Fairmat Extension Model Version 1.5.0 Quickstart

Contents

1 Introduction

This document shows how to use the extension points defined in the Fairmat core in order to write plug-ins. Fairmat plug-ins use the Mono.Addins extension model. For basic information on Mono.Addins see http://www.mono-project.com/Mono.Addins. By using Mono.Addins, new Fairmat's can be built by implementing one or more of the extension points defined by Fairmat that we have listed below. Plug-ins can use all the objects and classes defined in the Fairmat core: for details see the class documentation http://www.fairmat.com/software/DVPL.web/ .

2 Fairmat extension points

At the moment Fairmat can be extended on the points listed below. The following interfaces belongs to the name space DVPLI and are defined in assembly DVPLI.dll. In order to have more information and details about the interfaces, browse the classes documentation.

New extensions available in Fairmat 1.5.X:

- "/Fairmat/Shortcut" **interface** IShortcut

  Allows users to provide a shortcut definition separated from the main class providing a menu shown functionality.

New extensions available in Fairmat 1.3.X:

- "/Fairmat/ProjectTemplate" **interface** IProjectTemplate

  Allows users to define custom project templates

New extensions available in Fairmat 1.1.X:

- "/Fairmat/ProjectTemplate" **interface** IProjectTemplate

  Allows users to define custom project templates

- "/Fairmat/ContractMetric" **interface** IContractMetric

  Allows users to define custom objectives on the project: those metrics can be used by the Fairmat Optimizer (available in Fairmat Professional)

- "/Fairmat/RandomVariableOperation" **interface** IObjectOperation

  Populates the Random Variables contextual menu. It can be used to modifies properties or execute operations on random variables.

2.1 User Interface extension Points

- `"/Fairmat/ProcessTypeChoice" interface \ls{IEditableChoice}`

  Provides to the Fairmat user interface an option for creating a new stochastic process.

  An example of this interface can be found at https://github.com/fairmat/-InterestRatesModels/blob/master/HullAndWhiteOneFactor/HW1Choice.cs

- `"/Fairmat/SymbolChoice" interface IEditableChoice`

  Provides to the Fairmat user interface an choice for creating a user symbol (e.g. a constant).

  An example of this interface can be found at https://github.com/fairmat/-ModelingTools/blob/master/PFunction2D/PFunction2DSymbolChoice.cs

- `"/Fairmat/DataSourceChoice" interface IEditableChoice`

  Provides to the Fairmat user interface an option for creating a new data source.

- `"/Fairmat/ExternalTool" interface IExternalTool`

  Extends Fairmat's tools menu by describing a functionality available into the Fairmat workspace independently if a document is active or not.

- `"/Fairmat/DocumentToolUI" interface IDocumentToolUI`

  Extends Fairmat's tools menu by describing an operation performed on the entire document. It also handles the user interface and the operation itself.

- `"/Fairmat/CorrelationMatrixToolUI" interface IToolUI`

  Extends Fairmat's tools menu by describing an operation performed on the correlation matrix. It also handles the user interface and the operation itself.

- `"/Fairmat/SymbolToolUI" interface ItoolUI (optionally IProvider)`

  Defines the contextual menu for the objects in the symbol lists.

  An example of this interface can be found at https://github.com/fairmat/-ModelingTools/blob/master/DatesGenerator/DatesGeneratorContextItem.cs

- `"/Fairmat/OptionToolUI" interface IOptionToolUI`

  This interface extends Fairmat's options contextual menu by defining operation on a given option map element.

- `"/Fairmat/Editor" interface IEditor`

  Defines a User interface editor for a given object.

  An example of this interface can be found at https://github.com/fairmat/-ModelingTools/blob/master/DatesGenerator/DateSequenceForm.cs

- `"/Fairmat/Desktop/TopToolStrip" interface IToolStrip`

  Defines a tool-strip to be added in top-toolstrip panel (Desktop Version)

- `"/Fairmat/Desktop/OptionMapToolStrip" interface IToolStrip`

  Defines a tool-strip to be added in top-tool-strip panel (Desktop Version) and will be activated only when the option map is active.

- `"/Fairmat/Desktop/OptionMapDrawable" interface IOptionMapDrawable`

  Defines an object which contains information about how to visualize a logic option map's object when rendering the option map.

2.2 Input/output extension points

- `"/Fairmat/DataSourceReference" interface IDataSourceReference`

  Represents a reference to a generic data source: the data source can be connected to a given object of a Fairmat project. This object push data withing a Fairmat model.

- `"/Fairmat/NamedDataSourceReference" interface IDataSourceReference`

  Represents a reference to a generic data source: the data source can be connected to a given object of a Fairmat project. This object can be referred within a Fairmat expressions.

- `"/Fairmat/MarketDataProvider" interface IMarketDataProvider`

  Represents a reference to objects which are able to retrieve well defined structured data types (for example InterestRateMarketData).

2.3 Financial Modelling Extension Points

- `"/Fairmat/HistoricalInterestRates" interface IHistoricalInterestRates`

  Provides historical values of rates indices like Libor and Euribors.

- `"/Fairmat/ExtensibleProcess" interface IExtensibleProcess`

Describes a new stochastic process

An example of this interface can be found at https://github.com/fairmat/-EquityModels/blob/master/Heston/HestonProcess.cs

- `"/Fairmat/SimulatedRandomVariable" interface ISimulatedRandomVariable`

  Describes a random variable which realizations are simulated within the project.

- `"/Fairmat/RandomNumbersGenerator" interface IRandomNumbersGenerator`

  Implements an alternative random number generator.

  An example of this interface can be found at https://github.com/fairmat/-RandomNumberGenerators/blob/master/RandomSourcesSupport/RandomSourcesManager.cs

- `"/Fairmat/CustomVector" interface ICustomVector`

  Enables the definition of vector-like symbols which values are calculated from other inputs.

- `"/Fairmat/Analysis" interface IAnalysis`

  Performs an analysis made on a project. Built in pricing, sensitivities, greeks derivatives and other analysis implements that interface.

- `"/Fairmat/Estimator" interface IEstimator`

  Provides an estimation procedure for a given asset class.

  An example of this interface can be found at https://github.com/fairmat/-InterestRatesModels/blob/master/Pelsser/CapletEstimator.cs

- `"/Fairmat/AutomaticCalibration" interface IAutomaticCalibration`

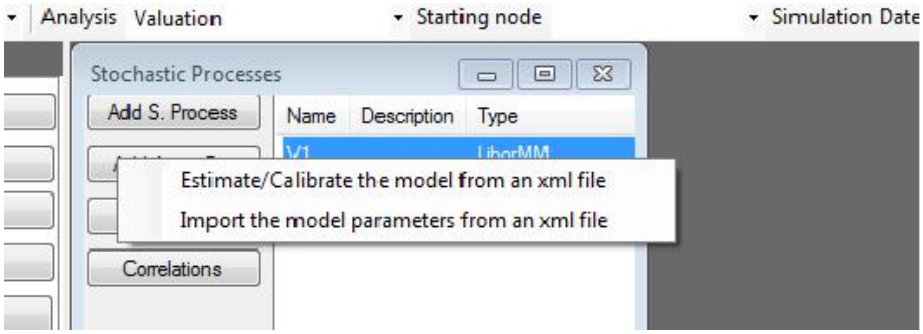
  Organizes the entire process of calibration, starting from market data recovery

- `"/Fairmat/DocumentTemplate" interface IDocumentTemplate`

  Generates a new Fairmat's document starting from a template.

- `"/Fairmat/StochasticProcessOperation" interface IObjectOperation`

  Populates the stochastic process contextual menu. It can be used to modifies properties or execute operations on stochastic processes. The available options appears as in the figure below.



- "/Fairmat/NumericalSolver" **interface** INumericalSolver

  Provides numerical a algorithm implementation. At the moment two numerical algorithms has been implemented (Monte Carlo Simulation and Binomial lattices)

- "/Fairmat/UIFunction" **interface** IUIFunction

  Defines a new scalar function available to the user interface. The function must return a scalar (double) value, and its arguments must be of the following two types, object or double

- "/Fairmat/UIConstant" **interface** IUIConstant

  Defines a new Reserved constant available to the Fairmat user interface.

- "/Fairmat/UserSettings" **interface** ISettings

  Allows plug-ins to define per user settings. Fairmat handles the editing/loading/saving for all the settings. A plug-in can retrieve a class which defines per-user settings by using the API defined in the class DVPLI.UserSettings. (For more details see the User Settings section later in this guide)

  An example of this interface can be found at https://github.com/fairmat/-RandomNumberGenerators/blob/master/RandomSourcesSupport/RandomSourcesSettings.cs

2.4 Misc Extension points

- "/Fairmat/Test" **interface** ITest

  Implements a test case.

- "/Fairmat/BigTest" **interface** ITest

  Implements a test case specifying that is resources (cpu/memory) consuming.

- "/Fairmat/ParametricTest" **interface** IParametricTest

Implements a test case which can be parametrized

- "/Fairmat/StartupOperation" **interface** ICommand

  Represents an operation that Fairmat will execute at startup.

  An example of this interface can be found at https://github.com/fairmat/-RandomNumberGenerators/blob/master/RandomSourcesSupport/RandomSourcesStartup.cs

- "/Fairmat/PerformanceStats" **interface** ...

  [Missing Description]

- "/Fairmat/PluginState" **interface** IPluginState

  Allows plug-ins to save their global variables.

- "/Fairmat/Objective" **interface** IObjective

  Describes an objective function for the goal seeker

3 OpenCL support

3.1 Interface IOpenCLCode

This is the basic interface plugins need to use in order to be able to take advantage of the OpenCL backend in Fairmat.

These shouldn't be considered stable and might change. Support in Fairmat for taking advantage of them will be available in future.

This interface is compromised of two properties: Arguments and Code.

```
public interface IOpenCLCode
{
  List<Tuple<string,object>> Arguments
  {
    get;
  }


  Dictionary<string,string>  Code
  {
    get;
  }

  bool IsOpenCLUsable
  {
    get;
  }
}
```

3.1.1 Arguments

The arguments property is a List of Tuple where the first element is the name give to an argument, while the second element is the actual Object. It can be of 3 types:

- double [] (double array)
- double
- ModelParameter (the parameter will be treated like a single double value)

If it's not one of these there could be runtime exception during the execution of the OpenCL simulator.

3.1.2 Code

The code is just a dictionary which changes its keys depending on the simulation to be done and the items are pieces of bare pieces of code implementing the requested code for the specific simulation. At the point of request it's assumed all variables are ready for use.

3.2 IsOpenCLUsable

This is used just to tell the simulator if the current instance of the plugin can handle OpenCL calculations. For example some settings might interfere with a proper use of the OpenCL backend, in these cases this is used to tell the simulator that the OpenCL implementation isn't usable. The value returned is a simple boolean, which is true if it's possible to use the instance of this plugin for the OpenCL simulation.

4 Stochastic processes

4.1 Definition and Simulation

Stochastic process are defined by a collections of interfaces and attributes.

4.1.1 Automatic Generation of required parameters

Starting from Fairmat 1.4.0, the plugin interface allows to define some parameters that will be created when a new instance of a stochastic process is added to the current model.

The objects referenced must be valid ModelParameters and also the attribute must only be applied to `IModelParameter` classes.

This is done through the use of a new attribute called `ExternalSymbolReference`.

```
[ExternalSymbolReference("Zrn",typeof(DVPLDOM.PFunction))]
```

The attribute has two arguments, in order of definition:

- A suggested name for the newly created parameters being added to the model.
- The type of object to create when crating the parameter.

Additionally there is a "third" argument which is taken from the `ModelParameter` definition which is its description. The description will be used to show to the user what is actually being created.

When the stochastic process is instanced the system will search for the `ExternalSymbolReference` attribute in the class and, for each, searches for suitable objects of the request type already defined in the project. If none are found, the user will be warned that a new one will be created, else the user will be requested to choose between creating a new one or using an already defined one.

When creating a new object the first parameter, containing a suggested name, is used by adding, after the provided string, a progressive number, which is increased when a new object with that name is created, to allow uniqueness inside the project.

If any `ExternalSymbolReference` is found in the class, the interface `IZeroRateReference`, which is now to be considered deprecated, will be ignored.

Example:

```
[ExternalSymbolReference("Zrn",typeof(DVPLDOM.PFunction))]
IModelParameter realZeroRate;

[ExternalSymbolReference("Zri",typeof(DVPLDOM.PFunction))]
IModelParameter secondaryZeroRate;
```

4.1.2 Markov Simulator

The most simple Markov simulators will just require to define in the code section the code to generate the α and the β component. The rest of the parameters, which aren't mentioned here, will be inferred from data already present in order to make the plugin with no OpenCL support work. The dictionary will contain, so, the α component at the key A, and the β component at the key B.

Both the code chunks can access these variables:

- x (at the current time position, so the x+1 at the previous step) as a pointer. So it's possible to access dimensions by accessing the component 0, 1, 2, etc.
- a or b as a pointer, depending if the code chunk is for A or B. It's possible to access dimensions by accessing the component 0, 1, 2, etc.
- step The current step in the simulation starting from 0.
- All the variables passed from the Arguments array as pointers, named like the first component of the Tuple.

4.1.3 IPostSimulationTransformation

In the case the plugin implements the IPostSimulationTransformation an additional component is required to be present in the dictionary in order to use OpenCL code: POSTRANSFORM. This Code is equivalent to the Transform() method which is part of the IPostSimulationTransformation interface

- x (at position of the current simulation start) as a pointer. So it's possible to access dimensions by accessing the component step*0, step*1, step*2, etc.
- dates The array of dates as pointer to doubles, each rappresents a step.
- stepN The total number of steps which have been done during the simulation (and also the amount of items in the dates array)
- All the variables passed from the Arguments array as pointers, named like the first component of the Tuple.

4.1.4 Markov Simulator Implementation Example

We will take Hull And White Community Edition to show an example of the code required to implement OpenCL support on top of a basic Markov simulation.

```
1  #region IOpenCLCode implementation
2  public List<Tuple<string, object>> Arguments
3  {
4    get {
5      List<Tuple<string, object>> arg =
6              new List<Tuple<string, object>>();
7      arg.Add(new Tuple<string, object>("alpha1", alpha1));
8      arg.Add(new Tuple<string, object>("sigma1", sigma1));
9      arg.Add(new Tuple<string, object>("SEMI_DRIFT", SEMI_DRIFT));
10     return arg;
11   }
12 }
13
14 public Dictionary<string,string> Code
15 {
16   get {
17     Dictionary<string, string> sources =
18               new Dictionary<string, string>();
19     sources.Add("B", "*b = sigma1;");
20     sources.Add("A", "*a = SEMI_DRIFT[step]-alpha1*x[0];");
21     return sources;
22   }
23 }
24
25 public bool IsOpenCLUsable
26 {
27   get
28   {
29     return true;
30   }
```

```
31 }
32 #endregion
```

4.2 Stochastic Process Calibration

Calibration procedure for stochastic processes are defined by the object implementing the `IEstimator` interface.

5 Case studies and examples

5.1 How to define new options blocks types using plug-ins

Even if Fairmat allows a lot of flexibility in defining the payoff, in some situations may be convenient to create new option map objects. In order to define a new option map leg the following objects must be created

- a class implementing the interfaces `IMarketObject`, `IDrawingProperties`. The class must implement the interfaces `IClosedFormValue` and `IClosedFormGreeks` if it implements closed form valuation, or `IExpressionPayoff`, or `ICodePayoff` if the class implement simulation
- a class implementing the interface `IOptionMapDrawable` and the Extension "/Fairmat/Desktop/OptionMapDrawable" in order to define the graphical aspect of the object.
- Choose a host (`OptionTreeExtensible` or `OptionTreeExtensibleOperator`)

5.2 How to define new stochastic process dynamics dynamics using plug-ins

Many process dynamic can be extended using plug-ins Fairmat enables two ways for defining new stochastic process dynamics:

1. By taking control of the entire simulation process using the interface `IFullSimulator`
2. If your process is a diffusion process, by implementing `IMarkovSimulator`.

In order to define a new stochastic process the following interfaces must be defined `IExtensibleProcess`, or IExtensibleProcessIR. The class must then define either `IMarkovSimulator` or `IFullSimulator`. The following interfaces are optional but are used by Fairmat if implemented `IParsable`, `IPostSimulationTransformation`.

5.2.1 Stochastic processes simulation workflow

Methods related to stochastic processes implementation are called in the following order:

1. Parses the object if the object implements `IParsable`.
2. The different queries are performed invoking `IExtensibleProcess` properties.
3. `IExtensibleProcess.Setup` is called
4. `IFullSimulator.Simulate` or `IMarkovSimulator.a/b` are invoker for every discrete time-step and given realization.
5. If defined `IPostSimulationTransformation.Transform` is called.

5.3 Application level variables

Data that must be shared at application level may be stored on the Engine.Globals dictionary. Defined keys are:

"MarketDataDBConnectionPath"

5.3.1 Examples

```
Engine.Globals.ContainsKey("MarketDataDBConnectionPath")
string path=Engine.Globals["MarketDataDBConnectionPath"] as string;
```

5.4 Extensions defaults

In many cases plug-ins allows users to extend the software capabilities, by, for example, defining new models for the underlying dynamics. In other contexts, plug-ins may offer new implementations for application-level tasks such random number generation or getting data for the preferred market data provider. Fairmat defines the following API for retrieving and registering a new default option for a given task.

- ```
  Mono.Addins.TypeExtensionNode
      DVPLI.ExensionsDefault.GetDefault(string extensionpath)
  ```

  Retrieves the default value for an extension path

- ```
  void DVPLI.ExensionsDefault.SetDefault(string extensionpath,
      Mono.Addins.ExtensionNode node)
  ```

  Register a news default for a given extension path. If the default has already been defined it will be replaced.

- ```
  Void DVPLI.ExensionsDefault.RegisterDefault(string
      extensionpath, Mono.Addins.ExtensionNode node)
  ```

  Register a new default without overwriting it in the case it has already been defined.

  In Fairmat 1.2 nodes which provides defaults are:
  ```

5.5 Handling Plug-ins settings

Fairmat allows plug-ins to define per-user-settings. Fairmat maintains those settings within different users sessions. Settings can stored using serializable classes. To use this feature, just create a class which implements the interface **DVPLI.ISettings**, and the class attribute

```
[Extension("/Fairmat/UserSettings")]
```

Then you can use the following attributes to decorate the class fields. Use the attribute **[SettingsContainer]** to specify the human readable name of the settings group defined by your UserSettings class and the attribute **SettingDescription** to specify that a given field can be edited by the user interface. For example:

```
[SettingDescription("Human readable name")] int fieldname;
```

SettingDescription works with every data type. Data input is enforced to be valid for the given type. In the case of enums, a list box will be displayed. If you need constraints to your data input, you can add specialized attributes to enforce the validation (at user interface level) of the settings.

- `[RangeSettingDescription]`

  Specifies the inclusive lower and upper bounds of an integer or real number. Example:

  ```
  [RangeSettingDescription("Number of iterations", 10, 100)]
      int iterations;
  ```

- `[PercentSettingDescription]`

  Displays a number or vector using percent

  Example:

  ```
  [RangeSettingDescription("List of confidence levels")] Vector
      confidenceLevels;
  ```

- `[ListSettingDescription]`

  Specifies that the elements of the field must be taken form a list. Example:

  ```
  [ListSettingDescription("Choice",new string[]{"A","B","C"})]
      string Choice;
  ```

  `[PathSettingDescription]`

  Enforces that a string must be a valid path in the running system. The user interface will also prompt a FileChoser dialog. Example:

  ```
  [PathSettingDescription("Exogenos TS realizations XML file")]
      string OutputFile;
  ```

- `[DirectorySettingDescription]`

  Enforces that a string must be a valid directory in the running system. The user interface will prompt a directory picker dialog. Example:

  ```
  [DirectorySettingDescription("Output directory")] string
      OutputDir;
  ```

  `[HideToUI]`

  Is a class level attribute and makes the class to behave like a setting (Fairmat will save it contents when opening and closing sections) but will be not visible to the user interface.

5.5.1 Settings usage and coding example

Below an example of a simple setting class

```
1  [SettingsContainer("Settings for My plugin")]
2  [Extension("/Fairmat/UserSettings")]
3  [Serializable]
4  public class MyUserSetting : ISettings
5  {
6      [SettingDescription("Parmater a")]//user readable name
7      public int a;
8      [SettingDescription("Paramerer b")]//user readable name
9      public double b;
10     [IsDirectory] [SettingDescription("output directory")]
11     public string output;
12
13 }
```

Fairmat will handle the manage user interface editing and saving of the settings. At the plug-in side use the following API to retrieve the settings.

```
MyUserSetting settings =
    DVPLI.UserSettings.GetSettings(typeof(MyUserSetting)) as
    MyUserSetting;
```

5.6 Case study: A simple extension

We describe step-by-step the operations needed to create a Fairmat plug-in with either MS Visual Studio or Monodevelop.

1. Create a new classlib project
2. Reference DVPLI.dll, DVPLDOM.dll, Mono.Addins.dll and Windows.Forms. You can find those references in the Fairmat directory (C:
   program files
   Fairmat Academic on windows or /usr/local/fairmat on linuxes).

   Remember to disable the option “copylocal” because those assemblies are already shipped with the software. In future versions they will be signed and will be available on the GAC.

3. Create the a classfile with the following code:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Windows.Forms;
4  using DVPLI;
5  using DVPLDOM;
6  using Mono.Addins;
7
8  [assembly: Addin("DocInfo", "1.0", Category = "Info")]
9  [assembly: AddinDependency("Fairmat", "1.0")]
10
11 namespace Info
12 {
13  [Extension("/Fairmat/DocumentToolUI")]
14   public class DocInfo : IDocumentToolUI
15       {
16             Document doc;
17           #region IDocumentToolUI Members
18
19           public IMyDocument Document
20           {
21               set
22                 {
23                     /* here save a reference to the document*/
24                     doc = (Document)value;
25                 }
26           }
27           #endregion
28
29           #region ICommand Members
30
31         public void Execute()
32         {
33             //get the reference to the first project
34             ProjectROV prj = (ProjectROV) doc.DefaultProject;
35             int options = prj.Map.Root.CountSubOptions();
36             int processes = prj.Processes.Count;
37             int symbols = prj.Symbols.Count;
38
39             MessageBox.Show(string.Format("Your project
                   contains {0} options,{1} processes and {2}
                   user defined symbols", options, processes,
                   symbols));
40         }
41
42         #endregion
43
44         #region IToolUIInfo Members
45
46         public string Category
47         {
48             get { return "Info"; }
49         }
50
51         public string Description
52         {
53             get {return "Document info";}
```

```
54          }
55
56          public string ToolTipText
57          {
58              get { return "Gives info about the project"; }
59          }
60
61          #endregion
62      }
```

4. Build the assembly and copy it in the Fairmat's Plugins subdirectory. Alternatively set the compiler to generate the output directly in the Plugins subdirectory.

5. Run Fairmat, open or create a new document and in the tool menu you will find a category voice "Info" with the "Document info" menu voice.

5.7 Additional resources

Several plugins are available in opensource form on github: https://github.com/fairmat, so they can be used as great examples for writing your own plugin. We have split plugins in several categories: Equity Models, Modeling Tools, Interest Rates Models and Random Number Generators.

Equity Models Contains several plugins implementing Equity Models for Fairmat. It includes Historical Simulator, Dupire and Heston.

https://github.com/fairmat/EquityModels

Modeling Tools Contains several tools to aid modeling in Fairmat. It includes PFunction2D and Dates Generator.

https://github.com/fairmat/ModelingTools

Interest Rates Models Contains several plugins implementing Interest Rates Models. It includes Hull and White One Factor, Hull And White Two Factors and the Pelsser's Squared Gaussian model.

https://github.com/fairmat/InterestRatesModels

Random Number Generators Contains a framework to handle random sources in an easy way, plus it includes an example using the qrng service in order to obtain random numbers.

https://github.com/fairmat/RandomNumberGenerators

6 Plug-ins writing tips

- All plug-ins standard output generated by Console.Write or Console.WriteLine calls, is redirected to the Fairmat log window.

- In order to control the amount of outputs a plug-in should write, you can condition the quantity of output to be written by reading at the (integer) value DVPLI.Engine.Verbosity. You can control the value of this variable from the user interface (Modeler preferences).