Delphi/400

FireDACを活用した Delphi/400ロジック最新化テクニック

株式会社ミガロ.
RAD事業部技術支援課
佐田　雄一

略　歴

生年月日:1985年12月6日
最終学歴:2009年　甲南大学 経営学部卒業
ミガロ入社年月:2009年04月　株式会社ミガロ. 入社
社内経歴:
2009年04月　システム事業部配属
2019年04月　RAD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の経験を経て、
現在はDelphi/400のサポート業務を担当。

1.はじめに

Delphi/400 Version 5が2000年に日本で販売を開始してから早くも20年が経過したが、実は当時から存在するロジックの大半が現在の最新バージョン(Delphi/400 10.2 Tokyo)でも利用することができる。しかし当時はWindows 95や98がまだ現役だった頃で、その後のWindows OS側の技術革新で、初期のコンポーネントやロジックの中には動作にあたって制約が生じているものも存在する。
本稿ではSQL等のIBM i側の資産や、最新のFireDAC接続を用いて、最新バージョンでさらに使いやすくなったコンポーネントに関するトピックと、先述の動作に制約が発生しているソースをリフレッシュするためのトピックの2つを紹介する。
なお、本稿ではDelphi/400 10.2 TokyoおよびV7R1以上のIBM i環境を使用している。

2.TClientDatasetに代わるデータセットTFDMemTableの活用術

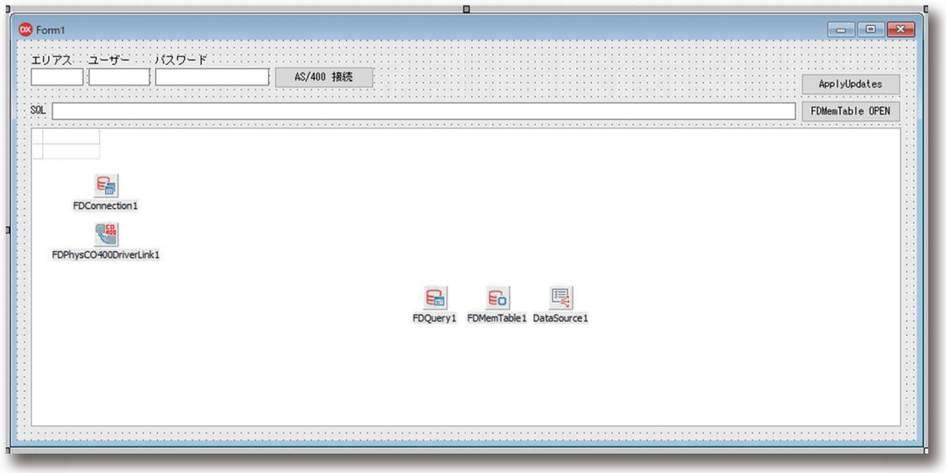
① TFDMemTableとは

IBMiのデータベースファイルからデータを参照して表形式で表示する際に、もっとも一般的なコンポーネントの配置順は「データセットコンポーネント(TFDQuery・TSQLQuery等)⇒TDataSetProvider⇒TClientDataSet⇒TDataSource⇒TDBGrid」となるだろう。
ここで使用するTClientDataSetコンポーネントは取得したデータを内部メモリでキャッシュするための汎用インメモリデータセットで、単独使用やdbExpress接続におけるキャッシュ機能の実現の為に用いられる。最新のDelphi/400においてもTClientDataSetやTDataSetProviderといったコンポーネントをこれまで通り使用することは可能であるが、古いBDE接続やdbExpress接続のアーキテクチャにあわせて作られており、Version 5の頃から基本的な仕様はそのままとなっている。また、Delphi開発元のエンバカデロ・テクノロジーズ社は今後の更新が行われる可能性は限りなく低いとしており、FireDAC接続でのデータキャッシュにおいては後継となるTFDMemTableコンポーネントの使用を推奨している。
dbExpress接続では単方向での通信だったものがFireDAC接続においては双方向で通信可能なため、FireDAC接続ではTClientDataSetを経由させずに「TFDQuery⇒TDataSource⇒TDBGrid」といったコンポーネントの配置が可能である。一方、旧バージョンにおいてBDE接続やdbExpress接続で作成されたアプリケーションをFireDAC接続に移行する場合には、元々TClientDataSetで使用されていた内部計算項目の使用や、古いParadoxに代わるローカルデータベースとしての活用がTFDMemTableコンポーネントによって可能になっている。
次項より、FireDAC接続のために最適化された新しいデータセット、TFDMemTableの使用方法について紹介する。

② TFDMemTableを使ったデータ照会方法

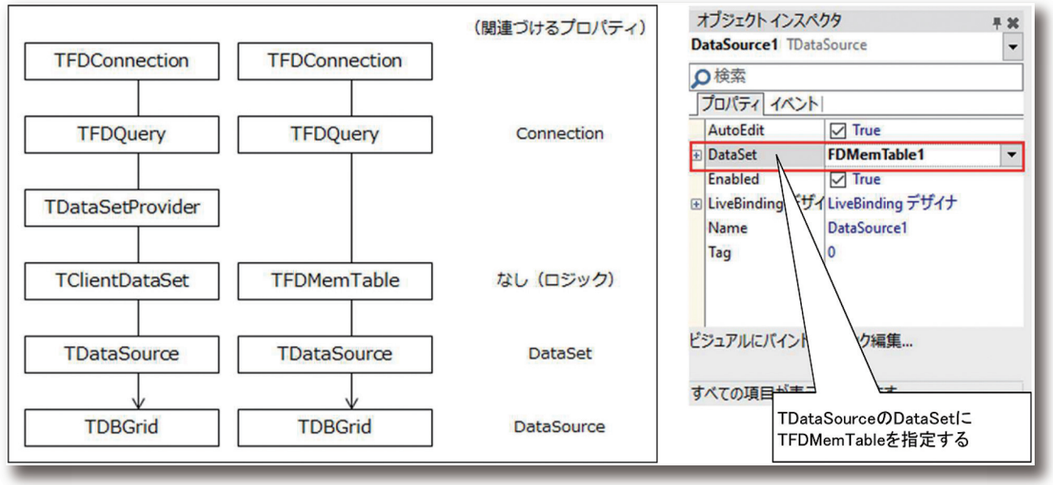
本項ではTClientDataSet・TDataSetProviderの代替として、TFDMemTableを使ってデータを取得する手順を紹介する。
Delphiを起動してプロジェクトを新規作成したら、まずフォーム上にFireDAC接続を行うために必要なTFDConnection・TFDPhysCO400DriverLinkと、TFDQuery・TFDMemTable・TDataSource・TDBGridの各データベースコンポーネントおよび、画面操作に使用するEditやButton等を配置する【図1】。TFDConnectionを使ったFireDAC接続の設定自体は過去のテクニカルレポートでも取り上げているため、本稿では割愛する。(2018年のテクニカルレポートに掲載の拙著「FireDAC実践プログラミングテクニック」を参照されたい。)

図 1　サンプルの画面設計



TDBGridのDataSourceプロパティにTDataSourceを設定するのは従来と同様だが、TDataSourceのDataSetプロパティにはTClientDataSetの代わりにTFDMemTableを設定する【図2】。また、オブジェクトインスペクタ上のプロパティではTFDMemTableとTFDQueryを接続できる箇所がないため、代替としてロジックを使って、TFDQueryで取得したデータをTFDMemTableへ連携する。連携する方法はいくつか存在する【ソース1】【ソース2】。

図 2　データセット指定



ソース 1

CloneCursorを使ったTFDMemTableでのデータ参照

```
FDQuery1.Close;
FDMemTable1.Close;
FDQuery1.SQL.Text := 'SELECT * FROM TESTTBL21 ';
FDQuery1.Open;
FDMemTable1.CloneCursor(FDQuery1, False, False);
```

ソース 2

Dataの代入を使ったTFDMemTableでのデータ参照

```
FDQuery1.Close;
FDMemTable1.Close;
FDQuery1.SQL.Text := 'SELECT * FROM TESTTBL21 ';
FDQuery1.Open;
FDMemTable1.Data  := FDQuery1.Data;
```


【ソース1】はCloneCursorメソッドを使用する方法で、【ソース2】はDataプロパティを代入する方法である。本項ではこの2パターンを例示したが、それぞれの特徴や利点を紹介する。

いずれの例においても事前にTFDQueryをオープンしておく点は同じであるが、【ソース1】でCloneCursorメソッドを行う場合は、パラメータにデータ取得元となるTFDQueryを指定することで、そのTFDQueryにあるデータのポインタのみを複製し、内部的なデータは同じものを共有するイメージとなる。

対して【ソース2】でDataを代入する方法ではその時点で取得済みのデータをそのままTFDMemTableにコピーするイメージとなる。そのため、TFDQuery側でFetchOptionsプロパティの設定によって最初の50件までしかレコードを取得していない場合、TFDMemTableにコピーされるレコードもその件数のみとなる。たとえば元のデータの件数が非常に多く時間がかかる場合にTFDQuery側でレコードの取得件数を制限する際には、CloneCursorメソッドを使用する場合よりも処理の高速化が期待できる。

照会したデータはTClientDataSetと同じようにIndexDefs・IndexNameプロパティによるローカルでの並び替えや、後述するJSON形式でのデータ出力などTFDQuery（またはTFDTable、以下同じ）では実現できない機能が存在する。なお、TClientDataSetにも同様にコピーが可能な「Data」プロパティが存在しているが、TFDMemTableのDataとは内部の型が異なるため互換性はなく、直接代入するようなロジックを記述するとコンパイルエラーになる。TClientDataSetからTFDMemTableへのデータ同期が必要な場合はCopyDataSetメソッドを使うと、異なる接続方式のデータセットからTFDMemTableにデータを転送することができる【ソース3】。パラメータに同じFireDAC接続のTFDQueryを指定することも可能で、この場合はDataを代入する際と同じ挙動になる。

ソース 3

他種のデータセットからTFDMemTableへのデータコピー

```
// [例] TClientDataSetでローカルデータを読み込み
ClientDataSet1.LoadFromFile('C:¥～～～¥DATA.xml');
// CopyDataSetでデータコピーするとTFDMemTableでも扱える
FDMemTable1.CopyDataSet(ClientDataSet1, [coStructure, coRestart, coAppend]);
```

③ TFDMemTableを使ったデータ更新方法

BDE接続・dbExpress接続においてはTClientDataSetのApplyUpdatesメソッドを使うことで、接続されたQueryコンポーネント経由でインメモリデータセットの変更内容をIBM iに更新できたが、TFDMemTableにおいてもロジックが少々異なるものの、ApplyUpdatesメソッドで同様の更新が可能である。
FireDAC接続においてはTFDQueryでも紐付くTDBGridを直接編集することや、Edit～Postのメソッドを行うことで内部的に更新SQLを発行し、接続先のDBに更新を行うことができる。(更新自体を可能にするためのUpdateOptionsプロパティの設定については、前章でも紹介した2018年のテクニカルレポートに掲載の拙著「FireDAC　実践プログラミングテクニック」を参照されたい。)
このとき、更新に使用するTFDQueryのCachedUpdatesプロパティがFalseならPostやDeleteの時点で即時更新が行われ、Trueなら更新情報を内部でキャッシュし、ApplyUpdatesメソッドを行った時点で一括更新が行われる。
BDE接続・dbExpress接続との違いとしては更新先ファイルへのジャーナル開始やトランザクション制御が不要な点が挙げられるが、複数件のデータを更新している途中でエラーになった場合のリスクを考慮すれば、従来通りジャーナルやトランザクション制御を使用しておいた方が安全である。

TFDMemTableを使用してApplyUpdatesメソッドを実行する際の前提条件としては、以下の3点があげられる。
・前項で紹介した2種類の手順のうち、**【ソース1】**のCloneCursorメソッドを使ってデータを取得している。
・ApplyUpdatesメソッドを実際に行うのは紐づいているTFDQueryとなるため、ApplyUpdates実行時までTFDQueryが開いたままになっている。(TFDMemTable側でApplyUpdatesメソッドを実行しても更新は反映されない)
・キャッシュ更新を有効にするため、先述のTFDQueryのCachedUpdatesプロパティがTrueになっている。
これらの条件が揃った状態でTFDMemTableのデータを変更し、**【ソース4】**のようなロジックでApplyUpdatesメソッドを実行すると、Open時からキャッシュされた変更がIBM　iに反映される。

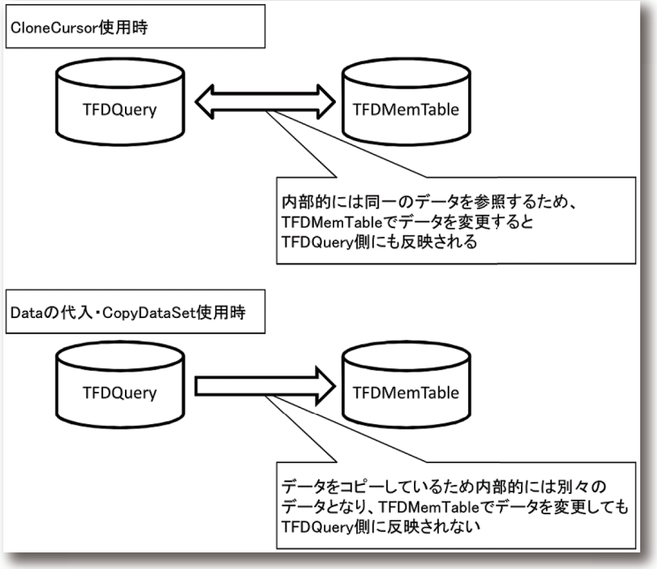
ソース 4

TFDMemTableのデータ更新

```
// 紐づくTFDQuery側でApplyUpdatesする
FDQuery1.ApplyUpdates(-1);
```

前項で紹介したデータ取得手順のうちCloneCursorメソッドを使う手順の場合はTFDQueryと同じデータを内部で共有しているためApplyUpdatesメソッドによって更新が反映されるが、Dataをコピーする手順やCopyDataSetメソッドを使用する手順の場合は別々のデータとなる**【図3】**ため、ApplyUpdatesメソッドを行ってもIBM iに更新は反映されない。

図 3　TFDMemTableへのデータ転送方法の違い



④TFDMemTableを使ったローカルDB活用

TFDMemTableはTClientDataSetと同じように、LoadFromFileメソッドやSaveToFileメソッドで外部データを読み込んだり書き出したりすることができる。例えば処理速度向上のためローカルPC上に値が長期間変わることのないマスタファイルのデータをコピーしておきたい場合に有効である。

保存先と形式をパラメータに指定してSaveToFileメソッドを実行すれば、指定した場所にファイルが出力される**【ソース5】**。

ソース 5

TFDMemTableのローカルデータ保存と読み込み

```
// ローカルデータにJSON形式で保存する例
FDMemTable2.SaveToFile('FDMem2.json', sfJSON);

// ローカルデータから読み込む例
FDMemTable2. LoadFromFile('FDMem2.json');
```

```
sfAuto   : 自動（拡張子で判断）
sfBinary : 専用バイナリ形式
sfXML    : XML形式
sfJSON   : JSON形式
```

TClientDataSetとの違いとしては、保存形式によって出力に使用するコンポーネントの配置が必要になること【図4】と、出力形式の違いである。前項で紹介したDataプロパティと同様にＴＦＤＭｅｍＴａｂｌｅで書き出すデータとTClientDataSetで書き出すデータの内容では大きく異なるため互換性は無いが、TFDMemTableではXML形式に加えてJSON形式でも出力することができる【図5-1～3】。TClientDataSetでXMLデータを出力した場合と同様に、これらを他のプログラムやデータベースで流用することが可能である。

図 4　TFDMemTable保存に必要なコンポーネント

図 5-1　TFDMemTableの出力前データ

SHSHCD	XX01	SHSHBT	SHSHNM	SHMKNM	SHSIZE	SHTANI	SHIRSU
101148	101148	クラブ	LEGACY BLACKアイアン（6本）ダイナミックゴールドシャフト	キャロウェイ	R	セット	
102079	102079	クラブ	2014 オノフ ドライバー（赤ドライバー） MP-514Dシャフト	⋕⋕ミガロ.		本	
102082	102082	クラブ	2014 オノフ TYPE-S （黒）ドライバー MP-614Dシャフト	⋕⋕ミガロ.	AA	本	
ABCDEFGHIJ	ABCDEF	クラブ	BIG BERTHA BETA ドライバー TYPE 2★	キャロウェイ		本	
ABCDEFGHI6	ABCDEF	クラブ【新】	2015 ONOFF フェアウェイ;WINGS KURO（ユーティリティー）	あ???	⋕⋕	本	
105567	105567	クラブ	IG BERTHA BETA アイアン（5本セット）	キャロウェイ		セット	
105974	105974	クラブ	2015 BIG BERTHA アルファ815 ダブルダイヤモンド ドライバ	キャロウェイ	141	本	
106516	106516	クラブ	JPX850 フォージドアイアン（6本セット）	ミズノ	141	セット	
106632	106632	クラブ	JPX850 ドライバー オロチパワーマキシマイザー	ミズノ		本	
106655	106655	クラブ	2015 R15 460 ドライバー ATTAS6☆ 6カーボンシャフト	テーラーメイド		本	
107301	107301	クラブ	2015 XR ドライバーTOUR AD MJ-6 シャフト	キャロウェイ	252	本	
107315	107315	クラブ	2015 XR ユーティリティー XR NS950スチール シャフト	キャロウェイ	252	本	
ABCDEFGHI4	ABCDEF	クラブ	2015 VOLTIO KABUTO 103 パター	カタナゴルフ		本	
107926	107926	クラブ	2015 FUTURA X5 デュアルバランス パター	スコッティキャメロン		本	
108036	108036	クラブ	2015 AMERICAN CHANPIONSHIP パター RJB6823	ベティナルディー	363	本	
108057	108057	クラブ	LUCKY 777 #7 SB（シングルベント） パター	オデッセイ		本	
108091	108091	クラブ	2015 FEMINA（フェミナ） パター TX-415Pシャフト	ヤマハ		本	
108125	108125	クラブ	2015 FEMINA（フェミナ） ユーティリティ TX-415Uシャフト	ヤマハ		本	

図 5-2　TFDMemTableのXML出力サンプル

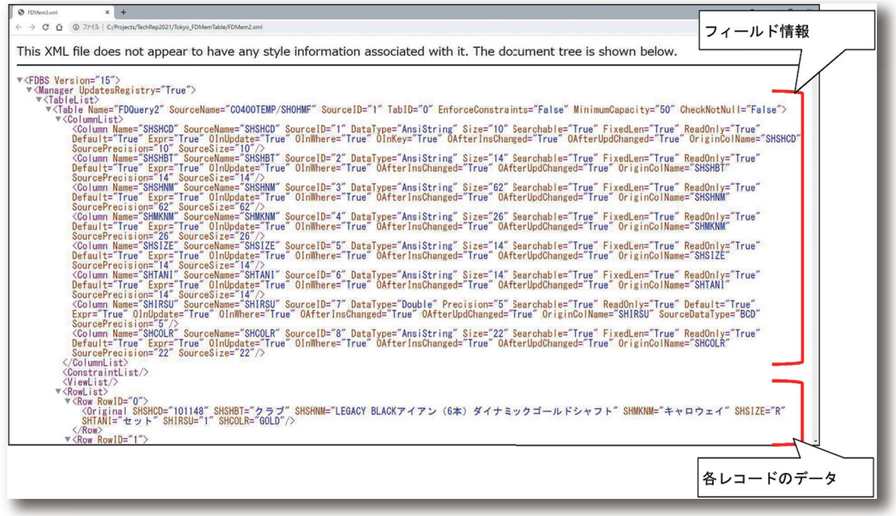
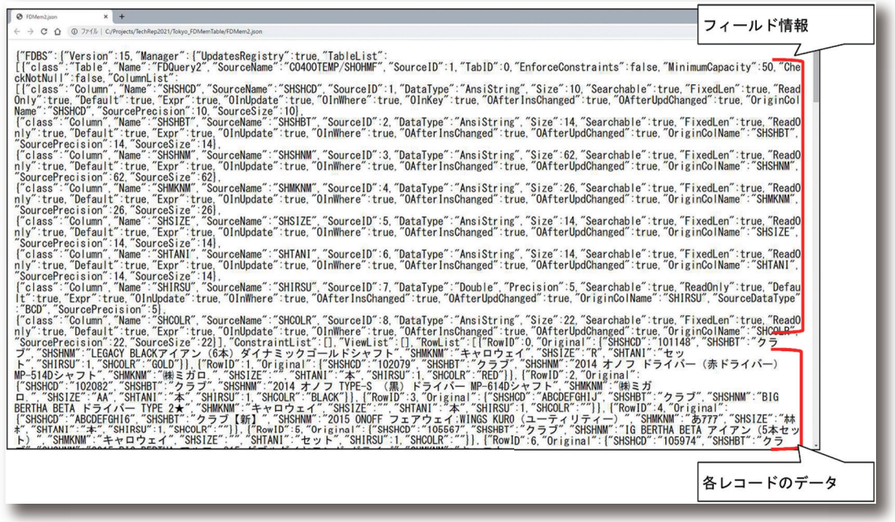


図 5-3　TFDMemTableのJSON出力サンプル



これ以外にTFDMemTableで実現可能な機能については、docwiki（エンバカデロ社の公式オンラインヘルプ）にも多数掲載されているので、そちらも参照されたい。

従来のTClientDataSetでもプログラムを作成することは可能だが、今後の新規開発においてはTFDMemTableコンポーネントの使用も検討いただきたい。

3. TSpool400に代わるSQLメインのスプール活用術

Delphi/400では、IBMiのスプールファイルを参照するためのTListSpool400・TSpool400コンポーネントが存在する。しかしこれらはBDEベースの古い技術で作成されており、最新のDelphi/400 10.2 Tokyoでも動作にはBDEが必要となっている。(Windows10ではBDEが互換での動作となるため、アプリケーションを実行するには管理者権限が必要で、かつ64bitアプリケーションには対応していない)

本章ではこのTListSpool400・TSpool400コンポーネントに代わり、FireDAC接続のSQLでスプール情報を参照する方法について紹介する。Delphi/400 10.2 Tokyoにおいては FireDAC接続も64bitアプリケーションには対応していないが、本章で記載の各SQLはdbExpress接続でも実行が可能である。64bitアプリケーションで本章のスプール活用を実施される場合は、dbExpress接続を利用して頂き、本文中の「TFDQuery」を「TSQLQuery」に読み替えていただきたい。

①スプールのリスト表示

従来はTListSpool400コンポーネントを使って指定したOUTQ内にあるスプールの一覧を表示し、その中にある個別のスプールの内容をTSpool400コンポーネントで表示していたが、この一連の操作をFireDAC接続のSQLで代替して行う手順を紹介する。

まず、TListSpool400コンポーネントでは「LibraryName」「OutQName」プロパティでOUTQのライブラリと名前を指定しているが、どのライブラリに目的のOUTQが存在するかわからない場合があっただろう。IBM i(V7R1以上)ではシステムライブラリ「QSYS2」内にOUTQの一覧情報を保持しており、『SELECT OUTQ, OUTQLIB, FILES, TEXT FROM QSYS2/OUTPUT_QUEUE_INFO』というSQLを実行することでその一覧を表示することができる。

(「OUTPUT_QUEUE_INFO」はビューの名前となっており、代わりに実ファイル名の「OUTQ_DTL」も指定可能である。またこのファイルにはDelphi/400が対応していない型のフィールドが存在するため、上記のフィールドのみを指定するのが望ましい。)

取得した4つのフィールドはそれぞれ

・OUTQ＝対象OUTQの名前
・OUTQLIB＝対象OUTQが格納されたライブラリ
・FILES＝現在そのOUTQに格納されているスプールの数
・TEXT＝テキスト

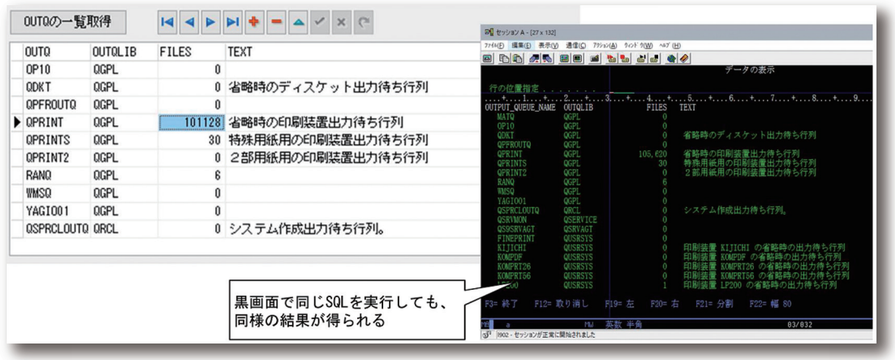
を示しているので、目的のOUTQの名前とライブラリを確認する。**【ソース6】【図6】**

ソース 6

IBM i 内のOUTQ一覧をSQLで取得する例(V7R1以上)

```
FDQuery1.Close;
FDQuery1.SQL.Text := 'SELECT OUTQ, OUTQLIB, FILES, TEXT FROM QSYS2/OUTPUT_QUEUE_INFO';
FDQuery1.FetchOptions.Mode := fmAll; // 一度に全件取得
FDQuery1.Open;
```

図 6　OUTQの一覧表示



次に指定したOUTQから個別スプールの一覧を取得する。こちらも先ほどと同様にIBM i(V7R1以上)でシステムライブラリ「QSYS2」内にスプールファイルの一覧情報を保持しており、

『SELECT SPOOLED_FILE_NAME, FILE_NUMBER, JOB_NAME,CREATE_TIMESTAMP FROM QSYS2/OUTPUT_QUEUE_ENTRIES WHERE OUTQ＝(OUTQ名) AND OUTQLIB＝(ライブラリ名) AND CREATE_TIMESTAMP＝(作成日時)』

というSQLを実行することで、その一覧を表示することができる。**【ソース7】【図7】**

ソース 7

OUTQ内のスプール一覧をSQLで取得する例(V7R1以上)

```
FDQuery2.Close;
FDQuery2.SQL.Text := ' SELECT SPOOLED_FILE_NAME, FILE_NUMBER, JOB_NAME, CREATE_TIMESTAMP ' +
                     ' FROM QSYS2/OUTPUT_QUEUE_ENTRIES ' +
                     ' WHERE OUTQ = :OUTQ AND OUTQLIB = :OUTQLIB ' +
                     ' AND CREATE_TIMESTAMP >= :CRTDATE ';
FDQuery2.ParamByName('OUTQ').AsString      := 'QPRINT';  // OUTQ名
FDQuery2.ParamByName('OUTQLIB').AsString   := 'QGPL';    // ライブラリー名
FDQuery2.ParamByName('CRTDATE').AsDateTime := StrToDateTime('2021/4/1'); // 作成日の最小値
FDQuery2.FetchOptions.Mode := fmAll; // 一度に全件取得
FDQuery2.Open;
```

日付時刻型で渡すため、文字列にして StrToDateTimeを行う

図 7　スプールの一覧表示

（「OUTPUT_QUEUE_ENTRIES」はビューの名前となっており、代わりに実ファイル名の「OUTQ_INFO」も指定可能である。またすべてのスプールの情報を取得しようとすると処理時間および処理結果の件数が膨大になるため、取得フィールドを絞るとともにWHERE句を使用することで処理時間の短縮を図っている。このほかにも、WHERE句に「JOB_NAME LIKE '%〜〜〜%'」を付加してスプールを発行したユーザー名で絞り込んだり、SQL文の末尾に「FETCH FIRST 〜〜 ROWS ONLY」を付加して取得件数を絞り込んだりするのも有効である。）

②個別スプールの取得

前項で取得したスプールの情報をパラメータとしてIBM iに対してCPYSPLFコマンドを発行することで、個別スプールの内容を物理ファイルに出力することができる。BDE接続を使用していた従来のTSpool400コンポーネントでも、内部的には同様のコマンドを発行してQTEMPに出力される一時ファイルを通して処理が行われていた。

CPYSPLFコマンドにはいくつかのパラメータが必要だが、それぞれ前項で取得したスプールファイル名、ジョブ名、タイムスタンプ等の各フィールド値を使用する。具体的には**【ソース8】**のように記述する。タイムスタンプは日付と時刻に分割する必要があるため、**【ソース8】**のロジックでうまくいかない場合はご利用のIBM i環境の設定値をご確認いただきたい。

また、出力先となる一時ファイルを事前に作成しておくのがポイントで、このファイルには対象スプールの桁数以上の長さを持つOタイプフィールド1つのみを定義しておく。作成方法はDDSの作成以外にも、**【図8】**のようなコマンド発行や、CREATE TABLEのSQL発行でも問題ない。この一時ファイルは本稿では便宜上通常のライブラリに作成しているが、QTEMPに作成しても問題はない。

ソース 8

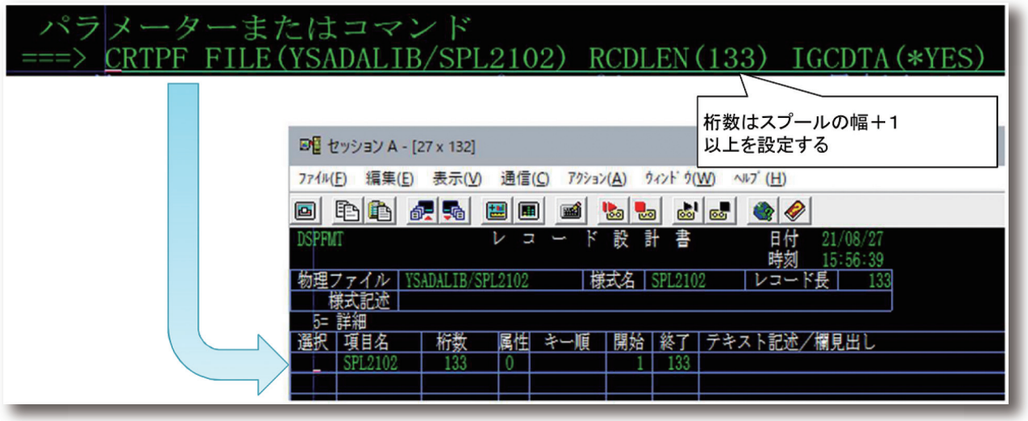
CPYSPLFコマンドを発行して一時ファイルにスプールを出力

```
var
  sFIL: String; // 出力先ファイル名
  sCMD: String; // コマンド文の保管
  sDAT: String; // タイムスタンプの保管
begin

  // 出力先ファイルの前回出力値をクリア（ファイルは既に存在する前提）
  sFIL := 'YSADALIB/SPL2102';
  AS400.RemoteCmd('CLRPFM FILE(' + sFIL + ')');

  // タイムスタンプの作成（書式例『'2020/02/03' '17:34:10'』）
  sDAT := Trim(FDQuery2.FieldByName('CREATE_TIMESTAMP').AsString);
  sDAT := QuotedStr(Copy(sDAT, 1, 10)) + ' ' +
          QuotedStr(Copy(sDAT, 12, 8));

  // コマンド文の作成
  sCMD := 'CPYSPLF FILE(' + Trim(FDQuery2.FieldByName('SPOOLED_FILE_NAME').AsString) + // スプールファイル名
          ') TOFILE('     + sFIL +                                                     // コピー先
          ') SPLNBR('     + Trim(FDQuery2.FieldByName('FILE_NUMBER').AsString) +       // スプールファイル番号
          ') JOB('        + Trim(FDQuery2.FieldByName('JOB_NAME').AsString) +          // ジョブ番号/ユーザー/ジョブ名
          ') CRTDATE('    + sDAT +                                                     // 作成日＋作成時刻
          ') TOMBR(*FIRST)' +  // 最初のメンバーに出力
          '  CTLCHAR(*FCFC) '; // 制御文字（改行・改ページ設定時に使用）

  // TAS400コンポーネントで作成したコマンドを発行し、一時ファイルに書き出し
  AS400.RemoteCmd(sCMD);
end;
```

【図8】で作成した一時ファイル

コマンドにはこの書式で
日付と時刻を分割して渡す必要がある

図 8　一時ファイルを事前に作成しておく



③ 取得したスプールの操作

前項のCPYSPLFコマンドで取得した個別スプールを参照するには、通常のファイルを参照する際と同様にTFDQueryで一時ファイルをSELECTする。すると、**【図9】**のように各レコードの1桁目に内部識別用の文字（ANSI用紙制御コード）を保持していることを確認できる。IBM iのスプールファイルをWRKSPLFコマンドで参照した場合や従来のTSpool400コンポーネントで参照した場合は、この制御コードによって空行や改ページの挿入を内部的に判断している。

主な制御コードはそれぞれ以下の意味を持っている。

- 「 」（スペース）：何もしない。
- 「0」：上に1行あけて出力する。
- 「-」：上に2行あけて出力する。
- 「+」：改行せずに前の行の内容を上書きする。（ただしスペース文字の場合は上書きしない）
- 「1」：改ページを行う。

これらを実装するためには**【ソース9】**のように記述する。

ソース 9

一時ファイル内のスプールをTMemoに出力

```
const
  cFILE = 'YSADALIB/SPL2102'; // 一時ファイル名
var
  sANS: String; // ANSI用紙制御コードの保管用
  sTXT: String; // 制御コードを除いた文字列

  i: Integer;       // For文用
  sBEF: AnsiString; // 結合先（上側）の行の文字列
  sAFT: AnsiString; // 結合元（下側）の行の文字列
begin

  // 内容をメモに表示
  FDQuery3.Close;
  FDQuery3.SQL.Text := ' SELECT * FROM ' + cFILE;
  FDQuery3.FetchOptions.Mode := fmAll; // 一度に全件取得
  try
    FDQuery3.Open;
    Memo1.Lines.Clear;

    while not FDQuery3.Eof do
    begin
      sTXT := FDQuery3.Fields[0].AsString;

      // 整形しないでそのまま出力する場合
      if not(chkSEIKEI.Checked) then
      begin
        Memo1.Lines.Add(sTXT);
      end

      // 整形する場合
      else
      begin
        // 1桁目＝ANSI用紙制御コード
        sANS := Copy(FDQuery3.Fields[0].AsString, 1, 1);
        // 2桁目以降＝制御コードを除いた文字列
        sTXT := Copy(sTXT, 2, Length(sTXT));

        // 0 ＝上に1行あけて出力する
        if (sANS = '0') then
        begin
          Memo1.Lines.Add('');
          Memo1.Lines.Add(sTXT);
        end

        // - ＝上に2行あけて出力する
        else
        if (sANS = '-') then
        begin
          Memo1.Lines.Add('');
          Memo1.Lines.Add('');
          Memo1.Lines.Add(sTXT);
        end

        // + ＝改行せずに前の行の内容を上書きする
        else
        if (sANS = '+') then
        begin
          // 結合先（上側）の行の文字列
          sBEF := AnsiString(Memo1.Lines[Memo1.Lines.Count - 1]);
          sBEF := AddSISO(sBEF); // シフト文字代替スペースの付加【ソース10】
          // 結合元（下側）の行の文字列
          sAFT := AnsiString(sTXT);
          sAFT := AddSISO(sAFT); // シフト文字代替スペースの付加【ソース10】
          // 結合先（上側）が短い場合は結合元（下側）と長さを合わせる
          if (Length(sBEF) < Length(sAFT)) then
          begin
            sBEF := sBEF + StringOfChar(' ', Length(sAFT) - Length(sBEF));
          end;
          // バイト単位で文字を上書きする
          for i := 1 to (Length(sAFT)) do
          begin
            if (sAFT[i] <> ' ') then
            begin
              sBEF[i] := sAFT[i];
            end;
          end;

          // 結合後の文字列をメモの最下行に再セット
          sBEF := RmvSISO(sBEF); // シフト文字代替スペースの除去【ソース11】
          Memo1.Lines[Memo1.Lines.Count - 1] := sBEF;
        end

        // 1 ＝改ページを行う
        else
        if (sANS = '1') then
        begin
          // 改ページ時の処理は要件によって異なるため割愛
          Memo1.Lines.Add('★★ここで改ページ★★');
          Memo1.Lines.Add(sTXT);
        end

        // スペース＝何もしない
        else
        begin
          Memo1.Lines.Add(sTXT);
        end;
      end;

      FDQuery3.Next; // 一時ファイルの次の行へ
    end;

  finally
    FDQuery3.Close;
  end;
```

ファイルの内容を順次Memo1に書き出す

1桁目と2桁目以降を分割する

1桁目＝「0」の場合の処理

1桁目＝「-」の場合の処理

1桁目＝「+」の場合の処理

1桁目＝「1」の場合の処理

1桁目＝それ以外の場合の処理

すると、【図9】のように取得されていたデータが【図10】のように整形される。

図 9 整形前のスプール内容イメージ

```
1  5770SS1 V7R1M0 100423                    ジョブの処理                          21/08/21  3:40:27     ページ   1
0  ジョブ. :   VALENCE6D    ﾏｰﾅﾞ. . . :   OTMHHTTP      番号 . . . . :   671593
-                                          ジョブ状況属性
0    ジョブの状況 . . . . . . . . . . . . . :  END
     現行ユーザー・プロファイル . . . . . . :  OTMHHTTP
     ジョブのユーザー識別 . . . . . . . . . :  OTMHHTTP
       設定したジョブ . . . . . . . . . . . :    *DEFAULT
     システムへの投入:
       日付 . . . . . . . . . . . . . . . . :  21/08/16
       時刻 . . . . . . . . . . . . . . . . :  06:38:39
     開始:
       日付 . . . . . . . . . . . . . . . . :  21/08/16
       時刻 . . . . . . . . . . . . . . . . :  06:38:39
     サブシステム . . . . . . . . . . . . . :  QHTTPSVR
       サブシステム・プールID . . . . . . . :
     投入したジョブ:
       ジョブ名 . . . . . . . . . . . . . . :  VALENCE6D
       ユーザー . . . . . . . . . . . . . . :  OTMHHTTP
       番号 . . . . . . . . . . . . . . . . :  671568
     ジョブのタイプ . . . . . . . . . . . . :  BCI
     特殊環境 . . . . . . . . . . . . . . . :  *NONE
     プログラム戻りコード . . . . . . . . . :  0
     制御付き終了要求 . . . . . . . . . . . :  NO
     システム . . . . . . . . . . . . . . . :  MIGAROP7
1
+  5770SS1 V7R1M0 100423                    ジョブの処理                          21/08/21  3:40:27     ページ   2
0  ジョブ. :   VALENCE6D    ﾏｰﾅﾞ. . . :   OTMHHTTP      番号 . . . . :   671593
```

最初の桁にANSI用紙制御コードを持っている

図 10 整形済みのスプール内容イメージ

```
★★ここで改ページ★★
 5770SS1 V7R1M0 100423                    ジョブの処理                          21/08/21  3:40:27     ページ   1
ジョブ. :   VALENCE6D    ﾏｰﾅﾞ. . . :   OTMHHTTP      番号 . . . . :   671593

                                          ジョブ状況属性

  ジョブの状況 . . . . . . . . . . . . . :  END
  現行ユーザー・プロファイル . . . . . . :  OTMHHTTP
  ジョブのユーザー識別 . . . . . . . . . :  OTMHHTTP
    設定したジョブ . . . . . . . . . . . :    *DEFAULT
  システムへの投入:
    日付 . . . . . . . . . . . . . . . . :  21/08/16
    時刻 . . . . . . . . . . . . . . . . :  06:38:39
  開始:
    日付 . . . . . . . . . . . . . . . . :  21/08/16
    時刻 . . . . . . . . . . . . . . . . :  06:38:39
  サブシステム . . . . . . . . . . . . . :  QHTTPSVR
    サブシステム・プールID . . . . . . . :
  投入したジョブ:
    ジョブ名 . . . . . . . . . . . . . . :  VALENCE6D
    ユーザー . . . . . . . . . . . . . . :  OTMHHTTP
    番号 . . . . . . . . . . . . . . . . :  671568
  ジョブのタイプ . . . . . . . . . . . . :  BCI
  特殊環境 . . . . . . . . . . . . . . . :  *NONE
  プログラム戻りコード . . . . . . . . . :  0
```

ANSI用紙制御コードに従って改行や結合を行ったあとの状態

「+」の制御時のみDelphi側で実装する手順が少々複雑で、IBM i側のシフト文字を考慮して文字列の結合を行う必要がある。具体的には、結合元と結合先の文字列の中の全角文字と半角文字の境界に疑似的にシフト文字の代替となるスペースを付加**【ソース10】**し、結合完了後には逆に全角文字と半角文字の境界にセットされた疑似的なスペースを除去する**【ソース11】**。また、**【ソース9】**のロジックで結合を行っている箇所はバイト単位で結合する必要があるため、AnsiStringにキャストしている。この結果、2バイト(全角)文字の1バイト目または2バイト目だけが結合対象となった場合は文字化けを起こすことがあるが、IBM i側でWRKSPLFコマンド等を使ってスプールの内容を参照しても同様の結果になる。

ソース 10

シフト文字代替スペースの付加を行う関数の例

```
{*************************************************************************
 目的: 文字列に対して、SI/SOの位置にダミーの半角スペースをセット
       (※uses節にSystem.AnsiStringsが必要)
 引数: 元の文字列     戻値: 整形された文字列
*************************************************************************}
function AddSISO(AStr: AnsiString): String;
var
  i: Integer;
  S: AnsiString;
begin
  // (初期値) 1バイト目が全角の場合
  if (System.AnsiStrings.ByteType(AStr, 1) = mbLeadByte) then
    S := ' '
  // (初期値) 1バイト目が半角の場合
  else
    S := '';

  // 文字列を確認し、全角と半角の切替ポイントにダミーの半角スペースをセット
  for i := 1 to Length(AStr) do
  begin
    S := S + AStr[i];

    if (System.AnsiStrings.ByteType(AStr, i) = mbSingleByte) and
       (System.AnsiStrings.ByteType(AStr, i + 1) = mbLeadByte) then
    begin
        S := S + ' ';
    end;

    if (System.AnsiStrings.ByteType(AStr, i) = mbTrailByte) and
       (System.AnsiStrings.ByteType(AStr, i + 1) = mbSingleByte) then
    begin
        S := S + ' ';
    end;
  end;

  Result := String(S); // 結果を返却
end;
```

ソース 11

シフト文字代替スペースの除去を行う関数の例

```
{*************************************************************************
 目的: 文字列に対して、SI/SOの位置にセットしたダミーの半角スペースをカット
 引数: 元の文字列(※SI/SO考慮済みの文字列が入る前提)     戻値: 整形された文字列
*************************************************************************}
function RmvSISO(AStr: WideString): String;
var
  i: Integer;
  bSO: Boolean;
begin
  Result := '';
  bSO    := False;
  if AStr <> '' then
  begin
    // 文字単位で後ろからカウント
    for i := Length(AStr) downto 2 do
    begin
      // 1つ後の文字でフラグセットされた半角スペース(SO)のとき、セットしない
      if (bSO) then
        bSO := False
      else
      // 半角スペース(SI)かつ1つ前の文字が全角のとき、セットしない
      if (AStr[i] = ' ') and (Length(AnsiString(AStr[i - 1])) = 2) then
      begin  end  // 何もしない
      else
      begin
        // 文字をセット
        Result := AStr[i] + Result;
        // 全角かつ1つ前の文字が半角スペース(SO)のとき、フラグセット
        if (Length(AnsiString(AStr[i])) = 2) and (AStr[i - 1] = ' ') then
          bSO := True;
      end;
    end;

    // 全角始まりでない場合は1文字目(SIでない)を最後に足す
    if (not bSO) then
      Result := AStr[1] + Result;
  end;
end;
```

なお、制御コードを全く使用せずにただスプールの内容を順次参照するだけでよい場合は、**【ソース8】**のCPYSPLFコマンド発行時に『CTLCHAR(*FCFC)』パラメータを除外すれば、ANSI用紙制御コードが無い状態の文字列を取得できる。状況に合わせて使い分けていただきたい。

4.まとめ

Delphi/400では過去バージョンでコーディングされたプログラムや古くから存在するIBM iの資産を活用できることが大きな利点ではあるが、Windowsや周囲の環境の技術革新に伴ってDelphi/400も進歩を続けており、特にアプリケーションを新規開発する場合は最新のFireDAC接続を活用した方が得策であると言える。
本稿の記述を参考に、最新バージョンのDelphiならびにDelphi/400に関心をもっていただければ幸いである。