**Breckenridge**

**New UI Newsfeed and Tasks**

**Breckenridge 7.0**

**James Abendroth**

**Version 0.3**

****

*HP Confidential - Need to Know  
©Copyright 2011-2012 Hewlett-Packard Company*

Contents

[Document History 3](#_Toc307478895)

[Overview 4](#_Toc307478896)

[Newsfeed and the Common UI 4](#_Toc307478897)

[Tasks and the Common UI 4](#_Toc307478898)

[Newsfeed API 5](#_Toc307478899)

[Newsfeed REST URIs 5](#_Toc307478900)

[POST URIs 5](#_Toc307478901)

[GET URIs 5](#_Toc307478902)

[DELETE URIs 5](#_Toc307478903)

[Newsfeed Event Data Structures 6](#_Toc307478904)

[Newsfeed API Return Messages 7](#_Toc307478905)

[Success 7](#_Toc307478906)

[Empty Message 7](#_Toc307478907)

[Invalid Status 7](#_Toc307478908)

[Invalid Date/Time 7](#_Toc307478909)

[Invalid Object ID 8](#_Toc307478910)

[Server Error 8](#_Toc307478911)

[Invalid Format 8](#_Toc307478912)

[Database Error 8](#_Toc307478913)

[Tasks API 8](#_Toc307478914)

[Tasks REST URIs 8](#_Toc307478915)

[POST URIs 8](#_Toc307478916)

[GET URIs 9](#_Toc307478917)

[DELETE URIs 9](#_Toc307478918)

[Task Data Structures 9](#_Toc307478919)

[Task API Return Messages 10](#_Toc307478920)

[Success 10](#_Toc307478921)

[Empty Message 10](#_Toc307478922)

[Invalid Status 11](#_Toc307478923)

[Invalid Date/Time 11](#_Toc307478924)

[Invalid Object ID 11](#_Toc307478925)

[Server Error 11](#_Toc307478926)

[Status Codes 11](#_Toc307478927)

[Permissions 12](#_Toc307478928)

[vCenter Events 12](#_Toc307478929)

[Object Removed Event 12](#_Toc307478930)

[Newsfeed and Task Event Maintenance 12](#_Toc307478931)

# Document History

|  |  |  |
| --- | --- | --- |
| **Version** | **Author** | **Notes** |
| 0.1 | James Abendroth | Initial draft |
| 0.2 | James Abendroth | Updated to show handling of all types of vCenter objects, not just hosts |
| 0.3 | James Abendroth | Updates from Zach’s comments |
| 0.4 | Andy Yates | Added object name field |

# Overview

The newsfeed feature of the new Breckenridge 7.0 UI will provide a single location for the user to view alerts and messages related to their systems. The newsfeed will exist in all supported IC4vC plug-in contexts in the vSphere UI.

The tasks feature will display a list of user-initiated tasks that have been performed on a host. The tasks view will also exist in all supported IC4vC plug-in contexts.

# Newsfeed and the Common UI

|  |  |
| --- | --- |
| The common UI will display a newsfeed summary as an app on the main host and cluster screens. The newsfeed summary will display ten of the most recent events in the newsfeed.  Each item in the newsfeed summary will display the following information:   * A status icon indicating the severity of the event (not shown in Figure 1) * Newsfeed event message * A date/time stamp indicating when the event occurred   Clicking the “Show more …” link will open a detailed view of newsfeed information. The detailed view will display all of the events ordered by date. In the detailed view, the user will have the option to sort and filter the newsfeed events by date, severity, and event message. | nf.png  Figure - Newsfeed Summary |

# Tasks and the Common UI

|  |  |
| --- | --- |
| A tasks icon will appear in the common UI header next to the object’s information. Hovering over the tasks icon will display a popup. The popup will list the most current tasks that apply to the object. | tasks.png  Figure - Tasks Icon |

The tasks popup will have a “Show more …” link at the bottom. Clicking the link will open a detailed view of tasks. As with the newsfeed, the user will be able to sort and filter tasks by date, message, and severity.

# Newsfeed API

Common UI plug-in modules must use the newsfeed API in order to send events to the newsfeed. The base URI of the newsfeed is:

https://<hpic4vc\_host>:<UIM\_port>/newsfeed/

## Newsfeed REST URIs

The newsfeed API will support GET, POST, and DELETE requests. GET requests will be used by the common UI to display the newsfeed data. POST and DELETE requests will be used by the plug-ins to send events to the newsfeed and to remove unwanted events.

### POST URIs

**/newsfeed/?moref=<mob\_type>:<mob\_value>&serverGuid=<serverGuid>**

To add an event for a specific object, a plug-in must post a newsfeed event data structure to the /newsfeed/ URI. The request must include the moref and serverGuid query string parameters. For example, if an event is being delivered for *HostSystem:host-53*, the URI would be /newsfeed/?moref=HostSystem:host-53. Newsfeed events can be delivered for host systems, virtual machines, datastores, and other vCenter objects.

Making a POST request without a moref attribute and serverGuid attribute will result in a *400 Bad Request* from UI manager.

### GET URIs

**/newsfeed/?moref=<mob\_type>:<mob\_value>&serverGuid=<serverGuid>**

Performing a get on /newsfeed will return all newsfeed events for a particular vCenter object identified by the moref query string attribute. For example, to obtain a list of newsfeed events for *HostSystem:host-53*, the URI would be /newsfeed/?moref=HostSystem:host-53.

Performing a get on the newsfeed URI and passing a cluster mob will return a list of all events for each host in the cluster.

**/newsfeed/?moref=<mob\_type>:<mob\_value>&serveGuid=<serverGuid>&top=<num>**

Including the top parameter in the query string will limit the number of newsfeed events returned. The events will be sorted by date in descending order and the first <num> events will be returned.

### DELETE URIs

**/newsfeed/?pluginSource=<plugin-id>**

Making a DELETE request with the pluginSource parameter will delete all newsfeed events that were created by a specific plug-in. The pluginSource value corresponds to the pluginSource field in the newsfeed event data structure below. If the pluginSource parameter is not specified, a 404 bad request will be returned.

## Newsfeed Event Data Structures

When posting an event for an object, the following data structure must be used. Note: the events parameter is a list so it is possible to send multiple newsfeed events in a single request.

{

"events": [

{

"objectName": "<string object’s name here>",

"message": "<strings file key here>",

"messageArguments": [],

"eventSource": "<source here>",

"pluginSource": "<source here>",

"status": "<status code here>",

"eventDate": "1319043434.528"

}

]

}

Notes on the events data structure:

* The value of the message field must be a key to a string defined in one of the plug-in strings files. For example, if the strings file defines a key as follows:  
  $.messages = {  
   fanFailed: 'Enclosure fan failed'  
  }  
  Then the message field would contain “messages.fanFailed”. If the message does not match a key in the strings file, the message will be displayed.
* Message arguments are an array of strings. The arguments will be substituted by position for %n tokens in the message field. For example:  
  A message of "Enclosure %0 fan %1 failed" with arguments of ["brkenc1", "2"] would be displayed in the UI as “Enclosure brkenc1 fan 2 failed”.
* Event date must be correctly formatted as shown above or an error will be returned. See the error messages section below. Event date must be sent as a timestamp value.
* The eventSource field is optional. It can be used to give the user more information about the source of the newsfeed event. For example, if an enclosure fan failure happens and an event is sent for a blade, the source can be set to the enclosure name, or, “OA”.
* The pluginSource field indicates which plug-in sent the event to the common UI. If this field is not specified, the event cannot be deleted through the DELETE request.

The data structure returned from a get request for an object will have the following format:

{

"result": {

"<object-id>": [

{

"\_id": "<event-id>",

"objectName": "<object name here>",

"message": "<message here>",

"messageArguments": [],

"source": "<source here>",

"status": "<status code here>",

"formattedStatus": "<status description here>",

"eventDate": "1319043434.528",

"formattedEventDate": "2011-10-19 11:59:20"

},

]

}

}

## Newsfeed API Return Messages

The common UI newsfeed handler will validate messages that it receives from the plug-ins. The message may be rejected if any errors are found. All return messages will have an HTTP status of 200 OK.

### Success

If the newsfeed event was successfully added, the following response will be returned. The list of “ids” contains the unique database IDs for each task that was added. These IDs can be used later to update tasks.

{

"errorCode": 0,

"errorMessage": "The newsfeed event was added successfully."

}

### Empty Message

If the message property is empty, the following error response will be returned:

{

"errorCode": 1,

"errorMessage": "The newsfeed service received a request with no message."

}

### Invalid Status

If the message contains a status code that is not in the *Status Codes* section, the following response will be returned:

{

"errorCode": 2,

"errorMessage": "The newsfeed service received a request with an invalid status code."

}

### Invalid Date/Time

If the message contains an invalid eventDate property, the following response will be returned:

{

"errorCode": 3,

"errorMessage": "The newsfeed service received a request with an invalid date/time."

}

### Invalid Object ID

If the request URI contains an empty object identifier, the following error will be returned:

{

"errorCode": 4,

"errorMessage": "The newsfeed service received a request with an invalid object ID."

}

Note: the object ID is not validated against the vCenter database. It must only not be empty.

### Server Error

If an error occurs on the server while trying to add the message to the database, the following response will be returned:

{

"errorCode": 5,

"errorMessage": "An error occurred while logging the newsfeed event."

}

### Invalid Format

If the object that is sent to the API is not in the correct format, the following response will be returned:

{

"errorCode": 6,

"errorMessage": "The request was not in the correct format."

}

### Database Error

If the UI manager was unable to connect to the database, the following response will be returned:

{

"errorCode": 7,

"errorMessage": "An error occurred while connecting to the database."

}

### Server GUID Error

If no vCenter GUID was supplied, the following error will be returned:

{

"errorCode": 8,

"errorMessage": " The supplied vCenter UUID is invalid."

}

# Tasks API

Common UI plug-in modules must use the tasks API in order to send events to the tasks list. The base URI of the common UI tasks list is:

https://<hpic4vc\_host>:<UIM\_port>/tasks/

Tasks are considered to be user-initiated events that require a separate type of notification than newsfeed events. Examples of tasks are starting and ending firmware update and toggling the UID.

## Tasks REST URIs

The tasks API will support GET, POST, and DELETE requests. GET requests will be used by the common UI to display the tasks. POST requests will be used by the plug-ins to send events to the tasks list. DELETE requests will be used to remove tasks from the database.

### POST URIs

**/tasks/?moref=<mob\_type>:<mob\_value>&serverGuid=<serverGuid>**

Posting a task data structure to /tasks/ will add a task entry for the object specified by the moref query string parameter. For example, if a task entry is being delivered for *HostSystem:host-53*, the URI would be /tasks/?moref=HostSystem:host-53&serverGuid=<server-guid>. Tasks can be delivered for hosts, virtual machines, datastores, and other vCenter objects.

Making a POST request without a moref attribute and serverGuid attribute will result in a *400 Bad Request* from UI manager.

**/tasks/?moref=<mob\_type>:<mob\_value>&serverGuid=<serverGuid>&taskId=<taskId>**

Including a taskId parameter in the query string will perform an update on the task whose \_id field matches the taskId parameter. For example, if a task initially has a start date but no end date, this method can be used to update the end date property of the task.

### GET URIs

**/tasks/?moref=<mob\_type>:<mob\_value>&serverGuid=<serverGuid>**

Performing a get on /tasks/ will return all newsfeed events for a particular object. For example, to obtain a list of tasks for *HostSystem:host-53*, the URI would be /tasks/?moref=HostSystem:host-53&serverGuid=<server-guid>.

Performing a get on the tasks URI and passing a cluster ID will return a list of all tasks for each host in the cluster.

**/tasks/?moref=<mob\_type>:<mob\_value>&serverGuid=<serverGuid>&top=<num>**

Specifying the top parameter in the query string will limit the number of tasks returned in the response. For example, to get the last 10 tasks for host-53 the request would be /tasks/?moref=HostSystem:host-53&serverGuid=<server-guid>&top=10.

### DELETE URIs

**/tasks/?pluginSource=<plugin-id>**

Making a DELETE request with the pluginSource parameter will delete all tasks that were created by a specific plug-in. The pluginSource value corresponds to the pluginSource field in the task data structure below. If the pluginSource parameter is not specified, a 404 bad request will be returned.

## Task Data Structures

The task structure returned from a get request for a specific object will have the following format:

{

"result": {

"<obj-id>": [

{

"\_id": "<database-task-id>",

"taskName": "<strings file key here>",

"taskNameArguments": [],

"startTime": "1319043434.528",

"formattedStartTime": "September 26, 2011 2:41:24 PM PDT",

"completedTime": null,

"formattedCompletedTime": null,

"status": "FAILED",

"userName": "Administrator",

"taskDetails": null,

"taskDetailArguments": []

},

]

}

}

When posting an event for a particular object, the following data structure format should be used. Note: the tasks parameter is an array so multiple tasks can be sent in a single request.

{

"tasks": [

{

"taskName": "<strings file key here>",

"taskNameArguments": [],

"startTime": "1319043434.528",

"completedTime": null

"status": "FAILED",

"userName": "Administrator",

"taskDetails": "<strings file key here>",

"taskDetailArguments": []

},

]

}

Notes on the tasks data structure:

* The value of the taskName and details fields must be a key to a string defined in one of the plug-in strings files. For example, if the strings file defines a key as follows:   
  $.messages = {  
   firmwareStarted: 'Firmware update started.'  
  }  
  Then the taskName field would contain “messages.firmwareStarted”. If the message does not match a key in the strings file, the message will be displayed.
* See the newsfeed data structure notes section for information on message arguments and formatting.
* If a task is in progress, the completedTime property should be set to null. If the field has a string value it will be validated. If it is null, common UI will skip validation.

## Task API Return Messages

### Success

If the task was successfully added, the following response will be returned:

{

"ids": ["4f0def4be46bc71f40000000"],

"error": {

"errorCode": 0,

"errorMessage": "The task was added successfully."

}

}

### Empty Message

If both the taskName and formattedTaskName properties are empty, the following error response will be returned:

{

"errorCode": 1,

"errorMessage": "The task service received a request with no task name."

}

### Invalid Status

If the message contains a status code that is not in the *Status Codes* section, the following response will be returned:

{

"errorCode": 2,

"errorMessage": "The newsfeed service received a request with an invalid status code."

}

### Invalid Date/Time

If any of the date/time properties contain an invalid date/time value, the following response will be returned:

{

"errorCode": 3,

"errorMessage": "The task service received a request with an invalid date/time."

}

### Invalid Object ID

If the request URI contains an invalid object identifier, the following error will be returned:

{

"errorCode": 4,

"errorMessage": "An object with the specified ID was not found."

}

### Server Error

If an error occurs on the server while trying to add the task to the database, the following response will be returned:

{

"errorCode": 5,

"errorMessage": "An error occurred while logging the task."

}

### Invalid Format

If the object that is sent to the API is not in the correct format, the following response will be returned:

{

"errorCode": 6,

"errorMessage": "The request was not in the correct format."

}

### Database Error

If the UI manager was unable to connect to the database, the following response will be returned:

{

"errorCode": 7,

"errorMessage": "An error occurred while connecting to the database."

}

### Server GUID Error

If no vCenter GUID was supplied, the following error will be returned:

{

"errorCode": 8,

"errorMessage": " The supplied vCenter UUID is invalid."

}

# Status Codes

The following status codes can be used for the “status” property. These status codes apply to both the newsfeed and task APIs.

* OK
* WARNING
* FAILED
* INFORMATION
* UNKNOWN

# Permissions

The plug-ins that call the newsfeed and task APIs must have a valid session key in order to add events and tasks. The session key will be validated when the API URLs are invoked.

# Newsfeed and Task Event Maintenance

The UIM database will maintain 60 days worth of events for newsfeed and tasks per host. UIM must monitor the database periodically and remove events that are older than 60 days.

If possible, the maintenance interval should be presented to the user as an option in the common UI. The available intervals should be 30 days, 60 days, 90 days, and 120 days.