Document Identifier: DSP2062

Date: 2022-09-15

Version: 1.0.1

Redfish Firmware Update White Paper

Supersedes: 1.0.0

Document Class: Informational

Document Status: Published

Document Language: en-US

CONTENTS

Foreword

The Redfish Firmware Update White Paper was prepared by the Redfish Forum of the DMTF.

DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. For information about the DMTF, see http://www.dmtf.org.

Acknowledgments

The DMTF acknowledges the following individuals for their contributions to this document:

- Bill Vetter - Lenovo
- John Leung - Intel Corporation
- Michael Raineri - Dell Inc.

1 Introduction

Firmware is the executable binary that a processor or micro-controller executes. A compute system may contain multiple firmware images. Hence, the capability to update these firmware images is important.

Redfish is an hardware management interface standard that is designed to be flexible, extensible, and interoperable. Redfish is composed of an interface specification and resource models. The resource models supported determine the capabilities of the management interface. This document describes the that is used to update the firmware images on a system.

This document helps implementers and clients understand the Redfish firmware update data model and how its used to update firmware images on a system.

2 Methods for firmware update

There are many mechanisms for transferring and installing the firmware on a target device.

2.1 Transferring the image via push and pull

The transfer of the firmware image to the platform can be performed by pushing the firmware image to the system or the system can pull the firmware image from an external location.

In a push firmware request, the firmware image is transferred in the POST request. Redfish specifies an HTTP multi-part format for this type of request.

In a pull firmware request, the location of the firmware image is provided in the POST request. The Redfish service transfers the firmware image from the specified location, before proceeding with the firmware update.

While the contents and format of the image are out of scope of this document, the firmware image transferred to the system may be a package consisting of multiple images. This can allow for multiple types of devices to be updated in a single update request.

2.2 Installing the image

The installation of the firmware image may be possible by a specific entity.

- By the BMC
- By the device that executes firmware image itself
- By software running standalone or within an operating system.

Figure 1 shows the flows of a firmware update. Understanding these flows is necessary to understand the failure and intervention modes, and the messages that are available.

On the left side of the diagram are the three installation entities described above.

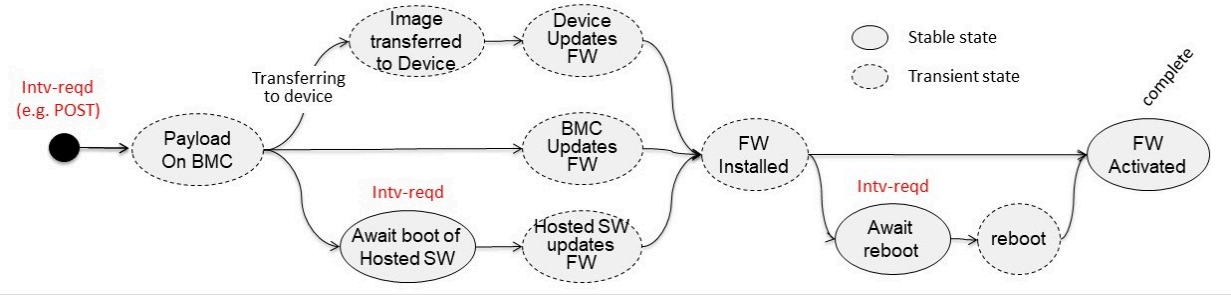

Figure 1

The installation via software may require that the system be booted in order to load the standalone software or the operating system. This is shown as an user intervention state on the lower flow.

Another user intervention state may exist between the firmware installed and firmware activated states. This is shown on the right side of the flow diagram. In this flow, user intervention such as a reset is required between the install and activation of the firmware image.

Now, looking the flow diagram from left of right, the general firmware update flow can be divided in several stages.

- Transfer the firmware image to the system
- Verification of the firmware image on the system
- Transfer of the firmware image to the device
- Verification of the firmware image on the device
- Installation of the firmware image on the device
- Activation of the firmware image on the device

3 Data model and operations

The following section describes the data model of the update service in Redfish, as well as the operations a client can invoke based on the properties found within the data model.

3.1 Update service

The `UpdateService` resource is the top level resource visible on the service root. It contains status and control indicator properties such as `Status` and `ServiceEnabled`. These are common properties found on various Redfish service instances.

The `FirmwareInventory` and `SoftwareInventory` properties contain a link to the collection of `SoftwareInventory` resources. The collection referenced by `FirmwareInventory` contains software components generally referred to as platform firmware and does not execute within a host operating system. The collection referenced by `SoftwareInventory` contains software components executed in the context of a host operating system, such as device drivers, applications, or offload workloads. The `SoftwareInventory` resource is described in the Software inventory section.

The `SimpleUpdate` action found within `Actions` contains information for how to perform an update request using a "pull" method. This is described further in the Simple update section.

The `MultipartHttpPushUri` property is used for performing a multipart HTTP POST operation for invoking an update operation. This is described further in the Multipart HTTP push update section.

The `HttpPushUri`, `HttpPushUriTargets`, `HttpPushUriTargetsBusy`, `HttpPushUriOptions`, and `HttpPushUriOptionsBusy` properties are used for performing a vendor-defined HTTP POST operation for invoking an update operation. These properties are described further in the Unstructured HTTP push update section. **Note:** Due to the vendor-specific details of this operation, this method has been deprecated in favor of multipart HTTP push updates.

Example `UpdateService` resource:


```
{
    "@odata.id": "/redfish/v1/UpdateService",
    "@odata.type": "#UpdateService.v1_8_0.UpdateService",
    "Id": "UpdateService",
    "Name": "Update service",
    "Status": {
        "State": "Enabled",
        "Health": "OK",
        "HealthRollup": "OK"
```

```
    },
    "ServiceEnabled": true,
    "MultipartHttpPushUri": "/redfish/v1/UpdateService/update-multipart",
    "Actions": {
        "#UpdateService.SimpleUpdate": {
            "target": "/redfish/v1/UpdateService/Actions/SimpleUpdate",
            "@Redfish.ActionInfo": "/redfish/v1/UpdateService/SimpleUpdateActionInfo"
        }
    },
    "FirmwareInventory": {
        "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory"
    },
    "SoftwareInventory": {
        "@odata.id": "/redfish/v1/UpdateService/SoftwareInventory"
    },
    "HttpPushUri": "/redfish/v1/UpdateService/update",
    "HttpPushUriTargets": [],
    "HttpPushUriTargetsBusy": false,
    "HttpPushUriOptions": {
        "HttpPushUriApplyTime": {
            "ApplyTime": "Immediate",
            "ApplyTime@Redfish.AllowableValues": [
                "Immediate",
                "OnReset",
                "AtMaintenanceWindowStart",
                "InMaintenanceWindowOnReset"
            ],
            "MaintenanceWindowStartTime": "2018-12-01T03:00:00+06:00",
            "MaintenanceWindowDurationInSeconds": 600
        }
    },
    "HttpPushUriOptionsBusy": false
}
```


3.1.1 Simple update

The `SimpleUpdate` action within the `UpdateService` resource is used to request the service to download an image from a remote server and perform an update using the image. This is also known as a "pull" update. If the action is not present, the service does not support simple updates. A client can invoke this type of update by performing an HTTP POST to the URI specified by the `target` property.


```
{
    "Actions": {
        "#UpdateService.SimpleUpdate": {
            "target": "/redfish/v1/UpdateService/Actions/SimpleUpdate",
            "@Redfish.ActionInfo": "/redfish/v1/UpdateService/SimpleUpdateActionInfo"
```

```
        }
    },
    ...
}
```

The contents of HTTP body in the POST operation contains parameters supplied by the client. The following parameters can be provided by the client.

Parameter	Type	Description
`ImageURI`	String	The URI of the image to download and install.
`TransferProtocol`	String	The network protocol that the update service uses to retrieve the image file located at the URI provided in the `ImageURI` parameter. **Note:** This parameter is only used when the scheme is not encoded in the `ImageURI` parameter. If the scheme is present, this parameter is ignored.
`Targets`	Array	An array of strings that are URIs to resources that indicate where to apply the image.
`Username`	String	The user name to access the URI specified by the `ImageURI` parameter.
`Password`	String	The password to access the URI specified by the `ImageURI` parameter.

It is recommended that services support the `@Redfish.ActionInfo` annotation so that clients can discover the supported parameters for the operation.

Example `ActionInfo` resource where `ImageURI` is required, and all other parameters are optional:

```
{
    "@odata.id": "/redfish/v1/UpdateService/SimpleUpdateActionInfo",
    "@odata.type": "#ActionInfo.v1_1_0.ActionInfo",
    "Id": "SimpleUpdateActionInfo",
    "Name": "Simple Update Action Info",
    "Parameters": [
        {
            "Name": "ImageURI",
```

```
            "Required": true,
            "DataType": "String"
        },
        {
            "Name": "TransferProtocol",
            "Required": false,
            "DataType": "String",
            "AllowableValues": [
                "HTTP",
                "HTTPS",
                "FTP"
            ]
        },
        {
            "Name": "Targets",
            "Required": false,
            "DataType": "StringArray"
        },
        {
            "Name": "Username",
            "Required": false,
            "DataType": "String"
        },
        {
            "Name": "Password",
            "Required": false,
            "DataType": "String"
        }
    ]
}
```

Example simple update request:

```
POST /redfish/v1/UpdateService/Actions/UpdateService.SimpleUpdate HTTP/1.1
Content-Type: application/json
Content-Length: <computed-length>

{
    "ImageURI": "https://192.168.1.250/images/bmc_update.bin"
}
```

3.1.2 Multipart HTTP push update

The `MultipartHttpPushUri` property within the `UpdateService` resource contains the URI for performing a multipart HTTP push update. If the property is not present, the service does not support multipart HTTP push updates. Clients can perform an RFC7578-defined HTTP POST operation on this URI to perform an update. The body of the request

contains multipart forms with parameters for the update and the image for updating. The following multipart forms are defined for this operation.

Form	Required	`Content-Disposition`	`Content-Type`	Description
Update parameters	Yes	`form-data; name="UpdateParameters"`	`application/json`	JSON-formatted part for passing the update parameters. The format of the JSON follows the definition of the `UpdateParameters` object in the `UpdateService` schema.
Update image	Yes	`form-data; name="UpdateFile"; filename=string`	`application/octet-stream`	Binary file to use for the update. The value of the `filename` field reflects the name of the file as loaded by the client.
OEM	No	`form-data; name="OemXXXX"`	OEM-defined	OEM-defined form. The value of the `name` field will contain `Oem` concatenated with the company identifier.

The "Update parameters" form can contain the following properties. None of the parameters are required, so an empty JSON object is used for cases where the client does not need to specify anything.

Name	Type	Description
`Targets`	Array	An array of strings that are URIs to resources that indicate where to apply the image.
`@Redfish.OperationApplyTime`	String	The Redfish Specification-defined "Operation apply time" parameter.
`Oem`	Object	OEM-defined parameters.

Example multipart HTTP push update request:

```
POST /redfish/v1/UpdateService/update-multipart HTTP/1.1
Content-Type: multipart/form-data; boundary=--------------------------d74496d66958873e
Content-Length: <computed-length>

----------------------------d74496d66958873e
```

```
Content-Disposition: form-data; name="UpdateParameters"
Content-Type: application/json

{}

----------------------------d74496d66958873e
Content-Disposition: form-data; name="UpdateFile"; filename="bmc_update.bin"
Content-Type: application/octet-stream

<software image binary>
```

3.1.3 Unstructured HTTP push update (deprecated)

> **Note:** Due to the vendor-specific details of this operation, this method has been deprecated in favor of multipart HTTP push updates.

The `HttpPushUri` property within the `UpdateService` resource contains the URI for performing an unstructured HTTP push update. If the property is not present, the service does not support unstructured HTTP push updates. Clients can perform an HTTP POST operation on this URI to perform an update. The contents of the HTTP POST operation are vendor-defined.

The following properties in the update service are used to control the behavior of the unstructured HTTP push update.

Property	Type	Description
`HttpPushUriTargets`	Array	An array of strings that are URIs to resources that indicate where to apply the image when a client performs an HTTP POST on the URI specified by `HttpPushUri`.
`HttpPushUriTargetsBusy`	Boolean	An indication of whether any client has reserved the `HttpPushUriTargets` property. Clients are expected to set this property to `true` prior to configuring an unstructured HTTP push update, and `false` after the unstructured HTTP push update has been performed. **Note:** Services do not enforce this reservation. Clients are expected to honor the reservation made by other clients.

Property	Type	Description
`HttpPushUriOptions`	Object	Contains additional configuration parameters for the update, such as options for controlling the apply time of the update.
`HttpPushUriOptionsBusy`	Boolean	An indication of whether any client has reserved the `HttpPushUriOptions` property. Clients are expected to set this property to `true` prior to configuring an unstructured HTTP push update, and `false` after the unstructured HTTP push update has been performed. **Note:** Services do not enforce this reservation. Clients are expected to honor the reservation made by other clients.

Example properties for unstructured HTTP push updates:

```
{
    "HttpPushUri": "/redfish/v1/UpdateService/update",
    "HttpPushUriTargets": [],
    "HttpPushUriTargetsBusy": false,
    "HttpPushUriOptions": {
        "HttpPushUriApplyTime": {
            "ApplyTime": "Immediate",
            "ApplyTime@Redfish.AllowableValues": [
                "Immediate",
                "OnReset",
                "AtMaintenanceWindowStart",
                "InMaintenanceWindowOnReset"
            ],
            "MaintenanceWindowStartTime": "2018-12-01T03:00:00+06:00",
            "MaintenanceWindowDurationInSeconds": 600
        }
    },
    "HttpPushUriOptionsBusy": false,
    ...
}
```

3.2 Software inventory

The `SoftwareInventory` resource contains a representation of either a firmware image or software image that has been activated on the system. The following properties can be found in the resource.

Property	Type	Description
`Updateable`	Boolean	Indicates whether the image can be updated by the update service.
`Manufacturer`	String	The name of the manufacturer or producer of the image. This property follows the Redfish-defined date-time format.
`ReleaseDate`	String	The date of release or production for the image.
`Version`	String	The version of the image.
`LowestSupportedVersion`	String	The lowest supported version of the image. Manufacturers recommend clients do not downgrade below the lowest supported version.
`SoftwareId`	String	The implementation-specific identifier for the image.
`RelatedItem`	Array	An array of Redfish resources associated with the image.
`UefiDevicePaths`	Array	An array of UEFI device paths for the components associated with the image.

Example `SoftwareInventory` resource representing a BMC image:

```
{
    "@odata.id": "/redfish/v1/UpdateService/FirmwareInventory/BMC",
    "@odata.type": "#SoftwareInventory.v1_2_3.SoftwareInventory",
    "Id": "BMC",
    "Name": "Contoso BMC Firmware",
    "Status": {
        "State": "Enabled",
        "Health": "OK"
    },
    "Updateable": true,
    "Manufacturer": "Contoso",
    "ReleaseDate": "2017-08-22T12:00:00",
    "Version": "1.45.455b66-rev4",
    "SoftwareId": "1624A9DF-5E13-47FC-874A-DF3AFF143089",
    "LowestSupportedVersion": "1.30.367a12-rev1",
    "UefiDevicePaths": [
        "BMC(0x1,0x0ABCDEF)"
    ],
    "RelatedItem": [
        {
```

```
            "@odata.id": "/redfish/v1/Managers/BMC"
        }
    ]
}
```

4 Firmware update workflow

The following is the recommended workflow for performing firmware updates that services should support.

4.1 Initiate an update operation

Firmware updates can be performed using one of two methods:

- If images are kept on a remote server and not stored locally by the client, the `SimpleUpdate` action is the recommended method. This is described further in the Simple update section.
- If images are local to the client, a POST operation to the URI specified by the `MultipartHttpPushUri` property is the recommended method. This is described further in the Multipart HTTP push update section.

There might be cases where a service only supports one method. Clients might need to support fallback logic, such as creating a temporary web server to host a local image if multipart HTTP push updates are not supported.

If performing a simple update, additional information might need to be provided in order for the service to retrieve the image, such as a user name or password. This information is provided as parameters to the `SimpleUpdate` action.

If service supports the `@Redfish.OperationApplyTime` parameter, a client can specify when the service begins the update operation. This parameter is provided in the body of the `SimpleUpdate` action, or in the "Update parameters" of the multipart HTTP request. The following values can be used for the `@Redfish.OperationApplyTime` parameter.

Value	Description
`Immediate`	The update operation starts immediately.
`OnReset`	The update operation starts on the next system or service reset.
`AtMaintenanceWindowStart`	The update operation starts at the start of the maintenance window. The `MaintenanceWindow` property contains the start time and duration of the maintenance window.
`InMaintenanceWindowOnReset`	The update operation starts on the next system or service reset when inside of the maintenance window. The `MaintenanceWindow` property contains the start time and duration of the maintenance window.
`OnStartUpdateRequest`	The update operation starts when the `StartUpdate` action is performed by a client.

If the response from the update operation contains a HTTP `202 Accepted` status code, the client will need to transition to monitoring the task. For all other cases, the client will need to parse the result of the update operation.

4.2 Task monitoring of the update operation

The response from the update operation should contain a HTTP `202 Accepted` status code, which indicates a task has been produced for the operation. The `Location` header in the response contains the URI of the task monitor. Clients perform monitoring of this URI as described in the "Asynchronous operations" clause in the Redfish Specification.

Example task response:

```
HTTP/1.1 202 Accepted
Location: /redfish/v1/TaskService/TaskMonitors/545

{
    "@odata.id": "/redfish/v1/TaskService/Tasks/545",
    "@odata.type": "#Task.v1_5_0.Task",
    "Id": "545",
    "Name": "Task 545",
    "TaskState": "Running",
    "StartTime": "2020-10-15T14:44+06:00",
    "TargetUri": "/redfish/v1/UpdateService/update-multipart",
    "TaskMonitor": "/redfish/v1/TaskService/TaskMonitors/545",
    "TaskStatus": "OK",
    "Messages": [
        {
            "@odata.type": "#Message.v1_1_1.Message",
            "MessageId": "Update.1.0.UpdateInProgress",
            "Message": "An update is in progress.",
            "Severity": "OK",
            "MessageSeverity": "OK",
            "Resolution": "None"
        },
        {
            "@odata.type": "#Message.v1_1_1.Message",
            "MessageId": "Update.1.0.TargetDetermined",
            "Message": "The target device '/redfish/v1/Managers/BMC' will be updated with ...",
            "Severity": "OK",
            "MessageSeverity": "OK",
            "MessageArgs": [
                "/redfish/v1/Managers/BMC",
                "Contoso BMC v2.50"
            ],
            "Resolution": "None"
        }
    ]
}
```

Once the task is complete, the client will need to parse the result of the update operation.

4.3 Update operation response management

The response body of the update operation will contain an error object with a set of messages, as described by the "Error responses" clause in the Redfish Specification. The messages in the response will need to be parsed to determine what updates failed, succeeded, or if other steps need to be taken. The Messages section contains common messages that can be found in the response.

The `OperationTransitionedToJob` message from the Update Message Registry requires special handling, since it indicates that a job has been created for carrying out the remainder of the update operation. In this case, the client will need to transition to monitoring the job. If the response does not contain this message, the update operation is complete and the messages in the response describe the outcome of the update operation.

Example update operation response:

```
HTTP/1.1 200 OK

{
    "error": {
        "code": "Update.1.0.OperationTransitionedToJob",
        "message": "The update operation has transitioned to the job at URI ...",
        "@Message.ExtendedInfo": [
            {
                "@odata.type": "#Message.v1_1_1.Message",
                "MessageId": "Update.1.0.OperationTransitionedToJob",
                "Message": "The update operation has transitioned to the job at URI ...",
                "Severity": "OK",
                "MessageSeverity": "OK",
                "MessageArgs": [
                    "/redfish/v1/JobService/Jobs/1"
                ],
                "Resolution": "None"
            }
        ]
    }
}
```

4.4 Job monitoring

Jobs allow for more complex interactions with clients, such as pausing until a client performs an operation that allows the job to continue. In many update scenarios, resets or other actions will be required at some point in the update flow. Tasks do not allow for these types of semantics. The Messages section contains common messages that can be found in the job.

Example job:

```
{
    "@odata.id": "/redfish/v1/JobService/Jobs/1",
    "@odata.type": "#Job.v1_0_5.Job",
    "Id": "1",
    "Name": "Job 1",
    "JobStatus": "OK",
    "JobState": "UserIntervention",
    "StartTime": "2020-10-15T14:44+06:00",
    "PercentComplete": 85,
    "CreatedBy": "admin",
    "Payload": {
        "TargetUri": "/redfish/v1/UpdateService/update-multipart"
    },
    "Steps": {
        "@odata.id": "/redfish/v1/JobService/Jobs/1/Steps"
    },
    "Messages": [
        {
            "@odata.type": "#Message.v1_1_1.Message",
            "MessageId": "Update.1.0.UpdateInProgress",
            "Message": "An update is in progress.",
            "Severity": "OK",
            "MessageSeverity": "OK",
            "Resolution": "None"
        },
        {
            "@odata.type": "#Message.v1_1_1.Message",
            "MessageId": "Update.1.0.TargetDetermined",
            "Message": "The target device 'Onboard NIC 1' will be updated with image ...",
            "Severity": "OK",
            "MessageSeverity": "OK",
            "MessageArgs": [
                "Onboard NIC 1",
                "Contoso NIC Bundle 6.55"
            ],
            "Resolution": "None"
        },
        {
            "@odata.type": "#Message.v1_1_1.Message",
            "MessageId": "Update.1.0.AwaitToActivate",
            "Message": "Awaiting for an action to proceed with activating image ...",
            "Severity": "OK",
            "MessageSeverity": "OK",
            "MessageArgs": [
                "Contoso NIC Bundle 6.55",
                "Onboard NIC 1"
            ],
            "Resolution": "Perform the requested action to advance the update operation."
        },
        {
```

```
            "@odata.type": "#Message.v1_1_1.Message",
            "MessageId": "Base.1.10.ResetRequired",
            "Message": "In order to complete the operation, a component reset is required ...",
            "Severity": "Warning",
            "MessageSeverity": "Warning",
            "MessageArgs": [
                "/redfish/v1/Systems/1/Actions/ComputerSystem.Reset",
                "GracefulRestart"
            ],
            "Resolution": "Perform the required reset action on the specified component."
        }
    ]
}
```


Clients should monitor the `JobState` property in the job resource to determine actions that need to be taken, or if the operation is complete. The following table gives recommendations for how to manage the different states.

`JobState` values	How to manage
`Completed`	The update operation has completed successfully or with warnings. The `Messages` array contains information about which devices were updated.
`Exception` or `Cancelled`	The update operation has ended with errors, or has been cancelled by an administrator. The `Messages` array contains information about which devices were updated, failed, or never attempted.
`UserIntervention`	The update operation requires some action in order to continue. The `Messages` array contains information about required actions to take for the update to continue. It's possible clients will require knowledge about the overall state of the system before taking actions. For example, if a system reset is required, clients should close processes on the system prior to issuing a reset.
Others	The update operation is still in progress. No actions required at this point.

Jobs allow for "steps" to be shown within a single job with the `Steps` property. For update operations, a step can be mapped to an update for an individual device. This allows for showing the progress of an update operation when multiple devices will be updated from a single update operation. The usage of the `StepOrder` property is not required for update operations. Services are expected to determine the appropriate order in which to update devices, and in many cases, perform parallel updates.

Messages important to the overall flow of the update operation in each of the steps should be reported in the overall job. This includes messages indicating a device has been identified for update, a device has updated successfully, a device has failed an update, or a specific action is required. Other types of messages should only be found in their

respective steps. The "Expose in root job" column in the tables found in the Messages section gives guidance for specific messages to report in the overall job.

Services might delay setting the `JobState` property of the overall job to `UserIntervention` if multiple steps are expected to reach the same state. This could be done to minimize the number of resets carried out by the client.

Once the job enters the `Completed`, `Exception`, or `Cancelled` state, the update operation is complete and the messages in the job describe the outcome of the update operation.

5 Messages

From Figure 1, the firmware update flow can be divided in several stages:

- Transfer the firmware image to the manager
- Verification of the firmware image on the manager
- Transfer of the firmware image to the device
- Verification of the firmware image on the device
- Installation of the firmware image on the device
- Activation of the firmware image on the device

During each stage, messages within the task, job, and error response provide information on the update operation's completion status or error conditions. The update operation's stage determines the potential messages that can reside in the `Messages` or `@Messages.ExtendedInfo` properties.

5.1 "Transfer the firmware image to the manager" messages

Registry	`MessageId`	Description	Expected client reaction	Expose in root job
Update	`UpdateInProgress`	Indicates that an update is in progress.	Continue to monitor the update progress.	N/A
Update	`OperationTransitionedToJob`	Indicates that the update operation transitioned to a job for managing the progress of the operation.	Monitor the job referenced by the message to track the progress of the update operation.	N/A
Base	`ResourceMissingAtURI`	Indicates that the operation expected an image or other resource at the provided URI but none was found.	Restart the operation with a new URI.	N/A
Base	`ResourceAtUriInUnknownFormat`	Indicates that the URI was valid but the resource or image at that URI was in a format not supported by the service.	Restart the operation with a new URI.	N/A

Registry	MessageId	Description	Expected client reaction	Expose in root job
Base	ResourceAtUriUnauthorized	Indicates that the attempt to access the resource, file, or image at the URI was unauthorized.	Restart the operation with a new URI.	N/A
Base	CouldNotEstablishConnection	Indicates that the attempt to access the resource, file, or image at the URI was unsuccessful because a session could not be established.	Restart the operation with a new URI.	N/A
Base	SourceDoesNotSupportProtocol	Indicates that while attempting to access, connect to, or transfer a resource, file, or image from another location that the other end of the connection did not support the protocol.	Restart the operation with a new URI.	N/A
Base	AccessDenied	Indicates that while attempting to access, connect to, or transfer to or from another resource, the service denied access.	Restart the operation with a new URI.	N/A

5.2 "Verification of the firmware image on the manager" messages

Registry	MessageId	Description	Expected client reaction	Expose in root job
Update	UpdateInProgress	Indicates that an update is in progress.	Continue to monitor the update progress.	No
Update	OperationTransitionedToJob	Indicates that the update operation transitioned to a job for managing the progress of the operation.	Monitor the job referenced by the message to track the progress of the update operation.	No
Update	TargetDetermined	Indicates that a target resource or device for a image has been determined for update.	Continue to monitor the update progress.	Yes

Registry	`MessageId`	Description	Expected client reaction	Expose in root job
Update	`AllTargetsDetermined`	Indicates that all target resources or devices for an update operation have been determined by the service.	Continue to monitor the update progress.	No
Update	`AwaitToUpdate`	Indicates that the resource or device is awaiting for an action to proceed with installing image.	Perform the requested operation found in other messages. If none exist at this time, check back later.	No
Base	`ResetRequired`	Indicates that a component reset is required for changes or operations to complete.	Perform the reset operation on the URI specified in the message.	Yes, but requires coordination with other steps and removal of duplicates.

5.3 "Transfer of the firmware image to the device" messages

Registry	`MessageId`	Description	Expected client reaction	Expose in root job
Update	`UpdateInProgress`	Indicates that an update is in progress.	Continue to monitor the update progress.	No
Update	`TargetDetermined`	Indicates that a target resource or device for a image has been determined for update.	Continue to monitor the update progress.	Yes
Update	`TransferringToComponent`	Indicates that the service is transferring an image to a component.	Continue to monitor the update progress.	No
Update	`TransferFailed`	Indicates that the service failed to transfer an image to a component.	Look at other messages to determine corrective actions.	Yes

5.4 "Verification of the firmware image on the device" messages

Registry	`MessageId`	Description	Expected client reaction	Expose in root job
Update	`UpdateInProgress`	Indicates that an update is in progress.	Continue to monitor the update progress.	No

Registry	MessageId	Description	Expected client reaction	Expose in root job
Update	TargetDetermined	Indicates that a target resource or device for a image has been determined for update.	Continue to monitor the update progress.	Yes
Update	VerifyingAtComponent	Indicates that a component is verifying an image.	Continue to monitor the update progress.	No
Update	VerificationFailed	Indicates that the component failed to verify an image.	Look at other messages to determine corrective actions.	Yes

5.5 "Installation of the firmware image on the device" messages

Registry	MessageId	Description	Expected client reaction	Expose in root job
Update	UpdateInProgress	Indicates that an update is in progress.	Continue to monitor the update progress.	No
Update	TargetDetermined	Indicates that a target resource or device for a image has been determined for update.	Continue to monitor the update progress.	Yes
Update	InstallingOnComponent	Indicates that a component is installing an image.	Continue to monitor the update progress.	No
Update	AwaitToActivate	Indicates that the resource or device is awaiting for an action to proceed with activating an image.	Perform the requested operation found in other messages. If none exist at this time, check back later.	No
Update	ApplyFailed	Indicates that the component failed to apply an image.	Look at other messages to determine corrective actions.	Yes
Update	UpdateSuccessful	Indicates that a resource or device was updated.	No further actions needed.	Yes
Base	ResetRequired	Indicates that a component reset is required for changes or operations to complete.	Perform the reset operation on the URI specified in the message.	Yes, but requires coordination with other steps and removal of duplicates.

5.6 "Activation of the firmware image on the device" messages

Registry	`MessageId`	Description	Expected client reaction	Expose in root job
Update	`UpdateInProgress`	Indicates that an update is in progress.	Continue to monitor the update progress.	No
Update	`TargetDetermined`	Indicates that a target resource or device for a image has been determined for update.	Continue to monitor the update progress.	Yes
Update	`ApplyingOnComponent`	Indicates that a component is applying an image.	Continue to monitor the update progress.	No
Update	`ActivateFailed`	Indicates that the component failed to activate the image.	Look at other messages to determine corrective actions.	Yes
Update	`UpdateSuccessful`	Indicates that a resource or device was updated.	No further actions needed.	Yes

6 ANNEX A (informative) Change log

Version	Date	Description
1.0.1	2022-08-04	Various typographic and grammar corrections.
		Removed redundant **Transferring the image to the system** section.
		Added statements to clarify that an image transferred to the service may contain multiple images.
1.0.0	2021-04-06	Initial release.

7 Bibliography

- IETF RFC7578, L. Masinter et al., Returning Values from Forms: multipart/form-data, https://www.ietf.org/rfc/rfc7578.txt