WebOTX アプリケーション開発ガイド

WebOTX Enterprise Service Bus チュートリアル

対応バージョン： 6.5、7.1

版数： 初版

リリース： 2007 年 7 月

目次

1.ESB チュートリアル

1.1.チュートリアル（その1）

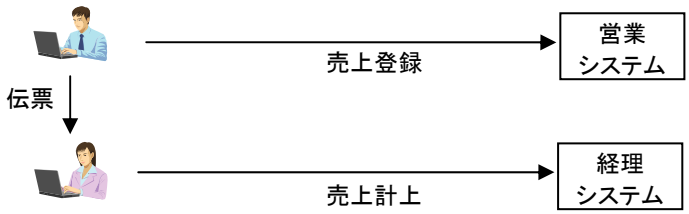
このチュートリアルでは、WebOTX Developer's Studio と WebOTX ESB を使ってサービス統合（システム統合）を実際に体験していただきます。今、ここに仮想の営業システムと経理システムがあるとして、それらを WebOTX ESB を使って統合することを例に、説明を進めていきます。WebOTX ESB を動作させるのに必要なサービスユニットやサービスアセンブリの作成に WebOTX Developer's Studio を使います。WebOTX Developer's Studio は、WebOTX ESB に搭載されているコンポーネントに対応するサービスユニットを簡単に作ることができます。また、WebOTX ESB に配備するサービスアセンブリも簡単に作成することができます。まずは、最初から説明の通りに操作して、開発の流れを理解してください。

このチュートリアルを実行する前に、WebOTX 開発環境(Windows 版)と WebOTX Enterprise Service Bus をインストールした環境を用意してください。

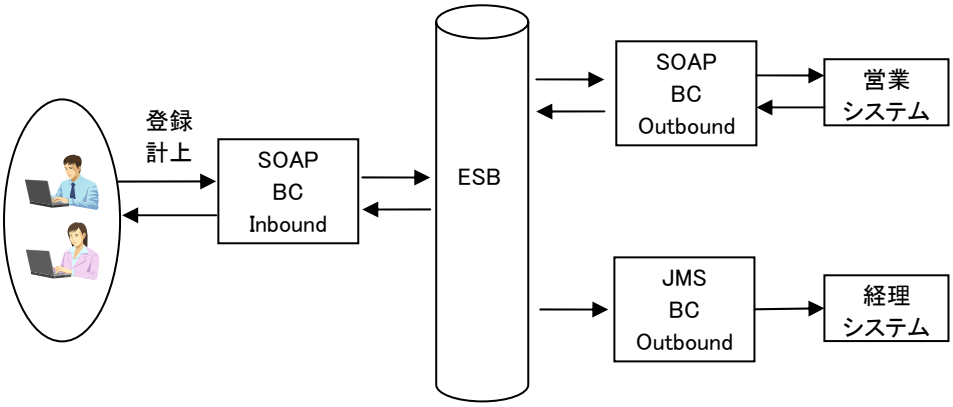
1.1.1.統合するサービス（システム）について

これから営業システムと経理システムという 2 つの別々なシステムを WebOTX ESB をつかって統合します。

システム統合されていないときのイメージ



WebOTX ESB による統合後のイメージ



図のように、システムが統合されていないときは、営業担当者が売上登録した結果の伝票を、経理担当者が手入力により個別に処理することになります。WebOTX ESB によりシステム統合すると、営業担当者も経理担当者も ESB に対してアクセスすることになり、各システムは ESB につながることになります。経理担当者は、伝票を受け取らなくても営業システムから売上データを検索し、経理システムに送ることが可能になります。

これからチュートリアルで作成する例では、ESB へアクセスする通信プロトコルを SOAP と仮定します。また、ESB から営業システムへの通信プロトコルを SOAP、経理システムへの通信プロトコルを JMS と仮定します。営業システムには、営業担当者により登録された売上データがあらかじめ登録されており、ID で検索すると該当する売上データを返します。また、経理システムは受け取った計上データ（売上データ）を自身の持つレジストリに書き込みます。

このチュートリアルでは、経理担当者が確認した売上データの ID を ESB に送り、ESB 側ではまず営業サーバからその ID によって該当する売上データを取得し、売上データを経理システムに送って計上処理を行うというシーケンスを実行します。

1.1.2.準備

まず、ESB につなげる 2 つのシステムとクライアントを準備します。各システム、クライアントのプログラムはすでに作成済みのものを提供しますので、ここではお手元の環境で動作させる作業を行っていただきます。

営業システム

営業システムは、売上データの検索を受け付けるシンプルな Web サービスです。Web アプリケーションとして実装されていますので、次の手順で Web コンテナに配備してください。

otxadmin コマンドを起動します。

```
> otxadmin
```

domain1 が起動していることを確認します。起動されていれば、「running」と表示されます。

```
otxadmin> list-domains
```

domain1 にログインします。domain1 は WebOTX インストール時に生成済みのドメインです。「ユーザ名」「パスワード」にはドメインログインに使用するユーザ名とパスワードを入力します。

```
otxadmin> login --user ユーザ名 --password パスワード --port 6212
```

「sales.war」を配備します。

```
otxadmin> deploy C:¥¥tmp¥¥sales.war
```

MEMO

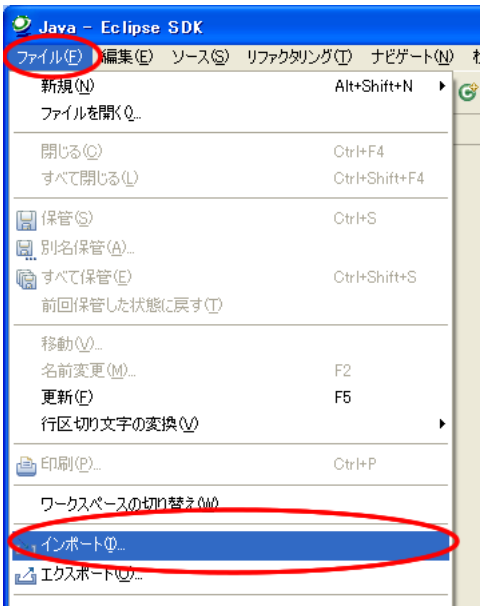
sales.war は、ESB の sample に含まれています。

経理システム

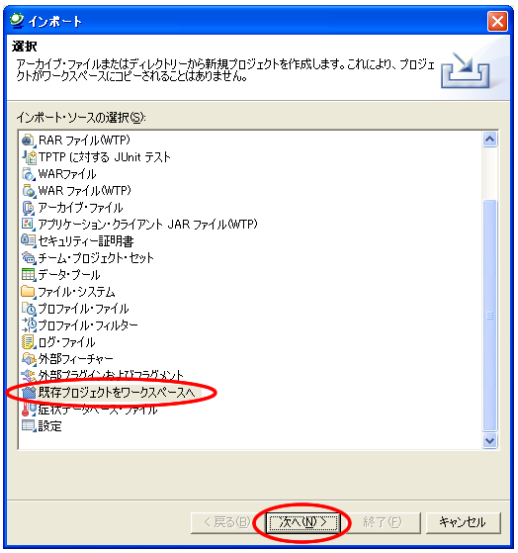
経理システムは、売上データの計上（経理システムが持つレジストリへの登録）を受け付けるシンプルな JMS クライアントで、WebOTX Developer's Studio で動作させる Java アプリケーションです。次の手順でプロジェクトをインポートして JMS クライアントの動作環境を作り、JMS サーバの設定、JMS クライアントの起動・メッセージ待ち受けを行います。

WebOTX Developer's Studio を起動します。**スタート**｜**すべてのプログラム**｜**WebOTX**｜**Developer's Studio** を起動します。

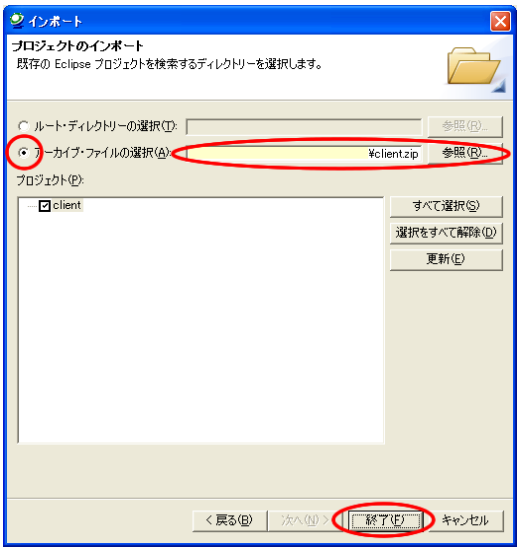
メニューから、**ファイル**｜**インポート**を実行します。



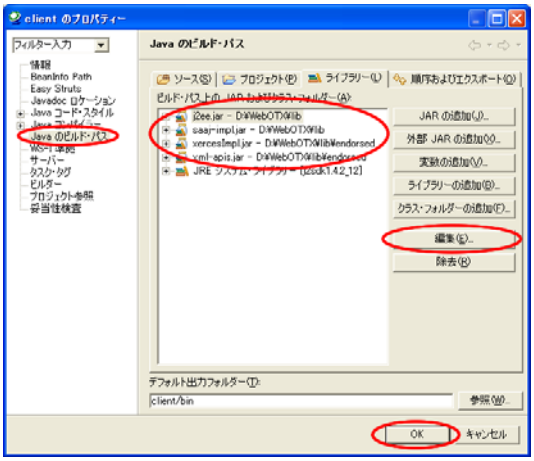
「**既存プロジェクトをワークスペースへ**」を選択し、[**次へ**]ボタンを押します。



ラジオボタンで「**アーカイブ・ファイルの選択**」を選択し、「**account.zip**」を指定し、[**終了**]ボタンを押します。Java パースペクティブのパッケージエクスプローラーで account プロジェクトが表示されることを確認します。



パッケージエクスプローラーで「**account**」のところにカーソルを置き、右クリックして「**プロパティ**」を実行します。「**Java のビルド・パス**」の「**ライブラリー**」タブで表示されるビルド・パスについて、WebOTX のインストールフォルダに合わせてパスの値を調整します。



続いて、JMS サーバの設定を行います。

otxadmin コマンドを起動します。

```
> otxadmin
```

domain1 が起動していることを確認します。起動していれば、「running」と表示されます。起動していない場合、domain1 を起動します。

```
otxadmin> list-domains
```

MEMO

account.zip は、ESB の sample に含まれています。

「パスワード」にはドメインログインに使用するユーザ名とパスワードを入力します。

```
otxadmin> login --user ユーザ名 --password パスワード --port 6212
```

使用する Queue の物理的送信先を otxadmin create-jmsdest コマンドで作成します。

```
otxadmin> create-jmsdest --desttype queue MyQueue
```

このクライアントアプリケーションから JNDI を通して Queue をルックアップするために必要なオブジェクトを作成します。

```
otxadmin> create-jms-resource --restype javax.jms.Queue --wojmsDestinationName=MyQueue
MyQueue
```

JMS サーバへの接続に必要な QueueConnectionFactory を作成します。QueueConnectionFactory も、クライアントアプリケーションから JNDI を通してキューをルックアップするために必要な管理オブジェクトを作成します。

```
otxadmin> create-jms-resource --restype javax.jms.QueueConnectionFactory MyQCF
```

同様に、応答用キューを以下のコマンドにより作成します。

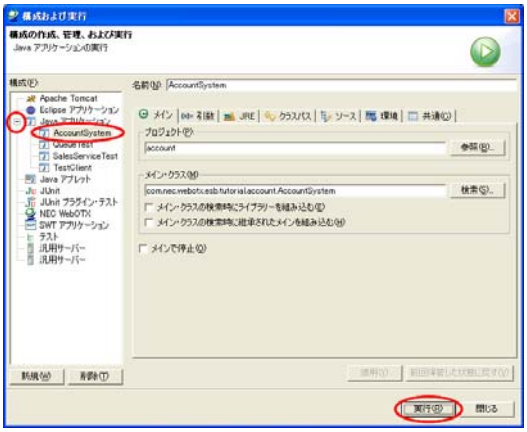
```
otxadmin> create-jmsdest --desttype queue JMSBCReplyQueue
```

```
otxadmin> create-jms-resource --restype javax.jms.Queue
--wojmsDestinationName=JMSBCReplyQueue jms/JMSBCReplyQueue
```

```
otxadmin> create-jms-resource --restype javax.jms.QueueConnectionFactory
jms/JMSBCReplyQueueConnectionFactory
```

続いて経理システムである JMS クライアントを実行し、メッセージの受信待機状態にします。

メニューから、**実行 | 構成および実行**を実行し、構成から **Java アプリケーション**配下の「**AccountSystem**」を選択し、[**実行**]ボタンを押します。



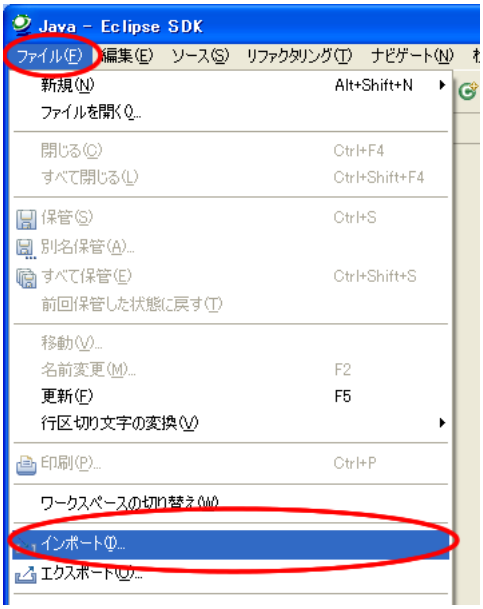
クライアント

ESB にアクセスするクライアントは、WebOTX Developer's Studio で動作させる Java アプリケーションです。次の手順でプロジェクトをインポートしてクライアントの動作環境を作ります。

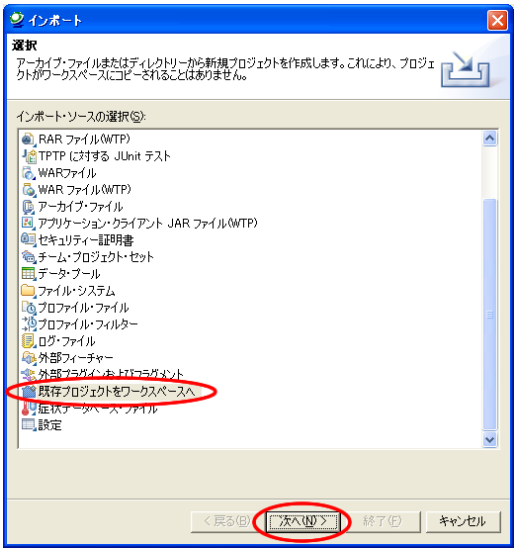
MEMO

<WebOTX_DIR>は WebOTX のインストールルートフォルダのことです。

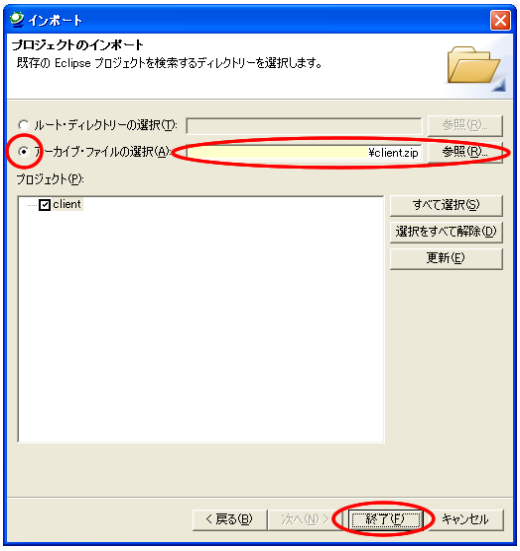
メニューから、**ファイル**｜**インポート**を実行します。



「**既存プロジェクトをワークスペースへ**」を選択し、[**次へ**]ボタンを押します。



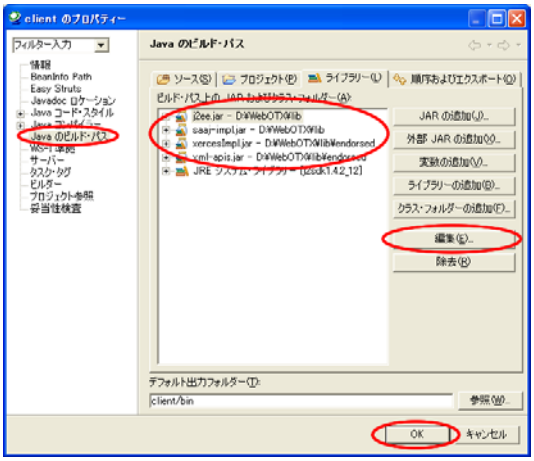
ラジオボタンで「**アーカイブ・ファイルの選択**」を選択し、「client.zip」を指定し、[**終了**]ボタンを押します。Java パースペクティブのパッケージエクスプローラーで client プロジェクトが表示されることを確認します。



MEMO

client.zip は、ESB の sample に含まれています。

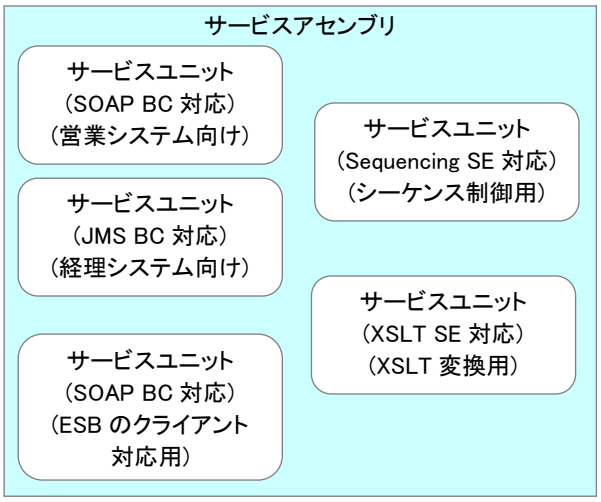
パッケージエクスプローラーで「**client**」のところにカーソルを置き、右クリックして「**プロパティ**」を実行します。「**Java のビルド・パス**」の「**ライブラリー**」タブで表示されるビルド・パスについて、WebOTX のインストールフォルダに合わせてパスの値を調整します。



1.2.チュートリアル（その2）

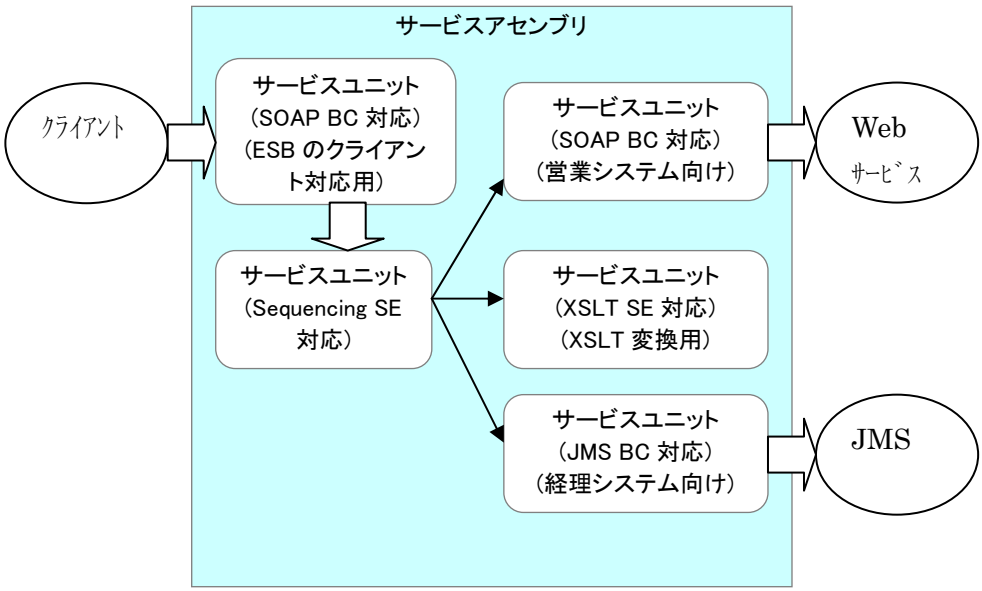
1.2.1.サービスアセンブリプロジェクトを作成する

WebOTX Developer’s Studio を起動し、サービスアセンブリの雛形を作成します。サービスアセンブリは、複数のサービスユニットをまとめて JBI コンテナに配備する単位です。サービスアセンブリプロジェクトは、これから作成するいくつかのサービスユニットをまとめるために使用するプロジェクトです。



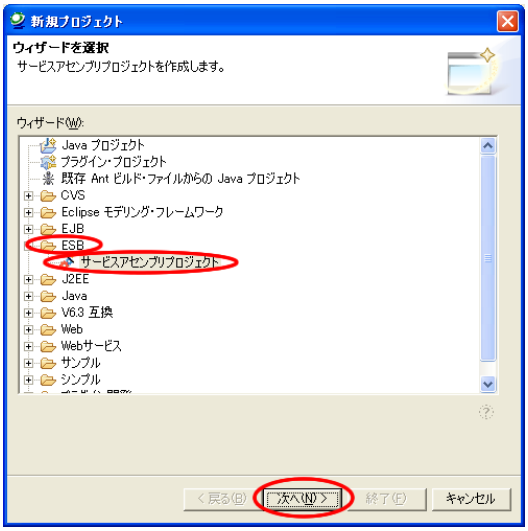
サービスアセンブリ内の各サービスユニットは、以下のように連携動作します。

クライアントからの要求を受けた SOAP　BC は、Sequencing SE を呼び出します。Sequencing SE は、外部 Web サービスを呼び出す SOAP BC を呼び出し、Web サービスからの応答を受け取ります。
Sequencing SE は、Web サービスからの応答を、XSLT SE を使用して JMS が期待するレイアウトに変換します。Sequencing SE は JMS BC を呼び出します。

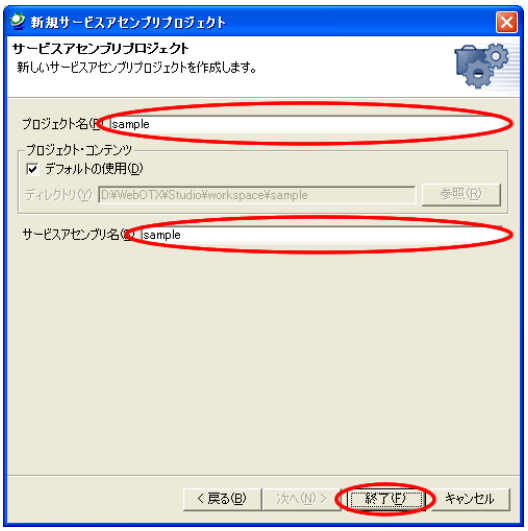


メニューから**ファイル**｜**新規**｜**プロジェクト**を選択して、**新規プロジェクト**ダイアログを表示させます。

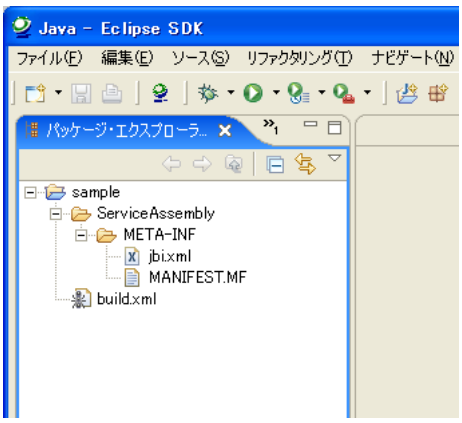
新規プロジェクト画面で **ESB**｜**サービスアセンブリプロジェクト**を選択し、[**次へ**]ボタンを押します。



プロジェクト名に「**sample**」、サービスアセンブリ名に「**sample**」を指定し、[**終了**]ボタンを押します。



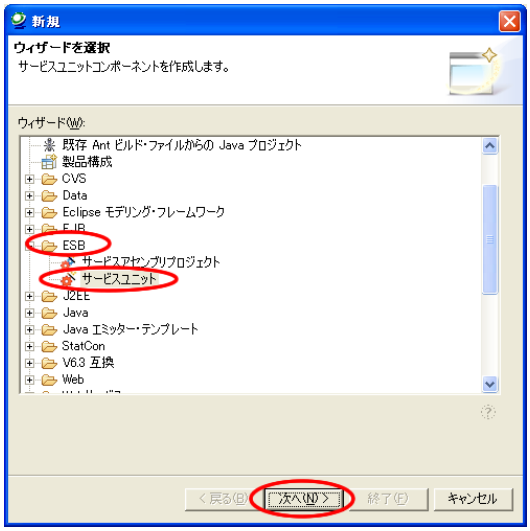
パッケージエクスプローラーで、sample プロジェクトが追加されたことを確認します。



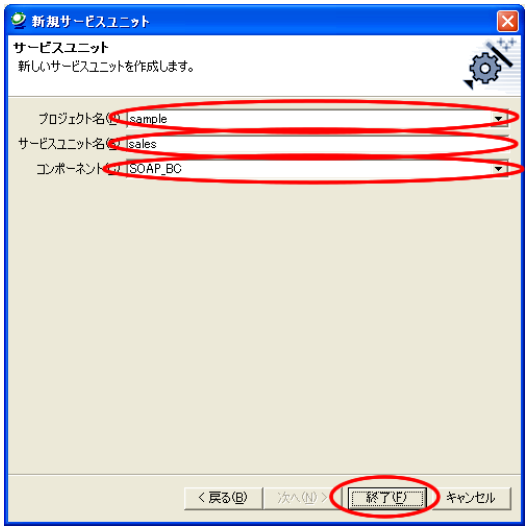
1.2.2.営業システム用のサービスユニットを追加する

メニューから、**ファイル**｜**新規**｜**その他**を選択します。

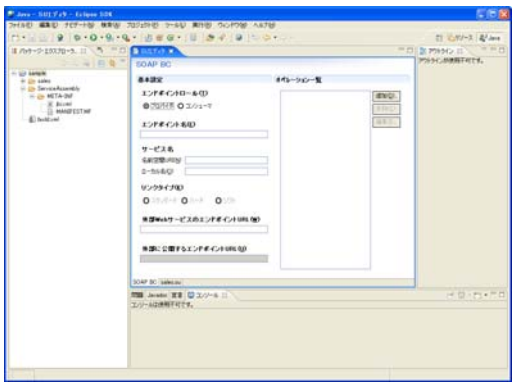
新規画面で **ESB**｜**サービスユニット**を選択し[**次へ**]ボタンを押します。



新規サービスユニット画面で、プロジェクト名で「**sample**」を選択し、サービスユニット名に「**sales**」を指定し、コンポーネントで「**SOAP_BC**」を選択し、[**終了**]ボタンを押します。



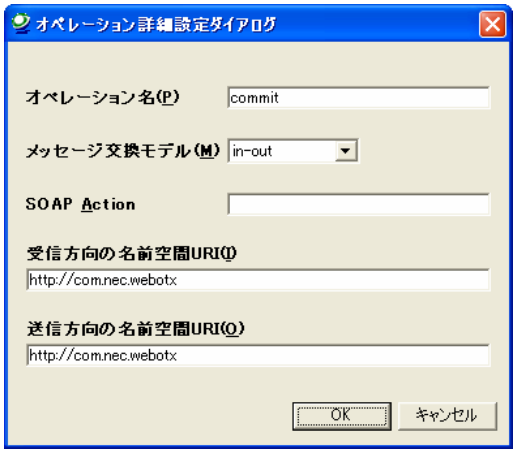
sample プロジェクトに sales フォルダが作成され、SOAP BC 用の SU エディタが開いていることを確認します。



SU エディタで、次の値を設定します。

設定名	設定値	備考
エンドポイント名	sales	
エンドポイントロール	プロバイダ	
サービス名 （名前空間 URI）	http://com.nec.webotx	
サービス名 （ローカル名）	sales	
外部 Web サービスのエンドポイント URI	http://localhost/sales/SalesService	ホスト名、ポート番号は sales.war を配備した場所に合わせます。

オペレーション一覧にオペレーションを追加します。「**オペレーション一覧**」の[**追加**]ボタンをクリックし、表示されたダイアログで、次のように設定を行い、[**OK**]ボタンを押します。



設定名	設定値	備考
オペレーション名	commit	
メッセージ交換モデル	in-out	
SOAP Action		空欄
受信方向の名前空間 URI	http://com.nec.webotx	
送信方向の名前空間 URI	http://com.nec.webotx	

もし、営業システムの Web サービスが WSDL を公開していれば、その内容を参照して SU エディタで設定を行います。以下に営業システムの WSDL と SU エディタでの設定との対応を示します。

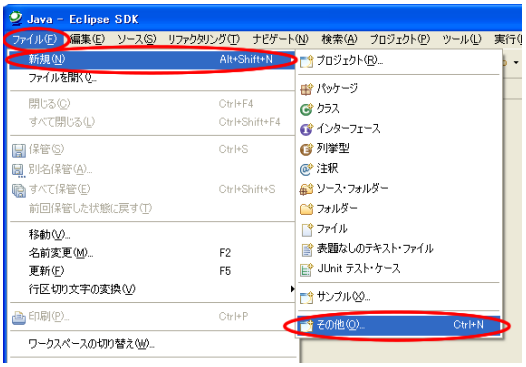
ここでは、すべての値を営業システムの Web サービスに合わせて設定する必要があります。もし、営業システムの Web サービスが WSDL を公開していれば、その内容を参照します。先に示した WSDL と対応させた場合、次のようになります。

設定名	WSDL 該当箇所	説明
外部 Web サービスのエンドポイント URI	①	
オペレーション名	②	オペレーション名は GUI 表示上の便宜的な名称です。ここに設定する値は、営業システムの Web サービスに送る SOAP メッセージの Body 要素直下の子要素の要素名です。
メッセージ交換モデル	③	input 要素のみ存在するときは in-only、input と fault 要素のみ存在するときは robust-in-only、input と output 要素が存在する場合は in-out を選択します。
SOAP Action	④	この例では空なので記入しませんが、そうでない場合はそのまま転記します。
受信方向の名前空間 URI	⑤	レスポンスメッセージの SOAP Body 直下の要素に修飾している名前空間 URI を指定します。
送信方向の名前空間 URI	⑥	リクエストメッセージの SOAP Body 直下の要素に修飾している名前空間 URI を指定します。

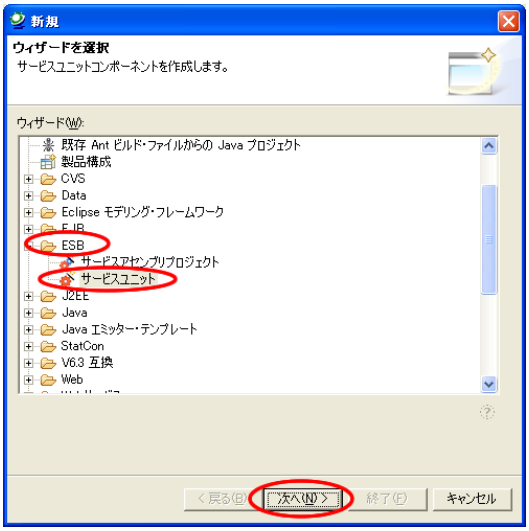
SU エディタの設定を保存して編集画面を閉じます。

1.2.3.経理システム用のサービスユニットを追加する

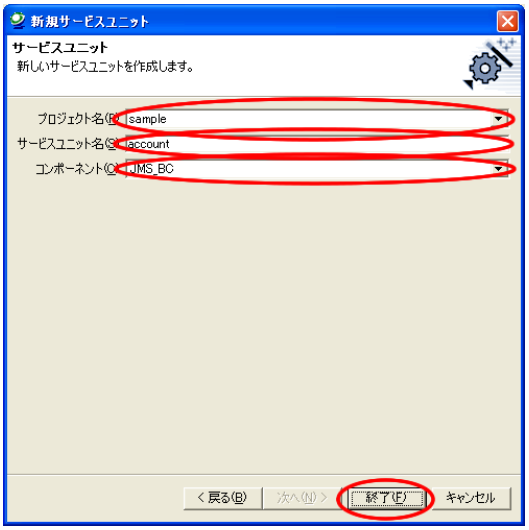
メニューから、**ファイル**｜**新規**｜**その他**を選択します。



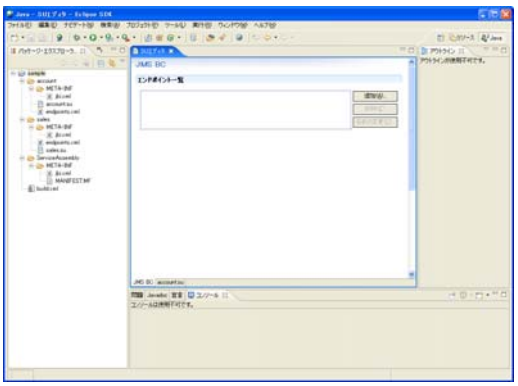
新規画面で **ESB**｜**サービスユニット**を選択し[**次へ**]ボタンを押します。



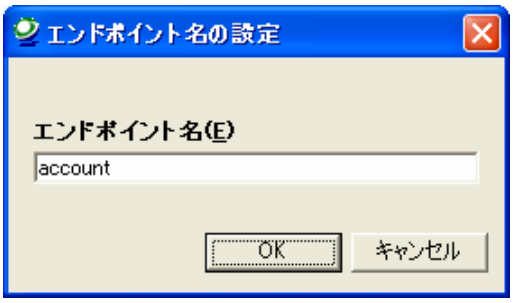
新規サービスユニット画面で、プロジェクト名で「**sample**」を選択し、サービスユニット名に「**account**」を指定し、コンポーネントで「**JMS_BC**」を選択し、[**終了**]ボタンを押します。



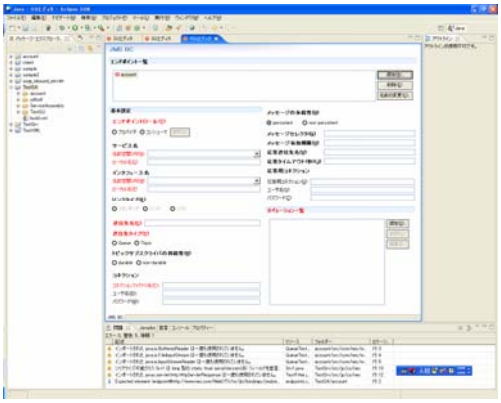
sample プロジェクトに account フォルダが作成され、JMS BC 用の SU エディタが開いていることを確認します。



「**エンドポイント一覧**」の「**追加**」をクリックして表示されるダイアログでエンドポイント名「account」を入力して「**OK**」をクリックします。



エンドポイントのパラメータの入力フォームが表示されます。次の値を設定します。

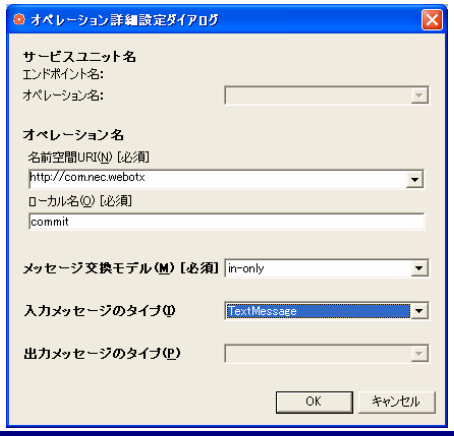


設定名	設定値	備考
エンドポイントロール	プロバイダ	
サービス名 （名前空間 URI）	http://com.nec.webotx	
サービス名 （ローカル名）	account	
インタフェース名 （名前空間 URI）	http://com.nec.webotx	
インタフェース名 （ローカル名）	account	
送信先名	MyQueue	経理システムとのやり取りに使うキューの JMS リソース作成時に指定した JNDI 名を指定します。
送信先タイプ	Queue	経理システム側は単数の JMS クライアントなので Queue を選択します。
コネクション （コネクションファクトリ名）	MyQCF	JMS のコネクションファクトリ名を指定します。
コネクション （ユーザ名）		空欄です。
コネクション （パスワード）		空欄です。
メッセージの永続性	persistent	
メッセージセレクタ		空欄です。
メッセージ有効期限		空欄です。
応答送信先		空欄です。

MEMO

表示されない場合はエンドポイント一覧に追加されるエンドポイント名をクリックしてください。

オペレーション一覧にオペレーションを追加します。「**オペレーション一覧**」の[**追加**]ボタンをクリックし、表示されたダイアログで次のように設定し、[**OK**]ボタンを押します。



設定名	設定値	備考
オペレーション名（名前空間 URI）	http://com.nec.webotx	
オペレーション名（ローカル名）	commit	
メッセージ交換モデル	in-only	経理システムは非同期でメッセージを受け取るだけという仕様なので in-only を選択します。
入力メッセージのタイプ	TextMessage	ノーマライズメッセージのコンテント（in 側）の内容を JMS で送るので TextMessage を選択します。BytesMessage は添付ファイルを送る場合に選択します。

SU エディタの設定を保存して編集画面を閉じます。

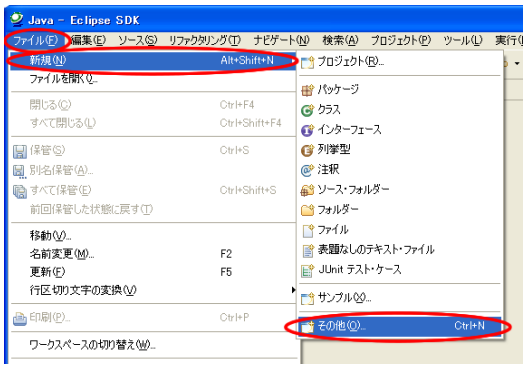
MEMO

JMS BC は、ノーマライズメッセージのコンテントの内容（XML）を TextMessage として送信します。また、添付ファイルは BytesMessage として byte 配列で送信します。
従って、経理システムは TextMessage の形式で送られてくる XML メッセージを処理できるアプリケーションになっています。

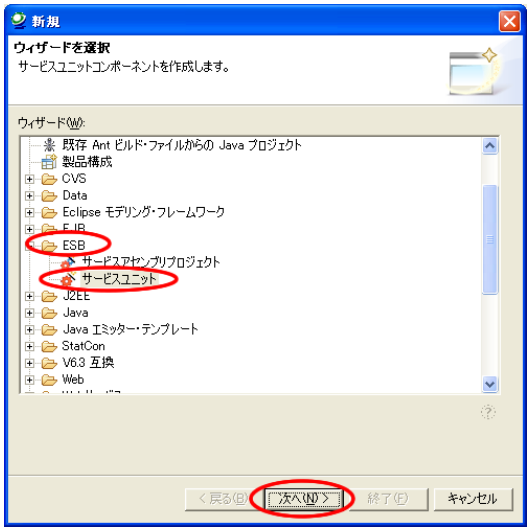
1.3.チュートリアル（その3）

1.3.1.XSL 変換のサービスユニットを追加する

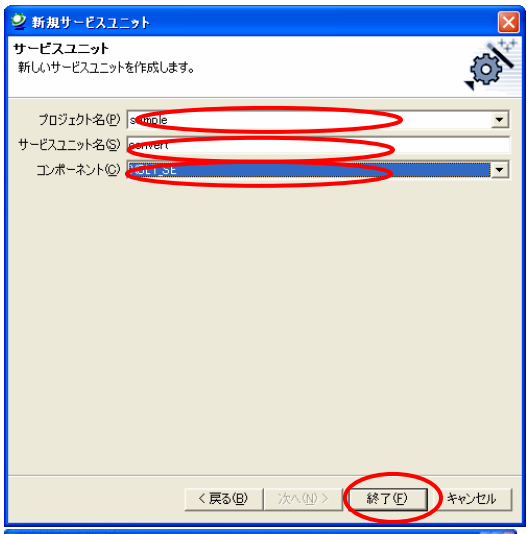
メニューから、**ファイル**｜**新規**｜**その他**を選択します。



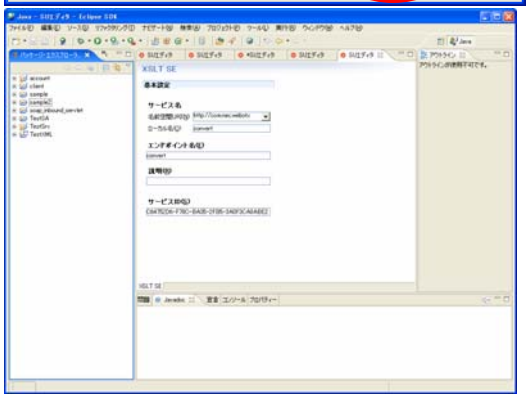
新規画面で **ESB**｜**サービスユニット**を選択し[**次へ**]ボタンを押します。



新規サービスユニット画面で、プロジェクト名で「**sample**」を選択し、サービスユニット名に「**convert**」を指定し、コンポーネントで「**XSLT_SE**」を選択し、[**終了**]ボタンを押します。



sample プロジェクトに convert フォルダが作成され、XSLT SE 用の SU エディタが開いていることを確認します。



SU エディタで、次の値を設定します。

設定名	設定値	備考
サービス名 （名前空間 URL）	http://com.nec.webotx	
サービス名 （ローカル名）	convert	
エンドポイント名	convert	
説明		空欄
サービス ID		規定値を使用する

XSLT SE では、外部 Web サービスから受け取った応答を、JMS の外部サービス認識できる形式に変換します。XSLTSE を使用する場合は、XSL 変換を行うための xslt を定義する必要があります。

本サンプルの、外部 Web サービスは、以下の XML イメージの応答を返却します。

```
<otx:commitResponse xmlns:otx="http://com.nec.webotx">
<ProductData>
  <id>1</id>
  <registDate>
    <year>2006</year>
    <month>7</month>
    <date>13</date>
    <hour>9</hour>
    <minute>10</minute>
    <second>11</second>
  </registDate>
  <deliveryDate>
    <year>2006</year>
    <month>7</month>
    <date>18</date>
    <hour>8</hour>
    <minute>30</minute>
    <second>0</second>
  </deliveryDate>
  <registerName>日電太郎</registerName>
  <productNumber>1</productNumber>
  <productName>バザールでござーるマウスパッド</productName>
  <price>1500</price>
  <suryo>1000</suryo>
</ProductData>
</otx:commitResponse>
```

また、JMS サービスは、以下の XML を期待しています。

```
<otx:commit xmlns:otx="http://com.nec.webotx">
<ProductData>
  <id>1</id>
  <productNumber>1</productNumber>
  <productName>バザールでござーるマウスパッド</productName>
  <price>1500</price>
  <suryo>1000</suryo>
  <shoukei>1500000</shoukei>
  </ProductData>
</otx:commit>
```

上記の XML 間のフォーマット変換を行うための、XSLT ファイルを作成します。本変換を行う XSLT ファイルは以下のようになります。convert 配下の temp.xslt に以下の記述を追加してください。

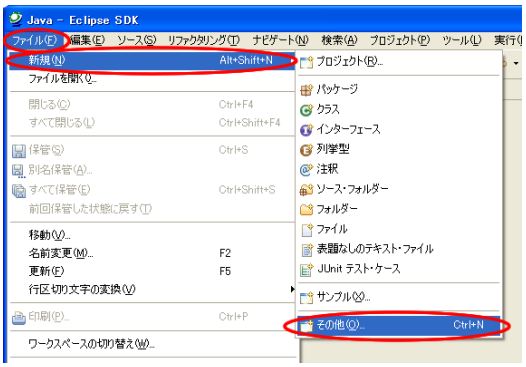
```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:template match="/*[local-name()=&quot;commitResponse&quot; and
namespace-uri()=&quot;http://com.nec.webotx&quot;]">
    <otx:commit xmlns:otx="http://com.nec.webotx">
      <xsl:apply-templates select="ProductData"/>
    </otx:commit>
  </xsl:template>
  <xsl:template match="ProductData">
    <ProductData>
    <id><xsl:value-of select="id"/></id>
    <productNumber><xsl:value-of select="productNumber"/></productNumber>
    <productName><xsl:value-of select="productName"/></productName>
    <price><xsl:value-of select="price"/></price>
    <suryo><xsl:value-of select="suryo"/></suryo>
    <xsl:param name="price">
      <xsl:value-of select="price"/>
    </xsl:param>
    <xsl:param name="suryo">
      <xsl:value-of select="suryo"/>
    </xsl:param>
    <shoukei><xsl:value-of select="$price*$suryo"/></shoukei>
    </ProductData>
  </xsl:template>
</xsl:stylesheet>
```

SU エディタの設定を保存して編集画面を閉じます。

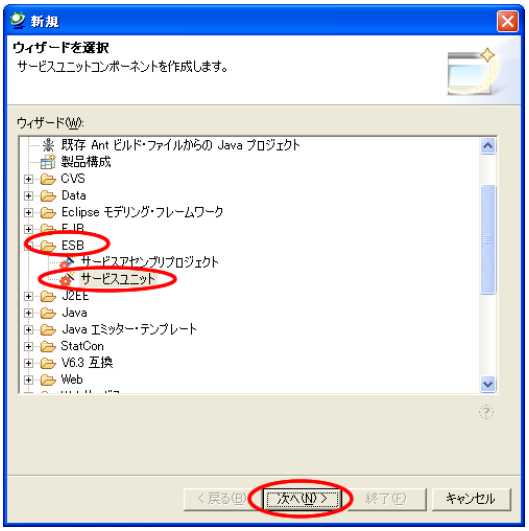
1.3.2.シーケンスを設定する

シーケンスを設定して、ESB に統合しているサービス（システム）の実行順序が「クライアントに公開したエンドポイント→営業システム→経理システム」となるように指定します。

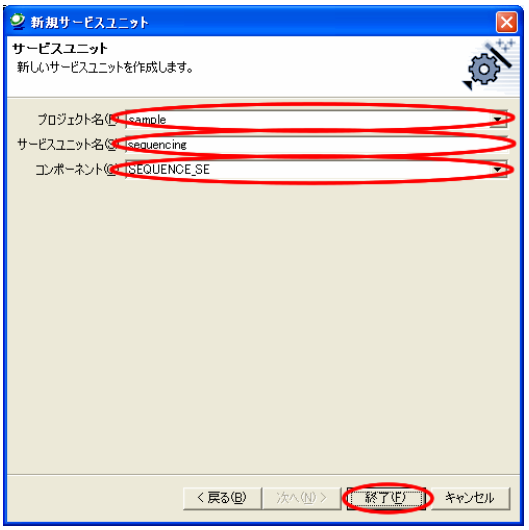
メニューから、**ファイル**｜**新規**｜**その他**を選択します。



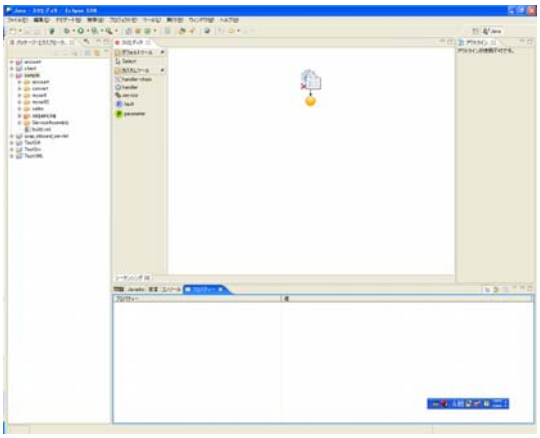
新規画面で **ESB**｜**サービスユニット**を選択し[**次へ**]ボタンを押します。



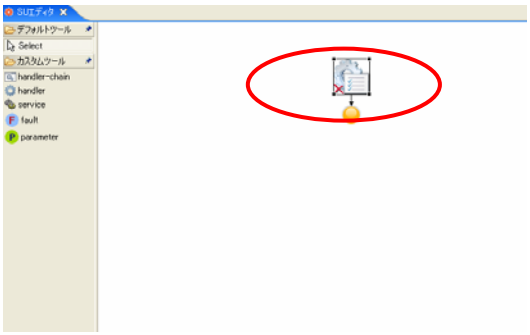
新規サービスユニット画面で、プロジェクト名で「**sample**」を選択し、サービスユニット名に「**sequencing**」を指定し、コンポーネントで「**SEQUENCE_SE**」を選択し、[**終了**]ボタンを押します。



sample プロジェクトに sequencing フォルダが作成され、シーケンシング SE 用の SU エディタが開いていることを確認します。

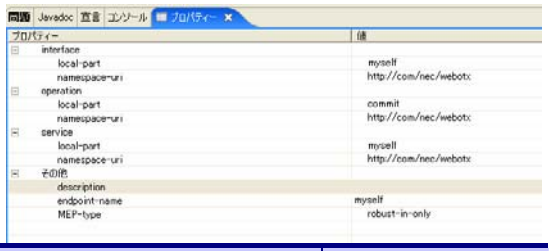


はじめに、シーケンスの開始部分となるクライアントに公開するエンドポイントについて設定します。SU エディタの左側のツール選択欄で「**Select**」を選択し、キャンバスの上部に表示されているオブジェクト（list-attributes オブジェクト）を選択します。すると、SU エディタの下のビューでプロパティシートが開きます。



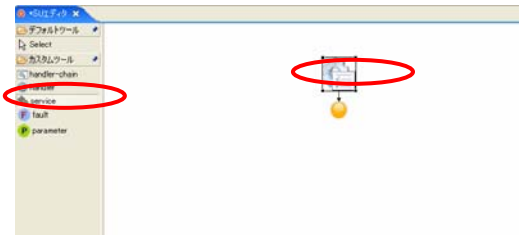
プロパティシートが表示されない場合は Eclipse のビューをプロパティに切り替えます。

プロパティシートは次の内容で設定します。

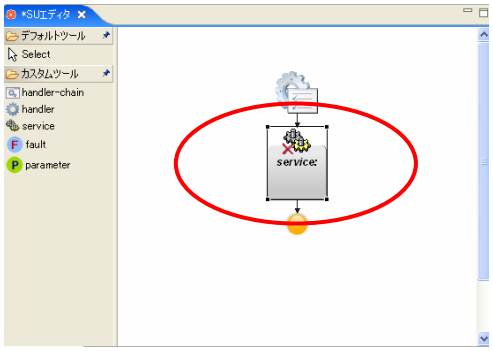


設定名		設定値	備考
interface	local-part	myself	
	namespace-uri	http://com.nec.webotx	
Operation	local-part	commit	
	namespace-uri	http://com.nec.webotx	
service	local-part	myself	
	namespace-uri	http://com.nec.webotx	
その他	description		空欄で構いません。
	endpoint-name	myself	
	MEP-type	robust-in-only	

次に、営業システムへのシーケンスを追加します。ツールバーの「**service**」を選択した後、list-attributes オブジェクトと end オブジェクトの間をクリックします。

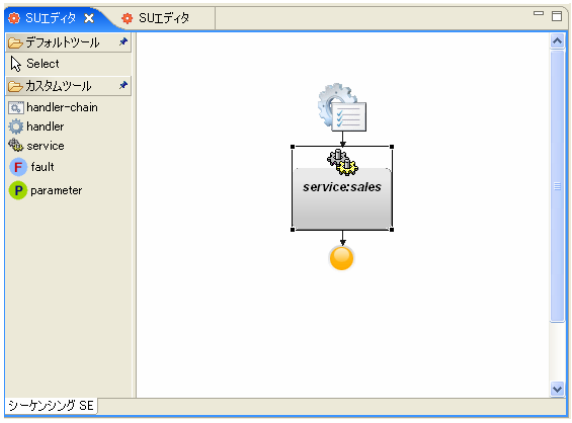


サービスのオブジェクトが生成されます。生成されたサービスオブジェクトをダブルクリックします。



営業システムの呼び出しを行うサービスユニット(**sales**)を選択し、エンドポイント(**sales**)オペレーション(**commit**)を選択し、OK を押下します。

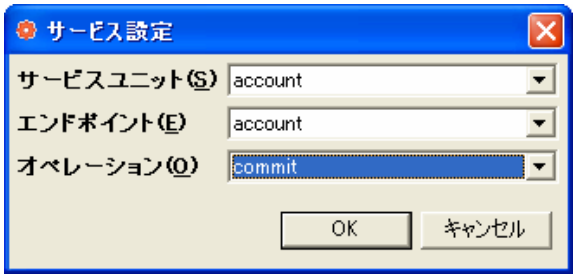
サービスを呼び出すための設定がプロパティシートに追加されます。



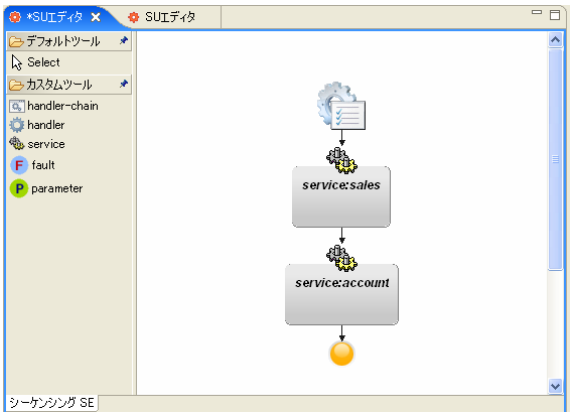
設定名		設定値	備考
interface	local-part	sales	
	namespace-uri	http://com.nec.webotx	
Operation	local-part	commit	
	namespace-uri	commit	
service	local-part	sales	
	namespace-uri	http://com.nec.webotx	
その他	description		空欄で構いません。
	endpoint-name	sales	
	MEP-type	in-out	

経理システムへのシーケンスも、同様に追加します。

追加したサービスを選択し、サービスユニット(account)エンドポイント(account)オペレーション(commit)を選択し。OK ボタンを押下します。

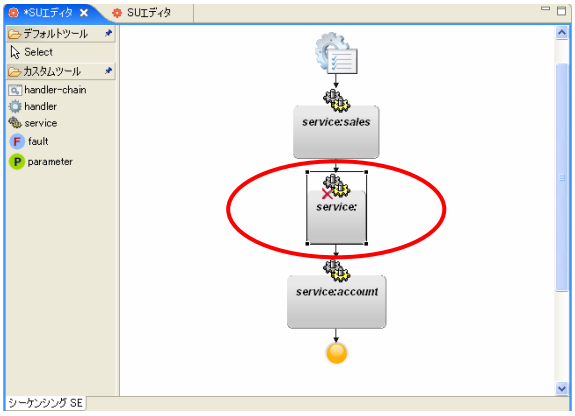


サービスを呼び出すための設定がプロパティシートに追加されます。



設定名		設定値	備考
interface	local-part	account	
	namespace-uri	http://com.nec.webotx	
Operation	local-part	commit	
	namespace-uri	http://com.nec.webotx	
service	local-part	account	
	namespace-uri	http://com.nec.webotx	
その他	description		空欄で構いません。
	endpoint-name	account	
	MEP-type	in-only	

営業システム(sales)からの応答データを経理システム(account)に通知するためのデータに変換するために XSLT SE のサービスを定義します。同様の手順で XSLT SE のサービスを **sales** と **account** の間に配置します。



追加したサービスを選択し、サービスユニット(cconvert)エンドポイント(convert)を選択し、OK ボタンを押下します。

設定名		設定値	備考
interface	local-part	convert	
	namespace-uri	http://com.nec.webotx	
Operation	local-part	convert	
	namespace-uri	http://com.nec.webotx	
service	local-part	convert	

	namespace-uri	http://com.nec.webotx	
その他	description		空欄で構いません。
	endpoint-name	convert	
	MEP-type	in-out	

SU エディタの設定を保存して編集画面を閉じます。

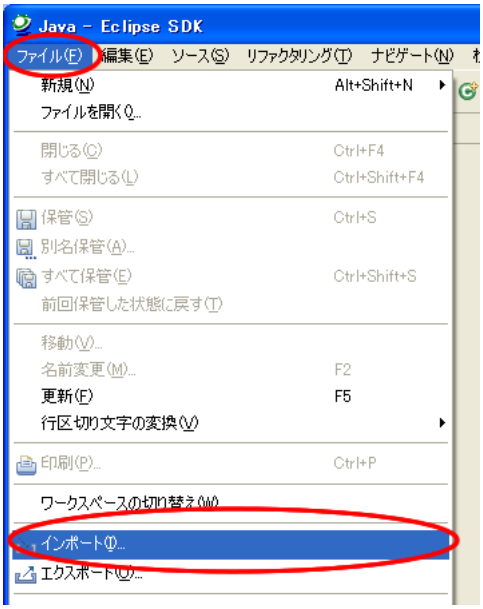
1.4.チュートリアル（その4）

1.4.1.クライアントに公開するエンドポイントを作成する

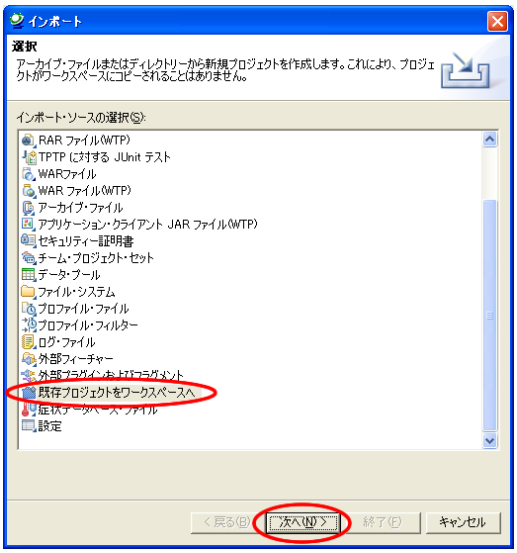
ESB で統合した 1 つのサービスをクライアントに公開するエンドポイントを作成します。まず、SOAP BC の inbound に必要な HTTP リスナの登録を行い、続けてサービスユニットを追加します。

まず、SOAP BC の inbound に必要な HTTP リスナの登録を行います。HTTP リスナはサーブレットとして実装された Web アプリケーションとして、Developer’s Studio の Web プロジェクトの形式で提供されています。ここでは、URL を設定し、WAR ファイルを作成し、Web コンテナに配備します。

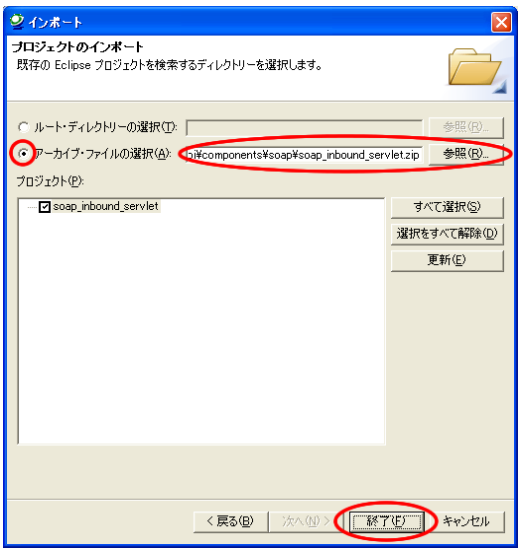
メニューから、**ファイル**｜**インポート**を実行します。



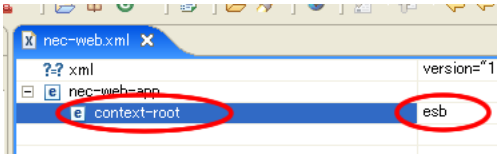
「**既存プロジェクトをワークスペースへ**」を選択し、[**次へ**]ボタンを押します。



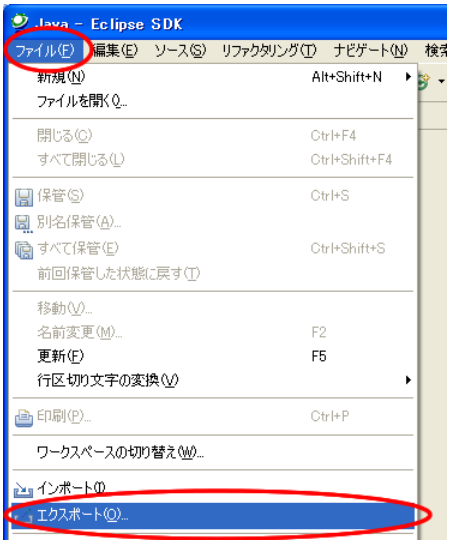
ラジオボタンで「**アーカイブ・ファイルの選択**」を選択し、
「<WebOTX_DIR>/jbi/components/soap/soap_inbound_servlet.zip」
を指定し、[**終了**]ボタンを押します。Java パースペクティブのパッケージエクスプローラーで soap_inbound_servlet プロジェクトが表示されることを確認します。



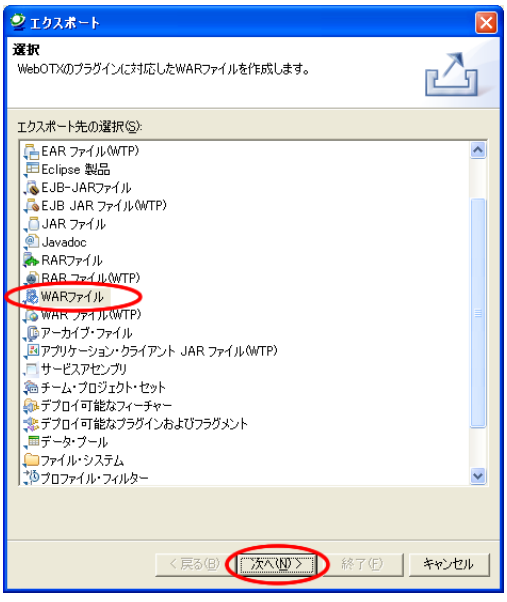
パッケージエクスプローラーで
/soap_inbound_servlet/WEB-INF/nec-web.xml をダブルクリックして編集画面に開き、**context-root** 要素の値「soap」を「**esb**」に変更し、編集結果を保存します。



メニューから、**ファイル**｜**エクスポート**を実行します。



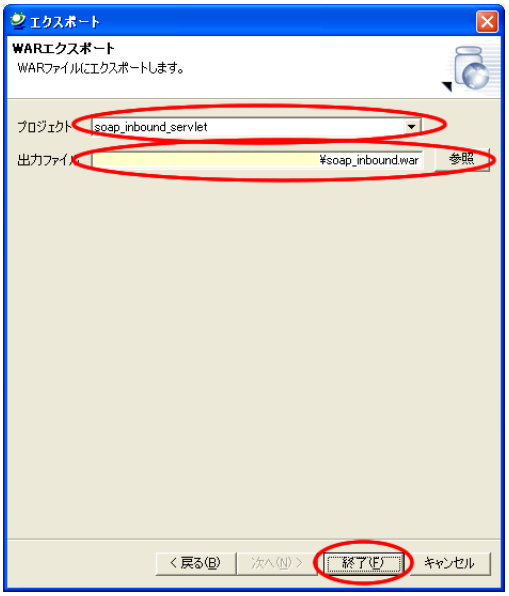
「**WAR ファイル**」を選択し、[**次へ**]ボタンを押します。



MEMO

<WebOTX_DIR>は WebOTX のインストールルートフォルダのことです。

プロジェクトは「**soap_inbound_servlet**」を選択し、出力ファイルに「**<任意のパス>¥soap_inbound.war**」を設定し、[**終了**]ボタンを押します。



続いて WAR ファイルを Web コンテナに配備します。

otxadmin コマンドを起動します。

```
> otxadmin
```

domain1 が起動していることを確認します。起動していれば、「running」と表示されます。起動していない場合は、domain1 を起動します。

```
otxadmin> list-domains
```

domain1 にログインします。domain1 は WebOTX インストール時に生成済みのドメインです。「ユーザ名」「パスワード」にはドメインログインに使用するユーザ名とパスワードを入力します。

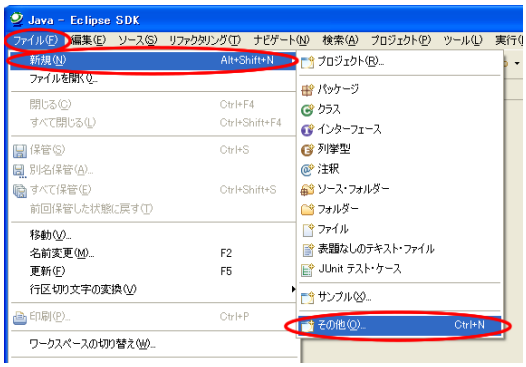
```
otxadmin> login --user ユーザ名 --password パスワード --port 6212
```

WAR ファイルを配備します。

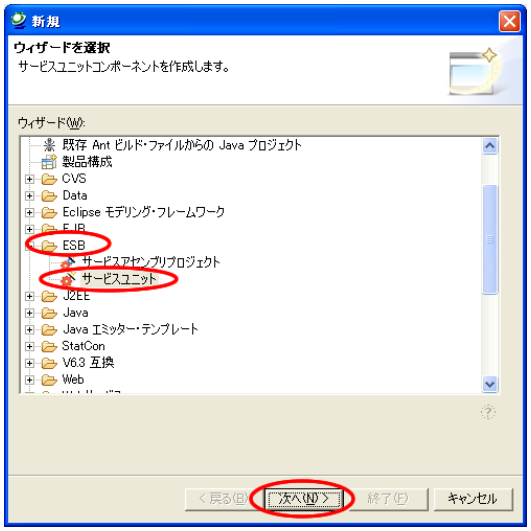
```
otxadmin> deploy <soap_inbound.war までのパス>¥¥soap_inbound.war
```

次にサービスユニットの追加を行います。

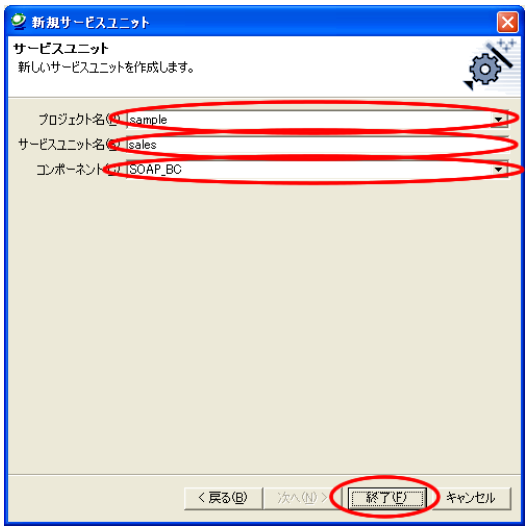
メニューから、**ファイル**｜**新規**｜**その他**を選択します。



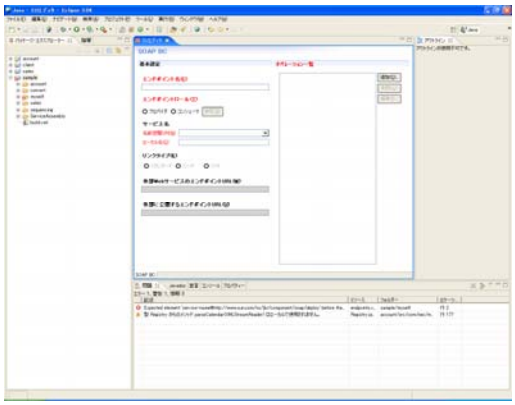
新規画面で **ESB｜サービスユニット**を選択し[**次へ**]ボタンを押します。



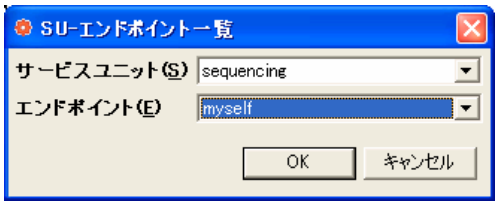
新規サービスユニット画面で、プロジェクト名で「**sample**」を選択し、サービスユニット名に「**myself**」を指定し、コンポーネントで「**SOAP_BC**」を選択し、[**終了**]ボタンを押します。



sample プロジェクトに myself フォルダが作成され、SOAP BC 用の SU エディタが開いていることを確認します。コンシューマを選択し、参照ボタンを押下します。



サービスユニット(sequencing)エンドポイント(myself)を選択します。

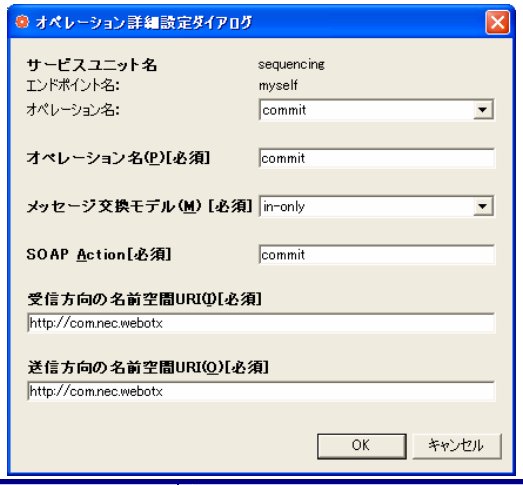


SU エディタで、次の値を設定します。

設定名	設定値	備考
エンドポイント名	myself	
エンドポイントロール	コンシューマ	
サービス名	http://com.nec.webotx	

（名前空間 URI）		
サービス名 （ローカル名）	myself	
外部に公開するエンドポイント URI	http://localhost/esb/account	ホスト名、ポート番号は soap_inbound.war を配備した場所に合わせます。

オペレーション一覧にオペレーションを追加します。「**オペレーション一覧**」の[**追加**]ボタンをクリックし、表示されたダイアログで、次のように設定を行い、[**OK**]ボタンを押します。



設定名	設定値	備考
オペレーション名	commit	
メッセージ交換モデル	robust-in-only	クライアントが ESB から受け取るべきメッセージの内容はないが、ESB でエラーや例外が起こった内容については受け取りたいため。
SOAP Action		空欄
受信方向の名前空間 URI	http://com.nec.webotx	
送信方向の名前空間 URI	http://com.nec.webotx	

クライアントに WSDL を公開する場合は、WebOTX Developer’s Studio の WSDL 生成ウィザードを使用して WSDL の雛形を作成し、WSDL エディタを使って ESB のシーケンスではじめに呼び出すサービス（この例では営業システム）が対応している XML スキーマを WSDL にインポートすると共に、SU エディタで入力した値を WSDL に反映させます。SU エディタで指定した値から WSDL へ反映させる値は次の通りです。

```
<wsdl:definitions name="esb"
                  targetNamespace="http://com.nec.webotx"
                  xmlns:tns="http://com.nec.webotx"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap
  <wsdl:types>
    <xsd:import namespace="http://com.nec.webotx" schemaLocation="
    <xsd:import namespace="http://com.nec.webotx" schemaLocation="
  </wsdl:types>
  <wsdl:message name="commitResponse">
    <wsdl:part name="commitResponse" element="tns:commitResponse"/
  </wsdl:message>
  <wsdl:message name="commitRequest">
    <wsdl:part name="commit" element="tns:commit"/>
  </wsdl:message>
  <wsdl:portType name="esb">
    <wsdl:operation name="commit">
      <wsdl:input message="tns:commitRequest"/>
      <wsdl:output message="tns:commitResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="esbSOAP" type="tns:esb">
    <soap:binding style="document" transport="http://schemas.xmlso
    <wsdl:operation name="commit">
      <soap:operation soapAction=""/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="esb">
    <wsdl:port name="esbSOAP" binding="tns:esbSOAP">
      <soap:address location="http://localhost/esb/account"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

SU エディタ設定名	WSDL 記述箇所	説明
外部に公開するエンドポイント URI	①	
オペレーション名	②	オペレーション名は GUI 表示上の便宜的な名称です。WSDL には、esb が受け取る SOAP メッセージの Body 要素直下の子要素の要素名を記述します。
メッセージ交換モデル	③	input 要素は全ての場合に必ず記述します。output 要素は in-out のときに追加します。fault 要素は SOAP detail に特定のメッセージが入るときのみ記述する必要があります。
SOAP Action	④	この例では空なので記入しませんが、そうでない場合はそのまま転記します。
受信方向の名前空間 URI	⑤	クライアントからのレスポンスメッセージの SOAP Body 直下の要素に修飾している名前空間 URI を指定します。
送信方向の名前空間 URI	⑥	クライアントからのリクエストメッセージの SOAP Body 直下の要素に修飾している名前空間 URI を指定します。

その他の雛形生成時に書き込まれた仮の値は、自由に変更してください。

SU エディタの設定を保存して編集画面を閉じます。

MEMO

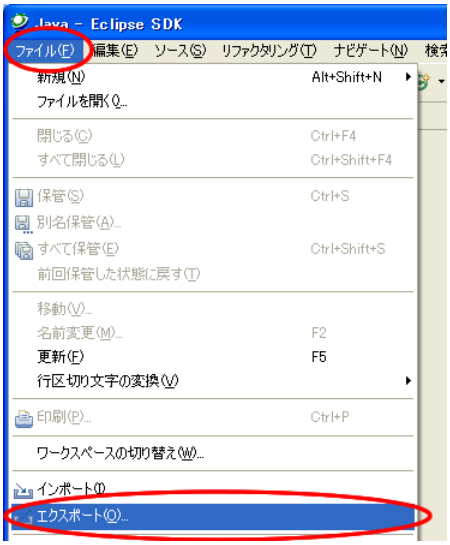
ESB 本体でエラーが発生した場合は、faultstring のみを使います。

1.5.チュートリアル（その5）

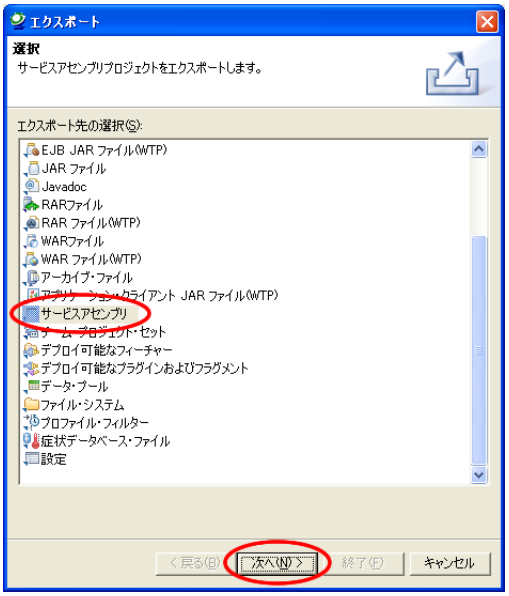
1.5.1.サービスアセンブリをエクスポートする

サービスアセンブリの ZIP ファイルを作成し、配備の準備をします。

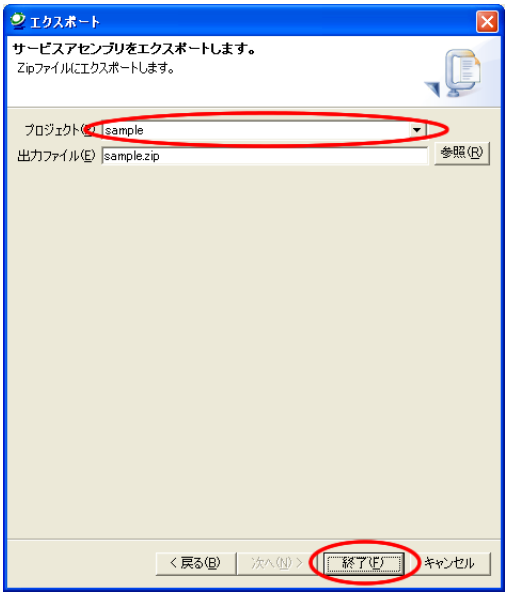
メニューから、**ファイル**｜**エクスポート**を実行します。



「**サービスアセンブリ**」を選択し、[**次へ**]ボタンを押します。



プロジェクトで「**sample**」を選択すると、出力ファイルも「**sample**」と設定されます。このままにしておくと、sample プロジェクトのルートに sample.zip が生成されます。その他の場所にしたい場合は、絶対パスを指定してください。ファイル名は **sample.zip** にします。最後に[**終了**]ボタンを押します。



1.5.2.サービスアセンブリを配備・起動する

WebOTX ESB をインストールしてあるマシンで、次のコマンドを実行します。

```
> <WebOTX_DIR>¥bin¥jbiadmin --user ユーザ名 --password パスワード --port 6212
```

サービスアセンブリを配備します。

```
jbiadmin > deploy-service-assembly <sample.zip までのパス>¥sample.zip
```

サービスアセンブリを起動します。

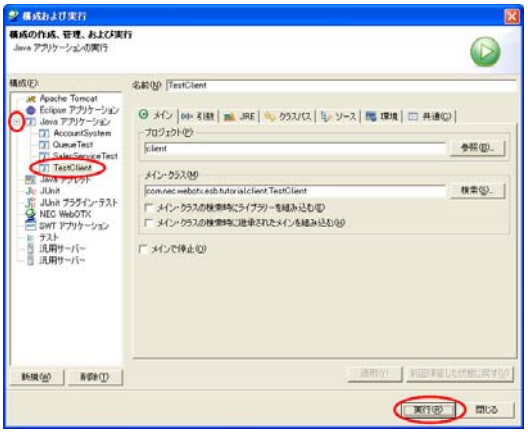
```
jbiadmin > start-service-assembly sample
```

1.5.3.クライアントを実行する

Developer’s Studio にインポートした client プロジェクトのクライアントプログラムを実行して、経理システムに売上が計上されたか確かめます。

account プロジェクトの xml フォルダにある registry.xml をダブルクリックして開きます。これが、経理システムで売上計上したデータを登録するレジストリです。初期状態では、<registry/>というルート要素があるのみです。内容を確認したら、一旦編集画面を閉じます。

メニューから、**実行｜構成および実行**を実行し、構成から **Java アプリケーション**配下の「**TestClient**」を選択し、[**実行**]ボタンを押します。



コンソールビューに「successful.」と表示されれば、正常に動作しました。問題がある場合、ESB から返ってきた Fault メッセージの内容（FaultString）が表示されます。

ここで再び registry.xml を開くと、売上計上したデータが追加されています。

以上で、このチュートリアルのシーケンスは動作し、営業と経理のシステム統合が完成しました。