A Comparison of Contract Express and DocXpress

Introduction

This document compares the engines underneath the hoods of Contract Express and DocXpress.

Both engines resolve the same complex problem:

> *How to establish the relevance of an instance[1] of a variable in a marked-up template document?*

but using radically different approaches.

Whereas Contract Express analyses the template document (at compile-time) to establish a usage expression for each variable, and evaluates (at run-time) that usage expression to establish its relevance, DocXpress evaluates the whole template document (at run-time) establishing relevant variables as and when they are encountered.

On the face of it Contract Express would appear to be the more efficient approach because it does much of its work at compile-time, whereas DocXpress does all of its work at run-time. Read on!

[1] An instance of a variable is a particular repetition of that variable.

Example

The markups employed by Contract Express and DocXpress are very different, but both accomplish the same three objectives:

- How to define some text that will appear in the assembled document based upon the evaluation of an expression (a field)
- How to include/exclude a portion of the template document based upon the evaluation of a boolean expression (a conditional span)
- How to repeat a portion of the template document based upon the evaluation of a numeric or list expression (a repeated span)

Contract Express uses character-based markup:

```
[A1 and A2{F1}[Repeat M{F2}[B1 or B2{F3}[Repeat N{F4}]]]]
```

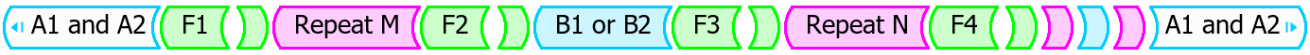
where:

`{F1}`, `{F2}`, `{F3}` and `{F4}` are fields

`[A1 and A2 ... ]` and `[B1 or B2 ... ]` are conditional spans

`[Repeat M ... ]` and `[Repeat N ... ]` are repeated spans

DocXpress uses Microsoft Word® content controls as markup:



where:

green content controls are fields

blue content controls are conditional spans

pink content controls are repeated spans.

How Contract Express Works

Contract Express analyses the template document at compile-time to establish the usage expressions for each individual variable.

A usage expression consists of one or more usage contexts because variables may appear multiple times throughout the template document, sometimes in different contexts.

A single usage context consists of its repetition context and its logical context.

A repetition context consists of zero or more list expressions where each member of the evaluated list represents an individual repetition.

A logical context is a boolean expression that is evaluated in the repetition context.

For the example above Contract Express establishes at compile-time the following usage contexts:

Variable	Repetition Context	Logical Context
A1		A2
A2		A1
F1		A1 and A2
M		A1 and A2
F2	1 to M	UnRepeated(A1) and UnRepeated(A2)
B1	1 to M	UnRepeated(A1) and UnRepeated(A2) and not B2
B2	1 to M	UnRepeated(A1) and UnRepeated(A2) and not B1
F3	1 to M	UnRepeated(A1) and UnRepeated(A2) and (B1 or B2)
N	1 to M	UnRepeated(A1) and UnRepeated(A2) and (B1 or B2)
F4	1 to M 1 to N (*for each 1 to M*)	UnRepeated(UnRepeated(A1)) and UnRepeated(UnRepeated(A2)) and (UnRepeated(B1) or UnRepeated(B2))

Conjunction

The logical contexts derived from a conditional span that involves a conjunction such as `A1 and A2` states that variable `A1` is only relevant if variable `A2` does not evaluate to false, and that variable `A2` is only relevant if variable `A1` does not evaluate to false.

Disjunction

The logical contexts derived from a conditional span that involves a disjunction such as `B1 or B2` states that variable `B1` is only relevant if variable `B2` does not evaluate to true, and that variable `B2` is only relevant if variable `B1` does not evaluate to true.

Repetition

The logical contexts for variables within a repeated span that reference variables outside of that span use the construct UnRepeated(...)[2].

Pros and Cons

The main (and only) pro for this approach is that the work involved in establishing these usage expressions is all done at compile-time.

There are two very serious cons to this approach:

- Variables that appear next to each other in the template document have separate, but identical, usage expressions.

  Consequently, the same usage expression will be evaluated multiple times with the same result each time[3].

- Usage expressions grow exponentially within nested conditional and repeated spans, especially those that involve conjunctions or disjunctions.

  There are real examples of individual usage contexts that have occupied more than 20 pages when pasted into a Microsoft Word® document.

[2] Evaluate the expression in the parent repetition context.

[3] Reminds me of the definition of madness – repeating the same behavior and expecting a different outcome.

How DocXpress Works

DocXpress has a much simpler model for establishing the relevance of individual variables.

Evaluate the whole template document at run-time by evaluating the expressions that occur within the marked-up repeated spans, conditional spans, and fields. Whenever the evaluation of an expression requires the value of an instance of a variable, that variable instance must be relevant.

This model would be extremely onerous if the whole template document were to be fully evaluated each and every time the value of any variable was changed, and so a few optimization techniques are employed.

Firstly, the template document is analysed (effectively compiled) once at the start of run-time to determine which variables are referenced in repeated spans, which variables are referenced in conditional spans, and which variables are referenced in fields. The links from variables back to those spans and fields are established.

Secondly, there are two distinct phases when evaluating the whole template document. The **render** phase evaluates the repeated spans, and the **instantiation** phase uses the rendered document to evaluate the conditional spans and fields.

When a variable is changed that is referenced in a repeated span, the whole document is **re-rendered** and **re-instantiated**. Such variables, however, are always a tiny minority.

When a variable is changed that is referenced in a conditional span (but not referenced in a repeated span), each of those rendered spans is re-evaluated and either introduced into the assembled document, removed from the assembled document, or left unchanged. Such variables are common but still in the minority.

When a variable is changed that is only referenced in fields, it is just those specific fields that are re-evaluated. This is often the vast majority of variables.

Performance Comparison

The performance comparison reported here used a variation of the accompanying template document **Viaweb NDA.docx**. The main characteristics of this template are typical of real-world documents, with a mixture of repeated spans (the number of other parties, the different categories of information that is disclosed), conditional spans (one-way disclosure, mutual disclosure, etc.), and fields (the agreement date, the number of other parties, the information disclosed, etc.).

Contract Express performed well up to 10 other parties, reasonably well up to 30 other parties, and badly or not at all beyond that figure.

DocXpress performed exceedingly well up to 400 other parties, reasonably well up to 4,000 other parties, and badly or not at all beyond that figure.

In addition, because DocXpress is a client-only web application there are no server requests to update the questionnaire, whereas Contract Express issues a server request for each and every navigation between questionnaire pages.

It was mentioned in the introduction that, on the face of it, Contract Express would appear to be the more efficient approach because it does much of its work at compile-time. The actual performance of DocXpress would suggest otherwise.