【セッションNo. 3】

プログラミングテクニックセッション1

基幹システムとの連携手法をご紹介
ーインターネットEDIとの連携ー

株式会社ミガロ.
システム事業部　システム2課
辻野　健

アジェンダ

1. インターネットEDIについて

2. DelphiからのEDIサービス利用方法
   - 2-1. 通信方式
   - 2-2. 通信データ形式
   - 2-3. EDIデータの送信
   - 2-4. EDIデータの受信
   - 2-5. RESTサーバーアプリ作成方法（補足）

3. まとめ

１．インターネットEDIについて

1．インターネットEDIについて

- EDIとは？

Electronic Data Interchange（電子データ交換）

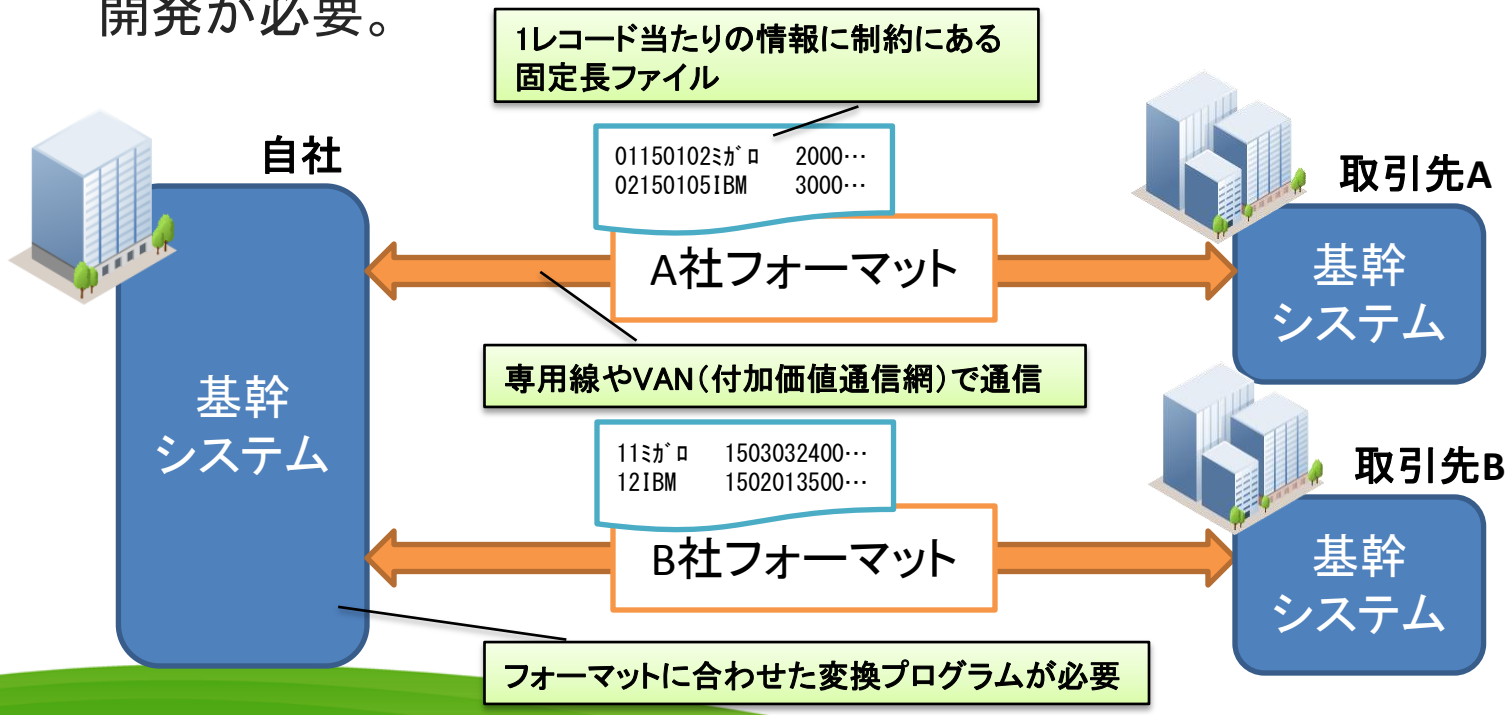
複数の企業間で、商取引のための各種情報（注文書や請求書等）を、
お互いのコンピュータが通信回線（ネットワーク）を介して交換すること。

（メリット）

- 複数の企業間での取引や精算を、広域かつリアルタイムに行える
- 企業内における事務手続き・作業量削減、業務コスト削減

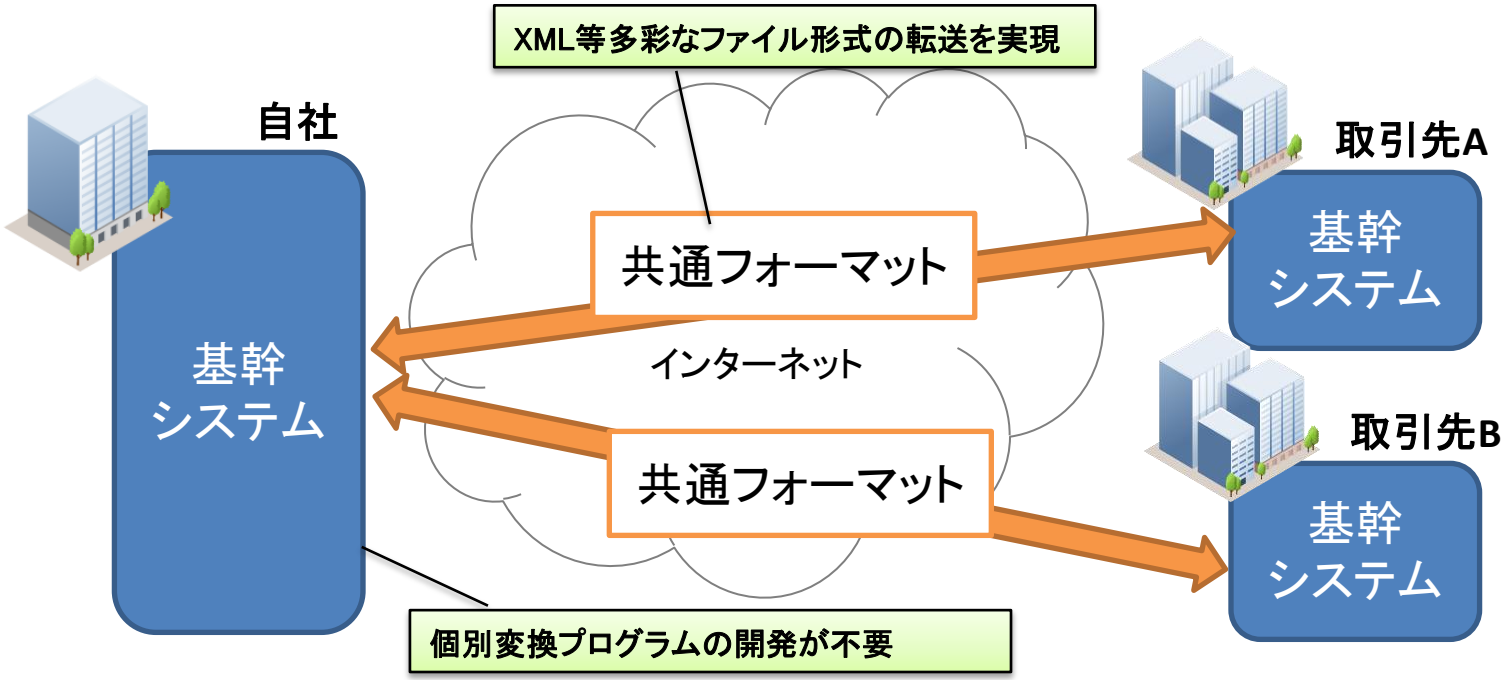
1．インターネットEDIについて

- 従来からあるEDI連携の課題
  - ① 通信手段に専用線やVAN（付加価値通信網）を利用しているため導入や運用にコストがかかる。
  - ② 一般的に低速な通信環境の為、転送データに固定長ファイルを使用することが多く、転送情報量に制限がある。
  - ③ 連携する企業ごとにフォーマットが異なる為、個別変換プログラムの開発が必要。



1．インターネットEDIについて

- EDI標準化の推進
  - ① 通信網のインターネット化により、回線コストが軽減された。
  - ② 回線速度が速くなり、大量のデータ転送が可能となった為、自由なデータ転送が可能になった。
  - ③ フォーマットの標準化により、連携が単純化・統一化された。



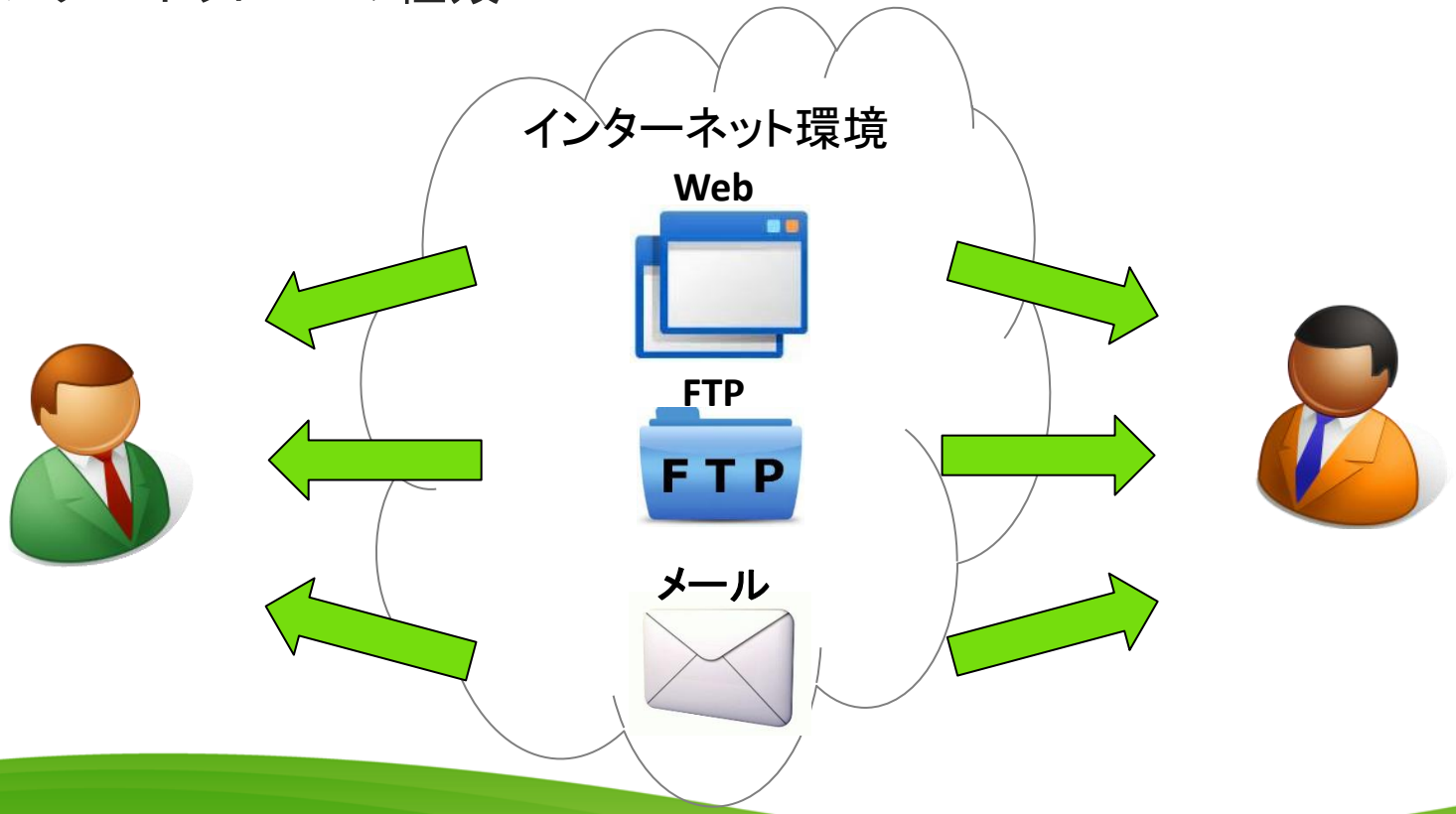
今回のテーマ ： インターネットを使用したEDIデータ連携

1．インターネットEDIについて

- インターネットEDIとは？

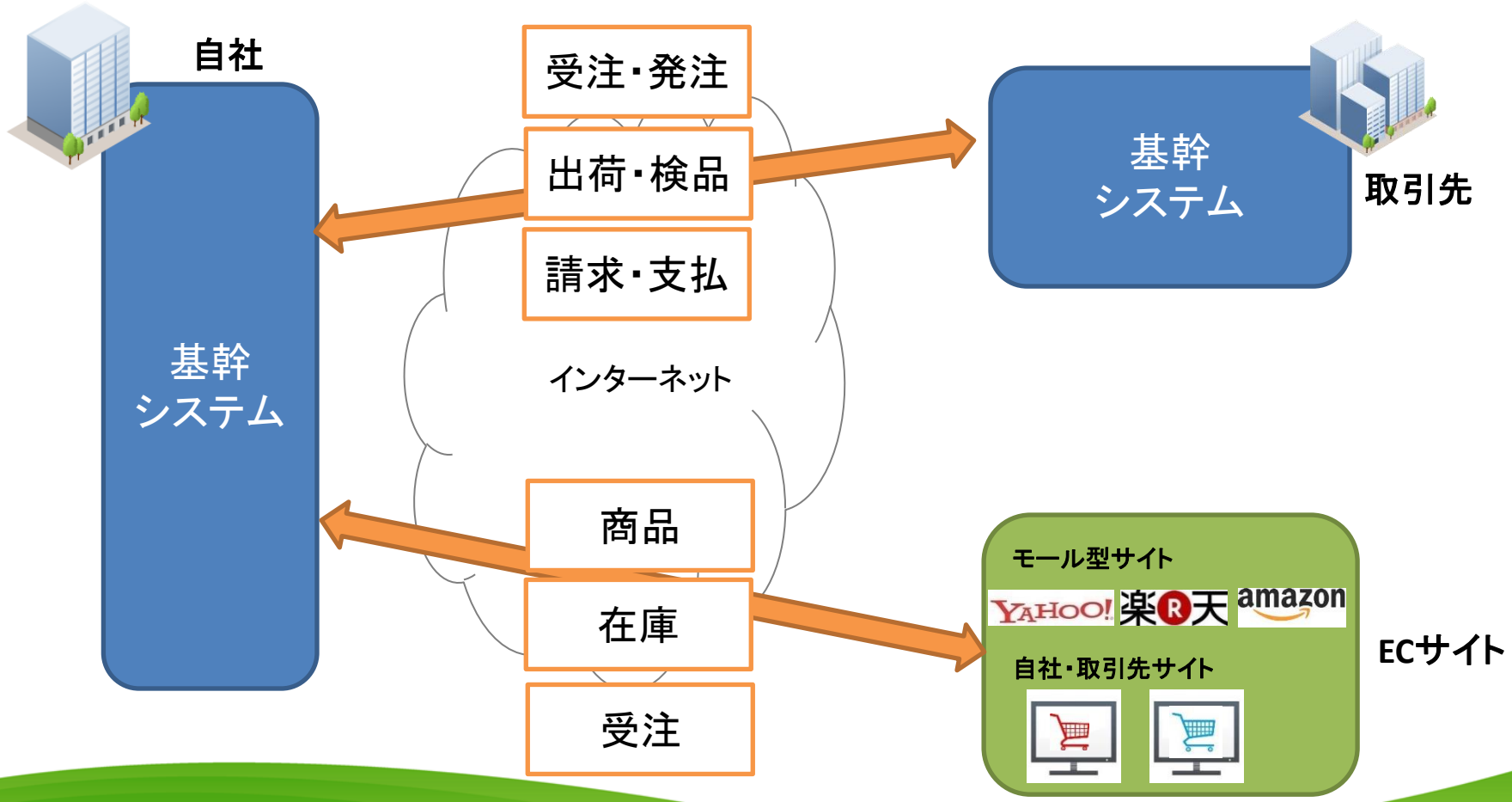
  従来のEDIは通信手段にコストがかかるが、インターネットを通信手段とすることによりコストを削減できるようになったEDI。

  インターネットEDIの種類



1．インターネットEDIについて

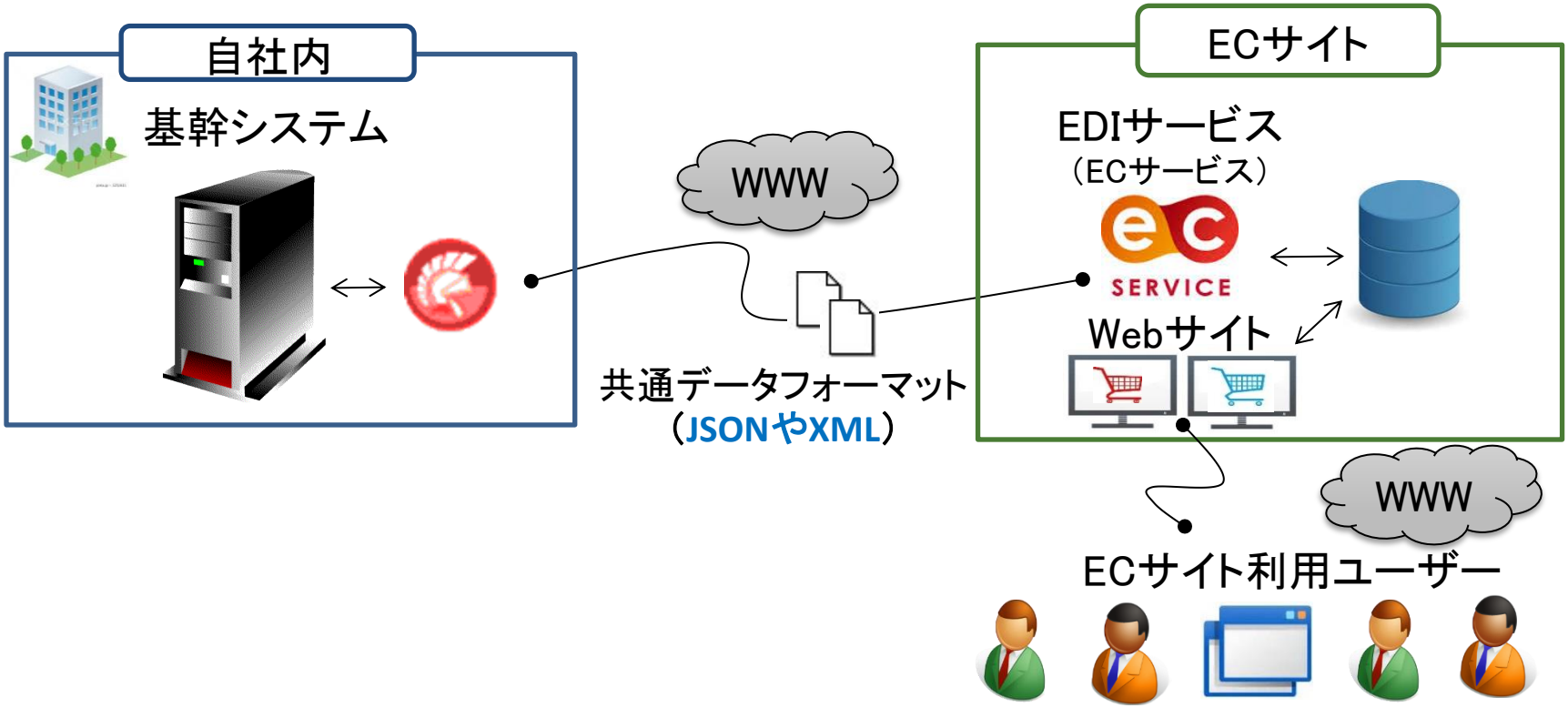
- インターネットEDIを使った連携
  インターネット化により、EDIは企業間通信のみならず、
  基幹システムとECサイト等とのデータ連携も実現。



1．インターネットEDIについて

- ECサイトが提供するEDIサービス

  基幹システムのデータを、提供されるEDIサービスを使って送信したり、
  ECサイトからのデータを基幹システムに受信することで、
  データ連携の自動化が可能になる。

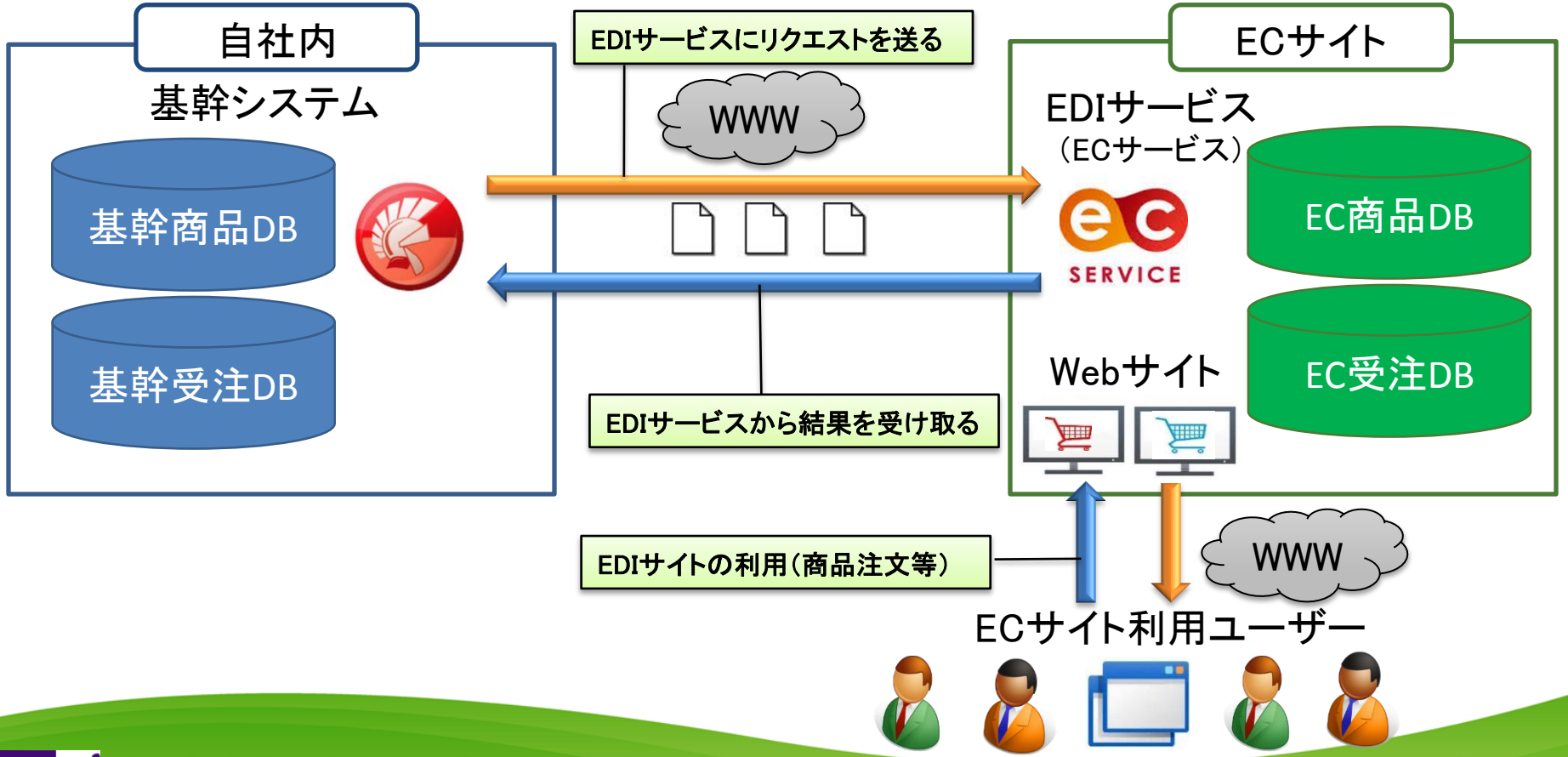


基幹システムとECサイトとの連携を例に、インターネットEDIの開発手法をご紹介

2. DelphiからのEDIサービス利用方法

2. DelphiからのEDIサービス利用方法

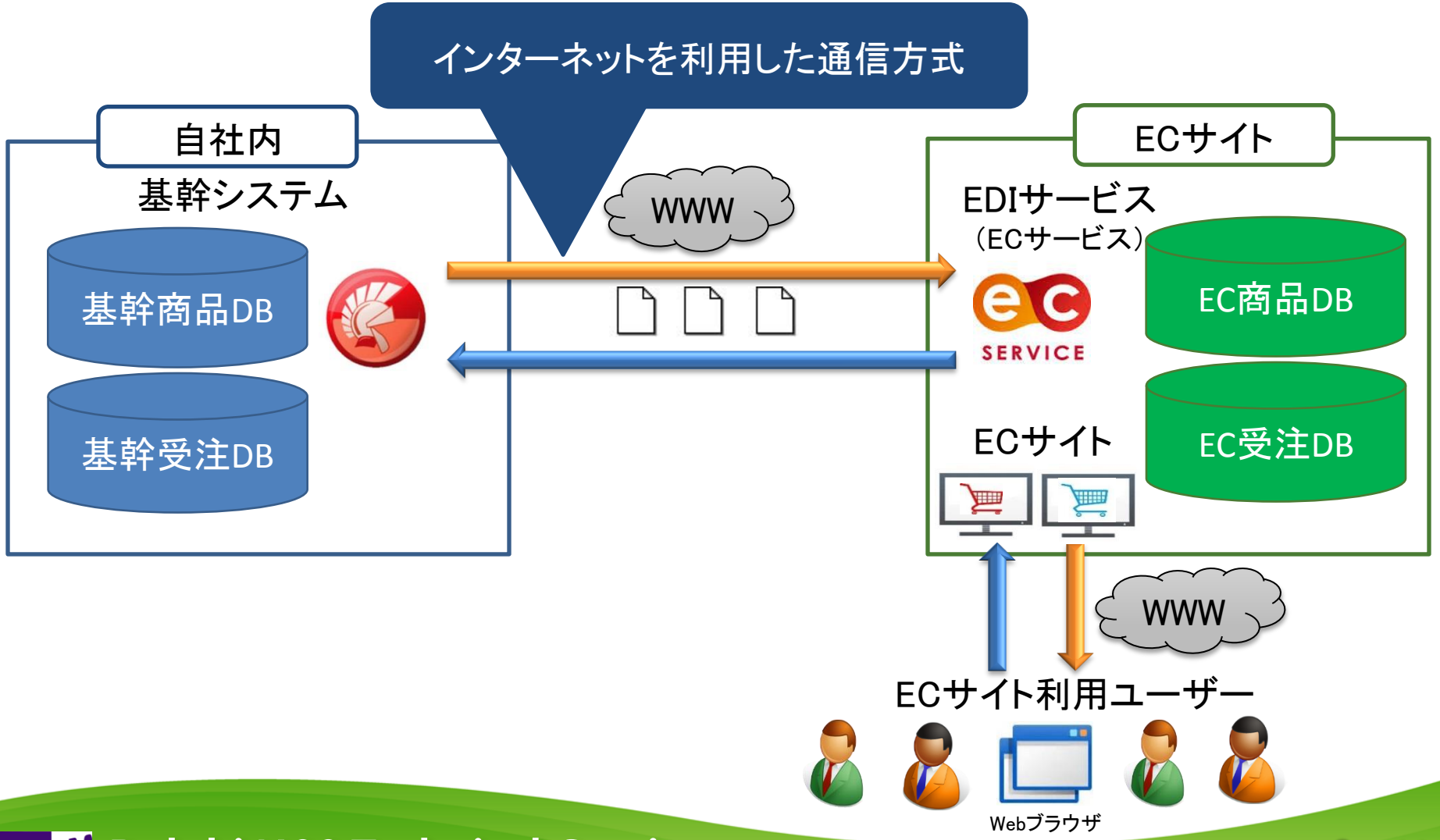
- DelphiのプログラムからAmazonや楽天などのインターネットEDIと連携してデータを送受信して活用する手法をご紹介。
  ＜仕組みの概要＞



2－1．通信方式

2－1. 通信方式

- インターネットEDIのサービスに通信する方式

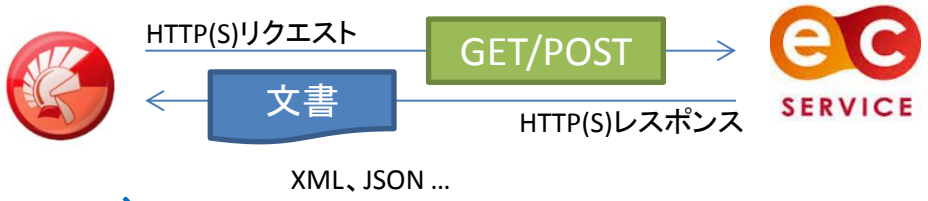


2－1. 通信方式

- インターネットを使用した主な通信方式

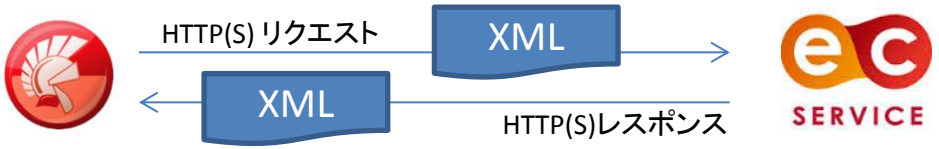
① REST（Representational State Transfer）

- GETメソッドやPOSTメソッドを使用し、HTTP（HTTPS）リクエストを特定のURLに送信することで、メッセージを受け取る。
- レスポンスの形式は指定されておらず、XMLだけでなく、JSONなど様々なデータ形式で通信を行う。



② SOAP（Simple Object Access Protocol）

- プログラム同士がSOAPメッセージというXMLによって、メッセージの交換を行う。
- HTTP（S）リクエスト、レスポンスともにXMLフォーマットのデータで通信を行う。

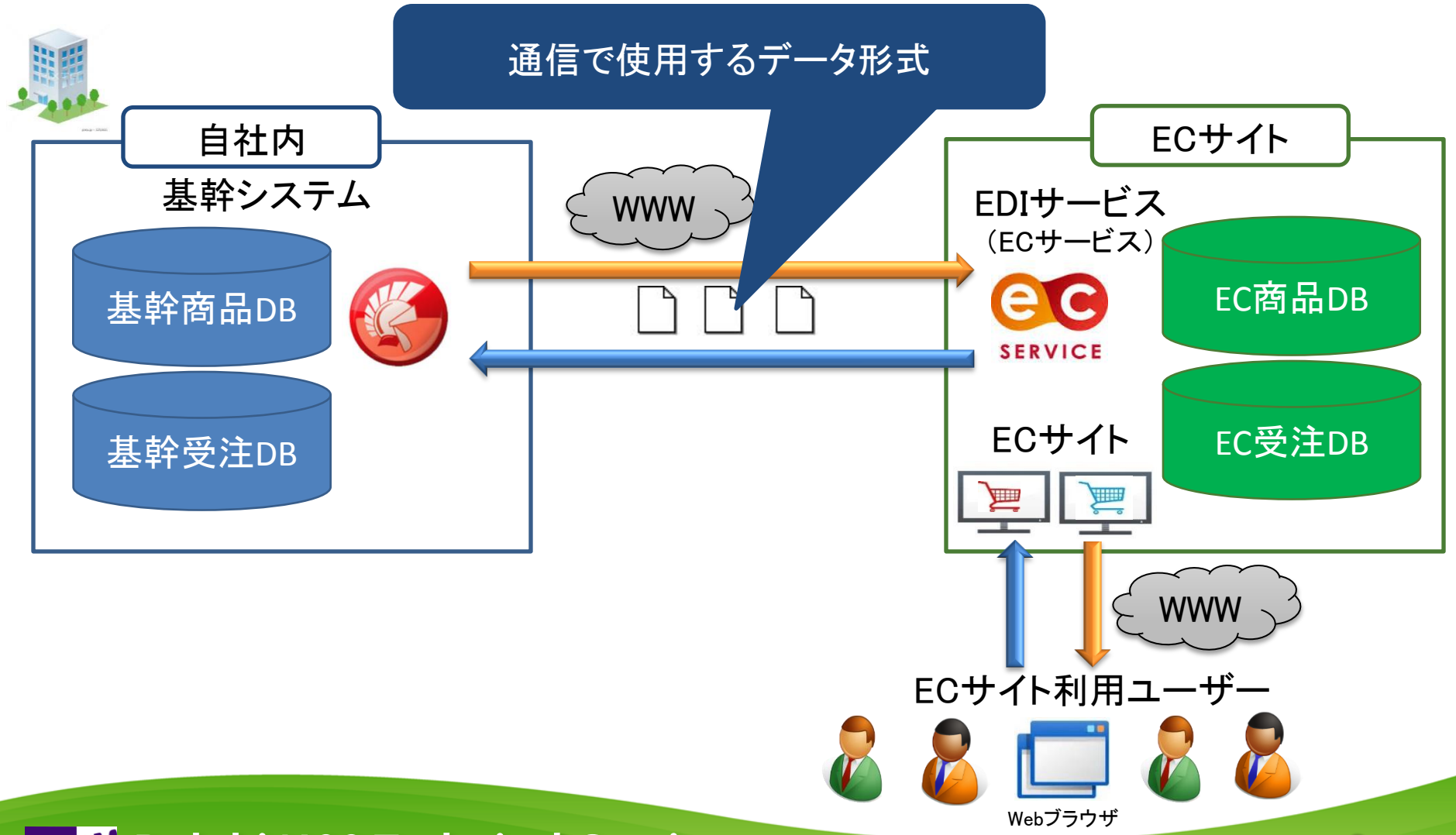


近年のインターネットEDIにおいては、RESTが主流になっています

2－2. 通信データ形式

2－2. 通信データ形式

- インターネットEDIのサービスと通信で使用するデータ形式

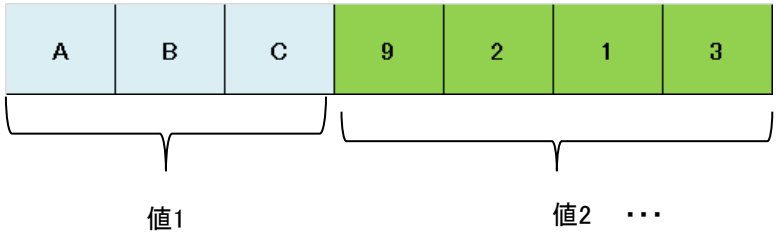


2－2. 通信データ形式

- インターネットEDIサービスとの通信において、
  これまではサービス毎に固有仕様が多かった形式も
  最近ではJSONやXMLといった標準化された形式が主流。

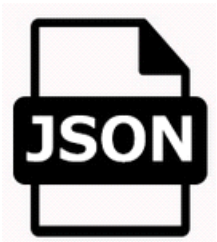
これまでの傾向

フォーマットがバラバラ
（固定長などもあり）



最近の傾向

標準化されたデータ形式
（JSONやXML）



2－2. 通信データ形式

JSON（JavaScript Object Notation）とは

- JavaScriptにおけるオブジェクト表記法を応用したデータ形式
- JavaScriptから扱いやすいため、Webシステム/サービスで主流となっている

例）

```
{
 "name": "ミガロ. ",
 "url": "http://www.migaro.co.jp/"
}
```

XML（eXtensible Markup Language）とは

- HTMLに代表されるマークアップ言語のデータ形式
- データ定義がシンプルなため汎用性が高く、簡易DBのような形式で扱えるため、Webに限らずデータ連携で採用されることが多い

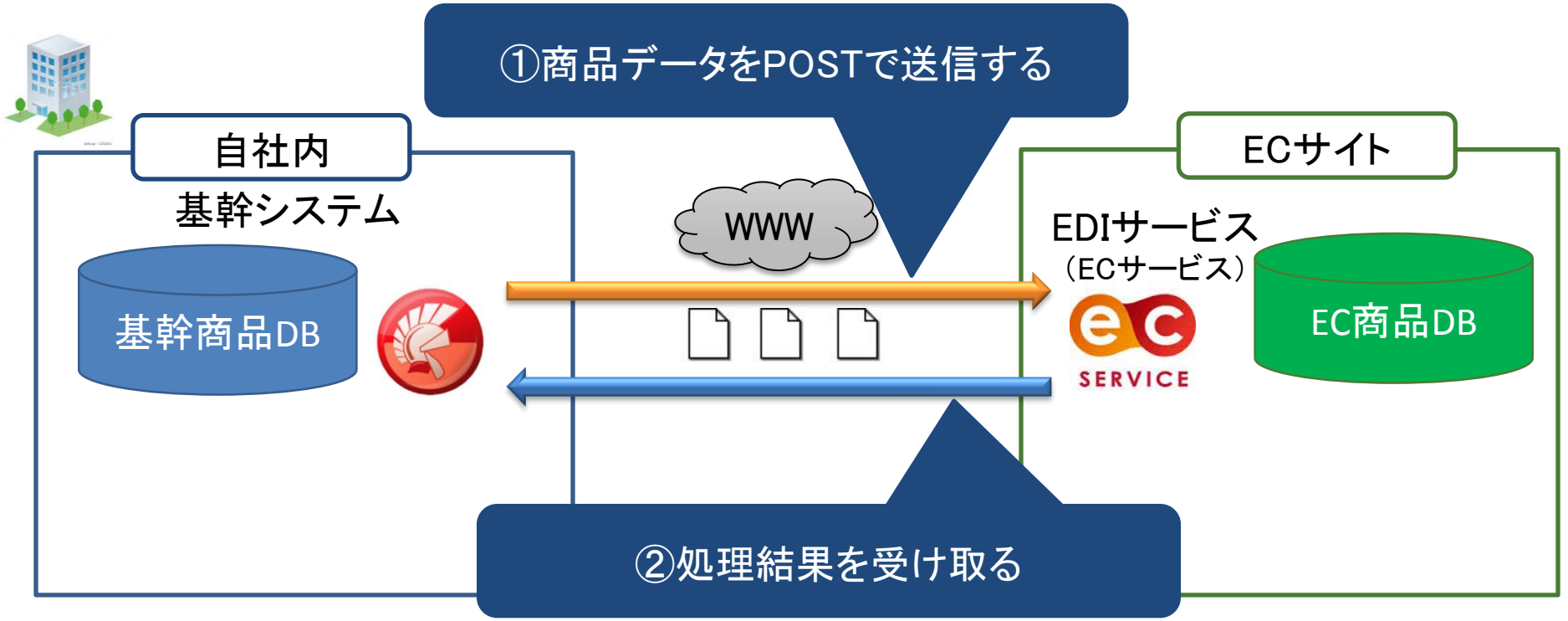
例）

```
<company>
 <name>ミガロ. </name>
 <url>http://www.migaro.co.jp/</url>
</company>
```

2－3．EDIデータの送信

2－3．EDIデータの送信

- 社内基幹システムからEDI商品データを送信（アップロード）する
  ＜処理概要＞



2－3. EDIデータの送信

- 社内基幹システムからEDI商品データを送信（アップロード）する

基幹システム

ECサイト

ECサイト上の商品データに反映

EC SERVICE

EC商品DB

EDI商品データを送信（アップロード）

2－3．EDIデータの送信

- 送信する商品データ(JSON形式例)

```
{
  "StandardBusinessDocument": {
    "message": {
      "listOfCatalogueItem": {
        "itemCode": {
          "buyerAssignedTradeItemIdentification": "N7001",
        },
        "itemName": {
          "tradeItemLongDescription": "ホッチキス",
          "tradeItemShortDescription": "サクリフラット",
        },
        "priceInfo": {
          "itemSellingUnitPrice": "9999",
          "itemNetUnitPrice": "0",
        },
        "tradingInfo": {
          "sellingStock": "500",
        }
      }
    }
  }
}
```

＜商品データ＞
・商品コード
・商品名
・商品略称
・単価
・原価
・在庫数

2－3. EDIデータの送信

①商品データをPOSTで送信する

JSONデータ作成処理

```
uses Data.DBXJSON;
```

JSONを取り扱うクラスを使用するためのユニットを追加

```
procedure TForm1. btnPOSTClick(Sender: TObject);
var
  idHTTP1 : TIdHTTP;
  PostStream : TStringList;
  ResStream  : TStringStream;

  joJSON: TJSONObject;
  joStandardBusinessDocument: TJSONObject;
  joMessage: TJSONObject;
  joListOfCatalogueItem: TJSONObject;
  joItemCode: TJSONObject;
  joItemName: TJSONObject;
  joPriceInfo: TJSONObject;
  joTradingInfo: TJSONObject;
```

HTTPSリクエストの送信にはTIdHTTPコンポーネントを使用

POSTメソッド実行時の引数用変数

JSONデータ作成に使用する
TJSONObjectクラスの変数

2－3. EDIデータの送信

①商品データをPOSTで送信する

JSONデータ作成処理

```
idHTTP1 := TIdHTTP.Create;
PostStream := TStringList.Create;
ResStream  := TStringStream.Create;
try
  idHTTP1.IOHandler := TIdSSLIOHandlerSocketOpenSSL.Create;

  SQLQuery1.SQL.Text := 'SELECT * FROM SHHNMAS WHERE SHRKFG = ''''';
  SQLQuery1.Open;
  while not SQLQuery1.Eof do
  begin
    joJSON                    := TJSONObject.Create;
    joStandardBusinessDocument := TJSONObject.Create;
    joMessage                 := TJSONObject.Create;
    joListOfCatalogueItem     := TJSONObject.Create;
    joItemCode                := TJSONObject.Create;
    joItemName                := TJSONObject.Create;
    joPriceInfo               := TJSONObject.Create;
    joTradingInfo             := TJSONObject.Create;
```

SSLを使用する為、TIdSSLIOHandlerSocketOpenSSLを使用
※ssleay32.dllとlibeay32.dllを「C:¥Windows¥System32」に配置
→詳細はembarcaderoのオンラインヘルプ参照
「http://docwiki.embarcadero.com/RADStudio/XE7/ja/Indy_ネットワーク接続のセキュリティ保護」

DBより未連携のデータを取得

JSONデータ作成用オブジェクトを生成

2－3. EDIデータの送信

①商品データをPOSTで送信する

JSONデータ作成処理

```
try
  joItemCode.AddPair('buyerAssignedTradeItemIdentification',
                     SQLQuery1.FieldByName('SHHNCD').AsString);
  joItemName.AddPair('tradeItemLongDescription',
                     SQLQuery1.FieldByName('SHHNNM').AsString);
  joItemName.AddPair('tradeItemShortDescription',
                     SQLQuery1.FieldByName('SHHNRK').AsString);
  joPriceInfo.AddPair('itemSellingUnitPrice',
                     SQLQuery1.FieldByName('SHTANK').AsString);
  joPriceInfo.AddPair('itemNetUnitPrice',
                     SQLQuery1.FieldByName('SHGENK').AsString);
  joTradingInfo.AddPair('sellingStock',
                        cdsSHOHIN.FieldByName('SHZKSU').AsString);

  joListOfCatalogueItem.AddPair('itemCode', joItemCode);
  joListOfCatalogueItem.AddPair('itemName', joItemName);
  joListOfCatalogueItem.AddPair('priceInfo', joPriceInfo);
  joListOfCatalogueItem.AddPair('tradingInfo', joTradingInfo);
```

各項目のJSONデータを作成

商品CD

商品名

商品略称

単価

原価

在庫数

各項目をカテゴリ毎に集約
itemCode ： 商品CD
itemName ： 商品名、
　　商品略称
priceInfo ： 単価、原価
tradingInfo ： 在庫数

2－3．EDIデータの送信

①商品データをPOSTで送信する

ECサイトへのPOST処理

```
joMessage.AddPair('listOfCatalogueItem', joListOfCatalogueItem);
joStandardBusinessDocument.AddPair('message', joMessage);
joJSON.AddPair('StandardBusinessDocument', joStandardBusinessDocument);

idHTTP1.Request.Clear;
PostStream.Clear;
ResStream.Clear;
PostStream.Add(StrToHex(joJSON.ToString));

idHTTP1.Post('https://XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX',
             PostStream, ResStream);

if ResStream.DataString = '1' then
begin
  SQLQuery2.SQL.Text := 'UPDATE SHHNMAS SET SHRKFG = ''1''' +
                        ' WHERE SHSHCD = :SHCD';
  SQLQuery2.ParamByName('SHCD').AsString :=
    SQLQuery1.FieldByName('SHSHCD').AsString;
  SQLQuery2.ExecSQL;
```

送信するJSON形式の構造に合わせ、データを作成

JSON形式のデータを引数用の変数へセット
※全角文字を16進数に変換

POSTメソッドにてJSONデータを送信
※第一引数にはAPIのURLを指定

ECサイトAPIのデータを正常更新した場合
※今回は正常更新時、'1'が返却

DBのデータを連携済に更新

「②処理結果を受け取る」

2－3．EDIデータの送信

①商品データをPOSTで送信する

ECサイトへのPOST処理

```
        end;
      finally
        FreeAndNil(joJSON);
      end;

      SQLQuery1.Next;
    end;
 finally
    SQLQuery1.Close;
    FreeAndNil(PostStream);
    FreeAndNil(ResStream);
  end;
end;
```

2－3. EDIデータの送信（XML版実装）

補足

- 送信する商品データ(XML形式例)

```
<BusinessDocument>
  <message>
    <listOfCatalogueItems>
      <itemCode>
        <supplierItemCode>N7001</supplierItemCode>
      </itemCode>
      <itemName>
        <tradeItemLongDescription>ホッチキス</tradeItemLongDescription>
        <tradeItemShortDescription>サクリフラット</tradeItemShortDescription>
      </itemName>
      <priceInfo>
        <itemSellingUnitPrice>
          <value>9999</value>
        </itemSellingUnitPrice>
        <itemNetUnitPrice>
          <value>0</value>
        </itemNetUnitPrice>
      </priceInfo>
      <tradingInfo>
        <SellingStock>500</SellingStock>
      </tradingInfo>
    </listOfCatalogueItems>
  </message>
</BusinessDocument>
```

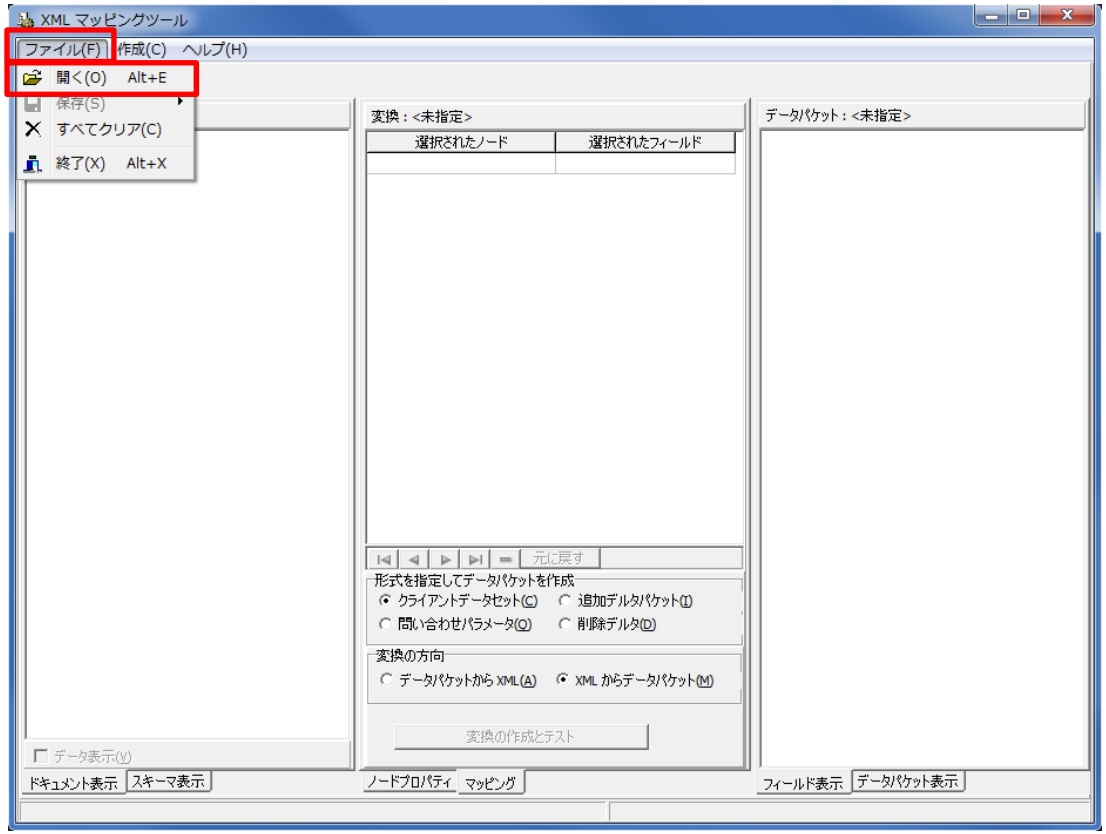
＜商品データ＞
・商品コード
・商品名
・商品略称
・単価
・原価
・在庫数

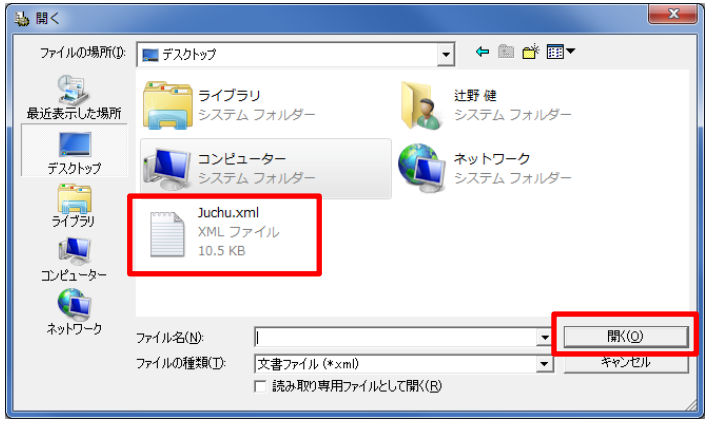
2－3. EDIデータの送信（XML版実装）

補足

XMLクラスの作成手順 I

- Delphiに付属している「xmlmapper.exe」を利用して、XMLスキーマを作成



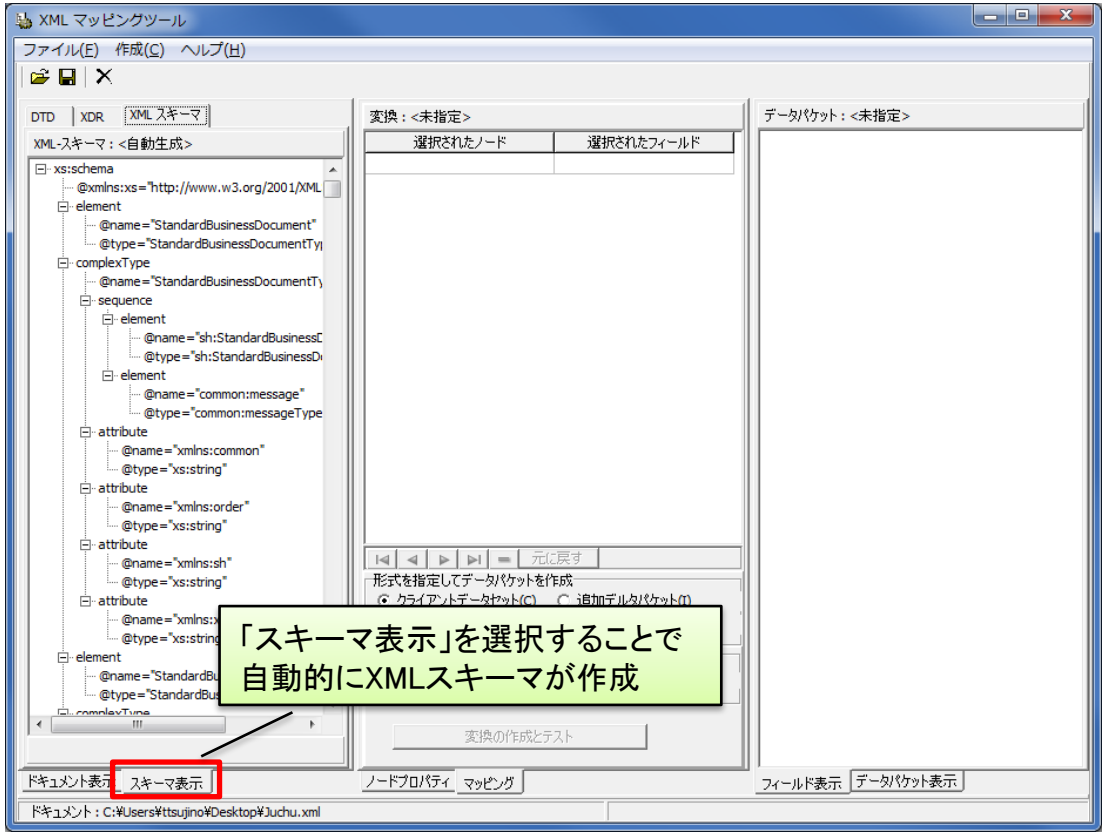


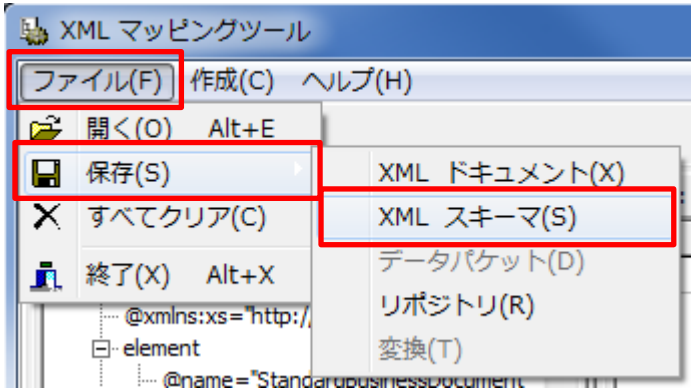
2－3. EDIデータの送信（XML版実装）

補足

XMLクラスの作成手順 II

- Delphiに付属している「xmlmapper.exe」を利用して、XMLスキーマを作成



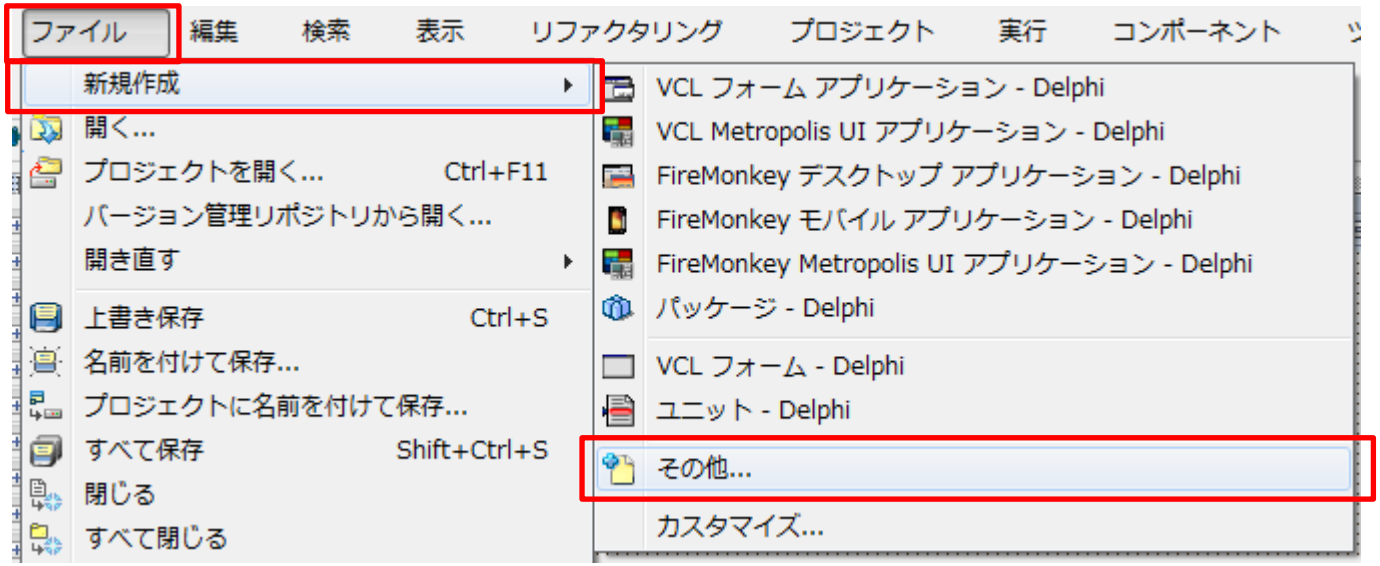


2－3. EDIデータの送信（XML版実装）

補足

XMLクラスの作成手順Ⅲ

- XMLスキーマを利用して、XMLのクラスを作成
  - ➢ [ファイル｜新規作成｜その他]を選択

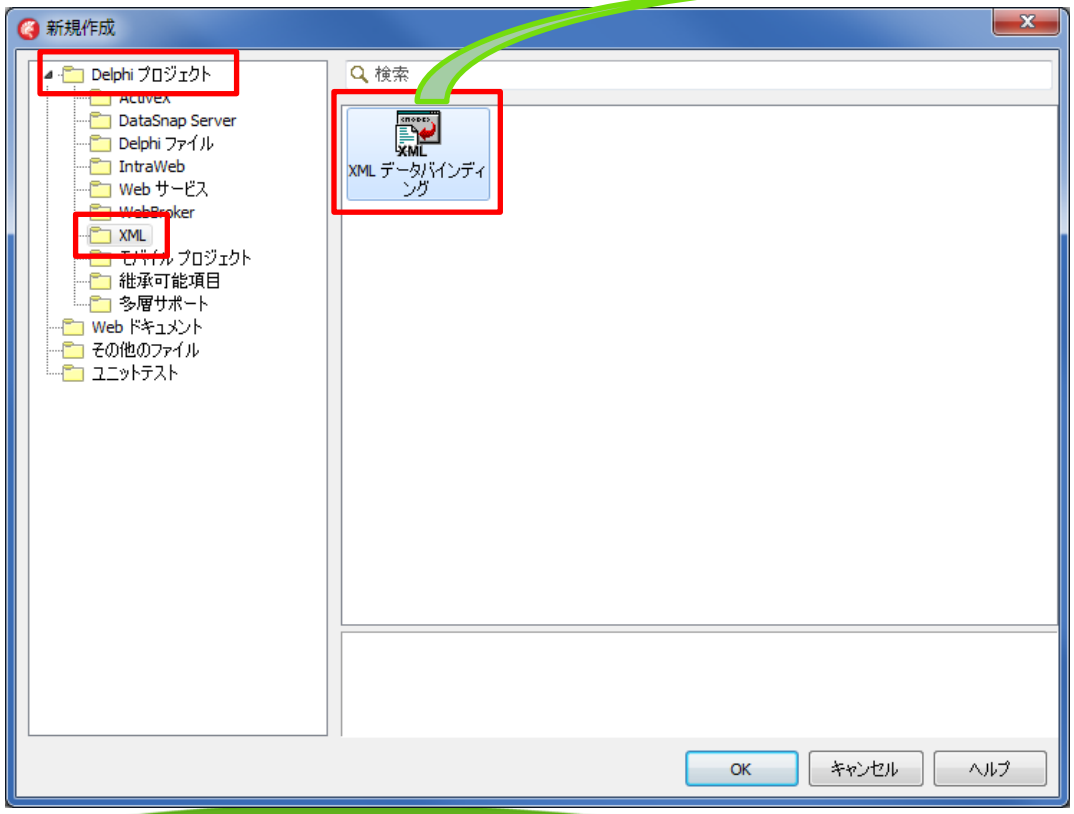


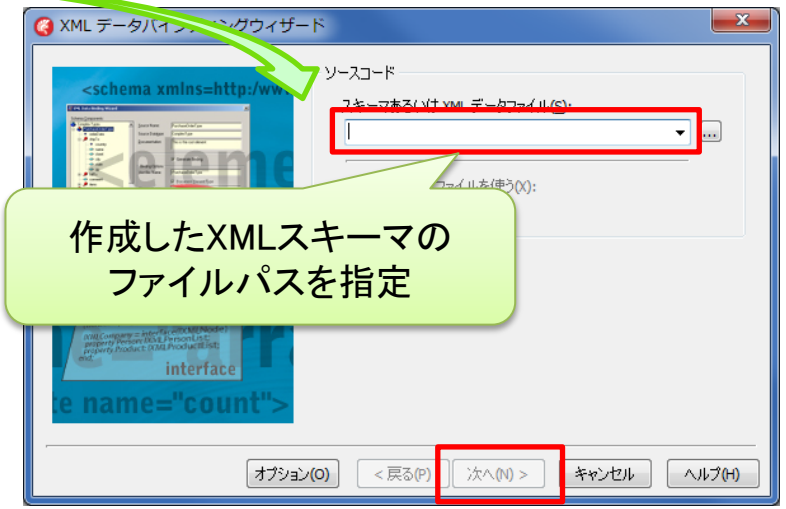
2－3. EDIデータの送信（XML版実装）

補足

XMLクラスの作成手順Ⅳ

- XMLスキーマを利用して、XMLのクラスを作成
  - ➢ [Delphiプロジェクト｜XML｜XMLデータバインディング]



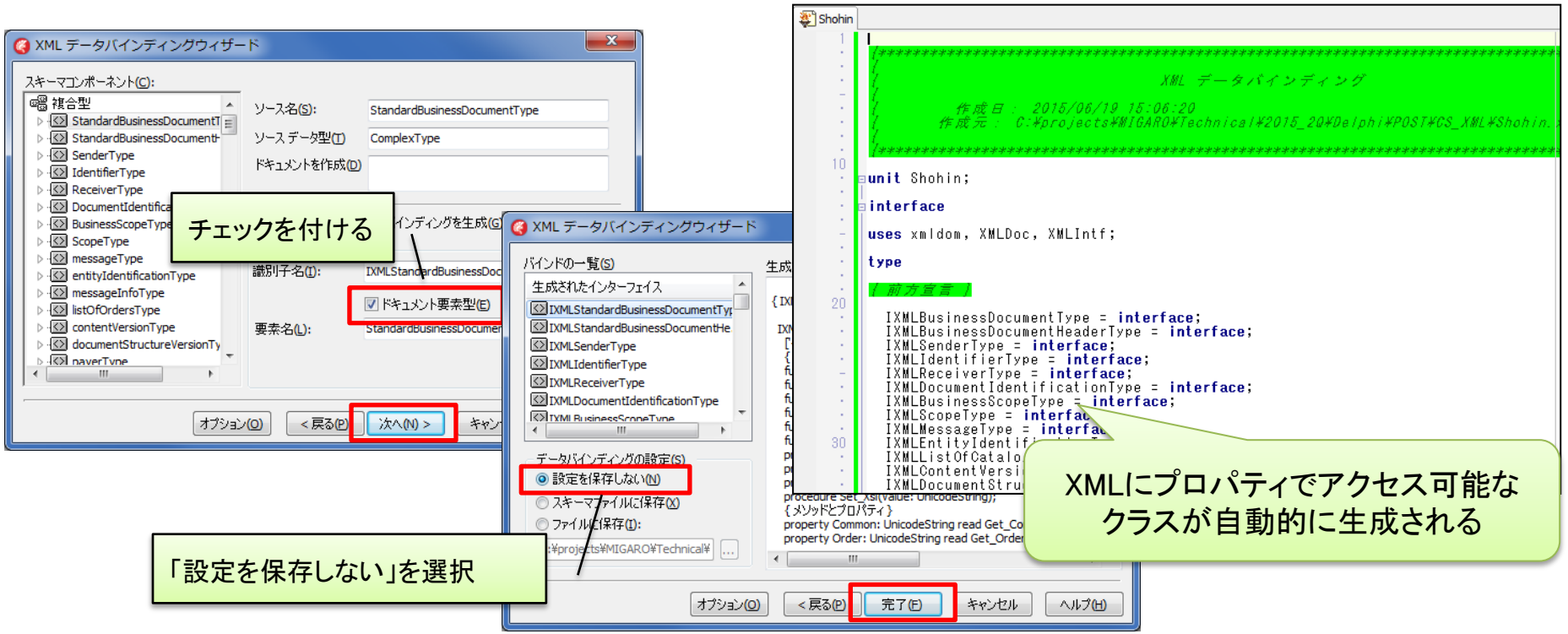


2－3. EDIデータの送信（XML版実装）

補足

XMLクラスの作成手順Ⅴ

- XMLスキーマを利用して、XMLのクラスを作成
  - ➢ XMLデータバインディングウィザードを完了まで進める



2－3. EDIデータの送信（XML版実装）

補足

①商品データをPOSTで送信する

XMLデータ作成処理

先程作成したXML専用クラスをuses節に追加

```
uses Shohin;
```

```
procedure TForm1.btnPOSTClick(Sender: TObject);
var
  idHTTP1 : TIdHTTP;
  PostStream : TStringList;
  ResStream  : TStringStream;

  xsbXMLBusinessDocument: IXMLBusinessDocumentType;
begin
  idHTTP     := TIdHTTP.Create;
  PostStream := TStringList.Create;
  ResStream  := TStringStream.Create;
  try
    idHTTP1.IOHandler := TIdSSLIOHandlerSocketOpenSSL.Create;

    SQLQuery1.SQL.Text := 'SELECT * FROM SHHNMAS WHERE SHRKFG = '''';
    SQLQuery1.Open;
```

POSTメソッド実行時の引数用変数

XMLを扱うための変数を定義

DBより未連携のデータを取得

2－3. EDIデータの送信（XML版実装）

補足

①商品データをPOSTで送信する

XMLデータ作成処理

```
while not SQLQuery1.Eof do
begin
  xsbXMLBusinessDocument := NewBusinessDocument;

  xsbXMLBusinessDocument.Message.ListOfCatalogueItems.ItemCode.SupplierItemCode :=
    SQLQuery1.FieldByName('SHSHCD').AsString;
  xsbXMLBusinessDocument.Message.ListOfCatalogueItems.ItemName.TradeItemLongDescription :=
    SQLQuery1.FieldByName('SHSHNM').AsString;
  xsbXMLBusinessDocument.Message.ListOfCatalogueItems.ItemName.TradeItemShortDescription :=
    SQLQuery1.FieldByName('SHSHRK').AsString;
  xsbXMLBusinessDocument.Message.ListOfCatalogueItems.PriceInfo.ItemSellingUnitPrice.Value :=
    SQLQuery1.FieldByName('SHTANK').AsString;
  xsbXMLBusinessDocument.Message.ListOfCatalogueItems.PriceInfo.ItemNetUnitPrice.Value :=
    SQLQuery1.FieldByName('SHGENK').AsString;
  xsbXMLBusinessDocument.Message.ListOfCatalogueItems.TradingInfo.sellingStock :=
    SQLQuery1.FieldByName('SHZKSU').AsString;
```

タグのみのXMLを作成

XMLにデータを追加

商品CD

商品名

商品略称

単価

原価

在庫数

2－3. EDIデータの送信（XML版実装）

補足

①商品データをPOSTで送信する

ECサイトへのPOST処理

```
    idHTTP1.Request.Clear;
    PostStream.Clear;
    ResStream.Clear;
    PostStream.Add(StrToHex(xsbXMLBusinessDocument.XML));

    idHTTP1.Post('https://XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX',
                 PostStream, ResStream);

    if ResStream.DataString = '1' then
    begin
      SQLQuery2.SQL.Text := 'UPDATE SHHNMAS SET SHRKFG = ''1''' +
                            ' WHERE SHSHCD = :SHCD';
      SQLQuery2.ParamByName('SHCD').AsString :=
        SQLQuery1.FieldByName('SHSHCD').AsString;
      SQLQuery2.ExecSQL;
    end;
  finally
    FreeAndNil(joJSON);
  end;
```

XML形式のデータを引数用の変数へセット
※全角文字を16進数に変換

POSTメソッドにてXMLデータを送信
※第一引数にはAPIのURLを指定

ECサイトAPIのデータを正常更新した場合
※今回は正常更新時、'1'が返却

DBのデータを連携済に更新

「②処理結果を受け取る」

2－3. EDIデータの送信（XML版実装）

補足

①商品データをPOSTで送信する

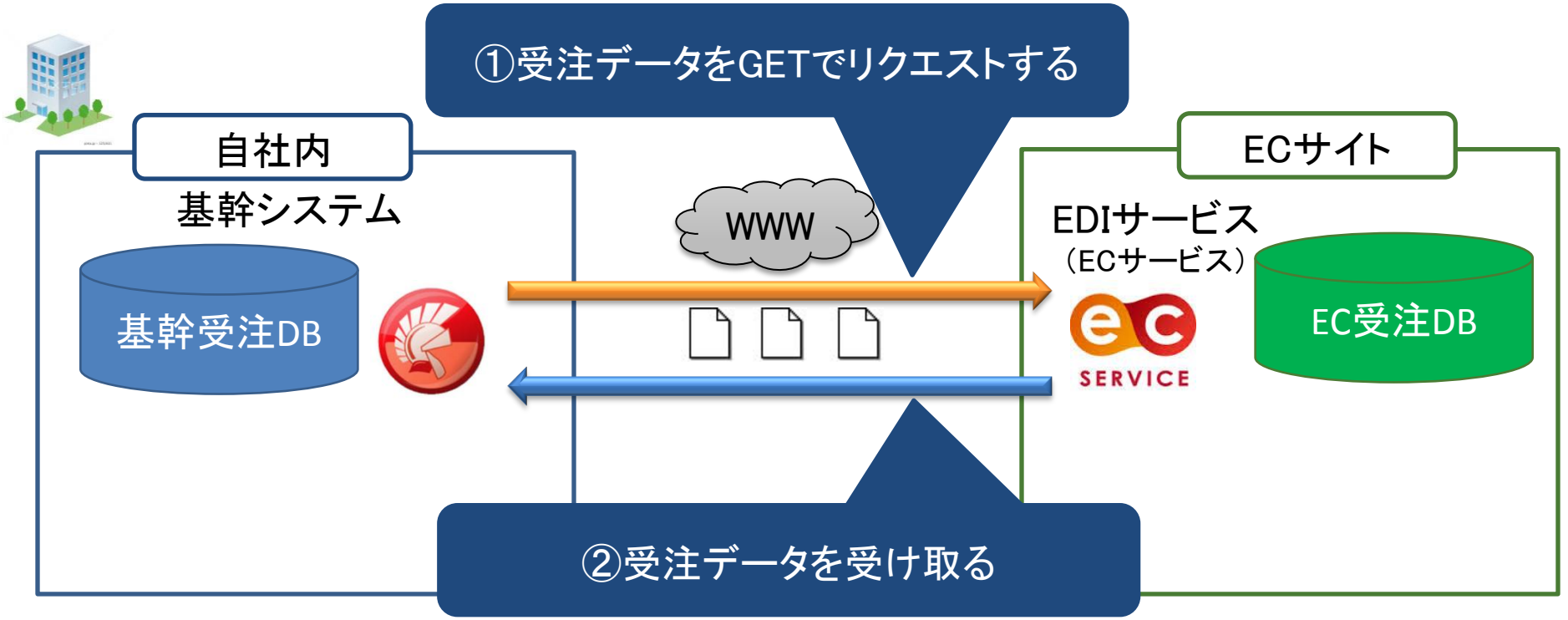
ECサイトへのPOST処理

```
      SQLQuery1.Next;
    end;
  finally
    SQLQuery1.Close;
    FreeAndNil(PostStream);
    FreeAndNil(ResStream);
  end;
end;
```

2－4. EDIデータの受信

2－4. EDIデータの受信

- 社内基幹システムにEDI受注データを受信（ダウンロード）する
  ＜処理概要＞

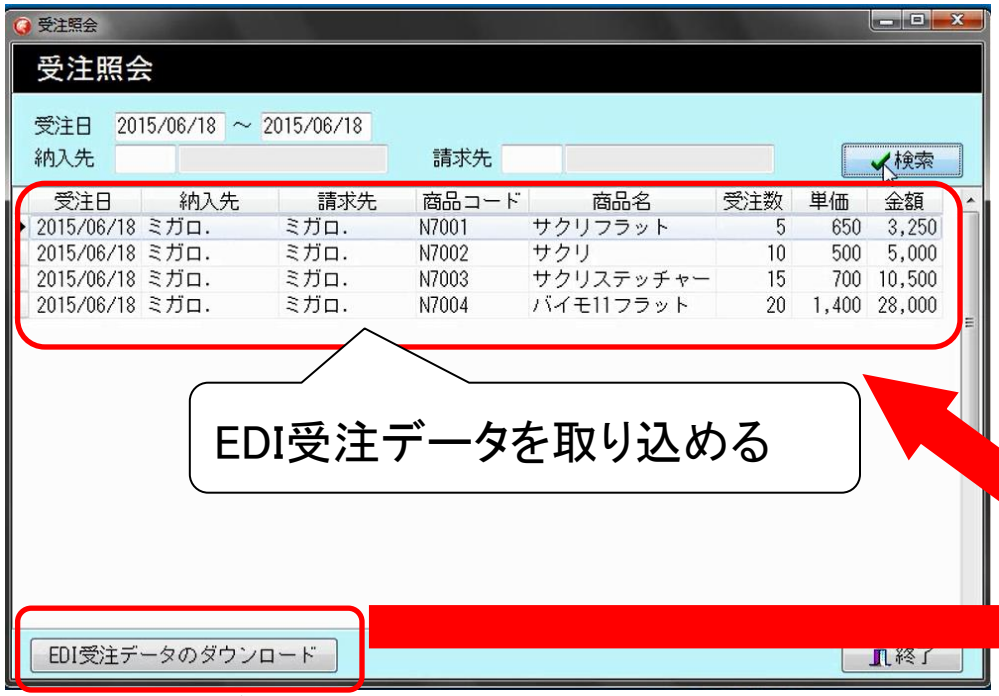


2－4. EDIデータの受信

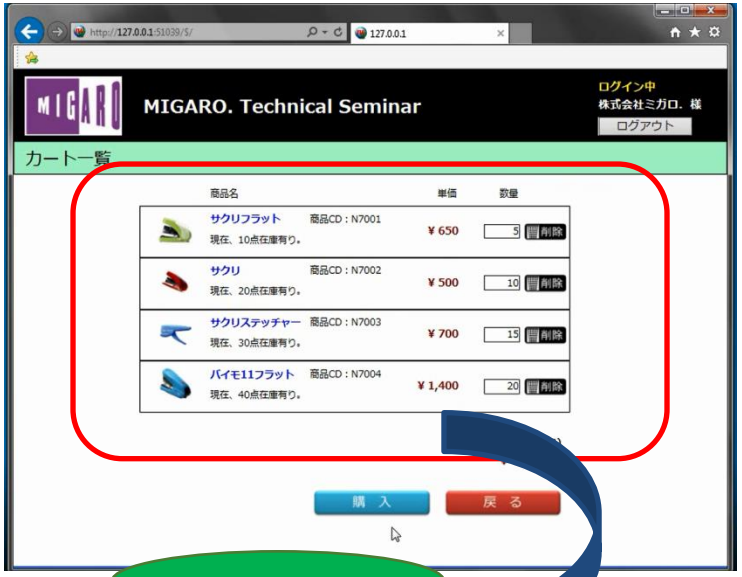
- 社内基幹システムにEDI受注データを受信（ダウンロード）する

基幹システム

ECサイト



EDI受注データを取り込める



EC SERVICE

EC受注DB

注文

EDI受注データをダウンロード

ECサイトで注文されている受注データ

2－4. EDIデータの受信

- 受信する受注データ(JSON形式例)

```
{
  "StandardBusinessDocument":{
    "DocumentBody":{
      "ListOfOrders":{
        "Order":{
          "OrderDate":"20150618",
          "ShipToCode":"00356",
          "PayeeCode":"00356",
          "LineItems":[
            {
              "ItemID":"N7001",
              "UnitPrice":"650",
              "Quantity":"3"
            },
            {
              "ItemID":"N7002",
              "UnitPrice":"500",
              "Quantity":"5"
            }
          ]
        }
      }
    }
  }
}
```

＜受注データ＞
・受注日
・納入先コード
・請求先コード
・商品コード
・単価
・受注数

2－4. EDIデータの受信

①受注データをGETでリクエストする

JSONデータ加工処理

```
uses Data.DBXJSON;
```

JSONを取り扱うクラスを使用するためのユニットを追加

```
procedure TForm1.btnGetClick(Sender: TObject);
var
  idHTTP: TIdHTTP;
  sData: String;
  iIndex: Integer;
  joJSON: TJSONObject;
  joStandardBusinessDocument: TJSONObject;
  joDocumentBody: TJSONObject;
  joListOfOrders: TJSONObject;
  joOrder: TJSONObject;
  jaLineItems: TJSONArray;
  joLineItem: TJSONObject;
begin
 idHTTP := TIdHTTP.Create;
  try
    idHTTP.IOHandler := TIdSSLIOHandlerSocketOpenSSL.Create;

    sData := IdHTTP.Get('https:// XXXXXXXXXXXXXXXXXXXXXXXXXXXXX');
```

JSONを扱うための変数を定義

SSLを使用

受注データ(JSON)を取得

2－4. EDIデータの受信

②受注データを受け取る

JSONデータ加工処理

```
joJSON := TJSONObject.Create;
try
  joJSON := joJSON.ParseJSONValue(sData) as TJSONObject;
  joStandardBusinessDocument := joJSON.GetValue('StandardBusinessDocument') as TJSONObject;
  joDocumentBody := joStandardBusinessDocument.GetValue('DocumentBody') as TJSONObject;
  joListOfOrders := joDocumentBody.GetValue('ListOfOrders') as TJSONObject;
  joOrder := joListOfOrders.GetValue('Order') as TJSONObject;

  SQLQuery1.SQL.Text := 'INSERT INTO JUCHTRN (' +
                        ' JUJUNO, JUJCDT, JUSPCD, JUSKCD, JUSHCD, JUTANK, JUJUSU' +
                        ') VALUES (' +
                        ' :JUNO, :JCDT, :SPCD, :SKCD, :SHCD, :TANK, :JUSU' +
                        ')';

  SQLQuery1.ParamByName('JCDT').AsInteger :=
    StrToIntDef(joOrder.GetValue('OrderDate').Value, 0);
  SQLQuery1.ParamByName('SPCD').AsString :=
    joOrder.GetValue('ShipToCode').Value;
  SQLQuery1.ParamByName('SKCD').AsString :=
    joOrder.GetValue('PayeeCode').Value;
```

オブジェクトを生成

JSONデータの構造を分割

DB更新用SQLの設定

JSONにアクセスし、必要な情報を取得

受注日

納入先CD

請求先CD

2－4. EDIデータの受信

②受注データを受け取る

JSONデータ加工処理

```
    jaLineItems := joOrder.GetValue('LineItems') as TJSONArray;
    for iIndex := 0 to (jaLineItems.Size - 1) do
    begin
      joLineItem := jaLineItems.Get(iIndex) as TJSONObject;

      SQLQuery1.ParamByName('SHCD').AsString :=
        joLineItem.GetValue('ItemID').Value;
      SQLQuery1.ParamByName('TANK').AsInteger :=
        StrToIntDef(joLineItem.GetValue('UnitPrice').Value, 0);
      SQLQuery1.ParamByName('JUSU').AsInteger :=
        StrToIntDef(joLineItem.GetValue('Quantity').Value, 0);

      SQLQuery1.ExecSQL;
    end;
  finally
    FreeAndNil(joJSON);
  end;
finally
  FreeAndNil(idHTTP);
end;
```

受注明細情報を取得

1明細分のデータを取得

商品CD

単価

受注数

2－4. EDIデータの受信（XML版実装）

補足

- 受信する受注データ(XML形式例)

```
<StandardBusinessDocument>
  <DocumentBody>
    <listOfOrders>
      <order>
        <tradeSummary>
          <dates>
            <orderDate>20150618</orderDate>
          </dates>
        </tradeSummary>
        <parties>
          <shipTo>
            <code>00356</code>
          </shipTo>
          <payee>
            <code>00356</code>
          </payee>
        </parties>
        <lineItem>
          <itemID>
            <gtin>N7001</gtin>
          </itemID>
          <amounts>
            <itemSellingPrice unitPrice="650"/>
          </amounts>
      ～　以下省略　～
```

＜受注データ＞
・受注日
・納入先コード
・請求先コード
・商品コード
・単価
・受注数

2－4. EDIデータの受信（XML版実装）

補足

①受注データをGETでリクエストする

XMLデータ加工処理

```
uses Juchu;
```

XMLスキーマより作成した専用クラスをuses節に追加

```
procedure TForm1.btnGetClick(Sender: TObject);
var
  idHTTP: TIdHTTP;
  sData: String;
  xdcXML: TXMLDocument;
  xsbXMLStandardBusinessDocument: IXMLStandardBusinessDocumentType;
  xsbXMLLineItemTypeList: IXMLLineItemTypeList;
  xsbXMLLineItem: IXMLLineItemType;
begin
 idHTTP := TIdHTTP.Create;
  try
    idHTTP.IOHandler := TIdSSLIOHandlerSocketOpenSSL.Create;
    sData := idHTTP.Get( 'https://XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' );
```

XMLを扱うための変数を定義

受注データ(XML)を取得

2－4. EDIデータの受信（XML版実装）

補足

②受注データを受け取る

XMLデータ加工処理

```
xdcXML := TXMLDocument.Create(Self);
xdcXML.LoadFromXML(sData);
xsbXMLStandardBusinessDocument := GetStandardBusinessDocument(xdcXML);
```

取得したXMLデータを専用のクラスに変換

```
SQLQuery1.SQL.Text := 'INSERT INTO JUCHTRN (' +
                      ' JUJCDT, JUSPCD, JUSKCD, JUSHCD, JUTANK, JUJUSU' +
                      ') VALUES (' +
                      ' :JCDT, :SPCD, :SKCD, :SHCD, :TANK, :JUSU' +
                      ')';
```

DB更新用SQLの設定

```
SQLQuery1.ParamByName('JCDT').AsInteger :=
  StrToIntDef(xsbXMLStandardBusinessDocument.DocumentBody.ListOfOrders.Order.
              TradeSummary.Dates.OrderDate, 0);
SQLQuery1.ParamByName('SPCD').AsString :=
  xsbXMLStandardBusinessDocument.DocumentBody.ListOfOrders.Order.Parties.ShipTo.Code;
SQLQuery1.ParamByName('SKCD').AsString :=
  xsbXMLStandardBusinessDocument.DocumentBody.ListOfOrders.Order.Parties.Payee.Code;
```

XMLにプロパティでアクセスし、必要な情報を取得

受注日

納入先CD

請求先CD

2－4. EDIデータの受信（XML版実装）

補足

②受注データを受け取る

XMLデータ加工処理

```
    xsbXMLLineItemTypeList :=
      xsbXMLStandardBusinessDocument.DocumentBody.ListOfOrders.Order.LineItem;
    for iIndex := 0 to (xsbXMLLineItemTypeList.Count - 1) do
    begin
      xsbXMLLineItem := xsbXMLLineItemTypeList.Items[iIndex];

      SQLQuery1.ParamByName('SHCD').AsString := xsbXMLLineItem.ItemID.Gtin;
      SQLQuery1.ParamByName('TANK').AsString :=
        xsbXMLLineItem.Amounts.ItemSellingPrice.UnitPrice;
      SQLQuery1.ParamByName('JUSU').AsString :=
        xsbXMLLineItem.Quantities.OrderQuantity.Quantity;

      SQLQuery1.ExecSQL;
    end;
  finally
    FreeAndNil(xdcXML);
  end;
end;
```

受注明細情報を取得

1明細分のデータを取得

商品CD

単価

受注数

DBを更新

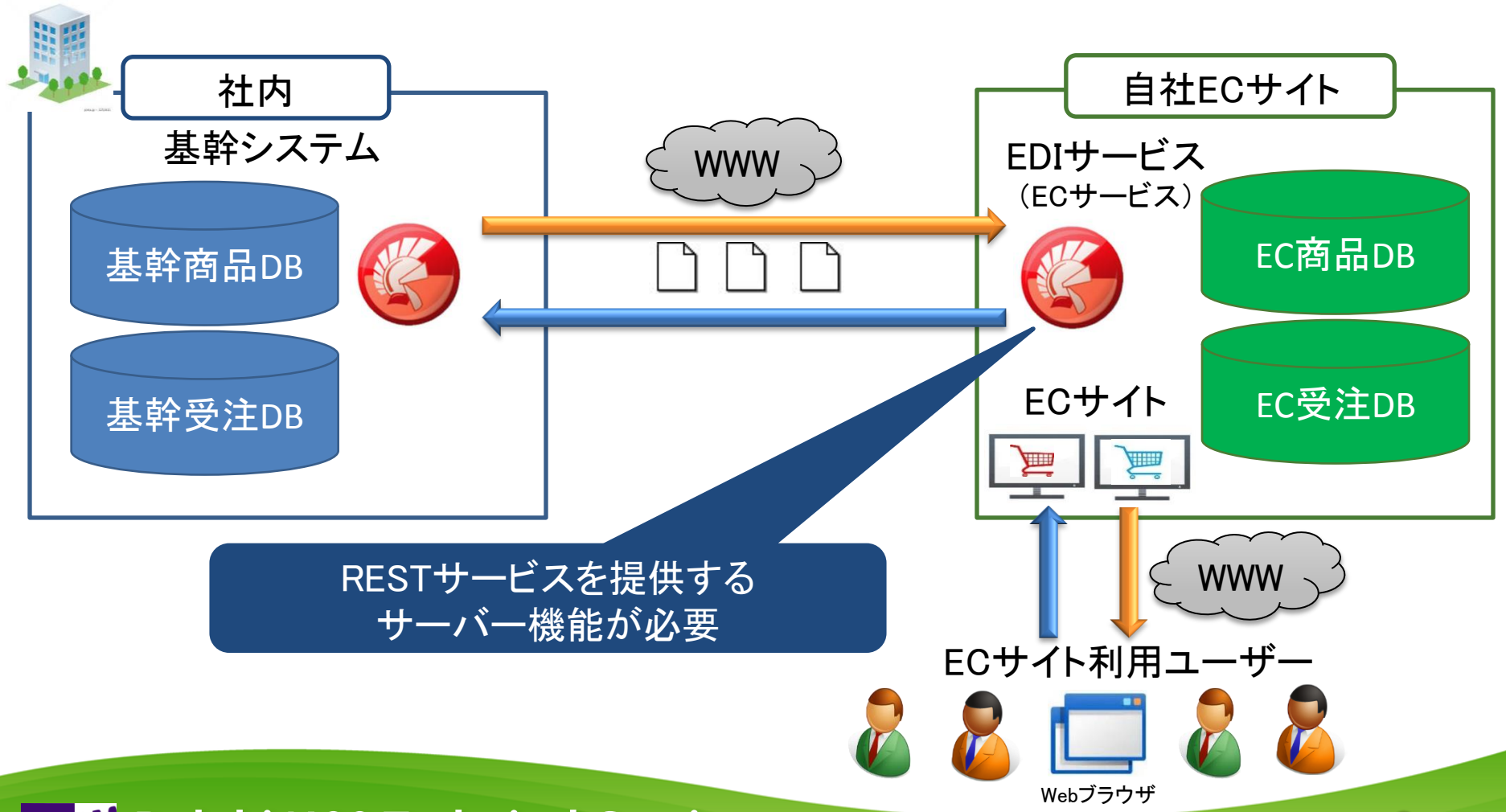
補足

2－5．RESTサーバー
アプリ作成方法

2－5. RESTサーバーアプリ作成方法

補足

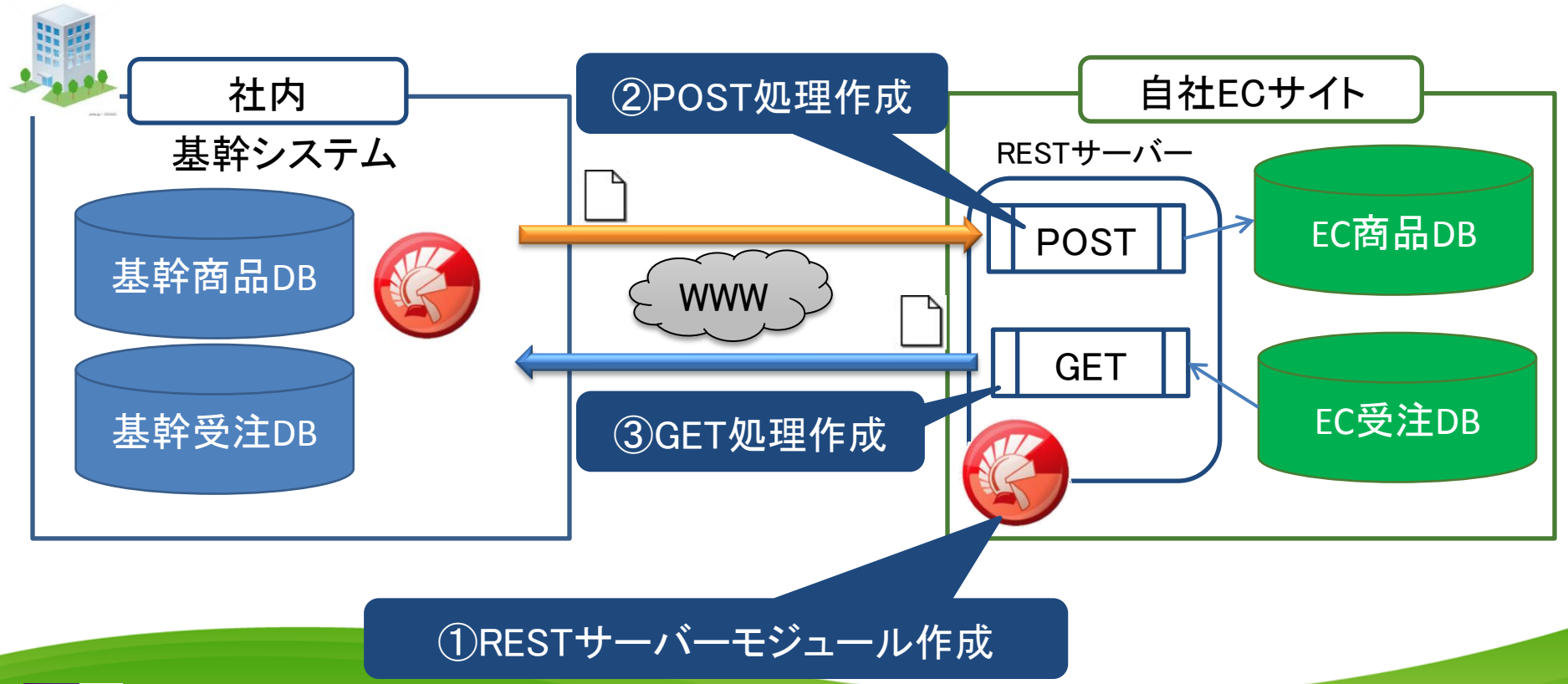
- 自社で構築したECサイトの場合、基幹システムや取引先と連携する為に、RESTサーバーアプリを構築することもできる。



2－5. RESTサーバーアプリ作成方法

補足

- 構築の手順
  - ① RESTサーバーアプリケーションモジュールの作成
  - ② POSTリクエストによるEDI商品データの受け取り
  - ③ GETリクエストに対するEDI受注データの配信

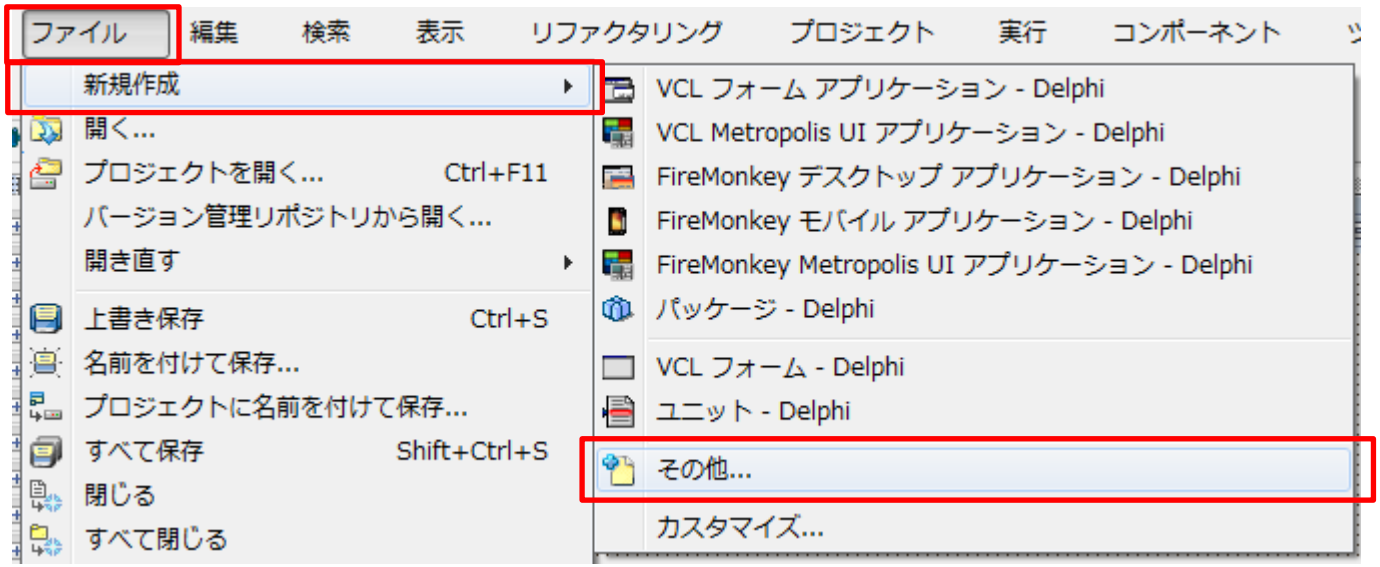


2－5. RESTサーバーアプリ作成方法

補足

① RESTサーバーアプリケーションモジュールの作成

- DataSnapServerを使用し、モジュールを作成
  - ➢ [ファイル｜新規作成｜その他]を選択

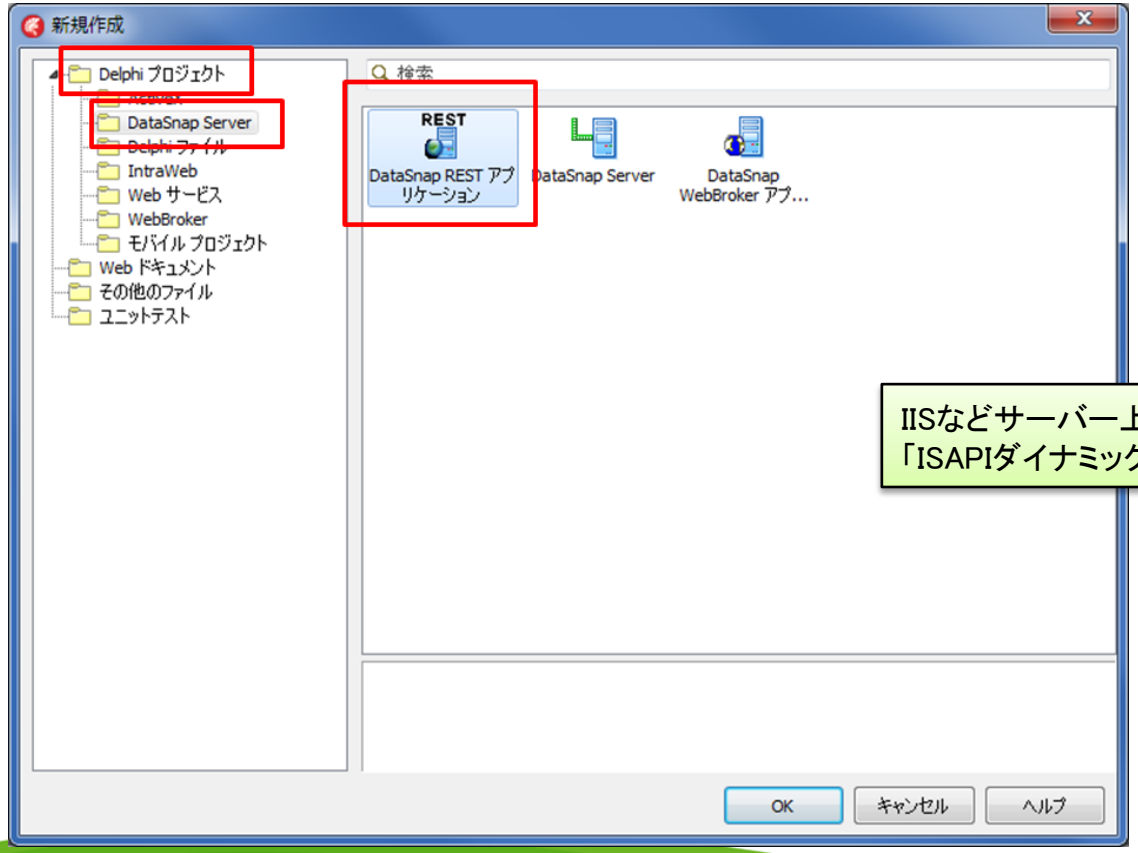


2－5. RESTサーバーアプリ作成方法

補足

① RESTサーバーアプリケーションモジュールの作成

- DataSnapServerを使用し、モジュールを作成
  - ［ファイル｜新規作成｜その他］を選択



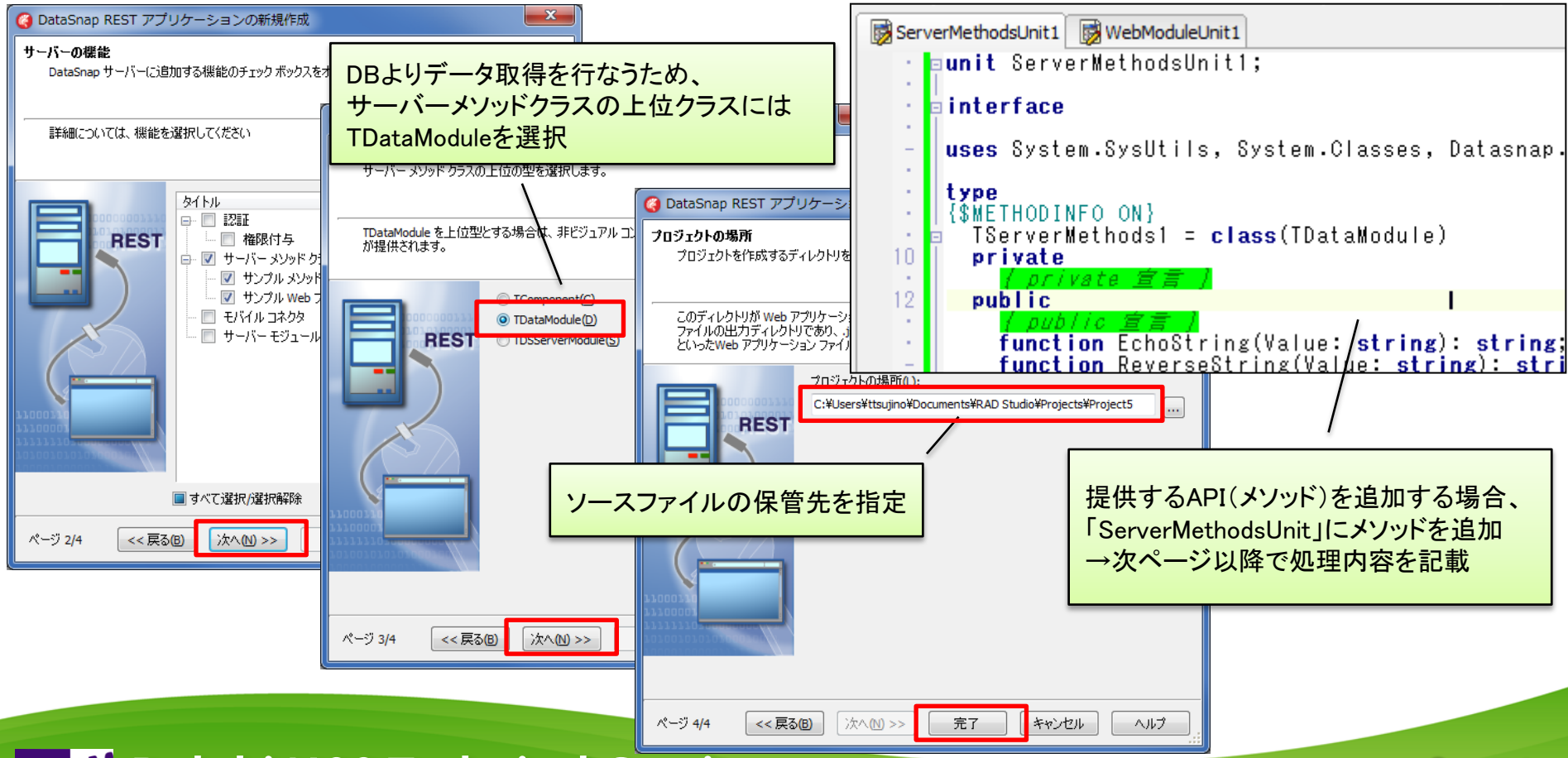
IISなどサーバー上で起動する場合は
「ISAPIダイナミックリンクライブラリ」を選択

2－5. RESTサーバーアプリ作成方法

補足

① RESTサーバーアプリケーションモジュールの作成

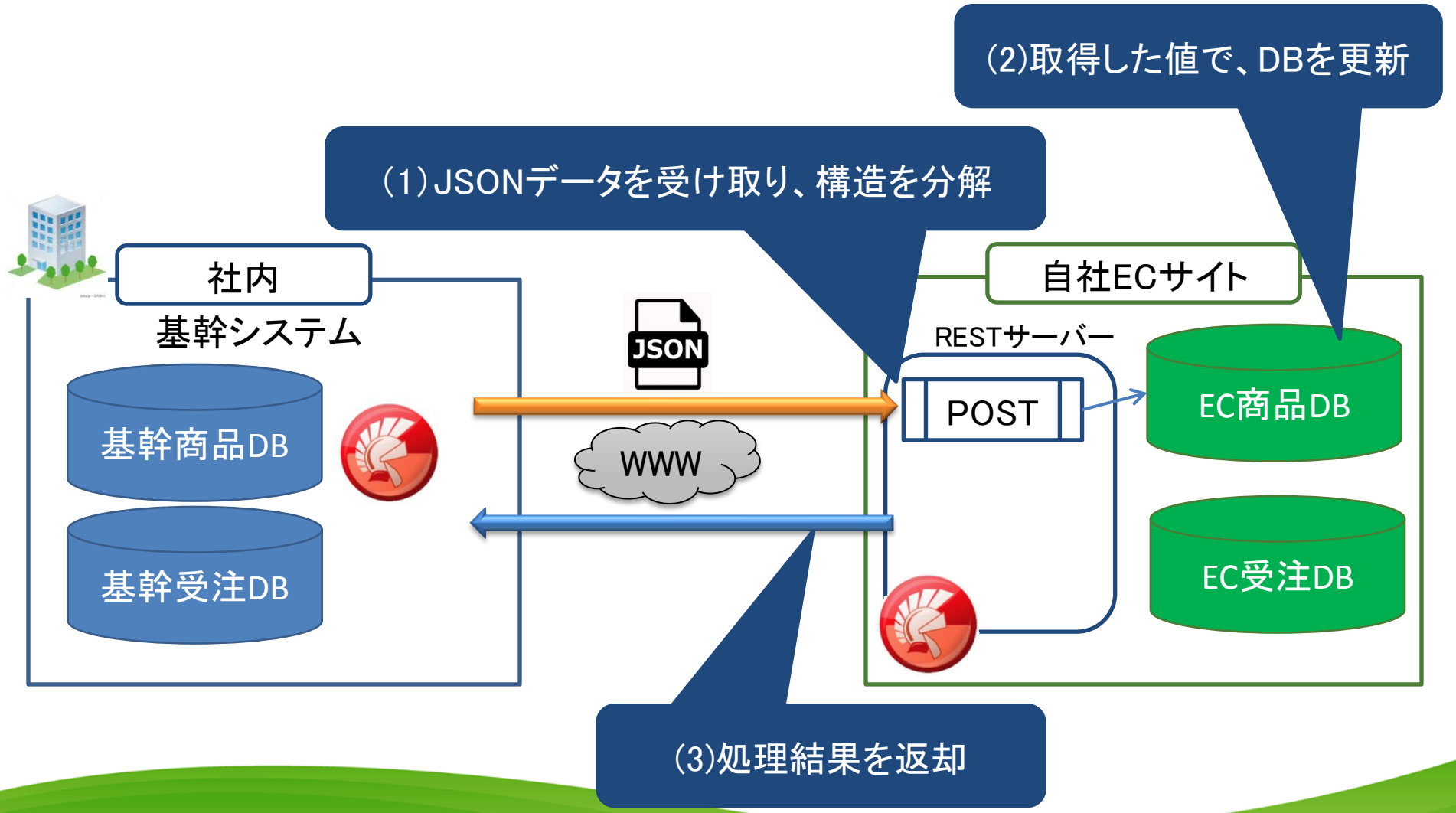
- DataSnapServerを使用し、モジュールを作成
  - ウィザードを完了まで進め、RESTアプリのモジュールを作成



2－5. RESTサーバーアプリ作成方法

補足

② POSTリクエストによるEDI商品データの受け取り



2－5. RESTサーバーアプリ作成方法

補足

② POSTリクエストによるEDI商品データの受け取り

- POSTメソッドでJSONデータとして連携する例
  - JSON形式のデータを加工し、DBを更新

ServerMethodsUnitのJSONデータよりDBを更新するメソッド

```
public
  function updateData(Value: TJSONObject): String;
```

POST用のメソッドを作成する場合には
必ず関数名を「update＋(任意名)」にする

```
uses Data.DBXJSON;
```

uses節にJSONを扱うためのユニットを追加

```
function TServerMethods1.updateData(Value: TJSONObject): string;
var
  joJSON: TJSONObject;
  joStandardBusinessDocument: TJSONObject;
  joMessage: TJSONObject;
  joListOfCatalogueItem: TJSONObject;
  joItemCode: TJSONObject;
  joItemName: TJSONObject;
  joPriceInfo: TJSONObject;
  joTradingInfo: TJSONObject;
begin
```

JSONの解析を行なうための変数を定義

2－5. RESTサーバーアプリ作成方法

補足

② POSTリクエストによるEDI商品データの受け取り

- POSTメソッドでJSONデータとして連携する例
  - JSON形式のデータを加工し、DBを更新

ServerMethodsUnitのJSONデータよりDBを更新するメソッド

JSONデータの構造を分割

```
joJSON := Value;
joStandardBusinessDocument := joJSON.GetValue('StandardBusinessDocument') as TJSONObject;
joMessage := joStandardBusinessDocument.GetValue('message') as TJSONObject;
joListOfCatalogueItem := joMessage.GetValue('listOfCatalogueItem') as TJSONObject;
joItemCode := joListOfCatalogueItem.GetValue('itemCode') as TJSONObject;
joItemName := joListOfCatalogueItem.GetValue('itemName') as TJSONObject;
joPriceInfo := joListOfCatalogueItem.GetValue('priceInfo') as TJSONObject;
joTradingInfo := joListOfCatalogueItem.GetValue('tradingInfo') as TJSONObject;
```

DB更新用SQLの設定

```
SQLQuery1.SQL.Text := 'UPDATE WSHNMAS SET' +
                      ' SHSHNM = :SHNM, SHSHRK = :SHRK, SHTANK = :TANK,' +
                      ' SHGENK = :GENK, SHZKSU = :ZKSU' +
                      ' WHERE SHSHCD = :SHCD';
```

2－5. RESTサーバーアプリ作成方法

補足

② POSTリクエストによるEDI商品データの受け取り

- POSTメソッドでJSONデータとして連携する例
  - JSON形式のデータを加工し、DBを更新

ServerMethodsUnitのJSONデータよりDBを更新するメソッド

```
SQLQuery1.ParamByName('SHNM').AsString :=
  joItemName.GetValue('tradeItemLongDescription').Value;
SQLQuery1.ParamByName('SHRK').AsString :=
  joItemName.GetValue('tradeItemShortDescription').Value;
SQLQuery1.ParamByName('TANK').AsInteger :=
  StrToIntDef(joPriceInfo.GetValue('itemSellingUnitPrice').Value, 0);
SQLQuery1.ParamByName('GENK').AsInteger :=
  StrToIntDef(joPriceInfo.GetValue('itemNetUnitPrice').Value, 0);
SQLQuery1.ParamByName('ZKSU').AsInteger :=
  StrToIntDef(joTradingInfo.GetValue('sellingStock').Value, 0);

SQLQuery1.ParamByName('SHCD').AsString :=
  joItemCode.GetValue('buyerAssignedTradeItemIdentification').Value;

SQLQuery1.ExecSQL;

Result := '1';
end;
```

DBへ更新する各項目の値を設定

商品名

商品略称

単価

原価

在庫数

キー情報（商品CD）

DBを更新

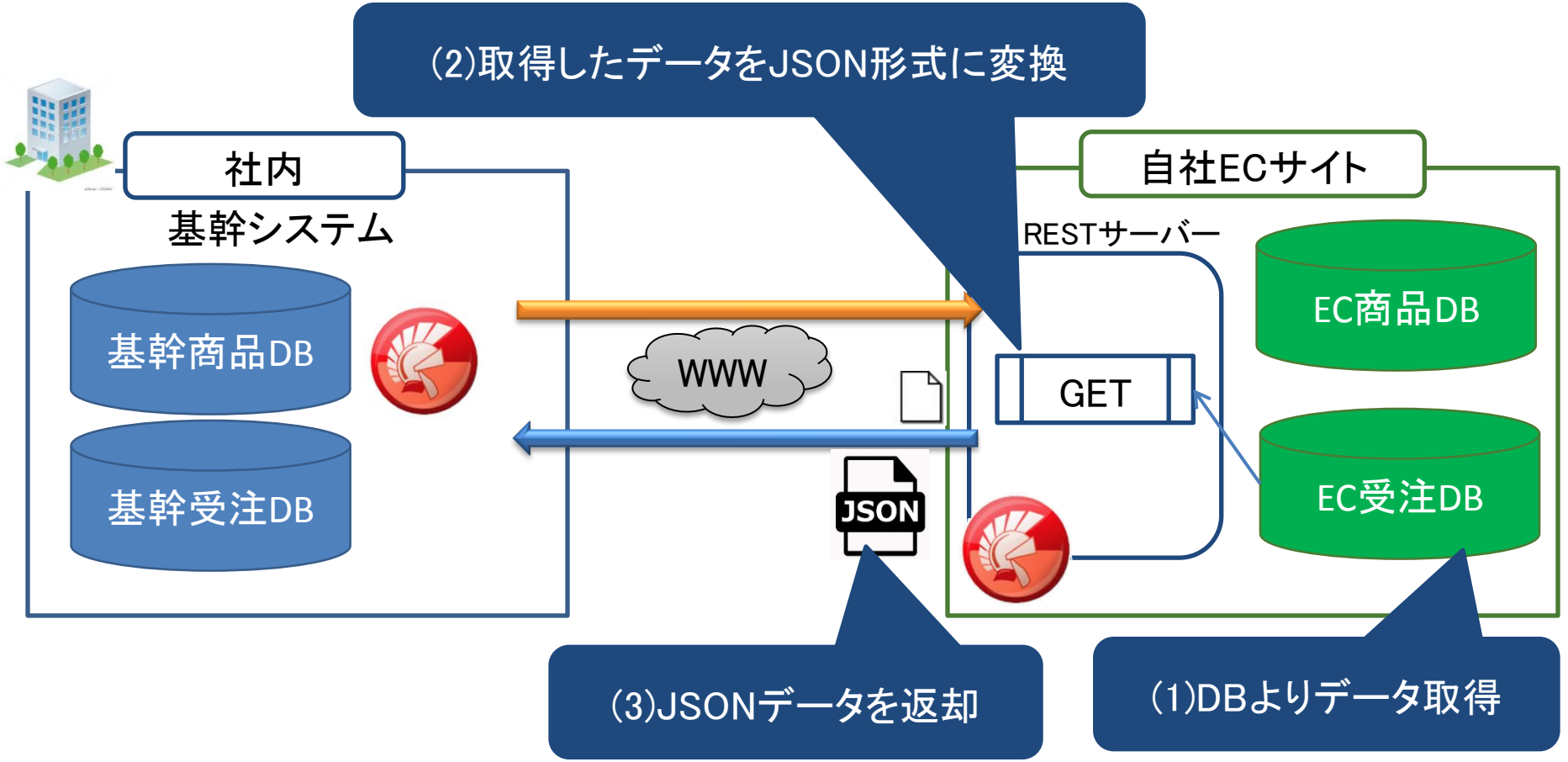
正しく更新出来た場合、'1'を返却

このメソッドを取引先より使用する場合、POSTメソッドのURLには
「https://（サーバー名）/datasnap/rest/TServerMethods1/Data」を指定

2－5. RESTサーバーアプリ作成方法

補足

③ GETリクエストに対するEDI受注データの配信



2－5. RESTサーバーアプリ作成方法

補足

③ GETリクエストに対するEDI受注データの配信

- GETメソッドでJSONデータとして連携する例

ServerMethodsUnitのJSONデータを返却するメソッド

```
  public
    function ReturnData(): String;
```

public宣言に関数を追加

```
uses Data.DBXJSON;
```

uses節にJSONを扱うためのユニットを追加

```
function TServerMethods1. ReturnData(): string;
var
  joJSON: TJSONObject;
  joStandardBusinessDocument: TJSONObject;
  joDocumentBody: TJSONObject;
  joListOfOrders: TJSONObject;
  joOrder: TJSONObject;
  jaLineItems: TJSONArray;
  joLineItem: TJSONObject;
begin
  SQLQuery1.SQL.Text := 'SELECT * FROM WJUCTRN';
  SQLQuery1.Open;
```

JSONの作成を行なうための変数を定義

受注データを取得

2－5. RESTサーバーアプリ作成方法

補足

③ GETリクエストに対するEDI受注データの配信

- GETメソッドでJSONデータとして連携する例

ServerMethodsUnitのJSONデータを返却するメソッド

```
joJSON := TJSONObject.Create;
joStandardBusinessDocument := TJSONObject.Create;
joDocumentBody := TJSONObject.Create;
joListOfOrders := TJSONObject.Create;
joOrder := TJSONObject.Create;
jaLineItems := TJSONArray.Create;
try
  joOrder.AddPair('OrderDate', SQLQuery1.FieldByName('JUJCDT').AsString);
  joOrder.AddPair('ShipToCode', SQLQuery1.FieldByName('JUSPCD').AsString);
  joOrder.AddPair('PayeeCode', SQLQuery1.FieldByName('JUSKCD').AsString);

  while not SQLQuery1.Eof do
  begin
    joLineItem := TJSONObject.Create;
    joLineItem.AddPair('ItemID', SQLQuery1.FieldByName('JUSHCD').AsString);
    joLineItem.AddPair('UnitPrice', SQLQuery1.FieldByName('JUTANK').AsString);
    joLineItem.AddPair('Quantity', SQLQuery1.FieldByName('JUJUSU').AsString);

    jaLineItems.Add(joLineItem);
```

JSONデータ作成用オブジェクトを生成

各項目のJSONデータを作成

受注日

納入先CD

請求先CD

明細データを作成

商品CD

単価

受注数

明細データを配列に追加

2－5. RESTサーバーアプリ作成方法

補足

③ GETリクエストに対するEDI受注データの配信

- GETメソッドでJSONデータとして連携する例

ServerMethodsUnitのJSONデータを返却するメソッド

```
      SQLQuery1.Next;
    end;
    joOrder.AddPair('LineItems', jaLineItems);
    joListOfOrders.AddPair('Order', joOrder);
    joDocumentBody.AddPair('ListOfOrders', joListOfOrders);
    joStandardBusinessDocument.AddPair('DocumentBody', joDocumentBody);
    joJSON.AddPair('StandardBusinessDocument', joStandardBusinessDocument);

    Result := HTTPEncode(joJSON.ToString);
  finally
    SQLQuery1.Close;
  end;
end;
```

返却するJSON形式の構造に合わせ、データを作成

作成したJSONデータを返却

このメソッドを取引先より使用する場合、GETメソッドのURLには
「https://（サーバー名）/datasnap/rest/TServerMethods1/ReturnData」を指定

3. まとめ

3. まとめ

- インターネットEDI
  - 通信方式やデータ形式が標準化されたことで
    基幹システムと外部（取引先やECサイト等）との連携が促進

- Delphiを使ったEDIサービス連携
  - REST方式を使うことで、データ連携が容易に可能
  - JSONやXMLなどの様々なデータ形式にも対応可能

- RESTサーバアプリの作成方法
  - DataSnap技術でRESTサービス自体の構築も可能

ご静聴ありがとうございました