FairPay

Documentation

Current Version: 01.02.02
Released: 01/28/05

Fairlight Consulting is a proud participant in the

PayPal Developer Network

We offer not only this core integration kit, but can code custom modules to suit your needs. Contact us with any specific inquiries.

Table of Contents

Introduction

FairPay is an enterprise-class integration kit that assists with integrating PayPal with any solution you need to program, utilising PayPal's IPN (Instant Payment Notification) functionality. This IPN technology lets you receive information about PayPal transactions "behind the scenes" so that you may take further automated action based on what transpired for any given transaction. Their technology is quite robust, occurs within seconds, and will retry upon failure to contact your site with the details. Of their solutions along these lines of integration, we at Fairlight Consulting are convinced it's the "best way to fly".

FairPay is not limited to being used only for shopping carts, or only for auctions. It handles every type of IPN transaction documented by PayPal in their Integration Guide. Using FairPay, you can very easily integrate PayPal into any solution your business or organisation may need, including:

- shopping carts
- custom store-fronts

- download access activation upon receipt of online payment
- tracking mass payments of affiliates or employees
- subscription management
- adjusting bookkeeping

...and much more!

If PayPal generates IPN data for a transaction, FairPay makes it easy for you to access all relevant data associated with it.

FairPay is small, fast, and robust. It installs and can be configured easily in about five minutes, and what you do with it from there is up to you. Your possibilities for use of FairPay are unlimited because you can use any language you choose to facilitate your tasks in a very modular fashion.

FairPay consists of two programs: *flipna* and *fairpay_sql_populate*. The *flipna* program handles all data receipt and validation of IPN's and stores the resultant data in an XML file. It then runs whatever commands you tell it to run upon receipt of an IPN. The *fairpay_sql_populate* module that is supplied will populate SQL relational database tables with all information specific to the transaction. This module will usually be the first command you run if you wish to track and maintain your IPN data locally in a database. Any "business rules" programs and their attendant logic (ie., creating an online account, or automating printing shipping labels) falls under business rules, and is written by the developer. However, you simply use a nicely laid-out database relation to access the relevant data for the specific transaction. The grunt-work of tracking all your information and making it available in an easy-to-utilise form is handled for you by FairPay, leaving you free to handle your specific operations rather than reinventing the wheel.

FairPay is modularly designed, so adding and removing modules is a trivial task. Each module can be written in any language you like. You might start off simply activating accounts for people to download software from your web site once they've paid. Later on, you can add another module that tracks and handles subscriptions for you, and simply "plug it in" as a separate module. Because FairPay handles all transaction types, you can dynamically enable or disable any part of your enterprise at any given time, making it an utterly flexible solution. You don't have to start out with everything designed. You can start with one modular application and add more as you need them. Your business rules can change without having to redesign one monolithic program, giving you flexibility and ease of mind knowing that your current solution will simply be augmented by whatever you add, not potentially harmed by an entire redesign of one single module or application. The engine remains the same, but you can change everything else at will. That is one of FairPay's strengths.

FairPay currently runs on Windows or any Unix-like system that can run perl 5.6 or higher. The SQL population module supports MySQL, PostgresSQL, and MS-SQL (the last is supported via ODBC).

Feature List

- Supports **all** PayPal transaction types
- Supports the following databases for data population with the *fairpay_sql_populate* module:
    - MySQL
    - PostgresSQL
    - MS-SQL (via ODBC)
- Comes with six required database tables for tracking your IPN data
- FairPay's tables may reside within *any* database, so it's ideal for use even in an ISP setting where you may only be provided one database
- Provides output to your applications in XML with a well-defined DTD, if access to the data directly from a file rather than (or in addition to) a database is desired
- Supports unlimited commands per transaction
- Supports business logic modules written in any language
- Supports access logging (unlimited, rolling, or disabled)
- Supports session-count limiting of command execution, with fully configurable retries
- Supports user-defined environment variables
- Designed with system security in mind
- Source code immediately available for modifications, if necessary or desired

---

Requirements

This package requires Perl version 5.6 or higher to run.

The *flipna* program requires the Perl CGI module (CGI.pm) and the Crypt::SSLeay module to be installed on your system to operate correctly:

In addition, to utilise the *fairpay_sql_populate* program, you must also have the perl XML::Parser module, the DBI module and the relevant one or more of the following perl modules installed:

- DBD::mysql
- DBD::Pg
- DBD::ODBC

You will also need a web server configured to allow CGI execution.

FairPay has been tested on Solaris, Linux, SCO UNIX OpenServer 5.0.6, and Windows 2000. It should encounter no difficulties on any platform on which perl will compile and run correctly.

Licensing

It should be noted that there is no demo version of FairPay, nor will there be a "Lite" version.

We will provide whatever pre-sales assistance is required to demonstrate whether FairPay is the right product for your needs. If you have questions or comments, please direct them to sales@fairlite.com without hesitation, and we will answer as promptly as possible.

We realise that software is an investment, and hope that you preview the documentation available for this product and make sure that it will suit your needs before purchasing a license. We also hope you avail yourself of the opportunity to ask any pre-sales questions you might have. All license sales are final and non-refundable.

FairPay is licensed at a cost of **$495.00 USD** *per domain name under which it is used, per machine that serves said domain*. This means that if you have *www.widgets.com* and use three servers to load-balance your domain's web traffic, you require three FairPay licenses. If you also wish to serve *www.sprockets.com*, you would need an additional license for each machine which served that domain.

Bulk discounts may be negotiated for purchases of more than five copies at the same time. Special pricing is available for ISP deployment.

Each license fee entitles you to use FairPay under one domain name on one machine, in any role you require. You may modify the program to further suit your needs, and are under no obligation to release changes back to Fairlight Consulting. However, derivative works and/or modified versions **may not** be resold or otherwise distributed. Similarly, you may not copy FairPay, modify it, and run the altered version on another machine. You must purchase another license to use it in any form, altered or otherwise, on an additional machine. You may use an altered version and the original version on the same machine under a single license, however. ***"Machine" shall be defined as one instance of an operating system, for the purpose of this license. Machines which run multiple concurrent operating systems (virtual machines) count as multiple machines, and require additional licenses for each instance.*** The licensee agrees to keep the source code confidential and protected.

Please note that you may utilise more than one copy of FairPay per single domain, per server **for testing purposes only**. This would apply in the case of using it for live work as well as with PayPal's Sandbox concurrently. However, you may **not** use this special provision (intended for testing purposes) to "get around" the need for more licenses. You should not be running copies of FairPay for more than one organisation, all from inside one domain. The intent of the license is that licenses are counted by both domain names and machines which serve them, but that one domain name equals one organisation, except in the case of special ISP licensing.

Upon receipt of payment for a license, access to the program will be generated for the licensee, and such information shall be delivered to the email address associated with the PayPal payment.

Upgrades for the product are *currently* free when moving to new minor and major versions. Fairlight Consulting reserves the right to change this policy in the future, with no prior warning.

There is ***no warranty*** for this software. This software is offered **"AS-IS"** and without warranties as to performance or merchantability or any other warranties, whether expressed or implied.

Good computing practice dictates that any program should be thoroughly tested with non-critical data before deploying it for production. The user assumes the entire risk of using the program. In no event shall Fairlight Consulting be held liable for loss of data, failure of performance, or any other damages, be they real or perceived.

If you agree to these terms, click here to order FairPay!

Already registered and have your account information?
Download FairPay!

Support

Extended (non-bug-related) support for FairPay is available at our standard hourly rates. Because we offer pre-sales assistance in determining if FairPay is right for your needs, and because documentation is readily available, anything not covered by either of these is deemed an at-cost support issue.

You may request *any* kind of technical assistance with FairPay by sending email to fairpay@fairlite.com, including bug reports and feature requests.

Feature requests may be commissioned for special functionality, if desired. Any non-commissioned requests are subject to being implemented soley at the discretion of Fairlight Consulting. This may include not being implemented at all, depending on how useful we think the feature would be in general. Commissioned requests can be price-negotiated based on whether the features requested are allowed to be re-integrated into the main product, or whether they shall remain proprietary and exclusive to the commissioning party.

We also offer consulting on how to achieve specific results using the product through this support mechanism, and would be happy to assist you in this regard.

Program Flow

The logic flow of FairPay is straightforward, and is as follows:

1. A PayPal transacton is sent to your *flipna* URL as configured in your PayPal account profile.
2. The web server (httpd) starts *flipna*.
3. The *flipna* program confirms the validity of the transacton with PayPal's servers.
4. The *flipna* program then writes an XML file into a spool directory, and then executes all commands listed in the *commands* file, in order. If the session count was exceeded for the configurable amount of retry time, the file is left in place. If all operations succeed, the XML file is removed after command execution.

   This command execution phase is where database population is achieved with *fairpay_sql_populate* or the program of your choice. You may insert other programs afterwards to handle "business rules" operations.

FairPay DTD

Should you ever need to directly access the XML files that *flipna* generates, the FairPay DTD for incoming data is as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE form_submission [
<!ELEMENT form_submission (form_field*)>
<!ELEMENT form_field (field_name,field_value+)>
<!ELEMENT field_name (#PCDATA)>
<!ELEMENT field_value (#PCDATA)>
<!ATTLIST form_field type (file|value|cookie|textarea)
"value">
<!ATTLIST form_field orig_filename CDATA #IMPLIED>
<!ATTLIST form_field size CDATA #IMPLIED>
]>
```

There are several things to note regarding how FairPay presents data:

- All fields that are submitted will be reported, even if empty. In the case of empty fields, the *field_value* element will be an empty container.
- All *field_value* contents will be encapsulated inside the **<![CDATA[]]>** syntax, so as to preserve any possible SGML input that might result from form data. In the case of empty containers, the CDATA block will not be present, and the container will truly be empty.

The type attributes **file**, **cookie**, and **textarea** are not used at this time. Currently, all fields are of type **value**.

Installation

Pre-Installation Steps

Before you install FairPay, please make sure you take the following steps.

1. Make sure you have Perl's **DBI** module installed.
2. Make sure you have at least one of the following Perl modules installed. if you wish to use *fairpay_sql_populate*:
    - **DBD::mysql** (for MySQL)
    - **DBD::Pg** (for PostgresSQL)
    - **DBD::ODBC** (for MS-SQL)
3. If you're using ODBC, make sure you have a DSN ready for use during installation.

Installation Steps

Installing FairPay is very straightforward, and involves a few simple steps.

FairPay's main program, *flipna*, should reside in a directory that has CGI execution enabled in your httpd configuration. The program's working directory, which will be named *fairpay* should reside somewhere **outside** of the web document space on your server for reasons of security.

If you are installing on *nix, make sure you log in as or su to the same user as should be running FairPay's component programs, prior to installation or any (re)confguration. If you do not, you will need to manually adjust permissions and ownerships accordingly.

To install FairPay, follow these steps:

1. Copy the distribution tarball to a directory where you have write permission.
2. Untar the distribution tarball.

3. Change directory into *fairpay-install*.
4. Run **perl ./install.pl** to begin the installation process.
5. Answer the location questions. **Only populate ODBC tables if you are using the ODBC driver.**
6. Adjust the ownership of the files and directories under FairPay's directory tree if necessary.
7. If you are not using the ODBC driver, but are using the MySQL or PostgresSQL servers, please use the *complete_mysql.sql* or *complete_pgsql.sql* files, respectively, to populate the server and database of your choice with the correct tables.
8. Proceed to the section on configuration.
9. After configuration is complete, configure the IPN section of your PayPal account profile to point to the URL for your copy of *flipna*

Upgrading

To upgrade FairPay, follow the steps for installation.

Please note that FairPay's installer will overwrite the old program file, but your entire *fairpay* directory tree will remain untouched by the process. All existing configuration will be retained.

Despite this fact, good computing practice dictates that one should make a backup of one's configuration before performing an upgrade. It is not necessary, but it *is* wise.

Configuration Instructions

In this section we will walk you through the relatively simple tasks involved in configuring FairPay for use.

Before we begin, it should be stated that in the following sections, the files mentioned reside in the main *fairpay* directory.

Environment File

Global environment variables that your commands require may be set by using the *environment* file. The presence of this file is optional.

The format of this file is identical to Bourne shell syntax, with one **VAR=value** assignment per line. There are a few exceptions to the Bourne-like quality of the syntax. Exports are not needed. FairPay handles this for you, as the file is not processed by a shell. Comments may be on individual lines, and blank lines are ignored. You may *not* put comments after value contents. The other notable difference is that you do not need to quote your values. Any quotes included (be they single or double) will be literally interpreted as part of the value for your variable.

Configuration File

The file *config* controls global behaviour for FairPay's handling of all applications. The presence of this file is optional, and if it is absent or empty, default values will be used.

All available options are described in detail on the Configuration Options page. The format of this file is one **option=value** assignment per line. Comments may be on lines by themselves, and blank lines are ignored.

Commands File

The heart of your interaction with FairPay, the file *commands* controls what commands are executed by *flipna*. This file **must** be present, and configured with the list of your commands for the program set.

The syntax allows for blank and separate comment lines. You may *not* place comments on the same line as a command. Each command should be on a separate line, and each will be processed in order at execution.

You may use any valid command on your system, and shell metacharacters (redirects, etc.) *are* allowed. Please note that FairPay clears the *$PATH* environment variable at the beginning of its execution, so you effectively have no path unless you set one in either the main or program set environment variable files. If you do not define a path, you will need to explicitly state full paths to your commands.

There are three "magic cookies" that allow you to give your commands access to the relevant files and data associated with a FairPay transaction. Wherever these tokens appear in your commands, they will be replaced with the appropriate pathname.

- **%i** - This token represents the XML data file generated by FairPay. This file contains the IPN data.
- **%l** - This token represents the internal lock number FairPay uses for your current transaction.
- **%n** - This token represents a unique ID that is used by both *fairpay_sql_populate* and any of your other applications and commands in your command list. The ID that this token will expand to is the value for which you would search for in the **flt_id** field within the database tables. You may alto use this ID to create transaction-specific temporary files between commands.

The *fairpay_sql_populate* program should always be called with **-x %i** as arguments, as this tells the program from which data file to populate database entries.

Should you be unable to utilise the command line to pass filenames or the internal lock number to your application, an alternate method is available in the form of environment variables. These do not go in the configuration file, but instead may be accessed from within your application:

- **FLIPNA_FILE** - Identical to the file that **%i** denotes in the commands file.
- **FLIPNA_LOCK_NUMBER** - Identical to the number that **%l** denotes in the commands file.
- **FLIPNA_ID** - Identical to the program name that **%n** refers to in the commands file.

Configuration Options

The following options are valid for FairPay's configuration files:

max_session_count

This option dictates how many instances of *flipna* may be executing external commands at the same time. This options is to prevent license user counts from being exceeded in any software with license managers, as well as to help throttle the load on a very busy system.

wait_interval

When *flipna* gets to the phase where it executes external commands, it performs a check to make sure that no more than a configurable number of instances of itself are doing so at the same time. If it detects that number are already running, it will retry the condition a configurable amount of times. This option sets the granularity for these retries. The argument is an integer expression denoting seconds. The default for this option is **1** second.

wait_attempts

The companion to *wait_interval*, this option denotes the number of retries to attempt before giving up and leaving the spool XML data file for the transaction in place for later perusal. The formula for total time spent retrying is *wait_interval* * *wait_attempts*. The default for this option is **600** attempts (10 minutes, given the default setting for *wait_interval*).

apache2

This option **must** be enabled with the Apache 2.x servers. Apache 2.x kills child processes whenever they close their main three file descriptors. This results in stranded files which would be cleaned under other web servers such as Apache 1.3.x. Set to **yes** if you are using Apache 2.x. Otherwise set to **no**, which is the default. If you set this to **yes**, please make sure your commands in the *commands* file are redirected somewhere harmless (/dev/null in *nix, for instance).

sandbox

This option dictates whether *flipna* will run its transaction validations against the PayPal Sandbox at PayPal Developer Central, or whether it will run its transaction validations against the live PayPal site. A setting of **yes** activates PayPal Sandbox testing mode. A setting of **no** enables live mode against the real site. (**Note: As detailed in the official PayPal Integration Guide, transactions made in the Sandbox cannot be verified against the live site, and vice versa.**

logging

This option dictates whether *flipna* and *fairpay_sql_populate* will log only fatal errors, or whether they'll log all transactions. A setting of **yes** enables logging of all transactions. The filenames in the *fairpay* directory are **FLIPNA.LOG** for *flipna* and **FAIRPAY_SQL.LOG** for *fairpay_sql_populate*.

log_size

This option sets the maximum number of lines in each logfile. If set to the word *unlimited* (or omitted), logs may grow infinitely. If set to an integer, logfile lines are kept to this many lines, with the oldest entries that exceed this threshhold being eliminated.

database_type

This option specifies which driver *fairpay_sql_populate* should use. The three supported options are **MySQL**, **PgSQL**, and **ODBC**.

database_host

This option tells *fairpay_sql_populate* which host to use for database population. A hostname or IP# is allowed.

database_name

This option tells *fairpay_sql_populate* which database name to use for database population. The database should contain the FairPay tables prior to use of.*fairpay_sql_populate*.

database_user

This option tells *fairpay_sql_populate* what username to use for database population.

database_pass

This option tells *fairpay_sql_populate* the password to use for database population.

FairPay Logs

FairPay logs its activities in the *fairpay* directory, and each program has its its own log. The log format is event-style, one line per event of any program. All program events are grouped together, between a beginning and end-of-processing pair event lines.

The format of any given line is:

- The program name as listed in the filesystem
- The process ID in square brackets
- The full date and time followed by a colon
- The text for the event.

Logging has three states: on, off, or rolling. If you set **logging** to on, logging will be performed. If you set it to off, only transactions with fatal errors will be logged. If you set **log_size** to an integer, the log will roll off old entries when it reaches that number of lines. The **log_size** will default to *unlimited* by default.

Database Schema

FairPay's *fairpay_sql_populate* program utilises six relational database tables for storage of IPN data from PayPal. In the tables below, SQL character data is labeled as "char", and numeric types are labelled as "decimal" or "integer", depending on which is appropriate. This is to abstract the data types for general use, independant of which database type you're using, as the low-level engine types differ between engines.

NOTE: *Always* check the *flt_verified* field for any transaction. Even if a transaction is shown as *INVALID*, it will be stored, along with all its details, both on the off-chance that there is a bug that caused a false negative, and so that you can track attempted payment spoofs. In your business rules, only actually financially process transactions marked as *VERIFIED*

In any table, any field for which there is no data will be a **NULL** field.

The layout of the tables is as follows, accompanied by explanations of what contents may be expected within each column:

fairpay_transactions - The heart of the FairPay database relation system, the majority of your transaction information will end up in this table. Any IPN transaction that has a tranaction ID will have a record here.

Column Name	Type/Length	Information and Details
txn_id	char (17)	A unique ID designated by PayPal to identify the transaction.
parent_txn_id	char (17)	The unique ID of the transaction for which the current transaction is a child. The presence of this ID usually indicates a refund or other kind of reversal.
txn_type	char (14)	Type of transaction. There are five types: • web_accept • cart • send_money • masspay • subscr_payment **Reversals will *not* have a transaction type.**
payment_type	char (7)	Either **instant** or **echeck** depending on the method of payment.
payment_date	char (25)	Date of payment. The format is: HH:MM:SS MON X, YYYY TZN
payment_status	char (17)	One of several results: • Cancelled_Reversal - Reversal was cancelled. • Completed - Initial payment completed, funds available. • Denied - Merchant (you) denied the transaction. • Failed - Payment failed. (Buyer bank account had insufficient funds.) • Pending - Payment pending. See **pending_reason**. • Refunded - Merchant (you) refunded the transaction. • Reversed - Chargeback or other kind of reversal. See **reason_code** as well. • Processed - Mass Payment has been processed but not yet completed (this result is for a **txn_type**of *masspay* only).
pending_reason	char (14)	One of several results: • address - No confirmed shipping address as specified by merchant's preferences. • echeck - Payment made by an eCheck which has not

		yet cleared. • intl - Merchant is a non-US account and has no withdrawal method. • multi_currency - Currency was not native and merchant does not have the conversion option enabled. • unilateral - Payment made by non-confirmed or non-registered email address. • upgrade - Payment made to an account that must be upgraded to Premiere or Business status. • verify - Merchant account is not yet verified. • other - Other reason for pending payments.
reason_code	char (14)	One of several results: • buyer_complaint • chargeback • guarantee • refund • other
mc_gross	decimal	The gross amount of payment made in **mc_currency**.
mc_fee	decimal	The fee charged made in **mc_currency**.
mc_currency	char (3)	One of several results: • USD - U.S. Dollars • GBP - British Pounds Sterling • EUR - Euros • CAD - Canadian Dollars • JPY - Japanese Yen
mc_handling	decimal	The amount of shipping paid in **mc_currency**. ***NOTE:*** This is not currently split out on "Buy Now" buttons!
mc_shipping	decimal	The amount of shipping paid in **mc_currency**. ***NOTE:*** This is not currently split out on "Buy Now" buttons!
settle_amount	decimal	The amount deposited after automatic currency conversion.
settle_currency	char (3)	The currency used for **settle_amount**

exchange_rate	decimal	Exchange rate for conversion.
payment_gross	decimal	Gross payment in USD. **This is a legacy field.**
payment_fee	decimal	Fee in USD. **This is a legacy field.**
business	char (127)	Email address of the recipient. Identical to **receiver_email** if payment was made to primary address for the account.
receiver_email	char (127)	Primary email address of the payment recipient.
receiver_id	char (13)	Unique ID assigned by PayPal to the recipient. Equivalent to the recipient's referral ID.
item_name	char (127)	Item name as passed by the merchant.
item_number	char (127)	Item number as passed by the merchant.
quantity	integer	Quantity of item purchased.
invoice	char (255)	Passthrough variable for invoice tracking. Omitted if not supplied by merchant.
custom	char (255)	Custom variables passed to merchant which are never seen by the customer.
memo	char (255)	Memo as entered on PayPal's site in the Note field.
tax	decimal	Tax charged on entire payment.
note	char (4000)	Extra note field.
option_name1	char (64)	Descriptive name of first of two possible (optional) options.
option_selection1	char (200)	Value of first of two possible (optional) options.
option_name2	char (64)	Descriptive name of second of two possible (optional) options.
option_selection2	char (200)	Value of second of two possible (optional) options.
notify_version	char (10)	Version number of PayPal's IPN service.
payer_id	char (13)	Unique ID assigned by PayPal to the payer's account.
num_cart_items	integer	Number of line items present in a shopping cart transaction.
for_auction	char (4)	Set to **true** if transaction is from an auction.
flt_record_processed	char (1)	A one character field to help any business-rules software determine if a record has been processed. Defaults to "N".
flt_created	char (14)	Time the record was created. Format is YYYYMMDDHHMMSS
flt_verified	char (8)	Either **INVALID** or **VERIFIED** depending on whether the IPN was validated by PayPal as authentic and generated by their system.

flt_id	char (64)	Unique ID assigned by *flipna* with which you may look up all records related to one IPN submission.

NOTES:

- The **txn_id** field is used to relate transactions to all other tables **except** *fairpay_buyers*.
- The **flt_id** field is used to look up the main transaction to get to the point where you know what the **txn_id** value is for all tables' parts of any transaction.
- To relate a transaction to a buyer, use the **payer_id** field.

---

fairpay_buyers - Every transaction (even of the Mass-Payment type) has a *payer_id* field. This is unique to every user. Relation between transactions and buyer information should be based on this field. All buyer-specific details will appear in this table, and will be kept as current and complete as possible.

Column Name	Type/Length	Information and Details
payer_id	char (13)	A Unique customer ID assigned to the payer's account by PayPal.
first_name	char (64)	The payer's first name.
last_name	char (64)	The payer's last name.
payer_business_name	char (127)	The business name of the payer, if given.
address_name	char (128)	The name of to be used for the shipping address if Gift Address is provided.
address_street	char (200)	Street address for shipping.
address_city	char (40)	City for shipping.
address_state	char (40)	State for shipping.
address_zip	char (20)	ZIP code for shipping.
address_country	char (64)	Country for shipping.
address_status	char (11)	One of two results: • confirmed • unconfirmed
payer_email	char (127)	Primary email address of the payer.

payer_status	char (10)	One of two results: • verified • unverified
flt_record_processed	char (1)	A one character field to help any business-rules software determine if a record has been processed. Defaults to "N".
flt_created	char (14)	Time the record was created. Format is YYYYMMDDHHMMSS
flt_id	char (64)	Unique ID assigned by *flipna* with which you may look up all records related to one IPN submission. Any time the buyer record is updated with more current information, this will be the last *flipna* ID that triggered such an update.

NOTES:

- The **payer_id** field is used to relate buyers to transactions.
- The **flt_id** field is updated any time a transaction is made with the same **payer_id**.

---

fairpay_auction - Information that is auction-specific will be stored in this table.

Column Name	Type/Length	Information and Details
txn_id	char (17)	A unique ID designated by PayPal to identify the transaction.
auction_buyer_id	char (64)	The eBay auction ID of the buyer. This is their screen name on eBay.
auction_closing_date	char (25)	The closing date of the auction. The format is: HH:MM:SS MON X, YYYY TZN
auction_multi_item	integer	A field that indicates which item number from an auction is being referenced. **NOTE: This field is not currently used. The PayPal Integration Guide is currently incorrect about how auction data is presented via IPN. We have confirmed this with PayPal as of January 25, 2005, and they agree there is a discrepancy between the documentation and the actual functionality. The reality is that all auction line items are presented as an individual transaction with its own *txn_id*, and all the data not in this table is currently tracked in *fairpay_transactions*. The guide says that every IPN from an auction will have the same *txn_id*, *mc_gross*, and *payment_gross*. Again, PayPal has verified for us that this is *incorrect* at this time.**

flt_record_processed	char (1)	A one character field to help any business-rules software determine if a record has been processed. Defaults to "N".
flt_created	char (14)	Time the record was created. Format is YYYYMMDDHHMMSS
flt_id	char (64)	Unique ID assigned by *flipna* with which you may look up all records related to one IPN submission.

NOTES:

None.

---

fairpay_cart - All shopping cart-specific information will be stored in this table.

Column Name	Type/Length	Information and Details
txn_id	char (17)	A unique ID designated by PayPal to identify the transaction.
cart_item_num	integer	The line number of the item in the shopping cart.
tax	decimal	The tax collected on the cart line item, if the item was taxed separately.
item_name	char (127)	The item number of cart line item.
item_number	char (127)	The item name of the cart line item.
option_name1	char (64)	Descriptive name of first of two possible (optional) options for the cart line item.
option_selection1	char (200)	Value of first of two possible (optional) options for the cart line item.
option_name2	char (64)	Descriptive name of first of two possible (optional) options for the cart line item.
option_selection2	char (200)	Value of first of two possible (optional) options for the cart line item.
quantity	integer	Quantity of the cart line item that was purchased.
mc_gross	decimal	The gross amount paid for the cart line item in **mc_currency**..
mc_handling	decimal	The amount of handling paid in **mc_currency**.
mc_shipping	decimal	The amount of shipping of handling paid in **mc_currency**.
flt_record_processed	char (1)	A one character field to help any business-rules software determine if a record has been processed. Defaults to "N".

flt_created	char (14)	Time the record was created. Format is YYYYMMDDHHMMSS
flt_id	char (64)	Unique ID assigned by *flipna* with which you may look up all records related to one IPN submission.

NOTES:

None.

---

fairpay_masspay - All information related to mass payments will be stored in this table,

Column Name	Type/Length	Information and Details
txn_id	char (17)	A unique ID designated by PayPal to identify the transaction.
receiver_email	char (127)	Email address of the payment recipient.
mc_gross	decimal	Gross amount paid in **mc_currency**.
mc_fee	decimal	Fee paid in **mc_currency**.
mc_currency	char (3)	One of several results: • USD - U.S. Dollars • GBP - British Pounds Sterling • EUR - Euros • CAD - Canadian Dollars • JPY - Japanese Yen
payment_gross	decimal	Gross payment in USD. **This is a legacy field.**
status	char (9)	One of several results: • Completed - Payment processed. • failed - Insufficient funds. • reversed - Payee refused payment or unilateral payments were unclaimed for 30 days. • pending - Unilateral payments still pending.
payment_fee	decimal	Fee in USD. **This is a legacy field.**
unique_id	char (17)	Unique ID assigned to this recipient by the merchant during

		the Mass Pay process.
flt_record_processed	char (1)	A one character field to help any business-rules software determine if a record has been processed. Defaults to "N".
flt_created	char (14)	Time the record was created. Format is YYYYMMDDHHMMSS
flt_verified	char (8)	Either **INVALID** or **VERIFIED** depending on whether the IPN was validated by PayPal as authentic and generated by their system.
flt_id	char (64)	Unique ID assigned by *flipna* with which you may look up all records related to one IPN submission.

NOTES:

- Each Mass-Pay record has a matching record in *fairpay_transactions*. You can track the overall **payment_status** of the record as opposed to the Mass-Pay status of the individual transaction (as well as other information) with a simple relation via **txn_id**.

---

fairpay_subscriptions - All subscription information will be stored in this table. This includes IPN transactions that will **not** show up in the *fairpay_transactions* table, as only payments themselves will have transaction ID's, and there are five other types of subscription IPN types.

Column Name	Type/Length	Information and Details
txn_id	char (17)	A unique ID designated by PayPal to identify the transaction.
txn_type	char (14)	One of several results: • subscr_signup - Sign-up • subscr_cancel - Cancellation • subscr_failed - Payment failure • subscr_payment - Subscription Payment • subscr_eot - End of Term • subscr_modify - User modification of subscription
subscr_date	char (25)	Date of subscription. The format is: HH:MM:SS MON X, YYYY TZN
subscr_effective	char (25)	Date on which subscriber modification will become effective. The format is: HH:MM:SS MON X, YYYY TZN

period1	char (10)	Length of first trial period. An integer, followed by a space and a single character: • D - days • W - weeks • M - months • Y - years
period2	char (10)	Length of second trial period. An integer, followed by a space and a single character: • D - days • W - weeks • M - months • Y - years
period3	char (10)	Length of regular subscription period. An integer, followed by a space and a single character: • D - days • W - weeks • M - months • Y - years
amount1	decimal	Amount for first trial period in USD. **This is a legacy field.**
amount2	decimal	Amount for first trial period in USD. **This is a legacy field.**
amount3	decimal	Amount for regular subscription period in USD. **This is a legacy field.**
mc_amount1	decimal	Amount for first trial period in **mc_currency**
mc_amount2	decimal	Amount for second trial period in **mc_currency**
mc_amount3	decimal	Amount for regular subscripton period in **mc_currency**
mc_currency	char (3)	One of several results: • USD - U.S. Dollars • GBP - British Pounds Sterling • EUR - Euros • CAD - Canadian Dollars • JPY - Japanese Yen

recurring	char (1)	Is the subscription recurring at the regular rate (**mc_amount3** and **period3**)? "1" means yes.
reattempt	char (1)	Retry on failed payments? A "1" means yes.
retry_at	char (25)	The date at which a retry will be attempted after a failed payment.
recur_times	integer	The number of payment installments that will occur at the regular rate (**mc_amount3** and **period3**).
username	char (64)	Username generated by PayPal for subscription access, if the account generation feature is enabled.
password	char (127)	Password generated by PayPal for subscription access, if the account generation feature is enabled.
subscr_id	char (19)	Unique ID assigned by PayPal for an individual subscription.
payer_id	char (13)	A Unique customer ID assigned to the payer's account by PayPal.
item_name	char (127)	Item name as passed by the merchant.
item_number	char (127)	Item number as passed by the merchant.
option_name1	char (64)	Descriptive name of first of two possible (optional) options.
option_selection1	char (200)	Value of first of two possible (optional) options.
option_name2	char (64)	Descriptive name of second of two possible (optional) options.
option_selection2	char (200)	Value of second of two possible (optional) options.
flt_record_processed	char (1)	A one character field to help any business-rules software determine if a record has been processed. Defaults to "N".
flt_created	char (14)	Time the record was created. Format is YYYYMMDDHHMMSS
flt_verified	char (8)	Either **INVALID** or **VERIFIED** depending on whether the IPN was validated by PayPal as authentic and generated by their system.
flt_id	char (64)	Unique ID assigned by *flipna* with which you may look up all records related to one IPN submission.

NOTES:

- Only subscription transactions of type *subscr_payment* will generate a matching entry in *fairpay_transactions*. All other subscription transactions appear only in this table.

Security Notes

FairPay was designed with security in mind. While it makes every attempt to be as secure as possible, you have to do your part as well.

The biggest pitfall of CGI programming is trusting external data and using it for system calls without vetting it first. This can be catastrophic, and lead to destruction of data. FairPay does *not* use any remote data in remote commands, nor does it provide you any facility to do so via the program set command file. However, it does not (and can not) prevent you from persuing this extremely hazardous practice within the applications you call from the command file. Those applications could conceivably call other programs unsafely, if you blindly trust data provided from outside your control and use it in a command line context. **DO NOT DO THIS.** Doing so is tantamount to leaving the keys to your car on the hood while you quickly nip into Wal*Mart for even one small item. There's a good chance your car won't be there when you get back. Likewise, bad things will inevitably happen if you commit this grave security error.

FairPay simply passes the data it obtains without parsing it or censoring it. As a generic application, it can not and should not do so. It is up to you to make sure that your application adequately scans and validates the incoming data to ensure it is safe for use in whatever context is applicable.

Security is one of the biggest issues with CGI programming. FairPay does its best to make the environment safe as possible without compromising the data integrity. It is your responsibility to ensure that your applications use the data safely.

For a complete list of points to check over in your integration efforts, please consult this more comprehensive list that we have released to the public to aid in CGI security education.

Testing Tips

Using FairPay is quite easy. It installs and can be configured in a matter of minutes. After that, however, you need to program your business rules. Further, you need to **test** your business rules. This would ordinarily be a tedious process, were it not for PayPal's great testing facilities.

PayPal provides a "sandbox" area which lets you test everything except reversals (like chargebacks), and auctions. To utilise this resource, follow these steps:

1. Utilise the setting **sandbox=on** in your *config* file for FairPay.
2. Go to PayPal Developer Central and create an account there.
3. Enter the Sandbox area

4. Create a business account and attach a bank account to it. **All financial information is pre-populated for testing purposes. No actual financial transactions ever take place in the Sandbox.** Verify the bank account. Also make sure that you verify your account in general. You will have to use the Developer Central webmail interface to view the mail from any Sandbox accounts.

5. Create a personal account and attach a credit card. As above, just accept the pre-populated financial information.

6. Configure your Sandbox business account's IPN settings. Turn IPN on, and enter the URL for your copy of flipna.

7. Start testing. You can make buttons, use Send Money, etc. All transactions in buttons should be sent to **www.sandbox.paypal.com** in your buttons code as pertains to the **webscr** location within buttons.

8. When you're done testing, utilise **sandbox=off** in your *config* file for FairPay. Configure your final solution with buttons that will utilise the live PayPal site (at **www.paypal.com** for all **webscr** locations within the buttons).

Please note that **flt_verified** will be *VERIFIED* for Sandbox transaction IPN's tested against the Sandbox. However, you cannot test Sandbox transaction IPN's against the live system, nor can you test live IPN's against the Sandbox. They are mutually exclusive. If you must test new modules while live ones remain in production, it is recommended that you utilise a second copy of FairPay in another location. This can be done by reinstalling to different locations on the same system using a different location for the *fairpay* directory and a different cgi-bin directory for *flipna*). In this manner, you can have one configuration for testing, and one for live use. This falls within acceptable use within the FairPay license.

Change Log

v01.00.00

Original code base.

---

v01.00.05

Bug fixes on database schema.

---

v01.01.00

Fixed schemas, revised code in some areas.

---

v01.02.00

Incorporated PgSQL and ODBC (MSSQL, Sybase, etc.) into the program. Reprogrammed row testing in all cases to work without using $db_sth->rows method.

v01.02.01 - 01/18/05

Fixed single-quoting in string fields for SQL.

v01.02.01 - 01/26/05

Completed documentation.

v01.02.02 - 01/28/05

Added *apache2* option to handle broken signal generation on closed file descriptors in Apache 2.0.xx (and possibly 2.1.xx).