MIGARO. TECHNICAL REPORT

ミガロ.テクニカルレポート

No.9
2016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た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9号を無事に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。

最近、AIやIoTなどの話題が盛んにニュースを賑わしています。これらのIT技術は、研究段階を経て実用・普及の段階を迎えたと考えられます。弊社でも最先端のIT技術を容易にご利用いただくために、IoT機能を強化した「Delphi/400 10 Seattle」を今年7月にリリースいたしました。ほかにも、マルチデバイス開発機能の強化、新データベースエンジンFireDAC対応、Windows10対応など、多くの改善を盛り込んだバージョンになっています。その機能の一端は本誌でも紹介していますので、ぜひご覧ください。

さて、今回の『Migaro.Technical Report』も従来と同様に、第1部は「Migaro.Technical Award 2016 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」「Business4Mobile」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、「IBM iプログラム資産をDelphi/400で見える化しアジャイル開発を実現した事例」や「5250画面プログラムにWindowsのような操作性を追加した事例」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、「iOSモバイルアプリケーションの開発テクニック」や「Beacon技術によるIoTの活用」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9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。

2016年秋

株式会社ミガロ.
代表取締役社長
上甲 將隆

Migaro.
Technical Award
2016

お客様受賞論文／ミガロ. テクニカルアワード

最優秀賞

IBM iの見える化で実現するアジャイル開発
―IBM iユーザーにありがちな困りごとを、RPGとDelphiで解決する

吉岡 延泰 様
日本調理機株式会社
情報システム部
主任

日本調理機株式会社
http://www.nitcho.co.jp/

心と体の健全な発育をサポートする学校給食、治癒意欲を促すおいしい病院食、心身をリフレッシュさせて生産性を高める社員食堂など、日本調理機はフードサービス産業を支える総合厨房機器メーカーとして、革新的なキッチンづくりと快適な作業環境を追求している。

開発の経緯

日本調理機は1993年にAS/400を基幹システムとして導入し、現在のIBM iに至るまで利用し続けている。

業務に合わせたシステム開発は内製で行っている。総合機器メーカーであることから、必要な業務アプリケーションの種類が多く、システムの構造化を進めていたが、長年の開発・改修によりシステムが肥大化。影響調査にかかる工数が増大していった。

システムの構造化は、プログラムをカプセル化して個々の品質を保つのに重要ではあるが、同時にシステム全体の把握が困難になるというジレンマが付きまとう。結果的に、大規模なシステム改修が難しくなるという悪循環に陥っていた。

その後、Delphi/400 XEを2011年に導入。データのエントリーはキーボードのみで操作できる利点などから、5250エミュレータを継続して使用していたが、参照・印刷などのアプリケーションは、表現力の高いDelphi/400で作成することが増えてきた。RPG、CLに加え、Delphiの習得が必須になったことで、開発者の育成にこれまで以上に時間がかかるようになる。

また一定の技術力を習得しても、システム全体の把握にはさらに多くの期間が必要であり、そのためシステムの現状を記した仕様書などの文書を並行して管理するのに、時間と手間がかかりすぎていた。

そこで既存のシステムを見える化し、少人数で高効率なアジャイル開発向けのシステムにシフトしていくことが課題になった。RPG、CL、Delphi/400という自社にとって使い慣れた技術の組み合わせで、開発・管理しやすい環境を構築していくことを決めた。

前提

5250のメニュー画面は、自社独自の構成になっている。【図1】

使用言語はRPGとCLであるが、RPGⅢとRPGⅣが混在している。RPGⅣの特記事項として、サービスプログラムのバインドを行っている点と、一部がフリーフォームで書かれている点がある。

解決したい既知の問題点

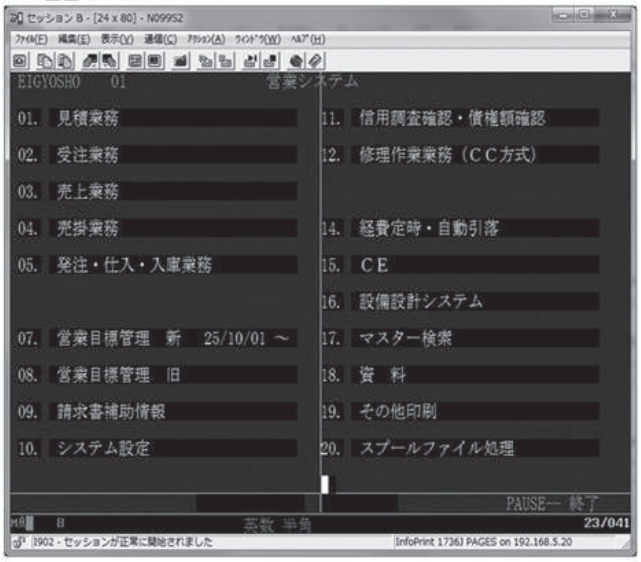
・構造化されたシステムの全体像を把握するのに時間がかかり、大規模な改修が容易に行えない。問題が発生した際にも、調査に時間がかかる場合が多かった。

・影響調査の際に、FNDSTRPDMの検索結果を元にしていたが、余計な検索結果が多く、精査に無駄な時間がかかっていた。見落としなどのヒューマンエラーも起こりやすい状況であった。

・物理ファイルに新規のフィールドを追加したいが、時間がかかるので、新たに物理ファイルを作成するケースが多かった。結果的にデータベースの正規化が正しく行えず、システムがより

図1

5250画面のメニューイメージ



メニューファイル(PFソース)

```
                          UNIQUE
R MENUDTR                 TEXT('メニュー')
  MEUSR          10A      COLHDG('ﾕｰｻﾞｰ ID')
  MEPNNO          2A      COLHDG('パネル番号')
  MEPNSQ          2S 0    COLHDG('表示順序')
  MECLPN          2A      COLHDG('呼出すパネル')
  MECLPG         10A      COLHDG('呼出す ﾌﾟﾛｸﾞﾗﾑ')
  METTL          32O      COLHDG('表示 ﾀｲﾄﾙ')
  MEPSWD         10A      COLHDG('パスワード')
K MEUSR
K MEPNNO
K MEPNSQ
```

図2

ライブラリ管理(PFソース)

```
                          UNIQUE
R KNRLIBR                 TEXT('ライブラリ管理')
  KLLIB          10A      COLHDG('ライブラリ')
  KLFLG           1A      COLHDG('1:USE 2:OLD 3:ELSE')
  KLBR1           1A      COLHDG('1:PGM 2:FILE')
  KLBR2           1A      COLHDG('1: 販  2: 製')
  KLCMT1         70O      COLHDG('コメント１')
  KLCMT2         70O      COLHDG('コメント２')
K KLLIB
```

サインオン管理(PFソース)

```
                          UNIQUE
R KNRSIGR                 TEXT('サインオン管理')
  KSUSER         10A      COLHDG('サインオンユーザ')
  KSPNL          10A      COLHDG('パネルユーザ')
  KSPGM          10A      COLHDG('スタートＣＬ')
  KSFLG           1A      COLHDG('1:USE 2:OLD 3:ELSE')
  KSBR2           1A      COLHDG('1: 販  2: 製')
  KSCMT1         70O      COLHDG('コメント１')
  KSCMT2         70O      COLHDG('コメント２')
K KSUSER
```

調査を行った項目(例)

現在使用しているサインオンユーザー、ライブラリの特定と限定。 ADDLIBLやCHGLIBLで、現行用ライブラリと過去ライブラリの両方にアクセスできるサインオンユーザーを無くし、現行データor過去データのアクセスを完全に分離する。(影響調査の範囲を限定する為。)
複数のライブラリに存在する物理ファイルの構造調査。
複数のライブラリに、同名のプログラムが存在していないかを調査。(全ての現行ライブラリの中で、プログラム名をユニークにする。)
別のライブラリの物理ファイルへ接続している従属論理ファイルの調査。
ソースとオブジェクトのどちらかが存在していないものを調査・補完する。
RPGとCLの構造化に関して、呼び出しに使用している命令や記述方法の調査。

複雑になるという悪循環を招いていた。

・使われていないオブジェクト、ソースメンバーを整理（棚卸し）したいが、量が膨大（約２万件）であるため、精査できずにいた。

事前調査

システムの構造をプログラムで解析する場合、ある程度のルールに則ってシステムが構成されている必要がある。

たとえば自社内には、「論理ファイルの作成場所は物理ファイルと同じライブラリにする」というルールや、「別のライブラリに同名のファイルを作成する場合、フィールドの構成などをまったく同じに保つ」「QDDSSRC メンバーとオブジェクトは、必ず同じライブラリ内に置く」などのルールがある。

しかし長年運用されてきたシステムであるため、例外が存在していないかを事前に調査・精査する必要があった。またサインオンユーザー、ライブラリともに、現行のデータを扱うものと、過去データ用のものが存在するので、それぞれの管理ファイル【図 2】を作成し、調査結果をエントリーした。

アジャイル開発向けシステムの構成と要素

使用するのは開発者であることから、使いやすさよりも短期間での構築に重きを置き、汎用性の高い２つのプログラム（Delphi）を、各ツールで利用できるようにした。【図 3】はファイル内容の xls ダウンロード、【図 4】は CSV からのアップロードを示している。

システムは、以下に示す（1）～（5）の影響調査から一括コンパイルまでを行う一連のツール類と、補助的にさまざまな用途に使用できる（その他 1）～（その他 4）のツール類で構成した。

（1）～（5）の概要を【図 5】に示す。

（1）オブジェクト情報出力

処理リストのライブラリ名を対象に、オブジェクト情報を収集する（オブジェクト名テーブルファイルへの書き出し）。現行のデータがあるライブラリは、ライブラリ管理ファイルから一括で処理リストにセットできるが、CSV ファイルから処理リストをエントリーすることもできる。ソースの一部を【図 6】に示す。

（2）クローラー

PRG や CL のソースを解析し、構成情報を収集するプログラム（以下、クローラー）は、処理リストを元に、各オブジェクトの親子関係（呼び出し元プログラム＝親、呼び出されるファイルやプログラム＝子）を、ツリー構造のファイル（以下、ツリーファイル）に書き出す。

その際、オブジェクト名テーブルを参照して情報を付与する（前提として、オブジェクトとソースが同一ライブラリ内にセットで存在していること）。ソースの一部を【図 7】に示す。

処理リストは 5250 画面から 1 件ずつ入力・削除できるが、一括エントリーの 3 つの方法を以下のように用意した。

1. 現行のメニューにあるすべてのプログラム
2. FNDSTRPDM のコピー結果
3. CSV ファイル（ツリーファイルの全更新 or 差分更新は、処理リストのエントリー方法で使い分ける）

（3）構造の把握（Delphi）

ツリーファイルに集められたシステムの構造データは、ツリーの展開表示プログラムで、各オブジェクトの親子関係を簡単に展開して把握できる。展開したいプログラム名称は、クローラーの処理リストやメニューファイルから選択したり、一度展開してから一部分だけを選択して展開することもできる。概要を【図 8】に示す。

（4）構造の検索・リスト化（Delphi）

影響調査の際には、ファイルを使用している親プログラムや、子プログラムに対する親プログラムの一覧を調べることが多い。ツリーファイル内を検索してリスト化するプログラムを使用することで、すぐに影響調査の結果をダウンロードできる。データは、改修にあたるメンバー間で使う資料にそのまま使用できる。イメージを【図 9】に示す。

（5）一括コンパイル

RPG や CL のプログラムを、一括で連続してコンパイルできる。処理リストは、（4）のダウンロードデータを使える。ソースの一部を【図 10】に示す。コンパイルの成功・失敗の結果の一覧を取得することもできる。

（その他 1）メニューの検索・展開（Delphi）

メニューのツリー展開表示・検索プログラム（Delphi）を作成した。ユーザーからの問い合わせ時に、対象のプログラムがどこのメニューから呼び出されたものかをすぐに見つけられる。

（その他 2）ソース検索結果のダウンロード

FNDSTRPRM の検索結果を、各ツールの処理リストなどに使いたい場合があるため、検索結果をダウンロードできるようにした。

（その他 3）従属論理ファイル調査プログラム

事前調査で従属論理ファイルリストの調査が必要だったが、定期的なシステム全体のメンテナンスにも利用できる。

（その他 4）ソースとオブジェクトの整合性調査・一括棚卸し

事前調査でも必要になるが、システム全体のメンテナンスにも使用できる。定期的に実施することで、ソースやオブジェクトの棚卸しが簡単にできる。

実際には、（2）～（5）以外のツールは事前調査の時点で有効なものである。

アジャイル開発向けシステム構築のポイント

システム自体は、既知の技術の組み合わせであり、共通のロジックの多い構成にしたので、開発期間はさほどかからなかった（1 人月程度）。

重要なのは開発よりも、事前調査とシステムの整備がしっかりできるかどうかである。想定外の部分があれば当然、解析結果から漏れてしまう。既存のシステム全体をよく理解した技術者の元で、システムがどのような技術を使い、どのよ

図3

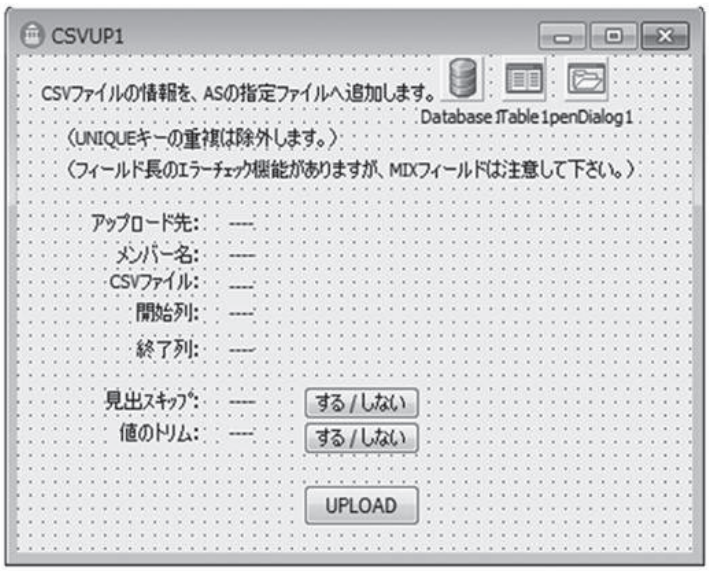
① 汎用ダウンロード（非表示）

パラメータ1	（必須）ライブラリ名
パラメータ2	（必須）ファイル名
パラメータ3	（必須）メンバー名　無しの場合は ' NONE'）
パラメータ4	（必須）　ローカル保存名
パラメータ5	（必須）見出し無し＝0/見出しカラム有り＝1/ フィールド名＝2/見出しカラム+フィールド名＝3
パラメータ6	（必須）列幅自動調整　有り＝1/無し＝0
パラメータ7	'OPEN' を指定すると、保存したファイルを開く。 /次に呼び出すプログラム　絶対パス 呼び出し。
パラメータ8	次に呼び出すプログラムへ渡すパラメータ。

STRPCCMDから実行。
ダウンロードだけでなく、ダウンロードしたデータを利用する別のプログラムを呼び出すこともできる。

図4

② 汎用アップロード（起動時にファイル選択ダイアログを表示）



パラメータ1	（必須）ライブラリ名
パラメータ2	（必須）ファイル名
パラメータ3	（必須）メンバー名　または 'NONE'）
パラメータ4	（必須）　開始列
パラメータ5	（必須）終了列
パラメータ6	（必須）見出し行スキップ　有り＝1/無し＝0
パラメータ7	（必須）値のトリム　有り＝1/無し＝0

STRPCCMDから実行。
桁のあふれなどをチェックして、CSVファイルの内容を物理ファイルに書き出す。

うな構成になっているのかを確認しながら構築することが重要である。

Delphi/400は、各ツールの処理間をCSVファイルでやり取りできるようにした汎用性・利便性と、ツリー構造の表示・展開など感覚的に理解できるGUIを備えたアプリケーションを短期間で実装できる点で効果を発揮した。

効果

長年の懸案であった問題点が解決され、システムの見える化と時間の大幅な短縮に成功した。

とくに物理ファイルへのフィールド追加に関しては､物理ファイル･論理ファイルのコンパイルなどの作業は手動で行う必要があるものの、調査から関連プログラムのコンパイルと確認までを30分程度で行えるようになり、限られた時間の中でもシステム全体をシンプルに最適化しながら開発を進めていけるようになった。

今後の展望と課題

ツリー表示は子から親への逆展開もできる。この結果は、システム改修の影響を受けるユーザーへの一括メール連絡や、プログラムの一括停止にも活用できる。

現在は定期的に各ツールを使用してメンテナンスしているが、バッチ処理にしてスケジューリング化すれば自動化できる。

課題としては、今後増えていくDelphiのアプリケーションへの対応がある。Delphiのソースファイル用のクローラーを別に作成するか、Delphiのアプリケーション自身が使用しているオブジェクトのリソース名をツリーファイルに書き出すようにすれば、RPGやCL等と同様に管理できるようになるはずだ。

総評

IBM iはさまざまな企業で長年使用される､堅牢性に優れたサーバーであるが､長く使用しているとそれに比例してシステムの複雑さを増していく。

RPGやCLだけでシステムの見える化を実現するのは難しいが、Delphi/400のグラフィカルなインターフェースや、高速で動作するSQLが扱えるQueryコンポーネントを活用すれば、短期間で十分使えるツールが開発できるとわかった。

システムがパンドラの箱になる前に、システム全体の健全さを保ちながら、効率よく開発できる環境を作っておくことは大切だと考える。

M

図5

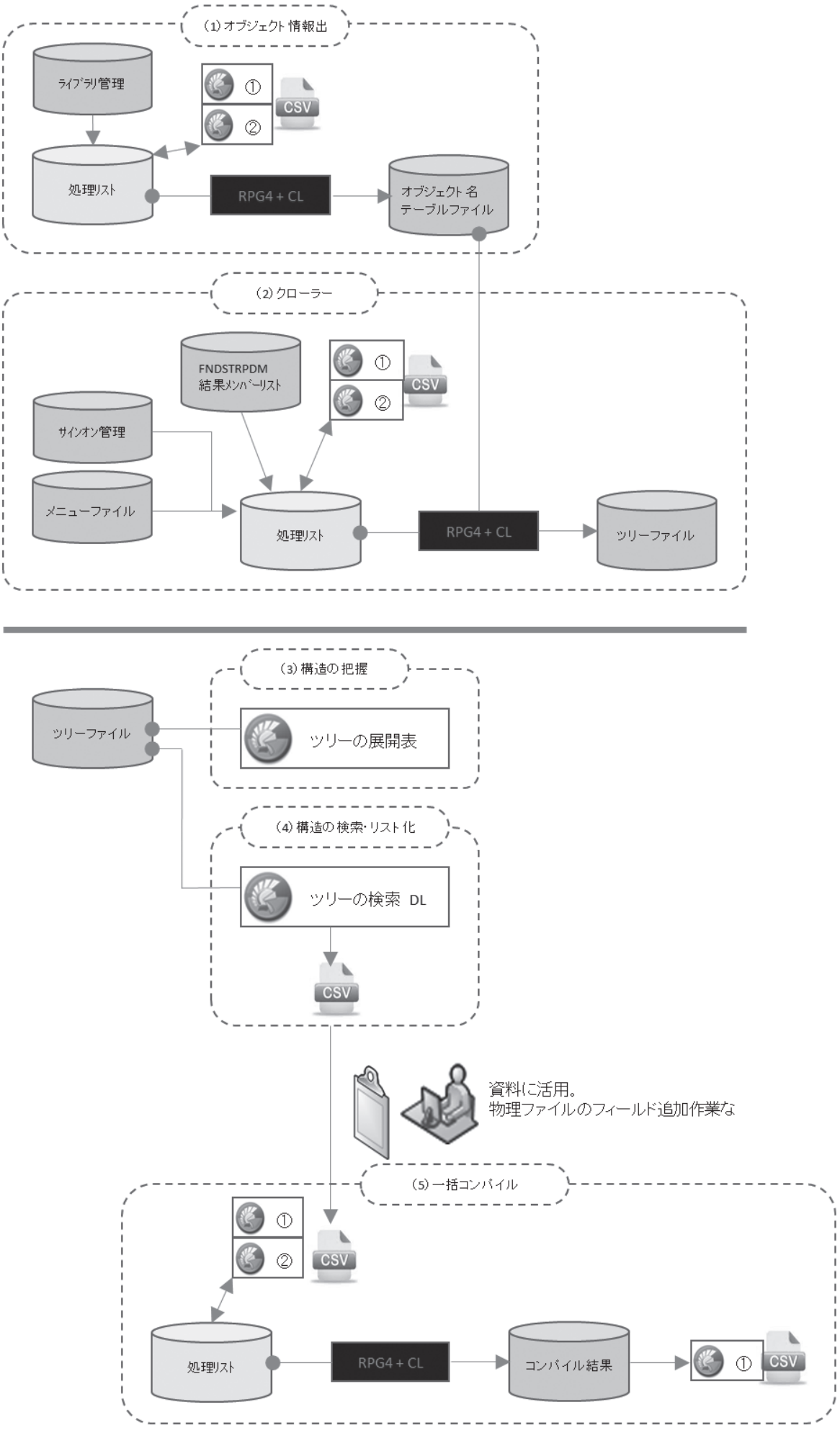

(1) オブジェクト情報出
ﾗｲﾌﾞﾗﾘ管理
①
②
CSV
処理ﾘｽﾄ
RPG4 + CL
オブジェクト名
テーブルファイル
(2) クローラー
FNDSTRPDM
結果ﾒﾝﾊﾞｰﾘｽﾄ
①
②
CSV
ｻｲﾝｵﾝ管理
メニューファイル
処理ﾘｽﾄ
RPG4 + CL
ツリーファイル
(3) 構造の把握
ツリーファイル
ツリーの展開表
(4) 構造の検索･リスト化
ツリーの検索 DL
CSV
資料に活用。
物理ファイルのフィールド追加作業な
(5) 一括コンパイル
①
②
CSV
処理ﾘｽﾄ
RPG4 + CL
コンパイル結果
①
CSV

図6

ライブラリからオブジェクト情報の一覧を書き出す （CLソースの一部）

```
PGM       PARM(&LIB)
DCL       VAR(&LIB) TYPE(*CHAR) LEN(10)
CLRPFM FILE(HONPRG0/OBJSTS)  MBR(*ALL)
DSPOBJD OBJ(&LIB/*ALL) OBJTYPE(*ALL) DETAIL(*FULL) +
OUTPUT(*OUTFILE) OUTFILE(HONPRG0/OBJSTS)  OUTMBR(*FIRST *ADD)
```

OUTFILEに指定したファイルは、ソースは存在せず、IBMiが作成する。
作成されたファイルには様々なオブジェクトの情報が含まれるが、利用しているフィールドのみ記述しておく。

オブジェクト名テーブルファイル

フィールド名	カラム	内容
ODLBNM	ライブラリー	
ODOBNM	オブジェクト	
ODOBTP	オブジェクト・タイプ	*PGM or *FILE
ODOBAT	オブジェクトの属性	CLP or RPG or RPGLE or DSPF or PF or LF
ODOBTX	テキスト記述	
ODCDAT	作成日付(MMDDYY)	
ODCTIM	作成時刻(HHMMSS)	
ODOBOW	オブジェクト所有者	
ODSRCF	ソース・ファイル名	
ODSRCL	ソース・ファイル・ライブラリー	
ODSRCM	ソース・ファイル・メンバー	
ODLDAT	変更日付(MMDDYY)	
ODLTIM	変更時刻(HHMMSS)	
ODCRTU	作成ユーザー	
ODUDAT	最終使用日付(MMDDYY)	
ODUCNT	使用日数カウント	

図7

QDDSSRCのソースメンバーは、RPGで直接読み取ることが出来ない為、
CPYFで一時的に物理ファイルにコピーする必要がある。80桁のフィールドが必要。

一時的に使用する物理ファイル(PFソース)

```
R PG011WR                   TEXT(' ソース書出 ')
  PWDATA        800         COLHDG(' ソース ')
```

ファイルにソースがコピーされたら、RPG4のプログラムで内容を読み取る。
ソースの種類はオブジェクト名テーブルファイルを参照することで、ソースの種類を判断できる。
読取り部分のソースは割愛するが、どんな呼び出し方をしているのかに合わせて、配列や関数でオブジェクト名称を取り出していく。
例えばRPGのソースであれば、プロンプトタイプの列が'F'ならファイル仕様が書かれている行だと分かる。
スキャンして'CALL'命令を見つければ、呼び出し先のプログラム名を取得できる。
フリーフォームで書かれたRPG4は'/FREE'を見つけて判断できる。
CLのソースでは、CALLの呼び出し先を括弧で括ったり、ライブラリを指定しないこともできるので、バリエーションが多い。

バインドされているサービスプログラムだけは、ソースからプログラムで判断するのが難しい為、
DSPPGMでソースをスプールに出力した結果を一時ファイルにコピーして読み取る。200桁のフィールドが必要。

一時的に使用する物理ファイル(PFソース)

```
R PG012WR                   TEXT(' 印刷内容書出 ')
  WDATA         2000        COLHDG(' 内容 ')
```

スプールして一時ファイルにコピーする(CLソースの一部)

```
/*   OUTOQ の設定                */
CHGPRTF     FILE(QPRINT)
MONMSG      MSGID(CPF7304)
DLYJOB      DLY(1)
/*   DSPPGM を印刷               */
DSPPGM      PGM(&LIB/&PGM) OUTPUT(*PRINT) DETAIL(*SRVPGM)
MONMSG      MSGID(CPF9811)
/*      スプール属性獲得  */
CALL        PGM(SPL000) PARM(&SPLF &JOB &USER &JOBNM &SPLNM)
MONMSG      MSGID(CPF3344)
CPYSPLF     FILE(&SPLF) TOFILE(PG012W) +
            JOB(&JOBNM/&USER/&JOB) SPLNBR(&SPLNM) TOMBR(&MBR)
MONMSG      MSGID(CPF3344)
```

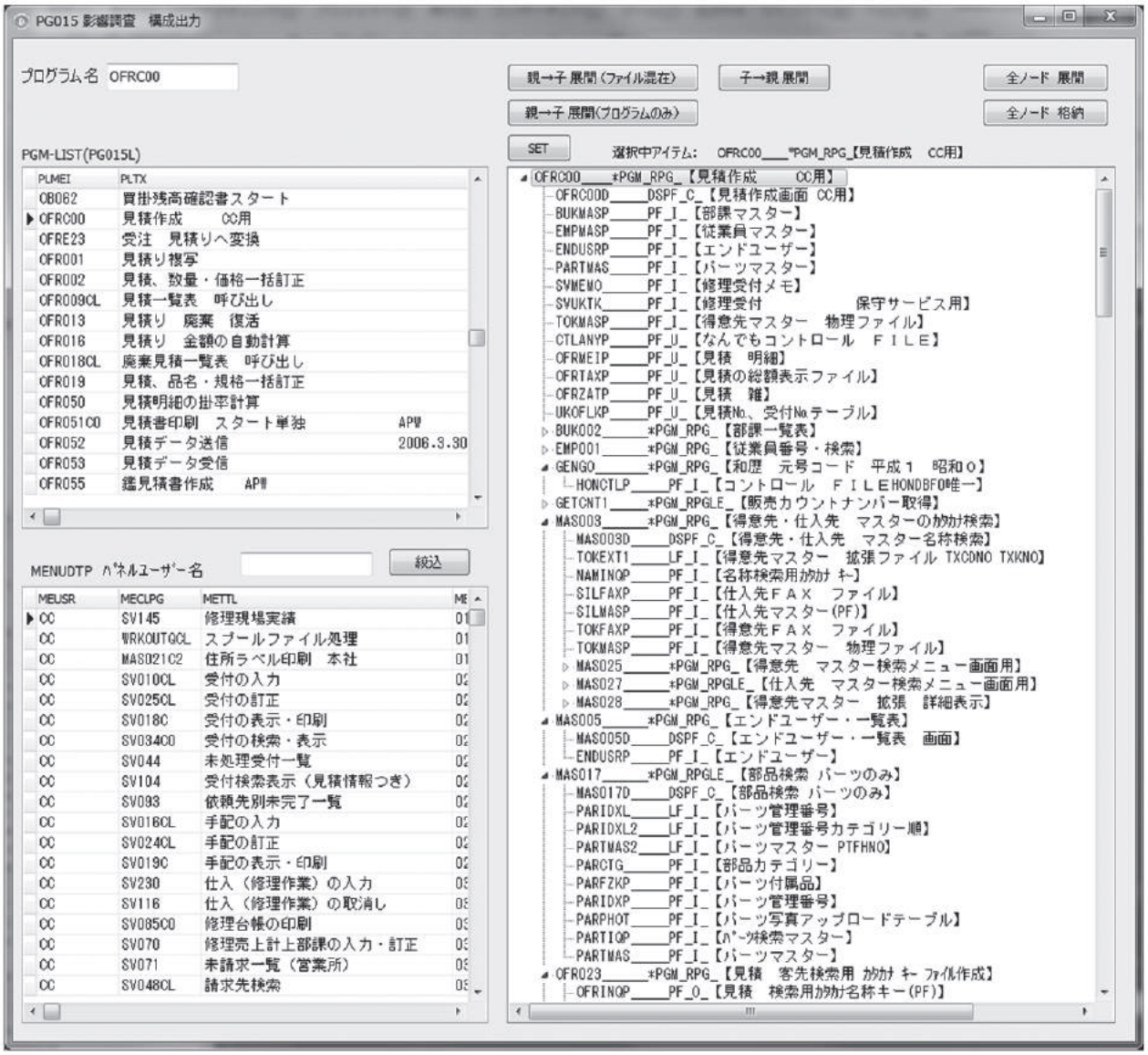
それぞれの一時ファイルをから取り出したオブジェクト名称などの情報をツリーファイルに書き出す。

ツリーファイル(PFソース)

```
                            UNIQUE
R PG015TRR                  TEXT('PGM-TREE')

  PROMEI        10          COLHDG(' 親プログラム名 ')
  PROTP          8          COLHDG(' 親タイプ ')
  PROZK         10          COLHDG(' 親属性 ')
  PROTX         500         COLHDG(' 親テキスト ')
  PROACS          6         COLHDG(' 親最終アクセス日 ')

  PRFMEI        10          COLHDG(' 使用ファイル ')
  PRFZK         10          COLHDG(' 使用ファイル属性 ')
  PRFUI          1          COLHDG(' 使用ファイル用途 ')
  PRFTX         500         COLHDG(' 使用 ﾌｧｲﾙﾃｷｽﾄ')

  PRKMEI        10          COLHDG(' 子プログラム名 ')
  PRKTP          8          COLHDG(' 子タイプ ')
  PRKZK         10          COLHDG(' 子属性 ')
  PRKTX         500         COLHDG(' 子テキスト ')
  PRKACS          6         COLHDG(' 子最終アクセス日 ')

  PRUEMP          4         COLHDG(' 更新者 ')
  PRUTIME        13         COLHDG(' 更新日時 ')
  PRUDSP         10         COLHDG(' 更新端末 ')

K PROMEI
K PRKMEI
K PRFMEI
```

図8

画面イメージ



展開したデータは、親のノードの下位に、
DSPF、参照ファイル、入力ファイル、更新ファイル、呼び出しプログラムの順に展開される。
階層構造が視覚的に分かりやすくなるように各ノードをパディングしている。

ツリーの展開表示部分に使用しているコンポーネントは、TQuery、TDataSource、TtreeViewの三点。

図8-2

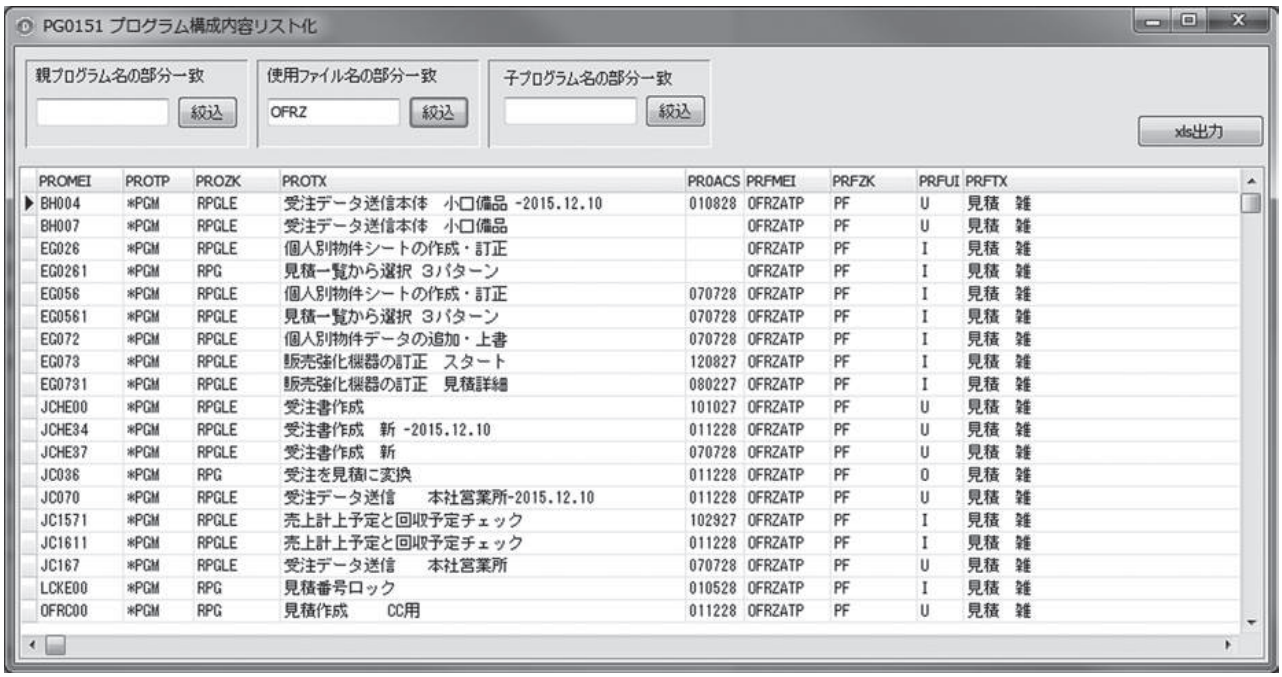
ツリーファイルの親子関係を展開する。(Delphiソース)

```
procedure TForm1.Button1_1Click(Sender: TObject);
// ■親→子に展開。 ファイル混在
var
  NODE_WRD: string;
  OYA_ND: TTreeNode;
  NX_ND: TTreeNode;
  CH_ND: TTreeNode;
  A_POSI: integer;
  PGM_STR: string;
  OBJ_NAME: string;
label tag_end_btn1_1;

begin
  // ツリーのアイテムをクリアしておく。
  TreeView1.Items.Clear;
  // 親プログラム名が入力されていなかったら終了
  if Edit1.Text = '' then
  begin
    goto tag_end_btn1_1
  end;
  // STEP1  親が一致するノードが一つでもあるか調べる。
  Query3.Close;
  Query3.SQL.Clear;
  Query3.SQL.Add('SELECT *');
  Query3.SQL.Add('FROM HONPRGO/PG015TR');
  Query3.SQL.Add('WHERE TRIM(PROMEI) = ' + '''' + Trim(Edit1.Text) + '''');
  Query3.Open;
  Query3.First;
  // 一つ目、親が一致したら、一件だけ親ノードを書く。
  if Trim(Query3.FieldByName('PROMEI').AsString) = Trim(Edit1.Text) then
  begin
    NODE_WRD := Query3.FieldByName('PROMEI').AsString + '___' +
      Query3.FieldByName('PROTP').AsString + '_' + Query3.FieldByName('PROZK')
      .AsString + '_' + '【' + Query3.FieldByName('PROTX').AsString + '】';
    TreeView1.SetFocus;
    OYA_ND := TreeView1.Items.Add(nil, NODE_WRD);
    OYA_ND.Selected := True;
  end;
  // 一致しなかったら終了する。
  if Trim(Query3.FieldByName('PROMEI').AsString) <> Trim(Edit1.Text) then
  begin
    goto tag_end_btn1_1
  end;
```

図8-3

```
  // ツリーにフォーカス
  TreeView1.SetFocus;
  // 全て展開する
  TreeView1.Selected.Expanded := True;
  // 最初のノードを選択
  TreeView1.Select(TreeView1.Items.GetFirstNode);
  OYA_ND := TreeView1.Selected;
  while OYA_ND <> nil do
  begin
    // 子のノードが存在するかチェックする
    CH_ND := TreeView1.Selected.getFirstChild;
    if CH_ND = nil then
    begin
      A_POSI := AnsiPos('__', TreeView1.Selected.Text);
      PGM_STR := copy(TreeView1.Selected.Text, 1, A_POSI - 1);
      // 現在選択ノードの次のノードNX_NDを記憶しておく
      NX_ND := TreeView1.Selected.GetNext;
      Query3.Close;
      Query3.SQL.Clear;
      Query3.SQL.Add('SELECT *');
      Query3.SQL.Add('FROM HONPRGO/PG015TR');
      Query3.SQL.Add('WHERE TRIM(PROMEI) = ' + '''' + Trim(PGM_STR) + '''');
      Query3.SQL.Add('ORDER BY PRKMEI,PRFUI,PRFZK');
      Query3.Open;
      Query3.First;
      while not(Query3.Eof) do
      begin
        if TreeView1.Selected <> nil then
        begin
          // オブジェクト名の桁を揃えてからノードを追加する。
          OBJ_NAME := Query3.FieldByName('PRFMEI').AsString +
            Query3.FieldByName('PRKMEI').AsString;
          OBJ_NAME := OBJ_NAME + StringOfChar('_', 10 - length(OBJ_NAME));
          NODE_WRD := OBJ_NAME + '__' + Query3.FieldByName('PRFZK').AsString +
            Query3.FieldByName('PRKTP').AsString + '_' +
            Query3.FieldByName('PRFUI').AsString + Query3.FieldByName('PRKZK')
            .AsString + '_' + '【' + Query3.FieldByName('PRFTX').AsString +
            Query3.FieldByName('PRKTX').AsString + '】';
          TreeView1.Items.AddChild(TreeView1.Selected, NODE_WRD);
          // 親ノードを展開
          TreeView1.Selected.Expanded := True;
          IF TreeView1.Selected.LEVEL > 50 THEN
          begin
            showmessage
              ('STOP NodeLevel Over 50  this routine is loop saspect ');
            goto tag_end_btn1_1;
          end;
        end;
        Query3.Next;
      end;
      TreeView1.Select(NX_ND);
    end;
    TreeView1.Select(OYA_ND);
    OYA_ND := TreeView1.Selected.GetNext;
    TreeView1.Select(OYA_ND);
  end;
tag_end_btn1_1:
```

図9



PROMEI	PROTP	PROZK	PROTX	PROACS	PRFMEI	PRFZK	PRFUI	PRFTX
BH004	*PGM	RPGLE	受注データ送信本体　小口備品 -2015.12.10	010828	OFRZATP	PF	U	見積　雑
BH007	*PGM	RPGLE	受注データ送信本体　小口備品		OFRZATP	PF	U	見積　雑
EG026	*PGM	RPGLE	個人別物件シートの作成・訂正		OFRZATP	PF	I	見積　雑
EG0261	*PGM	RPG	見積一覧から選択 3パターン		OFRZATP	PF	I	見積　雑
EG056	*PGM	RPGLE	個人別物件シートの作成・訂正	070728	OFRZATP	PF	I	見積　雑
EG0561	*PGM	RPGLE	見積一覧から選択 3パターン	070728	OFRZATP	PF	I	見積　雑
EG072	*PGM	RPGLE	個人別物件データの追加・上書	070728	OFRZATP	PF	I	見積　雑
EG073	*PGM	RPGLE	販売強化機器の訂正　スタート	120827	OFRZATP	PF	I	見積　雑
EG0731	*PGM	RPGLE	販売強化機器の訂正　見積詳細	080227	OFRZATP	PF	I	見積　雑
JCHE00	*PGM	RPGLE	受注書作成	101027	OFRZATP	PF	U	見積　雑
JCHE34	*PGM	RPGLE	受注書作成　新 -2015.12.10	011228	OFRZATP	PF	U	見積　雑
JCHE37	*PGM	RPGLE	受注書作成　新	070728	OFRZATP	PF	U	見積　雑
JC036	*PGM	RPG	受注を見積に変換	011228	OFRZATP	PF	O	見積　雑
JC070	*PGM	RPGLE	受注データ送信　本社営業所-2015.12.10	011228	OFRZATP	PF	U	見積　雑
JC1571	*PGM	RPGLE	売上計上予定と回収予定チェック	102927	OFRZATP	PF	I	見積　雑
JC1611	*PGM	RPGLE	売上計上予定と回収予定チェック	011228	OFRZATP	PF	I	見積　雑
JC167	*PGM	RPGLE	受注データ送信　本社営業所	070728	OFRZATP	PF	U	見積　雑
LCKE00	*PGM	RPG	見積番号ロック	010528	OFRZATP	PF	I	見積　雑
OFRC00	*PGM	RPG	見積作成　CC用	011228	OFRZATP	PF	U	見積　雑

図10

ソースの種類に合わせてコンパイルする(CLソースの一部)

```
IF (&TYPE *EQ 'RPG  ') DO
   CRTRPGPGM PGM(&LIB/&PGM) SRCFILE(&LIB/QDDSSRC)
             MONMSG MSGID(CPF1338) EXEC(GOTO CMDLBL(ERR))
ENDDO
IF (&TYPE *EQ 'DSPF ') DO
   CRTDSPF    FILE(&LIB/&PGM) SRCFILE(&LIB/QDDSSRC)
             MONMSG MSGID(CPF1338) EXEC(GOTO CMDLBL(ERR))
ENDDO
IF (&TYPE *EQ 'CLP  ') DO
   CRTCLPGM   PGM(&LIB/&PGM) SRCFILE(&LIB/QDDSSRC)
             MONMSG MSGID(CPF1338) EXEC(GOTO CMDLBL(ERR))
ENDDO
IF (&TYPE *EQ 'RPGLE') DO
   CRTBNDRPG PGM(&LIB/&PGM) SRCFILE(&LIB/QDDSSRC) SRCMBR(&PGM)
DBGVIEW(*SOURCE)
             MONMSG MSGID(CPF1338) EXEC(GOTO CMDLBL(ERR))
```

ゴールド賞

Windows Like 5250への道のり
―さまざまな場面で使えるDelphiおよびDelphi/400

小山 祐二 様
澁谷工業株式会社
経営情報システム部
課長代理

澁谷工業株式会社
http://www.shibuya.co.jp/

パッケージプラントを主力製品とする東証・名証１部上場の機械メーカー。とくに国内外の大手飲料メーカーに採用されているボトリングシステム製造では、世界トップの地位を確立している。近年では無菌化などの技術力を活かし、再生医療事業も積極的に展開している。

1.はじめに

当社は 1931 年創立、1949 年設立の会社である。今日まで多くのお客様に支えられ、2016 年に創立 85 周年を迎えた。創立以来、カスタマーファースト（お客様第一主義）を貫き、お客様のニーズに合わせたパッケージングプラントを、ターンキー（すぐに稼働できる状態）で提供するビジネスを主体としている。また最近では、再生医療分野にも進出している。

当社のホスト・コンピュータの変遷は、1972 年に S/32 を導入したことから始まる。その後、各種モデルを経て、現在のPureFlex System導入に至る。【図1】

その間、多くの基幹システムをキーボード操作入力（以下、CUI）主体の 5250 画面（以下、5250）で自社開発してきた。

近年における当社の基幹システムは、主に Delphi および Delphi/400 で構築しているが、膨大な旧資産の関係上、現在も多くの基幹システムが 5250 で稼働している。それに加え、今後も運用・開発面で 5250 を利用し続けることになる。【図 2】

2.5250の「操作性」評価

現在のインターフェースは、マウス操作入力（以下、GUI）とタッチ操作入力（以下、NUI）が主流である。

そこで、当社のエンドユーザー部門およびシステム部門のメンバー（以下、5250 利用者）の協力のもと、5250 の「操作性」に関してアンケートを実施した。

IBM i は、一般にユーザー評価が非常に高い（『日経コンピューター』 顧客満足度調査　ミッドレンジサーバー部門 18 年連続 1 位）。しかし、当社の 5250 利用者における「操作性」評価の結果とは反比例することがわかった。【図 3】一般ユーザーにおける 5250 の「操作性」評価も、同様だと推測する。

3.5250の「操作性」に対する要望

5250 利用者に、5250 の「操作性」に関する要望をアンケートし、以下にまとめた。【図 4】

(A) マウスホイールによる画面スクロール
(B) 右クリックによるコピー & 貼り付け　等
(C) スクロールバーによる画面スクロール
(D) ショートカットによるコピー & 貼り付け　等
(E) チェックボックスによる項目選択
(F) ダブルクリックによる実行キー打鍵
(G) ラジオボタンによる項目選択
(H) メニューバーによるプログラム（以下、PGM）実行
(I) ダブルクリックによるメニュー PGM 実行
(J) ダブルクリックによる機能キー打鍵

アンケート実施時点では、具体的な対

図1　ホスト・コンピュータの変遷

当社ホストコンピューターの変遷

1972年	…	1979年	…	1984年	…	1989年	…	1991年	…
S/32	…	S/34	…	S/38	…	AS400 (B50)	…	AS400 (D70)	…

1993年	…	1998年	…	2003年	…	2008年	…	2013年	…
AS400 (F70)	…	AS400 (S30)	…	iSeries (i825)	…	Power Systems (9409-M50)	…	Pure Flex System	…

図2　基幹システム構築状況

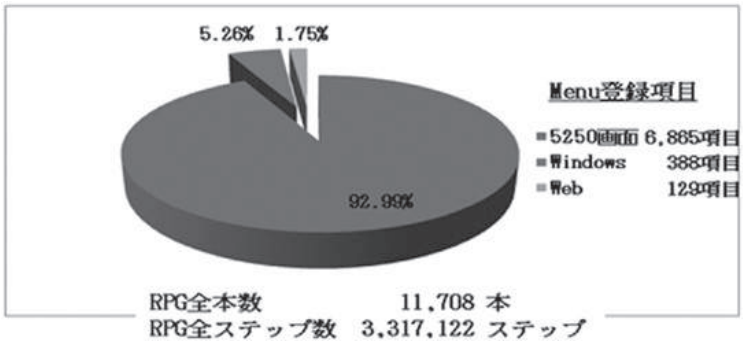


図3　5250「操作性」評価

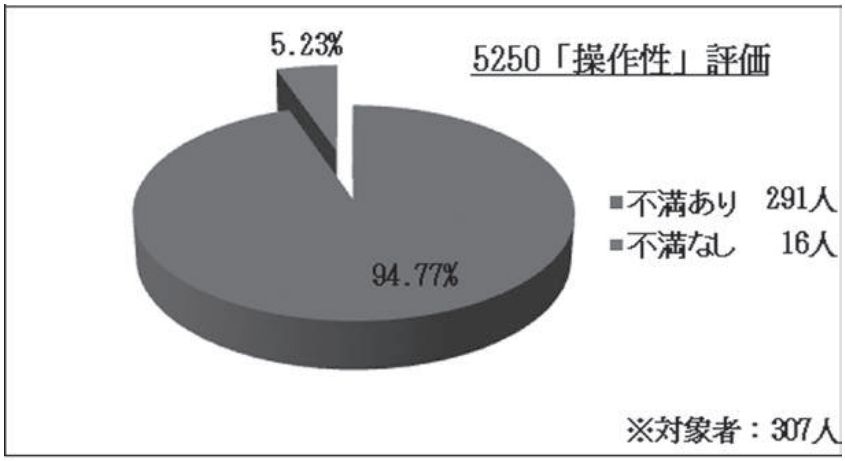
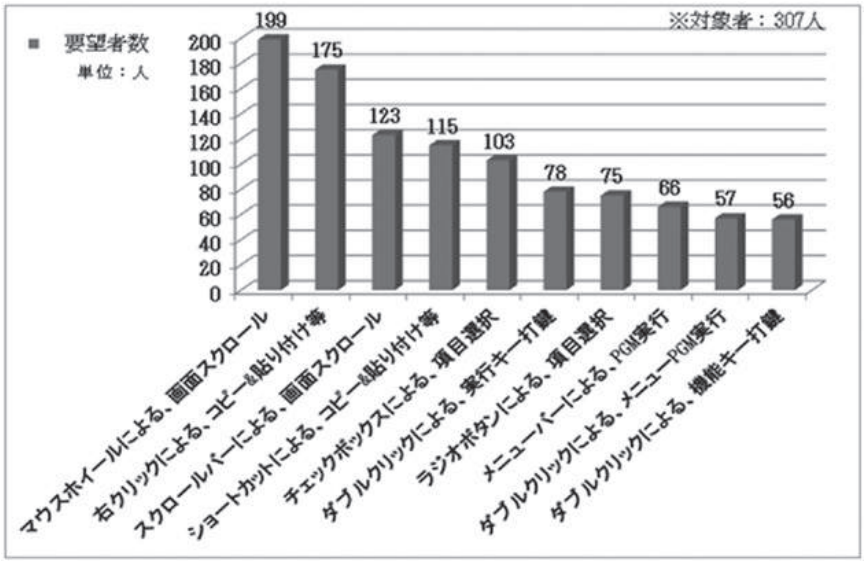


図4　5250「操作性」に対する要望1



応策はなかった。しかしゼロベース思考で、「5250 は CUI」という既成概念を捨て、「5250 でも GUI」との仮説思考をもつ。そこからポジティブ思考で、「Windows Like 5250」を模索することになった。

4.Windows Like 5250への道

最初に、各種既存機能（5250、RPG、画面ファイル等)を調査した。その結果、(C)（E)（F)（G)（H)（I)（J）は実現可能であるが、これらの説明は割愛する(必要であれば別途、問い合わせていただきたい)。本稿では、(B)（D)（A）の詳細を述べる。

4.1 (B) 右クリックによるコピー & 貼り付けと (D) ショートカットによるコピー & 貼り付けの設定方法

5250 既存機能では「ポップアップ・キーパッドの設定」【図 5】で、5250 右クリック時のメニュー設定が可能である。しかし初期値では、(B) は使えない。

調査の結果、「ユーザー定義」に以下の設定変更 ／追加で、(B) を実現した。

設定変更

① NumberOfPads = 3（新しいパッド 3 作成：2 → 3 へ設定変更）

設定追加

② NumberOfRowsPad_3 = 4
（パッド 3：行指定）
③ NumberOfColsPad_3 = 1
（パッド 3：列指定）
④ POP3-1-1 =［edit-copy］
（パッド 3：コピー機能割り当て）
⑤ POP3-1-2 =［edit-cut］
（パッド 3：切り取り機能割り当て）
⑥ POP3-1-3 =［edit-paste］
（パッド 3：貼り付け機能割り当て）
⑦ POP3-1-4 =［edit-clear］
（パッド 3：クリア機能割り当て）

また 5250 既存機能「キーボードの設定」【図 6】の「ユーザー定義」で、キーごとに各種機能の割り振りが可能である。調査の結果、「ユーザー定義」に以下の設定追加で、(D) を実現した。

設定追加

① C-KEY47 =［edit-cut］
② C-KEY48 =［edit-copy］
③ C-KEY49 =［edit-paste］

4.2 (B) 右クリックによるコピー & 貼り付けと (D) ショートカットによるコピー & 貼り付けの PC への展開方法

当社は、全国約 2000 台の PC で 5250 を利用している。そのため、これらの機能をどのように導入するかを検討した。

まず、他企業に問い合わせてみた。その結果、(B) の認識は低いが、(D) は高かった。そこでそれらの導入方法を問い合わせてみたが、有益な情報は得られなかった。

思考錯誤の末､IBM i の IFS(Integrated File System：IBM i の UNIX 互換ファイルシステム）を利用した。そこに設定変更用の Delphi/400 PGM を配置し、5250 利用者が 5250 メニューからその PGM を実行することで、該当 PC の設定変更を実現した。【図 7】【図 8】【図 9】【図 10】【図 11】(*1)

また設定変更時、各種情報を取得した。これは Delphi/400 を利用すれば、まったく問題なく取得可能である。そしてテーブルにトリガー設定を組み込み、エラー時には即座に電子メール配信する仕組みを構築した。【図 12】【図 13】【図 14】

(*1)　5250 画面から Delphi/400 プログラムを実行する方法は「Delphi/400 および Delphi/400 を利用したオンライン個人メニューの構築」（ミガロ .テクニカルレポート　No.7）を参照

4.3 (A) マウスホイールによる画面スクロール

5250 をスクロールする場合、Page Up/Page Down キーの打鍵が必要である。つまり、(A) を実現するには、何らかの方法で 5250 に Page Up ／ Page Down キー打鍵の代替が必要である。

Windows には、常駐 PGM がある。これを利用してマウスホイール操作を監視すれば、実現可能と考えた。後は、どのようにマウスホイール操作を監視するかである。

その後、Windows のメッセージ機能を知ることになった。それは、キーボードやマウスなどの操作情報を OS と PGM 間で受け渡す機能である。

調査後さらに、このメッセージを監視できるフック機能 (*2) について知った。その機能の実装は DLL 化する必要があるが、幸いなことに Delphi や Delphi/400 の開発環境でも、DLL 作成機能が備わっている。

そこで、以下のプロセスを実現する常駐 PGM を作成することで、(A) を実現した。【図 15】【図 16】【図 17】

① 「WH_GETMESSAGE」で、マウスホイール操作情報取得（DLL）
② 「GetforegroundWindow」で、最前面 Window 情報を取得
③　② が 5250 時、Page Up ／ Page Down キー打鍵機能送信（要 IME 機能考慮）
④　上記をスタートアップ登録

(補足)

マウスホイール操作では、「WH_MOUSEWHEEL」がある。これでマウスホイール操作情報は収集可能である。しかし筆者の知る限り、ScrollUp ／ Down が判断できない。多くのユーザーはここで挫折していると推測する。

(*2) 参考文献：Delphi Library［Mr. XRAY］ http://mrxray.on.coocan.jp/index.htm

5.取り組み実施後

取り組み実施後、先の要望に関する利用状況を確認した。【図 18】のとおり、かなり成果があったと考えている。

6.おわりに

IBM i は、安全性・堅牢性などで非常に評価の高いサーバーであるのは、多くのユーザーが認識している。しかしその評価に反比例し、現在の IBM i はさまざまな理由で属人化が進んでいる。また 5250 の「操作性」から、IBM i は古いマシンだと勘違いするユーザーも少なく

図5　ポップアップ・キーパッドの設定

図6　キーボードの設定

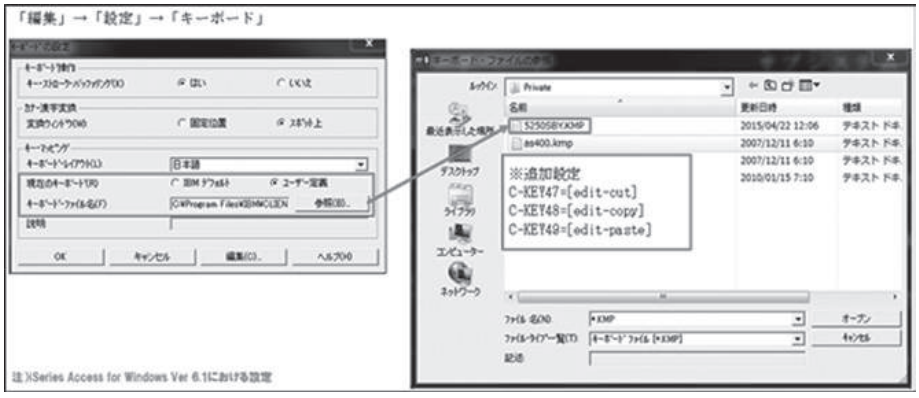
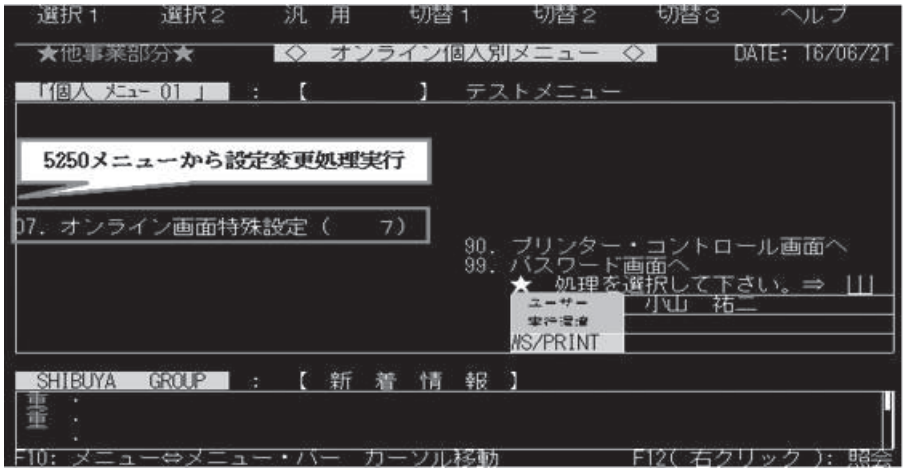


図7　5250メニュー例



ない。

確かにIBM i情報の少なさは、ユーザー共通の悩みだと思う。そのためユーザーは、IBM i関連で実現したいさまざまなプロセスを断念していると推測する。まさに、「IBM iが宝の持ち腐れ」になっていると感じている。非常にもったいない話である。

しかし、Delphi および Delphi/400は情報も多く、IBM i連携も非常に簡単である。また今回紹介した例や各種IBM i運用など、さまざまなプロセスで利用可能である。

今後も、各種プロセスを実現するツールとして、Delphi および Delphi/400を利用していきたい。

最後に、IBM i Access Client Solutionsについて。Windows 7の延長サポートは、2020年1月14日に終了する。しかし既存の5250は、Windows 10に未対応となり、IBM i Access Client Solutionsを利用することになる【図19】。これは既存の5250とほぼ同機能だが、今回紹介した(A)(B)(D)機能は標準で備わっている。

今回の機能を導入したいが、Windowsには詳しくない方は、早急にIBM i Access Client Solutionsの検証（データ転送、ODBC等）を始めるよう推奨する。

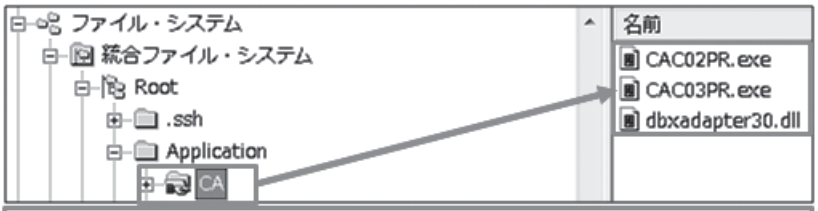
M

図8　CL PGM例

```
/*****************/
/*  IFS の接続 */
/*****************/

         CHGVAR     &W_CMD1 ('NET USE ¥¥' || &W_SYS |< '¥CA /USER:' || +
                      'A¥' ||&W_USER |<  '  || &W_PASS)
         STRPCCMD   PCCMD(&W_CMD1) PAUSE(*NO)

/************************/
/* 設定変更ＰＧＭ実行 */
/************************/

         CHGVAR     &W_CMD2 ('¥¥' || &W_SYS |< '¥CA¥CAC02PR.EXE ' || +
                    &W_JOB || ' 1   || &W_USER |< ' '  || &W_PASS  |< +
                      || &W_SYS |< ' '  || &W_USER2)
         STRPCCMD   PCCMD(&W_CMD2) PAUSE(*NO)
```

任意の文字が必要

図9　Delphi IFS配置例およびPGM実行制約条件



【ユーザーアカウント制御による、PGM実行制約条件】

① 設定変更PGM実行は、管理者権限が必要
② ユーザーアカウント制御画面が表示され、PC管理者権限パスワードが必要
③ Windowsセキュリティー画面が表示され、5250サインオンパスワードが必要

図10　DelphiおよびDelphi/400　コーディング例1

```
begin
  //1. パラメータ取得
  F_Session  := CAC03BFOM.Get_Parameter1() ;
  F_Runkubun := CAC03BFOM.Get_Parameter2() ;
  F_Host     := CAC03BFOM.Get_Parameter3() ;
  F_Directory:= CAC03BFOM.Get_Parameter4() ;
  F_User     := CAC03BFOM.Get_Parameter5() ;
  F_Password := CAC03BFOM.Get_Parameter6() ;
  F_User2    := CAC03BFOM.Get_Parameter7() ;

  //2. 実効PCのWindowsバージョン取得
  F_Winversion := CAC03BFOM.Get_Winversion() ;

  if (F_Winversion <> '8.1')  then
  begin

    MessageDlg('このPCのWindowsバージョンには対応していません。', mtError, [mbYes], 0);
    MessageDlg('「オンライン画面特殊設定」処理をキャンセルしました！', mtInformation, [mbYes], 0);
    exit;

  end;

  //3. コンピュータ名取得
  F_ComputerName := CAC03BFOM.Get_ComputerName(
  //4. IBM i接続
  CAC03BFOM.Str_Connection(F_User,F_Password);

  BringWindowtotop(F_Handle);

  //5. 処理済みチェック
  if CAC03BFOM.Chk_processed(F_ComputerName ,F_Session) then
  begin

    MessageDlg(                                                    rning, [mbYes], 0)
    MessageDlg(
    //IBM i切
    CAC03BFOM.
    exit ;

  end;

  //6. 管理者として実行
  CAC03BFOM.Run_administrator(F_Session,F_Runkubun,F_Directory,F_User,F_Password,F_Host,F_User2);

  //7. IBM i切断
  CAC03BFOM.End_Connection();

end.
```

※Delphi400にて接続

A54001

【設定ファイル変更プロシージャー】
※5250設定値を変更するため、必ず当運用決定者の責任の上、実行する事
（当社は、この処理中で、バックアップを作成している）

図11　DelphiおよびDelphi/400　コーディング例2

```
procedure TCAC03BFUM.Run_administrator(var F_Session  ,
                                       F_Runkubun ,
                                       F_Directory,
                                       F_User     ,
                                       F_Password ,
                                       F_Host     ,
                                       F_User2    :String) ;
var
  sei      : TShellExecuteInfoA ;
begin
 ZeroMemory(@sei, SizeOf(sei));
 sei.cbSize := SizeOf(sei);
 sei.fMask := SEE_MASK_FLAG_DDEWAIT or SEE_MASK_FLAG_NO_UI;
 sei.lpVerb := 'runas';
 sei.lpFile := PAnsiChar('CAC02PR.exe');
 sei.lpParameters:=PAnsiChar(F_Session  +
                             ' '        +
                             F_Runkubun +
                             ' '        +
                             F_User     +
                             ' '        +
                             F_Password +
                             ' '        +
                             F_Host     +
                             ' '        +
                             F_User2    );
 sei.lpDirectory:=PAnsiChar(F_Directory);
 sei.nShow := SW_SHOW ;
 try

   ShellExecuteEx(@sei) ;

 finally

 end;
end;
```

※管理者としてPGM実行

※Delphi400にてRPG実行

Call4001

図12　テーブル設定変更情報

COMPUTER	SESSION	C.A VER	WIN VER	区画名	.PMP ERR	.KMP ERR	実行者
COM01	SES01	5			0	0	
COM02	SES02	5			0		
COM03	SES03	5			0		
COM04	SES04	5	5.1	IBMi01	0	0	社員04
COM05	SES05	5	5.1	IBMi01	0	0	社員05
COM06	SES06	5	5.1	IBMi01	0		社員06
COM07	SES07	5	5.1	IBMi01	0		社員07
COM08	SES08	5	5.1	IBMi01	0		社員08
COM09	SES09	5	5.1	IBMi01	0	0	社員09
COM10	SES10	5	5.1	IBMi01	1	0	社員10

ポップアップキーバットの設定
エラーフラグ

キーボードの設定
エラーフラグ

エラー時
メールにて通知

図13　CL PGMからJava(電子メール配信)実行例

```
 桁 . . . . . :   1  71          走査検索
SEU==>                                                          SMP77AP
FMT **  ...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7
0026.00 /*******************/
0027.00 /* CCSID ⇒ 5035 */
0028.00 /*******************/
0029.00
0030.00           CHGJOB     CCSID(5035)
0031.00
0032.00 /***************/
0033.00 /* MAIL 送信 */
0034.00 /***************/
0035.00
0036.00           RUNJVA     CLASS(jp.co.shibuya.isd.smp77a) +
0037.00                      PARM(&SMTP &MAIL1 &MAIL2 &NAME &TITLE &TEXT +
0038.00                           &FOLDER &FLNM &DLT) +
0039.00           CLASSPATH('.:+
0040.00                     /jp/co/shibuya/isd/lib/mail.jar:+
0041.00                     /jp/co/shibuya/isd/lib/activation.jar')  +
0042.00           OUTPUT(*NONE)

F3= 終了  F5= 最新表示  F9=ｺﾏﾝﾄﾞの複写  F10=ｶｰｿﾙ移動  F11= 切り替え  F12= 取り消し
F16= 検索の反復    F24= キーの続き
```

図14　Java IFS配置例および配信電子メールイメージ

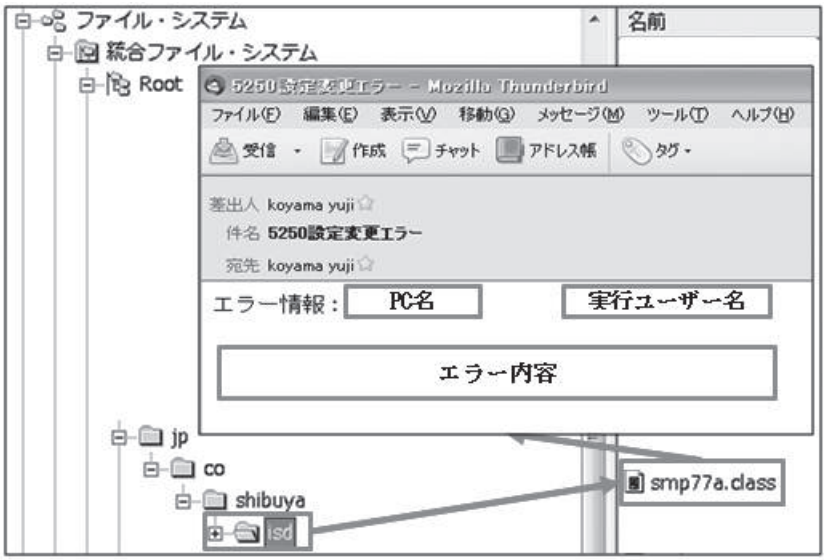


図15　(A)の操作イメージ

図16　フック関数「WH_GETMESSAGE」コーディング例（DLL側）

```
function StartGetMessageHook(HostWnd: HWND; AMsg: Integer): Boolean; stdcall;
var
  F_LMapWnd : THandle;
  F_LpMap   : Pointer;
begin

  Result := False;

  //メモリマップドファイル使用準備
  MapFileMemory(F_LMapWnd, F_LpMap);

  if F_LpMap = nil then
  begin
    F_LMapWnd := 0;
  end
  else
  begin

    //フック情報構造体初期化とフック関数の登録
    pHookInfo(F_LpMap)^.HostWnd    := HostWnd;
    pHookInfo(F_LpMap)^.RevMessage := AMsg;

    //フックをインストール
    //ローカルフックの場合は3番目の引数は0(null)でOK
    //グローバルの場合はDLLのインスタンス
    //4番目はフック対象のスレッドID(システム全体の時は0)
    pHookInfo(F_LpMap)^.HookHandle := SetWindowsHookEx(WH_GETMESSAGE,
                                                       @MyHookProc,
                                                       hInstance,
                                                       0);

    //フック成功
    if (pHookInfo(F_LpMap)^.HookHandle > 0) then
    begin

      Result := True;

    end;

    //メモリマップドファイル使用終了処理
    UnMapFileMemory(F_LMapWnd, F_LpMap);

  end;

  end;
```

図17　5250時、Page Up／Page Downキー打鍵 コーディング例（EXE側）

```
function StartGetMessageHook(HostWnd: HWND; AMsg: Integer): Boolean; stdcall;
var
  F_LMapWnd : THandle;
  F_LpMap   : Pointer;
begin

  Result := False;

  //メモリマップドファイル使用準備
  MapFileMemory(F_LMapWnd, F_LpMap);

  if F_LpMap = nil then
  begin
    F_LMapWnd := 0;
  end
  else
  begin

    //フック情報構造体初期化とフック関数の登録
    pHookInfo(F_LpMap)^.HostWnd    := HostWnd;
    pHookInfo(F_LpMap)^.RevMessage := AMsg;

    //フックをインストール
    //ローカルフックの場合は3番目の引数は0(null)でOK
    //グローバルの場合はDLLのインスタンス
    //4番目はフック対象のスレッドID(システム全体の時は0)
    pHookInfo(F_LpMap)^.HookHandle := SetWindowsHookEx(WH_GETMESSAGE,
                                                       @MyHookProc,
                                                       hInstance,
                                                       0);

    //フック成功
    if (pHookInfo(F_LpMap)^.HookHandle > 0) then
    begin

      Result := True;

    end;

    //メモリマップドファイル使用終了処理
    UnMapFileMemory(F_LMapWnd, F_LpMap);

  end;

  end;
```

図18　各種取り組み実施後の要望利用状況

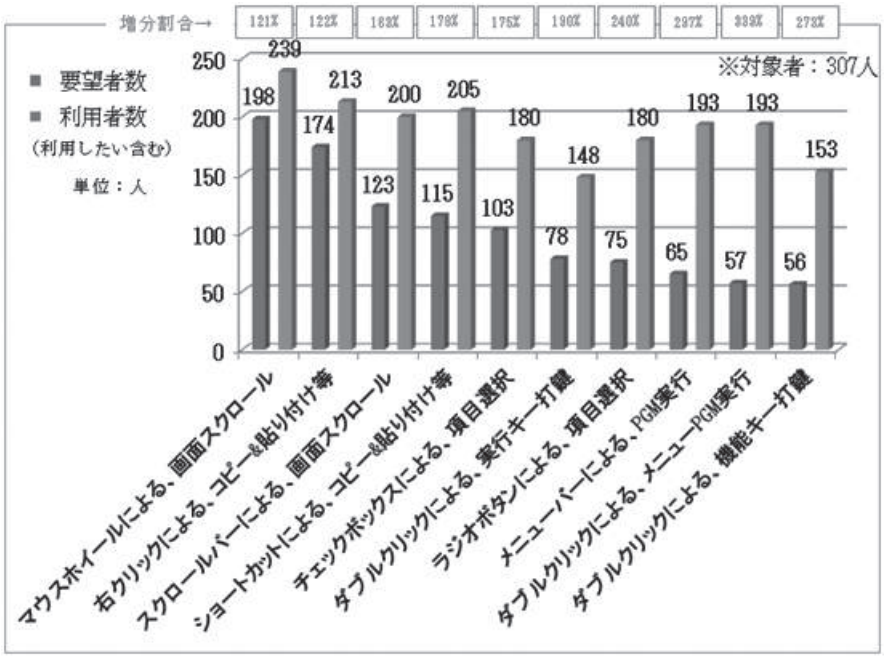
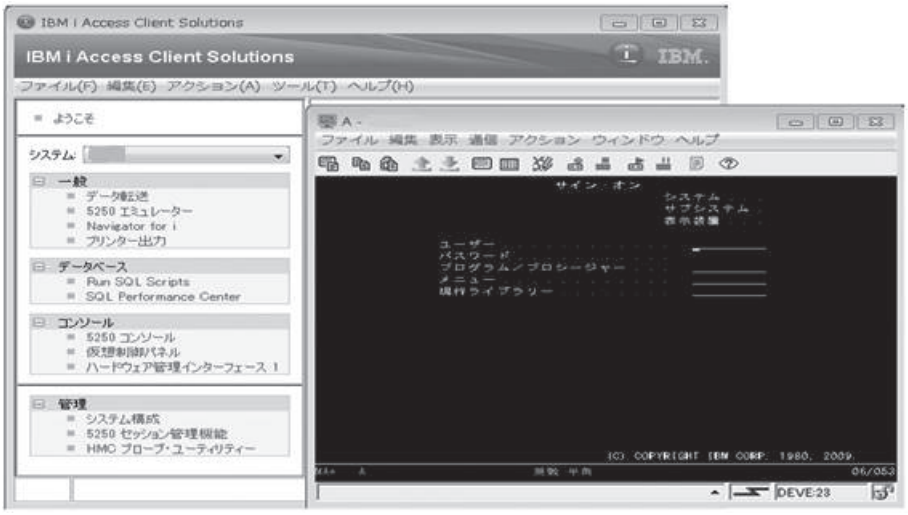


図19　IBM i Access Client Solutions

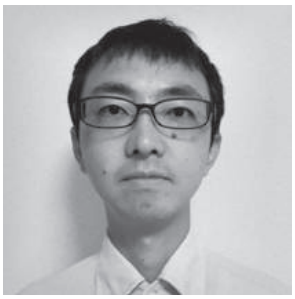


シルバー賞

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
管理部情報システム課
主事



株式会社佐賀鉄工所
http://www.satetsu.co.jp/

昭和 13 年創業。自動車用ボルトを専門領域とするリーディングカンパニーとして、日本はもちろん、海外でも高い評価を得ている。業界でも数少ない「一貫生産方式」を採用。さらに業界屈指の開発・試験設備を保有し、世界の自動車産業を「小さなボルトで大きく」支え続けている。

はじめに

昭和 13 年に創業した佐賀鉄工所は、昭和 20 年代後半より、高品質・高機能のボルトの提供に特化し、現在では日本はもちろん、海外でも高い評価を得るに至った。

この評価を励みに私たち佐賀鉄工所は、これからもボルトの専門技術者集団としてユーザーのニーズを的確に受け止め、最適な締結技術を提供し続けたいと考えている。

引張強さ 1200MPa 以上の性能を備えるボルトは、自動車のエンジン回りでは 40 ～ 50 本が使用され、ミッションなどでも活躍している。この自動車用の高強度ボルトは、高い信頼性を要求されることから、日本では生産するボルトメーカーが数社に限られている。

「高強度ボルトを生産できるかどうかが､ボルトメーカーの技術力を計る尺度」とまで言われる。当社では現在、高強度ボルトの生産が月産数百トンを超えている。

今後、エンジンの高性能化に伴い、さらに高いスペックが要求されている。当社では新材料により、さらに高強度のボルトの開発を進めている。

システム紹介

当社では日立のメインフレームを使用していたが、2011 年より IBM i を使用し始め、基幹システムはすべて IBM i で処理している。

エンドユーザーが利用する処理画面は､そのほとんどが Visual Basic（以下、VB）6 でインターフェースが構築されている。しかし最新 OS の導入に伴い、VB6 の開発および運用が困難になってきた。

そのため代替手段として導入したのが、IBM i と親和性が高く、さまざまな OS やデータベースに対応できる Delphi/400 であった。現在は、VB6 から Delphi/400 への移行を行っている。

プログラム管理ソフト開発の経緯

IBM i でのプログラムソース管理は、アイエステクノポート製の S/D Manager を使用しており、VB6 に関しては当社で開発したプログラム管理ソフトを使用していた。しかし、Delphi/400 のプログラム管理ソフトは未開発だったので、手動でファイルサーバーへのソース管理を行っていた。

Delphi/400 導入直後は、メインで開発する担当者が 1 人だけだったので、さほど問題はなかった。しかし次第に他の情報システム課員も開発する機会が多くなってきたため、プログラムソース管理（排他制御がないことによるバージョン不整合、および人為的ミスによるプログラムソース消失）に不安を抱えるようになっていた。

そこでプログラム管理ソフトを導入することにより、プログラムソース管理を厳格化し、ファイルサーバーにおける一元管理を実現した。Delphi/400 はオ

図1　チェックアウト概要図

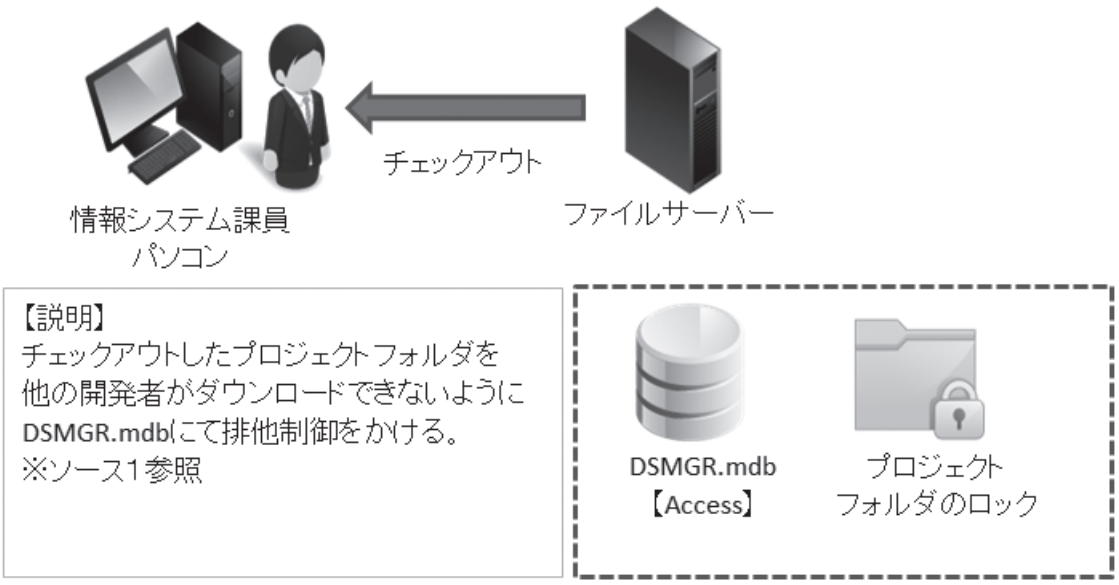
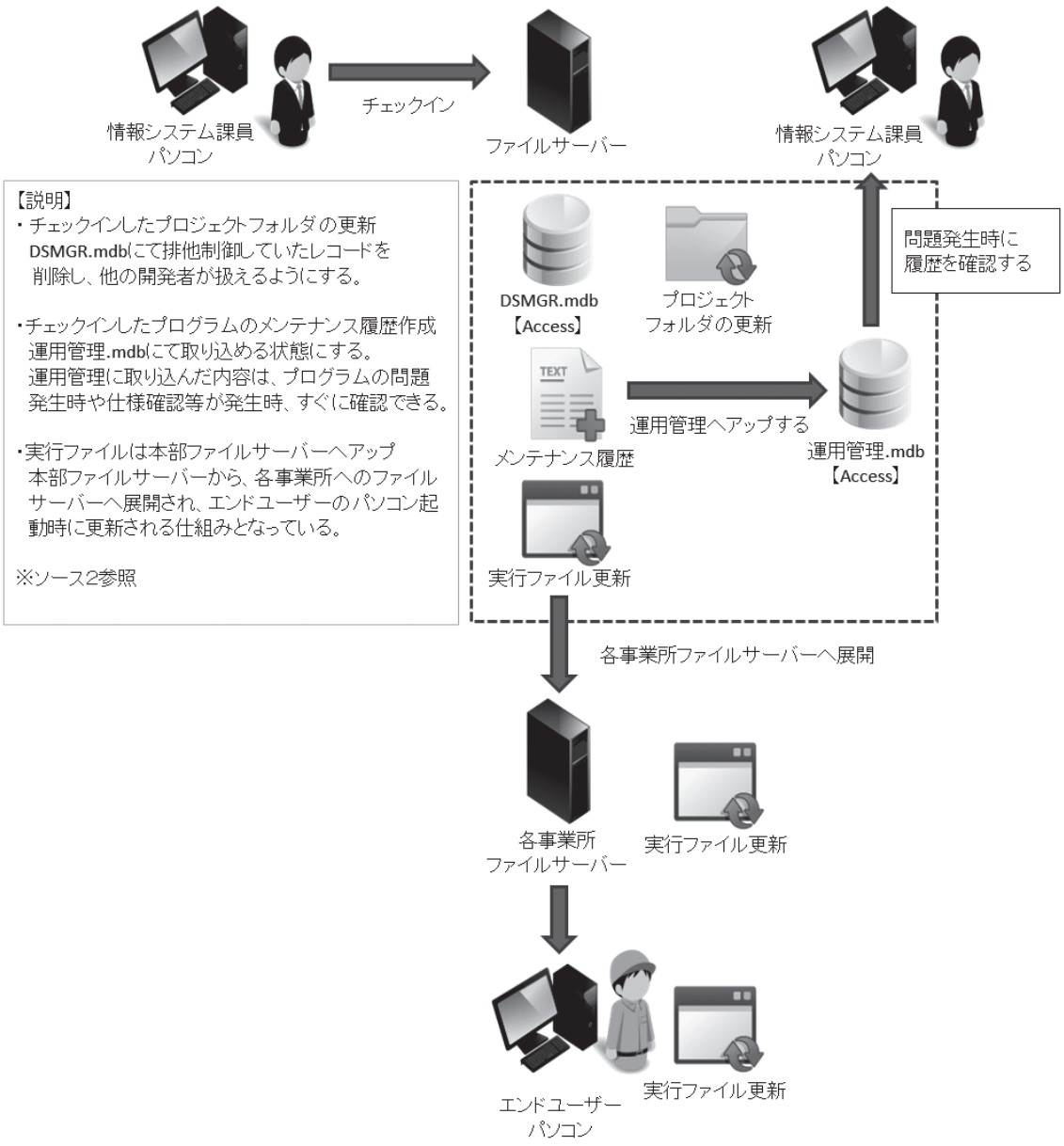


図2　チェックイン概要図



ブジェクト指向言語なので、プログラム開発を標準化すべく、開発した継承元プログラムおよび部品コンポーネントをすぐに開発端末に同期する仕組みを追加した。

Delphiプログラム管理ソフトの開発

要件定義

プログラム管理ソフトの開発に際して､次のシステム要件を決定した（【図 1】～【図 5】､および【ソース 1】【ソース 2】を参照）。

・プログラムソースの一元管理（排他制御）

チェックアウト時、プロジェクトフォルダを他の開発者がダウンロードできないように排他制御を実施する。チェックイン時は、チェックアウトしたプロジェクトフォルダについて排他制御を解除し、他の開発者が扱えるようにする

・修正前のプログラムソースのバックアップ

・継承元および部品コンポーネントの同期機能

チェックアウト処理画面で、継承元プログラムを情報システム課の PC に同期する機能を追加する

・チェックイン時のメンテナンス記録用ファイル作成

チェックインしたプログラムのメンテナンス履歴を作成し、運用管理データベースに取り込む

・プログラムソース開発状況の可視化

運用管理データベースに取り込んだ内容は、プログラムの問題発生時や仕様確認等が発生した場合に、すぐに確認できる体制とする

構築システムの仕様

・プログラムソースの開発状況は、ファイルサーバーの Access で管理する
・プログラムソースの排他制御は、同じくファイルサーバーの Access で管理する
・プログラムソースおよび実行ファイルは、決められたディレクトリへコピーする
・開発端末における開発用ディレクトリ構造は、すべて同じ構成とする

導入効果と今後の展望

現在、VB6 から Delphi/400 への移行も本格的に進み始め、Delphi/400 のプログラムソースの管理が多くなっている。開発したプログラム管理ソフトにより、ソース管理の人為的ミスを防げるようになった。プログラム管理の改善により、プログラム品質向上に少しでもつなげていきたい。

今後は、チェックインしたプログラムの旧バージョンをすぐに取り出せる機能を追加し、デグレ発生による応急処理の一環として機能追加を検討する予定である。

M

図3　チェックアウト処理内容

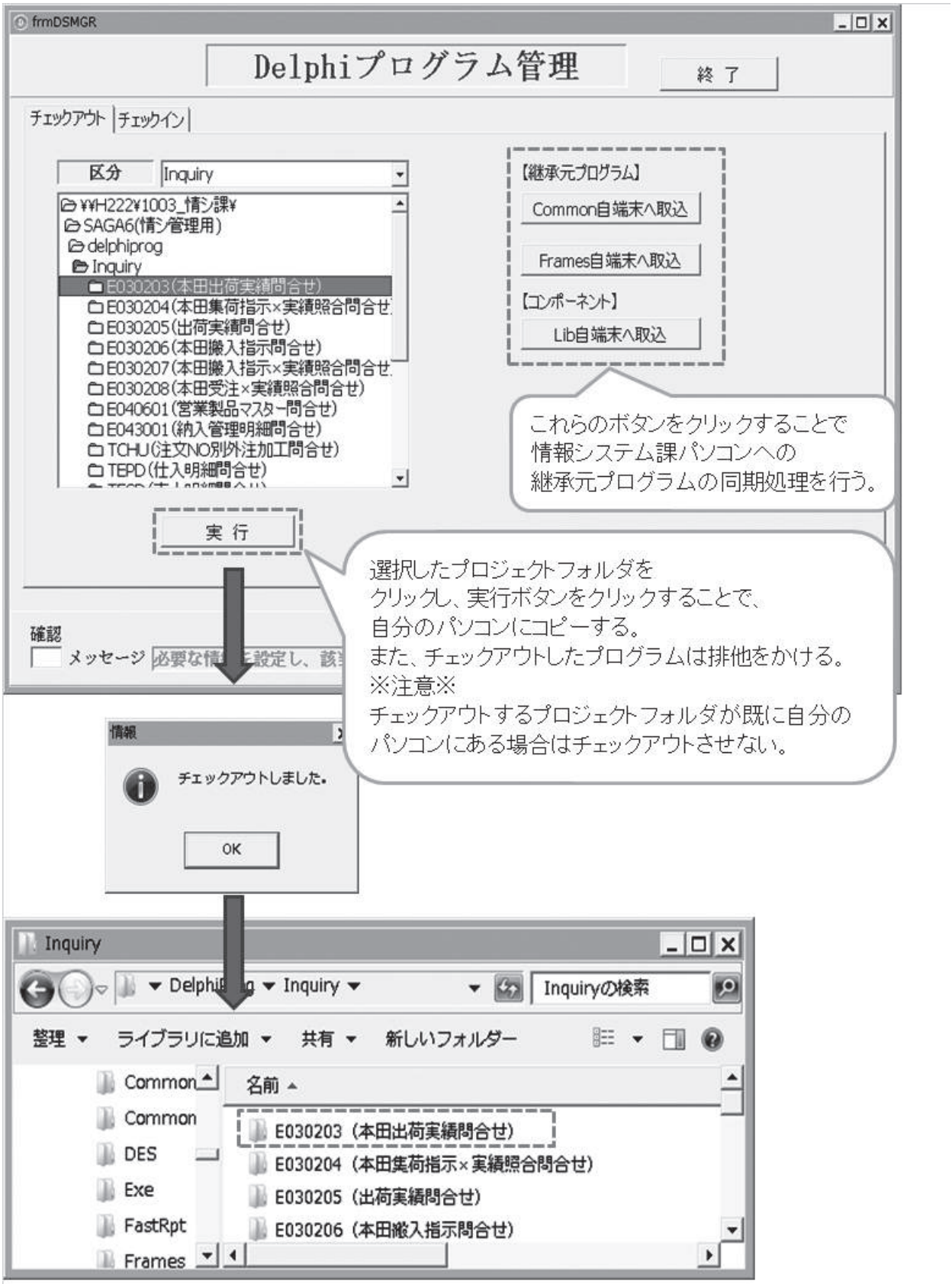


図4　チェックアウト取消処理内容

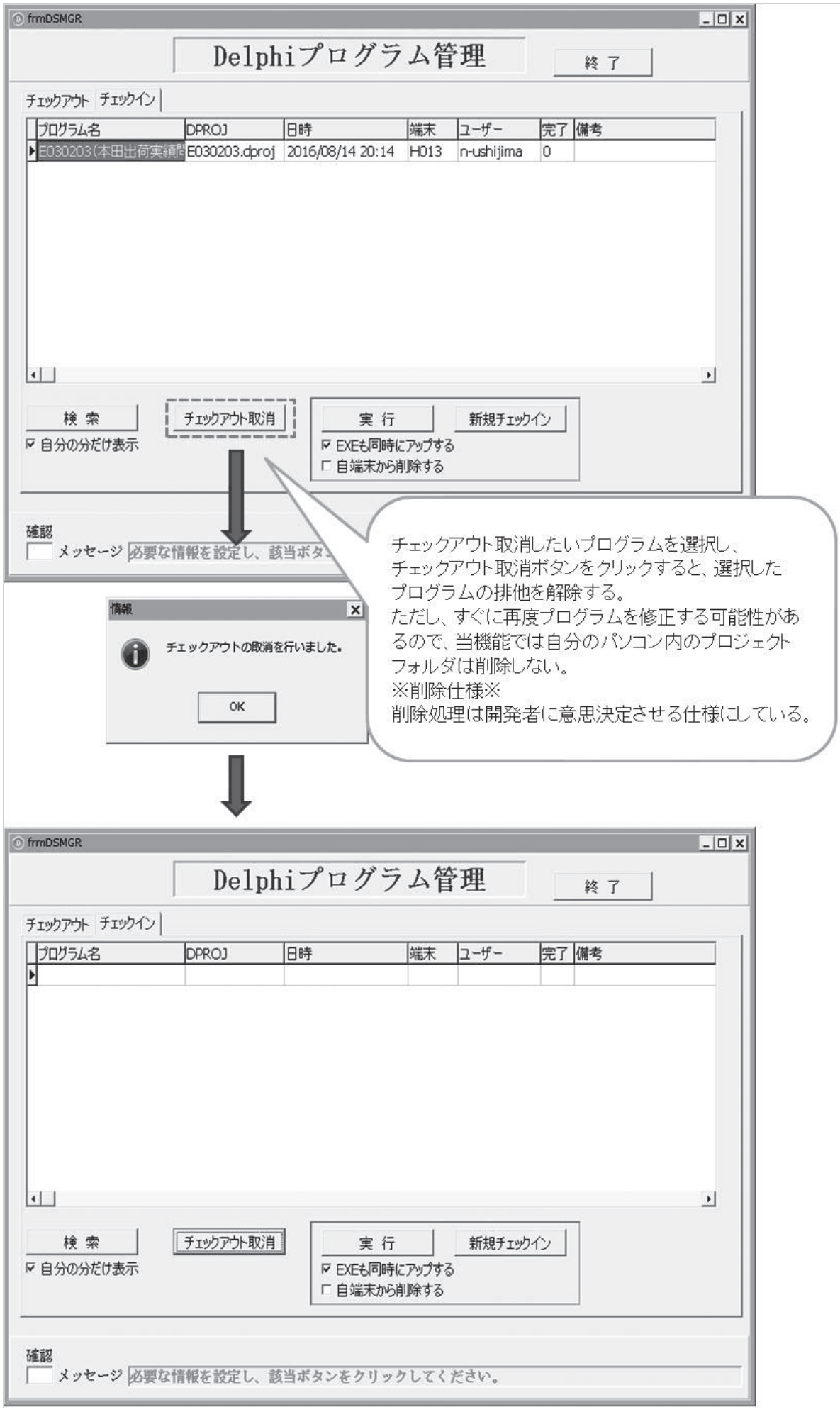
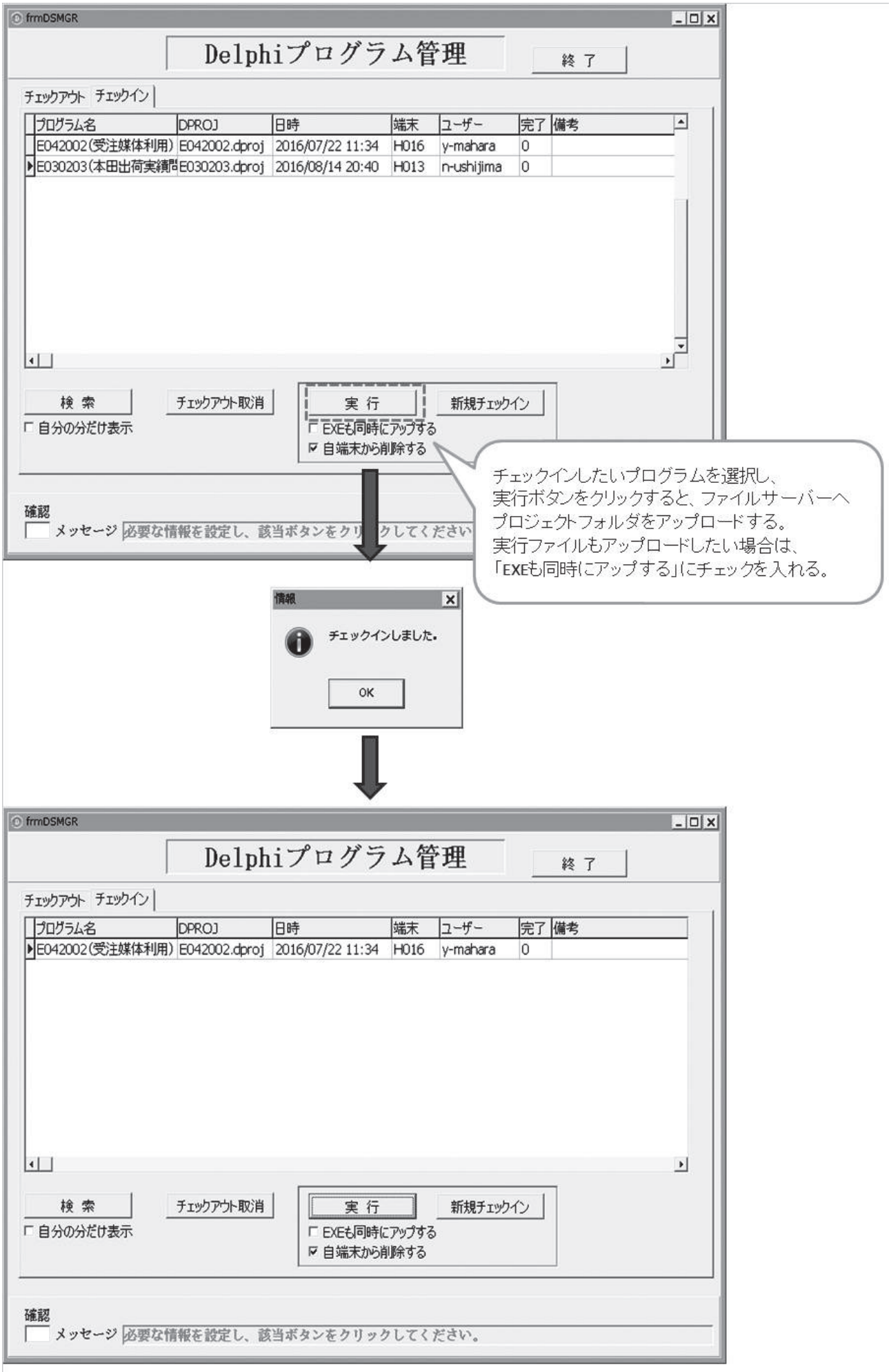


図5　チェックイン処理内容



ソース1　チェックアウト処理

```
{*************************************************************************
 目的: チェックアウト処理
 引数:
 戻値:
*************************************************************************}
procedure TfrmDSMGR.Check_Out;
var
  sInfolder,sInDir:string;
  CurDir : String;      //カレントディレクトリを格納する変数
  SelectFolder: String;  //フォルダのパスを格納する変数
  sREC:String;
  SHFILEOPSTRUCT : TSHFileOpStruct;
begin
  //プログラム分類名格納
  sInfolder :=
  Copy(Dir1.GetItemPath(Dir1.ItemIndex),Pos(Combo.Text,Dir1.GetItemPath(Dir1.ItemIndex) )
   + Length(Combo.Text) + 1);

  //チェックアウト先確認用パス設定(自端末)
  sIndir := CT_Path + combo.Text + '¥' + sInfolder + '¥';

  //自端末に同じフォルダがあるかチェックを行う。
  if DirectoryExists(sIndir) then
  begin
    MessageDlg('コピー先に同じフォルダがあります。確認してください。', mtInformation, [mbOK], 0);
    //procedure抜ける
    exit;
  end;

  //自端末コピー
  CurDir := Dir1.GetItemPath(Dir1.ItemIndex);        //from
  SelectFolder := CT_Path + combo.Text; //to
  //構造体の初期設定
  With SHFILEOPSTRUCT do
  begin
    wnd    := Handle;
    wFunc  := FO_COPY;
    pFrom  := PChar(CurDir + #0#0) ;
    pTo    := PChar(SelectFolder + #0#0);
    fFlags := FOF_ALLOWUNDO or FOF_FILESONLY;
    fAnyOperationsAborted := False;
    hNameMappings := nil;
    lpszProgressTitle := nil;
  end;
  //コピー実行
  SHFileOperation(SHFILEOPSTRUCT);

  with dmMain.cdsAccess do                                    排他フラグセット
  begin
    //ACCESSに排他フラグを立てる(1レコード追加)。
    Open;
    //追記モードオン
    Append;
    //追記レコードのパラメータ設定
    FieldByName('プログラム名').AsString   := sInfolder;
    FieldByName('DPROJ').AsString  := File1.Items[0];
    FieldByName('日時').AsString   := FormatDateTime('yyyy/mm/dd hh:mm', Now());
    FieldByName('端末').AsString   := GetComputerNameStr();
    FieldByName('ユーザー').AsString   := GetLoginNameStr();
    FieldByName('完了').AsString   := '0';
    FieldByName('PATH').AsString   := StringReplace(Dir1.GetItemPath(Dir1.ItemIndex),sInfolder,'', [rfReplaceAll, rfIgnoreCase]);
    //cds確定
    Post;
    //cdsとAccessの同期
    ApplyUpdates(-1);
    //レコードテキスト設定
    sREC := '"Check_Out","' + FieldByName('プログラム名').AsString + '","' +
          FieldByName('日時').AsString + '","' + FieldByName('端末').AsString + '","' +
          FieldByName('ユーザー').AsString + '"';
    //ログファイル追記
    WriteText(SV_Path + 'DSMGR.LOG',sREC);
    //cdsクローズ
    Close;
  end;

  Dialogs.MessageDlg('チェックアウトしました。', mtInformation, [mbOK], 0);

end;
```

ソース2　チェックイン処理

```
{**********************************************************************************
 目的: チェックイン処理
 引数:
 戻値:
**********************************************************************************}
procedure TfrmDSMGR.Check_In;
var
  sInfolder,sInDir,sPrgNm:string;
  sYYYYMMDD,sHHMMSS:string;
  CurDir : String;      //カレントディレクトリを格納する変数
  SelectFolder: String;  //フォルダのパスを格納する変数
  SHFILEOPSTRUCT : TSHFileOpStruct;
  sExeName : string;
  sREC:string;
begin
  with dmMain.cdsAccess do
  begin
    //カレントレコードのプログラム名の変数格納
    sPrgNm := FieldByName('プログラム名').AsString;
    //年月日の変数格納
    sYYYYMMDD := FormatDateTime('yyyymmdd', Now());
    //時分秒の変数格納
    sHHMMSS := FormatDateTime('hhmmss', Now());

    //EXE展開の場合は、EXEが圧縮してあるか応答画面を表示させる
    //「はい」と「いいえ」で選択させる形式とし、「いいえ」だとこれ以降の処理をさせない
    if EXEUP.Checked then
    begin
      if MessageDlg('EXEは圧縮してますか？', mtConfirmation, [mbYes, mbNo], 0) = mrNo then
      begin
        Dialogs.MessageDlg('圧縮後、再度チェックイン作業を行ってください。', mtInformation, [mbYes], 0);
        //procedure抜ける
        exit;
      end;
    end;

    //選択したプログラムが同端末かチェックする
    if FieldByName('端末').AsString <> GetComputerNameStr() then
    begin
      Dialogs.MessageDlg('チェックアウトした端末ではない為、チェックインできません。', mtWarning, [mbYes], 0);
      exit;
    end;
    //選択したプログラムが同ユーザーかチェックする
    if FieldByName('ユーザー').AsString <> GetLoginNameStr() then
    begin
      Dialogs.MessageDlg('チェックアウトしたユーザーではない為、チェックインできません。', mtWarning, [mbYes], 0);
      exit;
    end;

    //プログラム分類名格納
    sInfolder := StringReplace(FieldByName('PATH').AsString,SV_Path,'', [rfReplaceAll, rfIgnoreCase]);
    //プログラムパス格納
    sIndir := CT_Path + sInfolder + sPrgNm;

    //自端末に同じフォルダがあるかチェックを行う
    if DirectoryExists(sIndir) = False then
    begin
      MessageDlg('コピー元にチェックアウトしたフォルダがありません。', mtInformation, [mbOK], 0);
      exit;
    end;

    //バックアップフォルダに変更前のソースをコピーする
    //バックアップ用フォルダがない場合は作成する
    if DirectoryExists(SVBK_Path + sYYYYMMDD + '¥' + sInfolder) = False then
    begin
      ForceDirectories(SVBK_Path + sYYYYMMDD + '¥' + sInfolder);
    end;
```

ソース2　チェックイン処理-2

```
//変更前のソース格納フォルダをコピーする。
CurDir := SV_Path + sInfolder + sPrgNm + '¥';   //from

//バックアップ先に同プログラム格納フォルダがあった場合は、時分秒を付けたフォルダを別途作成し、そこに格納する。
//【理由】同日中にチェックアウト＆チェックインした場合に、バックアップフォルダ自体を上書きされるのを防ぐため。
if DirectoryExists(SVBK_Path + sYYYYMMDD + '¥' + sInfolder + sPrgNm + '¥') = False then
begin
  //同日バックアップ先ダブりなしの場合
  SelectFolder := SVBK_Path + sYYYYMMDD + '¥' + sInfolder + sPrgNm + '¥'; //to
end
else
begin
  //プログラム格納フォルダの後ろに時分秒を付加したフォルダにコピーする
  SelectFolder := SVBK_Path + sYYYYMMDD + '¥' + sInfolder + sPrgNm + '_' + sHHMMSS + '¥'; //to
end;

//構造体の初期設定
With SHFILEOPSTRUCT do
begin
  wnd    := Handle;
  wFunc  := FO_COPY;
  pFrom  := PChar(CurDir + #0#0) ;
  pTo    := PChar(SelectFolder + #0#0);
  fFlags := FOF_ALLOWUNDO or FOF_FILESONLY;
  fAnyOperationsAborted := False;
  hNameMappings := nil;
  lpszProgressTitle := nil;
end;
//コピー実行
SHFileOperation(SHFILEOPSTRUCT);

//h222の変更前ソースを削除する
//誤ってscreenrealフォルダを削除してしまった。修正すること。
DeleteDirectory(SV_Path + sInfolder + sPrgNm);

//端末の変更後ソースをh222にコピーする
CurDir := sIndir + '¥';         //from
SelectFolder := SV_Path + sInfolder + sPrgNm + '¥'; //to
//構造体の初期設定
With SHFILEOPSTRUCT do
begin
  wnd    := Handle;
  wFunc  := FO_COPY;
  pFrom  := PChar(CurDir + #0#0) ;
  pTo    := PChar(SelectFolder + #0#0);
  fFlags := FOF_ALLOWUNDO or FOF_FILESONLY;
  fAnyOperationsAborted := False;
  hNameMappings := nil;
  lpszProgressTitle := nil;
end;
//コピー実行
SHFileOperation(SHFILEOPSTRUCT);

//EXE展開の場合⇒h222へコピー
if EXEUP.Checked then
begin
  //exe名称作成
  sExeName := StringReplace(FieldByName('DPROJ').AsString,'.dproj','', [rfReplaceAll, rfIgnoreCase]);

  if FileExists(CT_Path + 'Exe¥' + sExeName + '.exe') then
  //コピーするexeがある場合⇒コピー実行
  begin
    CopyFile(PChar(CT_Path + 'Exe¥' + sExeName + '.exe'),
             PChar(SVEXE_Path + sExeName + '.exe'),false);
  end
  else
  //コピーするEXEがない場合⇒警告メッセージ表示
```

ソース2　チェックイン処理-3

```
      begin
        Dialogs.MessageDlg('コピーするEXEがありませんでした。', mtWarning, [mbYes], 0);
      end;
    end;

    //管理用ACCESSの完了マークを'1'にする(カレントレコード)
    //編集モードオン
    Edit;
    //編集レコードのパラメータ設定
    FieldByName('完了').AsString := '1';
    //レコードテキスト設定
    sREC := '"Check_In","' + FieldByName('プログラム名').AsString + '","' +
            FieldByName('日時').AsString + '","' + FieldByName('端末').AsString + '","' +
            FieldByName('ユーザー').AsString + '"';
    //運用管理登録用ファイル追記
    WriteText(SV_Path + 'DSMGR.TXT',sREC);
    //ログファイル追記
    WriteText(SV_Path + 'DSMGR.LOG',sREC);
    //cds確定
    Post;
    //cdsとAccessの同期
    ApplyUpdates(-1);

    //端末側のソースを削除する_しないの選択肢から判断し、ソース削除処理を実行する
    if DEVELOP.Checked then
    begin
      DeleteDirectory(sIndir);
    end;

    Dialogs.MessageDlg('チェックインしました。', mtInformation, [mbOK], 0);
  end;

end;
```

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
管理部
主務

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

IBM i の物流システム「SPURT（スパート）」による日々の業務の中で、①定型のエクセルを開きデータ打ち込み、②印刷、③捺印、④ FAX 機のスキャナ機能で読み取り、PDF に保存、という一連の作業が存在する。

これらは担当者の作業効率が悪く、作業改善要望が挙げられていた。

技術課題

(1) 定型の Excel 上に以下の条件を満たして、日付印と担当者印を作成することは可能か。

①日付印の日付欄に、発行日当日を設定する。
②担当者印に、入力担当者名を設定する。
③権限者のみが実行できるようにする。

(2) Excel 上にデータを埋め込み、印刷データを作成した後、PDF 変換はどう行うのか。

技術課題の解決策

IBM i 上の発注・見積等のデータを参照し、Delphi/400 により以下のプログラム対応を行った。

(1) 定型の Excel 上に、図形を用いて印鑑を作成する。【図 1】

①日付の部分は印刷外の固定セルに“=TODAY ()”を埋め込み、印鑑欄からそのセル値を参照することで日付印が作成できた。
②担当者印に設定する担当者名は、DB 上の担当者フィールドから文字列を参照した。
③権限マスターの設定により、本オペレーションが可能な担当者を登録可能とした。また事業部は大きく東・西に分かれているため、東・西の範囲を超えた登録はできない仕組みとした。

(2) PDF 保存は、Excel のエクスポート機能を利用して実現した。【図 2】【図 3】【図 4】

業務課題解決と効果

作業担当者は、座席から離れることなく一連の作業を行えるようになり、作業工数が大幅に削減された。今後は他業務での同様の作業を洗い出し、この仕組みに置き換えていく予定である。

M

図1　定型Excel

捺　印　請　求　書

平成　　年　　月　　日

印章名			
捺印書類名	1	加工作業発注書	通
	2	加工資材発注書	通
	3	加工作業・資材見積書	通
捺印書類の簡単な説明			
受注No.			
提出先			

部長印	請求者
ラ流通 16.6.3 事業部長	

図2　Delphi/400実行

図3　Delphi/400ソース（抜粋）

```
//エクセル起動
ExcelApplication1.Connect;
//エクセル非表示
ExcelApplication1.Visible[0] := false;
//ブック追加
ExcelWorkbook1.ConnectTo(ExcelApplication1.Workbooks.Add(gFileName, 0));
//シートを ExcelWorksheetに接続
ExcelWorksheet1.ConnectTo(ExcelWorkbook1.Worksheets['捺印請求書'] as _Worksheet);

//データセット
str_yymd := '平成' + IntToStr(StrToIntDef(Copy(wrk_date,1,4),0)-1988) + '年' + Copy(wrk_date,6,2) + '月' +
Copy(wrk_date,9,2) + '日';
str_cell := 'E4';
ExcelWorksheet1.Range[str_cell,str_cell].value2 := str_yymd;

str_cell := 'B11';
ExcelWorksheet1.Range[str_cell,str_cell].value2 :='  ' + #13#10  + Edt_JNO.Text;

if (gSU1 <> 0) then
begin
 str_cell := 'H6';
 ExcelWorksheet1.Range[str_cell,str_cell].value2 := ConvJ(IntToStr(gSU1)) + '通';
end;

  ・・・・・・・・（省略）

if (Trim(Edt_IKN.Text) <> '') then
begin
 str_cell := 'J14';
 ExcelWorksheet1.Range[str_cell,str_cell].value2 := Edt_IKN.Text;
end;

//エクセル終了
gFileNamep := 'C:¥WORK¥TEST¥捺印請求書イメージ'+ str_date + '_' + str_time + '.PDF';

objExcelBook := ExcelApplication1.ActiveWorkbook;
objExcelBook.ExportAsFixedFormat(Type    := xlTypePDF,
                  Filename := gFileNamep, IncludeDocProperties := False);
ExcelWorkbook1.Saved[0] := True;  //PDF

try
 ExcelWorksheet1.Disconnect;
 ExcelWorkbook1.Disconnect;
 ExcelApplication1.DisplayAlerts[GetUserDefaultLCID] := False;
 ExcelApplication1.Quit;
 ExcelApplication1.Disconnect;
except
 ExcelApplication1.Quit;
 ExcelApplication1.Disconnect;
end;
```

図4　実行結果PDF

捺　印　請　求　書

平成28年06月03日

印章名		
捺印書類名	1　加工作業発注書	通
	2　加工資材発注書	3通
	3　加工作業・資材見積書	通
捺印書類の簡単な説明 受注No.　JU12345		
提出先		

部長印	請求者
ラ流通 16.6.3 事業部長	テスト

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
システム部
係長

株式会社スマイル・ジャパン
http://www.sukoyaka-egao.jp/

健康食品の通信販売事業を行う。健康食品の研究・開発を通して、お客様の健康上の悩みを少しでも解消し、健やかにいきいきと毎日を送っていただくことを目的に活動している。

業務課題

通販業務での返品作業の現場から、以下の業務課題が挙げられた。

1　処理漏れが起こる（開封場所と事務処理（PC）場所が違うため）
2　顧客を限定させるまでに、時間がかかる場合がある

現状返品が発生した場合、返品の送り状か、同封されている郵振用紙または納品書で顧客を調べてから返品処理を行っている。【図 1】

全体の 8 ～ 9 割は郵振用紙を使用していることから、郵振用紙に印字されているバーコードを読み取り、そのまま返品処理を行うことで業務を効率化したい。

返品作業には箱を開けるスペースが必要なので持ち運びが楽なこと、またカメラを利用したいことから、モバイル端末が処理に最適と考え、IBM i の基幹システムにモバイルアプリを「ちょい足し」することを検討した。

技術課題

各プログラムの役割を、以下とした。

・モバイルアプリケーション：バーコード読み取り＋入力
・データスナップ：DB 中継
・RPG：IBM i データ更新

ミガロ .主催のモバイルファースト体験セミナーに参加し、モバイル開発の概要を習得。実際の開発作業では、以下の不明点が残った。

(a)「AS400」「Call400」のコンポーネントはどこで動作するか。
(b) モバイル端末で入力された情報を RPG までどのように受け渡すか。
(c) モバイル端末として iPhone を選定したが、iOS 開発環境をどのように構築するか。
(d) IIS での iOS アプリの配信方法はどのようなものか。
(e) モバイルにてリモートエラーの発生する原因は何か。

技術課題の解決策

ミガロ . のテクニカルサポートの協力を得て、すべての不明点を解決した。

以下の（a）～（e）は、技術課題の（a）～（e）に対応する。

(a) データスナップで IBM i、RPG との連携を行う。【図 2】

(b) 以下の手順でモバイル端末入力情報と RPG を連携する。

①既存 RPG 画面プログラムを元に、IBM i データ更新用バッチプログラムを開発【図 3】
②データスナップにて、function を作成【図 4】
③モバイルにて、データスナップ取り込み（1）【図 5】

図1　従来の返品処理画面

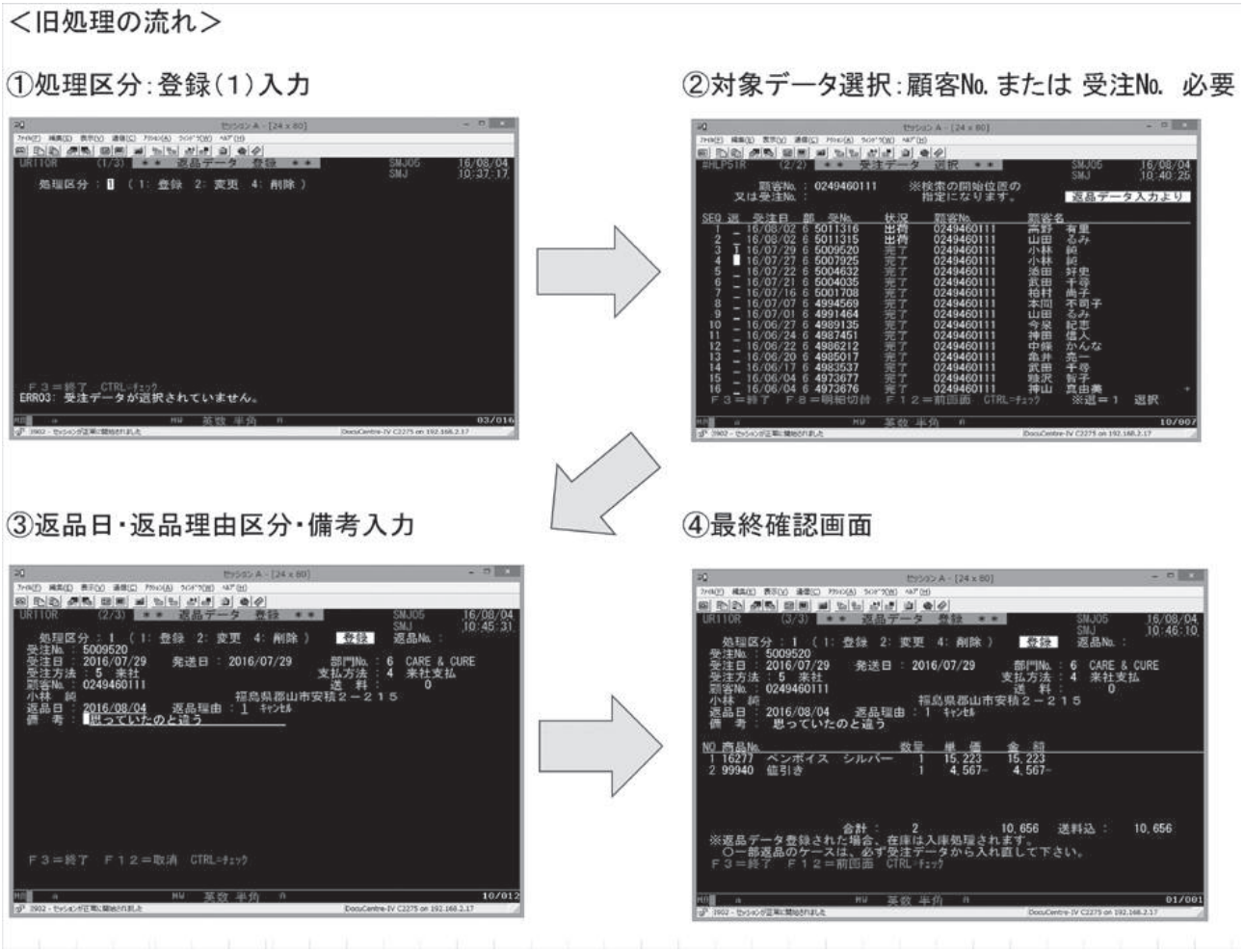
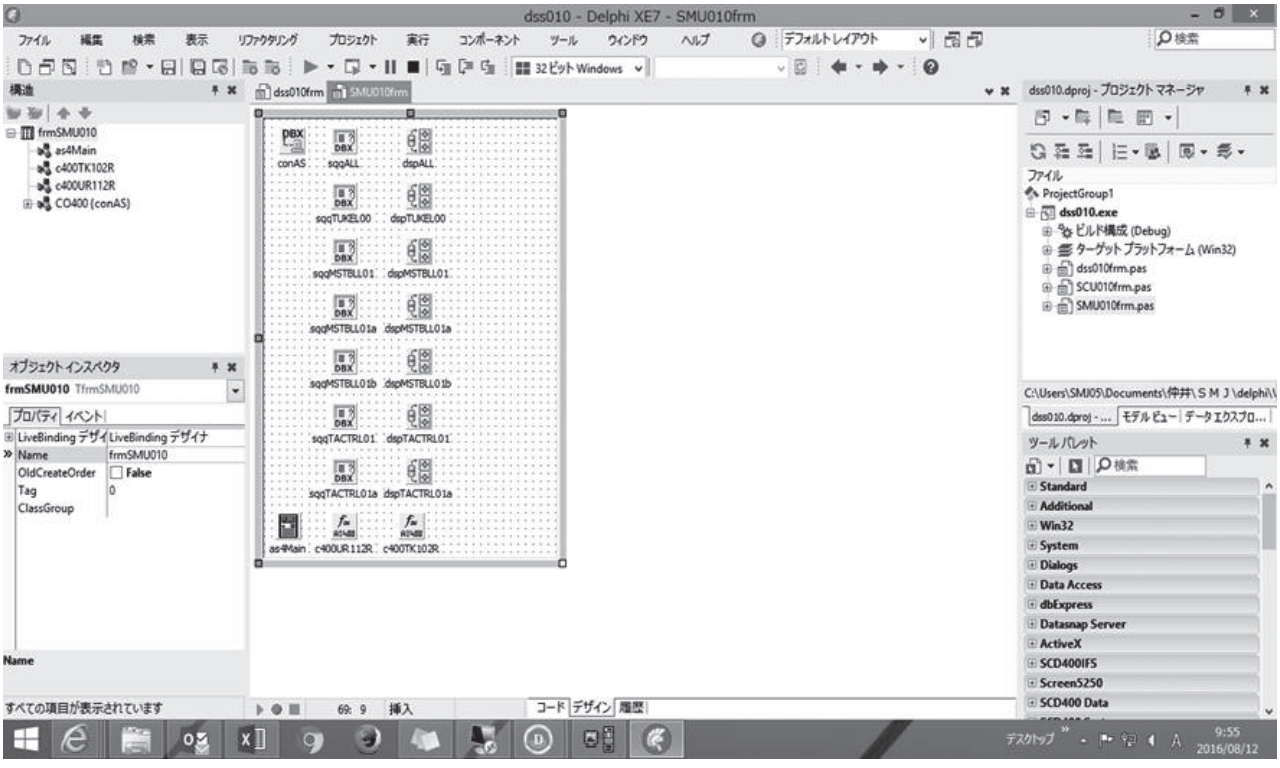


図2　AS/400、RPGとの連携手法（データスナップにて動作）



④モバイルにて、データスナップ取り込み（2）【図6】
⑤モバイルにて、データスナップのファンクション（パラメータ渡し）実行【図7】

（c）Appleの関連性（プロビジョニングとiOS、OSX、XcodeVer）について、ミガロ.のテクニカルサポートへ問い合わせて解決【図8】（①の手順の中で、利用モバイル端末のデバイス登録がないとiOSアプリはインストールできないので注意が必要）。

（d）IISからのアプリ配信について、豊鋼材工業様のテクニカルレポート（2015年）およびミガロ.のテクニカルサポートへの問い合わせで解決。【図9】

上記（c）と（d）を完了すると、モバイルからHTML経由で、SSLのインストールが可能となり、iOSアプリもインストール可能となる。

（e）データスナップにて「AS400」「Call400」を使用後、FalseはOKだが、freeも記述したためリモートエラーになっていた。そこで、freeのコメント化により解決した。【図10】

業務課題解決と効果

業務課題は、以下のとおり解決できた。【図11】

1　返品作業の処理漏れは、作業場所に持ち運び可能なモバイル導入により、発生しなくなった。

2　新しい業務手順ではバーコードを読むだけとなり、顧客No.などをまったく意識せずに処理可能となった。また作業時間は1/10となり、時間短縮にもつながった。

技術課題（c）iOS開発環境構築以外の開発工数は約2週間程であったが、iOS開発はハードルが高い印象をもった。

今回はiOSを採用したが、環境構築に手間がかかるので、表題の「ちょい足しモバイル」にはAndroidが適切と思われる。

IBM i 既存業務にモバイルを「ちょい足し」する今回の試みは、Delphi/400の新しいモバイル機能の充実とミガロ.テクニカルサポートにより実現できた。

M

図3　IBM i データ更新用バッチプログラムの開発

①IBM i データ更新用バッチプログラムの開発

モバイル入力処理の最後にバッチプログラムを呼び出し、IBMiデータベースを更新する。

このバッチプログラムは、既存のRPGの画面プログラムをベースとし、モバイル画面の入力データをパラメータで受け取れるようにしたもの。

また、IBMiデータベース更新部分は、既存RPGプログラムも新規バッチプログラムも処理が同様のため、既存プログラムのロジックを流用して開発した。

図4　function作成

②データスナップにて、functionを作成

```
  private
    { private 宣言 }
  public
    { public 宣言 }
    function EchoString(Value: string): string;
    function ReverseString(Value: string): string;
    function CALL400UR112R(P1: string; P2: string; P3: string; P4: string; P5: string; P6: string): string;
    function CALL400TK102R(P1: string; P2: string; P3: string; P4: string; P5: string): string;
  end;

function TfrmSMU010.CALL400UR112R(P1: string; P2: string; P3: string; P4: string; P5: string; P6: string): string;
begin
  as4Main.Active   := False;
  as4Main.PLUAlias := 'AS400';
  as4Main.Userid   := 'SMJ';  // サインオンユーザ
  as4Main.PWD      := 'SMJ';  // サインオンパスワード

  try

    as4Main.Active    := True;
    as4Main.RemoteCmd('CALL SLBPGM/INZBCH');

    c400UR112R.Value[0] := P1;
    c400UR112R.Value[1] := P2;
    c400UR112R.Value[2] := P3;
    c400UR112R.Value[3] := P4;
    c400UR112R.Value[4] := P5;
    c400UR112R.Value[5] := P6;
    c400UR112R.Value[6] := '';

    c400UR112R.Execute;

    Result := c400UR112R.Value[6];

  finally
    as4Main.Active   := False;
//    as4Main.free;
//    c400UR112R.free;
  end;

end;
```

図5　データスナップ取り込み（1）

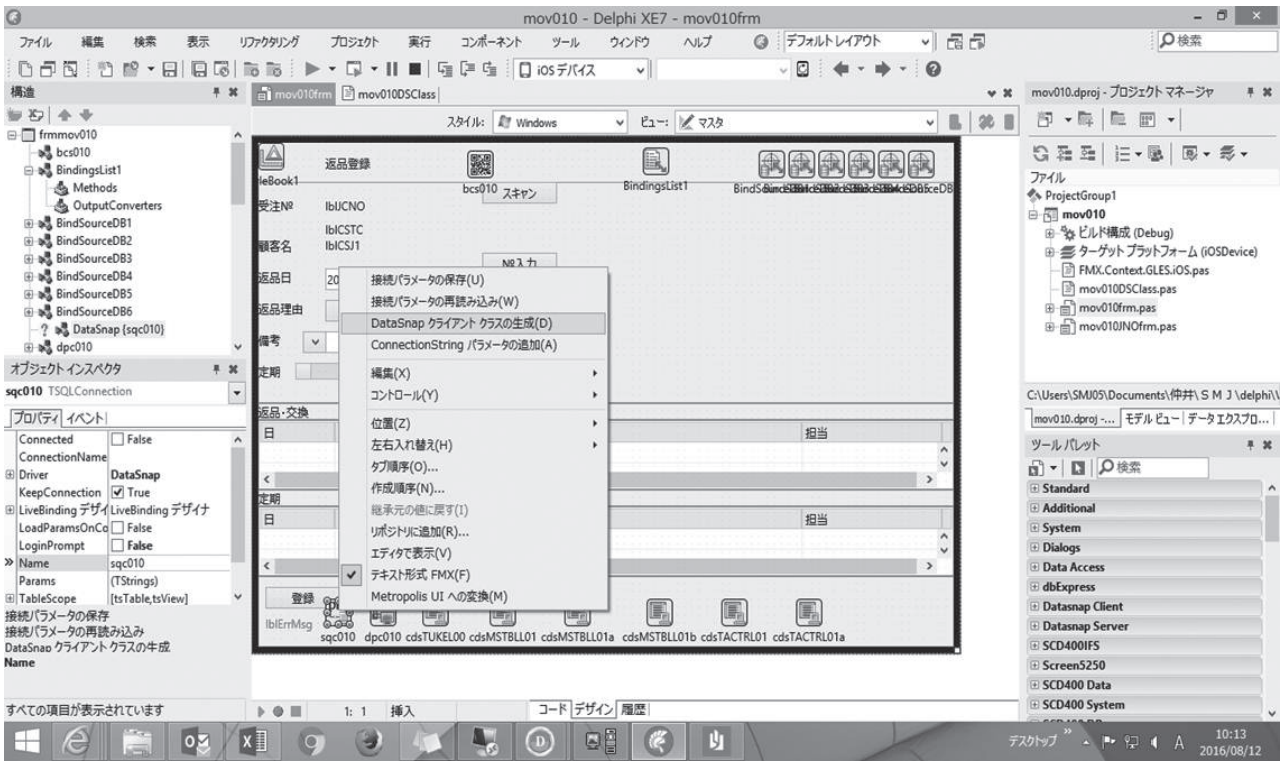


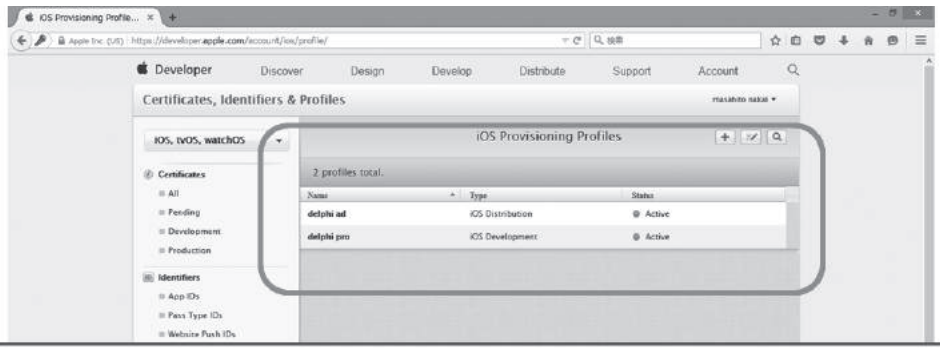
図6　モバイルにて、データスナップ取り込み

```
//
// DataSnap プロキシ ジェネレータにより作成。
// 2016/03/30 18:11:49
//

unit mov010DSClass;

interface

uses System.JSON, Data.DBXCommon, Data.DBXClient, Data.DBXDataSnap, Data.DBXJSON, Datasnap.DS

type
  TfrmSMU010Client = class(TDSAdminClient)
  private
    FEchoStringCommand: TDBXCommand;
    FReverseStringCommand: TDBXCommand;
    FCALL400UR112RCommand: TDBXCommand;
    FCALL400TK102RCommand: TDBXCommand;
  public
    constructor Create(ADBXConnection: TDBXConnection); overload;
    constructor Create(ADBXConnection: TDBXConnection; AInstanceOwner: Boolean); overload;
    destructor Destroy; override;
    function EchoString(Value: string): string;
    function ReverseString(Value: string): string;
    function CALL400UR112R(P1: string; P2: string; P3: string; P4: string; P5: string; P6: st
    function CALL400TK102R(P1: string; P2: string; P3: string; P4: string; P5: string): strin
  end;

implementation

function TfrmSMU010Client.EchoString(Value: string): string;
begin
  if FEchoStringCommand = nil then
  begin
    FEchoStringCommand := FDBXConnection.CreateCommand;
    FEchoStringCommand.CommandType := TDBXCommandTypes.DSServerMethod;
    FEchoStringCommand.Text := 'TfrmSMU010.EchoString';
    FEchoStringCommand.Prepare;
  end;
  FEchoStringCommand.Parameters[0].Value.SetWideString(Value);
  FEchoStringCommand.ExecuteUpdate;
  Result := FEchoStringCommand.Parameters[1].Value.GetWideString;
end;

function TfrmSMU010Client.ReverseString(Value: string): string;
```

図7　データスナップファンクション実行

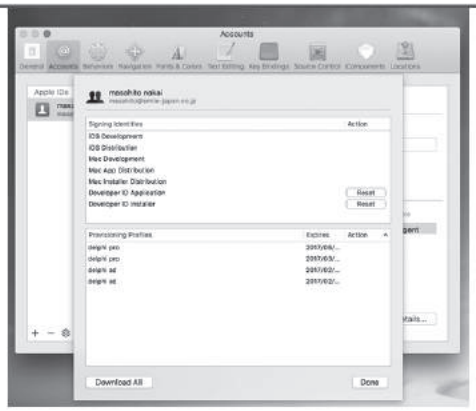
```
    Abort;
  end;
  if lblHRSN.Text = '' then
  begin
    ShowMessage('返品理由が未選択です');
    lblErrMsg.Text := '返品理由が未選択です';
    lblErrMsg.Visible := True;
    Abort;
  end;

  if  ( swhTK.IsChecked  )
  and ( lblBRSN.Text = '' ) then
  begin
    ShowMessage('定期休止理由が未選択です');
    lblErrMsg.Text := '定期休止理由が未選択です';
    lblErrMsg.Visible := True;
    Abort;
  end;

  sHPNB := FormatDateTime('YYYYMMDD',dteHPNB.Date);

  Temp := TfrmSMU010Client.Create(sqc010.DBXConnection);

  try

    lblHPNO.Text := Temp.CALL400UR112R('mov','SMJ',lblJCNO.Text,sHPNB,lblHRSN.Text,edtCMNT.Text);

    if swhTK.IsChecked then
    begin
      sRTN := Temp.CALL400TK102R('mov','SMJ',lblCSTC.Text,lblBRSN.Text,sHPNB);
    end;

  finally
    Temp.Free;

    cdsTUKEL00.Active   := False;
    cdsMSTBLL01.Active  := False;
    cdsMSTBLL01a.Active := False;
    cdsMSTBLL01b.Active := False;
    cdsTACTRL01.Active  := False;
    cdsTACTRL01a.Active := False;
    lblBRSN_T.Visible   := False;
    lblTK.Visible       := False;
    swhTK.Visible       := False;
    cmbBRSJ.Visible     := False;
```

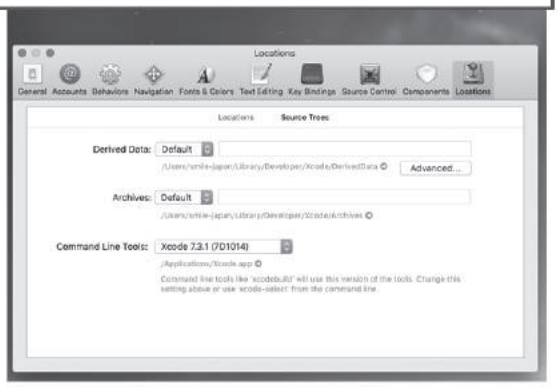
図8　iOS開発環境構築

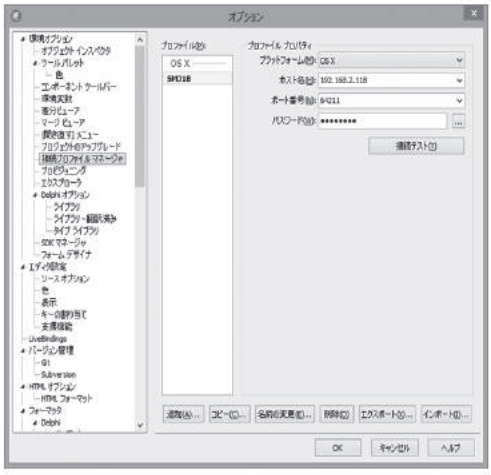


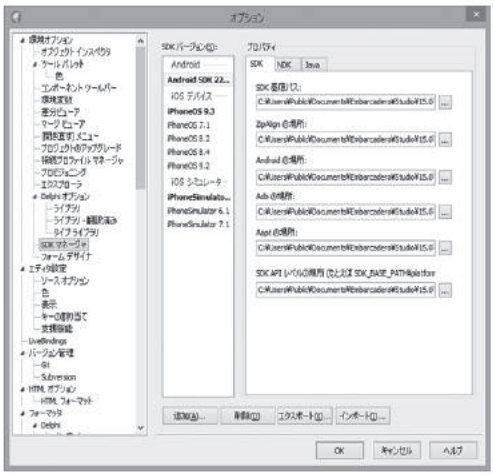
①Appleメンバーセンター（有料）にて、iOS Development(iOSの開発)と
iOS Distribution（iOSの配布）を登録する。
この一連の過程で、利用モバイル端末の登録も実施済みとなっている。

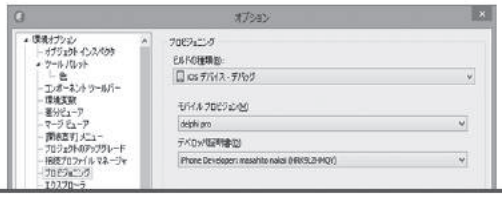
②Xcodeで上記ライセンスが利用できるように設定し、該当macにPA Server インストール







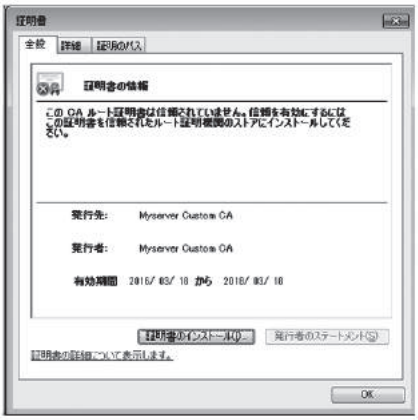




③Delphi にて、接続プロファイル・SDKの設定をすれば、デバッグ時は開発ライセンス、
配布作成時は配布ライセンス　を利用可能

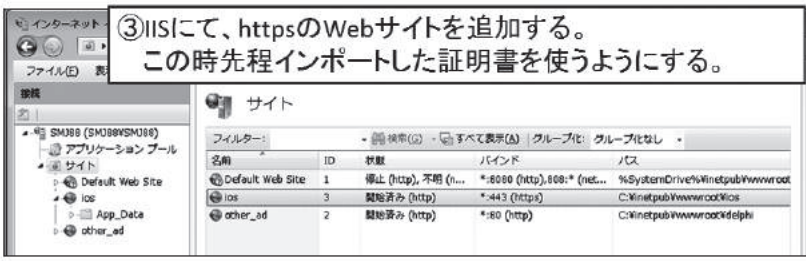
図9　IISにてiOSアプリの配信

①Mac にて作成した自己証明書を
IISのマシンにインストール



②Mac にて作成した自己証明書を
IISのサーバー証明書にインポートする

③IISにて、httpsのWebサイトを追加する。
この時先程インポートした証明書を使うようにする。



④httpsのフォルダにplistとipaを入れ、MIMEタイプに拡張子plistとipaを追加

⑤apple configurator2にて
ssl証明書を作成する。
この時①で作成した証明書を
使うようにする。

⑥httpのフォルダにssl証明書（mobileconfig）とhtmlを入れ
MIMEタイプに拡張子mobileconfigを追加

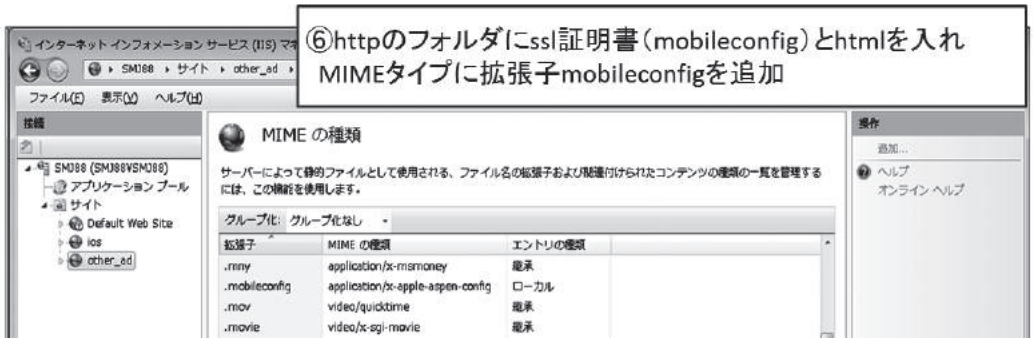
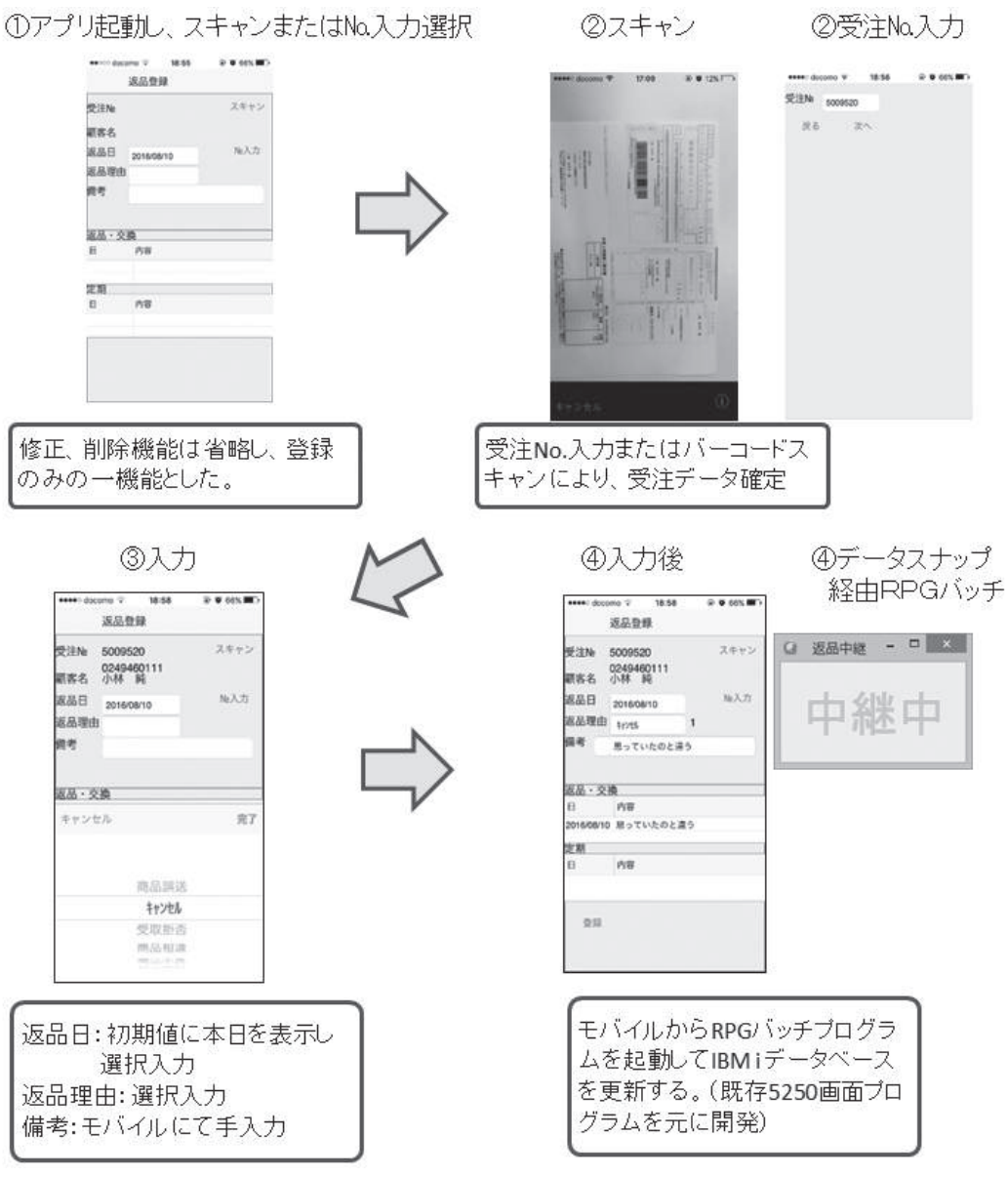


図10　リモートエラーの解決

```
begin
  Result := Value;
end;

function TfrmSMU010.ReverseString(Value: string): string;
begin
  Result := System.StrUtils.ReverseString(Value);
end;

function TfrmSMU010.CALL400UR112R(P1: string; P2: string; P3: string; P4: string; P5: string; P6: string): string;
begin

  as4Main.Active   := False;
  as4Main.PLUAlias := 'AS400';
  as4Main.Userid   := 'SMJ';  // サインオンユーザ
  as4Main.PWD      := 'SMJ';  // サインオンパスワード

  try

    as4Main.Active     := True;
    as4Main.RemoteCmd('CALL SLBPGM/INZBCH');

    c400UR112R.Value[0] := P1;
    c400UR112R.Value[1] := P2;
    c400UR112R.Value[2] := P3;
    c400UR112R.Value[3] := P4;
    c400UR112R.Value[4] := P5;
    c400UR112R.Value[5] := P6;
    c400UR112R.Value[6] := '';

    c400UR112R.Execute;

    Result := c400UR112R.Value[6];

  finally
    as4Main.Active   := False;
//    as4Main.free;
//    c400UR112R.free;
  end;

end;

function TfrmSMU010.CALL400TK102R(P1: string; P2: string; P3: string; P4: string; P5: string): string;
begin
```

図11　モバイルでの返品作業手順



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
代表取締役

株式会社ランドコンピュータ
http://www.rand.co.jp/

高校・大学等のコンピュータ教室で使われる「授業支援システム」の設計、開発、販売を行っており、学校等の教育関連施設に対して 5000 件以上の納入実績がある。業務ソリューションに必要なソフトウェア開発と、画像・音声処理などの機器製造をトータルで実施している。

業務課題

会社の受注状況は週初めの朝礼で報告しているが、欠席している人やパート社員には共有できていない。

技術課題

受注データは IBM i のデータベースから取得し、社内公開は Web 形式で行う。【図 1】

あまり工数をかけずに完成させたい。

技術課題の解決策

Word で HTML のテンプレートを作成し、数字が入る部分を［%○○%］というようにネーミング。【図 2】

HTML テンプレートへの受注残高の設定は、Delphi/400 の StringReplace 関数を使って、簡単にプログラミングできた。

Delphi/400 のプログラム概要は以下のとおりである。【図 3】

(1) Delphi/400 + dbExpress で、IBM i の受注データを抽出・集計
(2) TStringList で、テンプレート HTML を読み込む
(3) ［%○○%］の部分を検索して、受注データに置き換え
(4) Web 公開先に保存

また受注データの更新タイミングは、Windows サーバーがもつタスクスケジューラを使って簡単に実装した。

業務課題解決と効果

9 時と 17 時に、受注データを HTML に反映させることができ、当初の目的どおり、社内で受注状況の共有が実現できた。【図 4】

M

図1　システム構成

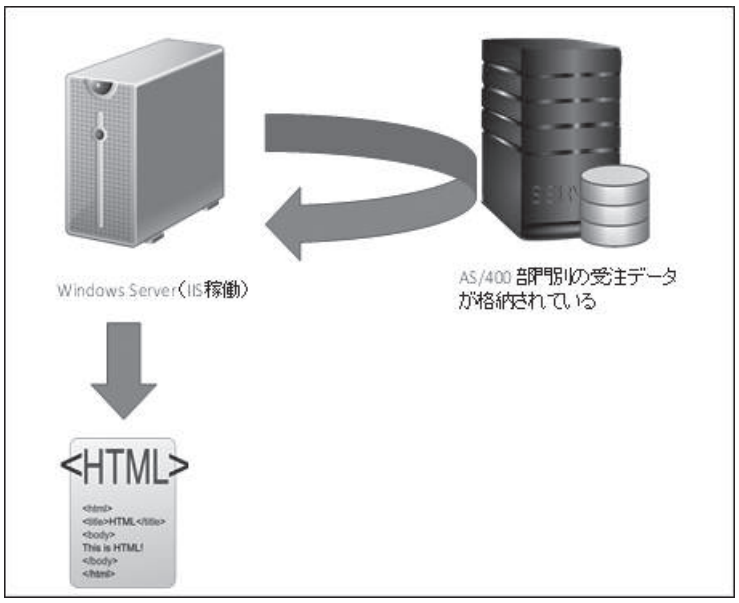
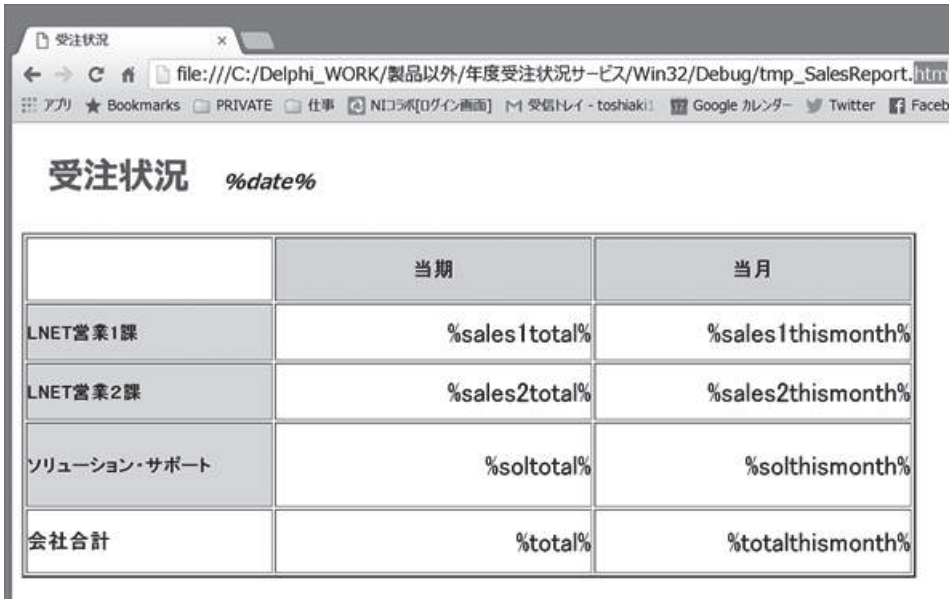


図2　テンプレート



	当期	当月
LNET営業1課	%sales1total%	%sales1thismonth%
LNET営業2課	%sales2total%	%sales2thismonth%
ソリューション・サポート	%soltotal%	%solthismonth%
会社合計	%total%	%totalthismonth%

図3　プログラム（抜粋）

1. Delphi/400＋dbExpressでAS/400の受注データを抽出、集計

```
sqlqryMain.SQL.Add('Select SHLRPN, SUM(SHTAMT) as SALES from SBKLGH' +
  ' Where SHORD >= ' + STStr +
  ' And SHORD <= ' + EnStr +
  //営業1課
  ' And (SHLRPN = 00453' +
  //営業2課
  ' Or SHLRPN = 00465)' +
  ' Group By SHLRPN');
sqlqryMain.Open;
```

※STStr、EnStrはstring型の変数で160401、170301といった日付が入っている

2. TStringListでテンプレートHTMLを読み込む

```
SaleRep := TStringList.Create;
SaleRep.LoadFromFile(m_WorkPath + 'tmp_SalesReport.htm', TEncoding.UTF8);
```

3. [%○○%]の部分を検索して、受注データに置き換え

```
// 営業1課
Idx := SaleRep.IndexOf('%sales1total%');
if Idx >= 0 then
begin
  SaleRep.Strings[Idx] := FormatFloat('#,##0', m_SalesTotal[0].ToSingle);
end;

// 営業2課
Idx := SaleRep.IndexOf('%sales2total%');
if Idx >= 0 then
begin
  SaleRep.Strings[Idx] := FormatFloat('#,##0', m_SalesTotal[1].ToSingle);
end;
```

4. Web公開先に保存

```
SaleRep.SaveToFile('C:\inetpub\wwwroot\SalesReport.htm');
```

図4　実行結果

受注状況　*2016/08/17 09:00*

	当期	当月
LNET営業1課	60,115,992	19,173,000
LNET営業2課	124,017,504	18,952,850
ソリューション・サポート	3,399,771	301,500
会社合計	187,533,264	38,427,352

Migaro.
Technical Report
2016

ミガロ.SE 論文／ミガロ. テクニカルレポート

前坂 誠二 ／ 大橋 拓也

株式会社ミガロ.

システム事業部 システム2課

[Delphi/400] iOSモバイルアプリ開発のデザイニングテクニック

略歴　前坂 誠二
1989 年 3 月 21 日生まれ
2011 年 関西大学文学部卒業
2011 年 4 月 株式会社ミガロ. 入社
2011 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、Delphi/400 の開発経験を積みながら、日々スキルを磨いている。

略歴　大橋 拓也
1992 年 7 月 4 日生まれ
2015 年 龍谷大学経営学部卒業
2015 年 4 月 株式会社ミガロ. 入社
2015 年 4 月 システム事業部配属

現在の仕事内容
Delphi、Delphi/400 を利用したシステム開発および保守作業を担当。開発スキルの向上を目指し、日々精進している。

1.はじめに

ここ数年で、企業でのスマートデバイス普及率は大きく跳ね上がり、それに伴ってモバイルアプリケーション開発の需要も高まっている。

モバイルアプリケーションの開発にあたり、多くの開発者が向き合うことになるのが画面デザインの部分ではないだろうか。なぜなら、PC アプリケーションとモバイルアプリケーションでは使用できる画面領域や操作方法が違うため、モバイルアプリケーション用に再デザインする必要があるからだ。

そこで本稿では、モバイルアプリケーション開発で課題となる画面デザインについて考察し、それらの課題を解消する開発テクニックを解説する。

スマートデバイスにはさまざまな種類があるが、企業で業務用に導入されているデバイスで圧倒的に多いのが iOS の iPad である。そのため、本稿ではデバイスとして iPad（mini）を題材に、Delphi/400 XE7 を使用する。もちろん画面デザインの考え方は、他のデバイスや OS であっても同じである。

2.iOSアプリケーションで使用できる画面領域

まず、PC（Windows）と iOS デバイスで作成されるアプリケーションの画面サイズについて整理する。

PC アプリケーションでは、端末のサイズにもよるが、1280 × 1024 や 1366 × 768 を最大サイズとして作成されることが多い。

それに対して、iOS デバイスのアプリケーションでは iPad で 768 × 1024、iPhone で 320 × 568 や 414 × 736 が最大サイズである。こうして見ると、意外にも iPad についてはサイズに大きな違いはないことがわかる。【図 1】

しかし PC と iOS デバイスのアプリケーションでは、操作方法が大きく異なる。PC では主にマウスやキーボードでの操作が多いので、画面項目のサイズや項目間の余白についてはあまり気にする必要はない。

それに対して iOS デバイスでは、主に指によるタッチ操作が主流となる。項目間の余白が少なかったり、項目自体のサイズが小さいと、ボタンの押し間違いや項目の選択ミスにつながる。

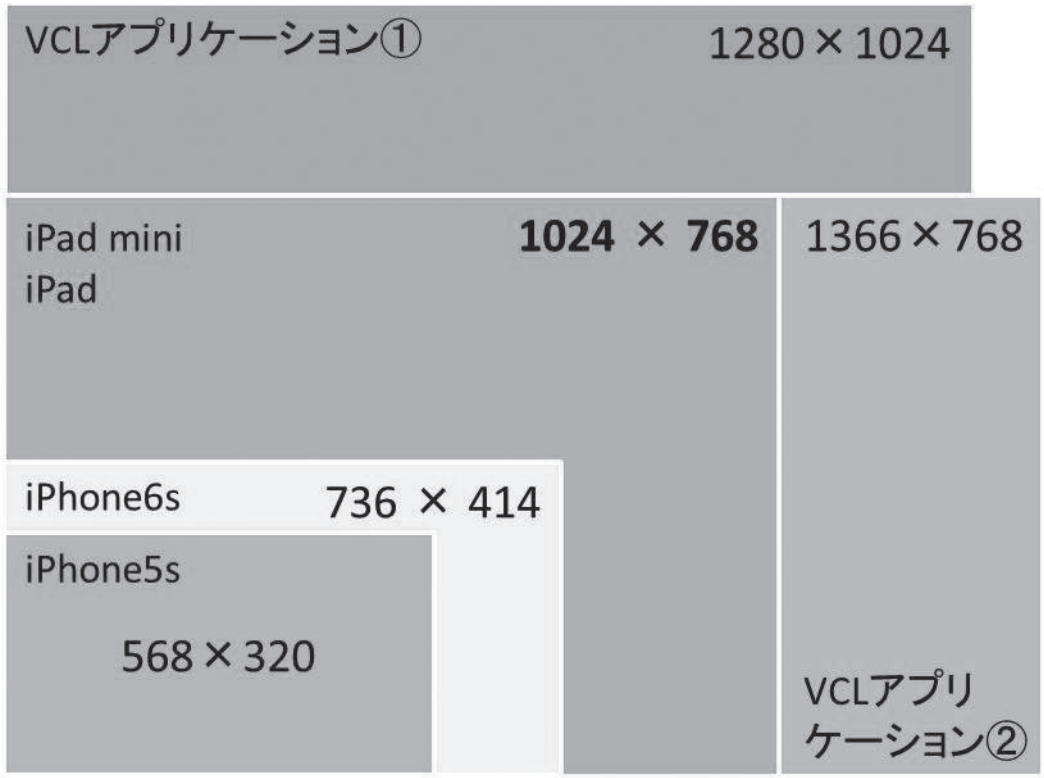
さらに iOS デバイスではキーボードが画面下から表示されるため、画面下部の領域は使用できなくなる可能性もある。

つまり iOS アプリケーションでは、画面サイズ自体の大きさは似ていても、項目間の余白や項目自体のサイズを意識した画面デザインを行うと、実際に使用できる画面領域は PC よりもかなり小さくなる。

3.画面デザインのポイント

ここまで、PC と iOS デバイスにおける画面サイズの違い、iOS デバイスで使用できる画面領域が少なくなる要因について解説した。次はそれらの内容を踏まえ、iOS モバイルアプリケーションの画

図1



※iOSはポイントで比較

図2

面デザインのポイントについて説明する。

1つ目のポイントは、コンポーネントの配置や表示方法を考慮することである。たとえばボタンであれば、アイコンのような見た目で表示することで余分なスペースを取らずに使用できる。またキーボードが表示された際には、画面全体をスクロール表示させることにより、見えなくなった領域も表示可能となる。

2つ目のポイントは、一度に表示する画面項目をなるべく少なくすることである。そのためにはスライドやポップアップを使用し、必要な場合にのみ項目を表示させる方法が有効である。

ただしこのとき、注意すべき点がある。それは、ポップアップやスワイプが有効となる条件をアプリケーション全体でルール決めしておくことである。ポップアップやスワイプが有効となる条件が各画面によってバラバラだと、使用するユーザーが混乱するからだ。

つまり、画面デザインのポイントは大きく以下の2点となる。

①コンポーネントの配置や表示方法を考慮する（4.で詳しく解説）
②必要な場合にのみ必要な情報を表示させる（5.で詳しく解説）

これらのポイントを考慮して開発することで、操作性を損なうことなく画面領域を有効活用できる。

4.画面領域を考慮したデザイン実装例①

前述した画面デザインのポイントを踏まえ、iOSモバイルアプリケーションでの実装方法について順に説明する。

なお、本稿の実装例ではVCLで作成したPCアプリケーションを元に、iOSアプリケーションを作成することを想定している。各機能の全体図は、【図2】のとおりである。

4-1. 新規プロジェクトの作成

まずは、[ファイル|新規作成]より「マルチデバイスアプリケーション-Delphi」を選択する。選択時にダイアログでテンプレート選択画面が表示されるので、「空のアプリケーション」を選択する。これでモバイルアプリケーションのプロジェクトが新規作成できる。【図3】

iOSモバイルアプリケーションの開発には、FireMonkeyを用いる。実装例の解説に入る前に、VCLでの作成時と異なる点について少し触れたい。

まず、「コンポーネントの階層化」についてである。VCLでは、TPanel等のコンテナコンポーネント群やフォームについては親コンポーネントとすることができる。それに対し、FireMonkeyではコンテナコンポーネントやフォームのみに制約されず、他のコンポーネントでも自由に親子関係をもたせることができる。【図4】

こうした階層化により、親コンポーネントに対する処理を子コンポーネントにも反映できる。またそれ以外にも、デザイン時にはオブジェクトをまとめて移動できるといった利点もある。

次に、「Visibleプロパティ」について説明する。Visibleプロパティは、項目の表示・非表示を設定するプロパティである。

VCLでは、設計画面上で設定値（True/False）を切り替えたとしても、常に表示された状態である。それに対してFireMonkeyの設計画面では、Visibleプロパティを Falseとした場合、設計画面上でも非表示となる。

そのため、非表示にしているコンポーネントを編集する際は、一時的にVisibleプロパティをTrueにして編集する必要がある。【図5】

その他にも細かい違いが多くあるが、本稿の実装例では以上の違いを念頭に置けば問題ない。

4-2. ポイント①の実装例

まずは、ポイント①「コンポーネントの配置や表示方法」を考慮した画面デザインの実装方法を説明する。こちらの実装例は、以下のとおりである。

・ボタンのアイコン化
・画面全体に対するスクロールの実装

4-3. ボタンのアイコン化の実装

PCアプリケーションでは、キャプションを表示してボタンの意味を伝えるのが一般的である。本稿の例でも、PCアプリケーションではボタンに「検索」とキャプションを設定している。

それに対してiOSモバイルアプリケーションでは、アイコン化することで画面領域を有効的に使用できる【図6】。実装方法も非常に容易なので、限られた画面領域を有効活用する最もシンプルなテクニックであるといえる。

実装方法は、[ツールパレット|Standard]よりTButtonを選択し、設置する。設置後、StyleLookUpプロパティから［searchtoolbutton］を選択する【図7】。以上の操作で、簡単にボタンをアイコン化できる。アイコンの見た目は、デバイスのOSによって用意されている。

しかし、このままではアイコンに枠や背景色がないので、タップできるかどうかの判断が難しい。そこで、TRectAngleコンポーネントと組み合わせる。これにより、枠や背景色を指定してカスタマイズできる。【図8】

さらに［ツールパレット|Shapes］からTRectAngleを選択する。設置後、[構造ビュー］にて設置したRectAngle1にButton1をドラッグ＆ドロップすることで、コンポーネントを階層化させる。【図9】

あとは、RectAngle1のColorプロパティやX/YRadiusプロパティ（オブジェクトの角の丸さを調整するプロパティ）を任意に変更することで、自由に枠や背景色を指定できる。実装例での設定プロパティは、【図10】のとおりである。

またPCのマウス操作と違い、スマートデバイスのタッチ操作では、意図したタッチポイントとずれて誤動作となることも多い。そのため、TRectAngleの上に配置しているTButtonコンポーネントは、TRectAngleよりもサイズを大きく設定しておいたほうが、タッチに反応しやすく、操作性がよくなる。

4-4. スクロール機能の実装

スクロール機能の実装は、入力項目の多い画面に有効なテクニックである。iOSデバイスでは画面項目への入力時に、画面下部にキーボードが表示される。その際、入力項目部全体にスクロールを設定しておくことで、項目がキーボードに隠れても自由にスクロールできる。

実装方法としては、［ツールパレット|Layouts］よりTVertScrollBoxを選択する。入力項目部に設置し、Alignプロパティを Clientにする。【図11】

図3

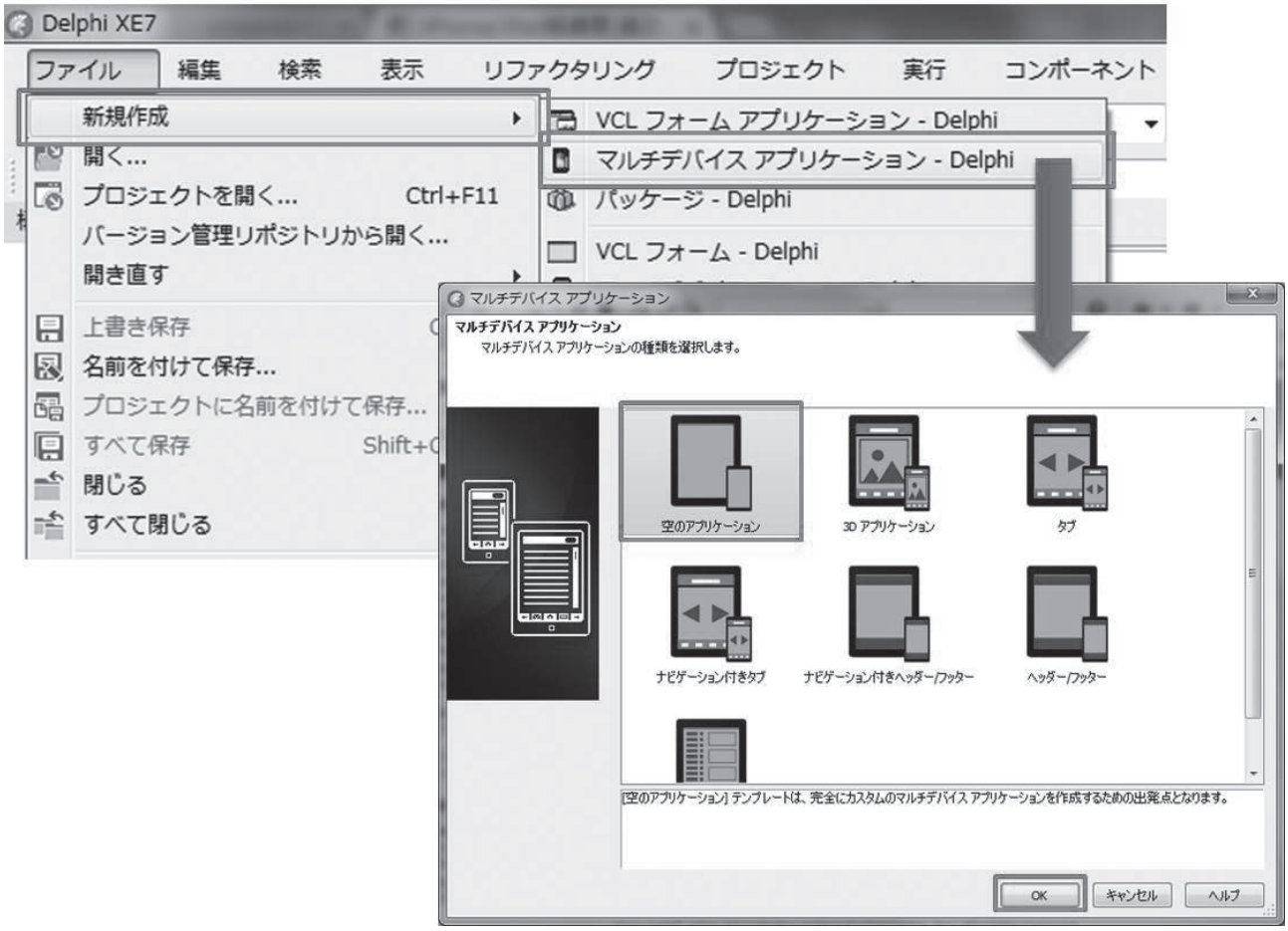

Delphi XE7
ファイル
編集
検索
表示
リファクタリング
プロジェクト
実行
コンポーネント
新規作成
開く...
プロジェクトを開く... Ctrl+F11
バージョン管理リポジトリから開く...
開き直す
上書き保存
名前を付けて保存...
プロジェクトに名前を付けて保存...
すべて保存
閉じる
すべて閉じる
VCL フォーム アプリケーション - Delphi
マルチデバイス アプリケーション - Delphi
パッケージ - Delphi
VCL フォーム - Delphi
マルチデバイス アプリケーション
マルチデバイス アプリケーション
マルチデバイス アプリケーションの種類を選択します。
空のアプリケーション
3D アプリケーション
タブ
ナビゲーション付きタブ
ナビゲーション付きヘッダー/フッター
ヘッダー/フッター
[空のアプリケーション] テンプレートは、完全にカスタムのマルチデバイス アプリケーションを作成するための出発点となります。
OK
キャンセル
ヘルプ

図4


<FireMonkey>
<VCL>
構造
FireMonkeyForm
Edit1
Button1
構造
VCLForm
Edit1
Button1
VCLでは階層化できないコンポーネント同士でも階層化が可能

あとは、任意でVertScrollBox1上に画面項目を設置していく。これにより入力項目部全体を自由に上下スクロールできるようになり、キーボード表示時に項目が隠れた場合も、下にスクロールすることで表示できる。【図12】

5.画面領域を考慮したデザイン実装例②

5-1. ポイント②の実装

ここでは、ポイント②「必要な場合にのみ必要な情報を表示」を踏まえた画面デザインの実装方法について説明する。本稿では、以下の2点を実装する。

・画面をスワイプすることで側面から情報を表示（スライド機能）
・ボタンタップで情報表示（ポップアップ機能）

こうしたスライド機能やポップアップ機能は実装が難しい、と思われるかもしれない。しかし、FireMonkeyではプロパティの設定や少々のコーディングで容易に実装できる。

5-2. スライド機能の実装

スライド機能は、画面の側面から指をスワイプさせることで隠れたリスト等を表示させる機能である。メニューを表示させたり、選択項目の詳細情報を表示させるなど、用途はさまざまである。

実装例では、PCアプリケーションでフッター部に配置しているボタンをスライド部に実装する。【図13】

［ツールパレット|Common Controls］からTMultiViewコンポーネントを選択し、設置する。その後、設置されたMultiView1のModeプロパティをDrawerに変更する。プロパティを変更するとVisibleプロパティがFalseとなるため、スライド内容編集時はVisibleプロパティを一時的にTrueにする。【図14】

以上で、スライド機能を実装できた。あとは任意にスライドで表示する内容を、【図15】のように設定すればよい。

5-3. ポップアップ機能の実装

ポップアップ機能は、ボタンタップなどの動作で画面項目の表示・非表示を切り替えるテクニックである。

こうしたポップアップ機能は、TPopupというコンポーネントを使用することで実装できる。実装例では、PCアプリケーションでヘッダー部のログイン情報を、ボタンタップで表示できるように実装する。

まず、ログイン情報を呼び出すボタンを設置する（ボタンの実装方法は4-4を参照）。実装例での設定プロパティは、【図16】のとおりである。

次に、ポップアップ部を実装する。まずは、［ツールパレット|Standard］よりTPopupを選択し、表示させたい位置に配置する。【図17】

配置時はVisibleプロパティがFalseとなっているので、編集時はTrueに変更する。続いて、［ツールパレット|Shapes］よりTCalloutRectangleを選択し、プロパティの設定を行う。【図18】

そして、最後に表示項目を配置する。本稿では、ログイン情報とログアウトボタンを実装するため、［ツールパレット|Standard］よりTLabelを必要数設置し、4.で解説したTRectAngleを組み合わせたボタンを配置する。あとは配置したコンポーネントを、【図19】のように階層化すれば完成である。【図20】

なお、ポップアップ表示・非表示の処理に関しては、ソースコードを記載する必要がある。こちらの実装方法については、【ソース1】に示す。

6.エフェクトを用いた画面の実装例

6-1. エフェクトの活用

ここまで画面領域を考慮した開発テクニックを題材に説明してきた。モバイルアプリケーションの開発では、画面領域の考慮のほかに、ユーザーが操作しやすい画面設計を行うことも大切な要素の1つである。

たとえば選択形式の画面項目の場合には、チェックボックスを使用することが多い。もちろん標準のTCheckBoxを使用しても問題はないが、本稿ではTCheckBoxの代わりに、エフェクトコンポーネントを利用した選択用ボタンの作成を推奨する。【図21】

選択用ボタンとは、【図21】のようにボタンタップによって凹凸が変化する機能を実装したボタンであり、以下にその実装方法について記載する。

まずエフェクトコンポーネントを使用する際は、4.で触れたコンポーネントの階層化の考え方が基本となる。この実装例では、TRectAngleにTBevelEffect、TInnerGlowEffect、TLabelを階層化させて、実装する。

6-2. エフェクト機能の実装

まず、TRectAngleとTLabelを画面に設置する。そこへ［ツールパレット|Effects］より非表示コンポーネントであるTBevelEffectおよびTInnerGlowEffectを設置し、各コンポーネントを【図22】のように階層化する。

TBevelEffectでは親コンポーネントの奥行きを設定でき、TInnerGlowEffectは親コンポーネントを内側に向けて発光させる効果がある。

まず、先ほど配置したBevelEffect1とInnerGlowEffect1のプロパティ設定を、【図23】のように設定する。あとは、Label1のOnMouseDownイベントに、【ソース2】のようにコーディングするだけで、実装は完了である。

7.まとめ

本稿では、モバイルアプリケーションの開発の際に課題となることが多い画面デザインに対する工夫を考察し、その具体的なテクニックを説明してきた。

iOSモバイルアプリケーションでは、配置する項目のサイズや項目間の余白を意識して画面デザインする必要があるので、使用できる画面領域は必然的に狭くなる。

本稿で紹介した画面領域を考慮した開発テクニックは、すべて容易に実装でき、初めてモバイルアプリケーションを開発する方であっても十分活用できる。

またモバイルアプリケーションの画面デザインで最も参考になるのは、実際のストアなどに公開されているアプリケーションである。とくにダウンロード数が多いアプリケーションは、利用するユーザーが多い分、画面デザインや操作性も洗練されていることが多い。

モバイルアプリケーションはデザインの工夫次第で、使い勝手が大きく変わるので、優れた画面設計や実装テクニックをどんどん取り入れていくことが重要である。 M

図5

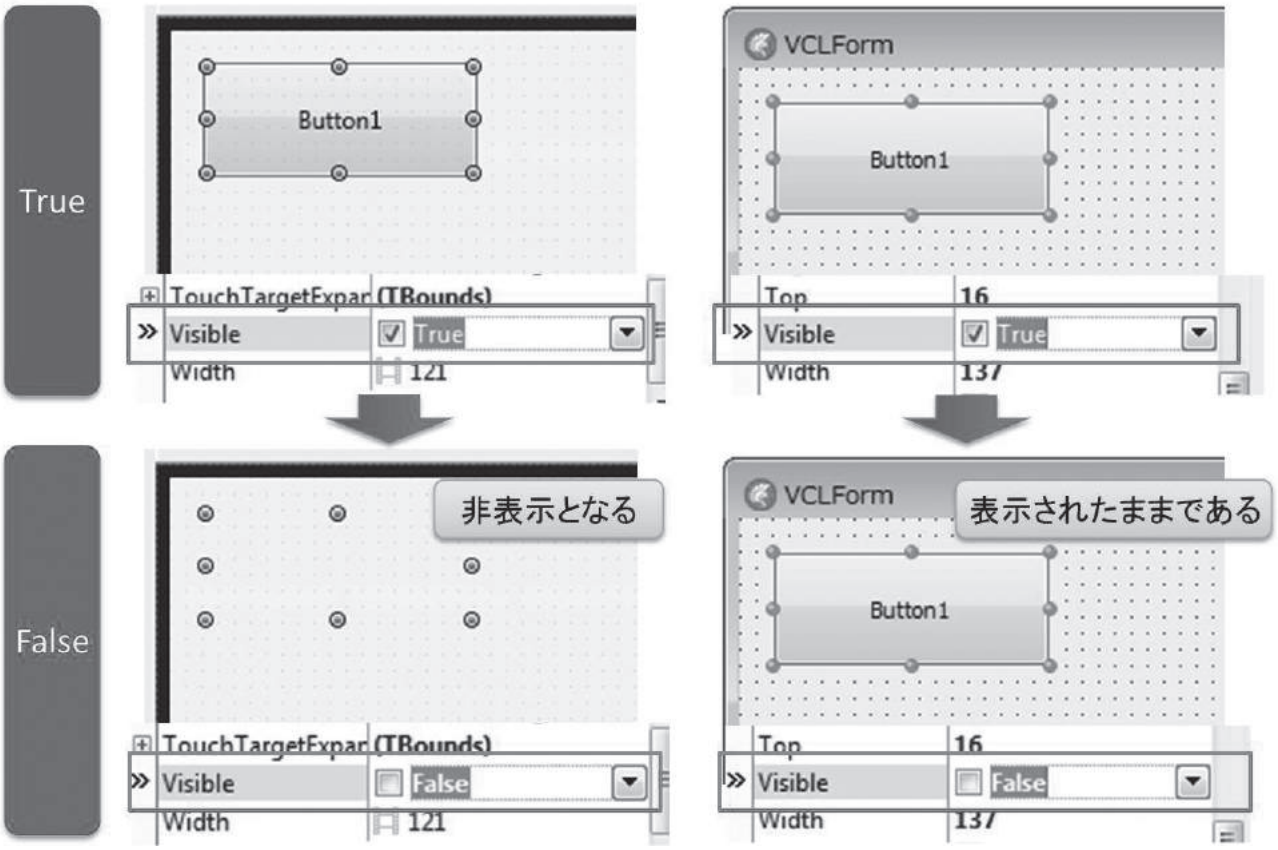

True
Button1
TouchTargetExpan (TBounds)
Visible
True
Width
121
VCLForm
Button1
Top
16
Visible
True
Width
137
False
非表示となる
TouchTargetExpan (TBounds)
Visible
False
Width
121
VCLForm
表示されたままである
Button1
Top
16
Visible
False
Width
137

図6

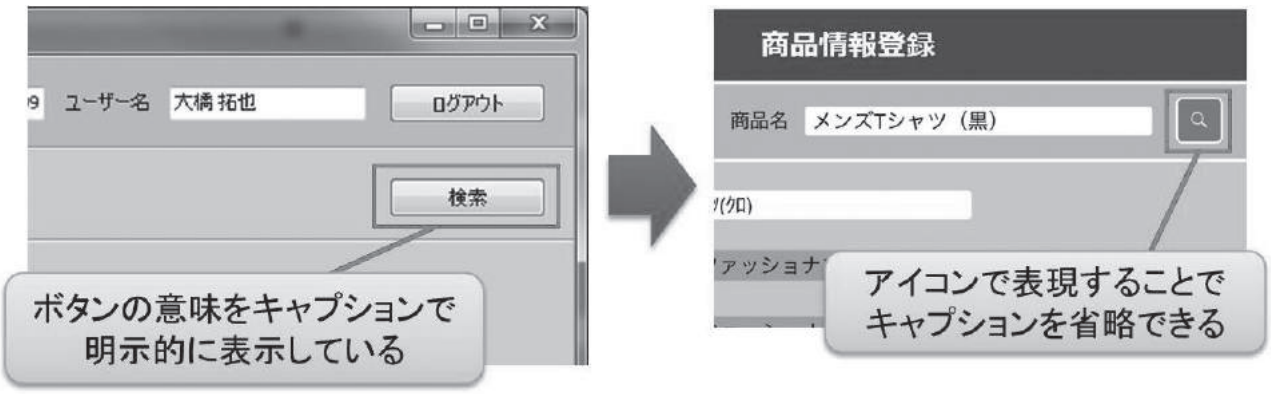
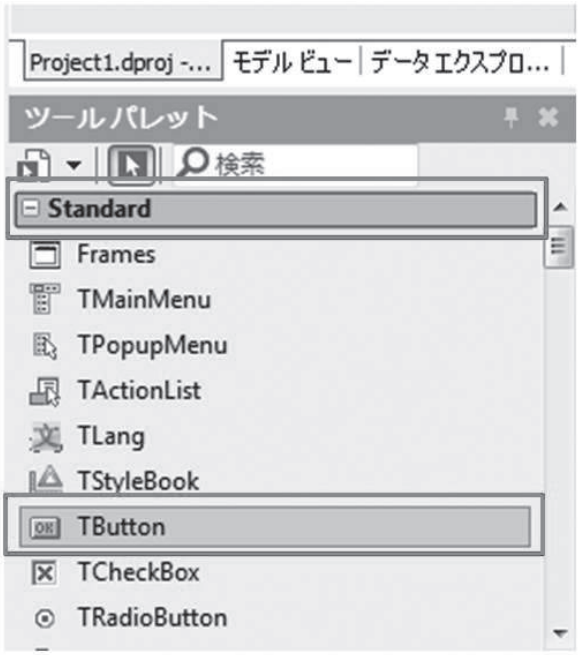
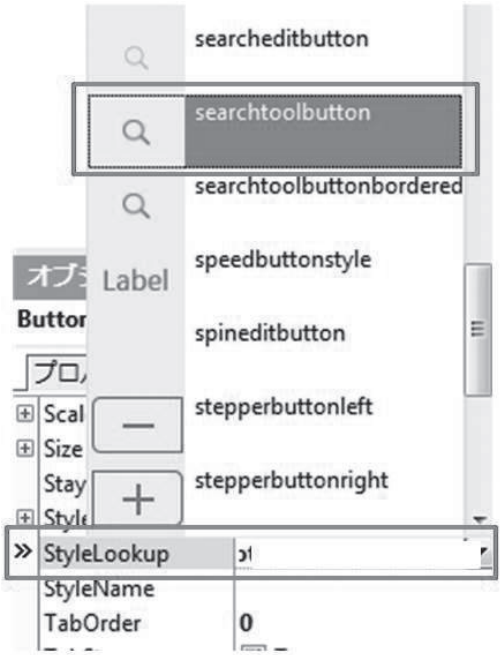

ユーザー名
大橋 拓也
ログアウト
検索
ボタンの意味をキャプションで
明示的に表示している
商品情報登録
商品名
メンズTシャツ（黒）
アイコンで表現することで
キャプションを省略できる

図7

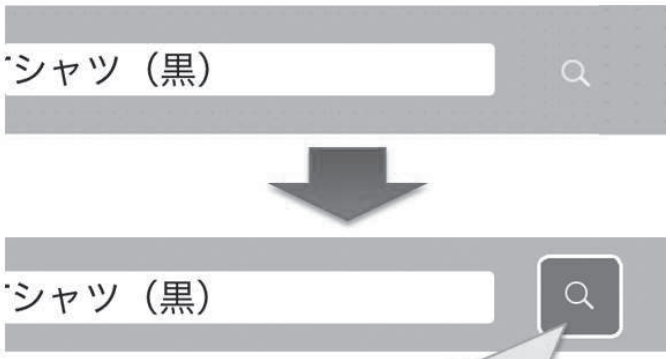


ツールパレットからTButtonを選択



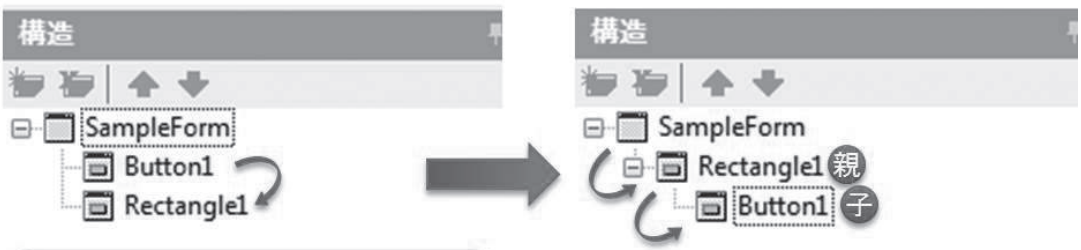
StyleLookupプロパティ設定

図8



TRectAngleと組み合わせることで
枠や背景色を設定することができる

図9



ドラッグ&ドロップで
親子関係を持たせる

階層構造ができる

図10

Button1設定プロパティ

Align	Center
StyleLookup	searchtoolbutton

RectAngle1設定プロパティ

Hight	38
Width	38
XRadius	5
YRadius	5

図11

図12

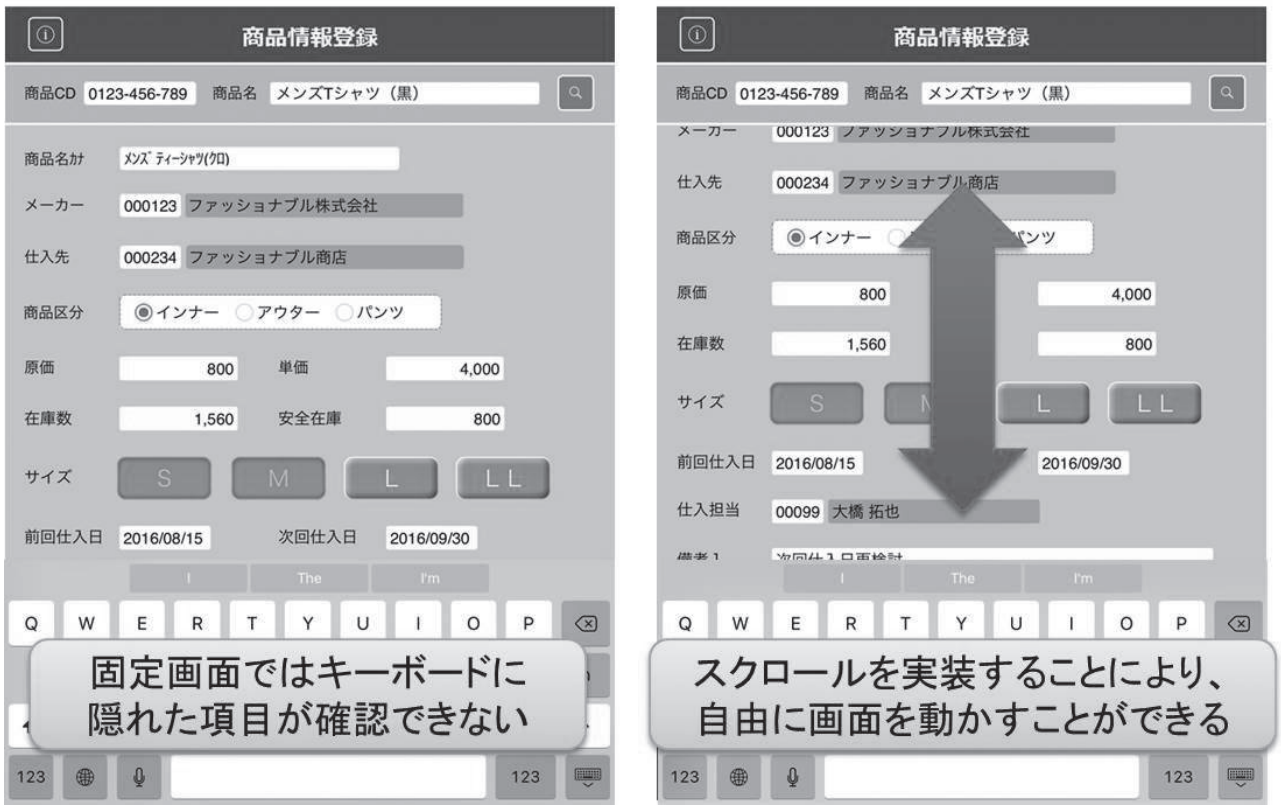

商品情報登録
商品CD 0123-456-789
商品名 メンズTシャツ（黒）
商品名カナ
メーカー 000123 ファッショナブル株式会社
仕入先 000234 ファッショナブル商店
商品区分 インナー アウター パンツ
原価 800
単価 4,000
在庫数 1,560
安全在庫 800
サイズ S M L LL
前回仕入日 2016/08/15
次回仕入日 2016/09/30
固定画面ではキーボードに
隠れた項目が確認できない
仕入担当 00099 大橋 拓也
スクロールを実装することにより、
自由に画面を動かすことができる

図13

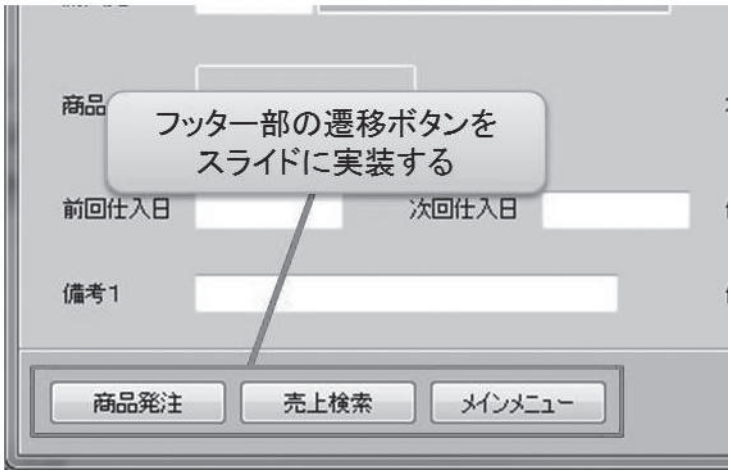

フッター部の遷移ボタンを
スライドに実装する
前回仕入日
次回仕入日
備考1
商品発注
売上検索
メインメニュー

図14

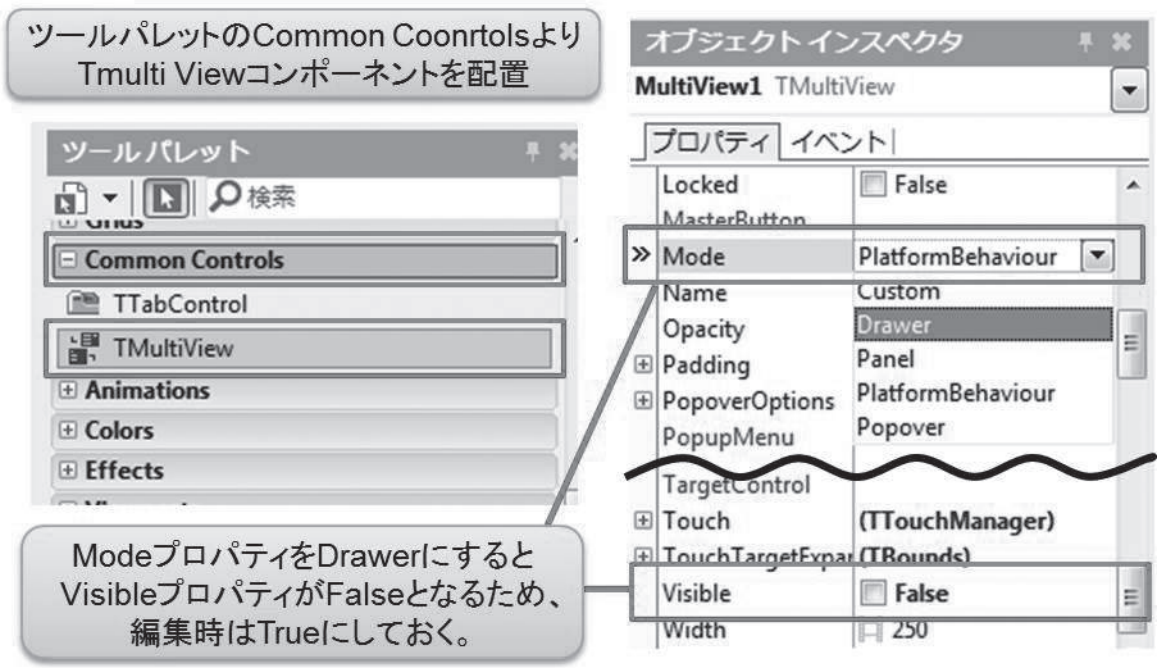


図15

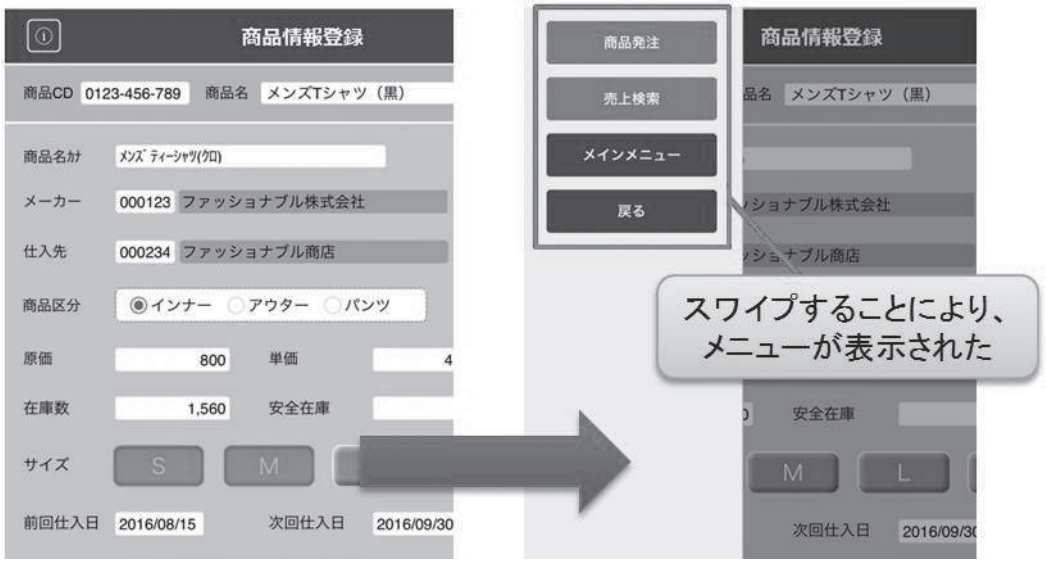


図16

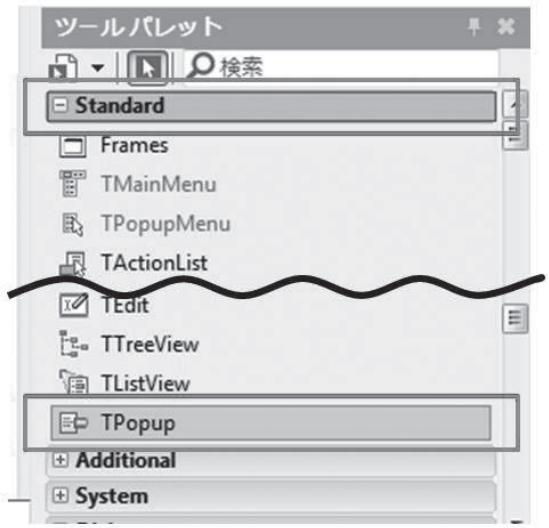
Button2設定プロパティ

Align	Center
StyleLookup	infotoolbutton

RectAngle2設定プロパティ

Fill - Color	Navy
Stroke - Color	White
XRadius	5
YRadius	5

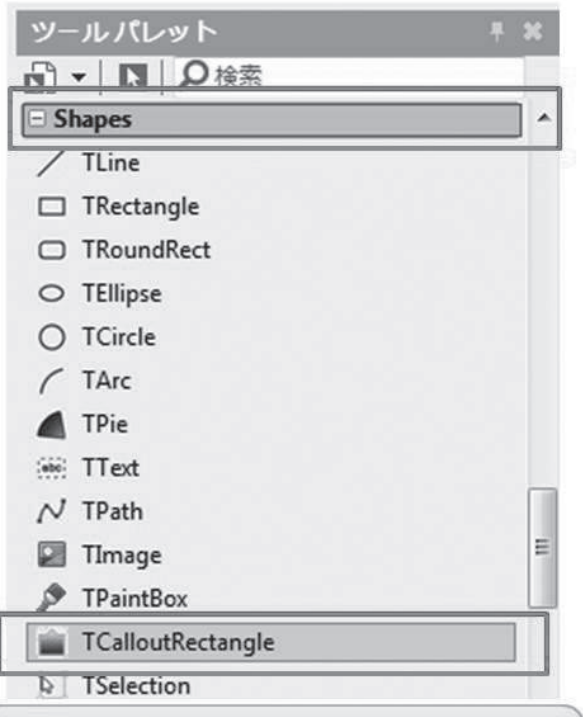
図17



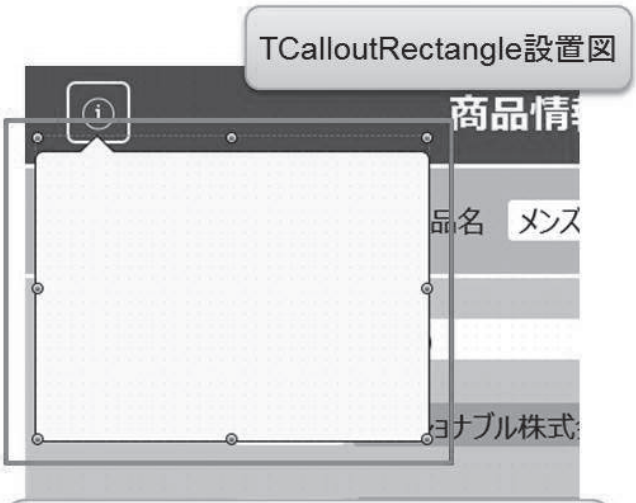
ツールパレットのStandardより
TPopupを設置する

ポップアップ表示をさせたい位置に
配置し、任意のサイズに設定する

図18



ツールパレットのShapesより
TCalloutRectangleを設置する



CalloutOffsetプロパティにより
吹き出し元の位置を変更できる

CalloutRectangle1設定プロパティ

Align	Client
CalloutOffset	31
Fill - Color	White
XRadius	5
YRadius	5

図19

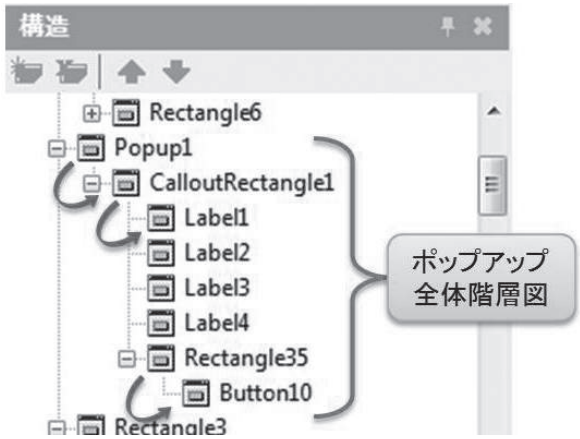
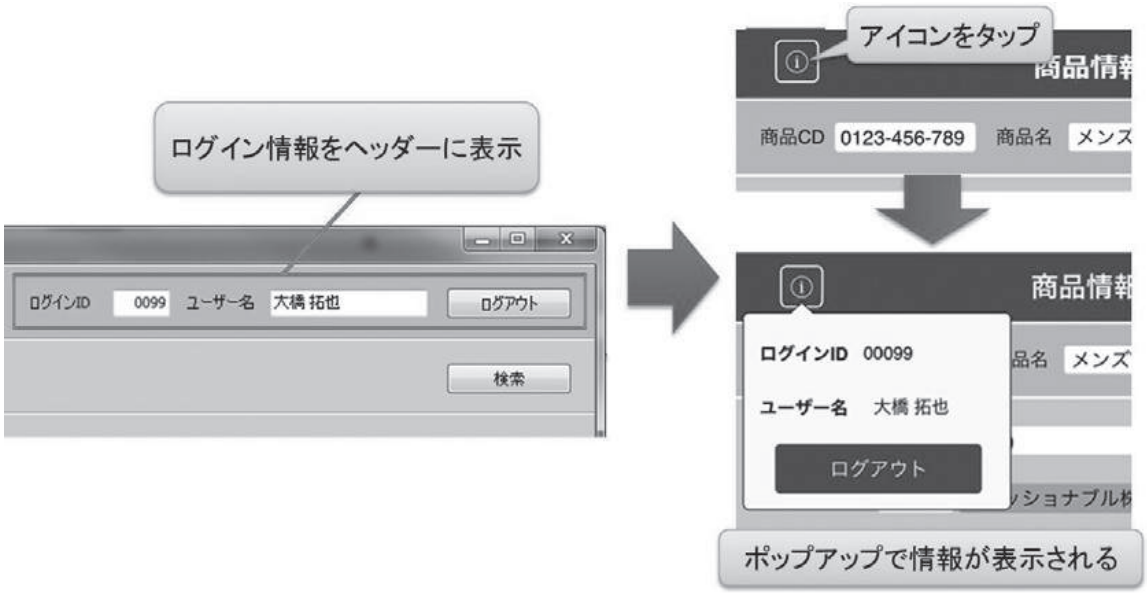


図20



ソース1

```
{*************************************************************************
 目的：ログイン情報押下時処理
 引数：
 戻値：
*************************************************************************}
procedure TForm1.Button2Click(Sender: TObject);
begin
  // ポップアップ表示切替
  Popup1.Visible := not Popup1.Visible;
  Popup1.BringToFront;
end;

{*************************************************************************
 目的：画面タッチ時処理
 引数：
 戻値：
*************************************************************************}
procedure TForm1.FormTouch(Sender: TObject; const Touches: TTouches;
  const Action: TTouchAction);
begin
  // ポップアップ終了
  if Popup1.Visible and
     (not Popup1.PointInObject(Touches[0].Location.X, Touches[0].Location.Y)) and
     (not Button2.PointInObject(Touches[0].Location.X, Touches[0].Location.Y)) then
  begin
    Popup1.Visible := False;
  end;
end;
```

アイコン以外の場所にタッチした場合、
ポップアップを非表示にする

図21

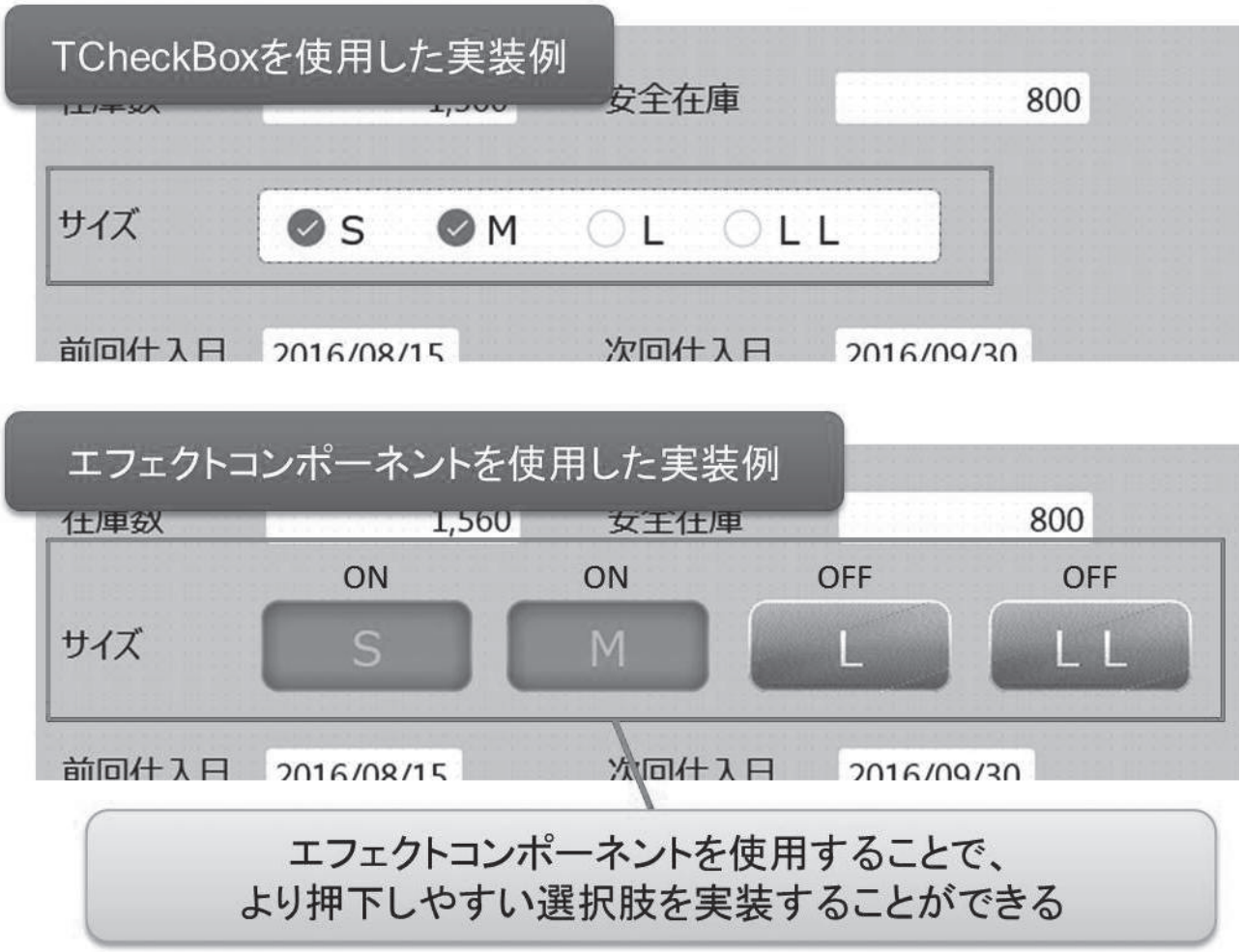

TCheckBoxを使用した実装例
安全在庫
800
サイズ
S
M
L
L L
前回仕入日
2016/08/15
次回仕入日
2016/09/30
エフェクトコンポーネントを使用した実装例
在庫数
1,560
安全在庫
800
ON
ON
OFF
OFF
サイズ
S
M
L
L L
前回仕入日
2016/08/15
次回仕入日
2016/09/30
エフェクトコンポーネントを使用することで、
より押下しやすい選択肢を実装することができる

図22

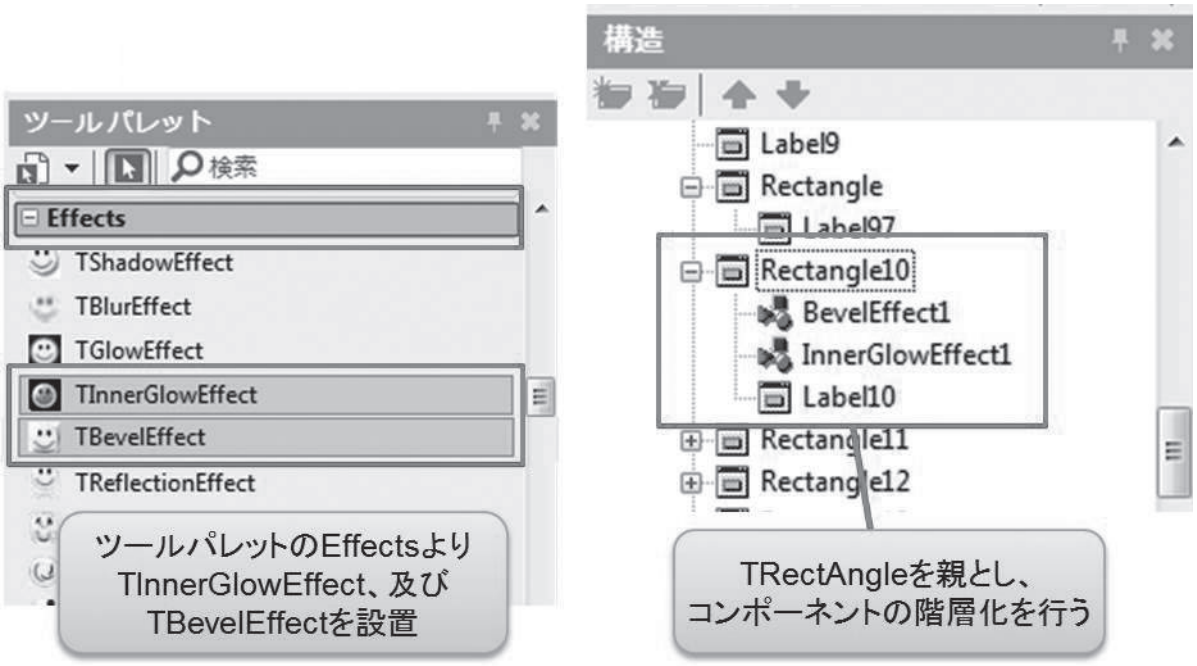
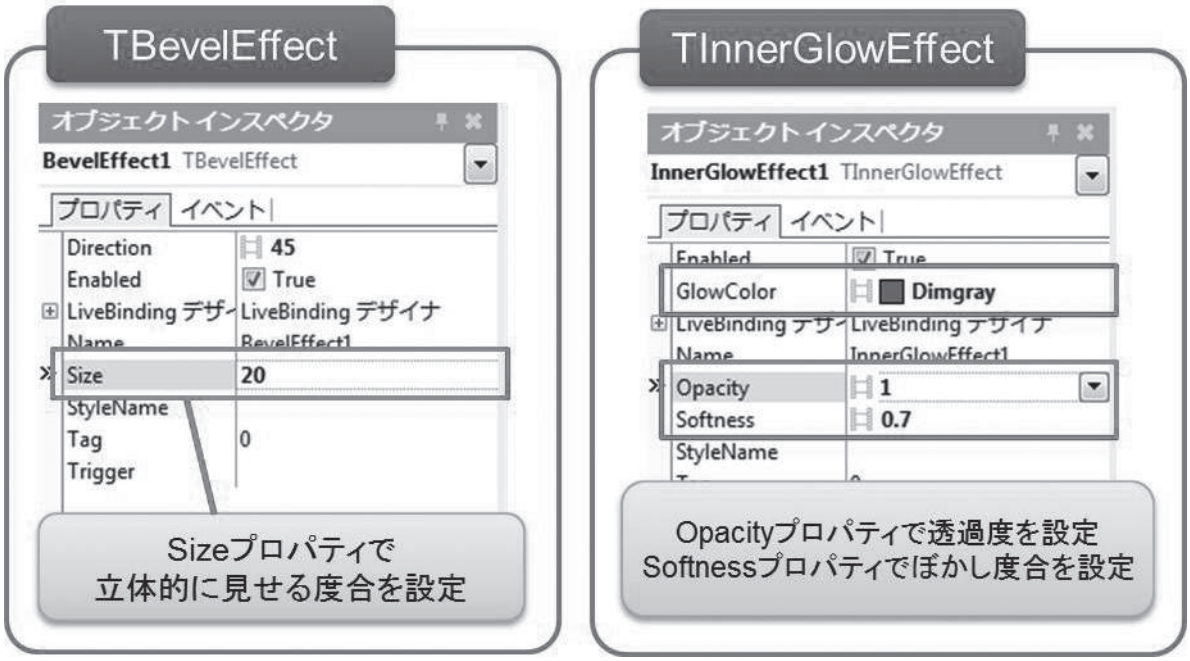

ツールパレット
検索
Effects
TShadowEffect
TBlurEffect
TGlowEffect
TInnerGlowEffect
TBevelEffect
TReflectionEffect
ツールパレットのEffectsより
TInnerGlowEffect、及び
TBevelEffectを設置
構造
Label9
Rectangle
Label97
Rectangle10
BevelEffect1
InnerGlowEffect1
Label10
Rectangle11
Rectangle12
TRectAngleを親とし、
コンポーネントの階層化を行う

図23



ソース2

```
  private
    { private 宣言 }
    FPushFlg: Boolean;  // 押下フラグ
  public
    { public 宣言 }
  end;

var
  FireMonkeyForm: TFireMonkeyForm;

implementation

{$R *.fmx}

{*******************************************************************************
 目的: 選択ボタン 押下時処理
 引数:
 戻値:
*******************************************************************************}
procedure TFireMonkeyForm.Label1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Single);
begin
  if FPushFlg then
  begin
    BevelEffect1.Enabled      := True;
    InnerGlowEffect1.Enabled  := False;
    Label1.Opacity            := 1;
    FPushFlg                  := False;
  end
  else
  begin
    BevelEffect1.Enabled      := False;
    InnerGlowEffect1.Enabled  := True;
    Label1.Opacity            := 0.5;
    FPushFlg                  := True;
  end;
end;
```

選択状態ON/OFF 切替用フラグ

ON から OFF へ変更の処理

OFF から ON へ変更の処理

【処理内容】
・TBevelEffectの切替(Enabled)
・TInnerGlowEffectの切替(Enabled)
・TLabelの透明度設定(Opacity)
・フラグの切替

福井 和彦

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] 新データベースエンジンFireDACを使ってみよう!

略歴
1972 年 3 月 20 日生まれ
1994 年 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を担当しており、要件確認から納品・フォローに至るまで、システム開発全般に携わっている。また、Delphi/400 の導入支援やセミナーの講師も行っている。

1.はじめに

業務アプリケーションを開発していくうえで、データベースの利用は不可欠である。そして、Delphi/400 はデータベースを使ったアプリケーション開発を得意としている。なぜなら、Delphi/400 はデータベースエンジンという機能を備えており、これによりどのデータベースに対しても、共通のプログラミングで簡単に開発できるからである。

このデータベースエンジンと各種データベースドライバを組み合わせることで、IBM i や SQL Server などさまざまなデータベースへ接続できる。【図 1】

これまでの開発で使用されてきたデータベースエンジンには、BDE や dbExpress がある。本稿で主題としている「FireDAC」は、BDE や dbExpress に続く新しいデータベースエンジンである。

FireDAC 自体は Delphi の XE3 から実装されており、これまでも Oracle や SQL Server などでは使用可能であった。そして Delphi 10 Seattle からは、Delphi/400 のドライバが対応し、IBM i でも活用可能になっている。

そこで本稿では、FireDAC で IBM i を利用する基本的な方法、そしてすでに BDE や dbExpress を使用して IBM i へ接続しているプログラムを FireDAC へ移行するポイントについて説明する。

2.FireDACを使った IBM i アプリケーション開発

2-1. FireDAC とは

本題に入る前に、FireDAC はこれまでのデータベースエンジンと比べて、どのような違いがあるかを確認する。FireDAC の特徴としては、以下の点が挙げられる。

(1) Windows 32bit ／ 64bit に対応
(2) FireMonkey に対応
(3) BDE のような初期インストールは不要（配布が簡単）
(4) 双方向データセット形式
(5) 高いパフォーマンスのデータアクセス（BDE と同等以上）

FireDAC と BDE、dbExpress との機能比較については【表 1】に示す。

これらの特徴や比較から、FireDAC は dbExpress と同様の環境対応機能があり、BDE と同等の双方向データセット形式を備えていることがわかる。

つまり、FireDAC は両データベースエンジンのよい部分を組み合わせた新しいデータベースエンジンといえる。次に、この FireDAC を使用して IBM i へ接続し、ファイルを参照する基本的な使い方を説明していく。

2-2. FireDAC の使い方

FireDAC で IBM i へ接続するために使用する基本コンポーネントには、以下が用意されている。

(1) TFDConnection

データベースへの接続を制御するコ

図1

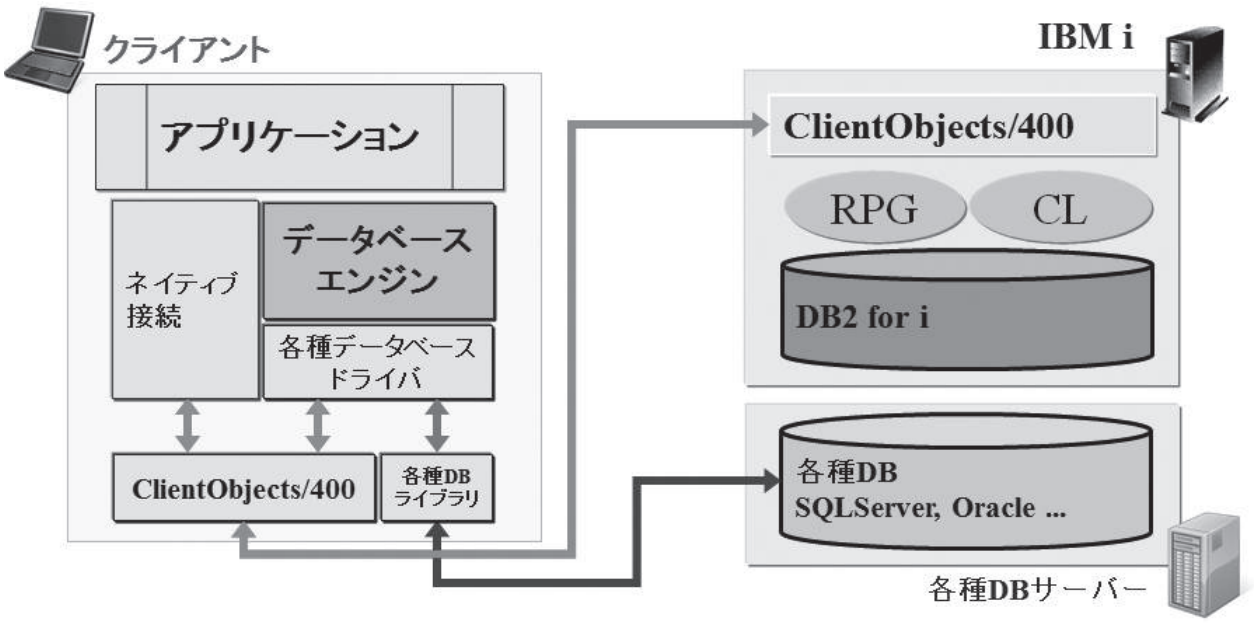


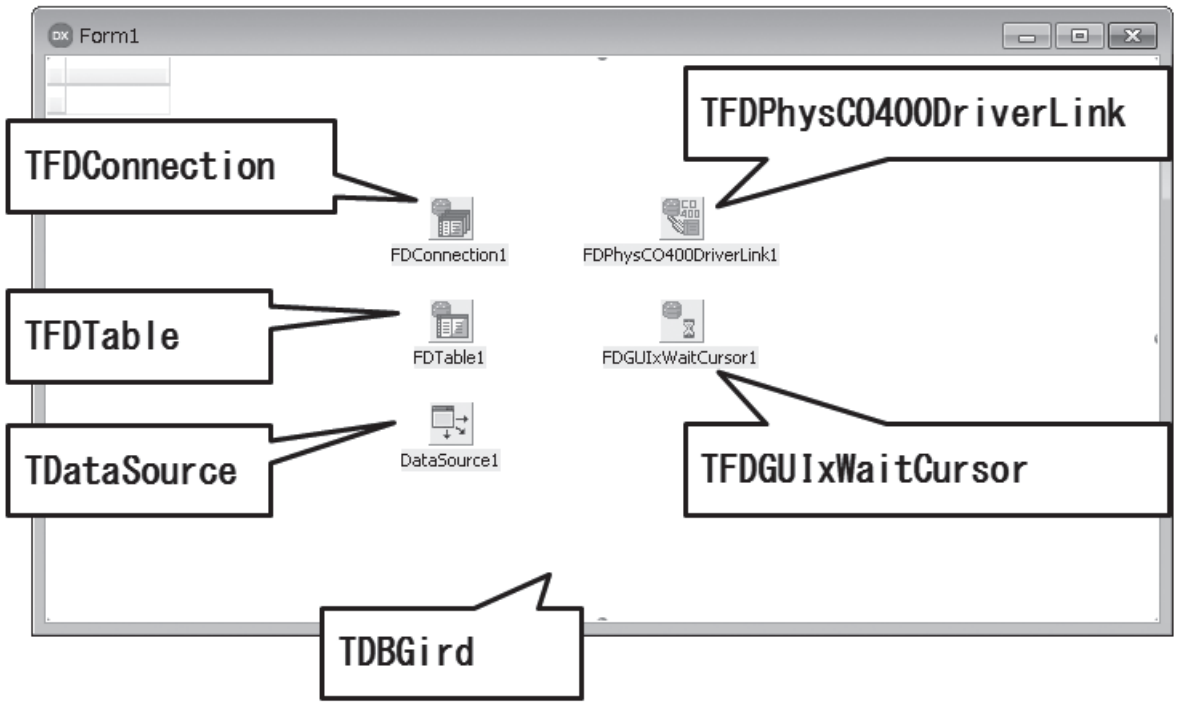
表1

FireDACと他のデータベースエンジンとの比較表

	FireDAC	dbExpress	BDE
Windows32bit	○	○	△
Windows64bit	○	○	×
FireMonkey	○	○	×
初期インストール	不要	不要	必要
データセット形式	双方向	単方向	双方向
速度	○	△	○

※dbExpressの速度は単方向でClientDataSetの使用率が高いため△としている

図2



ンポーネント
(2) TFDPhysCO400DriverLink
TFDConnection に Delphi/400 の IBM i 用ドライバ情報を提供するコンポーネント
(3) TFDTable
単一のファイルを指定して、データを取得・操作するコンポーネント
(4) TFDQuery
SQL を実行して、データを取得・操作するコンポーネント
(5) TFDGUIxWaitCursor
待機カーソルなどを制御するコンポーネント

それでは FireDAC を使用して IBM i へ接続し、ファイルを参照する手順を順番に確認していく。ファイルへの接続は、TFDTable コンポーネントを使用した基本的な構成とする。

①コンポーネントの配置

【図 2】に従って、新規フォームに各コンポーネントを配置する。配置できたら、FireDAC の各コンポーネントのプロパティを順番に設定していく。

最初に IBM i への接続設定を、TFDConnection コンポーネントで行う。フォームに貼り付けた TFDConnection コンポーネントをダブルクリックすると、FireDAC 接続エディタが起動する。【図 3】

起動した FireDAC 接続エディタの上部にある、接続定義名のプルダウンより“CO400DEF”を指定すると、パラメータが表示される。パラメータの“Database”“User_Name”“Password”“ODBCAdvanced”を、【図 4】に従って設定する。

TFDPhysCO400DriverLink コンポーネントと TFDGUIxWaitCursor コンポーネントは、フォームに貼り付けるだけでとくに設定を行う必要はない。

続いて、ファイルの参照設定を TFDTable コンポーネントで行う。TFDTable コンポーネントの Connection プロパティは、TFDConnection コンポーネントが自動で初期セットされているので、TableName プロパティに参照するファイル名を設定する。【図 5】

このプログラムでは、【図 4】の TFDConnection コンポーネントの ODBCAdvanced パラメータにライブラリ名を指定しているため、TableName プロパティではリストが自動表示されて選択できる。

あとは TDataSource コンポーネントの DataSet プロパティと、TDBGrid の DataSource プロパティを設定すれば、各コンポーネントの設定は完了となる。【図 6】

ここまでで、プログラム上の設定は完了である。

実際に FireDAC を使用してデータへアクセスするには、TFDTable コンポーネントの Active プロパティを True にする。これによって、TDBGrid 上にデータを表示できる。【図 7】

FireDAC は新しいデータベースエンジンではあるが、Delphi/400 ではこれまでのプログラムと互換性を維持できる形でコンポーネントが用意されている。ここまでの実装手順を確認すると、BDE や dbExpress の開発とほとんど違いはなく、また TClientDataSet を必要としない分、よりシンプルに開発できることがわかる（もちろん TClientDataSet を使用することも可能である）。

3. 既存プログラムの FireDACへの移行

3-1. FireDAC へのプログラム変更

ここまで FireDAC の新規プログラムを作成する方法を説明したが、次に BDE や dbExpress で作成されているプログラムを FireDAC へ移行する手順について説明する。

本稿では、次のような RPG を使った標準的な仕組みの照会画面を題材に、FireDAC へ変更するポイントを確認していく。

FireDAC へ変更する照会画面プログラムの処理

(1) データを RPG で抽出する
(2) 抽出データを QTEMP のワークファイルに作成する
(3) ワークファイルを BDE または dbExpress で画面表示する

3-2. BDE からの移行ポイント

ここでは、BDE から FireDAC への変更方法について説明する。変更する BDE の照会画面の構成は、【図 8】のとおりである。

照会画面の構成では、上段に抽出条件を指定する項目と検索ボタンを配置し、中段に明細表を配置している。動作としては、検索ボタンを押下することで抽出データを明細表に表示する。

また使用しているコンポーネントと設定しているプロパティについては、【表 2】と【図 9】のとおり、検索実行時のソースは【図 10】のとおりである。データ抽出には RPG を用いているので、TAS400 コンポーネントと TCall400 コンポーネントを使用しているが、データ抽出の処理ロジックは主題から外れるため、本稿では割愛する。

以下に、プログラムで BDE を使用している箇所を FireDAC に変更する手順を説明する。

①データベース接続処理を BDE から FireDAC へ変更

BDE を使用して作成した照会画面に、2-2 に記載した FireDAC の基本コンポーネントである「TFDConnection」「TFDTable」「TFDPhysCO400DriverLink」「TFDGUIxWaitCursor」の 4 つを、【図 11】のように配置し、TFDConnection コンポーネントのプロパティを設定する。

ただし BDE の照会画面では、【図 9】にあるように TDatabase コンポーネントのプロパティ設定をソースで行っている。そのため、TFDConnection コンポーネントには接続定義名に“CO400DEF”だけを設定し、残りの設定はソースで実装する。

ここから、ソースの変更箇所の詳細を説明する。

【図 12】のように、FormCreate イベントに記述している TDatabase コンポーネントの設定を、TFDConnection コンポーネントの設定に変更する。

ライブラリリストを使う場合、BDE では TDatabase コンポーネントの“LIBRARY NAME”に、“*LIBL”をセットするが、FireDAC の場合は TFDConnection コンポーネントの“ODBCAdvanced”に“LibraryOption=（ブランク、シングルコーテーションなし）”をセットする。これでデータベース接続処理の FireDAC への変更は、完

図3

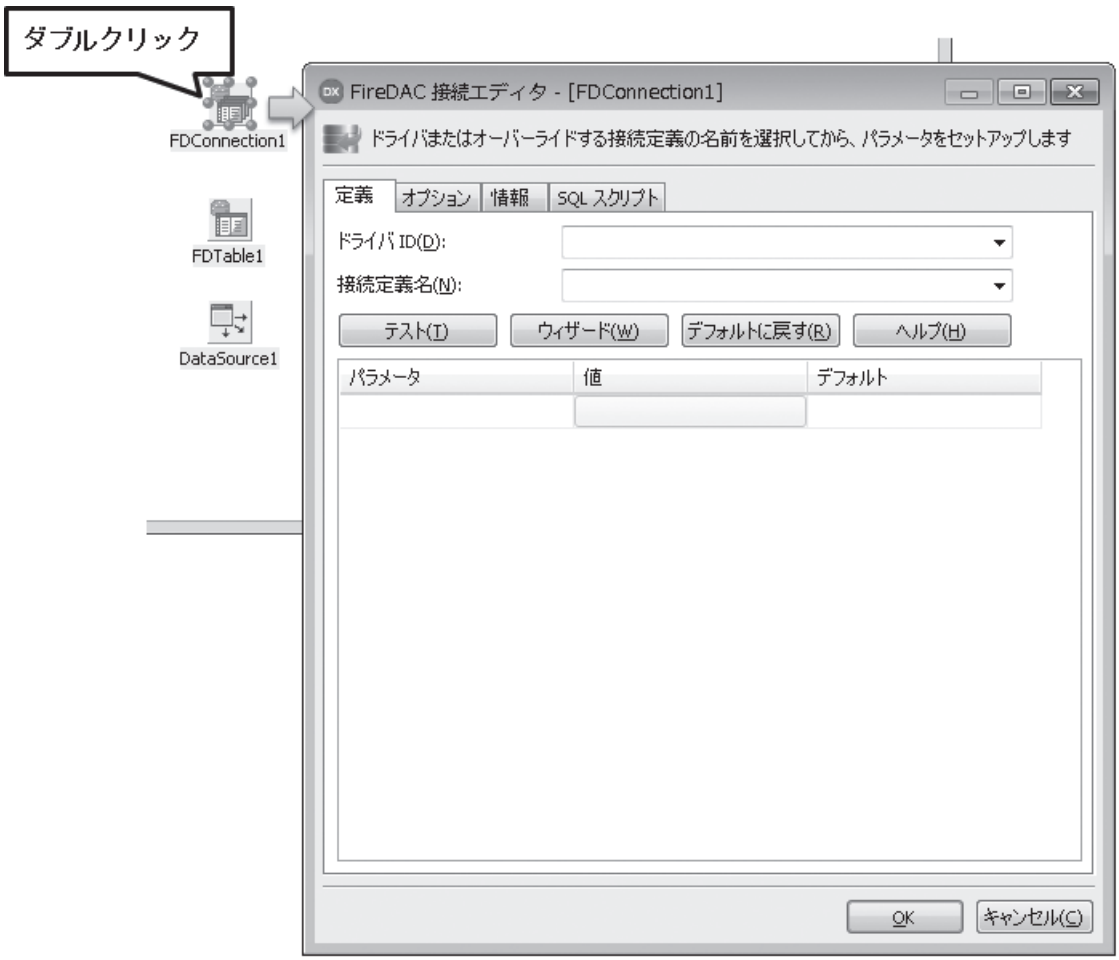
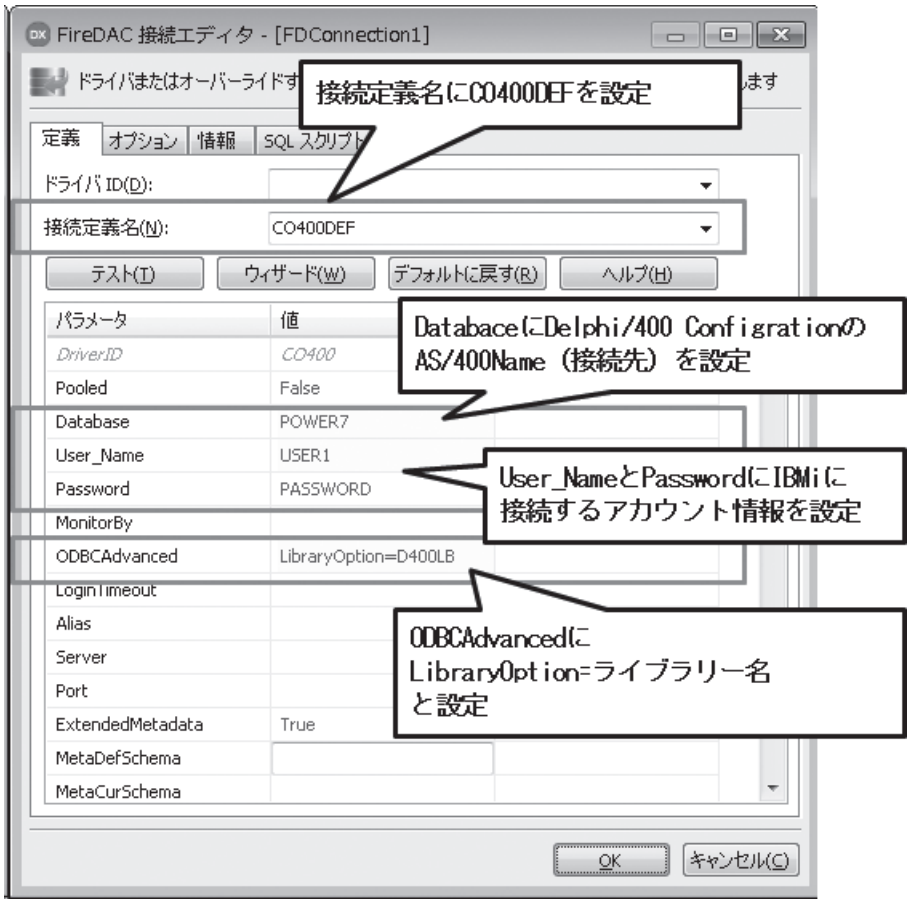

ダブルクリック
FDConnection1
FDTable1
DataSource1
FireDAC 接続エディタ - [FDConnection1]
ドライバまたはオーバーライドする接続定義の名前を選択してから、パラメータをセットアップします
定義
オプション
情報
SQL スクリプト
ドライバ ID(D):
接続定義名(N):
テスト(T)
ウィザード(W)
デフォルトに戻す(R)
ヘルプ(H)
パラメータ
値
デフォルト
OK
キャンセル(C)

図4


FireDAC 接続エディタ - [FDConnection1]
接続定義名にCO400DEFを設定
定義
オプション
情報
SQL スクリプト
ドライバ ID(D):
接続定義名(N):
CO400DEF
テスト(T)
ウィザード(W)
デフォルトに戻す(R)
ヘルプ(H)
パラメータ
値
DriverID
CO400
Pooled
False
Database
POWER7
User_Name
USER1
Password
PASSWORD
MonitorBy
ODBCAdvanced
LibraryOption=D400LB
LoginTimeout
Alias
Server
Port
ExtendedMetadata
True
MetaDefSchema
MetaCurSchema
DatabaseにDelphi/400 ConfigrationのAS/400Name（接続先）を設定
User_NameとPasswordにIBMiに接続するアカウント情報を設定
ODBCAdvancedにLibraryOption=ライブラリー名と設定
OK
キャンセル(C)

了である。

②データ表示処理をBDEからFireDACへ変更

次に、TFDTableコンポーネントのConnectionプロパティに、TFDConnectionコンポーネントが指定されていることを確認し、TableNameプロパティに参照するファイル名を設定する。

このプログラムでは、TFDConnectionコンポーネントにライブラリ名を指定していないため、リストから選択するのではなく、ファイル名を直接設定する。あとはTDataSourceコンポーネントのDataSetプロパティを、TTableからTFDTableに変更する。【図13】

最後に、【図14】のようにソース上でTTableコンポーネントを使用している部分をTFDTableコンポーネントに変更すれば、変更は完了となる。

変更したプログラムを実行すると、FireDACの仕組みでIBM iのデータを照会可能なことが確認できる。【図15】

ただし明細表のタイトル行やデータ書式を設定している場合、それが反映されていないはずである。

そのため、TFDTableコンポーネントで各フィールドの詳細設定を行う必要がある。項目ごとにDisplayLabelプロパティやDisplayFormatプロパティ等を設定していくのは面倒な作業だが、移行の場合はすでにBDEのTTableコンポーネントのフィールドに設定されているはずなので、それをコピーすればよい。

【図16】のように、TTableコンポーネントとTFDTableコンポーネントのフィールドエディタを開き、TTableコンポーネントの全フィールドを選択したあと、コピー＆ペーストでTFDTableコンポーネントのフィールドエディタへ貼り付ける。これだけでフィールド設定情報のコピーは完了である。TFDTableコンポーネントの各フィールドのプロパティを確認すると、TTableの設定がすべてコピーできているとわかる。

再度プログラムをコンパイルして実行すると、変更しているプログラムによっては、【図17】のようなエラーの出るケースがある。これは、FireDACのデータ型（数値）のマッピングルールが、これまでのデータベースエンジンと異なる場合に発生するエラーである。ここではTFDTableコンポーネントのフィールドはInteger型であるが、FireDACは数値型をBCD型として認識するため、このようなエラーが発生する。

このエラーに対応する方法として、FireDACにはデータ型のマッピングルールを変更する機能が用意されている。データ型のマッピングルールの変更は、TFDConnectionコンポーネントで設定できる。

まずは、TFDConnectionコンポーネントをダブルクリックして、FireDAC接続エディタを開く。次にオプションタブを選択し、オプションタブにある「継承したルールを無視」チェックボックスをチェックすると、データマッピングルールの明細が入力可能になる。

明細は【図18】のように設定し、OKボタンでFireDAC接続エディタを完了する。これで、FireDACで数値項目をBCD型で認識した場合、Integer型に変換可能となる。再度プログラムをコンパイルして実行すると、データ型のエラーが解消されて、正しく実行できる。【図19】

ここまで確認できたら、FireDACで従来どおりの動作が実現できたことになる。使わなくなったBDEのTDatabaseコンポーネントとTTableコンポーネントを削除して、FireDACへの移行が完了である。

3-3. dbExpressからの移行ポイント

ここでは、dbExpressからFireDACへの変更方法について説明する。変更するdbExpressの照会画面の構成は、【図20】のとおりである。

この照会画面の構成や機能は、前述したBDEの照会画面と同じである。また、使用しているコンポーネントと設定しているプロパティについては【表3】と【図21】のとおり、検索実行時のソースについては【図22】のとおりである。

BDEとの大きな違いはデータを表示する際に、TDataSetProviderコンポーネントとTClientDataSetコンポーネントを使用している点である。これはdbExpressを使用して、直接画面にデータを表示する際に必要な構成となっている。

以下に、dbExpressを使用している箇所をFireDACに変更する手順を説明する。

①データベース接続処理をdbExpressからFireDACへ変更

前述したBDEのケースと同様に、dbExpressを使用して作成した照会画面に、FireDACの基本コンポーネントである「TFDConnection」「TFDPhysCO400DriverLink」「TFDTable」「TFDGUIxWaitCursor」の4つを、【図23】のように配置し、TFDConnectionコンポーネントのプロパティを設定する。

ただし、dbExpressの照会画面もBDEのケースと同様に、TSQLConnectionコンポーネントのプロパティ設定をソースで行っている。従ってソースの変更は、【図24】のように、TFDConnectionを設定する。これで、データベース接続処理のFireDACへの変更が完了である。

②データ表示処理をdbExpressからFireDACへ変更

dbExpressからFireDACへの変更では、TClientDataSetコンポーネントをそのまま利用することもできる。その場合、フィールド設定の移行も必要ないので、BDEより簡単にFireDACへ移行できる。

まずはBDEのケースと同様に、TFDTableコンポーネントのTableNameプロパティに参照するファイル名を直接設定する。次に、TDataSetProviderコンポーネントのDataSetプロパティをTSQLTableからTFDTableに変更する。【図25】

データ表示処理のソースはTClientDataSetコンポーネントを残しているため、変更は不要である。したがって、ここまでの作業でFireDACへの変更は完了となる。ただしプログラムを実行すると、BDEのケースで説明したデータ型のマッピングエラーが発生する可能性がある。発生する場合は、同じ対応が必要となる。

プログラムをコンパイルして実行すると、FireDACで従来どおりの動作が確認できる。最後に、使わなくなったdbExpressのTSQLConnectionコンポーネントとTSQLTableコンポーネントを削除して、FireDACへの移行は

図5

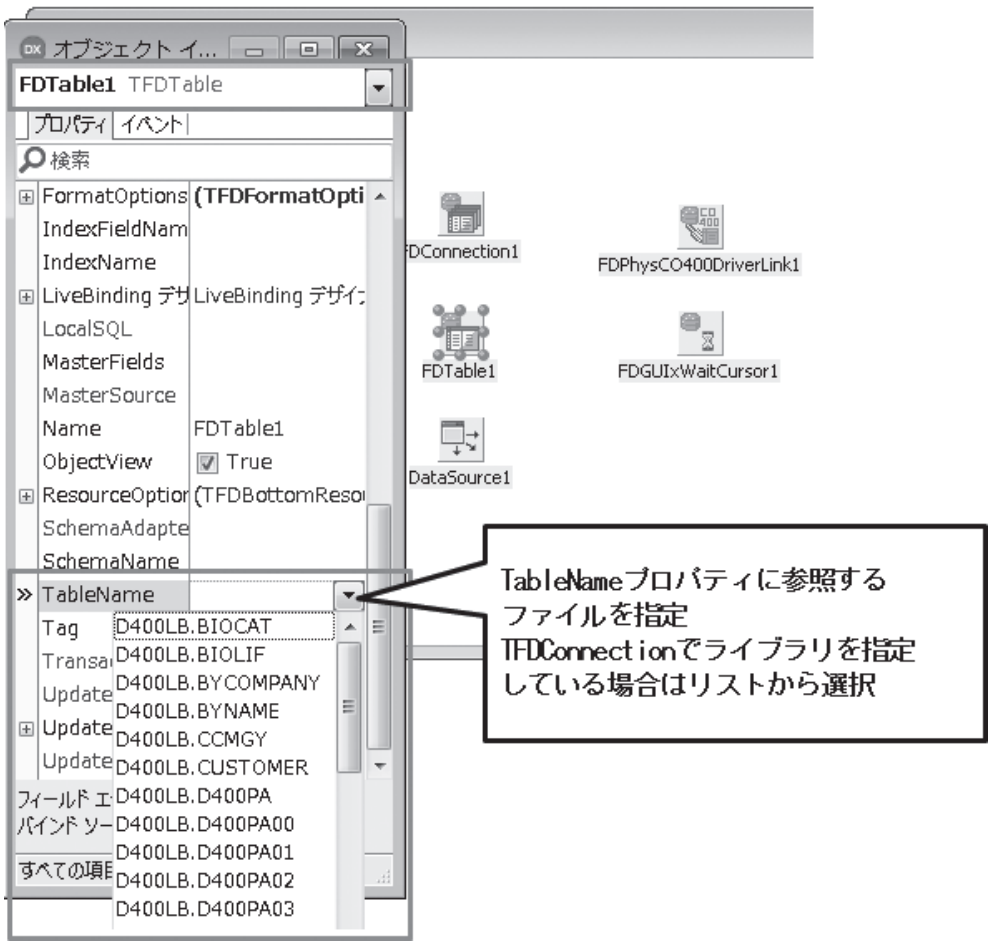

オブジェクト イ...
FDTable1 TFDTable
プロパティ
イベント
検索
FormatOptions
IndexFieldNam
IndexName
LiveBinding デザ
LocalSQL
MasterFields
MasterSource
Name
FDTable1
ObjectView
True
ResourceOptior
SchemaAdapte
SchemaName
TableName
Tag
D400LB.BIOCAT
D400LB.BIOLIF
D400LB.BYCOMPANY
D400LB.BYNAME
D400LB.CCMGY
D400LB.CUSTOMER
D400LB.D400PA
D400LB.D400PA00
D400LB.D400PA01
D400LB.D400PA02
D400LB.D400PA03
FDConnection1
FDPhysCO400DriverLink1
FDTable1
FDGUIxWaitCursor1
DataSource1
TableNameプロパティに参照する
ファイルを指定
TFDConnectionでライブラリを指定
している場合はリストから選択

図6


オブジェクト ...
DataSource1 TDataSource
プロパティ
イベント
検索
AutoEdit
True
DataSet
FDTable1
Enabled
True
LiveBinding デ
LiveBinding デザイナ
Name
DataSource1
Tag
0
FDConnection1
FDTable1
DataSource1
オブジェクト ...
DBGrid1 TDBGrid
プロパティ
イベント
検索
Constraints
(TSizeConstraints)
Ctl3D
True
Cursor
crDefault
CustomHint
DataSource
DataSource1
DefaultDrawir
True
DragCursor
crDrag
DragKind
dkDrag
DragMode
dmManual
FDConnection1
FDTable1
DataSource1

完了である。

4. まとめ

本稿ではFireDACの特徴や基本的な使用方法を確認し、BDEやdbExpressで作成されている既存プログラムからの移行ポイントを説明した。

BDEやdbExpressでプログラムを開発された経験があれば、FireDACがこれまでとほとんど同じ構成で使えることを確認いただけたと思う。既存のプログラムをFireDACへ移行する場合はいくつかのポイントがあるが、Delphi/400は非常に互換性が高いので、定型的な作業で簡単に変更できる。

本稿のノウハウを参考に、既存プログラムやこれからの新規開発でFireDACを活用していただければ幸いである。

M

図7

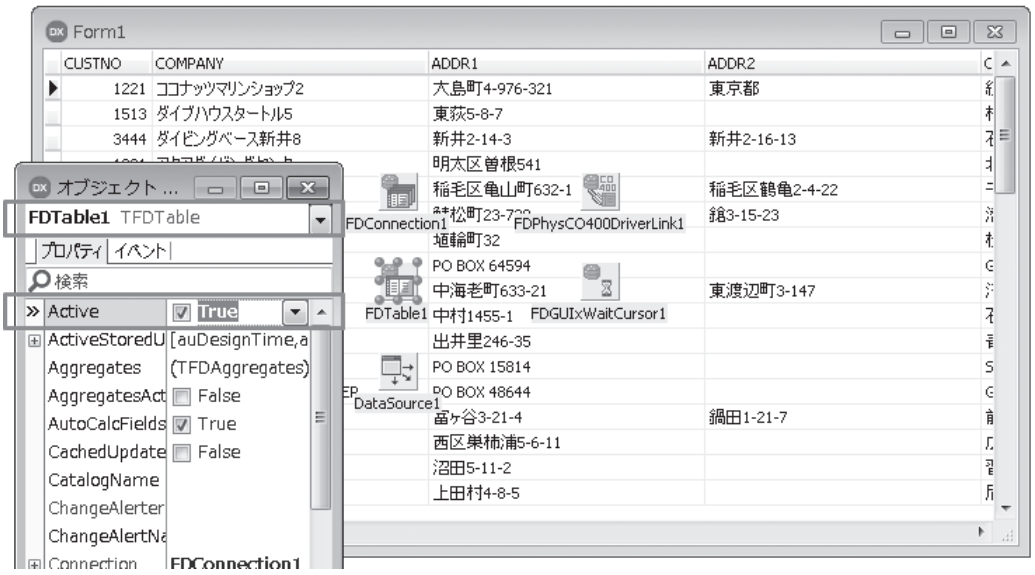


図8

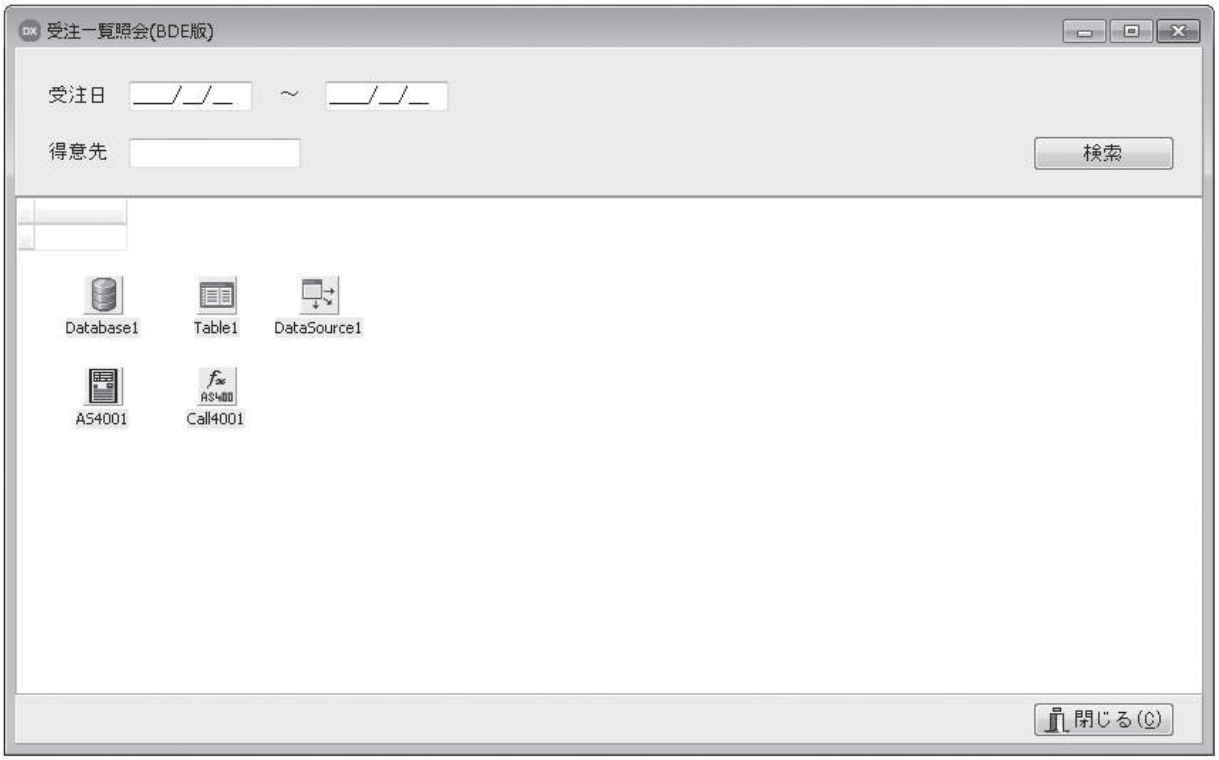
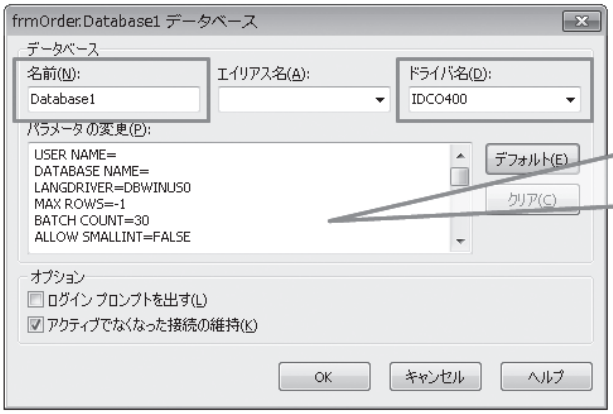


表2

BDE版照会画面　使用コンポーネント

使用コンポーネント	設定プロパティ	設定値
TDatabase	Params	【図9】参照
TTable	DatabaseName	Database1
	TableName	参照するファイル名
TDataSource	DataSet	Table1
TDBGrid	DataSource	DataSource1

図9



パラメータの設定はソースに記述（FormCreateイベント）

```
with Database1 do
begin
  Params.Values['USER NAME']     := 'USER';
  Params.Values['PASSWORD']      := 'PASSWORD';
  Params.Values['DATABASE NAME'] := 'POWER7';
  Params.Values['LIBRARY NAME']  := '*LIBL';
end;
```

図10

```
if Table1.Active then Table1.Close;
```

データ抽出前にTTableを閉じる

```
with Call4001 do
begin
  Value[0] := MaskEdit1.Text; // 受注日From
  Value[1] := MaskEdit2.Text; // 受注日To
  Value[2] := Edit1.Text;     // 得意先CD
  Value[3] := '';             // エラーCD
  Value[4] := '';             // エラーメッセージ

  Execute;

  // エラー処理
  if Value[3] <> '' then
  begin
    MaskEdit1.SetFocus;
    raise Exception.Create(Value[4]);
  end;
end;
```

データ抽出処理
TCall400コンポーネントを使用して
CLをCallし、QTEMPのワークファイル
へデータを出力

```
Table1.Open;
```

データ抽出後にTTableを開き
画面にデータを表示

図11

図12

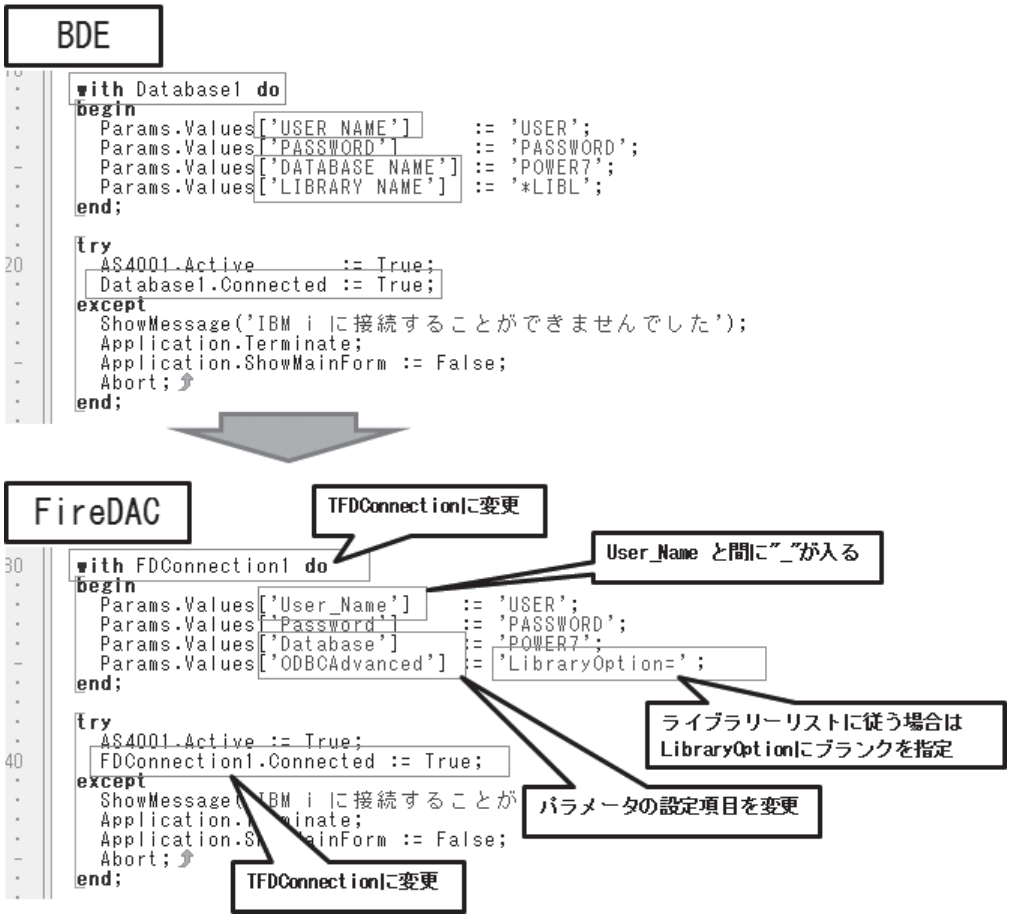

BDE
with Database1 do
begin
Params.Values['USER NAME'] := 'USER';
Params.Values['PASSWORD'] := 'PASSWORD';
Params.Values['DATABASE NAME'] := 'POWER7';
Params.Values['LIBRARY NAME'] := '*LIBL';
end;
try
AS400.Active := True;
Database1.Connected := True;
except
ShowMessage('IBM i に接続することができませんでした');
Application.Terminate;
Application.ShowMainForm := False;
Abort;
end;
FireDAC
TFDConnectionに変更
User_Name と間に"_"が入る
with FDConnection1 do
begin
Params.Values['User_Name'] := 'USER';
Params.Values['Password'] := 'PASSWORD';
Params.Values['Database'] := 'POWER7';
Params.Values['ODBCAdvanced'] := 'LibraryOption=';
end;
try
AS400.Active := True;
FDConnection1.Connected := True;
except
ShowMessage('IBM i に接続することが
Application.Terminate;
Application.ShowMainForm := False;
Abort;
end;
ライブラリーリストに従う場合は
LibraryOptionにブランクを指定
パラメータの設定項目を変更
TFDConnectionに変更

図13

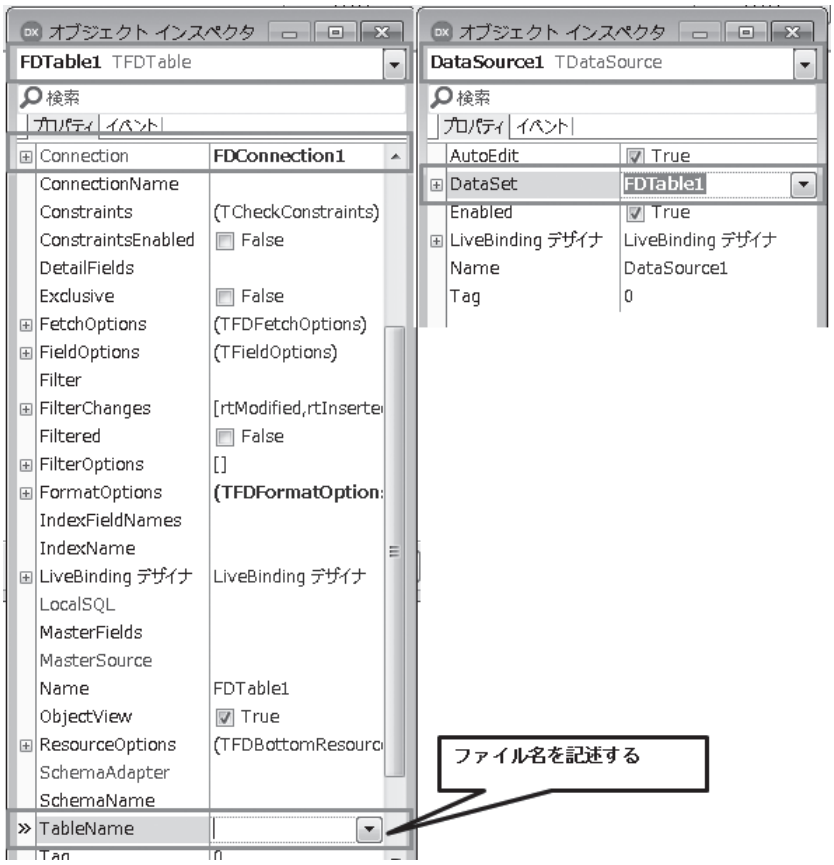

オブジェクト インスペクタ
FDTable1 TFDTable
検索
プロパティ イベント
Connection FDConnection1
ConnectionName
Constraints (TCheckConstraints)
ConstraintsEnabled False
DetailFields
Exclusive False
FetchOptions (TFDFetchOptions)
FieldOptions (TFieldOptions)
Filter
FilterChanges [rtModified,rtInserte
Filtered False
FilterOptions []
FormatOptions (TFDFormatOption
IndexFieldNames
IndexName
LiveBinding デザイナ LiveBinding デザイナ
LocalSQL
MasterFields
MasterSource
Name FDTable1
ObjectView True
ResourceOptions (TFDBottomResourc
SchemaAdapter
SchemaName
TableName
オブジェクト インスペクタ
DataSource1 TDataSource
検索
プロパティ イベント
AutoEdit True
DataSet FDTable1
Enabled True
LiveBinding デザイナ LiveBinding デザイナ
Name DataSource1
Tag 0
ファイル名を記述する

図14

BDE

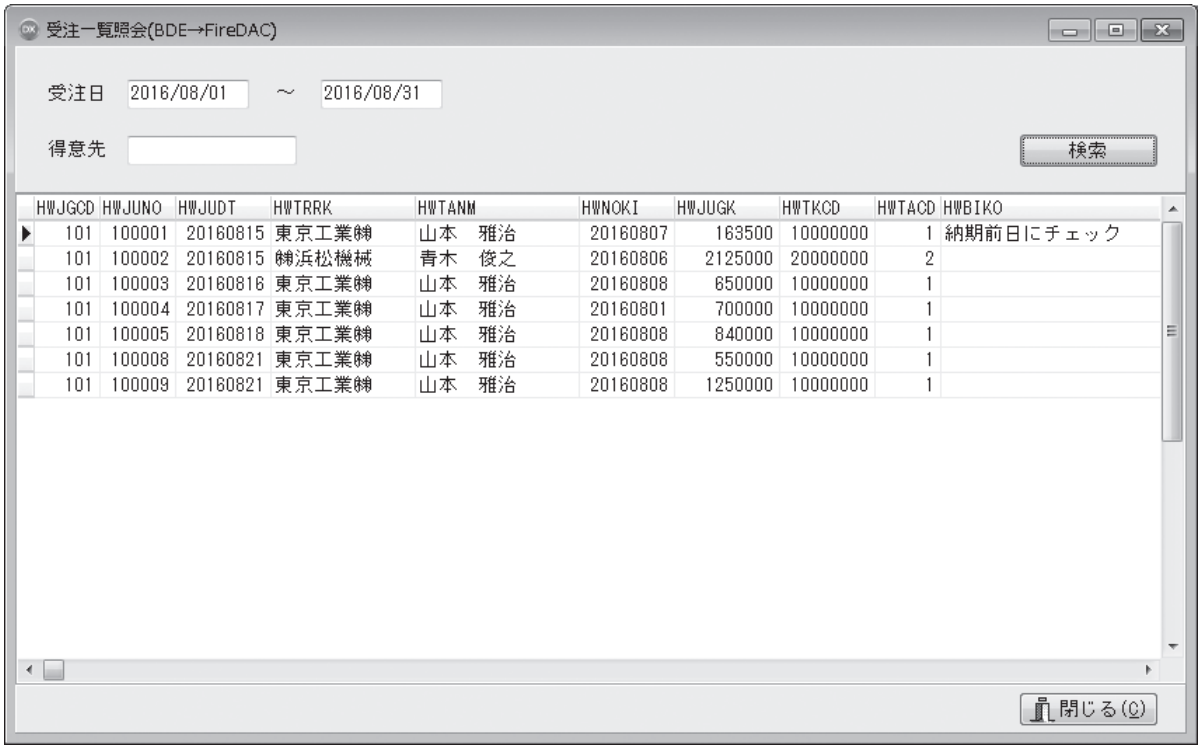
```
if Table1.Active then Table1.Close;

with Call4001 do
begin
  Value[0] := MaskEdit1.Text; // 受注日From
  Value[1] := MaskEdit2.Text; // 受注日To
  Value[2] := Edit1.Text;     // 得意先CD
  Value[3] := '';             // エラーCD
  Value[4] := '';             // エラーメッセージ

  Execute;

  // エラー処理
  if Value[3] <> '' then
  begin
    MaskEdit1.SetFocus;
    raise Exception.Create(Value[4]);
  end;
end;

Table1.Open;
```

TTableを使用している部分をTFDTableに変更

FireDAC

```
if FDTable1.Active then FDTable1.Close;

with Call4001 do
begin
  Value[0] := MaskEdit1.Text; // 受注日From
  Value[1] := MaskEdit2.Text; // 受注日To
  Value[2] := Edit1.Text;     // 得意先CD
  Value[3] := '';             // エラーCD
  Value[4] := '';             // エラーメッセージ

  Execute;

  // エラー処理
  if Value[3] <> '' then
  begin
    MaskEdit1.SetFocus;
    raise Exception.Create(Value[4]);
  end;
end;

FDTable1.Open;
```

図15



HWJGCD	HWJUNO	HWJUDT	HWTRRK	HWTANM	HWNOKI	HWJUGK	HWTKCD	HWTACD	HWBIKO
101	100001	20160815	東京工業㈱	山本　雅治	20160807	163500	10000000	1	納期前日にチェック
101	100002	20160815	㈱浜松機械	青木　俊之	20160806	2125000	20000000	2	
101	100003	20160816	東京工業㈱	山本　雅治	20160808	650000	10000000	1	
101	100004	20160817	東京工業㈱	山本　雅治	20160801	700000	10000000	1	
101	100005	20160818	東京工業㈱	山本　雅治	20160808	840000	10000000	1	
101	100008	20160821	東京工業㈱	山本　雅治	20160808	550000	10000000	1	
101	100009	20160821	東京工業㈱	山本　雅治	20160808	1250000	10000000	1	

図16

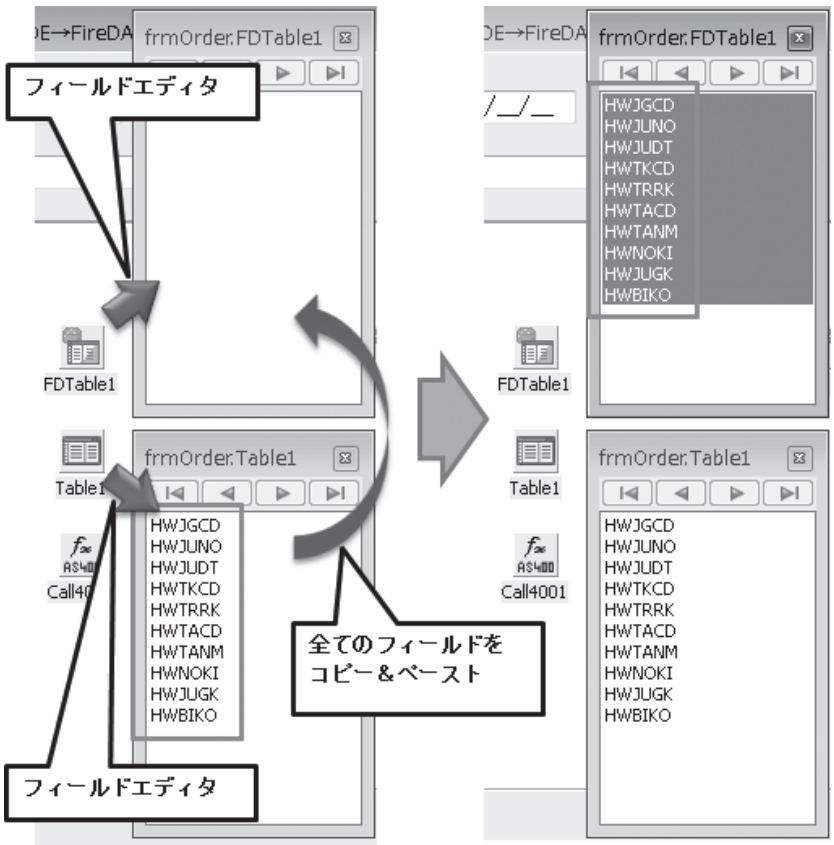

フィールドエディタ
frmOrder.FDTable1
frmOrder.Table1
FDTable1
Table1
Call4001
HWJGCD
HWJUNO
HWJUDT
HWTKCD
HWTRRK
HWTACD
HWTANM
HWNOKI
HWJUGK
HWBIKO
全てのフィールドを
コピー＆ペースト
フィールドエディタ

図17

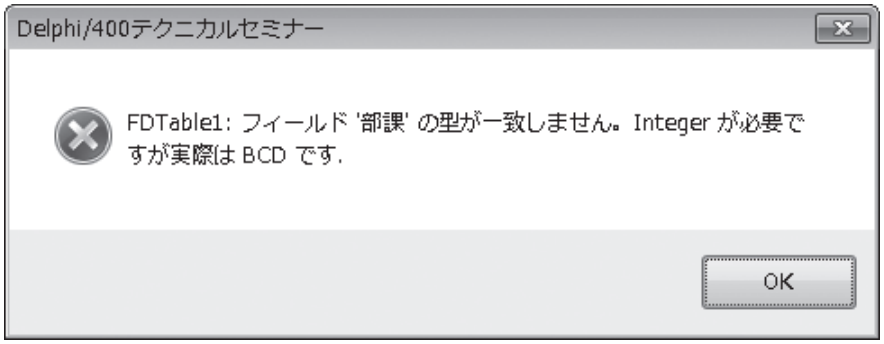

Delphi/400テクニカルセミナー
FDTable1: フィールド '部課' の型が一致しません。Integer が必要で
すが実際は BCD です.
OK

図18


FireDAC 接続エディタ - [FDConnection1]
定義
オプション
情報
SQL スクリプト
形式オプション
データ マッピング ルール
継承したルールを無視(G)
SourceDataType
TargetDataType
PrecMin
dtBCD
dtInt32

図19

部課	受注No.	受注日	得意先CD	得意先名	担当CD	営業担当者名	納期	受注金額	備考
101	100001	2016/08/15	10000000	東京工業㈱	1	山本　雅治	2016/08/07	163,500	納期前日にチェック
101	100002	2016/08/15	20000000	㈱浜松機械	2	青木　俊之	2016/08/06	2,125,000	
101	100003	2016/08/16	10000000	東京工業㈱	1	山本　雅治	2016/08/08	650,000	
101	100004	2016/08/17	10000000	東京工業㈱	1	山本　雅治	2016/08/01	700,000	
101	100005	2016/08/18	10000000	東京工業㈱	1	山本　雅治	2016/08/08	840,000	
101	100008	2016/08/21	10000000	東京工業㈱	1	山本　雅治	2016/08/08	550,000	
101	100009	2016/08/21	10000000	東京工業㈱	1	山本　雅治	2016/08/08	1,250,000	

図20

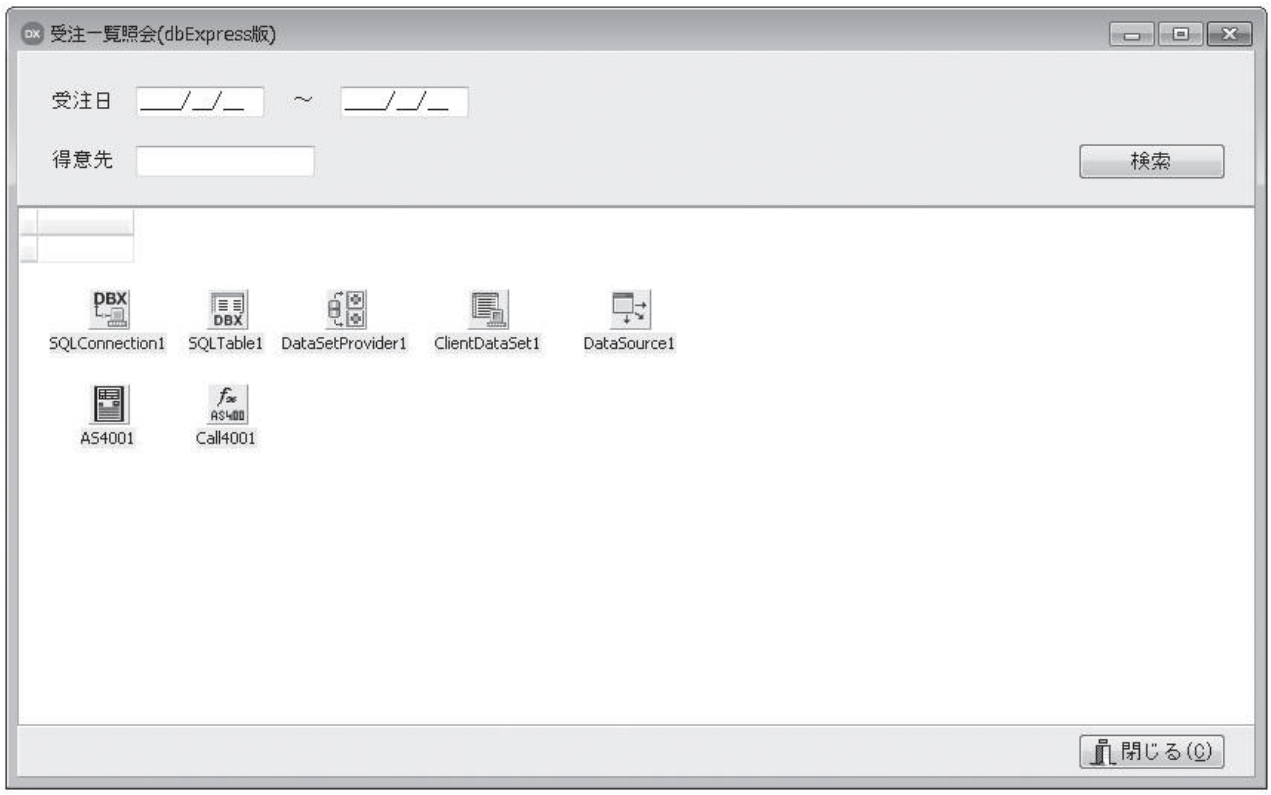


表3

dbExpress版照会画面　使用コンポーネント

使用コンポーネント	設定プロパティ	設定値
TSQLConnection	Params	【図21】参照
TSQLTable	SQLConnection	SQLConnection1
	TableName	参照するファイル名
TDataSetProvider	DataSet	SQLTable1
TClientDataSet	ProviderName	DataSetProvider1
TDataSource	DataSet	ClientDataSet1
TDBGrid	DataSource	DataSource1

図21

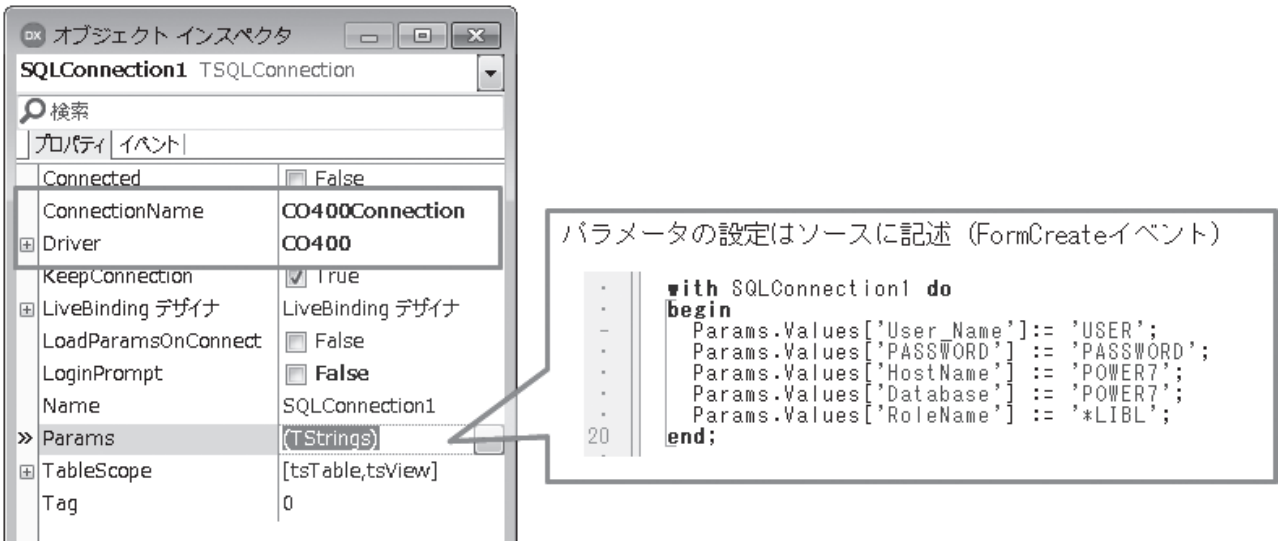


図22

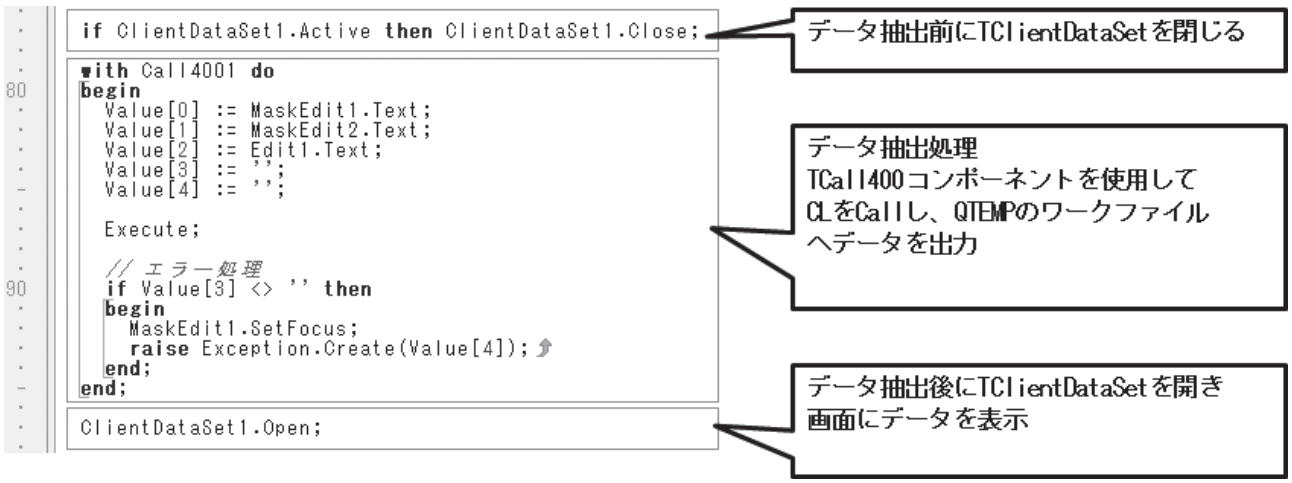


図23

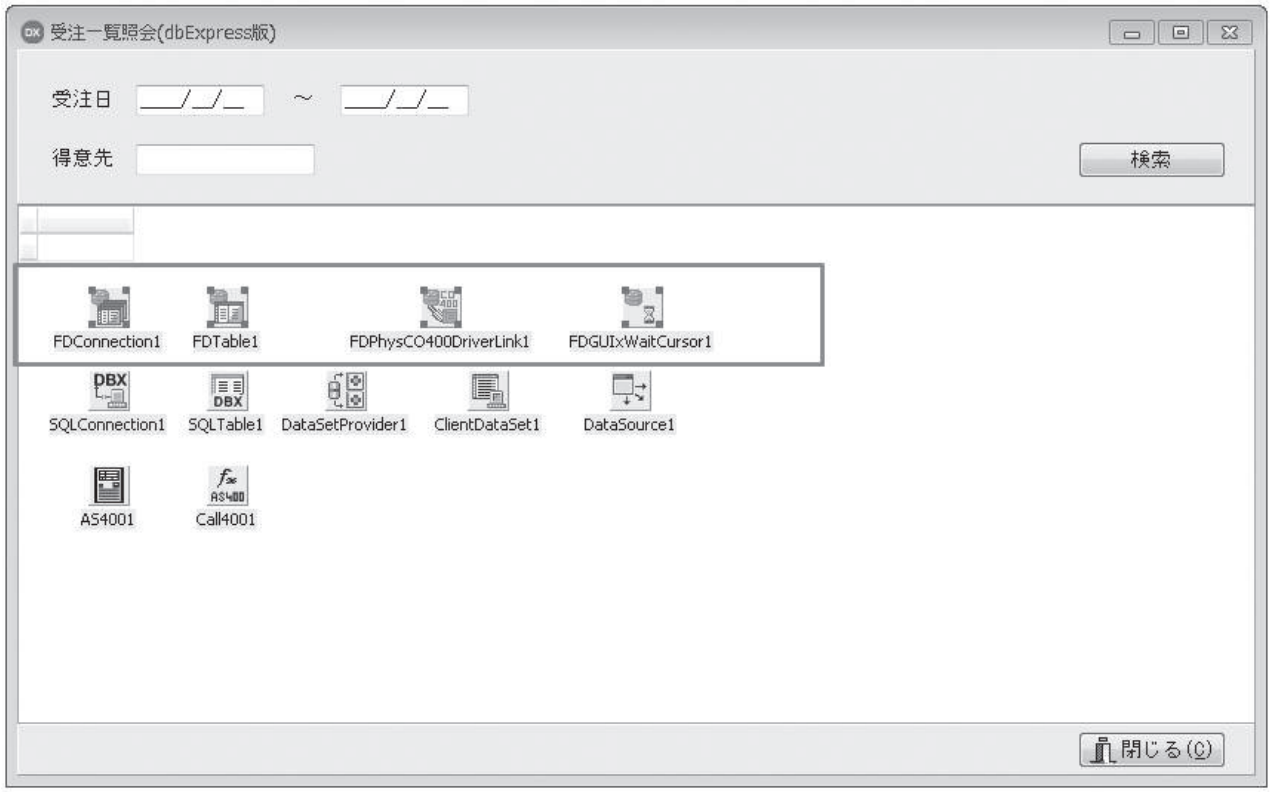


図24

dbExpress

```
with SQLConnection1 do
begin
  Params.Values['User_Name']:= 'USER';
  Params.Values['PASSWORD'] := 'PASSWORD';
  Params.Values['HostName'] := 'POWER7';
  Params.Values['Database'] := 'POWER7';
  Params.Values['RoleName'] := '*LIBL';
end;

try
  AS4001.Active := True;
  SQLConnection1.Connected := True;
except
  ShowMessage('IBM i に接続することができませんでした');
  Application.Terminate;
  Application.ShowMainForm := False;
  Abort;
end;
```

FireDAC

TFDConnectionに変更

パラメータの"HostName"は
FireDACでは不要

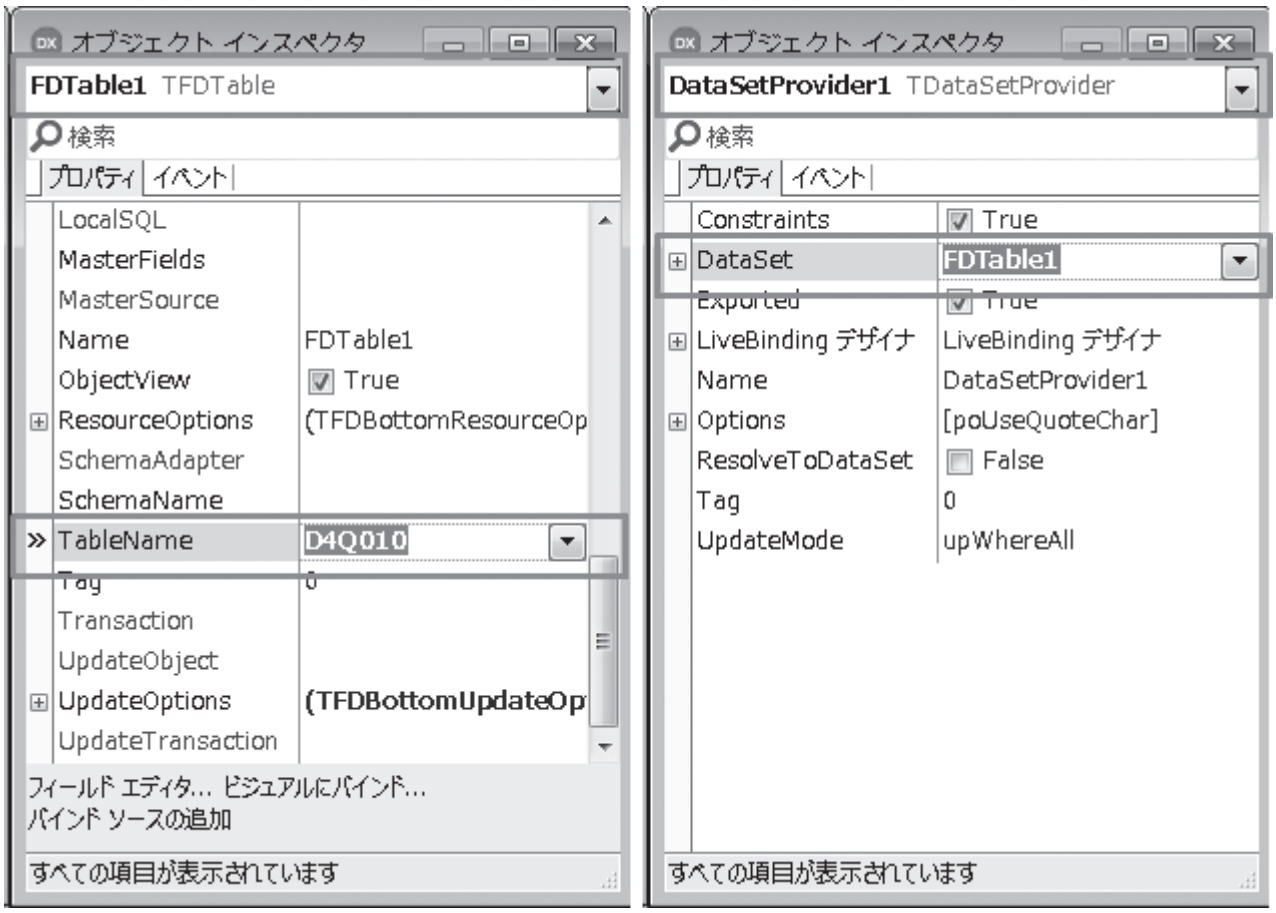
```
with FDConnection1 do
begin
  Params.Values['User_Name']     := 'USER';
  Params.Values['Password']      := 'PASSWORD';
  Params.Values['Database']      := 'POWER7';
  Params.Values['ODBCAdvanced'] := 'LibraryOption=''''';
end;

try
  AS4001.Active := True;
  FDConnection1.Connected := True;
except
  ShowMessage('IBM i に接続することが
  Application.Terminate;
  Application.ShowMainForm := False;
  Abort;
end;
```

ライブラリーリストに従う場合は
LibraryOptionにブランクを指定

パラメータの設定項目を変更

TFDConnectionに変更

図25


オブジェクト インスペクタ
FDTable1 TFDTable
検索
プロパティ
イベント
LocalSQL
MasterFields
MasterSource
Name
FDTable1
ObjectView
True
ResourceOptions
(TFDBottomResourceOp
SchemaAdapter
SchemaName
TableName
D4Q010
Tag
0
Transaction
UpdateObject
UpdateOptions
(TFDBottomUpdateOp
UpdateTransaction
フィールド エディタ...
ビジュアルにバインド...
バインド ソースの追加
すべての項目が表示されています
オブジェクト インスペクタ
DataSetProvider1 TDataSetProvider
検索
プロパティ
イベント
Constraints
True
DataSet
FDTable1
LiveBinding デザイナ
LiveBinding デザイナ
Name
DataSetProvider1
Options
[poUseQuoteChar]
ResolveToDataSet
False
Tag
0
UpdateMode
upWhereAll
すべての項目が表示されています

尾崎 浩司

株式会社ミガロ.

RAD事業部 営業・営業推進課

[Delphi/400]

Delphi/400最新プログラム文法の活用法

略歴
1973年8月16日生まれ
1996年 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. 製品の営業を担当。これまでのシステム開発経験を活かして、IBM iをご利用のお客様に対して、GUI化、Web化、モバイル化などを提案している。

1.はじめに

Delphi/400は、ビジュアルプログラミングと呼ばれる開発手法でアプリケーションを作成する。ビジュアルプログラミングとは、コンポーネントをフォームに配置し、プロパティを定義したうえで、必要に応じたユーザーのアクション（マウスをクリックする、キーボードで入力するといった操作）に対し、イベントハンドラと呼ばれるプログラムをコーディングしていく手法である。

このDelphi/400のコーディングに使用するのが、Object Pascalである。Object Pascalは、もともと教育用として開発されたPascal言語をオブジェクト指向プログラミングが行えるように拡張したもので、シンプルな文法やデータ型の厳格な型チェックを採用しているのが特徴である。

Delphi/400は、これまでのバージョンアップでさまざまな機能拡張を実施しているが、Ver.2009以降では、文法についても多くの新しい記述方法が追加されている。

本稿ではVer.2009以降に追加された文法について、具体例とともに説明する。

2.文法の高度な機能（Delphi/400 Ver.2009以降）

Delphi/400 Ver.2009では、それ以前のバージョンまでのShift-JISベースであった文字コード体系がUnicodeベースに大きく変更された。

このバージョンでは文字コード体系の変更とともに、ジェネリクスならびに無名メソッドという大きな文法の進化も見られる。以下に、この2つの概要ならびに活用例を説明する。

2-1. ジェネリクスとは

ジェネリクスとは一言でいうと、特定の型に依存しない実装を行うプログラミングスタイルのことである。具体例として、たとえばInteger型の変数A、BとChangeメソッドをもつTIntChangeクラスを考えてみる。【図1】

Changeメソッドは、変数AとBの値をひっくり返すだけの簡単な処理である。このクラスを使用するプログラムの実装例は、【図2】のとおりである。

このプログラムを実行し、ボタンをクリックすると、初期セットされたA＝100、B＝200の値がひっくり返り、画面上には結果として、A＝200、B＝100が表示される。

ここでは変数A、BにInteger型の整数値を使用したが、もしこれをDouble型やString型にしたい場合、それぞれのデータ型用のクラスを追加する。【図3】

【図1】と【図3】を比べると、データ型が異なる以外はまったく同じ処理であるとわかる。このように処理自体は変わらないのに、データ型が異なるためにそれぞれのクラスを作成するとなると、あらゆるデータ型への対応が必要になる。

こうしたケースで便利なのが、ジェネリクスである。【図1】のプログラムを

図1

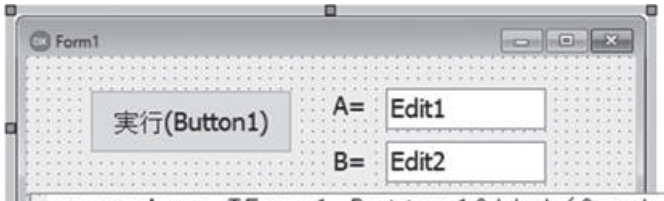
【宣言部】

```
type
  TIntChange = class
    A: Integer;
    B: Integer;
    procedure Change;
  end;
```

【宣言部】

```
{ TIntChange }

procedure TIntChange.Change;
var
  Temp: Integer;
begin
  //AとBの値をひっくり返す
  Temp := B;
  B := A;
  A := Temp;
end;
```

図2




```
procedure TForm1.Button1Click(Sender: TObject);
var
  obj: TIntChange;
begin
  //オブジェクト生成
  obj := TIntChange.Create;

  //値をセット
  obj.A := 100;
  obj.B := 200;

  //値の入れ替えを実施
  obj.Change;

  //結果を出力
  Edit1.Text := IntToStr(obj.A);  // <--- A=200
  Edit2.Text := IntToStr(obj.B);  // <--- B=100

  //オブジェクト破棄
  obj.Free;
end;
```

元にジェネリクスを使用したのが、【図4】である。

宣言部を見ると“＜T＞”となっているが、これが仮のデータ型を表しており、Tという名前で宣言されている（このシンボルTは慣例としてよく利用されるが、シンボルとして有効な名称であれば制限はない）。

このようにクラス上では仮のデータ型で宣言や実装を行い、クラスを使用するプログラム側で使用したいときに、実際のデータ型を指定できる。【図5】

この仕組みを使用すれば、どんなデータ型で処理が必要となっても同じクラスが利用できる。これがジェネリクスと呼ばれるものである。

2-2. ジェネリクス活用例

以下に、活用方法について考察する。最もよく使用されるのがコレクションである。

コレクションとは、複数の要素の集まりのことである。Delphi/400でおそらく一番よく使用されているコレクションは、TStringListである。TStringListは、文字列のリストを扱うクラスである。このコレクションを使用すると、文字列を動的配列として扱える。【図6】

ここではリストに対してAddメソッドを使用することで、文字列をリストに追加している。追加されたリストは、配列と同じように要素番号を指定することで、各要素値が取得できることがわかる。

ジェネリクスコレクションクラスであるTList＜T＞を使用すると、これと同じようなことが任意のデータ型で行える。ジェネリクスのコレクションを使用するには、uses節にGenerics.Collectionsを追加すればよい。このユニットには、TList＜T＞をはじめとするいくつかのジェネリクスコレクションクラスが定義されているので、これらを使用できる。

TList＜T＞を使用したリストのプログラムは、【図7】のとおりである。

このプログラムではTList＜Integer＞と定義しているので、Integer型の値をリストとして扱える。【図6】と比較すれば明らかだが、TStringListの場合とまったく同じ手法で、数値に対する動的配列が実現できる。Addメソッドで、Integer型の値を直接リストにセットしている。

便利なのは、Addメソッドには Integer型の値以外はセットできないことである。ジェネリクスで実際の型が決定すると、その型のみが使用できる。

さらに配列に対してリストを使用するメリットの1つとして、ソートが簡単に行える点がある。【図7】で、リストに値を追加した後に、「iList.Sort;」と1行追加すると、整数値の昇順にリストが並び替わる。ソートを配列で実現しようとすると、ロジックを作成せねばならないので、こうした場合にリストを使用するメリットがある。

もう1つの例は、TDictionary＜TKey,TValue＞である。このクラスはキーと値のセットをコレクションとして扱う。こちらも実装例を、【図8】で説明する。

このプログラムでは、キーにはString型を、値にはTCustomer型（Record型）を指定している。TDictionaryの場合も、コレクションの追加はAddメソッドで可能である。要素へのアクセスは、キー値を指定すればよい。

またこのコレクションクラスは、データの検索も容易である。たとえばTryGetValueメソッドを使用すると、存在しないキーを指定した場合、結果がFalseとなるので、入力妥当性チェックにも活用できる。リストの場合と違い、ディクショナリはキーを指定して任意のデータにアクセスできるので、応用範囲が広い。

ここまで、TList＜T＞ならびにTDictionary＜TKey,TValue＞を説明したが、これらのコレクションは使用時のデータ型に、値型を想定したものである。クラス型オブジェクトを想定したTObjectList＜T: class＞やTObjectDictionary＜TKey,TValue＞も用意されているので、用途に合わせて使い分けると効果的である。

2-3. 無名メソッドとは

無名メソッドとはその名のとおり、名前がついていないprocedureやfunctionのことである。通常変数には、値をセットできるが、無名メソッドを使用すると、手続きや関数自体を変数にセットできる。無名メソッドの簡単な使用例を、【図9】で説明する。

まず宣言部を確認する。ここでは、1つのString型引数をもつ手続き型の無名メソッドが保持できるデータ型として、TStrProc型を宣言している。

次に実装部を見ると、変数宣言部分（var）で、TStrProc型の変数pProcを宣言しているのがわかる。そして、この無名メソッド変数pProcに対して、名前のない手続き（procedure）を代入している。こう記述することで、通常の値を変数に代入するのとまったく同じ記述方法により、手続きや関数を変数に代入できる。

なお、無名メソッド変数に手続きや関数を代入した時点では、まだ無名メソッドは実行されない。実際に無名メソッドが実行されるのは、「pProc（‘テスト’）;」のように変数を使用したときである。

このように、無名メソッドは変数に代入できるのだが、それだけではなく、手続きや関数の引数に無名メソッドを渡すこともできる。具体例を、【図10】で説明する。

ここでの宣言部では、Integer型の引数を2つもつ関数型の無名メソッドが保持できるデータ型として、TCalcFunc型を宣言している。

実装部には、Calculate手続きを作成しているが、このサブルーチンは、2つの整数の引数とともにTCalcFunc型の引数を使用しているのがわかる。

つまりこのサブルーチンは、呼び出し側で定義された無名メソッドを受け取って処理を実行する。Button2のOnClickイベントでは、Calculate手続きを2回呼び出している。それぞれ引数として、異なる2つの値とともに、異なる無名メソッドを渡している。

このプログラムを実行して、Button2をクリックすると、メッセージボックスに計算式の異なる2つの処理結果が表示される。【図11】

このように無名メソッドを使用すると、通常の変数等と同じように手続きや関数を引数として渡せる。

2-4. 無名メソッド活用例

次に、名前をもたない無名メソッドについて、サブルーチンに対して無名メソッドを渡す仕組みの活用例を説明する。

図3

【宣言部】

```
type
  // Double型用
  TFloatChange = class
    A: Double;
    B: Double;
    procedure Change;
  end;

  // String型用
  TStringChange = class
    A: String;
    B: String;
    procedure Change;
  end;
```

【実装部】

```
{ TFloatChange }

procedure TFloatChange.Change;
var
  Temp: Double;
begin
  //AとBの値をひっくり返す
  Temp := B;
  B := A;
  A := Temp;
end;

{ TStringChange }

procedure TStringChange.Change;
var
  Temp: String;
begin
  //AとBの値をひっくり返す
  Temp := B;
  B := A;
  A := Temp;
end;
```

図4

【宣言部】

```
type
  // ジェネリッククラス
  TChange<T> = class
    A: T;
    B: T;
    procedure Change;
  end;
```

【実装部】

```
{ TChange<T> }

procedure TChange<T>.Change;
var
  Temp: T;
begin
  //AとBの値をひっくり返す
  Temp := B;
  B := A;
  A := Temp;
end;
```

IBM i（AS/400）をはじめ、各種データベースに対して更新処理を行うような場合、たとえばdbExpress接続では、【図12】のような処理を記述することが多い。

このプログラムのように、データベースへの更新処理は大きく次の3つから構成される。

①トランザクションの開始
②データの登録／変更／削除等の更新処理
③トランザクションのコミット（②でエラーの場合ロールバック）

たとえば受注と売上の各更新処理がある場合、一般にそれぞれの更新処理で①②③を記述する。しかしデータベースへの更新内容が異なっても、②が異なるだけで、①と③は共通の処理となる。【図13】

この場合に役立つのが、無名メソッドである。②の部分を無名メソッドとして、データベース更新処理の共通サブルーチンの引数とすればよい。【図12】のプログラムを修正し、無名メソッドを使用する例を、【図14】で説明する

このプログラム例では、引数のデータ型をTProc型としているが、これは引数をもたない手続き型の無名メソッド用にあらかじめ用意されたデータ型なので、これを使用すれば、とくに型の宣言をせずに無名メソッドが使用できる。

【図14】で定義したDataUpdateメソッドを呼び出すプログラムは、【図15】のとおりである。ここでは更新処理自体の無名メソッドを引数にセットして、DataUpdateメソッドを呼び出しているのがわかる。

以上、無名メソッドの使用例として、データベースの更新処理を説明したが、ほかによく使用される無名メソッドの活用方法として、TThread.CreateAnonymousThreadと無名メソッドを使用したスレッド（並列）処理がある。

これについては、2015年版ミガロ.テクニカルレポートにある『マルチスレッドを使用したレスポンスタイム向上』で詳しく説明しているので、そちらを参照してほしい。

3.最新文法活用TIPS（Delphi/400 Ver.2010以降）

ここからは、Delphi/400でプログラミングする際に便利な2つの文法活用TIPSを説明する。どちらもプログラム開発で非常に有用なので、ぜひ参考にしてほしい。

3-1. レコードヘルパ、クラスヘルパによる既存機能の拡張（Delphi/400 Ver.2010以降）

Delphi/400は、データ型の取り扱いが厳格である。

たとえば、String型とInteger型とで相互代入はできない。Integer型の変数iに、演算結果として整数値123がセットされていると、「ShowMessage（i）;」という手続きは、コンパイルエラーとなる。

これは、ShowMessage手続きの引数がString型を要求しているにもかかわらず、Integer型の変数をセットしているから発生するエラーである。では、プログラムのなかで演算された結果をメッセージボックスに表示するには、どうすればよいだろうか。

この場合、データ変換関数を使用するのが一般的である。先の例では、「ShowMessage（IntToStr（i））;」と記述すれば、演算結果をメッセージボックスに出力できる。このようにデータ変換の機能がサブルーチンとして定義されているので、それを利用する。しかしデータ変換を行うのに、その都度サブルーチンを使用するのは、いささかプログラミングが面倒である。

そこでDelphi/400 Ver.2010以降には、レコードヘルパという機能が用意されている。これは特定のレコードに対して、機能拡張をサポートする。通常、既存機能の拡張というと、オブジェクトクラスに対して「継承」を利用するのが一般的だが、レコードヘルパを使用すると、String型やInteger型といった組み込みデータ型に対しても機能を拡張できる。

とくにDelphi/400 Ver.XE5以降のObject Pascalでは、TIntegerHelperやTStringHelperといった定義済みのレコードヘルパクラスが用意されているので、既存機能の拡張を意識することなく、そのまま使用できる。

たとえばInteger型のレコードヘルパであるTIntegerHelperを使用すると、変数iに対して、「ShowMessage（i.ToString）;」のように記述できる。このようにレコードヘルパを使用すると、変数などに対して直接メソッドが記述できるので、コードの見通しがよくなる。

なお、このようにすぐに使用できるレコードヘルパは、SysUtilsユニットに定義されている。Delphiの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オンラインヘルプ（DocWiki）のSysUtilsユニットページを参照し、「レコードヘルパ」でページ検索すれば、定義済みのレコードヘルパを確認できる。【図16】

このレコードヘルパは、独自の定義も可能である。

IBM i（AS/400）を活用するアプリケーションでは、日付値を示すデータベースのフィールドとして数値8桁を定義することが多い。しかしDelphi/400では、日付値はTDate型を使用するのが一般的である。そこで以下に、TDate型の日付値をInteger型に変換するレコードヘルパの作成手順を説明する。

まず、宣言部にTDate型のレコードヘルパクラスと、そのなかに機能となるメソッド（ToIntegerメソッド）を宣言する【図17】。宣言が完了したら、［Ctrl］＋［Shift］＋［C］を押下し、実装部のテンプレートを作成のうえ、メソッド内に実装を記述する。【図18】

【図18】の実装例を見ると、Selfというキーワードがあるのがわかる。このSelfには、メソッドが実行される際のTDate型の日付値がセットされる。ここではSelfで指定された日付値に対し、FormatDateTime関数を使用して、いったん8桁の文字列に変換したのち、StrToInt関数で整数値に変換している。

レコードヘルパが完成すれば、使用方法は簡単である。たとえばフォーム上にあるTDateTimePicker（日付入力コンポーネント）にセットされたTDate型の値を、Integer型の値として取得するのは、【図19】のようなコードで記述できる。

TDate型の値に対し、直接ToInteger

図5

```
procedure TForm1.Button1Click(Sender: TObject);
var
  IntObj: TChange<Integer>;
  StrObj: TChange<String>;
begin
  //オブジェクト生成
  IntObj := TChange<Integer>.Create;
  StrObj := TChange<String>.Create;

  //値をセット
  IntObj.A := 100;
  IntObj.B := 200;

  StrObj.A := 'ABC';
  StrObj.B := 'DEF';

  //値の入れ替えを実施
  IntObj.Change;
  StrObj.Change;

  //結果を出力
  Edit1.Text := IntToStr(IntObj.A);  // <--- A=200
  Edit2.Text := IntToStr(IntObj.B);  // <--- B=100

  Edit3.Text := StrObj.A;            // <--- A='DEF'
  Edit4.Text := StrObj.B;            // <--- B='ABC'

  //オブジェクト破棄
  IntObj.Free;
  StrObj.Free;
end;
```

図6

```
procedure TForm1.Button1Click(Sender: TObject);
var
  sList: TStringList;   //文字列リスト
  i: Integer;
begin
  //文字列リストを生成
  sList := TStringList.Create;

  //文字列のリストを追加
  sList.Add('文字列１');
  sList.Add('文字列２');
  sList.Add('文字列３');

  //配列の要素にアクセス
  for i := 0 to sList.Count - 1 do
    ShowMessage(sList[i]);

  //リストの解放
  sList.Free;
end;
```

メソッドを記述してInteger型の値に変換できている。もちろん同様のことは、TDate型の値をInteger型に変換するためのデータ変換関数（function）を作成し、その関数を使用しても実装できるが、レコードヘルパを使用したコードのほうが読みやすいのは一目瞭然である。

ここで説明した例は、TDate型をInteger型に変換するレコードヘルパだが、もちろんInteger型をTDate型に変換するレコードヘルパも作成可能である。その場合、【図20】のような処理が考えられる。

ただし、レコードヘルパは1つのデータ型に対して1つしか使用できない点に注意が必要である。Delphi/400 Ver.XE5以降には、あらかじめ定義済みのTIntegerHelperが存在するので、【図20】の宣言を参照するプログラムでは、TIntegerHelperに定義されたメソッドが使用できなくなる。

すでに存在するデータ型のレコードヘルパと共用したい場合は、Integer型に対するエリアス（別名）を定義すればよい【図21】。ここでは、Integer型のエリアスとしてTDateInt型を定義している。それにより、独自に定義した【図21】のレコードヘルパを使用する場合には、TDateInt型でキャストすればよい。

たとえば、日付整数値が格納されたInteger型の変数iに対しては、「TDateInt（i）.ToDate」のように記述できる。【図22】

説明したのは組み込みデータ型に対するレコードヘルパだが、クラスに対してもクラスヘルパが使用可能である。クラスヘルパを使用すると、たとえば標準のコンポーネントに対して簡単に機能を追加できる。つまり独自の継承コンポーネントを作成することなく、機能拡張できるわけだ。

TEditの親クラスであるTCustomEditに対して、データ型の変換機能を実装した例を、【図23】で説明する。

このクラスヘルパを参照するプログラムでは、TCustomEditを継承したTEdit等の入出力コンポーネントに対し、直接TDate型やInteger型で値の取得ならびに代入が可能になる。【図24】

このようにレコードヘルパやクラスヘルパを作成すると、元のレコードやクラスに一切手を加えることなく、新しい機能が追加できるので、汎用ユニットとして定義できる。

3-2. ランタイムライブラリ（RTL）を活用したプログラム作成法

Delphi/400でプログラムを記述する際、前述したIntToStr関数などのデータ変換関数を使用することが多い。では、なぜ作成するプログラムで、IntToStr関数が使用できるのだろうか。

VCLとFireMonkeyのそれぞれで、新規プロジェクトを作成し、作成直後のForm1ユニット（Unit1.pas）を見ると、どちらもほぼ同じ構成であるのがわかる。【図25】【図26】

構成のなかで異なるのは、uses節の部分である。Object Pascalでuses節は、プログラムの実行に必要なほかの参照ユニットを表している。VCLかFireMonkeyかで、使用するビジュアルコンポーネントのフレームワークが異なるので内容も違っているのだが、よく見るとSystem.SysUtils、System.Variants、System.Classesの各ユニットはどちらのプロジェクトにも含まれているのがわかる。

冒頭のIntToStr関数は、System.SysUtilsユニットに定義された関数である。つまり、IntToStr関数が使用できるのは、ユニット参照されているからである。このIntToStr関数のようなアプリケーション開発で一般に使用されるサブルーチンは、ライブラリとして提供されており、Delphiランタイムライブラリ（RTL）と呼ばれている。

このRTLには多彩な機能が実装されており、プログラムで多様な機能を実現できる。RTLの多くはSystemユニットスコープに定義されており、DocWikiを参照しても多数のユニットが用意されている（http://docwiki.embarcadero.com/Libraries/Seattle/ja/System）。【図27】

このなかで、知っておくと役立つRTLを以下に説明する。

（1）System.IOUtils（Delphi/400Ver.2010以降）

System.IOUtilsは、Delphi/400 Ver.2010以降に追加されたRTLである。以前はディレクトリやファイル操作のプログラミングが少し面倒であったが、このユニットが追加されたことで扱いが簡単になった。

まず、TDirectoryクラスについて説明しよう。TDirectoryはディレクトリを操作するクラスである。たとえば、このクラスにはクラスメソッドDeleteが用意されており、これを使用すると特定フォルダを簡単に削除できる。【図28】

System.IOUtilsは標準でuses節に含まれていないので、個別に追加する。こうすれば、あとはクラスメソッドを呼び出すだけで使用できる。

このメソッドが便利な点は、フォルダ内にサブフォルダやファイルが存在していたとしても、一括削除できることだ。Delphi/400 Ver.2009以前の場合、同じ処理を実現するのに次のようなサブルーチンを作成する必要があった。

［フォルダ削除サブルーチンの処理ロジック］

①削除しようとするフォルダ内のすべてのファイルおよびフォルダを検索する
②ファイルならばDeleteFileを用いて削除し、フォルダならば再帰的に自身の関数処理を呼び出す
③フォルダの中身が空になったところで、RemoveDirectoryを用いてフォルダを削除する

TDirectoryクラスの追加により、簡単にフォルダ削除ができるようになった。ほかにもフォルダのコピー（TDirectory.Copy（）メソッド）や移動（TDirectory.Move（）メソッド）も用意されている。

次に、フォルダ内に含まれるファイルを一覧取得する処理を考えよう。これもDelphi/400 Ver.2009以前では、FindFirst関数やFindNext関数を使用しながらファイル名を取得し、サブフォルダについては、再帰処理を行う必要があった。しかしSystem.IOUtilsを使用すると、TDirectory.GetFilesメソッドで容易に取得できる。【図29】

TDirectory.GetFilesメソッドの引数に検索オプション（soAllDirectories）を付与するだけで、サブフォルダまで含めた一括検索ができる。

また【図29】のソースでは、for in

図7

```
uses Generics.Collections;

procedure TForm1.Button1Click(Sender: TObject); ...

procedure TForm1.Button2Click(Sender: TObject);
var
  iList: TList<Integer>; //ジェネリクスリスト
  i: Integer;
begin
  //整数値用リストを作成
  iList := TList<Integer>.Create;

  //整数値のリストを追加
  iList.Add(234);
  iList.Add(567);
  iList.Add(111);

  //配列の要素にアクセス
  for i := 0 to iList.Count - 1 do
    ShowMessage(IntToStr(iList[i]));

  //リストの解放
  iList.Free;
end;
```

図8

【宣言部】

```
type
  //顧客レコード
  TCustomer = record
    sName: String;  //顧客名
    sAddr: String;  //住所
    sTel : String;  //電話番号
  end;
```

【実装部】

```
uses Generics.Collections;

procedure TForm1.Button3Click(Sender: TObject);
var
  sDict: TDictionary<String, TCustomer>;
  rCust: TCustomer;
begin
  //ディクショナリを作成
  sDict := TDictionary<String, TCustomer>.Create;

  //ディクショナリに追加
  rCust.sName := '株式会社ミガロ.';
  rCust.sAddr := '大阪市浪速区湊町';
  rCust.sTel  := '06-1234-5678';
  sDict.Add('00001', rCust);

  rCust.sName := 'エンバカデロ';
  rCust.sAddr := '東京都文京区';
  rCust.sTel  := '03-1111-2222';
  sDict.Add('00002', rCust);

  rCust.sName := '日本アイ・ビー・エム株式会社';
  rCust.sAddr := '東京都中央区日本橋';
  rCust.sTel  := '03-3333-4444';
  sDict.Add('00003', rCust);

  //キーを指定してデータにアクセス
  with sDict['00002'] do
  begin
    ShowMessage(sName + ' ' + sAddr + ' ' + sTel);
  end;

  //データの検索
  if sDict.TryGetValue(Edit1.Text, rCust) then
    ShowMessage(rCust.sName)
  else
    ShowMessage('顧客コードが存在しません');

  //ディクショナリの解放
  sDict.Free;
end;
```

do ループを使用している点にも注目してほしい。従来からのカウンタ変数を使用した for ループだけでなく、このような配列などを使用した for ループ処理も記述できる。

ほかにもパス名、フォルダ名、ファイル名を操作する TPath クラスや、ファイルを操作する TFile クラスが用意されている。Delphi/400 Ver.2009 以前では、ファイルをコピーする関数が用意されておらず、Win32API を使用する必要があったが、TFile.Copy（）メソッドを使用すれば、API を意識せず簡単に実装できる。

（2）System.RegularExpressions（Delphi/400 Ver.XE 以降）

次に説明する System.RegularExpressions は、Delphi/400 Ver.XE 以降で使用可能な RTL で、いわゆる正規表現を実現する。

正規表現とは、文字列の集合を 1 つの文字列で表現する方法で、たとえば郵便番号やメールアドレスなど、特定の文字列パターンで表せるものをチェックするのに利用することが多い。これを使用したプログラムの例を、【図 30】で説明する。

ここでは TRegEx クラスの IsMatch メソッドを使用すると、文字列が指定された正規表現とマッチするかを確認できる。OnChange イベントなどで比較すると、入力途中の整合性チェックに活用できる。

RTL はほかにもいろいろあるが、知っていると便利なユニットを以下にいくつか説明する。

System.StrUtils は文字列処理関数が含まれており、たとえば LeftStr、MidStr、RightStr 関数を使用すると、Copy 関数を使わなくても、任意の位置の文字列を容易に取得できる。

System.DateUtils は、日付処理関数が含まれている。月末日を取得するのに、従来は翌月 1 日の日付 -1 という取得方法が一般的であったが、EndOfAMonth 関数を使用すると容易に取得できる。

System.Math は数値演算関数が含まれており、たとえば四捨五入は SimpleRoundTo 関数で容易に実行できる。

さらに Delphi/400 Ver.XE3 では、ZIP ファイルを扱うための System.Zip が、Delphi/400 Ver.XE7 では、JSON 文字列を扱うための System.JSON やインターネットエンコード、デコード処理を行うための System.NetEncoding が追加されており、バージョンアップのたびに便利な RTL が拡充されている。

4.まとめ

本稿では、Delphi/400 のコーディングで使用される Object Pascal の新しい文法に関するテクニックを取り上げて説明した。

Delphi/400 のコーディングに普段から使用している Object Pascal だが、本稿執筆に際してあらためて文法を調べてみると、Delphi/400 Ver.2009 以降で文法が大きく強化されていることがわかった。

本稿で説明した各文法は、いろいろな局面で活用できるので、ぜひ今後のアプリケーション開発時のコーディング技法としてチャレンジし、開発の幅を広げていただきたい。

M

図9

【宣言部】

```
type
  TStrProc = reference to procedure(sStr: String);
```

【実装部】

```
procedure TForm1.Button1Click(Sender: TObject);
var
  pProc: TStrProc;
begin
  //無名メソッドを変数pProcに代入
  pProc := procedure(sStr: String)
  begin
    ShowMessage(sStr);
  end;

  //変数pProcの使用
  pProc('テスト');
end;
```

図10

【宣言部】

```
type
  TCalcFunc = reference to function(a, b: Integer): Integer;
```

【実装部】

```
procedure Calculate(iValue1, iValue2: Integer; Calc: TCalcFunc);
var
  iRet: Integer;
begin
  //受け取った無名メソッドを実行
  iRet := Calc(iValue1, iValue2);

  //処理結果を出力
  ShowMessage(IntToStr(iRet));
end;

procedure TForm1.Button2Click(Sender: TObject);
begin

  //足し算を行う無名メソッドを引数にセット
  Calculate(3, 4, function(a, b: Integer): Integer
                  begin
                    Result := a + b
                  end);

  //掛け算を行う無名メソッドを引数にセット
  Calculate(6, 8, function(a, b: Integer): Integer
                  begin
                    Result := a * b
                  end);
end;
```

無名メソッドが引数となっている

無名メソッドを引数として
サブルーチンを実行
(異なるメソッド(条件)で
プログラムが実行できる)

図11

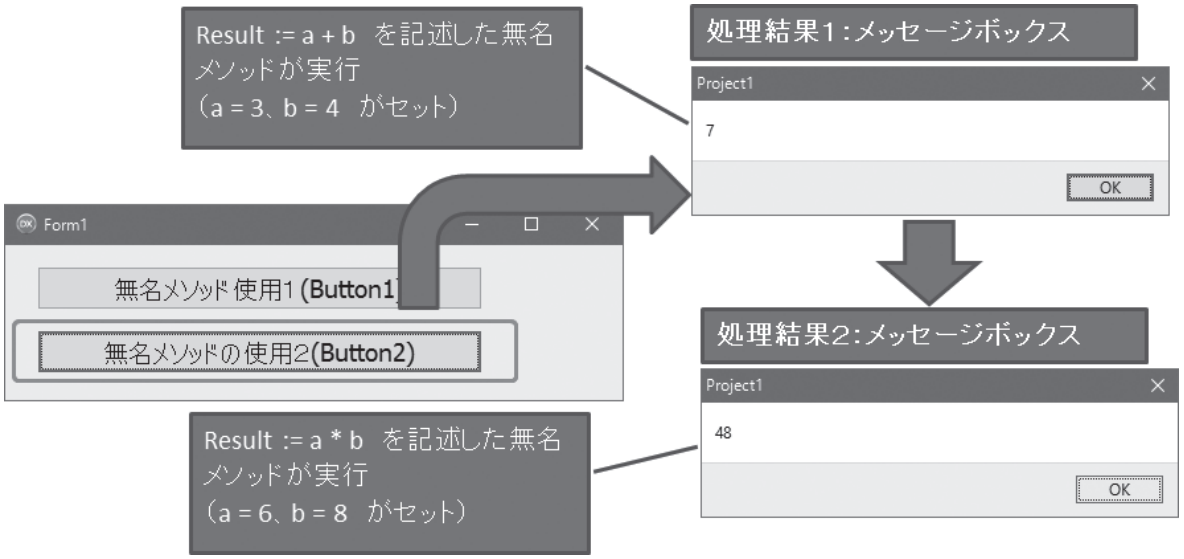


図12

```
procedure TdmMain.DataUpdate;
var
  dbTran: TDBXTransaction; //トランザクション変数
begin

  //①トランザクションの開始
  dbTran := SQLConnection1.BeginTransaction;
  try
    //②データの登録/変更/削除等の更新処理
    SQLQuery1.SQL.Clear;
    SQLQuery1.SQL.Add('UPDATE PNAME SET PNAME = ''テスト'' ');
    SQLQuery1.SQL.Add('WHERE PCODE = 1 ');

    SQLQuery1.ExecSQL;

    //③トランザクションのコミット
    SQLConnection1.CommitFreeAndNil(dbTran);

  except
    //例外処理
    //ロールバックを行う
    SQLConnection1.RollbackFreeAndNil(dbTran);
    raise;
  end;
end;
```

図13

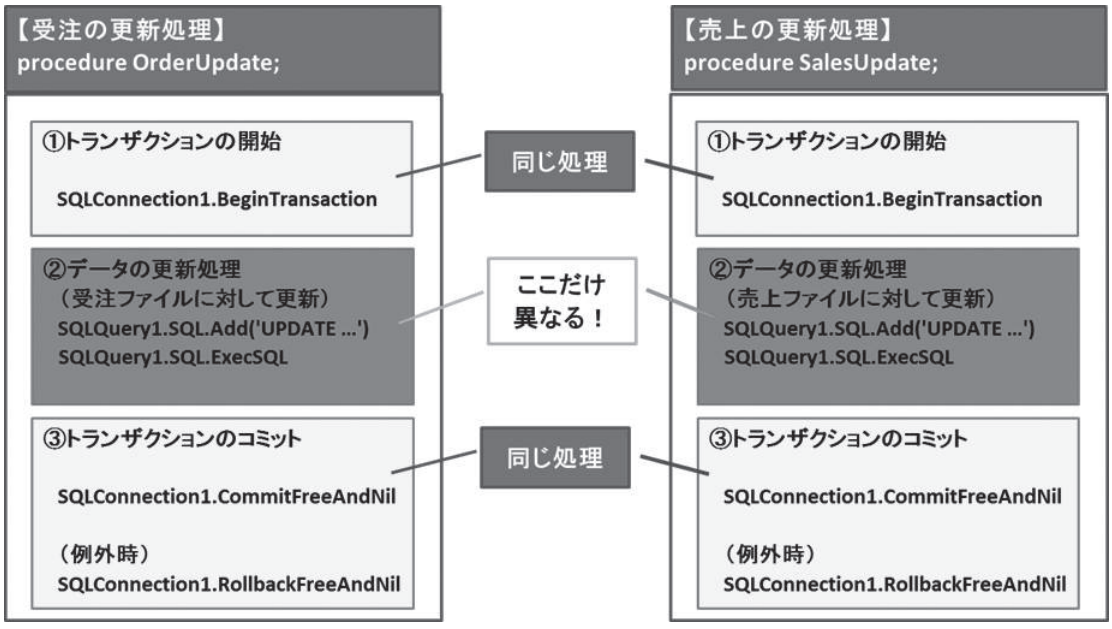


図14

```
procedure TdmMain.DataUpdate(SQLProc: TProc);
var
  dbTran: TDBXTransaction; //トランザクション変数
begin
  //①トランザクションの開始
  dbTran := SQLConnection1.BeginTransaction;
  try
    //②データの登録/変更/削除等の更新処理
    SQLProc;

    //③トランザクションのコミット
    SQLConnection1.CommitFreeAndNil(dbTran);

  except
    //例外処理
    //ロールバックを行う
    SQLConnection1.RollbackFreeAndNil(dbTran);
    raise
  end;
end;
```

図15

```
procedure TdmMain.OrderUpdate;
begin
  //データの更新部分を無名メソッドで指定
  DataUpdate(procedure ()
             begin
               //受注データの更新処理
               SQLQuery1.SQL.Clear;
               SQLQuery1.SQL.Add('UPDATE ... ');
                        :
               SQLQuery1.ExecSQL;
             end
             );
end;
```

図16

図17

【宣言部】
TDate型のレコードヘルパクラスとして、TDateToIntHelper型を宣言。
メソッドとして、整数値に変換する関数を定義。

```
type
  TDateToIntHelper = record helper for TDate
    function ToInteger: Integer;
  end;
```

図18

【実装部】
TDateToIntHelper型のメソッドであるToIntegerのロジックを記述。
元のレコード値(Self)に対して、整数値に変換した値をResultにセット。

```
{ TDateToIntHelper }

function TDateToIntHelper.ToInteger: Integer;
begin
  if Self <> 0 then
    Result := StrToInt(FormatDateTime('YYYYMMDD', Self))
  else
    Result := 0;
end;
```

図19

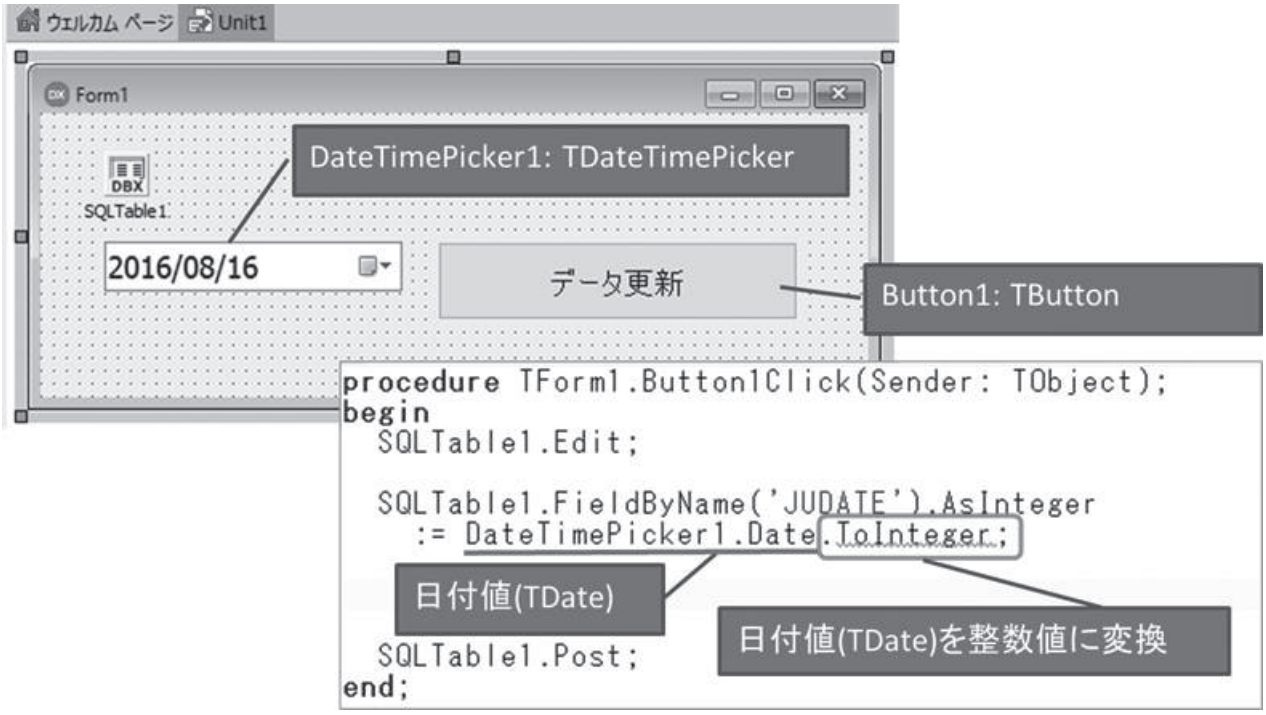


図20

【宣言部】
Integer型のレコードヘルパクラスとして、TIntToDateHelper型を宣言。

```
type
  TIntToDateHelper = record helper for Integer
    function ToDate: TDate;
  end;
```

【実装部】
TIntToDateHelper型のToDateメソッドを記述。

```
{ TIntToDateHelper }

function TIntToDateHelper.ToDate: TDate;
begin
  if Self <> 0 then
    Result := StrToDate(FormatFloat('0000/00/00', Self))
  else
    Result := 0;
end;
```

図21

【宣言部】
Integer型のエリアスとしてTDateInt型を宣言し、TDateInt型のレコードヘルパとして、TIntToDateHelper型を宣言。

```
type
  TDateInt = type Integer;     //Integer型のエリアス

  TIntToDateHelper = record helper for TDateInt
    function ToDate: TDate;
  end;
```

図22

【使用例】

```
procedure TForm1.Button4Click(Sender: TObject);
var
  i: Integer;
  d: TDate;
  s: String;
begin
  i := 20160816;  // 日付整数値

  d := TDateInt(i).ToDate;    //TDateToIntHelper のToDateメソッド
  s := i.ToString;            //TIntegerHelper のToStringメソッド
end;
```

図23

【宣言部】
TCustomEditのクラスヘルパとして、TCustomEditHelper型を宣言。
整数値と、日付値に対するプロパティを定義。

```
type
  TCustomEditHelper = class helper for TCustomEdit
  private
    function GetAsInteger: Integer;
    procedure SetAsInteger(const Value: Integer);
    function GetAsDate: TDate;
    procedure SetAsDate(const Value: TDate);
  public
    property AsDate: TDate read GetAsDate write SetAsDate;
    property AsInteger: Integer read GetAsInteger write SetAsInteger;
  end;
```

【実装部】
各プロパティに対する取得(Get)、書込み(Set)メソッドを実装。

```
{ TCustomEditHelper }

function TCustomEditHelper.GetAsDate: TDate;
begin
  Result := StrToDate(Self.Text);
end;

function TCustomEditHelper.GetAsInteger: Integer;
begin
  Result := StrToInt(Self.Text);
end;

procedure TCustomEditHelper.SetAsDate(const Value: TDate);
begin
  Self.Text := DateToStr(Value);
end;

procedure TCustomEditHelper.SetAsInteger(const Value: Integer);
begin
  Self.Text := IntToStr(Value);
end;
```

図24

【使用例】

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Edit1.AsInteger := 12345678;    // Editに数値をセット
  Edit2.AsDate := Date;           // Editに日付値をセット
end;
```

図25

【VCLフォームアプリケーション】

```
unit Unit1;

interface

uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,
  System.Classes, Vcl.Graphics, Vcl.Controls, Vcl.Forms, Vcl.Dialogs;

type
  TForm1 = class(TForm)
  private
    { Private 宣言 }
  public
    { Public 宣言 }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

end.
```

図26

【マルチデバイスアプリケーション】
(FireMonkey)

```
unit Unit1;

interface

uses
  System.SysUtils, System.Types, System.UITypes, System.Classes, System.Variants,
  FMX.Types, FMX.Controls, FMX.Forms, FMX.Graphics, FMX.Dialogs;

type
  TForm1 = class(TForm)
  private
    { private 宣言 }
  public
    { public 宣言 }
  end;

var
  Form1: TForm1;

implementation

{$R *.fmx}

end.
```

図27


http://docwiki.embarcadero.c...
System - RAD Studio API D...
Trend ツールバー
アカウント作成
ログイン
embarcadero
ページ
ソースを表示
履歴
表示設定
検索
表示
検索
RAD STUDIO SEATTLE
メイン ページ - Seattle ライブラリ リファレンス
ユニット リスト
Seattle のトピック
Seattle のコード例
新しいバージョン
Berlin ライブラリ
以前のバージョン
XE8 ライブラリ
XE7 ライブラリ
XE6 ライブラリ
XE5 ライブラリ
XE4 ライブラリ
XE3 ライブラリ
XE2 ライブラリ
System
親: ユニット リスト
Delphi ランタイム ライブラリ (RTL) のユニット、クラス、関数、型、変数、定数が含まれています。また、System.Bindings、System.Generics、System.Win の各ユニット スコープも含まれています。System ユニット スコープと System ユニットは同じである点に注意してください。System ユニットで公開されている Delphi 専用組み込みルーチンの一覧については、「Delphi 組み込みルーチン」を参照してください。
目次 [非表示]
1 ユニット スコープ
2 ユニット
3 クラス
4 ルーチン
5 型
6 変数
7 定数
ユニット スコープ
System.Analytics
System.Beacon
System.Bindings
LiveBinding フレームワーク エンジンのユニット スコープです。LiveBinding エンジンを実装しているユニットが含まれています。
System.Bluetooth
アプリケーションの動作デバイスの Bluetooth 機能を使用して、リモート デバイスで動作するアプリケーションに接続するためのクラスが用意されています。
ジェネリックス (後でインスタンス化されてパラメータとして使われる一般化された型を含むアルゴリズム) を使用するため

図28

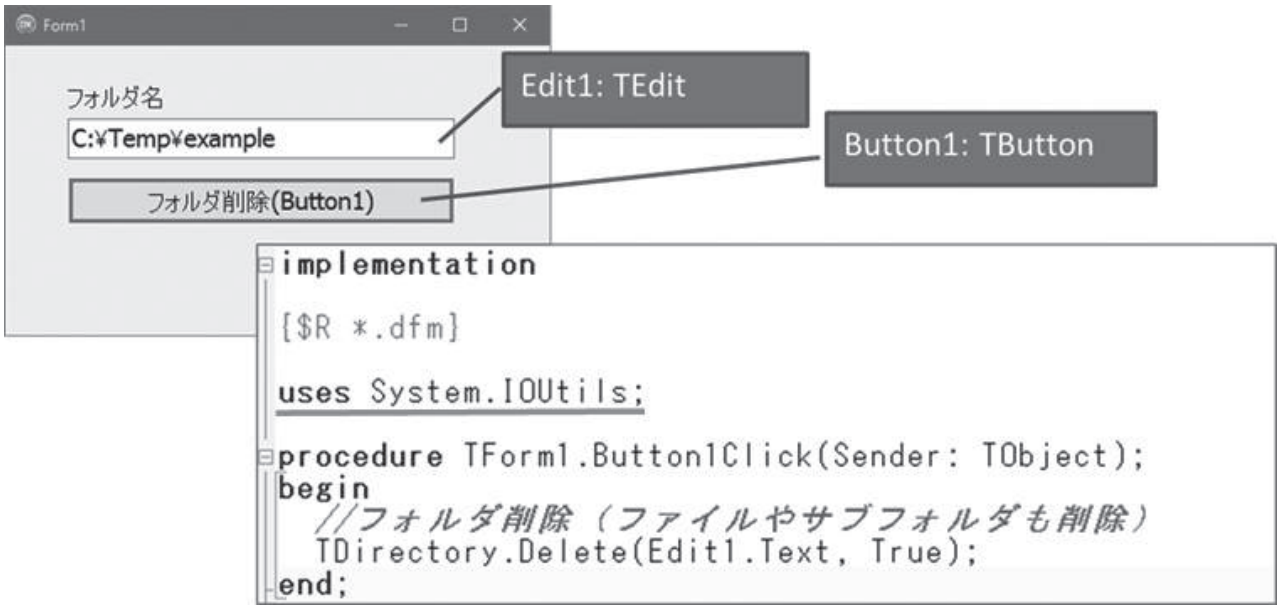

Form1
フォルダ名
C:¥Temp¥example
フォルダ削除(Button1)
Edit1: TEdit
Button1: TButton
implementation
{$R *.dfm}
uses System.IOUtils;
procedure TForm1.Button1Click(Sender: TObject);
begin
//フォルダ削除（ファイルやサブフォルダも削除）
TDirectory.Delete(Edit1.Text, True);
end;

図29

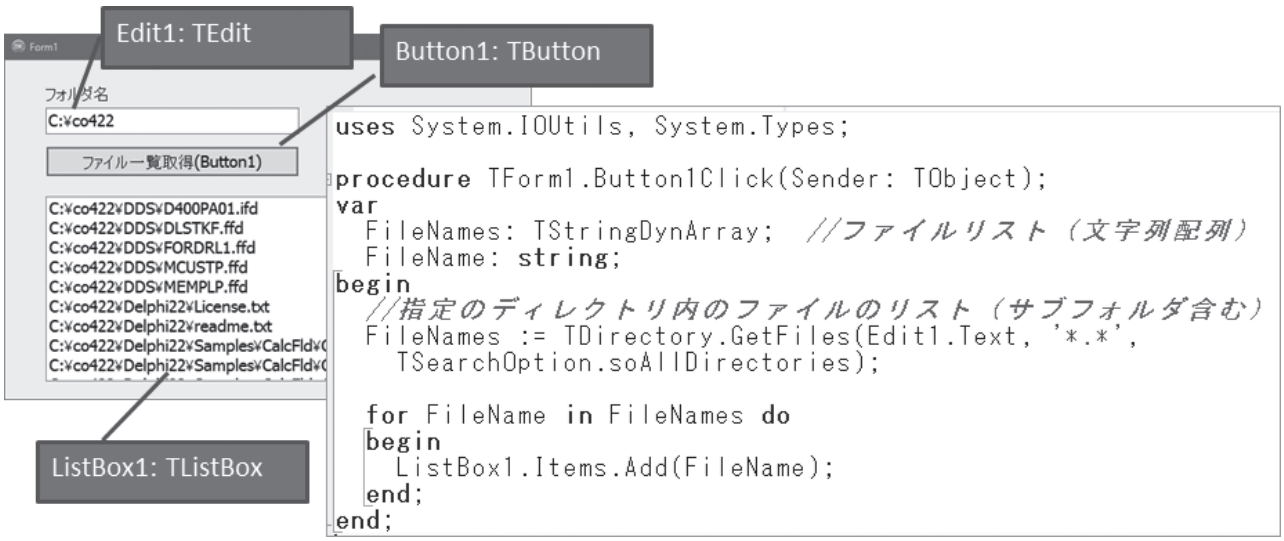
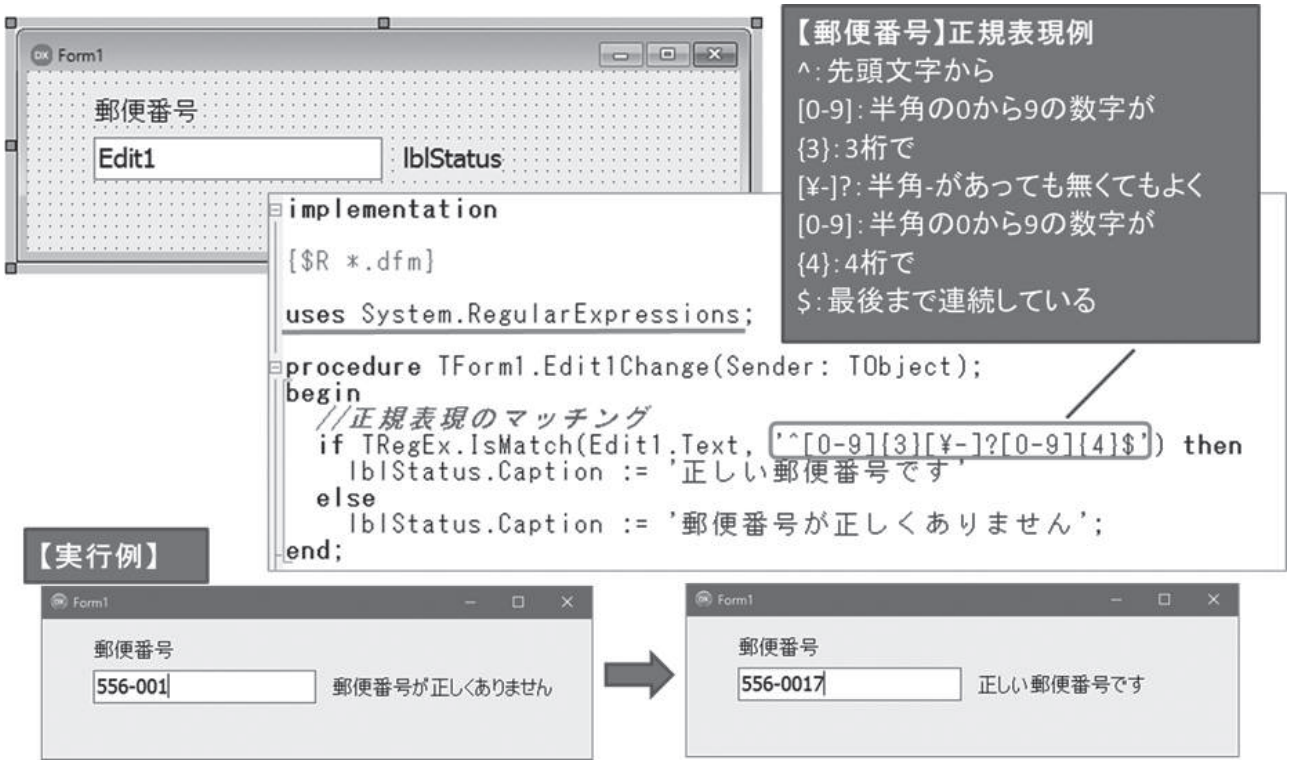

Edit1: TEdit
Button1: TButton
ListBox1: TListBox
フォルダ名
C:¥co422
ファイル一覧取得(Button1)
C:¥co422¥DDS¥D400PA01.ifd
C:¥co422¥DDS¥DLSTKF.ffd
C:¥co422¥DDS¥FORDRL1.ffd
C:¥co422¥DDS¥MCUSTP.ffd
C:¥co422¥DDS¥MEMPLP.ffd
C:¥co422¥Delphi22¥License.txt
C:¥co422¥Delphi22¥readme.txt
C:¥co422¥Delphi22¥Samples¥CalcFld¥
C:¥co422¥Delphi22¥Samples¥CalcFld¥
uses System.IOUtils, System.Types;
procedure TForm1.Button1Click(Sender: TObject);
var
FileNames: TStringDynArray; //ファイルリスト（文字列配列）
FileName: string;
begin
//指定のディレクトリ内のファイルのリスト（サブフォルダ含む）
FileNames := TDirectory.GetFiles(Edit1.Text, '*.*',
TSearchOption.soAllDirectories);
for FileName in FileNames do
begin
ListBox1.Items.Add(FileName);
end;
end;

図30


Form1
郵便番号
Edit1
lblStatus
【郵便番号】正規表現例
^:先頭文字から
[0-9]:半角の0から9の数字が
{3}:3桁で
[¥-]?:半角-があっても無くてもよく
[0-9]:半角の0から9の数字が
{4}:4桁で
$:最後まで連続している
implementation
{$R *.dfm}
uses System.RegularExpressions;
procedure TForm1.Edit1Change(Sender: TObject);
begin
//正規表現のマッチング
if TRegEx.IsMatch(Edit1.Text, '^[0-9]{3}[¥-]?[0-9]{4}$') then
lblStatus.Caption := '正しい郵便番号です'
else
lblStatus.Caption := '郵便番号が正しくありません';
end;
【実行例】
Form1
郵便番号
556-001
郵便番号が正しくありません
Form1
郵便番号
556-0017
正しい郵便番号です

宮坂 優大

株式会社ミガロ.

システム事業部 システム1課

[Delphi/400]

FastReportを活用した電子帳票作成テクニック

略歴
1982 年 11 月 19 日生まれ
2006 年 近畿大学 理工学部卒業
2006 年 4 月 株式会社ミガロ. 入社
2006 年 4 月 システム事業部配属

現在の仕事内容
主に Delphi/400 を利用したシステムの受託開発と MKS サポートを担当。Delphi および Delphi/400 のスペシャリストを目指して精進する毎日である。

1.はじめに

基幹システムの構築では帳票機能が不可欠であるが、最近ではプリンタに紙で出力する従来の帳票機能ではなく、電子帳票での開発も増えてきた。

電子帳票化を実現する場合、一番の目的はコスト削減であることが多い。電子化によって用紙はもちろん、トナーなどプリンタ関連の消耗品にかかるコストや、その運用・保守費用を削減できる。帳票の保管や閲覧がシステム上で行えると、物理的なスペース制約や帳票紛失のトラブル解決にもつながるので、業務的なメリットは大きい。

また電子帳票であれば、ネットワークを通じて取引先にデータとして渡せるので、BtoB などで必要とされることも多い。もちろん必要に応じて電子ファイルから紙帳票として印刷もできる。総じてデメリットは少ないといえる。

電子帳票の実現に際しては、専用のパッケージソフトを導入する場合も多いが、ソフトの仕様に帳票書式や業務を合わせる必要があったり、社内システムとの連携が難しいこともある。

そのため自社用のシステムを開発・運用している場合は、そのシステムのなかで電子帳票を実装する要望が多い。電子帳票の開発には技術的にハードルの高いイメージがあるが、Delphi/400 ではツールの機能を活用して容易に実現できる。

本稿では電子帳票化テクニックとして、帳票を画像ファイルで出力する方法と、電子帳票でとくに要望が多い電子データ印を押印する手法について説明していく。

なお電子帳票には pdf 形式や画像形式があるが、本稿では画像加工のテクニックを扱うため画像形式を題材としている。もちろん Delphi/400 では、pdf の出力も可能である。

2.FastReportを使用した電子帳票化

FastReport と は、Delphi/400 XE3 以降で新しくバンドルされた帳票ツールである。

本稿では、社内ワークフローで使用される購入申請書をテーマに、FastReport を使用して電子帳票（画像ファイル）を作成する。作成する購入申請書フォーマットは、【図 1】に示す。

Delphi/400 XE7 と FastReport を使用して、以下のように電子帳票を作成していく。

2-1. 帳票フォーマットの作成

電子帳票作成の準備として、印刷フォームに TfrxReport コンポーネントを貼り付ける。【図 2】

次に、レポートデザイナを使用して帳票フォーマットを作成する。レポートデザイナを起動するには、Delphi/400 の開発画面に貼り付けた TfrxReport コンポーネントをダブルクリックする。【図 3】

レポートデザイナを起動したあとは、自由に線（Line）、枠（Shape）、文字（Memo）を使用してデザインする。

まずは線を引き、レイアウトのイメージを作成する。

図1

購入申請書

申請日

購入予定

＜申請者＞

所属

氏名

下記の通り、備品購入を申請いたします。

品名	単価	数量	金額	摘要
		合計金額		

備考

承認者2印	承認者1印	申請者印

レポートデザイナの左側にあるコンポーネントパレットから描画を選択し、ポップアップから線オブジェクトを選択する。マウスでドラッグ＆ドロップするだけで、簡単に線を引ける【図 4】。線オブジェクトは縦もしくは横にしか引けないので、斜めの線を引く場合は対角線を使用する。

次に固定文字列を貼り付ける。固定文字列をレイアウトに貼り付けるには、テキストオブジェクトを使用する。

コンポーネントパレットからテキストオブジェクトを選択し、文字を貼り付けたい箇所をマウスでクリックする。そこでテキストを入力するダイアログが起動するので、テキストタブに出力したい文字列を入力する。【図 5】

日付や名前、データベースから取得する値を設定する場合も、固定文字列と同じようにテキストオブジェクトとして貼り付ける。

また、ダイアログ内で任意の変数を［変数名］と記述することで、文字列をプログラム内の変数として扱える（ソースは後述）。たとえば、［VALUE001］のように設定できる。【図 6】

フッター部備考の下ラインは、破線で設定する。破線で設定するには、該当の線オブジェクトを選択し、「Frame ｜ Style」プロパティを fsDot で設定する。【図 7】

最後に線オブジェクトとテキストオブジェクトを組み合わせて、フォーマットを作成していく。一通り帳票レイアウトの設計が完了すれば、そのままレポートデザイナを「×」ボタンで終了する。

これで帳票フォーマットは作成できたので、次にボタンを押下した際の出力ロジックについて説明する。

2-2. 帳票データの出力

前述のように作成した帳票フォーマットでは、変数として「VALUE001」を宣言しているが、帳票出力時には注意が必要である。変数を宣言している場合には、必ず値を設定しておかねば、実行時にエラーとなる。

そのため、初期化ロジックとして InitVariables という手続きを作成し、文字型であればブランクを設定し、それ以外の型であれば 0 を設定しておくと、値の設定漏れを防げる。【ソース 1】

次に、帳票で実際の変数値をセットする。帳票フォーマット作成時に変数として宣言した［VALUE001］に値をセットする場合は、Script.Variables［(変数名)］を指定する。【ソース 2】

もちろん IBM i や SQLServer などのデータベースから取得した値を設定することも可能である。

データベースや配列を扱う方法については、2014 年発行のミガロ .テクニカルレポート№ 7 にある『FastReport を使用した帳票作成テクニック』で詳しく説明しているので、参考にしていただきたい。

ここまでで、帳票への出力内容が完成したので、次に帳票を画像ファイルとして保存する手法を説明する。

2-3. 帳票画像ファイルの保存

通常、FastReport で作成した帳票は「frxReport1.Print」メソッドを使用し、プリンタへ印刷する。しかし本稿では電子帳票として出力するので、画像ファイルとしての保存方法を説明する。画像ファイルの形式は、一般的に JPEG の使用が多い。

FastReport で帳票を JPEG 画像ファイルとして出力するには、TfrxJPEGExport コンポーネントが使用できる。

まず、TfrxJPEGExport コンポーネントを画面に貼り付ける【図 8】。次に、TfrxJPEGExport コンポーネントにプロパティを設定する。保存する画像品質は、「Resolution」を指定する。【ソース 3】

帳票を出力するメインのロジックは、【ソース 4】のとおりである。画像ファイルは Stream 形式を使って出力する。ポイントは、TfrxJPEGExport の Stream に msJPEG として内部生成した「TMemoryStream」を割り当てる点である。

あとは、TfrxJPEGExport コンポーネントを TfrxReport コンポーネントの Export メソッドを使用して、msJPEG に画像ファイル情報を転送する（【ソース 4】の①）。最後に、転送された画像ファイルを SaveToFile で任意の場所に保存できる（【ソース 4】の②）。これで電子帳票としての出力・保存が完成である。

これにより、購入申請の情報を画面で入力し、「購入申請書を発行」ボタンを押すと、電子帳票として保存するプログラムが作成された。保存したあとは、帳票表示用の画面を起動し、購入申請書の電子帳票を画面表示できる（これも帳票が電子ファイルであるメリットといえる）。【図 9】

また電子帳票でも紙の帳票と同じように、押印を必要とすることが非常に多い。そこで次に、電子データ印を作成し、この電子帳票に押印するテクニックを説明する。

【図 9】のプログラム例では、表示された電子帳票の下にある「確認印」ボタンを押すことで、電子データ印を生成し、マウスで好きな位置に貼り付けられるようにしている。

3.電子データ印の作成

電子データ印といっても、内容としてはデータ印を画像として作成するだけである。

電子データ印作成の手順としては、まずデータ印の枠、文字列を画像ファイルとして作成する。そして 2 つの画像ファイルを 1 つに合成することで、電子データ印として完成する。この手法で電子データ印を作成すれば、電子帳票上での押印に使用できる。

3-1　電子データ印枠の作成

最初に、2 つの TBitmap コンポーネントを配置する。1 つは電子データ印の枠用（bmpBase1）、もう 1 つは電子データ印に出力する出力日、名前、所属部署など文字列用（bmpOver1）として使用する。

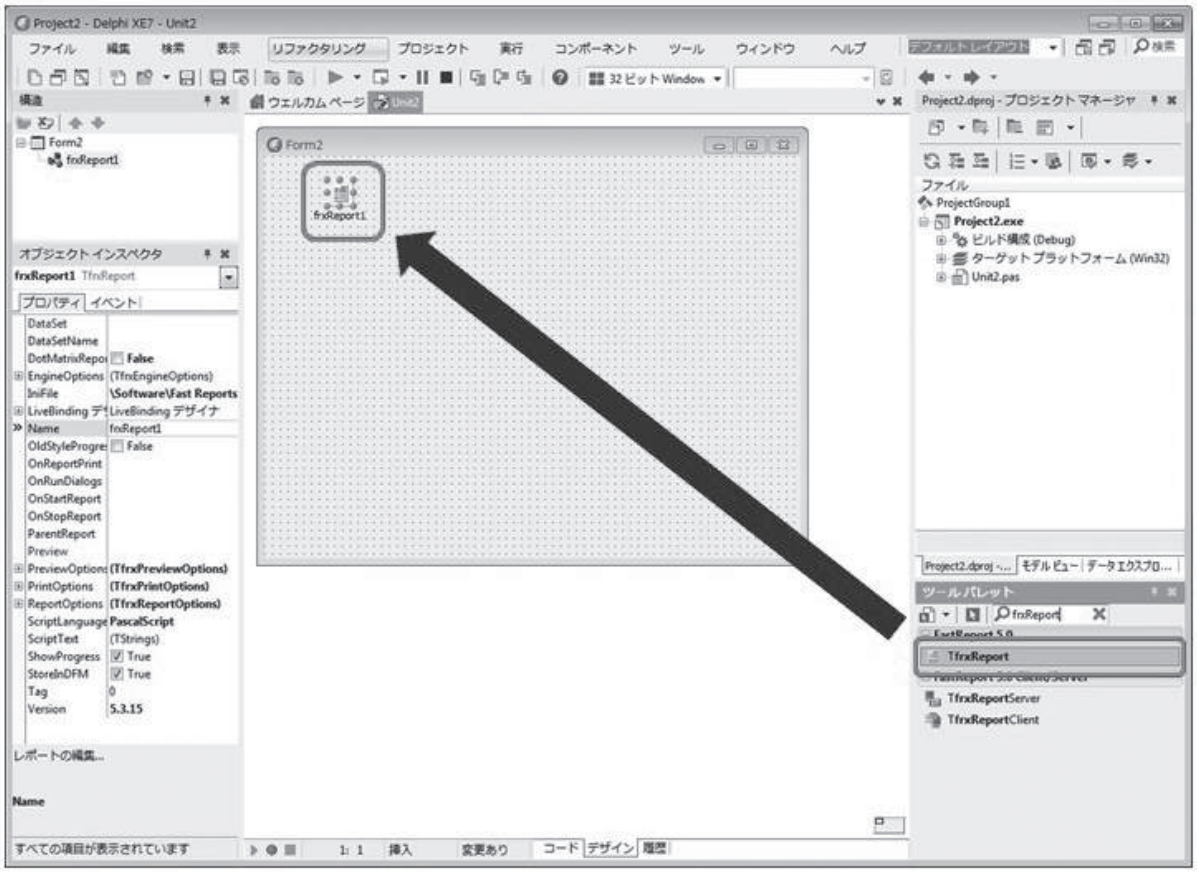
そして TImage コンポーネントを配置し、マウス操作で表示する電子データ印のプレビュー用（imgEditor）として使用する。

bmpBase1 と bmpOver1 はプログラム内で生成し、imgEditor は TImage コンポーネントを画面に貼り付ける。【図 10】

まずは電子データ印の枠色、線の太さを設定する。Ellipse メソッドを使用し、bmpBase1 に円を描画する。さらに、なかにある 2 本線を追加で描画する。【ソース 5】

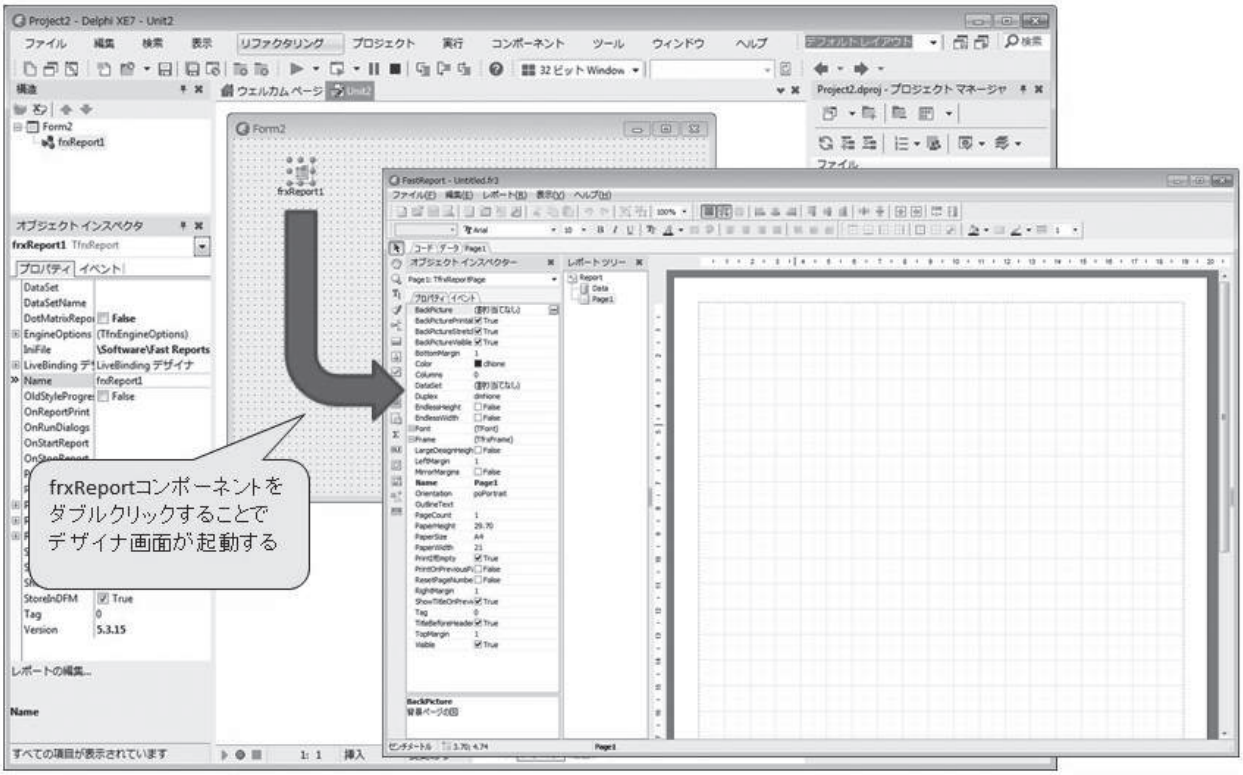
これで、電子データ印の枠が完成である。次に、この枠のなかに出力する文字列の作成ポイントを説明する。

図2



TfrxReportコンポーネントを貼り付ける

図3



レポートデザイナの起動

3-2　出力文字列の作成

電子データ印内に出力する文字列は、内容としてはただの文字であるが、画像化するため、フォントサイズや幅などの調整が重要になる。出力する文字列の内容は、上段に所属部署、中段に日付、下段に名前である。【図 11】

まず、上段に表示する所属部署の文字列を調整する。長い部署名になると、電子データ印の枠からはみ出すので、文字列の長さによってフォントサイズを変更する必要がある。

このプログラムでは、半角 8 文字まで出力できるように調整している。半角 1 ～ 6 文字ならフォントサイズ 15、半角 7 ～ 8 文字ならフォントサイズ 11 を設定する。これは電子データ印のサイズや、出力する文字列の想定によって変わってくる。

フォントサイズを決定したあとは、Canvas.TextOut で文字列を出力する。第 1 引数は X 座標の位置、第 2 引数は Y 座標の位置、第 3 引数は出力する文字列を指定する。X 座標の位置は、出力する文字列の長さによって調整する。【ソース 6】

次は、中段に表示する日付の文字列である。日付については、「YYYY/MM/DD」形式の半角 10 文字固定で出力するので、フォントサイズの調整は不要である。【ソース 7】

最後は、下段に表示する名前の文字列を調整する。電子データ印に表示する名前は、所属と同様にフォントサイズを調整し、bmpOver1 の Canvas に出力する。【ソース 8】

これで電子データ印に出力する文字列の調整が完了である。

3-3　電子データ印画像の作成

ここまでの作業で、 電子データ印の枠（bmpBase1）と電子データ印の文字列（bmpOver1）が準備できた。この 2 つを合成することで、電子データ印が完成する。【図 12】

画像を合成するには、StretchDraw メソッドを使用する。画像ファイルの合成は、単純に bmpBase1 の Canvas に bmpOver1 を描画するだけである。【ソース 9】

これで 2 つの画像ファイルを 1 つの画像ファイルに合成できた。電子データ印の画像ファイルの完成である。

補足として、この電子データ印を画面上で押印する際に、マウスでわかりやすく表示する方法を説明する。

まずプレビュー用 TImage コンポーネント（imgEditor）の Picture プロパティに電子データ印を読み込ませる。ここからはマウスが画面の帳票内にある場合のみ、マウス位置に電子データ印を表示する動作を実装していく。【図 13】

このマウスの動作は MouseMove、MouseLeave イベントを使うことで、簡単に実装できる。【ソース 10】

これで電子データ印の画像ファイルと、押印する際の画面動作プログラムを実装できた。

4.電子帳票への押印機能実装

2. で電子帳票の画像ファイルを作成し、3. では電子帳票に押印する電子データ印の画像ファイルを作成した。ここからは、この 2 つの画像ファイルを合成し、電子帳票上での押印を実現していく。

4-1 画像合成の準備

画面の帳票上でのマウス動作処理までを作成したが、さらにクリックした際、その電子帳票と電子データ印の画像ファイルを合成する処理を行う。【図 14】

この処理のために、2 つの TBitmap コンポーネントと 1 つの TJPEGImage コンポーネントを用意する。2 つの TBitmap コンポーネントのうち、1 つは電子帳票用（bmpBase2）、もう 1 つは電子データ印用（bmpOver2）、また TJPEGImage コンポーネントは JPEG ファイルとして取り扱うために使用する（jpgBase2）。

はじめに、電子帳票ファイルが JPEG 形式かどうかをチェックする。JPEG 形式だった場合には、Bitmap へ変換するために jpgBase2 に読み込み、その後、合成用に bmpBase2 へ再度読み込ませる。Bitmap 形式の場合は、直接 bmpBase2 に読み込ませる。【ソース 11】

JPEG、Bitmap 以外の画像ファイルを扱う場合は、いったん WICImage 型に読み込み直し、bmpBase2 にセットすることで対応する。

次に bmpBase2 の幅、高さを電子帳票ファイルの高さに設定し、押印する電子データ印を設定する。bmpOver2 は Assign メソッドを使用すれば、電子データ印画像を設定できる。【ソース 12】

4-2　電子押印機能の実装

最後に、bmpBase2 の StretchDraw メソッドを使って、bmpOver2 の押印を実装する。これは電子データ印自体を作成するときと同じ手法で、画像ファイル同士を合成する。

あとは押印した電子帳票を出力する TImage コンポーネントで読み込めば、処理が完了である。【ソース 13】【図 15】

これで電子帳票の作成、およびそれを応用した電子データ印の押印機能を実装できた。ここまでの処理ロジックは、【ソース 14】に実装ソース例をまとめているので、参考にしていただきたい。

社内システムでは、このような電子帳票を申請データとして保存し、ワークフローとして承認者がさらに押印する仕組みを構築することもできる。

5.まとめ

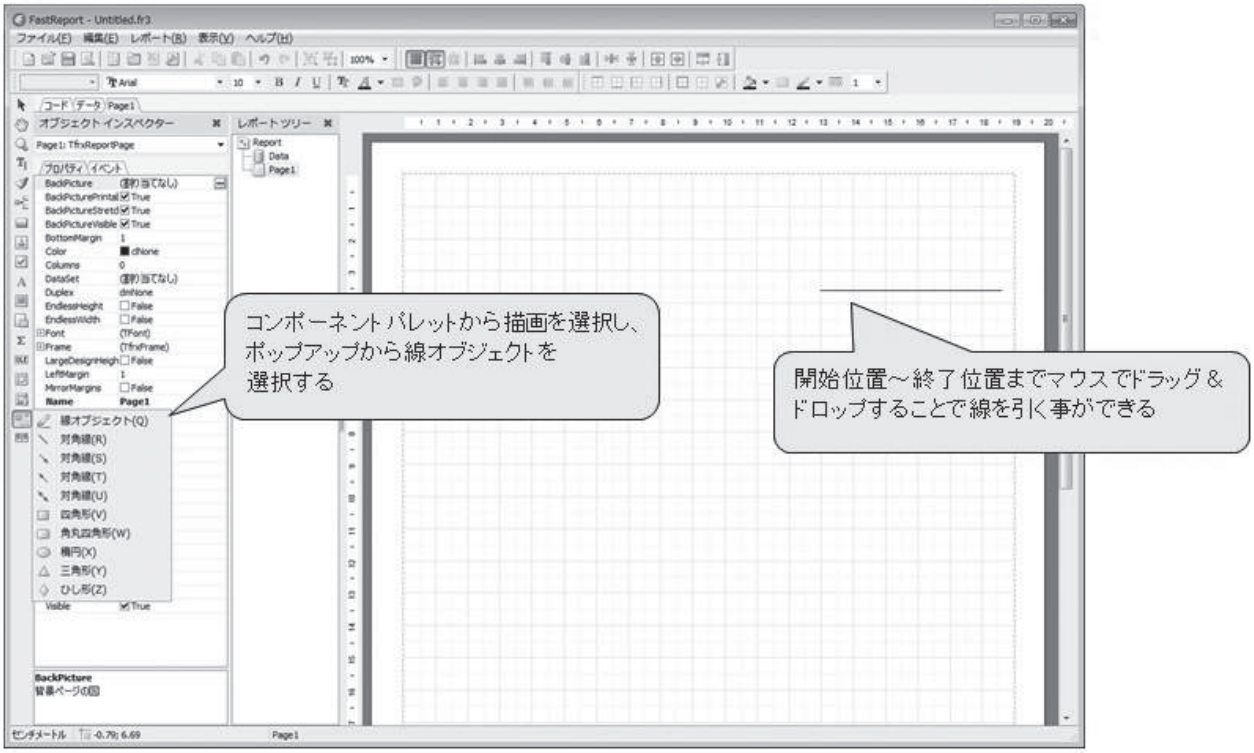
本稿では FastReport で電子帳票を作成する方法や、データ印画像の作成方法、電子帳票にデータ印を押印する方法を説明してきた。ここで紹介したテクニックを利用すれば、売上伝票や請求書、また商品画像等の電子帳票の作成も可能である。

画像や描画は一般には難しいプログラム分野であるが、FastReport が画像出力形式に対応しているので、Delphi/400 では簡単に電子帳票を実装できる。

冒頭でも述べたとおり、電子帳票化にはメリットも多く、FastReport では開発にも手間がかからない。帳票を開発する際には、電子帳票での出力も、主機能の 1 つとして組み込む価値が十分にある技術といえる。

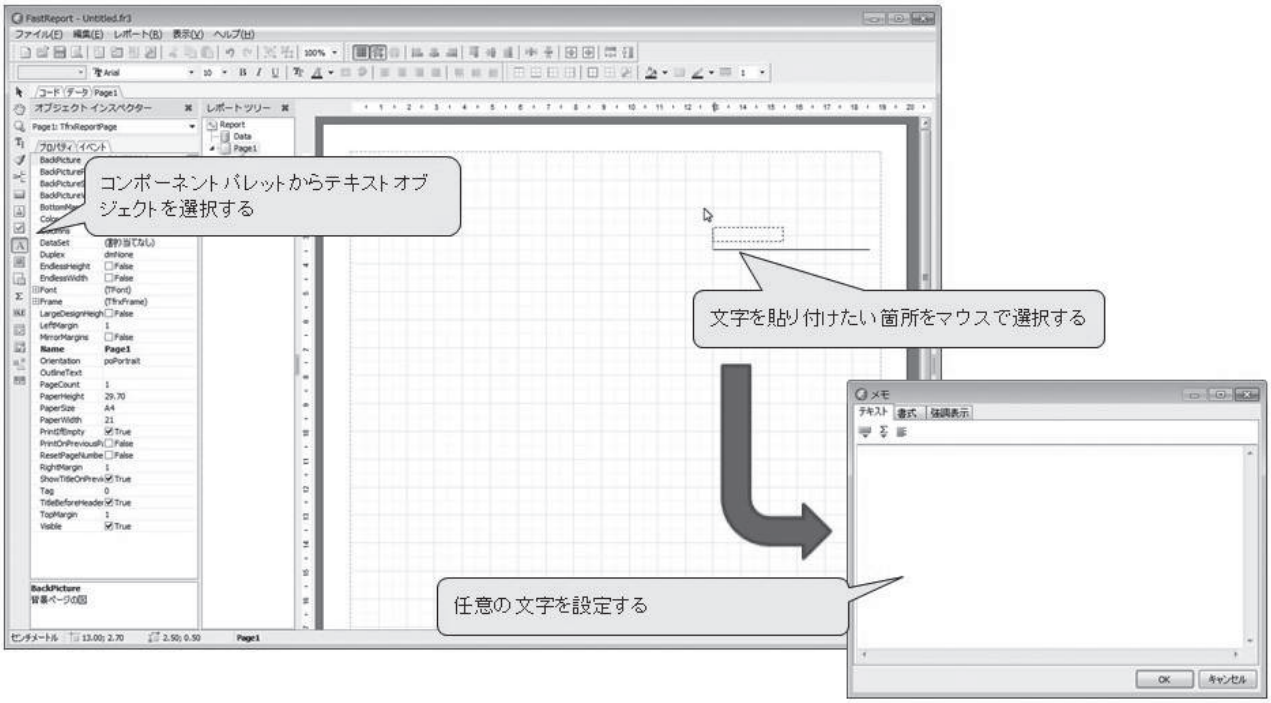
M

図4



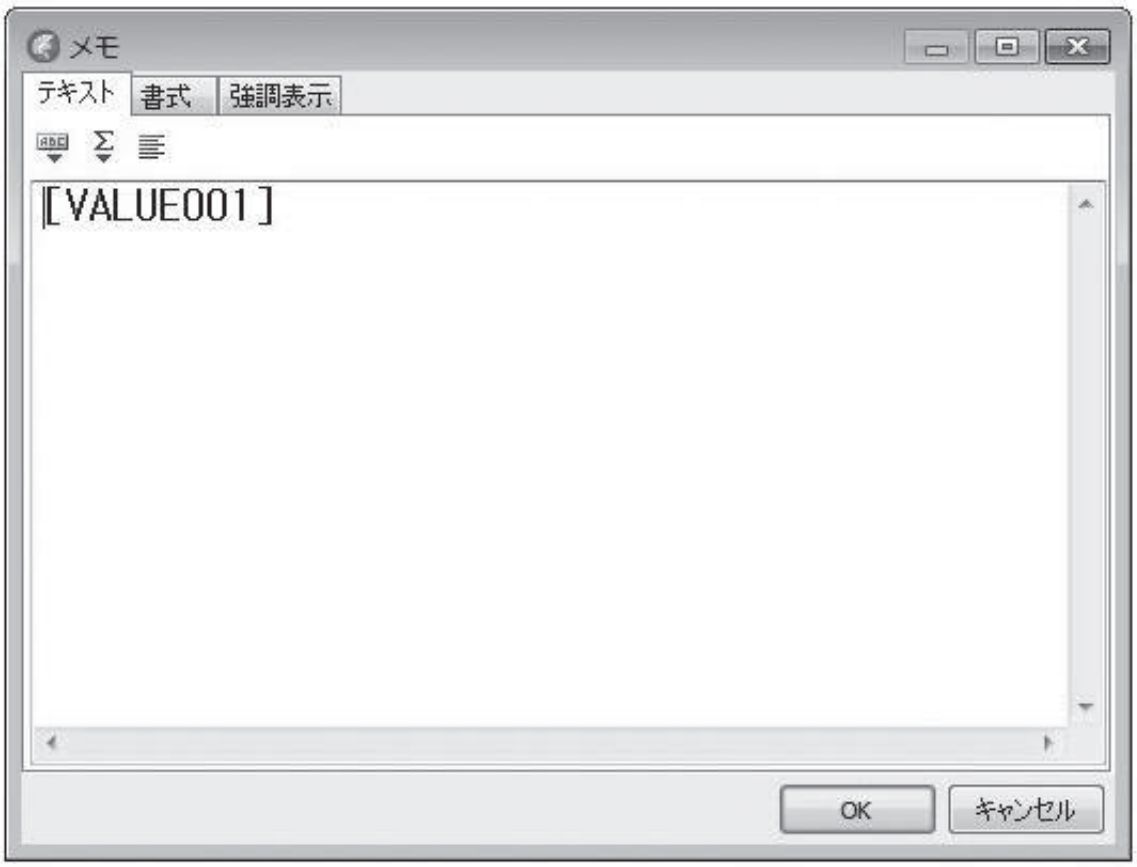
帳票フォーマットに線を引く

図5



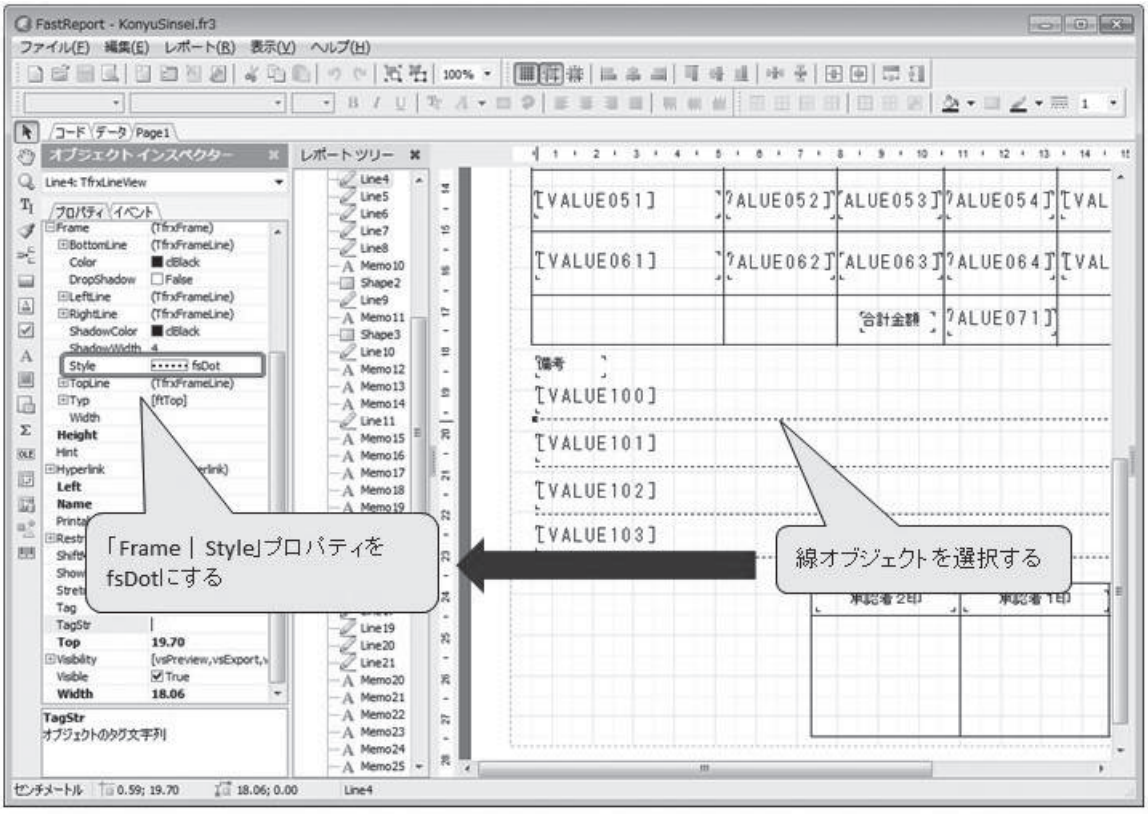
帳票フォーマットに文字を書く

図6



変数の定義

図7



破線の設定

ソース1

```
{*******************************************************************************
 目的：FastReportの変数の初期化
 引数：Afrx -  TfrxReportコンポーネント
 戻値：
*******************************************************************************}
procedure TfrmReportSample002.InitVariables(Afrx: TfrxReport);
var
  i: Integer;
begin
  //帳票フォーマット内で宣言した変数([VALUE001]等)に値をセットしない場合、
  //エラーとなるため、処理の最初にブランクもしくは0を初期セットする

  // 変数の初期化
  for i := 0 to Afrx.ComponentCount - 1 do
  begin
    // TfrxMemoviewだった場合
    if Afrx.Components[i] is TfrxMemoView then
    begin
      if (Pos('[', TfrxMemoView(Afrx.Components[i]).Memo.Text) > 0) and
         (Pos(']', TfrxMemoView(Afrx.Components[i]).Memo.Text) > 0) then
      begin
        // DisplayFormat.Kindが"fkText"だった場合、文字型と判断する
        if TfrxMemoView(Afrx.Components[i]).DisplayFormat.Kind = fkText then
        begin
          // 初期値としてブランクをセットする
          Afrx.Script.Variables[
          Copy(TfrxMemoView(Afrx.Components[i]).Memo.Text,
            Pos('[', TfrxMemoView(Afrx.Components[i]).Memo.Text) + 1,
            Pos(']', TfrxMemoView(Afrx.Components[i]).Memo.Text) -
            (Pos('[', TfrxMemoView(Afrx.Components[i]).Memo.Text) + 1) )] := '';
        end
        // DisplayFormat.Kindが"fkText"以外だった場合、数値型と判断する
        else
        begin
          // 初期値として"0"をセットする
          Afrx.Script.Variables[
          Copy(TfrxMemoView(Afrx.Components[i]).Memo.Text,
            Pos('[', TfrxMemoView(Afrx.Components[i]).Memo.Text) + 1,
            Pos(']', TfrxMemoView(Afrx.Components[i]).Memo.Text) -
            (Pos('[', TfrxMemoView(Afrx.Components[i]).Memo.Text) + 1) )] := 0;
        end;
      end;
    end;
  end;
end;
```

Initvariablesﾒｿｯﾄﾞを作成し、宣言した変数（[VALUE001]等）を初期化する

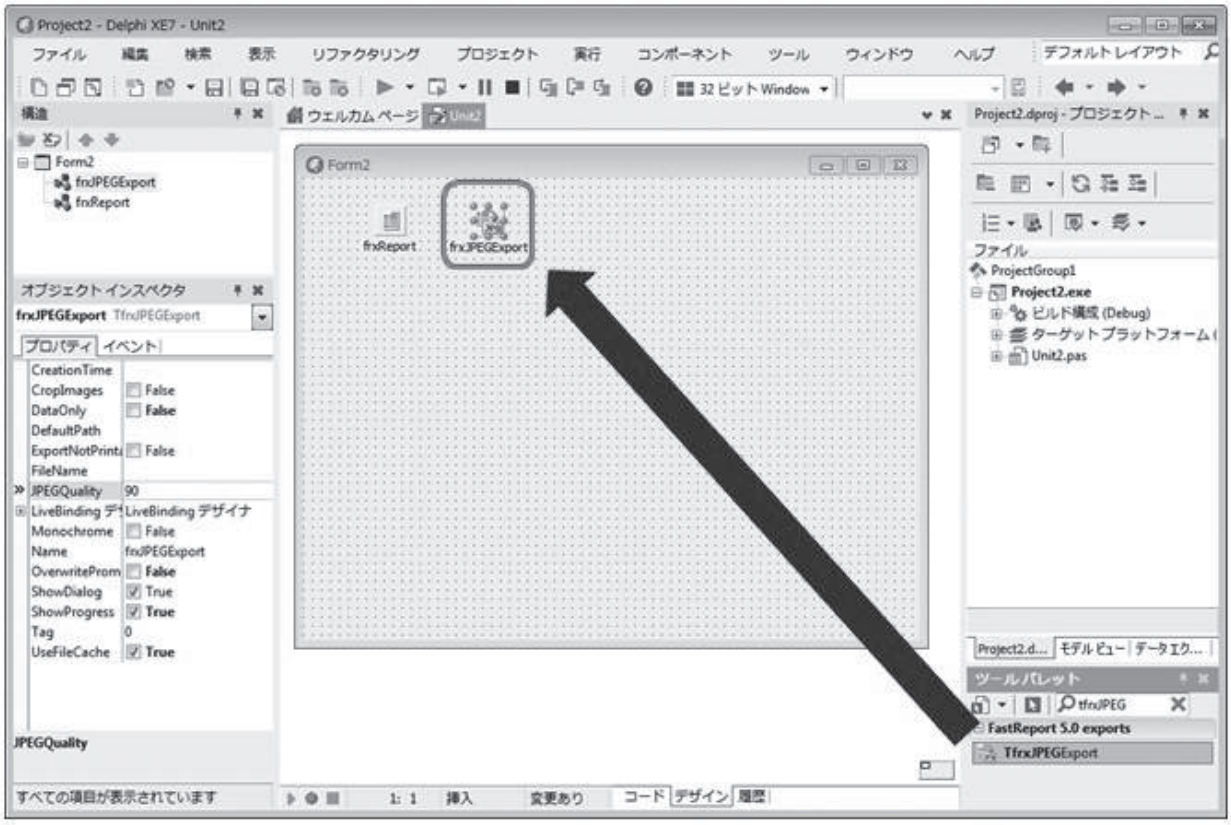
ソース2

```
{ ***** データを帳票にセット ***** }
//<ヘッダー>
frxReport1.Script.Variables['VALUE001'] := edtSinseibi.Text;    //申請日
frxReport1.Script.Variables['VALUE002'] := edtKonyuYotei.Text; //購入予定先
frxReport1.Script.Variables['VALUE003'] := edtSyozoku.Text;     //所属
frxReport1.Script.Variables['VALUE004'] := edtName.Text;        //氏名
```

各値は画面に配置したEditから取得する

変数に値をセットする

図8



TfrxJPEGExportコンポーネントを貼り付ける

ソース3

```
// JPEG変換用コンポーネントのプロパティを設定する
frxJPEGExport.Resolution    := 96;     // 解像度(dpi)
```

TJPEGExportコンポーネントのプロパティを設定

ソース4

```
{*******************************************************************************
 目的：購入申請書発行ボタン押下時
 引数：
 戻値：
*******************************************************************************}
procedure TfrmReportSample002.BitBtn5Click(Sender: TObject);
var
  sImagePath: String;      // 画像パス用
  msJPEG: TMemoryStream;   // JPEG作成用メモリストリーム
begin

  //帳票保存先フォルダの設定
  sImagePath := 'C:¥TechnicalReport¥Report¥';

  // 変数の初期化
  InitVariables(frxReport1);

  { ***** データを帳票にセット ***** }
  //<ヘッダー>
  frxReport1.Script.Variables['VALUE001'] := edtSinseibi.Text;    //申請日
  frxReport1.Script.Variables['VALUE002'] := edtKonyuYotei.Text; //購入予定先
  frxReport1.Script.Variables['VALUE003'] := edtSyozoku.Text;     //所属
  frxReport1.Script.Variables['VALUE004'] := edtName.Text;        //氏名

  //<フッター>
  frxReport1.Script.Variables['VALUE100'] := edtBiko1.Text;       //備考１行目
  frxReport1.Script.Variables['VALUE101'] := edtBiko2.Text;       //備考２行目
  frxReport1.Script.Variables['VALUE102'] := edtBiko3.Text;       //備考３行目
  frxReport1.Script.Variables['VALUE103'] := edtBiko4.Text;       //備考４行目

  { ***** 帳票画像作成 ***** }
  frxReport1.PrepareReport(True); // レポートに区切りを入れる
  // 保存先ディレクトリが存在しない場合、作成する
  if not(DirectoryExists(sImagePath)) then
  begin
    ForceDirectories(sImagePath);
  end;

  //帳票の画像保存メソッドの呼び出し
  try
    // JPEG作成用メモリストリーム作成
    msJPEG  := TMemoryStream.Create;
    try
      // JPEG変換用コンポーネントのプロパティを設定する
      frxJPEGExport1.Resolution    := 75;       // 解像度(dpi)

      // JPGに変換
      frxJPEGExport1.Stream := msJPEG;
      frxReport1.Export(frxJPEGExport1);

      // 保存
      msJPEG.SaveToFile(sImagePath+'SampleReport.jpg');

      // 読み込みボタン押下時用に保存先を内部変数に記憶
      sIMAGE := sImagePath+'SampleReport.jpg';

    finally
      // コンポーネントの解放
      FreeAndNil(msJPEG);
    end;
  except
    // 例外処理
    on E: Exception do
    begin
      raise;
    end;
  end;

  //データ印押印画面呼出
  frmReportSample001 := TfrmReportSample001.Create(Self);
  try
    frmReportSample001.FIMAGEPATH := sIMAGE;  // 帳票画像のパスを渡す
    frmReportSample001.ShowModal;             // データ印押印画面を起動する
  finally
    frmReportSample001.Release;
  end;


end;
```

InitVariables詳細は
【ソース2】参照

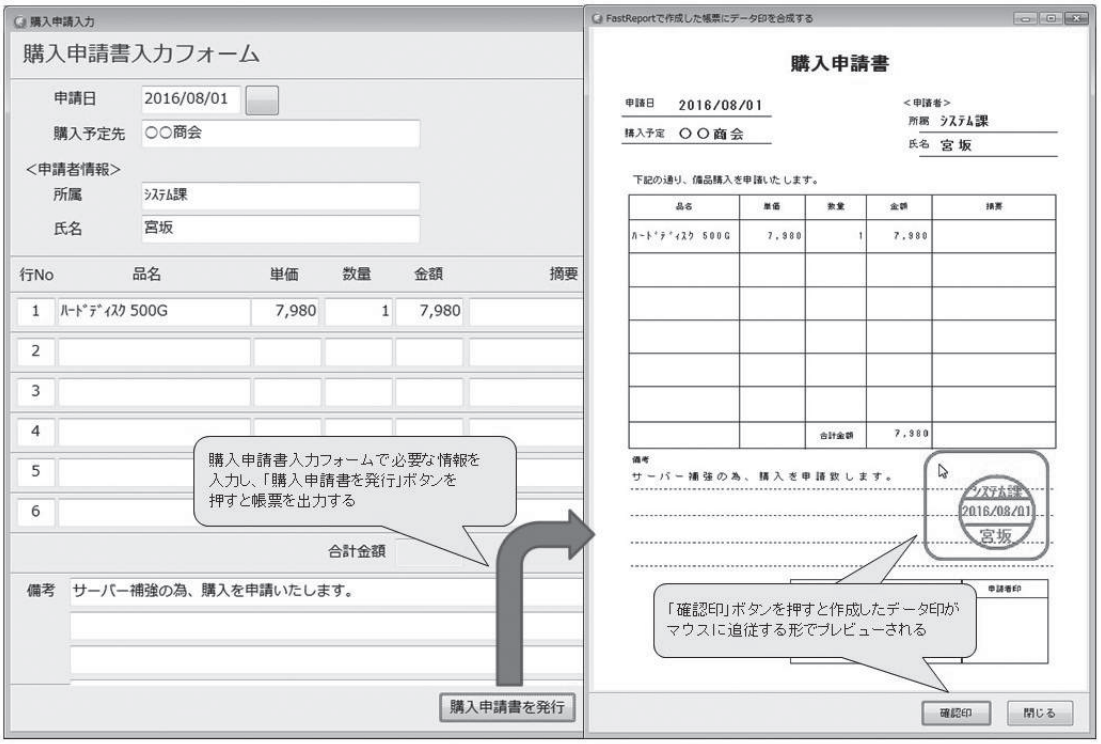
明細のデータセットについては省略

①

②

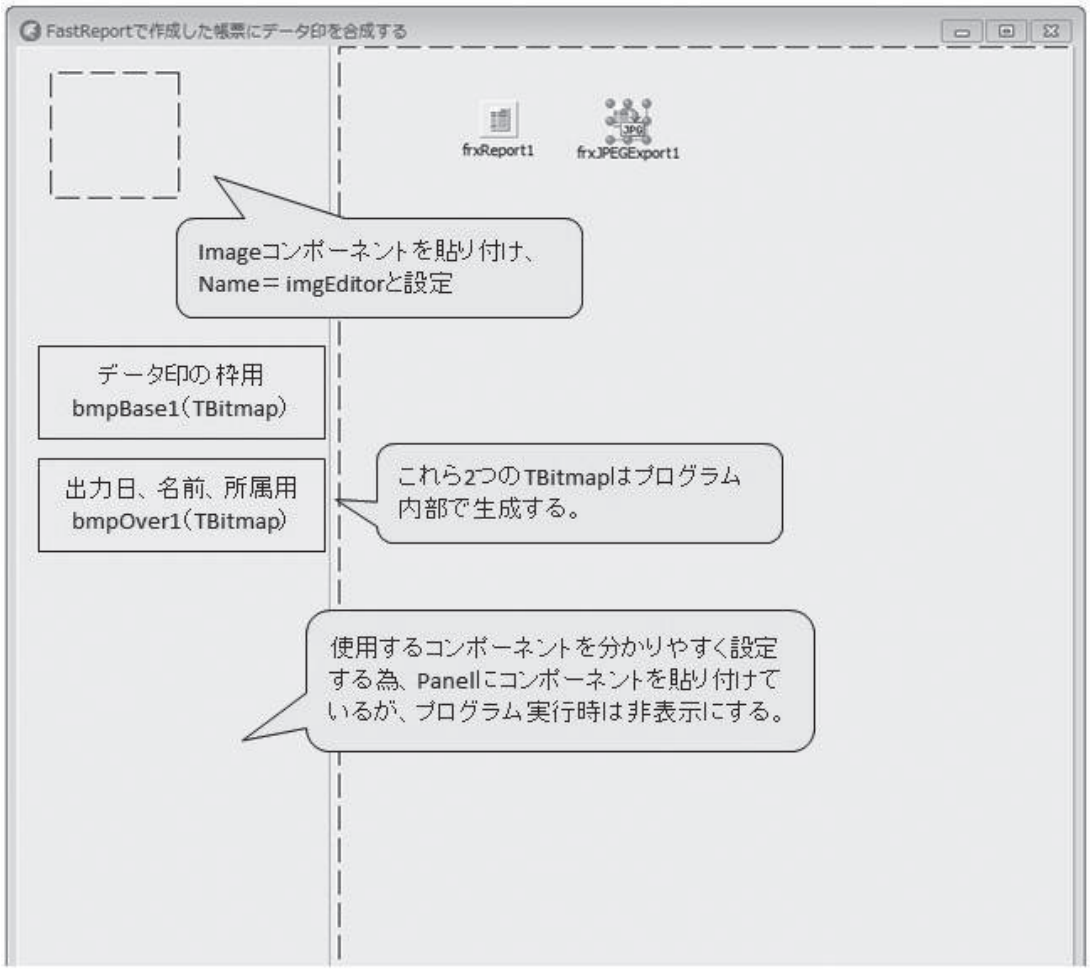
帳票画像ファイルを作成したら
データ印押印画面を呼び出す

図9



購入申請書入力フォーム

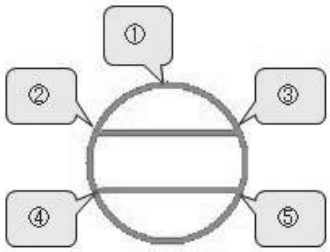
図10



購入申請書表示用子画面のコンポーネント配置

ソース5

```
bmpBase1.Canvas.Pen.Color := clRed;          // 枠線の色
bmpBase1.Canvas.Pen.Width := 4;              // 枠線の太さ
bmpBase1.Canvas.Ellipse(3, 3, 100, 100); // ①円のサイズ
bmpBase1.Canvas.MoveTo(10, 33);              // ②上の横線開始位置
bmpBase1.Canvas.LineTo(95, 33);              // ③上の横線終了位置
bmpBase1.Canvas.MoveTo(10, 68);              // ④下の横線開始位置
bmpBase1.Canvas.LineTo(95, 68);              // ⑤下の横線終了位置
```



データ印の枠(bmpBase1)の設定

図11

出力文字(bmpOver1)の設定

ソース6

```
//所属部署部分
//sSyozoku＝購入申請書入力フォームで入力した所属
iWidth := Length(AnsiString(sSyozoku));

if (iWidth <= 6) then
  bmpOver1.Canvas.Font.Size := 15
else
  bmpOver1.Canvas.Font.Size := 11;

bmpOver1.Canvas.Font.Color := clRed;
bmpOver1.Canvas.Font.Style := [fsBold];
case iWidth of // 文字列の長さによって位置を決定
  1: bmpOver1.Canvas.TextOut(48, 15, sSyozoku);
  2: bmpOver1.Canvas.TextOut(42, 15, sSyozoku);
  3: bmpOver1.Canvas.TextOut(36, 15, sSyozoku);
  4: bmpOver1.Canvas.TextOut(31, 15, sSyozoku);
  5: bmpOver1.Canvas.TextOut(25, 15, sSyozoku);
  6: bmpOver1.Canvas.TextOut(20, 15, sSyozoku);
  7: bmpOver1.Canvas.TextOut(14, 15, sSyozoku);
  else
    bmpOver1.Canvas.TextOut(10, 15, sSyozoku);
end;
```

文字列が1～6文字以下なら、FontSize＝15
文字列が7文字以上なら、FontSize＝11

文字列の長さによってフォントサイズを変更する(所属部分)

ソース7

```
//日付部分
//sToday＝購入申請書入力フォームで入力した申請日
bmpOver1.Canvas.Font.Size   := 12;
bmpOver1.Canvas.Font.Color  := clRed;
bmpOver1.Canvas.TextOut(7, 42, sToday);
```

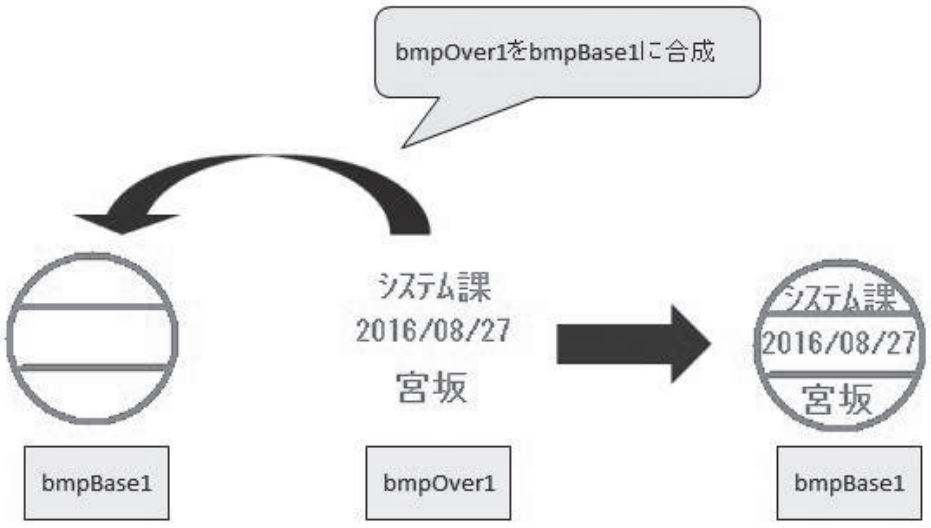
日付部分の設定

ソース8

```
//名前部分
//sMyName＝購入申請書入力フォームで入力した名前
iWidth := Length(AnsiString(sMyName));

if (iWidth <= 6) then
  bmpOver1.Canvas.Font.Size := 15
else
  bmpOver1.Canvas.Font.Size := 11;

bmpOver1.Canvas.Font.Color  := clRed;
bmpOver1.Canvas.Font.Style  := [fsBold];
case iWidth of // 文字列の長さによって位置を決定
  1: bmpOver1.Canvas.TextOut(63, 73, sMyName);
  2: bmpOver1.Canvas.TextOut(51, 73, sMyName);
  3: bmpOver1.Canvas.TextOut(39, 73, sMyName);
  4: bmpOver1.Canvas.TextOut(29, 73, sMyName);
  5: bmpOver1.Canvas.TextOut(19, 73, sMyName);
  6: bmpOver1.Canvas.TextOut(9,  73, sMyName);
  7: bmpOver1.Canvas.TextOut(5,  73, sMyName);
  else
    bmpOver1.Canvas.TextOut(-3, 73, sMyName);
end;
```

名前部分の設定

図12



データ印の合成イメージ

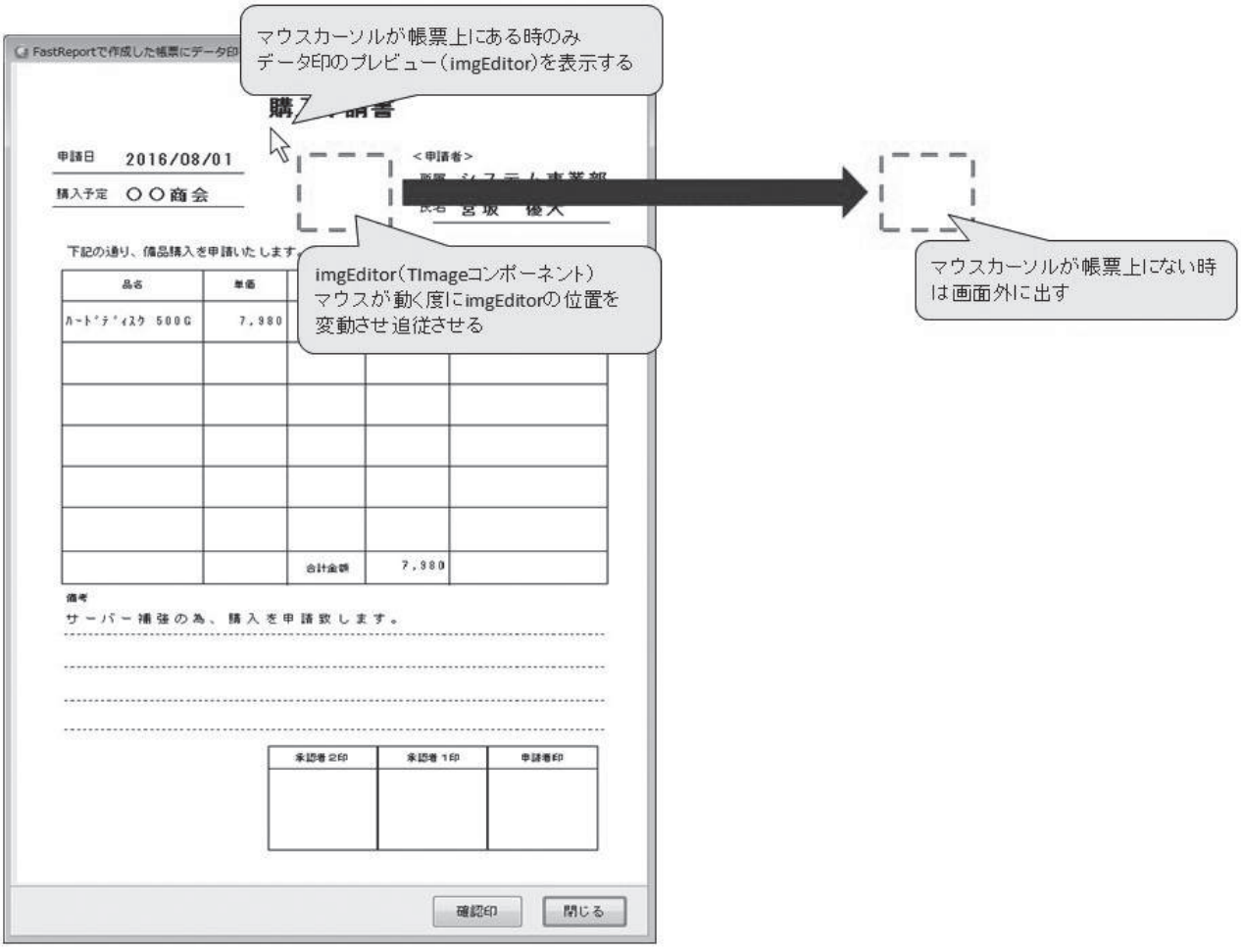
ソース9

StretchDraw関数を使用して
bmpOver1をbmpBase1に貼り付ける

```
//データ印の枠と文字列を融合する
bmpBase1.Canvas.StretchDraw(Rect(0, 0, bmpOver1.Width, bmpOver1.Height), bmpOver1);
```

画像の合成

図13



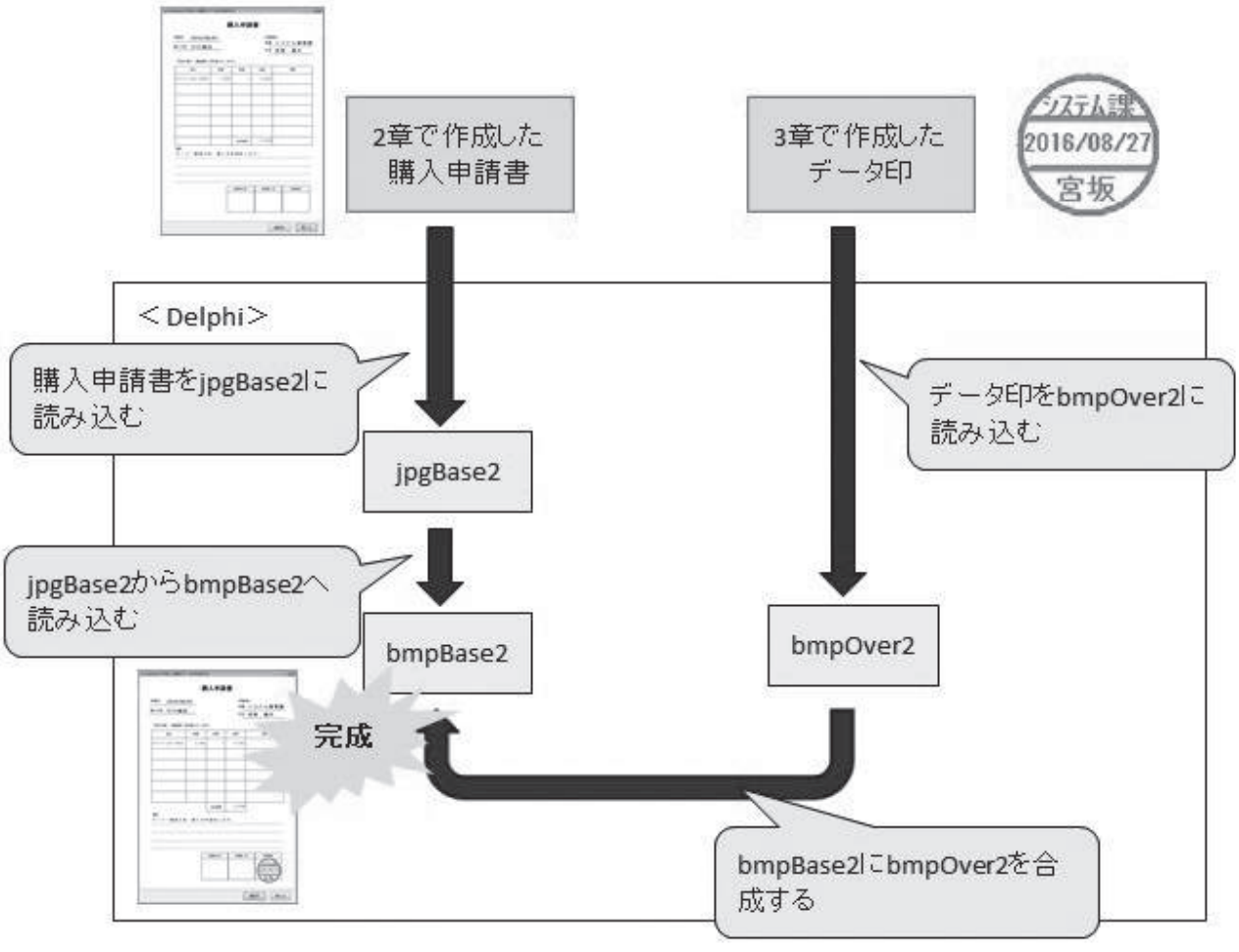
マウスカーソルに追従するプレビューイメージ

ソース10

```
{*************************************************************************************
 目的：画像マウス移動中処理
 引数：
 戻値：
*************************************************************************************}
procedure TfrmReportSample001.Image1MouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
  // bDateIn：プログラム内部データ印を作成したかどうかをBoolean型で保持
  if bDateIn then
  begin
    //データ印を作成した場合のみマウスに追従するImageコンポーネントを表示する
    imgEditor.Visible := True;

    //マウス移動時に追従するデータ印の位置調整
    imgEditor.Left    := X + 10;
    imgEditor.Top     := Y + 10;
  end;

end;

{*************************************************************************************
 目的：画像マウスフォーカス喪失時処理
 引数：
 戻値：
*************************************************************************************}
procedure TfrmReportSample001.Image1MouseLeave(Sender: TObject);
begin
  imgEditor.Visible := False;
end;
```

MouseMove、MouseLeaveイベント

図14



帳票とデータ印の画像合成イメージ

ソース11

```
// ベース画像の型を判別
if (Image1.Picture.Graphic is TJPEGImage) then
begin
  // ベース画像がJPEGの場合、一旦JPEG変数に読み込ませた上でbmpBase2に転送する
  jpgBase2.Assign(Image1.Picture.Graphic);
  bmpBase2.Assign(jpgBase2);
end
else
if (Image1.Picture.Graphic is TBitmap) then
begin
  // ベース画像がBitmapの場合、そのまま転送する
  bmpBase2.Assign(Image1.Picture.Graphic);
end;
```

帳票画像をjmpBase2を経由し、bmpBase2に読み込む

ソース12

```
// 上書きする画像を読み込む
bmpOver2.Assign(imgEditor.Picture.Graphic);
```

データ印画像をbmpOver2に読み込む

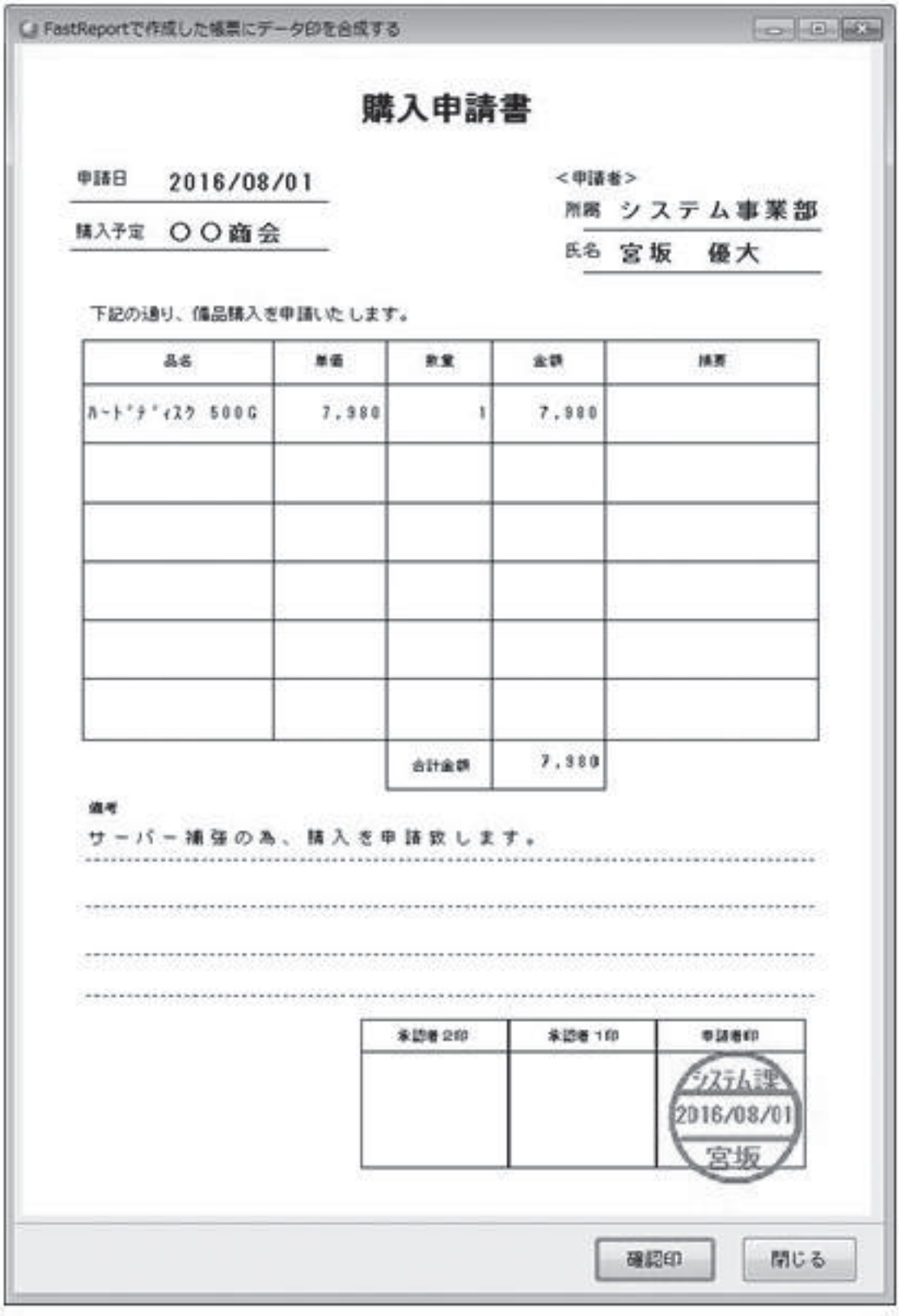
ソース13

```
// ベース画像に上書き画像を描画する
bmpBase2.Canvas.StretchDraw(Rect(X, Y, X+99, Y+102), bmpOver2);

// 合成したbmpBase2をImage1(購入申請書画像)に読み込む
Image1.Picture.Assign(bmpBase2);
```

マウスクリック位置と画像貼り付けサイズをRectの第3引数と第4引数で調整する。

帳票画像(bmpBase2)にデータ印画像(bmpOver2)を合成する

図15



完成した購入申請書

ソース14

```
{*****************************************************************************
 目的：画像マウス押下時処理
 引数：
 戻値：
*****************************************************************************}
procedure TfrmReportSample001.Image1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var
  sImagePath: String;      // 画像パス用
  bmpBase2: TBitmap;       // ベースとなる画像
  jpgBase2: TJPEGImage;    // ベース画像をjpg形式で保管
  bmpOver2: TBitmap;       // ベースに上書きする画像
begin
  // データ印を画像に合成する
//  ImageFusionEx(Image1, imgEditor, iHX, iHY);
  bmpBase2 := TBitmap.Create;      // ベース画像(Btimap)
  jpgBase2 := TJPEGImage.Create;   // ベース画像(JPEG)
  bmpOver2 := TBitmap.Create;      // 上書きする画像(Bitmap)
  try
    // ベース画像の縦・横幅を設定
    bmpBase2.Width  := Image1.Picture.Width;
    bmpBase2.Height := Image1.Picture.Height;
    // 上書きする画像の縦・横幅を設定
    bmpOver2.Width  := imgEditor.Picture.Width;
    bmpOver2.Height := imgEditor.Picture.Height;

    // ベース画像の型を判別
    if (Image1.Picture.Graphic is TJPEGImage) then
    begin
      // ベース画像がJPEGの場合、一旦JPEG変数に読み込ませbmpBase2に転送する
      jpgBase2.Assign(Image1.Picture.Graphic);
      bmpBase2.Assign(jpgBase2);
    end
    else
    if (Image1.Picture.Graphic is TBitmap) then
    begin
      // ベース画像がBitmapの場合、そのまま転送する
      bmpBase2.Assign(Image1.Picture.Graphic);
    end;

    //フルカラーを指定
    bmpBase2.PixelFormat := pf24bit;
    bmpOver2.PixelFormat := pf24bit;
    // CanvasにStretchDrawで描画する
    bmpBase2.Canvas.StretchDraw
      (Rect(0, 0, (bmpBase2.Width-1), (bmpBase2.Height-1)), bmpBase2);

    // 上書きする画像を読み込む
    bmpOver2.Assign(imgEditor.Picture.Graphic);

    // ベース画像に上書き画像を描画する
    bmpBase2.Canvas.StretchDraw(Rect(X, Y, X+99, Y+102), bmpOver2);

    // 合成したbmpBase2をImage1(購入申請書画像)に読み込む
    Image1.Picture.Assign(bmpBase2);

  finally
    // コンポーネントの解放
    FreeAndNil(bmpOver2);
    FreeAndNil(bmpBase2);
    FreeAndNil(jpgBase2);
  end;

  //帳票保存先フォルダの設定
  sImagePath := 'C:¥TechnicalReport¥Report¥';

  { ***** 帳票画像作成 印刷用 ***** }
  // 保存先ディレクトリが存在しない場合、作成する
  if not(DirectoryExists(sImagePath)) then
  begin
    ForceDirectories(sImagePath);
  end;

  Image1.Picture.SaveToFile(sImagePath+'Tyohyo.jpg');

end;
```

吉原 泰介

株式会社ミガロ.

RAD事業部 技術支援課

[Delphi/400] Beacon技術によるIoT活用の第一歩

略歴
1978年3月26日生まれ
2001年 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やJC/400の製品試験、および月100件に及ぶ問い合わ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

最近よく耳にするIoT（Internet of Things）とは、モノとインターネットをつなげて、データ収集、情報発信、自動的な運用動作などに活用する技術・考え方である。

PCやスマートデバイスであれば、もちろんそれ自体がネットワークを通じて情報を発信できるが、IoTではこれまでネットワークに接続していなかったモノこそが、重要な対象になる。【図1】

たとえば最近では、リストバンドで脈拍や移動距離などがネットワークで連携され、健康管理をスマートフォンで行えるようなIoTソリューションが増えてきた。

また車がネットワークにつながることで走行履歴、ガソリンの量、渋滞・災害情報などをリアルタイムに把握し、その状況に応じて最適なルートをナビゲーションするアプリケーションも研究されている（IoTによるカーナビの進化）。

IoTは定義としては非常に広いが、こうした技術や考え方が基盤となっている。そこでは、さまざまに電子化が進むなか、モノがどのように情報を発信するかが重要な鍵になる。

情報を発信する機器の1つに、Beaconがある。たとえば店舗に行くと、タイムセール情報や割引クーポンの情報が受信できるスマートフォンのアプリケーションが増えてきた。そうしたサービスの多くで、Beaconが使用されている。

IoTイコールBeaconというわけではないが、場所に特化したIoT技術として、最近とくに注目されている。

こうした状況を背景に、本稿ではIoTで活用されているBeaconの技術を題材とする。Beaconの基本的な情報から、IoTプログラムとして活用するため、Delphi/400による実装方法などについて説明する（本稿ではBeaconに対応したコンポーネントの使用が中心となるので、Delphi/400のバージョンは10 Seattleを対象にする）。

2. IoTとBeacon技術

2-1. Beaconとは

Beaconは、Bluetoothの信号を発信する機器である。その信号は数秒に1回、半径数十メートル範囲に発信される。ただし普通のBluetoothだとすぐに電力を使い果たしてしまうので、BeaconではBluetooth Low Energy（BLE）という、極力電力を使わない規格を使用している。【図2】

Beaconを使ったシステムでは、スマートフォンなどにインストールした専用アプリケーションでその信号を受信し、それをトリガーにして処理する（プログラムが動作する）仕組みとなっている。

Beaconを使う前提は、受け手が対応するアプリケーションをもっていて、BluetoothをONにするだけである。特別な操作は必要なく、信号に自動で反応するシンプルさがBeaconの魅力である。

次に、こうしたBeaconを使った活用

図1

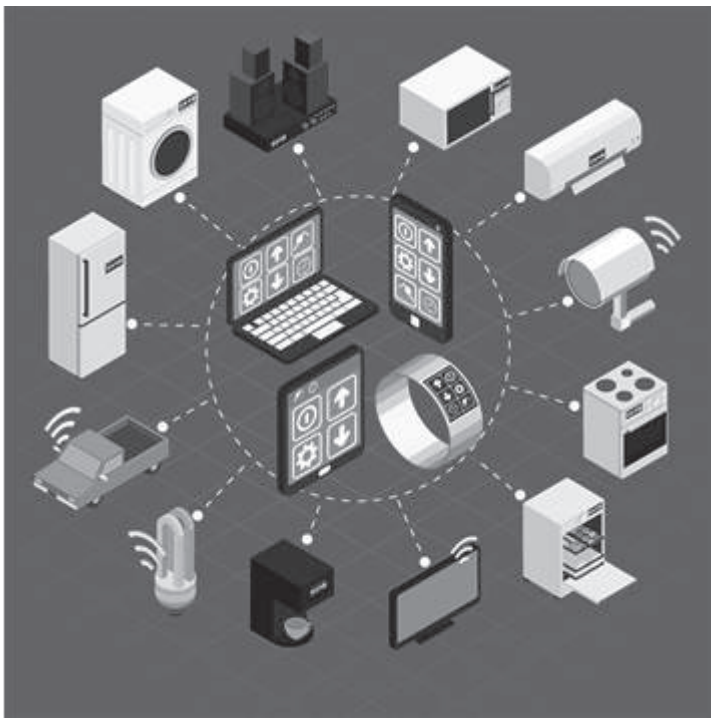


図2

Bluetooth™ 4.0 Low Energy

図3

出典：STOREBEACON iBeaconとは？

事例について考えてみる。

2-2. Beaconの活用事例

一般にBeaconを使う場合、店舗や工場内に信号を発信するBeacon機器を設置する。Beaconの活用用途としては、主に2つが考えられる。1つは情報の「通知サービス」、もう1つは施設内における高精度な「位置サービス」である。

たとえば店舗を訪れた顧客が専用アプリケーションをインストールしていれば、店舗に設置したBeacon機器の信号により、クーポンなどの特典を受信できるサービスが実現可能である。【図3】

Beacon機器から信号を受信したタイミングで、サーバーから情報を送ったり、受けたりできる。これが、「通知サービス」である。こうしたアプリケーションは、そのサーバーと連携できるので、BeaconとインターネットをIoTとしての活用につながる。【図4】

情報を受け取るだけではなく、たとえばオフィスでの活用例としては、社内にBeacon端末を設置し、従業員が出社すると自動的に出勤時間を打刻するといった勤怠管理の仕組みにも応用されている。

最近では、いつも買い置きする特定商品についてBeaconを利用することで、スマートフォンのアプリを経由して、自動的にショッピングサイトに自動注文できるといった仕組みも開発され始めている。

また、配置されたBeacon機器の信号をアプリケーションで受け取るということは、その機器の近くで行動していると推定できる。これを応用した技術が、「位置サービス」である。【図5】

位置サービスでは、後述するように、複数のBeaconで場所を特定していく必要があるが、これによって店舗のどの商品がある場所にいるか、どういった順番で場所を移動しているかなど、行動情報を蓄積してマーケティング分析にも利用されている。

屋内での位置サービスをビジネスに活用する例としては、大規模なショッピングモールでの道案内をはじめ、工場や倉庫など大きな施設内で作業工程上の動線としても利用されている。

また美術館や公共施設の展示物の前にBeacon機器を設置し、利用者が近付くと所持しているスマートフォンにその情報を通知するといった用途でも使われている。実際に宮崎県立西都原考古博物館では、Delphiで構築されたBeaconの館内案内システムが運用されている。

2-3. Beaconと類似する技術

Beaconのように信号を受信したり、位置情報を扱う技術はほかにも存在する。ここからは通知サービスと位置サービスを例にして、類似した技術と比較しながらBeaconの特徴を考えてみる。【図6】

まず通知サービスのように、近くで信号を受信することでアプリケーションを動作させる技術としては、NFC（Near Field Communication）が類似している。

NFCは、超近距離無線通信機能により通信する技術である。主に電子カードの決済などに使われており、技術としては似ているが、用途はBeaconと異なる。

BeaconとNFCの用途の違いとしては、距離と動作対象数が挙げられる。Beaconは十数メートルの範囲で機能するのに対して、NFCは接触に近い数センチの距離で機能する。そのため、Beaconはざっくりとした広範囲動作を得意とするが、NFCは近距離で精度の高い動作を得意とする。

また、Beaconは機器とアプリケーションが1対Nで動作するのに対して、NFCは1対1で動作させる。NFCが活用されているSuicaなど、交通カードを使った改札精算をイメージするとわかりやすい。

そのため、場所を起点とした一括情報サービスではBeacon、電子決済など個々の情報制御サービスではNFCが使われるといった違いがある。

また位置サービスの技術として広く活用されているものとして、GPSがある。「位置サービス＝GPS技術」と連想される場合も多い。しかしBeaconとGPSでは、得意とする分野が大きく異なる。

Beaconが屋内の精密な位置情報検知を得意とするのに対し、GPSは屋外での大まかな位置情報検知を得意とする。

またGPSは機器などの設備が不要な半面、建築物などで電波が精密に受信できないため、屋内での位置情報検知には活用できないことが多い。

一方Beaconは機器の配置を前提とするが、屋内で独自の位置を算出できるので、施設などでの精密な位置サービスに向いている。

次に、こうしたBeacon技術をDelphi/400により実装して活用する方法を検証していく。

3.Beaconを活用するプログラム開発

3-1. Beaconがもつ信号情報

まずBeaconをプログラムで扱ううえで、Beaconがどのような信号を発信しているかについて説明する。

Beacon機器が発信する信号の識別にはUUID、Major、Minorの情報が含まれている。これはBeacon機器に設定するもので（設定方法は機器によって異なる）、任意に指定できる。

一般には、下記のようなルールで設定することが多い。

- UUID：施設レベルで一意
- Major：フロア／エリアレベルで一意
- Minor：施設、フロア内での機器として一意

もちろん任意のルールで設定もできるが、こうした体系を組んだ指定のほうが、システムで組み込む際にどの施設のどのフロア（あるいはエリア）の機器かをわかりやすく管理できる。

3-2. Beaconで使用するコンポーネント

Delphi/400 10 SeattleではBeaconを使うために、TBluetoothLEやTBeaconというコンポーネントが用意されている。

TBeaconではIDなどを設定するだけで、対象のBeacon機器に反応して動作するアプリケーションを簡単に実装できる。

- TBluetoothLE コンポーネント

画面上にドラッグ＆ドロップするだけで、Beaconが使用するBluetoothLEに対応できる

- TBeacon コンポーネント

図4

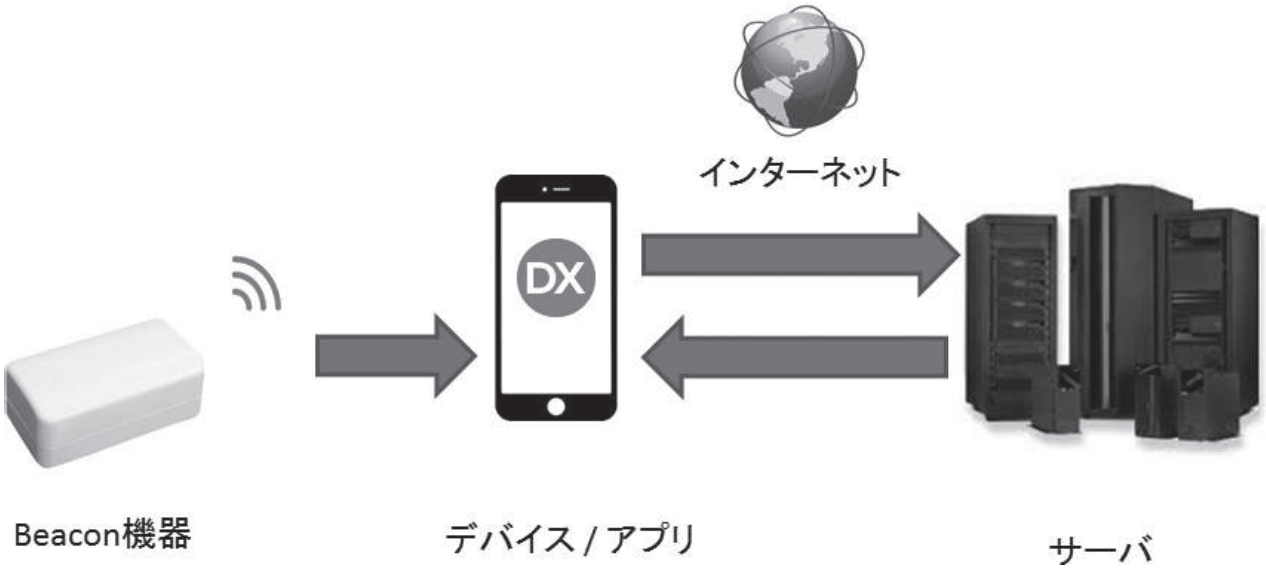
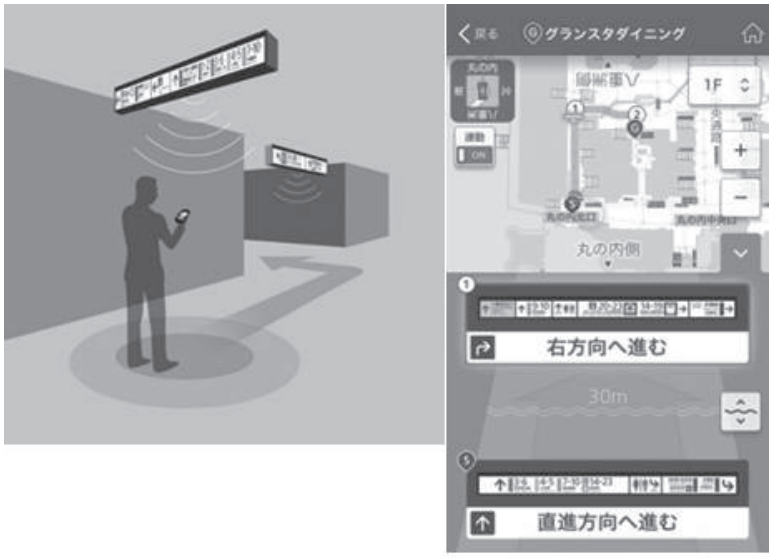


図5



出典：東日本旅客鉄道株式会社
実証実験「東京駅構内ナビ」

図6

Beaconの信号情報を感知・判別して、イベントを処理できる

3-3. Beaconを使ったプログラミング実装

Beaconを使ったプログラムを作成するにはコンポーネントの配置、コンポーネントの設定、そしてプログラミングによる処理作成が必要になる。ここでは、Beaconの信号に反応して広告画像を自動表示する簡単な通知サービス・アプリケーションを題材に、実装方法を説明していく。

まず、FireMonkeyで新規のマルチデバイスアプリケーションを作成し、画面フォーム上にTBeaconとTBluetoothLE、TImageコンポーネントを配置。TImageコンポーネントに、表示したい広告画像を設定しておく。【図7】

TBluetoothLEは貼り付けておくだけでBluetoothLEに対応できるので、特別な設定は必要ない。

TBeaconでは、簡単な設定とプログラミングを実装する。

まず設定内容として、対象とするBeacon機器のUUIDやMajor、Minorの値をプロパティに指定する（IDを機器に設定していない場合は、3-1.を参考）。TBeaconはアイテムとしてmonitorizedRegionsオブジェクトをもつので、このアイテムのMajor、Minor、UUIDプロパティで非常にわかりやすく設定できる。【図8】

IDのデフォルト値は、-1や0で設定される。Major、Minorの-1はすべてを対象にするので、別のエリアで同じUUID機器がある場合は注意が必要である。

またDataSnapサーバーと通信する場合はTSQLConnection、TDSProviderConnection、TClientDataSetなどのコンポーネントを配置・設定しておく（本稿ではDataSnapサーバーとの通信については割愛する）。

次に、Beacon機器の信号を受信した際のプログラム動作を実装する。TBeaconコンポーネントの基本的なイベントはOnEnter、OnExitである。このイベントで信号の受信成否を判定して、プログラミングが行える。

たとえばOnEnterイベントでTImageコンポーネントのVisibleをTrueにすれば、信号を受信して広告画像をアプリケーションで表示する。また同時にサーバーとのデータ連携が必要であれば、TClientDataSetを使ってサーバーへデータを登録するなど、IoTとしてシステムを連携できる。【ソース1】

信号を受信しなくなれば、OnExitイベントでVisibleをFalseにすることで広告画像を非表示にできる。【ソース2】

またアプリケーションの初期処理としてBeacon機能を有効にし、画像は非表示にしておく処理を、フォームのOnCreateイベントで作成しておく。【ソース3】

このプログラムをiOSやAndroidなどのスマートデバイスに対してコンパイルすると、アプリケーションは完成である。アプリケーションを実行して対象のBeacon信号を受信すると、広告画像が自動で表示される。【図9】

ここまで画像の表示・非表示の単純な実装方法を説明したが、同じイベントでサーバーに対して処理を行えば、Beacon機器の位置を起点にしてシステムを動かせる。

たとえば、Beacon機器を使った勤怠管理システムがこうした単純な仕組みで動作している。画像の表示の代わりに、受信時刻で入室データをDBへ登録しているのである。

同様に、工場の生産工程で作業場所に応じてチェック処理を自動的に起動したり、倉庫の商品棚管理では位置を自動登録するなど、場所を起点としたIoTシステムで活用されている。

また単純なBeacon信号の受信有無だけではなく、その信号電波の強弱によって機器までの近接度を判別することもできる。

近接度（ABeacon.Procimity）は、次の4段階で判定する。

- Immediate：0.5m未満
- Near：0.5m以上、1.5m以下
- Far：1.5mより遠い
- Away：距離判定不能

近接度をOnEnterイベントで判定する方法を【ソース4】に示すので、参考にしてほしい。このプログラムでは、近接度のメッセージを表示するように実装している。

アプリケーションを実行すると、【図10】のように近接度に応じたメッセージを受け取れる。

OnEnterイベントは信号エリアに入ったときのみ処理されるが、継続的にBeacon信号を処理したい場合には、OnBeaconProximityイベントを使うとハンドリングが可能である。

3-4. Beaconを使った位置検出「BeaconFance」

次に、Beaconを使った「位置サービス」について考えてみる。前節で大まかな近接度判定について解説したが、位置を検出するにはさらに高度な算出が必要になる。

アプリケーションが動作するデバイスとBeacon機器までの距離を算出する場合、Beaconの出力電波強度（TxPower）と受信時の電波強度（RSSI）を、所定の公式（フリスの伝達公式）で次のように計算する。

＜Beaconとの距離の計算＞
距離＝10^((TxPower − RSSI) / 20)

通常、Beaconの位置情報アプリではこうした計算をプログラムで複雑に組む必要があるが、Delphi/400ではSystem.Beacon.IBEACON.Distanceにメートル単位での距離を自動算出する機能があるので、簡単に距離を割り出せる。

これによってBeacon機器との距離は扱えるが、位置という意味では正確ではない。この距離は、Beacon機器を中心に円形状に発信された距離であり、方向までは特定できない。【図11】

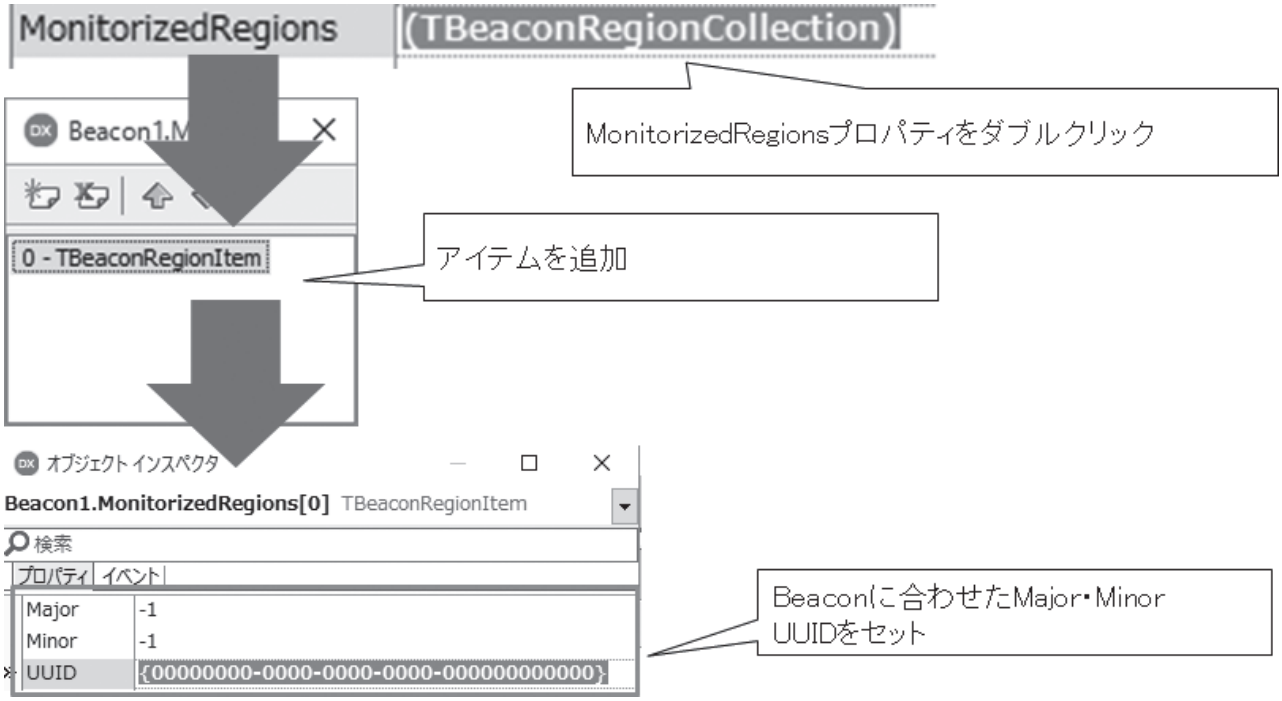
そのため位置測位に関しては、必ず3個以上のBeacon機器からの電波を計算する三角測量技術を使うことになる。

三角測量については算出法が複雑なので、本稿では割愛するが、複数のBeacon機器からの距離を加味した演算を繰り返し行う必要がある。

この位置測位をDelphi/400で簡単に解決するソリューションとして、Delphi開発元のエンバカデロ・テクノロジーズ社が販売する「BeaconFence」というサードパーティ製品がある。コンポーネントとして提供されているが、これを使用すると、こうした位置測位演算ロジックをプログラミングしなくても、ビジュ

図7

図8



ソース1

OnBeaconEnterイベント(Beaconエリアに入った処理)

```
procedure TForm1.Beacon1BeaconEnter(const Sender: TObject;
  const ABeacon: IBeacon; const CurrentBeaconList: TBeaconList);
begin
  Image1.Visible := True; //画像を表示
  ClientDataSet1.Execute; //サーバにデータを登録
end;
```

アルツール上の設定だけで位置サービスのアプリケーションを開発できる。【図 12】

BeaconFence は標準コンポーネントではないが、10 Seattle の GetIt パッケージマネージャに登録されており、簡単にインストールできる。GetIt パッケージマネージャは、［ツール |GetIt パッケージマネージャ］から起動して、対象製品を選択するだけですぐに利用可能である。【図 13】

ただし無償で使用できるのは、対象 Beacon 機器が 3 個までなので、実際の運用時には有償のライセンスが必要である。

4.Beaconの運用

前項でプログラムでの実装ポイントを考察したが、実際の運用ではアプリケーションだけではなく、Beacon 機器自体の知識・利用方法が重要になる。以下では、Beacon 機器の運用上のポイントについて補足する。

4-1. Beacon の運用の注意点

Beacon 機器は主要な規格として iBeacon や AltBeacon があり、Delphi/400 では双方に対応している。

Beacon 機器は信号を発信することを主な機能とするが、プログラムの実装でも触れたように、受信するアプリケーション側は、その信号の強さなどによって距離を判別している。

この信号の強さ（電波）はアプリケーションの動作に大きく影響するので、運用上の考慮点をいくつか把握しておく必要がある。

Beacon 機器の運用で注意すべきポイントは、大きく 3 つある。

1 つ目のポイントは、周波数帯である。Beacon は 2.4GHz の周波数帯を使用する機器であるが、この周波数帯は免許なしで使用できるので、さまざまなデバイスが発している。代表的なものとして電子レンジ、Wi-Fi、一部のデジタルコードレス電話などがある。

Beacon を配置する場所では、こうした機器が使用しているチャネルが重複しないよう配慮する必要がある。どういった周波数が使われているかは施設側に確認するか、あるいは専用アプリケーションなどで測定もできる。テストしたうえで、競合していないことの確認が必要である。

2 つ目のポイントは、モノによる電波の干渉である。たとえば電波が壁にぶつかれば、反射した電波も存在するので、場所によっては強い電波や弱い電波が混在する。また水を多く含む物体（生物を含む）は、電波を吸収しやすい。たとえば Beacon とデバイスの間に人がいると、電波が吸収され、正しく電波を受信できない可能性もある。【図 14】

そのため施設によっては、人が電波の直線上に入らないよう天井などの上に配置することも多い。ただしその場合は、天井からの距離も電波に影響することを考慮する必要がある。

3 つ目のポイントは、電力の確保である。Beacon 機器の電力供給方法には、主に電池型と給電型がある。【図 15】

通常、Beacon の設置は電源が近ければ給電型が便利だが、位置情報を必要とする場所では、電源が確保できない場合も多い（たとえば屋外のイベントなど）。そうしたケースでは、電池型がよく採用されている。

電池は製品にもよるが、大体 1 年ぐらいはもつ。数が多いと残量チェックなどは現実的ではないので、余裕のある定期交換を前提に運用を考えるのが一般的である。

4-2. Beacon のセキュリティ

Beacon のセキュリティ面について、考えてみる。

前述したように、Beacon は UUID、Major、Minor などの情報で識別されるが、その識別情報を知っていれば、意図的に複製できる。

iPhone では、iPhone 自体を Beacon 機器として利用するアプリケーションが AppStore で配布されている。プログラム開発では、そうしたアプリケーションで Beacon のテストが疑似的に行えるメリットがある。しかしこれは同時に、実際の Beacon 機器についても「なりすまし」が可能なことを意味する。

たとえばクーポンの Beacon 信号を複製されると、不正にそのクーポンサービスを利用される危険がある。【図 16】

社内や工場内のシステムなどでは、あまり考慮する必要はないかもしれないが、Beacon で公共に信号を発信する場合には、機密性の高いものは扱わないように注意すべきである。

もしセキュリティが必要な場合は、単純に Beacon 信号を受発信するだけでなく、アプリケーションが動作する条件をシステム的にガードする必要がある。たとえば並行してユーザー情報をチェックしたり、ログ出力で重複利用を防ぐなどの方法もセキュリティ的に有効である。

5.まとめ

本稿では、Beacon 機器の特徴や基本情報の確認、IoT での活用例などを考察してきた。また、そうした Beacon 機器をシステム活用するためのプログラム実装も、Delphi/400 では専用コンポーネントにより非常に簡単であることを説明した（通常は Bluetooth 情報を受け取る部分から、すべて独自のプログラムとして開発する必要がある）。

Beacon も今後はさらに機能が進化し、機器自体も低コスト化していくと思われる。たとえば運用上の考慮点として、電力について説明したが、現在では太陽電池などで、交換電源を必要としない Beacon 機器も開発されている。

この Beacon 技術は、東京駅構内でのナビゲーションサービスの実証実験でも使われており、これからの IoT 技術として大きな期待が寄せられている。【図 17】

Beacon はユーザー操作を必要としないので、浸透すれば多くの施設やシステムで利用できる可能性を秘めている。まだ技術展開が始まったばかりの IoT 分野だが、今までにないユーザーインターフェースやビジネスモデルを考えていくうえで、Beacon は非常に画期的で興味深い。Delphi/400 では簡単に実装できるので、IoT に取り組む第一歩として、システム開発に活用していきたい。

M

ソース2

OnBeaconExitイベント(Beaconエリアから出た処理)

```
procedure TForm1.Beacon1BeaconExit(const Sender: TObject;
  const ABeacon: IBeacon; const CurrentBeaconList: TBeaconList);
begin
  Image1.Visible := False; //画像を非表示
end;
```

ソース3

OnCreateイベント(初期処理)

```
procedure TForm1.FormCreate(Sender: TObject);
Begin
  Image1.Visible := False; //画像を非表示
  Beacon1.Enabled := True; //Beaconを有効化
end;
```

図9

ソース4

OnBeaconEnterイベント(Beaconエリアに入った処理) 応用

```
procedure TForm1.Beacon1BeaconEnter(const Sender: TObject;
  const ABeacon: IBeacon; const CurrentBeaconList: TBeaconList);
begin
  Image1.Visible := True;
  ClientDataSet1.Execute;
  //近接度を判定
  case ABeacon.Proximity of
    //0,5m未満
    TBeaconProximity.Immediate:
    begin
      ShowMessage('近接');
    end;
    //0.5m以上、1.5m以下
    TBeaconProximity.Near:
    begin
      ShowMessage('近い');
    end;
    //1.5mより遠い
    TBeaconProximity.Far:
    begin
      ShowMessage('遠い');
    end;
    //距離判定不能
    TBeaconProximity.Away:
    begin
      ShowMessage('測定不能');
    end;
  end;
end;
```

図10

図11

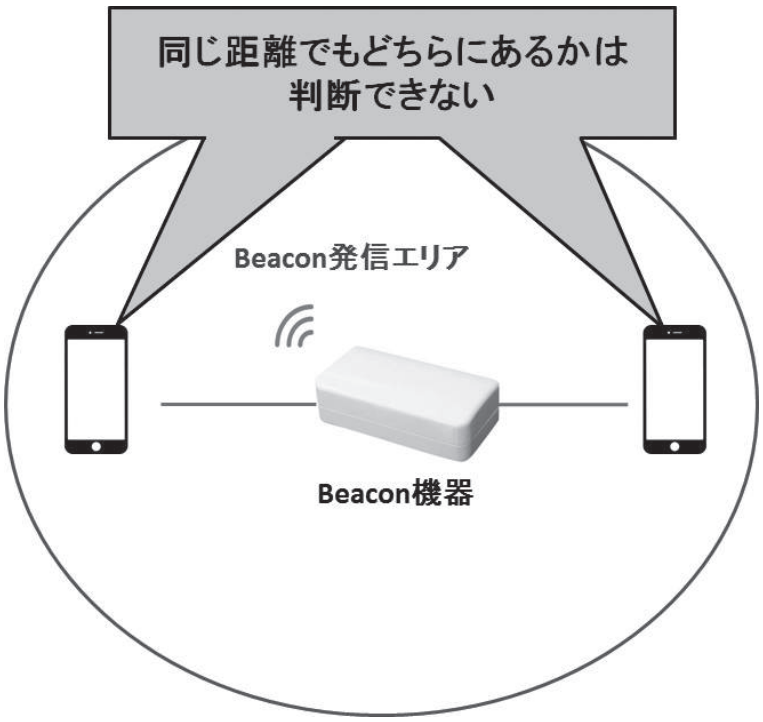
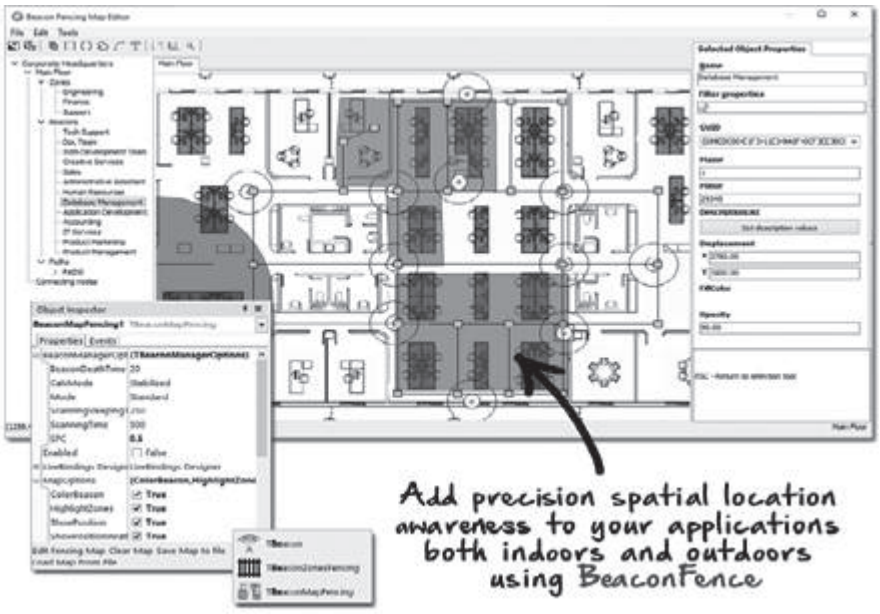


図12



出典：Embarcadero Technologies BeaconFence

図13


GetIt パッケージ マネージャ
検索
絞り込み
すべて
購入可能
無料
購入済み
インストール済み
ソート順
カテゴリ
All
Abbrevia 10.0
TurboPack
インストール
AsyncPro for VCL 1.0
TurboPack
インストール
BeaconFence 1.0
Embarcadero Technologies
今すぐ試す
インストール
完了!
閉じる
選択するだけで
自動インストールや
自動アンインストールが可能

図14


配置によって人やモノが電波を弱めてしまう

図15


電池型


給電型

図16

図17

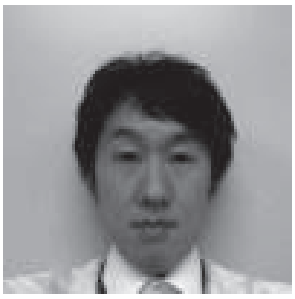
出典：ジャパンスマートナビ：国土交通省

國元 祐二

株式会社ミガロ.

RAD事業部 技術支援課

[SmartPad4i] 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



略歴
1979 年 3 月 27 日生まれ
2002 年 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
JC/400、SmartPad4i、Business4 Mobile 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

プログラム開発において、デバッグ作業は非常に重要である。

デバッグとは、作成したプログラムにバグがないかを確認するテストや、障害が発生した際にプログラムを動かしながら原因を調査する作業を意味する。つまりデバッグに精通していれば、開発時にバグを減らし、障害発生時に問題を早急に解決できる。

SmartPad4i のプログラム開発では、RPG、COBOL などの IBM i プログラムがビジネスロジックの中心となるため、プログラム開発時のデバッグ作業は 5250 エミュレータ上で行える。使い慣れた IBM i プログラム言語を使ってデバッグできるので、バッチジョブのデバッグ手順を知っていれば、開発時に困ることはない。

しかしアプリケーションの運用中に不定期に発生するエラーなど、再現できない障害は、デバッグ作業で調査するのが難しい。そういう場合は、調査のための知識と工夫が必要である。

また SmartPad4i の画面は HTML や CSS で作成するので、JavaScript で機能をカスタマイズすることも多い。そうした JavaScript でのカスタマイズ部分は、IBM i 側ではデバッグできないので、Web ブラウザ側のデバッグ機能を知っていると、開発や調査で非常に役立つ。

このようにデバッグ作業をいろいろな角度で行うほど、開発時のテストや障害解決の精度を上げられる。そのためには、デバッグや使用ツールについて知識を深める必要がある。

本稿では、IBM i 側でのデバッグと Web ブラウザ側のデバッグについて、知っておくと役立つ情報・テクニックを説明する。

2. IBM iでのデバッグ手法

2-1. IBM i プログラムでのデバッグ

SmartPad4i は画面には HTML を、ビジネスロジックには IBM i プログラム（RPG、COBOL など）を使って開発する。デバッグ作業は IBM i 上で行えるが、5250 画面の対話型ジョブではなく、バッチジョブとして動作している。対話型ジョブとは手順の若干違う部分があるので、注意が必要である。

まず、基本的な対話型ジョブのデバッグについて確認する（IBM i プログラムのデバッグではこれが基本となる）。

デバッグ手順は、次のとおりである。

・プログラム実行前にデバッグオプションを設定
・コンパイル
・STRDBG コマンドを実行
・ソースにブレークポイントを設定
・プログラムを動作させてデバッグ作業

これらの手順でポイントになる点を、いくつか補足する。

図1

デバッグオプション

RPG

CRTRPGPGM PGM(ライブラリ名/プログラム名)
SRCFILE(ライブラリ名/ソースファイル名)
SRCMBR(ソースファイルメンバー名)
OPTION(*SRCDBG)

ILERPG

CRTBNDRPG PGM(ライブラリ名/プログラム名)
SRCFILE(ライブラリ名/ソースファイル名)
SRCMBR(ソースファイルメンバー名)
DBGVIEW(*SOURCE)

図2

デバッグオプション

COBOL

CRTCBLPGM PGM(ライブラリ名/プログラム名)
SRCFILE(ライブラリ名/ソースファイル名)
SRCMBR(ソースファイルメンバー名)
OPTION(*SRCDBG)

図3

STRDBGコマンド

STRDBG PGM(ライブラリ名/プログラム名)
UPDPROD(*YES)
OPMSRC(*YES)

コンパイル時のデバッグオプション

RPG/400 のプログラム作成の場合、CRTRPGPGM コマンドでコンパイルを実行時に、ソース・リスト・オプションへ「*SRCDBG」を設定する。ILERPG の場合は、CRTBNDRPG コマンドでコンパイルを実行時、デバッグ用ビューに「*SOURCE」を設定する。【図 1】

COBOL の場合、CRTCBLPGM コマンドでコンパイルを実行時に、ソース・リスト・オプションに *SRCDBG を設定する。【図 2】

STRDBG コマンド

デバッグオプションを設定してコンパイルしたプログラムに対して、STRDBG コマンドを実行する。【図 3】

STRDBG コマンドを実行すると、5250 エミュレータ上でソースが表示されるので、ブレークポイントを設定する行を選択して、F6 キーを押下する。【図 4】

プログラムを実行すると、設定したブレークポイントでプログラムが停止してデバッグ調査が行える。対話型ジョブの RPG や COBOL であれば、この手順だけでデバッグが可能である。

しかし SmartPad4i のアプリケーションは前述したとおり、バッチジョブとして動作するため、次にそのポイントを説明する。

2-2. バッチジョブのデバッグポイント

バッチジョブのデバッグでは、前述したデバッグ手法に加えて、IBM i プログラム実行前に、サービス・ジョブ開始（STRSRVJOB）コマンドを使ってバッチジョブを指定する必要がある。

プログラム実行前には、通常の IBM i プログラムと同様に、デバッグオプションを設定してコンパイルを実行する。

次に、SmartPad4i アプリケーションを実行することで作成されるジョブの「ジョブ」「ユーザー」「番号」を、WRKACTJOB コマンドから取得する（番号はブラウザのタイトルバーに表示される）。【図 5】

ジョブの情報を取得したあと、5250 エミュレータ上で STRSRVJOB コマンド（サービスジョブ開始）を実行する。引数には確認したジョブ、ユーザー、番号を指定する。【図 6】

あとは対話型ジョブのデバッグと同じように、STRDBG コマンド（デバッグ開始）を実行する。【図 3】

5250 エミュレータ上でソースが表示されるので、ブレークポイントを設定する行を選択し、F6 キーを押下する。

ブラウザで SmartPad4i アプリケーションを操作すると、IBM i 側のプログラム処理で停止してデバッグ調査が行える。【図 7】

こうしたデバッグ手法を知っていれば、開発時のプログラム確認で非常に役立つ。ただし問題となる動作を確実に再現・実行できなければ、有効ではない。

たとえばアプリケーションの運用上は稀に発生するが、テストでは再現できない障害の場合は、IBM i プログラム側で定様式ダンプを出力する手法が有効である。

2-3. 定様式ダンプの活用

定様式ダンプとは、IBM i プログラムのフィールドの内容、データ構造の内容、配列やテーブルの内容、ファイル情報のデータ構造、およびプログラム状況のデータ構造を含むファイルである。

IBM i ではあらかじめプログラムに設定しておくと、エラーが発生したときに、定様式ダンプを自動で出力できる。この機能を利用すると、エラーが発生したあとに出力された情報から原因を調査できる。

通常、IBM i プログラムでエラーが発生した場合には、「ダンプを出力する」「終了する」などのメッセージ応答を行える。そのためこの応答を、自動的に「ダンプを出力する」で返すように設定しておく必要がある。

応答の設定は、IBM i のシステム応答リスト項目が有効である。システム応答リストを利用すると、IBM i 側のプログラムでエラーが発生した際に、自動的に応答できる。

応答リストは、【図 8】のコマンドで追加できる。使用する言語によって、設定するコマンドが異なるので、注意が必要である。

ADDRPYLE はシステム応答リスト項目を追加するコマンドで、MSGID に定義されたエラーが発生した際に、RPY で設定した応答メッセージを SEQNBR 順に返す。

これだけでシステム応答リスト項目の設定は完了である。ただし応答するプログラム側にも、システム応答リストを利用するように設定する。

SmartPad4i プログラムが起動時に実行する、SETENV の CL プログラムに自動応答を追加するとわかりやすい。【ソース 1】

以上で、定様式ダンプを自動出力する設定は完了である。

SmartPad4i のプログラムを実行して、IBM i プログラム側でエラーが発生した場合には、エラー発生時のダンプ内容がアウトキューの QEZDEBUG に QPPGMDMP のファイルとして出力される。【図 9】

出力されたダンプファイルを確認することで、再現が難しい現象でも、あとから発生原因を解析できる。特殊ではあるが、デバッグの手法としては、非常に有効なテクニックである。【図 10】

3.ブラウザでのデバッグ手法

3-1. Web やハイブリッドアプリケーションのデバッグ

一般的に Web やハイブリッドアプリケーションの開発では HTML、CSS、JavaScript を利用する。

SmartPad4i でも、ビジネスロジックは IBM i 側のプログラムで動作するが、こうした Web 側のカスタマイズ開発も可能である。

開発した HTML や CSS、JavaScript がどのように動作・表示されるかを確認するには、ブラウザで実際に実行するしかない。ブラウザでの実行は簡単だが、前述した IBM i プログラムのようにブレークポイントを設定して、プログラムコードを追うようなデバッグ調査は行えない。

これまで、Web アプリケーション開発ではこうした点が非常に面倒であったが、最近のブラウザではデバッグ専用機能が実装され、便利になっている。次に、このブラウザ自体のデバッグ機能について説明する。

3-2. ブラウザのデベロッパー・ツール

現在使われているブラウザには HTML や CSS、JavaScript を簡単にデバッグできるツールが搭載されている。

図4

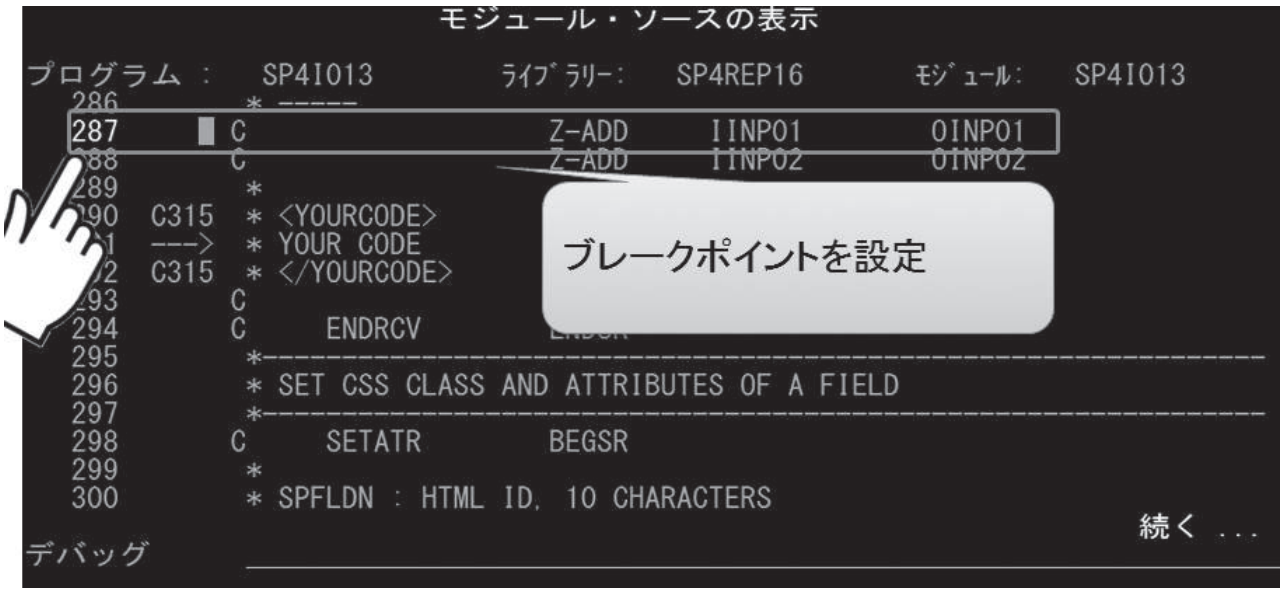
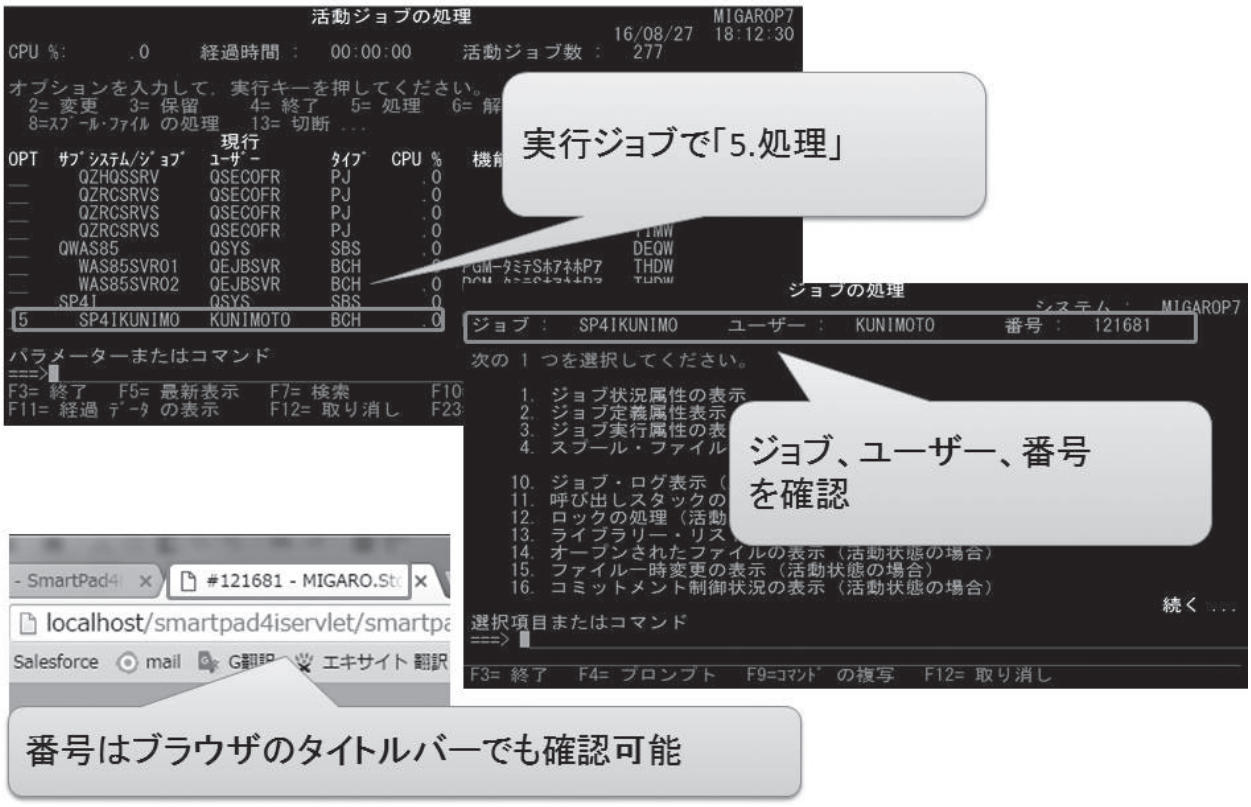

モジュール・ソースの表示
プログラム : SP4I013 ﾗｲﾌﾞﾗﾘｰ: SP4REP16 ﾓｼﾞｭｰﾙ: SP4I013
287 C Z-ADD IINP01 OINP01
C Z-ADD IINP02 OINP02
C315 * <YOURCODE>
---> * YOUR CODE
C315 * </YOURCODE>
C ENDRCV
* SET CSS CLASS AND ATTRIBUTES OF A FIELD
C SETATR BEGSR
* SPFLDN : HTML ID, 10 CHARACTERS
続く ...
デバッグ
ブレークポイントを設定

図5


活動ジョブの処理
MIGAROP7
16/08/27 18:12:30
CPU %: .0 経過時間 : 00:00:00 活動ジョブ数 : 277
オプションを入力して、実行キーを押してください。
2= 変更 3= 保留 4= 終了 5= 処理 6= 解
8=ｽﾌﾟｰﾙ･ﾌｧｲﾙ の処理 13= 切断 ...
現行
OPT サブシステム/ジョブ ユーザー タイプ CPU %
QZHQSSRV QSECOFR PJ .0
QZRCSRVS QSECOFR PJ .0
QZRCSRVS QSECOFR PJ .0
QZRCSRVS QSECOFR PJ .0
QWAS85 QSYS SBS .0
WAS85SVR01 QEJBSVR BCH .0
WAS85SVR02 QEJBSVR BCH .0
SP4I QSYS SBS .0
5 SP4IKUNIMO KUNIMOTO BCH .0
パラメーターまたはコマンド
===>
F3= 終了 F5= 最新表示 F7= 検索
F11= 経過 ﾃﾞｰﾀ の表示 F12= 取り消し
実行ジョブで「5.処理」
ジョブの処理
システム : MIGAROP7
ジョブ : SP4IKUNIMO ユーザー : KUNIMOTO 番号 : 121681
次の 1 つを選択してください。
1. ジョブ状況属性の表示
2. ジョブ定義属性表示
3. ジョブ実行属性の表
4. スプール・ファイル
10. ジョブ・ログ表示 (
11. 呼び出しスタックの
12. ロックの処理（活動
13. ライブラリー・リス
14. オープンされたファイルの表示（活動状態の場合）
15. ファイル一時変更の表示（活動状態の場合）
16. コミットメント制御状況の表示（活動状態の場合）
続く ...
選択項目またはコマンド
===>
F3= 終了 F4= プロンプト F9=ｺﾏﾝﾄﾞ の複写 F12= 取り消し
ジョブ、ユーザー、番号を確認
- SmartPad4
#121681 - MIGARO.St
localhost/smartpad4iservlet/smartpa
Salesforce mail G翻訳 エキサイト 翻訳
番号はブラウザのタイトルバーでも確認可能

最近のブラウザでは、Chrome が機能や動作速度で優れており、使用しているユーザーが最も多い。

そこで数種あるブラウザのなかから、本稿では Chrome ブラウザに標準搭載されている「デベロッパー・ツール」を題材に説明する。デベロッパー・ツールは Chrome ブラウザを導入していれば、無償で利用できる。

デベロッパー・ツールの実行方法は簡単である。Chrome ブラウザを選択した状態で F12 キーを押下、または「ブラウザのメニュー」→「その他のツール」→「デベロッパー・ツール」から起動できる。【図 11】

デベロッパー・ツールは、デフォルトではブラウザにドッキングした状態で表示される。ドッキングされた状態では使いづらい場合、デベロッパー・ツールのメニューから［Dock side］を選ぶことで、別ウインドウの表示に変更できる。【図 12】

3-3. JavaScript のデバッグ手法

ブラウザのデベロッパー・ツールでは、開発ツールのように JavaScript のソースへブレークポイントを設定し、ステップ実行や変数の内容をチェックしながら JavaScript を実行できる。これによって IBM i プログラムと同様に、JavaScript などのデバッグ作業が可能となる。

ここからは、実際に JavaScript のデバッグ方法について説明していく。

まず SmartPad4i アプリケーションを実行後、デベロッパー・ツールを起動する。【図 13】

メニューの「Sources」タブを選択後、ツリーに表示されるファイルを選ぶと、実行中の JavaScript ソースが表示される。【図 14】

表示されたソースの行番号をクリックすることで、ソースにブレークポイントを設定できる。ブレークポイントを設定しておくと、画面を操作して JavaScript が該当行に進んだ時点で停止する。

またブレークポイントを設定する別の方法として、JavaScript のソースに、「debugger;」と記述する方法もある。

debugger; が呼び出されると、ブレークポイントと同様に JavaScript を一時停止させられる。【図 15】

JavaScript の処理がブレークポイントに到達すると、ブラウザの画面側は停止状態になるので操作はできない。【図 16】

停止後は、右上のメニューで実行、停止、ステップ実行が可能となり、プログラムの実行内容を細かくチェックできる。【図 17】

また JavaScript のデバッグ時には、コンソールから任意の JavaScript コードを実行できる。コンソールはソース表示の下部にあり、Console タブを選んで利用する。

たとえば、コンソールで計算結果位置に“TEST”の文字列を出力する JavaScript を記述して実行すると、画面上に“TEST”が表示される。【図 18】

とくに特殊データや実行条件を必要とする場合、そうしたテスト環境を作らなくても簡単に指定できるので、調査時に便利である。

またデベロッパー・ツールでは、表示されたソースを直接編集することも可能である。この機能を使うと、デバッグをしながら JavaScript を修正でき、作業効率が非常によい。【図 19】

3-4. HTML のデバッグ手法

デベロッパー・ツールを利用すると、JavaScript だけではなく、HTML、CSS についても値を変更しながら表示確認できる。

使い方は、開発者ツールの「Elements」タブを選び、一番左上のアイコンを選択後、ブラウザに表示されている画面で確認したい項目をクリックするだけである。【図 20】【図 21】

項目を選択するとソース上の該当箇所が反転し、CSS で定義されている設定、画面上のサイズ、イベント処理などさまざまな情報を確認できる。

さらに表示された設定は、デベロッパー・ツールで変更すると、ブラウザ上の画面にも直接反映される。画面が思いどおりに調整できない場合は、画面を見ながらソースを変更できる。【図 22】

もちろんこの設定は一時的な変更なので、最終的には HTML や CSS の設定を再定義する必要はあるが、レイアウト調整はかなり効率化できる。

3-5. 通信内容のチェック

デベロッパー・ツールには、Web サーバーとブラウザ間の通信内容の詳細を確認する機能も搭載されている。

デベロッパー・ツールの［Network］タブを選択後に、SmartPad4i プログラムからサーバーにリクエストを送信すると、HTML、CSS、JavaScript ファイル、画像ファイルなどサーバーから受信するファイルのリストが表示される。

これは、画像や外部定義のファイルが読み込めない場合の確認に有効である。パス記述の誤りや、ファイルがサーバーに存在しないなどの誤りを即座にチェックできる。【図 23】

画像ファイルが存在しない場合などは、ブラウザ画面に表示されないので、比較的簡単に特定できる。しかし外部定義の CSS や JavaScript ファイルが読み込まれていない場合には、気づかないこともあるので、ネットワーク監視は有用である。

また画面表示の過程で必要とされる時間も確認でき、パフォーマンスの指標としても利用できる。

3-6. 他のブラウザツール

本稿では Chrome に搭載されているデベロッパー・ツールについて紹介してきたが、Internet Explorer、Microsoft Edge、FireFox にも開発者ツールは搭載されている。

それぞれにインターフェースは異なるが、ここで紹介したような機能は Chrome のデベロッパー・ツールと同じく標準搭載されているので、実際に利用しているブラウザを使うのがよい【図 24】。もちろんこれらのツールも、Chrome の「デベロッパー・ツール」と同じく、ブラウザに標準で搭載されている。

4.まとめ

以上、SmartPad4i を使った Web やハイブリッドアプリケーション開発で有効なデバッグテクニックを説明した。

デバッグでは IBM i 側とブラウザ側の両方で、さまざまな角度から調査するためのツールがすべて標準で用意されている。これらのツールは非常に便利で、優れた機能を備えている。

図6

STRSRVJOBコマンド

STRSRVJOB

STRSRVJOB JOB(番号/ユーザー/ジョブ)

図7

こうした機能を調査・検証するなかで、IBM i やブラウザの機能が日々進化していることをあらためて実感した。

プログラム開発やアプリケーション障害時の調査では、いかにデバッグテクニックを駆使できるかが、作業効率の観点では重要になる。開発者の方々には、こうした機能を作業効率アップにぜひ活用していただきたい。

M

図8

自動応答リストの追加コマンド

RPG

```
ADDRPYLE SEQNBR(9700) MSGID(RPG0000) RPY('D')
```

ILERPG

```
ADDRPYLE SEQNBR(9800) MSGID(RNQ0000) RPY('D')
```

COBOL

```
ADDRPYLE SEQNBR(9900) MSGID(LBE0000) RPY('D')
```

ソース1

自動応答を返答するための処理

```
0001.00 PGM
0002.00 CHGJOB INQMSGRPY(*SYSRPYL)
0003.00 CHGLIBL LIBL(SMPLIB  SP4I QTEMP QGPL)
0004.00 ENDPGM
```

図9

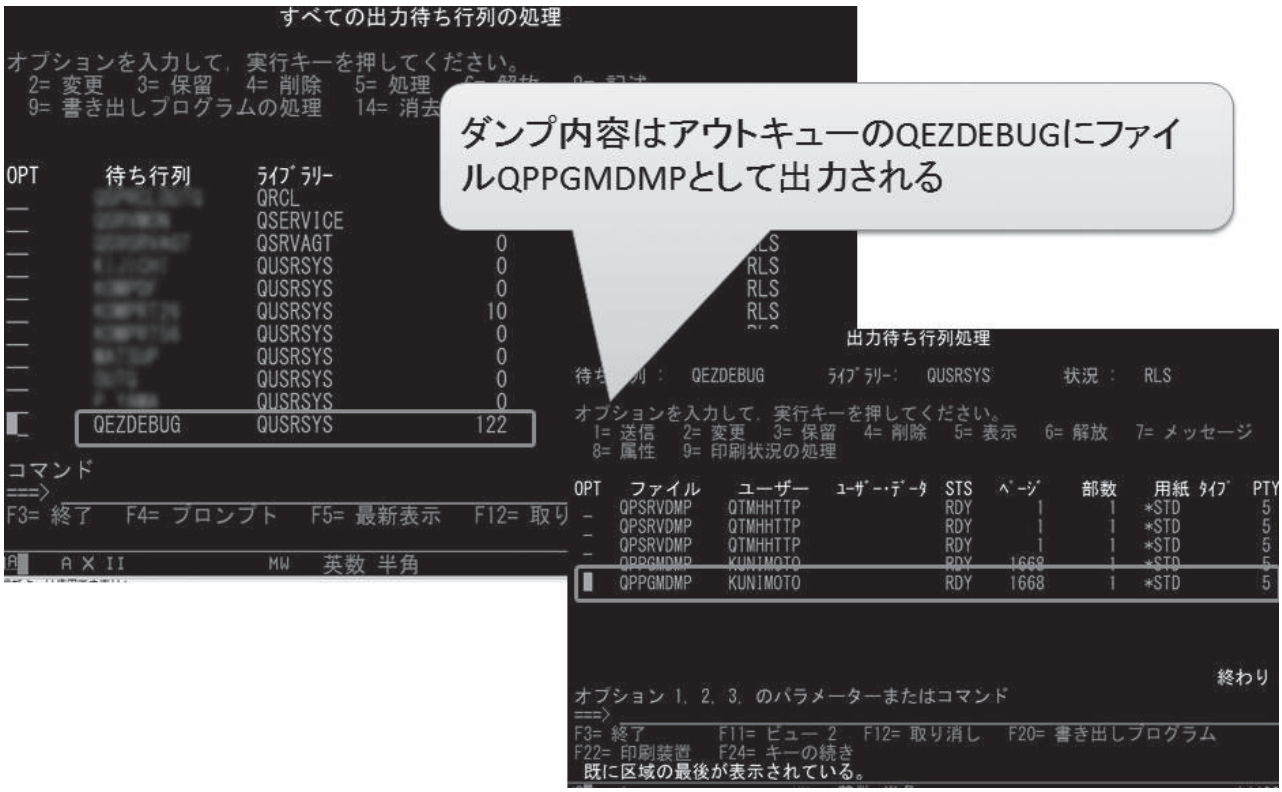

すべての出力待ち行列の処理
オプションを入力して、実行キーを押してください。
2= 変更 3= 保留 4= 削除 5= 処理
9= 書き出しプログラムの処理 14= 消去
OPT 待ち行列 ﾗｲﾌﾞﾗﾘｰ
QRCL
QSERVICE
QSRVAGT 0
QUSRSYS 0
QUSRSYS 0
QUSRSYS 10
QUSRSYS 0
QUSRSYS 0
QUSRSYS 0
QUSRSYS 0
QEZDEBUG QUSRSYS 122
コマンド
===>
F3= 終了 F4= プロンプト F5= 最新表示 F12= 取り
A X II MW 英数 半角
ダンプ内容はアウトキューのQEZDEBUGにファイルQPPGMDMPとして出力される
出力待ち行列処理
待ち行列 : QEZDEBUG ﾗｲﾌﾞﾗﾘｰ: QUSRSYS 状況 : RLS
オプションを入力して、実行キーを押してください。
1= 送信 2= 変更 3= 保留 4= 削除 5= 表示 6= 解放 7= メッセージ
8= 属性 9= 印刷状況の処理
OPT ファイル ユーザー ﾕｰｻﾞｰ･ﾃﾞｰﾀ STS ﾍﾟｰｼﾞ 部数 用紙 ﾀｲﾌﾟ PTY
QPSRVDMP QTMHHTTP RDY 1 1 *STD 5
QPSRVDMP QTMHHTTP RDY 1 1 *STD 5
QPSRVDMP QTMHHTTP RDY 1 1 *STD 5
QPPGMDMP KUNIMOTO RDY 1668 1 *STD 5
終わり
オプション 1, 2, 3, のパラメーターまたはコマンド
===>
F3= 終了 F11= ビュー 2 F12= 取り消し F20= 書き出しプログラム
F22= 印刷装置 F24= キーの続き
既に区域の最後が表示されている。

図10

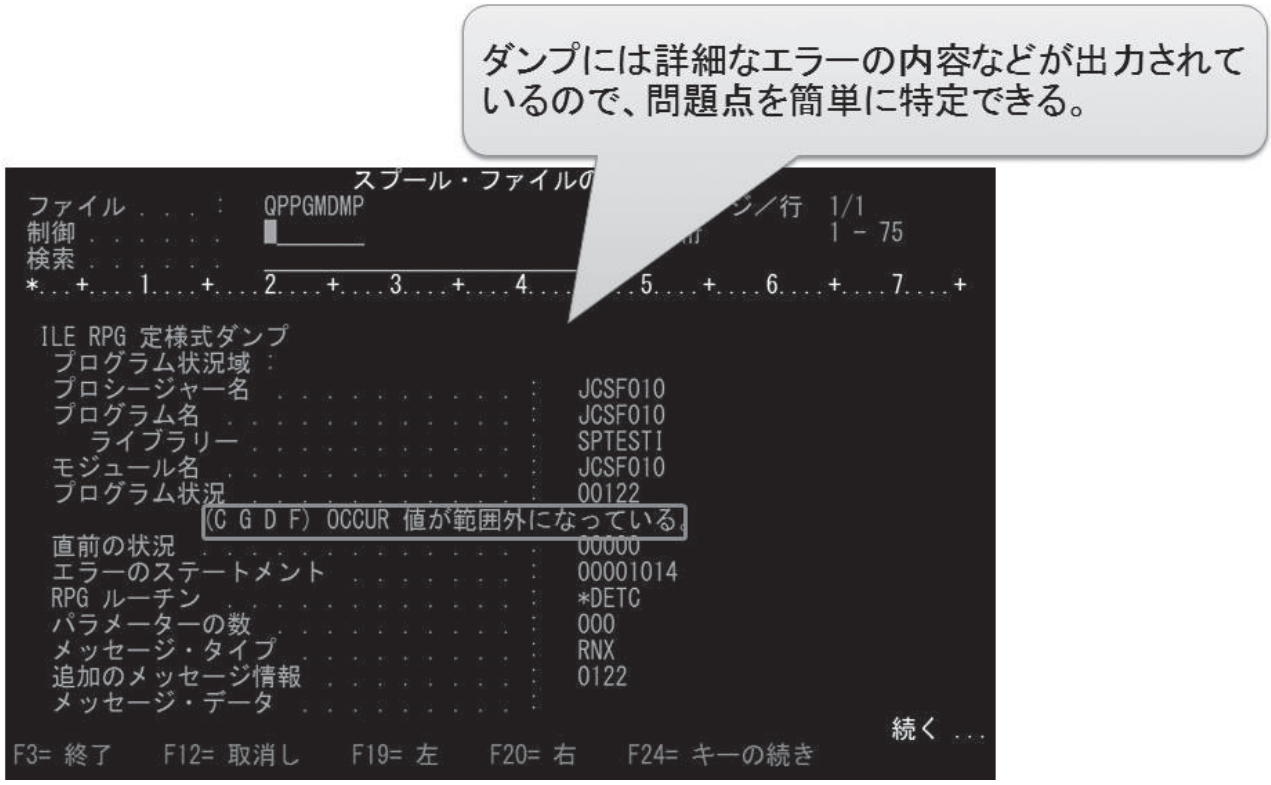

ダンプには詳細なエラーの内容などが出力されているので、問題点を簡単に特定できる。
スプール・ファイルの
ファイル . . . : QPPGMDMP
制御 . . . . . _____
検索 . . . . . _____
*...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+
ILE RPG 定様式ダンプ
プログラム状況域 :
プロシージャー名 . . . . . : JCSF010
プログラム名 . . . . . : JCSF010
ライブラリー . . . . . : SPTESTI
モジュール名 . . . . . : JCSF010
プログラム状況 . . . . . : 00122
(C G D F) OCCUR 値が範囲外になっている。
直前の状況 . . . . . : 00000
エラーのステートメント . . . . . : 00001014
RPG ルーチン . . . . . : *DETC
パラメーターの数 . . . . . : 000
メッセージ・タイプ . . . . . : RNX
追加のメッセージ情報 . . . . . : 0122
メッセージ・データ . . . . . :
続く ...
F3= 終了 F12= 取消し F19= 左 F20= 右 F24= キーの続き

図11


1. メニュー
2. その他のツール
3. デベロッパーツール
新しいタブ
新しいタブ(T) Ctrl+T
新しいウィンドウ(N) Ctrl+N
シークレット ウィンドウを開く(I) Ctrl+Shift+N
履歴(H)
ダウンロード(D) Ctrl+J
その他のツール(L)
編集 切り取り(T) コピー(C) 貼り付け(P)
設定(S)
ヘルプ(H)
終了(X) Ctrl+Shift+Q
閲覧履歴を消 Ctrl+Shift+Delete
拡張機能(E)
タスク マネー Shift+Esc
エンコード(E)
デベロッパー ツール(D) Ctrl+Shift+I

図12


Dock sideをUnlock into separate window
にして別ウインドウにすると使いやすい
別ウインドウ表示
Dock side
Hide console Esc
Search all files Ctrl + Shift + F
More tools
Shortcuts
Settings F1
Help

図13

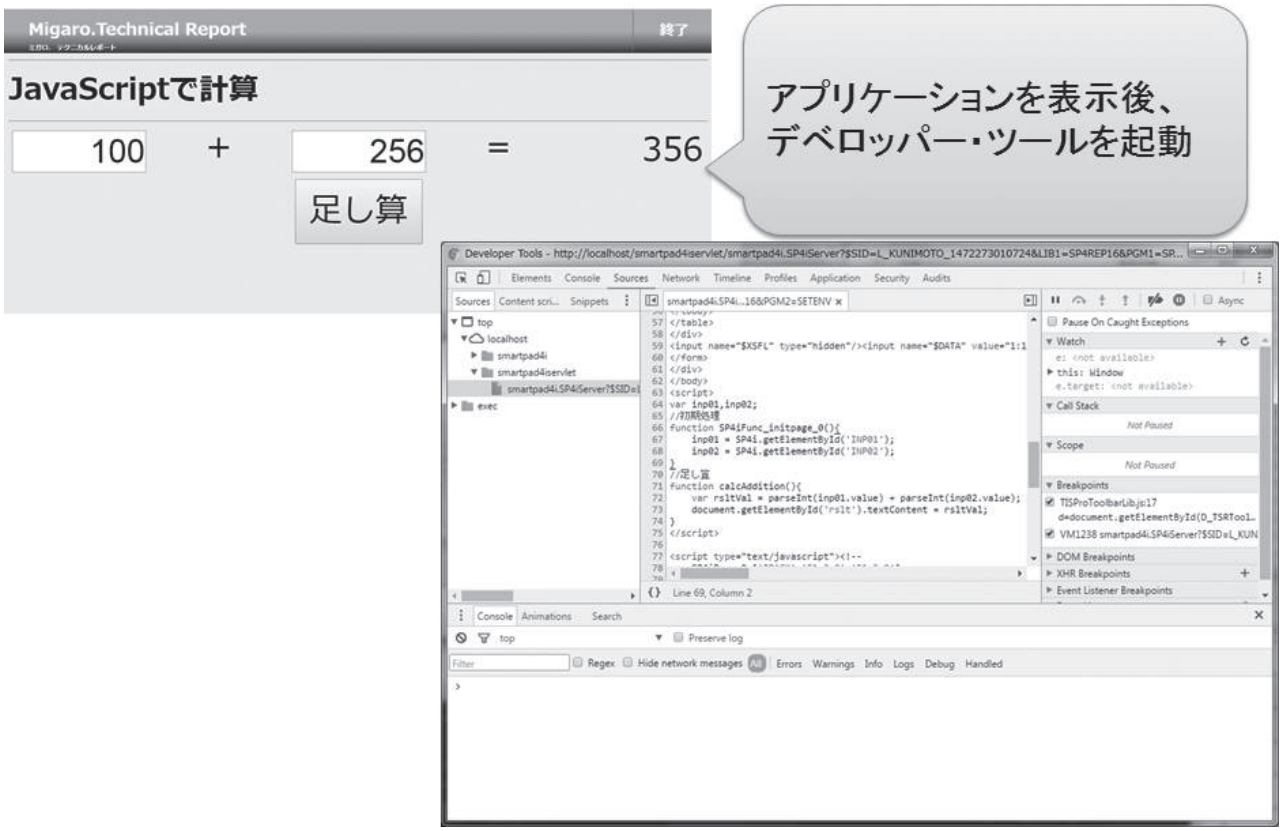

Migaro.Technical Report
終了
JavaScriptで計算
100
+
256
=
356
足し算
アプリケーションを表示後、
デベロッパー・ツールを起動

図14

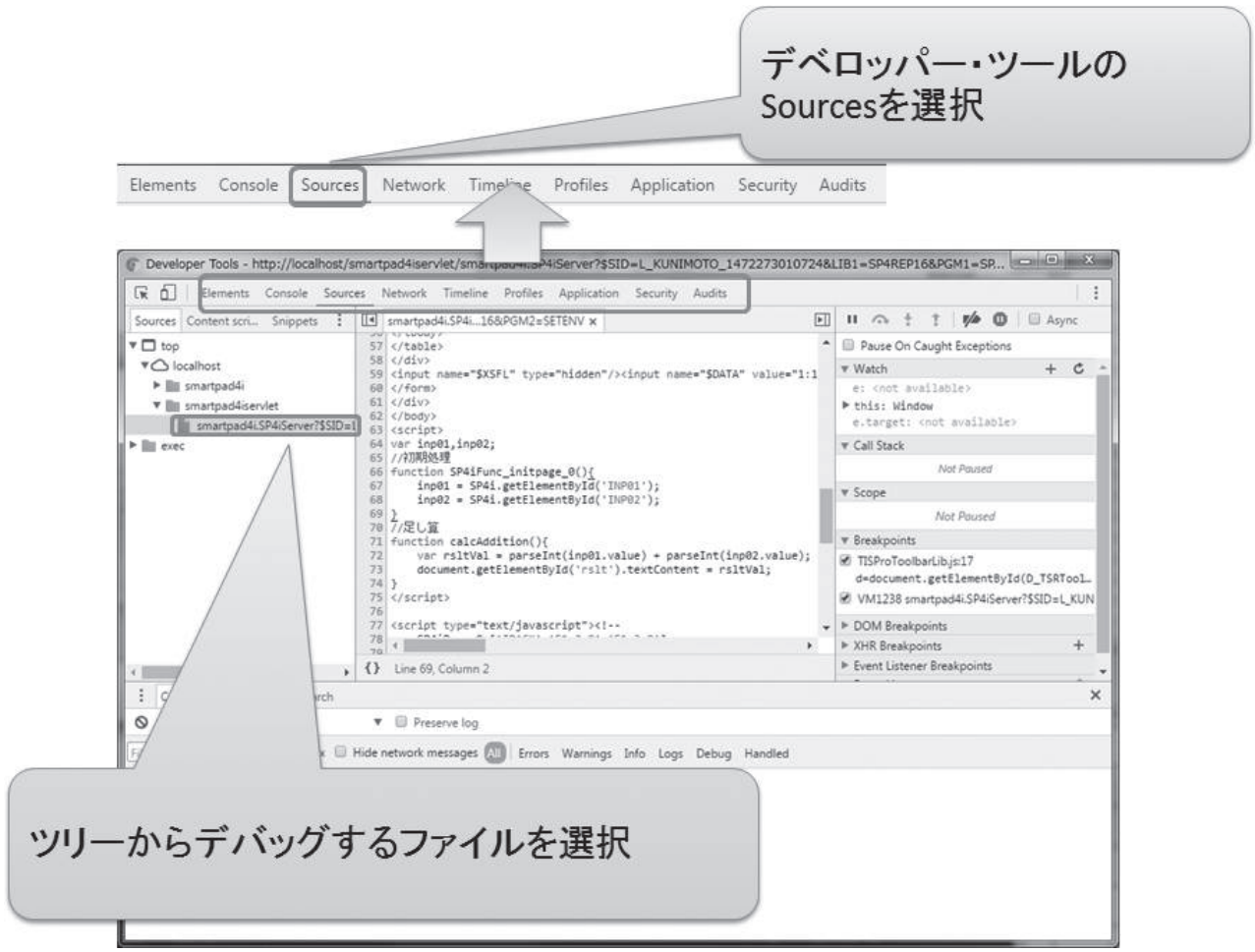

デベロッパー・ツールの
Sourcesを選択
Elements
Console
Sources
Network
Timeline
Profiles
Application
Security
Audits
ツリーからデバッグするファイルを選択

図15


行番号をクリックすることでブレークポイントが設定可能
ソースにdebugger;を記述することでもブレーク可能
63 <script>
64 var inp01,inp02;
65 //初期処理
71 function calcAddition(){
72 var rsltVal = parseInt(inp01.value) + parseInt(inp02.valu
73 document.getElementById('rslt').textContent = rsltVal;
74 }
75 </script>
71 function calcAddition(){
72 debugger;
73 var rsltVal = parseInt(inp01.value) + parseInt(inp02.valu
74 document.getElementById('rslt').textContent = rsltVal;
75 }
76 </script>

図16


localhost/smartpad4iservlet/smartpad4i.SP4iServer?$SID=L_KUNIMOTO_1472273010724&LIB1=SP4REP1
Migaro.Technical Report
Paused in debugger
JavaScriptで計算
0
+
0
=
足し算
ブレークポイントで停止
画面も停止中
59 <input name="$XSFL" type="hidden"/ ="$DATA" value=
60 </form>
61
62 </body>
63 <script>
64 var inp01,inp02;
65 //初期処理
66 function SP4iFunc_initpage_0(){
67 inp01 = SP4i.getElementById('I 1');
68 inp02 = SP4i.getElementById('I 02');
69 }
70 //足し算
71 function calcAddition(){
72 debugger;
73 var rsltVal = parseInt(inp01.value) + parseInt(inp02.valu
74 document.getElementById('rslt').textContent = rsltVal;
75 }
76 </script>

図17

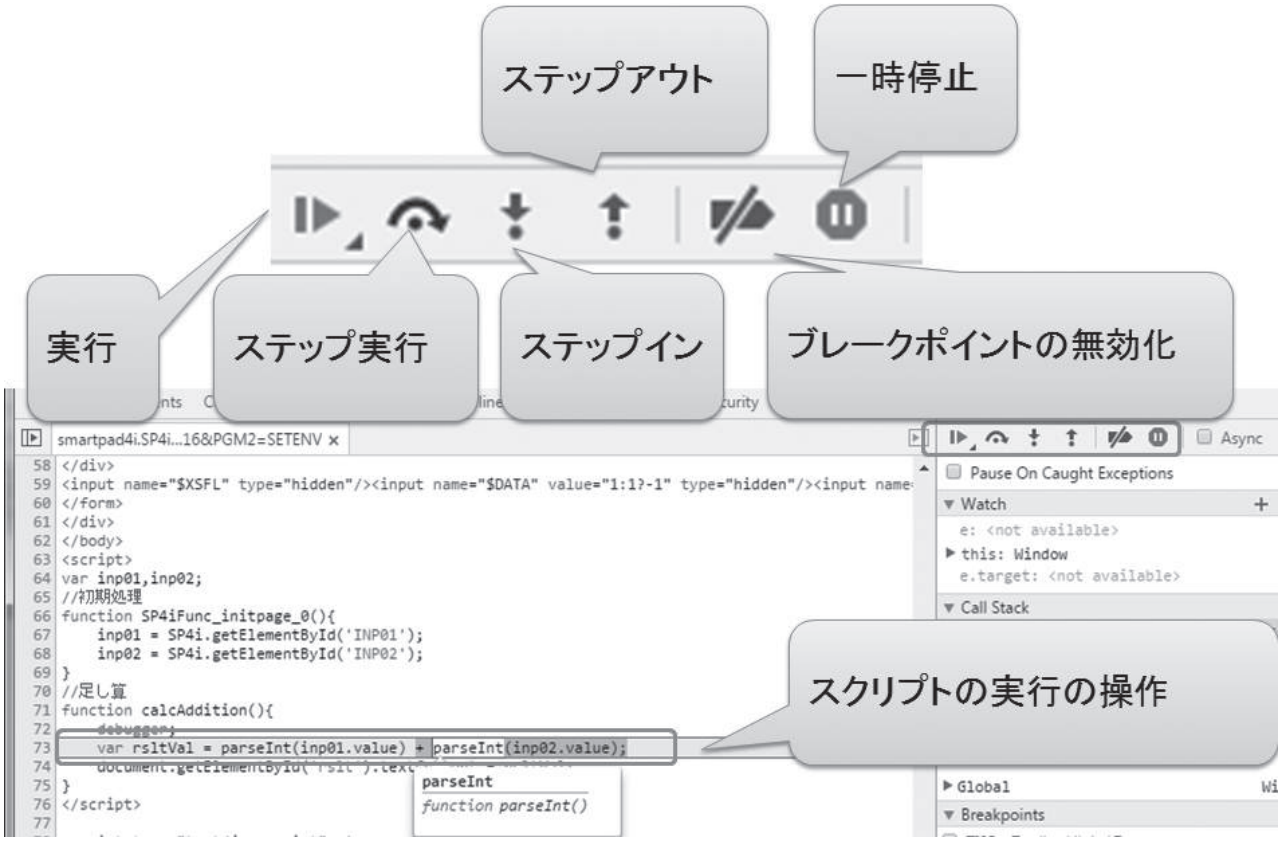

ステップアウト
一時停止
実行
ステップ実行
ステップイン
ブレークポイントの無効化
スクリプトの実行の操作
smartpad4i.SP4i...16&PGM2=SETENV
Async
Pause On Caught Exceptions
Watch
this: Window
Call Stack
Global
Breakpoints
parseInt
function parseInt()

図18


Migaro.Technical Report
JavaScriptで計算
0 + 0 =
足し算
コンソールを使うと
スクリプトが動的に実行可能
Console
Animations
Search
chrome-extension://oh...hhiiaoi
Preserve log
Filter
Regex
Hide network messages
All
Errors
> document.getElementById('rslt').textContent = "TEST"
< "TEST"
Migaro.Technical Report
JavaScriptで計算
0 + 0 = TEST
足し算
計算結果位置にTESTを出力

図19

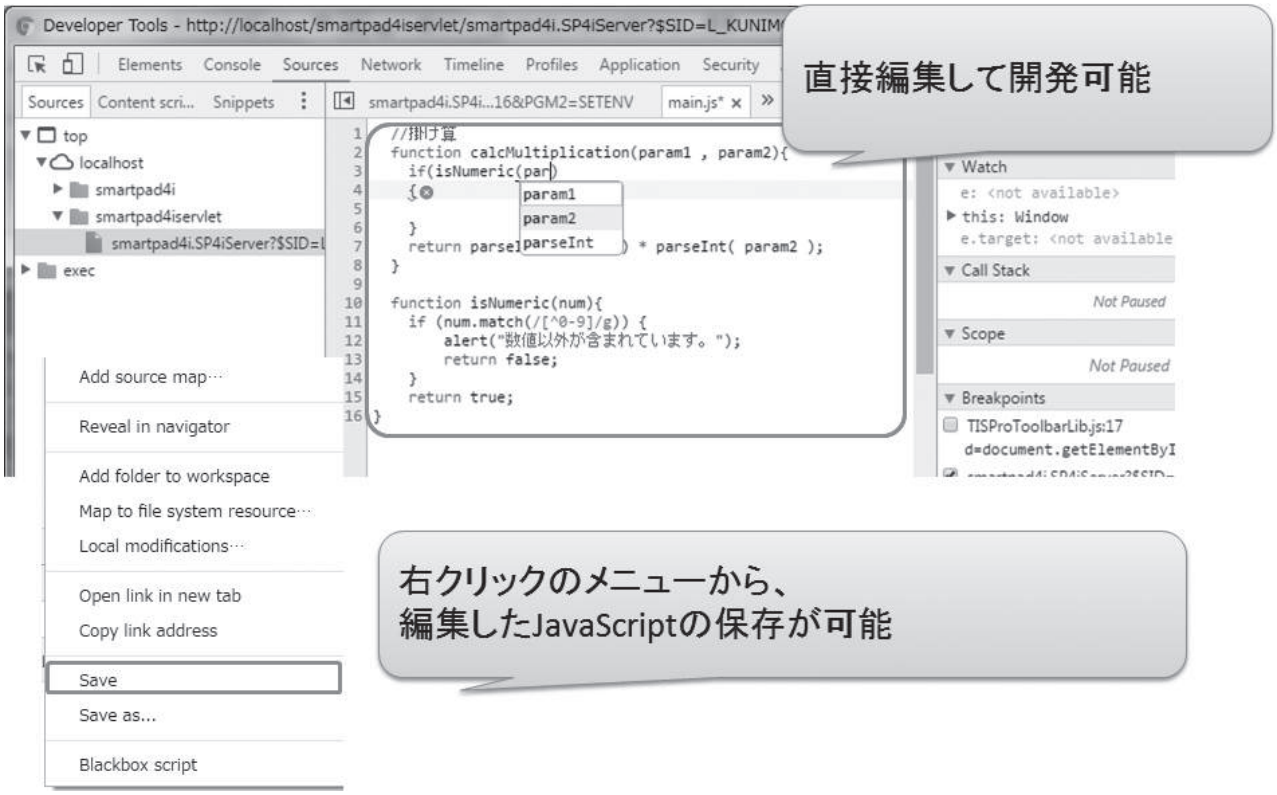

直接編集して開発可能
右クリックのメニューから、
編集したJavaScriptの保存が可能
Add source map…
Reveal in navigator
Add folder to workspace
Map to file system resource…
Local modifications…
Open link in new tab
Copy link address
Save
Save as...
Blackbox script

図20

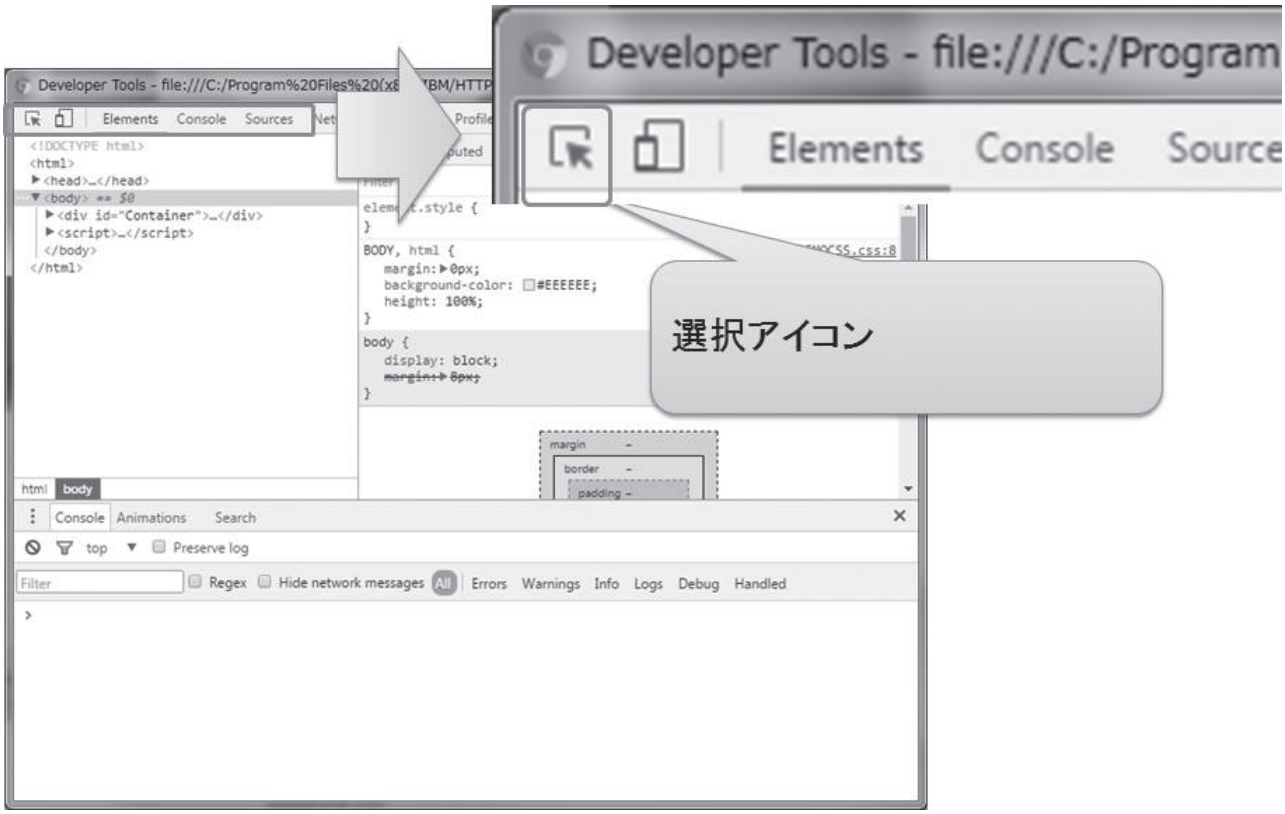

Developer Tools - file:///C:/Program
Elements
Console
Source
選択アイコン

図21

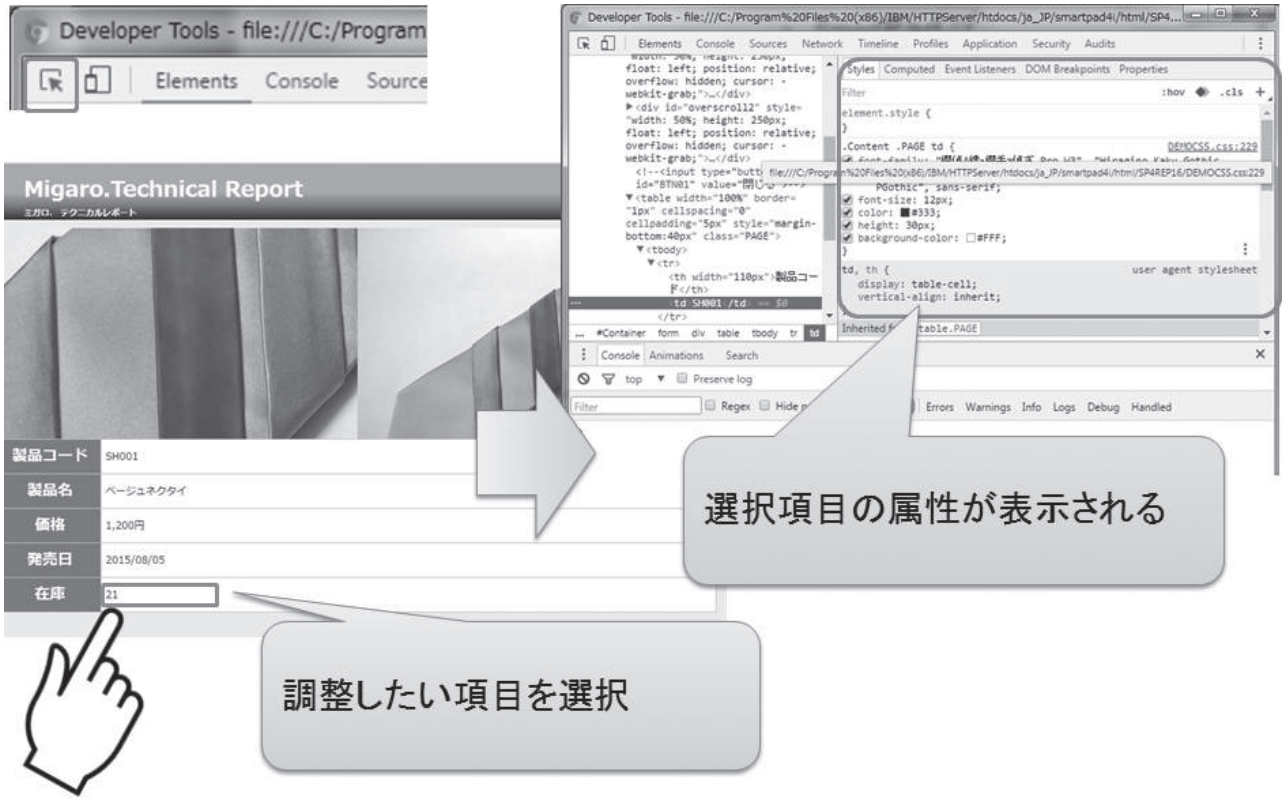

Developer Tools - file:///C:/Program
Elements
Console
Source
Migaro.Technical Report
ミガロ. テクニカルレポート
製品コード
SH001
製品名
ページュネクタイ
価格
1,200円
発売日
2015/08/05
在庫
21
選択項目の属性が表示される
調整したい項目を選択

図22

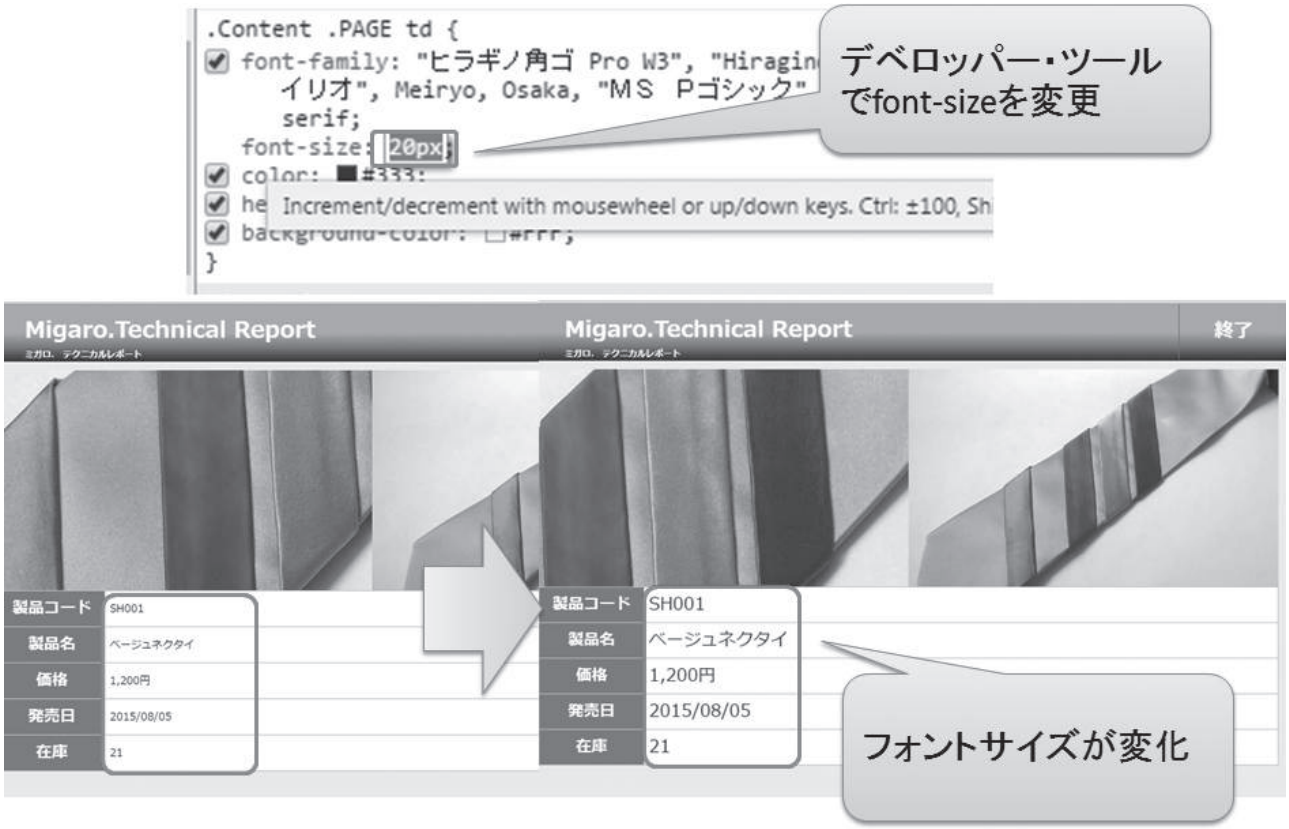

.Content .PAGE td {
font-family: "ヒラギノ角ゴ Pro W3", "Hiragin
イリオ", Meiryo, Osaka, "MS Pゴシック"
serif;
font-size: 20px;
color: #333;
background-color: #fff;
}
Increment/decrement with mousewheel or up/down keys. Ctrl: ±100, Sh
デベロッパー・ツール
でfont-sizeを変更
Migaro.Technical Report
ミガロ. テクニカルレポート
終了
製品コード
SH001
製品名
ページュネクタイ
価格
1,200円
発売日
2015/08/05
在庫
21
フォントサイズが変化

図23

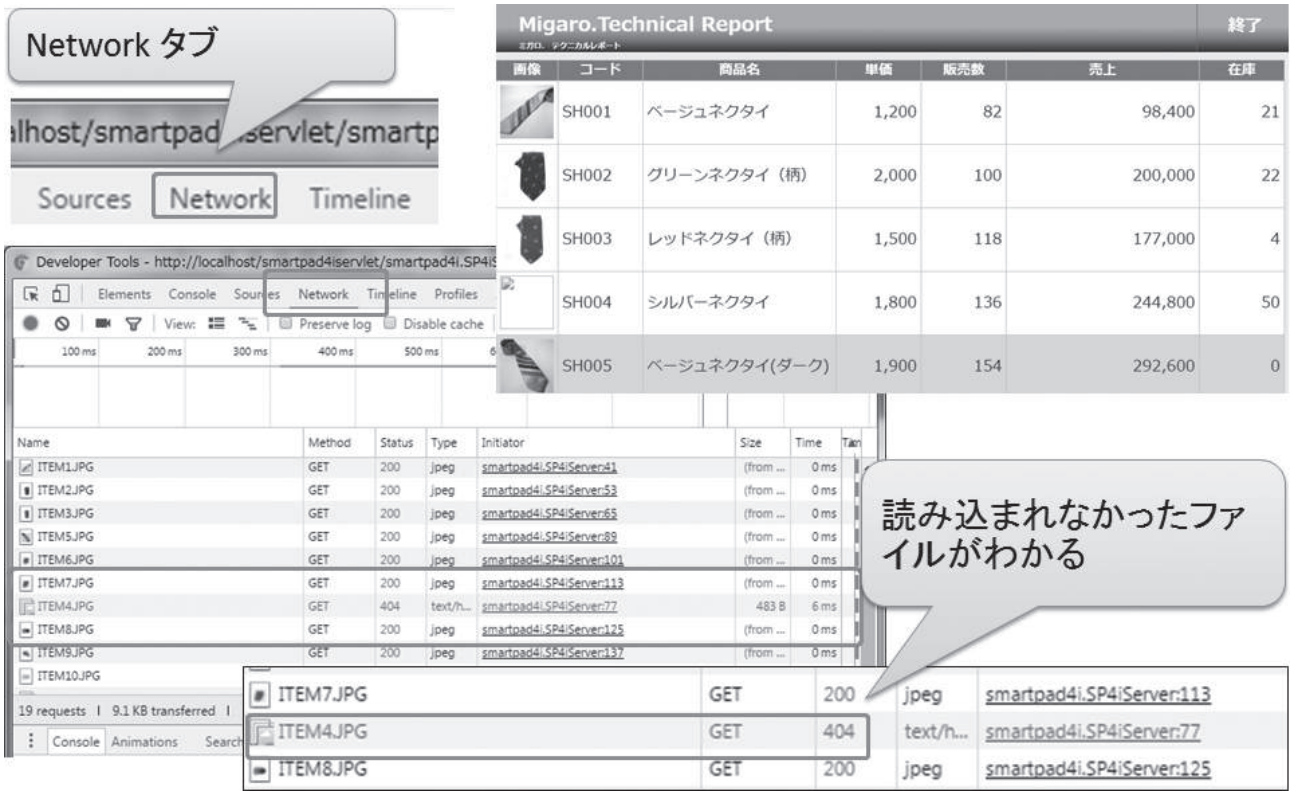
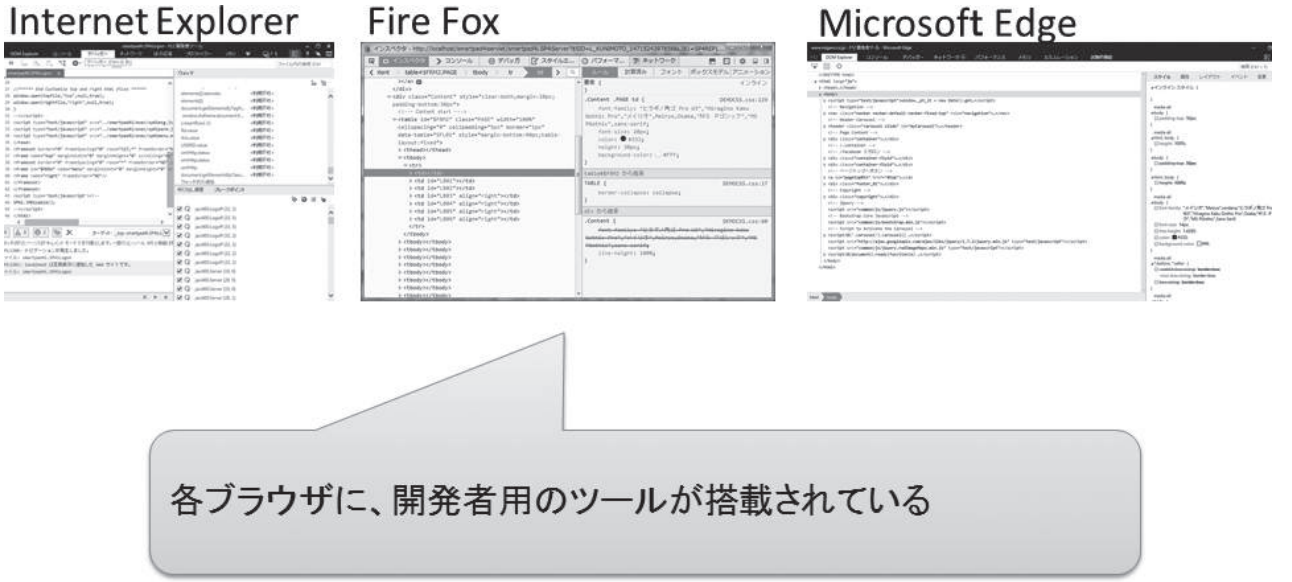

Network タブ
lhost/smartpad servlet/smartp
Sources
Network
Timeline
Migaro.Technical Report
終了
画像 コード 商品名 単価 販売数 売上 在庫
SH001 ベージュネクタイ 1,200 82 98,400 21
SH002 グリーンネクタイ（柄） 2,000 100 200,000 22
SH003 レッドネクタイ（柄） 1,500 118 177,000 4
SH004 シルバーネクタイ 1,800 136 244,800 50
SH005 ベージュネクタイ(ダーク) 1,900 154 292,600 0
Developer Tools - http://localhost/smartpad4iservlet/smartpad4i.SP4i
Elements Console Sources Network Timeline Profiles
Preserve log Disable cache
100 ms 200 ms 300 ms 400 ms 500 ms
Name Method Status Type Initiator Size Time
ITEM1JPG GET 200 jpeg smartpad4i.SP4iServer:41 (from ... 0 ms
ITEM2JPG GET 200 jpeg smartpad4i.SP4iServer:53 (from ... 0 ms
ITEM3JPG GET 200 jpeg smartpad4i.SP4iServer:65 (from ... 0 ms
ITEM5JPG GET 200 jpeg smartpad4i.SP4iServer:89 (from ... 0 ms
ITEM6JPG GET 200 jpeg smartpad4i.SP4iServer:101 (from ... 0 ms
ITEM7JPG GET 200 jpeg smartpad4i.SP4iServer:113 (from ... 0 ms
ITEM4JPG GET 404 text/h... smartpad4i.SP4iServer:77 483 B 6 ms
ITEM8JPG GET 200 jpeg smartpad4i.SP4iServer:125 (from ... 0 ms
ITEM9JPG GET 200 jpeg smartpad4i.SP4iServer:137 (from ... 0 ms
ITEM10.JPG
19 requests 9.1 KB transferred
Console Animations Search
読み込まれなかったファイルがわかる
ITEM7JPG GET 200 jpeg smartpad4i.SP4iServer:113
ITEM4JPG GET 404 text/h... smartpad4i.SP4iServer:77
ITEM8JPG GET 200 jpeg smartpad4i.SP4iServer:125

図24


Internet Explorer
Fire Fox
Microsoft Edge
各ブラウザに、開発者用のツールが搭載されている

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

あなたはブ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 技術支援課

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

MIGARO.
TECHNICAL
REPORT

Migaro.Technical Report
No.9 2016 年秋
ミガロ.テクニカルレポート

2016 年 11 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
〒106-0041
東京都港区麻布台1-4-3
エグゼクティブタワー麻布台 11F
TEL:03(5573)8601
FAX:03(5573)8602