# QL sample user guide

Sartorial Programming

26 October 2020

# Contents

# Chapter 1

# Introduction

This is a description of the entire service. As such it will appear at the start of the generated manual when we get to the point of generating manuals.

You should be describing the business functionality of the service.

Descriptions can be in multiple blocks for the same object being described, and they will be merged together in such cases. A new description will create a new paragraph.

# Chapter 2

# Data types

Each item of data in the library is of a particular data type. Arbitrarily complex data types can be built up of simpler types, with ultimately everything being broken down into the fundamental built-in types.

When describing the inputs to a function or data fields in a class we will use the term `attribute` indicating the data type and name of the input or class data field. Attributes in addition can be an array or a matrix. We will indicate a simple array by the notation `[]` after the data type. We will indicate a matrix by the notation `[,]` after the data type. Note that a matrix is a rectangular array where each row in the array has the same number of elements.

In this section we describe all the possible data types available in the library, starting from the built-in types and building up to the classes available in the library.

## 2.1 Built-in types

Built-in types are data types that are provided as standard for all services. These types are the building blocks of all data representations.

| **Typename** | **Description** |
|---|---|
| bool | True or false. |
| int | Positive or negative integer (32-bits). |
| double | Double precision real number (64-bits). |
| string | Standard string type. |
| Date | Date type. This will be translated into the appropriate type for each context. For Excel, this is the number of days since 1900. For Python, this is the `datetime.date` class. For .NET this is the `System.DateTime` class. |
| char | Single character. |
| unsigned int | Integer that will always be positive |

## 2.2 Enumerated types

Enumerated types are types which can take one of a finite number of possible values. At the C++ level, these are represented by an enum inside a class. The possible values hence are given the namespace of the class name which corresponds to the name defined in the documentation.

For the Excel interface, the values that can be used are strings. These will then be converted into the C++ enumerated type, and then used within the low-level library code.

In the documentation, we will give the name of the enumerated type, followed by a general description of how the type is used, and then followed by a list of the possible values (known as the enumerands).

### 2.2.1 BinaryOperator

Defines a binary operation.

**Possible Values:**

| Code | Strings | Description |
|---|---|---|
| `Add` | `Add` | |
| `Subtract` | `Subtract` | |
| `Multiply` | `Multiply` | |
| `Divide` | `Divide` | |
| `Power` | `Power` | |

## 2.3 Classes

Classes contain data represented by attributes and can have function represented by so-called class methods.

Attributes will have a name and data type, and contain data of the defined data type. Attributes can be scalars or arrays. The data type can be any previously defined data type, e.g another class, or a simple type, or an enumerated type, or one of the built-in primitive data types.

Classes can in general be serialised. This can be to a string format, or written to file in various formats, or transmitted via the network when using a client-server model.

In general, all of the attributes will be needed when we serialise the class, but not all of the attributes are needed for general information about the class. Hence some attributes are marked as private (only used in serialisation), and others as public (can be accessed directly from outside the class). At present we do not support changing the value of an attribute of class, but this may change in the future.

Classes can also have class methods. In the C++ interface these are C++ member functions. In the Excel interface class methods appear as functions where the name of the function starts with the class name followed by the method name. The first parameter of the function is the class instance (or object). In other interfaces to come later we will generally use the natural representation for each language supported. For example, Python and .NET both support classes, so we will have equivalent classes in these languages corresponding to the C++ class.

In the C++ interface we will use a form of shared pointer for representing the class. The normal constructors will be protected and you need to use the static `Make` method for each class in order to create the shared pointer instance for the class. By protecting the constructor in this manner, we make it impossible for client code to use raw pointers, and this should make it easier to avoid memory leaks in client C++ code.

### 2.3.1 Matrix

No description.

**Attributes:**

| Type and Name | Description |
|---|---|
| double data[,] | |

**Methods:** None

# Chapter 3

# Functions

This section describes the functions available in the library.

Note that when using Excel in conjunction with Objects (an Object is an instance of a class), we represent Objects via string handles. For functions which create objects, there will be one extra parameter to the function (or method). This parameter is called `baseName` and is needed to provide the first part of the string which forms the Object handle.

Note that when using the function you will need to use the overall namespace for the library as a prefix to the function name.

## 3.1 Add

Adds two numbers together.

This description will go into the header file and also into the tex file for this function. Each function gets its own tex file.

**Inputs:**

| Type and Name | Description |
|---|---|
| double x | First number to be summed. |
| double y | Second number to be summed. |

**ReturnType:** double

## 3.2 BinaryOp

Binary operation on two numbers.

**Inputs:**

| Type and Name | Description |
|---|---|
| double x | |
| BinaryOperator op | See BinaryOperator (page 4). |
| double y | |

**ReturnType:** double

## 3.3 DateToString

Tests that the QuantLib date format gives the correct date. We do this by using QuantLib's own date serialisation methods.

**Inputs:**

| Type and Name | Description |
|---|---|
| Date date | |

**ReturnType:** string

## 3.4 Determinant

No description.

**Inputs:**

| Type and Name | Description |
|---|---|
| double matrix[,] | |

**ReturnType:** double

## 3.5 MatrixMultiply

No description.

**Inputs:**

| Type and Name | Description |
|---|---|
| double m1[,] | |
| double m2[,] | |

**ReturnType:** double[,]

## 3.6 Timestamp

Shows the timestamp of the build. This is just for one file of the build, so for this to be accurate you will need to have done a full re-build!

**Inputs:** None

**ReturnType:** string

## 3.7 Version

Shows the version number. You can request details of the build as well.

**Inputs:**

| Type and Name | Description |
|---|---|
| bool showDetails | Optional (default = `True`). Show build details as well as just the version number |

**ReturnType:** string