

2021.1
Developer Guide

Contents

1. Developer Guide for WorkZone PDF 2021.1	4
2. What's new	5
3. API documentation	8
4. WorkZone PDF configuration	10
4.1 About WorkZone PDF Engine configuration settings	10
4.2 Defining parameters for multiple instances of the PDF Engine	12
5. Custom reports	14
5.1 About custom reports	14
5.1.1 Custom reports overview	14
5.1.2 Localization of custom report templates	16
5.1.3 Report references	17
5.2 Create a report template	20
5.2.1 The OData query	20
5.2.2 Create the Report JSON file	21
5.2.3 Create the XML model and the report templates	28
5.2.4 Create the Word template	30
Add the content controls	31
Example – Custom report Word template	37
5.2.5 Create the Excel template	38
5.2.6 Create the HTML template	43
5.2.7 Distribute custom reports to WorkZone Content Server	44
5.3 Update	44
5.3.1 Update the XML model and the templates	45
5.3.2 Preview custom reports by using the POST request	45
5.4 Report options	46
5.4.1 Cover pages	47
5.5 Troubleshooting custom reports and tips	49
6. PDF options	51
6.1 Custom headers and footers	51
6.1.1 Header and footer alignment and margins	56
6.1.2 Examples of custom headers and footers	57

6.2 Watermarks in reports	59
6.3 Global page numbering	60
6.4 Remerging documents	61
7. Terms and conditions	63

1. Developer Guide for WorkZone PDF 2021.1

You can use the WorkZone PDF Engine as a Web service or as a NuGet package.

In the first case, you need to [install the WorkZone PDF Engine](#), and then the API description will be available at `<PDF Engine url>/Help`, for example, [Sandbox](#).

In the second case, you can add the `KMD.WorkZone.Engine` NuGet package, which is available in the [KMD repository](#), to your project.

You must have an account to access the repository. If you do not have an account, contact KMD.

Related product documentation

- [WorkZone PDF Administrator Guide](#)
- [API documentation via Swagger UI](#)

WorkZone links

- [WorkZone documentation](#)
- [WorkZone support](#)
- [WorkZone website](#)
- [WorkZone portal](#)

2. What's new

2021.1

Adjusted PDF remerge logic

The initial Microsoft Word document and attached files are now embedded to a PDF file. When there is a request to remerge the PDF file, the embedded Word document is changed, and then new PDF file with the same attachments is generated.

2021.0

- Date format from the WorkZone Content Server is used in the report templates except when the format is specified manually.

2020.3

- Reports were added to the NuGet package for the WorkZone PDF service.

2020.2

- The procedure of reports update in the same release was simplified. The **Version** parameter now includes hotfixes such as builds. See [Create the Report JSON file](#).

2020.1

No changes in this release.

2020.0

No changes in this release.

2019.3

No changes in this release.

2019.2

No changes in this release.

2019.1

- Detailed information on how to access [API documentation](#) has been added to this guide. Note that WorkZone PDF API documentation is published via Swagger UI. This provides additional functionality such as API testing, code building, and other features.

2019.0

- You can [remerge Word documents](#) before converting to PDF.

2018.1

- **Deploy Custom Reports during installation or update:** By placing your custom report templates in the **Reports** sub-folder of the same folder where the KMD WorkZone PDF.exe program is located, you can automatically deploy your custom reports during installation or update when selecting the mandatory **Database Configuration** option.
- **Reports API replaces KMD.WorkZone.ODataQueries.TemplatingEngine.exe:** The Reports API has been extended to be able to create the XML model used for custom reports as well as to create the custom report templates themselves (Excel spreadsheets and Word documents). The Reports API replaces the

KMD.WorkZone.ODataQueries.TemplatingEngine.exe application which has been deprecated as of this release.

- **WorkZone Content Server language settings used by default:** If the WorkZone Content Server is used to process requests, for example if a report request uses OData connections, the language settings of the WorkZone Content Server are used unless the Accept-Language header is specified in the request. If the Accept-Language header is not specified and the WorkZone Content Server is not used to process requests, the en-GB language code is used as a default.

2018.0

The first version of the Developer Guide.

3. API documentation

Prerequisite: If your WorkZone PDF Engine instance uses Windows authentication, you must enter your WorkZone credentials to access WorkZone PDF API documentation.

WorkZone PDF API documentation contains API that help you adjust PDF settings, report settings, and communicate with WorkZone PDF.

Open the API documentation

1. Enter `http://<WorkZone URL>/Render/help` in a browser, for example, `http://db01/Render/help`.
2. You can see all available documentation for WorkZone PDF. Select what you need:
 - **API documentation via Swagger UI** – View a list of available API and access additional functionality such as API testing and code building.
 - **OpenAPI resources for advanced scenarios** – Open API as a JSON file to use it for the advanced features. See [Swagger official page](#).

Localization and language

You can use the Accept-Language header of the POST action in the WorkZone PDF API to specify the culture-specific text formatting for PDF conversion and to specify the expected language of the report content and selection of the correct localized report template. In general, the Accept-Language header is used by all aspects of the WorkZone PDF product to specify language and culture. Currently you can specify en-GB (for English) or da-DK (for Danish).

The language code is used for the report depends on the language settings of the WorkZone Content Server as well as the Accept-Language value.

If the Accept-Language header has been specified

If the Accept-Language header has been specified, the Accept-Language header will override all other options in the report and the language code contained the Accept-Language header will be used. The language code specified on the WorkZone Content Server will not be used.

If the Accept-Language header is not specified

If the Accept-Language header is not specified and if the API route used implies an OData connection, the language code specified on the WorkZone Content Server will be used. If OData-related requests are utilized, the Accept-Language header should be omitted from the Report request.

If the Accept-Language header is not specified and if the API route does not imply an OData connection, the language code "en-GB" will be used as a default.

4. WorkZone PDF configuration

4.1 About WorkZone PDF Engine configuration settings

A WorkZone PDF Engine configuration setting can be defined multiple places in the WorkZone program, for example you can define default values upon installation of the WorkZone PDF Engine and then specify different configuration parameters through the WorkZone Configurator or in the WorkZone PDF Engine web.config file as well as specifying parameters in the JSON file when generating PDF reports in the WorkZone Client.

Missing parameters

Parameters that are not implemented in the WorkZone Configurator module can be manually added to the **WZPDF_CONFIGURATION** table by using the **ScanSQL** program.

Parameter priorities

The following list illustrates the priority ranking of each WorkZone PDF Engine parameter, with the highest ranked parameter placed first.

The priority is for each individual parameter and not the entire WorkZone PDF Engine parameter stack.

1. Body: Parameters specified in the body of the report JSON file.
2. HTTPS Header: Parameters specified in the header of the API call.
3. Database settings: WorkZone PDF Engine settings specified by using the WorkZone Configurator or by using the **ScanSQL** program directly into the database in the **WZPDF_CONFIGURATION** table.
4. Web.Config settings: Settings specified in the Web.Config file.
5. Installation default values; The settings specified when the WorkZone PDF Engine was initially installed.

Example:

Parameter location	HTTP Header	Header Style, Font	Header Style, color
API Body	Overview	Font: Arial	-
Configuration Database*	-	Font: Comic Sans	-
Web.Config file	Central Reports	-	-
Installation default	Reports	Font: Verdana	Blue

* In the **WZPDF_CONFIGURATION** table.

In the example above

- The header "Overview" will be displayed in the Arial font when a PDF report is generated using the specific API call that contains the Header parameter in the body. If another PDF report is generated that does not use the API call, the header will be "Central Reports", displayed in the Comic Sans font.
- If the Web.Config file is deleted or of the Header and Header Style, Font parameters in the Web.config file are deleted, the PDF report will contain the header: "Reports" and the Header font: Comic Sans if the report is generated without using the API call.
- If the Web.Config file is deleted or of the Header and Header Style, Font parameters in the Web.config file are deleted, the PDF report will contain the header: "Overview" and the Header font: Arial if the report is generated using the API call.
- The header color will be Blue for all PDF reports as there are no parameters defined header color in the API Body, configuration database or the Web.config file.

Tip: If your installation of WorkZone PDF Engine contains many parameter settings in the web.config file and you want to keep the settings, make sure the values for the specific parameter settings are empty or undefined in the Configuration database.

4.2 Defining parameters for multiple instances of the PDF Engine

You can invoke multiple instances of the WorkZone PDF Engine, each with its own set of pre-defined parameters by using the **Target** PDF Engine custom parameter in the web.config file for the PDF Engine.

For example, the PDF Engine invoked to process PDFs from the WorkZone SmartPost module could be invoked with its own set of parameters while another instance of the WorkZone PDF Engine could be invoked with parameters defined for that instance only.

Each instance of the WorkZone PDF Engine and WorkZone PDF Crawler can therefore be defined with its own set of parameters.

Create parameter settings for PDF Engine instances

Since WorkZone PDF Engine and WorkZone PDF Crawler custom parameter settings are stored in the WZPDF_CONFIGURATION table, you should first create the parameter settings in the WZPDF_CONFIGURATION table for each instance you expect to use.

Creating the parameter settings first will give you a list of valid PDF Engine instances as well as their unique names, making it easier to specify which instance to invoke in the PDF Engine web.config file later.

Entries in the WZPDF_CONFIGURATION table can be made using the **ScanSQL** program distributed with the WorkZone product.

Example: Create parameter settings for SmartPost PDF Engine instance

In this example, a new set of parameter settings is created in the WZPDF_CONFIGURATION table for the PDF Engine instance named "SmartPost".

```
INSERT INTO WZPDF_CONFIGURATION(TARGET, NAME, CUSTOM_VALUE)
        VALUES('SmartPost', 'Watermark', 'PROTECTED COPY');
```

Unique instance names

You must uniquely name each instance you expect to use in the **Target** field of the WZPDF_CONFIGURATION table. Any custom parameters specified for that instance will be applied when the instance is invoked from the web.config file using the **Target** engine custom parameter.

Default instances

The WZPDF_CONFIGURATION table contains one named instance with the default value: PDFENGINE. You can specify as many instances in the WZPDF_CONFIGURATION table as you need.

Specify the PDF Engine instance in the web.config file

Once you have created the PDF Engine instances you want to use and defined the custom parameters for each individual instance, you can update the PDF Engine web.config file, using the **Target** PDF engine custom parameter to specify which instance you want to invoke.

Any custom parameters specified in the WZPDF_CONFIGURATION table will be applied to the invoked instance.

Incorrect Target parameter

If the **Target** PDF Engine custom parameter is incorrect - either because it is misspelled or because the instance named in the **Target** parameter does not correspond to the instance named in the Target field of the WZPDF_CONFIGURATION table, the request will fail with a 404 Not Found error and the error message: *The configuration for target"... " is not found..*

Custom Parameters not in the database

For custom parameters that are only specified in the request API or web.config files, you must include the custom parameters in the request since they will not be available from the WZPDF_CONFIGURATION table.

5. Custom reports

Reports in WorkZone Client allow you to gain a better overview of your cases and documents. Reports are usually run from the WorkZone Client program and are converted as PDF files.

Standard and custom reports

WorkZone Client contains several standard reports, which are available after a standard installation of the product.

The standard reports include a general classification scheme, which lists the case groups, case titles, and responsible units (if any), and a document list, which lists all parties and attached documents for a specific case.

WorkZone Client also contains tools that enable you to run customized reports in order to better reflect the reporting requirements of specific customers.

View reports

You can display your reports on the screen, send them directly to a printer or save them in another location.

5.1 About custom reports

5.1.1 Custom reports overview

You can create custom reports and make the reports available for users. Customized reports are made accessible to users in the **Reports** menu in WorkZone Client after they have been deployed.

OData queries and Word/Excel templates

WorkZone reports are based on OData queries and Microsoft Word or Microsoft Excel templates. OData queries are contained in the Report JSON in a file tree view and are used to generate the XML models used in the Word or Excel templates. Word or Excel templates are

corresponding office files that contain XML models and Content Controls (placeholders for data).

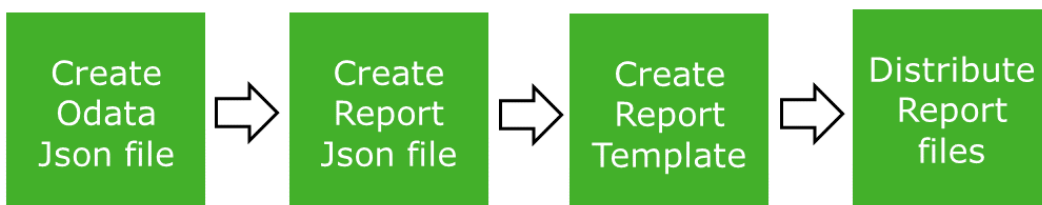
Report JSON file

When the custom report template is completed, a Report JSON file must be created. The Report JSON file contains information and metadata about the custom report template and any translated versions as well as all OData requests or queries used to populate the Content Controls in the template when the report is run.

Install the report

When the Report JSON file is completed, all custom report templates and the Report JSON file must be moved to the Report folder of the installation folder. The installer must then be run again in order to update the database and make the new custom report available for users through the **Reports** menu in WorkZone Client.

The custom report creation process



The custom report creation process

- Create and test the OData queries you want to use to generate the custom report data.
- Create the Report JSON file, and specify the titles and file names of the report templates as well as all other OData queries and JSON properties.
- Create the report template (Word or Excel) with the XML Model embedded by using the Report JSON file.

- In Microsoft Office, refine the report template by using the XML model to create content controls as well as normal Word/Excel formatting.
 - Create translated templates by copying the original report template and translating all text in the document. Do not translate the text inside the content controls.
- Add the report templates and the Report JSON file to the Report folder and deploy the report using the installer to make the reports available in the WorkZone Client.

5.1.2 Localization of custom report templates

Report templates can be localized, so that translated into other languages by directly translating all actual text in the report template and then saving the report template with a different name – for example, DocumentReport_DK for the report in Danish and DocumentReport_US for the same report in English.

The localized reports are later referenced in the Report JSON file to differentiate between the languages.

Additionally, based on the Accept-Language header value or the WorkZone Content Server language setting, the generated XML model is localized as well, and this can be seen from the model's node names and default text inside Content Controls on the report template files (.docx or .xlsx).

Tip: It is advisable to create the original report first, update the Report JSON file, and verify that the report is correctly deployed and displayed in WorkZone Client before creating translated versions of the report. You will still have to update the Report JSON file for every translated report you create.

Localizing content controls

Do not translate the placeholder text in the content controls, as the placeholder texts will be overwritten when the report is run.

5.1.3 Report references

Use report references to connect two or more reports in order to create a single, complex report as a combination of several simpler reports.

Report references are created and maintained in the Report JSON file.

Each report reference must contain the Report ID of the referenced report as well as all necessary Report Bindings. You can also specify the sorting order of the report references by setting an Order property for each referenced report.

The creation of reports consists of these general steps

1. Retrieve the report template (Word, Excel, or HTML)
2. Retrieve the report data in JSON format by using parameters obtained from WorkZone Client.
3. Bind the report data to the report template.
4. Include any document cover-pages or contents to the report.
5. Create subreports from the report references.
6. Generate parameters for the subreport from the report data (step 2) by using report by bindings.
7. Repeat steps 1-5 for each subreport and parameters.

The properties of the report references

Report ID

The unique identification number of the report, which is expressed as a GUID. Each report has a unique alphanumeric number used to identify and locate the report in the system, and the Report ID is used to locate and connect to the specific report you want to reference.

Bindings

Report Bindings describe how to obtain the parameters for the subreport from the report data.

You must specify the data you want returned in the properties of the Bindings object.

For each property, you must specify a property name and a property value.

The property name is the parameter name in the referenced report, and the property value is a JPath to the parameter data of the main report.

Example: Case and Record report

In this example, the Case report is the main report and the Record report is the subreport.

For the Case report, the report data would be: { Title: 'Case title', FileRecords: [{ ID: 1 }, { ID: 2 }, { ID: 3 }] }

Since the Record report (the subreport) uses the RecordKey as a parameter, the report binding would be { RecordKey: '\$.FileRecords[*].ID' }.

This report binding will result in the three different parameters: RecordKey = 1, RecordKey = 2, and RecordKey = 3, which are used to create 3 Record subreports.

Notes:

- If report bindings are not required to connect the reports - for example, if the sub-report does not contain any parameters - the bindings should be specified as an empty JSON.
- If the report bindings are incorrectly specified, a report will be created without subreports, because the connections required for the subreport would fail.

Limitations

Report references support single-value and multivalued parameters, but, if the subreport contains two or more single-value parameters, the entire report will fail.

- **Single-value parameters:** If the subreport contains the RecordKey: '' parameter (single-value parameter), the report binding will result in multiple subreport parameters: RecordKeys: [1], RecordKeys: [2], RecordKeys: [3].

In the example above, the three ID properties will result in three RecordKey parameters, because the RecordKey is a single-value (non-array) parameter.

- **Multivalue parameters:** If the subreport contains the RecordKeys: [] parameter (multivalue parameter), the report binding will result in only one subreport parameter RecordKeys: [1, 2, 3].
- **Single-value and Multivalue parameters:** If the subreport contains RecordKey: '' and RecordKeys: [] parameters (one single-value and other multivalued parameters), the report binding result in three subreport parameters:
 - RecordKey = 1 and RecordKeys: [1, 2, 3]
 - RecordKey = 2 and RecordKeys: [1, 2, 3]
 - RecordKey = 3 and RecordKeys: [1, 2, 3]
- **Two or more single-value parameters:** If the subreport contains the RecordKey1: '' and RecordKey2: '' parameters (two or more single-value parameters), the report binding will fail, and an error will be thrown.

Order

The order determines in which sequence the referenced reports are added to the referencing (main) report. Any number between 0 and 99 can be used as input, and the referenced reports will thereafter be sorted in ascending order by this value, for example, 1, 5, 7 11, 13, etc.)

If the Order object is not included in the report references or is incorrect, reports will be added to the report references as they are found rather than in a specified order.

Report references example

In this example, the Case report contains a report reference to the Record report. The Case report is the main report and the Record report is the subreport. The sorting order value is set as "1".

```
{
  "ReportId": "36a4cb00-2e92-4b5a-ac18-c1dbce60718d",
  "References": [
    {
      "ReportId": "2a66c0a3-f3c9-4729-a9d9-29dcf8edf381",
      "Bindings": {
        "RecordKey": "$.Files.FileRecords[*].RecordKey"
      },
    },
  ],
}
```

```
        "Order": 1
      }
    ],
    "Version": null,
    "EntityType": "File",
    ...
```

5.2 Create a report template

To create a Word or Excel report template, use this section as a guidance. Note that you must follow steps in the particular order:

1. [Create OData query](#)
2. [Create the report JSON file](#)
3. [Create the XML model](#)
4. Create the [Word](#), [Excel](#) or [HTML](#) template
5. [Distribute reports to WorkZone Content Server](#)

5.2.1 The OData query

OData queries are used to retrieve document data for the custom report and to create the XML model used to bind or merge retrieved data to the placeholders in the Word or Excel report templates (the Content Controls)

You cannot use Odata queries to retrieve document data directly from the WorkZone Content Server. Instead you must use the Report JSON file to contain the OData queries you want to use for the report.

The OData queries in the Report JSON file are based on standard OData queries using WorkZone specific rules of organizing OData queries and are are organized in a special tree structure in the Report JSON file.

Create OData queries

OData Queries are queries into the existing OData model that return requested data.

The OData queries can be created with a valid OData address (for example, `http://db01/OData`) and verified with existing data. You can use an Internet browser (for example, Microsoft Internet Explorer) and the WorkZone Query Builder tool to compose the OData queries.

You can also create the OData queries directly in the Report JSON file, but it is advisable to create and test OData queries before you create the Report JSON file in order to ensure that the OData queries return the requested document data correctly.

Example: OData query

You want to create a report that retrieves specific document metadata as well as any supplementary documents.

A query is created to retrieve the document metadata, such as type, summary, ID, and title.

```
http://db01/OData/Records  
( '1' )?$select=RecordKey,RecordNo,Title,RecordType_Summary
```

If you want data about the parent case, you can use the **Expand** parameter for the appropriate entity and request its properties in a **Select** option.

```
http://db01/OData/Records ( '1' )?$select=  
File/FileNo,File/Title&$expand=File
```

If you want Supplementary Documents for the document, you can modify the query as follows:

```
http://db01/OData/Records  
( '1' )/SupplementaryDocuments?$select=RecordKey,Title
```

5.2.2 Create the Report JSON file




The Report JSON file is a JSON file that contains all information necessary for successful execution of the report and is used to update report data in the WorkZone database.


The WorkZone PDF Installer program uses the Report JSON file to install custom reports when the **Configure Database** target option is selected during the PDF installation. You can also manually update existing reports and add new ones to the WorkZone database, but it is not recommended.

After you have created the OData queries, you must create a Report JSON file. As JSON files are normal text files, you can use any text-editor program to create and edit the Report JSON file.

The Report JSON file contents

The Report JSON file contains the following information

Parameter	Type, Length	Description	Required
ReportId	String, 38	A unique identifier, i.e. a GUID.	
Version	Number, 12	The version number of the report. The format is 8 to 10 ordinal numbers: for example, xxxxxxxxxx. The following algorithm is used for forming a version number: major+minor+build+revision (xx.xx.xxxxx.xx). If "minor" and "revision" parts contain 0, 0 may be omitted: 0x = x.	
EntityType	String, 30	The entity type for which the report is intended, for example, Case, Record, Party, Task, Process.	
Localizations	String, 255	The name, description, culture name, and file name (one or several localizations are allowed).	
OdataQueries	String, 2000	The OData queries used by the Report JSON.	
Parameters	String, 2000	JSON with parameters that will pass into the OData.	
Caption	String, 255	In reports, the caption value can be used in the following context:	

Parameter	Type, Length	Description	Required
		<ul style="list-style-type: none"> as a first-level bookmark (table of content item) in the PDF file in the custom headers or footers as a title of a converted PDF file that is saved on WorkZone Content Server <p>In Report JSON, you must specify a path to the caption value. See How to compose a JSON Path.</p> <p>Example:</p> <p>Report JSON: { "Files": { "UserKey": "C-123" } }</p> <p>A path in Report JSON: <code>\$.Files.UserKey</code></p> <p>Caption value: C-123.</p>	
DefaultOutput	String, 12	The file type of the report requested by WorkZone Client. Use the file extension in the upper-case format, for example, PDF, XLSX, DOCX, PBIX, etc.	
AccessCode		The read rights on report (see WorkZone Client permissions documentation).	
UpdateCode		Specifies Update rights on report (see WorkZone Client permissions documentation).	
System		Specifies type of the report: True – standard (deployed as a part of installation); False – customized for the customer's needs.	

Parameter	Type, Length	Description	Required
Default		Defines which report of the same EntityType WorkZone Client will use as the default one.	

The OData query section in the Report JSON file

The OData query section of the Report JSON file must contain the following:

- A root (or main) element
- A tree structure of elements, from the root entity to deeper related entities that correspond to parts of the resource path in the OData URL.
- A #Query property under each node with a value that either corresponds to the query options in OData URL (starts from "?") or contains the empty value "" if there are no query options for current resource.

If you want to retrieve an array of objects, you must encompass the objects in question with square brackets ([]).

Parameters passed to the Report JSON must be prefixed with @.

Example: OData section of the Report JSON file

```
{
  "Records": [
    {
      "#Query": "(' @RecordKeys ')?$select=File/FileNo,
      File/Title,RecordNo,RecordKey,Title,RecordType_Summary&$expand=File",
      "SupplementaryDocuments": [
        {
          "#Query": "?$select=RecordKey,Title"
        }
      ]
    }
  ]
}
```

Additional examples of OData queries can be seen in the **Standard Reports** in the **Reports** table (OData_queries column)

Tips:

- Do not include the same fields for the same entities in different places in the OData section.

For example, if you specify a field under a branch, make sure that the same field does not appear for parent elements when using the expandcommand.

- Check the syntax of the JSON file prior to executing the file. You can use various internet sources for the syntax check, for example, <http://jsoneditoronline.org/>.

Report JSON example

The following is an example of a Report JSON file. There are two custom report Word templates:

- DocListSuppDocs_En.docx (English version)
- DocListSuppDocs_Da.docx (Danish version)

Note the "Parameters" section

Tip: Remember to use hard brackets (`[]`) for multiple values. If you expect to only pass single values, use normal brackets (`{}`), according to standard JSON rules.

Example:

```
{
  "ReportId": "5c7f89d4-a0db-4365-bc08-48ffa7f193b0",
  "Version": null,
  "EntityType": "Records",
  "Localizations": [
    {
      "Name": "DocListSuppDocs",
      "Description": "A document list with supplementary documents",
      "CultureName": "en-GB",
      "FileName": "DocListSuppDocs_En.docx"
    },
    {
      "Name": "DocListSuppDocs",
```

```
        "Description": "En dokumentliste med bilag",
        "CultureName": "da-DK",
        "FileName": "DocListSuppDocs_Da.docx"
    }
],
"ODataQueries": {
    "Records": [
        {
            "#Query": "('@RecordKeys')?$select=File/FileNo,File/Title,
            RecordNo,RecordKey,Title,RecordType_Summary&$expand=File",
            "SupplementaryDocuments": [
                {
                    "#Query": ")?$select=RecordKey,Title"
                }
            ]
        }
    ]
},
"Parameters": {
    "RecordKeys": []
},
"DefaultOutput": "PDF",
"AccessCode": null,
"UpdateCode": "CONFIGADM",
"System": true,
"Default": true
}
```

The following report JSON examples are taken from the report JSON files included in the WorkZone suite installation and are therefore available to all users and not customized for any specific customer.

Agenda report JSON

The following example of a report JSON file is taken from the Agenda report which is installed with the WorkZone suite.

```
{
    "ReportId": "297ecb46-f2eb-49cd-b213-51146b225d3f",
    "Version": null,
    "EntityType": "Agenda",
    "Localizations": [
        {
            "Name": "Agenda",
            "Description": "An agenda with documents list",
            "CultureName": "en-GB",
```

```

        "FileName": "Agenda_En.docx"
    },
    {
        "Name": "Dagsorden",
        "Description": "Dagsorden med documentliste",
        "CultureName": "da-DK",
        "FileName": "Agenda_Da.docx"
    }
],
"ODataQueries": {
    "FileAgendas": {
        "#Query": "('@AgendaKey')?$select=SortGroup",
        "FileRef": {
            "#Query": ")?$expand=Text&$select=Title,Text/Text",
            "Records":
            [
                {
                    "#Query": ")?$expand=Record&$select=Record/RecordKey,Record/Title,Record/RecordType_Summary,Record/Extension,Record/State_Value"
                }
            ]
        }
    }
},
"Parameters": { "AgendaKey": "" },
"Caption": "$.FileAgendas.FileRef.Title",
"DefaultOutput": "PDF",
"AccessCode": null,
"UpdateCode": "CONFIGADM",
"System": true,
"Default": true
}

```

Case report JSON

The following example of a report JSON file is taken from the Case report which is installed with the WorkZone suite.

```

{
    "ReportId": "36a4cb00-2e92-4b5a-ac18-c1dbce60718d",
    "Version": null,
    "EntityType": "File",
    "Localizations": [
        {
            "Name": "Case",
            "Description": "A case with parties and documents",
            "CultureName": "en-GB",

```

```

        "FileName": "Case_En.docx"
    },
    {
        "Name": "Sag",
        "Description": "En sag med parter og dokumenter",
        "CultureName": "da-DK",
        "FileName": "Case_Da.docx"
    }
],
"ODataQueries": {
    "Files": {
        "#Query": "('@FileKey')?$expand=Parties/Name",
        "FileRecords": {
            "#Query": "('@FileKey')?$expand=Parties/Name",
            "FileRecords":
            [
                {
                    "#Query": "('@RecordKeys')?$expand=Parties/Name&$filter=Parties/Name eq 'A"
                }
            ]
        }
    }
},
"Parameters": {
    "FileKey": "",
    "RecordKeys": [ ]
},
"Caption": "$.Files.Summary",
"DefaultOutput": "PDF",
"AccessCode": null,
"UpdateCode": "CONFIGADM",
"System": true,
"Default": true
}

```

5.2.3 Create the XML model and the report templates

The XML Model used as a data source for the content controls in the report template is created using the WorkZone Reports API and the information contained in the Report JSON file - specifically the OData queries and the names of the report templates (Word documents, and Excel spreadsheets).

You can create the XML Model and the Word or Excel templates in one go or you can create an XML Model only and then use the XML Model in the Report templates (Word documents or Excel spreadsheets) you create afterwards.

Tip: Before you start, make sure the OData queries are valid and the OData URI is accessible.

Creating the XML Model

Open the WorkZone Reports API in a 3rd party Web debugger tool. While WorkZone allows you to access to the Reports API, there is no built-in application or program to manipulate the Reports API. You can use any 3rd party product to access the WorkZone Reports API, for example, Telerik Fiddler.

The GET action

Use the GET action of the Reports API and specify the URI of the Report JSON file that is to be used for the custom report as the parameter. The GET action of the Reports API only has this one parameter but if you do not specify the parameter correctly, the XML model and report template creation will fail.

When the GET action is executed, two things might occur, depending on the whether or not the report templates specified in the Report JSON in the **FileName** parameter exist in the same folder as the Report JSON file.

- If the folder containing the Report JSON file does not contain the report template(s) specified in the **FileName** parameter of the Report JSON, the report templates will be created with the XML Model embedded.
- If the folder containing the Report JSON file already contains the report template(s) specified in the **FileName** parameter of the Report JSON, the XML Model embedded in the report templates will be updated with the new XML Model.

Additional options

You can inspect the OData queries used in the Report API to create the XML model and to populate the report templates as well as display the XML Model generated by the Reports API by using the Accept header while making request to the Reports API.

The Accept header can be set to the following values:

- Text/Plain: Returns the OData queries used in the Report JSON file.
- Application/XML: Returns the XML model that will be generated by the GET action of the Report API.

Localizing the XML model

The generated XML Model can be localized based on the Accept-Language header value or the WorkZone Content Server language setting. The localization can be seen from the model's node names and default text inside content controls.

5.2.4 Create the Word template

You can create the Word template with the embedded XML model in one go by using the GET action of the Reports API.

Tip: Before you start, make sure the OData queries are valid and the OData URI is accessible.

To create the Word template

- Open the WorkZone Reports API in a 3rd party Web debugger tool.
- Use the GET action of the Reports API and specify the URI of the Report JSON file that is to be used for the custom report as a parameter to the GET action. If the Word document template which name is specified in the **FileName** parameter of the Report JSON is located in the same folder, then WorkZone PDF injects the XML model into the template and returns it to the user. Otherwise a blank template is created.

If the Word document template specified in the Report JSON already exists in the same location as the Report JSON file, the XML model of the Word document template will be updated with the new XML model created by the Report JSON.

After the report template has been created, you can modify the Word document template to conform to whatever requirements you have for the report the Word template is to produce by placing content controls and ordinary text in the template.

Additional options

View the OData queries

You can view the OData queries used to generate the XML model in the Report JSON by using the **Text/Plain** parameter in the Accept header while making requests to the Reports API.

View the XML Model

You can view the XML model which is generated by the Report JSON by setting the Accept header to **Application/XML** value while making requests to the Reports API.

Add the content controls

Content controls are placeholders for document data in Word templates. When the report is run, the XML model (custom XML) inside the Word template is merged with data retrieved from WorkZone Content Server with help of the OData queries in the Report JSON file. The data from custom XML is inserted in the content control place holders, according to the binding defined inside content controls.

Place the content controls where you want the document data displayed in the template. Different types of content controls can be selected, but, in most cases, it will be sufficient to select **Plain text**.

Another often used type of content controls is the **Date Picker**. The **Date Picker** content control enables you to define the date format directly in content control properties, and it can therefore contain data in date format.

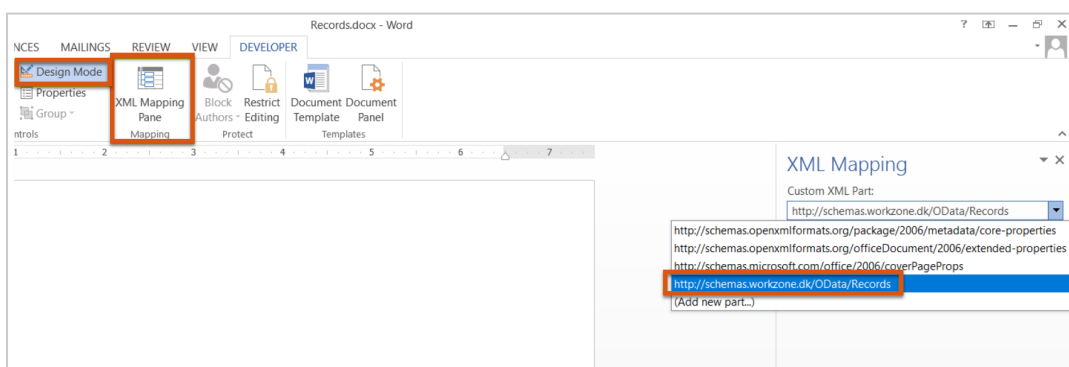
Note: You must be able to access the **Developer** tab in order to define and place content controls in the Word template.

To add a content control to the template

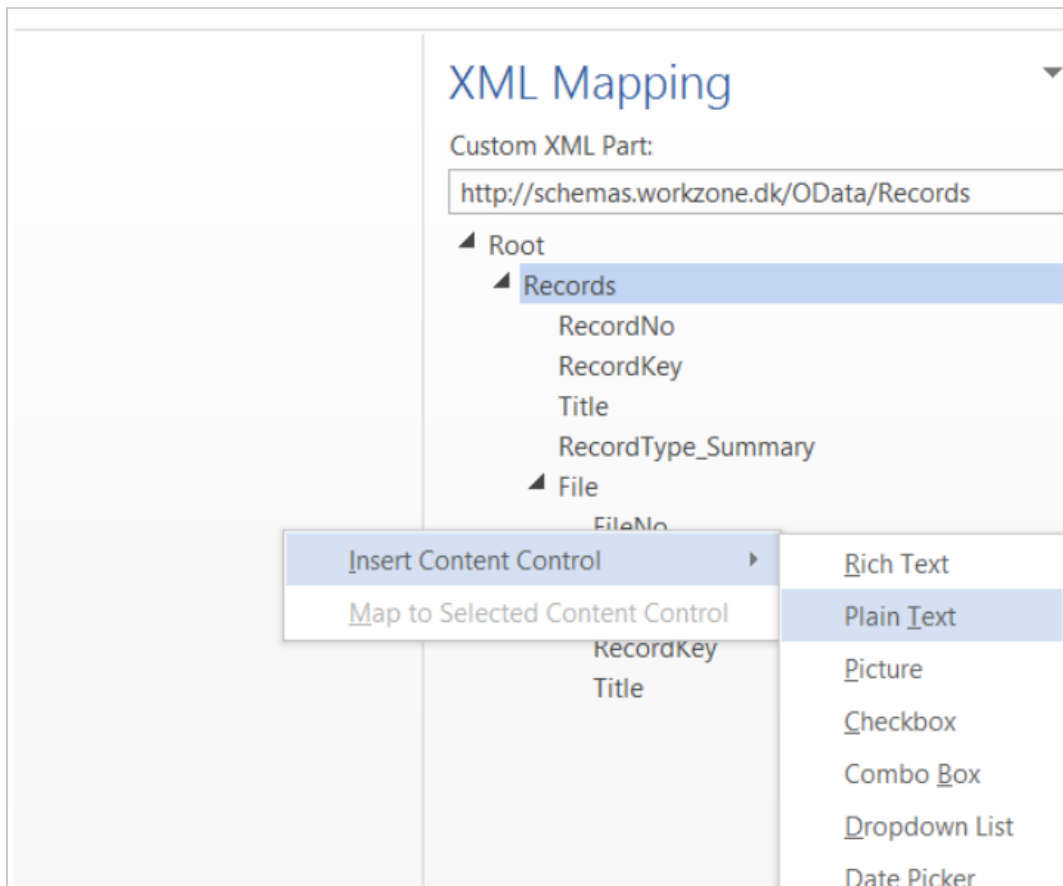
1. In the Word template, click the **Developer** tab > **XML Mapping** Pane to open the **XML Mapping** pane.
2. In the **XML Mapping** pane, select the XML model you want to create content controls for.
3. In the Word template, place the mouse cursor where you want the content control to be displayed.
4. In the tree view in the **XML Mapping** pane, right-click the data element you want displayed in the template, select **Insert Content Control**, and then select the element type, typically **Plain text**. The content control will be placed at the mouse cursor position.

Example: Creating content controls

In this example, the Records XML model located at <https://schemas.workzone.dk/Odata> has been selected.



When inserting the content control in the template, the records in the Record XML model and the Plain text option have been selected.



Content control properties

Define titles and access rights to the content controls in a document by directly editing the content control properties.

To set up the content control properties, select the content control in the report template, and either right-click, and select **Properties**, or click **Properties** in the **Developer** tab > **Controls** group.

Repeating content controls

A repeating content control repeats all content in the content control, including other content controls.

If you want to display a running list of elements attached to a main record, for example, display main document metadata in the upper half of the reports and display any

supplementary documents attached to the main document in the lower half of the report, you must use repeating content controls to contain the data content controls.

Repeating content controls can also contain and repeat other content controls, for example, inner and outer lists. See nested repeating content controls below.

To create repeating content controls

1. In the Word template, create a table, and place the table in the template.
2. Select the table row that is to contain the repeating content control.
3. In the **Developer** tab > **Controls** group, select the **Repeating Section Content Control** button to insert a repeating content control.

You can also right-click the data element in the **XML Mapping** pane and select the **Repeating** to insert a repeating content control.

4. Place the mouse cursor in the table cell where you want to insert a content control.
5. In the tree view in the **XML Mapping** pane, right-click the data element you want displayed, select **Insert Content Control**, and then select the element type, typically **Plain text**. The content control will be placed at the mouse cursor position.
6. Repeat steps 4 and 5 for each data content control you want represented in the table.

Tables should be used to contain repeating content controls in order to facilitate PDF conversion, but you can place repeating content controls directly in the report template if you want.

Example: Simple repeating content controls in a table row

In this example, a table has been created in the report template and made into a repeating section content control.



You can display content controls with start and end tags by selecting **Start/End tag** in the **Show as** field in the **Content Controls Properties** form.



The **RecordNo**, **RecordKey** and **Title** content controls have been added to selected repeating table cells, enabling the table to create new rows for each new attached document.



Nested repeating content controls

If you want to create a report with nested data - for example, create a list of supplementary documents for each main document - place the repeating control controls for the supplementary document inside the repeating content controls for the main document. This corresponds to nested loops in programming, where the supplementary document repeating content control is the inner loop and the main document repeating content control is the outer loop.

For example

- Main Document A
 - List of supplementary documents for Document A
- Main Document B
 - List of supplementary documents for Document B
- Main Document C
 - List of supplementary documents for Document C

- Main Document D
 - List of supplementary documents for Document D

Example: Nested repeating content controls

In the example below, the red content controls are repeating content controls for the main document data, and the blue content controls are repeating content controls for the supplementary document data.

Document List

Parties and Supplementary Documents

Records ()		Records ()
Document number:	Record No.: RecordNo (RecordNo)	
RecordKey (RecordKey)	Type: RecordType_Sum (RecordType_Summary)	
Title: Title (Title)		
Case No.: FileNo (FileNo)	Case title: Title (Title)	
Supplementary documents		
Document number	Title	
RecordKey (RecordKey)	Title (Title)	

Placeholder text in content controls

If the content control does not contain data, the placeholder text for the content control will still be displayed in the report when the report is run.

You can hide content control placeholder text for content controls that contain no data by adding a space after the content control end tag in the report template and then pressing backspace to delete the space. The content control tags will change appearance.

Example: Placeholder is text not displayed when data does not exist

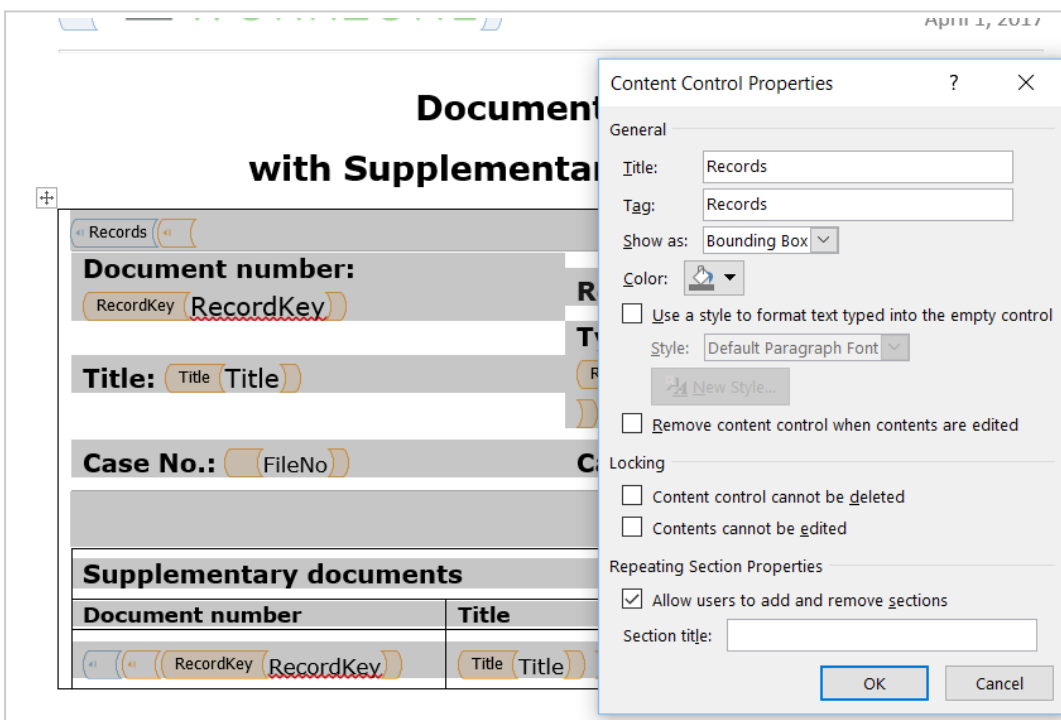
In this example, the **FileNo** and **Title** placeholder text for the content controls will be displayed in the report even when there is no data to display.



After the trailing space has been removed, the **FileNo** and **Title** content controls will not display placeholder text when there is no data to display, and they will change appearance:



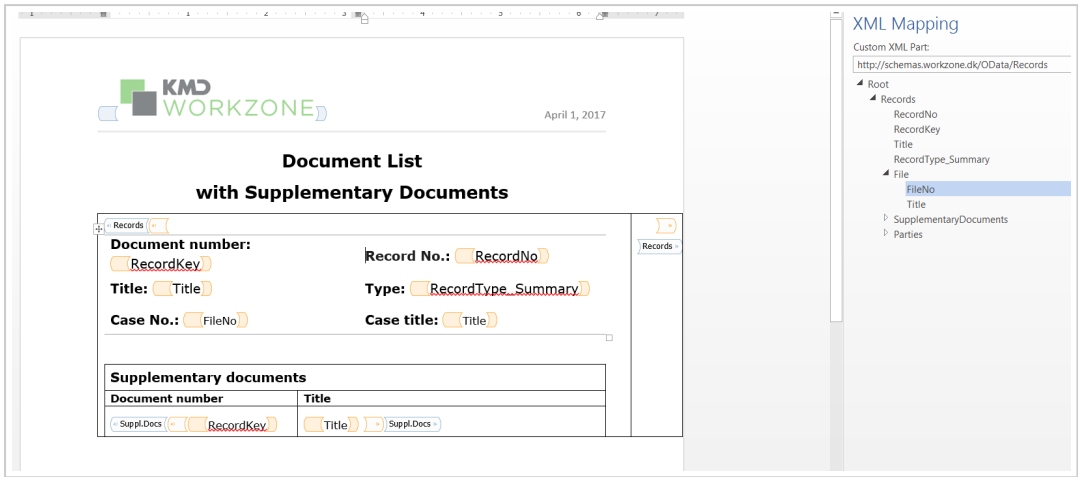
Tip: Define a title and/or tag for the content control in the **Content Control Properties** form to improve your overview of the relations between repeating elements and their corresponding entities.



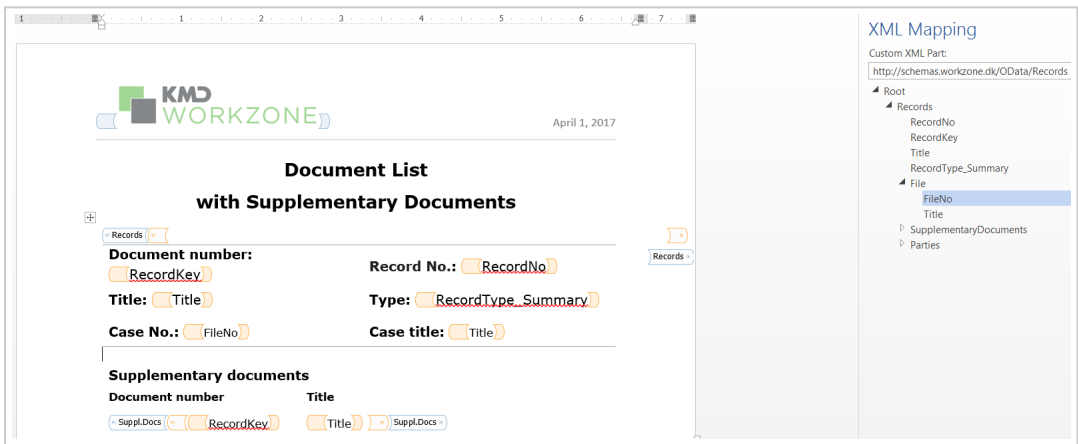
Example – Custom report Word template

The following images are screen shots from an example of a document list report that includes supplementary documents (if any).

The first template could look like this



With standard Word styling and formatting applied, the final template might end up looking like this:



5.2.5 Create the Excel template

You can create the Excel template with the embedded XML model by using the GET action of the Reports API.

Tip: Before you start, make sure the OData queries are valid and the OData URI is accessible.

Generate Excel template

1. Open the WorkZone Reports API in a 3rd party Web debugger tool, for example, Telerik Fiddler.

2. Use the GET action of the Reports API and specify the URI of the Report JSON file to be used for the custom report as a parameter to the GET action. Specify the template's language in the Accept-Language header.
3. The OData queries are used to build XML model document where corresponding OData entities and their relations are described without actual values. The XML model is used to automatically generate a corresponding XML schema, which formally describes validation rules for the elements in the XML model document.
4. The Excel report template is created. It has the name specified in the localized **FileName** parameter of the Report JSON. Save the Excel template to the same folder as the source Report JSON file.

Note: If the Excel report template specified in the Report JSON already exists in the same location as the Report JSON file, the XML Schema of the Excel template will be overwritten with the new XML Schema created by the Report JSON.

5. Adjust the generated XML schema to your actual business model and OData entities:
 - Use the POST action of the Reports API and specify the URI of the Report JSON file as a parameter to the POST action. Specify the template's language in the Accept-Language header and define some other parameters as test data.
 - If you have received the **200 Success** response – your XML schema is valid, please skip steps below.
 - If you have received the following error – **400 [XmlException] XML is not valid according to specified XML Schemas** – then read the issue's description in **Message**:

```
{
  "Exception": {
    "Type": "XmlException",
    "Message": "XML is not valid according to specified XML Schemas.",
    "Source": "KMD.WorkZone.Common",
    "StackTrace": "...",
    "InnerExceptions": [
      {
        "Type": "XmlSchemaValidationException",
        "Message": "The element 'Parties' in namespace
'http://schemas.workzone.dk/OData/Records' has invalid child element
'NameCode' in namespace 'http://schemas.workzone.dk/OData/Records'.
The element 'Records' in namespace
'http://schemas.workzone.dk/OData/Records' has invalid child element
'Created' in namespace 'http://schemas.workzone.dk/OData/Records'.
List of possible elements expected: 'Infoes, Classification' in
namespace 'http://schemas.workzone.dk/OData/Records'. The element
'Root' in namespace 'http://schemas.workzone.dk/OData/Records' has
invalid child element 'Records' in namespace
'http://schemas.workzone.dk/OData/Records'."
      }
    ]
  }
}
```

- Change the Excel template extension to .zip and open it. Move **xmlMaps.xml** from the **xl** folder outside the .zip package. The xmlMaps.xml file contains generated XML schema.
- Fix found and potentially dangerous issues in **xmlMaps.xml** by aligning it to the WorkZone Content Server data model.
- Overwrite the **xl\xmlMaps.xml** file with the updated one. Change the file extension back to .xlsx.
- Repeat first step again to verify XML schema.

Customize Excel template

Now you have a report template with the embedded XML model. It is currently empty and not yet adjusted to your company's needs.

To create a ready-to-use template, proceed with the following steps:

1. Open the generated template.
2. Ensure that you have the **Developer** tab added to the main ribbon. Click the **Source** button to open the **XML Source** pane.

See how to add the Developer tab to your Excel application:

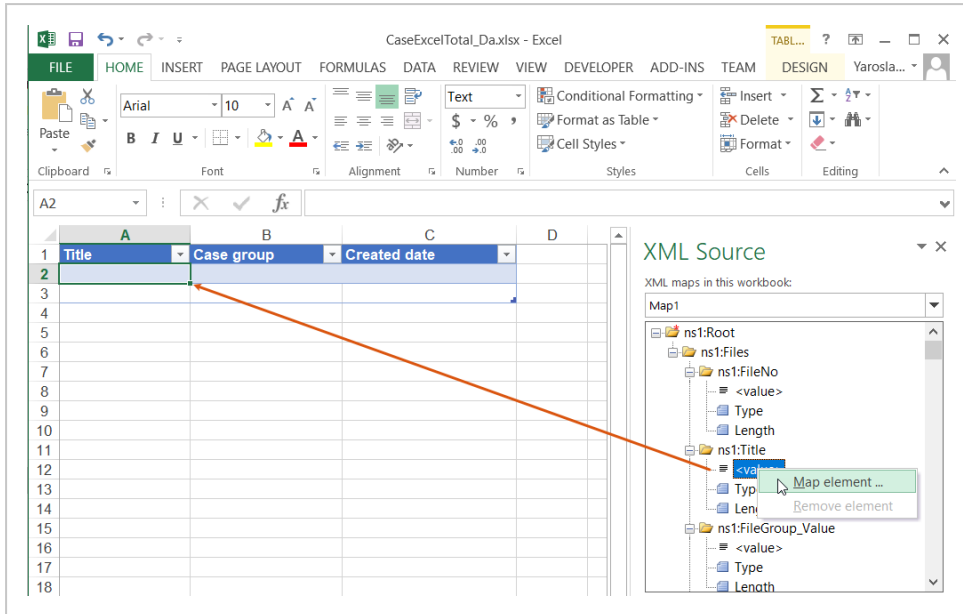
1. On the **File** tab, go to **Options > Customize Ribbon**.
 2. Select **Main Tabs** under **Customize the Ribbon**.
 3. Select the **Developer** check box and click **OK**.
3. Insert a Table object. Number of columns must reflect requirements to the report.
 4. Bind table cells and XML elements manually or by using formulas.

Manually

- In the Excel template, select the cell in which you want the XML element to be displayed.
- In the **XML Source** pane, right-click the needed data element and select **Map Element**.

Tip: You can also drag the XML element from the **XML Source** pane into the desired cell.

Note: XML elements in Excel are repeating by default. This means that, unlike the content controls in the Word template, you do not have to define special repeating XML elements to manage the display of multiple data records.



By using formulas

Use custom formulas to concatenate, format, or group values of XML elements.

Specify a custom formula in the needed cell in the following format: `Path -> Format`, where

- **Path** is a path to the needed element. Use character '/' as delimiter.
- **Format** is a string of properties separated by the '&' character. Optionally, you can define additional parameters by using the following delimiters:
 - `:` – define a type
 - `=` – define a style
 - `""` or `"` – define constants.

Examples:

- Files -> VFacetings/Path
- Files -> Parties/CustomLabel_Value & " - " & Parties/NameType_Value & " " & Parties/NameCode_Value & ": " & Parties/Name/Name1 & " " &

Parties/Name/Name2

- Files -> Infoes/CustomLabel_Value & ":" & Infoes/Info_Value
- Files -> Dates/CustomLabel_Value & ":" & Dates/DateStamp:DateTime=dd-MM-yyyy

5. Click **Save**.

6. Test the customized report template. To do this, use the POST action of the Reports API.

View OData query and XML model

- You can view the OData query used to generate the XML model in the Report JSON by using the **Text/Plain** parameter to the ACCEPT request to the Reports API.
- You can view the XML model which is generated by the Report JSON by setting the Accept header to **Application/XML** value when you make requests to the Report API.

5.2.6 Create the HTML template

The HTML template for the report is based on the Razor syntax. See more about Razor in the [Microsoft documentation](#).

Change logo in the report

If you want to change the logo in the report, we recommend that you embed the image data directly into the HTML template. Convert your image to base64 using any online conversion tool and embed the string into the template. Below is an example of the code used to embed the image. In this code, replace the **base64 string** placeholder with the string that you get for your image from the conversion tool.

```

```

5.2.7 Distribute custom reports to WorkZone Content Server

Once you have created the report template file from Microsoft Word or Excel and any localized (i.e. translated) versions of the report template as well as the Report JSON file, you can use the WorkZone PDF Installer to distribute the report(s) to your WorkZone Client users.

1. Create a folder called **Reports** in the same folder where the **KMD WorkZone PDF.exe** is located.
2. Place all custom report templates you want deployed during an installation or update in the **Reports** folder.
3. Run the WorkZone PDF installer. When the **Database Configuration** option is run, any reports placed in the **Reports** folder will be automatically deployed to the WorkZone Client by the installation process.

When the PDF Installer installation is completed, the **Reports** table will be updated with new entries for the custom reports.

Note: The **Template_key** corresponds to the Record Key of the report template document, which contains a Word template.

If you need to edit the report template, you can find the document by using WorkZone Client. After you have edited the document, you can save the document to the WorkZone Content Server.

Alternatively, you can create a new document with the Word template and then update the **Template_key** to the Record Key of the new report template.

5.3 Update

Update XML model

If you want to add new fields to the report template, you must update the embedded XML model in the Word or Excel templates. See [Update the XML model and the templates](#).

Update template body

If you want to complement the report template with any other changes, for example, to add a new content control for existing fields in XML model, then you must update the report template in WorkZone Client. To do this, open the document in WorkZone Client and apply your changes. See [Open a document in corresponding program](#).

5.3.1 Update the XML model and the templates

You can update the embedded XML model in the Word or Excel templates after it has been used to create Content Controls in Word or as an XML Source file in Excel, for example, if new fields have been introduced or existing fields have been deleted.

To update the XML model, proceed with the following steps:

1. Update the OData query in the Report JSON with the new fields you want included in the report templates and test the OData query.
2. Place the report template files (.docx or .xlsx) to be updated in the same folder as the Report JSON file .
3. Run the GET action of the Report API, defining the URI of the Report JSON. The XML model will be updated and the report templates will be updated with the new XML model.

Tip: Remember to save the report templates and reinstall the PDF Engine from the WorkZone Content Server to distribute the updated report, and update the Reports table with any updated or new field properties, such as JSON parameters.

Tip: See how to create the templates: [Create the Word template](#), [Create the Excel template](#), and [Create the HTML template](#).

5.3.2 Preview custom reports by using the POST request

Usually, you must re-install the PDF Engine in order to update the Report table, transfer the report templates, and update the Report/Print menu in WorkZone Client.

If you want to view the custom report during development, you can use a POST request to the Reports API with the Report JSON URI parameter to display your custom report instead. Using

POST requests will not deploy your custom report and make it available to WorkZone client users but it will display your report as it is, enabling you to review the report and make any required changes.

To view the custom report, you must provide:

- The relevant report parameters inside the POST request body, e.g. "{Parameters: {\"FileKeys\":[\"681\"]}}\" for the FileKey (Case) 681 same way the final report POST request is defined.
- A URI to the Report JSON file in the ReportJSON parameter of the POST request.

Note: In order to use the POST request to display the report, the report templates must be placed in the same location as the Report JSON.

Tip: You can see more information about creating POST requests for retrieving a report in the Report API documentation.

The KMD WorkZone PDF.exe program

The KMD WorkZone PDF.exe is used to install new versions of WorkZone as well as update existing versions. The program contains standard report templates which also are deployed during the installation process.

Note: Custom report templates in the **Reports** folder will overwrite any standard report templates contained in the KMD WorkZone PDF.exe program if their names are identical. This way, you can create your own version of the standard reports in WorkZone and be able to keep your changes during program updates.

5.4 Report options

When creating custom reports as well as running standard WorkZone reports you can define various report options in the report JSON file to further adjust how the final report is displayed.

The following options can be used with your custom reports.

Report options

- **Cover pages:** Include a cover page containing meta data for the following document for each document in the report. See [Cover pages](#).

PDF options

- **Bookmarks:** Change the default bookmarks by modifying the individual report request to use case and document titles as bookmarks.
- **Custom headers and footers:** Define your own header or footer globally or for each and define the alignment and placement of the header or footer as well.
- **Custom Watermark:** Define your own watermarks globally or for each local document and define the watermark formatting (color, font and/or transparency)
- **Global Page numbering:** Number all pages of a global PDF document or report in the header or footer of the document.

See [PDF options](#).

5.4.1 Cover pages

Cover pages contain meta data for the document in question, such as case number, document title, contacts. Cover pages can be generated with the report. They can also be included in the output file when the report document is converted and the report data contains information about documents.

You can set up a report to include a cover page for every document in the report and you can create a report that only consists of cover pages from each document by not including any underlying documents in the report, effectively creating an overview document of the underlying reports.

The option to include cover pages and/or document contents in the report must be specified as the following parameters:

- **IncludeDocumentCoverPage:** Generate a cover page for the report (True/False)
- **IncludeDocumentContents:** Generate report contents (True/False)

Set the relevant parameters' values for the specific report in the body of the report JSON file:

To only generate the cover page

- **IncludeDocumentCoverPage:** True
- **IncludeDocumentContents:** False

To generate the report without a cover page

- **IncludeDocumentCoverPage:** False
- **IncludeDocumentContents:** True

To generate the report with a cover page

- **IncludeDocumentCoverPage:** True
- **IncludeDocumentContents:** True

Note: The JSON file for each specific report must be edited in order to generate a cover page for the report.

Exapmle: **IncludeDocumentCoverPage** and **IncludeDocumentContents** in the Report JSON:

Header

```
Content-type: application/json
OData: http://db01/OData
Accept: application/pdf
```

Body

```
{
  Parameters: { FileKey: '141', RecordKeys: ['20', '18'] },
  IncludeDocumentCoverPage: true,
  IncludeDocumentContents: true
}
```

Adjusting the JSON file - \$Select clause specified

If the JSON report contains a specific **\$Select** clause for the underlying table, you must specifically include the **RecordKey** and **State_Value** fields in the underlying table in the

\$Select statement of the report Odata queries section in order to create cover pages and /or include underlying document contents in the report.

In the report JSON example, Agenda report (**See Also** section below), the **\$Select** statement in the Odata query is specified directly and the query must specifically contain the **RecordKey** and **State_Value** fields.

```
#Query": "?$expand=Text&$select=Title,Text/Text",
"Records": [
  {
    "#Query": "?$expand=Record&$select=Record/RecordKey,Record/Title,
    Record/RecordType_Summary,Record/Extension,Record/State_Value"
```

Adjusting the JSON file - \$Select clause not specified

If the JSON report does not contain a specific **\$Select** clause for the underlying table, all fields are by default selected, including the **RecordKey** and **State_Value** fields and you do not need to specifically include the **RecordKey** and **State_Value** fields.

In the Report JSON example, Case report (**See Also** section below), the **\$Select** statement in the Odata query is not needed and not specified. Therefore, the query does not specify the **RecordKey** and **State_Value** fields and all fields, including the **RecordKey** and **State_Value** fields, are selected by default.

```
...
  "FileRecords": [
    {
      "#Query": "('@RecordKeys')?$expand=Parties/Name&$filter=PartyName 'A'",
      "SupplementaryDocuments": [
        {
          "#Query": "?$expand=Parties/Name"
        }
      ]
    }
  ]
...

```

5.5 Troubleshooting custom reports and tips

Preview the report or JSON file

Use the **Accept** setting in the Post request Report API to view the resulting file as a JSON file, a Word/Excel document, or a PDF document. This enables you to track and view the faulty report in any of the three of the report stages:

- The Report JSON file
- The Word/Excel document
- The PDF document

Tip: You can also use Fiddler (or other 3rd party Web debugging tools) to create and verify your reports.

Reports with included document contents

If you want the report to include document contents, the report request must contain the following parameter IncludeDocumentContents=true.

See the Report API documentation.

Additionally, the following fields must be specified in the OData queries of the Report JSON file so that the PDF engine can recognize the document and retrieve the fields.

- RecordKey
- State_Value

The date is not displayed in the expected format

Please see the solution [here](#).

6. PDF options

When creating PDF documents you can define various PDF in the report file to further define the contents of the PDF file when generated.

The following PDF options can be used:

- Server side document merging: Merge PDF documents in one PDF document on the WorkZone server. See [Remerging documents](#).
- Custom Headers/Footers: Define your own header or footer globally or for each PDF document and define the alignment and placement of the header or footer as well. See [Headers and footers](#).
- Global Page numbering: Number all pages of a global PDF document or report in the header or footer of the document. See [Global page numbering](#).

6.1 Custom headers and footers

You can specify custom header and footer texts, merging document data with your own text instead of using the default header and footers for the report. The headers and footers can have global or local scope, depending on the documents in question.

Custom headers and footers can be specified as custom text or as placeholders. The placeholders can in their turn also contain custom text and/or document data.

Custom text

Headers and footers can be added to the report as custom text only, with the defined text being generated on each page of the report.

Example: Free text

```
Header: "Confidential - do not distribute!"
```

In this example, each and every page of the report document will contain the header text: *Confidential - do not distribute!*

You can define header and footer alignment, margins as well as style (font size and colors) to further customize your report headers and footers.

Placeholders

Headers and footers can also contain placeholders. Placeholders can be used to contain and display document data retrieved directly from the document itself. Placeholder values can also be defined directly in the report JSON and you can define custom text in the placeholders.

Placeholder custom text can also be combined with document data in the placeholders and a placeholder can contain multiple instances of custom text and document data.

Custom text in placeholders

You can specify custom text within the placeholder itself by entering the custom text in quotation marks within the placeholder brackets. { }. You can combine the placeholder custom text with placeholder tokens, creating labels or descriptive text for your tokens.

Example: Placeholder custom text with the **Title** token.

```
Header: "{"Title of document: " Title}"
```

If the value of the placeholder token **Title** is "Annual Report", the header will be displayed as: *Title of document: Annual Report*.

Document data in placeholders

Tokens are used in the placeholders to display document data. The relevant document data will be populated in the placeholder token when the report is run. The tokens conform to the field codes in Microsoft Word in order to maintain a degree of recognition with the Microsoft Word field codes.

You can combine the tokens with custom text in the place holder and you can specify multiple tokens and custom texts within a single placeholder. Placeholders with multiple custom texts and tokens can be complex and difficult to decipher but might be necessary as you only can define one header or footer per document. You can define multiple placeholders though.

Note: The placeholder tokens are placed without including spaces. You must explicitly define and insert any required spaces within quotation marks.

Including quotation marks in the placeholder custom text

You can include quotation marks in the placeholder custom text by using a backslash before the quotation mark.

Example: Include quotation marks in placeholder text

```
Header: "{\"Placeholder text \"\"Quotation text\"\" More placeholder text}\"
```

Result: *Placeholder text: "Quotation text" More placeholder text*

Placeholder tokens

When you use the **Select** clause in the report JSON, all the properties of the record that is selected are available as placeholder tokens and can be updated with actual values retrieved from OData (e.g. FileKey, PostList, DocLength, etc).

You must include the document contents to the report by setting **IncludeDocumentContents** parameter to **True** in the related POST request to the Reports API. The record's properties and tokens will then updated with actual values on the record's pages inside the report, whereas all the other pages of the report would not be updated.

No Select clause in the report JSON

When there is no explicit **Select** clause in the Report JSON, all fields are considered to be selected and are therefore also available as placeholder tokens, for example if a report query selects the whole record (Record is present in the query but no select clause for specific fields), all the record's properties will be available as placeholder tokens.

Expanded tables in the report JSON

Fields accessed through the Expanded command to the Record table in the report JSON are also available as placeholder tokens for headers and/or footers in the document, including custom fields have been added to the Record (Case), File (document) or contact (Contact) tables.

For an example of an Expand command, see [Create the Report JSON file](#)

Note: Custom fields are created and maintained in the WorkZone Configurator module.

Additional placeholder tokens

The following additional placeholder tokens can be used in the header and footer:

- **{Title}**: The document title – either an original file name or an empty string if not provided.
- **{Date}**: The current date based on defined culture settings. You can use the **AcceptLanguage** request HTTP header to set the culture to be used by the WorkZone PDF to handle the request. For example the date format DD.MM.YYYY or DD.MM.YY is particular to Denmark while MM/DD/YYYY is particular for the United States.
- **{Page}**: The current page number of the document.
- **{Numpages}**: The total amount of pages in the document.
- **{Caption}**: Return the report caption, if a caption has been defined.

The placeholder tokens can be used in any combination, even with additional text.

Headers or footers defined as an HTTP header will be global for all documents. Headers or footers defined in the JSON request body will override any defined global header or footer for the specific document where it is declared.

Example: Multiple tokens in a placeholder

```
Header: "{Page " of " Numpages}"
```

Result: If **Page** is 2 and **Numpages** is 25, the result will be: *Page 2 of 25.*

Empty document data

If the document data for a token in a placeholder is empty, that token will be displayed as an empty value and any defined custom text will be displayed.

However, if all the values of all the tokens in a placeholder are empty, the entire placeholder will not be displayed, even if there is custom text defined in the placeholder.

This enables you to display header or footer text only when related tokens are populated with data on record-related pages and not display header or footer text on other report pages - for example Case-related pages or report cover pages.

Example: Empty tokens

```
Header: "{ "State is " State ", Type is " Type }"
```

In this example, the header result will depend on the values of the tokens: **State** and **Type**.

Token values	State = A	State empty
Type = B	State is A, Type is B	State is "", Type is B
Type empty	State is A, Type is ""	""

Note: When both tokens are empty, the entire placeholder will not be displayed. As long as a single token contains a non-empty value, the placeholder will be displayed.

Places where headers and footers can be defined

Customized headers and footers for PDF documents are specified in the same fashion as specifying custom headers and footers for reports and use the same placeholder tokens.

Custom headers and footers, like nearly all parameters can be defined in the following places:

- In the Database configuration itself.
- In the HTTP headers for the entire request
- In the POST request body root of the request
- In the POST request body local for the request for the specific document
- In the GET URI

The list above starts at a global level, applying the header/footer parameter setting to all documents and then gets more specific to the individual document, depending on where the header/footer parameters are defined.

Locally defined parameter settings will override more globally defined parameter settings, for example a header defined in the GET Uri of the document will override any settings defined in the POST request, HTTP headers and Database configuration.

For more information about PDF headers and footers, see [API documentation](#)

6.1.1 Header and footer alignment and margins

You can define the alignment and as well as set the margins for your custom headers and footers in the document.

Alignment

Use the **HorizontalAlignment** and **Verticalalignment** parameters to define the alignment of the custom headers and footers in the report.

The following alignment settings can be specified for the parameters:

- **VerticalAlignment:**

- Top
- Middle
- Bottom

- **HorizontalAlignment:**

- Left
- Center
- Right

Margins

You can also adjust the placement of the custom header or footer relative to the page edge (the page height and width) by using the **HorizontalMargin** and **VerticalMargin** parameters.

You must specify the number of points (in apostrophes) to adjust the margin as an offset, relative to the page edge. The custom margin uses the header or footer alignment as the starting point.

Note: The horizontal and vertical margin points are defined as 72 points per inch.

Example: Horizontal margin specified for a right-aligned header.

```
HeaderStyle {HorizontalAlignment: 'Right', HorizontalMargin: '10' },
```

The horizontal margin parameter offsets the header by 10 points, relative to the right edge of the page, using the right-aligned header as the starting point.

6.1.2 Examples of custom headers and footers

The following are examples of various headers and footer texts and tokens.

Example: Tokens

```
Header: "{Page} of {Numpages}"
```

Notes: If the Page is 1 and Numpages is 45, the placeholder token will result in the header text "1of45" on the first page of a 45 page document.

If you want to include spaces in order to convert the header more readable, you must specifically define the required spaces like so:

```
Header: "{Page} " of " {Numpages}"
```

The result of this header is "1 of 45".

Example: Footer syntax

```
Footer: "Page {page}" of "{numpages}"
```

Where the Footer is an HTTP header. The footer text will be "Page 1 of 45" on the first page of a 45 page document.

Example: Footer syntax with a single placeholder and multiple placeholder tokens and multiple placeholder custom texts

```
Footer: "{"Page " Page " of " Numpages}"
```

The footer text will be "Page 1 of 45" on the first page of a 45 page document.

Example: Header syntax

```
{Documents: [{Uri: 'file:///d:/AnnualReport.msg', Header: 'This is a local header'}]}
```

Note that the header or footer can be plain text as the "*This is a local header*" text illustrates.

Example: Header and footer style examples

```

Footer: "Page {page} of {numpages}"

FooterStyle: { VerticalAlignment: 'Top', HorizontalAlignment: 'Left' }

{
  "Watermark": "Global Watermark",
  "WatermarkStyle": { "Color": "Black", "Transparency": "33", "Font":
"Arial"},
  "Header": "Global header",
  "Footer": "Global footer",
  "Documents": [
    {
      "Uri": "file:///d:/AnnualReport.msg",
      "Title": "Custom title",
      "Watermark": "Local Watermark",
      "WatermarkStyle": {"Color": "Cyan", "Transparency": "55",
"Algerian"},
      "Header": "This is local header",
      "HeaderStyle": { "HorizontalAlignment": "Center", "Horizo
"10", "Font": "Algerian", "FontSize": "10", "ForegroundColor": "Blue",
"BackgroundColor": "Transparent",
      "Opacity": "0.75",
      "HorizontalScaling": "1"
    },
    "Footer": "This is local footer.",
    "FooterStyle": {
      "Bold": false,
      "Font": "Arial"
    }
  ]
}

```

In this example, there is a global header and footer each being overwritten by a local, document header and footer defined later in the document section.

The document header is centered, with a 10 pixel offset margin and uses the Algerian font with a font size of 10 and a Blue font color.

The document footer style is more subdued, containing only the Arial font.

Example: Complex header with multiple tokens, placeholder custom text and quotation marks within the text

```

Header: "Confidential! {"Print \"date\" is:" date ". Page: " page} of
{Numpages}"

```

In this example, the **Date** is 05/06-2018, the **Page** is 3 and the **Numpages** is 23, the header text would look like this:

Confidential! Print "date" is 05.06.2018. Page: 3 of 23

Note the date formatting will depend on your local date settings.

6.2 Watermarks in reports

You can add your own custom texts as watermarks in reports in the Report JSON file.

Watermarks can be defined either globally for all documents in the request or at the document level - for each individual document.

Note: A watermark defined at the document-level will take precedence over any globally defined watermarks.

The watermark custom text is defined in the Watermark parameter while the watermark styling is defined using the **WatermarkStyle** parameter.

Watermark parameter

When you define the watermark text, you must include the text in quotation marks like this:

```
Watermark: "Draft Version"
```

WatermarkStyle parameter

Watermark text can be styled using the following style options:

- **Color:** The color of the watermark text. Enter any valid html string format such as standard color name, hex value, and RGB colors. For example: Standard color name: Red or Hex value: #FF0000 or RGB: 255,0,0
- **Transparency:** The transparency level of the watermark text. Most watermark text is faintly transparent to avoid overshadowing the main document text. Enter a value between 0 (not transparent) and 100 (completely transparent)
- **Font:** Enter a font name. The font must be accessible on the machine running the report. You do not need to specify font size as the watermark will automatically be re-sized to fit the page.

Example: Watermark and WatermarkStyle

```

{
  "Watermark": "Global Watermark",
  "WatermarkStyle": { "Color": "Black", "Transparency": "33", "Font":
"Verdana"},
  "Documents": [
    {
      "Uri": "file:///d:/AnnualReport.msg",
      "Title": "Custom title",
      "Watermark": "Local Watermark",
      "WatermarkStyle": { "Color": "Cyan", "Transparency": "40", "Font":
"Arial"}
    }
  ]
}

```

In this example, the global watermark "Global Watermark" is superseded by the "local Watermark" watermark on the local document. Note the formatting is also changed for the watermark of the local document.

6.3 Global page numbering

When creating a global PDF document based on one or more documents, you can number all pages in the global PDF document or report and display the page numbering in either the header or the footer of the PDF documents or of the report.

Microsoft Word field codes (`{page}` and `{numpage}`) are used as placeholder tokens for the page numbers.

Example: Examples of page numbering:

- `{page}` of `{numpage}` (da. `{page}` af `{numpage}`)
- Page `{page}` (da. Side `{page}`)
- `{page}`

Where `{page}` is the page number and `{numpage}` is the total number of pages in the global PDF document.

Tip: Do not include page numbering functionality on the individual report templates, as this will result in two different page numberings on the document -one from the report itself and one from the PDF version of the report. If the report PDF also is part of a larger, aggregated

PDF document, the two page numbering systems may display different values, formats and placement.

For more information about global page numbering of the global PDF document, see [API documentation](#).

6.4 Remerging documents

If you convert a Word document to PDF, you can update values of its content controls just after the conversion and later on whenever you need them to be updated. To do this, first you need to prepare the document for remerging and then remerge it. Note that this is not the global functionality of WorkZone PDF Engine, so you have to define the specific information in the POST requests to WorkZone PDF Engine.

Tip: The POST request can be sent in the JSON or Multipart Streams format.

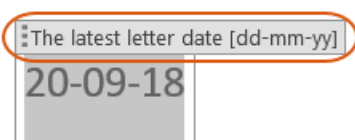
Step 1 — Preparation for remerging

To prepare a Word document for remerging, send the POST request to WorkZone PDF Engine with the following information:

- Word document for which you want to use the remerging functionality
- Titles of content controls whose values you want to update

See example of content control's title

Content control title in a regular mode:



Content control title in a design mode:

The latest letter date [dd-mm-yy] 20-09-18

Note: You have to specify the exact title in the requests.

In these examples, the exact title is: `The latest letter date [dd-mm-yy]`.

When converting multiple documents to one PDF, documents that are supposed to be remerged should be placed first in the request body and the documents that are not supposed to be remerged should be placed last.

Once this information is sent, it is saved in the resulting PDF document's custom properties.

Step 2 — Remerging

To replace old values with the new ones, send another POST request with the following information:

- PDF document for which you want to use the remerging functionality
- Titles and new values of content controls

The WorkZone PDF service updates the values.

For more information about remerging documents, see [WorkZone PDF API documentation](#).

Server side document merging

The PDF engine contains an API which enables you to perform document merging directly on the server using content controls and OData references.

Prerequisite: WorkZone for Office must be installed in order to use the server-side document merging.

7. Terms and conditions

Intellectual property rights

This document is the property of KMD. The data contained herein, in whole or in part, may not be duplicated, used or disclosed outside the recipient for any purpose other than to conduct business and technical evaluation provided that this is approved by KMD according to the agreement between KMD and the recipient. This restriction does not limit the recipient's right to use information contained in the data if it is obtained from another source without restriction set out in the agreement between KMD and the recipient or by law.

Disclaimer

This document is intended for informational purposes only. Any information herein is believed to be reliable. However, KMD assumes no responsibility for the accuracy of the information. KMD reserves the right to change the document and the products described without notice. KMD and the authors disclaim any and all liabilities.

Copyright © KMD A/S 2021. All rights reserved.