Validation Rules

To submit any form you need some rules to give inputs.

This is the list of rules you can use for fields.

1. required
    - **Returns FALSE if the form field is empty.**
2. valid_email
    - **Returns FALSE if the form field does not contain a valid email address.**
3. valid_emails
    - **Returns FALSE if any value provided in a comma separated list is not a valid email.**
4. min_length
    - **Returns FALSE if the form field is shorter than the parameter value.**
    - **Example: min_length[5]. Where 5 is a parameter of length.**
5. max_length
    - **Returns FALSE if the form field is longer than the parameter value.**
    - **Example: max_length[10]. Where 10 is a parameter of length.**
6. exact_length
    - **Returns FALSE if the form field is not exactly the parameter value.**
    - **Example: exact_length[10]. Where 10 is a parameter of length.**
7. matches
    - **Returns FALSE if the form field does not match the defined value of parameter.**
    - **Example: matches[field]. Where field is parameter name of field to match with.**
8. Alpha
    - **Returns FALSE if the form field contains anything other than alphabetical**

characters.

9. alpha_numeric
    - **Returns FALSE if the form field contains anything other than alpha-numeric characters.**
10. alpha_dash
    - **Returns FALSE if the field contains anything other than alpha-numeric characters, underscores or dashes.**
11. numeric
    - **Returns FALSE if the form field contains anything other than numeric characters.**
12. integer
    - **Returns FALSE if the form field contains anything other than an integer.**
13. decimal
    - **Returns FALSE if the form field contains anything other than a decimal number.**
14. is_natural
    - **Returns FALSE if the form field contains anything other than a natural number.**
15. is_natural_no_zero
    - **Returns FALSE if the form field contains anything other than a natural number, but not zero.**
16. valid_ip
    - **Returns FALSE if the supplied IP is not valid.**
17. sm_captcha_validate
    - **Returns FALSE if the EE Captcha is not valid. (Default for EE captcha field.)**
18. is_unique
    - **Returns FALSE id the field value match with the field in database table.**
    - **Example: is_unique[members.email]. Where "members" is table name and "email" is field of that table.**
19. exists_email

- **Returns FALSE if email field entered by user is not exists in members table.**
- **Used in forgot password form email field by default.**

20. auth_password
    - **Returns FALSE if user entered wrong password.**
    - **Use in edit profile form to validate current password.**

Hooks

Hooks are use to modify the data or writing your own flow on plugin without changing the core code of plugin. List of hooks given in smart members are listed below:

1. Build form starts:
    a. sm_build_form_start
    b. sm_before_logout_link
2. Build forms ends:
    a. sm_build_form_end
3. Start submit forms:
    a. sm_submit_form_start
    b. sm_logout_start
4. End submit forms:
    a. sm_submit_form_end
5. If errors found in forms:
    a. sm_error_in_form
    b. sm_outer_error
6. Initialize validation rules:
    a. sm_init_validation
7. View profile start

a. sm_view_profile_start

b. sm_view_profile_end

8. Before send email:
    a. sm_before_send_email
9. Total fields listed (Constructor Hook)
    a. sm_total_fields
10. Social Media login **(Pro Feature)**
    a. sm_before_social_login

    b. sm_after_social_login
11. Import member Hooks **(Pro Feature [EE4 only])**
    a. sm_element_before_import

    b. sm_element_after_import
12. You can also use ExpressionEngine default hooks given below:
    a. member_member_register_start

    b. member_member_register_errors

    c. member_member_register

    d. member_member_logout

    e. member_delete

    f. member_update_start

    g. member_update_end

    h. cp_members_validate_members

    i. member_register_validate_members

sm_build_form_start

This hook will call every time a forms build start. This hook let user to modify data before form generates.

Hook Name	sm_build_form_start

Hook Name	sm_build_form_start
Parameter 1	Source: Actual source of form.[i.e, “registration”, “edit_profile”, “forgot_password”, “reset_password”]
Parameter 2	Total Fields: Array of total possible fields.
Return	Total Fields.

---

sm_before_logout_link

This hook will call when generates logout url from {exp:smsp:logout}.
It has Logout URL in parameter of hook function.

Hook Name	sm_before_logout_link
Parameter	Logout URL
Return	Logout URL

---

sm_build_form_end

This hook will call every time a forms ends build. This hook will call before replace the actual data with tagdata. The parameter contains Source and the Variable array of form.

Hook Name	sm_build_form_end
Parameter 1	Source: Actual source of form.[i.e, “registration”, “edit_profile”, “forgot_password”, “reset_password”]
Parameter 2	Variable array of form parameters and hidden fields.
Return	Variable array of form parameters and hidden fields.

---

sm_submit_form_start

This hook will call when user submits a form. This hook will call before validation hook.

Hook Name	sm_submit_form_start
Parameter 1	Source: Actual source of form.[i.e, “registration”, “edit_profile”, “forgot_password”, “reset_password”]
Parameter 2	Parameter array passed in exp l oop of form

Return	Parameter array passed in exp l oop of form

---

sm_logout_start

This hook will call when a user click on logout URL. The data passed in parameter is the GET array which can secure action parameter or return url.

Hook Name	sm_logout_start
Parameter	GET array
Return	GET array

---

sm_submit_form_end

This hook will call when we submits a form without any error and it update the form data into database.

Hook Name	sm_submit_form_end
Parameter 1	Source: Actual source of form.[i. e, “registration”, “edit_profile”, “forgot_password”, “reset_pass word”]
Parameter 2	Return URL
Return	Return URL

Return	Return URL

sm_error_in_form

This hook will call when we found errors in our forms submitted data.

Hook Name	sm_error_in_form
Parameter 1	Source: Actual source of form.[i.e, “registration”, “edit_profile”, “forgot_password”, “reset_password”]
Parameter 2	Array of errors
Return	Array of errors

sm_outer_error

It is possible that error occurs in the form of the field of some xyz field and you forgot to add {error:xyz} in form. In that case error will show in ee default gray screen.
(Only called if error_reporting=”inline” is passed as parameter in form)

Hook Name	sm_outer_error
Parameter 1	Source: Actual source of form.[i.e, “registration”, “edit_profile”, “forgot_password”, “reset_pass

	word”]
Parameter 2	Array of errors
Return	Array of errors

---

sm_init_validation

This hook will call before initialization of validation array to field. So you can control rules of fields before assign rules to fields.

Hook Name	sm_init_validation
Parameter 1	Source: Actual source of form.[i.e, “registration”, “edit_profile”, “forgot_password”, “reset_password”]
Parameter 2	Array of validation array (Form settings)
Return	Array of validation array (Form settings)

---

sm_view_profile_start

This hook will call when you trigger {exp:smsp:profile} tag.

Hook Name	sm_view_profile_start

Hook Name	sm_view_profile_start
Parameter	NA
Return	NA

---

sm_view_profile_end

This hook will call before the actual profile data will replace with tagdata.
It will pass member data array in parameter and accept the same as return.

Hook Name	sm_view_profile_end
Parameter	Array of member data
Return	Array of member data

---

sm_before_send_email

This hook will call before any email sends with smart members. i.e., Registration email, forgot password email.

Hook Name	sm_before_send_email
Parameter 1	Source: Actual source of form.[i.e, “registration”, “edit_profile”, “forgot_password”, “reset_password”]

Parameter 2	Array of validation array (Form settings)
Return	Array of validation array (Form settings)

---

sm_total_fields

This is constructor hook. All possible fields that can exists in form or profile can contains. Array of this field is parameter of this hook.

Hook Name	sm_total_fields
Parameter	Array of total possible fields a form can contains.
Return	Array of total possible fields a form can contains.

---

sm_before_social_login (Pro Feature)

This hook will call before social login API call.

Hook Name	sm_before_social_login
Parameter	URL to be called for authenticate user with social site.

Return	URL to be called for authenticat e user with social site.

---

sm_after_social_login (Pro Feature)

This is constructor hook. All possible fields that can exists in form or profile can contains. Array of this field is parameter of this hook.

Hook Name	sm_after_social_login
Parameter	Return URL to be called after su ccessful authentication.
Return	Return URL to be called after su ccessful authentication.

---

sm_element_before_import (Pro Feature [EE4 >= v2.0.3])

This hook will call just before member will insert or update in database. Hook has one parameter that contains array that going to be insert/update so one can modify member data before insert it into the database.

Hook Name	sm_element_before_import
Parameter	Full Array that going to be save

	in database.
Return	Same array after modification.

---

sm_element_after_import (Pro Feature [EE4 >= v2.0.3])

This hook will call just after member will save in database. Hook has one parameter that contains array that just saved into the database and second parameter as member_id which holds the data.

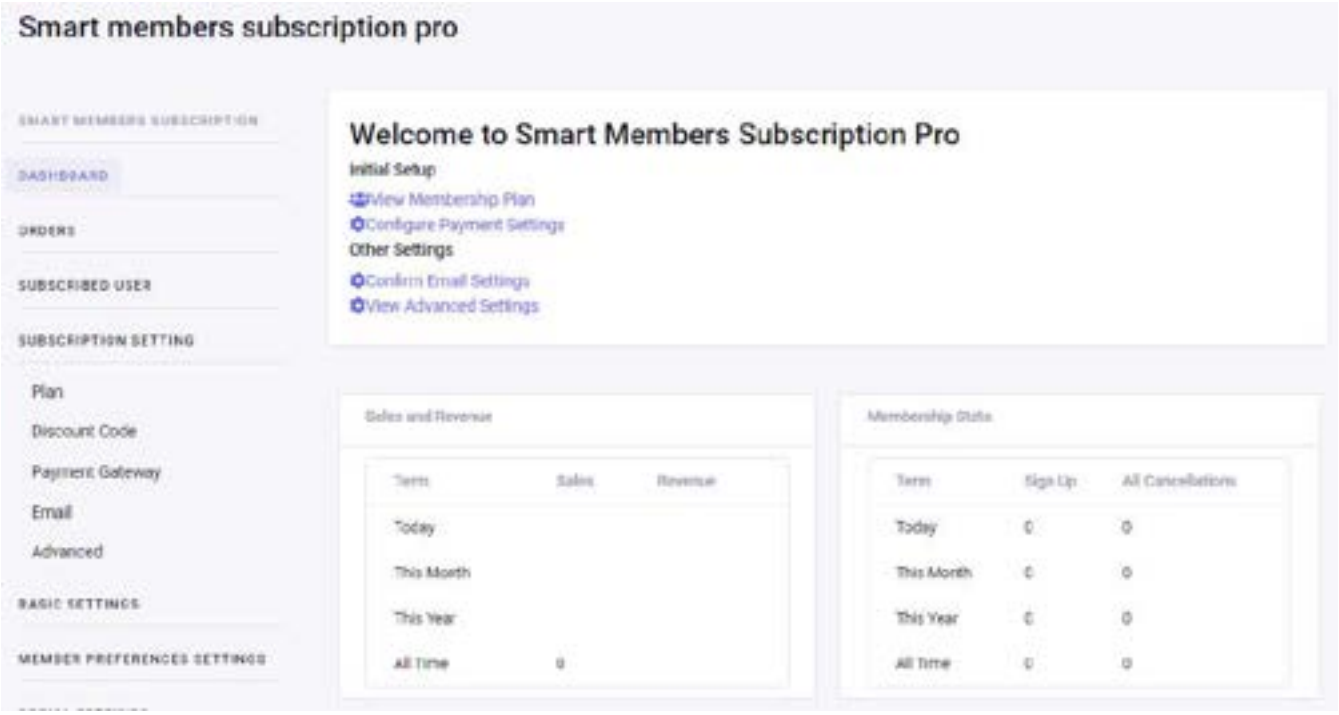
Hook Name	sm_element_after_import
Parameter 1	Full Array that saved in databas e.
Parameter 2	Member ID holding particular member.
Return	NA

Dashboard - New

Here the Dashboard have all the initial setup links and some other essencial links.

Dashboard also come with some reports Ex. Sales and Revenue, Membership Stats, Visits, View and Login and Recent Orders.

Dashboard

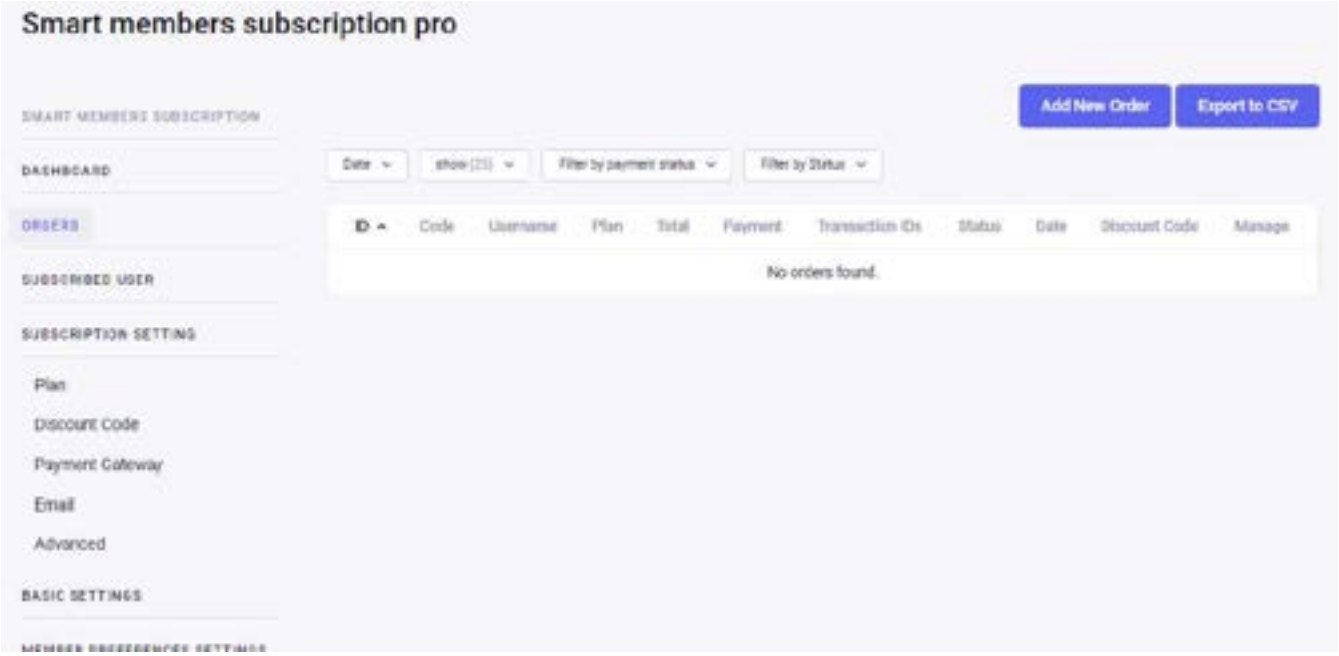


Orders - New

Here we have all the orders list with tha basic operation of Edit,Copy,Delete and Email.

You can add new order here and export all the orders data into CSV file.

Orders



Add New Order

Click on “Add New Order” button to add an order.

1. ID
    - **The ID will be generated when you save.**
2. Code
    - **This will be generated randomly.**
3. Membership Level ID
    - **Membership level ID generated when level created.**
4. Fill the following fields to complete Billing details.
    - **Billing Name**
    - **Billing Street**
    - **Billing City**
    - **Billing State**
    - **Billing Postal Code**
    - **Billing Country**
    - **Billing Phone**

5. Sub Total
    - **Total amount without tax and coupon amount.**
6. Tax
    - **Tax amount that you charge depends on locations or any other circumstances.**
7. Total
    - **Total amount should be subtotal + tax – couponamount.**
8. Payment Type
    - **Insert payment type here e.g. PayPal Express, PayPal Standard, Credit Card.**
9. Card Type
    - **Insert card type here e.g. Visa, MasterCard, AMEX, etc**
10. Account Number
    - **Insert Bank account number.(Obscure all but last 4 digits)**
11. Expiration Month
    - **Expiration Month of card**
12. Expiration Year
    - **Expiration Year of card**
13. Status
    - **Set status of order e.g. Cancelled,Error,Pending,Refunded,Review,Success,Token**
14. Gateway
    - **Select payment gateway that you made payment through e.g. stripe, paypal, authorize.net etc**
15. Gateway Environment
    - **Select payment gateway environment, either Testing or Sandbox.**
16. Payment Transaction ID
    - **Payment Transaction ID generated by the gateway. Useful to cross reference orders.**
17. Subscription Transaction ID

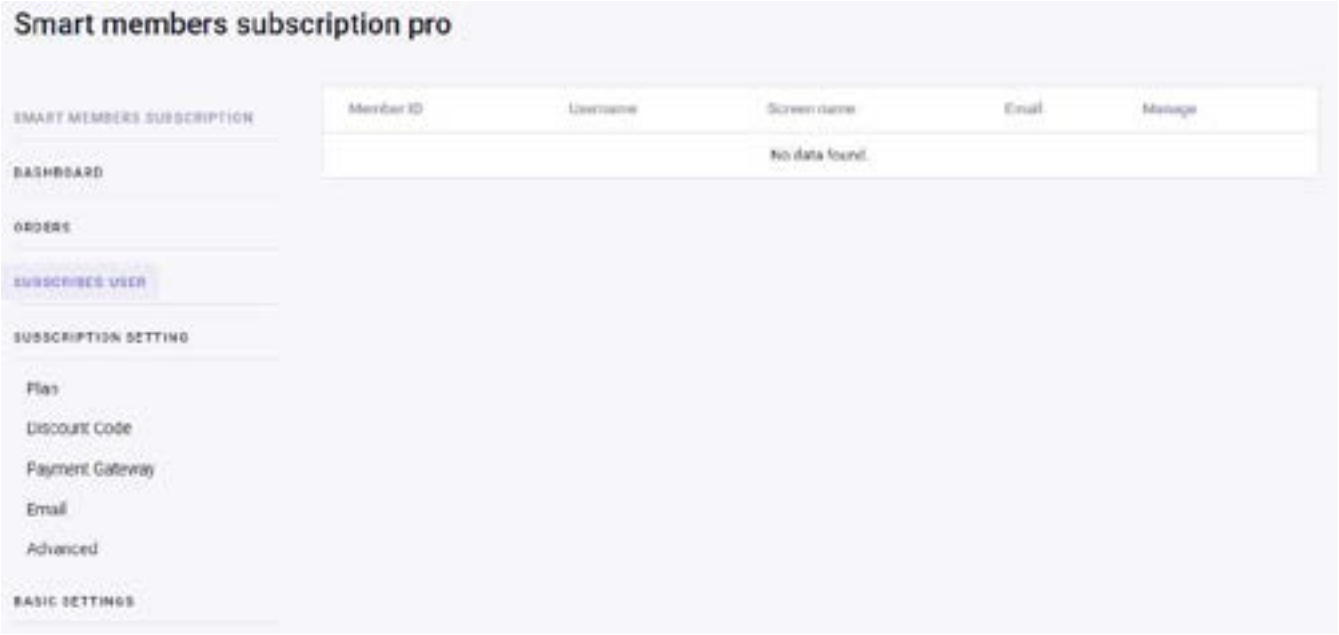
17. Subscription Transaction ID

    - **Subscription Transaction ID generated by the gateway. Useful to cross reference subscriptions.**

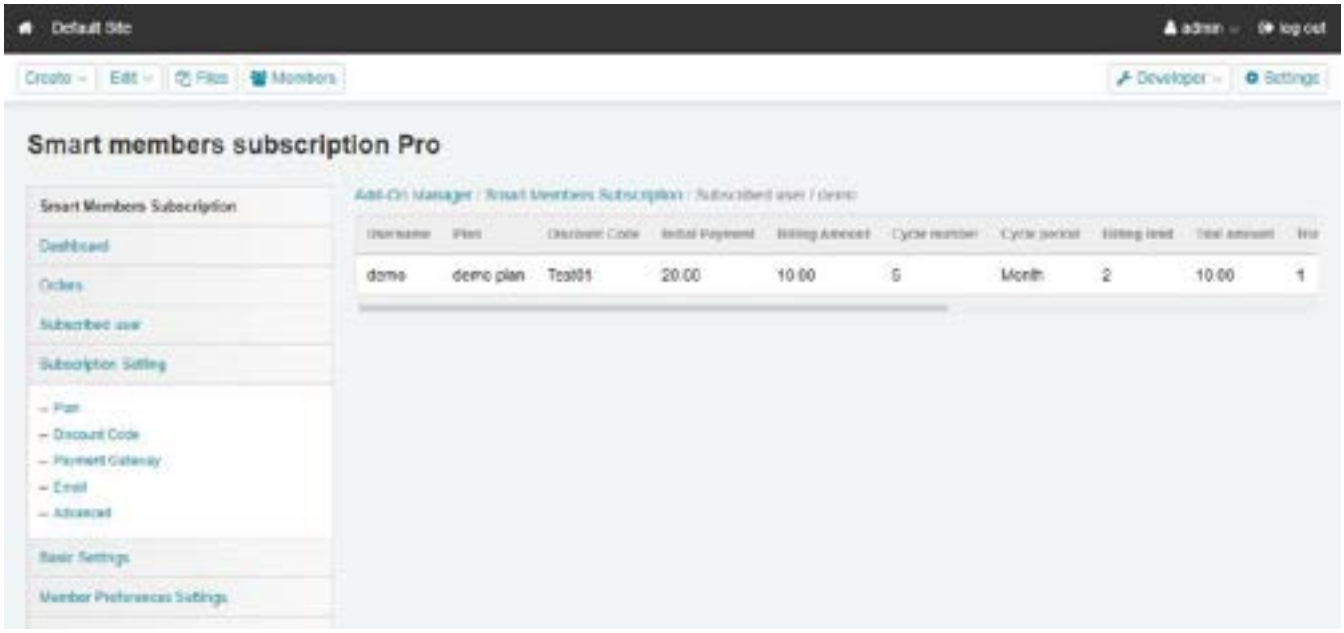
18. Date
    - **Generated by the gateway. Useful to cross reference subscriptions.**
19. Notes
    - **Any notes generated through gateway or others.**
20. And save the order

Subscribed Users - New Subscribed Users

Here you can see all the users who have purchased the subscription.



If you want to see the detailed information of the subscription, click on the eye icon in manage column of the table.



Subscription Settings - New

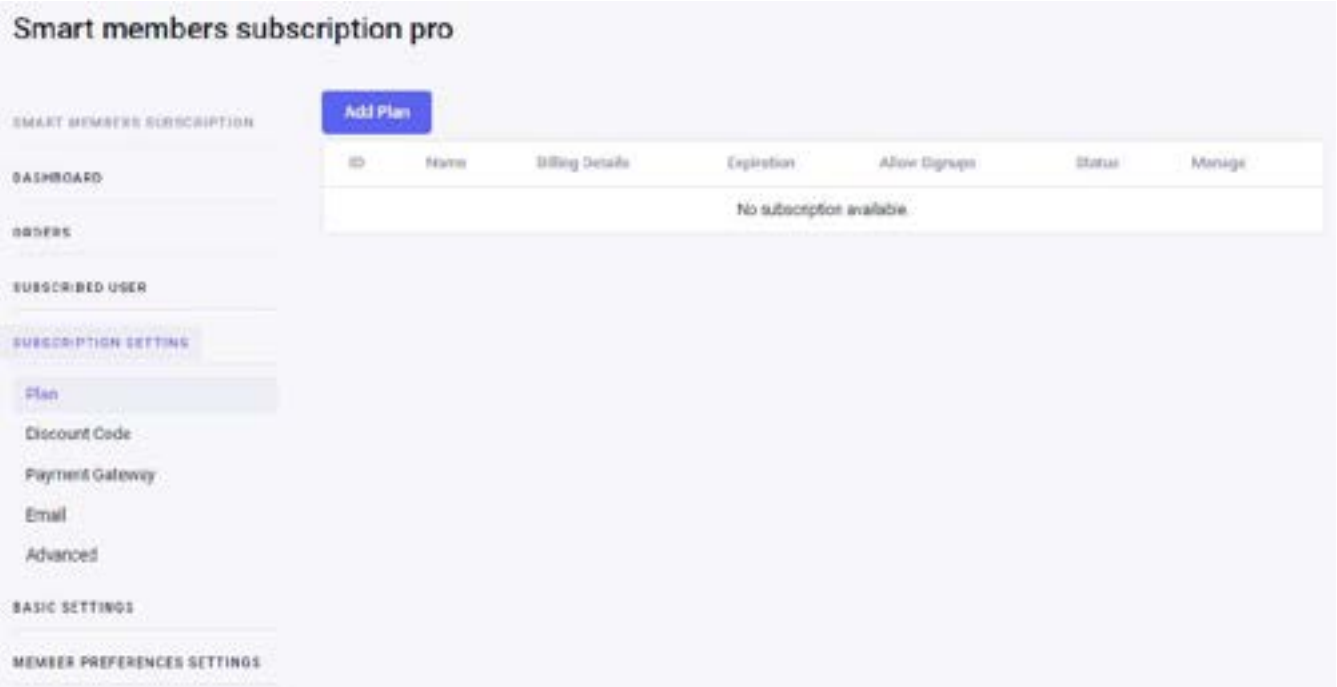
Subscription settings allows you to customize settings for Subscription Plan, Discount Codes,Payment Gateway, Email and some other advance setting as per your requirements.

Under this setting you can create subscription, discount code and can manage as well. Payment Gateway allows to setup payment environment, payment method and SSL settings.

Subscription Settings

1. Plan

- **All the plans list created by site owner for users. List can be manage by edit, copy and delete operations.**



Add New Plan

Click on “Add New” button to add new plan. And fill the following details

1. Name
    - **The name of your Membership level (public).**
2. Description
    - **Optional content shown on the checkout page.**
3. Confirmation Message
    - **Optional content shown on the checkout confirmation page.**
4. Billing Details:
    1. Initial Payment
        - **This is what is charged at checkout. E.g. $10.**
    2. Recurring Subscription
        - **Check if your level has a recurring payment.**
    3. Billing Amount
        - **The amount to be billed one cycle (day(s), week(s), month(s), year(s)) after the initial payment.**
    4. Billing Cycle

- **The total number of recurring billing cycles for this level, including the trial period (if applicable) but not including the initial payment. Set to zero if membership is indefinite. At the end of the last cycle, billing will stop, but the user will still have their membership level access unless you also set an expiration below.**

5. Custom Trial
    - **Check if your level has a custom trial period. (Optional; Trial Payment and Trial Period.**

6. Trial Billing Amount
    - **The amount (can be zero for free trial) and number of cycles to be billed (day(s), week(s), month(s), year(s)) after the initial payment.**

5. Other Settings
    1. Disable New Signups
        - **Disables new signups and hides the level from default page.**

6. Content Settings
    1. Channels
        - **Select the channels for which this plan has been created.**

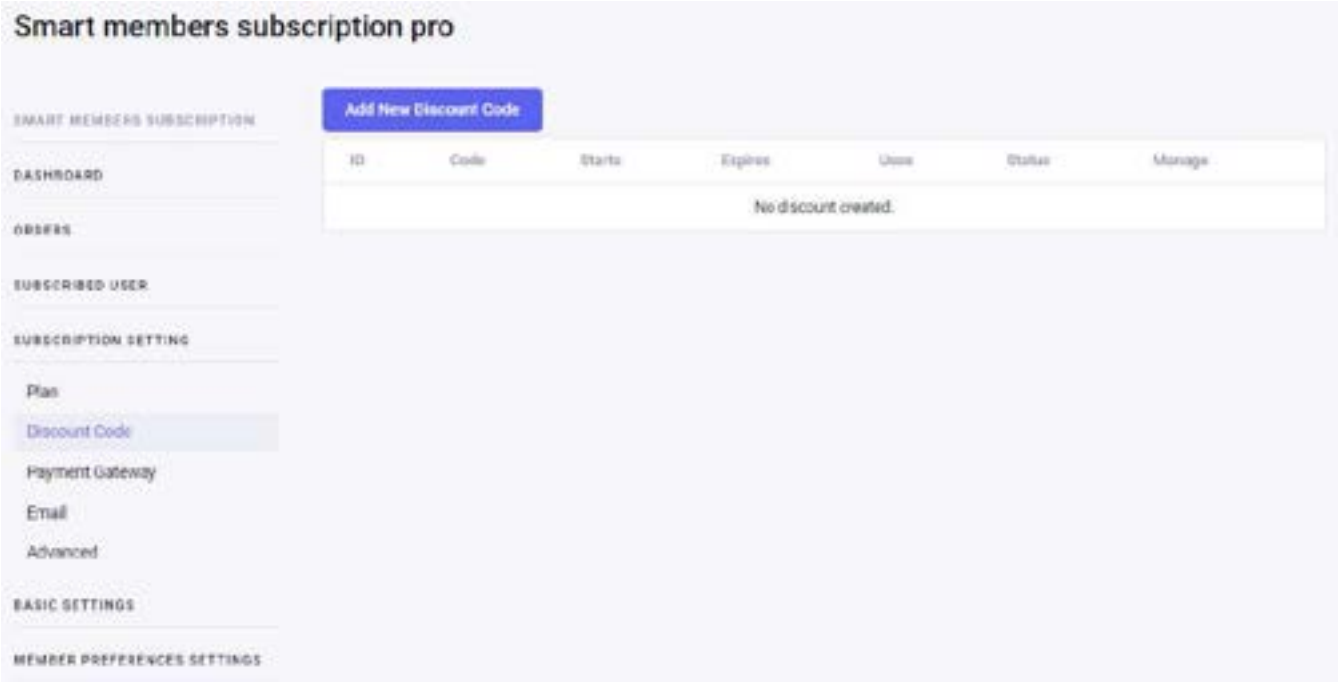
    2. Member groups
        - **Select the Member groups for which this plan has been created.**

    3. Categories
        - **Select the Categories for which this plan has been created.**

7. Save Plan

2. Discount Codes

All the discount code list created by site owner for users. List can be manage by edit, copy and delete operations.



Add New Discount Code

Click on “Add New Discount Code” button to add new discount code. And fill the following details

1. ID
    - **This is the auto-generated ID, internally tracked (not public).**
2. Code
    - **The value of the code members can enter at checkout (public).**
3. Start Date
    - **The date you want the code to begin working.**
4. End Date
    - **The date you want the code to expire / stop working.**
5. Uses
    - **How many times the code can be used (leave blank for unlimited uses).**
6. And Save.

Setting the Discount Code Pricing for a Plan

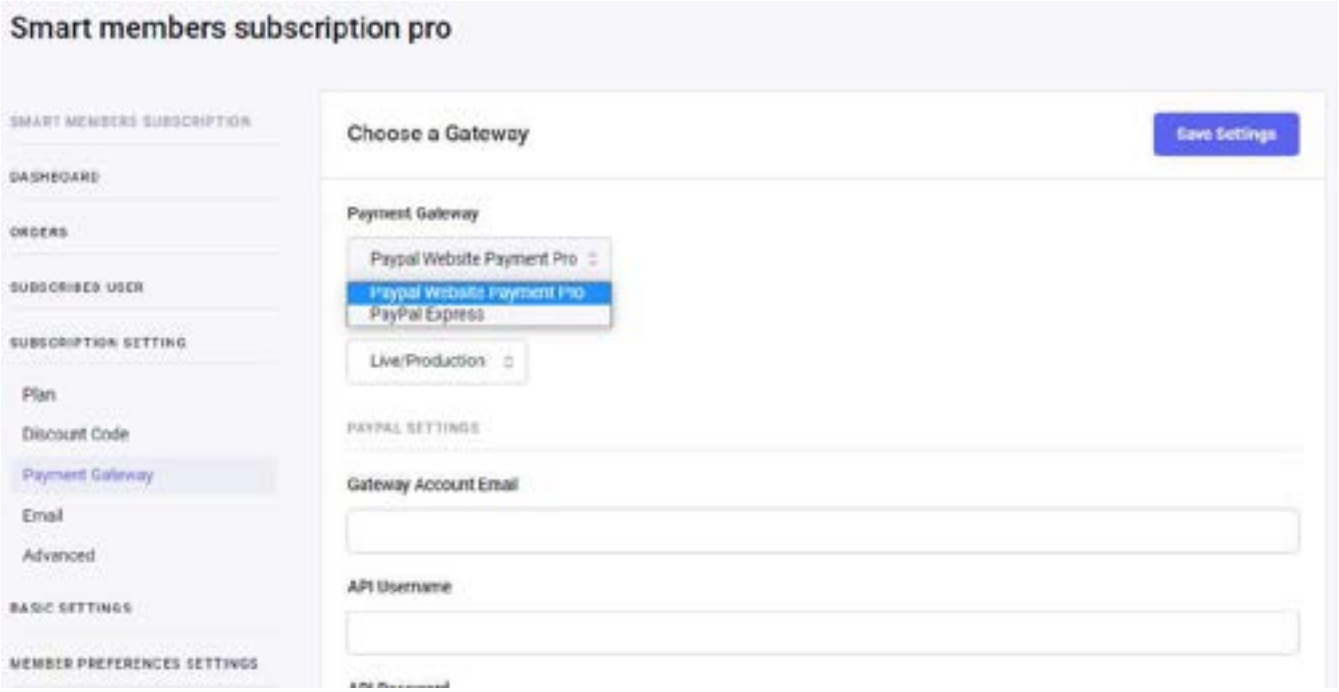
In section “Which Levels Will This Code Apply To?” check the available plans and fill

the following details.

1. Initial Payment
    - **The initial amount collected at registration.**
2. Recurring Subscription
    - **Check if this level has a recurring subscription payment.**
3. Billing Amount
    - **The amount to be billed one cycle after the initial payment.**
4. Billing Cycle Limit
    - **The total number of recurring billing cycles for this level, including the trial period (if applicable) but not including the initial payment. (Set to zero if membership is indefinite.)**
5. Custom Trial
    - **Check to add a custom trial period.**
6. Membership Expiration
    - **Check this to set when membership access expires.**
7. Expires In
    - **Set the duration of membership access. Note that the member's recurring subscription (if any) will be cancelled when the membership expires.**

3. Payment Gateway & Security

Payment Setting setup is simple just select a gateway and paste the appropriate API information into the Payment Gateway. Further you can setup for Currency,Tax and SSL.



Here is the detailed information about fields to be fill on this page.

1. Payment Gateway
    - **Click the drop-down and select one of the payment options available.**
2. Gateway Environment
    - **Choose the test or live gateway.**
3. Complete the fields for your chosen Gateway
    - **Complete the fields for your chosen Gateway. For Stripe, enter your Stripe secret key and publishable key. Other payment methods will have different processes.**
4. Sales Tax
    - **For this example, we are not charging Sales Tax. Always check with your accountant or your state tax office to see if tax is required and the tax rate for your memberships. More Information: Non-US Taxes | VAT Tax.**
5. Force SSL
    - **An SSL certificate can be purchased and setup by your website hosting provider. If you are unsure, GoDaddy is a good SSL provider where you can get a cheap SSL to use no matter where your site is used.**
6. SSL Seal Code

6. SSL Seal Code

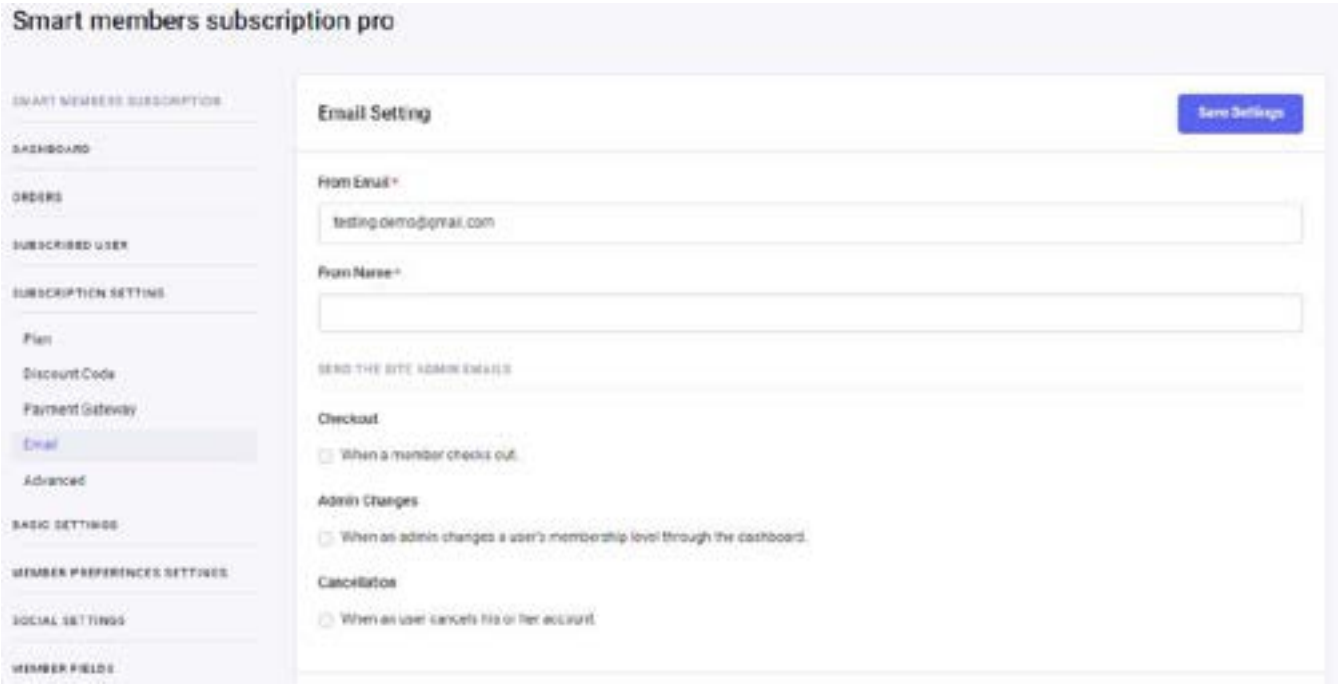
    - **Display an image and link so users can validate your site's SSL Certificate validity on membership checkout. (Note: if your provider's seal graphic displays via a script tag, you must use custom code to display the seal.) This is different from the "SSL Certificate", which is the private key certificate that must be installed by your web host.**

7. Extra HTTPS URL filters
    - **Check this if you are using SSL and have warnings on your checkout pages.**

8. Save Settings

4. Email

Email Setting page allows you to control emails sent by your site.



Here is the detailed information about fields to be fill on this page.

1. From Email and From Name
    - **Use these fields to change the email address and name used ExpressionEngine sends email communications to your members.**

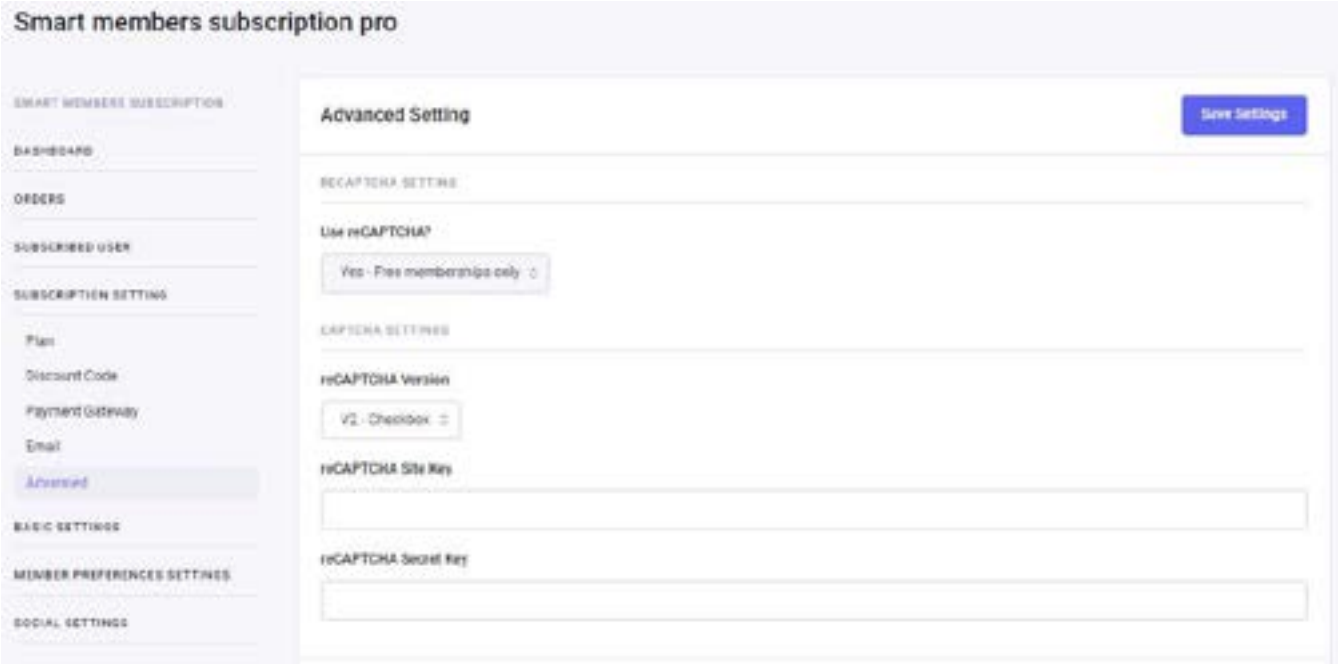
2. Send the site admin emails
    - **Keep all boxes checked so you receive email confirmations about**

membership activity on your site.

3. Save Setting

5. Advanced Setting

Advanced Settings admin page allows you to control other customization settings for your site.



Here are the field description and its uses.

1. Use reCAPTCHA?
    - **Requiring a credit card is the best spam prevention. But if you have a free level, you can turn on reCAPTCHA to add to the plugin's built-in spam prevention.**

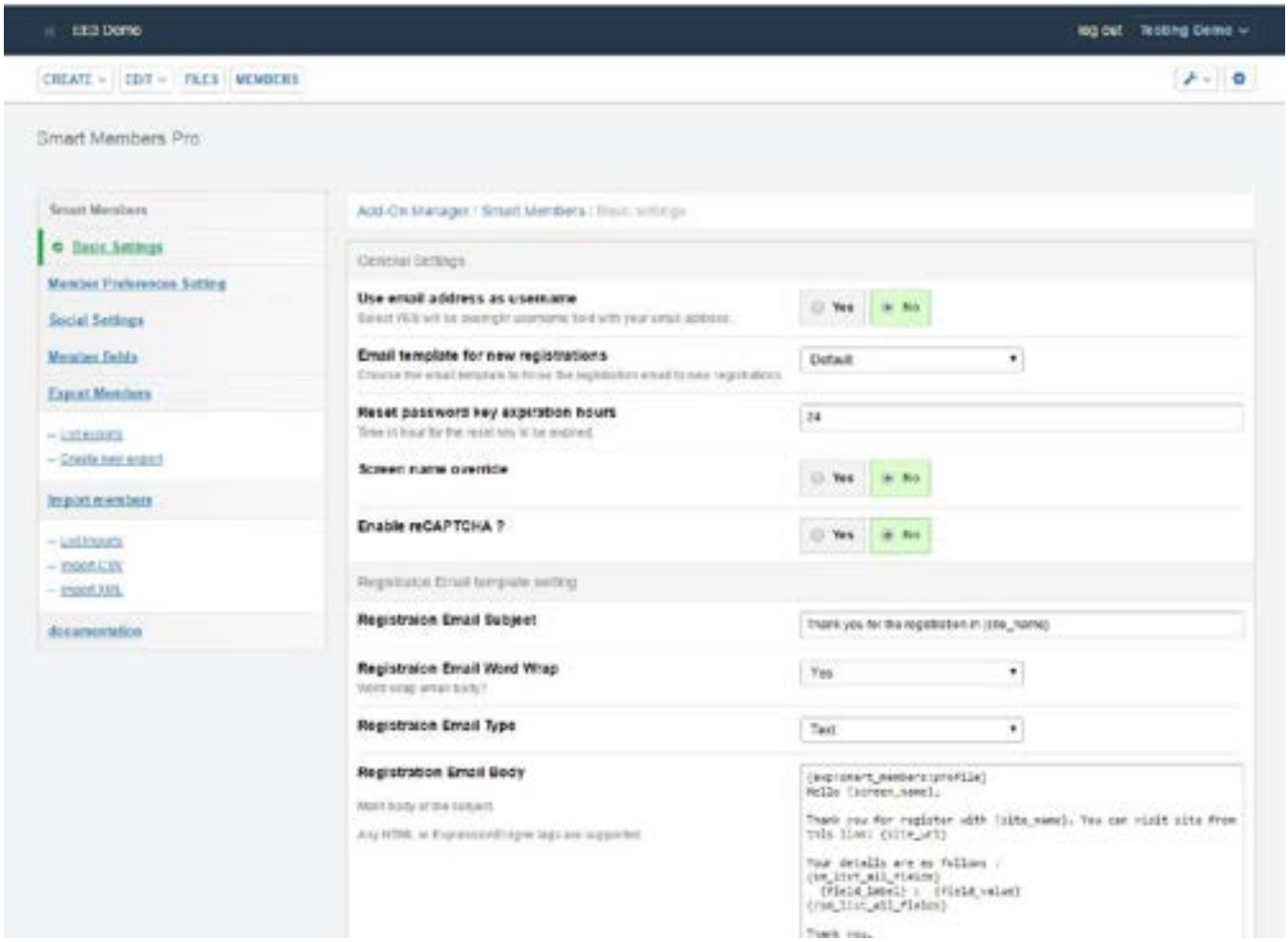
Basic Settings

Basic settings allows you to customize plugin settings as per your requirements. Form is easy to understand and settings will effect in every module we use in frontend.

Setting form allows you to mark username as password, override screen name

field, set registration email template and forgot password email template, set reset password template etc.

Basic setting form (Default view)



There are multiple fields that can change as per your requirements. Details about those fields are given below:

1. Use email address as username:

This field can be use to have same value for both username and email field. Checking this to YES will no longer need email field in registration page. You only need to use username field and put the type as email so one can not enter the text input rather then email address.

Example:

```
<input type="email" name="username" placeholder="Email Addr
```

```
ess">
{error:username} //if error_reporting="inline"
```

Note "type" of input is "email" and "name" of input is "username". Same will follow in edit profile module.

2. Email template for new registrations:

You have 2 options here. You can either put every code of email templates for registrations and forgot password in backend setting form or you can generate template to frontend.

Select "Default" if you want to use backend setting form email templates for send emails.

Select front end template if you want to generate your own templates to send emails to user. Select the templates from front end is necessary for this option.

3. Screen name override:

This option allows you to use any custom field as Screen name. That will took the field on registration process and override screen name.

As screen name field must be unique, If one take a field name and that name is already in screen name field of any other user, the default screen name will use instead of override.

4. Enable Re-captcha ?

Enabling this will ask you recaptcha key and secret. You can generate recaptcha key and secret from google recaptcha site.
URL: https://www.google.com/recaptcha/admin#list

Enter your domain and generate key and secret. You can then use google recapthca in your front end module by entering enable_recaptcha="yes" in parameter and code of google recaptcha in template:

```
{if captcha}
```

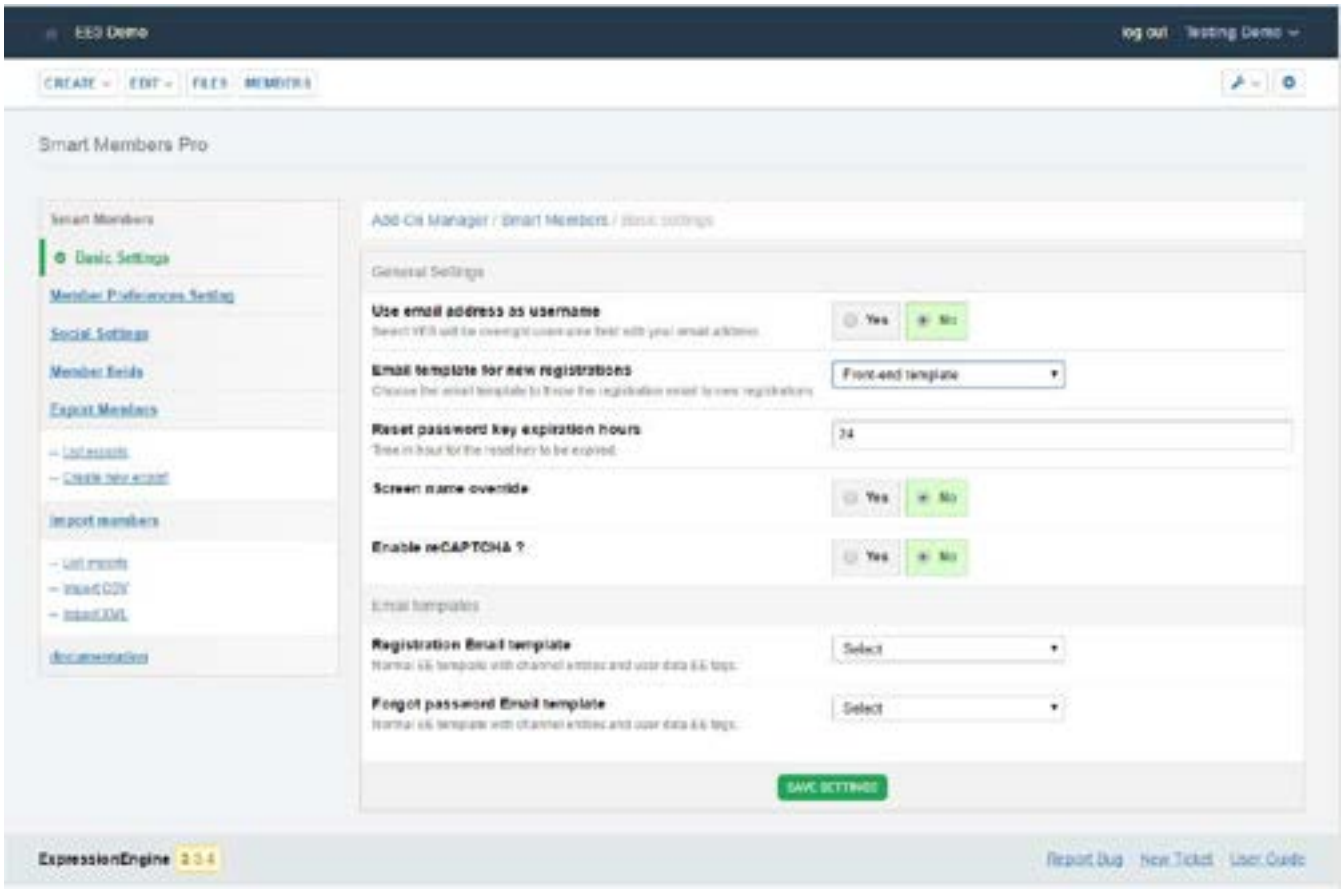
```
<p>
<label for="captcha">Please enter in the word you see:</label>
{captcha}
<input type="text" name="captcha" id="captcha" />
{error:captcha}
</p>
{if:elseif recaptcha}
<p style="margin: 0;"><label for="recaptcha">Click the checkbox
</label>
{recaptcha}
{error:recaptcha}
</p>
{/if}
```

Captcha and recaptcha to use in if else condition is helpful in case if you pass enable_recaptcha="yes" in parameter and not give API key and Secret in backend setting form. In such case it will display normal captcha in place of recaptcha.

5. Email template setting:

If you choose Email template for new registrations to default, You need to pass the email settings in form at down otherwise you only need to select the email templates.

Basic Settings (Email template view)

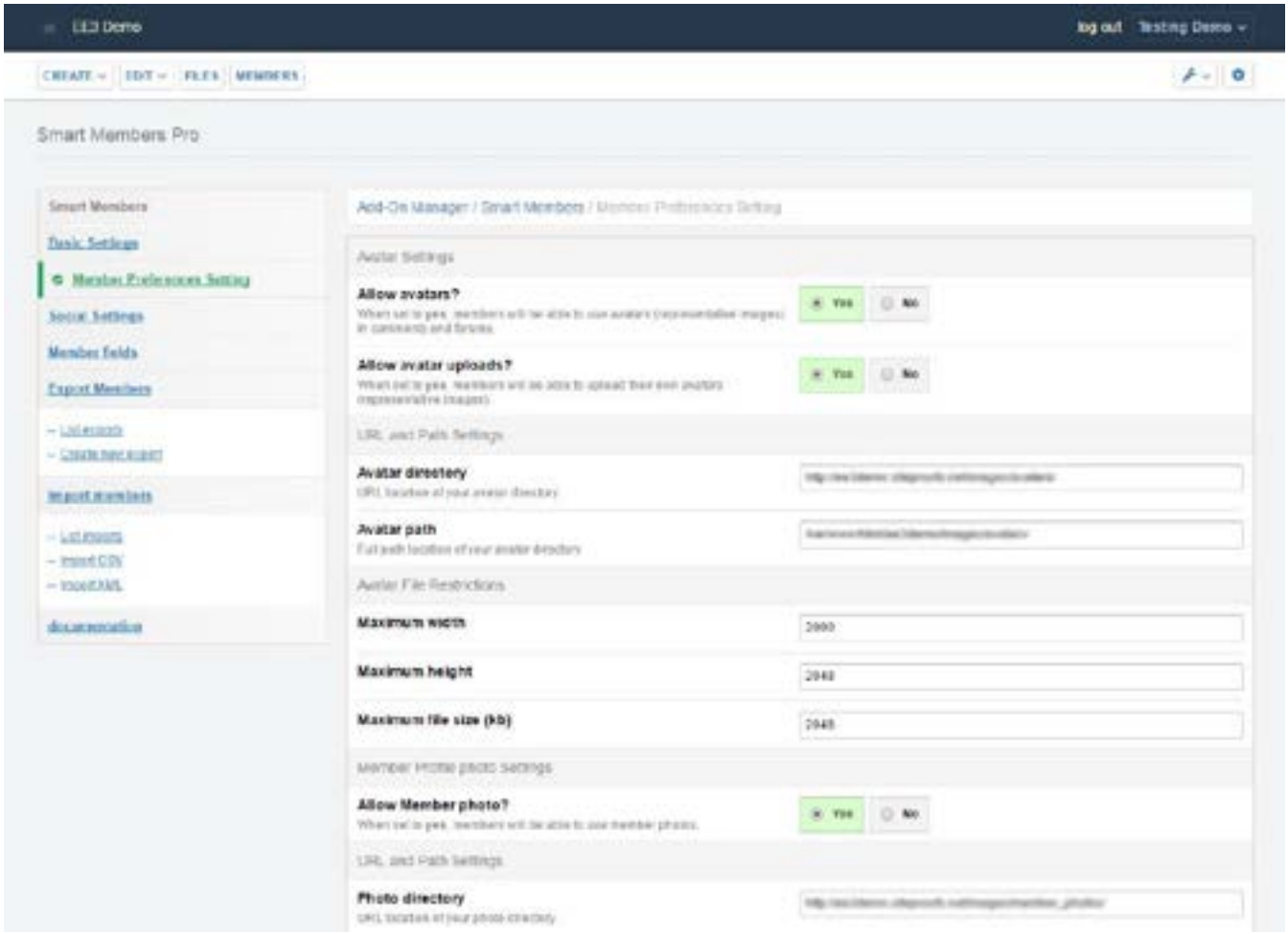


Member Preferences

Member preferences setting form is only developed for EE3 users as static images such as Photo file and Signature Image fields settings of path, height, width etc. Settings are available in member preferences in EE2, But such setting forms are not available for EE3 right now.

You can simply adjust the settings for photo file, avatar and signature image settings from this module.

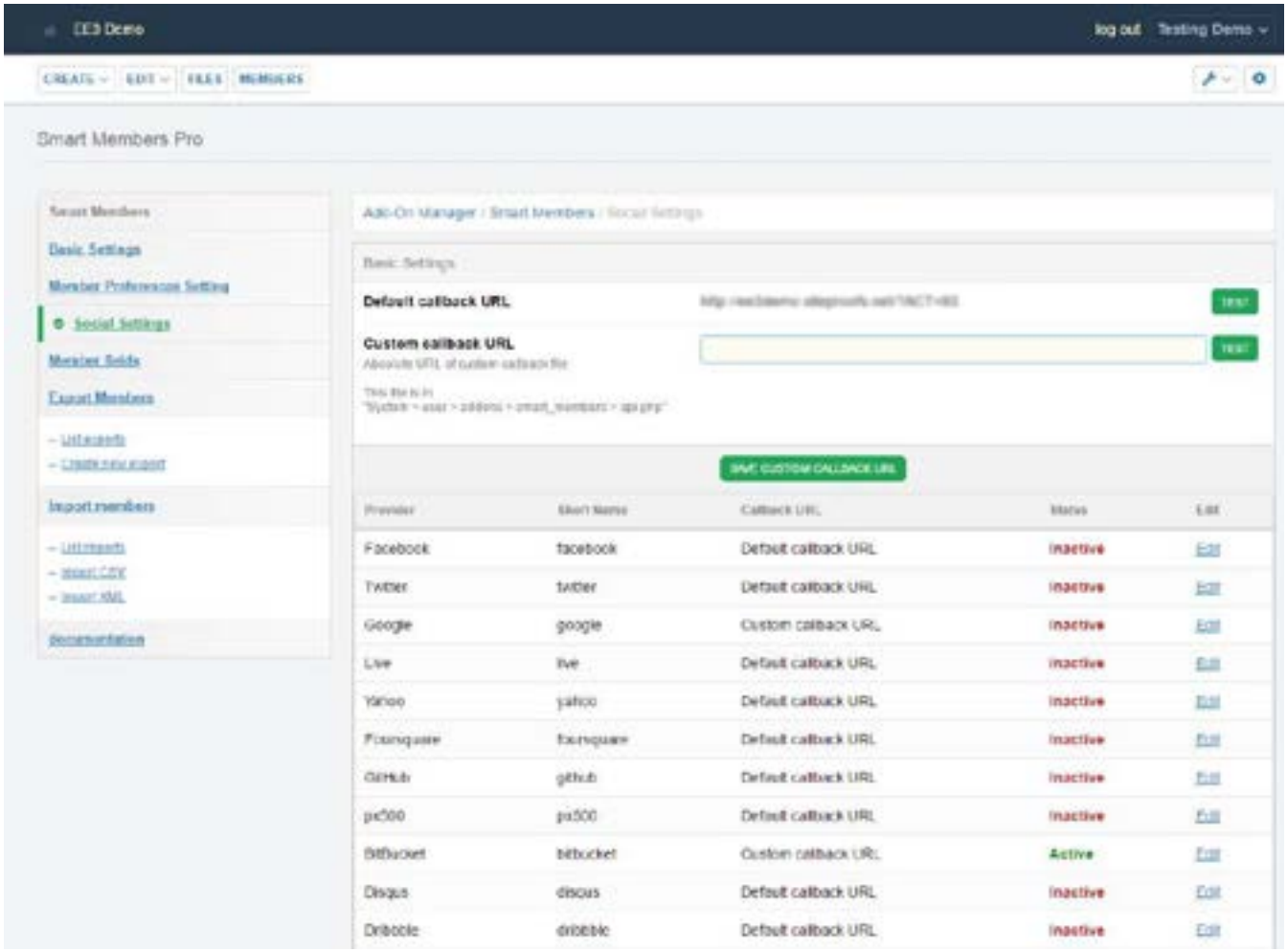
Member Preferences setting form



Social Settings

Social Login provides users to login with their social accounts. User can use login with Facebook, Twitter, Google and 20 more Social sites to login.

Social Settings List page



Every Social login API will ask for a callback URL. You can found a Callback URL at social Settings List page.

Callback URL:

Test the link is working or not by clicking TEST button at right. If link is not working please do check the link path is correct or not.

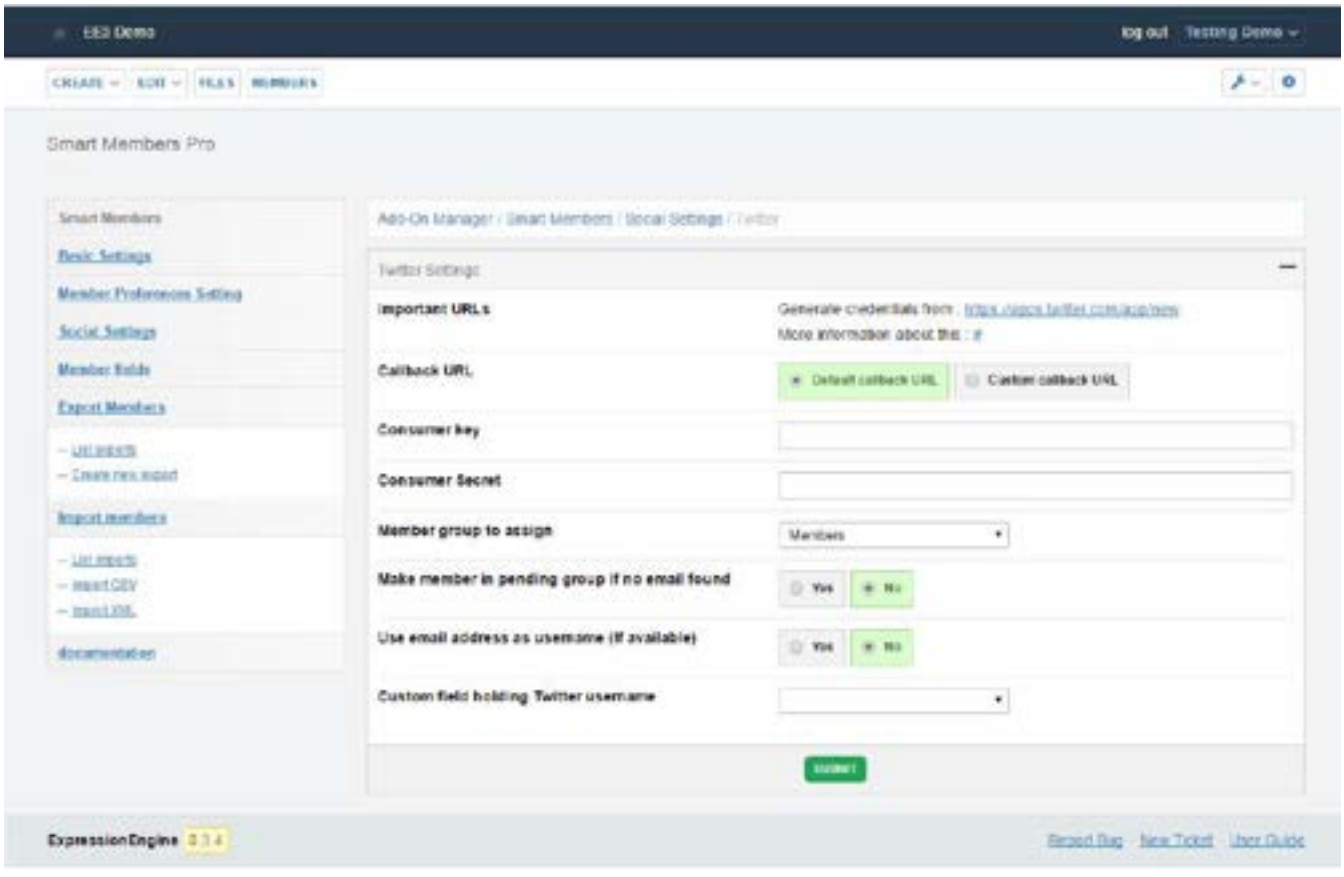
If Test goes success, You will see this page:

Smart members

Social Login API

API Working fine and ready to set as callback URL.

Social Settings Form:



Available Social sites:

- Facebook

- **Twitter**
- **Google**
- **Live**
- **Yahoo**
- **Foursquare**
- **GitHub**
- **px500**
- **BitBucket**
- **Disqus**
- **Dribbble**
- **Dropbox**
- **GitLab**
- **Instagram**
- **LastFM**
- **MailChimp**
- **Slack**
- **SoundCloud**
- **Vimeo**
- **Tumblr**

Facebook

To activate Facebook method in Social Login settings:

1. Generate Application ID and Application Secret From this link: https://developers.facebook.com/quickstarts/?platform=web
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need. Facebook API need exact callback URL. So you need to enter the callback URL with done parameter (EX: if your callback URL is

need to enter the callback URL with done parameter. (EX: if your callback URL is http://www.example.com/?ACT=85, Your callback URL for facebook will be http://www.example.com/?ACT=85&hauth.done=Facebook

3. Once you generate Application ID and Application Secret, Go to social settings page in Smart Members Subscription Pro and Edit Facebook settings.
4. Put your Application ID and Secret in their fields. Select the callback URL you have put in API.
5. Save the settings.

Twitter

To activate Twitter method in Social Login settings:

1. Generate Consumer key and Consumer Secret From this link: https://apps.twitter.com/app/new
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.
3. Once you generate Consumer key and Consumer Secret, Go to social settings page in Smart Members Subscription Pro and Edit Twitter settings.
4. Put your Consumer key and Consumer Secret in their fields. Select the callback URL you have put in API.
5. Save the settings.

Google

To activate Google method in Social Login settings:

1. Generate Client ID andClient Secret From this link: https://console.developers.google.com/
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need. Google API need exact callback URL. So you need to enter the callback URL with done parameter. (EX: if your callback URL is http://www.example.com/?ACT=85, Your callback URL for Google will be

http://www.example.com/?ACT=85&hauth.done=Google

3. Once you generate Client ID andClient Secret, Go to social settings page in Smart Members Subscription Pro and Edit Google settings.
4. Put your Client ID and Client Secret in their fields. Select the callback URL you have put in API.
5. Save the settings.

Live

To activate Live method in Social Login settings:

1. Generate Client ID and Client Secret From this link: https://apps.dev.microsoft.com/#/appList/create/sapi
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.
3. Once you generate Client ID and Client Secret, Go to social settings page in Smart Members Subscription Pro and Edit Live settings.
4. Put your Client ID and Client Secret in their fields. Select the callback URL you have put in API.
5. Save the settings.

Yahoo

To activate Yahoo method in Social Login settings:

1. Generate Client ID and Client Secret From this link: https://developer.yahoo.com/apps/create/
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.
3. Once you generate Client ID and Client Secret, Go to social settings page in Smart Members Subscription Pro and Edit Yahoo settings.
4. Put your Client ID and Client Secret in their fields. Select the callback URL you have put in API.

5. Save the settings.

Foursquare

To activate Foursquare method in Social Login settings:

1. Generate Client ID and Client Secret From this link: https://foursquare.com/developers/apps
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.
3. Once you generate Client ID and Client Secret, Go to social settings page in Smart Members Subscription Pro and Edit Foursquare settings.
4. Put your Client ID and Client Secret in their fields. Select the callback URL you have put in API.
5. Save the settings.

GitHub

To activate GitHub method in Social Login settings:

1. Generate Client ID and Client Secret From this link: https://github.com/settings/applications/new
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.
3. Once you generate Client ID and Client Secret, Go to social settings page in Smart Members Subscription Pro and Edit GitHub settings.
4. Put your Client ID and Client Secret in their fields. Select the callback URL you have put in API.
5. Save the settings.

px500

To activate px500 method in Social Login settings:

1. Generate Customer ID and Customer Secret From this link: https://500px.com/settings/applications
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.
3. Once you generate Customer ID and Customer Secret, Go to social settings page in Smart Members Subscription Pro and Edit px500 settings.
4. Put your Customer ID and Customer Secret in their fields. Select the callback URL you have put in API.
5. Save the settings.

BitBucket

To activate BitBucket method in Social Login settings:

1. Generate Customer Key and Customer Secret From this link: https://bitbucket.org/account/user/testing_eecms/api
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.
3. Once you generate Customer Key and Customer Secret, Go to social settings page in Smart Members Subscription Pro and Edit BitBucket settings.
4. Put your Customer Key and Customer Secret in their fields. Select the callback URL you have put in API.
5. Save the settings.

Disqus

To activate Disqus method in Social Login settings:

1. Generate Public Key and Secret Key From this link: https://disqus.com/api/applications/register/
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.

3. Once you generate Public Key and Secret Key, Go to social settings page in Smart Members Subscription Pro and Edit Disqus settings.
4. Put your Public Key and Secret Key in their fields. Select the callback URL you have put in API.
5. Save the settings.

Dribbble

To activate Dribbble method in Social Login settings:

1. Generate Client ID and Client Secret From this link: https://dribbble.com/account/applications/new
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.
3. Once you generate Client ID and Client Secret, Go to social settings page in Smart Members Subscription Pro and Edit Dribbble settings.
4. Put your Client ID and Client Secret in their fields. Select the callback URL you have put in API.
5. Save the settings.

Dropbox

To activate Dropbox method in Social Login settings:

1. Generate App Key and App Secret From this link: https://www.dropbox.com/developers/apps/create
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.
3. Once you generate App Key and App Secret, Go to social settings page in Smart Members Subscription Pro and Edit Dropbox settings.
4. Put your App Key and App Secret in their fields. Select the callback URL you have put in API.

5. Save the settings.

GitLab

To activate GitLab method in Social Login settings:

1. Generate Application ID and Application Secret From this link: https://gitlab.com/oauth/applications
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.
3. Once you generate Application ID and Application Secret, Go to social settings page in Smart Members Subscription Pro and Edit GitLab settings.
4. Put your Application ID and Application Secret in their fields. Select the callback URL you have put in API.
5. Save the settings.

Instagram

To activate Instagram method in Social Login settings:

1. Generate Client ID and Client Secret From this link: https://www.instagram.com/developer/clients/register/
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.
3. Once you generate Client ID and Client Secret, Go to social settings page in Smart Members Subscription Pro and Edit Instagram settings.
4. Put your Client ID and Client Secret in their fields. Select the callback URL you have put in API.
5. Save the settings.

LastFM

To activate LastFM method in Social Login settings:

1. Generate API key and Shared Secret From this link: http://www.last.fm/api/account/create
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.
3. Once you generate API key and Shared Secret, Go to social settings page in Smart Members Subscription Pro and Edit LastFM settings.
4. Put your API key and Shared Secret in their fields. Select the callback URL you have put in API.
5. Save the settings.

MailChimp

To activate MailChimp method in Social Login settings:

1. Generate Client ID and Client Secret From this link: https://us14.admin.mailchimp.com/account/oauth2/client/
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.
3. Once you generate Client ID and Client Secret, Go to social settings page in Smart Members Subscription Pro and Edit MailChimp settings.
4. Put your Client ID and Client Secret in their fields. Select the callback URL you have put in API.
5. Save the settings.

Slack

To activate Slack method in Social Login settings:

1. Generate Client ID and Client Secret From this link: https://api.slack.com/apps/new
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.

3. Once you generate Client ID and Client Secret, Go to social settings page in Smart Members Subscription Pro and Edit Slack settings.
4. Put your Client ID and Client Secret in their fields. Select the callback URL you have put in API.
5. Save the settings.

SoundCloud

To activate SoundCloud method in Social Login settings:

1. Generate Client ID and Client Secret From this link: http://soundcloud.com/you/apps/new
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.
3. Once you generate Client ID and Client Secret, Go to social settings page in Smart Members Subscription Pro and Edit SoundCloud settings.
4. Put your Client ID and Client Secret in their fields. Select the callback URL you have put in API.
5. Save the settings.

Vimeo

To activate Vimeo method in Social Login settings:

1. Generate Client ID and Client Secret From this link: https://developer.vimeo.com/apps/new
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.
3. Once you generate Client ID and Client Secret, Go to social settings page in Smart Members Subscription Pro and Edit Vimeo settings.
4. Put your Client ID and Client Secret in their fields. Select the callback URL you have put in API.

5. Save the settings.

Tumblr

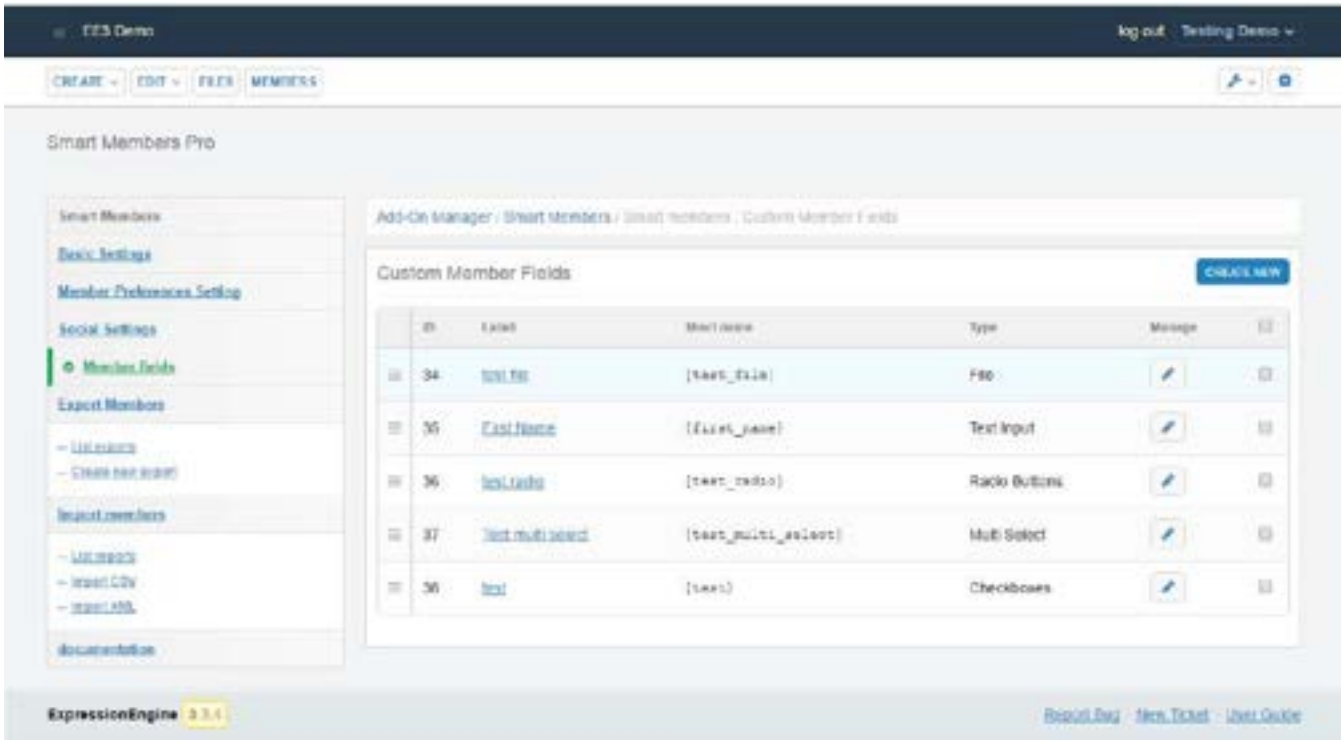
To activate Tumblr method in Social Login settings:

1. Generate OAuth consumer key and OAuth consumer secret From this link: https://www.tumblr.com/oauth/register
2. Enter Callback URL in API form. You can enter Default callback URL or Custom Callback URL as per your need.
3. Once you generate OAuth consumer key and OAuth consumer secret, Go to social settings page in Smart Members Subscription Pro and Edit Tumblr settings.
4. Put your OAuth consumer key and OAuth consumer secret in their fields. Select the callback URL you have put in API.
5. Save the settings.

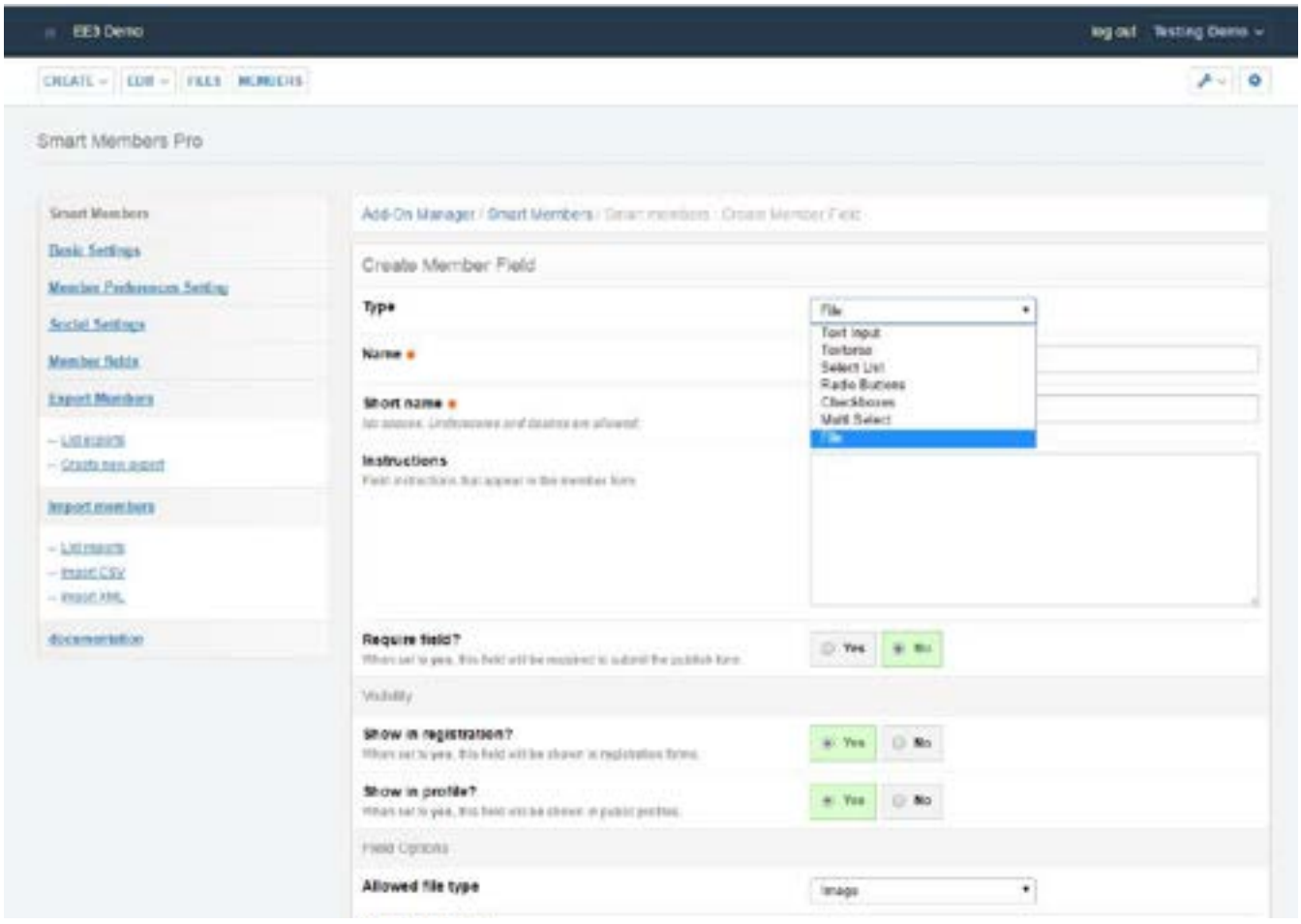
Member Fields

ExpressionEngine has limit for member field types. One can only select Text Input, Textarea and select dropdown as custom member fields. We introduce Multi select box, Radio buttons, Checkboxes and File fields to use as custom member fields.

Member Fields setting List Page:



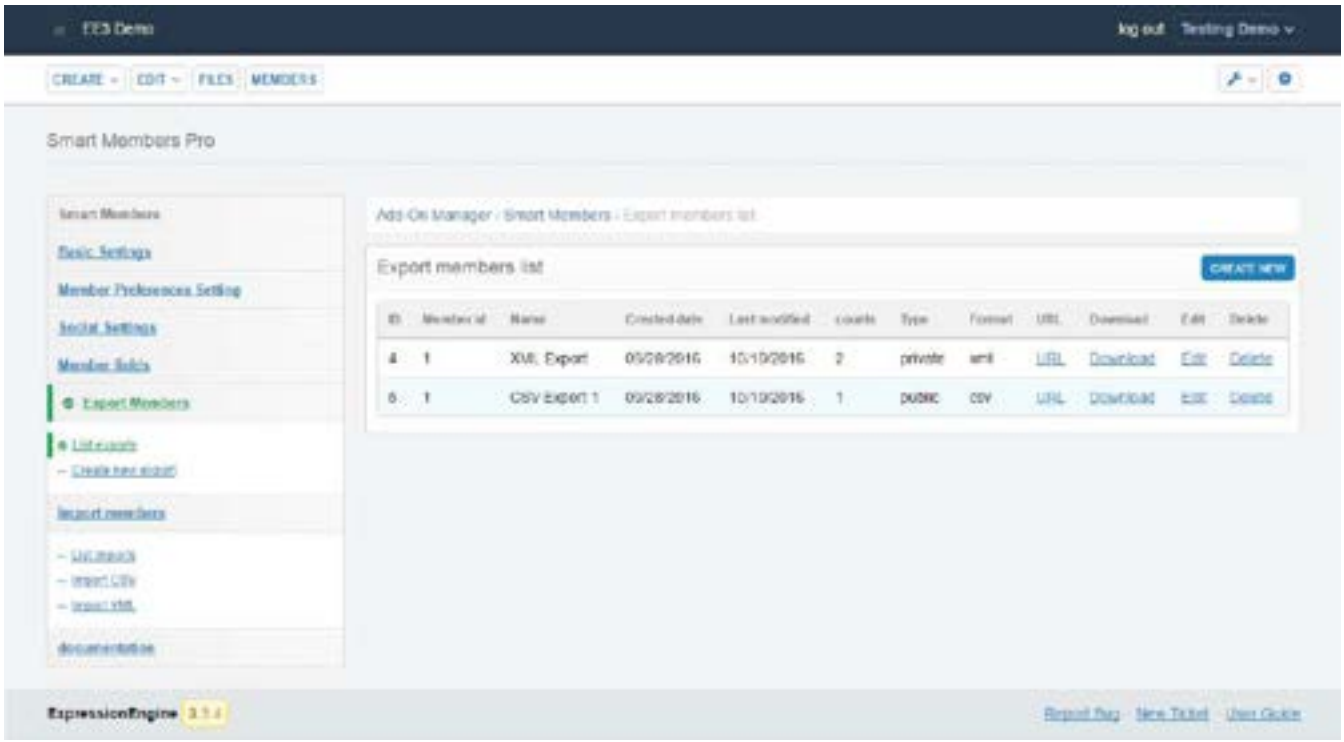
Create Field form:



Export Members

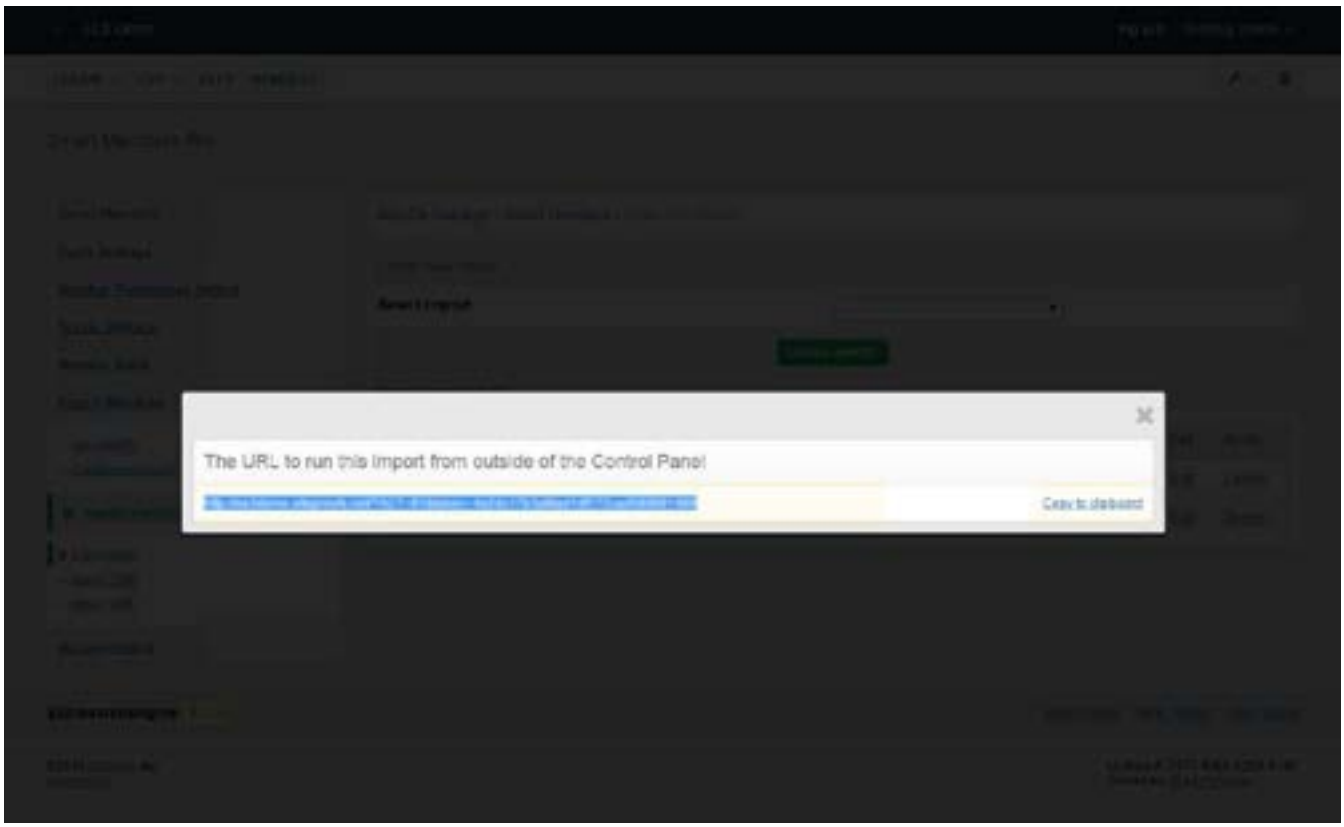
You can now export members with selected fields filtered by Groups. Exports are available to download in both CSV and XML format. You can save the exports to use it in future. Export can also made by out side of admin panel with auto generated links.

Export member List page:

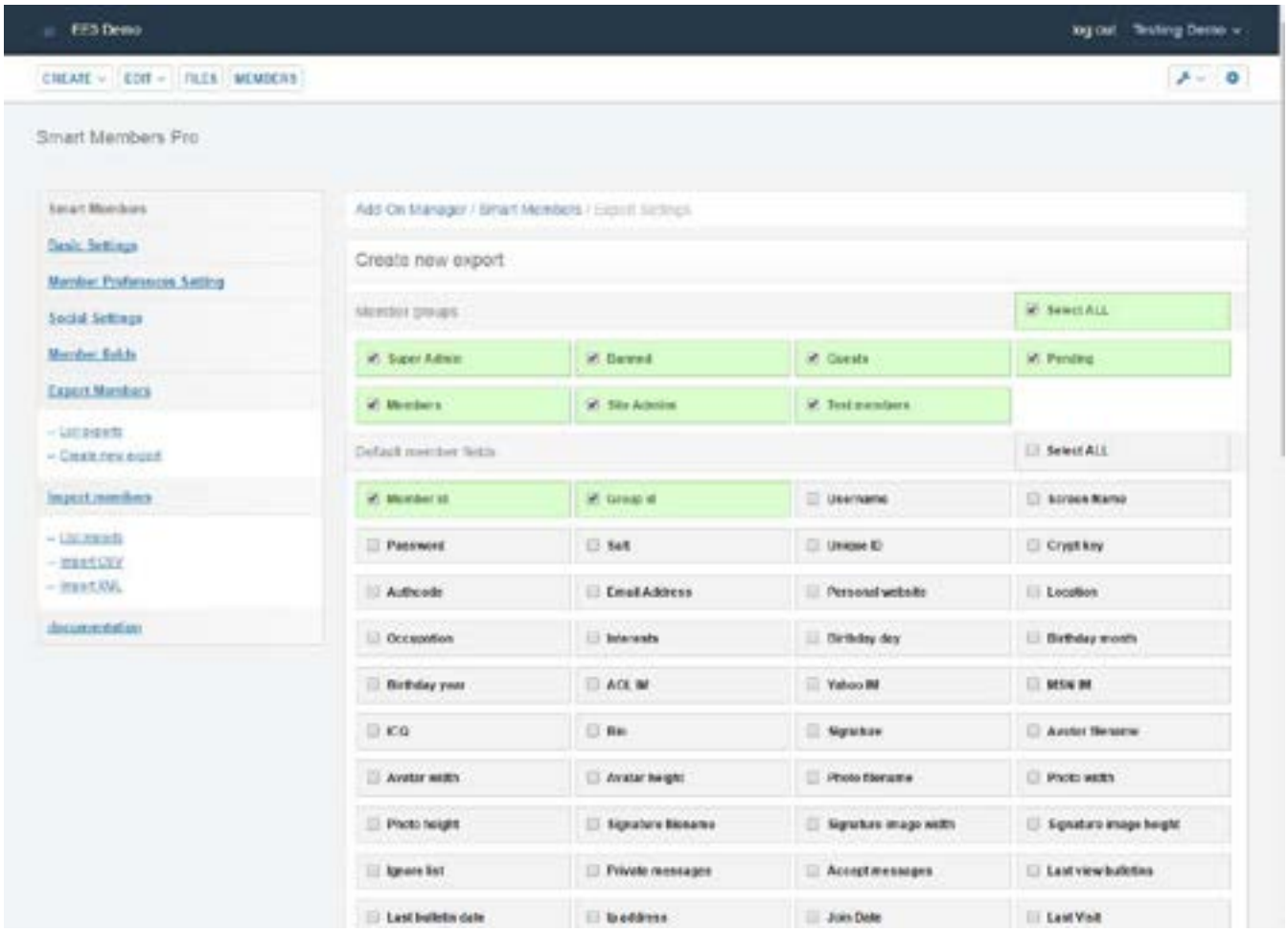


You can download, edit or Delete the exports from here. URL link is will popup the external URL to download export from outsite of EE. There is some security to download exports from outside of EE. If you make export private, No once can download it without login with your account. Public exports can download from outside of EE with anyone who logged in. If you want to download export from outside of EE without login, set “Access export URL without Login ?” to YES.

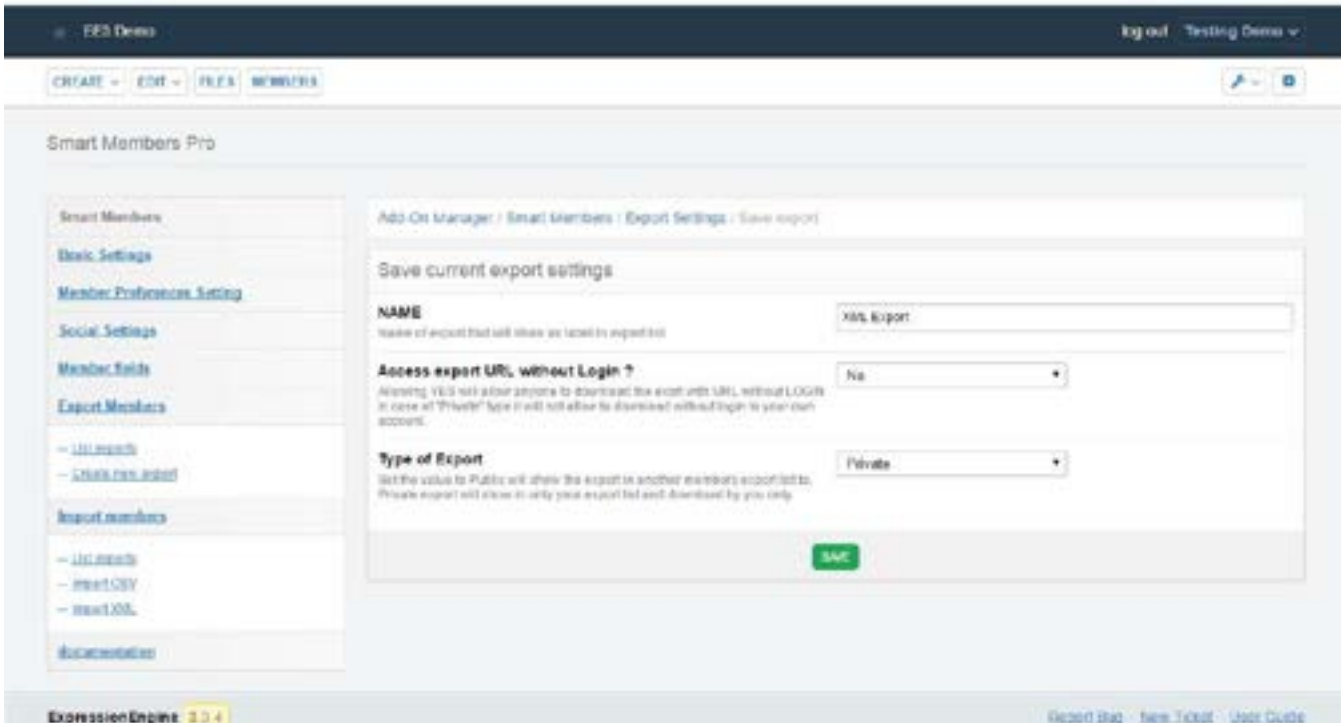
Popup URL to download export outside of EE:



Generate Export Form 1:



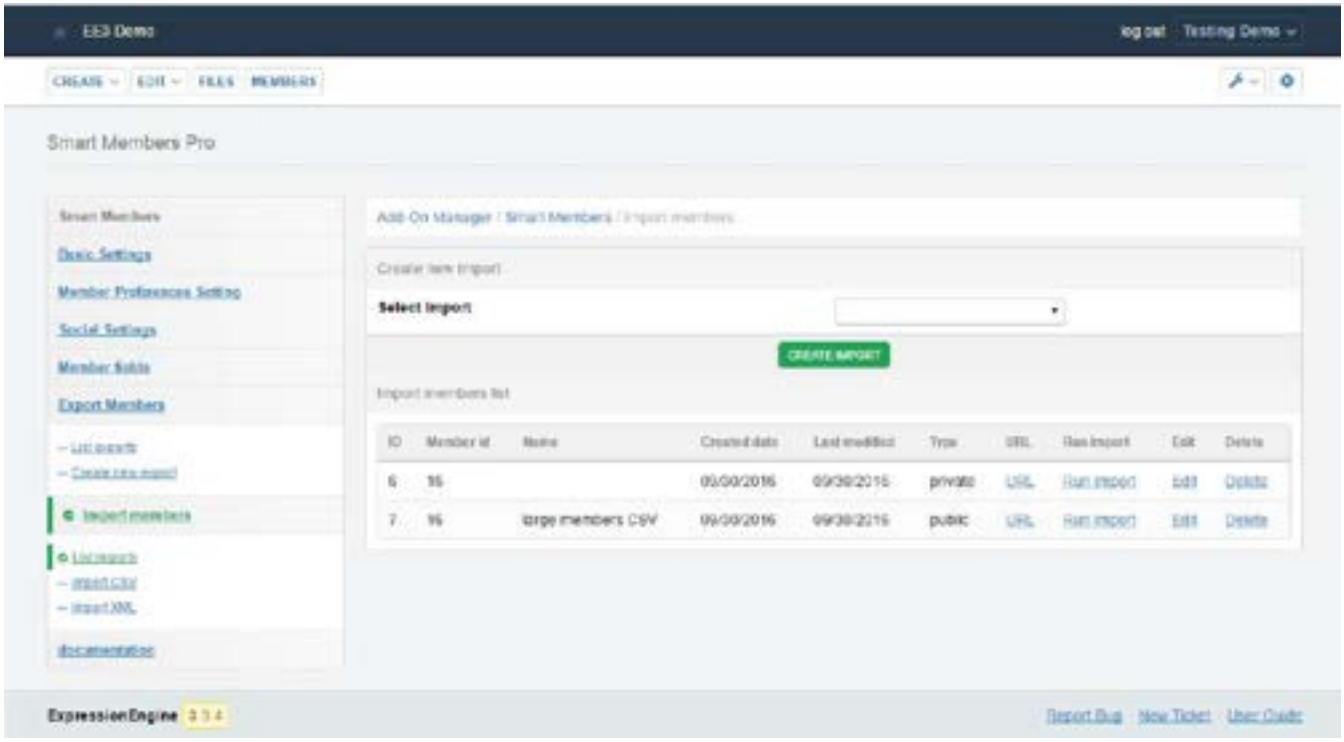
Generate Export Form 2:



Import Members

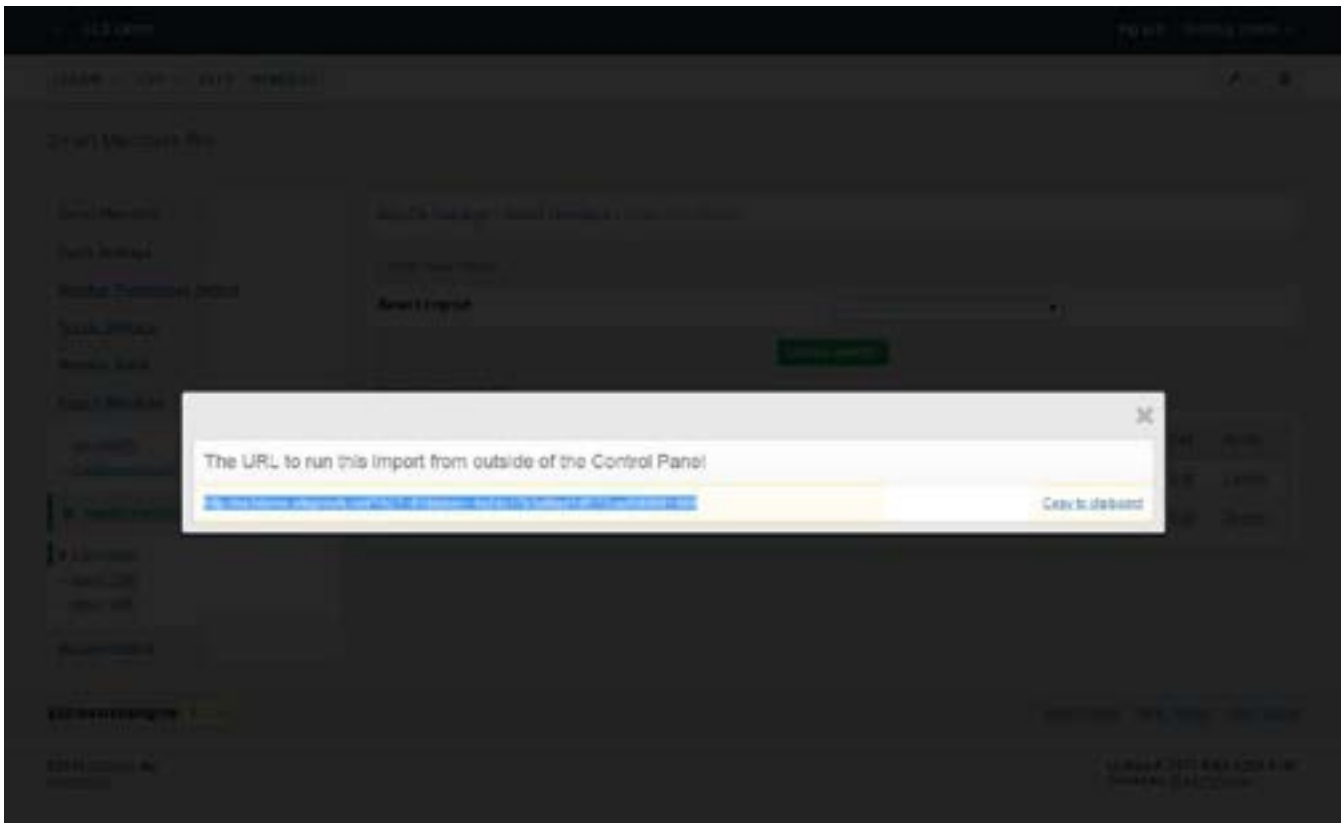
One can import members with selected fields with filtration. Filtration is done on many basis such as same email, same member ID, same screen name, Sanitize the unique fields or not etc. Import can done from both CSV and XML files at path of both server path or URL. You can save the imports to use it in future. Imports can also done by out side of admin panel with auto generated links.

Import Members List page:

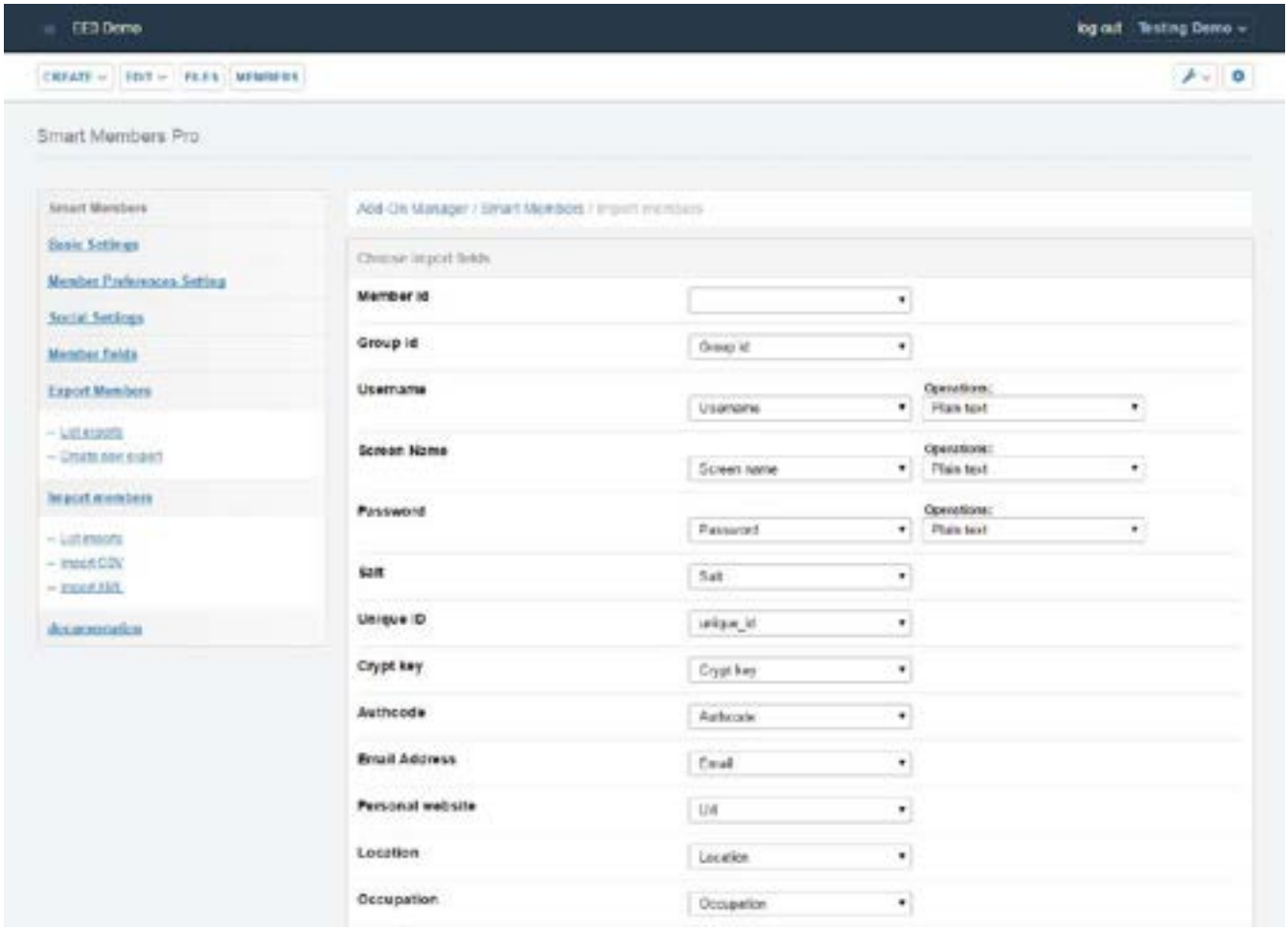


You can run, edit or Delete the imports from here. URL link is will popup the external URL to run export from outsite of EE. There is some security to run imports from outside of EE. If you make import private, No once can run it without login with your account. Public imports can run from outside of EE with anyone who logged in. If you want to run import from outside of EE without login, set “Access import URL without Login ?” to YES.

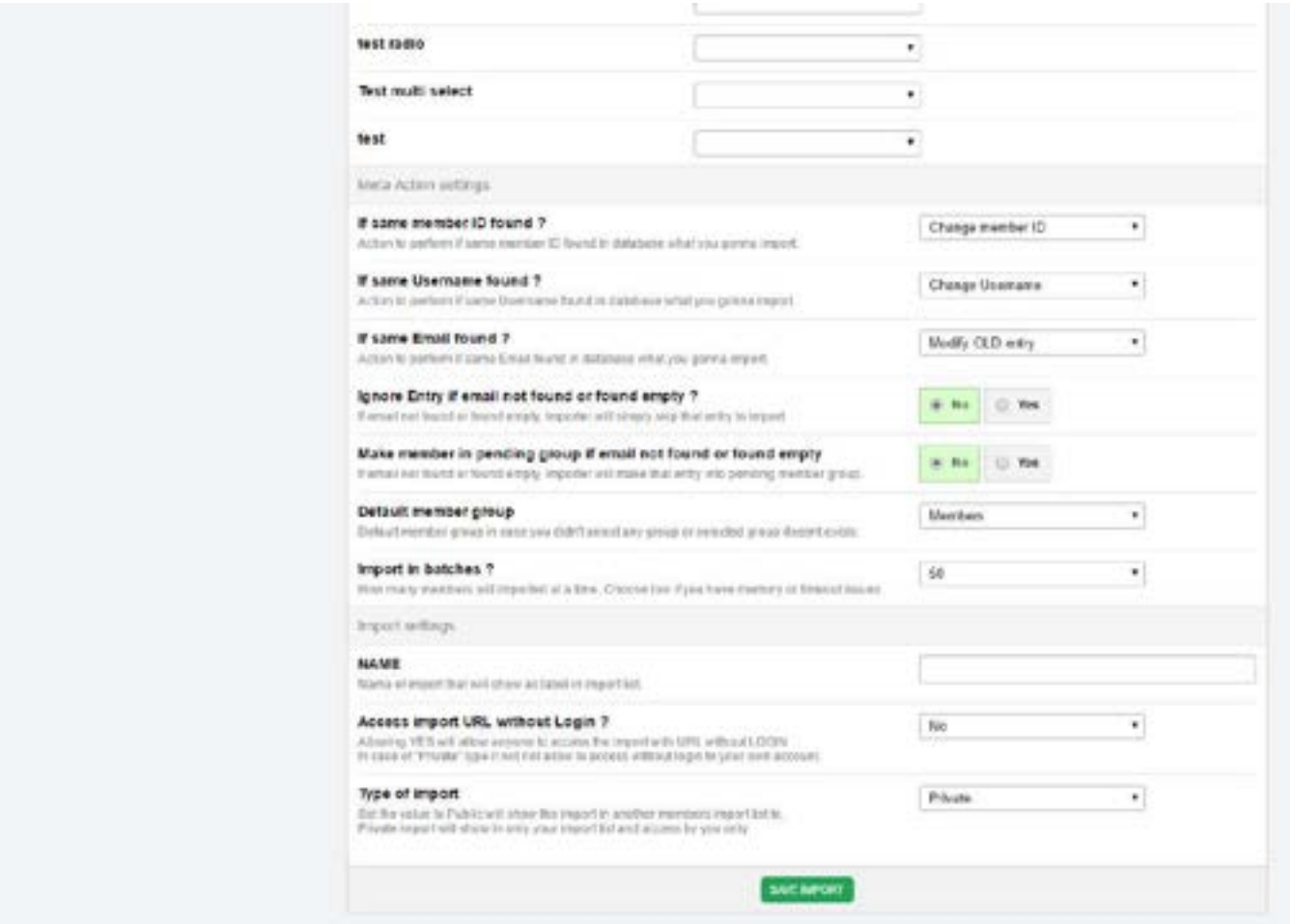
Popup URL to run import outside of EE:



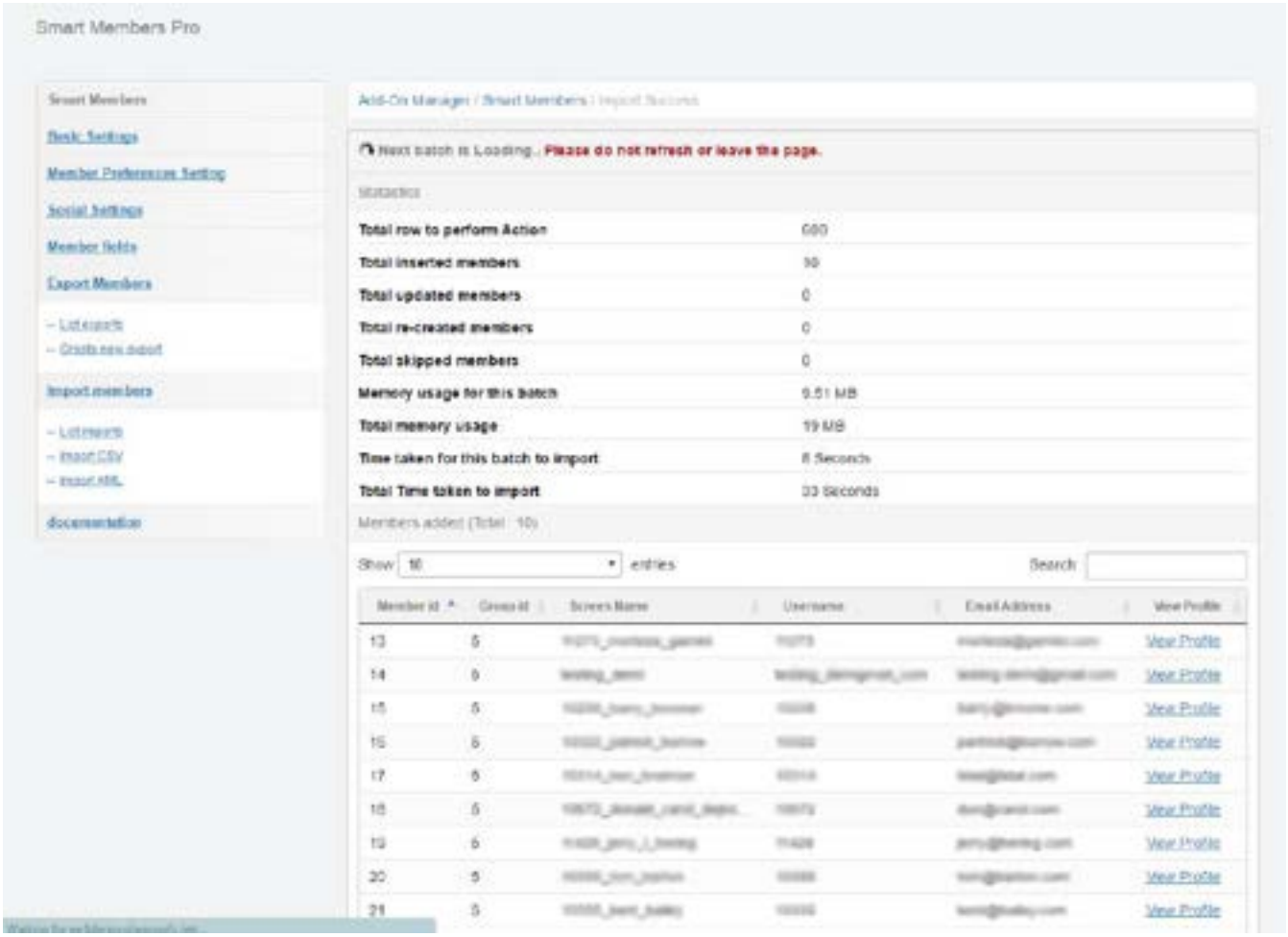
Import Member Choose fields:



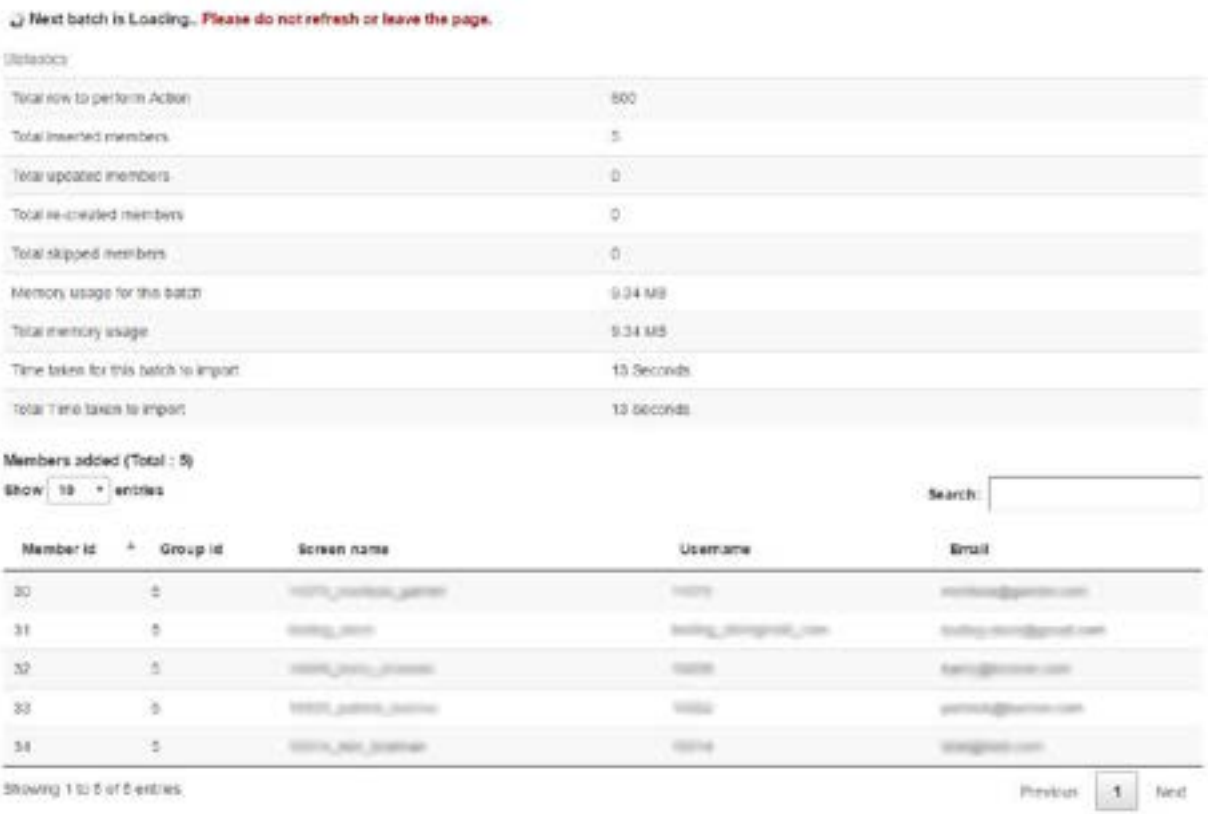
Import Member setup Configurations:



Run Import page inside EE:



Run Import page outside EE:



Registration

Registration module is use for register a user with your site. A registration module will process and react as per settings we have passed in member preferences and plugin setting form.

Tag for registration module will look like this.

```
{exp:smsp:register} Content data {/exp:smsp:register}
```

Parameters

1. group_id

2. allowed_groups
3. rule:FIELD_NAME
4. attr:ATTRIBUTES
5. return
6. error_reporting
7. wrap_errors
8. on_submit
9. secure_action
10. secure_return
11. enable_recaptcha

Fields

1. group_id
2. username
3. email
4. password
5. password_confirm
6. avatar_filename
7. photo_filename
8. sig_img_filename
9. yahoo_im
10. url
11. location
12. occupation
13. interests

14. aol_im
15. msn_im
16. icq
17. bio
18. signature
19. captcha
20. CUSTOM_FIELD
21. CUSTOM_CHECKBOX_FIELD **(Pro Feature)**
22. CUSTOM_RADIO_FIELD **(Pro Feature)**
23. CUSTOM_MULTI_SELECT_FIELD **(Pro Feature)**
24. CUSTOM_FILE_FIELD **(Pro Feature)**

Following Parameters can be use in Registration form

group_id

Group id of member groups to put the member in any specific group.

Example:

```
group_id = "5"
```

allowed_groups

Allowed groups parameter is used to give options to member for select any group from them. Use input field of group_id to enter the group id from defined allowed groups.

Example:

```
allowed_groups = "5|6|7"
```

rule:FIELD_NAME

Rule parameter is use to give custom rules to fields separate by pipe (|).

Example:

```
rule:username = “required”
rule:email = “required|valid_email|is_unique[members.email]”
```

attr:ATTRIBUTES

Parameter to add attributes in form. We can add classes, ids etc.

Example:

```
attr:id = “form_id”
attr:class = “form_class”
attr:name = “form_name”
attr:data-id = “form_data_id”
```

return

Return to any specific page after successful submission of form.

Example:

```
return = “smart-members/profile”
```

error_reporting

Error reporting format is defined by this parameter. It can be either “inline” or “outline”.

“Inline” error reporting will show the error in same page.
“outline” error reporting will show error in EE gray box in new page.

Example:

```
error_reporting=”inline”
```

wrap_errors

Use this parameter to wrap forms error in any span or div if set error_reporting="inline".

Example:

```
wrap_errors="<span class='error-inline'>|</span>"
```

on_submit

This parameter allows us to call any Javascript function on submit of form

Example:

```
on_submit="call_me( )"
```

secure_action

Secure action will post the data on secure site i.e., https

Example:

```
secure_action="yes"
```

secure_return

Secure return will return the page after submit of form on secure site i.e., https

Example:

```
secure_return="yes"
```

enable_recaptcha

This parameter enables recaptcha instead of normal captcha.

a. Scenario EE2:

i. If this parameter is set and API key and SECRET is not passed for recaptcha in backend, The normal captcha will show.

ii. If recaptcha API key and SECRET is passed in backend and this parameter not set:

1. If the page is registration page and you have set captcha as required from member preferences, normal captcha will show.
2. If the page is not registration page, neither captcha or recaptcha will show.

iii. If recaptcha API key and SECRET is passed in backend and this parameter is set:

1. If the page is registration page and you have set captcha as required from member preferences, The normal captcha will not show and recaptcha will override the settings.
2. If the page is not registration page, recaptcha will show.

b. Scenario EE3:

i. Same scenario like EE2. Only change is, there is backend member preference settings in EE3 that allows to not enter any captcha if member is logged in. So if you set this parameter and API key and SECRET is also passed, If you are logged in this recaptcha or captcha will not show until you set “Require CAPTCHA while logged in?” to “Yes” from backend member settings > captcha settings.

Example:

```
enable_recaptcha=”yes”
```

Different methods that can be use by user are given below.

Input

```
<input type=”email” name=”username” placeholder=”Email Addr
```

```
ess”>
{error:username} //if error_reporting=”inline”
```

Textarea

```
<textarea name=”signature”></textarea>
{error:signature} //if error_reporting=”inline”
```

Default File field (Avatar, Photo and Signature files)

```
<input type=”file” name=”avatar_filename” />
{error:avatar_filename} //if error_reporting=”inline”
```

Select Dropdown: (For member group)

```
{data_group_id}
{if data_group_id:count == 1}
<select name=”group_id”>
<option value=””></option>
{/if}
<option value = “{group_id_value}” > {group_id_label}</option>
{if data_group_id:count == data_group_id:total_results}
</select>
{error:group_id} //if error_reporting=”inline”
{/if}
{/data_group_id}
```

Select Dropdown: (For custom field)

```
{data_CUSTOM_FIELD}
{if data_CUSTOM_FIELD:count == 1}
<select name="CUSTOM_FIELD">
<option value=""></option>
{/if}
<option value = "{CUSTOM_FIELD_value}"> {CUSTOM_FIELD_label}
</option>
{if data_CUSTOM_FIELD:count == data_CUSTOM_FIELD:total_result
s}
</select>
{error:CUSTOM_FIELD} //if error_reporting="inline"
{/if}
{/data_CUSTOM_FIELD}
```

Captcha

```
{if captcha}
<p>
<label for="captcha">Please enter in the word you see:</label>
{captcha}
<input type="text" name="captcha" id="captcha" />
{error:captcha}
</p>
{if:elseif recaptcha}
<p style="margin: 0;">
```

```
<p style="margin: 0;

<label for="recaptcha">Click the checkbox</label>{recaptcha}
{error:recaptcha}
</p>
{/if}
```

Accept terms

```
<input type="checkbox" name="accept_terms" value="y" /> Acce
pt Terms?
{error:accept_terms} //if error_reporting="inline"
```

CUSTOM_CHECKBOX_FIELD (Pro Feature)

```
{if data_CUSTOM_CHECKBOX_FIELD:count == 1}
<p style="display: inline-flex; ">
{/if}
<input type="checkbox" name="CUSTOM_CHECKBOX_FIELD[]" val
ue="{CUSTOM_CHECKBOX_FIELD_value}" id="test_{data_CUSTOM
_CHECKBOX_FIELD:count}" {if CUSTOM_CHECKBOX_FIELD_value:ex
ists}checked{/if}>
<label for="test_{data_CUSTOM_CHECKBOX_FIELD:count}">{CUST
OM_CHECKBOX_FIELD_label}</label>
{if data_CUSTOM_CHECKBOX_FIELD:count == data_CUSTOM_CHE
CKBOX_FIELD:total_results}
{error:CUSTOM_CHECKBOX_FIELD}
</p>
```

```
</p>

{/if}
{/data_CUSTOM_CHECKBOX_FIELD}
```

CUSTOM_RADIO_FIELD (Pro Feature)

```
{data_CUSTOM_RADIO_FIELD}
{if data_CUSTOM_RADIO_FIELD:count == 1}
<p style="display: inline-flex;">
{/if}
<input type="radio" name="CUSTOM_RADIO_FIELD" value="{CUSTOM_RADIO_FIELD_value}" id="test_{data_CUSTOM_RADIO_FIELD:count}" {if CUSTOM_RADIO_FIELD_value:exists}checked{/if}>
<label for="test_{data_CUSTOM_RADIO_FIELD:count}">{CUSTOM_RADIO_FIELD_label}</label>
{if data_CUSTOM_RADIO_FIELD:count == data_CUSTOM_RADIO_FIELD:total_results}
{error:CUSTOM_RADIO_FIELD}
</p>
{/if}
{/data_CUSTOM_RADIO_FIELD}
```

CUSTOM_MULTI_SELECT_FIELD (Pro Feature)

```
{data_CUSTOM_MULTI_SELECT_FIELD}
{if data_CUSTOM_MULTI_SELECT_FIELD:count == 1}
<p style="display: inline-flex; ">
```

```
<p style="display: inline-flex;">

<select multiple name="CUSTOM_MULTI_SELECT_FIELD[]">
{/if}
<option value="{CUSTOM_MULTI_SELECT_FIELD_value}" {if CUSTO
M_MULTI_SELECT_FIELD_value:exists}selected{/if}>{CUSTOM_MULT
I_SELECT_FIELD_label}</option>
{if data_CUSTOM_MULTI_SELECT_FIELD:count == data_CUSTOM_
MULTI_SELECT_FIELD:total_results}
</select>
{error:CUSTOM_MULTI_SELECT_FIELD}
</p>
{/if}
{/data_CUSTOM_MULTI_SELECT_FIELD}
```

CUSTOM_FILE_FIELD (Pro Feature)

```
<p>
Old file name: {CUSTOM_FILE_FIELD}<br>
<input type="file" name="CUSTOM_FILE_FIELD">
{error:CUSTOM_FILE_FIELD}
</p>
```

Example:

```
{exp:smsp:register
allowed_groups="6|7|5"rule:username="required|valid_email|min_l
ength[5]"
```

```
ength[5]

rule:password="required|matches[password_confirm]|min_length
[5]"
rule:password_confirm="required|min_length[5]"
rule:group_id="required"
rule:state="required"
rule:first_name="required"attr:id="registration_id"
attr:class="registration_class"
attr:name="registration-form"
attr:data-id="registration_data_id_attr"return="smart-members/e
dit-profile"
enable_recaptcha="yes"
error_reporting="inline"
wrap_errors="<span class='error-inline'>|</span>"on_submit="re
gistration()"
}
<p>
<input type="text" name="CUSTOM_FIELD" placeholder="CUSTO
M FIELD">
{error:CUSTOM_FIELD}
</p>
{data_group_id}
{if data_group_id:count == 1}
<p>
<select name="group_id">
```

```
<option value=""></option>
{/if}
<option value="{group_id_value}">{group_id_label}</option>
{if data_group_id:count == data_group_id:total_results}
</select>
{error:group_id}
</p>
{/if}
{/data_group_id}
<p>
<input type="text" name="username" placeholder="Username">
{error:username}
</p>
<p>
<input type="text" name="email" placeholder="Email Address">
{error:email}
</p><p>
<input type="password" name="password" placeholder="Passwo
rd">
{error:password}
</p>
<p>
<input type="password" name="password_confirm">
{error:password_confirm}
```

```
</p><p>
<input type="file" name="avatar_filename" />
{error:avatar_filename}
</p>
{if captcha}
<p>
<label for="captcha">Please enter in the word you see:</label>
{captcha}
<input type="text" name="captcha" id="captcha" />
{error:captcha}
</p>
{if:elseif recaptcha}
<p style="margin: 0;">
<label for="recaptcha">Click the checkbox</label>
{recaptcha}
{error:recaptcha}
</p>
{/if}
<p>
<input type="checkbox" name="accept_terms" value="y" /> Acce
pt Terms?
{error:accept_terms}
</p>
<div class="register-register-click cf">
```

```
<input type="submit" class="register" value="Register">
</div>
{/exp:smsp:register}
```

Login

Login module is use for login the registered member. ExpressionEngine also provides default Login module. We are giving the Login module to provide the additional features the plugin provides such as Inline error reporting, Recaptcha, Login with wither Username or Email address etc.

Tag for Login module will look like this.

```
{exp:smsp:login} Content data {/exp:smsp:login}
```

Parameters

1. allowed_groups
2. rule:FIELD_NAME
3. attr:ATTRIBUTES
4. return
5. error_reporting
6. wrap_errors
7. on_submit
8. secure_action
9. secure_return

10. enable_recaptcha

Fields

1. group_id
2. username
3. email
4. password
5. auto_login
6. captcha

Following Parameters can be use in Login form

allowed_groups

Allowed groups parameter is used to give options to member for select any group from them. Use input field of group_id to enter the group id from defined allowed groups.

Example:

```
allowed_groups = "5|6|7"
```

rule:FIELD_NAME

Rule parameter is use to give custom rules to fields separate by pipe (|).

Example:

```
rule:username = "required"
rule:email = "required|valid_email|is_unique[members.email]"
```

attr:ATTRIBUTES

Parameter to add attributes in form. We can add classes, ids etc.

Example:

Example:

```
attr:id = “form_id”
attr:class = “form_class”
attr:name = “form_name”
attr:data-id = “form_data_id”
```

return

Return to any specific page after successful submission of form.

Example:

```
return = “smart-members/profile”
```

error_reporting

Error reporting format is defined by this parameter. It can be either “inline” or “outline”.

“Inline” error reporting will show the error in same page.
“outline” error reporting will show error in EE gray box in new page.

Example:

```
error_reporting=”inline”
```

wrap_errors

Use this parameter to wrap forms error in any span or div if set error_reporting=”inline”.

Example:

```
wrap_errors=”<span class=’error-inline’>|</span>”
```

on_submit

This parameter allows us to call any Javascript function on submit of form

Example:

```
on_submit="call_me( )"
```

secure_action

Secure action will post the data on secure site i.e., https

Example:

```
secure_action="yes"
```

secure_return

Secure return will return the page after submit of form on secure site i.e., https

Example:

```
secure_return="yes"
```

enable_recaptcha

This parameter enables recaptcha instead of normal captcha.

a. Scenario EE2:
    i. If this parameter is set and API key and SECRET is not passed for recaptcha in backend, The normal captcha will show.
    ii. If recaptcha API key and SECRET is passed in backend and this parameter not set:
        1. If the page is registration page and you have set captcha as required from member preferences, normal captcha will show.
        2. If the page is not registration page, neither captcha or recaptcha will show.

iii. If recaptcha API key and SECRET is passed in backend and this parameter is set:

1. If the page is registration page and you have set captcha as required from member preferences, The normal captcha will not show and recaptcha will override the settings.
2. If the page is not registration page, recaptcha will show.

b. Scenario EE3:

i. Same scenario like EE2. Only change is, there is backend member preference settings in EE3 that allows to not enter any captcha if member is logged in. So if you set this parameter and API key and SECRET is also passed, If you are logged in this recaptcha or captcha will not show until you set “Require CAPTCHA while logged in?” to “Yes” from backend member settings > captcha settings.

Example:

```
enable_recaptcha=”yes”
```

Different methods that can be use by user are given below.

Input

```
<input type=”email” name=”username” placeholder=”Email Addr
ess”>
{error:username} //if error_reporting=”inline”
```

Select Dropdown: (For member group)

```
{data_group_id}
{if data_group_id:count == 1}
<select name=”group_id”>
```

```
<option value=""></option>
{/if}
<option value = "{group_id_value}" > {group_id_label}</option>
{if data_group_id:count == data_group_id:total_results}
</select>
{error:group_id} //if error_reporting="inline"
{/if}
{/data_group_id}
```

Captcha

```
{if captcha}
<p>
<label for="captcha">Please enter in the word you see:</label>
{captcha}
<input type="text" name="captcha" id="captcha" />
{error:captcha}
</p>
{if:elseif recaptcha}
<p style="margin: 0;">
<label for="recaptcha">Click the checkbox</label>{recaptcha}
{error:recaptcha}
</p>
{/if}
```

Auto Login

```
<input type="checkbox" name="auto_login" value="y" /> Remember me?
{error:auto_login} //if error_reporting="inline"
```

Example:

```
{exp:smsp:login
rule:username="required|min_length[5]"
rule:password="required"allowed_groups="1|5|7"attr:id="login_id"
attr:class="login_class"
attr:name="login-form"
attr:data-id="login_datta_id_attr"return="/smart-members/edit-profile"error_reporting="inline"
wrap_errors="<span class='error-inline'>|</span>"enable_recaptcha="yes"on_submit="login()"
}
{data_group_id}
{if data_group_id:count == 1}
<p>
<select name="group_id">
<option value=""></option>
{/if}
<option value="{group_id_value}">{group_id_label}</option>
{if data_group_id:count == data_group_id:total_results}
```

```
</select>
{error:group_id}
</p>
{/if}
{/data_group_id}<p>
<input type="text" name="username" class="text required validat
e[required]">
{error:username}
</p>
<p>
<input type="password" name="password" class="password text
required validate[required]">
{error:password}
</p>
{if captcha}
<p>
<label for="captcha">Please enter in the word you see:</label>
{captcha}
<input type="text" name="captcha" id="captcha" />
{error:captcha}
</p>
{if:elseif recaptcha}
<p style="margin: 0;">
<label for="recaptcha">Click the checkbox to complete the captc
ha process</label>
```

```
{recaptcha}
{error:recaptcha}
</p>
{/if}
<label for="auto_login">
<input type="checkbox" name="auto_login" id="auto_login" valu
e="y" /> Auto Login
{error:auto_login}
</label>
<div class="login-login-click cf">
<input type="submit" class="login-popup-btn login" value="Logi
n" >
</div>
{/exp:smsp:login}
```

Social Login

Social Login module is use for login/register a member with your site. A Social login module will react as per settings you have made in backend.

Tag for Social Login module will look like this.

```
{exp:smsp:social_login} Content data {/exp:smsp:social_login}
```

Parameters

1. rule:FIELD_NAME
2. attr:ATTRIBUTES
3. return
4. no_email_return
5. error_reporting
6. wrap_errors
7. on_submit
8. secure_action
9. secure_return
10. enable_recaptcha
11. popup
12. remember_me
13. providers

Fields

1. providers

Following Parameters can be use in Social Login form

rule:FIELD_NAME

Rule parameter is use to give custom rules to fields separate by pipe (|).

Example:

```
rule:username = “required”
rule:email = “required|valid_email|is_unique[members.email]”
```

attr:ATTRIBUTES

Parameter to add attributes in form. We can add classes, ids etc.

Example:

```
attr:id = “form_id”
attr:class = “form_class”
attr:name = “form_name”
attr:data-id = “form_data_id”
```

return

Return to any specific page after successful submission of form.

Example:

```
return = “smart-members/profile”
```

no_email_return

Return to a decided page if Social API not return email address of user.

Example:

```
no_email_return = “smart-members/member-form”
```

error_reporting

Error reporting format is defined by this parameter. It can be either “inline” or “outline”.

“Inline” error reporting will show the error in same page.
“outline” error reporting will show error in EE gray box in new page.

Example:

```
error_reporting="inline"
```

wrap_errors

Use this parameter to wrap forms error in any span or div if set error_reporting="inline".

Example:

```
wrap_errors="<span class='error-inline'>|</span>"
```

on_submit

This parameter allows us to call any Javascript function on submit of form

Example:

```
on_submit="call_me( )"
```

secure_action

Secure action will post the data on secure site i.e., https

Example:

```
secure_action="yes"
```

secure_return

Secure return will return the page after submit of form on secure site i.e., https

Example:

```
secure_return="yes"
```

enable_recaptcha

This parameter enables recaptcha instead of normal captcha.

a. Scenario EE2:

i. If this parameter is set and API key and SECRET is not passed for recaptcha in backend, The normal captcha will show.

ii. If recaptcha API key and SECRET is passed in backend and this parameter not set:

1. If the page is registration page and you have set captcha as required from member preferences, normal captcha will show.

2. If the page is not registration page, neither captcha or recaptcha will show.

iii. If recaptcha API key and SECRET is passed in backend and this parameter is set:

1. If the page is registration page and you have set captcha as required from member preferences, The normal captcha will not show and recaptcha will override the settings.

2. If the page is not registration page, recaptcha will show.

b. Scenario EE3:

i. Same scenario like EE2. Only change is, there is backend member preference settings in EE3 that allows to not enter any captcha if member is logged in. So if you set this parameter and API key and SECRET is also passed, If you are logged in this recaptcha or captcha will not show until you set “Require CAPTCHA while logged in?” to “Yes” from backend member settings > captcha settings.

Example:

```
enable_recaptcha=”yes”
```

popup

Popup parameter will decide your request will sent to social API from same page or send in popup

Example:

```
popup="yes"
```

remember_me

remember_me parameter will allow user to save session for long to remember the user so user can avoid login again and again.

Example:

```
remember_me="yes"
```

providers

providers parameter will decide the provider you want to show in providers list to allow user to login with. You can ignore the paramter if you want to let allow all the active providers to allow to login with.

Example:

```
providers = "facebook|twitter|google"
```

Different methods that can be use by user are given below.

Select Dropdown (For Providers listing)

```
{providers}
{if providers:count == 1}
<p>
<select name="providers">
<option value=""></option>
{/if}
<option value="{provider_name}">{provider_label}</option>
```

```
{if providers:count == providers:total_results}
</select>
{error:providers}
</p>
{/if}
{/providers}
```

Captcha

```
{if captcha}
<p>
<label for="captcha">Please enter in the word you see:</label>
{captcha}
<input type="text" name="captcha" id="captcha" />
{error:captcha}
</p>
{if:elseif recaptcha}
<p style="margin: 0;">
<label for="recaptcha">Click the checkbox</label>{recaptcha}
{error:recaptcha}
</p>
{/if}
```

Example:

```
{exp:smsp:social_login
```

```
providers = “facebook|twitter|google”attr:id = “sl_id”
attr:class = “sl_class”
attr:name = “sl-form”
attr:data-id = “sl_datta_id_attr”rule:providers = “required”error_re
porting=”inline”
wrap_errors=”<span class=’error-inline’>|</span>”on_submit = “s
l()”return = “smart-members/profile”
no_email_return = “smart-members/edit-profile”remember_me =
“yes”
secure_action=”yes”
secure_return=”yes”enable_recaptcha = “yes”
popup = “yes”
}
{providers}
{if providers:count == 1}
<p>
<select name=”providers”>
<option value=””></option>
{/if}
<option value=”{provider_name}”>{provider_label}</option>
{if providers:count == providers:total_results}
</select>
{error:providers}
</p>
{/if}
```

```
{/providers}<div class="login-register-click cf">
<input type="submit" class="login-popup-btn register" value="Login With Social media" >
</div>
{/exp:smsp:social_login}
```

Edit Profile

Edit profile form is used to update the member static data as well as custom data. A member can also edit their profile username, email and password with this form.

Normal static fields and custom fields doesn't require current password to update the data.

Dependent fields for login and email such as username, email and password fields needs current password to be entered to update the fields.

Tag for Edit profile module will look like this.

```
{exp:smsp:edit} Content data {/exp:smsp:edit}
```

Parameters

1. allowed_groups
2. rule:FIELD_NAME
3. attr:ATTRIBUTES
4. return
5. error_reporting

6. wrap_errors
7. on_submit
8. secure_action
9. secure_return
10. enable_recaptcha
11. member_id
12. allowed_admin_groups

Fields

1. group_id
2. username
3. email
4. password
5. password_confirm
6. avatar_filename
7. photo_filename
8. sig_img_filename
9. yahoo_im
10. url
11. location
12. occupation
13. interests
14. aol_im
15. msn_im
16. icq

17. bio
18. signature
19. captcha
20. CUSTOM_FIELD
21. CUSTOM_CHECKBOX_FIELD **(Pro Feature)**
22. CUSTOM_RADIO_FIELD **(Pro Feature)**
23. CUSTOM_MULTI_SELECT_FIELD **(Pro Feature)**
24. CUSTOM_FILE_FIELD **(Pro Feature)**

Following Parameters can be use in Edit Profile form

allowed_groups

Allowed groups parameter is used to give options to member for select any group from them. Use input field of group_id to enter the group id from defined allowed groups.

Example:

```
allowed_groups = "5|6|7"
```

rule:FIELD_NAME

Rule parameter is use to give custom rules to fields separate by pipe (|).

Example:

```
rule:username = "required"
rule:email = "required|valid_email|is_unique[members.email]"
```

attr:ATTRIBUTES

Parameter to add attributes in form. We can add classes, ids etc.

Example:

```
attr:id = “form_id”
attr:class = “form_class”
attr:name = “form_name”
attr:data-id = “form_data_id”
```

return

Return to any specific page after successful submission of form.

Example:

```
return = “smart-members/profile”
```

error_reporting

Error reporting format is defined by this parameter. It can be either “inline” or “outline”.

“Inline” error reporting will show the error in same page.
“outline” error reporting will show error in EE gray box in new page.

Example:

```
error_reporting=”inline”
```

wrap_errors

Use this parameter to wrap forms error in any span or div if set error_reporting=”inline”.

Example:

```
wrap_errors=”<span class=’error-inline’>|</span>”
```

on_submit

—

This parameter allows us to call any Javascript function on submit of form

Example:

```
on_submit="call_me( )"
```

secure_action

Secure action will post the data on secure site i.e., https

Example:

```
secure_action="yes"
```

secure_return

Secure return will return the page after submit of form on secure site i.e., https

Example:

```
secure_return="yes"
```

enable_recaptcha

This parameter enables recaptcha instead of normal captcha.

a. Scenario EE2:
    i. If this parameter is set and API key and SECRET is not passed for recaptcha in backend, The normal captcha will show.
    ii. If recaptcha API key and SECRET is passed in backend and this parameter not set:
        1. If the page is registration page and you have set captcha as required from member preferences, normal captcha will show.
        2. If the page is not registration page, neither captcha or recaptcha will show.

iii. If recaptcha API key and SECRET is passed in backend and this parameter is set:

1. If the page is registration page and you have set captcha as required from member preferences, The normal captcha will not show and recaptcha will override the settings.
2. If the page is not registration page, recaptcha will show.

b. Scenario EE3:

i. Same scenario like EE2. Only change is, there is backend member preference settings in EE3 that allows to not enter any captcha if member is logged in. So if you set this parameter and API key and SECRET is also passed, If you are logged in this recaptcha or captcha will not show until you set “Require CAPTCHA while logged in?” to “Yes” from backend member settings > captcha settings.

Example:

```
enable_recaptcha="yes"
```

member_id

Use this parameter if you want to edit some other member’s information instead of current logged in member. You can pass member_id of member whose data you want to modify and can modify just like you edit your own profile.

If you want to edit username, email or password, You need to have a field name current_password. You need to enter password of current logged in member in this field.

Example:

```
member_id="25"
```

```
member_id="{segment_3}"
```

allowed_admin_groups

This parameter play main role if you want to modify member data of other users. You can restrict modifiers by their group. So if you want to have only super admin, site admins and modifiers to edit information of other members, Just pass their group_id in this parameter and they will be able to edit all information for particular member.

Note: That can be a major risk if you give any random people to change data of other members. It is requested to use this functionality wisely.

Example:

```
allowed_admin_groups="1"
```

```
allowed_admin_groups="1|6|7"
```

Different methods that can be use by user are given below.

Input

```
<input type="email" name="username" placeholder="Email Addr
ess">
{error:username} //if error_reporting="inline"
```

Textarea

```
<textarea name="signature"></textarea>
{error:signature} //if error_reporting="inline"
```

File field

```
<input type="file" name="avatar_filename" />

{error:avatar_filename} //if error_reporting="inline"
```

Select Dropdown: (For member group)

```
{data_group_id}
{if data_group_id:count == 1}
<select name="group_id">
<option value=""></option>
{/if}
<option value = "{group_id_value}" {if group_id == group_id_valu
e} selected{/if}> {group_id_label}</option>
{if data_group_id:count == data_group_id:total_results}
</select>
{error:group_id} //if error_reporting="inline"
{/if}
{/data_group_id}
```

Select Dropdown: (For custom field)

```
{data_CUSTOM_FIELD}
{if data_CUSTOM_FIELD:count == 1}
<select name="CUSTOM_FIELD">
<option value=""></option>
{/if}
<option value = "{CUSTOM_FIELD_value}" {if CUSTOM_FIELD_valu
e == CUSTOM_FIELD}selected{/if}> {CUSTOM_FIELD_label} </opti
```

```
{if data_CUSTOM_FIELD:value == data_CUSTOM_FIELD:option_value}selected{/if}> {CUSTOM_FIELD_label} </opti

on>
{if data_CUSTOM_FIELD:count == data_CUSTOM_FIELD:total_result
s}
</select>
{error:CUSTOM_FIELD} //if error_reporting="inline"
{/if}
{/data_CUSTOM_FIELD}
```

Captcha

```
{if captcha}
<p>
<label for="captcha">Please enter in the word you see:</label>
{captcha}
<input type="text" name="captcha" id="captcha" />
{error:captcha}
</p>
{if:elseif recaptcha}
<p style="margin: 0;">
<label for="recaptcha">Click the checkbox</label>{recaptcha}
{error:recaptcha}
</p>
{/if}
```

CUSTOM_CHECKBOX_FIELD (Pro Feature)

```
{data_CUSTOM_CHECKBOX_FIELD}
{if data_CUSTOM_CHECKBOX_FIELD:count == 1}
<p style="display: inline-flex; ">
{/if}
<input type="checkbox" name="CUSTOM_CHECKBOX_FIELD[]" val
ue="{CUSTOM_CHECKBOX_FIELD_value}" id="test_{data_CUSTOM
_CHECKBOX_FIELD:count}" {if CUSTOM_CHECKBOX_FIELD_value:ex
ists}checked{/if}>
<label for="test_{data_CUSTOM_CHECKBOX_FIELD:count}">{CUST
OM_CHECKBOX_FIELD_label}</label>
{if data_CUSTOM_CHECKBOX_FIELD:count == data_CUSTOM_CHE
CKBOX_FIELD:total_results}
{error:CUSTOM_CHECKBOX_FIELD}
</p>
{/if}
{/data_CUSTOM_CHECKBOX_FIELD}
```

CUSTOM_RADIO_FIELD (Pro Feature)

```
{data_CUSTOM_RADIO_FIELD}
{if data_CUSTOM_RADIO_FIELD:count == 1}
<p style="display: inline-flex;">
{/if}
<input type="radio" name="CUSTOM_RADIO_FIELD" value="{CUS
TOM_RADIO_FIELD_value}" id="test_{data_CUSTOM_RADIO_FIELD:
```

```
TOM_RADIO_FIELD_value}" id="test_{data_CUSTOM_RADIO_FIELD:

count}" {if CUSTOM_RADIO_FIELD_value:exists}checked{/if}>
<label for="test_{data_CUSTOM_RADIO_FIELD:count}">{CUSTOM_
RADIO_FIELD_label}</label>
{if data_CUSTOM_RADIO_FIELD:count == data_CUSTOM_RADIO_FI
ELD:total_results}
{error:CUSTOM_RADIO_FIELD}
</p>
{/if}
{/data_CUSTOM_RADIO_FIELD}
```

CUSTOM_MULTI_SELECT_FIELD (Pro Feature)

```
{data_CUSTOM_MULTI_SELECT_FIELD}
{if data_CUSTOM_MULTI_SELECT_FIELD:count == 1}
<p style="display: inline-flex; ">
<select multiple name="CUSTOM_MULTI_SELECT_FIELD[]">
{/if}
<option value="{CUSTOM_MULTI_SELECT_FIELD_value}" {if CUSTO
M_MULTI_SELECT_FIELD_value:exists}selected{/if}>{CUSTOM_MULT
I_SELECT_FIELD_label}</option>
{if data_CUSTOM_MULTI_SELECT_FIELD:count == data_CUSTOM_
MULTI_SELECT_FIELD:total_results}
</select>
{error:CUSTOM_MULTI_SELECT_FIELD}
</p>
```

```
</p>

{/if}
{/data_CUSTOM_MULTI_SELECT_FIELD}
```

CUSTOM_FILE_FIELD (Pro Feature).

```
<p>
Old file name: {CUSTOM_FILE_FIELD}<br>
<input type="file" name="CUSTOM_FILE_FIELD">
{error:CUSTOM_FILE_FIELD}
</p>
```

Example:

```
{exp:smsp:edit
allowed_groups="6|7|5"rule:username="required|valid_email|min_l
ength[5]"
rule:password="required|matches[password_confirm]|min_length
[5]"
rule:password_confirm="required|min_length[5]"
rule:group_id="required"
rule:state="required"
rule:first_name="required"attr:id="edit_profile_id"
attr:class="edit_profile_class"
attr:name="edit_profile-form"
attr:data-id="edit_profile_data_id_attr"return="smart-members/ed
```

```
it-profile"
enable_recaptcha="yes"
error_reporting="inline"
wrap_errors="<span class='error-inline'>|</span>"
on_submit="edit_profile()"
}
<p>
<input type="text" name="CUSTOM_FIELD" placeholder="CUSTO
M FIELD">
{error:CUSTOM_FIELD}
</p>{data_group_id}
{if data_group_id:count == 1}
<p>
<select name="group_id">
<option value=""></option>
{/if}
<option value="{group_id_value}" {if group_id_value == group_id}
selected{/if}>
{group_id_label}
</option>
{if data_group_id:count == data_group_id:total_results}
</select>
{error:group_id}
</p>
{/if}
```

```
{/data_group_id}
<p>
<input type="text" name="username" value="{username}">
{error:username}
</p>
<p>
<input type="text" name="email" value="{email}">
{error:email}
</p>
<p>
<input type="password" name="current_password">
{error:current_password}
</p>
<p>
<input type="password" name="password">
{error:password}
</p>
<p>
<input type="password" name="password_confirm">
{error:password_confirm}
</p>
<p>
{if avatar_filename}
<img src="{avatar_url}{avatar_filename}" height="100" width="10
```

0″>{/if}
<input type=″file″ name=″avatar_filename″ />
{error:avatar_filename}

{if captcha}

<label for=″captcha″>Please enter in the word you see:</label>
{captcha}
<input type=″text″ name=″captcha″ id=″captcha″ />
{error:captcha}

{if:elseif recaptcha}
<p style=″margin: 0;″><label for=″recaptcha″>Click the checkbox
</label>
{recaptcha}
{error:recaptcha}

{/if}
<div class=″edit-profile-click cf″>
<input type=″submit″ class=″edit″ value=″Update Profile″>

{/exp:smsp:edit}

Remove Images

Member can remove their images from “Edit member form”. This section contains removing of default images i.e., “Avatar file, Photo fIle and Signature image file”.

You can remove these images in 2 way.

Parameters

1. By submit the form:
2. By input field:

By submit the form

You need to pass an submit button named the file.

Examples:

To remove Avatar file pass this “name” attribute in submit button: (You can pass anything in “value” according to your form.)

```
<input type=”submit” name=”remove_avatar” value=”Remove Av
atar” >
<input type=”submit” name=”remove_avatar” value=”Submit” >
<input type=”submit” name=”remove_avatar” value=”Edit Profile”
>
```

To remove Photo file pass this “name” attribute in submit button: (You can pass anything in “value” according to your form.)

```
<input type=”submit” name=”remove_photo” value=”Remove Ph
oto” >
```

```
<input type="submit" name="remove_photo" value="Submit" >

<input type="submit" name="remove_photo" value="Edit Profile"
>
```

To remove Signature file pass this "name" attribute in submit button: (You can pass anything in "value" according to your form.)

```
<input type="submit" name="remove_sig_img" value="Remove Si
gnature">
<input type="submit" name="remove_sig_img" value="Submit" >
<input type="submit" name="remove_sig_img" value="Edit Profil
e" >
```

When you submit the form with this input type, it will remove the file when you submit the form (Only file entry will remove, Images will still there in image upload folder till user not upload another image.)

By input field

You can also pass a hidden value or radio button or javascript named with attribute remove name to remove that image.

Examples:

```
<input type='hidden' name='remove_avatar'>
<input type='hidden' name='remove_photo'>
<input type='hidden' name='remove_sig_img'>
```

or

```
<input type="radio" name="remove_avatar" value="remove"/> R
```

```
<input type="radio" name="remove_avatar" value="remove" /> R
emove Avatar?
<input type="radio" name="remove_photo" value="remove"/> R
emove Photo?
<input type="radio" name="remove_sig_img" value="remove"/>
Remove Signature?
```

Below is the example of JQuery remove:

Example:

```
{exp:smsp:edit
return="smart-members/edit-profile"
password_required="no"
error_reporting="inline"
wrap_errors="<span class='error-inline'>|</span>"
}
<p>
{if avatar_filename}
<div>
<img src="{avatar_url}{avatar_filename}" height="100" width="10
0">
<a href="javascript:void(0);" class="remove_sm_pic" add_field_attr
="remove_avatar">
Remove Avatar
</a><br>
```

```
</div>
{/if}
<input type="file" name="avatar_filename" />
{error:avatar_filename}
</p>
<p>
{if photo_filename}
<div>
<img src="{photo_url}{photo_filename}" height="100" width="10
0">
<a href="javascript:void(0);" class="remove_sm_pic" add_field_attr
="remove_photo">
Remove Photo
</a><br>
</div>
{/if}
<input type="file" name="photo_filename" />
{error:photo_filename}
</p>
<p>
{if sig_img_filename}
<div>
<img src="{sig_img_url}{sig_img_filename}" height="100" width
="100">
<a href="javascript:void(0);" class="remove_sm_pic" add_field_attr
```

```
="remove_sig_img">
Remove Signature
</a><br>
</div>
{/if}
<input type="file" name="sig_img_filename" />
{error:sig_img_filename}
</p><div class="login-edit-click cf">
<input type="submit" value="Update Account Information" >
</div>
{/exp:smsp:edit}
```

JQuery:

```
<script type="text/javascript">
$(document).ready(function() {
$(document).on('click', '.remove_sm_pic', function(event) {
event.preventDefault();
/* Act on the event */if(typeof($(this).attr('add_field_attr')) !== "un
defined")
{
if($(this).parent('div').find("input[name="+ $(this).attr('add_field_at
tr') +"]").length == 0)
{
$(this).parent('div').hide()
```

```
$(this).parent('div').append("<input type='hidden' name='"+ $(this).attr('add_field_attr') +"'>")
}
}
else
{
console.log('Cannnot remove')
}
});
});
</script>
```

View Profile

View profile module is to list the member(s) data. You can access each and every data except password with this module.

Tag for View profile module will look like this.

```
{exp:smsp:profile} Content data {/exp:smsp:profile}
```

Parameters

1. member_id
2. group_id
3. not_member_id

3. not_member_id
4. not_group_id
5. limit
6. order_by
7. sort

Fields

1. aol_im
2. avatar_url
3. avatar_filename
4. avatar_height
5. avatar_width
6. bday_d
7. bday_m
8. bday_y
9. bio
10. email
11. group_id
12. icq
13. interests
14. join_date
15. last_activity
16. last_bulletin_date
17. last_comment_date
18. last_entry_date

19. last_email_date
20. last_forum_post_date
21. last_view_bulletins
22. last_visit
23. location
24. member_id
25. msn_im
26. notepad
27. notepad_size
28. occupation
29. private_messages
30. photo_url
31. photo_filename
32. photo_height
33. photo_width
34. screen_name
35. signature
36. sig_img_url
37. sig_img_filename
38. sig_img_width
39. sig_img_height
40. total_comments
41. total_entries
42. total_forum_topics
43. total_forum_posts

44. username
45. url
46. yahoo_im
47. CUSTOM_FIELD

Labels

1. aol_im_label
2. avatar_filename_label
3. avatar_height_label
4. avatar_width_label
5. bday_d_label
6. bday_m_label
7. bday_y_label
8. bio_label
9. email_label
10. group_id_label
11. icq_label
12. interests_label
13. join_date_label
14. last_activity_label
15. last_bulletin_date_label
16. last_comment_date_label
17. last_entry_date_label
18. last_email_date_label
19. last_forum_post_date_label

20. last_view_bulletins_label
21. last_visit_label
22. location_label
23. member_id_label
24. msn_im_label
25. notepad_label
26. notepad_size_label
27. occupation_label
28. private_messages_label
29. photo_filename_label
30. photo_height_label
31. photo_width_label
32. screen_name_label
33. signature_label
34. sig_img_filename_label
35. sig_img_width_label
36. sig_img_height_label
37. total_comments_label
38. total_entries_label
39. total_forum_topics_label
40. total_forum_posts_label
41. username_label
42. url_label
43. yahoo_im_label
44. CUSTOM_FIELD_label

To list all the fields above there is single field. It will list all the static fields which is not empty as well as all the custom fields created by user.

1. sm_list_all_fields
    a. Parameters (It doesn't contain any parameter)
    b. Fields
        i. field_label
        ii. field_value
        iii. field_sort_name
        iv. field_db_name
        v. sm_list_all_fields:count
        vi. sm_list_all_fields:total_results

Example:

```
{sm_list_all_fields}
{field_label} : {field_value}
{/sm_list_all_fields}
```

Following Parameters can be use in View Profile page

member_id

Member ID of user you want to get the data of.
Not to use this parameter, Leave this parameter blank or pass "CURRENT_MEMBER" in parameter will extract profile data of current member.
Pass "ALL_MEMBERS" in parameter if you want to fetch all the members.

Example:

```
member_id = ""
```

```
member_id = “CURRENT_MEMBER”

member_id = “ALL_MEMBERS”
member_id = “53”
member_id = “50|51|52|53”
```

group_id

Member(s) of particular group you want to get data of.
Using this parameter will filter the output data with member group. If member group doesn’t exists in the groups passed in parameter, it will not show the data of member.
Passing nothing in this parameter will bring every groups except “banned (2)”, “Guests (3)” and “Pending (4)”.

Example:

```
group_id = “5”
group_id = “5|6|7”
```

not_member_id

You can pass the ID of the member(s) with pipe (|) separated to ignore record of that member.

Example:

```
not_member_id = “2”
not_member_id = “2|4”
```

not_group_id

You can pass the ID of the group with pipe (|) separated to ignore record of that group.

Example:

```
not_group_id = “5”
not_group_id = “5|6”
```

limit

You can Limit output rows by any value you want.

Example:

```
limit = “50”
```

order_by

This parameter use to start sorting via any proper field. You can use both default member field or custom member field to group by the result.

Example:

```
order_by = “member_id”
order_by = “first_name”
```

sort

You can sort the output by ASC or DESC.

Example:

```
sort = “asc”
sort = “desc”
```

How to render custome member fields

We have developed custem member field types like checkbox, dropdown, radio button, file, etc

user can render this custom member fields with some tags like below

user can render this custom member fields with some tags like below

If user want to render checkbox and multi-select dropdown then use below code.

Example:

```
{CUSTM_MEMBER_FIELD_SHORT_NAME}
{CUSTM_MEMBER_FIELD_SHORT_NAME:label} : {CUSTM_MEMBER_FIELD_SHORT_NAME:value}
{CUSTM_MEMBER_FIELD_SHORT_NAME}
```

If user want to render other then checkbox and multi-select dropdown (means single value custom field) then use below code.

Example:

```
{CUSTM_MEMBER_FIELD_SHORT_NAME}
```

OR

```
{CUSTM_MEMBER_FIELD_SHORT_NAME:label} : {CUSTM_MEMBER_FIELD_SHORT_NAME:value}
```

Example:

```
{exp:smsp:profile
member_id="ALL_MEMBERS"not_group_id="1|6"
not_member_id="14|1"
}
{if no_results} <p>no data found!!</p> {/if}<p>Your details of member id {member_id} are as follows:</p><p>All Fields:</p>
```

```
<ul>
{sm_list_all_fields}
<li><b>{field_label} : </b> {field_value}</li>
{/sm_list_all_fields}
</ul><p>Fields of interest:</p>
<ul>
<li> <b> {group_id_label} : </b> {group_id} </li>
<li> <b> {username_label} : </b> {username} </li>
<li> <b> {screen_name_label} : </b> {screen_name}</li>
<li> <b> {email_label} : </b> {email} </li>
</ul>{/exp:smsp:profile}
```

Forgot Password

Forgot Password module is use to send URL of reset password with key to reset the password of user.
Parameter of reset password template as well as forgot password email settings can be sent from backend smart members setting or can pass in this tag as parameter.

Tag for View profile module will look like this.

```
{exp:smsp:forgot_password}
... Content data ...
{/exp:smsp:forgot_password}
```

Parameters

1. rule:FIELD_NAME
2. attr:ATTRIBUTES
3. return
4. error_reporting
5. wrap_errors
6. on_submit
7. secure_action
8. secure_return
9. reset_password_template
10. email:subject
11. email:template
12. email:word_wrap
13. email:mailtype
14. enable_recaptcha

Fields

1. email
2. captcha

Following Parameters can be use in Forgot Password form

rule:FIELD_NAME

Rule parameter is use to give custom rules to fields separate by pipe (|).

Example:

```
rule:username = “required”
```

```
rule:email = “required|valid_email|is_unique[members.email]”
```

attr:ATTRIBUTES

Parameter to add attributes in form. We can add classes, ids etc.

Example:

```
attr:id = “form_id”
attr:class = “form_class”
attr:name = “form_name”
attr:data-id = “form_data_id”
```

return

Return to any specific page after successful submission of form.

Example:

```
return = “smart-members/profile”
```

error_reporting

Error reporting format is defined by this parameter. It can be either “inline” or “outline”.
“Inline” error reporting will show the error in same page.
“outline” error reporting will show error in EE gray box in new page.

Example:

```
error_reporting=”inline”
```

wrap_errors

Use this parameter to wrap forms error in any span or div if set error_reporting=”inline”.
If error_reporting is set to inline and not defined this parameter, It will take span to

display errors.

Example:

```
wrap_errors="<span class='error-inline'>|</span>"
```

on_submit

This parameter allows us to call any Javascript function on submit of form.

Example:

```
on_submit="call_me( )
```

secure_action

Secure action will post the data on secure site i.e., https.

Example:

```
secure_action="yes"
```

secure_return

Secure return will return the page after submit of form on secure site i.e., https.

Example:

```
secure_return="yes"
```

reset_password_template

Pass template path of reset password page. URL of this page will pass in forgot password email with reset token to user.
(Note: This can be passed from template as parameter or can set from backend member settings. If passed from backend don't pass this parameter.)

Example:

```
reset_password_template="smart-members/reset-password"
```

email:subject

Pass forgot password email subject in this parameter. Pass the value in this parameter will show as a subject of email.
(Note: This can be passed from template as parameter or can set from backend member settings. If passed from backend don't pass this parameter.)

Example:

```
email:subject = "Reset password request | {site_name}"
```

email:template

Pass forgot password email template path in this parameter. Template passed in this parameter will send as forgot password email body to member.
(Note: This can be passed from template as parameter or can set from backend member settings. If passed from backend don't pass this parameter.)

Example:

```
email:template = "smart-members/email-forgot-password"
```

email:word_wrap

Pass the word wrap settings in this parameter.
(Note: This can be passed from template as parameter or can set from backend member settings. If passed from backend don't pass this parameter.)

Example:

```
email:word_wrap = "yes"
```

email:mailtype

Pass email type in this parameter. Choosing right parameter is very important. If we choose "html", the normal text will execute in same line and it will not look proper in

choose "html", the normal text will execute in same line and it will not look proper in email. Same as if we choose "text" as parameter, It will not consider HTML tags such as <p> <b> etc. and the tags will appear in mail body.
(Note: This can be passed from template as parameter or can set from backend member settings. If passed from backend don't pass this parameter.)

Example:

```
email:mailtype = "text"
email:mailtype = "html"
```

enable_recaptcha

This parameter enables recaptcha instead of normal captcha.

a. Scenario EE2:
   i. If this parameter is set and API key and SECRET is not passed for recaptcha in backend, The normal captcha will show.
   ii. If recaptcha API key and SECRET is passed in backend and this parameter not set:
      1. If the page is registration page and you have set captcha as required from member preferences, normal captcha will show.
      2. If the page is not registration page, neither captcha or recaptcha will show.
   iii. If recaptcha API key and SECRET is passed in backend and this parameter is set:
      1. If the page is registration page and you have set captcha as required from member preferences, The normal captcha will not show and recaptcha will override the settings.
      2. If the page is not registration page, recaptcha will show.

b. Scenario EE3:
   i. Same scenario like EE2. Only change is, there is backend member preference settings in EE3 that allows to not enter any captcha if member

is logged in. So if you set this parameter and API key and SECRET is also passed, If you are logged in this recaptcha or captcha will not show until you set “Require CAPTCHA while logged in?” to “Yes” from backend member settings > captcha settings.

Example:

```
enable_recaptcha=”yes”
```

Different methods that can be use by user are given below.

Input

```
<input type=”email” name=”username” placeholder=”Email Addr
ess”>
{error:username} //if error_reporting=”inline”
```

Captcha

```
{if captcha}
<p>
<label for=”captcha”>Please enter in the word you see:</label>
{captcha}
<input type=”text” name=”captcha” id=”captcha” />
{error:captcha}
</p>
{if:elseif recaptcha}
<p style=”margin: 0;”>
<label for=”recaptcha”>Click the checkbox</label>{recaptcha}
```

```
{error:recaptcha}
</p>
{/if}
```

Example:

```
{exp:smsp:forgot_password
attr:id="fp_id"
attr:class="fp_class"
attr:name="fp-form"
attr:data-id="fp_data_id_attr"return="smart-members/send-forg-mail"error_reporting="inline"
wrap_errors="<span class='error-inline'>|</span>"on_submit="fp()"
enable_recaptcha="yes"reset_password_template="smart-members/reset-password"email:subject="Reset password request"
email:template="smart-members/forgot-password-email-template"
email:word_wrap="yes"
email:mailtype="html"
}
<p>
<input type="email" name="email">
{error:email}
</p>{if captcha}
```

```
<p>
<label for="captcha">Please enter in the word you see:</label>
{captcha}
<input type="text" name="captcha" id="captcha" />{error:captch
a}
</p>
{if:elseif recaptcha}
<p style="margin: 0;">
<label for="recaptcha">Click the checkbox</label>
{recaptcha}
{error:recaptcha}
</p>
{/if}
<p>
<input type="submit" class="forgot-pass" value="Send email" >
</p>
{/exp:smsp:forgot_password}
```

Reset Password

Reset Password module is use to set new password with help of token send by user in mail through forgot password email.

Tag for Reset password module will look like this.

```
{exp:smsp:reset_password}
```

... Content data ...

{/exp:smsp:reset_password}

Parameters

1. rule:FIELD_NAME
2. attr:ATTRIBUTES
3. return
4. error_reporting
5. wrap_errors
6. on_submit
7. secure_action
8. secure_return
9. enable_recaptcha
10. reset_code

Fields

1. password
2. password_confirm
3. captcha

Following Parameters can be use in Forgot Password form

rule:FIELD_NAME

Rule parameter is use to give custom rules to fields separate by pipe (|).

Example:

```
rule:username = “required”
rule:email = “required|valid_email|is_unique[members.email]”
```

attr:ATTRIBUTES

Parameter to add attributes in form. We can add classes, ids etc.

Example:

```
attr:id = “form_id”
attr:class = “form_class”
attr:name = “form_name”
attr:data-id = “form_data_id”
```

return

Return to any specific page after successful submission of form.

Example:

```
return = “smart-members/profile”
```

error_reporting

Error reporting format is defined by this parameter. It can be either “inline” or “outline”.
“Inline” error reporting will show the error in same page.
“outline” error reporting will show error in EE gray box in new page.

Example:

```
error_reporting=”inline”
```

wrap_errors

Use this parameter to wrap forms error in any span or div if set error_reporting="inline".
If error_reporting is set to inline and not defined this parameter, It will take span to display errors.

Example:

```
wrap_errors="<span class='error-inline'>|</span>"
```

on_submit

This parameter allows us to call any Javascript function on submit of form.

Example:

```
on_submit="call_me( )
```

secure_action

Secure action will post the data on secure site i.e., https.

Example:

```
secure_action="yes"
```

secure_return

Secure return will return the page after submit of form on secure site i.e., https.

Example:

```
secure_return="yes"
```

enable_recaptcha

This parameter enables recaptcha instead of normal captcha.

a. Scenario EE2:

i. If this parameter is set and API key and SECRET is not passed for recaptcha in backend, The normal captcha will show.

ii. If recaptcha API key and SECRET is passed in backend and this parameter not set:

1. If the page is registration page and you have set captcha as required from member preferences, normal captcha will show.

2. If the page is not registration page, neither captcha or recaptcha will show.

iii. If recaptcha API key and SECRET is passed in backend and this parameter is set:

1. If the page is registration page and you have set captcha as required from member preferences, The normal captcha will not show and recaptcha will override the settings.

2. If the page is not registration page, recaptcha will show.

b. Scenario EE3:

i. Same scenario like EE2. Only change is, there is backend member preference settings in EE3 that allows to not enter any captcha if member is logged in. So if you set this parameter and API key and SECRET is also passed, If you are logged in this recaptcha or captcha will not show until you set “Require CAPTCHA while logged in?” to “Yes” from backend member settings > captcha settings.

Example:

```
enable_recaptcha=”yes”
```

reset_code

Reset code is the token sent in forgot password email to user. You can simply set it parameterize from URL segment.

Example:

```
reset_code = “{segment_3}”
```

Different methods that can be use by user are given below.

Input

```
<input type=”email” name=”password” placeholder=”Password”<
{error:password} //if error_reporting=”inline”
```

Captcha

```
{if captcha}
<p>
<label for=”captcha”>Please enter in the word you see:</label>
{captcha}
<input type=”text” name=”captcha” id=”captcha” />
{error:captcha}
</p>
{if:elseif recaptcha}
<p style=”margin: 0;”>
<label for=”recaptcha”>Click the checkbox</label>{recaptcha}
{error:recaptcha}
</p>
{/if}
```

Example:

```
{if segment_3 != “reset-success”}
{exp:smsp:reset_password
reset_code=”{segment_3}”
rule:password=”required|matches[password_confirm]|min_length
[5]”
rule:password_confirm=”required|min_length[5]”attr:id=”rp_id”
attr:class=”rp_class”
attr:name=”rp-form”
attr:data-id=”rp_data_id_attr”return=”smart-members/reset-pass
word/reset-success”
enable_recaptcha=”yes”error_reporting=”inline”
wrap_errors=”<span class=’error-inline’>|</span>”
on_submit=”rp()”
}{if no_results}<h4>The reset token provided is invalid.</h4>
{/if}
<p>
<input type=”password” name=”password”>{error:password}
</p><p>
<input type=”password” name=”password_confirm”>{error:passw
ord_confirm}
</p>{if captcha}
<p>
<label for=”captcha”>Please enter in the word you see:</label>
{captcha}
```

```
<input type="text" name="captcha" id="captcha" />
{error:captcha}
</p>
{if:elseif recaptcha}
<p style="margin: 0;">
<label for="recaptcha">Click the checkbox</label>
{recaptcha}
{error:recaptcha}
</p>
{/if}
<p>
<input type="submit" class="reset" value="Reset password" >
</p>
{/exp:smsp:reset_password}{if:else}
<p> Password successfully updated. You can now <a href="/smar
t-members/login"> login</a> with new credentials.</p>
{/if}
```

Delete Member

Delete member module is use to allow member to delete himself from membership account of site. If one will delete his/her account, all the entries or data he/she has entered in the site will also deletes.

Any one can delete his/her account from site except super admin.

Any one can delete his/her account from site except super admin.

To delete the member user only needs his/her account password.

Tag for Delete member module will look like this.

```
{exp:smsp:delete}
... Content data ...
{/exp:smsp:delete}
```

Parameters

1. rule:FIELD_NAME
2. attr:ATTRIBUTES
3. return
4. error_reporting
5. wrap_errors
6. on_submit
7. secure_action
8. secure_return
9. enable_recaptcha

Fields

1. password
2. captcha

Following Parameters can be use in Forgot Password form

rule:FIELD_NAME

Rule parameter is use to give custom rules to fields separate by pipe (|).

Example:

```
rule:username = “required”
rule:email = “required|valid_email|is_unique[members.email]”
```

attr:ATTRIBUTES

Parameter to add attributes in form. We can add classes, ids etc.

Example:

```
attr:id = “form_id”
attr:class = “form_class”
attr:name = “form_name”
attr:data-id = “form_data_id”
```

return

Return to any specific page after successful submission of form.

Example:

```
return = “smart-members/profile”
```

error_reporting

Error reporting format is defined by this parameter. It can be either “inline” or “outline”.
“Inline” error reporting will show the error in same page.
“outline” error reporting will show error in EE gray box in new page.

Example:

```
error_reporting=”inline”
```

wrap_errors

Use this parameter to wrap forms error in any span or div if set error_reporting="inline".
If error_reporting is set to inline and not defined this parameter, It will take span to display errors.

Example:

```
wrap_errors="<span class='error-inline'>|</span>"
```

on_submit

This parameter allows us to call any Javascript function on submit of form.

Example:

```
on_submit="call_me( )
```

secure_action

Secure action will post the data on secure site i.e., https.

Example:

```
secure_action="yes"
```

secure_return

Secure return will return the page after submit of form on secure site i.e., https.

Example:

```
secure_return="yes"
```

enable_recaptcha

This parameter enables recaptcha instead of normal captcha

This parameter enables recaptcha instead of normal captcha.

a. Scenario EE2:
    i. If this parameter is set and API key and SECRET is not passed for recaptcha in backend, The normal captcha will show.
    ii. If recaptcha API key and SECRET is passed in backend and this parameter not set:
        1. If the page is registration page and you have set captcha as required from member preferences, normal captcha will show.
        2. If the page is not registration page, neither captcha or recaptcha will show.
    iii. If recaptcha API key and SECRET is passed in backend and this parameter is set:
        1. If the page is registration page and you have set captcha as required from member preferences, The normal captcha will not show and recaptcha will override the settings.
        2. If the page is not registration page, recaptcha will show.

b. Scenario EE3:
    i. Same scenario like EE2. Only change is, there is backend member preference settings in EE3 that allows to not enter any captcha if member is logged in. So if you set this parameter and API key and SECRET is also passed, If you are logged in this recaptcha or captcha will not show until you set “Require CAPTCHA while logged in?” to “Yes” from backend member settings > captcha settings.

Example:

```
enable_recaptcha=”yes”
```

Different methods that can be use by user are given below.

Input

```
<input type="password" name="password" placeholder="Passwo
rd">
{error:password} //if error_reporting="inline"
```

Captcha

```
{if captcha}
<p>
<label for="captcha">Please enter in the word you see:</label>
{captcha}
<input type="text" name="captcha" id="captcha" />
{error:captcha}
</p>
{if:elseif recaptcha}
<p style="margin: 0;">
<label for="recaptcha">Click the checkbox</label>{recaptcha}
{error:recaptcha}
</p>
{/if}
```

Example:

```
{exp:smsp:delete
attr:id="dm_id"
attr:class="dm_class"
attr:name="dm-form"
```

```
attr:data-id="dm_data_id_attr"return="smart-members/index"erro
r_reporting="inline"
wrap_errors="<span class='error-inline'>|</span>"on_submit="d
m()"
enable_recaptcha="yes"
}
<p>
<input type="password" name="password" placeholder="Passwo
rd">
{error:password}
</p>
{if captcha}
<p>
<label for="captcha">Please enter in the word you see:</label>
{captcha}
<input type="text" name="captcha" id="captcha" />
{error:captcha}
</p>
{if:elseif recaptcha}
<p style="margin: 0;">
<label for="recaptcha">Click the checkbox</label>
{recaptcha}
{error:recaptcha}
</p>
{/if}
```

```
<p>
<input type="submit" class="delete" value="Delete Account" >
</p>
{/exp:smsp:delete}
```

Logout

Logout module provides a user to successfully destroy the session and logging out from the site.

Tag for logout created in 2 ways. One is with closing tag and another one is without closing tag.

With closing tag, content code will look something like this:

```
{exp:smsp:logout return='smart-members/index'}
<p> <a href="{url}">Logout</a> </p>
{/exp:smsp:logout}
```

Without closing tag, content code will look like this:

```
<a href="{exp:smsp:logout return='smart-members/index'}">Log out</a>
```

Parameters

1. return
2. secure_return

Following Parameters can be use in Forgot Password form

return

Return to any specific page after successful submission of form.

Example:

```
return = “smart-members/profile”
```

secure_return

Secure return will return the page after submit of form on secure site i.e., https.

Example:

```
secure_return=”yes”
```

Plan - New

The plan tag is used to show the plan list for a logged-in member.

Tag for Plan module will look like this.

```
{exp:smsp:plan} Content data {/exp:smsp:plan}
```

Parameters

1. id
2. name

Fields

1. id
2. name
3. description
4. confirmation
5. initial_payment
6. billing_amount
7. cycle_number
8. cycle_period
9. billing_limit
10. trial_amount
11. trial_limit
12. allow_signups
13. expiration_number
14. expiration_period
15. recurring_subscription
16. custom_trial

The following Parameters can be used in the Plan tag.

id

ID of the plan.

Example:

```
id = “5”
```

You can assign multiple plan IDs by seperated pipe sign(|).

```
id = “5|3|2|6”
```

name

Name of the plan.

Example:

```
name = “Plan name”
```

You can assign multiple plan names by seperated pipe sign(|).

```
id = “Plan A|Plan B|Plan C”
```

Note: If no parameters are defined, all the plan lists will be shown.

Example:

```
{exp:smsp:plan}
{id}
{name}
{description}
{confirmation}
{initial_payment}
{billing_amount}
{cycle_number}
{cycle_period}
{billing_limit}
{trial_amount}
{trial_limit}
{allow_signups}
```

{expiration_number}

{expiration_period}

{recurring_subscription}

{custom_trial}

{/exp:smsp:plan}

Discount codes - New

A discount code tag is used to show the discount code list.

Tag for the Discount module will look like this.

{exp:smsp:discount_code} Content data {/exp:smsp:discount_code}

Parameters

1. id
2. code
3. start_date
4. expiry_date

Fields

1. id
2. code
3. start_date

4. expiry_date

5. uses

If you want to access the plan associated with this discount code. You can access it this way.

```
{discount_code:plan}
{plan:level_id}
{plan:code_id}
{plan:initial_payment}
{plan:billing_amount}
{plan:cycle_number}
{plan:cycle_period}
{plan:billing_limit}
{plan:trial_amount}
{plan:trial_limit}
{plan:expiration_number}
{plan:expiration_period}
{plan:recurring_subscription}
{plan:custom_trial}
{/discount_code:plan}
```

The following parameters can be used in the Discount tag.

id

ID of the discount code.

Example:

```
id="5"
```

```
id = "5
```

You can assign multiple plan IDs by seperating pipe sign(|).

```
id = "5|3|2|6"
```

code

Cupan code of the discount.

Example:

```
code = "ABCDEF"
```

You can assign multiple codes by seperating pipe signs (|).

```
code = "ABCDEF|QWERTY|ZXCVBN"
```

start_date

Start date of the discount.

Example:

```
start_date = "2021-04-08"
```

You can assign multiple start_date by seperating pipe sign(|).

```
start_date = "2021-04-08|2021-04-09|2021-04-10"
```

expiry_date

Expiry date of the discount.

Example:

```
expiry_date = "2021-04-08"
```

You can assign multiple expiry_date by seperating pipe sign(|).

```
expiry_date = "2021-04-08|2021-04-09|2021-04-10"
```

Note: If you do not assign any parameter, all the discount lists will be shown.

Example:

```
{exp:smsp:discount_code}
{id}
{code}
{start_date}
{expiry_date}
{discount_code:plan}
{plan:level_id}
{plan:code_id}
{plan:initial_payment}
{plan:billing_amount}
{plan:cycle_number}
{plan:cycle_period}
{plan:billing_limit}
{plan:trial_amount}
{plan:trial_limit}
{plan:expiration_number}
{plan:expiration_period}
{plan:recurring_subscription}
```

```
{plan:custom_trial}
{/discount_code:plan}
{/exp:smsp:discount_code}
```

Order list - New

The order list tag is used to show the all order list for a logged-in member.

Tag for Order list will look like this.

```
{exp:smsp:orders} Content data {/exp:smsp:orders}
```

Parameters

1. id
2. code
3. user_id
4. membership_id
5. status
6. timestamp

Fields

1. id
2. code
3. user_id
4. membership_id
5. billing_name

5. billing_name
6. billing_street
7. billing_city
8. billing_state
9. billing_zip
10. billing_country
11. billing_phone
12. subtotal
13. tax
14. couponamount
15. total
16. payment_type
17. cardtype
18. accountnumber
19. expirationmonth
20. expirationyear
21. status
22. gateway
23. gateway_environment
24. payment_transaction_id
25. subscription_transaction_id
26. timestamp
27. affiliate_id
28. affiliate_subid
29. notes

If you want to access the plan associated with this order. You can access it this way.

```
{orders:plan}
{plan:id}
{plan:name}
{plan:description}
{plan:confirmation}
{plan:initial_payment}
{plan:billing_amount}
{plan:cycle_number}
{plan:cycle_period}
{plan:billing_limit}
{plan:trial_amount}
{plan:trial_limit}
{plan:allow_signups}
{plan:expiration_number}
{plan:expiration_period}
{plan:recurring_subscription}
{plan:custom_trial}
{/orders:plan}
```

If you want to access the discount code applied to the order. You can access it this way.

```
{orders:discount_code_uses}
```

```
{discount_code_uses:id}
{discount_code_uses:code_id}
{discount_code_uses:code}
{discount_code_uses:user_id}
{discount_code_uses:order_id}
{/orders:discount_code_uses}
```

The following parameters can be used in the Discount tag.

id

ID of the order.

Example:

```
id = “5”
```

You can assign multiple plan IDs by seperating pipe signs (|).

```
id = “5|3|2|6”
```

code

Order reference code.

Example:

```
code = “VOX5R8PHQS”
```

You can assign multiple codes by seperating pipe signs (|).

```
code = “VOX5R8PHQS|VOX5R8PHQT|VOX5R8PHQU”
```

user_id

Member ID of the order who placed the order.

Example:

```
user_id = “1”
```

You can assign multiple user_id by separating pipe signs(|).

```
user_id = “1|2|3|4”
```

membership_id

Id of the subscribed plan.

Example:

```
membership_id = “1”
```

You can assign multiple membership_id by separating pipe signs(|).

```
membership_id = “1|2|3|4”
```

status

Status of the order.

Example:

```
status = “success”
```

You can assign multiple statuses by separating pipe signs (|).

```
status = “success|cancelled”
```

timestamp

date and time of the order placed.

Example:

```
timestamp = “2021-04-08”
```

Note: If you do not assign any parameter, all the orders will be shown for the logged-in member.

Example:

```
{exp:smsp:orders}
{id}
{code}
{user_id}
{membership_id}
{billing_name}
{billing_street}
{billing_city}
{billing_state}
{billing_zip}
{billing_country}
{billing_phone}
{subtotal}
{tax}
{couponamount}
```

```
{total}

{payment_type}
{cardtype}
{accountnumber}
{expirationmonth}
{expirationyear}
{status}
{gateway}
{gateway_environment}
{payment_transaction_id}
{subscription_transaction_id}
{timestamp}
{affiliate_id}
{affiliate_subid}
{notes}
{/exp:smsp:orders}
```

Get subscribed plan ID - New

To get the all subscribed plan ID for a logged-in member on the front side you can use below the example.

```
{exp:smsp:get_subscribed_planid}
```

Example:

```
{exp:smsp:get_subscribed_planid}
```

Output:

```
1,2,3,5,8
```

If you want to match the particular plan from the subscribed plan list of users, you can do it like this.

```
{if "{exp:smsp:get_subscribed_planid}" ~ '/\b'."{id}".'\b/'}
Selected
{if:else}
Not selected
{/if}
```

Cancel order - New

Tag for cancel the order will look like this.

```
{exp:smsp:paypal_cancel}
```

Example:

```
{exp:smsp:paypal_cancel order_id='{segment_2}'}
```

Checkout - New

The checkout form tag will help you to checkout for a subscription plan.

Tag for checkout form will look like this.

```
{exp:smsp:form} Content {/exp:smsp:form}
```

Parameters

1. plan_id
2. prefix
3. return
4. payment_confirmation
5. cancel

Fields

Fields are categories into two parts. Plan fields and form fields. You can access the plan fields below. If you assign the prefix as “smsp” then you can access the fields this way. You can change the prefix as per your requirement. If no prefix is defined “smsp” will take by default.

Plan Fields

1. {smsp:plan:name}
2. {smsp:plan:description}
3. {smsp:plan:initial_payment}
4. {smsp:plan:billing_amount}
5. {smsp:plan:cycle_period}

5. {smsp:plan:cycle_period}
6. {smsp:plan:cycle_number}
7. {smsp:plan:billing_limit}
8. {smsp:plan:trial_amount}
9. {smsp:plan:trial_limit}

Example:

```
{exp:smsp:form plan_id='{segment_3}' prefix='smsp' return="smsp/confirmation" payment_confirmation="smsp/ppexpressconfirmation" cancel="smsp/cancel"}
{smsp:plan:name}
{smsp:plan:description}
{smsp:plan:initial_payment}
{smsp:plan:billing_amount}
{smsp:plan:cycle_period}
{smsp:plan:cycle_number}
{smsp:plan:billing_limit}
{smsp:plan:trial_limit}
{smsp:plan:trial_amount}
{/exp:smsp:form}
```

Form Fields

1. {smsp:form:discount_code}
2. {smsp:form:first_name}
3. {smsp:form:last_name}

4. {smsp:form:email}
5. {smsp:form:address1}
6. {smsp:form:address2}
7. {smsp:form:city}
8. {smsp:form:state}
9. {smsp:form:postal_code}
10. {smsp:form:phone}
11. {smsp:form:country}
12. {smsp:form:card_number}
13. {smsp:form:expiry_month}
14. {smsp:form:expiry_year}
15. {smsp:form:cvc}
16. {smsp:form:captcha}
17. {smsp:checkout:error}

Example:

```
{exp:smsp:form plan_id='{segment_3}' prefix='smsp' return="smsp/confirmation" payment_confirmation="smsp/ppexpressconfirmation" cancel="smsp/cancel"}
<font color="red">{smsp:checkout:error}</font>
Apply Discount:<input type='text' name="discount_code" value="{smsp:form:discount_code}" />
First Name:<input type='text' name="first_name" value="{smsp:form:first_name}" required="true"/>
Last name:<input type='text' name="last_name" value="{smsp:form:last_name}" />
```

m:last_name}" />

Email:<input type='text' name="email" value="{smsp:form:email}" required="true"/>
Address1:<input type='text' name="address1" value="{smsp:form:address1}" required="true"/>
Address2:<input type='text' name="address2" value="{smsp:form:address2}"/>
City:<input type='text' name="city" value="{smsp:form:city}" required="true"/>
State:<input type='text' name="state" value="{smsp:form:state}" required="true"/>
Postal code:<input type='text' name="postal_code" value="{smsp:form:postal_code}" required="true"/>
Phone:<input type='text' name="phone" value="{smsp:form:phone}" required="true"/>
Country:<select name="country" value="{smsp:form:country}" required="true">
<option value="GB" data-countryISDCode="44">UK (+44)</option>
<option value="US" data-countryISDCode="1" selected='true'>USA (+1)</option>
<option value="IN" data-countryISDCode="91">India (+91)</option>
</select>
Card number:<input type='text' name="card_number" />

```
Month:<select name="expiry_month" value="{smsp:form:expiry_
month}">
<option>1</option>
<option>2</option>
<option>3</option>
<option>4</option>
<option>5</option>
<option>6</option>
</select>
Year:<select name="expiry_year" value="{smsp:form:expiry_yea
r}">
<option>2021</option>
<option>2022</option>
<option selected="">2023</option>
<option>2024</option>
</select>
CVC:<input type='text' name="cvc" value="132"/>
Captcha:{smsp:form:captcha}
<input type="submit" class="register" value="Register">
{/exp:smsp:form}
```

Show entries for the subscribed plan - Now

plan - New

If you want to restrict the channel entries based on the subscribed plan of a logged-in user, you can do this.

```
{exp:channel:entries entry_id="{exp:smsp:get_entries}" parse="inward"}
{title}
{/exp:channel:entries}
```

{exp:smsp:get_entries} tag will return the entry ID of the channels the plan created for. You can control the entries by selecting Channels, Member groups, and categories during the Plan creation.

Switch plan - New

If a user wants to switch to another plan then he will cancel the last plan and subscribe to a new plan. this can be done with a single click. here is the example.

```
{exp:smsp:form plan_id='{segment_3}' previous_plan="{segment_4}" upgrade_plan="yes" prefix='smsp' return="smsp/confirmation" payment_confirmation="smsp/ppexpressconfirmation" cancel="smsp/cancel"}
The fields will be the same as are in the checkout form
{/exp:smsp:form}
```

All the parameters and fields are the same as the

checkout plan **upgrade_plan="yes"** parameter is mandatory to switch the plan otherwise the page can misbehave.

Changelog

V1.0.0

V1.0.0

- **Initial Version**