OPENPRICING AND OPENRISK MANUAL

Andrew Colin

Flametree Technologies

C:\USERS\ANDREW\DROPBOX\OPENPRICING AND OPENRISK MANUAL.DOCX

This page intentionally left blank

CONTENTS

Introduction 5
Limitations in fixed income analytics platforms 5
Perturbational attribution 5
First-principles attribution 6
OpenPricing 6
OpenRisk 6
Plug-ins 8
What is the difference between an OpenPricing plug-in and an OpenRisk plug-in? 8
Using plug-ins 9
Using an OpenPricing plug-in 9
Using an OpenRisk plug-in 9
Data requirements 10
Overview 10
Security master data 10
Effective dating 12
User-defined fields 12
Weight and returns data 12
Yield curve data 14
Index data 14
Using plug-ins 15
Plug-ins with no arguments 15
Supplying additional information to plug-ins 15
Types of additional argument 15
Values of volatile market quantities 15
Values of yield curves 16
Values of indices 17
Multiple arguments 17
Passing historical data into plug-ins 17
Appendix A: Writing plug-in functions 19
Overview 19
Writing an OpenPricing plug-in 20
Data structures 20
Using additional arguments 21

Support functions 21

Sample OpenPricing plug-in code 22

Writing an OpenRisk plug-in 23

Sample OpenRisk plug-in code 23

Data integrity in plug-ins 25

Compiling and linking plug-ins 25

Installing a plug-in 25

Verifying plug-in compatibility 26

INTRODUCTION

This manual describes OpenPricing and OpenRisk, a recently developed set of additions to Flametree's FIA fixed income attribution engine.

- *OpenPricing* allows new security types to be defined via external pricing functions. These can be based on proprietary code, called from a third-party library, or written from scratch.
- *OpenRisk* allows the user to define new sources of risk in attribution reports, via external risk functions.

OpenRisk and OpenRisk allow FIA's capabilities to be extended and customized by Flametree's users to keep up with developments and innovations in the marketplace, thereby future-proofing your investment in our attribution technology.

LIMITATIONS IN FIXED INCOME ANALYTICS PLATFORMS

A common problem in the provision of fixed income analytics platforms is the pace of development in the fixed income markets. Frequently, new security types become available that cannot be handled by existing analytics libraries and software. In addition, the sheer number and complexity of security types traded in today's markets means that provision of detailed, accurate security modelling can exceed the capabilities of even the largest analytics providers.

For attribution, the main requirement is the ability to measure the effect of changes in the underlying market on the price, and hence the return, of an arbitrary security. Here we describe the two main approaches to this problem.

PERTURBATIONAL ATTRIBUTION

One solution is the introduction of 'one-size-fits-all' algorithms, in which a perturbational approach is used as a proxy for a full pricing function. Under this approach, a set of sensitivity measures are supplied which, when combined with appropriate market movements, provide close approximations to security-level returns.[1]

This approach has been widely adopted by vendors of attribution software. However, it requires that sensitivity measures such as modified duration, convexity and yield to maturity be supplied for all securities at all dates, which can cause severe operational difficulties (volume, accuracy, workflow).

In addition, the use of a perturbational model requires customization for particular security types. For instance,

- an inflation-linked bond requires an additional source of return due to inflation carry;
- a floating rate note has two different measures of sensitivity to interest rate changes;
- an Australian bond future generates no carry return.

All these idiosyncrasies must be known and accounted for before accurate attribution analysis can begin. This technique therefore only partially solves the problem of new asset types, but does so at considerable cost to the user.

[1] For a full account of the mathematics of perturbational models, see Colin, A., Mastering Attribution in Finance, FT Press, 2015

A further disadvantage of the perturbational model is that it models securities as a single cash flow. This can lead to substantial loss of accuracy for amortizing bonds and other security types with complex cash flow structures, such as MBS and ABS.

FIRST-PRINCIPLES ATTRIBUTION

An alternative approach is to supply individual pricing functions for each asset class, to calculate the timing and amounts of all cash flows, and to price these cash flows using standard discounting approaches. The security price is then the sum of the values of its discounted cash flows.

This approach uses order of magnitudes less data than the perturbational approach, since no risk numbers are required. More computation is required, but this is seldom a problem given low CPU costs.

The only disadvantage of this approach is the user must accept the vendor's pricing model. Updates and enhancements are at the discretion of the vendor, so the decision to lock in to a particular vendor can be a business-critical decision.[2]

OPENPRICING

FIA addresses this problem by writing the pricing library as a functionally distinct component from the core application. Each security modelled is linked to a particular pricing function that is provided in an external pricing library, typically a dynamic link library (for Windows) or a shared object (for Linux).

Unlike other commercial attribution systems, the user is free to write new pricing functions can be written and called from within FIA, using the API described in this document. Although it may never be necessary to use this ability, the user

The advantages of this separation between pricing and attribution are several-fold:

- New pricing functions for new instruments can be written and integrated within days, ensuring analytics support can keep up with new developments in the market.
- Changes can be made without reference to the vendor, retaining client control over deployment of new security types.
- Custom pricing functions and libraries can be added at will. If the user has a third-party or proprietary pricing library used for risk and hedging, this library can also be called from FIA.

OpenPricing therefore solves the problem of vendor lock-in, by allowing the user to take complete control of all pricing models used.

The requirement to use the OpenPricing API may never arise, as we supply a comprehensive library of security types. However, if

OPENRISK

FIA calculates all quantities that are related to the level of, or movements in, the yield curves that determine a security's price. These include

[2] FIA supports both approaches. If no pricing function is supplied, the security's returns are assumed to be modelled by the perturbational equation, in which case risk numbers must be supplied.

- *carry return*, generated by the passage of time
- *curve return*, generated by movements in the security's underlying risk-free curve
- *credit return*, generated by changes in the spread between the security's risk-free curve and its pricing curve

FIA also calculates a range of other minor curve-related effects, such as rolldown and convexity.

The common factor amongst all these returns is that they may be calculated from changes in the security's price. For instance, carry return may be calculated by pricing a security at the beginning and end of an interval but keeping all curve levels unchanged. In this case the only variable to change is time, so any change in the price of the security is generated by elapsed time.[3]

Similarly, curve return is calculated by pricing the security with two different sets of yield curves but at the same date. The change in price, and hence return, arising from the change in curve level is allocated to curve return.

These sources of return correspond to the main risks for the vast majority of fixed income instruments. However, there are a significant number of instruments that can generate return from additional non-curve risks. These include

- *Inflation carry* for inflation-linked securities. This return is generated by the indexation of the bond. As inflation pushes up prices, the coupons and maturity payment of the bond rise in step. The extra return generated by this feature is not driven by any yield curve, and must be calculated using values of the inflation index associated with the bond.
- *Interest return* on cash holdings. Although cash deposits generate return for the investor, this return cannot be measured using changes in the price of the security, as the price of cash is always the same.
- *Paydown*. Paydown return is generated when the principal on an amortizing security is repaid at a faster rate than expected. Depending on whether the security is trading at a premium or a discount, paydown return can be positive or negative.
- *Returns from options*. In principle, one could measure the return due to a bond's embedded option from changes in the bond's OAS (option-adjusted spread), minus the effect of any curve spreads. In practice, this would result in considerable uncertainty if the level of the credit curve is not known precisely. In addition, this method would not provide any further information on the returns generated by the bond's embedded option due to its idiosyncratic risks (passage of time, first and second order price changes in the underlying security, interest rates, volatility).

FIA therefore allows the user to define additional sources of risk for individual securities. For each source of risk, a risk plug-in is defined that calculates the return for the security due to that risk. When the attribution report is run, that source of return is shown on the attribution report.

[3] In practice, it is preferable to calculate the security's yield to maturity and to calculate the return from that, which avoids issues with coupon timing; but the principle is the same.

PLUG-INS

OpenPricing and OpenRisk are implemented using *plug-in* functions.

A plug-in is a small stand-alone program that reads in the values of quantities such as the current date, the structural measures of a security such as its maturity date and coupon, and the current yield curve for that security. An OpenPricing plug-in returns a security price at a given date, while an OpenRisk plug-in calculates a rate of return between two dates.

Each plug-in has a unique name describing the quantity that it calculates. For instance,

- `generic_bond_price` returns the dirty price of a generic bond
- `black_scholes_delta` returns the delta of an equity option priced using the Black-Scholes model
- `Inflation` supplies the return due to inflation carry for a given date interval for an inflation-linked bond.

Plug-ins exist inside compiled libraries. A single library can contain many plug-ins, or each plug-in can reside within its own library.

FIA is designed so that if the name of a plug-in is supplied in a security master record, then that plug-in is used when running attribution on that security. In effect, the user programs the FIA to use a particular pricing algorithm, or to calculate a desired set of returns, simply by supplying the names of the required plug-ins.

FIA supplies a ready-to-use library of plug-ins that cover most client requirements. Custom plug-ins can also be written by the user. Refer to Appendix A for more information on writing plug-ins.

WHAT IS THE DIFFERENCE BETWEEN AN OPENPRICING PLUG-IN AND AN OPENRISK PLUG-IN?

An OpenPricing plug-in calculates a price of a security at a given time, while an OpenRisk plug-in calculates the return of a particular risk between two dates.

Plug-ins for OpenPricing and OpenRisk share many features, so we treat them together in the following sections. Their main difference lies in their function signature. An OpenPricing plug-in has the function signature

```
double f( long t, FT_SECURITY_DATA s, FT_YIELD_CURVE c, ... );
```

while an OpenRisk plug-in has the function signature

```
double f( long start_t, long end_t, FT_SECURITY_DATA s, FT_YIELD_CURVE c, ... );
```

USING PLUG-INS

USING AN OPENPRICING PLUG-IN

To use an OpenPricing plug-in for pricing, write the name of the plug-in function to the ‘Pricing’ column (column 5) in the security master table. Note that

- At most one OpenPricing function can be written in this field.
- The OpenPricing function must exist within a pricing library
- The OpenPricing function cannot have been defined more than once.

If the ‘Pricing’ column is left blank, FIA assumes that the security is to be modelled using a perturbational model, in which case a yield to maturity and a modified duration must be supplied for all dates in the weights and returns table.

If any of these conditions are not met, FIA will exit with a diagnostic message when you try to run. Otherwise, FIA will use the named plug-in whenever the current security has to be priced.

USING AN OPENRISK PLUG-IN

To use an OpenRisk plug-in to calculate additional returns, write the names of the plug-in functions to the ‘Risk functions’ (column 6) column in the security master table, with additional arguments if required. Note that

- Any number of OpenRisk functions can be written in this field, as long as their names are separated by a vertical pipe ('|') symbol.
- The OpenRisk function must exist within a library
- The OpenRisk function cannot have been defined more than once.

If any of these conditions are not met, FIA will exit with a diagnostic message when you try to run. Otherwise, FIA will calculate the additional returns specified by the list of OpenRisk functions.

DATA REQUIREMENTS

OVERVIEW

FIA uses four two-dimensional arrays of information as its starting point, referred to for convenience as *tables*. These are

- the *security master table*
- the *weights and returns table*
- the *yield curve data* table
- the *index data* table.

Not all fields need be populated to run the FIA, depending on the plug-ins used. For instance, if the pricing plug-ins only use yield curves, the market data table can be left empty. Conversely, if all securities are priced using a supplied yield to maturity from the market data table, there is no need to supply yield curve data.

In addition, some table fields may be calculated rather than supplied from external sources. For instance, if an option requires the value of an underlying security's annualised volatility, then the user may opt to calculate this quantity from a supplied return within a plug-in. This capability can substantially reduce FIA's data requirements.

SECURITY MASTER DATA

The security master table contains information about each security, including the names of the security pricing function (used by OpenPricing) and any supplementary risk functions (used by OpenRisk).

Column	Field	Type	Description	Sample	Rules
1	Security ID	String	Unique identifier for security	CASH DE0000015100	Must have at least three characters
2	Security name	String	Name of security	MEGACORP 5% 1st JAN 2050	Must have at least three characters
3	Classification	String	Names of bucket and corresponding value	COUNTRY=US|INDUSTRY=TELECOMS	
4	Effective date	Date	Date at which record becomes effective. If no value is recorded, record is assumed active over all dates	20-DEC-2016	Can be blank
5	Pricing function	String	Plug-in names to be evaluated for this security, with arguments if required	GENERIC_BOND_PRICE GENERIC_BILL_PRICE	May contain a valid plug-in name. If field is blank, a perturbational calculation is performed.

6	Risk functions	String	Name of risk function(s) specifying which addition risks are to be calculated for this security.	INFLATION DELTA|GAMMA|THETA|RHO	Can either be left empty, or contain the names of one or more OpenRisk functions. If more than one function is recorded, names must be separated by a pipe ('|') symbol.
7	Credit rating	String	Credit rating of security	AAA, AA-, B+	
8	Effective exposure	Integer	Flag indicating whether effective exposure multiplier is zero or one	0,1	Should be zero for futures. If left unset, defaults to 1.
9	Currency	String	3-character currency code	AUD, USD, GBP	Currency codes must follow the ISO 4217 standard
10	Residual sector	String	Name of returns category into which to write residual return for current security	[Blank] Optionality return Country spread duration	
11	Curve	String	Name of yield curve used by this security for pricing	USD_CURVE	Must be defined in yield curve table
12	Start date	Date	Date at which security's cash flows start.	15-FEB-2020	Can be blank
13	Maturity date	Date	Date at which security matures and its cash flows stop	15-Dec-2025	Must either be blank or a string that can be parsed as a date, using the DateFormat or SecurityDateFormat strings
14	Coupon	Real	Annual coupon	0.05	Must be provided as a decimal. A 5% coupon is recorded as 0.05.
15	Frequency	Integer	Number of coupons per year	2	
16	Strike	Real	Strike price for option	100.00	
17	Term	Real	Term of security in years. Only applicable to sinking securities	30	
18	PSA rate	Real	Repayment rate for securities that	0.025	

			use the PSA prepayment model		
19	Paydown	Real	The fraction of the bond's principal that is outstanding at the given effective date	0.85	
20-148	User-defined	Real	User-defined terms	0.54	

Table 1: Security master table

EFFECTIVE DATING

The structural parameters of many securities change over time. To model this occurrence, FIA allows multiple records with the same security identifier (field 1) but different effective date (field 3). Each record becomes active on and after the effective date, superseding any previous record.

USER-DEFINED FIELDS

Fields in columns 16 to 144 are user-defined.

In some cases, additional parameters are needed to price a security, in addition to the standard measures recorded in columns 6 to 15. These parameters can be stored in user-defined fields, from where they are passed automatically to all plug-ins.

WEIGHT AND RETURNS DATA

This may not be required at all. However, if (for instance) a security has been defined so that its pricing function requires a yield to maturity, it should be recorded in this table.

The pricing and risk data table is also used to store quantities such as the price of an underlying security, if this is required to price an option or other derivative. This table is used to store security-specific quantities that may change on a daily basis.

In principle, all the quantities in this table may be calculated rather than supplied. However, in many cases it may be more convenient, or consistent with market practice, to supply one or more starting points from market data.

Column	Field	Type	Description	Sample	Rules	Required
1	Date	Date	Unique identifier for security	CASH DE0000015100	Must have at least three characters	Yes
2	Portfolio	String	Portfolio in which current security is held	STF1	Must have at least three characters	Yes
3	Security ID	String	Name of security	DE0000123456	Must have at least three characters	Yes

4	Market weight	Real	Weight of current security in portfolio	1200 0.1	Can be an asset allocation or a market value. If sum over portfolio is not 1, values are renormalized	Yes
5	Base currency return	Real	Base currency return of security over current interval	0.01	Must be entered in absolute terms (eg 0.56% is recorded as 0.0056)	Yes
6	Local currency return	Real	Local currency return of security over current interval	-0.0145	Must be entered in absolute terms (eg 0.56% is recorded as 0.0056)	Yes
7	YTM	Real	Yield to maturity			No
8	Modified duration	Real	First-order sensitivity of price to risk-free interest rate changes			No
9	Convexity		Second-order sensitivity of price to risk-free interest changes			No
10	OAS	Real	Option-adjusted spread			No
11	Spread duration	Real	First-order sensitivity of price to interest rate spread changes			No
12	Z-spread	Real	Z-spread			No
13	Price	Real	Price of security at current date			No
14	Volatility	Real	Annualised volatility			No
15	Delta	Real	First order sensitivity of an option's price to changes in the price of the underlying			No
16	Gamma	Real	Second order sensitivity of an option's price to changes in the price of the underlying			No
17	Theta	Real	Sensitivity of an option's price to changes in time			No
18	Rho	Real	Sensitivity of an option's price to changes in risk-free interest rates			No

19	Vega	Real	Sensitivity of an option's price to changes in volatility			No

Table 2: Market data table

YIELD CURVE DATA

This table contains all information on yield curves.

Column	Field	Type	Description	Sample	Rules
1	Date	Date	Unique identifier for curve	CASH DE0000015100	Must have at least three characters
2	Curve ID	String	Name of curve	USD Treasury curve	Must have at least three characters
3	Maturity	Double	Tenor point on yield curve, in years	10.5	Must be a real number
4	Yield	Double	Value of yield curve at given tenor	0.0543	Must be provided as a decimal. A 5% yield is recorded as 0.05.

Table 3: Yield curve table

Field 'Yield' can contain par or zero curve data, depending on requirements

INDEX DATA

This table contains single-valued time series such as the value of a stock index, LIBOR, BBSW, or inflation indices. Index data can also be used to record paydown schedules for individual securities and similar security-specific information.

Column	Field	Type	Description	Sample	Rules
1	Date	String	Date	23-Dec-2016	
2	Index	Date	Unique identifier for index	LIBOR	Must have at least three characters
3	Value	Real	Value of index	0.00344	

Table 4: Index table

USING PLUG-INS

The parse expression contains the name of one or more plug-in functions. These specify which calculated quantities will be computed for the current security.

The parse expression can vary between securities, depending on which quantities are required.

PLUG-INS WITH NO ARGUMENTS

Internally, each function has three default inputs:

- Current dates
- Security master information
- Yield curve

These inputs are set up within FIA and are always supplied by default to all plug-ins.

This is sufficient information for the majority plug-in functions to be evaluated. For instance, a generic corporate bond that is priced using discounted cash flows can have its coupon and maturity dates calculated using the security master information. The current value of the bond’s future cash flows can be calculated by using appropriate discount rates from the supplied yield curve, and the bond’s current price will then be their sum. In this case, the plug-in’s name is all that is required in the bond’s pricing function field.

SUPPLYING ADDITIONAL INFORMATION TO PLUG-INS

In some cases, more information must be supplied to call a plug-in, in addition to the three default inputs described above. For instance

- a floating rate note requires the value of an interest rate index such as LIBOR in order to calculate its coupons
- ,
- an option requires the price and volatility of the underlying security

None of these additional inputs are available from the security definition or the single yield curve associated with the security. Therefore, any further information required by a plug-in must be passed to the function using additional arguments.

TYPES OF ADDITIONAL ARGUMENT

FIA is designed so that any information available in its data tables can be passed to a pricing function.

FIA supports three types of argument:

VALUES OF VOLATILE MARKET QUANTITIES

This data is read from the weights and returns table, which contains data that can change on a daily basis, such as price, delta, gamma and convexity. Data from this table must refer to a particular security, which need not be the current security.

Values of market quantities are always referred to as

```
SECURITY_IDENTIFIER[FIELD_IDENTIFIER]
```

where

- `SECURITY_IDENTIFIER` is the ID code for a security, found in Column 1 of the security master data table, and
- `FIELD_IDENTIFIER` is one of `"YTM", "TTM", "MD", "CONVEXITY", "SMD", "OAS", "Z_SPREAD", "PRICE", "VOL", "DELTA", "GAMMA", "THETA", "RHO", "VEGA"`, and is always enclosed in square brackets.

For instance, the price of a security with identifier `BHP` is `BHP[PRICE]`. If a routine to price a derivative security called `OPTION_BHP` requires the price of `BHP`, this price must be passed as an argument: `OPTION(BHP[PRICE])`.

Similarly, the Z-spread of a corporate bond with identifier `CORP` is expressed as `CORP[Z_SPREAD]`.

Note that

- The security referred to by `SECURITY_IDENTIFER` must have been set up in the portfolio table. If the identifier of a security is used that does not exist, FIA will generate an error.
- Security labels must have an argument, and the argument must be one of the data labels in the above list. If a data label is used that is not defined, FIA will generate an error.
- Labels must be surrounded by square brackets.
- A security can have one, and only one, label for a given security. If you need the values of OAS and Z-spread for a bond CORP, this must be passed as two separate arguments `CORP[Z_SPREAD]` and `CORP[OAS]`, not as `CORP[Z_SPREAD, OAS]`.
- If the requested data does not exist in the security master table, an internal value of NaN ('not a number') will be returned. Plug-ins should always be written to detect this case and to exit with a diagnostic message.

VALUES OF YIELD CURVES

This data is read from the yield curve table. FIA allows points at a given tenor point on a yield curve to be passed as a function argument. For instance, the 10-year yield on a USD curve with name `USD_CURVE` is expressed as `USD_CURVE[10]`.

Note that

- The curve name must have been set up in the yield curve table. If a curve is referenced that does not exist, FIA will generate an error.
- Curves must have an argument, and this argument must be a real number, showing the required maturity point on the yield curve. Negative numbers are accepted, but this will return the shortest maturity yield available.
- The argument must be enclosed in square brackets.
- If the argument is lower that the shortest quoted maturity for the yield curve, the yield corresponding to the shortest quoted maturity is returned.
- If the argument is higher that the longest quoted maturity for the given yield curve, the yield corresponding to the longest quoted maturity is returned.

VALUES OF INDICES

This data is stored in the index table.

For instance, the value of LIBOR may be supplied to a floating rate note with pricing function `FT_FRN` by entering `FT_FRN(LIBOR)`.

- The index must have been set up in the portfolio table. If a non-existent index is accessed, FIA will generate an error.
- Unlike volatile market quantities and curve values, indices do not require an argument. If an argument is supplied, an error will be flagged.

MULTIPLE ARGUMENTS

Different types of argument can be mixed and matched for a single plug-in. For instance, suppose an exotic derivative has an associated plug-in called EXOTIC that requires the value of LIBOR over the last week, the value of the USD Treasury curve at the 10-year point, and the volatility of the FTSE index. Assuming these quantities are all stored in the appropriate tables, the plug-in is called as follows:

```
EXOTIC (LIBOR, USD_TREASURY[10], FTSE[VOL])
```

Multiple plug-in arguments must be supplied as a comma-delimited list and enclosed in round brackets.

FIA accepts up to 8 arguments for each plug-in.[4]

PASSING HISTORICAL DATA INTO PLUG-INS

Often, the only additional data required by a plug-in function is the current, or most recent value, of a variable. For instance, the Black-Scholes pricing routine only requires the current price and volatility of the option's underlying security to generate the current price.

However, in many cases a plug-in requires additional historical information on the variable's value. FIA meets this requirement by passing a complete history of each variable to the plug-in, up to and including its value at the current date. FIA's architecture allows this information to be made available with almost no run-time penalty.

For instance,

- an inflation-linked bond that uses the Canadian pricing model requires the most recent three values of the associated RPI (Retail Price Index). By passing a complete history of the RPI to the pricing function, this information is always available, as long as there is a sufficient history of the RPI in the index table.
- an option pricing routine requires the volatility of the underlying security over the previous year, which may not always be available. If the return history of the underlying security is available over the year, however, its annual volatility can be calculated by using the returns history for the security, relieving the user of the requirement to supply historical volatility.

[4] This number may be increased in future releases, depending on user requirements.

The time series passed to pricing functions contain both values of the given values and the dates at which they are quoted. This allows values at particular dates to be retrieved. For instance, an FRN may have its coupon reset depending on the value of LIBOR on the 1st of each month. By providing a daily history of LIBOR as a collection of dates and values, the pricing function can always retrieve the appropriate value of LIBOR by referring to dates, rather than just using the most recent entry.

For more information on the structure of time series used in OpenPricing and OpenRisk, see Appendix A.

APPENDIX A: WRITING PLUG-IN FUNCTIONS

OVERVIEW

A plug-in is a compiled function […]. Each function must have the signature

An OpenPricing plug-in has the function signature

```
double f( long t, FT_SECURITY_DATA s, FT_YIELD_CURVE c, ... );
```

while an OpenRisk plug-in has the function signature

```
double f( long start_t, long end_t, FT_SECURITY_DATA s, FT_YIELD_CURVE c, ... );
```

where the data structures **`FT_SECURITY_DATA`** and **`FT_YIELD_CURVE`** are defined below. Note that the ellipsis at the end of the function argument should be transcribed exactly as written here.

The library can be written in C, C++, or any other language that allows routines to be compiled into 64-bit DLL or shared objects that are compatible with the core FIA application.

To write a plug-in that calls a routine in another language such as Python, a wrapper function that can pass data from C to Python and back is required. Please contact Flametree technical support for more information.

Under Windows, plug-ins should be compiled and linked as 64-bit DLLs (dynamic link libraries).

Under Linux, plug-ins should be compiled as 64-bit shared objects.

Plug-ins written for FIA are 100% compatible with the Flametree FDE analytics engine, and vice versa.

WRITING AN OPENPRICING PLUG-IN

DATA STRUCTURES

Consider a simple OpenPricing routines with the following signature:

```
double TEST ( long settle, FT_SECURITY_DATA s, FT_YIELD_CURVE c, ... )
```

settle provides the date at which the pricing calculation is performed. This is supplied automatically from the analytics engine.

settle is a Julian date. A Julian date is a 'C' `long int` that measures the current date as an offset from January 1st 4713 BC. Julian dates are used throughout FIA as they can represent maturity dates later than 2038, which is not the case for the conventional `struct tm` structure in C.

FT_SECURITY_DATA is a 'C' structure defined in header file **OpenPricing.h.** This structure takes the form

```
typedef struct  {
    long start_date;
    long maturity;
    double coupon;
    double coupon_frequency;
    double term;
    double repayment_rate;
    double factor;
    double effective_exposure;
    double strike;
    double cash_rate;
    double user_defined [128];
} FT_SECURITY_DATA;
```

The values of all fields are filled in automatically, where available, from data supplied in the security master file. Additional pricing quantities can be passed to the plug-in function via the user-defined array at the end of the list.

FT_YIELD_CURVE is a set of nested 'C' structures as follows:

```
typedef struct {
    size_t n;
    FT_YIELD_TUPLE *data;
} FT_YIELD_CURVE;
```

and

```
typedef struct {
    double tenor;
    double yield;
} FT_YIELD_TUPLE
```

The final ellipsis in the plug-in's signature allows additional parameters to be passed in the form of **FT_TIME_SERIES** structures. Up to 8 additional time series may be passed to each plug-in, where each time series is defined using the structures

```
typedef struct {
    size_t n;
    FT_DATA_TUPLE *data;
} FT_TIME_SERIES;
```

and

```
typedef struct {
    long date;
    double value;
} FT_DATA_TUPLE;
```

USING ADDITIONAL ARGUMENTS

Plug-ins use some special 'C' syntax in order to handle addition arguments. If an extra time series is passed to a plug-in as an additional argument, its values can be accessed as follows:

```
FT_TIME_SERIES t1;
va_list ap;
va_start( ap, c ); // makes ap point to 1st unnamed arg
t1 = va_arg( ap, FT_TIME_SERIES );
double price = t1.data[0].value;
```

For a more detailed example that uses several values from the additional time series, refer to the inflation-linked bond code shown below. Multiple time series can be accessed using the same syntax.

SUPPORT FUNCTIONS

The module OpenPricing.cpp contains the following routines, which may be useful when writing custom plug-ins. Flametree's fixed income pricing routines make extensive use of these routines but the user is free to use alternatives, such as the Boost datetime library.

Name	**Description**
`void FromJul ( long, int*, int*, int* );`	Converts Julian date to conventional date (day, month, year)
`long ToJul ( int, int, int );`	Converts conventional date (day, month, year) to Julian date
`long PreviousCouponDate ( long, int );`	Decreases a date by a given number of months, and applies any corrections required (eg 31st August becomes 28th February, not 31st February, or 29th February on a leap year)
`double InterpolateYield ( const double, const FT_YIELD_CURVE& );`	Uses linear interpolation to calculate the yield at a given maturity for the supplied yield curve

`double price_discounted_cashflow ( const long, const long, const double, const int, const FT_YIELD_CURVE& )`	Given the current date, the date of a cashflow, and a curve, this routine returns the value of the cashflow when discounted at the appropriate rate.

SAMPLE OPENPRICING PLUG-IN CODE

The following code shows how to price a generic bond, and illustrates the usage of the various data validation routines.

```
///////////////////////////////////////////////////////////////////////////////
// Calculate generic bond price from cash flows, zero curve
///////////////////////////////////////////////////////////////////////////////


double FT_BOND_ZERO_CURVE ( long j_settle, FT_SECURITY_DATA s, FT_YIELD_CURVE c,
... )
{
    double coupon = s.coupon;
    double frequency = s.coupon_frequency;
    long j_maturity = s.maturity;
    long j_start_date = s.start_date;

    // Check that all compulsory settings are present
    FT_CheckParameterDefined ( coupon, "coupon", __FUNCTION__ );
    FT_CheckParameterDefined ( j_maturity, "maturity", __FUNCTION__ );
    FT_CheckParameterDefined ( frequency, "frequency", __FUNCTION__ );
    FT_CheckCurveDefined ( c, __FUNCTION__ );

    if ( isnan( ( double )j_start_date ) )
        j_start_date = 0;

    if ( frequency == 0 )
        frequency = 1;

    // If bond has already matured, return price of 1
    if ( j_settle >= j_maturity )
        return 1.0;
    else
    {
        long j_date = j_maturity;  // j_date is used as iteration variable

        // Price repayment of bond's principal
        double price = FT_PriceDiscountedCashflow ( j_settle, j_date, 1.0,  ( int
)frequency, c );

        // Calculate and price all other cash flows, taking account of start
        // date (if set)
        while ( ( j_date > j_settle ) && ( j_date > j_start_date ) )
        {
            price += FT_PriceDiscountedCashflow ( j_settle, j_date, coupon /
frequency,  ( int )frequency, c );
            j_date = FT_PreviousCouponDate ( j_date, ( int )frequency );
        }

        return price;
    }
}
```

WRITING AN OPENRISK PLUG-IN

All information supplied to OpenPricing plug-ins also applies to OpenRisk plug-ins. In particular, the contents of the structures FT_SECURITY_DATA and FT_YIELD CURVE are automatically supplied to OpenRisk plug-ins. However, rather than a single date, an OpenRisk plug-in is passed a start and end date. All this information is supplied automatically.

Additional data can be supplied to OpenRisk routines in exactly the same manner as for OpenPricing routines. For instance, a routine to calculate inflation carry might have the name 'Inflation'. As CPI data is required to calculate this quantity, inflation carry can be calculated for inflation-linked bonds by setting field 6 to

```
Inflation(CPI_INDEX)
```

where CPI_INDEX is the name of a CPI index, stored in the Index table.

VIEWING THE VALUES OF OPENRISK CALCULATIONS

If at least one security has an OpenRisk function set, then an additional source of return corresponding to that function will be displayed in all reports.

The name of the return source will be the name of the OpenRisk function. For instance, if the function is called 'Inflation', then a new column called 'Inflation' will appear in FIA's drill-down reports, in the summary attribution report, and in all other attribution reports.

SAMPLE OPENRISK PLUG-IN CODE

The following code shows how to calculate inflation carry, and illustrates the usage of the various data validation routines.

```
////////////////////////////////////////////////////////////////////////////////
// Calculate inflation rate for given index, date
////////////////////////////////////////////////////////////////////////////////

double CalculateIR ( const long date, const FT_TIME_SERIES t )
{
    if ( t.n < 2 )
        cerr << "ERROR: Not enough CPI index data for inflation calculation" <<
endl;

    long date_start = t.data[0].date;
    long date0 = t.data[1].date;
    long date1 = t.data[2].date;
    double level0 = t.data[1].value;
    double level1 = t.data[2].value;

    // Calculate unrounded IR
    double uIR = level0 + ( 1.0 * ( date - date_start ) / ( date1 - date0 ) ) *
       ( level1 - level0 );

    // Return IR rounded to 5 dp
    return ( int ) ( uIR * 100000 ) / 100000.0;
}

////////////////////////////////////////////////////////////////////////////////
//
// Return due to inflation between dates 'start_date' and 'end_date'
// Supplementary_data should be set to 'index', where 'index' is the name of the
// CPI index associated with this security
//
////////////////////////////////////////////////////////////////////////////////

double Inflation ( char* risk_name, long start_date, long end_date,
FT_SECURITY_DATA s, FT_YIELD_CURVE c, ... )
{
    FT_TIME_SERIES t;
    va_list ap;

    va_start( ap, c ); // makes ap point to 1st unnamed arg
    t = va_arg( ap, FT_TIME_SERIES );

    double IR0 = CalculateIR ( start_date, t );
    double IR1 = CalculateIR ( end_date, t );

    return ( IR1 / IR0 ) - 1.0;
}
```

DATA INTEGRITY IN PLUG-INS

As plug-ins are functionally separate from the host application, it is possible for errors to occur within plug-ins that are not handled by the core application. Since they are essentially separate programs, FIA cannot 'know' what data inputs are expected. For this reason, we strongly recommend that plug-ins perform their own error checking on supplied inputs.

To make this straightforward, FIA makes extensive use of the NaN (Not A Number) value, as defined in the IEEE 754 standard. A NaN value can be stored in a 'double' field but is distinct from any numerical value. The 'isnan' function tests whether a given number is a NaN or not.

If a scalar-valued input (such as a coupon, or a strike date) has not been supplied in the user's data, its value inside the plug-in will always be NaN. For instance, if no maturity date was supplied in a bond's security master record, the value of 'maturity' in a pricing plug-in will be 'NaN'. This makes testing for missing values very straightforward. Supplementary and user-defined data fields are treated in the same way.

Similarly, if a yield curve has not been set up, its number of elements will always be zero and this can also be tested.

Note that not every field needs every input to be set up. For instance, a perpetual bond has no maturity date, so this field need not be provided in the security master data, and a NaN value for maturity may be ignored. Similarly, a bond that is priced using a market YTM does not require a yield curve for pricing.

When writing custom plug-in functions, we recommend the following checks:

Use the `CheckParameterDefined f`unction to check whether required scalar-valued inputs (any item in structure FT_SECURITY_DATA; any time series value) have been provided. If not, this function displays the name of the field and the name of the calling function in a pop-up box to indicate what data was missing.

Use the `CheckCurveDefined` function to check whether yield curve data is available. This function works in the same way as `CheckParameterDefined` but for yield curves.

If an error occurs within the plug-in, return `NaN` (Not A Number). FIA is designed to recognise this value and exit gracefully.

COMPILING AND LINKING PLUG-INS

The procedure for compiling a plug-in depends on the operating system you are using. The plug-ins directory contains sample source and header files, together with sample make files that show how to run the build process. A C or C++ compiler will be required to build plug-ins together with GNU 'make' if you prefer to use our make files.

To verify that you have everything in place, we recommend that you build the sample library from scratch, using our supplied make files, before you start writing your own plug-ins.

INSTALLING A PLUG-IN

Custom plug-ins must be set up and installed as follows:

- Plug-ins must always be placed in the library subdirectory for FIA, which is currently the same directory as the core engine. FIA does not require a name or path for any plug-in function.

- Each user-defined plug-in must have a unique name.
- Plug-ins can be placed in separate library files or grouped together, as long as all libraries have the correct suffix for your operating system.

VERIFYING PLUG-IN COMPATIBILITY

When FIA runs, it scans the list of plug-ins provided in the security master table and checks that there is one and only one function corresponding to the name of each plug-in available in the plug-in libraries.

- If a plug-in uses a function name that is not found in any library file in the plug-in directory, FIA will abort with an error message.
- If more than one match is found for a plug-in's name in the library files in the plug-in directory, FIA will also abort with an error message.