TIBCO DataSynapse GridServer® Manager

COM Integration Tutorial

Version 7.1.0
July 2022

Contents

Accessing a Service from Visual Basic

This guide is your reference for developing COM applications that utilize TIBCO GridServer® installations. It is divided into several sections covering the fundamentals of how to use the GridServer COM API.

In this section, we demonstrate how to access a Service from a client application written in Visual Basic, using COMDriver. The example shows how a Visual Basic client can call COMDriver APIs to create and destroy a Service instance, execute the Service synchronously or asynchronously, and update the state of the Service. Although the Service in this example is a Java Service, client applications using COMDriver can access Services written in any server-side compatible language, provided that Service methods use strings for arguments and return values.

Getting Started

Installing the GridServer SDK

Before you begin, you must download and install the GridServer SDK, which is available from your Manager in the GridServer Administration Tool. The SDK includes the COMDriver. For more information about installing the SDK, see the *TIBCO GridServer® Developer's Guide*.

Setting up the Service

The Service in this example simulates a simple calculator, with functions for the four operators, plus a memory function. The Java source code is available in the GridServer SDK, in `/GridServerSDK-win32/examples/service/calculator/service/java/`.

The `JavaCalculator` class has a floating-point field called `memory`, and the following methods:

- `add`, `subtract`, `multiply`, `divide` — each takes two strings representing floating-point numbers, and returns a string representing the result of the arithmetic operation.
- `addToMemory` — takes in one string argument representing a floating-point number, and stores the sum of the input and `memory` values back in `memory`.
- `setMemory` — takes in one string argument representing a floating-point number and sets the `memory` field equal to it.
- `getMemory` — takes in a string argument, and returns the value of `memory`. (The argument is unused, but is required by the Services API.)

Before the client can access the Service, we have to register a Service Type and deploy the Service resources. Make sure the Service Type is registered under the name `JavaCalculatorExample` in the GridServer Service Type Registry; this is how we reference the Service from the client. The Service is automatically registered when GridServer is installed.

If the Grid Library has not been deployed, build and deploy it from `GridServerSDK-win32/examples/service/calculator/` by running `build-and-deploy.bat`.

To find out more about writing, registering, and deploying a Service, see the *TIBCO GridServer® Service-Oriented Integration Tutorial*.

Referencing COMDriver in the Visual Basic Editor

The COMDriver library has to be referenced in the Visual Basic client application. With the client application opened in Microsoft Visual Basic Editor, go to the **References** dialog box. Make sure that the reference to **DataSynapse Driver 1.0 Type Library** is checked. It must be referenced to `[COMDriver installation]\DSCOMDriver.exe`.

Client-side Code

Now that the Service has been registered and deployed, we can access it from the client through COMDriver. Let’s start by creating an instance of the Service.

Creating a Service Instance

To create an instance of the Service `JavaCalculatorExample` using COMDriver, use the `createService` function in the `DSCOMDRIVERLib` package.

```
Dim csfactory As New DSCOMDRIVERLib.ServiceFactory
Dim service As DSCOMDRIVERLib.service
Set service = csfactory.createService("JavaCalculatorExample", "0",
Nothing, Nothing)
```

The first line creates an instance of `ServiceFactory`. This is used to create TIBCO GridServer® Service instances. `ServiceFactory` is defined in the `DSCOMDRIVERLib` library, which resolves in `DSCOMDriver.exe`.

The second line declares a variable called `service` that holds the Service instance. This variable must be visible to any other part of the code that wishes to interface with the Service. You can see that the rest of the code in this section references this variable.

The third line calls the `createService` method, which takes four arguments:

- `serviceName` — The name of the registered Service (String).
- `initData` — The state initializing data for the Service (String).
- `options` — Service options; see the `Options` class in the `Service` JavaDoc for available options.
- `description` — Service descriptions; see the `Description` class in the `Service` Javadoc for available description options.

The third and fourth arguments are of type `Properties`, which is a set of name-value pairs. Consult the COMDriver documentation for details.

The method returns the created Service interface. In our example, an instance of the `JavaCalculatorExample` Service is initialized with the value `0`, with no options and description.

If the supplied Service name is not registered on TIBCO GridServer®, the application fails with a runtime error: `Service: servicename.csdd not found`.

Submitting Requests Synchronously

Work requests can be submitted to a Service synchronously or asynchronously. Let us go through submitting a request synchronously first. Depending on the number of input arguments you have for the Service function, you can use the `execute` or `executeWithArray` function.

Both `execute` functions take in three arguments:

1. `methodName` — the name of the method to invoke.
2. `data` — the input data.
3. `discriminator` — the discriminator used for this request; discriminators are discussed in a later section.

Both return the output data as a string. The input and output arguments are restricted to strings, for reasons of network serialization, and better interoperability between clients and Services of different languages.

The only difference between the two functions is that `executeWithArray` accepts an array of strings as the input data argument, whereas `execute` accepts a single string.

For example:

```
Dim numInMemory As String
numInMemory = service.execute("getMemory", "", Nothing)
Dim sum As String
Dim args(1) As String
args(0) = "25"
args(1) = "75"
sum = service.executeWithArray("add", args, Nothing)
```

If the Engine-side function being executed takes one or no arguments, it is more convenient to use `execute`; otherwise, place your arguments in an array, and pass them into `executeWithArray`. If the function takes no arguments, like the `getMemory` function, use `execute` with an empty string.

The `execute` functions do not return until the Service completes the request and returns the result. To have the function return immediately, use the submit function to submit requests asynchronously, as described below.

Submitting Requests Asynchronously

To submit requests to the Service asynchronously, use the `submit` functions. They follow the same format as `execute`. One of the two functions is more suitable depending on the number of input arguments the Service function takes: `submit` for zero or one argument, and `submitWithArray` for more than one argument.

The `submit` functions return immediately, with the invocation ID (a Long integer) as the output argument. The ID increments sequentially and starts at 0; it can be used to identify results when they are later collected.

```
Dim ID1 As Long
ID1 = service.submit ("getMemory ", "", Nothing)
Dim ID2 As Long
Dim args(1) As String
args(0) = "25"
args(1) = "75"
ID2 = service.submitWithArray("add", args, Nothing)
```

The results are collected by calling the `collectNext` function.

Collecting Results During Asynchronous Execution

Since Visual Basic is not multithreaded, there are no callbacks for results collection. The `collectNext` function is used for collecting results during asynchronous calls. The function returns one of the available results from the Service; if none are available, it waits for the number of milliseconds supplied as an input argument before returning. The timeout prevents a client application from waiting indefinitely on a Service.

`collectNext` returns the result as a `ServiceData` type, which includes the following properties:

- `Id` — The Service invocation ID, which can be used to match the result with the corresponding submission.
- `IsError` — A Boolean that signifies whether the execution resulted in an error; if so, the data property contains the error message.
- `Data` — The result (Service function returns value or error).

For the same reason as `execution` and `submit` functions, all output arguments are pure strings.

To make collecting multiple results easier, the `InvocationCount` parameter of a Service is equal to the number of outstanding invocations.

The following code demonstrates how one can implement results collection. It repeatedly checks a Service for available results. If there are results, one result is collected; the code checks to see if the result is an error, and prints out the result data accordingly. If there are no results, it waits for 1 second, and then loops. The loop ends when all the results have been collected.

```
While service.InvocationCount > 0
        Dim sd As ServiceData
        Set sd = service.collectNext(1000)
        If Not sd Is Nothing Then
                If sd.IsError Then
                        Debug.Print "collected " & sd.Id & " error: " & sd.Data
                Else
                        Debug.Print "collected " & sd.Id & ": " & sd.Data
                End If
        End If
Wend
Debug.Print "done collecting"
```

Stateful Service Operations

To better understand how to manage state from the client side, let us take a quick look at how Services on TIBCO GridServer® manage it. Let's consider our `JavaCalculatorExample` Service.

The state maintained by the Service is a field called `memory`. There are two methods that are responsible for updating the state:

- `addToMemory`
- `setMemory`

Although both methods update the state, TIBCO GridServer® has to handle them differently to ensure that the state is properly maintained. `addToMemory` refers to the previous value of `memory`, whereas `setMemory` does not. TIBCO GridServer® denotes the first type of method as an `appendStateMethod`, and the second type as `setStateMethod`.

When a Service instance is created on an Engine, the Engine replays all state-changing methods from the last `setStateMethod`, or from the beginning if there have been no `setStateMethods`. Depending on the duration of the request, it might be worthwhile to

follow `appendStateMethods` with `setStateMethods` periodically, to save memory on the Manager.

These special state-changing methods have to be identified when the Service is registered in TIBCO GridServer®. Within the Service configuration of the `JavaCalculatorExample`, under the section **ContainerBinding**, these method names must be included in the fields titled **appendStateMethods** and **setStateMethods**.

Once the methods have been registered on the Service, a client can only access them through the `updateState` functions. These functions behave in a similar way to `execute` functions. They take three input arguments:

- `methodName` — the name of the method to invoke;
- `data` — the input data;
- `append` — a Boolean signifying whether it is an `appendStateMethod` and updates the state accordingly.

These functions do not return output arguments.

In the same manner as `execute` and `submit`, one of two `updateState` methods are more suitable depending on the number of input arguments that the Service function takes. These functions are: `updateState` and `updateStateWithArray`.

The following Visual Basic code updates the `memory` of the `JavaCalculatorExample`:

```
service.updateState "setToMemory", "25", False
service.updateState "addToMemory", "75", True
```

The resultant value of `memory` is 100.

Discriminators

Discriminators allow Service requests to be performed only on Engines that meet the discriminator's requirements. To construct a discriminator, use the COM types `PropertyDiscriminator` and `PropertyComparator`. A `PropertyDiscriminator` is a collection of `PropertyComparators`, each of which expresses one requirement on Engines that must evaluate to true in order for the engine to qualify. See the COMDriver API reference for a full list of comparators and machine properties; it is located at `[COMDriver installation]/docs/COMDriver/DSCOMDriverRef.doc`.

The following Visual Basic code creates a new `PropertyDiscriminator`, which limits operation to machines with more than 200MB of free disk space, and the custom property `foo` equal to the value `bar`. It then submits a request to the `add` function with the `PropertyDiscriminator`.

```
Dim discrim As New PropertyDiscriminator
Dim pc As New PropertyComparator
pc.Init FREE_DISK_MB, "200", GREATER_THAN_EQ, False
discrim.Add pc
pc.Init "foo", "bar", EQUALS, False
discrim.Add pc
Dim args(1) As String
args(0) = "25"
args(1) = "75"
ID2 = service.submitWithArray("add", args, discrim)
```

The `Init` function of `PropertyComparator` initializes the comparator as comparison with a predefined Engine property, and takes four arguments:

- `name` — A string representing the name of the Engine property.
- `value` — A string representing the value of the Engine property.
- `op` — The comparison method, from the enumeration `DSComparatorType`.
- `nullCompare` — The result of the comparison when the value of the property is not defined on the Engine.

For more information about initializing `PropertyComparator`, see the COMDriver API reference (`[COMDriver install dir]\docs`). For more information about custom properties for Engines, refer to the “Using Conditions” topic of the *TIBCO GridServer® Developer’s Guide*.

Setting Driver Properties

COMDriver obtains its configuration from a file called `driver.properties`. It is possible to override the values in `driver.properties` programmatically, using the `DriverManager` class. This ability is most commonly used to set a user name and password for authentication to the Grid. Setting these values in the code allows different users to reference the same `driver.properties` file, and avoids having a cleartext password in the file. To set the user name and password in your code, place the following lines before any other use of COMDriver. This code assumes that `user` and `password` are variables that have been set to the desired user name and password.

```
Dim dm As New DSCOMDRIVERLib.DriverManager
dm.setProperty DSUSERNAME, user
dm.setProperty DSPASSWORD, password
```

Destroying Service Instances

When clients have completed submitting requests to a Service instance, it is responsible for destroying the instance to free resources on the server. To do so, simply call the destroy function on the Service. Any outstanding invocations are canceled.

Excel Demo

This section details an example of how Windows applications can interface with TIBCO GridServer® Services through COMDriver and Visual Basic. The client application is a Portfolio Valuation calculation in Microsoft Excel, and provides two ways of performing the calculation: locally in Visual Basic, or as a Java Service on TIBCO GridServer® using COMDriver.

The example is available from the COMDriver installation, and is located in `[DSCOMDriver installation]/examples/exceldemo`.

Getting Started

Before running the demo, you must enable Macros in Microsoft Excel, deploy the resources to TIBCO GridServer®, and register the Service Type.

Enabling Macros in Excel

By default, macros are not enabled in Excel spreadsheets. Enable them as follows:

1. Start Excel.
2. In Excel, choose **Tools > Macros > Security**.
3. Select **Medium Security**.

Deploying Resources

To deploy the resources used by the spreadsheet to your Engines, in the COMDriver installation, upload the Grid Library ZIP file in `examples/exceldemo/service/` directory in the GridServer Administration Tool on the **Grid Components > Services > Grid Libraries** page.

Registering the Service Type

After deploying the Excel demo, you must register the Service Type as follows:

1. Go to **Services > Services > Service Types**.
2. In the empty box at the bottom of the table, enter “exceldemo” and select **java** for the implementation. A new window appears with options for registering a Service.
3. In **className,** enter `exceldemo.TradeValuationService`.
4. In **Options**, set the following:
    a. Set **autoPackNum** to 5.
    b. Set **gridLibrary** to `exceldemo`.
    c. Set **gridLibraryVersion** to `1.0.0.1`.
5. Click **Save**.

Executing the Demo

Open the Excel file `OptionValuationDemo.xls`, and when prompted, enable Macros. After several seconds, a message appears that says: `GridServer connection established`. Click **OK**. If this does not occur, make sure the Manager is running and visible from your machine, and that you supplied the Manager name and port when you installed COMDriver.

To run the demo, either select **Use Local Spreadsheet** to run it locally, or **Use DataSynapse Grid**, to enable Excel to access the Java Service on GridServer, and then click **Value Portfolio**.

Note that the calculation completes faster if you select the **Use DataSynapse Grid** option, because calculations in Visual Basic are much slower than in Java.

The Visual Basic Code

To see the Visual Basic code within the Excel file, from the **Tools** menu, select the **Macro** menu, then click **Visual Basic Editor**. This opens up a Visual Basic Editor; select **Sheet1** in the Project Explorer to see the source code.

The code must be familiar to you if you read through the last Visual Basic example. On loading the Excel file, the `InitializeLC()` function is performed. It instantiates `ServiceFactory`, and creates an instance of the Service `exceldemo`.

```
Public Service As DSCOMDRIVERLib.Service
Sub InitializeLC()
    Dim f As New DSCOMDRIVERLib.ServiceFactory
    On Error GoTo iError
    If (Service Is Nothing) Then
        Set Service = f.createService("exceldemo", "", Nothing, Nothing)
    End If
    MsgBox ("GridServer connection established.")
    GoTo iEnd
iError:      MsgBox ("GridServer Service not found: " & "(" &
Err.Description & ".)")
iEnd:
End Sub
```

The Visual Basic function that executes the calculation on GridServer is called `runLiveCluster`. It calls the Java function `valueTrade` of the Service asynchronously using the `Submit` function in COMDriver, sequentially from the first to the last row. The function takes one string argument: all the values on the row of the deal, concatenated into one string, delimited by spaces.

The code correctly assumes that Invocation IDs returned by the `submit` function are sequential. The invocation ID of the last call is written to the variable `result`. This value is important for indexing the results later on.

```
For rwIndex = begRow To endRow
Dim a As String
a = Cells(rwIndex, 2).Value & " " & Cells(rwIndex, 3).Value & " " _
& "0.05 " & Cells(rwIndex, 4).Value & " " & Cells(rwIndex, 5).Value & " "
_
           & Cells(rwIndex, 6).Value & " " & Cells(rwIndex, 7).Value & "
" & sims
Dim result
result = Service.submit("valueTrade", a, Nothing)
Next rwIndex
```

Since `submit` is an asynchronous call, the results are collected separately. The code calls the `collectNext` function repeatedly until the number of results collected is equal to the number of rows. Alternatively, it can use `Service.InvocationCount`, to determine if there are still pending results. See the previous example for an implementation of this.

The row number is calculated by offsetting the invocation IDs by the variable `result`:

```
'   now collect
Dim count As Integer
While count < endRow - begRow + 1
Dim data As ServiceData
Set data = Service.collectNext(100)
If Not (data Is Nothing) Then
Cells(data.ID + begRow - (result - (endRow - begRow)), 7).Value =
data.data
count = count + 1
End If
Call updateProgress(count * 100# / (endRow - begRow + 1))
Wend
```

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

Documentation for TIBCO GridServer® is available on the TIBCO GridServer® Product Documentation page.

The following documents for this product can be found in the TIBCO Documentation site:

- TIBCO GridServer® Release Notes
- TIBCO GridServer® Installation
- TIBCO GridServer® Introducing TIBCO GridServer®
- TIBCO GridServer® Administration
- TIBCO GridServer® Developer's Guide
- TIBCO GridServer® Upgrade
- TIBCO GridServer® Security
- TIBCO GridServer® COM Integration Tutorial
- TIBCO GridServer® PDriver Tutorial
- TIBCO GridServer® Speedlink
- TIBCO GridServer® Service-Oriented Integration Tutorial

How to Contact TIBCO Support

Get an overview of TIBCO Support. You can contact TIBCO Support in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support website.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to TIBCO Support website. If you do not have a user name, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to TIBCO Community.

Legal and Third-Party Notices