Manual for developers

officeatwork API

Version 3.10

This Manual was created using officeatwork Advanced.
21. September 2010

officeatwork AG
Bundesplatz 12
CH-6300 Zug
Switzerland

H: 0900 566 088 (Hotline, CHF 3.50 per minute, on the Swiss landline network, price subject to change)
T: +41 (0)41 763 16 70
F: +41 (0)41 763 16 75

mail@officeatwork.com
www.officeatwork.com

Table of Contents

CHAPTER 1

About this Manual

The first chapter contains information regarding the structure of this book as well as its audience.

For whom this book is designed

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

What is covered in this manual

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.

What you already should know

You should be familiar with the general use of computers, especially with the XML notation. 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 or items in a list.
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`.

Feedback

We are very interested in your opinion. We kindly ask you to present us with any feedback you have, as well as with any other aspects of officeatwork, by writing an e-mail to feedback@officeatwork.com.

General contact details

officeatwork AG
Bundesplatz 12
6300 Zug
Switzerland

H: 0900 566 088 (Hotline, CHF 3.50 per minute, on the Swiss landline network, Price subject to change)
T: +41 (0)41 763 16 70
F: +41 (0)41 763 16 75

E-Mail: mail@officeatwork.com
Internet: www.officeatwork.com

CHAPTER 2

Help

This chapter explains the different help options available.

Manuals

officeatwork manuals are divided into the following categories:

Manuals for Users

- officeatwork 20-Minute Guide
- officeatwork User Manual*
- officeatwork Smart-Template Manager Manual
- officeatwork Smart-Content Manager Manual

Manuals for Administrators

- officeatwork Installation Guide
- officeatwork Output Method Instructions (OOMI) Guide
- officeatwork Master-Template Manager Manual**
- officeatwork Solution Manager Manual*

Manuals for Project Managers

- officeatwork Project Manager Manual*
- officeatwork Project Preparation Manual
- officeatwork sample solution specifications

Manuals for Developers

- officeatwork API

* some chapters published
** not yet published

The manuals can be downloaded from the officeatwork website
www.officeatwork.com – Download

Support

Support for officeatwork products can be obtained from your officeatwork project partner or from the producer itself.

The producer (officeatwork AG) offers the following support options:

Hotline 0900 566 088 (CHF 3.50 per minute, on the Swiss landline network)

Talk directly to our qualified officeatwork specialists. For personal support we recommend that you use our online-support-tool. The tool can be used without having to install it. The link to download the tool can be found on our website at www.officeatwork.com -> Support.

Premium support

Send your questions by e-mail to premiumsupport@officeatwork.com or call us at +41 (0)41 763 16 73 (officeatwork AG). Your questions will be answered by our officeatwork specialists with priority. If desired, your officeatwork solution will be analysed by our officeatwork specialists. If necessary, changes to your solution can be made at your request. For more information check out our website at www.officeatwork.com -> Support.

E-Mail support (free)

Send us your questions by e-mail to support@officeatwork.com. This support is free. Your questions will be answered by our support department on average within 5 to 10 working days. For more information check out our website at www.officeatwork.com -> Support.

1 Incident

You receive telephone support per incidence at a fixed price. Per single incident you will receive telephone support until the conclusion of your incident has been reached. For more information check out our website at www.officeatwork.com -> Support.

5 Incidents

5 incidents are the equivalent to five single incidents. With the 5-incident package you will receive five incidents for the price of four. For more information check out our website at www.officeatwork.com -> Support.

Unlimited support per month

You are provided with unlimited telephone support for one full month at a fixed price. For more information check out our website at www.officeatwork.com -> Support.

Manuals

You can download the most up-to-date manuals at www.officeatwork.com -> Download.

Newsletter

officeatwork-News is "good news" with added value. officeatwork-News informs you regularly about relevant topics regarding «corporate office automation». The qualified news keeps you up-to-date and helps you to increase the practical value of Microsoft Office. For more information and registration check out our website at www.officeatwork.com -> Support.

CHAPTER 3

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 error.

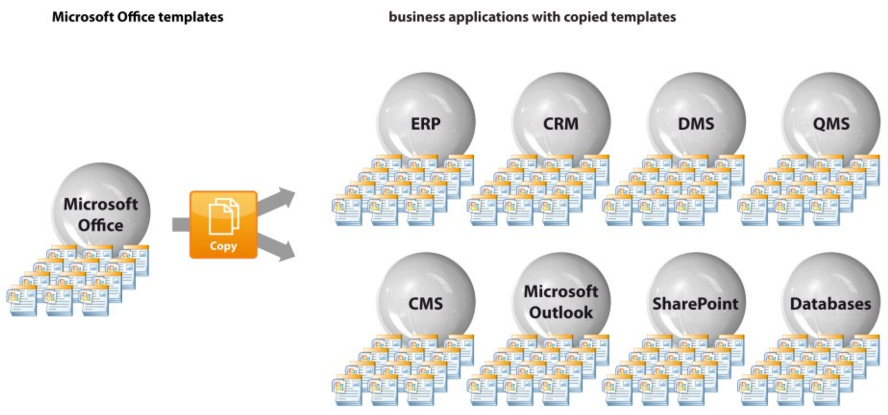

Figure 1: Typical Microsoft Office integration architecture

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.

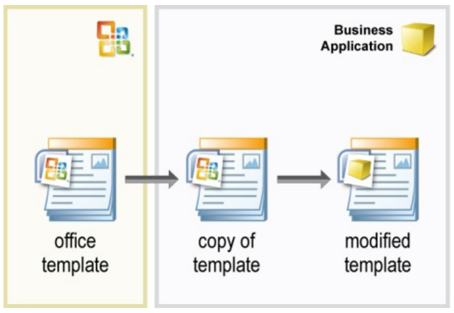

Figure 2: creating a business application specific office template

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.

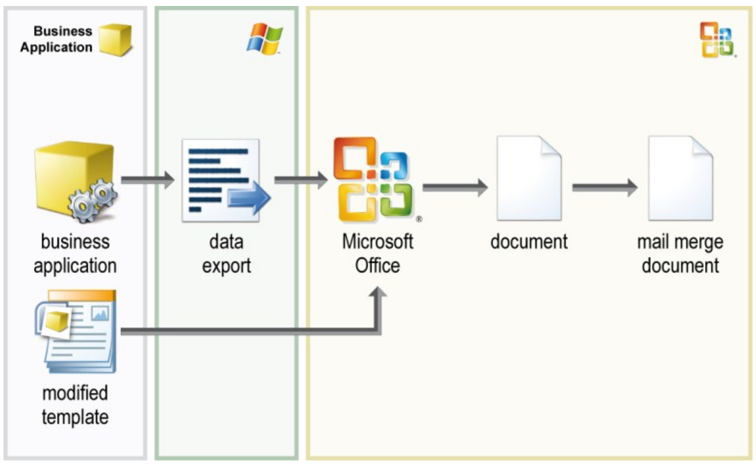

Figure 3: business application office integration concept via mail merge function

Pros:

- Using existing functions reduces effort of integration.

Cons:

- Duplication of already existing templates
- Direct dependency on the Office application version and its offered functionality
- In-depth knowledge about the Office Application required
- Testing for each new Office-version 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.

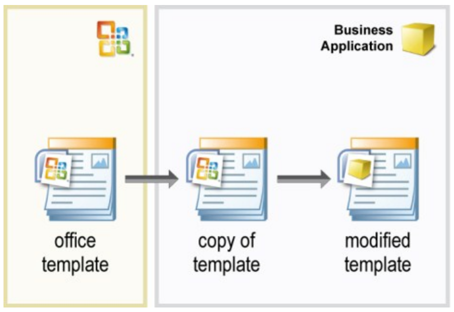

Figure 4: creating a business application specific office template

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).

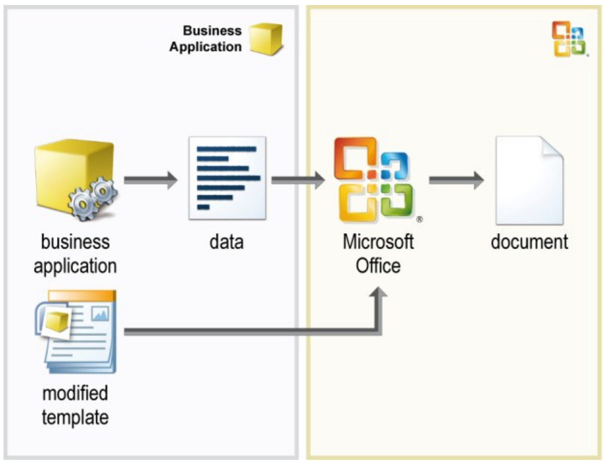

Figure 5: business application office integration concept via bookmarks, DDE/OLE and Co.

Pros:

- Highly flexible as the full object-model of the office applications are 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 Office-version 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.

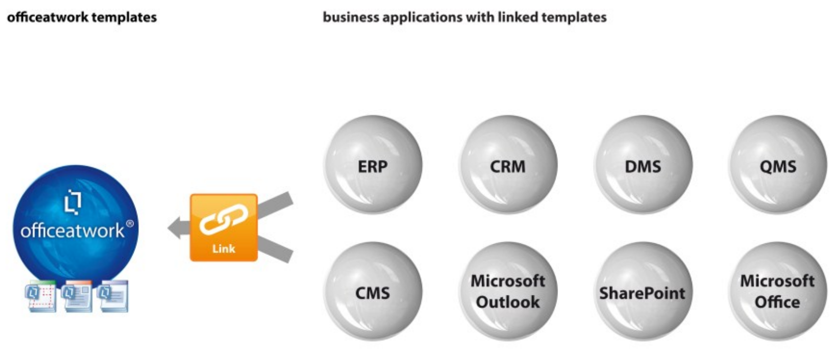

Figure 6: officeatwork integration concept

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.

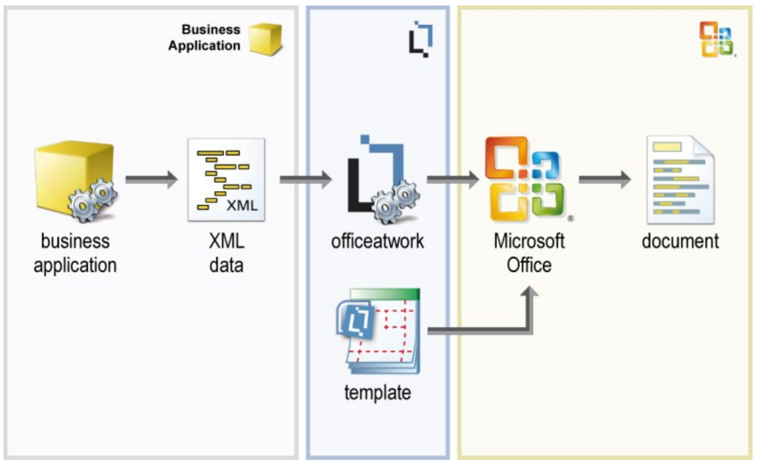

Figure 7: officeatwork business application integration concept

Pros:

- No duplication of already existing templates.
- 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 4

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.

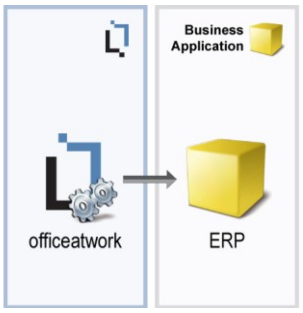

Figure 8: officeatwork interacting with a business application

Here the starting point is officeatwork. From within Microsoft Office/officeatwork data is fetched from your business application and retrieved into 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 other direction is from your Business application to officeatwork.

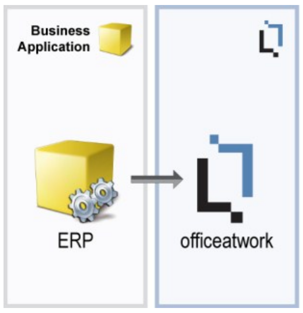

Figure 9: Business application interacting with officeatwork

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 the «officeatwork Director Manual».

Interaction concepts

officeatwork offers two very similar ways for you to create documents via the officeatwork API. The parameters of the two ways are the same, just the way officeatwork is involved differs. The first way is by calling a method and passing it XML parameters as an argument, the other is by saving the same arguments to a file and then sending the OS an Open command to execute that file. This way officeatwork gets the file and processes it, hence the API gets triggered.

Files

officeatwork API can be triggered by opening a file with the extension *.osc (officeatwork shortcut file). The content of that file must be in XML format and must follow the XML schemes of the officeatwork API. The following sample shows a simple example of such a *.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. Otherwise clicking on links pointing to OSC files will open the OSC file in your web browser instead of executing the officeatwork API. So make sure your web servers have a MIME type **application/osc** for the extension **.osc** defined.

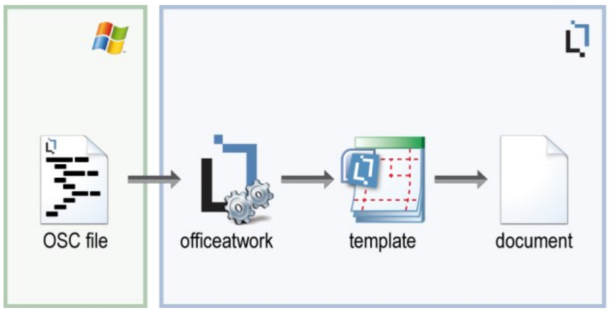

Figure 10: business application office integration concept officeatwork shortcut file

Please note that there is an equivalent method called «ExecuteXML» which can be passed the same XML information enclosed in the «Parameters» tag.

Method

officeatwork can also be triggered by calling specific officeatwork methods. All available methods will be discussed in a later chapter.

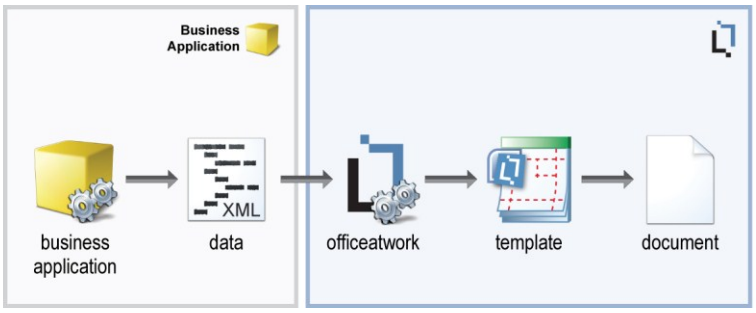

Figure 11: business application office integration concept via officeatwork specific method

The code sample shows how to call an officeatwork method within a VBA environment:

```
Sub CreateLetter ()

   Dim lOawAPI As oawAPI.API
   Dim lParam As String

   lParam = "<?xml version="1.0" encoding="ISO-8859-1"?>"
   lParam = lParam & "<Parameters>"
   lParam = lParam & "    <CreateDocument>"
   lParam = lParam & "        <TemplateID>Letter</TemplateID>"
   lParam = lParam & "    </CreateDocument>"
   lParam = lParam & "</Parameters>"

   Set lOawAPI = New oawAPI.API

   If (lOawAPI.ExecuteXML(lParam) = -1) Then
      MsgBox "successfully created document"
   Else
      MsgBox "the document could not be created"
   End If

   Set lOawAPI = Nothing

End Sub
```

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. If you do not know what XML is, we recommend that you first learn more about XML before you continue reading this manual.

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.

Sample encoding tag

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

XML Parameter

The XML Parameter consists of easy-to-follow instructions that are processed when passed to officeatwork. The following simple XML file is a sample that, when executed, will create a letter and present the finished document to the user.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Parameters>
    <CreateDocument>
        <TemplateID>letter</TemplateID>
        <ShowDocumentWizard>0</ShowDocumentWizard>
    </CreateDocument>
</Parameters>
```

Parameters tag

All instructions must be enclosed in a `<Parameters>` tag.

Syntax

```
<Parameters>
</Parameters>
```

Other XML tags on the root level of the XML file will be ignored by officeatwork. The «CreateDocument» tag within the «Parameters» tag will instruct officeatwork to create a new document. The tags within the «CreateDocument» tag describe how to create the document. The «TemplateID» tag lets officeatwork know what template to use for the creation of the new document. In this case it will be a document based on the template with the filename «letter». The «ShowDocumentWizard» tag is set to «0» so the document wizard will be completed without any user interaction necessary. All default values in the wizard will apply.

The same sample using VBA and the API Method provided by officeatwork would look as follows:

```
Sub CreateLetter ()

    Dim lOawAPI As oawAPI.API
    Dim lParam As String

    lParam = "<?xml version="1.0" encoding="ISO-8859-1"?>"
    lParam = lParam & "<Parameters>"
    lParam = lParam & "    <CreateDocument>"
    lParam = lParam & "        <TemplateID>Letter</TemplateID>"
    lParam = lParam & "        <ShowDocumentWizard>0</ShowDocumentWizard>"
    lParam = lParam & "    </CreateDocument>"
    lParam = lParam & "</Parameters>"

    Set lOawAPI = New oawAPI.API

    If (lOawAPI.ExecuteXML(lParam) = -1) Then
        MsgBox "successfully created document"
    Else
        MsgBox "the document could not be created"
    End If

    Set lOawAPI = Nothing

End Sub
```

In this sample lParam holds the XML parameter passed to officeatwork using the ExecuteXML function of officeatwork.

The template ID could be replaced with any other template ID available within your officeatwork solution. This way your business application could request any specific template. If no Template ID is passed (<TemplateID></TemplateID>) the user will be prompted with the Template Chooser to choose a template that shall be used to create a new document.

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 OSC-File as a template as a base to generate a new OSC-File. During this process it replaces placeholders within the OSC Template-File representing business application values with the proper values. The OSC-Template File may also contain specific business application instructions like loops or counters. The processed template is then processed by officeatwork.

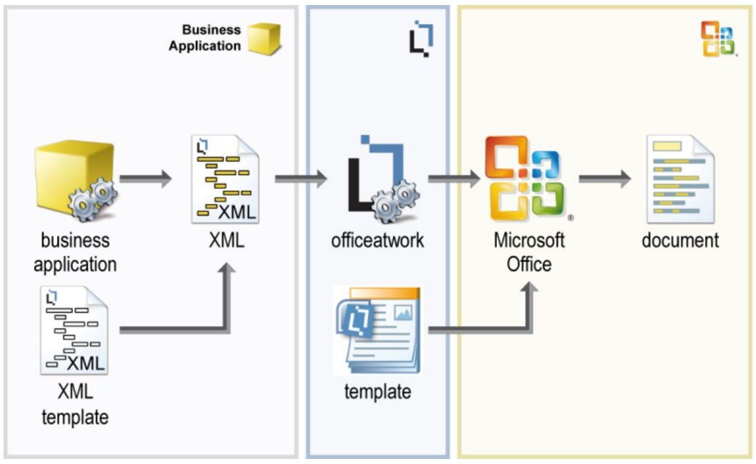

Figure 12: recommended architecture for creating an office document out of your business application via officeatwork

This architecture has many advantages:

- Limited programming necessary – the business application only needs to be taught how to process the OSC shortcut 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 OSC-File, you do not need to wait until the programming is finished to test the interface. You can just create sample osc-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.

Samples

Sample 1:

This simple OSC Template File creates an Invoice summary document and prints it with an officeatwork output-management variant. The document is closed without saving.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Parameters>
   <CreateDocument>
      <TemplateID>Invoice</TemplateID>
      <ShowDocumentWizard>0</ShowDocumentWizard>
      <Language>::ERP_account_lcid::</Language>
      <MasterProperties>
         <MasterProperty IDName="Company" Where="IDName" Is="::ERP_organisation_name::"/>
         <MasterProperty IDName="Contactperson" Where="UID" Is="2003121817293296325874" />
         <MasterProperty IDName="Signature1" Where="UID" Is="2003121817293296325874" />
         <MasterProperty IDName="Signature2" Where="UID" Is="2003121817293296325874" />
         <MasterProperty IDName="Recipient">
            <Field Name="CompleteAddress" Value="::ERP_invoice_address::"/>
         </MasterProperty>
         <MasterProperty IDName="CustomField">
            <Field Name="DocumentType" Value="::ERP_translation.invoice::"/>
            <Field Name="YourReference" Value="::ERP_invoice_ref::"/>
            <Field Name="Account" Value="::ERP_account_displayname::"/>
            <Field Name="Project" Value="::ERP_project_displayname::"/>
         </MasterProperty>
      </MasterProperties>
      <Bookmarks>
         <Bookmark Name="Subject" Value="::ERP_translation.no.:: ::ERP_invoice_number::"/>
      </Bookmarks>
      <Contents>
         <Content ID = "Invoice Summary Title"></Content>
 ::LOOP IncidentTotals::
         <Content ID = "Invoice Summary Item">
            <Value Name="GroupBy_incident_type" Value="::ERP_GroupBy_incident_type::"/>
            <Value Name="Sum_total" Value="::ERP_Sum_total::"/>
            <Value Name="Sum_vat" Value="::ERP_Sum_vat::"/>
         </Content>
 ::END LOOP IncidentTotals::
 ::LOOP InvoiceTotals::
         <Content ID = "Invoice Summary Total">
            <Value Name="Sum_total_incl_vat" Value="::ERP_Sum_total_incl_vat::"/>
            <Value Name="Sum_total_vat_relevant" Value="::ERP_Sum_total_vat_relevant::"/>
            <Value Name="Sum_total_vat_irrelevant" Value="::ERP_Sum_total_vat_irrelevant::"/>
            <Value Name="Sum_vat" Value="::ERP_Sum_vat::"/>
         </Content>
 ::END LOOP InvoiceTotals::
         <Content ID = "Invoice Accounting Details"></Content>
      </Contents>
      <Output>
         <Print UID="2007010510574768320716" ShowDialog="0" />
      </Output>
      <CloseDocument>-1</CloseDocument>
   </CreateDocument>
</Parameters>
```

Please note that we recommend using «::» as identifier for business application placeholders. Make sure you have removed all placeholders in the final OSC file that gets passed on to officeatwork.

Sample 2:

This sample creates two different documents. The first is a summary document used in the accounting department. The second is a document concerning the same data but offers more details. It is printed for internal use as well as an original to be sent to the customer. In the end the documents are closed without saving. The whole process is fully automated and requires no interaction from the user.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Parameters>

::LOOP Documents::
   <CreateDocument>
      <TemplateID>Letter</TemplateID>
      <ShowDocumentWizard>0</ShowDocumentWizard>
      <Language>::account_lcid::</Language>
      <MasterProperties>
         <MasterProperty IDName="Company" Where="IDName" Is="::officeatwork_organisation_name::"/>
         <MasterProperty IDName="Contactperson" Where="UID" Is="2003121817293296325874" />
         <MasterProperty IDName="Signature1" Where="UID" Is="2003121817293296325874" />
         <MasterProperty IDName="Signature2" Where="UID" Is="2003121817293296325874" />
         <MasterProperty IDName="Recipient">
            <Field Name="CompleteAddress" Value="::invoice_address::"/>
         </MasterProperty>
         <MasterProperty IDName="CustomField">
            <Field Name="DocumentType" Value="::translation.invoice::"/>
            <Field Name="YourReference" Value="::invoice_ref::"/>
            <Field Name="Account" Value="::account_displayname::"/>
            <Field Name="Project" Value="::project_displayname::"/>
         </MasterProperty>
      </MasterProperties>
      <Bookmarks>
         <Bookmark Name="Subject" Value="::translation.no.:: ::invoice_number::"/>
      </Bookmarks>
      <Contents>
         <Content ID = "Invoice Summary Title"></Content>
::LOOP IncidentTotals::
         <Content ID = "Invoice Summary Item">
            <Value Name="GroupBy_incident_type" Value="::GroupBy_incident_type::"/>
            <Value Name="Sum_total" Value="::Sum_total::"/>
            <Value Name="Sum_vat" Value="::Sum_vat::"/>
         </Content>
::END LOOP IncidentTotals::
::LOOP InvoiceTotals::
         <Content ID = "Invoice Summary Total">
            <Value Name="Sum_total_incl_vat" Value="::Sum_total_incl_vat::"/>
            <Value Name="Sum_total_vat_relevant" Value="::Sum_total_vat_relevant::"/>
            <Value Name="Sum_total_vat_irrelevant" Value="::Sum_total_vat_irrelevant::"/>
            <Value Name="Sum_vat" Value="::Sum_vat::"/>
         </Content>
::END LOOP InvoiceTotals::
         <Content ID = "Invoice Accounting Details"></Content>
      </Contents>
      <Output>
         <Print UID="2007010510574768320716" ShowDialog="0" />
      </Output>
      <CloseDocument>-1</CloseDocument>
   </CreateDocument>

   <CreateDocument>
      <TemplateID>Letter</TemplateID>
      <ShowDocumentWizard>0</ShowDocumentWizard>
      <Language>::account_lcid::</Language>
      <MasterProperties>
         <MasterProperty IDName="Company" Where="IDName" Is="::officeatwork_organisation_name::"/>
         <MasterProperty IDName="Contactperson" Where="UID" Is="2003121817293296325874" />
         <MasterProperty IDName="Signature1" Where="UID" Is="2003121817293296325874" />
         <MasterProperty IDName="Signature2" Where="UID" Is="2003121817293296325874" />
         <MasterProperty IDName="Recipient">
            <Field Name="CompleteAddress" Value="::invoice_address::"/>
         </MasterProperty>
         <MasterProperty IDName="CustomField">
            <Field Name="DocumentType" Value="::translation.Invoice::"/>
            <Field Name="YourReference" Value="::invoice_ref::"/>
            <Field Name="Account" Value="::account_displayname::"/>
            <Field Name="Project" Value="::project_displayname::"/>
         </MasterProperty>
      </MasterProperties>
      <Bookmarks>
         <Bookmark Name="Subject" Value="::translation.no.:: ::invoice_number::"/>
      </Bookmarks>
      <Contents>
         <Content ID = "Invoice Summary Title"></Content>
::LOOP IncidentTotals::
         <Content ID = "Invoice Summary Item">
            <Value Name="GroupBy_incident_type" Value="::GroupBy_incident_type::"/>
            <Value Name="Sum_total" Value="::Sum_total::"/>
            <Value Name="Sum_vat" Value="::Sum_vat::"/>
         </Content>
            ::END LOOP IncidentTotals::

               ::LOOP InvoiceTotals::
            <Content ID = "Invoice Summary Total">
               <Value Name="Sum_total_incl_vat" Value="::Sum_total_incl_vat::"/>
               <Value Name="Sum_total_vat_relevant" Value="::Sum_total_vat_relevant::"/>
               <Value Name="Sum_total_vat_irrelevant" Value="::Sum_total_vat_irrelevant::"/>
               <Value Name="Sum_vat" Value="::Sum_vat::"/>
            </Content>
            ::END LOOP InvoiceTotals::

            <Content ID = "Invoice Payment Instructions"></Content>
            <Content ID = "Invoice Marketing"></Content>
            <Content ID = "Invoice Journal Title"></Content>
            ::LOOP Journal::
          <Content ID = "Invoice Journal Subtitle">
             <Value Name="incident_type" Value="::incident_type::"/>
          </Content>

             ::LOOP JournalItems::
             <Content ID = "Invoice Journal Item">
                <Value Name="accounting_date" Value="::accounting_date::"/>
                <Value Name="subject" Value="::subject::"/>
                <Value Name="unit_price" Value="::unit_price::"/>
                <Value Name="quantity" Value="::quantity::"/>
                <Value Name="unit" Value="::unit::"/>
                <Value Name="total_without_discount" Value="::total_without_discount::"/>
                <Value Name="discount_pct" Value="::discount_pct::"/>
                <Value Name="discount_description" Value="::discount_description::"/>
                <Value Name="discount" Value="::discount::"/>
                <Value Name="total" Value="::total::"/>
                <Value Name="vat" Value="::vat::"/>
             </Content>
             ::END LOOP JournalItems::



          ::END LOOP Journal::

       </Contents>

       <Output>
          <Print UID="2007010510574768320716" ShowDialog="0" />
          <Print UID="2004543664767678679898" ShowDialog="0" />
       </Output>
       <CloseDocument>-1</CloseDocument>

    </CreateDocument>

::END LOOP Documents::

</Parameters>
```

CHAPTER 5

officeatwork «ExecuteXML» Method

Introduction

The officeatwork **ExecuteXML** method is the main API function that officeatwork offers. Its XML-Parameter allows you to interact with miscellaneous officeatwork functions.

When calling the officeatwork **ExecuteXML** method, remember to include the officeatwork ActiveX API-Component named **officeatwork API** in your project. Otherwise the methods are not available to you. The component is located in the **Common Folder** within your officeatwork application directory. The filename of the component is **oawAPI.exe**

Syntax

ExecuteXML (pParam : String) : Long

Parameters

The function **ExecuteXML** has the following parameters:

Name	**Description**
pParam	String. An XML string containing officeatwork specific instructions. The string has to be structured in a predefined format.

Return value

The **ExecuteXML** function returns the following values:

Type	**Description**
Long	can have one of the following values: **-1**, representing a successful execution – this means all instructions defined in the parameter were successfully executed. **0**, representing the value of an unsuccessful execution –

this means one or many instructions were not successfully executed.

CHAPTER 6

XML Parameter structure and conventions

Introduction

The XML parameter conforms with the following XML structure/convention.

Root Elements

There are two root elements available to you. The «CreateDocument» allows you to create a new document. All its sub-elements (instruction elements) describe how to specifically create a document. If you want to create multiple documents you just add multiple «CreateDocument» root elements to your XML. The second root element is «EditDocument». As the name already suggests, this will allow you to edit an existing document. Again all sub-elements describe in detail what to edit.

CreateDocument

Syntax

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

This element contains sub-elements.

CreateDocument Sample

This sample will create a new document based on the **letter** template.

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

EditDocument

Syntax

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

This element contains sub-elements.

EditDocument Sample

This sample will open an existing document and change its subject to «This is my new subject».

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Parameters>
   <EditDocument>
      <DocumentFullName>%Documents%\ExampleLetter.doc</DocumentFullName>
      <Bookmarks>
         <Bookmark Name="Subject" Value="This is my new subject"/>
      </Bookmarks>
   </EditDocument>
</Parameters>
```

Instruction Elements

The following instruction elements are available:

- DocumentFullName
- ProtectionType
- Password
- Language
- TemplateID
- ShowDocumentWizard
- ShowCustomDialog
- Contents
- Profile
- MasterProperties
- DocumentVariables
- CustomDocumentProperty
- BuiltInDocumentProperties
- Bookmarks
- Output

- SaveAsLocation
- CloseDocument
- ReplaceExisting

DocumentFullName

Allows you to define the name and path of an existing document to edit.

This element only works within an EditDocument root element. If this tag is missing within the EditDocument root element, the complete EditDocument element within your XML parameter will be ignored.

Syntax

```
<DocumentFullName> </DocumentFullName>
```

Content

The content complete file and path name of an existing document.

Sample

```
<DocumentFullName>%Documents%\Report2006.doc</DocumentFullName>
```

ProtectionType

Allows you to set a protection of a specific type for your document. This element is optional.

Syntax

```
<ProtectionType> </ProtectionType>
```

Content

Value representing the type of protection you want to apply to your document. Currently two ProtectionType values are allowed. **-1** stands for no protection. **2** stands for forms protection.

Sample Language Tag

```
<ProtectionType>2</ProtectionType>
```

Password

Allows you to set a password for your document. The document will not be protected only by setting a password. You also need to set a protection type to activate the document protection for your document. The password element is optional. If you include the ProtectionType element but omit the Password element, the document will be protected with no (empty) password.

Syntax

```
<Password> </Password>
```

Content

The password you want to apply to your document.

Sample

```
<Password>myPassword</Password>
```

Language

Allows you to set the document language.

Syntax

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

Content

Language ID (LCID). You can find a list of available Language ID's in the Appendix of this manual.

Sample

```
<Language>2055</Language>
```

TemplateID

Allows you to define the template that should be used to create a new document.

This element only works within the «CreateDocument» root element.

Syntax

```
<TemplateID> </TemplateID>
```

Content

A file name of a template. It is also possible to add the path to the file name. Without a path the first template with the defined file name that is found within the available template roots will be used to create a new document. If the content is left empty, the user will be prompted to choose a template manually with the Template Chooser dialogue.

Sample

```
<TemplateID>letter</TemplateID>
```

ShowDocumentWizard

Allows you to define if the document wizard will automatically be completed during the document creation process or if it shall halt during that process and present the wizard to the user for manual completion.

This element only works within a CreatDocument root element.

Syntax

```
<ShowDocumentWizard> </ShowDocumentWizard>
```

Content

String. If the value past equals **-1,** the wizard will not be completed automatically. Instead it will be presented to the user for manual completion. If the value past equals **0,** the wizard will automatically run through without user interaction. All the users default settings apply except automatically set values by the XML Parameter.

Default behaviour

If the ShowDocumentWizard is either missing a value or is completely missing within your XML Parameter, the default behaviour during the creation of a new document will present the document wizard to the user for manual completion.

Sample

```
<ShowDocumentWizard>-1</ShowDocumentWizard>
```

ShowCustomDialog

Allows you to define if the custom dialogue defined in a Smart Template will automatically be completed during the document creation process or if it shall halt during that process and present the dialogue to the user for manual completion.

Syntax

```
<ShowCustomDialog> </ShowCustomDialog>
```

Content

String. If the value past equals **-1,** the dialogue will not be completed automatically. Instead it will be presented to the user for manual completion. If the value past equals **0,** the dialogue will automatically run through without user interaction. All the default settings apply except automatically set values by the XML Parameter.

Default behaviour

If the ShowCustomDialog is either missing a value or is completely missing within your XML Parameter, the default behaviour during a creation of a new document will present the custom dialogue to the user for manual completion.

Sample

```
<ShowCustomDialog>-1</ShowCustomDialog>
```

Contents

Allows you to add Smart-Contents to your document. You can choose between adding single contents or all contents in a specific folder. You can also pass values along with your contents.

Syntax

```
<Contents>
  <Content ID="">
    <Value Name=" " Value=" "/>
  </Content>
  <ContentFolder ID="" />
</Contents>
```

Content

The «Contents» element can contain the following sub-elements.

Name	**Description**
Content	Element. This element represents an individual content.
ContentFolder	Element. This element represents a folder most probably containing numerous Smart Content files.

The «Content» element has the following attributes and sub-elements.

Name	**Description**

ID	String. The filename of the Smart Content.
LCID	String. Language ID. If omitted the Language ID of the destination document will apply.
Value	Element. This element contains a single name-value pair. If the associated content contains an appropriate value document function, the value of the value Element will be placed at the position of the document function in the process of inserting the content into the target document.

The «Value» element has the following attributes.

Name	**Description**
Name	String. Name of the value. Used to match a value document function name.
Value	Element. The actual value to be passed to the smart-content.

The «ContentFolder» element has the following attributes.

Name	**Description**
ID	String. The name (without path) of the folder containing Smart Contents.

Sample 1

```
<Contents>
   <Content ID = "01 Introduction">
      <Value Name="Object" Value="Three trees"/>
   </Content>
   <Content ID="01 Firmenbildung"></Content>
   <Content ID="02 Handelsregister"></Content>
</Contents>
```

Sample 2

```
<Contents>
   <ContentFolder ID="User Guide"/>
</Contents>
```

Profile

Allows you to define and to set a profile for the document.

Syntax

```
<Profile Where="" Is=""/>
```

Content

The Profile element has the following attributes:

Value	Description
Where	String. The field to be used to match the comparison value passed in the Is attribute.
Is	String. The value to be compared in the field defined in the Where attribute.

Default behaviour

If the Profile attributes are missing or the Profile element is omitted in your XML Parameter, the default behaviour during a creation of a new document will use the user's default profile during the process of creating a document. If the user has no default profile, the first profile will be used. If the user has no profile, no profile will be used.

Sample Profile tag

```
<Profile Where="IDName" Is="Miller Daniela"/>
```

MasterProperties

Allows you to set officeatwork specific Master Properties in your document.

Syntax

```
<MasterProperties>
  <MasterProperty IDName="" Where="" Is="" />
    <Field Name="" Value=""/>
  </MasterProperty>
</MasterProperties>
```

Content

The «MasterProperties» element has the following sub-elements.

Name	Description
MasterProperty	Element. This element represents an individual officeatwork Master Property.

The «MasterProperty» element has the following attributes and sub-elements.

Name	Description
IDName	String. The IDName of the Master Property.
Where	String. The field to be used to match the comparison value passed in the Is attribute. If omitted the default Master Property entry is selected.
Is	String. The value to be compared in the field defined in the Where attribute. If omitted the default Master Property entry is selected.
Field	Element. This element represents a single Master Property field.

The «Field» element has the following attributes.

Name	Description
Name	String. Name of the Master Property Field.
Value	String. The actual value to be passed to the Master Property field.

Sample

```
<MasterProperties>
   <MasterProperty IDName="Organisation" Where="IDName" Is="Zürich" />
   <MasterProperty IDName="Author" Where="IDName" Is="Muster Peter" />
   <MasterProperty IDName="Initiator" Where="IDName" Is="Muster Peter" />
   <MasterProperty IDName="Contactperson" Where="IDName" Is="Muster Peter" />
   <MasterProperty IDName="Signature1" Where="IDName" Is="Muster Peter" />
   <MasterProperty IDName="Signature2" Where="UID" Is="2003121817293296325874" />
   <MasterProperty IDName="Recipient">
      <Field Name="CompleteAddress" Value="Muster AG
                                    Marketing
                                    Frau Jolanda Muster
                                    Matten 12
                                    1111 Musterlingen"/>
      <Field Name="Company" Value="Muster AG"/>
      <Field Name="Department" Value="Marketing"/>
      <Field Name="FullName" Value="Frau Jolanda Muster"/>
      <Field Name="AddressStreet" Value="Matten 12"/>
      <Field Name="AddressZIP" Value="1111"/>
      <Field Name="AdressCity" Value="Musterlingen"/>
      <Field Name="Telephone" Value="+41 (0)44 444 4444"/>
      <Field Name="EMail" Value="jolanda.testerli@muster.com"/>
      <Field Name="Introduction" Value="Sehr geehrte Frau Muster"/>
      <Field Name="Closing" Value="Freundliche Grüsse"/>
   </MasterProperty>
</MasterProperties>
```

DocumentVariables

Allows you to set Word document variables. If the document variable does not exist, it will be created by officeatwork.

Syntax

```
<DocumentVariables>
  <DocumentVariable Name="" Value=""/>
</DocumentVariables>"
```

Content

The «DocumentVariables» element has the following sub-elements.

Name	Description
DocumentVariable	Element. This element represents an individual Word document variable.

The « DocumentVariable» element has the following attributes.

Name	Description
Name	String. The name of the Word document variable.
Value	String. The value to be allocated to the Word document variable

Sample

```
<DocumentVariables>
   <DocumentVariable Name="MeetingDate" Value="12.03.2004"/>" & _
   <DocumentVariable Name="MeetingLocation" Value="Room C324"/>" & _
</DocumentVariables>
```

CustomDocumentProperties

Allows you to set Word custom document properties. If the custom document property does not exist, it will be created by officeatwork.

Syntax

```
<CustomDocumentProperties>
  <CustomDocumentProperty Name="" Value=""/>
</CustomDocumentProperties>
```

Content

The «CustomDocumentProperties» element has the following sub-elements.

Name	Description
CustomDocumentProperty	Element. This element represents an individual Word custom document property.

The «CustomDocumentProperty» element has the following attributes.

Name	**Description**
Name	String. The name of the Word custom document property.
Value	String. The value to be allocated to the Word custom document property.

Sample

```
<CustomDocumentProperties>
   <CustomDocumentProperty Name="Price_Karton_SX12_H4" Value="23.00"/>
   <CustomDocumentProperty Name="Price_Tube_L3" Value="12.50"/>"
</CustomDocumentProperties>
```

BuiltInDocumentProperties

Allows you to set the value of Word built-in document properties.

Syntax

```
<BuiltInDocumentProperties>
   <BuiltInDocumentProperty Name="" Value=""/>
</BuiltInDocumentProperties>
```

Content

The «BuiltInDocumentProperties» element has the following sub-elements.

Name	**Description**
BuiltInDocumentProperty	Element. This element represents an individual Word built-in document property.

The « BuiltInDocumentProperty» element has the following attributes.

Name	**Description**
Name	String. The name of the Word built-in document property.
Value	String. The value to be allocated to the Word built-in document property.

List of modifiable built-in document properties

- Title
- Subject

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

Sample

```
<BuiltInDocumentProperties>
   <BuiltInDocumentProperty Name="Author" Value="Harry Smith"/>
   <BuiltInDocumentProperty Name="Subject" Value="Offer B763Gk"/>
</BuiltInDocumentProperties>
```

Bookmarks

Allows you to set the content of Word bookmarks.

Syntax

```
<Bookmarks>
   <Bookmark Name="" Value=""/>
</Bookmarks>
```

Content

The «Bookmarks» element has the following sub-elements.

Name	Description
Bookmark	Element. This element represents an individual Word bookmark.

The «Bookmark» element has the following attributes.

Name	Description
Name	String. The name of the Word bookmark.
Value	Element. The value to be allocated to the Word bookmark.

Sample

```
<Bookmarks>
   <Bookmark Name="size" Value="347.5"/>
   <Bookmark Name="orientation" Value="tall"/>"
</Bookmarks>
```

ServerProperties

Allows you to set the content of ServerProperties in Office 2007 Documents.

Syntax

```
<ServerProperties>
  <ServerProperty Name="" Value=""/>
</ServerProperties>
```

Content

The «ServerProperties» element has the following sub-elements.

Name	Description
ServerProperty	Element. This element represents an individual server property.

The «Serverproperty» element has the following attributes.

Name	Description
Name	String. The name of the server property.
Value	Element. The value to be allocated to the server property.

Sample

```
<ServerProperties>
   <ServerProperty Name="DocumentType" Value="Invitation"/>
   <ServerProperty Name="Segment" Value="Finance"/>"
</ServerProperties>
```

Output

Allows you to use one of the officeatwork output variants.

Syntax

```
<Output>
  <Print UID="" ShowDialog=""/>
  <Send UID="" ShowEmail="" FileName=""/>
  <Save UID="" ShowDialog="" Path="" FileName=""
ReplaceExisting=""/>
</Output>
```

Content

The «Output» element has the following sub-elements.

Name	**Description**
Print	Element. This element triggers an officeatwork print variation to be executed.
Send	Element. This element triggers an officeatwork send variation to be executed.
Save	Element. This element triggers an officeatwork save variation to be executed.

The «Print» element has the following attributes.

Name	**Description**
UID	String. Represents the UID of the desired officeatwork print variant.
ShowDialog	Element. Indicates if the print dialogue should be presented to the user during the print process. The value **-1** will show the dialogue, while the value **0** will suppress the dialogue.

The «Send» element has the following attributes.

Name	**Description**
UID	String. Represents the UID of the desired officeatwork send variant.
ShowEmail	String. Indicates if the e-mail should be presented to the user during the send process. The value **-1** will show the dialogue, while the value **0** will not show the e-mail.

The «Save» element has the following attributes.

Name	**Description**
UID	String. Represents the UID of the desired officeatwork save variant.
ShowDialog	String. Indicates if the save dialogue should be presented to the user during the save process. The value **-1** will show the dialog, while the value **0** will suppress the dialogue.
Path	String. Path for the file to be saved to.
FileName	String. File name to be used for the new file.

ReplaceExisting	String. You can define if an existing file shall be replaced with the new one. The value **-1** will replace any existing file, while the value **0** will not save the new file if a file already exists at the same destination location.

Sample 1

```
<Output>
   <Print UID="647564367536457 56" ShowDialog="-1"/>
</Output>
```

Sample 2

```
<Output>
   <Send UID="576867876875676546" ShowEmail="-1" FileName="Report.pdf"/>
</Output>
```

Sample 3

```
<Output>
   <Save UID="3453456857465475467 75" ShowDialog="-1" Path="C:\Reports\" FileName="Report.pdf"
ReplaceExisting="-1"/>
</Output>
```

SaveAsLocation

Allows you to define the location where an active document shall be saved.

Syntax

`<SaveAsLocation> </SaveAsLocation>`

Content

The content of the SaveAsLocation element is a path and file name.

Sample

```
<SaveAsLocation>%DESKTOP%\Invoice 5968463.doc</SaveAsLocation>
```

CloseDocument

Allows you to define if the active document shall be closed or left open.

Syntax

`<CloseDocument> </CloseDocument>`

Content

String. If the value past equals **-1,** the document will be closed. If the value past equals **0,** the document will remain open.

Default behaviour

If the CloseDocument tag is missing a value or is completely omitted from your XML Parameter, the active document will remain open.

Sample

```
<ShowCustomDialog>-1</ShowCustomDialog>
```

ReplaceExisting

Allows you to define, when the document is closed and saved, whether it shall replace any existing document with the same name and path.

Syntax

```
<ReplaceExisting> </ReplaceExisting>
```

Content

String. If the value past equals **-1,** any existing document will be overwritten. String. If the value past equals **0,** the active document will not be saved when it is closed, if already another document exists with the same filename at the same location.

Default behaviour

If the ReplaceExisting tag is missing a value or is completely omitted from your XML Parameter, the active document will not be saved if a file with the same filename and location already exists.

Sample

```
<ReplaceExisting>-1</ReplaceExisting>
```

Values

Allows you to set values for the appropriate Values() document functions within your document.

Syntax

```
<Values>
  <Value Name="" Value=""/>
</Values>
```

Content

The «Values» element has the following sub-elements.

Name	Description
Value	Element. This element represents an individual Value.

The «Value» element has the following attributes.

Name	Description
Name	String. The name corresponding to the name used in the Value() document function.
Value	Element. The value to be allocated to the appropriate Value() document function.

Sample

```
<Values>
   <Value Name="size" Value="347.5"/>
   <Value Name="orientation" Value="tall"/>"
</Values>
```

VBA Sample

The following example is written in VBA and creates a new document based on the letter template:

```
Sub OfficeatworkApiSample()

    Dim lOawAPI As oawAPI.API
    Dim lParam As String

    lParam = "<Parameters>" & _
                "<CreateDocument>"


    lParam = lParam & _
                    "<Language>" & _
                        "2055" & _
                    "</Language>" & _
                    "<TemplateID>" & _
                        "Letter" & _
                    "</TemplateID>" & _

    lParam = lParam & _
                    "<SaveAsLocation>" & _
                        "C:\MyDocuments\Document002.doc" & _
                    "</SaveAsLocation>" & _
                    "<ReplaceExisting>" & _
                        "-1" & _
                    "</ReplaceExisting>" & _
                    "<CloseDocument>" & _
                        "0" & _
                    "</CloseDocument>

    lParam = lParam & _
                    "<Contents>" & _
                        "<Content ID=""Karton_SX12_H4""/>" & _
                        "<Content ID=""Schlauch_L3"">" & _
                            "<Value Name=""PositionNr"" Value=""1""/>" & _
                            "<Value Name=""Description"" Value=""Schlauch mit T-Anschluss""/>" & _
                            "<Value Name=""Price"" Value=""31.70""/>" & _
                        "</Content>" & _
                        "<Content ID=""Karton_SX12_H4"" LCID=""2057""/>" & _
                        "<Content ID=""Titel"" Bookmark=""Text"" InsertionMethod=""0""/>" & _
                    "</Contents>"

    lParam = lParam & _
                    "<Profile Where=""IDName"" Is=""Standard""/>" & _
                    "<MasterProperties>" & _
                        "<MasterProperty IDName=""Recipient"">" & _
                            "<Field Name=""Company"" Value=""Muster AG""/>" & _
                            "<Field Name=""Department"" Value=""Marketing""/>" & _
                            "<Field Name=""FullName"" Value=""Frau Jolanda Testerli""/>" & _
                            "<Field Name=""AddressStreet"" Value=""Matten 12""/>" & _
                            "<Field Name=""AddressZIP"" Value=""1111""/>" & _
                            "<Field Name=""AddressCity"" Value=""Musterfingen""/>" & _
                            "<Field Name=""Telephone"" Value=""+41 (0)44 444 4444""/>" & _
                            "<Field Name=""EMail"" Value=""jolanda.testerli@muster.com""/>" & _
                            "<Field Name=""Introduction"" Value=""Sehr geehrte Frau Testerli""/>" & _
                            "<Field Name=""Closing"" Value=""Freundliche Grüsse""/>" & _
                        "</MasterProperty>" & _
                    "</MasterProperties>"

    lParam = lParam & _
                    "<DocumentVariables>" & _
                        "<DocumentVariable Name=""MeetingDate"" Value=""12.03.2004""/>" & _
                        "<DocumentVariable Name=""MeetingPlace"" Value=""Room C324""/>" & _
                    "</DocumentVariables>" & _
                    "<CustomDocumentProperties>" & _
                        "<CustomDocumentProperty Name=""Preis_Karton_SX12_H4"" Value=""23.00""/>" & _
                        "<CustomDocumentProperty Name=""Preis_Schlauch_L3"" Value=""12.50""/>" & _
                    "</CustomDocumentProperties>" & _
                    "<BuiltInDocumentProperties>" & _
                        "<BuiltInDocumentProperty Name=""Author"" Value=""Peter Muster""/>" & _
                        "<BuiltInDocumentProperty Name=""Subject"" Value=""Offerte Gartenbau""/>" & _
                    "</BuiltInDocumentProperties>" & _
                    "<Bookmarks>" & _
                        "<Bookmark Name=""Preis_Schraube_3d6"" Value=""0.25""/>" & _
                        "<Bookmark Name=""Preis_Mutter_3d5"" Value=""0.10""/>" & _
                    "</Bookmarks>"

    lParam = lParam & _
                "</CreateDocument>" & _
            "</Parameters>"
    Set lOawAPI = New oawAPI.API

    If (lOawAPI.ExecuteXML(lParam) = -1) Then
       MsgBox "successfully created document"
   Else
       MsgBox "the document could not be created"
    End If

    Set lOawAPI = Nothing

End Sub
```

CHAPTER 7

officeatwork «TemplateChooser» Method

Introduction

The officeatwork **TemplateChooser** method is an officeatwork API function that allows you to use the officeatwork Template Chooser. When called, it will present the Template Chooser to the user and will ask him to pick a template. Depending on the user's rights, the selection of the templates could be limited. The return value contains the corresponding TemplateID of the template that the user chose.

When calling the officeatwork **TemplateChooser** method, remember to include the officeatwork ActiveX API-Component named **officeatwork API** in your project. Otherwise the methods are not available to you. The component is located in the **Common Folder** within your officeatwork application directory. The filename of the component is «oawAPI.exe»

Syntax

TemplateChooser (pParam: String) : String

Sample

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

Parameter

The function `TemplateChooser` has the following parameters:

Name	**Description**
pParam	String. An XML string containing officeatwork specific instructions. The string has to be structured in a predefined format.

Root elements

The XML parameter has no root elements.

Return value

The `TemplateChooser` function returns the following values:

Type	Description
String	An XML string with various elements within the **<Results>** element.

Content

The <Results> element has the following sub-elements

Name	Description
Successful	Indicates whether the function has been successfully completed. **-1** successful **0** failed Sample: `<Successful>-1</Successful>`
Solution	Returns the ID of the currently active officeatwork solution. Sample: `<Solution>examplecom</Solution>`
TemplateID	Returns the ID of the chosen template. Sample: `<TemplateID>Letter</TemplateID>`
TemplatePath	Returns the path of the chosen template. Sample: `<TemplatePath>C:\myTemplates\</TemplatePath>`
TemplateFilename	Returns the filename of the chosen template: Sample: `<TemplateFilename>Letter.owt</TemplateFilename>`

TemplateFullname	Returns the filename including the path of the chosen template.
	Sample:
	`<TemplateFullname>`
	`C:\officeatwork\Solutions\MySolution\`
	`MasterTemplates\Letter.owt`
	`</TemplateFullname>`

Sample

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Results>
   <Successful>-1</Successful>
   <Solution>examplesolutioncom</Solution>
   <TemplateID>Letter</TemplateID>
   <TemplatePath> C:\officeatwork\Solutions\MySolution\MasterTemplates\</TemplatePath>
   <TemplateFilename>Letter.owt</TemplateFilename>
   <TemplateFullname>C:\officeatwork\Solutions\MySolution\MasterTemplates\Letter.owt</TemplateFullname>
</Results>
```

Appendix

Document Functions

ComposeHtmlSource()

Available

The ComposeHtmlSource() function is available in the following Templates and Contents:

- Signature Templates

Description

The ComposeHtmlSource() function is mainly used to include a picture into an HTML signature template. When executed, it will copy the picture file defined in the «Source path & file» parameter to the destination location defined in the «Destination path & file» parameter. Optionally, you can define a new name for the picture file copied to the destination location.

Outlook signatures: Please make sure you also include a folder that is created for your picture with the exact same name and location of your signature file, and subsequently place your picture in this folder. This is necessary as otherwise Outlook will not find the picture you have specified in your signature template.

Syntax

```
[[ComposeHtmlSource(Source path and file, Destination
path and file, Overwrite existing file)]]
```

Parameter

Source path & file	defines the file to copy and its location of origin
Destination path & file	defines the location and name of the copied file. If you do not include a file name, the name of the source file will be used for the file name at the destination location.
Overwrite existing file	Boolean, indicating whether to overwrite existing files at destination. **True** will overwrite existing files. **False** will not copy the file from the source if a file at the destination location already exists with the same name.

Example

```
[[ComposeHtmlSource(GetMasterPropertyValue("Organisation", "LogoSignature"), ".\" & AutoSignatureName &
"_files\logo.jpg", true)]]
```

Concat()

Available

The Concat() function is available in the following Templates and Contents:

- Excel Templates
- Signature Templates

Description

The Concat() function allows you to combine string elements.

Syntax

```
[[Concat(Text1, Text2, ...)]]
```

Parameter

Text	A text element.

Example

```
[[Concat(Translate("Doc.Phone"), ": ", GetMasterPropertyValue("Company", "EmailAddress2"))]]
```

GetMasterPropertyValue()

Available

The GetMasterPropertyValue() function is available in the following Templates and Contents:

- Excel Templates
- Signature Templates

Description

The IF() function is used to create conditional content. Should for instance the Translation «Doc.Phone» only be visible if the contact person has a phone assigned to him, then the IF() function can be used to specifiy that condition.

Syntax

Parameter

Test	Logical test like A=B or 1>2 or GetMasterPropertyValue("Company", "Email")=""
TrueValue	Return value if Test equals True
FalseValue	Return value if Test equals False

Example

```
[[IF (GetMasterPropertyValue("Company", "EmailAddress2")="", "", GetMasterPropertyValue("Company",
"EmailAddress2"))]]
```

If()

Available

The If() function is available in the following Templates and Contents:

- Excel Templates
- Signature Templates

Description

The IF() function is used to create conditional content. Should for instance the Translation «Doc.Phone» only be visible if the contanct person has a phone assigned to him, then the IF() function can be used to specifiy that condition.

Syntax

```
[[If(Test, TrueValue, FalseValue)]]
```

Parameter

Test	Logical test like A=B or 1>2 or GetMasterPropertyValue("Company", "Email")=""
TrueValue	Return value if Test equals True
FalseValue	Return value if Test equals False

Example

```
IF (GetMasterPropertyValue("Company", "EmailAddress2")="", "", GetMasterPropertyValue("Company",
"EmailAddress2"))
```

InsertBreak()

Available

The InsertBreak() function is available in the following Templates and Contents:

- Word SmartTemplate
- Word SmartContent

Description

This function implements a change of column, page, chapter, etc.

Syntax

```
[[InsertBreak ( Type : String, [ProtectionType : String]
)]]
```

Parameters

The function InsertBreak consists of the following parameters:

Name (Type)	**Description**
Type (String)	Describes the style of change. List of possible parameters - ColumnBreak - LineBreak - PageBreak - SectionBreakContinuous - SectionBreakEvenPage - SectionBreakNextPage - SectionBreakOddPage
ProtectionType (String)	Only applicable with SectionBreak types: Describes the kind of protection setting applicable. Is the Protection Type omitted then the section will be inserted without any protection. List of possible parameters: - Unprotected - Protected

Example 1

```
[[InsertBreak(PageBreak)]]
```

Example 2

```
[[InsertBreak(SectionBreakNextPage)]]
```

Example 3

```
[[InsertBreak(SectionBreakNextPage, Protected)]]
```

InsertContent()

Available

The Avaliable() function is available in the following Templates and Contents:

- Word SmartTemplate
- Word SmartContent

Description

This function inserts contents which are referred to by their names. The insertion is executed by an external interface. This happens while inserting contents or during the addition of Smart-Templates.

Syntax

```
[[InsertContent ( Name : String, [LCID : String])]]
```

Parameters

The function InsertContent consists of the following parameters:

Name (Type)	**Description**
Name (String)	Name of the content which is to be inserted. This can be executed in three different ways: - Alternative 1: Just the file name: officeatwork searches for the content in the current template solution and inserts it. - Alternative 2: Complete path with file name: The indicated content is inserted. - Alternative 3: System variable or officeatwork shortcut with offset path and file name: officeatwork clears the system variable or the officeatwork shortcut and inserts the content.

LCID (String)	Optional. LCID, the language in which the content is to be inserted, as long as it differs from the chosen document language and is defined as fixed language.

A list of the LCIDs is recorded in the appendix.

Example 1

```
T&C Article 1: [[InsertContent("T&cArt1")]]
```

Example 2

```
T&C Article 1:
[[InsertContent("J:\Content\CopyrightNote.owc")]]
```

Example 3

```
T&C Article 1:
[[InsertContent("%Documents%\ISOCertificate.owc")]]
```

Example 4

```
T&C Article 1: [[InsertContent("T&cArt1","2057")]]
```

Number()

Available

The Number() function is available in the following Templates and Contents:

- Word SmartTemplate
- Word SmartContent

Description

This function formats a numeric value.

Careful: when you format a number which includes separators that designate decimal points or numbers in thousands, and these do not match the system settings of the user, it is not possible for you to use these numbers for calculations with fields in Microsoft.

Syntax

```
[[Number ( Number : String, Format : String,
[NumberDecimalSymbol : String],
[FormatDecimalSymbol : String],
[FormatDigitGroupSymbol : String] )]]
```

Parameters

The function Number consists of the following parameters:

Name (Type)	Description
Number (String)	Number which needs to be formatted.
Format (String)	Format in which the number is to be transformed. 0 represents a digit. It either shows a specific number or a zero. If there is a number from the parameter numbers in place of the 0, then this number will be shown. If no number within the parameter is available, a zero will be shown. If the number consists of more decimal places to the right of the decimal point than defined in the format, then the number will be rounded to the defined number of decimal places. #represents a figure. It either shows part of a number or nothing. If there is a number from the parameter numbers in place of the #, then this number will be shown. If no number within the parameter is available, nothing will be shown. . represents the place of the decimal point. The point is replaced by the indicated symbol in the optional parameter FormatDecimalSymbol. If the parameter FormatDecimalSymbol is not designated, then the decimal point symbol specified in the user's system settings is used. , Represents the place where separators are used to show numbers in thousands. This symbol can be found repeatedly. The comma is replaced by the symbol FormatDigitGroupSymbol in the optional parameter. If the parameter FormatDigitGroupSymbol is not indicated, then the separator to show numbers in thousands, as specified in the user's system settings, is used.
NumberDecimalSymbol (String)	Optional. Decimal separator, showing the place of the decimal point which is formatted in the parameter number. If this parameter is not designated, then the decimal point symbol specified in the user's system settings is used.
FormatDecimalSymbol (String)	Optional. Decimal separator, showing the place of the decimal point which is formatted in the parameter number. If this parameter is not designated, then the decimal point symbol specified in the user's system settings is used.
FormatDigitGroupSymbol (String)	Optional. Separator, specifying numbers in thousands, which is used in the parameter format. If the parameter

FormatDigitGroupSymbol is not designated, then the symbol separating numbers in thousands specified in the user's system settings is used.

Example 1

The following example formats the number «123456.789» into «123'456.79»:

Price. `[[Number("123456.789", "#,##0.00", ".", ".", "'")]]`

Example 2

The following example formats the number which is calculated according to the formula `[[Value("UnitPrice")]]`

Price. `[[Number("[[Value("UnitPrice")]]", "#,##0.00", ".", ".", "'")]]`

Path()

Available

The Path() function is available in the following Templates and Contents:

- Word SmartTemplate
- Word SmartContent

Description

This function changes a path which contains system variables or officeatwork Solutions shortcuts.

Careful: If this formula is used within a field, then all single backslashes «\» will automatically be replaced by double backslashes «\\» as this is the standard notation for paths on Word fields.

Syntax

```
[[Path ( Path : String )]]
```

Parameters

The function Path consists of the following parameters:

Name (Type)	**Description**
Path (String)	Path which contains a System Variable (e.g. %ProgramFiles%) or an officeatwork Solution shortcut.

Example 1

```
{ INCLUDEPICTURE "[[Path("%Images%\ProductA.jpg")]]" }
```

Example 2

```
{ INCLUDEPICTURE "[[Path("%AppData%\Signature.jpg")]]" }
```

Example 3

```
{ INCLUDEPICTURE
"[[Path("%Images%")]]\\[[Value("ProductImage",
"ImageA.jpg")]]" }
```

Translate()

Available

The Translate() function is available in the following Templates and Contents:

- Word MasterTemplate
- Word SmartTemplate
- Word SmartContent
- Excel Templates
- Signature Templates

Description

The Translate() function returns a translation of a given label.

Syntax

```
[[Translate(Label[, LCID])]]
```

Parameters

The function Translate consists of the following parameters:

Name (Type)	Description
Label (String)	Label of which the appropriate translation is to be returned.
LCID (String)	Optional. LCID, the language in which the translation should be added, as long as it differs from the chosen document language and is defined as fixed language.

A list of the LCIDs is recorded in the appendix.

Example 1

```
The positioning is [[Translate("Doc.Right")]].
```

Example 2

```
The English term is [[Translate("Doc.Right","2057")]].
```

Value()

Available

The Value() function is available in the following Templates and Contents:

- Word MasterTemplate
- Word SmartTemplate
- Word SmartContent
- Excel Templates
- Signature Templates

Description

This function inserts a value, which is referred to by its name. The value is for instance passed on as a parameter of an external interface.

Syntax

```
[[Value ( Name : String, [Default : String] )]]
```

Parameters

The function Value consists of the following parameters:

Name (Type)	**Description**
Name (String)	Name of the parameters which are provided by the interface.
Default (String)	Optional. The Value which is inserted if the value of the parameter's name is not given. This means that if the value of the parameter's name is conveyed as empty, then empty will be inserted.

Example

`Position Nr. [[Value("PositionNr", "Pos Nr.")]],`
`[[Value("Description", "Descr.")]]` to the price of
`[[Value("Price")]]`

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

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

LCID's

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

Language	ID	Language	ID
French - Cameroon	11276	Inuktitut	1117
French - Canada	3084	Italian - Italy	1040
French - Democratic Rep. of Congo	9228	Italian - Switzerland	2064
French - Cote d'Ivoire	12300	Japanese	1041
French - Haiti	15372	Kannada	1099
French - Luxembourg	5132	Kanuri - Nigeria	1137
French - Mali	13324	Kashmiri	2144
French - Monaco	6156	Kashmiri (Arabic)	1120
French - Morocco	14348	Kazakh	1087
French - North Africa	58380	Khmer	1107
French - Reunion	8204	Konkani	1111
French - Senegal	10252	Korean	1042
French - Switzerland	4108	Kyrgyz (Cyrillic)	1088
French - West Indies	7180	Lao	1108
Frisian - Netherlands	1122	Latin	1142
Fulfulde - Nigeria	1127	Latvian	1062
FYRO Macedonian	1071	Lithuanian	1063
Gaelic (Ireland)	2108	Malay - Malaysia	1086
Gaelic (Scotland)	1084	Malay - Brunei Darussalam	2110
Galician	1110	Malayalam	1100
Georgian	1079	Maltese	1082
German - Germany	1031	Manipuri	1112
German - Austria	3079	Maori - New Zealand	1153
German - Liechtenstein	5127	Marathi	1102
German - Luxembourg	4103	Mongolian (Cyrillic)	1104
German - Switzerland	2055	Mongolian (Mongolian)	2128
Greek	1032	Nepali	1121
Guarani - Paraguay	1140	Nepali - India	2145
Gujarati	1095	Norwegian (Bokmål)	1044
Hausa - Nigeria	1128	Norwegian (Nynorsk)	2068
Hawaiian - United States	1141	Oriya	1096
Hebrew	1037	Oromo	1138
Hindi	1081	Papiamentu	1145
Hungarian	1038	Pashto	1123
Ibibio - Nigeria	1129	Polish	1045
Icelandic	1039	Portuguese - Brazil	1046
Igbo - Nigeria	1136	Portuguese - Portugal	2070
Indonesian	1057	Punjabi	1094

Language	ID
Punjabi (Pakistan)	2118
Quecha - Bolivia	1131
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

Language	ID
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

API Samples

OSC File-Samples

Bookmark

This sample OSC file shows how to create a document and set the bookmark **Subject** to **my new subject**. The template used is the **letter** template and the document language is set to **English UK (2057)**.

This sample can be found in the officeatwork example solution, which is part of the officeatwork application installer. The sample file is located in the folder **Examples\OSC Files** of the officeatwork solution and is named **Sample Bookmark.osc**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Parameters>
   <CreateDocument>
      <Language>2057</Language>
      <TemplateID>Letter</TemplateID>
      <ShowDocumentWizard>0</ShowDocumentWizard>
      <Bookmarks>
         <Bookmark Name="Subject" Value="my new subject"/>
      </Bookmarks>
   </CreateDocument>
</Parameters>
```

ContentFolder

This sample OSC file shows how to create a document that includes all Smart-Contents within the **Produkteofferte** folder including its subfolders. The template used is the **offer** template and the document language is set to **German Switzerland (2055)**.

This sample can be found in the officeatwork example solution, which is part of the officeatwork application installer. The sample file is located in the folder **Examples\OSC Files** of the officeatwork solution and is named **Sample ContentFolder.osc**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Parameters>
   <CreateDocument>
      <Language>2055</Language>
      <TemplateID>Offer</TemplateID>
      <ShowDocumentWizard>0</ShowDocumentWizard>
      <Contents>
         <ContentFolder ID = "Produkteofferte"/>
      </Contents>
   </CreateDocument>
</Parameters>
```

CustomFields

This sample OSC file shows how to create a document and set values of custom fields. The template used is the **Master-Template_Custom Fields_Master Properties** template and the document language is set to **English UK (2057)**.

This sample can be found in our officeatwork example solution, which is part of the officeatwork application installer. The sample file is located in the folder **Examples\OSC Files** of the officeatwork solution and is named **Sample CustomFields.osc**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Parameters>
   <CreateDocument>
      <Language>2057</Language>
      <TemplateID>Master-Template_Custom Fields_Master Properties</TemplateID>
      <ShowDocumentWizard>-1</ShowDocumentWizard>
      <MasterProperties>
         <MasterProperty IDName="CustomField">
            <Field Name="External" Value="EX3"/>
            <Field Name="Ref" Value="K.4587.322"/>
            <Field Name="No" Value="83672"/>
         </MasterProperty>
      </MasterProperties>
   </CreateDocument>
</Parameters>
```

Letter

This sample OSC file shows how to create a document based on the **Letter** template. The document language is set to **English UK (2057)**.

This sample can be found in our officeatwork example solution, which is part of the officeatwork application installer. The sample file is located in the folder **Examples\OSC Files** of the officeatwork solution and is named **Sample Letter.osc**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Parameters>
   <CreateDocument>
      <Language>2057</Language>
      <TemplateID>Letter</TemplateID>
      <ShowDocumentWizard>-1</ShowDocumentWizard>
   </CreateDocument>
</Parameters>
```

Multiple Documents

This sample OSC file shows how to create multiple documents in one go. The template used is the **letter** template and the document language is set to **German Switzerland (2055)**.

This sample can be found in our officeatwork example solution, which is part of the officeatwork application installer. The sample file is located in the folder **Examples\OSC Files** of the officeatwork solution and is named **Sample Multiple Documents.osc**.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<Parameters>
   <CreateDocument>
      <Language>2055</Language>
      <TemplateID>Letter</TemplateID>
      <ShowDocumentWizard>0</ShowDocumentWizard>
      <SaveAsLocation>%DESKTOP%\Letter Version 1.doc</SaveAsLocation>
      <ReplaceExisting>-1</ReplaceExisting>
      <CloseDocument>-1</CloseDocument>
      <Contents>
         <Content ID = "01 Firmenbildung"></Content>
         <Content ID = "02 Handelsregister"></Content>
         <Content ID = "03 Firmenrecherche"></Content>
         <Content ID = "04 Rechtsträger"></Content>
         <Content ID = "05 Firmenschutz"></Content>
      </Contents>
   </CreateDocument>
   <CreateDocument>
      <Language>2055</Language>
      <TemplateID>Letter</TemplateID>
      <ShowDocumentWizard>0</ShowDocumentWizard>
      <SaveAsLocation>%DESKTOP%\Letter Version 2.doc</SaveAsLocation>
      <ReplaceExisting>-1</ReplaceExisting>
      <CloseDocument>-1</CloseDocument>
      <Contents>
         <Content ID = "01 Firmenbildung"></Content>
         <Content ID = "03 Firmenrecherche"></Content>
         <Content ID = "05 Firmenschutz"></Content>
      </Contents>
   </CreateDocument>
   <CreateDocument>
      <Language>2055</Language>
      <TemplateID>Letter</TemplateID>
      <ShowDocumentWizard>0</ShowDocumentWizard>
      <SaveAsLocation>%DESKTOP%\Letter Version 3.doc</SaveAsLocation>
      <ReplaceExisting>-1</ReplaceExisting>
      <CloseDocument>-1</CloseDocument>
      <Contents>
         <Content ID = "02 Handelsregister"></Content>
         <Content ID = "04 Rechtsträger"></Content>
      </Contents>
   </CreateDocument>
</Parameters>
```

Multiple Recipients

This sample OSC file shows how to create a document with multiple recipients. Please note that the first two recipients are marked for printing and that the second recipient will be the active recipient selected in the document wizard and displayed on the document. The template used is the **Letter** template and the document language is set to **English UK (2057)**.

This sample can be found in our officeatwork example solution, which is part of the officeatwork application installer. The sample file is located in the folder **Examples\OSC Files** of the officeatwork solution and is named **Sample Multiple Recipients.osc**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Parameters>
   <CreateDocument>
      <Language>2057</Language>
      <TemplateID>Letter</TemplateID>
      <ShowDocumentWizard>-1</ShowDocumentWizard>
      <MasterProperties>
         <MasterProperty IDName="Recipient">
            <Field Name="IDName" Value="Sample Recipient 1"/>
            <Field Name="CompleteAddress" Value="Sample No 1
Address
Street
City"/>
            <Field Name="Company" Value="Company Sample 1"/>
            <Field Name="Department" Value="Department Sample 1"/>
            <Field Name="FullName" Value="FullName Sample 1"/>
            <Field Name="AddressStreet" Value="AddressStreet Sample 1"/>
            <Field Name="AddressZIP" Value="1111"/>
            <Field Name="AdressCity" Value="AdressCity Sample 1"/>
            <Field Name="Telephone" Value="+11 (0)11 111 1111"/>
            <Field Name="EMail" Value="1.1@examplesolution.com"/>
            <Field Name="RecipientPrint" Value="-1"/>
            <Field Name="RecipientActive" Value="0"/>
         </MasterProperty>
         <MasterProperty IDName="Recipient">
            <Field Name="IDName" Value="Sample Recipient 2"/>
            <Field Name="CompleteAddress" Value="Sample No 2
Address
Street
City"/>
            <Field Name="Company" Value="Company Sample 2"/>
            <Field Name="Department" Value="Department Sample 2"/>
            <Field Name="FullName" Value="FullName Sample 2"/>
            <Field Name="AddressStreet" Value="AddressStreet Sample 2"/>
            <Field Name="AddressZIP" Value="2222"/>
            <Field Name="AdressCity" Value="AdressCity Sample 2"/>
            <Field Name="Telephone" Value="+22 (0)22 222 2222"/>
            <Field Name="EMail" Value="2.2@examplesolution.com"/>
            <Field Name="RecipientPrint" Value="-1"/>
            <Field Name="RecipientActive" Value="-1"/>
         </MasterProperty>
         <MasterProperty IDName="Recipient">
            <Field Name="IDName" Value="Sample Recipient 3"/>
            <Field Name="CompleteAddress" Value="Sample No 3
Address
Street
City"/>
            <Field Name="Company" Value="Company Sample 3"/>
            <Field Name="Department" Value="Department Sample 3"/>
            <Field Name="FullName" Value="FullName Sample 3"/>
            <Field Name="AddressStreet" Value="AddressStreet Sample 3"/>
            <Field Name="AddressZIP" Value="3333"/>
            <Field Name="AdressCity" Value="AdressCity Sample 3"/>
            <Field Name="Telephone" Value="+33 (0)33 333 3333"/>
            <Field Name="EMail" Value="3.3@examplesolution.com"/>
            <Field Name="RecipientPrint" Value="0"/>
            <Field Name="RecipientActive" Value="0"/>
         </MasterProperty>
      </MasterProperties>
   </CreateDocument>
</Parameters>
```

Print

This sample OSC file shows how to create a document and print it using an officeatwork output variant. The template used is the **Baubewilligung** template and the document language is set to **German Switzerland (2055)**.

This sample can be found in our officeatwork example solution, which is part of the officeatwork application installer. The sample file is located in the folder **Examples\OSC Files** of the officeatwork solution and is named **Sample Print.osc**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Parameters>
   <CreateDocument>
      <Language>2055</Language>
      <TemplateID>Baubewilligung</TemplateID>
      <ShowDocumentWizard>0</ShowDocumentWizard>
      <Output>
         <Print UID="20030107111850943437505337" ShowDialog="0" />
      </Output>
      <CloseDocument>-1</CloseDocument>
   </CreateDocument>
</Parameters>
```

Recipient

This sample OSC file shows how to create a document and set its recipient information. The template used is the **Letter** template and the document language is set to **English UK (2057)**.

This sample can be found in our officeatwork example solution, which is part of the officeatwork application installer. The sample file is located in the folder **Examples\OSC Files** of the officeatwork solution and is named **Sample Recipient.osc**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Parameters>
   <CreateDocument>
      <Language>2057</Language>
      <TemplateID>Letter</TemplateID>
      <ShowDocumentWizard>-1</ShowDocumentWizard>
      <MasterProperties>
         <MasterProperty IDName="Recipient">
            <Field Name="CompleteAddress" Value="Muster AG
Marketing
Mrs. Jolanda Smith
Matten 12
1111 Musterlingen"/>
            <Field Name="Company" Value="Muster AG"/>
            <Field Name="Department" Value="Marketing"/>
            <Field Name="FullName" Value="Mrs. Jolanda Smith"/>
            <Field Name="AddressStreet" Value="Matten 12"/>
            <Field Name="AddressZIP" Value="1111"/>
            <Field Name="AdressCity" Value="Musterlingen"/>
            <Field Name="Telephone" Value="+41 (0)44 444 4444"/>
            <Field Name="EMail" Value="jolanda.smith@examplesolution.com"/>
            <Field Name="Introduction" Value="Dear Mrs. Smith"/>
            <Field Name="Closing" Value="Kind regards"/>
         </MasterProperty>
      </MasterProperties>
   </CreateDocument>
</Parameters>
```

Save

This sample OSC file shows how to create a document and save it using an officeatwork output variant. The template used is the **Baubewilligung** template and the document language is set to **German Switzerland (2055)**.

This sample can be found in our officeatwork example solution, which is part of the officeatwork application installer. The sample file is located in the folder **Examples\OSC Files** of the officeatwork solution and is named **Sample Save.osc**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Parameters>
   <CreateDocument>
      <Language>2055</Language>
      <TemplateID>Baubewilligung</TemplateID>
      <ShowDocumentWizard>0</ShowDocumentWizard>
      <Output>
          <Save UID="2006121210441235887611" ShowDialog="0" Path="%Desktop%" Filename="Baubewilligung.pdf"
ReplaceExistingFile="-1" />
      </Output>
      <CloseDocument>-1</CloseDocument>
   </CreateDocument>
</Parameters>
```

Send

This sample OSC file shows how to create a document and send it via email using an officeatwork output variant. The template used is the **Baubewilligung** template and the document language is set to **German Switzerland (2055)**.

This sample can be found in our officeatwork example solution, which is part of the officeatwork application installer. The sample file is located in the folder **Examples\OSC Files** of the officeatwork solution and is named **Sample Send.osc**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Parameters>
   <CreateDocument>
      <Language>2055</Language>
      <TemplateID>Baubewilligung</TemplateID>
      <ShowDocumentWizard>0</ShowDocumentWizard>
      <Output>
         <Send UID="2006121210395821292110" ShowEmail="-1" Filename="Baubewilligung.pdf" />
      </Output>
      <CloseDocument>-1</CloseDocument>
   </CreateDocument>
</Parameters>
```

VBA Samples

Content

This code sample shows how to create a document including numerous contents. The template used is the **Letter** template and the document language is set to **German Switzerland (2055)**.

```
Sub OfficeatworkApiSample()

    Dim lOawAPI As oawAPI.API
    Dim lParam As String

    lParam = "<?xml version=""1.0"" encoding=""ISO-8859-1""?>"
    lParam = "<Parameters>" & _
             "<CreateDocument>"


    lParam = lParam & _
                 "<Language>" & _
                     "2055" & _
                 "</Language>" & _
                 "<TemplateID>" & _
                     "Letter" & _
                 "</TemplateID>" & _
                 "<ShowDocumentWizard>" & _
                     "-1" & _
                 "</ShowDocumentWizard>"

    lParam = lParam & _
                 "<Contents>" & _
                     "<Content ID=""Sample""/>" & _
                     "<Content ID=""Sample with values"">" & _
                         "<Value Name=""Beschreibung"" Value=""Karton 1000 Stk. Tischset""/>" & _
                         "<Value Name=""Farbe"" Value=""blau/rot""/>" & _
                         "<Value Name=""Preis"" Value=""123.45""/>" & _
                     "</Content>" & _
                     "<Content ID=""Sample with values"" " & _
                             "LCID=""2057"" Bookmark=""Text"" InsertionMethod=""0"">" & _
                         "<Value Name=""Beschreibung"" Value=""Karton 1000 Stk. Tischset""/>" & _
                         "<Value Name=""Farbe"" Value=""blau/rot""/>" & _
                         "<Value Name=""Preis"" Value=""123.45""/>" & _
                     "</Content>" & _
                 "</Contents>"

    lParam = lParam & _
             "</CreateDocument>" & _
         "</Parameters>"

    Set lOawAPI = New oawAPI.API

    If (lOawAPI.ExecuteXML(lParam) = -1) Then
        MsgBox "Dokument wurde erstellt."
    Else
        MsgBox "Dokument konnte nicht erstellt werden."
    End If

    Set lOawAPI = Nothing

End Sub
```

Letter

This code sample shows how to create a document based on the **letter** template. The document language is set to **German Switzerland (2055)**.

```
Sub OfficeatworkApiSample()

    Dim lOawAPI As oawAPI.API
    Dim lParam As String

    lParam = "<Parameters>" & _
                "<CreateDocument>" & _
                    "<Language>" & _
                        "2055" & _
                    "</Language>" & _
                    "<TemplateID>" & _
                        "Letter" & _
                    "</TemplateID>" & _
                    "<ShowDocumentWizard>" & _
                        "-1" & _
                    "</ShowDocumentWizard>" & _
                "</CreateDocument>" & _
            "</Parameters>"

    Set lOawAPI = New oawAPI.API

   If (lOawAPI.ExecuteXML(lParam) = -1) Then
      MsgBox "successfully created document"
   Else
      MsgBox "the document could not be created"
   End If

    Set lOawAPI = Nothing

End Sub
```

Multiple Documents

This code sample shows how to create multiple documents in one go. The template used is the **Offer** template and the document language is set to **German Switzerland (2055)**.

```
Sub OfficeatworkApiMultiple()

   Dim lOawAPI As oawAPI.API
   Dim lParam As String

   lParam = "<?xml version=""1.0"" encoding=""ISO-8859-1""?>"
   lParam = "<Parameters>" & _
               "<CreateDocument>" & _
                   "<TemplateID>" & _
                       "Offer" & _
                   "</TemplateID>" & _
                   "<ShowDocumentWizard>" & _
                       "-1" & _
                   "</ShowDocumentWizard>" & _
                   "<Language>" & _
                       "2055" & _
                   "</Language>" & _
               "</CreateDocument>" & _
               "<CreateDocument>" & _
                   "<TemplateID>" & _
                       "Letter" & _
                   "</TemplateID>" & _
                   "<ShowDocumentWizard>" & _
                       "-1" & _
                   "</ShowDocumentWizard>" & _
                   "<Language>" & _
                       "2055" & _
                    "</Language>" & _
               "</CreateDocument>" & _
           "</Parameters>"

   Set lOawAPI = New oawAPI.API

   If (lOawAPI.ExecuteXML(lParam) = -1) Then
      MsgBox "successfully created docuent"
   Else
      MsgBox "the document could not be created"
   End If

   Set lOawAPI = Nothing

End Sub
```

Recipient

This code sample shows how to create a document including recipient information. The template used is the **Letter** template and the document language is set to **English UK (2057)**.

```
Sub OfficeatworkApiSample()

    Dim lOawAPI As oawAPI.API
    Dim lParam As String

    lParam = "<?xml version=""1.0"" encoding=""ISO-8859-1""?>"
    lParam = "<Parameters>" & _
                 "<CreateDocument>"


    lParam = lParam & _
                     "<Language>" & _
                         "2057" & _
                     "</Language>" & _
                     "<TemplateID>" & _
                         "Letter" & _
                     "</TemplateID>" & _
                     "<ShowDocumentWizard>" & _
                         "-1" & _
                     "</ShowDocumentWizard>"

    lParam = lParam & _
                     "<MasterProperties>" & _
                         "<MasterProperty IDName=""Recipient"">" & _
                             "<Field Name=""Company"" Value=""Muster AG""/>" & _
                             "<Field Name=""Department"" Value=""Marketing""/>" & _
                             "<Field Name=""FullName"" Value=""Mrs. Jolanda Smith""/>" & _
                             "<Field Name=""AddressStreet"" Value=""Matten 12""/>" & _
                             "<Field Name=""AddressZIP"" Value=""1111""/>" & _
                             "<Field Name=""AddressCity"" Value=""Musterlingen""/>" & _
                             "<Field Name=""Telephone"" Value=""+41 (0)44 444 4444""/>" & _
                             "<Field Name=""EMail"" Value=""jolanda.smith@examplesolution.com""/>" & _
                             "<Field Name=""Introduction"" Value=""Dear Mrs. Smith""/>" & _
                             "<Field Name=""Closing"" Value=""Kind regards""/>" & _
                         "</MasterProperty>" & _
                     "</MasterProperties>"

   lParam = lParam & _
                 "</CreateDocument>" & _
              "</Parameters>"

   Set lOawAPI = New oawAPI.API

   If (lOawAPI.ExecuteXML(lParam) = -1) Then
      MsgBox "successfully created document"
   Else
      MsgBox "the document could not be created"
   End If

    Set lOawAPI = Nothing

End Sub
```

Glossary

O

officeatwork Master-Template

An officeatwork Master-Template is comparable to a Word template (file ending in .dot). Within the Master-Template, the template format is allocated, the margins adjusted, the logos set up, etc. The Master-Template creates the necessary foundation for the corresponding type of document (such as a letter, fax, memo, etc.). An officeatwork Master-Template file has the ending **.owt**.

officeatwork Smart-Template

An officeatwork Smart-Template is a template which is connected to an officeatwork Master-Template. As well as this, an officeatwork Smart-Template contains different contents, which are managed either by the officeatwork Smart-Template Manager or the officeatwork Smart-Content Manager. An officeatwork Smart-Template has been created multilingually, so that every content can be processed in different languages. An officeatwork Smart-Template file has the ending **.ows**.

P

Power User

Proficient users who know how to use different computer applications very well are known as power users. A Microsoft Word power user therefore has a sound knowledge of Microsoft Word.

Index

officeatwork Ltd.
Bundesplatz 12
6300 Zug, Switzerland