N.B. This is the last version of the Reference to be released in this format. The latest content is now available at http://www.xpressdox.com/codex/userref/.

1. Introduction

XpressDox has been designed to complete marked-up templates in two general situations. The first is when data is captured and stored as part of an application (such as an accounting system, or any other system requiring the filling of standard documents). The second is to assist users such as secretaries who use the system within Microsoft Word (or, eventually, OpenOffice Writer) to produce standard documents based on letterheads, or similar uses.

In the latter situation (i.e. the system is used within Microsoft Word), typically the users will have many pre-defined documents (normally of the nature of letters, faxes, etc. but can include more substantial documents like agreements) which contain fixed text as well as text which varies from one usage of the document to another. An example would be a letter requesting payment of an outstanding account, where the layout and one or two paragraphs are constant, but the account information such as balance outstanding, name and address of debtor, etc., is variable.

2. Using Merge Fields to include variable data

These documents are really templates, and are constructed in MSWord with the fixed text typed as in any normal document, but the pieces of variable data are entered as "Merge Fields". A Merge Field is the name of the data element, enclosed inside the "field markers", which are << and >>. An example of a Merge Field would be <<BalanceOutstanding>>. The section **Notes on syntax** provides the rules for naming of data elements.

The templates are stored as Word XML documents, and the source of the data for the Merge Fields (called the "Data Set") is also represented in XML format. An example of the data for the letter requesting payment would be:

```
<?xml version = "1.0"?>
<letter>
      <Reference>MNO098765</Reference>
      <YourRef>CASE 012234</YourRef>
      <Addressee>Mr. J. Fish</Addressee>
      <AddressLine1>18 Long Road</AddressLine1>
      <AddressLine2>Cape Town</AddressLine2>
      <Date>2008/03/23</Date>
      <BalanceOutstanding>1010.25</BalanceOutstanding>
      <NatureOfAccount>Cheque</NatureOfAccount>
      <FeeEarner>Mr Murgatroyd</FeeEarner>
</letter>
```

3. Layout and processing

Certain layout and processing functions are available, and are specified as part of the Merge Field.

a. **ForEach**: where a group of data elements is repeated (for example a list of parties to an agreement), then these can be inserted into a document under control of a ForEach Command, for example:

   The parties are:
   <<ForEach(party)>>Surname: <<surname>>
   First names: <<firstnames>>

<<End()>>

b. **Sort**: this is actually a subcommand of the ForEach Command. For example, to sort the above list by surname, the ForEach would look like:

The parties are:
<<ForEach(party,surname,ascending,text)>>Surname: <<surname>>
First names: <<firstnames>>
<<End()>>

In this example the parties would be listed in ascending order of surname. The "ascending" and "text" are the order and data type, respectively, and these values could in fact have been omitted in this example, as they are the defaults. The other option for order is "descending", and the other option for data type is "number".

c. **List**: where it is necessary to format a list as comma delimited, with the word "and" between the last two names, this can be achieved using the List Command, for example:

The parties are: <<List(parties,firstname surname,!, , and)>>

i) Note the "!" in front of the comma. This is an "escape" character and is required before a comma, or left or right parenthesis whenever that character is required as is in the output text. This is because the characters "," "(" and ")" are used by XpressDox as command and function parameter list delimiters. That means that, if any of those three characters are required to be in the printed output (as is the case with the comma above), then they must be prefixed by !.

d. **If**: text can be included depending on the value of a data element in the Data Set. For example, suppose special wording is required when the BalanceOutstanding is greater than R10000; the If Command would be used like this:

We note that your account is overdrawn by an amount of R<<BalanceOutstanding>>. Please ensure that this amount is paid to us within 48 hours<<If(BalanceOutstanding > 10000)>> failing which you will be dealt with harshly<<End()>>.

The Else() command can be used in conjunction with If:

The document must also be signed by <<If(PartyType="Minor")>>the legal guardian<<Else()>>the party's marital partner<<End()>>.

e. **IncludeTemplate**: an entire template file can be inserted into the document at this point. The included template can contain Merge Field definitions and even issue further IncludeTemplate commands.

The description of the parties is: <<IncludeTemplate(PartyDescription)>>

Notice that PartyDescription is regarded by XpressDox as a file name – when no extension is supplied, XpressDox uses the XpressDox Template extension, i.e. .xdtpl.

Notice also that the full path of the included template is not specified. In a case like this, XpressDox will assume that the included template is in the same folder as the original template.

Please see the notes in the Configuration section on Helper Folders, for further template referencing features.

Please also see **Source and Destination Formatting with BaseTemplate, IncludeTemplate and InsertDocument.** below for more important information on the very flexible way that XpressDox has of formatting the result document.

f. **IncludeTemplateText**: this is similar to IncludeTemplate, but only the unformatted text of the template is inserted.

g. **Ordinal**: used within a ForEach to output the ordinal value (i.e. "first", "second", etc.) of the position in the list of the current item:

```
<<ForEach(party)>>
The <<Ordinal())>> party is <<firstname surname>>.
<<End(ForEach party)>>
```

This would result in something like this:

```
The First party is Fred Basset.
The Second party is Harry Smith.
The Third party is Ivan Bosman.
The Fourth party is Maximilian Jones.
The Fifth party is Johan Smit.
The Sixth party is William Wilberforce.
The Subsequent party is Mortimer Rodent.
The Subsequent party is Petrus du Toit.
```

The above demonstrates the default action for Ordinal.

Notes:

i) Only six values are provided, any number after six is rendered as "Subsequent".

ii) If there was only one party in the list, then the default value of <<Ordinal()>> is an empty string, i.e. the output of the above would look like:

```
The  party is Fred Basset.
```

To override the default behaviour, the options are coded as parameters, for example:

```
<<ForEach(party)>>
The
<<Ordinal(only,first,second,third,fourth,fifth,sixth,seventh,subsequent)>>
party is <<firstname surname>>.
<<End(ForEach party)>>
```

In the case of only one party in the list, this would then read:

```
The only party is Fred Basset.
```

The eight-member example above would become:

```
The first party is Fred Basset.
The second party is Harry Smith.
```

The third party is Ivan Bosman.
The fourth party is Maximilian Jones.
The fifth party is Johan Smit.
The sixth party is William Wilberforce.
The seventh party is Mortimer Rodent.
The subsequent party is Petrus du Toit.

h. **RunTemplates**: when a template is run from within MSWord, this command will cause the named templates to be run. The RunTemplates command should be the only text in a template, because everything else about the template is ignored.

<<RunTemplates(Letter,Contract,DebitOrder,FileCover)>>

The four templates in the list will be run, giving the user the ability to capture data for each one, although the system will by default use the data from the previous template in the list.

i. **RunWordMacro:** when a template is run from within MSWord, then this command will cause the specified macro to be run after the template filling has completed.

<<RunWordMacro(ChangeParagraphAlignment)>>

This will cause the macro "ChangeParagraphAligment" to be run. If the macro does not exist, a warning message is issued to the user.

j. **SetDocumentName:** this is part of a powerful feature, also in the MSWord context, which facilitates the naming of the merged document. This is more fully explained in the Configuration section.

k. **SetSavedDocumentFolder:** with this command the template author can designate a folder into which merged documents using this template are to be saved. If the folder reference is a relative reference then it is regarded as relative to the configured document save folder. See the Configuration section for more information.

l. **SetSavedDocumentFileName:** this is used by the template author to designate the name of the file to which the merged document must be saved. It can be an absolute file reference, or relative, in which case it is relative either to the folder indicated by a SetSavedDocumentFolder command on the same template, or relative to the configured Document Save Folder. See the Configuration section for more information.

m. **SetSavedDataFolder:** designates the folder to which the captured data must be saved. Can also be absolute or relative, with the relative address being relative to the configured Data Save Folder, or, if that is not specified, it is relative to the folder in which the merged document was saved. See the Configuration section for more information.

n. **SetSavedDataFileName:** similar to SetSavedDocumentFileName, but applies to the data file.

o. **When:** is similar to If but for situations where the text to be included or excluded is plain text, not including any formatting or layout:

<<When(Sex='F',she,he)>>

4. Letterheads

Letterheads are of special interest because they are used as the basis for many documents. With XpressDox it is possible to create one letterhead template, and by referring to it with the BaseTemplate Command, use that template as the basis for many other documents without having to specify the letterhead formatting in those documents.

A (very simple) sample letterhead would look like this:

> Your reference: <<yourref>> Our reference: <<ourref>>
> Date: <<date>>
>
> <<Addressee>>
> <<AddressLine1>>
> <<AddressLine2>>
>
> **Re: <<re>>**
>
> <<DocumentBody>>
>
> Per:
> <<FirmName>>
> <<FeeEarner>>

Supposing this template is stored as "Letterhead.xdtpl", then an example of a template using this as the base template (called the "source template") would look like:

> <<BaseTemplate(Letterhead)>>
> We note that your account is now overdue by an amount of
> <<BalanceOutstanding>>.
> Please make sure that payment is made forthwith.

Some notes:

a. By default, the entire content of the source template excluding the paragraph containing the BaseTemplate Merge Field will be inserted into the letterhead's <<DocumentBody>> Merge Field. The resulting template is called the "destination template".

b. The <<BalanceOutstanding>> Merge Field in the source template will be included in the destination template. Data elements for all the Merge Fields on both the source template and the base template (i.e. the letterhead) will need to be in the Data Set for the destination template.

5. Customizing and the use of Base Templates

Sometimes it would be useful to be able to customize a source template for more than just the letter body. For example, it may be that the Data Set contains data for more than one set of addressees and the source document needs to be able to indicate which of these addressees is to be included on the base template. This can be achieved using a ReplaceField Command. The earlier source template would then look something like this:

<<BaseTemplate(Letterhead.xml)>>
<<ReplaceField(Addressee)>><<DebtorName>><<ReplaceFieldEnd()>>
<<ReplaceField(AddressLine1)>><<DebtorAddressLine1>><<ReplaceFieldEnd()>>
<<ReplaceField(AddressLine2)>><<DebtorAddressLine2>><<ReplaceFieldEnd()>>
<<ReplaceField(FeeEarner)>>R.J. Smith-Jones<<ReplaceFieldEnd()>>
<<ReplaceField(DocumentBody)>>
We note that your account is now overdue by an amount of
<<BalanceOutstanding>>.
Please make sure that payment is made forthwith.
<<ReplaceFieldEnd()>>

Some notes:

a. When using the ReplaceField Command for any of the base template Merge Fields, the DocumentBody Merge Field must be specified (i.e. there is no default for it in this situation, unlike when no ReplaceFields are used, in which case the DocumentBody is the entire originating template, excluding the BaseTemplate Merge Field).

b. Note the replacement of the FeeEarner Merge Field: it is replaced not with another Field but with the constant text "R.J. Smith-Jones". This would be the situation where the source template is only ever used by a particular fee earner, and so the destination template needs to reflect that fee earner, and not any fee earner which may be in the Data Set.

c. When a "skeleton" letter is based on a letter head (a "skeleton" typically containing only the addressee information and salutation and closing), then the template author would like the user to be prompted in some way to fill in the remainder of the letter body manually. This is what the ComeHereAfterRun command was designed for – after a template is run then this command will cause the Word cursor to position at the point in the document where it appeared:

<<Addressee>>
<<Address1>>
<<Address2>>

Re:<<ReForLetter>>

<<**ComeHereAfterRun()**>>

Yours faithfully

6. Inserting documents whose file names are contained in data elements in the source data.

In the case of the BaseTemplate and IncludeTemplate commands, the name of the template to be included is known at the time the template is authored, and so the actual file name of the template is referenced in the command. In some situations, the name of a file to be included in the document is contained in the Data Set. In these cases, instead of IncludeTemplate, the command InsertDocument is used.

<<InsertDocument(FileNameField)>>

7. Formatting individual data elements.

Formatting functions can be applied to any data element name where the value of the data element is to be included in the resulting text, i.e. when the data element name would otherwise in a Merge Field all on its own, or inside a List command.

a. **Currency**: this function will translate a numeric value into a phrase representing an amount in a currency.

```
<<Currency(price,'Pound','Pounds','Penny','Pence', 'only','en','Leave')>>
```

The parameters are the singular and plural values of the denomination, the 'only' parameter is what must be printed if the number of pence is zero, and the last two parameters are the language and case conversion, respectively.

The above example would render the amount 1234 as "One Thousand Two Hundred and Thirty Four Pounds only".

i) **Rand**: this is a special version of Currency taking only an amount and case conversion as parameters.

```
<<Rand(price,"tolower")>>
```

This would render 123456 as "one thousand two hundred and thirty four rand and fifty six cents".

ii) **RandAfrikaans**: a special Afrikaans version of the Rand function.

```
<<RandAfrikaans(price)>>
```

For a "price" value of 1234.56, gives "Een Duisend Twee Honderd Vier en Dertig Rand Ses en Vyftig sent".

iii) Other special currency functions are: DollarsFrancaise, Dollars, EuroDeutsch, EurosFrancaise and Pounds.

b. **ExtractInitials**: takes, for example, a data element containing the full names of a person, and extracts the initials.

```
<<ExtractInitials(FirstNames,'.')>>
```

The second parameter to the function is the character (or characters) which appears between the letters of the initials.

c. **FormatNumber**: Numerous patterns can be used. Here follow some examples:

```
<<FormatNumber(age," 0.0")>>
<<FormatNumber(amount)>>
<<FormatNumber(price,'R#!,#0.00','ZA')>>
<<FormatNumber(balance,"###!,###!,##0.00;!(###!,###!,##0.00!)")>>
<<FormatNumber(IDNumber, "000000-0000-000")>>
```

Where the value of "age" is 100, the result will be 100.0

Where "amount" is 9876, the result will be 9,876.00 (the default format).

Where the value of "price" is "123456", the result will be R123 456,00. Note that the decimal format code "ZA" is required to indicate that the decimal point and thousands separator should be according to official South African

format (as required, for example, by the Deeds Registry). Note that the decimal format code "EU" will format the number according to European decimal rules, i.e. a full stop as the thousands separator and a comma as the decimal point.

Where "balance" is "123456.78" the result is 123,456.78 and where "balance" is "-123456.78" the result will be (123,456.78).

Note also the escaping of !, !(and !) where relevant.

Note also that the parameters of formatting functions (as against those of layout commands) must be enclosed in single or double quotes, unless their value is contained in a data element, in which case the data element name is not enclosed in quotes (in other words, it is those quotes, or their absence, that let's XpressDox know you want a formatting parameter as typed in the command, or as extracted from the Data Set, respectively).

d. **FormatDate**: date data elements can be formatted according to the format patterns defined for the .NET framework. For example:

```
<<FormatDate(dateofengagement,"yyyy-MM-dd")>>
<<FormatDate(dateofengagement,"dd MMM yyyy","af")>>
<<FormatDate(dateofdismissal,"d MMMM!, yyyy")>>
```

The first example would result in something like, respectively, 2008-02-29, 29 Februarie 2008, and 29 February, 2008 (not because the default language is English, but because the default is whatever is defined in the regional settings). Note the quotes and the escaped comma.

e. **GetItemOfElement**: the data element contains a number (an integer) which is used to get an item from a list, where the list is coded in the Merge Field itself:

```
<<GetItemOfElement(FullNames,2," "))>>
```

In the above example, if the value of the FullNames data element is "Johannes Petrus Stefanus du Toit", then the result of this Merge Field function is the string "Petrus"

When the index (the number 2 in the above example) is more than the number of elements in the data element, then the last element is returned. If there are no occurrences of the delimiter (the " " in the example), then the entire data element is returned.

f. **GetListItem**: the data element contains a number (an integer) which is used to get an item from a list, where the list is coded in the Merge Field itself:

```
<<GetListItem(BondNumber,First,Second,Third,Fourth,Fifth,Big)>>
```

If the value of the BondNumber data element is 2, then the result of this Merge Field function is the string "Second". If the value is not an integer, or is less than 1, then the result returned is "First", and if the value is 6 or more, then the result is "Big".

g. **HardSpace**: converts all normal spaces to hard spaces for insertion into a Word document:

```
<<HardSpace(FormatNumber(Amount,"#!,#0.00","ZA")
```

If the Amount data element is, e.g. 1200, then the above will yield 1 200,00, where the character between the 1 and 2 digits is hard space.

h. **IncrementDate**: this function will take a date and increment a certain number of days, months or years to the date, and return the resulting date in a format which can be printed, or can be passed to the FormatDate function.

<<IncrementDate(DateOfBirth,25, "y")>>

The above will add 25 years to the value in the data element "DateOfBirth".

Tomorrow is: <<IncrementDate("today",1, "d")>>

This will add 1 day to today's date. The string "today" is a special instruction to the function to use the current day's date.

About six months ago (on <<FormatDate(IncrementDate("today",-6, "m"),"d MMMM yyyy")>>) the weather was completely different.

Note how the increment units can be negative, and how you can pass the result of one function to another.

i. **Max**: calculates the maximum of two values:

The maximum of <<Number1>> and 100.1 is <<Max(Number1,100.1)>>.

j. **Min**: calculates the minimum of two values:

The maximum of <<Number1>> and 100.1 is <<Max(Number1,100.1)>>.

k. **Now**: inserts the current date and/or time into the document. It can be formatted in the same way as for "FormatDate":

<<Now("d MMMM!, yyyy")>>
<<Now("yyyy/MM/dd HH:mm:ss)>>
<<Now("d MMMM!, yyyy", "af")>>

On 1st January 2008 the last example would insert "1 Januarie, 2008".

l. **NumberPhrase**: this function will take a numeric value and translate it to a phrase (i.e. words) in the language specified:

<<NumberPhrase(area,"af","ToUpper")>>

If data element "area" had the value 1234 then the result would be "EEN DUISEND TWO HONDERD VIER EN DERTIG".

Notes:

i) The language code "af" is the ISO639-2 code for Afrikaans (see http://www.loc.gov/standards/iso639-2/php/code_changes.php).

ii) Other languages supported are "en" (UK English, the default), "fr" (French), "de" (German), "en-us" (where there is no "and" after an amount in hundreds, i.e. "101" becomes "One Hundred One"), "en-za" (which renders 1.10 as "One Comma One Zero").

iii) The "ToUpper" parameter is obvious, and could also be "ToLower", or "Leave" (which is the default).

m. **Replace**: use this to replace all occurrences of one string in a data element with another string:

<<Replace(Address,"Ave.","Avenue")>>

n. **StartsWithVowel**: the result of this function would typically not be included directly in the merged document, but would be used in a conditional, such as:

If <<When(StartsWithVowel(Fruit),an,a)>> <<Fruit>> is ripe it tastes better than if it is rotten.

The function assumes that "a", "e", "i", "o" and "u" (and their uppercase variants) are the only vowels. Sometimes letters like "H" are regarded as vowels. This would be indicated in the following way:

If <<When(StartsWithVowel(Fruit,"hH"),an,a)>> <<Fruit>> is ripe it tastes better than if it is rotten.

o. **TableLookup:** enables the calculation of an amount from a table. Typically this is used with tax tables, or other tables of tariffs.

The normal tax calculation for the 2008/09 tax year is based on this table (reproduced from the SARS web site):

Taxable income (R)	Rate of tax (R)
1 – 122 500	18% of each R1
122 501 – 195 000	21 960 + 25% of the amount above 122 500
195 001 – 270 000	40 210 + 30% of the amount above 195 000
270 001 – 380 000	62 710 + 35% of the amount above 270 000
380 001 – 490 000	101 210 + 38% of the amount above 380 000
490 001 and above	143 010 + 40% of the amount above 490 000

The TableLookup function representing this table would be:

<<TableLookup(Income,"122500;0;18,195000;21960;25,270000;40210;30,380000;62710;35,490000;101210;38,999999999;143010;40")>>

p. **ToUpper**: this will place the uppercase value of the data into the document.

<<List(parties,ToUpper(firstname) ToUpper(surname),!, , and)>>

q. **ToLower**: converts the value to lowercase.

<<ToLower(NatureOfAccount)>>

r. **ToSentence**: converts the value to lower case except for the first character which is converted to upper case.

<<ToSentence(ProductDescription)>>

s. **ToTitle**: converts the value to Title Case (i.e. the first character of each word (with some exceptions) is capitalized).

<<ToTitle(NamesOfParties)>>

t. **WindowsLogonUser**: yields the Windows Logon User.

The person running this template is logged on to Windows with the user name: **<<WindowsLogonUser()>>**.

8. Notes on syntax.

a. Data element names must consist only of alpha-numeric characters (A-Z, a-z, 0-9), a full stop, a hyphen and the underscore character(_).

   i) A full-stop can be part of the name, as long as it isn't the first character.

   ii) Data element names are case-sensitive: in other words "Address" and "ADDRESS" refer to two different data elements.

b. The Commands themselves are case insensitive – so ForEach, FOREACH and foreach are all acceptable.

c. The End Command which closes off ForEach and If commands is also case insensitive. Note that the End is always followed by "()" (to distinguish it from possible data elements named "end"), but the text within the two parentheses has no meaning to XpressDox and so can be used to make the context more apparent. For example:

```
<<ForEach(book)>>Title:     <<title>>
Author:          <<author>>
<<IF(author = "Stephen King")>> Be afraid, be very afraid.<<End(
IF)>><<End( foreach book)>>
```

d. The data element names are case sensitive, and must match the case of the data element in the Data Set.

e. Function calls can be nested. Please see the section on XSLT functions (12 below) concerning this.

f. Quote marks (i.e. "..." or '...' or "..." or '...') need only appear:

   i) in the conditional expressions in If and When commands when comparing with a string value, e.g. <<If(Answer = 'yes')>> but not when comparing with a numeric value, e.g. <<When(count(Child) > 1,children,child)>>;

   ii) around formatting parameters passed to functions, e.g. <<FormatNumber(Amount, "#!,0.00")>>;

   iii) in GetV and SetV functions around the variable names, and in SaveV around the variable name as well as the data element name.

   iv) in XSLT functions like concat if items being referred to are string literals.

9. Block commands

The ForEach and If commands are called "block commands" because they are terminated with a separate End command (and between the start and end there are other Fields). There is a special behaviour related to block commands to help with eliminating unwanted empty paragraphs when the start and end appear on different paragraphs. The rule can be stated as such:

When the block start Merge Field and matching block end Merge Field appear on different paragraphs, then:

a. If there is no text after the block start Merge Field on the paragraph on which it appears, then that paragraph will not be included in the output.

b. If there is no text before the block end Merge Field on the paragraph on which it appears, then that paragraph will not be included in the output.

c. Text to the left of the block start Merge Field and to the right of the block end Merge Field is never output.

That's why, in the example of Ordinal, the paragraphs containing the ForEach and the End Fields did not appear in the output.

Other examples:

i)

This bit before the Merge Field is ignored<<ForEach(party)>>The <<Ordinal()>> party is <<firstname surname>>.
<<End(ForEach party)>>This text after the End Merge Field is also ignored.

The above has exactly the same output as in the first example in 3.g above.

ii)

<<ForEach(party)>>The <<Ordinal()>> party is <<firstname surname>>.<<End(ForEach party)>>

This will look like:

The First party is Fred Basset.The Second party is Harry Smith.The Third party is Ivan Bosman.The Fourth party is Maximilian Jones.The Fifth party is Johan Smit.The Sixth party is William Wilberforce.The Subsequent party is Mortimer Rodent.The Subsequent party is Petrus du Toit.

iii)

<<If(Date = '2008/04/17')>><<ForEach(party)>>The <<Ordinal()>> party is <<firstname surname>>.
<<End(ForEach party)>><<End(if date)>>

The result will be that the If command will be lost (because of rule c above). The way to code this is:

<<If(Date = '2008/04/17')>>
<<ForEach(party)>>The <<(Ordinal()>> party is <<firstname surname>>.
<<End(ForEach party)>>
<<End(if date)>>

The block rules will ensure that the first and last two paragraphs in the above sequence will not be included in the output, which is probably what is required.

10. Source and Destination Formatting with BaseTemplate, IncludeTemplate and InsertDocument.

When including part (or all) of one document into another, which is the basis of the BaseTemplate, IncludeTemplate and InsertDocument commands, there is always an issue about what formatting (i.e. styles, fonts, etc.) should be applied in the resulting document.

XpressDox provides three options for this, viz. Source, Paragraph and Destination. These are shorthand (to reduce the amount of typing required in

Merge Fields) for "Apply Source Document Formatting", "Apply Destination Paragraph Formatting" and "Apply Destination Document Formatting" respectively. The relevant code is placed after the template name in the IncludeTemplate or BaseTemplate command, e.g.

```
<<IncludeTemplate(DirectorsNames,Paragraph)>>
<<BaseTemplate(LetterHead,Destination)>>
```

The three options are described below:

a. **Source**: What happens is that all the styles, fonts, etc. in the included ("source") document are copied (and renamed) into the destination document, and the text, tables, lists, etc., all retain the same format as they had in the original source document. The rest of the text in the destination document remains unaltered.

   This is the default for the BaseTemplate commands, but can, obviously, be overridden in any particular command.

b. **Paragraph**: Every paragraph in the text inserted from the source document (which in the case of the BaseTemplate command is actually the document issuing the BaseTemplate command) into the destination (i.e. the base template itself) is formatted according to the paragraph into which it is inserted. For example, if the <<DocumentBody>> Merge Field in the base template is fill justified, font Century Gothic and indented 2cm, then that is how all the paragraphs inserted from the source template will be formatted, regardless of how they appear in that source template.

c. **Destination**: No explicit reformatting is done. This is an interesting situation because it relies on Word's interpretation of styles, etc., to come into effect. It is best described by an example:

   Suppose the source template (the one whose text is being included into the destination template) has a Normal style which has font Arial and Space After 6 points. And suppose the Normal style in the destination template has font Courier New 18 points and Space After of 24 points. When a paragraph from the source template, in the Normal style, is inserted into the destination template, it will assume the styling of the destination's Normal style – i.e. 18 point Courier New with Space After of 24 points.

   You can see that as long as the design of the styles in the two templates is carefully organized, this can be a very powerful feature.

   This is the default for IncludeTemplate and IncludeDocument.

11. Data Capture

As indicated in the Introduction to this document, XpressDox has been designed to fill marked-up templates in two general situations. The first is when data is captured and stored as part of an application, and the second is to assist users such as secretaries who use the system within Microsoft Word to produce standard documents.

XpressDox has a special set of features when run from the Microsoft Word Add-in. In this context, a marked-up template can be "run", which has the following effects:

- When the user selects a template to be run, then XpressDox analyses the template for data elements;
- XpressDox presents the user with a data capture form, in which the user enters the values to be filled into the template;
- When the user accepts the values entered, XpressDox applies the transformation which formats and inserts the data elements.
- The resulting document and data captured are (optionally) stored by XpressDox (in locations specified and configured by the user). The data are then available for inserting into subsequent templates.

Depending on what (if any) formatting is applied to a data element in the template, XpressDox will infer the type of the data element as either a string, or a number or a date, and will assist the user in the capture of the data by use of specific controls on the capture form (for instance, a date data element will be captured using a calendar control).

a. **CaptureAsLongText and InsertFormattedText**: The data capture UI will provide a multi-line control into which the data for the data element can be captured.

<<CaptureAsLongText(PropertyDescription)>>

If this long text is included in the document via a simple Merge Field (i.e. in the above example <<PropertyDescrption>>), then any line breaks which the user may have typed (i.e. by pressing "Enter" in the data capture control) will be ignored.

<<CaptureAsLongText(PropertyDescription,6)>>

In the above example the parameter 6 requests that the data capture control on the form be approximately 6 lines long.

If it is required that the line breaks (or other formatting such as bold, underline or italic) in the captured data be included in the document, then the command InsertFormattedText must be used:

<<InsertFormattedText(Address)>>

The above example will permit the user to type the Address in as many lines as necessary, and include the Address in the document with the lines separated by line breaks.

Different parts of the "long text" can be formatted in the long text control (by the user) as bold, underlined or italic by using the MSWord shortcut keys for this formatting (viz. <Ctrl B>, <Ctrl U> and <Ctrl I>, respectively).

Note that when using the InsertFormattedText command, it is not necessary to use a CaptureAsLongText command, unless the number of lines in the data capture control is required to be different to the default, in which case the CaptureAsLongText command must appear in the template before the InsertFormattedText command.

b. **CaptureDataElement**: (Was originally called DefineDataElement which can still be used, for backward compatibility) The background to this command probably needs a bit of experience in authoring more complex templates. Because XpressDox uses XSLT technology to format the documents, it is possible (as described more fully below) to use XSLT and XPath expressions

to control some of the processing. In particular, this is the situation with the If and When commands, which, because of their potential complexity, XpressDox does not analyse for data elements to be captured. In some situations, the nature of the If/When condition will be such that one or more of the data elements in the If/When condition will not actually appear in the template to be inserted into the document; in other words, the data element values are only used to evaluate whether some other data element(s) should be included in the document. If the data elements only appear in the condition, then they will not be presented to the user in the data capture form, and then will be missing from the data and the If/When condition will always evaluate to "false".

So, to ensure that the data element IS presented to the user for capture in the data capture form, XpressDox provides the CaptureDataElement command. An example might be:

```
The following books are priced below R400:
<<ForEach(book)>>
<<If(price<400)>>Title:      <<Title>>
<<CaptureDataElement(price)>>
Author:              <<Author>>
Date Published:      <<FormatDate(datepublished,yyyy-MM-dd)>>
<<End( ForEach)>>
```

Note that the CaptureDataElement appears in a paragraph all on its own. XpressDox will remove the entire paragraph from the resulting document.

The above could have been written as follows for even better readability:

```
The following books are priced below R400:
<<CaptureDataElement(book/price)>>
<<ForEach(book)>>
<<If(price<400)>>Title:      <<Title>>
Author:              <<Author>>
Date Published:      <<FormatDate(datepublished,yyyy-MM-dd)>>
<<End( ForEach)>>
```

The "book/price" syntax is XPath syntax and means "the price element which is a child of the book element". XPath is a huge subject, not described here.

Because XPath is a huge subject and is used within XpressDox, one of the results is that there are many situations where the CaptureDataElement might come in useful.

c. **ChooseFromList**: This is a command which will present the user (in the data capture form) with a list of options from which to choose the value of a data element (rather than a free-format text field). This command has meaning only in the Word Add-in context. For example, the following will help the user capture one of South Africa's provinces:

```
I Live in <<ChooseFromList(province,Eastern Cape,Free
State,Gauteng,KwaZulu-Natal,Limpopo,Mpumalanga,Northern
Cape,North-West,Western Cape)>><<ToUpper(province)>>
```

Note that it is important to include a Merge Field indicating where the result of the data capture must be inserted – hence the <<ToUpper(province)>> in the example. If this is omitted then the data element will be captured but not appear in the document.

d. **ChooseFromRdbList**: This is very similar to ChooseFromList but instead of a drop-down list, the choices are presented in a group of "Radio Buttons". An example would be:

> The style of the presentation must be <<ChooseFromRdbList(style,free,formal,semi-formal)>><<style>>

e. **ChooseFromData**: Particularly in the context of using data from a data source such as a database (see the discussion of data sources below), it may be helpful to the user of a template to be able to choose a data element value from data in that data source. This can be achieved by the template author's use of the ChooseFromData command, an example of which would be:

> <<ChooseFromData(RequiredOrderId,Orders/OrderID)>>

f. **ChooseFromFile:** This command is similar to the ChooseFromDataSource command (see 14.b below), except that the data are defined either in a simple text file, or an XML file, and the file does not have to be configured into the Data Sources configuration.

> <<ChooseFromFile(lookups:Correspondents.xdtxt)>>
>
> <<ChooseFromFile(shared:Partners.txt)>>

Text file:

The first line of the text file will contain a list of data element names. These names are delimited by the tab character, or a comma (not both!).

The second and subsequent lines have data element values which correspond to the names in the first line. The values are delimited by the same character as was used in the first line.

The ChooseFromFile command presents the user with a drop-down list which includes in it the first data element value of each of the second and subsequent lines of the text file. When the user chooses a particular value in the drop-down list, then all the data elements corresponding to that line in the text file are included in the data set for the template being run.

The full syntax of the command is demonstrated by this example:

> <<ChooseFromFile(lookups:Partners.xdtxt,Choose the Partner,Refresh,PartnerSurname)>>

The second parameter is the caption which will appear on the data capture UI. For the third parameter ("Refresh"), please see section 14.c below. The fourth parameter is the name of the data element whose values must be displayed in the drop down list in the UI. If it is omitted, then the first data element in the file is used.

Note in the first example above that the file extension is ".xdtxt". Use of this extension is recommended (but not obligatory), as it is used within XpressDox to assist the template author in their tasks.

XML file:

Choosing data from an XML file is done with the same command and the functioning is identical, taking into account that the information in file is in XML format. The file extension for an XML file MUST be ".xml".

Please read section 14.b which includes a discussion on the IncludeFileData command.

g. **ChooseFromSamples:** This command is in all respects the same as the ChooseFromList command, except that in this case the user is permitted to type their own value and is not restricted to just the items in the list. Hence the list is just a list of examples, or samples.

h. **ChooseUsingCheckbox**: The data capture UI will represent this command as a CheckBox. The values of the data element thus chosen will be "true" or "false".

For example:

<<ChooseUsingCheckbox(PersonIsMale)>>
<<When(PersonIsMale = "true",he,she)>>

The command is quite customisable, so another variation could be:

<<ChooseUsingCheckBox(Sex,M,F,A check in the box means Male)>>
<<When(Sex = "M",he,she)>>

i. **ExcludeFromUI**: particularly in the situation where data are included in the data set from a data source, text file or perhaps Standard Data Elements, it may be that the template author does not want the user to be able to amend those data values in the data capture UI that XpressDox creates. Such data elements can be parameters to the ExcludeFromUI command, which will then exclude them from the UI.

<<ExludeFromUI(FirmAddress1,FirmAddress2,FirmAddress3,DirectorName,DirectorCellPhone,DirectorEmail>>

j. **Help:** this command can be used to provide help text for the Word Add-in data capture environment.

<<Help(Interest,Enter the rate of interest, excluding the % sign)>>

The Help command can appear anywhere in the template, not necessarily anywhere close to where the data element will be used.

An alternative way of supplying help text is to put the help text in the Merge Field for the data element, following a ? symbol which follows the normal Merge Field contents, e.g.:

<<FormatNumber(Interest,"#0.00")?Enter the rate of interest>>

Help text can be included in any Merge Field in this way.

k. **Required:** if a data element *must* have a value, then the Required command can be used to enforce this at data-capture time.

<<Required(AccountNumber)>>

<<Required(DateOfBirth,date)>>

The second example above illustrates using the second parameter to indicate the type of data to be captured into the required field.

l. **Tab:** with this command it is possible to split the data capture UI (for the main data element parent, repeated elements are not affected) into a number of "tabs", and to designate specific tabs under which specified data elements must be captured.

```
<<Tab(Financial,SellingPrice,CostPrice,DataOfSale)>>
<<Tab(Miscellaneous)>>
```

The above two commands will cause the UI to be split into two tabs, with captions "Financial" and "Miscellaneous". The data elements SellingPrice, CostPrice and DateOfSale will appear on the "Financial" tab, and all other data elements on the "Miscellaneous" tab.

If the command Tab(Miscellaneous) is omitted, then XpressDox will construct a tab captioned "General" on which all the other data elements will appear.

12. XSLT Functions.

XpressDox uses XSLT technology to insert data elements into Merge Fields. This means that a vast amount of functionality is available to the experienced user, especially the XSLT functions, which are used to format data.

XpressDox functions and XSLT functions can be mixed in one Merge Field, for example:

```
<<ExtractInitials(substring-before(FullNames,';'),'.'))>>
```

It is important to bear in mind that if a data element only ever appears as an argument to a nested function (as in the above two examples), then that data element will not be available to the dynamic data capture facility of XpressDox when used inside MSWord, as explained in 11.e above. To overcome this, if the data element is in fact to be made available to the data capture feature, then it will need to be defined to that feature using the CaptureDataElementcommand.

The Template Author's Toolkit will provide access to the most useful and popular of these functions, and will insert a CaptureDataElement command where relevant. The CaptureDataElement commands can be removed manually if they are superfluous. Normally a superfluous CaptureDataElement command will not cause any unwanted behaviour. An exception to this is if a data element has a specific data type (e.g. date, decimal or integer), then the CaptureDataElement emitted by the Template Author's Toolkit may not designate the correct data type. This can also be corrected manually, if necessary. Note that this only has bearing on the data capture interface within Word, and in particular has no bearing on how the data elements are merged into Merge Fields.

13. Variables

This concept enables the results of calculations to be stored for further use in the template, and even for future templates. Here follow some examples to illustrate this, assuming a Data Set which looks like:

```
<xml>
<librarianName>Mrs. J. Bloggs</librarianName>
```

```
<book>
      <title>Harry Potter</title>
      <author>J.K. Rowling</author>
      <costPrice>200</costPrice>
</book>
<book>
      <title>Carrie</title>
      <author>Stephen King</author>
      <costPrice>150</costPrice>
</book>
</xml>
```

a. Setting, getting and saving a simple variable.

<<SetV('TotalCost',sum(book/costPrice))>>

creates a variable named TotalCost with a value of 350.

<<GetV('TotalCost')>>

inserts the value of the TotalCost variable into the document.

<<SaveV('TotalCost','totalCost')>>

Creates a data element named "totalCost" as a direct descendant of the root element with the value of the TotalCost variable. This only happens AFTER the full template merge has completed, so the new data element will not be available in the template in which it is created.

b. A repeated variable.

<<ForEach(book)>>
<<SetV('SellingPrice',costPrice*1.5)>>

Cost Price: <<FormatNumber(costPrice,"#!,##0.00")>>
Selling Price: <<FormatNumber(xdox:GetV('SellingPrice'),"#!,##0.00)>>

<<SaveV('SellingPrice',concat('book[',position(),']/sellingPrice'))>>
<<End(forEach)>>

At the end of the above two examples, the XML Data Set would look like this:

```
<xml>
<librarianName>Mrs. J. Bloggs</librarianName>
<book>
      <title>Harry Potter</title>
      <author>J.K. Rowling</author>
      <costPrice>200</costPrice>
      <sellingPrice>300</sellingPrice>
</book>
<book>
      <title>Carrie</title>
      <author>Stephen King</author>
      <costPrice>150</costPrice>
      <sellingPrice>225</sellingPrice>
</book>
<totalCost>350</totalCost>
</xml>
```

14. Miscellaneous Advanced Features

a. Selecting a single instance of a repeated data element.

Sometimes it is required that only one instance of a repeated element is required. If the instance number is known at template authoring time, then this is fairly straight forward; to show the price of the second book in a list, for example, would be:

```
<<book[2]/price>>
```

Suppose it is up to the user to choose which book's price is to be shown. In other words, the element choice depends on the value of a data element which is only known after the data have been captured (or provided in some other way after the template has been authored). This is done by using the XPath "parameter" feature, i.e. a $ in front of the name of the data element containing the instance number:

```
<<book[$bookNumber]/price>>
```

b. Including data from a Data Source on a template with the IncludeDataSourceData, ChooseFromDataSource and IncludeFileData commands.

The section on Data Sources in 13 below describes in more detail what a Data Source is.

When a user runs a template from within Word, it is possible for them to select one or more Data Sources to provide data for that template. It is also possible for the template author to indicate that data for that template must be provided by a pre-configured Data Source (so that the user doesn't have to remember to choose it every time they run a template). This is done by use of the IncludeDataSourceData command:

```
<<IncludeDataSourceData(Outlook Contacts)>>
OR
<<IncludeDataSourceData(Outlook Contacts,,,PersonalContact)>>
```

The second example shows how the data from the data source is to be merged into the Data Set as part of the data element whose name is "PersonalContact".

When using a data source to produce a "mail-merge" type of document, then the IncludeDataSourceData command can include a "Range Restriction", which, in the data base sense, is a "where" clause (excluding the word "where"):

```
<<IncludeDataSourceData(Contacts
Spreadsheet,,range=IncludeInMailshot = 'yes')>>
```

This assumes that the data source "Contacts Spreadsheet" has a data field (column) called "IncludeInMailshot" and that all the rows which are required in the template have had the word "yes" put into that column.

It is also possible to select a single row from a Data Source, using the "id=" syntax, as in this command:

```
<<IncludeDataSourceData(FirmAddressAndDetails,Refresh,id=1)>>
```

This assumes that the Data Source has an "id" configured for it which is numeric in type and the above command will select the data for the row which has an "id" value of 1. (Note that if the id column in the Data Source is not numeric, then the value must be in single quotes, as in the previous example).

The ChooseFromDataSource command is very similar to the IncludeDataSourceData command, but will place a control on the Word Add-in data capture form, and the user will be able to choose from the data source and the data elements thus chosen will be included in the Data Set for the template. The second parameter is optional and is a caption to appear on the data capture UI, e.g.

<<ChooseFromDataSource(Clients)>>

<<ChooseFromDataSource(Clients,Select the Client)>>

This will enable the user to choose a row from the Clients data source and the data in that row will be included in the Data Set.

The IncludeFileData command functions in a similar way to IncludeDataSourceData, but instead of referring to a data source in the command, a file is referenced instead (the format of that file is defined in section 11.f above, i.e. "plain text" or XML). The file does not need to be configured as a data source, but functions in much the same way.

<<IncludeFileData(lookups:Clients.xdtxt)>>
OR
<<IncludeFileData(lookups:Transactions.xdtxt,,Account)>>

where "Account" is the name of the parent data element into which the data from the file should be merged.

When data elements are included in the data set via any of these commands, the data element names will be the same as

- For data sources: the column names in the database;
- For text files: the column names provided in the first line of the text file;
- For XML files: the names of the elements in the XML file.

For example, if a <<ChooseFromDataSource(Clients)>> command causes data to be read from the Clients data source, and the databse referenced has a column called "ClientName", then the merge field <<ClientName>> will cuase the name of the client to be merged inot the document.

c. The "Refresh" and "Save" options.

When running templates within the MS Word environment, the user has the option to run a template and then select "Use Other Data" – which means "use the Data Set saved from running this or another template on a previous occasion". This is a very useful feature, but has complications relating to data from Data Sources (and File Data).

The question arises about whether or not the data elements saved on the previous occasion should be used exactly as they were when the previous template was run. This is on the face of it what we would like. But, suppose some data elements were originally included from a Data Source (e.g. an

Excel spreadsheet or a database) and the data elements in the Data Source have subsequently changed, should the new, changed, data be used, or the data as saved when the last template was run?

In general, there is no hard and fast answer to this question, but in particular circumstances, the expectation would be that when data in a Data Source changes, it should affect not only new templates which reference that Data Source, but also templates using older data.

For this reason the IncludeDataSourceData, ChooseFromDataSource, IncludeFileData and ChooseFromFile commands all have a "refresh" option. This indicates that when the user selects "Use Older Data", then the data elements from the relevant Data Source must be refreshed from the Data Source, and not copied directly from the originally saved Data Set.

Examples of usage are:

<<IncludeDataSourceData(Clients,Refresh)>>

<<ChooseFromDataSource(Directors,Select the Director,Refresh)>>

A related option is the "Save" option – which governs what happens when the user changes a value which originated from a Data Source. The template author can decide whether those changed items should be changed in the Data Source or not, or whether the user should be prompted to confirm that the Data Source should be changed.

The "Save" option is connected to the "Refresh" option in that if the "Refresh" is not chosen, then "Save" cannot be chosen.

The full set of "refresh and save" options are, then:

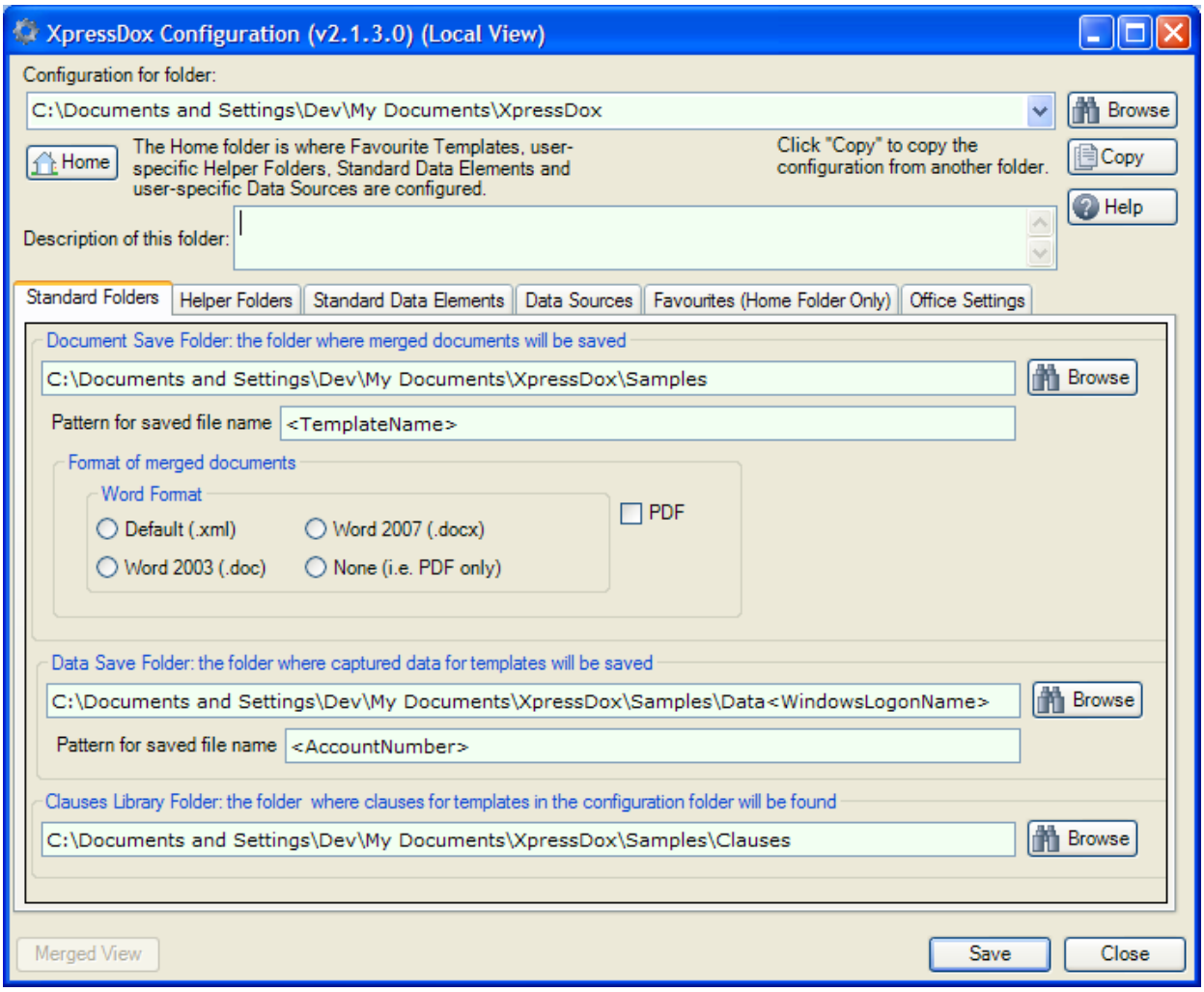
- Refresh
- RefreshNoSave (which is the same as Refresh)
- RefreshOptionalSave: the user is prompted to save any changes.
- RefreshSave: changes made by the user are saved without prompting the user.

The default for IncludeDataSourceData and ChooseFromDataSource is Refresh.

The default for IncludeFileData and ChooseFromFile is also Refresh.

No "Save" option is available for file data.

15. XpressDox Configuration



1. **Configuration for folder:**

   Every folder where templates can be stored (and from which they are selected with the "Run Template" function) can have a configuration saved in it. When a folder is selected as the "Configuration for folder", it means that the configuration settings entered on the screen will be saved in that folder. This in turn means that when a template is run from that folder, those configuration settings will be in effect for the "Run Template" operation.

   Whenever the user chooses a template to be run, whether from the "Run Template" button, or "Run Template with Existing Data", or when a template is run from the "Favourite Templates" menu, XpressDox will look up the configuration which is saved in the folder where that template is selected from, and apply that configuration. A user can run one template from one location and then switch to a template from another location, and the configuration for the particular location will be applied.

2. **Home folder:**

   Each user has a home folder, which is the folder My Documents\XpressDox. This folder will be created by XpressDox when XpressDox is installed. At that

time the installer will also put a sample template into the home folder, as well as the XpressDox documentation (which you are reading).

The "Home" button browses straight to the Home folder. When the Home folder is the one being configured, the Home button is disabled.

a. This is the only folder in which the Favourite Templates can be configured, and from which they will be read to produce the Favourite Templates menu in Word.

b. The Standard Data Elements settings for this folder will override any conflicting Standard Data Elements settings from any other folder (see below regarding "Merging" of configuration settings).

c. The Home Folder is the place where Office Settings are configured. The Office Settings are explained below.

3. Copy configuration from another folder:

This will give the user the option to browse to the configuration settings in another folder and copy those settings for this folder.

4. Description of this folder:

The user can enter a helpful description here.

5. Standard Folders Tab

a. Document Save Folder:

This will be the default folder to which documents (produced by running a template) will be saved. When a template has been run, the user will be asked to provide a name for the document via a "save file" dialog, and at that stage will be able to change the place where the document is saved.

i. Pattern for saved file name:

When XpressDox has merged the data into a template, it will prompt the user for the filename into which the merged document is to be saved. If a pattern is defined in this field, then the default filename for the document will be the result of inserting data elements into the pattern. In the example, the values of the TemplateName and AccountNumber data elements in the data captured for the template are inserted into the pattern, to give a default file name. The <> around the data element names identifies them as such.

Please see 12 below for more information regarding document and data file paths and names.

ii. Format of merged documents:

Merged documents are by default saved as Word XML documents. The user can decide for each template folder the format in which the documents merged from those templates must be saved: either as Word, PDF or both. In the case of Word format, the documents need not be saved as XML, but, for each folder, the

user can choose to save in .doc or .docx (for Word 2007) format instead

b. Data Save Folder:

The data captured as part running a template will be saved (with a name similar to that provided by the user when the document is saved – with the extension .xddata.xml instead of .xml) in this folder. If the user chooses a different folder to save the document from the one configured as the Document Save Folder, then the data will be saved to this new folder – i.e. the same folder as the document.

i. Pattern for saved file name:

If the data file name is required to be different from that chosen by the system as in b above, then a pattern can be inserted in this field. This is similar to the "Pattern for saved file name" as described for the Document Save Folder in a above.

The section **Advanced file handling:** below provides more information on how to use the "Patterns" for saved file names.

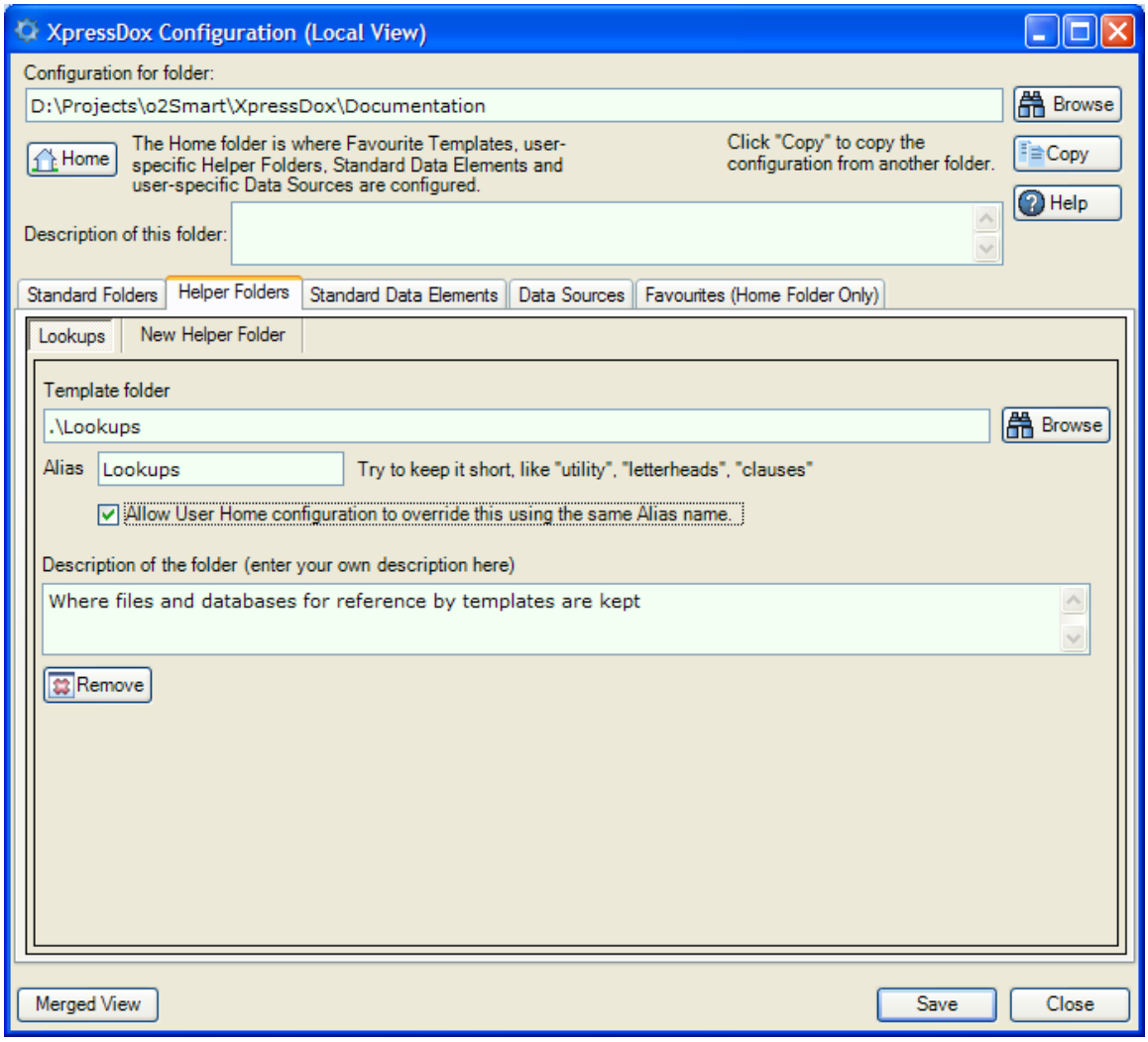
c. Clauses Library Folder:

When XpressDox finds an "InsertDocument" command (i.e. the command which inserts a document whose name is contained in a data element), it will prompt the user for the file name of the document, and will default the folder for that prompt to the Clauses Library Folder.

When this folder is defined for the Home folder configuration, then this folder becomes the "My Clauses" folder, which is used from the "My Clauses" toolbar button feature. This feature enables the user to choose a clause from the "My Clauses" folder, and that clause is inserted into the active Word document.

6. Helper Folders Tab:

Any number of Helper folders can be configured.



A typical example of a folder for which a shortcut is defined would be one in which letterheads or other standard templates like fax headers, etc., are kept. In this way these standard templates can be kept separately to the templates that use them.

a. Alias:

It would be possible to specify the full file path of the BaseTemplate in the command, i.e. something like:

<<BaseTemplate(C:\Documents and Settings\Dev\My Documents\XpressDox\Standard\LetterHead)>>

The file paths for these commands can become very long, and prone to typing mistakes, and so the concept of the "Alias" was introduced.

In a case like this, the Helper folder's alias would be something like "standard". A template in this folder would be referenced as in the following example:

<<BaseTemplate(standard:LetterHead)>>

Note that the template file name does not need an extension, the XpressDox extension ".xdtpl" will be used by default.

b. Description of the folder:

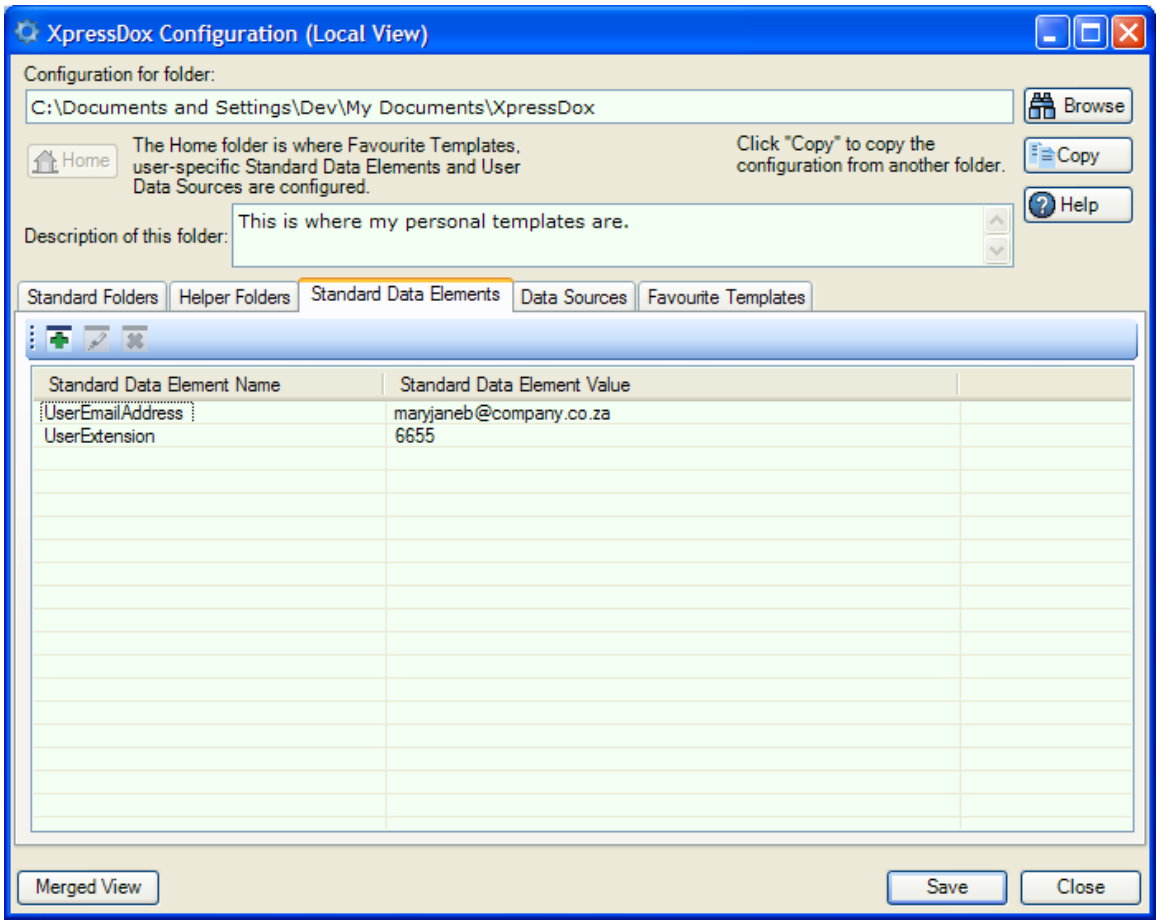
A free format description can be entered for reference sake so that in the case where someone has configured a number of Helper folders, they can remind themselves of what this particular one is for.

c. Allow User Home configuration to override:

When a template is from a folder, the list of helper folders configured for that template folder is joined to the list of helper folders in the user's Home configuration. If two helper folders from these two lists have the same alias, then by default the helper folder from the template configuration takes precedence over the same-named helper folder in the User Configuration, unless the "Allow User Home configuration ..." check box is checked, in which case the User Home configuration helper folder is allowed to take precedence.

7. Standard Data Elements:

These are data elements which appear as data elements on standard templates. A typical one is the user's email address or telephone extension on a letterhead.



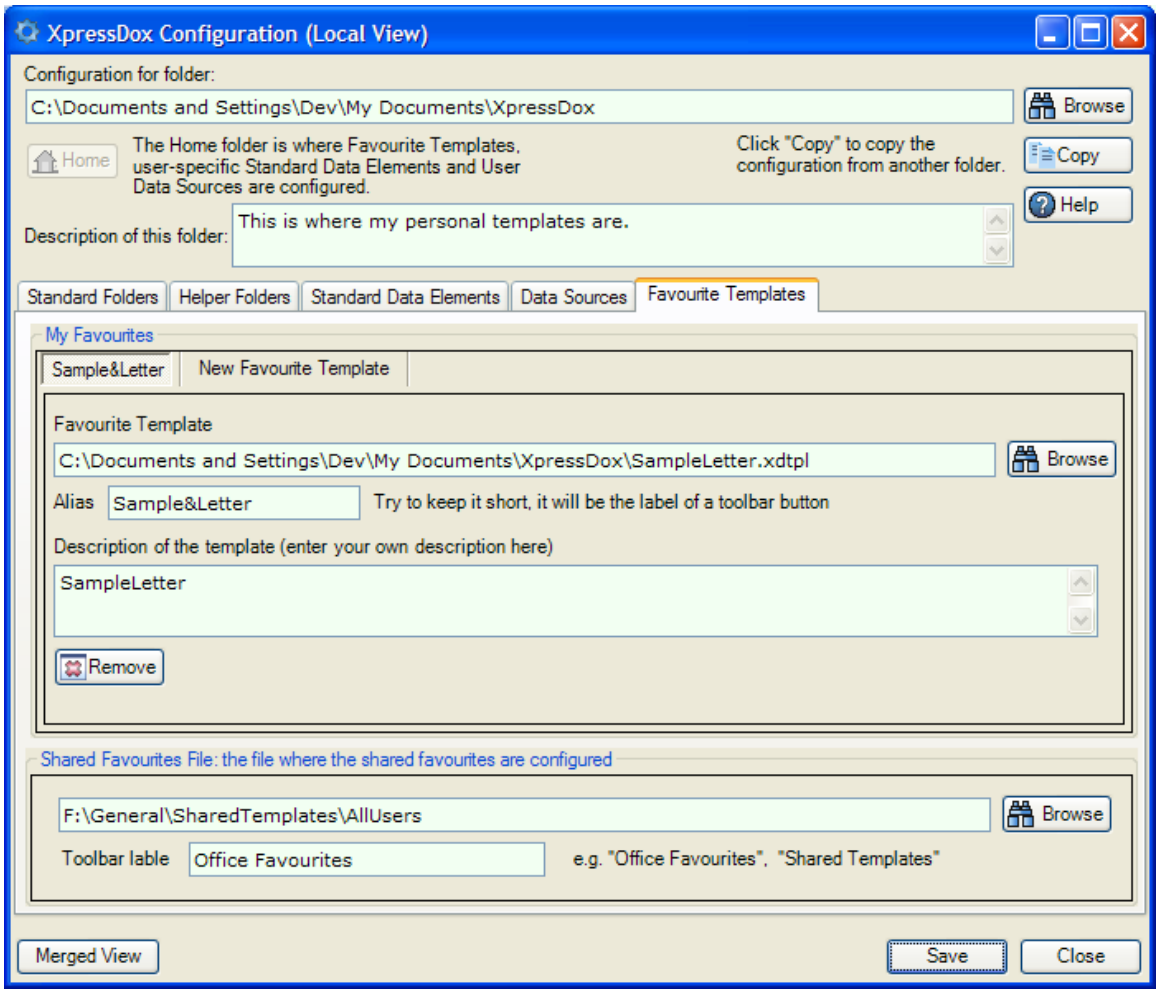
In order to save a secretary having to type their own details into the data capture UI every time they run a template, the Standard Data Elements

feature allows them to enter these details into the configuration of a folder. The values of the Standard Items are filled into any data capture UI in which the data elements with those names appear. At that point they can be left as-is in the data capture UI (which is presumably normally the case), or the user can change them for any specific template that is run. See "Merging configurations" below for the exact mechanism of how this works.

8. **Favourite Templates Tab:**

   This tab is only available for the Home folder. It enables the user to specify a number of templates that they use frequently, and these templates are then presented in the Word UI in a menu, so that the user can have quick access to them.

   Typically, users would enter their own favourites using the "Add to Favourites" toolbar button in the XpressDox toolbar (or ribbon in Office 2007). Favourites can only be removed from the list, or the alias or description altered, with the Configuration UI.



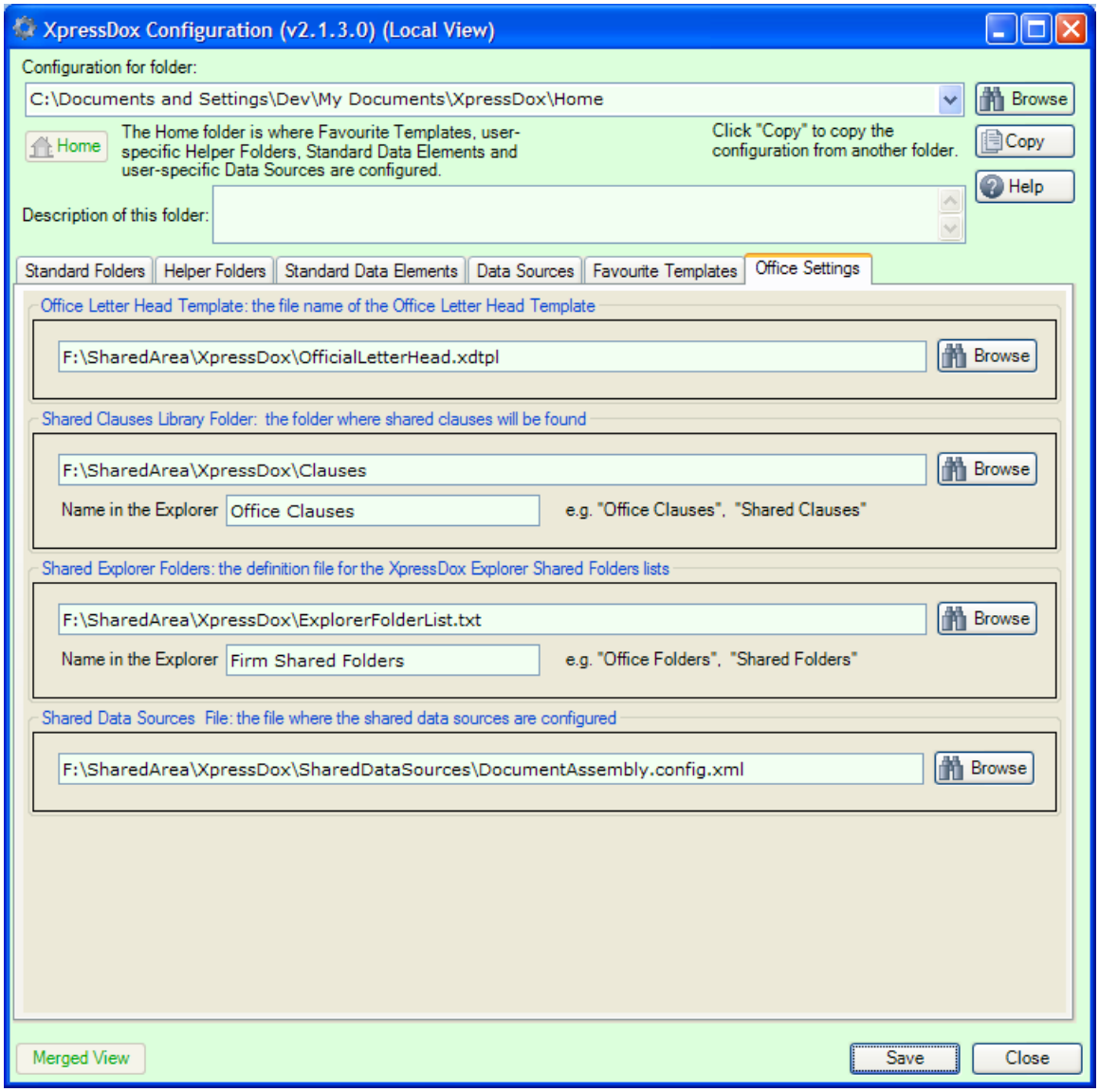
   a. The "Alias" is the short label that will be applied to the Word Ribbon Button (only in Word 2007). The "&" character is used by XpressDox to inform Word of the "Key Tip" shortcut key to be used for that function. In the absence of the "&", XpressDox will use the first character of the Alias as the shortcut key – but note in this example

that the first two favourite templates would then have the same shortcut key (viz. L).

b. The "Description" will appear as help text when the user hovers the mouse over the button.

c. Shared Favourites: these are only applicable to situations where there are many users who need to share the same templates. They would be configured by someone like a site IT Administrator. The mechanism is to configure them in the Home Folder of some or other user (maybe the Site IT Administrator themselves) and then copy the configuration file to the shared location.

9. Office Settings Tab:



a. Office Letter Head Template:

The Basic Template Author toolbar provides a function which inserts a BaseTemplate command into the template being authored, where the path to the template is the full file path configured in this area. This means that a template for the firm's letterhead can be saved in a centrally available locations, and all template authors will easily be able to refer to it in their templates.

b. Shared Clauses Library Folder:

This folder is configured in the users' Home Configuration. It is then used together with the "Office Clauses" toolbar button. When this feature is selected from within Word, then the XpressDox Explorer presents the contents of the Shared Clauses Library Folder to the user.

The user then selects a clause which is inserted into the active Word document.

c. Shared Explorer Folders:

Using the XpressDox Explorer, the user can create lists of folders under their "My Folders" entry in the Explorer treeview. A site administrator can set up folder lists which can then be made available to all users. The mechanisms for doing this is explained in the **XpressDox Supervisor's Guide**.

d. Shared Data Sources:

Data sources that can be shared by a number of users can be referenced from the user's Home Folder and then are available to all templates used by that user (as long as they are referenced in the relevant template with the IncludeDataSourceData or ChooseFromDataSource commands.

Shared data sources would be configured by someone like a site IT Administrator. The mechanism is to configure them in some or other folder and then copy the configuration file to the shared location. This is described more fully in the **XpressDox Supervisor's Guide**.

10. Merging configurations:

When a template is run from a specific folder (the "Source Folder"), the actual configuration settings used for that template (and, in fact, for all templates from that folder) are made up of the configuration settings saved in that folder, merged with any configuration settings for all the folders of which the source folder is a sub-folder. This means that if any of the Document Save Folder, Data Save Folder and any of the list of Helper Folders is missing from the Source Folder's configuration, then they are "filled in" from the configuration in the Source Folder's parent folder. And this continues up the chain of parent folders *mutatis mutandis*. The configuration thus arrived at is called the "Merged Configuration", and the chain of folders from the Source Folder up to its ultimate parent folder is called the "Configuration Chain".

This merging also applies to the Standard Data Elements – except in this case the Standard Data Elements configured in the Home Configuration are those which take precedence if there are any like-named items in the Source Folder's Merged Configuration. This is regardless of whether that Home Configuration folder is in the Source Folder's Configuration Chain or not.

Helper folder definitions and Data Source definitions are merged in a similar fashion. In these two cases, the Source Folder configuration takes precedence over the Home Configuration, unless explicitly overridden in the Source Folder configuration for a specific Helper Folder or Data Source.

11. No other configuration is needed:

Although other systems have concepts like "Private Folders" and "Shared Folders", these concepts are not necessary as such in XpressDox, as they can be provided for, and more flexibly, by the combination of the features of Merged Configuration and Helper Folders.

12. Advanced file handling:

As well as configuring the folders where documents and data are to be stored, the filename which will be given to the saved documents and data can be defined, using the "Pattern for saved file name" described in 5.a.i above. In addition to all of that, the template author has features which give even more fine-grained control over where documents and data are stored. These features are discussed here, even though strictly speaking they are not "configuration" issues, but they relate closely to the file locations which can be configured.

The features are controlled by the template commands SetDocumentSaveFolder, SetSavedFileName, SetDataSaveFolder and SetDocumentName.

a. SetDocumentSaveFolder:

This command (as with the others explained here) is entered in the template itself, for example:

<<SetDocumentSaveFolder(Documents\<AccountNumber>)>>

If the DocumentSaveFolder set in this way is a relative folder (as is the case with this example), then the folder configured into the Document Save Folder (i.e. the one discussed in 5.a above) is prefixed in order to arrive at an absolute folder path.

Note the <> characters around AccountNumber – these indicate that XpressDox must extract the value of the "AccountNumber" data element captured for the template, and insert that value into the folder name, giving a result, in this example, of "Documents\AA000123" (assuming the AccountNumber captured was AA000123). (Data elements enclosed in <> are called "File Path Merge Fields" in this document).

There are four "system values" which can be specified as File Path Merge Fields for the Advanced File Handling features, viz. TemplateName, DocumentName and WindowsLogonName.

(a) TemplateName is the name of the template file, excluding the path and the file extension. If the full path of a template is, say, C:\Documents and Settings\Fred\Templates\LetterToMom.xml then the TemplateName File Path Merge Field will have the value "LetterToMom".

(b) DocumentName is set by the template author using the SetDocumentName command within the body of the template.

(c) WindowsLogonName is the user name under which the user logged on to Windows.

(d) SourceDataFileFolder is the name of the folder in which a data file has been selected. If no data file was selected then the value of this File Path Merge Field is an empty string.

b. SetSavedDocumentFileName:

A file name specified with this command will override the "Pattern for saved file" in the configuration (if any). The saved file name can be relative or an absolute path, in the former case it is made absolute by applying either the path specified in a SetDocumentSaveFolder (on the

same template) or, if that is not specified, the configured Document Save Path.

An example would be:

<<SetSavedFileName(<DocumentName><PartyName>)>>

c. SetDataSaveFolder:

This command can be used to override the configuration setting of Data Save Folder (i.e. that described in 5.b above). The use of File Path Merge Fields and absolute and relative path naming are as described for the other commands in this section.

Note that the file name for the data is ALWAYS the same name as was applied to the merged document, but with the extension ".xddata.xml" instead of ".xml".

d. SetSavedDataFileName:

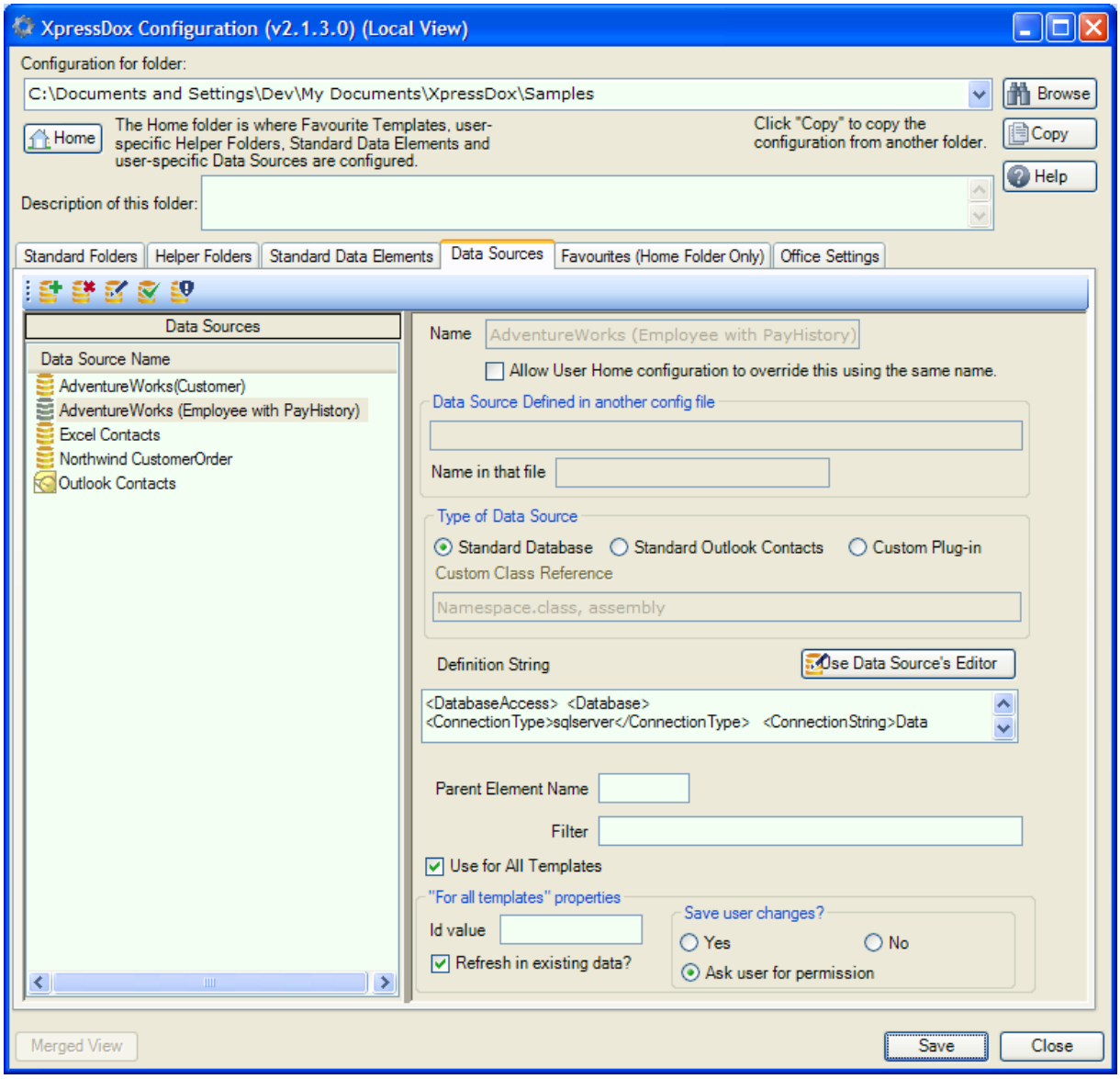
This functions in much the same way, for data file names, as SetSavedDocumentFileName functions for merged documents.

e. SetDocumentName:

This command enables setting a document name which can then be used to merge into a file name or folder name as the <DocumentName> File Path Merge Field.

It should be apparent that the Advanced File Handling features provide a powerful means towards designing a document production and management system which has many of the features of purpose-designed systems but with the simplicity of being defined and managed by the users.

13. Data Sources:



The Data Sources facility provides a way for data captured into some other system to be merged into templates.

For example, users may wish to be able to have address and other contact information from Outlook inserted into letters. This would be achieved by defining the Outlook Contacts as a Data Source. Then when a template is run, the user will be able to select that the Outlook Contacts data source be included in the template and will also be presented with a search interface to enable the relevant Contact to be selected. Thereafter the Outlook Contact data becomes available for insertion into the template, in exactly the same way as for any other data.

It would typically be the template author, not necessarily the user of the template, who would configure data sources in the system.

There are essentially three main types of Data Source:

- The Standard Database Data Source. The system provides a very comprehensive wizard-based configuration tool which enables the template author to configure Data Sources based on MS Access databases, Excel spreadsheets, SQL Server databases, and other database systems such as, for example, MySQL.
- The Standard Outlook Contacts Data Source. This provides access to the Outlook Contacts database, and can be configured to provide data from any Outlook Contacts folder: by default this is the current user's Contacts folder (and sub-folders).
- Custom Data Sources. This is a plug-in feature available to .NET developers who can then supply interfaces to any relevant data (such as accounting systems) where more sophisticated interaction with the data is required.

There are some concepts which apply to all Data Sources.

a. Name:

This is the name which commands such as <<IncludeDataSourceData(...)>> and <<ChooseFromDataSource()>> would use to refer to the data source.

b. Allow User Home configuration to override

When a template is run from a given folder, then the list of data sources defined in the configuration for that folder is merged with the list of data sources defined in the user's Home configuration. If two data sources have the same name, then the data source in the template folder configuration will take precedence over the same-named folder in the Home configuration. This behaviour can be modified on a per-data source basis by checking the "Allow User Home Configuration ..." checkbox for the data source in question.

c. The Definition String:

This is used to customise and configure a particular Data Source. For example, where a Data Source is a Standard Database Data Source, then the Definition String defines which database to use, and the tables, columns, search criteria, etc.

In the case of the Standard Outlook Data Source, the Definition String will be used to define the Outlook folder which must be looked up, as well as the Contact properties which should be presented in the data for insertion into templates.

The Definition String can be manually modified in the text box provided in the configuration UI. However, it may be that the Definition Strings have a complex structure, in which case the Data Sources themselves provide an editor which helps the user correctly construct the Definition String.

Some data sources, particularly some Custom Plug-ins, will have very simple (and perhaps empty) Definition Strings.

d. The Parent Element Name:

Particularly when a template may access data from more than one Data Source, there is the chance that there may be a clash of data element names. For example, an Employee database may be used to provide data about the person running the template (e.g. FirstNames, Surname, BusinessPhone) and these data elements would have the same names as similar data elements in the Outlook Contacts data source. To resolve this conflict, one or both of the data sources could have the Parent Element Name set, and this would be used in the template to resolve the name clash.

For example, the Employee data source could have the Parent Element Name set to "Employee", and then employee Merge Fields in the template would be something like:

<<Employee/FirstNames>> <<Employee/Surname>>

e. Filter

The filter can be used to limit the set of rows returned by the data set. For example, a filter of "Department=Personnel'" would typically restrict the data source to data only for the "Personnel" department.

f. Use for All Templates:

Suppose all templates are to contain signature information of the person running the template. This can be achieved in at least two ways:

1. The signature information (name, email, telephone extension) can be kept in each user's Standard Date Items in the configuration for their Home folder.
2. It may be that the company holds all that information in a central database. That database can then be defined to XpressDox as a Data Source, and
   a. each relevant template could then have a <<IncludeDataSourceData(Employee DataSource)>> command entered on it; OR
   b. In each user's Home configuration, the Employee DataSource could be defined with the "Use for All Templates" flag set, and the Id Value for that user's information in the Data Source entered. Then, without any further action on the user's part, their own Employee information will be made available for every template that they run.

The "Use for All Templates" indicator can be set on data sources in folders other than the user's Home folder. These data sources are then made available on all templates run from that folder only.

When a Data Source is used on all templates, the "Refresh" and "Save" options can be configured as for the IncludeDataSourceData and ChooseFromDataSource commands.

g. Test Data Source:

The "Test Data Source" toolbar button() will optionally present the Data Source's Search interface, and then show the selected XML data in a window. This is to help the template author get a view of what the data will look like, as well as show whether and how the Data Source user interaction works.

h. Prepare Schema:

When authoring a template, especially one in a suite of templates which will use the same data, it is difficult to remember what data elements have been used before. To help with this, the Template Author's Toolkit permits the author to choose specific templates which contain data elements which are to be re-used. It is also possible to choose a "schema" for a Data Source in the same way.

In order to generate this schema for subsequent use in the template Author's Toolkit, either use the Test Data Source feature above (which gives the option of producing a schema based on the data selected during the test), or use the "Make Schema" toolbar button ().

i. Data Source Defined in another config file:

This feature permits a data source defined in the configuration for one folder to be referenced from the configuration for another folder. This means that if the data source definition changes (e.g. if the server name or IP address of a data base changes) then the change need only be made in one place.

The data source in the referring configuration can be given a name different to that in the defining configuration. Hence the "Name in that file" field.

14. Choose your own field markers

The strings << and >> are used in this document as the markers which define a Merge Field in a template.

It is possible for these markers to be re-defined on any given installation of XpressDox. The decision to do this needs to be taken seriously, because as soon as any significant number of templates have been authored using a particular choice of Merge Field markers, a change to the markers will require a change to all the templates authored up to that point.

The Merge Field markers are defined in the file `SiteSettings.xml` which is in the folder where XpressDox was installed. This file is in XML format. It can be opened using Notepad, and the values of the <fieldStart> and <fieldEnd> XML elements changed, and the file saved again.

Some issues to bear in mind when doing this:

a. Be careful when modifying the `SiteSettings.xml` file.

b. It is best to use a pair (or more) of characters as the FieldStart and FieldEnd, and not just one character. For example, use *(and)* rather than (and). In fact use of some characters will cause strange behaviour, or cause XpressDox to fail in some situations ([and] are a case in point). Note that this applies only to the case where the [and

] are used on their own as start and end field markers. If used in conjunction with any other character (e.g. ![ ]!) the strange behaviour/failure will be avoided.

c. One character for FieldStart and one for FieldEnd CAN be used if characters are chosen which are not ever going to be used in the normal course of events in your particular environment. An example would be the "guillemot" characters « and » . The difficulty with this choice of characters is that they are not easily typed on the keyboard. On the other hand, the Template Author's Toolkit and toolbar in Word will assist greatly in this case.

Template Author's Toolkit

1. The Left Hand Tree View:

The left hand tree view of the Template Author's Toolkit has a large collection of commonly used Merge Fields, including formatting functions of various types.

Double-clicking an "Example" in the left hand tree view will cause an example of the Merge Field to be inserted into the template, with the data element name (usually) being selected so that it can be replaced with the actual data element name. If there is selected text already in the template when an "Example" is double-clicked, then the selected text is (usually) used to replace the name of the data element in the example.

Wizards work in a similar fashion, but instead of providing just an example, they will provide a place for all the parameters in the Merge Field to be completed.

2. The Right Hand Tree View:

The right hand tree view will initially contain a list of commonly used data element names. This list is defined in the template `CommonDataElementNames.xdtpl` which is installed with XpressDox in the `My Documents\XpressDox` folder. This demonstrates the feature whereby any XpressDox template can be included into the right hand tree view as a "schema", and then the data elements which were defined in that template are then available for insertion into the template currently being authored.

The `CommonDataElementNames` template can be modified to contain only the data elements that you require. Or, the template can be deleted (or renamed) to prevent it being loaded every time into the Toolkit.

When some data sources have been defined, and schemata for them created, the data source schemas can also be included in the right hand tree view to help the template author select correctly named data elements.

Double-clicking a data element will create a Merge Field containing that item. Or, if there is text selected in the template being authored, then the double-click will result in the selected text being replaced with the data element.

If the Shift key is held down while double-clicking, then the data element name is inserted into the template – i.e. no Merge Field is created. If any text is selected when the "Shift-Double-Click" is performed, that text is replaced by the data element name.

If a data element is dragged from the right hand tree view and dropped onto a Merge Field in the left hand tree view, then the data element is inserted into the relevant position in the Merge Field and the result included in the template.

If the Ctrl key is held down during a Double-click or a Drag-and-drop, then the fully qualified data element name will be inserted. This applies to data elements which are contained in parent elements in the data (i.e. either the parents are repeated elements or when the data source was created it was defined with a "Parent Element Name" supplied).

A fully qualified data element name would be, e.g. "Contact/Fullnames", or "InvoiceEntry/Amount". The unqualified names in these two cases would be, respectively, "Fullnames" and "Amount".

3. The Toolbar:

The toolbar has buttons for the following features:

a. Define the font and colour of the merge field text. This is for legibility. By default the font is Arial and the colour blue. The size of the text inside the merge field varies with the size of the surrounding text.

b. Format all merge fields in the current template with the defined font and colour. This allows the template author to type the merge fields "by hand" and then apply the defined font and colour to all of them.

c. Convert a template from another document assembly system. This feature will enable conversion of Word merge fields to XpressDox format, and also assists in converting templates marked up with codes form other document assembly systems.

d. Add a schema or template to the right-hand tree view, and also remove a schema or template from that tree view.

e. A Help button – causes the document you are reading to be presented in an easily searchable form. This same document is presented, in context, when the F1 button on the PC's keyboard is pressed.

The toolbar also includes a search function, which searches the left-hand tree view for the text typed into the text box on the toolbar. The <Ctrl-f> key combination is a shortcut to that text box.