# AtomicDB API in *Mathematica*

By Athanassios I. Hatzis, PhD - (C) April 2015

# Test Add and Get Commands

## Test Preparation

### Load Application

```
ClearAll["Global`*"]
```

```
LibraryFunctionLoad[FindLibrary["APIPrimitiveOperationsDemo"], "AtomicDBAddOn", {}, "Void"][]
```

### Shortened Commands

#### Primitive Operations Commands

```
login = ADBloginToServer;
getAny =  ADBgetAnything;
addAny = ADBaddAnything;
```

```
? ADBloginToServer
```

```
ADBloginToServer[localhost,user,password,application]
Login Primitive Operation of AtomicDB API. Login to AtomicDB server. Returns the Models as C♯ GenericList of IAMCore Keys
```

```
? ADBaddAnything
```

```
ADBaddAnything[Model(s),Concept(s),Item(s),Options]
ADD Primitive Operation of AtomicDB API. Usually it returns the model(s), or
     concept(s), or item(s) that have been added as a C♯ GenericList of GenericList of IAMCore Key–Value Pairs.
```

```
? ADBgetAnything
```

```
ADBgetAnything[Model(s),Concept(s),Item(s),Options]
GET Primitive Operation of AtomicDB API. Usually it returns
     model(s), concept(s), or item(s) as a C♯ GenericList of GenericList of IAMCore Key–Value Pairs.
```

#### Output Commands

```
nout  = PrintOut;
ntext = PrintOutText;
```

```
? PrintOut
```

```
PrintOut[expr]
prints expr in a open Wolfram notebook that has been set for output (Out)
```

```
? PrintOutText
```

PrintOutText[expr,style_String]
writes text cells that appear in an opened Wolfram System notebook (Out) with the specified style

### Transformation Commands

```
toLists = ADBobjToLists;
toRules = ADBobjToRules;
toRec = WLrecordsToADBrecords;
```

```
? ADBobjToLists
```

ADBobjToLists[IAM_Core Key]
Transforms an IAMCore key to Mathematica List of four Integers

```
? ADBobjToRules
```

ADBobjToRules[IAM_Core Key–Value Pair]
Transforms an IAMCore key–value pair to Mathematica Rule

```
? WLrecordsToADBrecords
```

WLrecordsToADBrecords[List of Lists of Values]
Transforms Mathematica nested Lists of records to C♯ GenericList of array of string records

## Titles

```
ntext["Notice: This version of AtomicDB AddOn is for demonstration purposes
   only, not for commercial or other business use !", "Subsubsubsection"]
```

```
ntext["AtomicDB Add-On in Mathematica", "Title"]
ntext["AtomicDB API Primitive Operations Package Test (Demo Version)", "Subtitle"]
ntext["By Athanassios I. Hatzis" <> " - (C) " <> DateString[], "Subtitle"]
ntext["This output has been generated automagically. " <> "☺", "Subsubtitle"]
```

## Description of this Demo

```
nout["In this demo we build first  a simple relational data model using the Wolfram List
   structure.Our relational model example includes two main tables STOCK and ORDER that
   are joined with a third junction table STOCK-ORDER. Then we convert this to AtomicDB
   data model by adding a new Model, then Concepts (columns) and Records (rows)."]
```

# Relational Model

```
ntext["Relational Model", "Subchapter"];
```

### Headers of the Tables

Headers are lists of column names, i.e. attribute names.

```
stockHeader = {"StockID", "StockNameEN", "StockPrice", "StockNameGR"};
```

```
orderHeader = {"OrderID", "OrderKey"};
```

```
soHeader = {"SOID", "SOOrderID", "SOStockID", "SOQuantity"};
```

### Body of the Tables

The body of the table is the relation data set and it is represented with a list of records. Each record is represented with a list of values.

```
stockRelData = {{991, "Pinto Beans", 11.1`, "Φασόλια Πίντο"},
    {992, "Kidney Beans", 9.85`, "Φασόλια Κόκκινα"}, {993, "White Beans",
     13.45`, "Φασόλια Άσπρα"}, {994, "Wax Beans", 18.72`, "Φασόλια Καναρίνια"}};

orderRelData = {{441, "1111-BZ"}, {442, "1117-CM"}, {443, "1118-SA"}, {444, "1119-TT"}};

soRelData =
   {{224, 441, 991, 1}, {225, 442, 992, 3}, {226, 443, 994, 2}, {227, 444, 993, 1}, {228, 441, 993, 3}};
```

### Relation Sets

```
ntext["Relations", "Section"];

ntext["STOCK Table", "Subsection"];

(stockRelSet = Insert[stockRelData, stockHeader, 1]) // TableForm // nout

ntext["ORDER Table", "Subsection"];

(orderRelSet = Insert[orderRelData, orderHeader, 1]) // TableForm // nout

ntext["STOCK-ORDER Table", "Subsection"];

(soRelSet = Insert[soRelData, soHeader, 1]) // TableForm // nout
```

# AtomicDB Model

```
ntext["AtomicDB Model", "Subchapter"];
```

## Login To Server

```
ntext["Login To Server", "Section"]

ntext["Existing Models", "Subsection"];

modelKeys = login["localhost", "System Administrator", "Wind0ws7", "ManageIT"]
« NETObject[System.Collections.Generic.List`1[IAMCore_SharpClient.Core_Key]] »

modelKeys@ToArray[]
{}

toLists /@ modelKeys@ToArray[]
{}

(toLists /@ modelKeys@ToArray[]) // nout
```

## Add A Model

```
ntext["Concept Map System", "Section"]

modelName = "Beans Stock-Order Model Example";

ntext["Add A New Model", "Subsection"];

res1 = addAny[Null, Null, modelName,
   addType → enAddModel]
« NETObject[System.Collections.Generic.
    List`1[System.Collections.Generic.List`1[IAMCore_SharpClient.Core_KeyValuePair]]] »

newModel = res1[0][0]
« NETObject[IAMCore_SharpClient.Core_KeyValuePair] »

newModel // toRules // nout
```

## Get Command

```
ntext["Get All Models", "Subsection"];

(res2 = getAny[Null, Null, Null]) // nout

res2@ToArray[] // nout

res2[0][0] // nout

ntext["Print Key-Value Pair of the first model", "Subsubsection"]

(firstModel = res2[0][0]) // toRules // nout
```

## Add Concepts to the Model

```
ntext["Add Concepts to the Model", "Subsection"];

stockConceptsNames = Insert[stockHeader, "StockNEXUS", 1];
orderConceptsNames = Insert[orderHeader, "OrderNEXUS", 1];
soConceptsNames = {"SONEXUS", "SOID", "OrderID", "StockID", "SOQuantity"};
```

### Add STOCK Group Concepts

```
ntext["Add STOCK Group Concepts", "Subsubsection"]

stockConcepts = addAny[newModel, Null, MakeNETObject[stockConceptsNames]]

« NETObject[System.Collections.Generic.
    List`1[System.Collections.Generic.List`1[IAMCore_SharpClient.Core_KeyValuePair]]] »

toRules /@ (stockConcepts[0]@ToArray[]) // TableForm // nout
```

### Add ORDER Group Concepts

```
ntext["Add ORDER Group Concepts", "Subsubsection"]

orderConcepts = addAny[newModel, Null, MakeNETObject[orderConceptsNames]]

« NETObject[System.Collections.Generic.
    List`1[System.Collections.Generic.List`1[IAMCore_SharpClient.Core_KeyValuePair]]] »

toRules /@ (orderConcepts[0]@ToArray[]) // TableForm // nout
```

### Add STOCK-ORDER Group Concepts

```
ntext["Add STOCK-ORDER Group Concepts", "Subsubsection"]

soConcepts = addAny[newModel, Null, MakeNETObject[soConceptsNames]]

« NETObject[System.Collections.Generic.
    List`1[System.Collections.Generic.List`1[IAMCore_SharpClient.Core_KeyValuePair]]] »

toRules /@ (soConcepts[0]@ToArray[]) // TableForm // nout
```

## Add Collections Auto-generated from Concepts

```
ntext["Data Holder System", "Section"]

ntext["Add Collections", "Subsection"];
```

### STOCK Group Collections

```
ntext["Add STOCK Group Collections", "Subsubsection"]
```

```
stockCollections = addAny[newModel, stockConcepts, Null,
  doAutoMap → True, addType → enAddCollectionsAutoGroup]
```

```
« NETObject[System.Collections.Generic.
   List`1[System.Collections.Generic.List`1[IAMCore_SharpClient.Core_KeyValuePair]]] »
```

```
toRules /@ (stockCollections[0]@ToArray[]) // TableForm // nout
```

### ORDER Group Collections

```
ntext["Add ORDER Group Collections", "Subsubsection"]
```

```
orderCollections = addAny[newModel, orderConcepts, Null,
  doAutoMap → True, addType → enAddCollectionsAutoGroup]
```

```
« NETObject[System.Collections.Generic.
   List`1[System.Collections.Generic.List`1[IAMCore_SharpClient.Core_KeyValuePair]]] »
```

```
toRules /@ (orderCollections[0]@ToArray[]) // TableForm // nout
```

### STOCK-ORDER Group Collections

```
ntext["Add STOCK-ORDER Group Collections", "Subsubsection"]
```

```
soCollections = addAny[newModel, soConcepts, Null,
  doAutoMap → True, addType → enAddCollectionsAutoGroup]
```

```
« NETObject[System.Collections.Generic.
   List`1[System.Collections.Generic.List`1[IAMCore_SharpClient.Core_KeyValuePair]]] »
```

```
toRules /@ (soCollections[0]@ToArray[]) // TableForm // nout
```

## Add Records

```
ntext["Add Records", "Subsection"]
```

### Add Records to the STOCK Group

```
ntext["Add STOCK Group Records", "Subsubsection"]
```

```
recSet1 = toRec[stockRelData]
recSet1@ToArray[] // TableForm
```

```
« NETObject[System.Collections.Generic.List`1[System.String[]]] »
```

```
991    Pinto Beans     11.1     Φασόλια Πίντο
992    Kidney Beans    9.85     Φασόλια Κόκκινα
993    White Beans     13.45    Φασόλια Άσπρα
994    Wax Beans       18.72    Φασόλια Καναρίνια
```

```
stockRecords = addAny[newModel, stockConcepts, recSet1,
  verboseOutput → True]
```

```
« NETObject[System.Collections.Generic.
   List`1[System.Collections.Generic.List`1[IAMCore_SharpClient.Core_KeyValuePair]]] »
```

```
toRules /@ (stockRecords[0]@ToArray[]) // TableForm // nout
```

```
ReleaseNETObject[recSet1]
```

### Add Records to the ORDER Group

```
ntext["Add ORDER Group Records", "Subsubsection"]
```

```
recSet2 = toRec[orderRelData];
recSet2@ToArray[] // TableForm
```

```
441    1111-BZ
442    1117-CM
443    1118-SA
444    1119-TT
```

```
orderRecords = addAny[newModel, orderConcepts, recSet2,
  verboseOutput → True]
```

```
« NETObject[System.Collections.Generic.
    List`1[System.Collections.Generic.List`1[IAMCore_SharpClient.Core_KeyValuePair]]] »
```

```
toRules /@ (orderRecords[0]@ToArray[]) // TableForm // nout
```

```
ReleaseNETObject[recSet2]
```

## Add Records to the STOCK-ORDER Group

```
ntext["Add STOCK-ORDER Group Records", "Subsubsection"]
```

```
recSet3 = toRec[soRelData];
recSet3@ToArray[] // TableForm
```

```
224    441    991    1
225    442    992    3
226    443    994    2
227    444    993    1
228    441    993    3
```

```
soRecords = addAny[newModel, soConcepts, recSet3,
  verboseOutput → True]
```

```
« NETObject[System.Collections.Generic.
    List`1[System.Collections.Generic.List`1[IAMCore_SharpClient.Core_KeyValuePair]]] »
```

```
toRules /@ (orderRecords[0]@ToArray[]) // TableForm // nout
```

```
ReleaseNETObject[recSet3]
```