

ClearPass Extension BlackBerry Unified Endpoint Manager



a Hewlett Packard
Enterprise company

ClearPass

ClearPass Extension – Blackberry Unified Endpoint Manager

TechNote

Change Log

Version	Date	Modified By	Comments
0.1	May 2017	Danny Jump	Initial Draft Version for internal review
0.2	June 2017	Robert Filer	Wordsmith and minor edits
1.0	June 2017	Danny Jump	First Published Version

Copyright

© Copyright 2017 Hewlett Packard Enterprise Development LP.

Open Source Code

This product includes code licensed under the GNU General Public License, the GNU Lesser General Public License, and/or certain other open source licenses. A complete machine-readable copy of the source code corresponding to such code is available upon request. This offer is valid to anyone in receipt of this information and shall expire three years following the date of the final distribution of this product version by Hewlett-Packard Company. To obtain such source code, send a check or money order in the amount of US \$10.00 to:

Hewlett-Packard Company
Attn: General Counsel
3000 Hanover Street
Palo Alto, CA 94304
USA

Please specify the product and version for which you are requesting source code. You may also request a copy of this source code free of charge at HPE-Aruba-gplquery@hpe.com.

Contents

Introduction and Overview.....	5
Software Requirements	5
ClearPass Installation and Deployment Guide.....	5
ClearPass Extensions.....	6
Pictorial View of the Integration.....	6
UEM Extension Installation	6
Create an Extension API Operator Profile.....	7
Create an API Client	8
Generate an Access Token.....	9
Go to the Extension APIs	10
Install the Extension	12
Query Extension after Installation	13
UEM Extension Configuration	15
Collecting Information to Configure the ClearPass UEM Extension	15
Creating Roles in UEM to restrict communication	16
Configure the Extension.....	18
Starting the Extension	19
Verify the Extension is Running	20
Troubleshooting the Extension	20
Configuring ClearPass Policy Manager.....	21
Add HTTP Authorization Source	21
Using data from UEM in a ClearPass Enforcement Policy	23
Appendix A – BlackBerry UEM Authentication Source	24
Appendix B – Additional Diagnostics / Support	25
Extension Service.....	25
Extension Logs/Debugging.....	25
Accessing Extension logs using ‘Collect Logs’.....	27
Appendix C – Considerations for Installing in a Cluster.....	28
Extension Installation Procedure for a Subscriber	28
Extension IDs in a Cluster	29

Figures

Figure 1: Pictorial view of the ClearPass and BlackBerry Solution.....	6
Figure 2: Duplicate API Guest Operator profile.....	7
Figure 3: Modifying Operator Profile permissions.....	8
Figure 4: Creating an API Client.....	9
Figure 5: Generate the Access Token	9
Figure 6: API Explorer UI	10
Figure 7: Accessing the Store API.....	10
Figure 8: Checking the Extension store for a particular Extension ID	11
Figure 9: Details of the Extension	11
Figure 10: Installing the Extension directly from the Extension store.....	12
Figure 11: Extension Status	13
Figure 12: Checking Extension Installation Progress.....	14
Figure 13: Response to check on progress of Extension installation	14
Figure 14: Get Extension Configuration	15
Figure 15: Response to a request for the current Extension configuration	15
Figure 16: UEM Role Summary.....	16
Figure 17: Creating the Admin Group with restricted permissions.....	17
Figure 18: BlackBerry UEM Directory Connection Configuration	17
Figure 19: Setting the configuration in the Extension	18
Figure 20: HTTP 204 response to the configuration PUT	18
Figure 21: Starting the extension.....	19
Figure 22: Expected HTTP response to InstanceStart	19
Figure 23: Detailed information on the running extension	20
Figure 24: Getting Debug Logs from the extension.....	20
Figure 25: Adding an HTTP authorization source	21
Figure 26: Adding HTTP authorization source credentials and Base URL	21
Figure 27: Adding HTTP authorization source query string and returned field definitions	22
Figure 28: Example of an Enforcement Policy utilizing attributes returned from BlackBerry UEM	23
Figure 29: Checking on extension service and how to start/stop the service	25
Figure 30: Changing logLevel to DEBUG	25
Figure 31: Turning on Debug logging for an extension.....	26
Figure 32: Accessing Logs in an extension from API Explorer UI	26
Figure 33: An example of the logs from the extension in the API Explorer UI.....	27
Figure 34: Extension logs location in 'Collect Logs' diagnostic GZ file.....	27
Figure 35: Subscriber with no "Generate Key" option.....	28
Figure 36: Generate API Key in Publisher and copy it	29
Figure 37: Entering the HTTP Authorization key into the Subscriber	29

Introduction and Overview

This TechNote describes how to use ClearPass Policy Manager and the ClearPass Unified Endpoint Management Extension to interoperate with the BlackBerry on-premise or cloud-deployed Unified Endpoint Management Extension [UEM]. UEM is the new device management platform from BlackBerry, previously known as BlackBerry Enterprise Server or BES where they have streamlined the management of mobile [Android, BlackBerry, iOS, Windows Phone] and desktop operating system such as macOS and Windows 10. UEM is the new platform that also enables the consolidation of Good technologies and extends the BES platform to MDM, MAM and MCM.

Software Requirements

The minimum software version required is ClearPass 6.6.1. At the time of writing, ClearPass 6.6.5 is the latest available and recommended release. Any subsequent ClearPass software release will support this integration. ClearPass runs on either hardware appliances with pre-installed software, or as a Virtual Machine on the hypervisors shown below. Hypervisors that run on a client computer such as VMware Player are not supported.

- VMware ESXi 5.0, 5.1, 5.5, 6.0, 6.5 or higher
- Microsoft Hyper-V Server 2012 or 2016 R2
- Hyper-V on Microsoft Windows Server 2012 or 2016 R2
- KVM on CentOS 6.6, 6.7, or 6.8.
- BlackBerry UEM 12.6 MR1



The API's utilized in the UEM 12.6 MR1 existed in UEM 12.5. At this time, only version 12.6 MR1 has been tested and certified.

ClearPass Installation and Deployment Guide

This document assumes your ClearPass environment is already configured and operational. If assistance with basic deployment is required, refer to the following deployment guide:
http://www.arubanetworks.com/techdocs/ClearPass/Aruba_DeployGd_HTML/Default.htm



www.arubanetworks.com

3333 Scott Blvd

Santa Clara, CA 95054

Phone: 1-800-WIFI-LAN (+800-943-4526)

Fax 408.227.4550

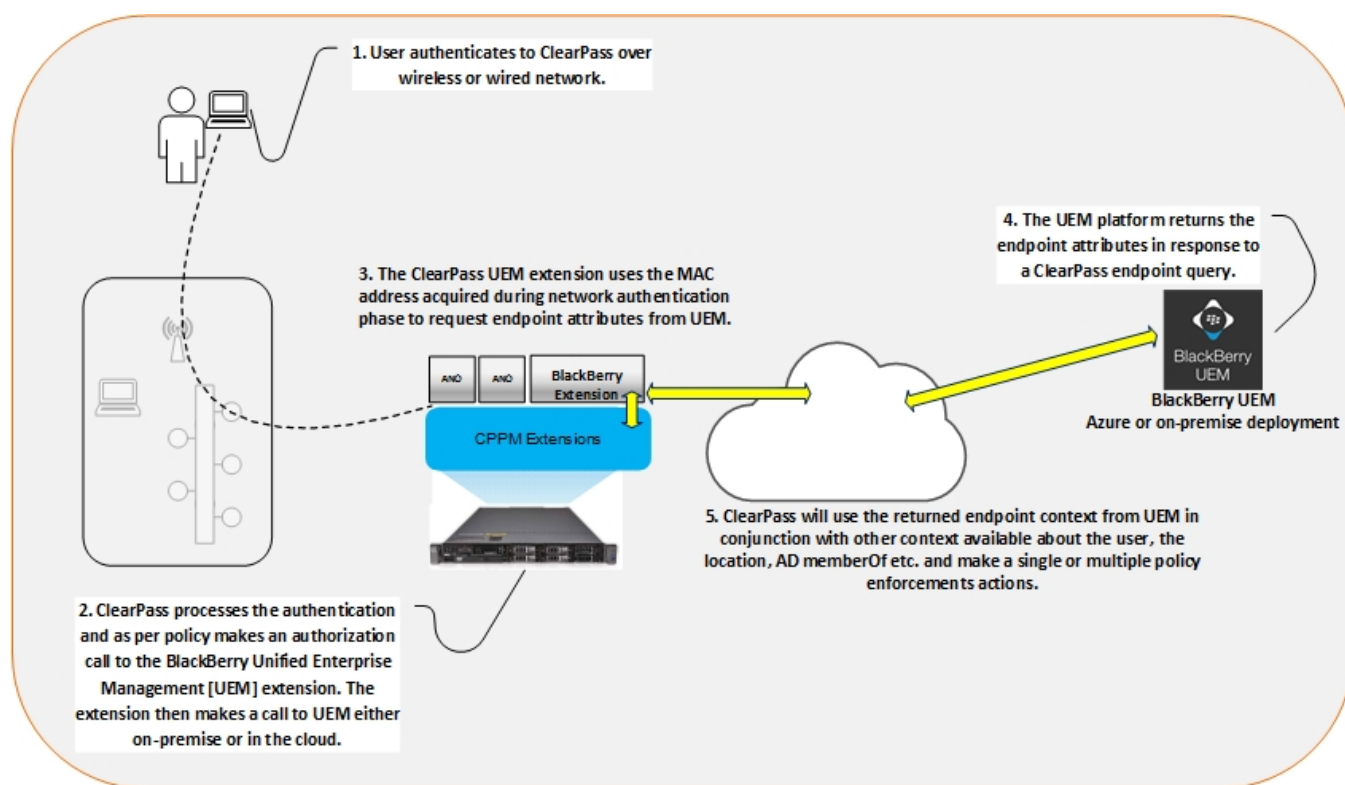
ClearPass Extensions

The integration between ClearPass Policy Manager and UEM is driven through a ClearPass capability known as Extensions, a sub-component of the ClearPass Exchange Integration framework. ClearPass Extensions are micro-services running on top of the base ClearPass platform. These micro-services enable Aruba to deliver new features outside of the main software release cycle and facilitate a faster time to market for specific features and integrations. Configuration and control of ClearPass Extensions is accomplished through the ClearPass REST APIs which are covered later in this document.

Pictorial View of the Integration

Below is a pictorial view of the integration components between ClearPass Policy Manager and BlackBerry UEM. In the below diagram, BlackBerry UEM is shown running in the Cloud. UEM is also supported as an on-premise deployment.

Figure 1: Pictorial view of the ClearPass and BlackBerry Solution



UEM Extension Installation

Installation of the UEM ClearPass Extension is performed via the REST API interface as it is for any Extension. ClearPass REST APIs were initially introduced in CPPM 6.5 and have continued to be enhanced in subsequent releases. Access to the APIs is through the following URL: **https://<ClearPass_IP>/api-docs**. Admin credentials are required to access the API interface.



The APIs to support ClearPass Extensions were initially released in ClearPass version 6.6.

Several components, and multiple steps, are required to complete the ClearPass configuration:

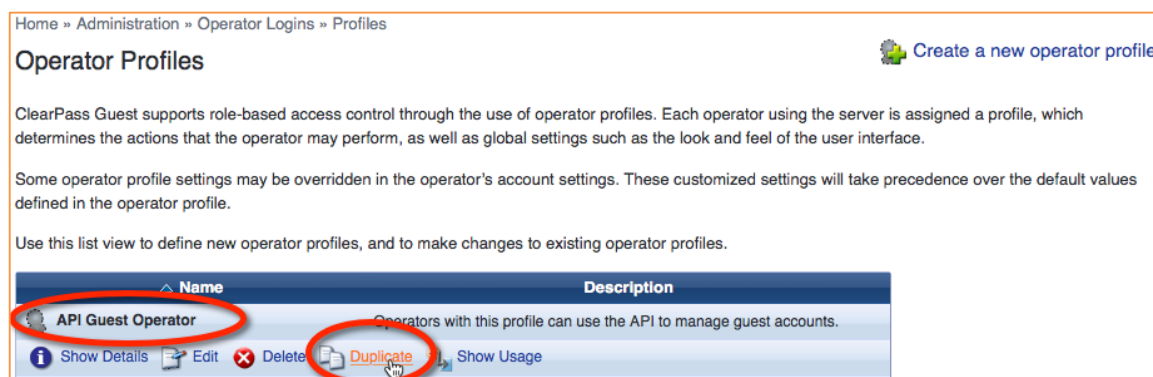
- Extension Installation
- Extension Configuration
- Policy Manager HTTP Authorization Source
- Policy Manager Enforcement Policy and Profiles

The ClearPass extension installation is shown below. The steps to create the Operator Profile and install the actual extension are the same for all Extensions; the only unique change will be the specific Extension ID.

Create an Extension API Operator Profile

Before setting up the API access, configure an **Operator Profile** that will be associated with the configured API client. This new **Operator Profile** will be used in the next section when creating the **API Client**. Login to ClearPass Guest and go to **Administration > Operator Logins > Profiles**. Next, **Click on API Guest Operator** and select **Duplicate**.

Figure 2: Duplicate API Guest Operator profile



ClearPass will copy the profile and name it **API Guest Operator (2)**. **Click Edit** on the new profile and rename it to be **API Extension Profile**. For the **Platform** privilege, change **No Access** to **Custom** and then set all the **Extension** options as shown below (furthest option to the right). Scroll to the bottom of the page and **Click Save**.

Figure 3: Modifying Operator Profile permissions for extensions

Platform Custom...

Select operator permissions for platform administration tasks.

Authentication	<input checked="" type="radio"/> No Access <input type="radio"/> Read Only <input type="radio"/> Full
Operators with the Authentication privilege can configure system authentication settings.	
Content Manager	<input checked="" type="radio"/> No Access <input type="radio"/> Read Only <input type="radio"/> Full
Operators with the Content Manager privilege can add and manage content items.	
Data Retention	<input checked="" type="radio"/> No Access <input type="radio"/> Read Only <input type="radio"/> Full
Operators with the Data Retention privilege can configure the data retention policy.	
Extension Instances - Configuration	<input type="radio"/> No Access <input type="radio"/> Read Only <input checked="" type="radio"/> Full
Operators with this privilege can configure the system's installed extensions.	
Extension Instances - Control	<input type="radio"/> No Access <input checked="" type="radio"/> Allow Access
Operators with this privilege can start, stop, and restart installed extensions.	
Extension Instances - Logs	<input type="radio"/> No Access <input checked="" type="radio"/> Allow Access
Operators with this privilege can read the logs from installed extensions.	
Extension Instances - Manage	<input type="radio"/> No Access <input type="radio"/> Read <input type="radio"/> Read, Write <input checked="" type="radio"/> Read, Write, Delete
Operators with this privilege can create, view and manage the system's installed extensions.	
Extension Store	<input type="radio"/> No Access <input checked="" type="radio"/> Allow Access
Operators with this privilege can query the extension store.	
Import Configuration	<input checked="" type="radio"/> No Access <input type="radio"/> Read Only <input type="radio"/> Full

In the next step, this profile will be used as the **Operator Profile** when generating the API Client.

Create an API Client

The first step in installing and enabling the ClearPass Extension is to create an API Client. An API Client provides authentication and authorization for the REST APIs. Authentication is performed using OAuth2, which is an authorization framework that enables applications to obtain limited access to data over a HTTP service without sharing their private credentials. Log into ClearPass Guest at **<https://<Your-ClearPass-Server>/guest>**.

Navigate to **Guest > Administration > API Services > API Clients** and create an API Client by entering the following:

1. **Client ID:** Free choice
2. **Operator Profile:** API Extension Profile [just created]
3. **Grant Type:** Client credentials

Figure 4: Creating an API Client

Home » Administration » API Services » API Clients

Create API Client

Use this form to create a new API client.

Create API Client	
* Client ID:	Remedy <small>The unique string identifying this API client. Use this value in the OAuth2 "client_id" parameter.</small>
Description:	<div></div> <small>Use this field to store comments or notes about this API client.</small>
Enabled:	<input checked="" type="checkbox"/> Enable API client
* Operator Profile:	API Extension Profile <small>The operator profile applies role-based access control to authorized OAuth2 clients. This determines what API objects and methods are available for use.</small>
* Grant Type:	Client credentials (grant_type=client_credentials) <small>Only the selected authentication method will be permitted for use with this client ID.</small>
Client Secret:	QW3BJWVbPXSBtmn4HTiHiZjcPl3WiPsHN45IDYJBManp <small>Use this value in the OAuth2 "client_secret" parameter. NOTE: This value is encrypted when stored and cannot be displayed again.</small>
Access Token Lifetime:	8 hours <small>Specify the lifetime of an OAuth2 access token.</small>

Create API Client Cancel

Click on **Create API Client** to save and create the API Client.

Generate an Access Token

Click **Generate Access Token** and then launch the **API Explorer**, as highlighted below at the bottom of the image.

Figure 5: Generate the Access Token

Remedy client_credentials 8 hours API Extension Profile
xo+wGyfD1kP+GD15OAGDq6rii1o4oYaNLzzZXJ+/UpYT

Edit Disable Delete **Generate Access Token**

Generated an access token for client_id 'Remedy'.

OAuth2 Access Token	
HTTP Authorization:	Bearer f2672bddff32a5106901a3c8a0de0f65731eb489
Expires:	Monday, 09 January 2017, 11:52 PM
Details:	Show

Developer Quick Start: To make an API call, start with this shell command:
`curl -i -H "Authorization: Bearer f2672bddff32a5106901a3c8a0de0f65731eb489" https://10.2.100.160/api/...`

- To use a HTTP method other than GET, add: `-X POST`
- To send JSON data with curl, add: `-H "Content-Type: application/json" -d '{"field":"value"}'`

Refer to the **API Explorer** for documentation and to try out API calls from your browser.

This will pre-populate the Authorization header in the API Explorer and permit commands to be run directly from the ClearPass UI.

Go to the Extension APIs

Next check whether ClearPass can communicate with the Extension store. This is an important step to ensure that Proxy-Servers and/or Firewalls are not blocking connectivity to the ClearPass Extension Store. Click on **Extension > Store**.

Figure 6: API Explorer UI

API Explorer		
API	Services	Versions
ApiFramework	ApiClient	v1
Authentication	AuthMethod	v1
Extension	Instance, InstanceConfig, InstanceLog, InstanceRestart, InstanceStart, InstanceStop, Store	v1

Under Store, Click **GET /extension/store/{id}**

Figure 7: Accessing the Store API

API Explorer – Extension-v1

[Back to API Explorer](#)

Authorization: Bearer 47a522d3f2deeb0fa26579ae8de3878cc1dde481

Instance : Manage the system's installed extensions

InstanceConfig : Configure an installed extension

InstanceLog : Read logs from an installed extension

InstanceRestart : Restart an installed extension

InstanceStart : Start an installed extension

InstanceStop : Stop an installed extension

Store : Query the extension store

GET /extension/store

GET /extension/store/{id}

Show/Hide | List Operations | Expand Operations

Show/Hide | List Operations | Expand Operations

Show/Hide | List Operations | Expand Operations

Show/Hide | List Operations | Expand Operations

Show/Hide | List Operations | Expand Operations

Show/Hide | List Operations | Expand Operations

Show/Hide | List Operations | Expand Operations

Find a extension in the store

Get details of an extension in the store

Notice that the Authorization header is populated. This is populated from creating the token in the previous step. Next expand the **GET /extension/store/{id}** and paste in the extension's store ID.



NOTE

As an example, let's use the store ID for the BlackBerry UEM Extension. It's a fixed value of **9b4262f7-07c6-4743-976a-192664fdca29**. Click on 'Try it out'!

Figure 8: Checking the Extension store for a particular Extension ID

GET

/extension/store/{id}

Get details of an extension in the store

Implementation Notes

Get details of an extension in the store

Response Class

Model | Model Schema

Store {
id (string, optional): ID of the extension,
name (string, optional): Name of the extension,
version (string, optional): Version number of the extension,
description (string, optional): Description of the extension,
icon_href (string, optional): URL for the extension's icon,
files (object, optional): Describes each of the file IDs required by this extension
}

Response Content Type

application/vnd.extension.v1+json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	9b4262f7-07c6-4743-976a-192664fdca29	URL parameter id	path	string

Error Status Codes

HTTP Status Code	Reason
200	OK
401	Unauthorized
403	Forbidden
404	Not Found
406	Not Acceptable
415	Unsupported Media Type

Try it out!

Hide Response

Request URL



Remember that the store ID **9b4262f7-07c6-4743-976a-192664fdca29** is unique to this version of the BlackBerry UEM Extension. The store ID will change as new versions of the Extension are published.

This will return details for the BlackBerry UEM Extension. This provides assurance that the correct authority is configured to allow access to the Store, and that this is the correct ID for the extension being installed.

Figure 9: Details of the Extension

Response Body

```
{  
  "id": "9b4262f7-07c6-4743-976a-192664fdca29",  
  "name": "bes-authsource",  
  "version": "1.0.0",  
  "description": "BES Authorization Source",  
  "icon_href": "https://us.blackberry.com/content/dam/blackberry-com/images/support/Apps/support-bbm-icon-large.jpg",  
  "_links": {  
    "self": {  
      "href": "https://10.2.100.160/api/extension/store/9b4262f7-07c6-4743-976a-192664fdca29"  
    }  
  }  
}
```

Blackberry UEM Extension

Install the Extension

After checking for extension visibility, the next step is to install it. Under **Instance > POST /extension/instance**, paste in the body as shown below:

{"state": "stopped", "store_id": "9b4262f7-07c6-4743-976a-192664fdca29"} and then **Click** on 'Try it out!'

Figure 10: Installing the Extension directly from the Extension store

POST /extension/instance

Install an extension

Implementation Notes

Installs an extension from the extension store.

The query parameter `async=true` may be provided to make this a synchronous operation. In this case, the operation produces JSON status updates during the operation using HTTP chunked transfer encoding.

The default is asynchronous operation. To check on the progress of the operation, use `GET /extension/instance/{id}`.

Response Class

Model | Model Schema

Instance {

`id (string, optional):` ID of the extension instance,

`state (string, optional) =` ['preparing' or 'downloading' or 'stopped' or 'running' or 'failed']: Current state of the extension,

`state_details (string, optional):` Additional information about the current state of the extension,

`store_id (string, optional):` ID of the extension in the store,

`name (string, optional):` Name of the extension,

`version (string, optional):` Version number of the extension,

`description (string, optional):` Description of the extension,

`icon_href (string, optional):` URL for the extension's icon,

`hostname (string, optional):` Hostname assigned to the extension,

`network_ports (array[InstanceNetworkPort], optional):` List of network ports provided by the extension,

`extension_hrefs (array[InstanceHref], optional):` List of URLs provided by the extension,

`files (object, optional):` Map of extension file IDs to local content items, with 'public' or 'private' prefix,

`internal_ip_address (string, optional):` Internal IP address of the extension

InstanceNetworkPort {

`description (string, optional):` Description of the service provided on this port,

`protocol (string) =` ['tcp' or 'udp']: Network protocol,

`host_port (integer):` Port number for the server,

`extension_port (integer):` Port number internal to the extension

InstanceHref {

`description (string, optional):` Description of the URL,

`href (string):` Server-relative URL path

}

Response Content Type

application/vnd.extension.v1+json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
body	<div><div><pre>{ "state": "stopped", "store_id": "9b4262f7-07c6-4743-976a-192664fdca29" }</pre></div></div>	Extension in the store	body	<div><div>Model Model Schema</div><div><pre>{ "state": "", "store_id": "", "files": "object" }</pre></div></div>

This returns an **ID** (Note: this is different from the store ID. It will be referred to as the run-time ID). Make a copy of this ID as it will be required later.



The below shows the state of the Extension as **preparing**. This indicates the extension is in the process of being downloaded. A typical installation should take just a few seconds to complete.

Figure 11: Extension Status

```
Response Body

{
  "id": "c7edd79e-a9e5-4982-a02d-577454a089ac",
  "state": "preparing",
  "state_details": "",
  "store_id": "9b4262f7-07c6-4743-976a-192664fdca29",
  "name": "",
  "version": "",
  "description": "",
  "icon_href": "",
  "_links": {
    "self": {
      "href": "https://10.2.100.160/api/extension/instance/c7edd79e-a9e5-4982-a02d-577454a089ac"
    }
  }
}
```

From the above, you can see the ID is **c7edd79e-a9e5-4982-a02d-577454a089ac**. This ID (the run-time ID) will be used to query and configure the extension.



Your run-time ID will differ from the one used here. It is generated per installation and is unique to each and every installation.

Query Extension after Installation

Under **Instance > GET /extension/instance/{id}** paste in the ID from the previous step: **c7edd79e-a9e5-4982-a02d-577454a089ac** and **Click 'Try it out!'**. Remember, this ID will be unique for every installation.

Figure 12: Checking Extension Installation Progress

GET `/extension/instance/{id}` Get details of an installed extension

Implementation Notes
Get details of an installed extension

Response Class
[Model](#) | [Model Schema](#)

Instance {
id (string, optional): ID of the extension instance,
state (string, optional) = ['preparing' or 'downloading' or 'stopped' or 'running' or 'failed']: Current state of the extension,
state_details (string, optional): Additional information about the current state of the extension,
store_id (string, optional): ID of the extension in the store,
name (string, optional): Name of the extension,
version (string, optional): Version number of the extension,
description (string, optional): Description of the extension,
icon_href (string, optional): URL for the extension's icon,
hostname (string, optional): Hostname assigned to the extension,
network_ports (array[InstanceNetworkPort], optional): List of network ports provided by the extension,
extension_hrefs (array[InstanceHref], optional): List of URLs provided by the extension,
files (object, optional): Map of extension file IDs to local content items, with 'public:' or 'private:' prefix,
internal_ip_address (string, optional): Internal IP address of the extension
}
InstanceNetworkPort {
description (string, optional): Description of the service provided on this port,
protocol (string) = ['tcp' or 'udp']: Network protocol,
host_port (integer): Port number for the server,
extension_port (integer): Port number internal to the extension
}
InstanceHref {
description (string, optional): Description of the URL,
href (string): Server-relative URL path
}
Response Content Type `application/vnd.extension.v1+json`

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	<code>c7edd79e-a9e5-4982-a02d-577454a089ac</code>	ID of the extension instance	path	string

Within the Body Response from this **GET** observe the following:

Figure 13: Response to check on progress of Extension installation

```
{
  "id": "c7edd79e-a9e5-4982-a02d-577454a089ac",
  "state": "stopped",
  "state_details": "Created",
  "store_id": "9b4262f7-07c6-4743-976a-192664fdca29",
  "name": "bes-authsource",
  "version": "1.0.0",
  "description": "BES Authorization Source",
  "icon_href": "https://us.blackberry.com/content/dam/blackberry-com/images/support/Apps/support-bbm-icon-large.jpg",
  "hostname": "82e927f1bdac",
  "network_ports": [],
  "extension_hrefs": [],
  "files": [],
  "_links": {
    "self": {
      "href": "https://10.2.100.160/api/extension/instance/c7edd79e-a9e5-4982-a02d-577454a089ac"
    }
  }
}
```

The details of the extension will be displayed (could be **downloading**, etc.). Eventually the state will change to either **stopped** or **failed**. In this example, it is clear that the installation is **created** and **stopped**.

UEM Extension Configuration

To complete the configuration of the BlackBerry UEM extension, there are a number of parameters which need to be collected and supplied to the Extension. Start with looking at the default configuration.

Below is an example of how to retrieve the current configuration of the extension. Under **InstanceConfig > GET /extension/instance/{id}/config**, paste in the run-time ID **c7edd79e-a9e5-4982-a02d-577454a089ac** and **Click 'Try it out!'** to return a copy of the current Extension configuration.

Figure 14: Get Extension Configuration

GET /extension/instance/{id}/config Get the configuration of an installed extension

Implementation Notes
Get the configuration of an installed extension.

An extension's configuration may be an arbitrary JSON object. Check the documentation for the extension to determine the settings that may be configured.

Response Class
Model | Model Schema
InstanceConfig {
}

Response Content Type: application/json

Parameter	Value	Description	Parameter Type	Data Type
id	<input type="text" value="c7edd79e-a9e5-4982-a02d-577454a089ac"/>	ID of the extension instance	path	string

Figure 15: Response to a request for the current Extension configuration

```
{
  "logLevel": "INFO",
  "verifySSLCerts": true,
  "besEndpoint": "https://127.0.0.1:8095",
  "tenantExternalId": "SRP_ID",
  "provider": "LOCAL",
  "domain": "",
  "username": "admin.user1",
  "password": "admin.password1"
}
```

The above example shows the default shipping configuration of the BlackBerry UEM extension. This will need to be modified as mentioned above to match your BlackBerry UEM deployment.

Collecting Information to Configure the ClearPass UEM Extension

Installation, Deployment and Configuration of BlackBerry UEM is beyond the scope of this document, but to get the basics of a UEM Server running you will have already applied for and received your SRP-ID.

BlackBerry has online resources for the deployment and configuration here:

<https://help.blackberry.com/en/blackberry-uem/12.6/>

Based upon the default configuration, a number of attributes are required. The **logLevel** and **verifySSLCerts** should be left at their default values unless advised by TAC. This leaves the following:

- **besEndpoint**
- **username**
- **password**
- **provider**
- **tenantExternalId**

Of these remaining attributes, **besEndpoint** should be set to the FQDN or IP address of the BES UEM server.

The **username** and **password** reflect the credentials the Extension will use to communicate with the BlackBerry UEM Server.

The **provider** attribute refers to the authentication source of the above credential being used. This can be one of three values, **LOCAL**, **LDAP** or **AD**. If using **AD** then you need to update the **domain** attribute shown above in the default configuration with your appropriate setting.

If using **LDAP** setting of the **domain** attribute it's optional. If using **LOCAL** then "domain" is ignored.

Creating Roles in UEM to restrict communication

The account created and configure and used to communicate between the ClearPass Extension is a UEM application account. As discussed above this account can be a LOCAL, AD or LDAP account. If you use a LOCAL UEM account then the account needs minimum access rights as shown below.

When signed into the UEM console as a Security Administrator create a role to limit the capabilities of the account used to communicate with UEM.

Go to **Settings -> Administrators -> Roles** [click on the RHS Icon] to create a new role

An example of our role is shown below. It shows one single user assigned to the role-group.

Figure 16: UEM Role Summary

The screenshot shows the UEM console interface. On the left, a sidebar contains navigation links: Dashboard, Users, Groups, Policies and profiles, Apps, Audit, Settings (highlighted with a red box), and Administrators (highlighted with a red box). Under 'Settings', there are sub-links: General settings, External integration, App management, Self Service, and Administrators (highlighted with a red box). Under 'Administrators', there are sub-links: Roles (highlighted with a red box), Users, and Groups. The main content area is titled 'Roles' and contains a table with the following data:

Name	Description	Number of users	Ranking
Security Administrator	Full permissions to manage the BlackBerry Enterprise Solution. Only administrators assigned this role can create and edit roles.	2	↓
Enterprise Administrator	Permissions to perform advanced administrative tasks.	0	↑ ↓
Senior HelpDesk	Permissions to perform intermediate administrative tasks.	0	↑ ↓
Junior HelpDesk	Permissions to perform basic administrative tasks.	0	↑ ↓
cppm-comms	role for cppm-communications	1	↑

At the bottom of the table are 'Cancel' and 'Save' buttons. A red arrow points to a plus icon in the top right corner of the Roles table.



This TechNote shows the use of a LOCAL account configured inside of the UEM application, not to be confused with a Windows LOCAL account.

When creating the role, set the minimum required permissions.

Figure 17: Creating the Admin Group with restricted permissions

Role

Roles
Users
Groups

Settings

Name*

cppm-comms

Description

role for cppm-communications

Group management

Specify the groups that the administrator can manage. To manage users that do not belong to a group, administrators must have permission to manage all groups and users.

☒ All groups and users
☐ Selected groups only

Users and Devices

The view permission must be selected to display the Users and Devices tab.

☒ View users and activated devices

☐ Create users
☐ Edit users
☐ Assign user roles
☐ Delete users
☐ Export user list
☐ Generate an activation password and send email
☐ Specify an activation password

Once the role is created, assign this to your user. This is the user that will be configured inside the ClearPass Extension and will be used to authenticate communication between the ClearPass UEM Extension and the BlackBerry UEM Server.

If you need to integrate BlackBerry UEM with the underlying Windows server/domain then under **Settings -> External Integration -> Company Directory** you can integrate UEM with either Microsoft AD or LDAP.

Figure 18: BlackBerry UEM Directory Connection Configuration

Apple Push Notification
Apple Device Enrollment Program
KNOX Mobile Enrollment
Company directory
Android for Work
SMTP server

Company directory connection ?

Configure the connection to Microsoft Active Directory or LDAP to import directory users and synchronize directory groups.

+ Add a Microsoft Active Directory connection
+ Add an LDAP connection

The remaining attribute, **tenantExternalId** is used to uniquely identify the deployed UEM Server instance. In BlackBerry speak it's called the Server Routing Protocol ID or SRP-ID for short. This is a value supplied directly by BlackBerry as you contract with them to use the BlackBerry MDM framework and services.

Once you have collected/generated all of the necessary attributes above, you need to update the Extension configuration as shown in the following steps.

Configure the Extension

Under **InstanceConfig > PUT /extension/instance/{id}/config** paste in the run-time ID **c7edd79e-a9e5-4982-a02d-577454a089ac**, copy and paste the configuration collected in the previous steps to the PUT method and **Click 'Try it out!'**. Remember this ID is unique to the example in this document, yours will differ.



If changing the configuration, you should restart the extension via **InstanceRestart > POST /extension/instance/{id}/restart**

Figure 19: Setting the configuration in the Extension

PUT /extension/instance/{id}/config Set the configuration of an installed extension

Implementation Notes
Set the configuration of an installed extension

Response Class
Model | Model Schema
InstanceConfig {
}
}

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	c7edd79e-a9e5-4982-a02d-577454a089ac	ID of the extension instance	path	string
body	<pre>{ "logLevel": "INFO", "verifySSLCerts": true, "besEndpoint": "https://my_ip_address_of_UEM:8095", "tenantExternalId": "S12345678", "provider": "LOCAL", "username": "local_user", "password": "local_user_password" }</pre>		body	Model Model Schema InstanceConfig { } }

Parameter content type: application/json

Below is the Response to the PUT. A successful result is indicated by response code of 204.

Figure 20: HTTP 204 response to the configuration PUT

Request URL
https://...:443/api/extension/instance/e80b28c8-743a-4361-8ece-34704f5438d1/config

Response Body
no content

Response Code
204

Starting the Extension

Under **InstanceStart** > **POST /extension/instance/{id}/start** paste in the extension ID and **Click** on **'Try it out!'**.

Figure 21: Starting the extension

POST /extension/instance/{id}/start

Implementation Notes
Start an installed extension

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	b4240b5d-93a5-445e-846c-1dbf8c30f8be	ID of the extension instance	path	string

Error Status Codes

HTTP Status Code	Reason
204	No Content
401	Unauthorized
403	Forbidden
404	Not Found
406	Not Acceptable
415	Unsupported Media Type
422	Unprocessable Entity

Try it out!

A successful response is indicated by a 204 result as shown below.

Figure 22: Expected HTTP response to InstanceStart

Request URL

https://10.2.100.160:443/api/extension/instance/b4240b5d-93a5-445e-846c-1dbf8c30f8be/start

Response Body

no content

Response Code

204

Verify the Extension is Running

Under **Instance > GET /extension/instance/{id}** copy and paste the run-time ID and **Click 'Try it out!'**. The state of the extension should now be "running". An example of the HTTP response is shown below:

Figure 23: Detailed information on the running extension

```
{
  "id": "c7edd79e-a9e5-4982-a02d-577454a089ac",
  "state": "running",
  "state_details": "Started 12 seconds ago",
  "store_id": "9b4262f7-07c6-4743-976a-192664fdca29",
  "name": "bes-authsource",
  "version": "1.0.0",
  "description": "BES Authorization Source",
  "icon_href": "https://us.blackberry.com/content/dam/blackberry-com/images/support/Apps/support-bbm-icon-large.jpg",
  "hostname": "82e927f1bdac",
  "network_ports": [],
  "extension_hrefs": [],
  "files": [],
  "internal_ip_address": "172.17.0.12",
  "_links": {
    "self": {
      "href": "https://10.2.100.160/api/extension/instance/c7edd79e-a9e5-4982-a02d-577454a089ac"
    }
  }
}
```



The "internal_ip_address" of the extension will be set by the extension service. This will be used for configuring the authorization source later in Policy Manager.

Troubleshooting the Extension

Under **InstanceLog > GET /extension/instance/{id}/log**, paste in the run-time ID. Enter a value for "tail", e.g. 100 will show the last 100 lines of output and then **Click on 'Try it out!'**. Note that other settings are applicable when getting logs, e.g. timestamps.

Figure 24: Getting Debug Logs from the extension.

GET /extension/instance/{id}/log Get the log output from an installed extension

Implementation Notes
Get the log output from an installed extension

Response Class
Model | Model Schema
InstanceLog {
 log (array[string], optional): Log messages generated by the extension instance
}

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	e80b28c0-743a-4361-8ece-34704f5438d1	ID of the extension instance	path	string
stdout	true (default)	Include extension's standard-output messages	query	boolean
stderr	true (default)	Include extension's standard-error messages	query	boolean
since	0	Specify a UNIX timestamp to only return log entries since that time	query	integer
timestamps	false (default)	Prefix every log line with its UTC timestamp	query	boolean
tail	all	Return this number of lines at the end of the logs, or "all" for everything	query	string

Configuring ClearPass Policy Manager

To complete the configuration, configure an authorization source within ClearPass. With BlackBerry UEM as an authorization source, ClearPass can check with UEM to see if the device is enrolled, managed, and in a valid compliance state before allowing it to connect. A common use case for ClearPass is to check the version of the installed operating system on the endpoint as the basis for applying specific access policy, if a known endpoint OS has recently been exposed to a OS vulnerability. Another popular use case is to use the ownership attribute to differentiate between a Corporate or Privately {BYOD} owned device. Finally, using context to know if the device is encrypted might be utilized to restrict access to financial applications for a user with an unencrypted device. These and/or other contextual attributes can be used to evaluate an endpoint at the time of network authentication.

Add HTTP Authorization Source

The first step is to add the authorization source. Under **Configuration > Authentication > Sources**, click **Add**.

Figure 25: Adding an HTTP authorization source

aruba ClearPass Policy Manager

Configuration » Authentication » Sources » Add - BlackBerry-authZ-source

Authentication Sources - BlackBerry-authZ-source

Summary General Primary Attributes

Name: BlackBerry-authZ-source

Description: BlackBerry authZ source

Type: HTTP

Use for Authorization: ☐ Enable to use this Authentication Source to also fetch role mapping attributes

Authorization Sources:

Backup Servers Priority:

Add Backup Remove

Click on Next. This will advance to the Primary Tab. Provide the connection details as mandated.



The Base URL IP address is what you captured in Figure 21 above.

Figure 26: Adding HTTP authorization source credentials and Base URL

Configuration » Authentication » Sources » Add - BlackBerry-authZ-source

Authentication Sources - BlackBerry-authZ-source

Summary General Primary Attributes

Connection Details

Base URL: http://172.17.0.11

Login Username: notused

Login Password:



It's mandated that a Login Username and Password be entered, but they are not used and can be set to anything you wish.

Click on Next. This will advance you to the Attributes Tab where you need to provide the authorization attributes. **Click on 'Add More Filters'.** Provide a Name for the filter and then a Filter Query. It's extremely important that the Filter Query is defined correctly. This is the query string that is sent to the BlackBerry UEM Extension asking for context about the endpoint. The query is indexed off the MAC Address of the authenticating endpoint. For ease in copying, the Filter Query is provided here.

?macAddress=%{Connection:Client-Mac-Address-Colon}

Next build out the definitions of the attributes that will be returned from the Filter Query. These attributes will subsequently be used within ClearPass policy evaluation and ultimately the enforcement policy applied. Below is a list of all the available fields. The list of fields you require could very likely just be a subset of that shown below. Only define what you need to expose in your enforcement policy.

Figure 27: Adding HTTP authorization source query string and returned field definitions

Name	Alias Name	Data type	Enabled As
1. uidid	uidid	String	Attribute
2. serialNumber	serialNumber	String	Attribute
3. isFound	isFound	Boolean	Attribute
4. displayName	displayName	String	Attribute
5. compromised	compromised	Boolean	Attribute
6. mdmEnabled	mdmEnabled	Boolean	Attribute
7. encryptionEnabled	encryptionEnabled	Boolean	Attribute
8. lastContactDate	lastContactDate	Date-Time	Attribute
9. macAddress	macAddress	String	Attribute
10. manufacturer	manufacturer	String	Attribute
11. model	model	String	Attribute
12. osVersion	osVersion	String	Attribute
13. outOfCompliance	outOfCompliance	Boolean	Attribute
14. ownership	ownership	String	Attribute
15. username	username	String	Attribute
16. error	error	String	Attribute
17. Click to add...			

Once the HTTP authorization source is defined you can use the returned attributes in your policy processing. Several options on how to use the results from the authorization query in an enforcement policy are shown below.



A copy of the above authorization source in XML format is in Appendix A.

Using data from UEM in a ClearPass Enforcement Policy

Multiple use cases exist for how the data that is returned from UEM can be used in your policy enforcement. In the example below, multiple checks are performed. Your enforcement policy will likely differ as required.

1. The first check is to ensure this is a known device and thus under management. This rule is looking for a boolean response of false, meaning this is **not** a managed device and treated accordingly.
2. If the device is not in compliance then apply an out_of_Compliance policy.
3. If the device is Corporately issued and managed, also ensure the device is encrypted. If all the above checks pass then in this example apply a privileged access policy.
4. Check the device has checked in to the BlackBerry UEM management platform in the last 24-hours. If not, then accordingly the health/posture of the endpoint can't be determined.
5. As a final check, as long as the device is known, thus managed, set a basic-level of access.

Figure 28: Example of an Enforcement Policy utilizing attributes returned from BlackBerry UEM

Configuration » Enforcement » Policies » Edit - BlackBerry Enforcement Policy	
Enforcement Policies - BlackBerry Enforcement Policy	
Summary	Enforcement
Rules	
Enforcement:	
Name:	BlackBerry Enforcement Policy
Description:	
Enforcement Type:	RADIUS
Default Profile:	[Deny Access Profile]
Rules:	
Rules Evaluation Algorithm:	First applicable
Conditions	Actions
1. (Authorization:BlackBerry-authZ-source:isFound EQUALS false)	BlackBerry endpoint unknown
2. (Authorization:BlackBerry-authZ-source:outOfCompliance EQUALS true)	BlackBerry endpoint out_of_Compliance
3. AND (Authorization:BlackBerry-authZ-source:isFound EQUALS true) AND (Authorization:BlackBerry-authZ-source:encryptionEnabled EQUALS true) AND (Authorization:BlackBerry-authZ-source:ownership NOT_EQUALS BYOD)	BlackBerry endpoint known_and_encrypted secure-level
4. (Authorization:BlackBerry-authZ-source:lastContactDate LESS_THAN %{Time Source:minus-24-hours})	BlackBerry endpoint check-enrollment
5. (Authorization:BlackBerry-authZ-source:isFound EQUALS true)	BlackBerry endpoint known basic-level

Different companies will have different views on enforcement policies and how to grant or deny access. The key take-away here is that you are using the authorization attributes received from BlackBerry UEM to drive the policy engine into making and taking different enforcement actions for the device as they authenticate on the network.

Appendix A – BlackBerry UEM Authentication Source

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<TipsContents xmlns="http://www.avendasys.com/tipsapiDefs/1.0">
  <TipsHeader exportTime="Wed May 31 15:52:27 PDT 2017" version="6.6"/>
  <AuthSources>
    <AuthSource description="BlackBerry authZ source" name="BlackBerry-authZ-source" isAuthorizationSource="false" type="HTTP">
      <NVPair value="http://172.17.0.11" name="base_url"/>
      <NVPair value="notused" name="username"/>
      <NVPair value="" name="password"/>
      <Filters>
        <Filter paramValues="" filterQuery="?macAddress=%{Connection:Client-Mac-Address-Colon}" filterName="UEM-filter">
          <Attributes>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="udid" attrName="udid"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="serialNumber" attrName="serialNumber"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="Boolean" aliasName="isFound" attrName="isFound"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="displayName" attrName="displayName"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="Boolean" aliasName="compromised" attrName="compromised"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="Boolean" aliasName="mdmEnabled" attrName="mdmEnabled"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="Boolean" aliasName="encryptionEnabled" attrName="encryptionEnabled"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="Date-Time" aliasName="lastContactDate" attrName="lastContactDate"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="macAddress" attrName="macAddress"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="manufacturer" attrName="manufacturer"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="model" attrName="model"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="osVersion" attrName="osVersion"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="Boolean" aliasName="outOfCompliance" attrName="outOfCompliance"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="ownership" attrName="ownership"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="username" attrName="username"/>
            <Attribute isUserAttr="true" isRole="false" attrDataType="String" aliasName="error" attrName="error"/>
          </Attributes>
        </Filter>
      </Filters>
    </AuthSource>
  </AuthSources>
</TipsContents>
```

Appendix B – Additional Diagnostics / Support

Extension Service

ClearPass Extensions are supported by a new system service that was added in ClearPass 6.6. This service should be running by default.



Restarting this service will affect all deployed and running extensions.

To check on the state and make changes to the service navigate to **Administration > Server Manager > Server Configuration [select your ClearPass node] > Service Control**. You can also start/stop the extension service from here. By default, this service is automatically started.

Figure 29: Checking on extension service and how to start/stop the service

Administration » Server Manager » Server Configuration - cppm6dot6-160					
Server Configuration - cppm6dot6-160 (10.2.100.160)					
System	Services Control	Service Parameters	System Monitoring	Network	FIPS
Service Name		Status		Action	
1.	AirGroup notification service	Running		Stop	
2.	Async DB write service	Running		Stop	
3.	Async network services	Running		Stop	
4.	ClearPass IPsec service	Running		Stop	
5.	DB change notification server	Running		Stop	
6.	DB replication service	Running		Stop	
7.	Extensions service	Running		Stop	

Extension Logs/Debugging

If you have a need to access the logs from inside the extension, you can turn on log collection from the API Explorer. Referencing the configuration previously used, adjust the **"logLevel"** to **"DEBUG"**. Post this using the API Explorer as shown below.



Remember, any change to an extension configuration mandates a restart of the extension.

Figure 30: Changing logLevel to DEBUG

```
{
  "logLevel": "DEBUG",
  "verifySSLCerts": true,
  "besEndpoint": https://my\_ip\_address\_of\_UEM:8095,
  "tenantExternalId": "S12345678",
  "provider": "LOCAL",
  "username": "local_user",
  "password": "local_user_password"
}
```

Figure 31: Turning on Debug logging for an extension

PUT /extension/instance/{id}/config Set the configuration of an installed extension

Implementation Notes
Set the configuration of an installed extension

Response Class
Model | Model Schema
InstanceConfig {
}

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	86400fc9-384e-420d-b1e5-b8aec008fd85	ID of the extension instance	path	string
body	<pre>{ "tokenEndpoint": "https://login.windows.net/d7c8b0de-4cde-40f7-86ec-09740d179124/oauth2/token", "nacEndpoint": "https://fef.msua05.manage.Microsoft.com/StatelessNACService/devices/", "clientId": "48d78b37-c768-4039-a102-1a221bebc2cc", "clientSecret": "bOliNN7uhA1RL9iLG2bi9NzG11of5+IDLx3dg1jAzI=", "resourceUri": "https://api.manage.Microsoft.com/", "verifySSLCerts": true, "logLevel": "DEBUG" }</pre>	body	Model Model Schema InstanceConfig { }	

Once you have configured the extension to capture logs, there are two methods to access them. The first is directly through the API Explorer and the second using the **“Collect Logs”** function.

Figure 32: Accessing Logs in an extension from API Explorer UI

GET /extension/instance/{id}/log Get the log output from an installed extension

Implementation Notes
Get the log output from an installed extension

Response Class
Model | Model Schema
InstanceLog {
 log (array[string], optional): Log messages generated by the extension instance
}

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
id	8bc143a8-6ba4-4b2a-80e3-512fb8d9837e	ID of the extension instance	path	string
stdout	true (default)	Include extension's standard-output messages	query	boolean
stderr	true (default)	Include extension's standard-error messages	query	boolean
since	0	Specify a UNIX timestamp to only return log entries since that time	query	integer
timestamps	false (default)	Prefix every log line with its UTC timestamp	query	boolean
tail	all	Return this number of lines at the end of the logs, or "all" for everything	query	string



You can also turn on timestamps by flipping the timestamps option and optionally limit the number of logs returned to say the last 100 rather than 'all' logs by specifying a number in the tail parameter. By default, all logs are returned with no timestamps.

An example of the output from the UI is below.

Figure 33: An example of the logs from the extension in the API Explorer UI

```
Response Body

{
  "log": [
    "\u001b[32m[2016-10-19 22:08:17.444] [INFO] epo - \u001b[39mServer running at port 80\n",
    "\u001b[36m[2016-10-19 22:09:03.196] [DEBUG] epo - \u001b[39mRequest URL: /?macAddress=42987b054b5c\n",
    "\u001b[36m[2016-10-19 22:09:03.204] [DEBUG] epo - \u001b[39mPo Host/port: 10.2.100.122:8443\n",
    "\u001b[36m[2016-10-19 22:09:03.205] [DEBUG] epo - \u001b[39mPo URL: /remote/aruba.executeQuery?target=EP0LeafNode&where=(eq%20EP0ComputerProperties.NetAddress%20%2242987b054b5c%22)&joinTables=EP0ComputerProperties&%3Aoutput=json\n",
    "\u001b[36m[2016-10-19 22:09:03.725] [DEBUG] epo - \u001b[39mSystem Not Found (42987b054b5c).\n",
    "\u001b[36m[2016-10-19 22:09:43.057] [DEBUG] epo - \u001b[39mRequest URL: /?macAddress=60672001E42A\n",
    "\u001b[36m[2016-10-19 22:09:43.060] [DEBUG] epo - \u001b[39mPo Host/port: 10.2.100.122:8443\n",
    "\u001b[36m[2016-10-19 22:09:43.060] [DEBUG] epo - \u001b[39mPo URL: /remote/aruba.executeQuery?target=EP0LeafNode&where=(eq%20EP0ComputerProperties.NetAddress%20%2260672001E42A%22)&joinTables=EP0ComputerProperties&%3Aoutput=json\n"
  ]
}
```



Remember after collecting logs or turning on DEBUG mode, please ensure you return it back to the INFO level. DEBUG mode should only be enabled under guidance from Aruba TAC.

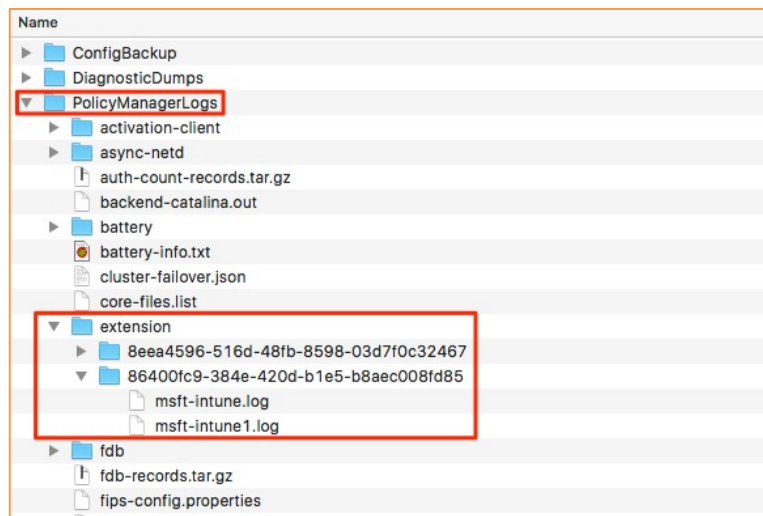
Accessing Extension logs using 'Collect Logs'

In addition to viewing the log messages as shown above, you can also configure the extension to log messages so that they can be collected and examined via the Policy Manager **'Collect Logs'** system function. This is extremely useful for Aruba TAC.

If there is a requirement for Aruba TAC to investigate a system issue, one of the items they regularly ask for is the system logs to aid with their diagnostic investigation. By default the "logLevel" is set to INFO, but TRACE, DEBUG, INFO, WARN, ERROR, or FATAL can also be set. Any of the levels will display the information for the selected state and lower. If INFO is selected, it will show messages for INFO, WARN, ERROR, FATAL.

After the logs have been collected and expanded, you can locate the extension logs in the following location **'PolicyManagerLogs->extension'** as shown below.

Figure 34: Extension logs location in 'Collect Logs' diagnostic GZ file



Appendix C – Considerations for Installing in a Cluster

All previous information in this TechNote applies to a single ClearPass node (publisher-only) deployment. When installing the extension subscriber nodes within a cluster, there are a number of things that should be considered.

Extension Installation Procedure for a Subscriber


When installing an extension on a subscriber, the workflow is a slightly different as there is no process to configure API Clients, generate OAuth2 Access Tokens, etc., as all the configuration is performed on the Publisher.

Installing an extension on a subscriber requires that you use the API Explorer REST interface to install the extension, using the API Explorer requires access to an OAuth2 Access Token. The problem is that on the subscriber node there is no option to generate the Access Token, as shown below.

Figure 35: Subscriber with no "Generate Key" option







Home » Administration » API Services » API Clients

API Clients

 This is a subscriber node, you should go to the [publisher node](#) to make changes.

The API clients you have defined are listed below.

Filter:

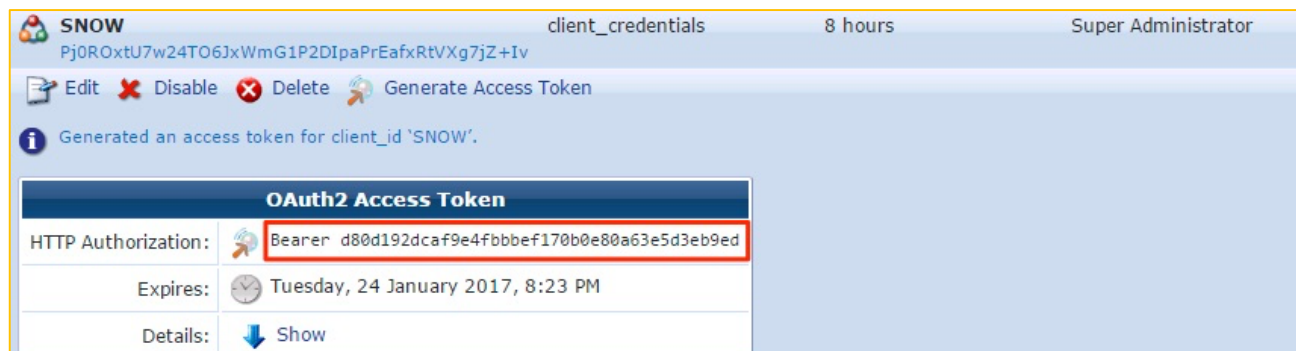
Client ID	Grant Types	Access Token	Operator Profile
 ePO	client_credentials	200 weeks	API Extension Profile
 Microsoft InTune 9U+WbIZ/NfhWA5smRRRig5Nzip/I7pM5ijCo2GFQ9RqQ	client_credentials	8 hours	Super Administrator
 Remedy xo+wGyfd1kP+GD15OAGDq6rii1o4oYaNLzzZXJ+/UpYT	client_credentials	8 hours	API Extension Profile
 SNOW Pj0ROxtU7w24TO6JxWmG1P2DIpaPrEafxRtVXg7jZ+Iv	client_credentials	8 hours	Super Administrator
 Edit			
 ucsapi	client_credentials	8 hours	Super Administrator

No Generate Key



The solution is to configure the API client on the Publisher, then generate an OAuth2 Access Token. Then COPY the HTTP Authorization key from the Publisher and use it on the Subscriber. Ensure you copy the "Bearer" portion of the key as well as shown below.

Figure 36: Generate API Key in Publisher and copy it

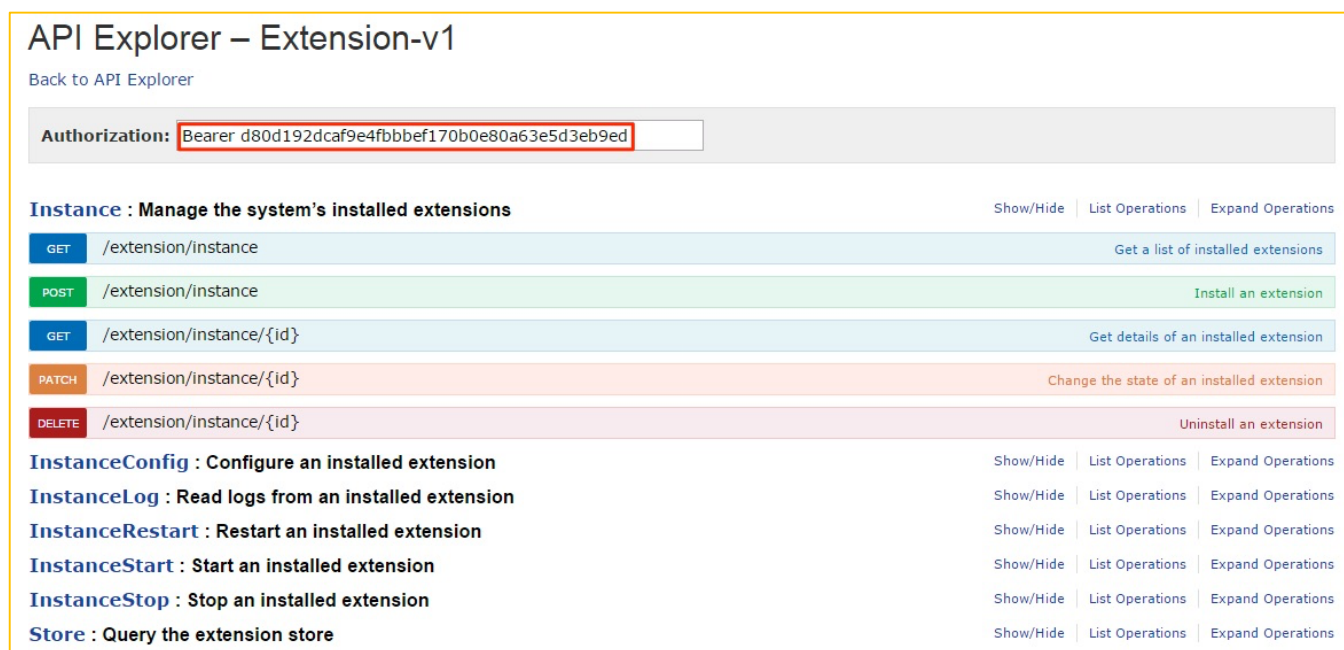


After you have copied the OAuth2 Access Token from the Publisher, go directly to the **https://<subscriber_node>/api-docs** URL on the subscriber.



Ensure you enter the Authorization Header in the API Explorer before using trying to use the functions.

Figure 37: Entering the HTTP Authorization key into the Subscriber



Extension IDs in a Cluster

Note that the store-ID never changes and is specific to a published version of the extension. The **Extension ID** refers to the Store ID of the extension. Installing an extension on five subscriber nodes would result in a different **run-time ID** on each node.