The Flametree Application Program Interface (API)

Author: Andrew Colin

Date: 30th May 2015

Revised: 27th June 2015

Status: Draft

Version: 1.02

Flametree API guide release 1.022
DISTRIBUTION STATUS: UNRESTRICTED

Contents

Introduction

The Flametree Application Program Interface (API) is a collection of function calls and related constants that allow the Flametree FIA attribution engine to be called from another program.

The Flametree attribution engine is written in C++. However, Flametree provide language wrappers that allow the core library to be called from a wide range of languages, including C++, C-Sharp, Python and Java.

Functions

The Flametree API contains the following functions:

```
int FIA_set_integer ( const int SYMBOLIC_CONSTANT, const int value );
int FIA_set_long ( const int SYMBOLIC_CONSTANT, const long value );
int FIA_set_double ( const int SYMBOLIC_CONSTANT, const double value );
int FIA_set_bool ( const int SYMBOLIC_CONSTANT, const bool value );
int FIA_set_string ( const int SYMBOLIC_CONSTANT, const string value );
int FIA_set_vector_string ( const int SYMBOLIC_CONSTANT, const
vector<string> value );
int FIA_set_matrix ( const int SYMBOLIC_CONSTANT, STRING_MATRIX value );
STRING_MATRIX FIA_get_matrix ( const int SYMBOLIC_CONSTANT);
FIA_run ( void );
```

Here, `SYMBOLIC_CONSTANT` identifies the quantity to be set, and `value` is its supplied value.

All parameters required to run the Flametree engine are passed via calls to functions with the prefix `'FT_set_'`, including the names of data arrays for input and output, switches that control the attribution analysis, values to specify the type and format of reports, and other information such as date format strings.

Parameters may be specified in any order, and `FT_run()` may be called multiple times in the same program. Since parameter settings are preserved between runs, this allows the user to run the engine multiple times in the same session. For instance, if a large number of portfolios are to be analysed using the same attribution settings, only the names of the data arrays need be changed between calls to `FT_run()`.

Results may be accessed via the contents of a `STRING_MATRIX` returned by `FIA_get_matrix`. Depending on the settings of various reporting switches, the FIA engine will also write CSV and XLS reports to disk.

It is typically not necessary to set every parameter, since FIA provides sensible defaults for all quantities. As a result, you may find it is only necessary to set a small number of parameters to achieve the results you require. Default values for each parameter are shown in the *Data Dictionary* tables later in this document.

Each call to an API function returns a status code. A value of ERR_OK = 0 indicates a successful call, and any other value indicates a failure.

This API is an example of a *data–driven* interface (also known as a *message-passing*, or *event-based* API). Instead of using a large collection of method calls, the API depends on a much smaller number of calls and *a data dictionary* of arguments. The advantages of using a data-driven API for the Flametree engine are that

- it is simple to understand and use;
- it is future-proof, in that planned additions to the engine's functionality in future releases will not break existing code that calls the library.

API constants

Data types

Data parameters used by the API are of the following standard C++ types:

- `integer`
- `double`
- `long`
- `boolean`
- `string`
- `vector<string>`
- `STRING_MATRIX`

where `STRING_MATRIX` is defined by

```
typedef vector <vector<string> > STRING_MATRIX;
```

Symbolic constants

A symbolic constant represents a system-defined integer that is used to refer to a particular library parameter. For instance, the symbolic constant that refers to the security master data array is `FT_SECURITY_DATA`, and the command to assign a data array called `security_data` to act as the security master data array is

```
FIA_parameter ( FT_SECURITY_DATA, security_data );
```

The naming conventions for symbolic constants are

- Each symbolic constant is a string delimited with an underscore character.
- Symbolic constants are always in capitals.
- All constants have the prefix 'FT'.
- The type of the constant is shown by the string immediately following the FT prefix. For instance, `FT_STRING_CARRY_DECOMPOSITION` refers to a string variable, while `FT_BOOL_ROLLDOWN_ATTRIBUTION` refers to a Boolean variable.

Since C++ allows overloading, the `FT_parameter` function can be used for all the above data types.

Required files

Any program that makes use of the Flametree API must include file `APIWrapper.h` via a command of the form

```
#include "APIWrapper.h"
```

This ensures that the API function prototypes are recognised and that the various symbolic constants used are resolved at link time. The values of all constants are generated by `enum` statements in the `APIWrapper.h` file.

The API also requires that you make the appropriate Flametree run-time library available to your program. The library typically has the name libfiaXX.dll (for Windows) and libfiaXX.so (for Linux), where XX is 32 or 64, depending on the user's requirements.

Sample usage

The following code fragment shows how to call the API from a C++ program.

```
{Include standard header files}
#include "APIWrapper.h"

{Set up STRING_MATRIX data arrays contains security, portfolio, benchmark and yield
curve data}
FIA_set_matrix ( FT_SECURITY_DATA, security_data );
FIA_set_matrix ( FT_PORTFOLIO_DATA, portfolio_data );
FIA_set_matrix ( FT_BENCHMARK_DATA, benchmark_data );
FIA_set_matrix ( FT_YIELDCURVE_DATA, yieldcurve_data );

{ Set up attribution analysis to calculate aggregated carry effects }
FIA_set_string ( FT_STRING_CARRY_DECOMPOSITION, "AGGREGATED" );

{ Decompose sovereign yield curve movements into shift, twist, curvature movements }
FIA_set_string ( FT_STRING_SOVEREIGN_CURVE_DECOMPOSITION, "STB" );

{ Only generate Excel output files }
FIA_set_bool ( FT_BOOL_CSV_REPORT, false );
FIA_set_bool ( FT_BOOL_XLS_REPORT, true );

{ Run the attribution analysis }
FIA_run();
```

A full set of demo files is available with your release installation.

Reporting and allocation options

Types of sectors

The sectors used by FIA fall into two separate groups.

The **first** sector type are those that are always available; these are PRICING, CURRENCY, CREDIT, CURVE, SUPPLEMENTARY_CURVE, MATURITY, DURATION, SECURITY.

- PRICING is the name of the pricing function, defined in column 5 of the security master file
- CURRENCY is the name of the security' currency, defined in column 6 of the security master file
- CREDIT is the security credit rating, either defined in column 11 of the security master file or assigned by default
- CURVE is the security's assigned risk-free yield curve, defined in column 8 of the security master file
- SUPPLEMENTARY_CURVE is the security's assigned supplementary credit curve, defined in column 9 of the security master file
- MATURITY is the maturity bucket into which the security falls at the report date. Maturity buckets are set up in the configuration file, depending on the buckets defined in FT_STRING_MATURITY_BUCKETS. Note that a security's maturity bucket may change over time as its maturity decreases. All maturity calculation and bucket assignments are handled automatically by FIA; there is no need to perform any maturity bucketing calculations.

- DURATION is the duration bucket into which the security falls at the report date. Duration buckets are set up in the configuration file, depending on the buckets defined in FT_STRING_DURATION_BUCKETS. Note that a security's duration bucket may change over time as its duration changes. All duration calculation and bucket assignments are handled automatically by FIA; there is no need to perform any duration bucketing calculations.
- SECURITY refers to individual securities. This variable is typically not required for asset allocation calculations. It is most often used at the end of a list of sectors in FT_STRING_REPORT_SECTORS, to indicate that a drill-down report's highest level of detail should be at the individual security level. If the SECURITY field is omitted, the lowest level of the drill-down report will correspond to the last sector given in the list.

The **second** sector type are those sector names defined in the Custom sectors column of the security master file. If a particular security has not had its sector type defined, a default value of Unassigned is used.

For instance, if you have defined a partition called COUNTRY, then this sector can also be used for reporting and for allocation calculations.

Lists of sectors

Some variables are used to hold a list of sectors in the form of a comma-delimited list.

The following variables hold a single comma-delimited list of sectors:

- FT_STRING_CARRY_ALLOCATION_SECTORS
- FT_STRING_CURVE_ALLOCATION_SECTORS
- FT_STRING_SPREAD_ALLOCATION_SECTORS
- FT_STRING_MIXED_ALLOCATION_SECTORS

These are all used to set up particular types of attribution analysis. For instance, FT_STRING_MIXED_ALLOCATION_SECTORS allows the user to specify the names and order in which asset allocation calculations should be performed for a Brinson analysis.

FT_STRING_REPORT_SECTORS is used for reporting instead of attribution calculation. This variable identifies a vector of string lists, where each vector of strings generates a separate drill-down report on the library's output.

Data dictionary

Run-time options

Symbolic constant	Description	Required?	Possible values	Example	Default
FT_INT_N_CORES	Number of cores to be used for processing	No	Any integer	8	Maximum number of cores available

Data array names

Constants in this section refer to arrays of data required by the API, rather than individual variables.

The structure and content of each array is also described in the related sections on the Flametree wiki.

All variables in this section have type STRING_MATRIX.

Symbolic constant	Description	Required?	Possible values	Example	Default
FT_MATRIX_PORTFOLIO	Name of portfolio weights and returns array	Yes	Any	N/A	N/A
FT_MATRIX_BENCHMARK	Name of benchmark weights and returns array	No	Any	N/A	N/A
FT_MATRIX_SECURITY	Name of security definitions array	Yes	Any	N/A	N/A
FT_MATRIX_YIELDCURVE	Name of yield curve array	Yes	Any	N/A	N/A
FT_MATRIX_MAP	Name of security map array containing security names that refer to the same security	No	Any	N/A	N/A
FT_MATRIX_STRESS	Name of stress test array, containing one or more stress test scenarios.	No	Any	N/A	N/A
FT_MATRIX_FX	Name of exchange rate array, containing currency codes, dates and rates.	No	Any	N/A	N/A
FT_MATRIX_INDEX	Name of file containing inflation indices, including file path if required, containing index names, dates and values.	No	Any	N/A	N/A

Notes

The benchmark file is optional. If no benchmark is specified, the program will run attribution on the supplied portfolio and generate an appropriate set of reports. The only exception is when the user has requested an asset allocation report. This type of analysis specifically requires a benchmark, so in this case FIA will generate an error message and halt.

Date format configuration

Symbolic constant	Description	Required?	Possible values	Example	Default
FT_STRING_DATE_FORMAT	User-supplied format for parsing dates from all files	No	See below	%d/%m/%Y	%Y-%m-%d
FT_STRING_SECURITY_DATE_FORMAT	User-supplied format for parsing dates in security file	No	See below	%d/%m/%Y	%Y-%m-%d
FT_STRING_PORTFOLIO_DATE_FORMAT	User-supplied format for parsing dates in portfolio returns files	No	See below	%d/%m/%Y	%Y-%m-%d
FT_STRING_BENCHMARK_DATE_FORMAT	User-supplied format for parsing dates in benchmark returns files	No	See below	%d/%m/%Y	%Y-%m-%d
FT_STRING_YIELDCURVE_DATE_FORMAT	User-supplied format for parsing dates in yield curve files	No	See below	%d/%m/%Y	%Y-%m-%d
FT_STRING_FX_DATE_FORMAT	User-supplied format for parsing dates in exchange rate files	No	See below	%d/%m/%Y	%Y-%m-%d
FT_STRING_INDEX_DATE_FORMAT	User-supplied format for parsing dates in inflation rate files	No	See below	%d/%m/%Y	%Y-%m-%d

Attribution calculation configuration

Symbolic constant	Description	Required ?	Possible values	Example	Default
FT_STRING_CARRY_DECOMPOSITION	String indicating type of carry return attribution required: • NONE: no return measured for the passage of time. • AGGREGATED: all return from carry assigned to one source. • PULL_TO_PAR: return decomposed into return from current yield and return from pull-to-par effects. • CREDIT_CARRY: carry return decomposed into return generated by risk-free yield, and return generated by spread above this yield. [Other values are reserved for future expansion]	No	NONE AGGREGATED PULL_TO_PAR CREDIT_CARRY	PULL_TO_PAR	AGGREGATED
FT_STRING_SOVEREIGN_CURVE_DECOMPOSITION	String indicating type of yield curve attribution required: • NONE: no return assigned to sovereign curve movements. The user may prefer not to display this field if no securities in the portfolio are affected by changes in the term structure.	No	NONE AGGREGATED DURATION STB KRD CCB PCA	DURATION	AGGREGATED

	• AGGREGATED: all return from sovereign curve movements assigned to one source • DURATION: return from sovereign curve assigned to parallel and non-parallel curve movements • STB: return from sovereign curve assigned to parallel shift, twist and curvature movements • KRD: return from sovereign curve assigned to movements at various maturities • CCB: twist motions measured by performing a least-squared fit of a first-order polynomial to the curve • PCA: models curve movements using principal component analysis [Other values are reserved for future expansion]				
FT_STRING_AVERAGE_CURVE_LEVEL	String indicating averaging routine to calculate mean level of yield curve. • ARITHMETIC: Simple arithmetic averaging of yield levels • TRAPEZOIDAL: Uses trapezoidal integration to remove effect of unequally spaced sample points on the yield curve • FIXED: The level is read from the level of the yield curve at the	No	ARITHMETIC TRAPEZOIDAL	ARITHMETIC	TRAPEZOIDAL

	maturity specified in FT_DOUBLE_FIXED_MATURITY [Other values are reserved for future expansion]				
FT_STRING_SPREAD_DECOMPOSITION	String indicating how credit or spread return should be calculated. NONE: No credit or spread returns are calculated. Any return generated by the security that has not been generated by sovereign yield curve effects is assigned to the `Residual` bucket. CREDIT: The program will calculate returns due to shifts between the various credit rating curves, if provided. Any remaining return is assigned to the `Residual` bucket. SECTOR: The program calculates the return due to shifts between the security's base curve and its sector curve. If no sector curve is provided, the sector spread return will always be zero. Any remaining return is assigned to the `Residual` bucket.	No	NONE CREDIT SECTOR	SECTOR	NONE
FT_DOUBLE_LOWER_TWIST_MATURITY	Lower maturity about which curve twists are measured. Only used if `SovereignCurveDecomposition=STB`.	No	>0	5	3
FT_DOUBLE_UPPER_TWIST_MATURITY	Upper maturity about which curve twists are measured. Only used if `SovereignCurveDecomposition=STB`.	No	>0	15	10
FT_STRING_CREDIT_RATING_LIST	List of credit ratings, indicating which credit curves in addition to the sovereign yield	No	See below	Aaa,Aa2,A2	AAA

	curve should be used for attribution. Only used if `SPREADDECOMPOSITION=CREDIT`				
FT_BOOL_INFLATION_ATTRIBUTION	Whether to display returns due to inflation If this field is not displayed, any returns from this source are assigned to `RESIDUAL`.	No	True, False	Yes	Yes
FT_BOOL_CASH_ATTRIBUTION	Whether to display returns due to interest on cash holdings If this field is not displayed, any returns from this source are assigned to `RESIDUAL`.	No	True, False	Yes	Yes
FT_BOOL_CONVEXITY_ATTRIBUTION	Whether to display returns due to security convexity If this field is not displayed, any returns from this source are assigned to `RESIDUAL`.	No	True, False	No	No
FT_BOOL_ROLLDOWN_ATTRIBUTION	Whether to display returns due to roll-down effects If this field is not displayed, any returns from this source are assigned to `RESIDUAL`.	No	True, False	No	No
FT_BOOL_PAYDOWN_ATTRIBUTION	Whether to display returns due to paydown If this field is not displayed, any returns from this source are assigned to `RESIDUAL`.	No	True, False	No	No
FT_STRING_MATURITY_LIST	List of maturities that define buckets used in reporting.	No	Any set of positive numbers. List does not need to be sorted. '0' is added automatically if not explicitly declared.	0,1,2,3,4,5,10,15	0,1,...,10,15,20,30
FT_STRING_DURATION_LIST	List of modified durations that define buckets used in reporting.	No	Any set of positive numbers. List does not need to be sorted. '0' is added automatically if not explicitly declared.	0,1,2,3,4,5,10,15	0,1,...,10,15,20,30

FT_STRING_KRD_LIST	List of key rate durations to be used if `SovereignCurveDecomposition=KRD`.	No	Any set of positive numbers. List does not need to be sorted. '0' is added automatically if not explicitly declared.	0,1,...,10,15,20, 30	0,1,...,10,15,20,30
FT_STRING_TENOR_LIST	List of tenor points on yield curve to be used if `CurveReport=true`.	No	Any set of positive numbers. List does not need to be sorted. '0' is added automatically if not explicitly declared.	0,1,...,10,15,20, 30	0.25,0.5,1,2,3,4,5,7, 10,15,20

Display options

Symbolic constant	Description	Required?	Possible values	Example	Default
FT_STRING_TIME_RETURN_LABEL	Label for time/coupon return. Only used when CouponDecomposition set to AGGREGATED	No	Any	Coupon	Coupon
FT_STRING_RUNNING_YIELD_LABEL	Label for return due to running yield. Only used when CouponDecomposition set to PULL_TO_PAR	No	Any	Running yield	Running yield
FT_STRING_PULL_TO_PAR_LABEL	Label for return due to pull to par effects. Only used when CouponDecomposition set to PULL_TO_PAR	No	Any	Pull to par	Pull to par
FT_STRING_RISK_FREE_CARRY_LABEL	Label for return due to risk free yield. Only used when CouponDecomposition set to CREDIT_CARRY	No	Any	Risk-free carry	Risk-free carry
FT_STRING_CREDIT_CARRY_LABEL	Label for return due to pull to par effects. Only used when CouponDecomposition set to CREDIT_CARRY	No	Any	Credit carry	Credit carry
FT_STRING_ROLLDOWN_RETURN_LABEL	Label for rolldown return	No	Any	Rolldown	Rolldown
FT_STRING_RESIDUAL_RETURN_LABEL	Label for residual return	No	Any	Residual	Residual
FT_STRING_SOVEREIGN_CURVE_RETURN_LABELS	Label for return due to changes in sovereign curve	No	Any	Sovereign curve	Sovereign curve
FT_STRING_DURATION_RETURN_LABEL	Label for return due to parallel changes in the sovereign curve. Only used when SovereignCurveDecomposition set to DURATION	No	Any	Duration	Duration
FT_STRING_NON_PARALLEL_CURVE_RETURN_LABEL	Label for return due to non-parallel changes in the sovereign curve. Only used when SovereignCurveDecomposition set to DURATION	No	Any	Non-parallel curve	Non-parallel curve
FT_STRING_SHIFT_RETURN_LABEL	Label for return due to parallel shift in sovereign curve. Only used when SovereignCurveDecomposition set to STB	No	Any	Shift	Shift
FT_STRING_TWIST_RETURN_LABEL	Label for return due to steepening or flattening in sovereign curve. Only used when SovereignCurveDecomposition set to STB	No	Any	Twist	Twist
FT_STRING_CURVATURE_RETURN_LABEL	Label for return due to increasing or decreasing curvature in sovereign curve. Only used when SovereignCurveDecomposition set to STB	No	Any	Curvature	Curvature

FT_STRING_SPREAD_RETURN_LABEL	Label for return due to movements in the country or sector spread	No	Any	Spread	Spread
FT_STRING_FX_RETURN_LABEL	Label for return due to changes in exchange rates	No	Any	FX return	FX return
FT_STRING_INFLATION_RETURN_LABEL	Label for return due to inflation	No	Any	Inflation return	Inflation return
FT_STRING_CASH_RETURN_LABEL	Label for return due to interest on cash accounts	No	Any	Cash deposits	Cash deposits
FT_STRING_UNATTRIBUTED_RETURN_LABEL	Label for return to due unattributed securities	No	Any	Unattributed	Unattributed
FT_STRING_TOTAL_RETURN_LABEL	Label for sum of all returns	No	Any	Total	Total

Mixed model allocation options

Symbolic constant	Description	Required?	Possible values	Example	Default
FT_STRING_MIXED_ALLOCATION_SECTORS	Sectors with which to perform allocation attribution on security-level returns. If this field is set to one or more valid sector names, returns will be decomposed using market value allocation. Allocation returns will be calculated using the supplied sector list, and a stock selection return will also be displayed. The mixed allocation model firstly calculates asset allocation returns for an entire portfolio, and then assigns any remaining return to a stock selection category. The mixed allocation model should be used for equity portfolios. It is also suitable for mixed portfolios that contain both equity and fixed income securities. If the portfolio has been managed from the top downwards in terms of security type and then country allocations, set `MixedAllocationSectors=pricing, country` (assuming a `country` variable has been defined). The program will then calculate the return made by the security type allocation decision, then the return made by the country allocation decision for each pricing type, and then a security selection return. If the configuration file has been set up to calculate fixed income returns, then these effects will be displayed in the system's report. If fixed income allocation decisions have been taken, then they will show up in the same report. For instance, if a configuration file requires both overall asset allocation returns and spread duration allocation returns, then values should be assigned	No	See below	See below	[Blank]

	to `MixedAllocationSectors` and `SpreadAllocationSectors`, and corresponding returns will be displayed on the attribution report. If equity attribution is to be run, the value of the `Residual` label should be changed to `Stock selection`. Equity securities have no interest rate exposures, so no carry return will be generated and all non-allocation return will be directed to this return category.				

Fixed income allocation attribution options

Symbolic constant	Description	Required?	Possible values	Example	Default
FT_STRING_CARRY_ALLOCATION_SECTORS	Sectors with which to perform allocation attribution on security-level carry returns. If this field is set to one or more valid sector names, carry returns will be decomposed using market value allocation. Allocation returns on carry will be calculated using the supplied sector list, and a stock selection return will also be displayed. If carry return is decomposed into multiple sources of sub-return (such as running yield and pull-to-par) then the overall allocation return for carry will be calculated, followed by individual stock selection returns, one for each source. For instance, if `CarryAllocationSectors` is set to `Credit, Duration` then the following four sources of carry return will be calculated: • Carry allocation return due to credit allocation decisions • Carry allocation return due to duration allocation decisions • Running yield return • Pull-to-par return If a portfolio does not generate any carry returns (for instance, if it only contains equities), no carry allocation or selection returns will be calculated.	No	See below	See below	[Blank]
FT_STRING_CURVE_ALLOCATION_SECTORS	Sectors with which to perform allocation attribution on security-level curve (non-carry) returns. Allocation returns will be calculated using the supplied sector list, and a stock selection return will also be displayed, decomposed into fixed income effects if required.	No	See below	See below	[Blank]

	For instance, if `CurveAllocationSectors` is set to `Credit`, `Country` and the fixed income effects are `Curve` and `Credit` then the following five sources of curve return will be calculated: • Market direction return • Duration allocation return due to credit duration allocation decisions • Duration allocation return due to country duration allocation decisions • Curve returns • Credit returns Note that curve and credit returns will be differ from the values calculated using a bottom-up approach, since a different attribution model is being used.				
FT_BOOL_RESIDUAL_AS_CREDIT	If there is significant credit return in the residual term, this switch labels return of type `residual` as having type `credit`, so it contributes to the curve allocation attribution calculation. This setting is only used if a value has been assigned to `CurveAllocationSectors`, so that spread duration allocation is being calculated. `ResidualCredit` should only be used for a portfolio where there is significant credit return, but no credit curve.	No	true,false	true	false

Smoothing options

Symbolic constant	Description	Required?	Possible values	Example	Default
FT_STRING_SMOOTHING_MODEL	String indicating type of smoothing algorithm required. Currently allowed values are • CARINO • GEOMETRIC [Other values are reserved for future expansion]	No	CARINO GEOMETRIC	CARINO	GEOMETRIC

Benchmark hedging options

Symbolic constant	Description	Required?	Possible values	Example	Default
FT_STRING_HEDGE_CURRENCY	Currency in which hedging is performed. `HedgeCurrency` must be a currency code of the same type supplied in the FX file. If `HedgeCurrency` is not set, no hedging is performed.	No	Any three-character ISO 4217 currency label	AUD	None
FT_DOUBLE_HEDGE_RATIO	Hedge ratio for hedging calculation. `HedgeRatio` must lie between 0 (no hedging) and 1 (100% hedging).	No	0.0 to 1.0	0.5	1.0

Futures cash offset options

Symbolic constant	Description	Required?	Possible values	Example	Default
FT_BOOL_USE_CASH_OFFSETS	Whether to use cash holdings in portfolio as offsets against futures holdings. If this switch is set to true, futures holdings are treated exactly like the underlying security, and a cash offset must be included in the portfolio's holdings to replicate the zero effective exposure of futures. If the switch is set to false (the default), the zero effective exposure of futures contracts is taken into account automatically when calculating performance. Many commercial attribution systems calculate and export cash offset holdings by default. Futures cash offset holdings must be set up as zero-interest, since they are an accounting convenience rather than a physical asset.	No	true, false	true	false

Reporting options

Symbolic constant	Description	Required?	Possible values	Example	Default
FT_BOOL_XLS_REPORT	Whether to generate Excel reports	No	true,false	true	true
FT_BOOL_CSV_REPORT	Whether to generate CSV reports	No	true,false	true	true
FT_BOOL_SUMMARY_RISK_REPORT	Whether to generate summary reports, showing returns from all risks aggregated over all dates and securities. Relative reports are generated if a benchmark is provided.	No	true,false	true	true
FT_BOOL_STRESS_REPORT	Whether to generate stress reports. Relative stress reports are generated if a benchmark is provided.	No	true,false	true	true
FT_BOOL_MATURITY_EXPOSURE_REPORT	Whether to generate maturity bucketed exposure reports. Relative exposure reports are generated if a benchmark is provided.	No	true,false	true	true
FT_BOOL_DURATION_EXPOSURE_REPORT	Whether to generate duration bucketed exposure reports. Relative exposure reports are generated if a benchmark is provided.	No	true,false	true	true
FT_BOOL_INTERACTIVE_RISK_REPORT	Whether to generate interactive risk reports (Excel only). Relative risk reports are generated if a benchmark is provided.	No	true,false	true	true
FT_BOOL_RAW_RETURNS_REPORT	Whether to generate reports showing unweighted returns	No	true,false	true	true
FT_BOOL_STATIC_RISK_REPORT	Whether to generate reports showing price, duration, yield, convexity	No	true,false	true	true
FT_BOOL_DATE_RISK_REPORT	Whether to generate reports showing portfolio returns and risks	No	true,false	true	true

	over time, both by time interval and cumulative				
FT_BOOL_SQL_DATA_REPORT	Whether to generate a CSV report showing weights and risk returns at the security level in a format suitable for importing into an SQL database	No	true,false	true	true
FT_BOOL_CURVE_REPORT	Whether to generate reports showing sovereign yield curves used in analysis broken down according to the requested decomposition.	No	true,false	true	true
FT_BOOL_TREE_MAP_REPORT	Whether to generate a CSV file containing data about the current portfolio suitable for generating a tree map report.	No	true,false	true	true
FT_VECTOR_STRING_REPORT_SECTORS	Names of sectors, in order, to be used when generating a drill-down reports. See the following section for more information on this option.	No	See below	See below	PRICING,CURRENCY,SECURITY
FT_BOOL_LOOK_THROUGH	For portfolios that contain one or more levels of subportfolios, whether to show the subportfolios in the same way as securities in the upper level portfolio, or to show the constituent securities only without any nesting structure	No	true,false	true	true
FT_BOOL_ROOT_LEVEL_ONLY	For portfolios that contain one or more levels of subportfolios, whether to generate reports only for the top level portfolio, or to generate stand-alone reports for all subportfolios.	No	true,false	true	true

Appendix A: Date format strings

FIA can parse a wide range of date formats. Date formats are constructed by appending a number of format specifiers, corresponding to the day, month and year fields of the supplied date, together with the field delimiters.

Formats correspond to those used by the C `strptime` function. The following formats are allowable:

Format	Example	Sample
%d	Day of month (1-31)	15
%m	Month number (1-12)	2
%b	Month in abbreviated or full form	February, Feb
%y	The year without century data (0-99). When a century is not otherwise specified, values in the range 69-99 refer to years in the twentieth century (1969-1999); values in the range 00-68 refer to years in the twenty-first century (2000-2068)	45
%Y	The year, including century	2015

Date format strings are constructed by aggregating the above format strings with the date delimiter. For instance, if your dates are in the `dd-mm-yy` format, set the format string to `%d-%m-%y`, using a hyphen as the delimiter.

The following table lists some example date format strings.

Date	Format
15/05/2013	%d/%m/%Y
15/05/13	%d/%m/%y
05/15/2013	%m/%d/%Y
05-15-2013	%m-%d-%Y
05152003	%m%d%Y
20031505	%Y%m%d
15-May-13	%d-%b-%y
15-May-2013	%d-%b-%Y

FIA allows you to specify a date format to be used for all files, a specific date format for each file, or any combination of the two. Formats are applied using the following schema:

- Any formats supplied in FT_STRING_SECURITY_DATE_FORMAT, FT_STRING_PORTFOLIO_DATE_FORMAT, FT_STRING_YIELD_CURVE_DATE_FORMAT, FT_STRING_FX_DATE_FORMAT are applied first to the corresponding files;
- Next, format FT_STRING_DATE_FORMAT is applied to any files that have not had a custom format applied;
- If FT_STRING_DATE_FORMAT has not been supplied, the default format is applied to any remaining files.

Example	Setting	Result
Example 1	No date format strings set	All dates from all files will be read using the default format string %Y-%m-%d
Example 2	FT_STRING_DATE_FORMAT is set to DEMO_STRING No other format strings set	All dates in all files will be read and parsed using DEMO_STRING
Example 3	FT_STRING_DATE_FORMAT is set to DEMO_STRING1, FT_STRING_SECURITY_DATE_FORMAT is set to DEMO_STRING2.	All dates in the security file will be read and parsed using DEMO_STRING2 All dates in other files will be read and parsed using DEMO_STRING1
Example 4	FT_STRING_DATE_FORMAT is not set FT_STRING_SECURITY_DATE_FORMAT is set to DEMO_STRING1	All dates in the security file will be read and parsed using DEMO_STRING1 All other dates will be read and parsed according to the default format string %Y-%m-%d

Appendix B: Symbolic constants

```
// File: APIWrapper.h
// Author: Andrew Colin
// Date: 29th May 2015

int FIA_parameter ( const int parameter_ID, const int value );
int FIA_parameter ( const int parameter_ID, const long value );
int FIA_parameter ( const int parameter_ID, const double value );
int FIA_parameter ( const int parameter_ID, const bool value );
int FIA_parameter ( const int parameter_ID, const string value );
int FIA_parameter ( const int parameter_ID, STRING_MATRIX& m );
int FIA_run ( void );

// Matrix-valued settings
enum {
    // Labels for data structures
    FT_MATRIX_PORTFOLIO,
    FT_MATRIX_BENCHMARK,
    FT_MATRIX_SECURITY,
    FT_MATRIX_YIELDCURVE,
    FT_MATRIX_STRESS,
    FT_MATRIX_MAP,
    FT_MATRIX_FX,
    FT_MATRIX_CPI,
    FT_MATRIX_RESULTS,
    FT_MATRIX_DIAGNOSTIC,

    // Switches relating to carry
    FT_STRING_CARRY_DECOMPOSITION,
    FT_STRING_CARRY_ALLOCATION_SECTORS,

    // Switches relating to sovereign curve decomposition
    FT_STRING_SOVEREIGN_CURVE_DECOMPOSITION,
    FT_STRING_AVERAGE_CURVE_LEVEL,
    FT_DOUBLE_LOWER_TWIST_MATURITY,
    FT_DOUBLE_UPPER_TWIST_MATURITY,
    FT_DOUBLE_SHIFT_MATURITY,
    FT_STRING_KRD,
    FT_STRING_CURVE_ALLOCATION_SECTORS,

    // Spread decomposition switches
    FT_STRING_SPREAD_DECOMPOSITION,
    FT_STRING_SPREAD_ALLOCATION_SECTORS,
    FT_STRING_RATINGS_LIST,

    // Minor FI returns
    FT_BOOL_ROLLDOWN_ATTRIBUTION,
    FT_BOOL_CONVEXITY_ATTRIBUTION,
    FT_BOOL_INFLATION_ATTRIBUTION,
    FT_BOOL_PAYDOWN_ATTRIBUTION,
    FT_BOOL_CASH_ATTRIBUTION,
    FT_BOOL_FX_ATTRIBUTION,

    // Attribution switches - mixed/hybrid
    FT_STRING_MIXED_ALLOCATION_SECTORS,

    // Report settings
    FT_BOOL_CSV_REPORT,
    FT_BOOL_XLS_REPORT,
    FT_BOOL_CSV_HEADER,
    FT_BOOL_RAW_REPORT,
```

```
    FT_BOOL_TREEMAP_REPORT,
    FT_BOOL_REPORT_FILE_NAMES,
    FT_BOOL_SORT_DESCENDING,
    FT_BOOL_ROOT_LEVEL_ONLY,
    FT_BOOL_LOOK_THROUGH,

    // Individual reports
    FT_BOOL_SUMMARY_RISK_REPORT,
    FT_BOOL_STRESS_REPORT,
    FT_BOOL_STATIC_RISK_REPORT,
    FT_BOOL_RAW_RETURNS_REPORT,
    FT_BOOL_INTERACTVE_RISK_REPORT,
    FT_BOOL_EXPOSURES_REPORT,
    FT_BOOL_DURATION_EXPOSURE_REPORT,
    FT_BOOL_MATURITY_EXPOSURE_REPORT,
    FT_BOOL_DATE_RISK_REPORT,
    FT_BOOL_EX_POST_RISK_REPORT,
    FT_BOOL_CURVE_REPORT,
    FT_BOOL_TRACKING_ERROR_REPORT,
    FT_BOOL_SQL_DATA_REPORT,

    // Labels
    FT_STRING_FX_RETURN_LABEL,
    FT_STRING_INTERACTION_LABEL,
    FT_STRING_TIME_RETURN_LABEL,
    FT_STRING_RUNNING_YIELD_LABEL,
    FT_STRING_PULL_TO_PAR_LABEL,
    FT_STRING_RISK_FREE_CARRY_LABEL,
    FT_STRING_CREDIT_CARRY_LABEL,
    FT_STRING_ROLLDOWN_RETURN_LABEL,
    FT_STRING_SPREAD_RETURN_LABEL,
    FT_STRING_RESIDUAL_RETURN_LABEL,
    FT_STRING_SOVEREIGN_CURVE_RETURN_LABEL,
    FT_STRING_DURATION_CURVE_RETURN_LABEL,
    FT_STRING_NON_PARALLEL_CURVE_RETURN_LABEL,
    FT_STRING_CONVEXITY_RETURN_LABEL,
    FT_STRING_TOTAL_RETURN_LABEL,
    FT_STRING_SHIFT_RETURN_LABEL,
    FT_STRING_TWIST_RETURN_LABEL,
    FT_STRING_CURVATURE_RETURN_LABEL,
    FT_STRING_INFLATION_RETURN_LABEL,
    FT_STRING_PAYDOWN_RETURN_LABEL,
    FT_STRING_CASH_RETURN_LABEL,
    FT_STRING_UNATTRIBUTED_RETURN_LABEL,
    FT_STRING_PRICE_RETURN_LABEL,

    // Report sector settings
    FT_STRING_MATURITY_BUCKETS,
    FT_STRING_DURATION_BUCKETS,
    FT_STRING_COUPON_BUCKETS,
    FT_STRING_TENOR,
    FT_VECTOR_STRING_REPORT_SECTORS,

    // General report settings
    FT_STRING_REPORT_DIRECTORY,
    FT_STRING_REPORT_FORMAT,

    // Date formats
    FT_STRING_DATE_FORMAT,
    FT_STRING_SECURITY_DATE_FORMAT,
    FT_STRING_PORTFOLIO_DATE_FORMAT,
    FT_STRING_BENCHMARK_DATE_FORMAT,
    FT_STRING_YIELDCURVE_DATE_FORMAT,
    FT_STRING_FX_DATE_FORMAT,
```

```
    FT_STRING_CPI_DATE_FORMAT,


    // Miscellaneous
    FT_STRING_START_DATE,
    FT_STRING_END_DATE,
    FT_STRING_BRINSON_MODEL,
    FT_STRING_SMOOTHING_MODEL,
    FT_BOOL_PRICE_RETURN,
    FT_BOOL_BASE_TO_LOCAL,
    FT_BOOL_USE_CASH_OFFSETS,
    FT_BOOL_RETURN_CODES,

    FT_BOOL_RESIDUAL_AS_CREDIT,
    FT_STRING_BASE_CURRENCY,
    FT_STRING_HEDGE_CURRENCY,
    FT_INT_N_CORES,
    FT_LONG_BATCH_ID
};
```

Appendix C: Calling the API from C++

The following code shows an example of calling the Flametree engine from a C++ program. The data is taken from the example 'Minimal portfolio and benchmark', supplied with FIA as a test portfolio.

The program should be compiled in 64-bit mode and linked with library file `libfia64.lib`. The libraries `libfia64.dll` and `liblxl.dll` should be in the same directory as the compiled calling program.

The following makefile fragment shows how to compile and link the demo program, using Visual C++:

```
all: fia64.exe

demo_cpp.obj: demo_cpp.cpp

      cl -c demo_cpp.cpp

# Stub program to call DLL

fia64.exe: demo_cpp.obj

      link demo_cpp.obj libfia64.lib /OUT:"fia64.exe"
```

```
// demo_cpp.cpp
// Simple demo code to show how to call FIA via the API in c++, using hard-coded data

#include <iostream>
#include <vector>

#include "APIWrapper.h"

using namespace std;

int main ( int ac, char * av[] )
{
    STRING_MATRIX security_data, portfolio_data, benchmark_data, yieldcurve_data,
results_data, diagnostic_data;

    // Load portfolio data
    const int P_ROWS = 6;
    const int P_COLS = 6;
    string p[][P_COLS] =
    {
        {"2-Aug-04", "fund1", "HUTC_65_151106", "0", "0", "0"},
        {"2-Aug-04", "fund1", "ABN_55_150908", "0", "0", "0"},
        {"2-Aug-04", "fund1", "IADB_575_150611", "0", "0", "0"},
        {"31-Aug-04", "fund1", "HUTC_65_151106", "0.32552", "0.01174", "0.01174"},
        {"31-Aug-04", "fund1", "ABN_55_150908", "0.30713", "0.01071", "0.01071"},
        {"31-Aug-04", "fund1", "IADB_575_150611", "0.36735", "0.01324", "0.01324"},
    };

    for ( int i = 0; i < P_ROWS; i++ )
    {
        vector<string> vec ( p[i], p[i] + P_COLS );
        portfolio_data.push_back( vec );
    }

    // Load security data
```

```
    const int S_COLS = 13;
    const int S_ROWS = 3;
    string s[][S_COLS] =
    {
        {"ABN_55_150908", "ABN_55_150908", "", "", "BOND", "AUD", "", "AUD_CURVE", "",
"15-Sep-08", "AAA", "0.055", "2"},
        {"IADB_575_150611", "IADB_575_150611", "", "", "BOND", "AUD", "", "AUD_CURVE",
"", "15-Jun-11", "AAA", "0.0575", "2"},
        {"HUTC_65_151106", "HUTC_65_151106", "", "", "BOND", "AUD", "", "AUD_CURVE",
"", "15-Nov-06", "AAA", "0.065", "2"}
    };

    for ( int i = 0; i < S_ROWS; i++ )
    {
        vector<string> vec ( s[i], s[i] + S_COLS );
        security_data.push_back( vec );
    }

    // Load curve data
    const int Y_COLS = 5;
    const int Y_ROWS = 24;
    string y[][Y_COLS] =
    {
        {"AUD_CURVE", "AAA", "0.25", "2-Aug-04", "0.052588"},
        {"AUD_CURVE", "AAA", "0.5", "2-Aug-04", "0.052916"},
        {"AUD_CURVE", "AAA", "1", "2-Aug-04", "0.052646"},
        {"AUD_CURVE", "AAA", "2", "2-Aug-04", "0.052841"},
        {"AUD_CURVE", "AAA", "3", "2-Aug-04", "0.054509"},
        {"AUD_CURVE", "AAA", "4", "2-Aug-04", "0.055645"},
        {"AUD_CURVE", "AAA", "5", "2-Aug-04", "0.056184"},
        {"AUD_CURVE", "AAA", "8", "2-Aug-04", "0.057278"},
        {"AUD_CURVE", "AAA", "9", "2-Aug-04", "0.057335"},
        {"AUD_CURVE", "AAA", "10", "2-Aug-04", "0.057646"},
        {"AUD_CURVE", "AAA", "15", "2-Aug-04", "0.059188"},
        {"AUD_CURVE", "AAA", "20", "2-Aug-04", "0.059328"},
        {"AUD_CURVE", "AAA", "0.25", "31-Aug-04", "0.052659"},
        {"AUD_CURVE", "AAA", "0.5", "31-Aug-04", "0.052798"},
        {"AUD_CURVE", "AAA", "1", "31-Aug-04", "0.052405"},
        {"AUD_CURVE", "AAA", "2", "31-Aug-04", "0.051921"},
        {"AUD_CURVE", "AAA", "4", "31-Aug-04", "0.053953"},
        {"AUD_CURVE", "AAA", "5", "31-Aug-04", "0.054423"},
        {"AUD_CURVE", "AAA", "7", "31-Aug-04", "0.055057"},
        {"AUD_CURVE", "AAA", "8", "31-Aug-04", "0.055477"},
        {"AUD_CURVE", "AAA", "9", "31-Aug-04", "0.05537"},
        {"AUD_CURVE", "AAA", "10", "31-Aug-04", "0.055694"},
        {"AUD_CURVE", "AAA", "15", "31-Aug-04", "0.057003"},
        {"AUD_CURVE", "AAA", "20", "31-Aug-04", "0.057143"}
    };

    for ( int i = 0; i < Y_ROWS; i++ )
    {
        vector<string> vec ( y[i], y[i] + Y_COLS );
        yieldcurve_data.push_back( vec );
    }

      // Load files
    FIA_set_matrix ( FT_MATRIX_SECURITY, security_data );
    FIA_set_matrix ( FT_MATRIX_PORTFOLIO, portfolio_data );
```

```
    FIA_set_matrix ( FT_MATRIX_BENCHMARK, benchmark_data );
    FIA_set_matrix ( FT_MATRIX_YIELDCURVE, yieldcurve_data );
    FIA_set_string ( FT_STRING_CARRY_DECOMPOSITION, "AGGREGATED" );
    FIA_set_string ( FT_STRING_SOVEREIGN_CURVE_DECOMPOSITION, "STB" );
    FIA_set_bool ( FT_BOOL_CONVEXITY_ATTRIBUTION, true );
    FIA_set_bool ( FT_BOOL_CSV_REPORT, false );
    FIA_set_bool ( FT_BOOL_XLS_REPORT, true );
    FIA_set_string ( FT_STRING_REPORT_DIRECTORY, ".\\demo" );
    FIA_set_string ( FT_STRING_DATE_FORMAT, "%d-%b-%y" );
    FIA_set_long ( FT_LONG_BATCH_ID, 999 );

    FIA_run();

    diagnostic_data = FIA_get_matrix ( FT_MATRIX_DIAGNOSTIC );

     for ( unsigned int i = 0; i < diagnostic_data.size(); i++ )
         cout << diagnostic_data[i][0] << endl;

       results_data = FIA_get_matrix ( FT_MATRIX_RESULTS );

     for ( unsigned int i = 0; i < results_data.size(); i++ )
       {
              for ( unsigned int j = 0; j < results_data[0].size(); j++ )
            cout << results_data[i][j] << ",";
              cout << endl;
       }

    return 0;
}
```

Appendix D: Calling the API from C#

The following code shows an example of calling the Flametree engine from a C# program. The data is taken from the example 'Minimal portfolio and benchmark', supplied with FIA as a test portfolio.

Assuming the program is called `demo_csharp.cs`, it should be compiled using the following command:

```
csc demo_csharp.cs wrapper.cs wrapperPINVOKE.cs StringVector.cs
StringMatrix.cs enum.cs
```

where files `wrapper.cs, wrapperPINVOKE, StringVector.cs, StringMatrix.cs, enum.cs` are supplied by Flametree. When the program is run, the library files `libfia64.dll and libxl.dll` should be present in the same directory.

```
// demo_csharp.cs
// Simple demo code to show how to call FIA via the API in c#, using hard-coded data
using System;
using FIA_CONSTS;

public class demo_csharp
{
    static void Main()
    {
        // Demo portfolio data
        const int P_ROWS = 6;
        const int P_COLS = 6;
        string[,] portfolio_data = new string[,]
        {
            {"2-Aug-04", "fund1", "HUTC_65_151106", "0", "0", "0"},
            {"2-Aug-04", "fund1", "ABN_55_150908", "0", "0", "0"},
            {"2-Aug-04", "fund1", "IADB_575_150611", "0", "0", "0"},
            {"31-Aug-04", "fund1", "HUTC_65_151106", "0.32552", "0.01174", "0.01174"},
            {"31-Aug-04", "fund1", "ABN_55_150908", "0.30713", "0.01071", "0.01071"},
            {"31-Aug-04", "fund1", "IADB_575_150611", "0.36735", "0.01324",
"0.01324"},
        };

        var p = new StringMatrix();
        for ( int i = 0; i < P_ROWS; i++ )
        {
            var v = new StringVector();
            for ( int j = 0; j < P_COLS; j++ )
                v.Add ( portfolio_data[i, j] );
            p.Add( v );
        }

        // Demo security data
        const int S_COLS = 13;
        const int S_ROWS = 3;
        string[,] security_data = new string[,]
        {
            {"ABN_55_150908", "ABN_55_150908", "", "", "BOND", "AUD", "", "AUD_CURVE",
"", "15-Sep-08", "AAA", "0.055", "2"},
            {"IADB_575_150611", "IADB_575_150611", "", "", "BOND", "AUD", "",
"AUD_CURVE", "", "15-Jun-11", "AAA", "0.0575", "2"},
```

```
            {"HUTC_65_151106", "HUTC_65_151106", "", "", "BOND", "AUD", "",
"AUD_CURVE", "", "15-Nov-06", "AAA", "0.065", "2"}
        };

        var s = new StringMatrix();
        for ( int i = 0; i < S_ROWS; i++ )
        {
            var v = new StringVector();
            for ( int j = 0; j < S_COLS; j++ )
                v.Add ( security_data[i, j] );
            s.Add( v );
        }

        // Load curve data
        const int Y_COLS = 5;
        const int Y_ROWS = 24;
        string[,] yield_data = new string[,]
        {
            {"AUD_CURVE", "AAA", "0.25", "2-Aug-04", "0.052588"},
            {"AUD_CURVE", "AAA", "0.5", "2-Aug-04", "0.052916"},
            {"AUD_CURVE", "AAA", "1", "2-Aug-04", "0.052646"},
            {"AUD_CURVE", "AAA", "2", "2-Aug-04", "0.052841"},
            {"AUD_CURVE", "AAA", "3", "2-Aug-04", "0.054509"},
            {"AUD_CURVE", "AAA", "4", "2-Aug-04", "0.055645"},
            {"AUD_CURVE", "AAA", "5", "2-Aug-04", "0.056184"},
            {"AUD_CURVE", "AAA", "8", "2-Aug-04", "0.057278"},
            {"AUD_CURVE", "AAA", "9", "2-Aug-04", "0.057335"},
            {"AUD_CURVE", "AAA", "10", "2-Aug-04", "0.057646"},
            {"AUD_CURVE", "AAA", "15", "2-Aug-04", "0.059188"},
            {"AUD_CURVE", "AAA", "20", "2-Aug-04", "0.059328"},
            {"AUD_CURVE", "AAA", "0.25", "31-Aug-04", "0.052659"},
            {"AUD_CURVE", "AAA", "0.5", "31-Aug-04", "0.052798"},
            {"AUD_CURVE", "AAA", "1", "31-Aug-04", "0.052405"},
            {"AUD_CURVE", "AAA", "2", "31-Aug-04", "0.051921"},
            {"AUD_CURVE", "AAA", "4", "31-Aug-04", "0.053953"},
            {"AUD_CURVE", "AAA", "5", "31-Aug-04", "0.054423"},
            {"AUD_CURVE", "AAA", "7", "31-Aug-04", "0.055057"},
            {"AUD_CURVE", "AAA", "8", "31-Aug-04", "0.055477"},
            {"AUD_CURVE", "AAA", "9", "31-Aug-04", "0.05537"},
            {"AUD_CURVE", "AAA", "10", "31-Aug-04", "0.055694"},
            {"AUD_CURVE", "AAA", "15", "31-Aug-04", "0.057003"},
            {"AUD_CURVE", "AAA", "20", "31-Aug-04", "0.057143"}
        };

        var y = new StringMatrix();
        for ( int i = 0; i < Y_ROWS; i++ )
        {
            var v = new StringVector();
            for ( int j = 0; j < Y_COLS; j++ )
                v.Add ( yield_data[i, j] );
            y.Add( v );
        }

        wrapper.FIA_set_matrix( FIA_CONST.FT_MATRIX_PORTFOLIO, p );
        wrapper.FIA_set_matrix( FIA_CONST.FT_MATRIX_SECURITY, s );
        wrapper.FIA_set_matrix( FIA_CONST.FT_MATRIX_YIELDCURVE, y );
        wrapper.FIA_set_string( FIA_CONST.FT_STRING_CARRY_DECOMPOSITION, "AGGREGATED"
);
```

```
        wrapper.FIA_set_string( FIA_CONST.FT_STRING_SOVEREIGN_CURVE_DECOMPOSITION,
"STB" );
        wrapper.FIA_set_bool( FIA_CONST.FT_BOOL_CONVEXITY_ATTRIBUTION, true );
        wrapper.FIA_set_bool( FIA_CONST.FT_BOOL_CSV_REPORT, false );
        wrapper.FIA_set_bool( FIA_CONST.FT_BOOL_XLS_REPORT, true );
        wrapper.FIA_set_string( FIA_CONST.FT_STRING_REPORT_DIRECTORY, ".\\demo" );
        wrapper.FIA_set_string( FIA_CONST.FT_STRING_DATE_FORMAT, "%d-%b-%y" );
        wrapper.FIA_set_long( FIA_CONST.FT_LONG_BATCH_ID, 999 );

        wrapper.FIA_run();

        StringMatrix results;
        results = wrapper.FIA_get_matrix( FIA_CONST.FT_MATRIX_RESULTS );

        Console.WriteLine( "DATE,PORTFOLIO,BENCHMARK,SECURITY,SOURCE,w,W,r,R" );

        for ( int i = 0; i < results.Count; i++ )
        {
            StringVector r = results[i];

            for ( int j = 0; j < r.Count; j++ )
            {
                Console.Write( r[j] );
                Console.Write( "," );
            }
            Console.WriteLine();
        }
    }
}
```

Appendix E: Calling the API from other languages

Flametree Technologies uses the SWIG (Simplified Wrapper and Interface Generator, http://www.swig.org/) package to generate language wrappers to allow the FIA engine to be called from other languages.

Please contact us if you have a requirement that is not covered elsewhere in this document.