



# BCPii - A RESTed development

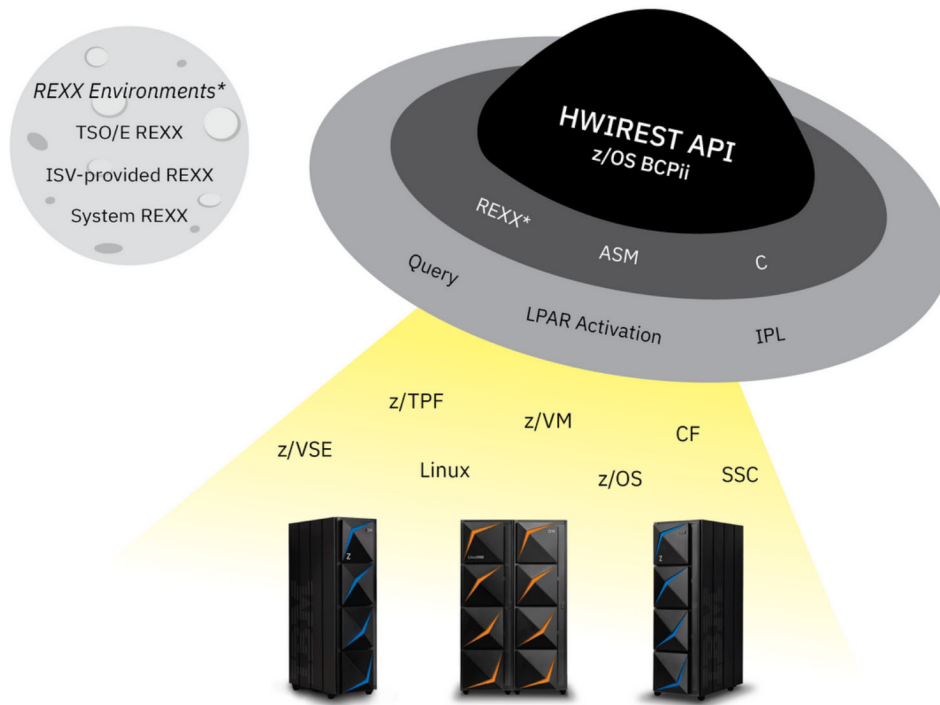
A new z/OS BCPii API, HWIREST, is introduced for the IBM z15. This API provides a simpler and more intuitive REST programming model for applications to use to access many previously unavailable attributes and instantly access future attributes that may become available.



By: Galina Gorelik, Neil Shah  
Published: February 25th, 2022  
Read time: 10 minutes



*This article was updated on August 11, 2022.*



Hopefully you got the play on words... (from the sitcom “Arrested Development”). For those programmers not familiar with BCPii, it provides a way for authorized applications running on z/OS to programmatically interact with IBM Z hardware. The application can query, change, and perform operational procedures, e.g., activating an LPAR, by using the set of provided BCPii APIs. These APIs use internal communication paths between the z/OS operating system and the hosting hardware, thus avoiding the IP network (intranet).

So, if you’re a z/OS systems programmer, what can you do with BCPii? Well, you can do some neat things. How many times have you needed to get the attributes of an LPAR (i.e., number of CPs or amount of memory)? Prior to BCPii, this required logging on to the HMC, going through the appropriate panels and updating an offline spreadsheet or document. But, as most of us found out, sometimes a systems programmer updates the LPAR configuration on the HMC and forgets to notify the document owner, leading to confusion and possible problems! So, what is a z/OS systems programmer to do? BCPii to the rescue!

However, as years and multiple new hardware releases have gone by, if you’re an existing BCPii user, you may have noticed BCPii started lagging a bit (or a lot) behind the attributes available on the SE/HMC GUI. Introducing HWIREST, the latest addition



to the set of BCPii APIs, which made its official debut in 2Q2021<sup>1</sup> on z15. If you fall into the existing group of users, this new addition gives you the ability to access and update many new attributes, including storage-class memory! It also gives TSO/E REXX and ISV REXX applications the ability to issue commands previously not permitted for those environments, for example, activate and load LPARs, and issue requests to the console.

## Let's meet the new API

HWIREST is a conduit that gives a z/OS application the ability to issue a core set of REST APIs as defined in “Chapter 11: Core IBM Z resources”, of the *IBM Z Hardware Management Console Web Services API* publication.




A PDF of the *IBM Z Hardware Management Console Web Services API* publication can be found on Resource Link: [www.ibm.com/servers/resourcelink](https://www.ibm.com/servers/resourcelink). Click Library on the navigation bar and then select your IBM Z server for a list of publications that are related to your server.

This single API is used to perform all forms of requests, from query through LPAR activation, all the while using the internal communication path to the hardware.


HWIREST adheres to the same set of BCPii authorization requirements that are in place for the existing APIs. So, if you already have a set of HWI.\* FACILITY class profiles defined, you can reuse them for your interaction with HWIREST. The main difference is the identification that is associated with each resource, CPC, LPAR, CapRec, and others. The existing APIs use the concept of a connection token to uniquely identify the resource after a connection to that resource is established. The token contains the `netid.nau` and possibly an additional name value that is associated with the specific resource. That information is then used to build the SAF profile to use for authorization of the request. Unlike its siblings, HWIREST does not have a connection token. Instead, each resource's identity is formed by a combination of the URI and target name that is passed as input on the request. That identity is then used to build the corresponding SAF profile.

You are probably thinking, “how do I obtain the URI and target name?” Numerous LIST operations are available for each supported resource:


- List CPC Objects - GET /api/cpcs

 NOTE: Each CPC entry returned will contain a location property; the entry with the value local is your LOCAL CPC.

- List Logical Partitions of CPC - GET /api/cpcs/{cpc-id}/logical-partitions

 NOTE: Each LPAR entry returned will contain a request-origin property; the entry with the value true is your LOCAL LPAR.

- List Group Profiles - GET /api/cpcs/{cpc-id}/group-profiles

 NOTE: Query a Group Profile's "effective-\*" properties to obtain real time group capacity values.

- List Image Activation Profiles - GET /api/cpcs/{cpc-id}/image-activation-profiles
- List Capacity Records - GET /api/cpcs/{cpc-id}/capacity-records ...and many more

Each LIST operation request returns various pieces of information for each entity, including the **object-uri** (CPCs, LPARs, Profiles) or **element-uri** (Capacity Record) and **target-name**.

For example, to obtain a list of CPCs you issue the **List CPC Objects** operation. That operation returns an array of CPC entities, each of which includes the unique URI for that CPC (value of the **object-uri** property), and the targeting information for that CPC (value of the **target-name** property). Those two pieces of information form the CPC identity.

{



```

"name": "CPC1",
"se-version": "2.15.0",
"location": "local",
"object-uri": "/api/cpcs/66666666-2222-bbbb-aaaa-aaaaaa",
"target-name": "IBM390PS.CPC1"
}

```

Your application builds on this data and/or reuses it for all subsequent requests that are associated with that CPC.

One example of a subsequent request is to query storage-related properties on that CPC. Specifically, you can query the total storage that is installed, the storage available for a customer, and the total VFM storage. The first step is to find the corresponding CPC properties names. The CPC Data Model section in chapter 11 of the *Hardware Management Console Web Services API* publication (Chapter 11 -> CPC object -> Data Model) contains all the various properties available for a CPC. In the Data Model section, scroll a bit to find the properties you are seeking, **storage-total-installed**, **storage-customer**, and **storage-vfm-total**:

Name	Qualifier	Type	Description
hardware-messages	(c)(pc)	Array of hardware-message objects	<p>The complete list of all CPC hardware messages, each identified by its URI. This list corresponds to the list provided by the <code>List CPC Hardware Messages</code> operation. If the CPC has no hardware messages, then an empty array is provided.</p> <p>The list of returned hardware messages can change as a result of new messages being dynamically added or removed by the infrastructure or due to hardware messages being deleted through the <code>Delete CPC Hardware Message</code> operation.</p> <p><b>Note:</b> This property is not returned by the <code>Get CPC Properties</code> operation, and only sessions associated with an HMC user with permission to the Hardware Messages task will receive a property-change notification for this property.</p>
storage-total-installed	—	Long	Amount of installed storage, in megabytes.
storage-hardware-system-area	—	Long	Amount of storage, in megabytes, reserved for the base hardware system area (HSA).
storage-customer	—	Long	Amount of storage, in megabytes, for use by the customer.
storage-customer-central	—	Long	Amount of storage, in megabytes, which is the central storage in use across the active partitions.
storage-customer-expanded	—	Long	<p>Amount of storage, in megabytes, which is the expanded storage in use across the active partitions.</p> <p>When the <code>iml-mode</code> is <code>"dpm"</code> this property will be set to 0.</p>
storage-customer-	—	Long	Amount of storage, in megabytes, which is not in use



available			
storage-vfm-increment-size <sup>1</sup>	—	Long	The increment size of any IBM Virtual Flash Memory (VFM) storage property, in gigabytes (GB), on this CPC and its logical partitions or partitions.
storage-vfm-total <sup>1</sup>	—	Long	The total amount of VFM storage, in gigabytes (GB), installed on this CPC. The valid value should be a multiple of the value indicated on the <b>storage-vfm-increment-size</b> property for this CPC.
maximum-hipersockets	—	Integer	The maximum number of HiperSocket adapters that may be created for the CPC when the CPC is enabled for DPM.

Figure 1: CPC Data Model

The next step is to figure out the syntax of the GET CPC Properties operation (Chapter 11 -> CPC object -> Get CPC Properties) that will be used to query those properties.

>

Group Object

✓

CPC object

>

Data model

List CPC Objects

Get CPC Properties

Update CPC Properties

Start CPC

Stop CPC

### Get CPC Properties

The Get CPC Properties operation retrieves the properties of a single CPC object designated by {cpc-id}. This operation is supported using the BCPii interface.

#### HTTP method and URI

```
GET /api/cpcs/{cpc-id}
```

In this request, the URI variable {cpc-id} is the object ID of the target CPC object.

**Query parameters:**

Name	Type	Rqd/Opt	Description
properties	List of String Enum	Optional	Filter string to limit returned properties to those that are identified here. This is a list of comma-separated strings where each string is a property name defined in the CPC object's data model.

Figure 2: Get CPC Properties operation

The operation syntax (Figure 2) is broken down as follows:

- **GET** is the required HTTP method, you'll use HWIREST's equivalent HWI\_REST\_GET constant.
- `/api/cpcs/{cpc-id}` is the URI for your CPC which you obtained by issuing the initial LIST CPCs request ->
 

```
"object-uri" = URI of CPC -> "/api/cpcs/6666666-2222-bbbb-aaaa-aaaaaa"
```
- **properties** and **cached-acceptable** are optional query parameters that you can append to the CPC URI to filter what CPC properties are returned and the source of those values. If the properties filter is omitted, all the properties that are defined in the CPC Data Model are returned in the response for this request. In this case, let's take advantage of the properties filter to explicitly state we want only the values for **storage-total-installed**, **storage-customer**, and



**storage-vfm-total** returned to us in the response body.



NOTE: For the best performance results always take advantage of the **cached-acceptable** property if it is available.

- In addition, you also need the targeting information for the CPC you are interacting with; this is the other piece of information you obtained by issuing the initial LIST CPCs request -> "target-name": "IBM390PS.CPC1"

When all the various pieces of information are combined, you can form the following HWIREST request:

```
requestParm.httpMethod = HWI_REST_GET
requestParm.uri         = '/api/cpcs/66666666-2222-bbbb-aaaa-aaaaaa?pro
                           storage-total-installed,storage-customer,sto
                           total&cached-acceptable=true'
requestParm.targetName  = 'IBM390PS.CPC1'
requestParm.requestBody = <leave blank since this REST API doesn't req
                           request body>
```



NOTE: Do not add extra / characters at the end of the URI. The URI must be exactly as shown in the documentation.

If the request is successful, your response parm contains the following output:

```
responseParm.responseDate = Wed, 06 May 2021 18:05:36 GMT
responseParm.requestId    = Sxbbbb-5555-1111-00000.1f3 Rxc
responseParm.httpStatus   = 200 (OK)
responseParm.responseBody =
{
  "storage-vfm-total":0,
  "storage-customer":2883584,
```





```

    "storage-total-installed":3145728
  }

```

Putting it all together into a picture, here is the request and response:

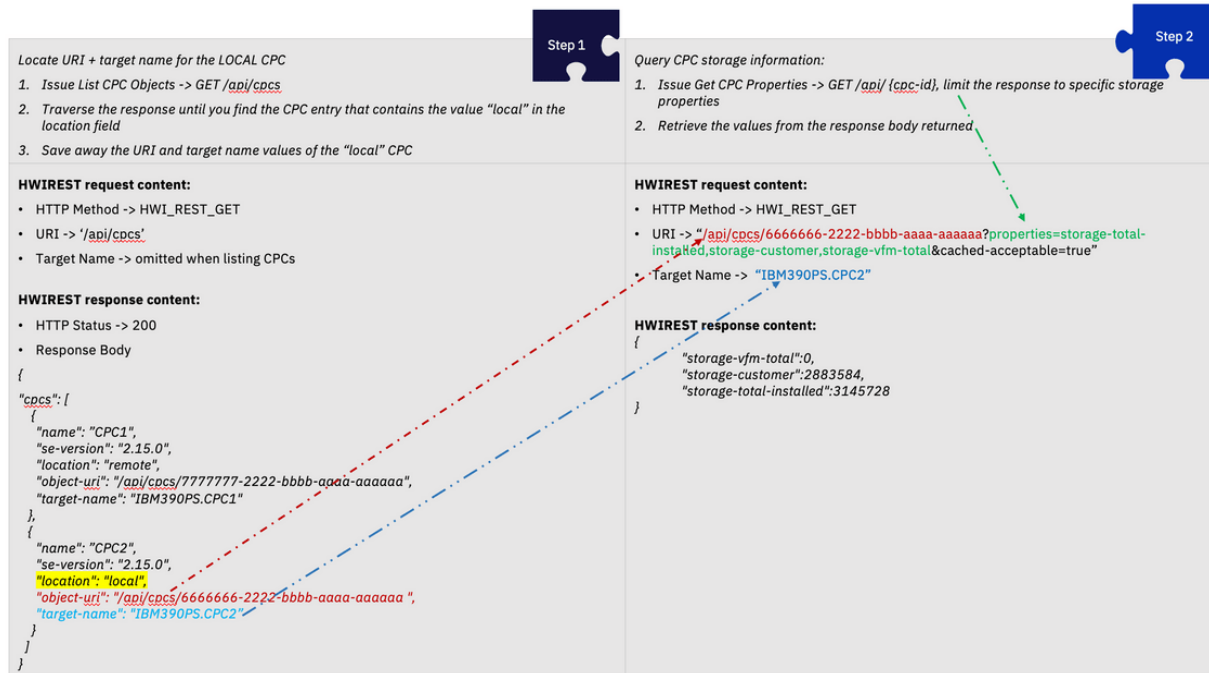


Figure 3: HWIREST request and response

One of the things you might have noticed in the previous example is that the response content is always in JSON format. HWIREST uses JSON as the format for the request and response data content. It supports both UTF-8 and IBM-1047-character encoding. In case you are worried you must code up a JSON parser yourself, I would like to shamelessly pitch and highly recommend the [z/OS client web enablement toolkit JSON Parser](#), which is built into the z/OS base, for all your JSON parsing needs.

The other thing that you might have noticed is that there is nothing that resembles a return code parameter that is typically found in a z/OS API. Don't fret! HWIREST is staying true to its RESTful nature; on success it returns an HTTP status value in the 200-299 range (Chapter 3. Invoking API operations -> HTTP status codes).

## HTTP status codes

The HMC API provides standard HTTP status codes in the response to requests to indicate the success or failure of the request. Unless stated otherwise in the description of an operation, the following general interpretations of the status code values apply.



HTTP status code	Description/Causes
200 (OK)	The request has succeeded completely. A response body is provided that contains the results of the request.
201 (Created)	The request has succeeded completely and resulted in the creation of a new managed resource/object. The URI for the newly created managed resource is provided in a <b>Location</b> header. (POST methods only)
202 (Accepted)	The request was successfully validated and has been accepted to be carried out asynchronously.
204 (No Content)	The request succeeded completely, and no additional response information is provided.
400 (Bad Request)	The request was missing required input, had errors in the provided input, or included extraneous input. Additional information regarding the error is provided in an error response body that includes a reason code with additional information.
403 (Forbidden)	Multiple error conditions result in this status code: <ul style="list-style-type: none"> <li>The request requires authentication but no <b>X-API-Session</b> header was provided, or one was provided but the session ID was invalid.</li> <li>The user under which the API request was authenticated is not authorized to perform the requested operation.</li> </ul>
404 (Not Found)	The URI does not designate an extant resource, or designates a resource for which the API user does not have object-access permission.
405 (Method Not Allowed)	The request specifies an HTTP method that is not valid for the designated URI.
406 (Not Acceptable)	The <b>Accept</b> header for the request does not include at least one content representation supported by the Web Services API.
409 (Conflict)	The managed resource is in an incorrect state (status) for performing the requested

Figure 4: HTTP status codes

A value outside of that range indicates an error and additional information that will help explain the error will be provided in the accompanying reason code and response body. In addition to documenting HTTP status codes, Chapter 3 in *Hardware Management Console Web Services API* publication is also where you'll find the error response bodies section that defines the various fields included in the response body on error and descriptions of the various reason codes you could encounter.

Now for a few words about the HWIREST parameters, two to be exact, one for the content of request (input) and one that contains the result of the request (output).

The request parameter is broken down into various fields that correspond to content in the *Hardware Management Console Web Services API*, though the names might vary slightly.

Request parameter fields

Equivalent content in the HMC Web Services API



Request parameter fields	Equivalent content in the HMC Web Services API
httpMethod	HTTP method for the REST API operation
uri	URI for the REST API operation
targetName	X-API-Target-Name request header
requestBody	Request body contents for the REST API operation
clientCorrelator	X-Client-Correlator request header
encoding	Value of the charset used to encode all the input and output
requestTimeout	Amount of time the SE is given to carry out the request before giving up

The response parameter is broken down in a similar fashion; its various fields correspond to content in the *Hardware Management Console Web Services API*.

Request parameter fields	Equivalent content in the HMC Web Services API
responseDate	Date response header
requestId	X-Request-Id response header
location	Location response header
responseBody	Response body contents for the REST API operation
httpStatus	HTTP Status Code
reasonCode	Reason

You might have noticed a bit of a trend by now. The *MVS Programming: Callable Services for High-Level Languages* publication helps you understand the syntax of



HWIREST, but after you learn how to interact with this API, you might never need to return to it again because all the content is in the *Hardware Management Console Web Services API*.

How do you know which API is applicable to HWIREST in the *Hardware Management Console Web Services API* section?

The full list of supported REST API operations through HWIREST can be found in “Appendix A: Base Control Program internal interface (BCPii)” in *Hardware Management Console Web Services API*. In addition to the information in Appendix A, each supported operation includes the verbiage: “This operation is supported using the BCPii interface.”

### List CPC Objects

The **List CPC Objects** operation returns a list of the zManager Web Services API capable managed CPCs. This operation is supported using the BCPii interface.

#### HTTP method and URI

```
GET /api/cpcs
```

#### Query Parameters

Name	Type	Rqd/Opt	Description
name	String	Optional	A regular expression used to limit returned objects to those that have a matching name property. If matches are found, the response will be an array with all objects that match. If no match is found, the response will be an empty array.

Figure 5: How to tell if an operation is supported using the BCPii interface

Now you’re thinking, okay but what about the various attributes, in the Data Model and in the operation descriptions. The attribute is supported unless there is an explicit note in the description that states: “This property is only returned when the web services interface was used for the request.”

Field name	Type	Description
object-uri	String/ URI	Canonical URI path of the CPC object
name	String	The name of the CPC object
status	String Enum	The current status of the CPC object <b>Note:</b> This property is only returned when the web services interface was used for the request.



Figure 6: How to tell if a property is not supported in BCPii

## Staying in sync with async

Before you try out the new API, let's talk about asynchronous requests (have I mentioned you can issue these out of TSO/E REXX and ISV REXX environments?). Some REST API operations, such as **Activate Logical Partition**, are asynchronous (Chapter 11 -> Logical Partition -> Activate Logical Partition):

### Activate Logical Partition

The **Activate Logical Partition** operation activates the Logical Partition object designated by *{logical-partition-id}*. This operation is supported using the BCPii interface.

#### HTTP method and URI

```
POST /api/logical-partitions/{logical-partition-id}/operations/activate
```

In this request, the URI variable *{logical-partition-id}* is the object ID of the target Logical Partition object.

#### Request body contents

The request body is expected to contain a JSON object with the following fields. If none of the optional fields are included, an empty request body must be supplied.

Field name	Type	Rqd/ Opt	Description
activation-profile-name	String (1-16)	Optional	The name of the activation profile to be used for the request. If not provided, the request uses the profile name specified in the <b>next-activation-profile-name</b> property for the Logical Partition object.
force	Boolean	Optional	Whether this operation is permitted when the logical partition is in <b>"operating"</b> status (true) or not (false). The default is false.

#### Response body contents

Once the operation is accepted, the response body contains a JSON object with the following fields:

Field name	Type	Description
job-uri	String/ URI	URI that may be queried to retrieve status updates.

#### Asynchronous result description

Once the operation has completed, a job-completion notification is sent and results are available for the asynchronous portion of this operation. These results are retrieved using the **Query Job Status** operation directed at the job URI provided in the response body.

The result document returned by the **Query Job Status** operation is specified in the description for the **Query Job Status** operation. When the status of the job is **"complete"**, the results include a job completion status code and reason code (fields **job-status-code** and **job-reason-code**) which are set as indicated in "Job status and reason codes" on page 1027. The **job-results** field is null when this operation is successful. When it is partially successful or not successful, the **job-results** field contains an object with the following field:

Figure 7: Asynchronous results



The response body that is returned contains a job URI. That job URI is associated with the asynchronous job processing operation (Chapter 7 -> Asynchronous job processing):

The screenshot shows the IBM Z: HMC Web Services API documentation for the 'Query Job Status' operation. On the left is a 'Bookmarks' sidebar with a tree view. The main content area is titled 'Query Job Status' and includes a description, the HTTP method and URI, and a table of response body contents.

**Bookmarks**

- Contents
- Figures
- Tables
- > Safety
- > About this publication
- > Part 1. Web Services API fundamentals
- ✓ Part 2. General services
  - ✓ Chapter 7. General API services
    - General API services operations summary
    - > Session management services
    - > Request aggregation services
    - ✓ Asynchronous job processing
      - Query Job Status**
      - Delete Completed Job Status
      - Cancel Job
  - ✓ Chapter 8. Inventory and metrics services
    - Inventory services operations summary
    - Metrics service operations summary
    - > Inventory service
    - > Metrics service

**Query Job Status**

The Query Job Status operation returns the status associated with an asynchronous job. This operation is supported using the BCPii interface and is the only mechanism to obtain information for asynchronous jobs.

**HTTP method and URI**

GET /api/jobs/{job-id}

**116 IBM Z: HMC Web Services API**

Level 04b

In this request, the URI variable {job-id} is the identifier of an asynchronous job associated with the API user, as returned in the response of the operation that initiated the job.

**Response body contents**

On successful completion, the response body is a JSON object with the following fields:

Field name	Type	Description
status	String Enum	An indication of the current disposition of the job. The possible values are as follows: <ul style="list-style-type: none"><li>"running" - indicates that the job was found and it has not ended at the time of the query.</li><li>"cancel-pending" - indicates that the job was found and it has not ended but cancellation has been requested.</li><li>"canceled" - indicates that the job's normal course of execution was interrupted by a cancel request.</li></ul> The successful or unsuccessful completion of the job is indicated by the <b>job-status-code</b> and <b>job-reason-code</b> fields.
job-status-code	Integer; Field provided only if status is "complete" or "canceled"	The job completion status code. This field is provided only if the <b>status</b> field is set to "complete" or "canceled". This field provides the major status code describing the success or failure completion of the asynchronous action represented by the job. It is expressed in terms of an HTTP status code (i.e. the HTTP status code that would have been returned for the operation had it been performed synchronously). The values provided here and their meaning depend on the particular action that is being performed asynchronously. The

Figure 8: Query Job Status operation

Your application uses HWIREST to poll that JOB URI to determine the result of the asynchronous request.



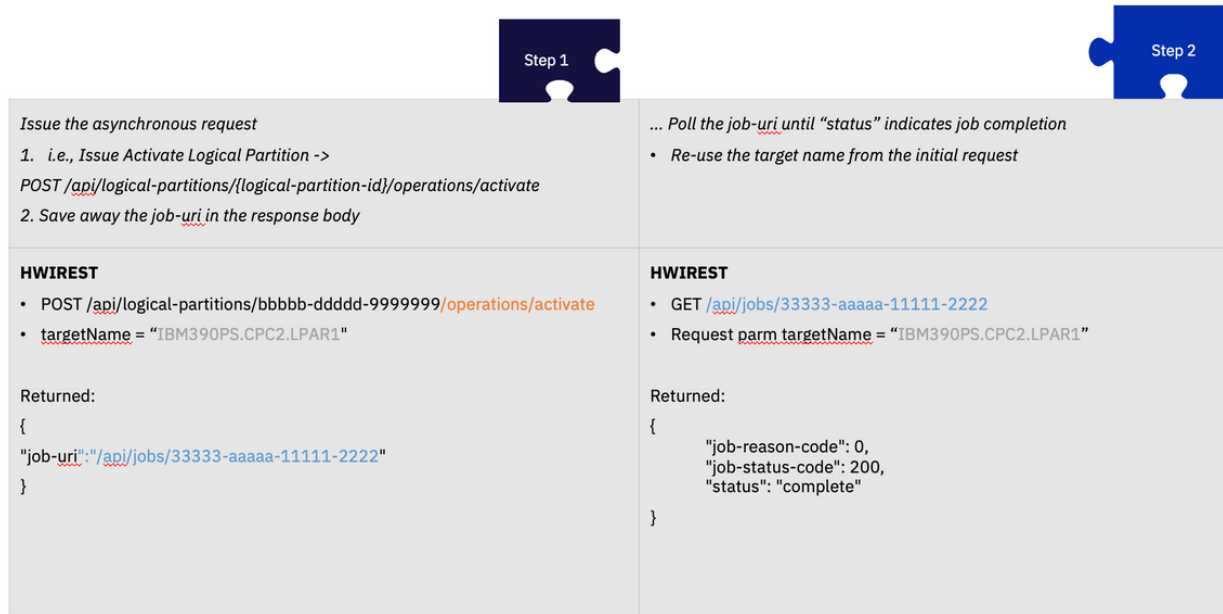


Figure 9: Issuing an asynchronous request

## Samples!

Have we got some samples for you! You can find them at:

<https://github.com/IBM/zOS-BCPii>. One of the samples, RXAUDIT1, goes to the CPC and lists attributes, number of general-purpose CPs, zIIPs, storage, etc., for all LPARs. It then stores the results, which are in .csv format, in a z/OS data set member. You can then import the content into Microsoft™ Excel or another application of your choice.

	A	B	C	D	E	F
1	LPAR Name	status	processor-usage	number-general-purpose	number-reserved-ge	number-general-purpose-cores
2	LPAR1	operating	shared	3	0	3
3	LPAR2	operating	shared	3	3	3
4	XC1	operating	dedicated	0	0	
5	ZVM1	not-operating				
6	LPAR3	not-activated				

Figure 10: Data obtained using the RXAUDIT1 sample

## In Summary

z/OS BCPii is excited to introduce you to its newest member, HWIREST API. This API provides a new REST-like interface for applications to use to access many previously unavailable attributes and allows TSO/E REXX and ISV REXX applications the ability to issue commands previously not permitted for those environments. In addition, as firmware updates are applied that introduce new attributes to currently supported resources, you no longer have to wait for z/OS to make its corresponding APAR available. In most cases, you can access the new attributes instantaneously!

## About the authors

Galina Gorelik is the product owner of z/OS BCPii and z/OS client web enablement toolkit.

Neil Shah is an IBM z/OS Systems Programmer.

*Anne Romanowski contributed to the editorial review of this article.*

*Image created by Izzi Cain.*

- 
1. HWIREST is available for BCPii on z/OS 2.4 with APAR OA60351 and is included in the base of later z/OS releases. An enhancement to HWIREST that allows access in TSO/E and ISV REXX environments to previously restricted REST API operations was made available in z/OS 2.4 and z/OS 2.5 with APAR OA61976. It is also included in the base of later releases. HWIREST requires a z15, SE 2.15.0 with MCL P46598.370, Bundle S38 or higher, and HMC 2.15.0 with MCL P46686.001, Bundle H25 or higher.[↩](#)





