SHARE in Pittsburgh – Session 15802

Introduction to the Hardware Management Console UI and API

Monday, August 4th 2014

Jason Stapels
HMC Development
jstapels@us.ibm.com
Agenda

• HMC User Interface
  • Tree Style and Classic Style
  • Changing Layout Preferences
  • Animated Tours & Tutorials
  • Classic Style Layout
  • Tree Style Layout
  • Wizard / Details Tasks

• HMC Application Programmer Interfaces
  • Overview
  • CIM
  • SNMP
  • Web Services API
HMC User Interface
HMC User Interface – Default UI Style

Tree Style (default)

- New users and new installs use Tree Style as default
- PEMODE and SERVICE Ids default to “Classic” style
- Upgrades will retain existing user preferences
- UI style change is immediate – no log off required

Classic Style (optional)
HMC User Interface – User Settings (Classic UI)

Launch User Settings To change UI Style
Launch User Settings To change UI Style
HMC User Interface – User Settings (UI Style)

- Select **UI Style** tab and then choose the user interface style

The new UI style will take affect immediately after clicking on **Apply** or **OK**
HMC User Interface – Getting Help

• Help can be found on IBM Information Center
  • http://pic.dhe.ibm.com/infocenter/hwmca/v2r12m1/index.jsp
HMC User Interface – Getting Help (cont.)

- Walkthroughs & tutorials can be found on IBM Resource Link
HMC User Interface – Classic Style Layout

View Navigation

Work Area

Tasks
HMC User Interface – Classic Style Layout (cont.)

- Tasks can be launched against targets:
  - Through a right-click pop-up menu
  - Dragging and dropping on the task
HMC User Interface – Classic Style Layout (cont.)

- Classic Style UI uses colors and patterns to identify statuses
  - Configured in the **User Settings** task

Acceptable Status colors can be configured in the **User Settings** task.
IBM® System z Hardware Management Console (HMC) 2.12.1

HMC User Interface – Tree Style Layout

- **Task Bar**
- **Work Pane**
- **Navigation Area**
- **Status Bar**

Diagram showing the layout of the HMC user interface with labeled sections for Task Bar, Work Pane, Navigation Area, and Status Bar.
HMC User Interface – Navigation Area

• Allows quickly jumping between sections of the Tree UI
• Navigate between previous selections
• Set a homepage
HMC User Interface – Work Pane

• Lists objects based on your current navigation area
• View properties and perform operations

Switch between different types of views
HMC User Interface – Work Pane (cont.)

(1) Expand / Collapse All
(2) Select / Deselect All
(3) Export Data
(4) Configure / Clear Filtering
(5) Edit / Clear Sorting
(6) Display / Hide / Reset Columns
(7) Filter on Specific Text
HMC User Interface – Work Pane (cont.)

- When filtering, you can select criteria to search against:
  - All
  - Name
  - System
  - Status
  - Description
HMC User Interface – Tasks Pad

• Displays valid actions based on the selected object(s)
• Automatically hides actions if the target(s) would be invalid
HMC User Interface – Tasks Index

- Lists all available tasks making it easy to search for a specific task.

Access from the Navigation Area

Filter On a Specific Keyword or Phrase
HMC User Interface – Status Area

• At a glance view showing the status of your machines

• Quick jump to:
  
  (1) Status Overview – Shows a summary of all affects objects
  (2) Exceptions – List all objects with exceptions
  (3) Hardware Messages – Lists all objects with hardware messages
  (4) Operating System Messages – Lists all objects with OS Messages

A disabled icon indicates that no notices of that type currently exist.
Clicking on a category takes you to a list of the affected objects.
HMC User Interface – Wizard Tasks

- Wizard style tasks guide you step by step

Completed and remaining steps are shown on left.

Uncommon options are default to standard values
HMC User Interface – Details Tasks

- Details tasks show all available options on a single panel.

A navigation bar allows quick jumping to the needed section.

Fields are grouped into logical sections that can be expanded and collapsed as needed.
HMC APIs
HMC APIs – Overview

• Application Programmer Interfaces (APIs) are designed to provide an open set of interfaces and a workstation platform for system management application providers
  • They specify how you can write your own components to interact with an HMC and the objects it manages

• The HMC provides three types of APIs
  • Common Information Model (CIM) Management Interface
  • Simple Network Management Protocol (SNMP)
  • Web Services API

• Enabling the APIs is done through the Customize API Settings task
HMC APIs – Using CIM

• Enable on the HMC
  • Launch the Customize API Settings task
  • Select the CIM tab
  • Check the Enable checkbox

• Configuring your CIM client
  • HMC only accepts SSL (HTTPS) client connections
  • Use the TCP/IP address of your HMC
  • CIM server listens for requests on TCP port 5989
  • Authenticate using the user name and password of an existing HMC user
  • Ensure user profile is configured to Allow remote access via the web

• Documentation available on IBM Resource Link:
  • SB10-7154-06 System z Common Interface Model Management Interface
HMC APIs – Using SNMP

• Enable on the HMC
  • Launch the Customize API Settings task
  • Select SNMP tab and the Enable checkbox
  • Enter in a community name and network
  • Add an SNMP user and select an access type
  • Enter the TCP/IP address of any SNMP trap servers (traps are sent to TCP port 162)

• Configuring your SNMP client
  • The HMC uses the standard SNMP UDP/TCP port 161
  • Authenticate using the configured community, user name and password

• Documentation available on IBM Resource Link
  • SB10-7030-16: System z Application Programming Interfaces
HMC APIs – Using Web Services API

• Enable on the HMC
  • Launch the **Customize API Settings** task
  • Select the **WEB Services** tab
  • Limit access based on IP as needed
  • Enable access control by user name

• Configuring your API client
  • HMC accepts standard SSL encrypted (HTTPS) requests on TCP port 6794
  • Authenticate using the user name and password
  • Ensure user profile has been configured to **Allow access to management interfaces**

• Documentation available on **IBM Resource Link:**
  • SC27-2626-00a: System z Hardware Management Console Web Services API
HMC APIs – HMC Web Services API Characteristics

• HMC Web Services API is the latest API implementation
  • Includes existing SNMP/CIM function
  • This API is the focus for future evolution
  • Existing SNMP and CIM APIs remaining in place with their existing capabilities, may be extended on a case-by-case basis

• Design based on current industry design practices
  • Requests and responses structured as web services based on REST design patterns
  • Data is represented in Javascript Object Notation (JSON)
  • Status and property change notifications delivered via Java Messaging Services (JMS)
  • HMC provides an embedded JMS broker configured to support API specific use

• HTTP over TCP/IP Sockets is underlying network transport, SSL for connection security
HMC APIs – What is a REST-oriented Web Service?

• REST = Representational State Transfer
  • A style of software interface design
  • Simplifies client – server interactions
  • Introduced in 2000 by Roy Fielding (phD dissertation)
  • Used widely in today’s world wide web services
  • Based on HTTP protocol

• Fundamentals
  • All actions are against a specific resource
  • The resource instance is identified in the HTTP URI for the web service call
  • Type of operation on that resource is specified by using HTTP “method”

• Standard HTTP methods apply across all resources
  • GET – Collect information about a resource
  • POST – Create a new resource, perform other type of operation
  • PUT – Complete update of a resource (all properties)
  • DELETE – Delete a resource

• Generic Examples:
  • To get list of virtual servers: GET /api/virtual-servers
  • To get information about a virtual server: GET /api/virtual-servers/1234
HMC APIs – Request Flow (Simplified Example)

Client Application

Socket Connection
To myhmc:6794 using SSL

HTTP Request
GET /api/ensembles

Socket Connection

HTTP Response
200 OK
content-type: application/json;charset=UTF-8
{"ensembles": [
{
"name": "HMC_R74_ENSEMBLE",
"object-uri": "/api/ensembles/1f7ffb02-de39-11e0-88bd-00215e67351a",
"status": "alternate-communicating"
}
]}

Support Element

Managed Obj Manager, etc.

JSON notation used for request and response bodies
HMC APIs – Javascript Object Notation (JSON)

- Lightweight data interchange format for use between applications
- Much simpler than XML, but still expressive enough
- Used by Google, Yahoo, Web 2.0 applications etc.
- Syntax and tutorials available at [www.json.org](http://www.json.org)
- JSON parsers widely available (e.g., at json.org)
- Becoming the standard notation used with REST-style APIs

```
{
  "cpcs": [
    {
      "name": "R34",
      "object-uri": "/api/cpcs/95780b76",
      "status": "service-required"
    },
    {
      "name": "R32",
      "object-uri": "/api/cpcs/37c6f8a9",
      "status": "operating"
    }
  ]
}
```
HMC APIs – What is Asynchronous Notification?

• Asynchronous Notification
  – WS APIs require remote client to repeatedly pull (poll) from HMC server to keep data current
  – Asynchronous notification prevents need for constant polling from remote client.
  – Allows HMC to push a notification to a remote client about events or state changes on the resource being managed by the HMC/SE e.g. a server is being deactivated

• JMS: Java Message Service
  – Used for asynchronous notification
  – JMS is an industry standard for messaging, based on J2EE
  – HMC embeds Apache ActiveMQ as its JMS provider
  – Non-Java clients can connect to broker as well, using STOMP protocol

1. Client connects to event topic, listens for message
2. Client is notified about event messages
3. Client can use event message data to obtain more details using WS API requests
Client Programming Considerations

- Web Services API design is client platform and client programming language neutral
  - This is a key reason behind the choice of a HTTP/Web Services style
  - Client platform can be Windows, AIX, Linux, zLinux, Mac, or z/OS or…
  - Clients can be written in programming languages like C/C++ or Java, or scripting languages like Python, Perl, etc.
  - No need for install HMC-specific client-side libraries to use APIs

- In choosing a client language, look for the following either as built-ins or available via add-on libraries:
  - Support for HTTP
  - Support for SSL
  - Support for creating and parsing JSON documents
  - (Optionally) Support for JMS connections to ApacheMQ using either OpenWire or STOMP protocols (if asynchronous notification capabilities are to be used)

- Python is a very good choice because support for all of the above is readily available
Getting Started with the API: Samples

- Python sample code is available on ResourceLink: http://www.ibm.com/servers/resourcelink
  Then navigate: Services / API / Web Services API Samples

- Package provides simple logon/logoff test script and a script that demonstrates how to create and delete a virtual server using the WS API

- Samples are based on a sample Python utility library (hmcapilib.py) that demonstrates best practices in using the API
  - Handles repetitive aspects of making API requests: logon, logoff, converting to/from JSON, setting HTTP headers, etc.
  - Includes error checking and capturing of error status/reason on errors
Q&A
The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

- APPN*
- CICS*
- DB2*
- DB2 Connect
e-business logo*
- Enterprise Storage Server*
- ESCON*
- FICON
- FICON Express
- GDPS*
- Geographically Dispersed Parallel Sysplex
- HiperSockets
- HyperSwap
- IBM
- IBM eServer
- IBM ^*
- IBM logo*
- IMS
- InfoPrint*
- Language Environment*
- MQSeries*
- Multiprise*
- NetView*
- On demand business logo
- OS/2*
- OS/390*
- Parallel Sysplex*
- POWER
- PR/SM
- Processor Resource/Systems Manager
- pSeries*
- RACF*
- Resource Link
- RMF
- S/390*
- S/390 Parallel Enterprise Server
- Sysplex Timer*
- TotalStorage*
- VM/ESA*
- VSE/ESA
- VTAM*
- WebSphere*
- zSeries*
- z/Architecture
- z/OS*
- z/VM*
- zSeries Entry License Charge

* Registered trademarks of IBM Corporation

The following are trademarks or registered trademarks of other companies.

- Java and all Java-related trademarks and logos are trademarks of Sun Microsystems, Inc., in the United States and other countries
- Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.
- Red Hat, the Red Hat "Shadow Man" logo, and all Red Hat-based trademarks and logos are trademarks or registered trademarks of Red Hat, Inc., in the United States and other countries.
- SET and Secure Electronic Transaction are trademarks owned by SET Secure Electronic Transaction LLC.

* All other products may be trademarks or registered trademarks of their respective companies.

Notes:
Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

Backup API Slides
HMC WS API and UI Provide Same Level of Function

• Example: Creating a Virtual Server on an IBM Blade

• Regardless of the interface used, this is accomplished through a series of steps:

  - Create virtual server
  - Assign virtual server to workload
  - Activate virtual server

  - Select hypervisor for new virtual server
  - Define virtual server characteristics
  - Define virtual server network connectivity
  - Add storage to virtual server
  - Specify virtual server options

  - Name, description
  - Virtual or dedicated virtual processors
  - Number of virtual processors
  - Amount of memory
  - Select from among defined virtual networks
  - Select from storage resources previously defined to hypervisor
  - Example: Specify the boot device type and instance:
    - Disk
    - Network
    - ISO
HMC WS API and UI Provide Same Level of Function (con’t)

• HMC UI: Steps are accomplished using panels in a wizard-style task
HMC WS API and UI Provide Same Level of Function (con’t)

• WS API: Steps are accomplished by calling management primitives of the API

Note: Function names listed below are conceptual, not the actual API syntax

- Create virtual server
- Assign virtual server to workload
- Activate virtual server
- Select hypervisor for new virtual server
- Define virtual server characteristics
- Define virtual server network connectivity
- Add storage to virtual server
- Specify virtual server options

- Call List Hypervisors function to obtain a list of hypervisors
  •<Invoking application selects desired hypervisor>
- Call Create Virt Server function specifying selected hypervisor as target and basic VS parameters to get base VS created
- Call List Virt Networks function to obtain current virtual networks
  •<Select network>
- Call Add Virt Adapter function specifying new VS as target and virtual network parameters
- Call List Stg Resources function to obtain list of available volumes
  •<Select volume>
- Call Add Virt Disk function specifying new VS as target and selected storage resource
  •<Select boot device>
- Call Update Virt Server function to set boot device
Web Services API Enablement

• WS API is Disabled by default

• Overall On / Off switch and other configuration via a new tab in the existing Customize API Settings task

• API enablement is done separately from enabling remove browser access to HMC

• Installation can also optionally control the IP addresses from which API connections can be made

• When enabled HMC listens for API connections on a different TCP/IP port
Web Services API Access Control

- Connection to API requires authentication under an HMC application login identity
  - All connections to the API specify an HMC user name and password
  - HMC local or LDAP validation of user name and password supposed, same as UI

- New User Profile option controls whether an HMC user can use the API or not

- Individual requests are authorized using the HMC's authorization controls
  - Requests always performed under an HMC user context

Getting Started with the API: Some Script Snippets (using Python)

• API is session-oriented: All requests are made in the context of an API session

• Basic pattern for an API client:
  1. Establish SSL socket connection with HMC
  2. Logon to open an API session
  3. Make requests using that API session
  4. Logoff to close the API session

 1. Establish an SSL socket connection with the HMC:

• Python code snippets illustrating these steps follows…

  # Connect to HMC at address <host> with 300 second request timeout
  conn = httplib.HTTPSConnection(host, 6794, timeout=300)
  conn.connect()
2. Log on to the HMC to open an API session:

```
# Log on to HMC as <userid> with password <password>
logon_req = {"userid": userid, "password": password}
req_body = json.dumps(logon_req)
req_hdrs = {"Content-Type": "application/json"}
conn.request("POST", "/api/sessions", req_body, req_hdrs)
```

```
response = conn.getresponse()
if response.status != 204:
    # If the response provides a body, always read it.
    resp_body = response.read()
if response.status != 200:
    # Handle failure (eg. wrong password)
    raise Exception("Request failed (status: %d)" % response.status)
```

```
# Retrieve session id from response for later use
logon_resp = json.loads(resp_body)
session_id = logon_resp["api-session"]
```
3. Make requests using the API session:

```python
# Issue request for HMC's properties
# Use the session id for the session we just created
req_hdrs = {"X-API-Session": session_id}
conn.request("GET", "/api/console", None, req_hdrs)
response = conn.getresponse()
if response.status != 204:
    resp_body = response.read()
if response.status != 200:
    raise Exception("Request failed (status: %d)" % response.status)

# Convert result JSON into Python objects for processing
console_props = json.loads(resp_body)
print "HMC name is %s." % console_props["name"]
```
4. Log off from the HMC to close the API session:

```python
# Log off from HMC to free session resources
req_hdrs = {"X-API-Session": session_id}
conn.request("DELETE", "/api/sessions/this-session", None, req_hdrs)
response = conn.getresponse()
if response.status != 204:
    resp_body = response.read()
    if response.status != 204:
        raise Exception("Request failed (status: %d)" % response.status)

# On success, no response to process from Logoff
```
Usage Example: zBXStorTool

- Developed by John Goodyear of the IBM Washington Systems Center

- Provides functions that simplify storage administration for zEnterprise zBX:
  - Export storage definitions for entire ensemble or filtered by hypervisor
  - Show relationship between virtual servers and the storage resources they use

- Provides a more comprehensive example of WS API usage

- Available from the Techdocs Library as document # PRS4856:

- Python script and whitepaper with client programming hints and tips
Usage Example: System z Mobile Application Proof of Concept

- Proof of Concept Mobile Application for monitoring and controlling a zEnterprise system from a mobile device
- Allows person on the IT floor to grab customizable subset of information about the machine
Mobile Application Proof of Concept – Under the Hood

- Mobile device communicates directly with HMC using the new Web Services API. No intermediate management server needed.

- RESTful orientation of the API makes this easy to do using standard application capabilities on these types of devices.