最優秀賞

Excelテンプレートを使用した帳票出力機能の開発
―セルフサービス化への道

駒田 純也 様
ユサコ株式会社
アカデミア事業部
技術部 開発課

ユサコ株式会社
http://www.usaco.co.jp/

海外の学術雑誌、書籍の輸入販売を中心に事業を展開している。とくに医学、薬学等の自然科学分野に強みをもつ。近年、学術情報媒体のIT化が進み、データベースや各種ソフトウェアの取り扱いを強化。学術情報を通じ、知的情報の創造、蓄積、共有による社会貢献を目指している。

Excelテンプレートを使用した帳票出力機能（以下、Excelテンプレート帳票）は、Excelで自由に帳票テンプレートを作成し、そのテンプレートファイルをDelphi/400で読み込み、Excelファイルとして帳票出力する機能である。【図1】

開発の経緯

Delphi/400以外で開発された社内システムの開発環境やプログラムの維持管理が負担になってきたため、今回Delphi/400に移行することになり、そのシステム内で独自に開発したExcelスタイルシートを使った帳票出力機能も何らかの形で移行する必要が出てきた。

基幹システムと同一のプラットフォームになることで、マスタデータを同期する必要がなくなり、ユーザーもログインするシステムが減り、デザインや操作感が統一されたインターフェースによる作業効率アップのメリットはある。しかしExcelスタイルシートを使った帳票出力は複雑なので、同様の仕組みにすると、ユーザーが自身でテンプレートを作成・編集するのは難しい。

そこで考えたのが、テンプレートとなるExcelのセルに出力キーワードを埋め込む方法である。この方法であれば、既存の見積書などのExcelファイルをそのまま使用してテンプレートを作れるので、既存資産が無駄にならない。使い慣れたExcelで帳票をデザインできるため、新しいことを覚える手間もほとんどかからない。

また1つのファイルだけで済むので、情報システム部門としても管理が容易である。

セルフサービス化の必要性

個人的な感覚であるが、とくにここ数年はWebシステムが大きく発展しており、当社でもセルフサービス型の製品をいくつか導入するに至っている。Delphi/400で自社開発したシステムでも、「ユーザーが自分自身でカスタマイズできる」機能を開発フレームワークに追加実装し、セルフサービス化を意識して改善に取り組んでいる。

ユーザーが思ったときに、思ったことができれば時間のロスを防げる。管理すべきシステムが増えれば増えるほど、それぞれに要する時間は減るが、システム間連携の設定・構築に時間を取られるので、情報システム部門としてもセルフサービス化の推進は必須だと考えている。

テンプレートをユーザーが自身で編集してアップデート

テンプレートを共有フォルダ上に配置することで直接編集したり、いったんローカルフォルダにコピーしてから編集しアップデートすることも可能となる。

出力キーワードは「$ {」で始まり、「}」で終わる形式で、その中にテーブル名（TClientDataSet名）とフィールド名の順に「/（スラッシュ）」で区切り、記述する。「Header」テーブルのフィール

図1　帳票出力結果

請求書

請求No:123456
請求日:2018/8/24

駒田 様

合計金額 ￥ 28,380 (税込)

行No	タイトル	単価	数量	金額
1	Delphiテクニック集	3,800	1	3,800
2	オブジェクト指向開発	4,980	1	4,980
3	帳票デザイナー	9,800	2	19,600

図2　明細を含む単票形式の帳票テンプレート

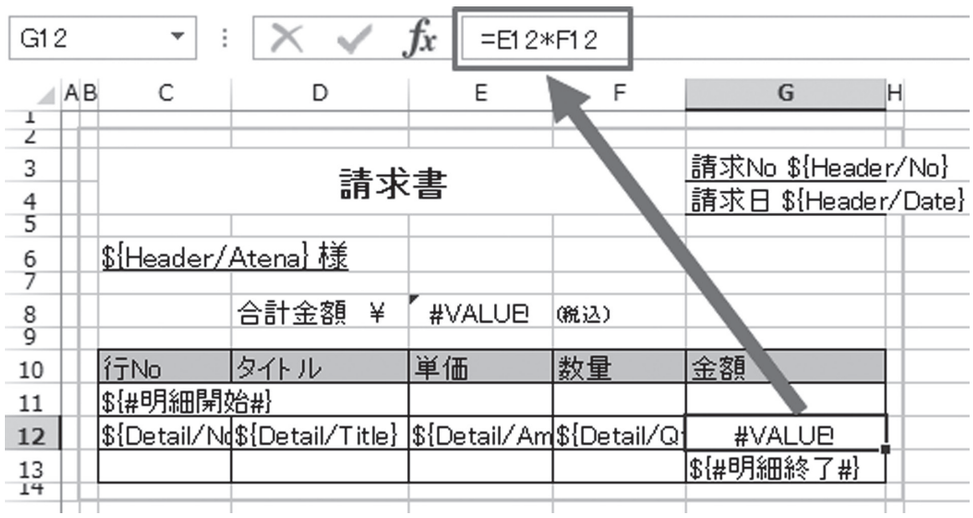
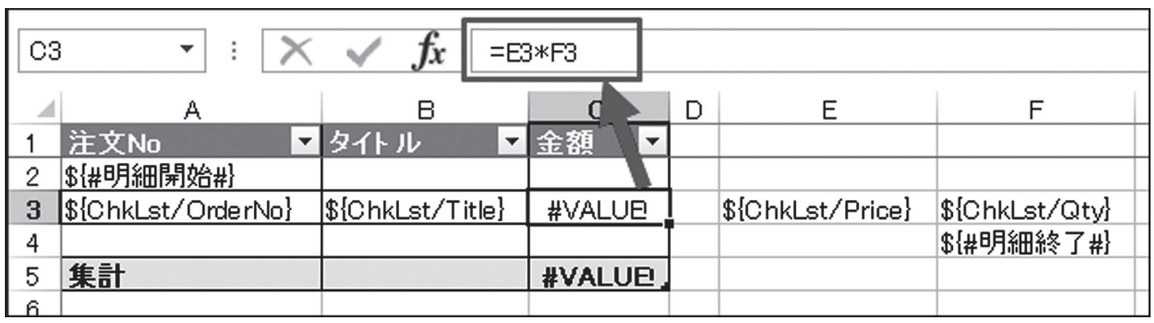


図3　リスト形式の帳票テンプレート



ド「Atena」を出力したい場合は、「${Header/Atena}」となる。「様」や「請求No」などの文字列も、同じセルに自由に記述してよい。

明細部分は、キーワード「${#明細開始#}」と「${#明細終了#}」で囲む形で定義する。明細キーワード行は出力時には削除されるので、実際に出力されることはない。

図3は価格と数量の計算結果のみを金額列に出力するようにしており、印刷範囲として設定した範囲外に価格と数量を出力し、Excelの計算式を使って金額を計算している。

このようにExcelの機能を利用すれば、工夫次第でいろいろな帳票を作成できる。【図2】【図3】

なおフィールド名はアルファベット表記であり、現場ユーザーにはわかりづらいため、当社では該当機能で利用可能なフィールドを検索できる汎用画面を用意している。実装も簡単で、セルフサービス化のハードルを下げる役割もあると考えている。【図4】

帳票出力処理の流れ

おおまかな流れは、【図5】のとおりである。説明の都合上、単票と呼ばれる【図1】のような帳票での明細部分以外を「単票部分」と書いている。

機能全体を1つのデータモジュールとして作成しており、出力キーワードを格納するTClientDataSet（cdsKeywd）を配置。デザイン上であらかじめ必要なフィールドとして、セル位置とその内容、テーブル名とフィールド名、明細部分の開始・終了・明細内部であることを示すフラグを格納するためのフィールドが存在している。【図6】

帳票出力部分のコーディングでは、データモジュール内の以下の引数をもつ関数に適切な引数を渡して呼び出すだけでよい。【図7】

* 引数1［i］：string型＝テンプレートFileのフルパス
* 引数2［i］：string型＝保存先Fileのフルパス
* 引数3[i]：TStringList型＝テンプレート内で指定されたテーブル名（TClientDataSet名）のリスト
* 引数4［i］：TForm型＝データ用TClientDataSetが定義されている対象フォーム
* 引数5［i］：Boolean型＝Excelを画面表示するかどうか

上記のうち、引数1と引数2はそれぞれのファイルへのフルパスで、引数1が存在しない場合はエラーとなる。引数2の保存については強制的に上書き保存したり、出力ファイル名に日付や実行ユーザー名を加えたりと、関数側で動作を設定している。

引数3は、帳票テンプレートに登場するテーブル名（TClientDataSet名）をすべて列挙したTStringList型の変数を渡す。

引数4は、引数3で渡されたTClientDataSet名をFindComponentで取得するため、各データセットが定義されている対象フォームを渡す。

引数5は処理終了時に、それまでバックグラウンドで処理していたExcelを画面表示するかどうかをBoolean型で渡す。Falseの場合、つまり画面表示しない場合のExcelプロセス終了はソースコード上で記述する必要があり、Trueの場合はExcel画面が表示されるためユーザーに委ねられる。

関数内部の処理の流れは、テンプレートファイルを開く、Cloneカーソルを作成する、各シートごとに処理する、といった順になっており、シートごとにキーワード解析、データ差し替え処理を実行している。【図8】【図9】

ブック全体のキーワード解析を最初に実行してから、各シートを処理する方法も考えられるが、速度的にそれほど差はなさそうなので、実装方法はコーディングの好みになるかもしれない。なお呼び出し元の画面に影響を与えないために、Cloneカーソルを作成している。

キーワード解析は、まず検索を行い、セル位置や内容をcdsKeywdへ登録するという手順で実行している。ポイントは、UsedRangeプロパティで使用されているセル範囲を取得してから行うこと。それにより範囲をあらかじめ絞れるので、余計なオーバーヘッドを減らせる。【図10】

データ差し替え処理は、単票部分と明細部分を別々にループ処理している。1つのループでも処理可能だが、ソースコードの可読性を優先した。同じcdsKeywd内に単票部分と明細部分が混在するため、Filterプロパティを使い絞り込んでから、それぞれのループを処理している。【図11】

明細部分のデータ差し替え処理に関する注意点には、Variant型の2次元配列を使用し、明細範囲を一括で書き込むことが挙げられる。もしセルを移動しながら明細を1セルずつ書き込んでいった場合、セル移動のオーバーヘッドが大きいため、速度が劇的に低下し、実用に耐えない。

配列をVariant型で定義することにも理由がある。たとえば文字列型で定義すると、セルの書式などが自動的に処理されないためである。【図12】

実際に出力して感じたこと

考えていたよりも使い勝手はよさそうな印象で、もともと利用対象として考えていたDelphi/400へ移行するシステムだけでなく、応用範囲は広がりそうである。とにかく帳票設計が簡単なので、ユーザーのアイデアを引き出すツールになるかもしれない。

当社はFastReportも導入しているが、このExcelテンプレート帳票があれば、必要ないかもしれない。

帳票サーバーも導入しており、バッチ処理で出力される帳票には威力を発揮しているが、入力チェックリストの類はスプールから出力指示を出さなくても、直後にデスクトップ画面にExcelとして表示したほうがデータとしても処理できるので、いろいろと手間を省けるように思う。

また出力関数を呼び出した際に、シートごとにキーワード解析とデータ出力を単票部分と明細部分に分けて実行する仕様なので、同様のループを複数回実行することでオーバーヘッドが速度的に問題にならないか心配であった。しかし今のところ、それほど気にならない状況である。

今後の課題・計画

課題としては、Excelへの依存が挙げ

図4　フィールド検索画面

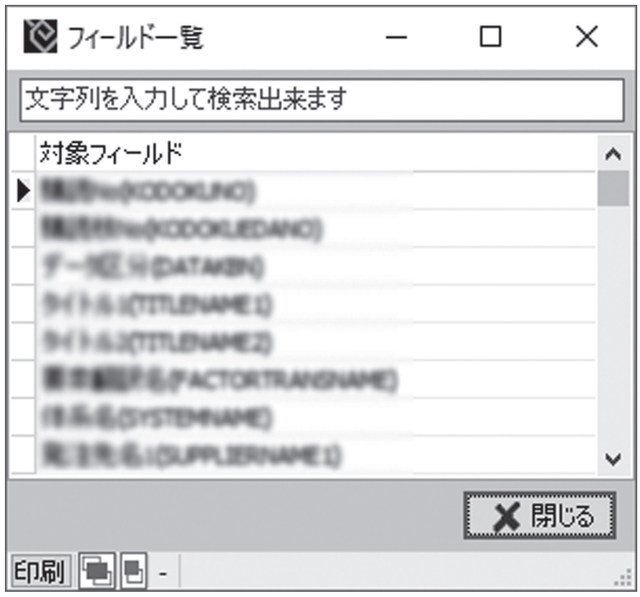
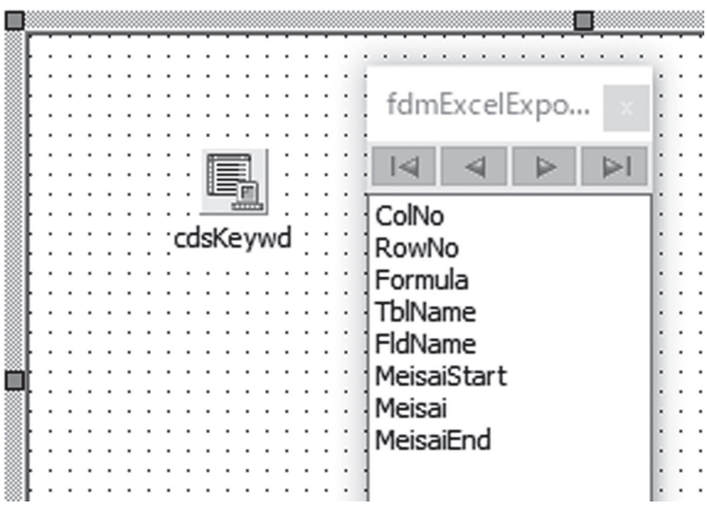


図5　おおまかなプログラムの流れ

```
◇テンプレート読込、シート数の確認
◇Clone カーソル作成
◆シート毎の処理(Loop)◆　ここから
  ◇使用されているセル範囲を取得
  ◇(最初の)キーワードを検索
  ◆明細の存在確認と明細範囲の保管(Loop)◆
  ◇(最初の)キーワードを再検索
  ◇キーワード先頭 Cell 保管
  ◆単票部分：キーワードを抽出し、cdsKeywd へ格納する(Loop)◆
  ◆明細部分：全 Cell の内容を、cdsKeywd へ格納する(Loop)◆
  ◆単票部分：データ差し替え(Loop)◆
  ◆明細部分：データ差し替え(Loop)◆
◆シート毎の処理(Loop)◆　ここまで
◇Excel ファイルを別名で保存 (テンプレートを上書きしない)
◇Excel 画面を表示 or バックグランドのままプロセスを終了させる
```

図6　出力キーワードを格納するcdsKeywd



られる。VBA の仕様が急に大きく変わるとは考えにくいが、コントロールできない部分であるため Office のバージョンアップなど大きな変化には注意が必要である。

今後の計画としては、ユーザー自身で各画面に対して帳票出力を設定できるような仕組みを考えている。すでに TClientDataSet のフィルタ内容をユーザー自身で編集し、10 個まで保存できる機能を作っているが【図 13】、同様に Excel テンプレートのファイルパスをいくつか登録しておき、選択した帳票を出力すれば短期間で作成できるはずである。

実はこの Excel テンプレート帳票は、当社で導入しているセルフ型の Web アプリ開発ツールにインスピレーションを受け開発したものであるが、そのツールにはテンプレート側とデータベース側フィールドのマッチングを設定する機能がある。テンプレート側のキーワード部分を自由に命名できるので、ユーザーにとって見やすいテンプレート作成が可能になっている。

現状ではフィールド物理名をキーワードにしているため、少々見づらい。そこで、そういったマッチング機能も検討している。あとは、マスタなどのコードからその名称に変換する処理を実行できるよう、明細部分のレコードごとにイベントが発生するような仕組みも検討している。

最後に

当社では本稿で紹介した以外にも、セルフ化を意識した機能の実装や、Delphi/400 以外でもセルフイノベーションを支援する製品を導入している。情報システム部門の人数は限られているので、現場の意欲を高め、その意欲を打ち消さないために、そういった環境の重要性と影響力の高さもひしひしと感じている。

今回のようなセルフ化や自動化によって、現場と情報システム部門の双方で削減できた時間を創造的な業務を行う時間へとシフトさせていく。そういったスパイラルへとつなげていきたい。

M

図7　呼び出し部分のサンプルコード

```
TemplateFilePath:='¥¥サーバ¥ExcelTemplate¥チェックリスト.xlsx'; //テンプレート
SaveFilePath:='¥¥サーバ¥リスト類¥チェックリスト_20180824_'+UserID+'.xlsx'; //保存先
CDSList:=TStringList.Create;
try
  CDSList.Add('cdsHeader');
  CDSList.Add('cdsDetail');
  isVisible:=False; //Excelをバックグラウンド処理
  sRet:=DataModule.ExcelOutput(TemplateFilePath,SaveFilePath,CDSList,Self,isVisible);
  if sRet <> '' then ShowMessage(sRet) else ShowMessage('Excel出力完了。');
finally
  CDSList.Free;
end;
```

図8　Excel OLE利用とテンプレート読み込みのサンプルコード

```
vExcel, vWkBook, vWkSheet: OleVariant;
hExcelhWnd: THandle;
~~~~~~~~~~~~~~~
vExcel:=CreateOleObject('Excel.Application'); //Excel OLE 準備
hExcelhwnd:=vExcel.hwnd; //Excel プロセス終了処理で利用する
vWkBook:=vExcel.WorkBooks.Open(TemplateFilePath); //テンプレート読込
```

図9　Cloneカーソル作成のサンプルコード

```
//Clone を作成し配列に格納する
SetLength(arCdsCln, CDSList.Count);
for i:=0 to (CDSList.Count - 1) do begin
  arCdsCln[i]:=TClientDataSet.Create(Self);
  arCdsCln[i].CloneCursor(TClientDataSet(Frm.FindComponent(CDSList[i])), False);
  arCdsCln[i].DisableControls;
  arCdsCln[i].Name:=CDSList[i];
end;
```

図10　キーワード検索のサンプルコード

```
vWkSheet, vRange, vCellTmp: OleVariant;
~~~~~~~~~~~~~~~
//データが格納されているセル範囲を取得(空白 Cell でも何かしていれば含まれる場合がある)
vRange:=vWkSheet.UsedRange;
//Range 内でキーワードセルを検索 ※実際のコーディングでは最初に検索した位置までループしている
vCellTmp:=vRange.Find['${*}'];
//キーワード情報を登録(単票部分)
cdsKeywd.AppendRecord([列, 行, セル値, テーブル名, フィールド名, 0, 0, 0]);
```

図11　単票部分のデータ差し替え処理サンプル

```
Field: TField;
vCellTmp: OleVariant;
～～～～～～～～～～～～
//cdsKeywd のループ処理
//対象セル値のキーワード部分をデータベースのフィールド値に差し替える
Field:=fFindClnCDS('cds' + cdsKeywd TblName.Value).FindField(cdsKeywdFldName.Value);
vCellTmp:=vWkSheet.Cells[cdsKeywdRow No.Value, cdsKeywdCol No.Value];
vCellTmp.FormulaR1C1:=StringReplace(cdsKeywdFormula.Value,
  '${' + cdsKeywdTblName.Value + '/' + cdsKeywdFldName.Value + '}', Field.DisplayText, []);

※「fFindClnCDS」は arCdsCln 配列から指定した Name を持つ Clone を TClientDataSet 型で返す関数
```

図12　明細部分の一括データ貼り付け処理サンプル

```
vWkSheet: OleVariant;
arMeisai: array of array of Variant; //明細部分のデータ格納配列
～～～～～～～～～～～～
//明細範囲に配列値を一括で貼り付ける
vWkSheet.Range[vWkSheet.Cells[開始行,列],vWkSheet.Cells[終了行, 列]].FormulaR1C1:=
  Variant(arMeisai);
```

図13　フィルタ内容保存機能