

【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

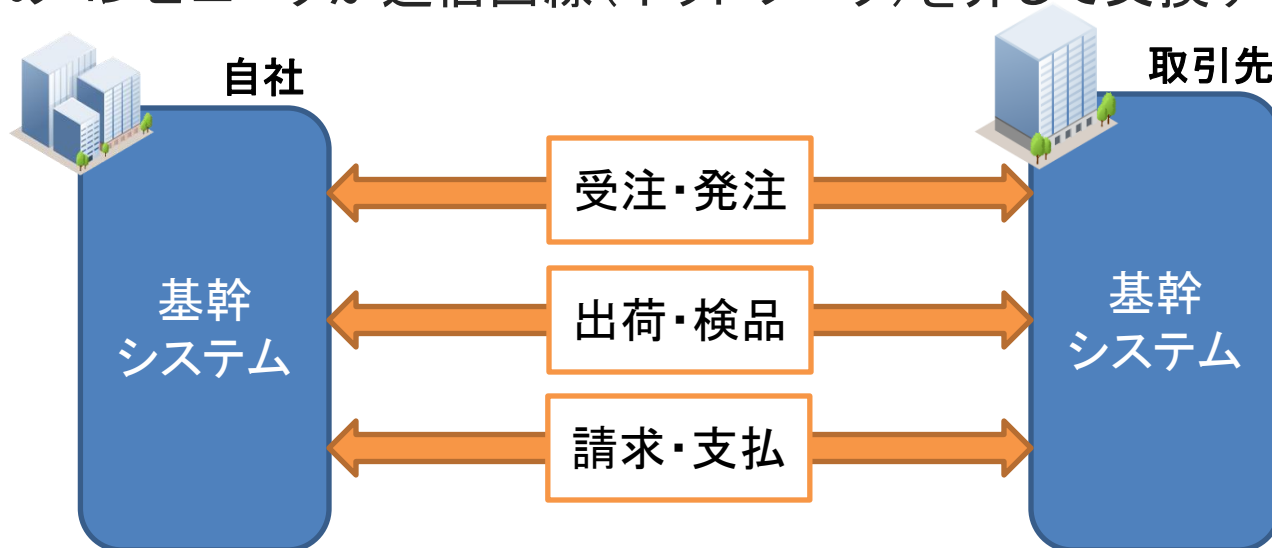
1. インターネットEDIについて

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。



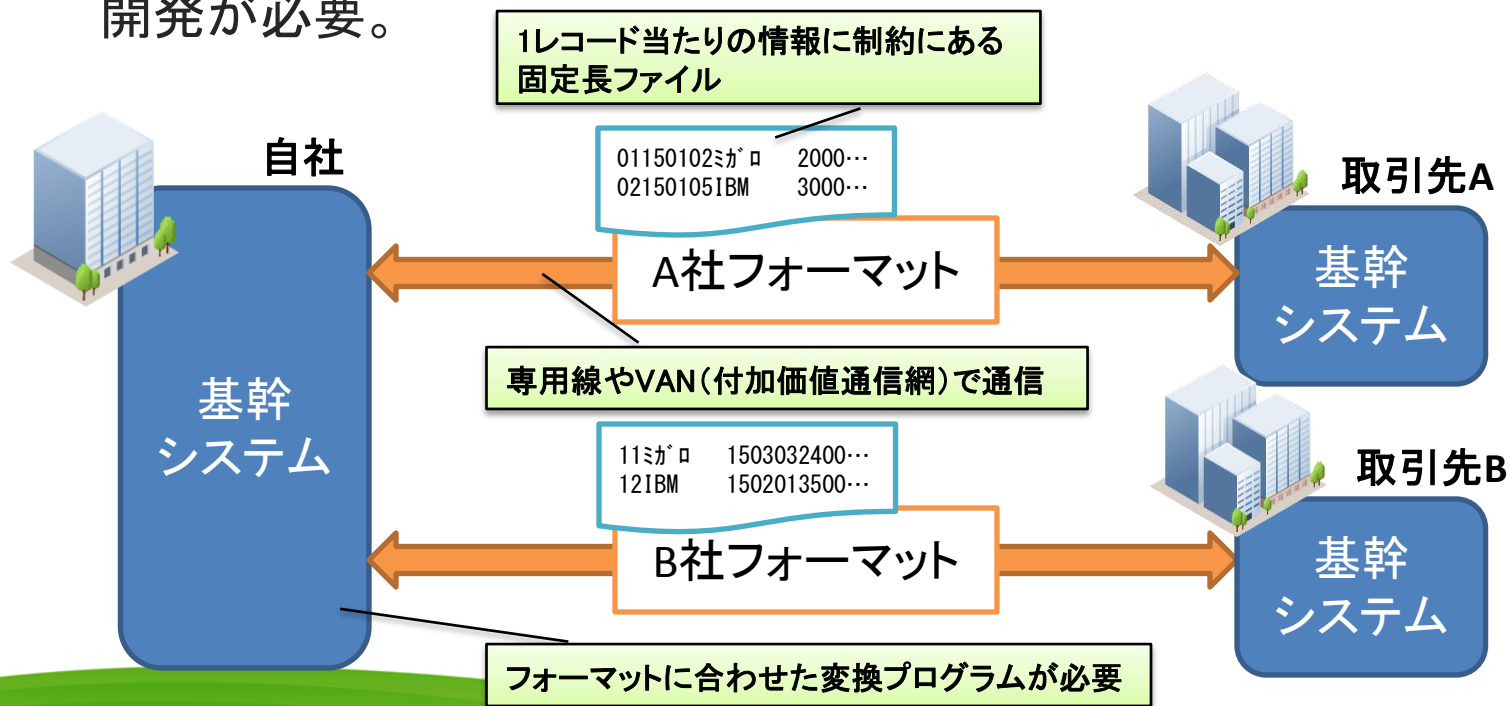
(メリット)

- 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

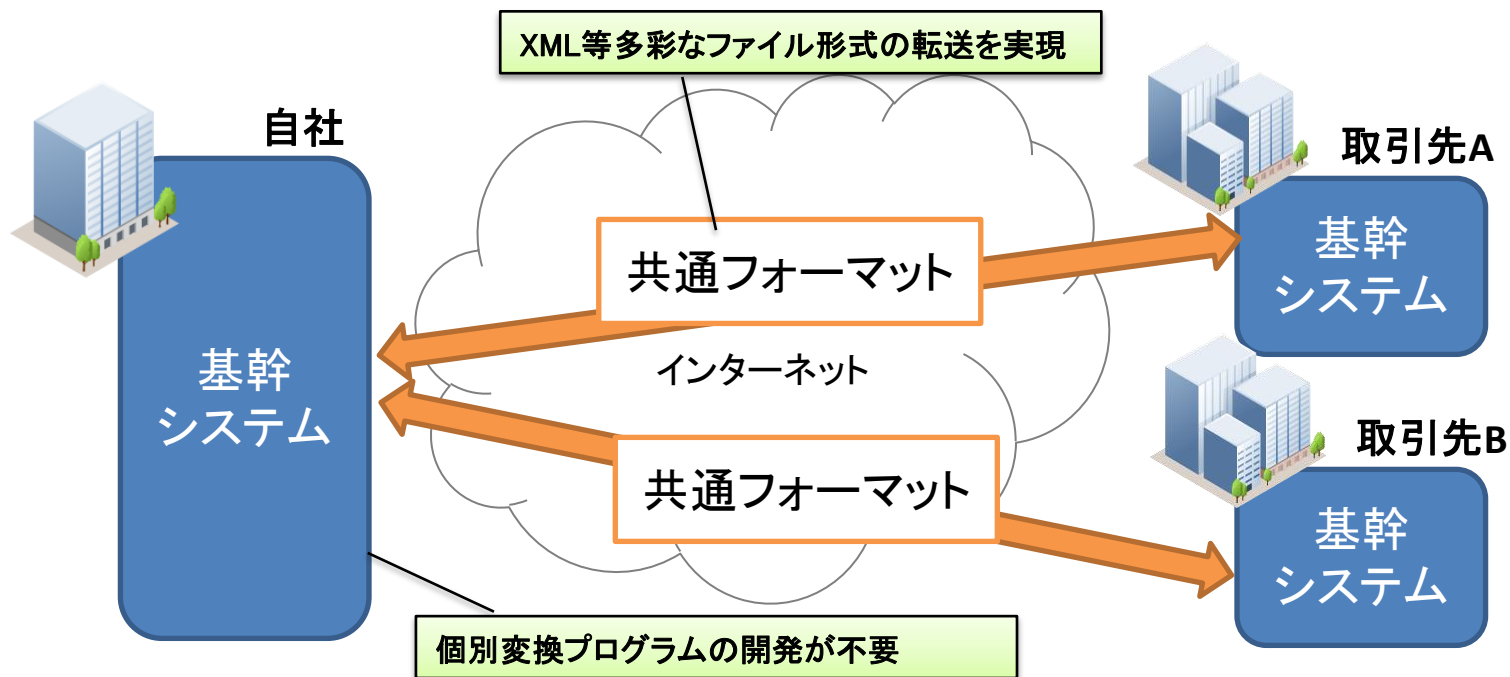
- ① 通信手段に専用線やVAN(付加価値通信網)を利用しているため導入や運用にコストがかかる。
- ② 一般的に低速な通信環境の為、転送データに固定長ファイルを使用することが多く、転送情報量に制限がある。
- ③ 連携する企業ごとにフォーマットが異なる為、個別変換プログラムの開発が必要。



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。



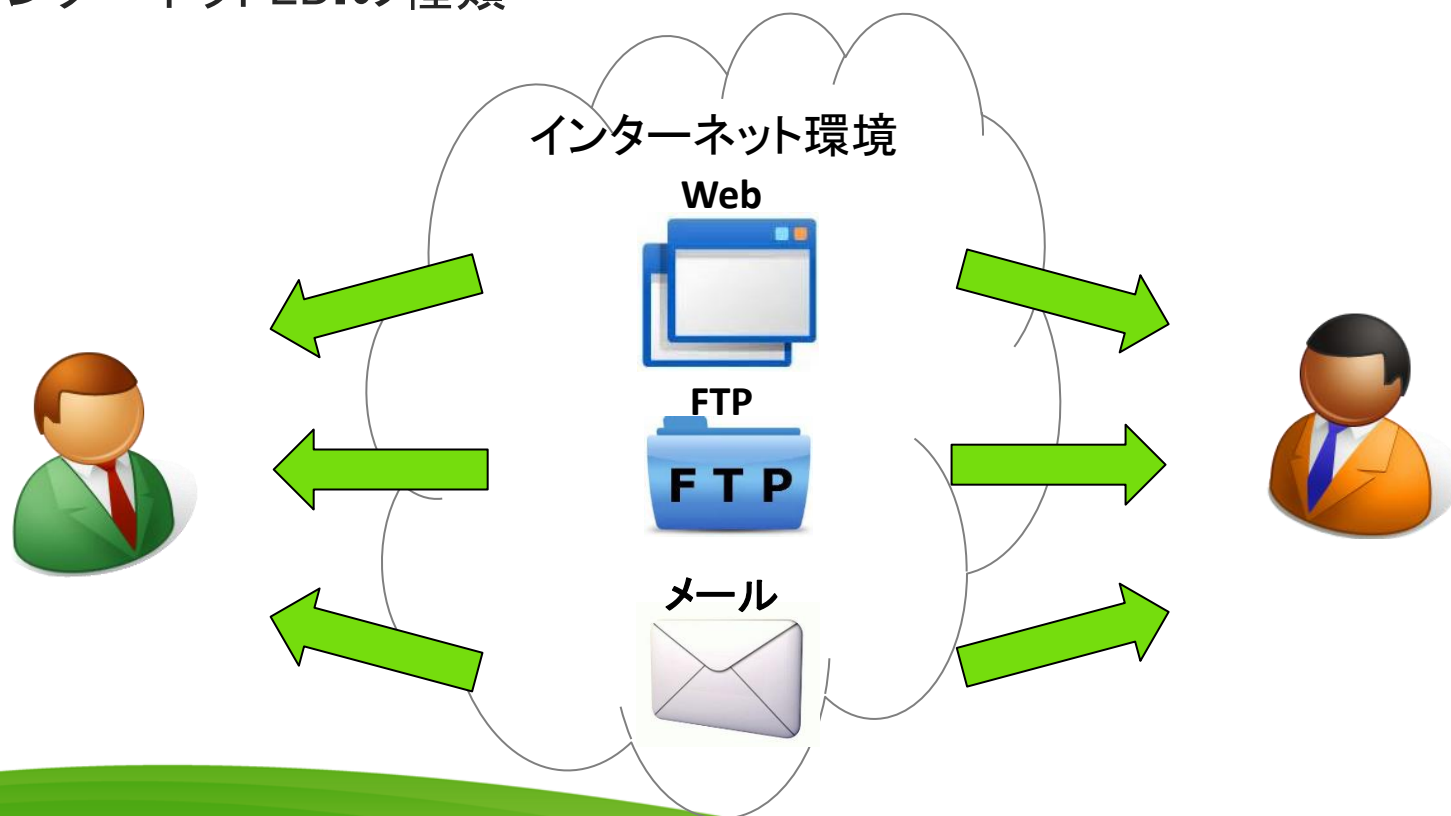
今回のテーマ : インターネットを使用したEDIデータ連携

1. インターネットEDIについて

- インターネットEDIとは？

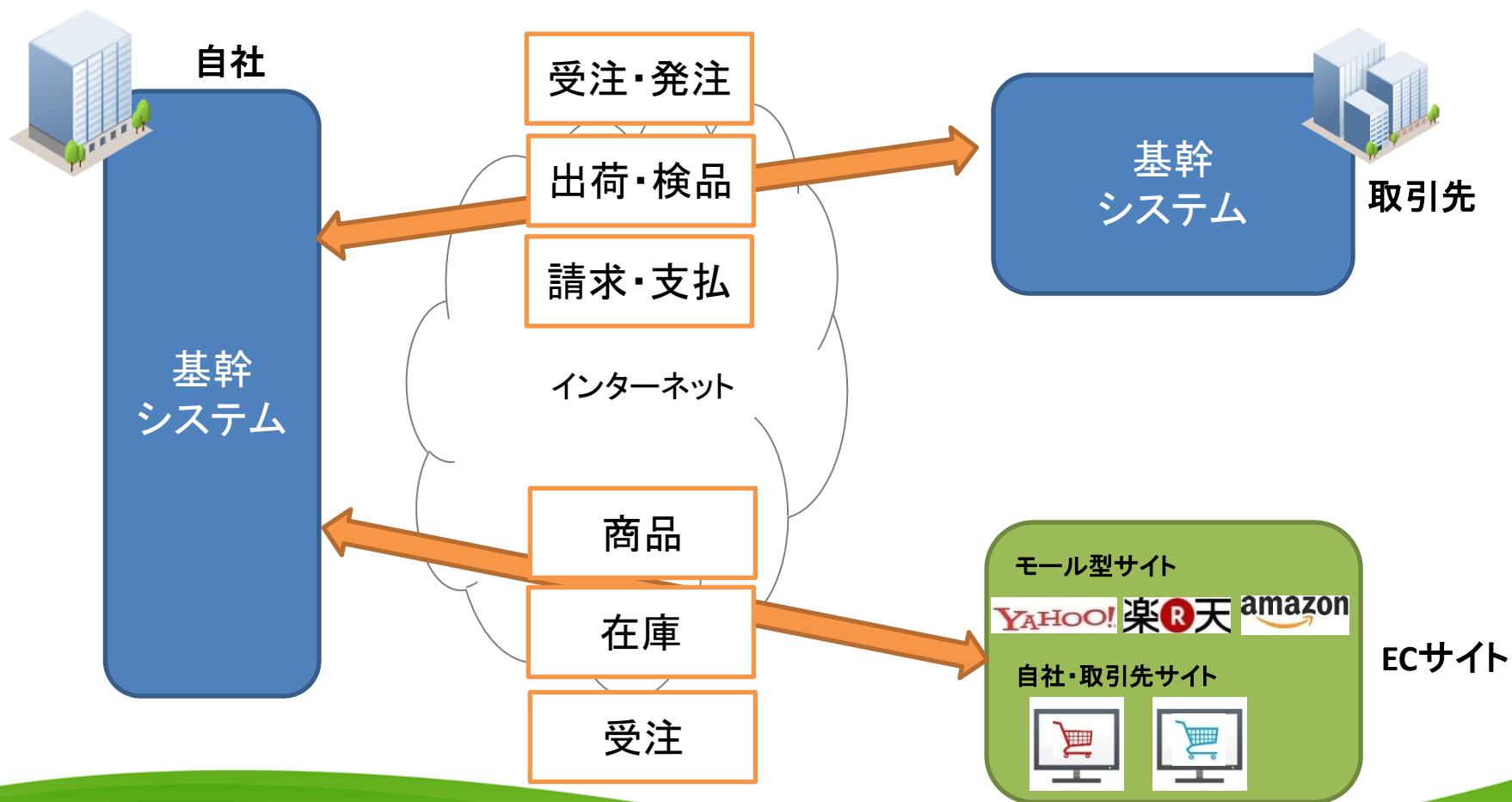
従来のEDIは通信手段にコストがかかるが、インターネットを通信手段とすることによりコストを削減できるようになったEDI。

インターネットEDIの種類



1. インターネットEDIについて

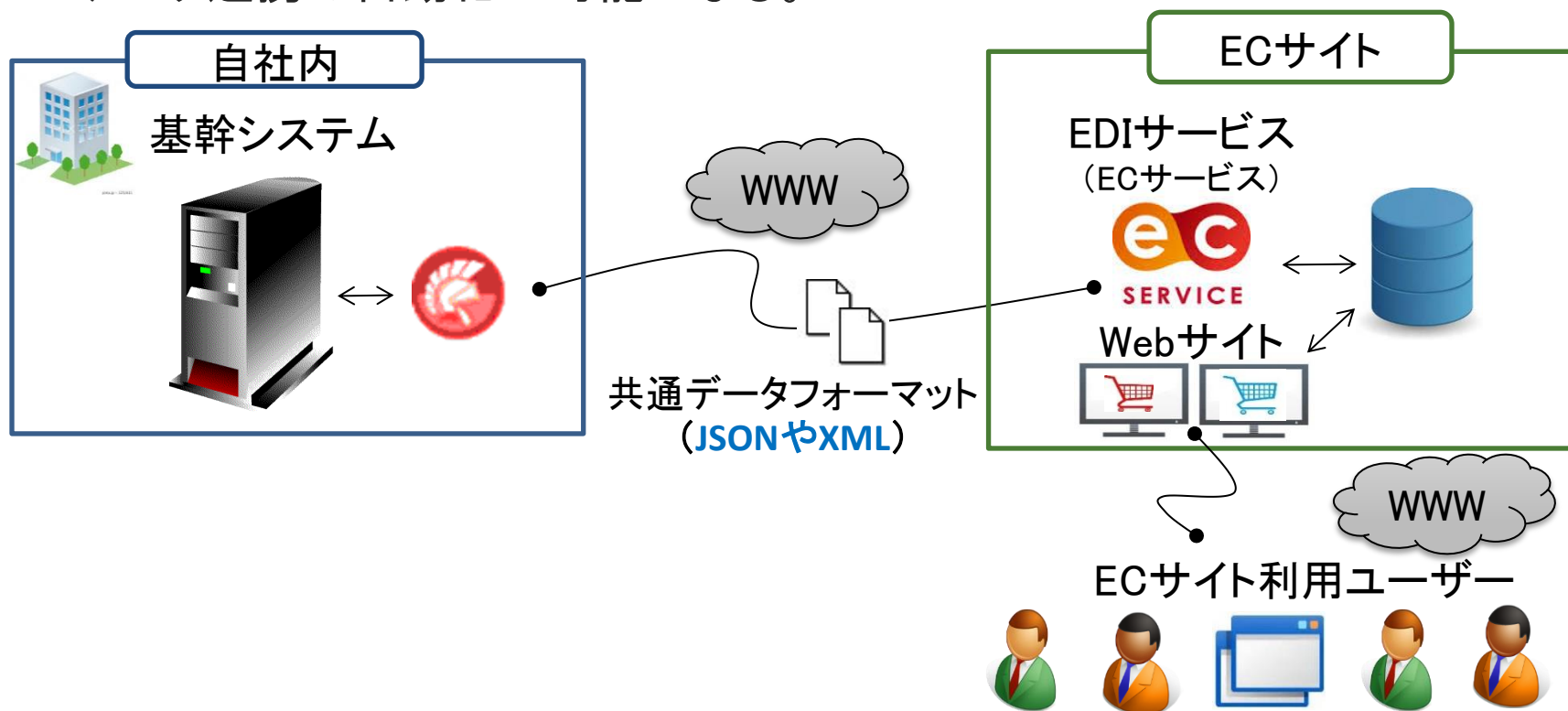
- インターネットEDIを使った連携
インターネット化により、EDIは企業間通信のみならず、
基幹システムとECサイト等とのデータ連携も実現。



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。



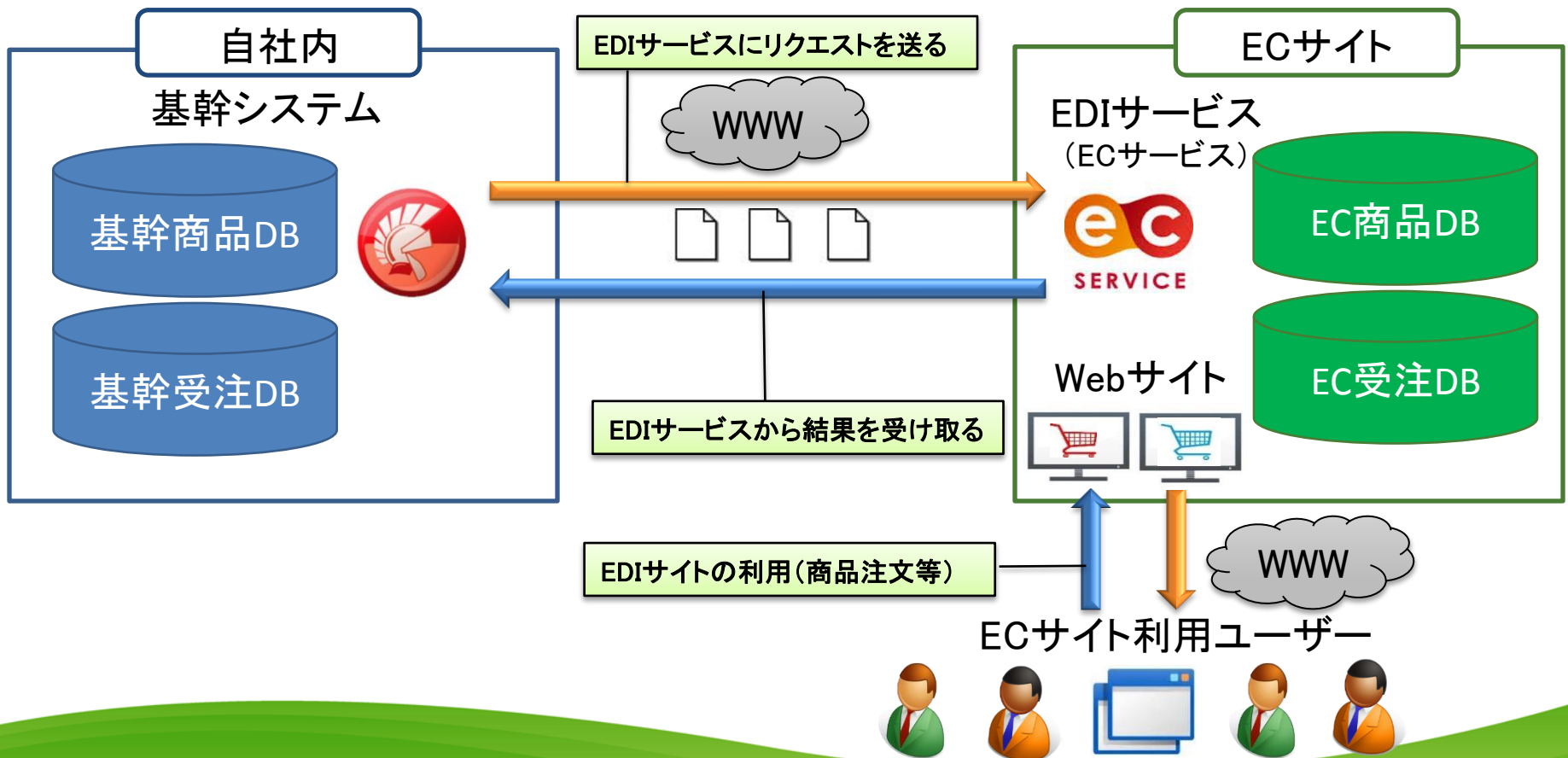
基幹システムとECサイトとの連携を例に、インターネットEDIの開発手法をご紹介します

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。

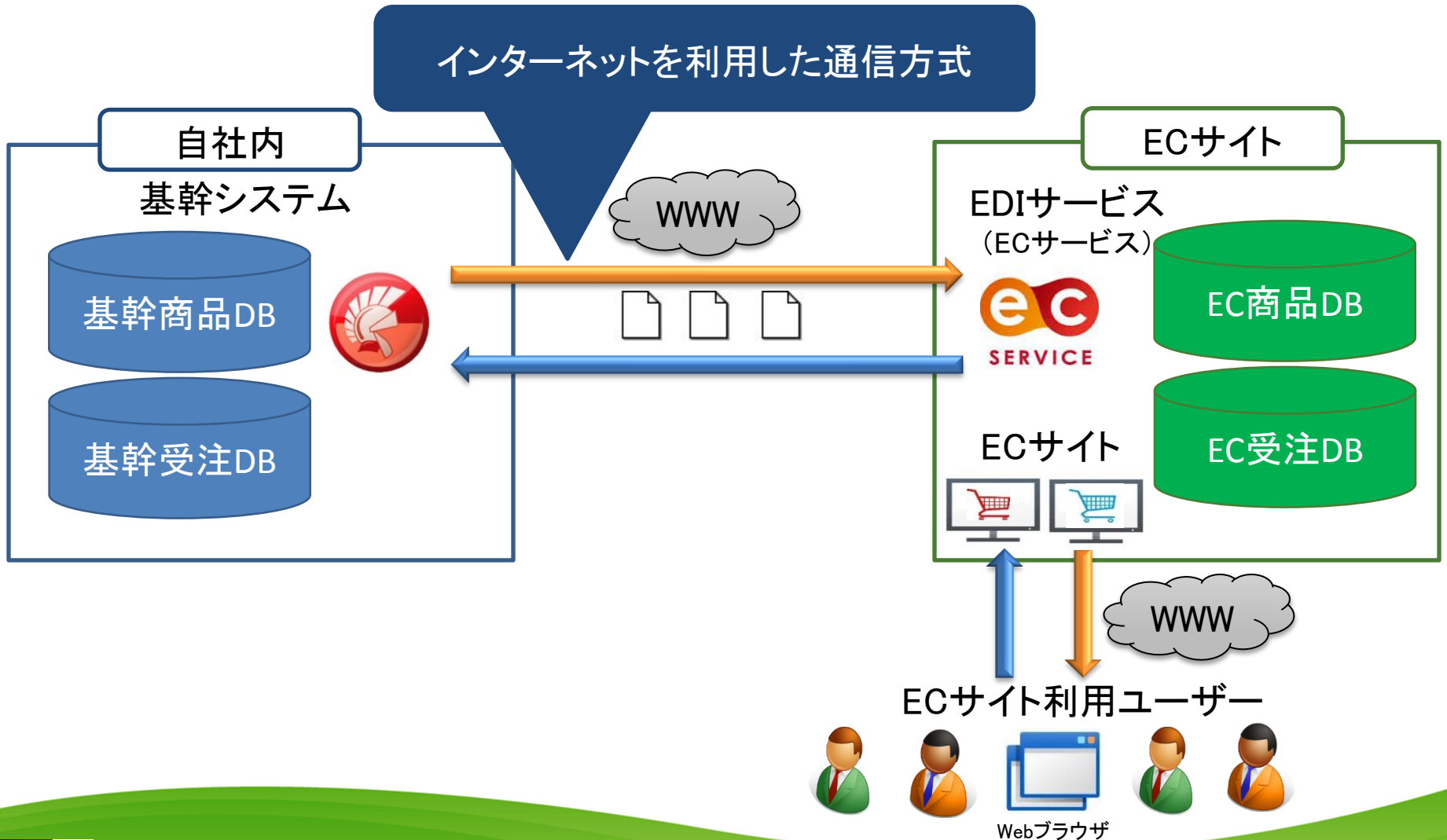
<仕組みの概要>



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

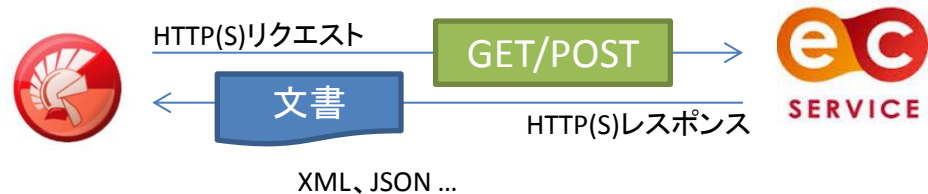


2-1. 通信方式

- インターネットを使用した主な通信方式

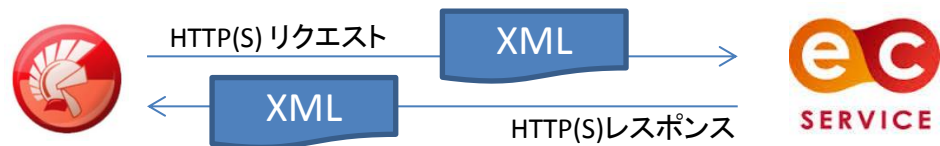
① REST (Representational State Transfer)

- 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。

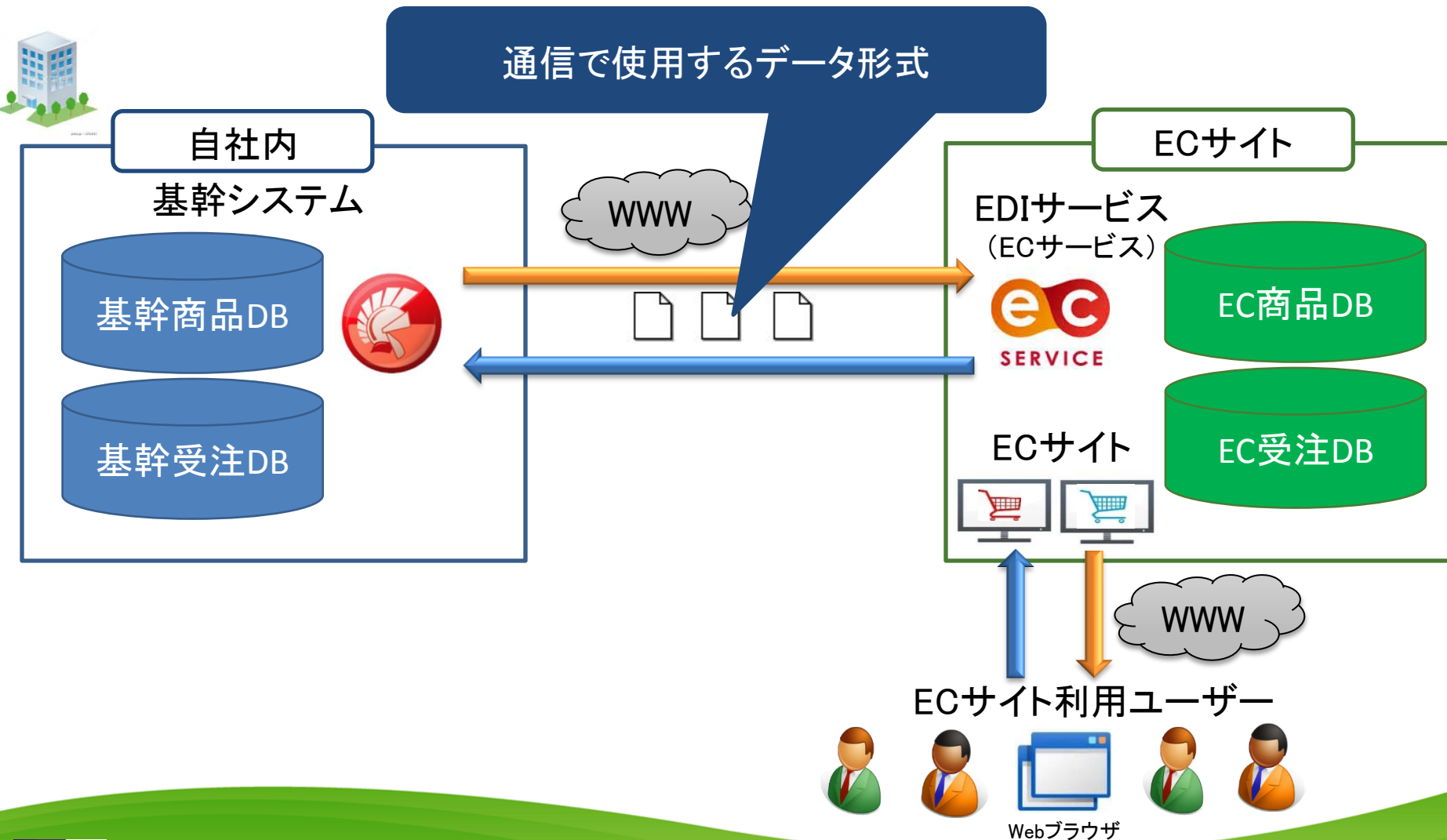


近年のインターネットEDIにおいては、RESTが主流になっています

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

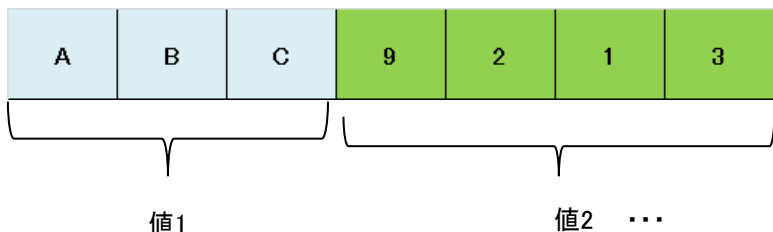


2-2. 通信データ形式

- 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。

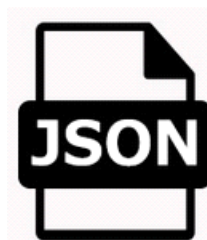
これまでの傾向

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のため汎用性が高く、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

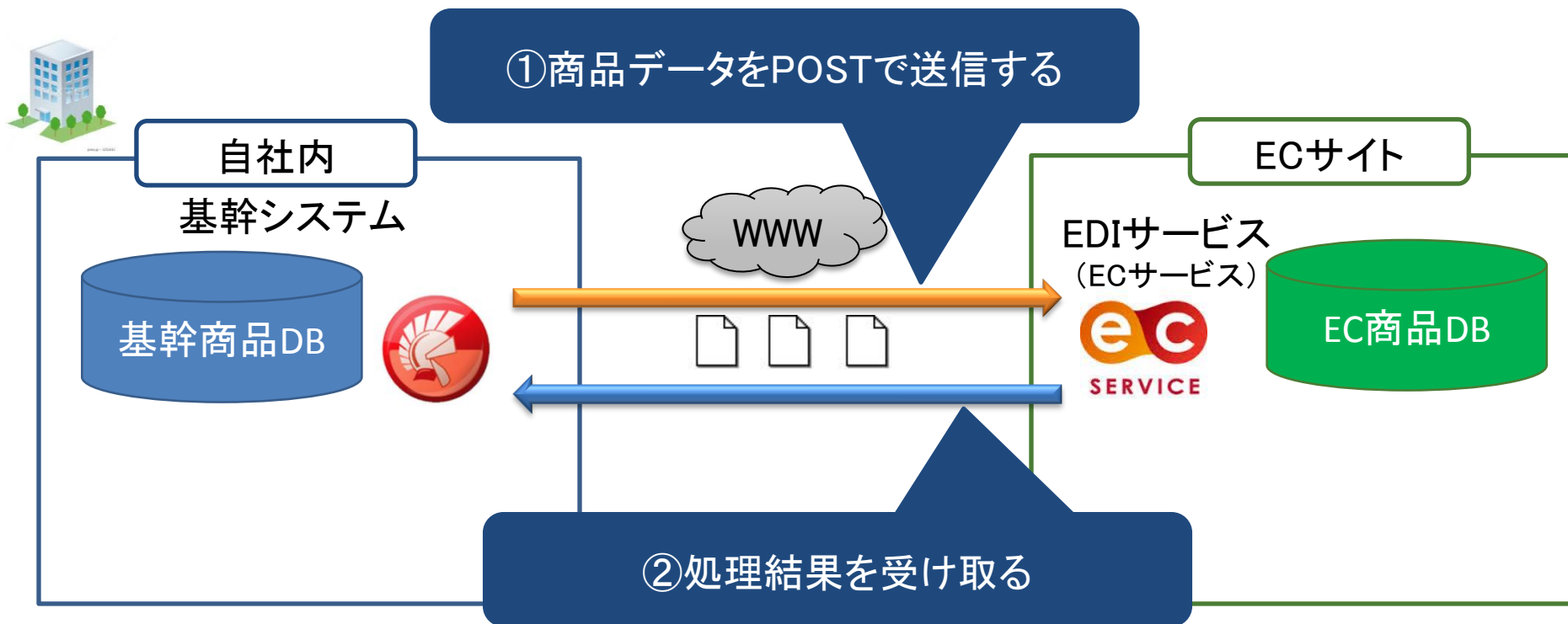
例)

```
<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

商品コード	N7001	
商品名	ホッチキス	
商品略称	サクリフラット	カナ名
販売単価	9,999	
原価		
在庫数	500	
販売開始日	2015/06/01	
販売終了日	2015/12/31	
<メーカー情報>		
正式商品名	サクリフラット	
希望小売価格	650	
EDJ商品データのアップロード		



ECサイト

ECサイト上の商品データに反映



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
```

HTTPSリクエストの送信にはTIdHTTPコンポーネントを使用

```
IdHTTP1 : TIdHTTP;
```

```
PostStream : TStringList;
```

POSTメソッド実行時の引数用変数

```
ResStream : TStringStream;
```

```
joJSON: TJSONObject;
```

```
joStandardBusinessDocument: TJSONObject;
```

```
joMessage: TJSONObject;
```

```
joListOfCatalogueItem: TJSONObject;
```

```
joItemCode: TJSONObject;
```

```
joItemName: TJSONObject;
```

```
joPriceInfo: TJSONObject;
```

```
joTradingInfo: TJSONObject;
```

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

各項目のJSONデータを作成

```
try
joItemCode.AddPair('buyerAssignedTradeItemIdentification',
    SQLQuery1.FieldByName('SHHNCB').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);
```

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);
```

送信するJSON形式の構造に合わせ、
データを作成

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

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

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

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

```
if ResStream.DataString = '1' then  
begin
```

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

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

DBのデータを連携済みに更新

「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）

- 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>
```

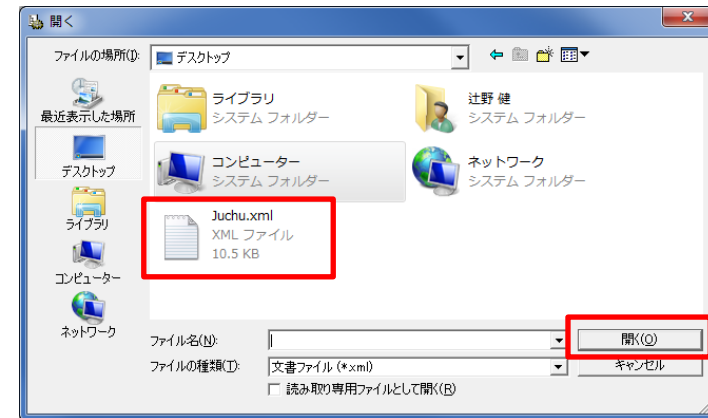
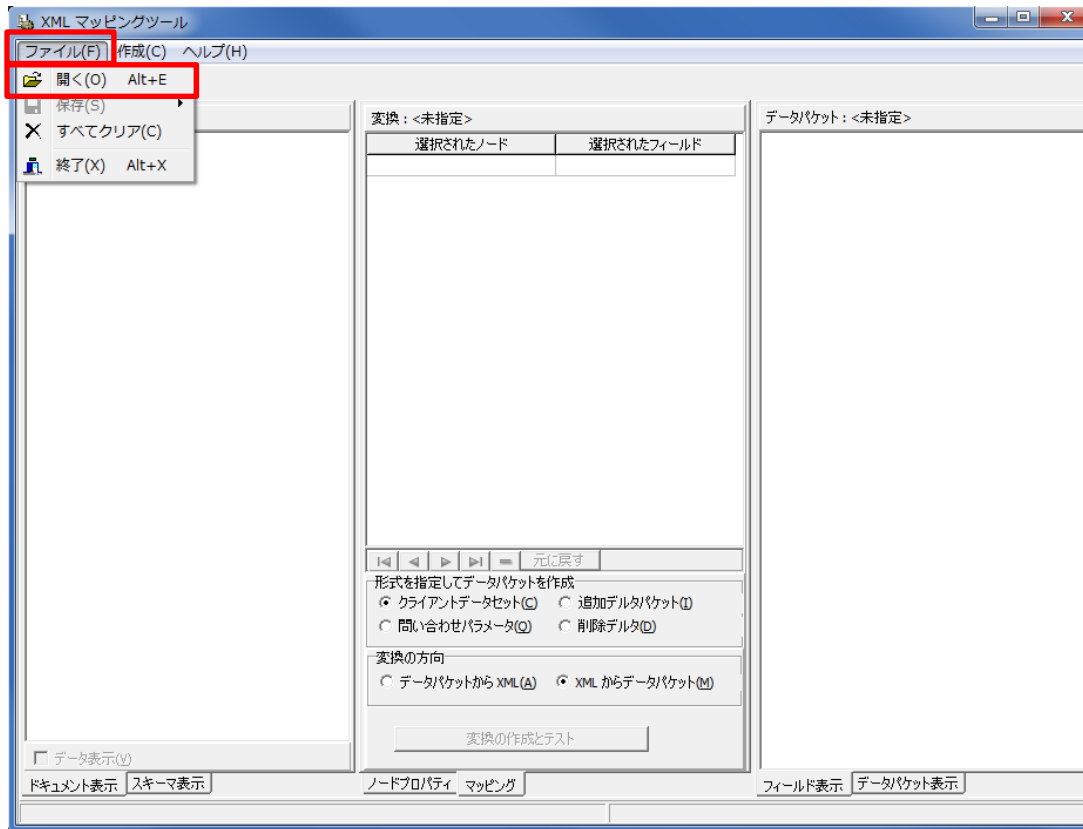
<商品データ>

- ・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）

XMLクラスの作成手順 I

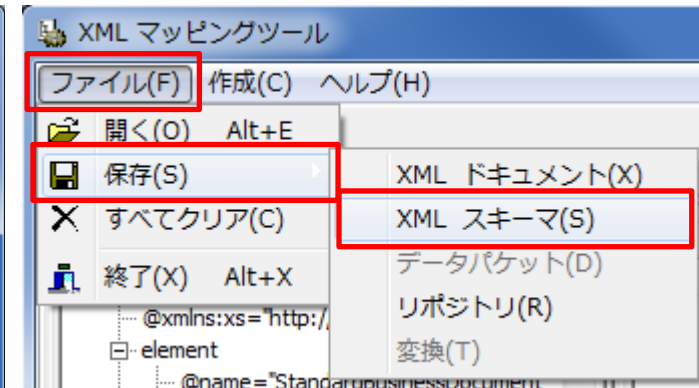
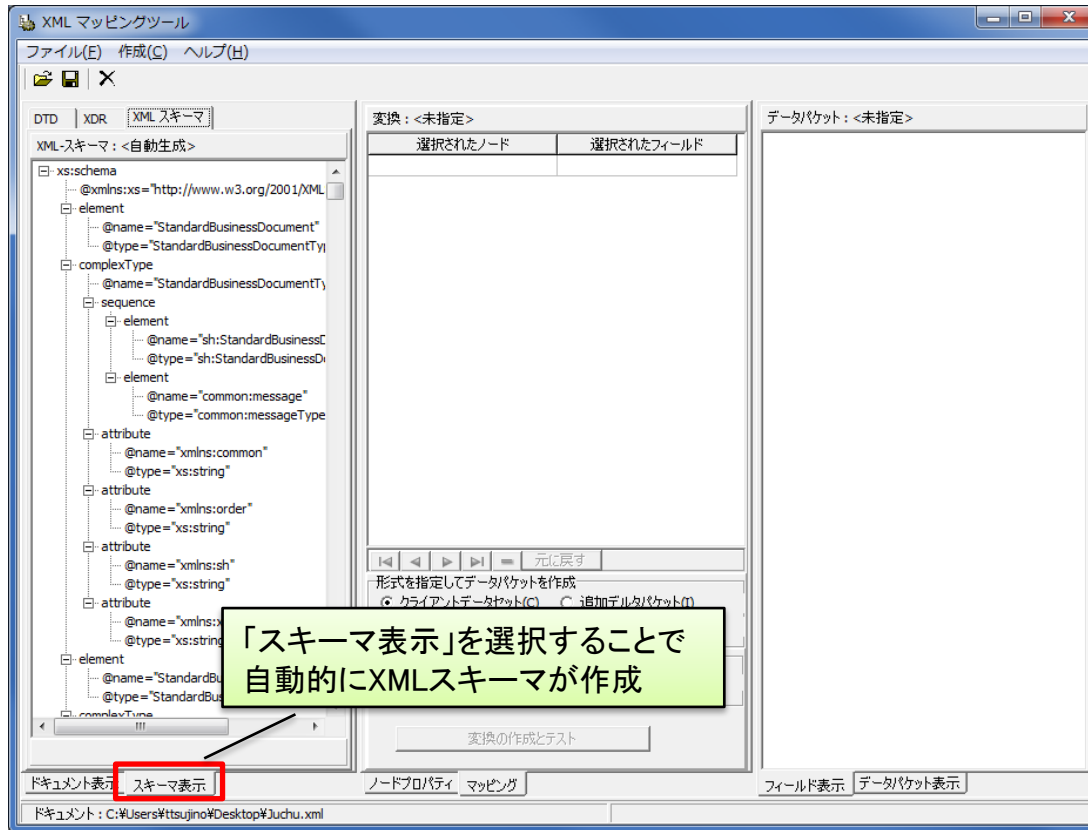
- Delphiに付属している「xmlmapper.exe」を利用して、XMLスキーマを作成



2-3. EDIデータの送信（XML版実装）

XMLクラスの作成手順 II

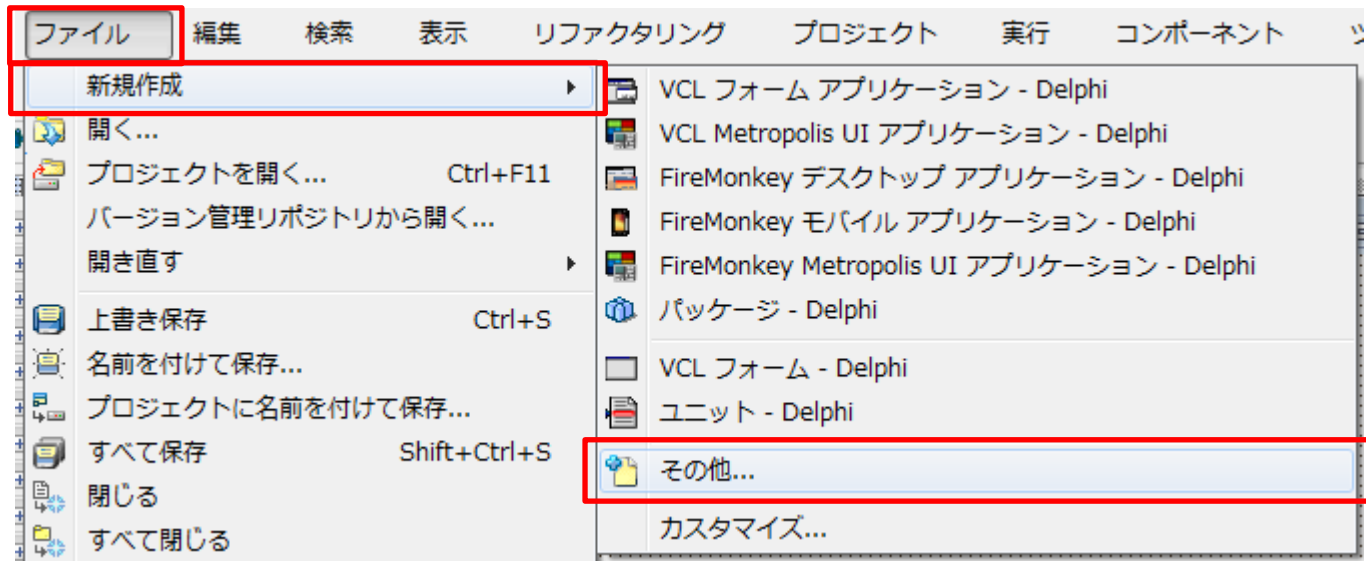
- Delphiに付属している「xmlmapper.exe」を利用して、XMLスキーマを作成



2-3. EDIデータの送信（XML版実装）

XMLクラスの作成手順Ⅲ

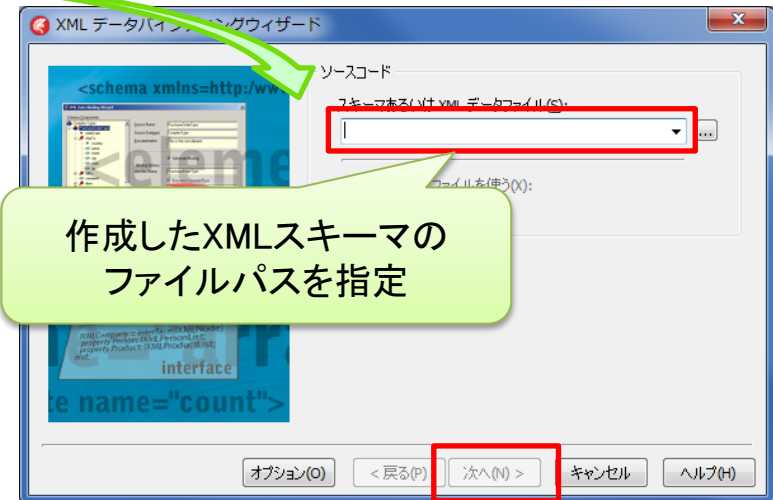
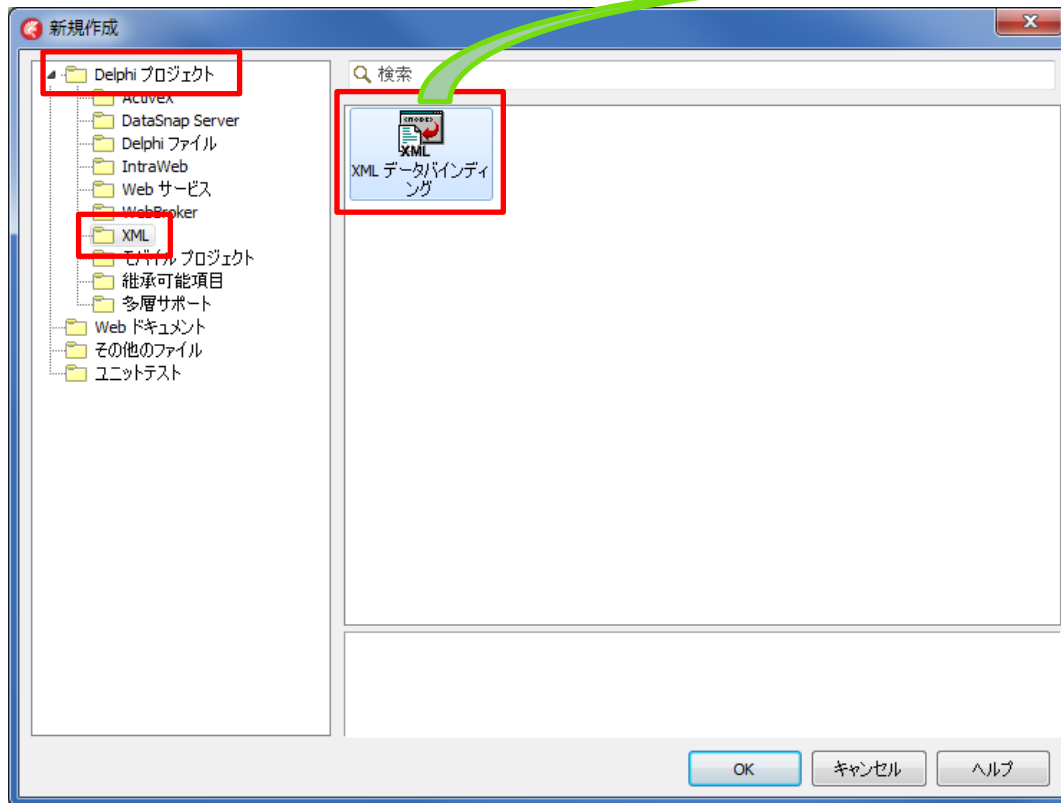
- XMLスキーマを利用して、XMLのクラスを作成
 - [ファイル | 新規作成 | その他]を選択



2-3. EDIデータの送信（XML版実装）

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]



作成したXMLスキーマの
ファイルパスを指定

2-3. EDIデータの送信（XML版実装）

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

XML データバインディングウィザード

スキーマコンポーネント:

- 複合型
 - StandardBusinessDocumentType
 - StandardBusinessDocumentHeaderType
 - SenderType
 - IdentifierType
 - ReceiverType
 - DocumentIdentificationType
 - BusinessScopeType
 - ScopeType
 - messageType
 - entityIdentificationType
 - messageInfoType
 - listOfOrdersType
 - contentVersionType
 - documentStructureVersionType
 - naverType

ソース名(S): StandardBusinessDocumentType

ソースデータ型(T): ComplexType

ドキュメントを作成(D)

識別子名(I): IXMLStandardBusinessDocumentType

要索名(L): StandardBusinessDocumentType

チェックを付ける

ドキュメント要素型(E)

XML データバインディングウィザード

バインドの一覧(G)

生成されたインターフェイス

- IXMLStandardBusinessDocumentType
- IXMLStandardBusinessDocumentHeaderType
- IXMLSenderType
- IXMLIdentifierType
- IXMLReceiverType
- IXMLDocumentIdentificationType
- IXMLBusinessScopeType

データバインディングの設定(S)

- 設定を保存しない(N)
- スキーマファイルに保存(O)
- ファイルに保存(I)

「設定を保存しない」を選択

XML データバインディング

作成日: 2015/06/19 15:06:20
作成元: C:\projects\MIGARO\Technical\2015_2Q\Delphi\POST\CS_XML\Shohin

```

unit Shohin;
interface
uses XmlDom, XMLDoc, XMLIntf;
type
  IXMLBusinessDocumentType = interface;
  IXMLBusinessDocumentHeaderType = interface;
  IXMLSenderType = interface;
  IXMLIdentifierType = interface;
  IXMLReceiverType = interface;
  IXMLDocumentIdentificationType = interface;
  IXMLBusinessScopeType = interface;
  IXMLScopeType = interface;
  IXMLMessageType = interface;
  IXMLEntityIdentificationType = interface;
  IXMLMessageInfoType = interface;
  IXMLListofOrdersType = interface;
  IXMLContentVersionType = interface;
  IXMLDocumentStructureVersionType = interface;
  IXMLNaverType = interface;
  procedure Set_Xml(value: UnicodeString);
  { メソッドとプロパティ }
  property Common: UnicodeString read Get_Common;
  property Order: UnicodeString read Get_Order;
  
```

XMLにプロパティでアクセス可能なクラスが自動的に生成される

完了(F)

2-3. EDIデータの送信（XML版実装）

①商品データをPOSTで送信する

XMLデータ作成処理

```
uses Shohin;
```

先程作成したXML専用クラスをuses節に追加

```
procedure TForm1.btnPOSTClick(Sender: TObject);
```

```
var
```

```
idHTTP : TIdHTTP;
```

```
PostStream : TStringList;
```

```
ResStream : TStringStream;
```

POSTメソッド実行時の引数用変数

```
xsbusbXMLBusinessDocument : IXMLBusinessDocumentType;
```

XMLを扱うための変数を定義

```
begin
```

```
idHTTP := TIdHTTP.Create;
```

```
PostStream := TStringList.Create;
```

```
ResStream := TStringStream.Create;
```

```
try
```

```
idHTTP.IOHandler := TIdSSLIOHandlerSocketOpenSSL.Create;
```

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

```
SQLQuery1.Open;
```

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）

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
```

タグのみのXMLを作成

```
  xsbXMLBusinessDocument := NewBusinessDocument;
```

XMLにデータを追加

```
  xsbXMLBusinessDocument.Message.ListOfCatalogueItems.ItemCode.SupplierItemCode :=  
    SQLQuery1.FieldByName('SHSHCD').AsString;
```

商品CD

```
  xsbXMLBusinessDocument.Message.ListOfCatalogueItems.ItemName.TradeItemLongDescription :=  
    SQLQuery1.FieldByName('SHSHNM').AsString;
```

商品名

```
  xsbXMLBusinessDocument.Message.ListOfCatalogueItems.ItemName.TradeItemShortDescription :=  
    SQLQuery1.FieldByName('SHSHRK').AsString;
```

商品略称

```
  xsbXMLBusinessDocument.Message.ListOfCatalogueItems.PriceInfo.ItemSellingUnitPrice.Value :=  
    SQLQuery1.FieldByName('SHTANK').AsString;
```

単価

```
  xsbXMLBusinessDocument.Message.ListOfCatalogueItems.PriceInfo.ItemNetUnitPrice.Value :=  
    SQLQuery1.FieldByName('SHGENK').AsString;
```

原価

```
  xsbXMLBusinessDocument.Message.ListOfCatalogueItems.TradingInfo.sellingStock :=  
    SQLQuery1.FieldByName('SHZKSU').AsString;
```

在庫数

2-3. EDIデータの送信（XML版実装）

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));
```

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

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

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

```
if ResStream.DataString = '1' then  
begin
```

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

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

DBのデータを連携済みに更新

「②処理結果を
受け取る」

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

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

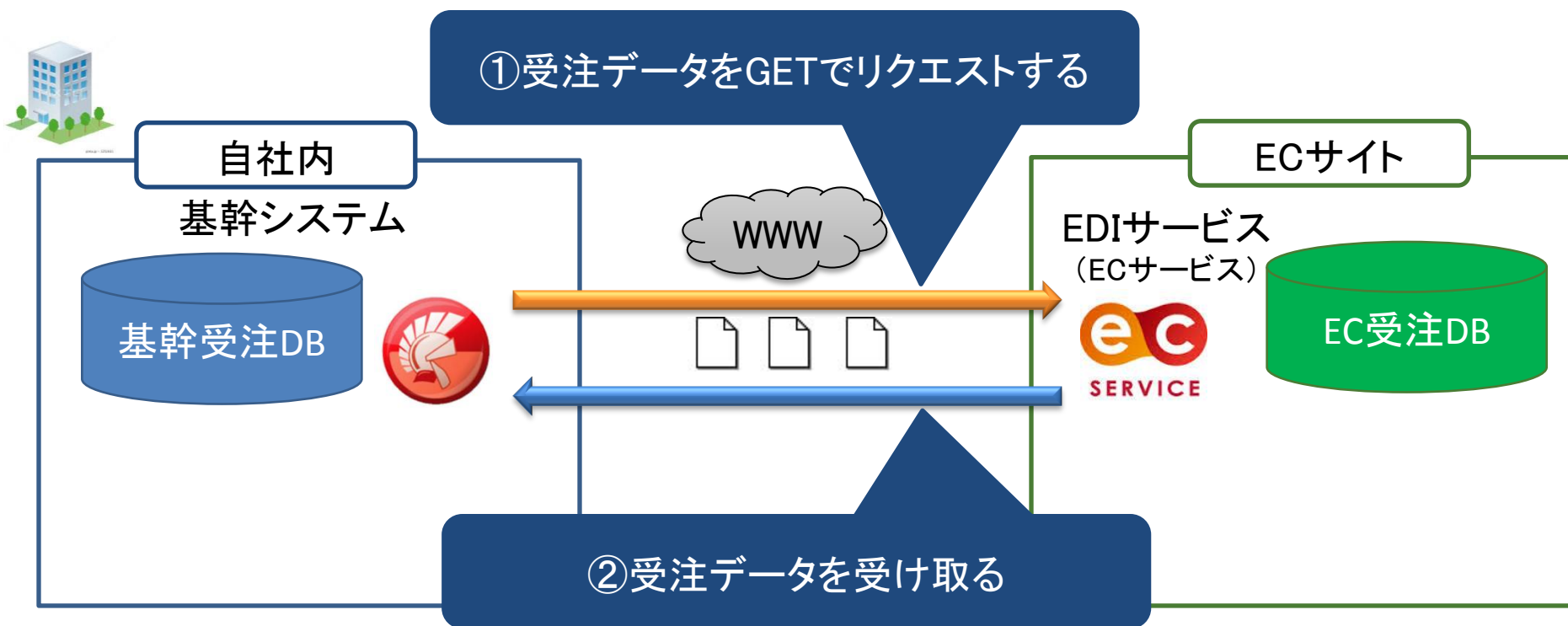
ECサイトへのPOST処理

```
    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

受注照会

受注日 2015/06/18 ~ 2015/06/18

納入先 請求先

受注日	納入先	請求先	商品コード	商品名	受注数	単価	金額
2015/06/18	ミガロ.	ミガロ.	N7001	サクリフラット	5	650	3,250
2015/06/18	ミガロ.	ミガロ.	N7002	サクリ	10	500	5,000
2015/06/18	ミガロ.	ミガロ.	N7003	サクリステッチャー	15	700	10,500
2015/06/18	ミガロ.	ミガロ.	N7004	パイモ11フラット	20	1,400	28,000

EDI受注データを取り込める

EDI受注データのダウンロード

MIGARO. Technical Seminar

カート一覧

商品名	単価	数量
サクリフラット 商品CD: N7001 現在、10点在庫有り。	¥ 650	5 <input type="button" value="削除"/>
サクリ 商品CD: N7002 現在、20点在庫有り。	¥ 500	10 <input type="button" value="削除"/>
サクリステッチャー 商品CD: N7003 現在、30点在庫有り。	¥ 700	15 <input type="button" value="削除"/>
パイモ11フラット 商品CD: N7004 現在、40点在庫有り。	¥ 1,400	20 <input type="button" value="削除"/>

注文



注文

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://XXXXXXXXXXXXXXXXXXXXXXXXXXXX');
```

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;
```

オブジェクトを生成

```
try
```

JSONデータの構造を分割

```
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;
```

DB更新用SQLの設定

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

JSONにアクセスし、
必要な情報を取得

```
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;
```

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）

- 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>
        </lineItem>
      </order>
    </listOfOrders>
  </DocumentBody>
</StandardBusinessDocument>
```

～ 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）

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;
```

XMLを扱うための変数を定義

```
sData: String;
```

```
xdcXML: TXMLDocument;
```

```
xsbXMLStandardBusinessDocument: IXMLStandardBusinessDocumentType;
```

```
xsbXMMLineItemList: IXMMLineItemList;
```

```
xsbXMMLineItem: IXMMLineItemType;
```

```
begin
```

```
idHTTP := TIdHTTP.Create;
```

```
try
```

```
idHTTP.IOHandler := TIdSSLIOHandlerSocketOpenSSL.Create;
```

受注データ(XML)を取得

```
sData := idHTTP.Get( 'https://XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' );
```

2-4. EDIデータの受信（XML版実装）

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

取得したXMLデータを
専用のクラスに変換

DB更新用SQLの設定

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）

②受注データを受け取る

XMLデータ加工処理

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

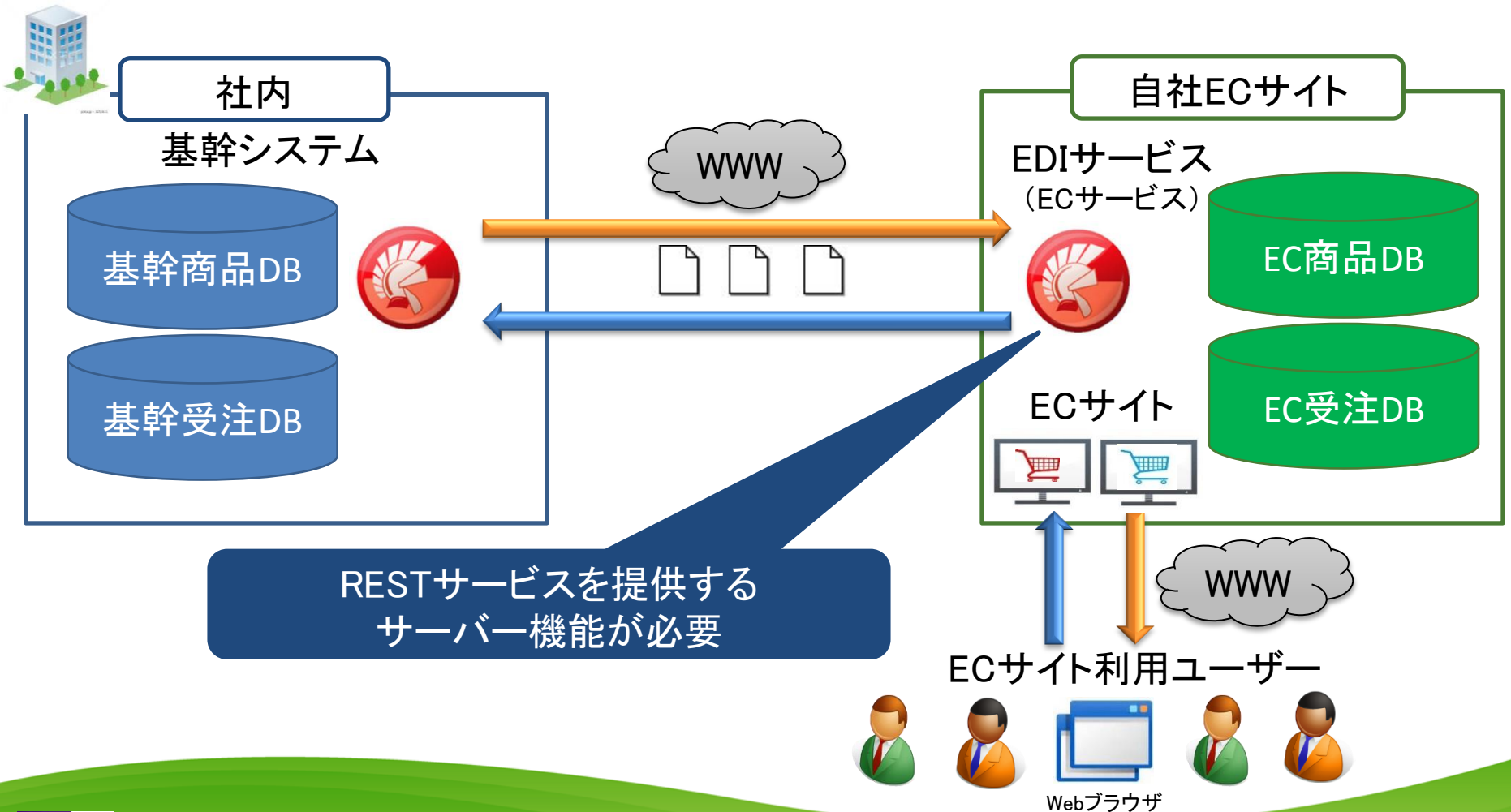
受注数

DBを更新

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

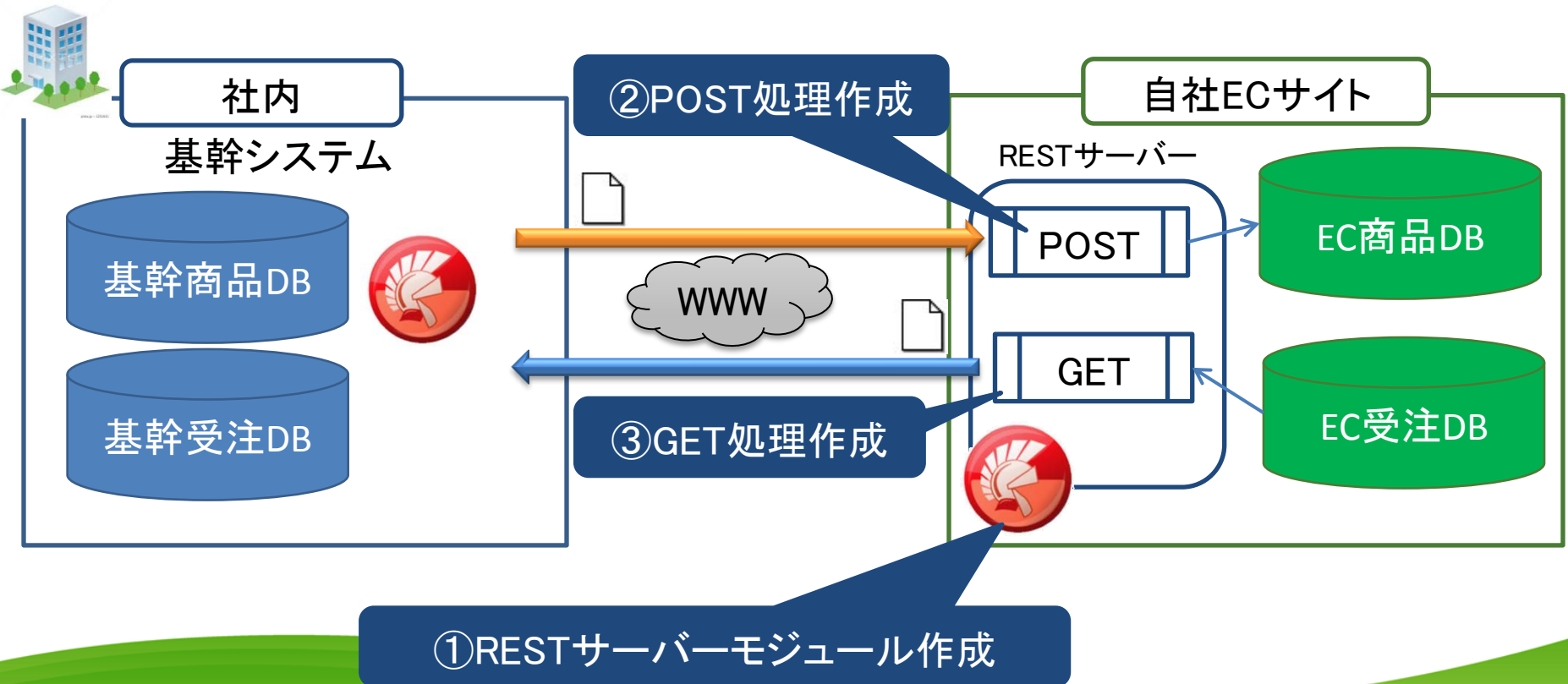
- 自社で構築したECサイトの場合、基幹システムや取引先と連携する為に、RESTサーバーアプリを構築することもできる。



2-5. RESTサーバーアプリ作成方法

構築の手順

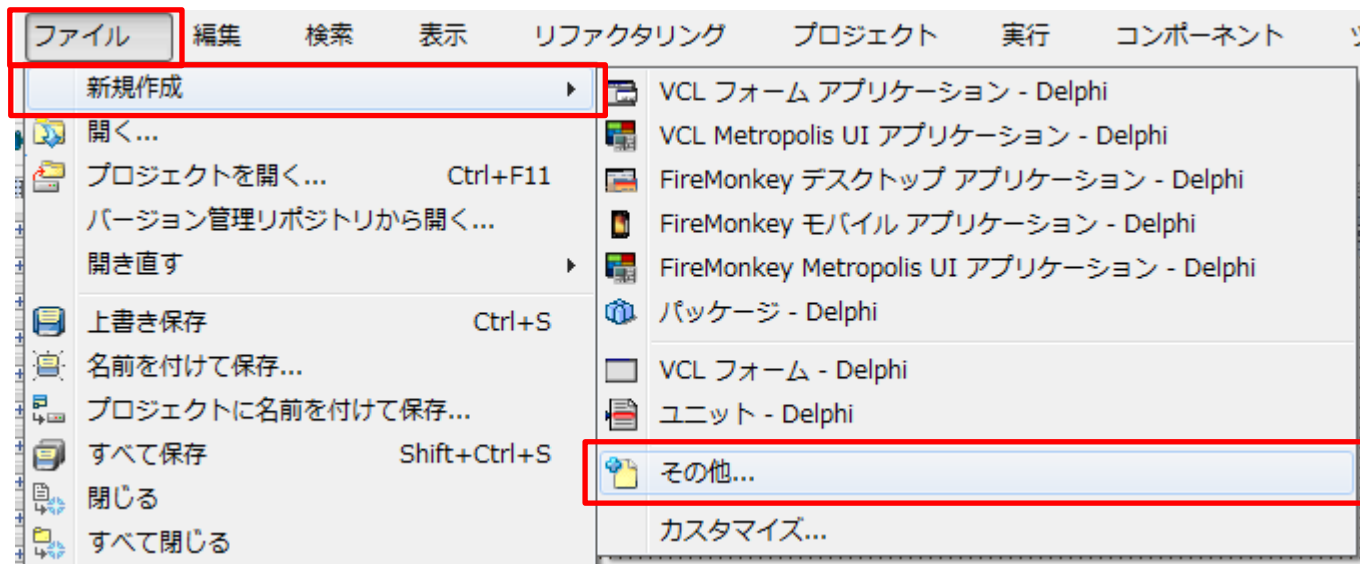
- ① RESTサーバーアプリケーションモジュールの作成
- ② POSTリクエストによるEDI商品データの受け取り
- ③ GETリクエストに対するEDI受注データの配信



2-5. RESTサーバーアプリ作成方法

① RESTサーバーアプリケーションモジュールの作成

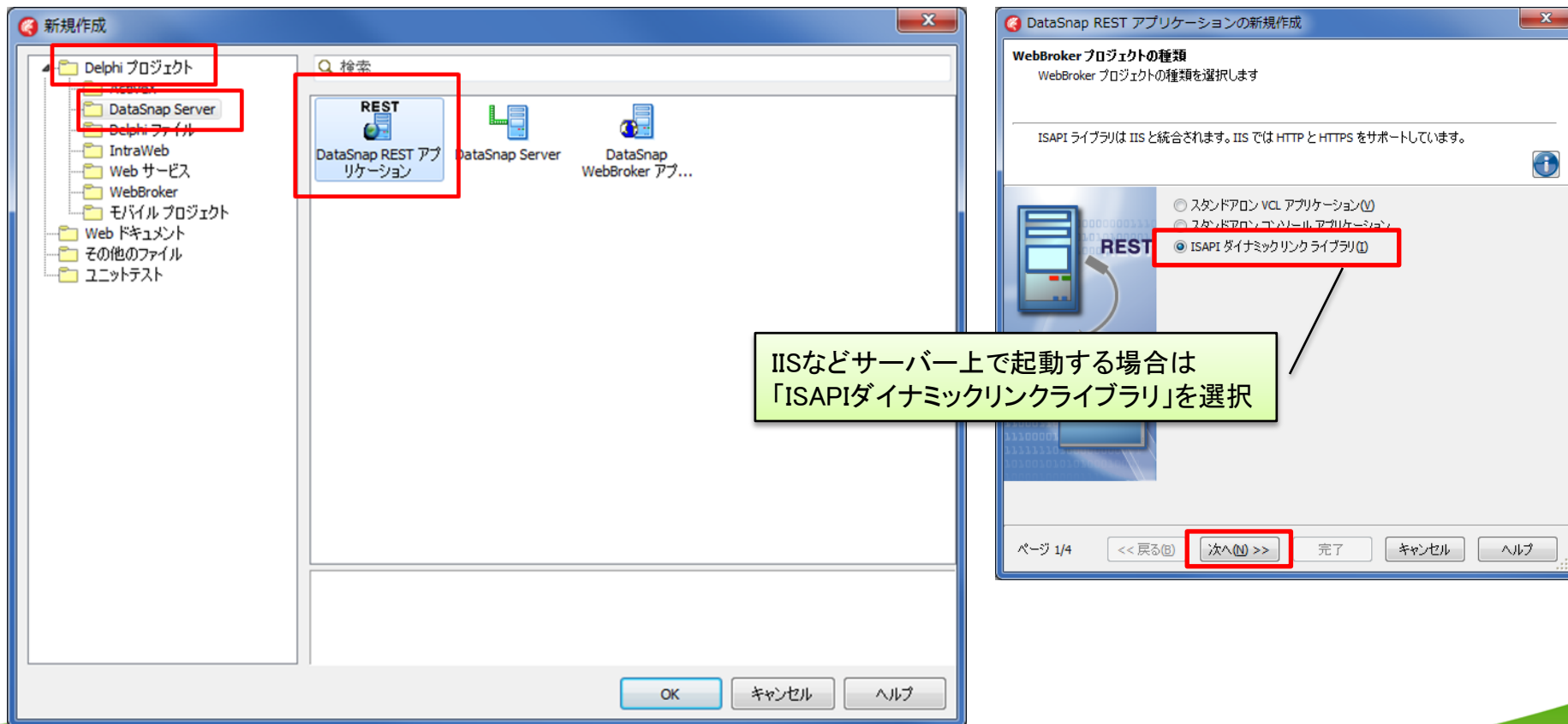
- DataSnapServerを使用し、モジュールを作成
 - [ファイル | 新規作成 | その他]を選択



2-5. RESTサーバーアプリ作成方法

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

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

The screenshot shows the 'DataSnap REST アプリケーションの新規作成' wizard. It is divided into several pages:

- Page 2/4:** 'サーバーの機能' (Server Capabilities). A list of features is shown, with 'REST' selected. A callout box states: 'DBよりデータ取得を行なうため、サーバーメソッドクラスの上位クラスには TDataModule を選択' (To retrieve data from the DB, select TDataModule as the base class for the server method class).
- Page 3/4:** 'プロジェクトの場所' (Project Location). A directory path is entered. A callout box says: 'ソースファイルの保管先を指定' (Specify the storage location for source files).
- Page 4/4:** '完了' (Finish). A callout box explains: '提供するAPI(メソッド)を追加する場合、「ServerMethodsUnit」にメソッドを追加 →次ページ以降で処理内容を記載' (When adding APIs (methods), add methods to 'ServerMethodsUnit' → record processing content from the next page onwards).

Code snippets from the 'ServerMethodsUnit1' unit are shown, illustrating the implementation of the 'EchoString' and 'ReverseString' methods within the 'TServerMethods1' class, which inherits from 'TDataModule'.

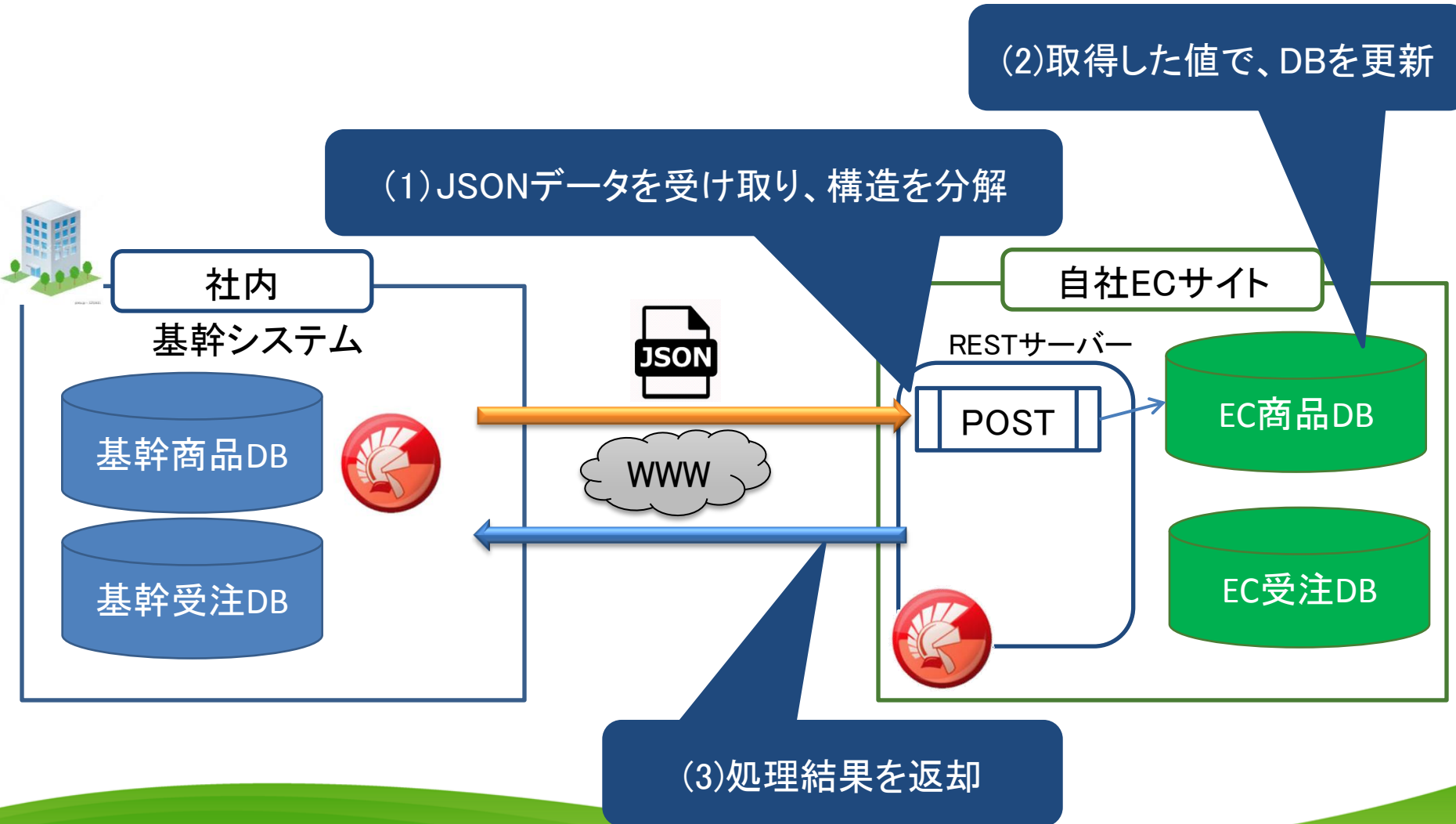
```

unit ServerMethodsUnit1;
interface
uses System.SysUtils, System.Classes, Datasnap;
type
{$METHODINFO ON}
TServerMethods1 = class(TDataModule)
private
    // private 宣言
public
    // public 宣言
function EchoString(Value: string): string;
function ReverseString(Value: string): string;
end;

```

2-5. RESTサーバーアプリ作成方法

② POSTリクエストによるEDI商品データの受け取り



(1) JSONデータを受け取り、構造を分解

(2) 取得した値で、DBを更新

(3) 処理結果を返却

2-5. RESTサーバーアプリ作成方法

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;
```

JSONの解析を行なうための変数を定義

```
begin
```


2-5. RESTサーバーアプリ作成方法

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

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);

```

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

```

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

```

キー情報(商品CD)

```
SQLQuery1.ExecSQL;
```

DBを更新

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

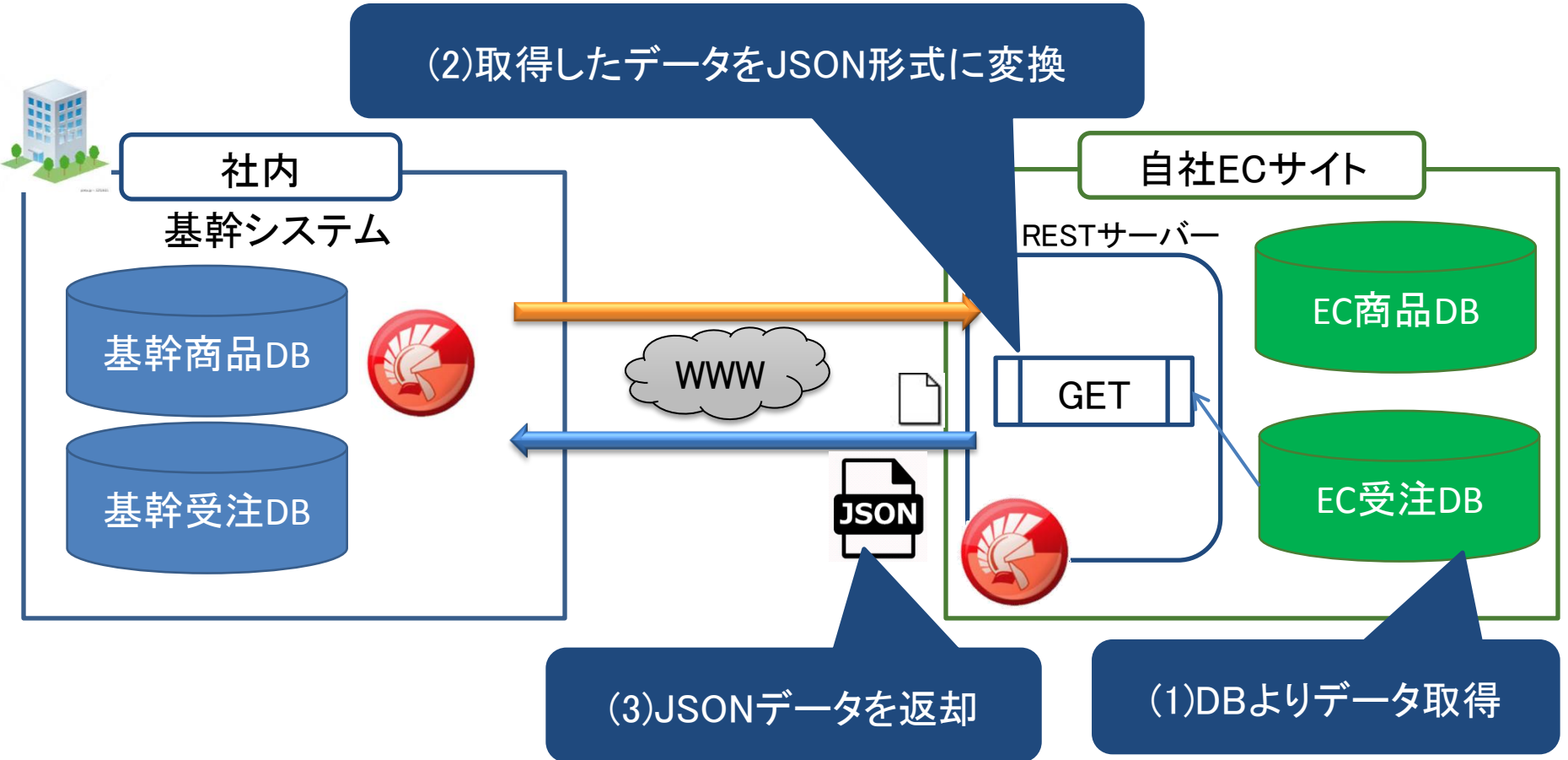
正しく更新出来た場合、'1'を返却

end;

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](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

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

- 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;
```

JSONの作成を行なうための変数を定義

```
begin  
SQLQuery1.SQL.Text := 'SELECT * FROM WJUCTRN';  
SQLQuery1.Open;
```

受注データを取得

2-5. RESTサーバーアプリ作成方法

- 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;
```

JSONデータ作成用オブジェクトを生成

```
try  
joOrder.AddPair('OrderDate', SQLQuery1.FieldName('JUJCDT').AsString);  
joOrder.AddPair('ShipToCode', SQLQuery1.FieldName('JUSPCD').AsString);  
joOrder.AddPair('PayeeCode', SQLQuery1.FieldName('JUSKCD').AsString);
```

各項目のJSONデータを作成

受注日

納入先CD

請求先CD

```
while not SQLQuery1.Eof do  
begin
```

明細データを作成

```
joLineItem := TJSONObject.Create;  
joLineItem.AddPair('ItemID', SQLQuery1.FieldName('JUSHCD').AsString);  
joLineItem.AddPair('UnitPrice', SQLQuery1.FieldName('JUTANK').AsString);  
joLineItem.AddPair('Quantity', SQLQuery1.FieldName('JUJUSU').AsString);
```

商品CD

単価

受注数

```
jaLineItems.Add(joLineItem);
```

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

- 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](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