

**myObjectiveOLAP provider
for Microsoft Excel**

Technical Reference Guide
Moo Release 1.4 RC
MOO-0014

May 2010

myObjectiveOLAP

myObjectiveOLAP provider for Microsoft Excel Developer and Function Guide Moo Release 0.9.1

MOO-0014

Copyright © 2009, 2010, SDMC Consulting Limited and/or its affiliates. All rights reserved.

Primary Author: Robert Taylor

Contributing Authors:

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. SDMC Consulting Limited and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

SDMC Consulting is a registered trademark of SDMC Consulting and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. SDMC Consulting Limited and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. SDMC Consulting Limited and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Preface.....	4
Audience	4
Documentation Accessibility.....	4
Related Documents	4
Passwords in Code Examples.....	4
Conventions.....	5
Introducing myObjectiveOLAP Provider.....	6
myObjectiveOLAP Provider <i>Overview</i>	6
myObjectiveOLAP Files and Components	6
myObjectiveOLAPXL.dll Exposed functions.....	7
Common Functions.....	8
GUI Functions.....	40
Common Options.....	45
Excel Reporting Functions	55
Excel Advanced Reporting Functions.....	57
Trouble Shooting Guide	59

Preface

This document is your primary technical reference for use of the **myObjectiveOLAP for Microsoft Excel** application.

This Preface contains these topics:

- Audience
- Documentation Accessibility
- Related Documents
- Passwords in Code Examples
- Conventions

Audience

myObjectiveOLAP provider for Microsoft Excel Technical Reference Guide is intended for programmers who are developing applications to access an Oracle OLAP database using the myObjectiveOLAP provider. This documentation is also valuable to systems analysts, project managers, and others interested in the development of database applications.

To use this document, you must be familiar with Microsoft , Visual Basic for Applications (VBA) or a comparable object orientated language.

Users should also be familiar with the use of the Oracle OLAP Data Manipulation Language (DML) or Oracle Express Stored Procedure language (SPL) to access information in OLAP database systems.

Documentation Accessibility

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that SDMC does not own or control. SDMC neither evaluates nor makes any representations regarding the accessibility of these Web site

Related Documents

For more information, see these Oracle resources:

Passwords in Code Examples

For simplicity in demonstrating this product, code examples do not perform the password management techniques that a deployed system normally uses in a production environment.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
<code>monospace</code>	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Introducing myObjectiveOLAP Provider

This chapter introduces myObjectiveOLAP, an implementation of a framework provider to the Oracle OLAP engine.

This chapter contains these topics:

- ? [myObjectiveOLAP data access: Products and Documentation](#)
- ? [myObjectiveOLAP Overview](#)
- ? [myObjectiveOLAP Files and Components](#)
- ? [myObjectiveOLAP Exposed functions](#)

myObjectiveOLAP Provider Overview

myObjectiveOLAP is an implementation of a data provider for the Oracle OLAP database. Using and inheriting interfaces from myObjectiveOLAP and Oracle ODP .Net frameworks.

The myObjectiveOLAP framework allows native providers to expose Oracle OLAP specific features and data types. The myObjectiveOLAP framework provides an automation layer, with high performance and robust data type control between the Microsoft Windows client application and the Oracle OLAP database.

The myObjectiveOLAP provider for MS Excel uses Oracle native APIs to offer fast and reliable access to the Oracle OLAP engine within the Oracle Database. It exposes many of the Oracle OLAP data manipulation language commands and functions to the client.

The myObjectiveOLAP framework offers additional API's and graphical interfaces for working with both data and structures of the Escendo (OFA / OSA) replacement products.

myObjectiveOLAP Files and Components

- ? [myObjectiveOLAPXL.dll core myObjectiveOLAP framework library](#)

myObjectiveOLAPXL.dll is the core library that is used by any of the myObjectiveOLAP applications. myObjectiveOLAPXL.dll is not optional and must be installed in order to connect Microsoft Excel to the Oracle OLAP database using the myObjectiveOLAP data provider.

myObjectiveOLAPXL.dll contains pre-defined functions which are exposed to Microsoft Excel.

myObjectiveOLAPXL.dll Exposed functions

These functions are grouped into four sections

Common Functions

Common Functions enable the end user to interact either with the myObjectiveOLAP library itself or to execute commands or retrieve output from the Oracle OLAP server side application.

This includes a number of low level API's that do minimal checking before attempting to execute within the server side environment. It is best practice to only use these API's if myObjectiveOLAP does not offer a function to do this for you. By using the myObjectiveOLAP functions in your code additional pre-execution checks are performed and enhanced error trapping is available to you.

GUI Functions

A number of functions offered by the myObjectiveOLAP framework can offer the end-user a graphical interface into the Oracle OLAP option

Common Options

Common Options enable the end-user to control Oracle OLAP server side options.

Reporting Functions

Excel worksheet cell based functions which can be used by end users to develop rich reporting solutions

Instantiating the myObjectiveOLAP Object Reference

Getting an object reference to myObjectiveOLAP from VBA

In all of the examples shown, you will see a preceding "o." in front of the myObjectiveOLAP function.

This is the object reference to the myObjectiveOLAP library. You must also generate an object reference either by using this example or creating your own.

To instantiate the myObjectiveOLAP from Microsoft Visual Basic for Applications or Microsoft Visual Basic you must bind myObjectiveOLAP to an object that you can then reference.

In all of the examples shown we do this check by calling the regQ function which is shown below.

The regQ function binds the myObjectiveOLAPXL.AddinModule to the Global object "o". Once "o" has been bound you can use it to reference the myObjectiveOLAP functions i.e. o.connect o.mooAttached etc.

```
Global o As Object
Global oregistered As Boolean

Public Function regQ() As Boolean

If oregistered = False Then
    Set o = Application.COMAddIns.Item("myObjectiveOLAPXL.AddinModule").Object
    oregistered = True
Else
    regQ = True
End If

End Function
```

Common Functions

connect

Initiates a connection to an Oracle OLAP instance. Requires a valid XML connection file to have been created in advance.

Syntax

```
connect
```

Example

```
Public Sub connect()  
  'Actually does the connection  
  
  If Not oregistered Then:   boo = regQ:  
  boo = o.connect  
  
  If boo = True Then  
    MsgBox "Connected OK"  
  Else  
    MsgBox "Not Connected"  
  End If  
  
End Sub
```

disconnect

Closes the current connection to an Oracle OLAP instance. No further Analytic Workspace operations are carried out including DML detaching the analytic workspace.

Syntax

```
disconnect
```

Example

```
Public Sub disconnect()  
'Disconnects Excel From Oracle OLAP  
  
If Not oregistered Then:   boo = regQ:  
boo = o.disconnect  
  
If boo = True Then  
    MsgBox "Disconnected OK"  
Else  
    MsgBox "Not Disconnected"  
End If  
  
End Sub
```

connectSpec

Initiates a connection to an Oracle OLAP instance. Unlike connect a valid XML connection file is not required. However the developer must provide the necessary connection information during the function call.

Syntax

```
connectspec( "[host]" "[sid]" "[port]" "[user]" "[password]" )
```

Example

```
Public Sub connectSpec()  
' Create a manual connection without a connection xml file  
  
Dim hostname As String  
Dim sid As String  
Dim port As String  
Dim username As String  
Dim password As String  
  
If Not oregistered Then: boo = regQ:  
  
hostname = "yourHostName"  
sid = "yourSid-orcl"  
port = "yourPort-1521?"  
username = "yourUserName"  
password = "yourUserPassword"  
  
boo = o.connectSpec(hostname, sid, port, username, password)  
  
If boo = True Then  
MsgBox "Connected OK"  
Else  
MsgBox "Not Connected"  
End If  
  
End Sub
```

connected

Returns TRUE if a current connection is open to an Oracle OLAP instance.

Syntax

```
connected
```

Example

```
Public Sub connected()  
    'Am I connected  
  
    If Not oregistered Then:    boo = regQ:  
    boo = o.connected  
  
    If boo = True Then  
        MsgBox "Yes, Connected"  
    Else  
        MsgBox "No, Not Connected"  
    End If  
  
End Sub
```

mooAWAttach

Attaches an Analytic Workspace in the specified position if it exists

Syntax

```
mooAWAttach [aw name] [position]
```

where [aw name] is the fully referenced analytic workspace name
[postion] is the position the referenced aw should be attached.

FIRST makes the aw you are attaching the current one.

LAST makes the aw you are attaching the last in the list excluding the express aw.

BEFORE puts the aw you are attaching before an aw which is already in the list.

AFTER puts the aw you are attaching after an aw which is already in the list.

Example

```
Public Sub mooAwAttach()  
'Attach Analytic Workspaces  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
If boo = True Then  
    †boo = o.mooAwAttach("GLOBAL.GLOBAL", "FIRST")  
    †boo = o.mooAwAttach("GLOBAL.GLOBAL", "AFTER EXPRESS")  
Else  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmooerr  
End If  
End Sub
```

mooAWDetach

Detaches an analytic workspace. MooAWDetach does not perform an update.

Syntax

```
mooawdetach(" [aw name] ")
```

Example

```
Public Sub mooAwDetach()  
'Detach Analytic Workspaces  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
If boo = True Then  
    †boo = o.mooAWDetach("GLOBAL.GLOBAL ")  
Else  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmooerr  
End If  
End Sub
```

mooAWAttached

Returns TRUE if the specified analytic workspace is attached (open)

Syntax

```
mooawattached(" [aw name] ")
```

Example

```
Public Sub mooAwAttached()  
'Check if an Aw is attached  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
boo = o.mooAwAttached("EXPRESS")  
If boo = False Then  
    MsgBox "No Not Attached"  
Else  
    MsgBox "Yes Attached"  
End If  
  
End Sub
```

wrap_runNonQ

A low level API that allows a client developer to execute Oracle OLAP DML statements directly within the Oracle OLAP environment. wrap_runNonQ does not request any output from the Oracle OLAP environment on execution.

Syntax

```
wrap_runnonq(" [OLAP DML] ")
```

Example

```
Public Sub wrap_runNonQ()  
'Example of wrap_runNonQ and wrap_GetDML  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
If boo = True Then  
    boo = o.wrap_runNonQ("shw tod")  
    If boo = True Then  
        Debug.Print (o.wrap_getDML) 'You Could also use .mooGetDML here  
    Else  
        Debug.Print o.getlastmoerr  
    End If  
Else  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmoerr  
End If  
  
End Sub
```

wrap_GetDML

A low level API that retrieves the output from the Oracle OLAP environment after execution of a DML statement via the wrap_runNonQ function.

Syntax

wrap_getdml

Example

```
Public Sub wrap_runNonQ()  
'Example of wrap_runNonQ and wrap_GetDML  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
If boo = True Then  
    boo = o.wrap_runNonQ("shw tod")  
    If boo = True Then  
        Debug.Print (o.wrap_getDML)  
    Else  
        Debug.Print o.getLastMooErr  
    End If  
Else  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getLastMooErr  
End If  
  
End Sub
```

olapQDR

An end user orientated function that can be used to return a value from an Oracle OLAP array by fully qualifying the coordinates within the array.

Syntax

```
olapqdr(" [valid QDR statement] ")
```

where QDR statements in the the form

```
cube(dim1 dimval1 dim2 dimval2 ....dimx dimvalx)
```

the QDR must be fully referenced for the stated cube otherwise an error will be returned.

Example

```
Public Sub olapQDR()  
'Pass a QDR to Oracle OLAP and return the result  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so bring up the command line  
If boo = True Then  
    boo = o.mooAwAttach("EXPRESS", "FIRST")  
    'INTL.MLANGMAP is a variable within the Express AW  
    'You can pass any valid OLAP qdr to olapqdr  
    Debug.Print (o.olapQDR("INTL.MLANGMAP(INTL.MLANG 'ENB')"))  
  
End If  
  
End Sub
```

getLastMooErr

Returns the text of any error trapped by the myObjectiveOLAP library. This includes errors that were not handed over to the Oracle OLAP environment as the API determined the construct was invalid.

Syntax

```
getlastmooerror
```

Example

```
Public Sub mooClearErr()  
  'Clear Errors  
  
  If Not oregistered Then:   boo = regQ:  
  Debug.Print o.getlastmooerr  'Show the last Error  
  Debug.Print o.mooClearErr    'Clear all error messages  
  Debug.Print o.getlastmooerr  'show that all error messages have been cleared  
  
End Sub
```

mooEncrypt

Accepts a string and returns an encrypted version of the string that can be used in constructing a valid XML connection file.

The mooEncrypt function can be used to generate a password that can be copied and pasted into an XML connection file. Note there is no decrypt equivalent function as it is anticipated that all Oracle account passwords could be reset by the local DBA. Should you require a password to be decrypted the encrypted string should be emailed to support@myObjectiveOLAP.com together with the necessary authority.

Syntax

```
mooencrypt(" [password] ")
```

Example

```
Public Sub mooEncrypt()  
'Generate an encrypted password for a connection xml file  
'The Output in the VBA immediate window can be pasted in to a connection file  
If Not oRegistered Then: boo = regQ:  
Debug.Print o.mooEncrypt("myPasswordHere")  
  
End Sub
```

mooClearErr

Clears the last error trapped by the myObjectiveOLAP library. Subsequent calls to getLastMooErr would result in a NULL being returned until the next error.

Syntax

```
mooClearerr
```

Example

```
Public Sub mooClearErr()  
'Clear Errors  
  
If Not oregistered Then: boo = regQ:  
Debug.Print o.getLastmooerr 'Show the last Error  
Debug.Print o.mooClearErr 'Clear all error messages  
Debug.Print o.getLastmooerr 'show that all error messages have been cleared  
  
End Sub
```

mooservErr

Returns the text of any error trapped by the Oracle OLAP environment.

Syntax

```
mooserverr
```

Example

```
Public Sub mooservErr()  
'Get the last Oracle OLAP error  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    boo = o.wrap_runNonQ("shw GenerateAnError")  
    Debug.Print o.mooservErr()  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmoerr  
End If  
End Sub
```

mooDimLen

Returns the maximum length of an Oracle OLAP dimension as supplied to the function. The result is the same as executing an obj(dimmax 'DIMNAME') within the Oracle OLAP environment.

Syntax

```
moodimlen(" [dim1] ")
```

Example

```
Public Sub mooDimLen()  
  
'Show the Total Number of Dimension Values of a given dimension  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
If boo = True Then  
    Debug.Print o.moodimlen("INTL.MLANG")  
Else  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmooerr  
End If  
End Sub
```

moostatlen

Returns the current length of a specified dimension if the dimension exists within the current Oracle OLAP session.

Syntax

```
moostatlen(" [dim1] ")
```

Example

```
Public Sub mooPushMooPopExample()  
'Example Using mooPush mooPop and mooStatlen  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
    Else  
        boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    boo = o.mooAwAttach("EXPRESS", "FIRST")  
    'show the current length of the INTL.MLANG dimension  
    Debug.Print o.mooStatlen("INTL.MLANG")  
    'push the dimension  
    boo = o.mooPushDims("INTL.MLANG")  
    'limit INTL.MLANG dimension to 1 value  
    boo = o.wrap_runNonQ("lmt INTL.MLANG to 1")  
    'show the current length of the INTL.MLANG dimension  
    Debug.Print o.mooStatlen("INTL.MLANG")  
    'pop the dimension  
    boo = o.mooPopDims("INTL.MLANG")  
    Debug.Print o.mooStatlen("INTL.MLANG")  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmooerr  
End If  
End Sub
```

mooExists

Enables the developer to identify if a given object exists within an Analytic Workspace in the current Oracle OLAP session

Syntax

```
mooexists(" [object name] ")
```

Example

```
Public Sub mooExists()  
  
If Not oregistered Then:   boo = regQ:  
boo = o.mooExists("ALLCOMPILE")  
  
If boo = True Then  
    MsgBox "Yes, ALLCOMPILE Exists"  
Else  
    MsgBox "No, ALLCOMPILE Does Not Exist"  
    Debug.Print o.getLastMooErr  
End If  
  
End Sub
```

mooseconds

Returns the value of Seconds from the Oracle OLAP environment.

Syntax

```
mooseconds
```

Example

```
Public Sub mooseconds()  
'Return the current value of seconds within Oracle OLAP  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Get the value of seconds  
If boo = True Then  
    Debug.Print (o.mooseconds)  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmoerr  
End If  
  
End Sub
```

moopushDims

Executes a PUSH of an Oracle OLAP Dimension if the dimension exists.

Syntax

```
Moopushdims(“ [object name] “)
```

Example

```
Public Sub moopushMooPopExample()  
'Example Using moopush moopop and mooStatlen  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    boo = o.mooAwAttach("EXPRESS", "FIRST")  
    'show the current length of the INTL.MLANG dimension  
    Debug.Print o.mooStatlen("INTL.MLANG")  
    'push the dimension  
    boo = o.moopushDims("INTL.MLANG")  
    'limit INTL.MLANG dimension to 1 value  
    boo = o.wrap_runNonQ("lmt INTL.MLANG to 1")  
    'show the current length of the INTL.MLANG dimension  
    Debug.Print o.mooStatlen("INTL.MLANG")  
    'pop the dimension  
    boo = o.moopopDims("INTL.MLANG")  
    Debug.Print o.mooStatlen("INTL.MLANG")  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmooerr  
End If  
End Sub
```

mooPopDims

Executes a POP of an Oracle OLAP Dimension if the dimension exists.

Syntax

```
Moopopdims(" [object name] ")
```

Example

```
Public Sub mooPushMooPopExample()  
'Example Using mooPush mooPop and mooStatlen  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    boo = o.mooAwAttach("EXPRESS", "FIRST")  
    'show the current length of the INTL.MLANG dimension  
    Debug.Print o.mooStatlen("INTL.MLANG")  
    'push the dimension  
    boo = o.mooPushDims("INTL.MLANG")  
    'limit INTL.MLANG dimension to 1 value  
    boo = o.wrap_runNonQ("lmt INTL.MLANG to 1")  
    'show the current length of the INTL.MLANG dimension  
    Debug.Print o.mooStatlen("INTL.MLANG")  
    'pop the dimension  
    boo = o.mooPopDims("INTL.MLANG")  
    Debug.Print o.mooStatlen("INTL.MLANG")  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmooerr  
End If  
End Sub
```

mooObjType

Enables the developer to identify the data type of an Oracle OLAP object

Syntax

```
mooobtype(" [object name] ")
```

Example

```
Public Sub mooObjType()  
  
If Not oregistered Then:   boo = regQ:  
  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
If boo = True Then  
    Debug.Print o.mooObjType("ALLCOMPILE")  
    Debug.Print o.mooObjType("INTL.MLANGMAP")  
End If
```

mooOpenDimType

Limits a specified dimension within the Oracle OLAP environment to a status of ALL.

Syntax

```
mooopendimtype(" [dim1 ]")
```

Example

```
Public Sub mooOpenDim()  
'Limits a specified dimension to all  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    boo = o.mooAwAttach("EXPRESS", "FIRST")  
    boo = o.wrap_runNonQ("lmt INTL.MLANG to 1")  
    Debug.Print o.mooStatlen("INTL.MLANG")  
    boo = o.mooOpenDim("INTL.MLANG")  
    Debug.Print o.mooStatlen("INTL.MLANG")  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmooerr  
End If  
End Sub
```

mooAllStat

Opens the status of all dimensions within the currently attached Analytic Workspace.

Syntax

```
mooallstat
```

Example

```
Public Sub mooAllstat()  
'Limits all dimensions in the current AW to all  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    boo = †o.mooAwAttach("EXPRESS", "FIRST")  
    boo = o.wrap_runNonQ("lmt INTL.MLANG to 1")  
    Debug.Print o.mooStatlen("INTL.MLANG")  
    boo = o.mooAllstat  
    Debug.Print o.mooStatlen("INTL.MLANG")  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmooerr  
End If  
End Sub
```

moosetLang

Sets the language that any internal messages that are generated by the moo library or menu items used by the graphical user interface. Default is EN.

Syntax

```
moosetlang(" [EN|FR] ")
```

Example

```
Public Sub moosetLang()  
If Not oregistered Then: boo = regQ:  
  
'Switch the text in the graphical forms to french  
o.moosetLang ("FR")  
o.mooshowConnFrm  
  
'Switch back to English  
o.moosetLang ("EN")  
o.mooshowConnFrm  
  
End Sub
```

mooUser

Returns the name of the user currently connected to Oracle OLAP.

Syntax

```
mooouser
```

Example

```
Public Sub mooUser()  
'Shows the current database user  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    Debug.Print o.mooUser()  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmoerr  
End If  
End Sub
```

mooHost

Returns the hostname of the server which the Oracle OLAP environment is hosted.

Syntax

moohost

Example

```
Public Sub mooHost()  
'Shows the hostname of the server which you are connect to  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    Debug.Print o.mooHost()  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmooerr  
End If  
End Sub
```

mooInstance

Returns the instance name of the Oracle OLAP environment.

Syntax

```
mooinstance
```

Example

```
Public Sub mooInstance()  
'Shows the SID of the Oracle database you are connect to  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    Debug.Print o.mooInstance()  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmoerr  
End If  
End Sub
```

mooSysTimeStamp

Returns the value of SysTimeStamp within the Oracle OLAP environment.

Syntax

moosystimestamp

Example

```
Public Sub mooSysTimeStampANDmooSysDate()  
'Shows the value of SysTimeStamp and sysDate  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    Debug.Print o.mooSysTimeStamp()  
    'returns ie. 11-MAY-10 17.58.17.851963 +01:00  
    Debug.Print o.mooSysDate()  
    'returns ie. 11-MAY-10  
  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmoerr  
End If  
End Sub
```

moosysDate

Returns the value of SysDate within the Oracle OLAP environment.

Syntax

```
moosysdate
```

Example

```
Public Sub mooSysTimeStampANDmooSysDate()  
'Shows the value of SysTimeStamp and sysDate  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    Debug.Print o.mooSysTimeStamp()  
    'returns ie. 11-MAY-10 17.58.17.851963 +01:00  
    Debug.Print o.mooSysDate()  
    'returns ie. 11-MAY-10  
  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmooerr  
End If  
End Sub
```

mooAnalyzeCube

Returns a one-dimension array object each value of the array is a string value of the dimensions of the specified Analytic Workspace cube.

Syntax

```
mooanalyzecube(" [cube name] ")
```

Example

```
Public Sub mooAnalyzeCube()  
'returns a single dimension array of all dimensions of the variable passed to  
mooAnalyzeCube  
  
Dim boo As Boolean  
Dim arr() As String  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    boo = to.mooAwAttach("EXPRESS", "FIRST")  
    arr = o.mooAnalyzeCube("INTL.MLANGMAP")  
    For i = 0 To UBound(arr)  
        Debug.Print (arr(i))  
    Next  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmooerr  
End If  
End Sub
```

mooClearAnalyzeCube

Clears the internal myObjectiveOLAP array storing the result of any mooAnalyzeCube call.

Syntax

```
mooClearAnalyzeCube
```

Example

```
Public Sub mooClearAnalyzeCube()  
'Clears any cached results of the mooAnalyzeCube function  
  
Dim boo As Boolean  
Dim arr() As String  
  
If Not oRegistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    boo = o.mooClearAnalyzeCube()  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmooerr  
End If  
End Sub
```

mooFreePages

Reports the FREEPAGES of the current analytic workspace. If no analytic workspace is attached zero will be returned.

Syntax

```
mooFreePages
```

Example

```
Public Sub mooFreePages()  
'Shows the freepages of the current AW  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    boo = o.mooAwAttach("EXPRESS", "FIRST")  
    Debug.Print o.mooFreePages()  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmooerr  
End If  
End Sub
```

GUI Functions

`mooshowsplash`

Displays the myObjectiveOLAP splash screen

Syntax

```
mooshowsplash
```

Example

```
Public Sub mooshowsplash()  
    'show the myObjectiveOLAP splash screen  
  
    Dim boo As Boolean  
  
    If Not oregistered Then: boo = regQ:  
    boo = o.mooshowsplash  
End Sub
```

mooShowConnFrm

Displays the myObjectiveOLAP Connection Editor Screen, this is used in creating a valid connection XML file for use with either the GUI connect or the connect function.

Syntax

```
mooShowConnFrm
```

Example

```
Public Sub mooCommandBar()  
    'show the myObjectiveOLAP command bar  
  
    Dim boo As Boolean  
  
    If Not oregistered Then:    boo = regQ:  
    boo = o.CommandBar  
End Sub
```

[mooCmd_line](#)

Displays a command line interface that can be used to interact directly with the Oracle OLAP environment in a similar way to Oracle OLAP Worksheet or Oracle OX products.

The command line interface also offers access to MooScript which enables the developer to interact both with internal myObjectiveOLAP structures and Oracle OLAP structures through an automation layer.

Syntax

```
mooCmd_line
```

Example

```
Public Sub moo_CmdLine()  
'Bring up the OLAP Console  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so bring up the command line  
If boo = True Then  
    boo = o.mooCmd_Line  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmoerr  
End If  
  
End Sub
```

ShowAvailAW

Displays a Selector style graphical attach tool that can be used to attach or detach available analytic workspaces.

Syntax

```
showavailaw
```

Example

```
Public Sub showAvailAW()  
'Display the Analytic Workspace Selector  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so bring up the Analytic Workspace selector  
If boo = True Then  
    boo = o.showAvailAW  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmoerr  
End If  
End Sub
```

[mooCommandBar](#)

Displays a floating menu containing icons used to connect to or disconnect from a host, attach an analytic workspace via the AW Manager, open the OLAP Console and the help system.

Syntax

```
moocommandbar
```

Example

```
Public Sub mooCommandBar()  
    'show the myObjectiveOLAP command bar  
  
    Dim boo As Boolean  
  
    If Not oregistered Then:    boo = regQ:  
    boo = o.CommandBar  
End Sub
```

Common Options

The following functions can be used to set Oracle OLAP Server side options. They are protected functions which use the myObjectiveOLAP sense-checking algorithm before being executed within the Oracle OLAP environment. Any errors are reported via: [getLastMooErr](#)

Documentation on the use of these options can be found in the “Oracle OLAP DML Reference Guide”

[mooSetNASpell](#)

Sets the Oracle OLAP NASPELL option

Syntax

```
mooSetNASpell(" [text]'NA' ")
```

Example

```
Public Sub mooSetNASpell()  
  
'Sets the NASpell Option in Oracle OLAP  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    boo = o.mooSetNASpell("0")  
    boo = o.wrap_runNonQ("show na"): Debug.Print (o.mooGetDML)  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmooerr  
End If  
End Sub
```

moosetNASkip

Sets the Oracle OLAP NASKIP option

Syntax

```
moosetnaskip(" [yes|no] ")
```

Example

```
Public Sub moosetNASkip()  
    'Sets the NASpell Option in Oracle OLAP  
  
    Dim boo As Boolean  
  
    If Not oregistered Then:    boo = regQ:  
  
    'Check Im connected if not connect  
    If Not o.connected Then  
        boo = o.connect  
        boo = o.connected  
    Else  
        boo = True  
    End If  
  
    'Connected so execute  
    If boo = True Then  
        boo = o.moosetNASkip("YES")  
        boo = o.wrap_runNonQ("show NASKIP"): Debug.Print (o.moosetDML)  
    Else  
        'Something went wrong print any error information  
        Debug.Print "Unable to Connect"  
        Debug.Print o.getlastmoerr  
    End If  
End Sub
```

moosetNASkip2

Sets the Oracle OLAP NASKIP2 option

Syntax

```
moosetnaskip2(" [yes|no] ")
```

Example

```
Public Sub moosetNASkip2()  
    'Sets the NASpell Option in Oracle OLAP  
  
    Dim boo As Boolean  
  
    If Not oregistered Then:    boo = regQ:  
  
    'Check Im connected if not connect  
    If Not o.connected Then  
        boo = o.connect  
        boo = o.connected  
    Else  
        boo = True  
    End If  
  
    'Connected so execute  
    If boo = True Then  
        boo = o.moosetNASkip2("NO")  
        boo = o.wrap_runNonQ("show NASKIP2"): Debug.Print (o.moosetDML)  
    Else  
        'Something went wrong print any error information  
        Debug.Print "Unable to Connect"  
        Debug.Print o.getlastmoerr  
    End If  
End Sub
```

moosetBadLine

Sets the Oracle OLAP BADLINE option

Syntax

```
moosetbadline(" [yes|no] ")
```

Example

```
Public Sub moosetBadLine()  
    'Sets the Badline Option in Oracle OLAP  
  
    Dim boo As Boolean  
  
    If Not oregistered Then:    boo = regQ:  
  
    'Check Im connected if not connect  
    If Not o.connected Then  
        boo = o.connect  
        boo = o.connected  
    Else  
        boo = True  
    End If  
  
    'Connected so execute  
    If boo = True Then  
        boo = o.moosetBadLine("YES")  
        boo = o.wrap_runNonQ("show BadLine"): Debug.Print (o.moosetBadLine)  
    Else  
        'Something went wrong print any error information  
        Debug.Print "Unable to Connect"  
        Debug.Print o.getlastmooserr  
    End If  
End Sub
```

moosetCommas

Sets the Oracle OLAP COMMAS option

Syntax

```
Moosetcommas(" [yes|no] ")
```

Example

```
Public Sub moosetCommas()  
  
'Sets the COMMAS Option in Oracle OLAP  
  
Dim boo As Boolean  
  
If Not oregistered Then: boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    boo = o.moosetCommas("NO") ' YES or NO  
    boo = o.wrap_runNonQ("show Commas"): Debug.Print (o.moosetDML)  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmoerr  
End If  
End Sub
```

moosetAWWAITTIME

Sets the Oracle OLAP AWWAITIME option

Syntax

```
moosetawwaittime(" [integer value] ")
```

Where the [integer value] is the number of seconds required. If zero is entered the value will be set to the default of 20.

Example

```
Public Sub mooSetAWWaitTime()  
  
'Sets the AWWaitTime Option in Oracle OLAP  
'If 0 is passed then the default 20 is applied  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    boo = o.mooSetAWWaitTime("0") ' seconds passed here  
    boo = o.wrap_runNonQ("show AWWaitTime"): Debug.Print (o.mooGetDML)  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmoerr  
End If  
End Sub
```

moosetDateFormat

Sets the Oracle OLAP DATEFORMAT option

Syntax

```
moosetdateformat(" [valid date format] ")
```

Example

```
Public Sub moosetDateFormat()  
    'Sets the DateFormat Option in Oracle OLAP  
  
    Dim boo As Boolean  
  
    If Not oregistered Then:    boo = regQ:  
  
    'Check Im connected if not connect  
    If Not o.connected Then  
        boo = o.connect  
        boo = o.connected  
    Else  
        boo = True  
    End If  
  
    'Connected so execute  
    If boo = True Then  
        ' boo = o.moosetDateFormat("<DD><MTXT><YY>") ' Format should be  
<DD><MTXT><YY>  
        boo = o.wrap_runNonQ("show DateFormat"): Debug.Print (o.moosetDML)  
    Else  
        'Something went wrong print any error information  
        Debug.Print "Unable to Connect"  
        Debug.Print o.getlastmoerr  
    End If  
End Sub
```

moosetDecimals

Sets the Oracle OLAP DECIMALS option.

Syntax

```
moosetdecimals(" [integer value] ")
```

where the [integer value] represents the number of decimals places required

Example

```
Public Sub moosetDecimals()  
  
'Sets the DateFormat Option in Oracle OLAP  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    boo = o.moosetDecimals("0")  
    boo = o.wrap_runNonQ("show DECIMALS"): Debug.Print (o.moosetDML)  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmoerr  
End If  
End Sub
```

moosetLikeCase

Sets the Oracle OLAP LIKECASE option.

Syntax

```
moosetlikecase(" [yes|no] ")
```

Example

```
Public Sub moosetLikeCase()  
'Sets the DateFormat Option in Oracle OLAP  
  
Dim boo As Boolean  
  
If Not oregistered Then: boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    boo = o.moosetLikeCase("NO")  
    boo = o.wrap_runNonQ("show LIKECASE"): Debug.Print (o.moosetDML)  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmoerr  
End If  
End Sub
```

moosetParens

Sets the Oracle OLAP PARENS option.

Syntax

```
moosetparens(" {yes|no} ")
```

Example

```
Public Sub moosetParens()  
'Sets the Parens Option in Oracle OLAP  
  
Dim boo As Boolean  
  
If Not oregistered Then:   boo = regQ:  
  
'Check Im connected if not connect  
If Not o.connected Then  
    boo = o.connect  
    boo = o.connected  
Else  
    boo = True  
End If  
  
'Connected so execute  
If boo = True Then  
    boo = o.moosetParens("no")  
    boo = o.wrap_runNonQ("show PARENS"): Debug.Print (o.moosetParens)  
Else  
    'Something went wrong print any error information  
    Debug.Print "Unable to Connect"  
    Debug.Print o.getlastmoerr  
End If  
End Sub
```

Excel Reporting Functions

Excel worksheet cell based functions which can be used by end users to develop rich reporting solutions.

mooDesc

Returns the Long Description of a dimension value for a given dimension

Syntax

```
moodesc("[dim]" "[dimval]")
```

Example

```
=moodDesc( "ACC" "ACCOUNT1" )
```

mooQ

Returns the contents of a cell based on the fully qualified reference for the given cube. mooQ can be seen in the recap function of ORACLE OLAP.

Syntax

```
mooq("[QDR]")
```

Where [QDR] is a fully qualified node of a cube

Example

```
=mooQ( "CUBE(DIM1, DIMVAL1, DIM2, DIMVAL2..... DIMx, DIMVALx)" )
```

mooQN

Returns the contents of a cell based on the fully qualified reference for the given numeric cube. This is a similar function to mooQ but which is optimised for numeric retrieval. mooQN **cannot** be seen in the recap function of ORACLE OLAP.

Syntax

```
mooqn("[QDR]")
```

Where [QDR] is a fully qualified node of a cube

Example

```
=mooQN( "CUBE(DIM1, DIMVAL1, DIM2, DIMVAL2..... DIMx, DIMVALx)" )
```

mooQT

Returns the contents of a cell based on the fully qualified reference for the given textual cube. This is a similar function to mooQ but which is optimised for textual retrieval. mooQT **cannot** be seen in the recap function of ORACLE OLAP.

Syntax

```
mooqn("[QDR]")
```

Where [QDR] is a fully qualified node of a cube

Example

```
=mooQT("CUBE(DIM1, DIMVAL1, DIM2, DIMVAL2..... DIMx, DIMVALx)")
```

Excel Advanced Reporting Functions

In addition to the cell based functions there are mooFR functions which can be used to retrieve data from Oracle OLAP in a very fast way. These retrieve data in to arrays rather single cells.

Please refer to the example code within the myObjectiveOLAP-Examples workbook when reviewing this explanation

In the examples taken from the myObjectiveOLAP-Examples workbook we are asking Oracle OLAP to return a two dimensional array (table) to VBA for the current status of the Customer and Time dimensions from the UNITS_CUBE_COST OLAP variable.

The result of using these three functions is the end generation of a report by Channel and Time.

Unlike the cell based reports; reports built from mooFR functions can automatically expand when new dimension values are added.

There are three functions:

mooFR

Returns a two dimensional array from a variable in Oracle OLAP

Syntax

```
mooFR ([DOWN_DIM], [ACROSS_DIM], [OLAP VARIABLE])
```

Example

```
array = (o.mooFR("CUSTOMER", "TIME", "UNITS_CUBE_COST")
```

The contents of the array (table) passed back can then either be further processed in VBA or passed directly to a worksheet within Excel for reporting.

To ensure valid data is returned by this function it is the responsibility of the calling application to ensure that only the DOWN and ACROSS dimensions have one or more values in status. If you are using mooFR against a variable with more than two dimensions you should limit the paging dimensions to only one value each.

mooFR can not be used against one dimensional variables, however, mooFRDescDown or mooFRDescAcross can be used in these circumstances, even for numeric data.

mooFR observes the current status of the DOWN and ACROSS dimensions within the Oracle OLAP database and it is the responsibility of the calling application or user to set these appropriately.

mooFRDescDown

Returns a one dimensional array on the Y axis from Oracle OLAP

Syntax

```
mooFRDescDown ([DIMENSION], [OLAP VARIABLE])
```

Example

mooFRDescDown is primarily used to return either row or column descriptive data, however, it can be used to return any one dimensional array back to the calling desktop application.

mooFRDescDOWN observes the current status of the DOWN dimension within the Oracle OLAP database and it is the responsibility of the calling application or user to set these appropriately.

mooFRDescDown should only be used against one dimensional variables

mooFRDescAcross

Returns a one dimension array on X axis from Oracle OLAP

Syntax

```
mooFRDescAcross ([DIMENSION], [OLAP VARIABLE])
```

Example

mooFRDescAcross is primarily used to return either row or column descriptive data, however, it can be used to return any one dimensional array back to the calling desktop application.

mooFRDescAcross observes the current status of the DOWN dimension within the Oracle OLAP database and it is the responsibility of the calling application or user to set these appropriately.

mooFRDescAcross should only be used against one dimensional variables

Trouble Shooting Guide

Error messages

Run-time error '438':

The following Run-time error '438' is return when any function is called that does not exist. This may be because a valid myObjectiveOLAP function has been mis-spelled.

