Building a REST service with integrated web services server for IBM i: Part 2

Deploying a simple RESTful application

By Nadir Amra, IBM Software Engineer
Published 08 May 2015 (updated 05 May 2024)

Abstract: Rapidly changing application environments require a flexible mechanism to exchange data between different application tiers. Representational State Transfer (REST) has gained widespread acceptance across the Web as the interface of choice for mobile and interactive applications.

You may already be using integrated web services server to expose ILE programs and service programs as SOAP-based web services. This series of articles introduces a powerful new feature of the integrated web services server – the ability to deploy ILE programs and services programs as RESTful web services. In this second installment, you will learn how to deploy a simple application as a RESTful web service.

Introduction

For several years now IBM i users have had the ability to deploy ILE programs and services programs as web services based on the SOAP protocol using the integrated web services server support that is part of the operating system. REST web services were not supported by the integrated web services server, until now.

This article is the second in a series of articles about the integrated web services server REST support. Future installments in this series will build upon the basic concepts:

- Part one starts out by explaining the basic concepts behind REST web services and how the integrated web services server supports REST services.
- In part two, we will take you through the steps of deploying a simple ILE application as a RESTful web service.
- Part three will take you through the steps of deploying a more complex ILE application that uses more of the REST features.

Prerequisites

Software

To get all the PTFs required by the integrated web services server in support of REST, you will need to load the latest HTTP Group PTF. The IBM Support web page *IBM i Group PTFs with level* lists the HTTP group PTFs for each of the supported releases of the IBM i operating system.

Note: The steps in this article were performed on IBM i 7.3. Panels may look different if you are on an older or newer release. And if you are on an older release, some features discussed in this article may be unsupported on that release.

Assumptions

Before reading this article, you should have read part one of the article in the series to have a basic understanding of REST principles and the terminology used.

A simple RESTful application

The example we will use in this discussion is an application written in ILE RPG that converts the temperature from Fahrenheit to Celsius. The application is packaged within the service program QIWSSAMPLE in library QSYSDIR. This service program is shipped as part of integrated web services server support and contains one exported procedure, CONVERTTEMP.

The source (see Listing 1) for the procedure can be found at the following path:
`/QIBM/ProdData/OS/WebServices/samples/server/ConvertTemp/CNVRTTMP.RPGLE`

Listing 1. RPG Source for simple REST application

```
     h nomain PGMINFO(*PCML:*MODULE)
```

```
d ConvertTemp     pr
d  tempIn                       10    const
d  tempOut                      10

p ConvertTemp     b                   export
d  ConvertTemp    pi
d  tempIn                       10    const
d  tempOut                      10

d tempI           s              8P 2
d tempO           s              8P 2
d value           S             50A
 /free
  value = %STR(%ADDR(tempIn));
  tempI=%DEC(value:7:2);
  tempO = (5/9)*(tempI - 32);
  value = %CHAR(tempO);
  tempOut = value;
  %STR(%ADDR(tempOut):10)=tempOut;
 /end-free
p ConvertTemp     e
```

The RESTful web service to be deployed is very simple. The only thing that is needed by the application is the temperature that is to be converted from Fahrenheit to Celsius. And what is returned is the temperature in Celsius.

Things to get done before deployment

It is always a good idea to sit down and figure out things before deploying a RESTful web service. When deploying a RESTful web service, you should have answers to the following questions at the bare minimum:

1. What HTTP methods will the resource support?
2. How do I want the URIs to look like?
3. What incoming content types should be supported?
4. What type of data should be returned?

Let us quickly go through these basic questions in the context of the simple application that will be deployed.

What HTTP methods will the resource support?

The supported HTTP methods are GET, POST, PUT, PATCH, and DELETE. Which HTTP methods you choose to support will affect how a client will send data. For example, if we wanted a resource method to receive XML or JSON documents, then we would probably bind the resource method to either the POST or PUT HTTP methods.

For the simple application that we will be deploying, the only input that is expected is the temperature that is to be converted, and this can be handled by having the client pass in the temperature as part of the URL. So, there is no payload that a client needs to send, and thus the HTTP method will be GET.

Since we only have one procedure (resource method), we are done.

How do I want the URIs to look like?

REST services are based on manipulating resources. Resources for RESTful services are addressable, and URLs are the primary way of achieving addressability in REST. The design of REST URIs is an art in itself and beyond the scope of this article. What I will say is simplicity and consistency is key.

For the temperature conversion example, we want the URI to look like the following:

`<context-root>/ftoc/{temp}`

Where *{temp}* is the temperature in Fahrenheit that is to be converted to Celsius. For example, if we wanted to know what the temperature 123 Fahrenheit is in Celsius, the URI that would be sent by a client would be as follows:

```
/web/services/ftoc/123
```

Note: The above URI assumes the default context root (`/web/services`) was not changed for the server.

What incoming content types should be supported?

Since there is no payload with the HTTP GET request, we really do not care what the incoming content-type is. In our example, we left the default, and that is to accept all content-types.

What type of data should be returned?

We have 3 choices. We can return XML, JSON, or both depending on what the client is willing to accept. In this example we will return JSON.

Now we are ready to deploy the simple application as a RESTful web service.

Step 1. Create the integrated web services server

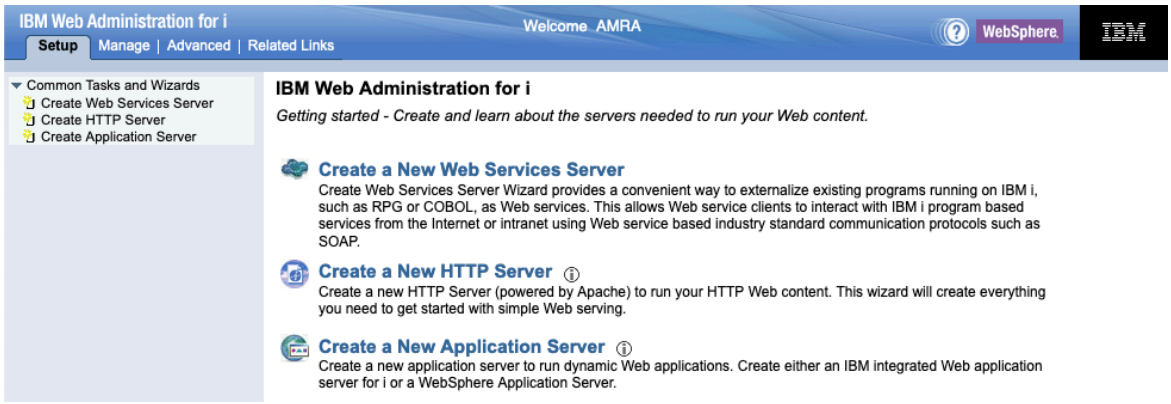
To deploy an ILE program object as a REST service, you need to have an integrated web services server created, and it must be version 2.6 or greater. If you have one already created, you can skip this section and go to the section titled “Deploy the ILE application as a REST web service”.

Nothing has changed as far as the steps to create an integrated web services server. The server can contain both SOAP and REST web services.

To launch the web services server wizard, you need to sign on to the Web Administration GUI for IBM i and click on the **Create Web Services Server** wizard link. Point your browser to the Web Admin GUI for IBM i by specifying the following URL:

http://hostname:2001/HTTPAdmin, where hostname is the host name of your server (note that if SSL has been configured for the Web administration server the URL would be https://hostname:2010/HTTPAdmin) and sign on. You must have *ALLOBJ and *IOSYSCFG special authorities to create a web services server, or, if you are on IBM i 6.1 or newer release, you must have been given permission to create web services servers. Launch the Create Web Services Server wizard by either clicking on the link in the navigation bar under the **Common Tasks and Wizards** heading, or on the main page of the **Setup** tab (see Figure 1 below).

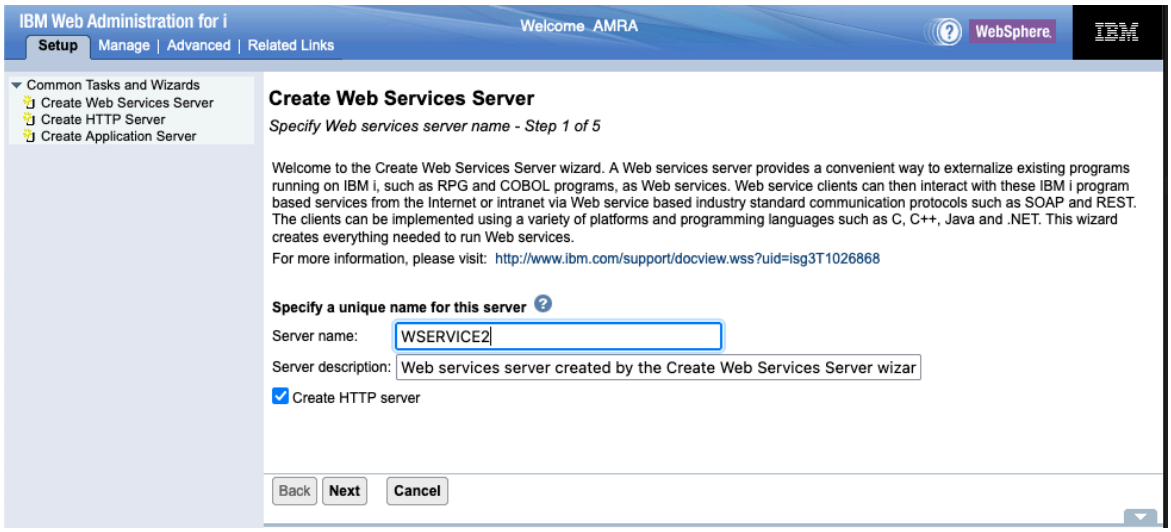
Figure 1. Links to Create Web Services Server



Step 1-1. Specify web services server name

You have the option of naming (see Figure 2) the web services server that is to be created. You can also provide a short description if you so choose. By default, an HTTP server associated with the integrated web services server is created. Deselect the option to create an HTTP server if you do not want an HTTP server associated with the integrated web services server.

Figure 2. Specify web services server name

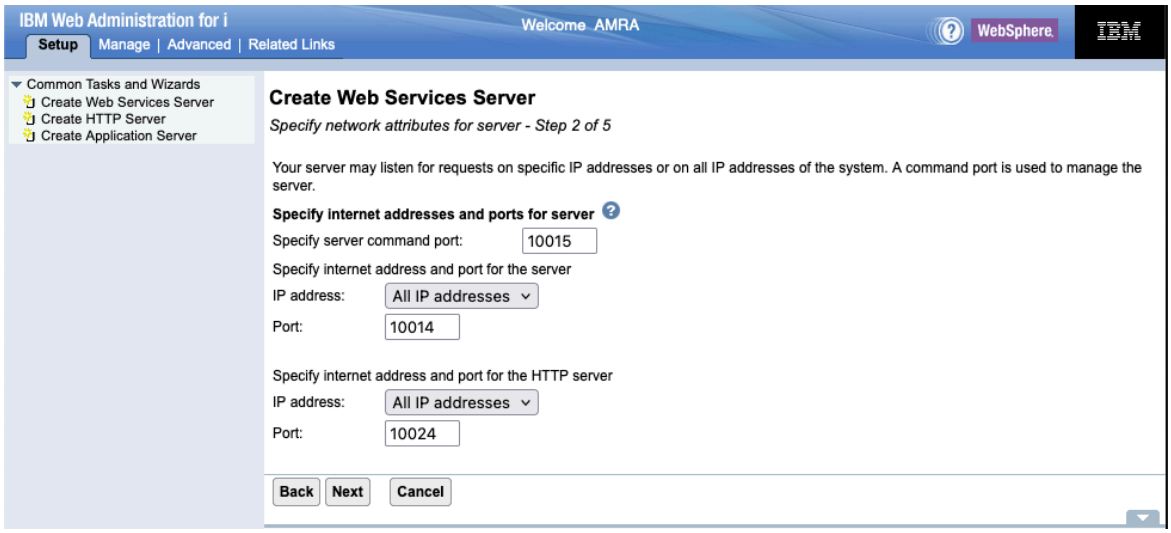


Accept the defaults and click on the **Next** button at the bottom of the form.

Step 1-2. Specify network attributes for server

Specify the IP addresses and ports for the server. The Web Admin GUI attempts to select ports that are unused (see figure 3). You can change the ports. For this example, we will use the default ports chosen.

Figure 3. Specify network attributes for the server

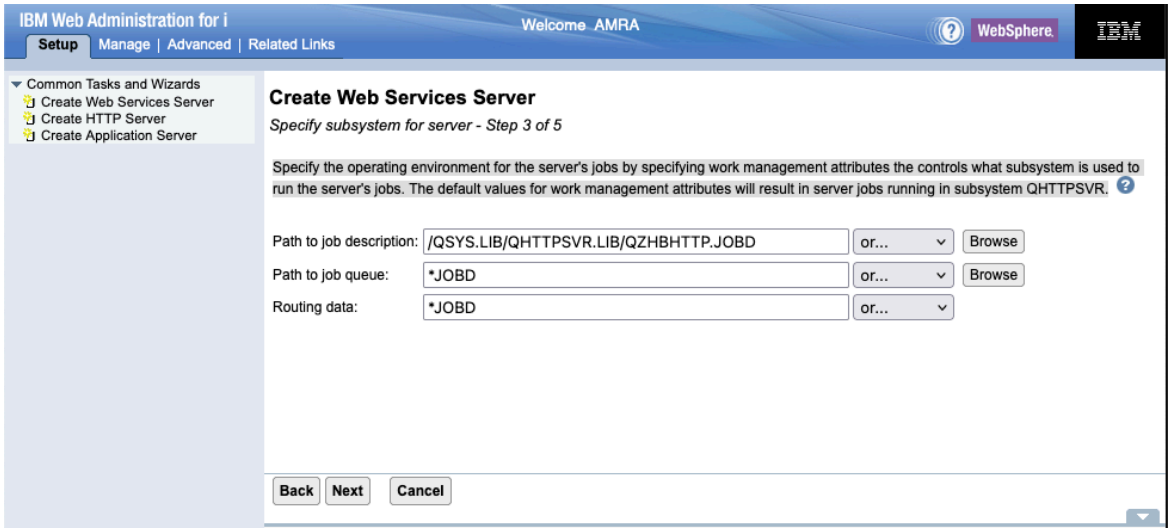


Click on the **Next** button at the bottom of the form.

Step 1-3. Specify subsystem for server

Specify the operating environment for the server's jobs by specifying work management attributes the controls what subsystem is used to run the server's jobs. The default values for work management attributes will result in server jobs running in subsystem QHTTPSVR (see Figure 4).

Figure 4. Specify subsystem for server



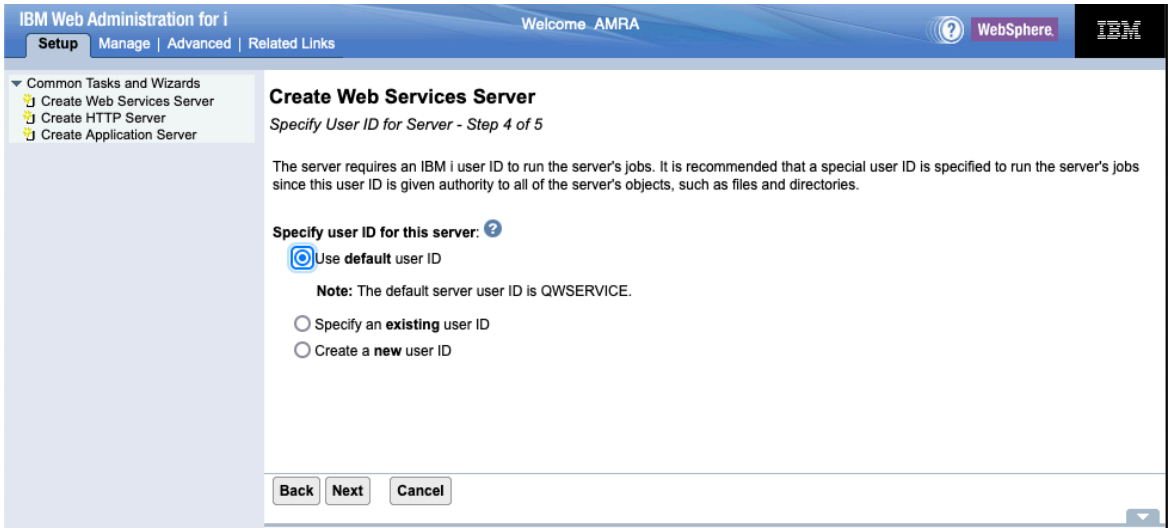
Click on the **Next** button at the bottom of the form.

Step 1-4. Specify server user ID

Specify the user ID to run the jobs associated with the server. You have the option of specifying an existing user ID, creating a new user ID, or using the default user ID. We will use the default user ID, QWSERVICE.

Note: Any user ID specified for the server must be enabled and the password set to a value other than *NONE. Ensure this is true for the specified user ID.

Figure 5. Specify user ID for the server

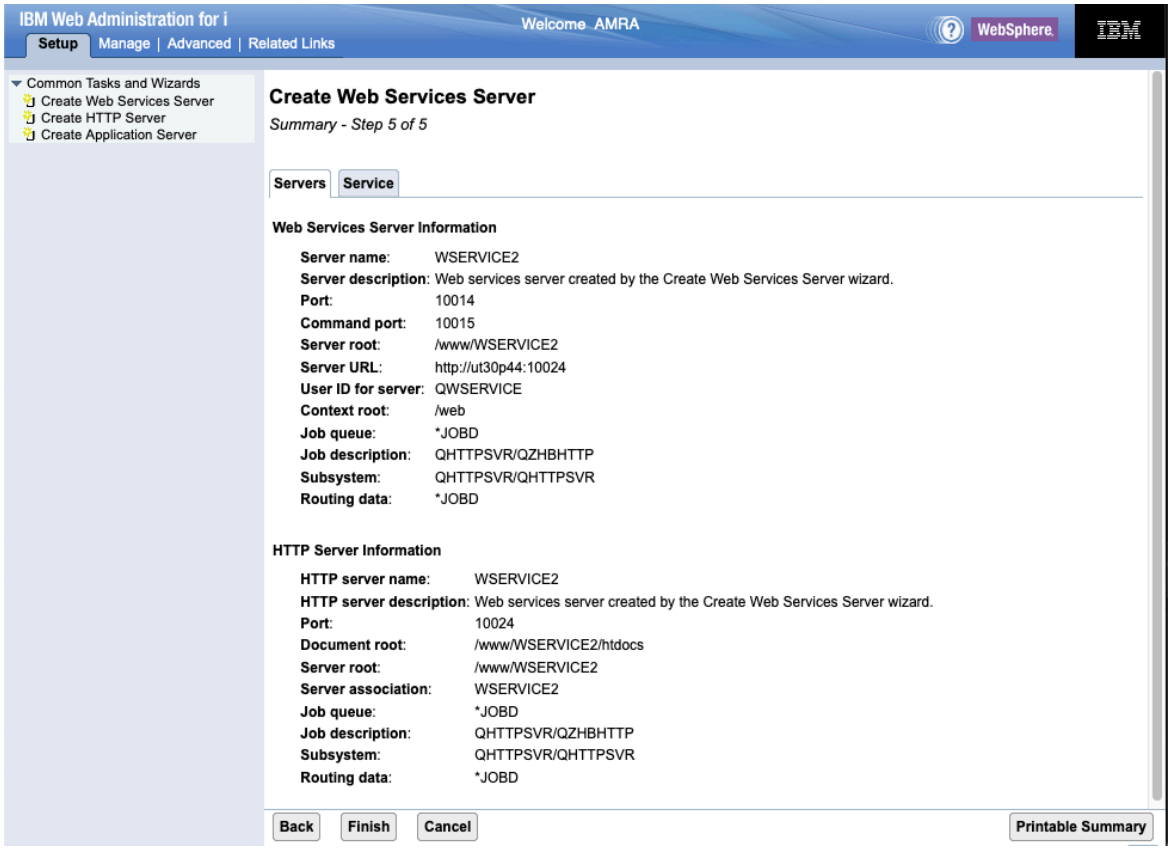


Click on the **Next** button at the bottom of the form.

Step 1-5. Summary

The wizard shows you a summary page (see Figure 6), giving you the chance to see the details relating to the web services server before it starts the task of creating the server.

Figure 6. Server creation summary



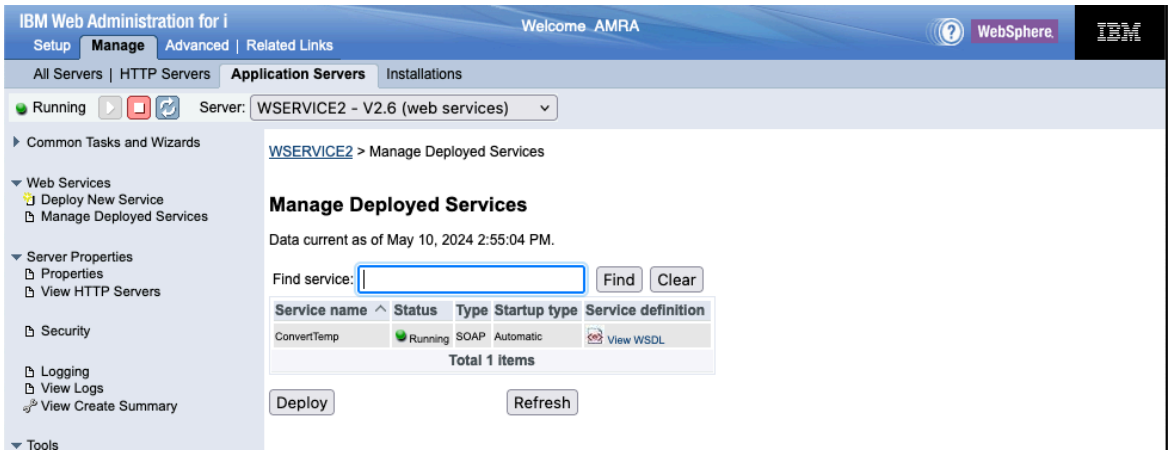
Clicking on the **Finish** button at the bottom of the summary page will kick off the creation of the server. After the server is created, the wizard will start the web services server and HTTP server. If all goes well, you will eventually see the server in **Running** as shown in Figure 7.

Figure 7. Server running

Congratulations, you have now successfully created an integrated web services server. If you click on **Manage Deployed Services** button, you will see the state and the deployed

services (a sample web service that is shipped with the server) active (green dot to the left of service name) as shown in in Figure 8.

Figure 8. Deployed services



The next steps will guide you through deploying your first ILE program object as a RESTful web service.

Step 2. Deploy the ILE application as a RESTful web service

The following table summarizes the various details of the RESTful web service that we will be deploying:

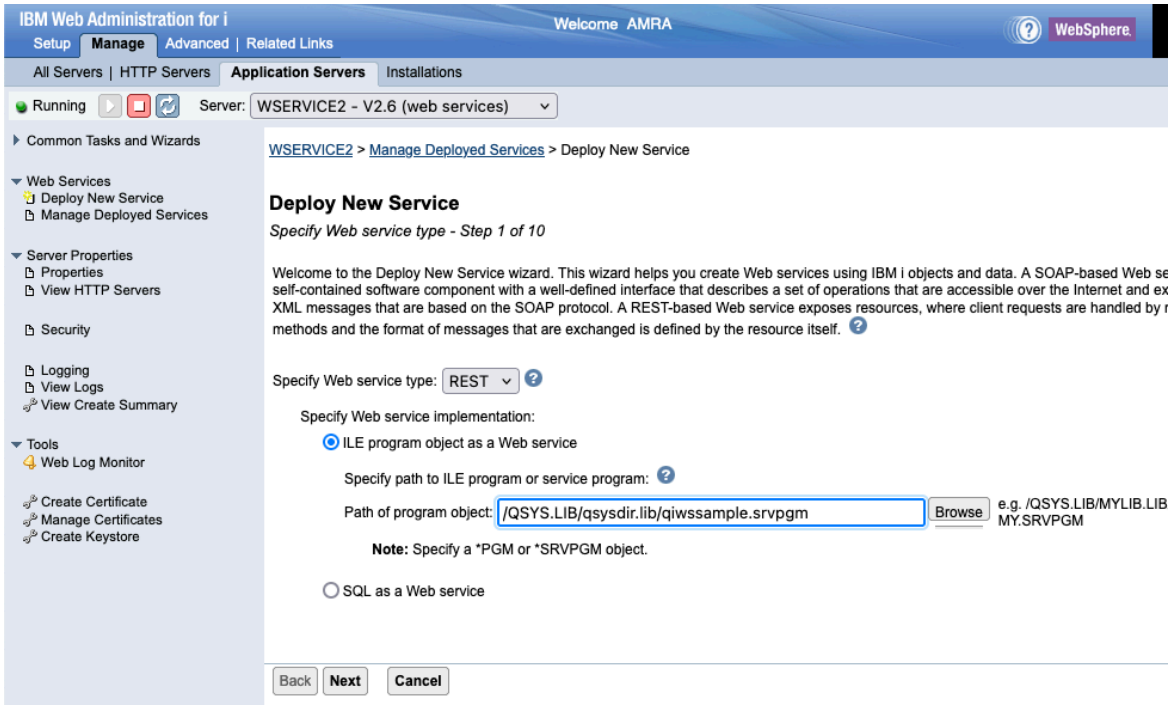
Table 1. REST information for RESTful web service

Procedure	CONVERTTEMP
URI	`../ftoc/{temp}`
HTTP method	GET
Query string	ignored
Request body	ignored
Response code	200 OK
Response body	JSON

Step 2-1. Deploy an IBM i program object as a web service

Click on the **Deploy New Service** wizard link that is in the navigation bar. You should see the panel in Figure 9.

Figure 9. Deploy web service – step 1



The panel gives you the option to either deploy a SOAP or REST web service. Since we are deploying a REST web service, we have selected the REST radio button. For REST, you have the option to deploy an ILE program or service program, or SQL statements as REST APIs. Since we are deploying an ILE service program, we specify an ILE program object name path from which the web service is generated. The path to the program object is /QSYS.LIB/QSYSDIR.LIB/QIWSSAMPLE.SRVPGM.

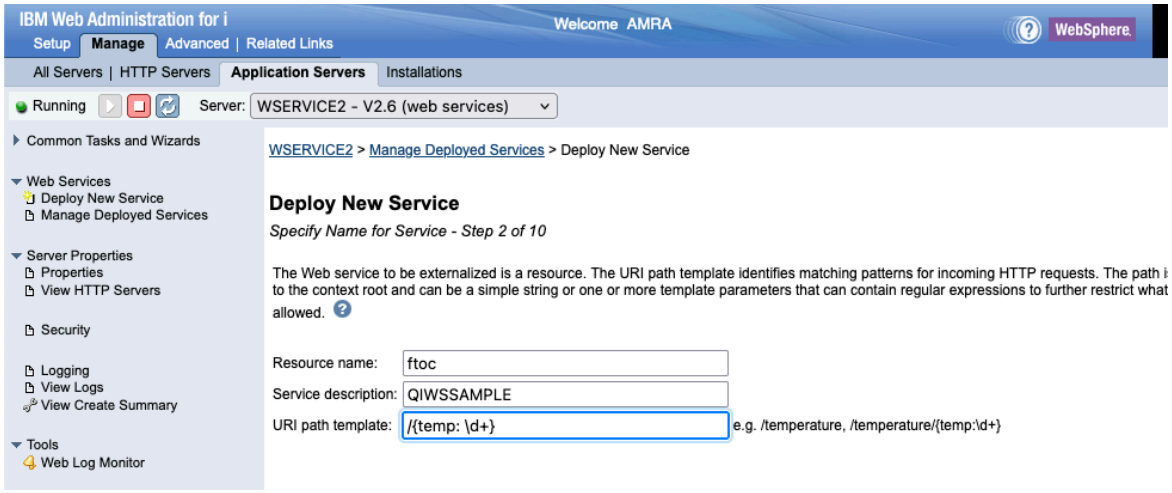
Note that there is two ways to locate the program object on the system. The default way is to specify the path to the program object. Another way is to search for the program object by browsing the integrated file system (IFS), which could take a while if a directory is specified that contains a lot of objects, such as /QSYS.LIB.

Click on the **Next** button at bottom of form.

Step 2-2. Specify name for the resource (web service)

Now we need to give the web service (i.e. resource) a meaningful service name and description. By default, the service name and description are set to the name of the selected program object (see Figure 10).

Figure 10. Deploy web service – step 2



The resource name has been changed to `ftoc`. In addition, you can set a URI path template for the resource. For this example, we have specified a URI path template for the resource that has a variable named `temp` and a regular expression that limits the value that can be specified for the variable to digits. This matches what we wanted the URI to look like, which is of the following form:

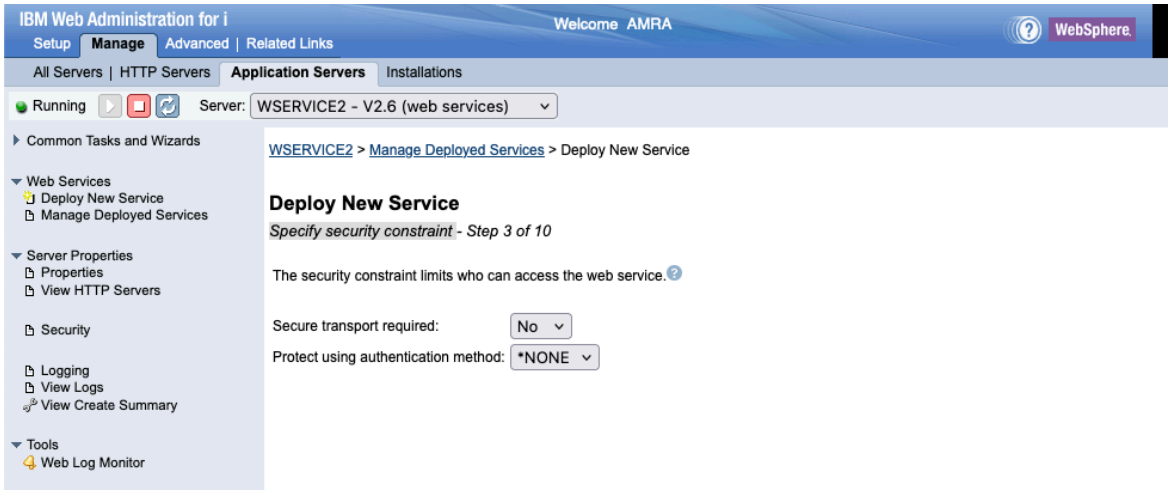
```
../ftoc/{temp}
```

Click on the **Next** button at bottom of form.

Step 2-3. Specify security constraint

The security constraint limits who can access the web service. To protect the web service, an authentication method other than *NONE needs to be specified (see Figure 11). If the web service is protected and roles have been defined, you will have the option to indicate what roles are authorized to the web service. If roles have not been defined, then all authenticated users are allowed access to the web service.

Figure 11. Deploy web service – step 3

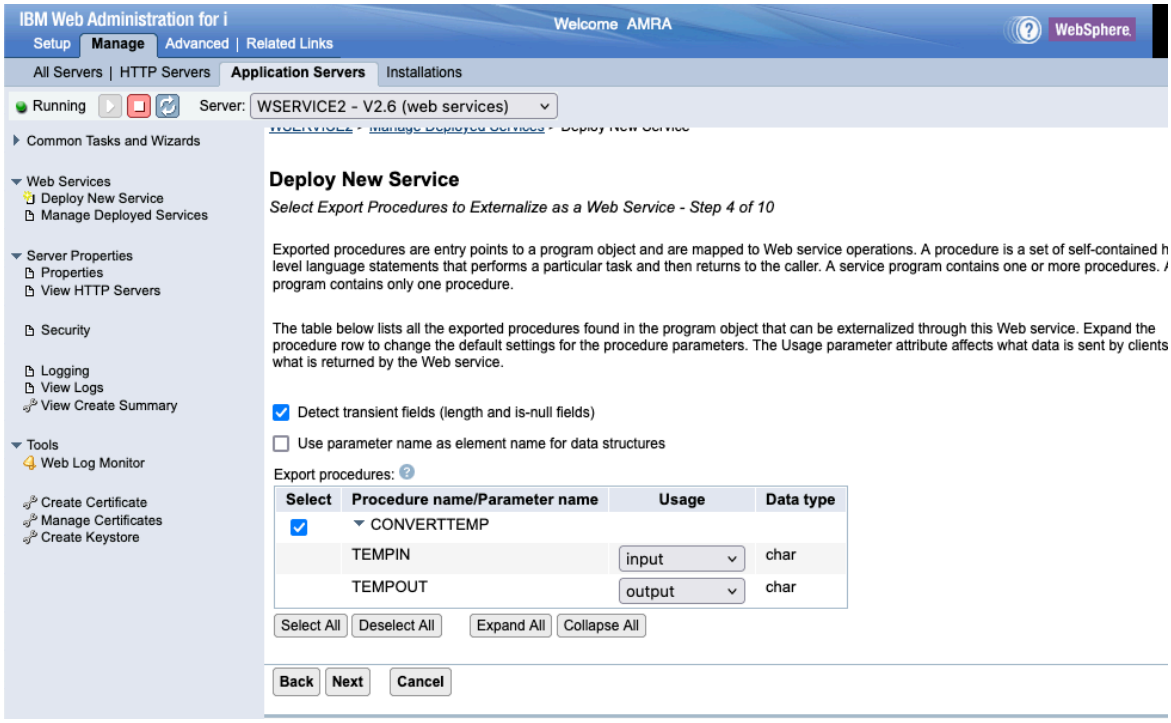


The security constraint panel is beyond the scope of this article. We accept the default values and click on the **Next** button at bottom of form.

Step 2-4. Select export procedures to externalize as resource methods

The wizard will show a list of exported procedures as shown in Figure 12. For service programs (object type of *SRVPGM), there may be one or more procedures. For programs (object type of *PGM), there is only one procedure, which is the main entry point to the program. Expanding the procedure row shows the parameters for the procedure and various parameter attributes.

Figure 12. Deploy web service – step 4



The parameter attributes are modifiable. In most cases you want to modify the parameter attributes to control what data is to be sent by web service clients and what data is to be returned in the responses to the client requests.

In this example, the TEMPIN parameter is an input parameter, and the TEMPOUT parameter is the output parameter. This means that a web service client will need to only pass data corresponding to the TEMPIN parameter, and the response to the client request will be returned in the TEMPOUT parameter.

Click on the **Next** button at bottom of form.

Parameter Case

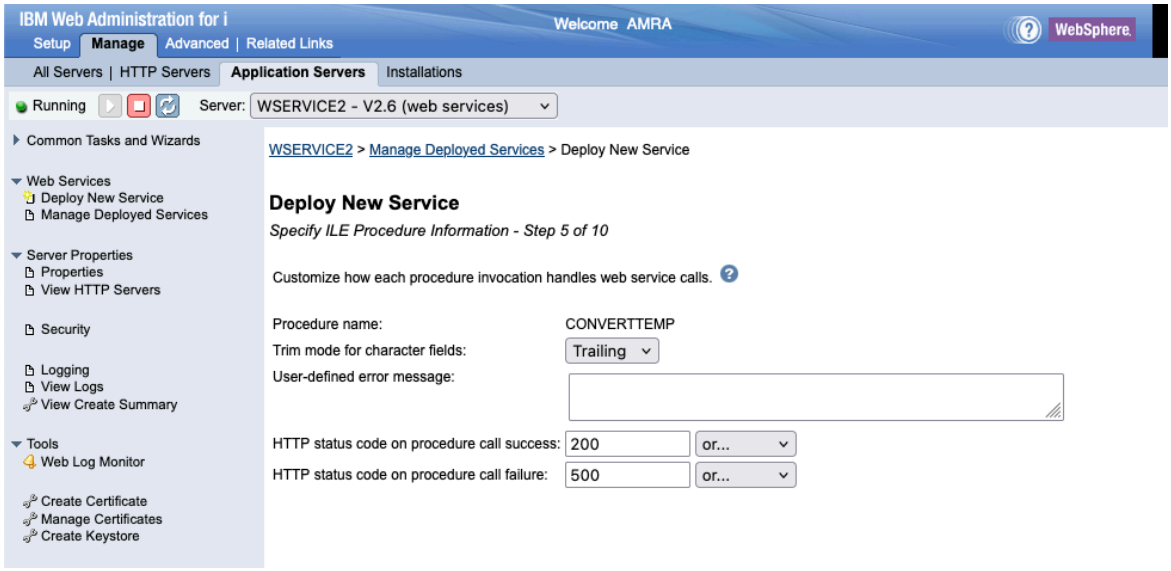
A new RPG enhancement has been released for IBM i releases 7.1 and 7.2 that allows you to control the identifier case of parameter names. See the following PTFs for details:

SI55531 7.2
SI55442 7.2
SI55340 7.1

Step 2-5. Specify ILE procedure information

This panel (Figure 13) is shown for each procedure to be deployed as a web service and allows you to indicate how each procedure invocation handles web service calls**.**

Figure 13. Deploy web service – step 5



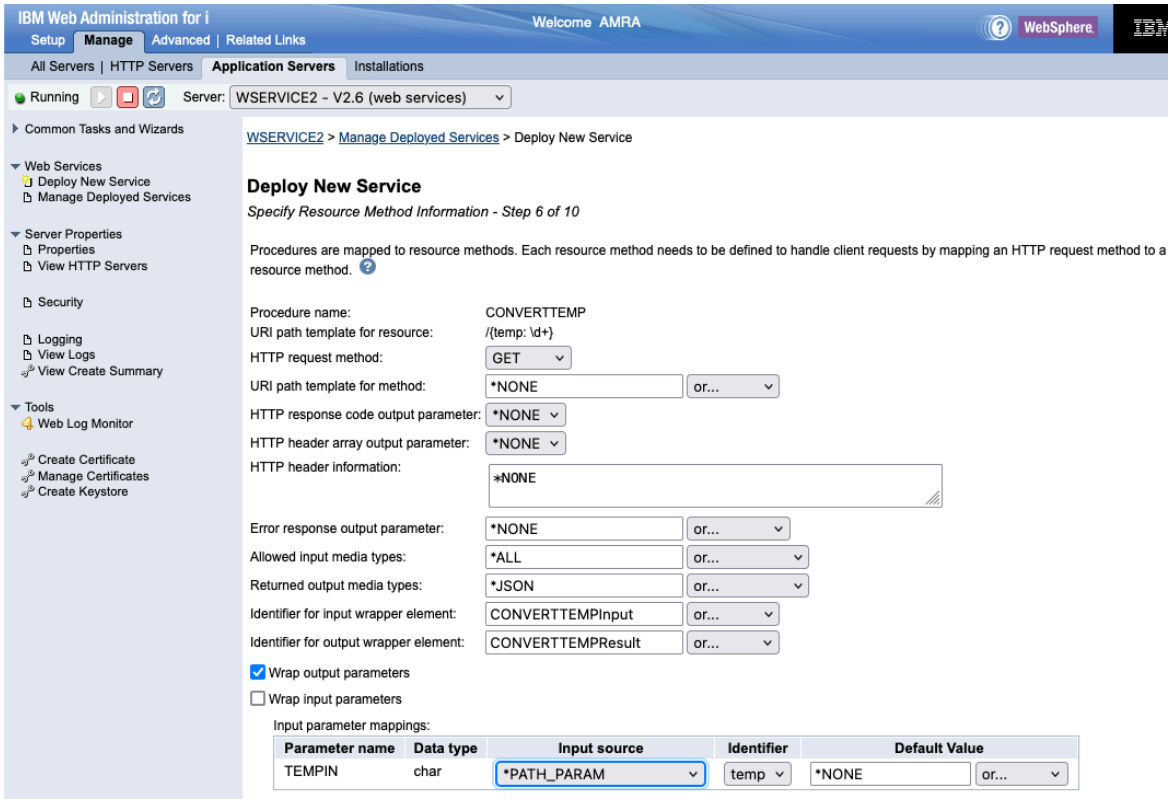
- **Trim mode for character fields**: Specify whether string data will have leading and/or trailing blanks removed.
    - **Trailing** indicates that trailing blanks should be removed.
    - **Leading** indicates that leading blanks should be removed.
    - **Both** indicates that leading and trailing blanks should be removed.
    - **None** indicates that leading and trailing blanks should not be removed.
- **User-defined error message**: Specify the error message that will be returned if an unexpected exception occurs. This message will replace the actual message returned by the operating system.
- **HTTP status code on procedure call success**: Specifies the HTTP status code that will be returned on a web service call that has run successfully. Note that if your program object returns the HTTP status code, it will override the value specified here.
- **HTTP status code on procedure call failure**: Specifies the HTTP status code that will be returned on a web service call that failed to run successfully.

Accept the defaults and click on the **Next** button at bottom of form.

Step 2-6. Specify resource method information

This panel (Figure 14) is used to specify various REST attributes on a per procedure basis.

Figure 14. Deploy web service – step 6



The first two lines are the procedure name and URI path template for the resource, respectively. We have chosen to bind the resource method (i.e. procedure) to the HTTP request method of GET. We did not specify a URI path template for the resource method and thus `*NONE` is specified. Similarly, there is no output parameter that will contain HTTP header data or the HTTP response code, so we specify *NONE for those fields. We are also not returning any hard-coded HTTP headers in the response, so *NONE is specified.

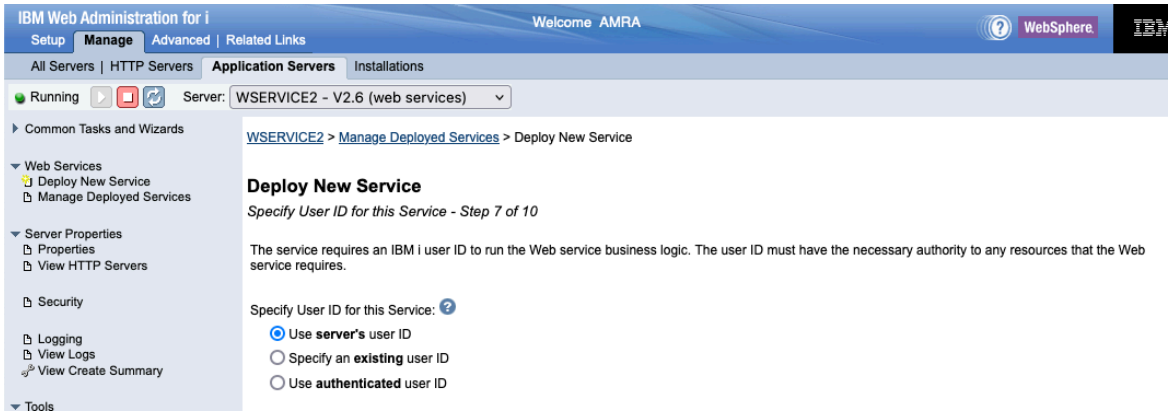
Because the REST API is bound to the GET HTTP method for which there is no payload, and thus no content type, the allowed input media types is set to `*ALL`. The REST web service will return JSON data, so we specify JSON.

Finally, we have chosen to unwrap the parameters so that we can inject a value into the parameter TEMPIN from a value in the URI. We specify the URI path template variable `temp` that was defined in the URI path template for the resource.

Step 2-7. Specify user ID for this service

We now need to specify the user ID that the service will run under. As shown in Figure 15, you can run the service under the server's user ID, or you can specify an existing user ID that the service will run under.

Figure 15. Deploy web service – step 7



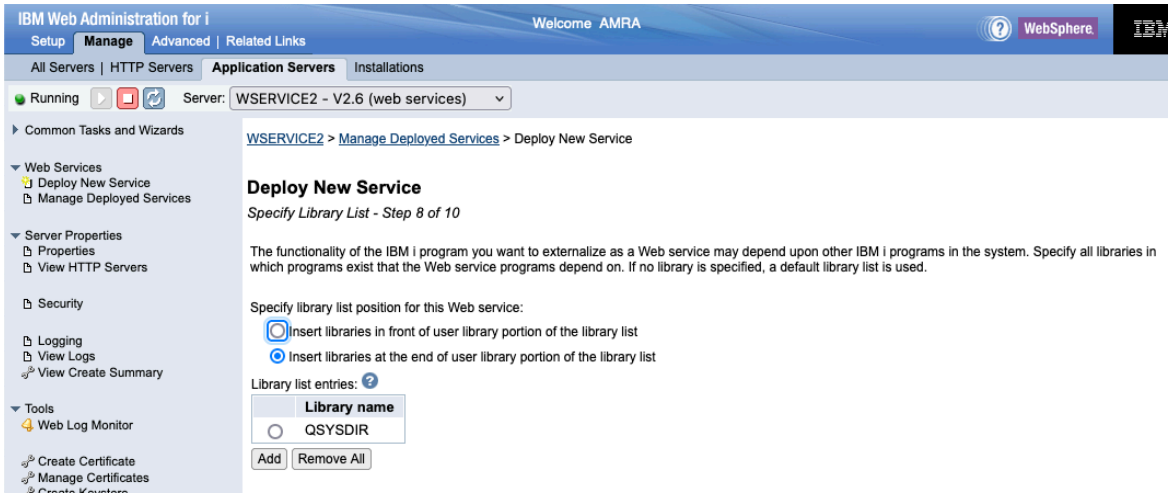
For the web service to run correctly, the user ID status must be set to *ENABLED and the password must be set to a value other than *NONE. If a user ID is specified that is disabled or has a password of *NONE, a warning message is displayed, and the service may not run correctly. In addition, ensure that the specified user ID has the proper authorities to any resources and objects that the program object needs, such as libraries, databases, and files.

In this example, we will accept the default. Click on the **Next** button of the form.

Step 2-8. Specify library list

Specify any libraries that the program object needs to function properly (see Figure 16).

Figure 16. Deploy web service – step 8

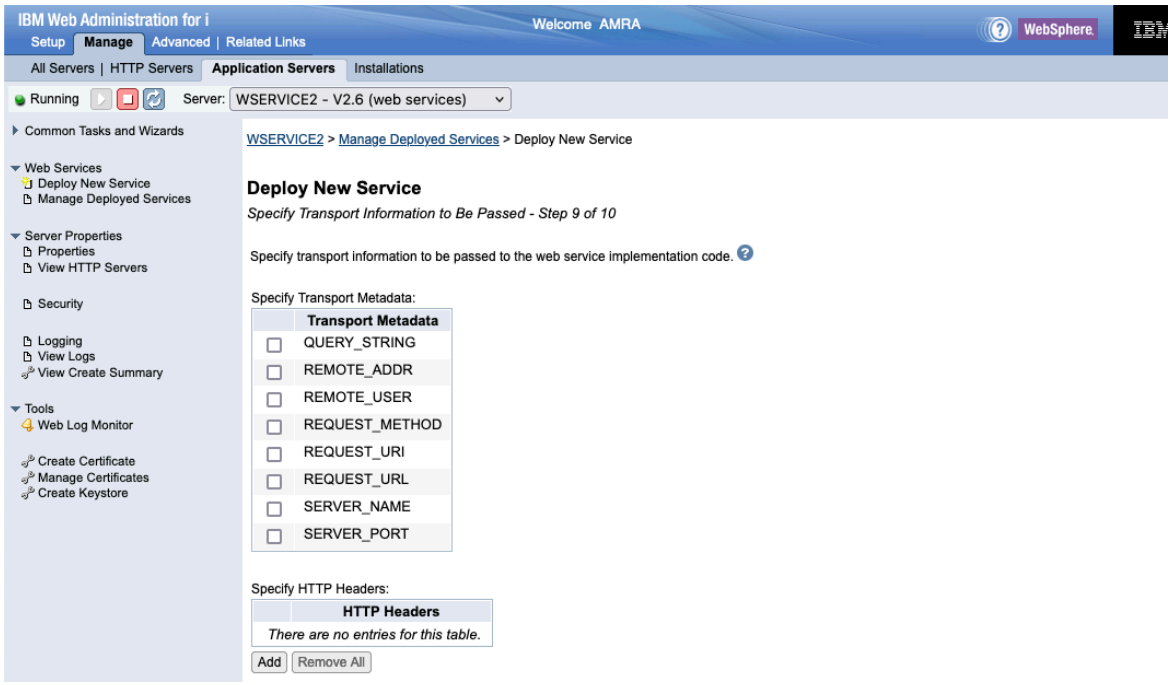


You have the option of putting the libraries at the start of the user portion of the library list or at the end of the user portion of the library list. Click on the **Next** button of the form.

Step 2-9. Specify transport information to be passed

Specify what transport information related to the client request is to be passed to the web service implementation code (see Figure 17). The information is passed as environment variables.

Figure 17. Deploy web service – step 9



For example, the transport metadata REMOTE_ADDR is passed to the web service implementation code in an environment variable named REMOTE_ADDR.

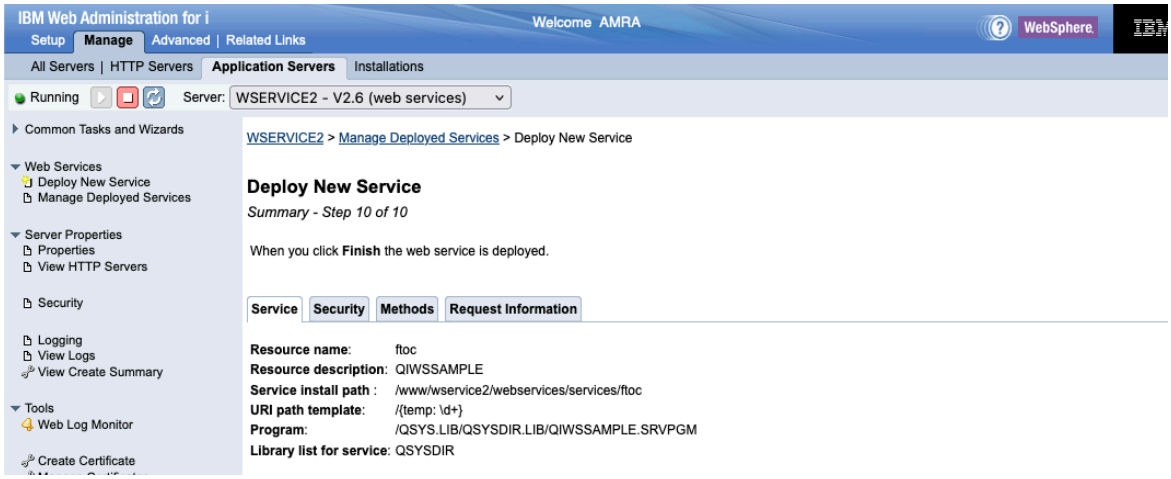
HTTP headers indicates what transport headers (e.g. HTTP headers) to pass to the web service implementation code. Transport headers are passed as environment variables. The environment variable name for HTTP headers is made up of the specified HTTP header prefixed with '`HTTP_`', all upper-cased. For example, if '`Content-type`' is specified, then the environment variable name would be '`HTTP_CONTENT-TYPE`'. If an HTTP header was not passed in on the web service request, the environment variable value will be set to the null string.

Click on the **Next** button of the form.

Step 2-10. Deploy web service – step 10

The web service deployment wizard shows you a summary page (see Figure 18), giving you a chance to see the details relating to the web service being deployed.

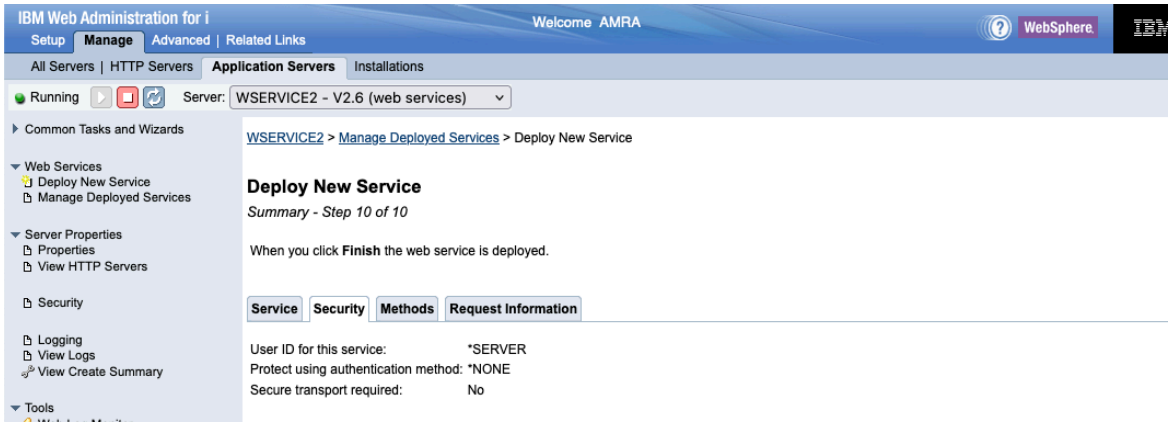
Figure 18. Deploy web service – step 10 (Summary – Services tab)



On the **Services** tab, you will see information about the service being deployed.

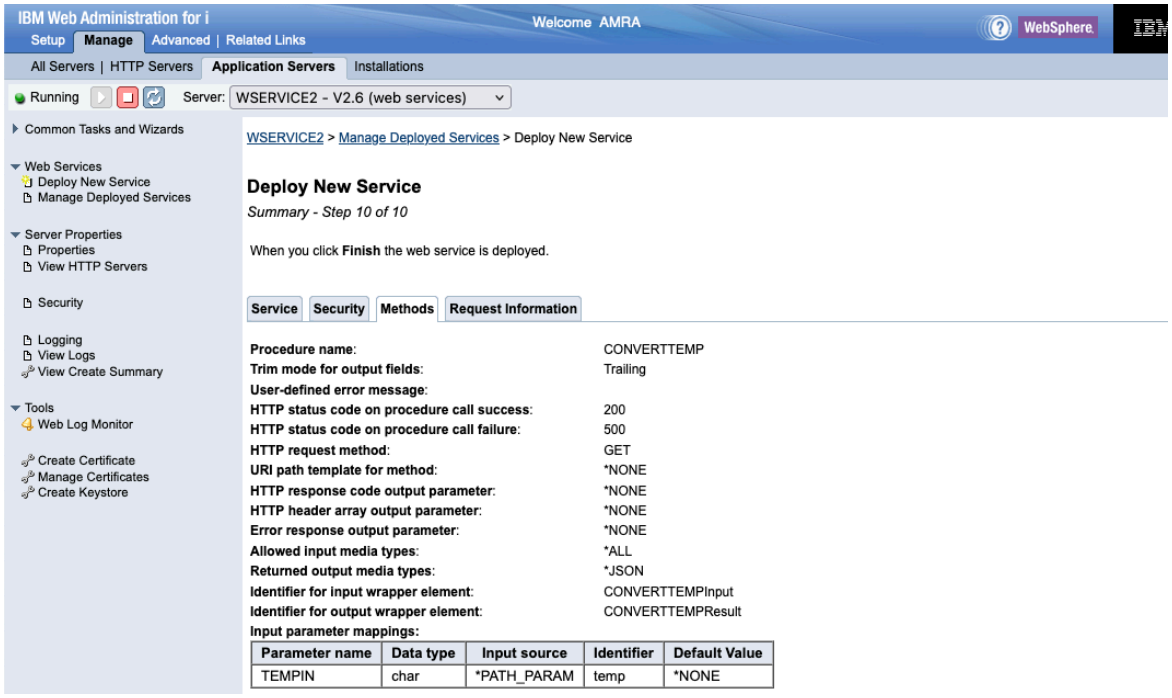
If you click on the **Security** tab, you will see security related attributes and constraints for the web service (see Figure 19).

Figure 19. Deploy web service – step 10 (Summary – Security tab)



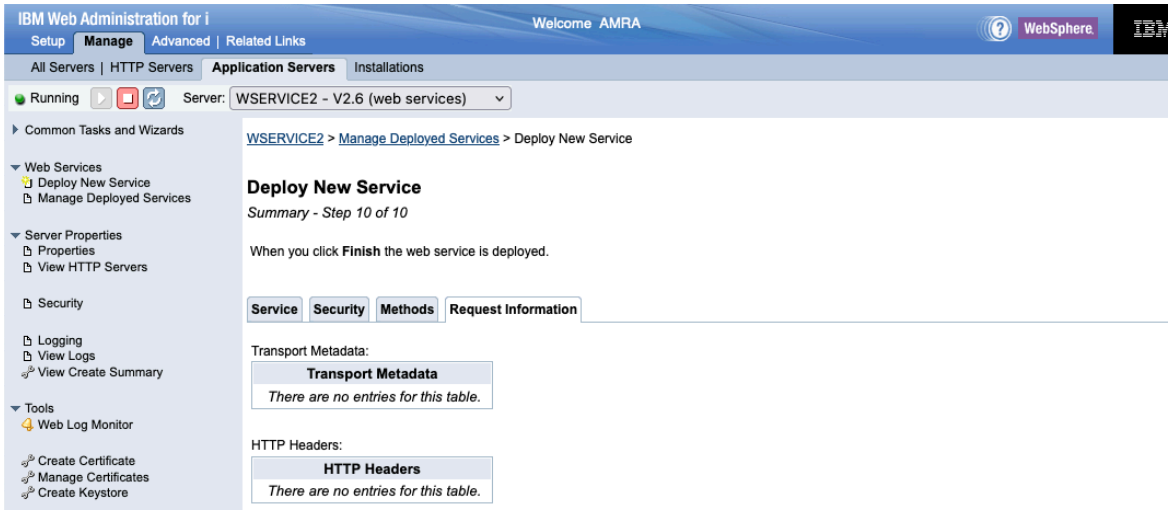
If you click on the **Methods** tab, you will see the resource methods that correspond to the procedures that were selected to be deployed (see Figure 20).

Figure 20. Deploy web service – step 10 (Methods tab)



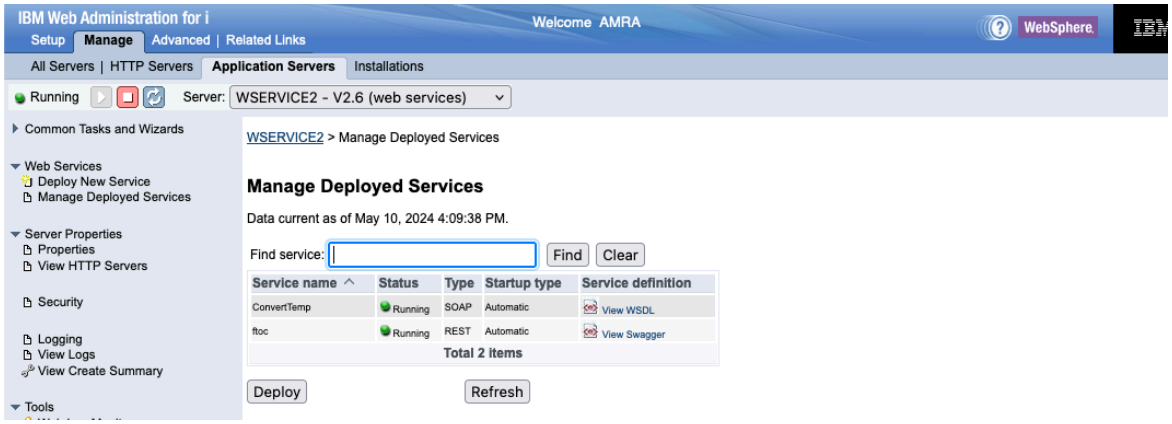
If you click on the **Request Information** tab, you will see the transport information to be passed to the web service implementation code (see Figure 21).

Figure 21. Deploy web service – step 10 (Request Information tab)



Clicking on the **Finish** button at the bottom of the summary page will kick off the installation process. When the web service is deployed the deployed service becomes active (green dot to the left of service name) as in Figure 22:

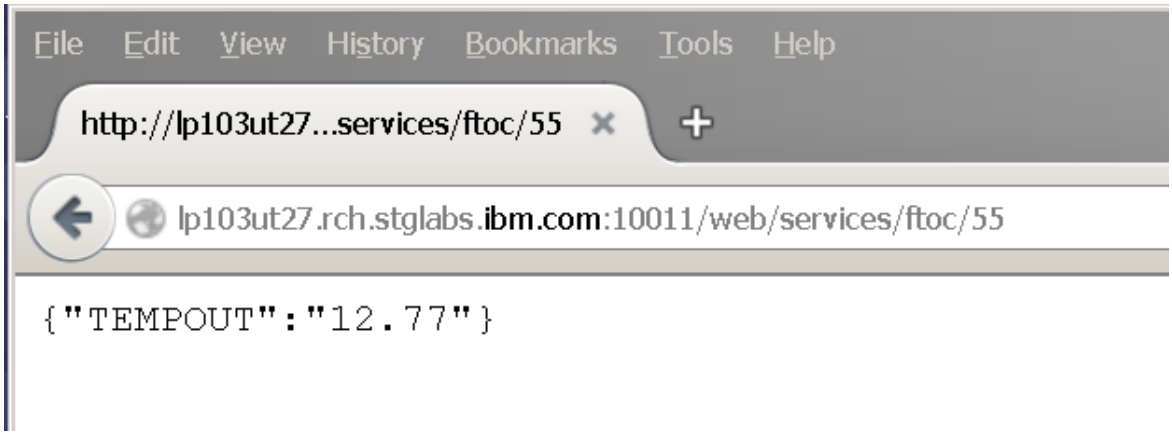
Figure 22. Successfully deployed RESTful web service



Congratulations, you have now successfully deployed your first ILE program object as a RESTful web service.

Since the web service that we deployed has a resource method that is bound to the HTTP GET request method, we can use any browser to quickly test the service (for services that use other HTTP request methods, an external tool needs to be used, such as SoapUI). Figure 23 shows the result of the request:

Figure 23. Testing web service



Summary

In part one of this series, you learn the basic concepts behind REST web services and how the integrated web services server supports REST services. In this article, you learned how to deploy a simple ILE application as a RESTful web service.

Part three will take you through the steps of deploying a more complex ILE application that uses more of the REST features.

Resources

- For everything about the integrated web services support on IBM i see the product web page.
- Parts one and three of the series can be found on the integrated web services web site.