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

This document contains various "recipes" for different aspects of XpressDox. It is intended to complement the User Reference. The latter document is comprehensive in that it covers all of the XpressDox commands and functions, but does so to a limited depth. The Cookbook is designed to give in-depth coverage of certain commonly used aspects.

The articles in the Cookbook are not arranged in any particular order, although hopefully the first few will be of immediate use. One topic which qualifies for first slot is actually at number 18. It would probably be helpful to visit that section early on.

The Cookbook is likely to be under construction for quite some time, if not always. We will especially take time to add to it when users report having trouble with any particular concept, or have requirements which are covered by XpressDox but are not apparent at first glance.

1. Create a letterhead

Firstly, in Word, create the letterhead document (e.g. with the company logo and contact information). You can probably just copy the existing company letterhead, if you have one.

The Merge Field <<DocumentBody>> must be placed in the document at the point where the body of the document will appear.

Now save this letterhead document as an XpressDox template (using the  button on the XpressDox toolbar).

- The letterhead can now be run as a template on its own, and the resulting merged document can be edited manually to insert the body of the document.
- If you have a number of standard letters which need to appear on the letterhead, then these are created as XpressDox templates with the text of the letter and relevant Merge Fields, such as <<Addressee>>, <<Address>>, <<Greeting>> (i.e. Dear ...), <<Salutation>>, in each of the standard letters.

  In the first line of the standard letter template, put the command <<BaseTemplate(Letterhead)>>. Then save the template (in the same folder as the letterhead) and run it – the text of the template, with completed Merge Fields, will appear inside the letterhead.

The `My Documents\XpressDox\Samples` folder has some examples of letterheads and templates that are based on them.

See the section "Use the same letterhead on many different templates" for more advanced uses of the BaseTemplate command.

2. Make it easy for all templates to use the same letterhead.

This is where the Helper folders in the configuration come into play.

The concept to bear in mind is this: when a template is run by XpressDox, the internal environment is set up according to the configuration of the folder in which that template was saved (i.e. the folder where the template is run from). In other words, every time a template is run, XpressDox makes sure that it's containing folder's configuration is active.

To make sure that those templates which require a letterhead all use the same letterhead, there are a few things to set up:

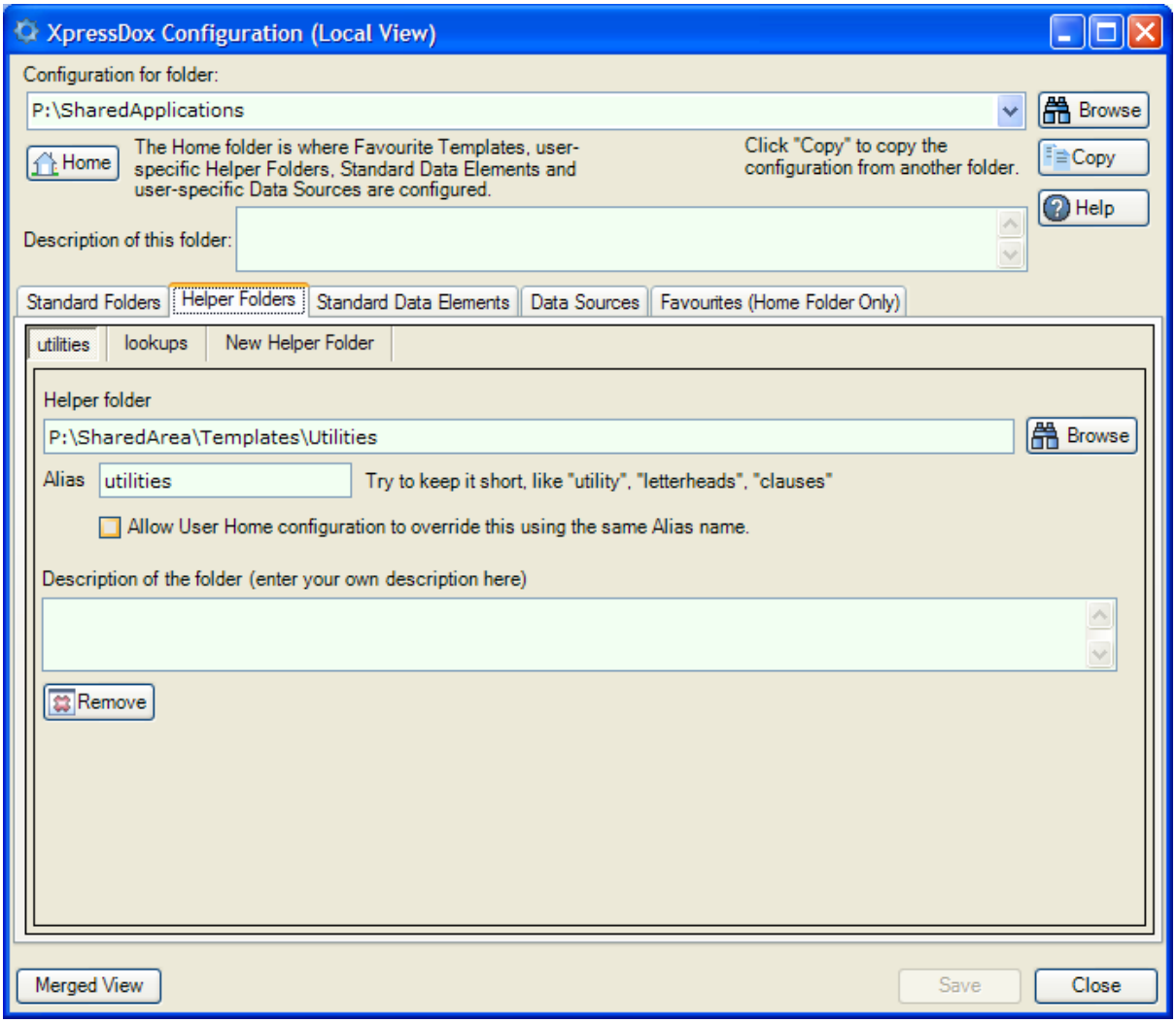
- First, the letterhead template must be created, and must be saved in a suitable location (in the case of a multi-user environment, a suitable location would be a shared folder on the network). For this example, suppose that

location and file name is
`P:\SharedArea\Templates\Utilities\Letterhead.xdtpl`

- A short name (Alias) must be decided on for the folder in which the `Letterhead.xdtpl` template is stored. In this example, "`utilities`" would be a good choice (note that the Alias does not *have to be* the name of a folder or sub-folder, so names like "`basetemplates`", "`docs`", etc. are equally acceptable)
- Every template requiring that letterhead must have the following command <<BaseTemplate(utilities:Letterhead)>> on the first line (on its own in that paragraph) of the template.
- When a template with the above <<BaseTemplate(…)>> command is run, XpressDox will look in the configuration for that template's containing folder to find the path to where the `Letterhead.xdtpl` template can be found. In other words, it will look for a definition of the Alias "`utilities`". The Alias is defined in the Helper Folders tab in the template folder's configuration. This is done as follows:
    - Run the XpressDox Configuration, select the template folder as the "Configuration for folder", and select the Helper Folders tab;
    - If there is no Helper folder with alias "`utilities`", then create one by pressing the "New Helper Folder" tab and enter the path (in this example `P:\SharedArea\Templates\Utilities`) and alias ("`utilities`").
    - Save the configuration.

This means that regardless of where a template is saved, as long as the folder in which it is saved has the "utilities" Helper Folder configured to the Letterhead template's containing folder, and the BaseTemplate command refers to the template as "`utilities:Letterhead`", then the correct letterhead will be used.

The configuration screen for a typical template folder which was configured as above would look something like:



3. Suppress Empty Data

Suppose you have an address captured as four separate data elements. Typically the Address section of a letter would then look like:

<<AddressLine1>>
<<AddressLine2>>
<<AddressLine3>>
<<AddressLine4>>
<<PostalCode>>

But sometimes the fourth, or maybe even the third, lines are empty, and you normally don't want to leave an empty line between the last non-empty address line and the Postal Code.

Here is what to do:

<<AddressLine1>>
<<AddressLine2>>
<<If(AddressLine3 != "") the symbol != means 'is not equal to'>>
<<AddressLine3>>
<<End()>>
<<If(AddressLine4 != "")>>
<<AddressLine4>>
<<End()>>
<<PostalCode>>

Notice that when an If command is the only text in a paragraph, then that paragraph will not appear in the merged document. The same applies to the End command (and numerous other commands, e.g. ForEach, Else, the various ChooseFrom commands, etc.) which appear all on their own in a paragraph.

4. Provide Default Values

Often there is a need to require that the user provide values for data elements, but that those data elements usually have the same value and only change for exceptions.

An example might be a discount percentage, or an amount such as a consultation fee. A typical paragraph might read:

The fee for this consultation is R<<FormatNumber(ConsultationFee)>>, but a discount of <<DiscountPercent>>% will be applied if the account is settled in full within 14 days of receipt hereof.

When a template with this paragraph is run, then the user will be presented with capture fields for ConsultationFee and DiscountPercent, but by default both of these fields will be empty.

Providing default values for those fields would be achieved as follows (assuming the default for the fee of R450 and the discount as 10%):

a. Create a text file which has two lines in it, and looks like this:

```
ConsultationFee,DiscountPercent
450,10
```

b. Save the file in the same folder as the template, and call it `Defaults.xdtxt`.

c. In the template itself, put the following command in a paragraph by itself:

<<IncludeFileData(Defaults.xdtxt,norefresh)>>

Those defaults will be applied when the template is run with "New Data". (And the "`norefresh`" parameter ensures that when the data set for this template is used subsequently, the default data from the defaults file will not be refreshed in the data set. For a fuller explanation of "`norefresh`", please see the User Reference and also example 17 below).

A note about text files: these can be created using NotePad, or even MSWord. When creating a text file in MSWord, type the file as you normally would, and then save it using the "Save As" feature, and choose "Plain Text (*.txt)" in the "Save as type" drop-down box.

5. You work for three bosses

You want to make sure that, when you run a letter template, that the correct boss's contact details appear on the letterhead.

Suppose your letterhead template has data elements <<DirectorName>> <<DirectorPhone>> <<DirectorEmail>> placed in the relevant locations on the letterhead.

The problem of getting the correct "Director..." information into these data elements has a number of solutions, but a simple one would be:

a. Create a text file formatted like this:

```
DirectorName,DirectorPhone,DirectorEmail
Frederick Basset,999-888,fbasset@company.com
Charles Brown,888-777,brownc@company.co.za
Michael Rowboat,555-444,mrowboat@company.com
```

Obviously with your own data in it – one line for each boss; but note that the names in the first line must be the same as the names of the data elements in the letterhead.

b. Save the file, either in the same folder as your templates are run from, or in a helper folder, and call it something like `BossData.xdtxt`. (N.B. if you create the file in MSWord, then it must be saved using "Save As ..." and choose the "Plain Text" option).

c. In any template which refers to the letterhead via a <<BaseTemplate()>> command, include a command:

<<ChooseFromFile(BossData.xdtxt)>>

Now, whenever you run a template, you will be presented with a drop-down listing Frederick Basset, Charles Brown and Michael Rodent. Choosing one of them will cause the associated data elements to be included in the data set for the template, and the correct values will be filled in to the relevant merge fields.

6. Working with Help.

Help text can be supplied for any data element, which means that in the data capture UI, when the user hovers the mouse over the field for that data element, then the help text is displayed in the help are in the UI.

There are two ways of providing help text for an element:

- In the merge field, before the closing field markers (i.e. >>), place the help text preceded by a question mark:

  <<AccountNumber?Enter the account number>>

- Use the <<Help>> command:

  <<Help(AccountNumber,Enter the account number)>>

Now, it might be that you have a number of templates in a template folder which are all related (in fact this will most often be the case), and it can become quite tedious to type the help text into all the relevant merge fields in every template.

As it happens, XpressDox handles the help text in such a way that if help is provided for a data element which is not captured for a particular template, then it is just ignored. This means that a time saving way of sharing the help text among all related templates would be to type all the <<Help>> commands for the relevant merge fields into a special template reserved only for help text, and then <<IncludeTemplate>> that help template into all the other templates.

To make sure that no formatted Word text inadvertently gets included from the help template, use the command <<IncludeTemplateText(HelpFile)>> (i.e. IncludeTemplateText rather than IncludeTemplate – the "Text" command includes only text, no formatting).

7. Working with repeated elements.

You have an application where a particular set of data are repeated. An example would be a Will where the names of the children are repeated. A typical template would have this snippet:

> My heir<<When(count(Child) > 1,s are, is)>> my child<<When(count(Child) > 1,ren)>>, viz. <<Foreach(Child)>><<When(position() > 1, AND)>><<Firstname>> <<Surname>> (age <<Age>> years) of <<AddressLine1>>, <<AddressLine2>><<End()>>

The above would appear somewhere in the Will and would cause each child's Firstname, Surname and 2 lines of address to be captured for that Will (with extra logic to control layout of the Address but this is left out so as not to obscure the point of this example). And the layout of the result in the merged document would

have the word "AND" (with a space on either side) preceding every heir's description except for the first.

During the course of the winding up of the estate, it might become necessary to write to each of the heirs separately, and for this purpose one or more letter templates are set up where the addressee information has to refer to just one of the heirs.

The simplest way of marking up the addressee portion of the letter template would be:

```
<<Child[$ChildNumber]/Firstname>> <<Child[$ChildNumber]/Surname>>
<<Child[$ChildNumber]/AddressLine1>>
<<Child[$ChildNumber]/AddressLine2>>
```

- This would cause XpressDox to include an area for the capture of a data element called "ChildNumber".
- The square brackets [] notation causes a process which could be called an "array element selection". Because the ChildNumber data element is captured as a number, it will cause the Child element with that number to be selected.
- The / before a data element name indicates that the data element after the / is a sub-element of the preceding element, the "parent" element.
- When data elements are referred to within the scope of a <<ForEach>> command, then the parent data element is inferred as the data element defined in the ForEach command. Hence in the first snippet, it wasn't necessary to refer to the Child's Firstname element as <<Child/Firstname>> (in fact it would be incorrect to do so).

Other ways of using the square bracket notation would be the following, where the names of the minor children (i.e. those with ages less than 18 years) are to be listed:

```
My minor child<<When(count(Child[Age < 18]) > 1,ren are, is)>> <<ForEach(Child[Age < 18])>><<When(position() > 1, AND )>><<Firstname>> <<Surname>><<End()>>
```

Referring to a repeated element when you know at template authoring time what its position is, is a simple case of typing in a constant value for the position:

```
My firstborn heir is <<Child[1]/Firstname>> <<Child[1]/Surname>>.
```

And, just for completeness, the last child would be:

```
My laat lammertjie is <<Child[last()]/Firstname>> <<Child[last()]/Surname>>.
```

At this point it makes sense to discuss another aspect of repeated elements: what if, from within a <<ForEach(Child)>> you want to refer to a data element that is NOT a sub-element of that Child data element? This is a pertinent question because the last bullet point above says that within the scope of a <<ForEach(Child)>> the parent data element of each data element referenced (i.e. Firstname, Surname, etc.) is inferred as the Child data element.

To illustrate this, using the example of the Will, it might be that the testator wants to indicate that each heir should be given the same fixed amount of money, but for some reason (if only for this example), to be totally pedantic the drafter of the Will wants to list the heirs by name and next to each name reflect this one fixed amount. We would probably expect a template snippet to look something like this:

```
<<ForEach(Child)>><<When(position() > 1, AND )>>To <<Firstname>> <<Surname>> the amount of R<<Share>><<End()>>.
```

However, seeing a ForEach like this, XpressDox would want to capture a separate Share data value for each Child, which is not what is intended here.

What IS intended is that the Share is captured once for the entire Will template, and that this data element is referred to in the ForEach. This can be achieved in one of the following ways:

- If the Share data element is referred to in a Merge Field anywhere in the template but NOT inside a <<ForEach(Child)>> then XpressDox will capture it in the data set at the correct level (i.e. the level of the parent data element of the Child data element).
- Otherwise, XpressDox can be forced to capture the Share data element at the correct level in the data set by use of the command <<CaptureDataElement(Share)>>.

Then, the previous snippet must be changed to read:

> <<ForEach(Child)>> <<When(position() > 1, AND)>>To <<Firstname>> <<Surname>> the amount of R<<**../**Share>><<End()>>.

Note the ../ notation – it is very reminiscent of the Windows (and DOS) file path notation, i.e. where ../ indicates "move up to the previous folder in the file path". In XML terms it means "move up one level in the data element path".

8. Doing calculations.

Simple calculations are simple to perform. The arithmetic operators are the normal ones except for division:

`+`, `-`, `*` (multiplication) and `div` (division).

Some examples would be:

> You owe me <<FormatNumber(Debt)>>. Pay within 5 days and I will give you a 10% discount – i.e. you will only need to pay <<FormatNumber(Debt * 0.9)>>.

A more formal way of calculating the amount less 10% would be <<Amount – (Amount * 10) div 100>>.

One thing to be careful of is to make sure that the arithmetic operators have at least one space on either side of them. Try it and see what happens if you leave out the spaces.

Suppose we have a template producing an invoice, or statement. The data could be located in a data source, or be captured by the user. It makes no difference where the data come from, the template and commands will be the same. Here's a snippet:

Product	Quantity	Unit Cost	Total	Total In VAT
<<ForEach(Transaction)>><<Product>>	<<Qty>>	<<UnitCost>>	<<UnitCost * Qty>>	<<FormatNumber(UnitCost * Qty * 1.14)>><<End(foreach)>>
Total				

9. Using variables.

People are often tempted to use the term "variable" for what XpressDox refers to as a Data Element. The distinction is an easy one: a Data Element has a constant

value throughout the template's lifetime – the value can only be changed when the user next gets a chance at it in the data capture UI, otherwise, once it's captured, that's its value.

XpressDox does, however, have variables: i.e. those data items whose value can change while a template is running.

Already in the previous example we have a need for a variable – an item whose value is going to change as the template gets run. This is in order to calculate the totals. The snippet to do this would be:

<<SetV("Total",0)>>

Product	Quantity	Unit Cost	Total	Total In VAT
<<ForEach(Transaction)>><<Product>><<SetV("TranCost",UnitCost * Qty)>><<SetV("TranCostPlusVAT",GetV("TranCost") * 1.14)>><<SetV("Total",GetV("Total") + GetV("TranCostPlusVAT"))>>	<<Qty>>	<<UnitCost>>	<<GetV("TranCost")>>	<<FormatNumber(GetV("TranCostPlusVAT"))>><<End(foreach)>>
Total				<<FormatNumber(GetV("Total"))>>

Here the extended cost ("unit cost times quantity") is also calculated and saved in a variable (variable "TranCost"). This variable's value is then multiplied by 1.14 to get the VAT inclusive amount ("TranCostPlusVAT"), and this VAT inclusive amount added to the running total, i.e. the variable "Total".

The <<SetV>> function sets the value of a variable, and <<GetV>> gets and returns the value of a variable.

10. Set up a multi-application environment saving merged documents and captured data in appropriate locations.

You are the site administrator for a large legal firm, and you have set up a number of XpressDox applications. Each application has its own set of templates which are located in a folder for each application. Say for example there is an Estates application, a Litigation application and a Conveyancing application, and these are in the folders `P:\SharedApplications\Estates`, `P:\SharedApplications\Litigation` and `P:\SharedApplications\Conveyancing`, respectively. Drive letter P is mapped to a shared folder on the network.

The default location for merged documents as well as captured data files is the same folder as that in which the template is located. Clearly this is a good place to start off, but for more sophisticated applications this is not really appropriate.

What this example will show is how to configure the three applications so that when a template is required for a particular client, then the template will be selected and run, and the merged document will be saved in a sub-folder of the application folder, where that sub-folder's name is somehow related to the name of the client for whom the template is being run (i.e. the sub-folder will have the client's name and/or account number or file number). The name of the merged document file in that sub-folder will be the same as the template file name.

The data for each client should be saved, in a similar manner, but all data for a client will be kept in one file (regardless of the template that the data were captured for) and that data file's name should also somehow related to the client's name.

Let's assume that the merged document sub-folder will be `MergedDocuments`, and that the saved data sub-folder will be `SavedData`.

Let's further assume that the way of identifying the client is via the firm's File Number which is assigned to that client (or to the particular case that is being worked on for that client). This file number needs to be captured for each template being run, and so the Merge Field <<FileNumber>> will appear on each template. You can insist that this data element is never left blank by using the command <<Required(FileNumber)>> rather than the simple Merge Field.

Now it's a case of configuring the three applications so that their merged documents and captured data are saved to the correct locations.

An example would be to configure the Standard Folders for the `P:\SharedApplications\Estates` application as follows:

Document Save Folder:
`P:\SharedApplications\Estates\MergedDocuments\<FileNumber>`

Pattern for saved file name:
`<TemplateName>`

Data Save Folder:
`P:\SharedApplications\Estates\SavedData`

Pattern for saved file name:
`<FileNumber>`

Then in the same way, the Litigation application could be configured as:

Document Save Folder:
`P:\SharedApplications\Litigation\MergedDocuments\<FileNumber>`

Pattern for saved file name:
`<TemplateName>`

Data Save Folder:
`P:\SharedApplications\Litigation\SavedData`

Pattern for saved file name:
`<FileNumber>`

However, XpressDox's "configuration merging" feature, together with the Windows File System's relative path addressing mechanism, makes this multi-application configuring a much simpler task.

So, instead of the above configuration settings being done, the configurations for the separate application folders should be left untouched – i.e. empty.

Then, the applications' common parent folder, i.e. `P:\SharedApplications` is configured instead, so that *its* Standard Folder Configuration looks like this:

Document Save Folder:
`.\MergedDocuments\<FileNumber>`

Pattern for saved file name:
`<TemplateName>`

Data Save Folder:
`.\SavedData`

Pattern for saved file name:
`<FileNumber>`

As long as the application folders' Standard Folder configurations have no entries in them, the merged configuration setting of the parent folder will apply, and the same

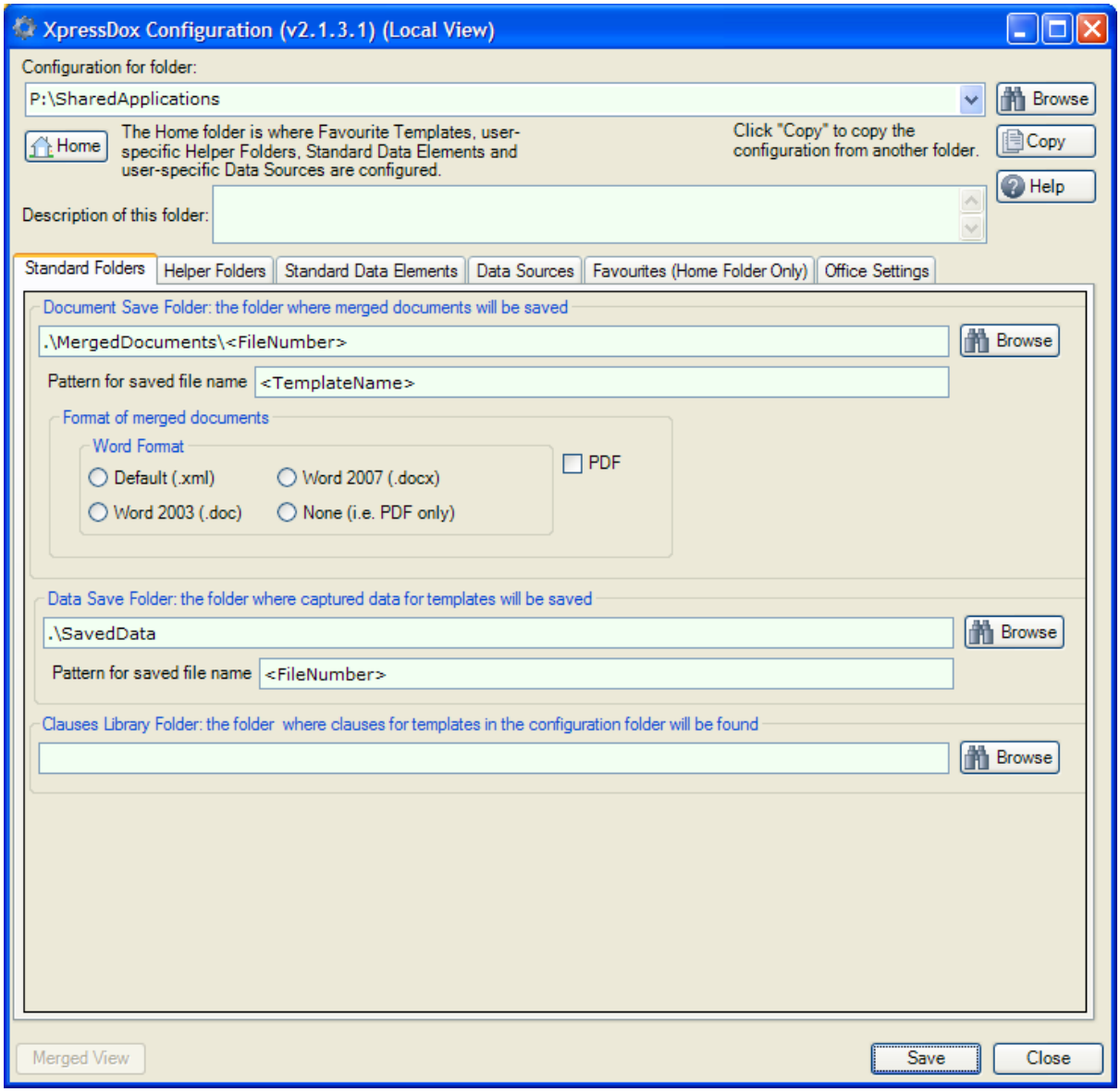
result will be obtained: i.e. the merged documents will be saved under the application's template folder in a folder called `MergedDocuments`, etc.

This will also mean that a new application added will inherit that same Standard Folder configuration automatically.

Lastly – the above won't really work unless users capture the correct FileNumber value when running a template. You can insist at least that a non-empty value is filled in by including the command <<Required(FileNumber)>> in each template – when XpressDox sees this it will not accept the "OK" button in the data capture UI until the user has supplied a value for the required data element(s).

Very lastly, the mechanism of providing help text for relevant data elements in all templates (as described in example 6) also applies to the <<Required>> command.

The configuration screen for folder `P:\SharedApplications` would look like this:



11. Make sure that the same letterhead template is used by all the letter templates in example 10.

Firstly, you will have set up a letterhead template as in example 2, and it will have been stored with a name like `P:\SharedArea\Templates\Utilities\Letterhead.xdtpl`.

All that's needed now is that the configuration for `P:\SharedApplications` (the one used in example 6) is configured so that it has a Helper folder with suitable Alias (in example 2 it was "utilities", and all the <<BaseTemplate>> commands referred to the letterhead template with the "utilities:" prefix), and which refers to the folder where the letterhead template is.

The configurations for the application folders (i.e. `P:\SharedApplications\Estates`, `P:\SharedApplications\Litigation` and `P:\SharedApplications\Conveyancing`) inherit the Helper Folders from their parent folder(s) in the same way as they inherit the Standard Folders. That means that as long as the application folders leave their Helper Folders unconfigured (or at least do not have a Helper called "utilities"), then they will inherit that "utilities" Helper from their parent folder.

All that is then needed is to make sure that the various templates (in the application folders) that will be based on the letterhead all refer to the letterhead template via the alias "utilities", i.e. <<BaseTemplate(utilities:Letterhead)>>.

12. Managing contact details on letters.

This is a variation of example 5, but on a larger scale.

It is supposed that you manage the templates for a large company with a large number directors and secretaries. Some secretaries work for more than one director, and others for only one (or the "secretary" is a director - i.e. by "secretary" we really mean the person running a template).

Company letterheads typically have a standard way of indicating the responsible person's contact details. In this discussion the "responsible person" is a director.

That means that typically a letterhead template would have a section dedicated to the director's details. A snippet of a letterhead might look like this:

> Yours sincerely
>
> _____
>
> <<DirectorName>>
> Extension: <<DirectorExtension>>
> Email: <<DirectorEmail>>

The issue is that we need to make sure that whoever is running a template that requires the director's details is able to easily provide them. (Example 5 used a text file with directors' details, and this is suitable for a small site where control is easy).

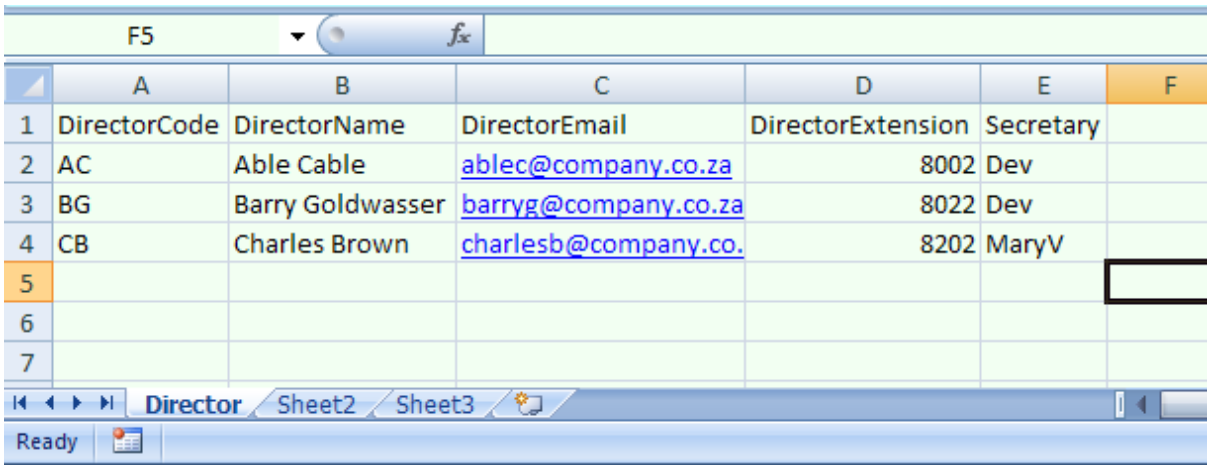
We need to have a database which is available to all users (i.e. people running templates) which can be maintained and kept up to date by a site administrator. As far as XpressDox is concerned this can be just about any database, but for purposes of discussion, this will be an Excel spreadsheet.

The workbook should have a spreadsheet called "Directors" and the first row should be the names of the data elements as they will appear in the template (i.e. in our example "DirectorName", "DirectorExtension", "DirectorEmail", etc.). It is a good idea to have a short code (e.g. the director's initials) which can be used as the "ID" of the data table, so a further column heading "DirectorCode" should also be

included. (In fact if you make this the first column in the spreadsheet then the data source wizard referred to below will treat it in the way we would like).

The rows under the first will then each contain the details for one director. There is a column called "Secretary", and this needs to contain the Windows logon user name of the person who will run templates for the director concerned.

The spreadsheet would look something like the screen shot below (although ultimately will contain many more lines):



	A	B	C	D	E	F
1	DirectorCode	DirectorName	DirectorEmail	DirectorExtension	Secretary	
2	AC	Able Cable	ablec@company.co.za	8002	Dev	
3	BG	Barry Goldwasser	barryg@company.co.za	8022	Dev	
4	CB	Charles Brown	charlesb@company.co.	8202	MaryV	
5						
6						
7						

Once the spreadsheet is complete it can be saved in a suitable location. For this example, suppose it is saved in exactly the same place as the letterhead template (at least it will be a shared location accessible by all users, and might also have suitable permissions to prevent unauthorised users from changing it).

The spreadsheet then needs to be made available to XpressDox by configuring it as a data source. The place to do this in this example would be the parent folder used in examples 7 and 11, i.e. the folder `P:\SharedApplications`.

Configuring a data source is done in the "Data Sources" tab in the configuration UI for the relevant template folder (i.e. `P:\SharedApplications`), as follows:

- Press the "New Data Source" button  . This starts a wizard.
- Choose "Excel Spreadsheet" from the list and press Next.
- Press the Browse button and locate the spreadsheet. (The wizard will construct a "Connection String" which can be left as-is). Press Next.
- Choose the Director spreadsheet from this list, and accept the suggestion of "Director" as to how you will refer to the table in XpressDox. Press Next.
- The wizard will suggest that "DirectorCode" be used as the "ID" of the table – accept that by pressing Next.
- The wizard now needs to know whether users must be able to select single rows from this spreadsheet (now called a "table" because it is being treated as a database) – the answer in this case is "yes", and so click the relevant option and press Next.
- We are using the standard way that XpressDox handles all databases, from Excel to Access to huge SQL Server databases. XpressDox will always construct a "search" UI which will be presented to users to enable them to choose a row from the table in the database. What this search UI looks like and how the user can filter it depends on the answers to the dialog now visible. The only thing to be provided at this stage is the column on which the filter will operate, and it is suggested that this should be "DirectorName". Select this and press Finish.

- The data source name should be changed now to "Directors" – this is how it will be reference by template commands in XpressDox. Press TAB after typing in the name.
- Now type the following into the "Filter" text box in the configuration UI: `Secretary='<WindowsLogonUser>'` (this is why the "Secretary" column needs to contain the Windows Logon name of the relevant users), and then save the configuration.
- You can test whether your setup is correct (and also get some idea of what the search UI looks like) by pressing the "Test Data Source" button  . After selecting a director, the system shows you what the XML data prepared for that director look like.
- You will then be asked whether you want a "schema" to be generated, and at this time the answer is "yes" – save the generated schema in the same folder as the spreadsheet, for future reference (this schema can be included in the right hand panel of the Template Author's Toolkit which will then able you to easily select the data elements from the data source making sure that they are correctly spelled, etc.)

Now you are ready to let the user capture their director's details:

- On the letterhead template, somewhere near the beginning (although from XpressDox's point of view the placing is irrelevant) put the command <<ChooseFromDataSource(Directors)>>.
- Run a template which is based on the letterhead, and see how the user will experience capturing their Director's information (you will have to have at least one row in the table with a "Secretary" value equal to your logon user name).

13. Looking up values in a table.

The legislated tariff for a particular type of fee that can be charged is worded something like this:

"Where the capital amount is less than or equal to 3000, the fee is 56;
Where the capital amount is greater than 3000 but less than or equal to 5000, the fee is 187;
Where the capital amount is greater than 5000 the fee is 277.

The fee can be calculated as in the following snippet:

```
The fee for capital amount <<CapitalAmount>> is
<<TableLookup(CapitalAmount,'3001;56;0!,5001;187;0!,999999;277;0')>>
```

Note the following:

- The "rows" in the table are separated by commas, which need to be escaped, and the items in each row are separated by a semi-colon.
- The table is enclosed in quotes.
- It would be possible for the table to be sourced from a data source, for example, in which case suppose the data element name in which it is stored is "FeeTable", then the Merge Field would be <<TableLookup(CapitalAmount,FeeTable)>>.

14. Capture a multi-line address in one field

You might not want to have to define 3 or 4 or some maximum number of address line data elements and have the user capture the address one line at a time into the data elements and then provide conditional formatting logic (as in example 3) to suppress the empty data elements.

XpressDox supports this by the concept of "long text fields". The command <<CaptureAsLongText(Address,4)>> will cause XpressDox to provide a 4-line text box in the data capture UI when the template is run. The lines of the address are then typed in by the user one per line (pressing the Enter key at the end of each line).

There is a quirk (or special feature) to this, though, and that is that the <<CaptureAsLongText>> command does exactly and only what it says: i.e. it captures the data element as "long text". What is does NOT do is cause the data element to be inserted into the document. This is because there are two ways in which the data element can be included in the document:

- <<Address>> will cause the lines of the Address to be included in one line. For example, the Address captured as:

  ```
  123 Long Street
  Winelands
  Cape Town
  ```

  would appear as `123 Long Street Winelands Cape Town` in the merged document.

- <<InsertFormattedText(Address)>> will cause the Address to be formatted in the merged document with each line of Address on its own line in the merged document.

The reason for the command being <<Insert**Formatted**Text>> and not something like <<InsertMultiLine>> is that the text captured inside a Long Text text box need not actually contain more than one line, but can be formatted with bold, italic and underline. The help text in the data capture UI describes how to apply that formatting.

15. Define layout in a base template.

Sometimes a letterhead needs to dictate not just things like the company's logo and fairly static items like the directors' names; but sometimes the position of the addressee information and data elements like "Your Reference" and "Our Reference" are required to be in specific positions within the page. These latter items will differ from one letter to another and therefore need to take the form of merge fields on the letterhead template itself.

For sake of argument, suppose it is the addressee information that is required to start in a specific location in the letter head. This can be achieved by putting merge fields into those specific locations, and a snippet of the letterhead would then look like this:

<<Addressee>>
<<AddressLine1>>
<<AddressLine2>>
<<AddressLine3>>
<<PostalCode>>

<<DocumentBody>>

Note that in Example 0 above the statement was made that all templates using the letterhead via a <<BaseTemplate>> command would contain the addressee merge fields, greeting and salutation, etc. In other words the placement of those merge fields on the letterhead would be dictated by the originating template, not the base template.

In this new scenario, the originating templates would (obviously) not have the addressee information on them (otherwise that information would be duplicated in the merged document).

Suppose we now have a small application, and we are going to be running templates and re-using the data sets captured for previous templates. In this application there are a number of templates for letters, some of which need to be addressed to the "buyer" and some to the "seller". Suppose also that there are some documents, such as an agreement, that require the address information of both buyer and seller to appear on them. That means that the data set for the agreement would have data elements like BuyerName, BuyerAddressLine1, BuyerAddressLine2, etc. and similar names for the seller (so that they could be referred to on the agreement by the correct data element names).

The issue is now this: how are we going to use the same letterhead to be a BaseTemplate for letters to the buyer as well as to the seller, when the merge fields on the letterhead are <<Addressee>>, <<AddressLine1>>, etc. when sometimes we would wish them to be <<BuyerName>>, <<BuyerAddressLine1>>, etc. and sometimes <<SellerName>>, <<SellerAddressLine1>>, etc.?

This is done with the <<ReplaceField()>><<ReplaceFieldEnd()>> commands which are coded into the originating template.

A typical template addressed to the buyer would previously have looked something like this:

<<BaseTemplate(Letterhead)>>

<<BuyerName>>
<<BuyerAddressLine1>>
<<BuyerAddressLine2>>

Dear Sir

Various issues have arisen of interest to buyers.

Yours Faithfully

…

Using the new letterhead template, which has the addressee information on it, the new originating document would look like this:

<<BaseTemplate(Letterhead)>>
<<ReplaceField(Addressee)>><<BuyerName>><<ReplaceFieldEnd()>>
<<ReplaceField(AddressLine1)>><<BuyerAddressLine1>><<ReplaceFieldEnd()>>
<<ReplaceField(AddressLine2)>><<BuyerAddressLine2>><<ReplaceFieldEnd()>>
<<ReplaceField(DocumentBody)>>

Dear Sir

Various issues have arisen of interest to buyers.

Yours Faithfully

…

<<ReplaceFieldEnd()>>

The <<ReplaceField(XXX)>> command means this:

"In the base template, find the merge field <<XXX>> and replace that merge field with whatever is between this ReplaceField command and its matching ReplaceFieldEnd()."

Note that the DocumentBody merge field now also needs a <<ReplaceField(DocumentBody)>> and a <<ReplaceFieldEnd()>>. This is the case whenever there are any ReplaceField commands in a template – what was initially

the entire document body now needs to be enclosed in a <<ReplaceField(DocumentBody)>> and <<ReplaceFieldEnd()>> pair.

Note that a <<ReplaceField>> cannot appear after a <<ReplaceFieldEnd()>> if the two are in the same paragraph.

Note also that just about anything (other than <<BaseTemplate>> and another <<ReplaceField()>>) can appear between a <<ReplaceField()>> and <<ReplaceFieldEnd()>>. For example, it might be that we can infer from the seller's address what the postal code should be, then we could do something like:

```
<<ReplaceField(PostalCode)>><<<If(SellerAddressLine2 =
"Pinelands")>>7405<<Else()>><<<If(SellerAddressLine2="Observatory")>>7935<<Else()>><<<PostalC
ode>><<<End(observatory)>><<<End(pinelands)>><<<ReplaceFieldEnd()>>
```

Then, regardless of what the user enters for PostalCode, when the suburb is Pinelands or Observatory, the correct postal code will be merged into the document.

Of course, in real life the second line of the address might not be the suburb, but this example at least illustrates the power of the ReplaceField command.

16. Make data source information available on all templates.

A bit of background: when XpressDox runs a template, it needs two things: firstly, the template itself, and secondly a Data Set – i.e. the data elements which will be merged into the template to form the final merged document.

In the simplest case, the Data Set is captured by the user in the XpressDox data capture UI (the UI is constructed dynamically by XpressDox depending on the merge fields specified in the template).

As we have seen from previous examples, data can be acquired from places other than the user running the template, e.g.

- from a Data Set saved when running another (or the same) template (this Data Set is what is chosen when the user clicks "Use Other Data" in the data capture UI);
- from a text file (using the <<ChooseFromFile>> or <<IncludeFileData>> commands);
- from a database or similar data source (using <<ChooseFromDataSource>> or <<IncludeDataSourceData>>.

The "Include..." commands are executed even before the UI is presented to the user. This means that the Data Set is already populated with data elements from the relevant file or data source before the user gets a chance to enter data. The fields in the UI with the same data element names as the data elements in the data source/file will then be displayed in the UI, for the user to peruse and possibly modify.

The "Choose..." commands are executed when the user presses the browser button on the UI. Once the user has made their choice, the data chosen are copied into the Data Set, and displayed in the UI.

Only those data elements which are referenced in the template are displayed – there can be many more which now form part of the Data Set but are invisible as yet because they are not referenced in the template being run.

Now: it will often be the case that there are data sources whose information should be available to all templates – either all templates in a particular folder, or all templates in a hierarchy of folders (as described in 10 above). (The kind of data required on all templates like this would be data like the company or branch contact details, and other company-related information). Making this data available on all templates could be achieved by the template author remembering to code the command <<IncludeDataSourceData>> on every template. This is open to error

because the Template Author might forget the command on one or more templates. A less forgettery-prone solution is to use the "Use for All Templates" feature in the data source configuration UI. If this option is selected in the configuration for the folder at the top of a folder hierarchy, then the data from that data source will be available to all templates run from any of the subsidiary folders.

A data source flagged "Use for All Templates" and configured in a user's Home folder will also be available on all templates run by that user, subject to the operation of the option "Allow User Home Configuration to override this using the same name" (a checkbox at the top of the data source configuration panel, under the data source name). In other words, if a user runs a template from folder A and there is a data source with name B configured for A or one of its parent folders, then if there is also a data source with name B configured in the user's Home folder, the user's data source will be used only if the data source in folder A has the "Allow User Home Configuration to override ..." option set.

17. Allowing the user to change data source data (and disallowing it).

Suppose you have a data base (it might be a SQL Server data base, or MS Access, or just a spreadsheet) which has customer names and addresses in it. And suppose this database is made available to various templates either by using an <<IncludeDataSourceData>> command, or by the "Use For All Templates" feature described in example 16 above.

Although a database like this is likely to be set up and maintained (i.e. backed up, etc.) by a database administrator, it will usually be the users who run the templates who will be aware of changes to customer data (i.e. changes to addresses and contact personnel). It would then make sense to allow those users to make the necessary changes in the database when they occur.

By default, when data from a data source is used on a template, the user is allowed to change the data in the capture UI, and that changed data appears on the merged document and is also saved in the data set for that template. Also, by default, XpressDox will give the user the option of saving that changed data in the underlying data source. This means that the changed data are available to all other users of the data source immediately.

However, it may be that you, as template author, either

- do not want the data changes made by users to be saved in the data source, or,
- you do in fact want the changes saved, but don't want to have XpressDox prompt the user to confirm this.

These two options are available via the "refresh and save" parameters which are passed to the <<IncludeDataSource>> and <<ChooseFromDataSource>> commands.

<<IncludeDataSourceData(Contacts,RefreshNoSave)>> will mean that user changes are not saved to the data source, as will <<IncludeDataSourceData(Contacts,NoRefresh)>>. On the other hand, <<IncludeDataSourceData(Contacts,RefreshSave)>> will cause XpressDox to save any changes. (The default "refresh and save" parameter is "`RefreshOptionalSave`".

Why the "Refresh" concept?

When a user chooses "Use Other Data" in the data capture UI, then they are given the chance to select the Data Set which was saved for a previous template, and this is then used as the Data Set for the current template. However, this Data Set may have been created and saved days or weeks (or longer) in the past, and the data may no longer be current. However, if some of the data elements come from a data source, that data source is likely to be more current than the saved Data Set from a

previous template, and so the "Refresh" options cause XpressDox to refresh the data from the data source.

"Refresh" and "Save" are linked, because it would be counterproductive to allow users to save their changes into the data source, only to find that when they "Use Other Data" the changes are essentially lost.

However, it might be that in some circumstances the data in the original Data Set should not be refreshed from the data source (this would be the case with example 4 above (Provide default values)). In this case the "`NoRefresh`" options make sense.

Note that "`RefreshOptionalSave`" is the default, and so if you do not want the user changes saved in the data source, and/or you do not want the data refreshed when the user chooses "Use Other Data", then you will have to designate either "`RefreshNoSave`" or (probably) "`NoRefresh`" (which implies no save) in the data source command.

The processing of <<IncludeFromFile>> and <<ChooseFromFile>> with regard to the "Refresh and save" options is almost the same as for the data source commands, but there is no "Save" concept when dealing with data from text files.

18. Moving around.

There was once a tourist travelling around in Ireland. On a certain occasion she wanted to go to Limerick, and so got into her hired car and started driving, knowing that she could always ask for directions. In the end, she stopped at the side of a country road and asked a local inhabitant if he could direct her to Limerick. The elderly gentleman looked slowly up the road, and then down the road, and then said (to be read in a nice thick Irish accent), "If Oi were going to Limerick, I wouldn't be starting here".

It's sometimes like that with the Microsoft file-and-folder explorers, which often enough open far away from where you want to be.

That's why we invented the XpressDox Explorer – which is the interface that is provided when you click "Save Template", "Open Document", and "Run Template".

The Explorer has two "most recently used" ("MRU") lists: one for files and one for folders. The dropdown list at the top of the UI is populated with the folder MRU and the right hand list view is populated with the file MRU when the "Recent Files and Folders" node in the tree view is selected. (Expanding the Recent Files and Folders node in the tree view shows all the recently used folders. In addition, the recently used files appear in the "File name" drop-down list).

When you select "Open Document" and "Run Template" then the Recent Files list is shown when the Explorer opens. When you select "Save Templates" then the list of files (and sub-folders) in the folder at the top of the folder MRU is shown.

This way, once you start working in a particular folder or folders, the MRU lists will tend to have the files from those folders and you will not be required to navigate too far and wide to find the templates or other files that you are working with.

Browsing to a folder

When pressing the "Explore" button you will be given a special version of a Windows *file* opening dialog. This is because the Windows folder opening dialog is the one that most often gives rise to the "if I were going to Limerick ..." situation. Amongst other things it doesn't show you the *contents* of a folder, and often we don't really know which folder we want unless we can see what's in it. So XpressDox gives you this special file opening dialog, which helps you select a folder.

Notice that the default file name in the File Name field is "Choose Folder.ext" – this is a hint to you that it is the *folder* that you are choosing. Even if you select a file in

this dialog (by double-clicking the file name) it is only the folder that is selected (and placed at the top of the folder MRU), but the file name will be put into the XpressDox Explorer's File Name drop down. This can be confusing at times (even to the author) but you will get used to it after using it a few times and the odd moment of confusion is definitely outweighed by the ease of finding the folder that you want.

Favourites

You can create lists of "favourite" folders. These lists appear under the "My Folders" node in the tree view.

Creating a list can be done by:

a. Right-clicking on a file in any view and selecting "Add to My Folders". It is the folder, not the file, that is added to My Folders. You will need to type in a list name, or select one which is already there, and the folder is added to that list with the name you give it.

b. Use the "Add to folder list" toolbar button at the top left of the Explorer UI.

A "Shared Folders" folder list can also be created which contains folders used commonly throughout an organisation with multiple users sharing the same lists of templates. See the Supervisor's Guide for instructions on how to implement this feature.

Handy navigation helps

a. In the "Open Document" mode you might want to view the files in the current folder in the folder MRU: do this by clicking the right-hand-most button next to the folder drop-down ();

b. To view the files in the same folder as a file in the Recently Used files list: right-click the file, and choose "Open Containing Folder".

c. If you need to open the merged document from a template in the Recently Used List (or any other list for that matter): right-click the file and choose "Open Merged Document Folder" and then select the desired merged document.

Template Usage

You will notice a text area at the bottom left of the Explorer. It is either yellow or pinkish, depending on what you're doing. When you save a template, you can type a description, or instructions on usage, into this area. Then, when a user sees that template in the Explorer, and clicks on it, these instructions are displayed in that area.

This is useful for the situation where you might have two or more templates which are similar in function and you need to guide the user as to which one to use.

19. Where have all my data elements gone (Part 1)?

XpressDox has a number of commands which are used to choose data elements, either from lists or data sources or whatever. These are the commands listed in the "Data Capture" node of the Template Author's Toolkit.

These commands are very literal in their behaviour, for example <<ChooseFromList(Province, …)>> will present the user with a drop-down list of options from which to choose the value of the data element named "Province". It does NOT cause the value of the data element to be inserted into the document. There needs to be a merge field such as <<Province>> (or maybe <<ToUpper(Province)>>, or something) which tells XpressDox where (and how) to insert the data element value.

Thus, it might happen every so often that everything works as expected: the user gets a nice list of options to choose from, the template runs smoothly, but the

chosen option never appears in the merged document. This can of course be exactly what you, the template author, desire, but it may be that you expected that XpressDox would insert the result of the user's choice at the point where the <<ChooseXXX>> command appeared in the template. Hence the question which is the heading of this topic. Hopefully the topic has answered the question.

A very simple example of why you might want the user to choose a value and not have that value in the merged document is the situation where you would use the choice in one or more <<If()>> commands to include or exclude paragraphs from the end result. Or, perhaps, you want the option to be captured when one template is run so that it can be used when running a subsequent template.

20. Where have all my data elements gone (Part 2)?

You have authored a template with some good conditional assembly in it, but the XpressDox data capture interface does not present some of the field for capture.

An example might be:

> I appoint <<Fullnames>>, and if <<When(GenderOfAppointee = "F",she,he)>> is unavailable, then <<When(GenderOfAppointee = "F",her,his)>> cousin.

Because the data element "GenderOfAppointee" only appears in the "when" conditional expression, XpressDox needs to be explicitly requested to capture this data item. This can be done via a Choose... command (e.g. <<ChooseFromRDBList(GenderOfAppointee,F,M)>>), or by using the command <<CaptureDataElement(GenderOfAppointee)>>.

21. Format an ID number.

You want to ensure that an ID number is captured as numeric digits, but that it is rendered in the merged document with formatting or spacing characters.

For example, the South African ID number consists of 4 parts: The first 6 digits are the date of birth of the person, the next four are a tie-breaker (so that people born on the same day get a unique number), and the next two have other meanings (in the old South Africa they referred to a person's racial classification). The last digit is a check digit helping to detect transcription errors.

Traditionally, the SA ID number has been written xxxxxx-xxxx-xxx, although writing it with no formatting delimiters is also legally acceptable (if not as easy on the human eye).

XpressDox's FormatNumber function will do two things:

1. When the data element is captured, a check is done to make sure it is numeric and the user warned if this is not the case;
2. It will format the number according to the formatting pattern provided.

Thus, the merge field to capture a South African ID number as numeric digits and then render it according to the traditional format is:

<<FormatNumber(IDNumber,"000000-0000-000")>>

22. Include a template whose file name is inferred from the data at run time.

Suppose you are running a system (i.e. XpressDox templates) for a small business and, instead of keeping your clients' addresses and addressee information in a database (or spreadsheet, which is the same thing, from XpressDox's point of view), you have decided to keep the address information as a set of templates, each with one client's address, laid out and formatted as you would like it to appear on a letter.

You have configured a helper folder whose alias is "`Clients`", and in that folder are templates with the names "Addr001.xdtpl", "Addr002.xdtpl", etc. Where the numeric parts of the template names are the account numbers of the clients.

The idea would be that each template requiring these client addresses would at least have a merge field to capture the account number. A snippet might look like this:

```
<<BaseTemplate(Letterhead)>>
<<Required(AccountNumber)>>

Dear Sir
```

At this point we would like to put something under the <<BaseTemplate()>> which would include the template with the client's address in it, something like

```
<<IncludeTemplate(clients:Address<AccountNumber>)>>
```

But, the IncludeTemplate command is actually executed *before the data capture form is presented to the user*, so it isn't possible to know the value of the AccountNumber data element at the time the IncludeTemplate executes.

That is why XpressDox has the InsertDocument command.

The difference between IncludeTemplate and InsertDocument is that IncludeTemplate takes a constant ("hard-coded") value as the name of the template (which is known to the template author at the time the template is coded), whereas InsertDocument is given the name of the template either in a data element or an XpressDox variable.

In the case of this example, if the address templates had been stored in the same folder as the template being run, and if their file names had been the account numbers, then we could have accessed them at run time like this:

```
<<BaseTemplate(Letterhead)>>
<<Required(AccountNumber)>>
<<InsertDocument(AccountNumber)>>

Dear Sir
```

This would work because the AccountNumber data element would actually contain the address template name, and as it has no helper folder alias in it, the document will be inserted from the template folder (i.e. the folder from which the originating template is run).

However, in this example, the address templates are in a helper folder (alias "Clients"), and the names of the address templates contain the AccountNumber but are not the AccountNumber.

This means we have to somehow construct the address template reference out of the helper alias, the AccountNumber data element value and the string "Addr". This is done using the "concat" function and constructing the result into an XpressDox variable. The resulting snippet would look like this:

```
<<BaseTemplate(Letterhead)>>
<<Required(AccountNumber)>>
<<SetV('AddressTemplate',concat('clients:Addr',AccountNumber))>><<InsertDocument(GetV(
'AddressTemplate'))>>

Dear Sir
```

Please note the use of quotes: the "hard wired" values are in quotes, whereas the data element name is NOT in quotes. This instructs the system to find the *value of the data element*, in this instance the value that the user entered for the AccountNumber data element in the capture screen.

If the "AddressTemplate" variable name had not been in quotes, the system would try to look up the value of a data element called "AddressTemplate", and use that as

the XpressDox variable name. It is theoretically possible for this to work, but in this instance it is not what is wanted.

23. Run a template for a number of data sets, and print the merged documents.

Remember that a "data set" in XpressDox terms is the set of data elements used in a template. The data values come either from being captured by the user, or from data source, or both. This data set is saved whenever a template is run, in a location and with a name depending on the configuration for the template folder.

By default the data set is saved in the same folder as the template, and with the same name as the saved merged document but with the extension "`xddata.xml`".

This example will assume you have set up an application in a manner similar to that described in example 10 above. In other words, the data sets all contain a data element with a client's account number, and, in particular, the data sets are saved in files whose name is the account number.

Suppose, then, you have decided to do something like send a newsletter to each client, and want to personalise it with information from the data set for each client.

For this you would use the XpressDox "Batch Run Templates" feature (the  button). This feature:

- presents you with a list of templates to choose - the one you want will either be in the Recent Files list, or else you use the Explorer to navigate to where the template is. Select the template.
- presents you with a list of data sets to choose – choose the ones you want.

The system then runs the template against each data set, and leaves the merged document open in Word.

Once all the documents have been merged, you can print them all using the Print All Documents button in the XpressDox Utilities toolbar, and then close them all with the Close All Documents.

Notice that in the first bulleted step above you could actually choose any number of templates to be run. If more than one template is chosen, the XpressDox will run each template against each of the chosen data sets.

An important point to be aware of is that if a template contains <<IncludeDataSourceData()>> or <<IncludeFileData()>> commands, then the XpressDox Batch Run feature will assume that ALL the data elements for the template(s) will be supplied by those data sources or data files. You will NOT then be presented with a list of data sets to choose. Have a look at example 2424 below.

24. Run a "mail merge".

You want to print a document for a number of clients, where all the client contact information as well as pertinent information for the document content is contained in a database.

The database is defined in a data source called "Clients". You want to send the document to all clients where the column "Mailshot" in the data base contains the value "yes".

A snippet of a template to do this would look like:

```
<<BaseTemplate(Letterhead)>>
<<IncludeDataSourceData(Clients,,range=Mailshot = 'yes')>>
<<ForEach(Client)>>
<<If(position() > 1)>>.....Page Break.....<<End()>>


Dear <<Salutation>>
```

Body of letter

<<End(foreach client)>>

The <<If(position() > 1)>> will insert a page break at the beginning of each client except for the first, so that the letter to each client appears on its own page. For readability, the "....Page Break...." above is not a real page break. In reality you would insert a Word page break, and make sure there are no paragraph ends between the <<If(position() > 1)>> and its <<End()>>. Have a look at the template "`Invitation(Like a Mailmerge)`" in the `My Documents\XpressDox\Samples` folder.

25. Why do my calculations not work?

In general, XpressDox handles calculations in an obvious way. For example, the merge field <<FormatNumber(Amount * 1.1)>> will show the Amount data element plus 10%. Or <<Amount1 + Amount2>> will show the sum of those two amounts.

But <<FormatNumber(Amount1+AnotherAmount-Amount2)>> will in all likelihood show "N" is not a valid digit. The problem is that the minus sign (or, in XML a "Dash") is a valid symbol in an element name, and so AnotherAmount-Amount2 is actually a data element, and will (probably) have no value, and so the XML processor will not know how to add it and return "NaN" (Not a Number) for its value.

A good rule, then is to put a space around ALL the arithmetic operators (i.e. around + - * div).

Note also that "/" is NOT the symbol for division: "div" is the symbol for division.

26. Conditional Inclusion of Paragraphs and Clauses.

Numerous instances of the use of <<If()>> commands appear in the preceding examples. There is no great difference between using an If command to include or exclude a few words or letters or to include or exclude a paragraph or group of paragraphs.

Instances where this would be applicable would be, for example, to include a party's date of birth when the party is an individual, or to include a longish paragraph describing a company when this is applicable (and exclude the date of birth).

A template snippet might look like this:

<<ChooseFromRDBList(PartyType,Individual,Company,Trust)>>
<<If(PartyType = 'Individual')>>
Date of birth: <<FormatDate(DateOfBirth, "yyyy-MM-dd")>>
<<Else()>>
Registration Number: <<RegistrationNumber>>
<<End()>>
<<If(PartyType = 'Company')>>
Carrying on business as <<TypeOfBusiness>> and registered according to the laws of Honduras.
<<End()>>

In most cases, the <<If()>> command used like this will be sufficient to include and exclude the relevant paragraphs depending on the values of data elements entered by the user.

Sometimes, though, this might not be sufficient, because there are far too many different options from which to choose. For example, if you were setting up a template for a title deed and wanted to have the user choose the title deed conditions, the problem becomes a huge one. This is because just about every property will have its own more or less unique set of title conditions, and to attempt to put "if" logic into a template to handle this situation would be impossible.

For a situation like this, the different clauses would be constructed and saved in an appropriate folder, or set of folders. The user would be required, when they run the template, to choose the clause, or clauses, which the document needs. This is achieved with the <<InsertDocument()>> command.

When XpressDox sees the InsertDocument command in the template it presents the user with a data capture control which, when selected, brings up the XpressDox Explorer. The Explorer is opened at the folder which the template author has configured in the template configuration as the "Library Folder". (It would make sense for the Library folder to be the folder where the clauses have been stored). As long as the files containing the clauses are reasonably named, the user will be able to choose the relevant clauses for the task at hand. (This is where the Explorer's "Template Usage" feature would be very useful in guiding a first-time user in their choice of clause).

A snippet of a template using an InsertDocument command might look like this:

> The property is to be transferred subject to the following conditions:
>
> 1. <<ForEach(ConditionClause)>>
> 2. <<InsertDocument(Clause)>>
> 3. <<End()>>

Notice that since the <<ForEach>> and <<End()>> are alone in their own paragraphs, those two paragraphs will not appear in the merged document. This will leave only the clauses selected by the user, suitably numbered.

When inserting documents into other documents in this way the styles of the source and destination documents play an important role. The default formatting option for InsertDocument (and for IncludeTemplate) is "Destination". This means that the documents will format as expected as long as the styles in the inserted document match (by name and formatting properties) the equivalent style in the destination document. Depending on how the template and clauses have been constructed, it may be that Source formatting would be preferable. But that takes us into Word styles, which is a topic too vast to be dealt with in this Cookbook.

27. When to use quotes and when not.

XpressDox tries to use quote marks as sparingly as possible. Rather, it insists on quote marks an as few places as possible. Template authors may find this disconcerting to begin with, particularly if they have been using other document assembly products where quote marks are used in many places where XpressDox does not require them.

The following is a re-print from the XpressDox User Reference:

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.

28. Format a date in Archaic form.

For fun, or perhaps a real need, you want to format a date in the old style, with dipthongs after the day number, etc., rendering "2009-06-23" as "the 23rd day of June in the year of Our Lord Two Thousand And Nine".

Here's how you would do it, supposing that the data element DateOfSignature hold the date to be formatted.

<<SetV('day',FormatDate(DateOfSignature,"dd")) this sets the variable "day" to the day number of the DateOfSignature data element>><<FormatNumber(GetV('day'),"#") the value of the day number is inserted into the document, with leading zeros suppressed>><<GetListItem(GetV('day'),st,nd,rd,th,th,th,th,th,th,th,th,th,th,th,th,th,th,th,th,th,th,st,nd,rd,th,th,th,th,th,th,th,st) the day number is used as an index into the list of dipthongs and picks out the correct one>> day of <<FormatDate(DateOfSignature,"MMMM") the month part of DateOfSignature is inserted into the document, spelled out fully>> <<SetV('SignYear',FormatDate(DateOfSignature,"yyyy")) the variable "SignYear" has its value set to the year part of the DateOfSignature>>in the year of Our Lord <<NumberPhrase(GetV('SignYear')) the numeric value of the variable "SignYear" is rendered in words>>.

After setting the command out as above, you can change the font on the <<GetListItem(...)>> command to be superscript, so the "23rd" will be rendered "23rd".

29. Handle gender.

Suppose you have a document which you want to personalise as far as pronouns are concerned. In other words, instead of having "he/she" in many places, you want to have only "he" or "she" depending on the sex of the party concerned. This could be done by determining the sex up front at the beginning of the template like this:

<<ChooseUsingCheckBox(PartyIsMale)>>

And then in the body of the template, whenever the document would previously have had "he/she", put:

<<When(PartyIsMale = "true",he,she)>>

While this is nice, it is bit tedious to type the <<When(PartyIsMale = "true",he,she)>> wherever it is required. If there are many places where "he/she" is to be replaced, then one way to create a shortcut is to use a variable, like this:

<< ChooseUsingCheckBox (PartyIsMale)>>
<<If(PartyIsMale = "true")>><<SetV("he","he")>><<Else()>><<SetV("he","she")>><<End()>>

Then, in all the places where "he/she" is to be replaced, instead of using the "When" above, just type:

<<GetV("he")>>

Even nicer is that you can handle "him/her" and "his/her" in the same way:

<< ChooseUsingCheckBox (PartyIsMale)>>
<<If(PartyIsMale = "true")>><<SetV("he","he")>><<Setv("him","him")>><<SetV("his","his")>><<Else()>><<SetV("he","she")>><<SetV("him","her")>><<SetV("his","her")>><<End()>>

Then the following becomes possible:

If <<GetV("he")>> makes a written request, then <<Getv("his")>> copy will be sent directly to <<Getv("him")>>.

Similar shortcuts can be made for handling plurals.