Manual for Developers

officeatwork DCML API

Version 4.9

This guide has been created using officeatwork Advanced.
7. May 2020

Table of Contents

About this guide

For whom is the guide intended

This book has been written for software developers that want to implement an XML interface to officeatwork.

What is covered in this guide

This manual illustrates the process of integrating an officeatwork interface in your application. It explains all parameters available to the developer. It also provides a best praxis architecture on how to implement the officeatwork interface.

Knowledge required

You should be familiar with the general use of computers, especially with the XML notation and Xpath expressions. Programming knowledge is of advantage.

Typographic conventions

Before reading this guide, you should be familiar with the typographic conventions used.

The following graphic descriptions highlight sections of text with particular significance.

Formatting Convention	Type of Information
Triangle ➢	Step-by-step procedure. You can follow these instructions to perform a specific task.
Bold Typeface	Objects needed for selection, such as menus, buttons, items in a list or table headers.
CAPITAL LETTERS	Key legends on the keyboard. For example `SHIFT`, `CTRL` or `ALT`.
KEY+KEY	Key combinations which must be pressed at the same time are marked with +. Examples: `CTRL+P` or `ALT+F4`.

C H A P T E R 1 D

Introduction

There are many reasons why business applications want to integrate with Microsoft Office. Here are a few reasons:

- Re-use of existing Templates
- Re-use of existing Corporate Design
- Re-use of user skills for editing documents

In order to better understand the challenges you face when integrating Microsoft Office into business applications, we will analyse a few of the most common concepts.

After that we will have a look at the officeatwork approach of bringing together your business application with Microsoft Office.

Microsoft Office integration concepts

Microsoft Office and business applications do not always concur. Basically all applications need specific and specially created templates, which in turn generate many different copied templates. Additionally, it is seldom the case that Microsoft Office data can be accessed from business applications such as ERP, CRM, DMS, etc.

These discrepancies mean that the necessary information needs to be recorded again and again. That is an absolute waste of time and also creates opportunities for errors.

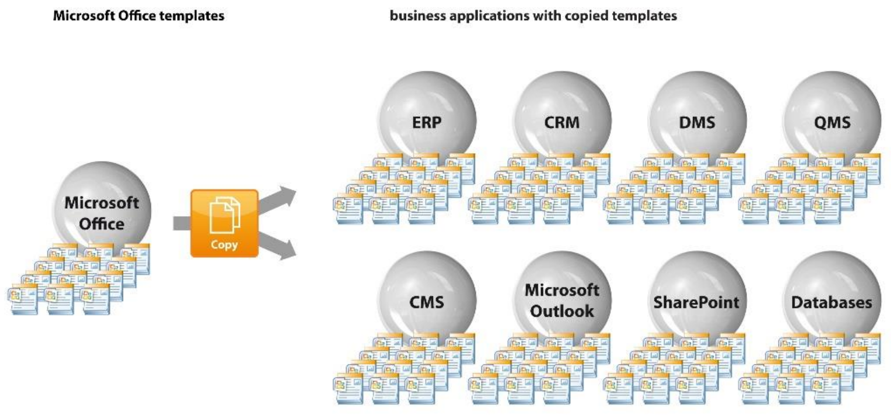

Figure 1: Typical Microsoft Office integration architecture

Integration via Mail-Merge

Typically a static Office template like for instance a «Letter.dot» file is imported into the business application. The template is then modified to include mail-merge fields as placeholders for the business application data.

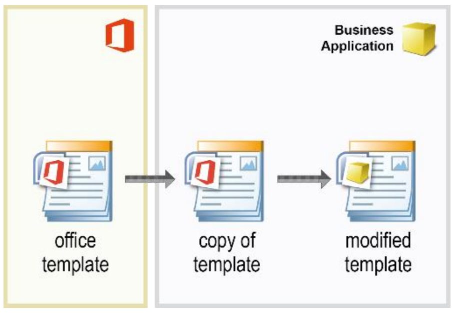

Figure 2: creating a business application specific Office template

The business application directly manipulates that document by controlling the mail-merge function of the Office application, by using VBA (Visual Basic for Applications or any other supported programming language). In this process it writes the business data to a mail-merge compatible file and then opens the template. This is when the user returns to finish the document using the mail-merge functionality.

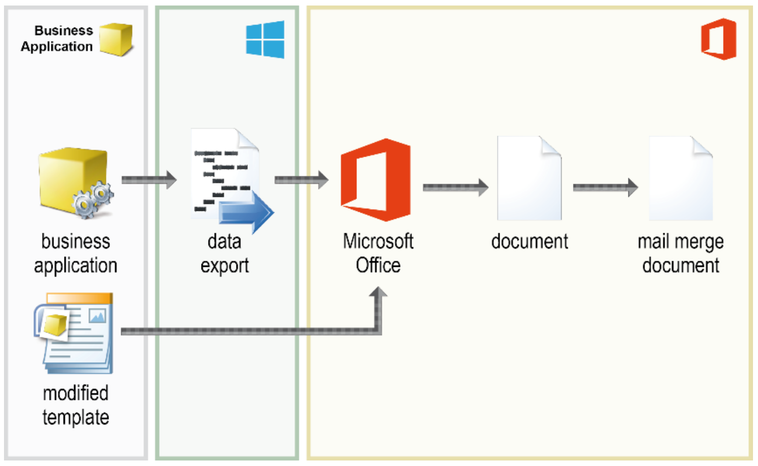

Figure 3: business application Office integration concept via mail-merge function

Pros:

- Using existing functions reduces effort of integration.

Cons:

- Duplication of already existing templates had to additional adjustments in case of design or data changes (e.g. telephone)
- Direct dependency on the Office application version and its offered functionality
- In-depth knowledge about the Office application required
- Testing for each new Officeversion is necessary
- Intensive maintenance
- The whole integration cycle needs to be done for each business application separately

Integration via Bookmarks, DDE/OLE and Co.

Typically a static Office template like for instance a «Letter.dot» file is imported into the business application. The template then gets modified to include bookmarks and other placeholders for the business application data.

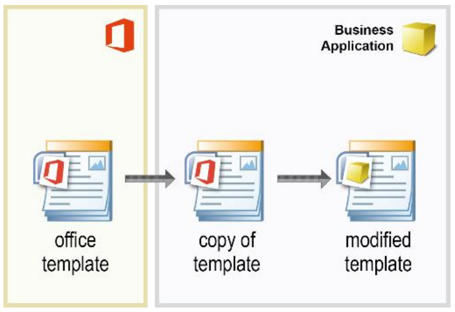

Figure 4: creating a business application specific Office template

The business application then directly manipulates that document by using VBA (Visual Basic for Applications or any other supported programming language). In this process it first generates a new document from the template and then writes the business data directly into it. It depends on the depth of the integration whether the document is presented to the user or processed for output directly by the application.

DDE (Dynamic Data Exchange) is a technology for communication between multiple applications under Microsoft Windows. A common use of DDE was for custom-developed applications to control off-the-shelf software. For example, a custom in-house application might use DDE to open a Microsoft Excel spreadsheet and fill it with data, by opening a DDE conversation with Excel and sending it DDE commands. Today, however, one could also use the Excel object model with OLE Automation (part of COM).

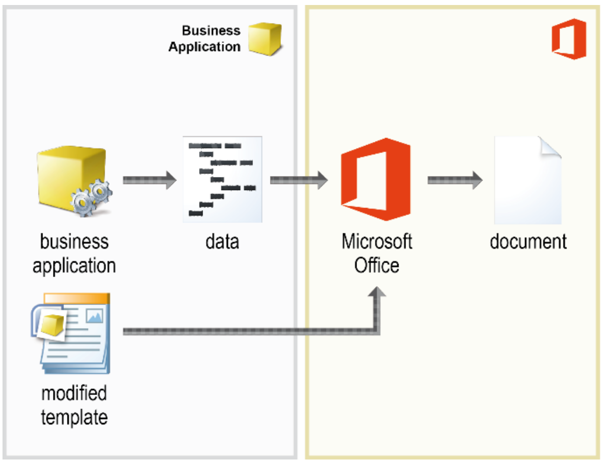

Figure 5: business application Office integration concept via bookmarks, DDE/OLE and Co.

Pros:

- Highly flexible as the full object-model of the Office applications his available to manipulate.
- Standardised interface between many different applications with the potential of reusing the knowledge gained in such an integration. (DDE/OLE)

Cons:

- Direct dependency on the Office application version and its offered functionality
- In-depth knowledge about the Office application required
- Testing for each new Officeversion is necessary
- Duplication of already existing templates
- Intensive maintenance
- The whole integration cycle needs to be done for each business application separately
- Enhancement in functionality often requires re-programming of the interface
- Limited to reduced function-set offered by server applications (DDE/OLE)
- All involved applications need to be running for this integration concept to work. (DDE/OLE)

officeatwork integration concept

officeatwork is a flexible link between Microsoft Office and your business applications. Personal and enterprise information can be automated and directly used in your Office documents. Your Office templates can be directly linked to your business applications like ERP, CRM, QMS, SharePoint, DMS, etc by using the officeatwork XML API interface. It is no longer necessary to duplicate templates.

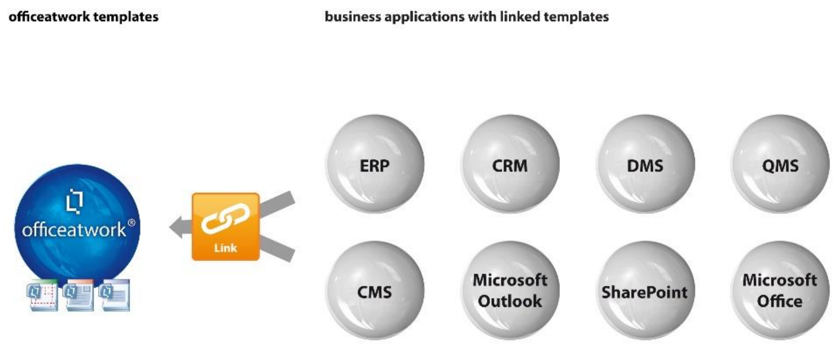

Figure 6: officeatwork integration concept

The main benefit of the officeatwork integration architecture is the fact that no longer copies of templates need to be imported into your business application. Instead your business application can link to existing Office templates using standard XML language. In this process the business application compiles its requirements and data into an XML string and passes that onto officeatwork. officeatwork will then process the XML automatically. Your business application does not need to understand how to create a document in Microsoft Office.

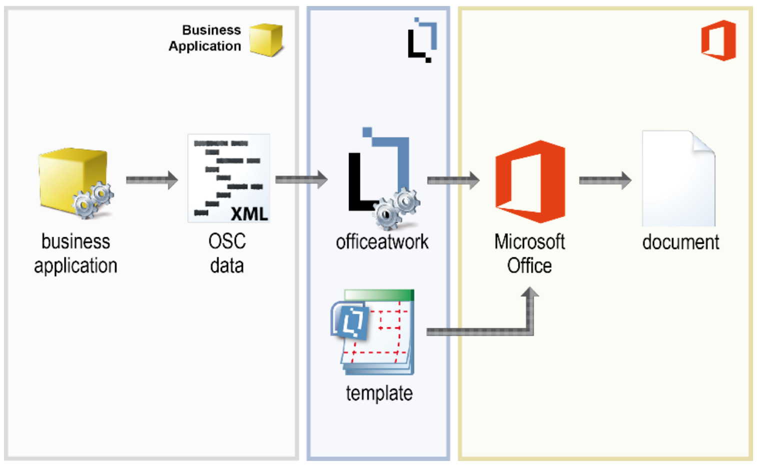

Figure 7: officeatwork business application integration concept

Pros:

- No duplication of already existing templates which therefore reduces maintenance when changing design, Logos, data (e.g. telephone)
- No direct dependency on the Office application version and its offered functionality.

- No in-depth knowledge about the Office application required.
- No testing for each new Office version is necessary.
- Maintenance friendly – no reprogramming for new templates or data attributes necessary.
- Business application can share same templates, no separate integration necessary.
- No re-programming of the interface to enhance functionality.

Cons:

- Limited to functionality offered by officeatwork.

CHAPTER 2

Overview

Benefit

With the «officeatwork DCML Engine Enterprise» you have an application to create documents in a fast and easy way without using an office application client side. You can reuse all items of an officeatwork repository like templates and contents.

With the «officeatwork EDC Server» you have a server application to create many documents in a fast and easy way without using an office application server side. You can reuse all items of an officeatwork repository like templates and contents.

Systems

The following systems are provided for creating documents:

- Enterprise Document Creation system:
- officeatwork EDC Server
- officeatwork DCML Engine Enterprise
- officeatwork repository

C H A P T E R 3

officeatwork Integration Architecture for Business Applications

Overview

officeatwork offers two directions of integrating with your business application. The first is from officeatwork to your business application.

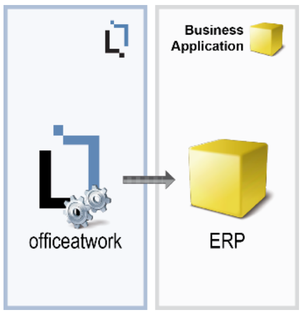

Figure 8: officeatwork interacting with a business application

Here the starting point is the officeatwork Client Suite. From within Microsoft Office officeatwork fetches data from your business application and retrieves them to your Office document. This option is mostly used to fetch address-information from for example your ERP or CRM system as well as user information from for example your Active Directory.

The first option is only supported by the officeatwork Client Suite.

The other direction is from your business application to officeatwork.

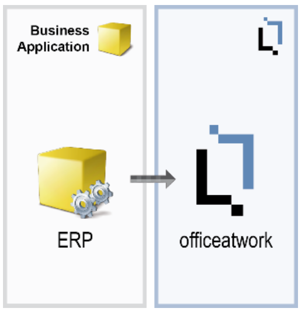

Figure 9: Business application interacting with officeatwork

Here the starting point is your business application. Your business application can be extended so that it can create documents in Microsoft Office using officeatwork functionality. It does this by sending standardised XML formatted instructions to officeatwork. A common example for this option is for instance the creation of a quote based on the information held within your ERP system.

This second option is where the officeatwork API comes into action. It is designed to enable your business applications to communicate in a standardised and flexible way with officeatwork via XML. XML is a widely accepted technology and recommended to be used to communicate between systems.

This book only covers the API integration option. All variations of the first integration option are explained in a separate manual.

Interaction concepts

officeatwork offers different ways for you to create documents via the officeatwork XML API. The parameters of the various ways are the same, just the way officeatwork is involved differs. The three available ways are:

- Passing the parameters to the officeatwork EDC Server using a REST Web-Service.
- Saving the parameters to a file and then sending the OS an Open command to execute that file on a computer having the officeatwork Client Suite installed.
- Passing the parameters to a method in an officeatwork ActiveX component installed with the officeatwork Client Suite.

officeatwork EDC Server

officeatwork XML API can be triggered by passing the parameters to the officeatwork EDC Server using a REST Web-Service. The technical requirements and the Web-Service interface are documented in a separate maual.

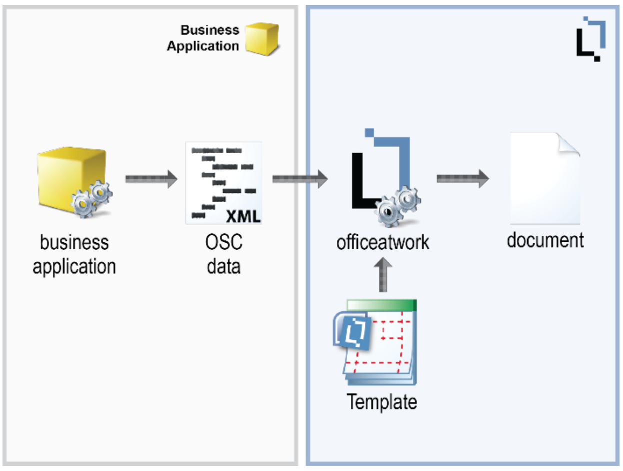

Figure 10: business application Office integration concept via officeatwork EDC Server

Open File with officeatwork Client Suite

officeatwork Client XML API can be triggered by opening a file with the extension *.OSC (officeatwork shortcut file) on a computer having the officeatwork Client Suite installed. The content of that file must be in XML format and must follow the XML schemes of the officeatwork XML API. The following example shows a simple example of such an *.OSC file.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Parameters>
   <CreateDocument>
      <TemplateID>letter</TemplateID>
   </CreateDocument>
</Parameters>
```

The structure of the XML API will be covered in a later chapter. At this point it is important to know that a simple XML formatted file will allow you to interact with officeatwork. Simply double-click your OSC file and officeatwork will open the file and execute the XML formatted instructions.

Please note that when you are working with a web server that serves html pages containing links to OSC files, you must add a MIME type for the OSC files on that web server. In addition to this setting you can replace the first line (encoding line) in the OSC file with the corresponding officeatwork document function. Otherwise clicking on links pointing to OSC files will open the OSC file in your web browser instead of executing the officeatwork XML API. So make sure your web servers have a MIME type **application/osc** for the extension **.osc** defined.

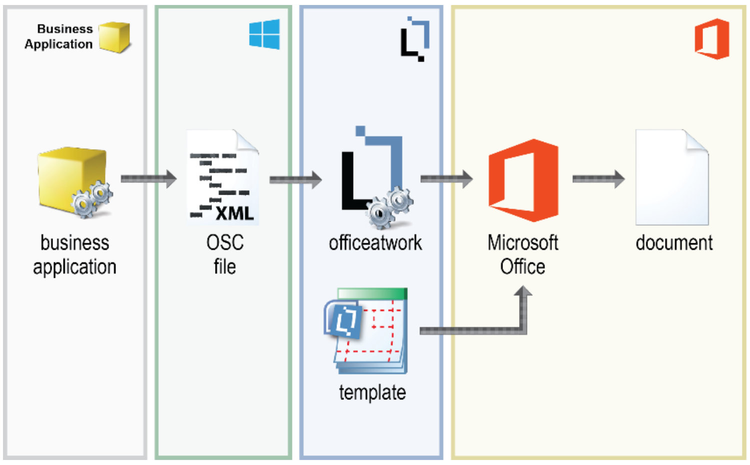

Figure 11: business application Office integration concept via officeatwork shortcut file

Open File with officeatwork DCML Engine

officeatwork DCML Engine XML API can be triggered by opening a file with the extension *.DCML (officeatwork DCML file) on a computer having the officeatwork DCML Engine installed. The content of that file must be in XML format and must follow the XML schemes of the officeatwork DCML Engine XML API. The following example shows a simple example of such an *.DCML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<DCML Version="1">

   <Control>
   </Control>

   <Instruction>
      <CreateDocument>
         <TemplateID>letter</TemplateID>
         <Language>de-ch>/Language>
      </CreateDocument>
   </Instruction>

   <Data>
   </Data>

</DCML>
```

At this point it is important to know that a simple XML formatted file will allow you to interact with officeatwork. Simply double-click your DCML file and officeatwork will create the file by the XML formatted instructions in background.

Basics

General

The XML parameter is, as the name already indicates, a parameter written in XML format. Therefore, all rules and regulations on how to present information in XML format apply.

Just as a reminder, we have listed a few characters that are used to structure the information in XML format and therefore are not allowed to be used elsewhere. If you want to use one of these characters for any purpose other than structuring your XML (for representing your business data for example), you must replace those characters with the equivalent replacement as listed below:

Reserved Character	equivalent replacement
&	&amp;
'	&apos;
>	&gt;
<	&lt;
"	&quot;

Encoding

If you plan to include special characters like **ä**, **é**, etc. within your XML parameter, you must include an encoding tag at the beginning of your XML file. This will make sure your special characters are correctly interpreted.

Example encoding tag

```
<?xml version="1.0" encoding="UTF-8"?>
```

Recommended integration architecture

Based on many different architectures, we observed that we clearly favour one specific architecture for many reasons. This architecture uses a mixture of the two available interaction concepts.

Overview

The business application uses an officeatwork DCML template file with the file extension *.DCMLT file as a base to generate a new DCML file. During this process it replaces placeholders within the DCMLT representing business application values with the proper values. The DCMLT may also contain specific business application instructions like loops or counters. This business values are all caled in the <Data> element in the DCML file. The resulting DCML file from the processed DCMLT is then processed by officeatwork.

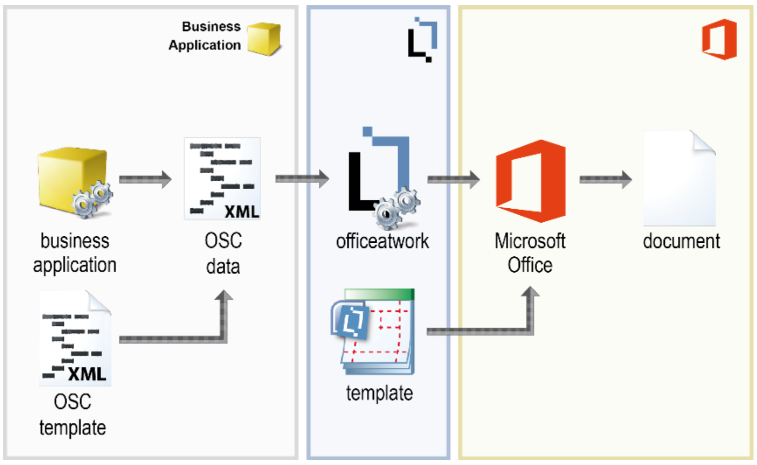

Figure 12: recommended architecture for creating an Office document out of your business application via officeatwork

This architecture has many advantages:

- Limited programming necessary – the business application only needs to be taught how to process the DCML template file. This can be implemented so that new data items available in your business application will not require the reprogramming of the interface to officeatwork. Neither does new officeatwork functionality require a reprogramming of the interface.
- Optimal job sharing – by separating the interface into different parts (Process, Definitions, Design) the development cycle and complexity is kept to a minimum.
- Easy testing – as the final result coming from your business application is an DCML file, you do not need to wait until the programming is finished to test the interface. You can just create example DCML files and double-click them to test your definitions and design.
- Simple Debugging – as the result coming from the business application is a file, it can easily be analysed. You can easily isolate individual parts of the file to find out any mistakes in the document creation process.

Examples

Example

This DCML template file creates an Invoice summary document and saves it using an officeatwork output method.
Before this DCML template will be processed by officeatwork the business application will replace all the formulas {{Formula(*)}} by the xml values.

```
<?xml version="1.0" encoding="UTF-8"?>
<DCML Version="1">

  <Instruction>
    <CreateDocument>

      <TemplateId>Invoice</TemplateId>
      <Language>[[XpathValue('//Account/Contact/LanguageId')]]</Language>

      <MasterPropertySets>
        <MasterPropertySet>
          <Id>First</Id>
          <MasterProperties>
            <MasterProperty>
              <Id>CustomField</Id>
                <Fields>
                  <Field>
                    <Name>DocumentType</Name>
                    <Value>[[XpathValue('//Translations/Invoice')]]</Value>
                  </Field>
                  <Field>
                    <Name>YourReference</Name>
                    <Value>[[XpathValue('//Account/ContactName')]]</Value>
                  </Field>
                  <Field>
                    <Name>Account</Name>
                    <Value>her als [[XpathValue('//Account/Name')]]</Value>
                  </Field>
                  <Field>
                    <Name>Project</Name>
                    <Value>[[XpathValue('//Project/Name')]]</Value>
                  </Field>
                </Fields>
            </MasterProperty>
          </MasterProperties>
        </MasterPropertySet>
      </MasterPropertySets>

      <Bookmarks>
        <Bookmark>
          <Name>Subject</Name>
          <Text>[[XpathValue('//Invoice/Summary/InvoiceNumber')]]</Text>
        </Bookmark>
      </Bookmarks>

      <Contents>
        <Content>
          <Id>Invoice Summary Title</Id>
        </Content>

       [[XpathLoop('InvoicePositions', '//Invoice/Positions/Position', '
        <Content>
          <Id>Invoice Summary Item</Id>
          <ContentControls>
            <PlainText>
              <Tag>incident_type</Tag>
              <Text>' & XpathLoopValue('InvoicePositions', 'type') & '</Text>
            </PlainText>
            <PlainText>
              <Tag>total</Tag>
              <Text>' & XpathLoopValue('InvoicePositions', 'total') & '</Text>
            </PlainText>
            <PlainText>
              <Tag>vat</Tag>
              <Text>' & XpathLoopValue('InvoicePositions', 'vat') & '</Text>
            </PlainText>
          </ContentControls>
        </Content>')]]

       [[XpathLoop('InvoiceSummary', '//Invoice/Summary/Position', '
        <Content>
          <Id>Invoice Summary Total</Id>
          <ContentControls>
            <PlainText>
```

```
              <Tag>total_incl_vat</Tag>
              <Text>' & XpathLoopValue('InvoiceSummary', 'vat_incl') & '</Text>
            </PlainText>
            <PlainText>
              <Tag>total_vat_relevant</Tag>
              <Text>' & XpathLoopValue('InvoiceSummary', 'vat_relevant') & '</Text>
            </PlainText>
            <PlainText>
              <Tag>total_vat_irrel</Tag>
              <Text>' & XpathLoopValue('InvoiceSummary', 'vat_irrel') & '</Text>
            </PlainText>
            <PlainText>
              <Tag>vat</Tag>
              <Text>' & XpathLoopValue('InvoiceSummary', 'vat') & '</Text>
            </PlainText>
        </Content>')]]

        <Content>
          <Id>Invoice Accounting Details</Id>
          <ContentControls>
            <PlainText>
              <Tag>Reference</Tag>
              <Text>[[XpathValue('//Account/Reference/Name')]]</Text>
            </PlainText>
          </ContentControls>
        </Content>
      </Contents>

      <Outputs>
        <Output>
          <View>Original</View>
          <Targets>
            <Target>
              <FileName>Invoice [[XpathValue('//Account/Name')]].docx</FileName>
              <FileFormat>DOCX</FileFormat>
              <Delivery>
                <Save>
                  <Path>%Desktop%</Path>
                </Save>
              </Delivery>
            </Target>
          </Targets>
        </Output>
      </Outputs>

    </CreateDocument>
  </Instruction>

  <Data>
    <Translations>
      {{GetDataFromBusinessApplication('Translations')}}
    </Translations>
    <Account>
      {{GetDataFromBusinessApplication('Account')}}
    </Account>
    <Project>
      {{GetDataFromBusinessApplication('Project')}}
    </Project>
    <Invoice>
      <Summary>
        {{GetDataFromBusinessApplication('Invoice Summary')}}
      </Summary>
      <Positions>
        {{GetDataFromBusinessApplication('Invoice Positions')}}
      </Positions>
    </Invoice>
    <Settings>
      {{GetDataFromBusinessApplication('Translations')}}
    </Settings>
  </Data>

</DCML>
```

Please note that we recommend using {{ and }} as identifier for placeholders for business applications. Make sure you have removed all placeholders in the final DCML file that gets passed on to officeatwork.

CHAPTER 4

DCML structure

The following chapter will explain the structure and the function of the elements in a DCML file.

DCML Main Elements

This chapter describes the main elements in the DCML file structure.

Example

The following example shows the root element and the main elements of the DCML structure.

```
<DCML>

       <Control>
       </Control>

       <Instruction>
       </Instruction>

       <Data>
       </Data>

</DCML>
```

Root Element <DCML>

The root element <DCML> is needed to identify a XML file as a DCML file. The DCML root element is mandatory.

The element contains the optional attribute version. The version declares the used DCML language version.

Main Element <Control>

The main element <Control> contains all elements to control the document creation process.
This element can contain DCML formulas (see chapter “DCML Formulas”) that references data from the main element <Data>.

This main element can be non valid XML.

Example

The following example generates a processing report. The report name references the contact name defined in the <Data> main element.

```
<DCML Version="1">

       <Control>
          <ProcessingReport>
             <FileName>Processing [[XpathValue('//Contact/Name')]].report</FileName>
          </ProcessingReport>
       </Control>

       <Instruction>
          <CreateDocument>
             <TemplateId>Letter</TemplateId>
             <Language>[[XpathValue('//Contact/Language')]]</Language>
             <Outputs>
                <Output>
                   <View>Original</View>
                   <Targets>
                      <Target>
                         <FileName>Letter for [[XpathValue('//Contact/Name')]].docx</FileName>
                         <FileFormat>docx</FileFormat>
                         <Delivery>
                            <Save>
                               <Path>%DESKTOP%</Path>
                            </Save>
                         </Delivery>
                      </Target>
                   </Targets>
                </Output>
             </Outputs>
          </CreateDocument>
       </Instruction>

       <Data>
          <Contact>
             <Name>Peter Miller</Name>
             <Language>1033</Language>
          </Contact>
       </Data>

</DCML>
```

Main Element <Instruction>

The main element <Instruction> contains all elements to generate or edit documents.
This element can contain DCML formulas (see chapter “DCML Formulas”) that references data from the main element <Data>.

This main element can be non valid XML.

Example

The following example creates a document based on the template “Letter” in the contact language defined in the <Data> main element. The document file name will include the contact name defined in the <Data> main element.

```
<DCML Version="1">

       <Control>
          <ProcessingReport>
             <FileName>Processing [[XpathValue('//Contact/Name')]].report</FileName>
          </ProcessingReport>
       </Control>

       <Instruction>
          <CreateDocument>
             <TemplateId>Letter</TemplateId>
             <Language>[[XpathValue('//Contact/Language')]]</Language>
             <Outputs>
                <Output>
                   <View>Original</View>
                   <Targets>
                      <Target>
                         <FileName>Letter for [[XpathValue('//Contact/Name')]].docx</FileName>
                         <FileFormat>docx</FileFormat>
                         <Delivery>
                            <Save>
                               <Path>%DESKTOP%</Path>
                            </Save>
                         </Delivery>
                      </Target>
                   </Targets>
                </Output>
             </Outputs>
          </CreateDocument>
       </Instruction>

       <Data>
          <Contact>
             <Name>Peter Miller</Name>
             <Language>1033</Language>
          </Contact>
       </Data>

</DCML>
```

Main Element <Data>

The main element <Data> contains all data elements that are needed during the document creation process. They can be referenced from the other main elements with DCML formulas (see chapter “DCML Formulas”).

The <Data> element has to be a valid XML without namespaces.

Example

The following example creates a document based on the template “Letter” with the language defined in the <Language> element in the <Data>. The document file name will include the contact name defined in the <Data> main element.

```
<DCML Version="1">

       <Control>
          <ProcessingReport>
             <FileName>Processing [[XpathValue('//Contact/Name')]].report</FileName>
          </ProcessingReport>
       </Control>

       <Instruction>
          <CreateDocument>
             <TemplateId>Letter</TemplateId>
             <Language>[[XpathValue('//Contact/Language')]]</Language>
             <Outputs>
                <Output>
                   <View>Original</View>
                   <Targets>
                      <Target>
                         <FileName>Letter for [[XpathValue('//Contact/Name')]].docx</FileName>
                         <FileFormat>docx</FileFormat>
                         <Delivery>
                            <Save>
                               <Path>%DESKTOP%</Path>
                            </Save>
                         </Delivery>
                      </Target>
                   </Targets>
                </Output>
             </Outputs>
          </CreateDocument>
       </Instruction>

       <Data>
          <Contact>
             <Name>Peter Miller</Name>
             <Language>1033</Language>
          </Contact>
       </Data>

</DCML>
```

Elements of <Control>

This chapter describes all the elements that can be used in the main element <Control>.

Processing Report

Allows you to generate a processing report.

Syntax

```
<ProcessingReport>
    <Filename></Filename>
    <Path></Path>
</ProcessingReport>
```

Content

The `ProcessingReport` element can contain the following sub-elements

Name	Description
Filename	Optional. This element defines the name of the generated processing report. Default: “Processing.report”
Path	Optional. This element defines the path of the generated processing report.

The file path is ignored by the EDC Server, because there you can only download the report.

Example 1

The following example generates a default processing report on the server with the default file name processing.report.

```
…
<Control>
   <ProcessingReport/>
</Control>
…
```

Example 2

The following example generates a processing report on the server with a defined file name.

```
…
<Control>
   <ProcessingReport>
      <Filename>Processing Report Letter.xml</Filename>
   </ProcessingReport>
</Control>
…
```

Example 3

The following example generates a processing report with the DCML Engine with a defined file name and path.

```
…
<Control>
   <ProcessingReport>
      <Filename>Processing Report Letter.xml</Filename>
      <Path>C:\officeatwork\Reports</Path>
   </ProcessingReport>
</Control>
…
```

Example 4

The following example generates a processing report with the DCML Engine. The report name is variable and contains the contact name defined in the <Data> main element.

```
…
<Control>
   <ProcessingReport>
      <FileName>Processing Report Letter for [[XpathValue('//Contact/Name')]].xml</FileName>
      <Path>%Desktop%\Reports</Path>
   </ProcessingReport>
</Control>
…
```

Elements of <Instruction>

The XML <Instruction> element conforms to the following XML structure/convention.

Root Elements

Within the <Instruction> element, main elements can be used to create or edit files. Each root element can contain instruction elements. An alphabetically list of the root elements with their explanation can be found later in this manual. The following table shows an overview of these root elements and the corresponding instruction elements. The availability is shown with one of the following two bullets:

● The functionality of these instruction elements are identically in all root elements these are available in.

○ The functionality of these instruction elements slightly differ in the various root elements. See root element descriptions for more details.

Environment ⇨	officeatwork EDC Server		officeatwork DCML Engine Standard		officeatwork DCML Engine Enterprise	
⇩ Instruction Elements / Root Elements ⇨	CreateDocument	EditDocument	CreateDocument	EditDocument	CreateDocument	EditDocument
`TemplateId`	●		●		●	
`DocumentId`		○		○		○
`Language`	●	●			●	●
`MasterPropertySets`	●	●			●	●
`MasterPropertySet`	●	●			●	●
`Id`	●	●			●	●
`Active`	●	●			●	●
`SelectedForOutput`	●	●			●	●
`Profile`					●	●
`Default`					●	●
`MasterProperties`	●	●			●	●
`MasterProperty`	●	●			●	●
`Id`	●	●			●	●
`Where`	●	●			●	●
`Is`	●	●			●	●
`Fields`	●	●			●	●
`Field`	●	●			●	●
`Name`	●	●			●	●
`Value`	●	●			●	●
`Bookmarks`	●	●	●	●	●	●

Environment ⇨	officeatwork EDC Server		officeatwork DCML Engine Standard		officeatwork DCML Engine Enterprise	
⇩ Instruction Elements Root Elements ⇨	CreateDocument	EditDocument	CreateDocument	EditDocument	CreateDocument	EditDocument
`Bookmark`	●	●	●	●	●	●
`Name`	●	●	●	●	●	●
`Text`	●	●	●	●	●	●
`BuiltInDocumentProperties`	●	●	●	●	●	●
`BuiltInDocumentProperty`	●	●	●	●	●	●
`Name`	●	●	●	●	●	●
`Text`	●	●	●	●	●	●
`CustomDocumentProperties`	●	●	●	●	●	●
`CustomDocumentProperty`	●	●	●	●	●	●
`Name`	●	●	●	●	●	●
`Text`	●	●	●	●	●	●
`CustomXmlParts`	●	●			●	●
`CustomXmlPart`	●	●			●	●
`XPath`	●	●			●	●
`Namespace`	●	●			●	●
`Value`	●	●			●	●
`DocumentVariables`	●	●	●	●	●	●
`DocumentVariable`	●	●	●	●	●	●
`Name`	●	●	●	●	●	●
`Text`	●	●	●	●	●	●
`ContentControls`	●	●	●	●	●	●
`PlainText`	●	●	●	●	●	●
`Tag`	●	●	●	●	●	●
`Text`	●	●	●	●	●	●
`Contents`	●	●	●	●	●	●
`Content`	●	●	●	●	●	●
`Id`	●	●	●	●	●	●
`Language`	●	●	●	●	●	●
`Bookmark`	●	●	●	●	●	●
`ContentControls`	●	●	●	●	●	●
`PlainText`	●	●	●	●	●	●
`Tag`	●	●	●	●	●	●
`Text`	●	●	●	●	●	●
`Picture`	●	●			●	●
`Tag`	●	●			●	●
`Source`	●	●			●	●
`BinaryData`	●	●			●	●
`Size`	●	●			●	●
`Outputs`	●	●	●	●	●	●
`Output`	●	●	●	●	●	●
`View`	●	●			●	●
`Targets`	●	●	●	●	●	●

Environment ⇨	officeatwork EDC Server		officeatwork DCML Engine Standard		officeatwork DCML Engine Enterprise	
⇩ Instruction Elements Root Elements ⇨	CreateDocument	EditDocument	CreateDocument	EditDocument	CreateDocument	EditDocument
`Target`	●	●	●	●	●	●
`  FileName`	●	●	●	●	●	●
`  FileFormat`	●	●	○	○	●	●
`  FileFormatOptions`	●	●			●	●
`  Delivery`	●	●	●	●	●	●
`    Save`	●	●	●	●	●	●
`      Path`	●	●	●	●	●	●
`      Open`			●	●	●	●
`    Download`	●	●				

CreateDocument

Allows you to create a new document. All its sub-elements describe how to create the document. To create multiple documents, add multiple `CreateDocument` root elements.

Syntax

```
<CreateDocument>
</CreateDocument>
```

Example

This example will create a new document based on the Letter template and sets the text for the bookmark subject to «This is the subject text».

```
<?xml version="1.0" encoding="UTF-8"?>
<DCML Version="1">
   <Instruction>
      <CreateDocument>
         <TemplateId>Letter</TemplateId>
         <Bookmarks>
            <Bookmark>
               <Name>Subject</Name>
               <Text>This is the subject text</Text>
            </Bookmark>
         </Bookmarks>
      </CreateDocument>
   </Instruction>
</DCML>
```

EditDocument

Allows you to edit an existing document. All its sub-elements describe how to edit the document. To edit multiple documents, add multiple `EditDocument` root elements.

Syntax

```
<EditDocument>
</EditDocument>
```

Example

This example will open an existing document and change the content of the subject bookmark to «This is my new subject».

```
<?xml version="1.0" encoding="UTF-8"?>
<DCML Version="1">
   <Instruction>
      <EditDocument>
         <DocumentId>%Documents%\ExampleLetter.docx</DocumentId>
         <Bookmarks>
            <Bookmark>
               <Name>Subject</Name>
               <Text>This is my new subject</Text>
            </Bookmark>
         </Bookmarks>
      </EditDocument>
   </Instruction>
</DCML>
```

Instruction Elements

All instruction elements are sub elements of the root elements <CreateDocument> or <EditDocument>.

The following instruction elements are available in DCML:

- Bookmarks
- BuiltInDocumentProperties
- Contents
- CustomDocumentProperties
- CustomXmlParts
- DocumentId
- DocumentVariables
- Language
- MasterPropertySets
- Outputs
- TemplateID

Bookmarks

Set text to Word bookmarks.

Syntax

```
   <Bookmarks>
      <Bookmark>
         <Name></Name>
         <Text></Text>
      </Bookmark>
</Bookmarks>
```

Content

The `Bookmarks` element can contain the following sub-elements:

Name	Description
Bookmark	Optional to many elements. This element represents an individual Word bookmark.

The `Bookmark` element has the following sub-elements:

Name	Description
Name	Required string. The name of the Word bookmark.
Value	Optional string. The value of the Word bookmark.

Example

This example sets the text values for the two bookmarks Subject and Text.

```
<Bookmarks>
   <Bookmark>
      <Name>Subject</Name>
      <Text>This is the Subject</Text>
   </Bookmark>
   <Bookmark>
      <Name>Text</Name>
      <Text>And this is the text</Text>
   </Bookmark>
</Bookmarks>
```

BuiltInDocumentProperties

Set the value of Word built-in document properties.

Syntax

```
<BuiltInDocumentProperties>
     <BuiltInDocumentProperty>
        <Name></Name>
        <Text></Text>
     </BuiltInDocumentProperty>
</BuiltInDocumentProperties>
```

Content

The `BuiltInDocumentProperties` element can contain the following subelements:

Name	Description
BuiltInDocumentProperty	Optional to many elements. This element represents an individual Word built-in document property.

The `BuiltInDocumentProperty` element has the following attributes:

Name	Description
Name	Required string. The name of the Word built-in document property.
Text	Optional string. The text of the Word built-in document property.

List of modifiable built-in document properties

- Title

- Subject
- Author
- Keywords
- Category
- Comments
- Manager
- Company

Example

This example sets the text values for the two Word built-in document property Author and Subject.

```
<BuiltInDocumentProperties>
   <BuiltInDocumentProperty>
      <Name>Author</Name>
      <Text>Harry Smith</Text>
   </BuiltInDocumentProperty>
   <BuiltInDocumentProperty>
      <Name>Subject</Name>
      <Text>Offer B763Gk</Text>
   </BuiltInDocumentProperty>
</BuiltInDocumentProperties>
```

ContentControls

Set text to plain text Content Controls.

Syntax

```
<ContentControls>
   <ContentControl>
      <Tag></Tag>
      <Text></Text>
   </ContentControl>
</ContentControls>
```

Content

The `ContentControls` **element can contain the following sub-elements:**

Name	**Description**
PlainText	Optional to many elements. This element represents an individual plain text Content control.

The `PlainText` **element has the following sub-elements:**

Name	**Description**
Tag	Required string. The tag of the plain text Content Control.
Text	Optional string. The value of the plain text Content Control.

Example

This example sets the text values for the two plain text Content Controls Subject and Text.

```
<ContentControls>
   <PlainText>
      <Tag>Subject</Tag>
      <Text>This is the Subject</Text>
   </PlainText>
   <PlainText>
      <Tag>Text</Tag>
      <Text>And this is the text</Text>
   </PlainText>
</ContentControls>
```

Contents

Inserts Smart-Contents to the document. Values can also be passed along with your contents. Please be aware that complex nested table scenarios via Contents might not be supported!

Syntax

```
<Contents>
      <Content>
         <Id></Id>
         <Language></Language>
         <Bookmark></Bookmark>
         <ContentControls>
            <PlainText>
               <Tag></Tag>
               <Text></Text>
            </PlainText>
            <Picture>
               <Tag></Tag>
               <Source></Source>
               <BinaryData></BinaryData>
               <Size>
                  <Scale>
                     <Height/>
                     <Width/>
                  </Scale>
               </Size>
            </Picture>
         </ContentControls>
      </Content>
   </Contents>
```

Content

The `Contents` element can contain the following sub-elements:

Name	Description
Content	Optional to many elements. This element represents an individual content, which will be inserted into the document.

The `Content` element has the following subelements:

Name	Description
ID	Required string. The filename without the path and the file extension of the Smart-Content.
Language	Optional string. This string is only considered in root elements CreateDocument and EditDocument and represents the language ID used to insert the content. If omitted the language ID of the destination document will apply.

Bookmark	Optional string. This string can overrule the default target defined in the Smart-Content.
ContentControls	Optional element. This element can contain <PlaintText> and <Picture> sub-elements

The DCML Engine Standard supports only the full file path for the ID attribute.

The `PlainText` element has the following sub-elements:

Name	**Description**
Tag	Required string. The tag of the plain text content control.
Text	Optional string. Sets the text of the plain text content control.

The `Picture` element has the following sub-elements:

Name	**Description**
Tag	Required string. The tag of the plain text content control.
Source	Optional string. URL or path to a picture.
BinaryData	Optional BinaryData. Picture as binary data.

The Picture Content Control supports pictures with a resolution more then 72dpi.

The optional `Size` element has the following sub-elements:

Scale	**Scales the picture height and width to the size of the target picture content control.**
Height	**Scales the picture to the height of the target picture content control. The width gets the same scale.**
Width	**Scales the picture to the width of the target picture content control. The width gets the same scale.**

Example 1

This example inserts three Smart-Contents on their predefined default target bookmark into the document.

```
<Contents>
   <Content>
      <Id>01 Introduction</Id>
   </Content>
   <Content>
      <Id>01 Explanation</Id>
      <Bookmark>SpecialBookmark</Bookmark>
   </Content>
   <Content>
      <Id>02 Conclusion</Id>
      <Language>2055</Language>
   </Content>
</Contents>
```

Example 2

This example inserts three Smart-Contents into the document. In the 3rd content the text for the two plaintext content controls Title and Descripton are set.

```
<Contents>
   <Content>
      <Id>01 Introduction</Id>
   </Content>
   <Content>
      <Id>01 Explanation</Id>
   </Content>
   <Content>
      <Id>02 Conclusion</Id>
      <ContentControls>
         <PlainText>
            <Tag>Title</Tag>
            <Text>Environment</Text>
         </PlainText>
         <PlainText>
            <Tag>Description</Tag>
            <Text>The environment is strongly influenced by ...</Text>
         </PlainText>
      </ContentControls>
   </Content>
</Contents>
```

Example 3

This example inserts four Smart-Contents. In the 3rd content the Path for the picture content control is set. In the 4th content the BinaryData is set (streamed from the business application).

```
<Contents>
   <Content>
      <Id>01 Introduction</Id>
   </Content>
   <Content>
      <Id>01 Explanation</Id>
   </Content>
   <Content>
      <Id>02 Conclusion</Id>
      <ContentControls>
         <PlainText>
            <Tag>Title</Tag>
            <Text>Environment</Text>
         </PlainText>
         <PlainText>
            <Tag>Description</Tag>
            <Text>The environment is strongly influenced by…</Text>
         </PlainText>
         <Picture>
            <Tag>ChapterPicture</Tag>
            <Source>T:\corporate\pctures\chapters\environment.jpg</Source>
         </Picture>
      </ContentControls>
   </Content>
   <Content>
      <Id>02 Conclusion</Id>
      <ContentControls>
         <PlainText>
            <Tag>Title</Tag>
            <Text>Traffic</Text>
         </PlainText>
         <PlainText>
            <Tag>Description</Tag>
            <Text>The traffic depends on…</Text>
         </PlainText>
         <Picture>
            <Tag>ChapterPicture</Tag>
            <Source>path</Source>
            <BinaryData>iVBORw0KGgoAAAANSU…</BinaryData>
         </Picture>
      </ContentControls>
   </Content>
</Contents>
```

Example 4

This example inserts two Smart-Contents into the document. In the 2nd content the BinaryData is set (streamed from the business application) and the size (height and width) of the picture will be set like the size of the target picture content control.

```
<Contents>
   <Content>
      <Id>01 Introduction</Id>
   </Content>
   <Content>
      <Id>02 Conclusion</Id>
      <ContentControls>
         <Picture>
            <Tag>ChapterPicture</Tag>
            <Source>%Logos%\Ol_Logo_Division_HR.jpg</Source>
            <Size>
               <Scale>
                  <Height/>
                  <Width/>
               </Scale>
            </Size>
         </Picture>
      </ContentControls>
   </Content>
</Contents>
```

CustomDocumentProperties

Set Word custom document properties. If the custom document property does not exist, it will be created.

Syntax

```
<CustomDocumentProperties>
      <CustomDocumentProperty>
         <Name></Name>
         <Text></Text>
      </CustomDocumentProperty>
   </CustomDocumentProperties>
```

Content

The `CustomDocumentProperties` **element can contain the following sub-elements:**

Name	Description
CustomDocumentProperty	Optional to many elements. This element represents an individual Word custom document property.

The `CustomDocumentProperty` **element has the following attributes:**

Name	Description
Name	Required string. The name of the Word custom document property.
Text	Optional string. The value of the Word custom document property.

Example

This example sets the text values for the two Word custom document properties ContractNumber and DocumentType.

```
<CustomDocumentProperties>
   <CustomDocumentProperty>
      <Name>ContractNumber</Name>
      <Text>73.254.256</Text>
   </CustomDocumentProperty>
   <CustomDocumentProperty>
      <Name>DocumentType</Name>
      <Text>Contract</Text>
   </CustomDocumentProperty>
</CustomDocumentProperties>
```

CustomXmlParts

Sets specific CustomXmlParts in the document. If the CustomXmlPart does not exist, it will be created.

Syntax

```
<CustomXmlParts>
     <CustomXmlPart>
        <Namespace></Namespace>
        <XPath></XPath>
        <Value></Value>
     </CustomXmlPart>
</CustomXmlParts>
```

Content

The `CustomXmlParts` element can contain the following sub-elements:

Name	**Description**
CustomXmlPart	Optional to many elements. This element represents an individual CustomXmlPart.

The `CustomXmlPart` element has the following sub-elements:

Name	**Description**
Namespace	Required string. The namespace of the target CustomXmlPart.
XPath	Required string. The XPath of the target CustomXmlPart.
Value	Optional string. The value of the target CustomXmlPart

Example

This example sets the values for the two CustomXmlParts '/ns:officeatwork/ns:ContractNumber' and '/ns:officeatwork/ns:DocumentType'.

```
<CustomXmlParts>
   <CustomXmlPart>
      <Namespace>http://schemas.officeatwork.com/CustomXMLPart</Namespace>
      <XPath>/ns:officeatwork/ns:ContractNumber</XPath>
      <Value>73.254.256</Value>
   </CustomXmlPart>
   <CustomXmlPart>
      <Namespace>http://schemas.officeatwork.com/CustomXMLPart</Namespace>
      <XPath>/ns:officeatwork/ns:DocumentType</XPath>
      <Value>Contract</Value>
   </CustomXmlPart>
</CustomXmlParts>
```

DocumentID

Unique ID can contain the name and path of an existing document to edit.

Syntax

```
<DocumentID></DocumentID>
```

Content

A unique document ID. This can be the complete path and file name of an existing document.

For use with the DCML Engine the DocumentID has to be a full file path of an existing document.

Example 1

```
<DocumentID>Document_20140329_428954</DocumentID>
```

Example 2

```
<DocumentID> Marketing Report 2014.docx </DocumentID>
```

Example 3

```
<DocumentID>G:\Reports\Marketing Report 2014.docx</DocumentID>
```

DocumentVariables

Set Word document variables. If the document variable does not exist, it will be created.

Syntax

```
<DocumentVariables>
     <DocumentVariable>
        <Name></Name>
        <Text></Text>
     </DocumentVariable>
  </DocumentVariables>
```

Content

Contains the **value** to be allocated to the Word document variable.

The `DocumentVariables` element can contain the following sub-elements:

Name	Description
DocumentVariable	Optional to many elements. This element represents an individual Word document variable.

The `DocumentVariable` element has the following attributes:

Name	Description
Name	Required string. The name of the Word document variable.
Text	Optional string. The value of the Word document variable

Example

This example sets the text values for the two document variables MeetingDate and MeetingLocation.

```
<DocumentVariables>
   <DocumentVariable>
      <Name>MeetingDate</Name>
      <Text>18.11.2015</Text>
   </DocumentVariable>
   <DocumentVariable>
      <Name>MeetingLocation</Name>
      <Text>Room C324</Text>
   </DocumentVariable>
</DocumentVariables>
```

Language

Sets the document language.

Syntax

```
<Language></Language>
```

Content

Language code. A list of available Language codes can be found in the Appendix of this manual.

Example

```
<Language>de-ch</Language>
```

MasterProperties

Sets officeatwork specific Master-Properties in the document.

Syntax

```
<MasterPropertySets>
     <MasterPropertySet>
       <Id></Id>
       <Active />
       <SelectedForOutput />
       <MasterProperties>
         <MasterProperty>
           <Id></Id>
           <Where></Where>
           <Is></Is>
           <Fields>
             <Field>
               <Name></Name>
               <Value></Value>
             </Field>
           </Fields>
         </MasterProperty>
       </MasterProperties>
     </MasterPropertySet>
</MasterPropertySets>
```

Content

The `MasterPropertySets` **element can contain the following subelements:**

Name	**Description**
MasterPropertySet	Optional to many elements. This element represents an individual officeatwork Master-Property Set.

The `MasterPropertySet` **element can contain the following sub-elements:**

Name	**Description**
Id	**Required string. The ID of the Master-Property Set.**
Active	Optional element. Sets this set to the acitve Set for the Document Wizard.
SelectedForOutput	Optional element. Selects this set for output.
MasterProperties	Optional element. This element represents a collection of officeatwork Master-Properties.
Profile	Optional element. This element represents the chosen profile.

The `Profile` **element can contain the following sub-elements:**

Default	**Required element.** Set the users default profile for created document.

The `MasterProperties` **element can contain the following sub-elements:**

Name	**Description**
MasterProperty	Optional to many elements. This element represents an individual officeatwork Master-Property.

The `MasterProperty` element can contain the following attributes and sub-elements:

Name	Description
Id	Required string. The ID of the Master-Property.
Where	Optional string. Field name to search in
Is	Optional string. Field value to search for
Fields	Optional element. This element represents a collection of fields of the Master-Property.

You can't use the <Where> and <Is> element at the same time you are using the <Fields> element.

The `Fields` element can contain the following sub-elements:

Name	Description
Field	Optional to many element.

The `Field` element has the following sub-elements:

Name	Description
Name	Required string. The name of the Master-Property field.
Value	Optional string. The value of the Master-Property field.

Example 1

This example inserts the data for the Master-Property Author from a database in the officeatwork repository where the field "Name" equal "Peter Costa" is.

```
<MasterPropertySets>
   <MasterPropertySet>
      <Id>FirstSet</Id>
      <Active />
      <SelectedForOutput />
      <MasterProperties>
         <MasterProperty>
            <Id>Author</Id>
            <Where>Name</Where>
            <Is>Peter Costa</Is>
         </MasterProperty>
      </MasterProperties>
   </MasterPropertySet>
</MasterPropertySets>
```

Example 2

This example sets the values for the fields of the Master-Property Recipient.

```
<MasterPropertySets>
   <MasterPropertySet>
      <Id>FirstSet</Id>
      <Active />
      <SelectedForOutput />
      <MasterProperties>
         <MasterProperty>
            <Id>Recipient</Id>
            <Fields>
               <Field>
                  <Name>FullName</Name>
                  <Value>Mr. John Miller</Value>
               </Field>
               <Field>
                  <Name>Street</Name>
                  <Value>Avenue 4th</Value>
               </Field>
               <Field>
                  <Name>Zip</Name>
                  <Value>48625</Value>
               </Field>
```

```
                <Field>
                   <Name>City</Name>
                   <Value>New York</Value>
                </Field>
                <Field>
                   <Name>CompleteAddress</Name>
                   <Value>Mr. John Miller
Avenue 4th
48625 New York</Value>
                </Field>
             </Fields>
          </MasterProperty>
       </MasterProperties>
   </MasterPropertySet>
</MasterPropertySets>
```

Example 3

This example unites the Example 1 and the Example 2.

```
<MasterPropertySets>
   <MasterPropertySet>
       <Id>FirstSet</Id>
       <Active />
       <SelectedForOutput />
       <MasterProperties>
          <MasterProperty>
             <Id>Author</Id>
             <Where>Name</Where>
             <Is>Peter Costa</Is>
          </MasterProperty>
          <MasterProperty>
             <Id>Recipient</Id>
             <Fields>
                <Field>
                   <Name>FullName</Name>
                   <Value>Mr. John Miller</Value>
                </Field>
                <Field>
                   <Name>Street</Name>
                   <Value>Avenue 4th</Value>
                </Field>
                <Field>
                   <Name>Zip</Name>
                   <Value>48625</Value>
                </Field>
                <Field>
                   <Name>City</Name>
                   <Value>New York</Value>
                </Field>
                <Field>
                   <Name>CompleteAddress</Name>
                   <Value>Mr. John Miller
Avenue 4th
48625 New York</Value>
                </Field>
             </Fields>
          </MasterProperty>
       </MasterProperties>
   </MasterPropertySet>
</MasterPropertySets>
```

Example 4

This example selects the users default profile for the created document.

```
<MasterPropertySets>
   <MasterPropertySet>
       <Id>FirstSet</Id>
       <Active />
       <SelectedForOutput />
       <Profile>
          <Default />
       </Profile>
   </MasterPropertySet>
</MasterPropertySets>
```

Outputs

Uses one or more of the officeatwork output variants to output the document in different variations.

Syntax

```
<Outputs>
     <Output>
        <View></View>
        <Targets>
           <Target>
              <FileName></FileName>
              <FileFormat></FileFormat>
              <FileFormatOptions></FileFormatOptions>
              <Delivery>
                 <Save>
                    <Path></Path>
                 </Save>
                 <Download />
              </Delivery>
           </Target>
        </Targets>
     </Output>
</Outputs>
```

Content

The `Outputs` element can contain the following sub-elements:

Name	**Description**
Output	Optional to many elements. This element defines an output variation

The `Output` element can contain the following sub-elements:

View	Optional element. Defines the applied view to the document
Targets	Optional element. This elements defines the targets for output

The `Targets` element can contain the following sub-elements:

Target	Optional to many elements. Defines a target for output

The `Target` element can contain the following sub-elements:

FileName	Required string. Defines the file name to be used for the new file
FileFormat	Required string. Defines the file format
FileFormatOptions	Optional string. Defines the file format to be used for the new file. Remark: The available file formats are defined in the appendix
Delivery	Required element. Defines the delivery method for the output

The `Delivery` element can contain the following sub-elements:

Download	Optional element. This element provides the document for downloading
Save	Optional element. This element triggers an officeatwork save variation to be executed.

The `Save` element can contain the following attributes:

Name	Description
Path	Required string. Defines the path for the file to be saved to.
Open	Optional element. This element triggers to open the document at the end of the process.

The DCML Engines supports only the Save `Delivery`.

The `Open` element is only available in the DCML Engines.

The DCML Engines replaces an existing file.

Only the DCML Engine Enterprise and the EDC Server supports the PDF `file format`s.

The `Path` attributes is optional for the EDC Server.

Example 1

This example creates a document named Offer.docx in the DOCX2010 (Office 2010 version) file format and saves it onto the users desktop.

```
<Outputs>
   <Output>
      <Targets>
         <Target>
            <FileName>Offer.docx</FileName>
            <FileFormat>DOCX2010</FileFormat>
            <Delivery>
               <Save>
                  <Path>%Desktop%</Path>
               </Save>
            </Delivery>
         </Target>
      </Targets>
   </Output>
</Outputs>
```

Example 2

This example creates a document named Offer.pdf with the view Original in the PDF 1.5 file format and saves it to a local folder.

```
<Outputs>
   <Output>
      <View>Original</View>
      <Targets>
         <Target>
            <FileName>Offer.pdf</FileName>
            <FileFormat>PDF15</FileFormat>
            <Delivery>
               <Save>
                  <Path>C:\Output</Path>
               </Save>
            </Delivery>
         </Target>
      </Targets>
   </Output>
</Outputs>
```

Example 3

This example creates a document named Offer.pdf with the view Draft in the newest PDF file format and provides it for downloading. This function is only within EDC Server available.

```
<Outputs>
   <Output>
      <View>Draft</View>
      <Targets>
         <Target>
            <FileName>Offer.pdf</FileName>
            <FileFormat>PDF</FileFormat>
            <Delivery>
               <Download />
            </Delivery>
         </Target>
      </Targets>
   </Output>
</Outputs>
```

Example 4

This example creates a document named Offer.docx in the newest DOCX file format, saves it to a local folder and opens it afterwards. This function is only available within the DCML Engine.

```
<Outputs>
   <Output>
      <Targets>
         <Target>
            <FileName>Offer.docx</FileName>
            <FileFormat>DOCX</FileFormat>
            <Delivery>
               <Save>
                  <Path>C:\Output</Path>
                  <Open />
               </Save>
            </Delivery>
         </Target>
      </Targets>
   </Output>
</Outputs>
```

Example 5

This example creates different outputs with different views at the same time.

```
<Outputs>
   <Output>
      <View>Original</View>
      <Targets>
         <Target>
            <FileName>Offer.docx</FileName>
            <FileFormat>DOCX2010</FileFormat>
            <Delivery>
               <Save>
                  <Path>%Desktop%</Path>
               </Save>
            </Delivery>
         </Target>
      </Targets>
   </Output>
   <Output>
      <View>Archive</View>
      <Targets>
         <Target>
            <FileName>Offer Archive Copy.pdf</FileName>
            <FileFormat>PDF15</FileFormat>
            <Delivery>
               <Save>
                  <Path>%Desktop%</Path>
                  <Open />
               </Save>
            </Delivery>
         </Target>
         <Target>
            <FileName>Offer.pdf</FileName>
            <FileFormat>PDF15</FileFormat>
            <Delivery>
               <Save>
                  <Path>X:\Archive\FilePicker</Path>
               </Save>
            </Delivery>
         </Target>
      </Targets>
   </Output>
</Outputs>
```

TemplateId

Defines the template that should be used to create a new document or presentation.

Syntax

```
<TemplateId></TemplateId>
```

Content

Possible variations:

- Template file name without extension
- Template file name with extension
- Full file path of the templaite with extension

A file name of a template without its file extension. Optionally the path and the extension can be added as well. If the path and the extension are omitted, then the first template with the corresponding file name in the current officeatwork solution will be used to create a new document.

The DCML Engine Standard supports only the full file path variation.

Example 1

```
<TemplateId>Letter</TemplateId>
```

Example 2

```
<TemplateId>C:\officeatwork\Solutions\examplesolutioncom\MasterTemplates\Le
tter.owt</TemplateId>
```

Example 3

```
<TemplateId>G:\SharedTemplates\Letter.dotx</TemplateId>
```

CHAPTER 5

DCML Formulas

Introduction

The DCML furmulas are resolved before the document will be created. With this formulas you can use the values from the <Data> element in other parts of the DCML file.

The implemented XPath functions are based on Microsoft MSXML.
Xpath Examples: https://msdn.microsoft.com/en-us/library/aa924034.aspx

Syntax

To make your Enterprise Document Creation process dynamic, you can use the formulas described in this chapter. This formulas need to conform to the following syntax rules:

- A function starts with two opening square "[[" and ends with two closing "]]" brackets.
- If a function is nested in another function it has NO leading and closing square brackets.
- The parameters are between a round opening "(" and a closing ")" bracket.
- Every parameter starts and ends with a single quote.
- The parameters in the list are separated by a comma.
- Every formula will be replaced by its own result.

The return value of every function by default is a string value. If there is another value type it will be declared.

Example

This is an abstract example of a DCML formula syntax.

```
Input:    <Value>[[FormluaName('Paramter 1', 'Paramter 2', [Optional Parameter n'])]]</Value>

Output:   <Value>Result of the formula</Value>
```

Ampersand &

Description

With the ampersand you can concatenate strings.

Syntax

```
"String 1" & "String 2"
```

Parameter

There are no parameters

Example 1

The following example concatenates two strings together.

```
Input:    "Hello " & "world!"

Output:   "Hello world!"
```

Example 2

The following example loops over contacts in the <Data> element. For each contact will be added a Master-Property with the field named “Fullname”. The value for the field is concatenation of the contact values first name and last name with a space between.

```
Input:    [[XpathLoop('ContactLoop', '//Contacts/Contact', '
          <MasterProperty>
             <Id>Recipient</Id>
             <Fields>
                <Field>
                   <Name>Fullname</Name>
                   <Value>' & XpathLoopValue('ContactLoop', '//Lastname') & ', ' &
                      XpathLoopValue('ContactLoop', '//Firstname') & '</Value>
                </Field>
             </Fields>
          </MasterProperty>')]]

Output:   <MasterProperty>
             <Id>Recipient</Id>
             <Fields>
                <Field>
                   <Name>Fullname</Name>
                   <Value>Miller Peter</Value>
                </Field>
             <Fields>
          </MasterProperty>
          <MasterProperty>
             <Id>Recipient</Id>
             <Fields>
                <Field>
                   <Name>Fullname</Name>
                   <Value>Jacob Susan</Value>
                </Field>
             <Fields>
          </MasterProperty>
          …
```

ADOData()

Description

The formula ADOData() gets all the data from a ADO data source in XML format.

Syntax

```
[[ADOData('Name', 'Source', 'Statement')]]
```

Parameter

Name	Name of the Provider
Source	Source of the Privder
Statement	SQL statement to select the data

To use this function «Microsoft Access Database Engine 2010» has to be installed on the EDC-Server.

Example 1

This example gets some data from an Excel file in the officeatwork Repository.

```
Input:    [[ADOData('System.Data.OleDb', 'Provider=Microsoft.ACE.OLEDB.12.0;Data
          Source=%Features%\DcmlFunctions\AdoData.xlsx;Extended Properties="Excel 8.0;HDR=YES"', 'SELECT *
          FROM Persons')]]

Output:   <Person>
             <LastName>Miller</LastName>
             <FirstName>Peter</FirstName>
          </Person>
          <Person>
             <LastName>Stevenson</LastName>
             <FirstName>Suzanna</FirstName>
          </Person>
```

Example 2

This example gets some data from an Access database located in the officeatwork Repository.

```
Input:    [[ADOData('System.Data.OleDb', 'Provider=Microsoft.ACE.OLEDB.12.0;Data
          Source="AdoData.accdb";Persist Security Info=False;', 'SELECT * FROM PersonData')]]

Output:   <Person>
             <Id>93564-09871</Id>
             <LastName>Schmidt</LastName>
             <FirstName>Eugen</FirstName>
          </Person>
          <Person>
             <Id>34581-91252</Id>
             <LastName>Durban</LastName>
             <FirstName>Svea</FirstName>
          </Person>
```

And()

Description

The conditional And() formula performs a logical-AND of its boolean operands.

Syntax

```
[[And('Operand 1', 'Operand 2', ['Operand n'])]]
```

Parameter

Operand 1	Operand with a boolean return value
Operand 2	Operand with a boolean return value
Operand n	Optional operand with a boolean return value
	The operands can contain logical operators.

Example 1

This example checks two operands and returns the boolean result.

```
Input:    <Result>[[And('2' = '2', 'a' = 'A', '3.4' = '03.40')]]</Result>

Output:   <Result>True</Result>
```

Example 2

In this example is the And() formula a part of the If() formula condition.

```
Input:    <Result>[[If(And('2' = '2', 'a' = 'A'), 'Condition is true', 'Condition is false')]]</Result>

Output:   <Result>Condition is true</Result>
```

Block()

Description

With the Block() formula you can concatenate some values separated by a specific separator.

Syntax

```
[[Block('Separator', 'Value 1', 'Value 2', ['Value n'])]]
```

Parameter

Separator	Defines the separator sign between the concatenated string of all values
Value 1	String to concatenate separated by the separator
Value 2	String to concatenate separated by the separator
Value n	Optional string to concatenate separated by the separator

Example 1

The following example blocks the values separated by the defined separator "::".

```
Input:    <Text>[[Block('::', 'Hello', 'world', '!')]]</Text>

Output:   <Text>Hello::world::!</Text>
```

Example 2

The following example blocks the values separated by the defined separator "linefeed".

```
Input:    <Text>[[Block('
                 ', 'Hello', 'world', '!')]]</Text>

Output:   <Text>Hello
          world
          !</Text>
```

If()

Description

With the If() formula you can create conditional content.

Syntax

```
[[If('Condition', 'ValueWhenTrue', 'ValueWhenFalse')]]
```

Parameter

Condition	Condition to test The condition can contain other DCML formulas logical operators.
ValueWhenTrue	Return value if the condition is True
ValueWhenFalse	Return value if the condition is False

Example 1

The following example checks if the value '4' is greater then the value '2'.

```
Input:    <Result>[[If('4' > '2', 'result is bigger than', 'result is smaller or equal than')]]</Result>

Output:   <Result>result is bigger than</Result>
```

Example 2

The following example checks two parts in the condition.

```
Input:    <Result>[[If(And('2' = '2', 'a' = 'A'), 'Condition is true', 'Condition is false')]] </Result>

Output:   <Result>Condition is true</Result>
```

Example 3

In the following example we insert a <content> element when we have some accounts in the <Data> element.

```
Input:    <Contents>
             [[If(XpathCount('//Accounts/Account') > '0', '
             <Content>
                <Id>Introduction Accounts</Id>
             </Content>', '')]]
          </Contents>

Output:   <Contents>
             <Content>
                <Id>Introduction Accounts</Id>
             </Content>
          </Contents>
```

Or()

Description

The conditional Or() formula performs a logical-OR of its boolean operands.

Syntax

```
[[Or('Operand 1', 'Operand 2', ['Operand n'])]]
```

Parameter

Operand 1	Condition with a boolean return value
Operand 2	Condition with a boolean return value
Operand n	Optional condition with a boolean return value
	The conditions can contain other DCML formulas logical operators.

Example 1

The following example checks two operands and returns the boolean result.

```
Input:    <Result>[[Or('2' = '2', 'a' = 'B', '3.4' = '19.78')]]</Result>

Output:   <Result>True</Result>
```

Example 2

In this example is the Or() formula is a part of the If() formula condition.

```
Input:    <Result>[[If(Or('2' = '2', 'a' = 'B'), 'Condition is true', 'Condition is false')]] </Result>

Output:   <Result>Condition is true</Result>
```

XpathCount()

Description

With the XpathCount() formula you can count the amount of specific subelements in the <Data> element in the DCML file.

Syntax

```
[[XpathCount('XpathExpression')]]
```

Parameter

XpathExpression	Xpath expression relative to the Data element in the DCML file

Example

The following example counts the amount of contact sub elements in the <Contacts> element of the <Data> element.

```
Input:    <AmountOfContacts>[[XpathCount('//Contacts/Contact')]]</AmountOfContacts>

Output:   <AmountOfContacts>17</AmountOfContacts>
```

XpathLoop()

Description

With the XpathLoop() formula you can loop over a subset of sub elements of the <Data> element in the DCML file.

Syntax

```
[[XpathLoop('LoopId', 'XpathExpression', 'Content')]]
```

Parameter

LoopId	Unique identifier (id) for this loop

XpathExpression	Xpath expression relative to the Data element in the DCML file
Content	The output content for each element in this loop

Example 1

The following example loops over all <Account> sub elements in the sub element <Accounts> in the <Data> element.

```
Input:    <Accounts>[[XpathLoop('AccountLoop', '//Accounts/Account',
                              '<AccountFound />')]]
          </Accounts>

Output:   <Accounts>
             <AccountFound />
             <AccountFound />
             <AccountFound />
             …
          </Accounts>
```

Example 2

The following example loops over all <Account> sub elements in the sub element <Accounts> in the <Data> element and reads the value of the <Name> sub element of each Account.

```
Input:    <Accounts>[[XpathLoop('AccountLoop', '//Accounts/Account', '<AccountName>' &
          XpathLoopValue('AccountLoop', '//Name') & '</AccountName>')]]</Accounts>
Output:   <Accounts>
             <AccountName>Miller Ltd.</AccountName>
             <AccountName>Jacob Industries</AccountName>
             <AccountName>Lifewood Foundation</AccountName>
             …
          </Accounts>
```

XpathLoopValue()

Description

With the XpathLoopValue() formula you retrieve values from the actual recsordset during a loop over a subset of sub elements of the <Data> element in the DCML file.

Syntax

```
[[XpathLoopValue('LoopId', 'XpathExpression')]]
```

Parameter

LoopId	Unique identifier (id) for this loop
XpathExpression	Xpath expression relative to the XpathLoop element in the DCML file

The XpathLoopValue() formula can only be used in combination with the XpathLoop() formula.

Example 1

The following example loops over all <Account> sub elements in the sub element <Accounts> in the <Data> element and reads the value of the <Name> sub element of each Account.

```
Input:    <Accounts>
          [[XpathLoop('AccountLoop', '//Accounts/Account', '<AccountName>' &
          XpathLoopValue('AccountLoop', '//Name') & '</AccountName>')]]
          </Accounts>
```

```
Output:   <Accounts>
             <AccountName>Miller Ltd.</AccountName>
             <AccountName>Jacob Industries</AccountName>
             <AccountName>Lifewood Foundation</AccountName>
             …
          </Accounts>
```

Example 2

The following example loops over all <Contact> sub elements in the sub element <Contacts> in the <Data> element. It reads the values of the <Name> and <Street> sub elements and concatenates the postal code and the city name of each Contact.

```
Input:    <Recipients>
             [[XpathLoop('ContactLoop', '//Contacts/Contact', '<Recipient>' & '
                <Name>' & XpathLoopValue('ContactLoop', '//Name') & '</Name>
                <Address>' & XpathLoopValue('ContactLoop', '//Street') & '</Address>
                <PostalcodeAndCity>' & XpathLoopValue('ContactLoop', '//PostalCode) & ' ' &
                XpathLoopValue('ContactLoop', '//City') & '</PostalcodeAndCity>
             </Recipient>')]]
          </Recipients>

Output:   <Recipients>
             <Recipient>
                <Name>Peter Miller</Name>
                <Address>Main Street 17</Address>
                <PostalcodeAndCity>8000 Zurich</PostalcodeAndCity>
             </Recipient>
             <Recipient>
                <Name>Susan Nicks</Name>
                <Address>Park Avenue 342</Address>
                <PostalcodeAndCity>6300 Zug</PostalcodeAndCity>
             </Recipient>
             …
          </Recipients>
```

XpathValue()

Description

With the XpathValue() formula you can retrieve values from the <Data> element into another part of the DCML file.

Syntax

```
[[XpathValue('XpathExpression')]]
```

Parameter

XpathExpression	Xpath expression relative to the Data element in the DCML file

Example

The following example reads the language id from the sub element <UserLanguage> of the <Data> element.

```
Input:    <LanguageId>[[XpathValue('//UserLanguage')]]</LanguageId>

Output:   <LanguageId>1033</LanguageId>
```

Appendix

FileFormats

Abbrevation	Description
DOCX	The newest supported DOCX[YYYY]-Format
DOCX2007	Microsoft Office Document Open XML Format 2007
DOCX2010	Microsoft Office Document Open XML Format 2010
DOTX	The newest supported DOTX[YYYY]-Format
DOTX2007	Microsoft Office Template Open XML Format 2007
DOTX2010	Microsoft Office Template Open XML Format 2010
PDF	The newest supported PDF[Version]-Format
PDF15	PDF Files in the PDF 1.5 Format
PDFA1b	PDF Files in the PDF/A-1b Format

File System Variables

% Shortcut %	Destination
ADMINTOOLS	<Benutzer>\Startmenü\Programme\Verwaltung
ALTSTARTUP	Startup
APPDATA	<Benutzer>\Anwendungsdaten
COMMONADMINTOOLS	All Users\Startmenü\Programme\Verwaltung
COMMONALTSTARTUP	Common Startup
COMMONAPPDATA	All Users\Anwendungsdaten
COMMONDESKTOPDIRECTORY	All Users\Desktop
COMMONDOCUMENTS	All Users\Dokumente
COMMONFAVORITES	All Users\Favoriten
COMMONPROGRAMS	All Users\Startmenü\Programme
COMMONSTARTMENU	All Users\Startmenü
COMMONSTARTUP	All Users\Startmenü\Autostart
COMMONTEMPLATES	All Users\Vorlagen
COOKIES	<Benutzer>\Cookies
DESKTOP	<Desktop>
DESKTOPDIRECTORY	<Benutzer>\Desktop
FAVORITES	<Benutzer>\Favoriten
FONTS	Windows\Fonts
HISTORY	<Benutzer>\Lokale Einstell.\Verlauf
INTERNET_CACHE	<Benutzer>\Lokale Einstell.\Temp. Internet Files
LOCALAPPDATA	<Benutzer>\Lokale Einstell.\Anwendungsdaten
MYPICTURES	Eigene Bilder
NETHOOD	<Benutzer>\Netzwerkumgebung
OFFICEATWORK	PROGRAMFILES & "\officeatwork"
PERSONAL	Eigene Dateien
PRINTHOOD	<Benutzer>\Druckumgebung
PROFILE	Benutzerprofil
PROGRAMFILES	C:\Programme
PROGRAMFILESCOMMON	C:\Programme\Gemeinsame Dateien
PROGRAMS	Startmenü\Programme
RECENT	<Benutzer>\Recent
SENDTO	<Benutzer>\SendTo
STARTMENU	<Benutzer>\Startmenü
STARTUP	Startmenü\Programme\Autostart
SYSTEM	GetSystemDirectory()
TEMP	TEMP

% Shortcut %	Destination
TEMPLATES	<Benutzer>\Vorlagen
WINDOWS	GetWindowsDirectory()

Remark: The set of available file system variables depends on the operating system version. For further information please check the website of the manufacturer.

LCID's

Language	ID	Language	ID
Afrikaans - South Africa	1078	Chinese - Macao SAR	5124
Albanian - Albania	1052	Croatian	1050
Amharic - Ethiopia	1118	Croatian (Bosnia/Herzegovina)	4122
Arabic - Saudi Arabia	1025	Czech	1029
Arabic - Algeria	5121	Danish	1030
Arabic - Bahrain	15361	Divehi	1125
Arabic - Egypt	3073	Dutch - Netherlands	1043
Arabic - Iraq	2049	Dutch - Belgium	2067
Arabic - Jordan	11265	Edo	1126
Arabic - Kuwait	13313	English - United States	1033
Arabic - Lebanon	12289	English - United Kingdom	2057
Arabic - Libya	4097	English - Australia	3081
Arabic - Morocco	6145	English - Belize	10249
Arabic - Oman	8193	English - Canada	4105
Arabic - Qatar	16385	English - Caribbean	9225
Arabic - Syria	10241	English - Hong Kong SAR	15369
Arabic - Tunisia	7169	English - India	16393
Arabic - U.A.E.	14337	English - Indonesia	14345
Arabic - Yemen	9217	English - Ireland	6153
Armenian - Armenia	1067	English - Jamaica	8201
Assamese	1101	English - Malaysia	17417
Azeri (Cyrillic)	2092	English - New Zealand	5129
Azeri (Latin)	1068	English - Philippines	13321
Basque	1069	English - Singapore	18441
Belarusian	1059	English - South Africa	7177
Bengali	1093	English - Trinidad	11273
Bengali (Bangladesh)	2117	English - Zimbabwe	12297
Bosnian (Bosnia/Herzegovina)	5146	Estonian	1061
Bulgarian	1026	Faroese	1080
Burmese	1109	Farsi	1065
Catalan	1027	Filipino	1124
Cherokee - United States	1116	Finnish	1035
Chinese - People's Republic of China	2052	French - France	1036
Chinese - Singapore	4100	French - Belgium	2060
Chinese - Taiwan	1028	French - Cameroon	11276
Chinese - Hong Kong SAR	3076	French - Canada	3084

Language	ID
French - Democratic Rep. of Congo	9228
French - Cote d'Ivoire	12300
French - Haiti	15372
French - Luxembourg	5132
French - Mali	13324
French - Monaco	6156
French - Morocco	14348
French - North Africa	58380
French - Reunion	8204
French - Senegal	10252
French - Switzerland	4108
French - West Indies	7180
Frisian - Netherlands	1122
Fulfulde - Nigeria	1127
FYRO Macedonian	1071
Gaelic (Ireland)	2108
Gaelic (Scotland)	1084
Galician	1110
Georgian	1079
German - Germany	1031
German - Austria	3079
German - Liechtenstein	5127
German - Luxembourg	4103
German - Switzerland	2055
Greek	1032
Guarani - Paraguay	1140
Gujarati	1095
Hausa - Nigeria	1128
Hawaiian - United States	1141
Hebrew	1037
Hindi	1081
Hungarian	1038
Ibibio - Nigeria	1129
Icelandic	1039
Igbo - Nigeria	1136
Indonesian	1057
Inuktitut	1117
Italian - Italy	1040
Italian - Switzerland	2064
Japanese	1041
Kannada	1099
Kanuri - Nigeria	1137
Kashmiri	2144
Kashmiri (Arabic)	1120
Kazakh	1087
Khmer	1107
Konkani	1111
Korean	1042
Kyrgyz (Cyrillic)	1088
Lao	1108
Latin	1142
Latvian	1062
Lithuanian	1063
Malay - Malaysia	1086
Malay - Brunei Darussalam	2110
Malayalam	1100
Maltese	1082
Manipuri	1112
Maori - New Zealand	1153
Marathi	1102
Mongolian (Cyrillic)	1104
Mongolian (Mongolian)	2128
Nepali	1121
Nepali - India	2145
Norwegian (Bokmål)	1044
Norwegian (Nynorsk)	2068
Oriya	1096
Oromo	1138
Papiamentu	1145
Pashto	1123
Polish	1045
Portuguese - Brazil	1046
Portuguese - Portugal	2070
Punjabi	1094
Punjabi (Pakistan)	2118
Quecha - Bolivia	1131

Language	ID
Quecha - Ecuador	2155
Quecha - Peru	3179
Rhaeto-Romanic	1047
Romanian	1048
Romanian - Moldava	2072
Russian	1049
Russian - Moldava	2073
Sami (Lappish)	1083
Sanskrit	1103
Sepedi	1132
Serbian (Cyrillic)	3098
Serbian (Latin)	2074
Sindhi - India	1113
Sindhi - Pakistan	2137
Singhalese - Sri Lanka	1115
Slovak	1051
Slovenian	1060
Somali	1143
Sorbian	1070
Spanish - Spain (Modern Sort)	3082
Spanish - Spain (Traditional Sort)	1034
Spanish - Argentina	11274
Spanish - Bolivia	16394
Spanish - Chile	13322
Spanish - Colombia	9226
Spanish - Costa Rica	5130
Spanish - Dominican Republic	7178
Spanish - Ecuador	12298
Spanish - El Salvador	17418
Spanish - Guatemala	4106
Spanish - Honduras	18442
Spanish - Latin America	58378
Spanish - Mexico	2058
Spanish - Nicaragua	19466
Spanish - Panama	6154
Spanish - Paraguay	15370
Spanish - Peru	10250
Spanish - Puerto Rico	20490
Spanish - United States	21514
Spanish - Uruguay	14346
Spanish - Venezuela	8202
Sutu	1072
Swahili	1089
Swedish	1053
Swedish - Finland	2077
Syriac	1114
Tajik	1064
Tamazight (Arabic)	414
Tamazight (Latin)	1119
Tamil	1097
Tatar	1092
Telugu	1098
Thai	1054
Tibetan - Bhutan	2129
Tibetan - People's Republic of China	1105
Tigrigna - Eritrea	2163
Tigrigna - Ethiopia	1139
Tsonga	1073
Tswana	1074
Turkish	1055
Turkmen	1090
Uighur - China	1152
Ukrainian	1058
Urdu	1056
Urdu - India	2080
Uzbek (Cyrillic)	2115
Uzbek (Latin)	1091
Venda	1075
Vietnamese	1066
Welsh	1106
Xhosa	1076
Yi	1144
Yiddish	1085
Yoruba	1130
Zulu	1077

Logical Operators

Logical Operators

Description

The logical operators are required to compare values with each other. The operators can be used with conditions in DCML formulas and Document functions.

Syntax

```
Value1 Operator Value2
```

Parameter

Operator	Logical Operator The following operators are available: > Greater >= Greater or equal < Less <= Less or equal <> Not equal = equal

Remark: When possible the values are compared as numbers, if not they are compared as letters.
Remark: The operators are case insensitive.

Example 1

The following example compares three conditions in a DCML formula and returns the boolean value as string.

```
Input:    <Result>[[And('2' >= '2', 'a' = 'A', '3.4' <= '03.40')]]</Result>

Output:   <Result>True</Result>
```

Example 2

The following example compares two conditions in a DCML formula and returns a defined string as result.

```
Input:    <Result>[[If(And('2' = '2', 'a' = 'A'), 'Condition is true', 'Condition is false')]]</Result>

Output:   <Result>Condition is true</Result>
```

Example 3

The following example compares two conditions in a document function and returns a defined string as result.

```
Input:    [[If(And("2" = "2", "a" = "A"), "Condition is true", "Condition is false")]]

Output:   Condition is true
```

C H A P T E R 6

Support

Get access to a wide range of support resources on officeatwork Connect (connect.officeatwork.com) such as:

- Knowledge Base
- Q & A
- Download Center
- Installers
- Manuals
- Video guides
- Forum
- Glossary
- etc.

To access officeatwork Connect you need to register your Microsoft-Account at www.officeatwork.com → Connect

---

All support options and resources can be found on the website www.officeatwork.com → Support

More services offered by officeatwork such as Education and Consulting can be found on the website www.officeatwork.com → Services

Index

officeatwork AG
Bundesplatz 12
6300 Zug, Switzerland

T +41 41 544 7100

www.officeatwork.com
mail@officeatwork.com