

**IBM Planning Analytics 2.0 SDK**

**How to use TM1 Server’s REST API in PAx to Create Integrated Solutions**

Guido Tejeda Davila, IBM, [gtejeda@us.ibm.com](mailto:gtejeda@us.ibm.com) (Berlin)

Hubert Heijkers, IBM, [hubert.heijkers@nl.ibm.com](mailto:hubert.heijkers@nl.ibm.com) (New Orleans)

Table of Contents

[Getting ready 3](#_Toc495614669)

[Introduction 4](#_Toc495614670)

[Configuration 4](#_Toc495614671)

[Getting Started 4](#_Toc495614672)

[Inserting a button and assigning a Macro 5](#_Toc495614673)

[Adding “Microsoft Scripting Runtime” reference 7](#_Toc495614674)

[Comments in the sample 7](#_Toc495614675)

[Saving 8](#_Toc495614676)

[TM1-Top in Excel 8](#_Toc495614677)

[The VBA TM1 REST API ‘library’ 8](#_Toc495614678)

[Class: TM1API 9](#_Toc495614679)

[Class: TM1User 10](#_Toc495614680)

[Class: TM1Session 12](#_Toc495614681)

[Class: TM1Thread 13](#_Toc495614682)

[Adding a Utilities module 17](#_Toc495614683)

[Bringing it all together 18](#_Toc495614684)

[Running the code 22](#_Toc495614685)

[We Value Your Feedback! 23](#_Toc495614686)

[Acknowledgements and Disclaimers 24](#_Toc495614687)

# Getting ready

To be able to give you the best experience possible, and to allow us, authors, to be able to make last minute changes to the setup, samples and instructions for this Hands-On Lab, and because in our experience there is always something that we want to change last minute😉, we’ve build in a ‘get out of jail free card’.

As such there are a couple of steps that need to be executed before your machine is ready.

1 – Grabbing the latest files for the update

The latest versions of the files needed on this box, and the sources you’ll be working with in this lab, are all kept together in one Git repository on github.com.

Open a command box and execute the following command to grab the content of this repository:

git clone https://github.com/hubert-heijkers/iau17hol

Now let’s go to the folder holding the actual update:

cd iau17hol\vmupdate

2 – Updating the Virtual Machine

Next, we’ll execute a little batch file that updates a bunch of files and does some set up needed for the lab later. This update can be executed by typing the following command in the command box:

vmupdate.bat

Your VM is now up to date. You can now find the latest version of this document here:

C:\HOL-TM1SDK\Documents

Having an electronic copy of the instructions, most notably the Word document, might come in handy later when you’ll be ‘writing’ some code;-).

That’s all, enjoy the lab!

# Introduction

This programming exercise will demonstrate how to access and consume TM1 Server’s, OData compliant, REST API in VBA, showing you how to issue a request, marshal the results and display the information returned in an Excel Workbook.

IBM Planning Analytics for Microsoft Excel (PAx) allows an existing connection to a TM1 Server to be utilized for this purpose, which avoids having multiple connections from a single client open.

# Configuration

On the virtual machine you are using for this Hands-On Lab the TM1 Server “Planning Sample” has already been configured to accept REST requests, over HTTP, on port 8000.

The only thing you’ll need to make sure is, PAx having a dependency on Planning Analytics Workspace, that PA Local is already started. If isn’t you can start it yourself by running the start.bat from the PAW installation folder which for this lab can be found here:

C:\paw\paw26

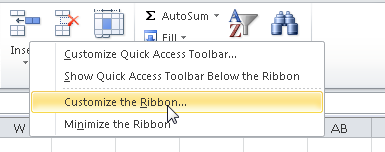
Obviously PAx itself has already been installed for this exercise as well.

# Getting Started

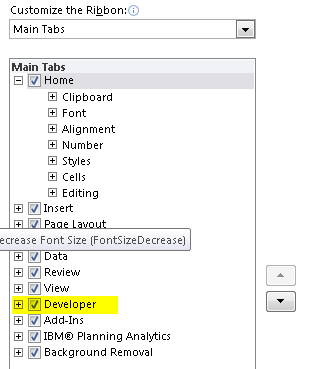
In the Windows Start Menu, click on the Excel icon to launch Microsoft Excel.



In order for a button to be added to the current Workbook, the Developer ribbon must be available. To get the Developer ribbon to display, right click on an empty portion on the existing ribbon (as shown below) and select “Customize the Ribbon”.



In the upcoming dialog look in the list to the right for “Developer” and check the checkbox next to it.



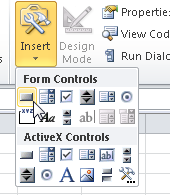
Click OK to accept the change and close the dialog.

Now that the Developer ribbon is available, let’s activate it to add a button to the Workbook.



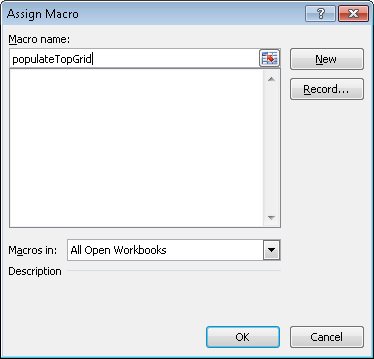
## Inserting a button and assigning a Macro

To add a button, click on the Insert icon and choose a button control.

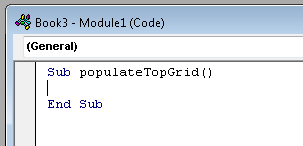


After drawing the button on the Workbook (recommended within A1 and C3), a dialog called “Assign Macro” will show.

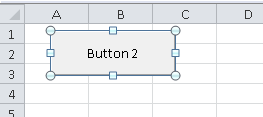
Replace the existing text with “populateTopGrid” and click on “New”.



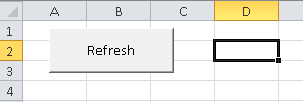
The Visual Basic for Applications (VBA) developer environment will open and show that a new module has been added to the Workbook. The code editor will show, that a new Macro has been added called Sub populateTopGrid.



Use ALT+Tab or the taskbar in order to switch back to the Excel Workbook. In Sheet1 of the Workbook the button will appear in the area drawn.



After adding the button, it will be in edit mode, which allows for the button caption to be modified. The suggested new caption is “Refresh”. When clicking out of the button into a cell, the button will go to run mode. In case needed, right clicking on the button will bring it back into edit mode.

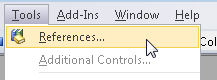


## Adding “Microsoft Scripting Runtime” reference

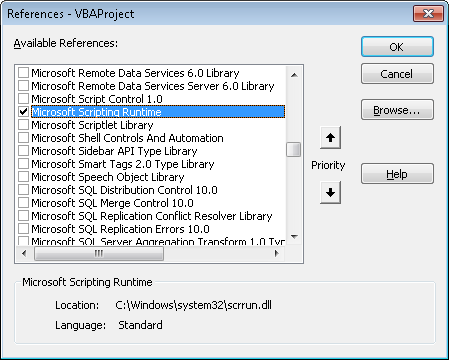
The example makes use of the dictionary object, which is contained within the Microsoft Scripting Runtime library.

To add the reference to the library use ALT+Tab or the taskbar to switch back to the VBA developer environment. In case the developer environment was closed, it can be re-opened by using the ALT+F11 from the Workbook

In the “Tools” menu, click on “References…”



In the list of Available References find “Microsoft Scripting Runtime” and check the checkbox next to the name as outlined in the screenshot below.



Click OK to accept the change and close the dialog.

## Comments in the sample

Lines starting with a single quote (') are comments in VBA. These lines are meant to be documentation of the code and do not need to be copied for the sample to work.

## Saving

It is recommended to frequently save your work. In Microsoft Excel CTRL+S, clicking on the disk icon in the toolbar or choosing the menu item File -> Save, will save the Workbook. Please keep in mind that during code execution (e.g. running code by pressing F5) or in debug mode, when the code execution has been paused, none of the methods of saving will work.

# TM1-Top in Excel

In this exercise we’ll grab the sessions and threads, like the TM1Top utility, and display the information in the current excel sheet. To get access to this information we’ll be connecting to the TM1 server of which we want to show this information, in this case our infamous Planning-Sample server, and, using TM1 server’s, OData compliant, REST API retrieve the information, marshal the information into an object structure and finally display the information in the Excel workbook.

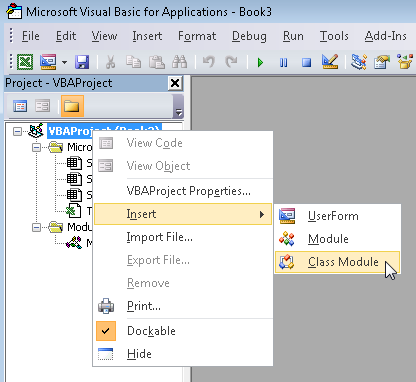
Part of this exercise involves converting the response of the server, which uses OData JSON format compliant JSON (JavaScript Object Notation), into a collection of objects representing the response. This response needs to get translated/marshalled into objects representing the response.

## The VBA TM1 REST API ‘library’

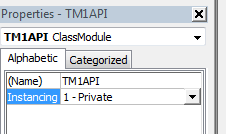
First, we’ll add some classes that collectively make up a little library that we use to de-serialize, a.k.a. marshalling, and represent the data that we retrieve, using the REST API, from the TM1 server.

By the way, this ‘library’ is by no means meant to be a complete representation of a TM1 Server REST API library for VBA. It only contains those functions needed for this lab example and by no means is intended to be production quality code. It might be a good starting point for one hopefully.

To insert a new class, go to the Project overview on the left-hand side of the VBA development environment, right click on “VBAProject (Book1)”, select “Insert…” and then “Class Module”



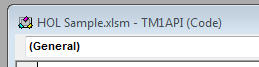
A new class module is now available. In the properties of the module (bottom left of the development environment), change the name of the class from Class1 to the name of the class that we’ll be creating, for example, our first class, TM1API.



### Class: TM1API

The TM1API class is the class which will hold the functions that we’ll use to de-serialize the responses we’ll get from our request(s) to the TM1 Server, returning us objects, or collections thereof, that represent the data returned by the server.

After renaming the name of the newly created class module the code below needs to be added to it. Click into the code window, which contains “TM1API” in the window title. Note that the name of the Workbook might be different in your case.



While indentation is optional, it greatly helps readability of the code.

Option Explicit

Public Function DeserializeSessionCollection(JSON As Object) As Collection

Dim oSessions As New Collection

If Not JSON.Members Is Nothing Then

Dim itemCount As Integer

itemCount = JSON.Members.Count

Dim iItem As Integer

For iItem = 0 To itemCount - 1

Dim Session As TM1Session

Set Session = DeserializeSession(JSON.Members.Item(iItem))

If Not Session Is Nothing Then

oSessions.Add Session

End If

Next iItem

End If

Set DeserializeSessionCollection = oSessions

End Function

Public Function DeserializeThreadCollection(JSON As Object) As Collection

Dim oThreads As New Collection

If Not JSON.Members Is Nothing Then

Dim itemCount As Integer

itemCount = JSON.Members.Count

Dim iItem As Integer

For iItem = 0 To itemCount - 1

Dim thread As TM1Thread

Set thread = DeserializeThread(JSON.Members.Item(iItem))

If Not thread Is Nothing Then

oThreads.Add thread

End If

Next iItem

End If

Set DeserializeThreadCollection = oThreads

End Function

Public Function DeserializeSession(JSON As Object) As TM1Session

Dim oSession As New TM1Session

If oSession.Deserialize(Me, JSON) Then

Set DeserializeSession = oSession

End If

End Function

Public Function DeserializeThread(JSON As Object) As TM1Thread

Dim oThread As New TM1Thread

If oThread.Deserialize(Me, JSON) Then

Set DeserializeThread = oThread

End If

End Function

Public Function DeserializeUser(JSON As Object) As TM1User

Dim oUser As New TM1User

If oUser.Deserialize(Me, JSON) Then

Set DeserializeUser = oUser

End If

End Function

### Class: TM1User

The TM1User class represents an User in the TM1 Server. For simplicity, and because we don’t need anything more in this lab example, we only gave a user a Name and FriendlyName property. The Name happens to be the unique identifier for a user whereas the friendly name is the name we’ll use to display in our workbook later on.

Create another class module by right clicking on the VBA Project (Book1), select “Insert…” and then “Class Module”.

Rename the new class module to “TM1User” and add the following code to it:

Option Explicit

Private m\_Name As String

Private m\_FriendlyName As String

Public Function Deserialize(oAPI As tm1api, JSON As Object) As Boolean

Deserialize = False

If Not JSON.Properties Is Nothing And JSON.Properties.Count > 0 Then

Dim propCount As Integer

propCount = JSON.Properties.Count

Dim iProp As Integer

For iProp = 0 To propCount - 1

Dim propertyName As String

propertyName = JSON.Properties.GetKeys().Item(iProp)

Select Case propertyName

Case "Name"

m\_Name = JSON.Properties.Item(propertyName).Value

Case "FriendlyName"

m\_FriendlyName = JSON.Properties.Item(propertyName).Value

'Additional properties we’d choose to implement would go here

End Select

Next iProp

Deserialize = True

End If

End Function

Public Property Get Name() As String

Name = m\_Name

End Property

Public Property Get FriendlyName() As String

FriendlyName = m\_FriendlyName

End Property

This is the first class in which you see a Deserialize function. This function contains the logic of converting the JSON body of an entity, in this case a User entity, returned by the server into an, User in this case, object.

We also added Property methods for every property we declared in the class. If you wanted to safe yourself some time you could simply make the properties themselves public, and remove the m\_, but that breaks with object oriented encapsulation rules. Nice side effect here is that all our objects, until we’d add functions or subs that would allow making changes, are read-only after having been marshalled.

We’ll use the same pattern in all of our classes that represent (entity) types in the server’s REST API.

### Class: TM1Session

The TM1Session class, as the name suggests, represents a session in the TM1 Server. A user connected, thru the REST API, to the server, gets assigned a session and any of the connections made referencing using that session, gets assigned a thread which, for the duration of such connection, will show up as one of the threads in the collection of threads associated with the session. The session also has a user associated with it, represented by the TM1User class we created just a minute ago, specifying which user connected to the server and therefore by definition the user of all the threads associated to this session.

Once again, another class module needs to get added to the project. Right click on the VBA Project (Book1), select “Insert…” and then “Class Module”.

Rename the new class module to “TM1Session” and add the following code to it:

Option Explicit

Private m\_ID As LongLong

Private m\_Context As String

Private m\_User As TM1User

Private m\_Threads As Collection

Public Function Deserialize(oAPI As tm1api, JSON As Object) As Boolean

Deserialize = False

If Not JSON.Properties Is Nothing And JSON.Properties.Count > 0 Then

Dim propCount As Integer

propCount = JSON.Properties.Count

Dim iProp As Integer

For iProp = 0 To propCount - 1

Dim propertyName As String

propertyName = JSON.Properties.GetKeys().Item(iProp)

Select Case propertyName

Case "ID"

m\_ID = JSON.Properties.Item(propertyName).Value

Case "Context"

m\_Context = JSON.Properties.Item(propertyName).Value

Case "User"

Set m\_User = oAPI.DeserializeUser(JSON.Properties.Item(propertyName))

Case "Threads"

Set m\_Threads = oAPI.DeserializeThreadCollection(JSON.Properties.Item(propertyName))

End Select

Next iProp

Deserialize = True

End If

End Function

Public Property Get ID() As LongLong

ID = m\_ID

End Property

Public Property Get Context() As String

Context = m\_Context

End Property

Public Property Get User() As TM1User

Set User = m\_User

End Property

Public Property Get Threads() As Collection

Set Threads = m\_Threads

End Property

Note that in the Deserialize function we are using methods from our TM1API class again to de-serialize entities, in this case a User, or collection of entities, the collections of threads associated with the session in this class.

Again, a pattern you’ll pattern you’ll see all over where there are relationships between types represented, in the REST API’s metadata by navigation properties.

### Class: TM1Thread

The TM1Thread class represents an active thread, or an active connection, in the TM1 Server. Any user with an active connection, irrespective of the API that is being used, has a thread associated to it. If that user is having the server perform some form of operation that we will be able to retrieve information about that operation as well as well as any locks that thread might be holding, how long that operation has been running, how much time the user has been waiting to get a lock, etc.

Note that whereas using any of the older APIs connections defined the life cycle of a ‘session’ effectively, that, with the introduction of the REST API, HTTP connections come and go and as such the Session they are connected to represent the life cycle of Session.

Create another class module named “TM1Thread” and add the following code to it:

Option Explicit

Private m\_ID As LongLong

Private m\_ThreadType As String

Private m\_Name As String

Private m\_Context As String

Private m\_State As String

Private m\_FunctionName As String

Private m\_ObjectType As String

Private m\_ObjectName As String

Private m\_RLocks As Integer

Private m\_IXLocks As Integer

Private m\_WLocks As Integer

Private m\_ElapsedTime As String

Private m\_WaitTime As String

Private m\_Info As String

Private m\_Session As TM1Session

Public Function Deserialize(oAPI As tm1api, JSON As Object) As Boolean

Deserialize = False

If Not JSON.Properties Is Nothing And JSON.Properties.Count > 0 Then

Dim propCount As Integer

propCount = JSON.Properties.Count

Dim iProp As Integer

For iProp = 0 To propCount - 1

Dim propertyName As String

propertyName = JSON.Properties.GetKeys().Item(iProp)

Select Case propertyName

Case "ID"

m\_ID = JSON.Properties.Item(propertyName).Value

Case "Type"

m\_ThreadType = JSON.Properties.Item(propertyName).Value

Case "Name"

m\_Name = JSON.Properties.Item(propertyName).Value

Case "Context"

m\_Context = JSON.Properties.Item(propertyName).Value

Case "State"

m\_State = JSON.Properties.Item(propertyName).Value

Case "Function"

m\_FunctionName = JSON.Properties.Item(propertyName).Value

Case "ObjectType"

m\_ObjectType = JSON.Properties.Item(propertyName).Value

Case "ObjectName"

m\_ObjectName = JSON.Properties.Item(propertyName).Value

Case "RLocks"

m\_RLocks = JSON.Properties.Item(propertyName).Value

Case "IXLocks"

m\_IXLocks = JSON.Properties.Item(propertyName).Value

Case "WLocks"

m\_WLocks = JSON.Properties.Item(propertyName).Value

Case "ElapsedTime"

m\_ElapsedTime = JSON.Properties.Item(propertyName).Value

Case "WaitTime"

m\_WaitTime = JSON.Properties.Item(propertyName).Value

Case "Info"

m\_Info = JSON.Properties.Item(propertyName).Value

Case "Session"

Set m\_Session = oAPI.DeserializeSession(JSON.Properties.Item(propertyName))

End Select

Next iProp

Deserialize = True

End If

End Function

Public Property Get ID() As LongLong

ID = m\_ID

End Property

Public Property Get ThreadType() As String

ThreadType = m\_ThreadType

End Property

Public Property Get Name() As String

Name = m\_Name

End Property

Public Property Get Context() As String

Context = m\_Context

End Property

Public Property Get State() As String

State = m\_State

End Property

Public Property Get FunctionName() As String

FunctionName = m\_FunctionName

End Property

Public Property Get ObjectType() As String

ObjectType = m\_ObjectType

End Property

Public Property Get ObjectName() As String

ObjectName = m\_ObjectName

End Property

Public Property Get RLocks() As Integer

RLocks = m\_RLocks

End Property

Public Property Get IXLocks() As Integer

IXLocks = m\_IXLocks

End Property

Public Property Get WLocks() As Integer

WLocks = m\_WLocks

End Property

Public Property Get ElapsedTime() As String

ElapsedTime = m\_ElapsedTime

End Property

Public Property Get WaitTime() As String

WaitTime = m\_WaitTime

End Property

Public Property Get Info() As String

Info = m\_Info

End Property

Public Property Get Session() As TM1Session

Set Session = m\_Session

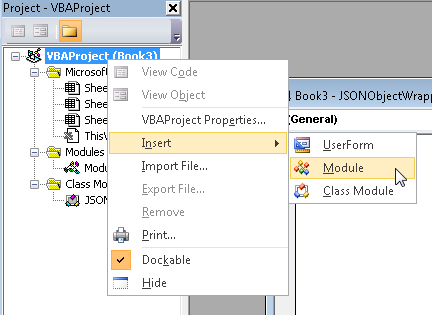
End Property

After seeing the previous two classes I’m sure you could have written this one yourself by now right;-?

## Adding a Utilities module

Now that we have are TM1 Server API, for as much as we’ll be using it in this lab sample, we’ll need to add some utility functions for maintaining the connection to TM1. Luckily PAx has us covered so what remains to be done is merely talk to it to have our requests send to the TM1 server.

In the Project overview on the left-hand side of the VBA development environment, right click on “VBAProject (Book1)”, select “Insert…” and then “Module”.



Even though not required, the module can be renamed to Utilities.

Add the following code to the newly created Utilities module.

Option Explicit

Dim m\_oCOAutomation As Object

Dim m\_oCAFE As Object

'Returns the instance of the Cognos Office Automation Object.

Public Property Get CognosOfficeAutomationObject()

On Error GoTo Handler:

'Fetch the object if we don't have it yet.

If m\_oCOAutomation Is Nothing Then

Set m\_oCOAutomation = Application.COMAddIns("CognosOffice12.Connect").Object.AutomationServer

End If

Set CognosOfficeAutomationObject = m\_oCOAutomation

Exit Property

Handler:

'<Place error handling here. Remember you may not want to display a message box if you are running in a scheduled task>

End Property

'Returns the instance of the Cognos Office Reporting Object.

Public Property Get Reporting()

On Error GoTo Handler:

'Fetch the object if we don't have it yet.

If m\_oCAFE Is Nothing Then

Set m\_oCAFE = CognosOfficeAutomationObject.Application("COR", "1.1")

End If

Set Reporting = m\_oCAFE

Exit Property

Handler:

MsgBox "Error"

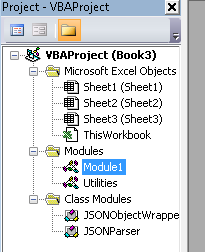
Err.Clear

End Property

## Bringing it all together

Now that we have the code to wrap what the TM1 Server will throw at us, and the help from PAx to connect us to the TM1 Server, sharing the session it already has with that server, we can now need the final snippet of code that, when pressing the button which we created earlier on our sheet already, asks PAx to send a request to the TM1 Server, de-serialize the response and, in this case, show the returned information in the Workbook on Sheet1.

This part of the code should go into the already existing module with the name “Module1”. To open the module, double click in the project overview on Module1.



There is existing code in this module, which was created when the button was added to the sheet. Complement the code with the code below.

Option Explicit

Sub populateTopGrid ()

Dim oAPI As New tm1api

Dim oSheet As Worksheet

Dim bScreenupdating As Boolean

Dim request As String

Dim response As Object

'get a reference to the first worksheet

Set oSheet = ThisWorkbook.Sheets("Sheet1")

'remember screenupdating property and turn it off to avoid flickering

bScreenupdating = Application.ScreenUpdating

Application.ScreenUpdating = False

'clear the range used to render the data

Range(oSheet.Cells(5, 1), oSheet.Cells(500, 16)).ClearContents

Range(oSheet.Cells(5, 1), oSheet.Cells(500, 16)).ClearFormats

'the request URL will be sent to the server

request = "/pm/tm1/server(Planning+Sample)/api/v1/Threads?$expand=Session($expand=User($select=Name,FriendlyName))"

'sending the request to the server

Set response = Reporting.ActiveConnection.Get(request)

'checking for a response

If Not response Is Nothing Then

Dim iRow As Integer

iRow = 5

'the response contains a collection wrapped in a JSON object and stored in a 'value' property

If Not response.Properties Is Nothing And response.Properties.Count() > 0 Then

Dim threadsJSON As Object

Set threadsJSON = response.Properties.Item("value")

Dim threadsCollection As Collection

Set threadsCollection = oAPI.DeserializeThreadCollection(threadsJSON)

'print the headers for the columns

With oSheet.Cells(iRow, 1)

.Value = "SessionID"

.Font.Bold = True

.Font.Color = RGB(150, 0, 0)

End With

With oSheet.Cells(iRow, 2)

.Value = "User"

.Font.Bold = True

.Font.Color = RGB(150, 0, 0)

End With

With oSheet.Cells(iRow, 3)

.Value = "TheadID"

.Font.Bold = True

.Font.Color = RGB(150, 0, 0)

End With

With oSheet.Cells(iRow, 4)

.Value = "Type"

.Font.Bold = True

.Font.Color = RGB(150, 0, 0)

End With

With oSheet.Cells(iRow, 5)

.Value = "Name"

.Font.Bold = True

.Font.Color = RGB(150, 0, 0)

End With

With oSheet.Cells(iRow, 6)

.Value = "Context"

.Font.Bold = True

.Font.Color = RGB(150, 0, 0)

End With

With oSheet.Cells(iRow, 7)

.Value = "State"

.Font.Bold = True

.Font.Color = RGB(150, 0, 0)

End With

With oSheet.Cells(iRow, 8)

.Value = "Function"

.Font.Bold = True

.Font.Color = RGB(150, 0, 0)

End With

With oSheet.Cells(iRow, 9)

.Value = "Object Type"

.Font.Bold = True

.Font.Color = RGB(150, 0, 0)

End With

With oSheet.Cells(iRow, 10)

.Value = "Object Name"

.Font.Bold = True

.Font.Color = RGB(150, 0, 0)

End With

With oSheet.Cells(iRow, 11)

.Value = "RLocks"

.Font.Bold = True

.Font.Color = RGB(150, 0, 0)

End With

With oSheet.Cells(iRow, 12)

.Value = "IXLocks"

.Font.Bold = True

.Font.Color = RGB(150, 0, 0)

End With

With oSheet.Cells(iRow, 13)

.Value = "WLocks"

.Font.Bold = True

.Font.Color = RGB(150, 0, 0)

End With

With oSheet.Cells(iRow, 14)

.Value = "Elapsed Time"

.Font.Bold = True

.Font.Color = RGB(150, 0, 0)

End With

With oSheet.Cells(iRow, 15)

.Value = "Wait Time"

.Font.Bold = True

.Font.Color = RGB(150, 0, 0)

End With

With oSheet.Cells(iRow, 16)

.Value = "Info"

.Font.Bold = True

.Font.Color = RGB(150, 0, 0)

End With

iRow = iRow + 1

'print the details for the individual threads

Dim threadCount As Integer

threadCount = threadsCollection.Count

Dim iThread As Long

For iThread = 1 To threadCount

Dim oThread As TM1Thread

Set oThread = threadsCollection.Item(iThread)

Dim oSession As TM1Session

Set oSession = oThread.Session()

If Not oSession Is Nothing Then

oSheet.Cells(iRow, 1) = oSession.ID

Dim oUser As TM1User

Set oUser = oSession.User()

If Not oUser Is Nothing Then

oSheet.Cells(iRow, 2) = oUser.FriendlyName

End If

Else

oSheet.Cells(iRow, 1).ClearContents

oSheet.Cells(iRow, 2).ClearContents

End If

oSheet.Cells(iRow, 3) = oThread.ID

oSheet.Cells(iRow, 4) = oThread.ThreadType

oSheet.Cells(iRow, 5) = oThread.Name

oSheet.Cells(iRow, 6) = oThread.Context

oSheet.Cells(iRow, 7) = oThread.State

oSheet.Cells(iRow, 8) = oThread.FunctionName

oSheet.Cells(iRow, 9) = oThread.ObjectType

oSheet.Cells(iRow, 10) = oThread.ObjectName

oSheet.Cells(iRow, 11) = oThread.RLocks

oSheet.Cells(iRow, 12) = oThread.IXLocks

oSheet.Cells(iRow, 13) = oThread.WLocks

oSheet.Cells(iRow, 14) = oThread.ElapsedTime

oSheet.Cells(iRow, 15) = oThread.WaitTime

oSheet.Cells(iRow, 16) = oThread.Info

iRow = iRow + 1

Next iThread

'setting the columns to autofit

oSheet.Columns("A:P").AutoFit

End If

End If

'resetting screenupdating

Application.ScreenUpdating = bScreenupdating

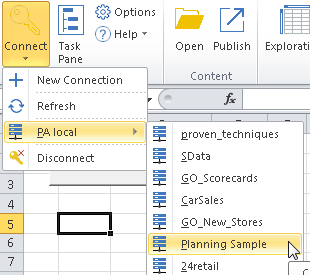
End Sub

## Running the code

Once the workbook has been saved, the code is now ready for execution. Due to the fact that the function name from the button assignment has been re-used, the button is now tied to the newly coded Macro.

Switch back to the Workbook by using ALT+TAB or the Windows taskbar.

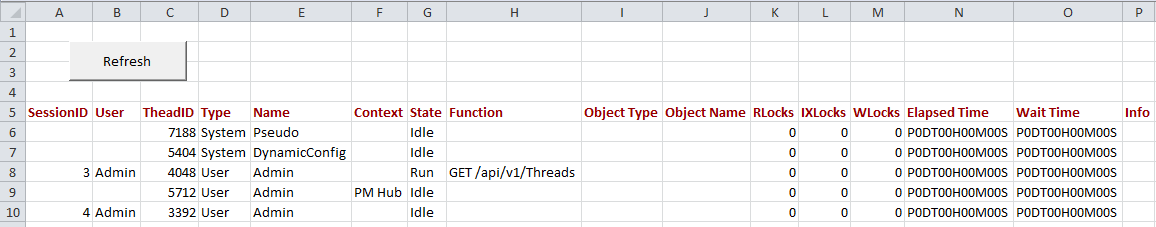
Before clicking on the button, please active the “IBM® Planning Analytics” ribbon and connect to the Planning Sample database using the “Connect” icon.



When prompted for login information, the user name/password is admin/apple.

After clicking on the Refresh button on the sheet, a grid should be displayed, showing thread information of the Planning Sample database.

The screenshot below may vary.



The complete sample, with all code, and a bonus feature, can be found here:

C:\HOL-TM1SDK\PAx

# We Value Your Feedback!

* Don’t forget to submit your IBM Analytics University session and speaker feedback! Your feedback is very important to us – we use it to continually improve the conference.
* Access the IBM Analytics University Conference Connect tool to quickly submit your surveys from your smartphone, laptop or conference kiosk.

# Acknowledgements and Disclaimers

Copyright © 2017 by International Business Machines Corporation (IBM). No part of this document may be reproduced or transmitted in any form without written permission from IBM.

**U.S. Government Users Restricted Rights — use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM.**

Information in these presentations (including information relating to products that have not yet been announced by IBM) has been reviewed for accuracy as of the date of initial publication and could include unintentional technical or typographical errors. IBM shall have no responsibility to update this information. **This document is distributed “as is” without any warranty, either express or implied. In no event shall IBM be liable for any damage arising from the use of this information, including but not limited to, loss of data, business interruption, loss of profit or loss of opportunity.** IBM products and services are warranted according to the terms and conditions of the agreements under which they are provided.

IBM products are manufactured from new parts or new and used parts.   
In some cases, a product may not be new and may have been previously installed. Regardless, our warranty terms apply.”

**Any statements regarding IBM's future direction, intent or product plans are subject to change or withdrawal without notice.**  
Performance data contained herein was generally obtained in a controlled, isolated environments. Customer examples are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual performance, cost, savings or other results in other operating environments may vary.

References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business.

Workshops, sessions and associated materials may have been prepared by independent session speakers, and do not necessarily reflect the views of IBM. All materials and discussions are provided for informational purposes only, and are neither intended to, nor shall constitute legal or other guidance or advice to any individual participant or their specific situation.

It is the customer’s responsibility to insure its own compliance with legal requirements and to obtain advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulatory requirements that may affect the customer’s business and any actions the customer may need to take to comply with such laws. IBM does not provide legal advice or represent or warrant that its services or products will ensure that the customer is in compliance with any law.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM’s products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a particular**, **purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

IBM, the IBM logo, ibm.com, Aspera®, Bluemix, Blueworks Live, CICS, Clearcase, Cognos®, DOORS®, Emptoris®, Enterprise Document Management System™, FASP®, FileNet®, Global Business Services®,  
Global Technology Services®, IBM ExperienceOne™, IBM SmartCloud®, IBM Social Business®, Information on Demand, ILOG, Maximo®, MQIntegrator®, MQSeries®, Netcool®, OMEGAMON, OpenPower, PureAnalytics™, PureApplication®, pureCluster™, PureCoverage®, PureData®, PureExperience®, PureFlex®, pureQuery®, pureScale®, PureSystems®, QRadar®, Rational®, Rhapsody®, Smarter Commerce®, SoDA, SPSS, Sterling Commerce®, StoredIQ, Tealeaf®, Tivoli® Trusteer®, Unica®, urban{code}®, Watson, WebSphere®, Worklight®, X-Force® and System z® Z/OS, are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at[: www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml).