patsystems
powers trades. builds exchanges. manages risk.

Introduction to Patsystems Trading API

Applying, Obtaining Help and Conforming your Application.

Version:	V1.7
Date:	19 February 2012

Any requests for further information or clarification of content should be referred to:

Api_sales@patsystems.com
patsystems (UK) Ltd
Hays Lane, Cotton Centre
London SE1 2QP

Tel: +44 (020) 7940 0490
Fax: +44 (020) 7940 0499

[Type text]

Please read the following information carefully

The information contained in this document is for guidance only. Patsystems accepts no responsibility and shall not be liable for the use of the information in this document or for any reliance placed by any person on any such information. Whilst Patsystems has taken all reasonable care to ensure that the information contained in this document is accurate on the date stated herein, Patsystems gives no warranty as to the accuracy or completeness of such information. Patsystems reserves the right to refuse any application for an API in its absolute discretion.

The application procedures described herein are subject to change without prior notice.

Contents

1 Introduction to the Trading API

You will have obtained this document because you are considering writing to Patsystems Trading API (the PATSAPI). This product provides a C callable library that will allow you to write your own trading application that can connect to our servers. Such an application may be an alternative GUI to our J-trader screen, an interface between a different system and our servers, or a program trading device. The document will explain to you:

- General information about developing to the API
- The cost of obtaining this
- How to apply for it and how your application is processed.
- How to obtain help during development
- How to conform your application with us

Please note that conforming your application to Patsystems does not confer any rights to use the system on you, the applicant. Patsystems reserves the right to revoke the license ID that it issues to you.

2 General Information about Developing to the PATSAPI

2.1 Costs

The API is free to download, but a conformance charge is levied at the time you wish to conduct your conformance test. This charge is currently USD1500. Server access is provided for free, although there are a limited number of connections and inactive connections are subject to removal. Patsystems wishes to encourage third parties to maintain their applications and upgrade to later APIs and provides this server access for that reason.

In a production environment, costs for using the API are negotiated with your clearing firm, not Patsystems. Costs mentioned here are for obtaining and conforming your application only.

2.2 PATSAPI platform and functionality

The Trading API is a C library supplied on the Windows platforms (NT, W2000). You can obtain access to futures trading functionality (logon, submit, amend and cancel orders, view positions and profit) and real-time view prices. It recognises all exchange supported order types.

The PATSAPI receives real-time price updates from the exchange via either a server at your connectivity provider or a local third party quote vendor.

The PATSAPI and connecting Transaction Server (at the connectivity partner's site) both support encryption through the industry standard SSL.

The following table lists a subset of the calls that are made available to you in the library. The actual list is far in excess of this list.

Function	Description
ptLogon	Initiate connection with server
ptSubscribePrice	Request price data from the market data server
ptPriceUpdate (callback)	Notification of an update in market data
ptGetCommodity	List commodity details available to trade
ptGetContract	List maturities available to trade
ptGetTrader	List trader accounts available to submit trades for
ptAddorder	Submit a trade
ptAmendOrder	Amend a trade
ptCancelOrder	Cancel a trade
ptOrder (callback)	Notification of an updated order state
ptGetOrder	Obtain order details and history
ptFill (callback)	Notification of fill details for an order
ptGetFill	Obtain fill details for an order

2.3 Development Environment

You will need to write your application in a language that supports multi threading such as C, C++ (e.g. MSVC++), C#, Delphi. The PATSAPI uses separate threads for function calls and delivering callbacks. In our experience, some languages cannot cope with the multi threading necessary. In our experience, people have not successfully used Visual Basic VB6 due to multi-threading issues. Use of Java is possible by writing a wrapper to the API.

PATSAPI function calls may change from one version to the next, but we try to support backwards compatibility or provide suitable notice of changes to call structures.

2.4 Connection

Your application will make calls to the PATSAPI to initiate two connections – one to the server controlling logon and trading, and the other to the market data server. Even if you do not plan on using market data you are required to make connection to this server.

It is possible to perform some development by connecting the PATSAPI to our demonstration API (demoapi.dll), which is a standalone system that attempts to mimic a connection to the regular trading servers. It can assist in the initial development of your application but cannot fully duplicate the behaviour of a real Patsystems server.

It is possible to obtain server time on our test servers in Chicago or London. These environments connect to the exchange supplied simulation markets. It should be noted that the exchange simulation markets are standalone markets and the prices will reflect the orders that are entered in these markets. They will not be tied in any way to production prices.

If you wish to perform testing against live prices in a simulation environment, you will need to obtain access to our J-simulator product. This environment provides paper trading for a limited number of commodities.

2.5 Getting Help and More Information

While you are developing your application, you can obtain help via the Developer Zone forums at our website, https://secure.patsystems.com/developer/ There is a guest login to these forums if you wish to see what sorts of discussions take place in our developer community. However, this guest login is restricted from entering any posts or downloading files from the forums.

.

3 How to Apply for the API

3.1 How to Apply

You first register at our website at https://secure.patsystems.com/developer/

During the application process, you will be asked to specify the connectivity provider (should you have one) you will use. We will ask for confirmation from the provider so that:

- You know your chosen connectivity provider will accept your proposed application.
- Our clients are made aware of who is planning to connect to the system using non-Patsystems applications written to our API, and can assess the impact of that application on their current system set-up.

When applying, please make sure that your mailbox is capable of receiving attachments that:

- Are up to 2MB in size
- Contain executable files

As an alternative, the kits and documentation can be downloaded from the developer zone.

Once the registration has been confirmed, you will receive an email that allows you to download the API. If you do not receive this, please contact us at api_sales@patsystems.com to find out why you have not received the information.

Registration and download of the API is free.

3.2 Developer Tools

The developers' pack provides a common set of tools to help through the various stages in the development cycle. You will find the following tools available to you

- Application Developers Kit documentation
- A login to forums and discussion boards at https://secure.patsystems.com/developer/
- An email support line at apisupport@patsystems.com for exchanging log files with out API staff.

- Server access to our test servers in London or Chicago. Apply for this through the apisupport@patsystems.com email address.

4 How to conform your application with us

Before you are able to use your application with a live Patsystems Trading Platform, you will require a license ID which will be obtained post conformance.

Although the API is free to download we charge for our conformance tests. The current fee is USD1500 (subject to change) and is payable at the time the conformance test is booked. You may pay this in either Pounds (Sterling) at request of the applicant or US Dollars and payment can be addressed to either the US or UK offices. Further conformance tests can be booked at additional cost (please see the Re-conformance section below).

Once you have conformed, your license number will be forwarded to the administrator at your Patsystems connectivity provider.

4.1 Naming your application

Before you begin any development work you will need to login to the Patsystems Developer Zone section of the Patsystems web site and register a name for your application and any additional contacts that may be involved throughout the work cycle. These will help our support staff in identifying you and should provide for a quicker resolution on any issues.

4.2 Testing your application

The kit comes with a DEMOAPI.DLL file that can be used in conjunction with the PATSAPI to get you started in your development. However, we strongly recommend that you connect to our test servers for the remainder of your development so you experience the behaviour of the production DLL.

We will leave your test logon available for an extended period – Patsystems wishes to encourage you to upgrade your application as new versions of the API are released. However, there are a limited number of connections and stale users are subject to removal.

Please be aware our test environment only supports a limited number of exchanges and that conditions on the Exchange simulation environments do not reflect live trading conditions. If you wish to test against live trading conditions, you can request access to the J-simulator contracts.

Note: The test system is taken down for essential maintenance work from time to time. Patsystems will endeavour to keep these times to a minimum.

4.3 Preparing for the conformance test

The conformance test scripts can be downloaded from the developer zone forums. Once you have developed your application to a stage where you want to take the conformance test, you should return to the developer zone where you will be presented with the next available slot in the conformance calendar. We will liase with you to identify what sections of functionality, if any, that you do not support in your application. These unsupported features will then be ignored in the conformance process and marked in our records as restrictions against the application and the clearer notified.

When we agree which areas of functionality will be tested, we arrange a time for you to take the conformance test and provide you with connection details.

You must use your production application identifier for the test. This assists us in our record keeping process. Should you wish to change the application identifier you may do so now and submit it to us for approval. We recommend names that identify you or your company rather than the type of application you have written.

4.4 How the conformance test is conducted

The conformance test is taken by connecting to our conformance system (using the IP addresses we sent you) and either by phone or web-chat liasing with Patsystems as you go through the conformance script that we have agreed with you. As the test progresses, the Patsystems tester will mark up their copy of the conformance script.

Once the test is finished, you should send us the log files (as explained in the conformance documentation). We will take a copy of any web-chat and these items will be reviewed and you will be informed as to whether your application has passed within 48 hours of having taken the test.

If your application has failed to conform, you will be notified as to why it failed. You should then make any changes needed to your application and book another conformance session. Patsystems cannot modify your code for you.

If your application passes the conformance tests, we will issue a license document that confirms the application that has conformed and notes any restrictions negotiated at the time of testing.

Please note that conforming your application to Patsystems does not confer any rights to use the system on you, the applicant. Patsystems reserves the right to revoke the license ID that it issues to you.

4.5 Obtaining a connection from your Patsystems Connectivity provider

If the connectivity provider is willing to permit you to use your application on their system, they will require the license details when setting up your account on their Patsystems Trading Platform.

If your potential Patsystems connectivity provider has changed while you have been developing your application, or should you wish to change post conformance, please inform the connectivity provider or us. The new provider will need to know license key, application identifier and restrictions.

4.6 Re-conformance

You should be aware that any significant changes in functionality to your application will require re-conformance and that your connectivity provider may revoke access if your application changes in a manner that detrimentally affects system performance.

Additional conformance tests are charged at a maximum rate of $500 at the discretion of Patsystems.

We may find it necessary to change our API from time to time in line with improvements to our core system. Information about these changes are published on our Developer Zone website. These changes do not usually require re-conformance (re-conformance is for significant changes in your application's behaviour).

Patsystems only supports the latest version of its API. The latest version of both the API and FIX gateway can be found on the Developer Zone website. However, Patsystems supports backwards compatibility for a number of version between the API and the Core servers, so your existing application can usually connect to the later versions that your clearer might be using.

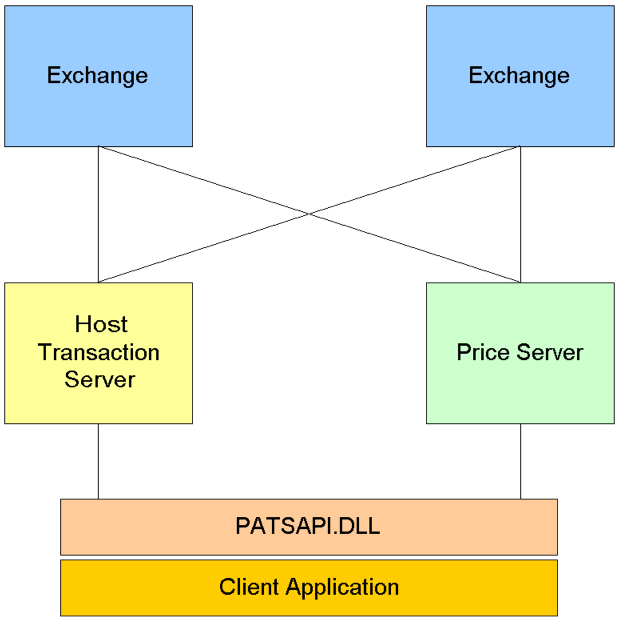
We strongly advise you make regular checks to this section of the web site to check for updates.

5 Developer Kit Samples

The following pages contain some examples taken from our Developer's Guide that you will receive in the kit. These samples are not from the latest data structures and are for informational purposes only.

5.1 Patsystems Architecture

A basic understanding of the architecture and terminology used by the Professional Automated Trading System will be useful when building the application. Note that some terms are interchangeable. Although this document seeks to be consistent, you may hear these terms when speaking to **Patsystems** support.



The client application uses the PATSAPI.DLL (the "API") to submit orders, receive fills and receive prices. The orders are sent to the Transaction Server (a.k.a. the "Host") where they undergo validation and are sent to the correct exchange. Acknowledgements, Rejections and Fills for these orders are returned by the exchanges to the Transaction Server. They are then routed to the API and notification given to the client application.

Prices are received from the exchanges and routed to the Price Server (a.k.a. the "Price Proxy" or the "Price Feed") which directs them to the API only if requested to do so.

5.2 Parameter Passing

The API will accept and return data in one of three formats.

1. Binary number
2. Single 1 byte character
3. Null-terminated character string

Normally, multiple fields will be passed or received. These fields are stored in a packed record - that is, fields in the record are not aligned on boundaries, they occupy sequential bytes in memory.

The binary numbers may be single byte or four byte integer in size. The null terminated string is an array starting at element zero and terminated by the null (ASCII 0) character. That is to say, strings are provided in "C" format.

The parameters passed into the routines are passed either by reference or by immediate value. When a variable is passed by reference, the 32 bit address of the variable appears on the stack. When a variable is passed by immediate value, the actual value appears on the stack. The parameter passing to the API conforms to the following rules.

1. Write access variables are always passed by reference.
2. Integer variables (8 or 32 bit) that are read-only are passed by immediate value.
3. Strings are passed by reference even if they are read-only.
4. Records are passed by reference even if they are read-only.

The Windows API call stack method is used when calling routines. That is, parameters are pushed onto the stack from right to left, registers are not used for parameters and the called routine removes the entries from the stack. Each parameter takes 32 bits regardless of actual size. This means that where an 8 bit integer is passed by immediate value, the entry will occupy the entire 32 bits of the stack entry (although the top 24 bits will be undefined).

5.3 Function Results

Most calls to PATSAPI return a function result as an integer. This value should be examined to decide if the call has succeeded or not. Ignoring the status may lead to unexpected behaviour in your application when communicating with the API.

Some functions do not return results. In most cases, these functions are simple routines requiring little or no validation. Care should still be exercised to ensure that correct data are passed to these routines. Unexpected behaviour may be experienced if invalid information is sent to these routines.

Success is indicated by a result code of zero (equivalent to the **Patsystems** supplied constant *ptSuccess*). Errors are indicated by a positive result. The actual value returned indicates the error condition.

The *ptGetErrorMessage* function can be used to obtain a descriptive error message for the returned value of an API call.

5.4 Initial tasks

There are some initial functions that must be executed before performing any trading using the API. These tasks configure the API for use and perform some basic checks. The initial tasks should be performed in the following order:

1. The API will generate and use files on a given path. The default path for these files is the path of the executable using the API, but this may not always be the most desirable directory in which to place the files for your application. To change the path, call *ptSetClientPath* before initializing the API.

2. Initialise API using *ptInitialise*. This initialises the data structures in the API and must be performed before any other steps. It also accepts the application id. and license number used later to verify access to Patsystems.

3. Set IP configuration details using the following two calls. These calls define the connection details of the Patsystems Transaction Server (Host) and the Patsystems Price Server

4. *ptSetHostAddress*

5. *ptSetPriceAddress*

6. Register the required callback routines using *ptRegisterLinkStateCallback* for the host connection and *ptRegisterCallback* to register for *ptLogonStatus* and *ptForcedLogout* callbacks.

7. Register any other optional callback routines *ptRegisterLinkStateCallback, ptRegisterMsgCallback, ptRegisterPriceCallback, ptRegisterOrderCallback, ptRegisterStatusCallback* and *ptRegisterContractCallback, ptRegisterExchangeRateCallback, ptRegisterConStatusCallback and ptRegisterOrderCancelFailureCallback.*

8. Set any other API control parameters. For example, by calling *ptNotifyAllMsgs*.

9. Set any diagnostic information flags using *ptEnable*. Use these only when initially developing your application, as the functionality results in reduced API efficiency and the diagnostic file can quickly grow to a large size.

10. Start the API processing by calling *ptReady*

After *ptReady* has been called, the API will attempt to connect to the Host using the IP values previously specified. The IP socket will undergo several rapid state changes before becoming connected. The normal state change sequence is Opened-Connecting-Connected.

If the link fails, it will move from the last state to Closed immediately and then wait for a configurable period before re-attempting the connection. This period has a minimum of 5 seconds.

Delphi example: The callback must be a **procedure**. Class methods will not be accepted.

```
type

  {Data passed back by API}
  LinkStateStruct = packed record
      OldState: byte;
      NewState: byte;
      end;

  {Structure is passed by ref.}
  LinkStateStructPtr = ^LinkStateStruct;

{Callback procedure executed by API when link changes}
procedure HostLink (data: LinkStateStructPtr);

begin
case data^.NewState of
  ptLinkOpened:      Form1.Link.Text := 'Opened';
  ptLinkConnecting:Form1.Link.Text := 'Connecting';
  ptLinkConnected: Form1.Link.Text := 'Connected';
  ptLinkClosed:      Form1.Link.Text := 'Closed';
  ptLinkInvalid:     Form1.Link.Text := 'Invalid';
  end;{case}
end;

.
.

{Register the callback in main code}
status := ptRegisterLinkStateCallback( ptHostLinkStateChange, HostLink );
```

Note that the Price Feed is not connected to at this stage. Prices are not available until after a successful log on.

Should it be necessary to disconnect from the servers and then to reconnect without having to close the application, a call to ptDisconnect should be made. This will close links to the host and price feed. It is then also possible to change address of either server by making calls to ptSetHostAddress or ptSetPriceAddress. To re-establish connections, the application must call ptReady, followed by a call to ptLogin.

5.5 Logging in to *Patsystems*

Before commencing to trade, you must complete an application logon to **Patsystems**. In this action the application supplies the Patsystems user name and password for the trader and these are validated on the Patsystems Transaction Server along with the application id. and license number specified in *ptInitialise*. A return status will be returned to your application via the *ptLogonStatus* callback when the logon has been validated.

Logon cannot occur until the API has connected to the Host. This is indicated by the *ptHostLinkStateChange* callback, which will show the old and new states of the IP socket defined as the Host socket in the previous phase.

Complete the logon process by taking the following actions.

1. Wait for *ptHostLinkStateChange* to show new state as "Connected".
2. Issue logon request by calling *ptLogon*.
3. Wait for *ptLogonStatus* callback to fire.
4. Call *ptGetLogonStatus* to obtain logon status details.
5. If status is *ptLogonSucceeded* then wait for *ptDataDLComplete* callback to fire.

The Patsystems trading engine uses the following information to determine if the log on is allowed:

User Name	
Password	
Application ID	entered in *ptInitialise*
License	entered in *ptInitialise*.
Environment	

If logon was not successful, then the reason will be supplied in the data returned by *ptGetLogonStatus*.

If the logon was successful, the API will attempt to connect to the Price Server using the IP address and socket defined in the set-up phase. The status of this connection will be reported by the *ptPriceLinkStateChange* callback.

Delphi example:

```
Uses patsIntf;

var
  Status: Integer;

{API executes this routine when log on result received}
procedure CheckLogon;

var
  LogonResult: LogonStatusStruct;
  Status: Integer;

begin
```

```
  Status := ptGetLogonStatus(@LogonResult);
  If Status = ptSuccess then
  begin
    if LogonResult.Status = ptLogonSucceeded then
      MainForm.OKToProceed := true
    else
      ShowMessage(LogonResult.Reason);
  end
  else
    ShowMessage(ptGetErrorMessage(Status));
end;

{Main code - register callback}
Status := ptRegisterCallback(ptLogonStatus, CheckLogon);
```

5.6 Making Trades

Once the application has logged on to the host, received the reference data update and subscribed to all the prices it wants, the application is in a position to enter trades into Patsystems. The following two points are key to this process:

- All orders processed by Patsystems are identified by a Patsystems Order ID

- All orders undergo several state changes during their lives.

Orders are initially entered into the system by calling *ptAddOrder* at which point the unique Patsystems Order ID identifier is not defined and a temporary identifier assigned of the letter “N” followed by an incrementing number. This temporary identifier can be used to query the order using *ptGetOrderByID* and is replaced by the Patsystems order identifier when the Patsystems servers receive the order and the order state callback shows as “Sent”. The callback for this state change will display the new (official) and old (temporary) Patsystems order identifiers.

When connecting to Linux or Solaris servers, the temporary identifier will not be seen. These servers allocate blocks of Patsystems order identifiers to each API connection and the queued state for the order will show the Patsystems order ID.

During its life, the order will undergo a number of state changes, identified by the *State* field returned by *ptGetOrder(ByID)*. These states are defined in the reference section for the *ptGetOrder* routine and the *ptOrder* callback.

Normally, whenever the order undergoes a state change the callback *ptOrder* will fire, returning the Patsystems Order ID of the order that has changed. There are two identifier fields, and old and new order id. This is used to tie the temporary identifier to the Patsystems order identifier at the point the order goes to the sent state for connections to a standard TAS. Connections to an S-TAS will contain the Patsystems Order ID from the queued order state.

As the order states change, new records are assigned to each order in the list of orders held in the API. The most recent record for each order reflects the most recent state and the earlier ones make up an order history (these history records are held in a separate list for each order). *ptGetOrder(ByID)* will only provide the most recent record pertaining to the order. To obtain historical order information, use the *ptCountOrderHistory* and *ptGetOrderHistory* functions.

Rapid order state changes will trigger the callback for each state change, but you may find that by the time you call the query function to find the new state, the underlying data has been updated to reflect the new order state, leading to the appearance that an order state has been missed. It is important to remember that the query function *ptGetOrder(ByID)* will return the most recent state and the “missing” state will be found in the order history.

As an example, an order might go through the following states in its life:

Queued, Sent, Working, Part Filled, Filled

Existing orders that are still active may be amended using the *ptAmendOrder* call or cancelled using the *ptCancelOrder* routine. The Patsystems order id is specified to these routines in order to identify the order. As well as cancelling a specific order, groups of orders may be cancelled using *ptCancelBuys*, *ptCancelSells* and *ptCancelAll*.

5.7 Data Types and Parameters

This chapter details the function calls provided by the API. The following conventions appear in the document:

Case sensitivity:	The routine names appear in the DLL export table exactly as they appear in this document. However, case-sensitivity is language dependant and some languages may resolve these references regardless of case.
Type “char”	This is a single byte ASCII character.
Type “byte”	This is a single byte integer number.
Type “integer”	This is a four byte integer number.
Type “string[n]”	Strings are zero-based arrays of ASCII 1 byte characters, terminated by the null character. Where an array limit “[n]” is specified the array is deemed to be defined as [0..n] of char. Where no array limit is specified, the string may be any length up to 255 as long as it is null terminated.
Type “struct”	Structures are always packed (byte aligned) records containing the preceding data types.
Floating Point Numbers	Floating point numbers are always passed as ASCII strings. Prices include implied decimal places for contracts ticked in fractions (e.g. 100.08 is $100^{8/32}$ if the contract is in $^{1/32nds}$).
Immediate mechanism	Applies only to read-only parameters of char or integer type. Immediate passing expects a value on the stack and takes up 32 bits regardless of size. For example, a char or byte will occupy the

	bottom 8 bits and the remaining 24 bits are ignored.
Reference mechanism	Applies to any write-access parameter and all string or structure parameters. Reference passing expects the address of the variable or structure on the stack.

5.8 Trading Functions

The following routines are used to manipulate the API for trading and process the results.

---

ptAddOrder

Arguments:	NewOrder	struct	read-only, by reference
	OrderID	string[10]	writeable, by reference
	Returns:	status code	

The ptAddOrder routine submits a new order to the Host. The routine does not verify that the order is sent or accepted by the exchange. This information will be available when the order state changes as indicated by the *ptOrder* callback. This function will return an error code if there is no connection to a transaction server.

The **Patsystems** order ID is automatically supplied by the host and this information will become available when the host processes the order. It will be returned by the *ptOrder* callback and it should be used to query the order details using *ptGetOrderByID*.

Synthetic orders are added using this same call, but are held locally in the API and not sent to the server until the trigger price is seen. Be aware that a disconnection of the price feed (either due to network errors or by calling *ptDisconnect*) will affect the ability to see this trigger price. However, the synthetic orders will still remain in the API, unless a logon is performed with a different username or with the reset flag set.

NewOrder	The address of a structure of type NewOrderStruct containing the order details.
OrderID	The address of a string[10] variable which will receive the temporary Order ID of the new Order. This will be 'S1' to 'Sn' for new synthetic orders or 'N1' to 'Nn' for new orders on regular TAS. When connecting to the S-TAS, the 'N1' style temporary id will not be seen. Instead, the actual order-id will be seen as the API pre-allocates real order-ids on the server.

NewOrderStruct is defined as:

TraderAccount	A string[20] variable containing a valid trader account for the user, as returned by ptGetTrader. May be set to any string that equates to a valid trader account for the logged in user.
OrderType	A string[10] variable containing the order type. This is one of the values returned by ptGetOrderType.
ExchangeName	A string[10] variable containing the exchange name for the order. This must be the valid exchange name for the contract, as returned by ptGetContract.
ContractName	A string[10] variable containing the contract name for the order. This is one of the ContractName values returned by ptGetContract.
ContractDate	A string[50] variable containing the contract date of the contract, as returned by ptGetContract.
BuyOrSell	A char variable either 'B' or 'S'.
Price	A string[20] variable containing the target price for the order. May be any real number. If the order type does not require a price, as defined by *ptOrderTypePriceRequired* then this field must be set to blank.
Price2	A string[20] variable containing a second price if required. For example, a limit price for a stop/limit order. Blank if not required.
Lots	An integer variable containing the number of lots for the order. May be any positive integer.
LinkedOrder	A string[10] variable containing the **Patsystems** Order ID of the order linked to this order if it is an OCO order.
OpenOrClose	A char variable either 'O' or 'C' indicating if the Order is to open or close the traders position.
Xref	An integer variable containing a user supplied cross-reference number. This cross-reference is no longer valid if the API is exited.

XrefP	An integer variable containing a user supplied cross reference number. This cross-reference persists even if the API is shutdown. The cross-reference field is one of the many fields used to identify an order when a state transition is made from queued to sent on a standard TAS. This identification is required in order to assign the Patsystems Order ID to the correct temporary 'Nn' order ID. Patsystems recommends specifying a unique XrefP along with every order, especially when rapid order entry is expected (that is, when there may be numerous identical orders all transitioning from queued to sent at the same time).
GoodTillDate	A string[8] variable containing the good till date for the order as CCYYMMDD, or blank if not required. For a good till cancelled order, leave this blank.
TriggerNow	A char variable either 'Y' or 'N' indicating if synthetic orders should be checked (and triggered if necessary) immediately rather than awaiting a price update message.
Reference	A string[25] variable containing a user supplied cross reference similar in function to the XrefP but allowing text. Should be specified in addition to, not as a replacement of, XrefP.

The routine returns the following error codes:

ptSuccess	The order has been sent to the host for processing (not the exchange).
ptErrNotInitialised	The API has not been initialised.
ptErrNotLoggedOn	The API has not logged on to the host.
ptErrPriceRequired	The order type requires a price and one was not provided.
ptErrInvalidPrice	The supplied price did not convert to a valid real number
ptErrUnknownAccount	The trader account does not match a known trader.
ptErrUnknownOrderType	The order type is not known to PATS.
ptErrUnknownContract	The contract name / date does not refer to a valid contract.
ptErrTASUnavailable	The Transaction Server is not connected.
ptErrMDSUnavailable	The Market Data Server is not connected.