

Workshare Compare Server 9.5.2

Developer Guide

Table of Contents

Chapter 1: Introduction.....	3
Introducing Workshare Compare Server.....	4
Communicating with Workshare Compare Server	4
Chapter 2: Documentation and Examples	5
Overview	6
Types of Comparison	6
Choosing which type of comparison to run	6
Immediate Comparisons	9
GET Request Example.....	9
POST Request Example	12
Queued Asynchronous Comparisons	14
Making a queued comparison request.....	15
Handling the response to a queued comparison request.....	15
Data Storage on Workshare Compare Server.....	17
DVJS Comparisons	17
What is DVJS?.....	17
Getting started with DVJS	18
The DVJS APIs	19
Starting DVJS comparisons.....	19
Displaying a DVJS comparison	19
Customizing DVJS	20
Customizing which elements of the UI are Visible	22
Customizing the Language of the UI.....	22
Customizing the Toolbar.....	24
Appendix A. Change Summary Information	25
RedlineML	26
RedlineML Schema	26
Character set values	33
ChangeT values	34

Chapter 1: Introduction

This chapter describes the functionality provided by Workshare Compare Server and how to communicate with Compare Server.

Note: *In this document, the terms Workshare Compare Server and Workshare Compare service are interchangeable. Workshare Compare Server is the name of the product but where you find references to Workshare Compare service, it is in order to be technically accurate.*

Introducing Workshare Compare Server

Workshare Compare Server is a .NET application that consists of Workshare's tried and tested comparison engine, and a series of RESTful APIs. It is a fast, performant and robust document comparison machine that accepts two source documents (rtf, doc, docx, pdf, txt), compares them, and provides the output as either:

1. A document, in the following formats - rtf, doc, docx, pdf, txt, track change document
2. A URL, that loads the comparison result into Workshare's browser-based UI
3. An XML change summary, detailing only the changes that have occurred, and associated information about them

Compare Server's outputs are known as 'Redlines', whether shown as a document or within Workshare's UI.

Using Compare Server, you can integrate Workshare's comparison technology into your application to:

- Provide extremely fast and robust document comparison, including change identification and extraction.
- Verify and highlight all changes and differences between drafts and versions, no matter how complex the document.
- Validate that all changes closely adhere to policies and procedures, and an approved boilerplate.

Once installed, you can launch the bundled Swagger documentation into the browser, read about each of the endpoints, and try them out. Compare Server comes bundled with sample documents to facilitate this.

Communicating with Workshare Compare Server

You can communicate with Workshare Compare Server over HTTP REST calls. To fully benefit from this guide, you should have an understanding of RESTful APIs.

Chapter 2: Documentation and Examples

This chapter provides examples of GET and POST requests and information about running different types of comparisons.

Overview

The RESTful API endpoints are documented here: [http://\[your server name\]/swagger](http://[your server name]/swagger).

Types of Comparison

- **Classic vs DVJS comparisons**
 - Classic comparisons are performed by making calls to `api/Compare` and the result is a comparison in document format (`docx/doc/pdf`) or in data format (`xml` as `RedlineML`). The comparison result is returned to the calling code which can take appropriate actions to use it or make it available to an end user.
 - DVJS comparisons are performed by making calls to the `api/UICompare` endpoint and the result is a URL which can be viewed in a web browser to show the comparison document and change summary information to the user.
- **POST vs GET comparisons**
 - POST comparisons are performed by sending the content of the two documents to be compared to Compare Server as part of an HTTP POST request in `multipart/form-data` format.
 - GET comparisons are performed by sending URLs from which the documents to be compared can be retrieved to Compare Server as parameters on an HTTP GET request.
- **Immediate vs asynchronous queued comparisons**
 - Immediate comparisons are performed by making a single HTTP request (either GET or POST). The server performs the comparison in response to receiving the request and the result of the comparison is returned as the response to the HTTP request. (Note that certain administrative configuration options may cause the request to be delayed or refused if the server is too busy.)
 - Asynchronous queued comparisons are performed by making an HTTP request (either GET or POST) to the 'Enqueue' method. For example, `/api/Compare/Enqueue` or `/api/UICompare/Enqueue`. On receipt of the request the server will place the comparison in its queue and return a URL that the client code can use to check for the completion status of the comparison. Comparisons are performed in the order that they are queued. When a comparison is complete, the response to the status request is a URL that will redirect to the location from which the result can be downloaded.

Choosing which type of comparison to run

The following questions can help you choose the most suitable way to integrate Compare Server for your use case.

Question 1: Where is the location of the documents you need to compare?

Answers	Solution
Your documents are already located on a web server or other web-based system and you can construct a URL that will allow them to be downloaded without encountering additional security prompts or requiring user interaction.	In this case you can compare documents using the GET HTTP method. You will need to construct URLs that allow each of the documents to be compared to be downloaded, and pass those URLs as parameters to the comparison request. Compare Server will download the documents using the provided URLs and then perform the comparison.
Your documents are located on a web server or web-based system, but require additional information (cookies or login) to download.	In any of these cases you will need to use the POST HTTP method when performing a comparison. Your code will need to fetch the content of the documents and include the content of both in the comparison POST request, which should be made in multipart/form-data format.
Your documents are located on the local disk or storage of the device requesting the comparison.	
Your documents are located in some other location.	

Question 2: How do you intend to use the comparison result?

Answers	Solution
You want to show the comparison document to the user requesting it in their web browser.	In this case you should make a comparison request to the UICompare endpoint (DVJS comparison) and redirect the browser location to the URL specified in the 'Location:' header of the response. This will load the comparison into the browser and allow the user to view the document and summary of changes. See DVJS Comparisons for further details on customizing the user experience.
You want to store or process the comparison result as a document.	In these cases you should make a comparison request to the Compare endpoint (classic comparison) and choose an appropriate response format ('Pdf' or 'Docx'). Your code will be able to retrieve the comparison result.
You want to allow the user to download the comparison result as a document.	

Answers	Solution
You want to process data that summarizes the changes between the two documents that are being compared.	In this case you should make a comparison request to the Compare endpoint (classic comparison) and choose the RedlineML (comparison data as XML) response format and then process the resulting data as appropriate.

Question 3: How urgently do you need the result?

Answers	Solution
The comparison is being run as part of a background task and reliability is more important than turnaround time.	In this case you should use the asynchronous queued comparison option to make requests. Send requests to <code>/api/Compare/Enqueue</code> or <code>/api/UICompare/Enqueue</code> to begin a comparison and then poll the status link for completion. Note that your server must be configured by the server administrator to use an SQL database to store the queued jobs otherwise jobs may be lost if the server or IIS restarts (see <i>Compare Server Admin Guide</i>).
The comparison result is needed as soon as possible.	In this case you should use the immediate comparison options, sending requests to <code>/api/Compare</code> or <code>/api/UICompare</code> . However see also questions 4 and 5.

Question 4: Will you be comparing very large documents or large PDF documents?

Answers	Solution
Yes.	For these types of comparisons, you should consider using the asynchronous queued requests. This type of comparison may take some time to complete (from 30 seconds to many minutes) and using immediate comparison requests may lead to timeouts in web requests that are intermittent and difficult to debug. Using asynchronous queued requests avoids this problem.

Question 5: How do you expect the server to be utilized?

Answers	Solution
You expect the server to have to deal with peaks of load which exceed its capability to process comparisons. For example, you intend to perform batch processing and submit large numbers of comparisons in a short period of time.	In this case you should consider using asynchronous queued requests to ensure that peaks of load are handled well with no risk of dropped requests.

Immediate Comparisons

The immediate comparison API requests will all attempt to start the execution of the comparison immediately when the request is received and will return the result of the comparison as the response to the original HTTP request.

GET Request Example

In this example, the request compares two documents from source URLs and applies the specified rendering options to the resulting redline. The redline is then saved to the current directory as an RTF called **GetTest.rtf**.

Example CURL command

```
curl -X GET --header 'Accept: application/rtf' 'http://[your server name]/api/Compare?originalSourceUrl=http%3A%2F%2Finstall.workshare.com%2Fcompare%2FSampleOriginal.doc&modifiedSourceUrl=http%3A%2F%2Finstall.workshare.com%2Fcompare%2FSampleModified.doc&outputFormat=Rtf&renderingOptions=DETECT%20LIST%20NUMBERING%20CHANGES%3D1%3BCOMPARE%20HEADERS%2FFOOTERS%3D0' -o GetTest.rtf
```

Explanation of the CURL command

Snippet	Function
<code>curl</code>	Beginning the request.
<code>-X GET</code>	Type of request.
<code>--header 'Accept: application/rtf'</code>	This part of the request is not required but if used, should match the output format application. These can be found on the Swagger page.

Snippet	Function
<code>-o GetTest.rtf</code>	Tells the request to save the response as the named file in the current directory. The format should request the specified output format. For combined requests in output, this should be .zip.

Explanation of request parameters

In the example CURL command above, the request URL is as follows:

Root

`http://[your server name]/WorkshareCompareApi/api/compare`

The root will vary according to what you selected during the installation.

Rest of query:

`originalSourceUrl=http%3A%2F%2Finstall.workshare.com%2Fcompare%2FSampleOriginal.doc&modifiedSourceUrl=http%3A%2F%2Finstall.workshare.com%2Fcompare%2FSampleModified.doc&outputFormat=Rtf&renderingOptions=DETECT%20LIST%20NUMBERING%20CHANGES%3D1%3BCOMPARE%20HEADERS%2FFOOTERS%3D0'`

An explanation of each element of the request URL is in the table below.

Name	Example value	Comments
Server name	<code>http://[your server name]</code>	
Virtual Directory Path	WorkshareCompareApi	If you install the Workshare Compare API in the root of a new web site then this component will not be needed in the URL.
Endpoint	<code>/api/Compare</code>	

Name	Example value	Comments
originalSourceUrl	http://install.workshare.com/compare/SampleOriginal.doc	<p>The URL from which to fetch the original document. http:// and https:// are supported.</p> <div data-bbox="927 411 1385 506" style="border: 1px solid orange; padding: 5px;"> <p>Note 1: The files need to be available for HTTP/S requests.</p> </div> <div data-bbox="927 541 1385 831" style="border: 1px solid orange; padding: 5px;"> <p>Note 2: This parameter must be URL Encoded before it's included in your request URL. You can convert between decoded and encoded URLs at this site: http://meyerweb.com/eric/tools/dencoder/</p> </div>
modifiedSourceUrl	http://install.workshare.com/compare/SampleModified.doc	<p>The URL from which to fetch the original document. http:// and https:// are supported.</p> <div data-bbox="927 993 1385 1115" style="border: 1px solid orange; padding: 5px;"> <p>Note: Follow the same guidance as indicated in the notes for originalSourceUrl.</p> </div>
outputFormat	<p>One of:</p> <ul style="list-style-type: none"> • Rtf (default) • Wdf • Doc • DocX • Pdf • RedlineMI • RedlineMIAndRtf • RedlineMIAndDoc • RedlineMIAndDocX • RedlineMIAndPdf 	<p>The format of the returned comparison. When more than one format is specified (ie RedlineMIAndXXX) the result is returned as a ZIP file containing both formats. The default output format is RTF if this option is not specified.</p> <p>WDF is the comparison format used by the Workshare Compare desktop application (comparisons created in WDF format can only be opened in the Workshare Compare desktop application).</p> <p>RedlineMI is an XML-based format that describes the changes between the two documents in an easy-to-use manner (see Appendix A: Change Summary Information).</p>

Name	Example value	Comments
renderingOptions	DETECT LIST NUMBERING CHANGES=1;COMPARE HEADERS/FOOTERS=0'	<p>Optional rendering options to customize the comparison process and the format of the comparison document.</p> <p>Rendering options are best generated by creating a rendering set using the Workshare Compare desktop application. The contents of the saved .set file can be passed as the value for this parameter. Semicolons can be used in place of \r or \n as line separators in this parameter.</p> <div style="border: 1px solid orange; padding: 5px; margin-top: 10px;"> <p>Note: For more information about rendering sets, see the Compare Server Rendering Set Guide.</p> </div>

POST Request Example

In this example, the request compares two documents from the local machine and applies the specified rendering options to the resulting redline. The redline is then saved to the current directory as a DOC called **PostTest.docx**.

Example CURL command

```
curl -X POST --header 'Content-Type: multipart/form-data' --header
'Accept: application/rtf' -F outputFormat=Docx -F
"file1=@documents/Comp2.docx" -F "file2=@documents/Comp1.docx" -F
'renderingOptions=Display Workshare Compare Footers=1; Inserted
Text Color=8388736 ' 'http://[your server name]/api/Compare' -o
PostTest.docx
```

Explanation of the CURL command

Snippet	Function
curl	Beginning the request.
-X POST	Type of request.

Snippet	Function
<code>--header 'Content-Type: multipart/form-data'</code>	Format of the request. Note: Must always be included for POST requests.
<code>--header 'Accept: application/rtf'</code>	This part of the request is not required but if used, should match the output format application. These can be found on the Swagger page.
<code>-o PostTest.docx</code>	Tells the request to save the response as the named file in the current directory. The format should request the specified output format. For combined requests in output, this should be .zip.

Explanation of request parameters

In the example CURL command above, the request URL is as follows:

```
outputFormat=Docx -F "file1=@documents/Comp2.docx" -F
"file2=@documents/Comp1.docx" -F 'renderingOptions=Display
Workshare Compare Footers=1; Inserted Text Color=8388736 '
'http://[your server name]/api/Compare
```

An explanation of each element of the request URL is in the table below.

Name	Example value	Comments
Server name	http://[your server name]	
Endpoint	/api/Compare	
originalDocument	file1=@documents/Comp2.docx	The location of the original document on the local system.
modifiedDocument	file2=@documents/Comp1.docx	The location of the modified document on the local system.

Name	Example value	Comments
<code>outputFormat</code>	One of: <ul style="list-style-type: none"> • Rtf (default) • Wdf • Doc • DocX • Pdf • RedlineMI • RedlineMIAndRtf • RedlineMIAndDoc • RedlineMIAndDocX • RedlineMIAndPdf 	<p>The format of the returned comparison. When more than one format is specified (ie RedlineMIAndXXX), the result is returned as a .zip file containing both formats. The default output format is RTF if this option is not specified.</p> <p>WDF is the comparison format used by the Workshare Compare desktop application (comparisons created in WDF format can only be opened in the Workshare Compare desktop applicaion).</p> <p>RedlineMI is an XML based format that describes the changes between the two documents in an easy-to-use manner (see Appendix A: Change Summary Information).</p>
<code>renderingOptions</code>	Display Workshare Compare Footers=1; Inserted Text Color=8388736	<p>Optional rendering options to customize the comparison process and the format of the comparison document.</p> <p>Rendering options are best generated by creating a rendering set using the Workshare Compare desktop application. The contents of the saved .set file can be passed as the value for this parameter. Semicolons can be used in place of \r or \n as line separators in this parameter.</p> <div style="border: 1px solid orange; padding: 5px; margin-top: 10px;"> <p>Note: For more information about rendering sets, see the Compare Server Rendering Set Guide.</p> </div>

Queued Asynchronous Comparisons

The immediate comparison API requests described above will all attempt to start the execution of the comparison immediately when the request is received and will return the result of the comparison as the response to the original HTTP request. Depending on the client being used to send the HTTP requests, timeouts may occur if the files to be compared are large or the server is busy. In some cases, it may be difficult to adjust the timeout limits of the HTTP requests being made, particularly if intermediate servers such as a proxy server are involved.

The queued, asynchronous comparison API addresses the issues with HTTP timeouts and also helps to manage server load when comparisons are submitted in batches.

Requests to the queued comparison endpoint are constructed identically to the GET and POST requests that are sent to the immediate comparison endpoints described above. The only differences are the URL to send the request to - which must have '/enqueue' appended to it - and the response to the initial request - which is no longer the comparison result.

Making a queued comparison request

To make a queued comparison request, follow the instructions in [Immediate Comparisons](#) to make either a POST or GET comparison request, but send the request to:

```
http://[your server name]/WorkshareCompareApi/api/compare/enqueue
```

If your request is valid then you will immediately receive a response with an HTTP Accepted (202) status code, indicating that the comparison that you have submitted has been allocated a place in the queue. If the request for a comparison is obviously invalid in some way (for instance only one document is included) then you will receive an appropriate error response to your request.

Note: Other errors (for example, the supplied URLs not being correct or the supplied documents not being in a supported format) will not generate an error when the comparison is queued as they only become apparent when the server attempts to run the comparison. In these cases, the details of the error will be retrieved via the status polling described below.

If you submit a queued GET comparison request then the URLs that specify the location of the source documents will only be fetched once the comparison reaches the head of the queue and is ready to execute. You should therefore ensure that if these URLs are time-limited, they will not expire before the comparison reaches the head of the comparison queue.

Handling the response to a queued comparison request

The response to the queued comparison request includes a status URL.

Your code must inspect the 'location:' header of the response to recover the status URL that your client code can use to monitor the progress of the queued comparison. The location header will take the form:

```
location: http://[your server name]/WorkshareCompareApi  
/api/Compare/<comparison id>/status
```

For example:

```
location: http://localhost/WorkshareCompareApi/api/Compare/
4c9ec86ec8edb5db41da5dd7850c695db9b2538f06216
/status
```

The status URL returned in the location header may then be polled intermittently by your code to determine when the comparison is completed. You should avoid polling continuously to avoid unnecessary load on Compare Server, although the server itself will delay responding to polling requests where the comparison is not yet complete to limit polling frequency.

Depending on the state of the comparison, you may receive one of the following responses from the comparison status request:

Response	Explanation
404 Not Found	The comparison ID in the status URL is invalid.
404 Not Found	The comparison referenced in the status URL has already completed and been downloaded.
202 Accepted	The comparison is still in the queue. The response will contain the following custom headers: <ul style="list-style-type: none"> • X-queued-for: <seconds in queue so far> • X-queue-length: <number of comparisons in queue>
202 Accepted	The comparison is being executed. The response will contain the following custom header: <ul style="list-style-type: none"> • X-processing-for: <seconds processing so far>
302 Found	The comparison has completed. The location header of the response will contain the URL from which the comparison result can be downloaded.
An error (4XX or 5XX) status code	The comparison failed. The status reason phrase will contain more detail on why the comparison failed.

Note: When the comparison completes, the “302 Found” response that redirects to the download link for the comparison result may be handled automatically by your HTTP client software. This means that your code may actually receive a “200 OK” response with the comparison result as the response body. You should either write code to cope with this eventuality or disable automatically following redirects when making HTTP requests to the status URL. You will need to check your HTTP library documentation to determine how to disable automatically following redirects.

Once a queued comparison is completed, the server will only store the comparison result for a limited period. The result is deleted after either of these two conditions is satisfied:

- One hour has passed since the comparison completed
- Fifteen minutes have passed since the first attempt to download the comparison result was initiated.

If you try to request the status or result of a comparison after it has been deleted by the server, then you will receive a “404 Not Found” response from the server.

Data Storage on Workshare Compare Server

When comparisons are performed using immediate comparison API requests, the documents to be compared and the comparison result file are not stored on the disk storage of Compare Server except (in certain use cases) as temporary files which are deleted when no longer needed.

When comparisons are performed using queued asynchronous comparison API requests, the source documents and comparison results must be stored on disk while the comparison is queued and waiting to be run and between comparison completion and download of the comparison result. The following table indicates what data is stored on disk and when it is removed.

Source documents – POST comparisons	The source documents are stored from the time that the request is submitted and are deleted when the comparison execution is finished (either successful completion or due to a failure to compare).
Source documents – GET comparison	The source documents are not stored on disk; however the URLs that are to be used to retrieve the source documents are stored in the queue database.
Comparison result – GET and POST comparisons	The comparison result is stored from the time that the comparison completes until it is deleted according to the rules above.

DVJS Comparisons

What is DVJS?

DVJS (short for DeltaView JavaScript) is a component of Workshare Compare Server that allows comparisons to be viewed and reviewed in the user’s web browser. While it is simple to take a PDF format comparison document and view it in the browser, DVJS provides richer functionality such as change navigation, change categorization and potential for customization by the integrator.

Use of DVJS only requires a modern, standards-compliant, web browser – no other software needs to be installed on the end user’s PC or device.

Note: *The only version of Internet Explorer that will work correctly is version 11; older versions are not supported.*

DVJS is designed to be integrated into existing web-based applications and workflows – for instance it could be integrated into a web-based content management system or intranet portal allowing documents stored in the system to be compared without requiring them to be downloaded to a PC.

Getting started with DVJS

Compare Server ships with a very simple, sample comparison web page that allows users to select and compare two documents located on their PC or device. This also provides the best place to get started with understanding how DVJS works and how to customize and integrate it into existing systems.

To start using the sample comparison web page:

4. In your web browser, navigate to: `http://[your server name]/WorkshareCompareApi/compare`
5. Follow the instructions to select two documents and compare them.

You may, if you wish, use the sample comparison page in production, either by pointing your users to the URL given above or by taking, customizing and hosting on your own server the following files:

```
http://[your server name]/WorkshareCompareApi/Content/deltaview.js
http://[your server name]/WorkshareCompareApi/js/demo-utilities.js
http://[your server name]/WorkshareCompareApi/js/scripts.js
http://[your server name]/WorkshareCompareApi/css/demo.css
http://[your server name]/WorkshareCompareApi/css/normalize.css
```

In addition, you will need the HTML content returned from the request to:

```
http://[your server name]/WorkshareCompareApi/compare
```

You will need to adjust for any changes in the relative paths of these files and also change the ‘compareServerUrl’ setting in scripts.js to point to the URL of your installed Compare Server.

Note: *The file ‘deltaview.js’ is a Workshare copyright and you must not modify it or use it except in conjunction with Workshare Compare Server’s DVJS features. You may modify and use the content of the other files listed above as the basis for your own integration to the DVJS functionality.*

The DVJS APIs

DVJS is based on two APIs that are implemented by Compare Server:

1. `http://[your server name]/WorkshareCompareApi/api/uicompare`

This API can be used to initiate comparisons that are to be shown in the browser using DVJS. The uicompare API has all the options of the standard compare API (GET and POST requests, immediate or queued comparisons). See below for further information about using this API.

2. `http://[your server name]/WorkshareCompareApi/api/comparison`

This API provides the resources needed to render the comparison in the browser. This API is reserved for the use of the `deltaview.js` front end code. It is deliberately not documented and may be changed in future versions of Compare Server. You should not make calls to this API from your code. You will not be eligible for support on any issues you may encounter and future updates to Compare Server may break your implementation.

Starting DVJS comparisons

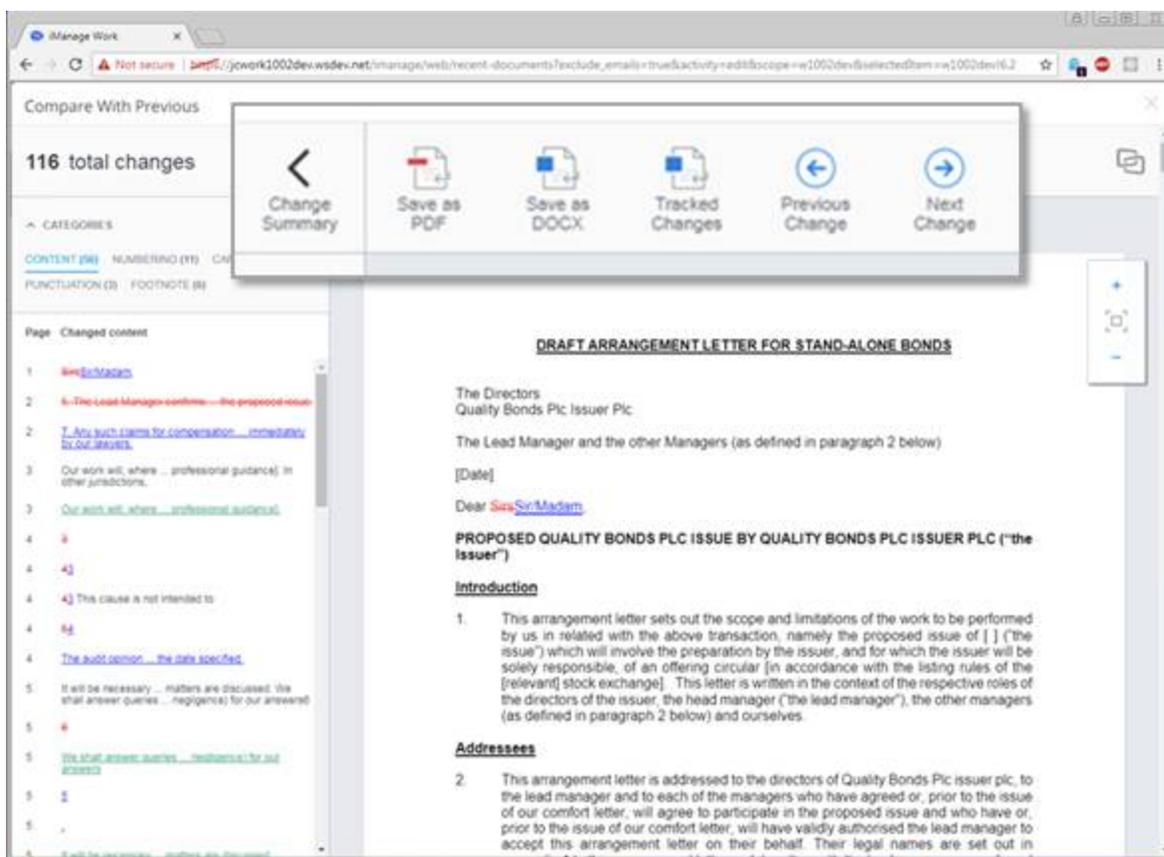
The simplest way to begin using DVJS is to perform a comparison by making a request to the uicompare API endpoint. All the options for performing a comparison against the regular compare API endpoint are available (GET and POST requests, immediate or queued comparisons). Refer to [Immediate Comparisons](#) and [Queued Asynchronous Comparisons](#) for full details on the parameters available and other details required to make a comparison request.

The response to a valid comparison request to the uicompare endpoint will have a Created (201) status code and will contain a 'location' header giving the URL at which the comparison result can be viewed.

In the case of queued requests to the uicompare endpoint, you must follow the procedure described in [Queued Asynchronous Comparisons](#) to check for the status of the comparison. Once the comparison is complete, the status request will return with a Found (302) status code and location header containing the URL at which the comparison result can be viewed.

Displaying a DVJS comparison

To display the comparison result to the user, the simplest approach is to browse to the URL that was obtained by performing the comparison. This will launch the DVJS redline viewer inside the browser window with no customization. The user will be able to explore the changes in the browser and download the resulting comparison in a variety of formats.



Note: There is no option to send the comparison via email using a desktop email client (for example Microsoft Outlook). This is not present because there is no widely supported mechanism for a web page to create an email with an attachment or content using a 'mailto:' link.

Customizing DVJS

For more control over the options available to the user when viewing the comparison, it is possible to create a web page which contains and customizes the DVJS viewer. Customizations can include

- Removing options from the heading buttons in order to prevent certain user actions such as downloading the comparison result.
- Adding options to the heading buttons in order to allow the user to take actions with the comparison that are specific to the integration. For example:
 - Saving the comparison back into a CMS or Intranet portal system
 - Emailing the comparison via a suitable webmail platform

The code below shows a minimal HTML page that will show the DVJS UI for a comparison, which can form a starting point for customization.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8"><meta name="viewport"
content="width=device-width,initial-scale=1">

  <title>DeltaView Document Comparison</title>
  <link href="path/to/deltaview.css" rel="stylesheet">
  <script type="text/javascript"
src="path/to/deltaview.js"></script>
  <script type="text/javascript">
      document.addEventListener("DOMContentLoaded",
function(event) {
    var options = {
      deltaviewId: 'deltaview-goes-here'
    };
    window.deltaview.init(options);
    // you must get the comparisonURL to this code - maybe in
a parameter of the URL
    window.deltaview.render(comparisonURL);
  });
</script>

</head>
<body>
  <div id="deltaview-goes-here"></div>
</body>
</html>
```

In order to turn the code above into a working sample, the following changes must be made:

- Ensure that the paths to the `deltaview.js` and `deltaview.css` files are set correctly
- Determine and implement a way for the `comparisonURL` variable to be set with the URL returned from the request to the `uicompare` endpoint. For testing purposes you may wish to perform a call to `uicompare` using the 'swagger' API test web page and then hard-wire the `comparisonURL` variable to the result

Customizing which elements of the UI are Visible

The visible elements of the UI can be customized by setting display options before calling `window.deltaview.init`. For example:

```
var options = {
  deltaviewId: 'deltaview-goes-here',
  display: {
    toolbar: true,
    statistics: false,
    redline: true,
    changeSummary: false,
  }
};
```

Customizing the Language of the UI

The UI language can be changed when calling `window.deltaview.init` by passing a second parameter. For example:

```
window.deltaview.init(options, window.deltaview.languages.italian);
```

Available predefined languages are currently: english, italian, french, spanish, mandarin. You can also define your own language.

If you want to add a custom language then you can define it in an object with the following structure and pass it as the second parameter to `deltaview.init()`:

```
window.dvlanguage = {
  toolbar: {
    pdf: 'Save as PDF',
    docx: 'Save as DOCX',
    tracked: 'Tracked Changes',
    summary: 'Change Summary',
    previous: 'Previous Change',
    next: 'Next Change'
  },
  changeList: {
    zeroState: 'Your documents have no changes.',
    totalChanges: 'total changes',
  }
};
```

```
originalDocument: 'Original',
modifiedDocument: 'Modified',
page: 'Page',
changedContent: 'Changed content',
categories: 'Categories'
},
changeListCategories: {
  footnote: 'footnote',
  endnote: 'endnote',
  header: 'header',
  footer: 'footer',
  //matching: 'matching', // not currently displayed
  suggested: 'suggested',
  //categorized: 'categorized', // not displayed - all
changes are children of this
  numbering: 'numbering',
  boilerplate: 'boilerplate',
  capitalization: 'capitalization',
  uncategorized: 'Content', // default first category
containing all ungrouped changes
  punctuation: 'punctuation',
  style: 'style',
  font: 'font',
  //whitespace: 'whitespace', //not currently displayed
  comment: 'comment'
},
loading: {
  pages: 'Loading pages...',
  comparing: 'Comparing...',
  firstDelay: 'Your documents are being compared. Please
be patient - comparing large documents can take some time.',
  secondDelay: 'Sorry for the delay, your documents are
still being compared. Please be patient - comparing large
documents can take some time.',
```

```

        error: 'Your documents have not been compared. Try
refreshing your page or start your comparison again.'
    }
};

```

Customizing the Toolbar

The following buttons are defined in the default toolbar and can be shown, hidden or customized by configuring the options before calling `window.deltaview.init`.

Button	Default Action
PDF	Download comparison in PDF format.
DOCX	Download comparison in DOCX format.
Tracked	Download comparison in DOCX format as tracked changes (not supported for comparisons with PDF source documents).
Email	No action (placeholder to allow email integration with webmail systems).
Print	No action.
Previous	Navigate to previous change.
Next	Navigate to next change.

For example, to hide the pdf button and show the print button:

```

var toolbar = {
  pdf: {display: false },
  print: {display: true },
};
var options = {
  deltaviewId: 'deltaview-goes-here',
  display: {
    homeButtons: toolbar,
    toolbar: true,
    redline: true,
    statistics: false,
    changeSummary: false,
  }
};

```

Appendix A. Change Summary Information

This appendix describes the schema for the Change Summary produced with a comparison and the character set values.

RedlineML

RedlineML is now the preferred format for extracting change summary information. It contains the entire content of the Redline document in an easy-to-work-with XML format.

RedlineML Schema

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="http://workshare.com/2010/RedlineML"
  elementFormDefault="qualified"
  xmlns="http://workshare.com/2010/RedlineML"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
>

  <xs:element name="document" type="documentT"/>

  <xs:simpleType name="cellStatusT" final="restriction" >
    <xs:restriction base="xs:string">
      <xs:enumeration value="normal" />
      <xs:enumeration value="inserted" />
      <xs:enumeration value="deleted" />
      <xs:enumeration value="moveSource" />
      <xs:enumeration value="moveDestinate"/>
      <xs:enumeration value="dead"/>
      <xs:enumeration value="padding"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:group name="content">
    <xs:sequence>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="paraMarker"/>
        <xs:element ref="change"/>
        <xs:element ref="field"/>
      </xs:choice>
    </xs:sequence>
  </xs:group>
</xs:schema>
```

```
<xs:element ref="bkmk"/>
<xs:element name="table" type="tableT"/>
<xs:element name="shape" type="shapeT"/>
<xs:element name="blob" type="blobT"/>
<xs:element name="pict" type="blobT"/>
<xs:element name="run" type="runT"/>
<xs:element ref="endNote"/>
<xs:element ref="footNote"/>
<xs:element ref="textbox"/>
<xs:element ref="comment"/>
</xs:choice>
</xs:sequence>
</xs:group>

<xs:attributeGroup name="insertedDeletedAttrs">
  <xs:attribute name="isInserted" type="xs:boolean"
use="optional" default="false"/>
  <xs:attribute name="isDeleted" type="xs:boolean"
use="optional" default="false"/>
</xs:attributeGroup>

<xs:group name="section">
  <xs:sequence>
    <xs:element type="sectionMarkerT" name="sectionMarker"/>
    <xs:group ref="content"/>
  </xs:sequence>
</xs:group>

<xs:group name="changeContent">
  <xs:sequence>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="paraMarker"/>
      <xs:element ref="field"/>
      <xs:element ref="bkmk"/>
    </xs:choice>
  </xs:sequence>
</xs:group>
```

```
<xs:element name="shape" type="shapeT"/>
<xs:element name="blob" type="blobT"/>
<xs:element name="pict" type="blobT"/>
<xs:element name="run" type="runT"/>
<xs:element ref="endNote"/>
<xs:element ref="footNote"/>
<xs:element ref="textbox"/>
<xs:element ref="comment"/>
</xs:choice>
</xs:sequence>
</xs:group>

<xs:complexType name="shapeT">
  <xs:group ref="content"/>
  <xs:attributeGroup ref="insertedDeletedAttrs"/>
</xs:complexType>

<xs:complexType name="blobT">
  <xs:attributeGroup ref="insertedDeletedAttrs"/>
</xs:complexType>

<xs:element name="change" type="changeT"/>
<xs:complexType name="changeT" >
  <xs:group ref="changeContent"/>
  <xs:attribute name="number" type="xs:integer" use="required"/>
  <xs:attribute name="type" type="xs:integer" use="required"/>
  <xs:attribute name="crossref" type="xs:integer"
use="optional"/>
</xs:complexType>

<xs:complexType name="documentT">
  <xs:sequence>
    <xs:group ref="content"/>
  </xs:sequence>
</xs:complexType>
```

```
<!--sadly the compositor can spit out content before its
first section marker-->
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:group ref="section"/>
  </xs:sequence>
</xs:sequence>
<xs:anyAttribute/>
</xs:complexType>

<xs:complexType name="sectionMarkerT">
  <xs:sequence>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element name="header" type="headerfooterT"/>
        <xs:element name="footer" type="headerfooterT"/>
      </xs:choice>
    </xs:sequence>
  </xs:sequence>
  <xs:attributeGroup ref="insertedDeletedAttrs"/>
</xs:complexType>

<xs:complexType name="tableT">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element name="row" type="rowT"/>
  </xs:sequence>
  <xs:attributeGroup ref="insertedDeletedAttrs"/>
</xs:complexType>

<xs:complexType name="rowT">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element name="cell" type="cellT"/>
  </xs:sequence>
  <xs:attributeGroup ref="insertedDeletedAttrs"/>
</xs:complexType>
```

```
<xs:complexType name="cellT">
  <xs:group ref="content"/>
  <xs:attribute name="cellStatus" type="cellStatusT"
use="optional" default="normal"/>
  <xs:attribute name="isInsertedColumn" type="xs:boolean"
use="optional" default="false"/>
  <xs:attribute name="isDeletedColumn" type="xs:boolean"
use="optional" default="false"/>
  <xs:attribute name="column" type="xs:integer" use="optional"/>
  <xs:attribute name="spanInfoOriginal" type="xs:string"
use="optional"/>
  <xs:attribute name="spanInfoModified" type="xs:string"
use="optional"/>
  <xs:attribute name="spannedInOriginal" type="xs:boolean"
use="optional"/>
  <xs:attribute name="spannedInModified" type="xs:boolean"
use="optional"/>
  <xs:attribute name="isMerged" type="xs:boolean"
use="optional"/>
</xs:complexType>

<xs:element name="textbox" type="textboxT"/>
<xs:complexType name="textboxT">
  <xs:group ref="content"/>
</xs:complexType>

<xs:element name="paraMarker" type="paraMarkerT"/>
<xs:complexType name="paraMarkerT">
  <xs:attribute name="listNumber" type="xs:string"
use="optional"/>
  <xs:attributeGroup ref="insertedDeletedAttrs"/>
  <xs:attribute name="isInsertedListItem" type="xs:boolean"
use="optional" default="false"/>
  <xs:attribute name="listLevel" type="xs:int" use="optional"/>
  <xs:attribute name="listId" type="xs:int" use="optional"/>
</xs:complexType>
```

```
<xs:complexType name="runT" mixed="true">
  <xs:attribute name="font" type="xs:string" use="required"/>
  <xs:attribute name="wasListNum" type="xs:boolean"
default="false" use="optional"/>
  <xs:attribute name="wasField" type="xs:boolean"
default="false" use="optional"/>
  <xs:attribute name="rtl" type="xs:boolean" default="false"
use="optional"/>
</xs:complexType>

<xs:complexType name="commentT">
  <xs:group ref="content"/>
  <xs:attributeGroup ref="insertedDeletedAttrs"/>
</xs:complexType>

<xs:element name="comment" type="commentT"/>

<xs:element name="footNote" type="footEndNoteT" />
<xs:element name="endNote" type="footEndNoteT" />
<xs:complexType name="footEndNoteT">
  <xs:group ref="content"/>
  <xs:attributeGroup ref="insertedDeletedAttrs"/>
</xs:complexType>

<xs:complexType name="headerfooterT">
  <xs:group ref="content"/>
  <xs:attribute name="type" type="hdrftrType" use="required"/>
  <xs:attributeGroup ref="insertedDeletedAttrs"/>
</xs:complexType>

<xs:simpleType name="hdrftrType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="left"/>
  </xs:restriction>
</xs:simpleType>
```

```
<xs:enumeration value="right"/>
<xs:enumeration value="first"/>
<xs:enumeration value="main"/>
</xs:restriction>
</xs:simpleType>

<xs:element name="field" type="fieldT"/>
<xs:complexType name="fieldT">
  <xs:sequence>
    <xs:element type="fieldCodeT" name="fieldCode"/>
    <xs:element ref="fieldResult" minOccurs="0"/>
  </xs:sequence>
  <xs:attributeGroup ref="insertedDeletedAttrs"/>
</xs:complexType>

<xs:complexType name="fieldCodeT">
  <xs:sequence maxOccurs="unbounded" minOccurs="1">
    <xs:choice>
      <xs:element ref="field"/>
      <xs:element name="run" type="runT"/>
      <xs:element ref="paraMarker"/>
      <xs:element name="blob" type="blobT"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>

<xs:element name="fieldResult" type="fieldResultT"/>
<xs:complexType name="fieldResultT">
  <xs:group ref="content"/>
</xs:complexType>

<xs:element name="bkmk" type="bkmkT"/>
<xs:complexType name="bkmkT">
  <xs:attribute name="name" type="xs:string" use="required"/>
```

```

    <xs:attribute name="start" type="xs:boolean" use="required"/>
  </xs:complexType>

</xs:schema>

```

Character set values

ANSI_CHARSET	0
DEFAULT_CHARSET	1
SYMBOL_CHARSET	2
SHIFTJIS_CHARSET	128
HANGEUL_CHARSET	129
HANGUL_CHARSET	129
GB2312_CHARSET	134
CHINESEBIG5_CHARSET	136
OEM_CHARSET	255
JOHAB_CHARSET	130
HEBREW_CHARSET	177
ARABIC_CHARSET	178
GREEK_CHARSET	161
TURKISH_CHARSET	162
VIETNAMESE_CHARSET	163
THAI_CHARSET	222
EASTEUROPE_CHARSET	238
RUSSIAN_CHARSET	204
MAC_CHARSET	77
BALTIC_CHARSET	186

ChangeT values

0	DELETION
1	MOVESOURCE
2	MOVEDESTINATION
3	INSERTION
4	FORMAT_CHANGE
13	MOVEDDELETION
14	STYLECHANGE_TEXT
15	STYLECHANGE_LABEL

 Workshare Ltd.
© 2018. Workshare Ltd. All rights reserved.

Copyright

Workshare Professional and Workshare DeltaView are registered trademarks of Workshare Ltd. Workshare Compare, Workshare Protect, Workshare 3, Workshare DeltaServer, SafetyGain, and the Workshare logo are trademarks of Workshare Ltd. All other trademarks are those of their respective holders.

Trademarked names may appear throughout this guide. Instead of listing these here or inserting numerous trademark symbols, Workshare wishes to state categorically that no infringement of intellectual or other copyright is intended and that trademarks are used only for editorial purposes.

Disclaimer

The authors/publishers of this guide and any associated help material have used their best efforts to ensure accuracy and effectiveness. Due to the continuing nature of software development, it may be necessary to distribute updated help from time to time. The authors would like to assure users of their continued best efforts in supplying the most effective help material possible.

The authors/publishers, however, make no warranty of any kind, expressed or implied, with regard to Workshare programs or help material associated with them, including this guide. The authors/publishers shall not be liable in the event of incidental or consequential damages in connection with, or arising out of, the programs or associated help instructions.

Revisions

Published for Workshare Compare Server 9.5: 14/9/18
Revised for Workshare Compare Server 9.5.1: 8/1/18
Revised for Workshare Compare Server 9.5.2: 17/5/18

Workshare Ltd., 20 Fashion Street, London E1 6PX www.workshare.com