ProForm

1.65 — Last update: 2016/08/03

MetaSushi LLC d.b.a RawaySmith

Table of Contents

ProForm User Guide

ProForm is an advanced drag and drop form management module for ExpressionEngine 2.0, designed to make creation and management of forms easier for developers and end users.
Features

ProForm has the following features:

- Forms fully configured in the Control Panel
- Drag & drop form layout in Control Panel
- Simple one line tag to render any form on your site
- Optional full ExpressionEngine template support for rendering multiple forms from a single custom template
- Multistep form support
- AJAX posting
- Mailing list opt-in
- CAPTCHA support to help prevent spam
- File uploads
- Send notifications, rendered using EE templates, to admins and/or any email address entered in the form
- CodeIgniter based validation including content filtering and encoding options (required, valid e-mail, strip HTML, base64 encode, etc.)
- Plentiful hooks, allowing third party customization
- Separate database table for each form – no more field count limits, easy to work with in custom code
- IP address and user agent recording
- Optional database Encryption
- Preset values for use in share forms such as Tell a Friend to prevent email spam

Installation

Installing ProForm is similar to installing any third-party Module into ExpressionEngine.

Installation Steps

Follow these steps to install ProForm:

- Download the ZIP file from the location where you purchased it.
- Extract the ZIP file.
- There are four folders within the extracted folder from the ZIP file. Copy each one to the matching location within your ExpressionEngine installation:
  /system/expressionengine/third_party/proform → /system/expressionengine/third_party/proform
  /system/expressionengine/third_party/prolib → /system/expressionengine/third_party/prolib
  /themes/third_party/proform → /themes/third_party/proform
  /themes/third_party/prolib → /themes/third_party/prolib
- Visit your control panel, usually located at:
  http://example.com/system/
- Click on Add-ons > Modules in the main menu.
- Within the list of Modules you will find an entry for ProForm, click on the Install link next to this entry.
- You will be taken to a Package Settings screen, listing entries for ProForm's components. Make sure that all components have Install selected.
- Click Submit to finish installation.
- After the module is installed, you can find a link to it's main page in the list at Add-ons > Modules.

On the main ProForm page, you may want to click the + Add link in the menu bar. This will add a main menu item for ProForm so you can get it to more easily.

You may now use the control panel page for ProForm to being Managing Forms.

Control Panel

ProForm allows full configuration of forms through the control panel interface.

- Form Management
- Field Management
- Module Settings
- Hidden Settings

Form Management

The main page of ProForm provides a list of all of the forms that have been created within the module.

Creating a New Form

To create a new form, use the Create a Form drop down menu to select one of the available types:

- Basic Form
- Share Form

When you create either type of form you will be asked to specify various settings settings as listed in the Form Settings section.

Basic Form

A Basic Form is used to create typical contact forms and other forms that save their data into the database. This data can be encrypted, and notification of new entries can be sent to various locations.

Share Form

A Share Form is meant to be used as a replacement to the Tell a Friend form. It does not save data to the database, but send the same types of notifications that can be sent for Basic Forms.

Form Settings

After creating a new form, the same settings that are set when it is initially created can be updated at any time by clicking the Edit Settings link on the form listing.

The following settings are available for the various types of forms.

These basic settings are available to all types of forms.

Basic Settings

Option	Description	Example Value
Full Form Name	Human friendly form name, used in UI and likely displayed on your website.	Contact Us
Form Short Name	Similar in concept to a “channel” name – used in templates to indicate what form’s information is to be retrieved in order to render the form, or to list it’s entries.	contact_us
Reply-To Address	Optional email address to use in the Reply-To header for all notifications sent from this form. If not provided, the global Module Setting of the same name will be used instead.	noreply@example.com
Reply-To Name	Optional name to use in the Reply-To header for all notifications sent from this form. If not provided, the global Module Setting of the same name will be used instead.	Example Co. Notifications

Notification List Settings

The Notification List is a preconfigured list of email addresses to send notifications to.

When a new entry is submitted to the form, each of the addresses in the Notification List will be sent a separate email message, generated from the selected template. The Notification List is available to all types of forms.

The Notification List has the following settings for each form.

Field	Description	Example Value
Enable Notification List	Turns the Notification List on or off. Notifications to the listed addresses will not be sent if this option is not checked.	n/a
Notification Template	ExpressionEngine Template used to render the emails sent to the Notification List. The available templates are listed from the Notification Template Group selected in Module Settings. For more information about what tags are available in a notification template, see Notification Templates.	n/a
Notification List	List of email addresses to send this notification to. This constitutes the actual Notification List itself. List each email address to send the notification to on it's own line.	admin@example.com support@example.com jdoe@example.com
Subject	The subject line of the notification email. This line is rendered as an ExpressionEngine template, and therefore can use conditionals and plugins in addition to the variables available in Notification Templates.	New submission to form: {form_name}
Reply-To Field	This option can be set to the name of a field, the value of which will be used as the notification's email address in the Reply-To header. This means that any recipient of the notification will automatically be able to reply to the email address entered into this field in the form. For instance, often times this option would be set to a field filled out by the visitor such as my_email_address. Whatever value is entered by the user would then be set as the Reply-To for the notifications. Assuming that the Notification List is sent to a customer service rep, or other person who would like to easily contact the submitter of the form, they can simply use their email client's Reply command to compose a new return message directly to the visitor.	submitter_email_address
Send Attachments?	This option enables sending of attachments to this notification group. All uploaded files from file fields on the form will be sent to this notification group as attachments.	n/a

Notification Field Settings

There are two groups of Notification Field Settings available for each form. Using a Notification Field allows the destination for your notifications to be entered by the visitor when they fill out the form.

These notification settings are useful in a situation where you want to provide a pre-set list of available contact addresses, and ask the user to pick one of these options. By specifying the field which contains these options as the Notification Field Setting's Email Field option, you will then get a notification sent only to that individual address.

You can also use a Notification Field setting group to send a notification a the visitor, thanking them for their submission.

This is different from the Notification List settings above, in that the entire list of addresses specified on the Notification List always receive the notification – while there is only ever one recipient of a notification sent from a Notification Field. Another difference is that the Notification List uses a static list of addresses defined on the form's settings, while Notification Fields take their options form a field instead.

Notification Field settings are available to all types of forms.

! Keep in mind that this feature provides what is essentially an open mail relay, which can allow anyone to send a message to any email address. It's very important that all forms that send notifications to addresses specified through a Notification Field have a CAPTCHA or other mechanisms in place to prevent the sending of spam.

Field	Description	Example Value
Enable Group	Turns the Notification Field on or off. Notifications to the addresses provided in the Email Field will not be sent if this option is not checked.	
Template	ExpressionEngine Template used to render the emails sent to the specified email address. The available templates are listed from the Notification Template Group selected in Module Settings. For more information about what tags are available in a notification template, see Notification Templates.	
Subject	The subject line of the notification email. This line is rendered as an ExpressionEngine template, and therefore can use conditionals and plugins in addition to the variables available in Notification Templates.	Thank you for submitting the {form_name} form
Email Field	The email address to send notifications to will be taken from this field in the form.	select_department
Reply-To Field	This option can be set to the name of a field, the value of which will be used as the notification's email address in the Reply-To header. This means that any recipient of the notification will automatically be able to reply to the email address entered into this field in the form. For instance, often times this option would be set to a field filled out by the visitor such as my_email_address. Whatever value is entered by the user would then be set as the Reply-To for the notifications. Assuming that the Notification Field is used to specify a particular	submitter_email_address

	customer service rep or department to send this notification to, that person can simply use their email client's Reply command to compose a new return message directly to the visitor.	
Send Attachments?	This option enables sending of attachments to this notification group. All uploaded files from file fields on the form will be sent to this notification group as attachments.	n/a

Advanced Settings

The Advanced Settings tab allows you to configure some additional values and extend the functionality of the form. These settings are dependent on which Form Drivers you have installed, so be sure to check the documentation for any additional Drivers to see if they provide additional Advanced Settings.

Some settings may be specific to the template you are using as well, meaning that they only have an effect if the template actually calls them.
All of these settings are available inside a {exp:proform:form} and {exp:proform:results} tags, as shown in the Variable column of this table. Some additional settings are available globally from the module's own Advanced Settings tab.

Name	**Variable**	**Description**	**Example Value**
Label for Extra 1 Label for Extra 2	{extra1_label} {extra2_label}	Changes the label visible under the Layout tab for the Extra 1 or 2 overrides. This can be useful to change the meaning of the Extra field in order to be more user friendly so the editors do not need to remember what you are using the Extra field for.	Wrapper HTML Class
Thank You Message	{thank_you_message}	Provides a custom message to be used in the default template and by the {exp:proform:simple} tag when the form has been submitted.	Thank you for submitting a Support Request! We will get back to you shortly.

Additionally there are some Hidden Settings available.

Form Layout

Forms have a user-defined layout that is created through the Form Layout tab of the form's settings.

You can get to this tab either by clicking it's name and then the Form Layout tab, or by clicking Form Layout directly from the from listing.

Layout Editing

The layout for a form is manipulated as a series of rows of fields. Each field can be moved between rows by clicking and dragging the field.

The Toolbox

All items which can be added to a form are accessible through the Toolbox, which lives under the Add an Item tab along the right side of the form edit screen's Layout page.

There are three types of items which can be added to the form directly from the Toolbox:

- New Fields – the first section contains a list of Field Types which can be created directly in the form
- Special Items – this type contains three subtypes:
- Headings – useful to break up the flow of a form into logical chunks
- HTML Blocks – used to insert any arbitrary HTML markup into the form
- Steps – used to create a multistep form with separately validated steps
- Library Fields – these fields have been predefined and possibly are in use in multiple forms

h3. Adding a Field

To add a new field to the form, click the type of the field you wish to create in the Toolbox under the Add an Item tab on the right side of the layout screen.

This will bring you directly to the New Field page, and return you to the form’s layout after adding the field to the form.

Field Overrides

Certain values can be overridden for each field on the form. To set a field’s overridden values, click on the field to select it. The Edit Field box on the right will then display the field’s current override values.

Field Override Values

Name	Template Var	Description
Field Label	{field_label}	Overrides the value returned for {field_label} for this field. Normally this value is taken from the field’s own settings. This allows the same field to have multiple labels depending on which form it is used on.
Field Default Value	{field_preset_value}	Allows entering the default value for this field.
Force Default Value	{field_preset_forced}	Removes the field from the list of fields returned by the {fields} variable pair, and forces the value to always be set to the Field Default Value. This prevents the visitor from changing the value of this field, even if they modify the POST form to change the data submitted.
Field Id	{field_html_id}	A value to be used as the field’s HTML id in the template.
Field Class	{field_html_class}	A value to be used as the field’s HTML class in the template.

Extra 1
Extra 2
	{field_extra1}
	{field_extra2}
		Extra meta values to be used by the template. These values can be used for any purpose desired, such as additional HTML classes or a custom field description for the form.

Show in Listing?	{show_in_listing}	Determines if the field will be shown in the entries listing for the form. Hiding fields can be helpful to make the entries listing more usable.
Show in Search?	{show_in_listing}	Determines if the field will be shown in the entries listing’s search box for the form.

Library Fields

Library Fields are created by marking a field as Reusable in its Field Settings.

After a field is marked as Reusable, it is placed into the Library section of the Toolbox. This allows you to easily and quickly add the field to other forms in the system simply by clicking it’s name.

To edit a field from the library, click it’s Edit… button. This allows you to modify the field’s settings, applying all of those changes to each form that uses the field from a single place.

Note that only fields that are not already assigned to the form will be listed in the Library.

Adding a Heading

Adding a heading to a form layout is very similar to adding a field. To add a new heading to a form, first click the form’s Edit Layout link from the form listing (or click the form’s Layout tab if you already have the form open). Then, scroll down in the Toolbox to the Special section and click the Heading button.

After adding a heading, it can be moved around the form layout similarly to a field. Simply click the heading to select it, then drag to move it to a new position. To edit a heading, click its Edit link after selecting it.

Each heading contains a single configuration value:

Name	Template Var	Description
Heading Label	{field_heading}	The value to use as the heading’s text. In the sample template and the markup generated by the Simple Form Tag, headings are added as <h3> HTML elements.

Adding a HTML Block

HTML Blocks allow you to add any kind of HTML to a form’s layout. They function almost exactly the same as Headings, but do not attempt to escape their output and provide a larger textarea for entering the markup.

To add a new HTML Block to a form, first click the form’s Edit Layout link from the form listing (or click the form’s Layout tab if you already have the form open). Then, scroll down in the Toolbox to the Special section and click the HTML Block button.

After adding a HTML Block, it can be moved around the form layout similarly to a field. Simply click the HTML Block to select it, then drag to move it to a new position. To edit a HTML Block, click its Edit link after selecting it.

Each HTML Block contains a single configuration value:

Name	Template Var	Description
HTML Content	{field_html_block}	The HTML content to inject into the form.

Multistep Forms

ProForm provides the capability to build multistep forms. Separating a form into multiple steps is extremely simple.

Adding a step to a form is similar to adding a heading or a field. To add a new step in a form, first click the form’s Edit Layout link from the form listing (or click the form’s Layout tab if you already have the form open). Then, scroll down in the Toolbox to the Special section and click the Step button. Enter a name for the step, then click Submit.

After adding a step, it can be moved around the form layout similarly to a field. Moving the step around the layout splits the form into different sized steps.

To move a step (and therefore move fields between them into different steps), simply click the step to select it, then drag to move it to a new position. To edit a step, click its Edit link after selecting it.

Step options:

Name	Template Var	Description
Step Label	{step}	The value to use as the step’s text. In the sample template and the markup generated by the Simple Form Tag, each step is represented as a tab.

Viewing and Exporting Entries

Entries for a form can be viewed by clicking the View Entries link of the form listing. The entries list for a form provides all of it's entries in a paginated list.
On the entries list for a form, you may export the entries from the form through the Export Entries button. The file will be downloaded in a CSV formatted text file.

Listing Configuration and View Entry

You can hide certain columns from the entries listing by turning off the Show in Listing? override for the field (see Overriding Field Settings). This can help make the view much more usable for more complex forms.

You may view all of the values recorded for an entry by clicking it's View link or it's ID in the entry listing. This will also show the full content of any longer text fields, which are truncated on the listing screen.

Deleting a Form

Deleting a form removes all of it's settings and deletes all of it's data from the database.

You can delete a form and all of it's data through the Delete command on the form listing. You must confirm the deletion in the following page.

> ! Warning Deleting a form cannot be undone. Old forms can be kept in the database indefinitely without any performance penalties.

Field Management

The Fields page in ProForm provides a list of all of the fields that are available to be used in forms.

Creating a New Field

There are two ways to create a new field. The first is to add a field from the Toolbox on an existing form's Layout screen. You can also use the Create a Field button on ProForm's global Fields page, accessible from ProForm's top navigation.

In either case, when creating a field you will be asked for the settings in the following section.

Field Settings

After creating a new field, the same settings that are set when it is initially created can be updated at any time by clicking either the Edit link for the field from a form's layout, or by clicking the field's name on the Fields page.

The following settings are available for fields.

Field Settings

Field	Description
Full Field Name (Label)	Human friendly field name, used in UI and used to label the field in the default template.
Field Name	Short field name which must be used as the actual name of the input or other HTML element in the form template.
Type	The type of the field. Most fields can be set as String fields. See the Field Types section below for more information.

Length	Length limit on the field. If the length is not specified, it will be set to 255 characters. This option is not required for many field types – only text, secure, and member_data require it.
Validation	Allows configuration of a large number of validation rules and filters to be applied to the value. This ensures that data entered matches expected patterns. See the Validation section below for more information on the types of rules available.
Placeholder	Optional HTML5 placeholder text to use for the field. Note that use of this depends on your implementation, if you use the `Full Form Tag`, and it is automatically used when you use the `Simple Form Tag`.
Upload Directory	See the **File** type in the next section.
Mailing List	See the **File** type in the next section.
Reusable	Places the field into the global `Library`, allowing you to add it to multiple forms. This is great for fields such as **First Name** which can be used on multiple forms and always have the same validation. The forms that this field is assigned to are displayed in the **Assigned Form** section under the Reusable checkbox.

! **Warning** If data has already been entered into any of the forms that use a field, changing the length of the field to be shorter than it was previously will likely cause data loss as any values that are too long will be truncated.

Field Types

ProForm supports multiple field types. Most fields can safely use the String type, however other field types can optimize the column chosen to store the data and work well with the default template code to suggest input types to correspond to the various field types. Additional types provide programmatic functionality such as subscribing to mailing lists.

Note that the Default HTML Control values provided here are based on the Sample Template, which is similar to the markup generated by the Simple Form Tag. If you don't like the elements that we generate by default, you can easily be customize your output by simply using your own template based on the Sample Template.

Field Types

Type	Default HTML Control	Description	Database Type
Checkbox	<input type="checkbox" … />	A simple boolean value stored as a y or n.	VARCHAR
Date	<input type="input" … />	A date field containing year, month, and day.	DATE
Date and Time	<input type="input" … />	A date field containing year, month, day as well as hours, minutes and seconds.	DATETIME
File	<input type="file" … />	A file upload field. **Type Options** When this type is selected, an additional option is available: **Upload Directory** – Directory to upload files into.	VARHCAR
Text	<input type="input" … /> or <textarea>	A simple text value. The length of the field determines which Database Type is used to store this field. If the length is 255 or less, a VARCHAR field is used, otherwise a TEXT field is used.	VARCHAR or TEXT
Textarea	<textarea>	A memo field, for longer form values such as messages.	TEXT
Number: Integer	<input type="input" … />	An integer numeric value.	INT
Number: Float	<input type="input" … />	A floating point numeric value.	FLOAT
Number: Currency	<input type="input" … />	A currency value with a fixed point.	DECIMAL
List	<select>, <input type="checkbox" … /> or <input type="radio" … />	A select box, set of checkboxes, or set of radio buttons allowing the visitor to choose between options. **Type Options** When this type is selected, these additional options are available: **Style** – The style of list you would like to use. • Select Box – Standard select HTML element, optionally with Multiselect support.	VARCHAR or TEXT

		• Check Boxes – A set of checkbox controls, one for each option. Inherently supports multiple selections.
		• Radio Buttons – A set of radio button controls, one for each option. Supports a single selection.
		
		Multiselect – Enables multiple selections (only shown when the style is Select Box).
		
		Options – A list of options available to the user. Uses either of the following formats for options, with one option on each line (these styles may be mixed within a single field if desired):
		
		`value1 : Label For Option 1`
		`value2`
		
		Option Groups
		To include groups of options, add a line with a value of "-" with it's label set to the name of the group. All styles of the List field type can render these option groups.
		
		For instance, the following option list would create two groups – one named "Billing Requests" and the other named "Sales Requests".
		
		`- : Billing Requests`
		`value1 : Label For Option 1`
		`value2`
		`value3 : Label For Option 3`
		`- : Sales Requests`
		`value4 : Label For Option 4`
		`value5 : Label For Option 5`	

Relationship	<select>	A select box allowing the visitor to choose amongst a set of Channel Entries. Additional options: **Channels** – Limits the available list of entries to the selected channels.	VARCHAR or TEXT

		Categories – Limits the available list of entries to the selected categories.	
Mailing List Subscription	<input type="checkbox" … />	When this checkbox is selected, the user will be subscribed to the mailing list selected. Because this uses ExpressionEngine's built in subscription system, the active user must be logged into a Member account for this to work properly. **Type Options** When this type is selected, an additional option is available: **Mailing List** – The list to subscribe the user to when this checkbox is set.	VARHCAR
Hidden	<input type="hidden" … />	A simple text field rendered as a hidden input field. This can be used to attach additional data to the form. This field type is always set to an internal length limit of 255.	VARCHAR
Member Data	<input type="hidden" … />	Stores a piece of member information associated with the logged in user along with the form fields. Fields with this type cannot be set through the field POST, but are always set on the backend to the active user's information. This field type is always set to an internal length limit of 255. When selected, an additional option is available: **Field** – Select the Member Field you wish to save into this form field on submission.	VARCHAR

Validation and Filtering

ProForm provides a robust set of validation rules and filtering operations. This system is based on the Form Validation Class provided by CodeIgniter.

Validation Rules

The following validation rules can be applied to a field. The **key** column specifies what key should be used with other features such as the message:*= param of the Full Form Tag.

Rule	Key	Description	Requires Param
Always Required	required	Marks the field as required on all forms that it is used on.	None
Matches Value	matches_value	Fails if the field's value does not match the value specified.	Value
Matches Field	matches	Fails if the field's value does not match the value of another field.	Field name
Min Length	min_length	Fails if the field's value is not of the required length.	Length
Max Length	max_length	Fails if the field's value is longer than the set length.	Length
Exact Length	exact_length	Fails if the field's value is shorter or longer than the set length.	Length
Alpha Characters Only	alpha	Fails if any characters not in the alphabet are used in the field's value.	None
Alpha Numeric Characters Only	alphanumeric	Fails if any characters not in the alphabet or the counting numbers (0-9) are used in the field's value.	None
Alpha Numeric Characters, Underscores and Dashes Only	alpha_dash	Fails if any characters not in the alphabet, the counting numbers (0-9), or underscores (_) or dashes (-) are used in the field's value.	None
Numeric Characters Only	numeric	Fails if any characters other than the counting numbers (0-9) are used in the field's value.	None
Numeric Characters and Dashes Only	numeric_dash	Fails if any characters other than the counting numbers (0-9), or dashes (-) are used in the field's value.	None
Integer Number	integer	Fails if the field's value is not an integer number.	None
Natural Number	is_natural	Fails if the field's value is not a natural number.	None
Natural Number other than zero	is_natural_no_zero	Fails if the field's value is not a natural number above or below zero.	None
Valid E-mail Address	valid_email	Fails if the field's value is not a properly formatted email address.	None
Valid E-mail Addresses separated by commas	valid_emails	Fails if the field's value is not list of one or more properly formatted email address separated by commas.	None
Valid IP Address	valid_ip	Fails if the field's value is not a properly formatted IPv4 dotted decimal address.	None
Valid Base 64 Encoded Value	valid_base64	Fails if the value is not a valid Base 64 value.	None

Filters

In addition to the validation rules, the following filters can be applied to the data before or after any validation to modify submitted values to match the expected values.

Rule	Description	Requires Param
Strip HTML (filter)	Removes all HTML tags from the value.	None
Trim (filter)	Removes any whitespace from the beginning and ending of the value.	None
Base 64 Encode (filter)	Encodes the value as Base 64.	None
Base 64 Decode (filter)	Decodes the value from Base 64.	None
URL Encode (filter)	Encodes the value according to URL specifications.	None
URL Decode (filter)	Decodes value from URL encoding.	None

Deleting a Field

Deleting a field removes all of it's settings, removes it from all forms that it is assigned to, deletes all of it's data from all forms in the database.

You can delete a field and all of it's data through the Delete command on the Fields page. You must confirm the deletion in the following page.

! **Warning** Deleting a field cannot be undone. Remember that **all** forms that have this field assigned will lose any data saved into it.

Module Settings

The following options are available to globally configure *ProForm*'s functionality.

Module Settings

Field	Description
Notification Template Group	Selects the template group which contains notification templates for your forms. This should be a template group dedicated only to email notification templates. Until a template group is created and selected here, notifications cannot be sent from form submissions.
From Address	Optional address to use in the notification email's From header. *
From Name	Optional name to use in the notification email's From header. *
Reply-To Address	Optional email address to use in the Reply-To header for all notifications sent by *ProForm*. Each form may also override this value in it's Form Settings.
Reply-To Name	Optional name to use in the Reply-To header for all notifications sent by *ProForm*. Each form may also override this value in it's Form Settings.

***** The From Address typically should match your ExpressionEngine email settings in order to avoid having your notifications marked as spam. If not provided, the value will default to that set in ExpressionEngine's global settings – so setting this option isn't recommended. Instead, use the Reply-To settings.

Advanced Settings

The Advanced Settings tab allows you to configure some additional values and extend the functionality of the module. Be sure to check the documentation for any additional Drivers to see if they provide additional Advanced Settings.

Some settings may be specific to the template you are using as well, meaning that they only have an effect if the template actually calls them.

All of these settings are available inside a {exp:proform:form} and {exp:proform:results} tags, as shown in the Variable column of this table. Some additional settings are available at the form level through the form's own Advanced Settings tab.

Name	Variable	Description	Example Value
Invalid Form Message	**{invalid_form_message}**	A message to display when the requested form isn’t found.	The requested form doesn’t exist. Please check the Form Directory for a list of available forms.
Thank You Message (Default)	**{thank_you_message}**	Provides a custom message to be used in the default template and by the {exp:proform:simple} tag when the form has been submitted. If a form provides it’s own **Thank You Message** through it’s Advanced Settings tab, that message will be used instead.	Thank you for submitting a form from the Form Directory. Your submission will be processed as soon as possible.

Additionally there are some Hidden Settings available.

Hidden Settings

The following settings are available for forms when certain options have been enabled in your global ExpressionEngine **config.php** file. This file is normally located at:

system/expressionengine/config/config.php

Certain options are only available in one Form Type or the other, as noted in the **Form Types** column.

! **Warning**

These options are considered experimental and should be used with caution. Always make sure to keep frequent backups and verify the behavior of any advanced settings you are using (for instance, if you use the options to store form data in an encrypted format – ensure you are able to decrypt data and move it to other systems you need to move it to).

There is **no support** for any options described here.

Advanced Option	Form Types	Description
Table Override	Basic Only	When set, this option redirects all data that would normally be stored in a database table generated by ProForm to a custom table you have already defined in the database. This option disables all manipulation of the table as well. Note that this option is still considered experimental. In order to use this option, you must enable it by editing your ExpressionEngine **config.php** file to include this line: $config['proform_allow_table_override'] = 'y';
Encrypt Data	Basic Only	Turns database encryption on or off for the current form. Note that this option cannot be changed after data has been entered into the form's entries. Encryption comes in two flavors – a simple and insecure XOR technique and a more secure method based on the Mcrypt extension, which must be installed. The default is the insecure XOR unless you have Mcrypt installed and have set a secure encryption key in your *config.php* file. Please refer to the CodeIgniter reference's warnings about their encryption class for more details and instructions for how to set your encryption key:

		http://codeigniter.com/user_guide/libraries/encryption.html (The config file is usually located at *system/expressionengine/config/config.php* instead of the location noted on that page.) Note that this option is still considered experimental. In order to use this option, you must enable it by editing your ExpressionEngine **config.php** file to include this line: $config['proform_allow_encrypted_form_data'] = 'y';

Tags

- Form Tags
- Results Tag

Form Tags

ProForm provides two variations of it's form tag. The goal of the two types of `Form Tags` are to allow a single template to render any form created through the ProForm drag and drop layout editor, with differing degrees of control over the output.

- {exp:proform:simple} – the Simple Form Tag – this tag allows you to render any form with one line of code
- {exp:proform:full} – the Full Form Tag – this tag allows you to customize the markup and behavior rendered forms

✻ **Crash Course** There are two methods you can use to render a form:

Method I. Simple Form Tag
Get a form running in less than two minutes! This option is very simple to use and offers either a pre-built set of CSS and JS, or the ability to use your own CSS styles and JS. Read the Simple Form Tag docs below for a two line example of how to style your forms exactly how you'd like without writing any HTML or template logic.

Method II. Simple Form Tag
This option allows you to control all of the HTML generated for the form. An excellent example template is provided to get you started. Read the Full Form Tag docs below to get started – look for the next **Crash Course** box in that section.

The single most important parameter is the form parameter, and the two most important variable pairs are the {fieldrows} and {fields} variable pairs.

Reading about this parameter and these variable pairs is the first step to creating or understanding a custom form layout template.

Simple Tag

The Simple Form Tag – {exp:proform:simple} – allows you to render a complete form built through the drag and drop interface with a single line of template code.

The only parmeter which is required for the Simple Form Tag is the form="" parameter, which must be set to the name of the form you wish to render. Be sure to see the Simple Form Tag Sample Templates section for examples of how to render your form with only one or two lines of template code.

- Parameters
- Sample Templates

Behind the Scenes – Customizing CSS and/or Js

The Simple Form Tag actually calls the Full Form Tag behind the scenes to generate its markup.

ProForm will never force any particular markup on you – you should have full control over the layout and implementation of your forms, if you so desire. At the same time, we provide the default markup generated by the simple tag in order to allow for getting started more quickly.

The simple tag also generates a default set of CSS and JS code to power the front-end of the form. You can disable this by using the {exp:proform:disable_head} tag before you call the {exp:proform:simple} tag. This allows you to use the built-in HTML but provide your own CSS and JS.

If you'd like to keep the default CSS and JS, but want to control where it is rendered, you can use the {exp:proform:head} tag to render the code explicitly where you would like it to go. If you want to put this "header" code into the footer of your site, use {exp:proform:disable_head} before calling the form tag in order to stop it from automatically rendering the CSS and JS, then include the {exp:proform:head} tag in your footer as you like.

Of course the most flexible method is to use the Full Form Tag to get complete control over 100% of the form markup, styling and scripting.

Simple Parameters

The Simple Form Tag supports all of the same parameters as the Full Form Tag **except** the variable_prefix and last_step_summary parameters since they do not make sense for a simple form.

See the full list of parameters for more information.

The simple tag also supports the following parameters:

- template="default"

template="default"

The **template** parameter allows you specify which internal template you would like ProForm to render for the tag.

These internal templates behave much like embed templates, but are stored directly under ProForm's package directory at this location:

system/expressionengine/third_party/proform/templates

If you need to use a custom template, it's recommended that you either use a true embed template or make a copy of the **default.html** internal template and customize it.

Example Usage

Assuming you've created a custom template file named **system/expressionengine/third_party/proform/templates/custom_template.html**:

{exp:proform:simple form="contact_us" template="custom_template"}

Sample Template Code

Using the Simple Form Tag couldn't be easier:

Example Usage

{exp:proform:simple form="contact_us"}

That's it! You're done.

If you'd like to control the CSS and JS of the form, you can include the **{exp:proform:disable_head}** tag and style the resulting markup however you like:

Disable Head Usage

The next code block shows how to disable the head output.

```
{exp:proform:disable_head} {!-- turn off the header code --}
<!-- include our custom form css and js -->
<link rel="stylesheet" media="screen, projection" href="{site_url}assets/css/
custom_form.css" />
<script type="text/javascript" src="{site_url}assets/js/custom_form.js"></script>
{exp:proform:simple form="contact_us"}
```

If you'd like even more control over the markup of the template, you can make use of the Full Form Tag.

Full Form Tag

The Full Form Tag – {exp:proform:form} – provides you with precise tools to control the exact HTML produced for your forms built through ProForm. Like the Simple Tag, it is capable of rendering any form created through the drag and drop interface.

Crash Course
The single most important parameter is the form parameter, and the two most important variable pairs are the {fieldrows} and {fields} variable pairs.
Reading about this parameter and these variable pairs is the first step to creating or understanding a custom form layout template.

- Parameters
- Variables
- Variable Pairs
- Sample Template

h2. Full Parameters

The following parameters are available. All are optional, with the exception of the form="“ parameter, which must be set to the name of the form you wish to render.

- Custom Params
- debug=“yes”
- dashes_in_class=“yes”
- dashes_in_id=“yes”
- error_delimiters=”<p>|</p>”
- error_url=“forms/contact-us/error”
- form=“contact_us”
- form_class=“form_layout_wide”
- form_id=“form_contact_us”
- form_url=“forms/contact-us”
- hidden_fields=“split”
- last_step_summary=“yes”
- message:required=“This field is required!”, message:*="“
- notify=“sample@example.com”
- set:*="“
- step=“1”
- site=“default_site”
- variable_prefix=“pf_”
- thank_you_url=“forms/thank-you”

Custom Params

Any parameters sent to the `Form Tag` which it does not know how to handle will be packed into the form session and sent along with the rest of the form’s data when it is submitted. These values can then be extracted on the **thank_you_url** or **error_url** pages with the Results Tag.

Because the form session is stored in the database temporarily and only referred to by an opaque hash value in the form’s submission, a user cannot discover the contents of parameters sent in this way to the **thank_you_url** or **error_url** pages.

debug=“yes”

The **debug** parameter turns on debug mode for ProForm, which will provide some additional information in order to assist in tracing it’s behavior.

dashes_in_class=“yes”

The **dashes_in_class** parameter causes the automatic form element class generated to use dashes instead of underscores.
For instance, this would change the class generated form a form named “contact_us” from “contact_us_proform” to “contact-us-proform”.
You can also completely override the class using the form_class paramter.

dashes_in_id=“yes”

The **dashes_in_id** parameter causes the automatic form element ID generated to use dashes instead of underscores.
For instance, this would change the ID generated form a form named “contact_us” from “contact_us_proform” to “contact-us-proform”.

You can also completely override the ID using the form_id paramter.

error_delimiters=”<p>|</p>”

The **error_delimiters** parameter allows you to specify custom tags to wrap around each error message generated for a field. This parameter requires two values – which must be separated by a pipe character as shown. The left side of the pipe is used to start the error block, and the right side is used to end it. The default values are:

Default error_delimiters

error_delimiters='<div class="error">|</div>'

error_url="forms/contact-us/error"

The **error_url** parameter allows you to override what URL the form redirected to when there is an error detected in the form's validation.

The default value is the current URL. Normally this should not be changed, as the `Form Tag` is not only used to display the form initially, but also used to display the form with error messages attached.

form="contact_us"

The **form** parameter is used to specify which form's information should be loaded. Since ProForm doesn't know which form you want to render, you need to specify using this parameter.

This value can be taken from a URL segment, or from an entry field. The form name can also be hard coded, although in general I recommend creating a more generic template that can be reused for multiple forms.
For technical reasons, form names always use underscores rather than dashes to separate words. However, if you specify the form name using dashes, it will be converted automatically for you. For instance, requesting a form named "contact-us" would load the form named "contact_us" automatically. You can get the name of the current form with dashes instead of underscores using the variable {form_name:dashes} within most tags in ProForm.

Crash Course

Next Step: After getting the correct form by using the **form** parameter as in one of the examples below, you'll want to look at it's structure using the {fieldrows} and {fields} variable pairs in order to recreate it's structure.

Example Usage

Assuming the URL for all forms on a site is **http://example.com/site/forms/form-name**, such as **http://example.com/site/forms/request-quote**, the **forms** template in the **site** template group could load the requested form in {segment_3} like shown in the next code block.

```
{exp:proform:simple form="{segment_3}"}
```

See the {if no_results} conditional for a way to know if the requested form name was valid or not.

This approach allows you to create a single URL that can serve all templates – such as /site/forms/contact-us.
Using this tag within an embed template, where you pass in the argument through an embed parameter instead of directly accessing a segment variable is even more powerful and allows you to reuse your custom display logic anywhere on the site for multiple embedded forms:

Example Usage

Assuming you have a **.form** template within a **global** template group that looked like this, with all your custom display logic:

```
{exp:proform:form form="{embed:form}"}
... custom display logic here ...
{/exp:proform:form}
```

Using it is quite simple:

```
{embed="gobal/.form" form="newsletter-subscribe"}
```

form_class="form_layout_wide"

The similarly to the **form_id** parameter, the **form_class** parameter allows you to override the default form class attached to the generated <form> tag.

The default form class takes the following form:

Default form_class

```
form_class="{form_name}_profom"
```

form_id="form_contact_us"

The **form_id** parameter allows you to override the default form ID generated in the resulting <form> tag. The default form ID takes the following form:

Default form_id

```
form_class="{form_name}_profom"
```

form_url="forms/contact-us"

The **form_url** parameter allows you to override what URL the form is set to post to.

The default value is the current URL. Normally this should not be changed, as the `Form Tag` handles the form submission when it sees the form's POST data from the browser.

hidden_fields="split"

The **hidden_fields** parameter allows you to switch behavior for hidden field types between two modes:

Mode	Description
split	The new default behavior – all hidden fields will be available only through the {hidden_fields} variable pair. Hidden fields will not be listed in either the {fieldset} or {fields} variable pairs.
hybrid	The old behavior – all hidden fields will be included in the {fieldset} and {fields} variable pairs. Their position is not predictable (which is why the new default mode was added), based on the order they were added to the form. Use of this mode is not recommended but may be required for older templates

	which rely on it. Even in hybrid mode, the new {hidden_fields} variable pair is still available to use if you wish.

message:required="This field is required!", message:*=""

The **message:** parameter allows you to specify a custom error message to be displayed for each validation rule. The * should be replaced with a *key** corresponding to the validation rule you wish to set the message for. You may use the *message:** parameter many times.

Within each error string, the sequence **%s** will be replaced with the label for the field which has the error. See the list of Validation Rules for a list of what keys to use for each validation rule.

Additionally, a special validation rule which applies only to List type fields and prevents them from accepting values which are not in their options can be set with the parameter **message:list_choice_invalid=""**.

Example Usage

```
{exp:proform:form form="contact_us" message:required="Please enter a value for
the field %s!"}
...
{/exp:proform:form}
```

notify="sample@example.com"

The **notify** parameter allows you to add additional email addresses to be sent the notification as configured on the form's Notification List settings. Note that this requires that the Notification List settings be properly configured in order to operate.

Example Usage
Multiple email addresses may be separated by a pipe character:

```
{exp:proform:form form="contact_us"
notify="sample1@example.com|sample2@example.com"}
...
{/exp:proform:form}
```

✻ **But there is a better way!** Based on other form systems you may have used in ExpressionEngine, your first tendency may be to hard-code email addresses into your form templates using this parameter, but I strongly advise that you attempt to make use of the Notification List by itself and place all email addresses to send notifications to in the form's Notification List Settings instead.

set:*=""

The **set** parameter allows you to specify a value to be saved for any field on the form. This can be used to attach information about a submission, such as custom user fields, order information, or whatever else you wish to save into the form.

The * should be replaced with a valid field name assigned to the form – often this will be the name of a hidden field. For instance, to set the value for a field named "user_address" we would use the parameter **set:user_address="100 Test Street"**. The set parameter can occur as many times as needed within a form tag, provided each field name used is unique.

✻ **Example Usage** Suppose we have an order ID from a custom ordering system that we need to save with a form submission. This ID is in the variable {order_id}. To save this into a ProForm entry, we can simply set the value for a field named "external_order_id" with the set parameter like so:

```
{exp:proform:form form="order_record" set:external_order_id="{order_id}}
...
{/exp:proform:form}
```

You may also use the {exp:proform:set} tag to do the same thing as the **set** parameter. This will allow you to store longer strings of text if needed:

✻ **Example Usage**

```
{exp:proform:set form="order_record"
field_name="external_order_id"}{order_id}{/exp:proform:set}
{exp:proform:form form="order_record"}
...
{/exp:proform:form}
```

step="1"

The **step** parameter allows you to explicitly request a particular step be loaded for the form.

By default, ProForm manages the active step using a session and hidden values passed inside the form. This parameter allows you to override this default behavior and specify the step to load based on your own criteria.

site="default_site"

The **site** param allows you to override the active site that ProForm looks for forms within. This allows you to make sure of forms from other sites within other sites in your MSM install.

last_step_summary="yes"

The **last_step_summary** parameter turns on a special mode for the last step of a multistep form, where all fields are returned by the {fields} loop. This can be used to present a summary of the form about to be submitted before the user actually submits it, giving them a chance to go back and change their responses.

Note that in order for this to work correctly, you should create an empty step at the end of the form without any fields inside of it. Otherwise the form will be expecting those fields to be set, and if they are required, it will not process the form submission.

This parameter is not supported by the `Simple Form Tag` since you cannot provide a custom loop for that tag and therefore it is not needed.

Example Usage

```
{exp:proform:form form="support_request" last_step_summary="yes"}
    <!-- Step tabs, hidden fields, etc. here -->
    {if on_last_step}
        <h3>Summary</h3>
        <p>Be sure to click *Submit* to send the request.</p>
        <p>If you need to make revisions, use the Previous button or the tabs 
above.</p>
        <hr/>
        {fields}
            {field_label}
                <p>*{field_label}:* {field_value}</p>
            {/field_label}
        {/fields}
        <hr/>
    {if:else}
        {fieldrows}
            {fields}
                <!-- Render fields normally based on their type (see the full 
example template) -->
            {/fields}
        {/fieldrows}
    {/if}
    <!-- Previous and Submit buttons and CAPTCHA here -->
{/exp:proform:form}
```

variable_prefix="pf_"

The **variable_prefix** parameter is used to specify a prefix to add to all variables provided by ProForm. This parameter is not supported by the `Simple Form Tag` since you cannot provide a custom loop for that tag and therefore it is not needed.

Example Usage

```
{exp:proform:form form="contact_us" variable_prefix="pf_"}
...
{/exp:proform:form}
```

For example, the following variables are renamed as shown when a prefix of **pf_** is used as in the example above:

Normal Name	Name With pf_ Prefix
{form_name}	{pf_form_name}
{form_type}	{pf_form_type}
{complete}	{pf_complete}
{errors}	{pf_errors}
{fields}	{pf_fields}
{field_value}	{pf_field_value}

thank_you_url="forms/thank-you"

The **thank_you_url** parameter allows you to override what URL the form redirected to after it's data has been saved to the database.

Typically, you will want to create a single thank you template to thank visitors for submitting all forms. This template should make use of the Results Tag to retrieve information about the particular form submitted.

Single Variables

The following variables are available within the `Form Tag`. Each of these variables can also be used as a conditional.

- {captcha}, {use_captcha}
- {checked:*}
- {complete}
- {error_count}
- {error:*}
- {form_id}
- {form_label}
- {form_name}
- {form_name:dashes}

- {form_type}
- {fields_count}
- {label:*}
- {on_first_step}
- {on_last_step}
- {step_count}
- {multistep}
- {value:*}

The following special conditional variables are also available as well:

- {if no_results}

{captcha}, {use_captcha}

The {captcha} variable is set to an <img /> tag, which displays a CAPTCHA image automatically. The characters in this image are checked against an input element you should create, also named **captcha**.

The {use_captcha} variable is set to true or false, depending on if a CAPTCHA is needed for the current user. Logged in members are never required to submit a CAPTCHA. If a visitor is not currently logged in, they will be required to enter a CAPTCHA.

Note that CAPTCHA validation is only activated if you actually reference the {captcha} variable in your template – if you do not, the CAPTCHA will not be checked for – allowing all visitors to submit forms.

Conditionals
The conditional {if captcha} will **NOT** work. You must instead use the {if use_captcha} conditional to check to see if a CAPTCHA is required.

Example Usage

```
{exp:proform:form form="contact_us"}
    {if use_captcha}
        <p>Enter this word: {captcha}</p>
        <p><input type="text" name="captcha" /></p>
        {if error:captcha}
            Please try again with the new word!
        {/if}
    {/if}
{/exp:proform:form}
```

{checked:*}

Provides a string which can used to recheck a checkbox when the form is reloaded. These variables are set when the form contains an error and needs to be redisplayed, **or** when a form step is reloaded (a step can be visited multiple times by the user).

The value will either be the string **checked="checked"**, or a blank string. Thus, it can be injected directly into a field in order to allow it to be rechecked, or it can be used in a conditional (since anything but a blank string and zero is considered "true").

Replace the * in the variable name with the name of a field.

The {field_checked} variable inside of a {fields} variable pair provides the same value for fields, and is easier to use when generating generic form markup.

Example Usage
Assuming there is one field defined on the form, named **agree_to_terms**.

```
{exp:proform:form form="contact_us" variable_prefix="pf_"}
    <p>Old Value for field: {checked:agree_to_terms}</p>
    {fields}
        <p><input type="checkbox" name="{agree_to_terms}" value="y"
{field_checked} /></p>
    {/fields}
{/exp:proform:form}
```

{complete}

If the form has been successfully processed and the thank_you_url was set to the same URL as the form itself (the default), this value will be set to be true so that you may display a thank you message instead of or in addition to the blank form.

{error_count}

Provides the total number of fields that contain errors on the form. If the form has just been loaded for the first time and there are no errors, this will be set to 0. A conditional on this variable is the easiest way to determine if the template is loading an initial form view or loading a failed attempt to submit the form.

{error:*}

Provides any errors that might be set for a particular field. Replace the * in the variable name with the name of a field.

This is most useful with the {captcha} variable in order to determine if a CAPTCHA error has occured. See that variable's Example Usage for an example.

{form_id}

Provides the unique numeric ID of the form.

{form_label}

Provides the friendly form label for the form. This can be any human readable string.

{form_name}

Provides the internal form name for the form. This should be all lowercase and only contained letters, numbers, and underscores.

{form_name:dashes}

Provides the internal form name for the form with all underscores changes to dashes. This can be used in URLs and other places where a version of the form name with dashes is desired. Using dashes in a form_name="“ parameter will still work correctly as it automatically changes all dashes encountered into underscores before doing the form lookup.

{form_type}

Provides the type of the form being rendered. This is either set “form” for Basic Forms or “share” for Share Forms. For more about the types of forms available and their behavior, see the Creating a New Form section.

{fields_count}

Provides the total number of fields assigned to the form.

{label:*}

Provides the label set for a particular field on the form. These variables are most useful for cases where you are not using the full {fields} loop and wish to just render the information for a small set of fields. See also the {value:*} variables.
Replace the * in the variable name with the name of a field to get it’s label.

The {field_value} variable inside of a {fields} variable pair provides the same value for fields, and is easier to use when generating generic form markup.

Example Usage
Assuming there is one field defined on the form, named **phone_number**.

```
{exp:proform:form form="contact_us" variable_prefix="pf_"}
    <p>The label for the phone_number field is: {label:phone_number}</p>
{/exp:proform:form}
```

{on_first_step}

Provides a boolean value indicating if the form is currently on it’s **first** step.

{on_last_step}

Provides a boolean value indicating if the form is currently on it’s **last** step.

{step_count}

Provides a count of the total number of steps in the form. All forms always have at least one step – this value can never be less than “1”.

{multistep}

Provides a boolean value indicating if the form has multiple steps or not.

{value:*}

Provides the value entered for a field on the form. These variables are set when the form contains an error and needs to be redisplayed, **or** when a form step is reloaded (a step can be visited multiple times by the user). These variables are most useful for cases where you are not using the full {fields} loop and wish to just render the information for a small set of fields. See also the {field:*} variables.

Replace the * in the variable name with the name of a field.

The {field_value} variable inside of a {fields} variable pair provides the same value for fields, and is easier to use when generating generic form markup.

Example Usage
Assuming there is one field defined on the form, named **first_name**.

```
{exp:proform:form form="contact_us" variable_prefix="pf_"}
    <p>Old Value for field: {value:first_name}</p>
    {fields}
        <p><input type="text" name="{field_name}" value="{field_value}" /></p>
    {/fields}
{/exp:proform:form}
```

Conditional Variables

Conditional variables can only be used in conditional statements. In `ProForm`, all normal variables are also conditional variables.

The following special conditional variables are available:

- {if no_results}

{if no_results}

The {if no_results} conditional works exactly as it does for the `Channel Entries` tag and other module tags: if the form requested is not found, this conditional's contents are returned instead of the rest of the template.

You cannot use a {if:else} clause with this conditional – it is parsed differently than normal conditionals to make it easier to insert without having to nest the rest of your template within a {if:else} clause.

Example Usage
As in the example for the form parameter, assuming the **forms** template in the **site** template group is setup to load any requested form, and we want to display an error message when the requested form isn't found, we could do this:

```
{exp:proform:form form="{segment_3}"}
    {if no_results}
    The requested form could not be found!
    {/if}
    {!-- if the form is found, we'll get to here and continue processing... --}
    {fieldrows}
        <p>
        {fields}
            <label for="{field_name}">{field_label}</label>
            <input type="{field_type}" name="{field_name}" value="{field_value}"
placeholder="{field_placeholder}" />
        {/fields}
        </p>
    {/fieldrows}
{/exp:proform:form}
```

Variable Pairs

The following variable pairs are available within the `Form Tag`:

- {errors}
- {fields}

- {fieldrows}
- {hidden_fields}
- {steps}

{errors}

The **errors** variable pair provides a list of any errors that were detected when submitting the form.

Example Usage
This example also demonstrates the {error_count} variable being used in a conditional to prevent rendering a wrapper for the errors when there are none.

```
{if error_count}
<div class="errors">
    <p>The form has the following errors:</p>
    <ul>
    {errors}
        <li>{error}</li>
    {/errors}
    </ul>
</div>
{/if}
```

{fields}

The **fields** variable pair provides a list of all of the fields in the selected form, regardless of the row they are set on.

Crash Course

Be sure to read the documentation for the {fieldrows} variable pair in order to see how to preserve the drag & drop layout for a form in your custom markup, including simple examples of the {fields} loop itself.

Also see the full Sample Template for a more complete example.

{fields} Variables

The following variables are available within each row of the {fields} variable pair.

- {field_id}
- {field_name}
- {field_label}
- {field_heading}
- {field_html_block}
- {field_type}
- {field_length}
- {field_is_required}
- {field_is_step}
- {field_error}
- {field_checked}
- {field_control}
- {field_preset_value}
- {field_html_id}
- {field_html_class}
- {field_extra_1}
- {field_extra_2}
- {field_validation:*}
- {field_value} – see also {field_values}
- {field_filename}
- {field_basename}
- {field_ext}
- {field_value_label}

{fields} Variable Pairs

The following variable pairs are available inside the {fields} pair.

- Field Type Pair
- {field_options} (formerly {field_setting_list})
- {field_values}
- {field_validation}

The following variable pairs are available within each row of the {fields} variable pair, but contain values only for List and Relationship type fields.

- {dropdown_style}
- {check_style}
- {radio_style}

{fields} Single Variables

{field_id}

Provides the unique ID of the field.

{field_name}

Provides the short alpha numeric name of the field. This should be used as the input or textarea element's name parameter in order to have ProForm process the field's value correctly.

{field_label}

A human friendly label for the field, typically contained within a label tag attached to the input element.

{field_heading}

For items that are actually Headings within a form, will be set to the heading value. If this value is blank, the item is either a normal field or a HTML block.

Headings are also rendered for steps beyond the first step. You can tell the difference by using the **{field_is_step}** variable, which will be set to “step” if the heading is from a step, or blank otherwise.

{field_html_block}

For items that are actually HTML blocks within a form, will be set to the block of HTML. If this value is blank, the item is either a normal field or a Heading.

{field_type}

The internal type name of the field.

ProForm 1.65

Previous versions of ProForm relied heavily on the {field_type} variable in custom templates to determine how to render each field based on it’s type. In ProForm 1.65+, each {field_type} value generates a custom tag pair which should be used instead of normal {if} conditionals. For instance, if you had previous written this line to check for checkboxes:

{if field_type == “checkbox”}…{/if}

You should change this to:

{checkbox}…{/checkbox}

In most cases, conditionals on the {field_type} variable should continue to function, however the custom tag pairs are **much** faster to render and should improve your form’s performance. All example code will now be provided in the new style.

{field_length}

The constraint set for this field. Defaults to 255 characters for all field types.

{field_is_required}

A flag indicating if this field is required or not. A conditional on this value will allow you to flag the field with a character or other visual indication that the field is required.

{field_is_step}

A flag indicating if this Heading is from a step or not. Headings are rendered for each step beyond the first step. This value will be set to “step” if the heading is from a step, or blank otherwise.

{field_error}

Presents a string value of an error for this field. If there is no error detected in the field, or if it is the initial form load, this value will be blank.

{field_checked}

A flag indicating if the field is checked. Use this to preserve values when errors have occurred in other fields.

{field_control}

The suggested input type to use to render this field. This will be set to one of the valid HTML input type values (“text” or “file”), or “textarea”.

{field_preset_value}

A preset value for this field as set in the Default Value field of the form's layout editor.

{field_html_id}

A HTML ID value for this field as set in the form's layout editor.

{field_html_class}

A HTML class value for this field as set in the form's layout editor.

{field_extra_1}

An extra meta value for this field as set in the form's layout editor.

{field_extra_2}

An extra meta value for this field as set in the form's layout editor.

{field_setting_multiselect}

Only valid for 'select' type fields. This field is set to "y" if the select is a multiselect, or a blank value otherwise.

{field_validation:*}

Provides a boolean value for each validation rule key, indicating if that validation rule is applied to this field or not. Replace the * with the validation rule you would like to check.

For a list of valid keys, see the complete list of Validation Rules.

Example Usage

This example checks to see which fields are required, for those that are, it will add a ** Required!* message next to the label.

```
{fields}
    <label>
        {field_label}
        {if field_validation:required}* Required!{/if}
    </label>
    <input name="{field_name}" />
{/fields}
```

{field_value}

The value of the field as it was submitted. Use this to preserve values when errors have occurred in other fields.

For File type fields, this contains the full URL to the uploaded file.

{field_filename}

Only valid for File type fields. The value of the filename saved in this field. Note that the filename will be slightly modified from what was originally uploaded for safety and to make sure it is unique.

{field_basename}

Only valid for File type fields. The value of the filename, without any extension.

{field_ext}

Only valid for File type fields. The extension (if present) for the uploaded file.

{field_value_label}

Only valid for List type fields. Provides the label of the selected option. This is useful for notification templates primarily, where you may want to display the label of the selected value rather than the actual value stored in the DB.

{fields} Variable Pairs

Field Type Pair

ProForm provides one variable pair for each type of field available. You use these pairs to provide for different rendering of each field based on its type.

For instance, you might use the following tag pairs to differentiate between text type fields and date type fields:

Example Usage

```
{fields}
    {text}
        &lt;textarea name="{field_name}" id="{field_name}"
            class="custom_class_here
{field_is_required}">{field_value}</textarea>
    {/text}
    {date}
        &lt;input type="text" name="{field_name}" id="{field_name}"
            class="date {field_is_required}" value="{field_value}"
            placeholder="{field_placeholder}" /><!-- date -->
    {/date}
{/fields}
```

{field_options}

Only valid for List and Relationship type fields. The {field_options} loop, which must be used inside of the {fields} pair, provides a list of all of the options available for fields with the type **list**.

Note

This variable was previously named {field_setting_list}.

For a Relationship type field, this is a dynamically generated list of the available options based on the configured Channels and Categories.
This variable pair has these variables available for each option:

- **{key}** – the value as stored in the database
- **{label}** – the label of the value shown to the user (formerly {row})
- **{selected}** – indicates if the value is currently selected
- **{number}** – the numeric position of the option, this value will change if options are re-ordered
- **{is_divider}** – indicates this item is a divider – see Option Groups section of the List field type options

- **{divider_number}** – numeric position of the divider, again this value will change if options are re-ordered

Example Usage

```
{fields}
{list}
  <select id="{field_name}" name="{field_name}">
    {field_options}
    <option value="{key}">{label}</option>
    {/field_options}
  </select>
{/list}
{/fields}
```

{field_values}

List type fields, this contains all selected values. It has the same variables available inside it as {field_options}.

{field_validation}

The {field_validation} loop, which must be used inside of the {fields} pair, provides a list of the validation rules attached to the current field.

This variable pair has two variables available for each validation rule applied to the field:

- **{rule_no}** – a counter starting at 1 for each rule applied (this is a temporary value which may change if the field is edited)
- **{rule}** – the rule key as defined in the list of Validation Rules

Example Usage

This example puts every validation key attached to a field into the field’s class=”“ HTML attribute.

```
{fields}
    <label>{field_label}</label>
    <input name="{field_name}" class="{field_validation}{rule}
{/field_validation}" />
{/fields}
```

List Style Variable Pairs

These pairs are available only form List and Relationship fields.

{dropdown_style}

Loops over all of the options for the field. This pair contains values only if the field’s style is set to “Select Box”. Provides two variables for each option:

- {key}
- {value}

{check_style}

Loops over all of the options for the field. This pair contains values only if the field’s style is set to “Checkboxes”. Provides two variables for each option:

- {key}
- {value}

{radio_style}

Loops over all of the options for the field. This pair contains values only if the field’s style is set to “Radio Buttons”. Provides two variables for each option:

- {key}
- {value}

{fieldrows}

The **fieldrows** variable pair provides a list of the selected form's field rows. Each of these rows represents one of the visible rows in the control panel view of the layout for a form.

Crash Course

Next Step: Within each row there is a nested {fields} list that provides the fields that are placed in each row. See the {fields} documentation for more information on the data available inside the nested fields pair. Take a look at the examples below for how this all fits together.

To preserve the view of the form as presented in the layout editor, you should create a new strip of elements of some kind for each row in the {fieldrows} loop. This would typically by a <ul> element with <li> elements floated to the left, a table <tr> with a <td> element for each field, or a number of other possibilities – the key is just that there should be a visual arrangement of the fields in a horizontal layout for each row.

Here are two examples of how this structure can be used to recreate the layout of the form. Also see the Sample Template for a more complete example incorporating this.

Example Usage
A simple layout using CSS and lists.

```
<style>
    /* Each row consists of an independent UL. Within this we will float each LI
```

```
to the left. */
    .form-wrap li {
        float: left;
    }
</style>
<div class="form-wrap">
{exp:proform:form form="{segment_3}"}
    {fieldrows}
        <ul> {!-- create a new UL for each row of the form --}
        {fields}
            <li> {!-- create a new LI for each field on a row --}
                <label for="{field_name}">{field_label}</label>
                <input type="{field_type}" name="{field_name}"
value="{field_value}"
                    placeholder="{field_placeholder}" />
            </li>
        {/fields}
        </ul>
    {/fieldrows}
{/exp:proform:form}
</div>
```

Example Usage
Using the dreaded tables for layout purposes…

```
{exp:proform:form form="{segment_3}"}
    <table> {!-- the exp:proform:form tag generates a <form> element, so this
must be inside it --}
    {fieldrows}
        <tr> {!-- create a new table row for each row of the form --}
        {fields}
            <td> {!-- create a new table cell for each field on a row --}
                <label for="{field_name}">{field_label}</label>
                <input type="{field_type}" name="{field_name}"
value="{field_value}"
                    placeholder="{field_placeholder}" />
            </td>
        {/fields}
        </tr>
    {/fieldrows}
    </table>
{/exp:proform:form}
```

{hidden_fields}

The **hidden_fields** variable pair provides a list of all hidden fields within the form. It otherwise functions almost identically to the normal {fields} loop. See the list of variables inside the {fields} loop above for more info on what is available inside this loop.

{steps}

The **step** variable pair provides a list all of the steps configured for the form. There is always at least one step – which will by default have the same name as the form itself. Additional steps can be added easily in the Form Layout.

Example Usage

```
{if multistep}
    <ul class="pf_steps">
        {pf_steps}
            <li><a href="#{pf_step_no}" class="pf_step
{pf_step_active}">{pf_step}</a></li>
        {/pf_steps}
    </ul>
{/if}
```

{steps} Variables

The following variables are available within each row of the {steps} variable pair.

- {step}
- {step_active}
- {step_no}

{steps} Single Variables

{step}

Provides the text for the step’s name as defined in the layout for the form.

{step_active}

Provides a string value that indicates if this step is active. If the step is active, this variable returns the word **active**, if not it returns a blank string. This may be used in a conditional (since anything but a blank string and zero is considered “true”), as well as directly in a class=”“ parameter for an HTML element, as in the example above.

{step_no}

Provides the number of the step. Steps use 1-based numbering: the first step is number 1, the second is number 2, etc.

Sample Template Code

See the full Sample Template for the `Form Tag`.

For an example of using AJAX with ProForm, see the AJAX Sample Template.

Results Tag

The Results Tag allows display of a thank you message along with the submitted data from a form.

Parameters

The Results Tag takes all of it's data from the visitor's session.

This tag should be used in the template that handles the thank_you_url path sent to the Form Tag.

It does however accept one parameter:

- debug="yes"

debug="yes"

The **debug** parameter turns on debug mode for ProForm, which will provide some additional information in order to assist in tracing it's behavior.

Single Variables

See the single variables of the Entries Tag tag for more information.

Additionally, any unknown custom parameters sent to the form tag will be available as single variables within the Results Tag.

Variable Pairs

See the variable pairs of the Entries Tag tag for more information.

Sample Template Code

Example Template

This tag should be used in the template specified by the URL passed in through a form tag's thank_you_url parameter.

```
{exp:proform:results}
    {if form_name}
        <h1>Thank you for submitting the <a
href="{path=forms/{form_name}}">{form_name}</a> form</h1>
        <p>We have received the following information which you submitted:</p>
        {fields}
            {field_name} = {field_value}<br/>
        {/fields}
        <p>Custom thank you message from the form:</p>
        <p>{thank_you_message}</p>
        {if referrer_url}
            <p>Return to <a href="{referrer_url}">previous page</a>.</p>
        {/if}
    {if:else}
        <h1>Error</h1>
        <p>Something went wrong, and we are unable to confirm your
submission.</p>
    {/if}
{/exp:proform:results}
```

Sample Templates

Simple tag example

URL: **example.com/forms/survey**

```
{exp:proform:simple form_name="{segment_2}"}
```

Full tag example

This is a sample template for the Full Form template tag.

URL: **example.com/forms/contact_us**

Please look at this GitHub Gist for an example template.

Ajax Example

Please look at this GitHub Gist for an example of Ajax usage. Ajax is not supported but should work fairly well if you are familiar with jQuery load() and ajaxForm style techniques.

This template is designed for a form named contact_us with these fields:

Label	Name	Type
Email Address	email_address	Text
Name	name	Text

Notification example

Basic notification template. Usually placed at **notifications.group/default.html** in your ExpressionEngine template directory.

```
{fields}
    {if field_type != "hidden"}
        {field_label}: {field_value}
    {/if}
{/fields}
```

Extending ProForm

There are two major ways to extend ProForm:

Standard ExpressionEngine extensions, making use of the extensive list of hooks available.

ProForm Drivers, deeply integrated into core ProForm code, drivers offer more powerful integration with the module.

- Extension Hooks
- Drivers

Extension Hooks

ProForm provides a number of hooks in order to allow easy customization of it's functionality.

Hooks

Hook	Params	Return
proform_form_start	$module, $form_obj	$form_obj

Called near the beginning of processing the {exp:proform:form} tag.

This hook can be used to temporarily modify the form's properties in order to change the behavior of the form tag. It's also a good point to make queries on any custom database columns or tables you may have added, in order ot use them in later hooks.

Hook	Params	Return
proform_form_preparse	$module, $tagdata, $form_obj, $variables, $var_pairs	**array**($tagdata, $form_obj, $variables, $var_pairs)

Called just before parsing the contents of the {exp:proform:form} tag.

This hook can be used to inject any custom parsing you want into the tagdata, as well as changing the variables parsed by the tag. Any custom variables added will be parsed automatically. Note that for variable pairs, you must add their name to the $var_pairs array.

Remember to pass back **all** of the noted values, enclosed in an array.

Hook	Params	Return
proform_entries_start	$module, $form_obj	$form_obj

This hook is called just prior to processing the {exp:proform:entries} tag.

Similarly to the proform_form_start hook, this can be used to modify the form's properties temporarily before it is used to process the entries request.

Hook	Params	Return
proform_entries_row	$module, $form_obj, $row_vars	**array**($form_obj, $row_vars)

This hook is called once for each row of data returned by the {exp:proform:entries} tag. You can use this hook to customize the database results before they are parsed.

Remember to pass back **all** of the noted values, enclosed in an array.

Hook	Params	Return
proform_entries_end	$module, $form_obj, $return_data	$return_data

This hook is called just prior to returning the HTML result to the template from the {exp:proform:entries} tag. You can use this hook to customize the results before they are displayed.

Hook	Params	Return
proform_forms_start	$module, $forms	$forms

This hook is after getting the list of forms to be passed back by {exp:proform:forms}. You can use this to dynamically hide particular forms or otherwise change the forms in the listing.

Hook	Params	Return
proform_forms_row	$module, $form_obj, $row_vars	$row_vars

This hook is called once for each of the database rows in {exp:proform:forms}, just prior to parsing it.

This hook can be used to customize the data sent to the parsing function by modifying $row_vars.

Hook	Params	Return
proform_forms_end	$module, $forms, $return_data	$return_data

This hook is called just before returning the result of the {exp:proform:forms} tag to the template. Use it to customize the parsing of the tag.

Hook	Params	Return
proform_process_start	$module, $form_obj, $form_config, $form_session	**array**($form_obj, $form_config, $form_session)

This hook is called at the beginning of processing a form submission. Use it to modify the form configuration, the form object, or change values in the session. Remember to pass back **all** of the noted values, enclosed in an array.

Hook	Params	Return
proform_validation_start	$module, $form_obj, $form_session, $data	**array**($form_obj, $form_session, $data)

This hook is called at the beginning of processing a form submission’s validation. Use it to modify the form configuration, the form object, or change form values to be processed. Remember to pass back **all** of the noted values, enclosed in an array.

Hook	Params	Return
proform_validation_check_rules	$module, $form_obj, $form_session, $data, $validation_rules	$validation_rules

This hook is called after the complete rule set for the form has been built, but before processing it. You can inject dynamic rules at this point by modifying the $validation_rules array.

Hook	Params	Return
proform_validation_field	$module, $form_obj, $data, $field, $field_error	$field_error

This hook is called after processing the validation for each field. If there is an error for the field, it will be passed in through $field_error.

This value can be modified by adding an error or removing an existing error to prevent validation from failing.

Hook	Params	Return
proform_insert_start	$module, $form_obj, $data	$data

This hook is called just before inserting the data for the form submission into the database table for the form. This can be used to push the final data array to an external source, generating some sort of custom notification, as well as changing the data to actually be stored in the DB.

Hook	Params	Return
proform_insert_start_session	$module, $form_obj, $form_session	$form_session

Similarly to proform_insert_start, this hook is called just before inserting the data for the form submission into the database table for the form. The entire form_session can be inspected and modified if needed.

Hook	Params	Return
proform_insert_end	$module, $form_session	none

This hook is called just after inserting the data for the form submission into the database. This can be used to push the final data array to an external source, generating some sort of custom notification, or a number of other purposes.

Hook	Params	Return
proform_insert_end_ex	$module, $form_obj, $form_session	none

This hook is called just after inserting the data for the form submission into the database. This can be used to push the final data array to an external source, generating some sort of custom notification, or a number of other purposes.

Hook	Params	Return
proform_no_insert	$module, $form_obj, $data	$data

This hook is called instead of proform_insert_start and proform_insert_end for *share* type forms, where no insert is needed. Remember that you **must** return the data array in order for the form submission to work correctly.

This can be used to push the final data array to an external source, generating some sort of custom notification, or a number of other purposes.

Drivers

Custom driver documentation to come.

Knowledge Base

Knowledge Base articles

CAPTCHA

[CAPTCHA ERROR] displayed instead of an image

The [CAPTCHA ERROR] comes from ExpressionEngine’s core libraries, and usually just means that you need to make the captcha directory writable.

Completely disable CAPTCHA

To disable CAPTCHA, completely remove all references to variables with “captcha” in their name from your template and it will not be required. Be sure to use Snaptcha or the built in SpamGuard if you do so to prevent most spam from getting through your form. If you choose to use Snaptcha, leave references to captcha in the template code and activate the Snaptcha extension, which will be used instead of the image files.

If you are using the simple tag, you can find the template under third_party/proform/templates/default.html. See Creating a custom template for more information on customizing this file safely.

Encryption key error

You may receive this message from ExpressionEngine:

“In order to use the encryption class requires that you set an encryption key in your config file.”

Some features of ProForm and other add-ons require a unique encryption key to be set in ExpressionEngine’s config file. The easiest fix is simply to set an encryption key in your **system/ expressionengine/config/config.php** file as documented in the CodeIgniter user guide.

Set this to a random string value:

```
$config[‘encryption_key’] = “YOUR KEY”;
```

The value you set should be exactly 32 characters long.

Form rendering performance

Previous versions of ProForm relied heavily on the {field_type} variable in custom templates to determine how to render each field based on it's type. This caused some forms to render very slowly.

In ProForm 1.65+, each {field_type} value generates a custom tag pair which should be used instead of normal {if} conditionals. For instance, if you had previous written this line to check for checkboxes:

```
{if field_type == “checkbox”}…{/if}
```

You should change this to:

```
{checkbox}…{/checkbox}
```

In most cases, conditionals on the {field_type} variable should continue to function, however the custom tag pairs are much faster to render and should improve your form's performance. All example code will now be provided in the new style.

List fields performance

If you have list or relationship type fields with hundreds or thousands of entries, and are still using pre-1.65 templates, you should update your template to use the latest variable pairs available. Due to the way that ExpressionEngine's conditional system works, previous versions would spend a lot of time rendering styles of lists that were not actually used. Instead of using a conditional such as

```
{if pf_field_setting_style == "check"}
{/if}
```

you should instead use:

```
{check_style}
{/check_style}
```

These template tags are replaced immediately when ProForm processes the template code, eliminating duplicate processing.

Generating a debug dump

Only do this if you have been asked to do so in a support thread.

Please do the following to generate a debug dump that will help us investigate this issue:

- Go to ProForm under Add-ons > Modules > ProForm.
- Click the Maintenance button in the top right.
- Click Export Forms & Fields.
- Select the form that has this issue.
- Click Generate Export.
- Email this file to us at support [at] metasushi.com and post a reply to your support thread letting us know you have sent it.

! Never send this file to anyone else, it contains sensitive information such as SMTP connection information.

Notifications cannot be sent

Please try each of these in sequence.

! **Notice**

You will need to have the assistance of a sysadmin on your server in order to debug and fix most email configuration issues.

Step 1

First, verify that you can send and receive email using normal ExpressionEngine methods. The same internal code is used to send notifications from ProForm.

1. Go to Tools > Communicate and send yourself a message.

2. After a few minutes, check your inbox and spam folders on your email service to see if you have recieved the message. If you have not, ProForm will not be able to send email either. See ExpressionEngine’s email configuration page and contact EllisLab for help with your email configuration.

Step 2

If you are able to send an email via Tools > Communicate but still get this error, usually this means that there is no template assigned to the form’s admin notification settings.

Create a template group:

1. Go to Design > Templates > Template Manager
2. Click New Group. I suggest naming the template “notifications”.
3. Click Submit

This template group MUST have a template with a name other than “index”. Templates named “index” CANNOT be used to send notifications, for security reasons:

1. Click the new Template Group in Template Manager
2. Click Create a New Template
3. Name the template default
4. Click Crete and Edit
5. Enter the sample template from this GitHub gist
6. Click Update

Make sure the notification template group is assigned:

1. Go to Add-ons > Modules > ProForm > Modules settings
2. Specify a template group for the setting Notification Template Group.
3. Click Submit

Make sure the form has default set as it’s notification template:

1. Go to Add-ons > Modules > ProForm > Edit Settings for the form
2. Under Notification List Settings, select “default”
3. Set a value for Subject such as “New request received”
4. Click Save Form at the bottom of the page

Your form should now have notifications configured.

Step 3

If the template was assigned properly, you will need to get more debug output.

1. Set email debugging in ExpressionEngine to be on. Go to Admin > Email Configuration in the main menu, then turn on the option Enable Email Debugging

2. Go to Tools > Communicate and send yourself a message. This should generate a large block of output from ExpressionEngine detailing why the message failed.

3. Give the contents to your sysadmin or post it to your support ticket with us.