Manage IBM CICS systems on the go: Prototype for IBM MobileFirst applications

Morihiro Kawanami (KAWANAMI@jp.ibm.com)
Consulting IT Specialist
IBM

Tomohiro Taguchi (TOMOTAG@jp.ibm.com)
Advisory IT Specialist
IBM

Learn how to create and implement a mobile solution prototype for IBM CICS systems management. By creating this prototype, you can improve the usability of CICS system management functions and incorporate them with your IBM MobileFirst applications. For example, you can monitor CICS resource usage remotely on your mobile devices wherever and whenever you need. In this prototype, the CICS storage status is displayed graphically on a mobile device for monitoring. This solution entails using a RESTful API, XSLT, and JSON technologies on CICS Transaction Server and IBM MobileFirst Platform Foundation.

Introduction

In this tutorial, you learn the technical aspects of a mobile prototype for IBM® CICS® systems management. This prototype improves the usability of CICS system management functions. You can then incorporate them with your IBM MobileFirst™ applications to provide an easy and prompt mobile solution for CICS systems management. This tutorial includes an example that shows how you can monitor CICS resource usage remotely on your mobile devices wherever you are and whenever you need.

Technology for prototype

The prototype that is described in this tutorial uses the following technology:

- CICS management client interface
- IBM MobileFirst Platform Foundation
- HTTP adapter

CICS management client interface

The CICS management client interface (CMCI) is a RESTful programming interface for systems management that was introduced in CICS Transaction Server (TS) V4.1. By using CMCI, you can
implement modernized system management functions that are equivalent to the CEMT commands that are used on the traditional 3270 user interface.

CMCI uses the following HTTP methods to interact with a CICS server:

- **GET** for Read
- **POST** for Create
- **PUT** for Update
- **DELETE** for Delete

You can develop your own management clients by using CMCI as an HTTP application. For example, IBM CICS Explorer uses CMCI internally.

Listing 1 shows the XML schema, CICSSystemManagement, that is used in the CMCI.

**Listing 1. CICSSystemManagement schema**

```xml
<xsd:schema targetNamespace="http://www.ibm.com/xmlns/prod/CICS/sm2int" elementFormDefault="qualified" version="3.0" smw2int:connect_version="0520">
  <xsd:include schemaLocation="http://ePlexA:52002/CICSSystemManagement/schema/Feedback.xsd"/>
  <xsd:include schemaLocation="http://ePlexA:52002/CICSSystemManagement/schema/CICSBridgeFacility.xsd"/>
  <xsd:include schemaLocation="http://ePlexA:52002/CICSSystemManagement/schema/CICSCFDTPool.xsd"/>
  <xsd:include schemaLocation="http://ePlexA:52002/CICSSystemManagement/schema/CICSDynamicStorageArea.xsd"/>
  <xsd:include schemaLocation="http://ePlexA:52002/CICSSystemManagement/schema/CICSPagePool.xsd"/>
  <xsd:include schemaLocation="http://ePlexA:52002/CICSSystemManagement/schema/CICSGlobalDynamicStorageArea.xsd"/>
  ...
</xsd:schema>
```

**IBM MobileFirst Platform Foundation**

IBM MobileFirst Platform Foundation is a platform for mobile application development and includes the following components:

- **MobileFirst Studio.** An integrated environment based on Eclipse for mobile application development.
- **Client-side runtime components.** A developer kit for client-side code that runs in mobile devices.
- **MobileFirst Server.** A scalable gateway that connects mobile devices to enterprise systems securely.
- **Application Center.** A mobile application distribution mechanism.
- **MobileFirst Operations Console.** A web-based user interface to manage and monitor mobile applications.

For development, you install the MobileFirst Platform Foundation plug-in into a PC environment, where you can code and test it locally. After testing is completed, you deploy the artifacts to the MobileFirst Server to connect mobile devices to back-end systems.

**HTTP adapter**

IBM MobileFirst Platform Foundation provides several adapters to connect to back-end systems, including an HTTP adapter, an SQL adapter, and a Cast Iron adapter. You can also build your own custom adapters. A mobile application starts a procedure in an adapter by using
the WL.Client.invokeProcedure method. Then, the procedure starts a service in a back-end system, by using the WL.ServerinvokeHttp method, and gets a response in the JavaScript Object Notation (JSON) format. Because of this mechanism, if you develop mobile applications, you can concentrate on developing front-end applications without needing to know details of the specifications to connect to the back-end system.

**Implementing the HTTP adapter**

We implemented an HTTP adapter to get information about CICS dynamic storage area (CDSA) usage by using a CMCI request. Figure 1 shows the architecture of our solution.

**Figure 1. Mobile solution architecture for CICS systems management**

Assuming that the target CICS regions are part of a CICSpux, CMCI requests are sent to a web user interface (UI) server.

First, we examine the kind of information that can be obtained from a CMCI request. Listing 2 shows an HTTP request to get information about dynamic storage area usage for CICS region CT51S4A1, which is a managed region in CICSpux "C68PLX."

**Listing 2. CMCI(HTTP) request**

```
GET http://G9252JX:51002/CICSSystemManagement/CICSDynamicStorageArea/C68PLX/CT51S4A1
```

Listing 3 shows the response to the request. The response is returned in the XML format. It contains too much information and is not easily handled by a mobile application that is written in JavaScript.

**Listing 3. CMCI(HTTP) response**

```
200 OK
Cache-Control: no-store
Date: Thu, 12 Mar 2015 01:51:03 GMT
Server: IBM_CICS_Transaction_Server/5.1.0(zOS)
Content-Type: application/xml; charset=UTF-8
Transfer-Encoding: chunked
Connection: Keep-Alive
<?xml version="1.0"?>
<response xmlns="http://www.ibm.com/xmlns/prod/CICS/smw2int"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  http://G9252JX:51002/CICSSystemManagement/schema/CICSSystemManagement.xsd"
  version="3.0" connect_version="0510">
```
To simplify the response output, we extract only the necessary information from the XML data for a mobile application developer. For example, to help us understand storage usage more easily and intuitively, we want to view a chart, like the example in Figure 2, on a mobile device.

**Figure 2. Storage usage chart**

![Storage usage chart](chart.png)

When we focus on the CDSA, only the bold parameters that are shown in Listing 4 are necessary for creating the chart.
Listing 4. CDSA information in the CMCI(HTTP) response

```
<cicsdynamicstoragearea _keydata="C3C4E2C140404040" accesstype="CICS" asactive="0" asubtotl="43" atbcushlimit="0" atbcushrel="0" cumcmnssusrs="0" cumunqssusrs="0" curralloc="2097152" curunqssusrs="0" cushion="65536" dsubtotl="43" extentsadde="0" extent:scurr="2" extent:sdeltd="0" eyu_cicsname="CT51S4A1" eyu_cicsrel="E680" eyu_reserved="0" fremtotl="4" gdsaactive="0" getmtotl="7" getstorsize="0" hmwcmnssusrs="4" hwmfree="200704" hwmgsaactv="0" hwmungssusrs="1" limit="5242880" location="BELOW" lwmfree="184320" memlimit="0" name="CDSA" nstgcurr="0" nstgtotl="0" pctfree="89.3" pgmepn="0" poolpoolffree="35.9" regmwait="0" rntpgprotect="REENTPROT" size="524288" stgcrcel="0" stgfsize="188416" stghw="524288" stghwsize="126076" stgsubp="37" stgprotect="ACTIVE" stgwait="0" stgshw="0" stgsoc="0" stgso="0000:00:06.000000" stgwait="0" stgtotl="0" tnmwaitmvs="0000:00:00.000000" trnisolation="ACTIVE" />
```

In the XML data that is received from the CMCI, each value is provided as an attribute of the `<cicsdynamicstoragearea>` element. We need the values of the following attributes of the `<cicsdynamicstoragearea>` element that has the `name="CDSA"` attribute:

1. **size.** The size of the dynamic storage area (DSA) in bytes.
2. **stgfsize.** The amount of free storage, including the cushion, in this DSA.
3. **curralloc.** The current amount of DSA that is allocated.
4. **limit.** The maximum amount of storage, as a total number of bytes, within which CICS can dynamically allocate storage for the four individual DSAs that are below the 16M boundary.

In Figure 3, the numbers correspond to the values for the numbered attributes in the previous list.

**Figure 3. Storage usage chart that maps to the CMCI response**

![Storage usage chart](image)

Similarly, for the read-only DSA (RDSA), shared DSA (SDSA), and user DSA (UDSA), we extract the size value and then the stgfsize value.

We can build an Extensible Stylesheet Language Transformation (XSLT) to parse or filter the XML data from the HTTP request and then transform it into a sophisticated JSON format. Listing 5 shows the XLST to extract the required information in the JSON format that is needed to create the chart that is shown in Figure 3.

**Listing 5. XSLT (filtered.xsl)**

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:h="http://www.w3.org/1999/xhtml"
    xmlns:cics="http://www.ibm.com/xmlns/prod/CICS/smw2int">
  <xsl:output method="text"/>
  <xsl:template match="/">
    {
    ```
'dynamicStorage': {
  <xsl:for-each select="//cics:cicsdynamicstoragearea">
    <xsl:choose>
      <!-- ********** DSA ********** -->
      <xsl:when test="/@name='CDSA'">
        'cdsa': {
          'used': <xsl:value-of select="./@size - ./@stgfsize"/>,
          'free': <xsl:value-of select="./@stgfsize"/>,
          'allocated': <xsl:value-of select="./@size"/>,
          'total_allocated': <xsl:value-of select="./@curralloc"/>,
          'total_limit': <xsl:value-of select="./@limit"/>
        },
      </xsl:when>
      <xsl:when test="/@name='RDSA'">
        'rdsa': {
          'used': <xsl:value-of select="./@size - ./@stgfsize"/>,
          'free': <xsl:value-of select="./@stgfsize"/>,
          'allocated': <xsl:value-of select="./@size"/>,
        },
      </xsl:when>
      <xsl:when test="/@name='SDSA'">
        'sdsa': {
          'used': <xsl:value-of select="./@size - ./@stgfsize"/>,
          'free': <xsl:value-of select="./@stgfsize"/>,
          'allocated': <xsl:value-of select="./@size"/>,
        },
      </xsl:when>
      <xsl:when test="/@name='UDSA'">
        'udsa': {
          'used': <xsl:value-of select="./@size - ./@stgfsize"/>,
          'free': <xsl:value-of select="./@stgfsize"/>,
          'allocated': <xsl:value-of select="./@size"/>,
        },
      </xsl:when>
      <!-- ********** EDSA ********** -->
      <xsl:when test="/@name='ECDSA'">
        'ecdsa': {
          'used': <xsl:value-of select="./@size - ./@stgfsize"/>,
          'free': <xsl:value-of select="./@stgfsize"/>,
          'allocated': <xsl:value-of select="./@size"/>,
          'total_allocated': <xsl:value-of select="./@curralloc"/>,
          'total_limit': <xsl:value-of select="./@limit"/>
        },
      </xsl:when>
      <xsl:when test="/@name='ERDSA'">
        'erdsa': {
          'used': <xsl:value-of select="./@size - ./@stgfsize"/>,
          'free': <xsl:value-of select="./@stgfsize"/>,
          'allocated': <xsl:value-of select="./@size"/>,
        },
      </xsl:when>
      <xsl:when test="/@name='ESDSA'">
        'esdsa': {
          'used': <xsl:value-of select="./@size - ./@stgfsize"/>,
          'free': <xsl:value-of select="./@stgfsize"/>,
          'allocated': <xsl:value-of select="./@size"/>,
        },
      </xsl:when>
      <xsl:when test="/@name='EUDSA'">
        'eudsa': {
          'used': <xsl:value-of select="./@size - ./@stgfsize"/>,
      </xsl:when>
  </xsl:for-each>
}</xsl:choose>
Listing 6 shows an HTTP adapter implementation that refers to the XSLT in Listing 5.

**Listing 6. Procedure for implementing the HTTP adapter**

```javascript
function getDynamicStorageFiltered(regionName) {
    path = '/CICSSystemManagement/CICSDynamicStorageArea/C68PLX/' + regionName;
    var input = {
        method : 'get',
        returnedContentType : 'xml',
        path : path,
        transformation : {
            type : 'xslFile',
            xslFile : 'filtered.xsl'
        }
    };
    return WL.Server.invokeHttp(input);
}
```

When we start the HTTP adapter implementation procedure, the output is in the JSON format (Listing 7).
Listing 7. Output data in the JSON format

```
{
  "dynamicStorage": {
    "cdsa": {
      "allocated": 524288,
      "free": 288704,
      "total_allocated": 2897152,
      "total_limit": 5242880,
      "used": 323584
    },
    "ecdsa": {
      "allocated": 28971520,
      "free": 180224,
      "total_allocated": 57671680,
      "total_limit": 838860800,
      "used": 20791296
    },
    "erdsa": {
      "allocated": 33554432,
      "free": 2146304,
      "used": 31408128
    },
    "esdsa": {
      "allocated": 1048576,
      "free": 983040,
      "used": 65536
    },
    "etdsa": {
      "allocated": 1048576,
      "free": 1015808,
      "used": 32768
    },
    "eudsa": {
      "allocated": 1048576,
      "free": 1048576,
      "used": 0
    },
    "gcdsa": {
      "allocated": 1073741824,
      "free": 1059061760,
      "total_allocated": 0,
      "total_limit": 0,
      "used": 14680064
    },
    "gsdsa": {
      "allocated": 0,
      "free": 0,
      "used": 0
    },
    "gudsa": {
      "allocated": 0,
      "free": 0,
      "used": 0
    },
    "rdsa": {
      "allocated": 262144,
      "free": 49152,
      "used": 212992
    },
    "sdsa": {
      "allocated": 262144,
      "free": 253952,
      "used": 8192
    },
    "usda": {
      "allocated": 1048576,
      "free": 983040,
      "used": 65536
    }
  }
}
```
By using a customized XSLT, the minimum required amount of information is extracted as JSON data. With this information, you can build presentation logic, including tables and graphs, easily in JavaScript.

Now that we have the information we need, we can use it to develop a mobile application as explained in the following section.

**Usage example: Developing a mobile application**

By using the HTTP adapter that is shown in Listing 6, you can develop a mobile application that helps to visualize DSA usage, similar to the example in Figure 4.
In this example, Flotr2 is a library based on Canvas and is used to create the graphs.

If you run the HTTP adapter procedure that is shown in Listing 6 and store the data that is obtained, you can monitor how usage changes over time. To run the HTTP adapter procedure periodically, you can use the Event Source function that the MobileFirst Platform provides, as shown in Listing 8.

**Listing 8. Event Source implementation**

```javascript
WL.Server.createEventSource({
  name: "monitorDynamicStorageEventSource",
  poll: {
    interval: 30,
    onPoll: "monitorDynamicStorage"
  }
});

function monitorDynamicStorage(){
  /* get and store Dynamic Storage Data */
  ...
}
```

In this code fragment, the monitorDynamicStorage function is called automatically every 30 seconds. Figure 5 shows the stored historical data as a graph.
You can apply the technique as described in this tutorial to implement the following scenario:

1. Set a threshold for the storage amount.
2. If the storage exceeds the threshold, send a push notification to the system administrator.
3. Change the DSA limit dynamically as a temporary workaround to avoid a short-on-storage situation.

**Conclusion**

In this tutorial, we showed how to implement an HTTP adapter that issues CMCI requests to get CICS runtime information about DSA usage for system administrators. Because CMCI is a standard function that CICS Transaction Server provides, you don't need to develop any applications for this scenario on the CICS side. The data that is returned from CMCI requests is in the XML format. However, you can transform it to the JSON format by using XSLT in the HTTP adapter to make it easier to handle in JavaScript. The JSON data is obtained by using the standard IBM MobileFirst Platform Foundation API. Therefore, if you develop mobile applications,
you can develop front-end applications easily without knowing the access method to the back-end system.

From a system administration perspective, you can use CMCI for other purposes. For example, you can use it for task status monitoring (such as to determine the number of tasks or the status of resident tasks). You can also use it for connection status between regions and for control of CICS resources (such as enabling and disabling them).

By using the technique that is introduced in this tutorial, you can develop your own mobile applications and customize them to meet your functional requirements.
Resources

- IBM Knowledge Center for CICS TS V5.2: CICS Management Client Interface
- IBM MobileFirst Platform: Getting Started 7.0
- Avoid common XSLT mistakes
- IBM developerWorks WebSphere zone
About the authors

Morihiro Kawanami

Morihiro Kawanami is a Consulting IT Specialist, who has 26 years of experience in IT services as an IBM CICS subject matter expert (SME) in Japan. His areas of expertise include working with a transaction processing system and an IBM Rational modernization solution.

Tomohiro Taguchi

Tomohiro Taguchi is an IT Specialist in Japan. He has worked in technical support primarily with IBM CICS family products, including CICS Transaction Server for z/OS, TXSeries, and CICS Transaction Gateway.

© Copyright IBM Corporation 2015
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)