7 Objects

F3 facilitates accurate and efficient valuation of instruments, and provides tools to actively manage risk and revalue portfolios. Objects are the building blocks of F3 functionality. In this chapter, we will discuss the fundamental concepts and tools of F3 objects.

7.1 Objects: An overview

F3 objects are the building blocks that you use to build larger pieces of functionality to solve a problem. Some examples of F3 objects are:

- F3 functions, such as ValueProduct
- Holiday or day count convention, such as NewYorkNoRolling
- Curves, such as Discount Curve and Libor Rate Curve
- Financial products, such as interest rate swaps
- Prebuilt Models in F3, such as EmptyModel, or Models that you created, such as Curve_USD_Bootstrapped

An example of objects as building blocks is the function ValueProduct. In F3, valuation and risk calculations, no matter how complex, are performed with ValueProduct. As mentioned earlier, ValueProduct is an F3 function, and therefore it is an object. It takes 4 required inputs: Model, Product, ValuationMethod, and Requests. Furthermore, not only that each one of these inputs is an object, it is also constructed by other lower-level objects. Recall that in the Fundamentals Tutorial chapter, you have seen the construction of a roll schedule. In the concept of objects, this roll schedule is one of the required inputs to build a fixed leg. The fixed leg is, in turn, one of the inputs required to build a vanilla interest rate swap. The vanilla interest rate swap, which is a Product object, is then one of the inputs required for ValueProduct, which is used to value the swap. Hence in valuing a product, you start with lower level objects, and proceed to build larger pieces of functionality to solve a problem.

Some objects (for example, LiborUSD3m and act/365f) are built-in. They are present when F3 is launched.

However, you can also construct new objects in order to extend the capability of existing objects. Typically, when an object is constructed, you specify a name for the object. If you do not specify a name, or provide an empty string as the name, a name will be automatically generated based on the function that constructs the object and its inputs. In this case, the object's name is always the first argument of the function.

	If you specify an object name, it should be unique. Otherwise, if an object is constructed with the same name as an existing object, the new object will overwrite the existing object during calculation.

7.1.1 Creating New Objects

In the Fundamentals Tutorial Chapter, you have seen objects that are capable of creating other objects. They start with Create, Add, Form, or Extend. A few example in that Chapter are CreateSingleCashflowProduct, CreateEmptyModel, and AddSimpleDiscountCurveToModel.

Here is another example of how such object works. In this example, we will create a constant function of your choice.

1. Start F3 Excel Edition.
2. In an empty cell, type **=CreateConstantFunction()**. Create the object by using the **Build formatted function call as array** macro.
3. In the *FunctionName* field, enter "myConstantFunction".
4. In the *Parameter* field, enter **=pi()** as an approximation to π.
5. In the *ParameterTag* field, leave it blank in this case, as this is only needed when doing risk calculations.
6. Select the yellow solution cell and press F2. Then press ENTER. You have now created the object **myConstantFunction**.

7.2 Objects Are Stored in Repositories

All objects are stored and contained in repositories. A repository is a collection of objects of the same kind. For example, objects that provide day count conventions are contained in the DayCountConvention Repository.

Most repositories are populated with some default objects. Whenever a new object is created, it appears in the appropriate repository.

To list the names of all repositories, use the function ListRepositories.

To list the names of all objects within a given repository, use the function ListMembers.

To search all objects that match a certain search expression, use the function FindObject.

To perform a search of objects across multiple repositories, use the function RepositorySearch.

We will demonstrate how to use these functions in the following sections.

7.2.1 Using ListRepositories

1. Start F3 Excel Edition.
2. In an empty cell, type **=ListRepositories()**. You will see the name of the first repository (by alphabetical order). The result is a one-row array of which only the first result is showing.
3. To display the full array, highlight sufficient cells (horizontally) and press F2. Then press CTRL+SHIFT+ENTER.
4. For a better view, you can display the returned repositories in a column. You can do this by transposing the output of the function call. Type **=TRANSPOSE(ListRepositories())**. Highlight sufficient cells (vertically) and press F2. Then press CTRL+SHIFT+ENTER.

7.2.2 Using ListMembers

1. To see the contents of a particular repository, use the function ListMembers.

   In an empty cell, type **=ListMembers()**. Create the object by using the **Build formatted function call as array** macro.

2. In the *RepositoryName* field, enter the name of the repository in question. When you type in the name of a repository, make sure you capitalize the name in exactly the same way as they appear in the result obtained from ListRepositories. Alternatively, you can link the *RepositoryName* argument in ListMembers to a repository appearing in your list from steps 3 or 4 from the previous example Using ListRepositories. The result is a one-row array.
3. To display the full array, click in the cell containing the function. Highlight sufficient cells (horizontally). Press F2. Then press CTRL+SHIFT+ENTER.
4. For a better view, you can transpose the result in a column. To do this, add "transpose()" to the function call, such that the function call now shows **=transpose(ListMembers(...))**, where "..." denotes the cell reference of the *RepositoryName* input. Highlight sufficient cells (vertically) and press F2. Then press CTRL+SHIFT+ENTER.

> If "#N/A" is returned by ListMembers for any repository, this means that the repository is empty.

7.2.3 Using FindObject

Because F3 has a large number of built-in objects, sometimes you may need help searching and identifying the specific name of the object to use. The function FindObject allows you to put in a search expression, and it returns a set of search results that could be a list of repositories, repository members, or ObjectInfo requests. For details on ObjectInfo, see Finding Out What Is In An Object

1. In an empty cell, type **=FindObject()**. Create the object by using the **Build formatted function call as array** macro.
2. In the *SearchExpression* field, enter your search word(s).
3. This function returns the search results in a three-column array. Highlight multiple rows of the three-column array containing the function call and press F2. Then press CTRL + SHIFT + ENTER. Note that the exact number of rows you need depends on the search results.
4. If the search results are repositories, then the first column contains the repository names, while the second and the third columns are empty.
5. If the search results are repository members, then the first column contains the repository names, and the second column contain the repository member names, while the third column is empty.
6. If the search results are ObjectInfo *requests*, then the first column contains the repository names, the second column contains the repository member names, and the third column contains the names of the matched requests.
7. You have constructed the **myConstantFunction** earlier. Since object names in F3 are case-sensitive, you may be uncertain about how the object names are capitalized after it has been created. To find out the precise name of your object, call FindObject using the search expression "constant function". In the search results, you should see that **OneDimensionalFunction** and **myConstantFunction** are returned as the repository name and the object name, respectively.

7.2.4 Using RepositorySearch

Sometimes, you may need help searching and identifying an object where you are uncertain the repository to which it belongs. In this case, you can use the function RepositorySearch.

1. In an empty cell, type **=RepositorySearch()**. Then create the object by using the **Build formatted function call as array** macro.
2. In the *Repositories* field, enter an array of repository names to search.
3. In the *SearchExpression* field, enter your search word(s).

4. This function returns the search results in a two-column array. Highlight multiple rows of the two-column array containing the function call and press F2. Then press CTRL + SHIFT + ENTER. Note that the exact number of rows you need depends on the search results.
5. In the first column, you will find the repository name. If the search returns no results, then the second column is empty. If the search returns a match, then the second column contains the repository member names.

7.3 Finding Out What Is In An Object

- **ObjectInfo**

  To obtain certain information about an object, use the function ObjectInfo.

  ObjectInfo takes in 3 inputs:

  - *RepositoryName*
  - *ObjectName*
  - *Requests*

  Different objects have different available requests. So you can query different information on different objects. To see all available requests for an object, use "ListAll" as the *Requests* input.

  You can make object-specific requests for a given object. Additionally, here is a general list of requests applicable to all objects:

  - **LabelResults**: This request will produce an output that includes the name of the request as a label. You cannot use LabelResults alone. It must be combined with at least one other request.

    When LabelResults is used together with another request, and if the other request returns a null result, then ObjectInfo also returns a null result. This is because LabelResults adds a label to the result from every request. If some request does not produce any rows of output, there is nothing to add label to.

    > In Excel, a null result is represented by **#N/A**.

  - **ListAll**: This request lists all available requests for the object. The list of requests is returned as a one-row array in alphabetical order.
  - **ObjectIsUserConstructed**: This request returns either True or False. True means that the object was created by a function call (created by user). False means that the object is built-in.

- **ObjectConstructionWarnings**: This request performs some basic checks on the object to detect any potential mistakes. Usually, it returns a null response, which is represented as #N/A in Excel. This indicates that no potential mistakes were found.

  If you get a warning from running this request, you should know that this is not the same as an error. However, you should still re-check the inputs used in constructing your object.

- **ObjectConstructionCalls**: This request returns the function call that created the object in XML format, and is only available to user-constructed objects. If you run this request on a built-in object, you will get a null response #N/A.
- **ObjectHasDependencies**: This request returns either True or False. True means some other objects were used when creating this one. False means no other objects were used while creating this object.
- **ObjectDependencies**: This request returns the object(s) that were used in the creation of this object. The output is a two-column table of object names and repository names.
- **ObjectUniqueIdentifier**: This request returns a unique identifier for any object. Two objects with the same unique identifier are considered equivalent. Note that this is a sufficient, but not necessary, condition for two objects to be equal.

- **DisplayObject**

  You use the DisplayObject function to display the properties of an object.

  DisplayObject takes in 3 inputs:

  - *ObjectType*, Name of repository to which the object belongs
  - *ObjectName*
  - *Boolean*, True or False value for displaying default values of optional arguments (The default value is set to True.)

  DisplayObject displays some basic information about an object. For built-in objects, DisplayObject only returns the object name, object type, and the fact that it is a built-in object. The output of this function has the following format for built-in objects:

ObjectName
ObjectType
<BuiltIn>

  For user-created objects, whether you or another user created them, DisplayObject returns the object's properties, which consist of object name, object type, constructor name, and a collection of argument and value pairs. The output the function has the following format:

ObjectName

ObjectType	
Constructor	
PropertyName1	**PropertyValue1**
PropertyName2	**PropertyValue2**
...	**...**
PropertyNameN	**PropertyValueN**

7.3.1 Using ObjectInfo

To query for information about an object:

1. In an empty cell, type **=ObjectInfo()**. Create the object by using the **Build formatted function call as array** macro.
2. In the *RepositoryName* field, enter "Function".
3. In the *ObjectName* field, enter "MaturityDate".
4. In the *Requests* field, enter "ListAll".

   The *request* argument in the function call is a string that describes the type of information you want from the object. To view all the possible types that could be asked of an object, call ObjectInfo with the request "ListAll". The result will be a list of all the possible requests for that particular object.

5. Call the function ObjectInfo and press F2. The result is a one-row array of which only the first result is showing. To display the full array, highlight sufficient cells containing the function call (horizontally) and press F2. Then press CTRL + SHIFT + ENTER.
6. For a better view, you can transpose the result into a column. To do this, add "transpose()" to the function call, such that the function call now shows **=transpose(ObjectInfo(...))**, where "..." denotes the cell references of the "Function", "MaturityDate", and "ListAll" inputs. Highlight sufficient number of cells (vertically) and press F2. Then press CTRL + SHIFT + ENTER.
7. Next, to see how many arguments the MaturityDate object takes, call ObjectInfo with the request **NumberOfArguments**. You will get the answer **3**, since the function MaturityDate takes 3 arguments.
8. Now, try the request **IsObjectConstructor**. You will get a result of **FALSE**, which means this particular function does not create an object when it is called. It is a "stateless" function,

in that it does not modify the state of the library when it is called. Many functions are "stateful", which means they do modify the repository when called.

> When using ObjectInfo, you need to supply both the name of the object and repository in which it is stored. This is necessary because objects in different repositories are allowed to have the same name.

9. You have constructed the **myConstantFunction** earlier. This object was constructed using the function **CreateConstantFunction**. To find out under which repository an object constructed by this function is stored, call ObjectInfo on CreateConstantFunction, with the request **ConstructedObjectType**. You will see that it returned **OneDimensionalFunction**. This is the name of the repository.

7.3.2 Using DisplayObject

To display properties of an object:

1. To display the name, type and properties of an object, use the function DisplayObject.
2. To show the property of the object "MaturityDate", in an empty cell, type **=DisplayObject()**. Create the object by using **Build formatted function call as array** macro.
3. In the *ObjectType* field, enter "Function". This is the name of the repository under which the object is stored.
4. In the *ObjectName* field, enter "MaturityDate".
5. In the *ShowDefaults* field, leave the input as "True".
6. Call the function DisplayObject and press F2. This function returns a column array. To see the full array, highlight sufficient cells containing the function call (vertically) and press F2. Then press CTRL+SHIFT+ENTER.
7. To show the property of the object **myConstantFunction**, use **OneDimensionalFunction** as the *ObjectType* argument, use **myConstantFunction** as the *ObjectName* argument. The result is again a column array. To see the full array, highlight sufficient cells (vertically) and press F2. The press CTRL+SHIFT+ENTER.

7.3.3 Using Delegated ObjectInfo Request

You have seen that a *request* in ObjectInfo is an argument that describes the type of information you want from the object. Now we introduce another useful feature you can use to query properties of sub-objects that were used in constructing an object.

In F3, frequently, you use an object of one repository to create an object of another repository. The sub-object responds to a list of available requests. Furthermore, being in a different repository, the constructed object, containing the sub-object, also responds to its own available requests. Even though the sub-object is now a part of the constructed object, you can still make delegated ObjectInfo request on the constructed object to get certain information on the sub-object. This is done by adding a prefix to the name of the request that is specific to the sub-object. When you do this, F3 delegates this request to the sub-object, and returns the result to you as if the request was made directly to the sub-object.

Let us consider an example of LiborUSD3m in the Index Repository. An object in this repository respond to a list of requests (in addition to the standard requests you can make using ObjectInfo). Furthermore, you can make delegated requests, prefixed with either **MarketConventions**, **EquityEntity**, or **CreditContract**. The request "MarketConventionsRollTenor" will return 3m as a result. Note that MarketConventions is a prefix.

Let us consider another example in the FuturesContract Repository. First, construct an object testFuturesContract by calling the function CreateFuturesContract. Use "Standard" as the *ContractType* argument, and "LiborUSD3m" as *Underlying* argument. Then you can make delegated requests, prefixed with **FuturesContractType**. When you make the request of "FuturesContractTypeValuePerBasisPoint" with the function ObjectInfo, ObjectInfo passes this request to the sub-object (of FuturesContractType), and returns the result as if you were making a request on "Standard" object directly. Note that FuturesContractType is a prefix. Since Standard object in the FuturesContractType Repository has a value per basis point of 25, when you make the delegated ObjectInfo request on testFuturesContract object, you also get the result of 25.

If a function has a delegated request, then this delegated request will also appear in the list of available requests when using "ListAll" in ObjectInfo. To tell which request on this list is a delegated request, look for a prefix to the request name. The prefix will be the name of another repository.

Note that not all information of the sub-object can be queried using the delegated request. The following requests of the sub-object will only return the result <Unable to handle request>:

- LabelResults
- ListAll
- ObjectIsUserConstructed
- ObjectConstructionWarnings
- ObjectConstructionCalls
- ObjectHasDependencies
- ObjectDependencies
- ObjectUniqueIdentifier

7.4 Exporting and Importing Objects

Objects can be exported and imported. You can export and import objects in two ways: by exporting or importing specified objects themselves, or by exporting and importing the "state" of a workbook.

7.4.1 Exporting and Importing Objects

You can specify certain objects to be exported. You can do this by using the function ExportObjects. Once exported, the objects can be imported or re-created in another file by using the function ImportObjects or ExecuteFunctionFile.

- **Using ExportObjects function**

  F3 has a built-in function **ExportObjects** that allows you to export one or more objects to a file.

  ExportObject takes 4 arguments:

  - *CallTarget*, name of the output file
  - *RepositoryNames*, array of repository name(s), one for each object
  - *ObjectNames*, name of the specified object(s) to be exported
  - *IgnoreMissing*, a Boolean value indicating error suppression

  For every object specified, ExportObjects exports the function call of the specified object, together with the all other objects that are required in constructing the specified object. These objects can be found on the list returned by calling the function ListObjectDependents.

  The function ListObjectDependents can return a two-column list containing all the named objects used in the construction of the specified object. This list is formed recursively, so that if object A depends on object B, which in turns depend on object C, then both B and C will appear in this list.

  > Note that ListObjectDependents returns "all" objects required to build the specific object, regardless of the repositories under which the objects are stored. However, when ExportObject is called, it only exports the specified object, as well as all required objects in the same repository as the specified object.

  The ExportObjects function creates a function file in F3ML format. This function file can then be used together with the function ImportObjects or ExecuteFunctionFile to re-create the objects.

- **Using ImportObjects or ExecuteFunctionFile function**

F3 has a built-in functions **ImportObjects** and **ExecuteFunctionFile** that allows you to import objects saved in a function file created by ExportObjects.

ImportObjects and ExecuteFunctionFile take 2 arguments:

- *InputSource*, the path of the objects to be imported
- *OutputTarget*, optional argument

These functions behave similarly, in that they both take the same arguments, and return the following two-column array:

- The number of function calls made
- The number of calls returned successfully
- The number of calls suppressed
- The number of calls that failed to execute

> The following objects will be suppressed when called:
>
> - InvokeCallback
> - SetCallLoggingTarget
> - SetResultLoggingTarget
> - CreateTCPSocket
> - CreateZMQSocket
> - CreateZMQSocket
> - CreateOutboundZMQConnection
> - CreateHTTPServer
> - CreateNameServiceRegistration
> - CreateNameServer
> - CreateServerConnection
> - BrowseServer

To see the full array, highlight 4 rows of two-column array containing the function call and press F2.

If you can see the two-column array, and that there were no suppressed calls, then you have imported the objects successfully.

If an error message is displayed when you call the function, re-check the path to the function file.

After you have successfully called the ImportObjects or ExecuteFunctionFile function, then the objects would also have been added to the appropriate repository.

7.4.2 Exporting and Importing the F3 State

After you made changes in your workbook, you can export the "state" of this workbook so that it can be read by another workbook. Note that this process can be done repeatedly, in that once the second workbook has incorporated all the changes in the first workbook, you can "export" the state of the second workbook so that it can be read by a third workbook. This is achieved in F3 Excel Edition by using the **Export repository content** macro, which can be found in the Logging section of the F3 ribbon. This series of F3 function calls and their results, or "state", will then be saved to a file that can be read by other workbooks in F3.

To export an F3 state from one workbook and import it to another workbook, perform the following steps:

1. Start a new session of F3 Excel Edition. Open the source workbook.
2. Calculate the workbook by pressing F9.
3. Navigate to the **Tools** area of the F3 ribbon, click **Logging**, and select the **Export repository contents** macro. Save the function call log file in the same location where you saved the source workbook. Loading this file with **ImportObjects** is equivalent to running the source workbook.
4. Restart the F3 Excel Edition.
5. Open the destination workbook and save.
6. Calculate the destination workbook by pressing F9.
7. In an empty cell in the destination workbook, create the object **ImportObjects** by using the **Build formatted function call as array** macro.
8. In the *InputSource* field, enter the path to the saved file. ImportObjects returns a two-column array. Highlight 4 rows of two-column array containing the function call and press F2. If an error message is displayed, re-check the path to the call log file.
9. The destination workbook now contains the F3 objects created by the source workbook. You can export the state of this workbook by repeating this process. Once exported, the call log file of the second workbook will contain all F3 objects created by the first and second workbooks.

<< PREVIOUS CHAPTER CONTENTS