佐田 雄一

株式会社ミガロ.

RAD事業部 技術支援課

[Delphi/400]

Enterprise Connectorsを利用したクラウド連携テクニック

略歴
1985年12月6日生まれ
2009年3月 甲南大学 経営学部卒業
2009年4月 株式会社ミガロ. 入社
2009年4月 システム事業部配属
2019年4月 RAD事業部配属

現在の仕事内容
Delphi/400でのシステム開発や保守作業の経験を経て、現在はサポート業務を担当している。

1. はじめに
2. Enterprise Connectorsとは
3. Enterprise Connectorsのインストール
4. Enterprise Connectorsの開発方法
5. クラウドサービスへの接続・活用方法
5-1. Twitterと連携した情報発信と解析
5-2. Googleスプレッドシートへのデータ出力
5-3. Google Driveでのデータ検索
6. まとめ

1. はじめに

基幹系の業務システムは長らく、社内上のサーバーにデータを蓄積するオンプレミス（自社運用）で稼働してきた。Delphi/400のシステムでも、IBM iを中心にオンプレミス環境で稼働していることが多い。

従来はすべてのシステムをオンプレミスで稼働させるのが一般的だったが、近年はWebを介して稼働するさまざまなクラウドサービスが台頭しており、それらを業務で活用することも増えている。オンプレミスのシステムとクラウドサービスは別々に稼働していることが多いのが現状であり、これらの連携が課題になることもあるだろう。このようなオンプレミスとクラウドサービスとの連携でも、Delphi/400が活用できる。

本稿では、Delphi/400からクラウドサービスへのアクセス用に用意されたEnterprise Connectorsを利用して、IBM iのデータとクラウドサービスを連携させる方法について紹介する。

2. Enterprise Connectorsとは

Enterprise Connectorsとは、エンバカデロ・テクノロジーズ社がCData Software社との提携によって、2017年から提供を開始したコンポーネント群で、Delphi/400に搭載されたデータベースエンジンFireDACを使用してクラウドサービスに接続する。

各クラウドサービスには、それぞれのベンダーが提供するWeb APIがあり、それを使用することでDelphi/400等の各種クライアントからクラウドサービスに接続できた。しかし「クラウドサービスごとに接続するAPIの仕様が異なり、個別にAPIの仕様を調査の上、実装しなければいけない」「API側の仕様のアップデート間隔が短いことが多く、仕様変更の都度、個別に修正を実装しなければならない」など、課題が多いのが現状である。

またAPIの基本設計自体が変わる、たとえばデータ形式そのものがXMLからJSONに変更された例も存在する。

Enterprise Connectorsは、こういったベンダー側のAPI仕様をラッピングし、FireDACの共通インターフェースで活用できるようにした。これにより各種クラウドサービスに対して、IBM iやオープン系のDBへのアクセスと同様に、SQLやストアドプロシージャを使用してアクセスできる。【図1】

Enterprise Connectorsのサブスクリプションには、主要な約80種類のクラウドサービスに接続可能な「Enterprise」版と、約150種類のクラウドサービスに接続可能な「Enterprise Plus」版が存在する。

本稿ではこれらの中から代表的なものとして、業務に限らず広く使われているTwitterおよびGoogleスプレッドシートを取り上げ、Delphi/400と連携可能な利用例について紹介する。

これらの連携を使用すれば、たとえば「IBM i上の売上データをもとにGoogle Drive上にGoogleスプレッドシートを作成する」「IBM i上の余剰在庫情報を

図1　Enterprise Connectorsの概要

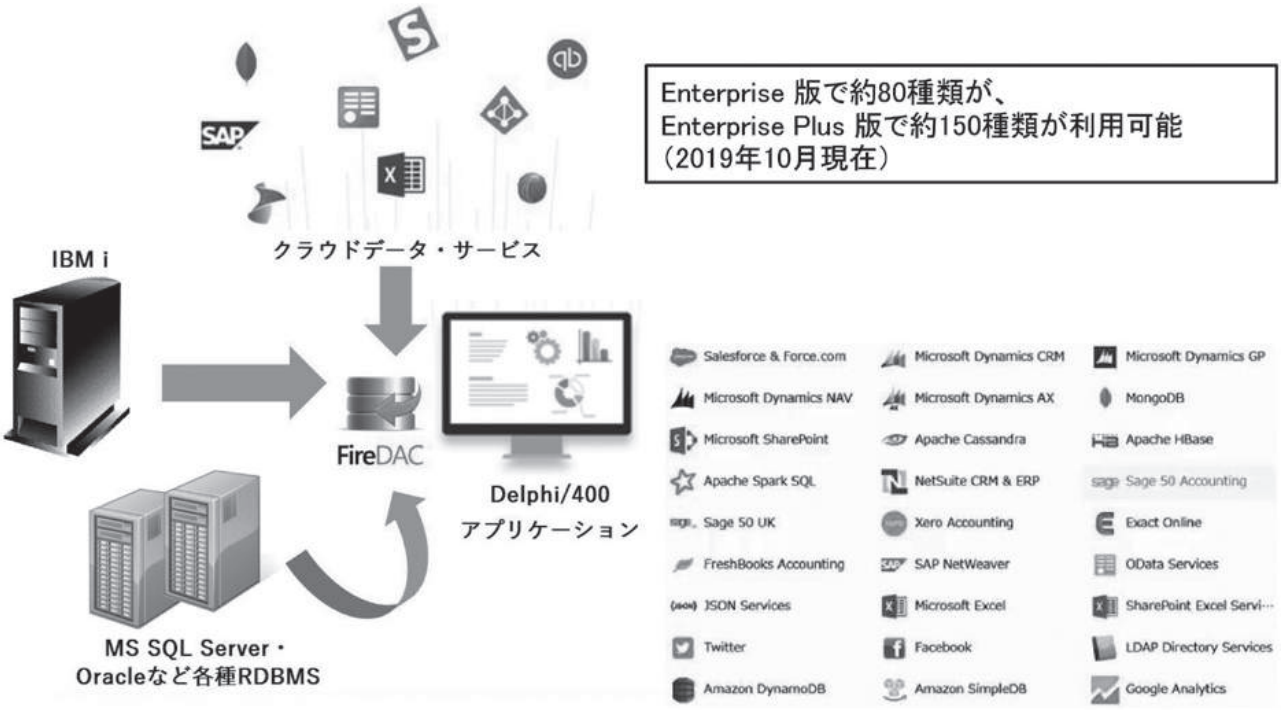
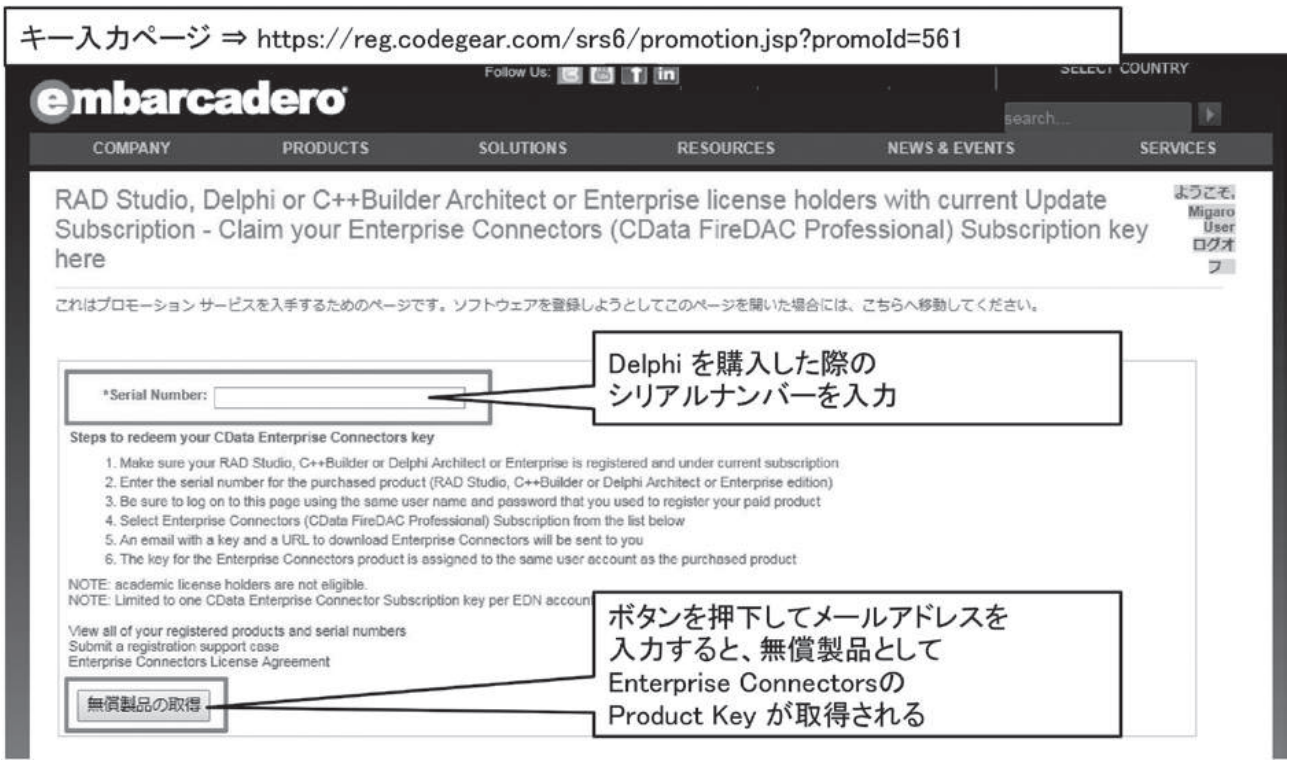


図2　Enterprise Connectorsの取得



もとに、Twitterにキャンペーン情報をツイートする」といった使い方ができる。

なおEnterprise Connectorsは、Delphi/400 Version 10.2 Tokyoからの対応となっており、本稿のサンプルでも同バージョンを使用している。また本稿で扱うそれぞれのコネクターは、「Enterprise」版のサブスクリプションで利用可能である。

3. Enterprise Connectorsのインストール

Enterprise Connectorsは、Delphi/400のオプションのライセンス製品であるが、アップデートサブスクリプションが有効なDelphi/400 10.2 Tokyoをご利用であれば「Enterprise」版を無償で取得できる。

まず、エンバカデロ・テクノロジーズ社のサイト（https://reg.codegear.com/srs6/promotion.jsp?promoId = 561）にアクセスし、Delphi購入時のシリアルナンバーを入力してProduct Keyを取得する。【図2】

次にCData Software社のサイト（https://www.cdata.com/firedac/download/）にアクセスし、必要なコネクターを選択すると【図3】のような画面が表示される。それぞれのコネクターは独立しており、コネクターごとに専用のインストーラと、それによってインストールされる専用のコンポーネントが個別に存在する。

先ほどのProduct Keyとそれを取得するために使用したメールアドレスを入力すると、対象のコネクターのインストーラがダウンロードされる。

インストールが完了すると、ヘルプドキュメントがスタートメニューに登録される。そのコネクター関連の各コマンドや使用方法が英語で記載されているので、開発時の大きな助けになるだろう。【図4】

4. Enterprise Connectorsの開発方法

本章では、各種コネクターの中でもシンプルで扱いやすい「CData CSV FireDAC Components」を使用して、CSVファイルにアクセスする方法を紹介する。

CSVといえば、通常はTStringListなどで読み書きすることが多いが、Enterprise Connectorsの使用によって、CSVデータに対して、一般的なDBアクセスと同様にSQLを使用したデータの抽出や更新が可能になるメリットがある。

それでは、具体的なサンプル作成を通じて開発方法を紹介する。今回のサンプルでは「CSVファイルを読み込み、TDBGridに表示」「IBM iから取得したデータをもとにCSVファイルにレコードを追加」の2つの機能を実現する。

Delphi/400 10.2 Tokyoを起動したら、アプリケーションを新規作成し、新しいフォームの上にコンポーネントを配置する。

まずEnterprise ConnectorsでCSVに接続してデータを読み書きするためのコンポーネントとして、「CData CSV FireDAC Components」をインストールした際に追加されるコネクター「TFDPhysCDataCSVDriverLink」を配置する。

これはIBM iへの接続におけるTFDPhysCO400DriverLinkに相当し、Enterprise Connectorsはコネクターごとのドライバリンクコンポーネントを定義するのがポイントである。また通常のFireDAC接続と同様にTFDConnection・TFDQueryと、取得結果を画面に表示するTDataSource・TDBGridを配置する。

次にIBM iに接続してデータを取得するコンポーネント群として、TFDConnection・TFDQueryおよび、IBM iに接続するためのコネクターとなるTFDPhysCO400DriverLinkを配置する。

あとは処理実行のためにTEditやTButtonなどを配置した結果、【図5】のような画面になる。

そしてTFDConnectionをダブルクリックして、FireDAC接続エディタを開き、ドライバIDに「CDataCSV」を選択する。すると【図6】のように、接続時の各種パラメータが自動でセットされる。

ローカルのCSVファイルに接続する場合は、「URI」パラメータに対象CSVのフォルダを指定する。また「UseRowNumbers」や「RTK」といったパラメータを【図6】のように設定する。

次に、接続するCSVを準備する。本稿のサンプルでは、【図7】のようなレイアウトで「WORKCSV.csv」というサンプルCSVを作成した。CSVのファイル名は､拡張子を除いた「WORKCSV」部分がテーブル名に、1行目のカンマ区切りテキストがそのままフィールドIDに、2行目以降のカンマ区切りテキストがレコードの内容となる。

(1) CSVファイルを読み込み、TDBGridに表示

画面設計が完了したら、【ソース1】のようにロジックを記述する。プログラムを実行して「①SQL SELECT」ボタンを押下することで、CSVの内容がDBとして明細に表示される。【図8】のTDBGridの内容は、単純な全件取得の結果であるが、一部フィールドだけの取得や、WHERE句を使用した絞り込みも可能である。

FireDAC接続エディタの設定で、「UseRowNumbers」をTrueに指定して書き込みを有効にした場合、行番号で各レコードをユニークキーとするための「RowNumber」というフィールドも追加される。【図8】では明示的に表示されたままにしているが、通常は列ごと非表示に設定しても問題ない。

(2) IBM iから取得したデータをもとにCSVファイルにレコードを追加

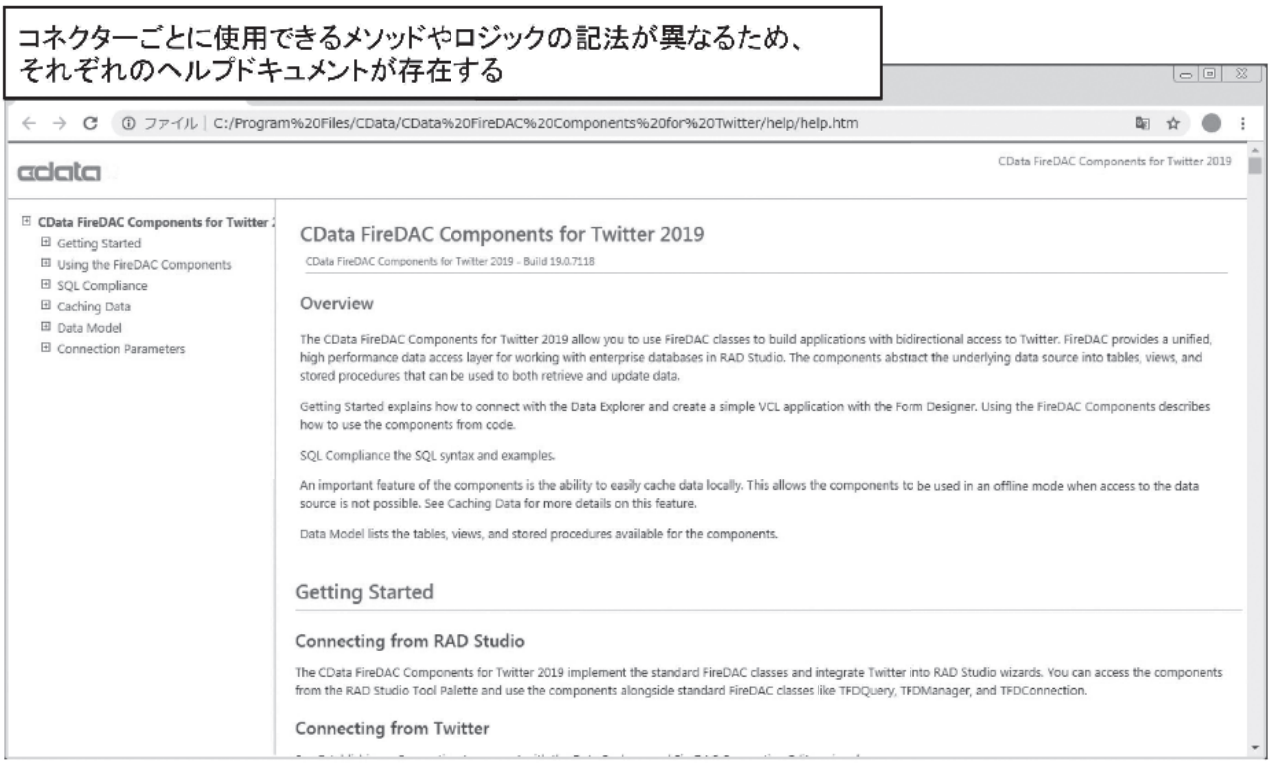
次に、【ソース2】のようにロジックを記述する。顧客コードに値を入力して「②登録」ボタンを押すと、IBM iから条件に合致するデータを取得し、その結果をSQLのINSERT文を使用してCSVに行を追加できる。【図9】

5. クラウドサービスへの接続･活用方法

前章では、Enterprise Connectorsの基本的な開発について紹介したが、ここでは具体的なクラウドサービスの連携例として、業務に限らず広く一般でも使用されるTwitterおよびGoogle Apps（Googleスプレッドシートおよび Google Drive）について紹介する。

図3　Enterprise Connectorsのインストール

図4　Enterprise Connectorsのヘルプドキュメント



5-1. Twitterと連携した情報発信と解析

最初に、有名人・著名人も数多く利用するソーシャルメディアの1つで、Google検索でもヒットしないような口コミをリアルタイムで調べられるTwitterの接続、利用方法を紹介する。

(1) Twitter APIのアプリケーション開発準備

まず、Twitterの開発者サイト（https://dev.twitter.com/apps）から、新しいTwitter Web APIを作成する。開発者サイトにログインしたら、「Create an app」ボタンを押し【図10】、作成するアプリケーションのタイトルや使用目的を英語で入力する。【図11】

入力が完了したら、Twitterアプリケーション開発に必要なAPIキーとアクセストークンを入手できる。【図12】【図13】

なお開発者サイトを利用するにあたっては、Twitterの開発者アカウントの作成が必要である。本稿では作成手順については割愛するが、Google等で「Twitter 開発者アカウント」と検索すれば、その時点での最新情報が参照できるだろう。

(2) Twitter APIを使用したツイート送信方法

APIを作成したら、次はDelphi/400側の開発を行う。ここでは、「CData Twitter FireDAC Components」を利用する。Delphi/400 10.2 Tokyoを起動したら、アプリケーションを新規作成し、新しいフォームに【図14】のようにコンポーネントを配置する。

前章のCSV接続と同様、Delphi/400でIBM iに接続してデータを取得するコンポーネント群と、Enterprise ConnectorsでTwitterに接続するコンポーネント群をそれぞれ配置する。なお、Twitterで使用するコネクターは、「TFDPhysCDataTwitterDriverLink」である。

前章のCSV接続と同じように、TFDConnectionをダブルクリックしてFireDAC接続エディタを開き、ドライバIDに「CDataTwitter」を選択すると、【図15】のように接続時の各種パラメータが自動でセットされる。

ここでセットが必要なパラメータは、接続目的にあわせて【図16】のように設定する。また「②選択行の商品についてツイートする」ボタン押下時の処理を、【ソース3】のように記述する。TFDQueryを使用して、テーブル「Tweets」にSQLでレコードを追加するだけで、ツイートが送信できる。

Delphi/400でIBM iからデータを取得し、【図17】のようにTDBGridに表示した状態から、「②選択行の商品についてツイートする」ボタンを押下すると、【図18】のようにツイートが送信される。送信されたツイートにはユニークキーとなるID（数字18～19桁）が文字型で採番されており、ブラウザのURLで確認できる。

送信時のSQLにより、「In_Reply_To_Status_Id」フィールドに別のツイートのIDをセットすると、そのツイートへのリプライになる。またツイートの本文中に別のツイートのURL「https://twitter.com/XXX/status/（ツイートのID)」を記載すると、そのツイートに対する引用リツイートになる。

本稿の例ではSQLのINSERT文でツイートを送信したが、逆にSQLのDELETE文を発行することで、自分が送信したツイートを削除することも可能である。その際には、対象ツイートのIDをWHERE句に指定する。

なお、それぞれのツイートはFireDAC接続時に認証を受けたユーザーのアカウントで発信されるため、当然のことながら他人のアカウントになりすますことはできない。

(3) Twitter APIを使用した各種データの参照方法

次に、Twitter APIを使用して各種テーブルや、キーワードを指定したツイートを検索するアプリケーションを作成する。

Delphi/400 10.2 Tokyoでアプリケーションを新規作成し、新しいフォームに【図19】のようにコンポーネントを配置する。【ソース4】のようにロジックを記述し、プログラムを実行する。「①タイムラインの表示」ボタンを押下すると、「SELECT * FROM Tweets」というSQLが発行され、自分（ログイン中のユーザー）と自分がフォローしているユーザーのツイートが一覧で表示される。【図20】

「②フォローしているユーザーを表示」「③自分がいいねを押した投稿を表示」「④フォロワーの一覧を表示」の各ボタン押下時の結果についても、【図20】で示したように参照できるので、ぜひ一度試していただきたい。

さて、ここで使用したテーブル「Tweets」は、タイムラインを表示するだけのテーブルではない。キーワードを指定して検索する機能も持っている。

【ソース5】のようにロジックを記述する。WHERE句にフィールド「Search Terms」を指定することで、指定されたキーワードを含む直近の6～9日以内のツイートを検索できる。【ソース5】のロジックでは、影響度が高いツイートを絞り込むため、「リツイート数が一定以上」「いいね数が一定以上」「日本語のツイートに限定」という条件も付けられるようにしている。

【図21】の明細は「Enterprise Connectors」というキーワードで、Twitter検索を行った結果である。

検索結果は、レコードの作成日時が新しい順（ツイートが新しい順、最近フォローされた順など）や、Twitterが独自に判断した影響度順で表示されており、ORDER BY句やインデックスの指定はできない。Twitter APIには、一定時間あたりの通信回数に上限が存在するからである。一定回数を超えて通信を行うと、【図22】のようなエラーが表示され、しばらく接続不能になる。

たとえばORDER BY句をSQLで指定すること自体は可能だが、そのSQLを実行すると、世界中からリアルタイムで流れ込む大量のツイートを取得・選別し続け、結果を表示するより先に、数秒～数十秒のうちに通信回数の上限に到達してエラーになる。

データを並べ替えたい場合は、TStringListやTClientDataSetなどのFireDAC接続から切り離されたローカルキャッシュ内に一度データを格納してから行うとよい。

なお、FireDACのデフォルト設定ではデータを1回の通信で50件ずつ取得するため、キーワードに対する検索結果が多い場合は、数分前までの新しいツ

図5　CSV操作PGM①（画面設計）

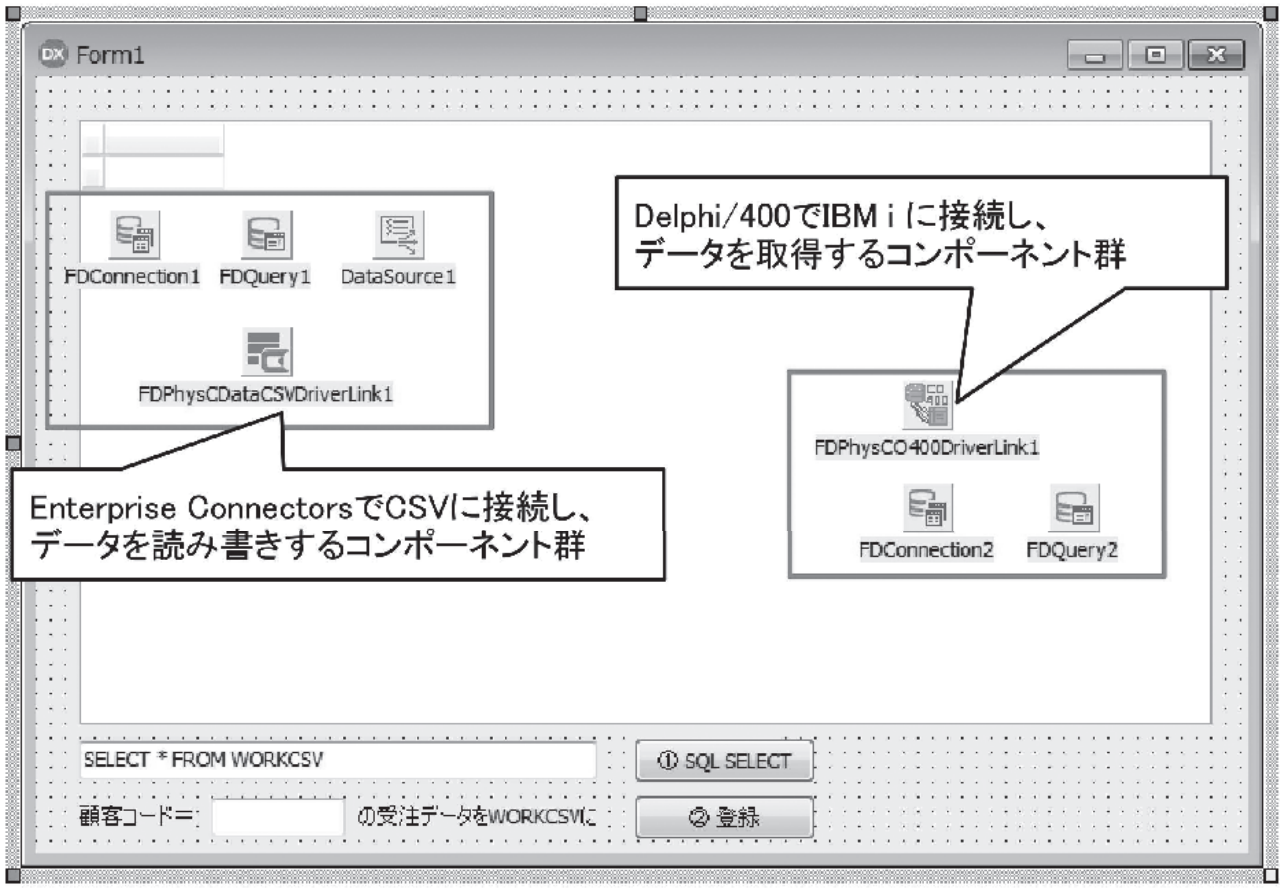
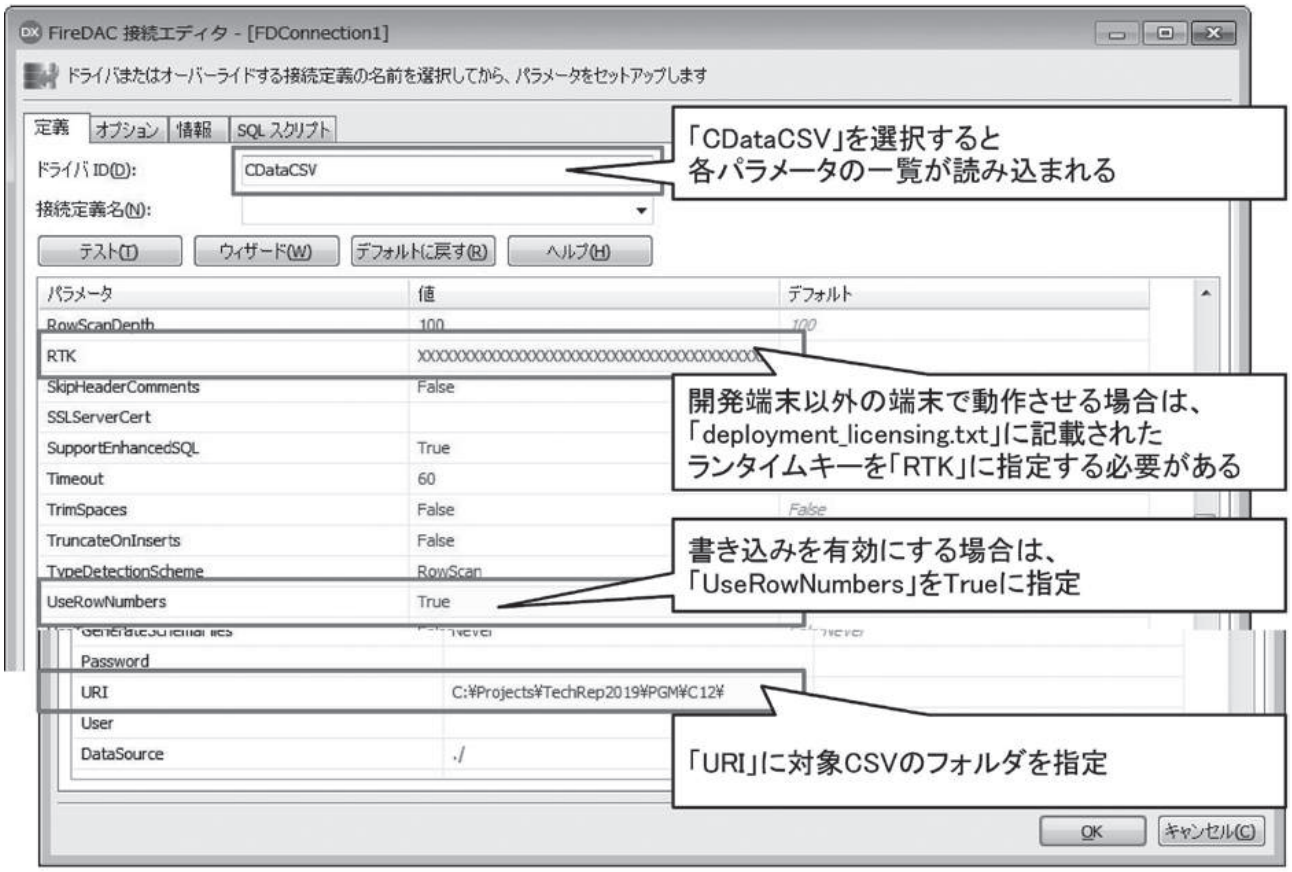


図6　CSV操作PGM②（接続パラメータ設定）



イートしか表示されないこともある。

このような場合には、【ソース 5】で指定したように、TFDQuery の FetchOptions.RowsetSize の設定値を増やすことで、1 回の通信あたりの取得レコード数が増え、より多くの取得結果が得られる。

ここまで Twitter API を利用したデータ送信およびデータ参照の手順を紹介したが、CData Twitter FireDAC Components では、目的に合わせたデータを解析できるよう、さまざまなテーブルやビューが準備されている。代表的なものを、【図 23】に記す。

5-2. Googleスプレッドシートへのデータ出力

次に、「CData Google Sheets FireDAC Components」を使用して、IBM i の売上データを「Google スプレッドシート」に出力する例を紹介する。また、出力した Google ドライブ上のスプレッドシートを、ローカルファイルとしてダウンロードする方法もあわせて紹介する。

これまで IBM i のデータを出力する場合、ローカル PC 上に CSV ファイルや Excel ファイルで出力することが多かったが、クラウド上の Google スプレッドシートにデータが出力できれば、ファイルの共有や同時編集など利便性が向上する。

Delphi/400 10.2 Tokyo を起動したら、アプリケーションを新規作成し、新しいフォームに、【図 24】のようにコンポーネントを配置する。

前章と同様、Delphi/400 で IBM i に接続してデータを取得するコンポーネント群と、Enterprise Connectors で Google スプレッドシートに接続するコンポーネント群をそれぞれ配置する。

なお Google スプレッドシートで使用するコネクターは、「TFDPhysCDataGoogleSheetsDriverLink」である。

前章と同じように、TFDConnection をダブルクリックして FireDAC 接続エディタを開き、ドライバ ID に「CDataGoogleSheets」を選択すると、【図 25】のように接続時の各種パラメータが自動でセットされる。本稿では、現在使用中の PC でログインしている自身の Google アカウントでの接続を前提とするため、【図 25】に記載の設定のみ行えばよい。

FireDAC 接続エディタの画面で、設定変更後に「テスト」ボタンを押すと、ブラウザが起動して【図 26】のような認証画面が表示されるので、このタイミングで Google アカウントへの権限を許可しておく（認証が必要なのは初回接続時のみ）。

この認証により、各処理はログイン中の自身の Google アカウントとして行われる。したがって、完成したアプリケーションを別の Google アカウントでログインした別の PC から起動した場合、初回接続時に同じようにブラウザが立ち上がり、【図 26】の認証画面が表示される。

本稿のサンプルでは、前章で CSV にデータを出力した際と同じデータを使用して、【ソース 6】のようにロジックを記述する。

スプレッドシートの新規作成では、SQL ではなくストアドプロシージャを発行している。Enterprise Connectors では、データの読み書きだけではなく、ファイルの作成や削除、ダウンロードなどの処理が行えるようサブメソッドがコネクターごとに用意されており、使用できる。

TFDQuery でもストアドプロシージャの発行は可能だが、【ソース 6】のように TFDStoredProc で発行するのがシンプルで効率がよい。Google スプレッドシートの操作には、今回使用したスプレッドシートの新規作成以外にも、シートの追加・コピー・削除や書式設定など、ストアドプロシージャで実現できるメソッドが数多く存在するので、ぜひヘルプドキュメントで詳細を参照いただきたい。

また本稿のプログラムでは、出力先スプレッドシートの「A」～「F」という列番号がそのままフィールド ID となっており、SQL で INSERT 文を発行する際にもそのように記述する。

完成したプログラムを実行して、「①出力」ボタンを押すと、ログイン中の Google アカウントが保持する Google スプレッドシートの一覧にファイルが新規作成され、その中に IBM i から取得したデータがセットされる。【図 27】【図 28】

また、【ソース 7】のようにダウンロードボタンのロジックを記述することで、このスプレッドシートをプログラム内から直接（ブラウザを起動せずに）、xlsx や PDF といった形式でダウンロードできる。【図 29】【図 30】

ここでも、ダウンロードの実行にはストアドプロシージャを使用している。

5-3. Google Driveでのデータ検索

最後に、「CData Google Drive FireDAC Components」を使用した、Google Drive のファイル検索について紹介する。

Delphi/400 10.2 Tokyo を起動したら、アプリケーションを新規作成し、新しいフォームに【図 31】のようにコンポーネントを配置する。

このプログラムでは IBM i と接続しないため、Enterprise Connectors で Google Drive に接続するコンポーネント群のみをそれぞれ配置する。なお Google Drive で使用するコネクターは、「TFDPhysCDataGoogleDriveDriverLink」である。

前章までと同じように、TFDConnection をダブルクリックして FireDAC 接続エディタを開き、ドライバ ID に「CDataGoogleDrive」を選択すると、【図 32】のように接続時の各種パラメータが自動でセットされる。

テスト接続時では、前章の Google スプレッドシートと同様に、【図 26】のような Google アカウントの認証画面が初回のみ表示されるので、権限を許可する。

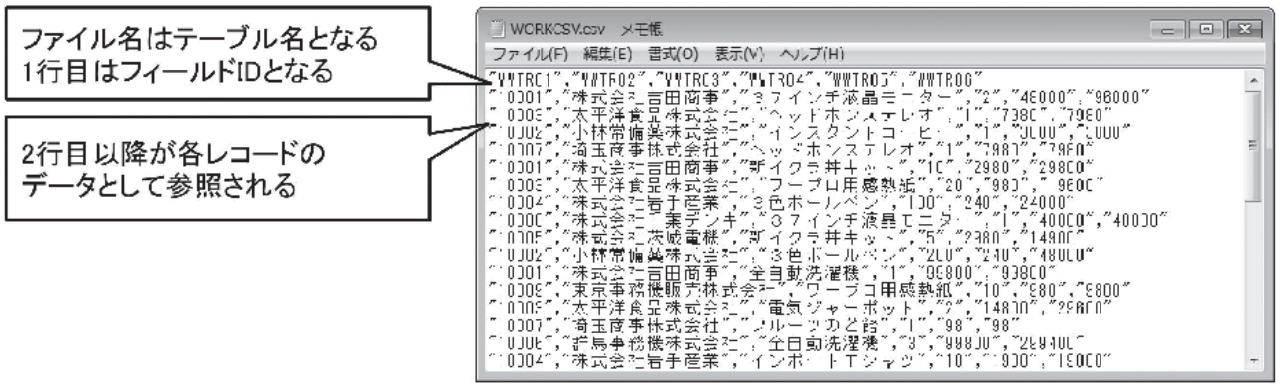
次に、【ソース 8】のようにロジックを記述する。

Google Drive の検索機能では、通常のファイル名やファイルの作成日時などによる絞り込みだけでなく、ファイルの内容を含むフルテキスト検索にも対応している。

たとえば前項で作成した Google スプレッドシートで、顧客名に「株式会社吉田商事」という値をセットしたセルが存在するが、Google Drive 側で検索条件に「吉田商事」と入力することで、セルの内容に「吉田商事」が含まれるスプレッドシートを検索結果として表示させられる。【図 33】

これは Google スプレッドシートのファイル保存先が Google Drive になっ

図7　CSV操作PGM③（CSVファイル設計）



ソース1

FireDAC接続したCSVからSELECTでデータ抽出

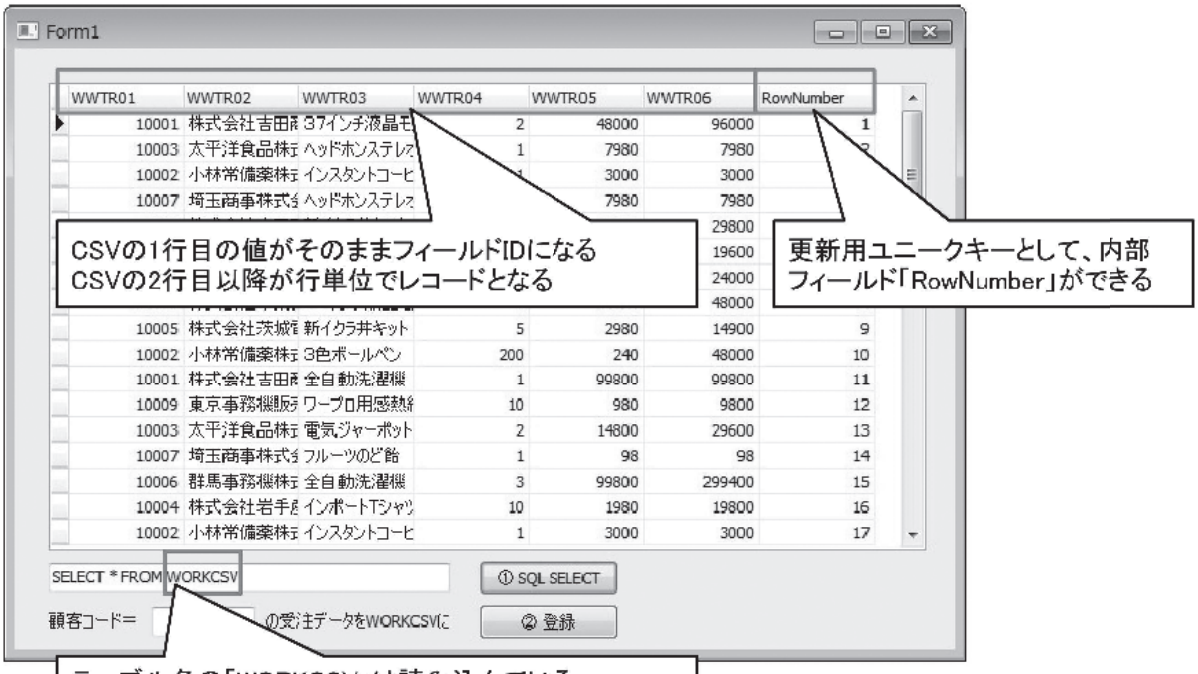
```
{*******************************************************************************
 目的：①SQL SELECTボタン 押下時処理
*******************************************************************************}
procedure TForm1.Button2Click(Sender: TObject);
var
  i: Integer;
Begin

  FDQuery1.Close;
  FDQuery1.SQL.Text := Edit1.Text;   // SELECT * FROM WORKCSV
  FDQuery1.Open;

  // 列幅の調整
  for i := 0 to (DBGrid1.Columns.Count - 1) do
  begin
    DBGrid1.Columns.Items[i].Width := 80;
  end;
end;
```

この3行だけでSELECTの
SQLを発行し、DBと同じように
CSVの内容を参照できる

図8　CSV操作PGM④（SELECTで内容表示）



ているためで、同じGoogleアカウントであれば、Google Drive上からもスプレッドシートの内容を参照できる。

前章で、スプレッドシートを新規作成した際に採番されたファイルのIDもGoogle Driveと共通なので、そのIDを含んだGoogle DriveのファイルURLを指定して実行することで、Googleスプレッドシートで対象のファイルが起動する。【ソース9】【図34】

Google Driveではこれ以外にも、各種SQLやストアドプロシージャを使用することで、ファイルのアップロード、ダウンロードやファイル名の変更・削除なども行える。これらの方法もそれぞれヘルプドキュメントに記載されているので、ぜひ一度挑戦していただきたい。

6. まとめ

本稿では、Enterprise Connectorsを使用したクラウドサービスとの連携テクニックについて紹介した。

今回はTwitter、Googleスプレッドシート、Google Driveという一般によく使用されるクラウドサービスを題材に連携方法を紹介したが、Enterprise Connectorsでは他にもSalesforceやkintoneといったエンタープライズ向けクラウドサービスや、MongoDB、Amazon DynamoDBといったようなNoSQLデータベースなど、これからの業務システムに多用する多彩なサービスへのアクセスを可能にしている。

本稿を参考に、さまざまなクラウドサービスとの連携を検討いただきたい。なお、全コネクターが利用可能なEnterprise Connectors Plusの購入をご検討の場合は、弊社営業までお気軽にお問い合わせいただければ幸いである。

M

ソース2

FireDAC接続したCSVにレコードを追加

```
{*****************************************************************************
 目的: ②登録ボタン 押下時処理
*****************************************************************************}
procedure TForm1.Button1Click(Sender: TObject);
var
  iTR01, iTR04, iTR05, iTR06: Integer; // 顧客コード、数量、単価、金額
  sTR02, sTR03: String;                // 顧客名、商品名
begin

  // Delphi/400側のQueryで対象データを抽出
  FDQuery2.Close;
  FDQuery2.SQL.Clear;
  FDQuery2.SQL.Add(' SELECT VFORDP.*, CSCSNM FROM VFORDP ');
  FDQuery2.SQL.Add(' LEFT JOIN VMCSTP ON (CSCSCD = ORCSCD) ');
  FDQuery2.SQL.Add(' WHERE VFORDP.ORCSCD = ' + Edit2.Text); // 顧客コード
  FDQuery2.Open;

  try
    // Enterprise Connectors側のQueryでワークファイルCSVに更新処理
    FDQuery1.SQL.Clear;
    FDQuery1.SQL.Add(' INSERT INTO WORKCSV ( ');
    FDQuery1.SQL.Add(' WWTR01, WWTR02, WWTR03, WWTR04, WWTR05, WWTR06 ');
    FDQuery1.SQL.Add(' ) VALUES ( ');
    FDQuery1.SQL.Add(' :TR01, :TR02, :TR03, :TR04, :TR05, :TR06 ');
    FDQuery1.SQL.Add(' ) ');

    while not FDQuery2.Eof do
    begin
      iTR01 := FDQuery2.FieldByName('ORCSCD').AsInteger;  // 顧客コード
      sTR02 := FDQuery2.FieldByName('CSCSNM').AsString;   // 顧客名
      sTR03 := FDQuery2.FieldByName('ORPRNM').AsString;   // 商品名
      iTR04 := FDQuery2.FieldByName('ORQTY').AsInteger;   // 数量
      iTR05 := FDQuery2.FieldByName('ORUNPR').AsInteger;  // 単価
      iTR06 := iTR04 * iTR05;                             // 金額

      FDQuery1.Close;
      FDQuery1.ParamByName('TR01').AsInteger    := iTR01; // 顧客コード
      FDQuery1.ParamByName('TR02').AsWideString := sTR02; // 顧客名
      FDQuery1.ParamByName('TR03').AsWideString := sTR03; // 商品名
      FDQuery1.ParamByName('TR04').AsInteger    := iTR04; // 数量
      FDQuery1.ParamByName('TR05').AsInteger    := iTR05; // 単価
      FDQuery1.ParamByName('TR06').AsInteger    := iTR06; // 金額
      FDQuery1.ExecSQL; // 更新実行

      FDQuery2.Next;    // Delphi/400側のQueryを次行へ
    end;

  finally
    FDQuery1.Close;
    FDQuery2.Close;
  end;
end;
```

IBM i からデータを抽出

CSVにデータを
登録するためのSQL文作成

CSVにデータを書き込み

図9　CSV操作PGM⑤（CSVへのINSERT）

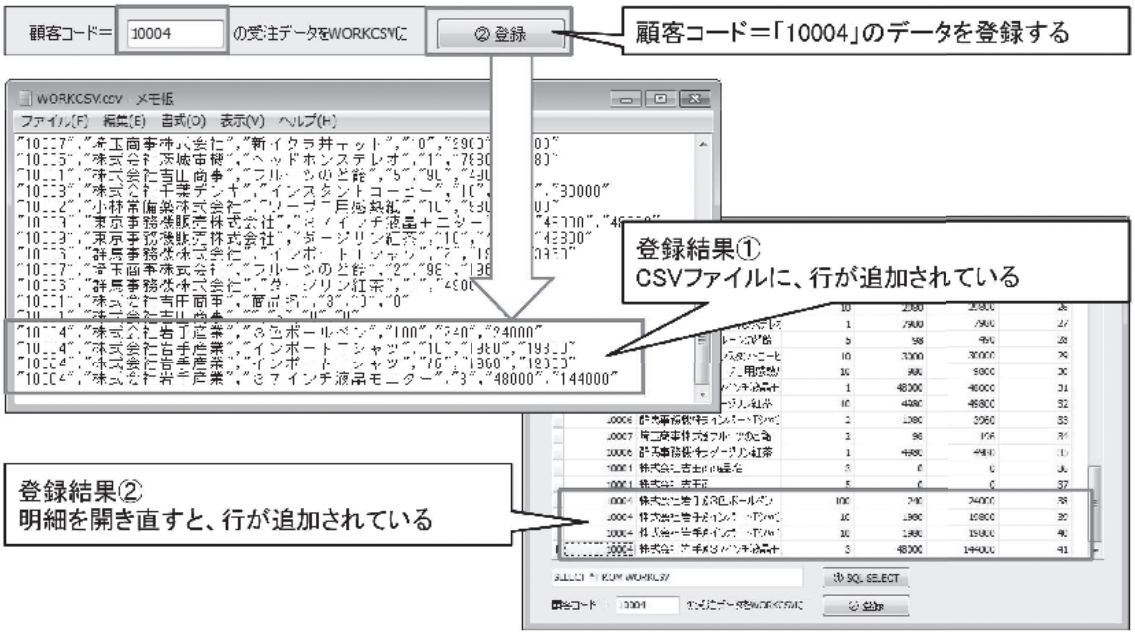


図10　Twitter Web APIの利用開始①

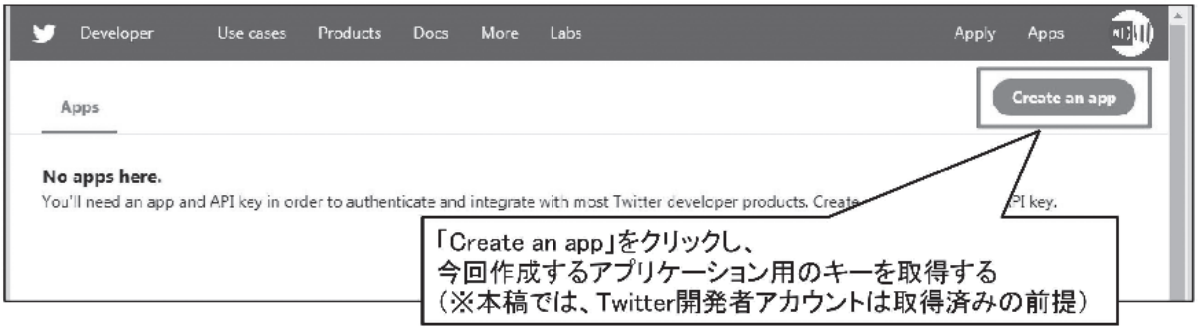


図11　Twitter Web APIの利用開始②

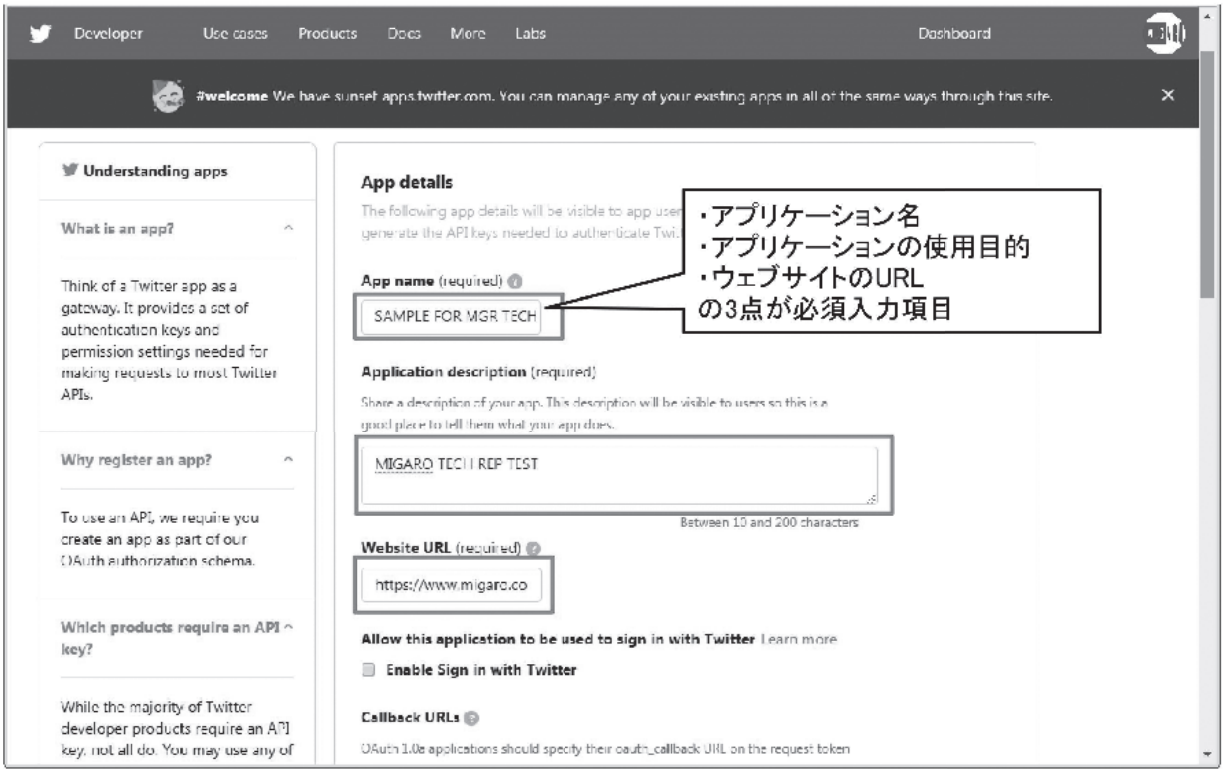


図12　Twitter Web APIの利用開始③

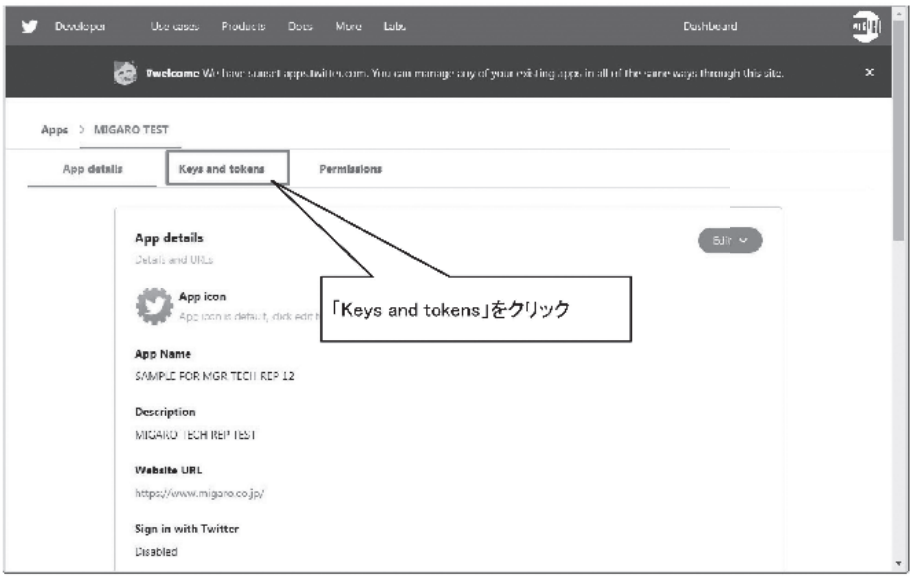


図13　Twitter Web APIの利用開始④

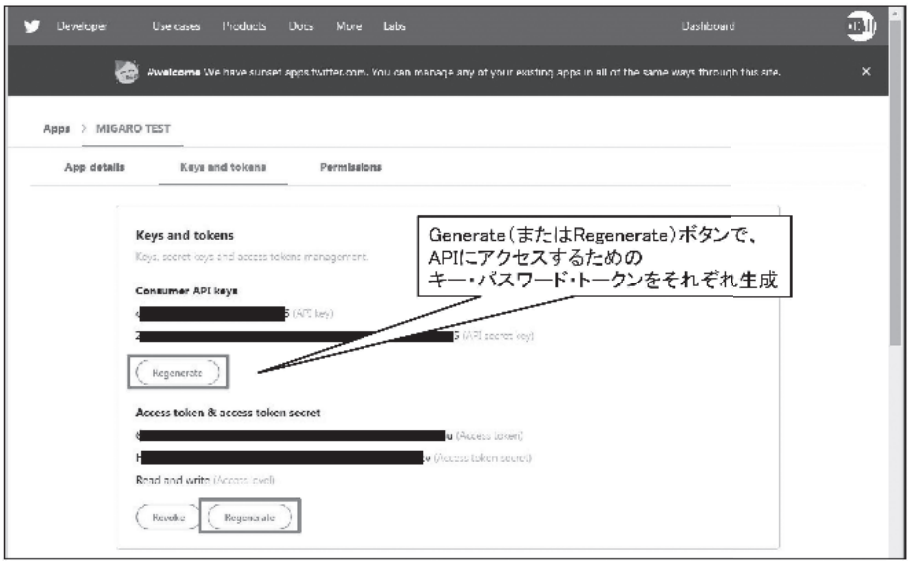


図14　Twitterのデータ送信①（画面設計）

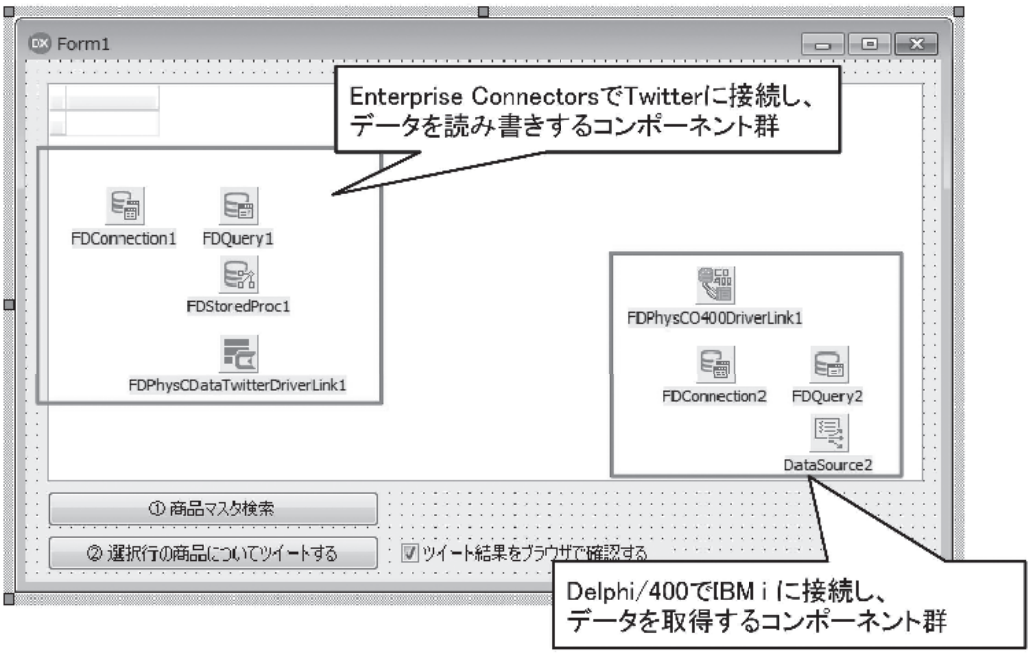


図15　Twitterのデータ送信②（接続パラメータ設定）

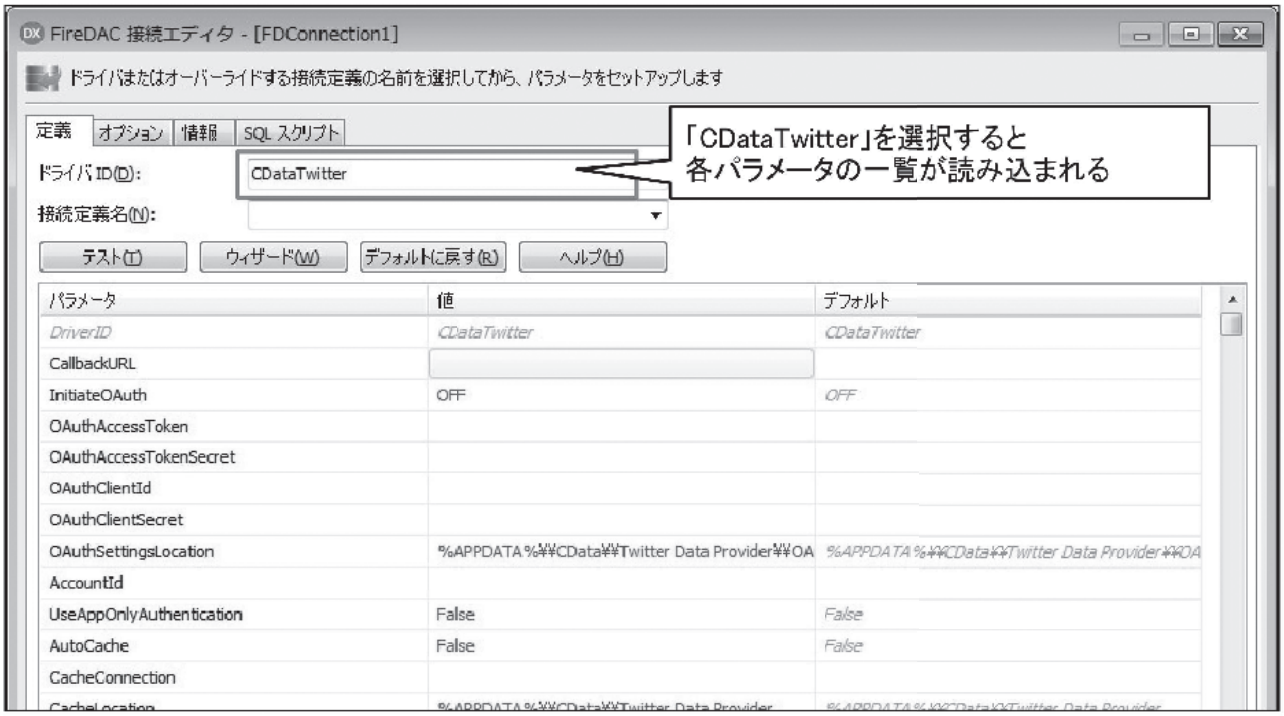


図16　Twitterのデータ送信③（接続パラメータ設定）

＜目的別 TFDConnectionのパラメータ設定値＞　　※「－－－」はデフォルト値のまま

接続目的 / パラメータ名	書き込み／削除を行う		照会のみ（書き込み／削除不要）
	開発者自身のアカウントでのみ書き込み／削除を行う	あらゆるアカウントでログインして書き込み／削除を行う	
OAuthClientId	APP作成時に取得した「API key」		
OAuthClientSecret	APP作成時に取得した「API secret key」		
RTK	開発端末以外の端末で動作させる場合は、以下の場所に記載されたランタイムキーを指定 C:¥Program Files¥CData¥CData FireDAC Components for Twitter¥deployment_licensing.txt		
UseAppOnlyAuthentication	－－－	－－－	True
CallbackURL	－－－	APP作成時に指定する「Callback URL」	－－－
InitiateOAuth	－－－	GETANDREFRESH	－－－
OAuthAccessToken	APP作成時に取得可能な「Access token」	接続時に都度関数で取得（詳細はヘルプドキュメントを参照）	－－－
OAuthAccessTokenSecret	APP作成時に取得可能な「Access token secret」		－－－
結果	開発者自身のアカウントでログインし、読み書きする	ブラウザが起動してAPIの認証画面が表示され、そこでログインしたアカウントで読み書きする	照会専用アカウントでログイン（ユーザー固有の情報は読み取れない）

ソース3

Twitter APIを利用したツイートの作成

```
{*****************************************************************************
 目的：②選択行の商品についてツイートする
*****************************************************************************}
procedure TForm1.Button2Click(Sender: TObject);
var
  sTEXT: String;
begin

  // 未検索時は何もしない
  if (not FDQuery2.Active) or (FDQuery2.Bof and FDQuery2.Eof) then
    Exit;

  // 本文の作成
  sTEXT := '今週のおすすめ商品！！『' +
           FDQuery2.FieldByName('SYHNMM').AsString +
           '』　単価：¥' +
           FormatFloat('#,0', FDQuery2.FieldByName('TANKA').AsInteger) +
           '　詳細はぜひ、ホームページまでアクセスしてください。 '+
           'http://technical-report.jp  #Delphi #Technical_Report';

  // つぶやく
  FDQuery1.SQL.Text := 'INSERT INTO Tweets (Text) VALUES (:TEXT)';
  FDQuery1.ParamByName('TEXT').AsMemo   := sTEXT;    // 本文
  FDQuery1.ExecSQL;

  // 自身の最新ツイートを取得してリンクを開く
  if CheckBox1.Checked then
  begin
    GetMyNewest;
  end;
end;

{*****************************************************************************
 目的：自分の最新ツイートをブラウザで開く
*****************************************************************************}
procedure TForm1.GetMyNewest;
var
  sMe, sID, sURL: String;
begin

  FDQuery1.Close;
  FDQuery1.SQL.Clear;
  FDQuery1.SQL.Add(' SELECT A.ID AS TWID, A.Text, A.From_User_Screen_Name AS MYNAME ');
  FDQuery1.SQL.Add(' from Tweets A ');
  FDQuery1.SQL.Add(' inner join AccountSettings B ');  // 自分のアカウント設定ビューをJOINする
  FDQuery1.SQL.Add(' on (B.Screen_Name = A.From_User_Screen_Name) ');  // 送信者＝自分に絞る
  try
    // 取得結果は常に新しい順なので、ORDERは不要(通信回数制限の原因になる)
    FDQuery1.Open;
    if not(FDQuery1.Bof and FDQuery1.Eof) then
    begin
      sID  := FDQuery1.FieldByName('TWID').AsString;
      sMe  := FDQuery1.FieldByName('MYNAME').AsString;
      sURL := 'https://twitter.com/' + sMe + '/status/' + sID;

      // ShellExecuteによって、通常使うブラウザでURLを開く
      // (※uses節に Winapi.ShellAPI が必要)
      // (※ツイートを直接表示するには、ブラウザ側がログイン状態になっている必要あり)
      ShellExecute(Application.Handle, 'OPEN', PChar(sURL), PChar(''), '', SW_SHOW);
    end;
  finally
    FDQuery1.Close;
  end;
end;
```

※①商品マスタ検索ボタンの処理は通常のDelphi/400の検索処理と同様のロジックのため、省略

本文の文字列に指定した内容が送信される

ハッシュタグやURLは本文中にそのまま記述する（複数ハッシュタグ使用時はスペースを空ける）

この行をエラー無く抜けたら、送信完了

今送信したツイートのIDと自分のアカウントIDを取得し、リンクさせるURLを生成する

図17　Twitterのデータ送信④（商品マスタ検索）

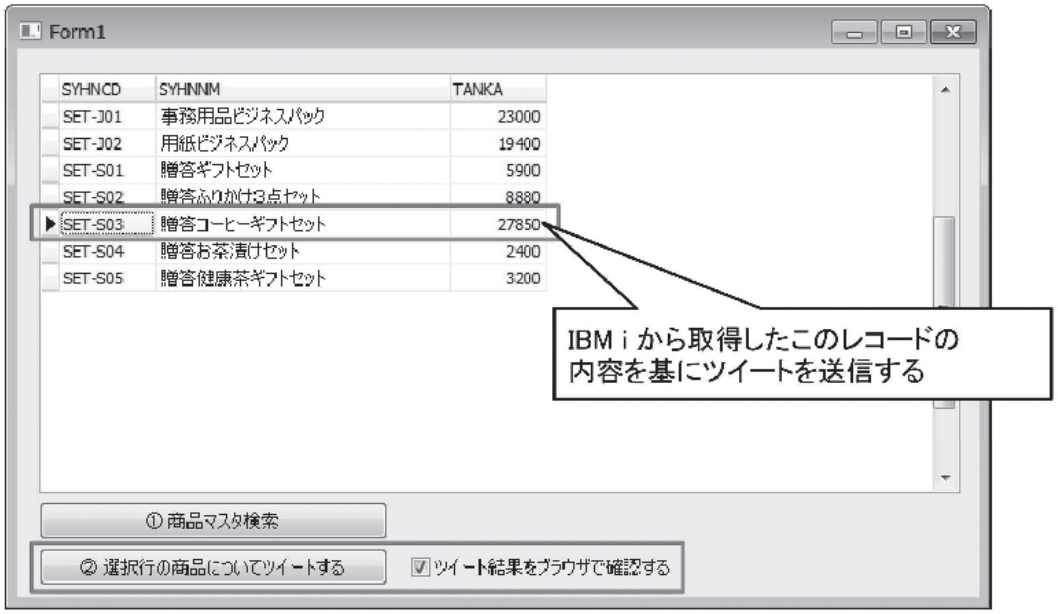


図18　Twitterのデータ送信⑤（ツイート送信結果）

図19　Twitterのデータ参照①（画面設計）

ソース4

Twitter APIを利用したツイートやアカウント情報の取得

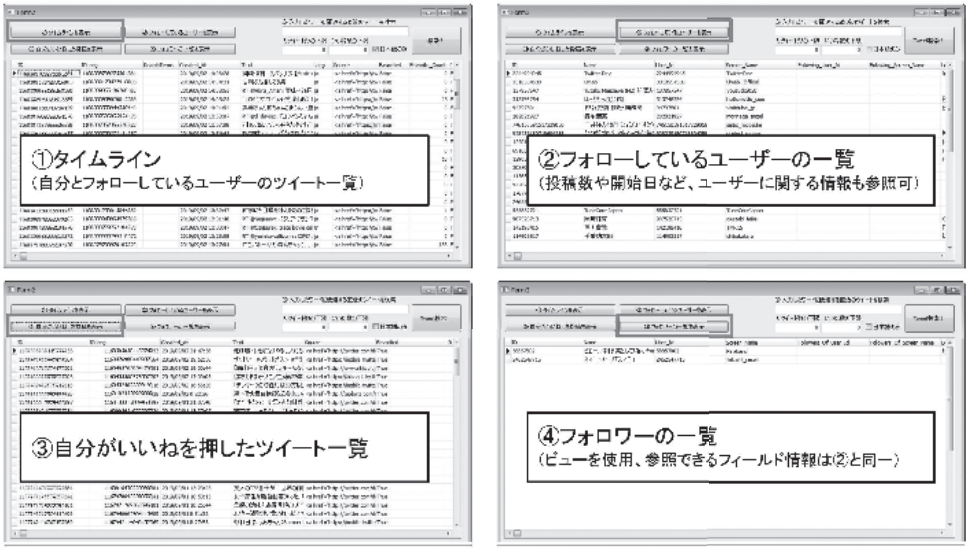
```
{**************************************************************************
 目的：①タイムラインを表示
**************************************************************************}
procedure TForm2.Button1Click(Sender: TObject);
begin
  // 自分および自分がフォローしているユーザーのツイートを一覧表示
  FDQuery1.Close;
  FDQuery1.SQL.Text := 'SELECT * FROM Tweets';
  FDQuery1.Open; // SQL実行
  SetColWidths;  // DBGridの列幅設定
end;

{**************************************************************************
 目的：②フォローしているユーザーを表示
**************************************************************************}
procedure TForm2.Button2Click(Sender: TObject);
begin
  // 自分がフォローしているユーザーの一覧表示
  // (※MaxUserLookup=最大取得人数、初期値はわずか100人なので拡張しておく)
  FDQuery1.Close;
  FDQuery1.SQL.Text := 'SELECT * FROM Following Where MaxUserLookup = 20000';
  FDQuery1.Open; // SQL実行
  SetColWidths;  // DBGridの列幅設定
end;

{**************************************************************************
 目的：③自分がいいねした投稿を表示
**************************************************************************}
procedure TForm2.Button3Click(Sender: TObject);
begin
  // 自分が「いいね」を送ったツイートを一覧表示
  FDQuery1.Close;
  FDQuery1.SQL.Text := 'SELECT * FROM Favorites';
  FDQuery1.Open; // SQL実行
  SetColWidths;  // DBGridの列幅設定
end;
```

「④フォロワーの一覧を表示」ボタンの処理は、
②の「Following」部分をビュー「Followers」に変更する
だけで、他の部分は同じとなるため省略

図20　Twitterのデータ参照②（参照結果）



※ 照会専用アカウントでログイン時は、③は参照不可。①②④についても対象ユーザーの指定が別途必要。

図21　Twitterのデータ参照③（参照結果）

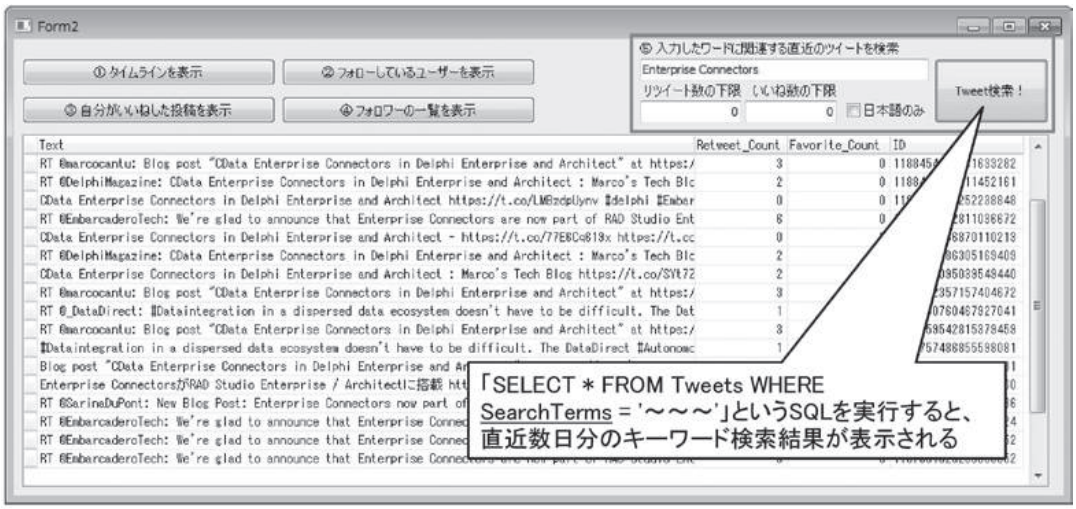
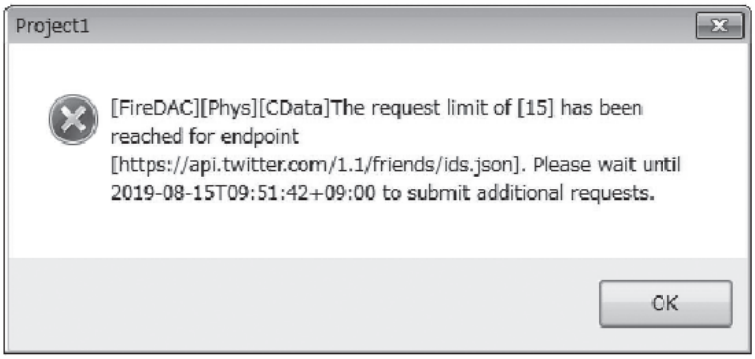


図22　Twitterのデータ参照④（通信回数上限について）



通信回数が規定数を超過すると、エラーでしばらく（最大15分）接続できなくなる

ソース5

Twitter APIを利用したツイートのキーワード検索

```
[*************************************************************************
 目的：⑤入力したワードに関連する直近のツイートを検索
*************************************************************************]
procedure TForm2.Button5Click(Sender: TObject);
begin
  if (Trim(LabeledEdit1.Text) = '') then
    raise Exception.Create('検索ワードは必ず指定して下さい。');

  // 対象データを抽出
  FDQuery1.Close;
  FDQuery1.SQL.Clear;
  FDQuery1.SQL.Add(' SELECT Text, Retweet_Count, Favorite_Count, ID');
  FDQuery1.SQL.Add(' FROM Tweets ');

  // 検索ワード（※SearchTermsをWHERE句に入れることで【ソース4】①とは検索条件が変わる）
  FDQuery1.SQL.Add(' WHERE SearchTerms = ''' + Trim(LabeledEdit1.Text) + ''' ');

  if (StrToIntDef(LabeledEdit2.Text, 0) > 0) then // リツイート数の下限指定時
  begin
    FDQuery1.SQL.Add(' AND Retweet_Count >= ' + LabeledEdit2.Text);
  end;
  if (StrToIntDef(LabeledEdit3.Text, 0) > 0) then // いいね数の下限指定時
  begin
    FDQuery1.SQL.Add(' AND Favorite_Count >= ' + LabeledEdit3.Text);
  end;
  if (CheckBox1.Checked) then  // 日本語のツイートに限定する時
  begin
    FDQuery1.SQL.Add(' AND Lang = ''ja'' ');
  end;

  FDQuery1.FetchOptions.RowsetSize := 100; // １回の通信で取得する件数
  FDQuery1.Open; // SQL実行
  SetColWidths;  // DBGridの列幅設定
end;
```

ツイートのキーワード検索を行うためのSQLを作成

FireDACの初期設定では50件ずつデータを取得するため、1回の通信（50件のみ取得）で必要十分なデータが得られない場合は、増やす
（※増やし過ぎるとレスポンスが遅くなったり、通信回数制限の原因になるため、注意）

図23　Twitterのデータ参照⑤（テーブル・ビューの一覧）

＜代表的なテーブルの一覧＞

テーブル名	概要
Favorites	・自分が「いいね」を送ったツイートを一覧表示 ・「いいね」の追加および解除
Following	・自分がフォローしているユーザーの一覧表示およびフォロー解除
Tweets	・自分および自分がフォローしているユーザーのツイートを一覧表示 ・WHERE句にSearchTermsを指定することで、特定の検索ワードで検索 ・ツイートの作成と削除

＜代表的なビューの一覧＞

ビュー名	概要
AccountSettings	自分のアカウントに関する設定を取得
AdInsights	自分が発信した広告に対する視聴者などの情報を取得
Followers	自分または指定したユーザーをフォローしているユーザーのリストを取得
Mentions	自分が受けた最新のメンション（自分宛ての返信も含む）を取得
Retweets	自分のリツイートのリストを取得
Trends	日時や国や地域（WoeID）を指定してトレンドトピックのリスト（50件）を取得
TweetStream	Twitterを流れるパブリックデータ（全ての公開ツイート）を取得
Users	キーワードや名前を基にユーザーの一覧を取得

※ 自分 = ログインによって認証されたユーザーを指す。
※ 照会専用アカウントでログイン時はユーザーを指定すれば参照できるものと、参照不可のものがある。更新は不可。

図24　Googleスプレッドシート①（画面設計）

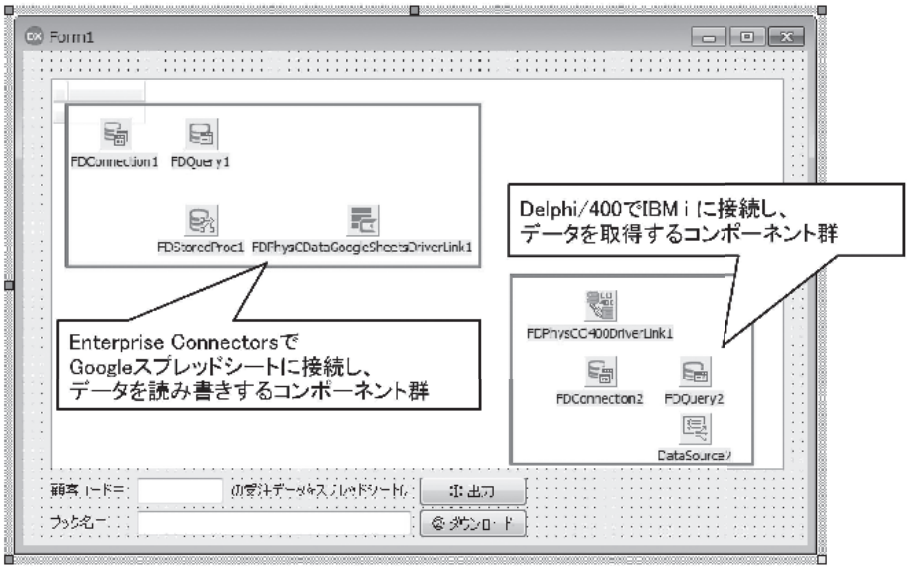


図25　Googleスプレッドシート②（接続パラメータ指定）

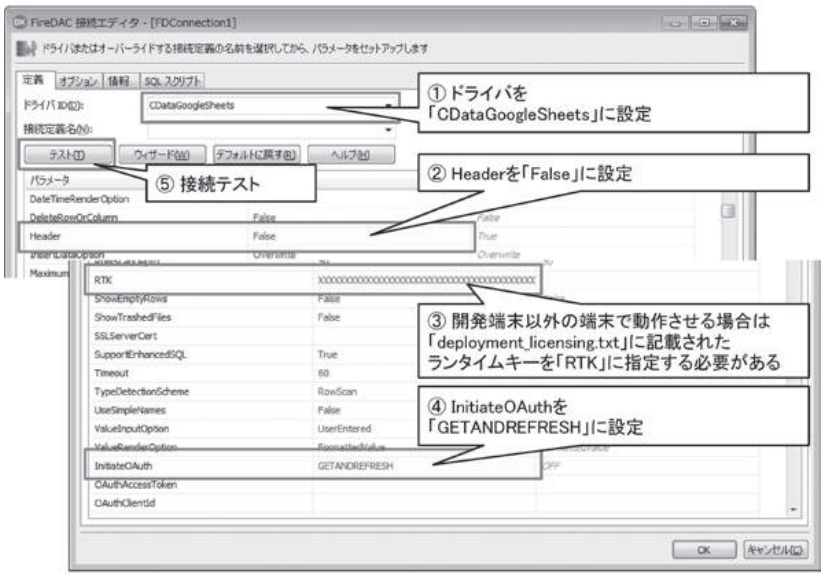


図26　Googleスプレッドシート③（初回接続時の認証）

現在ログイン中のGoogleアカウントで
APIを使用する権限を求められるため、許可する

ソース6

Google Sheets APIを使用したスプレッドシートへのデータ登録

```
{*******************************************************************************
 目的: ①出力ボタン 押下時処理
*******************************************************************************}
procedure TForm1.Button1Click(Sender: TObject);
var
  sXLSX: String;   // スプレッドシート名
  sSHTX: String;   // シート名
  iTR01, iTR04, iTR05, iTR06: Integer;  // 顧客コード、数量、単価、金額
  sTR02, sTR03: String;                 // 顧客名、商品名
begin

  // スプレッドシート名とシート名を定義（シート名は今回は初期値のままとする）
  sXLSX := 'レポート_' + FormatDateTime('YYYYMMDD_HHNNSS', Now);
  sSHTX := 'シート1';

  // 新しい空のスプレッドシートを作成
  FDStoredProc1.StoredProcName := 'CreateSpreadsheet';
  FDStoredProc1.Prepare;
  FDStoredProc1.ParamByName('Title').AsString := sXLSX; // スプレッドシート名
  FDStoredProc1.ExecProc;

  // 採番されたスプレッドシートのIDをグローバル変数に保持
  sXLSD := FDStoredProc1.ParamByName('Id').AsString;

  // Delphi/400側のQueryで対象データを抽出
  FDQuery2.Close;
  // (※この部分は【ソース2】と同一のため省略)
  FDQuery2.Open;

  // Enterprise Connectors接続
  if (not FDConnection1.Connected) then
  begin
    FDConnection1.Open;
  end;

  // Enterprise Connectors側のQueryでスプレッドシートに更新処理
  FDQuery1.Close;
  FDQuery1.SQL.Clear;
  FDQuery1.SQL.Add(' INSERT INTO ' + sXLSX + '_' + sSHTX + ' ( '); // テーブル名は「スプレッドシート名_シート名」形式
  FDQuery1.SQL.Add(' A, B, C, D, E, F ) VALUES ( ');              // 列IDの「A」～「F」がフィールドIDになる
  FDQuery1.SQL.Add(' :TR01, :TR02, :TR03, :TR04, :TR05, :TR06 ) ');

  // スプレッドシートの1行目にタイトル情報を登録
  FDQuery1.ParamByName('TR01').AsString := '顧客コード';
  FDQuery1.ParamByName('TR02').AsString := '顧客名';
  FDQuery1.ParamByName('TR03').AsString := '商品名';
  FDQuery1.ParamByName('TR04').AsString := '数量';
  FDQuery1.ParamByName('TR05').AsString := '単価';
  FDQuery1.ParamByName('TR06').AsString := '金額';
  FDQuery1.ExecSQL;   // INSERTのSQL実行

  // スプレッドシートの2行目以降に各レコード情報を登録
  while not FDQuery2.Eof do
  begin
    iTR01 := FDQuery2.FieldByName('ORCSCD').AsInteger;  // 顧客コード
    // (※この部分は【ソース2】と同一のため省略)
    iTR06 := iTR04 * iTR05;                             // 金額

    FDQuery1.ParamByName('TR01').AsString := IntToStr(iTR01); // 顧客コード
    FDQuery1.ParamByName('TR02').AsString := sTR02;           // 顧客名
    FDQuery1.ParamByName('TR03').AsString := sTR03;           // 商品名
    FDQuery1.ParamByName('TR04').AsString := IntToStr(iTR04); // 数量
    FDQuery1.ParamByName('TR05').AsString := IntToStr(iTR05); // 単価
    FDQuery1.ParamByName('TR06').AsString := FormatFloat('#,0', iTR06); // 金額
    FDQuery1.ExecSQL; // INSERTのSQL実行

    FDQuery2.Next;    // Delphi/400側のQueryを次行へ
  end;
end;
```

ストアドプロシージャを使用して
新しいスプレッドシートを作成

【ソース7】で使用するため、
ストアドプロシージャの戻り値として返る
スプレッドシートのIDを変数に保持する

スプレッドシートにデータを
登録するためのSQL文作成

1レコード目にタイトル情報の文字列を
セットすることで、結果のスプレッドシートも
1行目がタイトル行になる
（※接続時パラメータ「Header」＝Falseを指定）

1行目で全フィールドを文字型として定義した
ため、2行目以降も全て文字としてセット

図27　Googleスプレッドシート④（PGM実行時画面）

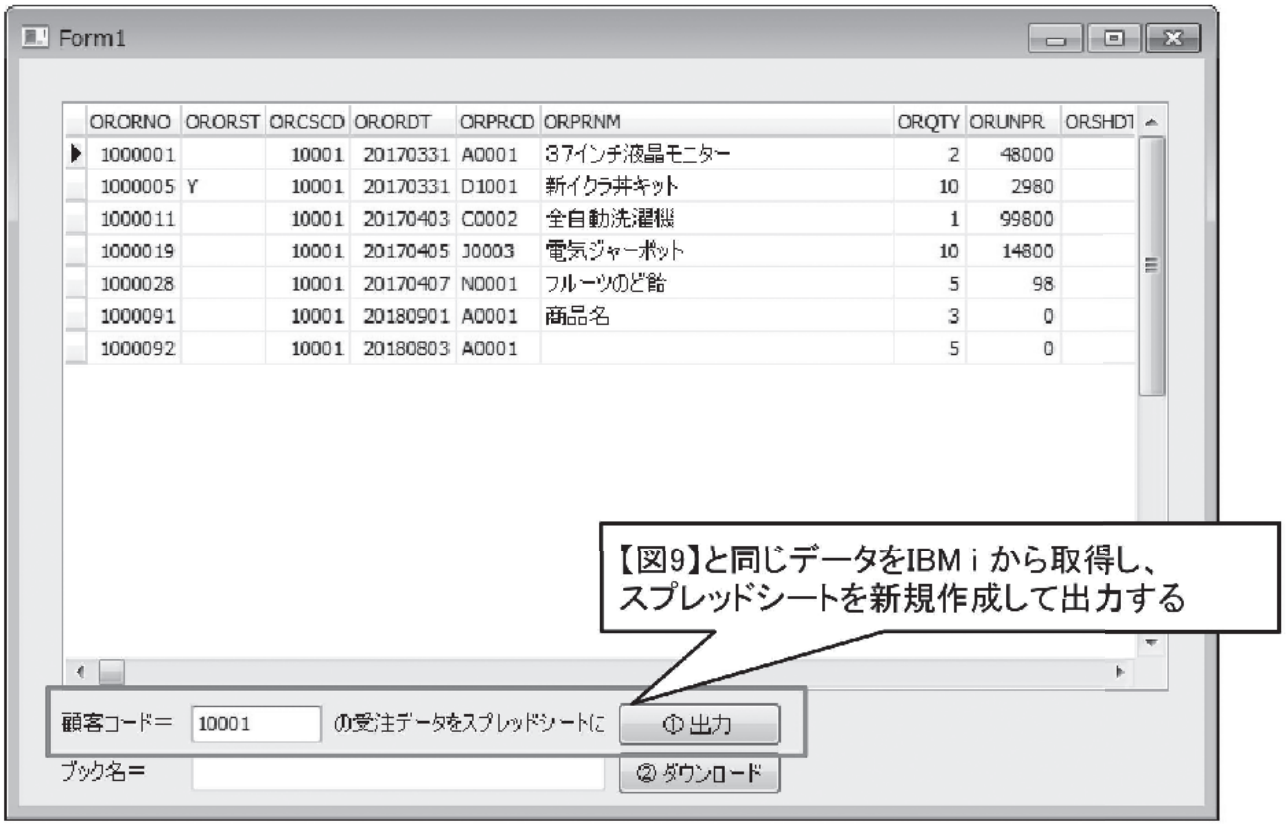
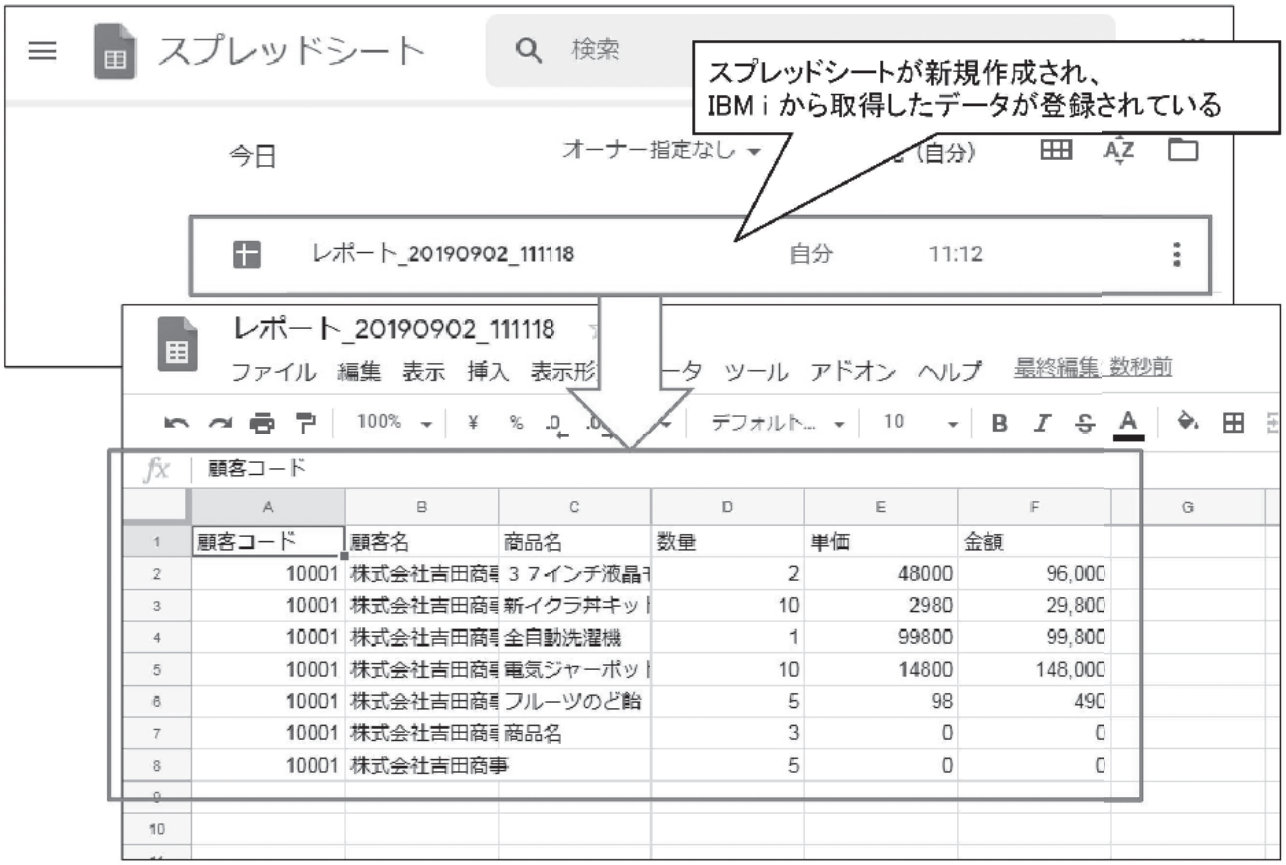


図28　Googleスプレッドシート⑤（出力結果）



ソース7

Google Sheets APIを使用したスプレッドシートのダウンロード

```
{*************************************************************************
 目的: ②ダウンロードボタン 押下時処理
*************************************************************************}
procedure TForm1.Button3Click(Sender: TObject);
var
  sID: String;
  sEXT: String;
begin

  // ダウンロード用に、ストアドプロシージャのパラメータを指定
  FDStoredProc1.StoredProcName := 'DownloadDocument';
  FDStoredProc1.Prepare;
  FDStoredProc1.ParamByName('Id').AsString := sXLSD;        // スプレッドシートのID
  FDStoredProc1.ParamByName('LocalFile').AsString := Edit3.Text; // ダウンロード先

  // ダウンロード先の拡張子によってファイル形式(MIME)を指定
  // (※xlsxの場合はデフォルト値なので指定不要)
  sEXT := UpperCase(ExtractFileExt(Edit3.Text));
  if (sEXT = '.PDF') then
  begin // 例: PDFの場合のMIME値
    FDStoredProc1.ParamByName('FileFormat').AsString := 'application/pdf';
  end;
  FDStoredProc1.ExecProc;     // ダウンロード実行
end;
```

【ソース6】で、スプレッドシートを
新規作成時に保持していた値

以下の4種類が指定可能

XLSX（初期値）：「application/vnd.openxmlformats-officedocument.spreadsheetml.sheet」

PDF：「application/pdf」

CSV（カンマ区切り）：「text/csv」

TSV（タブ区切り）：「text/tab-separated-values」

図29　Googleスプレッドシート⑥(ダウンロード)

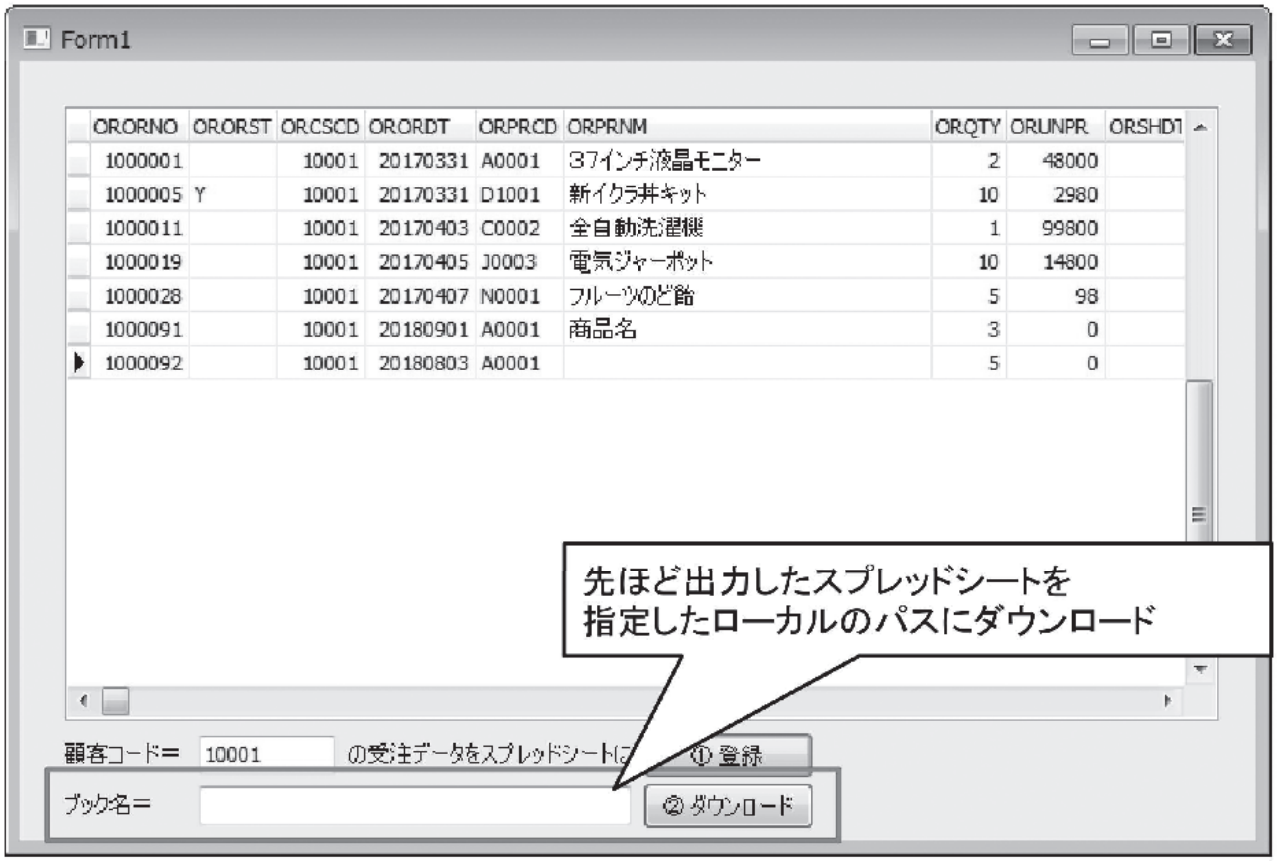


図30　Googleスプレッドシート⑦（ダウンロード結果）

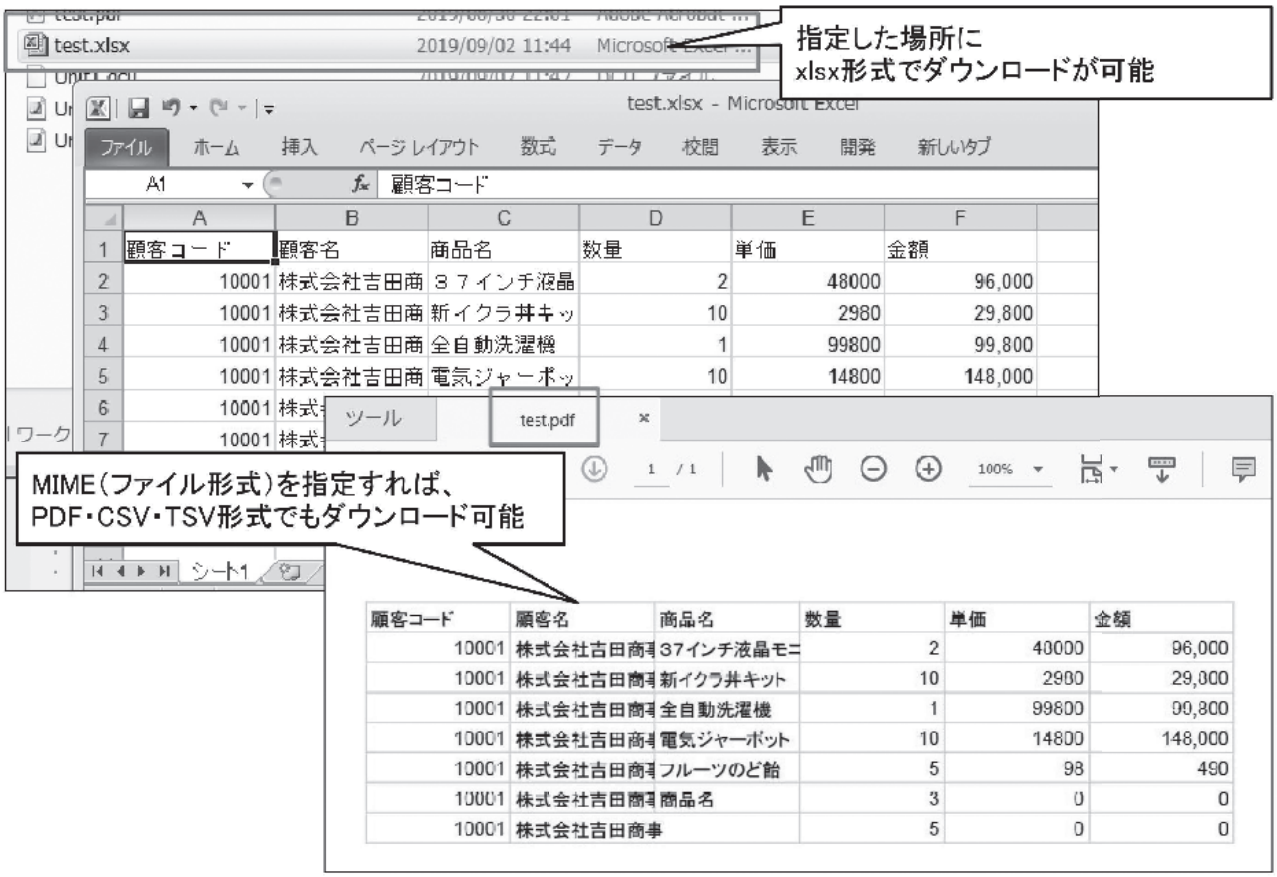


図31　Google Drive①（画面設計）

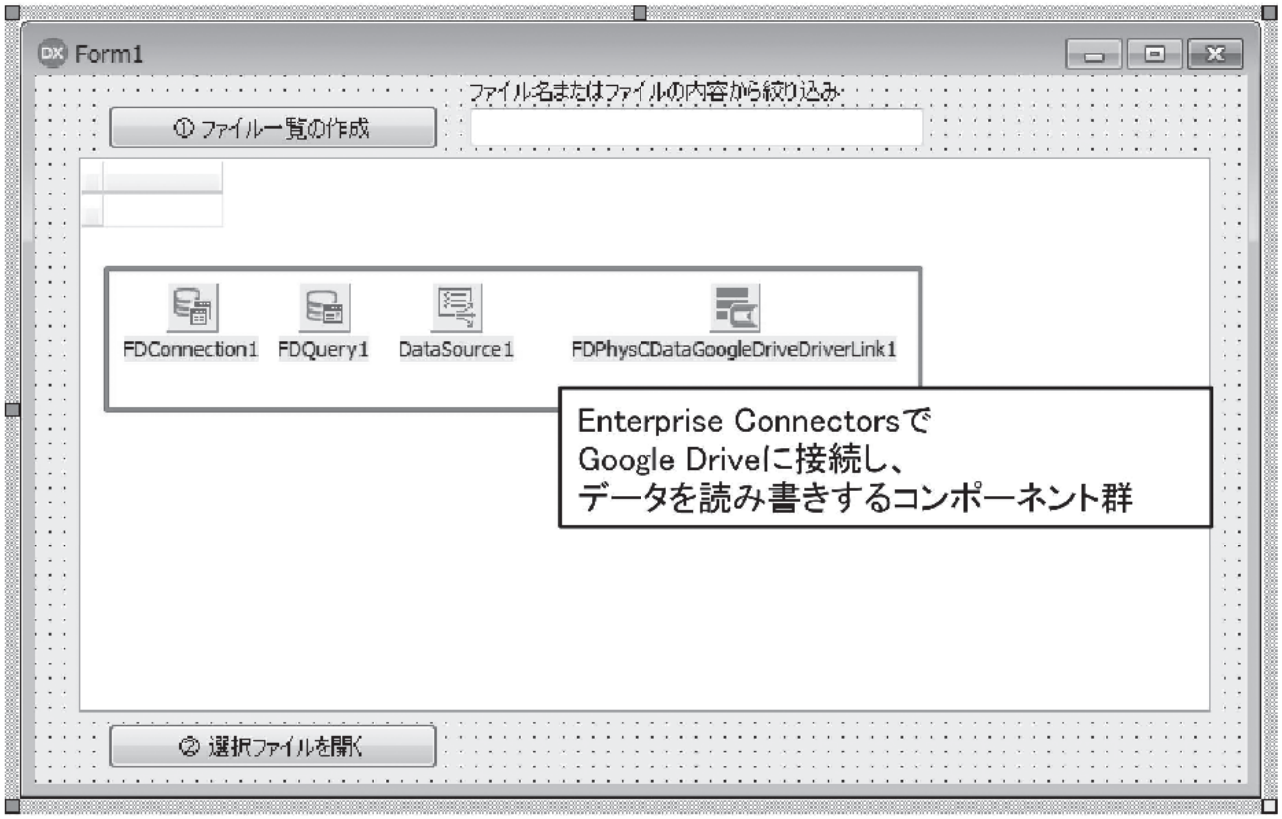


図32　Google Drive②（接続パラメータ設定）

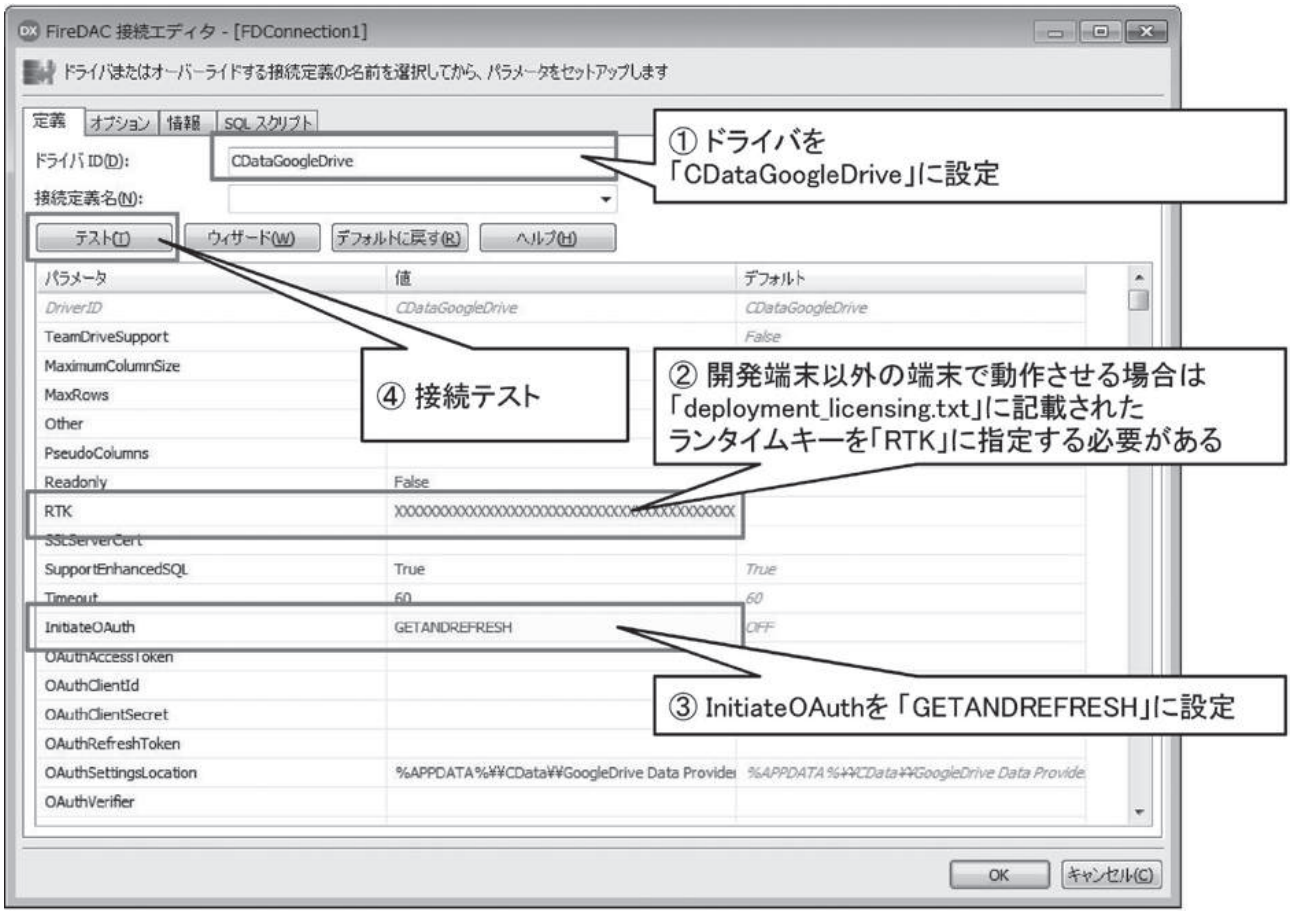
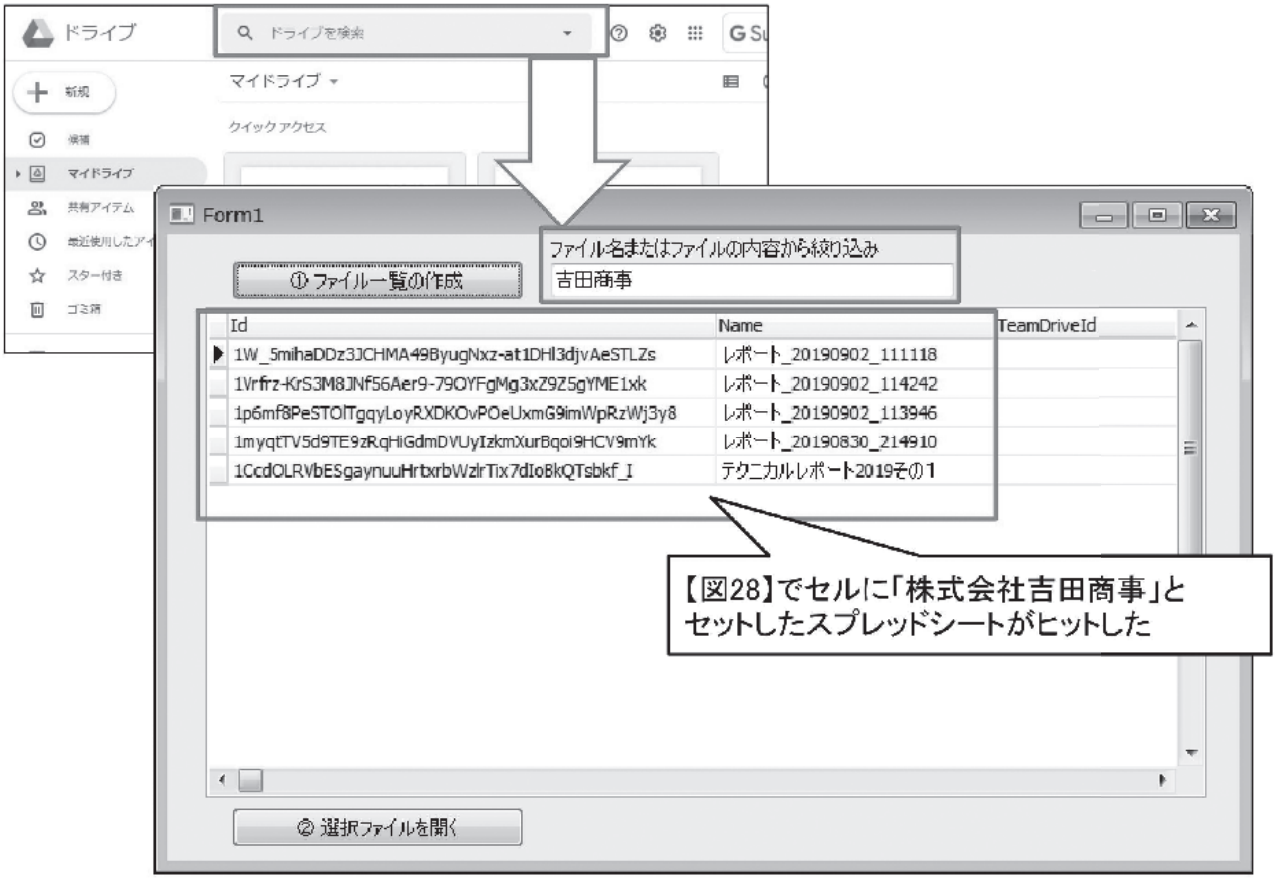


図33　Google Drive③（フルテキスト検索）



ソース8

Google Driveのファイル検索

```
{*************************************************************************
 目的: ①ファイル一覧の作成 押下時処理
*************************************************************************}
procedure TForm1.Button1Click(Sender: TObject);
begin
  // ファイルの一覧
  FDQuery1.Close;
  FDQuery1.SQL.Clear;
  FDQuery1.SQL.Add(' SELECT * FROM Files ');
  if (LabeledEdit1.Text <> '') then
  begin
    FDQuery1.SQL.Add(' WHERE Query = '' (fullText contains ¥''' + LabeledEdit1.Text + '¥'')'' ');
  end;
  FDQuery1.Open;
end;
```

ファイル内の文字列を含めて
フルテキスト検索

ソース9

Google Driveで検索したファイルのオープン

```
{*************************************************************************
 目的: ②選択ファイルを開く 押下時処理
*************************************************************************}
procedure TForm1.Button2Click(Sender: TObject);
var
  sURL: String;
begin
  // 共有可能なURLを実行する (※uses節に Winapi.ShellAPI が必要)
  sURL := 'https://drive.google.com/open?id=' + FDQuery1.FieldByName('ID').AsString;
  ShellExecute(Application.Handle, 'OPEN', PChar(sURL), PChar(''), '', SW_SHOW);
end;
```

図34　Google Drive④(ファイルのオープン)