最優秀賞

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

吉岡 延泰 様
日本調理機株式会社
情報システム部
主任

日本調理機株式会社
http://www.nitcho.co.jp/

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

開発の経緯

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

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

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

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

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

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

前提

5250のメニュー画面は、自社独自の構成になっている。【図1】

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

解決したい既知の問題点

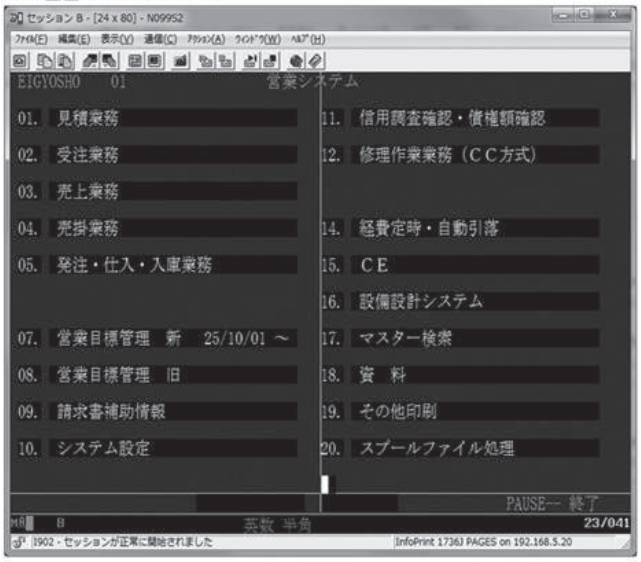
・構造化されたシステムの全体像を把握するのに時間がかかり、大規模な改修が容易に行えない。問題が発生した際にも、調査に時間がかかる場合が多かった。

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

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

図1

5250画面のメニューイメージ



メニューファイル(PFソース)

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

図2

ライブラリ管理(PFソース)

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

サインオン管理(PFソース)

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

調査を行った項目(例)

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

複雑になるという悪循環を招いていた。

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

事前調査

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

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

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

アジャイル開発向けシステムの構成と要素

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

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

（1）～（5）の概要を【図 5】に示す。

（1）オブジェクト情報出力

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

（2）クローラー

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

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

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

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

（3）構造の把握（Delphi）

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

（4）構造の検索・リスト化（Delphi）

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

（5）一括コンパイル

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

（その他 1）メニューの検索・展開（Delphi）

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

（その他 2）ソース検索結果のダウンロード

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

（その他 3）従属論理ファイル調査プログラム

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

（その他 4）ソースとオブジェクトの整合性調査・一括棚卸し

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

実際には、（2）～（5）以外のツールは事前調査の時点で有効なものである。

アジャイル開発向けシステム構築のポイント

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

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

図3

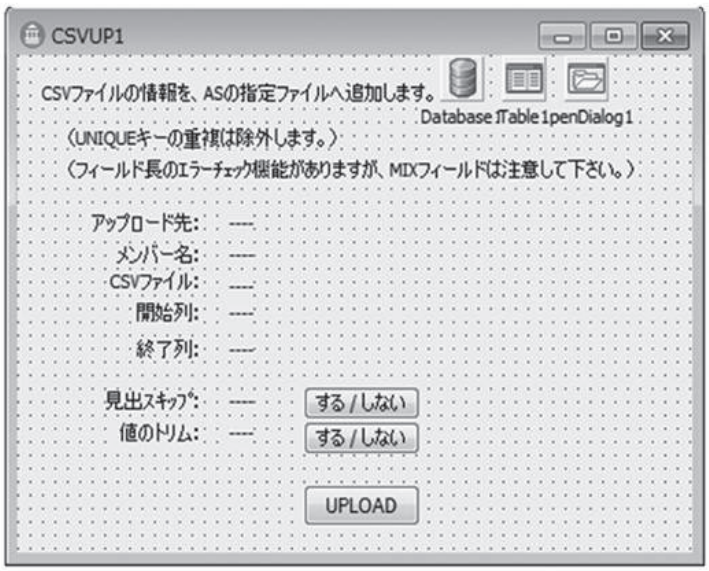
① 汎用ダウンロード（非表示）

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

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

図4

② 汎用アップロード（起動時にファイル選択ダイアログを表示）



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

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

うな構成になっているのかを確認しながら構築することが重要である。

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

効果

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

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

今後の展望と課題

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

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

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

総評

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

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

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

M

図5

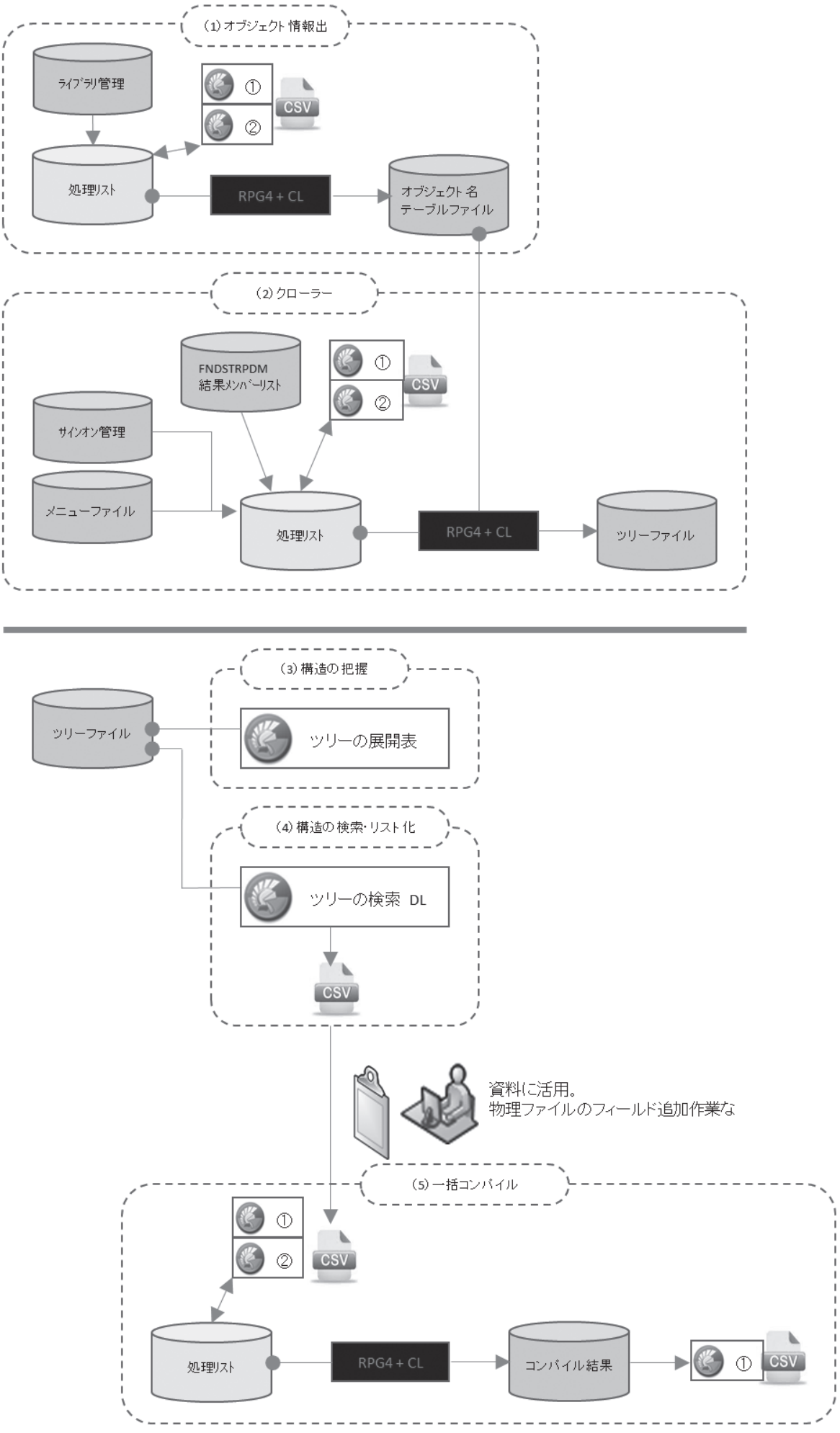

(1) オブジェクト情報出
ﾗｲﾌﾞﾗﾘ管理
①
②
CSV
処理ﾘｽﾄ
RPG4 + CL
オブジェクト名
テーブルファイル
(2) クローラー
FNDSTRPDM
結果ﾒﾝﾊﾞｰﾘｽﾄ
①
②
CSV
ｻｲﾝｵﾝ管理
メニューファイル
処理ﾘｽﾄ
RPG4 + CL
ツリーファイル
(3) 構造の把握
ツリーファイル
ツリーの展開表
(4) 構造の検索･リスト化
ツリーの検索 DL
CSV
資料に活用。
物理ファイルのフィールド追加作業な
(5) 一括コンパイル
①
②
CSV
処理ﾘｽﾄ
RPG4 + CL
コンパイル結果
①
CSV

図6

ライブラリからオブジェクト情報の一覧を書き出す （CLソースの一部）

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

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

オブジェクト名テーブルファイル

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

図7

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

一時的に使用する物理ファイル(PFソース)

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

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

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

一時的に使用する物理ファイル(PFソース)

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

スプールして一時ファイルにコピーする(CLソースの一部)

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

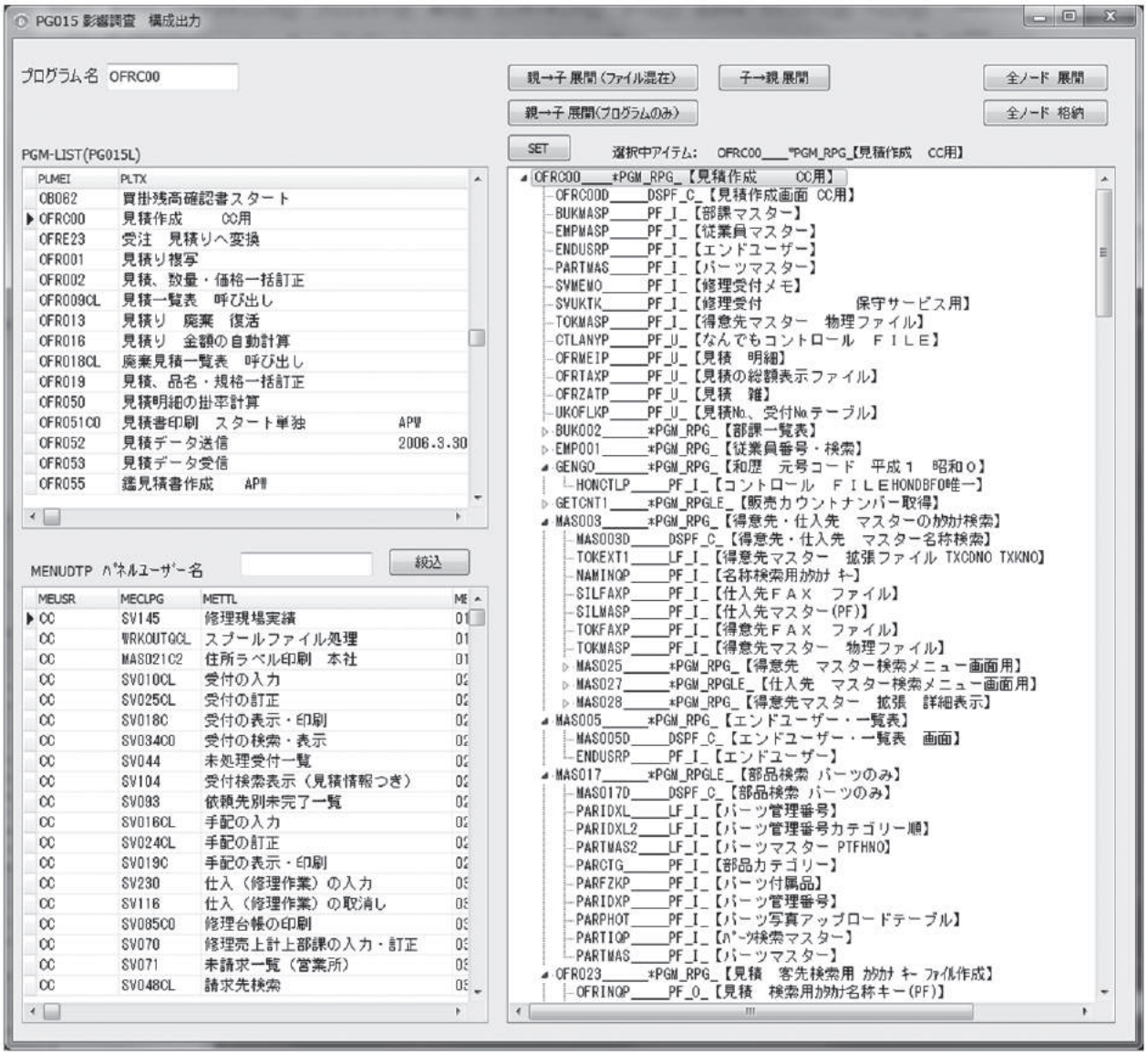
それぞれの一時ファイルをから取り出したオブジェクト名称などの情報をツリーファイルに書き出す。

ツリーファイル(PFソース)

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

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

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

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

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

K PROMEI
K PRKMEI
K PRFMEI
```

図8

画面イメージ



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

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

図8-2

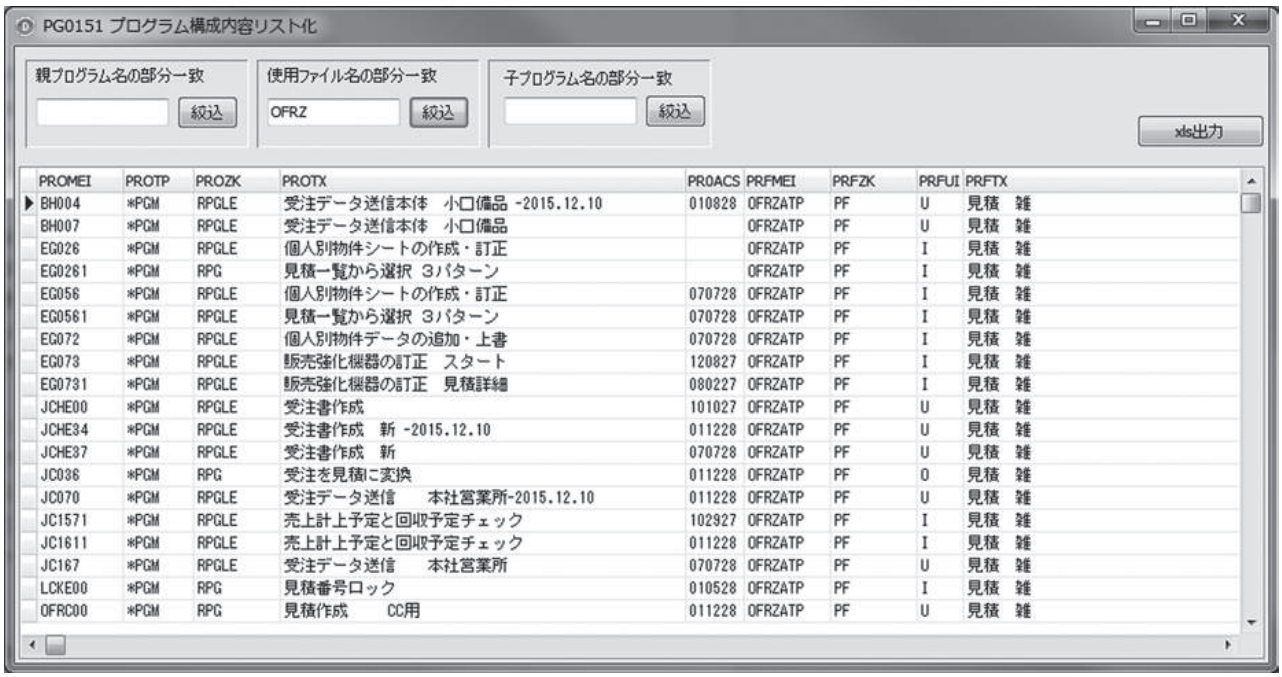
ツリーファイルの親子関係を展開する。(Delphiソース)

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

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

図8-3

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

図9



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

図10

ソースの種類に合わせてコンパイルする(CLソースの一部)

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