|  |
| --- |
|  |
| Online Payments |
|  |
|  |
|  |
|  |

# This document is split into 3 sections:

(1) How to determine the fee to charge a user.

(2) Working with your Payment Gateway to record payments/refunds in the system.

(3) How to report activity to the NES Finance system.

1. Determine the Fee to Charge

When the system is extended to take payments, it needs to determine how much to charge the delegate (if at all) on a course by course basis. The charging model allows for 5 different types of charge and each event must be tagged as belonging to one of these five types.

This flag could be held at a course “group” level if all will follow the same model/price point or at a course record level if costs are event specific. Historically it is held at a course record level and is selected by the course administrator before publication:

## Free

Clearly this event is free and that is all the system needs to know. No charge will be levied.

## Delegate Flat Fee

This open event has a charge and everyone booking onto it will be charged the same amount.

* The event will have a price point and that is all that needs assessed when determining the charge to levy.
* This is charging at the simplest level and the length of the event has no consequence to the price. Nor does the user account role.

## Sessional Multiplier

This open event has a sessional charge. A session is defined as a 3 to 4-hour block. *(A whole day event might therefore be thought of as a 2-session event. An evening event might be thought of as a 1-session event).*

* The event will have a sessional price point and one or more sessions associated with it.
* The charge levied will be determined by multiplying count(sessions) \* sessional price point
* This allows the system to automatically increase the charge depending on length of event.

## Role Sessional Multiplier

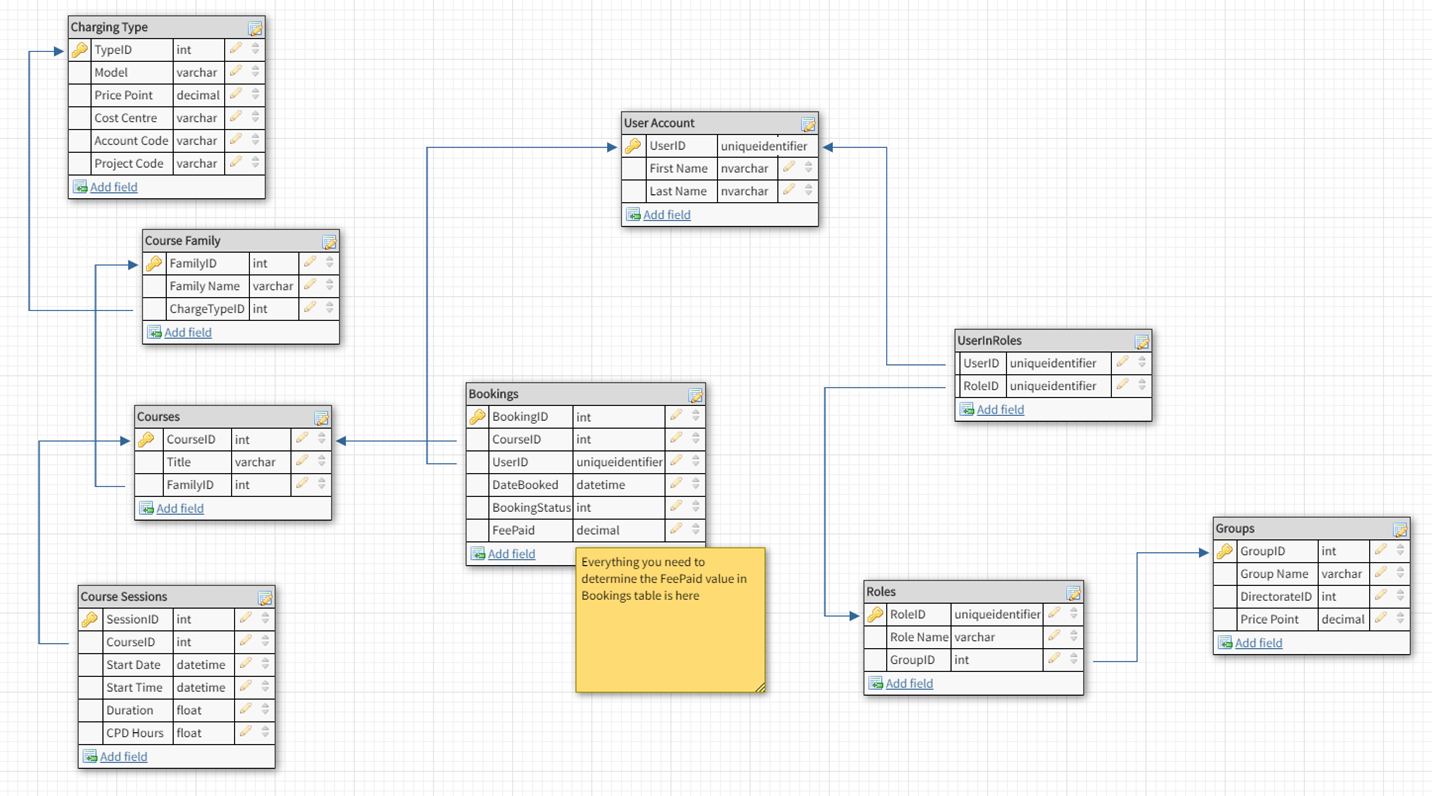
This open event has a sessional charge.

* Each delegate has a role and that role has an associated price point set. Roles are usually grouped (eg there might be a number of Dentist roles, but all exist within the group “Dentist”). The price point is usually associated with the group, not the role.
* The charge levied will be determined by multiplying count(sessions) \* role price point
* It can be seen from this that different roles get charged different amounts for the same event (eg Dentists might well have a higher role price point to Dental Nurses who sit in a different group called “DCP”)
* It also allows the system to automatically increase the charge depending on length of event.

## Group Flat Fee

This is a closed event. It is not published but does exist as an event in the system. It is an event where NES have arranged to deliver a training course to a pre-defined group of staff (usually in their practice) by invitation only.

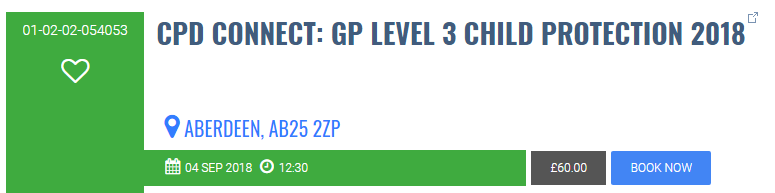
* The event will have a price point and that is all that needs assessed when determining the charge to levy.
* Payment is only taken once, from a pre-determined delegate, on behalf of the entire group
* Once that single payment is taken all delegates on the event are considered “booked and paid”



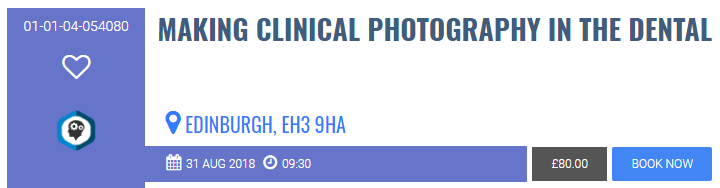
**FYI** This gets us as far as determining what to charge a delegate.

The database is extended further with several Transaction tables that hold actual payment gateway data. This is covered later

Let’s look at 4 examples on NES Portal



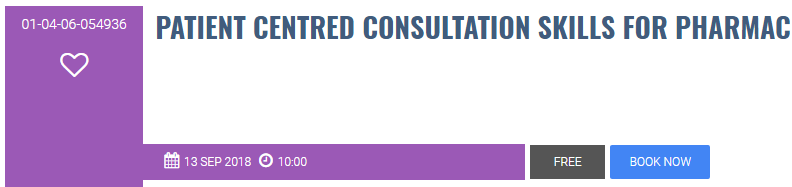
* This is an event that has been set to determine price by **Session Multiplier** (ChargingType.Model identifies this)
* ChargingType.PricePoint has the value of **£60.00** in it
* Count(SessionID)for this CourseID tells us there is only 1 record
* It’s therefore a 1 session event charging **£60.00** to anyone wishing to book it.



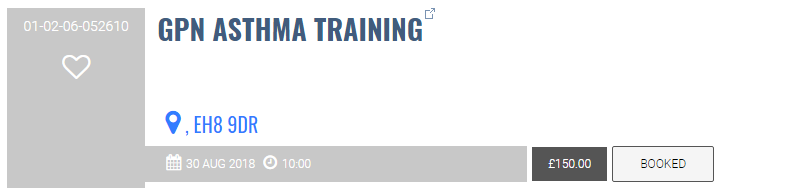
* This is an event that has been set to determine price by **Role Session Multiplier** (ChargingType.Model identifies this)
* ChargingType.PricePoint will be set to £0.00 and should be ignored
* This user has a role of Dentist (SpR). It belongs to the Dentist Group
* Group.PricePoint has the value of **£40.00** in it for the Dentist Group
* Count(SessionID)for this CourseID tells us there are 2 records
* It’s therefore a 2-session event changing £40.00 per session for Dentists
* The course is offered to this user at **£80.00**

Looking at the same event as above but this time with an account type of Dental Nurse…

* This is an event that has been set to determine price by **Role Session Multiplier** (ChargingType.Model identifies this)
* ChargingType.PricePoint will be set to £0.00 and should be ignored
* This user has a role of Dental Nurse. It belongs to the DCP Group
* Group.PricePoint has the value of **£15.00** in it for the DCP Group
* Count(SessionID)for this CourseID tells us there are 2 records
* It’s therefore a 2-session event changing £15.00 per session for Dental Nurses
* The same course is offered to this user at **£30.00**

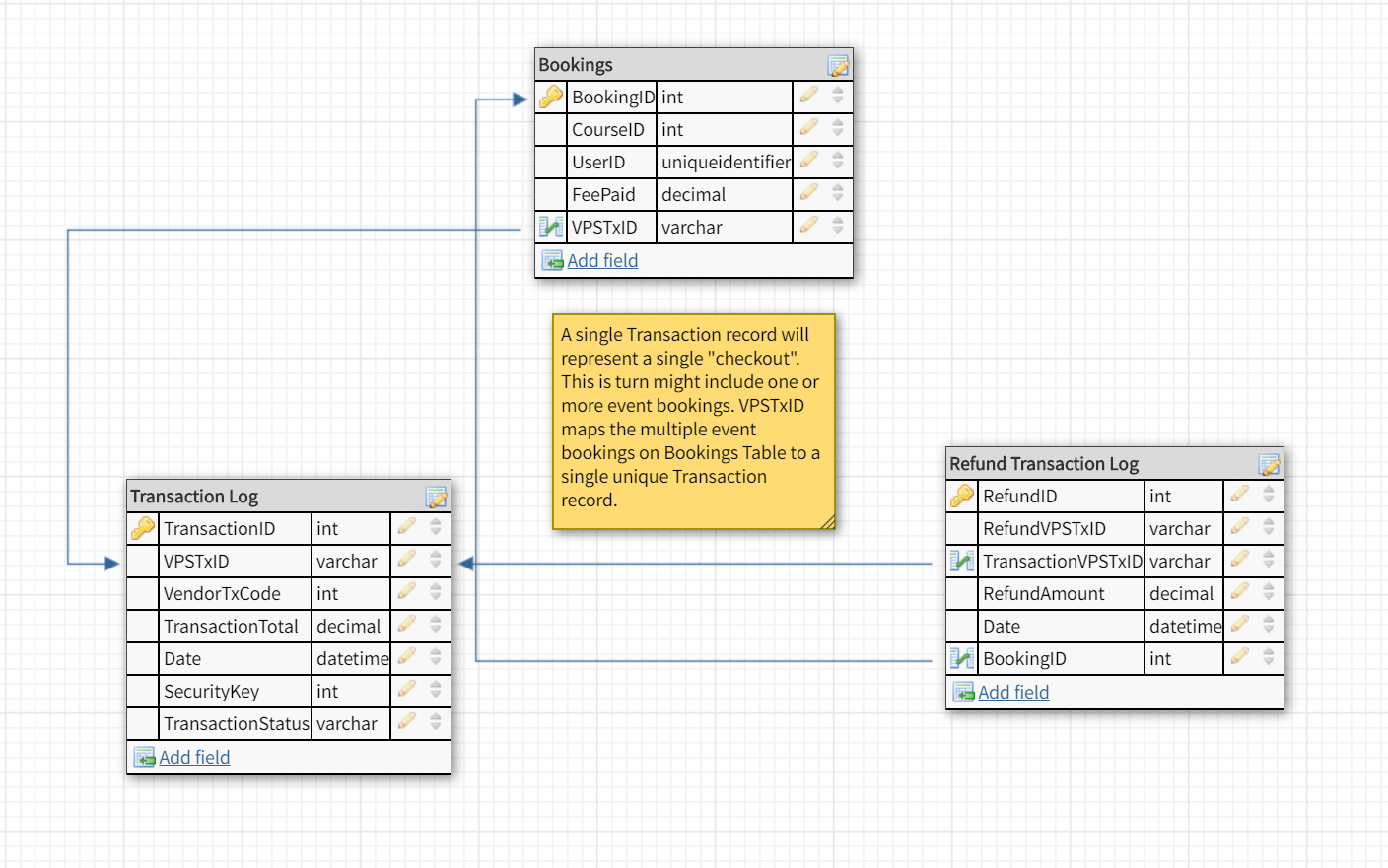


* This is a **Free** Event(ChargingType.Model identifies this)
* ChargingType.PricePoint has the value of **£0.00** in it
* Nothing else needs checked.
* The event is listed as **Free** to all users



* This is an event that has been set to determine price by **Delegate Flat Fee** (ChargingType.Model identifies this)
* ChargingType.PricePoint has the value of **£150.00** in it.
* Nothing else needs checked.
* It’s therefore an event charging **£150.00** to anyone wishing to book it.

1. Recording Payments and Refunds



Users expect, when any event carries a charge, to checkout once and pay for all items in the one transaction. The alternative is to checkout and pay for each event separately and this, understandably, is not what they would expect to do.

\*It is accepted however that any event that is free could be booked instantly upon clicking any **Book Now** button without going to a Basket and checking out.

## Payments

* One or more chargeable events will be added to the user’s basket and when payment is made, the total amount due will be taken in one transaction.
* A successful transaction will be returned with a unique ID and SagePay, our current payment gateway, refers to this as the **VPSTxID.** This Unique ID is referenced going forward in this document.
* The gateway will offer a great number of other details aside when a payment is successful, and these should all be stored in a Transaction table (the diagram above does not include them all).
* This includes the VPSTxID and the system should retro-fit that into the Bookings table, so each event that was paid for in this transaction can be referenced to the payment.
* If payment fails, the gateway will return a failed transaction record, that should also be stored in the Transaction table. It will not include a VPSTxID as this is seen as evidence of a successful transaction only.
* You will therefore have a record of all successful and unsuccessful transactions.

## The Payment Transaction Journey (Server Integration)

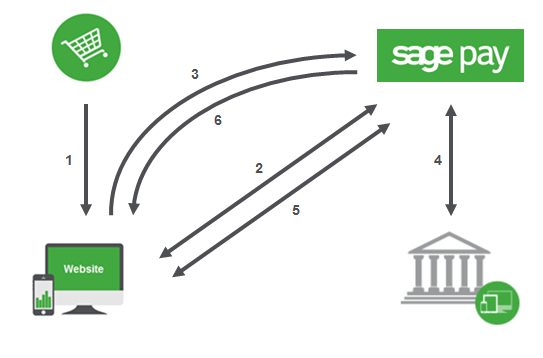
**Server Integration** is currently used. There are other methods of integrating with a Payment Gateway.

Using Server Integration means the user is sent to a Payment Gateway URL to key in card details (although using their InLine option, the user is not really that aware of this as you can embed it in your site).

The benefit of this is that no card details are either saved or transmitted from your system to the Payment Gateway. Nor are they retained.

This removes the risk of data being intercepted / storing (or loosing via a data breach) confidential credit or debit card information. It does, however, require the user to re-key card details on every visit.\*

*\*SagePay have introduced a second level of token identity. A user can create a card alias at checkout and that alias is saved on your system with a TokenID. The TokenID is saved at the Payment Gateway with the card details. It is therefore possible to check out using the alias, thus removing the need to re-key card details. We have not implemented this but see it as a good suggestion going forward.*



1. User has events in basket and clicks on the Checkout button
2. System contacts Payment Gateway with CreateToken=1 within its call. Gateway sends a URL back to system containing unique token ID embedded to be used in this transaction.
3. System directs user to this URL (payment page) and user keys in details. **Note** this payment page may look to the user like a system page but is in fact an embedded Payment Gateway URL.
4. Payment Gateway contacts the bank for authorisation. Reply will be Success of Fail
5. Payment Gateway sends system Success or Fail record for Transaction Log. If fail, will reset token ID for second attempt
6. Payment Gateway shows the user a Success of Fail message

## Refunds

Your payment gateway allows for refunds to be actioned by the user if you wish to enable it and it is suggested you implement this, on an event by event basis to the user via some Order History page.

Business rules will apply, and these can be identified by the Product Owner but ignoring that now, the process works as follows:

* The system first needs to identify how much should be refunded for this event. This will be the amount charged for the event in question and can be derived from the Bookings table
* Your payment gateway will be fed the amount derived and will also require not only the associated VPSTxID but also the **SecurityKey** stored against the initial transaction
* A successful refund can only be given to the same debit or credit card first used to pay for the event.
* If the refund is successful, you will receive a Successful Refund Record, and this should be stored in a Refund Log. It will include a new Refund VPSTxID different to the Transaction VPSTxID but both should be stored in the record created in the Refund Log.
* This allows you to reconcile refunds against initial payments.
* If the refund fails (usually due to banking rules) you will receive a Failed Refund Record with reasons and this too should be stored in the Refund Log.

1. Reporting Payments and Refunds

Your daily take must be reported to the NES Finance System. This historically has happened in the small hours of the following morning, reporting all activity between Midnight and Midnight for the “day before”.

In broad terms, the Finance system (CEDAR) needs a detailed list of Payments and Refunds for the day in question with the following totals:

1. The income total (excluding any refunds)
2. The financial movement total (including any refunds)

**eg**

|  |  |  |  |
| --- | --- | --- | --- |
| Ref | Description | Type | Total |
| 1 | Booking - Mr Smith – Course 000100: VPSTxID: 001 | credit | £120.00 |
| 2 | Booking - Mr Smith – Course 000200: VPSTxID: 001 | credit | £200.00 |
| 3 | Booking - Mrs Jones – Course 000100: VPSTxID: 002 | credit | £120.00 |
| 4 | Booking - Mrs Brown – Course 000300: VPSTxID: 003 | credit | £60.00 |
| **Total Income** | | | **£500.00** |
| 5 | Refund – Mr White – Course 000100: VPSTxID: 004 | debit | -£120.00 |
| 6 | Refund – Mrs Jones – Course 000300: VPSTxID: 005 | debit | -£60.00 |
| **Financial Movement** | | | **£320.00** |

## Notes

* Even though this is a statement of Financial Transactions, records are listed on an event by event basis, not at a transaction level.
* You will see from the example that Mr Smith bought 2 events in the same transaction; therefore, there are 2 records.
* Records will need to be extrapolated initially from the Bookings table and not the Transaction Log table.
* The Transaction Log table however is the single point of truth concerning actual payments taken so the sum(amount taken) for any booking record found and listed must equal the sum(amount taken) for any transaction records found, based on the VPSTxID that links the records. This should be checked when compiling the export.
* **If they do not, the export should not run.**
* Further, any records listed must be checked to have a status of “Booked” in the booking table and if any do not, they should be updated as part of the routine.
* Refunds are slightly simpler as events should only be cancelled / refund given on an event by event basis. There is no concept of a refund transaction having more than one event within it.
* The list of successful refund records in the Refund Log are easily extracted and from this subset of records. What should be checked is that these records exist in the booking table as “Cancelled” and if any do not, they should be updated as part of the routine.