JADE REST Services

Version 2018

Contents

JADE REST Services **4**

Introduction 4
Internet Information Services (IIS) 4
REST Services Applications and JadeRestService Class 5
Exercise 1 – Setting Up IIS 6
Exercise 2 – Creating a REST Provider 9
Making REST Requests from JADE 15
HTTP Request Types in JADE REST Requests 16
Exercise 3 – Creating a REST Client 16
Exercise 4 – Making a POST Request over REST 20

JADE REST Services

Introduction

JADE provides the ability to provide RESTful web services from a JADE database, as well as a limited capacity for acting as a REST client. The **JadeRestService** class handles the processing of REST requests, and the **JadeHTTPConnection** class can send **POST** and **GET** requests to a REST web server.

Representational State Transfer (REST) is a type of web service that receives and responds to HTTP requests in JSON (JavaScript Object Notation) or XML (eXtensible Markup Language). REST web services are a lightweight alternative to other web services such as SOAP (Simple Object Access Protocol) and WSDL (Web Services Description Language).

One aspect of REST web services that causes them to be more lightweight than SOAP-based web services is their property of statelessness. Statelessness means that a web server handling REST requests does not store any information about the client on the server. The client is responsible for maintaining any required state information and communicating its state to the server as part of any request that requires it. This means that, from the server's perspective, all requests are completely isolated from each other and as such, the cost of each client is lessened, improving scalability potential.

REST requests embed all required query parameters within the URL, meaning that the URL will contain not only the location of the web provider but also the type of request and any parameters needed for that request. This style of URL is known as a REST-style URL. For example, consider the following URL.

http://localhost/RestfulJade/jadehttp.dll/Customer/3.xml?RestService

This URL contains both where to find the web service (that is, the location of the resource) expressed in the **http://localhost/RestfulJade/jadehttp.dll/** part and also the request itself, expressed in the **Customer/3.xml?RestService** part. This request part of the REST-style URL expresses that the method name is **Customer**, the parameter to pass is **3**, the returned message should be in **XML** format, and the name of the JADE web services application is **RestService**.

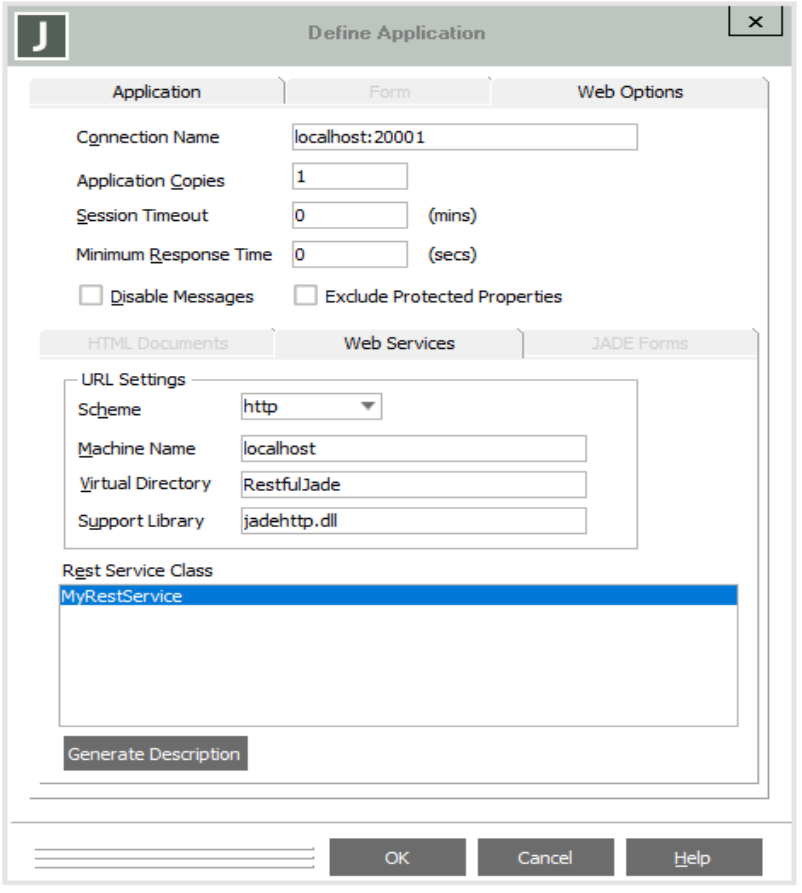
Internet Information Services (IIS)

Internet Information Services (IIS) is Microsoft's web server software for hosting web services. IIS is most commonly run on a dedicated machine running Windows Server but can also be run on a home computer running Windows 7 or later for smaller-scale operations.

When a machine running IIS receives a web request, it checks its list of applications for one that matches the request, and delegates the request to the web service provider specified in the application. For JADE databases acting as a web service provider, this is done by specifying the **jadehttp.dll** file, found within the **bin** directory of the JADE installation folder. This DLL file then sends the request through the port specified in the **jadehttp.ini** file. Whichever JADE application is running and listening for requests on the matching port handles the request and returns its reply through IIS and back to the client.

REST Services Applications and JadeRestService Class

JADE's implementation of REST web services relies on having an application of application type **Rest Services** (or **Rest Services, Non-Gui**) with a reference to a subclass of the **JadeRestService** class specified in the **Rest Service Class** list box on the **Web Options** sheet of the Define Application dialog.



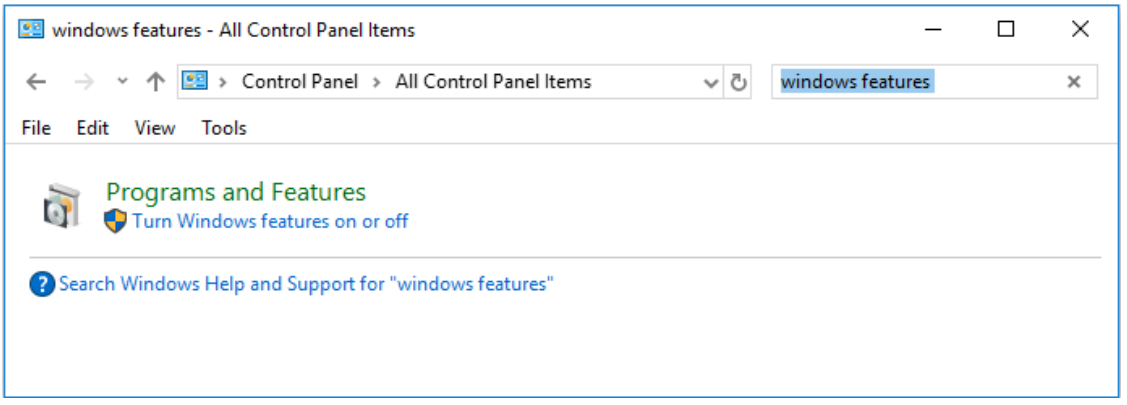
This application is the entry point into the JADE database and calls the **processRequest** method of the specified **JadeRestService** class subclass whenever a request is received on the port specified in the **Connection Name** text box.

The **processRequest** method of the **JadeRestService** class is responsible for determining which method should be called to fulfil the request, calling that method, and then calling the **reply** method of the **JadeRestService** class to send the returned value of the called method back to the client. To determine which method to call, it concatenates the type of request (that is, **GET**, **POST**, **PUT**, or **DELETE**) with the name provided in the URL. For example, a **GET** request with **Customer** as the provided name calls the **getCustomer** method.

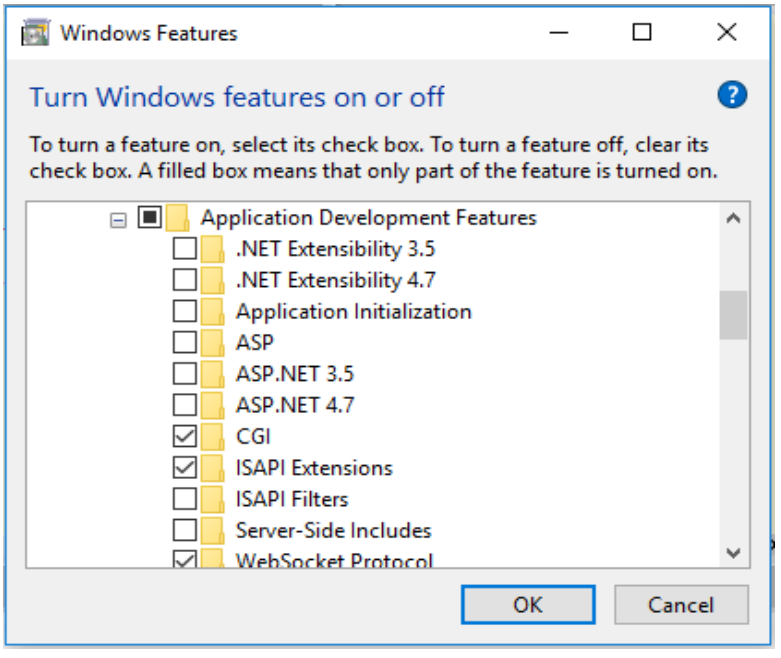
Exercise 1 – Setting Up IIS

In this exercise, you will set up IIS for use with REST web services.

1. Open the Windows Control Panel and search for **windows features**.



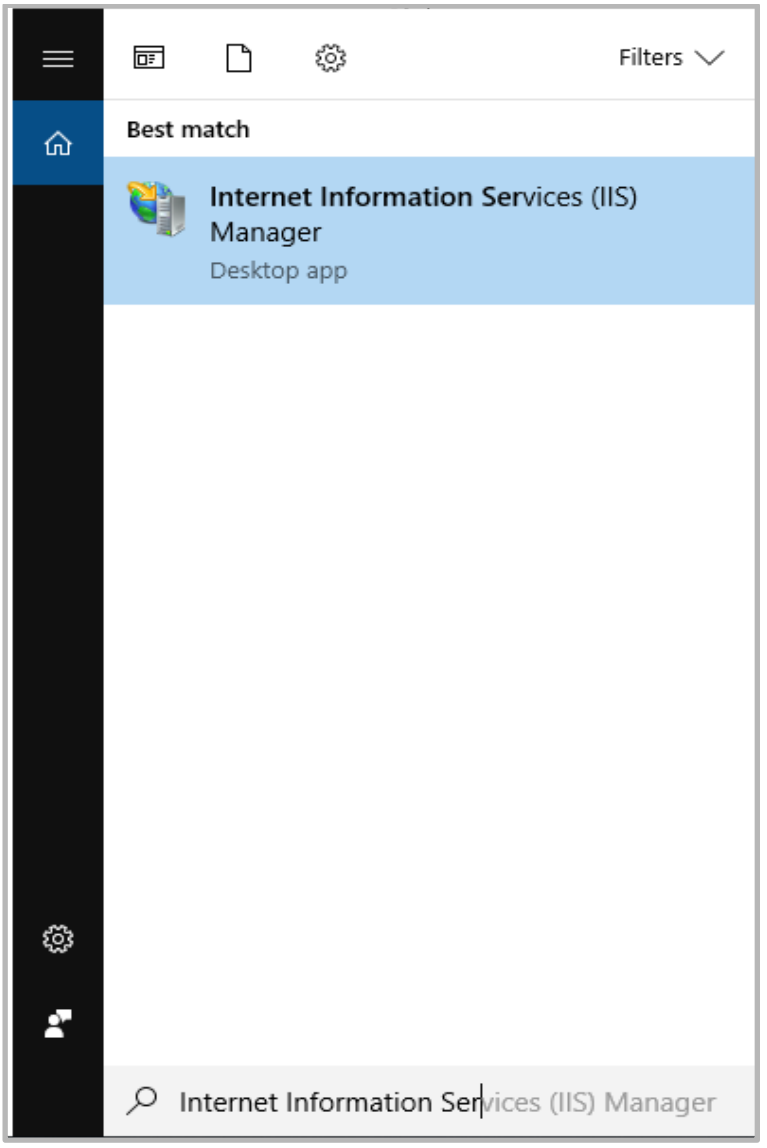
2. Select **Turn Windows features on or off**. The Windows Features window is then displayed.



3. From this window, expand **Internet Information Services**, then **World Wide Web Services**, then **Application Development Features**, and check the following check boxes.
   - CGI
   - ISAPI Extensions

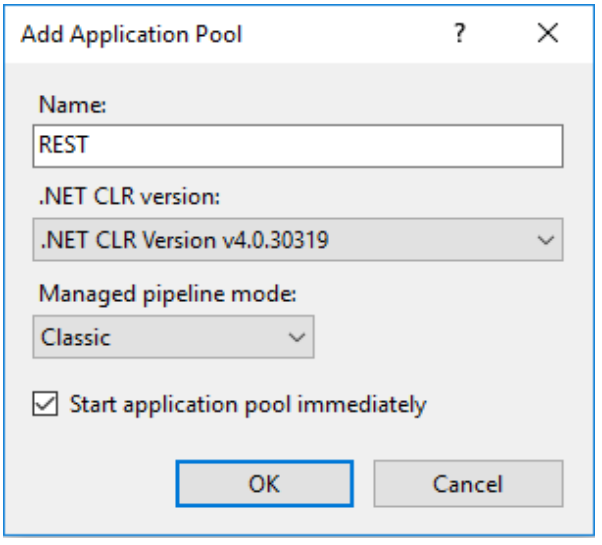
4. Open the Internet Information Services (IIS) Manager.

Tip The easiest way to access it is to search for it.

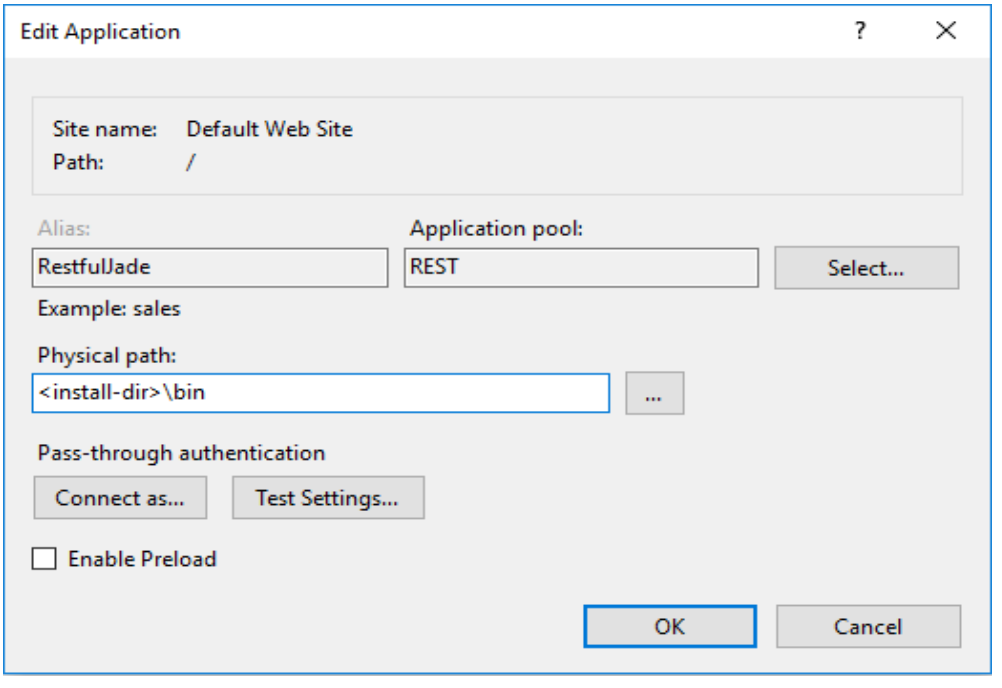


5. From the **Connections** panel on the left of the IIS Manager, select **Application Pools**.
6. From the **Actions** panel on the right, select **Add Application Pool**.

7. Call the application pool **REST** and set **Managed pipeline mode:** to **Classic**.

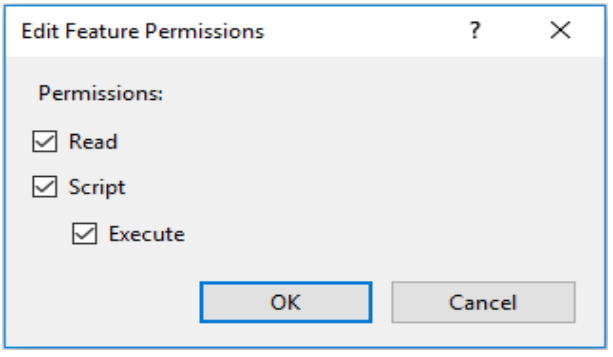


8. From the **Connections** panel, right-click **Default Web Site** and then select **Add Application**.
9. Call the application **RestfulJade** and set the physical path to the **bin** directory of your JADE installation.

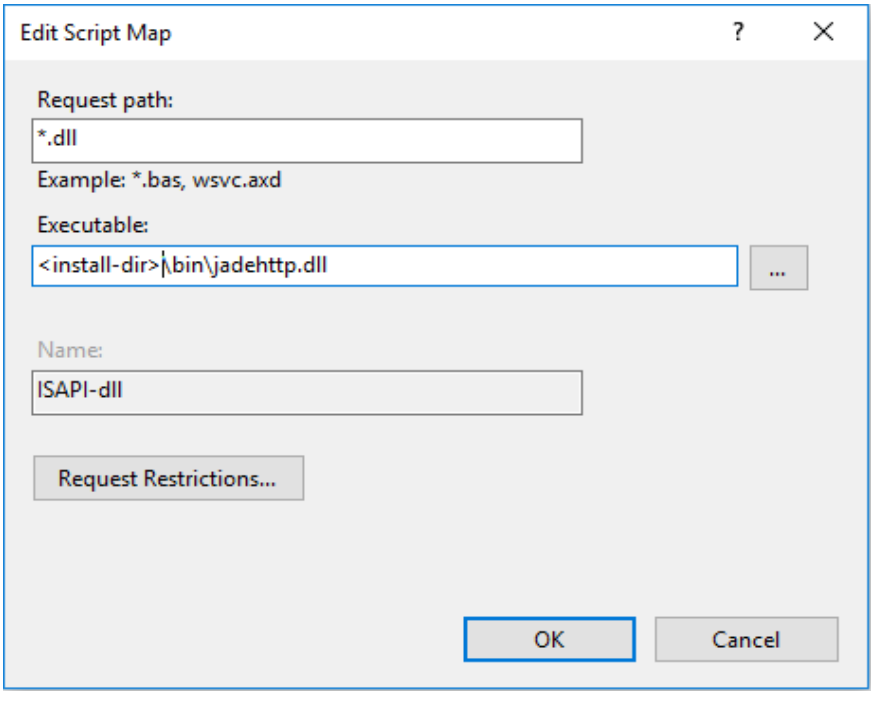


10. Select the newly created **RestfulJade** application and from the center panel, open **Handler Mappings**.

11. Select **CGI-exe** from the middle panel and then select **Edit Feature Permissions** from the **Actions** panel in the right. Check all check boxes.



12. Select **ISAPI-dll** from the middle panel and then select **Edit** from the **Actions** panel on the right. Set the executable to the **jadehttp.dll** file in the **bin** directory of your JADE installation.

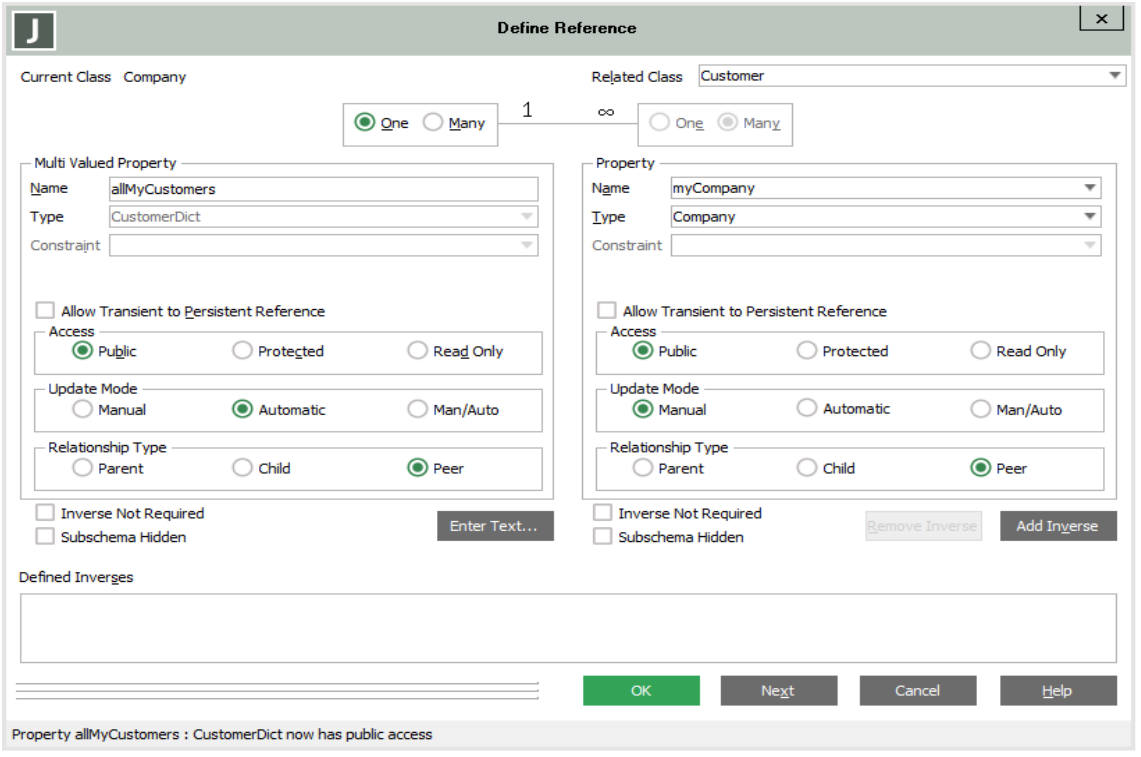


Exercise 2 – Creating a REST Provider

In this exercise, you will create a JADE application that provides REST web services.

1. Open JADE if it is not already open, and then create a schema called **RestfulSchema**.
2. In **RestfulSchema**, create a class called **Company** and another class called **Customer**.
3. To the **Customer** class, add a public attribute called **id** of type **Integer** and a public attribute called **name** of type **String**.
4. Search for **MemberKeyDictionary** (F4) and then add a subclass to it called **CustomerDict** that has membership **Customer** and key **id**.

5. To the **Company** class, add an inverse reference with **Customer**, as follows.



- Inverse property called **myCompany**
- Both access types public
- Automatic **allMyCustomers** to manual **myCompany**

6. Create a JadeScript **init** method, coded as follows, and then run it.

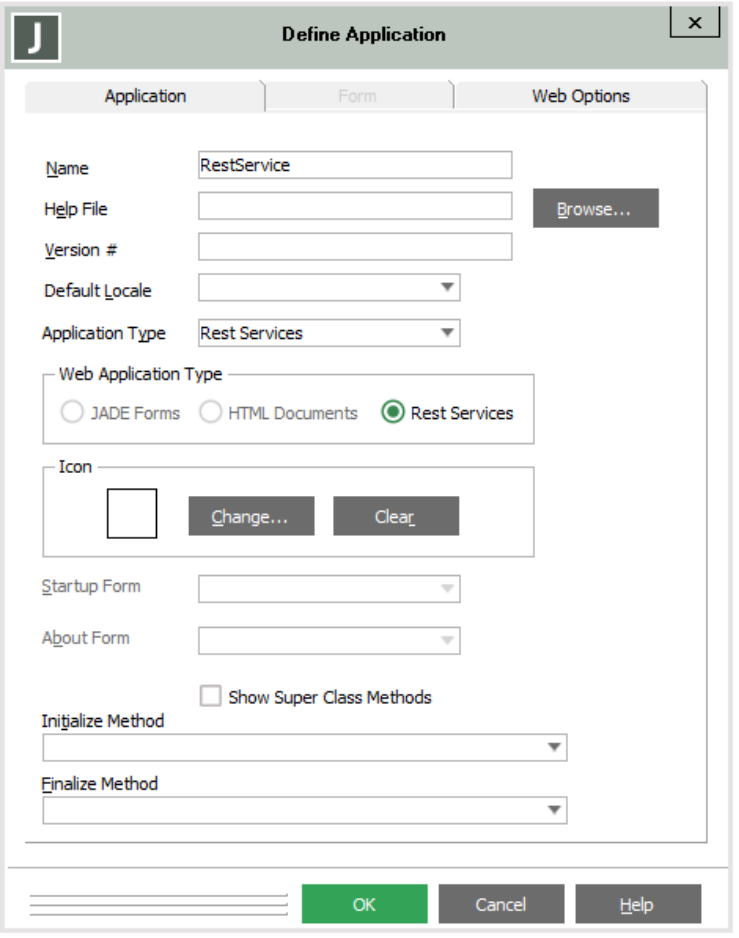
```
init();

vars
    company : Company;
    cust    : Customer;

begin
    beginTransaction;
    create company;

    create cust;
    cust.id := 1;
    cust.name := "Albert Brian";
    cust.myCompany := company;

    create cust;
    cust.id := 2;
    cust.name := "Carla Donaldson";
    cust.myCompany := company;

    create cust;
    cust.id := 3;
    cust.name := "Edward Fields";
    cust.myCompany := company;
    commitTransaction;
end;
```

7. In the Class Browser, search for **JadeRestService** (F4) and then add to it a subclass called **MyRestService**.

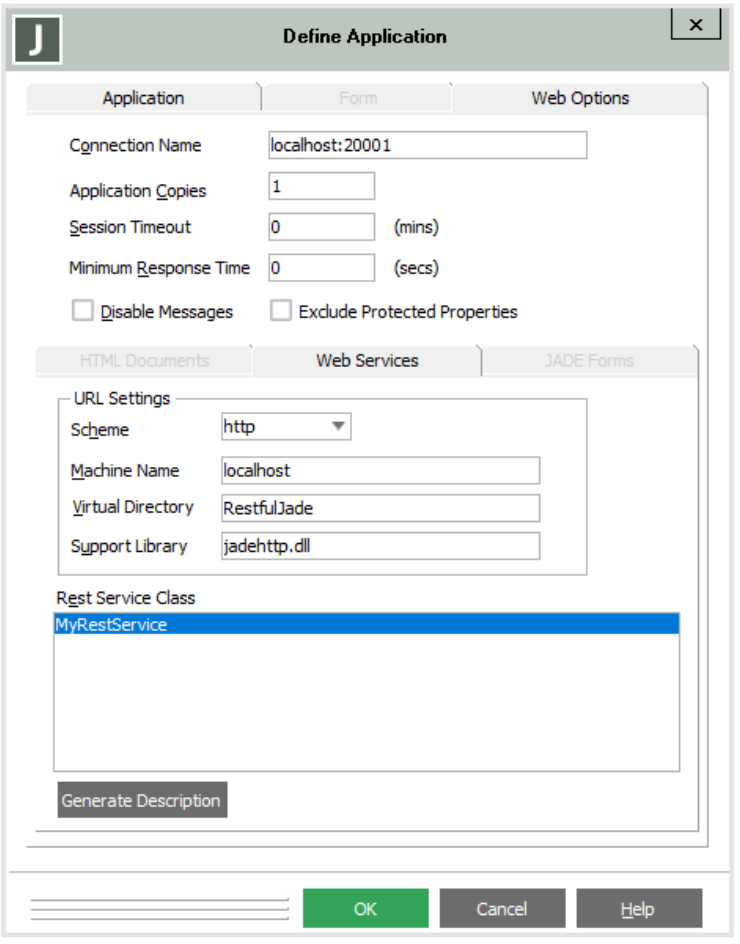
8. To the **MyRestService** class, add a method called **getCustomer**, coded as follows.

```
getCustomer(id : Integer) : Customer;
vars
    theCompany      : Company;
    theCustomer     : Customer;
    proxyCustomer   : Customer;

begin
    // Company.firstInstance is not scalable, but will serve for this exercise
    theCompany      := Company.firstInstance;
    theCustomer     := theCompany.allMyCustomers[id];

    // We need to send a proxy to avoid object access conflicts.
    proxyCustomer   := theCustomer.copySelf(true);
    return proxyCustomer;
end;
```

9. Open the Application Browser (CTRL+L) and add an application called **RestService**, with the application type **Rest Services**.



Select the **Web Options** sheet and specify the following.



- For the connection name, enter **localhost:20001**
- For the machine name, enter **localhost**
- For the virtual directory, enter the name of the IIS application created in Exercise 1 of this module; that is, **RestfulJade**
- All other settings can be left as the default values

10. Open a web browser and then navigate to the following URL.

    http://localhost/RestfulJade/jadehttp.dll/Customer/3/?RestService

Note that the URL has the following parts.

a. **http://localhost** means use the current computer as the web server

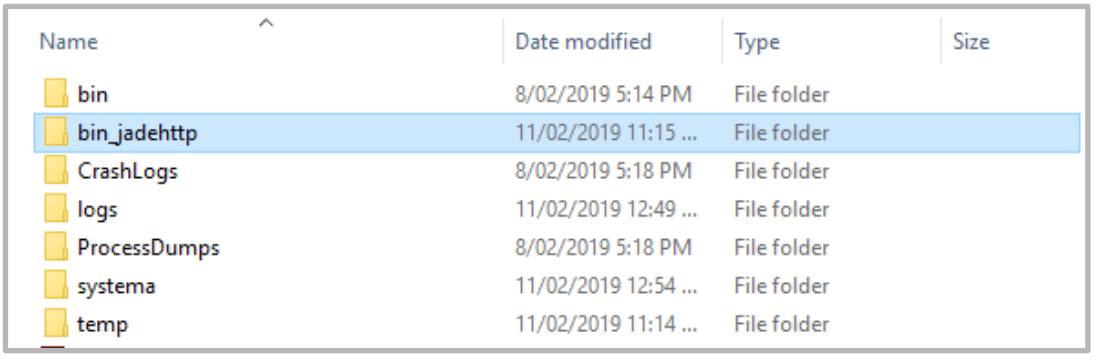
b. **/RestfulJade** means use the handler mappings specified in the **RestfulJade** IIS application; that is, the **jadehttp.dll** specified in the **ISAPI-dll** mapping

c. **/Customer/3** means call the **getCustomer** method with the parameter **3**

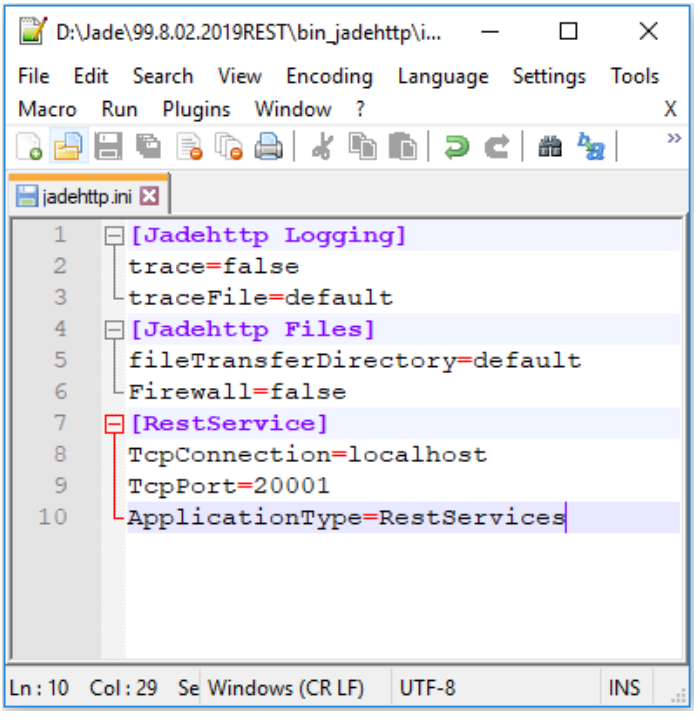
d. **/?RestService** means use the **RestService** JADE application

11. The following error is displayed.

Don't panic. As the **jadehttp.ini** file is not needed except when using JADE as an HTTP server, it is generated only when attempting to use JADE as an HTTP server for the first time. Navigating in File Explorer to your JADE installation folder, you will see the following folder has been created.



12. Open this folder, then edit the **ini** folder within it, adding the following [RestService] section to the **jadehttp.ini** file.



Note The **TcpPort** parameter value can be any unused port, but it must match the port number specified in the **Connection Name** text box on the **Web Options** sheet of the Define Application dialog for the JADE Rest Services application.

13. Reopen the http://localhost/RestfulJade/jadehttp.dll/Customer/3/?RestService URL. It now shows a JSON string representation of the **Customer** object, as follows.

```
{"id":3,"myCompany":null,"name":"Edward Fields"}
```

Note The default representation of objects is JSON. This could also be set explicitly; for example, http://localhost/RestfulJade/jadehttp.dll/Customer/3.json?RestService.

14. Modify the URL in the browser to the following, and then refresh the page.

    http://localhost/RestfulJade/jadehttp.dll/Customer/3.xml?RestService

    This will show the same customer, but this time in XML format rather than in JSON.

Note The **myCompany** reference is null, as the **Customer** object passed to the browser was a clone, and references of clones are not automatically cloned (although it can be set explicitly, if required).

Making REST Requests from JADE

In addition to functioning as a provider of REST web services, JADE can also act as a client.

JADE provides the **JadeHTTPConnection** class, which can send HTTP requests including those of the form required for REST web service providers.

Although the **JadeHTTPConnection** class provides many methods, the one we will use is the **getHttpPage** method, which returns a JSON or XML string containing the REST response and has the following parameters.

Parameter	Description
pVerb	The HTTP verb as a **String**; for example, **GET** or **POST**.
pServerName	The URL of the web server. It should contain the full REST-style URL.
pUrlAddress	Not needed for REST requests, so it can be null.
pUrlAddress	Not needed for REST requests, so it can be null.
pContentType	Not needed for REST requests, so it can be null.

For example, consider the following JadeScript method, which returns the same result as navigating to http://localhost/RestfulJade/jadehttp.dll/Customer/3.xml?RestService in a browser.

```
restfulExample();

vars
    httpConnection  : JadeHTTPConnection;
    restStyleURL    : String;

begin
    create httpConnection transient;
    restStyleURL := "http://localhost/RestfulJade/jadehttp.dll/Customer/3.xml?RestService";
    write httpConnection.getHttpPage("GET", restStyleURL, null, null, null);

epilog
    delete httpConnection;
end;
```

HTTP Request Types in JADE REST Requests

So far, we have only performed **GET** requests through REST web services. JADE REST services are also compatible with the **GET**, **POST**, **PUT**, and **DELETE** HTTP request verbs, as follows.

Verb	Description
GET	Retrieves a representation of a resource from a web server. This representation can be in JSON or XML format. **GET** requests should never modify any of the resources of the web server.
POST	Creates a new resource on the web server. Note that **POST** requests are not idempotent by default; that is, multiple identical **POST** requests create multiple identical resources.
PUT	Updates an existing resource on the web server. Unlike **POST** requests, **PUT** requests are idempotent. As such, multiple PUT identical requests cause only a single change to the web server's resources.
DELETE	Deletes a resource on the web server. Although usually idempotent, repeated identical **DELETE** requests return error 404, as the target resource no longer exists after the first deletion.

Note An *idempotent* request is one where making the request multiple times results in the same behavior as making the request a single time only.

Exercise 3 – Creating a REST Client

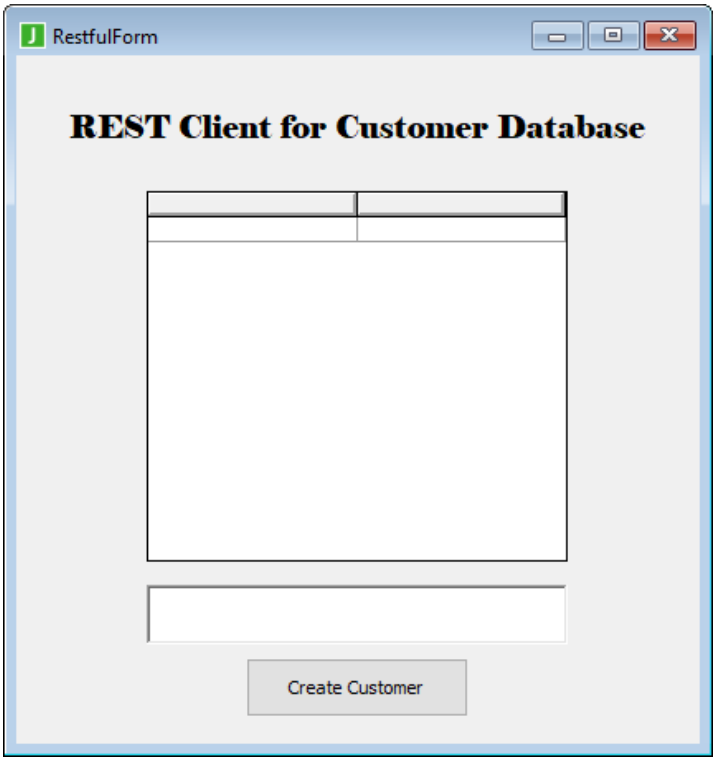
In this exercise, you will create a JADE application that acts a client to the web server created in the previous exercise and uses REST requests to read data from the **RestfulSchema** schema.

1. Create a schema named **RestfulClient**.

   Note As the REST client will not directly reference **RestfulSchema** but uses REST requests to obtain relevant data, it does not need to be in the same database (or even on the same physical machine). However, since we are assuming you do not have multiple JADE databases available, a JADE database can act as a client and a server at the same time for the purposes of this exercise.

2. From the **RestfulClient** schema, open the JADE Painter.

   Create a form called **RestfulForm** and add the following controls.



- A label called **lblTitle** with:
  - Caption: **REST Client for Customer Database**
  - Alignment: **7 - Center - MIDDLE**
  - FontName: **Elephant**
  - FontSize: **14**
- A table called **tblCustomers** with:
  - Columns: **2**
  - fixedColumns: **0**
- A text box called **tbCustomerName**
- A button called **btnCreateCustomer** with:
  - Caption: **Create Customer**

3. The first thing we will do is display the customers from the **RestfulClient** schema in the table. Add a class called **RestQuerier** to the **RestfulClient** schema.

4. Add the following class constants to the **RestQuerier** class.

   - **RestURL**, which is a **String** with definition **"http://localhost/RestfulJade/jadehttp.dll/"**
   - **RestApp**, a **String** with definition **"?RestService"**

Note As this is a **String** definition, the double quotes must be included.

5. Add a method called **getCustomers**, coded as follows, to the **RestQuerier** class.

```
getCustomers(xmlDoc : JadeXMLDocument io) typeMethod;

constants
    RestMethodName  = "AllCustomers";
    ResultType      = ".XML";
vars
    connection : JadeHTTPConnection;
    url : String;
begin
    create connection transient;
    url := RestURL & RestMethodName & ResultType & RestApp;
    xmlDoc.parseString(connection.getHttpPage("GET", url, null, null, null));
epilog
    delete connection;
end;
```

Note As the **RestQuerier** class has only methods and no properties, all methods are *type* methods, which means they can be called without creating a **RestQuerier** object.

The **xmlDoc** parameter is passed as an **io** parameter, so it is modified and returned to the caller (who is responsible for deleting the transient **JadeXMLDocument** when finished with it).

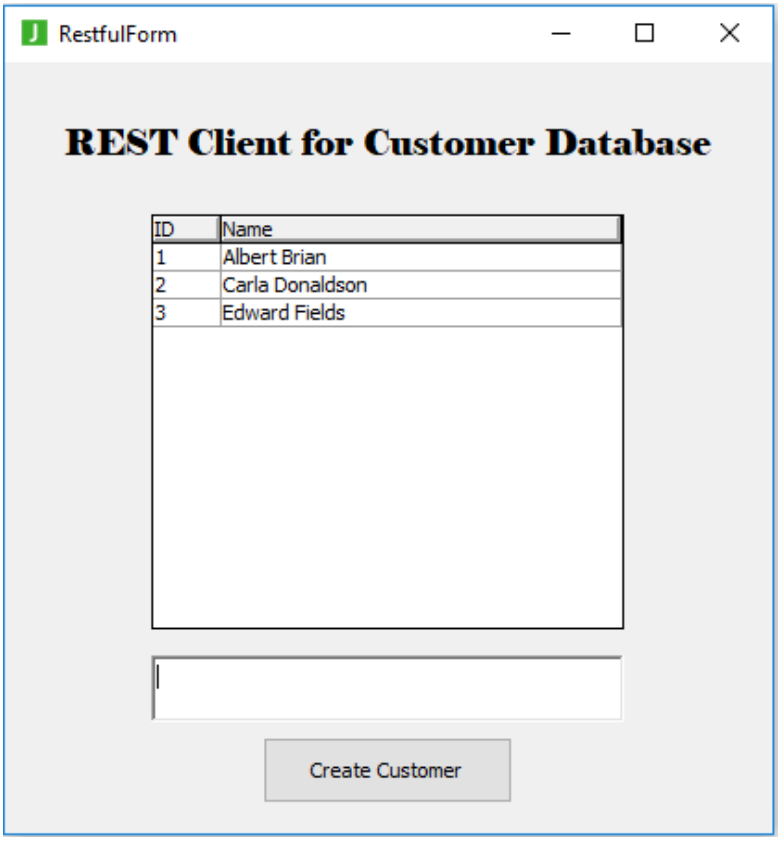
For more details about XML documents in JADE, see the XML module of the JADE Developer's course.

6. On the **RestfulForm** class, modify the Form Events **load** method, as follows.

```
load() updating;

vars
    xmlDoc      : JadeXMLDocument;
    elements    : JadeXMLElementArray;
    element     : JadeXMLElement;
    id          : String;
    name        : String;
begin
    // Sets up the table with correct headers
    self.tblCustomers.rows := 1;
    self.tblCustomers.accessCell(1, 1).text := "ID";
    self.tblCustomers.accessCell(1, 2).text := "Name";
    self.tblCustomers.accessColumn(1).widthPercent := 15;

    create xmlDoc transient;
    // This ClassName@methodName syntax invokes a type method.
    // Note that there is no RestQuerier object instantiated.
    RestQuerier@getCustomers(xmlDoc);

    create elements transient;
    xmlDoc.findElementsByTagName("Customer", elements);

    foreach element in elements do
        id   := element.getElementByTagName("id").textData;
        name := element.getElementByTagName("name").textData;
        tblCustomers.addItem(id & Tab & name );
    endforeach;

epilog
    delete xmlDoc;
    delete elements;
end;
```

7. Run the default **RestfulClient** application.

You should see that the table has been populated with the customers from the **RestfulSchema**.



Tip Ensure that the **RestfulSchema** application is running whenever using the **RestfulClient** application.

Exercise 4 – Making a POST Request over REST

In this exercise, you will enable the form created in the previous exercise to send **POST** requests to the REST web service so that you can add customers to the company defined in **RestfulSchema**.

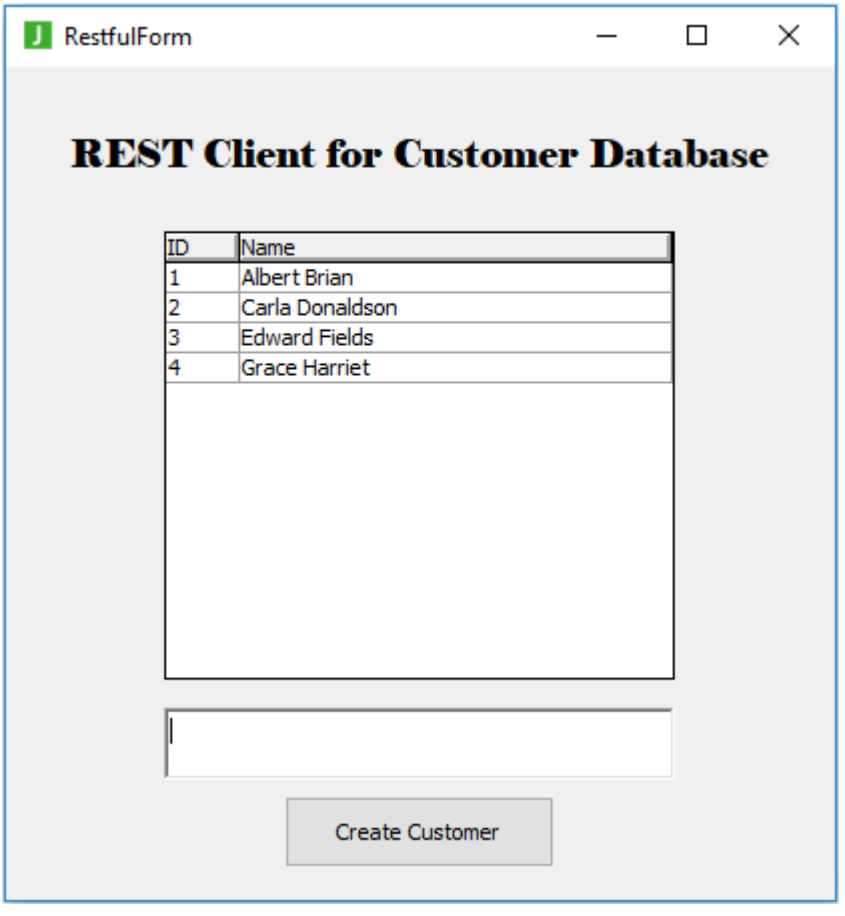
1. Add a *type* method called **postCustomer**, coded as follows, to the **RestQuerier** class of the **RestfulClient** schema.

```
postCustomer(name : String) typeMethod;

constants
    RestMethodName  = "Customer";
vars
    connection : JadeHTTPConnection;
    url : String;
begin
    create connection transient;
    url := RestURL & RestMethodName & "/" & name & RestApp;
    connection.getHttpPage("POST", url, null, null, null);
epilog
    delete connection;
end;
```

2. Modify the **click** method of the **btnCreateCustomer** control in the **RestfulForm** class, as follows.

```
btnCreateCustomer_click(btn: Button input) updating;

vars

begin
    RestQuerier@postCustomer(tbCustomerName.text);
    tbCustomerName.text := "";
    self.load();
end;
```

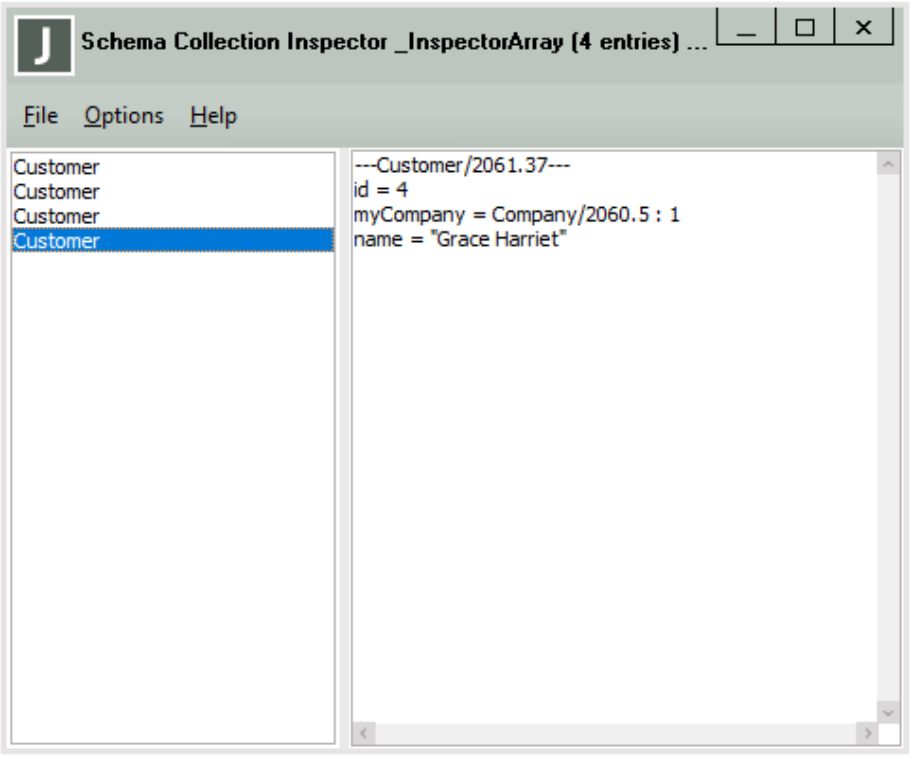
3. Open the **RestfulClient** application, enter **Grace Harriet** in the text box, and then click the **Create Customer** button.



Grace Harriet will be added to the **RestfulSchema** database so that she will appear in the list of customers when the form is automatically reloaded.

4. Open **RestfulSchema** in the Class Browser and then select the **Customer** class. Press CTRL+I, to open the Schema Collection Inspector.

5. Select the fourth **Customer** in the list.



You can see that **Grace Harriet** has been added to the database.