Transitioning to VirtualViewer® v5.x

VirtualViewer® HTML5 Java versions 5.0 and later require several changes to your content handler and may require updates to your custom Javascript integrations. These changes are generally small-scale but necessary to keep your content handler up to date and to keep VirtualViewer maintainable and modern through future VirtualViewer releases (as releases do not typically modify public API).

There are five major steps to updating your content handler and Javascript for VirtualViewer 5.x. While your organization may wish to make updates based on new features and fixes in each release, this guide is meant to be a starting point to get VirtualViewer 5.x up and running.

In order to transition successfully to VirtualViewr 5.x, perform the following:

1. Update your VirtualViewer imports and dependencies to their new package locations so your content handler builds and uses the correct code.
2. Ensure that all methods intended to be called by VirtualViewer are implementations of an interface.
   Note: While the content handler may still compile, VirtualViewer will not function if

this step is not completed.

3. Update the logic of your `saveDocumentComponents` method so it does not delete a document component (a bookmarks file, for instance) when it receives a null value. **Note**: While the content handler may still compile, VirtualViewer will not function if this step is not completed.
4. Throw a `VirtualViewerAPIException` instead of passing an error parameter in a `ContentHandlerResult`.
5. Modify the parameters of selected VirtualViewer Javascript API calls, which now take single objects instead of long lists of arguments (see the details).

Update imports and dependencies

Update your VirtualViewer imports and dependencies to their new package locations so your content handler builds and uses the correct code.

What to Do

Use the tables below, or your IDE, to convert your content handler's `import` list to refer to the new package locations.

More Details

VirtualViewer's package structure has changed substantially to make public classes more clearly accessible and to organize the codebase more logically in versions 5.0 and later. Some import statements in your content handler code may no longer point to the right place.

For the most part, only the import statement will need to be updated. In addition to some new interfaces which have been added in v5.0 and later, all classes containing `FlexSnapSI` have been renamed to say `VirtualViewer` instead.

For example, `FlexSnapSIContentHandlerInterface` is now `VirtualViewerContentHandlerInterface`, and `FlexSnapSIAPIException` is now `VirtualViewerAPIException`, and so on.

`com.snowbound.contenthandler`

A new package, `com.snowbound.contenthandler`, now holds all content handler-related code: interfaces, classes used in the content handler, and VirtualViewer's example FileContentHandler.

Note that both `com.snowbound.snapserv.servlet.FlexSnapSISaverInterface` and `com.snowbound.snapserv.servlet.FlexSnapSIContentHandlerInterface` are now included in `com.snowbound.contenthandler.interfaces.VirtualViewerContentHandlerInterface`.

`com.snowbound.contenthandler` Package Changes
Old package: `com.snowbound.snapserv.servlet.FileContentHandler` **New package:** `com.snowbound.contenthandler.example.FileContentHandler`
Old package: `com.snowbound.snapserv.servlet.ContentHandlerInput` **New package:** `com.snowbound.contenthandler.ContentHandlerInput`

Old package: com.snowbound.snapserv.servlet.FlexSnapSIAPIException **New package:** com.snowbound.contenthandler.VirtualViewerAPIException
Old package: com.snowbound.snapserv.transport.pagedataVirtualViewerFormatHash **New package:** com.snowbound.contenthandler.VirualViewerFormatHash
Old package: com.snowbound.snapserv.servlet.AIIAnnotationslnterface **New package:** com.snowbound.contenthandler.interfaces.AIIAnnotationslnterface
Old package: com.snowbound.snapserv.servlet.CacheValidator **New package:** com.snowbound.contenthandler.interfaces.CacheValidator
Old package: com.snowbound.snapserv.servlet.CreateDocumentlnterface **New package:** com.snowbound.contenthandler.interfaces.CreateDocumentlnterface
Old package: com.snowbound.snapserv.servlet.DocumentNoteslnterface **New package:** com.snowbound.contenthandler.interfaces.DocumentNoteslnterface
Old package: com.snowbound.snapserv.servlet.VirtualViewerContentHandlerlnterface **New package:** com.snowbound.contenthandler.interfaces. VirtualViewerContentHandlerInterface
Old package: com.snowbound.snapserv.servlet.FlexSnapSISaverlntefrace **New package:** com.snowbound.contenthandler.interfaces. VirtualViewerContentHandlerInterface
Old package: com.snowbound.snapserv.servlet.FlexSnapSIContentHandlerlnterface **New package:** com.snowbound.contenthandler.interfaces. VirtualViewerContentHandlerInterface
Old package: com.snowbound.snapserv.servlet.Watermarkslnterface **New package:** com.snowbound.contenthandler.interfaces.WatermarksInterface

com.snowbound.common

com.snowbound.common Package Changes
Old package: com.snowbound.snapserv.transport.PSPermissionsRecord **New package:** com.snowbound.common.permissions.PSPermissionsRecord
Old package: com.snowbound.snapserv.transport.PermissionsFactory **New package:** com.snowbound.common.permissions.PermissionsFactory
Old package: com.snowbound.snapserv.transport.PermissionsRecord **New package:** com.snowbound.common.permissions.PermissionsRecord
Old package: com.snowbound.snapserv.servlet.Annotationlayer **New package:** com.snowbound.common.transport.Annotationlayer
Old package: com.snowbound.snapserv.transport.pagedata.AnnotationWrapper **New package:** com.snowbound.common.transport.AnnotationWrapper
Old package: com.snowbound.clientcontentserver.ExternalReference **New package:** com.snowbound.common.transport.ExternalReference

Old package: `com.snowbound.common.transport.Permissionlevels` **New package:** `com.snowbound.common.transport.Permissionlevel`
Old package: `com.snowbound.snapserv.transport.pagedata.PermissionsEntities` **New package:** `com.snowbound.common.transport.PermissionsEntities`
Old package: `com.snowbound.snapserv.transport.pagedata.FlexSnapSISnowAnn` **New package:** `com.snowbound.common.transport.VirtualViewerSnowAnn`
Old package: `com.snowbound.snapserv.transport.pagedata.RasterMaster` **New package:** `com.snowbound.common.util.RasterMaster`

Removals

Several private VirtualViewer classes have been moved out of the `com.snowbound.common` package for clarity, so it's easier to identify what's public. The classes in `com.snowbound.common` and `com.snowbound.contenthandler` are public and available for use. Using classes in other namespaces should not be necessary and is not supported; those classes may be obfuscated.

Update your content handler

Ensure that all methods intended to be called by VirtualViewer are implementations of an interface in your content handler.

What to Do

- For each of these functions, make sure to add the interface that describes it to the `implements` list of your content handler. **This is the most important step.**
- Review the functions that are implemented, and not just stubbed out, in your content handler
- Implement any new functions that the interface requires
- Delete unnecessary and unwanted interfaces and unnecessary stubbed-out functions

More Details

VirtualViewer 5.0 introduced a more modular interface system to make it easier to construct content handlers without unwanted and unnecessary functionality. Any methods you do not want or need can be omitted by not implementing the required interface, instead of having to make a stubbed-out implementation.

If you are converting a pre-5.0 content handler, there may now be a mismatch between the interfaces that your old content handler implements and the actual functions that your content handler implements. This may be a silent problem that your IDE does not pick up on, so it is imperative to assess your content handler and add any new interfaces that are needed.

For example, the function `getAvailableDocumentIds` used to be described in `FlexSnapSIContentHandlerlnterface`. If you have `FlexSnapSIContentHandlerlnterface` implemented, and simply change that to `VirtualViewerContentHandlerlnterface` without adding the new `AvailableDocumentslnterface`, your code may silently fail.

VirtualViewer will check if your content handler implements `AvailableDocumentslnterface` before attempting to call `getAvailableDocumentIds`. If it doesn't implement the interface, your `getAvailableDocumentIds` function would simply never get called.

While a few interfaces have been combined and removed into `VirtualViewerContentHandlerlnterface`, more have been separated into feature-specific interfaces.

All of the interfaces can be found in the `com.snowbound.contenthandler.interfaces` package and are described below:

Interface name	**Description (and changes)**
`AIIAnnotationslnterface`	Defines `getAIIAnnotationsForDocument` (unchanged)
`Annotationslnterface`	Defines annotation retrieval and modification methods (previously defined in `FlexSnapSIContentHandlerlnterface`)
`AvailableDocumentslnterface`	Defines getAvailableDocumentIds (previously defined in `FlexSnapSIContentHandlerlnterface`)
`Bookmarkslnterface`	Defines bookmark retrieval and modification methods (previously defined in `FlexSnapSIContentHandlerlnterface`)
`CacheValidator`	Defines `validateCache` (unchanged)
`CreateDocumentlnterface`	Defines `createDocument` method (unchanged)
`DocumentNoteslnterface`	Defines document notes retrieval and modification methods (new `deleteNotesContent` method added)
`EventSubscriberlnterface`	Defines `eventNotification` method (previously defined in `FlexSnapSIContentHandlerlnterface`)
`SendDocumentlnterface`	Defines `sendDocument` method (previously defined in `FlexSnapSIContentHandlerlnterface`)
`VirtualViewerContentHandlerInterface`	Defines minimum necessary content handler methods for document manipulation (renamed from `FlexSnapSIContentHandlerInterface`)
`Watermarkslnterface`	Defines watermark retrieval and modification methods (new `deleteWatermarkContent` method added)

Update `saveDocumentComponents`

Update the logic of your `saveDocumentComponents` method so that it does not delete a document component (a bookmarks file, for instance) when it receives a null value (see the details).

WARNING: While the content handler may still compile, VirtualViewer will not function if this step is not completed.

What to Do

Remove deletion logic from `saveDocumentComponents`. If `saveDocumentComponents` receives no content for some component, it should not delete that component: it should take no action. **This is the most important step.**

Implement `deleteBookmarkContent`, `deleteNotesContent` and `deleteWatermarkContent` functions if using an interface that requires them.

More Details

The content handler function `saveDocumentComponents` should no longer delete bookmarks, document notes or watermarks when it receives a null for those components. Instead, new `deleteBookmarkContent`, `deleteNotesContent` and `deleteWatermarkContent` methods should be implemented.

To solve an issue with our saving workflow, deleting bookmarks, document notes and watermarks now have equivalent content handler methods `deleteBookmarkContent`, `deleteNotesContent`, and `deleteWatermarkContent`, similar to the existing `deleteAnnotationContent`. If no content is provided to `saveDocumentComponents` for bookmarks, document notes or watermarks, no operation should be performed. VirtualViewer will call the equivalent delete method after `saveDocumentComponents` if necessary.

With the old logic, `saveDocumentComponents` would handle everything. If it received a `ContentHandlerInput` with a set of document notes but with bookmarks set to null, it assumed that bookmarks should be deleted. That is, if a component was null in the `ContentHandlerInput`, it should not exist on the database.

Now, that same `null` means "no action." If `saveDocumentComponents` receives a `ContentHandlerInput` where bookmarks is set to `null`, it should not delete bookmarks.

In addition to solving a saving issue, this will allow VirtualViewer to be more efficient when saving your documents. If no change is made to the above objects when saving a document, no unnecessary operations will be requested of the content handler.

Update exception handling

Throw a VirtualViewerAPIException instead of passing an error parameter in a ContentHandlerResult.

What to Do

Locate places in your content handler where you call `ContentHandlerResult.ERROR_MESSAGE`, and replace those calls with logging statements or by throwing `VirtualViewerAPIException`.

Locate references to `FlexSnapSIAPIException`, and replace those references with `VirtualViewerAPIException`.

More Details

`ContentHandlerResult.ERROR_MESSAGE` has been removed. For serious errors that should interrupt the request and inform the user, throw `VirtualViewerAPIException`.

Errors that should not interrupt the request can still be logged with VirtualViewer by using the SLF4J Logger interface provided by `SnowLoggerFactory`. (`VirtualViewerAPIExceptions` are logged automatically.)

Modify parameters in the Javascript API

Modify the parameters of selected VirtualViewer Javascript API calls, which now take single objects instead of long lists of arguments.

What to Do

Locate calls to `saveDocument`, `exportDocument`, `printDocument`, and `emailDocument` in your Javascript code.

Translate the passed parameters into values in a parameter object, with the keys given in the JSDoc documentation.

More Details

Several Javascript API calls in previous versions took more than ten parameters, which is very fragile. Especially with Javascript, it's easy to forget a single parameter and produce unexpected behavior that's hard to track down.

Many of these API calls now take an `options` object as a parameter, which will hold named versions of the old parameters. The substance of the parameters hasn't changed, but they should be passed as part of an object instead.

For example, to manually perform a **Save As** operation on a document without its annotations but with its watermarks burned in, the old call would require eleven parameters, with only four set to non-default, non-null values.

Now, the same call would be much simpler and more readable, passing in only the non-default parameters:

```
virtualViewer.saveDocument("myDocumentID.pdf",
  { "newDocumentId": "myNewDocumentID.pdf",
    "includeWatermarks": true,
    "saveAsFormat": "PDF"})
```

The API calls have been simplified as follows:

Old signature	New function signature
virtualViewer.saveDocument(documentId, newDocumentId, newDisplayName, burnRedactions, includeRedactionTags, includeTextAnnotations, includeNonTextAnnotations, copyAnnotations, includeDocumentNotes, includeWatermarks, saveAsFormat, pageRangeType, pageRangeValue, copyWatermarks, options)	virtualViewer.saveDocument(documentId, options)
virtualViewer.exportDocument(exportFormat, fileExtension, includeTextAnnotations, includeNonTextAnnotations, burnRedactions, includeRedactionTags, includeDocumentNotes, includeWatermarks, pageRangeType, pageRangeValue)	virtualViewer.exportDocument(options)
virtualViewer.printDocument(documentId, printToPDF, annotations, redactions, redactionTags, watermarks, docNotes, pageRangeType, pageRangeVal)	virtualViewer.printDocument(options)
virtualViewer.emailDocument(emailFormat, includeTextAnnotations, includeNonTextAnnotations, burnRedactions, includeRedactionTags, includeDocumentNotes, includeWatermarks, pageRangeType, pageRangeValue, fromAddress, toAddresses, ccAddresses, bccAddresses, subject, emailBody)	virtualViewer.emailDocument(options)