MIGARO. TECHNICAL REPORT

ミガロ.テクニカルレポート

No.11
2018年秋

株式会社ミガロ.

CONTENTS 目次

ごあいさつ

いつもミガロ.製品をご愛用いただき誠にありがとうございます。

「ミガロ.製品をご利用中の技術者の皆様に、日々の開発に少しでもお役に立つような技術情報をご提供したい」という思いから2008年に創刊した『Migaro.Technical Report』は、このたび第11号を発刊する運びとなりました。これもひとえに、ご多忙中にもかかわらず『Migaro.Technical Award（お客様論文)』にご寄稿いただいた多くのお客様、ならびに『Migaro.Technical Report』に対して貴重なご意見・ご要望をお寄せ下さった皆様のご支援の賜物と、心より感謝をしております。

今年IBM iは、発表から30周年を迎えました。弊社では、お客様向けのシステム開発および開発ツールの販売を両輪として、長年にわたりIBM iとともに活動を続けてまいりました。主力製品のDelphi/400とSP4iは、IBM iをより便利にご利用いただくためのGUI化・Web化ツールとして、これまで大変多くのお客様にご利用いただいております。また、昨年より取り扱いを開始したValence（バレンス）は、超高速開発を実現する「ローコード開発プラットフォーム」へと進化し、IBM iの強力なツールとして積極的にご紹介してまいります。

さて、今回の『Migaro.Technical Report』も従来と同様に、第1部は「Migaro.Technical Award 2018お客様受賞論文」、第2部は「ミガロ.SE論文」の2部構成としています。

第1部の「Migaro.Technical Award」とは、日々アプリケーションの開発・保守に携わるエンジニアの方々の努力と創意工夫の成果を顕彰することを目的とし、「Delphi/400」「SP4i」「Valence」などの弊社製品をご利用中のユーザー様を対象に実践レポート（論文）を公募し、厳正な審査・選考のうえ表彰する制度です。昨年に引き続き、従来のお客様論文に当たる「部門1」と「業務課題を解決した開発技術・テクニック」を簡潔にまとめていただく「部門2」の2部門構成といたしました。

今回のお客様論文は、「Excelテンプレートによるカスタマイズ性の高い帳票出力機能の開発」や「iPadによる工場での製品検査システム開発」など、創意工夫にあふれる論文を多数ご寄稿いただきました。

第2部「ミガロ.SE論文」では、弊社SEによる技術論文を掲載しております。今回は、「モバイル開発などに役立つ新しいRAD Serverによる多層アプリケーション開発」や「天気予報など各種Webサービスとの連携手法」、「Googleマップとの連携手法」など、さまざまなテクニックを開発に活かしていただくための技術情報をご紹介しております。本レポートが少しでも皆様の開発・保守のお役に立てば幸いです。

最後に『Migaro.Technical Report』第11号を発刊するにあたりまして、多くのお客様・パートナー様にご支援、ご協力をいただきましたことを、この場をお借りして、あらためて厚く御礼を申し上げます。

2018年秋

株式会社ミガロ.
代表取締役社長
上甲 將隆

Migaro. Technical Award 2018

お客様受賞論文／ミガロ. テクニカルアワード

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

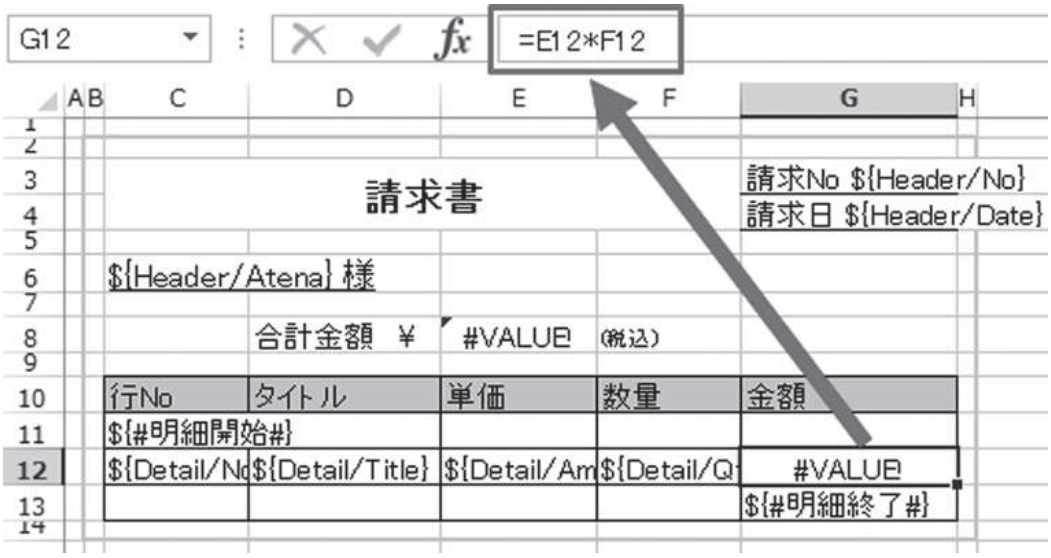
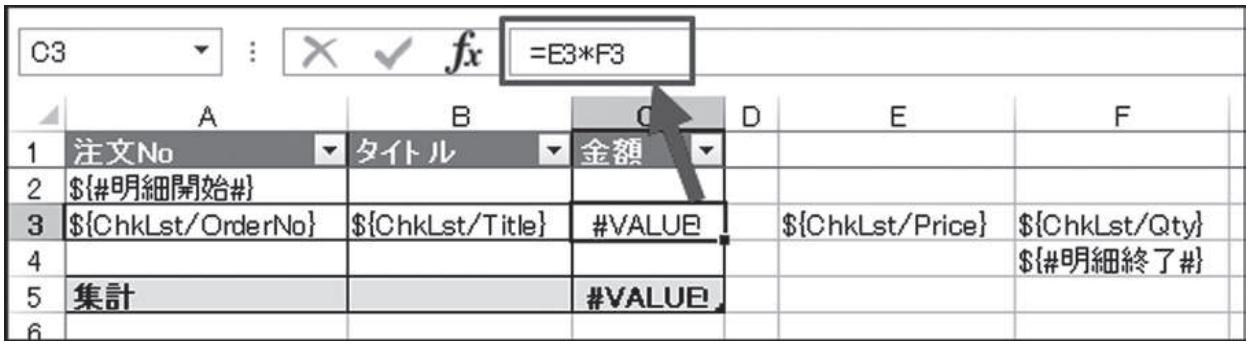


図3　リスト形式の帳票テンプレート



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

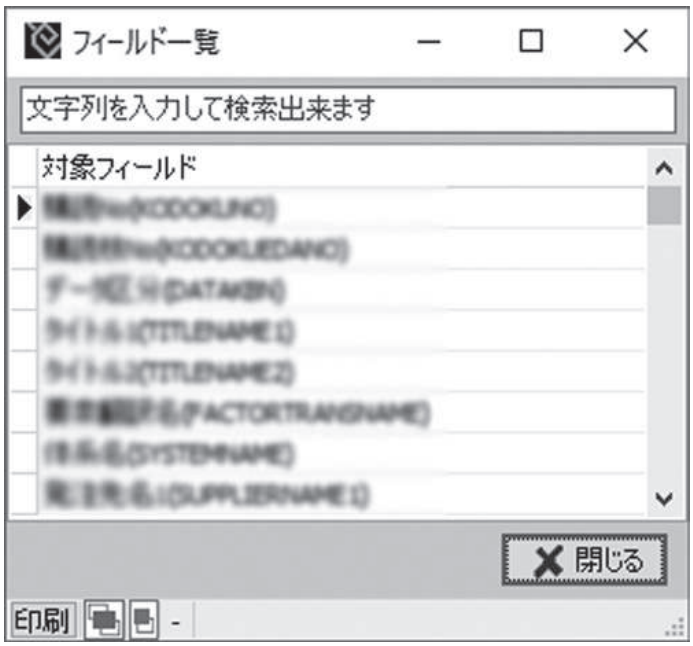
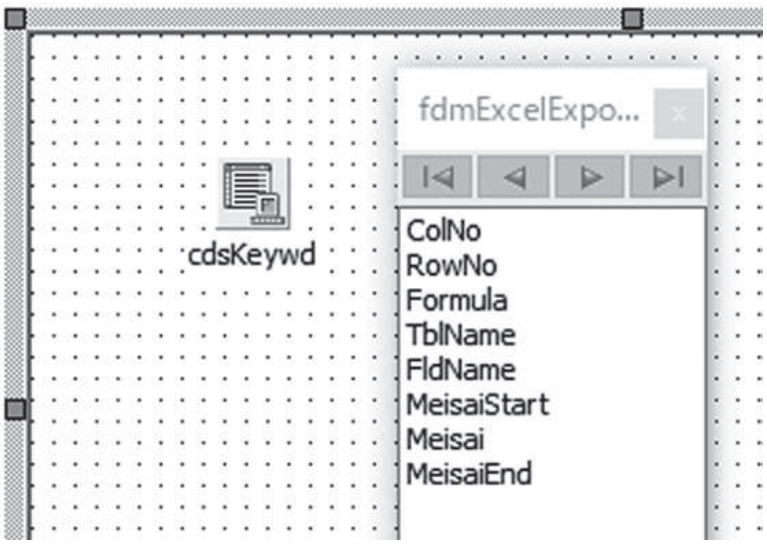


図5　おおまかなプログラムの流れ

```
◇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]);
```

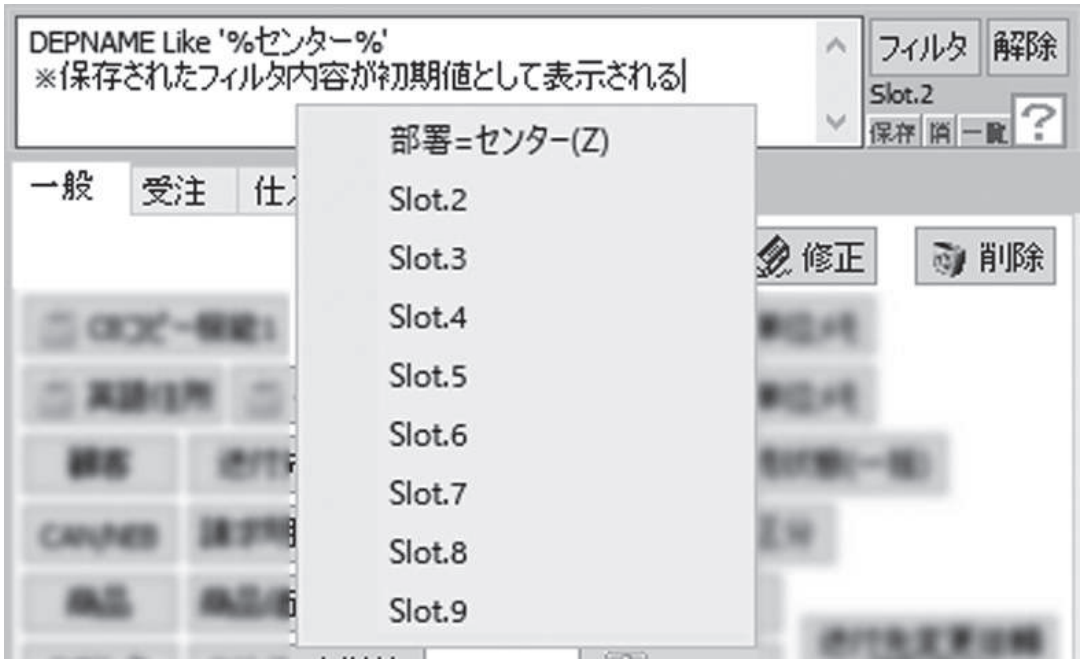
図11　単票部分のデータ差し替え処理サンプル

```
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



ゴールド賞

SP4iの活用による製品検査チェックシステムの構築
−iPadによる検査入力で各工程でのチェック漏れを防止する

八木 秀樹 様
極東産機株式会社
システム開発室
課長

極東産機株式会社
https://www.kyokuto-sanki.co.jp/

製造機器メーカーとして創業以来、職人さんの快適な職場環境作りと消費者の豊かな生活空間作りを２本柱として、伝統技術と先端技術の融合により、ユニークなオリジナル商品を開発。畳製造機器はもとよりインテリア施行省力機器、カーテン縫製機器等、幅広く事業を拡大している。

業務課題は検査漏れの撲滅

極東産機は、畳業界・インテリア施工業界のニーズに合わせたオリジナル製品を開発・製造している。工場では、畳・インテリアの施工時に利用する機械を製品として製造している。製品製造時の検査段階で、各検査項目の検査結果の記録を用紙に記入していたが、記入漏れがあった場合でも誤って完成処理されてしまうケースが見られた。

このため社内では、「検査チェック漏れまたは不合格の項目があれば、次の工程へ進まないようなシステムはできないか」「検査記録をタブレットなどでその場で入力したい」「検査項目の追加や改定を迅速に行いたい」「検査記録結果を画面で確認したい」といった要望が多かった。

以前、取引先との Web-EDI を JC/400 で構築したこともあり、JC/400 の後継製品でタブレットにも対応している SmartPad4i（以下、SP4i と記載）の採用を決めた。

採用決定理由は、次のとおりである。

(1) JC/400 のスキルをそのまま活かせる
(2) iPad などモバイル端末のシステムを開発できる
(3) IBM i とリアルタイムにデータ処理できることで、検査チェック漏れがあれば、次の工程へ進まないシステムの構築が可能である

システム構築･運用における問題点と対策

SP4i と iPad による工場での製品検査システム構築に際して、以下の課題を洗い出し、その対策を行った。

【課題 1】タブレットを工場内の各検査場所で使用したい。
【対策 1】工場内に無線 LAN 環境を増強した。

【課題 2】タブレットは iPad を採用したが、作業者が iPad 操作に不慣れで馴染めない。
【対策 2】iPad に馴染むため、若手中心で iPad の使用方法の講習会を開催した。教育を受けたメンバーから、他のメンバーへ指導してもらうこととした。

【課題 3】製品（機械）の種類（以下、機種）によって検査項目が異なる。
【対策 3】機種ごとの検査項目を IBM i のマスタに登録する。ユーザー自身が PC 端末から IBM i に登録できるようにした。

【課題 4】検査によっては数値の記入が必要なものがある。
【対策 4】数値入力にも対応できるようにした。また入力した数値によって、自動的に合否を判定できるようにした。

図1　検査担当者･承認者別メニュー

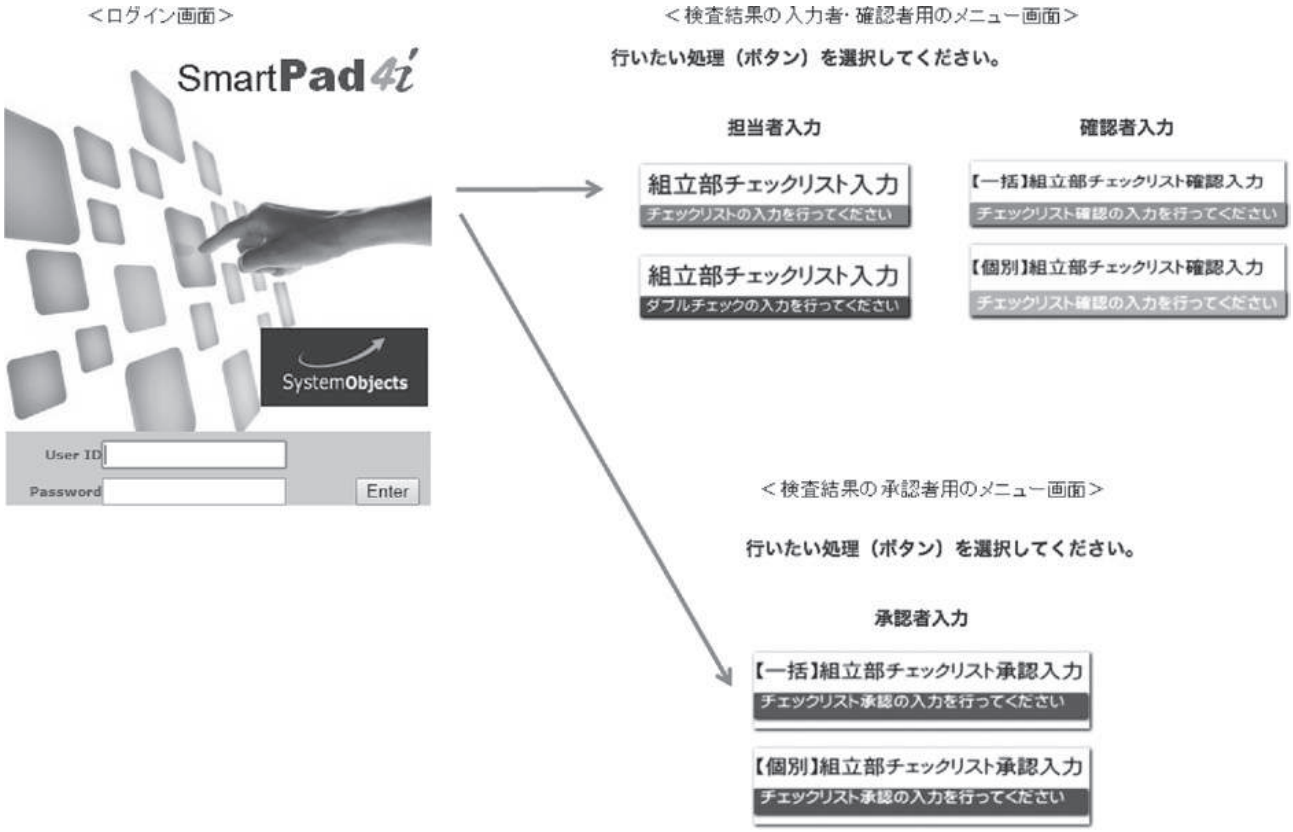
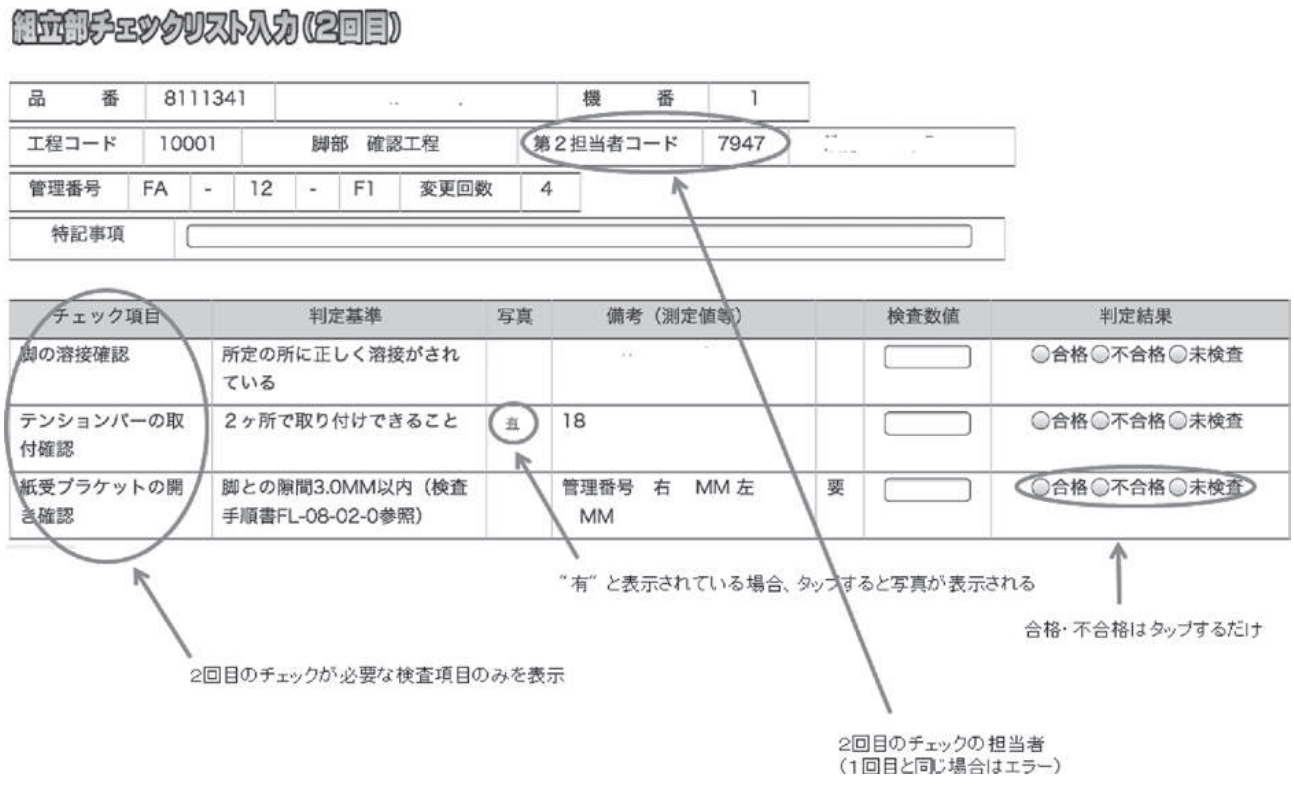


図2　2回目検査および写真連携



【課題5】すべての機種の検査について、紙のチェックリストからiPadでのチェック入力に変更できるか。

【対策5】機種ごとに具体的な検査項目が違うため、まずは主要1機種に限定しiPadに切り替えて運用することにした。その後、計画的にiPadで検査入力する機種を増やすことでスムーズに移行できた。

【課題6】検査チェックシステムの承認機能をiPadシステムでどのように実装するか。

【対策6】紙で運用していた場合は検査担当者、確認者、承認者と検印していた。iPadではログインIDによって、メニューを分けることで対応した【図1】。入力時の情報（いつ・誰が）は自動記録されるので、入力した際の情報の照会・確認が可能になった。

【課題7】重点検査項目についてはダブルチェック（他の担当が再度チェックして記入する）を行う必要があるが、同じ運用が可能か。

【対策7】各機種の検査項目の登録で、ダブルチェックが必要な項目を検査項目として登録する。基本的な検査チェックの入力項目は、1回目のチェックで入力し、別の担当者が2回目のダブルチェック項目だけを入力することとした。【図2】

【課題8】検査項目によっては、合格時の写真を見たい場合があるが対応は可能か。

【対策8】入力画面に写真有無の項目を設置し、写真「有」をタップすると合格時の写真を表示させることとした。【図2】

【課題9】紙の運用では特定の機種の検査時に、図面に検査結果を記入している。iPadで同様の対応が可能か。

【対策9】従来の紙の図面と同等のレイアウトをiPadの入力画面上に表示し、その画面に検査結果を入力することとした。【図3】

スムーズな稼働へ向けての工夫

今回のiPad検査システム開発プロジェクトを進めるにあたり、考慮した点を記載する。

(1) 検査対象製品を1機種限定で運用開始することとし、そのために必要なシステムの要望事項を絞り込み、システム構築を進めた。
(2) 検査項目のマスタ化を関連部門と進め、合否判定や数値入力、写真添付、図面添付などデータ整備を進めた。また、類似の機種は検査項目が共通である場合が多いので、検査項目の登録機能も共通化できるようにした。
(3) iPadの操作方法などシステム運用に詳しい現場メンバーを数名、事前に教育した。
(4) iPadからの入力をできるだけコード化することにより手入力を簡素化し、入力作業が負担にならないようにした。【図4】
(5) システム作成における疑問点を解消する。SP4iの開発テキストには、よく使用する機能の解説や事例があるため、テキストを読むだけでほとんどの問題点を解決できた。エラーの原因が推測できない場合は、ミガロ.のテクニカルサポートを積極的に活用した。このサポートにより、計画を遅らせることなく進められた。

SP4iの活用による製品検査チェックシステム本稼働の所感

「検査チェック漏れ防止」という目標の達成

検査担当（iPad）→確認担当（iPad）→承認者（iPad）→製品完成の入力（PC端末）のワークフローが確立し、検査チェックの入力漏れや検査結果が不合格であれば、次の段階へ進めないようにした。結果として、検査合格品の場合にのみ製品完成の入力が可能となる仕組みづくりができた。

直感的な操作性によるスムーズな運用

直感的に操作できる画面であるのに加え、iPadの操作説明会を事前に何回か実施したこともあり、運用開始後の問い合わせはほとんどなかった。担当者、確認者、承認者別のメニューも、各メニューの用途が一目でわかるようにコメントなどを工夫した。【図1】

またスマホの操作に慣れていたメンバーが多いことも、順調に運用が開始できた要因と考えられる。

iPadの利点

検査の都度、所定のパソコン設置場所へ登録しに行く必要がなく、iPadだけで入力できる【図2】。また検査結果入力後に、誰が検査したかを手元のiPadからすぐに確認できる【図5】。このため、iPadを利用した本システムは現場からは好評だった。

今後の展望

iPadを活用した生産現場の在庫管理システム（IBM i）をSP4iで構築済みだが、現在、このシステムのリニューアルに取り組んでいる。改善のポイントは、ハンディターミナルでは実現できなかった画面の情報量と操作性をiPadとiPad用のバーコードスキャナにより実現すること。これにより、在庫管理をさらに効率化する予定である。

今後とも、タブレット活用による生産現場での作業効率向上と見える化できるシステム改善をさらに推進していきたい。

M

図3　紙の図面の内容をiPad上に表示

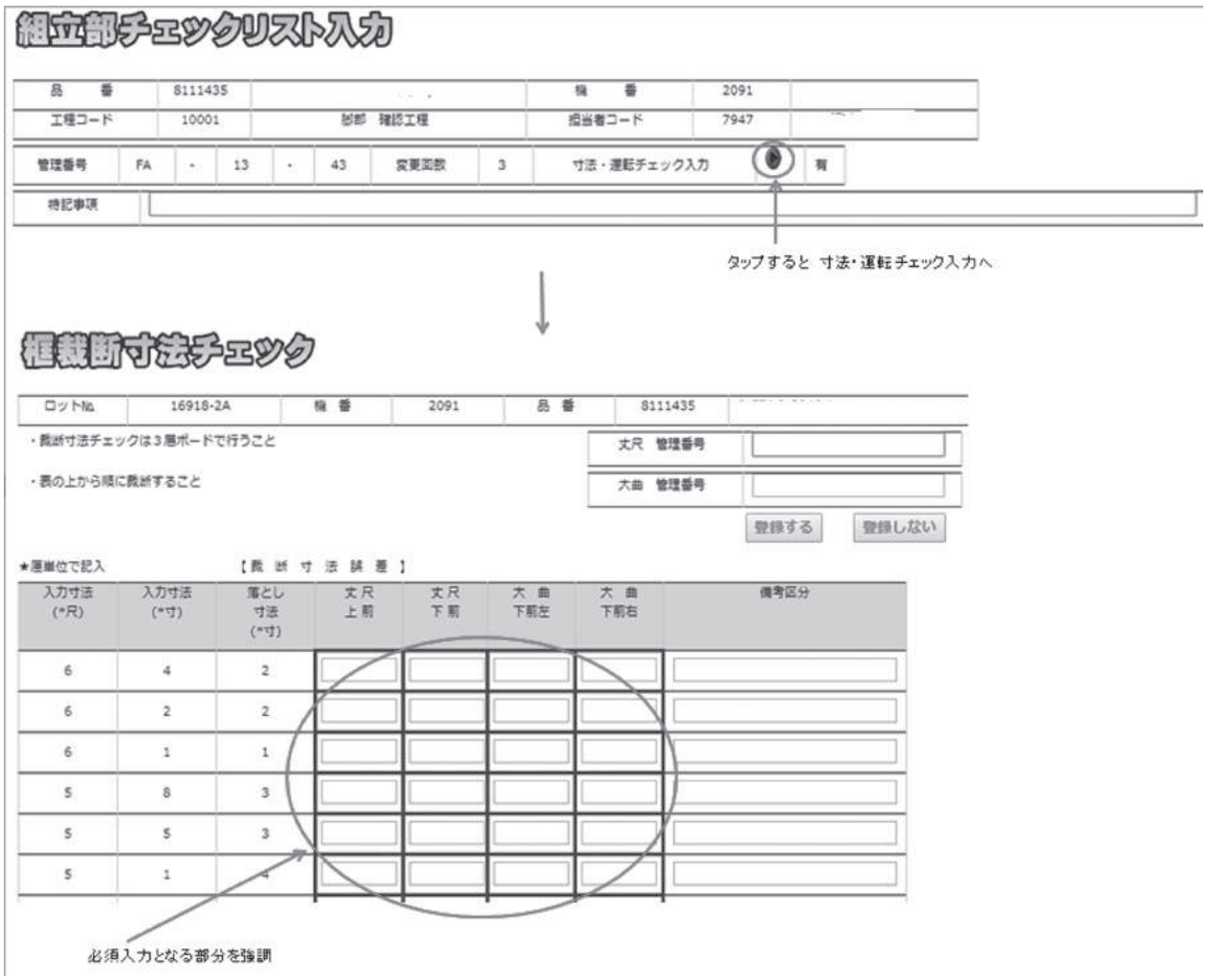


図4　検査チェック入力状況照会

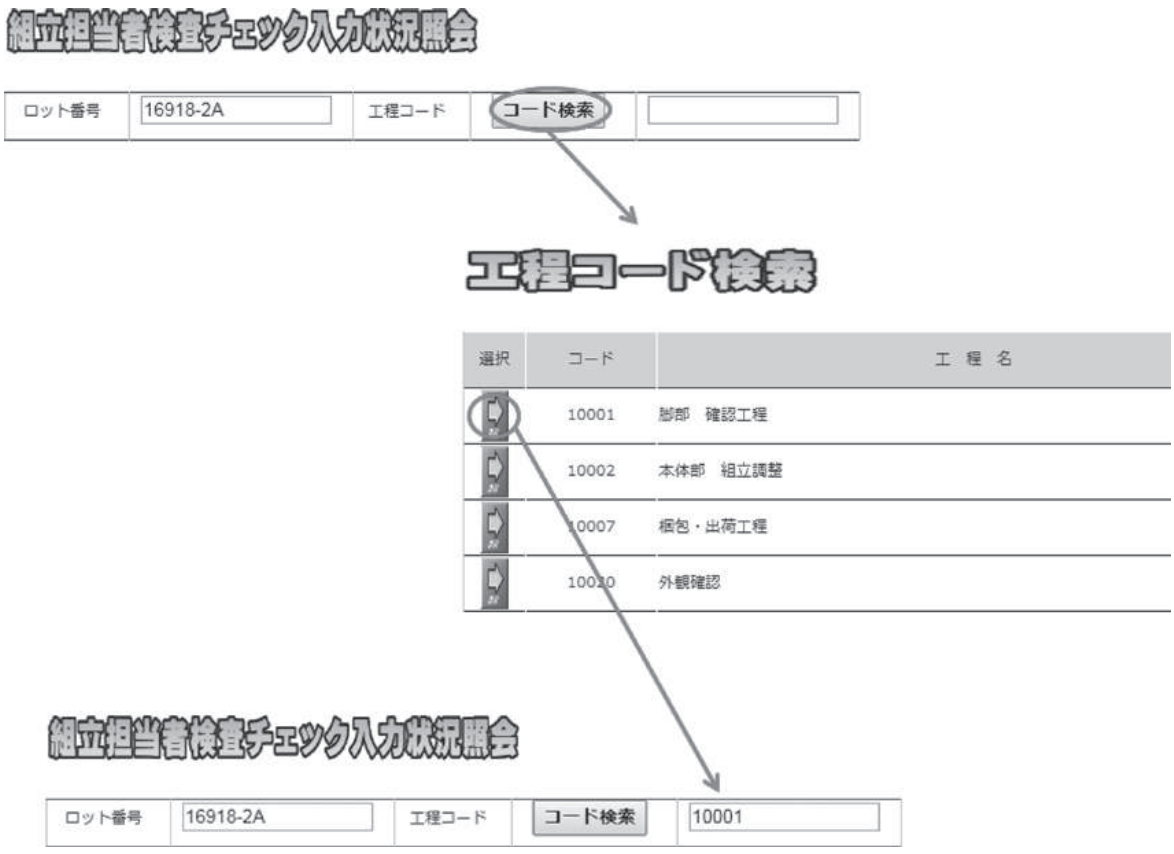


図5　検査結果の詳細照会

組立部チェックリスト確認詳細照会

品　番	811 :36		工程コード	10001	脚部 確認工程
機 番	5052	担当者	F001	第2担当者	8068

確認者	1601	

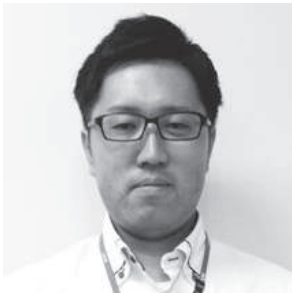
チェック項目	判定基準	備考（測定値等）	検査数値	判定結果
（1）脚部の開き具合	脚部上側内寸と支点板部内寸測定　（下側）－（上側）＝5mm～15mm		5.00	合 格
（2）原反芯棒、上蓋の取付確認	無理なく取り付くこと。			合 格

優秀賞

配車支援システムをDelphi/400で再構築

村上 稔明 様

ライオン流通サービス株式会社
事業部



ライオン流通サービス株式会社
http://www.lion-logi-s.co.jp/

ライオン株式会社100%出資の物流子会社として、全国のグループ物流拠点、および協力物流事業者への委託業務を統括。倉庫管理・在庫管理・輸配送管理など、グループの物流業務全般を担っている。 輸配送におけるCO_2削減など物流業務改善への積極的な取り組みを行っている。

業務課題

ライオングループの物流を担う当社では、配車支援システムにより、翌日の運送計画を立案している。

従来のVisual Basicで作成した配車支援システムには、以下の問題点があった。

(1) IBM i上の各データを毎回PCへダウンロードする必要があるため、処理に時間がかかる。
(2) 共有のPCで稼働しているため、別の担当者が操作中は、操作ができない
(3) メイン処理の「運行計画編集処理」では、ドラッグ&ドロップで使用できず操作しにくい。

これらの問題点を解決するために、IBM iのデータを直接参照し、各自のPCから操作可能なDelphi/400を使用して、システムを再構築することにした。

画面機能の詳細説明

メイン処理の「運行計画編集処理」をDelphi/400で新規作成した。

表示ボタンを押下すると、IBM iの運行関連DBをSQLで抽出し画面に表示する。【図1】

一番左の「車番」の列には、車両ごとの拠点（事業者）、予定／実績費用、可能積載重量が表示される。

第1運行以降の列には、配送拠点(FROM-TO)、到着／出発予定時間、編成番号、積載予定重量を表示する。当初スケジュールしていた車両から別の車両に運行計画を変更する場合、ドラッグ&ドロップによる運行編集が可能となった。

たとえば単一編成No.は、ドラッグ&ドロップにより運行編集できる【図2】。また同一編成No.の複数の運行計画をまとめて、ドラッグ&ドロップにより運行編集できる。【図3】

画面による業務課題の解決

各業務課題は、以下のとおり解決した。

(1) Delphi/400は、IBM iのDBから処理に必要なデータのみ読み込むため、全件ダウンロードによる処理時間は不要になった。
(2) 配車担当者は座席を移動することなく、各自のPCから配車支援システムの入力が可能となった。
(3) 運行計画のドラッグ&ドロップにより、新しい車両への割り付けが可能となった。さらに再計算処理（一時保管）ボタンの機能や、編成No.検索機能を新たに追加することにより、1日当たり数十分の時間短縮を実現した。

M

図1　初期画面

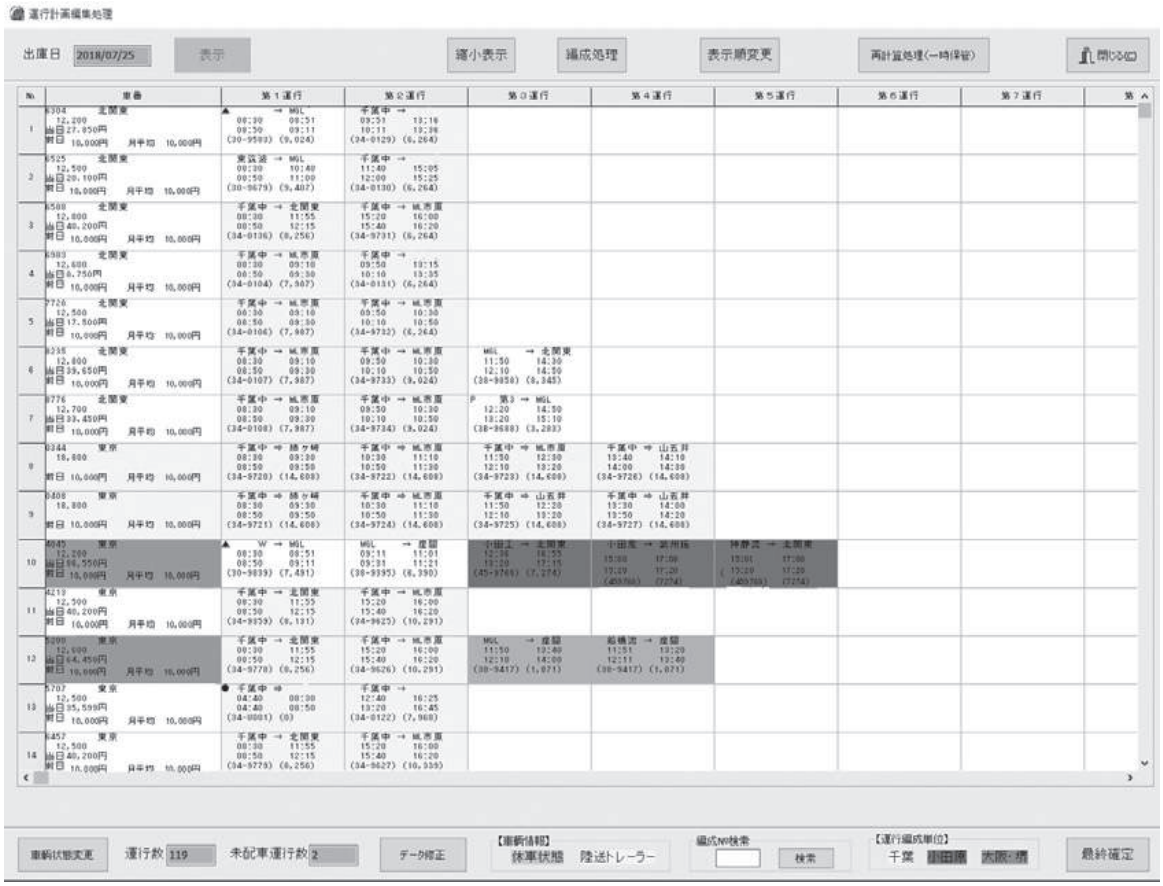


図2　運行計画ドラッグ&ドロップ（単一編成）

図3　運行計画ドラッグ&ドロップ（複数運行）

優秀賞

一般シール受注入力業務のDelphi/400化

寺西 健一 様

大阪シーリング印刷株式会社
IT 推進部
情報システム課
主査

大阪シーリング印刷株式会社
http://www.osp.co.jp/

1927 年創業以来、加工業としての本業に徹した堅実経営を貫き、主に凸版印刷を中心とした原紙製造から印刷までの一貫生産工程を軸に、全国をオンラインで結ぶ営業・生産ネットワークを活用。シール業界のリーディングカンパニーとして、印刷の枠を越えた総合パッケージメーカーとして事業を展開している。

業務課題

従来、一般シールの受注入力を PC の 5250 画面で行っていたが、画面項目が約 150 まで増加したため、以下のような問題により入力が困難となった。【図 1】

・タブが使えないので、すべての項目を 1 画面で入力する必要がある。
・画面項目を長い間追加してきた経緯により、項目の並びが業務の流れに沿った順番になっていない。
・画面上に説明が少ないため、慣れたユーザーしか入力ができない。
・連続入力時に 5250 セッションのマクロ機能を使用しているが、マクロ入力がずれた項目に登録され、画面がかたまる問題が指摘されていた。

そこで、GUI 化による問題解決に向けて、Delphi/400 で受注入力画面を作成・開発した。

技術課題

入力者が従来の画面に慣れているので、Delphi/400 画面にも同様の動きを組み込むことが課題となった。

具体的には、以下の 3 つの要望があった。

(a) 右 Ctrl キーによる実行動作
(b) タブによる複数画面の項目展開
(c) コピー新規登録、ひな形からの入力

また今回の GUI 化をきっかけに、以下の課題にも取り組んだ。

(d) 入力者ごとのデータ表示
(e) 難しい項目や専門用語などに対するヘルプ表示

技術課題の解決策

物理ファイルの構成を QTEMP ではなくメンバーを使用するようにしたうえで、上記の課題に対し、以下のように解決した。

(a) 右 Ctrl キーによる実行動作

Tform のイベントのキーボード操作に VK_CONTROL（仮想キー）を組み込み、5250 セッションによる右コントールキーを有効にした。【図 2】

(b) タブによる複数画面の項目展開

TPageControl を配置し、複数タブの入力項目を配置するように対応した。

(c) コピー新規登録、ひな形からの入力

コピー機能はメンバー指定からのデータのみを使い、そのままフィールドをコピーし、その値を埋める形で実現させた。

(d) 入力者ごとのデータ表示

メンバーに保存したデータを使用して、入力者ごとにデータを見られるようにした。具体的には、TTable コンポーネントでメンバー指定を行った。【図 3】

図1　従来の5250画面

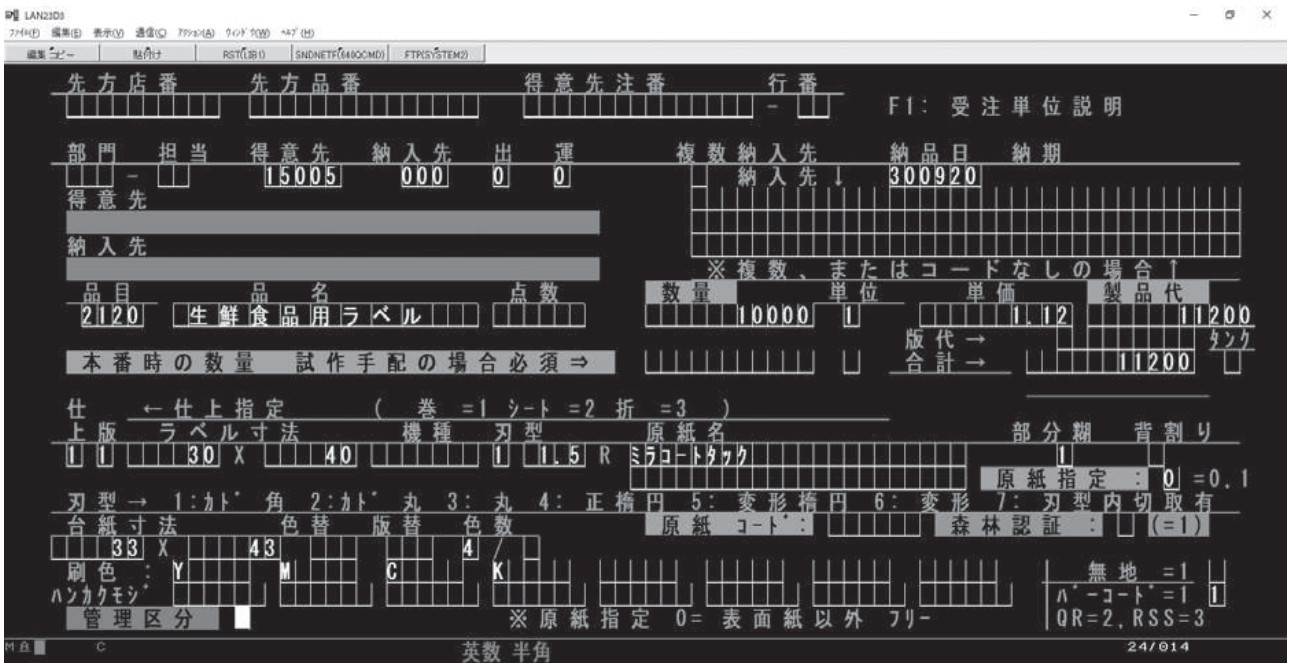


図2　右Ctrlキーの制御

```
// コントロールキーが押されたか？
if Key = VK_CONTROL then
begin
  // 右のコントロールキーが押されたか？
  if RControl then
  begin
    Key := 0;
    // フォーカスが移せるかどうか判別し、
    // Trueであればフォーカスをセットする。
    if BtnNext.CanFocus then
      BtnNext.SetFocus;
    BtnNext.Click;
  end;
end;
```

図3　物理ファイルのメンバー指定

```
// LBDL1F（入力中データ）
  TblLBDL1F.TableName := cWorkLib + 'LBDL1F(' + FileMember + ')';
//    WriteLog('TblLBDL1F.TableName');
```

(e) 難しい項目や専門用語などに対するヘルプ表示

ヘルプ専用画面を作成し、難しい項目や専門用語をクリックして表示できるようにした。

業務課題解決と効果

完成した画面は、業務の流れに沿ってタブごとに入力項目が整理されたことで、1件当たりの入力時間が5分から2分弱へと、大幅に工数を削減できた。【図4】

またヘルプ画面を表示させることにより、入力方法に関する知識をもたなくても入力できる画面を実現した。【図5】

今後も機能を追加し、より効率的に受注業務が行えるように進めていきたい。

M

図4　Delphi/400による受注入力画面

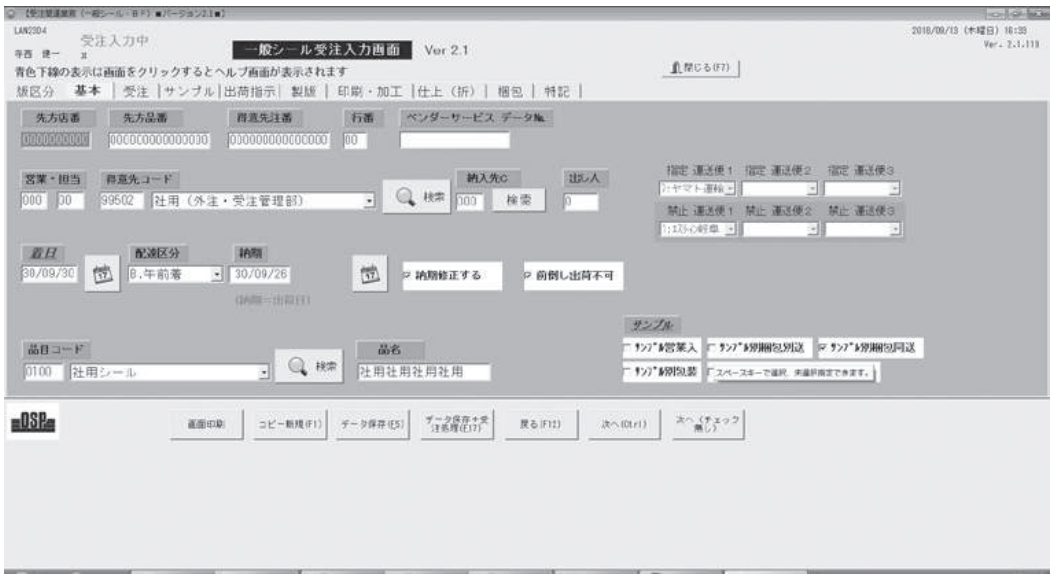
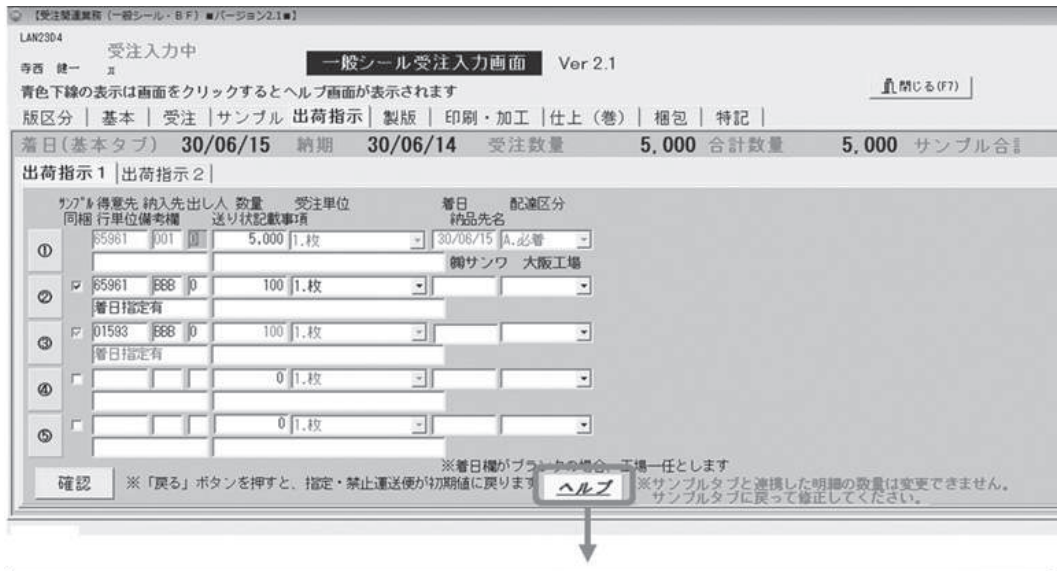
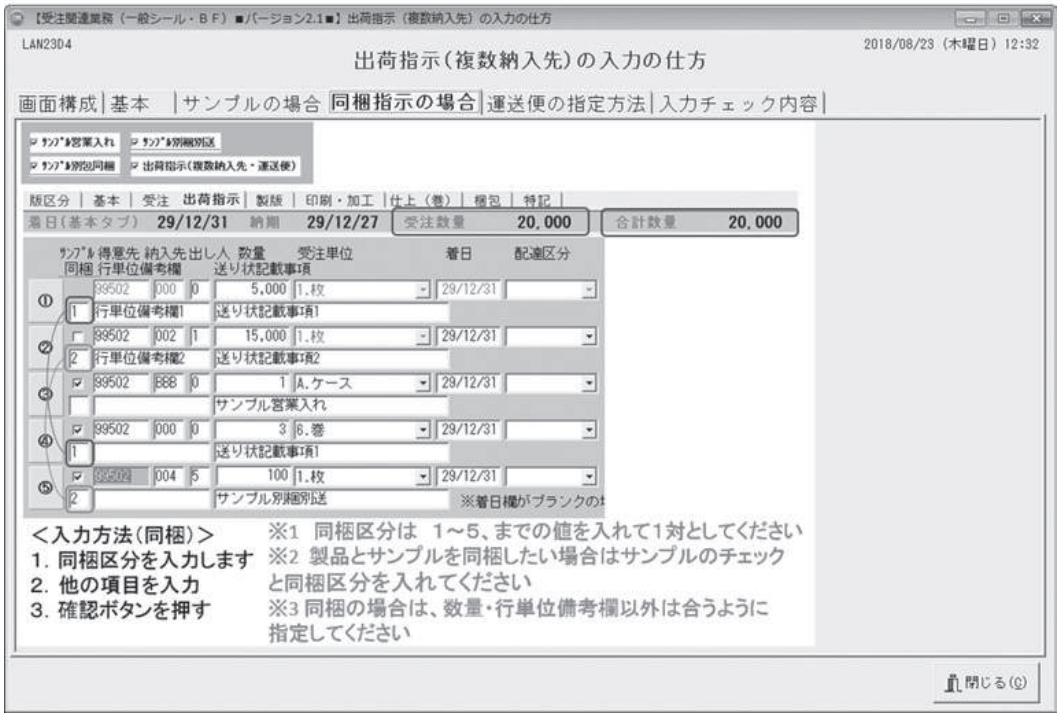


図5　ヘルプをクリックして詳細なヘルプ画面を呼び出し





優秀賞

Delphi/400による無線ハンディターミナルのデータ集約の仕組みの実装

寺西 健一 様

大阪シーリング印刷株式会社
IT 推進部
情報システム課
主査

大阪シーリング印刷株式会社
http://www.osp.co.jp/

1927 年創業以来、加工業としての本業に徹した堅実経営を貫き、主に凸版印刷を中心とした原紙製造から印刷までの一貫生産工程を軸に、全国をオンラインで結ぶ営業・生産ネットワークを活用。シール業界のリーディングカンパニーとして、印刷の枠を越えた総合パッケージメーカーとして事業を展開している。

業務課題

印刷工場において､印刷に用いる「版」の入庫を確認し、版管理データとしてIBM i 基幹システムに登録している。

従来の登録業務は、工場内の各作業場所でハンディターミナル（バッチ式）によるデータ読み取り･蓄積の後､ハンディターミナルを PC に設置した置台（通信ユニット）に置く、という 2 段階でデータ転送していたため､手間がかかっていた。

そこで、各作業場所から直接データの収集・登録を行うために、バッチ式に変えて無線ハンディターミナルを導入することとした。

技術課題

従来は 1 台の PC だけで実装された仕組みだったので、実績データのファイル名を固定させて FTP による通信を行っていた。しかし無線による通信に変わったため、任意のタイミングでデータを送信する仕組みの実装が必要となった。

具体的には、以下の 3 点を仕様として組み込む必要があった。

(a) IBM i へのデータ送信を定期的に自動処理する（注 1）
(b) ハンディターミナルからの実績データを 1 つにまとめて IBM i に送信
(c) ハンディターミナルからのファイル名競合の回避（注 2）

(注 1)　ハンディターミナルのデータは、中継用のサーバー区画（シンクライアント)を経由してIBM iに登録し､サーバー区画上で Delphi/400 アプリケーション（待ち受けアプリ）を起動する仕組みとした。中継用に、PC の代わりにシンクライアント区画を利用することにより、セキュリティ面と環境管理に関しても考慮した。【図 1】

(注 2)　複数のハンディターミナルから待ち受けアプリに同時にデータを送信する場合、Delphi/400 の待ち受けアプリ側で受信ファイル名が競合して、処理がロックする問題があった。ハンディターミナル側でも待ち状態になり、ロックが解除できなくなる。

技術課題の解決策

前述の技術課題に対して、Delphi/400 の待ち受けアプリにて、次のように解決した。

(a) IBM i へのデータ送信を定期的に反復処理

ハンディから実績データを IBM i へ送信する頻度を画面上で設定可能とした。具体的には、Timer を実行させる時間間隔を指定できるようにした。【図 2】

(b) ハンディターミナルからの実績データを 1 つにまとめて IBM i に送信

Delphi アプリケーションでは、ハンディから FTP 送信されたフォルダを、指定された時間間隔で検索（ファイル *.txt）し、ファイルがあれば、1 つのテ

図1　データフロー

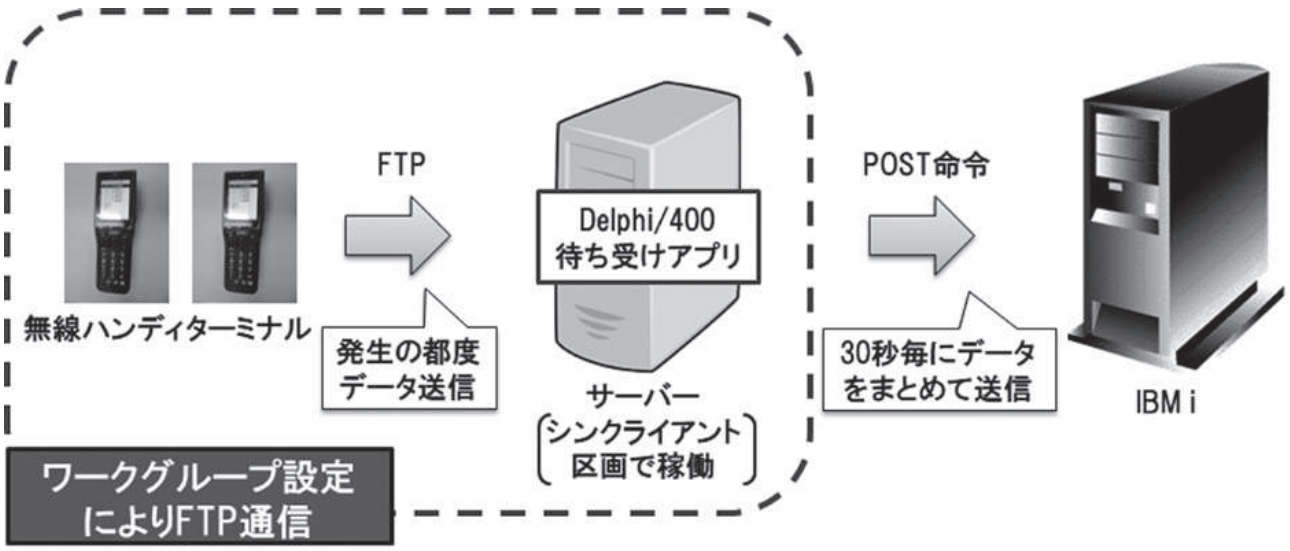
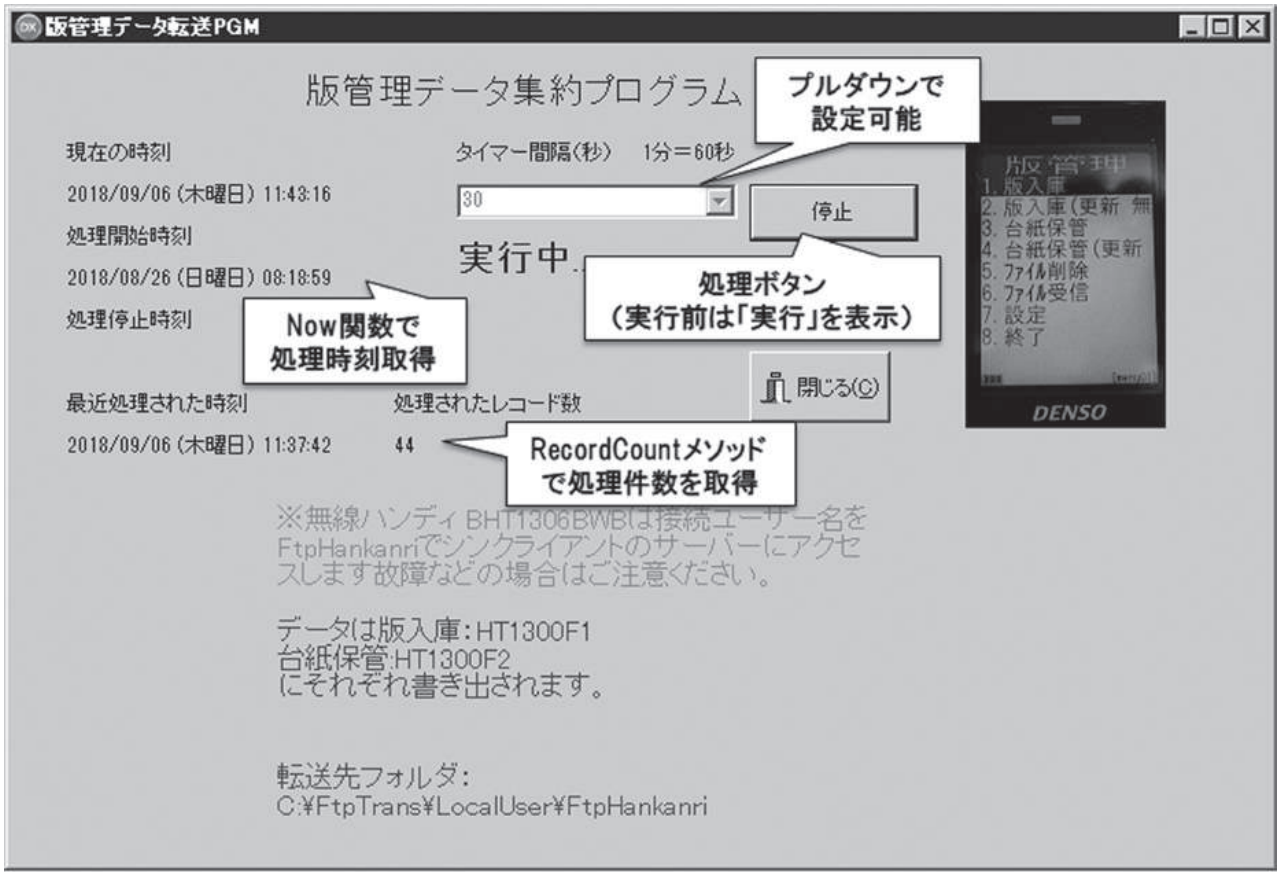


図2　待ち受けアプリ画面



キストデータにまとめる。まとめたデータはTtableコンポーネントを経由して、IBM iへ送信する。【図3】

(c) ハンディターミナルからのファイル名競合の回避

ハンディターミナル側のアプリケーションでは、待ち受けアプリに送信する実績データを、「端末ID＋年月日.txt」という形式にして可変のファイル名とすることで、同一ファイル名が競合することを回避した。

業務課題解決と効果

従来、置台式にて転送処理を行っていたため、置台を設定しているPCが変わるたびに転送を設定する必要があったが、無線化により環境整備の手間がなくなった。

また、現場の担当者からも置台まで行く動作が省略できてよい、との評価が寄せられた。具体的には、1日当たり30分／人の工数短縮という大きな効果を挙げることができた。

M

図3　待ち受けアプリソースコード

```
procedure TForm1.Timer2Timer(Sender: TObject);
var
slCSV:    TstringList;
slLine:   TStringList;
slGCSV:   TstringList;
i:        Integer;
SR:       TSearchRec;
//FND : Integer ;

 begin
      slCSV :=TStringList.Create;
      slLine :=TStringList.Create;
      slGCSV :=TStringList.Create;
```

FindFirst命令により、*.txtの内容のファイルをすべて読みながら、一つのファイルへ集約

```
 if FindFirst('C:¥FtpTrans¥LocalUser¥FtpSiage¥*.txt',FaAnyFile,SR) =0 then
  begin
     repeat
//datからの読み込み
    try
        slCSV.LoadFromFile('C:¥FtpTrans¥LocalUser¥FtpSiage¥'+ SR.Name);  //
    except
         on e0: Exception do
         begin
          Abort; // サイレント例外
         end;
    end;
    //レコードの行数分(Count-1)処理を繰り返す。
      for i:=0 to slCSV.Count - 1 do
   begin
       slLine.CommaText:=(slCSV[i]);
        if slLine.Count >1 then
   end;
```

```
//ファイルを保存(Basho.txt)
//       ShowMessage('Basho.txtファイルを保管');
      slGCSV.SaveToFile('c:¥Siage¥Siage.txt');
//処理した件数を画面へ表示
      Label9.Caption:=IntToStr(slGCSV.Count);
//処理した時刻を画面へ表示
       stClock2.Caption := FormatDateTime('yyyy/mm/dd (aaaa) hh:nn:ss',Now());
//ファイルがなくなったら探すのを止める。
    until FindNext(SR) <> 0;
     FindClose(SR);
  end;
```

TSearchRecを検索用に定義し、ファイルをすべて読み終わるとClose命令により閉じる

```
 try
   // CSVファイルをオープン
  AssignFile(CSVFile, 'C:¥Siage¥Siage.txt');
  Reset(CSVFile);
   try
     while (eof(CSVFile) = false) do
    begin
       // CSVの読み込み
       ReadLn(CSVFile, str);          // CSVの１行を読み込む
       st.CommaText := str;           // 文字列を分割する
       Number := StrToIntDef(st.Strings[1],0);
       case Number of
         //仕上げチェックの場合
         1 : begin
         //文字列をテーブルに書き込む
         //Table1.Append;                  // レコードの追加
              j := 0;
              TblSGCHKF1.Insert;                  // レコードの挿入
         //for～to～doは初めのパラメータが2つ目のパラメータの
         //値になるまで実行。
         //この場合、ファイルの件数分Table1(I5)にデータを
         //追加する。
             for j := 0 To st.Count-1 do
              TblSGCHKF1.Fields[j].AsString := st.Strings[j];
              TblSGCHKF1.Post;
```

まとめたファイルはTtableコンポーネント経由でPost命令でデータを直接書き出し

Migaro.
Technical Report
2018

ミガロ.SE 論文／ミガロ. テクニカルレポート

薬師 尚之

株式会社ミガロ.

システム事業部 システム2課

[Delphi/400]

OLEを利用したExcel出力のパフォーマンス向上手法

略歴
1985 年 11 月 22 日生まれ
2008 年 3 月 阪南大学 流通学部卒業
2008 年 4 月 株式会社ミガロ. 入社
2008 年 4 月 システム事業部配属

現在の仕事内容：
Delphi/400 を利用したシステムの受託開発を担当し、基本設計から納品・フォロー、保守作業に至るまで、システム開発全般に携わっている。

1.はじめに

Delphi/400 にバンドルされる帳票ツールを利用して帳票機能を開発することが多いが、Delphi/400 のバージョンアップによって帳票ツールが変更された場合、帳票機能を移行または作り直しの検討が必要となる（バンドルされる帳票ツールは、Delphi/400 Version 5 ～ 7 が「QuickReport」、7 ～ XE が「Rave Reports」、XE 3 以降は「FastReport」である）。

このような帳票ツールに依存した変更を解決する方法の 1 つとして、OLE（＊）を利用し Excel をベースにした帳票機能を実装することもできる。

OLE での Excel 利用は、Delphi/400 から比較的簡単に実現できるが、大量データの処理には向いておらず、パフォーマンスが落ちることもある。この課題についてはプログラムロジックを工夫することで解決が可能である。本稿では、OLE を利用した基本的な Excel 出力方法から、大量データを出力する場合のパフォーマンス向上手法を解説する。

2.OLEを利用した基本的なExcel出力

本章では、Excel 出力プログラムの作成例を題材に、OLE の基本的な操作方法について解説する。

Excel 出力プログラムの概要

- IBM i より売上情報を取得し、明細を Excel に出力する。
- 明細出力時、各営業所単位で合計金額を出力する。
- Excel 出力後に保存ダイアログを表示し保存する。

開発環境：Delphi/400 10.2 Tokyo および dbExpress

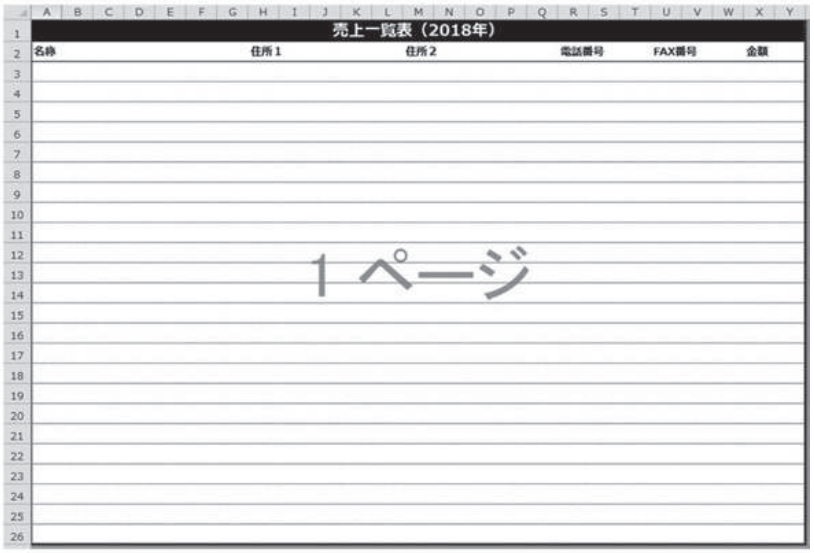
なお本稿では、帳票テンプレートは事前に Excel で作成したものを利用する。Delphi/400 から罫線や書式設定の操作を行うことは可能だが、その回数が多くなるほどパフォーマンスの低下に繋がる。そのため、本稿ではあらかじめ Excel テンプレート【図 1】を作成し、Delphi/400 からの操作回数を極力減らしている。また Excel でテンプレートを作成することで、帳票項目の書式設定も Excel 側に持たせることができるため、そうした書式変換のプログラミングも不要となる。

【図 1】の Excel テンプレートを利用した Excel 出力のロジックが【ソース 1】、【ソース 2】となる。

＊OLE：Object Linking and Embedding の略称

マイクロソフトが提供する機能の 1 つで、複数のアプリケーション間でデータの転送や共有を行うための仕組みを指す。Excel は OLE サーバーとなり、他のアプリケーションから操作可能（操作のためのメソッドが用意されている）。実行のためには、Excel が導入されている環境が前提となる。

図1　Excel テンプレート(例)



ソース1

```
procedure TForm1.Button1Click(Sender: TObject);
var
  ovExcel: OleVariant;
  ovWorkBook: OleVariant;
  ovWorkSheet: OleVariant;
  sFileName: String;
  iRow: Integer;
  sFL1: String;
  cShokei: Currency;
begin
  // 出力データ取得 …①
  SQLQuery1.Close;
  SQLQuery1.SQL.Text := 'SELECT * FROM TRURI ORDER BY URIFL1';
  SQLQuery1.Open;
  try
    // Excelオブジェクト生成 …②
    ovExcel := CreateOleObject('Excel.Application');
    try
      // Excel非表示 …③
      ovExcel.Visible := False;

      // フォーマット読み込み …④
      sFileName := IncludeTrailingPathDelimiter(ExtractFileDir(Application.ExeName)) + 'Format.xlsx';
      ovWorkBook := ovExcel.Workbooks.Open(sFileName);
      ovWorkSheet := ovWorkBook.WorkSheets[1];

      // 出力準備
      iRow := 3;
      cShokei := 0;
      sFL1 := SQLQuery1.FieldByName('URIFL1').AsString;  // 小計出力用営業所

      // データ出力
      while not SQLQuery1.Eof do
      begin
        // セルへ転送 …⑤
        ovWorkSheet.Cells[iRow,  1].Value := SQLQuery1.FieldByName('URIFL2').AsString;   // 名称
        ovWorkSheet.Cells[iRow,  8].Value := SQLQuery1.FieldByName('URIFL3').AsString;   // 住所１
        ovWorkSheet.Cells[iRow, 13].Value := SQLQuery1.FieldByName('URIFL4').AsString;   // 住所２
        ovWorkSheet.Cells[iRow, 18].Value := SQLQuery1.FieldByName('URIFL5').AsString;   // 電話番号
        ovWorkSheet.Cells[iRow, 21].Value := SQLQuery1.FieldByName('URIFL6').AsString;   // FAX番号
        ovWorkSheet.Cells[iRow, 24].Value := SQLQuery1.FieldByName('URIFL7').AsCurrency; // 金額

        // 小計を計算
        cShokei := cShokei + SQLQuery1.FieldByName('URIFL7').AsCurrency;
```

①出力データ取得

TSQLQueryを利用してIBM iのファイルよりExcel出力用のデータを取得する。

② Excelオブジェクト生成

CreateOleObjectの引数に「Excel.Application」を指定し、Excelのオブジェクトを生成する。また、変数ovExcelに代入することで生成したオブジェクトをOleVariant型で操作可能にする。前提として、Delphi/400でOLEの各メソッドを利用するにはuses節に「ComObj」を追加する必要がある。

③ Excel非表示

Excelを表示したままにすると、プログラムのExcel操作が画面ですべて表示されてしまいパフォーマンスも低下するため、ovExcel.VisibleをFalseにして非表示にする。

④フォーマット読込

Excelテンプレートのパスとファイル名を取得（変数sFileName）し、Workbooks.Openで開いたブックを変数ovWorkBookに代入する。ovWorkBookのWorkSheetsプロパティでブックのシート番号を指定し、シートオブジェクトを取得する。これで、①と同様に、ブックとそのシートをOleVariant型で操作可能にする。

⑤セルへ転送

ここでExcelのセルへ値を出力する。

ovWorkSheetのCellsプロパティでセル位置（行および列のインデックス）を指定し、IBM iのデータベース・ファイルより取得した値を代入する。

代入した値はセル側に設定されている書式が適用される。

⑥ Excel保存

TSaveDialogを配置して、Excel出力後に保存ダイアログを開く。そのダイアログで指定したパスをSaveAsメソッドの引数に渡すことで任意の場所にファイルを保存できる。またFilterプロパティの設定により、保存時に選択できる拡張子の制御が可能である。本稿では「*.xlsx,*.xls」の2種類を指定可能にしている。【図2】

ファイル保存時の注意点を補足しておく。

ダイアログ上で「.xls」を指定して保存した場合、保存したファイルを開く際に拡張子が正しくない旨の警告が表示される。【図3】

これは、Officeのバージョン2007以降、規定のデータ保存フォーマットが変わり、過去のOfficeとは互換性のない形式で保存されることになったためである。対応方法としては、拡張子に「.xls」が指定された場合、SaveAsメソッドの引数にファイル形式を表す定数「56」（Excel 97-2003 ブック）を指定することで後方互換に対応可能となる。これにより保存したファイルは、開いた際に警告が表示されなくなる。この定数はほかにもPDFやCSV形式で出力可能であり、その一覧は以下のWebサイトにて紹介されているので、参考にしてほしい。

●参考URL：

https://msdn.microsoft.com/ja-jp/vba/excel-vba/articles/xlfileformat-enumeration-excel

（Googleで”XlFileFormat列挙”を検索すると上位に表示される。）

⑦ Excel表示

処理終了後にovExcel.VisibleをTrueにしてExcelを表示する。データ出力が完了した状態でExcelが表示される。

⑧オブジェクト解放

生成したOleVariant型の変数をUnassignedで解放する。

これでExcel出力処理は完成である。このプログラムで実際にExcelでの帳票出力を行った結果が【図4】である。

ソースを見ればわかるとおり、OLEを利用したExcelの出力自体は比較的簡単に実装できる。しかし、この方法では、大量のデータを出力する際にパフォーマンスがかなり悪くなってしまう。その原因としては、Delphi/400からセルに値をセットする際にアプリケーション間で通信が発生する（以下、通信と表記）からだが、実はこれが処理時間が長くなる大きな要因となっている。

本稿のテンプレートを例にすれば、1明細あたり6項目存在するため、1行出力する度に6回の通信が発生する。これが数百、数千件と、扱うデータ件数や項目数が増えると、その分通信が繰り返されるため、パフォーマンスに影響するのは明白である。パフォーマンスを向上させるには、いかにDelphi/400とExcelとの通信回数を減らすかが重要である。その手法については次章で触れる。

3.Excelの出力パフォーマンス

2章では基本的なExcel出力の手法を説明した。本章ではパフォーマンスを向上させる手法、つまりExcelとの通信回数を低減する方法を紹介する。

Excelに値を出力する際、各項目を1セルずつ出力するのではなく、出力する値を2次元配列などに記憶させ、特定のタイミング（改ページ時など）で一括出力することにより、通信回数を格段に低減させることができる。

その方法は2種類あり、クリップボード、もしくはOLEバリアント配列を利用することで実現できる。

しかし、前者のクリップボードについては以下の課題点がある。

●クリップボードを利用する上での課題点

①クリップボードの内容がプログラムで書き換えられてしまう

② WindowsVista以降、クリップボードの動作が不安定

①の課題点

クリップボードは、列ごとに「#9」（タブコード）、改行ごとに「#13#10」（改行コード）のリテラルを挿入することで、複数行・列の内容を格納でき、またその内容をExcelに一括で出力できる。しかし、この手法はユーザーの意図しないところでアプリケーション側からクリップボードの内容が書き換えられてしまい、ユーザーのコピー&ペースト操作などに影響を与えてしまうことがある。

②の課題点

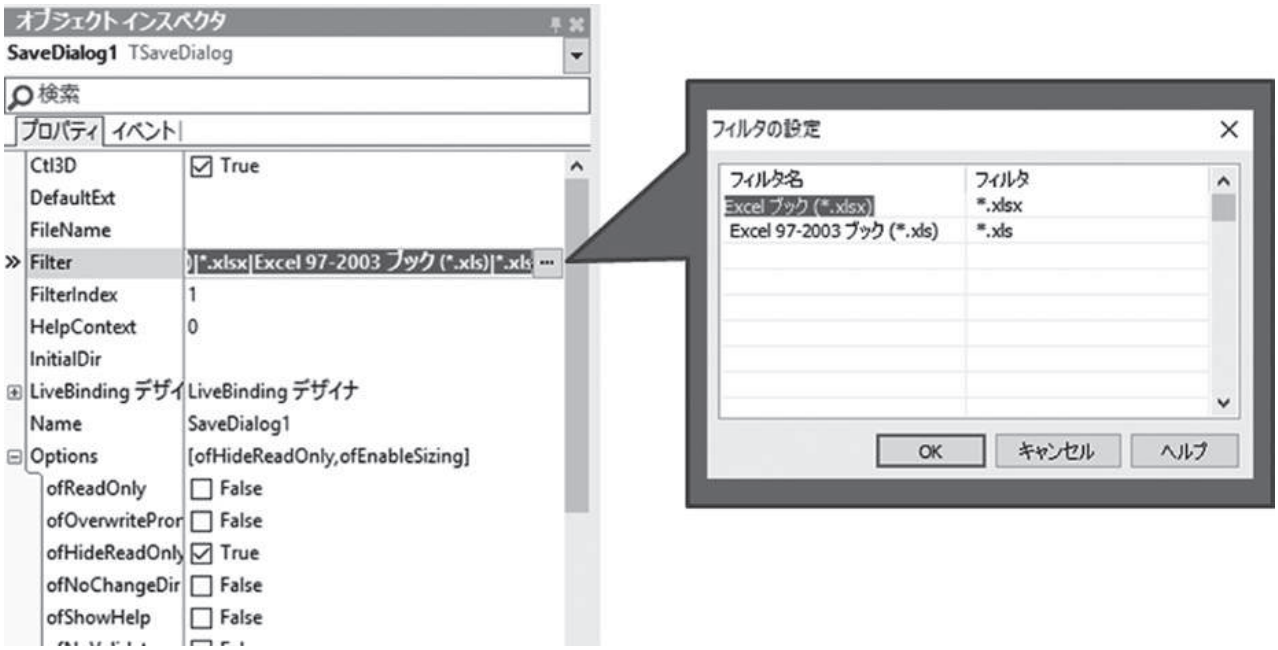
Windowsでは、Excel上でコピー&ペーストを繰り返し実施しているとクリップボードのエラーが発生するという事象がある。

発生する条件は不定だが、特に

ソース2

```
        // 次データへ
        SQLQuery1.Next;
        // 行数をインクリメント
        Inc(iRow);
        // 営業所が変更されたタイミングまたは最終行で小計を出力
        if (sFL1 <> SQLQuery1.FieldByName('URIFL1').AsString) or (SQLQuery1.Eof) then
        begin
          // 小計を出力
          ovWorkSheet.Cells[iRow, 20].Value := sFL1 + '  小計:';
          ovWorkSheet.Cells[iRow, 24].Value := cShokei;
          // 営業所を保持
          sFL1 := SQLQuery1.FieldByName('URIFL1').AsString;
          // 小計をクリア
          cShokei := 0;
          // 行数をインクリメント
          Inc(iRow);
        end;
      end;

      // Excel保存 ...⑥
      if SaveDialog1.Execute then
      begin
        if (LowerCase(ExtractFileExt(SaveDialog1.FileName)) = '.xls') then
        begin
          ovWorkBook.SaveAs(SaveDialog1.FileName, 56);
        end
        else
        begin
          ovWorkBook.SaveAs(SaveDialog1.FileName);
        end;
        ovWorkBook.Saved := True;
      end;

      // Excel表示 ...⑦
      ovExcel.Visible := True;
    finally
      // オブジェクト開放 ...⑧
      ovWorkSheet := Unassigned;
      ovWorkBook := Unassigned;
      ovExcel := Unassigned;
    end;
  finally
    SQLQuery1.Close;
  end;
end;
```

図2　TSaveDialog設定



Windows10 では動作が安定しないことが多い。現時点(2018 年 8 月現在)では、マイクロソフトより解決方法は明示されておらず、Windows Update による修正も実施されていない。

Delphi/400 からクリップボードを操作する際もこの影響を受ける可能性があるため、本稿ではクリップボードの利用、解説は割愛する。

後者の OLE バリアント配列は、2 次元のバリアント配列を生成し、配列に順番に値をセットすることで、複数列・行の情報を一括で Excel に出力する手法である。

この手法であれば、クリップボードの課題点の影響を受けずにパフォーマンス向上を実現することができるため、本稿では OLE バリアント配列を利用した Excel 出力の方法を解説する。

4.OLEバリアント配列を利用した実装

本章では、実際に OLE バリアント配列を利用したプログラムの実装例として、2 章のソースとの相違点を中心に解説する。(【ソース 3】、【ソース 4】)

①2 次元配列用 OLE バリアント変数を定義

変数 ovArray を OleVariant 型で宣言する。

②明細転送用の配列を準備

VarArrayCreate 関数で、変数 ovArray に明細転送用の配列を設定する。1 番目の引数は配列の要素(行および列のインデックス)を指定する。2 番目の引数は配列の要素型(varVariant)を指定する。ここでは Excel テンプレート 1 ページ分(行:24、列:25)で定義している。

③配列へ格納

ここでは Excel のセルに直接値をセットするのではなく、②で準備した配列に格納する。変数 ovArray に行および列のインデックスを指定(Excel のセル位置に該当する箇所)し、IBM i のデータベースより取得した値を順番に配列へ格納する。

④配列より Excel に転送

ovWorkSheet の Range プロパティで配列の行・列の数に合わせて Excel のセル範囲を指定し、変数 ovArray を代入することで、③で配列に格納した値を一括で出力することができる。これにより、複数行・列の出力を 1 回の通信で完了させることが可能である。

通常の静的配列では Range で指定した範囲に配列を代入することができない(型違いでコンパイルエラーとなる)。静的配列でも 1 行単位であれば出力可能だが、その場合は Excel との通信回数が多くなってしまう。そのため、配列は必ず OleVariant 型で定義する必要がある。

⑤配列を解放

VarClear で生成した OLE バリアント配列(ovArray)を解放する。

これで、OLE バリアント配列を利用した Excel 出力処理が完成した。出力結果としては【図 4】と同じになる。

本章の①〜⑤で解説した内容が、2 章のプログラムと異なる点である。別途、改ページが必要になる場合はテンプレートのシートを 1 ページ分コピーして最終行からペーストを行うか、もしくはあらかじめ 2 ページ目以降をテンプレート内に作成しておくことで対応が可能である。

また、配列に格納するデータ量に応じてメモリを消費するので、大量に格納してメモリ不足に陥らないためにも、一定のタイミングで出力するよう注意したい。

続いて、本章で作成したプログラムがどの程度パフォーマンスを向上させられたかを検証するため、実際に Excel 出力にかかる時間を計測、比較している。

検証用にデータを 2000 件準備し、それに伴いロジックを一部変更した(【ソース 3】の②で定義している配列の要素を行:2000 に変更)。

そして、2 章と本章のプログラム共に Excel オブジェクト生成からオブジェクト解放まで(＊ソース内コメント参照)にかかる時間を計測した結果が【図 5】である。

「セル単位で転送」ボタンの右側に、第 2 章のセル単位に出力した場合の計測値、「配列で一括転送」ボタンの右側には、本章の OLE バリアント配列を利用して出力した場合の計測値を表示している。

＜実行結果＞
「セル単位で転送」= 7.866 秒
「配列利用」= 0.964 秒

結果の差からわかるとおり、Excel との通信回数を減らすことで処理時間に明確な効果が出ている。

2 章のセル単位に出力する方法では、単純計算で明細 6 項目 × 2000 行で 12,000 回も Excel と通信を行う。これに対して、本章で解説した OLE バリアント配列を利用したプログラムならば通信回数が 1 回で済むため、パフォーマンスの向上にかなり貢献していることがわかる。

1 回の通信にかかる時間は微々たるものだが、扱うデータの件数や項目数が増えれば増えるほど、この差は顕著になってくるので、本章のテクニックが有効となる。

5.最後に

本稿では、OLE を利用した基本的な Excel 出力の方法と、Excel 出力のパフォーマンスを低下させる要因として、Excel 操作に伴って発生する通信があることを解説した。

OLE バリアント配列を利用する場合としない場合の計測値を比較すれば、Excel との通信回数の低減がパフォーマンス向上に役立つことがご理解いただけたと思う。これは、Excel を操作するプログラム全般で有効なテクニックである。今後、帳票機能を OLE で実装する際は、本稿で解説したパフォーマンス向上テクニックを役立てていただきたく思う。

M

図3　拡張子のエラー

図4　出力結果

売上一覧表（2018年）

名称	住所1	住所2	電話番号	FAX番号	金額
株式会社足立商店	東京都足立区	1－1－2	XXX-XXXX-XXX	XXX-XXXX-XXX	1,000
株式会社足立興業	東京都足立区	1－2－3	XXX-XXXX-XXX	XXX-XXXX-XXX	2,000
株式会社荒川商店	東京都荒川区	2－2－3	XXX-XXXX-XXX	XXX-XXXX-XXX	3,000
株式会社荒川興業	東京都荒川区	1－2－1	XXX-XXXX-XXX	XXX-XXXX-XXX	5,000
株式会社板橋商店	東京都板橋区	5－2－3	XXX-XXXX-XXX	XXX-XXXX-XXX	10,000
				東京営業所 小計：	21,000
株式会社池田商店	大阪府池田市	1－2－3	XXX-XXXX-XXX	XXX-XXXX-XXX	1,000
株式会社泉大津商店	大阪府泉大津市	12－2－3	XXX-XXXX-XXX	XXX-XXXX-XXX	2,000
株式会社泉佐野商店	大阪府泉佐野市	13－2－3	XXX-XXXX-XXX	XXX-XXXX-XXX	3,000
株式会社岸和田商店	大阪府岸和田市	14－2－1	XXX-XXXX-XXX	XXX-XXXX-XXX	4,000
				大阪営業所 小計：	10,000
八幡商店株式会社	京都府八幡市	1－5	XXX-XXXX-XXX	XXX-XXXX-XXX	3,600
株式会社亀岡	京都府亀岡市	5－2－3	XXX-XXXX-XXX	XXX-XXXX-XXX	5,500
				京都営業所 小計：	9,100
株式会社広島商店	広島県広島市	1－1－1	XXX-XXXX-XXX	XXX-XXXX-XXX	1,000
株式会社福山商店	広島県福山市	1－2－2	XXX-XXXX-XXX	XXX-XXXX-XXX	3,000
株式会社尾道商店	広島県尾道市	3－1	XXX-XXXX-XXX	XXX-XXXX-XXX	2,500
				広島営業所 小計：	6,500
株式会社津軽商店	青森県津軽市	1－2－2	XXX-XXXX-XXX	XXX-XXXX-XXX	1,500
株式会社北津軽商店	青森県きた津軽市	1－3－2	XXX-XXXX-XXX	XXX-XXXX-XXX	3,300
				青森営業所 小計：	4,800

ソース3

```
procedure TForm1.Button2Click(Sender: TObject);
var
  ～途中省略～(ソース1と同様)
  ovArray: OleVariant;  ...①
begin

  // 出力データ取得
  SQLQuery1.Close;
  SQLQuery1.SQL.Text := 'SELECT * FROM TRURI ORDER BY URIFL1';
  SQLQuery1.Open;
  try
    // Excelオブジェクト生成
    ovExcel := CreateOleObject('Excel.Application');
    try
      // Excel非表示
      ovExcel.Visible := False;

      // 明細転送用の配列を準備  ...②
      ovArray := VarArrayCreate([0, 23, 0, 24], varVariant);

      // フォーマット読み込み
      sFileName := IncludeTrailingPathDelimiter(ExtractFileDir(Application.ExeName)) + 'Format.xlsx';
      ovWorkBook := ovExcel.Workbooks.Open(sFileName);
      ovWorkSheet := ovWorkBook.WorkSheets[1];

      // 出力準備
      iRow := 0;
      cShokei := 0;
      sFL1 := SQLQuery1.FieldByName('URIFL1').AsString;  // 小計出力用営業所

      // データ出力
      while not SQLQuery1.Eof do
      begin
        // 一旦配列に格納  ...③
        ovArray[iRow,  0] := SQLQuery1.FieldByName('URIFL2').AsString;   // 名称
        ovArray[iRow,  7] := SQLQuery1.FieldByName('URIFL3').AsString;   // 住所１
        ovArray[iRow, 12] := SQLQuery1.FieldByName('URIFL4').AsString;   // 住所２
        ovArray[iRow, 17] := SQLQuery1.FieldByName('URIFL5').AsString;   // 電話番号
        ovArray[iRow, 20] := SQLQuery1.FieldByName('URIFL6').AsString;   // FAX番号
        ovArray[iRow, 23] := SQLQuery1.FieldByName('URIFL7').AsCurrency; // 金額

        // 小計を計算
        cShokei := cShokei + SQLQuery1.FieldByName('URIFL7').AsCurrency;
```

ソース4

```
        // 次データへ
        SQLQuery1.Next;
        // 行数をインクリメント
        Inc(iRow);
        // 営業所が変更されたタイミングまたは最終行で小計を出力
        if (sFL1 <> SQLQuery1.FieldByName('URIFL1').AsString) or (SQLQuery1.Eof) then
        begin
          // 小計を配列に格納
          ovArray[iRow, 19] := sFL1 + '  小計：';
          ovArray[iRow, 23] := cShokei;

          // 営業所を保持
          sFL1 := SQLQuery1.FieldByName('URIFL1').AsString;

          // 小計をクリア
          cShokei := 0;
          // 行数をインクリメント
          Inc(iRow);
        end;
      end;

      // 配列よりExcelに転送 …④
      ovWorkSheet.Range[ovWorkSheet.Cells[3, 1], ovWorkSheet.Cells[26, 25]].Value :=
        ovArray;

      // Excel保存
      ～途中省略～(ソース2と同様)

      // Excel表示
      ovExcel.Visible := True;
    finally
      // 配列を開放 …⑤
      VarClear(ovArray);

      // オブジェクト開放
      ovWorkSheet := Unassigned;
      ovWorkBook := Unassigned;
      ovExcel := Unassigned;
    end;
  finally
    SQLQuery1.Close;
  end;
end;
```

図5　測定結果の比較

佐田 雄一

Delphi/400 中級

株式会社ミガロ.

システム事業部 システム1課

[Delphi/400] FireDAC 実践プログラミングテクニック

略歴
1985 年 12 月 6 日生まれ
2009 年 3 月 甲南大学 経営学部卒業
2009 年 4 月 株式会社ミガロ. 入社
2009 年 4 月 システム事業部配属

現在の仕事内容：
Delphi/400 を利用したシステム開発や保守作業を担当。Delphi および Delphi/400 のスペシャリストを目指して精進している。

1.はじめに

Delphi/400 10 Seattle のリリース以降、新規開発や他の接続方式からの移行において最新のデータベースエンジンである FireDAC が採用されることが多くなった。2016 年版『ミガロ .テクニカルレポート』の SE 論文でも、「新データベースエンジン FireDAC を使ってみよう！」と題して FireDAC の基本的な使用方法を紹介している。

本稿では、FireDAC をさらに使いこなすための実践的なテクニックを検証し、紹介していく。データベースエンジンの操作には大きく分けてデータ取得とデータ更新があり、各章に分けて技術トピックをまとめている。

なお、本稿では Delphi/400 の最新バージョンである Delphi/400 10.2 Tokyo の環境を使用している。

2.FireDACのデータ取得に関するテクニック

2-1.データベースエンジン変更時のマッピングルール

Delphi/400 と IBM i との間でデータをやり取りする場合、値の受け渡しのために、データベースエンジンではコンポーネントでデータ型を自動的に設定する。

このデータ型のルールはデータベースエンジンによって異なり、BDE、dbExpress、FireDAC では一致しないデータ型もある。たとえば整数や小数 4 桁以内の実数は BCD 型、小数 5 桁以上の実数は FMTBCD 型として扱われる。【図 1】【図 2】

FireDAC で新規開発する場合はそのままで問題ないが、従来の BDE や dbExpress から FireDAC に移行するときは、データ型が異なる部分を変更しておく必要がある。たとえばデータの処理時に FireDAC が受け取るデータ型が、既存のプログラムで設定されているデータ型と差異がある場合、そのまま処理すると、エラーが発生してしまう【図 3】。

また、FireDAC で新規開発する場合でも、BCD 型のフィールドは内部的に数値を Currency として保持するため、「17.0 桁」のような小数 4 桁以内かつ Currency 型で扱えない巨大な桁数をセットしようとするとエラーになってしまう。【図 4】

これらの現象に対してマッピング（変換）のルールを設定することで、データベースエンジン間での違いを吸収し、差異があるデータ型を従来と同様のデータ型として扱うことが可能となる。以下に設定手順を紹介する。

まず、TFDConnection コンポーネントをダブルクリックすると、FireDAC 接続エディタが表示される。ここで設定した内容は、紐付く各 TFDQuery や TFDTable にも一括で適用される。個別の TFDQuery や TFDTable コンポー

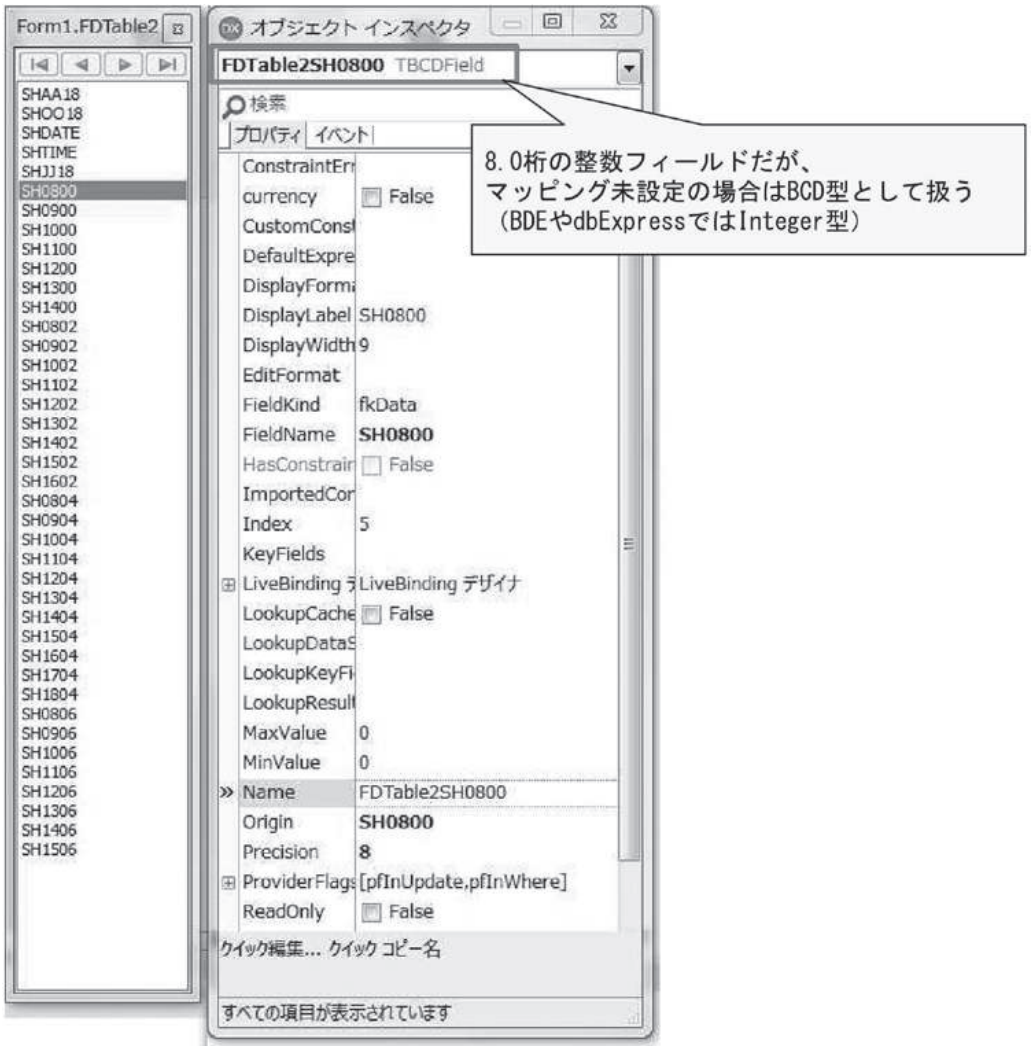
図1　接続方式とDelphi/400のデータ型

データタイプ	型	BDE	dbExpress	FireDAC
A	半角のみの文字型	TStringField	TStringField	TStringField
O	全半角混合の文字型	TStringField	TStringField	TStringField
J	全角のみの文字型	TStringField	TStringField	TStringField
L	日付型	TDateField	TDateField	TDateField
T	時刻型	TTimeField	TTimeField	TTimeField
Z	タイムスタンプ型	TDateTimeField	TSQLTimeStampField	TSQLTimeStampField ★
PまたはS	1～9桁の整数型	TIntegerField	TIntegerField	TBCDField ★▼
	10～18桁の整数型	TFloatField	TFloatField	TBCDField ★▼
	有効桁数が18以内 かつ 小数4桁以内の実数型	TFloatField	TFloatField	TBCDField ★▼
	上記以外の数値型	TFloatField	TFloatField	TFMTBCDField ★▼
B	2進数型	TIntegerField	使用不可	使用不可
H	16進数型	TStringField	使用不可	使用不可

★＝BDEから移行時にマッピングが必要

▼＝dbExpressから移行時にマッピングが必要

図2　フィールド型の違い



ネント単位でも同様の接続エディタを持っているため、特定の1つのコンポーネントのみ例外的に特殊処理が必要な場合は、個別に設定を行うことも可能である。

次にオプションタブを選択し、「継承したルールを無視」チェックボックスをオンにすると、データマッピングルールの明細が入力可能になる。エラーメッセージの内容が、たとえば先の【図3】のように「FDTable1: フィールド '○○○' の型が一致しません。Integer が必要ですが実際は BCD です .」と表示された場合、BCD 型で受け取る数値を Integer 型として認識させればエラーとならずに読み書きが行える。また、IBM i と通信を行う際に考慮が必要なフィールドの型は、【図1】のように数値以外にも存在するため、マッピングルール例としては【図5】のような設定も有効である。

なお、マッピングの設定を行っていないと、数値を BCD 型（TBCDField）として IBM i と値の受け渡しを行うことになるが、Delphi/400 側では TBCDField を内部的に Currency に一度変換するため、【図6】【図7】のように従来とおりのロジックでフィールド値の読み書きを行うことができる。

2-2.ソート順の指定方法

IBM i からデータを複数レコード取得する際、その並び順は大きく分けて「① EBCDIC 順」「② ASCII 順」「③到着順」の3種類のルールが存在する。EBCDIC 順は IBM i と同じ「A ～ Z → 0 ～ 9」の並び順、ASCII 順は Windows と同様の「0 ～ 9 → A ～ Z」の並び順、そして到着順は対象ファイルのレコード登録順（物理レコード順）となっている。【図8】

以下に、【図9】のようなファイルが存在した場合に、FireDAC においてそれぞれの並び順でデータを取得する方法を紹介する。

① EBCDIC 順

EBCDIC 順でデータの取得を行う場合、FireDAC では TFDQuery を使用する。TFDQuery の SQL 文内で ORDER BY 句を使ってフィールドの並び順を指定すると、取得されるデータは ORDER BY で指定されたフィールドで EBCDIC 順に並べることができる。【ソース1】【図10】。

フィールドに降順を指定する際は、IBM i の STRSQL コマンド実行時と同様に ORDER BY 句の中で降順指定フィールドの後ろに「DESC」と記載する。ちなみに、BDE や dbExpress では TClientDataSet と紐付けを行い、TDataSetProvider の設定値によって EBCDIC 順になるよう指定する方法がある。

② ASCII 順

ASCII 順でデータのソートを行う場合、Delphi/400 のクライアント側で Index を設定する方法が最もシンプルである。TFDTable を使用して接続する場合、TFDTable 自身または紐付けた TClientDataSet に、並べたい順にセミコロン区切りで IndexFieldNames プロパティを設定する。フィールドに降順を指定したい場合は、TFDTable の IndexFieldNames プロパティに「(フィールド ID):D」と指定することで、そのフィールドだけ降順にすることができる。【ソース2】【図11】

なお、従来の BDE や dbExpress で降順を指定する際に必要であった、TClientDataSet 側の IndexDefs および IndexName プロパティで降順フィールドを別途指定する方法は FireDAC でも使用可能である。

また、TFDQuery を使用している場合、TClientDataSet に紐付けた上、先述の IndexDefs および IndexName プロパティを指定する。降順がない場合は IndexFieldNames プロパティでもよい。この Index 指定は ORDER BY 句よりも優先されるため、ASCII 順の並びでデータが表示される。

③到着順

では、特に並び順を指定せずに TFDTable や TFDQuery をオープンするとどうなるのか。答えは到着順になる場合と、EBCDIC 順になる場合の両方がある。並び順を指定していない場合は、並び順が保証されておらず、どちらの並び順になるかは、IBM i の STRSQL コマンドで SQL を実行すると確認が可能である（STRSQL の結果と同じ並び順になる）。

意図的に到着順でレコードを表示させたい場合は、TFDQuery を使用し、SQL 文内で RRN（物理ファイルが内部保持している相対レコード番号）に対して ORDER BY 句を掛けることで、対象ファイルのレコードを到着順（＝レコードが追加された順）に並べることが可能となる。【ソース3】【図12】。

2-3.フェッチによるパフォーマンス効果

TFDConnection コンポーネントの FetchOptions プロパティによって、データをクライアント側へ転送する際の設定を変更できる。たとえば数十万件といった大量データをオープンしようとすると、それだけで処理に時間がかかることが多い。しかし FireDAC の場合は、初期設定で 50 件ずつレコードがフェッチされるようになっており、データを 50 件ずつ取得・表示することによって、オープン命令からすぐに 50 件のレコードが表示できる。【図13】

この際、明細表示後にカレントレコードが最終レコード（50 件目）まで到達した状態で次のレコードを取得しようとすると、次の 50 件を取得する。この件数は、FetchOptions プロパティ内の RowsetSize サブプロパティで変更ができる。この機能は、従来から使われる TClientDataSet の PacketRecords プロパティと同じような使い方ができる。

逆にデータの件数があまり多くなく、かつデータを一括で全件表示させる必要がある場合は、初期設定で 50 件ずつになっているレコードのフェッチの設定を無制限になるように設定できる。設定は FetchOptions プロパティ内の Mode サブプロパティを初期設定の「fmOnDemand」から「fmAll」に変更する。【図14】

なお、TClientDataSet と紐付けている場合は RowsetSize の値よりも TClientDataSet の PacketRecords の値が優先される。この構成では実際にデータを持つコンポーネントが TClientDataSet となるからである。

図3　マッピングエラー①

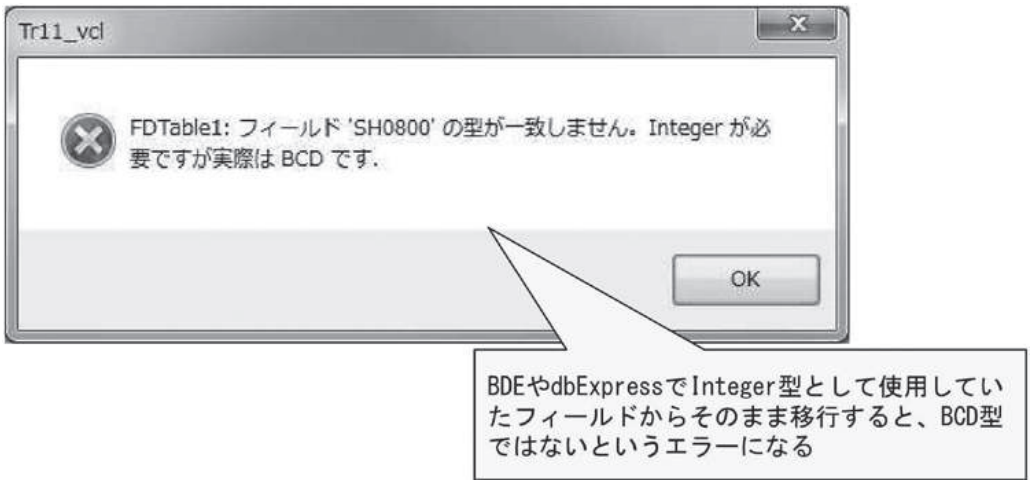
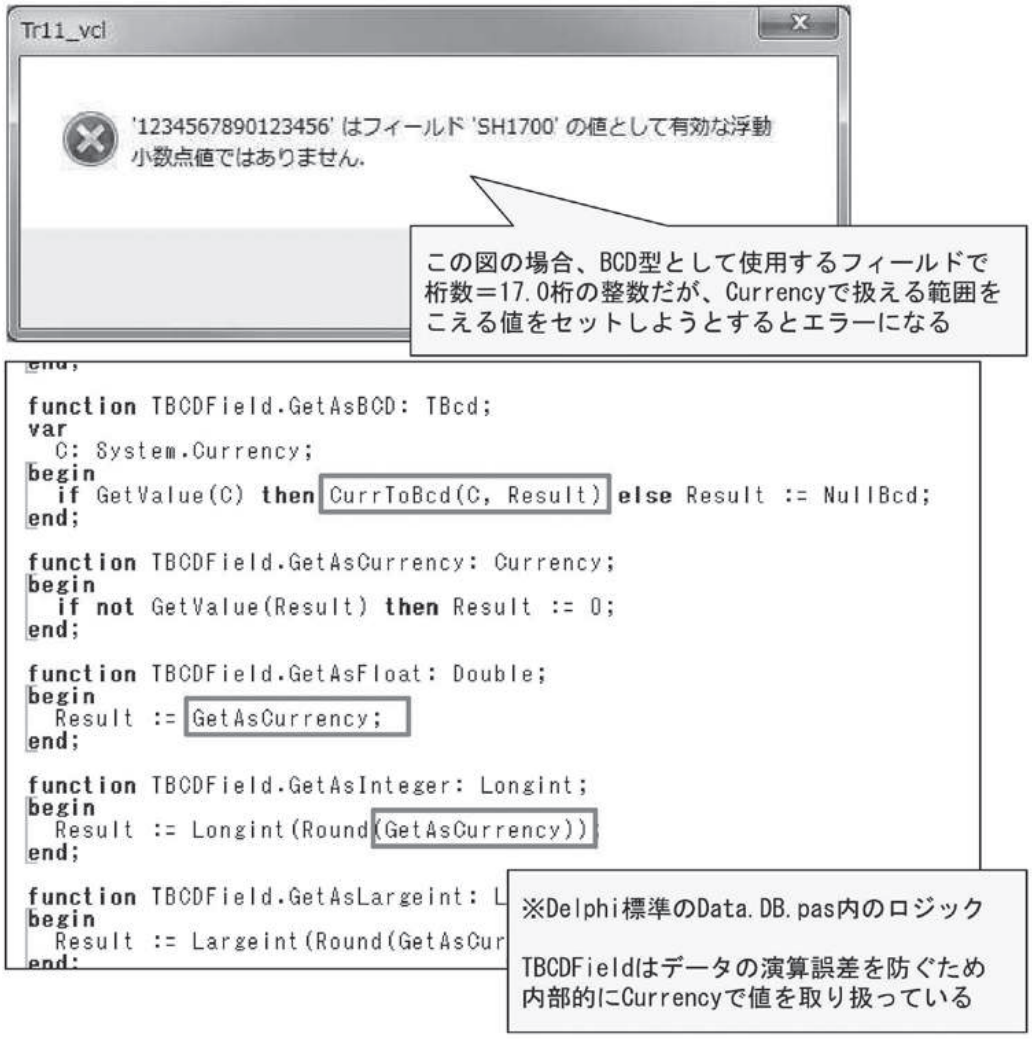


図4　マッピングエラー②



3.FireDACのデータ更新に関するテクニック

3-1.双方向更新機能の活用

データベースエンジンには読み込んだレコードを直接更新できる「双方向データセット形式」と、読み込み専用で更新はできないがパフォーマンスが高い「単方向データセット形式」がある。

従来のBDE接続ではTTableが双方向データセット形式で、dbExpressでは単方向データセット形式になっている。

FireDACではBDEと同様に双方向データセット形式となっているため、適切な設定を行えば読み込んだレコードへ直接更新することができる。以下に手順を記載する。

まず、先述のマッピングの設定と同様、TFDConnection側でUpdateOptionsプロパティを【図15】のように設定する。

この設定が行われていないとTFDTableやTFDQueryをオープンした際に読み取り専用として開くため、データの更新処理は正しく行えず、エラー等が発生する可能性がある。

次に、読み込んだデータを編集して更新する場合の設定を行う。それぞれ目的にあわせてUpdateOptionsプロパティのUpdateModeサブプロパティを【図16】のように設定する。

TFDTableにおける各処理では内部的にSQL文を生成・実行する仕組みを持っており、レコード参照・追加・変更・削除時に、実際にはそれぞれSELECT・INSERT・UPDATE・DELETEのSQLがTFDTableによって発行されている。

この中でUPDATEまたはDELETEを行う場合には、UpdateModeサブプロパティの設定値によって、どのレコードを更新対象とするかが決定する。【図16】のとおり、UpdateModeで設定した値によって、どのフィールドを更新条件とするかが決定してWHERE句を生成している。【図17】

続いて、更新条件とするフィールド側にも設定を行う。オブジェクトインスペクタ上にフィールドがある場合は【図18】のように指定する。また、設計画面上ではフィールドが存在せず、ソース内で設定を行う場合は【ソース4】のように指定する。なお、UpdateModeサブプロパティが「upWhereAll」の場合は全フィールドを更新条件とするため指定不要であるが、IBM iのデータにPC側で扱えないコード等が含まれていると条件が一致しない可能性があるので注意が必要である。

UpdateModeサブプロパティが「upWhereAll」以外の場合、この指定を行っておかないと、Postの際にどのフィールドを基準に更新（WHERE句を作成）するかプログラムが把握できないため、注意が必要である。「upWhereKeyOnly」の場合は更新条件となるフィールドがないため、プログラムが自動的にupWhereAllとして更新を行う。「upWhereChanged」の場合は値を変更したフィールドだけが更新条件になるため、同じ値の他レコードもすべて更新対象になってしまう。

3-2.FireDACのトランザクション制御

FireDACにおけるトランザクション制御は、従来のBDEと近い実装方法で行うことができる。設定方法を説明する。

まずTFDConnectionをダブルクリックしてFireDAC接続エディタを開き、ODBCAdvancedパラメータを【図19】のように設定する。このパラメータにはライブラリリストなどの他の指定もできるが、複数設定の指定を行う場合は、セミコロンで区切る必要がある。

各機能においてトランザクション処理を行う方法は、従来のBDEと同様である。【ソース5】

トランザクション制御において従来のBDEと異なり注意が必要となるポイントを少し補足する。それはジョブ終了時の制御である。

BDEでは、StartTransactionからCommitまでの間にTDatabaseとの接続を明示的に終了した場合、ジョブが終了するためトランザクションはロールバックされる。しかしFireDACでは、接続終了時のデフォルトの動作設定の初期値がCommitになっているため、トランザクションがコミットされる。これをBDEと同様に接続終了時のデフォルトの動作設定をRollbackにするために、【図20】のようにTxOptionsプロパティの設定を行う。なお、Delphi/400アプリケーションを強制終了した場合など、ジョブが異常終了する場合はBDEと同様ロールバックされる。

4.ファイルのメンバー制御

最後にデータの取得・更新に共通したテクニックとしてメンバーの制御方法を紹介する。IBM iでは1つのデータベース・ファイルに対して複数のメンバーを持つことができる。たとえば、ワークファイルを使用する際にユーザーごとにそれぞれメンバーを指定したい場合、従来のBDEではTTableのTableNameプロパティでメンバーを直接指定できるが、FireDACではSQLで動作するため直接指定できない。【図21】

このとき任意のメンバーに対して接続する方法は大きく2種類存在する。

1つ目は、ネイティブ接続と併用可能な場合にTAS400コンポーネントからOVRDBFコマンドを発行し、ファイル名を一時変更する方法である。【ソース6】のように、ロジック内でTFDTableをオープンする前にネイティブ接続側でOVRDBFコマンドを直接発行すれば、DLTOVRコマンドで解除するか、別のOVRDBFコマンドで上書きするまで、指定したメンバーを参照・更新できるようになる【図22】。必要に応じて、TFDTableをクローズした後に、DLTOVRコマンドで一時変更を削除すれば解除できる。

2つ目は、FireDAC単体でメンバーを制御する方法である。これはWebアプリケーションやDataSnapアプリケーションといったネイティブ接続とジョブが別れてしまう構成で有効な実装方法となる。

まずSQLで実行できる「CREATE ALIAS」「DROP ALIAS」でメンバーに対して別名を作成し、TableNameプロパティでそのエイリアスを指定する。【ソース7】【図23】

この場合はOVRDBFとは異なり、CREATE ALIASは別名で実体を作成するため、既に存在する名前でエイリア

図5　マッピング設定

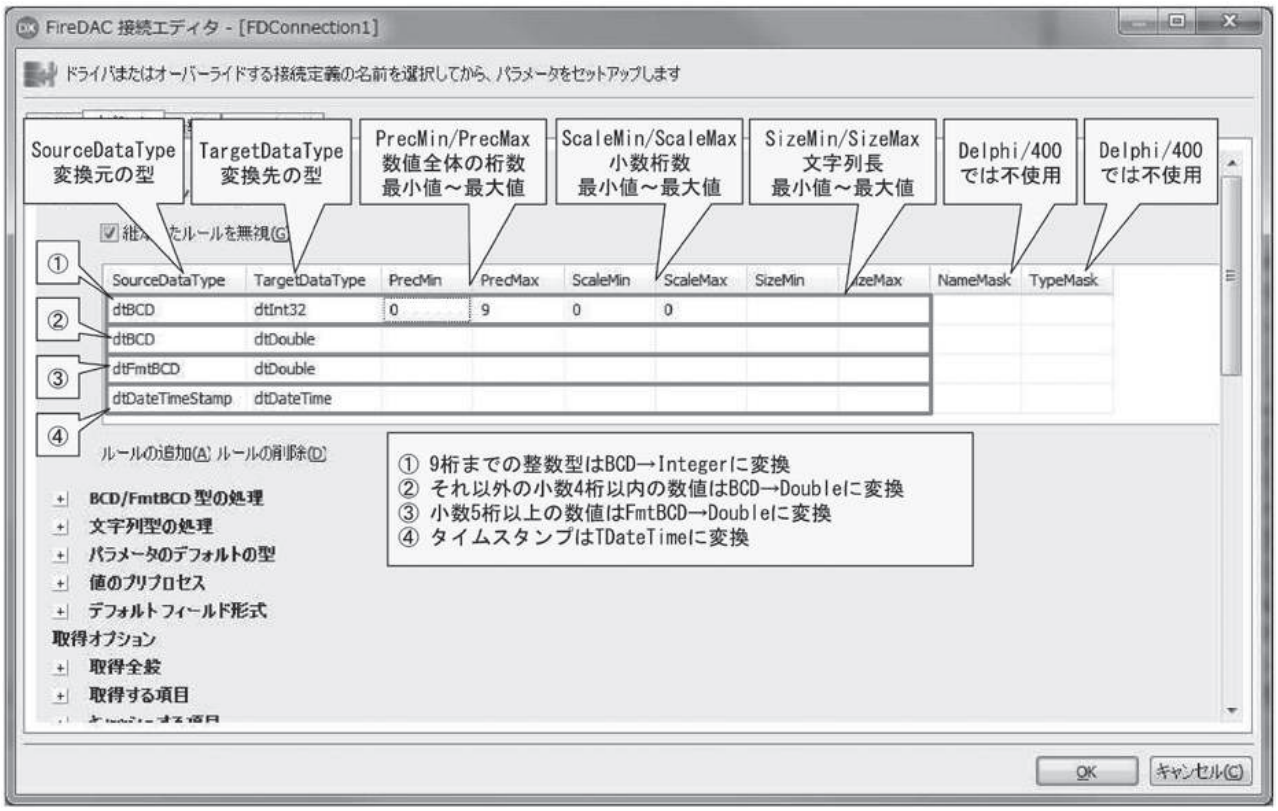
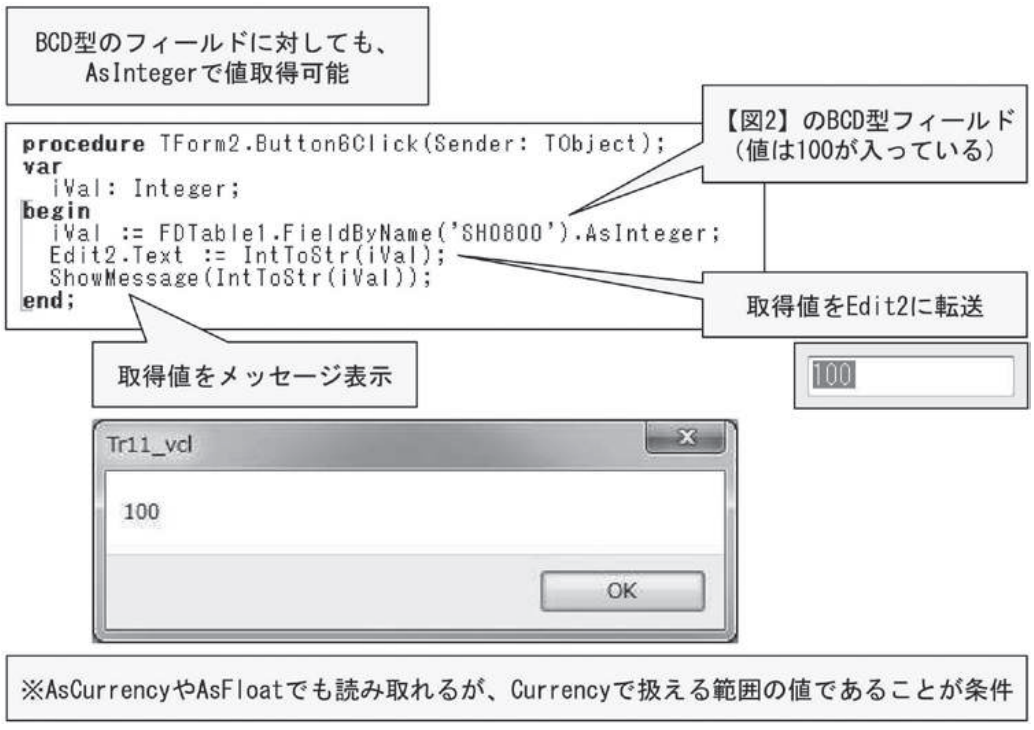


図6　FieldByNameのロジック



スを作成しようとするとエラーになってしまう。名前の一部にジョブ番号を使用するなど、エイリアス名が重複しないよう留意する必要がある。

以上がFireDACでのメンバーの制御テクニックである。

5.まとめ

本稿では、Delphi/400の開発者目線で、検証済みの技術ポイントを中心に、実践的なFireDACのテクニックを紹介した。

最近はWindows10対応などを中心にDelphi/400のバージョンアップを行う開発も多く、それに伴ってBDEやdbExpressからFireDACへ変更する機会が増えている。その中で、参照・更新や、メンバー処理にまつわるテクニックを把握しておけば、これまでのプログラムをスムーズに移行できる。

これまでのシステム開発経験を本稿で共有することで、Delphi/400の新機能であるFireDACを広く活用していただければ幸いである。

M

図7　FieldByNameのロジック

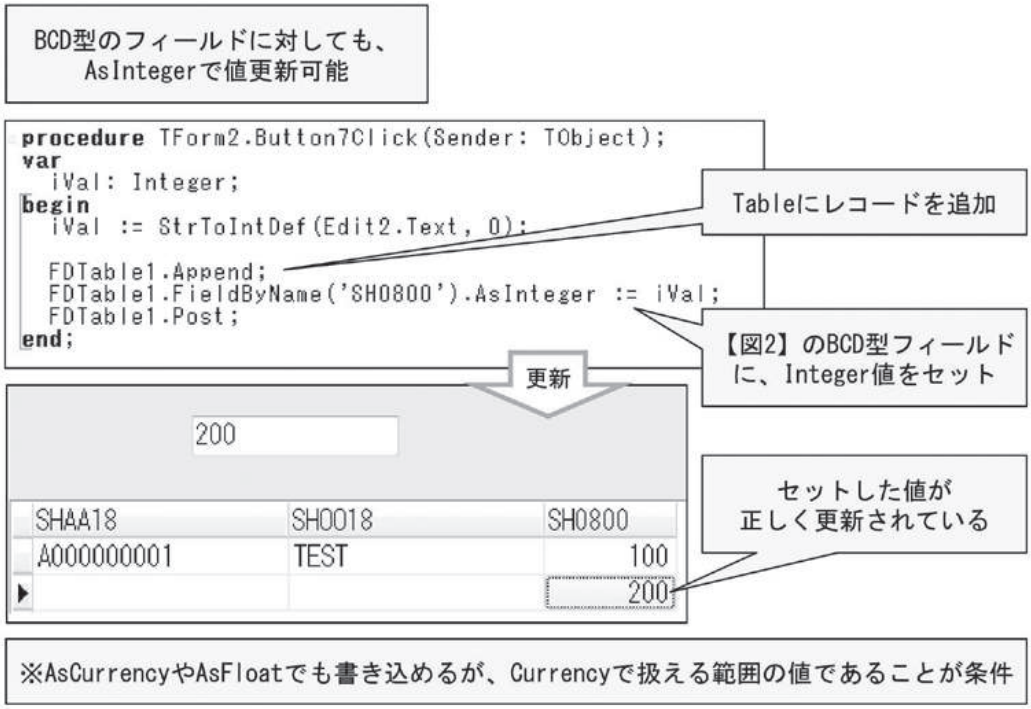


図8　データの並び順について

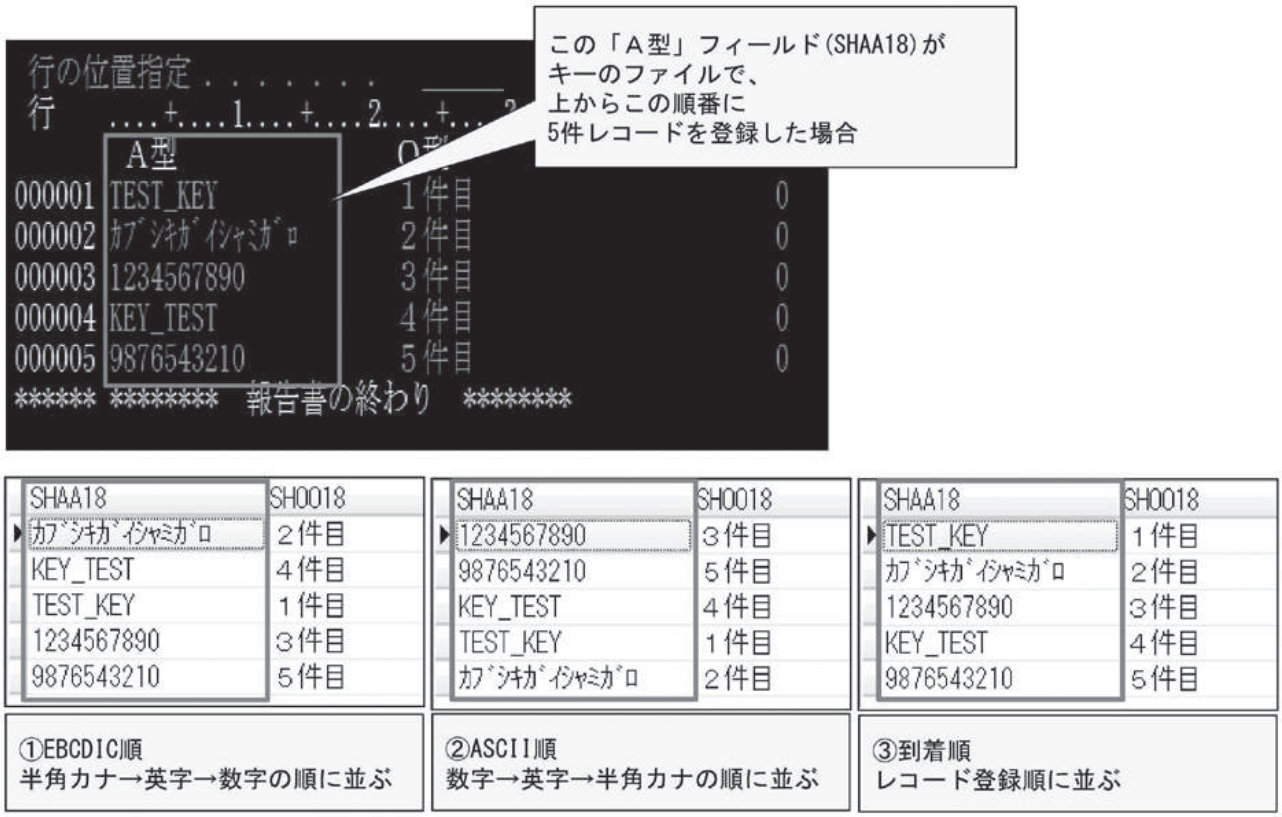


図9　データの並び順について

上からこの順番にレコードを登録したファイル
（登録後に一部レコードを削除し、現在は16レコード）

```
行       ....+....1....+....2....+....3....+..
         SHAA10      SHAA11       SHAA12
000001 MIGARO_001  SANKEIBLD_1  REPORT_00001
000002 MIGARO_001  AAAAAAAAA_1  REPORT_00002
000003 MIGARO_001  TECHNICAL_2  REPORT_00005
000004 MIGARO_001  123456789_2  REPORT_00006
000005 356356_002  TECHNICAL_1  REPORT_00009
000006 MIGARO_002  OSAKACITY_1  REPORT_00010
000007 MIGARO_002  XXXXXXXXX_2  REPORT_00013
000008 MIGARO_002  TECHNICAL_2  REPORT_00014
000009 356356_001  TECHNICAL_1  REPORT_00017
000010 356356_001  333333333_1  REPORT_00018
000011 356356_001  NANIWA_KU_2  REPORT_00021
000012 356356_001  ﾃｸﾆｶﾙﾚﾎﾞｰ_2  REPORT_00022
000013 MIGARO_002  SYSTEM1KA_1  REPORT_00025
000014 356356_002  MIGARON___1  REPORT_00026
000015 356356_002  MINATOMAC_2  REPORT_00029
000016 356356_002  XYZXYZXYZ_2  REPORT_00030
****** ********  報告書の終わり  ********
```

フィールドについて
（SHAA10～12はいずれもAタイプ）

①SHAA10＝キー、昇順
　値は英字始まりのレコードと
　数字始まりのレコードが混在

②SHAA11＝非キー
　レコードによって様々な値を登録

③SHAA12＝非キー
　"REPORT_"＋レコード登録時の連番を登録
　（※削除レコードの連番は欠番）

ソース1　EBCDIC順参照のソース記述例

```
procedure TForm1.Button4EClick(Sender: TObject);
begin
  FDQuery1.Close;
  FDQuery1.SQL.Clear;
  FDQuery1.SQL.Add(' SELECT * FROM TR11F05 ');
  FDQuery1.SQL.Add(' ORDER BY SHAA11, SHAA12 ');
  FDQuery1.Open;
end;
```

TFDQueryのSQLに、STRSQLで指定する際と
同じようにORDER BYを掛ける

図10　EBCDIC順参照の取得結果

SHAA10	SHAA11	SHAA12
▶356356_001	ﾃｸﾆｶﾙﾚﾎﾞｰ_2	REPORT_00022
MIGARO_001	AAAAAAAAA_1	REPORT_00002
356356_002	MIGARON___1	REPORT_00026
356356_002	MINATOMAC_2	REPORT_00029
356356_001	NANIWA_KU_2	REPORT_00021
MIGARO_002	OSAKACITY_1	REPORT_00010
MIGARO_001	SANKEIBLD_1	REPORT_00001
MIGARO_002	SYSTEM1KA_1	REPORT_00025
356356_002	TECHNICAL_1	REPORT_00009
356356_001	TECHNICAL_1	REPORT_00017
MIGARO_001	TECHNICAL_2	REPORT_00005
MIGARO_002	TECHNICAL_2	REPORT_00014
MIGARO_002	XXXXXXXXX_2	REPORT_00013
356356_002	XYZXYZXYZ_2	REPORT_00030
MIGARO_001	123456789_2	REPORT_00006
356356_001	333333333_1	REPORT_00018

SQLで指定した通りに、
EBCDIC順（半角カナ→英字→数字の順）で
SHAA11の昇順＞SHAA12の昇順に並ぶ

ソース2　ASCII順参照のソース記述例

```
procedure TForm1.Button4WClick(Sender: TObject);
begin
  FDTable1.Close;
  FDTable1.TableName := 'TR11F05';
  FDTable1.IndexFieldNames := 'SHAA11;SHAA12:D';
  FDTable1.Open;
end;
```

TFDTableにIndexFieldNamesを指定
・複数フィールドはセミコロンで区切る
・降順にしたいフィールドは後ろに「:D」を付ける

図11　ASCII順参照の取得結果

SHAA10	SHAA11	SHAA12
▶MIGARO_001	123456789_2	REPORT_00006
356356_001	333333333_1	REPORT_00018
MIGARO_001	AAAAAAAAA_1	REPORT_00002
356356_002	MIGARON___1	REPORT_00026
356356_002	MINATOMAC_2	REPORT_00029
356356_001	NANIWA_KU_2	REPORT_00021
MIGARO_002	OSAKACITY_1	REPORT_00010
MIGARO_001	SANKEIBLD_1	REPORT_00001
MIGARO_002	SYSTEM1KA_1	REPORT_00025
356356_001	TECHNICAL_1	REPORT_00017
356356_002	TECHNICAL_1	REPORT_00009
MIGARO_002	TECHNICAL_2	REPORT_00014
MIGARO_001	TECHNICAL_2	REPORT_00005
MIGARO_002	XXXXXXXXX_2	REPORT_00013
356356_002	XYZXYZXYZ_2	REPORT_00030
356356_001	ﾃｸﾆｶﾙﾚﾎﾟｰ_2	REPORT_00022

IndexFieldNamesで指定した通りに、
ASCII順(数字→英字→半角カナの順)で
SHAA11の昇順＞SHAA12の降順に並ぶ

ソース3　到着順参照のソース記述例

```
procedure TForm1.Button4TClick(Sender: TObject);
begin
  FDQuery1.Close;
  FDQuery1.SQL.Clear;
  FDQuery1.SQL.Add('  SELECT TR11F05.*, RRN(TR11F05) AS RRN1 FROM TR11F05 ');
  FDQuery1.SQL.Add('  ORDER BY RRN1 ');
  FDQuery1.Open;
end;
```

TFDQueryのSQLに、フィールドではなく
RRN(相対レコード番号)に対してORDER BYを掛ける

図12　到着順参照の取得結果

SHAA10	SHAA11	SHAA12	RRN1
▶MIGARO_001	SANKEIBLD_1	REPORT_00001	1
MIGARO_001	AAAAAAAAA_1	REPORT_00002	2
MIGARO_001	TECHNICAL_2	REPORT_00005	5
MIGARO_001	123456789_2	REPORT_00006	6
356356_002	TECHNICAL_1	REPORT_00009	9
MIGARO_002	OSAKACITY_1	REPORT_00010	10
MIGARO_002	XXXXXXXXX_2	REPORT_00013	13
MIGARO_002	TECHNICAL_2	REPORT_00014	14
356356_001	TECHNICAL_1	REPORT_00017	17
356356_001	333333333_1	REPORT_00018	18
356356_001	NANIWA_KU_2	REPORT_00021	21
356356_001	ﾃｸﾆｶﾙﾚﾎﾟｰ_2	REPORT_00022	22
MIGARO_002	SYSTEM1KA_1	REPORT_00025	25
356356_002	MIGARON___1	REPORT_00026	26
356356_002	MINATOMAC_2	REPORT_00029	29
356356_002	XYZXYZXYZ_2	REPORT_00030	30

SQLで指定の通りに、RRNの昇順でソートされるため
実質的に到着順(レコード登録順)にレコードが並ぶ
※ORDER BY句に「DESC」を付ければ、
　到着順の逆順で表示させることも可能

図13　フェッチのレスポンス

図14　フェッチ設定の変更

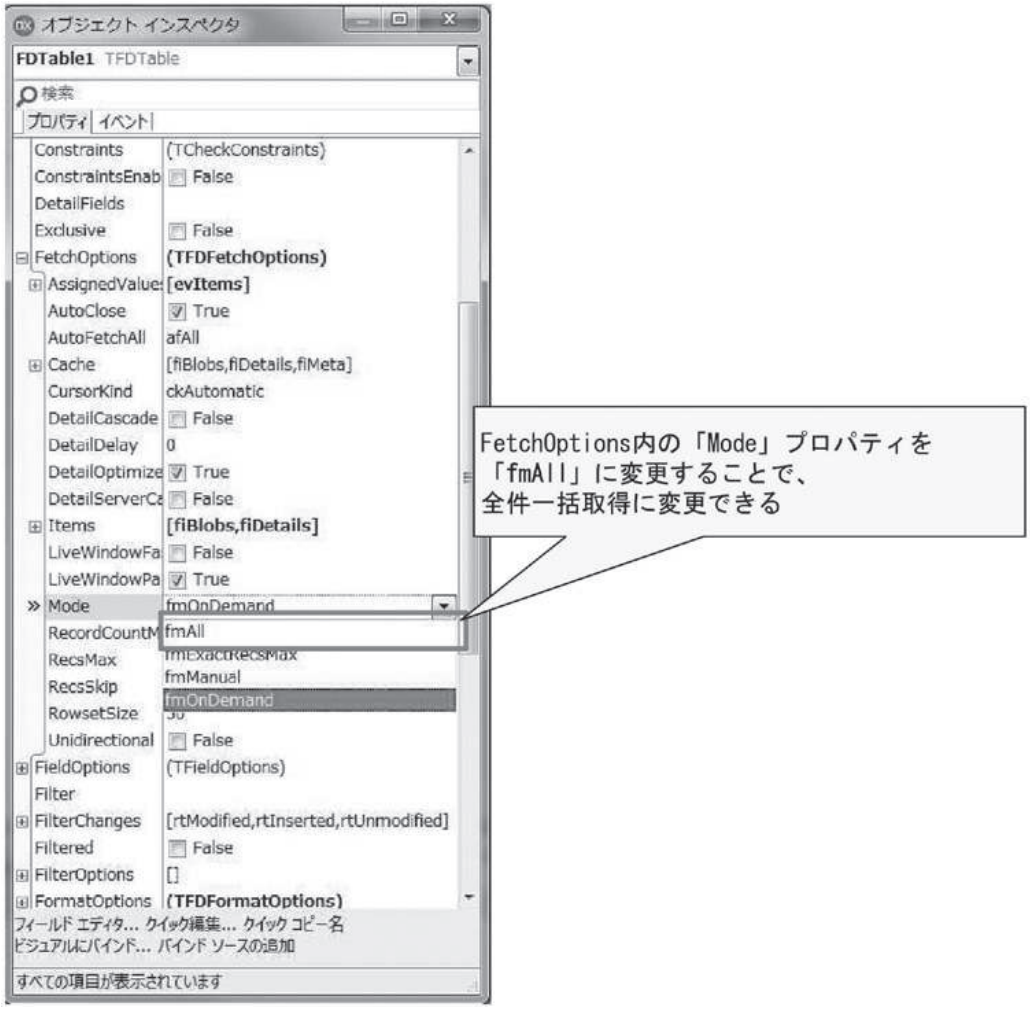


図15　UpdateOptionsの設定

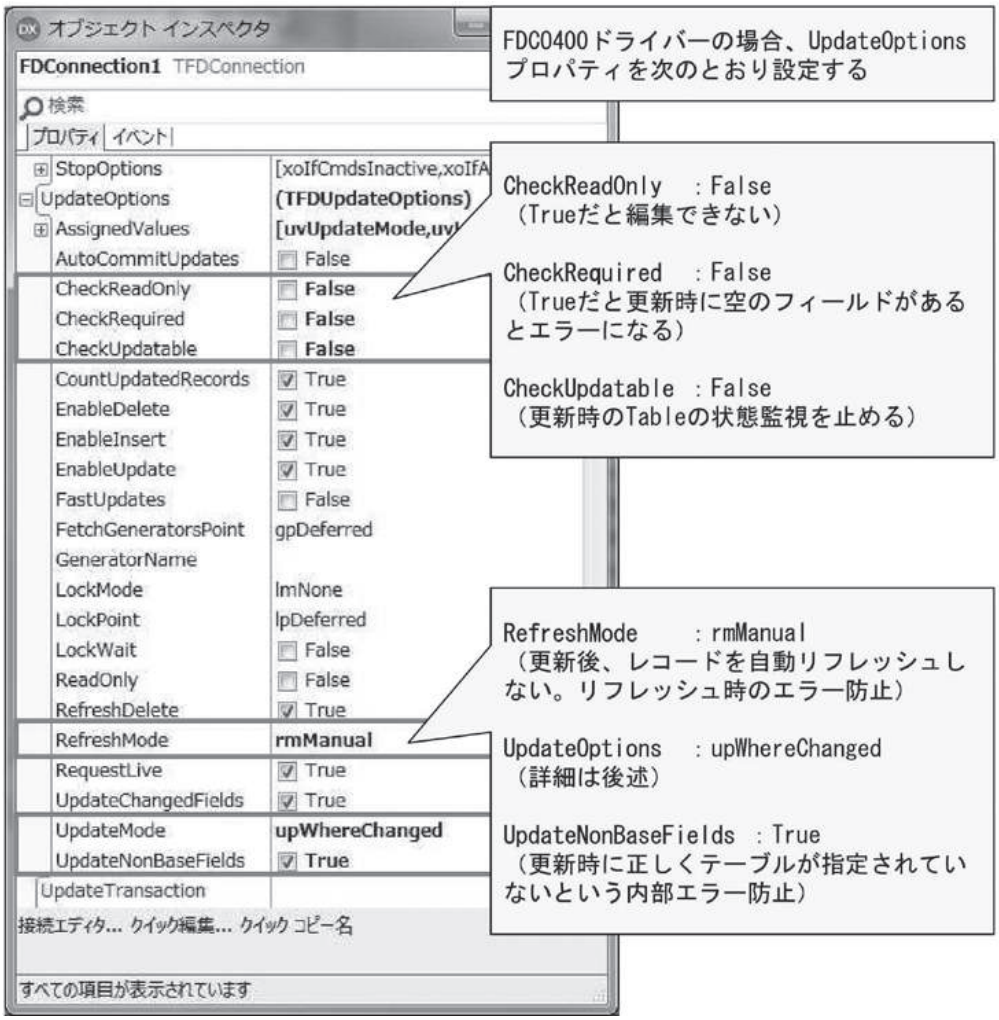


図16　UpdateOptionsの設定

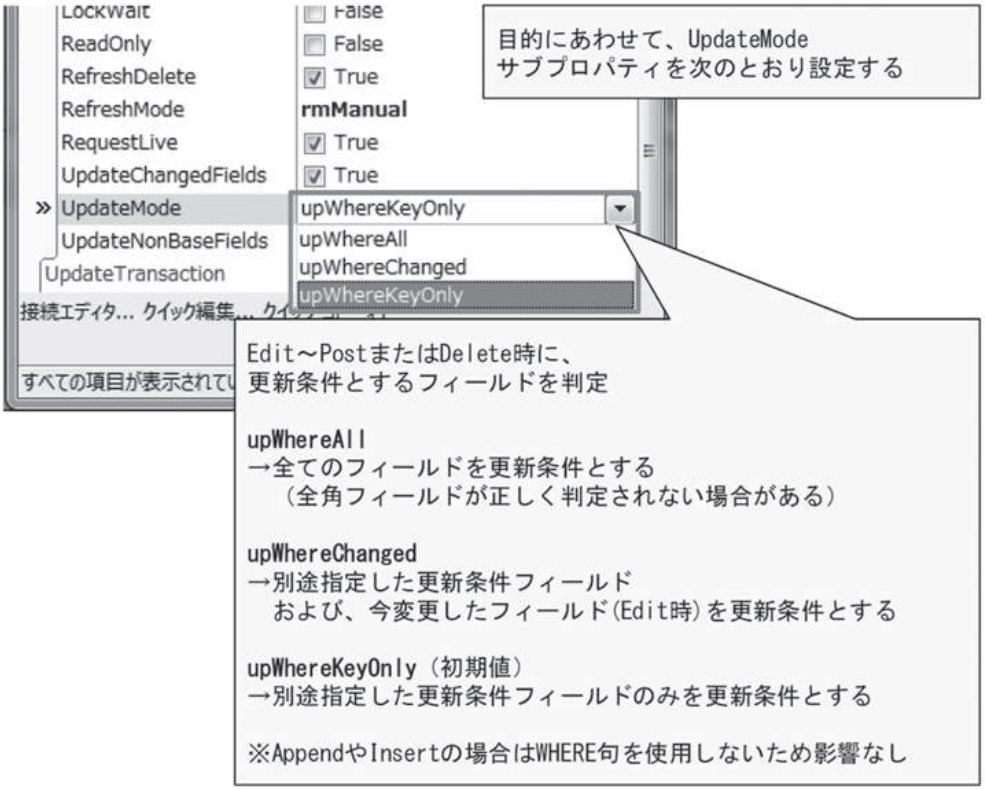


図17　更新条件とWHERE句

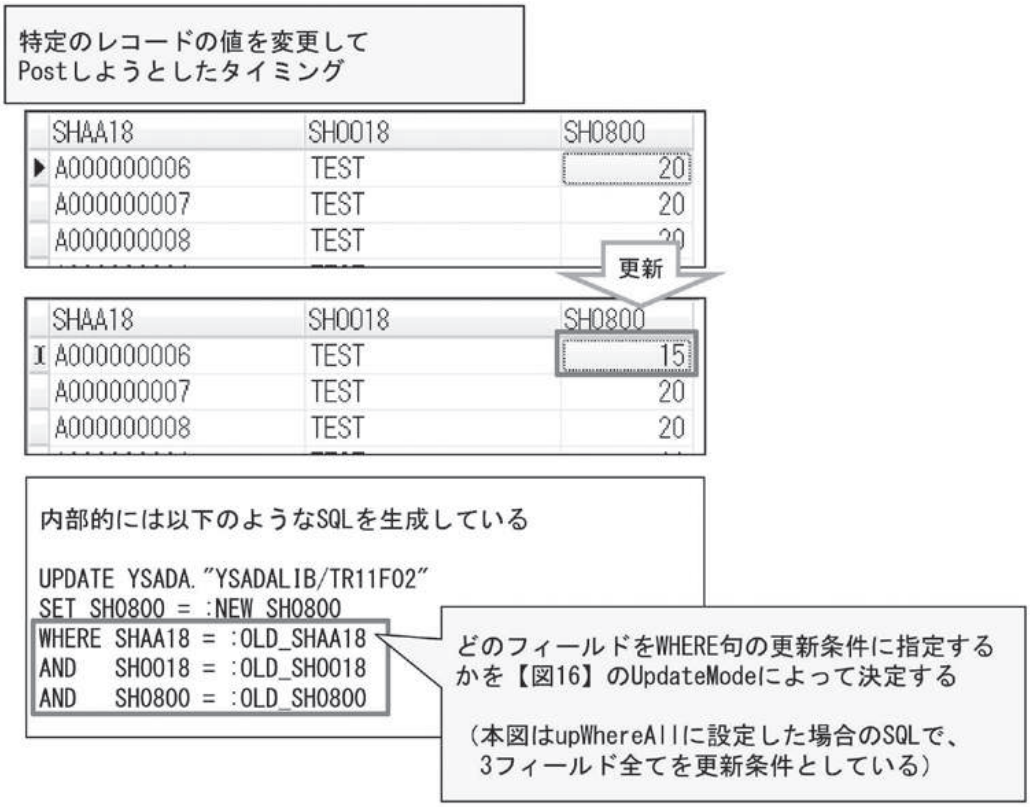
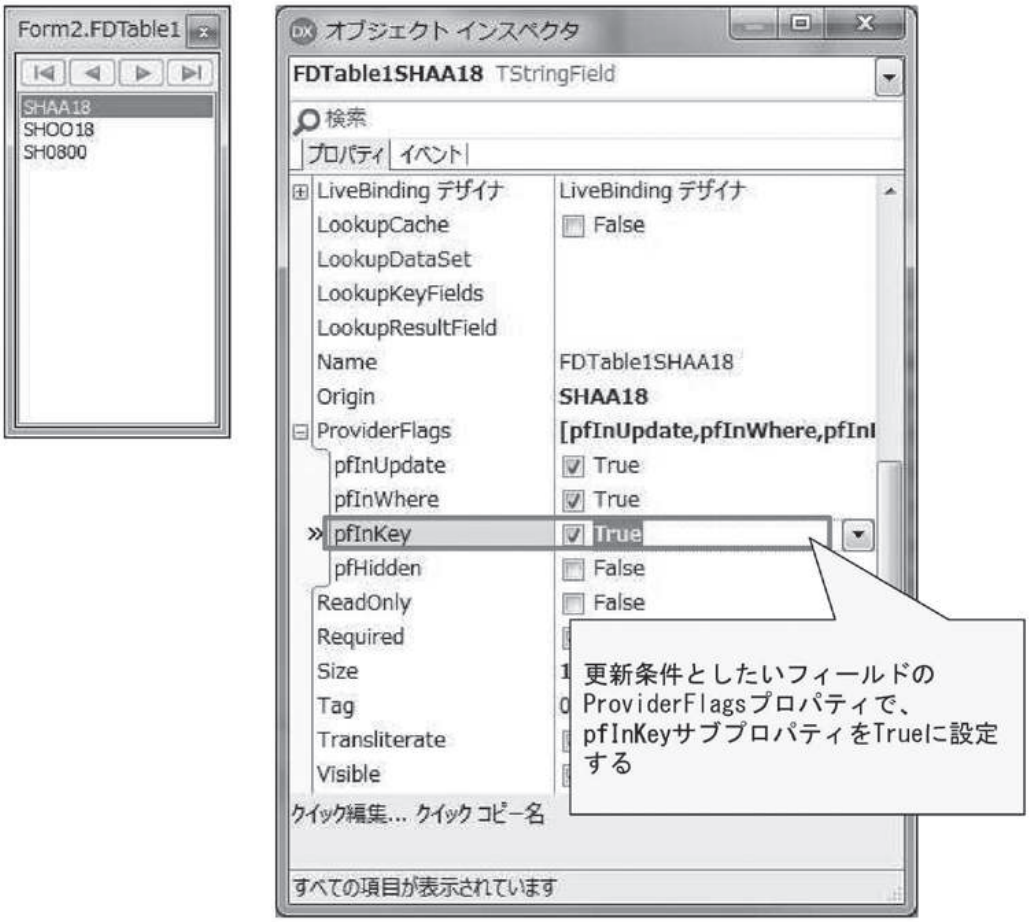


図18　更新条件フィールドの設定

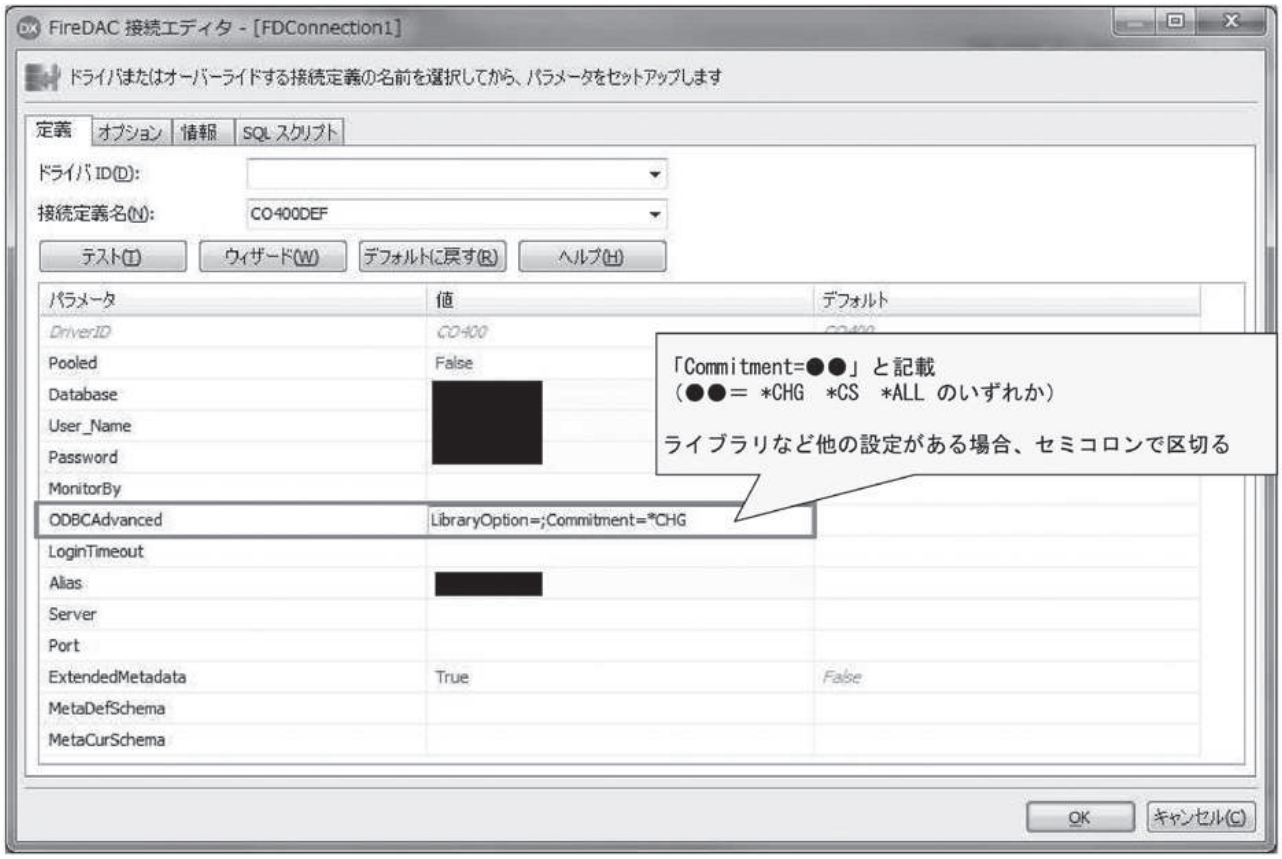


ソース4　更新キーフィールド設定のソース記述例

```
// 画面のデータをワークにセット
with tblUpdate do
begin
  TableName := 'TR11F02';
  Open;
    // 一意になるように更新キーフィールドを指定(複数指定可能)
    FieldByName('SHAA18').ProviderFlags := [pfInKey]; // キーフィールド
    First;

    for i := 1 to stgList.RowCount - 1 do
    begin
      Edit;
      // (各フィールド値セット)
      Post;
      Next;
    end;
  end;
end;
```

フィールドのProviderFlagsプロパティを
直接ソースで指定する

図19　トランザクションに必要な設定



ソース5　トランザクションのソース記述

```
procedure TForm2.SpeedButton4Click(Sender: TObject);
begin
  if (not FDConnection1.InTransaction) then
  begin
    FDConnection1.StartTransaction;
  end;
end;

procedure TForm2.SpeedButton5Click(Sender: TObject);
begin
  if (FDConnection1.InTransaction) then
  begin
    FDConnection1.Commit;
  end;
end;

procedure TForm2.SpeedButton6Click(Sender: TObject);
begin
  if (FDConnection1.InTransaction) then
  begin
    FDConnection1.Rollback;
  end;
end;
```

トランザクションの開始

トランザクションのコミット

トランザクションのロールバック

図20　明示切断時設定の変更点

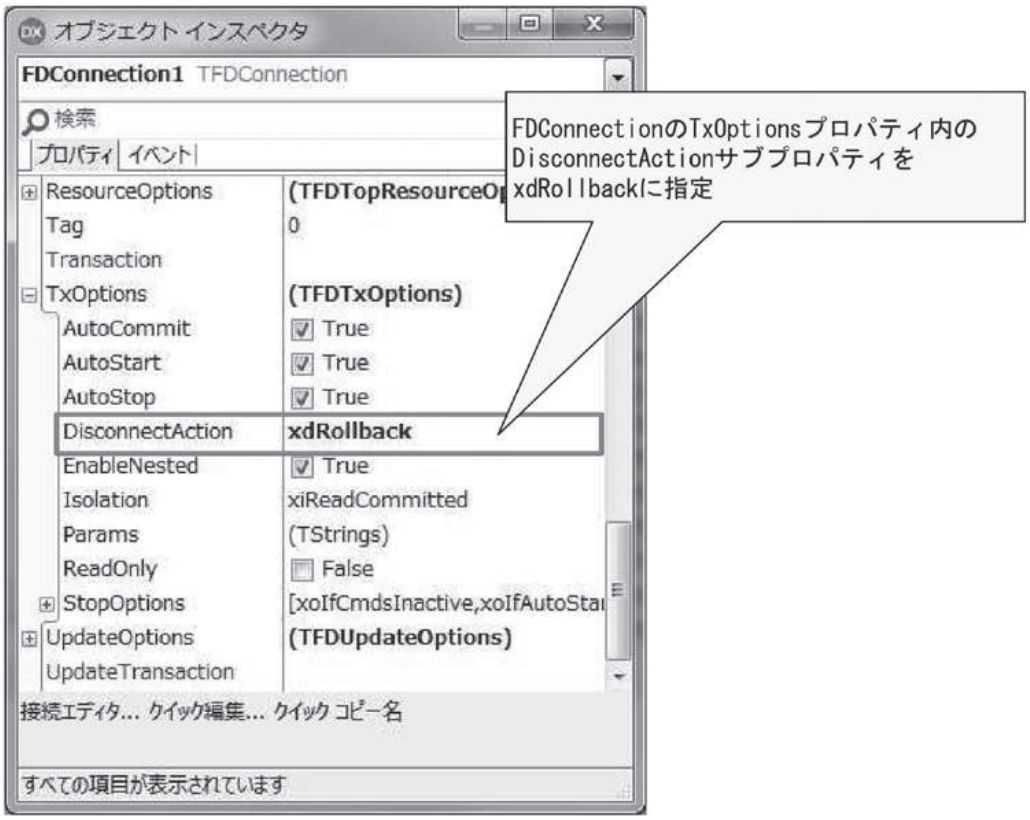


図21　TableNameのメンバー指定

ソース6　OVRDBFによるメンバー参照

```
procedure TForm1.Button4Click(Sender: TObject);
begin
  FDTable1.Close;
  FDTable1.TableName := 'TR11F05';
  AS4001.RemoteCmd('OVRDBF FILE(TR11F05) TOFILE(TR11F05) MBR(X_MBR1) OVRSCOPE(*JOB)');
  FDTable1.Open;
end;
```

RemoteCmdでネイティブ側からOVRDBFコマンドを発行し、メンバー指定を行う

テーブルのClose後、必要に応じてDLTOVRコマンドを発行し、メンバー指定を解除する
AS4001.RemoteCmd('DLTOVR FILE(TR11F05) OVRSCOPE(*JOB)');

図22　OVRDBFのメンバー参照結果

SHAA10	SHAA11	SHAA12
MEMBER	X_MBR1	TEST01
MEMBER	X_MBR1	TEST02
MEMBER	X_MBR1	TEST03
MEMBER	X_MBR1	TEST04

【図9】と同じファイルだが、
指定したメンバーのレコードが表示される

ソース7　ALIASによるメンバー参照

```
// ①テーブルオープンを行う処理
procedure TForm1.Button5Click(Sender: TObject);
begin
  FDTable1.Close;
  FDTable1.TableName := sAliasName;
  FDTable1.Open;  // ここで②が走る
end;

// ②テーブルオープン時イベント
procedure TForm1.FDTable1BeforeOpen(DataSet: TDataSet);
begin
  FDConnection1.ExecSQL('CREATE ALIAS ' + sAliasName + ' FOR TR11F05(X_MBR1)');
end;

// ③画面クローズ時イベント
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  FDTable1.Close; // ここで④が走る
end;

// ④テーブルクローズ時イベント
procedure TForm1.FDTable1AfterClose(DataSet: TDataSet);
begin
  FDConnection1.ExecSQL('DROP ALIAS ' + sAliasName);
end;
```

TableNameにエイリアスの名前を指定
※他のジョブと重複しない名前を
　予め変数にセットしておく
　（「〇〇＋ジョブ番号」形式を推奨）

メンバーを指定してエイリアスを生成

※FormDestroyでは④が走らない場合がある

エイリアスの削除

図23　ALIASのメンバー参照結果

SHAA10	SHAA11	SHAA12
MEMBER	X_MBR1	TEST01
MEMBER	X_MBR1	TEST02
MEMBER	X_MBR1	TEST03
MEMBER	X_MBR1	TEST04

【図22】と同様、指定したメンバーのレコードが表示される

尾崎 浩司

Delphi/400 中級

株式会社ミガロ.

RAD事業部 営業・営業推進課

[Delphi/400] RESTによるWebサービスを活用した機能拡張テクニック

略歴
1973 年 8 月 16 日生まれ
1996 年 3 月 三重大学 工学部卒業
1999 年 10 月 株式会社ミガロ. 入社
1999 年 10 月 システム事業部配属
2013 年 4 月 RAD 事業部配属

現在の仕事内容：
ミガロ. 製品の素晴らしさをアピールするためのセミナーやイベントの企画・運営等を主に担当している。

1.はじめに

近年アプリケーションの開発において、Web サービスの活用が盛んになっている。Web サービスとは、インターネット技術を応用し、他の Web サイト上のソフトウェアを呼び出して利用する仕組みのことである。【図 1】

現在では、大量のデータを蓄積している業者等が、そのデータを Web サービスの形で一般のユーザーやプログラマーに提供する事例が多くなっている。たとえば、ネットショッピングサイト大手のアマゾンには「Product Advertising API」、楽天には「楽天市場商品検索 API」といった Web サービスがあり、これら Web サービスを利用すると、サイト上の商品検索等を自分のプログラムに組み込むことができる。さらに近年では、従来の大量データをもつ業者だけでなく、IBM Watson のような自然言語を理解し、機械学習により人間の意思決定を支援するシステムまでもが、Web サービスとして利用可能になっている。

Web サービスは、インターネット技術を使用するのが特徴だが、その手法にはいくつかあり、代表的なのが SOAP（Simple Object Access Protocol） および REST（REpresentational State Transfer）である。

SOAP は、SOAP メッセージという XML によってメッセージ交換を行う方法で事前にやり取りの定義が必要なため、難易度が高い。最近は、よりシンプルな REST が主流である。本稿では、Delphi/400 を使用した REST による Web サービスの使用方法や機能拡張方法について説明する。なお、本稿のプログラムは Delphi/400 10 Seattle 以降の環境を前提としている。

2.RESTによるWebサービスとは?

REST とは、Web サービスの設計モデルのことで、ネットワーク上のデータ（リソース）を一意な URL で表すものである。サービスの URL に HTTP メソッドでアクセスすることでデータの送受信が行える。パラメータを指定して URL にアクセスすると特定の形式でデータが返ってくるものだ。データ形式には、XML あるいは JSON が利用可能であるが、近年は、よりシンプルな JSON が使われることが多い。

JSON と は、JavaScript Object Notation の略で、軽量のデータ交換フォーマットのことである。key（名前）と value（値）を「:」で対にして記述し、まとまりごとに { } で囲うといった表記法で、[ ] で配列を表現することもできる。たとえば、【図 2】のような JSON は、"result" というキーの配列の第一要素の中にある "score" というキーの値が 80 であると解釈できる。

では、ここで REST + JSON による Web サービスを試してみる。livedoor が提供する「お天気 Web サービス」を使ってみる。

http://weather.livedoor.com/weather_hacks/webservice

図1　Webサービス

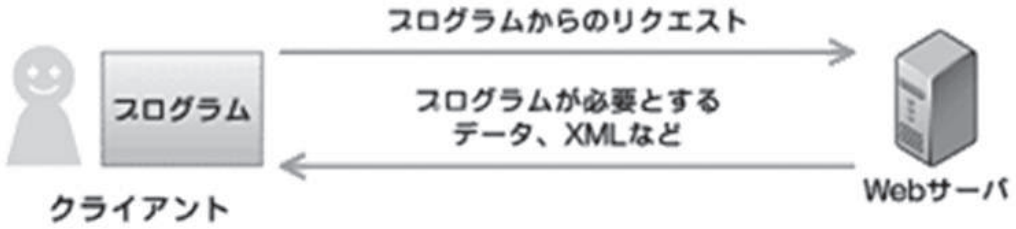
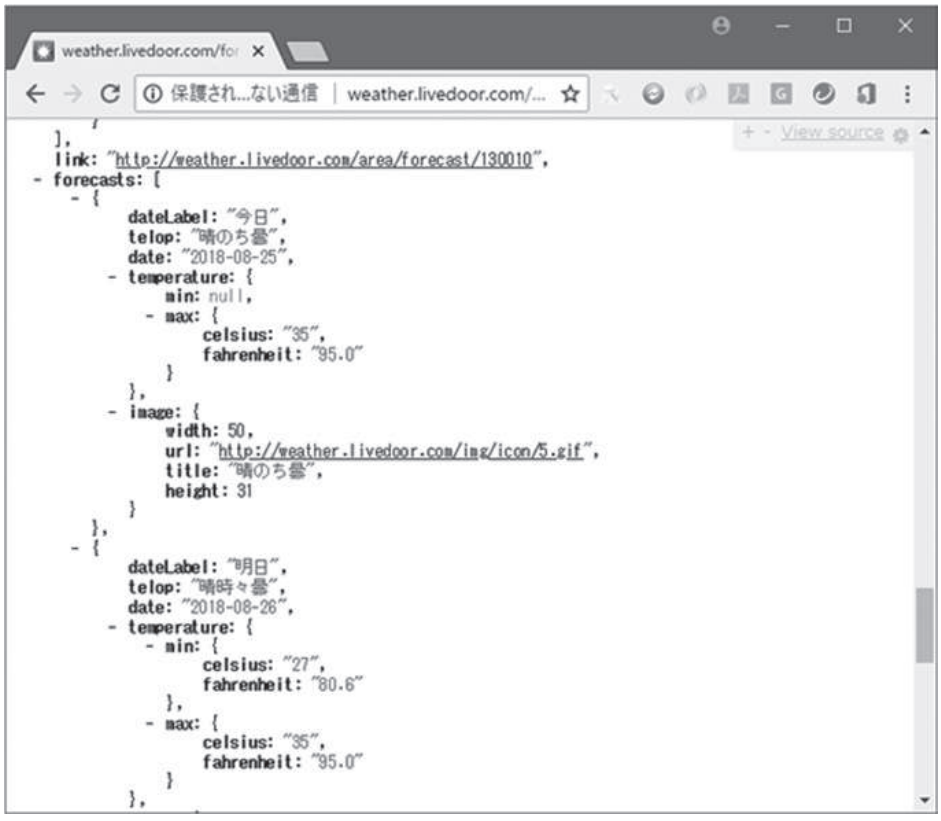


図2　JSONの例

```
{
"result":[

    {
     "subject":"english",
     "score": 80,
     "state":"good"
   },

    {
     "subject":"math",
     "score": 70,
     "state":"normal"
   }

]
}
```

“result”キーの値が配列となっており、配列要素1つ目の中にある、”score”キーの値が80である。

図3　お天気Webサービス実行例



（Google で“お天気 Web サービス”を検索すると上位に表示される「お天気 Web サービス仕様」。）

このサービスは、現在全国 142 カ所の今日・明日・明後日の天気予報・予想気温と都道府県の天気概況情報を提供するものである。

Chrome ブラウザを立ち上げてアドレス欄に

「http://weather.livedoor.com/forecast/webservice/json/v1?city=130010」

と入力してアクセスを行う。すると、【図 3】のような JSON が表示される。

これは、地域 ID=130010（東京都）の天気予報情報にアクセスした結果の JSON である。天気や最高気温などの情報が JSON の中に含まれていることがわかる。このように REST + JSON による Web サービスは、URL にパラメータを付けて呼び出すとレスポンスとして JSON データが返ってくることが確認できる。

3.REST機能を利用する方法

では、この「お天気 Web サービス」を Delphi/400 から使用する方法を検討する。Delphi/400 には、REST による Web サービスを使用するためのコンポーネントが用意されている。それが、「TRESTClient」、「TRESTRequest」そして「TRESTResponse」である。【図 4】

「TRESTClient」は、Web サービスへのリクエストを実行するコンポーネントで、サービスに対する HTTP 接続を管理し、HTTP ヘッダーおよびプロキシ サーバーを処理し、応答データを受け取るものである。「TRESTRequest」は、HTTP リクエストを形成するパラメータや設定をすべて保持する。「TRESTResponse」は、Web サービスからのすべての戻りデータを保持する。実際の設定は次のようになる。【図 5】

ポイントは、RESTClient1 コンポーネントの BaseURL プロパティに WEB サービスの基底 URL を指定すること、RESTRequest1 コンポーネントの Method プロパティに HTTP メソッドの種類を、Resource プロパティに実行パラメータを指定することである。

このプログラムの [検索] ボタンクリック時の処理は、【ソース 1】となる。

1 行目は、画面上で指定した地域 ID を Resource プロパティに記した "{CITY}" にセットする処理である。Params プロパティの AddItem メソッドがリクエストのパラメータを定義するメソッドである。2 行目は、Web サービスへのリクエスト実行になり、レスポンスの JSON 文字列を取得して Memo1 にセットするのが、3 行目である。実際に実行した結果が【図 6】となる。

REST による Web サービスによって JSON データが取得できることを確認したが、実際にはこの JSON データをパース（解釈）し必要な情報を抜き出す必要がある。Delphi/400 にはこの JSON を取り扱うためのユニットが用意されている。それが、”System.JSON”ユニットである。この中に、JSON オブジェクトを実装したクラス TJSONObject や、文字列、数値、オブジェクト、配列、true/false の型を持つすべての JSON クラスの上位クラス TJSONValue が用意されているので、これらを使用することでパースすることが可能である。

たとえば、【図 7】が JSON をパースして特定のキーの値を取得するロジック例である。

お天気情報の中から、今日と明日と明後日の天気を取得して表示する処理を実装してみる。

「お天気 Web サービス」の説明 Web ページを確認すると、レスポンスの仕様が記載されている。[forecasts] プロパティの中にある [date] プロパティが予報日、そして [telop] プロパティが天気である。なお、3 日分のデータは配列として定義されている。この情報をもとに JSON をパースして、天気予報を画面に表示するように改良したのが、【ソース 2】である。完成したプログラムを実行すると、【図 8】のように指定した地域の 3 日分の天気予測を取得することができる。

もう 1 つ Web サービスの例を紹介する。「HeartRails Geo API」（http://geoapi.heartrails.com/）である。これは、郵便番号／住所／緯度経度データ等の地理情報を提供する Web サービスである。この中に、「最寄駅情報取得 API」という機能があり、これは郵便番号を指定すると、その地区の一番近い最寄駅がわかる機能である。これを活用すると、たとえば社内の取引先マスタにある郵便番号を使用して、同じ最寄駅から歩いて訪問できる取引先をピックアップする使い方ができる。

サンプルプログラムを紹介する。REST コンポーネントの設定は、【図 9】、プログラムは【ソース 3】となる。この API は、パラメータ postal に郵便番号をセットし呼び出すと、レスポンスとして、response プロパティの station 配列の中にある prefecture プロパティには都道府県が、line プロパティに路線が、そして name プロパティには最寄駅がセットされる。作成したプログラムを実行して、郵便番号を入力し、[検索] ボタンをクリックすると、最寄駅が表示されることがわかる。【図 10】

この最寄駅情報取得 API も、先ほどの天気情報の Web サービスと全く同じやり方で処理ができることがわかる。このように、REST + JSON の Web サービスは、とても簡単に使用できるので、いろいろなサービスを試してみていただきたい。本稿執筆にあたり、Web サービスを調査したが、「API List 100+」（http://smsurf.app-rox.com/api/） というサイトが役立った。ここにはいろいろな Web サービスが一覧掲載されているので、便利な機能を見つけてみてほしい。【図 11】

4.IBM Watson API 活用方法

「コグニティブ」という言葉を聞いたことがないだろうか。日本語では「認知」のことで、ある事象についてコンピュータが自ら考え、学習し、自らの答えを導き出すシステムのことをいう。身近なところでは、iPhone の Siri やスマートスピーカー等が有名である。従来システムとの本質的な違いは、音声・画像・文章等の非定型データも処理できることである。従来システムがもつ定型データと組み合わせることで、人の作業を補助し、より便利なシステム構築が可能になる。この「コグニティブ」分野で IBM が提供するのが、Watson である。この

図4　お天気情報取得画面

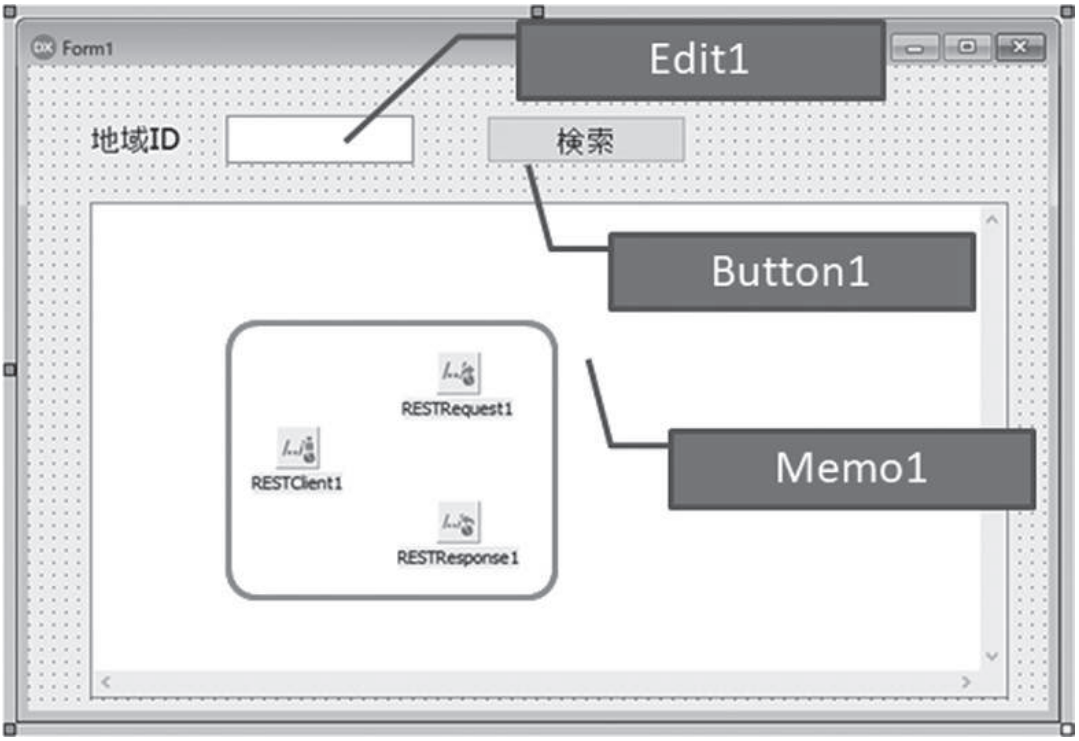
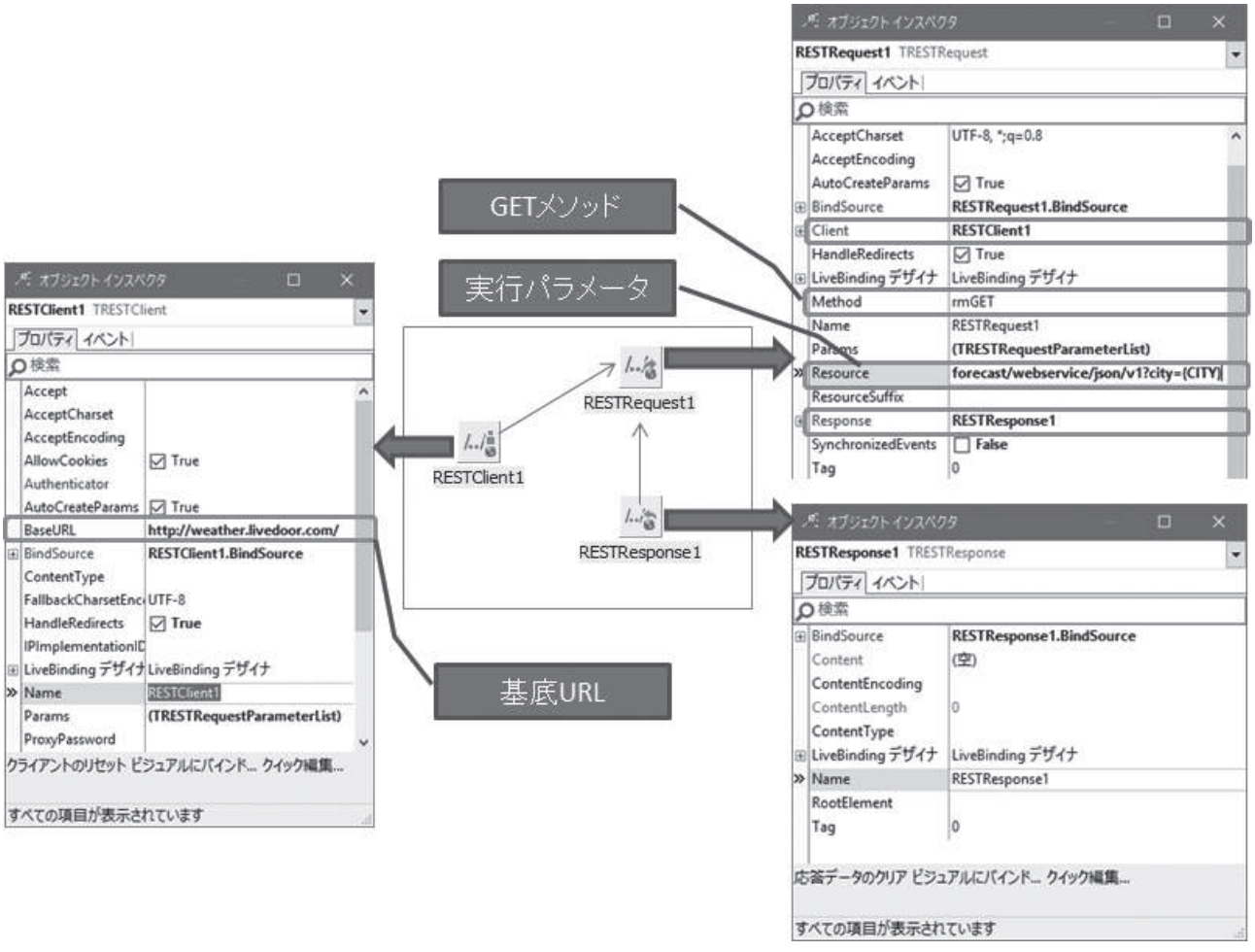


図5　TRESTコンポーネント設定



Watson も Web サービスとして利用することができるので本章で説明する。

Watson は、IBM のクラウドサービス（IBM Cloud）上で API として提供されており、クラウドサービスに登録すれば誰でも使用できる。本格的な利用には有償プランが必要だが、IBM Cloud の各種サービスを無料で使用できる「ライト・アカウント」があるので、こちらを使用するとよい。「ライト・アカウント」には、サービス使用量や機能の制約はあるが、クレジットカード登録不要で簡単に登録できる。

https://www.ibm.com/cloud-computing/jp/ja/bluemix/lite-account/

（Google で“IBM Cloud ライト・アカウント”を検索すると上位に表示される「IBM Cloud ライト・アカウント -Japan」。）

2018 年 8 月時点で確認した Watson の API が【図 12】である。調べたすべてのサービスが「ライト・アカウント」で使用可能だ。

今回は、Language Translator サービスを例に説明する。これは言語変換（翻訳）サービスで、特長としては、一般的な WEB 翻訳サービスと違い、専門用語等を個別に登録することや、機械学習によるカスタム翻訳モデルを作成することが可能で、高精度の翻訳が行えることである。このサービスを活用すると、海外担当者とのやり取り時の自動翻訳や、システムの多言語対応で、DB 上に日本語でしか保持していない情報を翻訳して、画面に出力することができる。

Watson は、サービスごとにインスタンスの作成が必要である。まず IBM Cloud（https://console.bluemix.net/）からサインインを行い、表示されたダッシュボード画面より、[リソースの作成] ボタンをクリックする。Watson をはじめとするサービス選択画面が表示されるので、[Language Translator] を選択する。サービス概要画面が表示され、プランの選択ができるようになるため、「ライト」が選択されていることを確認し、[作成] をクリックすれば、完了である。インスタンスの作成が完了すると、作成したサービスの管理画面が表示される。【図 13】

【図 13】の管理画面の中にある「資格情報」が重要である。2018 年 8 月現在サービスによって「資格情報」には 2 種類あり、ユーザーとパスワードが表示されるサービスと、API Key が表示されるサービスとがある。Language Translator サービスの場合、API Key が表示されるため、この API Key と URL を控えておけばよい。

また、管理画面には「API リファレンス」画面へのリンクがあり、そこにアクセスすれば API の仕様が記載されている。2018 年 8 月現在 Language Translator サービスは、V3 というバージョンになっており、API の仕様は、【図 14】のとおりである。

Delphi/400 から利用するポイントを説明する。今回は、画面上に日本語で入力したテキストを英語に翻訳するアプリを作成する。（英語から日本語への翻訳も可能にする。）

WatsonAPI の場合、資格情報が必要なため、認証が必要である。WatsonAPI では、基本認証を使用することができ、これは「THTTPBasicAuthenticator」コンポーネントが使用できる。

REST コンポーネントの設定は【図 15】のとおりである。

サービスの資格情報がユーザーとパスワードの場合は、そのまま「THTTPBasicAuthenticator」コンポーネントの Username プロパティと Password プロパティにセットすればよく、APIKey の場合は、Username プロパティに "apikey"、Password プロパティに資格情報の APIKey を入力すればよい。

今回のアプリの画面レイアウトは、【図 16】、プログラムは【ソース 4】のようになる。この Web サービスは、POST メソッドとなり、リクエスト本体に JSON 形式でパラメータを渡すところがポイントである。

実際に実行したアプリケーションが、【図 17】である。Watson API を利用するアプリケーションも REST + JSON で簡単に構築できることがわかる。

5.REST機能をもつコンポーネントの作成

今回、REST + JSON を使用した Web サービスの活用方法について具体例を挙げながら説明したが、Web サービスの課題点は、サービス提供者の都合により、サービスの仕様が変更されたり、サービス自体が終了してしまう可能性があることだ。

ある Web サービスを活用したアプリケーションを使用していた場合に、このような事態が発生すると、新しい仕様にあわせてプログラムを変更したり、あるいは代替サービスに置換したりといった作業が必要になる。こういったことを想定した場合、Web サービスの機能を個々のプログラムに都度記述する方法だと、修正ボリュームが多大になることが想像できる。

また、プログラムの中から Web サービスの部分を抜き出して修正しなければいけないため、煩雑な作業になることが予想される。

この問題を解決するには、どうすればよいか？　1 つの方法が各 Web サービスごとにコンポーネント化してしまうことである。そうすれば、Web サービスの仕様変更時にも、コンポーネントソースのみ修正し、各プログラムは、リコンパイルだけすれば済むはずである。

今回は、コンポーネント化の例として、前章で使用した Language Translator サービスのコンポーネント化を検討する。（TComponent を継承した TTranslator コンポーネント（非ビジュアルコンポーネント）を作成する。）

本稿では、コンポーネントそのものの基本的な作成手順については割愛するが、作成手順が分からない場合は、2012 年度版『ミガロ. テクニカルレポート』の SE 論文「カスタマイズコンポーネント入門」を参照していただきたい。

宣言部は、【ソース 5】となる。変換元の言語（SourceLanguage プロパティ）と変換後の言語（TargetLanguage プロパティ）、そして変換対象の文字列（Source プロパティ）を設定した後、Translate メソッドを実行すると翻訳が行われ、その結果は、Destination プロパティにセットされるという仕様を想定している。

ソース1　検索ボタンのOnClickイベント

```
uses REST.Types;

procedure TForm1.Button1Click(Sender: TObject);
begin
  //URLパラメータの指定
  RESTRequest1.Params.AddItem('CITY', Edit1.Text, pkURLSEGMENT);

  //リクエスト実行
  RESTRequest1.Execute;

  //レスポンスJSONを表示
  Memo1.Text := RESTResponse1.JSONText;
end;
```

図6　お天気情報取得実行例

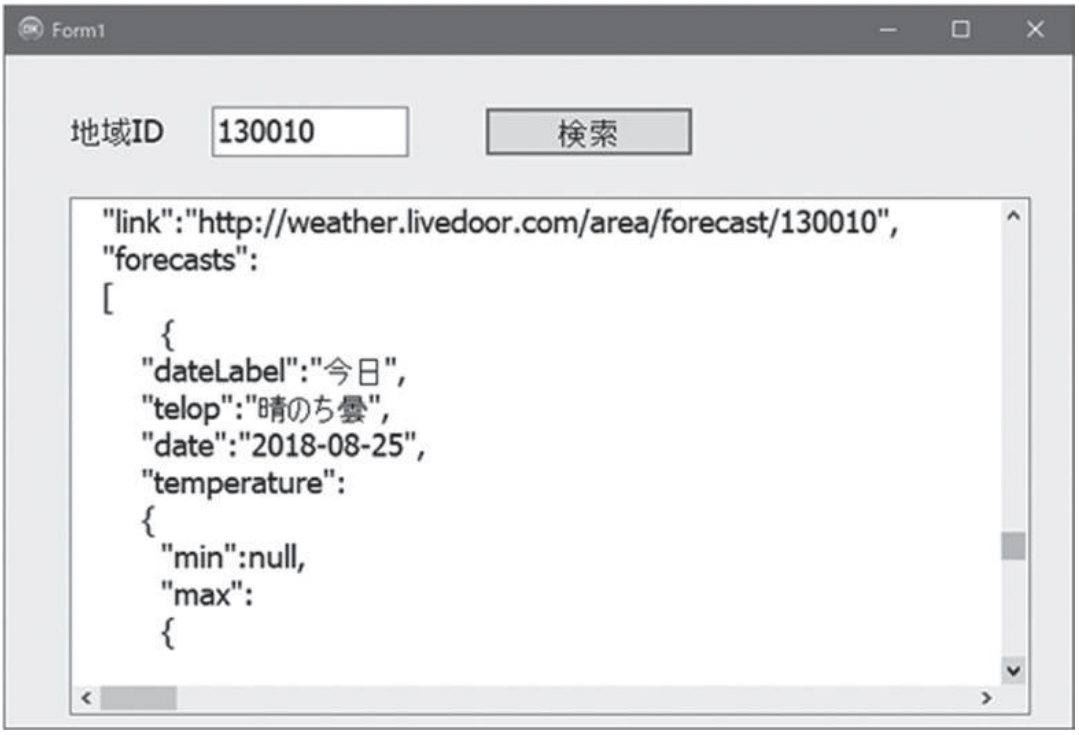


図7　JSON パース例

変数sRet の値

```
{
"result":[
    {
    "subject":"english",
    "score": 80,
    "state":"good"
  },
   {
    "subject":"math",
    "score": 70,
    "state":"normal"
  }
]
}
```

```
procedure TForm1.Button1Click(Sender: TObject);
var
  JSONValue: TJSONValue;
begin
  //JSONデータのパース
  JSONValue := TJSONObject.ParseJSONValue(sRet);
  //結果の取得
  ShowMessage(JSONValue.GetValue<string>('result[0].subject'));
end;
```

実行結果

実装部は、【ソース 6】および【ソース 7】となる。コンポーネントの生成時（Create メソッド）において、内部的に REST コンポーネントを生成し、Language Translator サービスの仕様に基づいたパラメータの設定を行っている。あとは、Translate メソッドにて、【ソース 4】と同様の変換処理を行っている。

このコンポーネントを使用したサンプルプログラムは、とてもシンプルである。画面レイアウトは、【図 18】、プログラムは【ソース 8】である。Web サービス自体をコンポーネント化しているため、API の仕様部分はこのプログラムには含まれていないことがわかる。これによって、将来 Web サービスが終了しても、コンポーネントの内容を別の Web サービスに変更すれば、個々のプログラムを変更する必要がなくなり、耐性の強いプログラムであることがわかるだろう。

6.さいごに

本稿では、REST による Web サービスを活用した Delphi/400 の機能拡張として、いくつかの Web サービスを使用した具体例を紹介してきた。REST + JSON 方式が簡単に Delphi/400 から活用できることがわかる。単純に REST コンポーネントを組み込むだけでも十分活用できるが、コンポーネント化まで検討することにより、より耐性の強い仕組みが作れるのである。ぜひ本稿を参考にいろいろな Web サービスの活用をご検討いただきたい。

M

ソース2　検索ボタンのOnClickイベント

```
uses REST.Types, System.JSON;

procedure TForm1.Button1Click(Sender: TObject);
var
  JSONValue: TJSONValue;
begin
  //URLパラメータの指定
  RESTRequest1.Params.AddItem('CITY', Edit1.Text, pkURLSEGMENT);

  //リクエスト実行
  RESTRequest1.Execute;

  //レスポンスJSONを表示
  JSONValue := RESTResponse1.JSONValue;

  //天気予報の取得
  EditDate1.Text := JSONValue.GetValue<string>('forecasts[0].date');
  EditDate2.Text := JSONValue.GetValue<string>('forecasts[1].date');
  EditDate3.Text := JSONValue.GetValue<string>('forecasts[2].date');
  EditWeather1.Text := JSONValue.GetValue<string>('forecasts[0].telop');
  EditWeather2.Text := JSONValue.GetValue<string>('forecasts[1].telop');
  EditWeather3.Text := JSONValue.GetValue<string>('forecasts[2].telop');
end;
```

図8　お天気情報取得実行例2

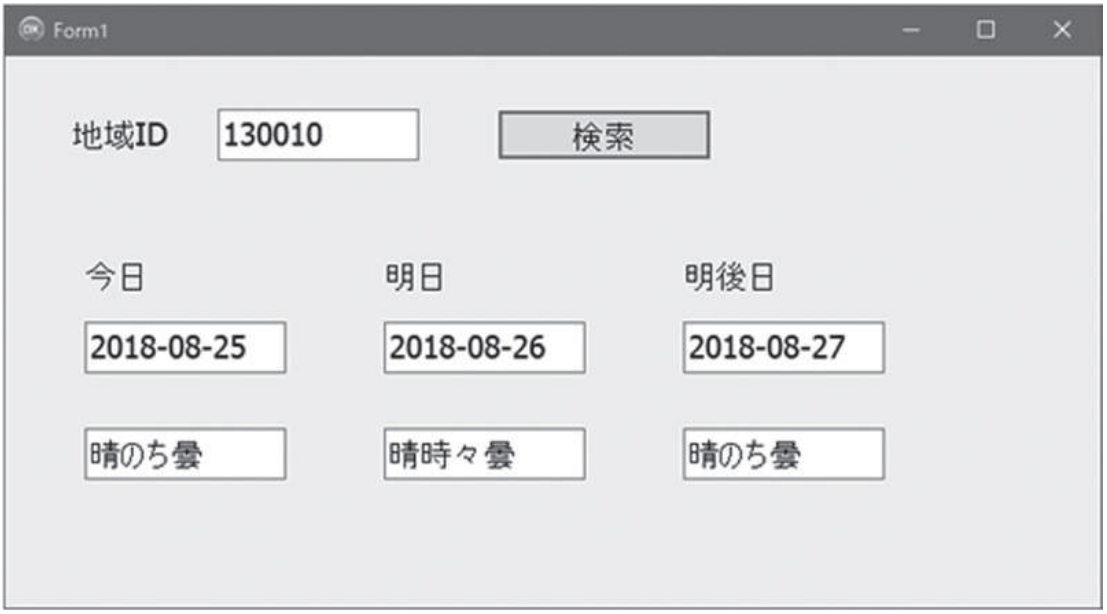
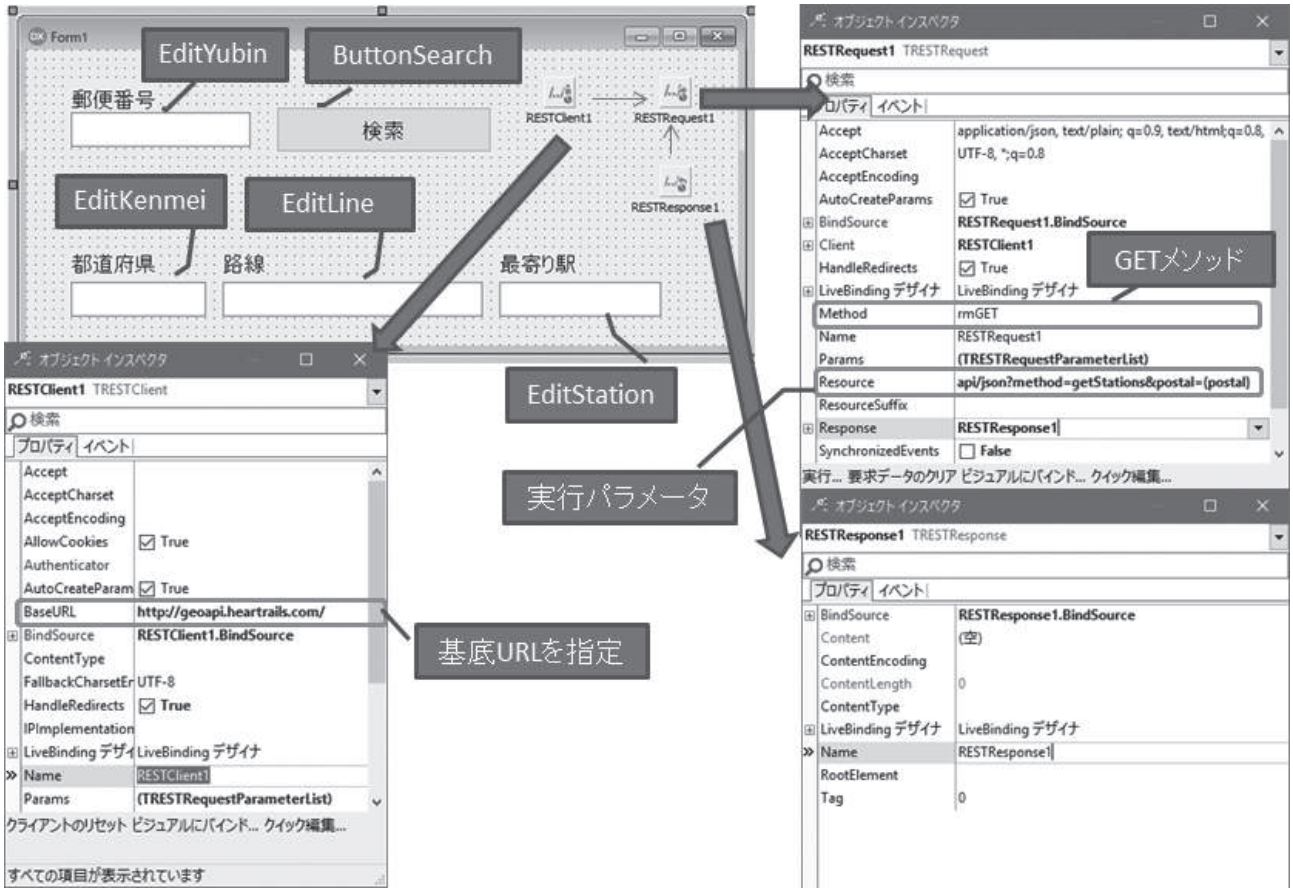


図9　最寄駅情報取得API



ソース3　検索ボタンのOnClickイベント

```
procedure TForm1.ButtonSearchClick(Sender: TObject);
var
  JSONValue: TJSONValue;
begin
  //URLパラメータの指定
  RESTRequest1.Params.AddItem('postal', EditYubin.Text, pkURLSEGMENT);

  //リクエスト実行
  RESTRequest1.Execute;

  //レスポンスJSON取得
  JSONValue := RESTResponse1.JSONValue;

  //JSONより値を取得
  EditKenmei.Text :=
    JSONValue.GetValue<string>('response.station[0].prefecture');
  EditLine.Text :=
    JSONValue.GetValue<string>('response.station[0].line');
  EditStation.Text :=
    JSONValue.GetValue<string>('response.station[0].name');
end;
```

図10　最寄駅情報取得

図11　Webサービス 一覧サイト

図12　Watson API一覧

分類	API種類	機能	ライト
照会応答系	Assistant（照会応答）	自然言語で対話可能なアプリケーションを、シンプルな開発ツールで迅速に構築	対応
言語系	Language Translator（言語変換）	コンテンツのテキストを、ある言語から別の言語にリアルタイムで翻訳	対応
	Natural Language Understanding(自然言語理解)	テキスト分析を行い、コンテンツから概念、エンティティー、キーワード、カテゴリー、感情、関係、意味役割などのメタデータを抽出	対応
心理系	Personality Insights（性格分析）	テキストから筆者のパーソナリティ（ビッグ・ファイブ、価値、ニーズ）の3つの特徴を推測	対応
音声系	Speech to Text(音声認識)	ディープ・ラーニングを活用して、音声からテキストを書き起こす	対応
	Text to Speech(音声合成)	テキストから自然な音声を合成	対応
知識探索系	Discovery(検索)	大量のデータを検索するとともに、適切な意思決定を支援	対応
	Knowledge Studio	業界や分野ごとの知識だけでなく、各分野の言葉の使われ方の微妙な違いをWatsonに教えることができるツール	対応
画像系	Visual Recognition(画像認識)	ディープ・ラーニングを使用して、画像に写った物体・情景・顔など様々なものを分析・認識	対応

※2018年8月現在

図13　Language Translatorサービス管理画面

図14　Language TranslatorサービスのAPI仕様

リクエスト
POST /v3/translate

※2018年8月現在
※API仕様の詳細は、APIリファレンスを参照

パラメータ	タイプ	内容
version	URLパラメータ	V3公開日（固定値：2018-05-01）
request	リクエストの本体	翻訳に必要な下記パラメータをJSON形式で指定
text	string []	翻訳したい元のテキストを指定
source	string	元テキストの言語を指定（ja,en…）
target	string	翻訳後の言語を指定（ja,en…）

レスポンス例

```
{
 "translations" : [ {
  "translation" : "Hi,"
 } ],
 "word_count" : 1,
 "character_count" : 5
}
```

パラメータ	内容
translations: [translation]	翻訳結果のテキスト
word_count	元テキストの単語数
character_count	元テキストの文字数

図15　Language Translatorサービス REST設定

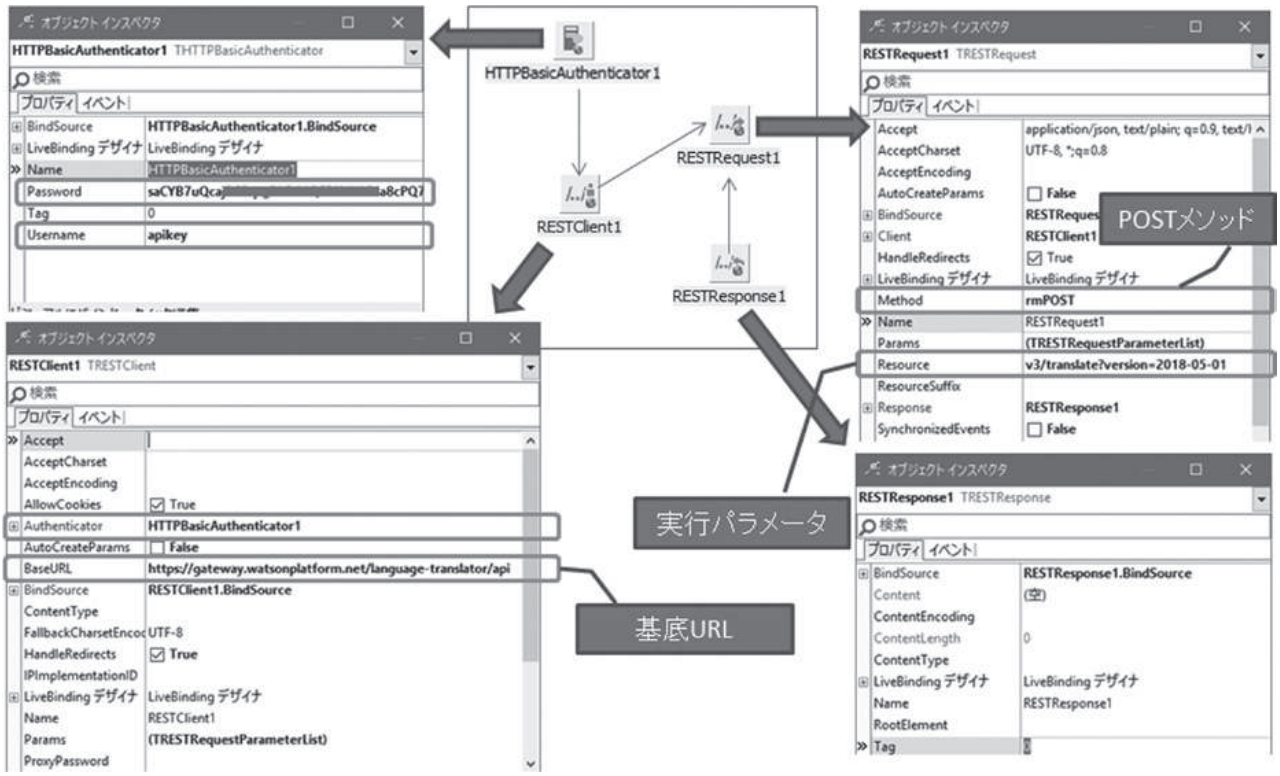
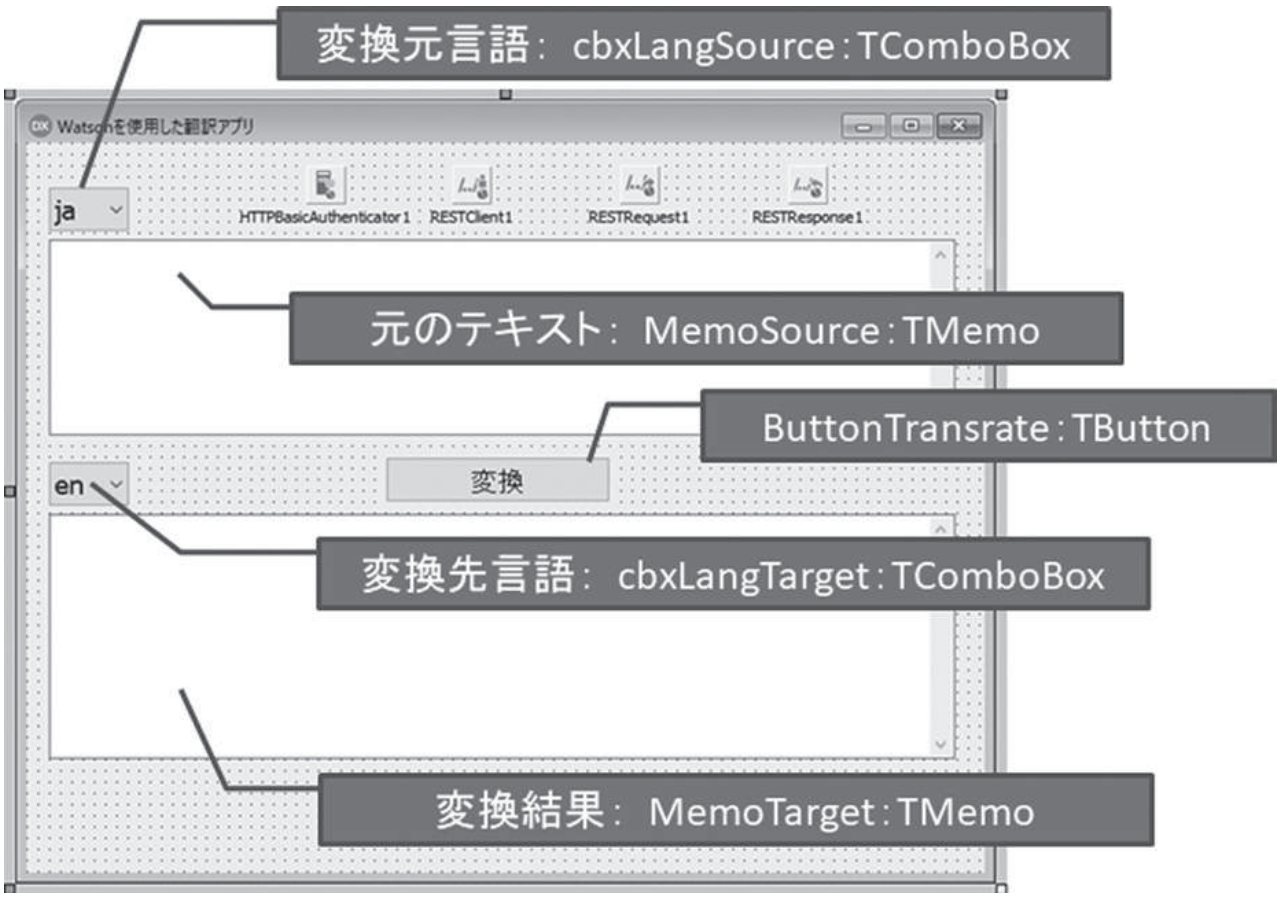


図16　翻訳アプリ画面レイアウト



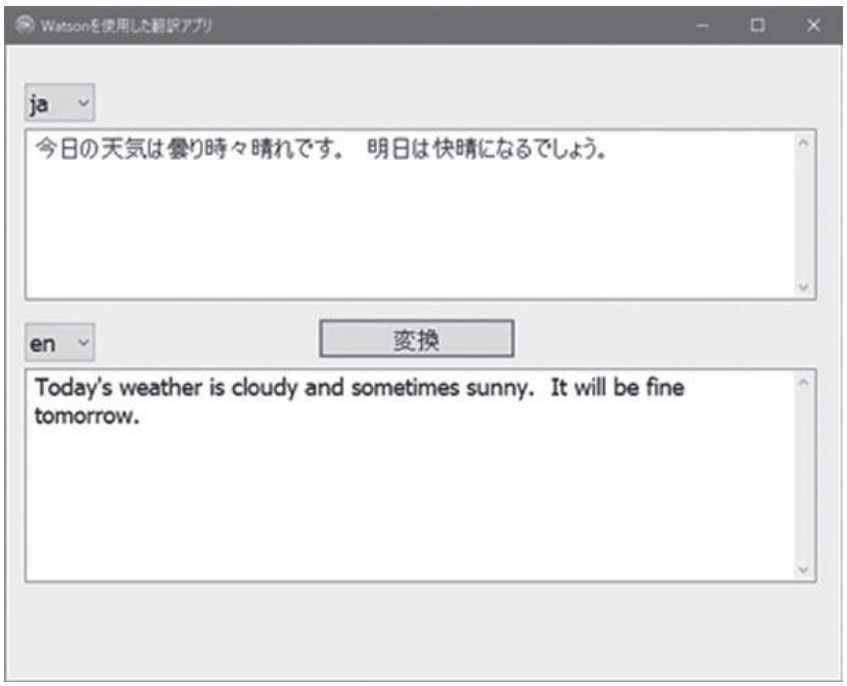
ソース4　変換ボタンのOnClickイベント

```
procedure TForm1.ButtonTransrateClick(Sender: TObject);
var
  RequestJson: TJSONObject;
  JSONValue: TJSONValue;
begin
  //翻訳パラメータの指定
  RequestJson := TJSONObject.Create;
  RequestJson.AddPair('text', MemoSource.Text);       // 翻訳対象文字列
  RequestJson.AddPair('source', cbxLangSource.Text); // 変換元言語
  RequestJson.AddPair('target', cbxLangTarget.Text); // 変換先言語

  //条件の指定
  RESTRequest1.Params.AddItem('', RequestJson.ToString, pkGETorPOST
    , [], ctAPPLICATION_JSON);

  //リクエストの実行
  RESTRequest1.Execute;

  //レスポンスJSONの取得
  JSONValue := RESTResponse1.JSONValue;

  //翻訳結果の取得と表示
  MemoTarget.Text
    := JSONValue.GetValue<string>('translations[0].translation');
end;
```

パラメータとなるJSON文字列を作成

POSTパラメータとしてJSON文字列をセット

図17　翻訳アプリ実行結果



ソース5　TTranslatorコンポーネントの宣言部

```
unit Translator;

interface

uses
  System.SysUtils, System.Classes, System.JSON, REST.Client,
  REST.Authenticator.Basic, Rest.Types, IPPeerClient;

type
  TLanguage = (Japanese, English);

  TTranslator = class(TComponent)
  private
    { Private 宣言 }
    FRestClienct: TRESTClient;
    FRestRequest: TRESTRequest;
    FRestResponse: TRESTResponse;
    FBasicAuthenticator: THTTPBasicAuthenticator;
    FSourceLanguage: TLanguage;
    FTargetLanguage: TLanguage;
    FSource: String;
    FDestination: String;
    function GetLanguageName(ALanguage: TLanguage): String;
  protected
    { Protected 宣言 }
  public
    { Public 宣言 }
    constructor Create(AOwner: TComponent); override;
    procedure Translate;
    property Destination: String read FDestination;
  published
    { Published 宣言 }
    property SourceLanguage: TLanguage read FSourceLanguage write FSourceLanguage;
    property TargetLanguage: TLanguage read FTargetLanguage write FTargetLanguage;
    property Source: String read FSource write FSource;
  end;

procedure Register;
```

ソース6　TTranslatorコンポーネントの実装部①

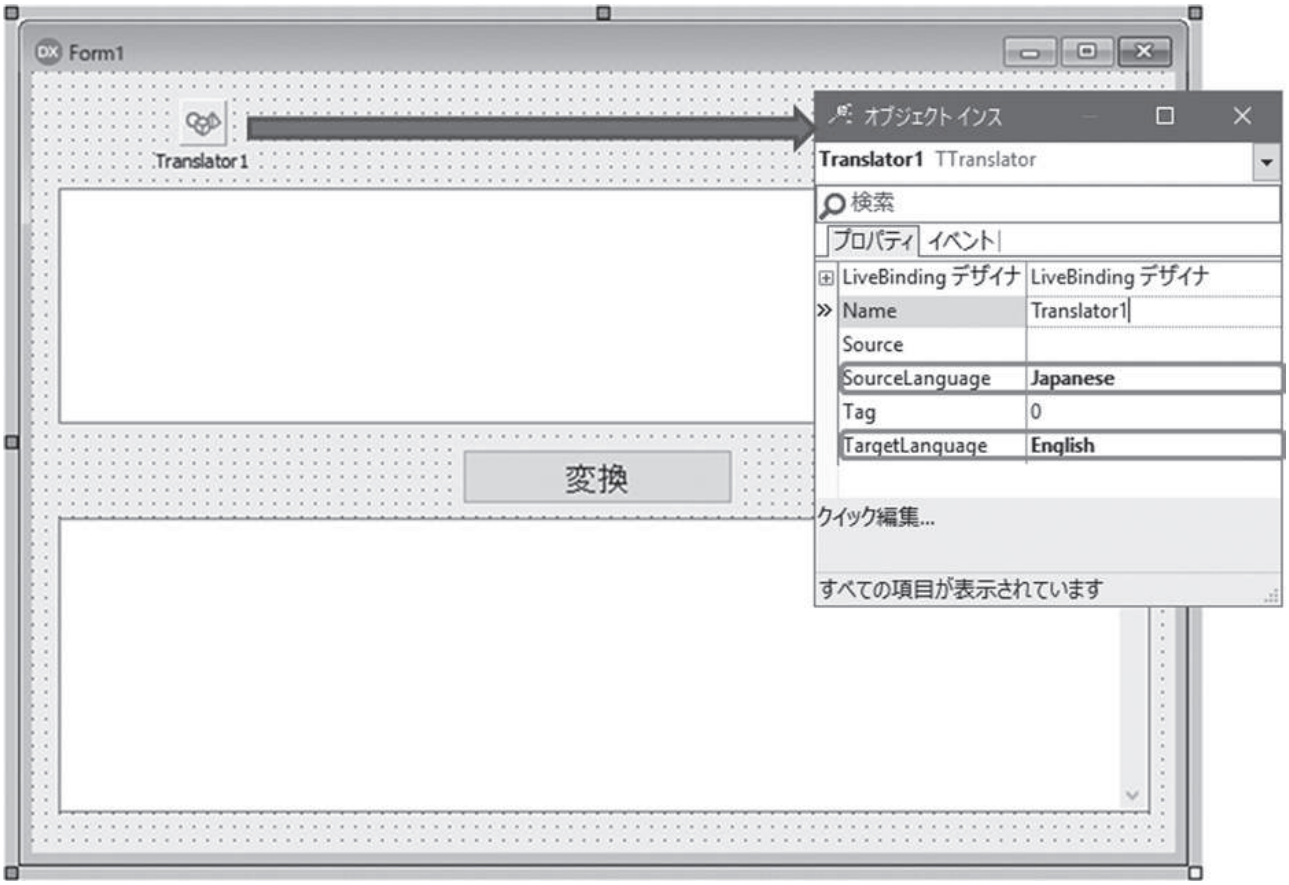
```
implementation

procedure Register;
begin
  RegisterComponents('Samples', [TTranslator]);
end;

{ TTranslator }

constructor TTranslator.Create(AOwner: TComponent);
begin
  inherited;
  //RESTコンポーネントの生成
  FRestClienct := TRESTClient.Create('https://gateway.watsonplatform.net'
    + '/language-translator/api');
  FRestRequest := TRESTRequest.Create(FRestClienct);
  FRestResponse := TRESTResponse.Create(Self);
  FBasicAuthenticator := THTTPBasicAuthenticator.Create('apikey',
    'saCYB7uO                                    69MxzH');

  //RESTコンポーネント関連付け
  FRestClienct.Authenticator := FBasicAuthenticator;
  FRestRequest.Response := FRestResponse;
  //実行メソッドおよびパラメータの初期設定
  FRestRequest.Method := rmPost;
  FRestRequest.Resource := 'v3/translate?version=2018-05-01';

  //プロパティ値の初期設定
  FSourceLanguage := Japanese;
  FTargetLanguage := English;
end;

function TTranslator.GetLanguageName(ALanguage: TLanguage): String;
begin
  case ALanguage of
    Japanese: Result := 'ja';  //日本語="ja"
    English: Result := 'en';   //英語="en"
  end;
end;
```

Language Translatorサービスの
RESTコンポーネントを生成

ソース7　TTranslatorコンポーネントの実装部②

```
procedure TTranslator.Translate;
var
  RequestJson: TJSONObject;
  JSONValue: TJSONValue;
begin
  //翻訳パラメータの指定
  RequestJson := TJSONObject.Create;
  RequestJson.AddPair('text', FSource);       // 翻訳対象文字列
  RequestJson.AddPair('source', GetLanguageName(FSourceLanguage)); // 変換元言語
  RequestJson.AddPair('target', GetLanguageName(FTargetLanguage)); // 変換先言語

  //条件の指定
  FRestRequest.Params.AddItem('', RequestJson.ToString, pkGETorPOST
    , [], ctAPPLICATION_JSON);

  //リクエストの実行
  FRESTRequest.Execute;

  //レスポンスJSONデータの取得
  JSONValue := FRESTResponse.JSONValue;

  //翻訳結果を変数に格納
  FDestination := JSONValue.GetValue<string>('translations[0].translation');
end;
```

図18　コンポーネント化した翻訳アプリ



ソース8　変換ボタンのOnClickイベント

```
procedure TForm1.ButtonTransrateClick(Sender: TObject);
begin
  //変換対象文字列のセット
  Translator1.Source := MemoSource.Text;

  //変換実行
  Translator1.Translate;

  //結果の表示
  MemoTarget.Text := Translator1.Destination;
end;
```

福井 和彦 / 小杉 智昭

Delphi/400 上級

株式会社ミガロ.

システム事業部 プロジェクト推進室

【Delphi/400】Google Maps Platformを使用したアプリケーション開発テクニック

略歴　福井 和彦
1972 年 3 月 20 日生まれ
1994 年 3 月 大阪電気通信大学 工学部卒業
2001 年 4 月 株式会社ミガロ. 入社
2001 年 4 月 システム事業部配属

現在の仕事内容
主に Delphi/400 を使用したシステムの受託開発全般に携わっている。

略歴　小杉 智昭
1973 年 5 月 26 日生まれ
1996 年 3 月 関西大学 工学部卒業
2002 年 3 月 株式会社ミガロ. 入社
2002 年 3 月 RRAD 事業部配属
2007 年 4 月 システム事業部配属

現在の仕事内容：
Delphi/400 を利用した受託開発とシステム保守、導入支援を担当している。

1.はじめに

Google は 2018 年 5 月 2 日（米国時間）に、従来の Google Maps API を同年 6 月 11 日（米国時間）より Google Maps Platform という新しいサービスに移行することを発表した。新しいサービスでは、これまでスタンダードプランとプレミアムプランの 2 種類あった料金プランが統合され、使用量に応じた従量課金の単一プランとなった。この新しい料金プランでは毎月 200USD 分が無料使用枠として付与されることになる。

これまでは Google Maps API の使用環境が社内イントラネットや会員制サイトなど、非公開環境の場合にはプレミアムプランへの契約（年間 10,000USD より）が必要であった。しかし Google Maps Platform では、使用環境が公開・非公開に関係なく従量課金の単一プランとなったため、社内ネットワークで使用する業務システムで、Google マップとの連携が行いやすくなった。

ただし、注意していただきたいのは、請求先アカウントの設定としてクレジットカード情報の登録が必須となる点である。詳しくは、下記「Google Cloud Japan 公式ブログ」の 2018 年 5 月 8 日の記事をご確認いただきたい。
https://cloud-ja.googleblog.com/2018/05

そこで本稿では、Delphi/400 で業務システムを開発する際に広く使われている VCL（C/S 型）のプログラムで、Google Maps Platform を使用して Google マップと連携する方法について説明する。

2.Delphi/400で Googleマップを表示する方法

2-1.Google Maps Platformの利用手続き

Google Maps Platform を使用する場合 API キーが必須となる。そして、API キーを取得するには Google アカウントが必要となるため、Google アカウントをお持ちでない場合は事前の作成が必要となる。

Google アカウントの作成サイト
https://accounts.google.com/signup/v2/webcreateaccount?hl=ja&flowName=GlifWebSignIn&flowEntry=SignUp

（Google で“Google アカウントの作成”を検索すると上位に表示される「Google アカウントの作成」。）

API キーを取得するには、Google アカウントへログインした後、次のサイトの「使ってみる」ボタン【図 1】より、手順に従って登録を行っていく。

Google Maps Platform 公式サイト
https://cloud.google.com/maps-platform/

詳しい登録手順は、さまざまなサイト

図1　Google Maps Platformの利用手続きについて①

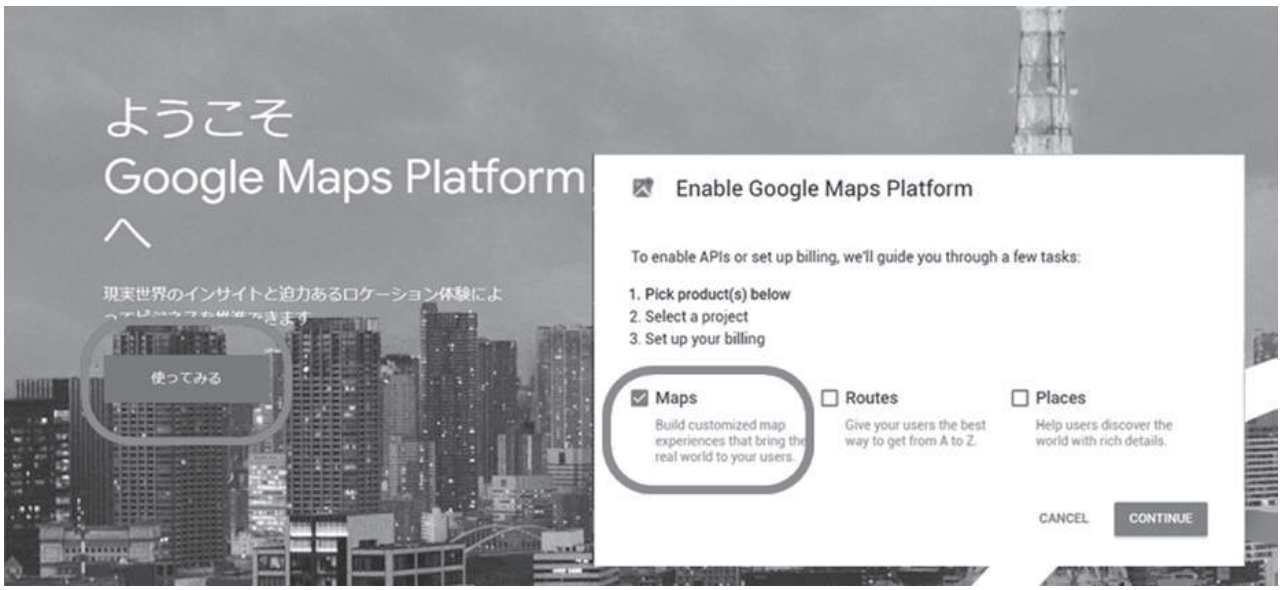
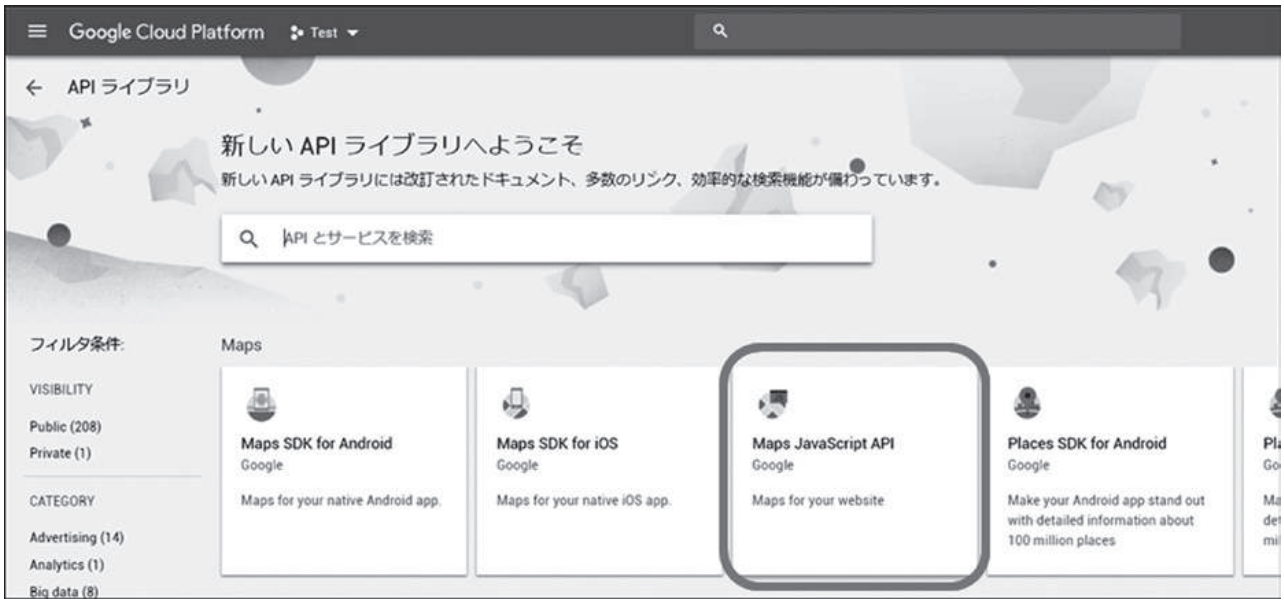


図2　Google Maps Platformの利用手続きについて②



で紹介されているためここでは割愛するが、ポイントは次の3点である。
(1) プロダクトの選択では「Maps」にチェックを付ける。【図1】
(2) API ライブラリより「Maps JavaScript API」を有効にする。【図2】
(3) Maps JavaScript API のキーの制限設定を行う。【図3】【図4】

登録が完了したら、「API とサービス」の認証情報より API キーを取得する。【図5】

2-2.Googleマップを表示する

VCL フォーム上に、2章で取得した API キーを設定した Google Maps Platform を使用して Google マップを表示してみる。

VCL フォームに Web サイトを表示するには「TWebBrowser」コンポーネントを使用する。TWebBrowser コンポーネントは標準では Internet Explorer7.0 の互換モードとして動作する。Google Maps Platform を使用して Google マップを表示する場合、HTML5 として宣言することが推奨されているため、InternetExplorer7.0 の互換モードでは表示されない。そこで、Internet Explorer の最新バージョンとして動作するように設定をする。【図6】のようにレジストリのキーに値を追加することで設定ができる。

設定が完了したら、Delphi/400 を起動して次の手順で Google マップを表示する画面を作成する。

(1) フォームに TWebBrowser コンポーネントを貼り付ける。【図7】

(2) Google マップ表示用 HTML を定数として定義する（【ソース1】を参照）。

この HTML に、2章で取得した API キーを組み込む。また、Google マップを表示する際、中心位置を緯度と経度で指定する「center」と、地図の拡大レベルを指定する「zoom」が必須指定となる。この HTML で指定している緯度と経度は、オーストラリアのシドニー近郊の位置を指定している。

＊ HTML は、次の Google のサイトで公開されているサンプル HTML を使用している。

参考サイト
https://developers.google.com/maps/documentation/
（上記 URL にある“Maps JavaScript API”のリンク先）

(3) フォームの OnShow イベントに【ソース2】のように記述する。

ここで OnShow イベントで行っていることについて説明を行う。

【ソース2のポイント①】

TWebBrowser コンポーネントは OLE コンテナであり、このコンポーネントを貼り付けただけでは InternetExplorer オブジェクト（以下、IE）がまだ準備されていない状態である。このままでは IE に HTML を流し込むための命令が使えないため、まず最初に Navigate メソッドを使ってブランクページを表示させる指示を行う。そうすることで IE を OLE コンテナ上に準備させる。

【ソース2のポイント②】

ポイント①の Navigate メソッドを実行後、Delphi/400 のプログラムは IE の準備完了を待たずに次へ進んでしまうため、IE の準備が完了したかどうかを ReadyState プロパティで確認する。ReadyState プロパティが返す状態は【表1】のとおりである。

ReadyState プロパティが READYSTATE_INTERACTIVE（IE の操作が可能になる状態）まで待ってから、表示したい HTML を IE に流し込む。

HTML を IE に表示させる方法はいくつかあるが、本稿では汎用性の高いストリームを使う方法で行う。ストリームを使うメリットは以下のとおりである。

●ストリームを使うメリット
・プログラムコード内に直接埋め込んだ HTML コードが利用可能。
・ファイルストリームと連携させることで HTML ファイルを利用可能。
・エンコード指定を行うことで、文字コードの変更が可能。

【ソース2のポイント③】

ポイント②で IE が操作可能になるまで待っているが、エラー等に備えて TWebBrowser コンポーネントの Document プロパティが利用可能になっていることを確認する。

次に文字列リストとストリームを準備する。ストリームは TMemoryStream を使用する。HTML の文字列定数（cGoogleMap）を文字列リストの Text プロパティへセットし、文字列リストの SaveToStream メソッドでストリームに書き込む。ただし、SaveToStream メソッドで書き込んだ直後はストリームの位置が HTML の末尾に移動してしまうため、ストリームの Seek メソッドで先頭位置へ戻している。（位置の変更は Position プロパティでも可能）

そして、IE の IPersistStreamInit インターフェースの Load メソッドでストリームを渡すのだが、Delphi/400 のストリーム（TStream 型）を COM ベースの IStream 型へ変換する必要があるため、TStreamAdapter クラスを経由して渡す。TStreamAdapter クラスは TInterfacedObject クラスを継承しており、自動解放されるため明示的な解放を行う必要はない。最後にストリームと文字列リストを解放して、IE への表示処理は終了となる。

完成した画面を実行すると、オーストラリアのシドニー周辺の地図が表示される。【図8】

3.Googleマップの機能を連携するテクニック

3-1.Googleマップにマーカーを立てる

この章では、サンプルファイルに登録されている複数の営業所の位置を、緯度と経度を使用して Google マップ上にマーカーを立てる方法について説明をする。完成イメージは【図9】となる。また、緯度と経度についてはファイルに登録されているものとする。この章以降で使用するサンプルファイルのイメージは、【図10】【図11】を参照していただきたい。サンプルファイルは、IBM i の特定のライブラリーに存在するものとする。

図3　Google Maps Platformの利用手続きについて③

図4　Google Maps Platformの利用手続きについて④

それでは、次の手順で画面を作成していく。

(1)【図 12】のように、フォームにTDBGrid コンポーネントとTWebBrowser コンポーネントを貼り付ける。そして、サンプルファイルを参照するためにFireDAC 関連のコンポーネントと TDataSource コンポーネントを貼り付ける。FireDAC 関連の各コンポーネントの設定は次のとおり。

●TFDConnection コンポーネントの設定
接続エディタにて設定する。
・接続定義名：CO400DEF
・Database：Delphi/400Configuration の AS/400Name
・User_Name：IBM i 接続ユーザー
・Password ：IBM i 接続ユーザーのパスワード
・ODBCAdvanced：LibraryOption =[サンプルファイルが有る Library 名]

● TFDQuery コンポーネントの設定
SQL プロパティにサンプルファイルを参照する以下の SQL 文を埋め込む。
(SQL 文)
SELECT * FROM [サンプルファイル名]
ORDER BY SPEGCD

TDBGrid コンポーネントのカラムには営業所名（SPEGNM）を設定する。

(2)Google マップ表示用 HTML は【ソース 3】【ソース 4】を参照。この章では HTML を 2 つに分割している。分割している理由については後述する。また【ソース 3】では、Google マップにマーカーを立てるために Google Maps Platform の API を使用した JavaScript「setMarker」を組み込んでいる。

【ソース 4】では、Google マップ表示後に地図の中心にしたい緯度と経度を組み込んでいる。本稿では「難波サンケイビル　本社」を中心にするため、その緯度と経度を組み込んでいる。そして 3 章と同様に、2 章で取得した API キーを HTML へ組み込む。

(3) フォームの OnShow イベントでは、Google マップ表示用 HTML を使用して TWebBrowser コンポーネントに Google マップを表示している。OnShow イベントに【ソース 5】【ソース 6】のように記述する。それでは OnShow イベントで行っている内容について詳しく見ていこう。

【ソース 5 のポイント①】
IBM i に接続し、TDBGrid コンポーネントに営業所名の一覧を表示する。

【ソース 5 のポイント②】
ここでは Google マップを表示する HTML を編集している。HTML の編集イメージは【図 13】を参照していただきたい。
まず、サンプルファイルを全件参照しながら、各レコードの緯度（SPLATI）と経度（SPLNGI）を取得する。そして【ソース 3】で組み込んだ、マーカーを立てる JavaScript「setMarker」の引数に取得した緯度と経度をセットする。この setMarker を【ソース 3】の HTML の続きとして、サンプルファイルに登録されているデータ全件分を結合する。こうすることで、1 回の HTML のリクエストで全件分の営業所のマーカーが付いた状態の Google マップを表示することができる。

【ソース 6 のポイント①】
【ソース 6】の Google マップを表示する処理については 3 章の内容と同じになる。ただし、Google マップを表示する HTML をセットする個所は、【ソース 5 のポイント②】で説明した編集結果をセットするようにする。

完成した画面を実行すると完成イメージ【図 9】のように、各営業所の位置にマーカーが立っている Google マップが表示される。

3-2.クリックしたマーカーの情報を取得する

この章では、3-1 で作成した画面に機能を追加し、クリックした Google マップのマーカーに保管している情報（営業所 CD）を取得して、その情報を基に営業所の詳細情報を表示する方法について説明する。ただし、マーカーの情報を取得するためには TWebBrowser コンポーネントを経由して JavaScript を実行して戻り値を取得する必要がある。しかし、TWebBrowser コンポーネントには JavaScript を実行するメソッドは実装されていない。独自で実装することも可能ではあるが簡単ではない。
そこで本稿では、JavaScript を実行するメソッドなど TWebBrowser コンポーネントでは実装されていないメソッドやプロパティが実装されている「TEmbeddedWB」コンポーネントというフリーのコンポーネントを利用する。TEmbeddedWB コンポーネントは bsalsa productions にて開発・公開されていたフリーのコンポーネント群の 1 つだが、現在は little earth solutions によって GitHub 上でメンテナンス・公開されている。

TEmbeddedWB コンポーネントのダウンロードサイト
https://github.com/littleearth/Delphi-EmbeddedWB

GitHub 上では Delphi 5 ～ Delphi 10 Seattle までが対象となっているが、Delphi 10.1 Berlin や Delphi 10.2 Tokyo 向けの dpk ファイルも存在する（＊ dpk ファイルの読み込み前に res ファイルを削除しなければ正しく取り込めない）。ダウンロードサイトより TEmbeddedWB コンポーネントをダウンロードして Delphi/400 の開発環境にインストールをする。

それでは、次の手順に従って、4 章で作成した画面に機能を追加していく。

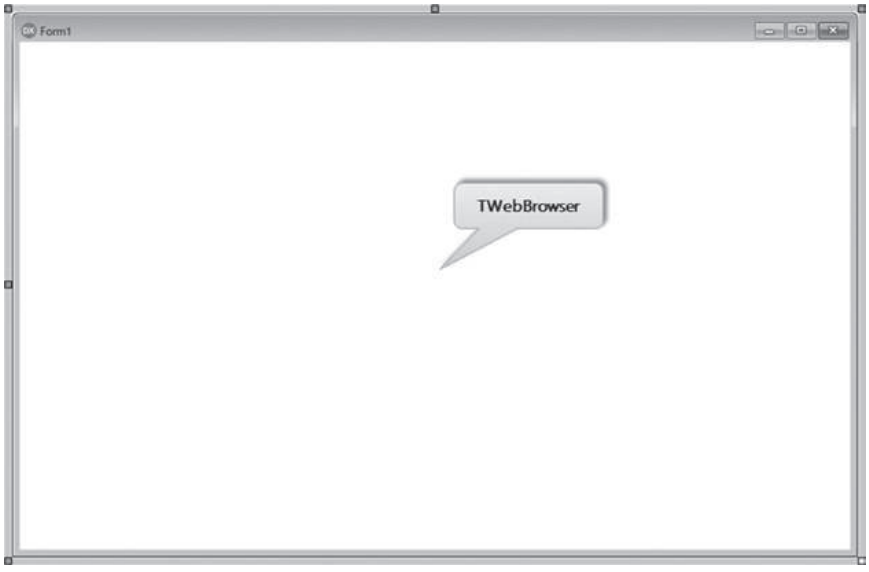
(1)【図 14】に従ってコンポーネントの追加／変更を行う。変更内容は、まず TWebBrowser コンポーネントを先程インストールした TEmbeddedWB コンポーネントに入れ替える。
次に、TTimer コンポーネントを追加し、Enabled プロパティを False、Interval プロパティを 200 に設定する。TTimer コンポーネントの用途については後述する。そして、営業所の詳細情報を表示するために TDBEdit を追加する。

図5　APIキーの取得

図6　Googleマップを表示する①

値（10進数）	値（16進数）	バージョン
11001	0x2AF9	Internet Explorer 11, Edgeモード（最新のバージョンでレンダリング）
11000	0x2AF8	Internet Explorer 11

図7　Googleマップを表示する②



(2) Google マップを表示する HTML【ソース 3】【ソース 4】に対して、【ソース 7】【ソース 8】のように変更を行う。変更内容は以下のとおり。

【ソース 8 の変更点①】
説明の順番が前後するが、グローバル変数である営業所 CD（mrkegcd）を追加宣言する。この営業所 CD（mrkegcd）を【ソース 7】内で使用する。

【ソース 7 の変更点①】
まず setMarker の引数に営業所 CD を受け取る項目（egcd）を追加する。

【ソース 7 の変更点②】
受け取った営業所 CD（egcd）を、マーカーに紐付けて保管する営業所 CD（marker.egcd）にセットする。

【ソース 7 の変更点③】
マーカーにクリックイベントを定義し、マーカーをクリックした際に、マーカーに保管している営業所 CD（marker.egcd）をグローバル変数の営業所 CD（mrkegcd）にセットする。

【ソース 7 の変更点④】
新たに営業所 CD（mrkegcd）の値を取得する JavaScript「getmrkegcd」を追加し、営業所 CD（mrkegcd）を戻り値として返すようにする。

(3) フォームの OnShow イベントを【ソース 9】に従って変更を行う。変更内容は、【ソース 7】にて setMarker の引数に営業所 CD が追加されたため、【ソース 5】の Google マップを表示する HTML を編集しているところで使用している setMarker の引数に、【ソース 9 の変更点①】のようにサンプルファイルの営業所 CD（SPEGCD）をセットするようにする。これでマーカーを立てる際に営業所 CD（SPEGCD）がマーカーに紐付いて保管される。

(4) Google マップのマーカーをクリックした時に、【ソース 7】で setMarker に定義したクリックイベントで、マーカーで保管している営業所 CD がグローバル変数の営業所 CD にセットされている。VCL からは TEmbeddedWB コンポーネントの OnClick イベント（以下 OnClick イベントとする）で、【ソース 7】で実装された getmrkegcd を実行してグローバル変数の営業所 CD を戻り値とし受け取るようにする。しかし、OnClick イベントの方がマーカーのクリックイベントよりも先に発生してしまうため、戻り値が正しく取得できない。そこで、OnClick イベントと、マーカーのクリックイベントが非同期で発生することを考慮し、TTimer コンポーネントで getmrkegcd の実行タイミングを遅らせる。

OnClick イベントと TTimer コンポーネントの OnTimer イベントを【ソース 10】のように実装する。

【ソース 10 のポイント①】
OnClick イベントでは、TTimer コンポーネントの OnTimer イベントを起動するのみとなる。

【ソース 10 のポイント②】
TTimer コンポーネントの OnTimer イベントで、OnClick イベント発生後（200 ミリ秒後）に getmrkegcd を実行して戻り値の営業所 CD を取得する。JavaScript である getmrkegcd を実行するには TEmbeddedWB コンポーネントに実装されている「ExecScriptEx」メソッドを使用する。ExecScriptEx メソッドは実行した Script の戻り値を取得できるメソッドで、語尾に“Ex”が付いていない「ExecScript」メソッドも実装されているが、こちらは Script を実行するのみのメソッドとなる。

【ソース 10 のポイント③】
取得した営業所 CD でサンプルファイルより該当営業所 CD を検索し詳細情報を表示させる。

ここまでで実装した「クリックしたマーカーの営業所 CD を取得する」処理のイメージは【図 15】を参照いただきたい。

完成した画面を実行すると完成イメージ【図 16】のように、Google マップ上のマーカーをクリックするごとに、画面左の営業所一覧はクリックした営業所にレコードが移動し、それに合わせて画面下の営業所詳細情報はクリックした営業所の情報が表示される。

3-3.Googleマップを移動・拡大する

この章では、3-1 から使用している画面にさらに機能を追加し、画面左の営業所一覧をクリックすると、Google マップの該当営業所のマーカーが真ん中に移動し、かつ拡大して表示する方法について説明する。

この章での変更はソースのみとなる。まずは Google マップを表示する HTML を【ソース 11】に従って変更する。

【ソース 11 の変更点①】
引数で受け取った緯度と経度を Google マップの中心にし、拡大表示する JavaScript「moveMapCenter」 を追加実装する。

次に、【ソース 12】に従って TDBGrid コンポーネントの OnClick イベントを実装する。【ソース 11】で実装した moveMapCenter にサンプルファイルの緯度と経度を引数として渡して実行する。

完成した画面を実行すると完成イメージ【図 17】のように、画面左の営業所一覧をクリックするごとに Google マップの該当営業所のマーカーが真ん中に移動し、かつ拡大して表示される。

以上で「Delphi/400 で Google Maps Platform を使用したアプリケーション開発テクニック」の説明は終了となる。

4.最後に

初めに述べたように、Google Maps Platform のサービス開始に伴い、非公開の社内用 Web アプリケーションや C/S 型アプリケーションでもプレミアムプランの契約なしに、API を用いた Google マップが利用できるようになった。今後は、そのプレミアムプラン契約がネックとなって利用を諦めていた非公開環境での活用も増えてくるだろう。

ただし、無料で使用できる枠には限度があり、無料で使用できる枠は使用するサービスによって異なるため、Google Maps Platform を利用する際には注意

ソース1　Googleマップ表示用HTML

```
  cGoogleMap = '<!DOCTYPE html>'
             + '<html>'
             + '  <head>'
             + '    <title>Simple Map</title>'
             + '    <meta name="viewport" content="initial-scale=1.0">'
             + '    <meta charset="utf-8">'
             + '    <style>'
             + '      /* Always set the map height explicitly to define the size of the div'
             + '       * element that contains the map. */'
             + '      #map {'
             + '        height: 100%;'
             + '      }'
             + '      /* Optional: Makes the sample page fill the window. */'
             + '      html, body {'
             + '        height: 100%;'
             + '        margin: 0;'
             + '        padding: 0;'
             + '      }'
             + '    </style>'
             + '  </head>'
             + '  <body>'
             + '    <div id="map"></div>'
             + '    <script>'
             + '      var map;'
             + '      function initMap() {'
             + '        map = new google.maps.Map(document.getElementById(''map''), {'
             + '          center: {lat: -34.397, lng: 150.644},'
             + '          zoom: 8'
             + '        });'
             + '      }'
             + '    </script>'
             + '    <script src="https://maps.googleapis.com/maps/api/js?key=[APIキー]&callback=initMap"'
             + '    async defer></script>'
             + '  </body>'
             + '</html>';
```

centerとzoomは必須オプション

2章で取得したAPIキーを設定する

ソース2　フォームのOnShowイベント

usesにWinapi.ActiveXを追加する

```
procedure TForm1.FormShow(Sender: TObject);
const
  [ソース1] (Googleマップ表示用HTML)
var
  sl: TStringList;
  ms: TMemoryStream;
begin
  WebBrowser1.Navigate('about:blank') ;

  while WebBrowser1.ReadyState < READYSTATE_INTERACTIVE do
    Application.ProcessMessages;

  if Assigned(WebBrowser1.Document) then
  begin
    sl := TStringList.Create;
    try
      ms := TMemoryStream.Create;
      try
        sl.Text := cGoogleMap;
        sl.SaveToStream(ms);
        ms.Seek(0, 0);
        (WebBrowser1.Document as IPersistStreamInit).Load(TStreamAdapter.Create(ms));
      finally
        ms.Free;
      end;
    finally
      sl.Free;
    end;
  end;
end;
```

ポイント①

ポイント②

ポイント③

「定数」→「文字列リスト」

「文字列リスト」→「ストリーム」

「ストリーム」→「IE」

が必要となる。詳しくは、下記参考サイトで確認していただきたい。

●参考サイト
https://cloud.google.com/maps-platform/
（上記 URL のメニューにある“料金”のリンク先）

M

表1　ReadyStateプロパティが返す状態

ReadyStateプロパティの戻り値	状態
READYSTATE_UNINITIALIZED	IEに対する初期化未完了
READYSTATE_LOADING	IEのロード中
READYSTATE_LOADED	IEのロード完了・操作不可能
READYSTATE_INTERACTIVE	IEの操作可能
READYSTATE_COMPLETE	IEの全データ読込完了

図8　Googleマップを表示する③

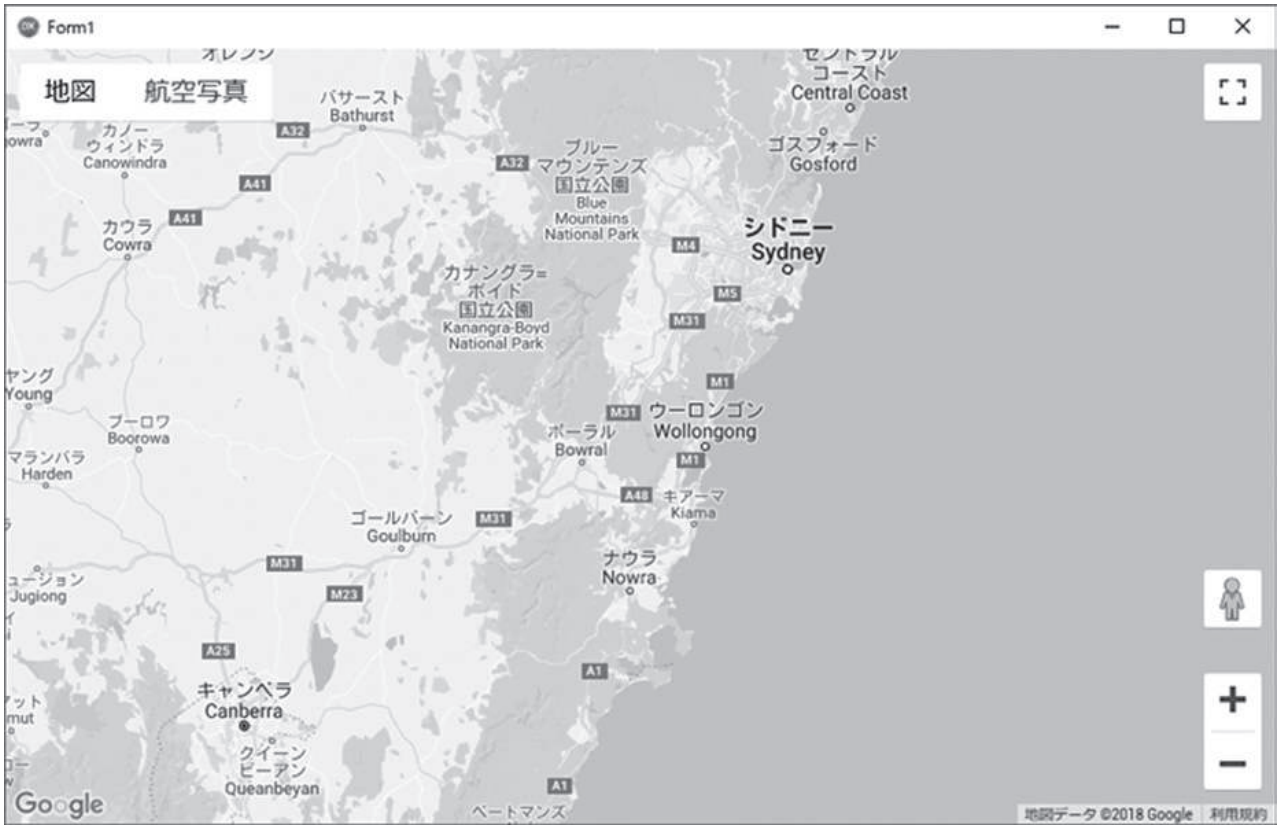


図9　Googleマップにマーカーを立てる①

図10　Googleマップにマーカーを立てる②

サンプルファイルイメージ

No.	Key	項目ID	項目名	属性（桁数）
1	1	SPEGCD	営業所CD	A（6）
2		SPEGNM	営業所名	O（32）
3		SPEGRN	営業所略称	O（22）
4		SPPOST	郵便番号	A（8）
5		SPADR1	住所1	O（42）
6		SPADR2	住所2	O（42）
7		SPTELN	電話番号	A（15）
8		SPFAXN	FAX番号	A（15）
9		SPLATI	緯度	S（10, 7）
10		SPLNGI	経度	S（10, 7）

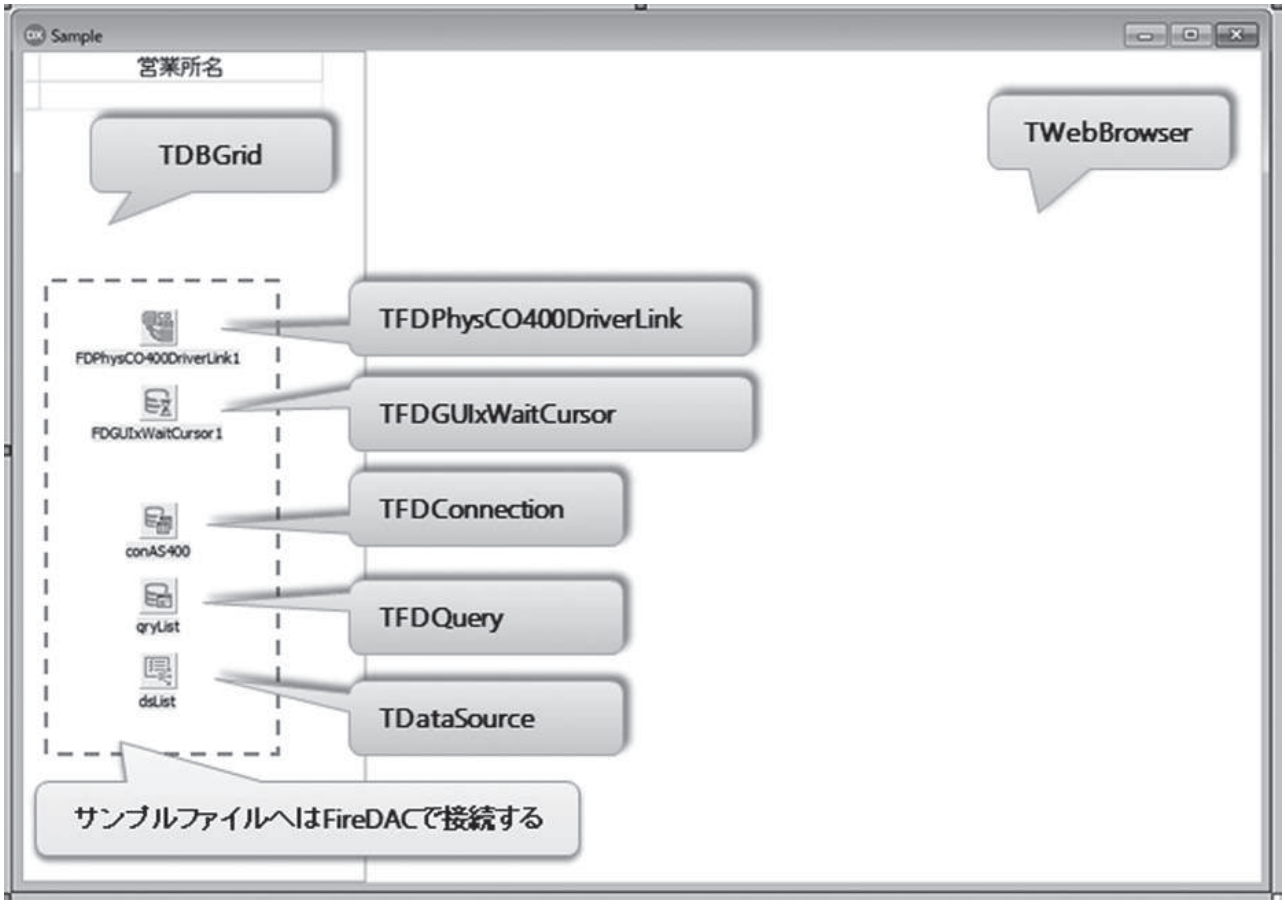
図11　Googleマップにマーカーを立てる③

サンプルデータイメージ

営業所CD	営業所名	営業所略称	郵便番号	住所1	住所2
100001	難波サンケイビル　本社	難波サンケイ	556-0017	大阪府大阪市浪速区湊町～	難波サンケイビル～
200001	霞が関　営業所	霞が関	100-0013	東京都千代田区霞が関～	霞が関東急ビル～

電話番号	FAX番号	緯度	経度
06-6631-8601	06-6631-8603	34.6651420	135.4955550
03-5510-5701	03-5510-5702	35.6714410	139.7447100

図12　Googleマップにマーカーを立てる④



ソース3　Googleマップ表示用HTML①

```
GMapHTML_1
    = '<!DOCTYPE html>'
    + '<html>'
    + '  <head>'
    + '    <title>Simple Map</title>'
    + '    <meta name="viewport" content="initial-scale=1.0">'
    + '    <meta charset="utf-8">'
    + '    <style>'
    + '      /* Always set the map height explicitly to define the size of the div'
    + '       * element that contains the map. */'
    + '      #map {'
    + '        height: 100%;'
    + '      }'
    + '      /* Optional: Makes the sample page fill the window. */'
    + '      html, body {'
    + '        height: 100%;'
    + '        margin: 0;'
    + '        padding: 0;'
    + '      }'
    + '    </style>'
    + '    <script type="text/javascript">'
    + '      // マーカーセット用関数'
    + '      function setMarker(lat, lng) {'
    + '        var latlng = new google.maps.LatLng(lat, lng);'
    + '        var marker;'
    + '        marker = new google.maps.Marker({map: map, position: latlng});'
    + '      }'
    + '      window.onload = function() {';
```

GoogleMaps Platformの
APIで、緯度と経度から
マーカーをセットする

ソース4　Googleマップ表示用HTML②

```
GMapHTML_2
    = '      };'
    + '    </script>'
    + '  </head>'
    + '  <body>'
    + '    <div id="map"></div>'
    + '    <script>'
    + '      var map;'
    + '      function initMap() {'
    + '        map = new google.maps.Map(document.getElementById(''map''), {'
    + '          center: {lat: 34.665160, lng: 135.495414},'
    + '          zoom: 14'
    + '        });'
    + '      }'
    + '    </script>'
    + '    <script src="https://maps.googleapis.com/maps/api/js?key=[APIキー]&callback=initMap"'
    + '    async defer></script>'
    + '  </body>'
    + '</html>';
```

難波サンケイビル 本社の緯度/経度を設定
表示直後、難波サンケイビル 本社を中心に
する

2章で取得したAPIキーを設定する

ソース5　フォームのOnShowイベント

usesにWinapi.ActiveXを追加する

```
procedure TfrmSample.FormShow(Sender: TObject);
const
  [ソース3] (Googleマップ表示用HTML①)
  [ソース4] (Googleマップ表示用HTML②)
var
  sl: TStringList;
  ms: TMemoryStream;
  sMap: string;
begin
  // IBMiへ接続
  conAS400.Connected := True;
  // 営業所リスト表示
  qryList.Open;
  // 営業所の緯度と経度でマーカーをセットするJavaScriptをHTMLに埋め込む処理
  sMap := GMapHTML_1;
  qryList.DisableControls;
  try
    qryList.Last;
    while not qryList.Bof do
    begin
      sMap := sMap + ' setMarker('
                   + qryList.FieldByName('SPLATI').AsString + ','
                   + qryList.FieldByName('SPLNGI').AsString + ');';
      qryList.Prior;
    end;
  finally
    qryList.EnableControls;
  end;
  ↓ [ソース6]へ続く
```

ポイント①

ポイント②

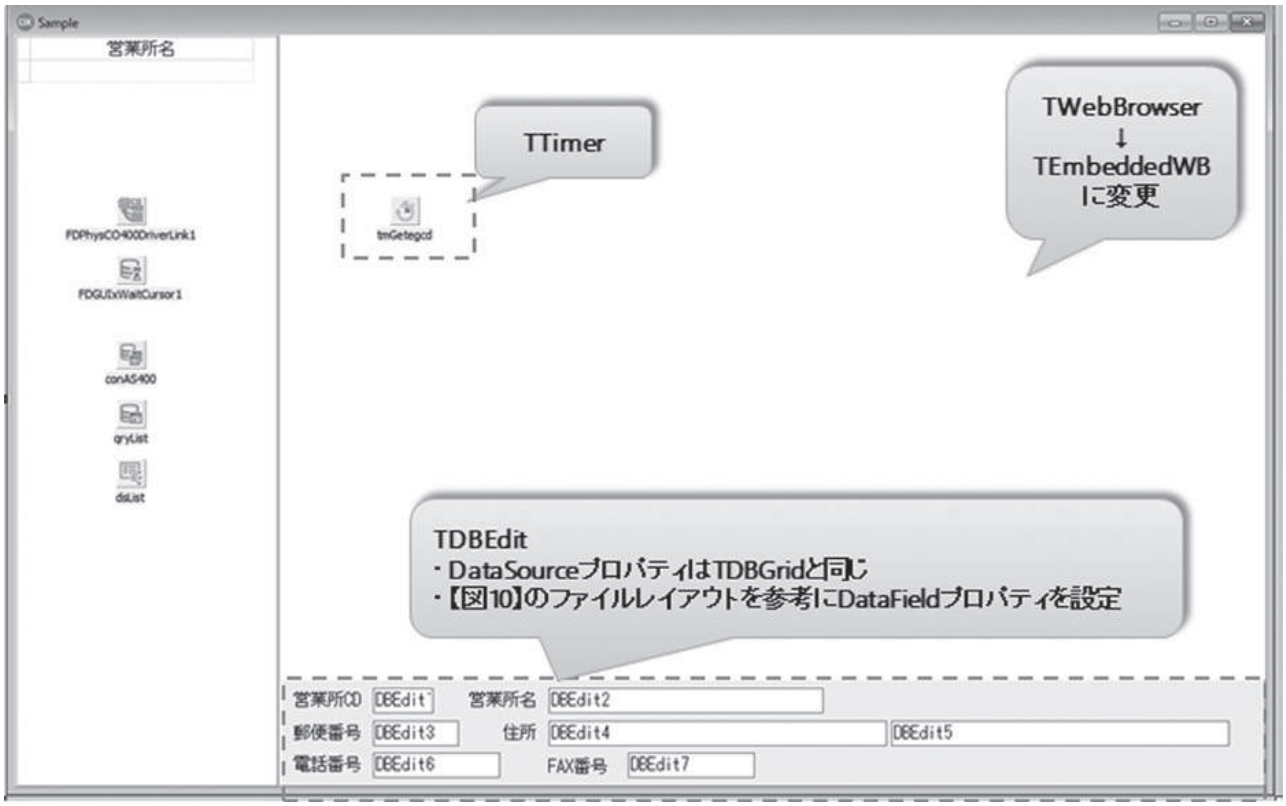
ソース6　フォームのOnShowイベント

```
  ↓ [ソース5]の続き

 // Googleマップ表示
  wbMap.Navigate('about:blank') ;
  while wbMap.ReadyState < READYSTATE_INTERACTIVE do
    Application.ProcessMessages;

  if Assigned(wbMap.Document) then
  begin
    sl := TStringList.Create;
    try
      ms := TMemoryStream.Create;
      try
        sl.Text := sMap + GMapHTML_2;
        sl.SaveToStream(ms);
        ms.Seek(0, 0);
        (wbMap.Document as IPersistStreamInit).Load(TStreamAdapter.Create(ms));
      finally
        ms.Free;
      end;
    finally
      sl.Free;
    end;
  end;
end;
```

ポイント①

図13　Googleマップにマーカーを立てる⑤

図14　クリックしたマーカーの情報を取得する①



ソース7　Googleマップ表示用HTML①【ソース3】からの変更点

```
GMapHTML_1
  = 【ソース3】と同じ ～

  // マーカーセット用関数
  + '      function setMarker(lat, lng, egcd) {'
  + '        var latlng = new google.maps.LatLng(lat, lng);'
  + '        var marker;'
  + '        marker = new google.maps.Marker({map: map, position: latlng});'
  + '        marker.egcd = egcd;'
  + '        google.maps.event.addListener(marker, ''click'', function() {'
  + '          mrkegcd = marker.egcd;'
  + '        });'
  + '      }'
  // 営業所CD (mrkegcd) の値を取得する関数
  + '      function getmrkegcd(){'
  + '        return (mrkegcd);'
  + '      }'
  + '      window.onload = function() {';
```

変更点①
引数に営業所CDを追加

変更点②
引数で受け取った営業所CDを
マーカーに紐付けて保管

変更点③
マーカーのクリックイベントを
定義し、クリック時にマーカー
に保管している営業所CDを変
数に退避

変更点④
マーカークリック時に、変数に
退避した営業所CDを取得する
JavaScript

ソース8　Googleマップ表示用HTML②【ソース4】からの変更点

```
GMapHTML_2
  = '      };'
  + '    </script>'
  + '  </head>'
  + '  <body>'
  + '    <div id="map"></div>'
  + '    <script>'
  + '      var map;'
  + '      var mrkegcd;'
  + '      function initMap() {'
  + '        map = new google.maps.Map(document.getElementById(''map''), {'
  + '          center: {lat: 34.665160, lng: 135.495414},'
  + '          zoom: 14'
  + '        });'
  + '      }'
  + '    </script>'
  + '    <script src="https://maps.googleapis.com/maps/api/js?key=[APIキー]&callback=initMap"'
  + '    async defer></script>'
  + '  </body>'
  + '</html>';
```

変更点①
マーカークリック時、マーカーに保管している営業所CDを
退避する変数を宣言
この変数の値をgetmrkegcd関数で戻り値として返す

ソース9　フォームのOnShowイベント【ソース5】からの変更点

```
procedure TfrmSample.FormShow(Sender: TObject);

  【ソース5】と同じ ～

  // 営業所の緯度と経度でマーカーをセットするJavaScriptをHTMLに埋め込む処理
  sMap := GMapHTML_1;
  qryList.DisableControls;
  try
    qryList.Last;
    while not qryList.Bof do
    begin
      sMap := sMap + ' setMarker('
                   + qryList.FieldByName('SPLATI').AsString + ','
                   + qryList.FieldByName('SPLNGI').AsString + ','
                   + QuotedStr(qryList.FieldByName('SPEGCD').AsString) + ');';
      qryList.Prior;
    end;
  finally
    qryList.EnableControls;
  end;

  【ソース5】と同じ ～
```

変更点①
setMarkerの引数に営業所CDを追加

ソース10　TEmbeddedWBコンポーネントのOnClickイベント

```
procedure TfrmSample.wbMapClick(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  // Googleマップのマーカークリックが後に発生するため、Timerでタイムラグを調整
  tmGetegcd.Enabled := True;
end;
```

ポイント①

TTimerコンポーネントのOnTimerイベント

```
procedure TfrmSample.tmGetegcdTimer(Sender: TObject);
var
  vResult: OleVariant;
  segcd: string;
begin
  tmGetegcd.Enabled := False;;

  vResult := wbMap.ExecScriptEx('getmrkegcd', ['']);
  segcd := vResult;

  qryList.Locate('SPEGCD', segcd, []);
end;
```

TTimerコンポーネントのIntervalプロパティは200ミリ秒で設定

ポイント②

ポイント③

図15　クリックしたマーカーの情報を取得する②

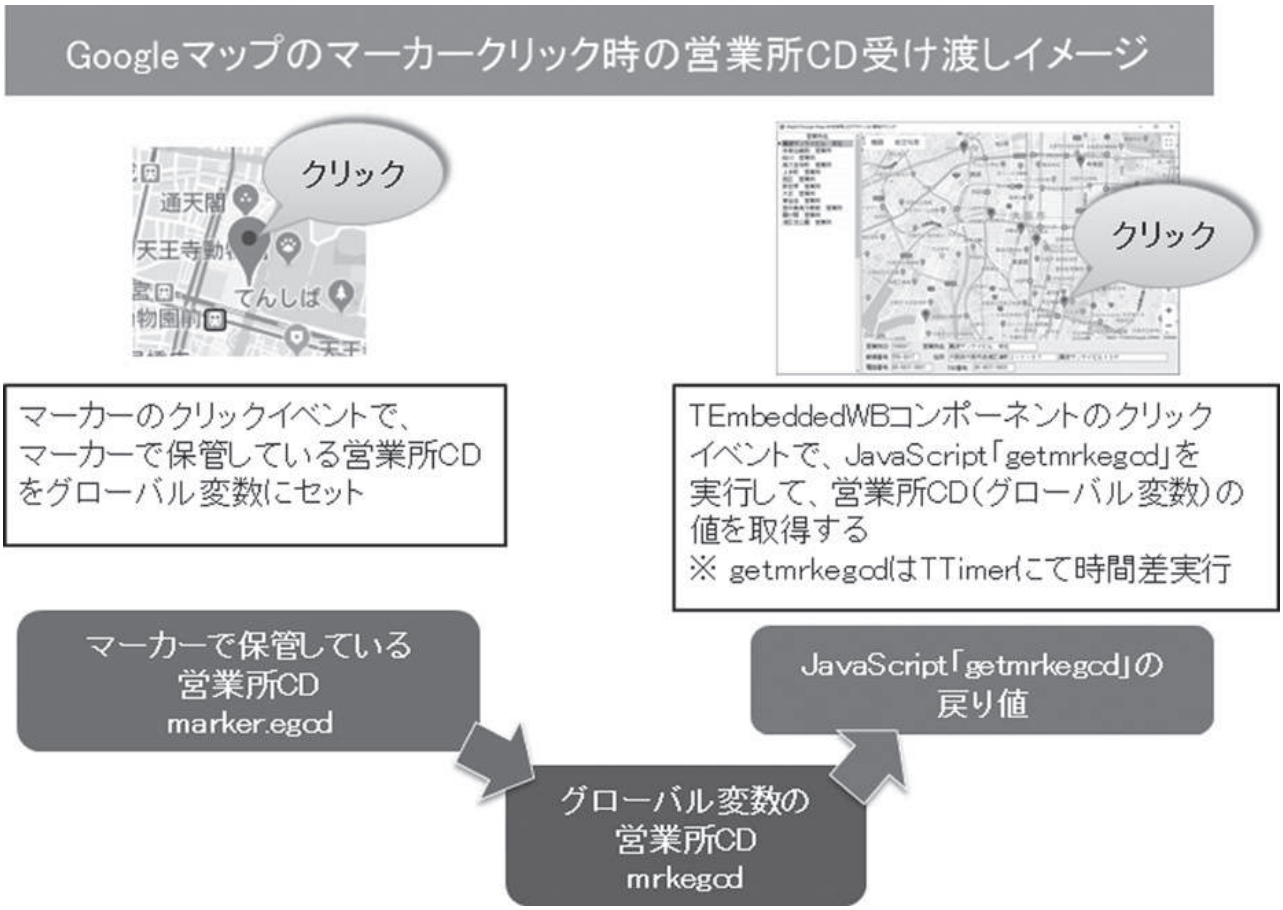


図16　クリックしたマーカーの情報を取得する③

ソース11　Googleマップ表示用HTML①JavaScriptの追加

```
GMapHTML_1
   = 【ソース7】と同じ ～

   // 営業所CD (mrkegcd) の値を取得する関数
   + '      function getmrkegcd(){'
   + '        return (mrkegcd);'
   + '      }'

  // 経度と緯度をMapの中心にする関数
  + '      function moveMapCenter(lat, lng){'
  + '        var latlng = new google.maps.LatLng(lat, lng);'
  + '        map.setCenter(latlng);'
  + '        map.setZoom(18);'
  + '      }'

   + '      window.onload = function() {';
```

変更点①
引数で受け取った緯度と経度を
Googleマップの中心にして、拡大
表示するJavaScript

緯度と経度を中心にする

拡大表示

ソース12　TDBGridコンポーネントのOnClickイベント

```
procedure TfrmSample.dbgListCellClick(Column: TColumn);
begin
  wbMap.ExecScript('moveMapCenter('
                   + qryList.FieldByName('SPLATI').AsString + ','
                   + qryList.FieldByName('SPLNGI').AsString + ')', 'JavaScript');
end;
```

【ソース11】で実装したmoveMapCenterを実行

図17　Googleマップの移動と拡大①

吉原 泰介

Delphi/400 上級

株式会社ミガロ.

RAD事業部 技術支援課

[Delphi/400]

RAD Serverを使った新しい多層アプリケーション構築

略歴
1978 年 3 月 26 日生まれ
2001 年 3 月 龍谷大学 法学部卒業
2005 年 7 月 株式会社ミガロ. 入社
2005 年 7 月 システム事業部配属
2007 年 4 月 RAD 事業部配属

現在の仕事内容
Delphi/400 を中心に製品試験および月 100 件に及ぶ問い合せサポートやセミナー講師などを担当している。

1.はじめに

ここ数年で Delphi/400 によるモバイルアプリケーションの業務開発も多くなってきた。そうしたモバイルアプリケーション開発において、IBM i を利用するために重要となるのが中間サーバーを使った多層構成の仕組みである。【図1】

Delphi/400 でこうした多層アプリケーションを開発する場合、サーバーもクライアントも Delphi/400 だけですべて開発することができる強みがある。この中間サーバーのアプリケーション開発では、DataSnap Server という技術が使われているが、Delphi/400 の最新版である 10.2 Tokyo では、新しく RAD Server というサーバー構築技術が利用できるようになっている。

本稿ではこの新しい RAD Server についての特徴や実装方法、そして RAD Server を活用した他言語アプリケーションとの連携テクニックなどの応用について検証した内容を説明する。

なお本稿では RAD Server が利用できる Delphi/400　10.2Tokyo の環境を前提としている。

2.RAD Serverの特徴

新しく利用できるようになった RAD Server について、主な特徴を確認する。

まず RAD Server は、DataSnap Server と同様に、多層アプリケーションの中間サーバー部分を実装する技術である。多層アプリケーションとは、複数層で構成されたアプリケーションのことで、たとえば Web アプリケーションやモバイルアプリケーションで IBM i にアクセスする場合には、中間層のサーバーを経由する 3 層構成となる。この中間サーバーのアプリケーションを実装できるのが DataSnap Server や RAD Server である。

ではこの 2 つの技術にどのような特徴の違いがあるかをまとめてみる。

まず、従来のバージョンでも使用可能な DataSnap Server は、次のような特徴を持つ。

● DataSnap Server
・多層アプリケーションの開発を可能にする SDK
・サーバー機能はプログラムで開発する必要がある
・開発での実装となるため、プログラムの自由度が高い
・TCP/IP、HTTP（S）、REST、JSON、COM などの標準技術をサポート

次に RAD Server は次のような特徴を持つ。

● RAD Server
・多層アプリケーションの REST API を公開するサーバー
・サーバーで必要となる高度な機能がいくつも提供されている
・ユーザー管理機能、認証機能、分析機能などの標準機能を豊富に搭載

図1　中間サーバーを使った多層構成例

図2　DataSnap ServerとRAD Serverの違い

	DataSnap Server	RAD Server
機能開発	全て開発で実装が必要	必要な部分のみ開発
標準通信	TCPIP/HTTP(S)	HTTP(S) （REST形式固定）
DBエンジン	FireDAC、 dbExpress	FireDAC
モバイル対応機能	開発が必要	Push通知、デバイス認証等が標準機能
管理ツール	開発が必要	標準で付属（分析も可能）
ライセンス	開発ライセンスに含まれる（Enterprise以上）	開発ライセンスに1サイトライセンス付属（10. 2 Tokyo Enterprise以降）

図3　RAD Serverを使った多層アプリケーションの実装例

・HTTP（S）、REST、JSON などの標準技術をサポート

細かい比較については【図 2】にまとめている。

比較するとわかるが、最も大きな違いは DataSnap Server はサーバーアプリケーションを細かく自由に開発することを目的とした技術で、RAD Server は大枠が完成されたサーバーアプリケーションを、カスタマイズをして使う技術ということである。

RAD Server は標準の機能がいろいろ揃っている分、DataSnap Server と違ってアプリケーションの仕組みは REST 形式で固定となっている。REST とは Representational State Transfer の略で、そのサービスの URL が持つメソッドにアクセスすることでデータの送受信をステートレスで行う技術である。汎用性が高く、Web やモバイルのアプリケーションで広く使用されている。そのため、Delphi/400 のアプリケーションだけに限定されずさまざまなアプリケーションから活用することもできる（これについては後述する）。

どちらが優れているかは開発するアプリケーションの要件によっても異なるが、シンプルな機能の中間サーバーであれば RAD Server の標準機能が強みを発揮できる。

この 2 つの技術は、中間サーバーで担っている役割は似ているが、実際の実装手順は異なる部分も多い。そのため、次章では新しい RAD Server の基本的な実装手順について確認していく。

3.RAD Serverの実装手順

RAD Server で構築するアプリケーションは、DataSnap Server と同様に、サーバーアプリケーションとそれを利用するクライアントアプリケーションの 2 種類で構成される。それぞれの実装手順を 2 つのステップに分けて説明していく。

・3-1. サーバーアプリケーションの構築
・3-2. クライアントアプリケーションの構築

なお本稿では、【図 3】に示すように PC やモバイルのクライアントアプリケーションから IBM i のデータを取得する基本的な多層アプリケーションを題材とする。

3-1.サーバーアプリケーション構築手順

初めに中間サーバーに実装するアプリケーションを構築する。構築は RAD Server にウィザードが用意されているため、それほど難しいものではない。

【手順①プロジェクトの作成】

新規作成より、RAD Server（EMS）パッケージを選択してウィザードを起動する。【図 4】

EMS という名称は、RAD Server 関連の機能を意味する。

RAD Server が当初 EMS Server という製品名であったため、機能やコンポーネント名で使われていることが多い。

【手順②リソースの指定】

次にウィザードに従って設定を進めることになるが、【図 5】のように「リソースを含むパッケージを作成する」を選択して、そのリソース名を任意で命名する。

RAD Server は REST 形式になるため、URL でアクセスする際に利用する機能をリソースとして実装する。ここで設定しておくと、指定したリソース名のソースが作成される。たとえば【図 5】のように「CUST」というリソース名を指定すると、実行時に下記のような REST サービスとして呼び出すことができる。

http（s）:// サーバー /CUST

【手順③機能の指定】

②で指定したリソースに対して、どういった機能のテンプレートを作成するかを【図 6】で設定する。デフォルトは基本機能としてデータを取得する Get、GetItem が選択されているが、Put、PutItem、DeleteItem といった更新系の機能も用意されている。なお名称に Item と付く機能は、パラメータを渡して処理できることを意味している。

【手順④自動生成されるソース】

ここまでのウィザード操作が完了すると、【図 7】のように中間サーバーアプリケーションのソース一式が自動生成される。

次の手順からはこのソースに必要なプログラムを実装していく。

【手順⑤コンポーネントの配置と設定】

自動生成されたリソースに機能を実装するにあたって、まずは必要なコンポーネントを配置する。RAD Server で使えるデータベースエンジンは最新の FireDAC のみに限定されている。そのため、ここではデータ取得（Get）の仕組みを作ることを前提に FireDAC の主要なコンポーネントを配置する。【図 8】

・TFDConnection
・TFDPhysCO400DriverLink
・TFDTable
・TFDSchemaAdapter
・TFDStanStorageJSONLink
・TFDGUIxWaitCursor

TFDConnection 〜 TFDTable および TFDGUIxWaitCursor については、一般的な FireDAC のアプリケーションで実装する内容である。本稿では FireDAC の詳しい使い方は割愛するが、【図 9】のように TFDConnection や TFDTable を設定する。

RAD Server の構築でポイントになるのは TFDSchemaAdapter と TFDStanStorageJSONLink コンポーネントである。これは RAD Server が REST 形式として動作することが前提となるため、TFDTable で取得したデータを JSON 形式に変換する機能の実装に使用する。【図 9】で TFDTable の SchemaAdapter プロパティに TFDSchemaAdapter を設定しているのは、そのためである。

【手順⑥データ取得の機能を実装】

次に RAD Server がリクエストに応じてデータを返す機能をプログラムで実装する。ウィザードでリソースの Get 機能を指定しておけば、自動的に Get メソッドが作成されているので、ここで必要な処理だけを数行コーディングする。【ソース 1】

図4　プロジェクトの作成

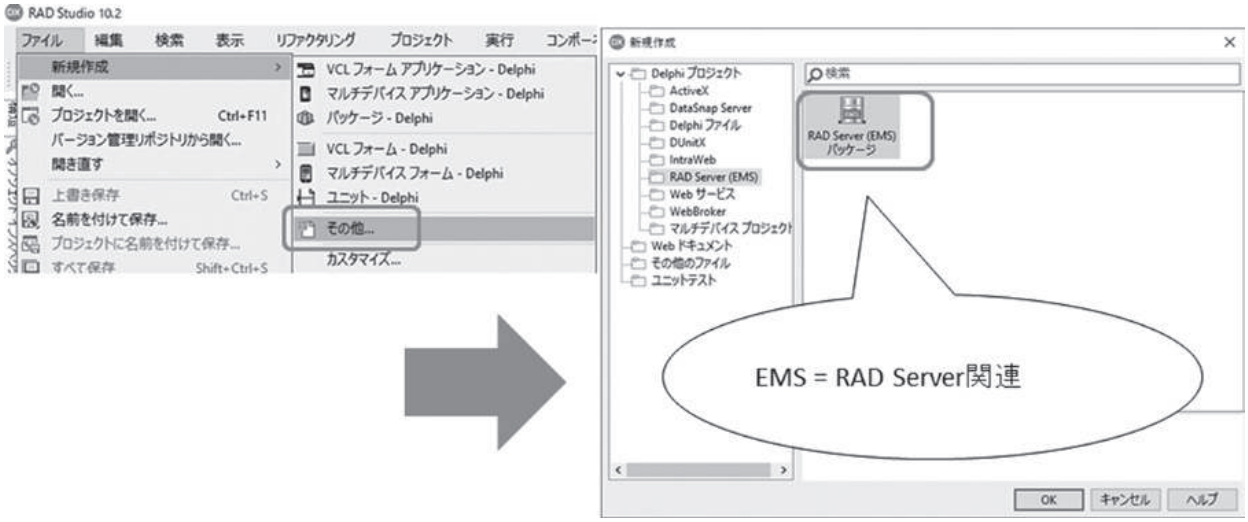


図5　リソースの指定

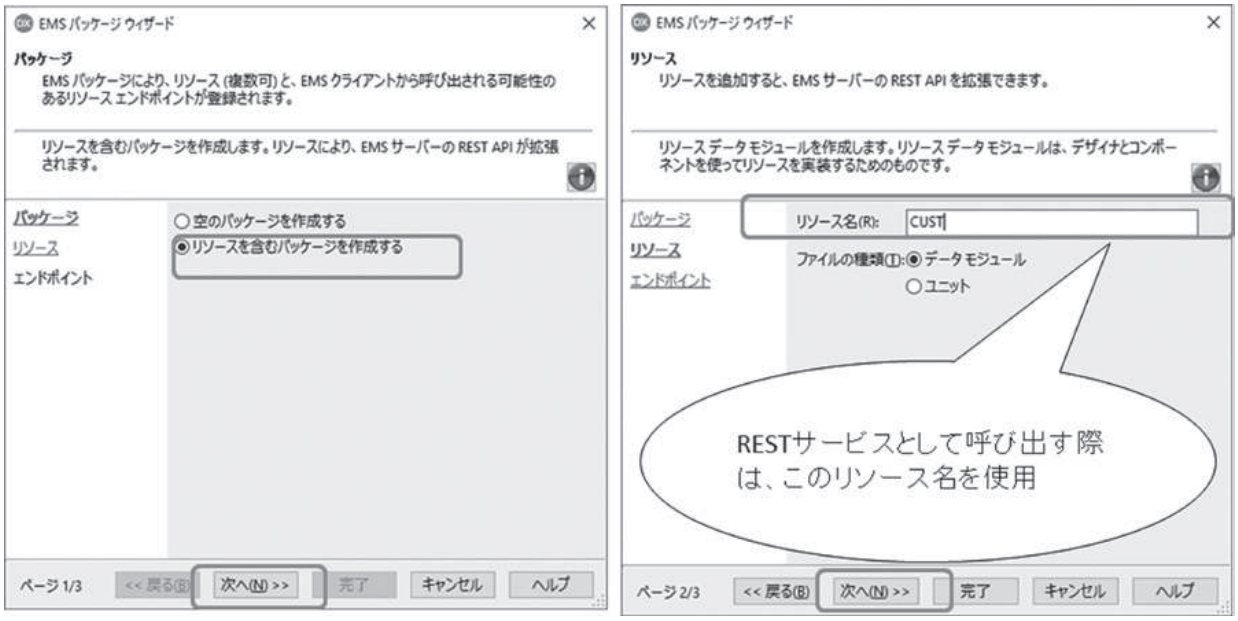
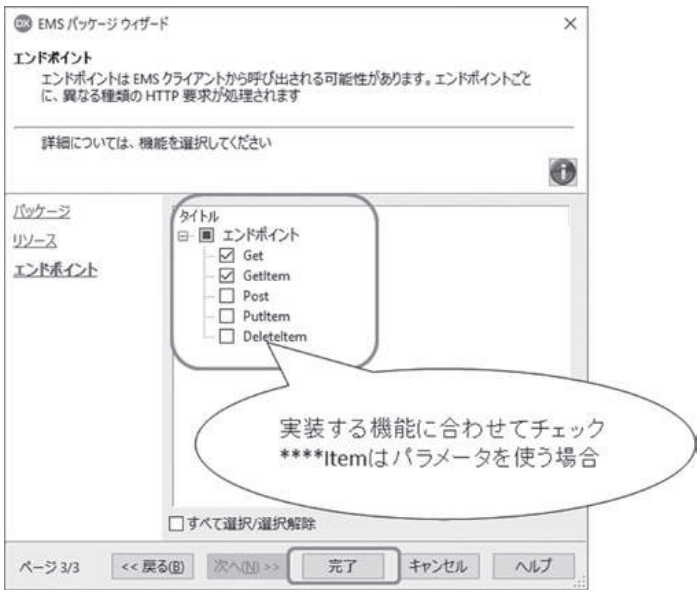


図6　機能の指定



処理内容としてはTFDTableでデータを開き、そのデータをTFDSchemaAdapterでJSON形式のストリームに変換するだけである。この時、JSON形式を指定するためにTFDStanStorageJSONLinkが必要となっている。また図6でItemを使った機能を実装する場合は、パラメータであるARequestを使ってARequest.Params.Values['XXXX']という形で簡単に利用することができる。

プログラムの実装はこれで一通り完了である。

【手順⑦コンパイルと実行】

最後に作成したRAD Serverの中間サーバーアプリケーションをコンパイルして実行する。ただし初回のコンパイル時にはコンポーネントのパッケージ開発時と同様（RAD Serverもパッケージ方式）にパッケージの参照確認やInterBaseの設定確認のダイアログが表示されるので応答が必要となる。【図10】

InterBaseは意識して使用するわけではないが、RAD Serverが標準で備える管理分析機能等で内部的に使用している。なおInterBaseのライセンスはRAD Serverに含まれているので別途購入の必要はない。

コンパイルが完了してアプリケーションを実行すると、【図11】のような画面が起動して完成である。この画面では、自動的にアクセスログが出力されるようになっており、クライアントからリクエストがあるとこの画面にログが表示される。

この起動画面はあくまでRAD Serverに用意された標準のexeアプリケーションである。DataSnap Serverと違い、作成したプログラム自体がサーバーとして起動しているわけではない。

コンパイルしたリソースはbplとして作成されてRAD Serverに読み込まれている。運用環境の構築やリソースの配布をする際には、エンバカデロ社の下記オンラインヘルプに詳細情報が公開されているため、参考いただきたい。

http://docwiki.embarcadero.com/RADStudio/Tokyo/ja/

3-2.クライアントアプリケーション構築手順

次に、前節で作成したRAD Serverアプリケーションに接続してデータを取得するクライアントアプリケーションを構築する。

【手順①プロジェクトの作成】

新規作成よりマルチデバイスアプリケーションを選択して、FireMonkeyの新規プロジェクトを作成する。【図12】

今回はモバイルでの動作を見るためにFireMonkeyで作成しているが、デスクトップアプリケーションとしてVCLで作成する場合も実装内容は同じである。

【手順②コンポーネントの配置・設定】

新規作成したアプリケーションに必要なコンポーネントを配置する。【図13】

・TEMSProvider
・TEMSFireDACClient
・TFDSchemaAdapter
・TFDTableAdapter
・TFDMemTable
・TFDGUIxWaitCursor

このアプリケーションでもFireDACコンポーネントを使用するが、直接IBM iに接続するわけではなく、前節で作成したRAD Serverアプリケーションに接続してデータを取得する仕組みとなる。そのため、TFDConnectionではなく、TEMSProviderおよびTEMSFireDACClientの専用コンポーネントを使用する（EMSはRAD Server関連を意味する）。

またRAD Serverより取得するデータはJSON形式で送られてくるため、それをDelphi/400のデータセット形式に変換して戻すためにTFDSchemaAdapterを使用する。それぞれのプロパティ設定は【図14】に示す。

ポイントとしてはTEMSProviderで接続するRAD ServerのIPやポートを指定し、TEMSFireDACClientで呼び出すリソース情報を設定する点である。リソース名はRAD Serverのウィザードで指定したものを設定する。

なおデータの画面表示はFireMonkeyなので、【図15】のようにライブバインディングを使って実装できる。VCLの場合はTDBGridを使うこともできる。

【手順③RAD Serverの呼び出し機能の実装】

最後にRAD Serverからデータ取得のGet機能をRESTで呼び出す処理をプログラムで実装する。RAD Serverはほとんどのアクセス制御を前手順のコンポーネントで処理してくれるため、コードは1行で済む。【ソース2】

これでクライアントアプリケーションのデータアクセスのプログラムが完成したことになる。RAD ServerとDataSnap Serverの開発は似ているが、RAD Serverでは標準機能がコンポーネントなどで備わっている分、簡単に実装ができる部分も多い。

【手順④コンパイル・実行】

完成したプログラムを目的のプラットフォームを指定してコンパイルして実行すると、WindowsやiOS、あるいはAndroid向けにアプリケーションを利用することができる。【図16】

全体的な仕組みは冒頭で説明した【図3】のとおりである。

4.RAD Serverの管理分析機能

前章でRAD Serverの基本的な実装手順を説明したが、RAD Serverが持つ機能について少しだけ補足を加える。

3-1.で実装したRAD Serverのアプリケーションは、起動画面でアクセスログが確認できることを説明したが、利用できる機能はそれだけではない。

RAD Serverは管理機能を標準で備えているため、たとえばサーバーに対するアクセス分析などの機能は開発しなくとも自動で実装されている。使い方としては起動画面メニューの「コンソールを開く」ボタンをクリックする。【図17】

そうすると【図18】のようなコンソールのログイン画面がブラウザで起動する。このログインはユーザー名がconsoleuser、パスワードがconsolepassがデフォルトになっている。

図7　自動生成されるソース

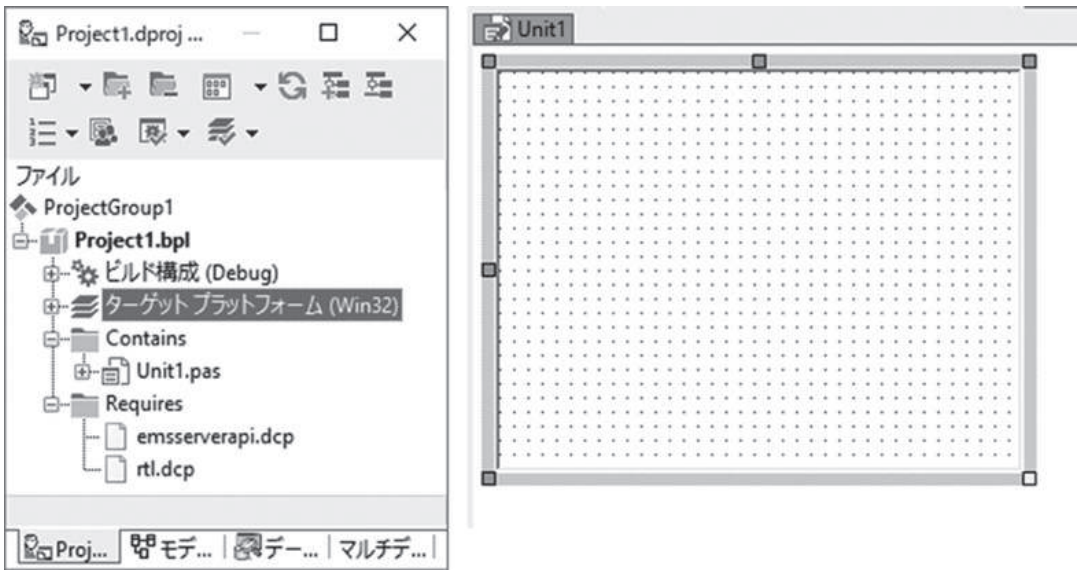


図8　サーバー機能のコンポーネント配置

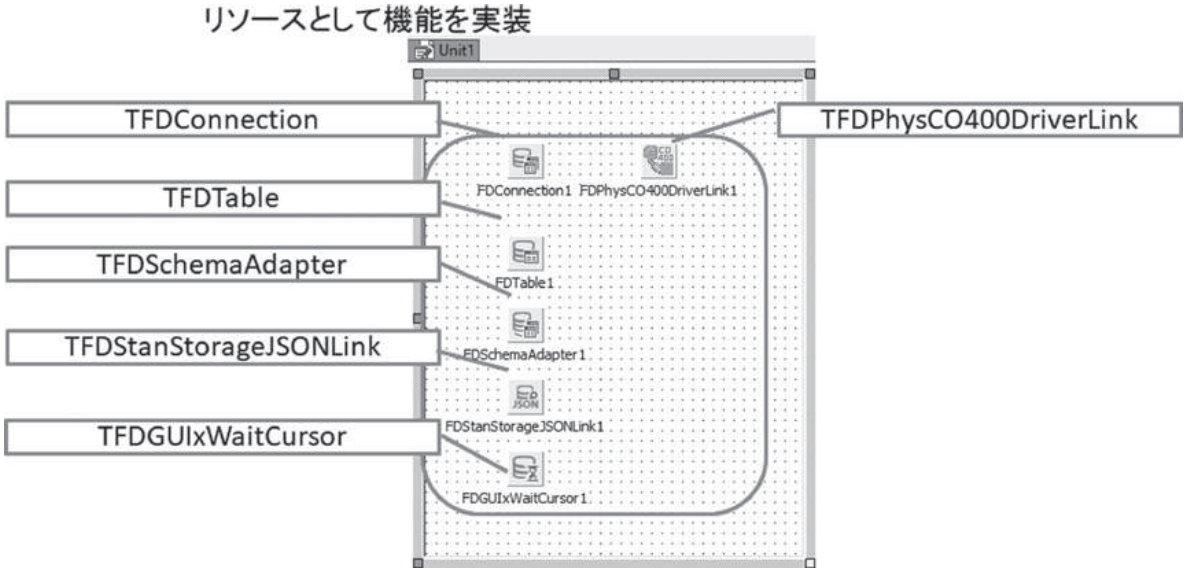


図9　コンポーネントの設定

このアカウント情報はRAD Server上の下記iniファイルに設定されており、テキスト編集で変更が可能である。

C:¥Users¥Public¥Documents¥Embarcadero¥EMS¥emsserver.ini

ログインすると、【図19】のようにRAD Serverにアクセスするクライアントの状況や作成したRESTリソースの使用頻度について、グラフを使ったビジュアル分析が行える。これによってどの機能がよく使われているか、どんな時間帯に処理が集中しているか、などの把握も容易である。

5.補足:他言語からの利用

ここまでRAD Serverを使った基本的なアプリケーションの実装内容や機能についてまとめてきたが、最後にRAD Serverをさらに活用するための使い方を考察していく。

RAD ServerがREST形式になることは冒頭で説明したが、このREST形式はここまでの実装でも説明したとおり、JSONを利用できる。このREST/JSONの組み合わせは、非常によく使用される通信方式であるため、実はDelphi/400以外の他言語アプリケーションでも利用可能である。

本稿では他言語の例として、Delphi言語と同じ開発元であるエンバカデロ・テクノロジーズ社が提供するSencha Architectを題材としてRAD Serverの利用を説明する。Sencha Architectとは、表、グラフ、ツリーなどの多彩なコンポーネントをドラッグ&ドロップして簡単にHTML5対応のWebアプリケーションを構築できる開発ツールである。【図20】

このSencha ArchitectはREST/JSONデータと連携してデータを扱うこともできるため、RAD Serverとも容易に連携ができる。

● RAD Serverの実装調整

3章で実装したGET機能のJSONはDelphi/400のコンポーネントに特化した作り方となっているため、もっと一般的なJSON形式に調整する。【ソース3】

● Sencha Architectからの利用

本稿ではSencha Architectの詳しい開発手順については割愛し、ポイントになる部分を説明する。

Sencha ArchitectにもDelphi/400と同じようなコンポーネント（部品）が用意されているため、RAD Serverから取得したデータを表示するためにGrid Panelというコンポーネントをプログラムに配置する。【図21】

このGrid PanelはDelphi/400のTDBGridと扱いが似ている。

次にGrid Panelのデータ参照元を指定する。

これはGrid PanelにGrid Builderというウィザード機能が用意されており、【図22】のようにJSON WebService形式でRAD ServerのURLが参照元となるように設定する。これによってフィールド情報なども自動で取り込んで定義できる。

このデータをGrid Panelの参照元にセットすれば、Delphi/400と同じようにSencha Architect上でRAD Serverからの情報を表示することができる。【図23】

このプログラムをSencha Architectでコンパイルして実行するとブラウザでWebアプリケーションとして実行され、RAD Serverの情報を表示することが可能である。【図24】

Sencha Architectの細かい設定手順は省略しているが、RAD ServerのREST/JSON形式は非常に汎用性が高く、別の言語でも簡単に扱えることがわかる。

もちろんSencha Architectに限らず、他の言語でもREST/JSONが扱えれば同じように使用が可能である。これを理解していればRAD Serverをより広い範囲で活用することができる。

6.おわりに

本稿では、新しい多層アプリケーションの構築技術としてRAD Serverの特徴や実装手順について説明した。従来のDataSnap Serverと違って、RAD Serverは開発形式がある程度限定されているものの、あらかじめ専用コンポーネントやサーバーの管理機能を備えているため、効率よくアプリケーションを実装するには効果的である。

また5章で述べたように、このRESTサービスは非常に汎用性が高く、Delphi/400以外の言語でも利用することができる。

今後、クラウドやソリューションを含めて多様化していくシステム連携の中心となるサーバーとして活用が期待できる。

M

図10　初回コンパイル時の応答

ソース1　データ取得の機能を実装

```
procedure TCUSTResource1.Get(const AContext: TEndpointContext;
const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
var
  oStr: TMemoryStream;
begin
  oStr := TMemoryStream.Create;

  // クエリの実行結果をスキーマアダプタから
  // メモリストリーム経由で返す
  FDTable1.Open;
  FDSchemaAdapter1.SaveToStream(oStr,TFDStorageFormat.sfJSON);
  AResponse.Body.SetStream(oStr,'application/json', True);
  FDTable1.Close;
end;
```

図11　RAD Serverの起動

図12　クライアントアプリケーションの作成

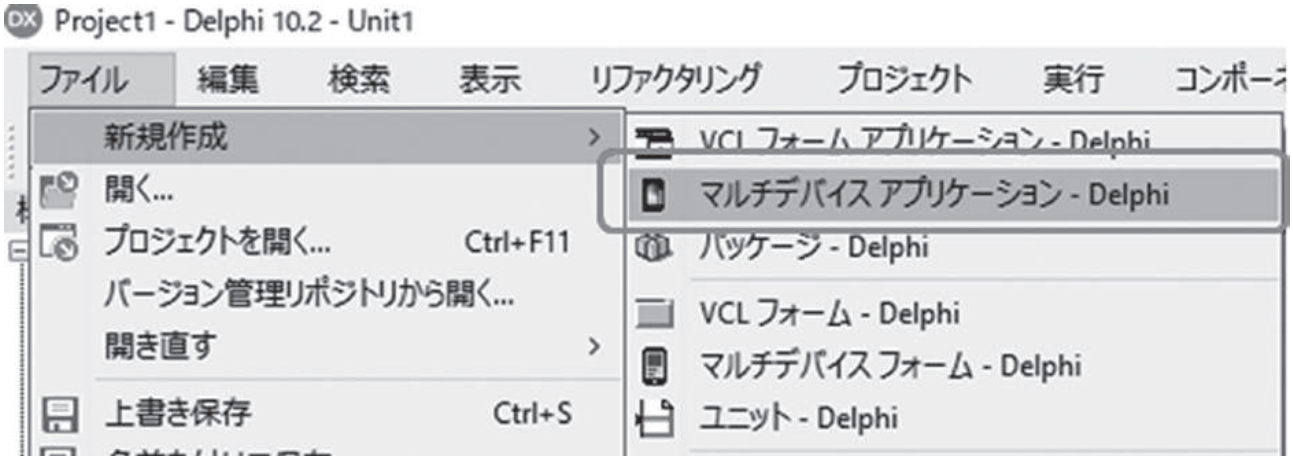


図13　クライアント機能のコンポーネント配置

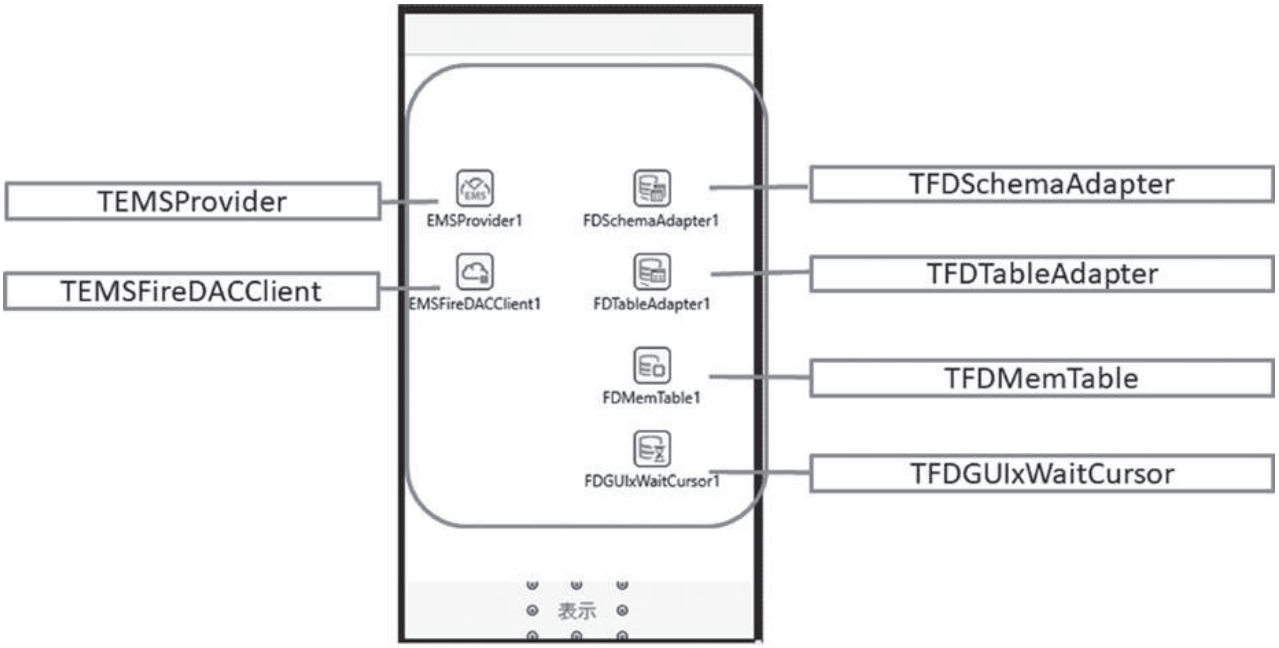


図14　コンポーネントの設定

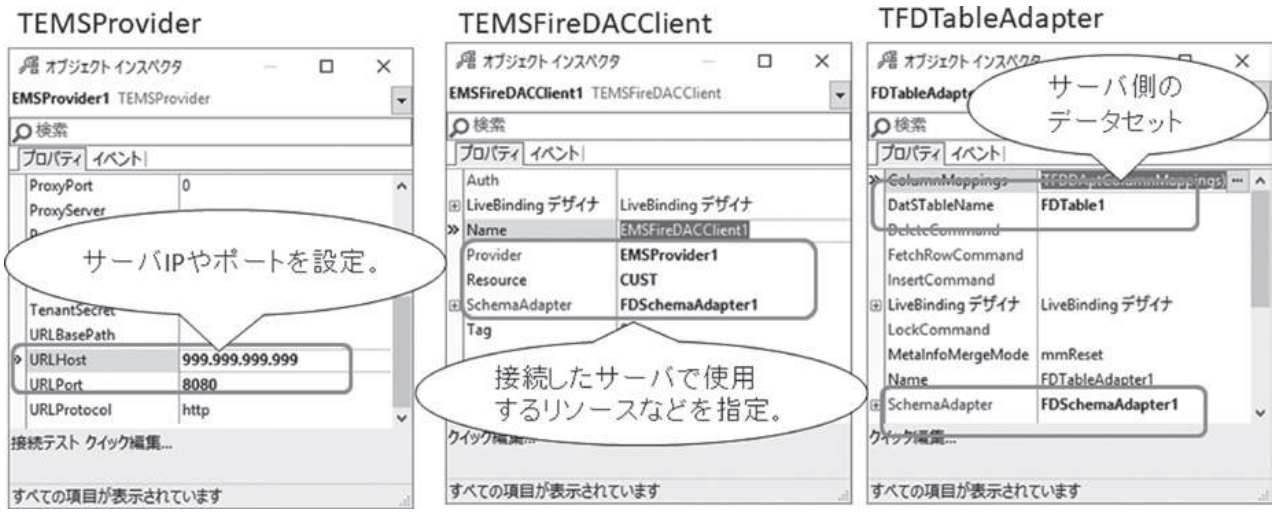
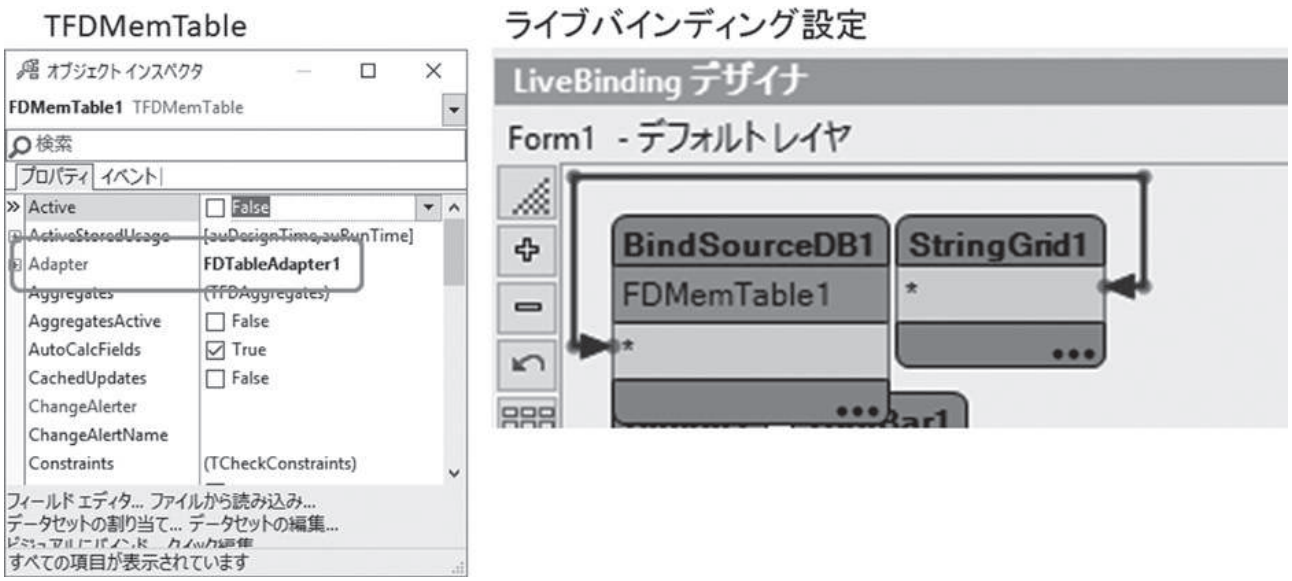


図15　データの表示設定



ソース2　データ取得の機能を実装

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  //設定しているリソースのGetDataを呼び出す
  EMSFireDACClient1.GetData;
end;
```

図16　プラットフォーム別にコンパイル・実行

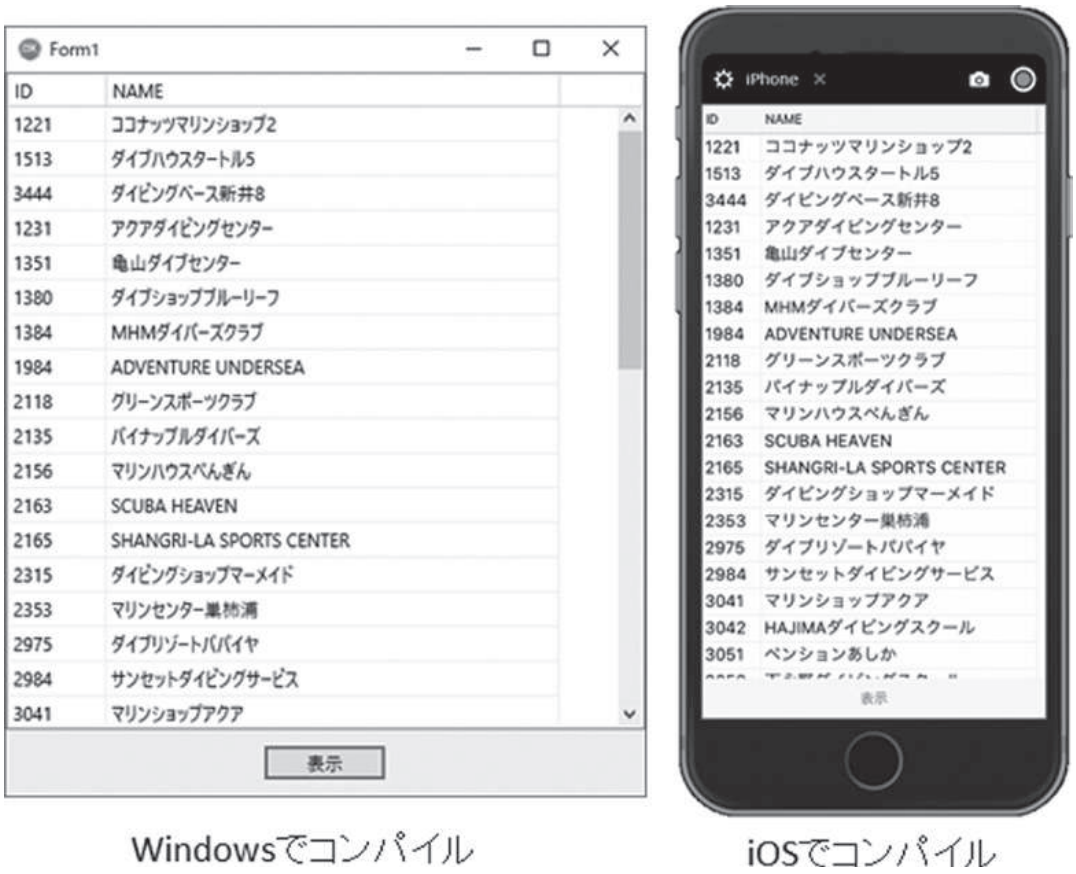


図17　コンソールの起動

図18　コンソールのログイン

図19　RAD Serverによる分析機能

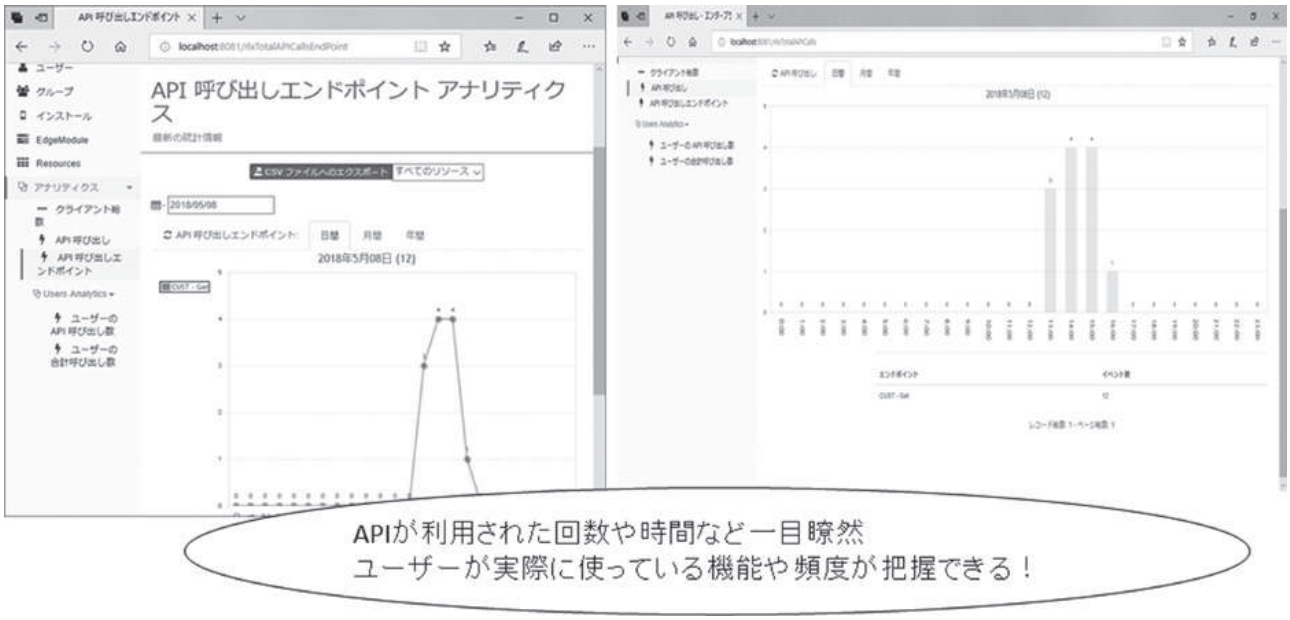


図20　WebアプリケーションにSencha Architect

ソース3　一般的なJSON形式でRESTの結果を戻す

```
procedure TCUSTResource1.Get(const AContext: TEndpointContext;
const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
var
  JSONResponse: TJSONObject;
  JSONArray: TJSONArray;
  JSONRecord: TJSONObject;
  aField: TField;
  count: Integer;
begin
  JSONResponse := TJSONObject.Create;
  JSONArray    := TJSONArray.Create;
  count := 0;
 // データセットの結果を1件づつ取得してJSONArrayに追加する
  FDTable1.Open;
  while (not(FDTable1.Eof)) do
  begin
    JSONRecord := TJSONObject.Create;

    for aField in FDTable1.Fields do
    begin
      JSONRecord.AddPair(LowerCase(aField.FieldName),aField.AsString);
    end;

    JSONArray.Add(JSONRecord);
    FDTable1.Next;
    inc(count);
  end;

  AResponse.Body.SetValue(JSONArray,True);
end;
```

図21　Sencha Architectの開発（GridPanel）

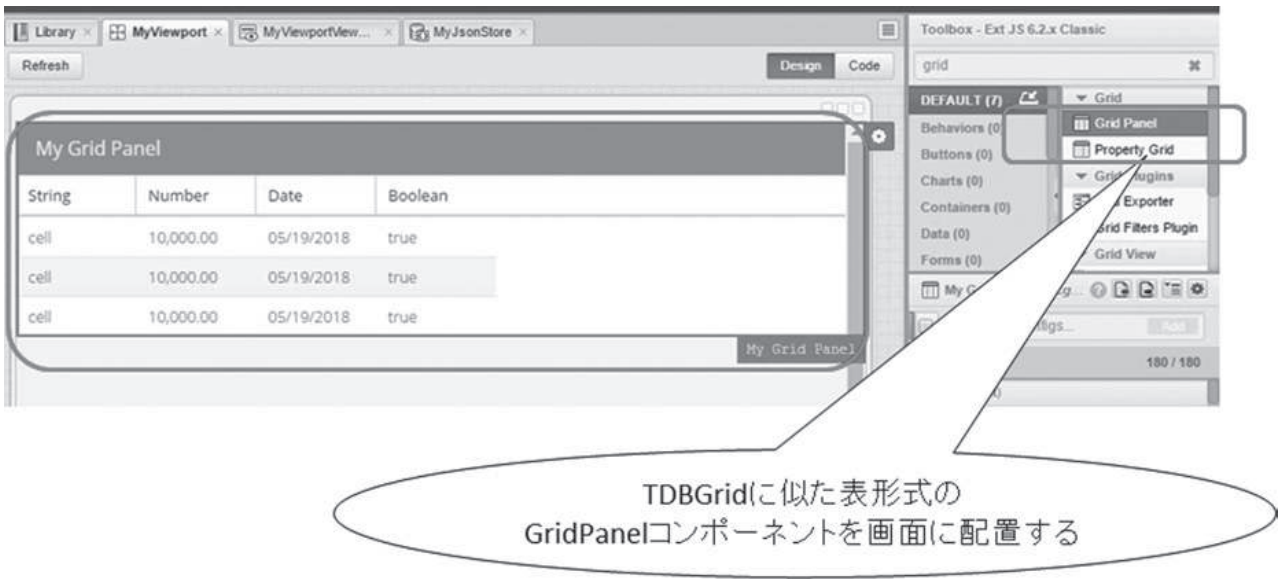


図22　Grid Panelのデータ参照設定

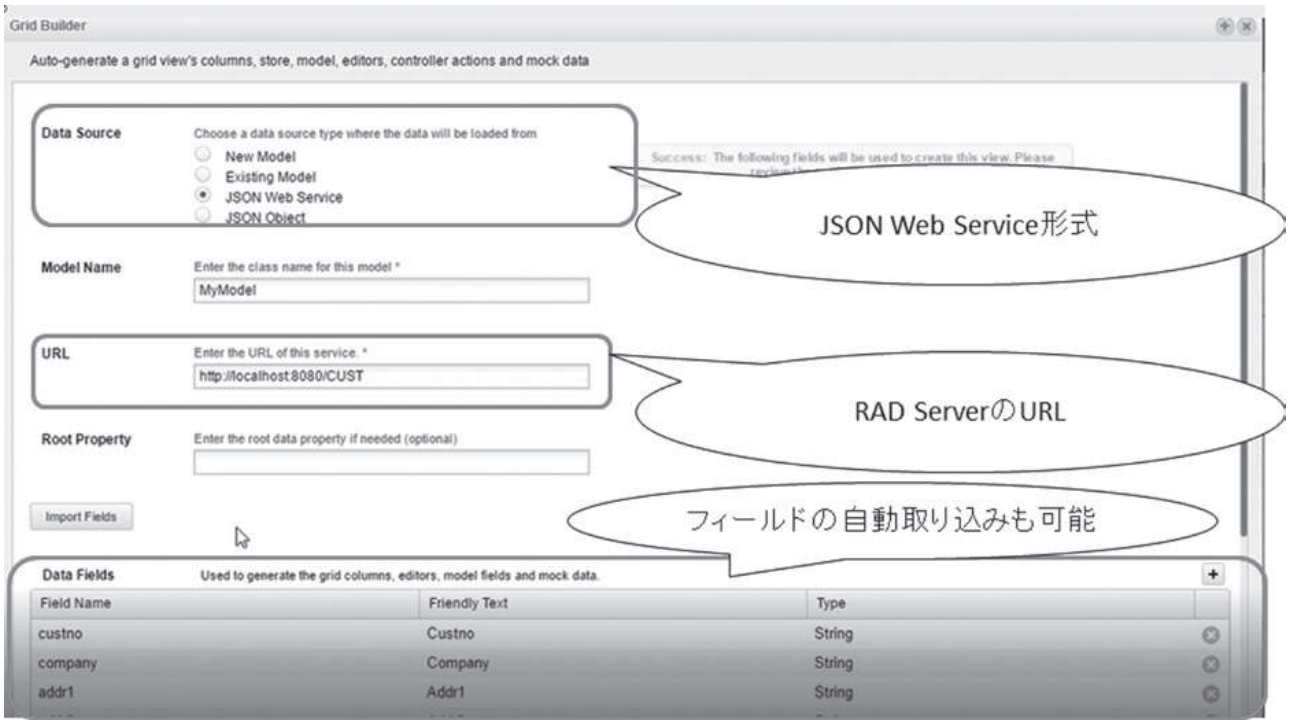


図23　Grid Panelへの表示

図24　Sencha ArchitectアプリからRAD Serverを利用

吉原 泰介 / 國元 祐二

株式会社ミガロ.

RAD事業部 技術支援課

【SmartPad4i】JC/400からSP4iへのマイグレーションノウハウ

略歴 吉原 泰介
1978 年 3 月 26 日生まれ
2001 年 3 月 龍谷大学 法学部卒業
2005 年 7 月 株式会社ミガロ. 入社
2005 年 7 月 システム事業部配属
2007 年 4 月 RAD 事業部配属

現在の仕事内容
Delphi/400 を中心に製品試験および月 100 件に及ぶ問い合せサポートやセミナー講師などを担当している。

略歴 國元 祐二
1979 年 3 月 27 日生まれ
2002 年 3 月 追手門学院大学　文学部アジア文化学科卒業
2010 年 10 月 株式会社ミガロ. 入社
2010 年 10 月 RAD 事業部配属

現在の仕事内容
SmartPad4i(JC/400)、Business4 Mobile、Valence の製品試験やサポート業務、導入支援などを行っている。

1.はじめに

最近の Web アプリケーションは C/S アプリケーションと近い機能や操作性を持ち、基幹システムの一部として利用されることも多くなっている。C/S アプリケーションと比べて便利な点は、PC にあらかじめ搭載された Web ブラウザで動作するため、運用環境の構築が非常に容易な点である。

JC/400 は、そうした Web アプリケーションを IBM i の RPG を中心に開発できるツールとして実績がある。

しかし Web アプリケーションもここ数年で大きく環境が変わってきている。それは Web ブラウザが多様化したことである。

以前は Windows にインストールされている Internet Explorer（以下、IE）が標準 Web ブラウザとして使われることが圧倒的に多く、Web アプリケーションも IE の動作を基準としていた。しかし最近は、Google Chrome（以下、Chrome）や Safari、Firefox などさまざまな Web ブラウザが使われるようになっている。【図 1】

これはモバイルの普及が大きく影響しており、単純に Windows を標準とした Web ブラウザよりも、モバイルを含めたさまざまな OS に対応した高機能な Web ブラウザが標準となってきている。

そのため、Web アプリケーションも各種 Web ブラウザに対応した動作を求められることが多くなってきた。いわゆるクロスブラウザ対応である。

JC/400 の Web アプリケーションの動作環境は、従来の IE に限定されるが、こうした背景に合わせた Web アプリケーションへの対応方法が用意されている。それは JC/400 の後継である SmartPad4i（以下、SP4i）へのマイグレーションである。

SP4i では前述のクロスブラウザ対応に加え、モバイルでもハイブリッド Web アプリケーションとして使用可能である。

モバイルでは、カメラや GPS などのネイティブ機能も活用することが可能である。もちろん SP4i は JC/400 の後継であるため、仕組みとしては大きく変わらず、ほとんどの部分は SP4i がアーキテクチャの違いを吸収してくれる。そのため、マイグレーションでは JC/400 のプログラムを若干手直しすることで、そのまま移行できる。

本稿では、JC/400 から SP4i へプログラムのマイグレーションを題材に手順やポイントを説明する。

＊マイグレーション対象の SP4i は 2018 年 8 月時点で最新バージョンの SmartPad4iV2.1.8D とする。

2.システム環境･設定の違い

JC/400 から SP4i へのプログラムのマイグレーションを行う前に、環境の違いを把握しておく。RPG を中心としてアプリケーションを開発・実行できる大きな仕組みは SP4i でも変わらないが、サブシステム名やポートなどは製品が異

図1　Webブラウザの使用率

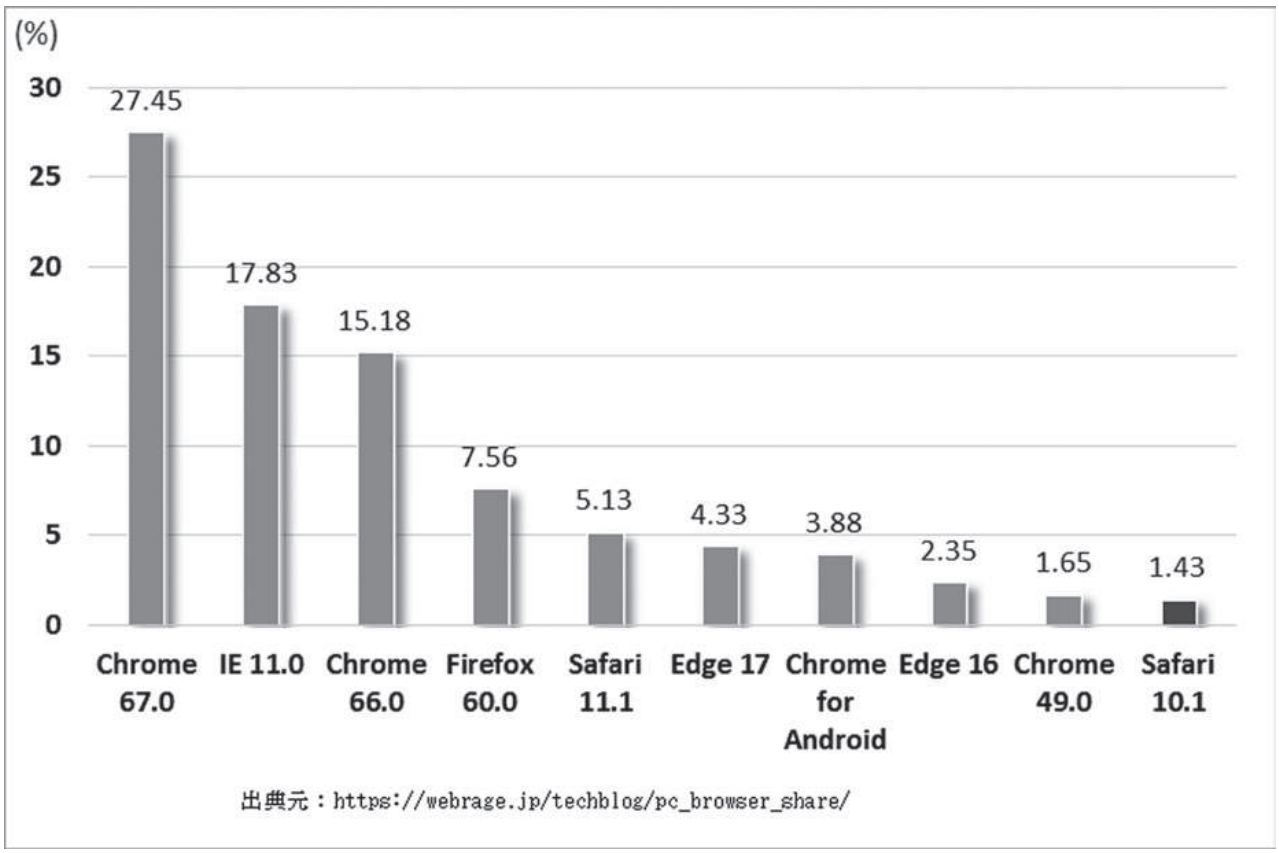
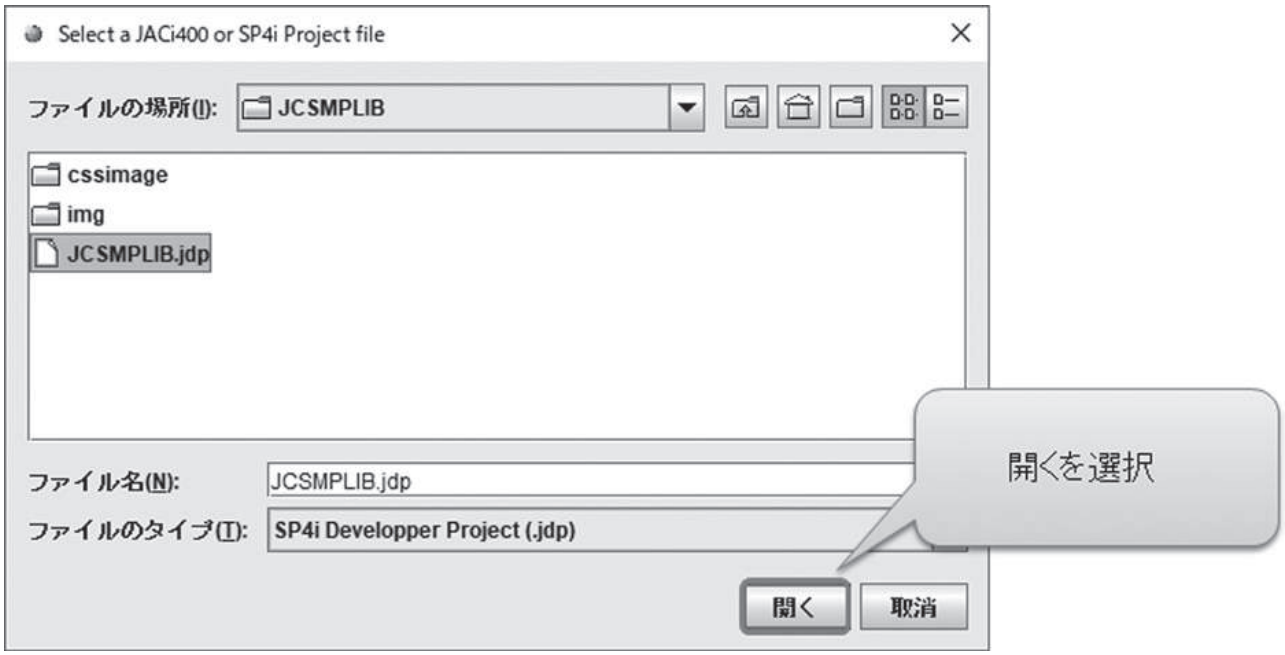


図2　SP4iサブシステム

図3　プロジェクトを読み込む



なるため、刷新されている。(＊SP4iの初期バージョンではJC/400と共通の部分もある)

まず、サブシステムはログインメニューとアプリケーションの2つが稼働しており、次のような違いがある。【図2】

●ログインメニューのサブシステム
CO405JTCP　→　CO406JTCP

●アプリケーションのサブシステム
JACI400　→　SP4I

実際のプログラムでこのサブシステム名の違いを考慮する必要はないが、実行するジョブの確認などで重要になってくるので、新しいサブシステム名を把握しておく必要がある。

また接続するポート番号も19003から19004に変更となっているので、Webサーバー環境やDesigner(後述)の接続設定では注意していただきたい。

これらの新しい環境を前提に、次の章からはプログラムのマイグレーションについて詳しくポイントを説明する。

3.プログラムのマイグレーションポイント

JC/400のプログラムをSP4iのプログラムにマイグレーションするにあたって、プログラムを3つのカテゴリに分けてポイント整理する。

①プロジェクト
- アプリケーション全体の構成や設定

② IBM iプログラム
- 環境設定のCLプログラムやメインとなるRPGプログラム

③画面プログラム
- 画面を構成するHTMLやJavaScriptプログラム

これら3つのカテゴリのプログラムがJC/400のアプリケーションを構成しているので、これらに若干の変更を加えればSP4iのアプリケーションとして使用可能である。

本章では、この3つをカテゴリごとに変更ポイントとしてまとめている。

3-1.プロジェクトのマイグレーション

アプリケーションのプロジェクトには全体のソース構成や設定が保存されている。JC/400では［JC/400 Designer］というツールを使ってプロジェクトを作成したり、IBM iへのRPGプログラムの自動生成を行うが、SP4iでも同様のツール［SmartPad4i Designer］が用意されている。この［SmartPad4i Designer］を使ってJC/400の既存プロジェクトを読み込み、再配布を行う。

詳しい手順は次のとおりである。

SmartPad4i Designerの操作手順

①プロジェクトを読み込む

SmartPad4i Designerの「ファイル」メニューから拡張子がjdpのプロジェクトファイルを選択して読み込む。【図3】

②配布の接続設定を行う

読み込んだプロジェクトの設定はJC/400のままになっているため、接続はキャンセルして、接続設定のポート番号を変更する。ポート番号は前章で述べたとおり19004になる。【図4】

③配布先の設定を行う

SmartPad4i Designerの「オプション」メニューからHTMLファイルのパスとWEBサーバーの配布先のパスを設定する。【図5】

HTMLファイルパスはSP4iの場合、次のようなパスになるので変更が必要である(SmartPad4iの製品パスが含まれる)。

「Webサーバー\htdocs\ja_JP\smartpad4i\html\ライブラリ名」

WEBサーバーへの配布ルートの選択については、JC/400と同じであれば変更の必要はない。

④配布を行う

SmartPad4i Designerの「配布」メニューからIBM iへ配布を行い、既存のRPGを更新する。【図6】

同様に「配布」メニューからWEBサーバーへ配布を行う。【図7】

この作業によって、JC/400で構成されているRPGプログラムをSP4iの形式へ自動的に組み換えることができる。これでプロジェクトの変更は完了である。

3-2.IBM iプログラムのマイグレーション

次に、IBM iのプログラムについてSP4iで変更すべき点を確認する。IBM iのプログラムはCLとRPGの2つで構成されている。

①環境設定用のCLプログラム

CLプログラムでは、実行時にライブラリリストを設定している。JC/400ではJACI400DEV、JACI400というライブラリを使用しているが、SP4iではSP4Iというライブラリに変わるため、この記述を変更する必要がある。【ソース1】

XXXLIBはプロジェクトのライブラリを指す。

②メインのRPGプログラム

RPGプログラムのロジックは変更する必要がなく、定型的な変更作業になる。これはJC/400で用意されているRPG上のAPIやフィールド変数の名前がSP4i用に変わっているために行う作業である。ただし、3-1の作業で基本的には自動で変更が適用されているので、ほとんど変更の必要はない。

プログラムの内容によっては自動変換できない部分も残ってしまうが、コンパイルすると必ずエラーになるため、エラーになった部分を定型的に置き換える作業となる。APIやフィールド変数は数が多いため、ソースの変更例ではなく、リストとして【図8】【図9】【図10】にまとめているので作業時に参考いただきたい。

コンパイルが無事通れば、IBM iプログラム変更は完了である。

3-3.画面プログラムのマイグレーション

画面プログラムはHTMLとJavaScriptで構成される。独自に作り込んでいる画面プログラムは自由度が高いため、本稿ですべての変更点を挙げること

図4　接続設定

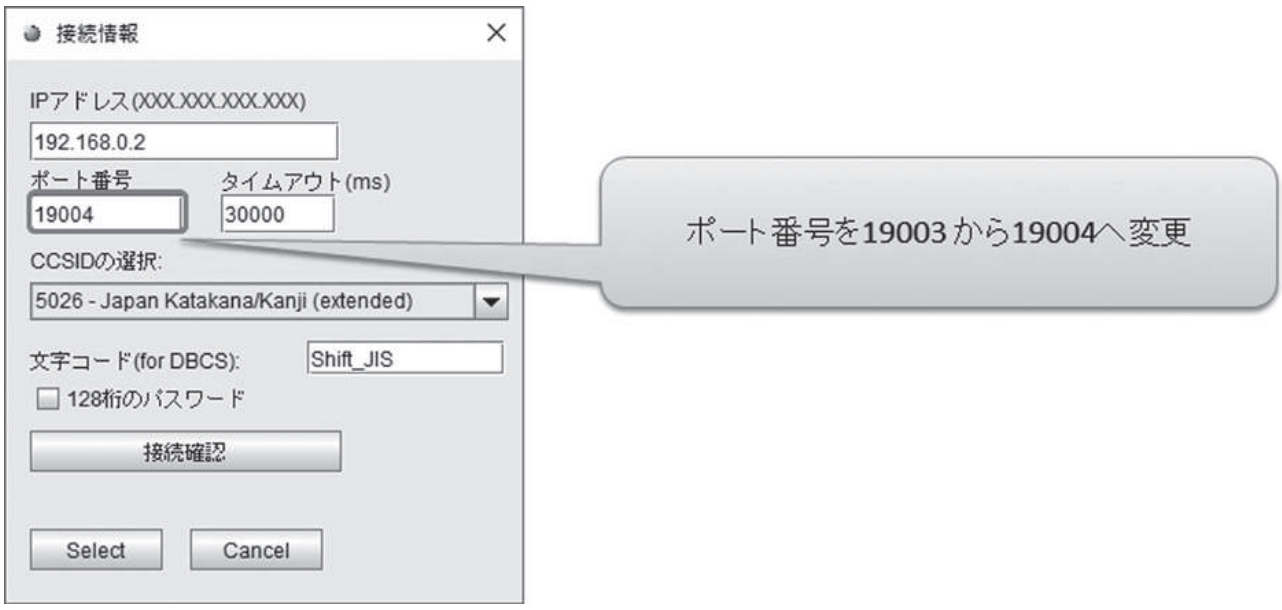
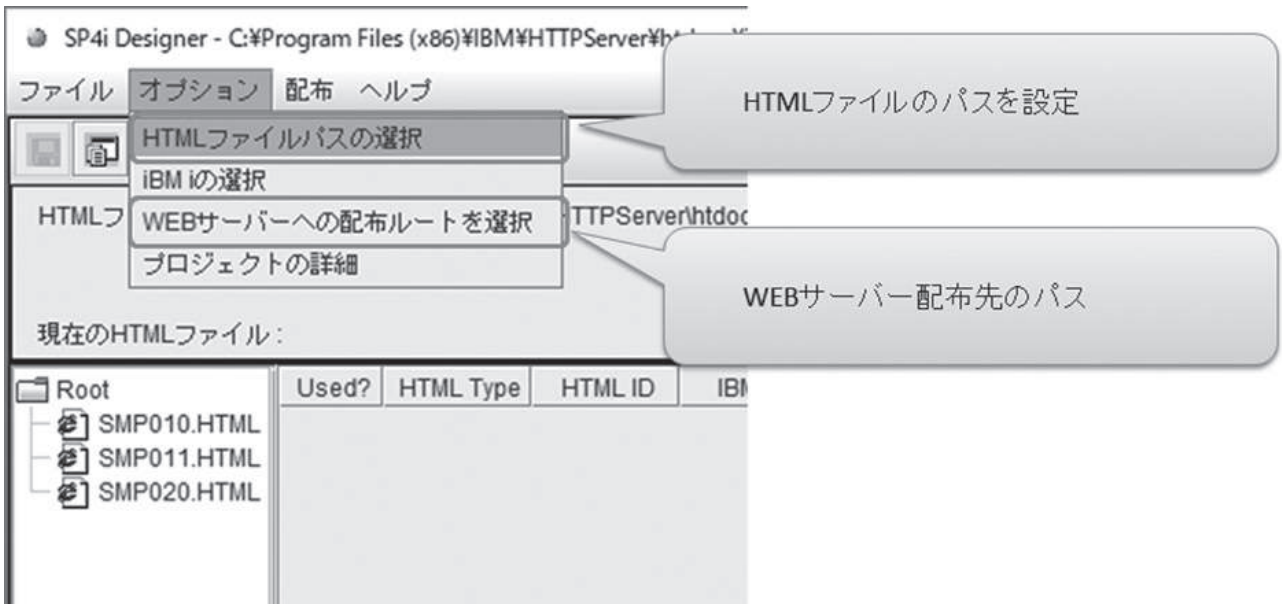


図5　接続設定



はできないが、SP4i に直接関連する関数の使用について説明する。

JC/400 利用時に JavaScript を記述している場合、画面の項目値（要素）を取得する関数として、document.getElementById 関数、document.getElementsByName 関数を使用することが多い。この関数を SP4i で使用する場合は、それぞれ、SP4i.getElementById 関数、SP4i.getElementsByName 関数として変更する必要がある。【ソース 2】【ソース 3】

また画面プログラムについては、基本的に変更を必要としないが、IE に限定した JavaScript の機能（たとえば ActiveX など）を独自に使っている場合は、ほかの Web ブラウザでは機能しないので注意が必要である。

3-1 ～ 3-3 の修正が完了すれば、JC/400 からマイグレーションした SP4i のアプリケーションが完成である。

これにより、今まで IE だけで利用していた Web アプリケーションが、他の Web ブラウザでも実行できるようになる。【図 11】は、IE で実行する JC/400 のアプリケーションの例、【図 12】が SP4i へマイグレーションして、Chrome や Edge で実行した例である。

SP4i で実行すれば IE はもちろん、Windows10 に搭載の新 Web ブラウザ Edge、Chrome、Safari、Firefox 等、企業で使用されるであろう、ほぼすべての Web ブラウザで利用することができる。

4.マイグレーションに伴う拡張性

前章までの内容で基本的なマイグレーションは完了である。SP4i では IE 以外の Web ブラウザで実行できるため、クロスブラウザ対応ができたことになるが、メリットはそれだけではない。SP4i では、HTML5 に対応しているため css3 を利用してデザインが可能となり、レスポンシブデザインで画面も実装できる。【図 13】

レスポンシブデザインとは 1 つの HTML からデバイスの画面サイズに合わせて複数の見え方で表現する手法である。

また jQuery などのオープンソースを組み込んだカスタマイズを行うこともできる。

jQuery とは、アメリカのプログラマー John Resig（ジョン・レッシグ）氏によって開発・公開された JavaScript 用のライブラリである。jQuery は著作権表示を消さなければ、商用・非商用を問わず、誰でも自由に利用することができるメジャーなオープンソースである。jQuery を使うメリットは大きく 2 つある。

Ⅰ. jQuery の JavaScript を使うとコーディング量が減らせる
Ⅱ. jQuery に対応したオープンソースの部品が利用できる

Ⅰについては、JavaScript で複数行にわたるソースコードも、jQuery では 1 つのメソッドで実現できる場合も多く、ソースコードを簡略化することができる。

Ⅱについては、jQuery を利用したオープンソースの部品も Web 上には多く公開されており、たとえば「OVERSCROLL.JS」というオープンソースを利用すると、ヘッダーを固定したままスクロールできる便利なサブファイルの表部品を利用可能である。【図 14】【図 15】

あくまで jQuery の一例ではあるが、活用すると使いやすい画面を労なく作成できる。jQuery などを使った詳しい拡張方法については、2015 年のテクニカルレポート No.8 に掲載されている「スマートデバイス開発で役立つ 画面拡張テクニック」を参照いただきたい。

このように SP4i にマイグレーションしたアプリケーションでは、使用できる Web ブラウザの種類が増えるだけでなく、新しい画面設計や機能拡張にも対応できるメリットがある。

5.補足:自動ログオン機能の強化

最後に JC/400 から SP4i で強化された環境の違いとして自動ログオン機能について補足する。自動ログオン機能とは、ログオン画面のユーザー／パスワードの入力を省略してアプリケーションを使用できる機能である。SP4i ではこの自動ログオンの機能が便利に強化されている。

自動ログオン機能を使用する場合、ログオンの入力を省略する代わりに、固定のアカウントが使用される。

JC/400 では、signon.txt というアカウントファイルで固定のユーザープロファイルを指定できる。これで自動ログオン可能だが、ログオンするユーザーは signon.txt のユーザープロファイルに固定されてしまう。

この機能が強化された SP4i では自動ログオンに使うアカウントファイルを HTML ごとに指定できるようになっている。具体的には name 属性 SIGNON の value 値に「自動ログオンするアカウントファイル」を設定できる。【ソース 4】

この SP4i の自動ログオンの仕組みによって、JC/400 で固定されていたユーザープロファイルを自由に使い分けることができる。【図 16】

たとえば部門や役職によって権限やメニューを自動ログオンするユーザープロファイルで制御ができるので、SP4i では利用ユーザーにも柔軟に対応できるようになる。

6.おわりに

本稿では、新しい Web ブラウザ環境への対応方法として、JC/400 から SP4i へのマイグレーションのポイントを説明した。

マイグレーションの内容としては、JC/400（IE 限定）のプログラムがそのまま SP4i（別の Web ブラウザ）で動かせる内容となっているが、4 章で例示したように HTML5 などの機能や jQuery などのライブラリを組み込めば、今まで以上に綺麗なデザインで高機能な Web 画面にカスタマイズしていくことができる。

また冒頭でも述べたとおり、SP4i はハイブリッド Web アプリケーションとしての機能を持っているため、単純に Chrome や Safari で動かせるようになるだけでなく、モバイル向けのアプリケーションも開発できる。本稿では SP4i のモバイルアプリケーション開発までは説明をしていないが、RPG をベースとしたアプリケーションでカメラやバーコード、マップなどの機能が活用で

図6　IBMiへ配布

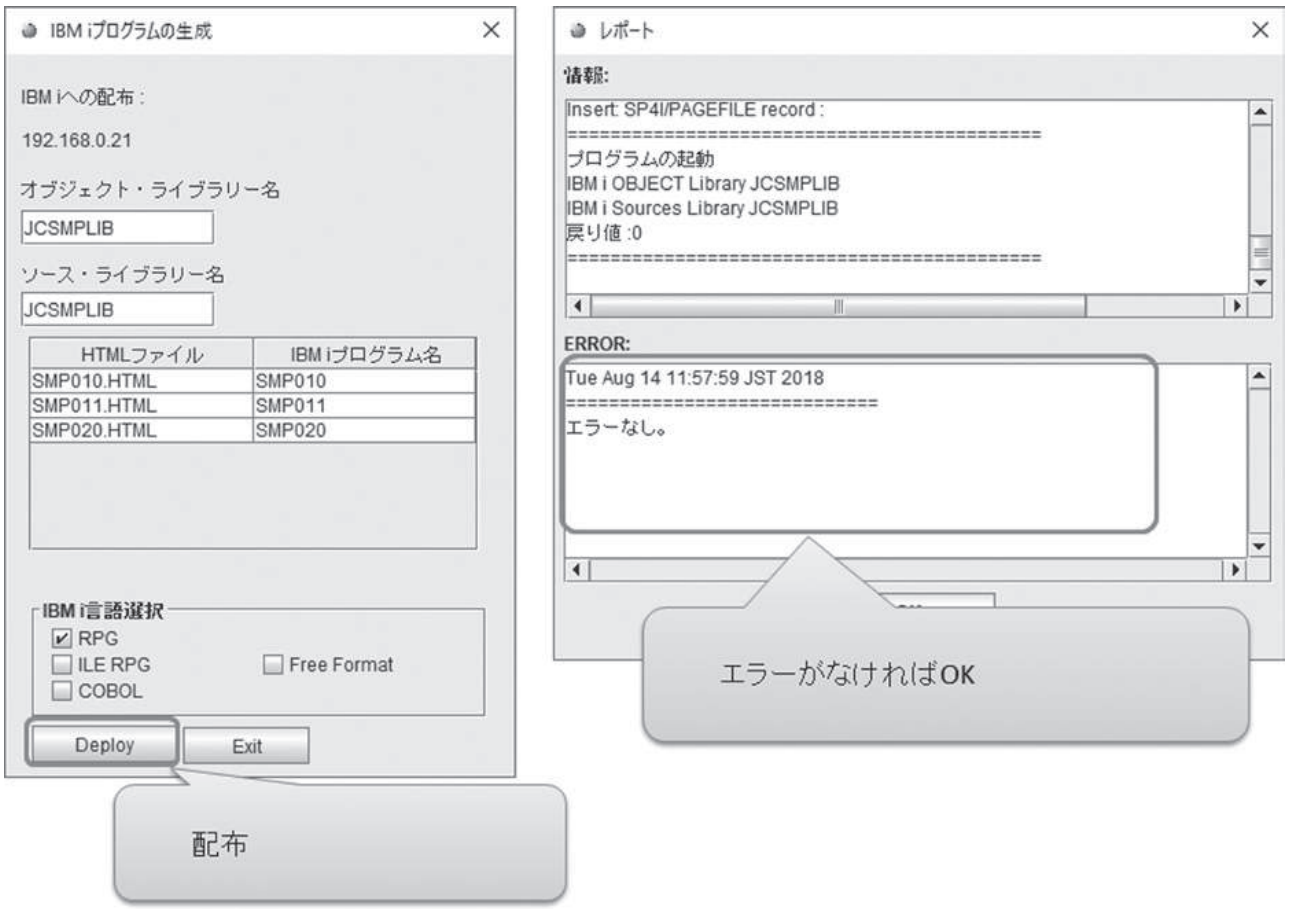
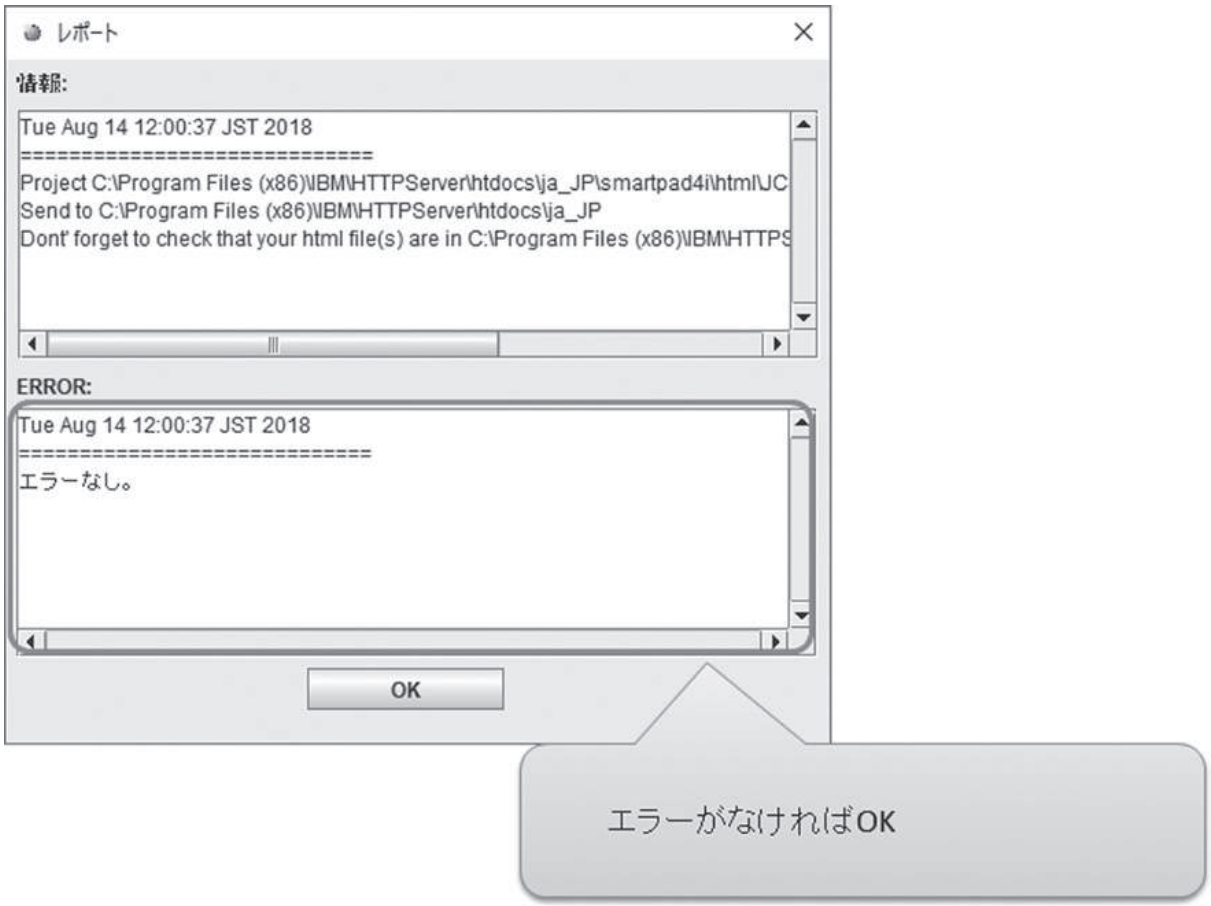


図7　WEBサーバーへ配布



きるので、RPG で実現できるシステムの幅が大きく広がる。【図 17】

このマイグレーションを入り口に、SP4i を新しいアプリケーション開発の開拓にも役立てていただければ幸いである。

M

ソース1　JC/400のライブラリリスト

```
0001.00 PGM
0002.00 CHGLIBL LIBL(XXXLIB JACI400DEV JACI400 QTEMP QGPL)
0003.00 ENDPGM
```

SP4iのライブラリリスト

```
0001.00 PGM
0002.00 CHGLIBL LIBL(XXXLIB SP4I QTEMP QGPL)
0003.00 ENDPGM
```

図8　変更が必要なAPI名

JC/400	SP4i	備考
JACIINIT	SP4IINIT	初期化
JCSEND	SPSEND	ブラウザ側に送信
JCRECV	SPRECV	ブラウザ側から受信
JACISATR	SP4ISATR	SETATR の機能
JACIPHONE	SP4IPHONE	クライアント端末情報取得機能
JACIIPADR	SP4IIPADR	IP アドレス取得
JACISPFM	SP4ISPFM	CSV ファイル出力
JACIOFCK	SP4IOFCK	DATAQ KEY を取得
JACIOFCF	SP4IOFCF	データベースファイルを送る
JACIOFCO	SP4IOFCO	クライアント PC 上に新しいファイルを作る/開く
JACIOFCW	SP4IOFCW	開いたファイルの書き込み
JACIOFCC	SP4IOFCC	開いたファイルを閉じる
JACIOFCD	SP4IOFCD	クライアント PC のファイルを実行、開く
JACIOFCACK	SP4IOFCACK	OFFICE 機能からの通知を待つ

図9　変更が必要なフィールド変数名①

JC/400	SP4i	備考
JCLIBC	SPLIBC	ライブラリ
JCHNDL	SPHNDL	ハンドル
JCLIB	SPLIB	ライブラリ
JCFILE	SPFILE	HTML ファイル名
JCRETN	SPRETN	リターンコード
JCRCDN	SPRCDN	レコード名
JCMULT	SPMULT	複数行のフラグ
JCRCDL	SPRCDL	レコード長
JCCSRF	SPCSRF	カーソルフィールド
JCCSRL	SPCSRL	カーソル行
JCFLGS	SPFLGS	処理フラグ
JCNBRL	SPNBRL	ループ行
JCLOOP	SPLOOP	ループ変数
JCTIMO	SPTIMO	タイムアウト
JCACTN	SPACTN	アクションコード
JCACTL	SPACTL	アクションコード行

図10　変更が必要なフィールド変数名②

JC/400	SP4i	備考
JCSFLS	SPSFLS	サブファイルループ開始行
JCSFLR	SPSFLR	サブファイルループ終了行
JCDUMY	SPDUMY	予備フィールド
JCTABN	SPTABN	タブ用フィールド
JCFILC	SPFILC	HTML ファイル
JCBUFF	SPBUFF	データ送信用
JCBUF1	SPBUF1	データ送信用
JCFLDN	SPFLDN	SETATR HTML の ID を設定
JCORDF	SPORDF	SETATR 対象行を指定
JCORD1	SPORD1	SETATR サブファイル対象開始行
JCATRC	SPATRC	SETATR 属性コード
JCCLAS	SPCLAS	SETATR クラス名
JCELEM	SPELEM	SETATR ラジオボタンの要素指定
JCATRB	SPATRB	SETATR バッファー
JCRETC	SPRETC	SETATR 結果コード
JCL10X	SPL10X	サブファイルレコード開始行
JCL90X	SPL90X	サブファイルレコード終了行

ソース2　JC/400のgetElementById

```
var inp01 = null;
function initpage(){
 inp01 = document.getElementById('INP01');
}
```

SP4iのgetElementById

```
var inp01 = null;
function initpage(){
 inp01 = SP4i.getElementById('INP01');
}
```

ソース3　JC/400のgetElementsByName

```
var items = null;
function initpage(){
 items = document.getElementsByName ('MYNAME');
}
```

SP4iのgetElementsByName

```
var items = null;
function initpage(){
 items = SP4i.getElementsByName ('MYNAME');
}
```

図11　JC/400アプリ実行画面

図12　SmartPad4iアプリ実行画面

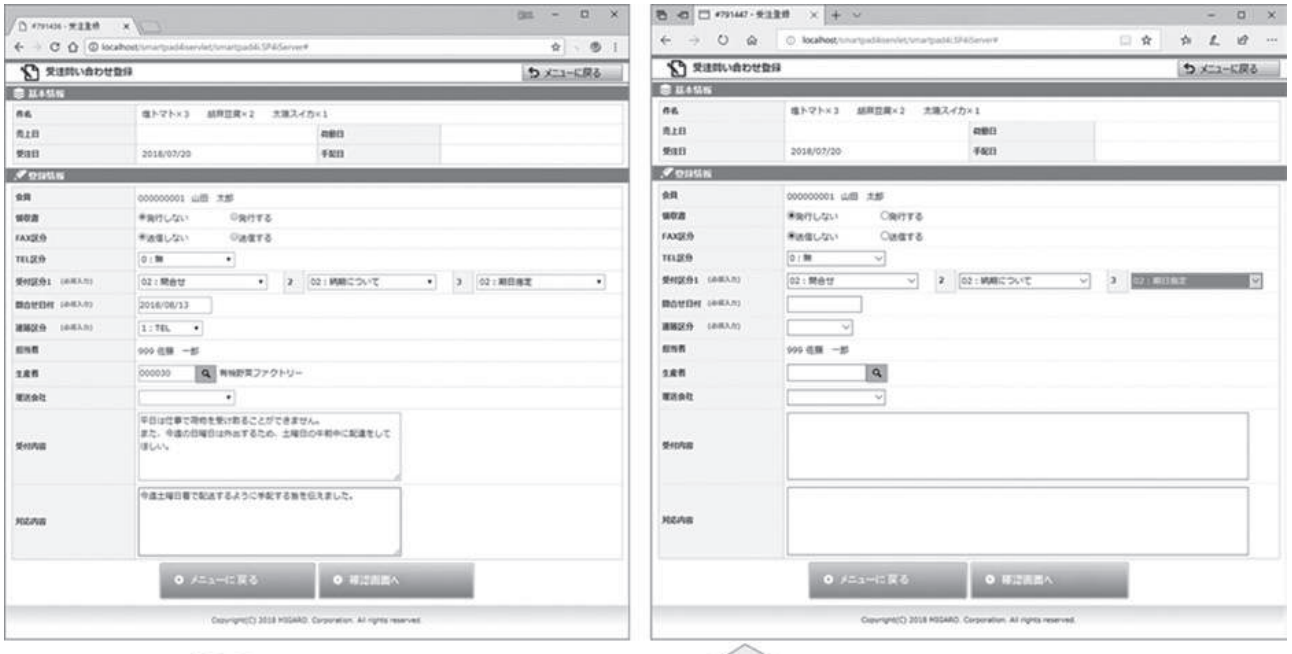


図13　画面サイズに合わせたレスポンシブ画面

図14　通常のサブファイルの一覧表

図15　オープンソース部品を使った一覧表

ソース4

```
1  <HTML>↵
2  <HEAD>↵
3    <SCRIPT type="text/javascript" src="sp4imsg.js"></SCRIPT>↵
4    <SCRIPT type="text/javascript" src="sp4iparm.js"></SCRIPT>↵
5    <SCRIPT type="text/javascript" src="sp4ilogon.min.js"></SCRIPT>↵
6  </HEAD>↵
7  <BODY>↵
8    <FORM method=POST>↵
9      <input type="hidden" name="SIGNON" value="profile/signonJCSMPLIB.txt">↵
10   </FORM>↵
11 </BODY>↵
12 <SCRIPT type="text/javascript">JScriptInit();</SCRIPT>↵
13 </HTML>↵
```

図16　自動ログオンの仕組み

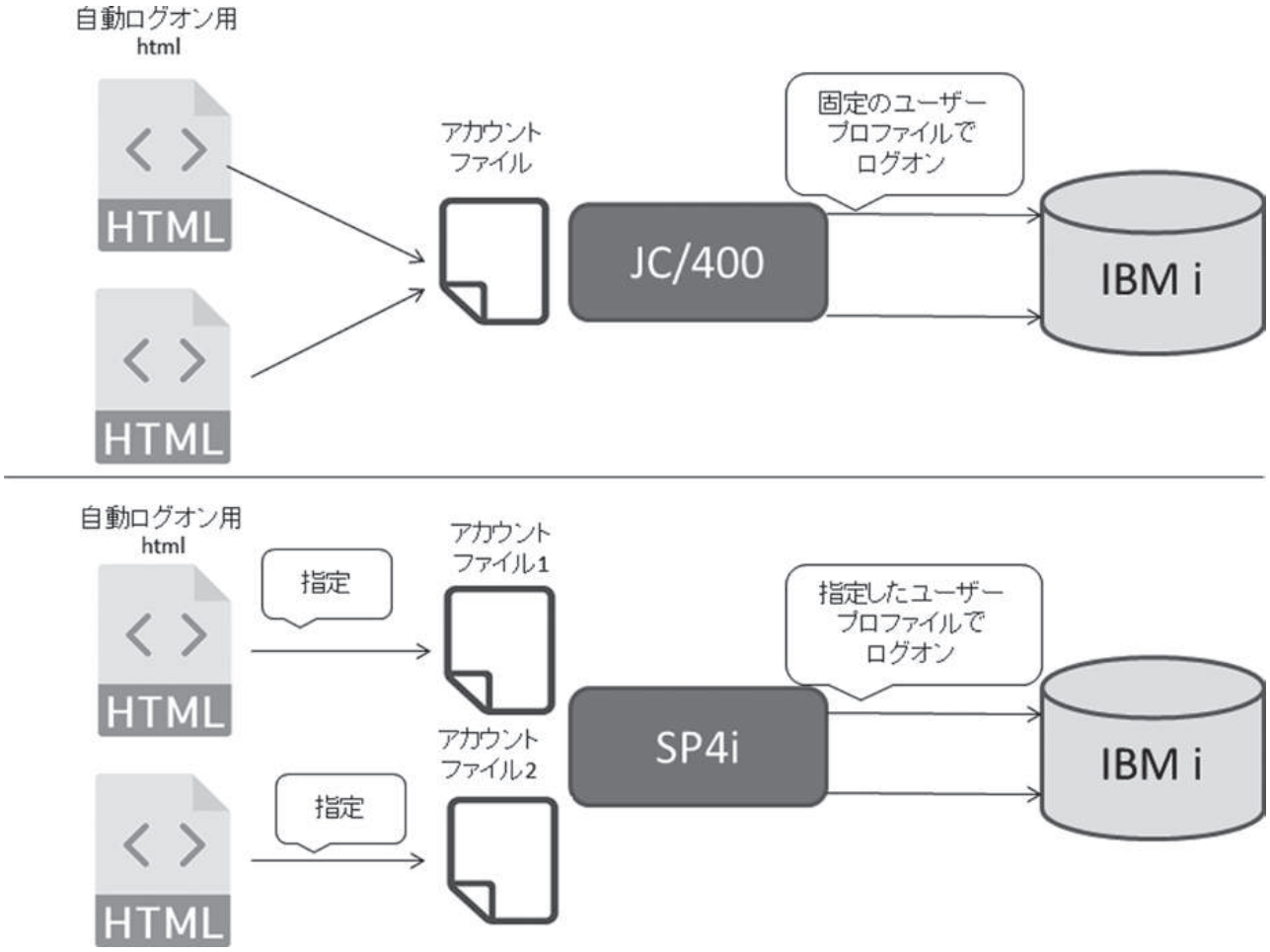


図17　モバイルのアプリケーション

Migaro. Technical Report
既刊号バックナンバー

電子版・書籍(紙)媒体で提供中!
http://www.migaro.co.jp/contents/support/technical_report/

No.1　2008 年秋

お客様受賞論文

●最優秀賞

直感的に理解できるシステムを目指して―情報の“見える化”の取り組み
石井 裕昭様／豊鋼材工業株式会社

●ゴールド賞

運用部間にサプライズをもたらした Delphi/400
春木 治様／株式会社ロゴスコーポレーション

●シルバー賞

JACi400 使用による Web アプリケーション開発工数削減
中富 俊典様／日本梱包運輸倉庫株式会社

Delphi/400 を利用した Web 受注システム
飯田 豊様／東洋佐々木ガラス株式会社

●優秀賞

Delphi/400 による販売管理システム（FAINS）について
藤田 建作様／株式会社船井総合研究所

技研化成の新基幹システム再構築
藤田 健治様／技研化成株式会社

SE 論文

はじめての Delphi/400 プログラミング
畑中 侑／システム事業部 システム２課

Delphi/400 と Excel との連携
中嶋 祥子／ RAD 事業部 技術支援課

連携で広がる Delphi/400 活用術
尾崎 浩司／システム事業部 システム２課

フォーム継承による効率向上開発手法
吉原 泰介／ RAD 事業部 技術支援課

API を利用した出力待ち行列情報の取得方法
鶴巣 博行／ RAD 事業部 技術支援課

Delphi テクニカルエッセンス Q&A 集
吉原 泰介／ RAD 事業部 技術支援課

JACi400 を使って RPG で Web 画面を制御する方法
松尾 悦郎／システム事業部 システム２課

あなたはプラインドタッチができますか？
福井 和彦／システム事業部 システム１課

No.2　2009 年秋

お客様受賞論文

●最優秀賞

JACi400 で 既存 Web サービスの内製化を実現
佐々木 仁志様／株式会社ジャストオートリーシング

●ゴールド賞

.NET 環境での Delphi/400 の活用
福田 祐之様／林兼コンピューター株式会社

●シルバー賞

5250 で動作する「中古車 在庫照会プログラム」の GUI 化
佐久間 雄様／株式会社ケーユー

●優秀賞

Delphi による 輸入システム「MISYS」の再構築
秦 榮禧様／株式会社モトックス

Delphi/400 による 物流システムの再構築
仲井 学様／西川リビング株式会社

Delphi/400 で開発し 3 台のオフコンを 1 台の IBM i へ統合
島根 英行様／シルフ

SE 論文

JACi400 環境でマッシュアップ！
岩田 真和／ RAD 事業部 技術支援課

Delphi/400 を利用したはじめての Web 開発
福岡 浩行／システム事業部 システム２課

Delphi/400 を使用した Web サービスアプリケーション
尾崎 浩司／システム事業部 システム３課

Delphi/400 によるネイティブ資産の応用活用
吉原 泰介／ RAD 事業部 技術支援課 顧客サポート

RPG でパフォーマンスを制御
松尾 悦郎／システム事業部 システム１課

MKS Integrity を利用したシステム開発
宮坂 優大・田村 洋一郎／システム事業部 システム１課

No.3　2010 年秋

お客様受賞論文

●最優秀賞

建物のクレーム情報管理システム「アフターサービス DB」について

大橋 良之様／東レ建設株式会社

●ゴールド賞

Delphi/400 で「写真管理ソフト」と「スプールファイルの PDF 化ソフト」を自社開発

寒河江 幸喜様／日綜産業株式会社

●シルバー賞

Delphi/400 で鉄鋼受発注業務を統一し 鉄鋼 EDI も実現

柿本 直樹様／合鐵産業株式会社

●優秀賞

Delphi/400 で EIS（Executive Information System）の高速化

小島 栄一様／西川計測株式会社

イントラでの PHP-Delphi-RPG 連携

仲井 学様／西川リビング株式会社

Delphi/400 を使った取引先管理システム

大崎 貴昭様／森定興商株式会社

SE 論文

Delphi/400 ローカルキャッシュ活用術

中嶋 祥子／ RAD 事業部 技術支援課

Delphi/400 帳票開発ノウハウ公開

尾崎 浩司／システム事業部 システム３課

Delphi/400 でドラッグ&ドロップを制御

辻林 涼子／システム事業部 システム２課

Delphi/400 のモジュールバージョン管理手法

前田 和寛／システム事業部 システム２課

Delphi/400　Web からの PDF 出力

福井 和彦・清水 孝将／システム事業部システム３課・システム２課

Delphi/400 で Flash 動画の実装

吉原 泰介／ RAD 事業部 技術支援課 顧客サポート

No.4　2011 年秋［創立 20 周年記念号］

お客様受賞論文

●最優秀賞

全社の経費処理業務を効率化した「e 総務システム」

鈴木 英明様／阪和興業株式会社

●ゴールド賞

「Web 進捗管理システム」でリアルタイム性を実現

堀内 一弘様／エスケーロジ株式会社

●シルバー賞

「営業奨励金申請書」をたった２日間で開発

簑島 宏明様／株式会社ケーユーホールディングス

液体輸送における「配車支援システム」の構築

桂 哲様／ライオン流通サービス株式会社

SE 論文

グラフ活用リファレンス

中嶋 祥子／ RAD 事業部 技術支援課

Web サービスを利用して機能 UP ！

福井 和彦・畑中 侑／システム事業部 システム２課

OpenOffice 実践活用

吉原 泰介／ RAD 事業部 技術支援課 顧客サポート

VCL for the Web 活用 TIPS 紹介

尾崎 浩司／システム事業部 プロジェクト推進室

JC/400 で JavaScript 活用

清水 孝将／システム事業部 システム１課

jQuery 連携で機能拡張

國元 祐二／ RAD 事業部 技術支援課 顧客サポート

No.5　2012 年秋［創刊 5 周年記念］

お客様受賞論文

【部門 1】

●最優秀賞

JC/400 による取引先との Web-EDI システム構築

久保田 佳裕様／極東産機株式会社

●ゴールド賞

Delphi と Excel を使用した帳票コストの削減

大久保 治高様／合鐵産業株式会社

もっと見やすく、もっと使いやすい画面を

新谷 直正様／株式会社アダル

【部門 2】

●優秀賞

Delphi/400 で確認業務の効率化

為国 順子様／ベネトンジャパン株式会社

取引先申請システムでの稟議書作成ワークフロー

大崎 貴昭様／森定興商株式会社

Delphi/400 で IBM i のストアードプロシージャを利用し、SQL 処理を高速化

島根 英行様／シルフ

SE 論文

InstallAware を使った Delphi/400 運用環境の構築

中嶋 祥子／ RAD 事業部 技術支援課 顧客サポート

カスタマイズコンポーネント入門 Delphi/400 開発効率向上

前田 和寛／システム事業部 システム 2 課

Delphi/400 スマートデバイスアプリケーション開発

吉原 泰介／ RAD 事業部 技術支援課 顧客サポート

DataSnap を使用した 3 層アプリケーション構築技法

尾崎 浩司／システム事業部 プロジェクト推進室

JC/400 でポップアップウィンドウの制御&活用ノウハウ

清水 孝将・伊地知 聖貴／システム事業部 システム 1 課

【創刊 5 周年記念】

ミガロ.SE 座談会―お客様と共に歩む、お客様への熱い思い

No.6　2013 年秋

お客様受賞論文

【部門 1】

●最優秀賞

自社用開発フレームワークの構築

駒田 純也様／ユサコ株式会社

●ゴールド賞

Delphi/400 で CTI 開発および関連機能組み込み

仲井 正人様／株式会社スマイル・ジャパン

●シルバー賞

IBM WebFacing から JC/400 への移行・リニューアル手法

八木 秀樹様／極東産機株式会社

Delphi/400 と Delphi を利用した IBM i 資源の有効活用

小山 祐二様／澁谷工業株式会社

発注システムを VB から Delphi へ移植しリニューアル

川島 寛様／株式会社タツミヤ

【部門 2】

●優秀賞

5250 画面を使用せずに AS/400 スプールファイルをコントロールする

白井 昌哉様／太陽セメント工業株式会社

Delphi/400 を利用した 承認フロー導入による IT 内部統制構築

塚本 圭一様／ライオン流通サービス株式会社

SE 論文

FastReport を使用した帳票作成入門

尾崎 浩司／ RAD 事業部 営業推進課

Delphi/400 で開発する 64bit アプリケーション

吉原 泰介／ RAD 事業部 技術支援課 顧客サポート

Web コンポーネントのカスタマイズ入門

佐田 雄一／システム事業部 システム 1 課

Indy を利用したメール送信機能開発

辻野 健・前坂 誠二／システム事業部 システム 2 課

Windows テキストファイル操作ノウハウ

小杉 智昭／システム事業部 プロジェクト推進室

JC/400 Web アプリケーションのユーザー管理・メニュー管理活用術

吉原 泰介・國元 裕二／ RAD 事業部 技術支援課 顧客サポート

No.7　2014 年秋

お客様受賞論文

【部門 1】

●最優秀賞

Delphi/400 による生産スケジューラの再構築

柿村 実様／東洋佐々木ガラス株式会社

●ゴールド賞

Delphi/400 および Delphi を利用したオンライン個人別メニューの構築

小山 祐二様／澁谷工業株式会社

●シルバー賞

IBM i と Delphi/400 のコラボレーション

新谷 直正様／株式会社アダル

●シルバー賞

荷札発行システムリプレースについて

仲井 学様／西川リビング株式会社

【部門 2】

●優秀賞

Delphi/400 バージョンアップのためのクライアント環境構築

普入 弘様／株式会社エイエステクノロジー

●優秀賞

外出先からメールでリアルタイム在庫を問い合せ

島根 英行様／シルフ

SE 論文

iOS/Android ネイティブアプリケーション入門

吉原 泰介／ RAD 事業部 技術支援課

ファイル加工プログラミングテクニック

小杉 智昭／システム事業部 プロジェクト推進室

FastReport を使用した帳票作成テクニック

前坂 誠二／システム事業部

大量データ処理テクニック

佐田 雄一／システム事業部

スマートデバイス WEB アプリケーション入門

尾崎 浩司／ RAD 事業部 営業推進課

國元 祐二／ RAD 事業部 技術支援課

No.8　2015 年秋

お客様受賞論文

【部門 1】

●最優秀賞

iPod Touch の業務利用開発と検証

石井 裕昭様／豊鋼材工業株式会社

●ゴールド賞

ブランク加工図管理システムの構築

小山 祐二様／澁谷工業株式会社

●シルバー賞

Delphi/400 でスプールファイル管理（WRKSPLF コマンドの活用）

三好 誠様／ユサコ株式会社

●シルバー賞

予算管理システムの構築

川島 寛様／株式会社タツミヤ

●シルバー賞

送状データ送信システムの Web 化について

仲井 学様／西川リビング株式会社

【部門 2】

●優秀賞

繰り返し DB 参照時の ClientDataSet の First 機能について

牛嶋 信之様／株式会社佐賀鉄工所

●優秀賞

IBM i のカレンダーを基準に他のシステムを稼働

福島 利昭様／株式会社ランドコンピュータ

SE 論文

フレームを利用した開発手法

前坂 誠二／システム事業部 システム 2 課

Windows タブレット用にカスタムソフトウェアキーボードを実装

福井 和彦／システム事業部 プロジェクト推進室

マルチスレッドを使用したレスポンスタイム向上

尾崎 浩司／ RAD 事業部 営業・営業推進課

Android アプリケーションの NFC 機能活用

吉原 泰介／ RAD 事業部 技術支援課 顧客サポート

スマートデバイス開発で役立つ画面拡張テクニック

國元 祐二／ RAD 事業部 技術支援課 顧客サポート

No.9　2016 年秋

お客様受賞論文

【部門 1】

●最優秀賞

IBM i の見える化で実現するアジャイル開発

吉岡 延泰様／日本調理機株式会社社

●ゴールド賞

Windows Like 5250 への道のり

小山 祐二様／澁谷工業株式会社

●シルバー賞

Delphi プログラム管理ソフトの開発

牛嶋 信之様／株式会社佐賀鉄工所

【部門 2】

●優秀賞

Delphi/400 を利用した定型業務の PDF 化

佐藤 岳様／ライオン流通サービス株式会社

●優秀賞

ちょい足しモバイル

仲井 正人様／株式会社スマイル・ジャパン

●優秀賞

AS/400 の受注データを Web で社員に公開

福島 利昭様／株式会社ランドコンピュータ

SE 論文

iOS モバイルアプリ開発のデザイニングテクニック

前坂 誠二／システム事業部 システム 2 課

新データベースエンジン FireDAC を使ってみよう！

福井 和彦／システム事業部 プロジェクト推進室

Delphi/400 最新プログラム文法の活用法

尾崎 浩司／ RAD 事業部 営業・営業推進課

FastReport を活用した電子帳票作成テクニック

宮坂 優大／システム事業部 システム 1 課

Beacon 技術による IoT 活用の第一歩

吉原 泰介／ RAD 事業部 技術支援課 顧客サポート

Web & ハイブリッドアプリ開発で役立つ IBM i & ブラウザデバッグテクニック

國元 祐二／ RAD 事業部 技術支援課 顧客サポート

No.10　2017 年秋

Migaro.Technical Report 創刊 10 周年記念

パートナー様からの祝辞

武藤 和博様／日本アイ・ビー・エム株式会社

藤井 等様／エンバカデロ・テクノロジーズ日本法人

Serge Charbit 様／ SystemObjects Corporation

飯田 恭子様／アイマガジン株式会社

お客様からの祝辞・お客様の声

石井 裕昭様／豊鋼材工業株式会社

牛嶋 信之様／株式会社佐賀鉄工所

大崎 貴昭様／森定興商株式会社

川島 寛様／株式会社タツミヤ

久保田 佳裕様／極東産機株式会社

駒田 純也様／ユサコ株式会社

小山 祐二様／澁谷工業株式会社

寒河江 幸喜様／日綜産業株式会社

佐々木 仁志様／株式会社ジャストオートリーシング

佐藤 岳様／ライオン流通サービス株式会社

白井 昌哉様／太陽エコブロックス株式会社

仲井 学様／西川リビング株式会社

福島 利昭様／株式会社ランドコンピュータ

お客様座談会

石井 裕昭様／豊鋼材工業株式会社

駒田 純也様／ユサコ株式会社

寒河江 幸喜様／日綜産業株式会社

仲井 学様／西川リビング株式会社

上甲 將隆／株式会社ミガロ.

司会　飯田 恭子様／アイマガジン株式会社

お客様受賞論文

【部門 1】

●最優秀賞

貸金庫と鍵のマッチング業務を Delphi/400 で実現
―文字認識データと基幹システムデータを統合

佐藤 正様／株式会社富士精工本社

●ゴールド賞

Windows タブレット導入による工作部材料受入業務改革

小山 祐二様／澁谷工業株式会社

【部門 2】

●優秀賞

Delphi/400 を利用した各拠点 PING コマンド簡素化

松垣 秀昭様／ライオン流通サービス株式会社

汎用的な帳票出力画面

牛嶋 信之様／株式会社佐賀鉄工所

バーコードリーダー読み取り後、次の入力位置にカーソルを自動遷移させる技術

上総 龍央様／キョーラクシステムクリエート株式会社

IBM i のスプールファイル参照機能を Web で構築

福島 利昭様／株式会社ランドコンピュータ

SE 論文

デスクトップアプリケーション開発でも
役立つ FireMonkey 活用入門

尾崎 浩司／ RAD 事業部 営業・営業推進課

Delphi/400 バージョンアップに伴う文字コードの違いと制御

宮坂 優大／システム事業部 システム 1 課

FastReport への効率的な帳票レイアウトコンバート

畑中 侑／システム事業部 システム 2 課

IBM i トリガー機能を活かしたセキュリティログ対応

八木沼 幸一／システム事業部 プロジェクト推進室

アプリケーションテザリングを利用した PC &モバイルアプリケーション連携

吉原 泰介／ RAD 事業部 技術支援課 顧客サポート

SmartPad4i の運用で役立つ WEB サーバー機能

國元 祐二／ RAD 事業部 技術支援課 顧客サポート

MEMO

MIGARO.
TECHNICAL
REPORT

Migaro.Technical Report
No.11　2018 年秋
ミガロ.テクニカルレポート

2018 年 12 月 1 日 初版発行

◆発行
株式会社ミガロ.
〒 556-0017
大阪府大阪市浪速区湊町 2-1-57 難波サンケイビル 13F
TEL：06（6631）8601　FAX：06（6631）8603
http://www.migaro.co.jp/

◆発行人
上甲 將隆

◆編集協力
アイマガジン株式会社

◆デザインフォーマット
近江デザイン事務所

MIGARO. TECHNICAL REPORT

ミガロ.テクニカルレポート

株式会社ミガロ.

http://www.migaro.co.jp/

本社
〒556-0017
大阪市浪速区湊町2-1-57
難波サンケイビル 13F
TEL:06(6631)8601
FAX:06(6631)8603

東京事業所
〒100-0013
東京都千代田区霞が関3-7-1
霞が関東急ビル 2F
TEL:03(5510)5701
FAX:03(5510)5702