Pay@Table

Usage and protocol v0.2

WESTPAY

Contents

1 Document Notes

1.1 History

This document series begins at v2.0. There was no v1.x document because Pay@Table was not widely used and was only offered on Westpay's Classic terminal range. From June of 2021, however, the Pay@Table protocol and module was updated for the newer Carbon terminal range. This has been labelled Pay@Table v2 and the documentation is numbered to match releases of Pay@Table.

1.1.1 Version 2.0, 06/07/2021

Initial release

2 Introduction

2.1 What is Pay@Table

Pay@Table is a module that runs on Westpay Carbon terminals. It is intended for use with the C100 series of terminals that include WiFi and a printer.

Pay@Table is intended for use in a hospitality environment. The terminal can be taken to a customer's table for them to pay their bill. The module offers:

- Waiter ID entry.
- Table listing and selection.
- Part or full payment.
- Payment by card, card or supported alternative payment methods (e.g. Swish)
- Bill splitting.
- Order summary at the point of payment for easy verification.
- Pro-forma printing.
- Table receipt printing for accounting purposes.
- Department and mode selection for restaurants with different areas and purposes, e.g. seating area vs bar vs takeaway queue.

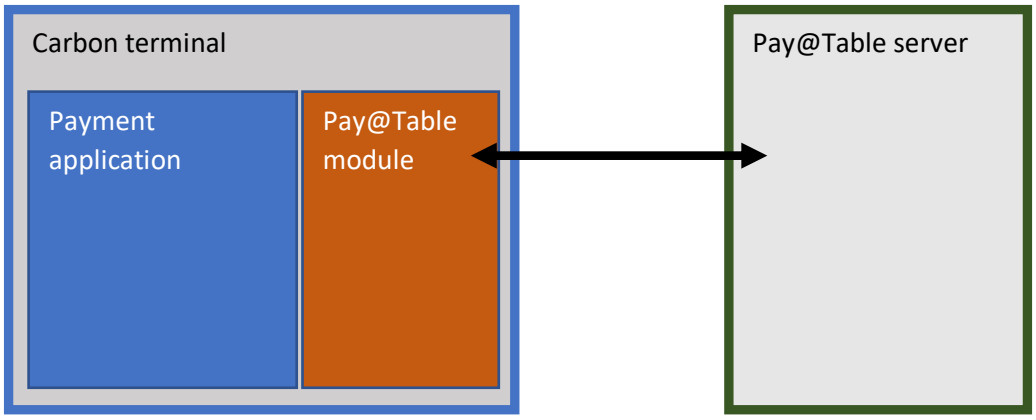
Pay@Table does not work in isolation. It requires a server that provides information on:

- Amount to pay
- Table status
- Order details

Westpay does not provide this server. It is the responsibility of integrators to develop a server for this purpose, and one purpose of this document is to make that easier to achieve.

2.2 Architecture

The Pay@Table module is not part of the Westpay payment application, but it works closely with the terminal.



The payment application has different operating modes. For Pay@Table the payment application must be in Standalone mode, which means that the transaction source comes from inside the terminal (specifically the Pay@Table module). The Pay@Table terminal will be supplied pre-configured for this, but there is a

way to switch from Standalone mode to EPAS mode (where the transaction sources come from an external EPAS client) so it is worth being aware of these different fundamental modes.

2.3 The Pay@Table server

The Pay@Table server does a lot of the work in the system. It connects the restaurant POS system to the terminal.

The Pay@Table module in the terminal gets all its information from the server, but also provides details to the server. So the module might say to the server “Fetch me all the orders for waiter 15 in the takeaway queue”. The server can then apply some filtering to only send the orders that are relevant for that terminal.

The server must track what payments have been made on a table / food order. It can also provide the terminal with information of the order itself so that the terminal can show a summary on screen and on paper.

The protocol between the terminal and the server is based around a sequence of simple XML message & response pairs. It will be discussed later in this document.

The link between the terminal and the server is a TCP/IP socket connection. Due to the nature of the Pay@Table product this is normally over WiFi.

3 Usage

This section describes how Pay@Table is normally used. There are, however, various settings that influence this, so "normal" is not a very rigid definition here. The descriptions will cover the main variations to this usage pattern.

3.1 First time start-up

On the first start-up the Pay@Table module will ask for any information that is missing. The required information is:

- The terminal ID, which is issued by Westpay
- The Pay@Table server's IP address and port number, which is managed by the client
- The authorisation (SPDH) server's IP address and port number, which is supplied by Westpay
- The configuration (PPL) server's IP address and port number, which is also supplied by Westpay

Once set, these can be changed via the Pay@Table menu.

3.2 Department / fetch mode

Before looking at the details of Pay@Table operation, it will be useful to discuss 'department' and 'fetch mode'. These are fields that Pay@Table uses. A 'department' in this context is an area of the restaurant, e.g. it could be "outside terrace" or "main restaurant", but it could also be "takeaway queue" or "members bar". The values of these fields are defined by the Pay@Table server.

The 'fetch mode' then defines what kind of orders are used in the department. Normally "table" might be used, i.e. in a restaurant the terminal works with the order relating to a table. But in the takeaway queue the fetch mode might be "ticket", and in the bar it might be "bar tab". The values of the fetch modes are also defined by the server.

Each department can define zero or more fetch modes. The intention here is that even if a terminal is currently assigned to the takeaway area, it could be temporarily switched over to handling tables if there are a lot of tables needing to be processed.

The purpose of these is to allow the server to filter search results. The value of these options does not matter at all to the Pay@Table module, but it includes them in the request for an open table to the server. This allows the server to filter the results it sends back to the terminal when it asks for a list of tables / tickets / bar tabs, etc.

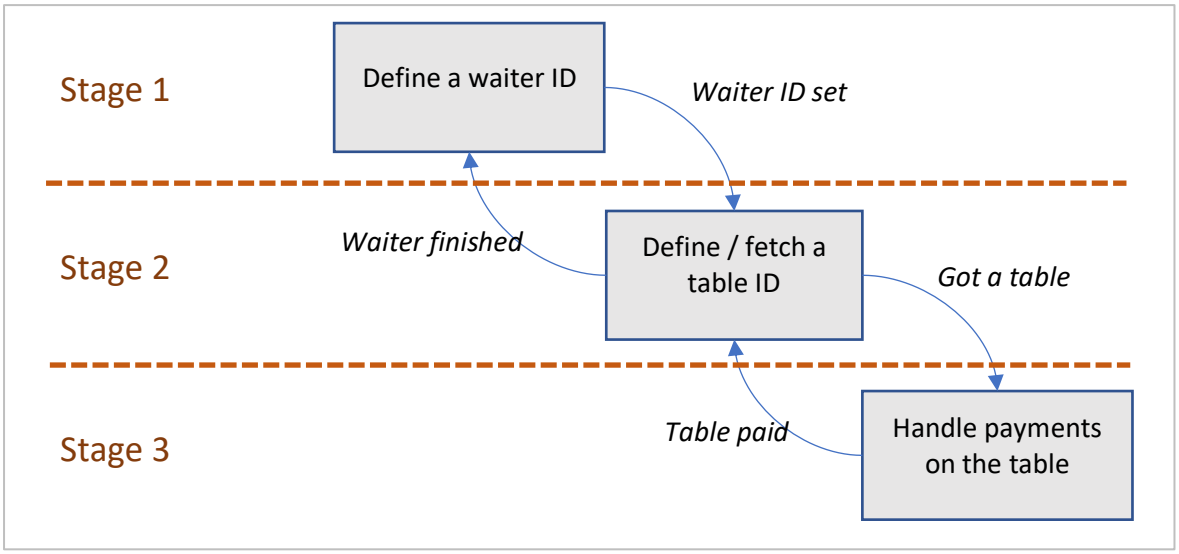
For example, if a terminal is set up to work in the takeaway area with a takeaway fetch mode then the server should probably not send a list of bar tabs or restaurant tables that are open. If, however, the terminal is in the takeaway area but asking for tables or bar tabs then the server may choose to send the full list of open tables or bar tabs because that terminal is being used in a special way.

The interpretation of the department and fetch mode values is entirely up to the server. In the Pay@Table module these are selected by the waiter, but the values that are selected are only used in requests sent to the server.

If department and fetch mode is supported in the server, and if the department and fetch mode have not yet been selected by the waiter, the waiter is prompted to choose these as part of the start-up procedure.

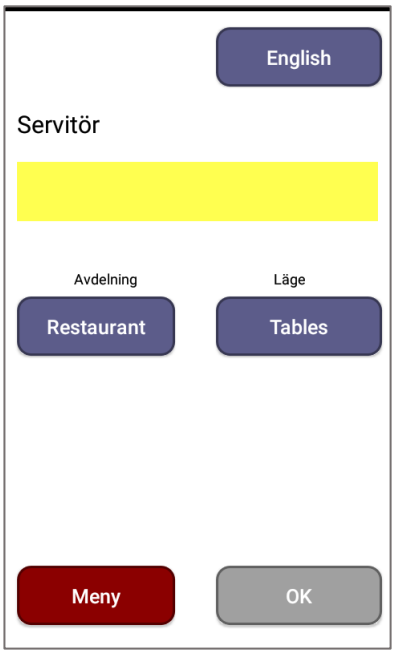
3.3 Normal usage

The normal usage pattern of Pay@Table is summarised below.



Note that we use the word “table” here, even if the department / fetch mode combination does not use “table”. This is just for simplicity and clarity.

3.3.1 Stage 1: Waiter identification



The first step of normal Pay@Table usage is waiter identification. This can be done either by entering a waiter ID on the keypad or by swiping a Micros-compatible login card.

Pressing the cancel key at stage 1 brings up the Pay@Table menu.

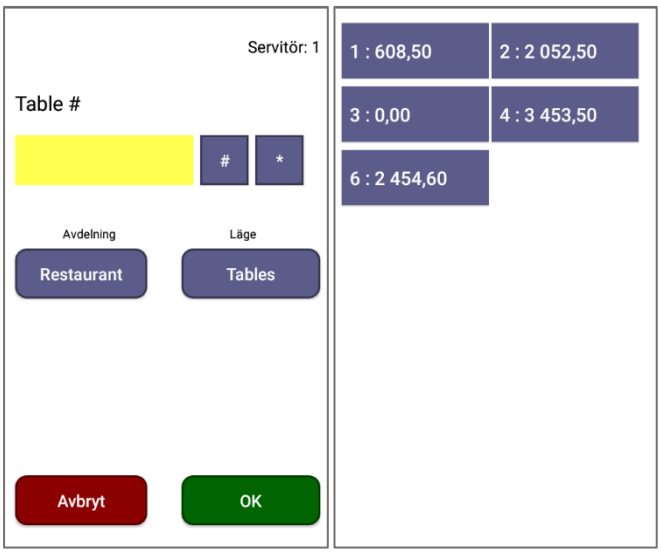
There is an option in the Pay@Table menu to pre-define the waiter ID. In this case stage 1 is skipped completely and the Pay@Table module will always go straight to stage 2. This option might be of interest if the restaurant has no interest in knowing which waiter is handling which orders, so pre-defining the waiter ID will simply speed up the process of handling payments.

The waiter ID is used by the Pay@Table module in messages to the server but it is not used in processing any data that comes back from the server.

If multiple languages are configured, the waiter can change the language from the waiter ID entry screen.

The waiter can choose to change the department and / or fetch mode in the waiter identification screen.

3.3.2 Stage 2: Table identification



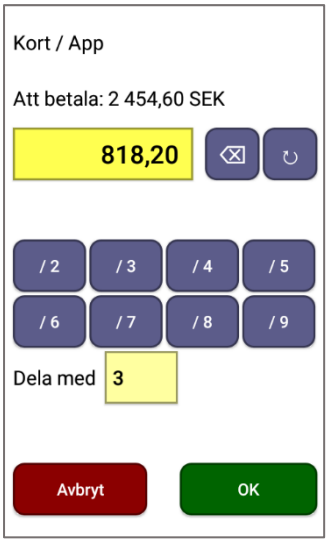
When the waiter ID is chosen, the next stage is to select an open table with a payment to be made.

In table selection the waiter can enter a table ID, or they can leave the input box blank. The terminal then requests a list of open tables from the server and, if there is more than one sent back, the waiter can select from the results.

The list of tables that are sent back by the server is for the server to decide. It is given whatever the waiter entered in the table ID input, along with the department ID and fetch mode ID, and from that the server should be able to produce a reasonable list of tables for the waiter to select from, or just one table if that is possible.

The waiter can choose to change the department and / or fetch mode in the table selection screen.

3.3.3 Stage 3: Payment



Once the table has been selected the terminal asks for information on the table, including the amount to pay and the food order details if available. These are shown on screen for the waiter to verify that they are working with the correct table data, and the waiter is prompted to select a payment method. Payment methods are either “card / app”, which is handled in the terminal, or “cash” which is handled directly with the restaurant.

“Card” as a payment method is presented as “card / app” because it can mean more than just card payments. Swish is one example that is supported.

Cash as a payment method can be enabled and disabled in the Pay@Table menu. Card / app payments are always enabled.

Once the payment method is selected the waiter is prompted for an amount to pay. The amount is pre-filled with the whole unpaid balance of the table, but the waiter can type in a new amount if needed.

The waiter can also choose to split the bill by simple division, and eight shortcut buttons are provided to allow splitting by two up to splitting by nine. The waiter can also manually enter an amount to divide the bill by, up to a maximum of 99.

If card / app payment was selected, then the payment application in the terminal is asked to run a transaction for the selected amount. If cash was selected, the terminal assumes that the waiter has correctly received the value of cash that was entered.

If less than the full amount is paid then the Pay@Table module returns to the start of stage 3, i.e. it updates the table details and prompts for the payment method again.

If bill splitting was selected, then the split amount is retained when the amount entry screen is shown. But if the waiter modifies the amount the previous split is forgotten.

If only a partial payment is needed, e.g. some guests have left the table and wish to pay their share, but others have remained, then the waiter can press cancel to return to stage 2. Otherwise stage 3 continues until the balance of the table reaches zero.

Once the balance is cleared the waiter may be offered the option to print an accounting receipt, which can be used for expenses claims, etc. This behaviour is configurable in the Pay@Table menu.

Note that cash payments in Sweden are done in whole SEK values. Minor units are left at zero. Card payments are normally editable at the sub-unit level, but this is configurable in the Pay@Table menu.

4 Additional functions

4.1 Merchant password

Some functions and menu items need a merchant password to be entered. This password is defined in your terminal's configuration and it can be seen in the Westpay Access portal.

A test password of "963741" is accepted in a development terminal.

If the merchant password is entered in the context of the Pay@Table menu, it is considered valid for as long as the menu is open. This avoids having to keep entering the password for different menu options.

4.2 Direct transactions

The Pay@Table component supports 'direct transactions', which are transactions that are initiated with no table / order information. The waiter just selects the transaction type and enters the amount. These can be useful in some situations, such as if a customer wants to buy a piece of merchandise after paying for their meal.

Direct transactions are available from the Pay@Table menu. Purchase and refund transactions must be enabled first, however, and that requires the merchant password to be entered. The option to enable direct transactions is also in the Pay@Table menu.

Direct transactions do not involve cash.

Once enabled, direct transactions will be available for one hour.

4.3 Operating modes

There are three operating modes for the Pay@Table system.

4.3.1 Pay@Table

This is the normal operating mode, and it is the focus of this document.

4.3.2 Pay@Bar

This is a streamlined version of Pay@Table, designed for rapid processing of payments. The significant differences are:

- A pre-defined waiter ID and table ID are used.
- Cash is not allowed.
- Bill splitting is not allowed.
- Partial payments are not allowed.

4.3.3 Pay@Counter

Selecting Pay@Counter switches the terminal into EPAS mode, which allows the terminal to work with an EPAS POS system. The mode switch is protected by the merchant password for safety.

In EPAS mode the terminal can be switched back to Standalone mode, which will reactivate Pay@Table.

A discussion of EPAS is beyond the scope of this document.

4.4 Reports

A series of reports are offered by the terminal, covering settings and transaction details. These are covered in the section on the menu options.

4.5 Manual updates

When payments are made on a table the details are sent to the Pay@Table server. If for some reason the terminal cannot communicate with the server it will print a manual update slip that can be used to update the POS system with the amount paid.

5 Menu structure

The Pay@Table menu is opened by pressing the red Cancel key in the waiter ID entry screen.

The menu options are explained below.

5.1 Direct Transactions

5.1.1 Purchase

See section 4.2. Requests an amount then starts a purchase transaction for that amount.

5.1.2 Refund

See section 4.2. Requests an amount then starts a refund transaction for that amount.

5.1.3 Reprint receipt

Reprints the receipt for the last direct transaction that was completed.

5.1.4 Enable direct transactions

Enables / disables direct transactions.

5.1.5 Enable ECR direct transactions

Enables / disables ECR direct transactions.

5.1.6 Enable receipt printing

Enables / disables receipt printing when a ECR direct transactions are used.

5.2 Pay@Table Settings

5.2.1 Accept cash

Controls whether cash is accepted as a payment method in Pay@Table.

5.2.2 Merchant receipt

Controls whether a merchant receipt is printed in Pay@Table.

5.2.3 Cent input

Controls whether the waiter can enter the cent value (sub-units of currency) when setting the payment amount. Note that a cash transaction in Swedish Kronor automatically disables cent input, regardless of this setting.

5.2.4 Buzzer

Controls whether the buzzer sounds for key presses and screen taps.

5.2.5 Table closing options

When a table is completely paid for the table is considered closed. At that point there are a few options for what happens.

- **Automatic, no receipt:** The table is closed automatically, and no receipt is printed. No options are offered.

- **Automatic with receipt**: The table is closed automatically, and a receipt is printed. No options are offered.
- **Timeout**: A choice of receipt / no receipt is given, as well as the option to reverse the transaction. The display closes after 60 seconds.
- **Manual**: The same as timeout but the display closes after 120 seconds.

The automatic options are quickest since they do not show a display that offers receipt and reversal choices.

5.2.6 Protocol settings

These options allow different protocol elements to be disabled. This is for the legacy support of older Pay@Table servers, and normally these should all be enabled.

- **Pro forma:** Indicates if the Pay@Table server supports the pro forma request message.
- **Release table:** Indicates if the Pay@Table server supports the release table request message.
- **Fetch tables:** Indicates if the Pay@Table server supports the fetch tables request message.
- **Loyalty:** Indicates if the Pay@Table server supports the loyalty request message.

5.2.7 Pay@Table waiter ID

This options allows the Pay@Table waiter ID to be pre-defined. This means that the normal Pay@Table process will skip stage 1, where the waiter ID is provided. This is useful if the installation site does not care which waiter is using the terminal.

5.3 Transaction Settings

5.3.1 Enable tipping

Controls whether tipping is enabled for card / app transactions. Note that tipping must also be enabled as an option in the payment application's configuration, and that is not under the control of the Pay@Table module.

5.3.2 Signature

Controls whether signature is an acceptable means of cardholder verification in a card transaction.

5.4 Terminal Configuration

5.4.1 Network config

Open the native Android network configuration settings.

5.4.2 WiFi config

Open the native Android WiFi configuration settings.

5.4.3 Payment mode

See section 4.3 for more information.

- **Pay@Table:** Selects Pay@Table mode.
- **Pay@Bar:** Selects Pay@Bar mode.
- **Pay@Counter:** Selects Pay@Counter mode and switches the terminal in to EPAS operation.

5.4.4 Server

Changes settings that are required to process card / app transactions. These settings are also required during the initial setup of the terminal. See section 3.1.

- **Terminal ID:** Defines the logical ID of the terminal.
- **Pay@Table server IP address:** Defines the IP address and port of the Pay@Table server.
- **SPDH server IP address:** Defines the IP address and port of the authorisation (SPDH) server.
- **PPL server IP address:** Defines the IP address and port of the configuration (PPL) server.

5.5 Reports

5.5.1 Reconciliation

Prints a report of transaction totals between a start and end date. The report will run from 00:00 on the start date to 23:59 on the end date, so if the start and end date are both the same then a full day's transaction totals will be printed.

5.5.2 Transactions

Prints a summary of all the transactions between a start date and time and an end date and time.

5.5.3 Manual update

Prints a report of manual update entries to update the POS system. See section 4.5.

5.5.4 Terminal information

Prints a report that includes some terminal details such as the serial number, server details and configuration files in use.

5.5.5 Terminal settings

Prints a report that summarises some of the settings available through the menus.

5.6 Terminal Management

5.6.1 Send logs

Uploads the terminal log files to a FTP server. The FTP server address and port is specified after selecting this menu option.

5.6.2 Terminal system menu

Opens the system menu of the payment application on the terminal. This is not related to Pay@Table functionality. It is possible to switch from EPAS mode back to Standalone in this menu.

5.6.3 Send store & forward queue

The store & forward queue is where transactions are stored that must be uploaded to the authorisation server. If there has been a communications problem or a server problem then this queue can hold several transactions waiting for upload. This menu option attempts to upload the queue to the authorisation server.

5.6.4 Delete store & forward queue

Deletes entries from the store & forward queue. **Note:** deleting transactions from the queue can cause payments and / or refunds to be unprocessed, and this should only be done if absolutely necessary.

6 Pay@Table Protocol

The protocol between the terminal and the Pay@Table server is a simple one that uses XML messages. The key points of the protocol are:

- All messages originate from the terminal.
- Every message from the terminal is a request, and each one expects a response from the server.
- A new connection to the server is opened for each request / response pair.
- The default port number for the server is 45017. This is configurable in the Pay@Table module if necessary.

6.1 Message header

Every message that is sent has a four-byte header to indicate the length of the following data. This header is encoded MSB first, so a XML message that is 700 bytes long, which is 0x2BC, would have a header encoded as:

```
0x00 0x00 0x02 0xBC
```

followed by the XML message. The whole packet would, therefore, be 704 bytes long.

The server should allow a zero-length message, i.e. a packet that is only four zero bytes and nothing else. This is valid in protocol terms since it indicates a packet with zero payload. This could be used by the terminal to verify that the connection to the ECR works, but without sending any meaningful or problematic data.

6.2 Message structure

Every message between the terminal and server has the same XML structure in the outer layers. Each message is encapsulated in a “rtmp” element, and then within either a “request” or “response” element, such as this example:

```
<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <request protocol="2.0" sequenceNumber="19" serialNumber="1043000833" type="OpenTable"
terminalId="80000091" tableId="2" waiterId="1" />
</rtmp>
```

6.2.1 Request / response message types

The request / response type is defined in the **type** attribute of the request / response element. It takes one of the following values:

Type	Purpose
OpenTable	Requests details of an open table.
UpdateTable	Registers a payment on a table.
CloseTable	Indicates that a table can be closed.
FetchDepartments	Requests a list of departments and fetch modes.
FetchTables	Requests a list of open tables.
TableReceipt	Requests the data needed for a table receipt.
ReverseTable	Indicates that a payment on a table should be reversed.

ReleaseTable	Indicates that an open table can be released.
ProFormaReceipt	Requests the data required for a pro-forma receipt.
Loyalty	Indicates that a loyalty card has been accepted in the terminal.
TerminalRegistration	Indicates that the terminal is about to switch over to EPAS mode (Pay@Counter).

The request and response elements have a minimum set of attributes that are present in every instance. They are:

Attribute	Usage	Purpose
protocol	Request and response	Indicates the protocol level. The value at time of writing is fixed at "2.0".
sequenceNumber	Request and response	Indicates a sequence number in the request, and the response must use the same value. This allows the sender to match the response to the request.
serialNumber	Request and response	Indicates the serial number of the terminal that sent the request, and the response must use the same value. This allows the sender to verify that the response it received was intended for that terminal.
type	Request and response	The message type. The response much use the same value, whether it recognises the value or not.
terminalId	Request only	Indicates the terminal ID to the server.
responseCode	Response only	Indicates the outcome of the response.

The request and response elements also use a set of optional attributes that are dependent on context. They are:

Attribute	Usage	Purpose
tableId	Request only	Indicates the table ID that has been selected on the terminal.
waiterId	Request only	Indicates the waiter ID that has been selected on the terminal.
departmentId	Request only	Indicates the department ID that has been selected on the terminal.
modeId	Request only	Indicates the mode ID that has been selected on the terminal.
amount	Request only	Provides the transaction amount to the server.
extra	Request only	Provides the tip amount to the server.
paymentType	Request only	Indicates the type of payment method used, either "cash" or "card".
referenceNumber	Request only	Indicates the transaction reference number, which is used if any later follow-up is needed.
financialInstitution	Request only	Provides a three-letter code for the bank / payment provider behind a payment. Not used with cash payments.

loyaltyCardNumber	Request only	Provides the card number of a loyalty card used during a card transaction.
ipAddress	Request only	Indicates the IP address and port number of the terminal's EPAS protocol server.
epasModeOn	Request only	Indicates that EPAS mode is enabled when set to "on".
tableText	Response only	Provides an optional label that identifies the table. Used only in listing FetchTables responses.
controlBox	Response only	Indicates the control box number on a table receipt.
footer	Response only	Holds text for a receipt footer.
header	Response only	Holds text for a receipt header.
receiptNumber	Response only	Indicates the receipt number.

6.2.2 Response elements: table

Some server responses can contain one or more **table** elements.

An example of a table element is:

```
<table tableId="6" tableText="Table 6" totalAmount="254100">
  <payments>
    <payment type="cash" amount="45000" />
    <payment type="cash" amount="60000" />
  </payments>
  <order>
    <food name="Yellowtail Sashimi" price="15500" count="1" />
    <food name="Grillat rökt sidfläsk" price="8500" count="2" />
    <food name="Entrecôte" price="46500" count="1" />
    <food name="Oxfilé" price="44500" count="1" />
    <food name="Lammasado" price="32000" count="1" />
    <food name="Achaval-Ferrer malbec" price="89000" count="1" />
    <food name="Kaffe" price="3200" count="3" />
  </order>
</table>
```

A table element has two attributes:

- **tableId**: The ID of this table
- **totalAmount**: The total order amount for this table. This is *not* the amount remaining to pay, it is the total whether paid or unpaid.

> **Note:** All amounts in Pay@Table are expressed in subunits of the currency that is configured in the terminal. So, for example, an amount of €145.00, £145.00 or 145:- is represented as 14500.

The table element can contain a **payments** element, which contains zero or more **payment** elements. These record payments that have been made on the terminal. Each payment element has two attributes:

- **type:** the payment method used, either "cash" or "card".
- **amount:** the amount of the payment in sub-units.

The table element can also contain an **order** element, which contains zero or more **food** nodes. These tell the terminal which individual items have been ordered for the table. Each order element has three attributes:

- **name:** the name or label of the item.
- **price:** the price in sub-units of the item.
- **count:** the number of these items in the order.

6.2.3 Response elements: departments

When responding to a FetchDepartments request the server can provide a **departments** element, which contains zero or more **department** elements.

And example of this is:

```
<departments>
  <department id="d1" label="Restaurant">
    <modes>
      <mode id="table" label="Tables" prompt="Table #" />
      <mode id="bar" label="Bar tabs" prompt="Bar tab" />
      <mode id="ticket" label="Tickets" prompt="Ticket #" />
    </modes>
  </department>
  <department id="bar" label="Bar area">
    <modes>
      <mode id="bar" label="Bar tabs" prompt="Bar tab" />
      <mode id="table" label="Tables" prompt="Table #" />
      <mode id="ticket" label="Tickets" prompt="Ticket #" />
    </modes>
  </department>
  <department id="takeaway" label="Takeaway">
    <modes>
      <mode id="ticket" label="Tickets" prompt="Ticket #" />
      <mode id="table" label="Tables" prompt="Table #" />
      <mode id="bar" label="Bar tabs" prompt="Bar tab" />
    </modes>
  </department>
</departments>
```

Each department element has three attributes:

- **id:** the ID of the department, which will be included in FetchTables requests
- **label:** the name of the department that will be used on-screen in the Pay@Table module

Each department element contains a **modes** element, which contains one or more **mode** elements. Each mode element has three attributes:

- **id:** the ID of the fetch mode, which will be included in FetchTables requests
- **label:** the name of the fetch mode that will be used on-screen in the Pay@Table module.
- **prompt:** the prompt that will be used on-screen in the Pay@Table module to ask for a search term, e.g. a table number, or a ticket number, etc.

Using departments and fetch modes

Each department element provides information on a department / section / centre. These could be physically separate areas within a restaurant, or they may be functionally different. The purpose of these departments is to allow the server to filter the list of tables that are offered to the terminal in a FetchTables response.

When the waiter selects a department and a fetch mode, the IDs are sent to the server in a FetchTables request, in the **departmentId** and **modeId** attributes respectively. For example, here the waiter has selected the 'Restaurant' department and the 'Tables' fetch mode:

```
<request protocol="2.0" sequenceNumber="9" serialNumber="1043000833" type="FetchTables"
terminalId="80000091" tableId="1" waiterId="1" departmentId="d1" modeId="table" />
```

The department list example above re-uses the mode ID values (“table”, “ticket”, and “bar”) in each department, but the server could assign unique mode IDs to simplify understanding the FetchTables request, e.g. a mode ID of “restaurant_tables” would combine the department and mode into a single expression.

There are no rules to control how the server interprets these department and mode combinations. Using the example above, a combination of the “bar” department and the “bar” mode would tell the server to only fetch bar tabs, while the “bar” department and the “ticket” mode might say that the terminal in the bar area is being used to handle takeaway tickets so maybe the server will return takeaway tickets first but also, after the tickets, a list of open bar tabs for flexibility. It is up to the server developers to work out the best use of this according to the needs of the working environment.

6.2.4 Response elements: vat

When providing a table receipt the server can indicate the value added tax (VAT) components of the bill in one or more **vat** elements. These are only used for adding to the receipt output.

An example of this is:

```
<vat percent="25" amount="36750" />
<vat percent="12" amount="11064" />
```

Each vat element has two attributes:

- **percent:** The VAT percentage band.
- **amount:** The total VAT amount, in sub-units, for this band.

6.2.5 Error handling and response codes

Pay@Table uses a very small set of response code values. This is simply because the context of an error is very specific to the server, and the server is therefore best placed to describe what the error is.

Response codes are interpreted as follows:

Code	Meaning	Purpose
0	Success.	The server should return this when the request was successfully handled.
80	Generic error	Indicates that the server was unable to handle the request.
81	Waiter ID invalid	Indicates that the server does not recognise the waiter ID or that the waiter ID is invalid.

When the server has a problem processing a request it can return an informative message in the **errorString** attribute, as in this example:

```
<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <response protocol="2.0" sequenceNumber="31" serialNumber="1043000833" type="UpdateTable"
responseCode="80" errorString="POS system failure">
</rtmp>
```

The Pay@Table module will report an error and will display the contents of the **errorString** attribute so that the waiter can take appropriate action.

If the server returns error code 81, indicating an invalid waiter ID, then an appropriate error message is displayed. But also the Pay@Table module will clear the entered waiter ID, forcing the waiter to provide their ID again.

6.3 Message types

6.3.1 Login

The Login message requests the name of a waiter. This allows the terminal to show the name of a waiter rather than just an id.

If a waiter element is provided in the response, and the waiter ID in the response matches the ID entered on the terminal, then the terminal will use the waiter name. Otherwise it will continue to display the waiter ID as it is currently. The waiter name can be set to an empty string if there is no name to be shown. This will prevent the ID being displayed, but will also avoid the need to have a name available.

If the waiter ID is invalid the server can respond with the appropriate error response to indicate this.

The waiter name will persist until a new waiter ID is required. The terminal will not store a set of ID / name pairs because this can change beyond the terminal's knowledge.

If no response is received to this message, or the response indicates that the message is not supported, the P@T module will not ask again until the terminal is restarted. This is to minimise any impact on existing installations.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <request protocol="2.0" sequenceNumber="4" serialNumber="A103000554" type="Login"
terminalId="80000091" tableId="1" waiterId="1" />
</rtmp>


<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <response protocol="2.0" sequenceNumber="4" serialNumber="A103000554" type="Login"
responseCode="0">
    <waiter id="1" name="John Smith" />
  </response>
</rtmp>
```

6.3.2 OpenTable

The OpenTable message requests details of an open table. An open table is one where there is a payment to be made.

The OpenTable response should contain a single **table** element, assuming the requested table ID is valid.

Note that the department and mode IDs are included in the request if they have been set. These can help the server to interpret the value entered as the table ID.

Example:

```
<rtmp>
  <request protocol="2.0" sequenceNumber="56" serialNumber="1043000833" type="OpenTable"
terminalId="80000091" tableId="6" waiterId="1" departmentId="d1" modeId="table" />
</rtmp>
```

```
<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <response protocol="2.0" sequenceNumber="56" serialNumber="1043000833" type="OpenTable"
responseCode="0">
    <table tableId="6" totalAmount="254100">
      <payments>
        <payment type="cash" amount="45000" />
        <payment type="cash" amount="60000" />
      </payments>
      <order>
        <food name="Yellowtail Sashimi" price="15500" count="1" />
        <food name="Grillat rökt sidfläsk" price="8500" count="2" />
        <food name="Entrecôte" price="46500" count="1" />
        <food name="Oxfilé" price="44500" count="1" />
        <food name="Lammasado" price="32000" count="1" />
        <food name="Achaval-Ferrer malbec" price="89000" count="1" />
        <food name="Kaffe" price="3200" count="3" />
      </order>
    </table>
  </response>
</rtmp>
```

6.3.3 UpdateTable

The UpdateTable request informs the server that a payment has been taken for a table. In addition to the standard attributes, it Includes:

- The amount in sub-units,
- The value of any tip that was given,
- The payment method,
- The transaction reference,
- The financial institution, if the payment type is not cash.

The response, if successful, should contain an updated **table** element for the selected table.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <request protocol="2.0" sequenceNumber="4" serialNumber="1043000833" type="UpdateTable"
terminalId="80000091" tableId="6" waiterId="1" amount="50000" extra="0" paymentType="card"
referenceNumber="800000910017" financialInstitution="SHB" />
</rtmp>

<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <response protocol="2.0" sequenceNumber="4" serialNumber="1043000833" type="UpdateTable"
responseCode="0">
    <table tableId="6" totalAmount="254100" departmentId="1">
      <payments>
        <payment type="card" amount="50000" />
      </payments>
      <order>
        <food name="Yellowtail Sashimi" price="15500" count="1" />
        <food name="Grillat rökt sidfläsk" price="8500" count="2" />
        <food name="Entrecôte" price="46500" count="1" />
        <food name="Oxfilé" price="44500" count="1" />
        <food name="Lammasado" price="32000" count="1" />
        <food name="Achaval-Ferrer malbec" price="89000" count="1" />
        <food name="Kaffe" price="3200" count="3" />
      </order>
    </table>
```

```
  </response>
</rtmp>
```

6.3.4 CloseTable

The CloseTable requests informs the server that a table has been completely paid for. The response, if successful, should contain an updated **table** element for the selected table.

Note that the server could ignore CloseTable requests, since it already knows when a table is paid for from the UpdateTable requests. So the server could implement an auto-closing methodology instead of waiting for the CloseTable request.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <request protocol="2.0" sequenceNumber="7" serialNumber="1043000833" type="CloseTable"
terminalId="80000091" tableId="6" />
</rtmp>

<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <response protocol="2.0" sequenceNumber="7" serialNumber="1043000833" type="CloseTable"
responseCode="0">
    <table tableId="6" totalAmount="254100" departmentId="1">
      <payments>
        <payment type="card" amount="50000" />
        <payment type="cash" amount="204100" />
      </payments>
      <order>
        <food name="Yellowtail Sashimi" price="15500" count="1" />
        <food name="Grillat rökt sidfläsk" price="8500" count="2" />
        <food name="Entrecôte" price="46500" count="1" />
        <food name="Oxfilé" price="44500" count="1" />
        <food name="Lammasado" price="32000" count="1" />
        <food name="Achaval-Ferrer malbec" price="89000" count="1" />
        <food name="Kaffe" price="3200" count="3" />
      </order>
    </table>
  </response>
</rtmp>
```

6.3.5 FetchDepartments

The FetchDepartments request asks the server to provide a list of departments and fetch modes. This is explained in some detail in section 6.2.3

Note that the department list will sometimes be fetched before a waiter ID has been entered. The server must, therefore, be able to work with an empty **waiterId** attribute.

Example:

```
<rtmp>
  <request protocol="2.0" sequenceNumber="3" serialNumber="1043000833"
type="FetchDepartments" terminalId="80000091" tableId="6" waiterId="1" />
</rtmp>

<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <response protocol="2.0" sequenceNumber="3" serialNumber="1043000833"
type="FetchDepartments" responseCode="0">
    <departments>
      <department id="d1" label="Restaurant">
```

```
        <modes>
          <mode id="table" label="Tables" prompt="Table #" />
          <mode id="bar" label="Bar tabs" prompt="Bar tab" />
          <mode id="ticket" label="Tickets" prompt="Ticket #" />
        </modes>
      </department>
      <department id="bar" label="Bar area">
        <modes>
          <mode id="bar" label="Bar tabs" prompt="Bar tab" />
          <mode id="table" label="Tables" prompt="Table #" />
          <mode id="ticket" label="Tickets" prompt="Ticket #" />
        </modes>
      </department>
      <department id="takeaway" label="Takeaway">
        <modes>
          <mode id="ticket" label="Tickets" prompt="Ticket #" />
          <mode id="table" label="Tables" prompt="Table #" />
          <mode id="bar" label="Bar tabs" prompt="Bar tab" />
        </modes>
      </department>
    </departments>
  </response>
</rtmp>
```

6.3.6 FetchTables

The FetchTables request asks the server to provide a list of open tables. The request may include a table ID that has been entered by the waiter on the terminal. This table ID can be treated as a precise match, in which case no more than one table would be returned, or as a search term that would cause multiple tables to be returned. The exact interpretation of it is up to the server.

If no table ID is provided then the server should return all the open tables that are appropriate, given the waiter, department and fetch mode IDs.

The FetchTables response should contain a list of table elements. If the FetchTables responses contain **tableText** attributes then that text will be used in the list of tables that are presented to the waiter. If not, the table ID is used.

- If more than one table element is provided then the waiter will be prompted to select a table from the list.
- If only one table element is provided then that is selected automatically.

Example:

```
<rtmp>
  <request protocol="2.0" sequenceNumber="41" serialNumber="1043000833" type="FetchTables"
terminalId="80000091" waiterId="1" departmentId="d1" modeId="table" />
</rtmp>

TX 17/06/2021 17:25:08
<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <response protocol="2.0" sequenceNumber="41" serialNumber="1043000833" type="FetchTables"
responseCode="0">
    <table tableId="1" totalAmount="53850">
      <payments />
      <order>
        <food name="Lasagna" price="12500" count="2.5" />
        <food name="Vitlöksbröd" price="4500" count="2" />
        <food name="Kronenbourg" price="6800" count="2" />
      </order>
```

```
    </table>
    <table tableId="3" totalAmount="20000">
      <payments>
        <payment type="cash" amount="20000" />
      </payments>
      <order>
        <food name="Burger" price="20000" count="1" />
      </order>
    </table>
    <table tableId="6" totalAmount="254100">
      <payments />
      <order>
        <food name="Yellowtail Sashimi" price="15500" count="1" />
        <food name="Grillat rökt sidfläsk" price="8500" count="2" />
        <food name="Entrecôte" price="46500" count="1" />
        <food name="Oxfilé" price="44500" count="1" />
        <food name="Lammasado" price="32000" count="1" />
        <food name="Achaval-Ferrer malbec" price="89000" count="1" />
        <food name="Kaffe" price="3200" count="3" />
      </order>
    </table>
  </response>
</rtmp>
```

6.3.7 TableReceipt

The TableReceipt request asks the server to provide details for an accountable receipt. This can be requested by the waiter when the table has been closed and there is nothing more to pay.

The TableReceipt response element supports additional attributes that are included in the printed receipt:

- controlBox
- receiptNumber
- header
- footer

The TableReceipt response contains a **table** element along with **vat** elements that provide the VAT component of the payments.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <request protocol="2.0" sequenceNumber="6" serialNumber="1043000833" type="TableReceipt"
terminalId="80000091" tableId="6" />
</rtmp>

TX 17/06/2021 15:45:06
<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <response protocol="2.0" sequenceNumber="6" serialNumber="1043000833" type="TableReceipt"
responseCode="0" controlBox="1" header="Test restaurant" footer="Thank you for visiting"
receiptNumber="1">
    <table tableId="6" totalAmount="254100" departmentId="1">
      <payments>
        <payment type="card" amount="50000" />
        <payment type="cash" amount="204100" />
      </payments>
      <order>
        <food name="Yellowtail Sashimi" price="15500" count="1" />
        <food name="Grillat rökt sidfläsk" price="8500" count="2" />
        <food name="Entrecôte" price="46500" count="1" />
```

```
        <food name="Oxfilé" price="44500" count="1" />
        <food name="Lammasado" price="32000" count="1" />
        <food name="Achaval-Ferrer malbec" price="89000" count="1" />
        <food name="Kaffe" price="3200" count="3" />
      </order>
    </table>
    <vat percent="25" amount="36750" />
    <vat percent="12" amount="11064" />
  </response>
</rtmp>
```

6.3.8 ReverseTable

The ReverseTable request informs the server that a reversal has been made, which is where the most recent payment is cancelled by the waiter. The server should undo the payment and update the table totals appropriately. The ReverseTable attributes are used in the same way as an UpdateTable request, and the only difference is the nature of the change to the table amounts.

The reference number in the ReverseTable request should match the reference number in the previous UpdateTable request.

The ReverseTable response follows the pattern of the UpdateTable response, i.e. it includes an updated table element.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <request protocol="2.0" sequenceNumber="18" serialNumber="1043000833" type="ReverseTable"
terminalId="80000091" tableId="6" waiterId="1" amount="65000" extra="0" paymentType="cash"
referenceNumber="009118132517" />
</rtmp>

<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <response protocol="2.0" sequenceNumber="18" serialNumber="1043000833" type="ReverseTable"
responseCode="0">
    <table tableId="6" totalAmount="254100" departmentId="1">
      <payments />
      <order>
        <food name="Yellowtail Sashimi" price="15500" count="1" />
        <food name="Grillat rökt sidfläsk" price="8500" count="2" />
        <food name="Entrecôte" price="46500" count="1" />
        <food name="Oxfilé" price="44500" count="1" />
        <food name="Lammasado" price="32000" count="1" />
        <food name="Achaval-Ferrer malbec" price="89000" count="1" />
        <food name="Kaffe" price="3200" count="3" />
      </order>
    </table>
  </response>
</rtmp>
```

6.3.9 ReleaseTable

The ReleaseTable request informs the server that the waiter is finished with a table, even though the table may still have payments to be made.

The purpose of this message is to allow the server to implement a table locking mechanism if it wants to ensure that only one waiter can modify a table at a time. If this is required then in the period between an OpenTable and a corresponding ReleaseTable or CloseTable the table would be considered locked to the

waiter ID being used in these requests, and an OpenTable performed by a different waiter ID would get an error response with an appropriate error message.

If no locking mechanism is needed then the server can simply return a success response but otherwise ignore this message.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <request protocol="2.0" sequenceNumber="12" serialNumber="1043000833" type="ReleaseTable"
terminalId="80000091" tableId="1" waiterId="1" />
</rtmp>

<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <response protocol="2.0" sequenceNumber="12" serialNumber="1043000833" type="ReleaseTable"
responseCode="0" />
</rtmp>
```

6.3.10 ProFormaReceipt

The ProFormaReceipt request asks the server for the data required to be able to print a pro-forma receipt. The waiter can request a pro-forma receipt whenever a payment is being made.

The ProFormaReceipt response should contain a single **table** element.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <request protocol="2.0" sequenceNumber="5" serialNumber="1043000833"
type="ProFormaReceipt" terminalId="80000091" tableId="6" />
</rtmp>

<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <response protocol="2.0" sequenceNumber="5" serialNumber="1043000833"
type="ProFormaReceipt" responseCode="0">
    <table tableId="6" totalAmount="254100">
      <payments>
        <payment type="cash" amount="42400" />
      </payments>
      <order>
        <food name="Yellowtail Sashimi" price="15500" count="1" />
        <food name="Grillat rökt sidfläsk" price="8500" count="2" />
        <food name="Entrecôte" price="46500" count="1" />
        <food name="Oxfilé" price="44500" count="1" />
        <food name="Lammasado" price="32000" count="1" />
        <food name="Achaval-Ferrer malbec" price="89000" count="1" />
        <food name="Kaffe" price="3200" count="3" />
      </order>
    </table>
  </response>
</rtmp>
```

6.3.11 Loyalty

The Loyalty request informs the server that a loyalty card has been presented as part of a card payment. If the card is accepted and processed by the server then a response code of 0 will indicate success back to the payment application. A non-zero response code will indicate that the card was not accepted.

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <request protocol="2.0" sequenceNumber="4" serialNumber="1043000833" type="Loyalty"
terminalId="80000091" tableId="1" loyaltyCardNumber="6002019220470012" amount="60800" />
</rtmp>

<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <response protocol="2.0" sequenceNumber="4" serialNumber="1043000833" type="Loyalty"
responseCode="0" />
</rtmp>
```

6.3.12 TerminalRegistration

If the terminal is switched into EPAS mode (see section 4.3.3 and section 5.4.3) then the last message to the server is a TerminalRegistration request. It contains information on the EPAS connection that is needed to run transactions, specifically in the **ipAddress** attribute which contains the terminal's IP address and the EPAS service port number.

Note that **epasModeOn** is only used in this context, and therefore it will always have the value "on".

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <request protocol="2.0" sequenceNumber="24" serialNumber="1043000833"
type="TerminalRegistration" terminalId="80000091" ipAddress="192.168.0.250:3000"
epasModeOn="true" />
</rtmp>

<?xml version="1.0" encoding="utf-8"?>
<rtmp>
  <response protocol="2.0" sequenceNumber="24" serialNumber="1043000833"
type="TerminalRegistration" responseCode="0" />
</rtmp>
```

7 Direct Transactions in Pay@Table

Direct Transactions in Pay@Table allow the ECR to perform transactions directly and thus bypass the normal Pay@Table process of selecting a department and a table that the transaction is related to.

7.1 Separate communication channel

In Pay@Table network traffic always begins in the terminal with a request and the server provides a response, that is the client is on the terminal and the server on the ECR.

Regarding direct transaction it is the opposite, the server is on the terminal and the client on the ECR. The direct transaction adds a new communications channel, where the Pay@Table module listens for incoming messages on a socket connection. This will run alongside the existing communications functionality.

7.1.1 Security

Direct transactions will only be accepted from the same IP address that is configured for Pay@Table, and the direct transaction requests must include the terminal's ID so that the terminal knows the message has been sent to the correct device.

7.1.2 Messaging

For simplicity the direct transaction messaging will involve a single request sent to the terminal and a single response. The request will include the details of the transaction to be performed, and the response will include the transaction response and receipt details (where appropriate).

The messaging will include status details to handle errors.

7.2 Network connection

The Pay@Table component will operate a socket server. It will listen on the same port number that is configured for use with Pay@Table. So if, for example, the Pay@Table component is configured to connect to the server on port 45017 (which is the default) then the Pay@Table will listen on port 45017 for direct transaction messages.

Note that the direct transactions server will use the port number that is configured at the time when it is started. If the port number is changed on the terminal then that will not impact the direct transactions server until the next time the server is started.

Each message, in either direction, will begin with a four-byte header that indicates the length, MSB first, of the message contents that follow. This is the same mechanism that is used for Pay@Table messaging.

The Pay@Table component will only accept connections from IP address that is configured for the Pay@Table server. This is to ensure that the terminal 'belongs' to a single server.

7.3 Direct transaction availability

The Pay@Table component will only allow direct transactions while it is idle. This means the Pay@Table component must be at the stage of getting the waiter ID or the table ID. If the waiter has selected a table then the Pay@Table component will not allow any direct transactions until the table has been finished processing (typically this means when a CloseTable or ReleaseTable message has been completed with the Pay@Table server).

Note that the direct payments will not be available if the Pay@Table module is configured for Pay@Bar mode. In this case the login request's *direct* attribute will have the 'unavailable' value.

7.4 Receipts

The Pay@Table module will not print receipts when it is handling a direct transaction from the server. In common with other ECR-driven protocols, the receipt printing will be handled by the ECR that initiated the transaction.

The ECR will get the data elements needed to produce a receipt, rather than a pre-formatted version.

7.5 Status codes

The message wrapper, covered below, has a *status* attribute in responses. The values of the status attribute are listed below. Where the result is not 0 the *errorString* attribute will contain further details of the failure.

0	Successful transaction. "Success" here refers to processing the message and starting the transaction. It does not refer to whether or not a transaction is approved.
0x100	Unspecified error. Only used if one of the other error codes does not apply. The *errorString* attribute will give more details.
0x101	Terminal is busy. This can mean a transaction is already in progress, or the Pay@Table application is preparing to run a transaction.
0x102	Transaction not valid. This could be an invalid transaction type, or some other problem with the requested transaction.
0x103	Message not valid. The XML message in *patdirect* could not be de-serialised from XML.
0x104	No transaction available. When an abort is requested, but no transaction is pending.
0x105	No transaction match. When an abort is requested, but the pending transaction is not the same as the one the abort request concern.

7.6 Message wrapper

Direct transaction messages will have a common outer XML block. This block will be a 'patdirect' element that indicates the protocol version, message ID, message type, and terminal ID.

There will be three types of wrapper message, request, response and abort.

Request message:

```
<?xml version="1.0" encoding="utf-8"?>
<patdirect protocol="1.0" type="request" id="msg1" terminalId="80000091">
      ...  transaction details go in here
</patdirect>
```

Response message:

```
<?xml version="1.0" encoding="utf-8"?>
<patdirect protocol="1.0" type="response" id="msg1" terminalId="80000091"
status="0" errorString="" >
      ...  transaction result and receipt details go in here
</patdirect>
```

Abort message:

```
<?xml version="1.0" encoding="utf-8"?>
<patdirect protocol="1.0" type="abort" id="msg1" terminalId="80000091" />
```

The response will echo the message ID (from the 'id' attribute) so that the sender can match the response to the request.

The abort message is sent to the terminal to request that an ongoing transaction should be aborted. The client should then wait for a response message containing the transaction result and receipt.

> **Note:** there is no guarantee that the transaction will be aborted – if the transaction was already complete when the abort message was sent then there is nothing to abort, and the transaction result will not be affected by the abort request.

7.7 Specifying a transaction

The Pay@Table module has an interface that it uses when working with the payment application in the terminal. For simplicity, the proposal here is to use the same classes that are used in that interface.

The way to do this is:

1. Use the POSInterface.cs source file, which defines the classes used to drive transactions and handle receipts.
2. Create an instance of the TransactionData class, and populate it according to the requirements of the transaction. Create instances of any contained classes that are needed.
3. Use the XmlSerializer class to create an XML serialised representation of the TransactionData class.
4. Provide that XML within the *patdirect* element.

Here is an example of creating a TransactionData instance:

```
var transaction = new TransactionData();
transaction.Amounts = new TransactionAmounts(608.50, 0, 0, 0);
transaction.Type = TransactionType.Purchase;
transaction.Options = new TransactionOptions();
transaction.Options.DisabledFeatures.Add(TransactionFeatures.PinBypass);
transaction.Options.DisabledFeatures.Add(TransactionFeatures.Tipping);
```

When the instance of TransactionData is done, it can then be processed like this:

```
XmlDocument xd = new XmlDocument();
XmlSerializer s = new XmlSerializer(typeof(TransactionData));
using (MemoryStream ms = new MemoryStream())
{
    s.Serialize(ms, transaction);
    ms.Flush();
    ms.Seek(0, SeekOrigin.Begin);
    xd.Load(ms);
}
// The first node, index 0, is the XML doctype header.
// The serialised transaction data is the second node.
XmlNode xmlTransaction = xd.ChildNodes[1];
```

The TransactionData instance, in XML, is now in *xmlTransaction*. The actual XML looks like this:

```
  <TransactionData xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Type>Purchase</Type>
    <Amounts>
      <TotalAmount>60850</TotalAmount>
      <CashbackAmount>0</CashbackAmount>
      <VatAmount>0</VatAmount>
      <CashAdvanceChargeAmount>0</CashAdvanceChargeAmount>
      <GratuityAmount>0</GratuityAmount>
      <SelectedCurrency>0</SelectedCurrency>
    </Amounts>
    <Options>
      <DisabledFeatures />
      <TippingRules>
        <MaximumTip>60850</MaximumTip>
        <TipPrompt>AmountPlusExtra</TipPrompt>
        <AllowSkipOnOk>true</AllowSkipOnOk>
      </TippingRules>
      <SelectedCurrency>0</SelectedCurrency>
    </Options>
  </TransactionData>
```

The value of using serialisation, however, means that the transaction can be set up in a class instance, making use of default values, enumerations, etc. and this means the XML that results is guaranteed to be correct.

The resulting XmlNode can then be added to the message as a child of the *patdirect* element.

When this is received in the Pay@Table component, the component will attempt to de-serialise the contents as a TransactionData class and the result will be identical to the TransactionData class instance that was set up in the ECR.

7.8 Handling a transaction result

The response received from the terminal will normally contain two child elements under *patdirect*. One is an instance of the TransactionReport class and the other is an instance of the ReceiptData class. There will only be a ReceiptData if a card is accepted during the transaction. If, for example, the transaction is cancelled or fails before a card is accepted then there will be no receipt data.

These classes, and the classes they use, are also in POSInterface.cs.

The mechanism for deserialising is like this:

```
XmlDocument receivedXml = new XmlDocument();
XmlNode txnResult = receivedXml.SelectSingleNode("TransactionReport");
if (txnResult != null)
{
    XmlSerializer deserialise = new XmlSerializer(typeof(TransactionReport));
    using (XmlNodeReader nr = new XmlNodeReader(txnResult))
    {
        TransactionReport tr = deserialise.Deserialize(nr) as TransactionReport;
    }
}
```

(Note: this code has not been tested, and may need modified for a working implementation.)

7.9 Limitations

7.9.1 Transaction types

The TransactionData class allows for several transaction types. The Pay@Table module will only allow purchase, refund, and reversal transactions.

7.9.2 Signature verification

If signature verification is needed then the cashier must verify the cardholder signature after the transaction is complete. If the signature is not valid then the transaction must be reversed.

7.9.3 Voice referral

If a transaction requires voice referral then it will be declined by the Pay@Table component.

7.9.4 Obscure card requirements

If a card is configured to require any of the following checks or additional pieces of information, the transaction will be declined:

- Payment code
- Embossed imprint

7.9.5 VAT amount

Some cards are configured so that they must ask for a VAT amount. If this happens, a VAT amount of zero will be specified by the Pay@Table component.

7.9.6 Fallback to magstripe

There is functionality in the payment application where if a chip card is swiped in the magnetic stripe reader then the ECR can choose whether or not to accept the card. The Pay@Table component will not allow this fallback.

7.9.7 DCC on original purchase

When a card that is eligible for DCC is used in a refund, the payment application must ask if DCC was used on the original purchase. The Pay@Table component will decline this transaction, rather than require the cashier to provide input. This should not be a problem since a refund transaction is likely to be very rare in a Pay@Table direct transaction situation.