

Managing Content Uploads Via API

Updated: October 2026







www.vitrium.com

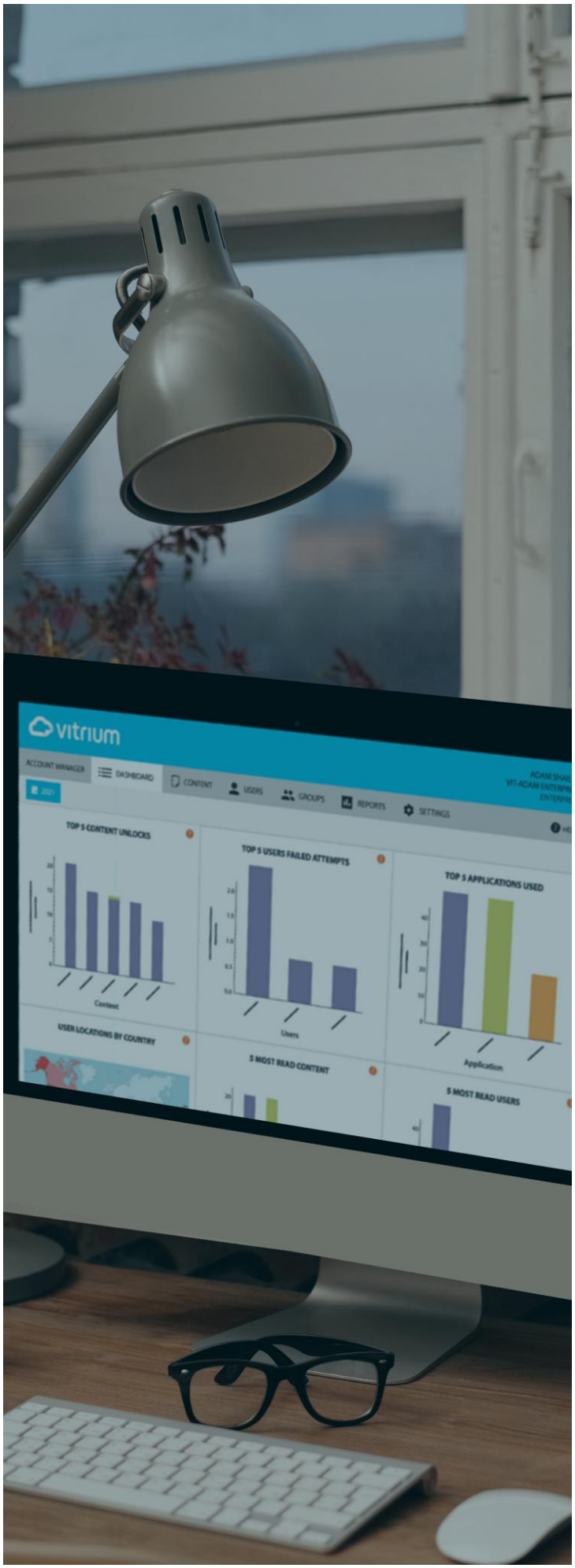


TABLE OF CONTENTS

Table of Contents ii
Managing Content via API 3
Types of Content Uploads 3
Content Processing Sub Tasks 3
Checking Task Processing Status 5
Web Hooks for Notifications 10
Large File Uploads Using /File Endpoint ..14
Shift Annotations Up/Down By n Pages ...20

MANAGING CONTENT VIA API

Types of Content Uploads

Content uploads on the USA MS Azure instance are either new or replace content transactions:

NEW	**POST** https://api.protectedpdf.com/api/2.0/doc
REPLACE	**PUT** https://api.protectedpdf.com/api/2.0/doc/<DocIdGUID>

NOTE: Vitrium also supports chunked file uploads with a FILE endpoint. A chunked file upload is a technique for transferring large files by breaking them into smaller, manageable pieces (chunks) and uploading them individually to a server. Once all pieces are received, the server reconstructs them into the original, complete file. This method is primarily used to bypass file size limits, increase speed, and enhance reliability when uploading large data over unstable networks. Chunk file upload is recommended for files that are more than 100 MB in size.

Refer to the section below entitled **Chunked Upload Using /File Endpoint** for more details.

Content Processing Sub Tasks

When uploading content using either **POST** or **PUT**, make sure to use the parameter **"AsyncAPI": true** which allows your integration to upload and not have to wait for a "processing completed" response. Instead, you will get a JSON response immediately that has numerous attributes such as Vitrium's internal doc Id, Version Id, DocCode, Web Viewer URL and much more. This is especially useful if you have many files to upload. By default, the Doc API uses **"AsyncAPI": true** and content upload API's will not respond until all sub-processing tasks are completed.

List of Processing Sub Tasks

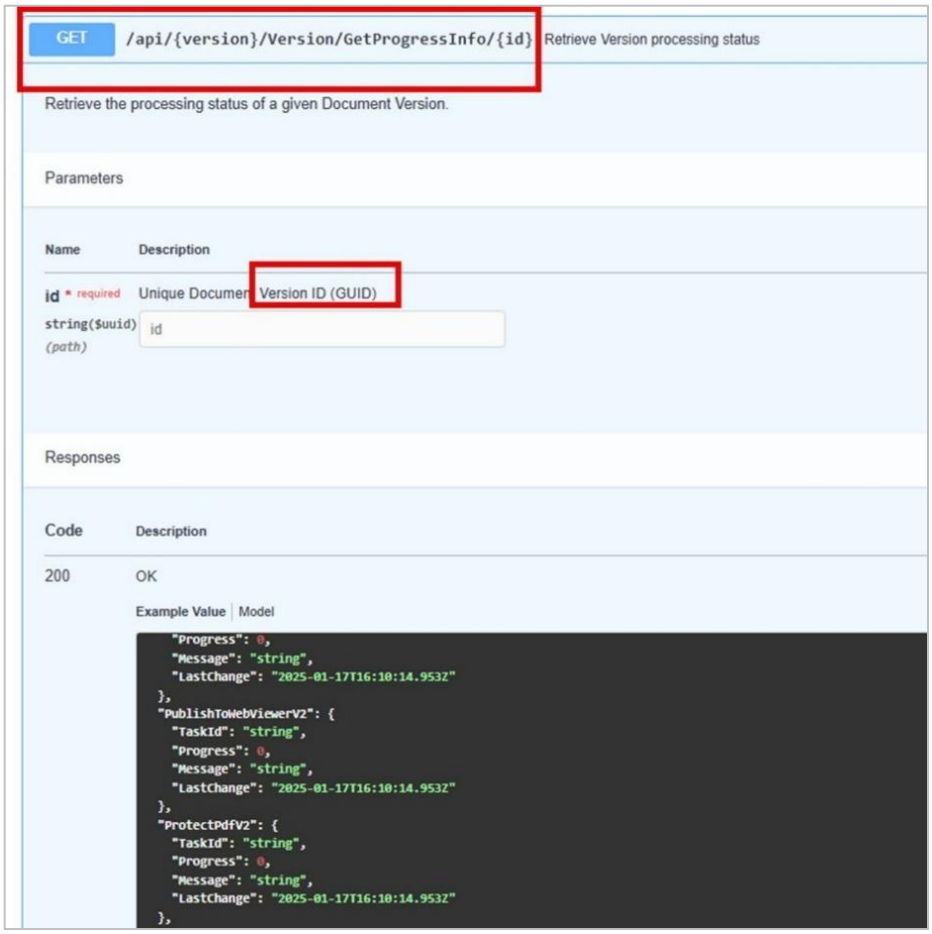
SubTask Name	Description
ConvertToPdf	Convert MS Office/OpenOffice file to PDF
ProtectPdf	Protected PDF (Legacy)
PublishToWebViewerOBSOLETE	Publish to Web Viewer (Legacy)
GenerateThumbnail	Generate a User Portal/Web Viewer thumbnail
ConvertToWeb	Convert video to Web
ConvertImageToPdf	Convert image to PDF
FullTextIndex	Full text index
GenerateVideoThumbnail	Generate video thumbnails for display in the User Portal
DecodeCapturedContent	Decode captured content – Content Tab
ConvertAudioToWeb	Convert audio to Web
OptimizeImage	Optimize Image
PublishToWebViewerV2	Publish to Web Viewer V2
ProtectPdfV2	Protected PDF V2
UniqueZipFolder	Unique zip folder (*available for custom installation only)*
MakeUniqueProtectedPdf	Make Unique Protected PDF (*available for custom installation only)*

Checking Task Processing Status

To check on the processing status of your uploaded file(s), use

GET ../Version/GetProcessingInfo API.

This API will provide a response with its sub tasks statuses.



There are several tasks generated for each content that's uploaded, and each task will have its own status. **"AsyncAPI": true** means, take my file and give me a response immediately but process the file when you can. The onus is on the developer/app to check the processing state of that file later.

If you're not using it, have a look at **GetProgressInfo** API to check the status of an upload. This API uses the Version ID as the parameter and will return the status of the content processing tasks.

JSON Response

When you upload a document, the JSON response returns a lot of metadata, including the VersionId (Id).

The version ID is near the end of the response. The following is a sample payload from a GetMostRecentVersion API call to highlight the difference between Document ID and (Version) ID.

```
{
"Results": [
  {
          "DocumentId": "e14d75f7-d399-44e6-b015-6f7258544034",       DOC ID
          "Document": {
                  "InternalId": 2373987,
                  "Version": "5.3",
                  "VersionCount": 1,
                  "MostRecentActiveVersionId": "e058f921-60a1-4826-83b9-e57fbab862c5",
                  "EnableVersionNotification": true,
                  "OptionFloatingLoginForm": false,
                  "DocBaseUrl": null,
                  "OptionDrmFlags": null,
                  "AccountId": "bc77e8f0-eab0-4f02-bcbf-aca76d46085e",
                  "EipUrl": null,
                  "FolderId": "fc5130be-4a39-4426-b1a7-94b6fa266c68",
                  "FolderIdPath": "FC5130BE-4A39-4426-B1A7-94B6FA266C68",
                  "Title": "0000-140C-20697A-002BFB4D-raw",
                  "Description": "<p><span style=\"font-family: Arial, sans-serif; font-size:
14pt;\"><br></span></p>",
                  "Authors": null,
                  "FileName": "0000-140C-20697A-002BFB4D-raw",
                  "YearCreated": 2024,
                  "PolicyId": "c5d1d3c4-4baa-4c0f-bfbc-890337d49c4f",
                  "CustomField": null,
                  "Policy": {
                    "Id": "c5d1d3c4-4baa-4c0f-bfbc-890337d49c4f",
                    "Name": "default",
                    "Type": "FullDrm",
                    "ContentType": "Document",
                    "PrintType": "HighResolution",
                    "AllowCopy": false,
                    "PackageKey": "",
                    "FormId": "627aa43f-fc36-4176-a19c-c9da850972e4",
                    "AllowBuildInLoginTemplate": false,
                    "CaseSensitivePassword": false,
                    "PreventScreenCapture": 0,
                    "AcroJsGosBehaviourType": "PromptAndCloseDocument",
                    "AcroJsGosUnlimitedBehaviourType": "PromptAndCloseDocument",
                    "DisableAnnotations": false,
                    "WatermarkIds": [
                            "4efddf51-8fb6-4096-81a8-e9b95877bdeb"                    ],
                    "Watermarks": [
                            {
                                    "Id": "4efddf51-8fb6-4096-81a8-e9b95877bdeb",
                                    "Name": "Bottom Multi Lines",
                                    "SetContentAt": "DocumentUnlockTime",
                                    "FontName": "Times-Roman",
                                    "FontSize": 7,
                                    "Color": "Blue",
                                    "Opacity": 0.81,
                                    "PositionX": 27,
                                    "PositionY": 5,
                                    "Width": 600,
                                    "Height": 36,
                                    "Rotation": 0,
                                    "Repeat": false,
                                    "TextualContent": "Licensed to/autorisé à John Doe, 00000565,
_userName_, by/vendu par Vitrium Systems under licence with DRM Council of Canada/_newLine_sous licence
avec le Conseil canadien des DRM. Contract expires/contract date d'expiration 2025-06-27._newLine_No
reproduction or networking permitted without licence from DRM Council of Canada._newLine_Aucune
```

```
reproduction ou réseautage n'est permis sans licence du Conseil canadien des DRM.",
                                    "SkipPages": null,
                                    "IsTrueWatermark": false,
                                    "CustomField": null,
                                    "CreatedOn": "2020-06-30T15:17:28.487Z",
                                    "Alignment": 0,
                                    "VideoVersionUsage": null,
                                    "DocumentVersionUsage": null                    }
                    ],
                    "OfflineDays": 0,
                    "AllowOfflineAccess": false,
                    "CustomField": null,
                    "CreatedOn": "2017-08-14T23:26:40.33Z",
                    "VersionUsage": null,
                    "ThumbnailScale": 0,
                    "DontOptimizeImages": true,
                    "Resolution": null,
                    "LosslessConversion": false,
                    "ShowInPortal": true,
                    "ImageSlicingEnabled": false,
                    "FlattenToImage": null,
                    "StartDate": null,
                    "ExpiryDate": null,
                    "RelativeExpiry": null,
                    "OfflineDuration": null,
                    "IpAddressesMax": null,
                    "IgnoredIpAddresses": null,
                    "LocationRestrictions": [],
                    "LocationPermits": [],
                    "RestrictedIpAddresses": null                },
                "WebViewerDocEncryptionEnabled": false,
                "AllowUserDefinedWebViewerEncryptionPassword": false,
                "SkipWebViewerProcessing": false,
                "SkipPpdfProcessing": false,
                "StoreOnly": false,
                "Tags": [],
                "Type": 1,
                "PublishDate": null,
                "DeletionDate": null,
                "Id": "e14d75f7-d399-44e6-b015-6f7258544034",
                "ExternalKey": null,
                "CreatedOn": "2024-12-21T13:45:37.767Z",
                "LastUpdatedOn": "2025-01-17T16:13:43.1014533Z",
                "Status": "Active"   },
          "Name": "Version: 1",
          "InternalId": 3173916,
          "DocCode": "0000-1C55-243963-00306E1C",
          "WebViewerType": "PDF",
          "WebViewerUrl": https://view.protectedpdf.com/ VYzFWa,
          "DownloadUrl": https://api.protectedpdf.com/api/2.0/version/file/e058f921-60a1-4826-83b9-
e57fbab862c5?downloadToken=yjoGpYp5t5mVMoFG13m5NGeuAd6ywMgwCN%2BhqPpW6IbdpirpRY488VHf2K4kYx7xNU2mKPyOU%2Fyc
e%2FjSJ%2FCvNL5KxYFcUXgPyk8ZfFi0CZcvR9vINLD1LvyjhBgNUqp3Q4DzVtwZ7suy49StB8PglJC1M%2FSvlSms,
          "ProtectionPassword": "G[mzgD:@)9jW>Y?H2n-A#V (mzgD:@)9jW%3eY?H2n-A#V) (mzgD:@)9jW%3eY?H2n-A#V
(mzgD:@)9jW%3eY?H2n-
A#V))]d8O9#LExNn8e^r2f;2)RA.Jm1$|u/y9S_/I}RfRz%LV)Z:V0AH}j;E{iy$H?Uuz0^qt*j@OykUn{@bN8rbu}7aZ*J@cpXSeTx6E)k
pQa:d?91A6Xzks:A8;pQ$78j9aR]TcFXeYCLU>ea1QIEE][@lT{$z38(s|UD()h91AUMq)*@zOzwy4e)nroAnc8#0U3c$at))koE6@V3mOO
lDLT{xm0behWYdtwVPo/Y6*%O",
          "DoNotEncryptContents": false,
          "EncryptOnlyEmbeddedFiles": false,
          "UniqueDocCopyId": null,
          "SkipPages": null,
          "OriginalFileName": null,
          "RawFileName": "0000-1C55-243963-00306E1C-raw.pdf",
```

```
            "ProtectedFileName": "0000-1C55-243963-00306E1C-protectedV2.pdf",
            "ProcessInfo": {
                    "ProtectPdfV2": {
                      "TaskId": "c372bd91-e3b9-4dbb-8fac-99911329bfde",
                      "Progress": 100,
                      "Message": null,
                      "LastChange": "2024-12-21T13:45:43.083"
                    },
                    "PublishToWebViewerV2": {
                      "TaskId": "29987b18-6bd0-419c-a1d8-9a789ad253a5",
                      "Progress": 100,
                      "Message": null,
                      "LastChange": "2024-12-21T13:45:39.4"
                    }
            },
            "AsyncAPI": true,
            "IsReplacement": false,
            "AnnotationShifts": null,
            "VideoAspectRatio": null,
            "WebViewerBlobId": null,
            "Id": "e058f921-60a1-4826-83b9-e57fbab862c5",    VERSION ID
            "ExternalKey": null,
            "CreatedOn": "2024-12-21T13:45:37.767Z",
            "LastUpdatedOn": null,
            "Status": "Active"
    }
],
"TotalRecords": 1,
"LastVersion": 0,
"ExtendedData": {
    "FolderMapping": {
            "fc5130be-4a39-4426-b1a7-94b6fa266c68": "Main Folder"
    }
}
}
```


CRITICAL TO KNOW: When you **UPLOAD NEW** content, there's only 1 version record in existence, but when you **REPLACE** that content, there is a 2nd temporary version record created.

This is important to note because when you try to get the status of a **NEW** upload, that version record id will always remain. BUT when you **REPLACE** content, the version record id is only available temporarily and only while the back end has time to perform the content processing. Once that processing completes "successfully", the original files in the content repository (on disk or in blob storage) will be overwritten with the replaced content and the temporary version record in the db will be deleted, leaving the original version and doccode in place.

IMPORTANT TO KNOW: The API GetProgressInfo will return an error if that temporary version record has been deleted (for a replacement content upload) and when that replacement content upload has been completed its tasks with success.

IMPORTANT TO KNOW: If you're keeping track of Vitrium DocCodes in your system, then you only want to save the new upload DocCode and not the DocCode that's provided for a content replacement. The last element in the DocCode is computed from the version record and when the temporary version record is deleted that DocCode no longer exists.

The **GET ../Version/GetProcessingInfo** will return the status of each type of processing and indicate if it failed or if it's still in the queue being processed. The 2 primary tasks that you're interested in are **ProtectedPdfV2** and **PublishToWebViewerV2**. Each task will reflect their own status in the Progress value:


```
"ProcessInfo": {
        "ProtectPdfV2": {
          "TaskId": "c372bd91-e3b9-4dbb-8fac-99911329bfde",
          "Progress": 100,
          "Message": null,
          "LastChange": "2024-12-21T13:45:43.083"                },
        "PublishToWebViewerV2": {
          "TaskId": "29987b18-6bd0-419c-a1d8-9a789ad253a5",
          "Progress": 100,
          "Message": null,
          "LastChange": "2024-12-21T13:45:39.4"                  }
```


Progress of 0 means it's pending to be processed (or is underway); 1-99 means the video processing is at that percent complete; 100 means success/complete; a negative value is an error, and the **Message** will contain the application stack dump which is the reason for the error. In most cases, the error for a PDF being uploaded might have passwords or the file is somehow corrupt.

Web Hooks for Notifications

Recent Vitrium Security builds provide the ability to setup “web hooks” (notifications), and these can send a set of query parameters to an external page with the results. Using this type of webhook can greatly help in monitoring upload statuses and can be integrated into a workflow. Alternatively, you can configure the system to send email when content processing completes, but in your case, that would be a lot of email, and a human may not want to manage those. The better email option would be to only send for failed processing but in the case of stalled processing (state=0), it won't trigger an event as it hasn't failed yet. You could even define webhook and an email notification for failures. If you’re interested in this, please let us know, and we’ll prepare insight.

> A useful site for testing webhooks is https://webhook.site/
>
> The site will generate a GUID based URL that can be added to your Vitrium Admin UI webhook configuration in Notification Settings page.

The following are sample payloads that our webhooks could generate:

Content Processing Succeeded


```
{
  "Context":{
    "FolderPath":[
      "329e56fd-e349-4d35-947b-3b71bafd9084",
      "f5c563a4-9290-4ec4-9350-d06bd7bdb3ea"
    ],
    "DocumentId":"404ffa31-5f28-4a88-9ea0-7720ef972036",
    "VersionId":"7cbf444e-1b6f-4e3d-91e3-fcaa16866d76",
    "DocCode":"0000-2CE7-2050B2-002BEEB4",
    "Metadata":{
      "ContentType":"std",
      "Title":"My PDF File Title",
      "VersionName":"Version: 1",
      "UserSpecificWatermarkTemplates":null
    },
    "ExternalKey":null,
    "Status":{
      "IsActive":true,
      "IsMostRecentVersion":true,
      "IsMostRecentVersionActive":true
    },
    "Alias":"wUMGuB",
    "WebViewerUrl":"https://view-beta-lb.vitrium.com/wUMGuB",
    "DeletionDate":null,
    "ProcessInfo":{
      "ConvertToPdf":{
        "TaskId":"b5a23417-debe-4ff5-8ff3-8c662796363a",
        "Progress":100,
        "Message":null,
        "LastChange":"2024-06-11T13:25:33.293Z"
```

```
      },
      "ProtectPdfV2":{
        "TaskId":"33dc49d0-62f2-40ca-ad5c-7ca78d975b80",
        "Progress":100,
        "Message":null,
        "LastChange":"2024-06-11T13:25:34.31Z"
      },
      "PublishToWebViewerV2":{
        "TaskId":"2490109a-45fb-4966-90df-c21c7b812717",
        "Progress":100,
        "Message":null,
        "LastChange":"2024-06-11T13:25:34.037Z"
      }
    },
    "ProcessInfoString":"{\"ConvertToPdf\":{\"TaskId\":\"b5a23417-debe-4ff5-8ff3-
8c662796363a\",\"Progress\":100,\"Message\":null,\"LastChange\":\"2024-06-
11T13:25:33.293Z\"},\"ProtectPdfV2\":{\"TaskId\":\"33dc49d0-62f2-40ca-ad5c-
7ca78d975b80\",\"Progress\":100,\"Message\":null,\"LastChange\":\"2024-06-
11T13:25:34.31Z\"},\"PublishToWebViewerV2\":{\"TaskId\":\"2490109a-45fb-4966-90df-
c21c7b812717\",\"Progress\":100,\"Message\":null,\"LastChange\":\"2024-06-11T13:25:34.037Z\"}}"
  },
  "PrevVersionContext":null,
  "EventId":"32ff6b9c-c930-48d2-b28f-08aa02ac2a73",
  "AccountId":"90c9996a-cbea-4783-8e70-4741b6bca7bf",
  "Time":"2024-06-11T13:25:34.309653Z",
  "Source":"Vitrium.Security.Service, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
}
```


Content Processing Failed


```
{
  "Context":{
    "FolderPath":[
      "329e56fd-e349-4d35-947b-3b71bafd9084",
      "f5c563a4-9290-4ec4-9350-d06bd7bdb3ea"
    ],
    "DocumentId":"7c87c0ad-e846-4f0b-bb30-5fbcbea205e5",
    "VersionId":"117210d9-7b13-437e-8085-52f4cfcd2adb",
    "DocCode":"0000-2CE7-2050B3-002BEEB5",
    "Metadata":{
      "ContentType":"std",
      "Title":"My Other PDF File Title",
      "VersionName":"Version: 1",
      "UserSpecificWatermarkTemplates":null
    },
    "ExternalKey":null,
    "Status":{
      "IsActive":true,
      "IsMostRecentVersion":true,
      "IsMostRecentVersionActive":true
    },
    "Alias":"YPnPZF",
    "WebViewerUrl":"https://view-beta-lb.vitrium.com/YPnPZF",
    "DeletionDate":null,
    "ProcessInfo":{
      "ProtectPdfV2":{
        "TaskId":"1b1e55db-bde3-427f-89a0-dcbd992304b9",
        "Progress":-6,
        "Message":"InvalidDocumentFormatException:\r\n    at
Vitrium.ppdf.Injector.InjectorV2.Inject(Document doc, Stream raw, String pathTemplate, IDictionary`2
```

```
keyValuePairs, String obscuringImagePath, String uniqueLoginObscureImagePath, Stream protectedOutput,
String clientMessagePath, String apiSecretKey, Double saveOfflineEventsMinimumReaderVersion) in
D:\\TC1\\work\\4e9523c364f07ac2\\Vitrium.ppdf.Injector\\InjectorV2.cs:line 66\r\n   at
Vitrium.ppdf.Injector.PdfProtector.ProtectFullDrmV2(Document doc, String rawFilePath, String
outputFilePath, String formFilePath, String obscuringFilePath, String uniqueLoginObscureImagePath, String
messagesFilePath, String apiSecretKey, Double saveOfflineEventMinReaderVersion, FileFormatType type,
Boolean officeFormat) in D:\\TC1\\work\\4e9523c364f07ac2\\Vitrium.ppdf.Injector\\PdfProtector.cs:line
75\r\n   at Vitrium.Security.Agent.ProtectPdf.Program.Main(String[] args) in
D:\\TC1\\work\\4e9523c364f07ac2\\Vitrium.Security.Agent.ProtectPdfV2\\Program.cs:line 92\r\n",
        "LastChange":"2024-06-11T13:27:20.347Z"
      },
      "PublishToWebViewerV2":{
        "TaskId":"1c9be443-155f-4fd7-ae82-e52927f5cf48",
        "Progress":100,
        "Message":null,
        "LastChange":"2024-06-11T13:27:46.9Z"
      }
    },
    "ProcessInfoString":"{\"ProtectPdfV2\":{\"TaskId\":\"1b1e55db-bde3-427f-89a0-
dcbd992304b9\",\"Progress\":-6,\"Message\":\"InvalidDocumentFormatException:\\r\\n   at
Vitrium.ppdf.Injector.InjectorV2.Inject(Document doc, Stream raw, String pathTemplate, IDictionary`2
keyValuePairs, String obscuringImagePath, String uniqueLoginObscureImagePath, Stream protectedOutput,
String clientMessagePath, String apiSecretKey, Double saveOfflineEventsMinimumReaderVersion) in
D:\\\\TC1\\\\work\\\\4e9523c364f07ac2\\\\Vitrium.ppdf.Injector\\\\InjectorV2.cs:line 66\\r\\n   at
Vitrium.ppdf.Injector.PdfProtector.ProtectFullDrmV2(Document doc, String rawFilePath, String
outputFilePath, String formFilePath, String obscuringFilePath, String uniqueLoginObscureImagePath, String
messagesFilePath, String apiSecretKey, Double saveOfflineEventMinReaderVersion, FileFormatType type,
Boolean officeFormat) in
D:\\\\TC1\\\\work\\\\4e9523c364f07ac2\\\\Vitrium.ppdf.Injector\\\\PdfProtector.cs:line 75\\r\\n   at
Vitrium.Security.Agent.ProtectPdf.Program.Main(String[] args) in
D:\\\\TC1\\\\work\\\\4e9523c364f07ac2\\\\Vitrium.Security.Agent.ProtectPdfV2\\\\Program.cs:line
92\\r\\n\",\"LastChange\":\"2024-06-11T13:27:20.347Z\"},\"PublishToWebViewerV2\":{\"TaskId\":\"1c9be443-
155f-4fd7-ae82-e52927f5cf48\",\"Progress\":100,\"Message\":null,\"LastChange\":\"2024-06-
11T13:27:46.9Z\"}}"
  },
  "PrevVersionContext":null,
  "EventId":"dc1bd551-e1a4-482b-a9fe-9ff486a07913",
  "AccountId":"90c9996a-cbea-4783-8e70-4741b6bca7bf",
  "Time":"2024-06-11T13:27:46.9100701Z",
  "Source":"Vitrium.Security.Service, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
}
```

You also have other notifications available including when a Staff User Logs in.

Staff User Login


```
{
  "IPAddress":"84.69.16.62",
  "User":{
    "Id":{
      "AccountId":"90c9996a-cbea-4783-8e70-4741b6bca7bf",
      "Id":"03926a66-4bfb-4323-be28-8f632187215b"
    },
    "Username":"JohnDoe",
    "IsActive":false,
    "Notes":null,
    "ExternalKey":null,
    "CustomField":null,
    "ForcePasswordReset":false,
    "EmailAddress":"johnDoe@SomeonesDomainName.com",
    "LastPasswordChangeDate":null,
    "UniqueToken":null,
    "IsEnabled":false
  },
  "EventId":"26b82892-73b1-4246-b407-58ba8fed7bb8",
  "AccountId":"90c9996a-cbea-4783-8e70-4741b6bca7bf",
  "Time":"2024-06-11T13:32:43.044221Z",
  "Source":null
}
```

Large File Uploads Using /File Endpoint

When uploading large files more than 100 MB, you'll need to send that file in ***chunks***. The process of uploading a file to Vitrium in chunks consists of an initial request which initializes the upload process and returns a tracking GUID, several subsequent requests sending pieces of the content until the entire file is uploaded, and a final request that initiates processing of the uploaded file.

IMPORTANT NOTE: Upload chunk size is currently coded to 4MB and is not configurable.

In the Admin UI, by default, Vitrium initiates the /File chunked upload request when the content length exceeds 100MB.

Initial POST Request

The initial POST request includes content metadata and initializes the upload process. Existing active session is needed and the request requires the usual authentication headers:

- **X-VITR-ACCOUNT-TOKEN**: Vitrium Account ID (GUID)
- **X-VITR-SESSION-TOKEN**: Active Vitrium Session ID (GUID)

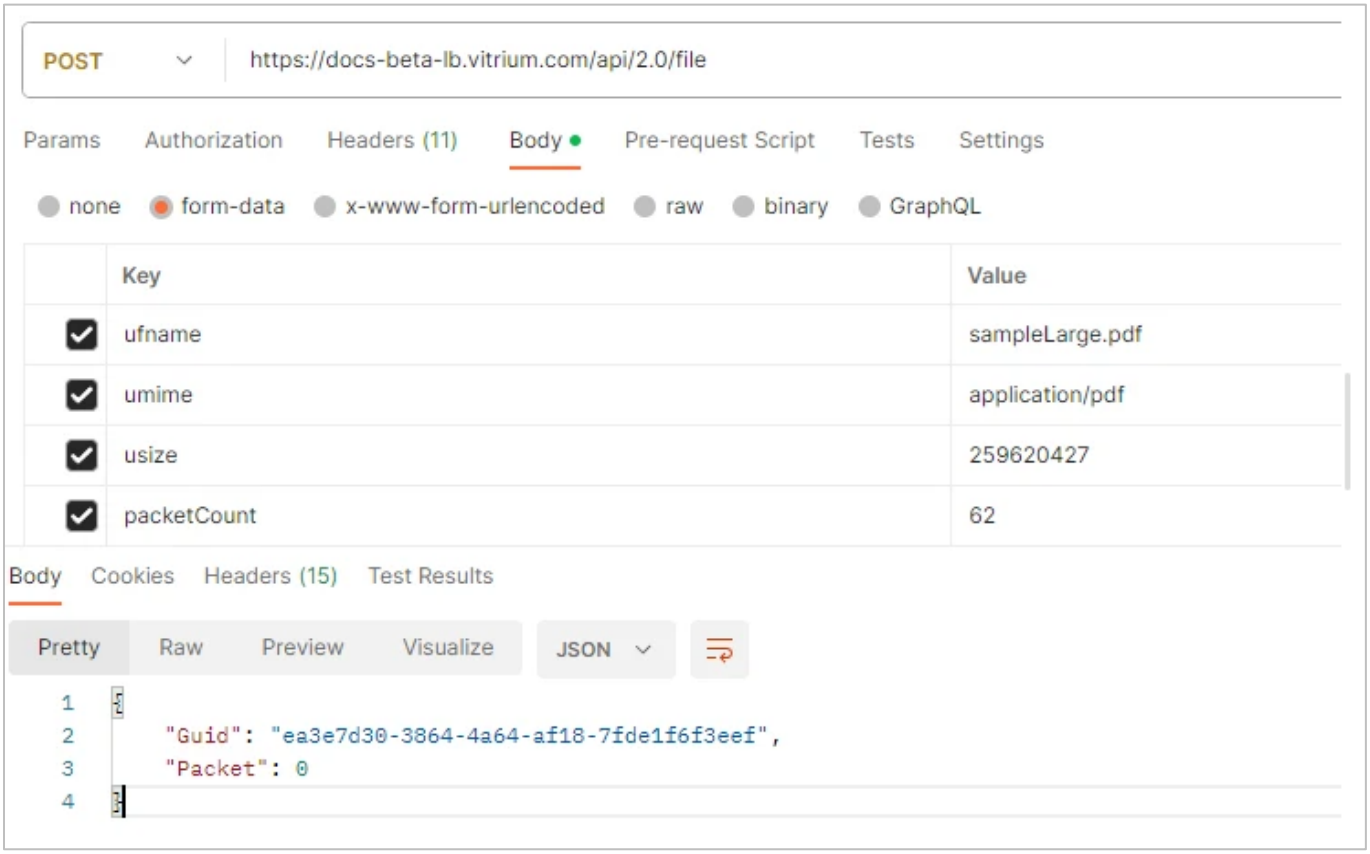
Content-Type of the request must be multipart/form-data, and the following fields are required in the request body:

Content Type	Description
ufname	Name of the file as it will appear in Vitrium after processing (eg. “test.pdf”). This does not necessarily need to match the source file name.
Umime	MIME type of the content (normally “application/pdf”
Usize	Size of the content in bytes (eg. 259620427)
packetCount	Content size in bytes divided by the chunk size of 4194304 bytes

The initial request returns a response in **application/json** format containing a tracking GUID and initial packet count of 0:

```
{
    "Guid": "ea3e7d30-3864-4a64-af18-7fde1f6f3eef",
    "Packet": 0
}
```

Postman Example



Raw Request Reference

```
POST /api/2.0/file HTTP/1.1
X-VITR-ACCOUNT-TOKEN: 39ea45e3-87ce-15a4-9d15-311ce64ef882
X-VITR-SESSION-TOKEN: 5a0f6354-babe-eacf-8765-7febc673a6d3
User-Agent: PostmanRuntime/7.32.3
Accept: */*
Cache-Control: no-cache
Postman-Token: e318977d-babe-eaba-dade-56d654f55ea9
Host: docs-beta-lb.vitrium.com
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

```
Content-Type: multipart/form-data; boundary=--------------------------
897166259437533347861911
Content-Length: 520

----------------------------897166259437533347861911
Content-Disposition: form-data; name="ufname"
sampleLarge.pdf
----------------------------897166259437533347861911
Content-Disposition: form-data; name="umime"
application/pdf
----------------------------897166259437533347861911
Content-Disposition: form-data; name="usize"
259620427
----------------------------897166259437533347861911
Content-Disposition: form-data; name="packetCount"
62
----------------------------897166259437533347861911--
```

Vitrium's Typescript Admin UI Implementation

```
 1. class ChunkedUploadRequest implements ICancelHandle
 2. {
 3.     private packetSize = 4 * 1024 * 1024;
 4.     private abortController: AbortController = null;
 5.     private currentPacket : number = 0;
 6.     private totalPackets: number;
 7.     private aborted : boolean =false;
 8.
 9.     constructor(private url: string, private file: File,
10.                 private onSuccess: Function,
11.                 private onProgress: (progress:number)=>void,
12.                 private onError: (response: TypeError)=>void)
13.     {
14.         this.totalPackets = Math.ceil(this.file.size / this.packetSize);
15.         this.startUpload();
16.     }
17.
18.     private prepareMetadata()
19.     {
20.         return {
21.             ufname : this.file.name,
22.             umime : this.file.type || "application/octet-stream",
23.             usize : this.file.size.toString(),
24.             packetCount: this.totalPackets.toString()
25.         };
26.     }
27.     private getFetchOptions(body: FormData| Blob) :RequestInit
28.     {
29.         this.abortController = new AbortController();
30.         return {
31.             signal: this.abortController.signal,
32.             method: 'POST', // *GET, POST, PUT, DELETE, etc.
33.             mode: 'cors', // no-cors, *cors, same-origin
34.             cache: 'no-cache',
35.             headers: Request.defaultRequestHeadersProvider.getHeaders(this.url),
36.             body: body
37.         };
```

```
38.     }
39.
40.     startUpload()
41.     {
42.         const metadata = this.prepareMetadata();
43.
44.         const formData = new FormData();
45.         formData.append('ufname', metadata.ufname);
46.         formData.append('umime', metadata.umime);
47.         formData.append('usize', metadata.usize);
48.         formData.append('packetCount', metadata.packetCount);
49.
50.         this.sendFetchRequest(this.url, this.getFetchOptions(formData));
51.     }
52.
53.     cancel() {
54.         if (this.abortController !== null)
55.             this.abortController.abort();
56.
57.         this.aborted = true;
58.     }
59.
60.     private sendFetchRequest(url: string, options: RequestInit)
61.     {
62.         if (this.aborted)
63.         {
64.             return;
65.         }
66.
67.         fetch(url, options).then(response=>{
68.             if (response.ok)
69.             {
70.                 return response.json();
71.             }
72.             else
73.             {
74.                 throw new Error(  "Invalid Response: "+response.status)
75.             }
76.         }).
77.         then(this.sendNextPacket.bind(this)).
78.         catch((reason :TypeError)=> {
79.             if(reason.name === "AbortError") {
80.                 console.log("Request aborted");
81.             }
82.             else
83.             {
84.                 this.onError(reason);
85.             }
86.         });
87.     }
88.
89.     private sendNextPacket(packetInfo: {Guid: string })
90.     {
91.         if (this.currentPacket < this.totalPackets)
92.         {
93.             this.onProgress( this.currentPacket/this.totalPackets);
94.
95.             const packetNum = ++this.currentPacket;
96.             const packet = this.getDataPacket(packetNum);
97.             this.sendFetchRequest(this.url+"?guid=" + packetInfo.Guid + "&packet=" +packetNum  +
"&total=" +this.totalPackets, this.getFetchOptions(packet));
98.         }
99.         else
```

```
100.         {
101.             this.onProgress(1);
102.             this.onSuccess(packetInfo.Guid);
103.         }
104.     }
105.
106.
107.     private getDataPacket(packetNumber): Blob {
108.         const startByte = (packetNumber - 1) * this.packetSize;
109.         const endByte = startByte + this.packetSize;
110.
111.         if ("slice" in this.file) {
112.             return this.file.slice(startByte, endByte);
113.         } else if ("mozSlice" in this.file) {
114.             return (<any>this.file).mozSlice(startByte, endByte);
115.         } else if ("webkitSlice" in this.file) {
116.             return (<any>this.file).webkitSlice(startByte, endByte);
117.         }
118.
119.         throw new Error("unsupported operation slice");
120.     }
121. }
122.
```

NOTE: The only time that packet size is less than **4194304** bytes is in the case of the last packet.

Algorithm Description

1. Send the initial request

   Send packets from 1 to N, one by one. There are mandatory parameters in the URL:

   a) G**uid** – id of the file obtained in the initial request (for example “a6a02a88-5c46-4845-ba92-3cd13563b021”)

   b) **Packet** – which part of the file this request contains (starting from 1)

   The response will look like this for the first packet sent:
   {"Guid":"a6a02a88-5c46-4845-ba92-3cd13563b021","Packet":1}

   The “Packet” field contains a packet which was received by server so send “Packet” + 1 next. If the value in “Packet” field is “-1”, there was an error then it needs to be sent again.

   If the “Packet” field value equals the total packet count, you successfully uploaded the file.

2. Now you can use the doc/version upload API to create the content, just instead of “rawfile” in form data add “filename” form data which contains the <guid>.<extension> string (for example “a6a02a88-5c46-4845-ba92-3cd13563b021.pdf”).

Final POST Request to the /api/2.0/doc API Example

```
----------------------------222307083911869350021886512941
Content-Disposition: form-data; name="filename"

a6a02a88-5c46-4845-ba92-3cd13563b021.pdf
----------------------------222307083911869350021886512941
Content-Disposition: form-data; name="settings"

{
  "EnableVersionNotification": true,
  "Type": 2,
  "PolicyId": "f8a41a4c-0c14-4df7-82dc-fde8d33450eb",
  "Title": "LargeFileSample",
  "SkipWebViewerProcessing": false,
  "SkipPages": null,
  "FileName": "SampleLarge",
  "VersionName": "Version: 1",
  "FolderId": "67e24e05-89b8-4fe9-bd5a-bce170d0b861",
  "AsyncAPI": true
}

----------------------------222307083911869350021886512941—
```

Shift Annotations Up/Down By n Pages

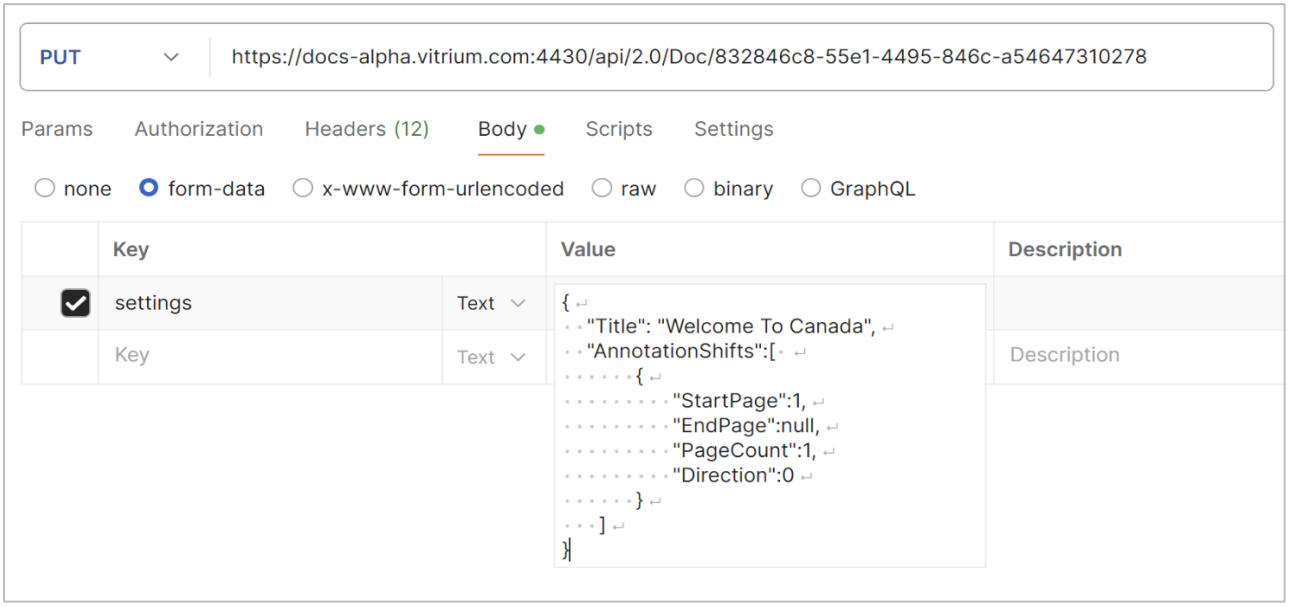
If you replace content and remove or add **entire pages** in the middle of the document, then the user annotations previously added to the last version of the content will be misaligned and Vitrium provides and API call that can be used to shift those user annotations to realign on the pages.

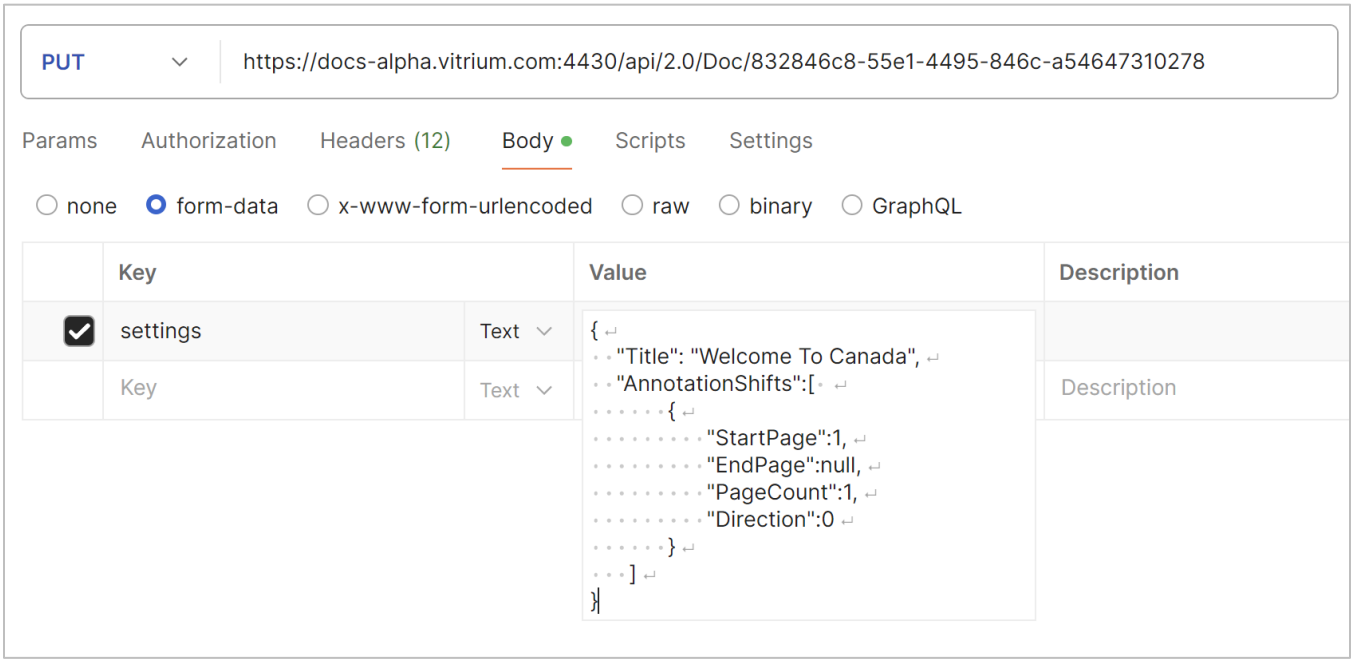
If you need to shift user annotations up or down when replacing a content title, use the API **PUT Doc Id** and include the following annotation shift parameters:

```
{
  "Title": "Welcome To Canada",
  "AnnotationShifts":[
      {
         "StartPage":1,
         "EndPage":null,
         "PageCount":1,
         "Direction":0
      }
   ]
}
```

StartPage means "starting on page n ..."
EndPage means "up to this page ... (or null until end of file or beginning of file)
PageCount means "number of pages to shuffle the annotations by"
Direction = 0 is down, Direction = 1 is up.

NOTE: Title is required in a Document PUT call, and the parameter must be supplied as a "settings" multi-part parameter. See image below.


PUT
https://docs-alpha.vitrium.com:4430/api/2.0/Doc/832846c8-55e1-4495-846c-a54647310278
Params
Authorization
Headers (12)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
Key
Value
Description
settings
Text
Key
Text
Description
{
"Title": "Welcome To Canada",
"AnnotationShifts":[
{
"StartPage":1,
"EndPage":null,
"PageCount":1,
"Direction":0
}
]
}


PUT
https://docs-alpha.vitrium.com:4430/api/2.0/Doc/832846c8-55e1-4495-846c-a54647310278
Params
Authorization
Headers (12)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
Key
Value
Description
settings
Text
Key
Text
Description
{
"Title": "Welcome To Canada",
"AnnotationShifts":[
{
"StartPage":1,
"EndPage":null,
"PageCount":1,
"Direction":0
}
]
}