MIGARO. TECHNICAL REPORT

ミガロ.テクニカルレポート

No.8
2015年秋

株式会社ミガロ.

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。

さて、「ミガロ.製品をご利用中の技術者の皆様に、日々の開発に少しでもお役に立つような技術情報をご提供したい」という思いから 2008 年に創刊した『Migaro.Technical Report』は、このたび第 8 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・ご要望をお寄せくだ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。

昨今では、スマートフォンやタブレット導入の一層の進展に伴い、基幹業務でモバイル機器を活用されるお客様が増加しております。これに伴い、弊社でも各製品のモバイル対応機能を強化してまいりました。本年 5 月にリリースした Delphi/400 XE7 は、従来のマルチデバイス対応をさらに進め、PC はもちろん、iOS、Android 等、異なる端末向けプログラムを「1 ソース」で開発することで、高い開発生産性を実現します。

今回も従来と同様に、第 1 部は「Migaro.Technical Award 2015 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」「JC/400」「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、『iPod Touch の業務利用開発と検証』や『ブランク加工図管理システムの構築』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、『マルチスレッドを使用したレスポンスタイム向上』や、モバイル開発の応用手法となる『Android NFC 機能のネイティブアプリケーション実装』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 8 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。

2015 年秋

株式会社ミガロ.
代表取締役社長
上甲 將隆

Migaro. Technical Award 2015

お客様受賞論文／ミガロ. テクニカルアワード

最優秀賞

iPod Touchの業務利用開発と検証
―多目的端末としてどこまでできるか?

石井 裕昭 様
豊鋼材工業株式会社
製造総括部 部長

豊鋼材工業株式会社
http：//www.yutaka-steel.co.jp/

鋼板加工のトータルコーディネーターとして、レベラー・スリット・ 溶断・開先・穴明け・プレス・ベンディング・溶接・マシニング・ショット・プライマー設備を有機的に活用し、幅広い産業分野のお客様に商品・サービスを提供している。九州・沖縄エリアでは業界トップシェア。伊藤忠丸紅鉄鋼・新日鐵住金系の会社である。

開発の経緯

豊鋼材工業は2005年より、新生産管理システムとして工場内のハンディターミナルやバーコードプリンタなどのハードウェアの整備と、Delphi/400導入による工程管理、トレーサビリティの強化、業務効率化、見える化などを推進してきた。

その一方、工場の現場サイドから情報を利用・参照したり、簡単に情報を入力して作業効率を高めるための端末に関しては、使用環境や適正な端末形態、開発ツール、価格等の制約より整備が進んでいなかった。

このような中で、写真を扱う業務で写真データの一貫管理のニーズが発生し、さらに他の業務用途でも機能拡張のニーズがあることから、Delphi/400の最新バージョン化と開発環境の整備により、従来の業務プロセスを大きく改善できるのではないかと考えた。

なお、従来から使用している生産管理システムは「Delphi/400 Ver2007」で継続開発しており、開発環境が「Delphi/400 XE5」(当時の最新バージョン）に変わる場合、Unicode対応のための既存ソースの改修や、サードパーティのコンポーネントの対応などでコンバージョンに要する工数が膨大になると判断し、既存の開発ライセンスを継続し、並行運用にすることとした。今後は、用途、利用端末により開発・運用環境を使い分けていく予定である。

工場で使用する端末については、業務の内容や用途によって適切なOS、サイズ等は異なるものの、まずは写真撮影用途を意識してiPod Touchでの機能検証から事前スタディを開始した。

iOSのiPod Touchを端末として選定した理由は、端末メーカー、モデルによる挙動の違いを意識しなくてよく、端末を随時追加する際に、ほぼ同じモデルを提供できるのと、電話機能がなく月々の通話料が発生しないからであった。

iPod Touchで開発・検証した機能

事前スタディで、iPod Touchで実現したい機能の実装可否と、業務用途として実際に耐え得るかの確認を行った。期待している機能は、以下の通りである。

① Delphi/400で開発する新たな用途、およびGUI化済みの既存業務機能
②従来よりハンディターミナルの5250エミュレータ用に作られ、運用されている機能（5250エミュレータは株式会社ヒットの「WaveLink TE」を利用）
③通話、メッセージなどのコミュニケーション機能（Skype利用）

①で新たに開発するのは、工程写真・鋼材写真の撮影・保管管理機能、作業予定・順番情報公開機能、生産完了実績入力機能、作業時間（日報）報告機能などである。一部はすでにネイティブアプリとして開発・運用済みであるが、今後も継続的にニーズ掘り起こし、開発を進め

表1　iPod Touchでの開発･検証内容

分類	項目	内容、備考
新規Delphi開発機能	・工程、鋼材ステンシル管理用カメラ ・作業予定ロット、順番公開 ・生産実績入力 ・稼動日報 ・出荷準備(梱包)済現品情報、など	新しいDelphi開発環境にて、従来管理できていなかった情報を取得可能とする。時間を要していた入力業務を簡易化するアプリを開発し、効率化と管理レベル向上を図る。
既存ホスト業務	・生産実績入力(完成メニュー) ・出荷チェック ・現品票作成 ・ロケーション管理 ・棚卸し　など	従来ハンディーターミナル用に作られている各種業務メニューをWaveLinkというアプリで表示し、既存の業務を行う。 バーコード読取専用機ではないため、読取のレスポンス等に応じて適用業務は選定する。 既存機器の約1/3以下のコストで機能を実現したい。
コミュニケーション機能	・Skype(無料通話)による iPod等の登録ユーザー間通話 メッセージ機能など	無料アプリSkypeにより無線LANエリア内での Skype同士の通話を可能化。 またはメッセージ、ビデオ通話など




表2　iPod Touchでの写真撮影機能開発の目的

分類	目的	従来の課題と経緯
品質保証体制の強化(トレーサビリティ)	▪ 品質証明としての写真情報を管理維持する基盤づくり ▪ 鋼材の保有･使用履歴の客観的証拠として原則全鋼板のステンシル写真を保管 → (鋼材制限の緩和=滞留材削減) ▪ 生産履歴･鋼材識別情報と写真の関連性の維持とサーバー内一元管理	要求時のみ個々のカメラで撮影された工程写真データが工場スタッフ、営業担当のPCの任意のフォルダーに保存され、フォルダー、ファイル名等で整理。保管ルール取決め無し。 品質証明書類として鋼材のステンシル写真要求増。使用時にステンシル部分無いと使用制限され滞留在庫の一因。またステンシル有無の管理も必要。 事後に特定の生産履歴、又は特定の鋼材について要求されても、有無の確認、検索は困難。
関連作業の効率化	▪ 1台／機種の保有で機種間のカメラ探し(オペレータ)をなくす ▪ 工場スタッフによる撮影をオペレータ作業に完全移行し、撮影待機(スタッフ、オペレータ共)、連絡のロス時間等をなくす ▪ 工場スタッフのカメラ(メモリカード)回収、データ仕分け作業をなくす ▪ 営業担当者の提出書類と写真内容の照合作業を半自動化	限られた台数のデジカメを共有使用で、手元に無い場合は都度探す必要があった。 工場スタッフが撮影時は対象材処理の都度、連絡を取る、予定時間を確認など必要。双方の待機時間の無駄。 写真データ回収を意識する必要があり、回収データを客先毎に仕分けには黒板の工事名を全て確認必要。 板番、ロット番号等との関連も写真の画像を一枚毎に開いて確認するしかなく、手間を要する。

る。

②は、バーコード、QR コードの読み取りを iPod Touch のカメラ機能で行うことによる、操作性、レスポンス、読み取り可否等の確認である。実用に耐え得るかがポイントとなる。これが有効であれば、既存の高価なハンディターミナルの代替機として低コストでの導入が可能となる。

③は、WiFi 環境の工場内や事務所間で無料通話が行えれば利便性が高く、構内電話が不要になる可能性もある。

以上のポイントを【表 1】にまとめる。

Delphi/400 XE5による写真撮影機能の開発

従来の課題とニーズ

構造物等で使用される鋼材は、使用箇所などの条件を考慮して、さまざまな材質で注文を受ける。鋼材の材質は、外見では判断できないため、工場における在庫管理、識別管理と、確実な使用実績の報告により、使用鋼材の証明書を提出する。通常は、鉄鋼メーカーが発行するミルシート（鋼材検査証明書）から該当分を提出することが多い。最近は、一時期の鋼材偽装問題も影響してか、写真での証明提出を求められるケースが多くなっている。

品質保証体制強化を含めた開発の目的を【表 2】にまとめる。また、この場合、写真撮影から提出までのプロセスで、次のような問題があり、業務効率化の妨げとなっていた。

・デジタルカメラの台数制約による共有化→カメラ探し時間
・スタッフによる撮影データの回収
・スタッフが撮影する場合の、タイミングの連絡、待ち時間の発生など
・回収した撮影データ（複数のお客様分が混在）から特定分のみを抽出する手間（個々の画像確認し選別）
・撮影データと生産実績データ（基幹情報）の関連がなく、事後の検索が困難
・お客様からの要求がなく撮影していない鋼材は、事後の提出が不可能

上記の問題を解決するために、iPod 端末は 1 つの作業機器に対して 1 台を割り当て、写真撮影に関するアプリは、次のような仕様で開発することにした。

・撮影した写真データは、ネットワーク上のサーバーに自動的に保存する。
・写真データのファイル名は、検索の容易性を考慮し、キー情報を含むユニークな値とする。
・キーとなる情報は、鋼材の識別番号もしくは作業ロット番号とし、撮影前に QR コードまたはバーコードのいずれかで読み取る。
・IBM i のデータベースに、写真のファイル名、保存先フォルダー名、撮影日時、鋼材番号、作業ロット番号などを持つ写真データ抽出用のレコードを、写真 1 枚ごとに書き込む。
・保存された写真データを特定し、抽出・整理する機能は、IBM i の生産実績データベース、鋼材データベース等と関連させて既存の生産管理システムに追加する（操作は既存の PC）。

以上の処理フローを【図 1】に、iPod Touch での画面操作の流れを【写真 1】～【写真 2】に示す。

これらの仕様を実現するための iPod アプリ開発は、ミガロ．のサポートを受けながら主に次のポイントを押さえて進めた。

システム機器の構成は【図 2】の通りである。

QRコード、バーコード読み取り

Delphi/400 XE5 には、QR コード、バーコード読み取り用のコンポーネントは含まれていないため、当社では TMS 社のフリーコンポーネント「TTMSFMXZBarReader」を登録して開発を進めた。

読み取り処理の実装自体は比較的容易である。使用したコンポーネントでのコード記述例を【ソース 1】に示す。

このコードからわかるように読み取り画面（カメラ）は Show メソッドで起動し、読み取りが成功すると自動的にコンポーネントの GetResult イベントが動く。GetResult には読み取られた文字が引数で渡されるので、この文字列を利用するようにコード記述するだけである。ただし、一次元バーコード読み取りの場合、桁数にもよるが、ある程度の大きさが必要となり、当社では鋼材ラベルのバーコードは約 10％ほど幅を拡大した。QR コードでは一次元バーコードよりレスポンスはかなり改善されるが、こちらもサイズによっては読み取り不能となる。

有料アプリなどではストレスなく読める場合もあるため、改良の可能性はあると思われるが、現状では手がかりがない。ハンディターミナル相当の業務を iPod 等で代替するには、特に QR コード読み取りの操作性とレスポンスが最大のポイントであり、頻度にもよるが一定の読み取りレベルに到達できれば、その汎用性からもさまざまな業務で置き換えが進む可能性がある。

IBM iの情報抽出と表示

DataSnap サーバー経由での取得となるが、抽出条件（SQL 文）は、ClientDataSet の CommandText に記述する。TQuery ではオブジェクトインスペクターの SQL プロパティで Lines への記述であったので長文でも問題ないが、CommandText プロパティは複数行表示不能のため、プログラムで行を分けてコードを記述した方がよい。行分け時は、＋でつなぐテキスト間のスペースを忘れないようにする。なお、抽出したデータを表示する DBGrid のようなコンポーネントがないので、各項目の意味がわかるように Label にタイトルとレコードの値を組み合わせて表示した。

また、方法は省略するが、ListView コンポーネントで、LiveBinding デザイナーによって ClientDataSet と関連づけて表示することもできる。

オブジェクトインスペクタのプロパティでの SQL 記述の比較を【図 3】に示す。　また、コードでの記述方法の比較を【ソース 2】に示す。

写真データのサーバー保存

撮影された iPod の写真データを、FTP プロトコルを用い、あらかじめ準備されたサーバー上のフォルダに jpg 形

図1　写真撮影、保管、利用のフロー

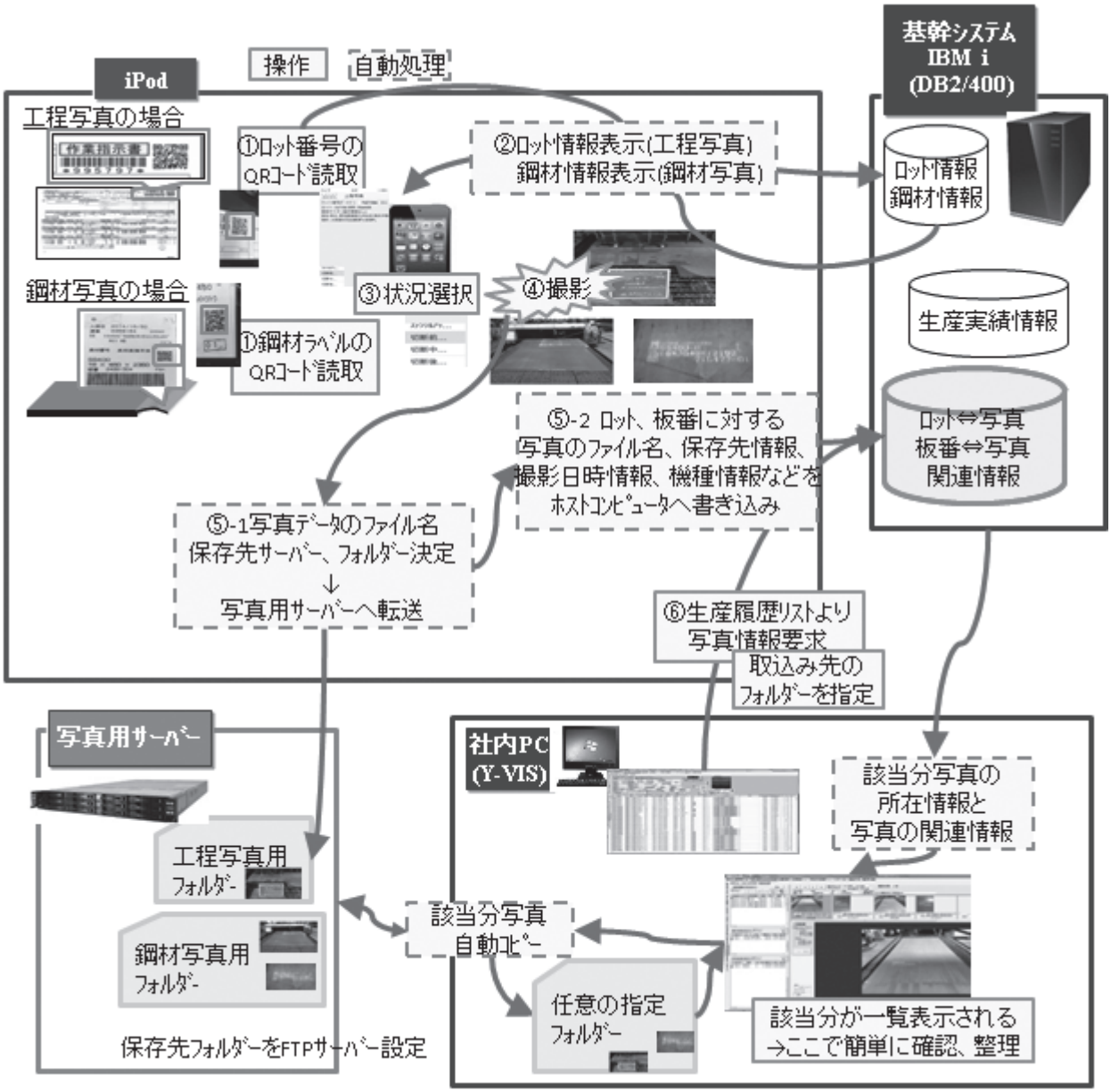


写真1　カメラ起動までの操作画面

式で保存している。具体的には、IdFTPコンポーネントを配置して保存先のサーバー（hostプロパティでIPアドレス指定）にConnectメソッドで接続し、指定のフォルダーへChangeDirメソッドで移動の上、Putメソッドでファイルを保存できる。なお、移動元ファイルは、Camera撮影のActionで撮影後に実行されるOnDidFinishTakingイベントで得られるBitmapのイメージをファイルに割り付けたものである。

IBM iのデータベースへの情報書き込み

これもClientDataSetのCommandTextに記述するSQL文（Insert、Updateなど）で、ClientDataSet.Executeで実行する。

写真アルバムへの保存はオプションで

当初、トラブルを想定して端末の写真アルバムにも撮影データを保存していた。しかし、コードの不備でメモリが消費されアプリのクラッシュが頻発したため、メモリ解放するようにコードを改修するとともに、アルバムへの保存は初期設定でのオプションとした。【ソース3】にこの部分の処理を示す。

保存された写真データを特定し、抽出･整理する機能

これは、従来の生産管理システム（Y-VIS）への機能追加で対応した。生産実績を、特定の条件で絞り込む機能は従来から使われているが、新たに追加された写真の保存実績のデータベースとのリレーションにより、該当する写真ファイルのフルパスを取得し、任意のフォルダにファイルコピーできる。また、抽出されたファイルをImageコンポーネントにそれぞれ割り付けて表示しているが、この部分では、特に目新しい機能は使用していない。なお、画面上で選択した写真はNewtone社製品のImageKitを使い、任意の拡大、パンウインドウ表示などの機能を実装した。操作画面のイメージを【図4】～【図6】に示す。

iOSアプリ使用のポイント、注意点

iOSでIBM iに接続し、機能を利用するにはDataSnapサーバーのDataSnapアプリを介する必要があり、DataSnapアプリを常時起動させておく必要がある。

当社ではこれを、フォームアプリケーションではなくサービスアプリケーションとしているが、何らかの原因でフリーズまたは終了しているケースがあったため、タスクスケジューラで定期的にbatファイルによる再起動等を行っている。

サービスの再起動は、下記の2行で実行される。

```
net stop "ServerContainer1"
net start "ServerContainer1"
```

また、複数の端末がDataSnap経由でアクセスする場合は、セッションの輻輳により処理が停止することがあるので、DataSnapサーバーアプリ側のTSQLConnectionのパラメータで、マルチセッションの使用可否を決める［Decimal Separator］をYに設定することが重要であり、またアクセス前後での接続､切断を漏れなく行う必要がある。【ソース4】にDataSnapサーバーとの接続、切断の処理方法を記述した。

アプリ更新の通知と作業による確実な更新処理の実装も、端末を配布した後の運用では重要である。

当社では、アプリ内の選択機能の切り替え時に、サーバーで管理する最新版バージョンと自アプリのバージョン情報を比較し、バージョンが上がっている場合は更新用のWebページが開き、更新を促す。更新が試されないと毎回、リンクページが開くことになる【図7】。なお、サーバーでの最新版確認は、DataSnapアプリに実装した関数で取得している。

開発用のiPod端末で検証されたアプリはアプリケーションサーバーに保存し、各iPod端末のブラウザのリンクよりインストールあるいは更新するが、iOS 7.1以降は社内配布用Webサーバーの SSL証明書をあらかじめ各端末に導入しておく必要がある。この手続きはMac、サーバー等で行うが、詳細な手順は、ネット情報またはミガロ．のサポートなどに確認が必要である。また開発ライセンス、証明書等には期限があるので定期的な更新を要する。

開発ライセンス登録～インストール、配布に必要な手続き、設定と各構成機器の関連を【図8】にまとめるが、高いセキュリティと厳密なライセンス管理のために、かなり煩雑な手順を踏む必要がある。

システム立上げ時に発生した問題点

システム立上げ時に発生した主なトラブルについて、【表3】にまとめた。

QRコードと1次元バーコードの併用への対応

ラベルは、既存のバーコードリーダーに対応するものや、QRコードを表記しないものが混在しているため、バーコード読み取り時の処理には工夫が必要である。通常、QRコードは1次元バーコードよりも多くの文字情報を保有できるので、さまざまな運用を想定して記載する情報を決定している。

同一の帳票に印字されたQRコードと1次元バーコードを同じ用途で使用する場合、1次元バーコードの情報は、QRコードでは先頭に配置する方がよい。実際にQRコードと1次元バーコードを併用している例を【図9】に示す。

iOSで5250エミュレータ使用

当社では従来、棚卸しをはじめとしてさまざまな用途でハンディターミナルを利用してきたが、機能追加、改修時の容易さ、リアルタイム性等の理由から、ハンディターミナルに5250エミュレータを搭載し、IBM iに直結する方式で運用してきた。

今回の検討にあたっても、iPodに5250エミュレータを搭載しIBM i上の従来資産を活用して業務が行えるのであれば、積極的に取り組みたいと考えていた。

調査の結果、iPodで利用可能な5250エミュレータとして株式会社ヒットの「WaveLink TE」があることがわかり、

写真2　写真撮影から保存までの画面

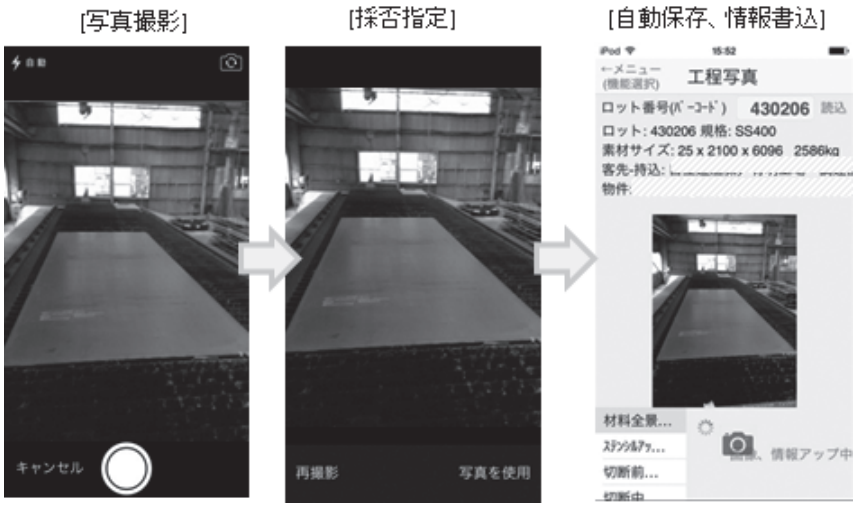
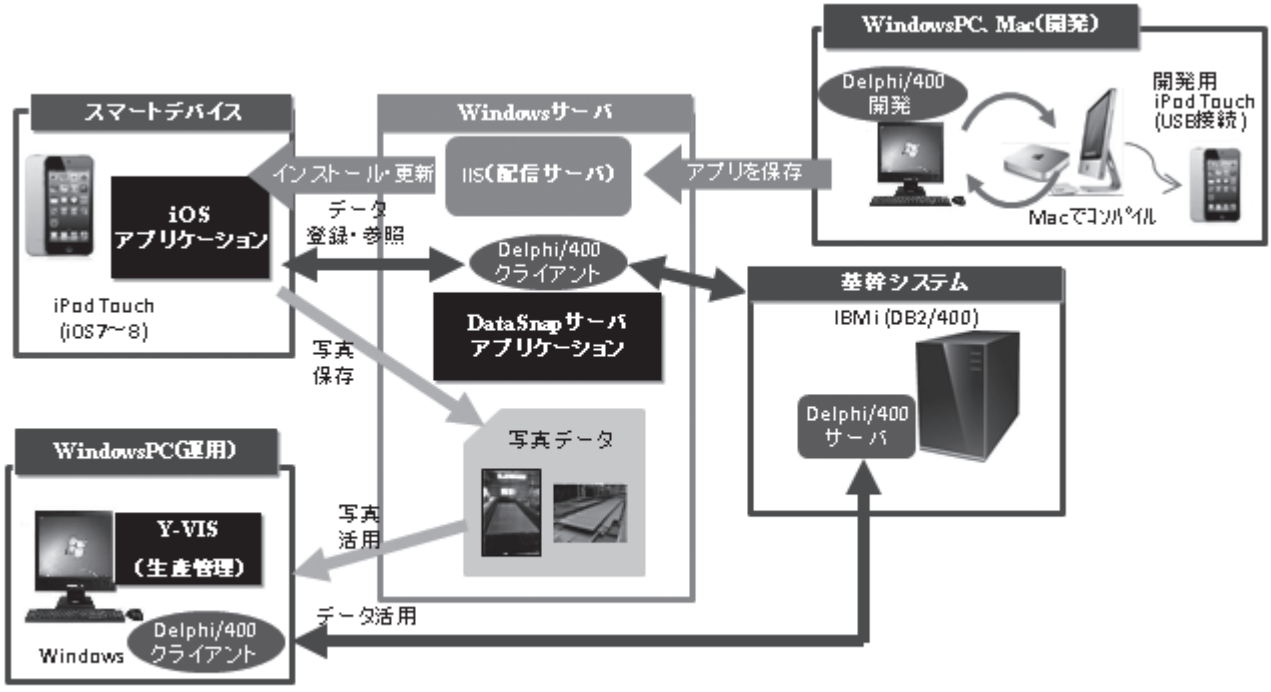


図2　iOSデバイスに関連するシステム機器構成



ソフトキーを数字キー主体の現場仕様にカスタマイズできれば、業務で利用可能と判断した。

ただし、ハンディターミナル用の既存フォームと処理コードは1次元バーコード用に作られているため、iPodでQRコードを読み取ると、保有する情報量の違いにより、入力フィールドの桁数を超えた文字が、次のフィールドまたは同じフィールドに書き込まれてしまう。

「WaveLink TE」ではこのような場合、オーバーフロー分を切り捨てる、または分割して次のフィールドに書き込む、といった設定をオプションで選択できるので、柔軟な対応が可能であった。【図10】

また、ハンディターミナルの場合、QRコード等のスキャンはボタン操作であり、照射部分がレーザー光などで確認できるが、「WaveLink TE」では、画面のソフトキーでカメラを起動し（【図11】）、表示部分で対象を確認、ピント合わせで読み取り、という手順を踏むため、最初は慣れを要した。iPodは、ハンディターミナルと比べて、携帯性、バッテリーの継続時間、コストなどで有利なため、各用途での運用可否を見極めていく予定である。

5250エミュレータの業務機能のGUI化

従来の5250エミュレータ画面と、同じ機能をDelphi/400 XE5で開発したアプリ操作画面との比較を【図12】に示した。この機能は、製品の生産後に残った鋼材の寸法、形状を登録する処理であるが、Delphi/400 XE5のGUIアプリの方が、入力フィールドの間違い、寸法位置の勘違いなどのミスが起こりにくく、直感的な操作が可能である。この例のように、従来の5250画面操作で処理しにくい機能については、Delphi/400 XE5で作り直しを検討したい。

コミュニケーション機能（Skype）検証

今回の目的は業務アプリの開発であり、通話は主目的でないため、毎月の通話料発生を避けるためにiPod Touchを選択したが、無料のアプリ「Skype for iPhone」を使って通話を含めたコミュニケーションが行えないか検証した。

検証の結果は下記の通りである。

◎通話機能

Skypeによる工場内での通話は、着信の認識、通話音量、音声出力位置などの点から、現状のままでは実用的でないことがわかった。具体的には、呼出音が工場内ではほとんど聞こえず、バイブレーションの機能もないため、着信に気づかないことが多い。ただし着信履歴は画面上に残るため、事後の折り返し電話は可能である。なお、iPodのマイク、スピーカの位置は通常のスマートフォンと異なるため通話時は注意が必要で、上下、表裏それぞれ逆の方がよい（【図13】）。またマイク付きイヤホン、またはヘッドセットの利用が推奨されている。

◎メッセージ機能

緊急を要さない場合については、メッセージ機能は有効である。使い方はスマートフォンなどのショートメールやLINE等と同様で、簡単である（【図14】）。写真もカメラ連動でその都度送付できるので、情報を視覚的に伝えたい場合に便利であり、さらに必要であればビデオ通話も可能なので、状況をより具体的に伝えることができる。

今後の狙い

今回、Delphi/400 XE5へのバージョンアップと新しい開発環境の整備にあたって、予算申請の際に課題として挙げた、日報（業務時間報告）、鋼材の置場管理、作業予定開示などの機能については、参照させる情報量に応じて、iPodまたはタブレット端末（iPad、Windows Tabletなど）を視野に入れて開発し、業務の効率化、新しい価値の創造につなげていきたいと考えている。

M

ソース1　バーコード読み取りりのコード記述例

```
//TMS社のFreeコンポーネントであるTTMSFMXZBarReaderを使用

// TMS社FreeTool　一番下のページ
 //http://www.tmssoftware.com/site/freetools.asp

①TMSFMXZBarReaderコンポーネントをフォームに配置します。

②バーコードリーダーの起動（Editの Onclickイベント等）
  TMSFMXZBarReader1.Show;

③TMSFMXZBarReaderの GetResultイベントで値をセット
//読み取れたらこのイベントが自動的に動く
  procedure TForm1.TMSFMXZBarReader1GetResult(Sender:  TObject; AResult: string);
  begin
     Edit1.text := AResult;
  end;


2回目のバーコード読込が完了しない現象に対して、動的にTTMSFMXZBarReaderを生成するよう変更した。

//グローバル変数としてTMSFMXZBarReader を宣言
 private
  { private宣言 }
  TMSFMXZBarReader :TTMSFMXZBarReader;

//バーコード撮影時にTMSFMXZBarReader を生成
procedure TForm2. Edit1Click(Sender: TObject);
begin

//TMSFMXZBarReader が存在すれば解放
 if Assigned(TMSFMXZBarReader1) then
 begin
  TMSFMXZBarReader1.Free;
 end;

 //TMSFMXZBarReaderを生成
 TMSFMXZBarReader1 := TTMSFMXZBarReader.Create(Self);
 //作成済みのイベントを設定
 TMSFMXZBarReader1.OnGetResult := TMSFMXZBarReader1GetResult;
 //TMSFMXZBarReaderを実行
 TMSFMXZBarReader1.Show;
end;
```

図3　SQLのコードの静的設定の比較

ソース2　SQLのコード記述の違い

ClientDataSetのCommandTextの場合

```
TestCDS.CommandText:=
 'SELECT  L.LO2001,L.LO2027,L.LO2017,L.LO2018,H.JU1012,TRIM(K.KAK06K)||''~''||TRIM(H.JU1017) AS KAK,Z.SOZ007,Z.SOZ008,J.SJU005'
 +' FROM WRKLIB/LOTF2P L'
 +' LEFT JOIN DTALIB/JVUHADP H ON L.LO2005>0 AND H.JU1099=LEFT(L.LO2004,1) AND H.JU1003=CAST(RIGHT(L.LO2004,5) AS NUMERIC(5)) '
 +' LEFT JOIN DTALIB/JVUMEIP M ON L.LO2005>0 AND M.JU2002=LEFT(L.LO2004,1) AND M.JU2003=CAST(RIGHT(L.LO2004,5) AS NUMERIC(5)) AND M.JU2004=L.LO2005 '
 +' LEFT JOIN MSTLIB/KYAKUP K ON K.KAK004=H.JU1013 AND K.KAK005=H.JU1016 '
 +' LEFT JOIN WRKLIB/SOZSUP J ON J.SJU001=L.LO2001 LEFT JOIN MSTLIB/SOZUKEP Z ON Z.SOZ016>0 AND Z.SOZ092=J.SJU003 '
 +' WHERE L.LO2001='+Trim(LotNumEdit.text)+' AND L.LO2005>0 ';
```

Queryのの場合

```
 TestQuery.SQL.add('SELECT  L.LO2001,L.LO2027,L.LO2017,L.LO2018,H.JU1012,TRIM(K.KAK06K)||''~''||TRIM(H.JU1017) AS KAK,Z.SOZ007,Z.SOZ008,J.SJU005');
 TestQuery.SQL.add('FROM WRKLIB/LOTF2P L');
 TestQuery.SQL.add('LEFT JOIN DTALIB/JVUHADP H ON L.LO2005>0 AND H.JU1099=LEFT(L.LO2004,1) AND H.JU1003=CAST(RIGHT(L.LO2004,5) AS NUMERIC(5)) ');
 TestQuery.SQL.add('LEFT JOIN DTALIB/JVUMEIP M ON L.LO2005>0 AND M.JU2002=LEFT(L.LO2004,1) AND M.JU2003=CAST(RIGHT(L.LO2004,5) AS NUMERIC(5)) AND M.JU2004=L.LO2005 ');
 TestQuery.SQL.add('LEFT JOIN MSTLIB/KYAKUP K ON K.KAK004=H.JU1013 AND K.KAK005=H.JU1016 ');
 TestQuery.SQL.add('LEFT JOIN WRKLIB/SOZSUP J ON J.SJU001=L.LO2001 LEFT JOIN MSTLIB/SOZUKEP Z ON Z.SOZ016>0 AND Z.SOZ092=J.SJU003 ');
 TestQuery.SQL.add('WHERE L.LO2001=' + Trim(LotNumEdit.text) + ' AND L.LO2005>0 ');
```

ソース3　写真アルバム保存時のコード

```
if PhotoAlbumYNSwitch.IsCheckedthen
begin
CRImage:=BitmapToUIImage(image); //CRImage:UIImage→varで宣言
UIImageWriteToSavedPhotosAlbum(
  (CRImage asILocalObject).GetObjectID,nil,nil,nil);
 CRImage.release;//これが無いとすぐにクラッシュ
end;
```

図4　抽出した写真の一覧表示画面

図5　抽出した写真の要否選択

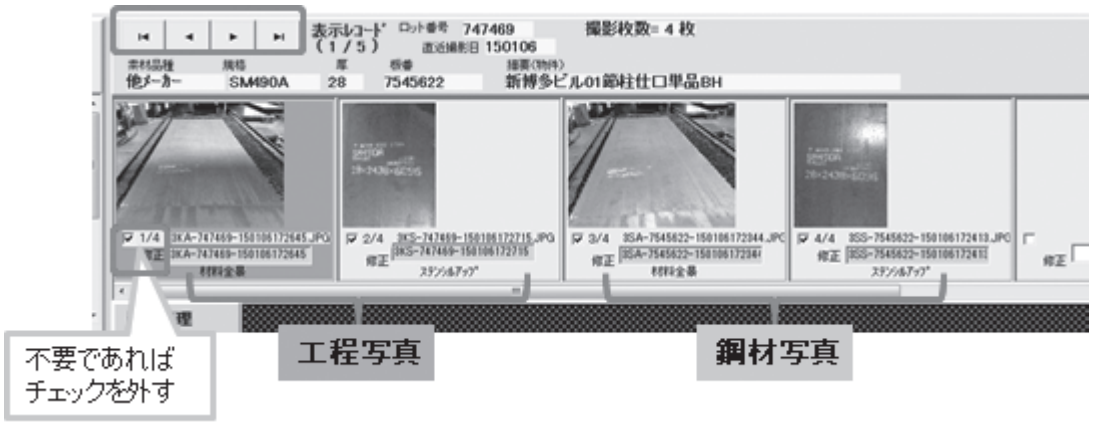


図6　写真のファイル名の自動整理機能

ソース4　DataSnapサーバー接続・切断のコード

```
 functionTForm2.SvrConnectChk:boolean;
begin
  try
   SQLConnection1.Connected := False;
   SQLConnection1.Connected := True;
   result:=true;
 except
   showmessage('スナップサーバー接続できません、連絡のこと');
   result:=false;
   abort;
 end;
end;

function TForm2.SvrDisConnect:boolean;
begin
 try
   SQLConnection1.Connected := False;
   result:=true;
 except
   result:=false;
   abort;
 end;
end;
```

図7　アプリ更新のリンクページ

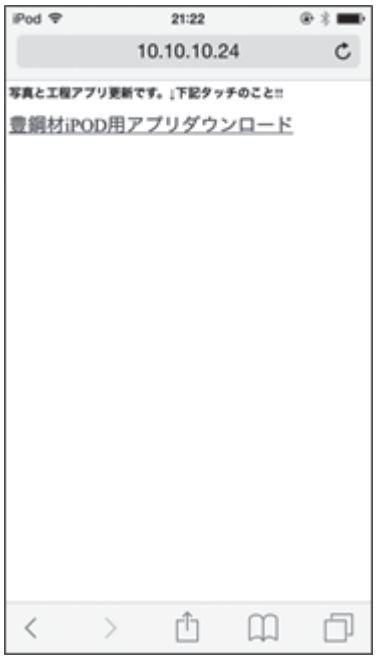


図8　開発ライセンス登録～配布に必要な設定等のイメージ

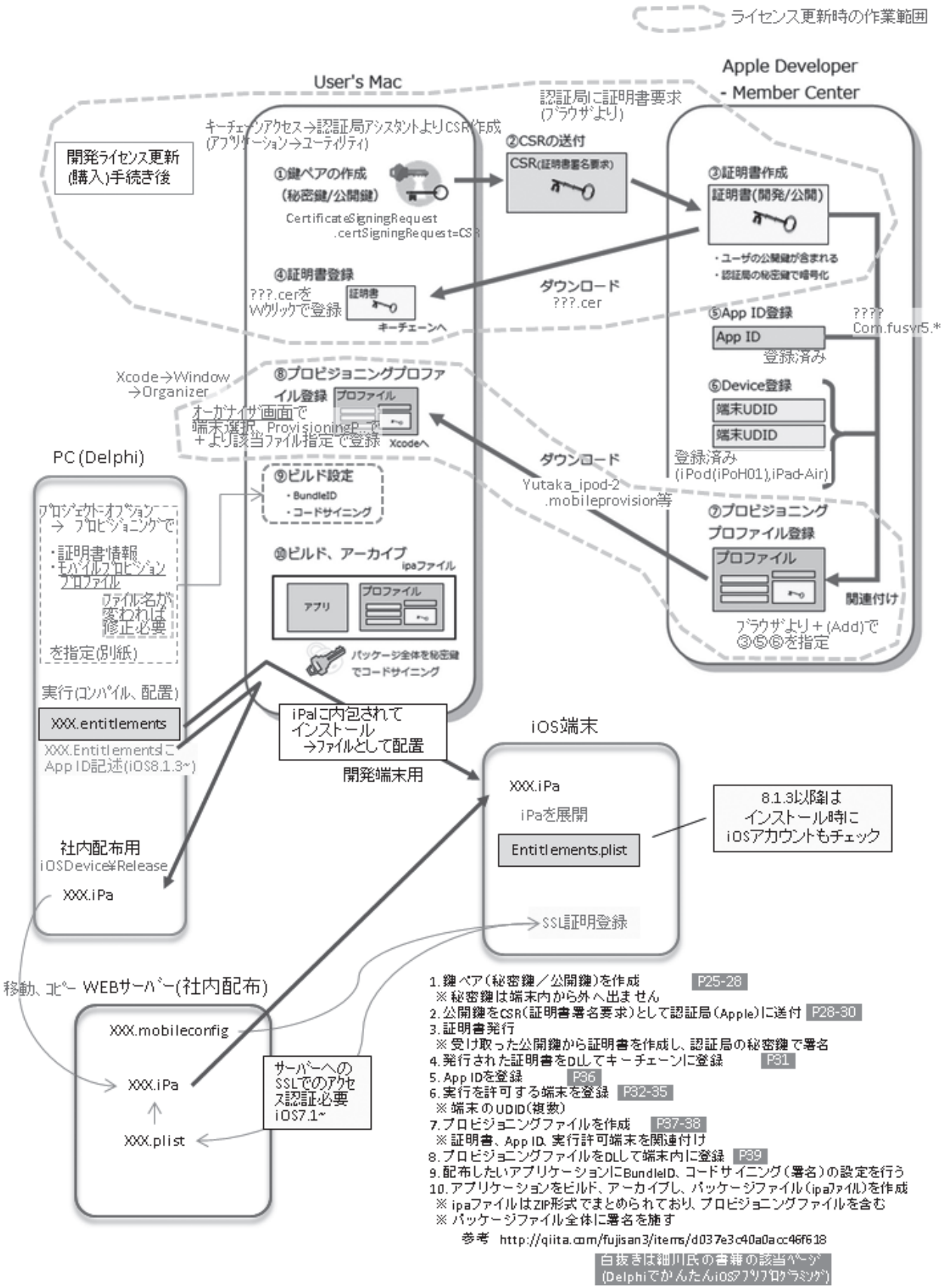


表3　立上げ時の主なトラブル

項目	内容
ｻｰﾊﾞｰｱｸｾｽのﾀｲﾐﾝｸﾞで固まる 一つの端末で固まると以降は別端末でも固まってしまう (DataSnapｻｰﾊﾞｰ機能の再起動が必要)	【原因】 DataSnapｻｰﾊﾞｰ側のﾏﾙﾁﾀｽｸ処理対応が設定されていない 【対応】 ①DataSnapｻｰﾊﾞｰｱﾌﾟﾘ側のTSQLConnectionのﾊﾟﾗﾒｰﾀ[Decimal Separator]を Y に設定 →以降良好 ②iPodからもDataSnapｻｰﾊﾞｰの再起動可能とするｱﾌﾟﾘを追加開発
撮影後にｱﾌﾟﾘが突然終了してしまう	【原因】撮影した写真ﾃﾞｰﾀをiPod内のｱﾙﾊﾞﾑに自動保存する処理でﾒﾓﾘｰを消費していた 【対応】該当部分でのﾒﾓﾘｰをｸﾘｱするよう修正 → 以降良好
ｻｰﾊﾞｰへのﾃﾞｰﾀ保存に時間がかかる	【原因】 無線LANの通信が遅い可能性 【対応】 無線LANの中継機を追加　→ 以降良好
DataSnapｻｰﾊﾞｰのｻｰﾋﾞｽが停止、ﾌﾘｰｽﾞ	【原因】 不明 【対応】 定期的な再起動(ｽｹｼﾞｭｰﾗで自動処理) → 様子見中
アプリの更新、ｲﾝｽﾄｰﾙが出来ない	【原因】 iOS8.1.3以降の仕様変更 【対応】 XXX.EntitlementsにAppIDを記述　→ 以降良好
アプリの更新、ｲﾝｽﾄｰﾙが出来ない (更新ﾍﾟｰｼﾞのﾘﾝｸﾀｯﾁで処理が開始せず)	【原因】 不明 【対応】 iOSを立上げ直して再トライ → 不能時は再トライ

図9　QRコード、一次元バーコードを併用での情報例

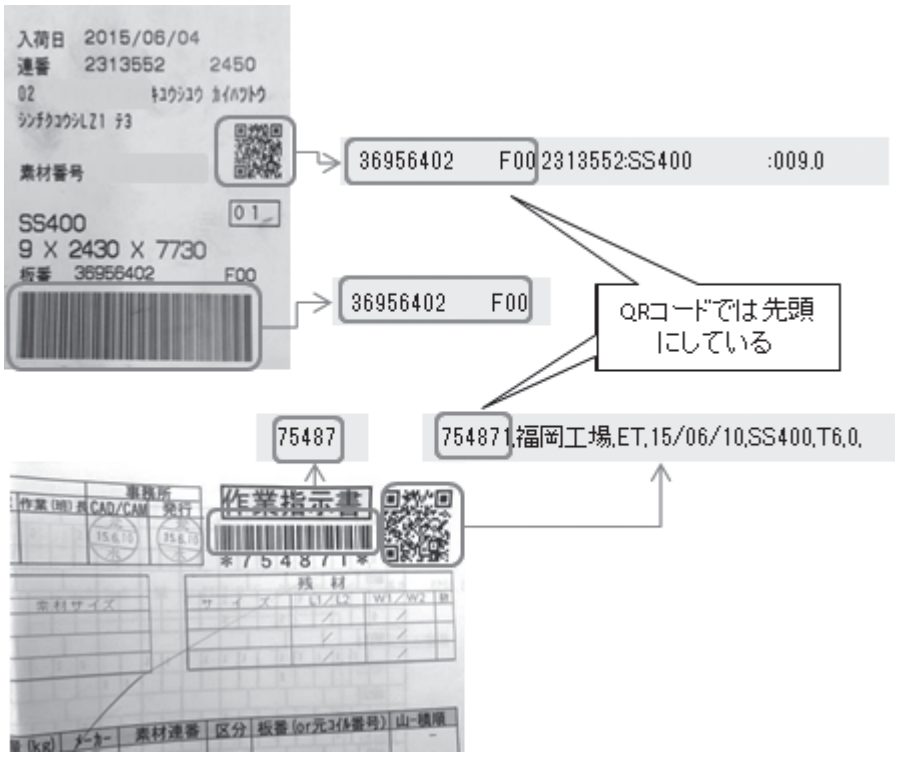


図10　WaveLink TE（5250エミュレータ）の文字数オーバー時の設定

図11　WaveLink TEでのバーコード読取起動

図12　同等機能での5250画面とGUI画面比較

図13　iPod Touchのマイク、スピーカーの位置

http://akiueo.blogspot.jp/2012/09/ipod-touchiphone5.html より

図14　iPod Touchのメッセージ機能使用例

ゴールド賞

ブランク加工図管理システムの構築
―入力インターフェースの可能性を求めて

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。特に、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。

はじめに

澁谷工業株式会社（以下、当社）は、「カスタマーファースト」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」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」事業にも進出している。

当社のホストコンピュータの変遷は、S/32 から始まり現在の PureFlex System に至っている。そして長い歴史の中で、基幹システムの多くを 5250（エミュレータ）画面上で構築してきた。しかし、新たにシステムを構築する場合は、Delphi/400 と Delphi を主に利用している。

当社における基幹システムは、すべて当部署で開発している。しかし最近は、ICT 知識を持つエンドユーザーが Office 製品などを使って、自ら業務システムを管理することが増えてきた。

ところが、データ量の増加などによって運用／管理が困難になると、当部署にシステムの再構築を依頼してくるケースも多くなっている。

本稿は、そうしたケースの１つである「ブランク加工図管理システム」（以下、当システム）の構築例である。既存のブランク加工図ファイルの有効利用と、入力インターフェースを工夫して構築したシステムの内容を紹介する。

前提

ブランク加工図とは、部品を加工する前の素材図面のことである。前提として、完成部品はブランク加工図の作成以前にあらかじめ設計されてあり、その品番、品名が IBM i 上の基幹システムに登録されている。完成部品とブランク加工図の関係は、1 対 N 個に紐付いており、完成部品の品番、品名を基に、ブランク加工図のファイル名を決定している。

現状

最初に、エンドユーザーが行っていた処理の概要を説明する。

管理者

1. CAD ソフトでブランク加工図作成
2. アクセス制限付きファイルサーバーに、品名で「あかさたな」別にフォルダー管理
3. ブランク加工図に品番を含むファイル名を付け、保存　【図 1】

利用者

1. 基幹システムより、該当ブランク加工図の品名／品番を検索
2. 該当フォルダー内で図面検索
3. 2 より、基幹システム連携　【図 2】

以上をまとめると、エンドユーザー側では手作業による運用を行っていたため、IBM i 上のマスター（完成部品の品番・品名）は台帳の役割しか果たしておらず、ファイルサーバー上のブランク加工図とシステム的な連携がとれていなかった、ということである。

当初は、管理すべきブランク加工図の件数が少なかったため、あまり問題にならなかった。しかし、登録件数の増加に

図1

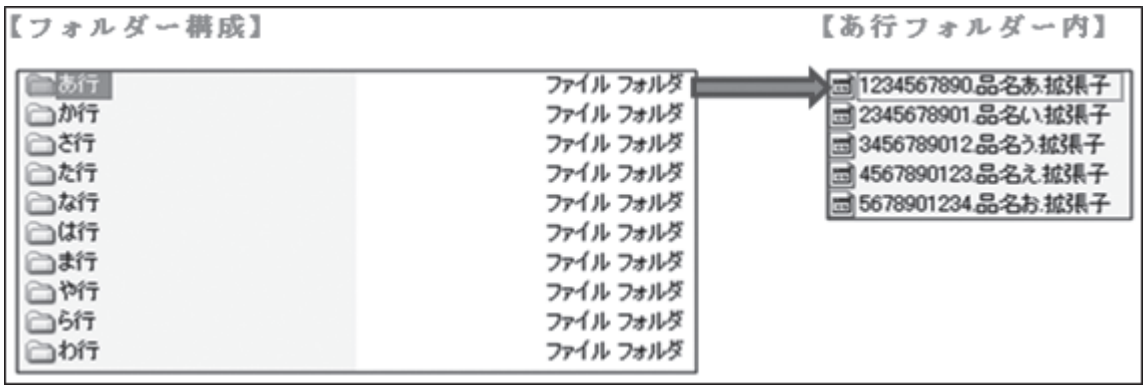

【フォルダー構成】
【あ行フォルダー内】
あ行 ファイル フォルダ
か行 ファイル フォルダ
さ行 ファイル フォルダ
た行 ファイル フォルダ
な行 ファイル フォルダ
は行 ファイル フォルダ
ま行 ファイル フォルダ
や行 ファイル フォルダ
ら行 ファイル フォルダ
わ行 ファイル フォルダ
1234567890.品名あ.拡張子
2345678901.品名い.拡張子
3456789012.品名う.拡張子
4567890123.品名え.拡張子
5678901234.品名お.拡張子

図2

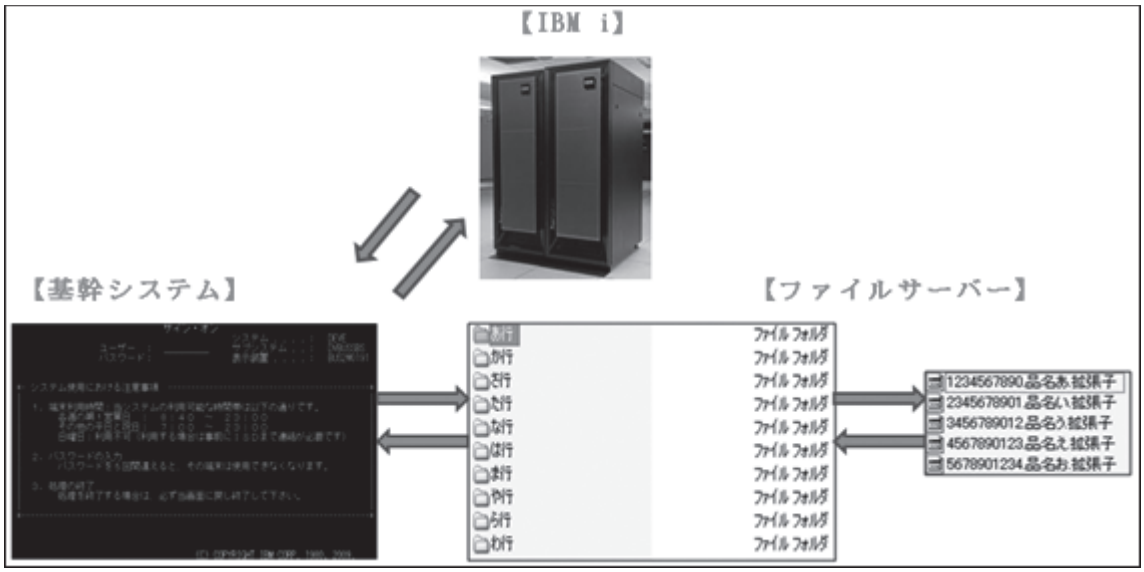
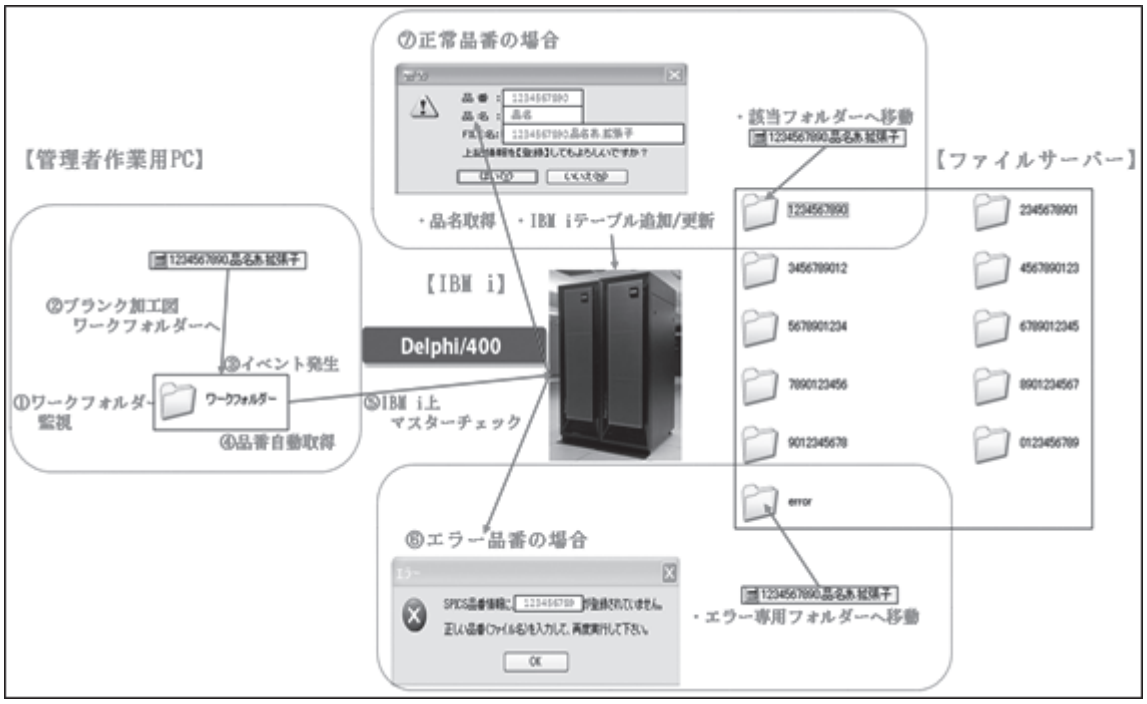

【IBM i】
【基幹システム】
【ファイルサーバー】
1234567890.品名あ.拡張子
2345678901.品名い.拡張子
3456789012.品名う.拡張子
4567890123.品名え.拡張子
5678901234.品名お.拡張子

図3


⑦正常品番の場合
・該当フォルダーへ移動
【管理者作業用PC】
【ファイルサーバー】
・品名取得
・IBM iテーブル追加/更新
【IBM i】
1234567890.品名あ.拡張子
②ブランク加工図
ワークフォルダーへ
③イベント発生
Delphi/400
①ワークフォルダー
監視
ワークフォルダー
④品番自動取得
⑤IBM i上
マスターチェック
⑧エラー品番の場合
・エラー専用フォルダーへ移動
error

伴い運用／管理の負荷が増大し、管理者側では「登録ミス」の頻度が高くなり、利用者側では「検索時間の増加」と、検索しても「間違った図面が表示される」などの問題が起きていた。

このような状況の中で、当部署にシステム構築の依頼がきた。

現状分析

問題を整理するために、まず現状分析を行った。その内容が以下である。

・ブランク加工図を作成／照会するCADソフトが複数ある。
・管理者は、基幹システムで品名／品番を検索後、手作業でファイル名を付け、ファイルサーバー上のフォルダー内に保存する。
・利用者は、基幹システムで図面情報を検索し、手作業でファイルサーバー上のフォルダーを参照後、該当する品番を検索する。
・その後、基幹システムと連携する。

当部署では、非常に無駄の多い運用であると判断し、再構築へ向けて関係者とミーティングの場を持った。

関係者との意見交換

意見交換の場で出されたエンドユーザーからの要望をまとめると、以下の通りになる。

（ア）簡単に、ブランク加工図の登録処理を行いたい。
（イ）正確に、ブランク加工図の登録処理を行いたい。
（ウ）簡単に、品番および品名で検索を行いたい。
（エ）簡単に、複数のCADソフトで作成したブランク加工図を照会したい。
（オ）簡単に、完成図面を照会したい。
（カ）簡単に、各種基幹システムとの連携を行いたい。

システム構築前に考えたこと

（ア）（イ）に関して。一般的な登録方法は、該当品番を手入力し、エラーチェック後、各種登録処理を行うことである。しかし当社では、作成済みのブランク加工図だけでも数千点ある。また、今後の新規登録を考えると、このオペレーションをエンドユーザーに依頼するのはあまりにも負荷が大きく、現実的でない。

（ウ）に関して。検索の都度、ファイルサーバーにアクセスして該当ファイルを見つけるのは可能である。しかし、この方法は効率が悪い。

検討の結果、上記をいかに吸収するかが、当システム構築の成功のカギと考えた。

打開策

（ア）については、既存のブランク加工図ファイルを有効利用したいところである。また（イ）を実現するには、IBM i上の基幹システムとの連携が必須になる。

そこで、実現性はひとまず度外視して、次のようなプロセスを検討した。【図3】

①システムで、ワークフォルダー内を監視する。
②既存のブランク加工図ファイルを、ワークフォルダーに保管。
③②の処理後、イベントを発生させる。
④ブランク加工図ファイル名から、品番を自動取得する。
⑤IBM i上のマスターによる妥当性チェック、および品名取得を行う。
⑥エラー品番の場合、エラーメッセージを表示。その後、該当ファイルを、エラー専用フォルダーに移動する。
⑦正常品番の場合、IBM i上のブランク加工図管理テーブルを更新後、ファイルを、ファイルサーバー上の該当フォルダーに移動する。

IBM iとの連携は、Delphi/400があるので問題ない。しかし当時、①の制御方法がわからなかった。

そこで、いろいろと調査した結果、Windows APIを使ってフォルダー内を監視できることがわかった。また、インターネットでも（*注）有益な情報を取得でき、この問題を解決できた。

*注　http：//bit.ly/migaro08_02

システム概要

構築したシステムは、以下の通りである。

当部署
①初期設定として、エンドユーザー側管理者の登録【図4】

管理者【図5】【図6】
①他管理者登録
②ワークフォルダー設定
③ブランク加工図保管フォルダー設定
④③のアクセス権限設定
⑤②に既存登録ブランク加工図ファイル保管（ドラッグ＆ドロップ可）
⑥IBM i上マスターによる妥当性チェック
⑦エラー品番の場合、メッセージ表示後、図面をエラーフォルダーに移動（品番修正後、再度ワークフォルダーへ移動）
⑧正常品番の場合、図面を③に移動後、IBM i上ブランク加工図管理テーブル更新

管理者／利用者【図7】【図8】【図9】
①CADソフト登録
②図番／品名による検索
③指定CADソフトによるブランク加工図照会 ／拡張子（PDF等）によるファイル照会
④完成図面照会
⑤基幹システム連携

構築後の考慮点

構築したシステムに対するエンドユーザーの評価は上々であった。しかしある時、保管図面とIBM iデータの差異を発見した。原因は、システムを利用せずに図面登録した、例外オペ―レーションの結果である。万が一、例外オペレーションを行うと、整合性がとれなくなる。

そこで打開策として、保管図面とIBM iデータをチェックし、その差異を照会することにより、この問題を回避した。そして運用ルールを再度徹底したことも幸いし、それ以降現在まで、問題なく運用が継続している。

さらに、ワークフォルダーを監視していない状況で、ワークフォルダー内に各

図4


TBL01B
変　更
◇　テーブルマスタ登録　◇
TABLE-KEY
テーブルKey
説　明
ブランク加工図管理者
有効期間
～ 99/99/99
TABLE-CD
TABLE-CD 説明
USERID01
小山　祐二

図5


①他管理者登録
ブランク加工図管理者登録(IMC21HFOM)
検索項目
ユーザーID
SBY00000
社員名
№
ユーザーID
社員名
1
USERID01
小山　祐二
2
USERID02
他　担当者

図6

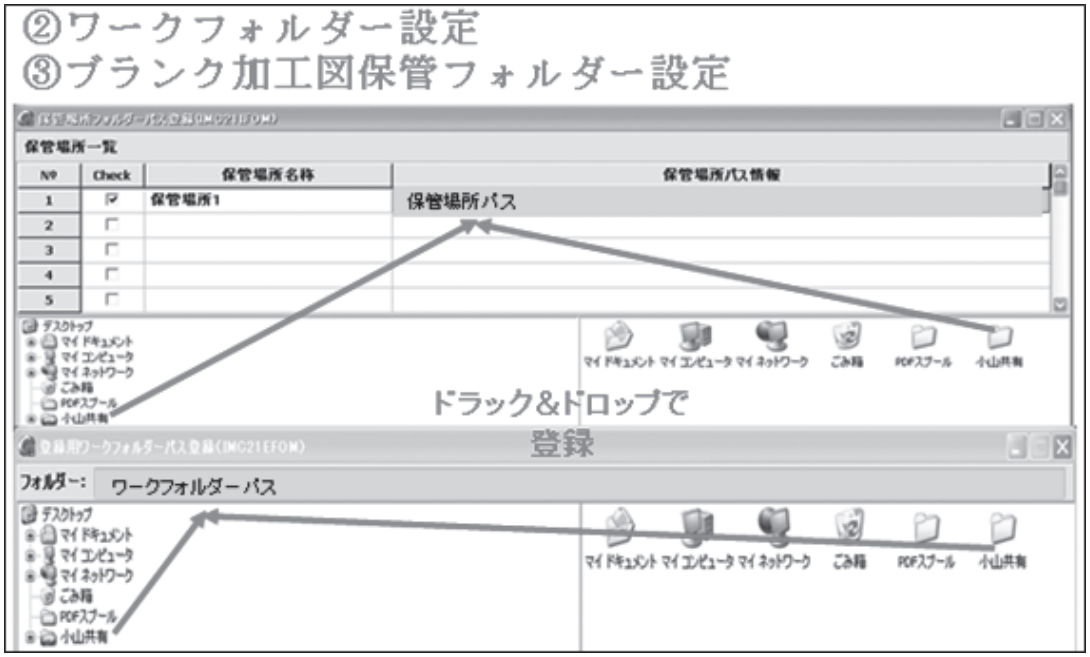
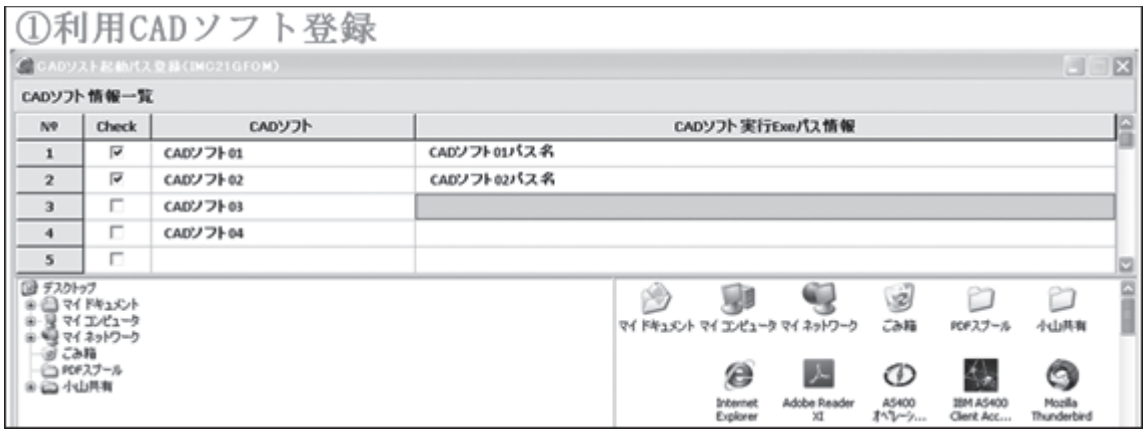

②ワークフォルダー設定
③ブランク加工図保管フォルダー設定
保管場所一覧
保管場所名称
保管場所パス情報
保管場所1
保管場所パス
ドラック&ドロップで
登録
フォルダー:
ワークフォルダーパス

図7


①利用CADソフト登録
CADソフト情報一覧
CADソフト
CADソフト実行Exeパス情報
CADソフト01
CADソフト01パス名
CADソフト02
CADソフト02パス名
CADソフト03
CADソフト04

ブランク加工図ファイルが保管されることも想定し、ワークフォルダー内ブランク加工図の「一括取り込み」機能も追加した。【図 10】

最後に

構築にあたって当初、エンドユーザー側から「データの蓄積はエンドユーザー側で少しずつ行う」との提案があった。再構築の担当者としては、その言葉に甘えたい気持ちもあったが、その蓄積作業を自分が行うことを考えると、既存のブランク加工図ファイルを利用して、蓄積作業の負荷を軽減したいと考えた。それが、今回のシステムにつながっている。

現在、システムのトレンドは「作る」から「使う」にシフトしつつある。私はその動向を全否定するものではないが、それが加速した 5 年後、10 年後を想像すると、企業システムに差がなくなるのではないかと思える。つまり、システムの差別化によって他社より優位に立つことができなくなるため、経営における ICT の付加価値は低くなる。そして、「使う」ことがメインとなるため、企業システムを創造的発想でデザインできる人が少なくなる。そうなると、システム化の難易度が高い場合、できない理由を並べて、「だからできない」と結論付ける人が多くなるのではないかと思う。

私は、「どうすればできるか」を皆で考え、できないと思ったことでも、できる道筋をファシリテートできるよう、日頃から心がけていきたいと思っている。また、そんな考えを持つ後輩を育成していきたいとも考える。今回の論文が、皆様の気付きとなれば幸いである。

M

図8

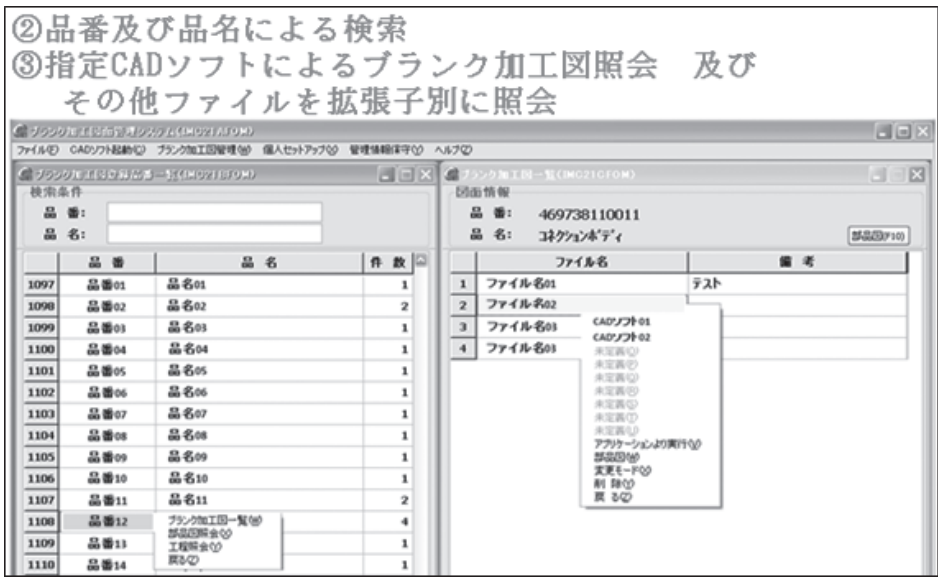

②品番及び品名による検索
③指定CADソフトによるブランク加工図照会　及び
その他ファイルを拡張子別に照会

図9

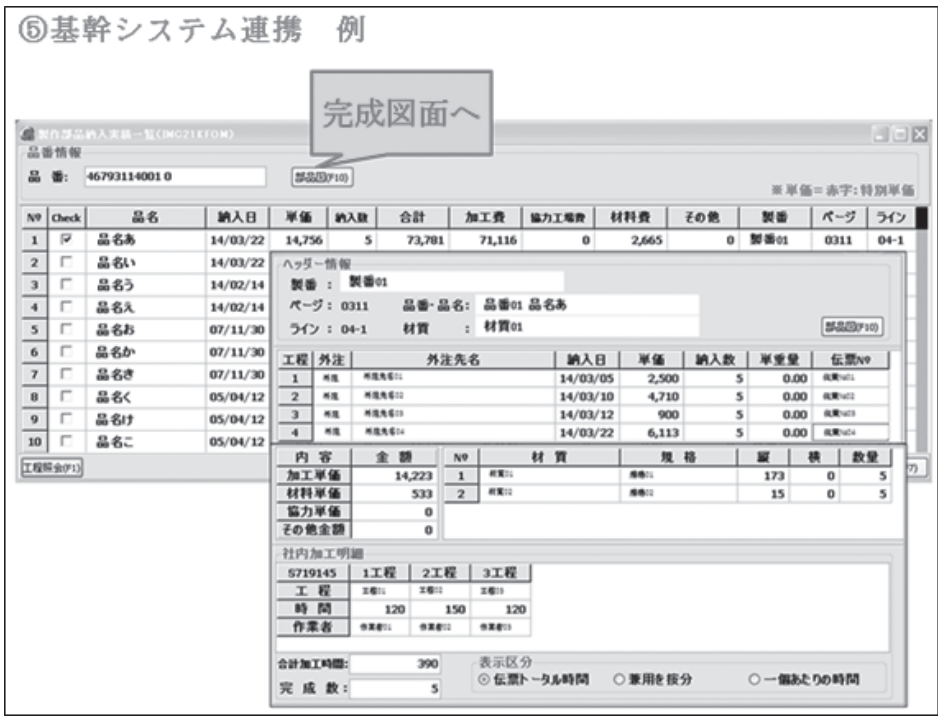

⑤基幹システム連携　例
完成図面へ

図10

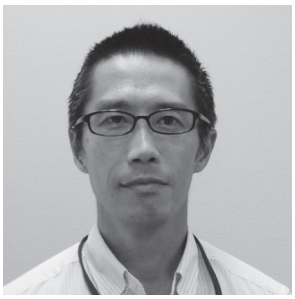

※SPICS(Shibuya Production Information Conterol System)
Delphi/400
「IBM iテーブル登録情報」
「ファイルサーバー内　登録情報」

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）

ー i5コマンドを有効に活用する

三好 誠 様
ユサコ株式会社
システムグループ



ユサコ株式会社
http：//www.usaco.co.jp/

海外の学術雑誌、書籍の輸入販売を中心に事業を展開している。特に医学、薬学等の自然科学分野に強みを持つ。近年、学術情報媒体の多様化に伴い、電子ブック、データベース、各種ソフトウェアの取り扱いを強化。学術情報を通じ、知的情報の創造、蓄積、共有による社会貢献を目指している。

1.ユサコの事業内容、および当該システム関連部門・部署の事業内容

学術情報の提供を通じ、知的情報の創造、蓄積、共有のお手伝いをする会社

当社ユサコは創業以来、海外からの雑誌や書籍の輸入販売を行っており、近年では電子ジャーナル、データベースも取り扱う。雑誌や書籍等の輸入販売というと単なる書店の取次のように思われるが、医学や化学など学術研究に関連したものを中心に、知的情報の創造、蓄積、共有を通じて社会に貢献することを目指している。

この中で当方が所属するシステムグループは、少人数ながらも受発注を中心とした社内基幹システムの管理、開発から、インフラ整備やユーザーサポートまでトータルに会社を支えている。

2.アプリケーションの開発経緯、および概要

現行 System i5 のリプレースが急務

当社の基幹システムが稼働している System i5 は現在、リプレースの必要性が急務となっている。OS、ハードウェアともに導入から月日が経過しており、パフォーマンス、保守体制ともに、最新のものに切り替える時期が来ている。

現行システムを構築している CASE ツールがネック

単に OS やハードウェアを切り替えるだけなら、十分に検証を行い、ソースやオブジェクトを移行すればよい。しかし当社で一番問題となっているのは、現行システムの開発やコンパイル環境である CASE ツールだ。当時は 5250 画面やプログラムを簡単に自動生成してくれる便利なものであったが、このツールがなんと、新しい OS では動作が保障されない。

現行 System i5 の 5250 画面から、Delphi/400 による GUI への移行の準備

上記の問題があり 5250 画面を利用したままでは画面のメンテナンスができず、新しいハードウェアにも移行できない。かといって現在のプログラムを解析して同じ 5250 画面を再生成するのでは生産性がないし、今後の大量の CL や RPG を新たにメンテナンスするには、当社のシステムグループの人数では困難である。そこで、Delphi/400 による GUI 画面への移行を通じ、CASE ツールからの解放を目指しているところである。

GUI 化による OUTQ の管理、スプールファイルの見える化も必要

通常の対話画面はもちろんだが、System i5 の OUTQ を管理する機能も Delphi/400 に移行する上で必要だ。長年 System i5 を利用しているだけあって、コマンド自体は作成した画面から呼び出しているものの、WRKOUTQ コマンドや WRKSPLF コマンドの機能操作

図1　帳票サーバー利用の構成図

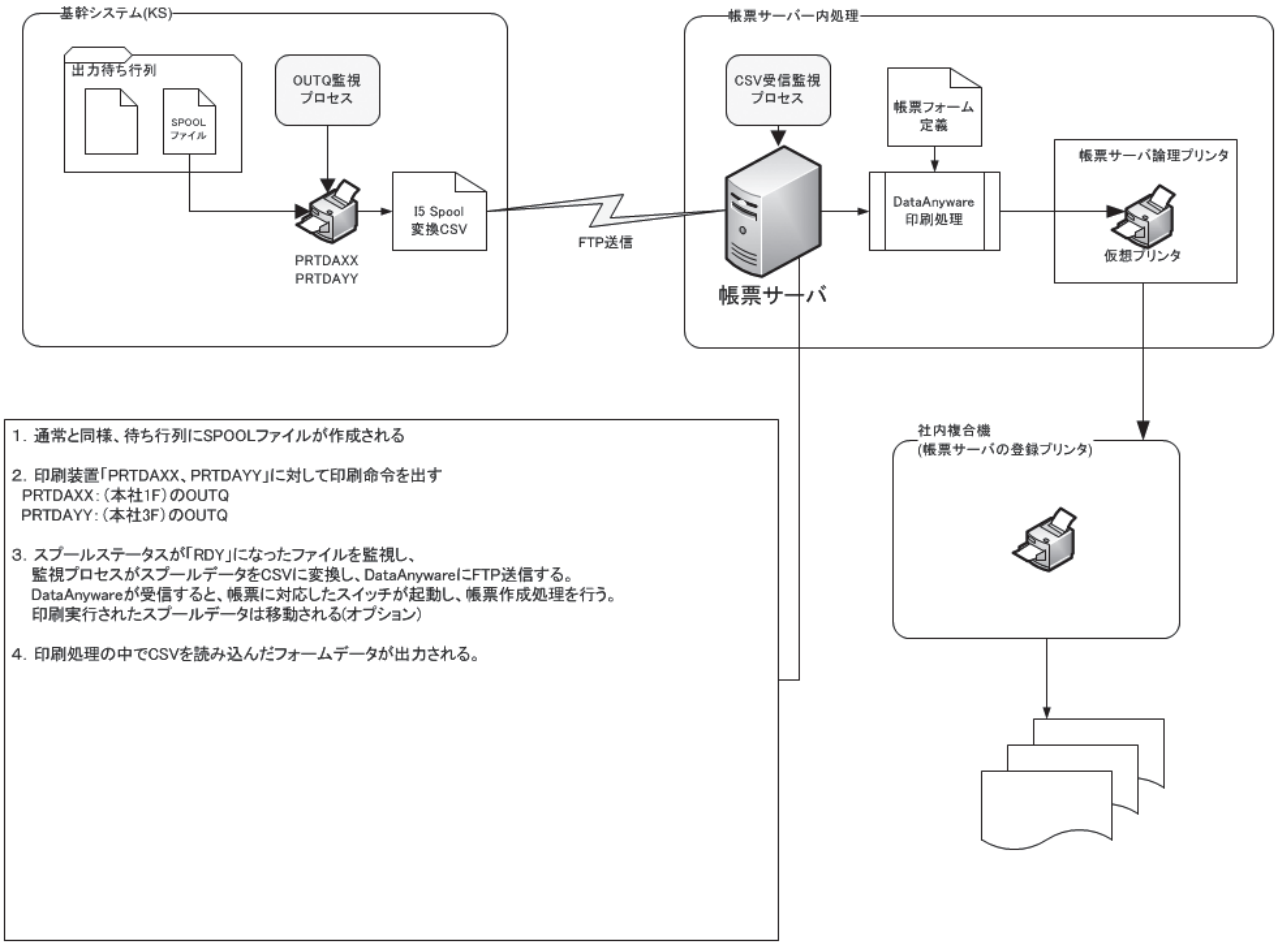
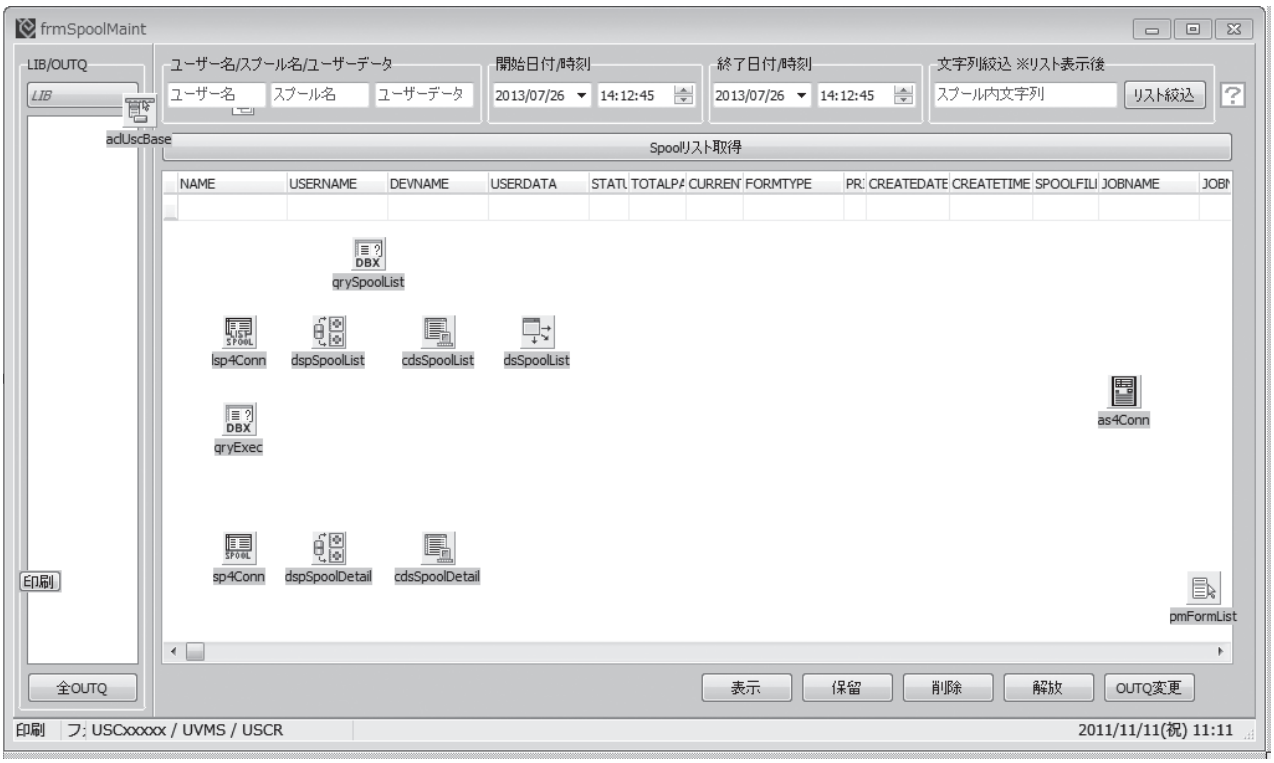


図2　ListSpool400利用時の開発画面



はユーザーも日々使用しており、熟練している。スプールファイルの移動､解放、表示などの機能は必須である。

多数のスプールファイル、OUTQ に対応できる機能要件

当社は帳票サーバーを介して一般プリンタに出力できる環境を整えており【図1】、プリンタ対応する OUTQ をそれぞれ準備しているが、そのほか保管用、出力済用に多くの OUTQ を用意している。また、当社の雑誌等商品の受注で、1 年で多くのトランザクションレコードが発生する。ゆえに注文書などを作成する際にはスプールファイルの数も多くなるし、繁忙期になると 1 つのスプールファイルに対する枚数も膨大になる。

5250 画面のようにスムーズな表示とアクションができることが必要だ。

3.取組内容（メインテーマ）

TListSpool400 ではライブラリーの指定と、スプールファイル数が多い時に課題

まず始めたのは、ツールとして提供されている TListSpool400 の活用である。非常に便利なコンポーネントであり、スプール一覧の取得に適しているため、当コンポーネントを採用して開発を始めた。スプール一覧を取得して、かつユーザー名、スプールファイル名、出力時間等で絞り込む機能を検討する。【図 2】

開発は順調に進み、いったんリリースを行ったが、検索対象が複数の OUTQ にまたがる時、またスプール件数が多い時、すべてのスプール一覧を取得してからの絞り込みしかできないため、非常に時間がかかることがわかった。複数の OUTQ を選択して OUTQ 横断検索を行える機能も備えていたが、時には数十分かかるため、実用に耐えない。

WRKSPLF の PRINT 出力を利用

i5 の最新 OS では簡単にできるのかもしれないが、スプールファイルの復元等を行う場合、CPYSPLF コマンドでスプールファイルをファイル化し、CPYF コマンドで復元印刷したりする。WRKSPLF コマンドは検索条件があらかじめ設定できることもあり【図 3】、条件さえ付加できれば結果を返すのが速い。PRINT オプションを付けることで、この速度を活かせないだろうか。

スプールの DB 化した結果を QTEMP に保存

上記 WRKSPLF コマンドを利用し、得られた結果のスプールファイルを CPYSPLF コマンドで DB 化し、QTEMP に保存する CL を作成する【図 4】。スプールファイルには当然 WRKSPLF のヘッダー等も含まれており、全体としては不定形の形だ。しかしフィールド分割したファイル【図 5】をスプールファイルのコピー先にしておけば、すべて文字列ながらエラーを発生させることなくファイルに取り込める。ヘッダーは取得時に取り除くこととする。

QTEMP に作成したファイルを Delphi から取得

ファイル化してしまえば、Delphi/400 から容易にアクセスできる。前述の通り、ファイル化したスプールファイルには不要なヘッダーデータも含まれているので、STATUS フィールドに必ず入る値で結果の絞り込み（RDY,HLD,SAV..）を行えば、スプール一覧だけに結果を絞り込める。WRKSPLF を PRINT 実行した時のスプールファイル（QPRTSPLF）も検索結果に出てきてしまうので、これも除外する SQL を準備しておく【図 6】。OUTQ だけは WRKSPLF コマンドのパラメータにないので、入力があれば条件に追加できるようにする。【図 7】

Delphi/400 から WRKSPLF コマンドの実行と、ClientDataSet への取得

SQL のベースができたら、検索実行時に WRKSPLF コマンドの実行と、取得した結果を ClientDataSet への取得を行う。【図 8】

スプール管理の一般的な機能に対応

ClientDataSet に取得さえできれば、スプールの保留から解放、移動まで容易に可能だ。取得したスプール情報から、それぞれコマンドを呼出し、実行できる機能を実装する。【図 9】

スプール文字列絞込の機能まで対応

絞り込んだスプール一覧から、印字されている文字列でフィルタリングできる機能があるとよい。スプール内容を読み取るには Spool400 コンポーネントがあるが、同様に印字ページが多いと、ClientDataSet に展開してしまうと時間がかかってしまう。文字列があるかどうかを知るだけなら、これも i5 上で DB 化したスプールファイルから文字列検索するのでよい。QTEMP にスプールデータを展開する CLP を作成し、SQL でこのファイルにアクセスする。【図 10】【図 11】【図 12】

スプール表示も CPYSPLF を使用して ClientDataSet に表示

スプール表示についても CPYSPLF を活用して代用する。ListSpool400、Spool400 は BDE を使用するが､この際、BDE 不要で Delphi/400 を動作できるようにした【図 13】。スプール文字列絞込で活用した CLP からスプールを SQL で取得する。CLP から呼び出す RPG でページ番号を振ってあるので【図 14】、この番号でフィルタ機能を利用し、疑似ページめくり機能を有効にする【図 15】。表示したスプールからも、Windows で行うような、文字列検索をできるようにした。【図 16】

検索結果の文字列の場所を覚えておき、次に検索が実行された時、次に見つかる文字列からスタートする。

4.ノウハウ、教訓、エンドユーザからの評価、今後の予定･計画

GUI 画面へは移行の途中であり、スプール管理画面もまだ、積極的には利用されておらず、ユーザーからの反応はまだ多くは得られていない。しかしながら、出力したスプールを検索できる機能については、今後大いに効率を上げられる見込みだ。GUI ベース画面への移行が進んでいけば現行のスプールを使用する必要もなくなり、Delphi/400 で利用できるレポートツールに切り替えていくことになるだろうが、当面はこの機能を役立てることになるだろう。

M

図3　WRKSPLFのパラメータ

図4　WRKSPLFの結果をQTEMPに保存するCLP

```
        PGM       PARM(&USER &USRDTA &SPLF &STDATE &STTIME &EDDATE +
                    &EDTIME &CURSS &MSGID &MSG)

/*****************************************************************/
/*表題:WRKSPLFの結果をQTEMPにファイル保存する                    */
/*****************************************************************/

        /*パラメータ*/
        DCL       VAR(&USER) TYPE(*CHAR) LEN(10)
        DCL       VAR(&USRDTA) TYPE(*CHAR) LEN(10)
        DCL       VAR(&SPLF) TYPE(*CHAR) LEN(10)
        DCL       VAR(&STDATE) TYPE(*CHAR) LEN(8)
        DCL       VAR(&STTIME) TYPE(*CHAR) LEN(6)
        DCL       VAR(&EDDATE) TYPE(*CHAR) LEN(8)
        DCL       VAR(&EDTIME) TYPE(*CHAR) LEN(6)
        DCL       VAR(&CURSS) TYPE(*CHAR) LEN(1)
        DCL       VAR(&MSGID) TYPE(*CHAR) LEN(7)
        DCL       VAR(&MSG) TYPE(*CHAR) LEN(132)
        /* RTVJOBA */
        DCL       VAR(&JOBNAME) TYPE(*CHAR) LEN(10)
        DCL       VAR(&JOBUSER) TYPE(*CHAR) LEN(10)
        DCL       VAR(&JOBNBR) TYPE(*CHAR) LEN(6)
        /* OUTPUT LIB/FILE */
        DCL       VAR(&TPLIB) TYPE(*CHAR) LEN(10)
        DCL       VAR(&SPFILE) TYPE(*CHAR) LEN(10)

@START:
        CHGVAR    VAR(&TPLIB) VALUE('QTEMP') /* QTEMP */
        CHGVAR    VAR(&SPFILE) VALUE('WRKSPLFL') /*SPLF TO FILE*/

        RTVJOBA   JOB(&JOBNAME) USER(&JOBUSER) NBR(&JOBNBR)
        OVRPRTF   FILE(QPRTSPLF) OUTQ(PRT01) /* SPLFNAME(&SPFILE) */
        MONMSG    MSGID(CPF0000) EXEC(GOTO @ERR)

        /*ユーザーセッション'Y'のとき、ジョブ情報で検索*/
        IF        COND(&CURSS *EQ 'Y') THEN(DO)
          WRKSPLF   SELECT(*ALL *ALL *ALL &USRDTA *ALL &SPLF) +
                      JOB(&JOBNBR/&JOBUSER/&JOBNAME) +
                      PERIOD((&STTIME &STDATE) (&EDTIME &EDDATE)) +
                      OUTPUT(*PRINT)
          MONMSG    MSGID(CPF0000) EXEC(GOTO @ERR)
        ENDDO
        ELSE      DO
          WRKSPLF   SELECT(&USER *ALL *ALL &USRDTA *ALL &SPLF) +
                      PERIOD((&STTIME &STDATE) (&EDTIME &EDDATE)) +
                      OUTPUT(*PRINT)
          MONMSG    MSGID(CPF0000) EXEC(GOTO @ERR)
        ENDDO

        DLTF      FILE(&TPLIB/&SPFILE)
        MONMSG    MSGID(CPF0000)

        CRTPF     FILE(&TPLIB/&SPFILE) SRCFILE(COMMONLIB/QDDSSRC) +
                    SRCMBR(SPFILED) IGCDTA(*NO) OPTION(*NOLIST +
                    *NOSOURCE) MAXMBRS(*NOMAX) SIZE(*NOMAX)
        MONMSG    MSGID(CPF0000) EXEC(GOTO @ERR)

        CPYSPLF   FILE(QPRTSPLF) TOFILE(&TPLIB/&SPFILE) +
                    JOB(&JOBNBR/&JOBUSER/&JOBNAME) SPLNBR(*LAST) +
                    CTLCHAR(*FCFC)
        MONMSG    MSGID(CPF0000) EXEC(GOTO @ERR)
        GOTO      CMDLBL(@END)
@ERR:
        RCVMSG    MSGTYPE(*LAST) MSGID(&MSGID) MSG(&MSG)
@END:     ENDPGM
```

図5　分割ファイルのDDS（SPFILED）

No	PK	内部名	英字名	表記	型	位置	長	桁	小数点	DEF	Null可
1		SPCTRL		印刷制御文字	A	1	1	0	0		N
2		SPBLANK			A	2	1	0	0		N
3		BLANK01			A	3	1	0	0		N
4		SPLNAM	Name	スプール名	A	4	10	0	0		N
5		BLANK02			A	14	1	0	0		N
6		USRNAM	UserName	ユーザ名	A	15	10	0	0		N
7		BLANK03			A	25	1	0	0		N
8		DEVNAM	DevName	装置名	A	26	10	0	0		N
9		BLANK04			A	36	1	0	0		N
10		USRDTA	UserData	ユーザーデータ	O	37	10	0	0		N
11		BLANK05			A	47	1	0	0		N
12		STATUS	Status	ステータス	A	48	3	0	0		N
13		BLANK06			A	51	2	0	0		N
14		TTLPAG	TotalPages	総ページ数	A	53	5	0	0		N
15		BLANK07			A	58	6	0	0		N
16		CURPAG	CurrentPage	現在ページ	A	64	5	0	0		N
17		BLANK08			A	69	1	0	0		N
18		FRMTYP	FormType	フォームタイプ	O	70	10	0	0		N
19		BLANK09			A	80	2	0	0		N
20		PRIOTY	Priority	優先順位	A	82	1	0	0		N
21		BLANK10			A	83	2	0	0		N
22		CRTDAT	CreateDate	作成日	A	85	8	0	0		N
23		BLANK11			A	93	1	0	0		N
24		CRTTIM	CreateTime	作成時刻	A	94	8	0	0		N
25		BLANK12			A	102	1	0	0		N
26		SPLNO	SpoolFileNumber	スプールNo	A	103	6	0	0		N
27		BLANK13			A	109	1	0	0		N
28		JOBNAM	JobName	ジョブ名	A	110	10	0	0		N
29		BLANK14			A	120	1	0	0		N
30		JOBNO	JobNumber	ジョブNo	A	121	6	0	0		N
31		BLANK15			A	127	1	0	0		N
32		OUTNAM	OutqName	OUTQ名	A	128	10	0	0		N
33		BLANK16			A	138	1	0	0		N
34		OUTLIB	OutqLibraryName	OUTQライブラリー	A	139	10	0	0		N
35		BLANK17			A	149	2	0	0		N
36		ASP	ASP	ASP	A	151	3	0	0		N
37		BLANK18			A	154	1	0	0		N
38		LSTDAT	LastUseDate	最終使用日	A	155	8	0	0		N
39		BLANK19			A	163	1	0	0		N
40		SPLSIZ	SpoolFileSize	スプールファイルサイズ	A	164	9	0	0		N
41		BLANK20			A	173	30	0	0		N

※CPYSPLF実行時に追加される
※CPYSPLF実行時に追加されるブランク

図6　QTEMPのWRKSPLF結果（WRKSPLFL）にアクセスするSQLベース

```
//スプールファイル取得用SQL文のベース
SQLSPBase:='SELECT ';
SQLSPBase:=SQLSPBase+' TRIM(SPLNAM) AS Name, ';
SQLSPBase:=SQLSPBase+' TRIM(USRNAM) AS UserName, ';
SQLSPBase:=SQLSPBase+' TRIM(DEVNAM) AS DevName, ';
SQLSPBase:=SQLSPBase+' TRIM(USRDTA) AS UserData, ';
SQLSPBase:=SQLSPBase+' TRIM(STATUS) AS Status, ';
SQLSPBase:=SQLSPBase+' TRIM(TTLPAG) AS TotalPages, ';
SQLSPBase:=SQLSPBase+' TRIM(CURPAG) AS CurrentPage, ';
SQLSPBase:=SQLSPBase+' TRIM(FRMTYP) AS FormType, ';
SQLSPBase:=SQLSPBase+' PRIOTY AS Priority, ';
SQLSPBase:=SQLSPBase+' TRIM(CRTDAT) AS CreateDate, ';
SQLSPBase:=SQLSPBase+' TRIM(CRTTIM) AS CreateTime, ';
SQLSPBase:=SQLSPBase+' TRIM(SPLNO) AS SpoolFileNumber, ';
SQLSPBase:=SQLSPBase+' TRIM(JOBNAM) AS JobName, ';
SQLSPBase:=SQLSPBase+' TRIM(JOBNO) AS JobNumber, ';
SQLSPBase:=SQLSPBase+' TRIM(OUTNAM) AS OutqName, ';
SQLSPBase:=SQLSPBase+' TRIM(OUTLIB) AS OutqLibraryName, ';
SQLSPBase:=SQLSPBase+' TRIM(ASP) AS ASP, ';
SQLSPBase:=SQLSPBase+' TRIM(LSTDAT) AS LastUseDate, ';
SQLSPBase:=SQLSPBase+' TRIM(SPLSIZ) AS SpoolFileSize ';
SQLSPBase:=SQLSPBase+' FROM QTEMP.WRKSPLFL ';
SQLSPBase:=SQLSPBase+' WHERE TRIM(STATUS)
    IN (''RDY'',''OPN'',''DFR'',''SND'',''CLO'',''HLD'',''SAV'',''WTR'',''FIN'',''PND'',''PRT'',''MSGW'') ';
   //ヘッダー等も含まれてくるため
SQLSPBase:=SQLSPBase+' AND TRIM(SPLNAM) <> ''QPRTSPLF'' '; //WRKSPLF印刷原本は除く
```

WHERE条件でSTATUSでの値絞込と、WRKSPLF印刷の原本を除外する。

図7　動的にWHERE文の追加

```
{-------------------------------------------------------------------------------
* 表題:Where文以下の条件式を返す
* 引数1[i]:PrmName:条件式で使用されるパラメータ名
* 引数2[i/o]:FstCon:条件式の先頭かどうか ※参照渡し
* 戻値:条件式
*-------------------------------------------------------------------------------}
function TfrmSpoolMaint.fGetWhereCon(PrmName: string; var FstCon: Boolean): string;
var
  strSQL: string;
begin

    strSQL:=' AND ';

  if PrmName = 'pOUTQ' then begin
//    strSQL:=strSQL+' UPPER(TRIM(SUBSTRING(SPTEXT,126,10))) = :pOUTQ '; //フィールド分割前
    strSQL:=strSQL+' UPPER(TRIM(OUTNAM))) = :pOUTQ '; //ベンダーCD
  end;
```

図8　WRKSPLF結果の取得

```
{-------------------------------------------------------------------------------
* 表題:検索クエリの実行
*-------------------------------------------------------------------------------}
procedure TfrmSpoolMaint.fOpenQuery;
var
  strSQL: string;
  fstCondition: boolean; //条件式の先頭かどうか
  intRecCnt: Integer;
  strCmd: string;
  strUserName, strSpoolName, strUserData: string;
  strStartDate,strStartTime: string;
  strEndDate,strEndTime: string;
  strCurSession: string;
  SaveCursor: TCursor; //現在のマウスカーソル
begin
  { SQL文の作成 }

  if (edtUserName.Text='') and (edtUserData.Text='') and (edtSpoolName.Text='') and (edtOUTQ.Text='') then begin
    if cfMessageDlgEx('ユーザー名、ユーザーデータ、スプールファイル名、
    OUTQがブランクの場合時間がかかる場合がありますが、実行しますか？',
    mtConfirmation, [mbYes,mbNo], 2) <> mrYes then Abort;
  end;

  //WRKSPLF→QTEMP/WRKSPLFTOFへ。画面入力がないとき*ALLで取得
  if edtUserName.Text='' then strUserName:='*ALL' else strUserName:=edtUserName.Text; //ユーザー名
  if edtUserData.Text='' then strUserData:='*ALL' else strUserData:=edtUserData.Text; //ユーザーデータ
  if edtSpoolName.Text='' then strSpoolName:='*ALL' else strSpoolName:=edtSpoolName.Text; //スプールファイル名
  if chkCurSession.Checked then strCurSession:='Y' else strCurSession:=''; //現在のセッション

  DateTimeToString(strStartDate, 'yyyyMMdd',dtpStartDate.DateTime);
  DateTimeToString(strStartTime, 'HHmmss',dtpStartTime.DateTime);
  DateTimeToString(strEndDate, 'yyyyMMdd',dtpEndDate.DateTime);
  DateTimeToString(strEndTime, 'HHmmss',dtpEndTime.DateTime);
  strCmd:='CALL PGM(COMMONLIB/WRKSPLFTOF) ';
  strCmd:=strCmd+' PARM('''+strUserName+''' '''+strUserData+''' '''+strSpoolName+''' ';
  strCmd:=strCmd+' '''+strStartDate+''' '''+strStartTime+''' '''+strEndDate+''' '''+strEndTime+''' '''+strCurSession+''' ''''''')';

  fdmCommon.as4Main.RemoteCmd(strCmd); //WRKSPLF実行

  //ここから結果FILEをSQL
  strSQL:=SQLSPBase;
  fstCondition:=True;
  if edtOUTQ.Text <> '' then  strSQL:=strSQL+fGetWhereCon('pOUTQ',fstCondition); //OUTQ条件の追加
  //--------------------------------

  SaveCursor:=Screen.Cursor;
  Self.Enabled:=False;
  try//マウスカーソル操作
    Screen.Cursor:=crHourGlass; //砂時計
    { 件数の事前確認 }
    qryCount.Close;
    qryCount.SQL.Clear;
    qryCount.SQL.Add('SELECT COUNT(*) FROM ( '+strSQL+' ) src');
    fSetSQLParam(qryCount); //パラメータ設定
    qryCount.Open;
    Application.ProcessMessages; //応答なしメッセージ回避
    intRecCnt:=qryCount.Fields[0].AsInteger;
    qryCount.Close; //セッションの解除
    if intRecCnt >= 200 then begin
      if cfMessageDlgEx('検索結果が200件以上('+IntToStr(intRecCnt)+'件)ありますが、実行しますか？',
      mtConfirmation, [mbYes,mbNo], 2) <> mrYes then Abort;
    end;
    //--------------------------------

    { 検索処理 }
    cdsSpoolList.Close;
    cdsSpoolList.IndexName:=''; //並び替えクリア
    qrySpoolList.SQL.Clear;
    qrySpoolList.SQL.Add(strSQL);
    fSetSQLParam(qrySpoolList); //パラメータ設定
    cdsSpoolList.Open;
    Application.ProcessMessages; //応答なしメッセージ回避
    //--------------------------------

    if (cdsSpoolList.RecordCount>0) then begin //対象リストがあるとき
      edtStrings.Enabled:=True;
      btnStringsFilter.Enabled:=True;
      btnStringsFilterOff.Enabled:=True;
    end
    else begin
      edtStrings.Enabled:=False;
      btnStringsFilter.Enabled:=False;
      btnStringsFilterOff.Enabled:=False;
    end;

    lblRecordCount.Caption:=cfGetRecCnt(cdsSpoolList,''); //レコード数の表示
  finally
    Screen.Cursor:=SaveCursor; //保存していたカーソルに戻す
    Self.Enabled:=True;
  end;

end; ┗
```

図9　スプール管理コマンド

```
{-------------------------------------------------------------------------------
* 表題:Spool解放ボタン(RLSSPLF)
*-------------------------------------------------------------------------------}
procedure TfrmSpoolMaint.btnRelease_OnClick(Sender: TObject);
var
  i: Integer;
begin
  inherited;

  if cdsSpoolList.Active=False then Exit;
  if cdsSpoolList.RecordCount=0 then Exit;

  try
    //選択状態になっているか
    if dbgMain.SelectedRows.Count > 0 then begin
      for i:=0 to dbgMain.SelectedRows.Count - 1 do begin
        //解放実行
        cdsSpoolList.Bookmark:=dbgMain.SelectedRows[i];
        fReleaseSplF;//RLSSPLF実行
        if not cdsSpoolList.Modified then cdsSpoolList.Edit;
        cdsSpoolList.FieldByName('STATUS').AsString:='*RELEASE';
      end;
    end else begin
    end;
  except
    on e: Exception do begin //Exceptionクラスは全例外クラスの基本クラスであり、全例外をトラップ出来る
      ShowMessage('Exception:'+e.ClassName+'/'+e.Message);
    end;
  end;

end;

{-------------------------------------------------------------------------------
* 表題:Spool削除ボタン(DLTSPLF)
*-------------------------------------------------------------------------------}
procedure TfrmSpoolMaint.bbtnDelete_OnClick(Sender: TObject);
var
  i: Integer;
begin
  inherited;

  try
    //選択状態になっているか
    if dbgMain.SelectedRows.Count > 0 then begin
      for i:=0 to dbgMain.SelectedRows.Count - 1 do begin
        //削除確認
        if cfMessageDlgEx('削除しますか？', mtConfirmation, [mbYes,mbNo], 2) <> mrYes then Abort;
        //削除実行
        cdsSpoolList.Bookmark:=dbgMain.SelectedRows[i];
        fDeleteSplF;//DLTSPLF実行
        if not cdsSpoolList.Modified then cdsSpoolList.Edit;
        cdsSpoolList.FieldByName('STATUS').AsString:='*DELETE';
      end;
    end else begin
    end;
  except
    on e: Exception do begin //Exceptionクラスは全例外クラスの基本クラスであり、全例外をトラップ出来る
      ShowMessage('Exception:'+e.ClassName+'/'+e.Message);
    end;
  end;

end;
```

```
{-----------------------------------------------------------------------------
* 表題:CHGSPLFA実行
*-----------------------------------------------------------------------------}
procedure TfrmSpoolMaint.btnOUTQChange_OnClick(Sender: TObject);
var
  i: Integer;
  strLIB: String;
  strOUTQ: String;
begin
  inherited;

  if cdsSpoolList.Active=False then Exit;
  if cdsSpoolList.RecordCount=0 then Exit;
  if edtChgOUTQ.Text='' then begin
    ShowMessage('変更先OUTQを指定して下さい。');
    Abort;
  end;

  strLIB:='*LIBL';
  strOUTQ:=edtChgOUTQ.Text;

  try
    //選択状態になっているか
    if dbgMain.SelectedRows.Count > 0 then begin
      for i:=0 to dbgMain.SelectedRows.Count - 1 do begin
        //解放実行
        cdsSpoolList.Bookmark:=dbgMain.SelectedRows[i];
        fChangeSplF(StrLIB,StrOUTQ);//CHGSPLFA実行
        if not cdsSpoolList.Modified then cdsSpoolList.Edit;
        cdsSpoolList.FieldByName('STATUS').AsString:='*CHANGE';
        cdsSpoolList.FieldByName('OUTQLIBRARYNAME').AsString:=strLIB;
        cdsSpoolList.FieldByName('OUTQNAME').AsString:=strOUTQ;
      end;
    end else begin
    end;
  except
    on e: Exception do begin //Exceptionクラスは全例外クラスの基本クラスであり、全例外をトラップ出来る
      ShowMessage('Exception:'+e.ClassName+'/'+e.Message);
    end;
  end;
end;
{-----------------------------------------------------------------------------
* 表題:HLDSPLF実行
*-----------------------------------------------------------------------------}
procedure TfrmSpoolMaint.fHoldSplF;
var
  strCmd: String;
  strSplF,strJobName,strJobUser,strJobNumber,strSplNumber: String;
begin
  with cdsSpoolList do begin
    strSplF:=Trim(FieldByName('NAME').AsString);
    strJobName:=Trim(FieldByName('JOBNAME').AsString);
    strJobNumber:=Trim(FieldByName('JOBNUMBER').AsString);
    strJobUser:=Trim(FieldByName('USERNAME').AsString);
    strSplNumber:=Trim(FieldByName('SPOOLFILENUMBER').AsString);
  end;

  strCmd:='HLDSPLF FILE('+strSplF+') '+
          'JOB('+strJobNumber+'/'+strJobUser+'/'+strJobName+') '+
          'SPLNBR('+strSplNumber+')';
  fdmCommon.as4Main.RemoteCmd(strCmd);
end;
```

図10　スプールデータのQTEMP取得

```
            PGM         PARM(&SPLNAME &JOBNBR &JOBUSER &JOBNAME &SPLNMBR +
                          &MSGID &MSG)

/*************************************************************************/
/*表題:CPYSPLFの結果をQTEMPにファイル保存する                            */
/*************************************************************************/

            /*パラメータ*/
            DCL         VAR(&SPLNAME) TYPE(*CHAR) LEN(10)
            DCL         VAR(&JOBNAME) TYPE(*CHAR) LEN(10)
            DCL         VAR(&JOBUSER) TYPE(*CHAR) LEN(10)
            DCL         VAR(&JOBNBR) TYPE(*CHAR) LEN(6)
            DCL         VAR(&SPLNMBR) TYPE(*CHAR) LEN(10)
            DCL         VAR(&MSGID) TYPE(*CHAR) LEN(7)
            DCL         VAR(&MSG) TYPE(*CHAR) LEN(132)
            /* RTVJOBA */
            /* OUTPUT LIB/FILE */
            DCL         VAR(&TPLIB) TYPE(*CHAR) LEN(10)
            DCL         VAR(&SPFILE) TYPE(*CHAR) LEN(10)

 @START:
            CHGVAR      VAR(&TPLIB) VALUE('QTEMP') /* QTEMP */
            CHGVAR      VAR(&SPFILE) VALUE('CPYSPLFL') /*SPLF TO FILE*/

            DLTF        FILE(&TPLIB/&SPFILE)
            MONMSG      MSGID(CPF0000)

            CRTPF       FILE(&TPLIB/&SPFILE) SRCFILE(COMMONLIB/QDDSSRC) +
                          SRCMBR(SPFILEC) IGCDTA(*NO) OPTION(*NOLIST +
                          *NOSOURCE) MAXMBRS(*NOMAX) SIZE(*NOMAX)
            MONMSG      MSGID(CPF0000) EXEC(GOTO @ERR)

            CPYSPLF     FILE(&SPLNAME) TOFILE(&TPLIB/&SPFILE) +
                          JOB(&JOBNBR/&JOBUSER/&JOBNAME) SPLNBR(&SPLNMBR) +
                          CTLCHAR(*FCFC)
            MONMSG      MSGID(CPF0000) EXEC(GOTO @ERR)

            /*ページ番号採番*/
            CALL        PGM(COMMONLIB/SPPAGEUP)
            MONMSG      MSGID(CPF0000) EXEC(GOTO @ERR)
            GOTO        CMDLBL(@END)
 @ERR:
            RCVMSG      MSGTYPE(*LAST) MSGID(&MSGID) MSG(&MSG)
 @END:       ENDPGM
```

図11　DB化スプールファイルのDDS（SPFILEC）

No	PK	内部名	英字名	表記	型	位置	長	桁	小数点	DEF	Null可	
1		SPCTRL		印刷制御文字	A	1	1	0	0		N	
2		SPBLANK			A	2	1	0	0		N	
3		SPTEXT		TEXT	O	3	200	0	0		N	※スプール文字列
4		SPPAGE		PAGE	A	203	9	0	0		N	※ページ番号

※DDSは上記の名前となっているが、初めからSPYSPLFLでも良い。

図12　取得一覧から印字文字列で絞込み

```
{-------------------------------------------------------------------------------
* 表題:リスト絞込ボタン押下
*-------------------------------------------------------------------------------}
procedure TfrmSpoolMaint.btnStringsFilter_OnClick(Sender: TObject);
var
  strCmd: string;
  intRecCnt: Integer;
  SaveCursor: TCursor; //現在のマウスカーソル
begin
  inherited;

  if cdsSpoolList.Active=False then Exit;
  if cdsSpoolList.RecordCount=0 then Exit;

  cdsSpoolList.First;

  prbProgressBar.Visible:=True;
  prbProgressBar.Position:=0;
  prbProgressBar.Max:=cdsSpoolList.RecordCount;

  SaveCursor:=Screen.Cursor;
  Self.Enabled:=False;
  try//マウスカーソル操作
    Screen.Cursor:=crHourGlass; //砂時計
    try

      while not cdsSpoolList.Eof do begin
        strCmd:='CALL PGM(COMMONLIB/CPYSPLFTOF) ';
        strCmd:=strCmd+' PARM(';
        strCmd:=strCmd+' ""+cdsSpoolList.FieldByName('NAME').AsString+""; //スプール名
        strCmd:=strCmd+' ""+cdsSpoolList.FieldByName('JOBNUMBER').AsString+""; //スプール名
        strCmd:=strCmd+' ""+cdsSpoolList.FieldByName('USERNAME').AsString+""; //ユーザー名
        strCmd:=strCmd+' ""+cdsSpoolList.FieldByName('JOBNAME').AsString+""; //ジョブ名
        strCmd:=strCmd+' ""+cdsSpoolList.FieldByName('SPOOLFILENUMBER').AsString+""; //スプール番号
        strCmd:=strCmd+' "" "" )';
        fdmCommon.as4Main.RemoteCmd(strCmd); //WRKSPLF実行

        { 件数の事前確認 }
        qryCount.Close;
        qryCount.SQL.Clear;
        qryCount.SQL.Add('SELECT COUNT(*) FROM QTEMP.CPYSPLFL WHERE UPPER(SPTEXT)
          LIKE '+cfQStr('%'+UpperCase(edtStrings.Text)+'%'));
        fSetSQLParam(qryCount); //パラメータ設定
        qryCount.Open;
        Application.ProcessMessages; //応答なしメッセージ回避
        intRecCnt:=qryCount.Fields[0].AsInteger;
        qryCount.Close; //セッションの解除

        if intRecCnt = 0 then cdsSpoolList.Delete
        else cdsSpoolList.Next;

        if prbProgressBar.Position<prbProgressBar.Max then prbProgressBar.StepIt//進捗進行
        else cdsSpoolList.Last;//終了

        prbProgressBar.Repaint;
        dbgMain.Repaint;

      end;

      lblRecordCount.Caption:=cfGetRecCnt(cdsSpoolList,''); //レコード数の表示
    except
      on e: Exception do begin //Exceptionクラスは全例外クラスの基本クラスであり、全例外をトラップ出来る
        ShowMessage('Exception:'+e.ClassName+'/'+e.Message);
      end;
    end;
  finally
    Screen.Cursor:=SaveCursor; //保存していたカーソルに戻す
    Self.Enabled:=True;
    prbProgressBar.Visible:=False;
  end;

end;
```

図13　スプール表示画面

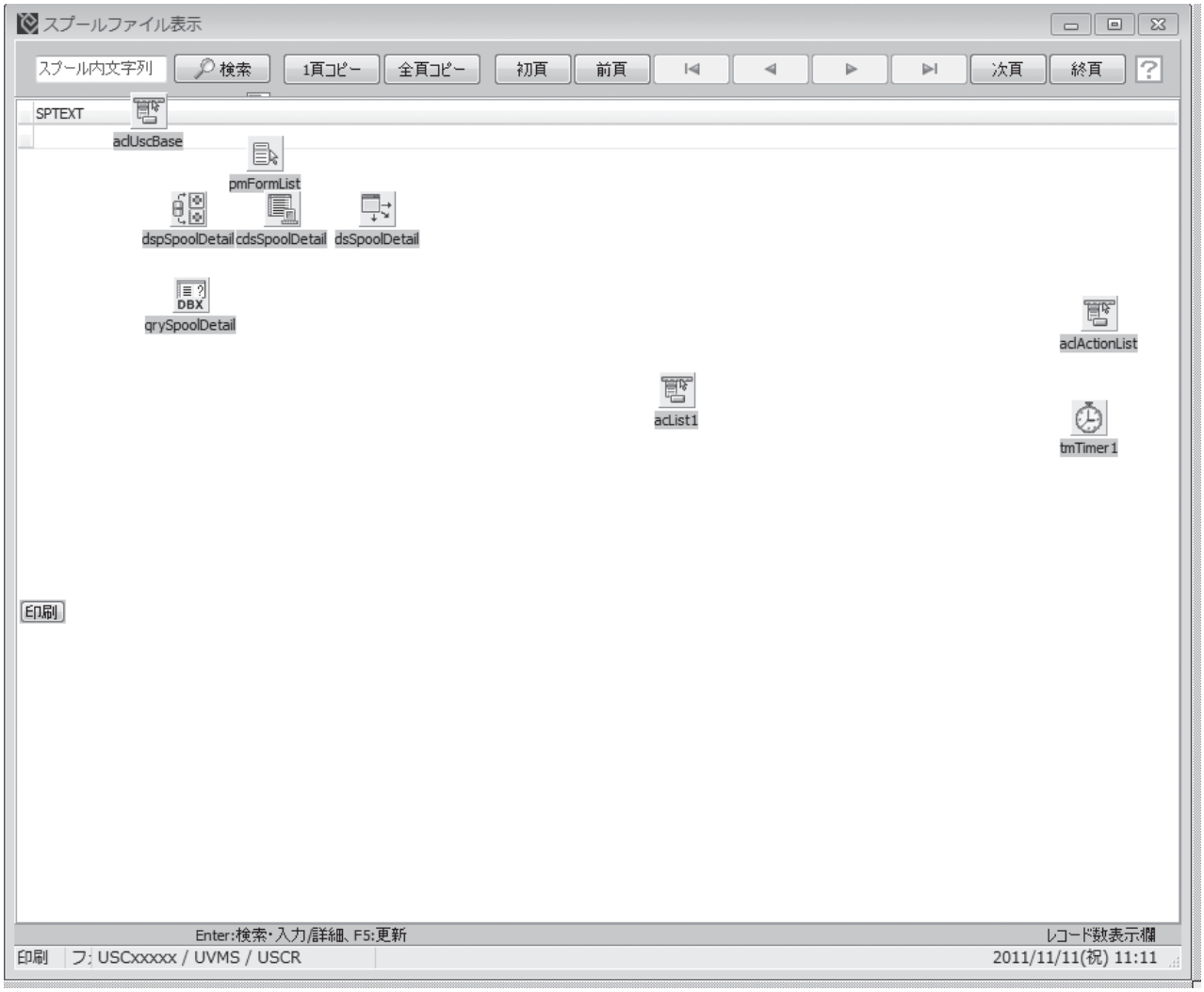


図14　スプールデータのページカウントRPG(LE)

```
FCPYSPLFL  UP   E           DISK
D* SQL RETURN CD
D CNT             S             5S 0
 /FREE
     IF SPCTRL='1';
       CNT = CNT + 1;
     ENDIF;
     SPPAGE=CNT;
     UPDATE    SPRECD;
 /END-FREE
```

SPCTRLに制御文字'1'があればページカウントして更新

図15　スプール表示のページ移動

```
{-----------------------------------------------------------------------------
* 表題:初頁ボタン実行
*-----------------------------------------------------------------------------}
procedure TfrmSpoolDisp.btnFirst_OnClick(Sender: TObject);
var
  intCurPage: Integer;
begin
  inherited;

  intCurPage:=1;

  //開始頁
  cdsSpoolDetail.Filter:='SPPAGE='+IntToStr(intCurPage);
  fPageChange;
  lblRecordCount.Caption:=fGetPageCnt(intCurPage,intMaxPage);

  fSelectInfo_Clear;//選択文字列情報のクリア

end;

{-----------------------------------------------------------------------------
* 表題:終頁ボタン実行
*-----------------------------------------------------------------------------}
procedure TfrmSpoolDisp.btnLast_OnClick(Sender: TObject);
begin
  inherited;

  //最終頁
  cdsSpoolDetail.Filter:='SPPAGE='+IntToStr(intMaxPage);
  fPageChange;
  lblRecordCount.Caption:=fGetPageCnt(intMaxPage,intMaxPage);

  fSelectInfo_Clear;//選択文字列情報のクリア
end;

{-----------------------------------------------------------------------------
* 表題:次頁ボタン実行
*-----------------------------------------------------------------------------}
procedure TfrmSpoolDisp.btnNext_OnClick(Sender: TObject);
var
  intCurPage: Integer;
begin
  inherited;

  intCurPage:=cdsSpoolDetail.FieldByName('SPPAGE').AsInteger+1;

  //現ページより+1
  cdsSpoolDetail.Filter:='SPPAGE='+IntToStr(intCurPage);
  fPageChange;
  lblRecordCount.Caption:=fGetPageCnt(intCurPage,intMaxPage);

  fSelectInfo_Clear;//選択文字列情報のクリア

end;

{-----------------------------------------------------------------------------
* 表題:前頁ボタン実行
*-----------------------------------------------------------------------------}
procedure TfrmSpoolDisp.btnPrior_OnClick(Sender: TObject);
var
  intCurPage: Integer;
begin
  inherited;

  intCurPage:=cdsSpoolDetail.FieldByName('SPPAGE').AsInteger-1;

  //現ページより-1
  cdsSpoolDetail.Filter:='SPPAGE='+IntToStr(intCurPage);
  fPageChange;
  lblRecordCount.Caption:=fGetPageCnt(intCurPage,intMaxPage);

  fSelectInfo_Clear;//選択文字列情報のクリア

end;

{-----------------------------------------------------------------------------
* 表題:頁変更時のボタン利用変更
*-----------------------------------------------------------------------------}
procedure TfrmSpoolDisp.fPageChange;
begin

  if cdsSpoolDetail.FieldByName('SPPAGE').AsInteger=1 then begin
    btnPrior.Enabled:=False;
    btnFirst.Enabled:=False;
  end else begin
    btnPrior.Enabled:=True;
    btnFirst.Enabled:=True;
  end;

  if cdsSpoolDetail.FieldByName('SPPAGE').AsInteger=intMaxPage then begin
    btnNext.Enabled:=False;
    btnLast.Enabled:=False;
  end else begin
    btnNext.Enabled:=True;
    btnLast.Enabled:=True;
  end;

end;
```

図16　ブラウザで文字列検索するような機能を搭載する

```
{-------------------------------------------------------------------------------
* 表題:文字列検索実行
*-------------------------------------------------------------------------------}
procedure TfrmSpoolDisp.bbtnSearch_OnClick(Sender: TObject);
var
  i: Integer;
  SaveCursor: TCursor; //現在のマウスカーソル
begin
  inherited;

  if edtStrings.Text='' then Exit;

  if dbgSpoolDetail.SelectedIndex=0 then dbgSpoolDetail.SelectedIndex:=1 ;//選択状態
  if intSelectedRow<>dbgSpoolDetail.SelectedIndex then intOffSet:=1 ;//

  SaveCursor:=Screen.Cursor;
  Self.Enabled:=False;
  i:=0;
  try//マウスカーソル操作
    Screen.Cursor:=crHourGlass; //砂時計

    while (not cdsSpoolDetail.Eof) do begin

        i := PosEx(UpperCase(edtStrings.Text),UpperCase(dbgSpoolDetail.SelectedField.Text),intOffSet);
      if i>0 then begin

        Break;
      end
      else begin
        intOffSet:=1 ;//初期化
          cdsSpoolDetail.Next;
      end;

      if cdsSpoolDetail.Eof then
        if cdsSpoolDetail.FieldByName('SPPAGE').AsInteger<intMaxPage then btnNext_OnClick(nil)
        else ShowMessage('見つかりませんでした。');

    end;
  finally
    Screen.Cursor:=SaveCursor; //保存していたカーソルに戻す
    Self.Enabled:=True; //オフにすると選択状態が表示されない
  end;

  if i>0 then begin
    //入力状態にする。
    dbgSpoolDetail.SetFocus;
    dbgSpoolDetail.EditorMode := True;

    TDummyDBG(dbgSpoolDetail).InplaceEditor.SelStart := i - 1;
    TDummyDBG(dbgSpoolDetail).InplaceEditor.SelLength := Length(edtStrings.Text);
    intOffSet:=TDummyDBG(dbgSpoolDetail).InplaceEditor.SelStart+TDummyDBG(dbgSpoolDetail).InplaceEditor.SelLength+1;
    intSelectedRow:=dbgSpoolDetail.SelectedIndex;
  end

end;
```

検索結果の文字列の場所を覚えておき、次に検索が実行されたとき、次に見つかる文字列からスタートする

図17　完成図（WRKSPLFを利用した場合の開発画面）

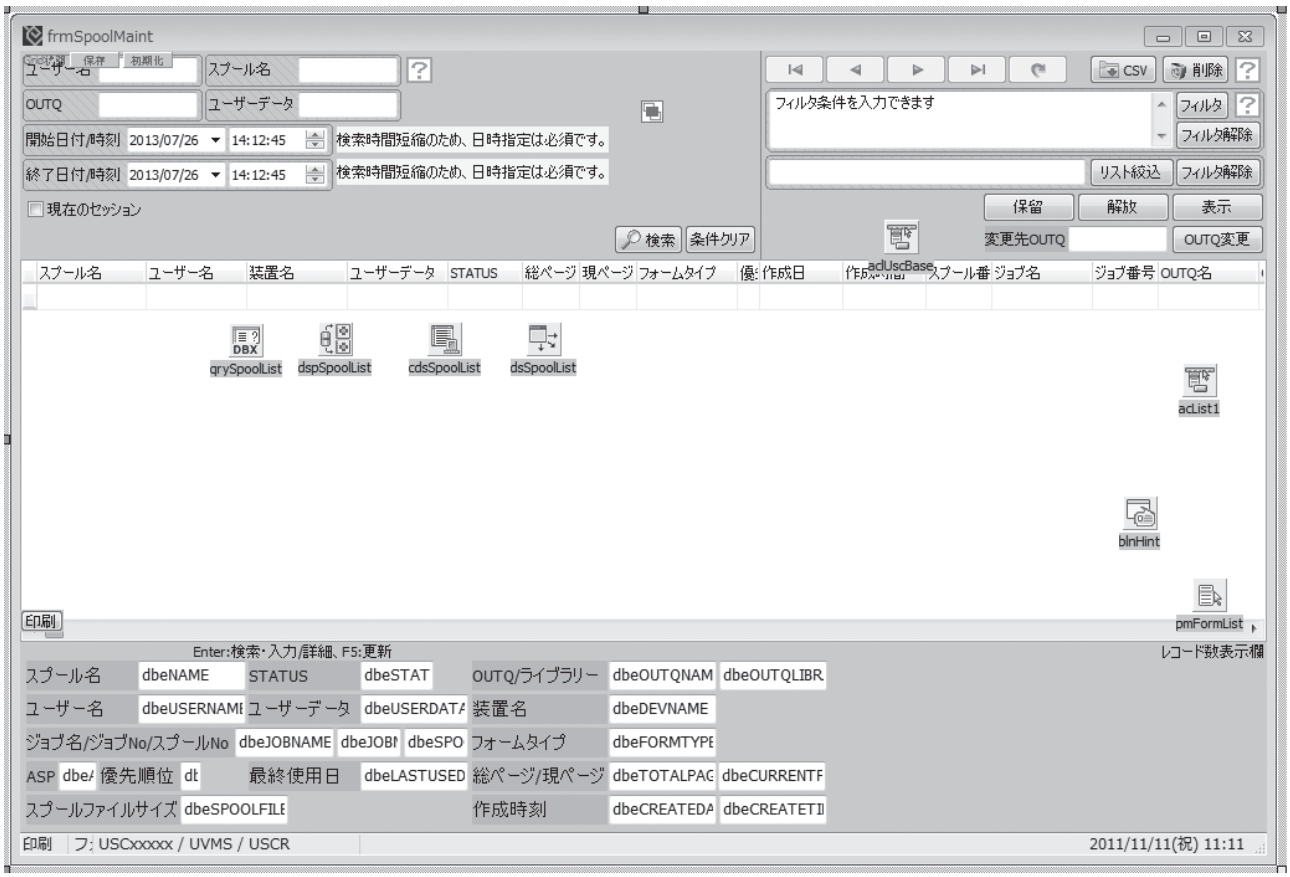
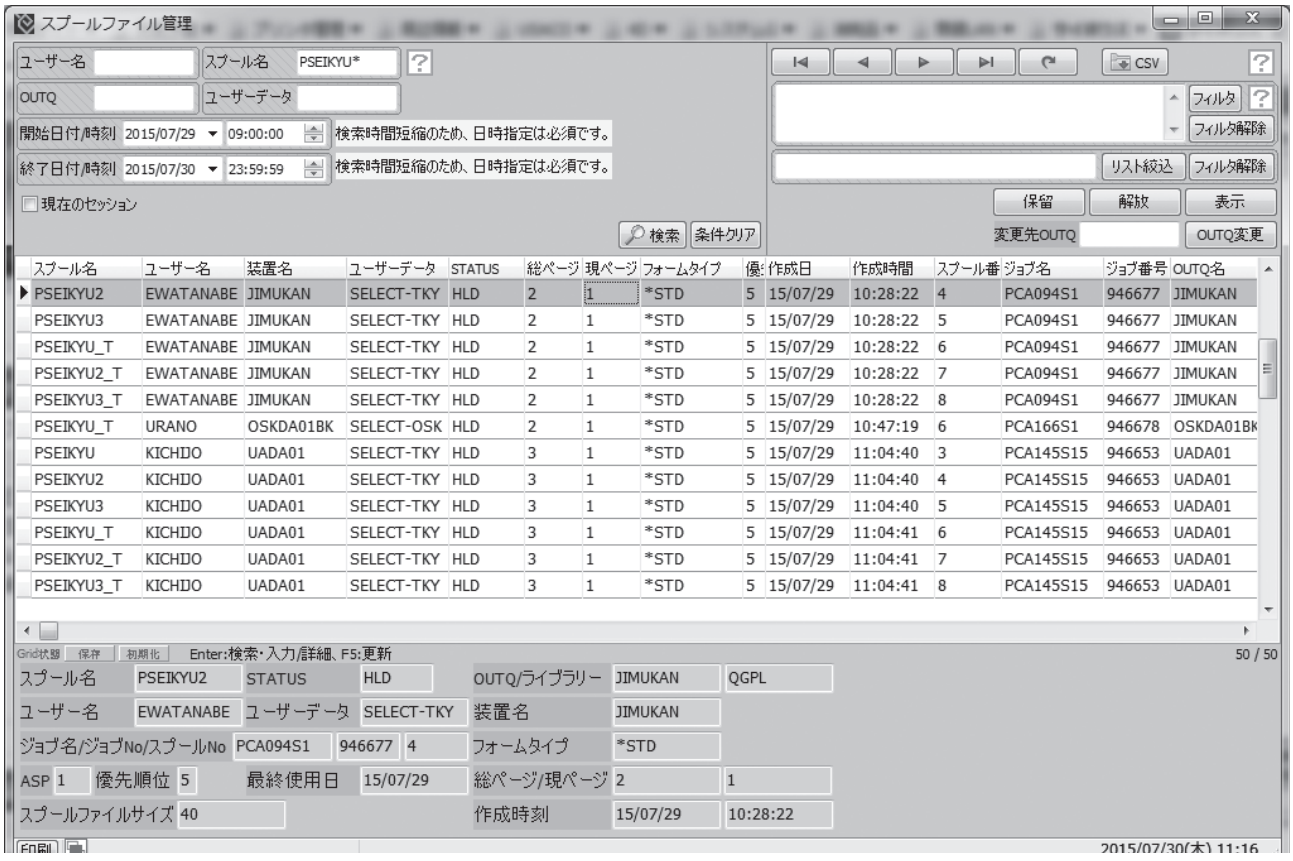


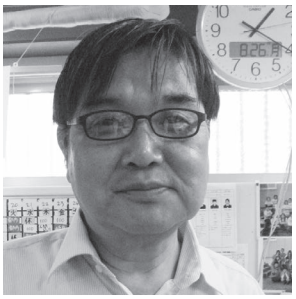
図18　完成図（実行画面）



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
管理部 情報システム課
開発担当
課長



株式会社タツミヤ
http：//www.tatsumiya.jp/

婦人服専門店として、北は北海道から南は沖縄まで、全国に約 400 店舗を展開。独自の情報システムの構築により、全店舗へのすばやいトレンド商品の供給を実現し、豊かなファッション文化を提案している。

1.予算管理について

当社では、従来から使用している「発注システム」に対して定期的なプログラムメンテナンスと関連システムへの拡張を行っている。発注システムは仕入のシステムが中心であり、店舗の在庫管理と深く関わっている。【図 1】

一方、発注システムの運用にあたっては、商品の仕入れを予算（販売計画）以上に行うと店舗在庫が膨れ上がる可能性があるので、月中はもとより年度初めに立案する予算が非常に重要である。しかし、過去のデータを参照して行う月次および年度の予算立案は、業務に精通したベテラン社員でないと行えない難しい業務である。

そこで、入社 2、3 年の若手社員でも、業務経験に応じて容易に予算を立案できるシステムが必要との議論になり、今回、商品部の予算担当者と話し合いながら予算管理システムを構築した。

過去の販売データは従来、Excel でまとめていた。しかし、Excel の計算機能を使うよりも、基幹システムから直接データを取得したいとの要望が予算担当者から出されたため、Delphi/400 を使い、「年初予算データ」「年初商品勘定予算登録」「確定予算登録」「修正仕入予算登録変更」などの画面を持つ予算管理システムを開発した。

年度予算の立案作業は、実務上、年度初めの約 2 カ月前から始まる。そのため最後の 2 カ月余りは実績が未確定なので推定値を使用しなければならず、非常に難しい。しかし将来の予測が可能であれば（本システムでは 2 年後まで）、予測値と実績を合わせた検討により、迅速な予算立案が行える。

開発した予算管理システムは、まだ修正すべき点や不足な点があるが、本稿では、完成したところまでを報告する。

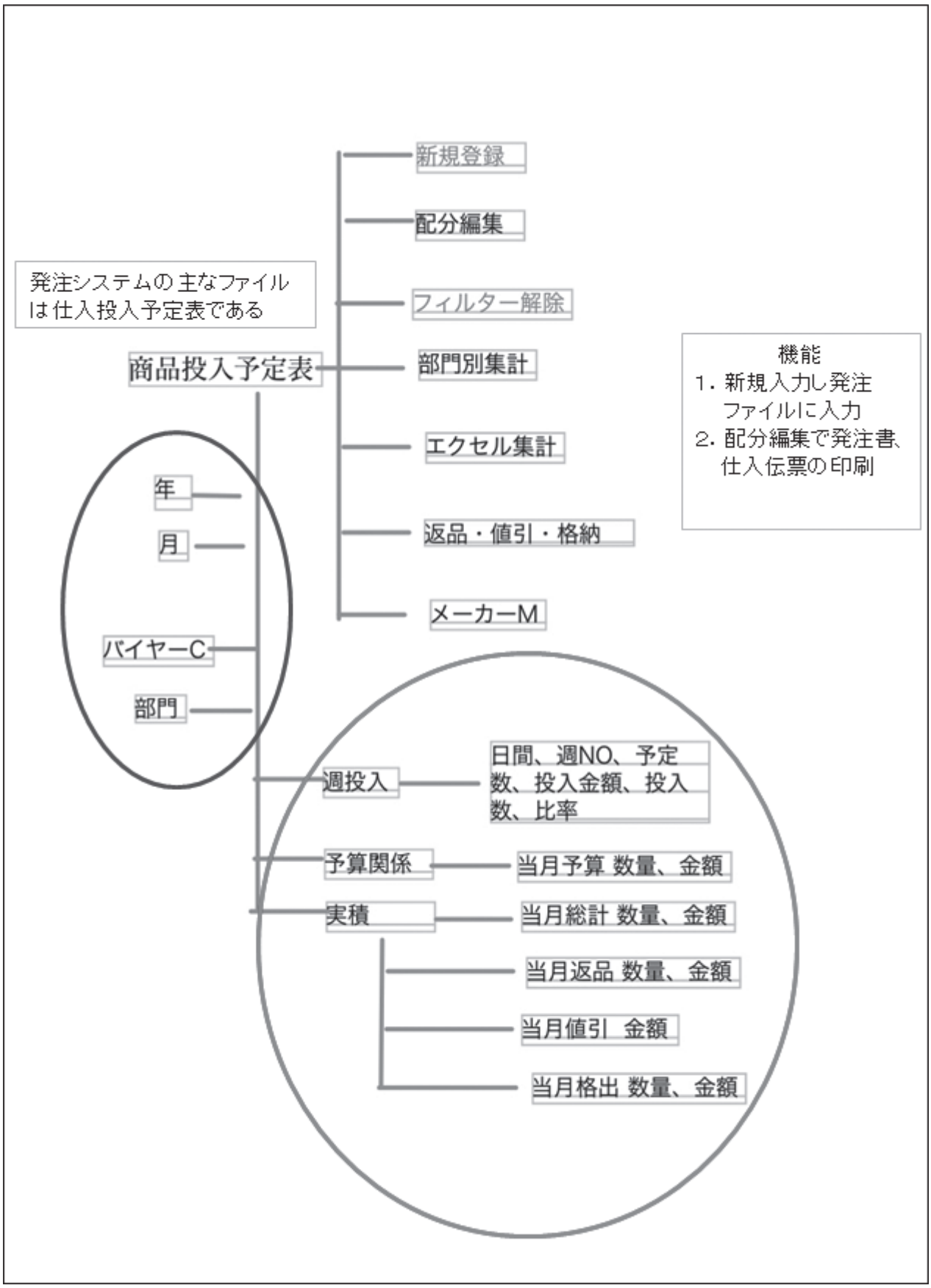
2.年初予算決定フローについて

年初予算は、システムでは、年度計の売上高（計画）を入力した後、売上指数（年全体に対する各月の比率）と会社別売上比率（タツミヤグループの株式会社タツミヤと株式会社エスティーサービスの比率）を決定することにより、「粗利高予算」、「売上原価」、「在庫回転率」などが自動的に算出されるロジックとしている。

入力画面ではシミュレーションも可能とし、各入力項目を登録後、算出結果をファイルに更新して再現できるようにした。このようなことは、Excel のワークシート上でも可能だが、Delphi/400 を使ってデータベースから年度データを与え、計算結果を表示できるシステムにすれば、関連する適用業務にただちに反映できる。入力画面の裏では、年初予算決定フローのロジックが走っている仕組みである。【図 2】

予算管理関連の総合メニューと、年初予算登録画面を【図 3】で示した。予算管理の対象項目は、「年初商品勘定予算登録」画面で表示する通りである（在庫回転率、売上高予算、売上予定比率、売上原価、等）。

図1　発注システムの概要



月別の年間予算は、年度計予算と期首在庫が決まると計算で求められる（【図2】の決定フロー）。従って、年度計予算を予測できるシステムがあれば、予算作成に有意義である。次に年度計について述べる。

3.年度計の算出について

予測手法の導入

今年度や次年度の売上を予測する拠りどころとなるのは、過去数年間の売上実績データである。一般に、過去の時系列データから未来のデータを予想するには、求めたい年度のデータ（売上等）が、年度（2015年等）に対して、どのような関数になっているかを決定する必要がある。

たとえば、最も単純な例として、2010年 = 100、2011年 = 110、2012年 = 120、2013年 = 130、2014年 = 140、という時系列データがあれば、2015年の予想値は誰しも「150」と考えるだろう。これは、1次式の関数を当てはめる単純な例だが、実際は、指数的に売上が伸びる時系列データもあれば、各年度の売上実績に誤差が入っていることもあるので、事はそう単純ではない。

では一般的に、1次式ではないn次式の関数を想定し未来を予測するにはどうするか。

未来データの予測には、実際に過去のデータに当てはめて最も矛盾のなかった関数を求め、それを未来に適用すればよい。

ここで、過去の年度ごとに「理論値」（想定した関数に基づく）と「実績値」の差分の2乗をとり、求めた各年度の値の和（対象の過去年度分の合計）が最も小さくなるようにする（最小2乗法）、という統計的な手法を採用する。このようにして導いたn次式の関数は、「直交多項式」と見なすことで式が単純な形になり、現実に解ける計算式になる。

話が複雑になったが、過去の時系列売上データから未来の売上データを予測するための統計的手法、とご理解いただきたい。

予測手法の詳細

分散分析の統計的手法を用いて、実際にDelphiでロジックを開発していった。その作業では、『中小企業のための未来予測計算システム』（新藤瞳著、日刊工業新聞社）を参考にした。【計算式1】の数式は参考文献からの引用である。細かい理論になるので、ご興味があれば参照いただきたい。

計算にあたって、分散分析では2乗の計算が主であり、桁落ちの対策として3桁を使用し、単位を千万円にした。【ソース1】で、分散分析の計算と結果表示プロシージャーのコードを記述したので参照されたい。

実際の算出結果について

結果を【図4】に示す。予算管理メニューから「年初予算データ」→「年初データ作成」→「過去データ参照」で、【図4】が表示される。

今まで理論的なことを述べてきたが、この予測値は実務上では参考値となる。妥当な数値であるかは次の段階で決定される。会社の営業目標としての年度値は、これに何パーセントかを上乗せするのが基本である。このようにして決定した年度計の売上予算値が、前章（2. 年初予算フローについて）で説明した「年初商品勘定登録」画面によって月別に展開される。

4.予算と実績の比較

年度初めに予算を決定するが、その予算が実績に対してどうだったかを見る画面が必要である。

【図5】は、全社の予算と実績を確認するための画面である（メニューの「年初予算データ」より実行）。これによりマクロ的に実績評価ができる。

部門別などの予算と実績は、日報として常に表示している【図6】。また、当社では、各部門別に、売上、仕入、在庫の実績を評価する別システムも走らせており、今回の予算管理システムから評価の基となるデータを渡している。

仕入予算管理の売上実績は、当月の前日までの売上である。このままでは予算と比較できないため、その月の月末になった時の推定金額（売上予測）を算出する。売上予測と予算を比較して予算比を算出している。仕入予算管理表は、予算メニューの年初予算データから表示される。

売上予測金額を算出するためには日割り予算が必要である。原則前年実績値を曜日対応させて日割を求めている【図7】。予算管理メニューの「曜日対応登録変更」画面で入力する。仕入予算管理表は「曜日対応登録変更」で作成された日割計算に基づいて算出している。

5.確定予算と修正予算

予算管理システム関連ファイル（確定ファイル、修正ファイル、年度データ、等）の簡単なまとめを示す。【図8】

実務上、次月の予算を当月中旬までに決定したものを最終的な予算としている。これを当社では「確定予算」と呼んでいる。仕入、売上、在庫のそれぞれについて予算を確定する。前月に決定した「確定予算」に対する当月中の変更は、仕入予算のみ対応することとし、売上や在庫の状況を見て、仕入予算を修正していく。これを「修正予算」と呼んでいる。

確定予算の入力画面を【図9】に示す。確定予算は金額に加えて数量も登録している。各部門の単価は前年実績に基づき（実績高÷数量）で算出し、今回の確定予算高を前年単価で割って今回の数量を算出する。

仕入予算の修正は、週別に入力できるようにしている。仕入予算は週単位で検討しており、月の1週目、2週目レベルで、各店舗に仕入商品を配布するか見るためである。また、この結果は、発注システムの仕入検討画面の週別表示に反映させている。

予算管理で最も特徴的なのは、発注金額が各部門の仕入予算を上回ると発注システムで発注登録ができないことで、過剰な仕入を未然に防ぐ役目を仕入予算に持たせている。

6.今後の課題

以上、予算管理システムについて述べてきた。部門、店番などのカテゴリーによりさまざまなデータを作成しており、過去データの蓄積がかなりある。情報部門はもとより予算担当部門ではデータマイニングをさらに充実させて、迅速かつ正確なデータの提供が急務である。Delphiで今後どこまでできるか、さらに突き詰めていきたい。

M

図2　年初予算決定フロー

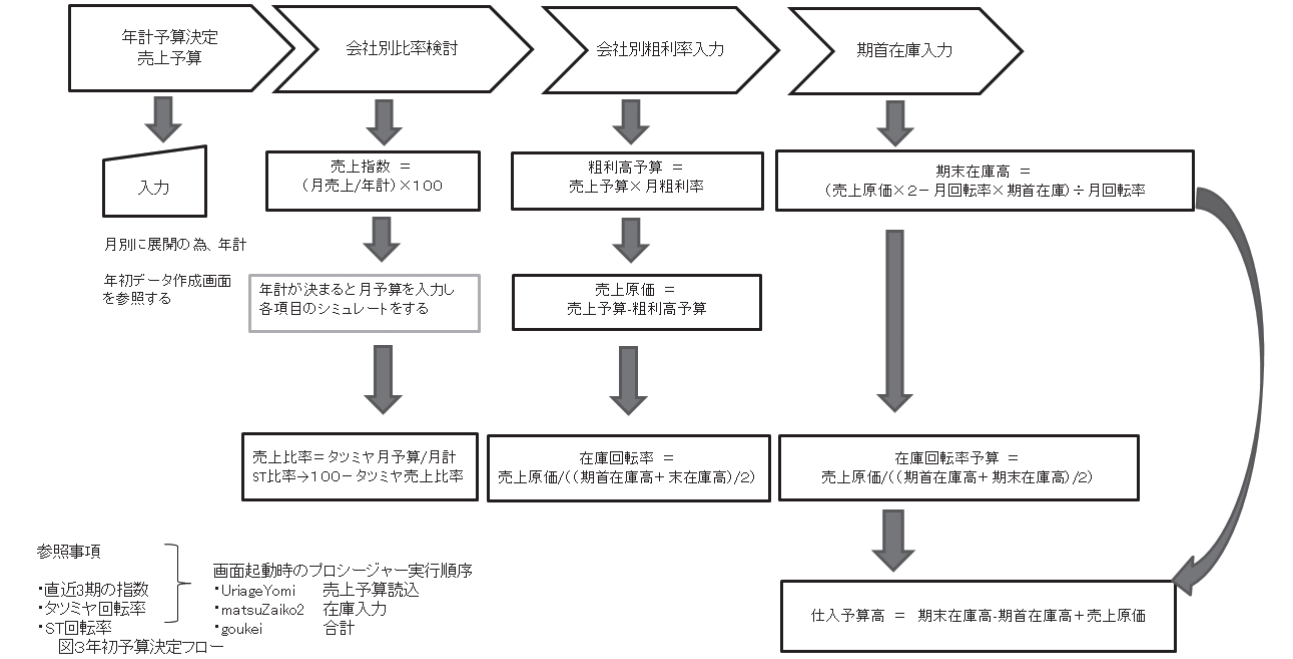
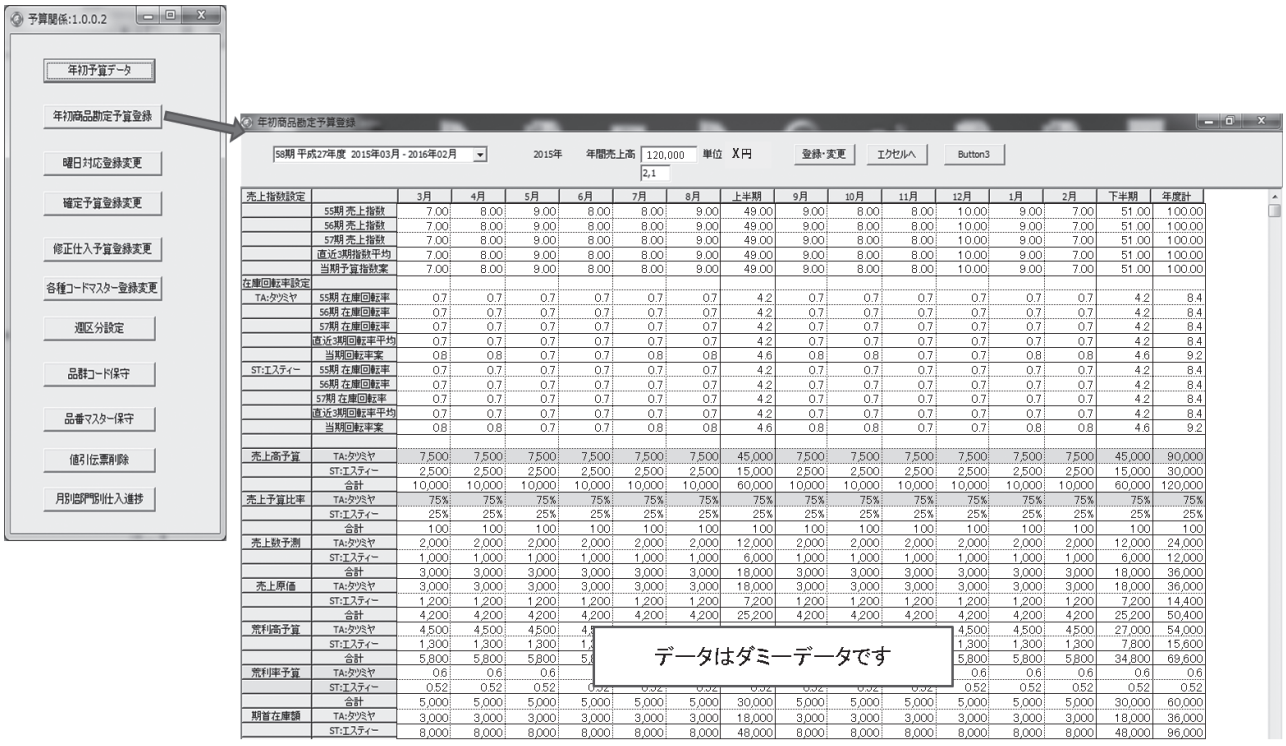


図3年初予算決定フロー

図3　予算管理メニューと年初商品勘定予算登録画面



売上指数設定		3月	4月	5月	6月	7月	8月	上半期	9月	10月	11月	12月	1月	2月	下半期	年度計
	55期 売上指数	7.00	8.00	9.00	8.00	8.00	9.00	49.00	9.00	8.00	8.00	10.00	9.00	7.00	51.00	100.00
	56期 売上指数	7.00	8.00	9.00	8.00	8.00	9.00	49.00	9.00	8.00	8.00	10.00	9.00	7.00	51.00	100.00
	57期 売上指数	7.00	8.00	9.00	8.00	8.00	9.00	49.00	9.00	8.00	8.00	10.00	9.00	7.00	51.00	100.00
	直近3期指数平均	7.00	8.00	9.00	8.00	8.00	9.00	49.00	9.00	8.00	8.00	10.00	9.00	7.00	51.00	100.00
	当期予算指数案	7.00	8.00	9.00	8.00	8.00	9.00	49.00	9.00	8.00	8.00	10.00	9.00	7.00	51.00	100.00
在庫回転率設定																
TA:タツミヤ	55期 在庫回転率	0.7	0.7	0.7	0.7	0.7	0.7	4.2	0.7	0.7	0.7	0.7	0.7	0.7	4.2	8.4
	56期 在庫回転率	0.7	0.7	0.7	0.7	0.7	0.7	4.2	0.7	0.7	0.7	0.7	0.7	0.7	4.2	8.4
	57期 在庫回転率	0.7	0.7	0.7	0.7	0.7	0.7	4.2	0.7	0.7	0.7	0.7	0.7	0.7	4.2	8.4
	直近3期回転率平均	0.7	0.7	0.7	0.7	0.7	0.7	4.2	0.7	0.7	0.7	0.7	0.7	0.7	4.2	8.4
	当期回転率案	0.8	0.8	0.7	0.7	0.8	0.8	4.6	0.8	0.8	0.7	0.7	0.8	0.8	4.6	9.2
ST:エスティー	55期 在庫回転率	0.7	0.7	0.7	0.7	0.7	0.7	4.2	0.7	0.7	0.7	0.7	0.7	0.7	4.2	8.4
	56期 在庫回転率	0.7	0.7	0.7	0.7	0.7	0.7	4.2	0.7	0.7	0.7	0.7	0.7	0.7	4.2	8.4
	57期 在庫回転率	0.7	0.7	0.7	0.7	0.7	0.7	4.2	0.7	0.7	0.7	0.7	0.7	0.7	4.2	8.4
	直近3期回転率平均	0.7	0.7	0.7	0.7	0.7	0.7	4.2	0.7	0.7	0.7	0.7	0.7	0.7	4.2	8.4
	当期回転率案	0.8	0.8	0.7	0.7	0.8	0.8	4.6	0.8	0.8	0.7	0.7	0.8	0.8	4.6	9.2
売上高予算	TA:タツミヤ	7,500	7,500	7,500	7,500	7,500	7,500	45,000	7,500	7,500	7,500	7,500	7,500	7,500	45,000	90,000
	ST:エスティー	2,500	2,500	2,500	2,500	2,500	2,500	15,000	2,500	2,500	2,500	2,500	2,500	2,500	15,000	30,000
	合計	10,000	10,000	10,000	10,000	10,000	10,000	60,000	10,000	10,000	10,000	10,000	10,000	10,000	60,000	120,000
売上予算比率	TA:タツミヤ	75%	75%	75%	75%	75%	75%	75%	75%	75%	75%	75%	75%	75%	75%	75%
	ST:エスティー	25%	25%	25%	25%	25%	25%	25%	25%	25%	25%	25%	25%	25%	25%	25%
	合計	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
売上数予測	TA:タツミヤ	2,000	2,000	2,000	2,000	2,000	2,000	12,000	2,000	2,000	2,000	2,000	2,000	2,000	12,000	24,000
	ST:エスティー	1,000	1,000	1,000	1,000	1,000	1,000	6,000	1,000	1,000	1,000	1,000	1,000	1,000	6,000	12,000
	合計	3,000	3,000	3,000	3,000	3,000	3,000	18,000	3,000	3,000	3,000	3,000	3,000	3,000	18,000	36,000
売上原価	TA:タツミヤ	3,000	3,000	3,000	3,000	3,000	3,000	18,000	3,000	3,000	3,000	3,000	3,000	3,000	18,000	36,000
	ST:エスティー	1,200	1,200	1,200	1,200	1,200	1,200	7,200	1,200	1,200	1,200	1,200	1,200	1,200	7,200	14,400
	合計	4,200	4,200	4,200	4,200	4,200	4,200	25,200	4,200	4,200	4,200	4,200	4,200	4,200	25,200	50,400
荒利高予算	TA:タツミヤ	4,500	4,500	4,500	4,5							4,500	4,500	4,500	27,000	54,000
	ST:エスティー	1,300	1,300	1,300	1,3							1,300	1,300	1,300	7,800	15,600
	合計	5,800	5,800	5,800	5,8							5,800	5,800	5,800	34,800	69,600
荒利率予算	TA:タツミヤ	0.6	0.6	0.6								0.6	0.6	0.6	0.6	0.6
	ST:エスティー	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52	0.52
	合計	5,000	5,000	5,000	5,000	5,000	5,000	30,000	5,000	5,000	5,000	5,000	5,000	5,000	30,000	60,000
期首在庫額	TA:タツミヤ	3,000	3,000	3,000	3,000	3,000	3,000	18,000	3,000	3,000	3,000	3,000	3,000	3,000	18,000	36,000
	ST:エスティー	8,000	8,000	8,000	8,000	8,000	8,000	48,000	8,000	8,000	8,000	8,000	8,000	8,000	48,000	96,000

計算式1

データとしては過去5年間の売上高実績を使用し2年間の予測値を算出する。
求める予測値は年次系列で常に1の一定間隔のデータであり、チェビシェフの直交多項式を用いると予測年度の売上高 y は、以下で求めることができる。

$$y=b_0+b_1(t-\bar{t})+b_2\left\{(t-\bar{t})^2-\frac{k^2-1}{12}\cdot h_A^2\right\}+b_3\left\{(t-\bar{t})^3-\frac{3k^2-7}{20}(t-\bar{t})h_A^2\right\}+\cdots\cdots$$

1次項　　2次項　　3次項

t = 予測する年度　　$\bar{t}$ = 観測年度(過去5年間)の平均値
h_A = 年度の間隔 (当システムでは常に1)　　k = 使用した年度数 (当システムでは5)
b_1、b_2、b_3の項を1次項、2次項、3次項とし、各年度データをA_iで表すと、前式の多項式の係数を以下で求める

$$b_0=\bar{A}_i,\quad b_1=\frac{\sum A_i W_i^{(1)}}{r\cdot\lambda S\cdot h_A^1},\quad b_2=\frac{\sum A_i W_i^{(2)}}{r\cdot\lambda S\cdot h_A^2},\quad b_3=\frac{\sum A_i W_i^{(3)}}{r\cdot\lambda S\cdot h_A^3},\ \cdots\cdots$$

先ず1次項のみを求め、2次項以降については分散分析の中で誤差項にプールし、
1次項 $b_1(t-\bar{t})$ の式を使って求める。つまり$b_1=\frac{\sum A_i W_i^{(1)}}{r\cdot\lambda S\cdot h_A}$で求めることができる。
各変動の分解はつぎの公式による

	変動	自由度	分散	F値	
全変動	$S_T=\sum y_{ij}^2-T^2/n$	$f_T=n-1$			
傾向変動	$S_A=\sum A_i^2/j-T^2/n$	$f_A=i-1$		S_A/f_A	$(S_A/f_A)/(S_E/f_E)$
季節変動	$S_B=\sum B_j^2/i-T^2/n$	$f_B=j-1$		S_B/f_B	$(S_B/f_B)/(S_E/f_E)$
誤差変動	$S_E=S_T-S_A-S_B$	$f_E=f_T-f_A-f_B$	S_E/f_E		

ここで、S_Tは全変動、iは実績年数の数で5、Jは月数で12である。
Tは年の合計値。nは全データ数で12×5=60である。iは1から5、jは1から12まで。

ソース1　分散グリッド表示のソースコード

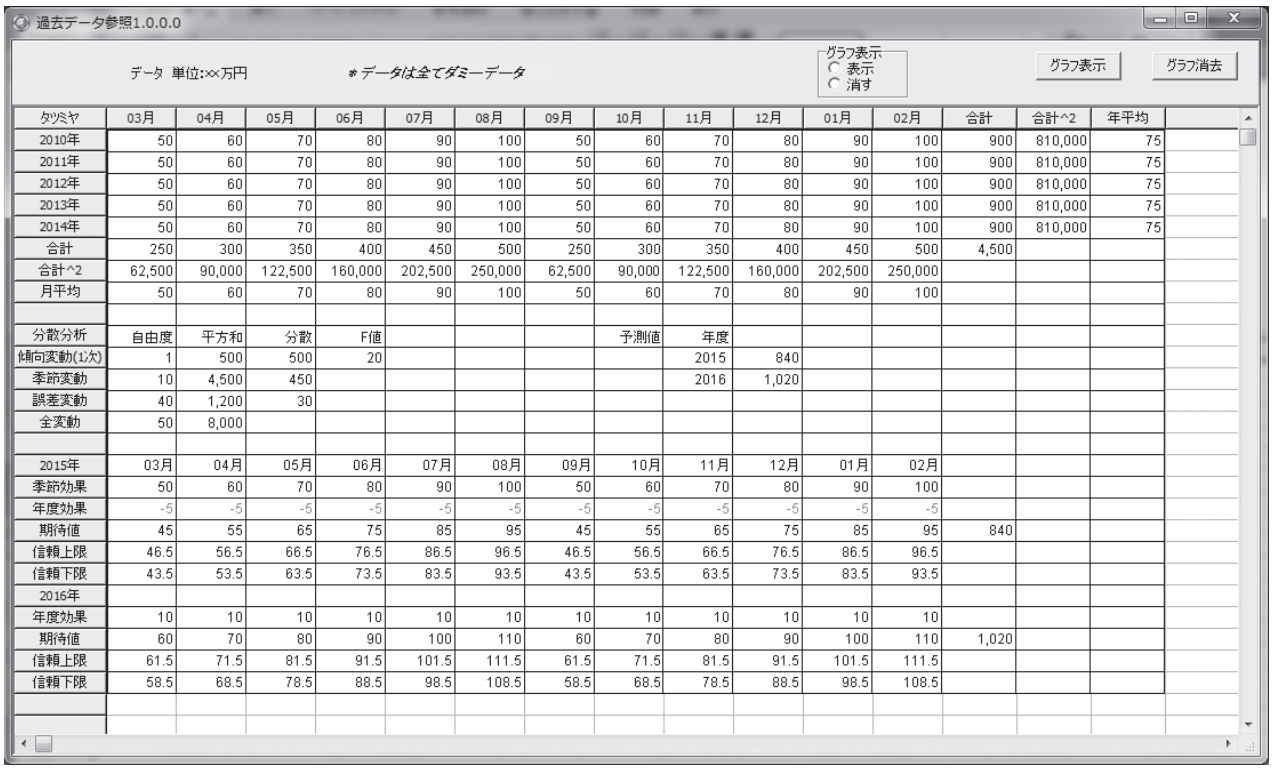
```
procedure TfrmTbunsan.Grhyoji;
var
I,j:integer;
nen,intNen:integer;
strYYMM,YY,MM:string;
Gtotal,Ntotal,Rtotal,Ltotal,T,Ai,BJ,Yij:Double;
St,Sa,Sb,Se:double;
S1,S2,S3,ST1:double;
Aw1,Aw2,Aw3:double;
B0,B1:double;
meanT:double;
M0:array[1..12] of string;
L0:Array[1..12] of double;
L1:Array[1..12] of double;
CTotal:array[1..5] of double;

begin
//変数の初期化
St:=0;Se:=0;Sa:=0;Sb:=0;
Aw1:=0;Aw2:=0;Aw3:=0;
S1:=0;S2:=0;S3:=0;ST1:=0;
//月を配列にいれる
m0[1]:='03';m0[2]:='04';m0[3]:='05';m0[4]:='06';m0[5]:='07';m0[6]:='08';
m0[7]:='09';m0[8]:='10';m0[9]:='11';m0[10]:='12';m0[11]:='01';m0[12]:='02';
  stringGrid1.RowCount:=52;
  stringGRid1.ColCount:=20;
  stringGRid1.ColWidths[0]:=50;
  //**************************************************
  //月の表示　　変数クリヤー
  //**************************************************
  stringGRid1.Cells[0,0]:='タツミヤ';
  Gtotal:=0; Ntotal:=0;Rtotal:=0; Ltotal:=0;T:=0;Yij:=0;
  for i:=1  to 12 do
  begin
    stringGrid1.ColWidths[i]:=60;
    stringGrid1.Cells[i,0]:=m0[i]+'月';
    L0[i]:=0;L1[i]:=0;   //月計
  end;
  for I := 1 to 5 do
    Ctotal[i]:=0;   //年度計

  stringGrid1.Cells[13,0]:='合計';
  stringGrid1.Cells[14,0]:='合計^2';
  stringGrid1.Cells[15,0]:='年平均';
  //
  stringGrid1.Cells[0,6]:='合計';
  stringGrid1.Cells[0,7]:='合計^2';
  stringGrid1.Cells[0,8]:='月平均';
  //表題列
  nen:=strToint(copy(frmDkento.edit4.Text,1,4));
  nen:=nen-6;
  //データ入力
  with DataModule2 do
  begin
  //
      sqlQuery1.SQL.Clear;
      sqlQuery1.SQL.Add('SELECT SUM(TURIGK) FROM FLIB450/ZKTBUTREC');
      sqlQuery1.SQL.Add('WHERE TYYMM=:YYMM');
```

```
    //
    case frmDkento.RadioGroup1.ItemIndex of
    1:
    begin
        sqlQuery1.SQL.Add('AND TKAISY=:TKAISY');
        sqlQuery1.ParamByName('TKAISY').AsString:='T';
    end;
    2:
    begin
        sqlQuery1.SQL.Add('AND TKAISY=:TKAISY');
        sqlQuery1.ParamByName('TKAISY').AsString:='S';
    end;
    end;
//
   Yij:=0;
   for i:=1 to 5 do //年度
   begin
   meant:=0;
    stringGrid1.Cells[0,I]:=intTostr(nen+I)+'年';
    YY:=intTostr(nen+I);
    //meanT:=meant+(YY);
    Ctotal[j]:=0 ; Ntotal:=0;
      for J := 1 to 12 do  //月
      begin
          strYYMM:=YY+M0[j];
          if J>=11 then
          begin
             intNen:=0;
             intNen:=strToint(YY);
             intNen:=intNen+1;
             YY:=intTostr(intNen);
          end;
          sqlQuery1.ParamByName('YYMM').AsInteger:=strToint(strYYMM);
          try
              sqlQuery1.Open;
              ur[I,J]:=(sqlQuery1.FieldByName('00001').AsFloat/100000000);//データ格納 I:年  J:月
              stringGrid1.Cells[J,I]:=format('%n',[ur[i,j]]);
              //Ctotal[I]:=Ctotal[I]+ur[i,j];   //年度計
              Yij:=Yij+Ur[i,j]*ur[i,j]; //データの2乗和
          finally
              sqlQuery1.Close;
          end;
          //

      end;
      //
   end;
   for I := 1 to 5 do   //年度計
   begin
       for J:=1 to 12 do
       begin
          Ctotal[I]:=Ctotal[i]+ur[I,j];
          L0[J]:=L0[J]+Ur[I,j];
          //Bj :=Bj + L0[J]*L0[J];
       end;
       stringGRid1.Cells[13,I]:=format('%n',[Ctotal[I]]);
       stringGRid1.Cells[14,I]:=format('%n',[Ctotal[I]*CTOTAL[I]]); //2乗和
       stringGRid1.Cells[15,I]:=format('%n',[Ctotal[I]/12]);        //年の平均
       //
       Gtotal:=Gtotal+Ctotal[I];
       Ai:=Ai+Ctotal[I]*Ctotal[I];
       case I of
```

```
        1:
        begin
          aw1:=Aw1+Ctotal[1]*-2;
          Aw2:=Aw2+CTotal[1]*2;
          Aw3:=Aw3+CTotal[1]*-1;
        end;
        2:
        begin
          aw1:=Aw1+Ctotal[2]*-1;
          Aw2:=Aw2+CTotal[2]*-1;
          Aw3:=Aw3+CTotal[2]*2;
        end;
        3:
        begin
          aw1:=Aw1+Ctotal[3]*0;
          Aw2:=Aw2+CTotal[3]*-2;
          Aw3:=Aw3+CTotal[3]*0;
        end;
        4:
        begin
          aw1:=Aw1+Ctotal[4];
          Aw2:=Aw2+CTotal[4]*-1;
          Aw3:=Aw3+CTotal[4]*-2;
        end;
        5:
        begin
          aw1:=Aw1+Ctotal[5]*2;
          Aw2:=Aw2+CTotal[5]*2;
          Aw3:=Aw3+CTotal[5]*1;
        end;
        end;
        Series1.Add(Ctotal[i],stringGRid1.Cells[0,I]);
    end;
    for J:= 1 to 12 do
    begin
        stringGrid1.Cells[J,6]:=format('%n',[L0[J]]);
        stringGrid1.Cells[J,7]:=format('%n',[L0[J]*L0[J]]);
        stringGrid1.Cells[J,8]:=format('%n',[L0[J]/5]);
        bj:=bj+L0[J]*L0[J];
    end;
      stringGrid1.Cells[13,6]:=format('%n',[Gtotal]);
      stringGRid1.Cells[13,7]:=format('%n',[BJ]);
      stringGRid1.Cells[14,6]:=format('%n',[Ai]);
      //
end;
//
stringGrid1.Cells[1,9]:=format('%n',[Yij]);
stringGrid1.Cells[2,9]:=format('%n',[Gtotal]);
stringGrid1.Cells[3,9]:=format('%n',[Ai]);
stringGrid1.Cells[4,9]:=format('%n',[Bj]);
//
Sa:= Ai/12-Gtotal*Gtotal/60;
Sb:= bj/5- GTotal*Gtotal/60;
St:= yij  -GTotal*Gtotal/60;
S1:=aw1*aw1/10;
S2:=Aw2*Aw2/14;
S3:=Aw3*Aw3/10;
ST1:=Ai-Gtotal*Gtotal/5;
//
stringGRid1.Cells[0,10]:='分散分析';
stringGRid1.Cells[0,11]:='1次効果';
stringGRid1.Cells[0,12]:='2次効果';
```

```
  stringGRid1.Cells[0,13]:='3次効果';
  stringGRid1.Cells[0,14]:='誤差';
  stringGRid1.Cells[0,15]:='誤差プール';
  //stringGRid1.Cells[1,11]:='4';
  //stringGRid1.Cells[2,11]:=format('%n',[Sa]);
  //stringGRid1.Cells[3,11]:=format('%n',[Sa/4]);
  //stringGRid1.Cells[4,11]:=format('%n',[(Sa/4)/((St-Sa-Sb)/44)]);
  stringGRid1.Cells[1,11]:='1';
  stringGRid1.Cells[2,11]:=format('%n',[S1]);
  stringGRid1.Cells[3,11]:=format('%n',[S1/1]);
  stringGRid1.Cells[1,12]:='1';
  stringGRid1.Cells[2,12]:=format('%n',[S2]);
  stringGRid1.Cells[3,12]:=format('%n',[S2/1]);
  stringGRid1.Cells[1,13]:='1';
  stringGRid1.Cells[2,13]:=format('%n',[S3]);
  stringGRid1.Cells[3,13]:=format('%n',[S3/1]);
  stringGRid1.Cells[1,14]:='1';
  stringGRid1.Cells[2,14]:=format('%n',[ST1-S1-S2-S3]);
  stringGRid1.Cells[1,15]:='3';
  stringGRid1.Cells[2,15]:=format('%n',[S2+S3+(ST1-S1-S2-S3)]);
  stringGRid1.Cells[3,15]:=format('%n',[(S2+S3+(ST1-S1-S2-S3))/3]);
  stringGRid1.Cells[4,11]:=format('%n',[(S1/1)/((S2+S3+(ST1-S1-S2-S3))/3)]);
  stringGrid1.Cells[1,16]:='4';
  stringGrid1.Cells[2,16]:=format('%n',[St1]);
  //
  stringGRid1.Cells[0,18]:='季節変動';
  stringGRid1.Cells[1,18]:='11';
  stringGRid1.Cells[2,18]:=format('%n',[Sb]);
  stringGRid1.Cells[3,18]:=format('%n',[Sb/11]);
  stringGRid1.Cells[4,18]:=format('%n',[(Sb/11)/((St-Sa-Sb)/44)]);
  stringGRid1.Cells[0,19]:='誤差変動';
  stringGRid1.Cells[2,19]:=format('%n',[St-Sa-Sb]);
  stringGRid1.Cells[3,19]:=format('%n',[(St-Sa-Sb)/44]);


  stringGRid1.Cells[1,10]:='自由度';
  stringGRid1.Cells[1,19]:='44';

  stringGRid1.Cells[2,10]:='平方和';
  stringGRid1.Cells[3,10]:='分散';
  stringGRid1.Cells[4,10]:='F値';
  if (Sa/4)/((St-Sa-Sb)/44)>=4.06 then
     stringGRid1.Cells[5,11]:='*';
  if (Sa/4)/((St-Sa-Sb)/44)>=7.20 then
     stringGRid1.Cells[5,11]:='**';
  if (Sb/4)/((St-Sa-Sb)/44)>=4.06 then
     stringGRid1.Cells[5,18]:='*';
  if (Sb/11)/((St-Sa-Sb)/44)>=7.20 then
     stringGRid1.Cells[5,18]:='**';
  stringGRid1.Cells[0,20]:='全変動';
  stringGRid1.Cells[1,20]:='59';
  stringGRid1.Cells[2,20]:=format('%n',[St]);
  //予測値
  b0:=Gtotal/5;
  B1:=aw1/10;
  stringGrid1.Cells[0,22]:='予測値';
  stringGrid1.Cells[0,23]:='年度';
  stringGrid1.Cells[1,23]:=intTostr(nen+6);
  stringGrid1.Cells[2,23]:=Format('%n',[b0+b1*((Nen+6)-meanYear)]);
  stringGrid1.Cells[1,24]:=intTostr(nen+7);
  stringGrid1.Cells[2,24]:=Format('%n',[b0+b1*((Nen+7)-meanYear)]);
end;
```

図4　予測計算結果



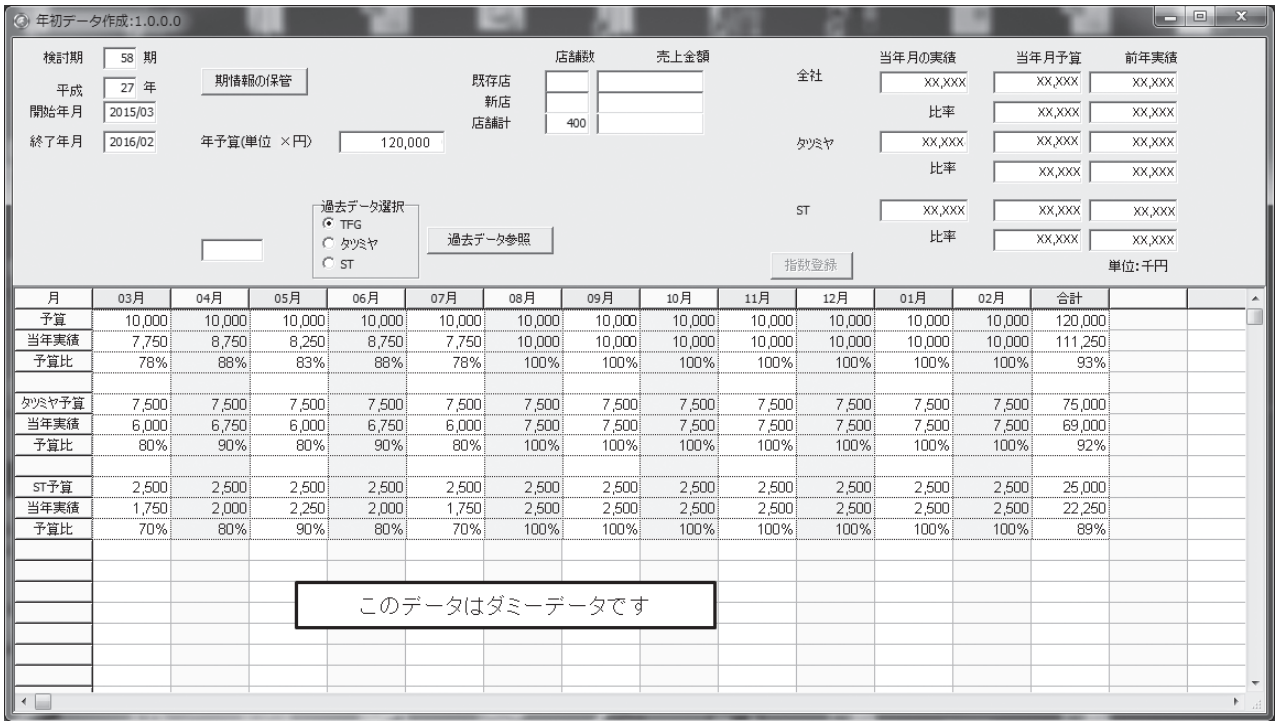
過去データ参照1.0.0.0

データ 単位:××万円　　＊データは全てダミーデータ

グラフ表示: ○表示 ○消す　　グラフ表示　　グラフ消去

タツミヤ	03月	04月	05月	06月	07月	08月	09月	10月	11月	12月	01月	02月	合計	合計^2	年平均
2010年	50	60	70	80	90	100	50	60	70	80	90	100	900	810,000	75
2011年	50	60	70	80	90	100	50	60	70	80	90	100	900	810,000	75
2012年	50	60	70	80	90	100	50	60	70	80	90	100	900	810,000	75
2013年	50	60	70	80	90	100	50	60	70	80	90	100	900	810,000	75
2014年	50	60	70	80	90	100	50	60	70	80	90	100	900	810,000	75
合計	250	300	350	400	450	500	250	300	350	400	450	500	4,500		
合計^2	62,500	90,000	122,500	160,000	202,500	250,000	62,500	90,000	122,500	160,000	202,500	250,000			
月平均	50	60	70	80	90	100	50	60	70	80	90	100			
分散分析	自由度	平方和	分散	F値				予測値	年度						
傾向変動(1次)	1	500	500	20					2015	840					
季節変動	10	4,500	450						2016	1,020					
誤差変動	40	1,200	30												
全変動	50	8,000													
2015年	03月	04月	05月	06月	07月	08月	09月	10月	11月	12月	01月	02月			
季節効果	50	60	70	80	90	100	50	60	70	80	90	100			
年度効果	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5	-5			
期待値	45	55	65	75	85	95	45	55	65	75	85	95	840		
信頼上限	46.5	56.5	66.5	76.5	86.5	96.5	46.5	56.5	66.5	76.5	86.5	96.5			
信頼下限	43.5	53.5	63.5	73.5	83.5	93.5	43.5	53.5	63.5	73.5	83.5	93.5			
2016年															
年度効果	10	10	10	10	10	10	10	10	10	10	10	10			
期待値	60	70	80	90	100	110	60	70	80	90	100	110	1,020		
信頼上限	61.5	71.5	81.5	91.5	101.5	111.5	61.5	71.5	81.5	91.5	101.5	111.5			
信頼下限	58.5	68.5	78.5	88.5	98.5	108.5	58.5	68.5	78.5	88.5	98.5	108.5			

図5　年初データ作成



年初データ作成:1.0.0.0

検討期 58 期　　平成 27 年　　開始年月 2015/03　　終了年月 2016/02

期情報の保管

年予算(単位 ×円) 120,000

	店舗数	売上金額
既存店		
新店		
店舗計	400	

過去データ選択: ●TFG ○タツミヤ ○ST　　過去データ参照

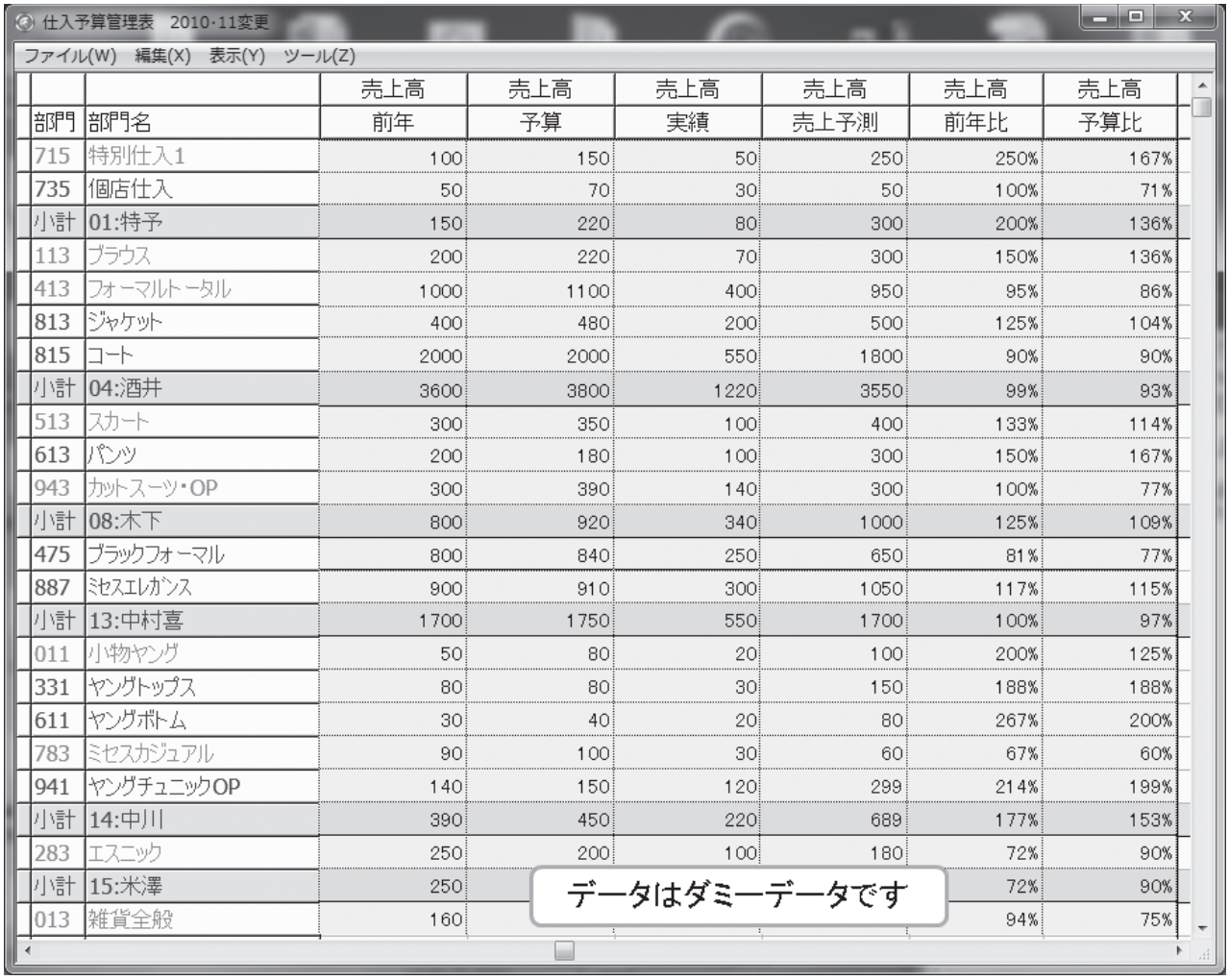
	当年月の実績	当年月予算	前年実績
全社	XX,XXX	XX,XXX	XX,XXX
比率		XX,XXX	XX,XXX
タツミヤ	XX,XXX	XX,XXX	XX,XXX
比率		XX,XXX	XX,XXX
ST	XX,XXX	XX,XXX	XX,XXX
比率		XX,XXX	XX,XXX

指数登録　　単位:千円

月	03月	04月	05月	06月	07月	08月	09月	10月	11月	12月	01月	02月	合計
予算	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000	10,000	120,000
当年実績	7,750	8,750	8,250	8,750	7,750	10,000	10,000	10,000	10,000	10,000	10,000	10,000	111,250
予算比	78%	88%	83%	88%	78%	100%	100%	100%	100%	100%	100%	100%	93%
タツミヤ予算	7,500	7,500	7,500	7,500	7,500	7,500	7,500	7,500	7,500	7,500	7,500	7,500	75,000
当年実績	6,000	6,750	6,000	6,750	6,000	7,500	7,500	7,500	7,500	7,500	7,500	7,500	69,000
予算比	80%	90%	80%	90%	80%	100%	100%	100%	100%	100%	100%	100%	92%
ST予算	2,500	2,500	2,500	2,500	2,500	2,500	2,500	2,500	2,500	2,500	2,500	2,500	25,000
当年実績	1,750	2,000	2,250	2,000	1,750	2,500	2,500	2,500	2,500	2,500	2,500	2,500	22,250
予算比	70%	80%	90%	80%	70%	100%	100%	100%	100%	100%	100%	100%	89%

このデータはダミーデータです

図6　仕入予算管理表



部門	部門名	売上高 前年	売上高 予算	売上高 実績	売上高 売上予測	売上高 前年比	売上高 予算比
715	特別仕入1	100	150	50	250	250%	167%
735	個店仕入	50	70	30	50	100%	71%
小計	01:特予	150	220	80	300	200%	136%
113	ブラウス	200	220	70	300	150%	136%
413	フォーマルトータル	1000	1100	400	950	95%	86%
813	ジャケット	400	480	200	500	125%	104%
815	コート	2000	2000	550	1800	90%	90%
小計	04:酒井	3600	3800	1220	3550	99%	93%
513	スカート	300	350	100	400	133%	114%
613	パンツ	200	180	100	300	150%	167%
943	カットスーツ・OP	300	390	140	300	100%	77%
小計	08:木下	800	920	340	1000	125%	109%
475	ブラックフォーマル	800	840	250	650	81%	77%
887	ミセスエレガンス	900	910	300	1050	117%	115%
小計	13:中村喜	1700	1750	550	1700	100%	97%
011	小物ヤング	50	80	20	100	200%	125%
331	ヤングトップス	80	80	30	150	188%	188%
611	ヤングボトム	30	40	20	80	267%	200%
783	ミセスカジュアル	90	100	30	60	67%	60%
941	ヤングチュニックOP	140	150	120	299	214%	199%
小計	14:中川	390	450	220	689	177%	153%
283	エスニック	250	200	100	180	72%	90%
小計	15:米澤	250				72%	90%
013	雑貨全般	160				94%	75%

図7　曜日対応

前年データ 日	曜日	前年売上	本年曜日 日	曜日	週区分	前年曜日対応 日	曜日	前年売上	シェア	修正シェア	本年 日々予算	予算 累計予算	年予算対 累計シェア	週間シェア	前年売上	シェア	日々予算	累計シェア	週間シェア	前年売上	シェア	日々予算	累計シェア	エスティー 週間シェア
01	金	36,764	01	土	1	02	土	5,000	3.0%		5,500	5,500	3.4%		3,300	2.0%	3,630	3.0%		1,700	1.0%	1,870	3.4%	
02	土	35,461	02	日	1	03	日	6,000	3.7%	6.7%	6,600	12,100	7.0%	10.4%	3,960	2.4%	4,356	3.7%	6.7%	2,040	1.2%	2,244	4.0%	7.4%
03	日	36,943	03	月	2	04	月	4,000	2.4%		4,400	16,500	9.5%		2,640	1.6%	2,904	2.4%		1,360	0.8%	1,496	2.7%	
04	月	29,909	04	火	2	05	火	5,000	3.0%		5,500	22,000	12.5%		3,300	2.0%	3,630	3.0%		1,700	1.0%	1,870	3.4%	
05	火	31,998	05	水	2	06	水	6,000	3.7%		6,600	28,600	16.2%		3,960	2.4%	4,356	3.7%		2,040	1.2%	2,244	4.0%	
06	水	30,925	06	木	2	07	木	7,000	4.3%		7,700	36,300	20.4%		4,620	2.8%	5,082	4.3%		2,380	1.5%	2,618	4.7%	
07	木	31,081	07	金	2	08	金	6,000	3.7%		6,600	42,900	24.1%		3,960	2.4%	4,356	3.7%		2,040	1.2%	2,244	4.0%	
08	金	31,944	08	土	2	09	土	5,000	3.0%		5,500	48,400	27.1%		3,300	2.0%	3,630	3.0%		1,700	1.0%	1,870	3.4%	
09	土	30,581	09	日	2	10	日	4,000	2.4%	22.6%	4,400	52,800	29.6%		2,640	1.6%	2,904	2.4%	22.6%	1,360	0.8%	1,496	2.7%	24.8%
10	日	29,236	10	月	3	11	月	5,000	3.0%		5,500	58,300	32.6%		3,300	2.0%	3,630	3.0%		1,700	1.0%	1,870	3.4%	
11	月	32,113	11	火	3	12	火	6,000	3.7%		6,600	64,900	36.3%		3,960	2.4%	4,356	3.7%		2,040	1.2%	2,244	4.0%	
12	火	32,677	12	水	3	13	水	4,000	2.4%		4,400	69,300	38.7%		2,640	1.6%	2,904	2.4%		1,360	0.8%	1,496	2.7%	
13	水	29,710	13	木	3	14	木	5,000	3.0%		5,500	74,800	41.8%		3,300	2.0%	3,630	3.0%		1,700	1.0%	1,870	3.4%	
14	木	31,695	14	金	3	15	金	6,000	3.7%		6,600	81,400	45.4%		3,960	2.4%	4,356	3.7%		2,040	1.2%	2,244	4.0%	
15	金	42,280	15	土	3	16	土	7,000	4.3%		7,700	89,100	49.7%		4,620	2.8%	5,082	4.3%		2,380	1.5%	2,618	4.7%	
16	土	39,086	16	日	3	17	日	6,000	3.7%	23.8%	6,600	95,700	53.4%		3,960	2.4%	4,356	3.7%	23.8%	2,040	1.2%	2,244	4.0%	26.2%
17	日	41,756	17	月	4	18	月	5,000	3.0%		5,5									1,700	1.0%	1,870	3.4%	
18	月	34,884	18	火	4	19	火	4,000	2.4%		4,4									1,360	0.8%	1,496	2.7%	
19	火	35,694	19	水	4	20	水	5,000	3.0%		5,5									1,700	1.0%	1,870	3.4%	
20	水	33,920	20	木	4	21	木	6,000	3.7%		6,6									2,040	1.2%	2,244	4.0%	
21	木	32,816	21	金	4	22	金	4,000	2.4%		4,400	122,100	68.0%		2,640	1.6%	2,904	2.4%		1,360	0.8%	1,496	2.7%	
22	金	31,269	22	土	4	23	土	5,000	3.0%		5,500	127,600	71.0%		3,300	2.0%	3,630	3.0%		1,700	1.0%	1,870	3.4%	
23	土	35,126	23	日	4	24	日	6,000	3.7%	21.3%	6,600	134,200	74.7%		3,960	2.4%	4,356	3.7%	21.3%	2,040	1.2%	2,244	4.0%	23.5%
24	日	36,178	24	月	5	25	月	7,000	4.3%		7,700	141,900	79.0%		4,620	2.8%	5,082	4.3%		2,380	1.5%	2,618	4.7%	
25	月	32,648	25	火	5	26	火	6,000	3.7%		6,600	148,500	82.6%		3,960	2.4%	4,356	3.7%		2,040	1.2%	2,244	4.0%	
26	火	31,880	26	水	5	27	水	5,000	3.0%		5,500	154,000	85.7%		3,300	2.0%	3,630	3.0%		1,700	1.0%	1,870	3.4%	
27	水	30,628	27	木	5	28	木	4,000	2.4%		4,400	158,400	88.1%		2,640	1.6%	2,904	2.4%		1,360	0.8%	1,496	2.7%	
28	木	31,944	28	金	5	01	金	5,000	3.0%		5,500	163,900	91.2%		3,300	2.0%	3,630	3.0%		1,700	1.0%	1,870	3.4%	

図8　予算ファイルの関係

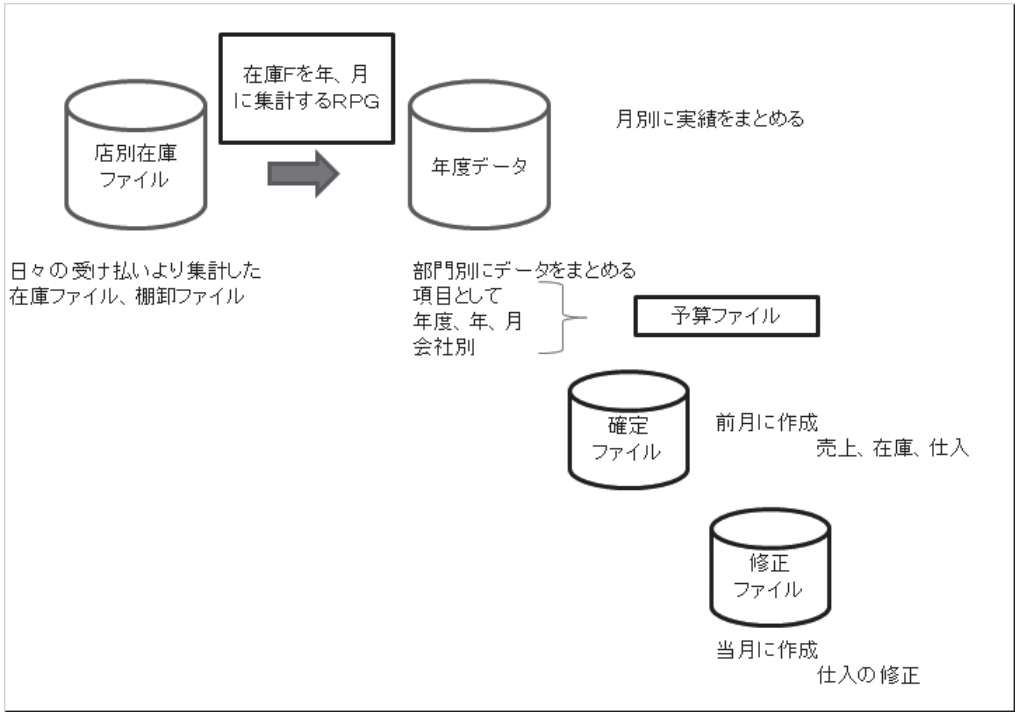
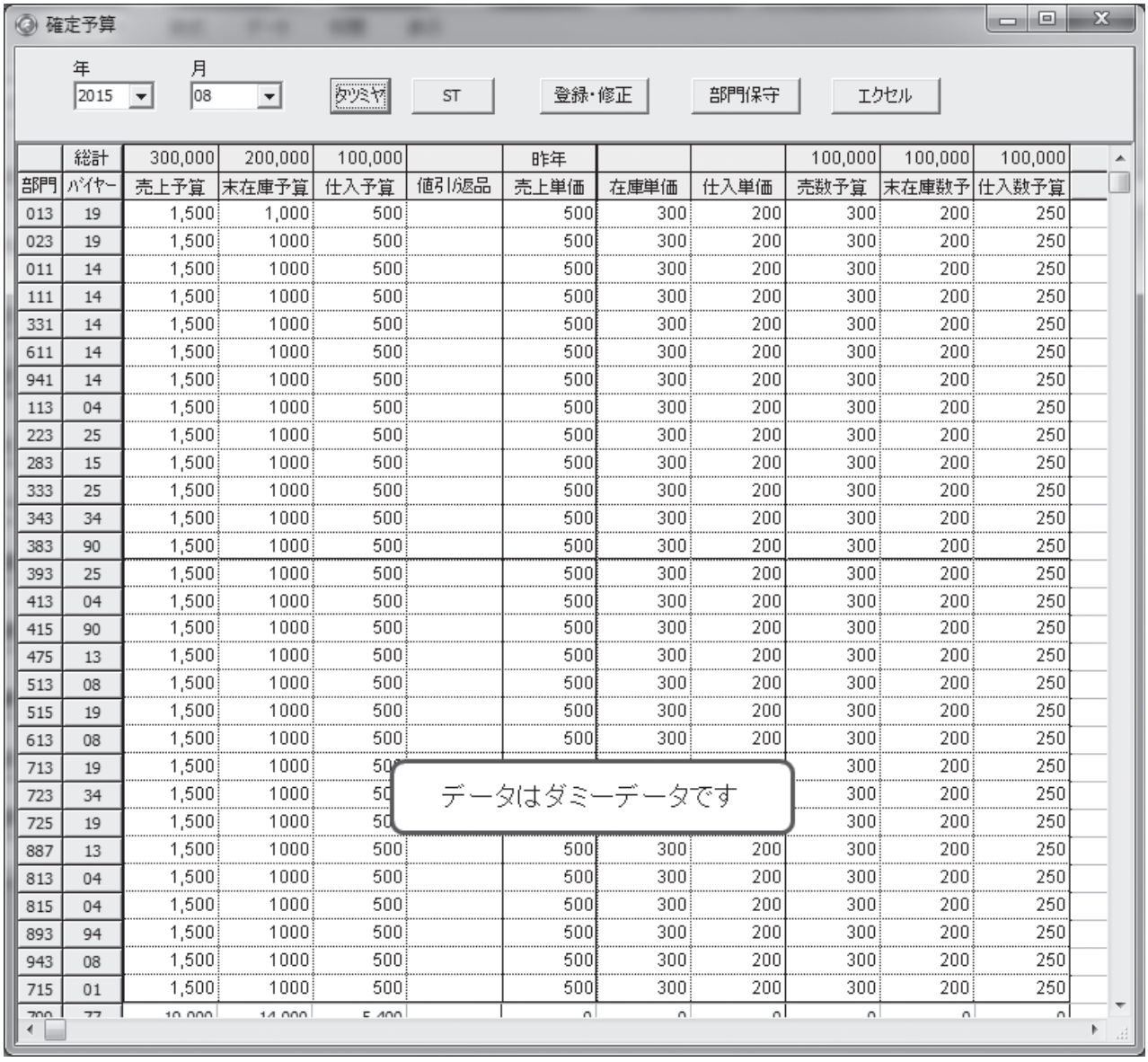


図9　確定予算入力



	総計	300,000	200,000	100,000		昨年			100,000	100,000	100,000
部門	バイヤー	売上予算	末在庫予算	仕入予算	値引/返品	売上単価	在庫単価	仕入単価	売数予算	末在庫数予	仕入数予算
013	19	1,500	1,000	500		500	300	200	300	200	250
023	19	1,500	1000	500		500	300	200	300	200	250
011	14	1,500	1000	500		500	300	200	300	200	250
111	14	1,500	1000	500		500	300	200	300	200	250
331	14	1,500	1000	500		500	300	200	300	200	250
611	14	1,500	1000	500		500	300	200	300	200	250
941	14	1,500	1000	500		500	300	200	300	200	250
113	04	1,500	1000	500		500	300	200	300	200	250
223	25	1,500	1000	500		500	300	200	300	200	250
283	15	1,500	1000	500		500	300	200	300	200	250
333	25	1,500	1000	500		500	300	200	300	200	250
343	34	1,500	1000	500		500	300	200	300	200	250
383	90	1,500	1000	500		500	300	200	300	200	250
393	25	1,500	1000	500		500	300	200	300	200	250
413	04	1,500	1000	500		500	300	200	300	200	250
415	90	1,500	1000	500		500	300	200	300	200	250
475	13	1,500	1000	500		500	300	200	300	200	250
513	08	1,500	1000	500		500	300	200	300	200	250
515	19	1,500	1000	500		500	300	200	300	200	250
613	08	1,500	1000	500		500	300	200	300	200	250
713	19	1,500	1000	50					300	200	250
723	34	1,500	1000	50					300	200	250
725	19	1,500	1000	50					300	200	250
887	13	1,500	1000	500		500	300	200	300	200	250
813	04	1,500	1000	500		500	300	200	300	200	250
815	04	1,500	1000	500		500	300	200	300	200	250
893	94	1,500	1000	500		500	300	200	300	200	250
943	08	1,500	1000	500		500	300	200	300	200	250
715	01	1,500	1000	500		500	300	200	300	200	250
799	77	10,000	14,000	5,400		0	0	0	0	0	0

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
システム部
グループリーダー

西川リビング株式会社
http：//www.nishikawa-living.co.jp/

「眠り」から「健康」を創造し、より快適な暮らしを提案する西川リビング。時代のニーズにあわせた寝具・寝装品や健康機能商品の開発を行っている。商品だけでなく快適な眠りのためのライフスタイルもあわせトータルな提案を行う。創業 1566 年、来年 2016 年には 450 周年を迎える。

西川リビング株式会社は、寝具・寝装品の取り扱いを主力とし、インテリア用品や生活雑貨など、暮らしに関わる幅広い商品の提供を通して、快適な暮らしをサポートする企業である。

本稿では、Delphi で開発した Web システムである「送状データ送信システム」について紹介する。

送状データとは

まず、「送状データ」とは何かを説明する。送状データとは、送状を印刷するためのデータである。当社が自社で印刷するのではなく、送状データを各運送会社に送信し、運送会社側で印刷していただいている。

各運送会社との通信は、従来、ホストコンピュータである IBM i から VAN 会社を経由して全銀 TCP/IP 手順で行っていた。データの内容は、出荷の予定データや実績データなどで、251 バイトの固定長テキストデータである。【図 1】

送状データ送信方法の変更

この送状データの送信方法の変更を検討することになった。最大の理由は、コスト削減である。従来の方法は VAN 会社経由であるため、ランニングコストがかかっていた。その通信費を長年当たり前のように支払ってきたが、単純に考えて、インターネットを利用することによって、通信費を「0」にできる。

そこで、運送会社で使用する専用 Web サイトを当社から提供し、運送会社がインターネット経由で専用 Web サイトにログインし、当社があらかじめ用意した送状データを取得するシステムを考えた。この着想は、量販店関係で普及している流通 BMS がヒントである。

その結果、インターネットを流通 BMS で利用することにより、通信費を大きく削減するシステムを構築できた。また従来とは比較にならないほどのスピードで受信できるようになり、受信時間の短縮も実現した。世間から見れば遅すぎる対応かもしれないが、当社にとっては業務のインターネット利用を進めていくきっかけとなったシステムである。

システム設計

送状データはもともと固定長で、運送会社側の変換ソフトで変換され取り込まれていた。新しいシステムは、システム品質の維持などを総合的に判断し、運送会社側の変換ソフトをそのまま使用する前提で設計した。そこで、送状データを作成する既存のバッチシステムに、下記の機能を追加するようにした。

- 送状データを作成するバッチシステムに、Web 用ファイルに抽出する機能を追加した。
- 運送会社別に制御するマスターを新設し、Web 対象にするかどうかを選択できるようにした（新旧のシステムを並行して使うことも可能）。
- 運送会社が Web から取得する送状データのファイルは累積型とし、再ダ

図1　リプレイス前のシステム構成

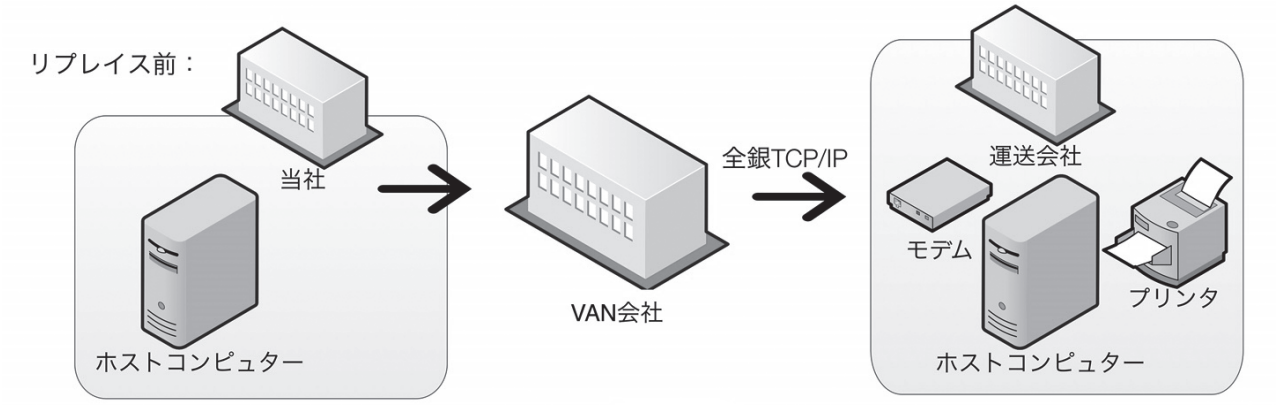


図2　データ受信画面

ウンロードを可能にした。

Web 画面は、極力シンプルになるように意識して設計した。開発したのは、通常使用する「データ受信」画面と、正常にダウンロードできなかった場合に使用する「データ再受信」画面の 2 画面のみである。

「データ再受信」画面では同じデータを何度も取得できるよう、初回ダウンロード時に「バッチNo.」を更新するようにした。この「バッチNo.」で過去のデータを抽出し、繰り返しダウンロードが可能になる。【図 2】【図 3】

サーバーは、SSO（シングルサインオン）対応の Web 管理システムの配下にあり、複数の Web システムが稼働中の既存 Web サーバーを使用した。開発には、Delphi/400 XE5 を使用し、開発パートナーとしてミガロ .の協力を得た。【図 4】

システム開発

ミガロ .と当社で役割を分担し、設計・開発を進めた。ミガロ .は Web アプリケーション関連の開発、当社は IBM i 側のシステム改修の担当である。開発スケジュールは、設計も含めて 1 カ月半とし、当初は非常に順調に進んだ。ミガロ .との取り組みは 4 回目で、今回も非常にスムーズにコミュニケーションがとれたと思う。

並行テスト時のトラブル発覚

Web 側とホスト側の両方のアプリケーション開発が完了し、並行テストの期間を 3 週間ほど設けた。全銀で取得したテキストデータと Web から取得したテキストデータの比較確認が、テストの内容である。ただし、目視による確認ではあてにならないため、両方のテキストデータを読み込んでチェックするツールを Delphi で作成した。

ところが、ここで問題が発覚した。ずれが生じているデータが多数出てきたのである。

前述の通り、送状データは固定長である。IBM i 上の 1 つの項目を SQL で取得しているが、このデータには 2 バイト文字と 1 バイト文字が混在する。ずれの原因は、この項目で使われているシフトコードだった。

全銀で取得したデータの場合、通信ソフトはこのシフトコードを半角スペースで置き換えていたが、SQL で取得した Web データからはシフトコードが消滅していたことが判明した。このため、Web データは、シフトコード分のレングスが短くなっていたのだ。

この時は目の前が真っ暗になった。ミガロ .に協力いただいていろいろ検証してみたが、IBM i 側で対応するしかなさそうだった。藁をもつかむ思いで、ホストシステムの保守・開発で支援していただいている常駐のベテラン SE に相談をしたところ、「Web 用の累積ファイルを作成する RPG プログラムの中で、シフトコードがあれば 1 バイトスペースを付加するロジックを追加すればいいのではないか」とのアドバイスがあった。

当初開発したシステムでは、Web 用の累積ファイルのフィールドを 251 バイトぴったりにしていたが、それを 300 バイトに拡張し、抽出プログラムにスペース付加のロジックを追加した。ドキドキしながらチェックツールで確認して見た。

その結果は・・・・・、データにまったくずれはなく、完全に一致していた。固定長の恐るべき落とし穴であった。

導入後の評価

・コストの改善

前述の通り、コストについては明快に結果が出ている。今回のシステムで使用したのは既存のサーバー等であるので、初期コストは開発費用のみである。数カ月で元が取れる見込みである。

・処理スピードの改善

従来、5 分から 10 分ほどかかっていた受信処理が、ログイン／ログアウト含め 1 分ほどで完了するようになった。

・マイナス点（Web アプリの宿命）

以前はスケジュール機能で自動受信できていたが、Web アプリのため手動となった。

・これからの展開

今回は 1 社のみの導入だったが、他の運送会社への展開も計画中である。

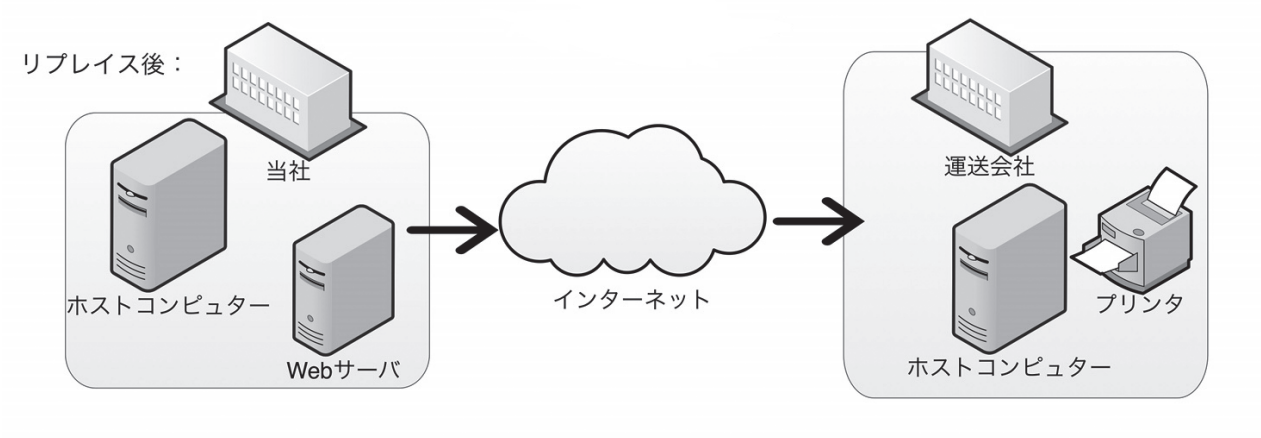
以上、Delphi/400 を活用した Web システム開発を紹介した。

当社では現在、Delphi/400 で開発した Web システムが複数稼働している。また、計画中の開発案件もあり、Delphi/400 は今後もコスト削減や業務改善に大いに貢献してくれるものと期待している。

M

図3　データ再受信画面

図4　リプレイス後のシステム構成

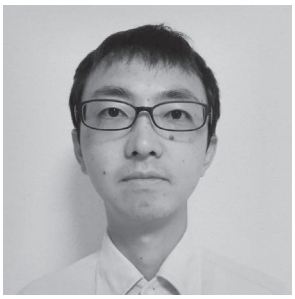


優秀賞

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
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/

1938（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。

業務課題

画面上の StringGrid へ客先ファイルのレコードを取り込む際、長い待ち時間が発生していた。

技術課題

汎用的に客先ファイルを読み込むためのパラメータ管理マスターを用意している。そのパラメータ管理マスターを頻繁にアクセスするため、かなりの処理時間がかかっていた。

技術課題の解決策

ClientDataSet を初期レコードから参照させるように何度も Close → Open するロジックにしていたが、一度 Open したら、繰り返し参照時には、First 機能で初期レコードにカーソルを位置づかせて処理するロジックに変更した。【図 1】【図 2】【図 3】

業務課題解決と効果

画面上の StringGrid への取り込み時間が 5 分かかっていたのが、1 ～ 2 秒で取り込めるようになった。待ち時間がなくなり業務効率が上がった。また、他の客先ファイルでは、1 週間に 1 回、件数の多いファイルを処理する場合がある。その場合、1 時間以上かかっていたが、10 ～ 20 秒ほどで処理できるようになった。

M

図1　ロジック変更点

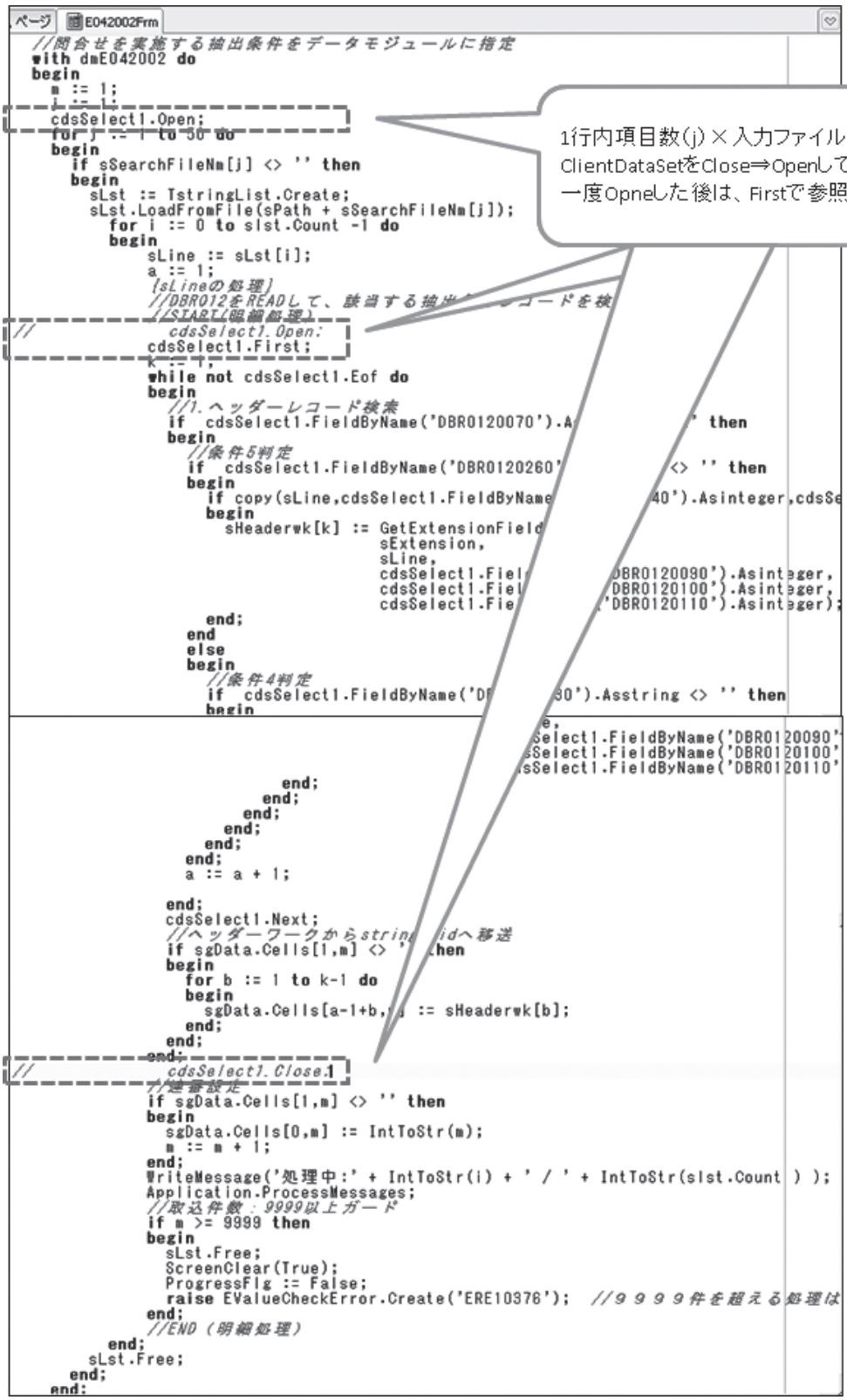

1行内項目数(j)×入力ファイルのレコード数(i)分、
ClientDataSetをClose⇒Openしていたが、
一度Opneした後は、Firstで参照させるように変更した。

図2　処理の流れ


受注媒体利用
得意先C
パターン
021
1
01
取込(F5)
全選択解除(F6)
確認
メッセージ
受注媒体利用
得意先C
パターン
021
1
01
指示区分(F4)
通常
取消
取込(F5)
全選択解除(F6)
登録(F7)
終了(F8)
クリア(F9)
品番
注文数
納入指示年月日
オーダーNo
1 6208816590 000004 131125 J564925
2 6208816590 000004 131125 J564925
3 22X2212670 000500 131125 G321809
4 01010E0816 000800 131128 G321809
5 01010E1020 000600 131127 G321846
6 01010E1025 000500 131127 G321846
7 01010E1030 000500 131129 G321846
8 01010E1030 000500 131127 G321846
9 01010E1030 000500 131125 G321846
10 01010E1040 000002 131129 G321846
11 01010E1040 000400 131129 G321846
12 01010E1040 000400 131128 G321846
確認
メッセージ
取り込みました
« SAGA ▸ DATA ▸
整理
開く
印刷
電子メールで送信する
KEYENCE
N201
NTC
NTC_201505
NTC_201505
PerfLogs
Program File
SAGA
DATA
RCVDATA
104S-190843205097-01-01.txt
(2ff)104S-190843205097-01-01.txt
@@104S-190843205097-01-01 - コピー.txt
_104S-190843205097-01-01 - コピー.txt
@@@104S-190843205097-01-01.txt
@104S-190843205097-01-01.txt
bk1_104S-190843205097-01-01.txt
_104S-190843205097-01-01.txt
注文データ20131118(104S-190843205097-01-01).txt
種類: テキスト ドキュメント
サイズ: 3.55 KB
104S-190843205097-01-01.txt 更新日時: 2014/01/23 11:50
テキスト ドキュメント
サイズ: 489 KB

図3　パラメータ管理マスター

```
A*****************************************************************
A*      DBR012 (ﾊﾝﾖｳﾊﾞｲﾀｲｺｳﾓｸﾊﾟﾗﾒｰﾀｶﾝﾘﾏｽﾀｰ)              *
A*****************************************************************
A                      UNIQUE
A       R DBR012R
A         DBR0120010    3A       COLHDG('会社C')
A         DBR0120020    1A       COLHDG('会社区分C')
A         DBR0120030    2A       COLHDG('パラメータ種別')
A         DBR0120040    2A       COLHDG('パターンＮＯ')
A         DBR0120050   12A        COLHDG('登録ID')
A         DBR0120060    2A       COLHDG('枝番')
A         DBR0120070    2A       COLHDG('レコード区分')
A         DBR0120080   22O        COLHDG('抽出項目名')
A         DBR0120090    6S       COLHDG('抽出開始桁位置')
A         DBR0120100    6S       COLHDG('抽出桁数')
A         DBR0120110    3S       COLHDG('抽出項目番号')
A         DBR0120120    6S       COLHDG('条件値1開始桁位置')
A         DBR0120130    6S       COLHDG('条件値1桁数')
A         DBR0120140   30O        COLHDG('条件値1')
A         DBR0120150    6S       COLHDG('条件値2開始桁位置')
A         DBR0120160    6S       COLHDG('条件値2桁数')
A         DBR0120170   30O        COLHDG('条件値2')
A         DBR0120180    6S       COLHDG('条件値3開始桁位置')
A         DBR0120190    6S       COLHDG('条件値3桁数')
A         DBR0120200   30O        COLHDG('条件値3')
A         DBR0120210    6S       COLHDG('条件値4開始桁位置')
A         DBR0120220    6S       COLHDG('条件値4桁数')
A         DBR0120230   30O        COLHDG('条件値4')
A         DBR0120240    6S       COLHDG('条件値5開始桁位置')
A         DBR0120250    6S       COLHDG('条件値5桁数')
A         DBR0120260   30O        COLHDG('条件値5')
A         DBR0120270   30O        COLHDG('登録者')
A         DBR0120280    4S       COLHDG('登録年')
A         DBR0120290    2S       COLHDG('登録月')
A         DBR0120300    2S       COLHDG('登録日')
A         DBR0120310    2S       COLHDG('登録時')
A         DBR0120320    2S       COLHDG('登録分')
A         DBR0120330    2S       COLHDG('登録秒')
A         DBR0120340   30O        COLHDG('更新者')
A         DBR0120350    4S       COLHDG('更新年')
A         DBR0120360    2S       COLHDG('更新月')
A         DBR0120370    2S       COLHDG('更新日')
A         DBR0120380    2S       COLHDG('更新時')
A         DBR0120390    2S       COLHDG('更新分')
A         DBR0120400    2S       COLHDG('更新秒')
A       K DBR0120010
A       K DBR0120020
A       K DBR0120030
A       K DBR0120040
A       K DBR0120050
A       K DBR0120060
```

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
代表取締役

株式会社ランドコンピュータ
http://www.landcomp.co.jp

高校、大学など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行っている。

業務課題

IBM i 以外にもサーバー類があるが、会社の休日・祝日も電源が入っている状態になっており、無駄になっている。また、無人環境での作動状態で、セキュリティ上も望ましくない。

技術課題

IBM i にはカレンダー機能があり、祝日や会社の休日を登録してある。このデータを利用して、他のサーバー類を WakeOnLAN で動作させることはできないか。

技術課題の解決策

ミガロ.テクニカルサポートよりアドバイスを受け、IBM i の次回起動日、起動時刻を取得できるようになった。そのデータを基に他のサーバー類の電源を管理することができた。Delphi の標準にはない Delphi/400 コンポーネントを使用した効果により、シンプルなコーディングを実現できた。【図 1】【図 2】

業務課題解決と効果

常時稼働サーバーから WakeOnLAN のパケットを投げるプログラムを作成。休日は、IBM i も含めサーバー類の電源が勝手に入ることはなくなり、電気代の節約とセキュリティの改善を実現した。

M

図1　システム構成図

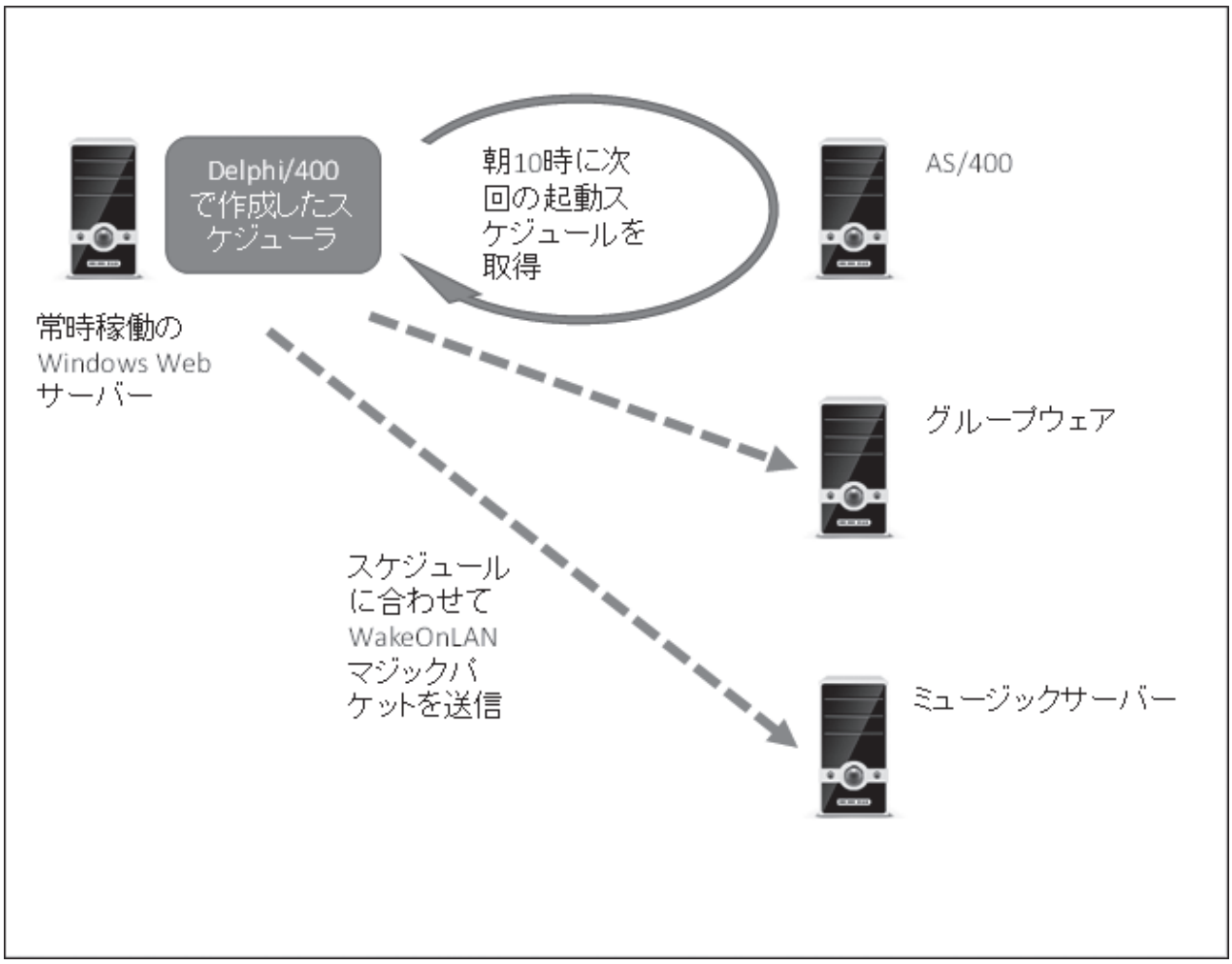


図2　ソースコードサンプル

```
Delphi/400のcmd400コンポーネントで下記のコマンドを実行する
(ミガロ.より受領したサンプルソースコード)

Cmd400.CommandLine.Text := 'RTVSYSVAL  SYSVAL(QIPLDATTIM) RTNVAR(&P.1)';
Cmd400.Execute;

Edit1.Text := Copy(Cmd400.Value[0], 0, 6);
Edit2.Text := Copy(Cmd400.Value[0],11, 6);
```

Migaro.
Technical Report
2015

ミガロ.SE 論文／ミガロ. テクニカルレポート

前坂 誠二

株式会社ミガロ.

システム事業部 システム2課

[Delphi/400] フレームを利用した開発手法

略歴
1989 年 3 月 21 日生まれ
2011 年 3 月 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。

1.はじめに

プログラムを開発する上で、開発効率・保守性の向上は非常に重要である。開発効率とは、いかに少ない労力で開発が完了するかを表した度合いであり、保守性とは既に作成されたプログラムに対して、維持・管理の容易さを表したものである。

これらを向上させるためには、共通化という考え方が非常に有効である。理由としては、処理を共通化することで、同一の処理を複数回記述する手間が不要となり、プログラムを修正する際には、共通処理の修正のみで作業が完了するからである。

Delphi/400 のプログラムにおいて、処理を共通化する方法には、大きく以下の 4 つの方法がある。

①共通関数ユニットによる処理の共通化
②コンポーネントによる部品の共通化
③継承によるクラスの共通化
④フレームによる画面一部の共通化

本稿では、これらの方法から④のフレームによる共通化の方法を題材としている。

「2.」ではフレームを使用することによるメリット、「3.」ではフレームを作成する手順、「4.」ではフレームの利用方法、「5.」では応用として IBM i 連携を行ったフレームの作成方法について紹介する。なお、本稿でのプログラム例は、Delphi/400 XE7 を使用している。

2.フレームについて

2-1. フレームとは

フレームとは、複数のコンポーネントを含めた画面の一部を 1 つにまとめるクラスのことである。作成したフレームは 1 つのコンポーネントとして流用して貼り付けることができる。

Delphi 言語の特徴には、ツールパレットやオブジェクトインスペクタを利用したビジュアル開発を行える点がある。しかし、開発の際、【図 1】のように、複数画面で部分的に同じ画面設計を作成しなければならない場合がある。そういった時に、今回紹介するフレームが非常に有効である。

2-2. フレーム利用によるメリット

フレームを利用すると、処理が共通化され、「1. はじめに」でも紹介したように、開発効率・保守性が向上する。

フレームでは、他の処理を共通化する方法と異なり、コンポーネントの配置についても共通化できるため、各画面で個別にコンポーネントを配置する必要がなくなり、開発工数の短縮につなげることができる。

3.フレームの作成手順

3-1. C/S アプリケーションでの作成手順

C/S アプリケーションでは、以下の 4 ステップによりフレームを作成できる。

1. ツールバーより、「ファイル」→「新規作成」→「その他」の順で選択する。【図 2】

図1

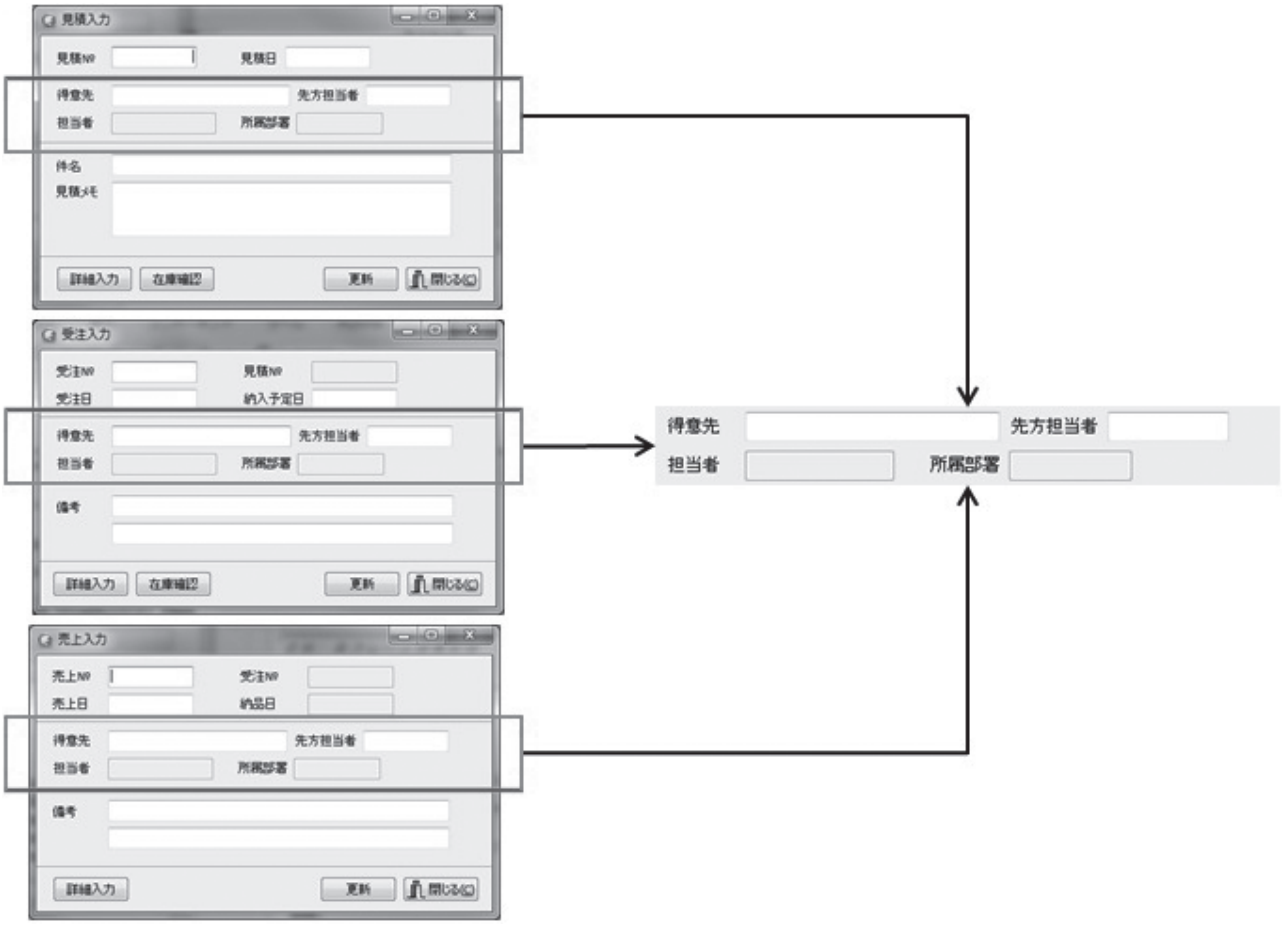
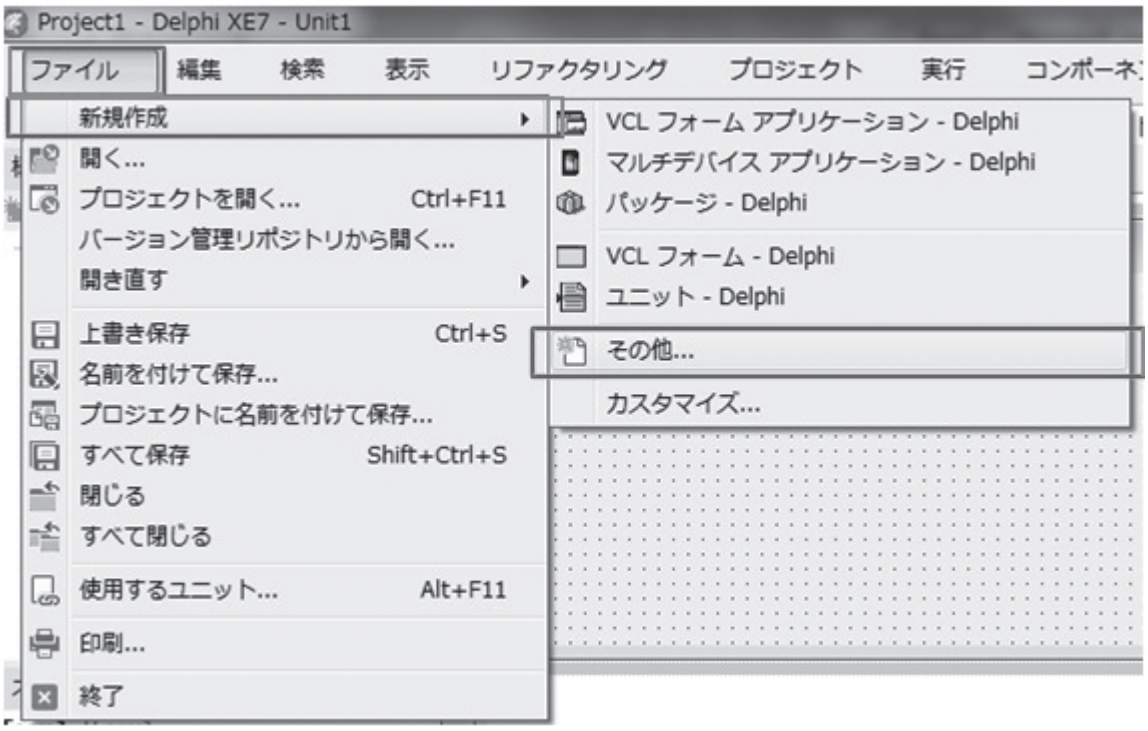

見積入力
見積№
見積日
得意先
先方担当者
担当者
所属部署
件名
見積メモ
詳細入力
在庫確認
更新
閉じる
受注入力
受注№
見積№
受注日
納入予定日
得意先
先方担当者
担当者
所属部署
備考
詳細入力
在庫確認
更新
閉じる
売上入力
売上№
受注№
売上日
納品日
得意先
先方担当者
担当者
所属部署
備考
詳細入力
更新
閉じる
得意先
先方担当者
担当者
所属部署

図2


Project1 - Delphi XE7 - Unit1
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
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 Ctrl+S
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 Shift+Ctrl+S
閉じる
すべて閉じる
使用するユニット... Alt+F11
印刷...
終了
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
ユニット - Delphi
その他...
カスタマイズ...

2. 新規作成のダイアログが起動するので、「Delphi プロジェクト」→「Delphi ファイル」より、VCL フレームを選択する。【図 3】
3. 配置したいコンポーネントを貼り付ける。【図 4】
4. 任意のファイル名を入力し、ファイルを保存する。【図 5】

3-2. Web アプリケーションでの作成手順

Delphi/400 の Web アプリケーション構築フレームワークである「IntraWeb」（VCL for the Web）においても、フレーム作成は可能である。また、C/S アプリケーションでの作成手順と同様に、以下のステップで簡単に作成できる。

1. ツールバーより「ファイル」→「新規作成」→「その他」の順で選択する（C/S アプリケーションと同様）。【図 2】
2. 新規作成のダイアログが起動するので、「Delphi プロジェクト」→「IntraWeb」より、NewFrame を選択する。【図 6】
3. 配置したいコンポーネントを貼り付ける。【図 7】
4. 任意のファイル名を入力し、ファイルを保存する（C/S アプリケーションと同様）。【図 5】

3-3. フレームの継承

フレームにおいても、フォームと同様に、継承による作成が可能である。今回は商品リストのフレームを生成し（fraShohin01）、そのフレームを継承して、単価、在庫数、数量、金額の項目を追加したフレーム（fraShohin02）を作成する。【図 8】

まずは、「3-1.」で紹介した手順で、fraShohin01 のフレームを作成する。その後、以下の手順で fraShohin02 のフレームを継承する。

1. ツールバーより「ファイル」→「新規作成」→「その他」の順で選択する。【図 2】
2. 新規作成のダイアログが起動するので、「Delphi プロジェクト」→「継承可能項目」より、fraShohin01 のフレームを選択する。【図 9】
3. 継承されたフレームに、追加で配置したいコンポーネントを貼り付ける。【図 10】
4. 任意のファイル名を入力し、ファイルを保存する。【図 5】

フレームの継承では、継承元フレーム、継承先フレームそれぞれを各画面で利用することが可能である。【図 11】

3-4. フレーム内でのプログラミングポイント

＜フレーム生成時・破棄時のイベント記述＞

フレームは、フォームと同じようにツールパレットからコンポーネントを選択し、配置することで設計を行うことができる。しかし、フレームのオブジェクトインスペクタを確認してみると、フレームはフォームとは違い、生成時の OnCreate イベントや破棄時の OnDestroy イベントが存在しない。【図 12】

そのため、フレームで生成時や破棄時の処理を記述したい場合は、上位クラスより Create、Destroy を継承して内部的にロジックを記述する必要がある。

ロジックの記述方法については、public 宣言にて、Create の場合は constructor、Destroy の場合は destructor と定義する。この際、上位クラスより処理を継承するため、override と記述する点に注意する。

public 宣言にて Create および Destroy の定義を行った後、「Ctrl + Shift + C」キーを押下すると、処理の記述部が補完されるため、フレーム生成時の処理および破棄時の処理を記述する。【ソース 1】

なお、フレーム破棄時の処理を記述する場合は、inherited の前に処理を記述する点に注意する。理由としては、inherited 処理にて、フレーム自体のメモリが解放されるため、記述した処理が正しく動作しない可能性があるためである。

＜プロパティ定義の利用＞

フレームを作成する際、フレーム利用画面との値の受け渡しのために、プロパティ定義を利用することをお勧めする。これにより、フレーム利用画面では、フレームに配置しているコンポーネントを直接指定しなくても値の受け渡しが可能となる。コンポーネントを直接指定しない利点としては、フレーム側のコンポーネントの変更などがあった場合でも、フレーム利用画面側のロジックを修正する必要がなくなる。また、プロパティ定義の read や write に項目ごとのメソッドを記述することで、コンポーネントの Enable 制御や色の変更などを簡単に実装することが可能となる。【ソース 2】

4.フレームの利用方法

4-1. ツールパレットへの追加

フレームを、ツールパレットに追加するには、まずフォームデザイナで作成したフレームを開く。次に右クリックでポップアップメニューを開き、「パレットに追加」を選択する。【図 13】

「パレットに追加」を選択すると、【図 14】のようなダイアログが開く。

ツールパレットで表示させたいコンポーネント名、パレットページ名、設定したいアイコンを選択する。すると、【図 15】のようにツールパレットに作成したフレームが追加される。

4-2. アプリケーションへの追加

作成したフレームをアプリケーションに追加する方法は 2 つある。

①ツールパレットで追加したフレームを選択し、対象画面のフォームデザイナに貼り付ける。【図 16】
②ツールパレットより、「Standard/Frames」を選択し、プロジェクトに登録されているフレームの一覧から、対象のフレームを選択する。【図 17】

4-3. フレームの動的生成

フレームはフォームデザイナ上で貼り付けて使用するだけでなく、ロジックで動的に生成することも可能である。【図 18】

まず、ソースの Uses に生成させたいフレームを定義する（本稿の場合は、ShohinFra02 を定義）。そして、動的生成の方法は【ソース 3】のように、対象のフレームを Create した後、フレーム名や配置などのフレームのプロパティを設定する。フレーム名については、同一の名前で生成した場合、エラーとなるため注意が必要である。また、生成したフレームを破棄する場合は、FreeAndNil（対象フレーム）により破棄することが

図3

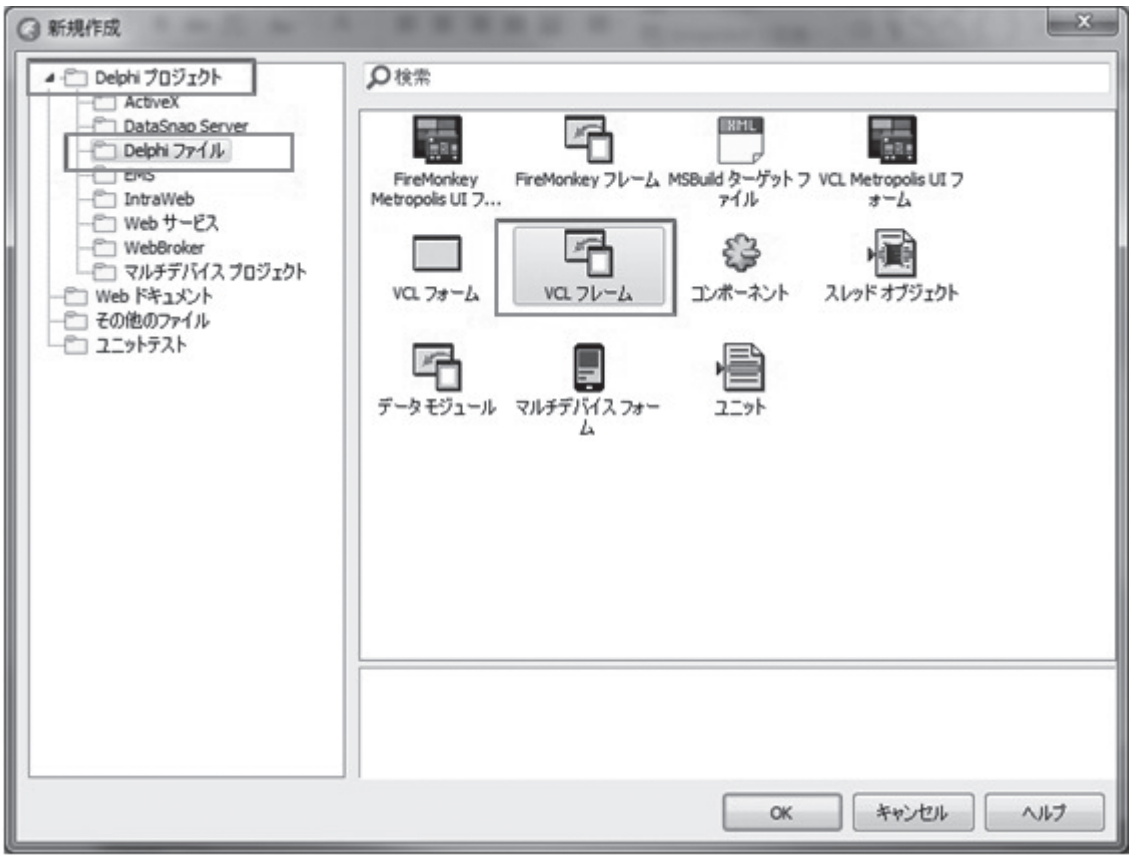
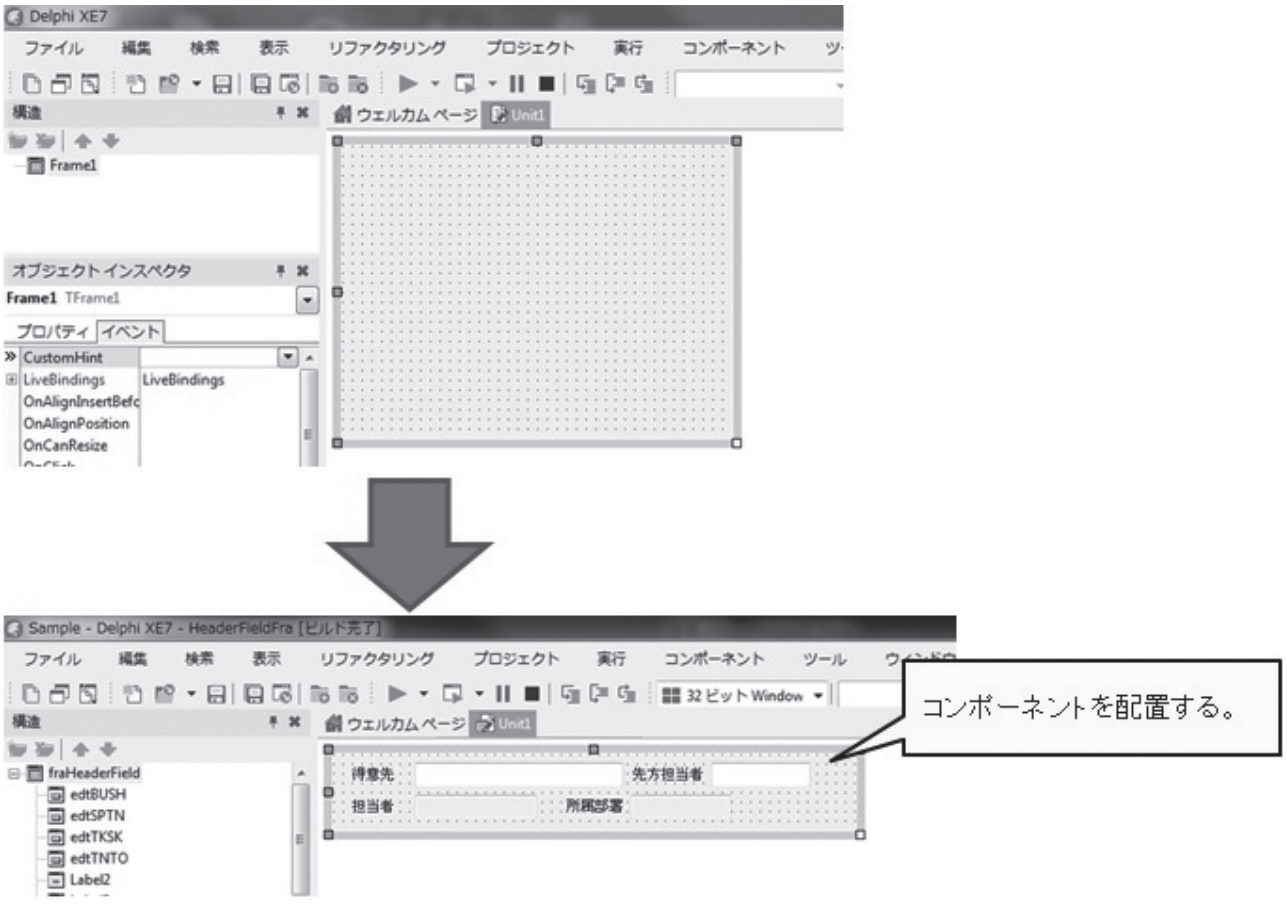

新規作成
Delphi プロジェクト
ActiveX
DataSnap Server
Delphi ファイル
IntraWeb
Web サービス
WebBroker
マルチデバイス プロジェクト
Web ドキュメント
その他のファイル
ユニットテスト
検索
FireMonkey Metropolis UI フ...
FireMonkey フレーム
MSBuild ターゲット ファイル
VCL Metropolis UI フォーム
VCL フォーム
VCL フレーム
コンポーネント
スレッド オブジェクト
データ モジュール
マルチデバイス フォーム
ユニット
OK
キャンセル
ヘルプ

図4


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
構造
ウェルカム ページ
Unit1
Frame1
オブジェクト インスペクタ
Frame1 TFrame1
プロパティ イベント
CustomHint
LiveBindings
OnAlignInsertBefo
OnAlignPosition
OnCanResize
Sample - Delphi XE7 - HeaderFieldFra [ビルド完了]
32 ビット Window
fraHeaderField
edtBUSH
edtSPTN
edtTKSK
edtTNTO
Label2
得意先
先方担当者
担当者
所属部署
コンポーネントを配置する。

できる。

4-4. フレーム利用画面でのプログラミングポイント

＜フレーム項目値の取得について＞

フレーム利用画面で、フレーム項目の値を取得する2つの方法を紹介する。【ソース4】

①フレーム内コンポーネントを直接指定して値を取得する方法

フレーム名を指定し、そのあとにフレーム内で配置しているコンポーネント名を指定する。

②フレームのプロパティ定義名を指定して値を取得する方法

フレーム内で項目ごとにプロパティ定義を行い、プロパティのwriteメソッドで、画面値のセット処理を行っている場合は、フレーム名＋プロパティ定義名を指定する。

①②どちらの方法でも値の取得自体は可能であるのだが、3-4.で紹介した通り、作成後にコンポーネントの変更があった場合やプログラムのメンテナンスのしやすさを考慮すると、②の方法を使用した方がよい。

＜フレーム利用画面で可能なフレーム項目の変更＞

フレーム利用画面側で、フレームに配置している項目の削除は禁止されている。コンポーネントを選択し、Deleteボタンなどで削除しようとすると、【図19】のようにエラーが表示される。

ただし、以下の内容については、フレーム利用画面側で個別に設定・変更が可能である。

①フレームで配置しているコンポーネントのサイズ変更や位置の移動
②フレームで配置しているコンポーネントのプロパティ値の変更

フレーム利用画面側で変更した内容については、【図20】のようにフレーム自体には反映されないため、画面ごとに個別に設定を変更することも可能である。よって、もし特定の項目を使用したくない場合は、項目のVisibleプロパティをFalseに設定し、非表示にするとよい。

＜フレーム配置項目の処理追加＞

【ソース5】のように、フレーム内に既に処理を組み込んでいる項目に対して、フレーム利用画面側でさらに処理を追加したい場合、【ソース6】のように記述する。フレーム利用画面で同一項目のイベントをダブルクリックすると、既にフレーム側で処理が記述されている場合、「フレーム名.同一項目のイベント(Sender)」といった内容が自動で生成される。フレーム内のイベントより前に処理を追加したい場合、この記述の前に処理を記述するとよい。

5.フレーム内でのIBM i処理

5-1. 使用するコンポーネントとフレーム

本章では、フレームの応用例として、C/SアプリケーションでのIBM iへの接続を行ったフレームの作成および利用方法について紹介する。フレームは「3.」で作成したfraShohin01およびfraShohin02のフレームを使用する。

fraShohin01のフレームでは、TDBLookUpComboBoxを使用し、マスタの内容をリスト形式にて表示する。【図21】

今回の例では、商品マスタを参照する。【図22】

5-2. フレームからのIBMi接続

フレームからIBM i接続を行う方法は、フォーム画面からIBM i接続を行う場合と同様の手順で実装できる。

今回はdbExpress接続を使用し、IBM iとの接続処理を行う。dbExpressのコンポーネントはフレームの中でも定義できるが、フレームごとに新しい接続を定義するとアプリケーション全体での接続が複数になってしまうため、接続を処理するTSQLConnectionは共通のデータモジュールなどを参照するように設計した方がよい。

まず、データモジュールを作成し(dmMain)、TSQLConnectionのコンポーネントを配置する。

このTSQLConnectionのコンポーネントのConnectionNameプロパティには、IBM iに接続するためのCO400 Connectionを設定し、接続パラメータを指定しておく。

fraShohin01のフレームでは、TSQLQuery、TDataSetProvider、TClientDataSet、TDataSourceを配置し、Usesには先ほど作成したデータモジュール（dmMain）を追加する。各IBM iとの接続コンポーネントの設定については、【図23】に示す。

また、TDBLookUpComboBoxのプロパティ設定は【図24】の通りとする。

5-3. フレーム内/利用時のプログラミングポイント

＜フレーム内のプログラミング＞

fraShohin01のフレームでは、商品マスタを参照し、リスト形式で表示する。今回、リスト内容のセットは、SetListItemという名前のサブルーチンにて行う。

リスト内容のセット処理は、フレーム利用画面で、呼び出しが行えるように、public宣言にて記述する。本稿では、データの取得をSQLで行うため、まずはSQL文の記述を行い、その後にClientDataSetのOpen処理を実行して、リスト内容の取得を行う。また、その際にBlankAddのプロパティがTrueで渡された場合は、先頭行をブランク行とするように処理を記述する。【ソース7】【ソース8】

fraShohin01を継承したfraShohin02のフレームでは、商品のリストを選択した時に単価・在庫数をセットするロジックを記述する。【ソース9】

＜フレーム利用画面でのプログラミング＞

フレーム利用画面では、「4.」で紹介したフレームを動的生成する処理を利用する。【ソース3】のロジックに、フレームで定義しているSetListItemの呼出し処理を追加し、リスト選択が可能な詳細入力画面を起動する。【ソース10】

6.まとめ

本稿では、開発効率および保守性を向上させるための手法の一つであるフレームについて紹介した。フレームはフォームと同様にフォームデザイナにて、ツールパレットからコンポーネントを貼り付

図5

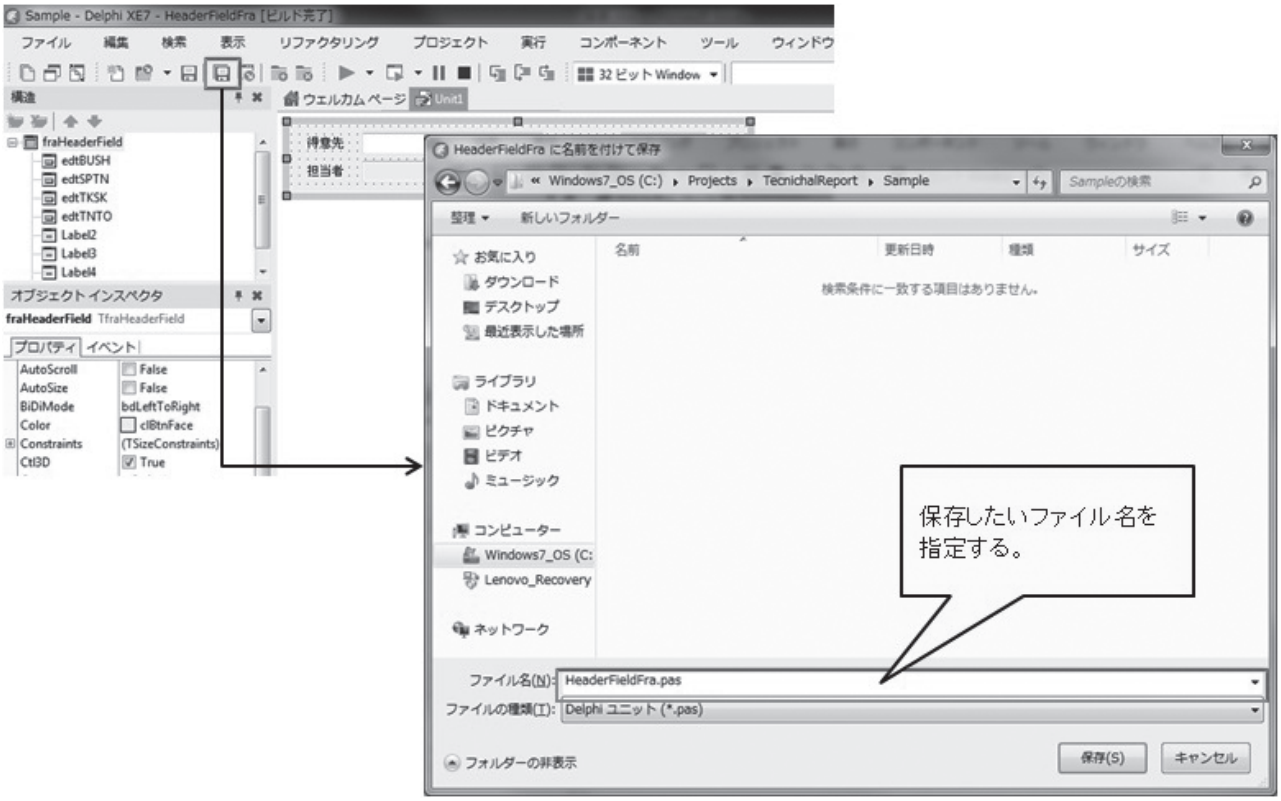
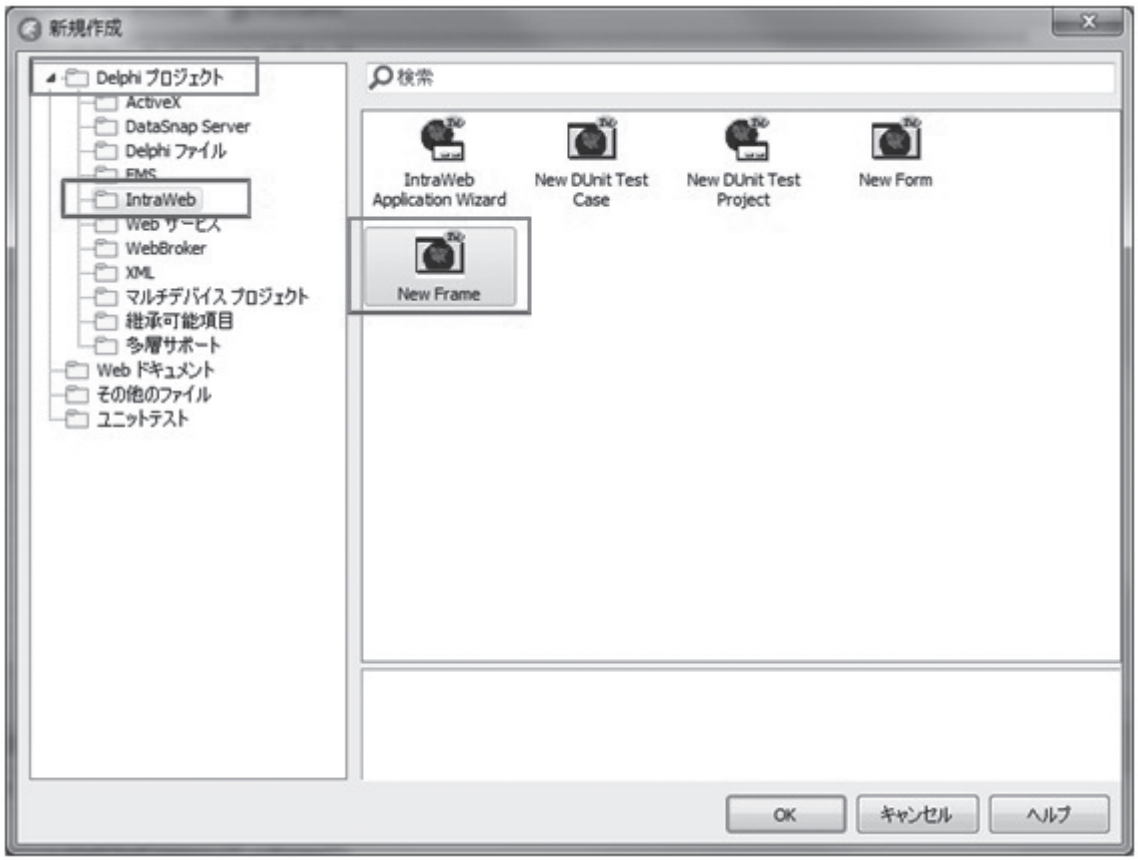

Sample - Delphi XE7 - HeaderFieldFra [ビルド完了]
ファイル 編集 検索 表示 リファクタリング プロジェクト 実行 コンポーネント ツール ウィンドウ
構造
fraHeaderField
edtBUSH
edtSPTN
edtTKSK
edtTNTO
Label2
Label3
Label4
オブジェクト インスペクタ
fraHeaderField TfraHeaderField
プロパティ イベント
AutoScroll False
AutoSize False
BiDiMode bdLeftToRight
Color clBtnFace
Constraints (TSizeConstraints)
Ctl3D True
ウェルカム ページ
Unit1
得意先
担当者
HeaderFieldFra に名前を付けて保存
Windows7_OS (C:) ▸ Projects ▸ TecnichalReport ▸ Sample
Sampleの検索
整理
新しいフォルダー
お気に入り
ダウンロード
デスクトップ
最近表示した場所
ライブラリ
ドキュメント
ピクチャ
ビデオ
ミュージック
コンピューター
Windows7_OS (C:
Lenovo_Recovery
ネットワーク
名前
更新日時
種類
サイズ
検索条件に一致する項目はありません。
保存したいファイル名を
指定する。
ファイル名(N): HeaderFieldFra.pas
ファイルの種類(T): Delphi ユニット (*.pas)
フォルダーの非表示
保存(S)
キャンセル

図6


新規作成
Delphi プロジェクト
ActiveX
DataSnap Server
Delphi ファイル
FMS
IntraWeb
Web サービス
WebBroker
XML
マルチデバイス プロジェクト
継承可能項目
多層サポート
Web ドキュメント
その他のファイル
ユニットテスト
検索
IntraWeb Application Wizard
New DUnit Test Case
New DUnit Test Project
New Form
New Frame
OK
キャンセル
ヘルプ

けるだけで設計が可能であるため、作成方法については通常のフォーム開発と同じである。

また、フレームを使用する際は、作成したフレームをフォームに貼り付けるだけでコンポーネント同様に使用できるため、誰でも容易にフレームを利用した共通化が行える。

「1.」で説明した通り、Delphi/400では処理を共通化する手法がいくつか用意されている。

コンポーネントや継承といった方法は、一般的によく使用されているが、今回紹介したフレームは意外と使われていない（知られていない）ことも多い。

しかしこのフレームを使った開発は、コンポーネント作成より簡単で、さらに画面の一部分だけを共通化できるため、画面単位の継承よりも流用性が高い。この機能を知った上で開発を行えば、必ず開発効率、保守性に役立つはずである。

M

図7

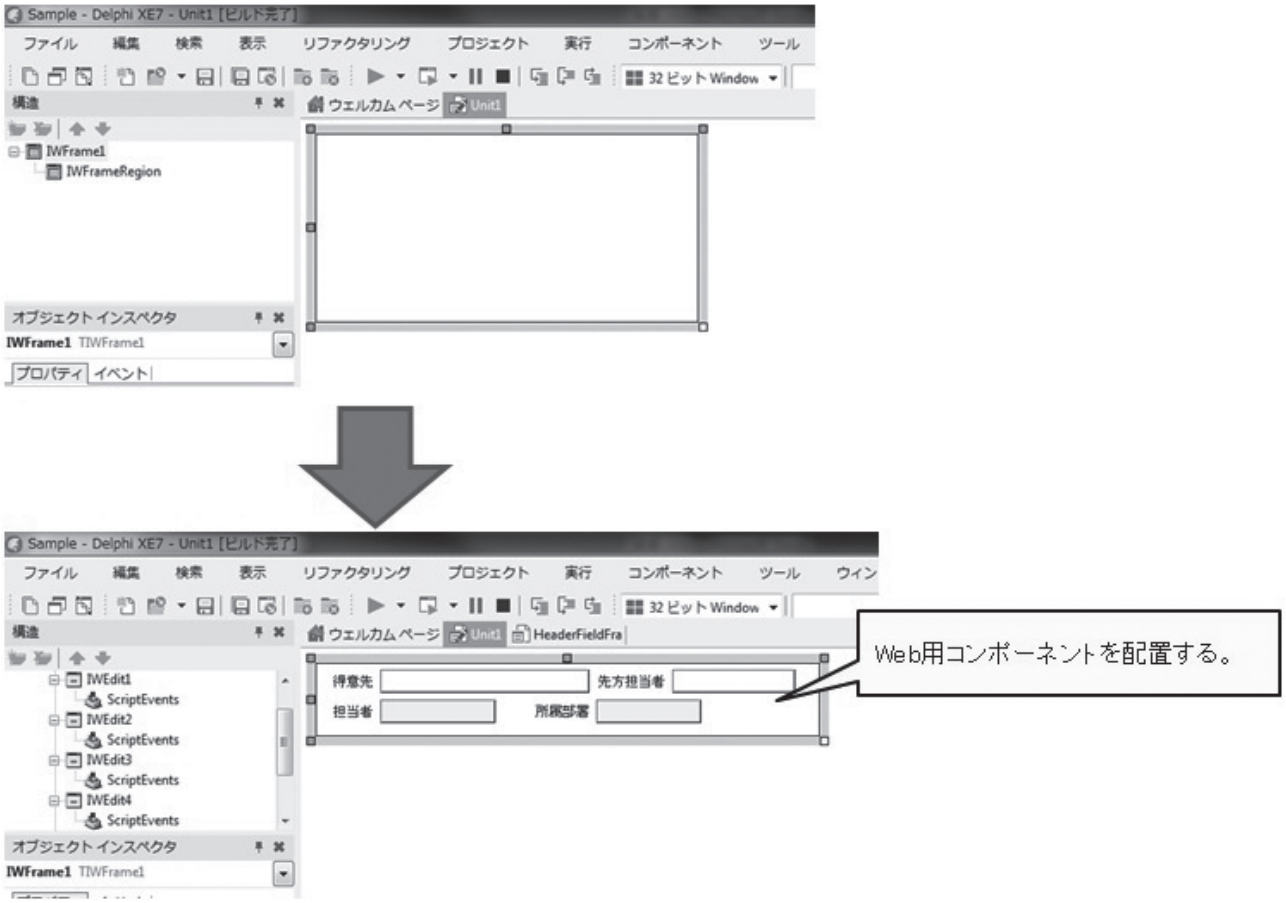

Sample - Delphi XE7 - Unit1 [ビルド完了]
ファイル 編集 検索 表示 リファクタリング プロジェクト 実行 コンポーネント ツール
32 ビット Window
構造
ウェルカムページ Unit1
IWFrame1
IWFrameRegion
オブジェクト インスペクタ
IWFrame1 TIWFrame1
プロパティ イベント
Sample - Delphi XE7 - Unit1 [ビルド完了]
ファイル 編集 検索 表示 リファクタリング プロジェクト 実行 コンポーネント ツール ウィン
32 ビット Window
構造
ウェルカムページ Unit1 HeaderFieldFra
IWEdit1
ScriptEvents
IWEdit2
ScriptEvents
IWEdit3
ScriptEvents
IWEdit4
ScriptEvents
得意先
先方担当者
担当者
所属部署
オブジェクト インスペクタ
IWFrame1 TIWFrame1
Web用コンポーネントを配置する。

図8

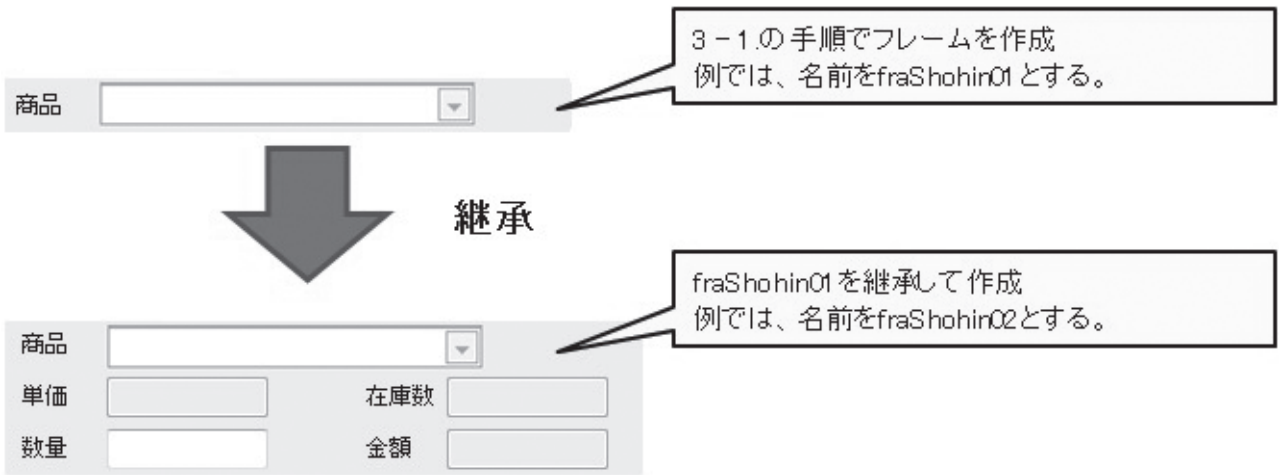

商品
3－1の手順でフレームを作成
例では、名前をfraShohin01とする。
継承
商品
単価
在庫数
数量
金額
fraShohin01を継承して作成
例では、名前をfraShohin02とする。

図9

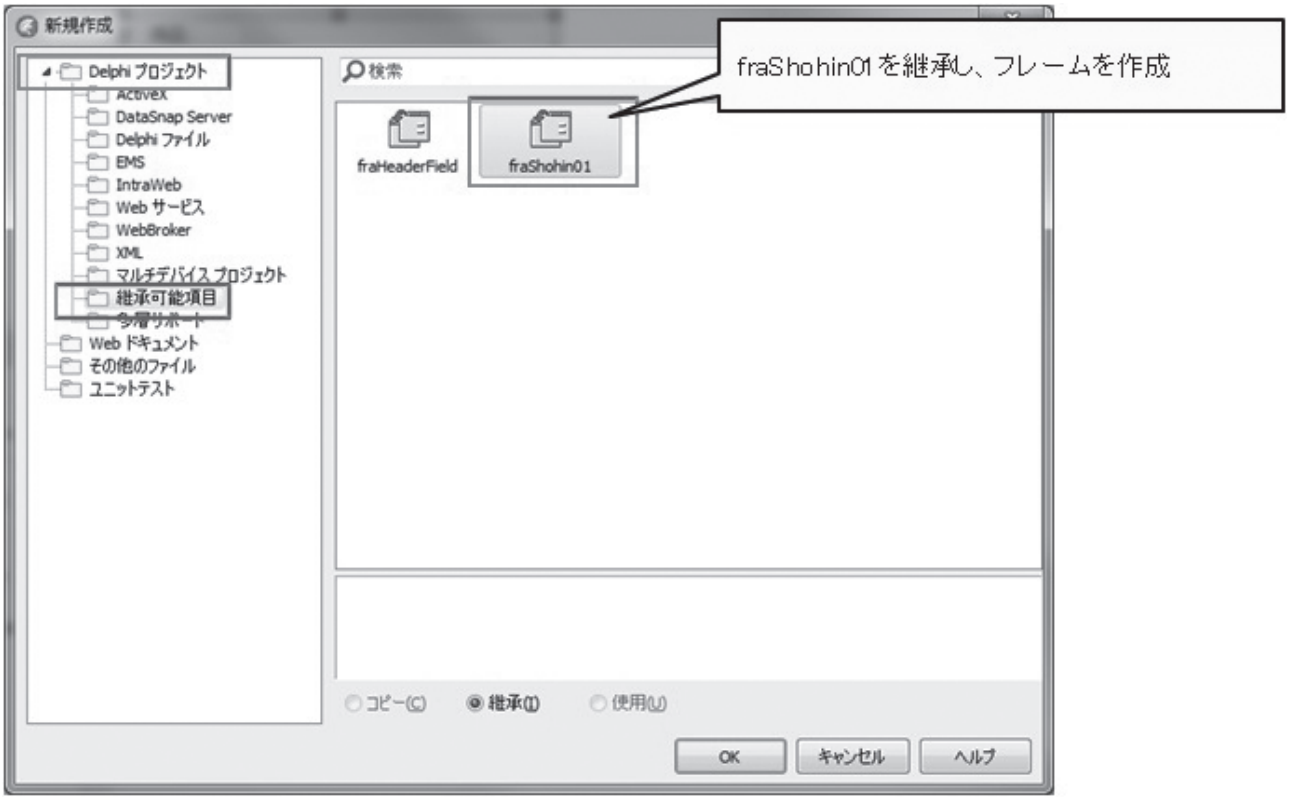

新規作成
Delphi プロジェクト
ActiveX
DataSnap Server
Delphi ファイル
EMS
IntraWeb
Web サービス
WebBroker
XML
マルチデバイス プロジェクト
継承可能項目
多層サポート
Web ドキュメント
その他のファイル
ユニットテスト
検索
fraHeaderField
fraShohin01
fraShohin01を継承し、フレームを作成
コピー(C)
継承(I)
使用(U)
OK
キャンセル
ヘルプ

図10

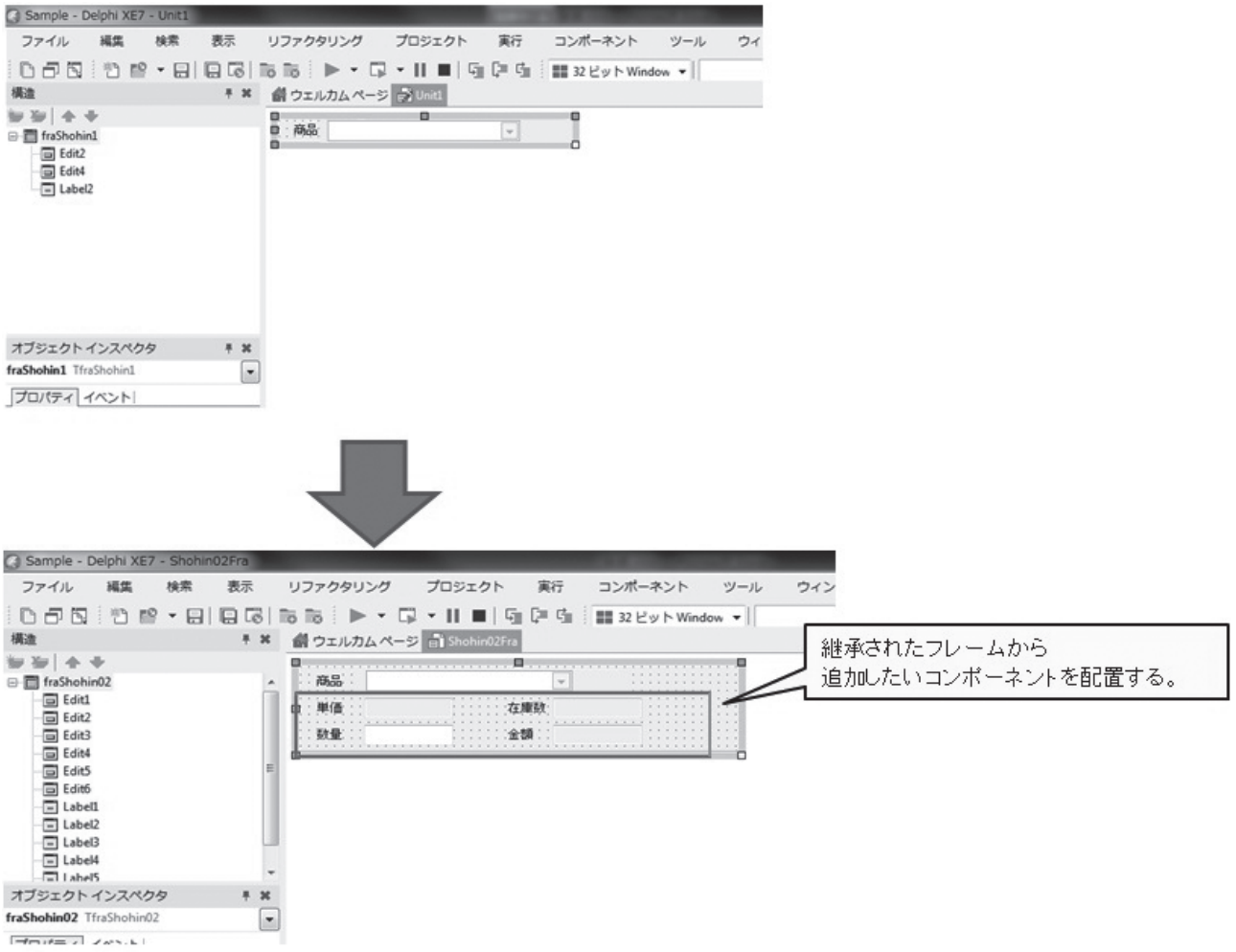

Sample - Delphi XE7 - Unit1
ファイル 編集 検索 表示 リファクタリング プロジェクト 実行 コンポーネント ツール
32 ビット Window
構造
ウェルカム ページ
Unit1
fraShohin1
Edit2
Edit4
Label2
商品
オブジェクト インスペクタ
fraShohin1 TfraShohin1
プロパティ イベント
Sample - Delphi XE7 - Shohin02Fra
ファイル 編集 検索 表示 リファクタリング プロジェクト 実行 コンポーネント ツール
32 ビット Window
構造
ウェルカム ページ
Shohin02Fra
fraShohin02
Edit1
Edit2
Edit3
Edit4
Edit5
Edit6
Label1
Label2
Label3
Label4
Label5
商品
単価
在庫数
数量
金額
オブジェクト インスペクタ
fraShohin02 TfraShohin02
継承されたフレームから
追加したいコンポーネントを配置する。

図11

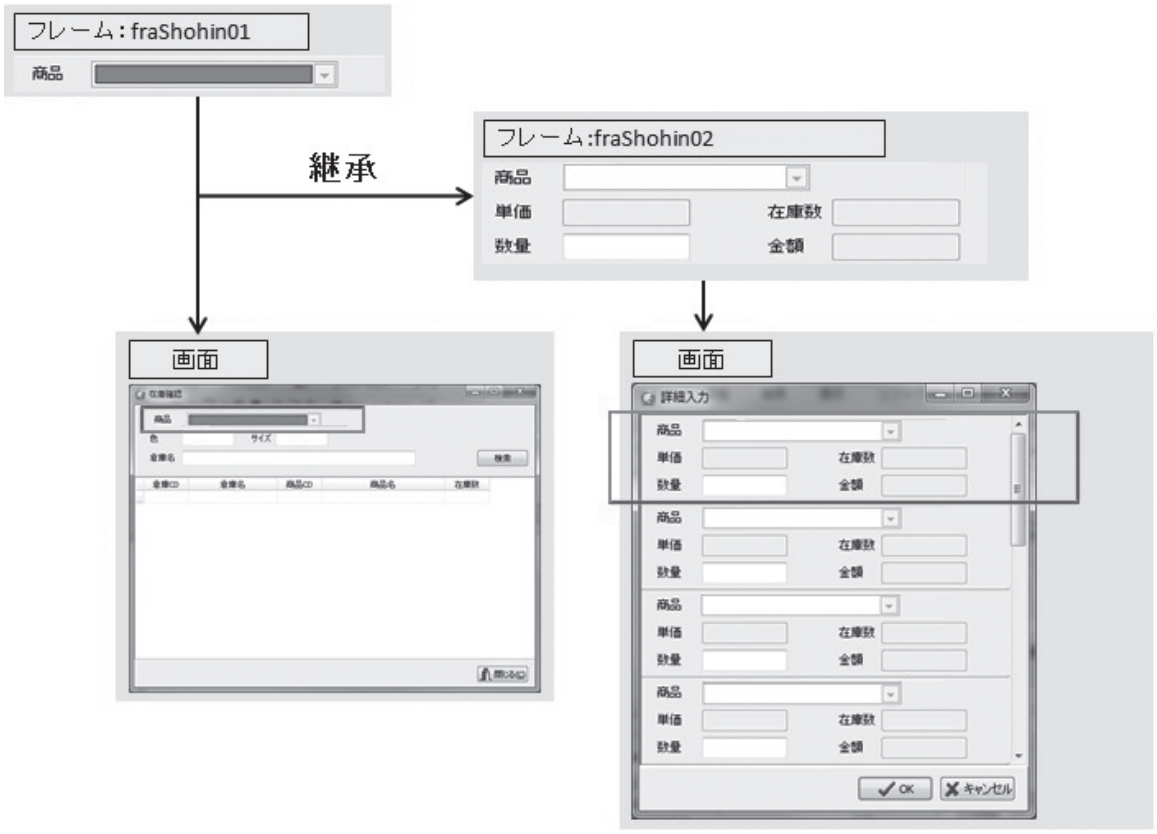
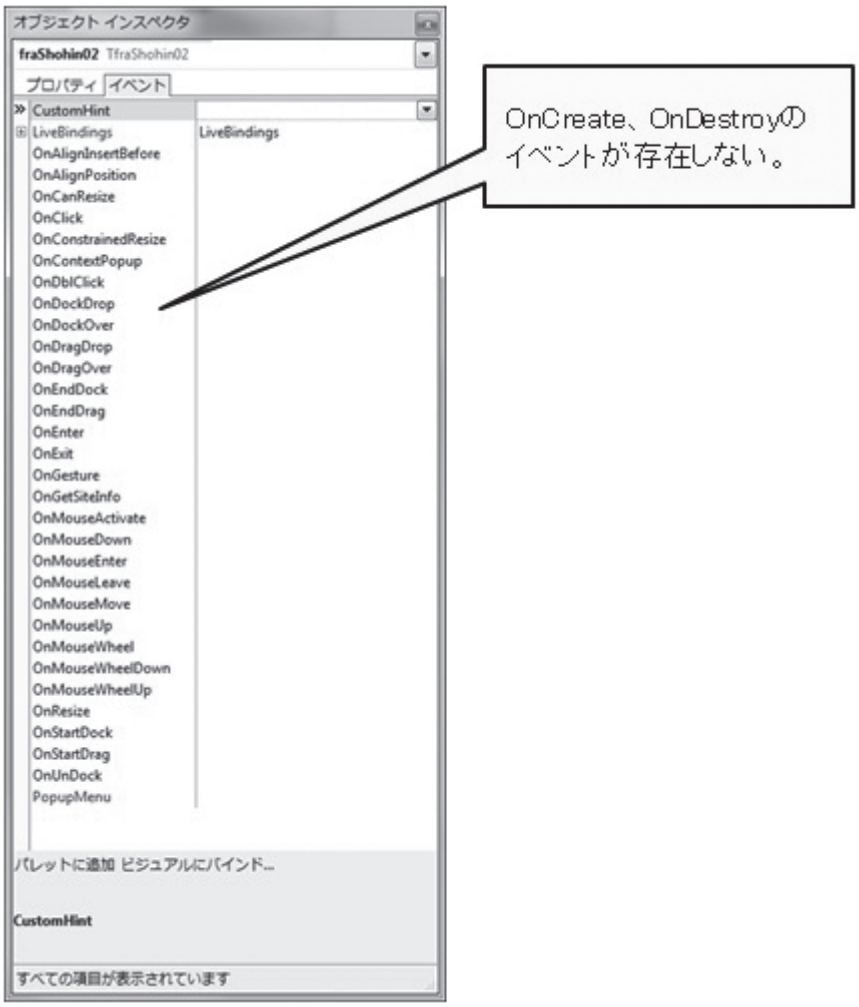

フレーム：fraShohin01
商品
継承
フレーム:fraShohin02
商品
単価
在庫数
数量
金額
画面
画面
詳細入力
商品
単価
在庫数
数量
金額
OK
キャンセル

図12


オブジェクト インスペクタ
fraShohin02 TfraShohin02
プロパティ
イベント
CustomHint
LiveBindings
OnAlignInsertBefore
OnAlignPosition
OnCanResize
OnClick
OnConstrainedResize
OnContextPopup
OnDblClick
OnDockDrop
OnDockOver
OnDragDrop
OnDragOver
OnEndDock
OnEndDrag
OnEnter
OnExit
OnGesture
OnGetSiteInfo
OnMouseActivate
OnMouseDown
OnMouseEnter
OnMouseLeave
OnMouseMove
OnMouseUp
OnMouseWheel
OnMouseWheelDown
OnMouseWheelUp
OnResize
OnStartDock
OnStartDrag
OnUnDock
PopupMenu
パレットに追加 ビジュアルにバインド...
CustomHint
すべての項目が表示されています
OnCreate、OnDestroyの
イベントが存在しない。

ソース1

```
  private
    { Private 宣言 }
  public
    { Public 宣言 }
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  end;

implementation

{$R *.dfm}

{ TfraShohin01 }

constructor TfraShohin01.Create(AOwner: TComponent);
begin
  inherited;

  // フレーム生成時の処理を記述

end;

destructor TfraShohin01.Destroy;
begin
  // フレーム解放時の処理を記述

  inherited;
end;

end.
```

publicにて、Create、Destroyを宣言

フレーム生成時の処理を記述

フレーム破棄時の処理を記述
inheritedにて、フレーム自体のメモリを解放

ソース2

```
  private
    { Private 宣言 }
    FTNKA: Currency;
    FKING: Currency;
    FZAIK: Currency;
    FSURY: Currency;
    procedure SetKING(const Value: Currency);
    procedure SetSURY(const Value: Currency);
    procedure SetTNKA(const Value: Currency);
    procedure SetZAIK(const Value: Currency);
    function GetKING: Currency;
    function GetSURY: Currency;
    function GetTNKA: Currency;
    function GetZAIK: Currency;
  public
    { Public 宣言 }
    property TNKA: Currency read GetTNKA write SetTNKA;     // 単価
    property ZAIK: Currency read GetZAIK write SetZAIK;     // 在庫数
    property SURY: Currency read GetSURY write SetSURY;     // 数量
    property KING: Currency read GetKING write SetKING;     // 金額

    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
  end;
```

画面の配置項目をプロパティ定義する

プロパティへ値を渡す(例:金額)

```
{*********************************************************************************
 目的：金額取得処理
 引数：
 戻値：
*********************************************************************************}
function TfraShohin02.GetKING: Currency;
var
  sTemp: string;
begin
  sTemp  := StringReplace(edtKING.Text, ',', '', [rfReplaceAll]);
  Result := StrToCurrDef(sTemp, 0);
end;
```

KINGプロパティの値を取得するタイミングで処理が走る。

画面値のカンマ区切りを外しプロパティに値をセット

プロパティより値を受け取る(例:金額)

```
{*********************************************************************************
 目的：金額セット時処理
 引数：
 戻値：
*********************************************************************************}
procedure TfraShohin02.SetKING(const Value: Currency);
begin
  FKING := Value;

  edtKING.Text := FormatFloat('#,0', FKING);

  if FKING < 0 then
    edtKING.Font.Color := clRed;
end;
```

KINGプロパティに値がセットされたタイミングで処理が走る。

フォーマットを編集し、画面のEditにセット
値が0より小さい場合、文字を赤色に設定

図13


得意先
先方担当者
担当者
所属部署
パレットに追加(P)
編集(W)
コントロール(X)
ビジュアルにバインド(Y)...

図14


コンポーネント テンプレートの情報
コンポーネント名(C) TfraHeaderFieldTemplate
パレット ページ名(P) TechnicalReport
パレット アイコン：
変更(A)...
OK
キャンセル
ヘルプ
コンポーネント名・パレットページ名・
アイコンを任意で指定

図15


ツール パレット
検索
Tethering
IntraWeb Standard
IntraWeb Data
IntraWeb Control
IntraWeb Authentication
Cloud
AFControls
AFDBControls
AFFrame
SCD400IFS
Screen5250
SCD400 Data
SCD400 System
SCD400 DB
SCD400 Tools
TechnicalReport
TfraHeaderFieldTemplate
図14にて設定した情報で追加される

図16

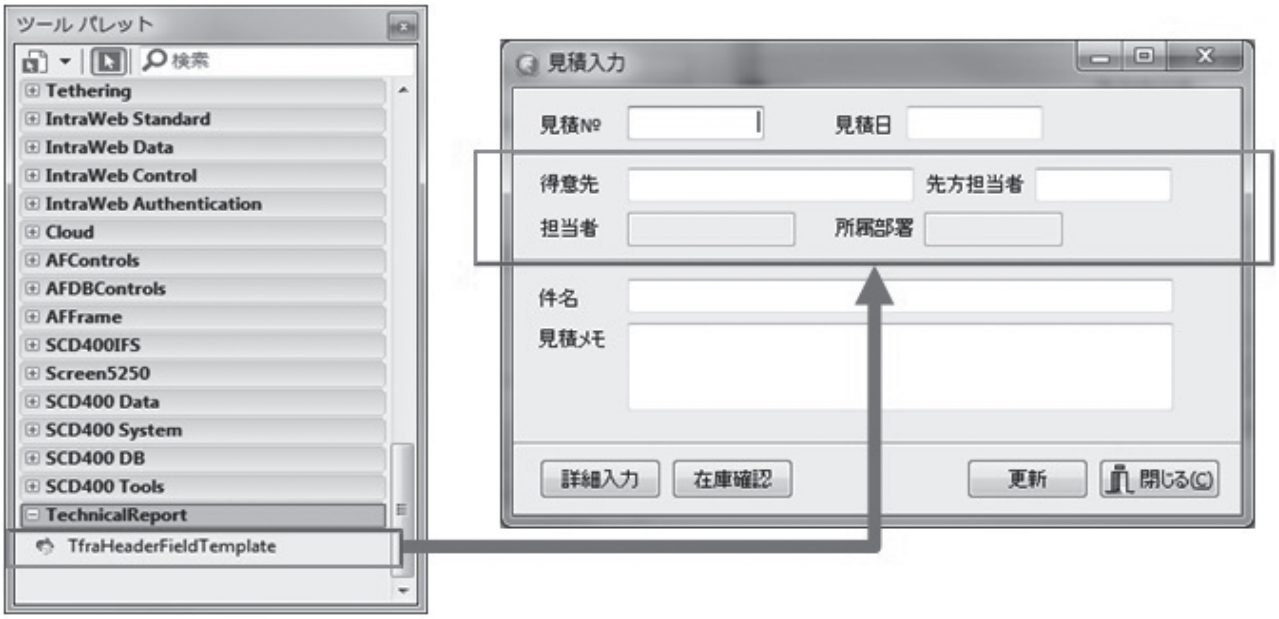

ツール パレット
検索
Tethering
IntraWeb Standard
IntraWeb Data
IntraWeb Control
IntraWeb Authentication
Cloud
AFControls
AFDBControls
AFFrame
SCD400IFS
Screen5250
SCD400 Data
SCD400 System
SCD400 DB
SCD400 Tools
TechnicalReport
TfraHeaderFieldTemplate
見積入力
見積№
見積日
得意先
先方担当者
担当者
所属部署
件名
見積メモ
詳細入力
在庫確認
更新
閉じる(C)

図17

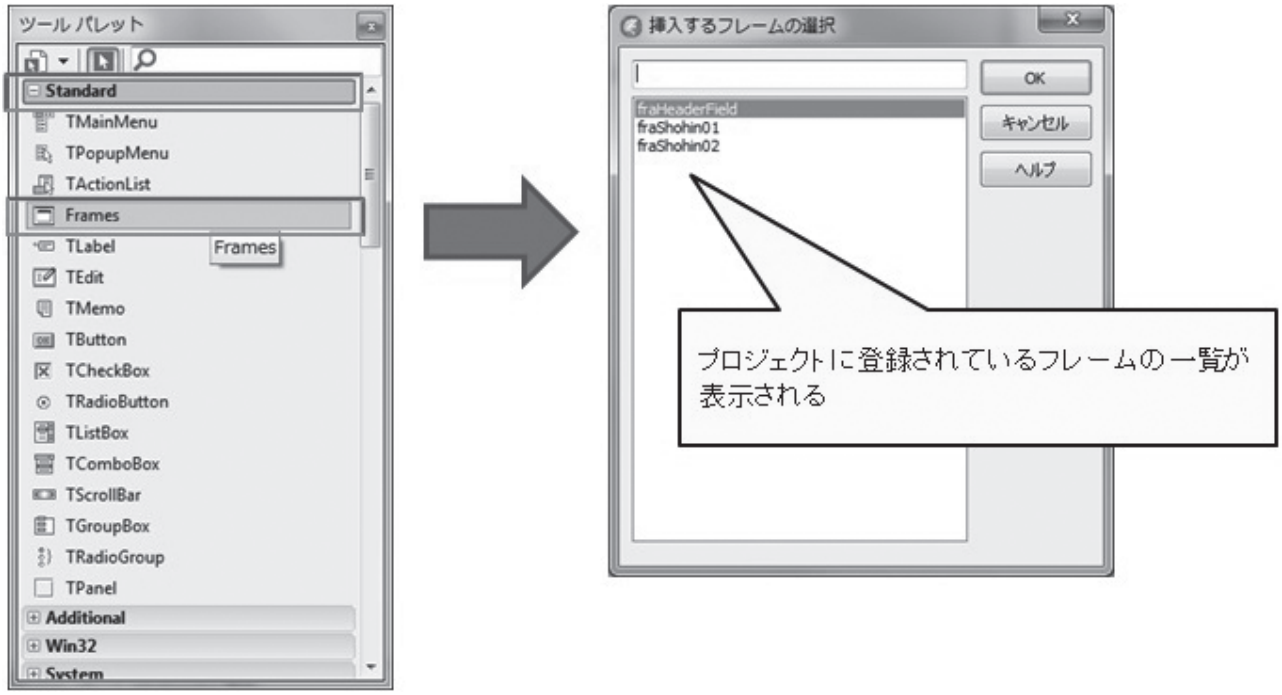

ツール パレット
Standard
TMainMenu
TPopupMenu
TActionList
Frames
TLabel
TEdit
TMemo
TButton
TCheckBox
TRadioButton
TListBox
TComboBox
TScrollBar
TGroupBox
TRadioGroup
TPanel
Additional
Win32
System
Frames
挿入するフレームの選択
fraHeaderField
fraShohin01
fraShohin02
OK
キャンセル
ヘルプ
プロジェクトに登録されているフレームの一覧が表示される

図18

ソース3

```
{*******************************************************************************
 目的：画面表示時処理
 引数：
 戻値：
*******************************************************************************}
procedure TfrmDetail.FormShow(Sender: TObject);
var
  i: Integer;
  fraShohin02: TfraShohin02;
  wCtl: TControl;
begin

  // フレーム初期化
  for i:= ScrollBox1.ControlCount-1 downto 0 do
  begin
    wCtl := ScrollBox1.Controls[i];

    if (wCtl is TfraShohin02) then
    begin
      FreeAndNil(wCtl);
    end;
  end;

  // フレーム生成
  for i := 0 to 10 do
  begin
    fraShohin02          := TfraShohin02.Create(Self);     // フレーム生成
    fraShohin02.Name     := 'fra' + FormatFloat('00', i);  // フレーム名
    fraShohin02.TabOrder := i - 1;                         // タブ順
    fraShohin02.Parent   := ScrollBox1;                    // 親コンポーネント
    fraShohin02.Top      := Height * (i);                  // フレームの位置設定
    fraShohin02.Align    := alTop;                         // フレームを上から整列

    fraShohin02.BlankAdd := True;                    // リストにブランク行追加
    fraShohin02.SetListItem;                         // リスト内容の設定

  end;

end;
```

ScrollBox上で生成されている
フレーム(TfraShohin02)の破棄

ScrollBox上にフレーム
(TfraShohin02)を生成

ソース4

更新ボタンの
OnClickイベント

①フレームに配置しているコンポーネントを
直接指定する場合

```
{*************************************************************************************
 目的：更新ボタン押下時処理
 引数：
 戻値：
*************************************************************************************}
procedure TfrmEstimates.bbtnUpdateClick(Sender: TObject);
begin

  SQLQuery1.ParamByName('TKSK').AsString := fraHeaderField1.edtTKSK.Text;   // 得意先
  SQLQuery1.ParamByName('SPTN').AsString := fraHeaderField1.edtSPTN.Text;   // 先方担当者
  SQLQuery1.ParamByName('TNTO').AsString := fraHeaderField1.edtTNTO.Text;   // 担当者
  SQLQuery1.ParamByName('BUSH').AsString := fraHeaderField1.edtBUSH.Text;   // 所属部署

  SQLQuery1.ExecSQL;

end;
```

フレーム名＋コンポーネント名

②フレームのプロパティ定義名を
指定する場合

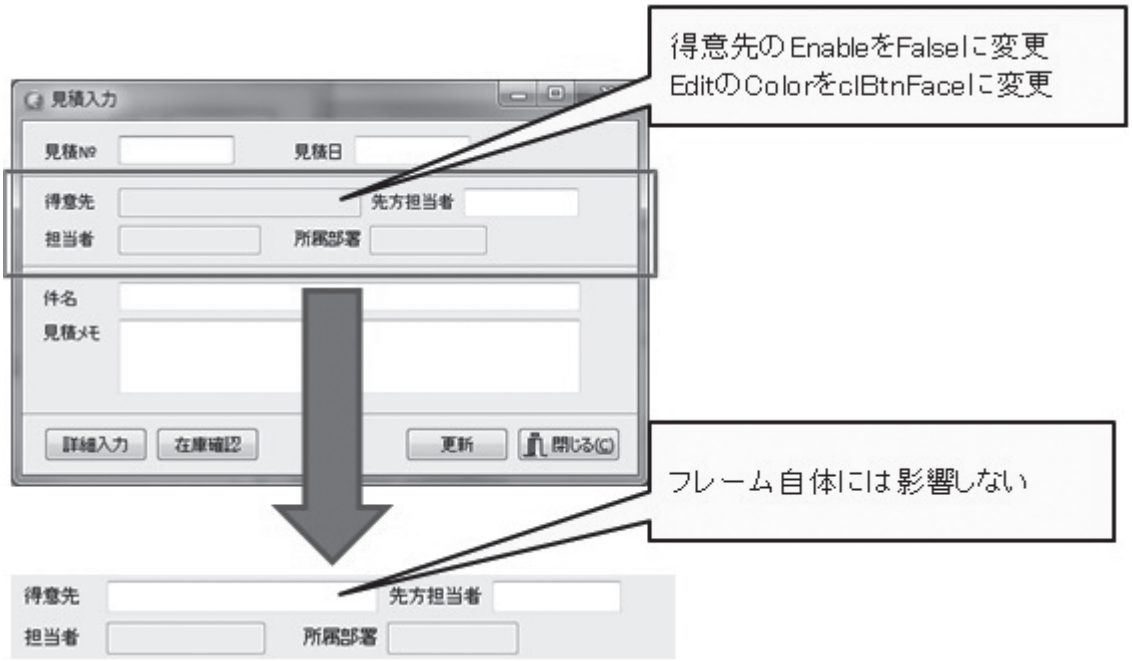
```
{*************************************************************************************
 目的：更新ボタン押下時処理
 引数：
 戻値：
*************************************************************************************}
procedure TfrmEstimates.bbtnUpdateClick(Sender: TObject);
begin

  SQLQuery1.ParamByName('TKSK').AsString := fraHeaderField1.TKSK;   // 得意先
  SQLQuery1.ParamByName('SPTN').AsString := fraHeaderField1.SPTN;   // 先方担当者
  SQLQuery1.ParamByName('TNTO').AsString := fraHeaderField1.TNTO;   // 担当者
  SQLQuery1.ParamByName('BUSH').AsString := fraHeaderField1.BUSH;   // 所属部署

  SQLQuery1.ExecSQL;

end;
```

フレーム名＋プロパティ定義名

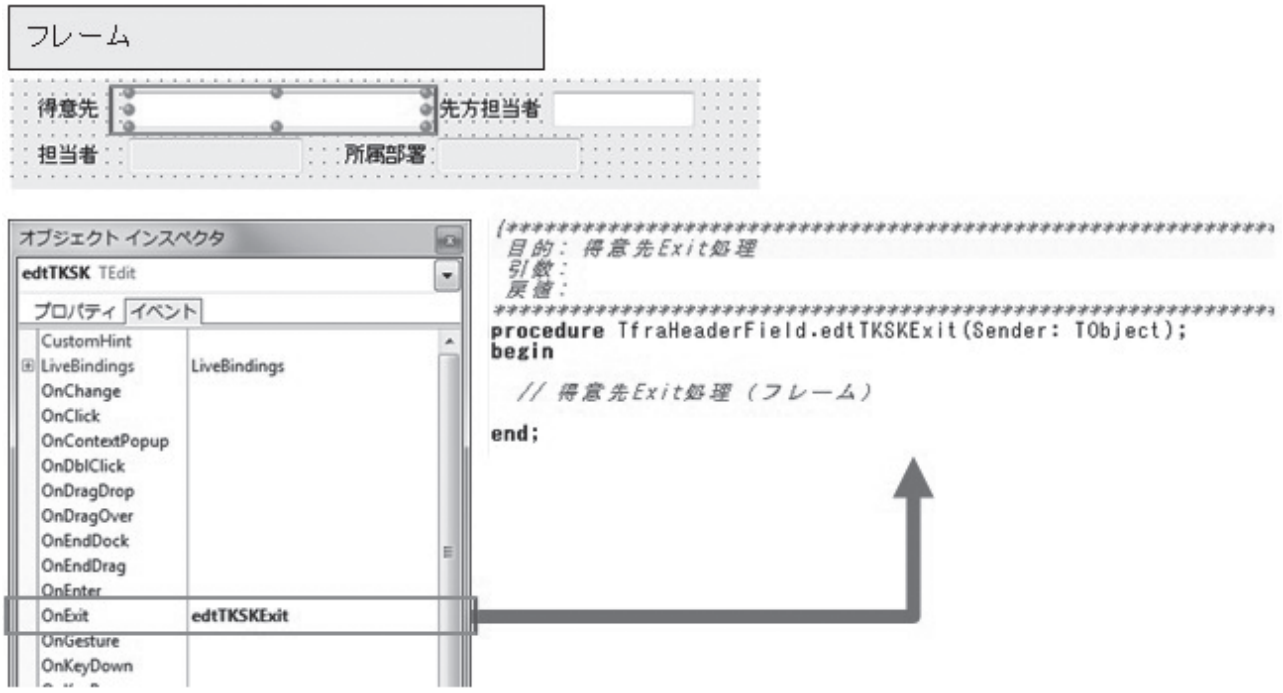
図19

削除しようとしたコンポーネント名

図20


得意先のEnableをFalseに変更
EditのColorをclBtnFaceに変更
見積入力
見積№
見積日
得意先
先方担当者
担当者
所属部署
件名
見積メモ
詳細入力
在庫確認
更新
閉じる(C)
フレーム自体には影響しない
得意先
先方担当者
担当者
所属部署

ソース5


フレーム
得意先
先方担当者
担当者
所属部署
オブジェクト インスペクタ
edtTKSK TEdit
プロパティ
イベント
CustomHint
LiveBindings
LiveBindings
OnChange
OnClick
OnContextPopup
OnDblClick
OnDragDrop
OnDragOver
OnEndDock
OnEndDrag
OnEnter
OnExit
edtTKSKExit
OnGesture
OnKeyDown
{***
目的：得意先Exit処理
引数：
戻値：
***}
procedure TfraHeaderField.edtTKSKExit(Sender: TObject);
begin
// 得意先Exit処理（フレーム）
end;

ソース6

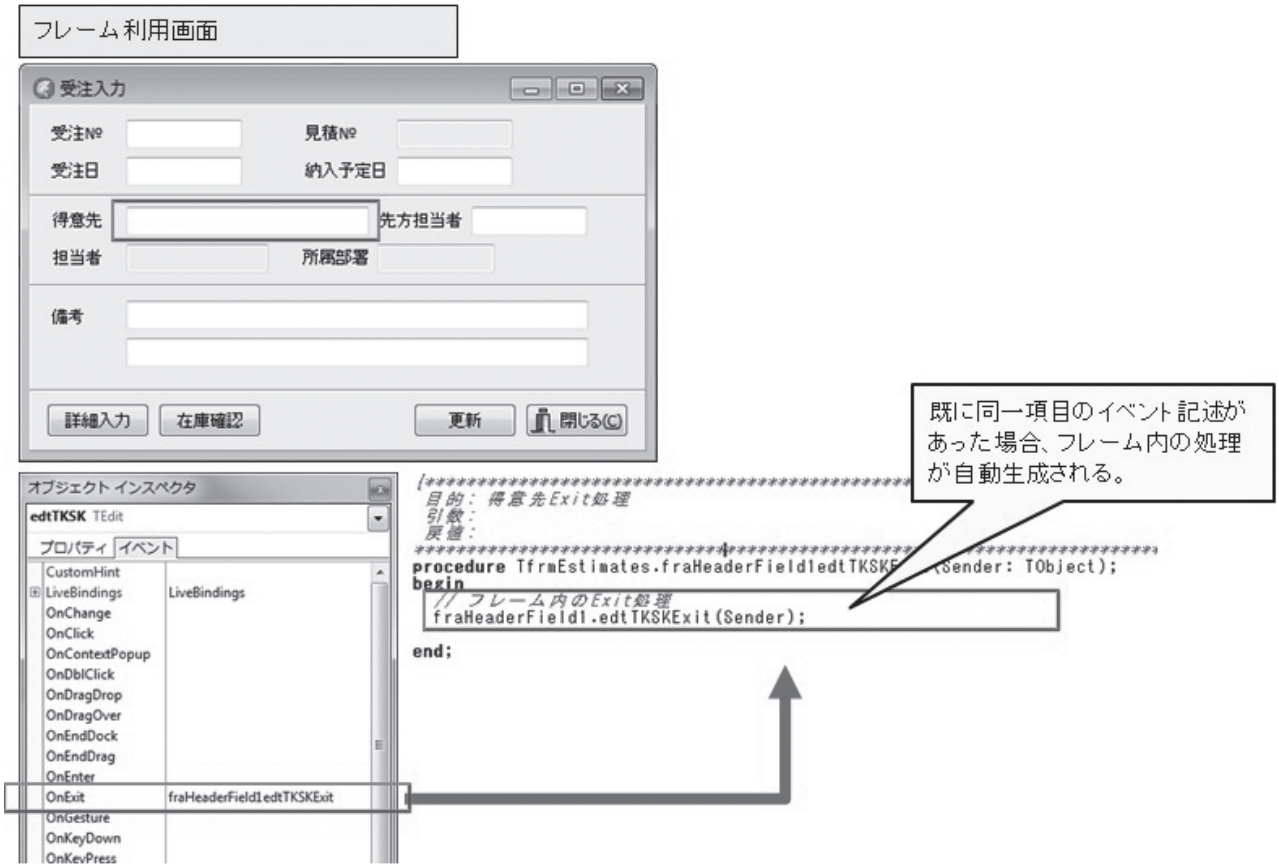

フレーム利用画面
受注入力
受注№
見積№
受注日
納入予定日
得意先
先方担当者
担当者
所属部署
備考
詳細入力
在庫確認
更新
閉じる(C)
オブジェクト インスペクタ
edtTKSK TEdit
プロパティ
イベント
CustomHint
LiveBindings
OnChange
OnClick
OnContextPopup
OnDblClick
OnDragDrop
OnDragOver
OnEndDock
OnEndDrag
OnEnter
OnExit
OnGesture
OnKeyDown
OnKeyPress
fraHeaderField1edtTKSKExit
目的：得意先Exit処理
引数：
戻値：
procedure TfrmEstimates.fraHeaderField1edtTKSK
(Sender: TObject);
begin
// フレーム内のExit処理
fraHeaderField1.edtTKSKExit(Sender);
end;
既に同一項目のイベント記述があった場合、フレーム内の処理が自動生成される。

図21

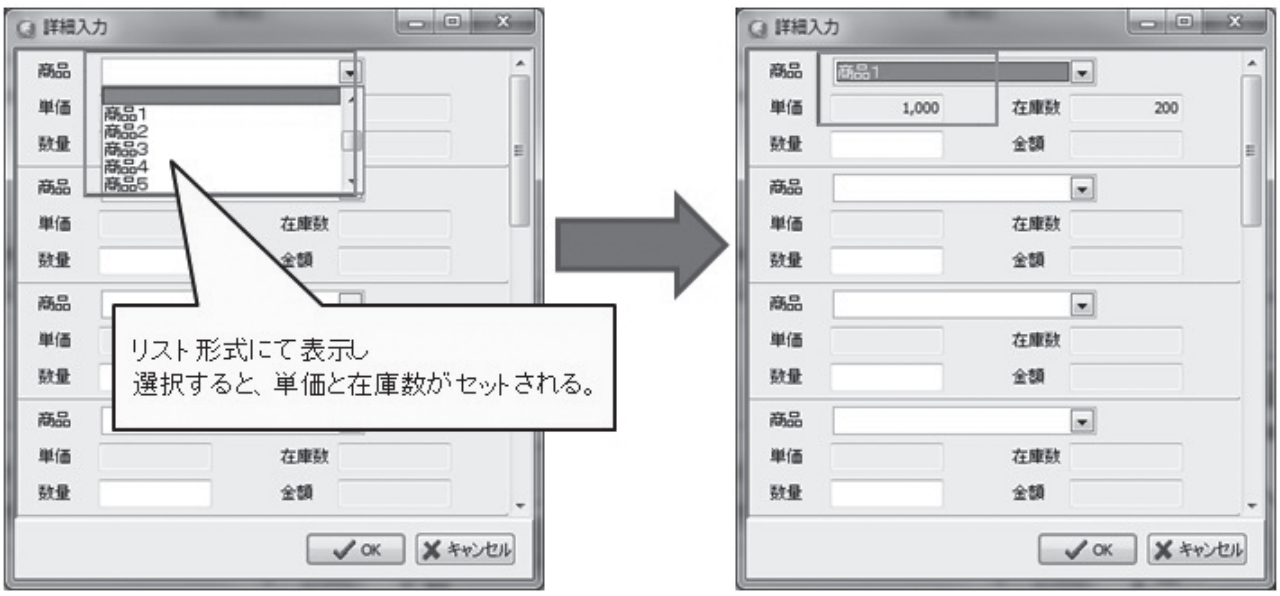

詳細入力
商品
単価
数量
在庫数
金額
商品1
商品2
商品3
商品4
商品5
OK
キャンセル
リスト形式にて表示し
選択すると、単価と在庫数がセットされる。
1,000
200

図22

商品マスタレイアウト

DSPPMT　　レ コ ー ド 設 計 書　　日付 15/08/17
時刻 18:57:46

物理ファイル	MAELIB/MGSHOHPF	様式名	MGSHOH	レコード長	6
様式記述	商品マスタサンプル				

5= 詳細

選択	項目名	桁数	属性	キー順	開始	終了	テキスト記述／欄見出し
_	MGSHCD	4	A		1	4	商品 CD
_	MGSHNM	48	O		5	52	商品名
_	MGTANK	6 0	S		53	58	単価
_	MGZAIK	6 0	S		59	64	在庫数

図23

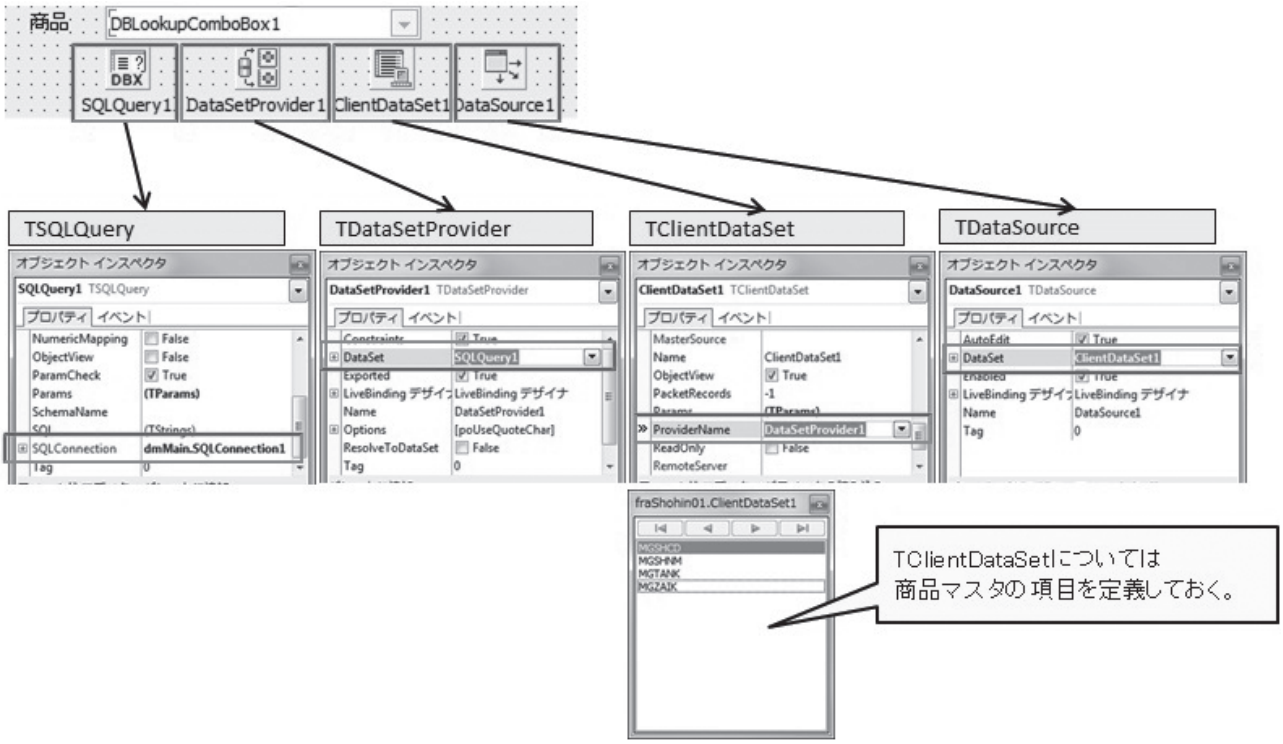


図24

TDBLookUpComboBoxのプロパティ設定

プロパティ名	設定値
ListSource	DataSource1
KeyField	MGSHCD
ListField	MGSHNM

ソース7

fraShohin01フレーム

```
private
  { Private 宣言 }
  FSHNM: String;
  FSHCD: String;
  FBlankAdd: Boolean;
  procedure SetSHCD(const Value: String);
  procedure SetSHNM(const Value: String);
  function GetSHCD: String;
  function GetSHNM: String;
public
  { Public 宣言 }
  constructor Create(AOwner: TComponent); override;
  destructor Destroy; override;

  property BlankAdd: Boolean read FBlankAdd write FBlankAdd;  // ブランク行
  property SHCD: String read GetSHCD write SetSHCD;           // 商品コード
  property SHNM: String read GetSHNM write SetSHNM;           // 商品名

  procedure SetListItem;
end;
```

ソース8

fraShohin01フレーム

```
{*****************************************************************************
 目的：リスト内容設定処理
 引数：
 戻値：
*****************************************************************************}
procedure TfraShohin01.SetListItem;
begin

  // データ取得SQL設定
  SQLQuery1.SQL.Text := ' SELECT * FROM MGSHOHPF ';

  // データセットのClose
  ClientDataSet1.Close;

  // データセットのOpen
  ClientDataSet1.Open;

  // 対象データがない場合
  if ClientDataSet1.IsEmpty then
  begin
    // データセットのClose
    ClientDataSet1.Close;
    Exit;
  end;

  // BlankAddプロパティがTrueの場合
  if (FBlankAdd) then
  begin
    try
      // ブランク行を追加
      ClientDataSet1.First;
      ClientDataSet1.Insert;

      ClientDataSet1.FieldByName('MGSHCD').AsString  := '  ';   // 商品コード
      ClientDataSet1.FieldByName('MGSHNM').AsString  := '  ';   // 商品名
      ClientDataSet1.FieldByName('MGTANK').AsInteger := 0;      // 単価
      ClientDataSet1.FieldByName('MGZAIK').AsInteger := 0;      // 在庫

      ClientDataSet1.Post;
    except
      ClientDataSet1.Cancel;
    end;
  end;

  // 初期値の設定
  if FBlankAdd then
  begin
    // 先頭行
    DBLookupComboBox1.KeyValue :=
      ClientDataSet1.FieldByName('MGSHCD').AsString;
  end
  else
  begin
    DBLookupComboBox1.KeyValue := '';
  end;
```

ソース9

fraShohin02フレーム


```
{*************************************************************************
 目的：単価セット時処理
 引数：
 戻値：
*************************************************************************}
procedure TfraShohin02.SetTNKA(const Value: Currency);
begin
  FTNKA := Value;

  edtTNKA.Text := FormatFloat('#,0', FTNKA);
end;

{*************************************************************************
 目的：在庫数セット時処理
 引数：
 戻値：
*************************************************************************}
procedure TfraShohin02.SetZAIK(const Value: Currency);
begin
  FZAIK := Value;

  edtZAIK.Text := FormatFloat('#,0', FZAIK);
end;

{*************************************************************************
 目的：商品選択時処理
 引数：
 戻値：
*************************************************************************}
procedure TfraShohin02.DBLookupComboBox1Click(Sender: TO
begin
  inherited;

  TNKA := ClientDataSet1.FieldByName('MGTANK').AsInteger;  // 単価
  ZAIK := ClientDataSet1.FieldByName('MGZAIK').AsInteger;  // 在庫数

end;
```

単価、在庫数のプロパティ定義にデータセットの値をセット

ソース10

フレーム利用画面

```
{*************************************************************************
 目的：画面表示時処理
 引数：
 戻値：
*************************************************************************}
procedure TfrmDetail.FormShow(Sender: TObject);
var
  i: Integer;
  fraShohin02: TfraShohin02;
  wCtl: TControl;
begin

  // フレーム初期化
  for i:= ScrollBox1.ControlCount-1 downto 0 do
  begin
    wCtl := ScrollBox1.Controls[i];

    if (wCtl is TfraShohin02) then
    begin
      FreeAndNil(wCtl);
    end;
  end;

  // フレーム生成
  for i := 0 to 10 do
  begin
    fraShohin02           := TfraShohin
    fraShohin02.Name      := 'fra' + FormatFloa                    // フレーム名
    fraShohin02.TabOrder  := i - 1;                                // タブ順
    fraShohin02.Parent    := ScrollBox1;                           // 親コンポーネント
    fraShohin02.Top       := Height * (i);                         // フレーム位置
    fraShohin02.Align     := alTop;

    fraShohin02.BlankAdd := True;                                  // リストにブランク行追加
    fraShohin02.SetListItem;                                       // リスト内容の設定
  end;

end;
```

フレーム利用画面は
SetListItemを呼び出すだけでリスト内容を設定できる。
BlankAddプロパティをTrueに設定すると、ブランク行を追加。

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]

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

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

ここ数年、業務用としてタブレットを導入、または導入を検討する企業が増えている。そのタブレットを導入する際に重要な検討項目となるのが、OS の選択である。

Delphi/400 では iOS や Android のアプリケーションも開発できるが、PC と同様に Windows を搭載したタブレットを選択する企業も少なくない。それは、これまで購入してきた Windows ソフトウェアや、現在使用中の業務システムをタブレットでもそのまま使いたいという理由からである。

Windows タブレット用に新たにアプリケーションを開発する場合、その方法自体は、PC 用のアプリケーション開発と同じである。ただし Windows タブレットのソフトウェアキーボードは、iOS や Android のそれと比べると操作性に難があり、不便と感じられることがある。

そこで本稿では、ユーザーにとって操作性のよい Windows タブレット用ソフトウェアキーボードの実装方法を紹介する。

2.ソフトウェアキーボードについて

2-1. ソフトウェアキーボードとは

本題に入る前に、ソフトウェアキーボードについて少し説明する。ソフトウェアキーボードとは、物理的なキーボード機器を使用せずに画面上でキーボードを表示し、マウスクリックや画面タッチによって文字や数値等を入力できるソフトウェアである。

2-2. Windows 標準のソフトウェアキーボード

ここでは、Windows 8 搭載のタブレットに装備されている標準ソフトウェアキーボードについて基本的な動作を確認する。

ソフトウェアキーボードを表示するには、タスクバーに表示されているキーボードイメージのアイコンをタッチする。【図 1】

先ほど、Windows タブレットの標準ソフトウェアキーボードは、「操作性に難があり、不便と感じられることがある」と記したが、具体的には次のような点である。

①アプリケーションの入力項目にフォーカスが移っても、ソフトウェアキーボードは自動的に表示されず、タスクバーのキーボードアイコンをタッチして表示させなければならない。【図 2】
②ソフトウェアキーボードを開くと、アプリケーションの画面に被ってしまい入力項目が隠れてしまう場合がある。【図 3】

タブレット用のアプリケーションを開発する際、上記が問題となることが多い。そこで、それを改善するため標準のソフトウェアキーボードは使用せずに、アプリケーションの中でソフトウェアキーボード機能を実装する方法を紹介す

図1


ここをタッチする

図2

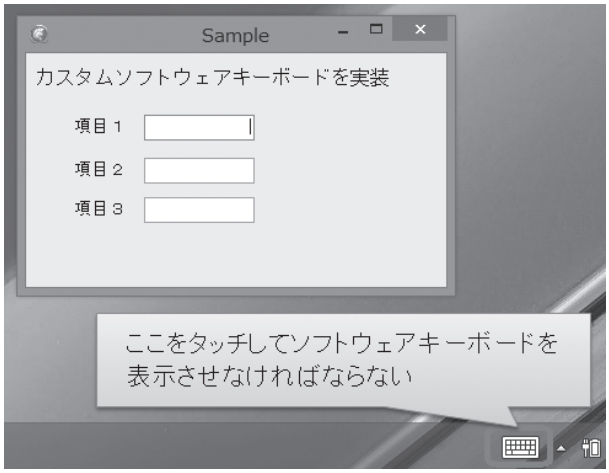

Sample
カスタムソフトウェアキーボードを実装
項目１
項目２
項目３
ここをタッチしてソフトウェアキーボードを
表示させなければならない

図3


Sample
カスタムソフトウェアキーボードを実装
項目１

る。

2-3. カスタムソフトウェアキーボードを独自実装

タブレットでは、ユーザーインターフェースとしてコンボボックスやボタン類を多用し、キーボード入力を極力少なくするのがセオリーである。しかし、数量や金額などの数値に関しては、リストから選択させるわけにもいかず、キーボードからの入力が必要になることが多い。そこで、これから紹介するソフトウェアキーボードは、タブレットで使用率が高い、テンキーのキーボードを題材として説明していく。

3.TTouchKeyboardコンポーネントでの実装

3-1. TTouchKeyboardコンポーネントの実装手順

Delphi/400 では、Ver.2010 から TTouchKeyboard コンポーネントが実装されている。このコンポーネントを使用することで、アプリケーション中にソフトウェアキーボード機能を簡単に実装できる。

TTouchKeyboard コンポーネントをフォームに配置すると、ソフトウェアキーボードがフォーム上に表示される。またプロパティの設定だけで、通常の文字キーボードとテンキーのイメージを切り替えることができる。【図 4】

ここから、TTouchKeyboard を使用したソフトウェアキーボードの実装手順について説明する。

①コンポーネント配置

新規フォーム上に TEdit コンポーネントを 2 個と、TTouchKeyboard コンポーネントを配置する。【図 5】

② TTouchKeyboard のプロパティ設定

TTouchKeyboard コンポーネントの Layout プロパティを「NumPad」に設定し、テンキーイメージにする。【図 6】

③実装完了

TTouchKeyboard コンポーネントとフォームのサイズを整えて実装は完了となる。【図 7】

TTouchKeyboard コンポーネントの Layout プロパティを変更した以外、特にソースを記述する必要はないので、非常に簡易に実装できる。

3-2. TTouchKeyboard コンポーネントの機能

この節では、実際の動作について確認する。

アプリケーションを実行すると、フォーカスが Edit1 にある状態で画面が表示される。ソフトウェアキーボードの「1」「2」をタッチすると、Edit1 に「1」「2」がセットされる。【図 8】

次に、Edit2 にフォーカスを移してソフトウェアキーボードの「4」「5」をタッチすると、Edit2 に「4」「5」がセットされる。【図 9】

このように TTouchKeyboard コンポーネントは、同一フォーム上のアクティブなコンポーネントに対して値をセットするため、入力項目があるフォーム上に配置する必要はあるものの、ソフトウェアキーボードを簡単に実装したい場合に便利なコンポーネントである。

ただし、アプリケーションによっては、ソフトウェアキーボードを表示する領域が画面デザイン上、難しい場合もある。そうした場合には、アプリケーションとは別のウインドウにソフトウェアキーボードを分離させる必要がある。そこで、別のウインドウとしてソフトウェアキーボードを作成し、アプリケーションに実装する方法を次章で紹介していく。

4.カスタムソフトウェアキーボードを実装

4-1. ソフトウェアキーボードの開発手順

この章では、【図 10】のように入力画面とソフトウェアキーボードを別フォームとして作成し、ソフトウェアキーボードが入力画面に対してキーボードとして動作するように実装する方法を説明する。この実装方法では、簡易な TTouchKeyboard コンポーネントを使わずに独自にソフトウェアキーボードを作成するため、細かい制御を自由に実装できる。

まず、ソフトウェアキーボードを新しいフォームとして作成する。今回作成する画面イメージは【図 11】の通りである。

機能としては、数値入力、マイナス入力、小数点入力、BackSpace、フォーカスの移動（次項目、前項目）、入力項目値の全選択機能を実装する。では、作成手順を順番に説明する。

①コンポーネント配置

【図 11】に従って、新規フォーム上に TBitBtn コンポーネントを 16 個配置し、Caption プロパティを設定する。続いて【表 1】に従って、各 TBitBtn コンポーネントの Name プロパティを設定する。

②フォームのプロパティ設定

フォームの BorderStyle プロパティを「bsSingle」に設定して、ソフトウェアキーボードの画面サイズを変更不可にする。次に FormStyle プロパティを「fsStayOnTop」に設定し、ソフトウェアキーボードが常に手前で表示されるようにする。【図 12】

③グローバル変数の宣言

グローバル変数として DeActivateForm を記述しておく【ソース 1】。これは DeActivateForm 変数で受け取ったフォームに対して、別ウインドウのソフトウェアキーボードでボタンタッチされた結果を反映するためである。

④フォームの onCreate イベントハンドラの実装

フォームの onCreate イベントハンドラを次のように実装する。【ソース 2】

ここでは、btnKey0 ～ btnKey9、btnKeyDot、btnKeyMinus の Tag プロパティに、ボタンの Caption に該当する文字コードを設定しておく。今回は数値と記号のみだが、文字キーボードを実装する場合には「Ord('A')」といった形で、引数にアルファベットを指定することもできる。

次に、btnKeyBS の Tag プロパティには BackSpace キーに該当する制御コードを設定し、btnNext と btnPrior の Tag プロパティには、前項目への移動なのか、次項目への移動なのかを判断するための区分を設定しておく。ここで設定した内容は、後で説明する TBitBtn コンポーネントの onClick イベントハンドラで使用することになる。そして、ソフトウェアキーボードの Top と Left を指定し、初期表示位置を画面右下になるよう設定しておく。

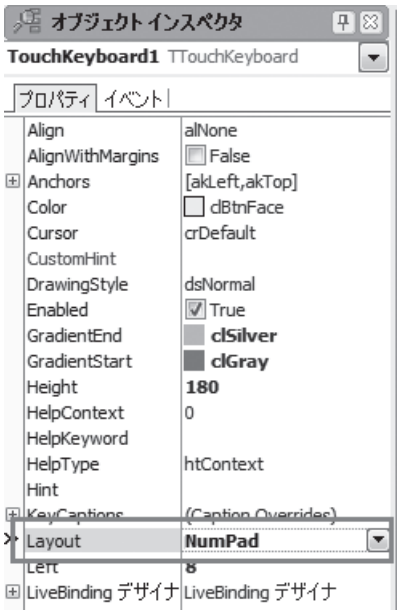
図4


通常のキーボード
テンキー
プロパティの設定で切り替え

図5


Sample
Edit1
Edit2

図6


オブジェクト インスペクタ
TouchKeyboard1 TTouchKeyboard
プロパティ
イベント
Align alNone
AlignWithMargins False
Anchors [akLeft,akTop]
Color clBtnFace
Cursor crDefault
CustomHint
DrawingStyle dsNormal
Enabled True
GradientEnd clSilver
GradientStart clGray
Height 180
HelpContext 0
HelpKeyword
HelpType htContext
Hint
KeyCaptions (Caption Overrides)
Layout NumPad
Left 8
LiveBinding デザイナ

⑤ TBitBtn コンポーネントの onClick イベントハンドラ

各 TBitBtn コンポーネントの onClick イベントハンドラを実装する。イベントハンドラはボタンごとに用意するのではなく、機能ごとに共通化して 3 つのイベントハンドラを用意する。【ソース 3】

各イベントハンドラのソースは、【ソース 4】のようになる。そして、各 TBitBtn コンポーネントの onClick イベントハンドラを次のように設定する。【表 2】

btnKeyBoardClick では、DeActivate Form 変数で受け取ったフォームを BringToFront を使用してアクティブにし、フォームのアクティブコンポーネントに対して TBitBtn の Tag プロパティに設定されている文字コードや制御コードを、PostMessage 関数を使用してメッセージ送信している。この実装で、ソフトウェアキーボードの数値、記号、「BS」を押した結果が入力項目に反映されるようになる。

btnSelectALL で は、btnKeyBoard Click 同様にフォームをアクティブにし、フォームのアクティブコンポーネントに対して値の全選択命令をメッセージ送信している。

btnFocusControl では、btnKeyBoard Click 同様にフォームをアクティブにし、TBitBtn の Tag プロパティに設定されている区分によってフォームのアクティブコンポーネントの次項目または前項目にフォーカスを移動させる命令をメッセージ送信している。

以上でソフトウェアキーボードの作成は完了である。

4-2. 入力画面への実装手順

続いて、入力画面へソフトウェアキーボードを実装する方法について説明する。実装の手順は、次の通りである。

①入力画面の作成

前節のソフトウェアキーボードと同じプロジェクト内に、新規フォームを追加して TEdit コンポーネントを 3 個配置する【図 13】。そして、ソフトウェアキーボードのユニットを参照しておく。また、プロジェクトオプションのフォームの設定では、ソフトウェアキーボードも自動生成の対象としておく。【図 14】

②フォームの onShow イベントハンドラの実装

ソフトウェアキーボードは【図 14】で自動生成されているため、フォームの onShow イベントでは、ソフトウェアキーボードを Show することで表示させることができる。【ソース 5】

③フォームの onDeactivate イベントハンドラの実装

ソフトウェアキーボードのボタンを押した際、入力中のフォームからソフトウェアキーボードへ制御が移る。この時にソフトウェアキーボードの Deactivate Form 変数に入力中のフォームをセットしておくため、フォームの onDeactivate イベントハンドラを次のように実装する。【ソース 6】

以上で入力画面への実装は完了である。

4-3. カスタムソフトウェアキーボードの機能

実行して動作を確認する。アプリケーションを実行すると入力画面とソフトウェアキーボードが表示され、入力画面の Edit1 にフォーカスが設定される。【図 15】

入力画面とソフトウェアキーボードは別々のフォームになっているので、ソフトウェアキーボードは画面上の好きな位置に移動できる。

実際にソフトウェアキーボードを使用して入力してみよう。ソフトウェアキーボードの「1」「2」をタッチすると Edit1 に「1」「2」がセットされ、「全選択」をタッチすると、入力内容が全選択された状態となる。【図 16】

この状態で Edit1 をタッチ長押しでポップアップメニューを開き、コピーを選択すると、入力中の値をコピーすることもできる。

次に、「BS」をタッチすると、Edit1 に入力されていた「12」がクリアされる。さらに「次項目」を押せば Edit2 へ移り、「前項目」を押せば Edit1 へフォーカスを戻すことができる。

このようにソフトウェアキーボードを独自に作成する場合は、TTouchKeyboard コンポーネントよりも実装に手間がかかるが、別ウインドウとして制御したり、独自の機能を実装できる利点がある。

ここで、Windows タブレットのソフトウェアキーボードが、どのように改善されたのかをまとめておく。

1 つ目の「アプリケーションの入力項目にフォーカスが移っても、ソフトウェアキーボードは自動的に表示されず、タスクバーのキーボードアイコンをタッチしなければならない」に関しては、ソフトウェアキーボードを常に表示させておくことで、入力項目にフォーカスが移っても、すぐに入力できるようになり解決できる。

2 つ目の「ソフトウェアキーボードを開くと、アプリケーションの画面に被ってしまい入力項目が隠れてしまう場合がある」に関しては、画面起動時にソフトウェアキーボードを入力項目に被らない位置に表示させることで解決できる。ただし、今回の説明で使用した入力画面は項目も少なく画面サイズも小さいため、入力項目に重ならない位置に表示できた。しかし、入力項目が多い画面では、ソフトウェアキーボードがどうしても入力項目に被ってしまうケースが出てくる。

そこで、次節では入力項目にソフトウェアキーボードが重なってしまった場合に、ソフトウェアキーボードの位置を変える拡張方法について紹介する。

4-4. カスタムソフトウェアキーボードの移動

ここでは、【図 17】のように Edit2 にソフトウェアキーボードが被っている状態で、Edit2 にフォーカスを移した際にソフトウェアキーボードを下方向にずらす調整方法を説明する。

この制御を実装するには、入力項目である各 TEdit コンポーネントの onEnter イベントハンドラを利用して実装する。イベントハンドラは【ソース 7】のように共通化した 1 つのイベントハンドラとして用意する。

イベントハンドラのソースを【ソース 8】に示す。

作成した共通イベントハンドラは、各 TEdit コンポーネントの onEnter イベントに設定する。

【ソース 8】について、ソースコードのポイントを説明する。

図7

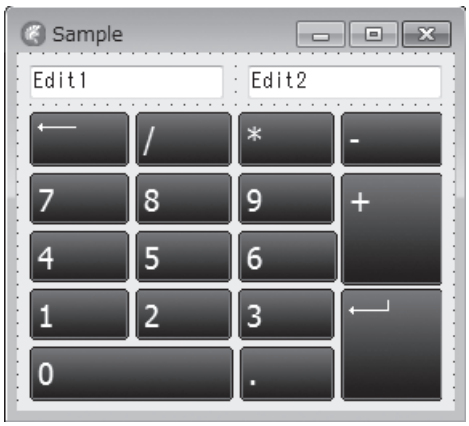

Sample
Edit1
Edit2
←
/
*
-
7
8
9
+
4
5
6
1
2
3
↵
0
.

図8

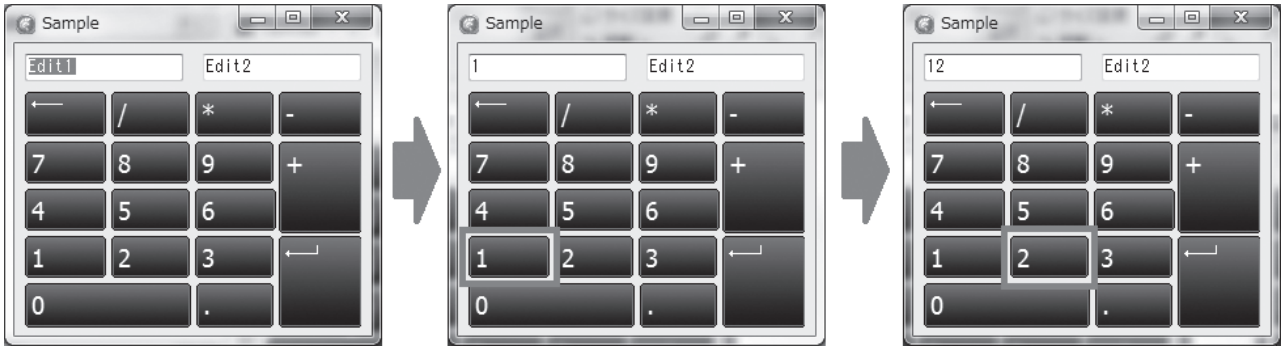
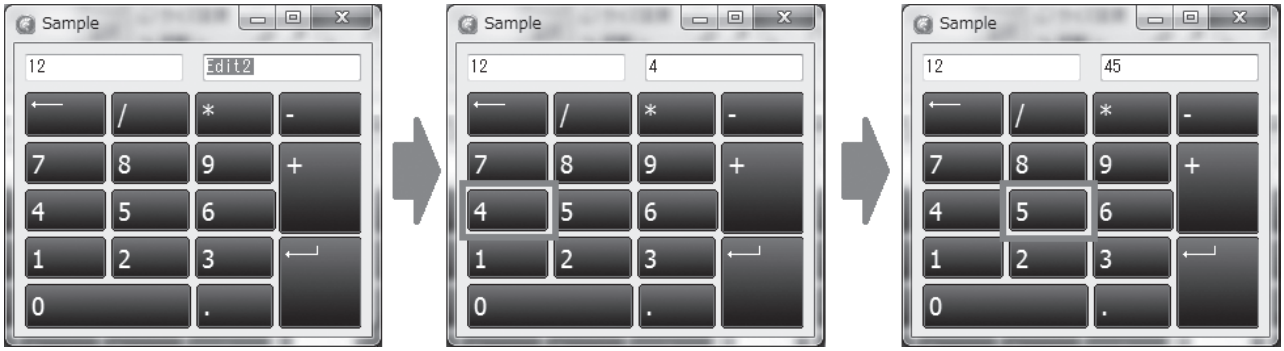

Sample
Edit1
Edit2
1
12

図9


Sample
12
Edit2
4
45

位置の調整については、最初にTEditコンポーネントの位置（Top、Left）とソフトウェアキーボードの位置（Top、Left）を、スクリーンを基準にして算出する。

次に、重なり合う部分を計算するIntersectRect関数を使用して、TEditコンポーネントとソフトウェアキーボードの重なりを求める。今回は上下方向の重なりを判断している（if rRes.Height > 0 then）。横方向の重なりを判断したい場合は、rRes.Widthを使用することで判断できる。

そして、重なりがあった場合にソフトウェアキーボードを、TEditの底辺がTop位置になるよう下方向に移動させるのだが、その前に下方向に移動した場合のソフトウェアキーボードの底辺の位置を計算し（iBot := rCon.Top + rCon.Height + frmKeyBoard.Height + 5）、スクリーン下にはみ出ないかを判断する（if iBot > Screen.Height then）。

スクリーン下にはみ出る場合は、TEditのTop位置がソフトウェアキーボードの底辺になるよう上方向に移動し（frmKeyBoard.Top := rCon.Top - frmKeyBoard.Height - 5）、はみ出なければ下方向に移動させる（frmKeyBoard.Top := rCon.Top + rCon.Height + 5）。

実装内容は以上である。

次に、実際にアプリケーションを実行して動作を確認する。Edit2とEdit3にソフトウェアキーボードが重なっている状態で、フォーカスをEdit1→Edit2→Edit3と移動させると、ソフトウェアキーボードがフォーカスの移動に合わせて下方向に移動していくことが確認できる。【図18】

また、Edit3にフォーカスが移動した際、ソフトウェアキーボードがスクリーン下にはみ出る場合は、【図19】のように上方向に移動する。

こうした実装を行うことで、前節で述べていた入力項目が多い画面であっても、フォーカスの移動に合わせてソフトウェアキーボードの位置を調整し、入力項目を見えるようにできるので、ユーザーが使用する際に非常に便利である。また、この方法を用いれば、離れた位置にあるソフトウェアキーボードを入力項目の近くに移動させたり、入力項目からフォーカスが抜けた時に画面外にソフトウェアキーボードを移動させる、といった制御も可能である。

5.まとめ

本稿では、Windowsタブレットの標準ソフトウェアキーボードが持つ不便さの解消を目的に、アプリケーションによる実装方法を紹介した。またデスクトップPCやノートPCで使用するアプリケーションの場合でも、今回のソフトウェアキーボードを実装することで、キーボードを使用することなくマウスのみで入力操作が可能な画面設計を行える。

実装方法については、TTouchKeyboardコンポーネントを使った簡単な方法と、独自にソフトウェアキーボードを作成する方法を取り上げたが、どちらも有効な方法なので、アプリケーションの画面設計や用途によって使い分けるとよい。

また今回は、単純なテンキーのキーボードを題材にしたが、同様の方法で文字キーボードの実装や、新しい機能の追加も可能である。

Windowsアプリケーションについても、タブレットなどのスマートデバイス端末での使用が増えてきている。そうした開発を行う中で、デスクトップPCやノートPCにはなかった新しい課題に取り組むことも多い。

開発方法はこれまでと同様であっても、使用するデバイスが変わると、使い勝手や求められるユーザーインターフェースも違ってくる。こうしたことを常に頭において設計を工夫することが、これからのアプリケーション開発では重要である。

M

図10

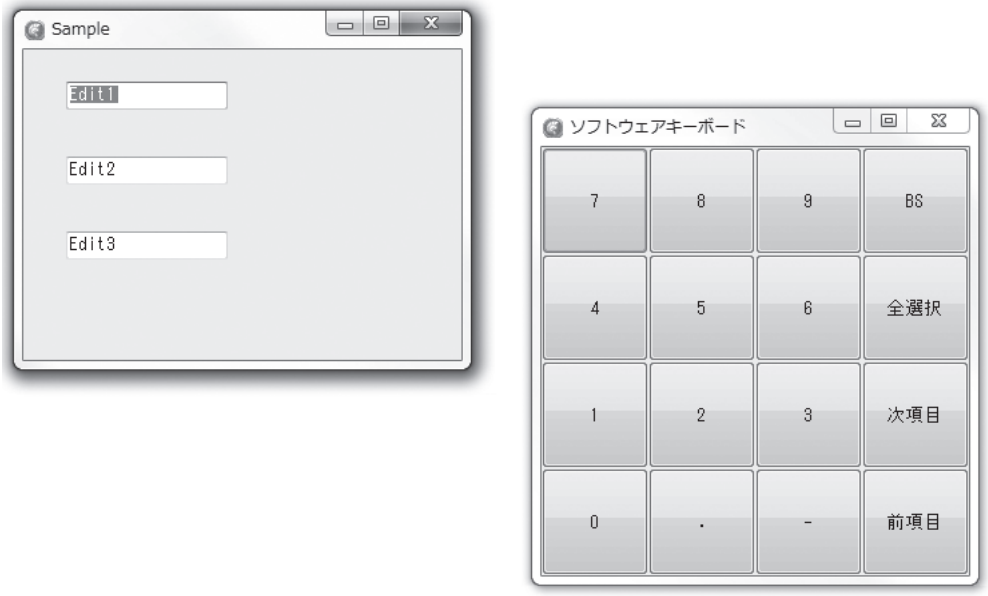


図11

表1

TBitBtnコンポーネントのNameプロパティの設定

Caption	Name
0	btnKey0
1	btnKey1
2	btnKey2
3	btnKey3
4	btnKey4
5	btnKey5
6	btnKey6
7	btnKey7
8	btnKey8
9	btnKey9

Caption	Name
.	btnKeyDot
-	btnKeyMinus
BS	btnKeyBS
全選択	btnSelAll
次項目	btnNext
前項目	btnPrior

図12

ソース1

```
var
  frmKeyBoard: TfrmKeyBoard;
  DeActivateForm: TForm;

implementation
```

ソース2

```
procedure TfrmKeyBoard.FormCreate(Sender: TObject);
begin
  // TBitBtnのTagプロパティを設定
  btnKey7.Tag      := Ord('7');
  btnKey4.Tag      := Ord('4');
  btnKey1.Tag      := Ord('1');
  btnKey0.Tag      := Ord('0');

  btnKey8.Tag      := Ord('8');
  btnKey5.Tag      := Ord('5');
  btnKey2.Tag      := Ord('2');
  btnKeyDot.Tag    := Ord('.');

  btnKey9.Tag      := Ord('9');
  btnKey6.Tag      := Ord('6');
  btnKey3.Tag      := Ord('3');
  btnKeyMinus.Tag  := Ord('-');

  btnKeyBS.Tag     := $08;

  btnNext.Tag      := 0;
  btnPrior.Tag     := 1;

  // ソフトウェアキーボードの初期表示位置を設定
  Top  := Screen.WorkAreaTop  + Screen.WorkAreaHeight - Height;
  Left := Screen.WorkAreaLeft + Screen.WorkAreaWidth  - Width;
end;
```

ソース3

```
  procedure btnKeyBoardClick(Sender: TObject);
  procedure btnSelectALL(Sender: TObject);
  procedure btnFocusControl(Sender: TObject);
private
  { Private 宣言 }
public
  { Public 宣言 }
end;
```

ソース4

btnKeyBoardClick

```
procedure TfrmKeyBoard.btnKeyBoardClick(Sender: TObject);
begin
  if Assigned(DeActivateForm) then
  begin
    DeActivateForm.BringToFront;
    PostMessage(DeActivateForm.ActiveControl.Handle, WM_CHAR, (Sender as TBitBtn).Tag, 0);
  end;
end;
```

DeActivateForm変数で受け取ったフォームをアクティブにする

TBitBtnのTagプロパティに設定されている
キャラクターコードや制御コードをメッセージ送信

btnSelectALL

```
procedure TfrmKeyBoard.btnSelectALL(Sender: TObject);
begin
  if Assigned(DeActivateForm) then
  begin
    DeActivateForm.BringToFront;
    PostMessage(DeActivateForm.ActiveControl.Handle, EM_SETSEL, 0, -1);
  end;
end;
```

DeActivateForm変数で受け取ったフォームをアクティブにする

値の全選択命令をメッセージ送信

btnFocusControl

```
procedure TfrmKeyBoard.btnFocusControl(Sender: TObject);
begin
  if Assigned(DeActivateForm) then
  begin
    DeActivateForm.BringToFront;
    PostMessage(DeActivateForm.Handle, WM_NEXTDLGCTL, (Sender as TBitBtn).Tag, 0);
  end;
end;
```

DeActivateForm変数で受け取ったフォームをアクティブにする

TBitBtnのTagプロパティに設定されている区分によって、
次項目or前項目にフォーカスを移動させる命令をメッセージ送信

表2

TBitBtnコンポーネントのonClickイベントハンドラの設定

TBitBtn	onClick
btnKey0	btnKeyBoardClick
btnKey1	btnKeyBoardClick
btnKey2	btnKeyBoardClick
btnKey3	btnKeyBoardClick
btnKey4	btnKeyBoardClick
btnKey5	btnKeyBoardClick
btnKey6	btnKeyBoardClick
btnKey7	btnKeyBoardClick
btnKey8	btnKeyBoardClick
btnKey9	btnKeyBoardClick

TBitBtn	onClick
btnKeyDot	btnKeyBoardClick
btnKeyMinus	btnKeyBoardClick
btnKeyBS	btnKeyBoardClick
btnSelAll	btnSelectALL
btnNext	btnFocusControl
btnPrior	btnFocusControl

図13

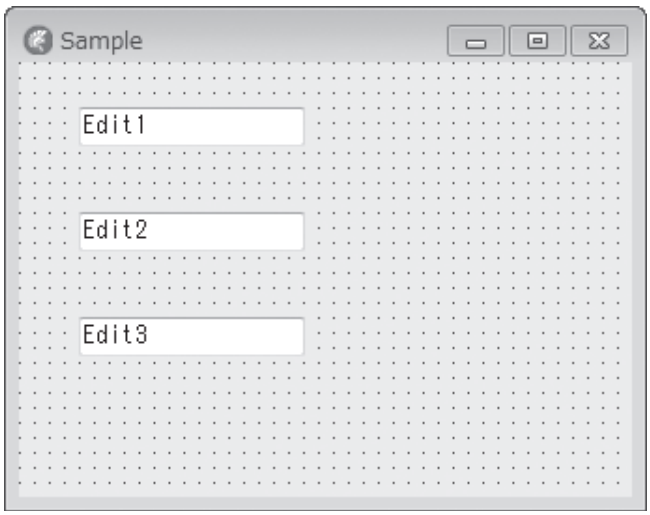


図14

ソース5

```
procedure TForm1.FormShow(Sender: TObject);
begin
  // ソフトウェアキーボードを表示
  frmKeyBoard.Show;
end;
```

ソース6

```
procedure TForm1.FormDeactivate(Sender: TObject);
begin
  // ソフトウェアキーボードの"DeactivateForm"に自身をセット
  DeactivateForm := Self;
end;
```

図15

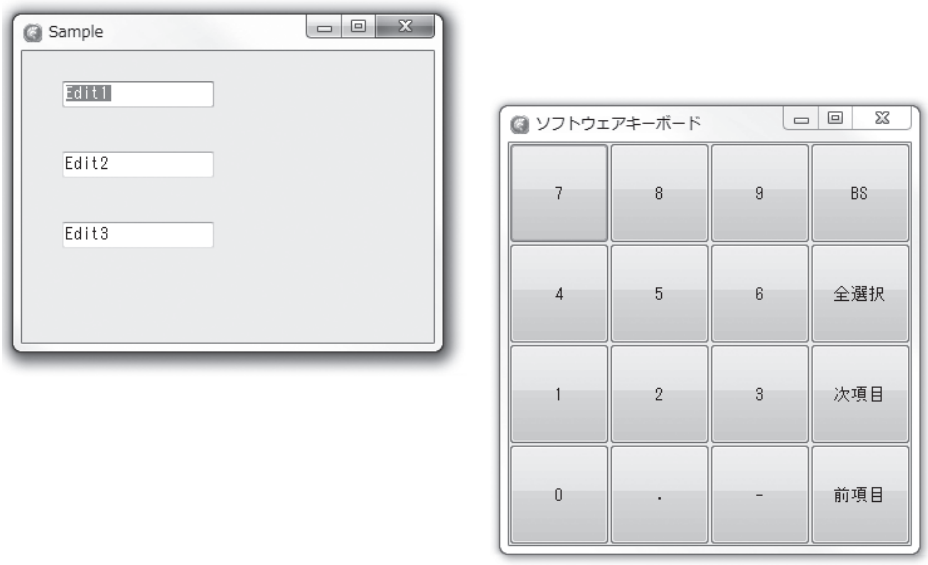
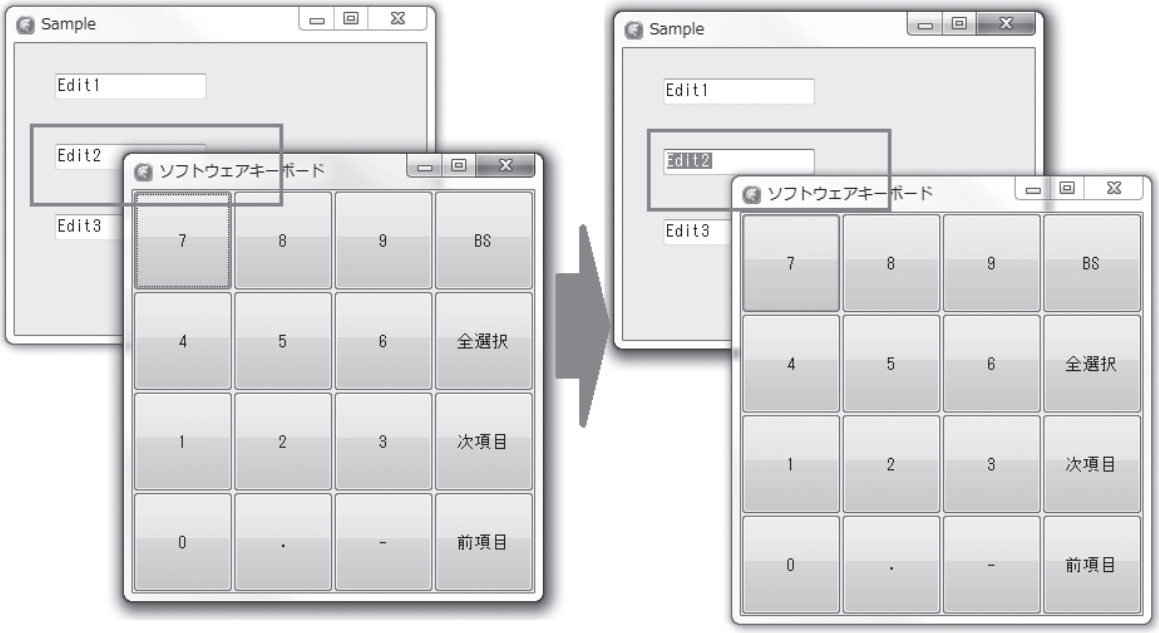

Sample
Edit1
Edit2
Edit3
ソフトウェアキーボード
7
8
9
BS
4
5
6
全選択
1
2
3
次項目
0
.
-
前項目

図16


Sample
Edit1
Edit2
Edit3
ソフトウェアキーボード
7
8
9
BS
4
5
6
全選択
1
2
3
次項目
0
.
-
前項目

図17


Sample
Edit1
Edit2
Edit3
ソフトウェアキーボード
7
8
9
BS
4
5
6
全選択
1
2
3
次項目
0
.
-
前項目
Sample
Edit1
Edit2
Edit3
ソフトウェアキーボード
7
8
9
BS
4
5
6
全選択
1
2
3
次項目
0
.
-
前項目

ソース7

```
    procedure FormDeactivate(Sender: TObject);
    procedure EditEnter(Sender: TObject);
  private
    { Private 宣言 }
```

ソース8

```
procedure TForm1.EditEnter(Sender: TObject);
var
  rCon, rFrm, rRes: TRect;
  pCon: TPoint;
  iBot: Integer;
begin
  if (Sender is TWinControl) and (Assigned(frmKeyBoard)) then
  begin
    // コンポーネントの位置をスクリーン座標で求める
    pCon.X := (Sender as TWinControl).Left;
    pCon.y := (Sender as TWinControl).Top;
    pCon   := Form1.ClientToScreen(pCon);

    // コンポーネントのRect
    rCon.Top    := pCon.Y;
    rCon.Left   := pCon.X;
    rCon.Height := (Sender as TWinControl).Height;
    rCon.Width  := (Sender as TWinControl).Width;

    // キーボードのRect
    rFrm.Top    := frmKeyBoard.Top;
    rFrm.Left   := frmKeyBoard.Left;
    rFrm.Height := frmKeyBoard.Height;
    rFrm.Width  := frmKeyBoard.Width;

    // コンポーネントとフォームの重なりを求める
    IntersectRect(rRes, rCon, rFrm);

    // 重なりが有った場合キーボードを移動
    if rRes.Height > 0 then
    begin
      // キーボードを移動した結果のフォームの下位置を求める
      // [TEditのTop]+[TEditのHeight]+[キーボードのHeight]+[調整値]
      iBot := rCon.Top + rCon.Height + frmKeyBoard.Height + 5;

      // キーボードがスクリーンの外にはみ出したかどうかを判断
      if iBot > Screen.Height then
        frmKeyBoard.Top := rCon.Top - frmKeyBoard.Height - 5  // TEditの上に移動
      else
        frmKeyBoard.Top := rCon.Top + rCon.Height + 5;        // TEditの下に移動
    end;
  end;
end;
```

図18

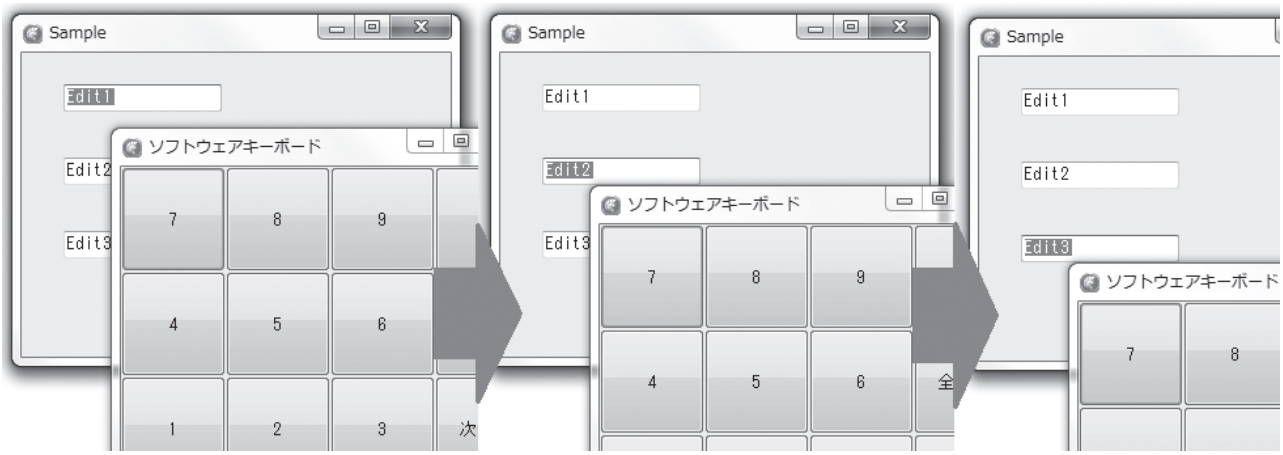

Sample
Edit1
Edit2
Edit3
ソフトウェアキーボード
7
8
9
4
5
6
1
2
3
次
全

図19


ソフトウェアキーボード
7
8
9
BS
4
5
6
全選択
1
2
3
次項目
0
.
-
前項目
Sample
Edit1
Edit2
Edit3

尾崎 浩司

株式会社ミガロ.

RAD事業部 営業・営業推進課

[Delphi/400] マルチスレッドを使用したレスポンスタイム向上

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. 製品の素晴らしさをアピールするためのセミナーやイベントの企画・運営などを主に担当している。

1.はじめに

アプリケーション開発において一般的に重要なのは、仕様通りの動作ができること、画面の使い勝手がよいことなどが挙げられるが、もう一つ重要な要素は、処理レスポンスである。せっかくの便利なアプリケーションであっても、処理レスポンスが悪いとユーザーはなかなか利用してくれない。しかし、大量データの処理や複雑な業務ロジックの実行は、一般的に時間がかかる場合が多い。

本稿では、複雑で処理時間がかかる処理をいかにユーザーが快適に使えるものにするかについて、技術的な解決手法を紹介する。

2.スレッドについて

2-1. スレッドとは

アプリケーションの処理を考える上で、「スレッド」と「プロセス」という概念は非常に重要である。Windows アプリケーションは、通常「プロセス」という単位で処理が行われる。実行中のプロセスは、Windows のタスクマネージャーでも確認することができる。【図 1】

プロセスとは、アプリケーションの実行単位である。つまり、プロセスは、それぞれ固有のメモリ空間をもって実行される独立したアプリケーションとして扱われる。

もう一つの概念に「スレッド」がある。スレッドとは、プロセスの中で、1 つ、あるいは複数動作するプログラムの一連の流れである。プロセスとは違い、スレッドは、1 つのアプリケーション内で、同じメモリ空間を共有して動作する。【図 2】

2-2. シングルスレッドとマルチスレッド

プログラムは、「順次処理」「分岐処理」「繰り返し処理」の組み合わせで構成されており、通常のアプリケーションでは、これらが一つずつ順番に処理されるのが一般的である。このようなアプリケーションの処理を「シングルスレッド」という。

これに対し、複数の処理を並行して行うアプリケーションも作成できる。このようなアプリケーションの処理を「マルチスレッド」といい、プログラムのコードが同時に複数個実行される。【図 3】

アプリケーションをすべてマルチスレッドにすれば、アプリケーションの処理速度が速くなるように思えるかもしれない。しかし実際は、そうならない。なぜならば、CPU は通常、1 度に 1 つの処理しか実行できないからである。マルチスレッドアプリケーションは、確かに複数の処理を同時に実行しているように見えるが、それは CPU が複数の処理を高速に切り替えて実行しているだけである。【図 4】

つまり、複数処理をマルチスレッドにしても全体の処理時間は変わらない。むしろ、CPU を切り替える時間分だけオーバーヘッドがかかるため、遅くなる場合もある。

（ただし、現在のコンピュータで使用される CPU は、マルチコアが主流のた

図1

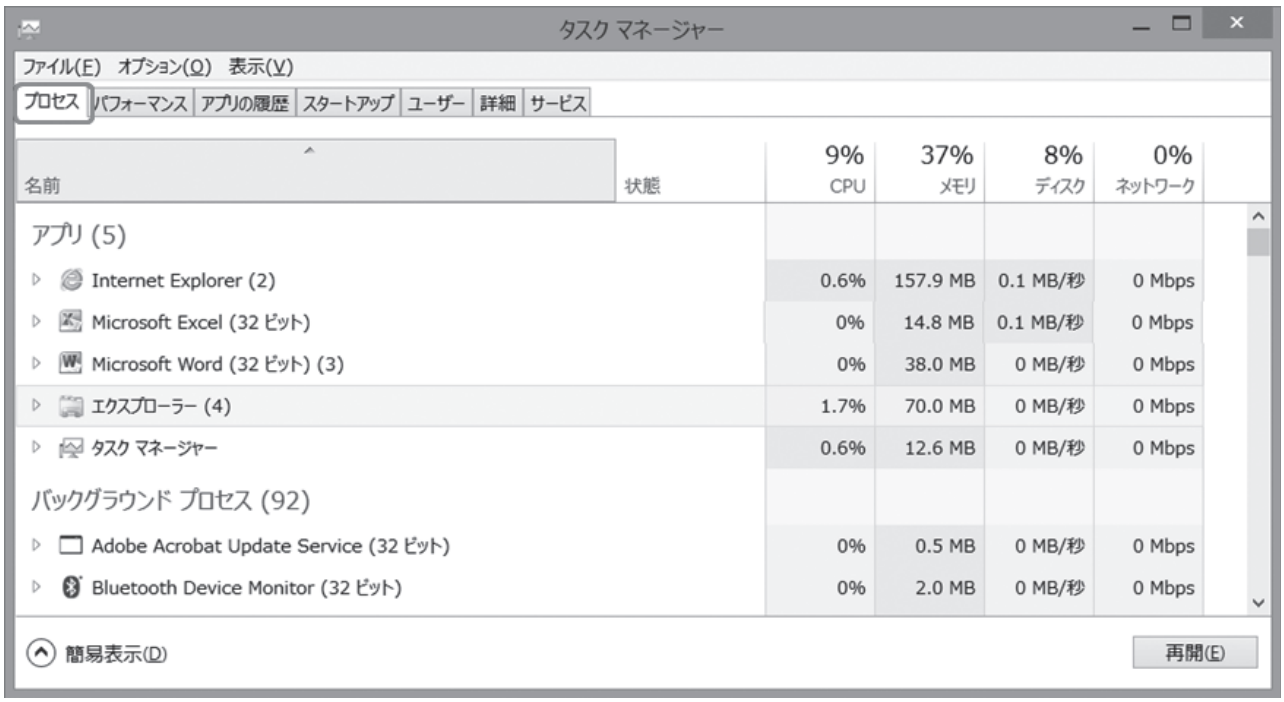
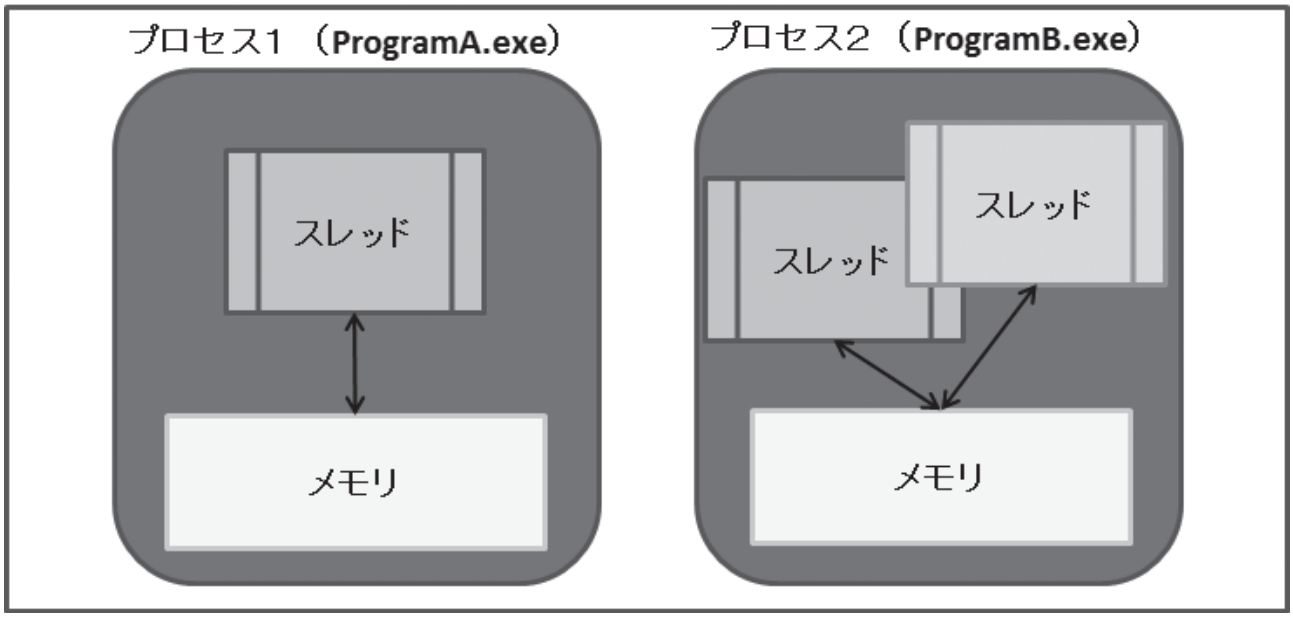

タスク マネージャー
ファイル(F) オプション(O) 表示(V)
プロセス パフォーマンス アプリの履歴 スタートアップ ユーザー 詳細 サービス
名前
状態
9% CPU
37% メモリ
8% ディスク
0% ネットワーク
アプリ (5)
Internet Explorer (2) 0.6% 157.9 MB 0.1 MB/秒 0 Mbps
Microsoft Excel (32 ビット) 0% 14.8 MB 0.1 MB/秒 0 Mbps
Microsoft Word (32 ビット) (3) 0% 38.0 MB 0 MB/秒 0 Mbps
エクスプローラー (4) 1.7% 70.0 MB 0 MB/秒 0 Mbps
タスク マネージャー 0.6% 12.6 MB 0 MB/秒 0 Mbps
バックグラウンド プロセス (92)
Adobe Acrobat Update Service (32 ビット) 0% 0.5 MB 0 MB/秒 0 Mbps
Bluetooth Device Monitor (32 ビット) 0% 2.0 MB 0 MB/秒 0 Mbps
簡易表示(D)
再開(E)

図2


プロセス1 （ProgramA.exe）
スレッド
メモリ
プロセス2 （ProgramB.exe）
スレッド
スレッド
メモリ

め、マルチスレッド化により、CPU の処理が分散される効果は期待できる。）

2-3. マルチスレッドの利点

マルチスレッドの利点について考察してみる。一番大きな利点としては、「重い処理」を実行した時の「レスポンスタイム（応答時間）」の改善である。「レスポンスタイム」とは、処理を実行してから最初の反応が返ってくるまでの時間のことだ。

ここで、シングルスレッドで重い処理を実行するサンプルプログラムを考えてみたい。このプログラムでは、フォーム上に、ビジュアルコンポーネントとして、TButton、TMemo、TStringGrid を配置し、データベースにアクセスするために TSQLConnection、TSQLQuery を配置している。【図 5】

また、「データ取得」ボタン（btnGetData）のクリックイベントは、SQLQuery1 からデータを全件取得し、StringGrid1 に内容を書き出す処理である。【ソース 1】

このプログラムで数万件以上の抽出データを用意して、実行した場合、しばらくの間、画面がすべて固まってしまう（Memo1 に値を入力することもできない）。そして、全件抽出処理が終了して初めて、画面に応答がある。このように処理に時間がかかると、シングルスレッドのアプリケーションは、画面の応答が止まってしまうのである。

次に、このような現象を回避し、レスポンスタイムを改善する方法として、スレッドの活用を考えてみる。考え方としては、重い処理の部分を別のスレッド（サブスレッド）として実行できるようにすればよい。メインスレッドは、サブスレッドが開始したら、そのまま処理を終了する。【図 6】

この方法によって、メインスレッドで処理を実行した後、重い処理はサブスレッドで処理されるため、アプリケーションの画面は応答が止まることなく使用できる。

このようにマルチスレッドの利点は、重い処理で処理時間がかかる時に、レスポンスタイムを格段に向上させられることである。

次節から、具体的な開発方法を紹介する。

3.TThread（スレッドクラス）について

3-1. TThread クラス作成方法

Delphi/400 では、マルチスレッド処理を簡単に実装するために TThread クラスを用意している。プロジェクトに TThread クラスを追加する手順は、次の通りである。

プロジェクトファイルを開いている状態で、［ファイル］→［新規作成］→［その他］を選択し、表示される新規作成ダイアログで［Delphi ファイル］→［スレッドオブジェクト］を選択する。そして、スレッドオブジェクトの新規作成ダイアログで、これから作成するスレッドクラス名を入力して［OK］ボタンを押下する。【図 7】

これで、新しいスレッドクラスを持つユニットが新規に作成できる。【ソース 2】

生成されたスレッドクラスには、Execute メソッドが定義されているので、このメソッドの実装部に、スレッドとして実行したい処理を記述すればよい。たとえば、先ほどのシングルスレッドで記述した【ソース 1】の処理をスレッドクラスに移行すると、【ソース 3】のような実装になる。

3-2. メインスレッドからの呼出し方法

スレッドクラスを作成したら、このスレッドをメインスレッドから呼び出す必要がある。呼び出し方は簡単で単純にスレッドオブジェクトを生成するだけだ。シングルスレッドで記述した【ソース 1】のボタンクリックイベントを、先ほどのスレッドを生成するロジックに変更すればよい。【ソース 4】

変更が完了したら、アプリケーションを実行して確認する。シングルスレッドの場合と異なり、「データ取得」ボタンを押下後、すぐに画面応答ができることがわかる。

（Memo1 に即座に値を入力できる。）

このようにスレッドクラスを用意してメインスレッドからスレッドオブジェクトを生成するだけで、マルチスレッドプログラムが開発できる。

なお、【ソース 3】で作成したプログラムは、スレッドクラス（TGetDataThread）の中で、直接 Form1 を参照していることがわかる。このままでは、同じスレッド処理を別のフォームからも使用したいとなった時に具合が悪い。どうすれば汎用的になるかというと、画面操作に必要な VCL コンポーネントを、スレッドクラスのコンストラクターで受け渡しできるようにすればよい。VCL コンポーネントの受け渡しを加えたソースを【ソース 5】に示す。

【ソース 5】では、メインフォームで使用していた SQLQuery1、StringGrid1 を受け渡しできるように、コンストラクターに 2 つの引数を追加している。受け取った引数をスレッドクラスのプライベート変数に代入し、スレッド内部ではその変数を使って処理を行うようにしている。こうすることで、スレッドクラスはフォームの依存がなくなるため、より独立性の高いプログラムにすることができる。なお、コンストラクターの実装部で、inherited によって、TThread の Create メソッドを呼び出しているが、この時の引数 False は、スレッドが生成後ただちに実行されることを表している。また、FreeOnTerminate プロパティを True に指定しているが、これはスレッド処理が終了した時に、スレッドオブジェクトが自動的に破棄されるようにする設定である。

メインスレッドの呼出し側は、スレッドに渡したい VCL コンポーネントを指定する。【ソース 6】

これで再度アプリケーションを実行すると、先ほどと同じ動作となることが確認できる。スレッドクラスにおいて、フォームの依存性をなくしたため、たとえば別のフォームで異なる SQLQuery を使用した画面においても、同じスレッドクラスが使用できる。

4.スレッド使用時の留意点

4-1. マルチスレッドアプリケーションの留意点

前節で作成したプログラムは、マルチスレッドアプリケーションであるが、実は 2 つの留意点がある。

アプリケーションを実行させ、「データ取得」ボタンを押下し、サブスレッドが動いている間に、アプリケーションを［×］ボタンで終了すると、実行時エラー

図3

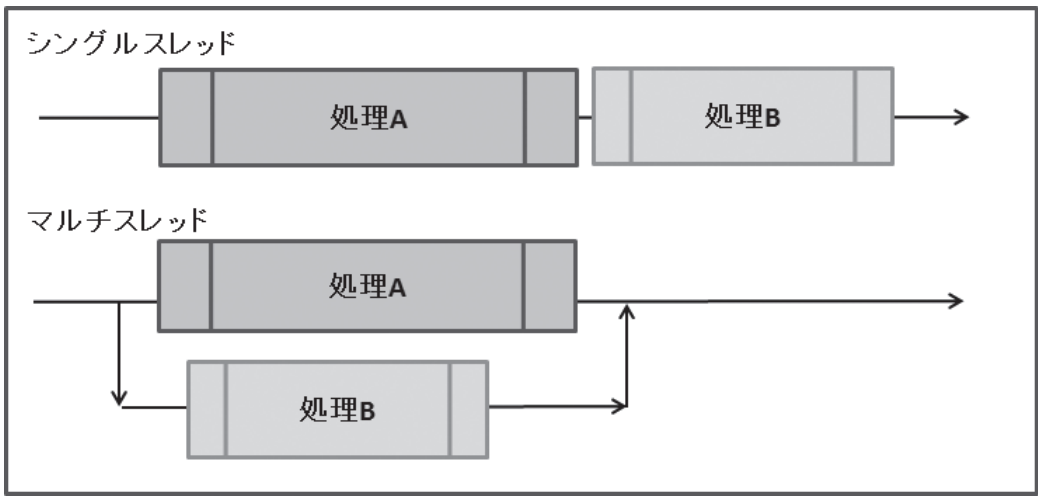

シングルスレッド
処理A
処理B
マルチスレッド
処理A
処理B

図4

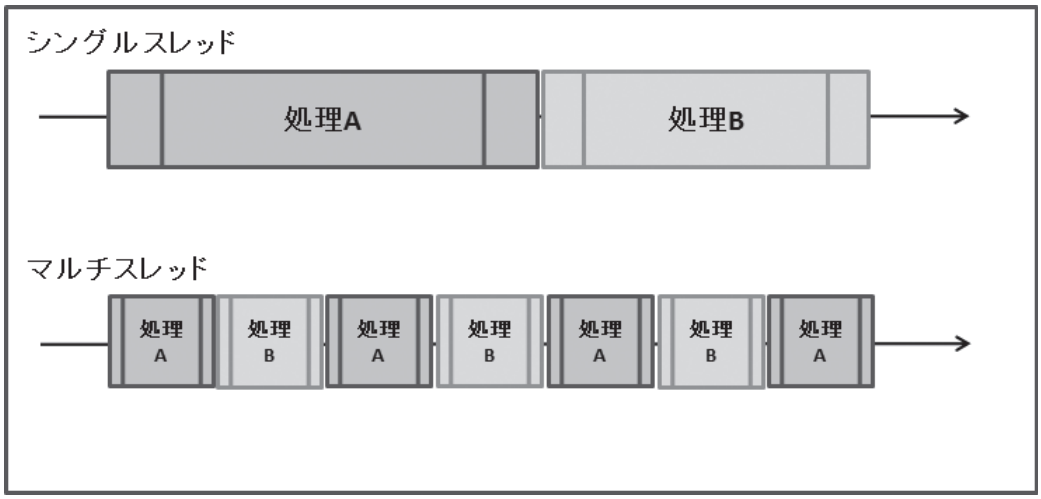
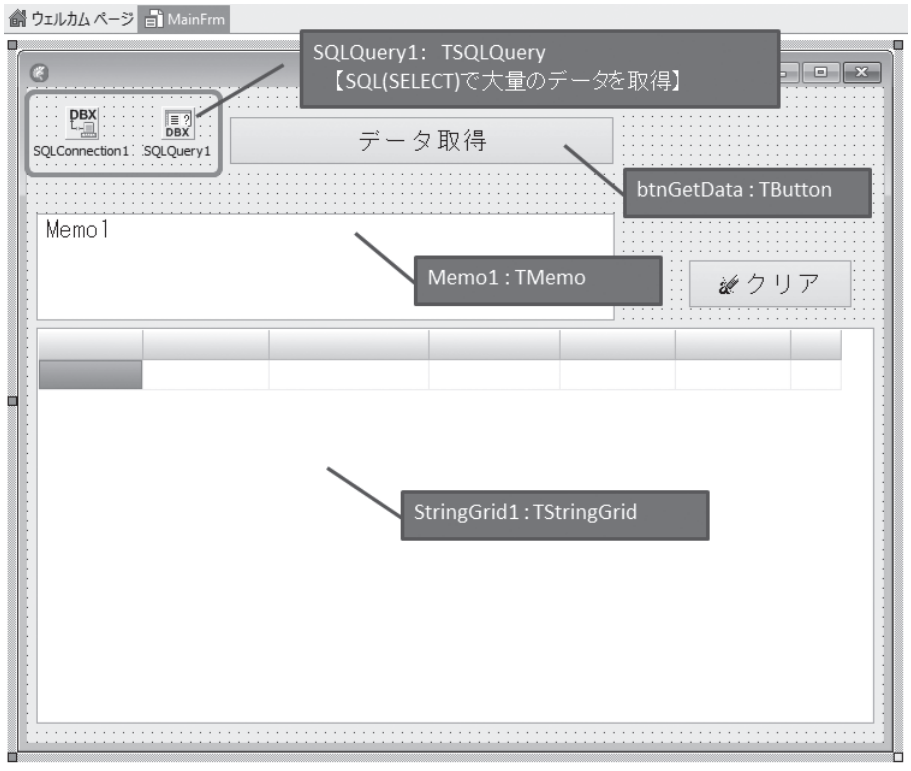

シングルスレッド
処理A
処理B
マルチスレッド
処理 A
処理 B
処理 A
処理 B
処理 A
処理 B
処理 A

図5


ウェルカムページ
MainFrm
SQLQuery1: TSQLQuery
【SQL(SELECT)で大量のデータを取得】
SQLConnection1
SQLQuery1
データ取得
btnGetData : TButton
Memo1
Memo1 : TMemo
クリア
StringGrid1 : TStringGrid

が発生してしまう。【図8】

なぜエラーが起こるかというと、問題はサブスレッドの処理にある。Delphi/400アプリケーションでは、VCLコンポーネント（ビジュアルコンポーネント）をサブスレッドの中で直接操作することができない。つまり、VCLコンポーネントは必ずメインスレッド側で操作する必要がある。サブスレッド側でVCLコンポーネントを使用したい場合には、いったんメインスレッドを一時停止させ、サブスレッドのVCLコンポーネント操作をメインスレッド側に割り込ませる必要がある。これが、マルチスレッドアプリケーションを構築する際の1つ目の留意点である。【図9】

2つ目の留意点は、スレッド内の「繰り返し処理」実行時に、いつでも処理が中断できるようにスレッドの終了確認を行う必要があるということだ。つまり、繰り返し処理の中で、適宜スレッドの終了通知が出ているかどうかを確認し、スレッド外部から終了通知が出されたら、いつでもスレッド処理を終了できるような処理にする必要がある。

次節では、これらの留意点に関する具体的な解決手順を紹介したい。

4-2. Synchronizeの使用方法

1つ目の留意点であるVCLコンポーネントの操作だが、これは、Synchronizeメソッドの使用により対応可能である。Synchronizeメソッドとは、サブスレッド処理側から、メインスレッド側に制御を移し、VCLコンポーネントの操作等を行う特定の手続きを実行させるものである。具体的には、VCLコンポーネントの操作を行う手続き（procedure）をスレッドクラスに作成し、Executeメソッドの中で、Synchronizeメソッド経由して作成した手続きを呼び出せばよい。

先ほどの【ソース5】を改良し、Executeメソッドの中にSynchronizeメソッドを追加したものを【ソース7】に示す。

Synchronizeメソッド経由でVCLコンポーネントを操作する手続きを呼び出すと、その手続きが実行されている間、元のメインスレッド側処理は待機状態になる。

この仕組みでアプリケーションを実行すると、先ほどと同じように「データ取得」ボタンを押下し、サブスレッドが動いている間に、アプリケーションを［×］ボタンで終了しても、正しくアプリケーションを終了できる。

このようにSynchronizeメソッドを使用することで、サブスレッド側からVCLコンポーネントを操作できるが、このVCL操作の手続きで時間がかかってはいけないので注意が必要である。その間メインスレッドの停止状態が続くため、【ソース8】に示すようなプログラムは作成してはならない。

4-3. スレッド中断方法

2つ目の留意点であるスレッドの終了確認だが、これは、「繰り返し処理」における条件において、Terminatedプロパティをチェックすればよい。これを入れることにより、処理を中断させたい時に安全にスレッドを終了できる。Executeメソッドの中にTerminatedプロパティのチェックを入れたものを【ソース9】に示す。

このようなスレッド処理にしておくと、重い処理の実行時に、途中で中断するようなことも行えるようになる。たとえば、今回のサンプルプログラムに処理の中断機能を追加してみる。まずフォーム上に「中止」ボタン（btnAbort）を配置する。【図10】

次にスレッドオブジェクトを扱う変数を宣言部に追加し、「データ取得」ボタンのクリックイベントにて、スレッド生成時に変数に代入するように変更する。そうしておくことで、別のイベントでスレッドオブジェクトの操作が可能になる。

「中止」ボタンのクリックイベントでは、スレッドオブジェクト変数に対し、Terminateメソッドを実行するだけでよい。改良したプログラムを【ソース10】に示す。

このプログラムを実行すると、「データ取得」ボタン押下し、サブスレッドが動いている間に、「中止」ボタンを押下すると、すぐにスレッドが中断されることがわかる。

このようにマルチスレッドアプリケーションとして実装すると、中断処理も容易に実装できるので、時間がかかる処理を開発する際には非常に有効である。

5.CreateAnonymousThreadを使用したスレッド

5-1. シンプルなスレッドの利用

前節までが、TThreadクラスの使用方法である。Delphi/400では、このTThreadクラスを使用することでマルチスレッドアプリケーションが作成できるが、スレッド実行したい処理ごとにスレッドクラスを作成しなければならないため、実装に手間がかかる。汎用的なスレッドクラスであればこの形がよいが、たとえばある画面の一部だけスレッドを使用したい場合にも、都度スレッドクラスを生成するのは少々面倒である。

実は、Delphi/400 XE以降であれば、もっとシンプルにマルチスレッドアプリケーションを作成できる。

Delphi/400 XE以降では、CreateAnonymousThreadメソッドが用意されているため、このメソッドを使用すると、メインスレッドの中に直接、サブスレッドを無名メソッドとして記述できる。【図11】

本稿の初めに作成したシングルスレッドアプリケーションの【ソース1】を基にCreateAnonymousThreadメソッドを使用するように改良したプログラムが【ソース11】である。

【ソース1】と【ソース11】を比べると、CreateAnonymousThreadで処理を括っている部分以外、ほとんど変わらない。

このようにマルチスレッドアプリケーションをシングルスレッドアプリケーション同様に一つの手続きに集約できるため、単純なプログラムとして記述することができる。

ただし、このプログラムも考慮点はある。サブスレッド中で直接VCLコンポーネントを操作しているため、TThreadクラスの場合と同様、VCLコンポーネントはメインスレッドで操作しなければならない。

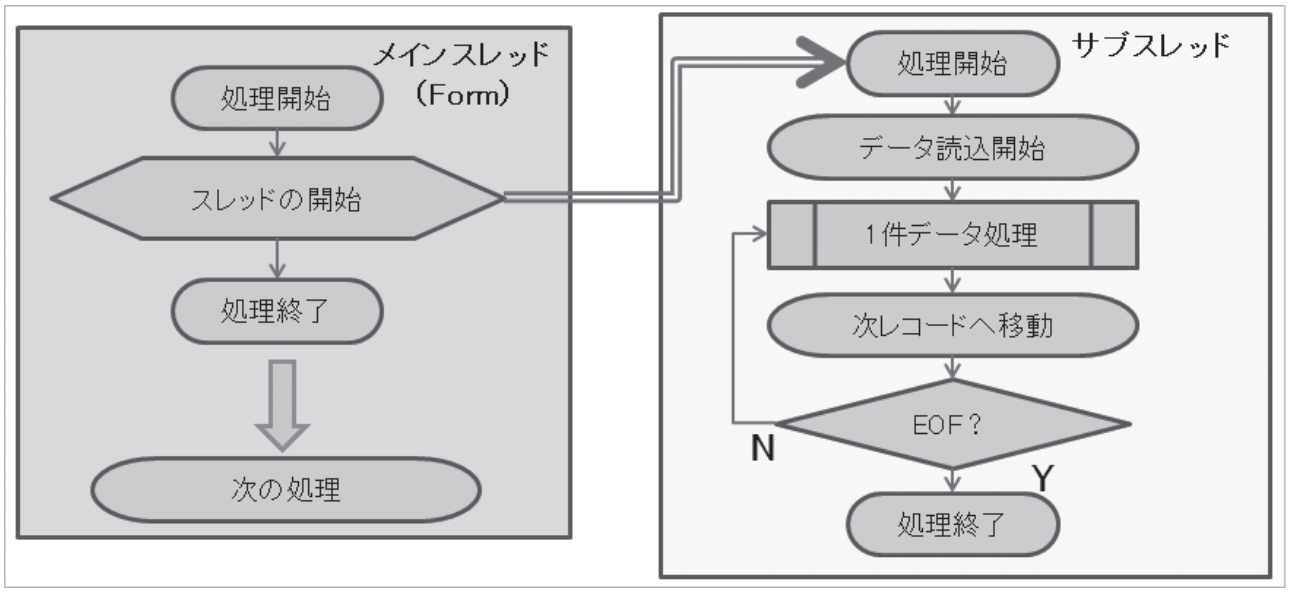
5-2. Synchronizeの使用方法

実は、Synchronizeメソッドも無名メソッドを使用することで、よりシンプルに記述できる。【図12】

ソース1

```
procedure TForm1.btnGetDataClick(Sender: TObject);
var
  i, iRow: Integer;
begin
  //初期化
  iRow := 0;

  SQLQuery1.Active := True;
  try
    //繰り返し
    while (not SQLQuery1.Eof) do
    begin
      Inc(iRow);   //カウントアップ
      StringGrid1.RowCount := iRow + 1;
      //グリッドにデータを書き出す
      for i := 0 to SQLQuery1.FieldCount - 1 do
        StringGrid1.Cells[i, iRow] := SQLQuery1.Fields[i].Text;
      SQLQuery1.Next;
    end;
  finally
    SQLQuery1.Active := False;
  end;
end;
```

図6



この方法を使用すると、安全なマルチスレッド処理を一つのサブルーチンにまとめられる。【ソース 11】を改良したプログラムを【ソース 12】に示す。

このように安全なマルチスレッドアプリケーションを一つのイベントの中にまとめて書けるため、TThread クラスを別途作成しなくとも、簡単にアプリケーションのマルチスレッド化が可能となる。

開発時の使い分けとしては、汎用的なスレッド処理や、実行中の中断処理などを含むスレッドは、TThread クラスを使用し、特定の場面だけで使用するスレッドは、CreateAnonymousThread メソッドで対応することが望ましい。

6.まとめ

本稿では、マルチスレッドを使用してレスポンスタイムを向上させる手法として、TThread クラスおよび CreateAnonymousThread メソッドについて説明してきた。

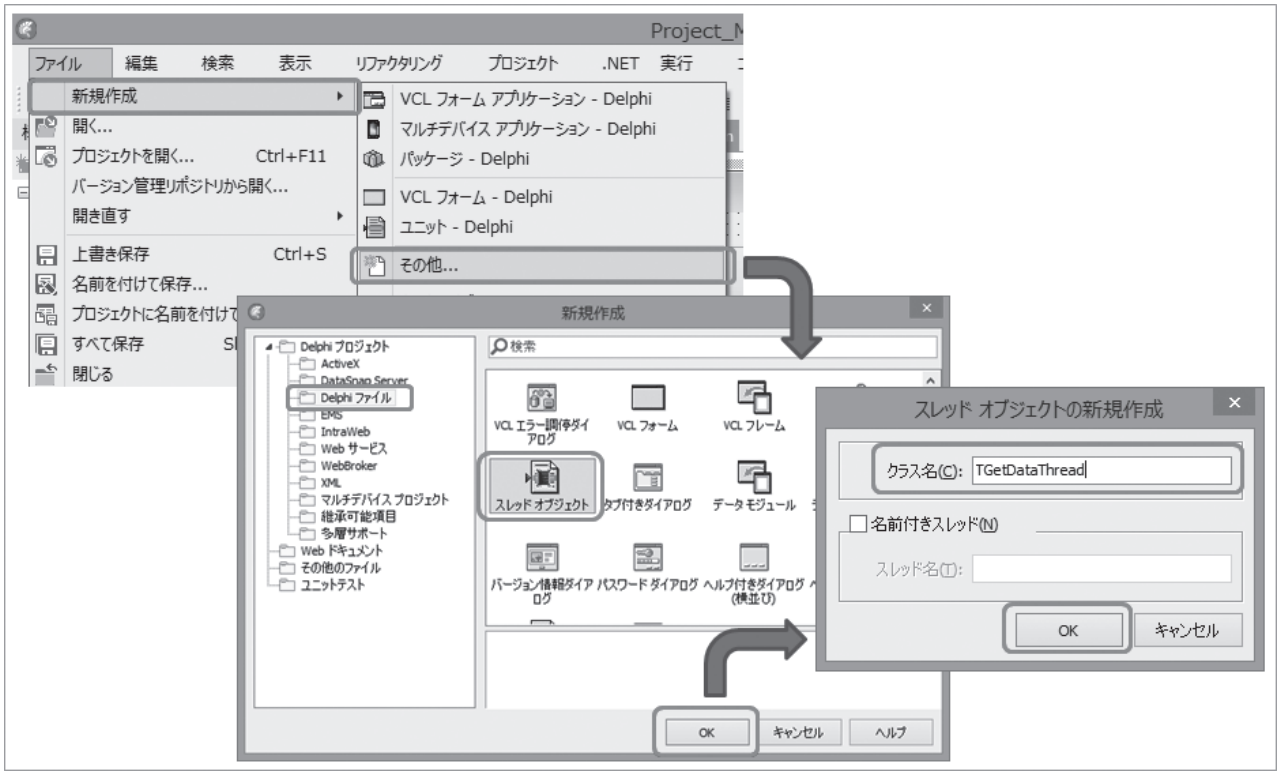
冒頭でも述べた通り、スレッドを使用しても、全体の処理時間が短縮するわけではないが、応答時間が早いと、ユーザーはアプリケーションを快適に使用できる。

またスレッド処理を開発する場合、メインスレッドとサブスレッド間で VCL コンポーネント操作の競合や、Synchronize メソッド内の処理時間など、考慮点がいくつかある。

しかし、ユーザーにとって使いやすいアプリケーションを実現するという意味で、マルチスレッド開発は非常に有用であり、手間をかける価値が十分にある技術と言える。

M

図7



ソース2

```
unit Unit1;

interface

uses
  System.Classes;

type
  TGetDataThread = class(TThread)
  private
    { Private 宣言 }
  protected
    procedure Execute; override;
  end;

implementation


{ TGetDataThread }

procedure TGetDataThread.Execute;
begin
  { スレッドとして実行したいコードをここに記述してください }
end;

end.
```

ソース3

```
unit ThreadUnit;

interface

uses
  System.Classes;

type
  TGetDataThread = class(TThread)
  private
    { Private 宣言 }
  protected
    procedure Execute; override;
  end;

implementation

uses MainFrm;  //---- メインフォームを参照

{ TGetDataThread }

procedure TGetDataThread.Execute;
var
  i, iRow: Integer;
begin
  //初期化
  iRow := 0;

  //メインフォームを使用
  with Form1 do
  begin
    SQLQuery1.Active := True;
    try
      //繰り返し
      while (not SQLQuery1.Eof) do
      begin
        Inc(iRow);   //カウントアップ
        StringGrid1.RowCount := iRow + 1;
        //グリッドにデータを書き出す
        for i := 0 to SQLQuery1.FieldCount - 1 do
          StringGrid1.Cells[i, iRow] := SQLQuery1.Fields[i].Text;
        SQLQuery1.Next;
      end;
    finally
      SQLQuery1.Active := False;
    end;
  end;
end;

end.
```

フォーム側のSQLQuery1
StringGrid1を操作

ソース4

```
implementation

{$R *.dfm}

uses ThreadUnit;  //スレッドユニットを追加


procedure TForm1.btnGetDataClick(Sender: TObject);
begin
  //スレッドの生成
  TGetDataThread.Create;
end;
```

ソース5

```
unit ThreadUnit;

interface

uses
  System.Classes, vcl.Grids, Data.SqlExpr;

type
  TGetDataThread = class(TThread)
  private
    { Private 宣言 }
    FStringGrid: TStringGrid;
    FQuery: TSQLQuery;
  protected
    procedure Execute; override;
  public
    constructor Create(AStringGrid: TStringGrid; AQuery: TSQLQuery); virtual;
  end;

implementation

{ TGetDataThread }

constructor TGetDataThread.Create(AStringGrid: TStringGrid; AQuery: TSQLQuery);
begin
  FStringGrid := AStringGrid;
  FQuery := AQuery;

  inherited Create(False);      //False指定でスレッド生成時、即実行
  FreeOnTerminate := True;      //スレッド終了時にオブジェクト破棄
end;

procedure TGetDataThread.Execute;
var
  i, iRow: Integer;
begin
  //初期化
  iRow := 0;

  FQuery.Active := True;
  try
    //繰り返し
    while (not FQuery.Eof) do
    begin
      Inc(iRow);   //カウントアップ
      FStringGrid.RowCount := iRow + 1;
      //グリッドにデータを書き出す
      for i := 0 to FQuery.FieldCount - 1 do
        FStringGrid.Cells[i, iRow] := FQuery.Fields[i].Text;
      FQuery.Next;
    end;
  finally
    FQuery.Active := False;
  end;
end;

end.
```

TStringGrid、TSQLQueryを使用する為にユニットを追加

スレッド側で使用するVCLコンポーネントを変数として定義

メインフォーム側から、VCLコンポーネントがセットできるよう、コンストラクターの引数を追加

メインフォームに依存しないロジックに修正

ソース6

```
procedure TForm1.btnGetDataClick(Sender: TObject);
begin
  //スレッド生成
  TGetDataThread.Create(StringGrid1, SQLQuery1);
end;
```

図8

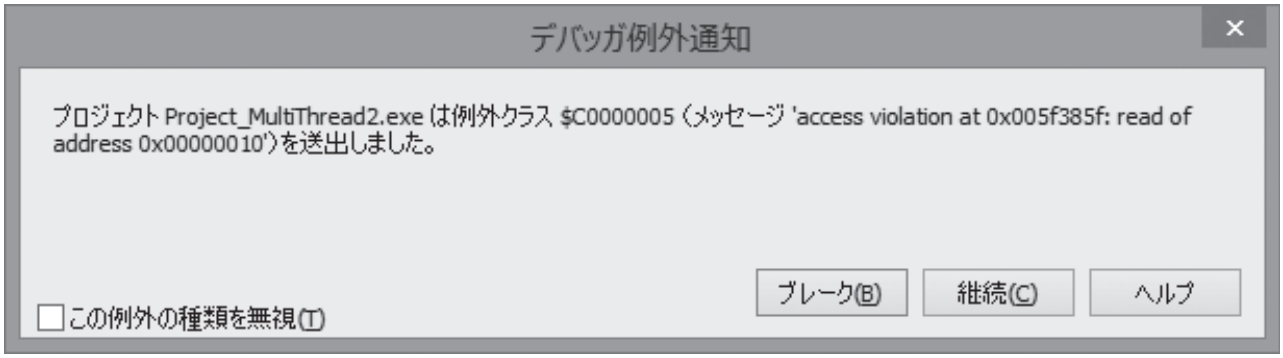
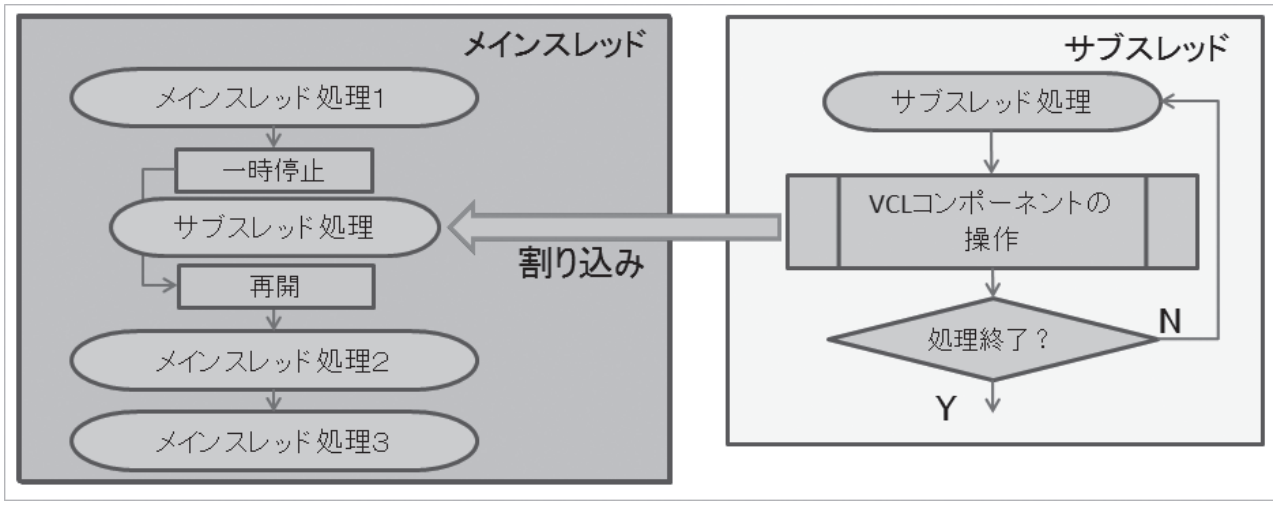


図9



ソース7

```
unit ThreadUnit;

interface

uses
  System.Classes, vcl.Grids, Data.SqlExpr;

type
  TGetDataThread = class(TThread)
  private
    { Private 宣言 }
    FStringGrid: TStringGrid;
    FQuery: TSQLQuery;
    FRow: Integer;          //現在処理行を保持するグローバル変数
  protected
    procedure Execute; override;
    procedure VCLDraw;              //VCL操作を行う手続き
  public
    constructor Create(AStringGrid: TStringGrid; AQuery: TSQLQuery); virtual;
  end;

implementation

{ TGetDataThread }

procedure TGetDataThread.Execute;
begin
  //初期化
  FRow := 0;

  FQuery.Active := True;
  try
    //繰り返し
    while (not FQuery.Eof) do
    begin
      Inc(FRow);   //カウントアップ

      Synchronize(VCLDraw); //メインスレッドを待機させて処理実行

      FQuery.Next;
    end;
  finally
    FQuery.Active := False;
  end;
end;

procedure TGetDataThread.VCLDraw;
var
  i: Integer;
begin
  FStringGrid.RowCount := FRow + 1;
  //グリッドにデータを書き出す
  for i := 0 to FQuery.FieldCount - 1 do
    FStringGrid.Cells[i, FRow] := FQuery.Fields[i].Text;
end;

end.
```

Executeメソッドと、VCLDrawメソッドの両方で行番号変数を使用する為、グローバル変数として定義

VCLコンポーネントを操作する処理を手続きとして宣言

Synchronizeメソッドを経由してVCLDraw手続きを実行

VCLDraw手続き実行中、メインスレッドは一時停止となる。

VCLコンポーネント(StringGrid)を操作する処理を記述

ソース8

```
procedure TGetDataThread.Execute;
begin
  //初期化
  FRow := 0;
  Synchronize(VCLDraw); //メインスレッドを待機させて処理実行

end;

procedure TGetDataThread.VCLDraw;
var
  i: Integer;
begin
  FQuery.Active := True;
  try
    //繰り返し
    while (not FQuery.Eof) do
    begin
      Inc(FRow);   //カウントアップ
      FStringGrid.RowCount := FRow + 1;
      //グリッドにデータを書き出す
      for i := 0 to FQuery.FieldCount - 1 do
        FStringGrid.Cells[i, FRow] := FQuery.Fields[i].Text;

      FQuery.Next;
    end;
  finally
    FQuery.Active := False;
  end;
end;

end.
```

繰り返しが行われる為、メインスレッドが待機状態のままとなってしまう。

ソース9

```
procedure TGetDataThread.Execute;
begin
  //初期化
  FRow := 0;

  FQuery.Active := True;
  try
    //繰り返し
    while (not FQuery.Eof) and (not Terminated) do
    begin
      Inc(FRow);   //カウントアップ

      Synchronize(VCLDraw); //メインスレッドを待機させて処理実行

      FQuery.Next;
    end;
  finally
    FQuery.Active := False;
  end;
end;
```

終了通知(Terminated)がFalseの場合のみ、処理を継続する

図10

ソース10

```
unit MainFrm;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, DBXDynalink, FMTBcd, DB,
  SqlExpr, StdCtrls, Buttons, Grids, Mask, DBClient, ThreadUnit;

type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    btnGetData: TBitBtn;
    SQLConnection1: TSQLConnection;
    SQLQuery1: TSQLQuery;
    btnClear: TBitBtn;
    Memo1: TMemo;
    btnAbort: TBitBtn;
    procedure btnGetDataClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure btnClearClick(Sender: TObject);
    procedure btnAbortClick(Sender: TObject);
  private
    { Private 宣言 }
    GetDataThread: TGetDataThread; // スレッドオブジェクト変数
  public
    { Public 宣言 }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.btnGetDataClick(Sender: TObject);
begin
  //スレッドの生成
  GetDataThread := TGetDataThread.Create(StringGrid1, SQLQuery1);
end;

procedure TForm1.btnAbortClick(Sender: TObject);
begin
  if Assigned(GetDataThread) then
    GetDataThread.Terminate; //スレッド中断
end;
```

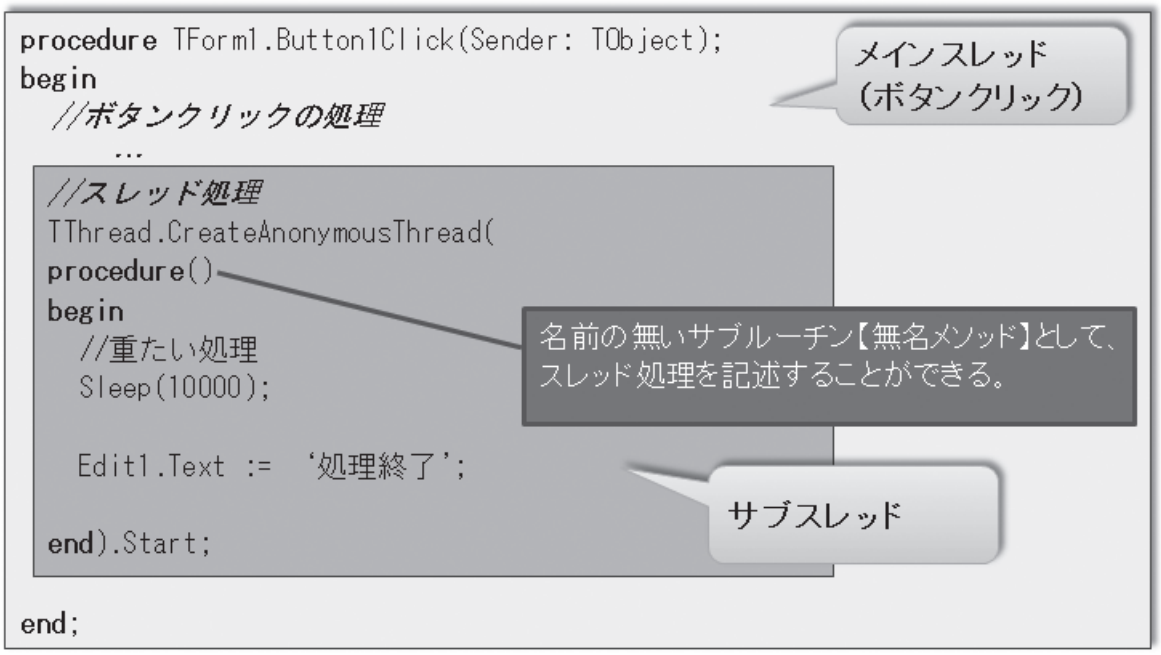
宣言部にスレッドユニットの参照を追加

スレッドオブジェクト変数をグローバル変数として追加

生成したスレッドオブジェクトを変数に代入

スレッドに終了を通知

図11


procedure TForm1.Button1Click(Sender: TObject);
begin
//ボタンクリックの処理
...
//スレッド処理
TThread.CreateAnonymousThread(
procedure()
begin
//重たい処理
Sleep(10000);
Edit1.Text := '処理終了';
end).Start;
end;
メインスレッド
(ボタンクリック)
名前の無いサブルーチン【無名メソッド】として、
スレッド処理を記述することができる。
サブスレッド

ソース11

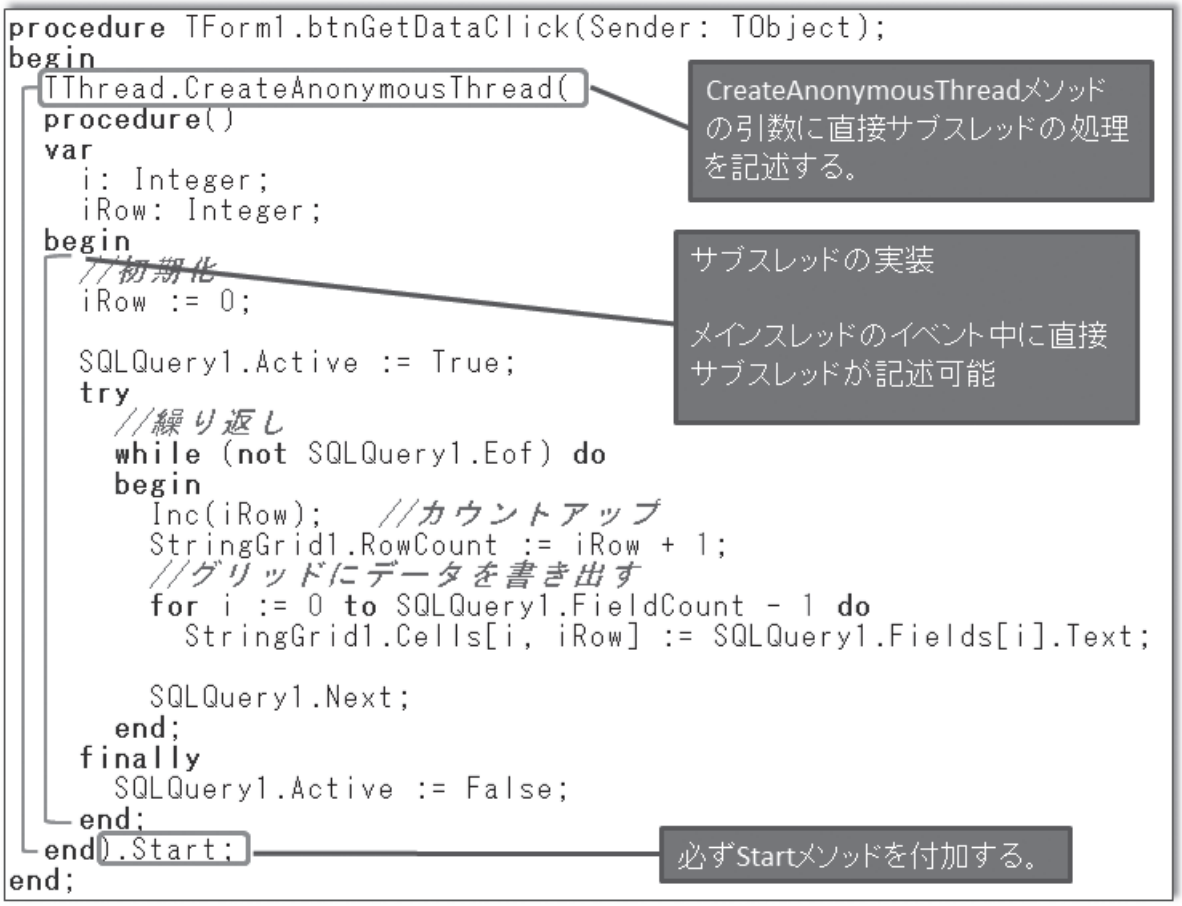
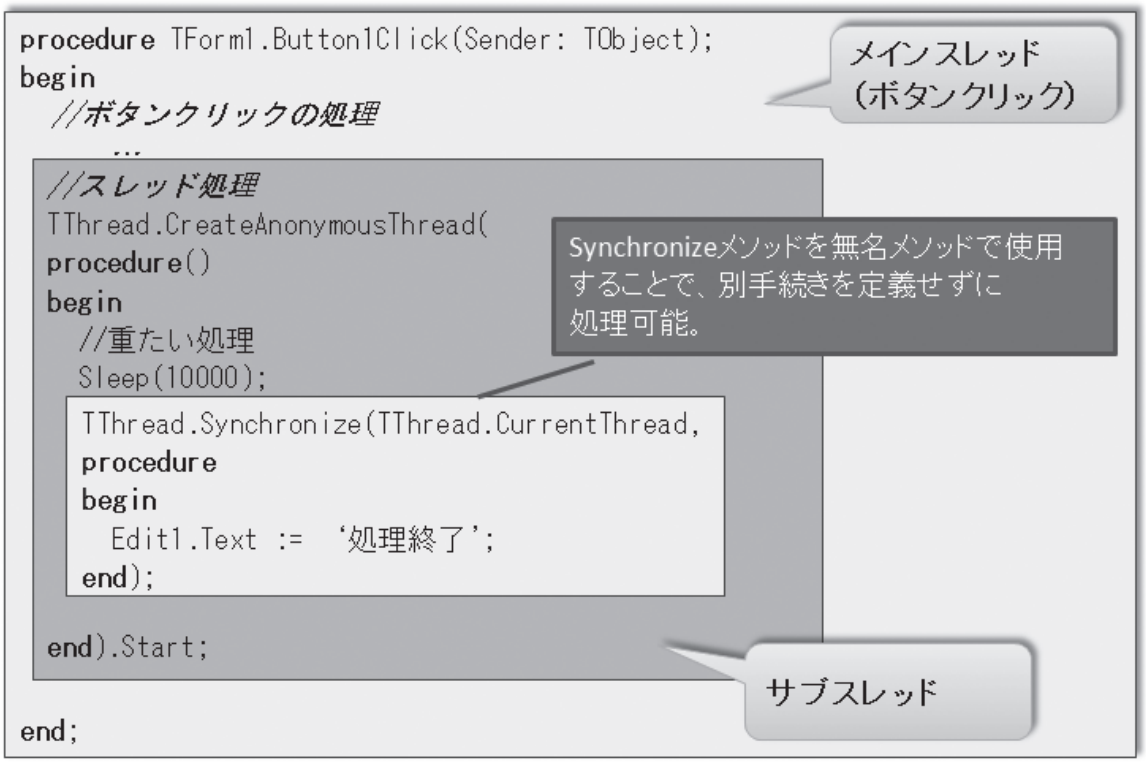

procedure TForm1.btnGetDataClick(Sender: TObject);
begin
TThread.CreateAnonymousThread(
procedure()
var
i: Integer;
iRow: Integer;
begin
//初期化
iRow := 0;
SQLQuery1.Active := True;
try
//繰り返し
while (not SQLQuery1.Eof) do
begin
Inc(iRow); //カウントアップ
StringGrid1.RowCount := iRow + 1;
//グリッドにデータを書き出す
for i := 0 to SQLQuery1.FieldCount - 1 do
StringGrid1.Cells[i, iRow] := SQLQuery1.Fields[i].Text;
SQLQuery1.Next;
end;
finally
SQLQuery1.Active := False;
end;
end).Start;
end;
CreateAnonymousThreadメソッド
の引数に直接サブスレッドの処理
を記述する。
サブスレッドの実装
メインスレッドのイベント中に直接
サブスレッドが記述可能
必ずStartメソッドを付加する。

図12



ソース12

```
procedure TForm1.btnGetDataClick(Sender: TObject);
begin
  TThread.CreateAnonymousThread(
  procedure()
  var
    iRow: Integer;
  begin
    //初期化
    iRow := 0;

    SQLQuery1.Active := True;
    try
      //繰り返し
      while (not SQLQuery1.Eof) do
      begin
        Inc(iRow);   //カウントアップ

        //メインスレッドを待機させて処理実行
        TThread.Synchronize(TThread.CurrentThread,
        procedure
        var
          i: Integer;
        begin
          StringGrid1.RowCount := iRow + 1;
          //グリッドにデータを書き出す
          for i := 0 to SQLQuery1.FieldCount - 1 do
            StringGrid1.Cells[i, iRow] := SQLQuery1.Fields[i].Text;
        end);

        SQLQuery1.Next;
      end;
    finally
      SQLQuery1.Active := False;
    end;
  end).Start;
end;
```

Synchronizeメソッドの引数にVCLコンポーネントを操作する処理を記述する。

VCLコンポーネントの操作をイベント中に直接記述可能

吉原 泰介

株式会社ミガロ.

RAD事業部 技術支援課 顧客サポート

[Delphi/400 XE5／XE7]
AndroidアプリケーションのNFC機能活用

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 件に及ぶ問合せサポートやセミナ講師などを担当している。

1.はじめに

近頃では、物を購入する際に直接お金で支払うのではなく、クレジットカードやおサイフケータイなどを使ってデジタルに決済することが多くなってきた。たとえば、電車などの交通機関では、改札は物理的な切符よりもICカードでの決済のほうが主流になっている。

こうしたICカードの読み取りには、もちろんプログラムによる制御が行われている。この技術は、これまでは限定的な場面や機器で使われていたため、どちらかというと「特殊な技術」と考えられることが多かったが、スマートフォンの急速な普及により、身近なところで活用される機会が増えてきた。また最近ではIoT（Internet of Things）に関する技術が注目されており、NFCはその有効な手段の1つである。

本稿では、こうした背景を踏まえ、ICカードなどの情報をやり取りするためのNFCという通信技術を題材としている。NFCの基本的な情報から、Delphi/400のプログラムで実装する手法までを紹介する。

なお本稿では、スマートデバイスを使ったAndroidの開発が中心となるため、Delphi/400のバージョンはXE5とXE7を対象としている。

2.NFCについて

2-1.NFCとは

ICカードなどを使う際に、多くの場合はNFCと呼ばれる通信技術が使われている。NFCは「Near Field Communication」の略称で、無線通信の国際規格である。十数センチの距離で通信を行える小電力無線通信技術で、最近はスマートフォンやデジタルカメラ、電化製品などで広く採用されている。

使用できる製品には、【図1】のようなロゴマークが付けられていることが多い。NFCでは、ICカードやスマートフォンなどの対応機器を近距離でかざすだけで、電子マネー決済やデータ転送などの情報のやり取りができる。このNFCの特徴は、通信を行う機器間において複雑な操作を必要としない点である。

2-2. 身近で使われるNFC

身近なところでは、先ほど触れた交通機関のICカード（Suica等）などの電子マネーのほかに、おサイフケータイや免許証などにもICチップが搭載され、NFCが活用されている。【図2】

企業では、ビルの入退館管理にICカードを利用したり、工場などでは商品にRFIDなどを付け、製造工程や在庫管理に活用している場合も多い。

2-3. NFCの規格

NFCは、大きく分けて、以下の3つの機能で構成されている。

①リーダライター機能
（NFCタグに読み書きする）
②カードエミュレーション機能
（ICカードで決済する）
③P2P機能
（NFCデバイス同士で通信する）

図1　NFCロゴマーク

図2　NFCが使われるカード等

図3　NFCの規格

図4　Felica対応カード

NFC は、これらの機能に応じて、RFID 関連（①②）と通信関連（③）として規格されている。【図 3】

RFID と は、「Radio Frequency Identification」の略称で、無線通信による認証技術である。IC カードなどは、この RFID の規格に属する。また簡易に読み書きできるものとして、NFC タグと呼ばれる安価なシール形式のタグも普及している。

通信関連は、Bluetooth や Wi-Fi、P2P などの分野で規格されている。本稿では主に RFID を中心に説明していく。

RFID の中にも、いくつかの規格があり、IC カードによっても使われている規格が異なる。

たとえば、TypeA（ISO14443）は、成人識別 IC カード「taspo（タスポ）」で使われていることで有名である。TypeB は、役所関連でよく使用されており、免許証がその代表である。

しかし、日本で一番使われているのは TypeA でも TypeB でもなく、「FeliCa（フェリカ）」である。FeliCa はソニーが開発した独自の無線通信技術規格で、当初 TypeC 認定を目指していたが、認定はまだされていない。

FeliCa は、Suica や ICOCA などの交通カードに代表され、日本の IC カードのほとんどはこの FeliCa が採用されている。【図 4】

FeliCa の特徴は、ソニーが細部まで厳しく規格を決めているため、曖昧な情報が少なく、IC カードへのアクセスが極めて速い点である。

このアクセス速度、反応速度は、NFC の近距離通信では非常に重要である。たとえば、駅の改札で IC カードの反応が悪ければ使い物にならないが、FeliCa の反応速度は 1 秒もかからないため、交通カードのほとんどで FeliCa が採用されている。

2-4. バーコードとの違い

情報を読み取るという技術では、従来から使われているバーコードや QR コードもある。

ここで、NFC とバーコードや QR コードとの違いについて考えてみる。

一番大きな違いとしては、バーコードと QR コードは一方通行の読み取りしかできないのに対して、NFC は IC カードや NFC タグに読み書きができるという点である。

たとえば、バーコードや QR コードで商品を検品する場合、読み取る機器側で情報を収集するが、NFC では読み取りだけでなく商品側の RFID に書き込みもできるので、「検品済み」といったステータスを書き込むこともできる。

またバーコードや QR コードは、一度印刷してしまうと変更できないが、NFC は読み書きできるので、同じ媒体を何度でも使い回せる。読み取り専用の DVD と読み書き可能な DVD の違いと同じである。

操作性では、バーコードや QR コードを使うと読み取り部分にピントを合わせる必要があるが、NFC は近距離でかざすだけで読み取れる。非常に簡単で扱いやすいことも、NFC のメリットの 1 つである。【図 5】

もちろん、すべてがバーコードや QR コードより NFC が優れているわけではない。上記のよう便利さがある半面、バーコードや QR コードは紙媒体に印刷すれば使えるが、NFC は RFID などの IC チップが必要となるため、媒体のコストという点では非常に高くなる。安価な NFC タグであっても 1 枚あたり 100 ～ 200 円かかるため、大量な媒体が必要な場合、読み取りだけならバーコードのほうが採用されやすい。

そのため用途によって、NFC とバーコード、QR コードの使い分けが重要である。

3.NFCの活用

3-1. NFC の活用とアプリケーション

NFC には、バーコードなどと違い、読み書きできる特徴があることを説明したが、実際にどのような用途のアプリケーションに適用できるかを、A・B・C のパターンで分類してみた。

A. IC カードなどの読み取り・書き込み
＜用途例＞

IC カードの情報を読み込んだり、決済ができる。また、前述したように、入退室の認証管理や製品の検品・工程管理といった用途でも使用できる。

＜ NFC を使ったアプリケーション例＞
マルチ残高リーダ【図 6】

・IC カードの電子マネー残高をスマートデバイスでチェックできる。
https：//play.google.com/
（「マルチ残高リーダ」で検索）

B. デバイスの自動設定切替
＜用途例＞

NFC タグを使ってデバイスの設定を自動的に切り替える。機内モードや Wi-Fi の自動設定など、個人用途だけでなく商用施設の入り口などで使用される場合もある。

＜ NFC を使ったアプリケーション例＞
NFC タスクランチャー【図 7】

・自動切替の設定等を NFC タグに保存できる。スマートフォンでその NFC タグをかざせば、設定内容を自動切替する。
https：//play.google.com/
（「NFC タスクランチャー」で検索）

C. アプリやサイトの自動起動・連携
＜用途例＞

NFC タグを使ってアプリケーションや URL を自動起動できる。スマートポスターを使ったマーケティングでは、ポスターに IC チップが埋め込まれており、スマートデバイスをかざすと情報を入手することができる。またアクセス情報は IC チップにも書き込まれ、ポスターの場所ごとに地理的なアクセス情報などの集計・分析に使われる。

またポスター同様に、NFC タグをかざすことで、すぐに twitter と連動するような個人利用のアプリケーションにも使われている。

＜ NFC を使ったソリューション例＞
スマートポスター【図 8】
http：//www.hayato.info/tapee/

＜ NFC を使ったアプリケーション例＞
たっちなう【図 9】

・特定の NFC タグをかざすと twitter を連携できる。
https：//play.google.com/
（「たっちなう」で検索）

図5

QRコード読み込み　　NFCタグ読み込み

図6　マルチ残高リーダー

図7　NFCタスクランチャー

3-2. NFC とスマートデバイス

最近のスマートデバイスでは、NFC は重要な標準機能の 1 つとなっている。スマートデバイスといえば、iPhone ／ iPad の iOS と、Android が主流である。iOS では iPhone 6 で NFC が搭載されるようになったが、残念ながら iOS の NFC 機能は「Apple Pay」に限定されており、ユーザーが開発するアプリケーションで NFC を使う方法は提供されていない。

一方、Android では、NFC が標準搭載された OS 2.3 以降の端末では、ユーザーが NFC を活用したアプリケーションを作成できる。先に取り上げたアプリケーション例も Android である。そのため本稿では、NFC を使う題材として Android を取り上げることにした。

4.Delphi/400からのNFC機能利用

本章では、ここまで説明してきた NFC 機能のうち、IC カードや NFC タグなどの RFID 情報を、Delphi/400 で開発した Android アプリケーションから読み書きする方法を紹介していく。

Delphi/400 で Android の NFC 機能を使う場合は、専用のコンポーネントなどはないため、プログラムで NFC の呼出しを実装する必要がある。下記のサイトでは、Delphi での NFC 実装方法を、XE5 や XE7 のバージョンごとに非常に詳しく説明している。実際に動作するサンプルも用意されている。

Delphi and NFC on Android
（2014 年 9 月 10 日の記事）
http：//blog.blong.com/

ただし、このサイトの記事は英語なので、ここでダウンロードできるサンプルを題材に、NFC の実装に必要なポイントを説明していく。

ダウンロードできるサンプルは、「NFC_Samples_XEX.7z」というファイルである。7z というファイル圧縮形式なので、7-Zip 等の解凍ソフトを準備する必要がある。ファイルを展開すると Delphi のソースが一式揃っており、そのままコンパイルして使うことができる。【図 10A】【図 10B】

4-1. デバイスの設定

本稿では、NFC を標準で実装している Android を対象にするが、NFC を搭載しているからといって無条件に動作するわけではない。機種や OS によって違いはあるが、まずは、デバイスメニューのネットワーク関連の設定で、NFC 通信を有効にしておく必要がある。【図 11】は一例である。これを有効にしておかないと、NFC を搭載していても、RFID などにデバイスが反応しない。

4-2. プロジェクトの設定

次に、プログラムのプロジェクト設定を確認する。

「4-1.」ではデバイス側を設定したが、アプリケーション側も設定が必要となる。Android のプログラム作成時に、［プロジェクト］→［オプション］の［使用する権限］で細かい機能権限を設定できる。

ここで、NFC の機能をチェックして True に設定する【図 12】。これによりアプリケーションで NFC の機能を使用することが許可される。

4-3. XML の定義

NFC のアプリケーションにおいて、プログラムでどのような NFC 情報を扱うかは、XML ファイルを使って定義する。【図 13】

たとえば、スマートフォンに NFC 情報をかざしてアプリケーションを起動するには、対象となる NFC 情報を決めておく必要がある。この情報には AndroidManifest.xml、nfc.tec-discovered.filter.xml という 2 つの XML を使用する。

サンプルプログラムを確認すると、［プロジェクト］→［配置］で AndroidManifest.xml ファイルが組み込まれているのがわかる【図 14】。この xml ファイルはプロジェクトと同じフォルダに存在する。AndroidManifest.xml ファイルを Delphi 上で開くと、中身を確認できる（テキストエディタでも可能）。

ポイントは大きく 2 つある。

1 つは、「android.hardware.nfc」という設定を True で設定しておくことである【図 15】。これは、NFC のデバイス（ハードウェア）機能を要求するという設定になる。

もう 1 つは、「ACTION_NDEF_DISCOVERED」という設定である。これは、アプリケーションで扱いたい NFC のデータタイプを検知したら、アプリケーションを起動できるようにするフィルタ（intent-filter と呼ぶ）の種類の設定である。これによって使用する NFC のタイプを特定し、必要な NFC 情報を受信した場合だけ、アプリケーションでデータを読み込むことができる【図 16】。また、先に触れた NFC の規格にある TypeA や TypeB、FeliCa など対象のタイプを細かく定義することもできる。その場合、res\xml フォルダに「nfc.tec-discovered.filter.xml」という xml を用意し、対象となる規格リストを記述しておく。【図 17】

こうした細かい設定をしておくと、たとえば FeliCa だけに反応するアプリケーションを開発できる。

4-4. JNI の組み込み

実際に、Android プログラムから NFC の機能を使用する場合は、JNI（Java Native Interface）経由で Android SDK ／ API の android.nfc.NfcAdapter を利用する必要がある。

JNI は、Java で作成されたプログラムを他の言語から呼び出すためのインターフェース仕様である。しかし、これは名前の通り、Java モジュールであるため、Delphi には標準で用意されていない。

ただし、今回のサンプルでは、この JNI を利用するためのラッパーソース（Androidapi.JNI.Nfc.pas）が Common フォルダに用意されている。これを利用すると、Delphi から JNI の機能を簡単に呼び出すことができる。【図 18】

また、NFCHelper.pas というユニットも用意されており、NFC を扱う際に非常に便利である。このユニットには、NFC に対する Read や Write などのメソッドなどが定義されている。

4-5. NFC タグの読み込み実装

NFC の読み込みは、NFCHelper の HandleNfcTag を使って取得できる。【図 19】

NFC の読み書きをテストする場合、「2.NFC について」でも説明した、簡易

図8　スマートポスターのマーケティング

図9　たっちなう

な読み書きが可能なNFCタグと呼ばれるタグシールを使うと便利である。サンプルでは、スキャンからの起動時（OnFormActivateイベント）に取得している。【ソース1】

新規のNFCタグを読み込んだ場合はEmptyが表示されるが、書き込み情報がある場合、テキストに出力できる。たとえば免許証などを読み込むとTypeBとして表示されるので、正しく読み込めていることがわかる【図19】。ただし、免許証や交通カードなど秘匿性があるデータは、単純には取得できないようになっている。

4-6. NFCタグの書き込み実装

NFCの書き込みは、HelperのWriteTagTextを使って実装できる【図20】。サンプルでは、「WriteTag」ボタンの押下時（OnClickイベント）に書き込みをしている【ソース2】。このようにNFCHelper.pasを利用すると、NFCに対する実装が非常に簡単に行える。

これで、Delphi/400を使ったNFCに対する基本的な読み書きの実装方法が確認できた。

今回利用したNFCHelper.pasには、StringToJString等のJavaのデータと互換性をもつ関数も用意されているので、一通り機能を確認してから利用するとよいだろう（無駄にプログラムを作成しなくて済む）。

5.Android以外でのNFC利用（補足）

5-1. Windowsでの NFC利用

本稿では、Androidを使ったプログラムを題材に説明してきた。iOSのNFC機能は、「Apple Pay」に限定されていることを先述したが、PCについても調査した。

PC（Windows）ではあまり聞くことがないが、実はWindows 8やWindows 10ではNFC機能に対応している。

PCであまり使われていない理由としては、Windows（OS）が対応していてもハードウェアが対応していないと、NFCを利用できないからである（タッチ画面と同様）。そのため、WindowsOSでNFC機能に対応していても、PC自体がNFCに対応していないと使えないというのが実態である。

現実的には、外付けのNFCアダプターなどを用意することで対応が可能である。またこうしたNFCアダプターには、ハードに合わせたSDK（開発ツール）などが用意されていることが多く、たとえば下記のような製品では、Delphiなどのソースサンプルも提供されている。

＜NFC開発スタートキット101-AB＞
http：//www.orangetags.jp/

こうした外付けのNFCアダプターを利用すれば、バーコードリーダーなどをPC側の入力として利用し、FeliCaなどのICカードを読み取ったデータをアプリケーションの入力として利用できる。【図21】

またSDK（開発ツール）が用意されている場合は、これが本稿でのJNIのようなインターフェースとして使用できるため、NFCアダプターとの連携の開発をしなくともプログラムでは簡単に実装することが可能である。

6.まとめ

本稿ではNFCの基本情報の確認と、Androidを使ったDelphi/400プログラムの実装方法について検証・説明した。NFCを使ったプログラムは、スマートデバイスの普及に伴い、アプリケーションに求められる一般的な機能になると考えられる。なぜなら、より簡単でわかりやすい操作がアプリケーションにとって非常に重要な要素だからである。身近なところで、改札や入退室でのICカード使用の浸透が、それを裏付けている。

今、スマートデバイスは企業でも業務利用が広がり、PCアプリケーションとは異なる新しいユーザーインターフェースが求められている。NFC機能も、スマートデバイスの新しいユーザーインターフェースの1つである。今後は、特にIoTも視野に入れ、新しいユーザーインターフェースを利用する先進のビジネスモデルを考えていくことが必要だろう。そうした際、NFCは不可欠の技術になることを、今回の検証で確信できた。

M

図10A　ダウンロードソース

名前	更新日時	種類
res	2014/09/09 3:31	ファイル フォル...
AndroidManifest.template.xml	2014/09/04 7:16	XML ドキュメント
MainFormU.fmx	2014/09/09 3:32	FireMonkey フ...
MainFormU.pas	2014/09/09 3:33	Delphi ソース フ...
NFC_Sample.deployproj	2014/08/08 6:54	DEPLOYPROJ フ...
NFC_Sample.dpr	2014/09/04 6:31	Delphi プロジェ...
NFC_Sample.dproj	2014/09/09 3:34	Delphi プロジェ...
Readme.txt	2014/08/21 5:16	テキスト文書

図10B　サンプルプログラム

図11　NFCのデバイス設定

図12　NFCの使用権限設定

図13　XMLによる読み込み制御

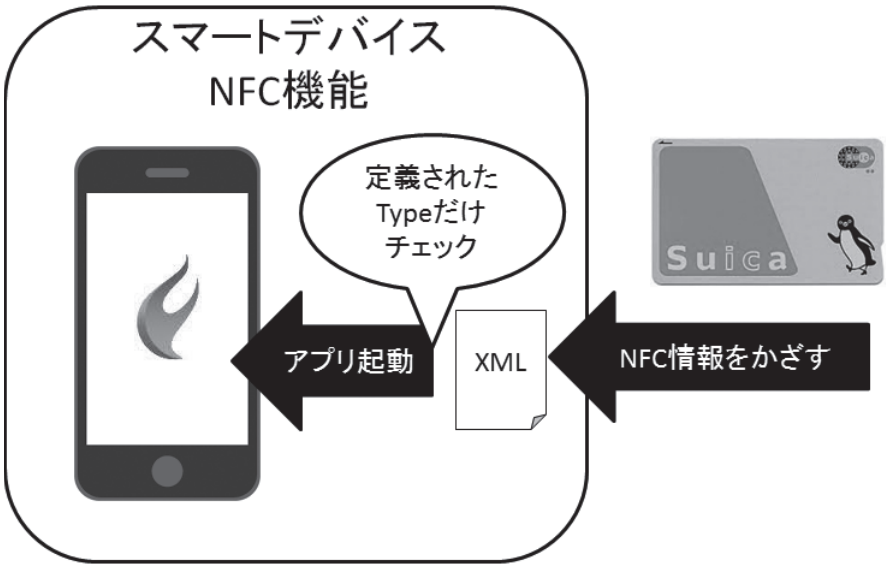


図14　AndroidManifest.xmlの配置

図15　NFC機能の要求

```
    <uses-feature android:name="android.hardware.nfc" android:required="true" />
</manifest>
```

図16　Intent-filter

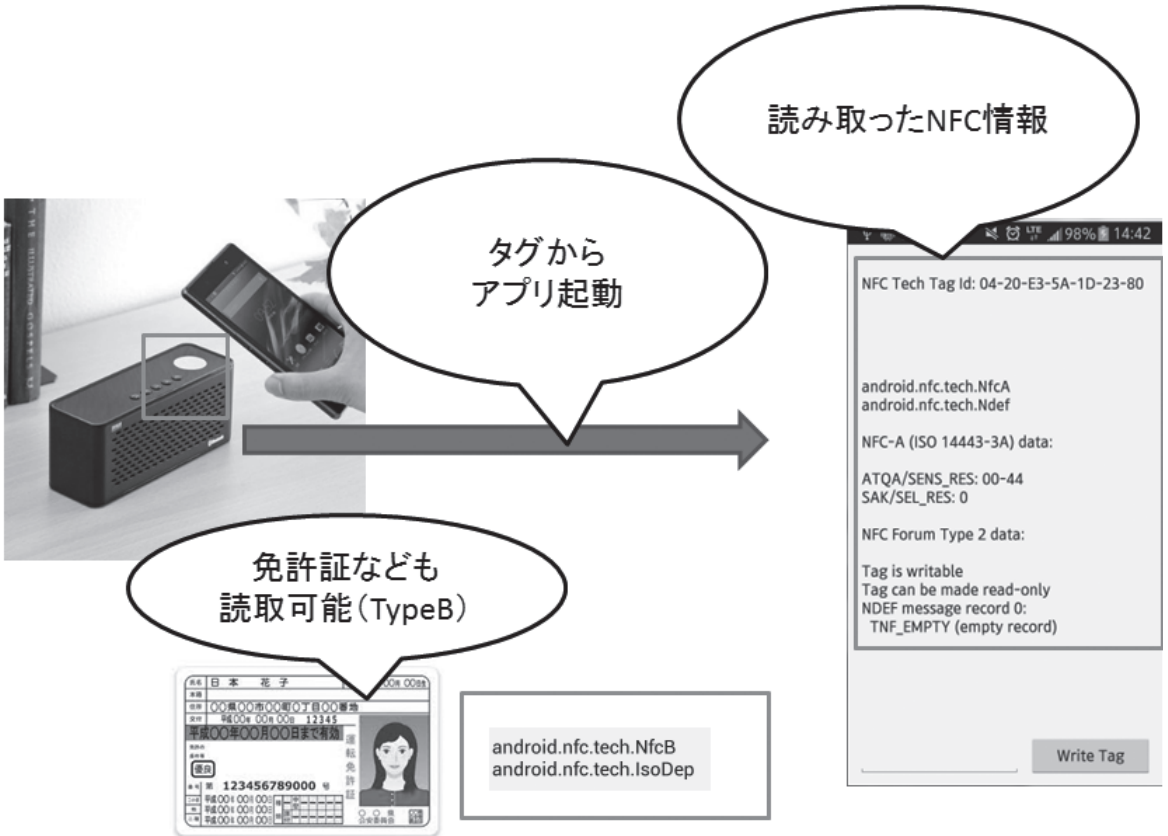
```
<intent-filter>
    <action android:name="android.nfc.action.NDEF_DISCOVERED"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain" />
</intent-filter>
```

図17　対象規格のリスト

```
<resources xmlns:xliff="urn:oasis:names:tc:xliff:document:1.2">
    <tech-list>
        <tech>android.nfc.tech.IsoDep</tech>
    </tech-list>
    <tech-list>
        <tech>android.nfc.tech.NfcA</tech>
    </tech-list>
    <tech-list>
        <tech>android.nfc.tech.NfcB</tech>
    </tech-list>
    <tech-list>
        <tech>android.nfc.tech.NfcF</tech>
    </tech-list>
    <tech-list>
        <tech>android.nfc.tech.NfcV</tech>
```

図18　JNIを使うためのソース

図19　NFCタグの読み取り

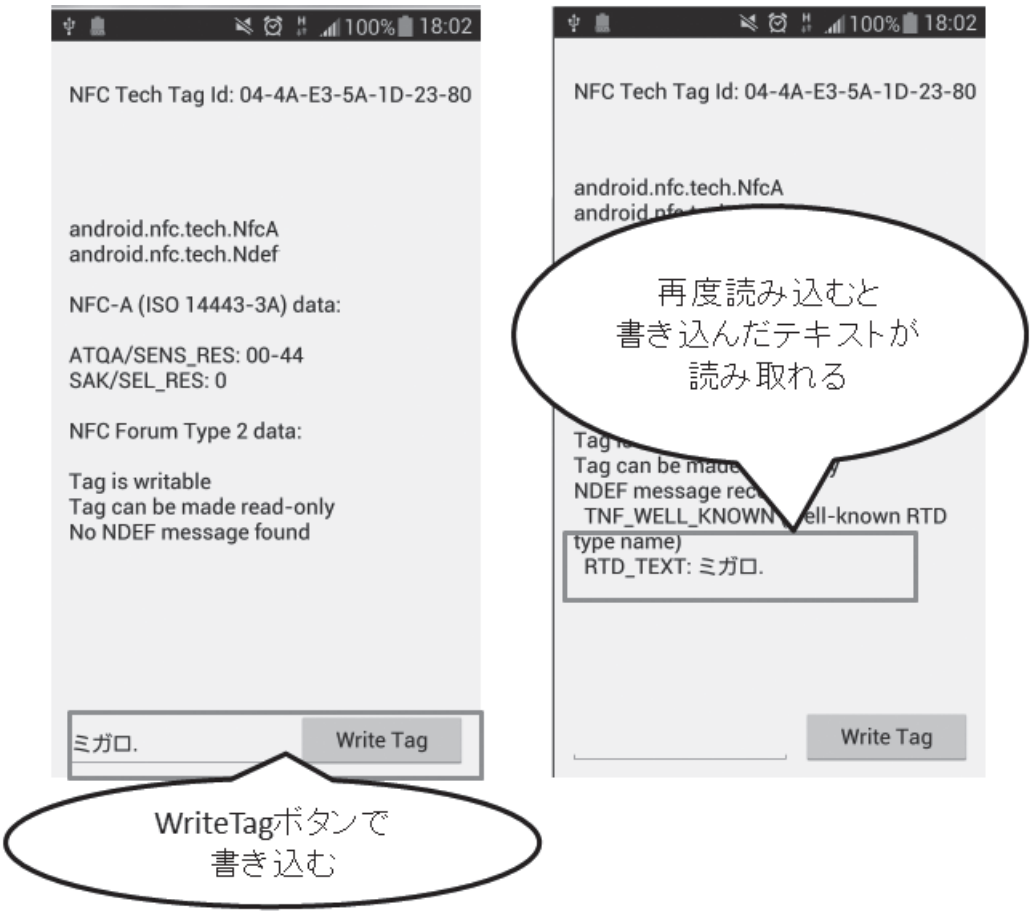


ソース1

NFCタグ読取処理例(Android)

```
procedure TMainForm.FormActivate(Sender:  TObject);
var
  Intent: JIntent;
  TagParcel: JParcelable;
  Tag: JTag;
begin
  Intent := SharedActivity.getIntent;
  //インテントが有効かチェック
  if Intent <> nil then
  begin
    //NFCのアクションを選別
    if TJNfcAdapter.JavaClass.ACTION_NDEF_DISCOVERED.equals(Intent.getAction)  or
       TJNfcAdapter.JavaClass.ACTION_TECH_DISCOVERED.equals(Intent.getAction)  or
       TJNfcAdapter.JavaClass.ACTION_TAG_DISCOVERED.equals(Intent.getAction)  then
    begin
      TagParcel := Intent.getParcelableExtra(TJNfcAdapter.JavaClass.EXTRA_TAG);
      if TagParcel <> nil then
      begin
        Tag := TJTag.Wrap((TagParcel as ILocalObject).GetObjectID);
      end;
      InfoLabel.Text := '';
      //タグの読み込み情報を書き出し
      NFCTagIdLabel.Text  := HandleNfcTag(Tag,
        procedure (const Msg: string)
        begin
          InfoLabel.Text := InfoLabel.Text + Msg + LineFeed;
        end);
    end;
  end;
end;
```

図20　NFCタグの書き込み



ソース2

NFCタグ書込処理例(Android)

```
procedure TMainForm.TagWriteButtonClick(Sender: TObject);
var
  NfcAdapter: JNfcAdapter;
  TagParcel: JParcelable;
  Tag: JTag;
  Intent: JIntent;
begin
  //NFCアダプターをハンドリング
  NfcAdapter := TJNfcAdapter.JavaClass.getDefaultAdapter(SharedActivityContext);
  if (NfcAdapter <> nil) and NfcAdapter.isEnabled then
  begin
    //インテントを取得
    Intent := SharedActivity.getIntent;
    TagParcel := Intent.getParcelableExtra(TJNfcAdapter.JavaClass.EXTRA_TAG);
    if TagParcel <> nil then
    begin
      Tag := TJTag.Wrap((TagParcel as ILocalObject).GetObjectID);
      //タグに情報書き込み
      if not WriteTagText(TagWriteEdit.Text,  Tag) then
        raise Exception.Create('Error connecting to tag');
    end;
  end
  else
    raise Exception.Create('NFC is not available');
end;
```

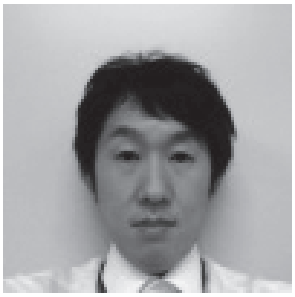
図21　NFCアダプター例

國元 祐二

株式会社ミガロ.

RAD事業部 技術支援課 顧客サポート

[SmartPad4i] スマートデバイス開発で役立つ画面拡張テクニック
―オープンソースライブラリの活用



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

アプリケーションの開発では、無償で公開されている「オープンソース」を利用する開発者が多い。オープンソースとは、ソフトウェアのソースコードを、インターネットなどを通じて無償で公開し、誰でもそのソースコードの改良、再配布が行えるものである。そのためオープンソースでは、独自に開発された機能や画面部品などを公開しているものも多い。

そうした機能や部品を含めたソースをまとめて提供しているものを「オープンソースライブラリ」と呼ぶ場合もある。もちろんオープンソースは開発言語などによっても公開されている内容はさまざまであるため、誰もが使っているわけではないが、スマートデバイスを含むWebアプリケーションの開発においては、使用されることが多い。

なぜならWebアプリケーションは開発言語が違っても、基本的にブラウザ上で動作する点が共通しており、ブラウザ上で動作する言語としてはJavaScriptが一般的に使用されるためである。

つまり、Webアプリケーションという大きな括りでJavaScriptが言語として共通しており、これに統一されて対応したオープンソース開発が非常に発展／充実しているのである。

もちろん、JC/400やSmartPad4iを使ったWebアプリケーション開発でも、JavaScriptを組み込んだ拡張が可能である。本稿では、スマートデバイスでも有用な、SmartPad4iアプリケーションで利用できるオープンソースについて考察を行い、具体的な例を紹介する。

2.オープンソースライブラリの利点

2-1. オープンソースライブラリとは

前節で少し触れたが、「オープンソースライブラリ」は、オープンソースを基盤に作成された汎用的な機能や部品などをまとめたプログラムである。オープンソースライブラリはインターネット上に無償で公開されている場合が多く、ダウンロードして簡単に入手できる。またWebアプリケーションのオープンソースの9割（*注1）は、jQueryと呼ばれるオープンソースライブラリをベースに作成されている。

*注1　Q-Success社のW3Techs（World Wide Web Technology Surveys）の調査結果
http：//w3techs.com/technologies
（左列一覧から「JavaScript Libraries」を選択）

2-2. jQueryについて

jQueryとは、アメリカのプログラマーJohn Resig（ジョン・レシグ）によって開発・公開されたJavaScript用のオープンソースライブラリである。jQueryは著作権表示を消さなければ、商用・非商用を問わず、誰でも自由に利用できる。

jQueryを組み込んで使用すると、本来JavaScriptで大量のプログラムコードを記述しなければ実装できない処理

図1

No.	会員名（漢字）	会員名(カナ)	性別	生年月日	入会日
00000001	細川 エリカ	ホソカワ　エリカ	女性	1967/06/07	2012/01
00000002	大原 結衣	オオハラ　ユイ	女性	1994/03/20	2012/02
00000003	藤澤 南朋	フジサワ　ナオ	女性	1978/09/17	2012/03
00000004	松田 恵麻	マツダ　エマ	女性	1938/01/26	2012/03
00000005	有村 理紗	アリムラ　リサ	女性	1970/04/09	2012/04
00000006	安藤 扶樹	アンドウ　モトキ	男性	1950/04/10	2012/04
00000007	若山 弘也	ワカヤマ　ヒロナリ	男性	1938/01/27	2012/05
00000008	菊田 竜也	キクタ　タツヤ	男性	1976/09/24	2012/05/10
00000009	寺脇 育二	テラワキ　イクジ	男性	1947/01/09	2012/05/22
00000010	高見 浩正	タカミ　ヒロマサ	男性	1955/02/15	2010/05/28
00000011	村井 莉央	ムライ　リオ	女性	1992/06/09	2010/06/12
00000012	高井 雄太	タカイ　ユウタ	男性	1932/04/18	2010/06/12
00000013	福沢 愛梨	フクザワ　アイリ	女性	1984/12/06	2010/06/14
00000014	藤井 奈々	フジイ　ナナ	女性	1946/03/03	2010/07/19
00000015	楠 ひろ子	クスノキ　ヒロコ	女性	1947/05/18	2010/07/25
00000016	おかやま 芳正	オカヤマ　ヨシマサ	男性	1948/10/04	2010/07/26
00000017	吉田 徹	ヨシダ　トオル	男性	1949/05/14	2012/05/11
00000018	宮坂 大樹	ミヤサカ　ヒロキ	男性	1978/05/15	2012/05/20
00000019	塚田 一	ツカダ　ハジメ	男性	1951/03/10	2012/05/28
00000020	山上 くるみ	ヤマガミ　クルミ	女性	1988/03/16	2012/06/08
00000021	植木 信吾	ウエキ　シンゴ	男性	1983/11/05	2012/06/13
00000022	小池 圭	コイケ　ケイ	男性	1988/09/06	2012/06/27
00000023	宮迫 礼子	ミヤサコ　レイコ	女性	1949/05/15	2012/07/02

図2

No.	会員名（漢字）	会員名(カナ)	性別	生年月日	入会日
00000009	寺脇 育二	テラワキ　イクジ	男性	1947/01/09	2012/05/22
00000010	高見 浩正	タカミ　ヒロマサ	男性	1955/02/15	2010/05/28
00000011	村井 莉央	ムライ　リオ	女性	1992/06/09	2010/06/12
00000012	高井 雄太	タカイ　ユウタ	男性	1932/04/18	2010/06/12
00000013	福沢 愛梨	フクザワ　アイリ	女性	1984/12/06	2010/06/14
00000017	吉田 徹	ヨシダ　トオル	男性	1949/05/14	2012/05/11

を、jQueryで用意された機能で簡単に実装することができる。

たとえば、HTMLの要素取得やイベント処理、アニメーション、Ajax処理など、さまざまな機能を提供しており、非常に優れたプログラムであるため、「2-1.」で挙げたように現在このjQueryをベースにしたオープンソースやWebアプリケーションが非常に増えている。

このjQueryの代表的なメリットは、大きく3つ挙げられる。

①JavaScriptの開発効率

先に述べた点と重複するが、JavaScriptで実装するプログラムにおいて、通常記述しなければならないコードをjQueryの機能を利用することでシンプルに実装できる。

たとえば、通常のJavaScriptでは、数十行にわたって記述するようなプログラムロジックであっても、jQueryの機能を使うと数行で記述できてしまう場合も珍しくない。小規模のプログラムであればあまり差はないが、複雑なプログラムを作成する場合には、JavaScriptと比較してコード量が大幅に違ってくるため、開発効率という点で非常にメリットがある。

②クロスブラウザ対応

クロスブラウザとは、Webサイトや Webアプリケーションが、どのWebブラウザでも同じ表示、同じ動作ができることである。JavaScriptはブラウザごとに利用できるメソッドやプロパティが異なるため、JavaScriptによる開発でさまざまなブラウザに対応するには、多くのパターンに対応した大量のプログラムコードを記述する必要がある。

しかし、jQueryでは、こうした異なるブラウザのメソッドやプロパティの違いをjQueryの機能が処理してくれるため、開発者がブラウザの違いを意識して開発をする必要がない。特にスマートデバイスでは、PCと同じ種類のブラウザであっても、別バージョンとなっている場合も多く、非常に役に立つ。

③jQueryオープンソースの利用

「2-1.」で述べた通り、公開されているオープンソースではjQueryをベースにしているものが圧倒的に多いので、jQueryを組み込んでおくことで、こうしたオープンソースを利用することができる。これによってjQueryの機能だけでなく、オープンソースを活用できる範囲を大幅に広げることが可能である。

これらのメリットから、jQueryは公開から急速にシェアを伸ばしてきた。w3techs.comの調査(2-1項の注1)では、インターネット上のすべてのWebサイトにおけるjQueryの利用率は6割と公表している。

2-3. オープンソースライブラリのメリット

オープンソースライブラリは、通常インターネットなどで公開・配布されているため、非常に多くの開発者が利用し、機能が洗練されたものが多い。さらに、これを利用した開発者もWeb上で利用方法などの参考ソースを公開することが多く、使い方に困ることも少ない。そのため、Webアプリケーションや JavaScriptに詳しくない開発者であっても、高度な機能や便利な部品をオープンソースライブラリによって簡単に組み込むことができる。これが最大のメリットである。ただし、無償で公開されているオープンソースであるため、組み込んだアプリケーションでの動作確認などは利用者がしっかりテストする必要がある。

3.SmartPad4iで活用できるオープンソースライブラリ

この章では、SmartPad4iで活用できるオープンソースライブラリの具体的な例や実装方法を紹介していく。

オープンソースである場合、SmartPad4iアプリケーションに組み込んで利用できることも確認が必要であり、本稿では検証を行った、次の4つのオープンソースライブラリの活用例を挙げる。

・スクロールバーを制御できるオープンソースライブラリ
・一覧テーブルを最適表示するオープンソースライブラリ
・ツールチップを利用できるオープンソースライブラリ
・モバイル専用オープンソースライブラリ

3-1. スクロールバーを制御できるオープンソースライブラリ

1つ目は、SmartPad4iで簡単に実装でき、スクロールバーを便利に拡張するオープンソースライブラリ「OVERSCROLL」について紹介する。

通常、一覧テーブルの表示で多くの明細行数を出力すると、ブラウザ全体のスクロールが発生する。たとえば、上部ヘッダに処理ボタンを配置している画面でスクロールをすると、画面全体（処理ボタンごと）がスクロールしてしまい、処理ボタンの操作が不便になる。【図1】

HTMLとcss（スタイルシート）で制御して上部ヘッダを固定することもできるが、その場合、スマートデバイスのブラウザではスクロールバーが表示されない。スクロールバーがないと、現在のデータの位置が判断しにくい。

オープンソースライブラリの「OVERSCROLL」を利用すると、スクロールバーを表示する一覧テーブルが作成できる。

この「OVERSCROLL」は、下記URLからダウンロードできる。【図3】

http：//azoff.github.io/overscroll/

ダウンロードしたzipファイルを展開すると、\test\resources\simple.htmlにサンプルが含まれているので、具体的な実装内容を確認できる。また、ダウンロードしたリソースにはjQuery自体も含まれているので、jQueryを別途ダウンロードして用意する必要はない。

最初に、jQueryとOVERSCROLLのJavaScriptファイルをHTMLと同階層に配置後、外部参照として定義する。【ソース1】

次に、スクロールする対象の要素をタグで囲む。今回は、tableタグが対象になる。また、一覧テーブルのタイトル項目（ヘッダ要素）を表示し続けるため、タイトル項目のみ別テーブルに分けている。【ソース2】

最後に、bodyタグの後にscriptタグを追加し、id属性overscrollの要素でoverscrollメソッドを呼び出せばよい。【ソース3】

この実装だけで、上部ヘッダの表示を固定しながら、一覧データだけをスクロールバーで制御できる使いやすい画面

図3

ソース1

```
<head>
  <meta charset="Shift_JIS" />
  <meta name="viewport" content="width=device-height" />
  <meta name="format-detection" content="telephone=no" />
  <title>MIGARO.Technical Report</title>
  <script src="jquery.min.js"></script>
  <script src="jquery.overscroll.js"></script>
</head>
```

jquery.min.js と overscroll.js をhtmlと同階層に配置後、外部参照として読み込み

を作成することができる。また、「OVERSCROLL」のスクロールバーは、一覧表示以外にも利用可能なため、たとえば画面内ですべて表示しきれない高画質の画像をスクロール形式で表示できる。【図4】

3-2. 一覧テーブルを最適表示するオープンソースライブラリ

2つ目は、一覧テーブルを簡単にレスポンシブデザインにすることができるオープンソースライブラリ「FooTable」を紹介する。レスポンシブデザインとは、css（スタイルシート）を利用して、1つのHTMLからデバイスの画面サイズに合わせて表示を切り替えるデザイン手法である。

たとえば、PCとスマートデバイスとでは、画面サイズが異なるため、PCブラウザ向け画面と、スマートデバイス向け画面の2つを用意したい場合がある。画面サイズを最適化するだけであれば簡単だが、表示項目数などを制御する場合、通常2種類のHTMLを作成する必要がある。しかし、レスポンシブデザインでは1つのHTMLをデバイスのサイズで判断して、cssを切り替えて表示することができる。【図5】

たとえば、トヨタ自動車のWebサイトはレスポンシブデザインに対応している。PCブラウザではトップメニューが横並びになっているが、スマートフォンで参照時にはメニューが折りたたまれて表示される。【図6】

通常、レスポンシブデザインに対応するには、HTMLとcssを工夫・調整して定義する必要があるが、オープンソースライブラリの「FooTable」は簡単にレスポンシブデザインに変更することができる。たとえば、「FooTable」を利用すると、【図7】【図8】のような画面表示が実現できる。

PCのブラウザを利用してアプリを実行すると、一覧テーブルのすべての列項目を表示する。【図7】

同じ画面をタブレットでアクセスすると、一部の列項目が非表示になるが、タッチするとデータを展開表示できる。【図8】

スマートフォンの場合も、同様に一部の列が非表示になり、タッチすることでデータを展開表示できる。【図9】

このように表示するデバイスに合わせて最適な表示に切り替えてくれるのが「FooTable」の機能である。

この「FooTable」は、下記URLからダウンロードできる。【図10】

https：//github.com/fooplugins/
（一覧から「FooTable」を選択）

ダウンロードしたdemos\index.htmlには「FooTable」の利用方法が詳しく記載されている。

まず、展開したファイルのcssフォルダとjsフォルダを、SmartPad4iのHTMLを格納しているフォルダへ配置する。配置後、headタグに「FooTable」のJavaScriptファイルとcssファイル、jQueryファイルの外部参照を定義する。【ソース4】

次に、サブファイルのtableタグに「FooTable」の設定を追加する。「FooTable」では、まずtableタグのclass属性にfootableを設定する。そして、thまたはtdタグにdata-class属性を追加してexpandを設定する。この設定により、デバイスサイズが小さい場合に「＋」が表示されて、展開時に「－」が表示される。【図11】

次に、data-hide属性を設定する。data-hide属性を設定することでスマートフォンとタブレットの利用時に列項目の表示、非表示が制御できる。【ソース5】

最後に、JavaScriptをbodyタグの後に追加する。【ソース6】

class属性にfootableを指定した要素に対してfootableメソッドを実行することで「FooTable」の機能が有効になる。breakpointsパラメータでは、タブレットとスマートフォンのデバイス横幅の区切りを指定している。

こうした設定をして「FooTable」を使用すると、簡単にデバイスごとの一覧テーブルを最適化してレスポンシブ表示できる。

3-3. ツールチップを利用できるオープンソースライブラリ

3つ目は、入力欄のツールチップ表示が可能になるオープンソースライブラリ「Mouseinfobox plugin for jQuery」を紹介する。

ツールチップとは、画面上の要素（入力欄等）にマウスやカーソルが設定された時に表示されるナビゲーションで、選択された項目に関するヘルプのようなものだ。ツールチップ表示は、入力項目などに利用すると、ユーザーにとってわかりやすい操作説明を提供できる。【図12】

たとえば、郵便番号を7ケタの数値で入力後、入力欄横のボタンをタッチしてほしい場合など、ツールチップを表示してユーザー操作を促すことができる。入力時に、操作方法が表示されると、ユーザーにとって次の操作がわかりやすく、また画面で常に表示されているわけではないので、画面デザイン的にも煩雑にならない。このオープンソースライブラリもHTMLとJavaScriptを記述することで簡単に利用することができる。

「Mouseinfobox plugin for jQuery」は、下記URLからダウンロードできる。【図13】

http：//portfolio.cmegnin.fr/mouseinfobox/

まず、展開したファイルのcssフォルダとjsフォルダをSmartPad4iのHTMLを格納しているフォルダへ配置後、headタグに「Mouseinfobox plugin for jQuery」のスクリプトファイルとcssファイル、jQueryファイルを外部参照として定義する。【ソース7】

ツールチップに表示する文字は、inputタグのtitle属性として定義できる。【ソース8】

最後に、JavaScriptをbodyタグの後に追加する。【ソース9】

JavaScriptでは入力欄の要素をjQueryのセレクタ用関数$()を使って選択し、要素のinfoBoxメソッドを呼び出すことでツールチップが表示できる。このinfoBoxメソッドではいくつかのパラメータが設定可能である。今回は、4つのパラメータを指定している。

＜animationパラメータ＞
ツールチップの表示方法を定義できる。

＜opacityパラメータ＞
ツールチップの背景色を透過にできる。

ソース2

テーブルのヘッダ要素を別テーブルで定義する

```
<table width="100%" border="1px" cellspacing="0" cellpadding="5px"
  <thead>↵
    <tr>↵
      <th width="110px" >No.</th>↵
      <th width="150px">会員名（漢字）</th>↵
      <th>会員名(カナ)</th>↵
      <th width="40px">性別</th>↵
      <th width="120px">生年月日</th>↵
      <th width="120px">入会日</th>↵
    </tr>↵
  </thead>↵
</table>↵
<!-- overscroll -->↵
<div id="overscroll" style="height:390px">↵
  <table id="SFL1" width="100%" border="1px" cellspacing="0" cellp
    <tbody>↵
      <tr>↵
        <td id="LCNO" width="110px">00000001</td>↵
        <td id="LCNM1"  width="150px"> </td>↵
        <td id="LCNM2"> </td>↵
        <td id="LSEX" width="40px">男性</td>↵
        <td id="LBIRT" width="120px"> </td>↵
        <td id="LENTD" width="120px"> </td>↵
      </tr>↵
    </tbody>↵
  </table>↵
</div>↵
<!-- /overscroll -->↵
```

Div要素にid属性　overscrollを設定Table要素の外側にタグに設定する

ソース3

```
  </body>↵
  <script>↵
    $(function () {↵
      $("#overscroll").overscroll();↵
    });↵
  </script>↵
</html>↵
↵
```

id属性 overscrollを付けた要素に対してスクロールの設定メソッドを実行

＜bottomパラメータ＞
ツールチップを下から上へアニメーションで表示できる。

＜useMouseパラメータ＞
ツールチップの表示位置を指定することができる。指定できるoffsetX、offsetYはツールチップの表示位置である。X軸、Y軸の値をピクセル単位で指定して位置を調整できる。

なお、設定可能なパラメータは、ダウンロードしたファイルに含まれるdemo.htmlに詳しく記載されている。

3-4. モバイル専用オープンソースライブラリ（フレームワーク）

最後に、スマートフォンやタブレット専用に最適化したインターフェース、画面部品で表示するオープンソースライブラリ「jQuery Mobile」を紹介する。

HTMLでラジオボタンや、チェックボックスを作成した場合、スマートフォンのように画面が小さいと、ラジオボタンやチェックボックスが押しにくく、不便な場合がある。これはPC向けのWebブラウザのインターフェースが、マウスのクリック操作やキーボード操作を前提に作られているためである。

このjQuery Mobileは、タッチ操作を前提としたインターフェースに最適化してくれるオープンソースライブラリである。

SmartPad4iでは、スマートデバイス向けのアプリケーションを作成できるため、jQuery Mobileを組み込んだインターフェースの拡張が有効である。【図14】

jQuery Mobileは下記URLからダウンロードできる。【図15】

https：//jquerymobile.com/

ダウンロードしたjQuery Mobileを展開して、SmartPad4iのテンプレートのHTMLと同一階層に展開したディレクトリごとに配置する。次に、展開したファイルをheadタグ内で外部参照として定義する。【ソース10】

後は、HTMLをjQuery Mobileのフレームワークに沿って定義するだけでインターフェースをスマートデバイス専用に変更することができる。

jQuery MobileのHTMLの記述方法については、HTML5の記述をベースにdata-［xxx］属性などの拡張属性を設定して定義する。data-［xxx］属性はHTML5用の属性である。

jQuery Mobileでは、このdata-［xxx］属性を利用してJavaScriptを記述しなくても、インターフェースを拡張できる。

ここではラジオボタンを例に、拡張方法を説明する。

jQuery Mobileのラジオボタン表示では、HTML5で定義されたfieldsetタグを使う。fieldsetタグは、フォームの入力項目をグループ化する際に使用するタグである。

このタグのdata-role属性にcontrolgroupを設定することでラジオボタンがグループ化できる。グループ化された要素は、互いに余白なしで配置される。そして、ラジオボタンの記述表示は、labelタグでラジオボタンを囲むことで可能になる。【ソース11】

また、jQuery Mobileでは、ラジオボタンを横並びに表示することもできる。横並びは単純にfieldsetタグのdata-type属性にhorizontalを設定するだけである。【ソース12】

このようにjQuery Mobileを利用すると簡単にスマートデバイス対応のインターフェースを作成できる。もちろん、例に挙げたラジオボタン以外のさまざまな画面部品を同じような簡単な設定だけで拡張が可能である。なお、jQuery Mobileは下記のURLでデモが公開されているので、他の画面部品の具体的な使用方法についてもソース付きで確認できる。

http：//jquerymobile.com/demos/

4.オープンソース実装の最適化

4-1. 読み込まれるリソースのサイズ

前章の例では、各オープンソースライブラリの各リソースをダウンロードしてWebサーバーに配置している。そのファイルのリソースには、*.cssや*.jsファイルの他に*.min.cssや*.min.jsのファイルが含まれている。ファイル名にminが付いたファイルと、付いていないファイルの2種類が存在するのには理由がある。

minが付いていないファイルでは、cssの設定やJavaScriptのプログラムソースが読みやすいように、改行やインデントを含めたファイルとして用意されている。

一方、minが付いたファイルは、容量を小さくするために、css適用時やJavaScriptの実行時に不要な改行やインデント、または変数名などを圧縮して、サイズをできるだけ小さくしたファイルとして用意されている。実際にWebアプリケーションが実行時に読み込むのはminがついた*.min.cssや*.min.jsのファイルである。

小さなファイルサイズのcssやJavaScriptファイルではパフォーマンスに影響が出ることはないが、ファイルのサイズが大きくなるにつれてWebアプリケーションの初期読み込みの時間がかかるようになる。cssやJavaScriptファイルを圧縮する理由は、リソースのダウンロード時間を減らし、Webアプリの初期実行速度を向上させるためである。

次節では、作成したcssやjsファイルを圧縮し最適化する方法について説明する。

4-2. リソースの圧縮方法

cssやJavaScriptの圧縮には、専用のツールやWebサービスを利用することができる。今回は、cssとJavaScriptの両方を圧縮できるWebサービス「Online JavaScript/CSS Compressor」を紹介する。

Online JavaScript/CSS Compressorは、http：//refresh-sf.com/にブラウザでアクセスしてcssやJavaScriptを入力欄に張り付けてボタンを実行だけで利用することができる。【図16】

プログラムで利用しているcssを圧縮する場合、Inputの入力欄にcssのソースを張り付けて、cssボタンをクリックする。すると、圧縮が実行され結果が表示されるのでSaveボタンから圧縮されたcssファイルをダウンロードできる。【図17】

例として、12.9KBだったcssファイルを圧縮すると5.2KBに圧縮することができた。もちろん、JavaScriptファ

図4

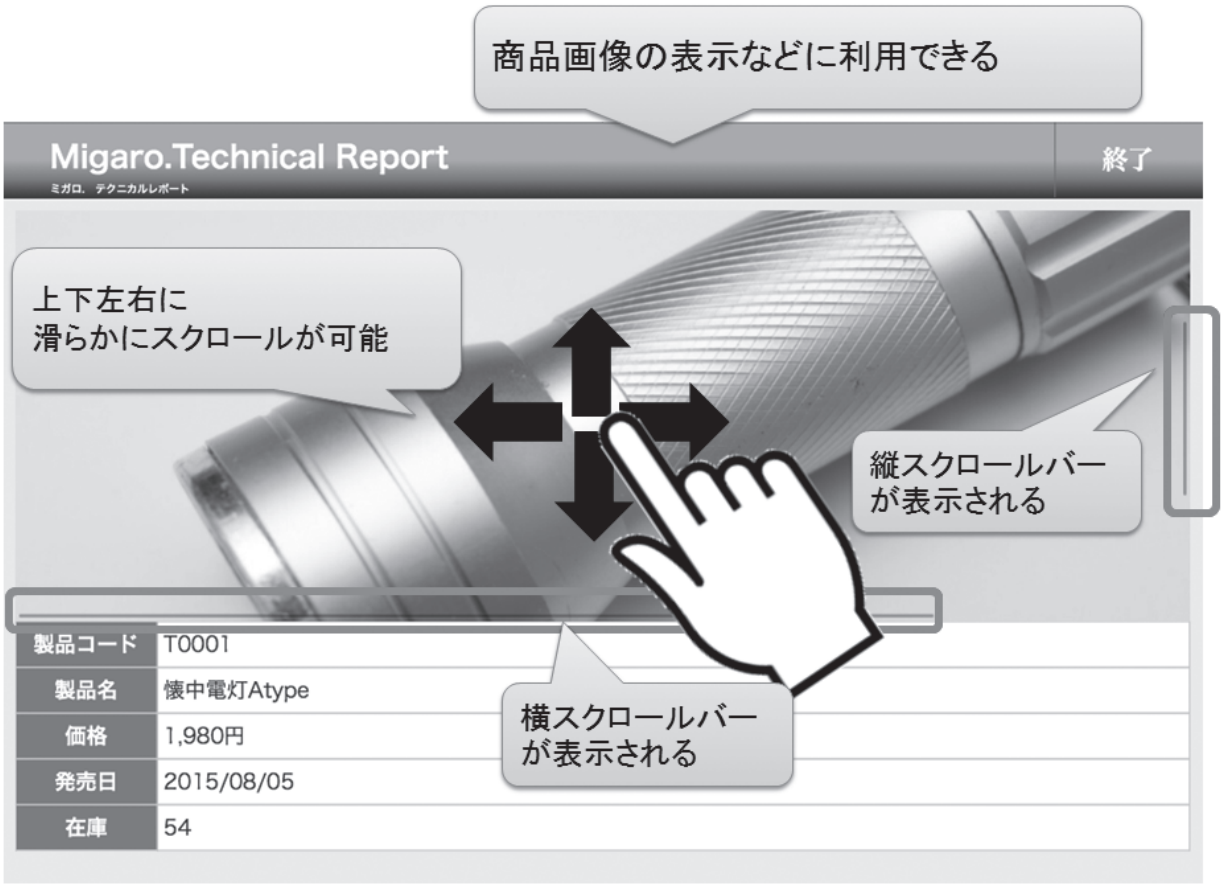
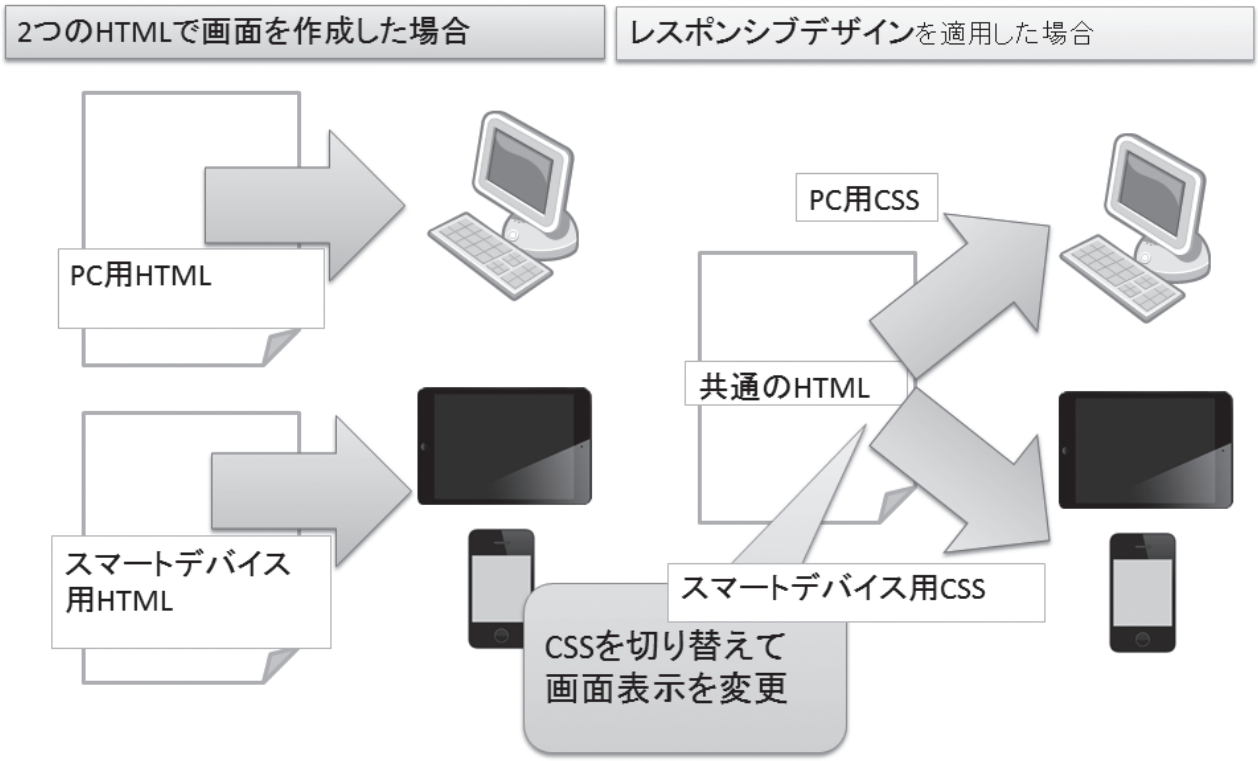

商品画像の表示などに利用できる
Migaro.Technical Report
ミガロ. テクニカルレポート
終了
上下左右に
滑らかにスクロールが可能
縦スクロールバー
が表示される
横スクロールバー
が表示される
製品コード T0001
製品名 懐中電灯Atype
価格 1,980円
発売日 2015/08/05
在庫 54

図5


2つのHTMLで画面を作成した場合
レスポンシブデザインを適用した場合
PC用HTML
スマートデバイス
用HTML
PC用CSS
共通のHTML
スマートデバイス用CSS
CSSを切り替えて
画面表示を変更

イルでも同様に使用できる。入力欄にソースを張り付けて、JavaScript ボタンをクリックすることで圧縮できる。このように Online JavaScript/CSS Compressor を利用すると、簡単に css や JavaScript のリソースを圧縮できる。【図 18】

css や JavaScript のサイズが大きいほど効果があるため、ファイルが大きくなった場合には有効である。

5.まとめ

本稿では、オープンソースライブラリを利用して SmartPad4i アプリケーションの画面部品を拡張する例をいくつか紹介した。冒頭でも説明したが、Web アプリケーションではオープンソースが非常に発展しており、オープンソースライブラリもさまざまな機能や画面部品が公開されている。

製品ソフトウェアと異なり、開発元などで保証されないオープンソースということで、アプリケーション開発での使用を躊躇する場合もある。しかし、オープンソースはすべてのソースコードが公開されているため、使用する部分のソースをチェックして、自社のプログラムとして把握したり、部分的に流用することも可能である。

また、オープンソースライブラリのメリットでも挙げたように、既に完成している機能を利用できるのは、それだけ開発の工数を短縮できるということである。

本稿で紹介した 4 つのオープンソースライブラリも、利用するのは簡単であるが、同じ機能を一から開発するのであれば、かなりの開発工数・期間を想定する必要がある。

SmartPad4i アプリケーションの開発で、画面機能の拡張を考える場合に、同じ機能のオープンソースライブラリが公開されていれば、一度それを組み込んで試してみることをお勧めする。

M

図6


PC（PCブラウザの場合）
スマートフォン(モバイルブラウザの場合)
ボタンタッチ
メニュー表示
TOYOTA
楽しさ、サイズ以上！
Sienta
クルマを探す
toyota.jp
カーラインナップ
トヨタの最新技術
ご購入サポート
アクセサリー
アフターサービス
U-Car（中古車）
おすすめ情報
企業・IR情報
リコール情報
ログイン

図7


PC（PCブラウザの場合）
Migaro.Technical Report
ミガロ．テクニカルレポート
終了
No.
男性
女性
全て
入会日
検索
条件クリア
選択
No.
会員名(漢字)
会員名(カナ)
性別
生年月日
入会日
00000001 細川 エリカ ホソカワ エリカ 女性 1967/06/07 2012/01/05
00000002 大原 結衣 オオハラ ユイ 女性 1994/03/20 2012/02/12
00000003 藤澤 南朋 フジサワ ナオ 女性 1978/09/17 2012/03/21
00000004 松田 恵麻 マツダ エマ 女性 1938/01/26 2012/03/30
00000005 有村 理紗 アリムラ リサ 女性 1970/04/09 2012/04/02
00000006 安藤 扶樹 アンドウ モトキ 男性 1950/04/10 2012/04/04
00000007 若山 弘也 ワカヤマ ヒロナリ 男性 1938/01/27 2012/05/02
00000008 菊田 竜也 キクタ タツヤ 男性 1976/09/24 2012/05/10
00000009 寺脇 育二 テラワキ イクジ 男性 1947/01/09 2012/05/22
列が全て表示される

図8


iPad (タブレットの場合)
Migaro.Technical Report
会員名(カナ) と 生年月日の列が非表示になる。
行をタッチ
非表示の列はタッチして展開表示できる。
会員名(カナ): フジサワ ナオ
生年月日: 1978/09/17

図9

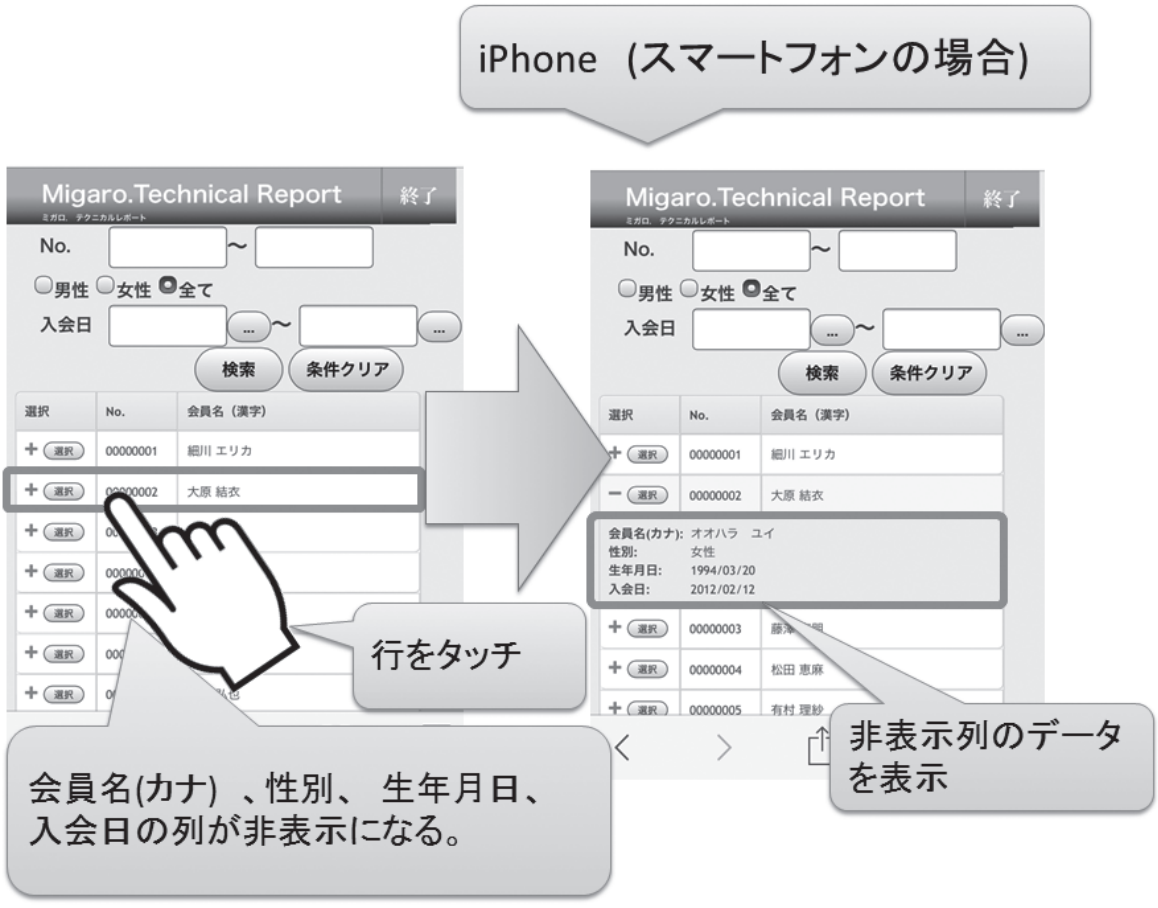

iPhone (スマートフォンの場合)
Migaro.Technical Report
終了
行をタッチ
会員名(カナ) 、性別、 生年月日、入会日の列が非表示になる。
非表示列のデータを表示

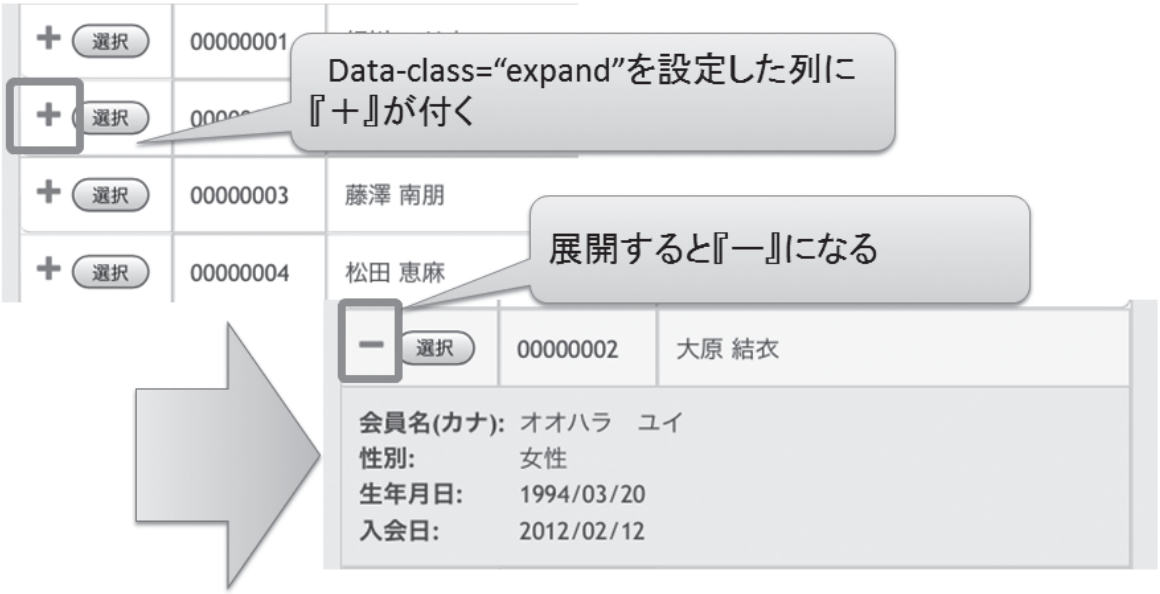
図10

ソース4

```
<head>↵
  <meta charset="Shift_JIS" />↵
  <meta name="viewport" content="width=device-height" />↵
  <meta name="format-detection" content="telephone=no" />↵
  <title>MIGARO.Store System</title>↵
  <link href="DEMOCSS.css" type="text/css" rel="stylesheet" >↵
  <link href="css/footable.core.min.css" type="text/css" rel="stylesheet">↵
  <link href="css/footable.standalone.min.css" type="text/css" rel="stylesheet">↵
  <script src="jquery.min.js"></script>↵
  <script src="js/footable.js?v=2-0-1" ></script>↵
<head>↵
```

fooTableのcssと jQuery, FooTableのjsファイルを外部参照として読み込み

図11



ソース5

```
<table id="SFL1"  border="1px" cellspacing="0" cellpadding="5px"
  style="margin-bottom:40px;" class="footable table">
```

class属性にfootableを指定

```
<thead>
  <tr>
    <th width="70px">選択</th>
    <th width="70px" data-class="expand" >No.</th>
    <th >会員名（漢字）</th>
    <th width="450px" data-hide="phone,tablet">会員名(カナ)</th>
    <th width="60px" data-hide="phone">性別</th>
    <th width="100px" data-hide="phone,tablet">生年月日</th>
    <th width="100px" data-hide="phone">入会日</th>
  </tr>
</thead>
```

data-class属性にexpandを指定

data-hide属性には、タブレット表示時に非表示にする列にはtabletを設定、スマートフォン表示時に非表示にする列にはphoneを設定

ソース6

```
<script>
  $(function() {
    $('.footable').footable({
      breakpoints: {
        phone: 640,
        tablet: 1024
      }
    });
  });
</script>
```

bodyタグ後に、スクリプトを追加、class属性にfootableを指定した要素にfootableメソッドを呼び出す。
breakpointsパラメータでは、タブレットとスマートフォンのデバイス横幅の区切りを指定

図12

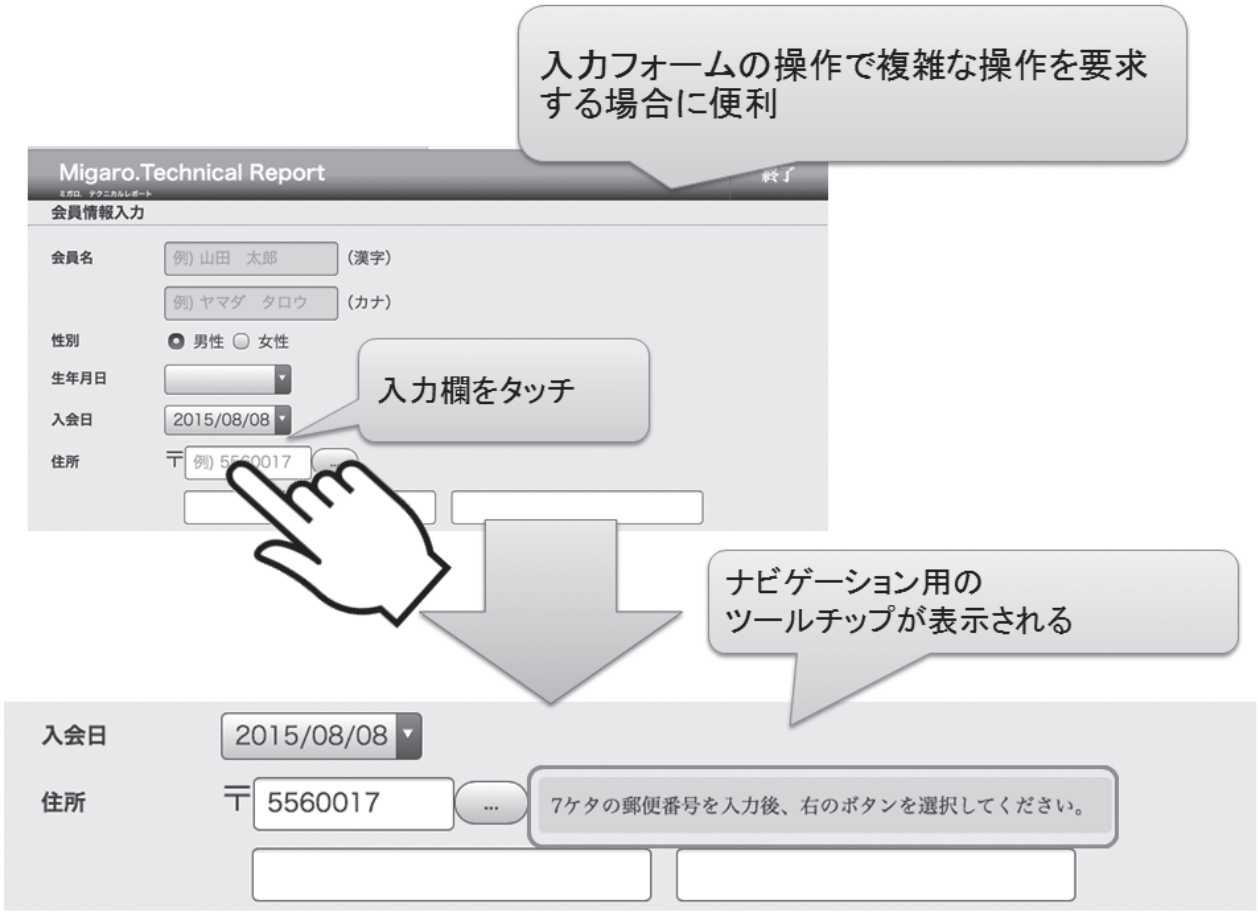
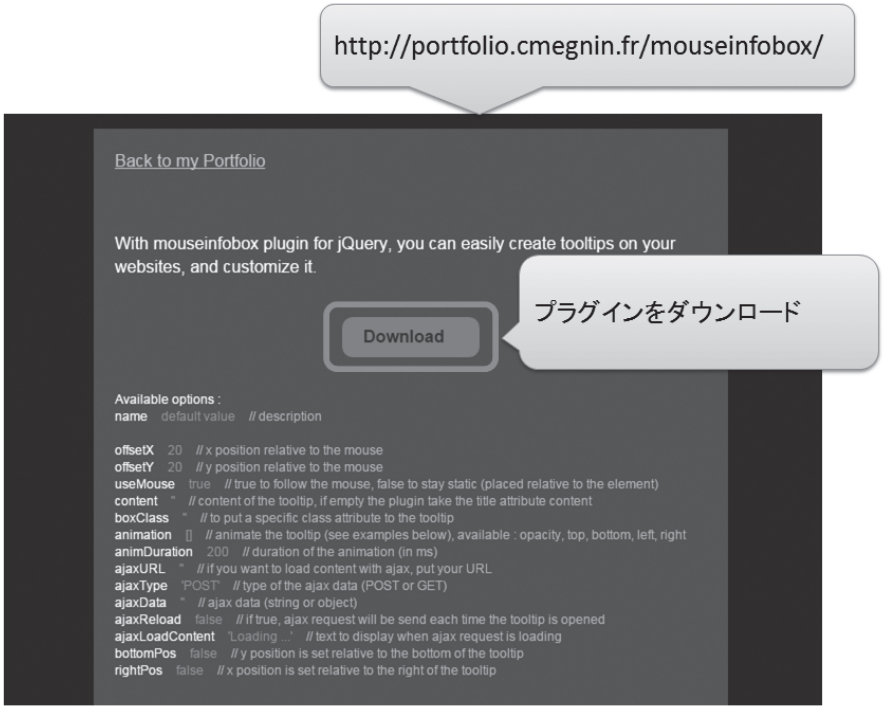


図13



ソース7

```
<head>
  <meta charset="Shift_JIS" />
  <meta name="viewport" content="width=device-height" />
  <meta name="format-detection" content="telephone=no" />
  <title>MIGARO. Customer System</title>
  <link rel="stylesheet" href="DEMOCSS.css" type="text/css">
  <link rel="stylesheet" href="css/jquery.mouseinfobox.css" type="text/css"/>
  <script src="js/jquery-1.6.2.min.js"></script>
  <script src="js/jquery.mouseinfobox.js"></script>
</head>
```

外部参照として読み込み

ソース8

```
<input type="text" title="7ケタの郵便番号を入力後、右のボタンを選択してください。"
  name="INAD1" id="INAD1" style="width:140px;float:left"
    maxlength="8" placeholder="例) 5560017">
```

入力欄のtitle属性に表示したいツールチップの文字を定義

ソース9

```
<input type="text" title="7ケタの郵便番号を入力後、右のボタンを選択してください。"
  name="INAD1" id="INAD1" style="width:140px;float:left"
    maxlength="8" placeholder="例) 5560017">
```

```
<script>
  $('#INAD1').infoBox({
    animation: ['opacity', 'bottom'],
    useMouse: false,
    offsetX: 230,
    offsetY: 0
  });
</script>
```

bodyタグ後に、スクリプトを追加、id属性を指定してinfoBoxメソッドを呼び出す。

図14

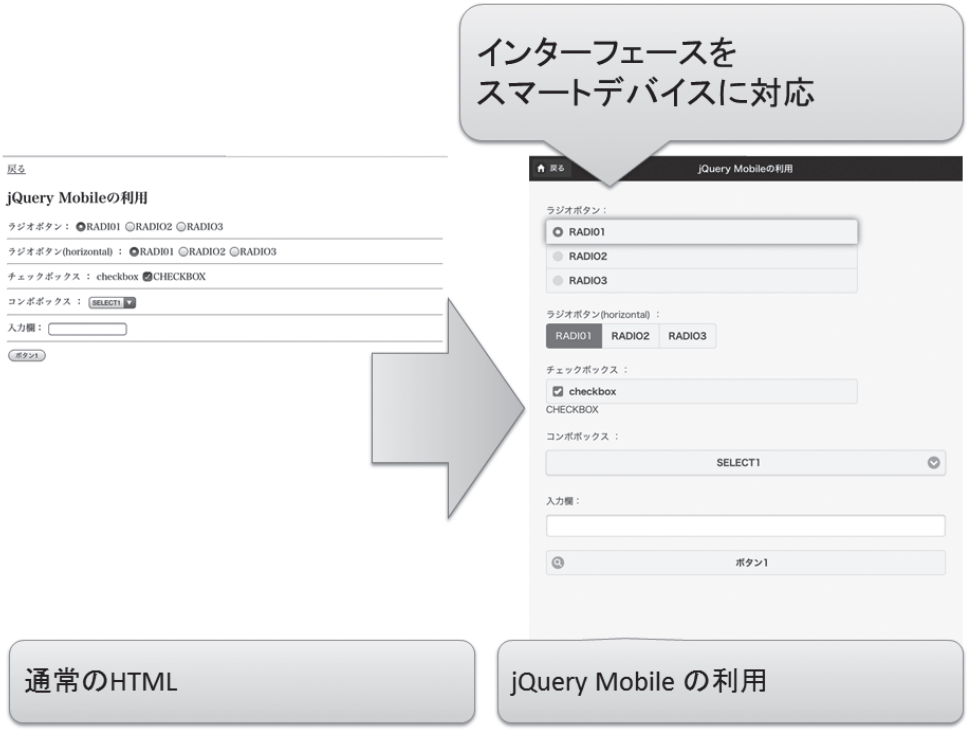
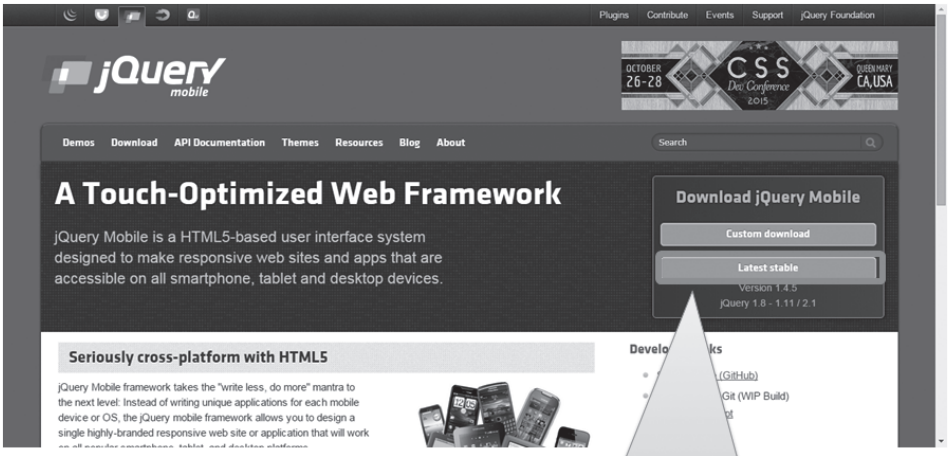


図15

https://jquerymobile.com/



「Latest stable」からダウンロード

ソース10

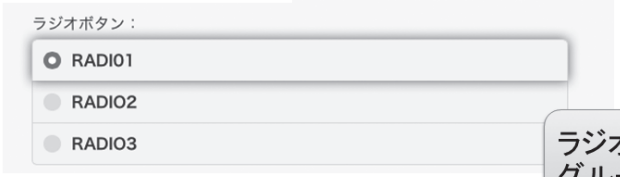
jQuery Mobile 設定

```
<head>↵
  <link rel="stylesheet" href="mobile.css" />↵
  <link rel="stylesheet" href="jquerymobile/jquery.mobile-1.4.5.min.css" />↵
  <script src="js/jquery.js"></script>↵
  <script src="jquerymobile/jquery.mobile-1.4.5.min.js"></script>↵
</head>↵
```

外部参照として読み込み

ソース11

Radioボタン表示例



ラジオボタンの要素をグループ化する

```
<fieldset data-role="controlgroup" >↵
  <label><input id="RD01" name="RD01" ↵
          type="radio" value="1" />radio1</label>↵
  <label><input id="RD01" name="RD01" ↵
          type="radio" value="2" />radio2</label>↵
  <label><input id="RD01" name="RD01"↵
          type="radio" value="3" />radio3</label>↵
</fieldset>↵
```

Labelタグを使って項目名を表示

ソース12

Radioボタン横並び表示例

タッチ操作で扱いやすい

Date-type属性に
horizontalを設定

ラジオボタン(horizontal) ：

RADIO1 RADIO2 RADIO3

```
<fieldset data-role="controlgroup"↵
   data-type="horizontal">↵
  <label><input id="RD02" name="RD02"↵
          type="radio" value="1" />radio1</label>↵
  <label><input id="RD02" name="RD02"↵
          type="radio" value="2" />radio2</label>↵
  <label><input id="RD02" name="RD02" ↵
          type="radio" value="3" />radio3</label>↵
</fieldset>↵
```

図16

http://refresh-sf.com/

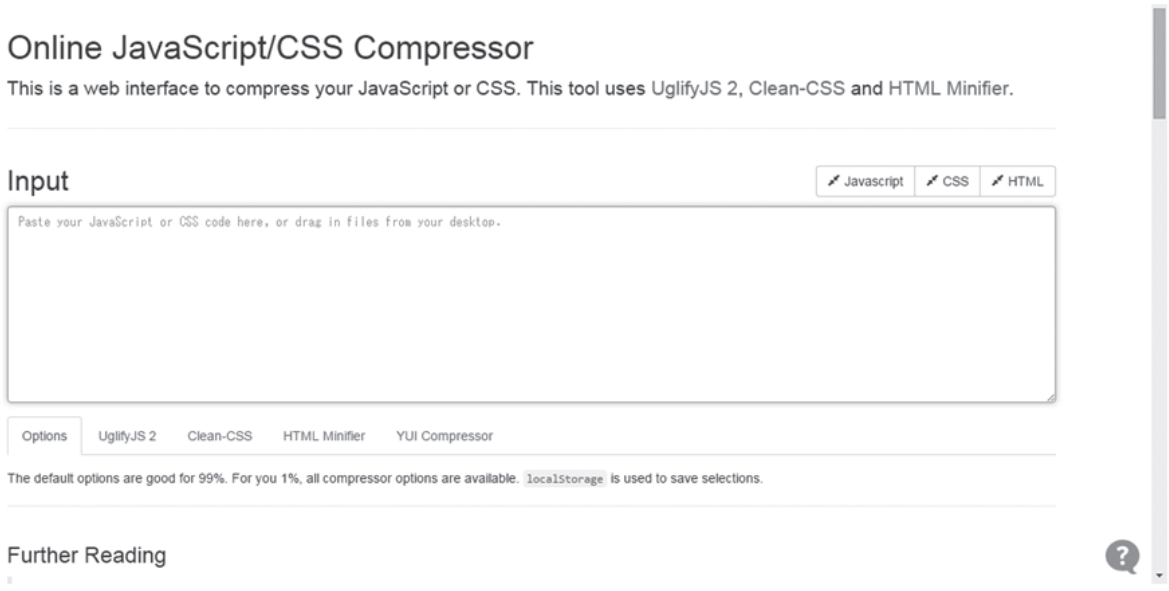


図17

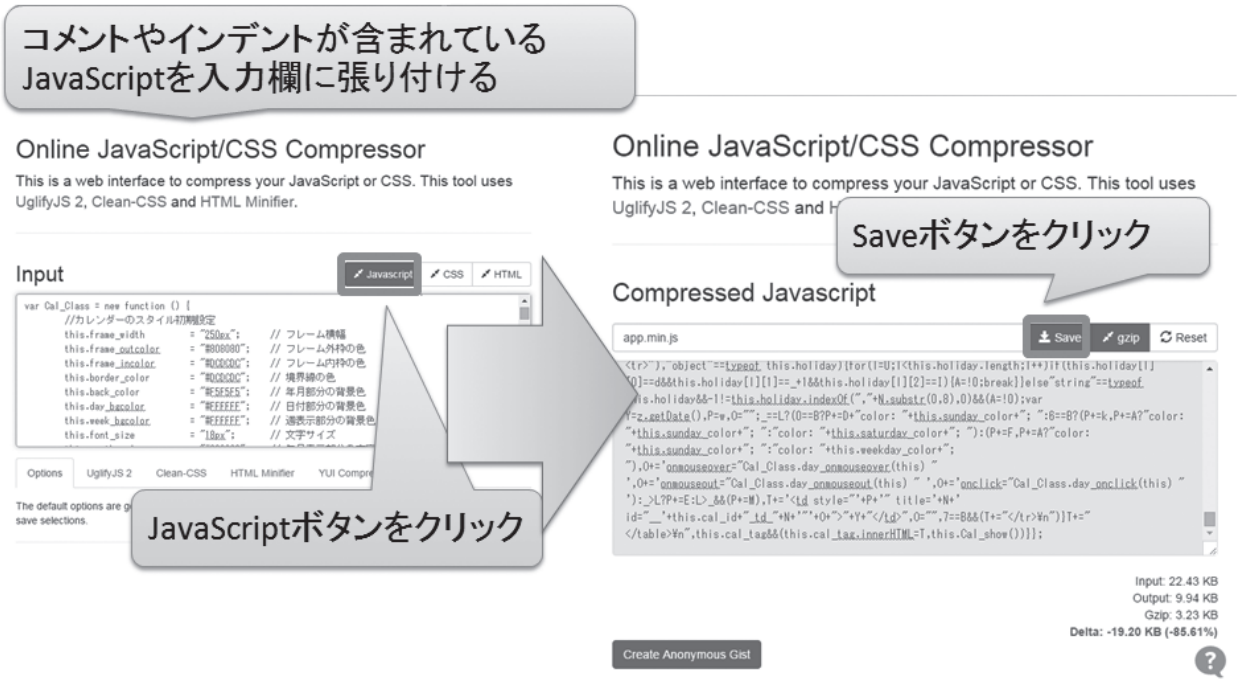

コメントやインデントが含まれている
JavaScriptを入力欄に張り付ける
Online JavaScript/CSS Compressor
This is a web interface to compress your JavaScript or CSS. This tool uses
UglifyJS 2, Clean-CSS and HTML Minifier.
Input
Javascript
CSS
HTML
Options
UglifyJS 2
Clean-CSS
HTML Minifier
JavaScriptボタンをクリック
Online JavaScript/CSS Compressor
Saveボタンをクリック
Compressed Javascript
app.min.js
Save
gzip
Reset
Input: 22.43 KB
Output: 9.94 KB
Gzip: 3.23 KB
Delta: -19.20 KB (-85.61%)
Create Anonymous Gist

図18


コメントやインデントが含まれている
css, JavaScript
コメント,インデント,改行が削除されて
最適化された css, JavaScript
Online JavaScript/CSS Compressor
This is a web interface to compress your JavaScript or CSS. This tool uses
UglifyJS 2, Clean-CSS and HTML Minifier.
Options
UglifyJS 2
Clean-CSS
HTML Minifier
YUI Compressor
圧縮処理

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

MEMO

MIGARO.
TECHNICAL
REPORT

Migaro.Technical Report
No.8 2015 年秋
ミガロ.テクニカルレポート

2015 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