# 簡単リッチWebアプリケーション作成！Biz/Browser入門（後編）

■出だし好調、Biz/Browser V

アクシスソフト社によるとBiz/Browser Vのリリース後、Biz/Browserのダウンロード件数が伸びている様です。やはりBiz/Designerの無償化が大きいのでしょうか。アクシスソフト社のBiz/Browser Vプロモーションイベントにもかつて導入を検討されていた企業が再び戻ってくるなど、改めて期待も高まっている様です。

今までも雑誌のRIA／リッチクライアントの特集などでは海外のプラットフォームに並んで紹介されてきたBiz/Browserですが、今後益々注目されていくプラットフォームかもしれません。

■Biz/Browserの得手不得手

さて前回の説明で、Biz/Browserでは直観的なGUI操作で画面レイアウトが作成できることと、その内部処理はイベントドリブン方式であることが理解できたと思います。

これらは非常にVBアプリケーションの開発と似ています。

また、用意されている画面コントロールやイベントもVBと似ているため、VBで作成されたアプリケーションを再現することが比較的容易にできます。

実際筆者も、Biz/Browserの適用を相談されるケースでは、既存のVBアプリケーションの画面レイアウトや操作性を維持したまま移行したいという要望が多いです。

その際は、主に以下の様なメリット/デメリットがあることを伝えつつ、検討を促すようにしています。

【Biz/Browserのメリットデメリット】

　メリット

　　・一般的なWebアプリケーションより高速（AjaxによりBiz/Browserと同等のUIを作成した場合）

　　・一般的なWebアプリケーションより通信量が少ない

　　・VBライクな画面の作成が容易（VBアプリケーションの再現が容易）

　　・入力制御用コントロールが標準機能として豊富に用意されている

　　・クライアント側のローカルリソースにアクセスできる

　　・外部デバイスとの連携ができる

　　・.Netクラス実行機能（今回のバージョンより追加）

　　・日本のメーカなので問い合わせがしやすい（今回のBiz/Browser Vのリリースに伴いより更に体制を強化したとの事）

　デメリット

　　・VBより低速

　　・一般的なWebアプリケーションよりクライアント側メモリを消費する

　　・世の中に多く出回っている便利なJavaScriptライブラリやcssを適用できない

　　（デザイン的に今時のお洒落なユーザインタフェースが苦手）

　　・.NetやJava等に比べてインターネットで取得できる技術情報が少ない

　　・.NetやJava等に比べてBiz/Browserの技術者が少ない

　　・Biz/Browserのインストールが必要

　　・Biz/Browserのライセンス料がかかる

筆者の私見として、もっとザックリと言ってしまえば、不特定多数のコンシューマ向けで、お洒落なWebアプリケーションを作成したければHTML5やCSS、JavaScriptに対応した一般的な技術を、エンタープライズ向けで、見た目の派手さより操作性やパフォーマンスを重視するならBiz/Browserを、といった住み分けで考えています。

■本当に高機能？新スプレッドシートコントロールを使ってみる

Biz/Browserは業務画面で必要となりそうな入力制御用コントロールを標準機能として搭載しているのが一つの売りとなっています。

表形式のコントロールに関しても Spread や FlexView といったコントロールが提供されています。

しかし、これらは実はあまり高機能とは言えず、例えばグレープシティ社製品のSPREADと同等の機能を実現しようとすると、前述のコントロールをかなりの拡張する必要がありました。

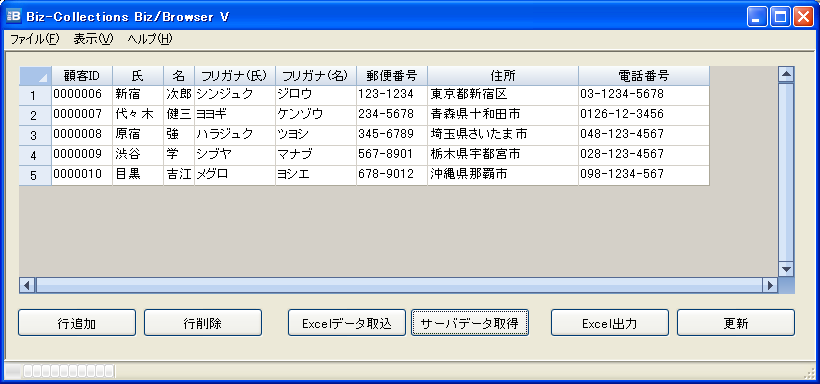
筆者もFlexViewを独自に拡張したライブラリを持っていますが色々な機能を盛り込んだ結果、さすがにパフォーマンスに難が出て、大量データの処理には不向きなものとなっています。ですので、大量データの処理が必要な場合は、その拡張ライブラリは使用せず、改めてFlexViewに、そのアプリケーションに必要な機能のみを実装するようにしています。

そんな折、Biz/Browser Vは、今回Excelに匹敵する高機能スプレッドシートコントロールを標準搭載した、と大々的にアピールしています。

これが本当であれば大変喜ばしいことなので、早速、新しく追加されたSSpreadコントロールを触ってみたいと思います。

今回作成するサンプルアプリケーションは主に以下の処理を行います。

画面イメージ



　処理①　ExcelからSSpreadにデータをインポート

　処理②　SSpread上でデータを編集（行の追加削除、サーバ送受信によるデータ取得）

　処理③　処理②のデータをサーバへ送信、又は、Excel出力

今回は、前回説明できなかったサーバ通信処理も説明しようと思いますが、ここではサーバアプリケーションの開発方法までは説明できない為、サーバ側に以下のファイルを置き、それを取得してサーバ処理結果とすることとします。

実際の開発でも、Biz/Browser側の単体テストでこのような手法を良く使います。

Webサーバについては、今回も前編同様にIISサーバをローカルPC内に立てたと想定し説明します。

以下のファイルを、IISのホーム仮想ディレクトリ「C:\Inetpub\wwwroot」に配置して下さい。

customerList.csv

0000006,新宿,次郎,シンジュク,ジロウ,123-1234,東京都新宿区,03-1234-5678

0000007,代々木,健三,ヨヨギ,ケンゾウ,234-5678,青森県十和田市,0126-12-3456

0000008,原宿,強,ハラジュク,ツヨシ,345-6789,埼玉県さいたま市,048-123-4567

0000009,渋谷,学,シブヤ,マナブ,567-8901,栃木県宇都宮市,028-123-4567

0000010,目黒,吉江,メグロ,ヨシエ,678-9012,沖縄県那覇市,098-1234-567

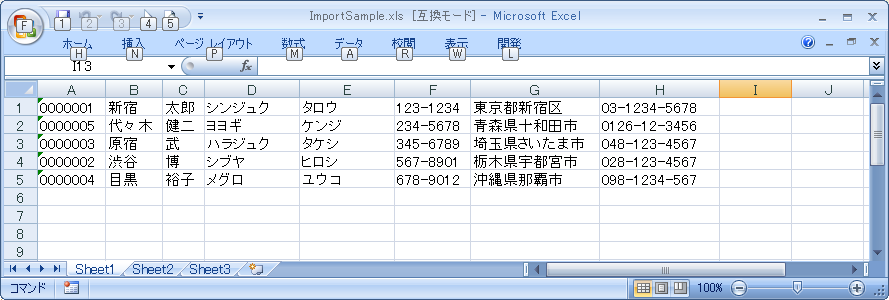
update.csv

更新が完了しました！

また、アプリケーションから取り込むエクセルファイルもローカルPCのどこかに保存しておいてください。

ImportSample.xls

。



なお、サーバ処理結果をCSV形式で返すのは、CRSにはCSVDocumentクラスというCSV操作に特化したクラスがあり、それを利用することで比較的簡単に内部処理を実装できることから、Biz/Browserアプリケーションでは一般的となります。

勿論、Biz/BrowserはXMLにも対応しているので、通信内容がXMLでも構いません。

■画面作成

まず、前編同様にGUI操作で画面イメージを作成します。

**②**

**③**

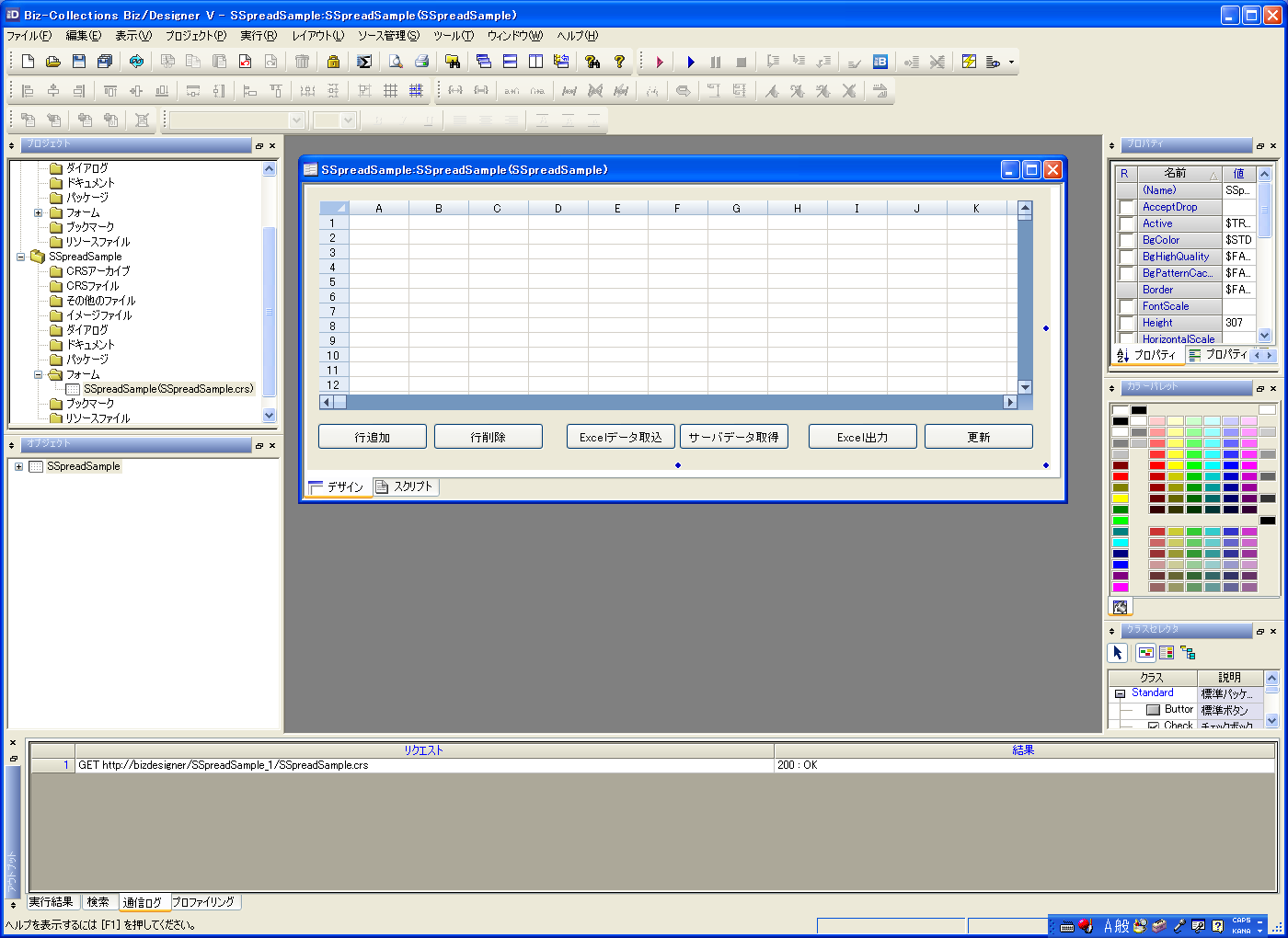
**④**

**⑤**

**⑥**

**⑦**

**①**



|  |  |  |  |
| --- | --- | --- | --- |
| **番号** | **クラス名** | **オブジェクト名** | **説明** |
|  | SSpread | ssSample | 一覧データ表示・編集用 |
|  | Button | btnRowInsert | 一覧表・行追加用ボタン |
|  | Button | btnRowDelete | 一覧表・行削除用ボタン |
|  | Button | btnImport | Excelデータ取込用ボタン |
|  | Button | btnGetData | サーバデータ取得用ボタン |
|  | Button | btnExport | Excelファイル出力用ボタン |
|  | Button | btnUpdate | サーバ更新用ボタン |

この時点でSSpreadオブジェクトの宣言に位置情報と大きさが記述されています。

□SSpreadオブジェクト宣言部

SSpread ssSample {

X = 13;

Y = 14;

Width = 776;

Height = 228;

}

今回のサンプルではこれ以外のSSpread操作は、実行時の初期化処理と各ボタンが押されたときに行います。

同様の処理をオブジェクト宣言部に予め記述しておくことは可能ですが、その内容が反映されるのは実行時のみの様です。

□実行時の初期化処理

ssSample{ } で囲むことで、そのオブジェクト配下のメンバに直接アクセスすることが出来る。（ssSampleオブジェクトへのパスの記述が省略できる）

行インデックスに 0 を指定することでヘッダ部を操作できる。

行インデックスに-1 を指定することですべての行を指定できる。

実行時のみSSpread初期化用関数を呼び出す。

SSpread初期化用に用意した関数。

if ( !$DESIGNTIME ) {

/\* 初期化 \*/

initializeSpread();

}

/\* SSpreadの初期化 \*/

function initializeSpread(){

setColTitle();

setWidthHeight();

ssSample{

MaxRows = 0;

MaxCols = 8;

AllowDragDrop = $FALSE;

}

}

/\* 列のタイトル設定 \*/

function setColTitle(){

ssSample{

SetText(1, 0, "顧客ID");

SetText(2, 0, "氏");

SetText(3, 0, "名");

SetText(4, 0, "フリガナ(氏)");

SetText(5, 0, "フリガナ(名)");

SetText(6, 0, "郵便番号");

SetText(7, 0, "住所");

SetText(8, 0, "電話番号");

}

}

/\* SSpreadの幅と高さを設定 \*/

function setWidthHeight(){

ssSample{

ColWidth(1) = 60;

ColWidth(2) = 50;

ColWidth(3) = 30;

ColWidth(4) = 80;

ColWidth(5) = 80;

ColWidth(6) = 70;

ColWidth(7) = 150;

ColWidth(8) = 130;

RowHeight(-1) = 19;

}

}

■Excelデータをインポート

各ボタンの処理は OnTouch イベントハンドラに記述します。

まず、Excelデータ取込用ボタンの処理を作成してみます。

SSpreadにはExcelファイルを読み込んでその内容をインポートする機能があり、これを使います。

※ただしこれは、対応しているのがExcel 2003までの形式となる為、Excel2007以降のファイル形式を取り込みたい場合は、Excel 2003までのxls形式で保存し直す必要が有ります（2011/07/??※日付忘れない※時点）。

以下、ソースの説明です。

"Excelデータ取込"ボタンの定義。

Excelファイルのシート（インデックス０）からデータを取り込む。

Excelファイルを操作する為のFileSystemオブジェクトを定義。

ローカルPC内のファイルを選択する為のダイアログボックスをポップアップしExcelファイルをFileオブジェクトに格納。

ダイアログによるファイルの選択が中止された場合の例外処理。

SSpreadオブジェクトの各種再設定処理

ボタンが押された時のOnTouchイベントハンドラを定義。

処理本体を記述した関数を呼び出すだけ。

Button btnImport {

X = 281;

Y = 256;

Width = 120;

Height = 29;

Title = "Excelデータ取込";

function OnTouch(e){

SSpreadSample.importExcel();

}

}

～中略～

function importExcel(){

if(//.MessageBox("表示中のデータは破棄されます。\r\nよろしいですか？",

"確認",

$OKCANCEL + $ICONQUESTION) != OkSelected){

return;

}

/\* Excelシートからデータを取得し、SSpreadに表示 \*/

var fs = New FileSystem();

var f;

Try{

f = fs.OpenDialog("ファイルを開く", "\*.xls", "xls", "ImportSample.xls");

}Catch(exp){

if(exp.Code == 21){

return;

}else{

throw exp;

}

}

ssSample{

ImportExcelSheet(f, 0);

/\* 列タイトル・プロパティの再設定 \*/

SSpreadSample.setColTitle();

MaxRows = DataRowCnt;

MaxCols = 8;

AllowDragDrop = $FALSE;

}

}

スプレッドの見た目を整える処理なども記述されていて複雑に見えますが、Excelデータを取り込んで表示する処理そのものはSSpreadに用意されているImportExcelSheetメソッドを呼び出すだけとなります。

なお、今回のサンプルでイベントハンドラ内の記述は、関数を呼び出すだけのシンプルな記述としています。

特にイベントハンドラ内に複雑な処理を記述しても問題は有りませんが、筆者が実際の業務アプリケーションを構築する際には、ソースの可読性を意識して、画面オブジェクトの定義部分は極力シンプルにして、煩雑な処理は分けて書くようにしています。

これ以外のボタンの処理も説明していきます。

□行追加用ボタンのイベントハンドラから呼ばれる処理

追加したい行数を引数に設定。

MaxRowsプロパティの値を変更することで行数を変更する。

/\* 行追加 \*/

function insertRow(rowCount){

ssSample{

MaxRows = MaxRows + rowCount;

}

}

□行削除用ボタンのイベントハンドラから呼ばれる処理

SelBlockCol、SelBlockCol2は、選択中のセルの左端と右端の列番号を取得するプロパティ。行選択がされている場合は-1が返る。

DeleteRowsメソッドで行を削除。

/\* 選択された行の削除 \*/

function deleteRow(){

var deleteRowCount = 0;

ssSample{

/\* 1行以上の行が選択されたときのみ動作 \*/

if(SelBlockCol == -1 && SelBlockCol2 == -1){

deleteRowCount = SelBlockRow2 - SelBlockRow + 1;

DeleteRows(SelBlockRow, deleteRowCount);

/\* 削除行分だけ全体行数を詰める \*/

MaxRows = MaxRows - deleteRowCount;

}

}

}

□Excelファイル出力用ボタンのイベントハンドラから呼ばれる処理

ExportExcelBookメソッドでファイル出力。

/\* Excelにエクスポート \*/

function exportExcel(){

var fs = New FileSystem(FileSystem.PUBLIC\_ROOT);

var f;

Try{

f = fs.SaveDialog("ファイルに保存", "\*.xls", "xls", "ExportSample.xls");

}Catch(exp){

if(exp.Code == 21){

return;

}else{

throw exp;

}

}

ssSample.ExportExcelBook(f);

}

■サーバ通信処理

ここまでの説明でExcelファイルから読み込んだデータをスプレッドシート上で編集し、またExcelファイルへ出力する処理は記述できました。

しかし、実際の業務アプリケーションではBiz/BrowserからWebサーバへアクセスしデータの取得、更新を行うことになると思います。

少しSSpreadの説明から離れますが、サーバ通信処理の説明をしておきます。

今回はサーバとの通信処理を一つの関数にまとめました。その通信処理用関数は、引数に①サーバのURL（ルートからの絶対アドレス）、②サーバへ送信するCSV文字列、③サーバへのリクエスト形式（GET又はPOST）、を指定します。

戻り値はサーバとの通信結果にあたるCSV文字列です。

□サーバ通信処理用関数

POSTメソッドでリクエストを送信

GETメソッドでリクエストを送信

リクエストのメッセージボディにCSVを追加。

リクエストを生成。

セッションオブジェクトを取得。

/\* サーバ通信 \*/

function communicateHttpResponse(requestUrl, csvString, methodName){

var session = findHttpSession("http://localhost");

var request = session.CreateRequest(Str(requestUrl), "UTF-8");

if(csvString != null){

request.addBody(csvString);

}

var response;

switch (methodName) {

case "GET":

response = session.get(request);

break;

case "POST":

response = session.post(request);

break;

}

return response.read();

}

この関数を使って、サーバデータ取得用ボタンとサーバ更新用ボタンの処理を実装します。

□サーバデータ取得用ボタンイベントハンドラ

サーバからCSVデータを取得する。

サーバから取得したCSV文字列をスプレッドに反映させる処理。（後述）

function OnTouch(e){

/\* データ送受信 \*/

var csvData = SSpreadSample.communicateHttpResponse("CustomerList.csv", null, "GET");

/\* SSpreadに反映 \*/

SSpreadSample.putCSVStringDataToSSpread(csvData);

}

□サーバ更新用ボタンイベントハンドラ

サーバから取得したCSV文字列をスプレッドに反映させる処理。（後述）

サーバへCSV文字列を送信し、結果を取得する。

function OnTouch(e){

/\* CSVデータ取得 \*/

var csvString = SSpreadSample.getCSVFromSSpread();

/\* データ送受信 \*/

var returnString = SSpreadSample.communicateHttpResponse("update.csv", csvString, "POST");

/\* 受信メッセージを表示 \*/

//.MessageBox(returnString);

}

続いて、この処理の中で行っている、スプレッドの処理を説明します。

□CSV文字列をスプレッドに設定する処理

CSV文字列をタブ区切り文字列に変換した後、Clipプロパティを使用して、指定したセルブロックのデータを一括で設定。

Row,Col,Row2,Col2プロパティでデータを設定するセルブロック（範囲）を指定。

function putCSVStringDataToSSpread(csvString){

/\* データ件数をカウント \*/

var csvDoc = new CSVDocument();

csvDoc.parse(csvString);

var rowCount = csvDoc.rows;

/\* SSpreadに反映 \*/

ssSample{

SSpreadSample.insertRow(rowCount);

Row = DataRowCnt + 1;

Col = 1;

Row2 = Row + rowCount;

Col2 = 8;

Clip = csvString.replace(",", "\t");

}

}

□スプレッドからCSV文字列を取得する処理

タブ区切り文字列をCSV文字列に変換。

Clipプロパティを使用して、指定したセルブロックのデータを一括で取得。

function getCSVFromSSpread(){

var clipData;

/\* データ部をTSVテキストで取得 \*/

ssSample{

Row = 1;

Col = 1;

Row2 = DataRowCnt;

Col2 = DataColCnt;

clipData = Clip;

}

/\* タブをカンマに変換 \*/

var csvString = clipData.replace("\t", ",");

return csvString;

}

以上がサンプルアプリケーションの機能の説明となります。

早速アプリケーションを実行して、エクセルやサーバと連携しているのを確認してみてください。

サンプルアプリ「Sample.zip」

**メモ**

ローカル上でIISを使用してサンプルを実行する場合、更新ボタンをクリックするとHTTP405エラーが発生することがあります。

これはIIS側でPOSTメソッドが許可されていないために発生している可能性があります。

その場合は、コンピュータの管理画面からWebサイトのプロパティを選択して、[ホームディレクトリ]⇒[構成]⇒[マッピング]⇒[追加]をクリックして、

拡張子に「.csv」、実行ファイルのパスにIISのシステムディレクトリにある「ssinc.dll」、動詞に「POST」を追加します。

もしOKボタンがグレーアウトされて押せない場合は、一度、実行ファイルの部分をクリックすると押せるようになります（古いIISのバグの様です）。

今回SSpreadを使ってみた感想としては、今までのSpreadやFlexViewの様に、CSV形式との連携を意識した作りになっていないのは少し不便に感じました。

とは言え、SSpreadは本当に豊富な機能をそろえており、今回説明した機能はほんの一部で、それ以外も見渡してみたところ、業務アプリケーションに必要となりそうなものは一通り装備されていると言って良さそうです。

例えると、当時、業務用VBアプリケーション開発でスタンダードの様に使われていたグレープシティ社のSPREAD（当時のバージョン）と同等レベルの機能を持っている様です。

これを標準機能として持っているのはBiz/Browser にとって強みと言えます。

■Biz/Browserの中にブラウザを表示

さて、今回のサンプルの中では紹介できませんでしたが、Biz/Browser Vから搭載された機能で、筆者の興味を引いた機能を紹介しておきます。

その一つにHtmlViewクラスがあります。

HtmlViewによりBiz/Browserの内部でInternet Explorerを起動することが出来るようになります。

今まではInternet Explorerの中でBiz/Browserを起動できたのですが、なかなか制御が難しくあまり使いたい機能ではありませんでした。

ですので、今回の逆転の発想は、大したことがなさそうで、結構使い勝手が良く、嬉しい機能の一つです。

これによりインターネットをBiz/Browserアプリケーションから閲覧できるのは勿論のこと、サーバ側にあるPDFファイルを表示し印刷することなども容易になります。

□画面イメージ



この他にも、.NETクラスライブラリの呼び出し機能や圧縮通信、非同期通信機能など柔軟なアプリケーション開発を可能とする、色々な機能が追加されています。

以前、Biz/Browserの導入を検討して断念した人も、改めて今回のV に触れてみると良いように思います。

■最後に

さて、前後編にわたり色々とBiz/Browserについて説明してきましたが、このBiz/Browserの最大の特徴は、有る意味、日本製であると言えるかもしれません。

機能的に海外製のRIA／リッチクライアント・プラットフォームに負けているとは思いません。

Biz/Browserはあくまでもエンタープライズ向けに特化したプラットフォームですから、守備範囲は狭いものの、その範疇の中では海外製のプラットフォームよりも優位な点は多々あります。

また、流行り廃りの早い海外製品に対して、アクシスソフト社の永井社長は「アクシスソフトがある限りBiz/Browserをサポートし続ける」と恒久的なサポートを宣言しています。

この様な背景も踏まえ、Microsoft系かオープンソース系かという議論の中に、日本製は？という判断要素を取り入れては如何でしょうか。

以上