



# Programmer Manual

**xFlow v5.x**

Created by:  
Envex Developments Inc.  
<http://www.envex.com/>

Version 1.1  
April 24<sup>th</sup>, 2005

# Table of Contents

- 1.) [Overview](#)
- 2.) [File and Directory Structure](#)
- 3.) [Global Variables](#)
- 4.) [Database Modification Variables](#)
- 5.) [Transaction Variables](#)
- 6.) [Geneology Variables](#)
- 7.) [Example Scripts](#)

# 1.) Overview

xFlow v5.x allows software developers to integrate third party applications with xFlow. When one of many actions occur, xFlow will check to see if a third party script exists for that action, and if exists, will automatically execute the script. This allows for a great amount of functionality, and almost complete flexibility of xFlow.

This manual will explain to programmers exactly how to integrate third party scripts into xFlow. Please note, all third party scripts must be written in Perl. xFlow places no restrictions on the third party scripts, and executes them within an `eval {}` block, allowing for full usage of Perl.

This manual is only meant for experienced Perl programmers, who have a need to integrate third party applications into the xFlow.

**NOTE:** Customer support does not provide any technical support for any problems in creating or maintaining third party scripts.

## 2.) File and Directory Structure

All third party scripts must be uploaded into the `/xflow_data/scripts/` directory of the web server. Every time one of many actions occur, xFlow will check to see if a script exists at the needed location, and if exists, will automatically execute that script. The below table lists the exact location where all scripts must be placed, and the action which will execute them.

All locations listed in the below table are relative to the `/xflow_data/scripts/` directory. For example, if the location is `"/database/create.pl"`, the actual location of the script is `/xflow_data/scripts/database/create.pl`.

Action	Script Location	Description
New member created	<code>/database/create.pl</code>	When a new member is added to the database.
Member deleted	<code>/database/delete.pl</code>	When any member is deleted from the database.
Member activated	<code>/database/activate.pl</code>	When any member in the database, is activated.
Member deactivated	<code>/database/deactivate.pl</code>	When any member in the database is deactivated.
Member type changed	<code>/database/member_type.pl</code>	When the member type of any member changes.
Transaction added	<code>/transaction/[TRANSID].pl</code>	When a new transaction is placed into the database, on any member's account. The <code>[TRANSID]</code> represents the ID# of the transaction which was added. ( <i>explained below</i> )
Downline position created	<code>/geneology/create/[PROGRAMID].pl</code>	When a downline position is created in the <code>PROGRAMID</code> program structure.
Downline position deleted	<code>/geneology/delete/[PROGRAMID].pl</code>	When a downline position is deleted from the database, in the <code>PROGRAMID</code> program structure.
Matrix filled	<code>/geneology/matrix_fill/[PROGRAMID].pl</code>	When a member fills their matrix with the maximum number of members, in the <code>PROGRAMID</code> program structure.

As you have noticed, the location for all transaction scripts is `/transaction/[TRANSID].pl`. To determine `[TRANSID]`, go to the *Transaction->Transaction Settings* menu of the Admin Control Panel. Next, click on the appropriate *View Details* link, to find the needed transaction. The table on the next page contains an ID column, which is the `[TRANSID]`. For example, if the ID# of the transaction is 108, the location of the script would be `/transaction/108.pl`.

The same goes for the *[PROGRAMID]* in the genealogy scripts. To determine *[PROGRAMID]*, simply go to the *Setup->Program Settings* menu of the Admin Control Panel. A table will be displayed, which includes the ID# of every program structure.

## 3.) Global Variables

This section explains the global variables, which are available to all third party scripts, regardless of the action which was performed.

### 3.1 The %PROFILE hash

With every action, the %PROFILE hash will be available to the third party script. This hash will contain the member's profile, which the action is being executed against. For example, if a member is being created or deleted, the hash will contain the profile of the member being created / deleted.

The only keys in the %PROFILE hash will be the fields of the member database, and the values being the member's personal profile. For example, if you wanted the member's username, you would use the `$PROFILE{username}` variable, or if you wanted the member's ID#, you would use the `$PROFILE{id}` variable.

Please note, all keys of the hash are in lower case, and all spaces in the database fields are replaced with underscores "\_". For example, in order to retrieve the "First Name" of a member, you would use the `$PROFILE{first_name}` variable.

### 3.2 The %EXTRA hash

The %EXTRA hash will also be available to all third party scripts executed, regardless of the action. This hash contains additional information on the member, such as the status, member type, and default program structure. The below table describes all variables available within the %EXTRA hash.

Variable	Name	Description
<code>\$EXTRA{status}</code>	Member status	A one digit number, representing the status of the member. Will be one of the following: <ul style="list-style-type: none"><li>• 1 – Active</li><li>• 2 – Inactive</li><li>• 4 – Pending</li></ul>
<code>\$EXTRA{sponsor}</code>	Placement sponsor	The ID# of the member's placement sponsor
<code>\$EXTRA{program}</code>	Default program structure	A number, representing the ID# of the default program structure the member is currently in. You can see the ID# of all program structures through the <i>Setup-&gt;Program Settings</i> menu of the Admin Control Panel.
<code>\$EXTRA{member_type}</code>	Member type	A number, representing the ID# of the member type, the member is currently assigned. You may see the ID# of all member types through the <i>Setup-&gt;Member Settings</i> menu.

\$EXTRA{'join_date'}	Date joined	The date the member was added to the database. Formatted in YYYY-MM-DD.
\$EXTRA{'join_time'}	Time joined	The exact time the member was added to the database. Formatted in HH:MM:DD.

## 4.) Database Modification Variables

As explained in [Section 2. File and Directory Structure](#), xFlow will automatically execute a third party script when one of the following actions are taken:

- New member created
- Member deleted
- Member activated
- Member deactivated
- Member type changed

Apart from the global variables, explained in [Section 3. Global Variables](#), the only additional variables available are when the member type is changed. These additional variables are:

Variable	Name	Description
<code>\$EXTRA{'old_type'}</code>	Old member type	A number, representing the ID# of the member type, which was assigned to the member, before the member type was changed.
<code>\$EXTRA{'new_type'}</code>	New member type	The ID# of the member type, which the member was changed to. Same as the <code>\$EXTRA{'member_type'}</code> .

Other than the global variables, and two above mentioned variables, no other information is available to third party scripts, when any database modification action takes place.

## 5.) Transaction Variables

Every time a new transaction is added to the database, regardless of the status, xFlow will check to see if a third party script exists for that transaction, and if needed, execute the script. On top of the global variables explained in [Section 3. Global Variables](#), the %TRANS hash is also available, which contains the following variables:

Variable	Name	Description
\$TRANS{'id'}	Transaction ID#	The unique ID# of the transaction
\$TRANS{'userid'}	Member ID#	The ID# of the member who has been assigned this transaction. Same as the \$PROFILE{'id'} variable.
\$TRANS{'index_id'}	Transaction Index ID#	The ID# of the transaction from the index, which can be found through the <i>Transaction-&gt;Transaction Settings</i> menu.
\$TRANS{'recurring_id'}	Recurring ID#	If a recurring transaction, the unique ID#, pointing to another database table, which contains additional information on the recurring transaction, such as next payment date, ect..
\$TRANS{'amount'}	Amount	Amount of the transaction
\$TRANS{'status'}	Status	A one digit number, representing the status of the transaction. Will be: <ul style="list-style-type: none"> <li>• 1 – Approved</li> <li>• 2 – Declined</li> <li>• 3 – Error</li> <li>• 4 – Pending</li> <li>• 5 – Expired (pending)</li> </ul>
\$TRANS{'method'}	Method	The transaction method, which will be: <ul style="list-style-type: none"> <li>• 1 – Deposit</li> <li>• 2 – Withdraw</li> </ul>
\$TRANS{'coupon'}	Coupon Code	If a coupon code was used, this variable will contain information on the coupon. Formatted in <b>T,A</b> , where: <ul style="list-style-type: none"> <li>• T – Discount Type</li> <li>• 1 – Dollar Amount</li> <li>• 2 – Percentage</li> <li>• A – Amount Discounted</li> </ul>
\$TRANS{'date'}	Transaction Date	The date the transaction was added to the database. Formatted in YYYY-MM-DD
\$TRANS{'time'}	Transaction Time	The exact time the transaction was added to the database. Formatted in HH:MM:SS
\$TRANS{'payment_id'}	Payment ID#	If payment was required, the ID# of the payment method used.

<code>\$TRANS{'merchant_ref'}</code>	Merchant Reference #	If payment was required, the reference number assigned by your payment processor. If your payment processor ever needs to be contacted regarding the transaction, this is the number to give them.
<code>\$TRANS{'reference'}</code>	Reference ID#	If this transaction is referenced to another transaction, the ID# of the referenced transaction.

**NOTE:** Be sure to check the `$TRANS{'status'}` variable in your script! xFlow will execute the script every time the transaction is added, regardless of the status.

If the transaction is part of a recurring transaction, and the `$TRANS{'recurring_id'}` variable is greater than 0, the `%TRANS_RECURRING` hash will also be available, which includes the following variables:

Please note, to save space, `$TRANS_RECURRING` has been replaced with `$REC` in the below table. So, if you want the 'start\_date' variable, you would actually use `$TRANS_RECURRING{'start_date'}`.

Variable	Name	Description
<code>\$REC{'id'}</code>	Recurring ID#	The unique ID# of the recurring transaction.
<code>\$REC{'status'}</code>	Status	A one digit number, representing the status of the recurring transaction. Will be: <ul style="list-style-type: none"> <li>• 1 – Active</li> <li>• 2 – Inactive</li> <li>• 3 – Cancelled</li> <li>• 4 – Pending</li> </ul>
<code>\$REC{'amount'}</code>	Amount	The amount of the recurring transaction. Please note, this is not applicable for recurring commission transactions, as the amounts are always taken from the transaction index.
<code>\$REC{'start_date'}</code>	Date Started	The date which the recurring transaction was first added to the database. Formatted in YYYY-MM-DD
<code>\$REC{'start_time'}</code>	Time Started	The exact time which the recurring transaction was added to the database. Formatted in HH:MM:SS
<code>\$REC{'lastpay_date'}</code>	Last Payment Date	The date which the recurring transaction was last processed. Formatted in YYYY-MM-DD
<code>\$REC{'lastpay_time'}</code>	Last Payment Time	The exact time which the recurring transaction was last processed. Formatted in HH:MM:SS
<code>\$REC{'nextpay_date'}</code>	Next Payment Date	The next date which the recurring transaction will be processed again.

		Formatted in YYYY-MM-DD
\$REC{'nextpay_time'}	Next Payment Time	The time which the recurring transaction will be processed again. This variable is basically useless, as recurring transactions are only processed once every 24 hours.

## 6.) Geneology Variables

There are also special variables available to the different geneology actions as explained in [Section 2. File and Directory Structure](#), which are:

- Downline position created
- Downline position deleted
- Matrix filled

The %GENEOLOGY hash is available to the above three actions, and contains the following information:

Variable	Name	Description
\$GENEOLOGY{'program_id'}	Program ID#	A number representing the ID# of the program structure. You may view the ID# of all program structures through the <i>Setup-&gt;Program Settings</i> menu.
\$GENEOLOGY{'position_id'}	Position ID#	A unique number assigned to every downline position.
\$GENEOLOGY{'phase'}	Phase	The phase of the program structure. For example, if the member has cycled through a matrix three times, this variable would be 3.

## 7.) Example Scripts

The below script is located at */xflow\_data/scripts/database/create.pl*, and is executed every time a new member is created. The script saves the member's username, e-mail address and password to a text file, if the member is active

```
#!/usr/bin/perl

## Make sure member is active
exit(0) unless $EXTRA{'status'} == 1;

## Gather needed information
$newline = join ":", $PROFILE{'username'}, $PROFILE{'email'}, $PROFILE{'password'};

## Append information to text file
open FILE, ">>results.txt";
print FILE "$newline\n";
close FILE;

## Exit
exit(0);

## Add a 1 to ensure no errors
1;
```

The below script is located at `/xflow_data/scripts/transaction/113.pl` and is automatically executed every time the transaction ID# 113 is added to the database. The script adds the transaction ID#, amount, and member username, to a text file, only if the transaction is approved.

```
#!/usr/bin/perl

## Make sure member is active
exit(0) unless $TRANS{'status'} == 1;

## Gather needed information
$newline = join ":", $TRANS{'id'}, $TRANS{'amount'}, $PROFILE{'username'};

## Append information to text file
open FILE, ">>results.txt";
print FILE "$newline\n";
close FILE;

## Exit
exit(0);

## Add a 1 to ensure no errors
1;
```