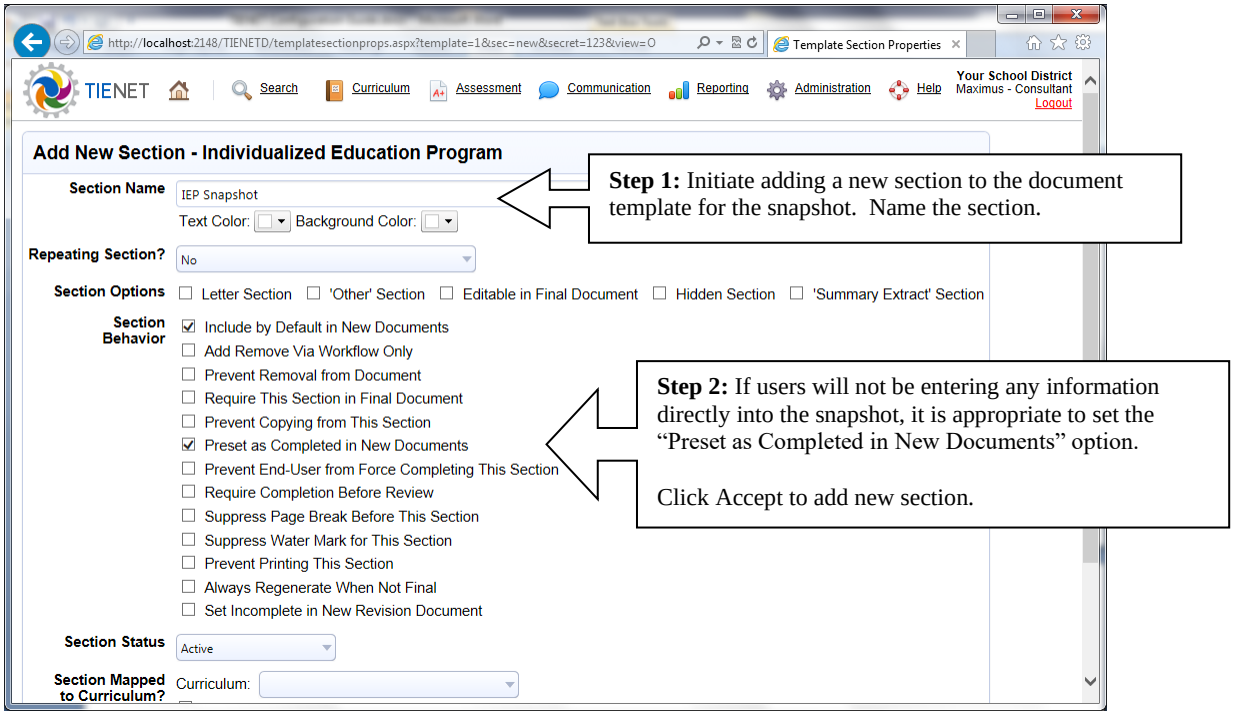


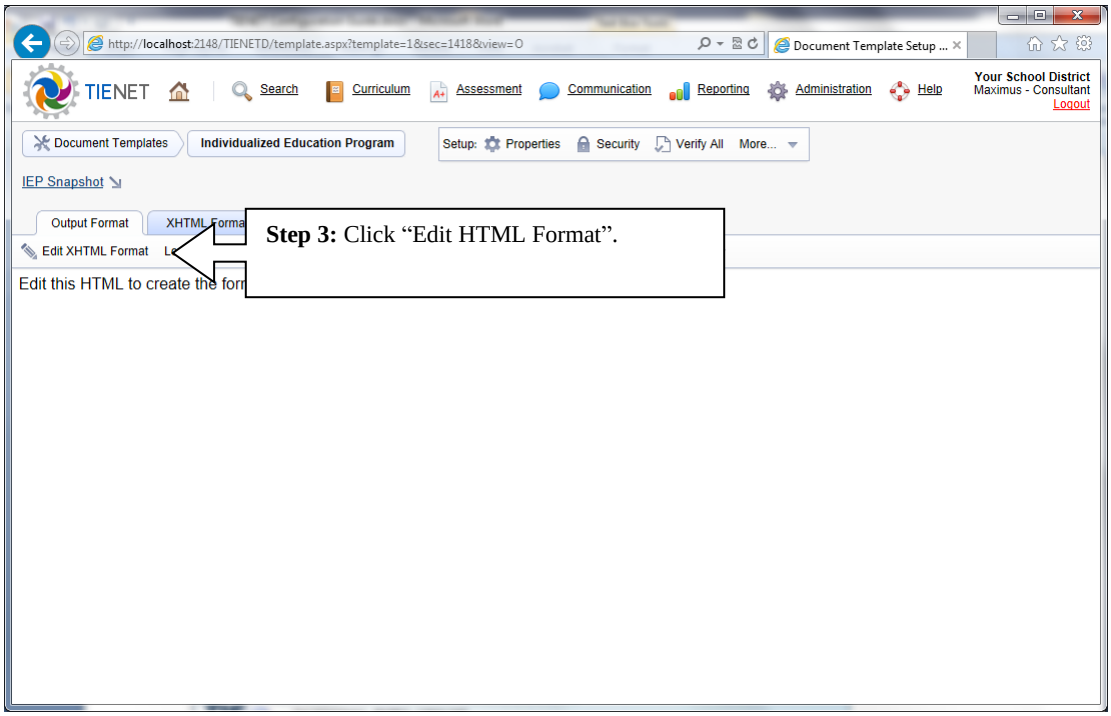
FYI – About Data Gathering Report: If you wish to see possible data gather sources for a particular section, bring up that section in the document template via "Data Schema". Switch to the "Linked Fields" display and click the Data Gathering Report link.

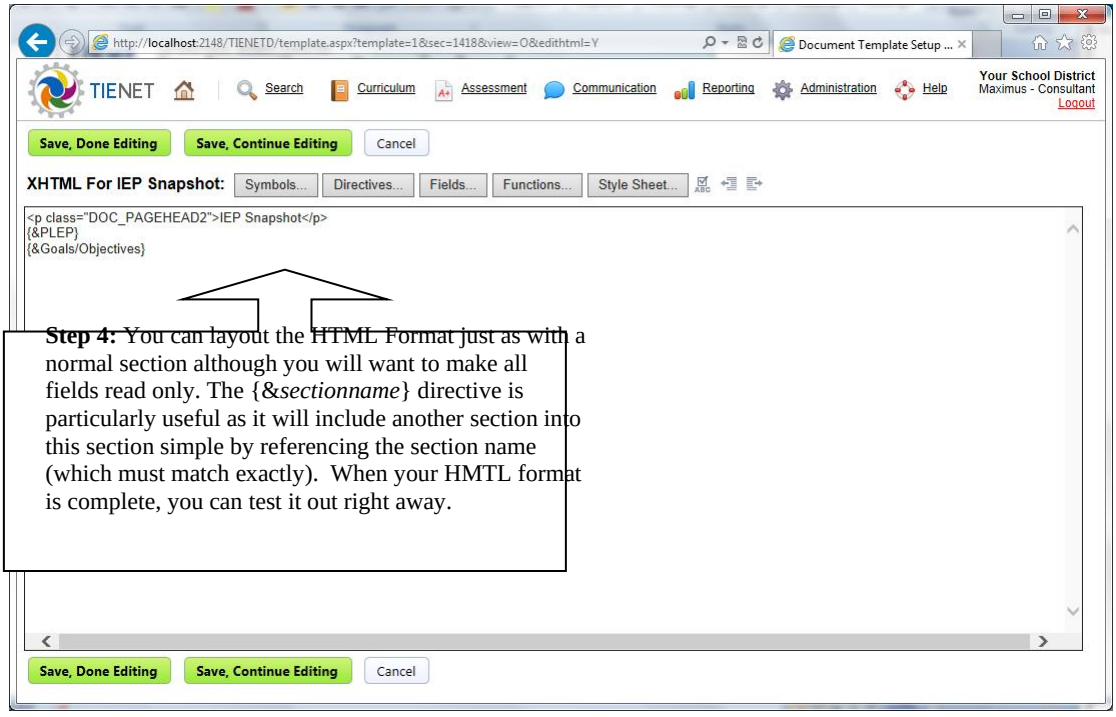
FYI – Data Gathering versus Data Flow: When a new document is created, first any fields marked for "data flow" from profile to document are copied into the new document. Then the "data gathering" process is applied which could possibly override values copied from profile to document via "data flow". This functions differently than the process of explicitly copying from a previous document that the user can select via the user interface. When a user explicitly copies from a previous document, fields configured to flow from profile to document are never copied.

IEP "Snapshot" Section

An "IEP Snapshot" section is simply a section at the end of the IEP template that repeats the most critical information from the IEP document to produce an abbreviated version of the IEP document. There are several techniques that facilitate configuring such a section. Follow the steps below:







Repeating Sections/Rows and Child Templates

The district may need to be able to have multiple copies or instances of a particular section within the same document. To provide this capability, you need to configure that template section as a "repeating section".

This feature can be used, for example, to implement a goals/objectives section. Alternatively, the district may require a table of information with rows and columns, including the ability to add additional rows as needed. This can be provided using the "repeating rows" feature. To implement a repeating section or repeating rows within a section, you must first create a child template within the document template. A child template is simply a storage area for a subset of fields/values that can repeat themselves in the same document, either in the form of a repeating section or repeating rows within a section. So the child template has its own data dictionary and the data records it contains are generally referred to as "child documents".

A child template can be used to store and provide the data simultaneously for a repeating section and a repeating row layout in another section. It is also possible to implement a child template within a child template (i.e. a grand-child template), which can be used to implement repeating rows inside of repeating sections.

Creating a Child Template: To create either a repeating section or repeating rows, you must first determine the exact set of fields with values that will be repeated and create a child template containing those fields. To create a child template, follow these steps:

1. Navigate into the configuration for the document template for which you want to add a child template, make sure a configuration task is selected, and then select 'Add New Child Template' from the top-most More dropdown menu. A dialog box appears allowing entry of key properties of the new child template.

2. To support a repeating section or basic repeating rows, leave the "Parent Child Template" property as n/a. If you do not see this option, it is because no child template already exists that could be a parent child template. Note that this property is generally only used to support inner repeating rows nested within out repeating rows or repeating section.

3. Leave the "Child Template Type" as its default value "Standard (Repeating)". The other option will not support repeating sections or repeating rows.

4. In the "Child Template ID" field, enter a identifier that must be unique across all child templates in the document template.

5. In the "Child Template Name" field, enter a descriptive name for the child template. The name will appear in the user interface for end users, so the name should work grammatically in phrases like 'Insert New____'.

6. All other options will be covered later in this section and can be left as the default settings until you are ready to configure them. At this point, click Accept to add the child template.

7. At this point, the child template will appear in the flyout menu, and is in fact already selected. With the child template selected, can add the fields you need to the child template using the "Add Field" dropdown menu. These are the fields that would repeat in either the repeating sections or repeating rows. Note that calculated fields in a child template can reference both fields in the child template and in the main document template.

Creating a Repeating Section: Once you have created a child template, you can create a repeating section based on that child template. You add a repeating section the same way that you add a non-repeating section, except that in the dialog box for adding the new section, you set the "Repeating Section?" dropdown to the child template you just created. When configuring this new repeating section, keep in mind the following points:

- When defining the HTML format for the repeating section, you can reference repeating fields from the child template as well as non-repeating fields from the main document template. However, only the repeating fields from the child template will be editable on that section.

- When defining "section actions" for the repeating section, formulas can reference both the repeating fields from the child template as well as non-repeating fields from the main document template.

- The end user with editing rights to the repeating section can add new "instances" or "copies" of the repeating section as needed. Since the user can view or edit one instance at a time, there is a dropdown menu that allows the user to select a particular instance to view or edit. By default, the instances are labeled by number (1,2,3….). However, to make it user-friendly, you can specify that a particular field in the child template supplies user-friendly labels for this dropdown. Edit the field's properties and check the "Child Document Title" property for that field. This property is available for character fields, short text fields, long text fields, keyword fields, and profile reference fields (e.g. staff fields). Other data types are not supported. The property can also be applied to a calculated field for these same data types as long as the formula for the calculated field only references fields in the child template (does not use the dot operator to reference specific keyword table fields or fields in other profiles). Note that you can work around any of these limitations by setting up a character data field and writing to it from the Data Form API, as follows:

    1. Create the character data field, which in this example is named MenuText.

    2. Place it on the form such that it is invisible but still writeable using JavaScript and the Data Form API. The @ modifier is used to do this, i.e. {MenuText:@}. However during development, you can use {MenuText:@"V"} to keep it visible temporarily.

    3. For each field directive that contributes to the menu text, introduce the J modifier to identify a new javascript function you will create in the JavaScript area to set the value of MenuText. The JavaScript function can utilize methods like DataFormAPI.getFieldKeywordTableValues to pull any values out of the keyword table (e.g. Description) and combine that with other fields. Below is an example of such a JavaScript function.

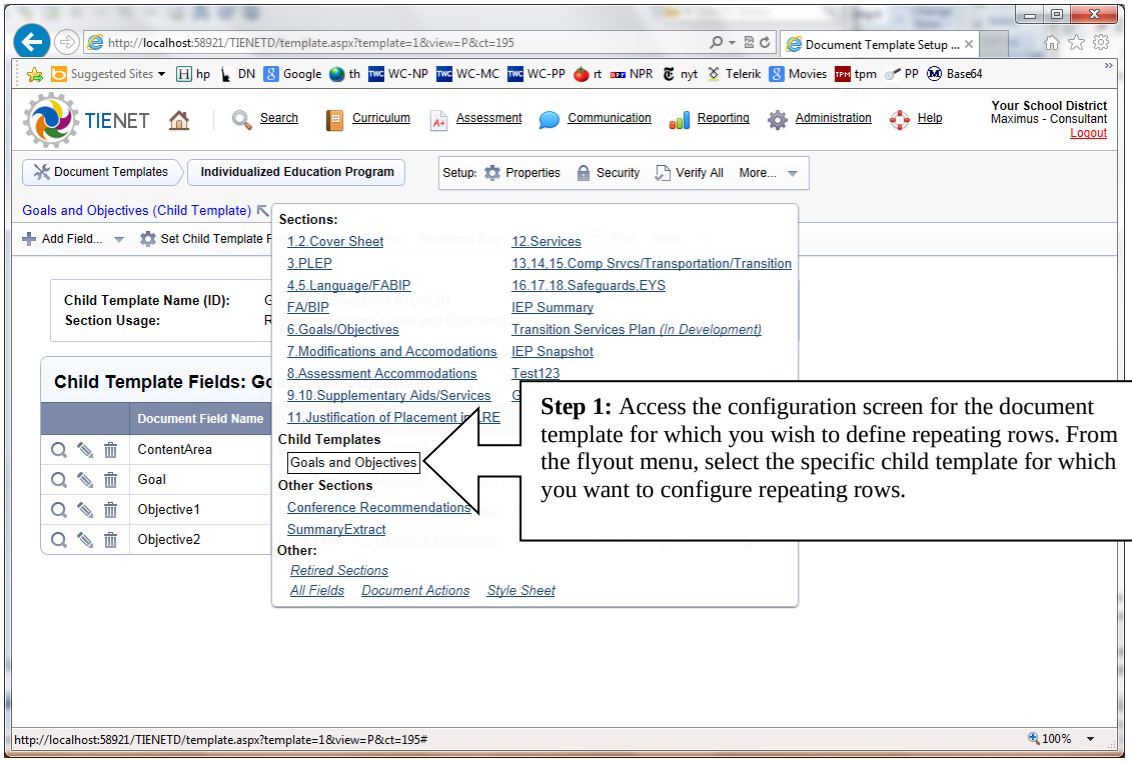
```
function updateMenuText()
{
    var keywordValue = DataFormAPI.getFieldValue('KeywordField');
    var characterValue = DataFormAPI.getFieldValue('CharacterField');
    if (!keywordValue) {
       //If no keyword chosen, then write character field to Menu Text
        DataFormAPI.setFieldValue(characterValue, 'MenuText');
    }
```

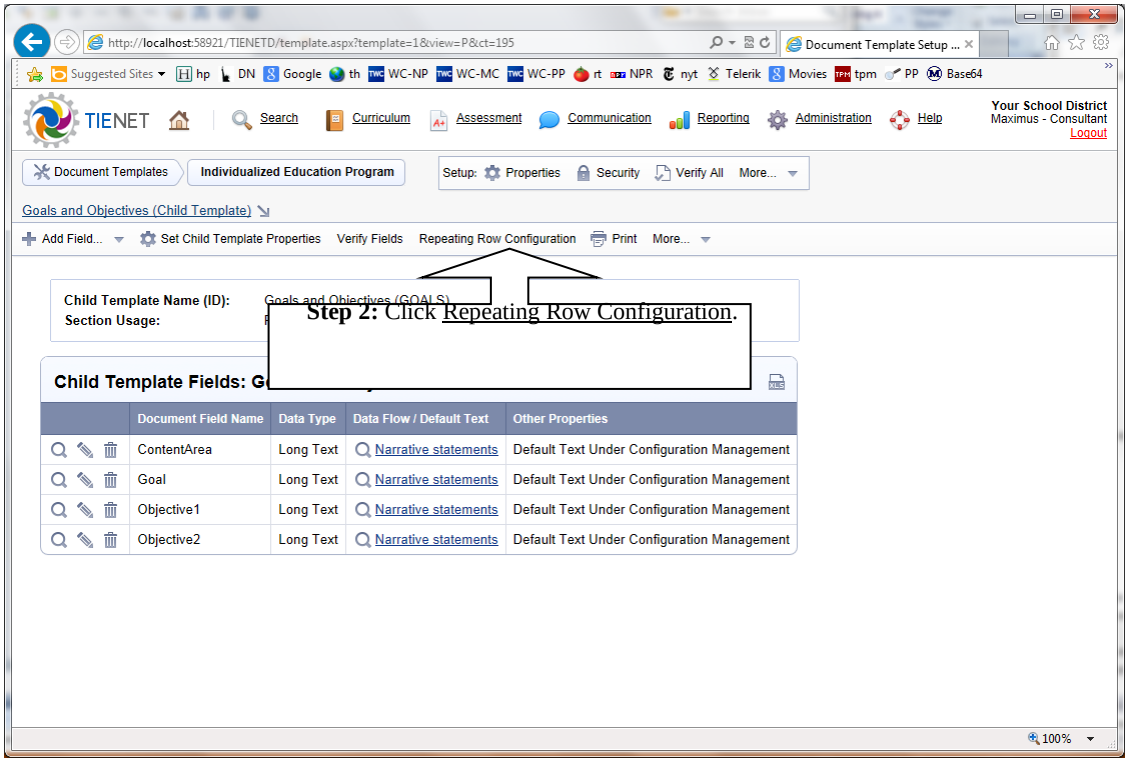
```
    else {
        //If keyword chosen, then write Description column of keyword to Menu
Text.
        DataFormAPI.getFieldKeywordTableValues('KeywordField',
            function(strFieldName, keywordRow) {
                DataFormAPI.setFieldValue(keywordRow.Description, 'MenuText');
            }
        )
    }
}
```

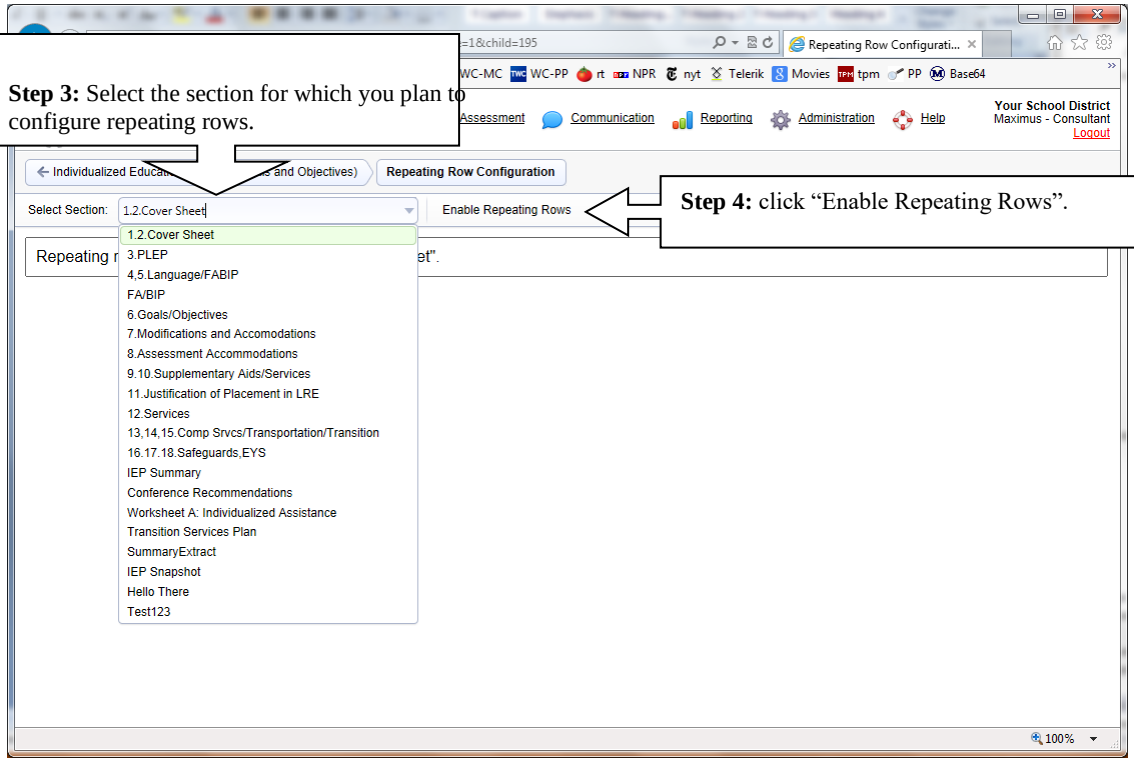
- By default, a user with edit rights to a repeating section will be able to add and delete instances of the repeating section. However, it is possible to "automate" the addition and removal of instances of the repeating section and hide the options that allow the user to do it. First, you would edit the properties of the repeating section and set the "Hide Repeating Section Insert Delete Links" property. You can then develop "Modify Data" section actions that add or remove child documents when executed. When the child documents are added or removed, the corresponding instances of the repeating section are added or removed. Another possibility is to allow the user to add the child documents via repeating rows in a different section, and then the repeating section will automatically be updated to match the repeating rows given that they are both based on the same child template.

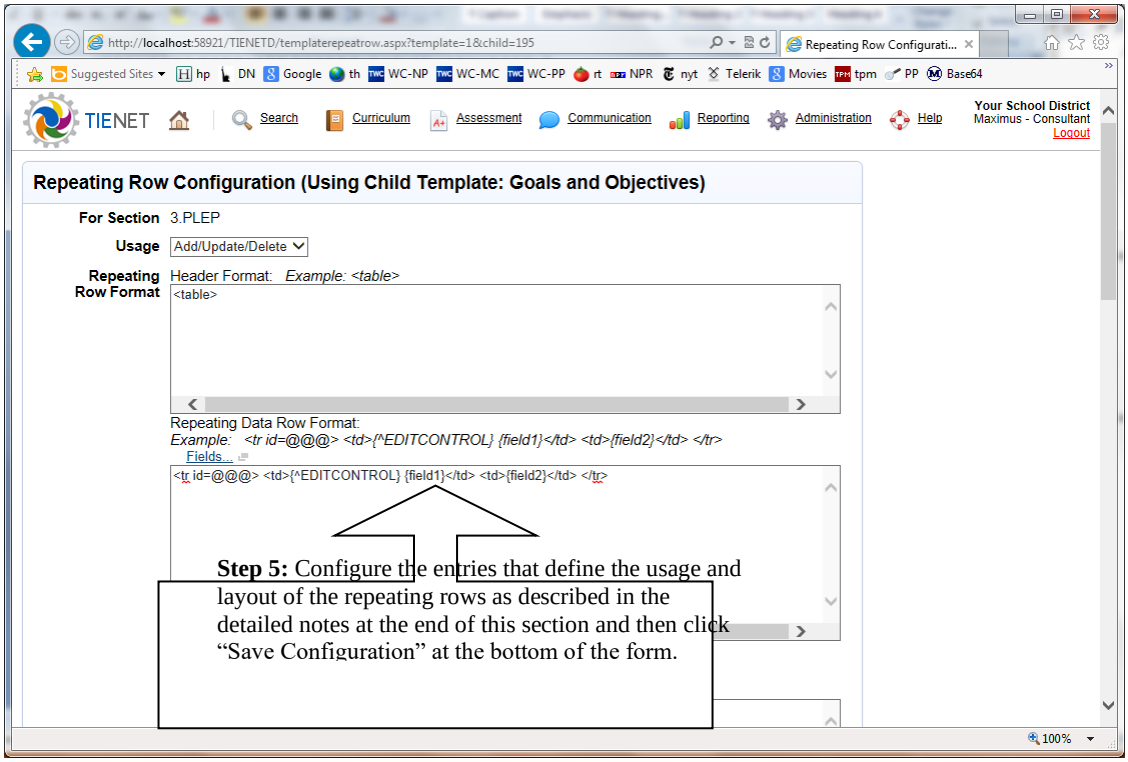
- Certain use cases may require you to prevent an otherwise authorized user from deleting a repeating section copy based on fields in the repeating section and/or based on the user profile. You can create a section action with the "Prevent Deleting Repeating Section Copy" trigger to accomplish this.

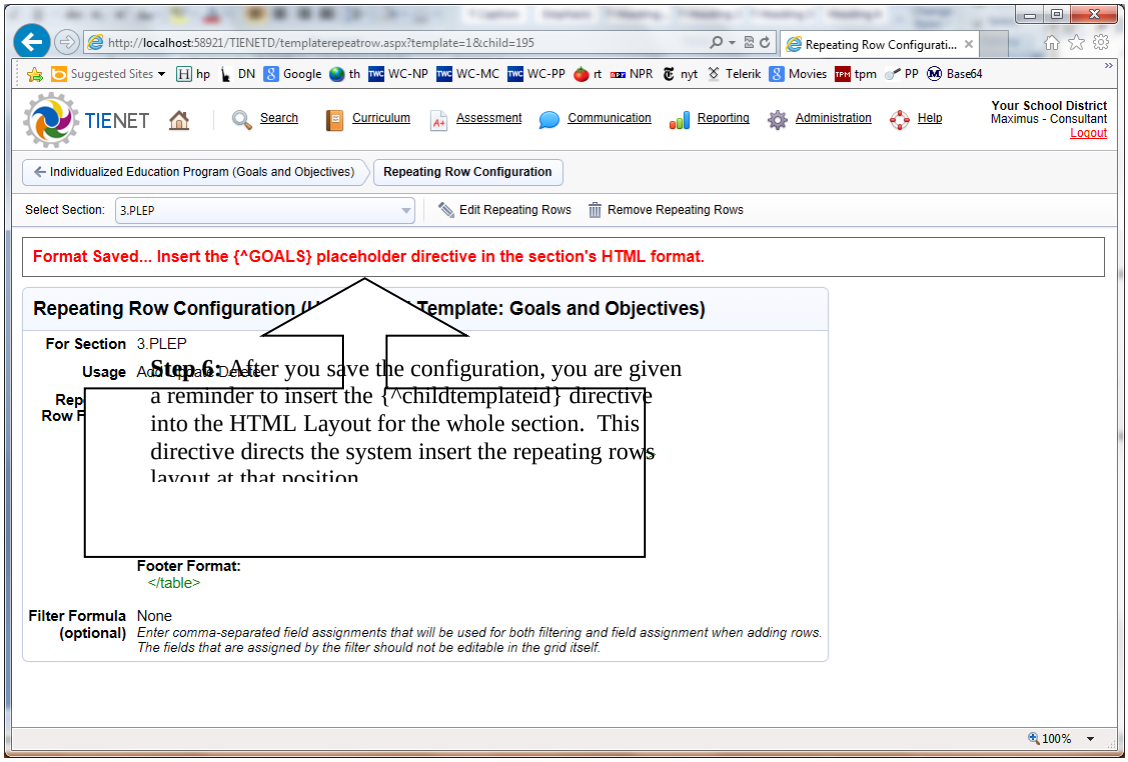
Configuring Repeating Rows: Once you have created a child template, you can configure a repeating rows layout within a particular template section. Follow the steps below to configure repeating rows:











The following describes the various configuration fields for configuring repeating rows:

- **Usage:** This option specifies how the repeating row layout will behave when the user is editing the section.
    - **View Only:** With this option selected, the repeating rows will be view only and no fields in the repeating rows will be editable, even in situations when the end-user can edit that section as a whole. This can be useful when the underlying data for the child template is entered somewhere else, perhaps in a repeating section elsewhere in the document.
    - **Update Only:** With this option selected, fields in the various repeating rows can be edited, but rows cannot be added and deleted by the end user who is editing that section. As with "View Only", this is useful primarily when the repeating data for the child template is initially inserted somewhere else, perhaps in a repeating section elsewhere in the document.
    - **Add/Update/Delete:** With this option selected, the end user can add, update and delete rows while edited that section.
- **Header Format:** This is an HTML format that is rendered at the start of the layout. For example, it might include a<table> tag and column headers. Note that this format can reference fields in the main document template, but not the child template.

- **Repeating Row Data Layout:** This HTML format is rendered for each repeating row. For example, it might include table row (enclosed by <tr> and </tr>) and normally references various child template fields. Note that this and other HTML formats in the repeating row layout can reference styles defined in the document template style sheet, and this is in fact advised. If the usage is set to Add/Update/Delete, this HTML format must meet several additional requirements. First, there must be an initial opening HTML tag that has an id of @@@, for example <tr id=@@@>. Without this, the user interface for adding, deleting and moving rows will not work properly. Secondly, the HTML format must include a special {^EDITCONTROL} placeholder directive that specifies where PowerSchool Special Programs should render icons that allow the user to add, delete or move rows.

- **Minimum # Rows:** You can use this to optionally specify a minimum number of rows that should be outputted. If actual number of data rows in a specific document is less than this configured value, then the "Empty Data Row Layout" configuration field described below will be used to render an HTML layout for non-existent rows. For example, if there are five actual rows in a document, and this configured value is 10, then the "Empty Data Row Layout" will be rendered five times. This is primarily used for cosmetic purposes, since a repeating row layout with zero rows might look strange.

- **# Extra New Rows For Adding:** For Add/Update/Delete usage only. Indicates the number of new empty rows to present to the user when in edit mode to allow the user to quickly enter multiple new rows. When the user fills in the empty rows, the user needs to click "Save and Continue" to generate a new set of empty rows unless the "0 (fast mode)" option in this dropdown is selected, which allows the user to simply add new rows as needed with no need to click "Save and Continue". More information about the "fast add" mode is given later in this section.

- **Empty Data Row Layout:** If there are fewer data rows than the "Minimum # Rows" value specified above, this field specifies the HTML layout to be used to render non-existent rows.

- **Footer Format:** This is an HTML format that is rendered at the end of the layout. For example, it might include a closing</table> tag. Note that this format can reference fields in the main document template, but not the child template. If the "fast mode" described in the description above for "# Extra New Rows For Adding" is configured, the footer format should include an {^EDITCONTROL} directive to indicate where the add row icon will be. You can also supplement the add row (plus) icon with a text label using the L modifier, i.e. {EDITCONTROL:L"Add Item"}.

- **Optional Filter Formula:** This formula, which can reference child template fields and document template fields, can be used to select a subset of data rows to display as repeating rows. However, when the usage is set to Add/Update/Delete, one must instead enter comma-separated field assignments that will be used for both filtering and field assignment when adding rows. The fields that are assigned by the filter, in this case, should not be editable in the grid itself.

- **Sort Formula #1-#4:** These optional sort formula(s), which can reference repeating fields and non-repeating fields, can be used to sort the rows in a particular order, with descending order as an option.

  The behavior of the sort formula is very different for Add/Update/Delete repeating row grids than it is for "View Only" or "Update Only" grids.

  Sort Formulas for "Add/Update/Delete" grid: In this case, the sort determines the order in which the rows are stored in the database as the user enters them, and if the same data also appear somewhere else in the document, the same ordering would by default be used there as well. Note that if this type of grid has no explicit sort, the end user will be able manually set the order of the rows (using arrow buttons). It is important to note that sorting takes place only when the section is saved, and so specifying a sort formula or changing it will not retroactively impact previous documents.

  Sort Formulas for "View Only" or "Update Only" grid: In this case, the sort is applied to the rows while they are being displayed. This means that if the same data appears somewhere else in the document, the sort will have no effect there.

  The "Optional Sort Group Header Format" can be used to create sub-headers at each configured level of sorting. The format is inserted into the document whenever the corresponding sort values change. These formats can include data references to the corresponding sort value fields or, more generally, any values that would be the same in all the rows encompassed by the sort group header.

Using the #IF directive in a repeating row grid: The #IF directive will work as expected as long as it does not reference fields in the child template, but if it does reference child template fields, then it will not work as expected in newly added rows that have not been saved yet. The issue is that a PowerSchool Special Programs formula can only be evaluated against real data rows, and the new repeating rows have no underlying data until they are saved. For this reason, it is better to use the #JSIF directive instead when the logic is based on child template fields. But if for some reason you must use the #IF directive, be aware that the #IF directive simply assumes the formula is true in new unsaved repeating rows. This allows for a work-around where you can put the content you do not want in new repeating rows in the #ELSE part of the directive.

Implementing "fast add" mode: This is a special mode of Add/Update/Delete repeating rows only that allows new rows to be added to the grid without postback. This mode is set by selecting "0 (fast mode)" for the "# Extra New Rows For Adding" field in the repeating row layout. To make this mode work correctly, you must configure a {^EDITCONTROL} directive somewhere in the "Footer Format" (described below) to serve as a placeholder for where the "add new row" icon will be. The end user will click this icon to add a new empty row. An example of such a footer format is as follows:

```
{#IFEDIT}<tr><td colspan=3>{^EDITCONTROL}</td></tr>{#ENDIF}
</table>
```

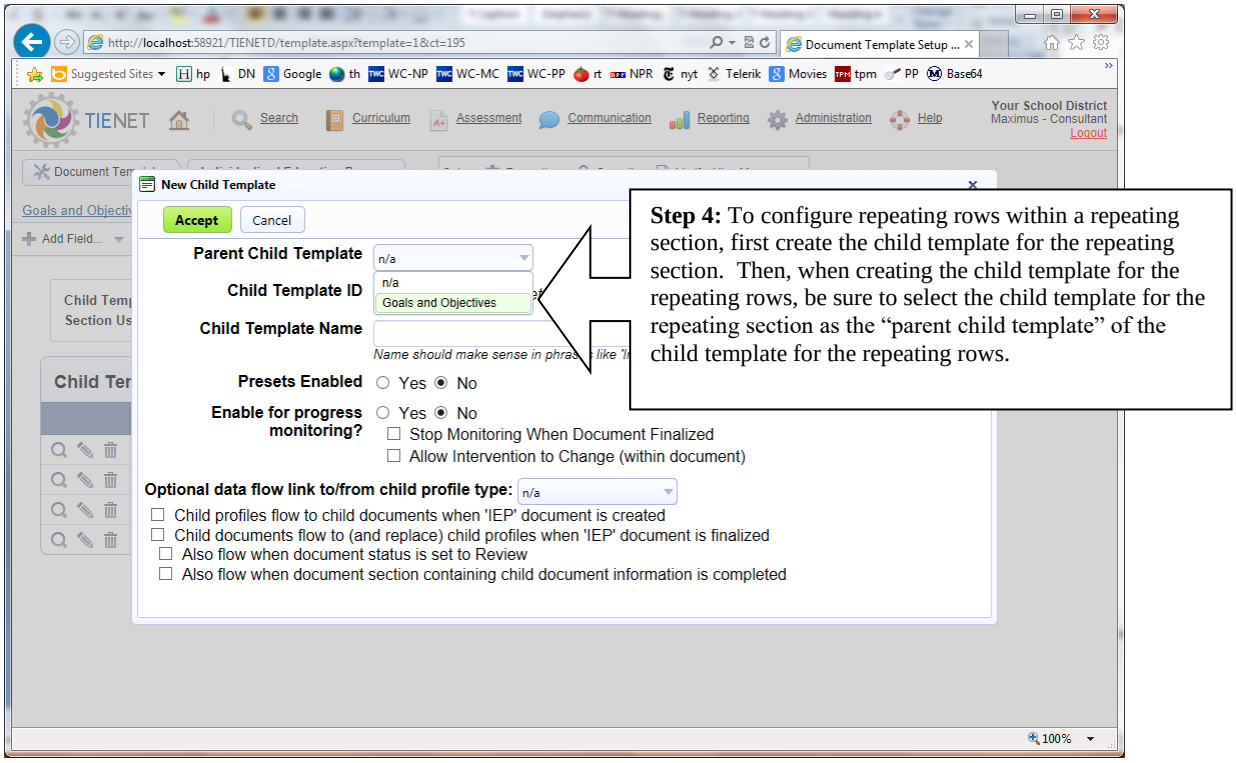
When fast mode is enabled, it is very important that the "Repeating Row Data Layout" be encapsulated within a single HTML tag. If you want the repeating row data layout to have two or more table row tags (<tr>), you can encapsulate the table rows within a single tbody tag. Similarly if you want the repeating row data layout to have two or more paragraph tags (<p>), you can encapsulate the paragraph tags within a single div tag. If this is not done or if there is other malformed HTML in the layout, the add button and/or the shift row buttons will not function correctly.

Implementing a requirement that the user enter at least one repeating row: To require the user to enter at least one repeating row for the section to be considered complete, simply introduce ":R" into the main repeating rows directive, as follows: {^INNERROWS:R}.

Allowing a user to drill down into repeating rows to access repeating sections: If you have a scenario where a document has repeating rows to contain the main details and then later repeating sections for more detailed information, it may be desirable to incorporate "zoom" icons in the repeating rows to allow the user to click into the corresponding repeating section. To implement this, simply introduce ":V" into the main repeating rows directive, as follows: {^DETAILROWS:V}. The zoom icons will always appear except on printed output. If there are certain users who should see the repeating rows but not the repeating sections, it is best not to use this feature since the zoom icons always appear.

Preventing a user from deleting particular repeating rows: Certain use cases may require you to prevent an otherwise authorized user from deleting certain rows based on fields in those rows and/or based on the user profile. You can create an "AllowDelete" logical field to allow or prevent rows from being deleted. Typically this would be a calculated field based on fields in the repeating rows and/or based on the user. If the "AllowDelete" logical field exists, then the delete icon is only shown for repeating rows for which "AllowDelete" is true.

Configuring Repeating Rows within a Repeating Section: You can implement repeating rows within a repeating section. Start by setting up the repeating section with its corresponding child template. Then, when creating the child template for the repeating rows, be sure to select the child template for the repeating section as the "parent child template" of the child template for the repeating rows. Other than that, the process is exactly the same as configuring repeating rows for a non-repeating section.

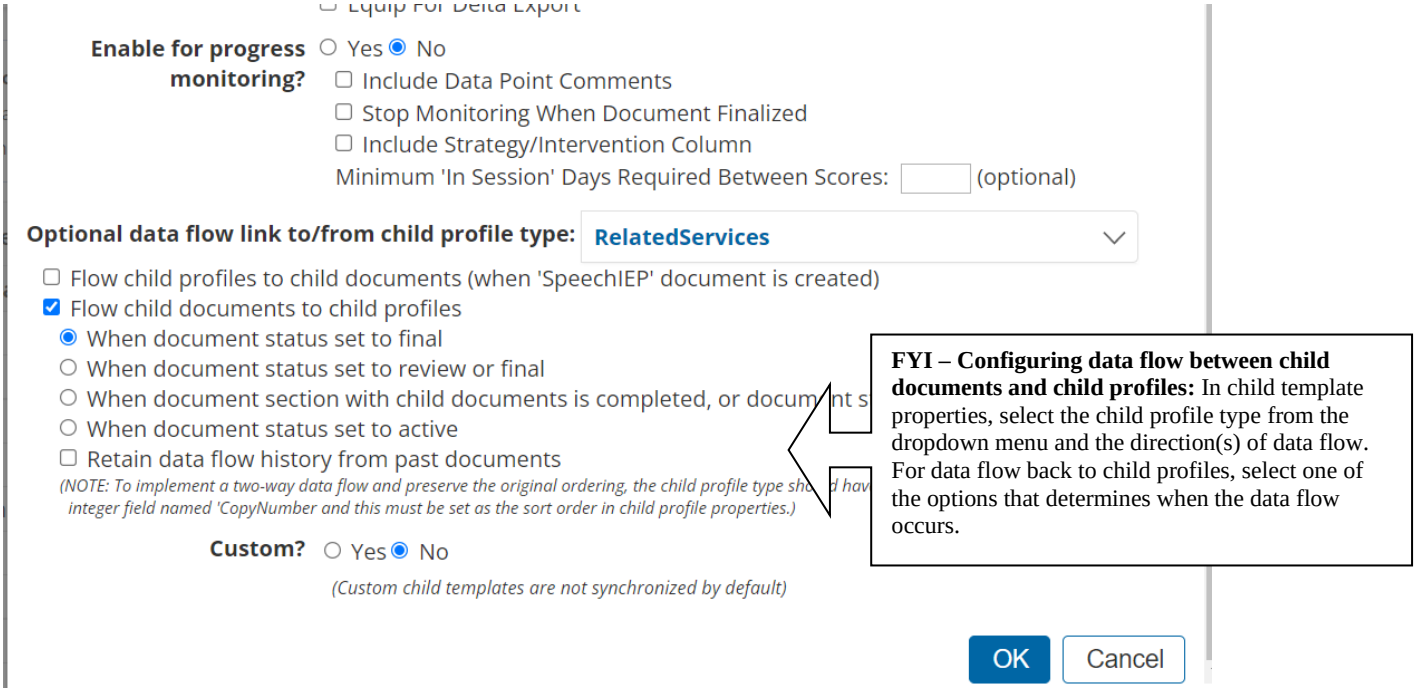


Configuring Repeating Rows that Selectively Introduce Repeating Sections: This describes how to support a scenario where you have 1) repeating rows that support adding and deleting rows, 2) repeating sections based on the same child template that are either view only or update only (no directly adding or deleting repeating sections), and 3) repeating sections should be omitted for some of the repeating rows. This scenario can be supported by introducing a logical field named "IncludeAsRepeatingSection" into the child template which can be either a data field or a calculated field. If a calculated field, you will find it can only reference other fields in the same child template (which helps guarantee that inclusion behaves in a deterministic manner). Also note that if you change the criteria associated with IncludeAsRepeatingSection after documents have already been created, the existing documents will not automatically update to reflect the new criteria. Lastly, you might want to mark the IncludeAsRepeatingSection as an internal value only so that it does not appear on report creation screens.

Nested Repeating Rows: Nested repeating rows (repeating rows inside of repeating rows) are configured by defining a repeating row layout for a grand-child template, and then positioning a reference to it (e.g. {^INNERDATA}) in the repeating body format of the outer repeating rows.

Optional data flow link to/from child profile type: Circumstances may arise where it would be useful to have the data in child documents flow to child profiles of the main profile triggered by specific events such as document finalization, and/or for data in child profiles to automatically populate child documents of a new document that is created. To set this up, simply access the child template properties, select the child profile type and the directions of data flow as shown in the screen below. Note that this kind of data flow relies on fields in the child template and child profile having the same names and data types. If either the name or the data type differs, no data will flow. For fields that have a length (e.g. character field), the

length should also be the same in the child profile and child template. Clicking Verify All will identify mismatches that may exist. Note that it is acceptable for a calculated field to flow to a data field although this clearly will not allow bi-directional flow.



The following data flow options are available in the child template properties screen:

- **Flow child profiles to child documents:** If enabled, any child profiles will flow to child documents when the document is created based on matching field names and data types.

- **Flow child documents to child profiles:** If enabled, any child documents will flow back to child profiles. The timing of when this occurs depends on the sub-option selected, as follows:

    - **When document status is set to final:** Child documents flow back to child profiles only when the document is set to final.

    - **When document status is set to review or final:** Child documents flow back to child profiles only when the document is set to either review or final.

    - **When document section with child documents is completed, or document is set to final:** Child documents flow back to child profiles when the section that contains them (either as repeating section or repeating rows) is completed. The child documents will flow back again when the document is set to final.

      (Advanced) There may be certain cases where a child template calculated field flows back to a child profile data field, and the calculated field formula refers to a top-level field

editable in a different section that does not contain the child template. In this edge case, it may be desirable to have the child documents flow back when the section where the top-level field is editable is completed, given that this top-level field will influence the data that flows back. You can accomplish this by enabling the "Trigger Child Profiles Flowback When Referenced Fields Edited" field property of the child template calculated field. Note that this property is only visible when the child template is configured to flow back to child profiles when the section containing the child template is completed.

- **When document status is set to active:** This option will only appear if the document template supports active documents. If this option is set, the child documents will flow back when the document is set as the active document, which occurs only after the document is set to final.

- **Retain data flow history from past documents [New in 21.11.1.0]:** When child documents flow back to child profiles, by default, any existing child profiles are wiped out and replaced by a new set of child profiles copied from the child documents. However, if you enable this property, then only child profiles that may have flowed earlier from the same document are replaced by a new set of child profiles copied from that document. This allows a history to accumulate across multiple documents. Note that prior to version 21.11.1.0, you would need to enable the "Child Document Data Flow History Enabled" property in the child profile type to achieve the same effect, but that child profile property is no longer needed and has been deprecated and removed as of 21.11.1.0.

- **Replace child profiles from original document with those from revision document:** As of 21.11.1.0, this option is only available when the document template supports revision documents and the "Retain data flow history from past documents" property described above is enabled. When this option is enabled (along with "Retain data flow history from past documents"), then when the child template flows back for a revision document, the child profiles for the original document or the previous revision are removed. Without this option enabled, child profiles that flow back from each revision will be retained.

How to limit which rows flow back from child template to child profile type: If a logical data or calculated field named AllowFlowback is added to the child template, only child documents for which AllowFlowback is true will flow back, and so this gives you precise control over which child documents flow back. Note that if child documents with AllowFlowback=true flow back during an initial flowback, but then flowback a second time with AllowFlowback having changed to false, child profiles from these child documents will be removed.

How to enable reflow from child profiles when the document is updated from the profile: Under certain conditions, child templates with "child profile to child document" data flow can be configured to "reflow" when a draft document is updated from the profile at a time later than its creation. At this time, this is not supported for child templates used in any repeating sections. A prerequisite to enabling "reflow" is to configure a profile reference field in the child template that refers to the source child profile type and has the "Child Document Title" and "Read Only" properties enabled. Note that only one field in the child template can have the "Child Document Title" property enabled. After

configuring this field with the required properties, a "Reflow When Draft Document is Opened" checkbox option will appear in child template properties. When this is enabled, the initial data flow upon document creation will automatically populate the profile reference field with the "Read Only" and "Child Document Title" properties so that it refers to the source child profiles. When the draft document is later updated from the profile, child documents that refer to a source child profile will be updated (or deleted if the child profile has been deleted), and any new child profiles will be inserted with the profile reference field populated.

Implementing a Goals Section

A goals section in an Individualized Education Program or similar plan typically has the following characteristics:

1. Allows entry of multiple goals, and multiple objectives for each goal.

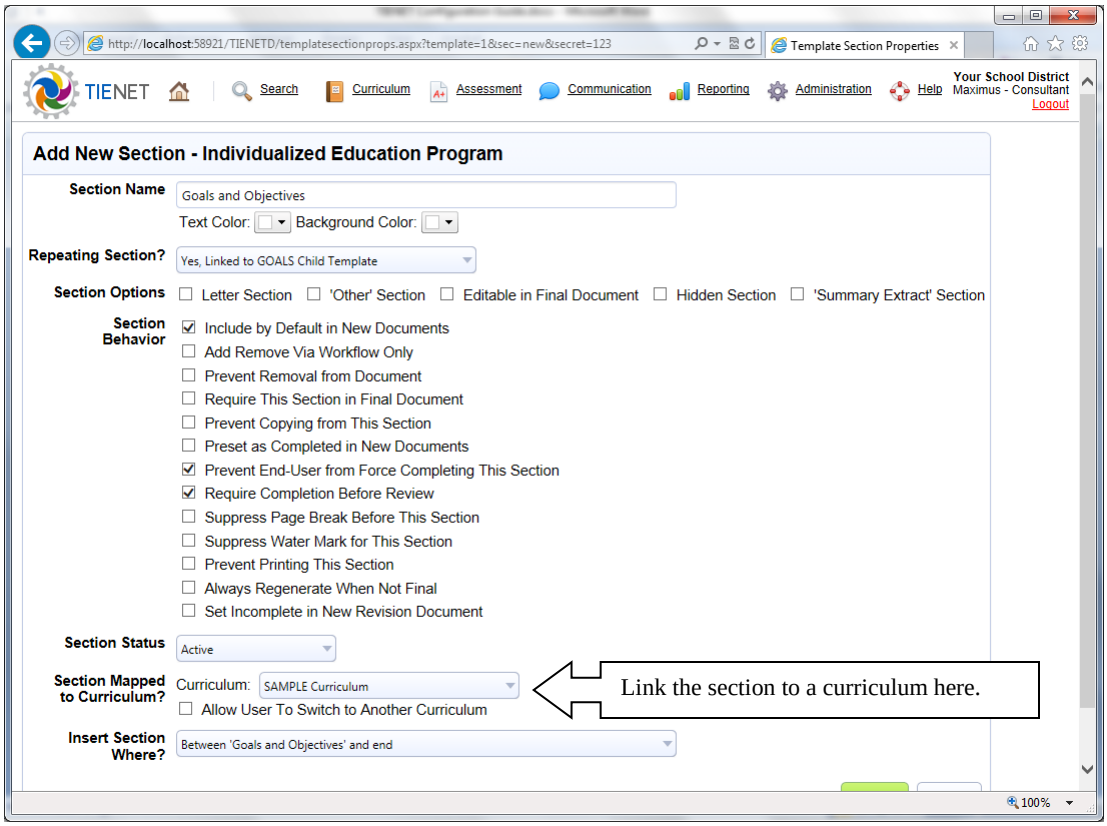
2. Either requires or allows the user to populate certain text fields by selecting or referencing curriculum standards.

3. Supports subsequent reporting or monitoring of student progress towards the goals.

Creating the Goals Section: The first characteristic above is typically accomplished by implementing the section as a "repeating section" with a corresponding "Goals" child template as described in the previous section of this chapter. In most cases, this repeating section will repeat for each goal, although alternative models are supported as well. Multiple objectives for each goal can be implemented as repeating rows within the repeating sections.

It can be beneficial to enable the "Presets Enabled" property of the goals child template. The presets productivity feature allows individual end users to save personal sets of commonly used goals and objectives under a name so that users can insert those presets into documents they are working for. End users can also share their presets with other users.

Mapping the Goals Section to Curricula: To allow end-users to populate fields in the section from the curriculum, the section must be first linked to a curriculum using section properties as shown in the screen shot below. If the section will only be populated from one curriculum, simply select that curriculum in the section's properties. If the section will be populated from more than one curriculum, then select a default curriculum in the section's properties and check the "Allow User to Switch to Another Curriculum" checkbox.



The fields in the section must also be mapped to the names of the curriculum outline levels they will be populated from. There are two m to this mapping.

The "implicit mapping" approach is more useful when the section will only be mapped to one curriculum. In this approach, the mapped fields the same names as the curriculum outline levels (specifically the singular form). Specifically, you need to add character or long text fields to the child template that have the same name as the corresponding curriculum level (singular form). The deepest level represented in the section can have multiple character or long text fields suffixed with a number. For example, if your curriculum has levels with the singular name of Subject, Goal, and Objective, then you might add long text or character fields to the child template with the names: Subject, Goal, Objective1, Objective2, Objective3, Objective4. Alternatively, you can have repeating rows within the repeating section to implement the objectives, and in this case, you could just have a single field named Objective within these repeating rows. Either way, with implicit mapping, you perform the mapping by naming the fields to match the outline level names.

The "explicit mapping" approach is often required when the section will be mapped to two or more curricula and the curriculum outline levels are named differently in each curriculum. Because the levels are named differently, the implicit mapping approach will not work here. In the explicit mapping approach, a text field can be mapped to any number of curriculum outline level names across multiple curricula. In the directive for each text field to be mapped to the curriculum, you use the F modifier to specify a comma-delimited list of curriculum outline level names (singular form) to which the text field will be mapped. For

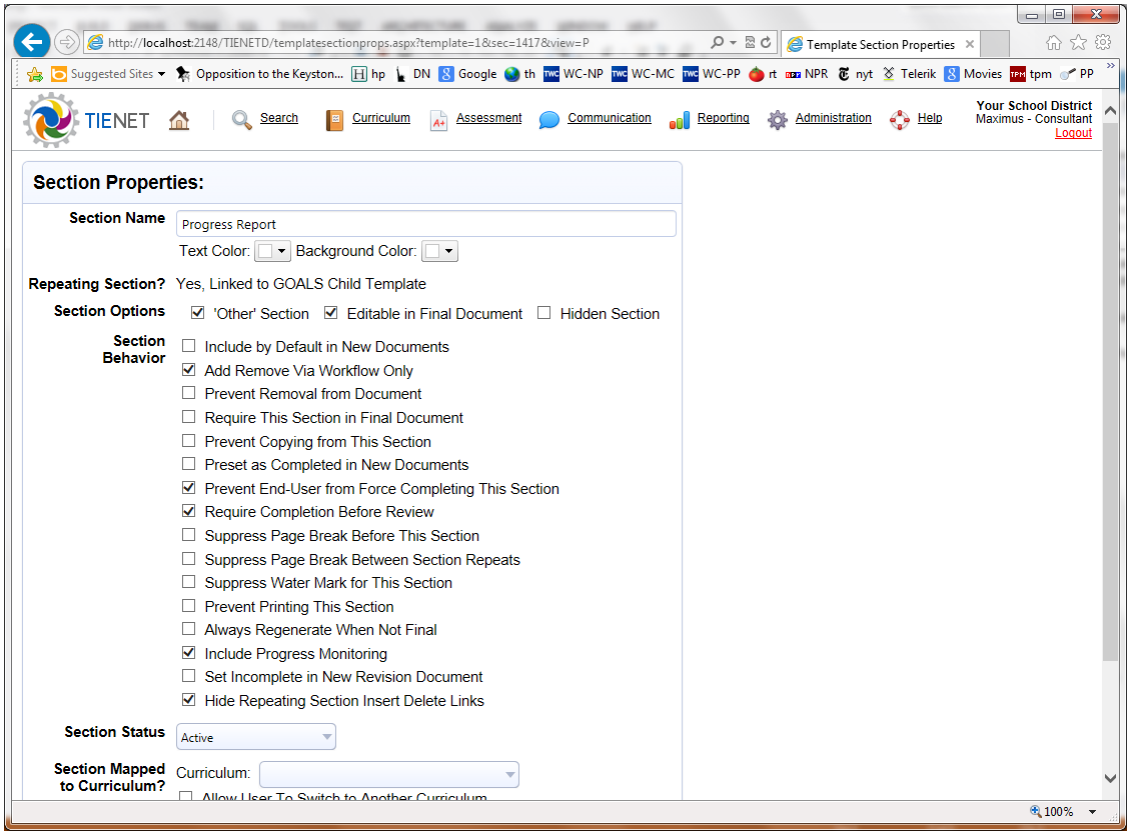
example {Standard:F"Anchor Standard,State Standard"} maps the "Standard" field to both "Anchor Standard" and "State Standard". As a special case, if the level names are prefixed with a tilde character (e.g. F"~Anchor Standard, State Standard "), the curriculum statement labels will be inserted into the long text field instead of the descriptions.

An additional consideration is whether the form needs to dynamically change based on which curriculum is selected. For example, perhaps certain form fields must be presented for one curriculum and different fields must be presented for other curricula. To support this, you will want to have a keyword dropdown on the form that allows the user to pre-select the curriculum on the form itself. The keyword table must provide the curriculum root corresponding to each keyword. This can be done either by making the keywords identical to the curriculum root, or alternatively by adding a character column to the keyword table named "CurriculumRoot" that identifies the curriculum root for each keyword. You will use either #JSIF or auto-postback with #IF statements to dynamically change the form depending on what the user selects from the keyword field. Finally, you will want the user to go directly to the pre-selected curriculum and not be able to switch to another curriculum. To implement this, you introduce the C modifier to the keyword field directive, which instructs the system to bring the user to the pre-selected curriculum when the user clicks the "Select from Curriculum" button. Specifically, you will want to use the C"=" variation, that further instructs the system to not allow the user to change the curriculum in the curriculum popup, and to furthermore prevent the user from bringing up any curriculum until one is selected in the keyword field. An example of the full directive is {ReferenceCurriculum:C"="}.

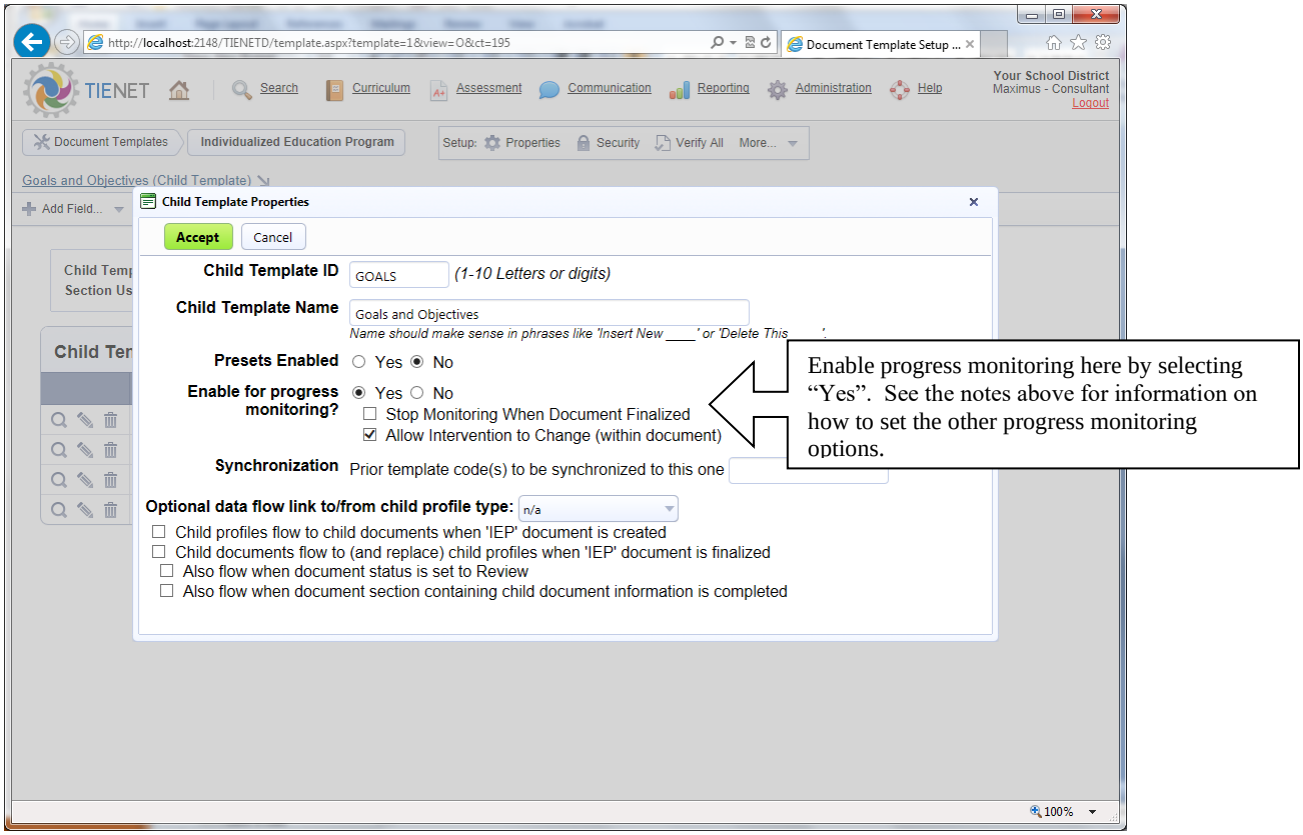
You can also override the label of the "Select from Curriculum" button. To set the label to "Select from Standards" for example, include the directive below at the top of the section: {*<SELECTFROMCURRICULUM>:L"Select from Standards"}.

Configuring Progress Reporting/Monitoring: A progress report is typically implemented as a separate repeating section based on the Goals child template with text boxes to support entry of the narrative progress. There are certain options you will want to set in the section properties for the progress report to make it behave as expected:

- Enable the "Other Section" option in section properties, which essentially establishes the section as an addendum to the main document.
- Enable the "Editable in Final Document" if you wish to allow users to enter progress after the IEP or plan is finalized.
- Make sure the "Hide Repeating Section Insert Delete Links" behavior option is enabled to prevent users from adding or removing goals from the progress report. Goals should only be added or edited from the main goals section.
- Consider turning off the 'Include by Default in New Documents' option and turning on 'Add Remove Via Workflow Only'. Then implement a document action that adds the progress report section when the document is finalized, or some other appropriate time.



To add support for progress monitoring, edit the properties of the goals child template and enable progress monitoring. For IEP goals, the "Stop Monitoring When Document Finalized" check box option should be disabled since progress reporting occurs after the document is finalized. The "Allow Intervention to Change" option should typically also be checked for IEP goals, which allows the end user to modify the intervention over the course of progress monitoring.



Enabling progress monitoring for the goals child template automatically creates the following fields in the child template:

- UseProgressMonitoring (logical): Determines whether progress monitoring is enabled for particular goals. If not enabled, then anecdotal/subjective approaches would be used instead. Not that the "UseProgressMonitoring" field can be made editable for every goal, or set to a default value.
- BaselineDate (date): When the goal line and progress monitoring in general begins.
- BaselineScore (score): The baseline data point that defines where the goal line begins.
- TargetDate (date): When the goal line and progress monitoring in general ends.
- TargetScore (score): The goal data point that defines where the goal line ends.
- TrackStudentErors (logical): Determines whether the user is allowed to track error rates while progress monitoring. This field defaults to false, and as such, can be ignored or utilized in the configuration as the use case demands. It can be exposed to the user as a document checkbox or its default value can be set to true if student errors should be tracked in all cases.

- UseInitialDataPointAsBaseline (logical): Determines whether the initial data point entered during the progress monitoring process should flow back and populate the BaselineScore field. This field defaults to false, and as such, can be ignored or utilized in the configuration as the use case demands. It can be exposed to the user as a document checkbox or its default value can be set to true if the data point should flow in all cases. If exposed to the user as a checkbox, it is important to use #JSIF or auto-postback to hide the BaselineScore field which would otherwise be a required field.

These fields are generally entered by the user in the main goal and objectives section, but then utilized on the corresponding progress reporting/monitoring section to support progress monitoring. Typical basic HTML formatting for these fields in the goal section is shown below:

```
<tr><td><b>Start Date:</b> {BaselineDate}     <b>End Date</b> {TargetDate}
</td></tr>

<tr> <td><b>Baseline Data Point:</b> {BaselineScore}    <b> Goal:</b> {TargetScore}
</td></tr>

{#IFEDIT}<tr><td><b>Initiate Progress Monitoring:</b>
{UseProgressMonitoring:A}<br><br></td></tr>{#ENDIF}
```

To include the progress monitoring chart in the progress report section, you will need to enable the "Include Progress Monitoring" checkbox in the section properties of the progress report. You will also need to insert a {&ProgressMonitoring} directive at the point in the HTML where you want the chart to appear.

In certain circumstances, you may want to stop progress monitoring (stop the collection of data points while continuing to show the already collected data points and the chart). To do this, use a construct as follows:

```
{#IF ProgressMonitoringStopped}
   {&ProgressMonitoring:R}
{#ELSE}
   {&ProgressMonitoring}
{#ENDIF}
```

The {&ProgressMonitoring:R} includes progress monitoring in read only mode only.

Section Extension Child Templates

[New in 23.11.1.0] A document template is limited to about 1024 top-level fields: Normally, this is not a problem, but if a document template has many sections or has been around for a long time, the limit can become a problem. For go-forward model configuration where it is known that this limit will be approached, "section extension child templates" can be used to effectively support an unlimited number of fields that behave like top-level fields. Note that when the platform estimates that there is room for less than 100 additional top-level fields in the document template, it will start showing a warning and a remaining estimate when adding new fields. This warning may be helpful when deciding whether to use section extensions.

At a high level, a section extension child template is a child template that can contain at most one child document/row per top level document, and this is absolutely enforced even by SQL Server itself. Therefore, you can add certain fields to the child template that would otherwise need to be top-level fields with the knowledge that there will be no repetition of the data. Configuring with section extensions will be familiar if you have configured a repeating section as the techniques are very similar, but getting the best results requires a thorough awareness of best practices for using it. At the simplest level, you follow these steps:

1. Add a new child template and set the "Child Template Type" property to "Section Extension (Child Template)".

2. Add fields to this new child template that might otherwise need to be top-level fields.

3. Create a new section that is linked to this child template in the same way that a repeating section is linked to a normal repeating child template.

4. You can use both top-level fields and section extension fields on the section. **Important:** You will find that both top-level fields and section extension fields will be editable in the section. This is different than a repeating section where only fields in the child template are editable.

Beyond these basic steps, be aware of the following practices:

- Section extension fields do not support data flow to top-level fields or child profile fields. If you need fields on the section that require data flow, simply make those top-level fields and they can be editable on the same section as the section extension fields.

- Section actions and HTML format embedded formulas for sections linked to a section extension can freely mix top-level fields and section extension fields in a manner similar to repeating section. The best practice is that fields that need to be referenced from document actions should be top-level fields. While it is possible to reference any field from any section extension from a document action, this access will be less performant than top-level fields.

- Section extension fields can be more difficult to pull into reports. If one is aware that certain fields will drive document logic or reporting requirements outside of the section extension context, make those top-level fields. Less critical fields can be positioned in the section extension child template.

- Just like a normal section that is not linked to a child template, a section linked to a section extension can be configured to have repeating rows and even nested repeating rows using standard repeating child templates. The section extension child template itself does not repeat and therefore cannot be configured for use in repeating rows, nor can it have grandchild templates of its own.

- Two or more sections can be linked to the same section extension child template, which is particularly useful when those sections have a lot of interrelated logic that reference common fields. On the other hand, one can add multiple section extension child templates and link unrelated sections to them.

- A section linked to section-extension fully supports lookup/non-lookup field combinations; however, both the lookup field and the non-lookup field must be from the same template (either both are from the top-level document template, or both are from the section extension child template).

- A section linked to a section-extension fully supports signing areas configured either DocuSign style or the newer digital signature style. In this type of section, if a DocuSign style directive maps a signer field/role back to a field, the mapped field must be from the same template (either both are from the top-level document template, or both are from the section extension child template).

- If one needs to access field values from a section extension when the formula context is the top-level document, there are ways to do that. Since those ways are less performant than accessing top-level fields, this is to be avoided where reasonable (by using top-level fields for those cases in the first place), but nonetheless situations may surface where such access is required. From the top-level context, one can use the syntax X.Y where X is the section extension child template ID and Y is the name of the section extension field. Note that this same syntax is supported when the formula context is the section extension child template itself, and is no less performant, but is not necessary since the name of the field by itself will do.

- Aggregate formula functions like TopOneValueOf, SumOf, MinOf, and MaxOf are not supported for section extension child templates. The X.Y syntax described in the previous point achieves the same goal for section extension child template as TopOneValueOf does for repeating child templates. However, the Exists function is supported for section extension child templates and in some cases will be more performant than the X.Y syntax. For example, EXISTS(X, Y='Value1' AND Z='Value2') is more performant than X.Y = 'Value1' AND X.Z='Value2'). The reason for this is that the EXISTS function reaches into the section extension to check values only one time.

Additional, more subtle, nuances are as follows:

- Manual copying of document sections from another document by the end-user works the same whether a section is not linked to a child template or the section is linked to a section-extension child template. For example, consider the case where two sections share the same section extension but have different fields from the section extension associated with them. If the end-user copies only one of the two sections, only section extension field values associated with the section copied by the end user will copy.

- If the end-user fills in section extension fields, but then the sections associated with the section extension are removed from the document before the document is finalized, all fields in the section extension will effectively reset their values at the time the document is finalized (by automatically removing the one child document/row for the section extension child template).

One may wonder when to use section extensions versus the 'Compress Long Text Fields' document template option which also is useful given the approximate 1000 field limitation. The 'Compress Long Text Fields' document template option is more useful for document templates that have been in production for some time and therefore cannot easily be rearchitected to use section extensions. Section extensions are useful for future and upcoming configuration work. Note that there is no mechanism to convert existing fields and child templates into section extension child templates. Such a mechanism was considered, however, the architecture of such functions as digital signature, profile archive and student transfer envelope rely on fields remaining in the same template and not being moved to other templates.

Storing Uploaded Files Associated with Profiles & Documents

There are four approaches for storing uploaded files associated with documents and/or profiles. Each approach has strengths and weaknesses as described below:

1. **File-Based Documents:** Each file-based document is essentially a collection of one or more files. This is the simplest approach to implement since it does not require any configuration other than to assign a few privileges. . However, since it has a number of weakness detailed below, other approaches should be used instead when building state models.

    - Any user who can view a particular profile's documents via the 'Access Documents' privilege can access any file-based documents for the profile. If more fine-grained security is needed, one of the other file storage options should be implemented.

    - A user with access to a file-based document can access the individual files directly from a profile's document list.

    - Users with the 'Maintain Own File Based Documents'privilege will be able to create file-based documents and upload files into them. Users with the 'Edit File Based Documents From Other Users' will be able to edit file-based documents created by other users except for those that have been set to final. Users with the 'Unfinalize File Based Documents' privilege will be able to set final file-based documents back to draft or review mode, and so this privilege should be restricted to system administrators.

- **Weaknesses:** 1) All such documents are listed in a generic "Other Documents" category rather then a specific category, 2) security is all or nothing in the sense that you cannot allow a security group to access certain file-based documents and not others. Note that the "File Attachments Only Document Template" approach describe below was specifically designed to address these weaknesses.

2. **Document File Attachments:** With this approach, files are uploaded and attached to documents from a document template.

    - File attachments are enabled in the configuration of a document template by setting the "Allow File Attachments" option in the document template properties.

    - Any user who can access a document can also access its file attachments. However, the ability to upload and attach files is controlled by three document template security rights, namely "Attach Files", "Edit Files Attached by Others", and "Attach Files to Final Documents".

    - Users can see and access the individual files directly from the profile's document list as well as from within the document.

3. **File Attachments Only Document Template:** This approach combines some of the best features of the two previous approaches and involves configuring a document template that will appear on the dropdown menu for creating a new document. But the subsequent user interface will essentially mimic that for uploading a file-based document.

    - To create a document template for file attachments only, set the "Use for File Attachments Only" document template property while creating it.

    - The security rights are identical to that for document file attachments.

    - Document templates of this type can be assigned a document template category. Hence documents of this type can appear in a specific document category other than the 'Other' default category.

    - In advanced applications, you may wish to add fields to the document template that data flow from the profile in order to "capture" the values of those fields at the time the document was created. Additionally, you can even create guided actions that get triggered when such documents are finalized. In short, documents of this type can participate in workflow.

    - It is possible to write reports on documents of this type, whereas there is no mechanism for writing reports on simple file-based documents.

4. **File Fields:** You can add one or more file fields to the data dictionary of a document or profile. Each file field stores both the file name and binary image of an uploaded file. This approach has both advantages and disadvantages as shown below.

    - You can precisely position each file field on a specific section of a document or profile.
    - File fields are very useful when there are multiple independent requirements to upload external documentation for a given PowerSchool Special Programs document. For example, if the user is required to upload one file into one section of a document, and another file into another section of the document, the only reliable way to implement this is to use file fields. With file attachments, you could not be sure what file is associated with what section.
    - By default, files stored in file fields do not appear in a document list like file attachments do, in which case the only way to access a file stored in a file field is to navigate to where the file field is on a specific section. However, file fields in document templates have an optional checkbox property labeled "Show File as Read Only File Attachment". When this property is enabled, the file stored in the field will also show as a file attachment just like any other file attachment with the exception that the file can only be modified or removed via the file field. This property is only available in the context of a top level document template and not in child templates.
    - File fields can be configured as required fields.
    - When a file field is referenced in a formula, it gives the file name as a character value. A file field be tested within a formula to see whether it has a value yet (i.e. a file) using "FileFieldName IS EMPTY".
    - When using file fields to allow the user to upload more than one file there are limitations to be aware of, and related best patterns and practices as described below.

Use File Fields to Support Uploading Multiple Files:

When there is a need to use file fields to allow uploading multiple files in a document, the standard solution is to add the file field to a child template. The first challenge here is that the file field will only be editable after a child document is saved or otherwise stored in the database. This is not an issue when the child template of the file field is the basis of a repeating section but can be a problem when the file field is in repeating rows, especially in the fast add mode. The issue can also be relevant when the file filed is in a child profile. One technique is to simply inform the end user of this using markup like the examples below:

The example below only works in profiles and child profiles.

```
{#IFNEW}
    Files can be attached after saving this item
{#ELSE}
    {FileField}
{#ENDIF}
```

The example below will work in repeating rows under the condition that the repeating rows have editable fields other than the file field.

```
{#IF IDT IS EMPY}
    Files can be attached after saving this item.
{#ELSE}
    {FileField}
{#ENDIF}
```

To implement repeating rows that only present a file field to the user, follow this approach:

1. First, note that this approach will not work in the file field will be put in a grand-child template to be used in repeating rows nested within repeating rows. An alternative technique to be described later in this section may be applicable to that scenario.
2. Create a child template with the file field plus an additional date/time field with a recommended name of FileDateTimeAdded with a default value of CurrentDateTime() and optionally marked as "Internal Value" in its properties. Set up the repeating rows with the "Input Sort Formula #1" set to the data time field.
3. Set the repeating rows to fast add mode with no minimum number of rows, but do not configure the {^EDITCONTROL} directive in the footer.
4. At this point, the repeating rows will behave like fast add but will not have any way to add rows. You can now add that yourself by creating a section action (perhaps named "Add File") with a trigger type of "Modify Data Upon Save" and the "Link to Button Directive" option. Place your own button next to the repeating rows to add a file via a {*Add File:L"Add File"} directive. You can optionally specify a CSS class for the button directive with the C modifier.
5. Consider how to handle when the user does not supply a file for a row that has been added. Such empty rows can be removed via a document action when the document is finalized. The file field can optionally be made required, which will force the user to explicitly delete unused rows.

In some scenarios where the requirement is to allow the user to upload up to N files at a particular point on a form, it may make sense to consider the following pattern that leverages JSIF and a finite number of file fields numbered 1 through N in the same record. If you need to attach 1-N

files directly to a child profile, this will be the only option. It may also make sense if you already have repeating rows using fast add but need to allow attaching 1-N files to each repeating row. In this case, the previously described approach will not currently support putting the file field in a grand child template configured for repeating rows. The following configuration in a child profile form will provide the end user with an experience that as soon as they upload a file, the next file field appears up to four files. It also nicely handles the user deleting any one of the four files later.

```
{FileField1:!}
{#JSIF FileField1|FileField2,tag=span}{FileField2}{#ENDIF}
{#JSIF FileField2|FileField3,tag=span}{FileField3}{#ENDIF}
{#JSIF FileField3|FileField4,tag=span}{FileField4}{#ENDIF}
{-FileField2,FileField3,FileField4}
```

If you have repeating rows where it is necessary to allow the user to attach up to N files to each repeating row, the following variation of the above behaves nicely:

```
{#IF IDT IS  EMPTY}
    Files can be attached after saving this item
{ELSE}
    {FileField1:!}
    {#JSIF FileField1|FileField2,tag=span}{FileField2}{#ENDIF}
    {#JSIF FileField2|FileField3,tag=span}{FileField3}{#ENDIF}
    {#JSIF FileField3|FileField4,tag=span}{FileField4}{#ENDIF}
    {-FileField2,FileField3,FileField4}
{ENDIF}
```

Self-Service Documents

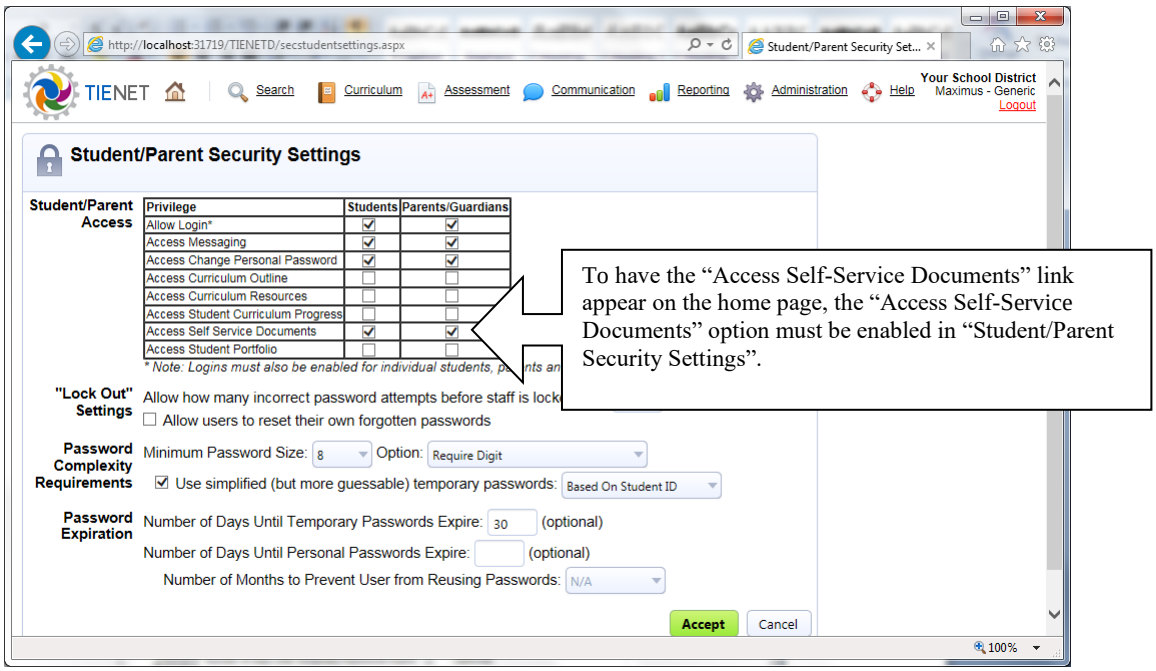
The self-service documents feature can be used in two contexts:

1. It can be used to grant student, parent and guardian users view access, and potentially even edit access, to the corresponding student documents without allowing access to other students' documents.

2. It can also be used to grant staff access to their own documents without allowing access to documents of other staff. However, as of Version 14.1, there is a more robust alternative to staff self-service documents. See the "Document Owner Security" section on page 243 for more information on this recommended alternative.

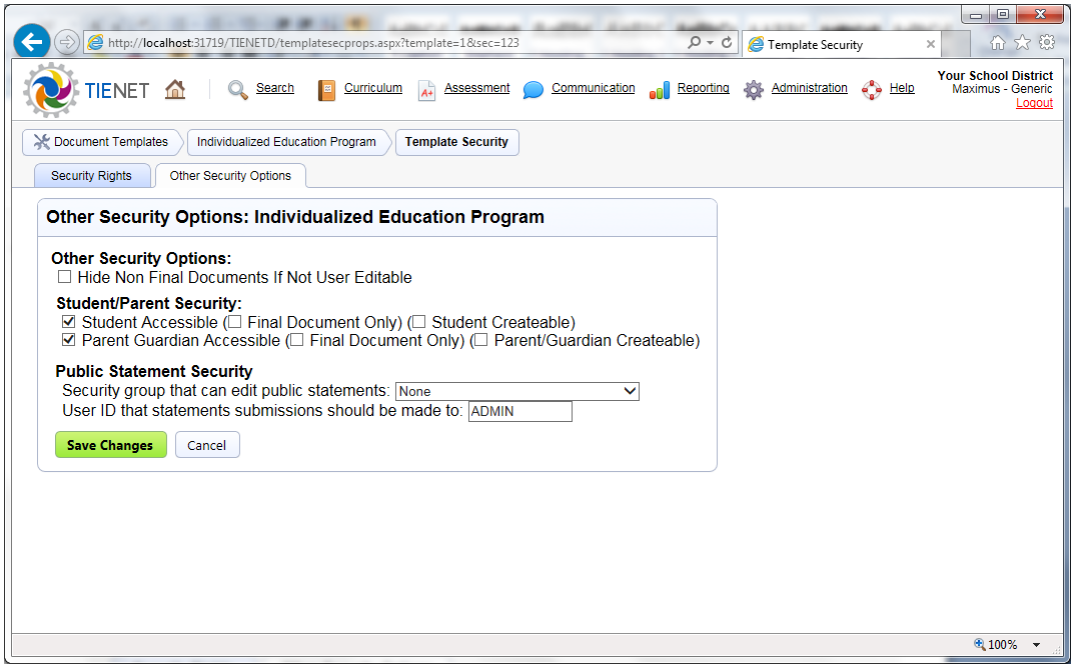
To configure one or more student document templates for self-service access by students, parents and/or guardians, follow the steps below:

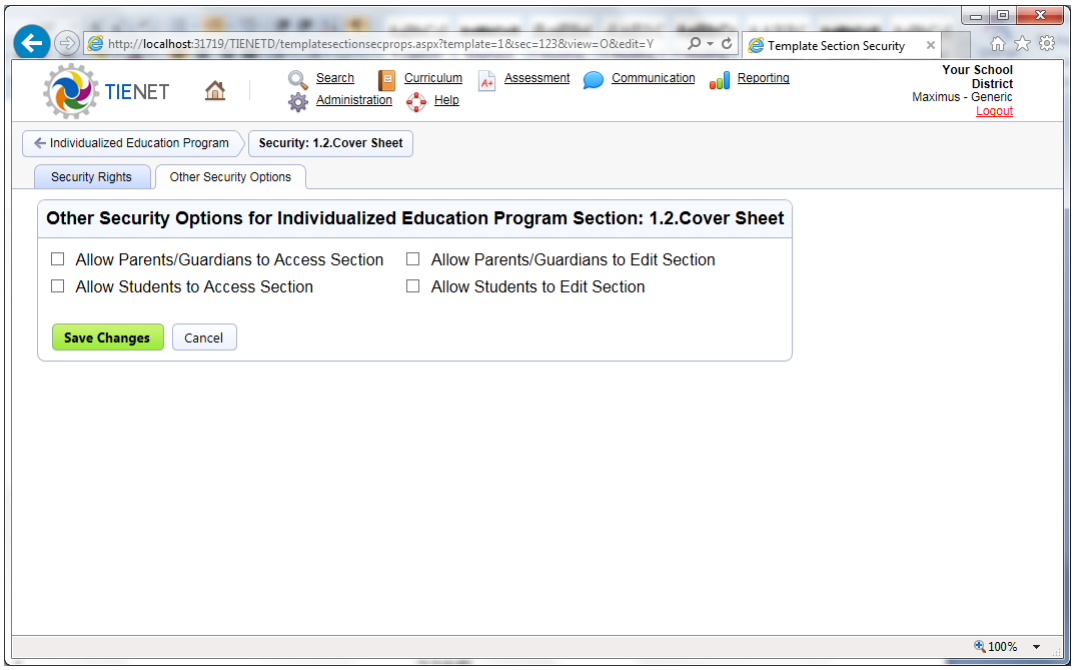
1. First you will need to make sure that the "Access Self-Service Documents" link is on the home page of students, parents and/or guardians. Select "Security" from the "Administration" menu. Click the "Student/Parent Security" tab and then click "Security Settings". Make sure the "Access Self-Service Documents" checkbox for students and/or parents/guardians.



2. In each document template for which you want to enable student and/or parent/guardian access, go the document template's main security screen, and access the student/parent security options on the "Other Security Options" tab as shown below. However, you will also need to enable access to each individual section of the document template as shown in the second screen shot below.

The Student/Parent securityoptions are on the main document template security screen in the "Other Security Options" tab.





3. Of course, you will need to activate the logins of the students and/or parents/guardians for them to be able to access self-service documents. This is described in the system administration guide.

Self-Creatable and Self-Editable Documents: The security options for self-service documents allow for a scenario of students, parent and/or guardians editing documents or even creating documents. Here are some tips for these scenarios:

- Documents, when editing in self-service mode, only support section actions with trigger types of "Modify Data Upon Save" and "Rollback Saving Section". Document actions are not supported in self-service mode. You can implement a "Modify Data Upon Save" section action to queue the document to be set to final. Documents queued this way do not change status immediately, but rather change status during non-peak hours by the PowerSchool Special Programs background service.

- Note that users of self-service documents can never finalize their own documents. But it is possible to establish a document action to automatically queue a document to be set to final, or to use "mass finalization" approaches at the end of the school year.

- If allowing self-creation of documents, you may wish to establish some controls over the user's ability to create multiple documents of this type all at once. The "Require Previous Document to be Finalized before New One Created" and the related "Within the Same School Year Only" template options are the easiest ways to establish control.

Staff Self-Service Documents: The procedure to configure staff self-service documents is almost exactly the same as for student self-service documents, but with two key differences. The first is that such document templates would be for the "Staff" profile type and any documents created would be staff documents. The other difference is that to have the "Access Self-Service Documents" link appear on the home page, users must be granted the special access privileges labeled "Access Self-Service Documents". But as stated earlier in this section, the "document owner security" functionality is a better approach to these scenarios.

Document Template Translations

The main focus of this section is how to translate a document template so that it supports one or both of the following two modes.

Translation: Documents are written in the default language (e.g. English) and then later translated into one or more other languages.

Localization: Documents are written directly in one or more languages other than the default language. Once the document template itself is translated, individual documents are not translated because they are written directly in the intended language.

Before translating document templates, it should be kept in mind that there is a broader checklist of items that must be completed related to this. These are as follows:

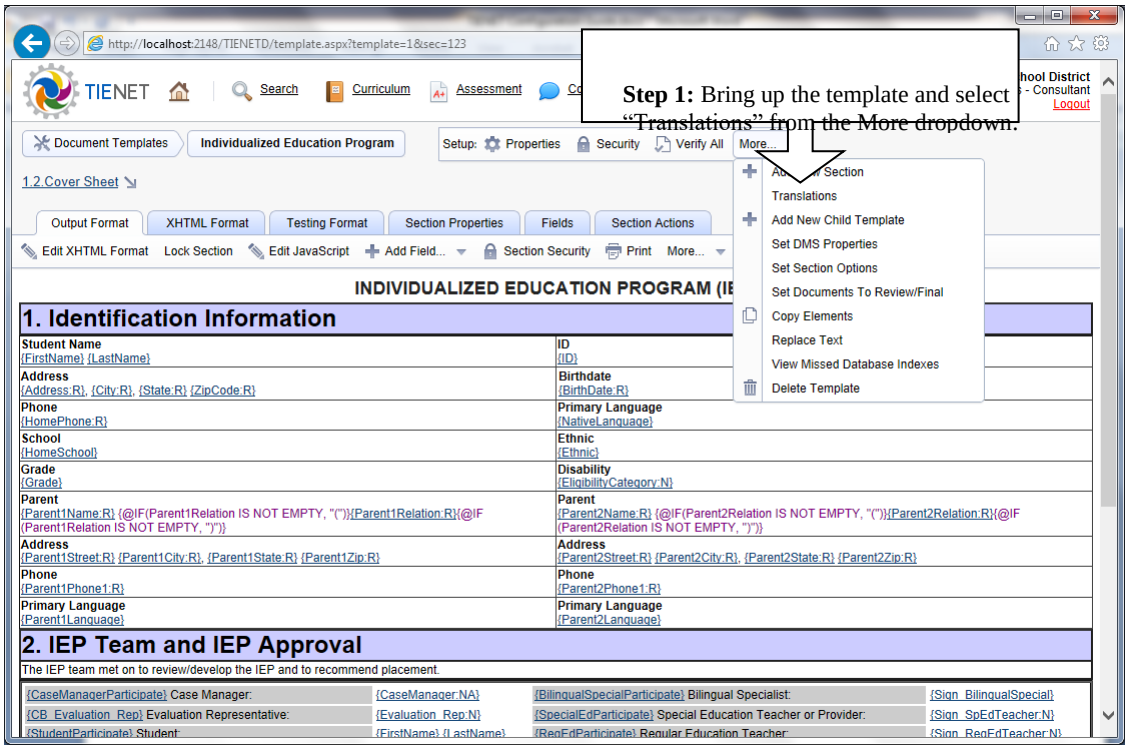
If machine translations are to be supported, the PowerSchool Special Programs servers must be configured with a Google Language Services API account as described in the "Installation and Management Guide".

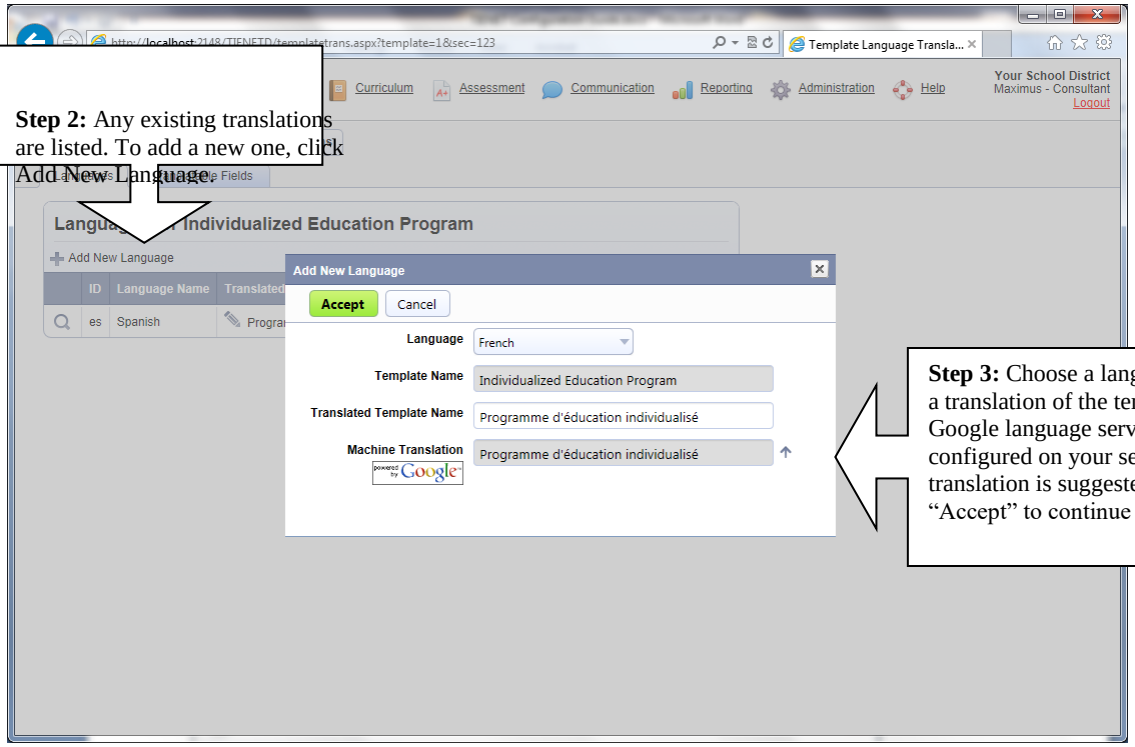
If a goals and objectives curriculum will be used, that should be translated as covered in the "Curriculum Development Guide".

The translated document templates will typically have dropdown menus that utilize text from keyword tables. For example, if the text values are currently being obtained from a keyword table field named "Description", a new field named "Description_xx" must be added to the keyword table and populated to contain the translated values. In the new field's name, replace the xx suffix with the ISO language code for the required language (e.g. es for Spanish).

Translation security privileges and document template rights must be assigned to those security groups that will be performing translation on an ongoing basis.

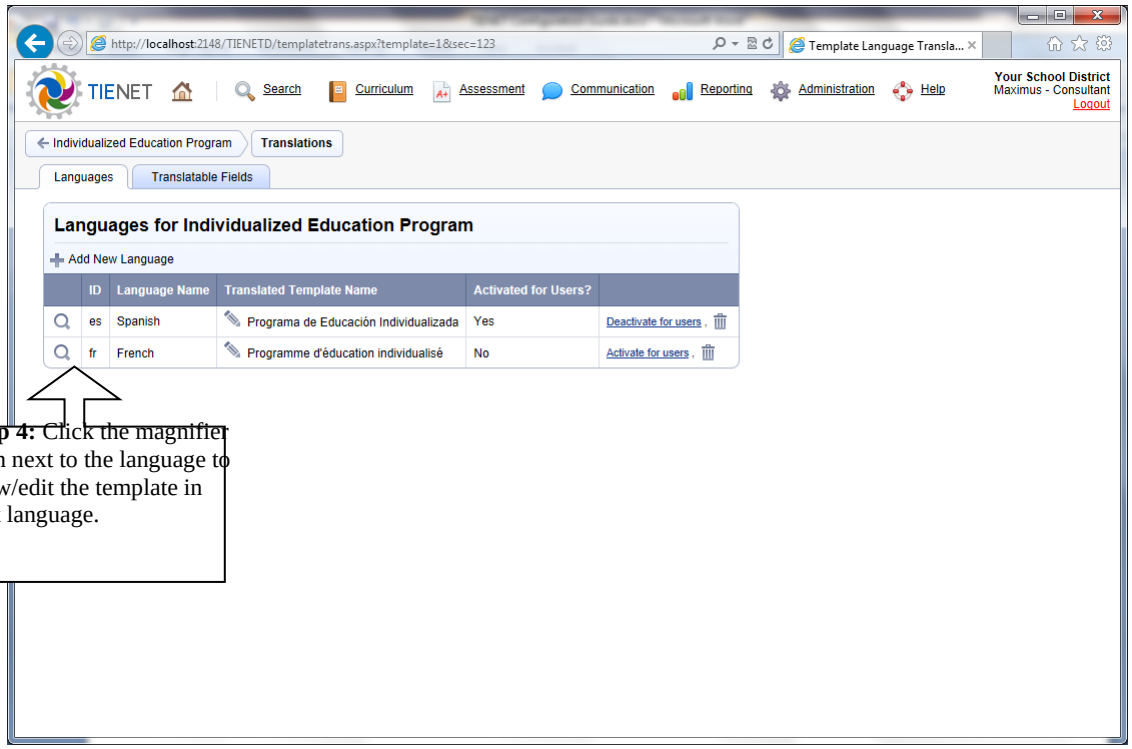
To set up a translation for a document template, follow the steps below:





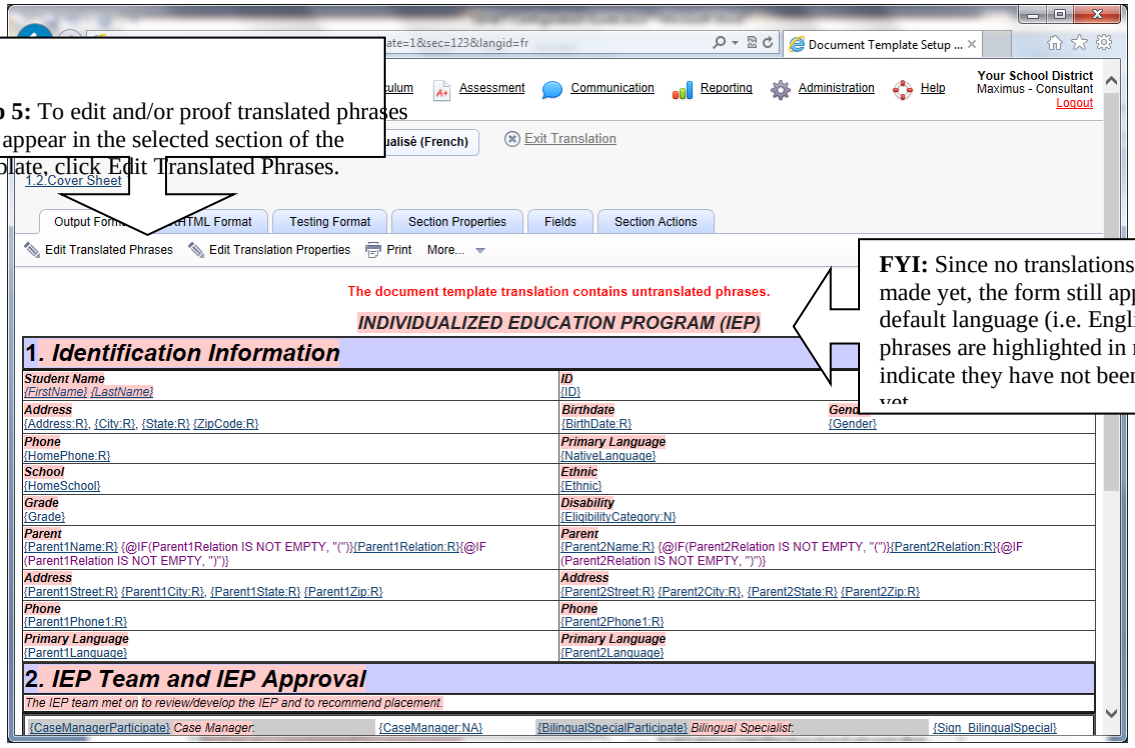
Step 2: Any existing translations are listed. To add a new one, click Add New Language.

Step 3: Choose a language and also provide a translation of the template name. If Google language services has been configured on your server, a machine translation is suggested for you. Click "Accept" to continue and add the language.



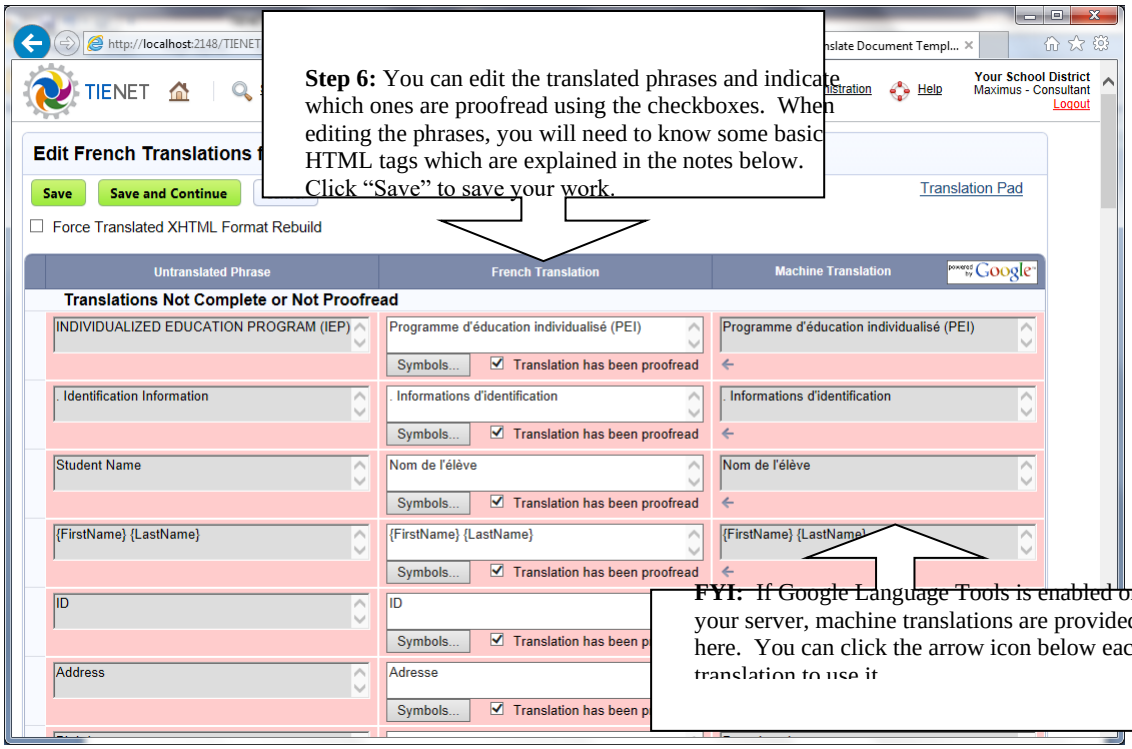
Step 4: Click the magnifier icon next to the language to view/edit the template in that language.

Step 5: To edit and/or proof translated phrases that appear in the selected section of the template, click Edit Translated Phrases.

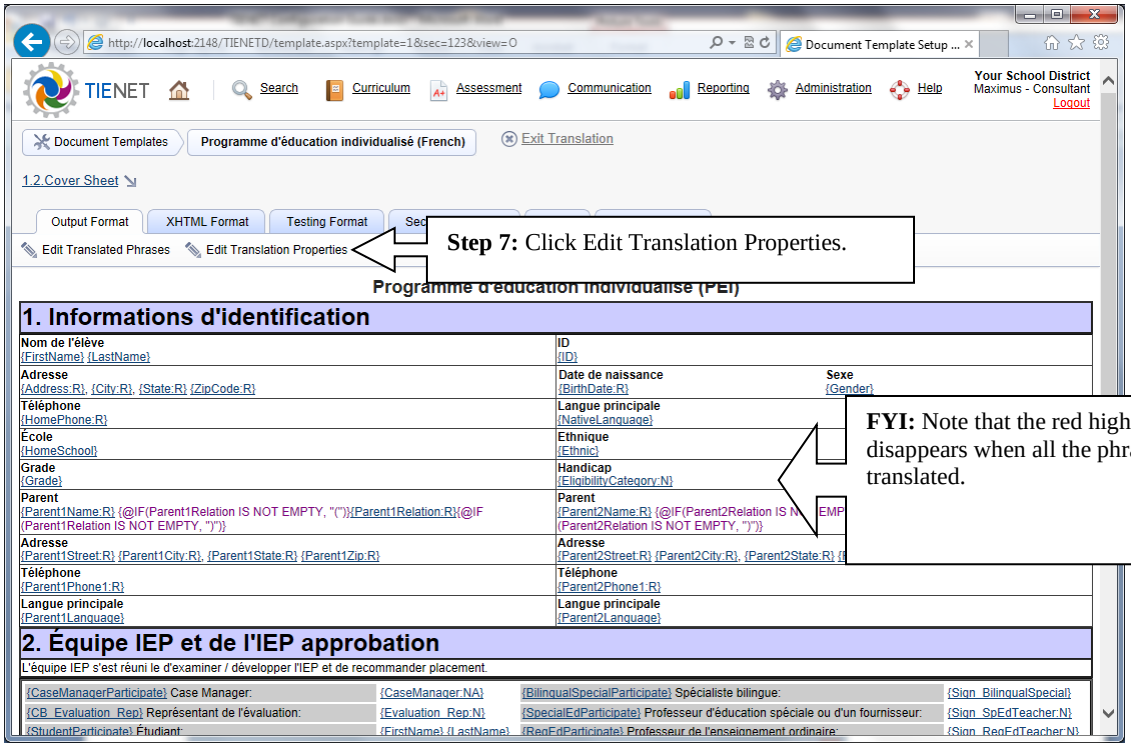


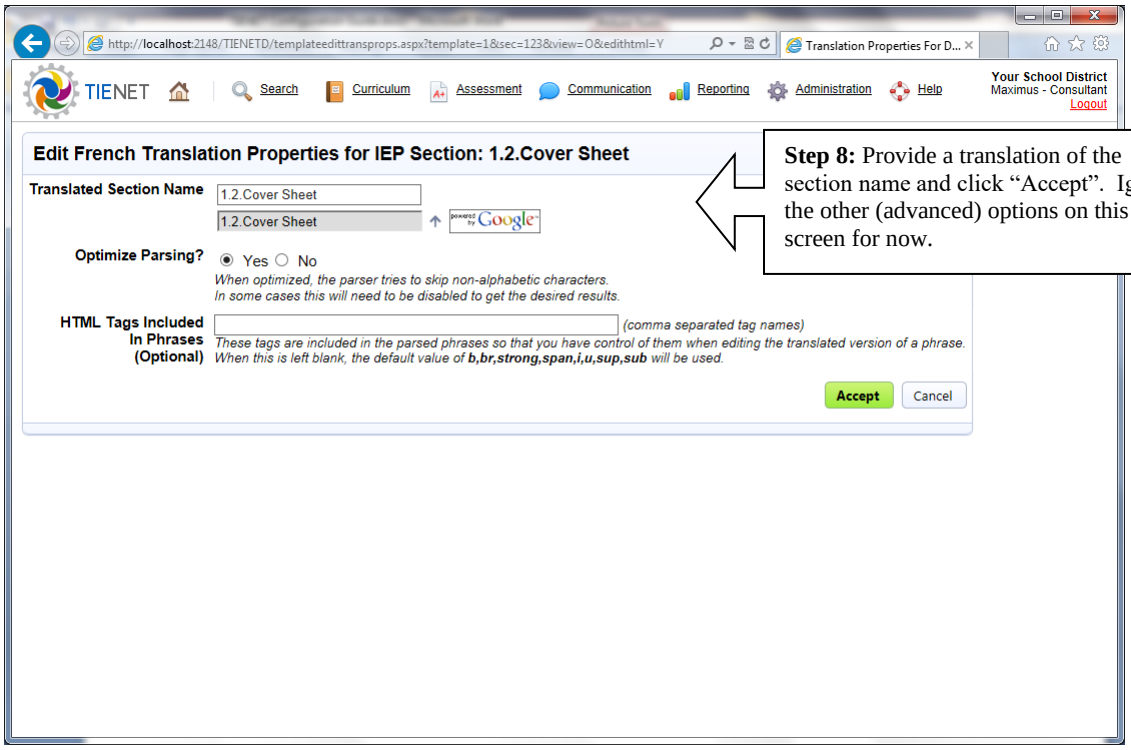
FYI: Since no translations have been made yet, the form still appears in the default language (i.e. English) but phrases are highlighted in red to indicate they have not been translated yet.

Step 6: You can edit the translated phrases and indicate which ones are proofread using the checkboxes. When editing the phrases, you will need to know some basic HTML tags which are explained in the notes below. Click "Save" to save your work.



FYI: If Google Language Tools is enabled on your server, machine translations are provided here. You can click the arrow icon below each translation to use it.





FYI – About HTML Tags: You will notice translated phrases that contain HTML tags. This is necessary to ensure that phrases include enough text to have a sufficient context for translation. Common tags to see

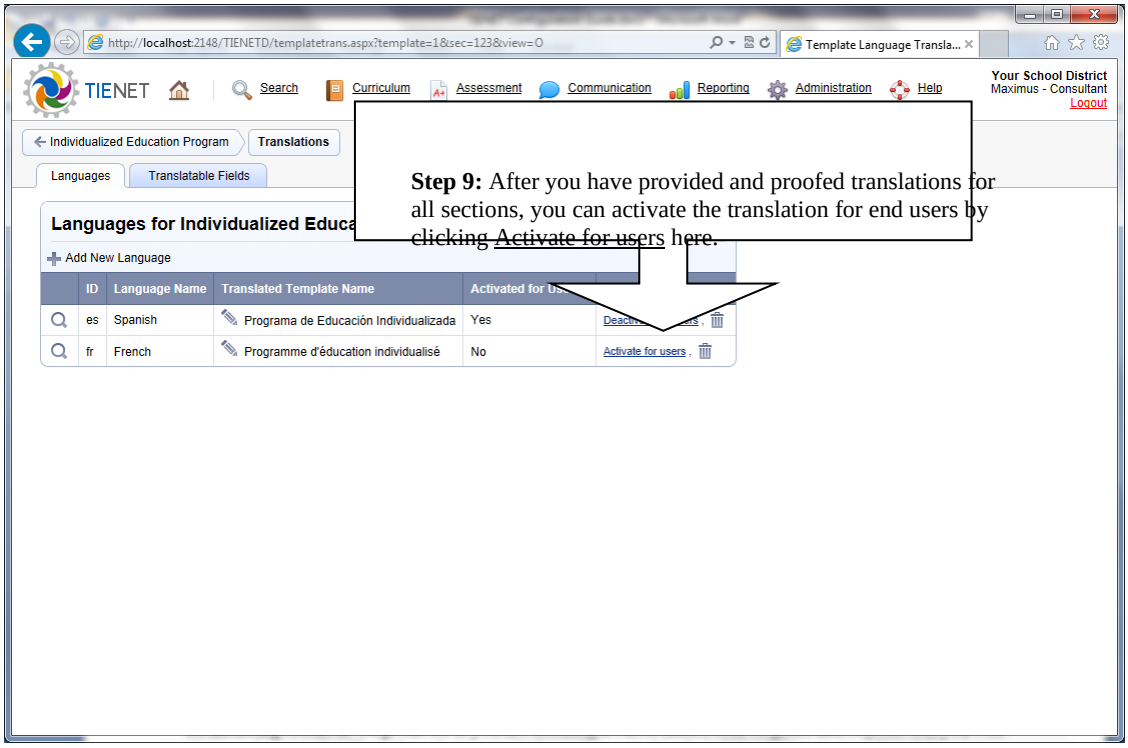
in a translated phrase are given below. You can control exactly which tags are included in translated phrases by clicking the Edit Translation Properties link available when viewing a template section.

text	Text is in bold face.
 	Line break
text	*Text is in bold face.*
text	Text span for applying style
italicized	*Italicized text*
<u>text</u>	Underlined text
^{text}	*Superscript text*
_{text}	*Subscript text*

FYI - Designing Templates for Better Translation – It is possible to design the non-translated HTML format of a section in a way that either facilitates or hinders translation. For example, if the intention of the non-translated format is to show the sentence "That boy is smart." with one word per line, either of the formats below could be used to achieve this. However, using the first format, the sentence will be translated as a complete sentence, while using second format; each individual word will be translated by itself (which will not result in an acceptable translation).

That
boy
is
smart.

That boy is smart.



Re-Translating a Modified Template: When a template form is modified (in the default language), you will need to provide translations for any new phrases introduced and you can also retire translations of any phrases that are no longer in se. If you select the language as described earlier and look at the form, you will see the new phrases in read. Follow the same steps as shown previously to provide translations for the new phrases. On the phrases translation screen, you will find that the phrases are generally sorted into up to four categories: not translated, translated but not proofread, already proofread, and unused. There is a button to eliminate the unused phrases.

Translating Goals & Objectives: If you are utilizing a curriculum (bank of goals and objectives), you will also need to translate that. Translating curriculums is covered in the "Curriculum Development Guide". You should also review the material on translating documents in the "Special Programs Users Guide".

Localizing: At the beginning of this section, it was explained that "localizing" allows documents to be written directly in one or more languages other than the default language. To allow this with a document template, you need to do the following three steps:

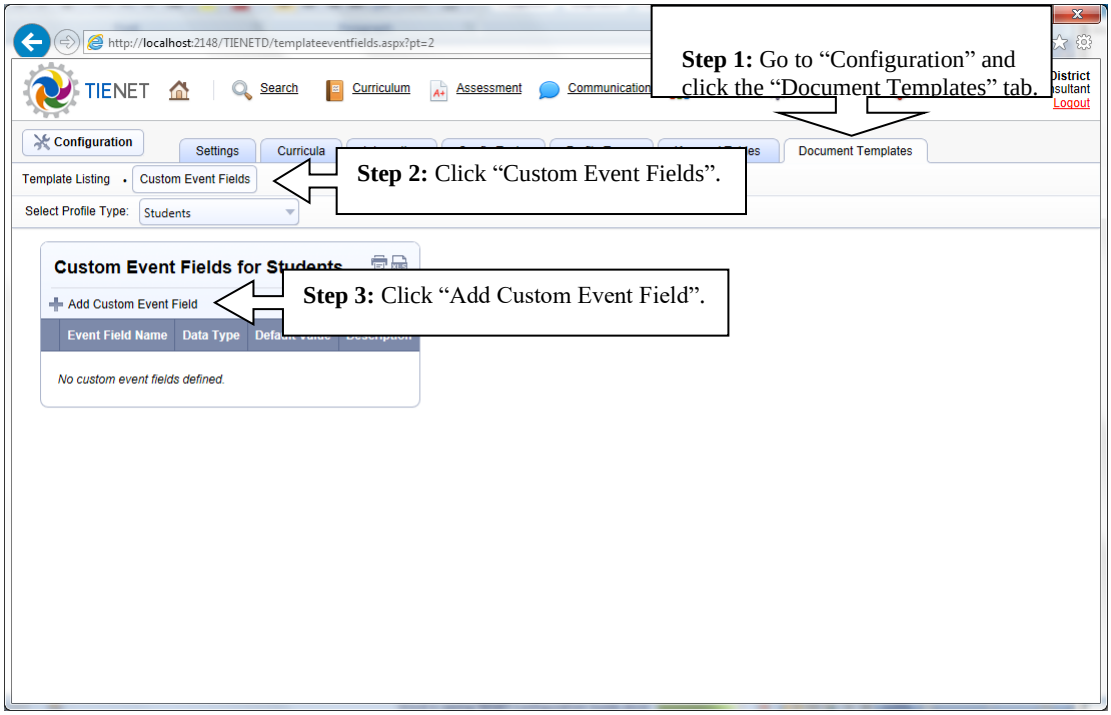
Translate the document template as described previously.

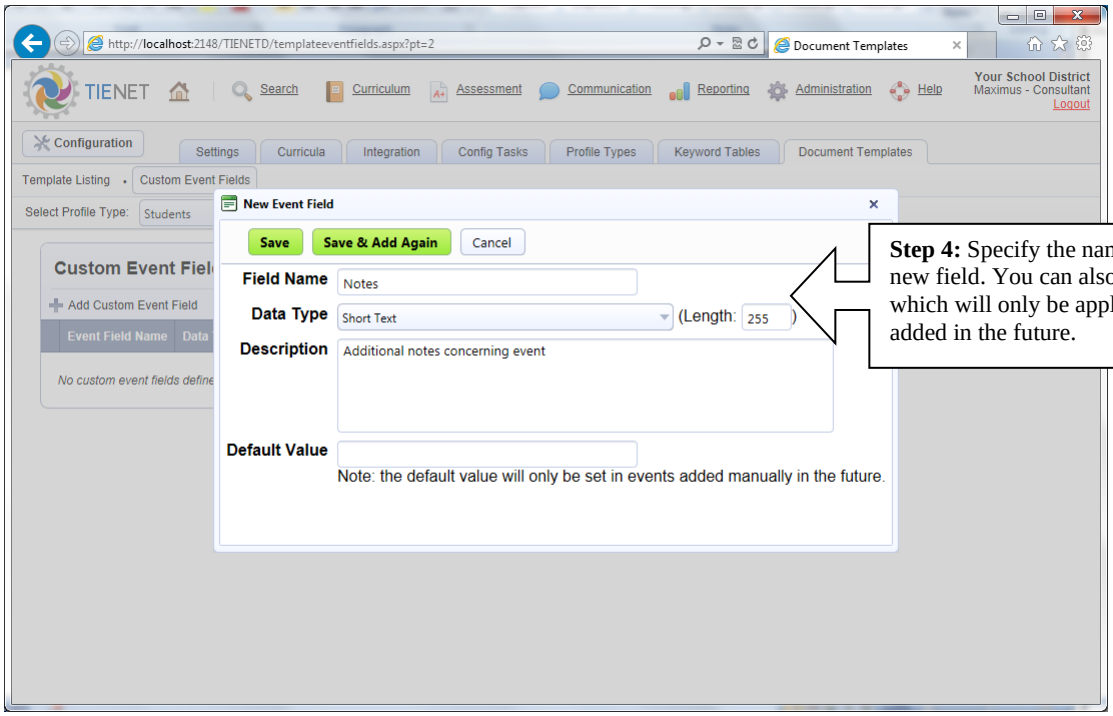
Go the document template properties, scroll down to "Language Options" and enable the "Allow writing documents directly in language(s) into which template has been translated?" option.

You will also need to provide translations for the following other aspects of the document template configuration: document actions, section actions, child template name, and the data dictionary (optional). It is also important not to forget to provide translations for the keyword tables as described earlier.

Customizing Document Events

Custom fields can be added to document events as follows:





FYI – Accessing event data from PowerSchool Special Programs **formulas:** You can apply the aggregate functions (EXISTS, COUNT, SUMOF, MINOF, MAXOF, TOPONEVALUEOF) to event data from several different contexts as identified below. The filter formulas for the aggregate functions can reference built-in fields (Student, EventDateTime, SourceUserIDName, Subject, Description, Automatic) as well as custom fields.

To access student event data from a student formula, you can use a format like EXISTS(events_, Automated=false and CallType=Phone). Note that events_ (with an underscore suffix) is a special keyword that refers to the event data in this context. This particular example would select students for whom there exist events that are manual (Automated=false) and for which CallType (a hypothetical custom field) is equal to Phone.

To access event data from a document-based formula, you can use a similar format. In this case, you can use "events_" to refer to any events for the student the document is for. Or you can use docevents_ to refer to only events linked to the document.

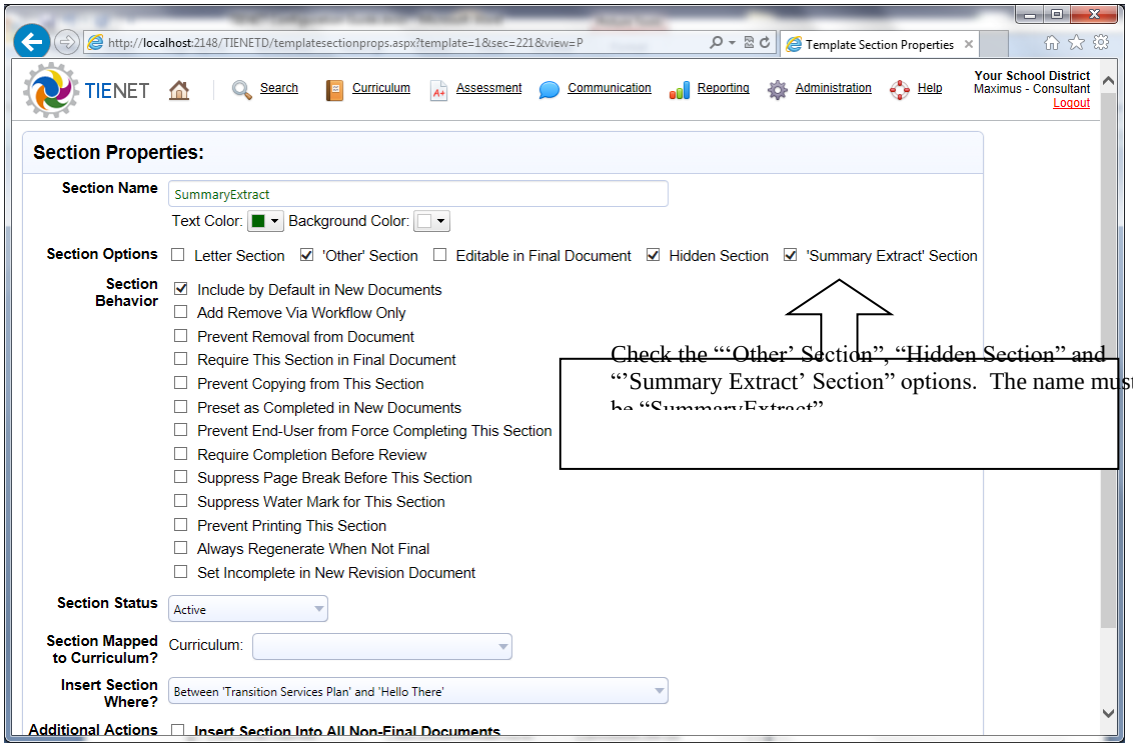
Customizing the Documents List View with Summary Extracts

The documents list view (i.e. Student Documents List) can be customized by configuring "summary extracts" for document templates that pull a key data elements out of the documents and into the student documents list view. Such customized views are linked to specific document template categories, and so the views appear when the student documents list is filtered to the corresponding document template category.

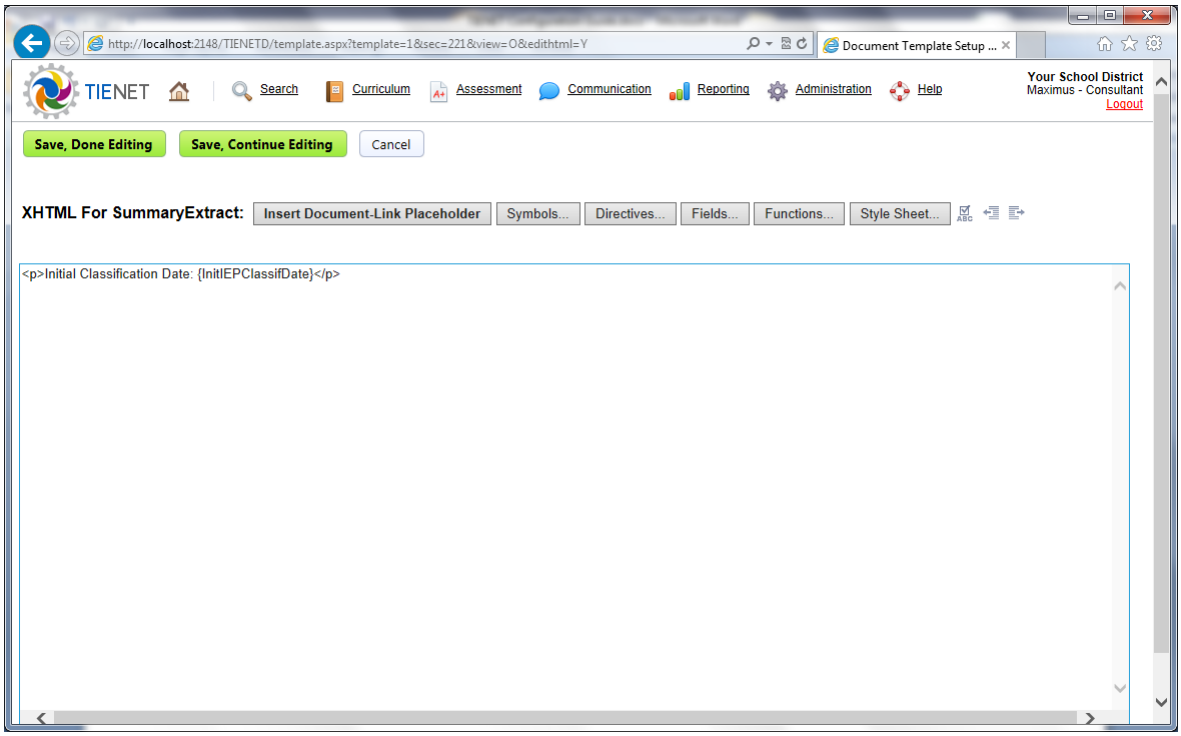
Since this feature is linked to a specific document template category, you will want to follow the steps below for each document template in the category that will have the customized view.

1. On the document template properties screen, enable the "Show Summary Extract" option.

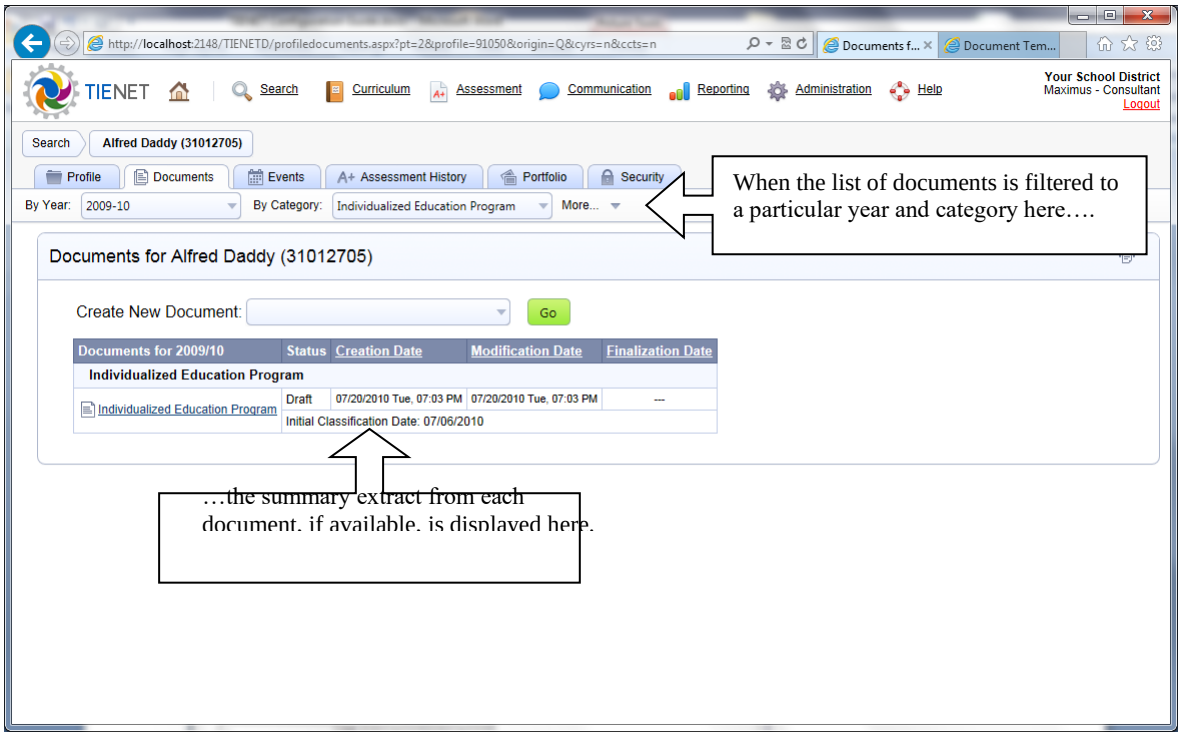
2. Create a new section named exactly "SummaryExtract". Enable the "Sub-section of 'Other'", "Is Hidden" and "Section is summary extract" options, and link in any data fields that you want to appear in the customized view. It is only practical to include a small number of key fields from the document.



3. Define the HTML layout that will be included in the customize documents list view. Note that the layout should be relatively small in size, otherwise it won't be practical to display inside the documents list.



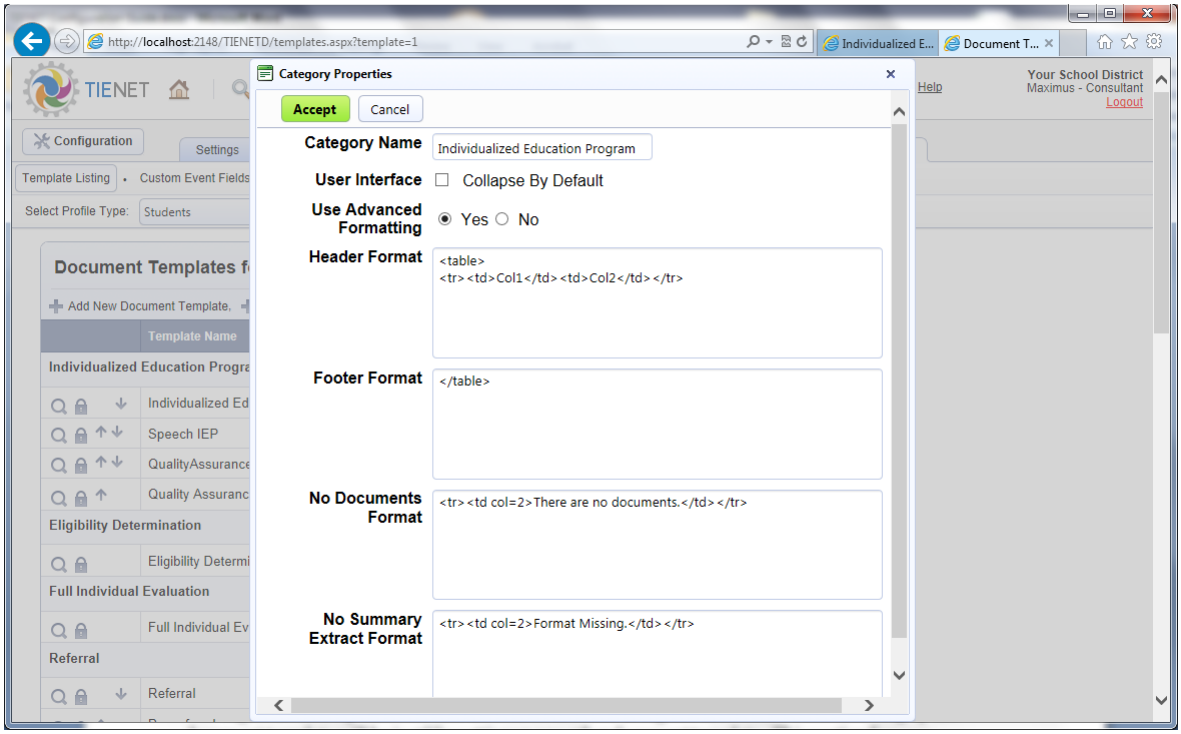
4. Test out the customized view by bringing up a (student) documents list with documents of this category, and then filter the documents list to that category.



Advanced Customization: In the example above, the customized layout is included in the existing document list view. It is also possible to take over the formatting of the documents list entirely. The approach is very similar to that used to configure the layout for a repeating rows grid. The main steps are:

Define the summary extract section as previous, but with advanced customization, be sure to include the pseudo-tag {&DocumentLink} as a placeholder for where the document link will appear.

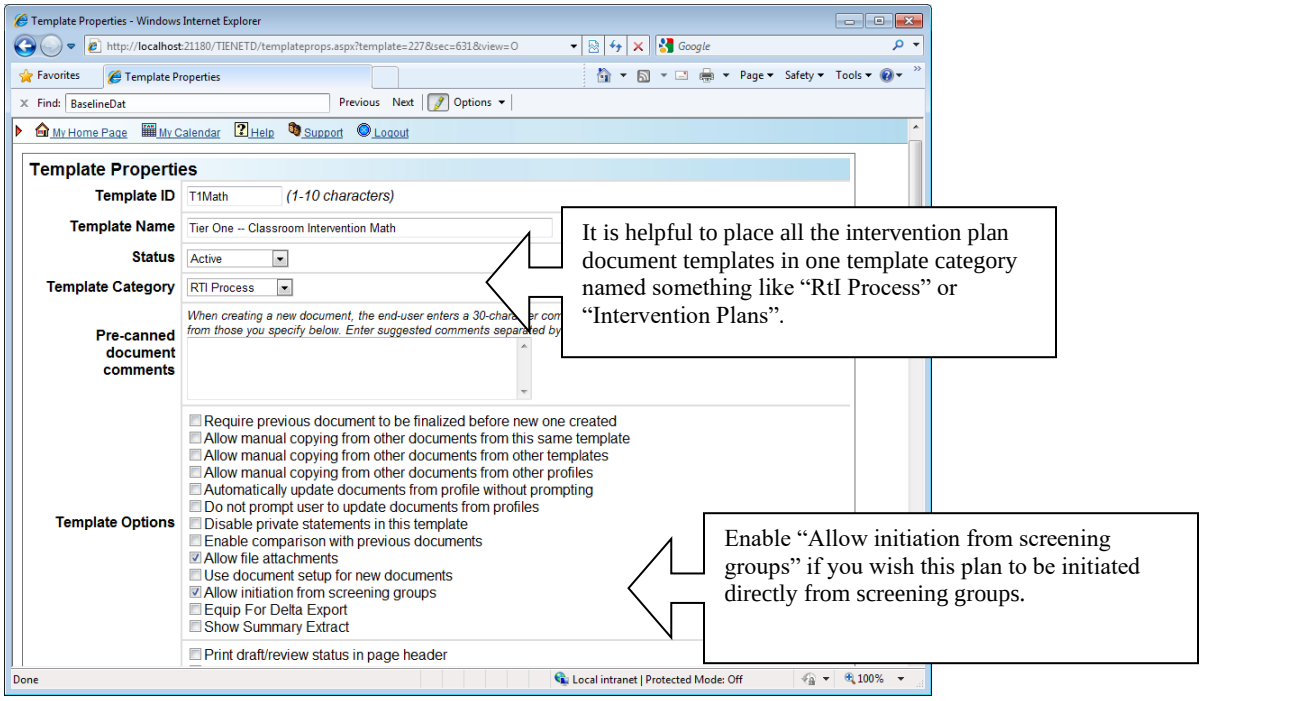
Edit the properties of the document template category, turn on advanced formatting, and define a format for the header, footer, and placeholder formats for when the category has no documents or when a specific document template in the category has no summary extract. A tabular view can be implemented by putting "<table><tr><th>header</td></tr>" in the header, </table> in the footer, and making the summary extract for each document template an HTML table row.

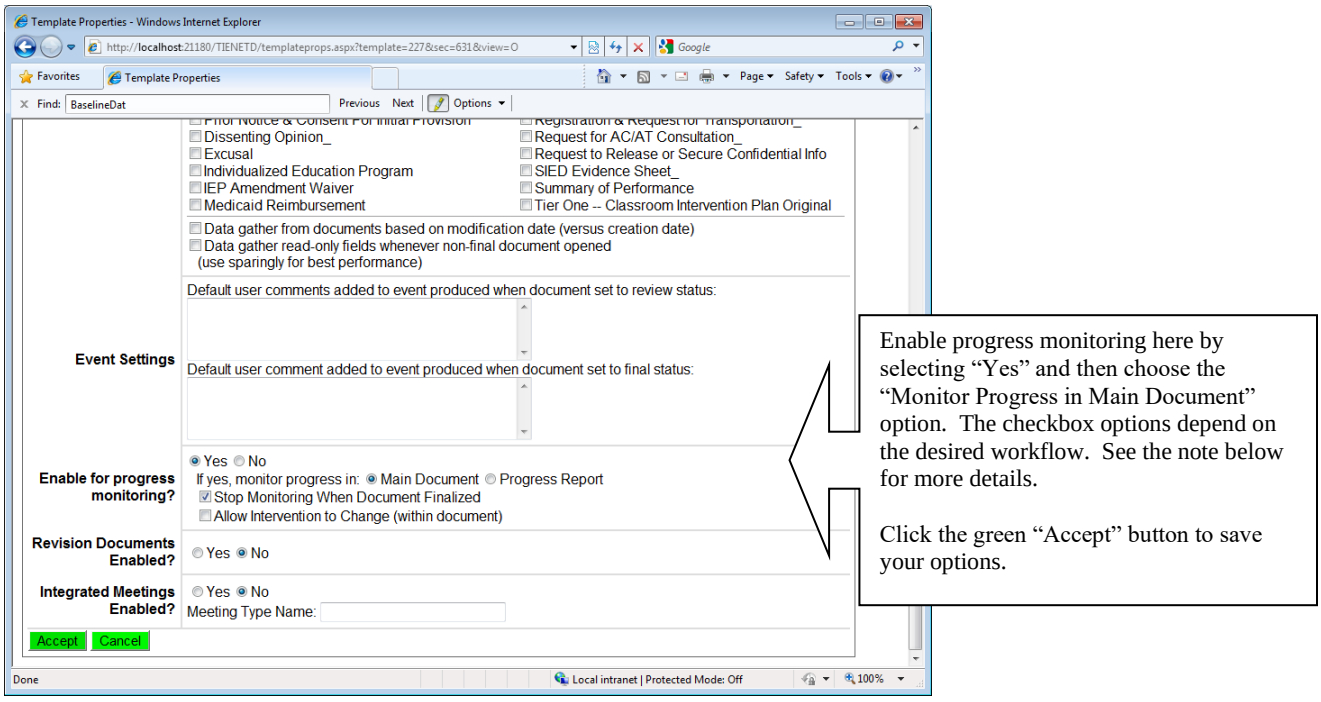


RtI Intervention Plan Document Templates

The functionality described here is only available if the database is licensed for use with the Response to Intervention (RtI) module. With the assumption that this license exists, this section describes how to configure an intervention plan document for RtI. Generally the document template will be aligned with existing intervention plan forms that the school district is using and most of the work in setting up the document template will be just like setting up any other document template. This section focuses specifically on how to enable progress monitoring and charting to occur directly within the context of the intervention plan document.

Make sure that the template properties described below (that are required for progress monitoring) are set.





About progress monitoring options: When enabling progress monitoring as in the figure above, there are several checkbox options that determine the user process flow. Typically a decision about the student will be made as the result of progress monitoring, and finalizing the document should lock in that decision,

potentially triggering conditional document actions that launch the next step in the process. In this case, check the "Stop Monitoring When Document Finalized" option to prevent the user from entering more data points once the document is finalized. Another choice is whether to allow intervention changes within the same intervention plan, or whether to alternatively require a new intervention plan document to be created when there is a change in intervention. The "Allow Intervention to Change (within document)" controls this choice.

Enabling progress monitoring automatically creates five critical fields that should be linked to a section, namely BaselineDate (date), BaselineScore (score), TargetDate (date), TargetScore (score) and UseProgressMonitoring (logical). The dates and scores define the goal line. The UserProgressMonitoring field allows progress monitoring to be turned on/off in individual documents since it may be desired that anecdotal/subjective approaches be used in certain situations. The "UseProgressMonitoring" field can be made editable in every document, set to a default, or set by a guided action. Configure the section in which you want to place these fields. Typical HTML formatting for these fields is shown below.

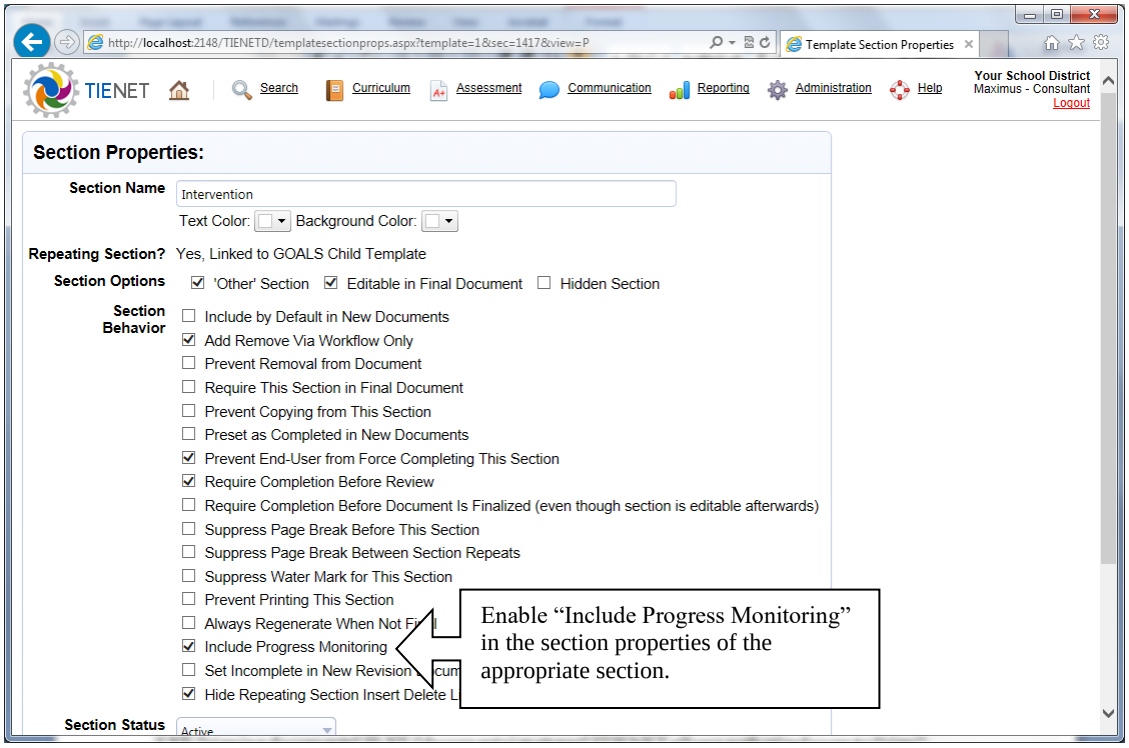
```
<tr><td><b>Start Date:</b> {BaselineDate}     <b>End Date</b> {TargetDate}
</td></tr>{#ENDIF}{#IF ImprovementMath=True}<tr><td> <b>Extended End Date:</b>
{ExtendedEndDateMath}{#ENDIF}</td></tr>

<tr> <td><b>Baseline Data Point:</b> {BaselineScore}    <b> Goal:</b> {TargetScore}
</td></tr>

{#IFEDIT}<tr><td><b>Initiate Progress Monitoring:</b>
{UseProgressMonitoring:A}<br><br></td></tr>{#ENDIF}

{#IFVIEWAND UseProgressMonitoring=TRUE}<tr><td><b>Progress Monitoring Results:</b>
<br>{&ProgressMonitoring}</td></tr>{#ENDIF}
```

Indicate which document template section will contain the progress monitoring data points and chart. Typically this is the same section that contains the critical fields described in the previous step. To specify the section, go to the section, click Edit Section Security and enable the "Include progress monitoring" option. By default, the progress monitoring data points and chart will be appended to the end of the section. However, you can optionally define precisely where the progress monitoring data points and chart will appear by inserting a {&ProgressMonitoring} placeholder in the HTML format (as in the last line of the sample HTML above).



Configuring Bulk Operations on Documents

This section describes how to configure scenarios where users can process multiple documents at once (create, print, apply pre-defined modifications).

- There is a new document template option labeled 'Allow bulk document creation'. If enabled via Document Template Properties, users authorized to create such documents can go to Students Search > Utilities > Bulk Document Creation to access a utility that allows them to select the document template and a set of student profiles for which to bulk-create documents. Note that the 'Allow bulk document creation' option cannot be enabled for a document template if the 'Use Document Setup for New Documents' option is enabled.

    - Note that if you want to link this to a bulk printing operation, add a date or date/time field named "DocumentPrintedOn" to the document template. PowerSchool Special Programs will automatically populate a field with this name when the document is rendered for printing. Then you can create a list report of documents that need printing using the formula "DocumentPrintedOn Is Empty" and users can bulk-print from that report.

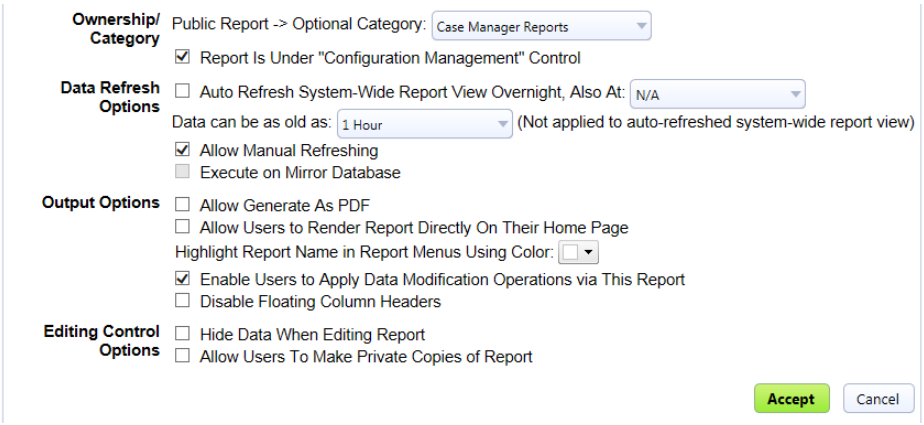
    - Also, note that it is possible now to set up section actions that trigger upon being rendered for printing, and therefore can queue the document to be set to final.

- It is now possible to enable users to apply pre-defined data modification operations on documents via a list report. This can be used to implement specialized scenarios such as a bulk document approval process. To configure this, follow these steps:

    1. Configure a document action with a "triggering event" of bulk operation and that performs the desired data modification on the document.

    2. Set up a list report that presents the profiles or documents for which users should be able to apply the pre-defined data modification operation. Then go to the report properties of the list report and enable the "Report if under 'Configuration Management' option." Then look for the "Enable Users to Apply Data Modification Constraints via This Report" option (under 'Output Options') and enable it too. This second option will only appear if an eligible document action has been created.

    3. Make sure that the security for the list report only grants access to staff users who should be able to execute the bulk action(s). This is your primary way of controlling who can execute bulk actions. If you link the document action to a security group, that will also restrict who can execute that particular bulk operation.



Master/Detail Documents

This feature enables the configuration of a special relationship between a "master" child template and a "detail" document template. An end-user editing a "master" child document through repeating rows or repeating sections can initiate the creation of a "detail" document by clicking a button (configured in the HTML format for the repeating row or section), or alternatively this can be automated programmatically through a document or section action. The "detail" document is bound to the "master" child document, enabling data flow back and forth between them. If the master document is deleted or undeleted, then corresponding detail documents are deleted or undeleted at the same time.

At the configuration level, the constraints are as follows:

- Only one "detail" document template can be created for a given "master" child template.

- Creation of the "detail" document template is initiated through the configuration of the "master" child template, although from that point forward, the configuration of the "detail" document template can proceed just as with any other document template.

- PowerSchool Special Programs will not allow the "master" child template to be deleted unless the "detail" document template is deleted first.

At the data level, the constraints are as follows:

- At most one detail document can be created for each master child document, and therefore a master child document will have zero or one detail documents bound to it.

- If the master child document is deleted for any reason, the detail document is automatically deleted. However, the user interface prevents master child documents with detail documents from being manually deleted through the repeating section or repeating row user interface. If the detail document is deleted directly, the corresponding master child document is no longer bound to a detail document and therefore can be deleted through the repeating row or section user interface.

The master child template will support special button directives in repeating row or section layouts to create or view the corresponding detail documents. The directives can be placed inside of #IF conditional directives for additional control over when they appear.

- The create button for a child document will only be visible if the corresponding detail document does not exist yet. The directive must reference a "pseudo-action" named "CREATEDETAIL" and will generally take the form {*CREATEDETAIL:L"Create Detail"}. Note that "CREATEDETAIL" is a pseudo-action and not an actual section action or document action that you need to configure. The directive also supports CSS class and other attributes just like a button directive. Creation of the detail document via this button will operate even if the user has no editing rights to the detail document, although such a user would not be able to edit the detail document that got created.

- The view button for a child document will only be visible if the corresponding detail document exists. The directive must reference a pseudo-action named "VIEWDETAIL" and will generally take the form {*VIEWDETAIL:L"View Detail"}. It also supports CSS class and other attributes just like a button directive. The button assumes the user has at least view access to the detail document given that the user has access to the master document. This means that the view button does not currently support the scenario where a user can access the master document but has no view rights to the detail document.

When a detail document is created, data will flow to the detail document using the normal data flow options that are available when a workflow document is created. In addition to this, a detail document template supports several specialized data modification methods (in document and section actions) that can pass data back and forth between the master child document and the detail document. Note that these special modification methods are configured in the detail document and therefore take the perspective of the detail document.

- **Set Fields from Master Child Document:** Writes fields in the detail document using values from master child document. The trigger for this is typically the creation of the detail document.

- **Update Master Child Document:** Writes fields in the master child document using values from the detail document.

- **Insert Master Child Document:** Inserts a new master child document using values taken from the detail document. Note that the new master child document will initially not have a detail document linked to it, so once the document or section action is executed, there is no link between the new master child document and the current detail document that initiated its creation.

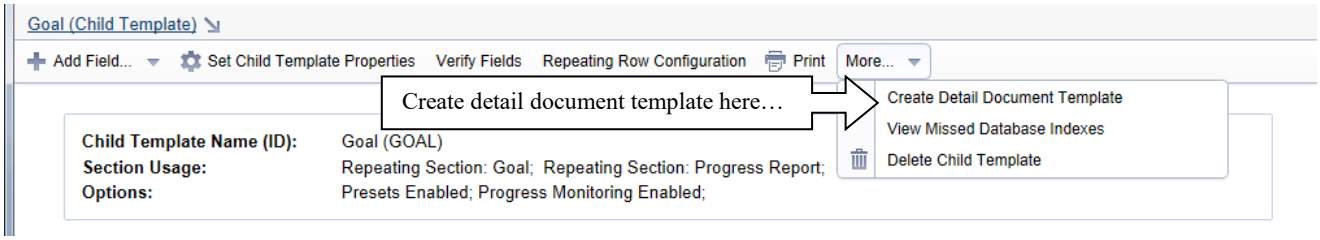
Additionally there is a special modification method that can be configured in the master document template as follows:

- **Create Detail Documents:** Creates detail documents for the master child documents that meet the specified criteria (and that do not already have a detail document).

Since detail documents are not created manually, it may be useful to automatically set the comment of detail documents using a document action.

An example scenario where the master/detail document feature is very useful is when activities need to be assigned to various staff in a master document, and then detail documents are launched for each activity so that the assigned staff member can record details related to that activity. The "document owner security" model can be used to make sure staff can only see their own detail documents and not those assigned to other staff. The document and section actions described previously can capture data entered into the detail documents and move them back into the parent.

To get started, go to the configuration of the document template, select the child template from the section flyout menu, and then select "Create Detail Document Template" from the "More" dropdown menu.



Language Localization

Localization, by definition, is the process of preparing a system to appear in a non-default language (e.g. French, Spanish, etc). Localizing PowerSchool Special Programs requires consulting with PowerSchool to make sure the user interface itself is localized for the desired language, and then translating all the relevant configuration: forms, templates, document actions, section actions data dictionary, etc. Translation of the configuration can be done through the configuration user interface by an authorized user.

The "System Administration Guide" describes how to translate document templates into a non-default language so that documents originally written in the default language (i.e. English) can then be translated into that language. Although "localization" also requires document templates to be translated, "localization" as described here assumes documents will be written directly in the non-default language instead of being translated.

To localize the configuration of PowerSchool Special Programs, perform the following high level steps:

For each keyword table used for keyword selection fields or multiple value fields, make sure there is either a 'Name' or 'Description' character column containing default language text. Then add an additional character column with the name 'Name_*languagecode*' or 'Description_*languagecode*' where *languagecode* is the two-character ISO code for the language (e.g. es for Spanish, fr for French, etc). Populate this column with the translations.

Translate the document templates as described in the "System Administration Guide". Then for full localization, do the following:

1. Enable the 'Allow writing documents directly in language(s) into which template has been translated?' language checkbox option in the template options for each document template.

2. Translate the name of each document template and document template category.

3. Translate the name of each section within each document template.

4. Translate all document and section actions, and optionally the captions for the document data fields.

5. For each profile type, translate all section names, HTML formats, constraint messages, profile field captions and the caption for the profile type itself.

6. Translate relevant standard reports (list reports and multi-dimensional reports). Translating advanced reports is not supported at this time, but a workaround is to duplicate the desired sections inside of the existing advanced report and localize just those duplicate sections into the target language.

Configuring DocuSign Integration

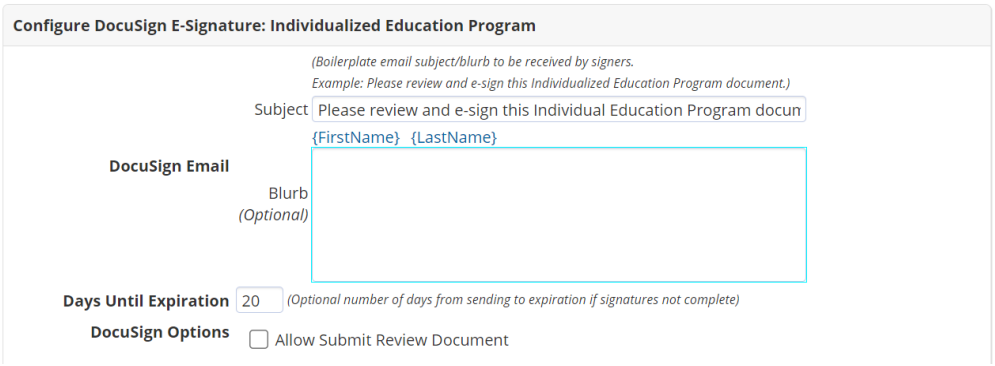
PowerSchool Special Programs allows configuration of DocuSign integration for specific document templates that require signatures, such as parental consent documents. DocuSign configuration for document templates can lie dormant in a customer's database until a DocuSign account is configured by the customer. This allows a state/province model database to be delivered fully ready for DocuSign, and then activating the integration requires only establishing a DocuSign account.

Only finalized documents can be sent to DocuSign for e-signature by signers. At the time of submission, each signer is identified by name and email address. The signer data is supplied to the DocuSign service via its API along with the finalized document in the form of a PDF file containing embedded "signing

tabs". Note that DocuSign itself refers to the package containing the signer data and PDF as an "envelope". The document (or "envelope" as referred to by DocuSign) initially has a status of "sent". But the status will ultimately change to "voided" if canceled by the sender, "declined" if rejected by any one signer, or "completed" when all signers have signed the document. The completed PDF (signed by all parties) is stored back in PowerSchool Special Programs as a file attachment to the document for easy access.

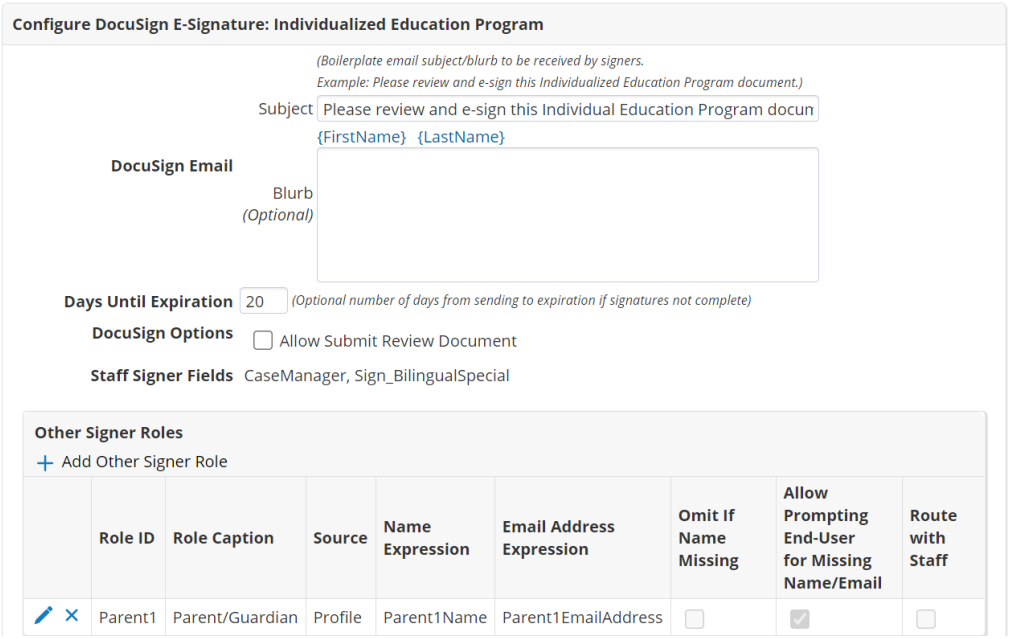
One can configure document templates to be integrated with DocuSign with or without a DocuSign account in place. The DocuSign account is only needed for testing the configuration, but since testing is always recommended, a DocuSign account can be established by selecting "Configuraton" from the "Administration" menu, clicking the "Integration" tab and then "DocuSign Integration".

Step 1: To configure a particular document template for DocuSign integration, log in and access that document template's main configuration screen. Select a configuration task and then select "Configure DocuSign E-Signature" from the upper "More" dropdown menu. As shown in the screen shot below, you are now prompted for DocuSign email boilerplate text that will go out to signers, and an optional number of days from sending to expiration after which pending documents sent out for signatures will automatically be voided if signatures are not completed by then. Accepting this configuration is technically sufficient for the system to recognize the document template as "DocuSign integrated" and, if a DocuSign account is also configured, to show authorized end-users the option to submit documents for e-signature. However, more steps are needed to make the integration fully functional, and these steps are covered next.



Step 2 - Configure staff signer role(s): Each "staff signer role" is based on a staff field in the document. When in operation, the DocuSign integration identifies who the potential staff signers are for a document by extracting staff from any staff fields in the document template, or any child template of the document template, that are marked with the 'Require Signature' field property. So a necessary step is to set this property in the appropriate staff fields. If the field property/option is not visible, verify that you have completed Step 1

Step 3 - Configure "Other Signer Role(s)": Typically these roles are used for students, parents and guardians; but they can also be used for non-lookup staff roles for which there is not a staff profile field. If other signer role(s) are necessary for the document template, select "Configure DocuSign E-Signature" from the upper "More" dropdown menu (as per step 1) and observe that you can add other signer roles as shown in the screen shot below.



For each such signer role, it will be necessary to query either the profile or the document for the name and email address of the person who will be the signer. If you are expecting the queried name or email address to potentially have empty values, you can optionally set a checkbox option for that signer role to allow the system to prompt the end-user for the missing information at the time of submitting the document for e-signature. The following specific configuration elements are needed for each of the other signer roles:

- **Role ID:** This is a unique identifier that is used to associate e-signature tabs with the role.
- **Role Caption:** Identifies a label for the signer role, such as "Parent". This caption is only displayed in the user interface for submitting documents for e-signature and monitoring their status. The role caption will never be shown within DocuSign to the actual signers of the document.
- **Source:** Identifies where the name and email address of the signer will be queried from. At this time, "profile" and "document" are the possible sources.
- **Name Expression:** A formula expression in the context of the source (profile or document) that produces the name of the signer in First Last format.
- **Email Address Expression:** A formula expression in the context of the source (profile or document) that produces the email address of the signer. This is typically just the name of a field containing email addresses.
- **Allow Prompt for Missing Name/Email:** If checked, and the name and/or email address values are blank in the context of submitting a particular document for e-signature, the end user is prompted and can supply those missing values at the time of submission. If this option is not

checked, the missing information will be called out, but the user will be prevented from submitting the document for e-signature.

- **Route with Staff:** If checked, then the corresponding signer role will always be routed at the same time as signers based on staff profile fields.

Note that is also possible to configure other signer roles in child templates to support repeating signer data. From the document template configuration, choose the child template from the fly out menu, and then select "Set Child Template DocuSign Properties". Field maping is not supported in child templates at this time.

Step 4 - Configure E-signature tab directives: Each staff field-based and other signer role should have one or more associated "signer tab" directives in the document template section designed for signature collection. The table below delinates all the variations in the syntax of signer tab directives. Note that all names used in the directive syntax are required to be alphanumeric (underscore allowed).

Directives	Explanation
{'*StaffFieldName*:SignHere} {'*RoleId*:SignHere}	Creates a "sign here" tab associated with the specified staff field or other signer role.
{'*StaffFieldName*:SignHere[*instancename*]} {'*RoleId*:SignHere[*instancename*]}	If there is a need to have the same person sign a document in more than one place, you can have more than one sign here directive for the same person, but each one must be identified with an instance name.
{'*StaffFieldName*:InitialHere} {'*RoleId*: InitialHere} {'*StaffFieldName*:InitialHereLO} {'*RoleId*: InitialHere:O}	Creates an "inital here" tab associated with the specified staff field or other signer role. An "O" modifier can be used here to make a particular "onitial here" field optional for the signer.
{'*StaffFieldName*: InitialHere[*instancename*]} {'*RoleId*: InitialHere[*instancename*]}	If there is a need to have the same person initial a document in more than one place, you can have more than one initial here directive for the same person, but each one must be identified with an instance name.
{'*StaffFieldName*:DateSigned} {'*RoleId*:DateSigned}	Creates a "date signed" tab associated with the specified staff field or other signer role. Note that DocuSign automatically populates the tab with the date of signing.
{'*StaffFieldName*:DateSigned[=*datefield*]} {'*RoleId*:DateSigned[=*datefield*]}	Creates a "date signed" tab that is mapped to a date or date/time field in the document template. When the signers have all completed their signatures, the date or date/time will be written back to the document field.
{'*StaffFieldName*:CheckBox} {'*RoleId*:Checkbox}	Creates a "checkbox" tab associated with the specified staff field or other signer role. This syntax should only be used when there is only one checkbox for the signer. If there will be more than one checkbox for the signer, the "instance name" syntax in the next row should be used instead. In DocuSign, a signer is not required to set the value of the checkbox, and may leave it unchecked. To require a signer to make an explicit choice (i.e. give consent or deny consent), use "radio" tabs instead. See the notes below the table for more information on checkboxes.
{'*StaffFieldName*:CheckBox[*instancename*]} {'*RoleId*:Checkbox[*instancename*]}	If there will be more than one checkbox for a signer, a unique instance name must be assigned to each checkbox using this syntax. Note that the instance name only needs to be unique for checkboxes associated with the given signer. Two different signers may have a checkbox with the same instance name.
{'*StaffFieldName*:CheckBox[=*logicalfield*]} {'*RoleId*:Checkbox[=*logicalfield*]}	Creates a checkbox tab that is mapped to a logical field in the document template. When the signers have all completed their signatures, the value of the DocuSign checkbox will be written back to the document logical field.
{'*StaffFieldName*:Radio(*groupname*)[*instancename*]} {'*RoleId*:Radio(*groupname*)[*instancename*]}	Creates a "radio" tab associated with the specified staff field or other signer role. Radios are often preferable to checkboxes because the signer is required to make an explicit choice. The "group name" is used to group radios together into mutually exclusive groups. The instance name is required to uniquely identify each

	individual radio, and the instance name must be unique within the same group for the same signer. See the notes below this table for more information on radios.
{'*StaffFieldName*:Radio(*groupname*)[=*logicalfield*.true]} {'*StaffFieldName*:Radio(*groupname*)[=*logicalfield*.false]} {'*RoleId*:Radio(*groupname*)[=*logicalfield*.true]} {'*RoleId*:Radio(*groupname*)[=*logicalfield*.false]}	Creates a "radio" tab that is mapped to a document template logical field and a specific value or either true or false. When the signers have all completed their signatures, the specified logical value will be written back to the document logical field. For best results, the document logical field should allow empty values and default to empty so that it is unambiguous whether or not a value has been written back. See the notes below the table for more information on radios.
{'*StaffFieldName*:Radio(*groupname*)[=*keywordfield.keyword*]} {'*RoleId*:Radio(*groupname*)[=*keywordfield.keyword*]}	Creates a "radio" tab that is mapped to a document template keyword field and a specific keyword value. When the signers have all completed their signatures, the specified keyword value will be written back to the document keyword field.
{'*StaffFieldName*:Text:W*width*} {'*RoleId*:Text:W*width*}	Creates a "textbox" tab of the specified width, for example, {'Parent1:Text:W600}. By default, the textbox is optional for the signer, but one can add an additional R modifier to make the textbox required, as in {'Parent1:Text:W600R}. Multi-line word wrapping can be enabled on the textbox by specifying its height using an additional H modifier, as in {'Parent1:Text:W600H200}.
{'*StaffFieldName*:Text[*instancename*]:W*width*} {'*RoleId*:Text[*instancename*]:W*width*}	If there will be more than one textbox for a signer, a unique instance name must be assigned to each checkbox using this syntax. Note that the instance name only needs to be unique for textboxes associated with the given signer. Two different signers may have a textbox with the same instance name.
{'*StaffFieldName*:Text[=*characterfield*]:W*width*} {'*RoleId*:Text[=*characterfield*]:W*width*}	Creates a "textbox" tab that is mapped to a document template character fieldvalue. When the signers have all completed their signatures, the specified character value will be written back to the document keyword field.

Radios and Checkboxes: A "radio" or "checkbox"directive will both render a checkbox whether or not DocuSign is active. When a document is sent to DocuSign for signing, DocuSign will actually render its own checkboxes and it is helpful to leave plenty of space for DocuSign to do that. In DocuSign, a signer is not required to set the value of a checkbox, and may leave it unchecked. Radios are often preferable to checkboxes because the signer is required to make an explicit choice. The example formatting below implements a requirement for the signer to explicitly choose giving consent or not giving consent.

```
<div>Do you consent to the plan outlined in this document?   <label>{'ParentSigner:Radio(Consent)[=DocuSignConsent.true]}
I give consent</label>   <label>{'ParentSigner:Radio(Consent)[=DocuSignConsent.false]} I do not give consent</div>
```

Note that if the fields being written to (i.e. DocuSignConsent in the example above) are not explicitly displayed in a document section, it is a good idea to mark them in the data dictionary as "Internal Values" so that they do not mistakenly get deleted.

In some cases, you will want to have a standard checkbox data field that effectively gets swapped with a DocuSign checkbox tab when the PDF is generated for DocuSign. In this case, have the logical field directive right next to the tab directive and apply the built-in "HIDEIFDOCUSIGN" CSS class to the logical field directive. Additionally, you will want to modify the tab directive so that it does not automatically render a checkbox outside of Docusign. To do that, modify the tab directive to add add an "I" (for invisible) modifier such as at the end of the following syntax: {'StaffFieldName:Radio(groupname)[?keywordfield.keyword]:I}.

FYI – Important: A document will only be sent to a particular signing role if one or more signing tabs associated with that role are found in sections included in the finalized document. This enables signing roles to be conditionally included or excluded using #IF logic, or conditional inclusion of sections.

To trouble shoot issues, it is helpful to understand how the signing tabs work "under the hood". A directive like {'CaseManager:SignHere} is rendered as the HTML shown below. The "ESIGNTAB" CSS style is designed to keep the"[CaseManager:SignHere]" text invisible in the document.

```
<span class="ESIGNTAB">[CaseManager:SignHere:]</span>
```

The HTML coding above is embedded in the PDF document to DocuSign. DocuSign recognizes the coding and positions the tabs accordingly. PowerSchool Special Programs injects the appropriate coding into final documents whether or not a DocuSign account is in place, which enables past documents to be potentially signed once an account is in place. However, documents finalized before the signer tab directives were configured in the section HTML format will not be sent to DocuSign successfully because they will not have the HTML coding.

Step 5 (define reporting and workflow): There are a number of functions documented in the "Formula Reference" document that are specifically related to DocuSign E-signature. These functions can be used in the context of a report definition to pull out documents that have a particular state in the signing process.

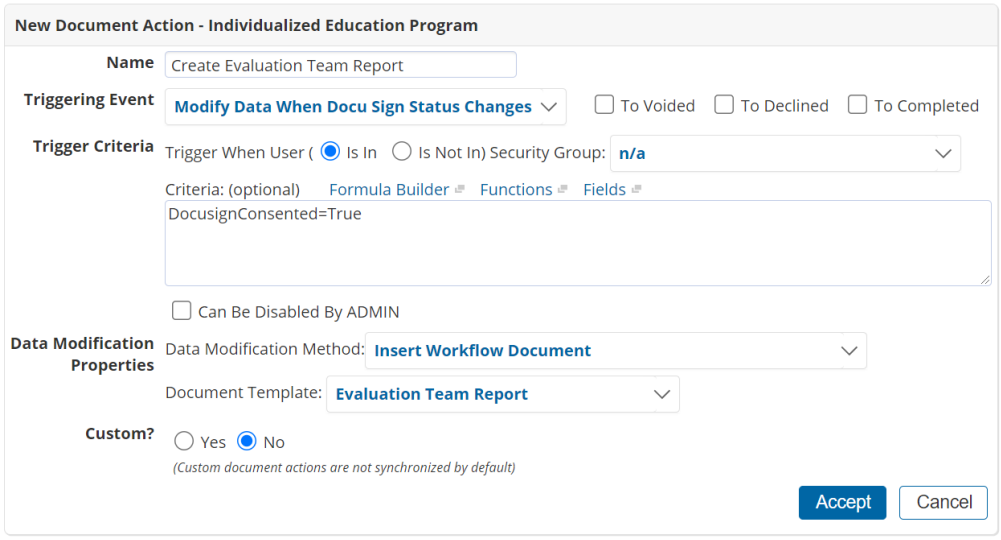
With regard to workflow, it is important to be aware of the distinction between between users authorized to "submit" documents for E-Signature versus users authorized to "send" documents to DocuSign. Most pricing plans for DocuSign limit the number of users who can send documents via DocuSign. To overcome this limitation, users who are not provisioned as DocuSign users can nonetheless be authorized with a "Submit to DocuSign" document template security right, which authorizes them to submit the document to another user who has been provisioned as a DocuSign user and can therefore send documents via DocuSign. DocuSign users are provisioned within the DocuSign administrative dashboard. The corresponding users in PowerSchool Special Programs must be granted the "Send Documents via Docu Sign" student privilege. The "Send Documents via Docu Sign" privilege can be granted at different organizationa levels to support, for example, having one sender per school.

It is helpful here to have an understanding of the distinct states that a document goes through in the DocuSign process:

- **Submitted:** The document has been submitted to an authorized sender (user provisioned within DocuSign) but has not actually been sent yet via DocuSign.
- **Sent and Pending:** This status means that an authorized user, provisioned as a DocuSign user and with the "Send Documents via Docu Sign" privilege, has sent the document to DocuSign for signing. DocuSign manages the signing process forward. Pending refers to the fact the document has not reached one of the three end states yet.
- **Voided:** This is one of three possible end states. This means that the sender has voided/canceled the signing of a document. Signers are notified and processing stops. A document is also voided if it times out by exceeding the allowed amount of time for the sending process. Note that a reason for voiding is captured along with this state.

- **Declined:** This is one of three possible end states. If any signer declines the document, the document as a whole is now declined and processing stops. Note that a reason for declining is captured from that signer.
- **Completed:** If all signers sign the document, the process is complete and the document is considered signed.

When processing of a document is complete and the state changes to either voided, declined or completed, then the submitter and/or sender are automatically notified with a message that contains a link bank to the document. If additional workflow is needed, a document action can be created with a "Modify Data When Electronic Signature Status Changes" trigger, which is further qualified to be triggered when the status changes to one or more specific values (see below). It can be assumed that any document field values mapped to signing tabs will be written back before any document action triggers are evaluated, which allows the workflow logic to be based on such mapped field values.



Omitting certain document sections from DocuSign [New in 20.11]**:** There is a document template section option named "Omit from Electronic Signature", which when enabled, will result in the section being omitted from the document sent to DocuSign for signing. This option is overrideable by the system administrator.

Docusign Conditional Logic Directives [New In 20.11.1.0]: The following directives are helpful in controlling certain section content designed for DocuSign:

- {#IF_DOCUSIGN_ACCOUNT_EXISTS}*content*{#ENDIF} includes the content only if a DocuSign account has been configured by the customer.
- {#IF_NOT_DOCUSIGN_ACCOUNT_EXISTS}*content*{#ENDIF} includes the content only if the customer has not configured a DocuSign account.
- {#IF_DOCUSIGN_STAFF_SIGNING_ENABLED}*content*{#ENDIF} includes the content only if the customer has not disabled staff signing for the document template.

- {#IF_DOCUSIGN_STAFF_SIGNING_DISABLED}*content*{#ENDIF} includes the content only if the customer has disabled staff signing for the document template.

Configuring Digital Signature

PowerSchool Special Programs allows the configuration of Digital Signature for specific document templates that require signatures, such as parental consent documents. Digital Signature configuration for document templates can lie dormant in a customer's database until a Digital Signature license is configured by the customer. This allows a state/province model database to be delivered fully ready for Digital Signature, and then activating the functionality only requires licensing and enabling Digital Signature.

The Digital Signature configuration process was built on three principles: 1) configurators should not need to completely refactor a document to add Digital Signature to it, 2) customers using the model but not using Digital Signature should not see any change to how their document functions, and 3) the configuration process should align with existing signature functionality as much as possible.

As a result, configuring Staff and Other Signers is done through the same configuration that is needed for DocuSign. In addition, reporting on Digital Signature is done using the same methods as DocuSign. Configuring which fields need to be filled out by which signers follows a different configuration method.

Activating Digital Signature: Digital Signature is active when it is both licensed and enabled. Templates can be configured for Digital Signature even when Digital Signature is not active, however the Digital Signature configuration cannot be tested without activating Digital Signature.

Licensing Digital Signature: Digital Signature can be licensed using the Manage Client Licenses tool. Once the license "e Signature" has been activated, admin users and staff users who can view configuration will gain access to the Digital Signature Settings tab.

Enabling Digital Signature: Digital Signature can be enabled from the Digital Signature Settings page, located at Administration -> Configuration -> Settings -> Digital Signature Settings. Admins and Consultants can enable Digital Signature. Staff users who can view configuration can view the Digital Signature settings but cannot modify the Digital Signature settings. Once enabled, users are able to submit documents for signing and sign documents.

When Digital Signature is active (i.e. licensed and enabled), templates with Digital Signature configured will behave differently. Staff users filling out a document will still be able to enter data into signature areas, however any required fields in signature areas will not be required. Once a signature request is submitted, signature areas will only be editable when accessed through the Signature Wizard by the right signer. When signing, required fields will be required again. The Signature Wizard is used to guide users through the Signing process and collect the signature data. When all signers have completed signing, the collected signature data is saved to the document.

When Digital Signature is not active (i.e. unlicensed or disabled), the document will continue to function like it did prior to the introduction of Digital Signature.

Configuration Step 1: A Digital Signature configuration must be created for templates that are being configured for Digital Signature. This works the same as DocuSign configuration, **Step 1**.

Step 2 - Configure staff signer role(s): Staff Signer roles must be created for templates that have staff signers. This works the same as DocuSign configuration, **Step 2**, except that a Staff Profile Reference field's Description value is used as the Role Caption if it exists. Signers added through this method can sign through the application, as well as through the signature request email.

Step 3 - Configure "Other Signer Role(s)": Other Signer roles must be created for templates that have non-staff signers. This works the same as DocuSign configuration, **Step 3**.

Step 4 - Configure Digital Signature areas: Digital Signature is configured using the #IFSIGN directive and the 'G', 'S', 'I', and 'L' directive modifiers. These tools can be used to define certain areas of the document as signature areas associated with a signer role. When Digital Signature is enabled, fields in these areas can only be modified by the signer with the specified signer role. These areas cannot be edited by the staff member filling out the rest of the document or other signers. When Digital Signature is disabled, the fields in these areas act normally.

Directive	Explanation
{#IFSIGN RoleID} … {#ENDIF} {#IFSIGN RoleID1, RoleID2, RoleID3, …} … {#ENDIF}	The RoleID must match the role ID of one of the staff signers associated with the document, or one of the other signers associated with the document. When signers are signing the document, the content within the #IFSIGN directive is only editable for the signer with a matching Role ID. When staff are editing (not signing) the document before a signature request is submitted, fields that appear within the #IFSIGN directive will be editable to allow staff to pre-fill certain values that the signers will be able to edit while signing. Multiple RoleIDs can be associated with a single #IFSIGN directive. When multiple roles are listed, each matching role can sign the area. If multiple signers sign the same area, all responses are displayed on the document and signed PDF, but only the first response will be saved to the document's data. **Warning**: RoleID is case sensitive and must exactly match the configured RoleID.
{#IF_ESIGNATURE_ENABLED} … {#ENDIF}	Includes the content if Digital Signature is enabled. Alternatively, formula field, ESignEnabled, can be used. Eg. {#IF . ESignEnabled} {#ENDIF}
{#IF_NOT_ESIGNATURE_ENABLED} … {#ENDIF}	Includes the content if Digital Signature is not enabled.
{#IF_STAFF_SIGNING_ENABLED} … {#ENDIF}	Includes the content if staff signing is currently enabled for this document template.
{#IF_STAFF_SIGNING_DISABLED} … {#ENDIF}	Includes the content if staff signing is currently disabled for this document template.

{#IFVIEW} … {#ENDIF}	The behavior of #IFVIEW directives (including #IFVIEWAND, #IFVIEWOR, etc.) changes when the document is accessed through the signature wizard. Includes the content if it is not in a signature area with a RoleID matching that of the current signer.
{#IFEDIT} … {#ENDIF}	The behavior of #IFEDIT directives (including #IFEDITAND, #IFEDITOR, etc.) changes when the document is accessed through the signature wizard. Includes the content if it is a signature area with a RoleID matching that of the current signer.

The G field directive modifier can be used to control the behavior of a field in an #IFSIGN directive:

Directive Modifier	Explanation
{LogicalValue:S} {LogicalValue:SL} {LogicalValue:SL"Label"}	When rendering the document in the signature wizard, the logical value field will be rendered as a Sign button. When clicked, the Sign button will be replaced by the signer's signature. When rendering the signed PDF, a value of True will display the signer's signature. A value of False will not display anything. When rendering for anything else, nothing will be displayed. When rendering with an 'L' directive modifier, a line will be placed underneath the sign button, signer's signature, or empty area. When rendering with an 'L:"text here"' directive modifier, a line and label, with the wording "text here", will be placed underneath the sign button, signer's signature, or empty area.
{LogicalValue:I} {LogicalValue:IL} {LogicalValue:SI"Label"}	When rendering the document in the signature wizard, the logical value field will be rendered as a Initial button. When clicked, the Initial button will be replaced by the signer's initials. When rendering the signed PDF, a value of True will display the signer's initials. A value of False will not display anything. When rendering for anything else, nothing will be displayed. When rendering with an 'L' directive modifier, a line will be placed underneath the initials button, signer's initials, or empty area. When rendering with an 'L:"text here"' directive modifier, a line and label, with the wording "text here", will be placed underneath the initials button, signer's initials, or empty area.
{FieldName:G}	Removes the field from the signature area. In other words, this field will be editable by the staff user filling out the document, and will not be editable by the signer whose RoleID matches that of the #IFSIGN directive.
{TextFieldName:G"system:FullName"} {TextFieldName:G"system:Initials"} {DateTimeFieldName:G"system:SignatureDate"}	Associates the field with one of the system's signature fields. When the system field is updated by the Signature Wizard, this field's value will also be updated. This field will otherwise be read-only in all circumstances.
{FieldName:Modifiers:G"OtherFieldName:Modifiers"}	When Digital Signature is active, OtherFieldName will be rendered with the specified modifiers.

	When Digital Signature is not active, FieldName will be rendered with any other specified modifiers. This directive modifier is typically used to provide an alternate Logical Value field for a signature when the form already contains something like a text field for the signature. This directive modifier is equivalent to {#IF_NOT_ESIGNATURE_ENABLED} {FieldName:Modifiers} {#ELSE} {OtherFieldName:Modifiers} {#ENDIF}

Step 5 (define reporting and workflow): You can report on Digital Signature data using the same methods outlined in DocuSign configuration's **Step 5**.

Other Digital Signature Settings: The Digital Signature Settings page, accessed through Administration -> Configuration -> Settings -> Digital Signature Settings, contains a couple of other settings that will control Digital Signature behavior.

The **Document Delivery Options** can be used to determine what is sent to signers after the signing process has been completed. If no options are checked, signers are not notified when the signing process has been completed.

The **Default Routing Order** can be used to determine the standard routing order preferred by the district.

The **PDF Creation Options** can be used to tell the system how to handle #IFSIGN areas with data from multiple signers. “One PDF with all signer’ responses” will always generate one PDF, with any areas with data from multiple signers duplicated for each signer. “Separate PDFs for each non-staff signer when responses differ” will generate separate PDFs when there are any areas with data from multiple signers, otherwise it will generate one PDF.

The **Notify staff when a signed document is modified** setting can be used to control the info message to staff signers when they access a document that has been modified after it was signed. When checked, the info message is displayed. When unchecked, the info message is not displayed.

The **Staff Email Address Field** can be used to determine what Staff profile field contains the email address of staff members.

The **Digital Signature-Ready Document Templates** table can be used to disable Digital Signature for staff users on specific document templates.

In addition, specific sections of document templates can be **Omitted from Digital Signature** by selecting the “Omit from Electronic Signature” property for that section.

Tips and Tricks

When using the #IFSIGN directive with multiple RoleIDs, make sure that the HTML content of the directive includes the entire element, not just the opening or closing tag since the area may be repeated.

For example, this will have unexpected results:

```
{#IFSIGN RoleID1, RoleID2}
<table><tr><td>Label</td><td>{SignatureField}</td></tr>
 {#ELSE}
<table><tr><td>Other non-signature text</td></tr>
 {#ENDIF}
 </table>
```

Instead, the closing tag should be included in the #IFSIGN area.

```
{#IFSIGN RoleID1, RoleID2}
<table><tr><td>Label</td><td>{SignatureField}</td></tr></table>
 {#ELSE}
<table><tr><td>Other non-signature text</td></tr></table>
 {#ENDIF}
```

#IFSIGN areas should be editable while in draft or review so that the data can be 'pre-filled' before signers sign.

The response priority for signers who have already signed an active signature request will not be modified when this expression is modified. If there are multiple possible priorities, be sure to leave gaps between them so you can easily add priorities between existing priorities later (i.e. use 0, 100, 200, 300 instead of 0, 1, 2, 3).

Other Configuration for Digital Signature

ESignature Configuration

For the most part, a document's ESignature Configuration acts the same whether it's used with Digital Signature or DocuSign. However, there are a couple of settings that will behave differently.

The **eSignataure Email Subject** and **Blurb** are both optional. If either are not specified, the system will provide a default subject/blurb for Digital Signature emails.

The **Allow Submit Review Document** option allows Digital Signature submission while the document is in Review. When used with Digital Signature, this setting also allows Digital Signature submission while the document is in Draft, although the document will need to be promoted to review before the submission process can be completed.

Other Signer roles can have a **Response Priority Expression** specified. This expression is only used with the #IFSIGN digital signature configuration method. When multiple signers have entered data for the same fields, the first signer to submit their response is the one whose data will be used on the final document. This expression can be used to override that default behavior. The data from the signer with the highest response priority will be used on the final document. If multiple signers have the same response priority then the data from the first signer among that group will be used on the final document. This expression is run when the signer's data changes, and the result is saved to the signer's ResponsePriority field.

Section Actions

The **Prevent Section Completion** section action has some options that relate to Digital Signature:

- "Only when editing document" means that the section action will not prevent section completion while signing the document and will only prevent section completion while editing the document.
- "Only when signing document" means that the section action will not prevent section completion while editing the document and will only prevent section completion while signing the document.
- "When editing and signing document" means that the section action will always prevent section completion - both while editing the document and while signing the document.
- "Only when signing if Digital Signature is enabled; otherwise, when editing" means the section action will behave differently depending on whether Digital Signature is enabled. If Digital Signature is enabled, then the section action will function like "Only when signing document", and if Digital Signature is disabled, then the section action will function like "Only when editing document". This option can be used to create a section action that will be useful both for customers using Digital Signature, and those opting not to use it at all.

In addition, the system will only run section actions where one or more fields referenced by the trigger criteria are editable by the signer. If none of the fields referenced by the trigger criteria are editable by the signer, then the section action is not checked for that signer. This prevents actions meant for one signer role from preventing the completion of the signing process by another signer role.

Document Actions

The **Modify Data when Electronic Signature Status Changes** document action applies to both DocuSign and Digital Signature. When the document's signature request is updated to one of the specified statuses, the rest of the document action will be run.

Section Properties

The **Omit from Electronic Signature** section property applies to both DocuSign and Digital Signature. When selected, the section will be excluded from the Signature Wizard and signed PDF.

Using DocuSign Configuration with Digital Signature

Models configured for DocuSign are also compatible with Digital Signature. No configuration changes should be needed for DocuSign documents, although there may be edge cases where the two systems are incompatible – see the **Tips and Tricks** section below.

Submitting a document configured for DocuSign to Digital Signature follows the same process as submitting a DocuSign document to DocuSign. Once submitted, the document switches to the Digital Signature signing process.

Transitioning to Digital Signature from DocuSign: If a DocuSign account exists and Digital Signature is unlicensed or disabled, the system will submit new signature requests to DocuSign. Once Digital Signature has been licensed and enabled, the system will submit new signature requests to Digital Signature, even if the DocuSign account still exists. Existing signature requests that were made to DocuSign and have not been completed will continue to communicate with DocuSign.

Signed Documents: As each signer signs the document, the document will be updated to include that signer's responses. These responses will be seen in view, edit, and print mode. Signatures will only be included on the final signed PDF.

Tips and Tricks

- Repeating rows for child templates that contain DocuSign fields must not be set to View Only. The child template must at least be updatable so that the DocuSign fields can be updated from the Signature Wizard.
- Since DocuSign documents signed with Digital Signature will display all tab responses on the signed document, some bindings that are mapped to document fields may display duplicate values. For example:

  ```
  {PGExcusalParent1Date}{'ParentSign:DateSigned[=PGExcusalParent1Date]}
  ```

  Will display the date twice on a signed document – once from the standard binding, and once from the DocuSign binding. If the binding is changed to something like:

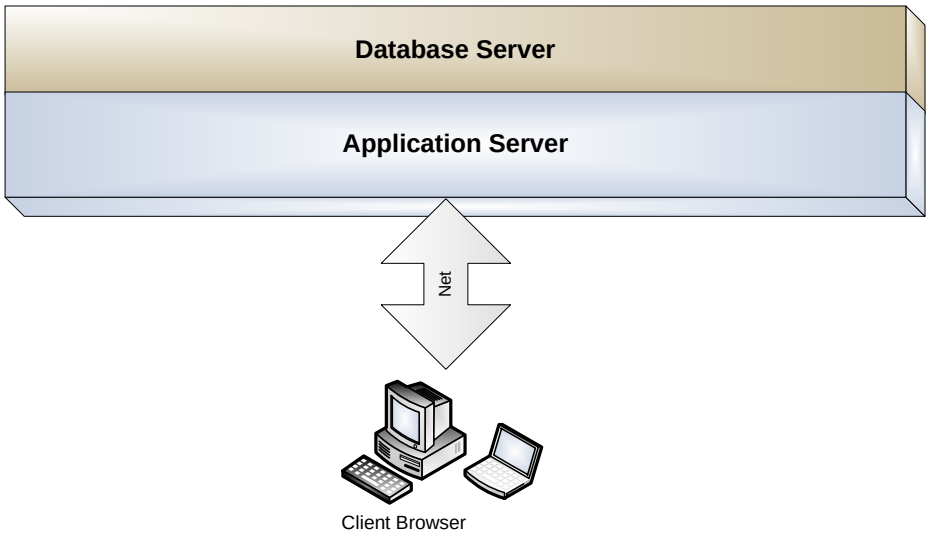
  ```
  {#IF_NOT_ESIGNATURE_ENABLED}{PGExcusalParent1Date}{#ENDIF}
   {'ParentSign:DateSigned[=PGExcusalParent1Date]}
  ```

  Then the value will display once.

- Fields directives that directly reference the profile (such as `Profile.Parent1GetLetters`) will use the latest data from the profile when rendering a document for signing. If the profile is changed after a signing request is initiated, then a document that is being signed will also change to match the profile. If this behavior is not desirable, a document field that has a data flow from the original profile field should be used in place of the direct profile field reference.

Optimizing Document Speed and User Responsiveness

To optimize the speed and responsiveness of documents, it is helpful to be aware of the level of the system architecture in which various configuration features operate. Below is a simplified system architecture diagram that is helpful for this purpose. Configuration features that interact with the database server are the slowest. Configuration features that execute in the client browser will be very fast. Therefore, whenever it is possible to switch from features that interact with the database with those that execute in the client browser (or the application server), the user experience will be faster.



The follow approaches can be used to replace configuration features that interact with the database server with those that execute in the client browser or on the application server:

- Wherever possible, replace "auto-postback", which interacts with the database server and is noticeably slow, with JSIF which executes immediately in the client browser or on the application server.

- Where possible, replace #IF statements with JSIF plus "simulate=if" keyword. A section that has many #IF statements, especially in a repeating row layout, can slow down the user experience. But JSIF plus the "simulate=if" keyword exactly simulates the behavior of an "{#IF formula}" directive without any extra interaction with the database server. With the "simulate=if" keyword, JSIF does not render any enclosing tags nor any JavaScript triggering behavior, and therefore it behaves identically to the #IF directive. Simple instances of {#IF formula} can often be converted directly to {#JSIF expression, simulate=if} with no other changes required. A section that has many #IF statements, especially in repeating rows, can run noticeably faster if they are converted to JSIF with simulate=if. Note that "simulate=ifviewor" and "simulate=ifeditor" are also supported and simulate #IFVIEWOR and #IFEDITOR respectively.

- Where possible, switch to the "fast add" feature for repeating row layouts that allow addition of rows. This will speed up the user experience greatly.

SQL Timeout Errors: The practices described below can correct certain situations where there are SQL timeout errors, deadlock errors, or sluggish performance in documents.

- **Minimize use of "dot" operator:** A lot of references to Profile.*field* in document/section actions and document forms can lead to poor performance (including SQL timeout errors and deadlock errors). The best practice is to move the values of such profile fields into the document using one-way dataflow (with automatic updating). Then these values can be referenced directly in the document. This approach has a secondary historical benefit in that it captures the original profile value fields at the time the document was finalized.

- **Optimize use of aggregate functions:** Child template calculated fields that use aggregate functions (EXISTS, COUNT, etc) can be a source of performance degradation. The system will load the value of every data and calculated field in the child template prior to rendering the repeating rows/sections. So if such calculated fields are not displayed in every single repeating row or section, they are being loaded whether or not particular values are displayed. To optimize this, such calculated fields can be marked as "Internal Value Only" which prevents the preloading of their values. Then in the HTML section, reference the calculated field using a formula directive {=*CalculatedField}* instead of a field directive {*CalculatedField*}. Note that this optimization will not yield any benefit if the calculated field is a simple calculation (no aggregates) or if it is displayed on every repeating row or section.

- **Optimize Indexes:** ClickConfiguration > Profile Types > More… > View Index Suggestions to access a screen that suggests additional quick search indexes that would make a difference based on the recent actual user workload in the database. Specifically the screen suggests specific fields where performance could be improved by assigning the quick search index property and/or the "Include with Quick Search Indexes" property. Thw "Include with Quick Search Indexes" property enables the field's value to be stored in quick search indexes without acting as the main index key.

Printing Special Document Content for Student/Parent/Guardian

A document template can be configured to allow a user to print a version for the document for a student, parent or guardian with special content added or removed. To do this, in the HTML format for any section, use the following CSS classes to mark content to be visible to staff only, students only, or parents/guardians only. Not that "staff only" is the default mode of printing and viewing for all users who are not students, parents or guardians.

- STUDENTONLY for example, content
- STAFFONLY for example, content
- PARENTGUARDIANONLY for example, content

Note that applying the above classes is enough to have the document appear as expected if student, parent, guardians are able to log into their respective portal and view the document. But one additional step is needed to allow staff to print the document for a student, parent or guardian (since when staff print documents, by default they will print the staff version of any document). In document template properties for the document template, you will need to check "Enable Print for Parents Guardians" and/or "Enable Print for Students" under print options. Once enabled, when staff use "Print Selected Sections", the popup will have a "Print For?" option that defaults to staff.

Configuring Forms

Chapter 8

HTML

The following requirements must be met to be XHTML compliant:

- All tag and attribute names must be in lower case.
- All open tags must have a corresponding close tag, or if the tag is self-contained such as
, the self-contained tag should end in />.
- Attribute values should be enclosed in quotes.
- Tags should be well formed, i.e. <label>Name</label> is not well-formed but <label>Name</label> is.

One of the most important best practices of modern web design is separation of semantic content (HTML) from presentation (CSS). Semantic content refers to the information and its meaning as stored in a particular document, and presentation refers to how that information is rendered and presented in the browser. The best practice is to remove as much presentation information as possible from the HTML document so that it focuses purely on content and meaning, and use separate CSS styles to instruct the browser how to present that information in a particular context. Following this best practice yields the following benefits:

- The volume and complexity of HTML can be significantly reduced, and so the maintenance effort is reduced as well.
- It becomes much easier to apply a consistent look and feel across documents and sections of documents.
- There will be much fewer problems resulting from multiple authors with different markup styles if the best practices are followed, and the resulting HTML markup is much leaner.

CSS

All document template sections and other configurable HTML formats occur in the larger context of a PowerSchool Special Programs page. As a result, they automatically benefit from PowerSchool Special Programs CSS defaults that are established in its master style sheet to help ensure consistency across browsers. For example, all tags are set to have zero margins with a font of "Arial, Helvetica, sans-serif". Many other commonly used tags have a padding of zero.

```
* {margin: 0; font-family: Arial,Helvetica,sans-serif; }
p, ul, li, body, form {padding: 0;}
hr {color:#999999;}
```

```
img {border:0;}
em {font-style:italic; font-weight:normal; }
strong {font-style: normal; font-weight: bold; }
table {font-weight:normal}
```

It is possible to override the above defaults or to establish entirely new tag defaults for just the configurable portion of PowerSchool Special Programs pages that is rendered from your HTML content. This is possible because PowerSchool Special Programs always renders profile form section content within a …. tag pair and similarly always renders all document template content inside a …. tag pair. With this knowledge, you can set and/or override default styles in several ways.

If you want to apply styling to ALL tags inside of the DOCCONTENT division, you can create the following style.

```
#DOCCONTENT * {font-size: 12pt;}
```

The asterisk ensures that the style is applied to all tags within content rendered from HTML formats. Note that if you omitted the asterisk, the styling would be applied to the division tag and inherited by most tags inside it, but with the notable exception of table tags; hence the asterisk is critical to ensuring that everything is affected.

You can also apply the style to certain tags anywhere they appear inside the DOCCONTENT division, as follows below.

```
#DOCCONTENT p, ul, li {font-size: 12pt;}
```

If you wish to make sure content is not included in printed output, you can use the "DISPLAYONLY" class built into PowerSchool Special Programs.

It is important to note that when a document is finalized, unlike the HTML format of the sections, the style sheet at the time of finalization is not preserved. Therefore, when changing the style sheet, changes must be made in a way that ensures backward compatibility. The style sheet is not preserved because, if every final document effectively had its own style sheet, certain functionality like bulk printing would not work correctly.

HTML Best Practices

These practices focus on separation of content from presentation.

Tags like <b>, <i> and<u> have no semantic meaning and are only used to determine appearance. Consider using in place of <b>, and in place of italics. Actually, the semantic meaning of is to "emphasize text", and that of is to "strongly emphasize text". However, PowerSchool Special Programs uses CSS defaults to make display in bold-face, and in italics.

Use <ul> and to create lists of information in a semantic way that assists screen readers and is better for the visually impaired. If you don't like the bullets, you can control that with CSS (list-style-type:none).

Avoid using tag attributes intended to control appearance, and avoid placing styling information directly in tags except for one-off exceptions. Even class attributes can be avoided to an extent by using techniques described earlier based on id. The more presentation information that stays in the style sheet, the leaner your HTML formats will be.

In many cases, it is possible to avoid using tables to control layout by instead using something like the following:

```
<div style="float:left">left-column content</div>
<div style="float:right">right-column content</div>
<div style="float:clear">Continued content occupying both columns</div>
```

Accessibility

Be mindful of best practices to enhance accessibility of forms, and at the same time, this will ensure that the forms are compliant with governmental legislation like Section 508 in the USA. One important best practice is to use semantic HTML for content and CSS for presentation as described in the previous section. When markup is laden with tags, attributes and hacks designed to have it look a particular way for current browsers, it will be much less intelligible to screen readers and other aids for people with visual challenges.

Another important practice is to test that all fields have associated labels. A simple way to test that an existing form meets this best practice is to click on all the field labels in edit mode. When you click a field label, the field should receive the input focus. If the field does not receive the input focus, the field has no associated label (or possibly you clicked the wrong text and some other text is actually the label).

In some cases, PowerSchool Special Programs automatically renders labels according to best practice. This is the case with the {field:T"label"} or {field:L"directives"} directives and when dual Yes/No checkboxes are rendered for a logical field that allows empty values (checkboxes are labeled Yes and No respectively). In other cases, you may need to supply your own label, and in order to meet accessibility standards, you must make sure your label is explicitly associated with the field. There are two ways to do this:

1. Simply wrap a label around the field, as follows:
   <label>Field Label: {fieldname}</label>

2. Make an explicit association as follows:
   <label for="$fieldname">Field Label:</label> {fieldname}

In the second approach above, note the '$' in front of the field name in the label tag. The '$' instructs the system to make any necessary adjustments to the field name that are needed to complete the association. For example, if the field is in repeating rows, the $ will instruct the system to adjust the reference to the field as needed for each repeating row. The '$' also validates the field name, such that if it not valid, it will show as a misconfiguration. Finally, it future proofs the implementation so that it will work as expected in

future platform versions. Note also that a few special cases are also supported. To label the No checkbox of a logical field that allows empty values, use the <label for="$LogicalField!"> synxtax (note the exclamation point). To label the time field of a date/time field, use the <label for="$DatetimeField#TIME"> syntax.

In a profile form section, if you want to reference a particular value of a multiple value field, then use markup like that below. If you omit the $, the association will not be made.

```
<label for="$fieldname[value]">Field Label:</label>  {fieldname[value]}
```

Best Practices to Support Translations

When configuring forms, one must be mindful to avoid certain practices that prevent the translation engine from working the way that it should. Typically, the patterns to avoid involve forcing the system to use untranslated values in translated forms. Here are things to avoid:

1. {=*KeywordField*.Description} forces the system to always display the untranslated value. But administrators will add additional columns like Description_es to hold the translated descriptions that should be displayed instead in translation mode. {=*KeywordField*} allows the system to choose the best column based on the current language. However {=*KeywordField}* should be replaced with {*KeywordField*} or {*KeywordField*:R} (to force read only) for efficiency reasons.

2. {=*KeywordField*.*OtherKeywordField*.Description} should be replaced with {=*KeywordField*.*OtherKeywordField*} to allow the system to display the appropriate description column based on the language.

3. {=SELECTBYVALUE(SortOrder, 1:"Special Education Services", 2: "Career and Technology Education Services", 3: "Related Services")} prevents translation because it hard codes English within a formula. This should be replaced with something that allows the phrases to be extracted for translation into multiple languages, such as: {#IF SortOrder =1}Special Education Services{#ELSEIF SortOrder =2}Career and Technology Education Services{#ELSEIF SortOrder =3}Related Services{#ENDIF}

4. {=IFELSE(UseFieldA=true,FieldA,FieldB} (where FieldA and FieldB are translatable long text fields) prevents translation by hardwiring the English versions of the field values into the directive. The formula language does not reference translated values of fields. Therefore this directive should be written as {#IF UseFieldA}{FieldA}{#ELSE}{FieldB}{#ENDIF}.

Field Directives & Validation Capabilities by Data Type

From a design standpoint, it is best to choose the data type for each field (e.g. character, numeric, logical, etc) based strictly on the nature of the data rather than how it will be presented to the user on a form (e.g. checkbox, textbox, dropdown, etc). However, when designing a form, control over the presentation of each field is desirable. PowerSchool Special Programs typically supports several alternative presentations for each data type thereby allowing the ideal data type to be chosen without being overly concerned with presentation limitations. Below are details on the presentation and validation capabilities for each data type. All editable fields and data types support the basic field labeling options, but the information below focuses on what is different about each data type.

- **Character:** Character fields are always presented as a single-line text box in edit mode. The width of the text box can be specified with the "W" modifier. You can specify a width in characters, for example {*fieldname*:W20}, or as a percentage of the available width; for example {*fieldname*:W99%}. In a document template, the "S" modifier specifies that the character field be spell-checked, whereas by default it is not spell checked. When not in edit mode, the character field is generally displayed as text with a few exceptions. If the character field has either the "URL" or "Email Address" field property option, it will be displayed as web or email hyperlink respectively.

  From a data validation standpoint, the following field properties will automatically enable validation of the field and rejection of invalid values: "URL", "Email Address", "Require Numeric", and "Require Alphanumeric". Other types of validation checks can be implemented by creating a section action with a triggering event of "Rollback Saving Section", and which have the "Anchor user message to first referenced field" option checked.

- **Logical:** Logical fields that do not allow empty values are always presented as checkboxes. Logical fields that allow empty values are presented, by default, as two checkboxes labeled "Yes" and "No", but can alternatively be presented as a dropdown menu by introducing the "D" field modifier. Either way, the labels associated with true and false values can optionally be configured as follows: {*logicalfield*:+"*labelforyes*"-"*labelforno*"}. The "Yes" value is normally displayed before the "No" value, but that order can be reversed by including the "N" modifier.

  When the logical field allows empty values, the two checkboxes can be positioned in two separate places on the form by using two separate directives. Use{*logicalfield*:+"*labelforyes*"} to position the checkbox for the true value, and {*logicalfield*:-"*labelforno*"} to position the checkbox for the false value.

  A group of related logical fields can be laid out in a way that enforces a requirement that the user must check at least one checkbox with the overall behavior consistent with them being a single required field. The red/yellow colors are applied and change as expected. This is configured as follows:

  1. Arrange the logical fields on the form inside of a common parent element such as , <p> or <fieldset>. Note that this common element will change color when the first checkbox is checked.

2. In the directive for each of the logical fields, include an O modifier to specify a unique group name. For example, {fieldname:L"label"O"groupname"}. The group name is simply an identifier that is used to group the checkboxes together.

- **Date and Date/Time:** A date field is presented as a textbox with an associated calendar popup when in edit mode. The "W" width modifier is supported as with the character field. The "O" modifier can be used to specify a precise output format to be used when not in edit mode. An example of an output format is "ddd, dd/mm/yyyy" which would output something like "Mon, 02/28/2012". The following character placeholders can be used in the overall format:

    - d (day, 1-31), dd (day, 01-31)
    - ddd (day of week, Mon-Fri), dddd (day of week, Monday-Friday)
    - h (hour, 1-12), hh (hour, 01-12), H (hour, 0-23), HH (hour, 00-23)
    - m (minutes, 0-59), mm (00-59)
    - M (month, 1-12), MM (month, 01-12)
    - MMM (Jan-Dec), MMMM (January-December)
    - s (seconds, 0-59), ss (seconds, 00-59)
    - tt (AM/PM)
    - yy (year with 2 digits), yyyy (year with 4 digits)
    - Non-letter characters are copied to the output as is, except that \ is an escape character. For example, {MyDateTime:O"HH\h\mm"} would output a time that looks like 23h12.

  A date/time field will include an additional textbox for time. The time portion can optionally be positioned separately from the date portion using two separate directives. The {*datetimefield*:D} directive (upper-case "D" modifier) will position the date portion and { *datetimefield*:d} directive (lower-case "d" modifier) will position the time portion. Note that you cannot have an editable time component without an editable date component, although you can make the date component invisible using the "@" modifier as long as date component will always be filled with a default value.

  From a data validation standpoint, the following field properties will automatically enable validation of the field and rejection of invalid values: "No Past Dates in Draft", "No Future Dates", and "Time Required" (date/time fields only). Due to the way the future moves into the past over time, the "No Past Dates in Draft" is only enforced when the document is in draft mode (as the label suggests). The "Time Required" option is only for date/time fields. If this option is not enabled, the user is allowed to omit the time portion. Other types of validation checks can be implemented by creating a section action

with a triggering event of "Rollback Saving Section", and which have the "Anchor user message to first referenced field" option checked.

- **Numeric and Integer:** Numeric and integer fields are presented, by default, as a textbox in edit mode, and support the width modifier as with the character field. The "O" modifier can be used to specify a precise output format to be used when not in edit mode. Specific examples are given below.

    - O"c" 14.1 => $14.10, £14.10, … (currency, server culture)
    - O"X2" 15 => 0F (hexadecimal from integer only)
    - O"x4" 254 => 00ffe (hexadecimal from integer only)
    - O"n0" integer 9999001 => 9,999,001
    - O"n2" numeric 9999001.1 => 9,999,001.10
    - O"#.##" 0.3678 => .37, 0.3 => .3
    - O"0.00" 0.3678 => 0.37, 0.3 => 0.30
    - O"p" .371 => 37.1% (percentage)
    - O"p1" .37 => 37.0% (percentage)

  Numeric fields can have a "Currency" field property option that modifies the behavior of the numeric field in two ways: 1) expands capacity to support 10 digits left of the decimal point, 2) displayed as currency (e.g. $9.10) on forms.

  An integer field that has a minimum value and a maximum value can alternatively be presented as a dropdown menu of all integer values between and including the minimum and maximum. Simply include the "D" modifier. You can also specify a numeric interval along with the "D" modifier. For example, if a field has a minimum value of 5 and a maximum value of 25, you can present {5, 10, 15, 20, 25} in the dropdown using the syntax (*myinteger*:D"5"} where 5 is the interval.

  From a validation standpoint, the minimum and maximum values optionally specified in the field properties are enforced whether or not the field is presented in the form of a dropdown menu. Other types of validation checks can be implemented by creating a section action with a triggering event of "Rollback Saving Section", and which have the "Anchor user message to first referenced field" option checked.

- **Short Text, Long Text:** You can specify a width modifier just like with the character field. But in addition, you can specify a height (in lines of text) using the "H" modifier, for example {longtextfield:W99%H9}. Depending on the context, the "S" modifier can also be used to modify the behavior of the text box as follows:

- If used in a profile form with a short-text field, a spell-check link will be included next to and for the field when in editing mode.

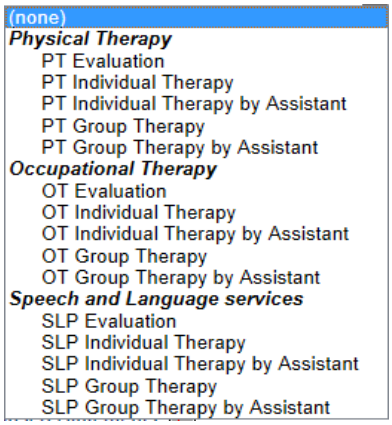
- If used in conjunction with a long text field in a document template, an "Insert Statements" link will appear even if the field presently has no public statements.

- **School Year:** Fields of type "school year" are always presented as dropdown menus. If used in conjunction with a school year type data field, the *filterspec* will be a range of years relative to the current school year. For example F"-1,+1" will include last year, the current year, and the next school year. F"-10,0" will include the last ten years and the current year, but not future years.

- **Keyword Selection:** Keyword selection fields by default are presented as dropdown menus in edit mode. If the keyword table has either a Name or Description column, values from that column are displayed in the dropdown; otherwise, the values from the Keyword column are displayed. Normally, when values from a Name or Description column are used and those values are longer than 25 characters, the description may be truncated at the first left parenthesis character to remove excessive detail. Including the N modifier prevents this truncation.

  A keyword selection field can also be presented as mutually exclusive checkboxes using the "D" modifier. The checkboxes can be configured to be stacked horizontally, vertically or in a grid. Simply specify the directive as {keywordfield:DW###) where ### should be replaced by the desired number of columns. If this number is equal to or greater than the number of keywords, this will result in a horizontal layout. If the W modifier is omitted, the default number of columns is one resulting in a default vertical layout. Any number between results in a grid layout.

  The "F" modifier is used to filter the keywords by specifying a filter column in the keyword table. The general format is{*KeywordField*:F"*FilterColumn*"}. The dropdown will omit keywords for which the filter column is EMPTY or FALSE. If the filter column is itself a keyword field referencing a second keyword table, in addition to omitting keywords with an empty value in the filter column, the dropdown menu is automatically presented in a hierarchical format where the filter column is assumed to identify the categories for the keywords. In the example shown below, "diagnostic code" keywords are categorized by service type.



It is also possible to filter a "dependent" keyword field dropdown based on the value selected from a "master" keyword field dropdown (also known as cascading dropdown menus) using the general format as follow:

{*MasterField*:T} {*DependentField*:O"*MasterField*"F"*FilterColumn*"}

Note that there are two keyword tables involved: the one for the master field and the one for the dependent field. For the above to work properly, two conditions must be met: 1) verify that if the O modifier is omitted, the dependent field is presented hierarchically as shown in the previous screen image, 2) the filter column must point to the keyword table used for the master field (specified with the O modifier).

The F and O modifier combination will work in other situations as well. Assume that there is a "ServiceType" field (keyword dropdown) and a "DiagnosticCode" field (keyword dropdown). The keyword table containing the service types has a column named "Category" that identifies the service category corresponding to each service type. The keyword table containing the diagnostic codes also has a column named "ServiceCategory" that identifies the service category corresponding to each diagnostic code. Then directives like those shown below can be used to link the DiagnosticCode field such that the DiagnosticCode field is dynamically filtered based on the ServiceType field.

{Service:LF"Category"} {DiagnosticCode:LF"ServiceCategory"O"ServiceType"}

- **Profile Reference:** By default, profile reference fields appear as a textbox with a lookup link next to it in edit mode. As with all text boxes, the "W" modifier can be used to change the width. When not in edit mode, the name of the referenced profile is displayed. If people (students, staff) are being referenced, the name is displayed in Last, First format by default. However, the "N" modifier can be added to the directive to override this and display the name in First Last format.

  When not in edit mode, the name of the referenced profile can be displayed as a hyperlink to the actual profile by enabling the "Display as Hyperlink" field property option. When this option is enabled, the field will be displayed on any forms as a hyperlink that the user can click to open a popup window showing the corresponding profile. For example if this option is enabled for a staff profile reference field, the field will be displayed as a hyperlink that the user can click to view the staff profile.

  The "F" modifier can be used to specify a filter for the profiles displayed in the lookup popup window. The format is {*profilereference*:F"*formula*"}. The formula is a PowerSchool Special Programs formula the confines the results available in the lookup popup window. The formula should be in the context of the profile type being looked up. When a filter is used within a document, in some circumstances, it is possible to embed information from the document in the formula using a placeholder enclosed with $ characters. The placeholder can be a name of a document field linked to the same section. Alternatively, it can be the symbol "Profile" which embeds a reference to the profile of the document. For example, in a student document, the directive below filters the lookup window to any teachers of any classes the student is in.

```
{Teacher:LF"EXISTS(ClassStaffRoster,
  EXISTS(ClassStudentRoster,Student=$Profile$))"}
```

Note that the profile filter formula only applies to the popup selection window. Because the end user can enter an ID directly, it also necessary to have a section action with a triggering event of "Rollback Saving Section" to validate the value.

Using JavaScript and the DataFormAPI

There are two basic methods of embedding JavaScript functions in document and profile forms (edit mode only): 1) form level events and 2) field level events. The JavaScript functions you write can call the built in "Data Form API" to interact with the form in a way that is largely protected against "breaking" in future versions of PowerSchool Special Programs. When configuring document templates, the JavaScript code for any given template section is maintained in the JavaScript configuration tab for that section. In profile forms, you must surround any JavaScript functions in script tags and embed that in the HTML of a section (in Version 15.0, this will be transitioned to a JavaScript tab like in document templates). In the remainder of this section, there are three main topics: form level events, field level events, and the Data Form API.

Form Level Events: If you define the functions listed below, PowerSchool Special Programs will automatically recognize and call them at the appropriate time when the user is editing documents and profiles.

- OnFormLoad() gets called once the form is fully loaded.
- OnFormSubmit() gets called when the form is being submitted for saving. The function can conditionally stop the submission of the form by returning a false value.
- OnFormGetCurRoot() is a special function that can be defined for a repeating document section for which the user will select statements from the curriculum. This function is called when the user clicks the "Select from Curriculum" button. It can return the desired curriculum root or an empty string to revert to the default behavior. It can also return a string in the format "*-message"* to display the user an alert message rather than opening the curriculum (perhaps telling the user to take some other action first). Finally, it can return "*=curroot"* which has the effect of directing the user to the specified curriculum, and hiding any curriculum dropdown which might allow the user to switch to another curriculum.

Below is a very simple example of how form submission can be stopped after prompting by the user. Once the Data Form API is covered, you will see how you can implement richer examples.

```
function OnFormSubmit()
{
        return confirm('Are you sure you want to submit?');
}
```

OnFormLoad and OnFormSubmit are also supported in service capture. For a profile form section used in service capture, the OnFormSubmit function can receive a bSchedulingServices parameter, which will be true only when the user is scheduling new services in the future (see below). When an HTML layout with this kind of OnFormSubmit is used outside of the service capture context, the bSchedulingServices parameter will be undefined.

```
function OnFormSubmit(bSchedulingServices)
{
        if (bSchedulingServices) {
                return confirm('Are you sure you want to submit scheduled services?');
        }
        else {
                return true;
        }
}
```

Field Level Events: To implement field level events, the "J" modifier is used in a field directive to specify a JavaScript function be called when the field's value changes during editing. For fields involving a textbox, the JavaScript function will be called when the field looses the user focus (e.g. user tabs out of field). In the field directive, you specify the JavaScript function as in the example {Goal:W100H5J"api:OnChangeGoal"} where OnChangeGoal is the name of the function (note the "api:" prefix which will be discussed later). The name of the field that changed is passed as a parameter to the function so that the new value of the field can readily be obtained using the Data Form API (see example function below). Also, if the field in question has an auto-postback, the JavaScript function will be called just before the auto-postback occurs.

```
function OnChangeGoal(strFieldName) {
  alert('New value is: ' + DataFormAPI.getFieldValue(strFieldName));
}
```

Field level events can also be used in any repeating row scenario. In this case, the notification function receives a second parameter identifying the specific row of the field that changed, and in turn this "row ID" can be passed to API functions as illustrated below.

```
function OnChangeGoal(strFieldName, strRowID) {
  alert('New value is: ' + DataFormAPI.getFieldValue(strFieldName, strRowID));
}
```

In the example directive {Goal:W100H5J"api:OnChangeGoal"}, the name of the JavaScript function has an "api:" prefix. If the "api:" prefix is omitted, the function will be called with a very different parameter, namely a reference to the DOM object that triggered the change. This is a legacy approach that will not be

supported in future versions of PowerSchool Special Programs since it requires internal knowledge of details that may change in future versions. So it is recommended that the "api:" prefix always be used.

Data Form API: The API is a set of built-in functions that your JavaScript functions can call to interact with the form in a way that is unlikely to break in future versions of PowerSchool Special Programs. The API also enables many interesting scenarios that are not possible without it. It is important to note, however, that the API functions are designed to work with editable fields on the form. If you wish to use the API to write to a field but not expose the field as editable to the end user, the field directive modifiers ^ and @ provide alternative methods for that (see the list of field directive modifiers in Appendix B).

Before getting to the API functions, it is helpful to define certain common parameters that appear across many of the build in functions. These parameters are defined below:

- `strFieldName`: You pass in the name of a field using this parameter. If you are working with a multiple-value field in the context of a profile, specify strFieldName using the format *fieldname*[*valuename*].

- `strRowID`: When applying the API to repeating rows, you will supply this parameter to identify a particular repeating row. If not applying the API to repeating rows, this parameter should be omitted. Note that it is generally the last parameter in each function so that it can be omitted without affecting other parameters. When using the J modifier with a repeating row field, the row ID is passed to the notification function as a second parameter making it available for use in API functions. Another way of obtaining a row ID is to add a special purpose button directive that calls a javascript function you specify that receives the row ID as its only parameter. The directive will have the format {*<ROWBUTTON>:J"*FunctionName*"L"*Label*"}. Supply the function in the JavaScript tab such that it takes a single parameter, for example: function MyRowFunction(rowID).

- `strChildTemplateID and strParentRowID:` Some functions operate on a repeating row layout (multiple repeating rows), and such functions include strChildTemplateID as a parmeter to identify the underlying child template. In most situations strChildTemplateID is the same Child Template ID seen in the child template properties. The one exception to this is the scenario of repeating rows within repeating sections. In this case only, strChildTemplateID should be specified using a format of *OuterChildTemplateIDT$InnerChildTemplateID*. This format should not be used in the context of repeating rows nested in repeating rows. If the intent is to operate on the inner repeating row layout of nested repeating rows, such functions only operate on the inner repeating rows of a single specified parent row. So in this case, the parent row ID is specified in the strParentRowID parameter. In all other situations, strParentRowID should be omitted.

The API functions are listed below. It should be noted that JavaScript is a case sensitive language so generally it is advisable to match the case of configured names in PowerSchool Special Programs.

- `DataFormAPI.getParentRowID(strRowID)`
  `In a nested repeating row situation, this returns the row ID of the outer row corresponding to the inner row specified by strRowID.`

- DataFormAPI.fieldExists(strFieldName, *strRowID*)
  Returns true/false depending on whether the specified field exists on the form as an editable field.

- DataFormAPI.fieldIsVisible(strFieldName, *strRowID*)
  Returns true/false depending on whether the specified field exists on the form as an editable field that is also visible on the form. The typical reason a field might exist but not be visible is that the field is currently hidden by a JSIF directive.

- DataFormAPI.setFieldFocus(strFieldName, *strRowID*)
  Sets the user focus to the specified editable field, and if necessary, scrolls the field into view.

- DataFormAPI.scrollFieldIntoView(strFieldName, bAnimate, *strRowID*)
  Scrolls the specified editable field into view. If bAnimate is true, the field will be flashed to call attention to it.

- DataFormAPI.getFieldDataType(strFieldName, *strRowID*)
  Returns a string indicating the PowerSchool Special Programs data type of the specified editable field. This can be one of the following: ID, CHAR, LOGICAL, DATE, DATETIME, NUMERIC, INTEGER, SHORTTEXT, LONGTEXT, SYEAR, IMAGE, FILE, KEYWORD, or PROFILE.

- DataFormAPI.getFieldValue(strFieldName, *strRowID*)
  Returns the value of the specified editable field as a JavaScript value. A table of information on how PowerSchool Special Programs data types are represented as JavaScript values is found below this list of functions. As explained in the table, some data types return a question mark string "?" if the field contains an invalid value (integer, numeric, date), and other data types require a "J" modifier to equip the field for access by getFieldValue and setFieldValue. Also note that if the field is a lookup in combination with a non-lookup, this function will get the value of either the lookup or non-lookup depending on what is selected.

- DataFormAPI.getIsNonLookupSelected(strFieldName, *strRowID*)
  When the directive specifies a lookup/non-lookup combination as in the directive below, this function returns whether the non-lookup is selected. Be sure to specify the lookup field name and not the non-lookup field name.
  {LookupField:O"NonLookupField"J"api:onChangeFunction"}.

- DataFormAPI.setFieldValue(newValue, strFieldName, *strRowID*)
  Sets the value of the specified editable field with a JavaScript value. You can always pass null in the newValue parameter to clear a field. A table of information on how PowerSchool Special Programs data types are represented as JavaScript values is found below this list of functions. As explained in the table, some data types require a "J" modifier to equip the field for access by getFieldValue and setFieldValue.

- DataFormAPI.getFieldKeywordTableValues(strFieldName, *callback_function, strRowID*)
  Works with a keyword fields only and produces a JavaScript object with all the field values in the

keyword table row associated with the specified field's current value. For the callback function parameter, pass the name of another function that will receive the keyword table row values once the values are retrieved. This is an important point. The getFieldKeywordTableValues method does not immediately return with the keyword table row values. Rather the request to obtain the keyword table row values is initiated. When the results are ready, the callback function is called and passed three parameters as follows: strFieldName, keyword table row values (as a single object), and strRowID if provided. For example, if the keyword table has field/column named "Category" and "keywordrow" is the parameter containing the results, then keywordrow.Category will reference the value of the category field.

For example, the following line of Javascript, obtains the Description associated with the keyword selected in the keyword field named 'KeywordField', and writes that to a character field named 'TextField'.

```
DataFormAPI.getFieldKeywordTableValues('KeywordField',
        function(strFieldName, keywordRow) {
                DataFormAPI.setFieldValue(keywordRow.Description, 'TextField');
        } )
```

- `DataFormAPI.getChildTemplateID(strRowID)`
  Returns the child template ID of the specified row. This can be used to obtain an appropriate strChildTemplateID parameter value for the functions that follow.

- `DataFormAPI.getRowIDsWithVisibleField(strChildTemplateID, strFieldName, strParentRowID)`
  Returns an array of row IDs for all repeating rows for which the specified field exists and is visible. strChildTemplateID identifies the repeating rows, and strParentRowID identifies the parent row in a nested repeating row situation (see the information on parameters prior to this list of functions).

- `DataFormAPI.getVisibleFieldNamesInRow(strRowID)`
  Returns an array of field names for fields that are visible in the specified row.

- `DataFormAPI.applyNotificationFunctionToFieldsInRows(strChildTemplateID, strFieldName, notificationFunction, strParentRowID)`
  If you have written a field notification function for use with field level notifications in a repeating row scenario, you can use this function to call that notification function for all instances of that field across all rows in which the field exists and is visible. Typically, this is called from OnFormLoad to make sure the function is called once initially for each repeating row. Tbe notificationFunction parameter is simply the exact name of the function (no quotes), which is case sensitive.

- `DataFormAPI.addRepeatingRow(strChildTemplateID, strParentRowID)`
  Use this function to add a repeating row to a repeating row layout. "Fast Add" must be enabled for this to work. Note that the function returns the row ID of the newly added row so that the API can subsequently be used to write values to its fields.

- `DataFormAPI.selectMsgGroup(strCallbackFunctionName)`
  Calling this function pops up the screen that allows the user to select a user messaging group. You pass the function the name of a "call back" JavaScript function that you write. When the user selects a messaging group, the call back function is automatically called and passed a single parameter, namely an array of User IDs that are in the message group. Note that the array of User IDs may contain User IDs for students, parents, guardians, etc. Your method can focus on staff User IDs by ignoring any User ID that does not start with the letter U. At the end of this section, you will find example functions that illustrate an interesting scenario using this method.

- `DataFormAPI.invokeLookup(strFieldName, strRowID)`
  If the specified field has a lookup link, this programmatically invokes the lookup link.

- `DataFormAPI.invokeNonLookup(strFieldName, strRowID)`
  If the specified field has a non-lookup link, this programmatically invokes the lookup link.

- `DataFormAPI.validateAddress(searchString, providedAddress)`

  Calling this function pops up the screen that allows the user to select address. You pass address string to be searched against the MAR API and providedAddress in given format to be displayed on the modal popup.

```
var providedAddress= {
  "StreetName": "CHESAPEAKE STREET",
  "StreetName2": "",
  "Quadrant": "SE",
  "City": "WA",
  "State": "DC",
  "Zipcode": "20032",
  "Ward": "8"
}
```

  Pop up contains two butons '`Use recommended address`' and '`Use the address I provided`'. Click events for both these buttons should be handled in 'model'. If user selects a recommended adderess it will be avialable in a global variable called selectedAddress.

```
SelectedAddress:
 {
    "MarId": "24445",
    "FullAddress": "425 CHESAPEAKE STREET SE",
    "AddressNumber": "425",
    "AddressNumberSuffix": null,
    "StreetName": "CHESAPEAKE",
    "Quadrant": "SE",
    "Zipcode": "20032",
    "Status": "ACTIVE",
    "Ward": 8,
    "StateCode": "11",
    "StateAbbreviation": "DC"
}
```

  Signature of the click events:

```
UseRecommendedAddress_OnClick = function() {
```

```
//Set address fields from SelectedAddress object.
//Operations to be done on click of ‘Use recommended address’ button.
DataFormAPI.closeModalPopup();
        return false;
  };

UseProvidedAddress_OnClick = function() {
//Operations to be done on click of ‘Use the address I provided’ button.
DataFormAPI.closeModalPopup();
        return false;
  };
```

Field Value Representation in JavaScript

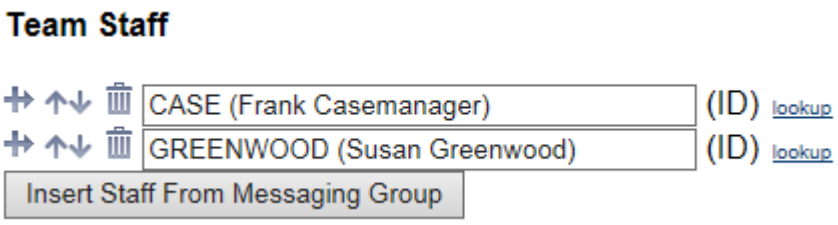
When using the API, it is important to know how PowerSchool Special Programs field values are represented as JavaScript values in the API. These are the types of values that will be returned from getFieldValue and that you will pass to setFieldValue.

PowerSchool Special Programs Data Type	JavaScript Value
ID, Character, Short Text, Long Text	JavaScript string
Logical	JavaScript bool (true/false) or null for empty value
Date, Date / Time	JavaScript Date object or null for empty value (see http://www.w3schools.com/jsref/jsref_obj_date.asp). Note: getFieldValue will return "?" string if field contains an entry that is not a valid date value.
Numeric	JavaScript float or null for empty value. Note: getFieldValue will return "?" string if field contains an entry that is not a valid numeric value.
Integer	JavaScript int or null for empty value. Note: getFieldValue will return "?" string if field contains an entry that is not a valid integer value.
School Year	JavaScript int (containing the initial year) or null for empty value
Image	Javascript string containing binary data in base 64 format or null for empty value.
File	Not supported by API.
Keyword	Javascript string containing the keyword, or null for empty value. To use a keyword field with getFieldValue and setFieldValue, the field must have a "J" modifier. Although the "J" modifier is normally used to specify a JavaScript notification function, that is not necessary. A plain "J" modifier is adequate.

Profile	Javascript string containing the profile ID or null for empty value. Such values may also be in the format *ID (Name)*. The name part is treated as informational and is ignored, for example, by setFieldValue. To use a profile reference field that is presented as a dropdown menu with getFieldValue and setFieldValue, the field must have a "J" modifier. Although the "J" modifier is normally used to specify a JavaScript notification function, that is not necessary. A plain "J" modifier is adequate.

Exposing Value of PowerSchool Special Programs Formulas to the API: If you wish to access the value of a PowerSchool Special Programs formula from JavaScript, you can use the {*=jsvarname=formula}* variation of the basic formula directive. This directive exposes the value of the formula to JavaScript in a format consistent with the Data Form API. The value is accessed from JavaScript code via a reference to window.form$ *jsvarname*. You can also configure a directive with a formula in the context of the Globals profile using the format {=@$*jsvarname=globalsformula*}.

Example Scenario: Consider a scenario where a document has repeating rows that allow the user to select any number of staff. But you also want to provide the user a button that brings up the user message group selection window and allows the user to select a message group. Upon selection, the message group staff users are inserted into the repeating rows, but in a way that avoids duplicate staff.



The following assumptions are in place:

- The child template has a child template ID of 'StaffCT'.
- The staff profile reference field within the child template is named 'StaffRef'.
- The repeating row layout utilizes the "fast add" approach.

The following HTML markup reproduces the editable list of staff shown above including the button.

```
<p><b>Team Staff</b></p>
<br />
```

```
{^StaffCT}
{#IFEDIT}
<button onclick="DataFormAPI.selectMsgGroup('InsertStaff');return false;">Insert Staff From Messaging Group</button>
{#ENDIF}
```

Note that the button calls the API function with a callback function named 'InsertStaff'. If a button is specified this way in the HTML, it is important that the onclick method returns false at the end, and that it only appear in edit mode. The definition of the InsertStaff method is shown below. Note that this also uses a number of API calls:

```
function InsertStaff(astrUserID) {
    var strChildTemplateID = 'StaffCT';
    var strFieldName = 'StaffRef';
    var astrRowID = DataFormAPI.getRowIDsWithVisibleField(strChildTemplateID, strFieldName);
    // Process each user id, but only staff users beginning with letter U
    for (var idxUser = 0; idxUser < astrUserID.length; idxUser++) {
        var strUserID = astrUserID[idxUser];
        var idxOpenParen = strUserID.indexOf('(');
        if (idxOpenParen > 0) strUserID = strUserID.substr(0, idxOpenParen);
        strUserID = strUserID.trim().toUpperCase();
        var bFound = false;
        var strRowID_Empty = null;
        // Search for staff ID in current rows, but also find first empty row in case
        //      it can be used if staff ID is not found.
        for (var idxRow = 0; idxRow < astrRowID.length; idxRow++) {
            var strUserID_Check = DataFormAPI.getFieldValue(strFieldName, astrRowID[idxRow]);
            idxOpenParen = strUserID_Check.indexOf('(');
            if (idxOpenParen > 0) strUserID_Check = strUserID_Check.substr(0, idxOpenParen);
            strUserID_Check = strUserID_Check.trim().toUpperCase();
            if (strUserID == strUserID_Check) {
                 bFound = true;
                 break;
            }
            else if (strUserID_Check == '' && !strRowID_Empty) {
                strRowID_Empty = astrRowID[idxRow];
            }
        }
        if (!bFound) {
            // If staff ID was not found, then use empty row if available, eitherwise insert new row.
            if (!!strRowID_Empty) {
                DataFormAPI.setFieldValue(astrUserID[idxUser], strFieldName, strRowID_Empty);
            }
            else {
                var strRowID = DataFormAPI.addRepeatingRow(strChildTemplateID);
                DataFormAPI.setFieldValue(astrUserID[idxUser], strFieldName, strRowID);
            }
        }
    }
}
```

In the context of documents only, several functions are available that are not part of the API but which nonetheless can be called to invoke one of the save or cancel editing buttons. These functions are as follows:

- `doSaveDoneEditing()`: This function is quivalent to clicking the "Save, Done Editing" button.

- doSaveContinueEditing() or doSaveContinueEditing(bSkipAttemptComplete): This function is equivalent to clicking the "Save, Continue Editing" button. Normally, clicking the "Save, Continue Editing" button also checks whether the section meets all completion requirements, and if so, marks it as complete. To avoid checking for section completion, the doSaveContinueEditing function takes an optional bSkipAttemptComplete parameter for which you can pass true to skip the behavior (e.g. doSaveContinueEditing(true)).

- doCancelEditing(): This function is equivalent to clicking the "Cancel, Editing" button.

Conditional Form Logic: #JSIF Directive

The #JSIF (javascript if) directive is designed to conditionally include or exclude regions of a form using a method that is very fast and responsive to end users. #JSIF supports a conditional expression syntax that is lean and performant on both the server and in the browser. There is a similar #IF directive, described in the next section, that can evaluate much more complex conditions than #JSIF, but for performance reasons, should only be used when #JSIF cannot meet the logical requirements.

The basic syntax of #JSIF is given below. Nesting and #ELSE is supported, but #ELSEIF is not supported at the time of writing.

{#JSIF *fieldlogic*, *optionname1=value1, optionname2=value2, …*}

Content

{#ENDIF}

As the simplest case, "*fieldlogic*" in the syntax above can be the name of a single field. If the field is a logical field, the content is shown if and only if the logical field is true. The logical field name can be prefixed with a ~ character to reverse the logic and show the content only if the logical field is false. For other data types, the content is shown if and only if the field has a value, or you can precede the field name with a ~ character to show the content if the field is EMPTY. You can also reference a keyword field with the format *keywordfield.keyword* to show the content only if a specific keyword is selected. To specify multiple keywords, use the format *keywordfield.keyword1/keyword2/keyword3*. Alternatively, use the format *keywordfield.logicalfield* where *logicalfield* is the name of a logical field in the keyword table that identifies those keywords with true values in the column. If the logical field name is coincidentally also the name of a keyword, the logical field name takes precedence.

[New in 21.11.1.0] In documents, you can reference DocStatus using formats such as DocStatus.Review, DocStatus.Draft/Review, or ~DocStatus.Final.

"fieldreference(s)" can also be a list of field references separated by either an ampersand (&) to designate "AND" logic or a pipe (|) to designate "OR" logic. Any field reference can be preceded with a ~ character to negate the logic for that one field. For example {#JSIF checkbox1&checkbox2&~checkbox3} will show the content if checkbox1 and checkbox2 are checked and checkbox2 is not checked. If you need to mix "AND" and "OR" logic, you can use parentheses to nest the logic.

As shown in the syntax, the JSIF directive can include comma-separated options, as follows:

collapse=true (By default, the HTML formatting is rendered invisible using the CSS visibility style, but it still takes up space on the form. Enabling this option causes the HTML formatting to collapse when not enable using the CSS display=none style).

cssclass=*somecssclass* (If specified, this applies a css class to the enclosing tag as described for the tag option.)

editonly=true (If this option is enabled, the HTML formatting will only be visible in edit mode.)

not=true (Obsolete, but still supported for backwards compatibility)

simulate=if (Specifies that the JSIF directive should simulate the behavior of an "{#IF formula}" directive. In this case, the JSIF will render neither any enclosing tags nor any javascript triggering behavior, such that it behaves identically to the #IF directive. In fact, suitable instances of {#IF formula} can be converted directly to {#JSIF expression, simulate=if} with no other changes required. A section that has many #IF statements, especially in repeating rows, can run noticeabley faster if they are converted to JSIF with simulate=if. Note that "simulate=ifviewor" and "simulate=ifeditor" are also supported and simulate #IFVIEWOR and #IFEDITOR respectively.

static=true (This specifies that the content will not change state based on immediate user input. In effect, this is an optimization that instructs the system to not generate any javascript triggering behavior).

tag=span (By default the HTML formatting is enclosed in a div tag which is used to control the visibility of the content. This option allows a different tag to be specified, such as span or p.

whitespace=true (When this option is enabled, the space that the content normally occupies when shown will remain as white space when the content is not shown. This is particularly use in conjunction with the span tag to prevent the form from shifting around when the content is displayed or hidden.)

insertafter=*keyword* (This can be used immediately below a keyword field presented as a set of mutually exclusive checkboxes. The JSIF content will then be moved below the checkbox for specified keyword. The entire JSIF statement would typically be something like {#JSIF SomeKeywordField.SomeKeyword,insertafter=SomeKeyword} so that the content appears below a particular checkbox after clicking it.

The directive {-*field1,field2,field3…*} can be very useful in conjunction with JSIF. This directive, available only in documents, resets the specified fields (comma-separated list of field names) to their default values, but only if the fields are not included on the section/form at the time the user submits the form. This directive is compatible with the #IF and #JSIF directives. In the example below, the

{-OtherText} directive ensures that the OtherText field is reset to its default value if the Service field is on the form but the OtherText field is not:

```
{#IF StudentNeedsService}
    <label>Service:{TypeOfService}</label>
    {#IF JSIF Service.Other,tag=span}
        <label>Other: {OtherText}</label>
    {#ENDIF}
    {-OtherText}
{#ENDIF}
```

Conditional Form Logic: #IF Directive(s)

In addition to #JSIF, which is designed to operate dynamically both in the browser as well as on the server, there are a number of #IF style conditional directives that are executed strictly on the server. Below are the most commonly used #IF Directives(s), however, there are additional ones documented in appendix B.

Conditional Directives	Explanation
{#IFEDIT} *HTML Formatting* {#ENDIF}	The HTML formatting between the {#IFEDIT} and {#ENDIF} directives is only included when the form is used in edit mode.
{#IFVIEW} *HTML Formatting* {#ENDIF}	The HTML formatting between the {#IFEDIT} and {#ENDIF} directives is only included when the form is used in view mode.
{#IF formula} HTML Formatting {#ENDIF}	The HTML formatting between the {#IF *formula*} and {#ENDIF} directives is only included if the formula evaluates to true. If this is used in a profile form section, the formula is expressed in terms of the profile fields. If this is used in a document template section, the formula is expressed in terms of the document fields.
{#IFVIEWAND formula} HTML Formatting {#ENDIF}	The HTML formatting between the {#IFVIEWAND *formula*} and {#ENDIF} directives is included if the form is in view mode and if the formula evaluates to true. **[New in 20.11.4.0]** There is also a {#IFVIEWAND^ *formula*) version of this directive

	(i.e. with a caret character) that can be used in a child profile form to allow specification of the formula in the context of the parent profile. This allows the directive to behave as expected when the form is used to add a new child profile.
{#IFVIEWOR formula} HTML Formatting {#ENDIF}	The HTML formatting between the {#IFVIEWOR *formula*} and {#ENDIF} directives is included if the form is in view mode (formula ignored in this case), or if the form is in edit mode and the formula evaluates to true. **[New in 20.11.4.0]** There is also a {#IFVIEWOR^ *formula*) version of this directive (i.e. with a caret character) that can be used in a child profile form to allow specification of the formula in the context of the parent profile. This allows the directive to behave as expected when the form is used to add a new child profile.
{#IFEDITAND *formula*) *HTML Formatting* {#ENDIF}	The HTML formatting between the {#IFEDITAND *formula*} and {#ENDIF} directives is included if the form is in edit mode and if the formula evaluates to true. **[New in 20.11.4.0]** There is also a {#IFEDITAND^ *formula*) version of this directive (i.e. with a caret character) that can be used in a child profile form to allow specification of the formula in the context of the parent profile. This allows the directive to behave as expected when the form is used to add a new child profile.
{#IFEDITOR *formula*) *HTML Formatting* {#ENDIF}	The HTML formatting between the {#IFEDITOR *formula*} and {#ENDIF} directives is included if the form is in edit mode (formula ignored in this case), or if the form is in view mode and the formula evaluates to true. **[New in 20.11.4.0]** There is also a {#IFEDITOR^ *formula*) version of this directive (i.e. with a caret character) that can be used in a child profile form to allow specification of the formula in the context of the parent profile. This allows the directive to

	behave as expected when the form is used to add a new child profile.
{#IF_RO formula} HTML Formatting {#ENDIF}	If the formula evaluates to true, then any fields in the content are forced to be read only. **[New in 20.11.4.0]** There is also a {#IF_RO^ *formula*} version of this directive (i.e. with a caret character) that can be used in a child profile form to allow specification of the formula in the context of the parent profile. This allows the directive to behave as expected when the form is used to add a new child profile. Tip: If you want a portion of a section to be read-only for all staff users (versus ADMIN/CONSULTANT), use {#IF_RO User IS NOT EMPTY). Note that the User keyword is not empty only for staff users.
{#IF_USER *staff_formula*} HTML Formatting {#ENDIF}	If the current user meets the criteria defined in the staff-based formula (or if the user is the ADMIN or CONSULTANT), the HTML formatting is included. This is only available for profile forms, not document templates (where it would cause problems with finalized documents). Note that the "User" keyword is unncessarty and not supported in user formulas like this.
{#IF_EDITANDUSER staff_*formula*} HTML Formatting {#ENDIF}	If the form is in edit mode and also the current user meets the criteria defined in the staff-based formula (or if the user is the ADMIN or CONSULTANT), the HTML formatting is included.
{#IF_USER_RO *staff-formula* } HTML Formatting {#ENDIF}	If the current user meets the criteria expressed in the staff-based formula, the contents will be read-only.
{#IF_ADMINCONSULTANT} *HTML Formatting {#ENDIF}	If the current user is an ADMIN user, CONSULTANT user or staff user with the 'Access Admin Form Content' privilege, the HTML formatting is included. This is only available for profile forms, not document templates (where it would cause problems with finalized documents).

{#IF_ADMIN} HTML Formatting {#ENDIF}	If the current user is an ADMIN user or a staff user with the 'Access Admin Form Content' privilege, the HTML formatting is included. This is only available for profile forms, not document templates (where it would cause problems with finalized documents).
{#IF_CONSULTANT} HTML Formatting {#ENDIF}	If the current user is a CONSULTANT user, the HTML formatting is included. This is only available for profile forms, not document templates (where it would cause problems with finalized documents).
{#IF_EDITANDADMINCONSULTANT} HTML Formatting {#ENDIF}	If the form is in edit mode and also the current user is an ADMIN user, CONSULTANT user or staff user with the 'Access Admin Form Content' privilege, the HTML formatting is included.
{#IF_EDITANDADMIN} HTML Formatting {#ENDIF}	If the form is in edit mode and the current user is an ADMIN user or staff user with the 'Access Admin Form Content' privilege, the HTML formatting is included.
{#IF_EDITANDCONSULTANT} HTML Formatting {#ENDIF}	If the form is in edit mode and the current user is a CONSULTANT user, the HTML formatting is included.
{#IF_EDITANDSTAFFUSER} HTML Formatting {#ENDIF}	If the form is in edit mode and also the current user is a staff user, the HTML formatting is included.
{#IF_USERORFINAL *staff-formula*} HTML Formatting {#ENDIF}	If the current user meets the criteria defined in the staff-based formula (or if the user is the ADMIN or CONSULTANT), the HTML formatting is included. This is only available for document templates and only when the current section has the "Always Regenerate When Not Final" property.

{#IFFIELD *fieldname*} HTML Formatting {#ENDIF}	This directive is available only in profile forms and is intended to provide support for field level security. If you wish to exclude part of the form section markup if a particular field is unavailable due to field level security, enclose that part of the markup with this directive. This will conditionally include the markup only if the specified field is available.
{#IF_NO_PREVIEW} Html formatting to hide {#ENDIF}	The user caseload customize column screen displays student profile forms with as many fields as possible in the form of checkboxes. In some cases, this may result in unwanted content appearing on that screen. The new directive syntax below can be used to suppress such content on that screen and similar screens
{#IF_STAFFUSER} HTML Formatting {#ENDIF}	**[New In 21.11.2.0]** This directive, which is available only in profiles (not documents), includes the HTML formatting if and only if the current user is a staff user.

Each {#IF…} style directive has a corresponding {#ENDIF} directive and may also have {#ELSE} or {#ELSEIF…} in between. In all of these directives, the keyword immediately after the pound sign (e.g. "IFEDIT" in {#IFEDIT}) must be in upper case. Any word(s) immediately after #ENDIF are ignored so that you can document what the #ENDIF corresponds to, for example {#ENDIF-IFVIEW}.

If is useful to compare the #IF style directives with #JSIF. In fact, you will want to use #JSIF instead of #IF whenever possible because #JSIF is more performant is more performant than #IF. But it is not always possible. There are three main differences to consider:

- The #JSIF syntax for logical expressions is much simpler than the PowerSchool Special Programs formulas used in #IF directives. If the logic you need is too complex for #JSIF and requires a full PowerSchool Special Programs formula, then you will need to use one of the #IF variants.

- The #JSIF directive renders a specified tag around the content (e.g. div, p, span, etc) whereas #IF directives do not.

- Evaluating #JSIF expressions is more performant than evaluating PowerSchool Special Programs formulas for #IF directives. Use #JSIF wherever possible. The simulate keyword can be used to specify that the #JSIF directive should simulate the behavior of an "{#IF formula}" directive. In this case, the #JSIF directive will render neither any enclosing tags nor any javascript triggering behavior, such that it behaves identically to the #IF directive. In fact, suitable instances of {#IF formula} can be converted directly to {#JSIF expression, simulate=if} with no other changes

required. A section that has many #IF statements, especially in repeating rows, can run noticeabley faster if they are converted to JSIF with simulate=if. Note that "simulate=ifviewor" and "simulate=ifeditor" are also supported and simulate #IFVIEWOR and #IFEDITOR respectively.

Auto-Postback: If you have determined that #JSIF cannot be used for a situation, but still want the form to exhibit dynamic changes as soon as the user changes the value of a field, such as a checkbox, you can trigger an auto-postback. Auto-postback is exactly equivalent to the user clicking "Save & Continue" immediately after changing the value of a particular field (e.g. clicking a checkbox). To configure a field to initiate auto-postback, introduce the "A" modifier into the directive for that field. There are several considerations:

- Auto-postback is much slower than #JSIF. The form displays a "Please wait…" message to the user, but nonetheless, too many auto-postbacks in a document template can make the user experience clunky. This is why #JSIF is preferred wherever possible.

- Unlike #JSIF, auto-postback actually saves the data entered so far. The positive side of this is that frequent saving makes it less likely the user will ever loose data.

Pop-up Help Definitions and Guides

To set up pop-up help definitions or guides in a document template, first you must add a style and two JavaScript functions to the style sheet:

First add the following style definition to the style sheet:

```
.DOC_GUIDE {position:absolute; display:inline; padding:5px; border:4px ridge #0000DD;
background:#EEEEEE; font-size:10pt; z-index:9; border-radius: 10px;}
```

Next add the following javascript block to the end of the style sheet below any style block(s):

```
<script language=javascript>
function ToggleGuide( idGuide)
{
   var objGuide = document.getElementById(idGuide);
   objGuide.style.display=objGuide.style.display == 'none' ? '' : 'none';
   return false;
}
function HideThisGuide(idGuide)
{
 var objGuide = document.getElementById(idGuide);
 if (objGuide) {
   objGuide.style.display='none';
 }
 else {
   alert(idGuide + ' not found.');
 }
```

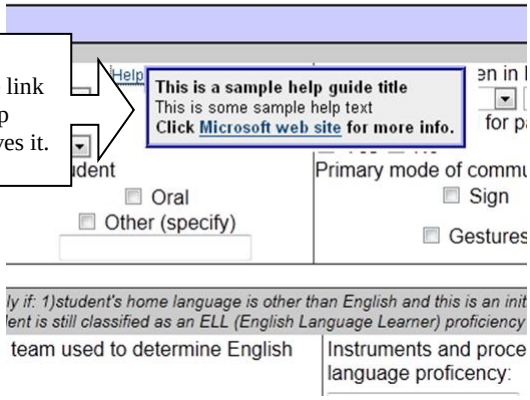
```
 return true;
}
</script>
```

Next add a block like the following for each help popup:

```
{#IFEDIT}
<a href="#" onclick="return ToggleGuide('GUIDE_XXXX')">Help</a>
<p id=GUIDE_XXXX class=DOC_GUIDE style="display:none" onclick="return HideThisGuide(this.id)">
<b>This is a sample help guide:</b><br/>
This is some help text<br/>
<b>Click <a href="http://www.microsoft.com" target=new>Microsoft web site</a> for more info:</b><br>
</p>
{#ENDIF}
```

But be sure to replace "GUIDE_XXXX" in each block with a unique identifier (e.g. GUIDE_PresentLevels) and customize the help content.

FYI: The popup help looks something like this. Clicking the link toggles the appearance of the help text. Clicking the help text removes it.



Note that it is very helpful to use style sheet entries to ensure a consistent appearance to all of your help guides.

Associating a Help Guide with a Field: If you want a help guide to disappear when the user changes the value of the field, first make sure you assign the guide identifier to use the format GUIDE_*FieldName*. Next introduce an additional JavaScript function to the style sheet as follows:

```
<script language=javascript>
function HideThisFieldGuide(strFieldName)
{
  HideThisGuide ('GUIDE_' + strFieldName);
}
</script>
```

Finally, reference the HideThisFieldGuide function in the field directive as in the following example:

```
{FieldName:J"api:HideThisFieldGuide"}
```

Using JSIF to Automatically Invoke a Help Guide: You can place a help guide inside of a JSIF block to cause the help guide to automatically appear when the conditions are met. Note the use of editonly=true in the JSIF directive which ensures the help guide only appears in edit mode. Also note the use of tag=span which ensures that the help guide appears inline with the content.

```
{#JSIF formula,editonly=true,tag=span}
<span id="GUIDE_FieldHelp" class="DOC_GUIDE" onclick="return HideThisGuide(this.id)">
   Help Guide Text Goes Here
</span>
{#ENDIF}
```

Assessment Directives

Assessment directives are preceded with an exclamation point, and allow scores and information from the assessment repository and/or curriculum to be included in a form. These directives have a general format as follows:

{!CurRoot=SAMPLE;name1=value1;name2=value2}

To include a curriculum statement description in a form, use a directive with a format like the following:

{!CurRoot=SAMPLE;Label=10101}

To include assessment total scores of various types, use a directive with a format like the following:

{!CurRoot=SAMPLE;AssessAdminID=100;ScoreType=Percentage}

{!CurRoot=SAMPLE;AssessmentName=Math Test Form B;ScoreType=Percentage,HistoryYear=2005}

AssessAdminID is the numeric ID of the assessment administration. Alternatively you can use AssessmentName, in which case, the system determines the latest assessment administration for the named assessment that the student participated in. In this case, the HistoryYear parameter can be included to limit it to particular year (more on this below). ScoreType is one of {RawPoints, MaxRawPoints, Percentage, RubricLevel, RubricSymbol, Percentile, NCE, GE, Stanine, Scaled, GAP and Other}. It can also be a combination of one of the numeric score types and either RubricLevel or RubricSymbol (example: ScoreType=Percentage/RubricLevel}.

The HistoryYear parameter is very useful in conjunction with the AssessmentName parameter. If the HistoryYear parameter is omitted, the latest score is retrieved even if it is from years ago. The HistoryYear parameter can be absolute, for example, 2005 refers to school year 2005/2006. Or if the HistoryYear parameter is zero or negative, it is assumed to be relative to the current year (e.g. 0, -1, -2, -4, -5).

There is an additional "Filter" parameter that can be used in conjunction with the AssessmentName parameter to selectively select scores based on assessment repository filter/analytical fields. This takes the form "Filter=FieldName=Value" where FieldName is the name of an filter field in the assessment

repository and Value is an appropriate value for that field. The example below only pulls out a score for which the student was in grade 05 (at the time of taking the assessment).

{!CurRoot=SAMPLE;AssessmentName=Math Test Form B;ScoreType=Percentage, Filter=Grade=05}

To include assessment section scores of various types, use a directive with a format like the following:

{!CurRoot=SAMPLE;AssessAdminID=100;SectionID=Reading;ScoreType=Percentage}

{!CurRoot=SAMPLE; AssessmentName=Math Test Form B;SectionID=Reading;ScoreType=Percentage}

In this case, SectionID is the exact user-defined ID of the section, and ScoreType can be the same types as for total scores.

In some cases, it might be useful to have the score and its label formatting not appear at all if the score is not available for the student. In this case, you can specify a format in the directive, such as Format= Score:$. The $ character is a placeholder for where the score appears in the format. If $ is omitted, the score appears after the label. Note that the format is only displayed if the specified score or value is actually found for a student; therefore, the format is very useful for making sure there are no labels with empty score values.

To include assessment curriculum statement scores of various types, use a directive with a format like the following:

{!CurRoot=SAMPLE;AssessAdminID=100;Label=10101;ScoreType=Percentage}

{!CurRoot=SAMPLE; AssessmentName=Math Test Form B;Label=10101;ScoreType=Percentage}

In this case, SectionName is the exact name of the section, and ScoreType is one of {RawPoints, MaxRawPoints, Percentage, RubricLevel}.

To include the date when an assessment was administered or taken, use a directive with a format like the following:

{!CurRoot=SAMPLE;AssessAdminID=100;Date=Y}

{!CurRoot=SAMPLE;AssessmentName=Math Test Form B;Date=Y}

When displaying numeric scores, you can also use the "Translate" parameter to include an arbitrary translation of score values into alphanumeric labels. The example below sets up a letter grade translation. Note that the labels must be listed from lowest score to highest score.

{!CurRoot=SAMPLE;AssessmentName=Math Test Form B; Translate=F,50:D,70:C,80:B,90:A}

The following brings all this together with some examples:

Score Category	Examples
Total Score	{!CurRoot=SAMPLE;AssessAdminID=100;ScoreType=Percentage} {!CurRoot=SAMPLE;AssessAdminID=100;ScoreType=Rubric} **Requires assessment rubric*
Section Score	{!CurRoot=SAMPLE;AssessAdminID=100;SectionID=Reading;ScoreType=Percentage} {!CurRoot=SAMPLE;AssessAdminID=100;SectionID=Reading;ScoreType=Rubric} **Requires assessment rubric*
Curriculum Statement Score	{!CurRoot=SAMPLE; AssessmentName=Math Test Form B;Label=10101;ScoreType=Percentage} {!CurRoot=SAMPLE; AssessmentName=Math Test Form B;Label=10101;ScoreType=Rubric} **Uses curriculum rubric*

Profile Grid Directives

Profile grid directives allow you to display child profiles of the current profile being viewed in a table/grid format. The directive can be used from within a profile form section or within a document template section. The display is always view only. When used in a document template, the information will always reflect the last time the section was saved or finalized since the output of the section is generally rendered and stored in the database. The general syntax is given below:

{%childprofiletypename: column1_expr:"column1 title", column2_expr: "column2 title",…(logicalfilter)[sort1_expr, sort2_expr, …]-modifiers}

The *childprofiletypename* can be the name of either a standard child profile type or a many-to-many (M-M) child profile type. The column expression designates what data should appear in the table, and what the column names should be. The other two major components – logical filter, and sort value list – are optional.

In the case that the child profile type is a M-M profile type, the context for the columns, filter and sort will be the other parent profile type. For example, when displaying the class roster in a student profile, the fields that you would show would be from the classes that the student is enrolled in. However, since M-M profile types can also have data fields, it is also possible to force the "context" profile type to be the actual M-M profile type. This is done by prefixing the child profile type name with =, as shown below:

{%=childprofiletypename: column1_expr:"column1 title", column2_expr: "column2 title",…(logicalfilter)[sort1_expr, sort2_expr, …]}

As a special case, it is also possible to apply the profile grid directive to another top-level profile type that is related to the current profile type via a profile reference field that is marked as a quick search field. The syntax for this is shown below:

{%=toplevelprofiletypename.profilereference: column1_expr:"column1 title", column2_expr: "column2 title",…(logicalfilter)[sort1_expr, sort2_expr, …]}

Sort expressions can be appended with **DESC** to indicate that records should be sorted in descending order. Note that a DESC can appear in each sort expression (if it is absent then ascending sort for that expressions will be used) and must appear within the square brackets.

Valid modifiers are:

V – Indicates that view icons should be presented when applicable to the user to allow viewing child records. This is not available for M-M child profile types

E – Indicates that view/add/edit/delete icons should be presented to the user to allow viewing, adding, editing and deleting child records if authorized by user security. This is not available for many-to-many child profile types.

I – Enables precise control of which icons may show. The "I" modifier may be followed by any of V (view), A (add), E (edit) or D (delete) to show the respective icons if authorized by user security. This is not available for many-to-many child profile types.

R### – Establishes a maximum number of rows to display (replace ### with the specific number).

For example:

{%=Caseload: Student:Student [Profile_Created_On DESC, Student.ID]}

The output that is rendered is a generic HTML table with known CSS classes which can be defined in the style sheet for the profile form sections or document template. The format is as follows:

```
<table class="CHILDGRID_TABLE">
<tr class="CHILDGRID_HDRROW">
        <th class="CHILDGRID_HDRCELL">column1_title</th>
        <th class="CHILDGRID_HDRCELL">column2_title</th>
</tr>
<tr class="CHILDGRID_ROW">
        <td class="CHILDGRID_CELL">row1 column1</td>
        <td class="CHILDGRID_CELL">row1 column2</td>
</tr>
<tr class=CHILDGRID_ROW>
        <td class=CHILDGRID_CELL>row2 column1</td>
        <td class=CHILDGRID_CELL>row2 column2</td>
</tr>
</table>
```

Example

{%ClassStudentRoster: ID:"Class ID", Name:"Name", Location:"Location", ClassType:"Class Type"(Language=English)[Name]}

ClassStudentRoster - this directive references the M-M profile type

ID:"Class ID" – the first column of the grid, will be titled *Class ID* and evaluates to the ID of the related class

Name:"Name" – the second column of the grid, will be titled *Name* and evaluates to the name of the related class
Location:"Location" – the third column of the grid, will be titled *Location* and evaluates to the location of the related class
ClassType:"Class Type" – the forth column of the grid, will be titled *Class Type* and evaluates to the class type of the related class
(Language=English) – filter that limits the list to classes that only have been marked as English
[Name] – sort criteria that indicates that the results should be sorted by class name

The profile grid directive can also be used to present document data in a student profile section. Data from both top level document templates and child templates can be included. The first two examples below illustrate presenting data from a top level document template (identified by DocTemplateID) and from a child template (identified by ChildTemplateID) respectively. You can also present data from group intervention documents linked to the student, and this is shown in the third example. The –V modifier in the examples below specify that view icons should be included to allow the user to drill down into the actual documents.

- {%DocTemplateID: DocHistoryYear:"Year", DocStatus: "Status" (DocHistoryYear=HistoryYear) [Document_Created_On]-V}
- {%DocTemplateID.ChildTemplateID: DocHistoryYear:"Year", DocStatus: "Status" (DocHistoryYear=HistoryYear) [Document_Created_On]-V}
- {%GroupPlan.Students: DocHistoryYear:"Year", DocStatus: "Status" (DocHistoryYear=HistoryYear) [Document_Created_On]-V}

It is also possible to create a profile grid that displays data from events (e.g. Student Events tab), as shown below. The -V modifier is not supported in this case.

- {%Events_: EventDateTime: "Date/Time", Subject: "Subject", SourceUserIDName : "User ID/Name"}

Integrating profile constraints into profile grid: In some use cases, it may be useful to have a checkbox column in the profile grid with a button in the column header that allows the end user to apply a specified profile constraint to the checked profiles. To configure this, follow the steps below:

1. Set up a profile constraint for the profile type displayed in the profile grid. The profile constraint must be set up to evaluate when the user explicitly clicks a button or as a bulk operation. The constraint must also have a constraint action of "Set Field Values".

2. The profile grid column must be an expression of type logical (true/false). This will control whether the checkbox is enabled or not for that row, which allows for conditionally enabling or disabling each row checkbox.

3. The column title must embed the name of the constraint using either of the following two formats below. The column title becomes the label of the button. If just the constraint name is provided, the constraint name is used to label the button.

   [*ConstraintName*]

   Column Title[*ConstraintName*]

Document Forms and Profile Grids: Profile grids can be used in document forms, but there are limitations. Icons are not supported at all.

Configuring Packages

Chapter 9

Frequently Asked Questions

What are packages?

Packages are a pre-defined set of modules that can be licensed in customer databases, specifically: Service Capture, SpecialEducation, RTI, Section 504, ELL, and Gifted Talented. The model configuration team, when developing a standard state or province model, can tag certain content (document templates, reports, etc.) with packages such that only customers licensed to use those packages can access that content.

How does the model configuration team indicate what packages a particular state/province model currently supports?

The model configuration team should license the supported packages (and only the supported packages) in the relavant model database (i.e. XXMODEL, XXMODELCR, XXMODELAPP).

Can the the model configuration team define entirely new packages?

No, the package definitions are embedded in the software itself.

What happens when a new customer is instantiated from a model database with respect to packages?

By default, a newly cloned customer database will have the same licensed packages that are set in the model itself. Unlicensed packages should then be removed. Packages should not be licensed to a customer that are not set in the model itself because that will have no effect and will be misleading.

How do I see what packages are licensed in a particular database from the front end?

The ability to see packages from the front end is only possible if configured in a globals profile form, and that is where it would be viewed if configured. The {#PACKAGES} directive in a globals profile form shows the packages licensed in the current database.

What content can be associated with packages?

The model configuration team can associate certain types of content with packages within a standard state/province model. Content not associated with any packages at all are available to all customers using the model. Once content is associated with one or more packages, the content may be hidden for customers that do not have at least one of the associated packages. The following types of cotent may be associated with packages.

1. **Document Templates:** Set using document template properties for a single document template or, for bulk setting, use Administration > Configuration > Document Templates > More > Set Template Properties.

2. **Keyword Tables:** Set using keyword table properties.

3. **Standard Reports:** Applies only to standard reports that are under configuration management only. Set using report properties for a single report, or for bulk setting, use Reporting > Standard Reports > Utilities > Apply Option to Reports.

4. **Profile Form Sections:** Set using profile form section properties.

What effect do packages have on users?

It is important to note that content (e.g. document templates) not associated with any packages at all are available to all customers using the model. It is only when content is associated with one or more packages that the content may become unlicensed to customers not having those packages.

- Staff users will not see unlicensed content at all, for example, unlicensed document templates will not appear in the new document dropdown menu.

- System administrators will not see the unlicensed content in the administrative area, for example, they will not see unlicensed keyword tables (to view/edit) or document templates (to set security for).

- Consultant users will see the unlicensed content listed in the configuration UI. The unlicensed content will have a "no edit" icon instead of a magnifier icon.

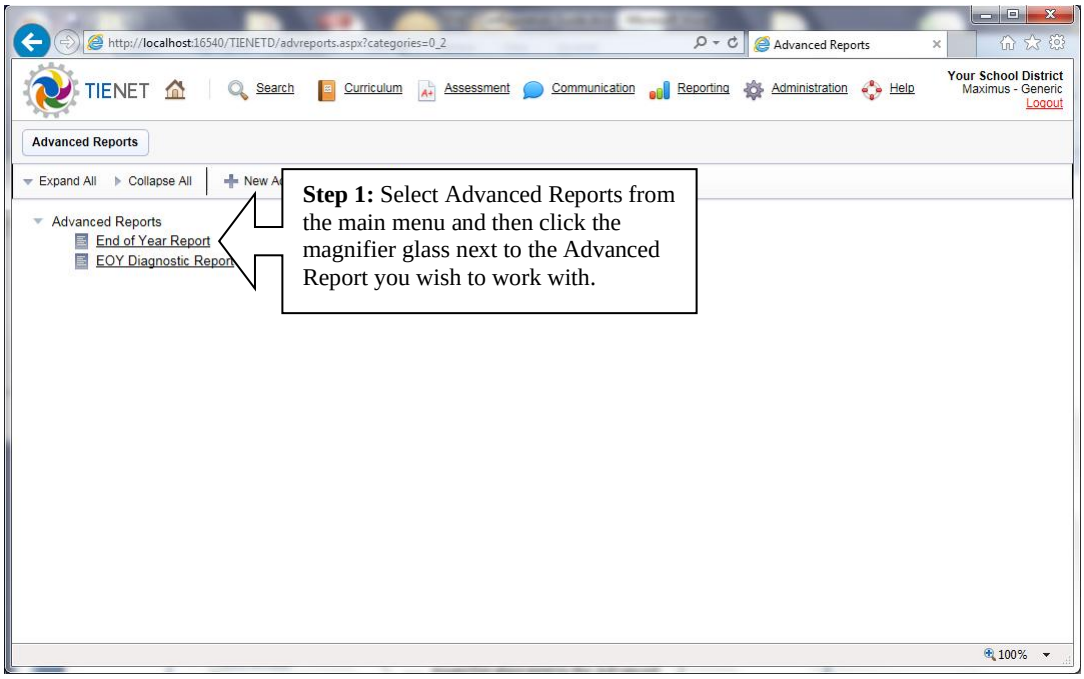
Advanced Reporting

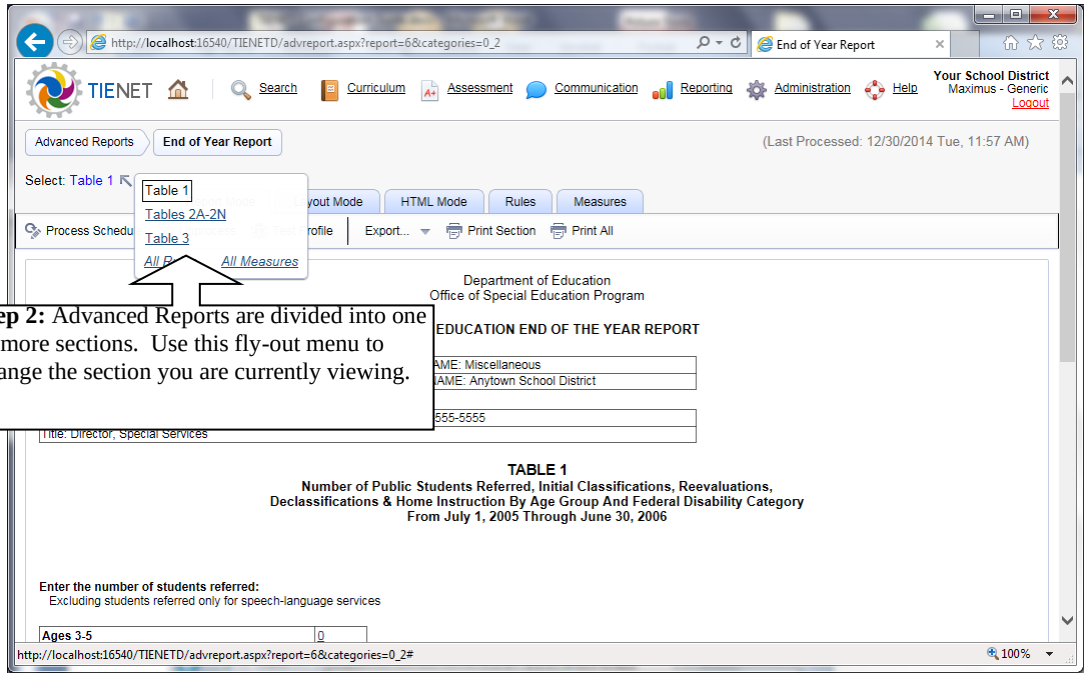
Chapter 10

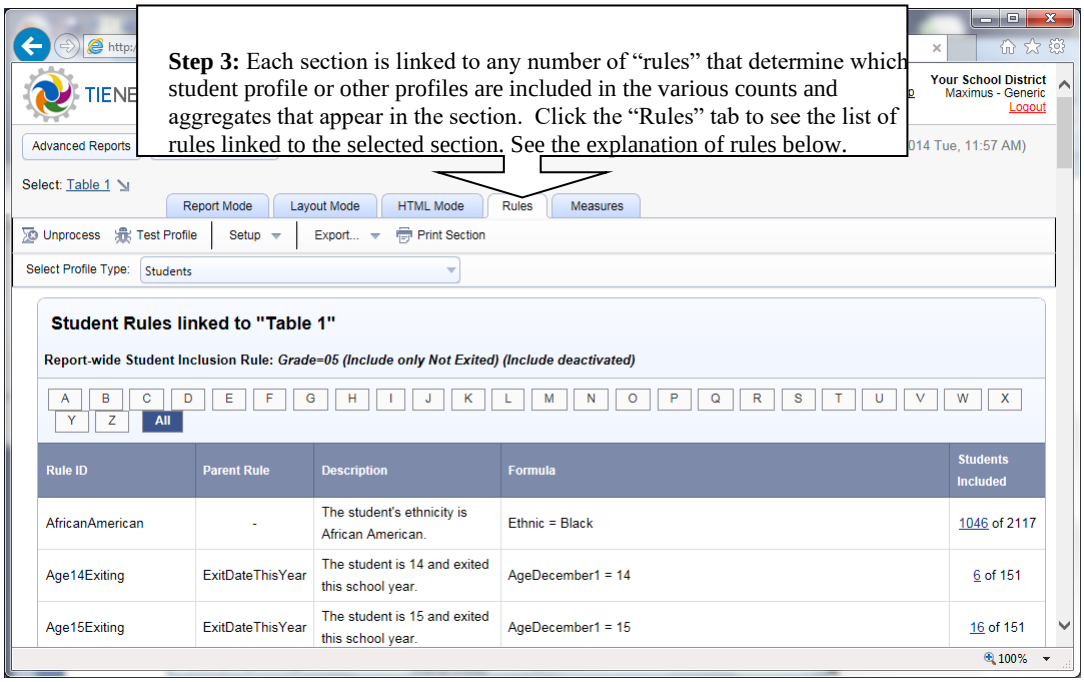
The Advanced Reporting module is designed to produce complex state and federally mandated reports using a rules-based engine that enables straightforward definition and processing of reports.

Advanced Report Overview

The best way to become familiar with the Advanced Reporting module is to "explore" an existing Advanced Report, as follows:



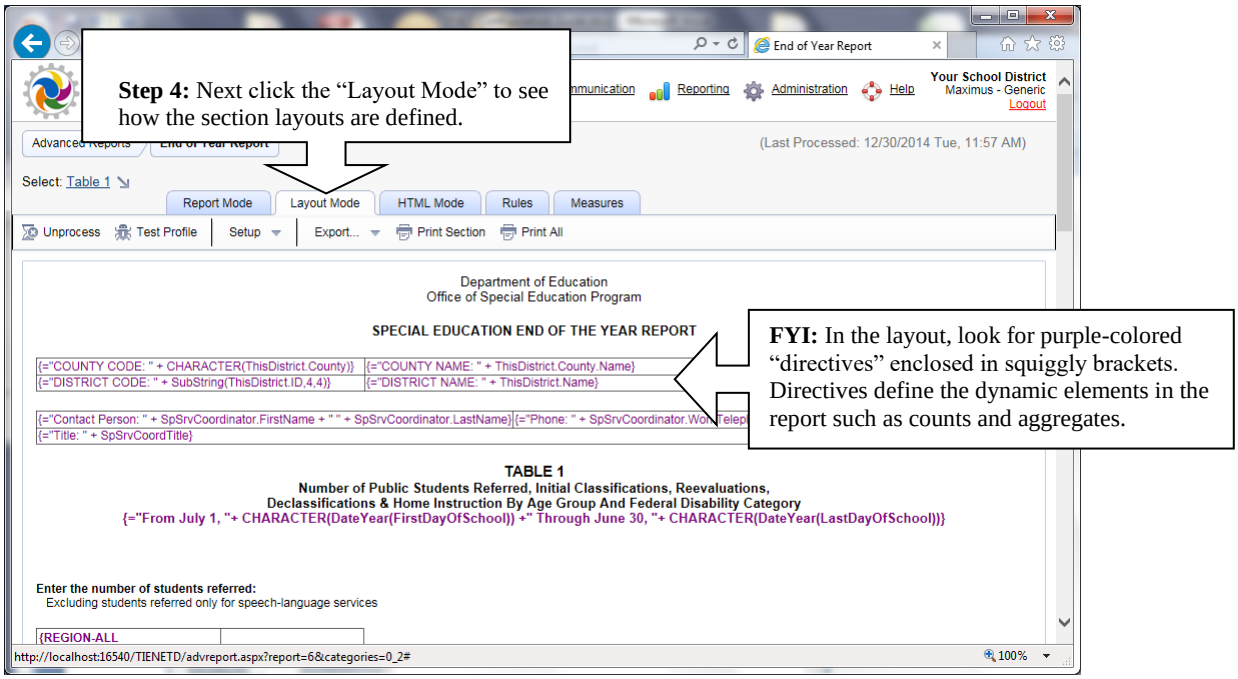


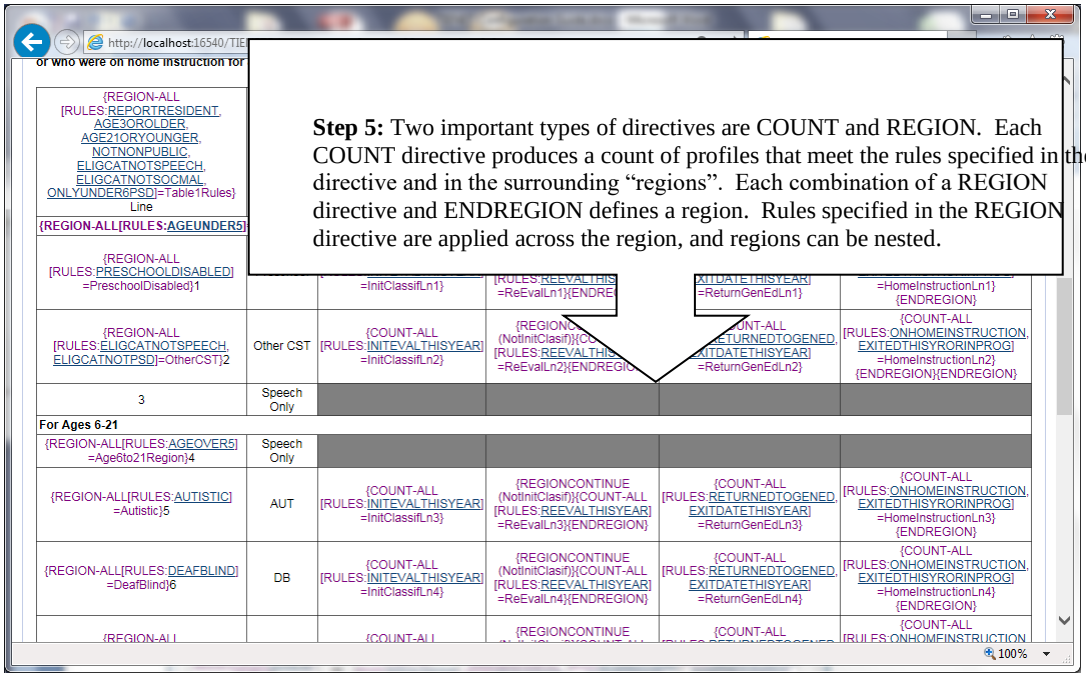


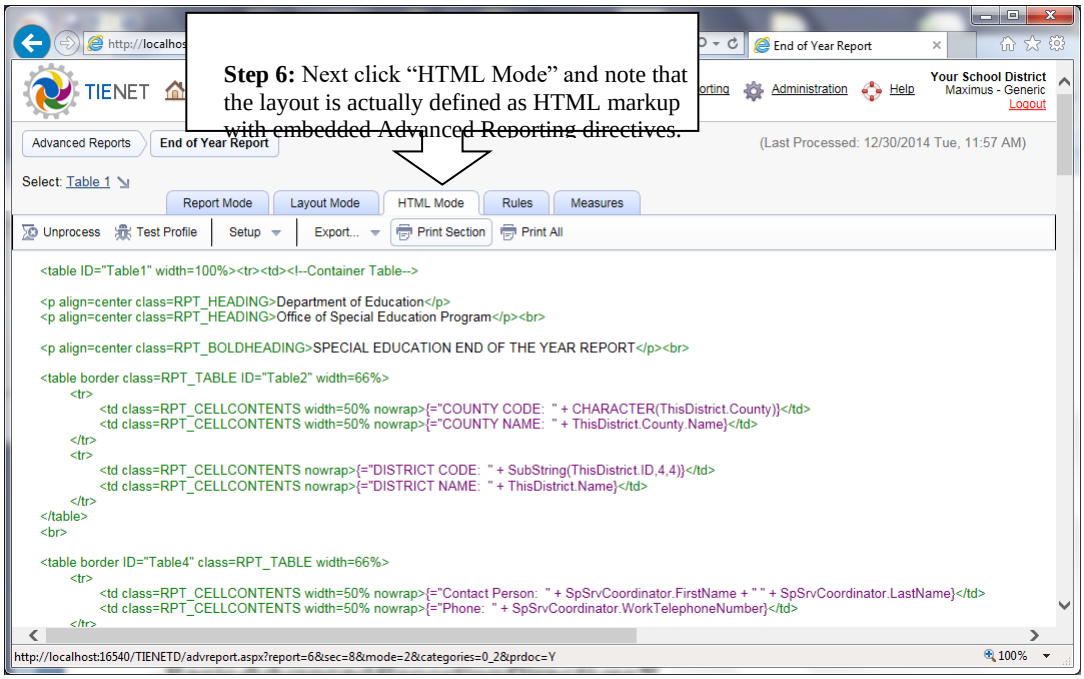
FYI – About "Rules": A rule is an essential building block in an Advanced Report. Rules are applied to profiles (e.g. student profiles) to determine which profiles should be included or counted in the various numeric cells (counts or aggregates) on the report. A rule is composed of the following:

- **Rule ID:** An alphanumeric identifier that identifies the rule and allows it to be uniquely referenced.

- **Formula:** A logical formula that determines which profiles are included in the rule.
- **Description:** The description is ideally the formula expressed in plain English.
- **Parent Rule:** Each rule can optionally have a parent rule. If a rule has a parent rule, profiles can only be included in the rule if they are included in the parent rule. Parent rules enable a hierarchy of rules to be defined, resulting in better organization and more efficient report processing.







Basic Advanced Reporting Directives

To become familiar with the various types of directives described below, you can look to see how they are used in the existing Advanced Reports:

{=globalsformula} *Specifies a computed value based on global fields. Examples:*

- **{=DistrictName}**
- **{=CountyName}**
- **{=SuperintendentName}**

{COUNT-*logic*[Rules: *ruleids*]=*cellid*} *Specifies a student count or other count. The "logic" specifier is optional and, if specified, is one of ALL, ANY, NOTALL or NOTANY. If omitted, it is assumed to be ALL. The "ruleids" is a comma delimited list of rule identifiers. Examples are as follows:*

- **{COUNT[Rules: GRADE01,GRADE02]}** *Count of all student profiles included in all the specified rules, namely GRADE01, GRADE02.*
- **{COUNT-ALL[Rules: GRADE01,GRADE02]}** Same as above, but in this case, the ALL logic is explicitly specified.
- **{COUNT-ANY[Rules: GRADE01,GRADE02]}** *Count of all student profiles included in ANY of the specified rules.*
- **{COUNT-NOTANY[Rules: GRADE01,GRADE02]}** *Count of all student profiles NOT included in ANY of the specified rules.*
- **{COUNT-NOTALL[Rules: GRADE01,GRADE02]}** *Count of all student profiles NOT included in ALL of the specified rules.*
- **{COUNT-ALL[Rules: GRADE01,GRADE02]=Grade12Cell}** This version of the directive gives the numeric cell count a Cell ID of Grade12 so that it can be referenced from other directives as described next.

{COUNT-*logic*[Cells: *cellids*]=*cellid*} *Specifies a student count or other count based strictly on inclusion in other cells which are referenced by Cell ID.*

- **{COUNT-ANY[Cells: GRADE12Cell,GRADE34Cell]}** *Count of all student profiles included in any of the other cells identified by cell ID, namely GRADE12Cell, GRADE34Cell.*

{REGION-*logicfunction*[*Rules: ruleids*]=*regionid*}*content*{ENDREGION} or **{REGION-*logicfunction*[*Cells: cellids*]=*regionid*}*content*{ENDREGION}** *Specifies that any other numeric cells included in the content of this region (between REGION and ENDREGION) should be filtered to the student profiles or other profiles included in the region's rule logic. The syntax of the REGION directive is the same as for COUNT directive. You can optionally specify a Region ID which can referenced in REGIONCONTINUE directive, described next.*

- **{REGION-ALL[Rules: SpeechImpaired,Grade01]}*content*{ENDREGION}** *Any cells between the REGION and ENDREGION directives are filtered to those profiles included in the region. Regions can be nested.*

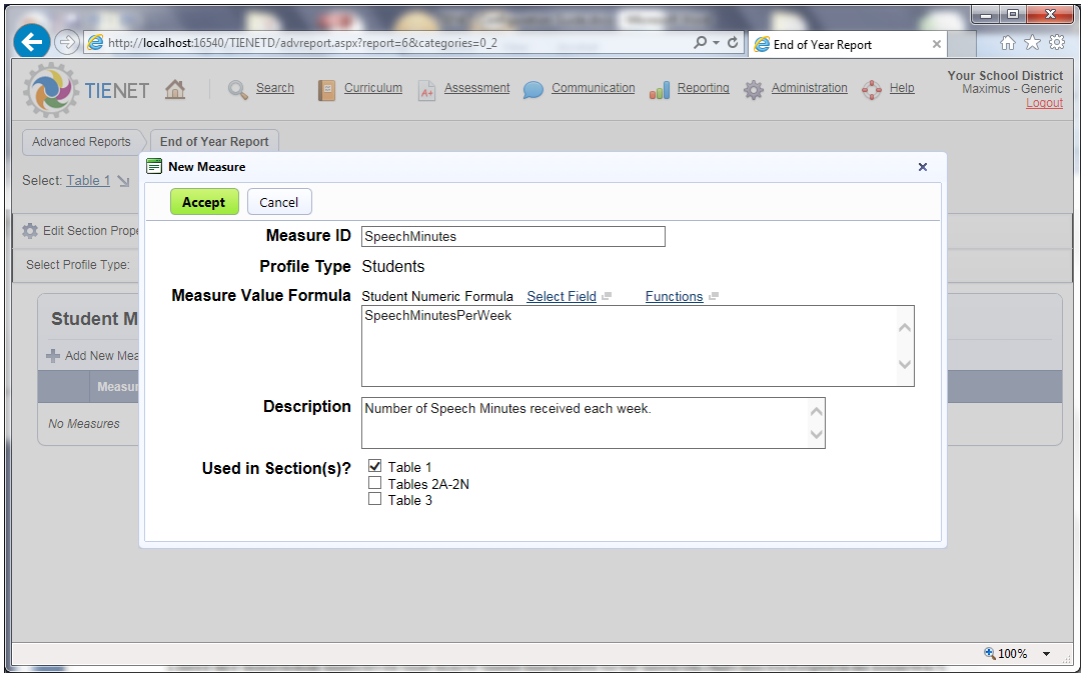
{REGIONCONTINUE(*regionid*)}*content*{ENDREGION} *This set of directives continues a previous region identified by its region ID. In effect, it allows you to create a discontinuous region.*

Numeric cells in an Advanced Report are typically organized into tabular grids. The optimal way to define regions in the HTML formatting for a tabular grid is given below. Note that within the <table> </table> tags, the Advanced Reporting directives should only appear inside <th> or <td> tags, otherwise strange display problems can result. Also note that a region is defined for the table, each row, and each column. Each column region is defined as a discontinuous region using REGIONCONTINUE. Each column region is defined in the corresponding column header, but is then re-continued in each row.

```
{REGION[Rules:ruleids]=TABLEA}
<table>
<tr>
        <th>{REGION[Rules:ruleids]=TABLEAC1}Column 1 Header{ENDREGION}</th>
        <th>{REGION[Rules:ruleids]=TABLEAC2}Column 2 Header {ENDREGION}</th>
        <th>{REGION[Rules:ruleids]=TABLEAC3}Column 3 Header {ENDREGION}</th>
</tr>
<tr>
        <td>{REGION[Rules:ruleids]=TABLEAR1}{REGIONCONTINUE(TABLEAC1}r1c1{ENDREGION}</th>
        <td>{REGIONCONTINUE(TABLEAC2}r1c2{ENDREGION}</th>
        <td>{REGIONCONTINUE(TABLEAC3}r1c3{ENDREGION}{ENDREGION}</th>
</tr>
<tr>
        <td>{REGION[Rules:ruleids]=TABLEAR2}{REGIONCONTINUE(TABLEAC1}r2c1{ENDREGION}</th>
        <td>{REGIONCONTINUE(TABLEAC2}r2c2{ENDREGION}</th>
        <td>{REGIONCONTINUE(TABLEAC3}r2c3{ENDREGION}{ENDREGION}</th>
</tr>
</table>
{ENDREGION}
```

Advanced Report Measures

Rules in combination with the directives described above allow you to produce any imaginable student counts or other counts on an Advanced Report. However, there may be cases where you need to incorporate other types of aggregates such as averages, sums, minimums and maximums. To include these types of aggregates on a report, you need one ore more measures. A measure is a numeric value defined by a numeric formula and identified by a Measure ID. The screen below shows the screen used to define a measure.



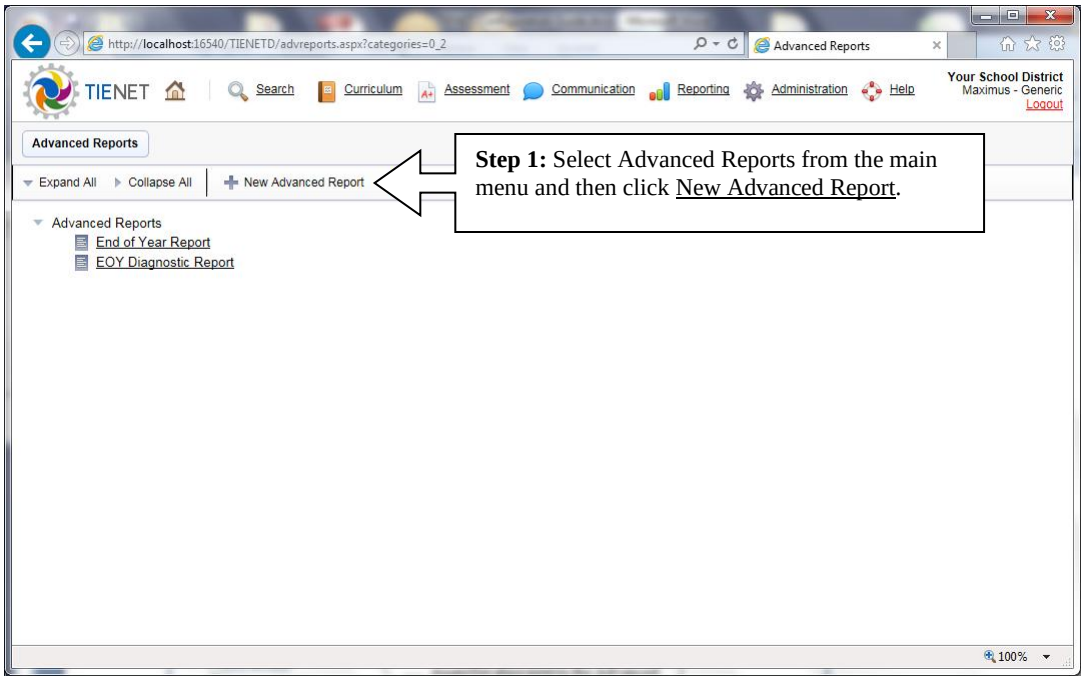
There are additional directives that allow these measures to be used on Advanced Reports as follows:

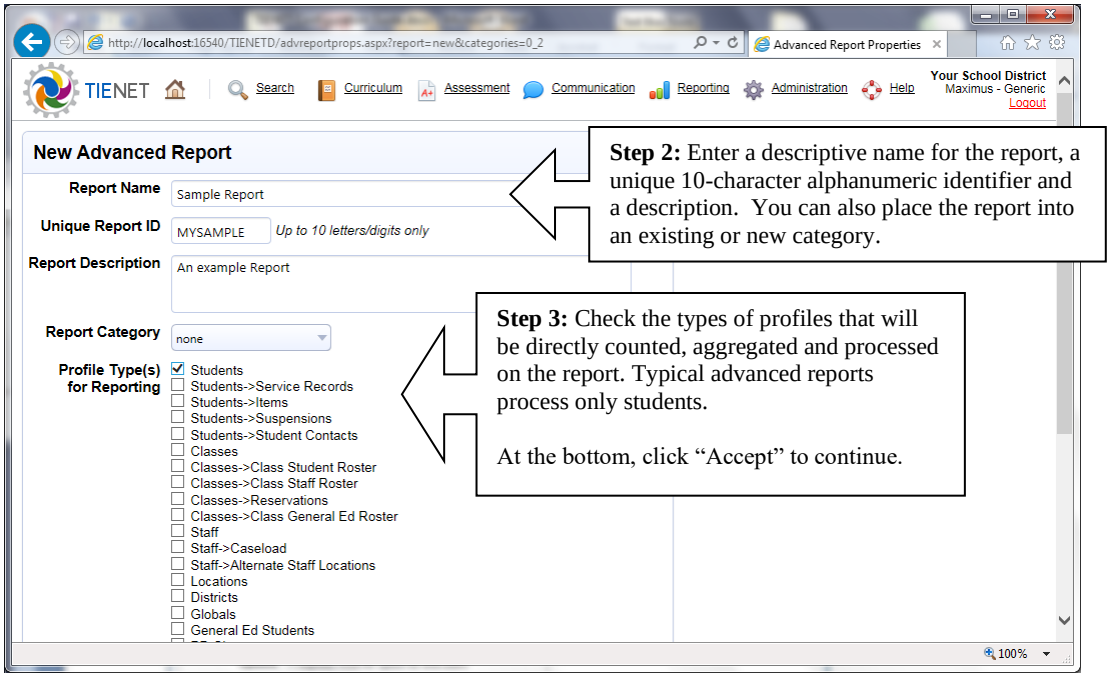
- **{COUNT(*measureid*)-*logic*[Rules: *ruleids*]=*cellid*}** or **{COUNT(*measureid*)-*logic*[Cells: *cellids*]=*cellid*}**
- **{SUM(*measureid*)-*logic*[Rules: *ruleids*]=*cellid*}** or **{SUM (*measureid*)-*logic*[Cells: *cellids*]=*cellid*}**
- **{AVG(*measureid*)-*logic*[Rules: *ruleids*]=*cellid*}** or **{AVG(*measureid*)-*logic*[Cells: *cellids*]=*cellid*}**
- **{MIN(*measureid*)-*logic*[Rules: *ruleids*]=*cellid*}** or **{MIN(*measureid*)-*logic*[Cells: *cellids*]=*cellid*}**
- **{MAX(*measureid*)-*logic*[Rules: *ruleids*]=*cellid*}** or **{MAX(*measureid*)-*logic*[Cells: *cellids*]=*cellid*}**

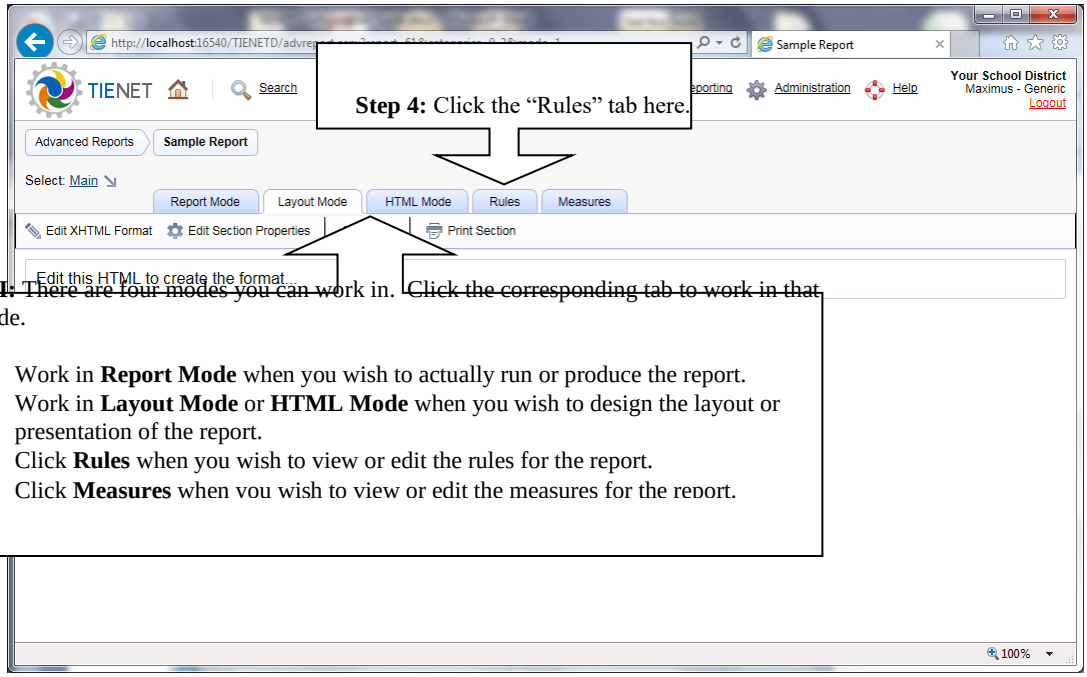
The syntax of these directives is very similar to the syntax for the basic COUNT directive. The new variation of the COUNT directive shown here (with the *measureid* parameter) is like the basic COUNT directive but further filters the count to profiles that have a non-EMPTY measure value. The other directives respectively produce the sum, average, minimum or maximum of the measure value across the included profiles.

Creating an Advanced Report

This section will lead you through creating a very simple advanced report. In the following sections, more advanced techniques will be introduced.

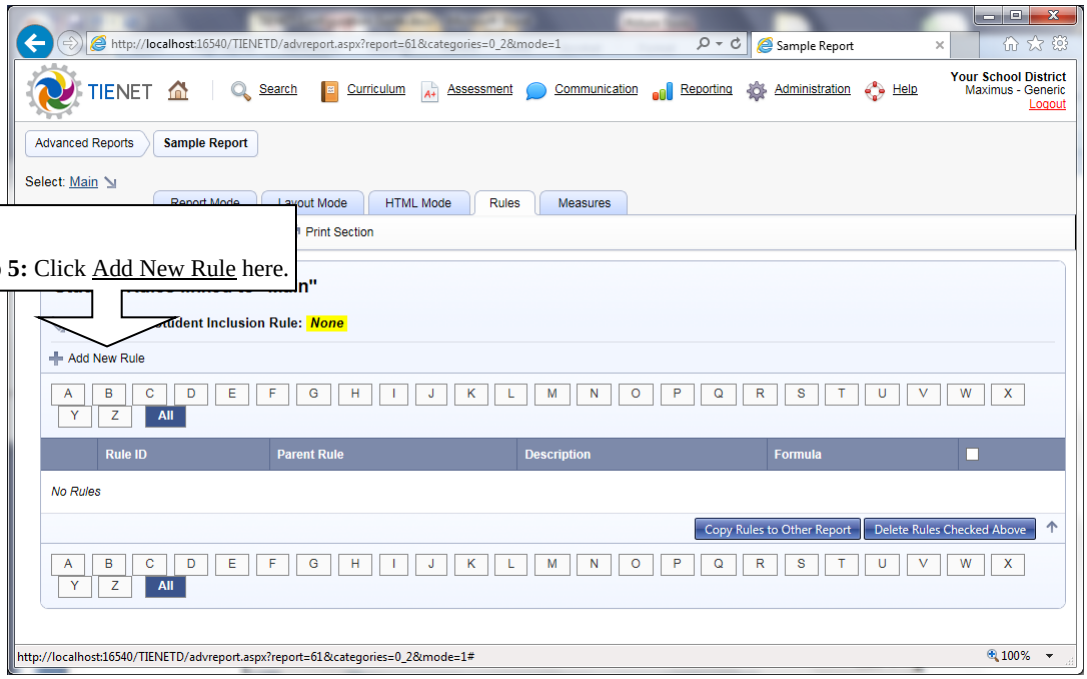


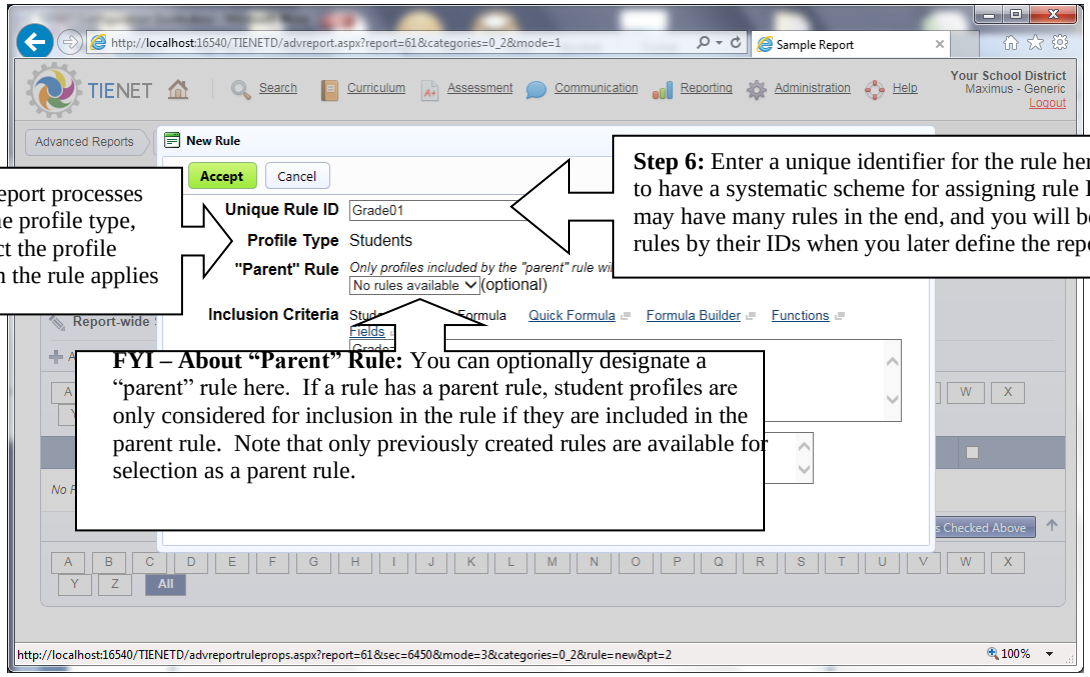


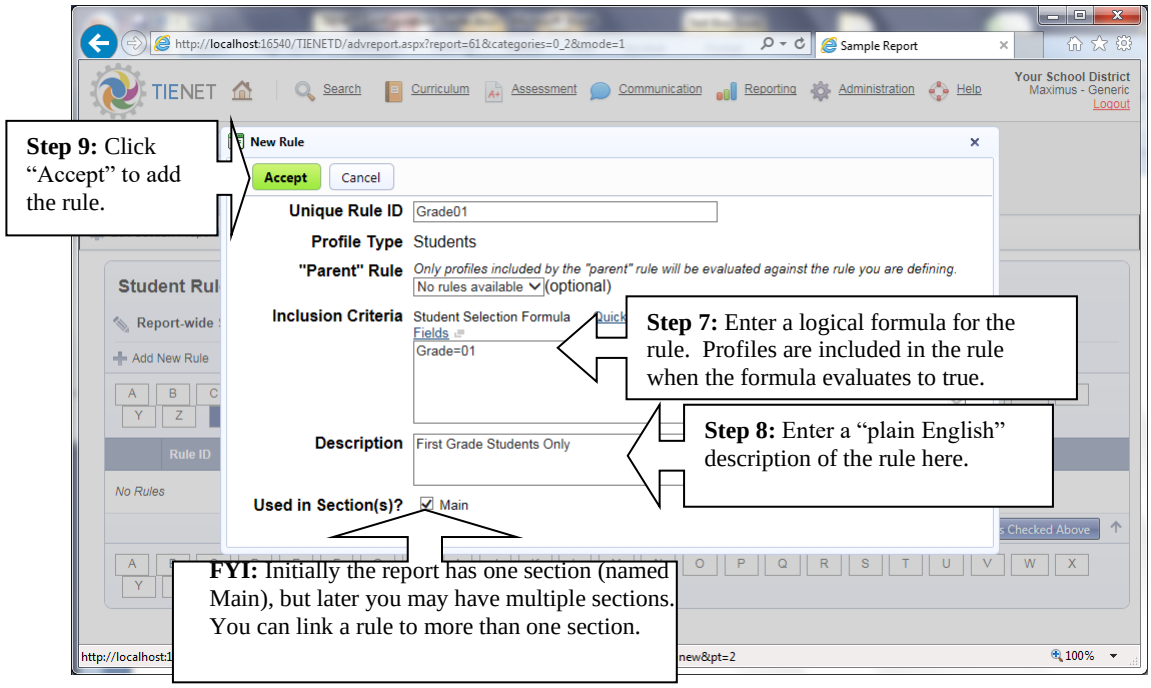


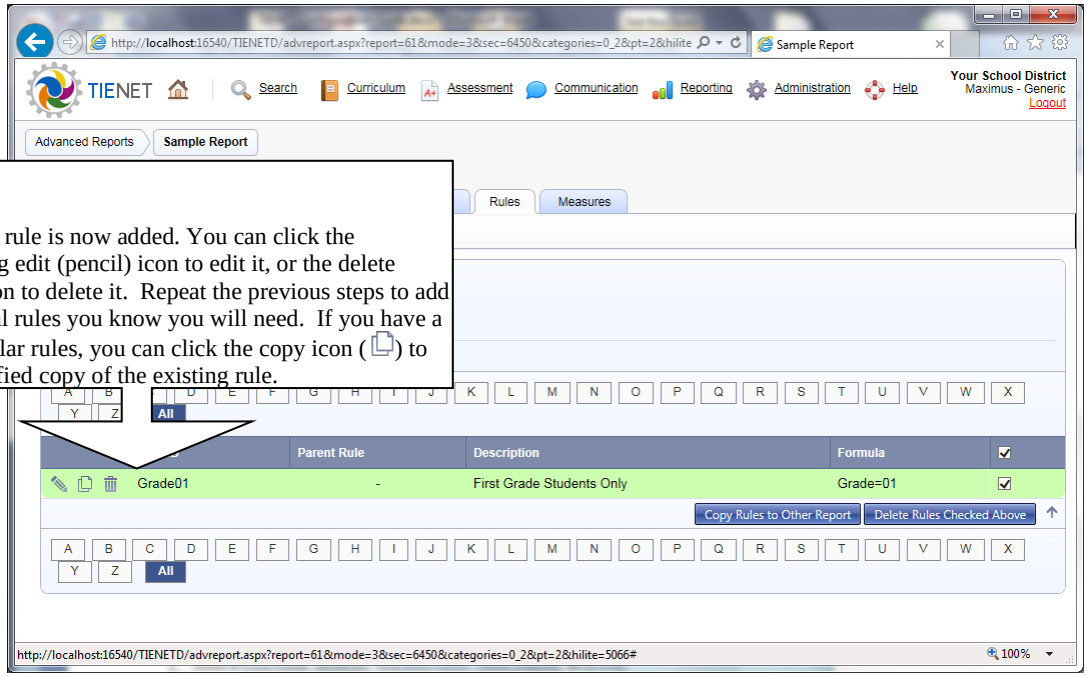
FYI: There are four modes you can work in. Click the corresponding tab to work in that mode.

- Work in **Report Mode** when you wish to actually run or produce the report.
- Work in **Layout Mode** or **HTML Mode** when you wish to design the layout or presentation of the report.
- Click **Rules** when you wish to view or edit the rules for the report.
- Click **Measures** when you wish to view or edit the measures for the report.

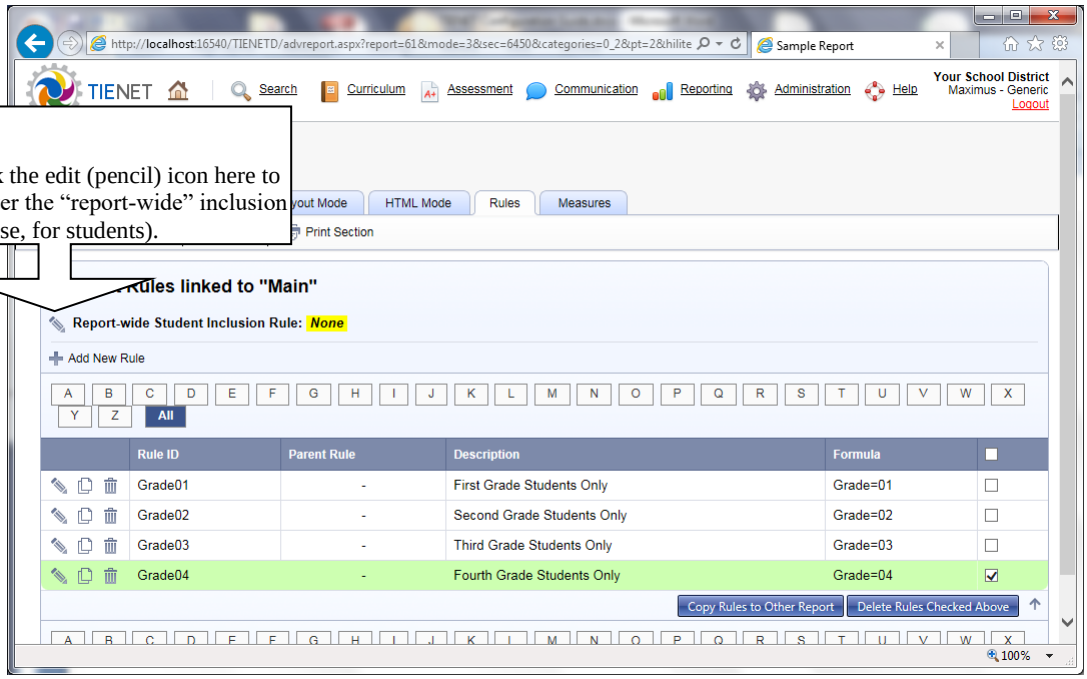




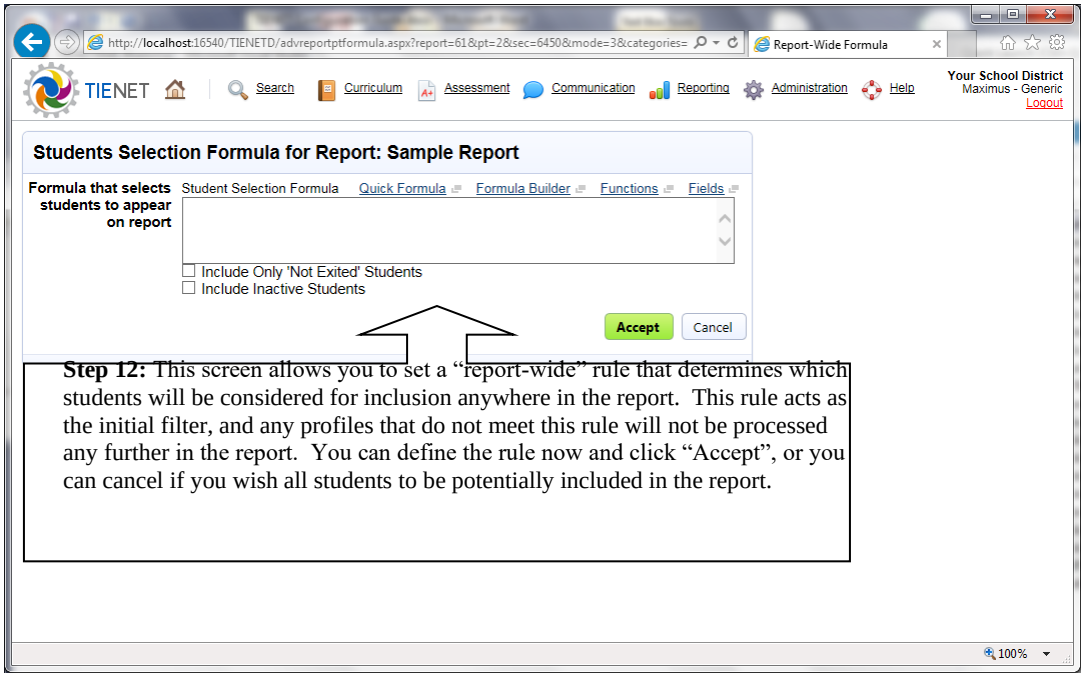




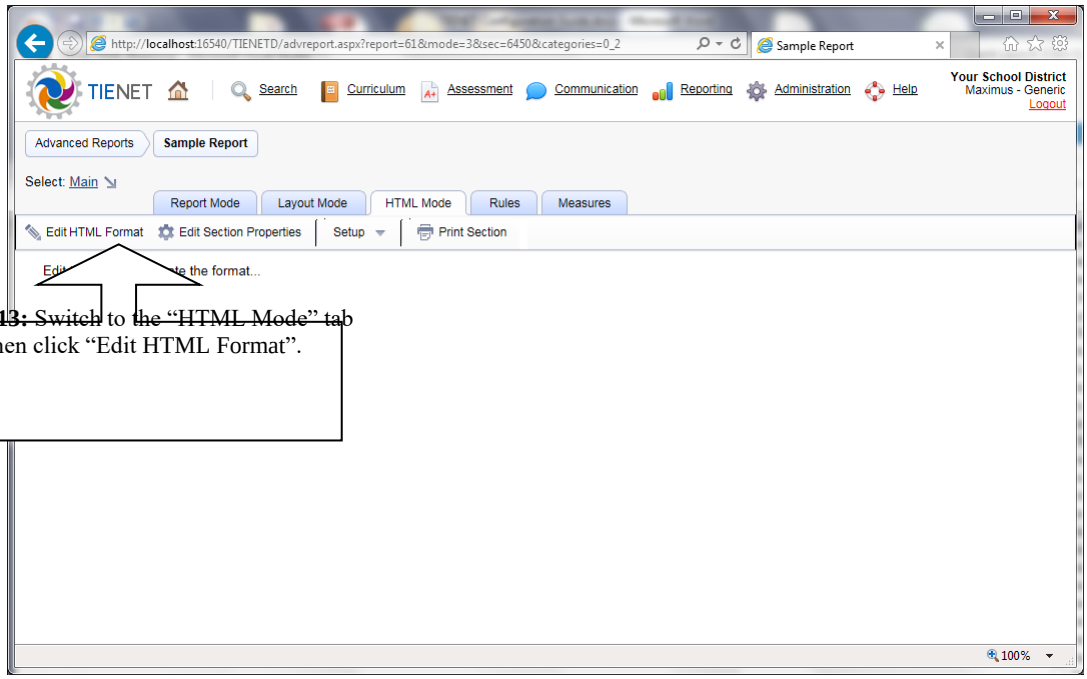
Step 10: The rule is now added. You can click the corresponding edit (pencil) icon to edit it, or the delete (trashcan) icon to delete it. Repeat the previous steps to add any additional rules you know you will need. If you have a series of similar rules, you can click the copy icon ( ) to make a modified copy of the existing rule.



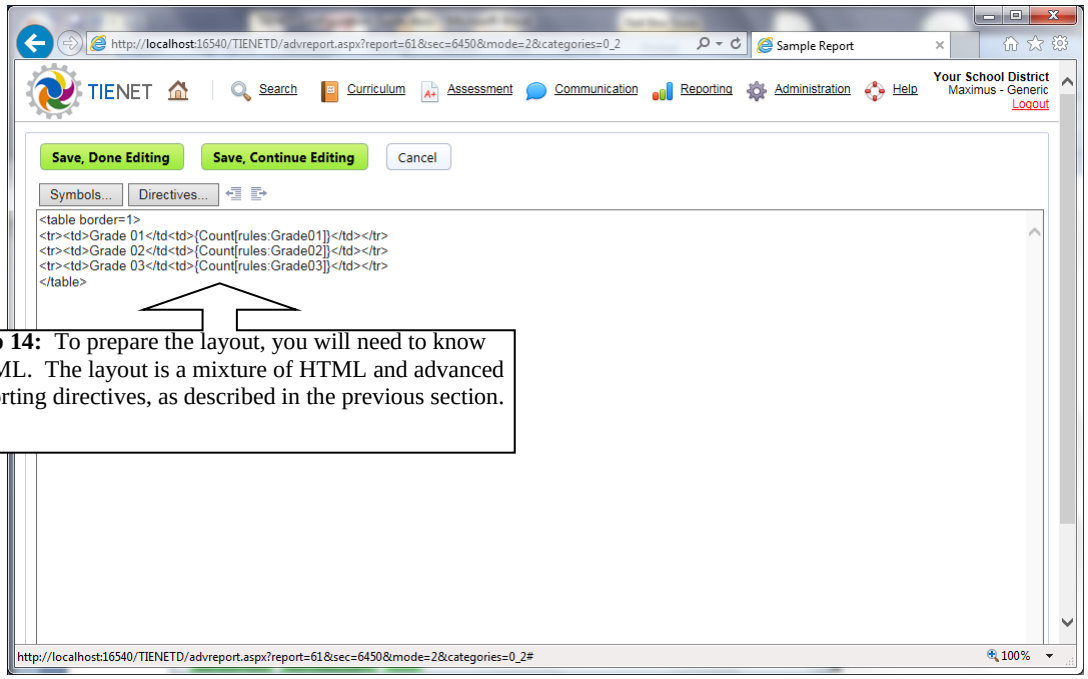
Step 11: Click the edit (pencil) icon here to see how to enter the "report-wide" inclusion rule (in this case, for students).



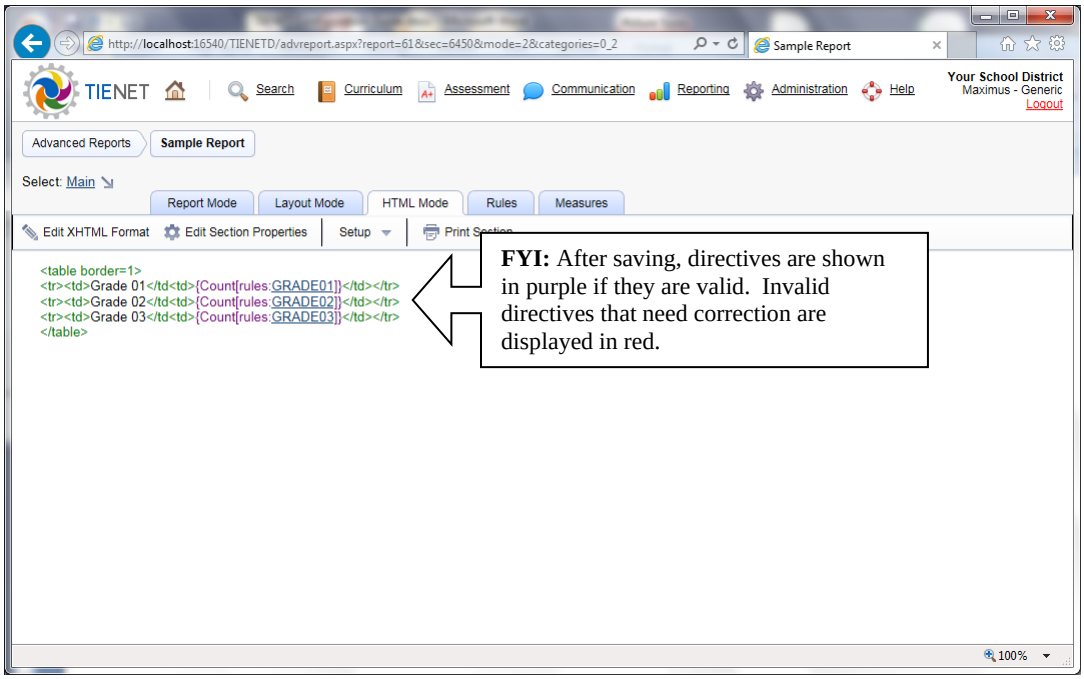
Step 12: This screen allows you to set a "report-wide" rule that determines which students will be considered for inclusion anywhere in the report. This rule acts as the initial filter, and any profiles that do not meet this rule will not be processed any further in the report. You can define the rule now and click "Accept", or you can cancel if you wish all students to be potentially included in the report.



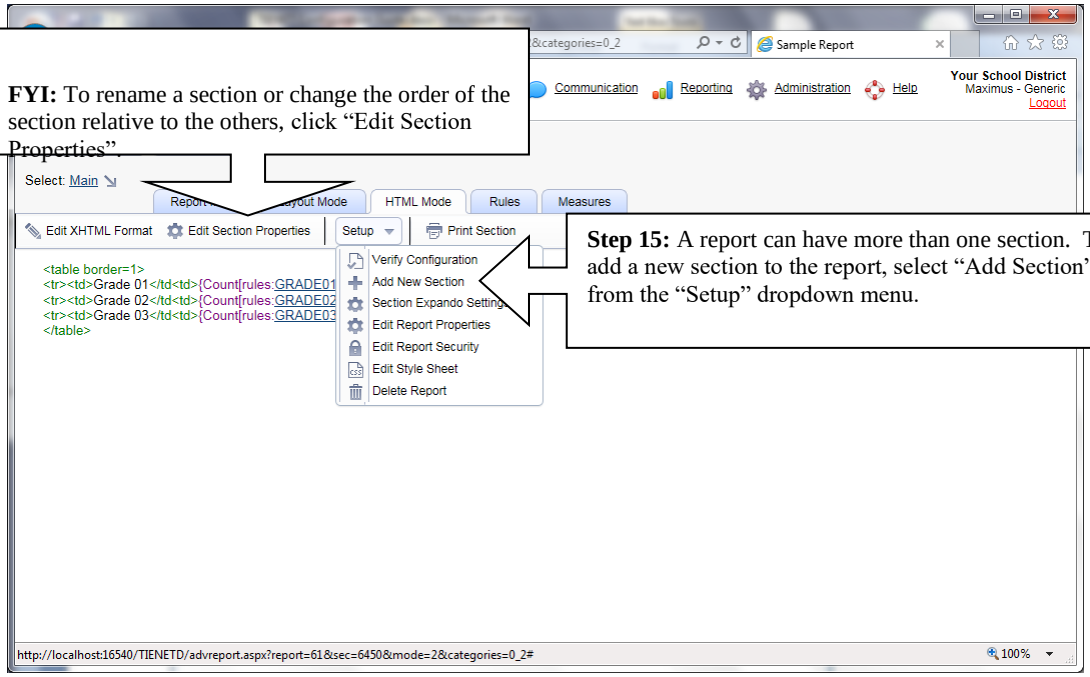
Step 13: Switch to the "HTML Mode" tab and then click "Edit HTML Format".

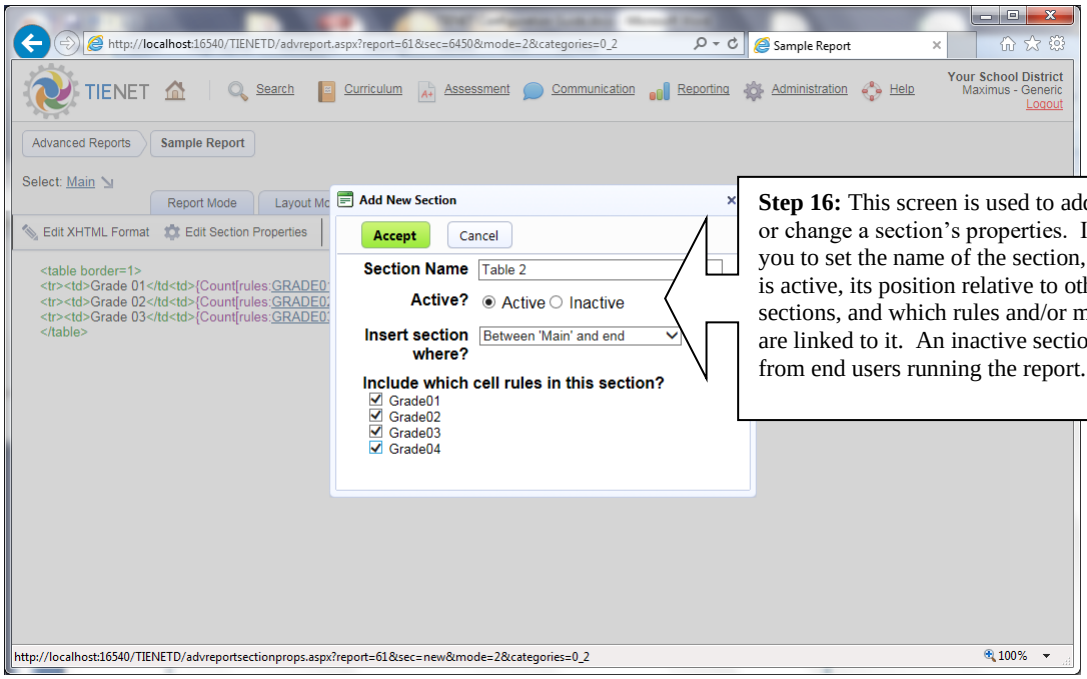


Step 14: To prepare the layout, you will need to know HTML. The layout is a mixture of HTML and advanced reporting directives, as described in the previous section.

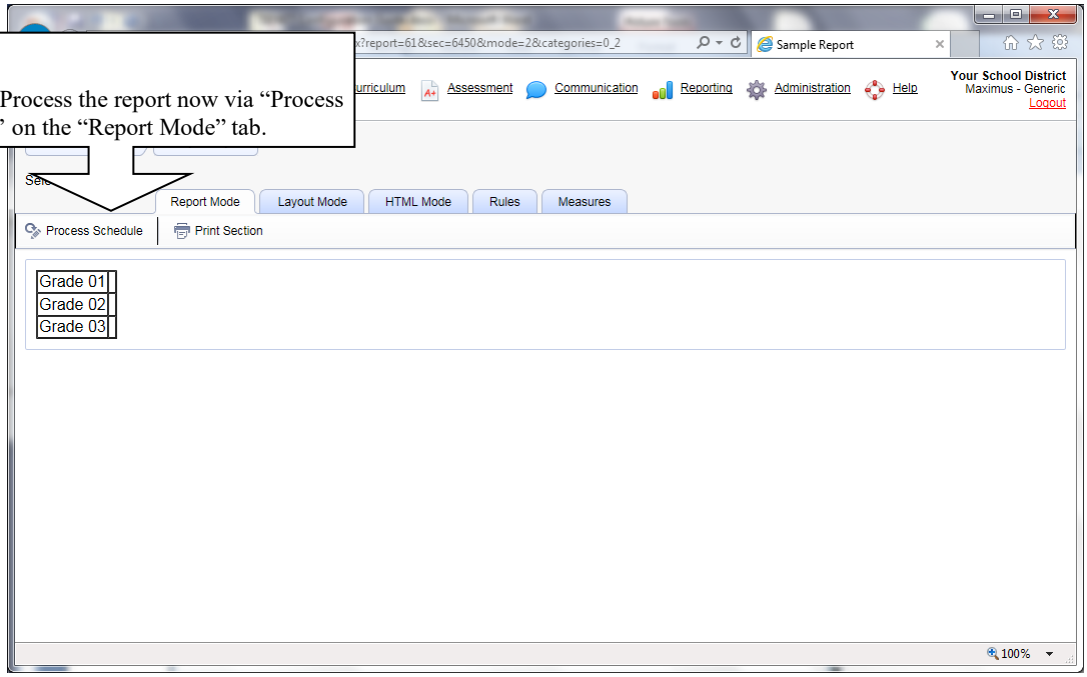


FYI: After saving, directives are shown in purple if they are valid. Invalid directives that need correction are displayed in red.

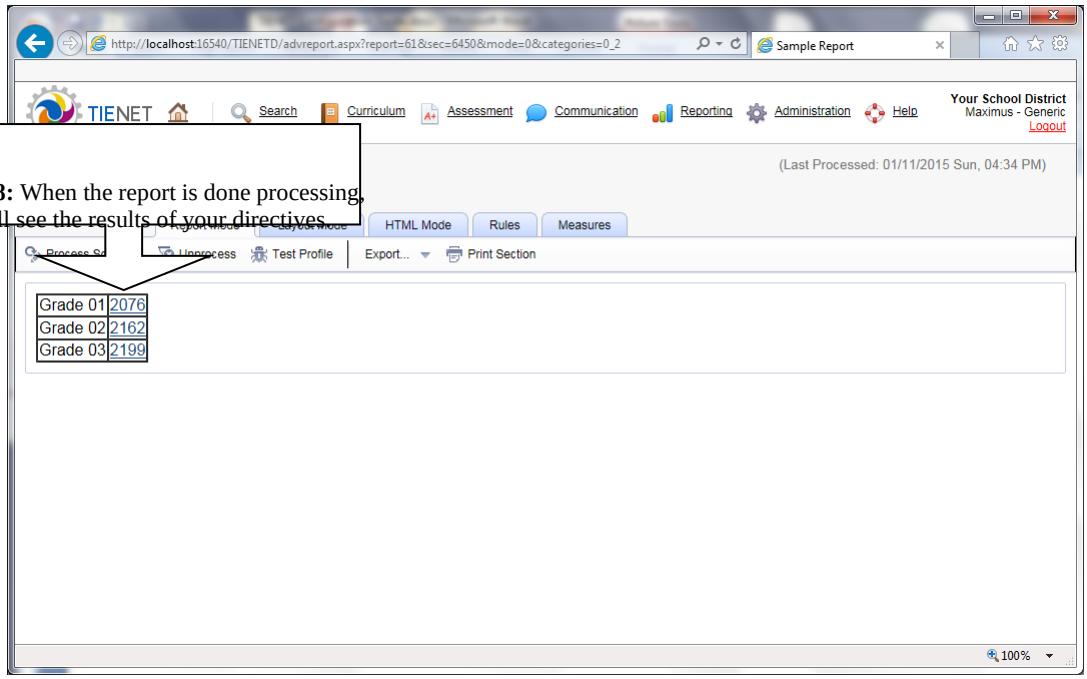




Step 17: Process the report now via "Process Schedule' on the "Report Mode" tab.



Step 18: When the report is done processing, you will see the results of your directives.



Command (#) Directives

There are some additional directives that may be useful:

{#IF *formula*}*content*{#ENDIF} If you wish to include content conditional upon the result of a logical formula, enclose the content between **{#IF *formula*}** and **{#ENDIF}**. Note that the formula may only reference global fields.

{#EVAL *expression*} or {#EVAL *expression*: *#.##*} This directive is useful for transforming the numeric output of named numeric cells. Simply write an algebraic expression referencing the named cells using operators +, -, *, /, % (modulus) with standard parenthesis as needed. When using division, caution should be used to avoid "divide by zero" errors. Additionally it should be understood that some aggregate values may result in NULL (i.e. SUM of a measure where no actual numeric values are summed). The functions listed below are supported and can assist with these cases. Note that the user will be unable to drill down into an #EVAL expression and so this should only be used when the result cannot be produced by the standard techniques described earlier. The **{#EVAL *expression*: *#.##*}** variation allows you to specify the number of decimal places, especially in situations where division is used. For example **{#EVAL *expression*: *#.###*}** specifies three decimal places for display purposes.

IIF(*logicalexpression, alternative1, alternative2*} returns alternative1 is the logicalexpression is true, otherwise it returns alternative2. This can be used to avoid "divide by zero" errors, e.g. IIF(divisor <> 0, numerator / divisor, null). It can also be used to transform numeric cells into text output, i.e. IIF(*cellid* > 1000, 'HIGH', 'LOW')

ISNULL(*expression, alternateifnull)* returns expression if it is not null, otherwise it returns alternateifnull.

Adding Charts to Advanced Reports

You can add colorful pie, bar and column charts to your advanced reports provided that the numbers that comprise the numeric series to be charted are already numeric cells on the same section of the advanced report. This is accomplished using chart directives as follow:

{$PIE(*cellid*:"*label*",...)[title="MyChart",TitleFontSize=14,Height=400,Width=400,DataFontSize=12,ThreeDee=true]}

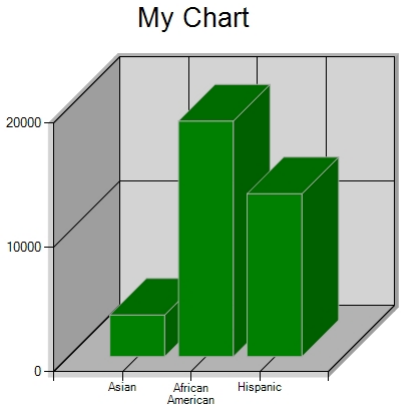
{$BAR(*cellid*:"*label*",...)[title="MyChart",TitleFontSize=14,Height=400,Width=400,MeasureName="Population",DataFontSize=12,ThreeDee=true,Cylinder=true]}

{$COLUMN(*cellid*:"*label*",...)[title="MyChart",TitleFontSize=14,Height=400,Width=400,MeasureName="Population",DataFontSize=12,ThreeDee=true,Cylinder=true]}

As an example, if you want to produce a pie chart using two aggregates in your advanced report, make sure that each aggregate cell to be charted has an explicit cell ID (described in previous sections). If those cell IDs were Resident and Received, your directive might look like:

{$PIE(Resident:"Resident Students", Received:"Students Received From Other Districts") [title="Resident Versus Received", TitleFontSize=14, Height=400, Width=400, DataFontSize=12, ThreeDee=true]}

The best way to determine how the additional named properties work is to experiment with then.



Sample Chart

As an example, if you want to produce a pie chart using two aggregates in your advanced report, make sure that each aggregate cell to be charted has an explicit cell ID (described in previous sections). If those cell IDs were Resident and Received, your directive might look like:

{$PIE(Resident:"Resident Students", Received:"Students Received From Other Districts") [title="Resident Versus Received", TitleFontSize=14, Height=400, Width=400, DataFontSize=12, ThreeDee=true]}

The best way to determine how the additional named properties work is to experiment with then.

The PIE directive also supports these additional attributes that can be used to improve the presentation of the resulting pie chart:

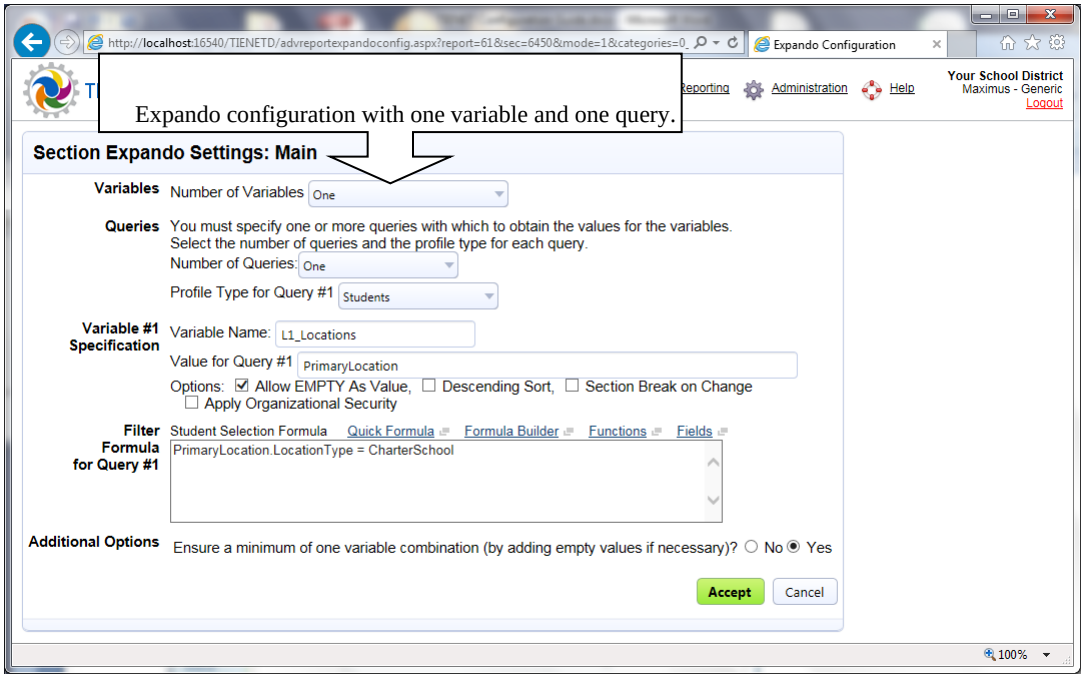
- LabelPieOutside =true (moves slice labeling outside of the pie, which may make the chart less busy)
- AppendValuesToLabels=true (results in the numeric values being appended to the pie-slice labels, since otherwise they do not appear)
- LabelPieWithValues=true (directly labels slices with the numeric values, but this should be used in conjunction with ShowLegend=true since otherwise the text labels will not be visible)

"Expando" Report Sections

If a report has rows of data and aggregates where each row corresponds to a school, the previous techniques only allow you to configure the report if you know at the time of configuring the report exactly what each school will be, and in this case, you would have to "hard code" the schools into the report requiring you to go back and change the configuration any time the schools on the report need to change. Many typical

reports have qualities that are similar to this. To implement reports like this without "hard coding", you configure an "expando" section.

To set up an "expando" section, create or go to a section that you would like to set up as "expando". Then select "Section Exando Settings" from the "More dropdown". The expando settings screen allows you to define one or more "expando variables" and how they get populated. Note that an "expando section" is simply a section that has one or more expando variables. When the advanced report is generated, PowerSchool Special Programs will take the expando section and repeat it for each unique combination of values that have been populated into the expando variables. As an example, if you would like to create an expando section that gets repeated for each charter school attended by at least one student in the database, then (as in the screen below), you could set up one expando variable named "L1_Location" that gets populated by querying the value of the PrimaryLocation field from the students profile type where the PrimaryLocation is a charter school (see below). You can name the expando variable whatever you like, but as a suggested convention, you can prefix it with L1_ and where there is more than one, L2_, L3_, etc. Prefixing expando variables this way makes them easy to identify in formulas later.



The above example with charter schools has only one expando variable. However, consider a report that will have a row of aggregate data for each unique combination of grade level, ethnic and gender found in the student population. This example would warrant three expando variables for grade level, ethnic and gender respectively.

In some cases, more than one query is needed to fully and accurately populate the expando variables. In the example below, two queries populate the L1_Location expando variable with all charter schools that are either the primary location or the secondary location. PowerSchool Special Programs will populate the expando variable with all the unique values found in both queries.

By default, PowerSchool Special Programs will repeat the entire section for each unique set of values populated into the expando variables. As an alternative, you can configure the section such that a portion of the section repeats, so that with more unique combinations, the one section expands instead of repeating in its entirety. To configure this, just enclose the repeating portion of the HTML format with {#EXPANDO} and {#ENDEXPANDO}.

You can use rules and measures anywhere within an expando section. However, to be useful, the expando section will need to have rules and/or measures that reference the populated values of the expando variables. Standard rules and measures cannot reference expando variables. However, once you have an expando section, you will see links that allow you to create "expando rules" and "expando measures". T he process is similar, except that only "expando rules/measures" can reference expando variables and "expando rules/measures" can only be used in an expando section.

When setting up expando variables in the expando configuration screen, there are some additional options that require explanation:

- Allow EMPTY as Value: If this option is set for an expando variable, then PowerSchool Special Programs allows the expando variable to be EMPTY. In this case, the values with which the expando variable is populated could include the EMPTY value if EMPTY results from the query. If this option is turned off, the EMPTY value is disallowed and filtered out.

- Descending Sort: When the expando section is repeated for each unique value(s) in the expando variable(s), the sections are repeated according to the natural ordering for that variable (i.e. alphabetical order, keyword order, numerical order, etc.) Enabling this option reverses the natural order.

- Section Break on Change: This option is only relevant if the {#EXPANDO}{#ENDEXPANDO} directives are used to repeat only a portion of a section. If enabled, PowerSchool Special Programs will generate an entirely new section wherever there is a change in any of the expando variable values.

- Apply Organizational Security: The option can only be enabled if the corresponding expando variable is a profile selection for an organizational location, and is actually only relevant if the repeating section approach is used. If enabled, PowerSchool Special Programs checks the user's security and only allows the user to see instances of the section corresponding to organizational locations that the user has profile view access to. To determine profile view access, PowerSchool Special Programs checks profile view privileges for the first profile type the report is aggregating data for (typically Students, or if not students, then typically Staff).

- Ensure a minimum of one variable combination (by adding empty values if necessary)? Yes/No: There is the possibility that zero values will be queried for the expando variables, and that would result in zero repetitions of the expando section causing it not to appear at all in the report output. If this option is turned on, PowerSchool Special Programs will check if zero values have been queried, and if so, will populate the expando variables with an empty value ensuring that at least one repetition is made.

Setting Drilldown Columns

By default, the system infers default columns when drilling down into each report cell based on cell rules and other heuristics, and an end user can customize the inferred default columns. However, one has the option of configuring a specific set of columns in a particular order, which then replaces the inferred columns as the default columns. To do this, select a configuration task, switch to the Layout tab, and select Setup > Set Drilldown Columns. The end user can still customize the default columns whether they are inferred or preset as part of the report definition. Since the preset columns are part of the report definition, they are synced by the CMT.

Optimizing Advanced Report Performance

The performance (and time to process) of an advanced report can often be improved by the following practices.

1. Be sure to utilize the report-wide filter rules and make them as restrictive as possible. This often does more than anything else to reduce the amount of processing.

2. If two or more rules are always applied to the same report cells or areas, you can often reduce processing by merging those rules into a single rule by combing the formulas with 'OR' logic.

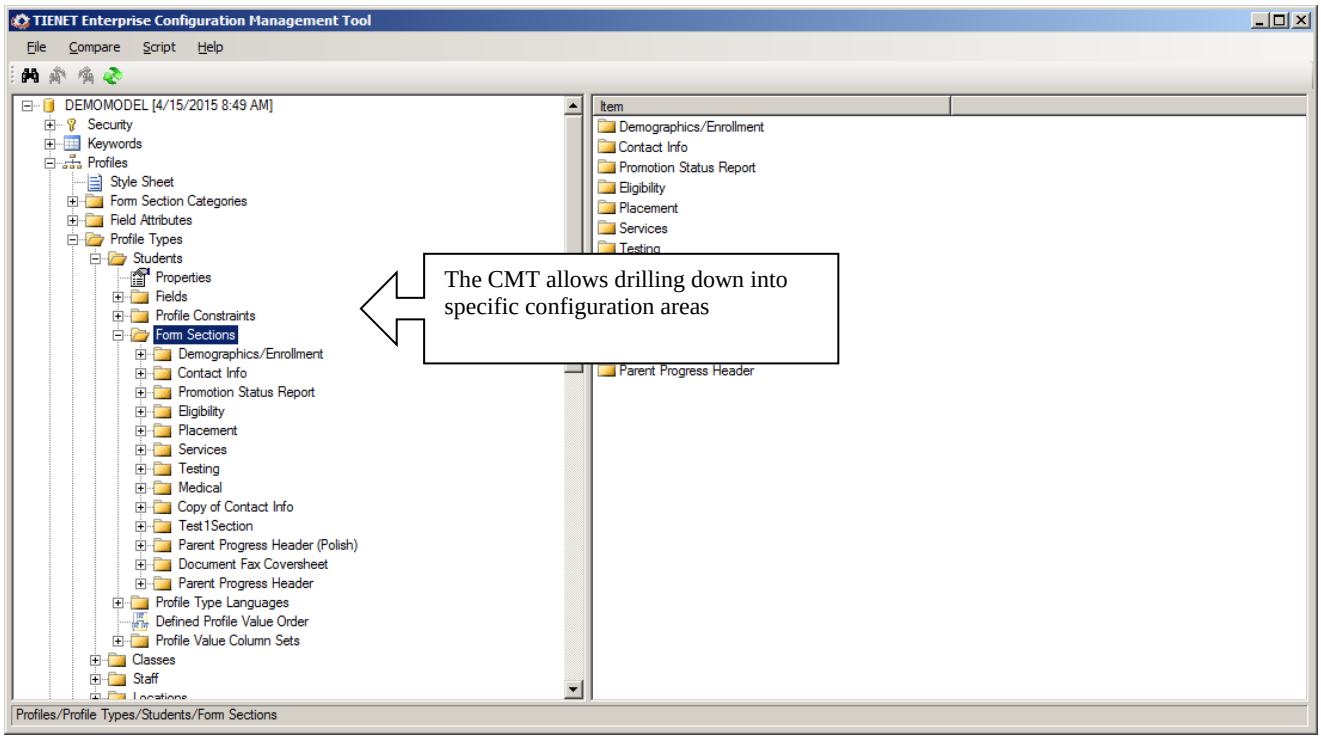
Configuration Management Tool

Chapter 11

The Configuration Management Tool (CMT) is a server application used to extract, store, compare, and synchronize configurations of PowerSchool Special Programs databases.

CMT Overview

When you open the configuration for a database in the CMT, the information is separated into two panes. The left pane allows drilling down into specific configuration areas while the right pane displays contained configuration elements.



The loaded configuration can be saved to a file on disk for later reference. These files can be used to track configuration changes to a database over time.

The configuration is organized into various repositories:

- **Security**: includes security settings (Admin, Consultant, Staff, Parents & Students, Audit Log) as well as Staff security groups with their granted and denied privileges
- **Keywords**: includes all keyword table configuration and data

- **Profiles**: includes overall profile configuration (global stylesheets, form section categories, field attributes), profile types configuration (fields, form sections, constraints, security, translation), update scripts, placement definitions, and custom UDF configuration

- **Documents**: includes overall document configuration (global stylesheets, document template categories), document template configuration (fields, sections, child templates, actions, security, translation), and custom event fields.

- **Reports**: includes all advanced reports, any standard (ad-hoc) reports that have been placed under configuratuion management, and security associated with the reports

- **Integration**: includes profile and document import layouts, and direct integrations (such as Active Directory)

- **Messaging**: includes messaging settings for the database

- **Help**: includes help settings and external help resources

One of the main uses of of the CMT is to create and/or change configuration in a base database (such as a development or state model database), and when complete use the CMT to propagate those changes to one or more databases.

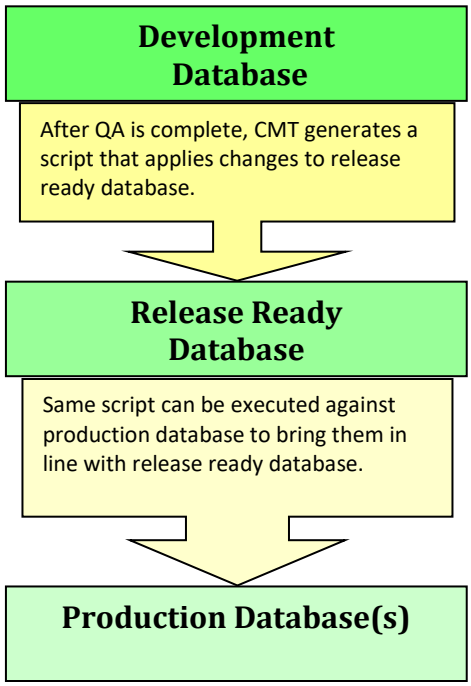


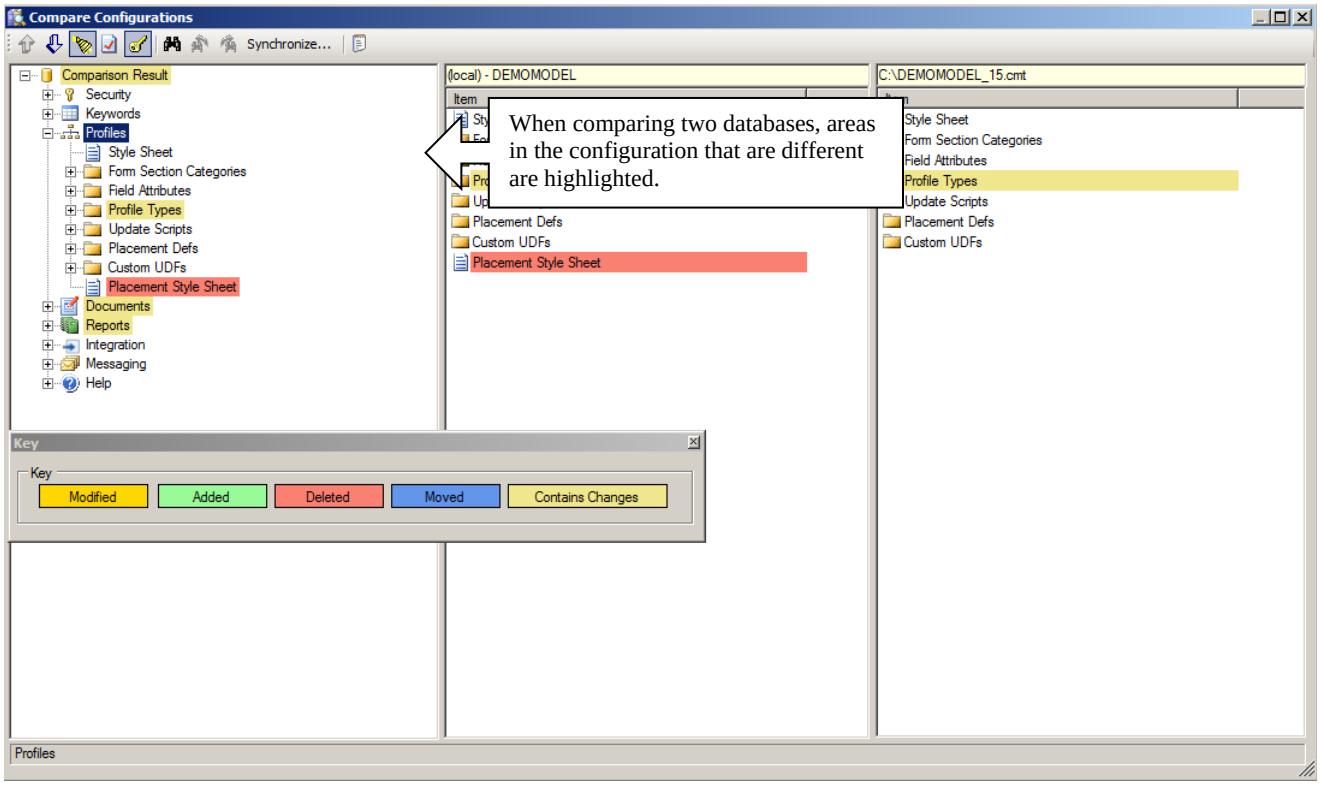
Figure 10.1 – Developing and Propagating Configurations

Model Versions

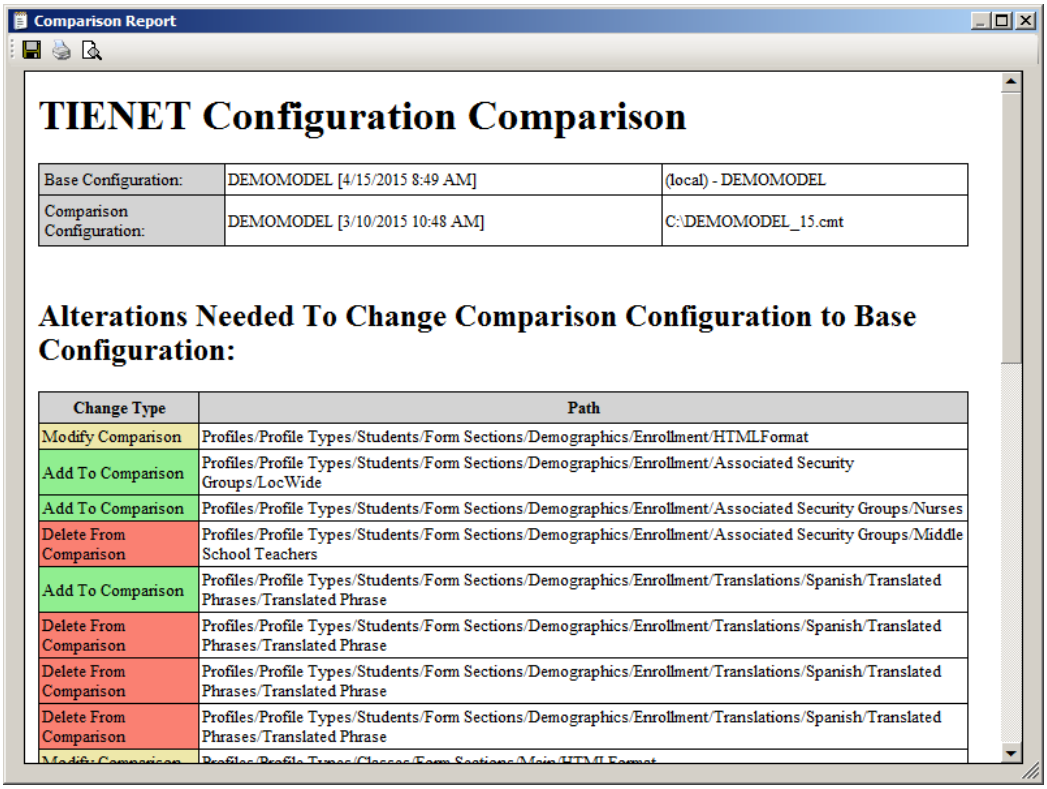
As of Version 17.1, model versions are explicitly supported in the configuration. To establish the model version, go to Configuration > Profile Types > More > Set Model Version (after selecting a configuration task). Model versions have the format <ALPHA>.<year:YYYY>.<month:1-12>.<iteration>, for example, NY.2017.11.0. The CMT recognizes the model version, and when generating a script that represents a model version upgrade (e.g. NY.2017.11.0 to NY.2017.11.1), the generated script will verify that the target database is on the old model version and then set the new model version on the target database after making all changes.

Comparing Databases

The first step to copying configuration changes from one database to another is comparing their configuration. This is accomplished by loading the "base" database into the CMT and then comparing it to a "comparison" database. .

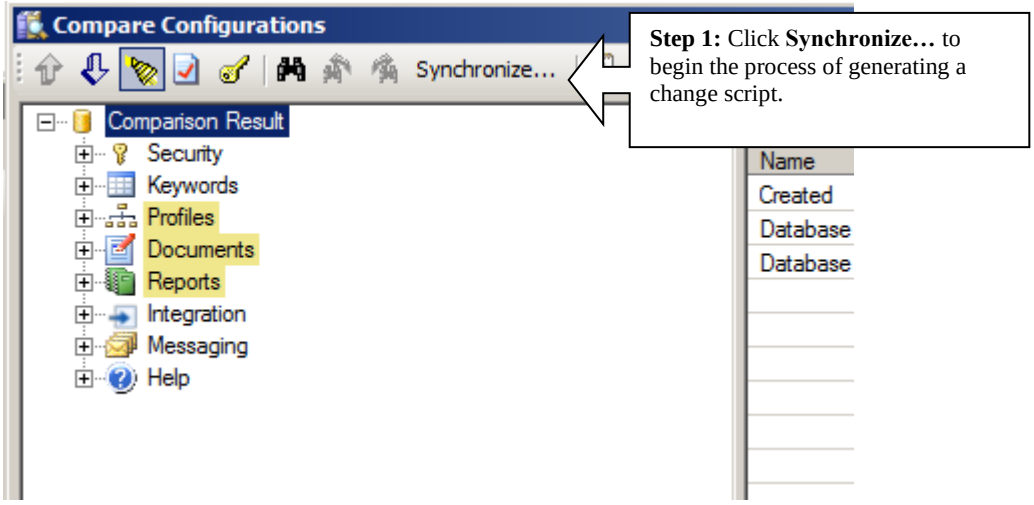


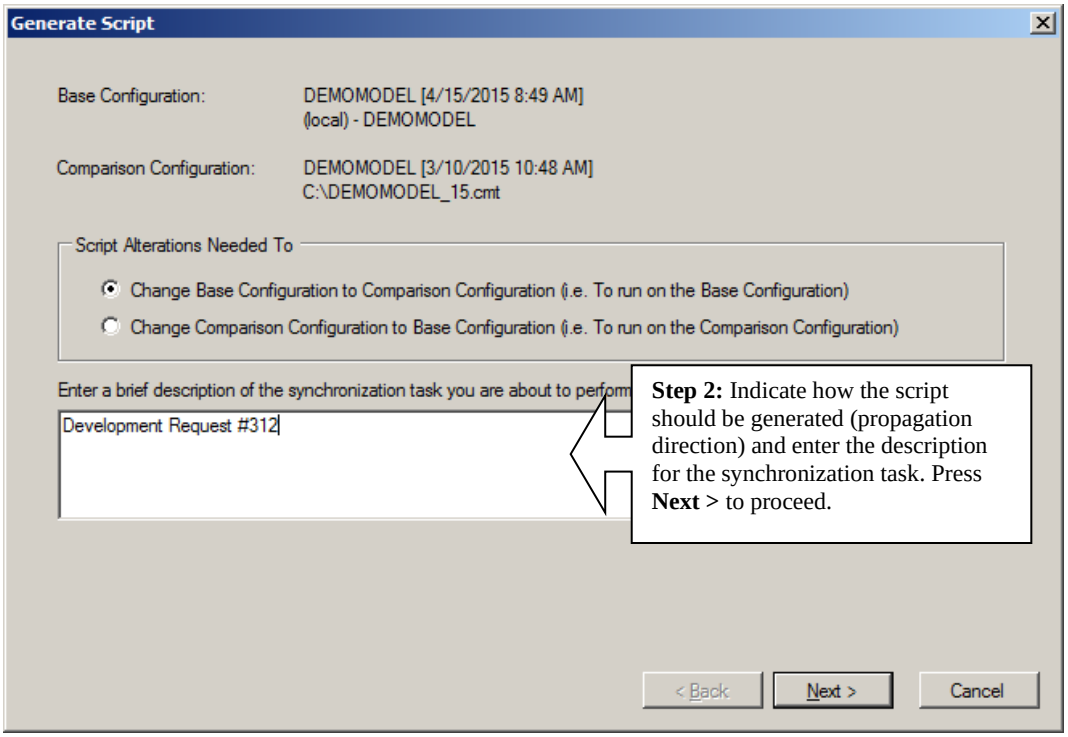
A basic report of changes items can be viewed by clicking on the **View Report…** button on the toolbar.

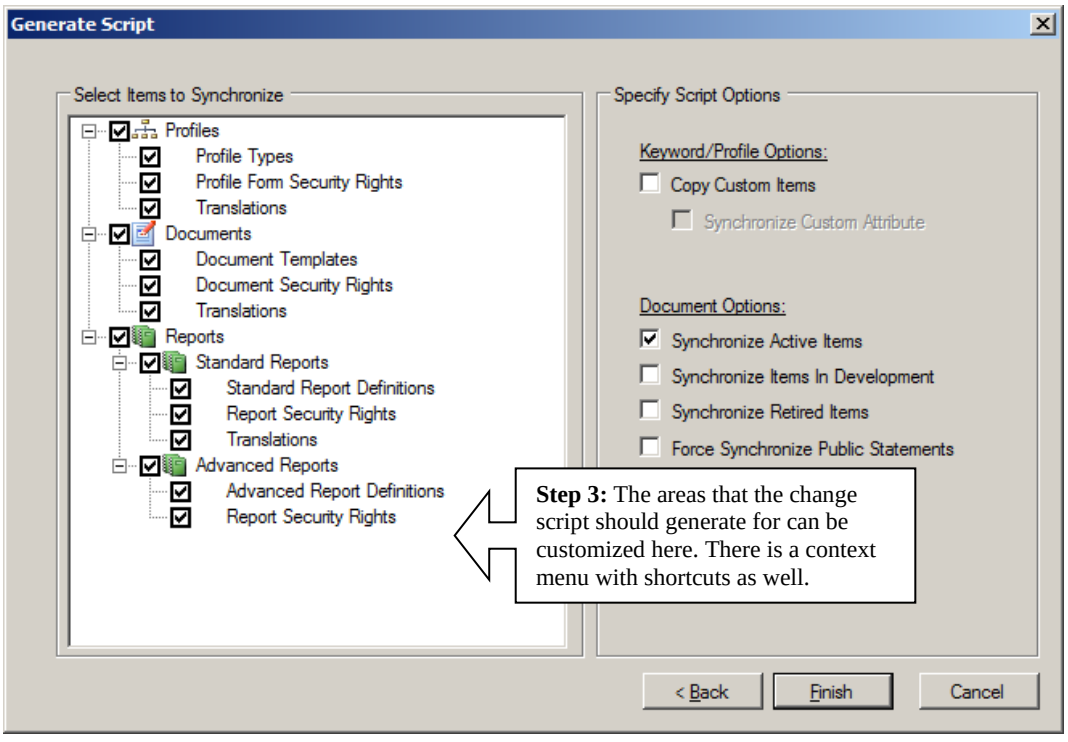


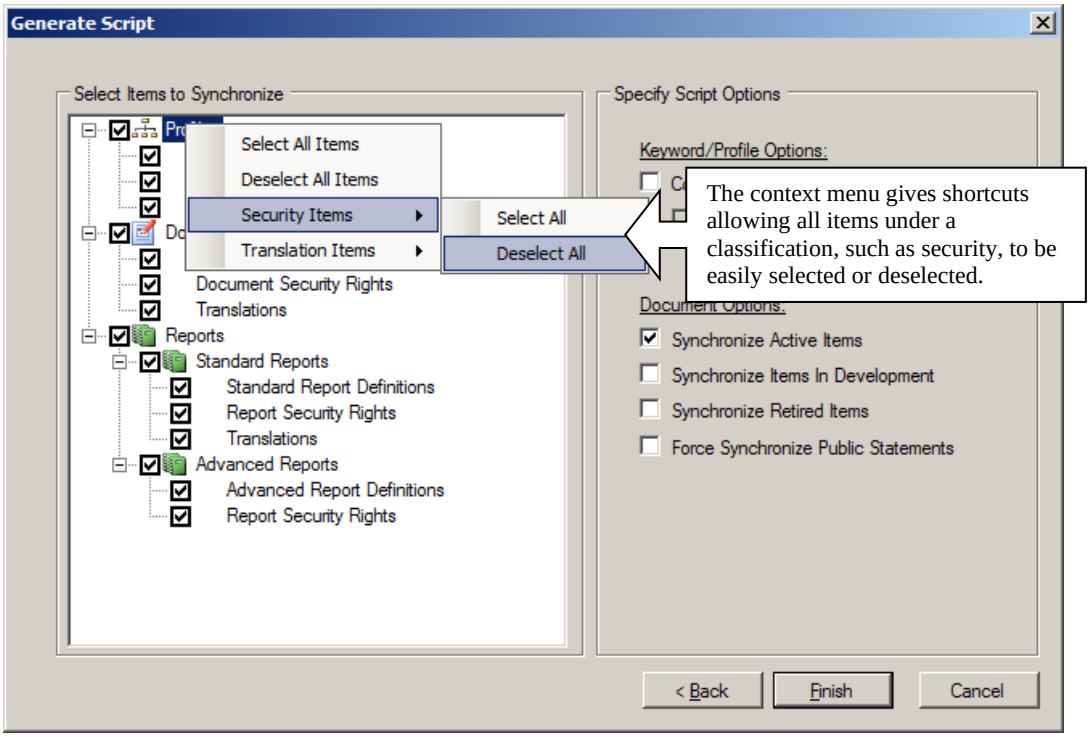
Synchronizing Configuration

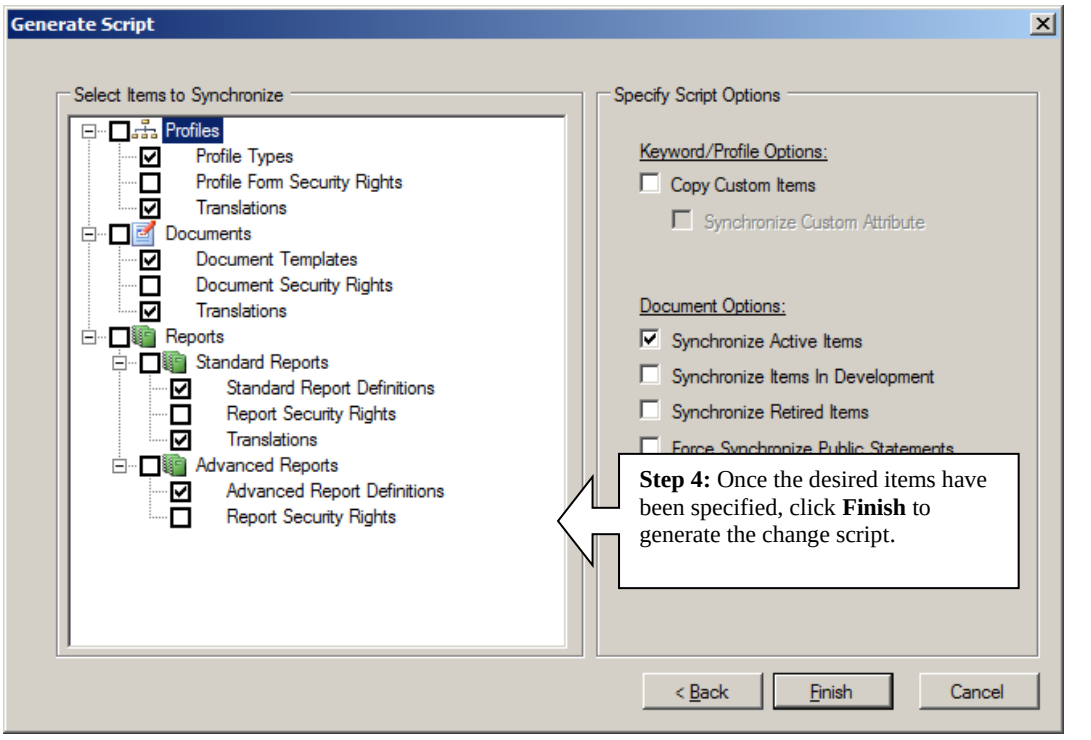
The configuration changes can be copied to a database by using the **Synchronize** functionality in the CMT. This will generate a script with commands that edit the target configuration to match the source database. It is possible to synchronize all changes or only subsets of changes.

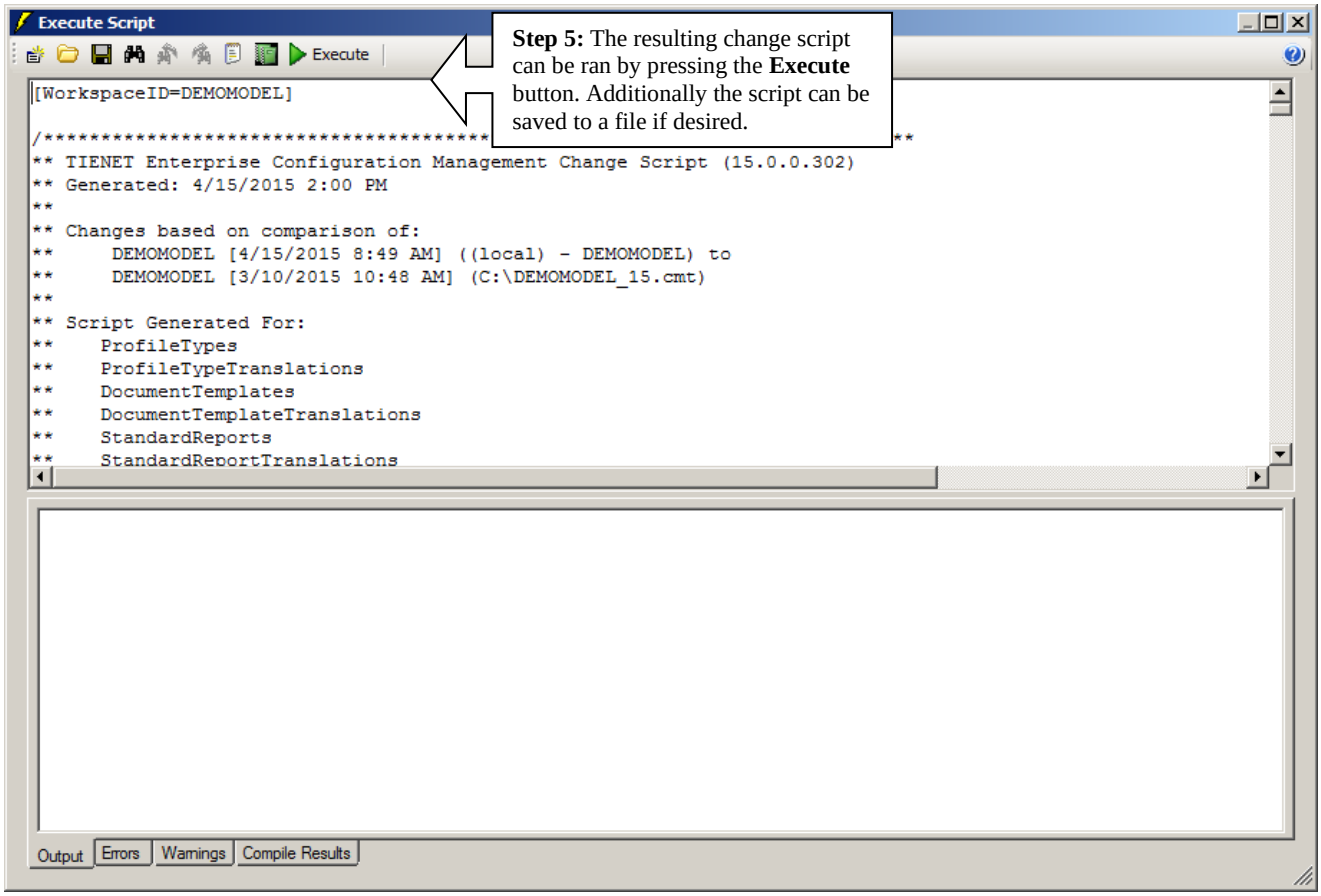




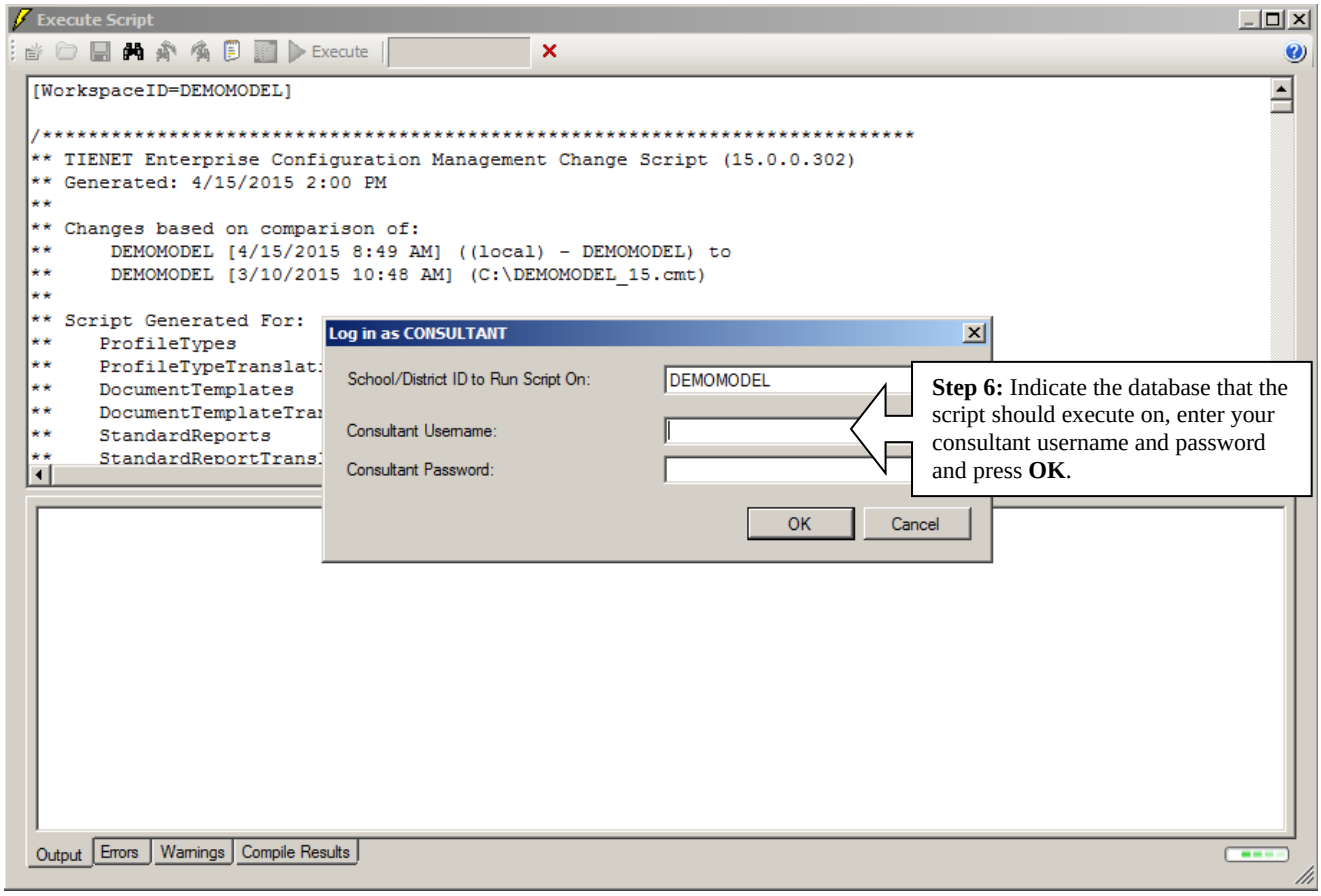


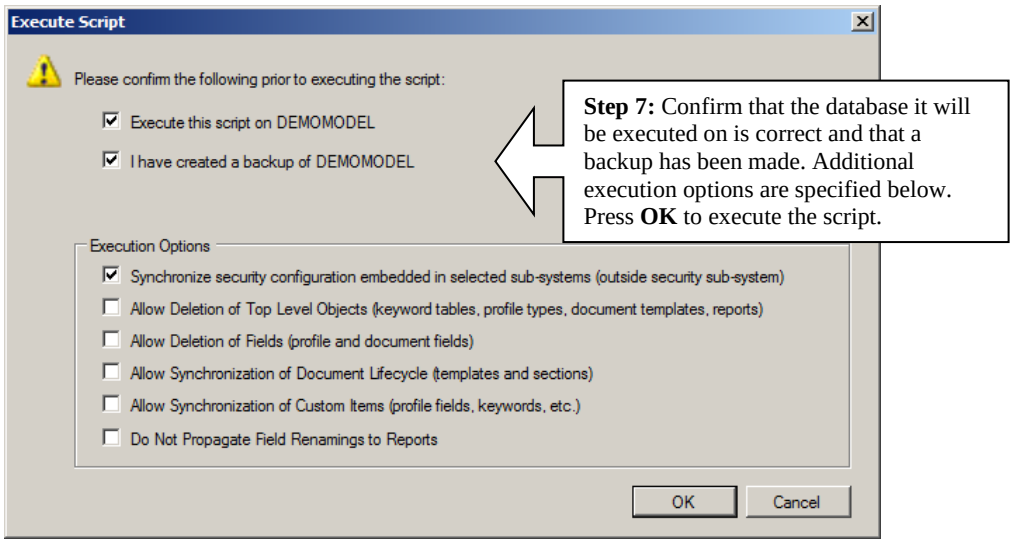


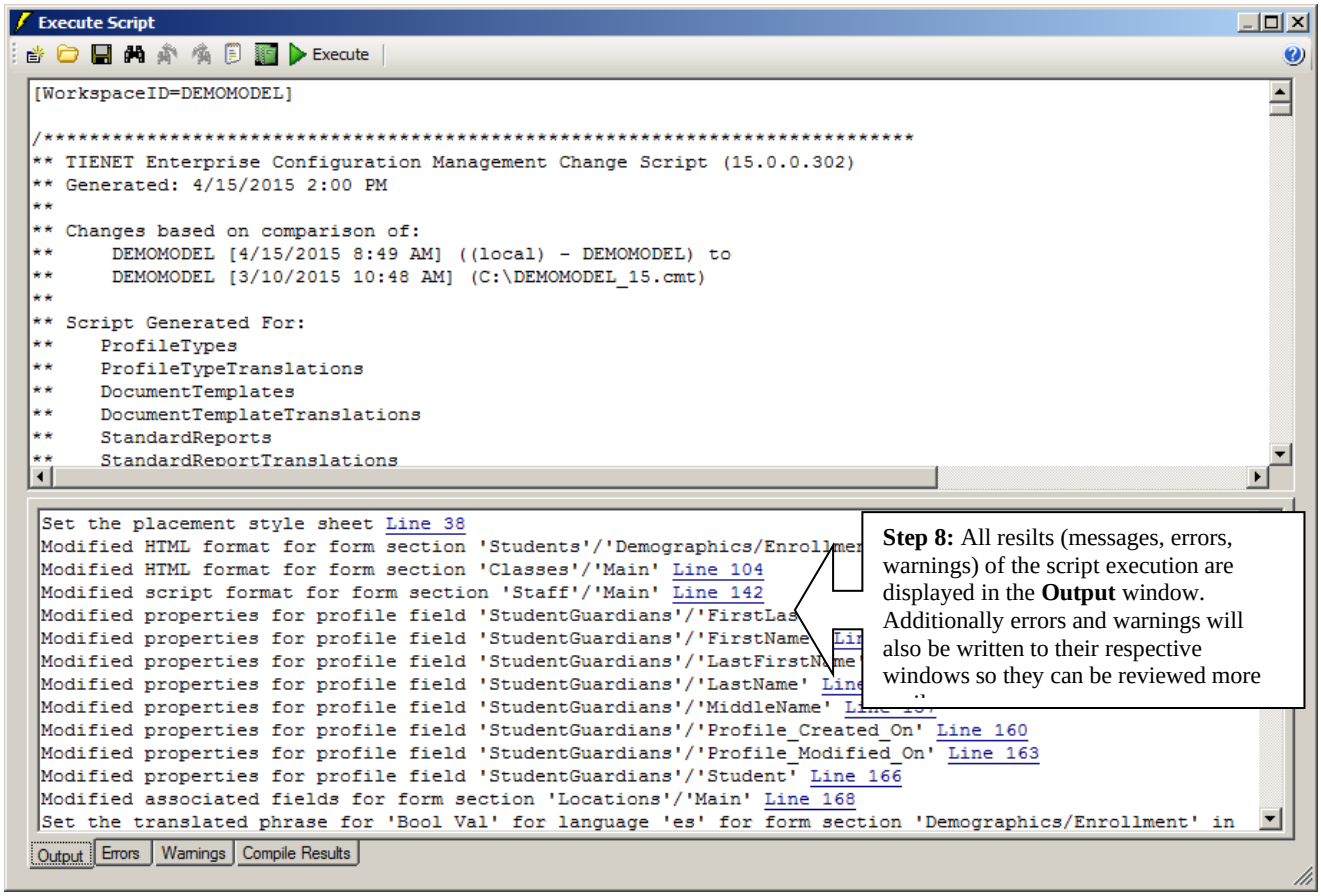




Step 5: The resulting change script can be ran by pressing the **Execute** button. Additionally the script can be saved to a file if desired.







CMT Renaming Fields and Other Objects

Whenever renaming a configuration object, such as a field, there is a danger that when the change is synchronized to other databases with the CMT, that the CMT will actually delete the object under the old name and create it under the new name rather than renaming the object. This can and will result in data loss. For this reason, the properties screen for the following object types has a prior names field that allow you to identify one or more prior names:

- Profile Type (set when changing plural name)
- Profile Field (set when changing field name)
- Document Template or Child Template (set when changing Template ID)
- Document Template Sections (set when changing section name)
- Document fields (set when changing field name)
- Keyword Tables (set when changing keyword table name)

- Keyword Table Fields (set when renaming keyword table fields)

The CMT will use the old names to look for objects under the old name and rename them to the new name instead of deleting and adding.

CMT Data Transformations

Data transformation and migration capabilities are sometimes necessary when introducing new field structures tha replace existing field structures. Such transformations can be inserted into a generated CMT script by manually inserting a line at the appropriate point in the script. The inserted line should have the format given below where <inlinescript> has exactly the same syntax as a profile update script.

```
+Profiles.ExecuteInlineUpdateScript( "<inlinescript>");
```

Note that inserting a new line into a script adds an additional step, and to account for that, the "steps" number in the following line near the beginning of the script should be manually incremented for each step manually inserted.

```
Initialize(steps: 36);
```

Data transformations may also be required for document data. Except where explicitly specified with a where clause, such transformations are applied to all documents including final documents (which may be unfinalized) and soft-deleted documents (which may be undeleted). However, operations that affect the section structure of existing documents are prohibited as explained in the table below.

The following operations on document data are supported:

Transformation	Explanation and example
Update top level document data	Top level documents cannot be created or deleted in a script but top level field values can be updated using the syntax below: UPDATE #<*DocTemplateID*> SET TargetField1=SrcExpression1, TargetField2=SrcExpression2… WHERE *FilterCriteria*
Insert (grand) child documents	Child documents associated with repeating sections are prohibited from being inserted since that affects the section structure of existing documents, therefore this command is restricted to repeating row data. Source data expressions for inserting child document data will be in the context of the top level document. Source data expressions for inserting grand child document data will be in the context of parent child documents but may also reference top level field values. Child documents and grand child documents can be inserted using the syntax below.

	INSERT #<*DocTemplateID*>#<*ChildTemplateID*> (*TargetField1,TargetField2…*) (*SourceExpression1,SourceExpression2…*) WHERE *FilterCriteria* INSERT #<*DocTemplateID*>#<*ChildTemplateID*>#<*GrandTemplateID*> (*TargetField1,TargetField2…*) (*SourceExpression1,SourceExpression2…*) WHERE *FilterCriteria* Note that each insert statement variation above will insert at most one new child document row per document, or at most one new grand child document per parent child document. The DISTINCT keyword, which is supported for similar syntax used for profiles, is not supported for document data. **New In 20.6.4.0:** The following advanced insert syntax allows you to specify a specific source template to insert rows from (potentially multiple rows). To insert child documents from a specified child template source, use the following syntax: INSERT #<*DocTemplateID*>#<*ChildTemplateIDA*> (*TargetField1,TargetField2…*) FROM #<*DocTemplateID*>#<*ChildTemplateIDB*> (*SourceExpression1,SourceExpression2…*) WHERE *FilterCriteria* To insert grand child documents from a specified top-level child template that is not the parent of the grand child template, use the following syntax: INSERT #<*DocTemplateID*>#<*ChildTemplateIDA*>#<*GrandTemplateID*> (*TargetField1,TargetField2…*) FROM #<*DocTemplateID*>#<*ChildTemplateIDB*> (*SourceExpression1,SourceExpression2…*) WHERE *FilterCriteria*
Update (grand) child documents	The following syntax updates (grand)child document fields to expressions that can reference parent fields. The syntax is given below: UPDATE #<*DocTemplateID*>#<*ChildTemplateID*> SET *TargetField1=SourceExpression1, TargetField2=SourceExpression2…* WHERE *FilterCriteria* UPDATE #<*DocTemplateIDT*>#<*ChildTemplateID*>#<*GrandTemplateID*> SET *TargetField1=SourceExpression1, TargetField2=SourceExpression2…* WHERE *FilterCriteria* **New In 20.6.4.0:** More advanced update operations are also possible where a (grand)child template is updated from a specific source using a row-to-row join expression:

	To update child documents from a specified child template source for which row matches can be defined, use the following syntax: UPDATE #<*DocTemplateID*>#<*ChildTemplateIDA*> FROM #<*DocTemplateID*>#*ChildTemplateIDB* SET *TargetField1=SourceExpression1, TargetField2=SourceExpression2…* WHERE Target*Expression1=SourceExpression1 and TargetExpression2=SourceExpression2….* To update grand child documents from a specified top-level child template that is not the parent of the grand child template, and for which row matches can be defined, use the following syntax: UPDATE #<*DocTemplateID*>#<*ChildTemplateIDA*>#<*GrandTemplateID*> FROM #<*DocTemplateID*>#*ChildTemplateIDB* SET *TargetField1=SourceExpression1, TargetField2=SourceExpression2…* WHERE Target*Expression1=SourceExpression1 and TargetExpression2=SourceExpression2….*
Delete (grand) child documents	Child documents associated with repeating sections are prohibited from being deleted since that affects the section structure of existing documents. But child documents associated with repeating rows can be deleted using the syntax below (the filter criteria may reference both child document field values as well as top level field values): DELETEFROM #<*DocTemplateID*>#<*ChildTemplateID*> WHERE *FilterCriteria* DELETEFROM #<*DocTemplateIDT*>#<*ChildTemplateID*>#<*GrandTemplateID*> WHERE *FilterCriteria*

CMT Conditional Logic and Other Special Cases

A CMT release script can now be manually modified to execute certain lines if a condition based on the globals profile is true. Just enclose the lines dependent on the global profile as follows:

```
if (Versioning.IfGlobalsFormulaIsTrue("globals_conditional_formula")) {
   conditional changes go here
}
```

To preset the ADMIN override setting for a document behavior field to either true or false, insert a statement like this into the release script:

Documents.SetDocFieldBehaviorOverride("*document_template_id*", "*behavior_field_name*", *true/false*);

PowerSchool SIS Integration

Chapter 12

PowerSchool Special Programs has a rich integration with PowerSchool SIS. Some of the features are enabled through configuration within PSSP.

Overview

After you load the PSSP plugin into PS SIS, several integration features become available within the SIS:

1. Single Sign-On (SSO) into PSSP
2. Embedded content in PS SIS
3. Special Program alerts displayed in PS SIS

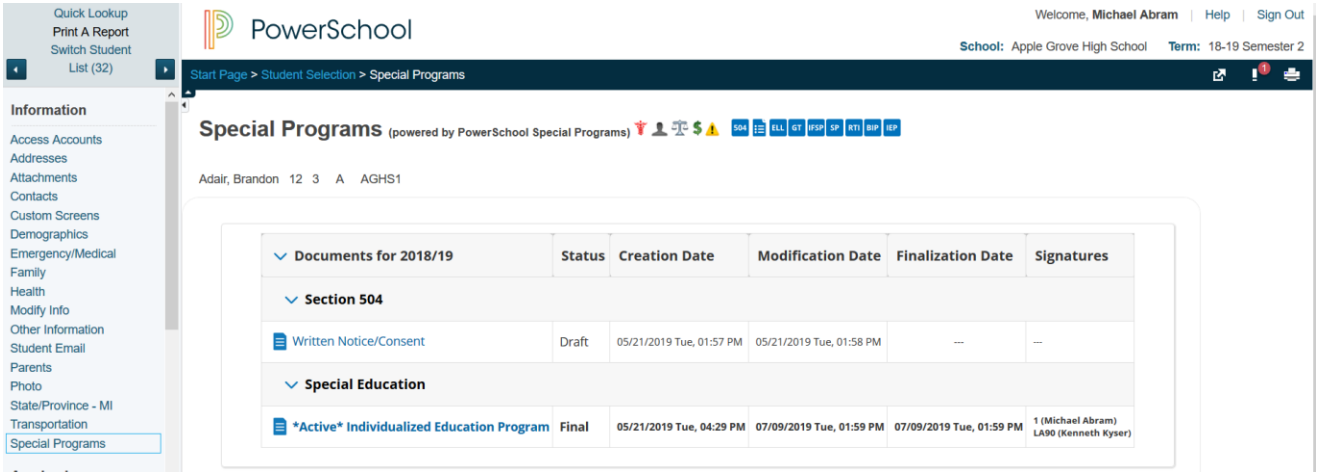
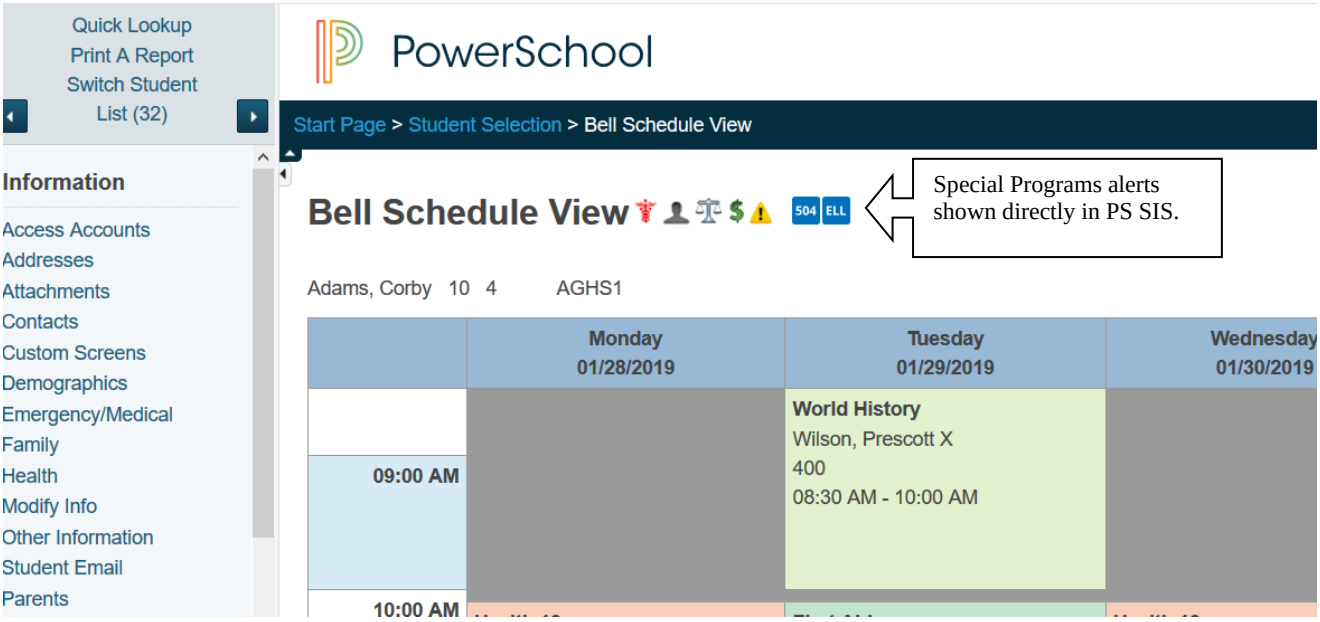
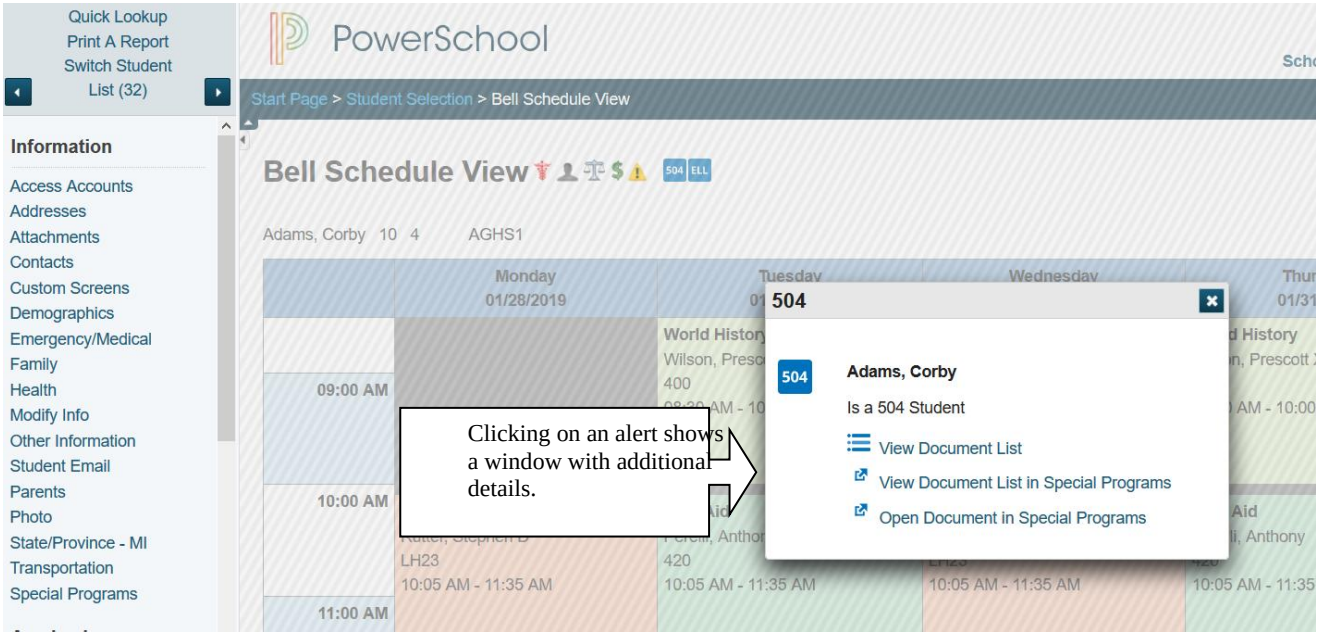


Figure 2 Embedded Special Programs Content

Alerts

Special Programs alerts can be configured to write values into PS SIS that indicate certain things for a student – these are called **Alerts**. In PS SIS alerts are displayed in student screens. This gives the ability for teachers to see that a particular student is currently in one or more special programs.





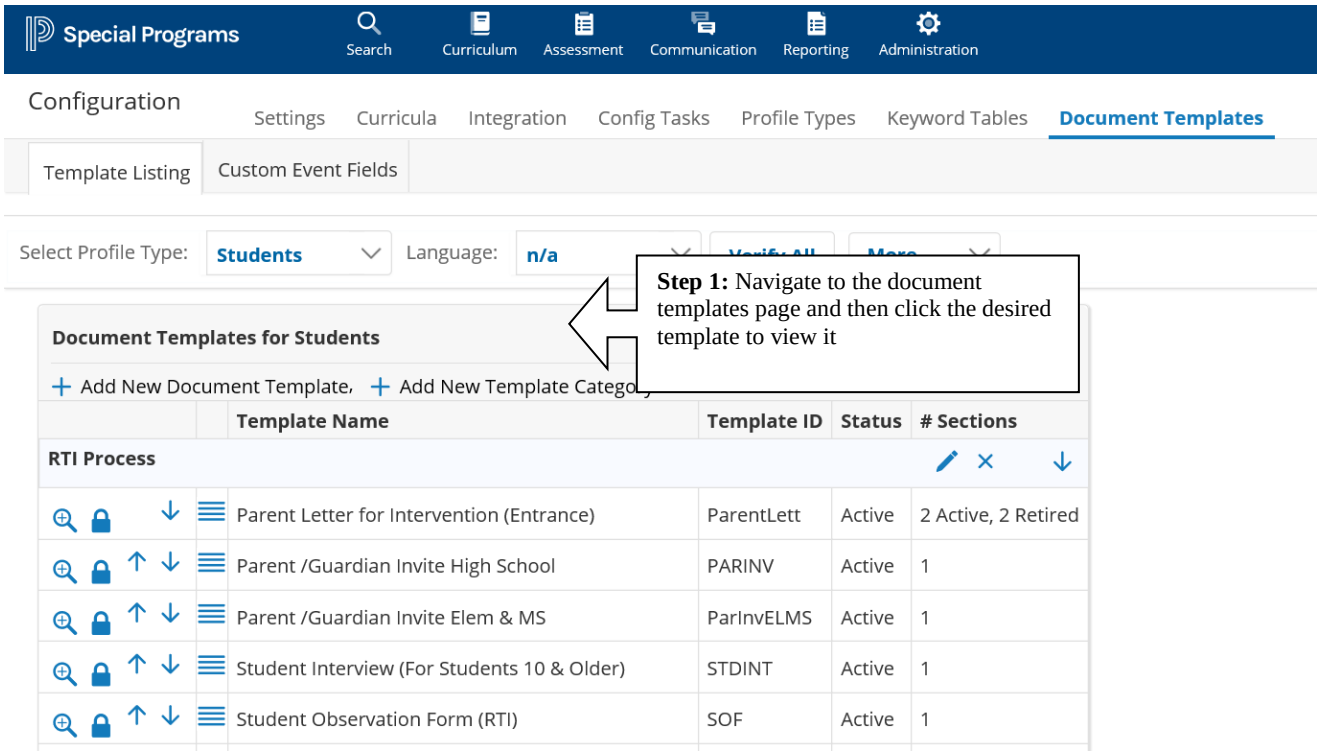
Each alert has 3 links:

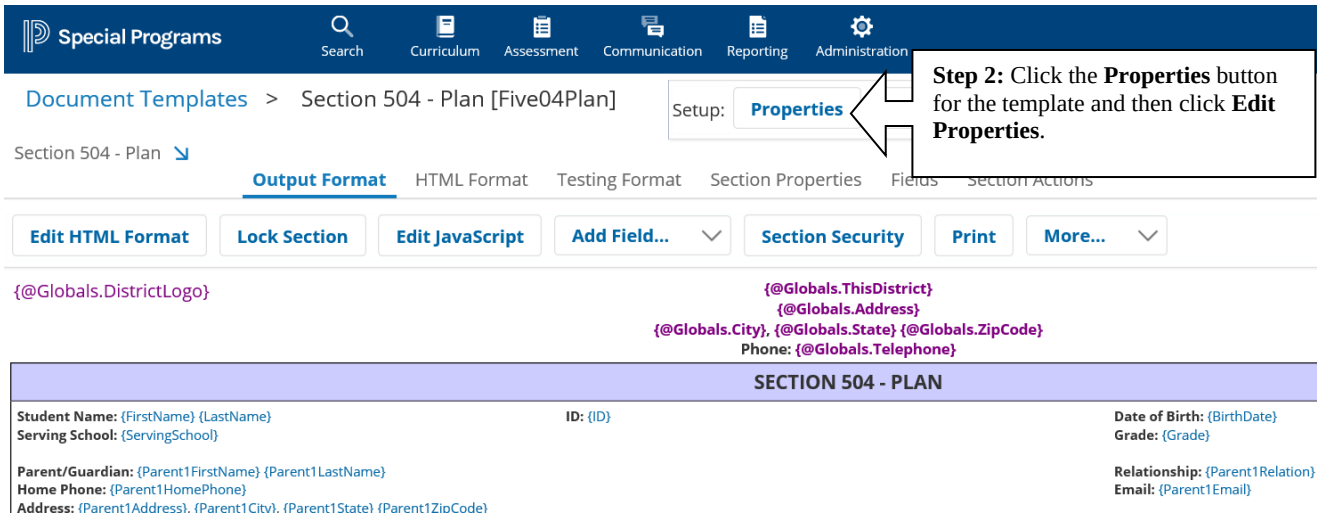
1. **View Document List** – view the embedded special programs page in PS SIS that shows the documents list for the student

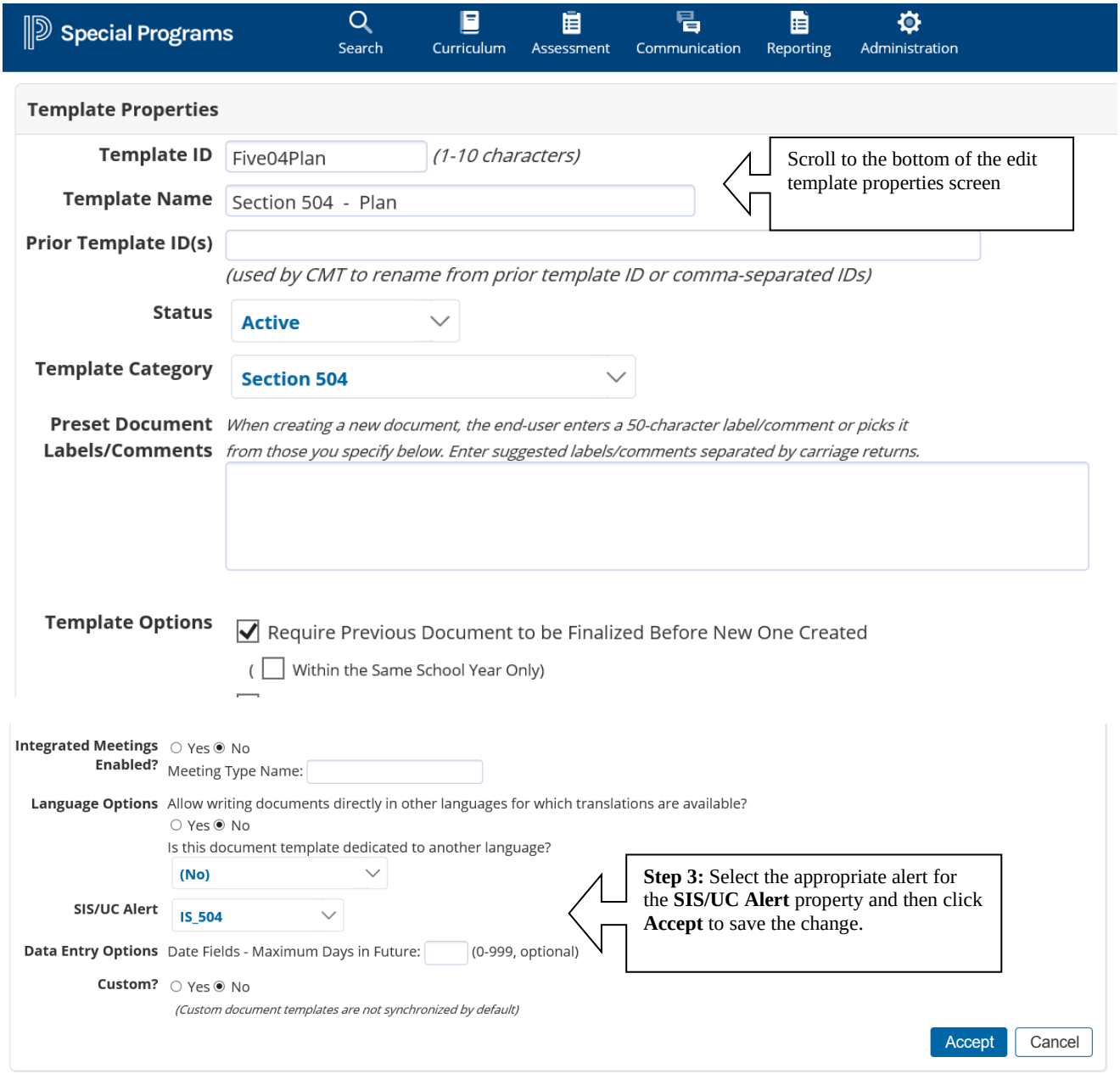
2. **View Document List in Special Programs** – SSO into PSSP and go to the documents list page for the student

3. **Open Document in Special Programs** – if configured correctly, this will SSO into PSSP and take the user to the document relating to the special programs alert

In order to get the 3rd link to work, some configuration is needed to associate an alert with one or more document types. This is configured in the individual document templates as a property.







Note that an alert can be associated with more than one document type. The reason for this is that it is not uncommon to have several document types used for the same purpose. For example, there may be several IEP document templates that are used depending on the age of the student.

.

HTML Format Directives Reference

Appendix A

HTML format directives are essentially commands enclosed in {squiggly} brackets that are embedded in an HTML format. To generate the actual output, PowerSchool Special Programs parses all directives in an HTML format and replaces them with whatever the directives stipulate: a field value or edit box, a correct pronoun for the student's gender, the result of a calculation, etc. PowerSchool Special Programs recognized the directives below when embedded in an HTML format, and in many cases, when embedded in narrative public/private statements or default text for a long text field.

Data Directive	Explanation
{FirstName} {LastName}	These directives make it easy to personalize text and use the student's name.
{.he,she} {.him,her} {.his, her} {.his, hers}	These directives are replaced with a pronoun appropriate for the student. In the directive, be sure you use the case that you want. For example, at the beginning of the sentence, use {.He, .She}, but in the middle of a sentence, instead use {.he,.she}.
{~*snippetname*}	Used to identify positions in the HTML format where the system administrator or CONSULTANT may insert "snippets", which are custom fragments of HTML formatting used to insert custom form content. The snippet HTML format may reference custom document fields. Each snippet is identify by the snippet name that you use in the directive, therefore snippet names must be unique within each section.

{=*formula*} {=~formula}	Allows you to insert the result of any valid PowerSchool Special Programs formula calculation in the format. If the format is for a document template, the formula must be expressed in terms of the document fields. If the format is for a profile form section, the formula must be expressed in terms of the profile field. The second format with the ~ character suppresses the white space that would otherwise be rendered if the calculated value is empty. This is useful in the case below to prevent a blank line if the calculated value is empty. {=@~*formula*}
{=^*parent_profiletype_formula}* {=^~*parent_profiletype_formula}*	In a document template, this syntax allows embedding of a formula directly in the context of the profile instead of the document. This directive will render as expected when using the print blank document feature, but otherwise this should be used sparingly both for performance reasons and to preserve the historical nature of the document (since the profile contains current data). An example is that if {=Profile.Location} is changed to {=^Location}, it will render even when printing a blank document. This syntax can be used for a similar purpose in child profile forms. In a child profile form, a calculated field in the parent profile could be referenced through an embedded calculation such as {=Student.Age}. However, this calculation would not show when adding a new child profile because it is based on the current child profile that does not exist yet. The {^*parent_formula}* syntax allows the formula to be written directly in the context of the parent profile such that it will be shown when adding a new child profile. In the example, {=Student.Age} would become {=^Age}. The version of the directive with the ~ character works the same as {=~formula} to suppress white space. For details, see explanation for {=~formula}.

{=*@globalsformula*} {=*@~globalsformula*}	Allows you to insert the result of a PowerSchool Special Programs formula expressed in terms of fields in the "Globals" profile. Every database has a single globals profile that contains system-wide or district-wide fields such as "DistrictSuperintendent". This is the most efficient way to create formulas based strictly on global fields and will work as expected even when printing blank documents. The version of the directive with the ~ character works the same as {=~formula} to suppress white space. For details, see explanation for {=~formula}.
{=$*jsvarname=formula*} {=@$*jsvarname=globalsformula*}	These variations of the PowerSchool Special Programs formula directive expose the value of the formula to JavaScript in a format consistent with the Data Form API. The value is accessed from JavaScript code via a reference to window.form$*jsvarname*.
{*fieldname*}	When the format is being used to display a form for viewing data, this directive inserts the value of the field. When the format is being used to display a form for editing purposes, this directive inserts an editable field such as a text box, check box, or dropdown menu.
{*fieldname*:*modifiers*}	Same as above, but allows for inclusion of various modifiers that affect the look and feel of the field. A complete list of modifiers is presented later in this section. Keep in mind that multiple modifiers can be included, and there should be no separation between them, for example, you could have a directive such as {StudentIsReady:D+"Ready"-"Not Ready"} for a logical field named StudentIsReady. These modifiers specify that a dropdown menu be used to enter this field, and that "Ready" or "Not Ready" be used in place of "Yes" or "No".

{*-fieldnames*}	This directive resets the specified fields (comma-separated list of field names) to their default values, but only if the fields are not visible (in editable form) on the respective form at the time the user submits the form. This directive is compatible with the #IF and #JSIF directives. In the example below, the {-OtherText} directive ensures that the OtherText field is reset to its default value if the Service field is on the form but the OtherText field is not: {#IF StudentNeedsService} <label>Service:{TypeOfService}</label> {#IF JSIF Service.Other,tag=span} <label>Other: {OtherText}</label> {#ENDIF} **{-OtherText}** {#ENDIF}
{**constraintname*}	In a profile form, depending on the evaluation method of the constraint, this directive either anchors a constraint user message to a particular point on the form or generates a constraint button. Note that constraint user messages are displayed above the form by default, but can alternatively be anchored to a particular field or to a precise place on the form (using this directive). If the constraint has an evaluation type of 'Explicit Button', the directive instead generates a button on the form that, when clicked, evaluates the constraint for execution. By default, the button shows only when the form is in edit mode. By default, the label of the button is the constraint name and the CSS class "CONSTRAINTBUTTON". This can be modified as described in the next row below.

{**constraintname*:*modifiers*}	Modifiers only apply if the directive is for a button (see row above). The full set of possible modifiers for a constraint button is {**constraintname*:L"*label*"C"*css class*"VE}. The L modifier specifies a button label other than the constraint name. The C modifier specifies a CSS class name other than "CONSTRAINTBUTTON". V specifies that the button will be shown when the form is in view mode, and VE specifies that the button will be shown in both view and edit mode.
{**sectionactionname*}	In a document template, this directive anchors the user message for a section action with a "Rollback Saving Section" or "Prevent Section Completion" trigger, or generates a button for a section action with a "Modify Data Upon Save" trigger. The button only shows in edit mode. By default, the button label is the name of the section action and the CSS class name is "SECTIONACTIONBUTTON". This can be modified as described in the next row below.
{**sectionactionname*:*modifiers*}	The full set of possible modifiers for a section action button is {**sectionactionname*:L"*label*"C"*css class*"}. The L modifier specifies a button label other than the section action name. The C modifier specifies a CSS class name other than "SECTIONACTIONBUTTON". It is possible to have a button not linked to a section action that acts like the "Save, Continue Button". The format for this is {*:L"*label*"} (not the colon after the asterisk). The A modifier makes the button appear as a hyperlink instead of a button.

{|*profilesectionname*:*modifiers}* {|*childprofiletypename*:*modifiers}*	For document templates only. When a user is editing a section of a document, it may be usedful to give the user quick view access to a particular section of the corresponding profile (e.g. student profile) at a particular point in the editing process. This directive produces a button (or alternatively a hyperlink) that opens a specified profile form section or list of child profiles in a popup window for quick access. The full syntax for a button is {|profile_section_name:C"CSSClass"L"label"}. If the CSS Class or label is missing, defaults will be used. {|profile_section_name:AC"CSSClass"L"label"} will produce a hyperlink (note the A modifier). Use the plural name of the child profile type in place of the section name to open a popup with a list of child profiles. Note that the button or hyperlink will only appear in edit mode.
{!*assessmentdirective*}	Assessment directives are preceded with an exclamation point, are supported for student profile form sections only, and allow scores and information from the assessment repository and/or curriculum to be included in a profile form section. These are covered in detail in the next section of this appendix
{&*sectionname*}	Supported in HTML formats for document template sections only. At runtime, this directive is replaced with the HTML output for the named section from the same document. This is very useful for quickly building an IEP section that will work as a "Snapshot" or shortened IEP. Note that the section with this directive (not the referenced section) should have the "Always Regenerate When Not Final" property; otherwise, the included content may not be kept up to date.

{&*templatecode*::*sectionname*::*filter*}	Supported in HTML formats for document template sections only. At runtime, this directive is replaced with the HTML output from the referenced other document and section. The filter is used to select a particular source document from the template since there may be more than one. The filter is a comma-separated list of one or more join equalities where the left side is an expression in the context of the current/target document, and the right side is an expression in the context of the source document. For example, the following takes a section from a source IEP document from the same school year: {&IEP::Services::DocHistoryYear=DocuHistoryYear}. If more than one potential source document exists in the filter, the one that was last created is selected. If no source document is found, of if it does not include the section, no content is included. Note that you can design a section in the source document designed eclusively for referencing from another document. Just enable the "Is Hidden" and "Other" options in this source section. Note that the section with this directive (not the source section) should have the "Always Regenerate When Not Final" property; otherwise, the included content may not be kept up to date.
{%*childprofilegrid*}	Is replaced by a formatted grid (HTML Table) listing rows from child profiles of the profile for which the form is being rendered. Also works with documents, although note that the information is not drawn from document fields, but rather child profiles of the current profile. This kind of directive is covered in detail later in this section.

{^*childtemplateid*} {^*childtemplateid*:RH}	Used to position a repeating row grid in a repeating document section. This also supports several modifiers. {^*childtemplateid*:R}implements a requirement that the user must enter at least one row of data for the section to be complete. {^*childtemplateid*:H}hides the entire repeating row grid if it contains no rows of data. The grid is not hidden in edit mode so that rows can be entered.
{'*tabid:tabtype*}	This directive is used to position a DocuSign e-signature tab. The tabid is either the name of a staff profile reference field marked as a DocuSign signer, or the tab ID for other signers configured for the document template. The tab variant is one of the following: FullName, SignHere, DateSigned, Checkbox.
{>}	Used to position a dynamic barcode in a document section. Note that the directive should be exactly {>}, otherwise, the platform will interpret as {>regionname}, described next.
{>*namedblock*}	Integrates a read-only clone of a named block of content marked in another source section within the same document template. The named block should be marked in the source section as follows: {#BLOCK-<name>} content {#ENDBLOCK} **Limitations:** Blocks defined in the HTML format of a repeating section can only be used in another repeating section based on the same child template. Blocks cannot be nested. While a block is allowed to include a {^repeatingrow} directive, this directive will not work as expected in another section that uses the block unless the other section also has a repeating row layout defined for the same child template.

Conditional Directives (available in profile form sections and document template sections)	
Conditional Directives	Explanation
{#BLOCK-<name>} content {#ENDBLOCK}	Marks a named block of section content such that another section can include that block in view-only form via a {>*namedblock*} directive. **Limitations:** Blocks defined in the HTML format of a repeating section can only be used in another repeating section based on the same child template. Blocks cannot be nested. While a block is allowed to include a {^repeatingrow} directive, this directive will not work as expected in another section that uses the block unless the other section also has a repeating row layout defined for the same child template.
{#IFNEW} *HTML Formatting* {#ENDIF}	This is available for profiles only and includes the HTML formatting if and only if the user is manually adding a new profile. Once the profile is saved and created, this directive will not include the HTML formatting.
{#IFEDIT} *HTML Formatting* {#ENDIF}	The HTML formatting between the {#IFEDIT} and {#ENDIF} directives is only included when the form is used in edit mode.
{#IFVIEW} *HTML Formatting* {#ENDIF}	The HTML formatting between the {#IFEDIT} and {#ENDIF} directives is only included when the form is used in view mode.
{#IF formula} HTML Formatting {#ENDIF}	The HTML formatting between the {#IF *formula*} and {#ENDIF} directives is only included if the formula evaluates to true. If this is used in a profile form section, the formula is expressed in terms of the profile fields. If this is used in a document template section, the formula is expressed in terms of the document fields.

{#IFVIEWOR formula} HTML Formatting {#ENDIF}	The HTML formatting between the {#IFVIEWOR *formula*} and {#ENDIF} directives is included if the form is in view mode (formula ignored in this case), or if the form is in edit mode and the formula evaluates to true.
{#IFEDITOR formula) HTML Formatting {#ENDIF}	The HTML formatting between the {#IFEDITOR *formula*} and {#ENDIF} directives is included if the form is in edit mode (formula ignored in this case), or if the form is in view mode and the formula evaluates to true.
{#IF-RO formula} HTML Formatting {#ENDIF}	If the formula evaluates to true, then any fields in the content are forced to be read only.
{#IF_USER *formula*} HTML Formatting {#ENDIF}	If the current user meets the criteria defined in the staff-based formula (or if the user is the ADMIN or CONSULTANT), the HTML formatting is included. This is only available for profile forms, not document templates (where it would cause problems with finalized documents).
{#IF_EDITANDUSER *formula*} HTML Formatting {#ENDIF}	If the form is in edit mode and also the current user meets the criteria defined in the staff-based formula (or if the user is the ADMIN or CONSULTANT), the HTML formatting is included.
{#IF_USER_RO *formula* } HTML Formatting {#ENDIF}	If the current user meets the criteria expressed in the staff-based formula, the contents will be read-only.
{#IF_ADMINCONSULTANT} HTML Formatting {#ENDIF}	If the current user is an ADMIN user, CONSULTANT user or staff user with the 'Access Admin Form Content' privilege, the HTML formatting is included. This is only available for profile forms, not document templates (where it would cause problems with finalized documents).
{#IF_ADMIN} HTML Formatting {#ENDIF}	If the current user is an ADMIN user or staff user with the 'Access Admin Form Content' privilege, the HTML formatting is included. This is only available

	for profile forms, not document templates (where it would cause problems with finalized documents).
{#IF_CONSULTANT} HTML Formatting {#ENDIF}	If the current user is the CONSULTANT, the HTML formatting is included. This is only available for profile forms, not document templates (where it would cause problems with finalized documents).
{#IF_EDITANDADMINCONSULTANT} HTML Formatting {#ENDIF}	If the form is in edit mode and also the current user is the ADMIN or CONSULTANT, the HTML formatting is included.
{#IF_EDITANDADMIN} HTML Formatting {#ENDIF}	If the form is in edit mode and also the current user is the ADMIN, the HTML formatting is included.
{#IF_EDITANDCONSULTANT} HTML Formatting {#ENDIF}	If the form is in edit mode and also the current user is the CONSULTANT, the HTML formatting is included.
{#IF_USERORFINAL *formula*} HTML Formatting {#ENDIF}	If the current user meets the criteria defined in the staff-based formula (or if the user is the ADMIN or CONSULTANT), the HTML formatting is included. This is only available for document templates and only when the current section has the "Always Regenerate When Not Final" property.
{#IFFIELD *fieldname*} HTML Formatting {#ENDIF}	This directive is available only in profile forms and is intended to provide support for field level security. If you wish to exclude part of the form section markup if a particular field is unavailable due to field level security, enclose that part of the markup with this directive. This will conditionally include the markup only if the specified field is available.
{#JSIF *fieldreference(s), options*} HTML Formatting {#ENDIF}	See the " In the context of documents only, several functions are available that are not part of the API but which nonetheless can be called to invoke one of the save or cancel editing buttons. These functions are as follows:

	• doSaveDoneEditing(): This function is quivalent to clicking the "Save, Done Editing" button. • doSaveContinueEditing() or doSaveContinueEditing(bSkipAttemptComplete): This function is equivalent to clicking the "Save, Continue Editing" button. Normally, clicking the "Save, Continue Editing" button also checks whether the section meets all completion requirements, and if so, marks it as complete. To avoid checking for section completion, the doSaveContinueEditing function takes an optional bSkipAttemptComplete parameter for which you can pass true to skip the behavior (e.g. doSaveContinueEditing(true)). • doCancelEditing(): This function is equivalent to clicking the "Cancel, Editing" button. Conditional Form Logic: #JSIF Directive" section on page 333.
{#IF_SERVICECAPTURECONTEXT "*modifiers*"}HTML Formatting{#ENDIF}	Supported in service capture forms only. This allows arbitrary content to be included or excluded based on the service capture context (individual service, group service, etc). The modifiers are exactly the same ones supported by the "M" field modifier described in the next table.
{#IFTRANS languageCode} HTML Formatting {#ELSEIFTRANS *languageCode*} HTML Formatting… {#ELSE} HTML Formatting {ENDIF}	Used in document templates only. This allows the document template to show different content depending on the language version that is being viewed or edited. The content in the #ELSE block is shown for the default (non-translated) version of the document.
{#TRANS_BLOCK_BEGIN} content {#TRANS_BLOCK_END}	Used in document templates only. Used to ensure that the enclosed content is translated as a single block.

{#TABLE_HEADER} <table> <thead style="display:table-header-group"> <tr> <th>colheader1</th> <th>colheader2</th> </tr> </thead> {#END_TABLE_HEADER} <tr><td>cellcontent1</td> <td>cellcontent2</td></tr> {#TABLE_FOOTER} </table> {#END_TABLE_FOOTER}	Support in profile form sections only. These directives can be used to construct a list style view that spans multiple forms, based on an HTML table construct. When multiple forms are printed, the content between {#TABLE_HEADER} and {#END_TABLE_HEADER} are included only for the first form printed, and the content between {#TABLE_FOOTER} and {#END_TABLE_FOOTER} are included only for the last form printed. The <thead> style used in the example to the left ensures that the column headings will print on each page if the list runs to multiple pages. **Important:** If you use this in a form section, be sure to enable the "No Page Break When Multiple Forms Printed" checkbox in section properties or strange results will occur! **FYI – About Vertical Column Headers:** To achieve vertically oriented column headers, add the style ".VERTICAL{ width:1px; writing-mode:tb-rl; }" to the style sheet and use it as a class for your column headers, <td class=VERTICAL>

FYI-Important: About Conditional Directives: The word immediately after the pound sign (e.g. "IFEDIT" in {#IFEDIT}) must be in upper case. Any word(s) immediately after #ENDIF are ignored so that you can document what the #ENDIF corresponds to, for example {#ENDIF-IFVIEW}.

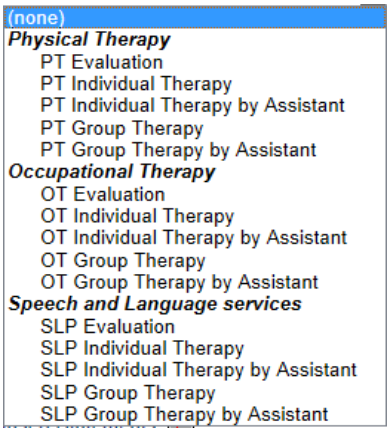
The follow modifiers are supported for the {*fieldname*:*modifiers*} directive described previously:

Field Directive Modifiers (available in profile form sections and document template sections)	
Modifier	Explanation
~	Suppress the white space that is by default inserted for an empty field.
!	In profile forms only, marks the field as required. The form will not be accepted until this field is filled in.
@	(ADVANCED) This is supported in document templates only and makes the field invisible to the user when in edit mode, but such that it is still an internally

	"editable" field that is writable by JavaScript. The modifier has no effect in view mode. A variation of this modifier (currently supported for character and non-stylized long text fields only) allows the invisible field to still display its value as plain text while in edit mode, even when the field has been modified by JavaScript. Instead of a directive like {Field:@} which only gets you invisibility, use {Field:@"V"} which prevents the editable field from showing, but automatically displays the value as plain text in the same position. It may also be useful to apply any of the above variations to a field that is marked as read only in the data dictionary. In this case, you can include an E modifier, such as {Field:@E}, to force it to be editable but the @ modifier still makes it invisible to the end user.
^	(ADVANCED) This is supported for character, integer, numeric, date, date/time, keyword, shorttext, and long text fields (except stylized text fields) and makes the field writable by javascript but appears to the user as a grayed out read-only field. NOTE: In versions prior to 22.6.4.0, only the character, shorttext, and long text data types were supported.
A	This is supported in document templates only. It is used in conjunction with dropdown and checkbox fields only. If the user clicks a checkbox or changes a dropdown with this modifier, the "Save & Continue Editing" button will automatically be invoked.
C C"="	Keyword field: If used in a section linked to multiple curriculum outlines (e.g. IEP goal section), this modifier directs the system to use the keyword identified in this table as the root of the curriculum that should appear when the user clicks the button to select from a curriculum. If you use the C"=" variation, that further instructs the system to not allow the user to change the curriculum in the curriculum popup, and to furthermore prevent the user from bringing up any curriculum until one is selected in this field. Note that the keyword table must provide the curriculum root corresponding to each keyword. This can be done either by making the keywords identical to the curriculum root, or alternatively by adding a character column to the keyword table named "CurriculumRoot" that identifies the curriculum root for each keyword.
D	The behavior of this modifier depends on the data type: • If used with a logical (yes/no) field, the Yes/no options will be presented in a dropdown menu rather than using checkboxes.

	• If used with an integer field that has a minimum value and a maximum value, the integer field will be presented as a dropdown menu of all values between the minimum and maximum inclusive. You can also specify an interval. For example, if a field has a mimimum value of 5 and a maximum value of 25, you can present {5, 10, 15, 20, 25} in the dropdown using the syntax (myinteger:D"5"} where 5 is the interval. • If used in with a keyword field, the options will be presented as mutually exclusive checkboxes instead of a dropdown menu. Mutually exclusive checkboxes can also be configured as horizontal list of checkboxes or checkboxes in a grid layout. Simply specify the directive as {keywordfield:DW###) where ### should be replaced by the desired number of columns. If this number is equal to or greater than the number of keywords, this will result in a horizontal layout. If the W modifier is omitted, the default number of columns is one resulting in a default vertical layout. Any number between results in a grid layout. • If used in with a date/time field, the field will show only the date component if the "D" is upper case or only the time component if the "d" is lower case. This allows you to position the date and time components in different positions on the form (you have two directives – one with D and one with d). Note that you cannot have an editable time component without an editable date component, although you can make the date component invisible using the "@" modifier as long as date component will always be filled with a default value.
E	For document templates only. In certain circumstances, you might want to force a field that is otherwise marked as read-only in the data dictionary to be editable in a specific special situation. To do this, apply the E modifier.
F"*filterspec*"	If used in conjunction with a keyword selection field, the *filterspec* must be the name of the keyword table field/column to be used for filtering. The dropdown menu will only show keywords for which the specified keyword table field/column is neither EMPTY nor FALSE. You can also specify multiple fields/columns in the *filterspec* by separated them with columns. If the first keyword table field/column referenced in *filterspec* is itself a keyword selection field referencing a different keyword table, the dropdown menu will have a hierarchical organization that categorizes the keywords based on the values in that field/column. An example would be "diagnostic code" keywords that are categorized by the service type to which they apply. In this case, 'service type'

would be a field/column in the diagnostic codes keyword table that references a services types keyword table. See how this looks below.



If used in conjunction with a profile reference field, the *filterspec* will be a logical criteria formula for filtering the results displayed by the lookup popup window. The formula should be in the context of the profile type being looked up. Note that when a filter formula is applied using the F modifier this way, the lookup results immediately appear without the user needing or even being allowed to conduct a general search. But if you would like the user to be able to "escape" the filter and conduct a general search, you can add a hint in the form of a comment appended to the end of the formula, like this F"*formula*;AllowSearch". Note that the filter formula does not prevent the user from directly typing an ID in the textbox that is outside of the filter. If strict validation is required, a section action must be added to supplement the filter.

In certain circumstances, it is possible to embed information from the current profile/document section into the filter formula for a profile reference field using a placeholder enclosed with $ characters. The placeholder can be a name of a profile/document field linked to the same profile/document section. Documents (not profiles) also support the symbol "Profile" enclosed in $ characters, which embeds a reference to the profile of the document. For example, in a student document, the directive below filters the lookup window to any teachers of any classes the student is in.

```
{Teacher:LF"EXISTS(ClassStaffRoster,
  EXISTS(ClassStudentRoster,Student=$Profile$))"}
```

If used in conjunction with a school year type data field, the *filterspec* will be a range of years relative to the current school year. For example F"-1,+1" will

	include last year, the current year, and the next school year. F"-10,0" will include the last ten years and the current year, but not future years. If used in conjunction with a long text field on a section linked to the curriculum, this can specify a comma-delimited list of curriculum outline level names (singular form) to which the long text field will be mapped. The end-user will only be able to insert statements at those levels into the box. If the level names are prefixed with a tilde character (e.g. F"~AnchorStandard"), the curriculum statement labels will be inserted into the long text field instead of the descriptions.
F"*field/column"* O"*masterfield*"	The previous description for the F modifier describes the capability to produce a "hierarchical" keyword dropdown menu if the F modifier references a keyword table field/column that provides categories (i.e. a keyword table field pointing to a second keyword table that delineates the categories). In this specified situation only, the O modifier can be introduced in conjunction with the F modifier to dynamically filter the keyword dropdown based on the user's selection from a specified "master" keyword field dropdown (identified by *masterfield).* As an example, assume that there is a "ServiceType" field (keyword dropdown) and a "DiagnosticCode" field (keyword dropdown). The keyword table containing the dialog codes has a column named CodeServiceType that identifies the service type corresponding to each diagnostic code. Then a directive like {DiagnosticCode:LF"CodeServiceType"O"ServiceType"} can be used to link the DiagnosticCode field such that the DiagnosticCode field is dynamically filtered based on the ServiceType field. The F and O modifier combination will work in other situations as well. Assume that there is a "ServiceType" field (keyword dropdown) and a "DiagnosticCode" field (keyword dropdown). The keyword table containing the service types has a column named "Category" that identifies the service category corresponding to each service type. The keyword table containing the diagnostic codes also has a column named "ServiceCategory" that identifies the service category corresponding to each diagnostic code. Then directives like those shown below can be used to link the DiagnosticCode field such that the DiagnosticCode field is dynamically filtered based on the ServiceType field. {Service:LF"Category"} {DiagnosticCode:LF"ServiceCategory"O"ServiceType"}

	Note: This type of filtering should also be backed up by profile constraints, since it is possible for a user to use the back button to "trick" this filtering system.
H##	Used in edit mode only for data fields with a data type of short-text or long-text. Gives the height (in lines) of the multi-line text box (replace ## with the actual height).
I##	Used only for long text fields in document templates only. Gives an alternative height in lines (replace ## with the actual height) used when the document is printed with extra space for handwriting (an option when printing a document). If used in conjunction with a logical field, please review the Digital Signature area for more information.

J	This modifier allows you to write a javascript notification function that automatically gets called when the value of the field changes on a form in edit mode. In the field directive, you specify the javascript function as in the example {Goal:W100H5J"api:onChangeGoal"} where onChangeGoal is the name of a function you define in the JavaScript tab. The javascript function will receive as a parameter the name of the field that has changed. If the field in question has an auto-postback, the javascript function will be called just before the auto-postback occurs.
	
	`function OnChangeGoal(strFieldName) {`
	`  alert('New value is: ' +`
	`DataFormAPI.getFieldValue(strFieldName));`
	`}`

K	In profile forms only, this identifies the field as a critical field with an asterisk. This is a visual effect only and entry of the field is not required by virtue of including this modifier.
L L"fieldlabel"	Automatically renders a field label for the field, both in viewing and editing mode. This assumes the fields are being laid out in a two column HTML table where the left column contains field names and the right column contains field values. Hence this directive is replaced with something like <tr><td>fieldlabel</td><td>fieldvalue</td></tr>.

	PowerSchool Special Programs will render a default field label based on the field name, but you can override that by including a specific field label in quotes. If used in conjunction with the S or the I directive with a logical field, please review the Digital Signature area for more information.
M"SDXI"	Used in service capture (service record) forms only to indicate whether a field should be suppressed or not in various situations concerning whether the user has selected multiple students, multiple days, group service, individual service, etc. The M field attribute supports a number of options, as follows: S – Suppresses field if entering common information for multiple students. s – Suppresses field if not entering common information for multiple students. D – Suppresses field if entering common information for multiple days. d – Suppresses field if not entering common information for multiple days. X – Suppresses field if scheduling services in the future. x– Suppresses field if not scheduling services in the future, which implies recording services in the past. I – Suppresses field if entering individual service records after having entered common information for multiple students. G – Suppresses field if entering common information for group service (versus individual service). Note that prior to Version 11.0, selecting multiple students always implied group service, however, it is now possible to enable an option in the service record profile form section that supports a new scenario where the user selects multiple students and then creates individual (not group) service records at different times on the same day(s). g – Suppresses field if not entering common information for group service. See the notes for the G option above. M – Suppresses field if entering common information for multiple students, but for individual student service, not group service. Note that prior to Version 11.0, selecting multiple students always implied group service, however, it is now possible to enable an option in the service record profile form section that supports a new scenario where the user selects multiple students and then

	creates individual (not group) service records at different times on the same day(s). m – Suppresses field if not entering common information for multiple students, but for individual student service, not group service. See the notes for the M option. **FYI –** You can control the inclusion of arbitrary form content using the **Use of {#IF_SERVICECAPTURECONTEXT "*options*"}*content*{#ENDIF}** directive. The options here are the same options that are available for the M field modifier.
N	The behavior of this modifier varies according to the data type: Profile Reference: If the referenced profile type contains people (students, staff), this causes the referenced person's name to be displayed in first last-name format, rather than last-name, first format. Logical field: If the logical field allows EMPTY values, this causes the No option to be presented before the Yes option (by default, Yes is presented before No). Keyword Selection: Normally, when keyword descriptions appear in a dropdown, and the descriptions are greater than 25 characters, the description may be truncated at the first parenthesis to remove excessive detail. Including the N modifier prevents this truncation.
O	If used with a profile reference field, can be used to specify the name of a character field that will store a "Non-Lookup" text value where the user can type whatever the user wishes. For example, the following will allow the user to choose between a nurse staff lookup field and a non-lookup alternative character field. <label>Nurse: {Nurse:O"NurseNonLookup"}</label> The "O" modifier can be used with a logical field checkboxes to organize them into a "require one or more checks" group in a way that is consistent with them being a single field. The red/yellow colors are applied and change as expected. This is implemented as follows: • Arrange the logical fields on the form inside of a common parent element such as , <p> or <fieldset>. Note that this common element will change color when the first checkbox is checked.

	• In the directive for each of the logical fields in the group, use the "O" modifier to specify a "group name" that is unique to the group of checkboxes. For example, {logicalfield:L"label"O"groupname"}. If used with a date or datetime field, allows you to specify a precise output format for the value. An example of an output format is "ddd, dd/mm/yyyy" which would output something like "Mon, 02/28/2012". The following placeholders can be used in the format: • d (day, 1-31), dd (day, 01-31) • ddd (day of week, Mon-Fri), dddd (day of week, Monday-Friday) • h (hour, 1-12), hh (hour, 01-12), H (hour, 0-23), HH (hour, 00-23) • m (minutes, 0-59), mm (00-59) • M (month, 1-12), MM (month, 01-12) • MMM (Jan-Dec), MMMM (January-December) • s (seconds, 0-59), ss (seconds, 00-59) • tt (AM/PM) • yy (year with 2 digits), yyyy (year with 4 digits) • Non-letter characters are copied to the output as is, except that \ is an escape character. For example, {MyDateTime:O"HH\h\mm"} would output a time that looks like 23h12. If used with a numeric or integer field, allows you to take control over the output format for the value. Examples are below: • O"c" 14.1 => $14.10, £14.10, … (currency, server culture) • O"X2" 15 => 0F (hexadecimal from integer only) • O"x4" 254 => 00ffe (hexadecimal from integer only) • O"n0" integer 9999001 => 9,999,001 • O"n2" numeric 9999001.1 => 9,999,001.10 • O"#.##" 0.3678 => .37, 0.3 => .3 • O"0.00" 0.3678 => 0.37, 0.3 => 0.30 • O"p" .371 => 37.1% (percentage) • O"p1" .37 => 37.0% (percentage)

P	If used with a character field, the field value is treated as a fully qualified URL of an image, and the image itself is displayed instead of the URL text (when not in edit mode).
R	The field is read-only or view-only, even if shown in a form being used for editing.
R"*StaffRefField*"	This specifies that the field will be conditionally readonly depending on who the current user is with respect to the staff reference field specified by *StaffRefField*. The field will be editable in either of the following two cases: 1) the current user is ADMIN or CONSULTANT, 2) the current user is a staff user and the *StaffRefField* field is either empty or points to the current staff user. In all other situations, the field will be read only. Note that *StaffRefField* must appear somewhere on the same form for this to work. Often used with the U"*triggerfield*" modifier to allow a user to take ownership of the values of a group of fields on a form. In the example below, the user who sets the value of indicator locks down not only its value but the value of the comment field so that other users (other than ADMIN or CONSULTANT) cannot edit them. The identity of the user and the date and time of this event is tracked as well. <label>Indicator:{Indicator:R"IndicatorUser"}</label> <label>Comment:{IndicatorComment:R"IndicatorUser"}</label> <label>User: {IndicatorUser:U"Indicator"}</label> <label>Date: {IndicatorDate:U"Indicator"}</label>
S	The behavior of this modifier depends on the context. • If used in a profile form with a short-text field, a spell-check link will be included next to and for the field when in editing mode. • If used in conjunction with a long text field in a document template, an "Insert Statements" link will appear even if the field presently has no public statements. • If used in conjunction with a basic character field in a document template, the field is spell checked. By default, character fields are not spell checked. • If used in conjunction with a logical field, please review the Digital Signature area for more information.

S"*categoryfield*"	This can be used to potentially reduce the number of distinct long text fields needed in a document section. The main concept is that there is a keyword field on the form that works in conjunction with a long text field. The keyword field selects the category of "insert statements" that appears when the user clicks the "Insert Statements" link for the long text field. Furthermore the user cannot change the category that has been automatically selected (the category dropdown is not visible in the insert statements popup window). The S"*categoryfield*" attribute is applied to the long text field, and *categoryfield* identifies the keyword field which must be editable on the same form. They keyword table for the keyword field must be set up in a particular way. It must include a character column named "StatementsCategory" that specifies the category name corresponding to each keyword. If language localization is being used, there can be an additional column for each localized language named "StatementsCategory_xx" where xx is the ISO language code (e.g. es for Spanish). This would specify the localized category name for the language. Note also that it is important to place the insert statements for the long text field "under configuration management" so that they can only be edited in a controlled fashion (that will not break the relationship to the keyword table). Lastly, use of this feature will disable "private" insert statements because those would not be differentiated by category.
T T"fieldlabel"	Similar to the L directive above, but this does not assume that fields are being laid out in an HTML table, and therefore renders something like fieldlabel: fieldvalue.
U	As of Version 17.1, this modifier is supported for character fields (documents only) to support situations where the value is editable in the section but must be dynamically displayed and updated elsewhere in the same section. There are three specific scenarios where this is supported: 1) A top level character field is editable in a section but the value must be dynamically displayed in all repeating rows on the same section. In the repeating row, the outer field is referenced via {*FieldName*:U} where U sets up the dynamic updating. When the value of the editable field is changed, the values are automatically updated in each repeating row when the editable field looses focus.

	2) Similarly, in a repeating section, a character field that is editable may be displayed in each repeating row nested within the repeating section. 3) A field is editable in the section but the value must be mirrored in another place on the same section, but not in repeating rows. In this case, the mirrored field should be marked with both R (for read only) and U (for automatic updating).
U"*triggerfield*"	Instructs the system to automatically update or stamp the field when the value of another field specified by *triggerfield* changes in value. The field being stamped is automatically made readonly to the user since its value is being stamped. The value to stamp the field with is inferred from the data type of the field. At this time, only three data types are supported for the field being stamped. If it is a date or date/time field, it is stamped with today's date or now's date/time respectively assuming that the trigger fields's new value is not empty, but if the trigger field's new value is empty, the date or date/time field is cleared. If the field being stamped is a staff reference field, it is stamped with a reference to the staff profile of the current user (or EMPTY if current user is not a staff profile user) assuming that the trigger fields's new value is not empty, but if the trigger field's new value is empty, the staff reference field is cleared. Other data types are not supported at this time. IMPORTANT: the triggering field must appear as an editable field on the same form as the field being stamped, and the directive for the triggering field must appear in the layout before that of the stamped field. The stamped field must be on the form at the time form is submitted. If the stamped field is omitted via a conditional directive, it will not be stamped. If necessary, you can diagnose this by searching the page source for "SELFUPDATE" and see which fields to be stamped are on the form at any time. Often used with the R"*StaffRefField*" modifier to allow a user to take ownership of the values of a group of fields on a form. In the example below, the user who sets the value of indicator locks down not only its value but the value of the comment field so that other users (other than ADMIN or CONSULTANT) cannot edit them. The identity of the user and the date and time of this event is tracked as well. <label>Indicator:{Indicator:R"IndicatorUser"}</label> <label>Comment:{IndicatorComment:R"IndicatorUser"}</label> <label>User: {IndicatorUser:U"Indicator"}</label> <label>Date: {IndicatorDate:U"Indicator"}</label>
V"field"	If used in conjunction with a multiple-value field, this modifier allows you to specify a field in the associated keyword table to use for filtering the values (determining a subset of values to be displayed). Only values for which the

	specified field is not EMPTY or is not FALSE in the keyword table will be displayed.
W## W##%	Used in edit mode for data fields that use a text box. Gives the width of the text box in number of characters based on an average character width (replace ## with the actual width). For long-text fields only, you can also express the width as a percentage of the available space (e.g. W50%).
Z	The behavior of the Z modifier depends on the data type of the field: • **Keyword Fields:** In certain scenarios, it may be useful to have a keyword field dropdown next to a second editable field, but then upon saving, the value of the second field is displayed embedded in the middle of the keyword description. The "Z" modifier for the keyword field directive makes it possible to embed macros in keyword descriptions that get evaluated when the form is not in edit mode. For example, a keyword field named "Measure" has a keyword description of "Student will score between {MinScore} and {MaxScore}", which includes macros for MinScore and MaxScore. The markup for Measure, MinScore, and Maxscore fields can be something like {Measure:TZ}{#IFEDIT}{MinScore:T},{MaxScore:T}{#ENDIF}. In this fashion, the MinScore and MaxScore fields only appear as separate fields in edit mode. In view mode and due to the "Z" modifier, the MinScore and MaxScore field values appear embedded in the keyword description. • **Profile Reference Fields:** This is supported in profile forms only at this time. In profile forms, the selection for a profile reference field to a child profile can now be embedded inline in form rather than as a lookup. In a profile form for a top level profile, one can configure a profile reference field to a child profile of the current top level profile. Similarly, in a profile form for a child profile, one can configure a profile reference field to a different child profile type that shares the same top level profile. In either case, the profile reference field by default appears with a lookup link that opens a model dialog box allowing the user to select a particular child profile. In some cases, it may be desirable to present the selection embedded inline in the form itself so that the selection is always visible, and this is what is new in 19.4. An example would be a service record that has a field named "Mandate" that references a different "Mandates" child profile. In this case, the directive {Mandate:Z"inline"} presents the selection inline in the form. If you would like to have control over the size of the inline selection in the form, use a format like {Mandate:Z"inline,800-400") to

	specify the width and height of the area. If the inline selection is dependent on other fields on the same form that may dynamically change, you can programmatically cause the inline selection to refresh using the `DataFormAPI.`invokeLookup method.
+"*wordforyes*" -"*wordforno*" _"*wordfornone*"	If used in conjunction with a logical field either configured to use a dropdown or to have separate checkboxes for "Yes" and "No", these modifiers allow you specify text to be used as labels in place of the default "Yes" or "No". Note that when a template is translated into another language, these labels extracted and translated within the directive. If the logical field allows empty values and is being displayed as a dropdown menu (with the D modifier), you can also customize the label for the EMPTY value using the underscore modifier. When there are separate checkboxes, the plus and and minus modifiers can be used (with or without the label strings) to have separate directives for the Yes and No checkboxes allowing for additional layout flexibility. In the directive for the Yes checkbox, include the plus modifier but not the minus modifier. In the directive for the No checkbox, include the minus modifier but not the plus modifier. These modifiers are also useful for calculated logical fields to control how they are presented. By default. such fields are presented a read-only checkbox. You can present it instead as dual checkboxes using {LogicalField:+"On"-"Off"} or as a simple text value using {LogicalField:D+"On"-"Off"}.
-"*wordfornone*"	Can also be used in conjunction with a keyword field dropdown menu to set the word used for the EMPTY value which is by default "none". Note that when a template is translated into another language, this word/string is extracted and translated within the directive.