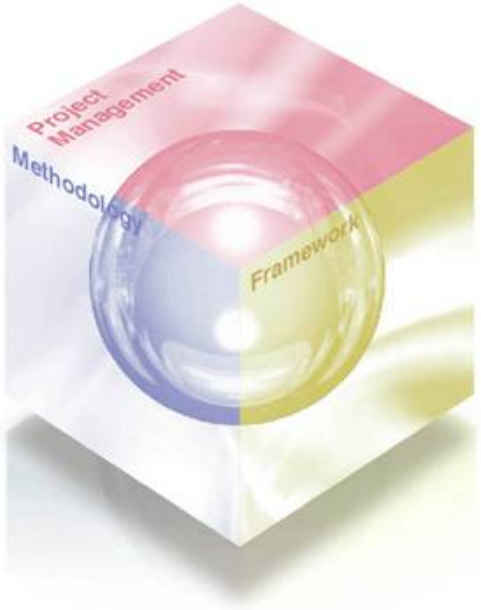
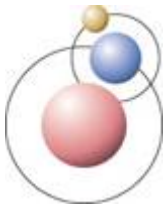
TERASOLUNA® Client Framework for .NET 2.1.0.1
アーキテクチャ説明書



株式会社NTTデータ

TERASOLUNA®
BEST SOLUTION FOR SYSTEM DEVELOPMENT

NTT DATA



本ドキュメントを使用するにあたり、以下の規約に同意していただく必要があります。同意いただけない場合は、本ドキュメント及びその複製物の全てを直ちに消去又は破棄してください。

1. 本ドキュメントの著作権及びその他一切の権利は、NTTデータあるいはNTTデータに権利を許諾する第三者に帰属します。
2. 本ドキュメントの一部または全部を、自らが使用する目的において、複製、翻訳、翻案することができます。ただし本ページの規約全文、およびNTTデータの著作権表示を削除することはできません。
3. 本ドキュメントの一部または全部を、自らが使用する目的において改変したり、本ドキュメントを用いた二次的著作物を作成することができます。ただし、「参考文献：TERASOLUNA Client Framework for .NET アーキテクチャ説明書」あるいは同等の表現を、作成したドキュメント及びその複製物に記載するものとします。
4. 前2項によって作成したドキュメント及びその複製物を、無償の場合に限り、第三者へ提供することができます。
5. NTTデータの書面による承諾を得ることなく、本規約に定められる条件を超えて、本ドキュメント及びその複製物を使用したり、本規約上の権利の全部又は一部を第三者に譲渡したりすることはできません。
6. NTTデータは、本ドキュメントの内容の正確性、使用目的への適合性の保証、使用結果についての的確性や信頼性の保証、及び瑕疵担保義務も含め、直接、間接に被ったいかなる損害に対しても一切の責任を負いません。
7. NTTデータは、本ドキュメントが第三者の著作権、その他如何なる権利も侵害しないことを保証しません。また、著作権、その他の権利侵害を直接又は間接の原因としてなされる如何なる請求（第三者との間の紛争を理由になされる請求を含む。）に関しても、NTTデータは一切の責任を負いません。

本ドキュメントで使用されている各社の会社名及びサービス名、商品名に関する登録商標および商標は、以下の通りです。

- Microsoft、Visual Stuido、Windows、.NET Frameworkは、米国Microsoft Corp.の米国及びその他の国における登録商標または商標です。
- TERASOLUNAは、株式会社NTTデータの登録商標です。
- その他の会社名、製品名は、各社の登録商標または商標です。

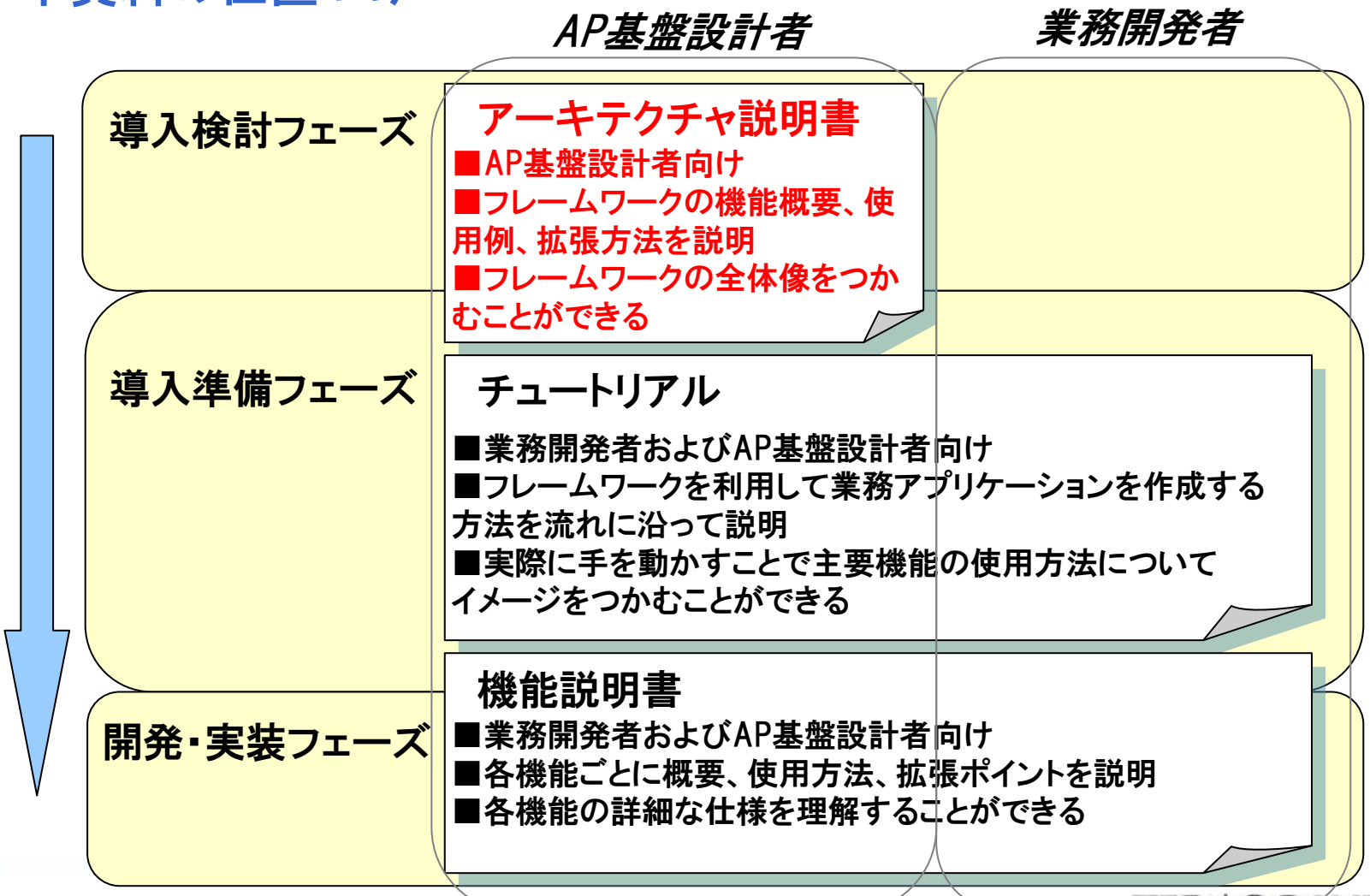
目次

はじめに(1/5)

- 概要
  - 本資料は、クライアントフレームワーク「TERASOLUNA Client Framework for .NET 2.1」を解説した資料である
  - TERASOLUNA Client Framework for .NET 2.1は、.NETフレームワーク上で動作する、リッチクライアントアプリケーション構築のためのフレームワークである

はじめに(2/5)

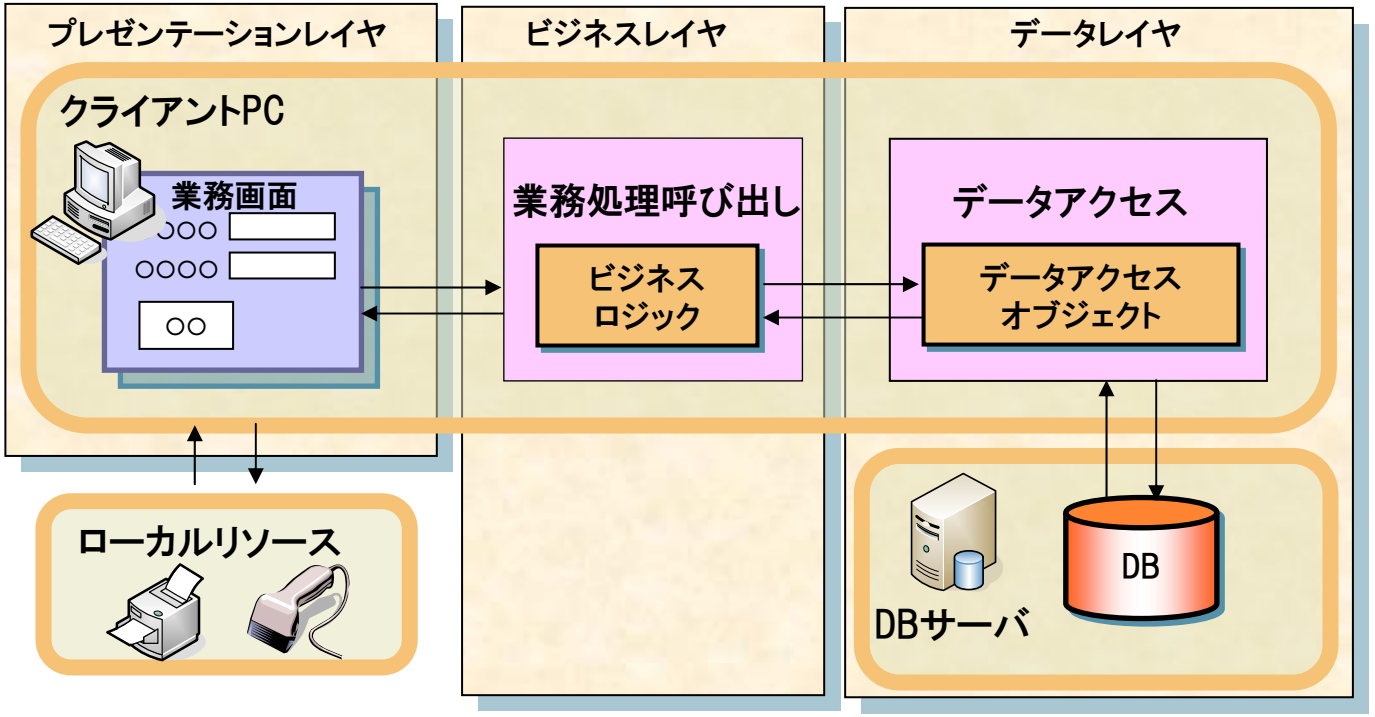
本資料の位置づけ



はじめに(3/5)

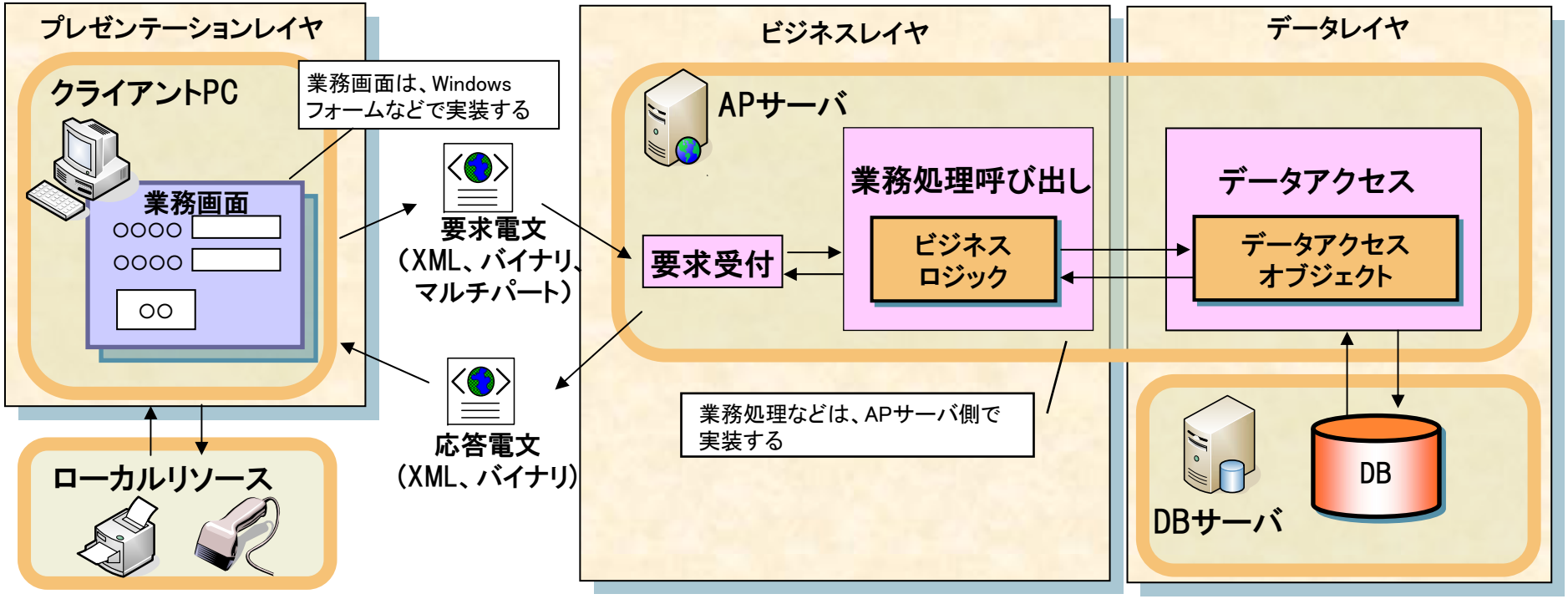
TERASOLUNA Client Framework for .NETが想定するクライアントサーバ型システム

- クライアントアプリケーションで業務画面処理、業務処理実行、データアクセスを行う
  - 要件に応じて、バーコードリーダやOCRなど、クライアントのローカルリソースにもアクセスする
- サーバにはデータベースのみ配置する



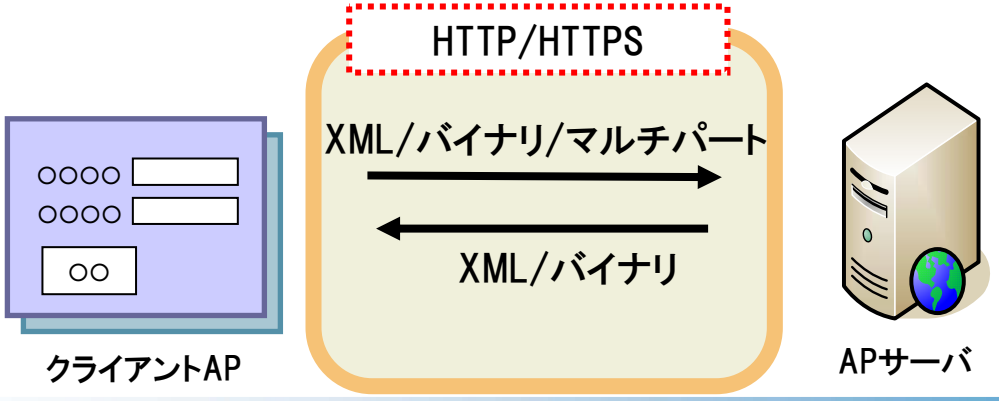
はじめに(4/5)

- TERASOLUNA Client Framework for .NETが想定するリッチクライアント型システム
  - クライアントアプリケーションでは主に業務画面処理を行う
    - 要件に応じて、バーコードリーダやOCRなど、クライアントのローカルリソースにもアクセスする
  - サーバアプリケーションでは、業務処理実行、データアクセスを行う



はじめに（5/5）

- サーバ・クライアント間の通信方式の特徴
  - クライアント・サーバ間は、HTTPまたはHTTPSでの通信とする
  - HTTPボディ部の形式はXML形式、バイナリ形式、マルチパート形式の3パターンを前提としている
    - XML通信
      - 要求がXML形式、応答がXML形式
    - ファイルアップロード
      - 要求がマルチパート形式、応答がXML形式
      - 要求がバイナリ形式、応答がXML形式
    - ファイルダウンロード
      - 要求がXML形式、応答がバイナリ形式
  - サーバはセッション状態を保持しない



動作環境

動作確認済環境

対応OS

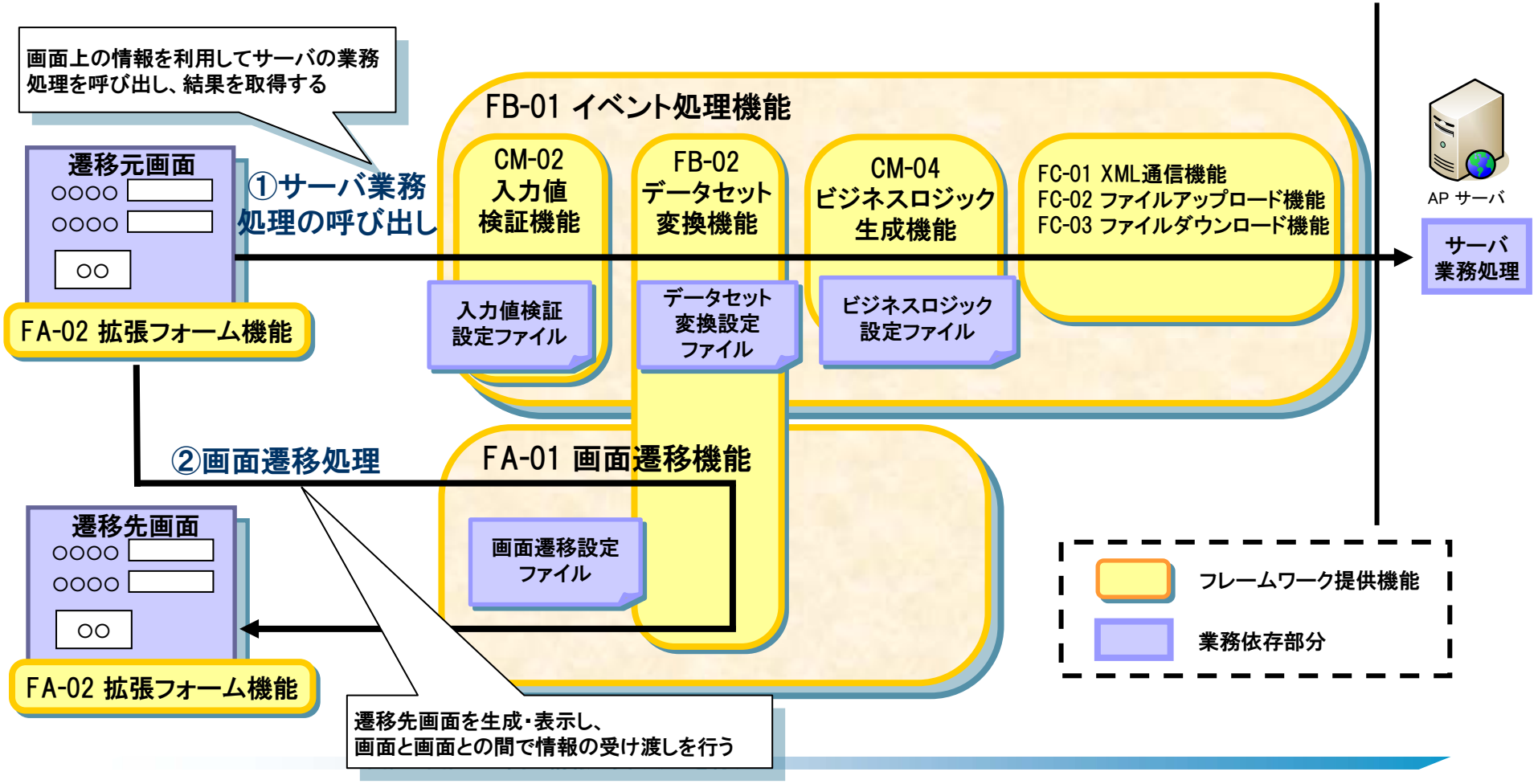
- Windows XP Professional SP2
- Windows Vista Enterprise

.NET Framework

- .NET Framework 2.0
- .NET Framework 2.0 SP1
- .NET Framework 3.0
- .NET Framework 3.5

アーキテクチャ概観

- クライアントアプリケーションにおける①サーバ業務処理の呼び出し、②画面遷移処理を実現するための各機能を提供



機能概要

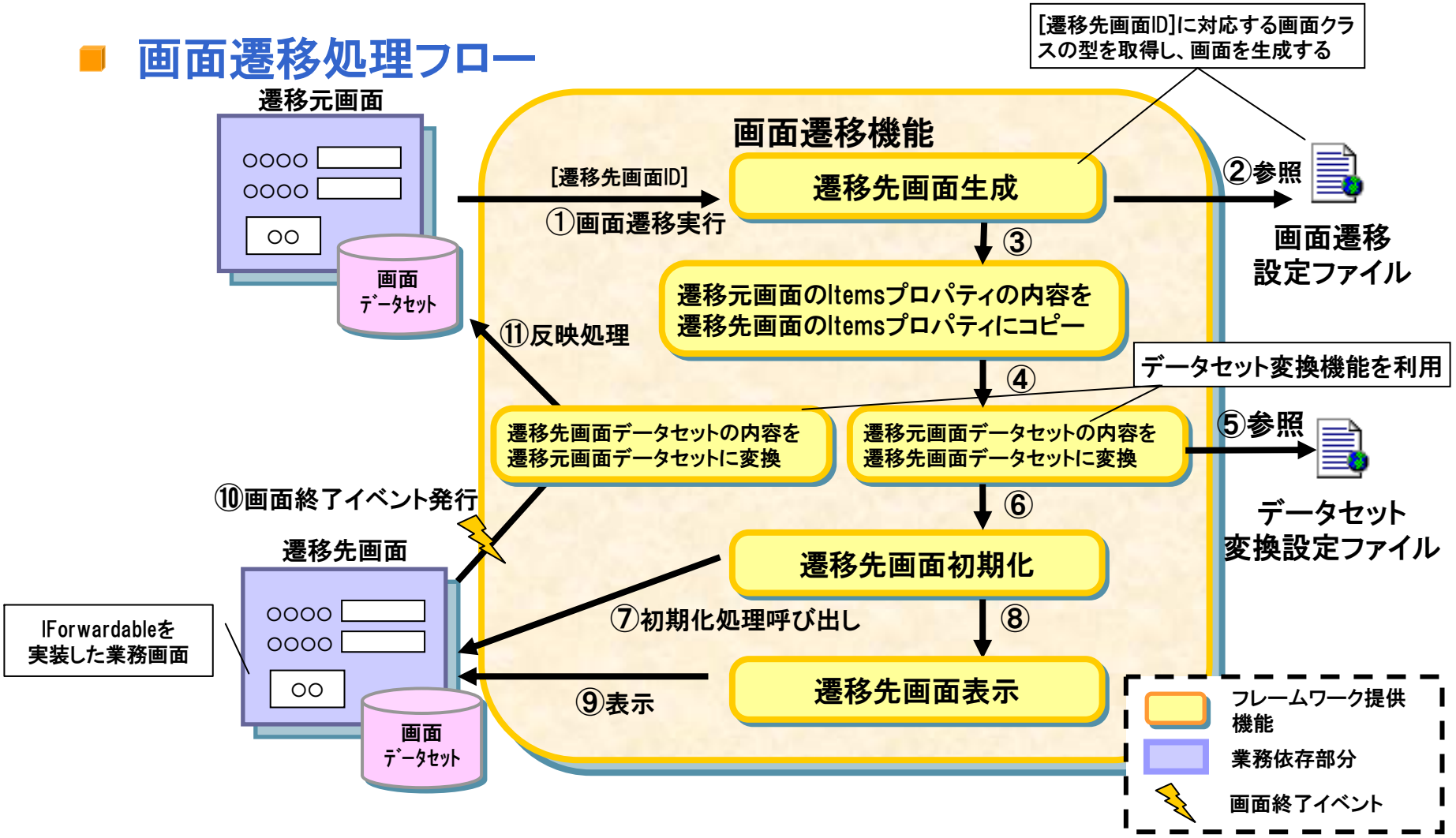
- FA-01 画面遷移機能
  - 画面遷移設定ファイルに定義した画面情報に従って、画面遷移する機能を提供する
- FA-02 拡張フォーム機能
  - 画面クラスの標準実装としてFormBaseクラスを提供する
- FB-01 イベント処理機能
  - 業務処理呼び出しの処理フローを定型化し、Visual Studioのデザイナを用いてビジネスロジックや入力値検証、各種イベント処理等を簡単に実行できる仕組みを提供する
- FB-02 データセット変換機能
  - 2つのデータセット間でデータ変換を行う機能を提供する
- FC-01 XML通信機能
  - HTTPまたはHTTPSでサーバとXML通信を行う機能を提供する
- FC-02 ファイルアップロード機能
  - サーバにファイル(バイナリ)やマルチパート形式のデータをアップロードする機能を提供する
- FC-03 ファイルダウンロード機能
  - サーバからファイル(バイナリ)をダウンロードする機能を提供する

FA-01 画面遷移機能 – 概要

- 画面遷移設定ファイルに定義した画面情報に従って、画面遷移する機能を提供する
  - デザイナで遷移元画面に画面遷移コンポーネント(FormForwarder)を追加し、コンポーネントのプロパティで遷移先画面の設定ができる
  - 画面遷移設定ファイルに定義した画面IDで遷移先画面を指定することで画面間の依存関係を疎にできる
  - 遷移元・遷移先画面でデータの受け渡しをする機能を提供する
    - 画面のItemsプロパティの値
    - 画面データセットの値
  - イベントにより遷移先画面が閉じられたことを遷移元画面に通知する機能を提供する

FA-01 画面遷移機能 – 動作イメージ

画面遷移処理フロー



FA-01 画面遷移機能 – 使用方法(1/6)

アプリケーション構成ファイルの設定

利用する画面遷移設定ファイルのパスを設定する

- 設定ファイルは複数定義することができる

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="viewConfiguration"
             type="TERASOLUNA.Fw.Client.Configuration.View.ViewConfigurationSection,TERASOLUNA.Fw.Client"/>
    </configSections>
     <viewConfiguration>
      <files>
        <file path="config¥view_maintenance.config"/>
        <file path="config¥view_petshop.config"/>
        <file path="config¥view_foodshop"/>
      </files>
    </viewConfiguration>
</configuration>
```

画面遷移設定ファイルのパス

FA-01 画面遷移機能 – 使用方法(2/6)

画面遷移設定ファイルの設定

- 遷移先画面の「画面ID」と「画面クラスの完全修飾名」を設定する

画面遷移設定ファイル設定例

```
<?xml version="1.0" encoding="utf-8"?>
<viewConfiguration xmlns=" http://www.terasoluna.jp/schema/ViewSchema.xsd">
  <view id="inputForm" type="SampleApplication.InputForm, SampleApplication"/>
  <view id="outputForm" type="SampleApplication.OutputForm, SampleApplication"/>
</viewConfiguration>
```

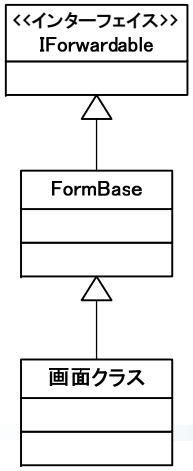
画面ID

画面クラスの完全修飾名

FA-01 画面遷移機能 – 使用方法(3/6)

画面クラスの実装

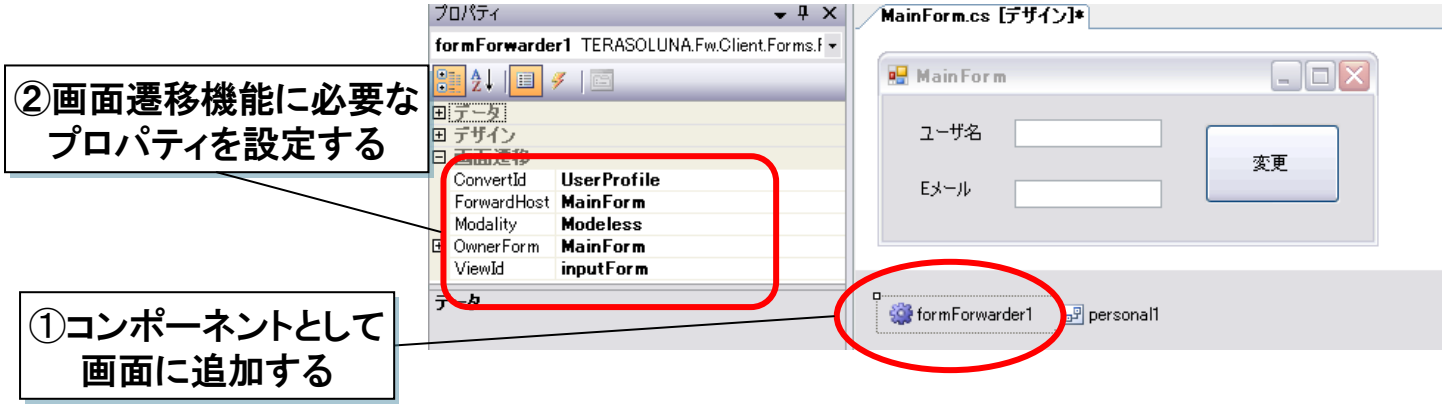
- 画面遷移機能を利用するには、画面クラスに遷移可能画面インターフェイス(IForwardable)を実装する
  - Itemsプロパティ
    - 画面が外部へ公開するコレクション
    - 他画面とのデータのやりとりに利用する
  - ViewDataプロパティ
    - 画面データセットを保持する
  - Initメソッド
    - 画面を初期化する処理を実装する



※IForwardableを標準実装したFormBaseクラスが拡張フォーム機能で提供されているため、画面クラスはFormBaseを継承すればよい

FA-01 画面遷移機能 – 使用方法(4/6)

FormForwarderコンポーネントの追加とプロパティの設定



<< 画面遷移機能に関わるプロパティ >>

プロパティ名	説明
ConvertId	データセット変換設定ファイルに定義したコンバートID
ForwardHost	遷移元画面
Modality	遷移先画面の表示方法(アプリケーションモーダル/フォームモーダル/モードレス)
OwnerForm	遷移先画面の親となるフォーム
ViewId	遷移先画面の画面ID

FA-01 画面遷移機能 – 使用方法(5/6)

画面遷移の実行

- FormForwarderのExecuteメソッドを呼び出して画面遷移を実行する
- 実行結果はForwardResultオブジェクトとして返却される

- 以下に、画面遷移を実行し、その結果で処理を分岐する実装例を示す

画面からの呼び出しコード例

```
//ログイン画面を表示し、自分自身を閉じる。
private void DisplayLogonForm()
{
    ForwardResult result = formForwarder1.Execute();
    if (result == ForwardResult.Abort)
    {
        throw new ApplicationException();
    }
    this.Close();
}
```

※　標準の画面遷移機能では、元の画面の制御は行わない

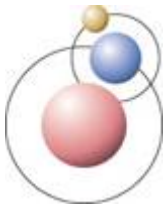
FA-01 画面遷移機能 – 使用方法(6/6)

- 遷移先画面の終了イベント
  - FormForwarderは遷移先画面が閉じられたことの通知を受けるイベントを提供する
    - FormClosing
      - 遷移先画面が閉じられようとしている際に発生する
    - FormClosed
      - 遷移先画面が閉じられた際に発生する
  - 以下に、FormClosedイベントに登録するイベントハンドラの実装例を示す

FormClosedイベントハンドラの実装例

```
private void formForwarder1_FormClosed(object sender, ForwardableFormCloseEventArgs e)
{
    // 遷移先画面から遷移元画面にデータを受け渡す
    if (e.Items.Contains["Result"] && e.Items["Result"] is LoginResult)
    {
        LoginResult result = e.Items["Result"];
        this.DispatchForm(result);
    }
    else
    {
        throw new ApplicationException();
    }
}
```

イベントオブジェクトは遷移先画面の情報を保持する



FA-01 画面遷移機能 – 拡張方法

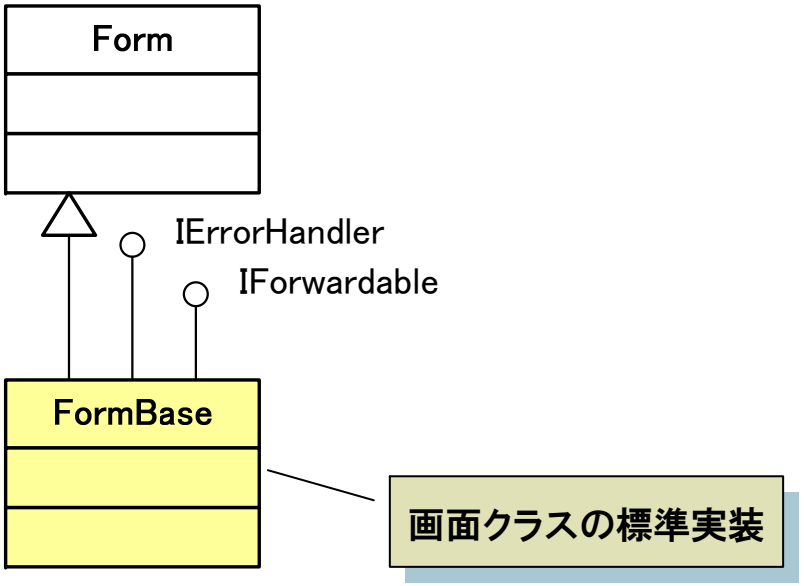
- 独自の画面遷移機能を実装する場合
  - フレームワークに沿って独自の画面遷移機能を使用する場合、FormForwarder、あるいは基底クラスのForwarderを継承したクラスを作成して拡張する

例:

- 同一Form内でPanelなどを遷移させる機能
- 画面遷移実行時に、ユーザーの認証状態によって処理を分岐する機能
- 同一画面の複数表示抑制機能

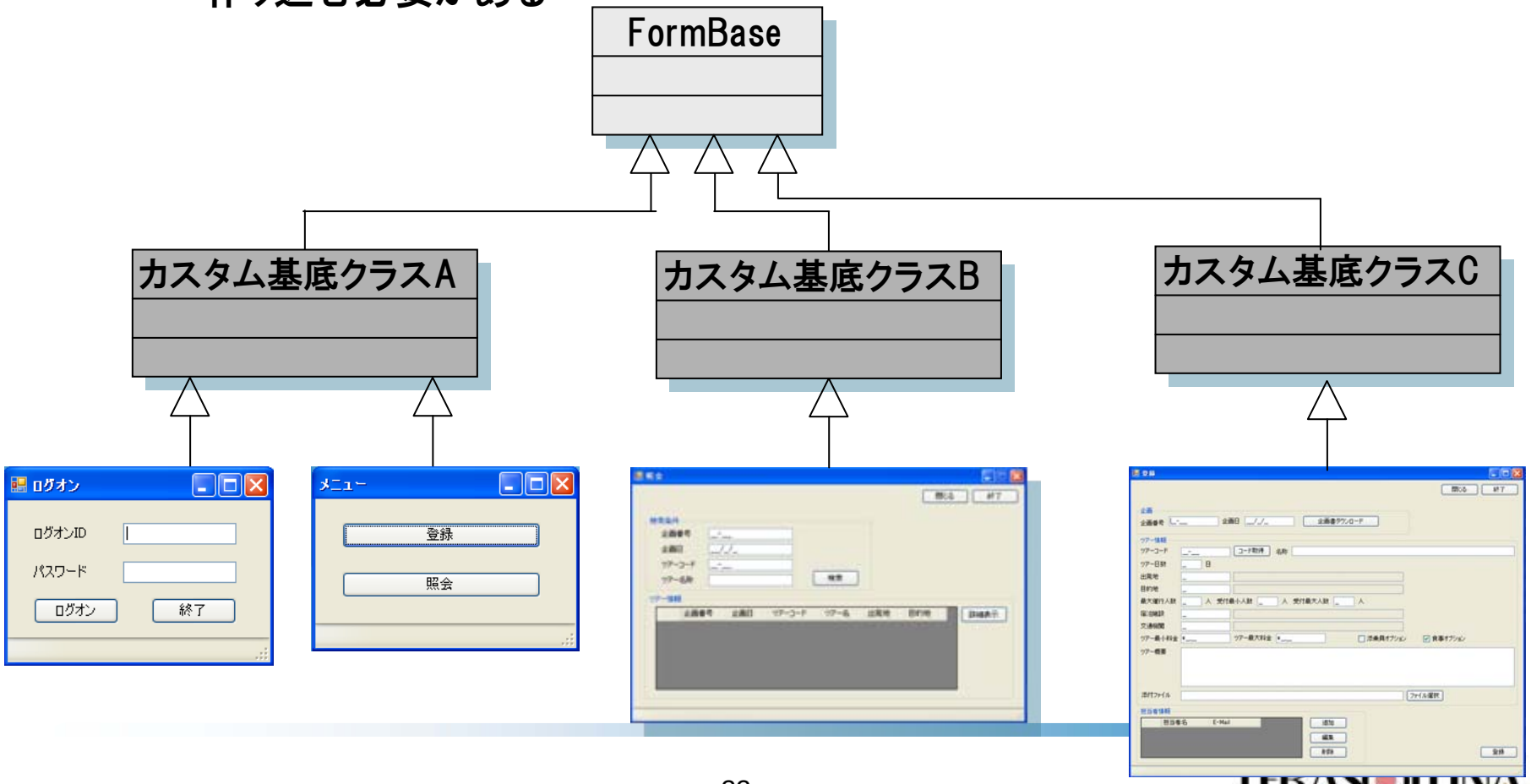
FA-02 拡張フォーム機能 – 概要

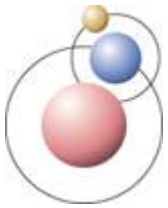
- 画面クラスの標準実装としてFormBaseクラスを提供する
  - イベント処理機能のエラー処理ハンドラインターフェイス(IErrorHandler)を実装
    - イベントコントローラのErrorHanderプロパティに設定するインスタンスとなる
  - 画面遷移機能の遷移可能画面インターフェイス(IForwardable)を実装



FA-02 拡張フォーム機能 – 使用方法

- 業務で利用する画面クラスはFormBaseを派生して設計する
  - エラー処理の詳細、画面の初期化処理などは業務によって異なるため、作り込む必要がある





FA-02 拡張フォーム機能 – 拡張方法(1/2)

- 独自のエラー処理を実現したい場合は、エラー処理ハンドラインターフェイス(IErrorHandler)のメソッドをオーバーライドする
  - HandleError(string resultString, IList<MessageInfo> messages, object view)メソッド
    - 標準実装では、引数に渡されたmessagesからエラーパスを取得し、view(データセット)の対応する項目に対してエラーメッセージを設定する
  - ClearError(object view)メソッド
    - 標準実装では、引数に渡されたview(データセット)に設定されているエラーメッセージをクリアする

FA-02 拡張フォーム機能 – 拡張方法(2/2)

- 独自の画面遷移処理を実現したい場合は、遷移可能画面インターフェイス(IForwardable)のメソッドをオーバーライドする
  - Itemsプロパティ
    - 画面が外部へ公開するコレクション
    - 画面間でデータをやりとりするときに利用する
  - ViewDataSetプロパティ
    - 画面データセットを保持する
  - Init()メソッド
    - 画面を初期化する処理を実装する
    - 初期化成功時にtrue、初期化失敗時にfalseを返却する
    - 標準では空実装なので、Formの初期化が必要な場合はオーバーライドする

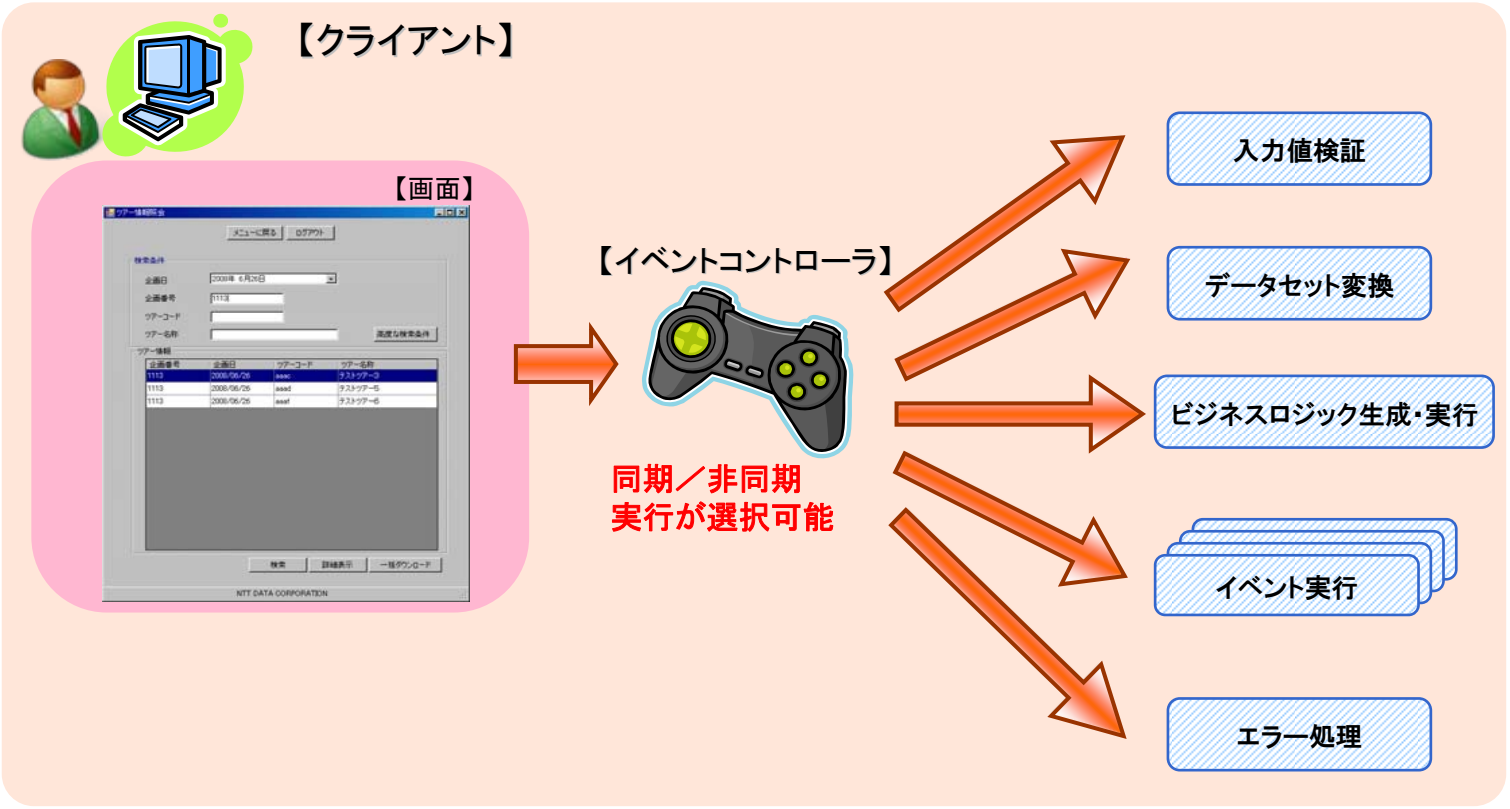
FB-01 イベント処理機能 – 概要(1/5)

- 業務処理呼び出しの処理フローを定型化し、Visual Studioのデザイナを用いてビジネスロジックや入力値検証、各種イベント処理等を簡単に実行できる仕組みを提供する
  - 5つの処理フローを定型化する
    - 入力値検証
    - データセット変換
    - ビジネスロジック生成・実行
    - イベント実行
    - エラー処理
  - 開発コストの高いビジネスロジックの非同期処理をサポートする

FB-01 イベント処理機能 – 概要 (2/5)

業務処理呼び出しの動作概念図

- イベントコントローラ(イベント処理コンポーネント)を通して各処理が実行される



FB-01 イベント処理機能 – 概要 (3/5)

- 入力値検証
  - 入力値検証設定ファイルを用いて、画面に入力された項目を検証する
    - 入力値検証の詳細に関しては、『CM-03 入力値検証機能』を参照
- データセット変換
  - 画面から入力されたデータをビジネスロジックに渡し、ビジネスロジックから返却されたデータを画面へ反映する
    - データセット変換の詳細に関しては、『FB-02 データセット変換機能』を参照
- ビジネスロジック生成・実行
  - 業務処理および通信処理を行うビジネスロジックを生成・実行する
    - ビジネスロジック生成の詳細に関しては、『CM-04 ビジネスロジック生成機能』を参照
    - ビジネスロジックの実行は同期・非同期をサポートする

FB-01 イベント処理機能 – 概要 (4/5)

イベント実行

- 以下の7種類のイベントを提供している

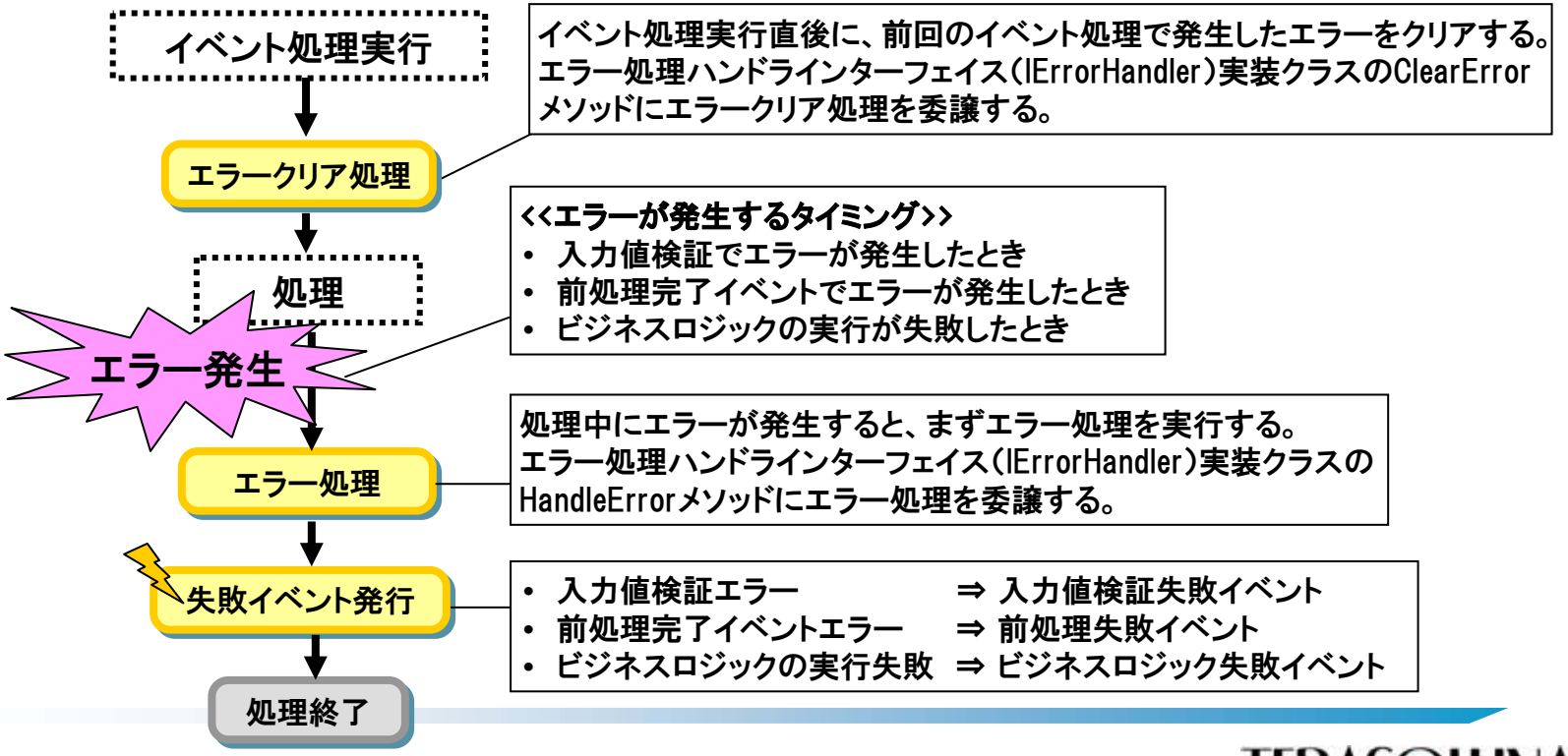
項番	イベント名	説明
1	カスタムチェックイベント	イベント処理機能が実行する入力値検証の直後に発生する。 独自の入力チェック(相関チェックなど)を行う場合に利用する。
2	入力値検証失敗イベント	カスタムチェックイベントを含む、入力値検証でエラーとなった場合に発生する。
3	前処理イベント	全ての入力チェックが成功し、ビジネスロジック入力オブジェクトが作成された段階で発生する。ビジネスロジック入力オブジェクトに対して独自の処理を行う場合に利用する。
4	前処理失敗イベント	前処理イベント中でエラーとなった場合に発生する。
5	ビジネスロジック失敗イベント	ビジネスロジックが失敗(BLogicResultのResultStringが”success”以外)した場合に発生する。
6	非同期処理完了イベント	ExecuteAsyncによって実行された非同期処理が完了した場合に発生する。 本イベントは同期処理の場合は実行されない。
7	進行状況通知イベント	ビジネスロジックの進行状況が変化した場合に発生する。ビジネスロジックが IProgressChangedEventInvokerインターフェイスを実装している場合のみ利用できる。

FB-01 イベント処理機能 – 概要 (5/5)

エラー処理

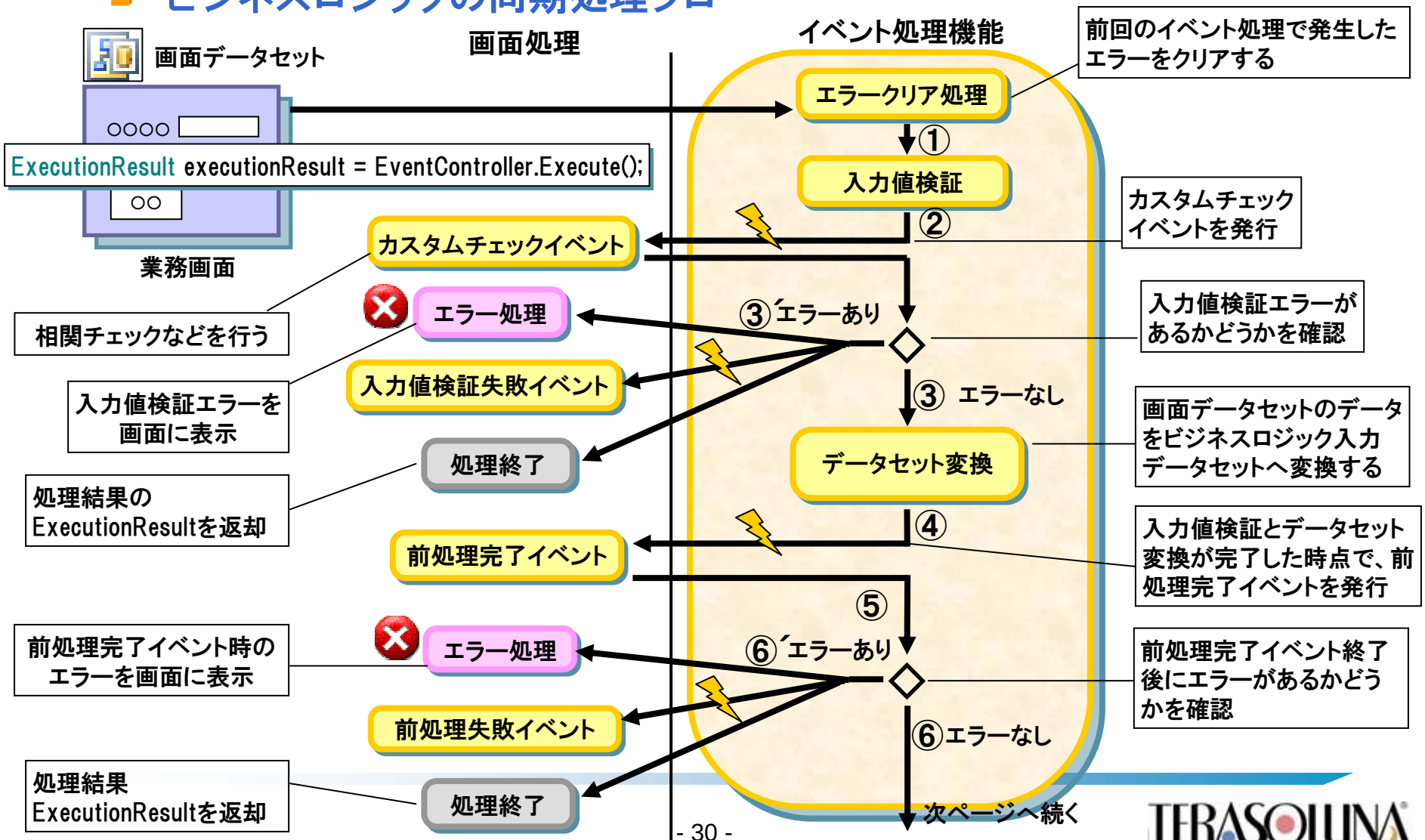
◆イベント処理全体で統一的なエラー処理とエラークリア処理を行う

- エラー処理は入力値検証エラーなどのエラー情報を画面へ表示する処理を行う
- エラークリア処理はイベント処理実行の都度、画面へ表示されているエラー情報をクリアする



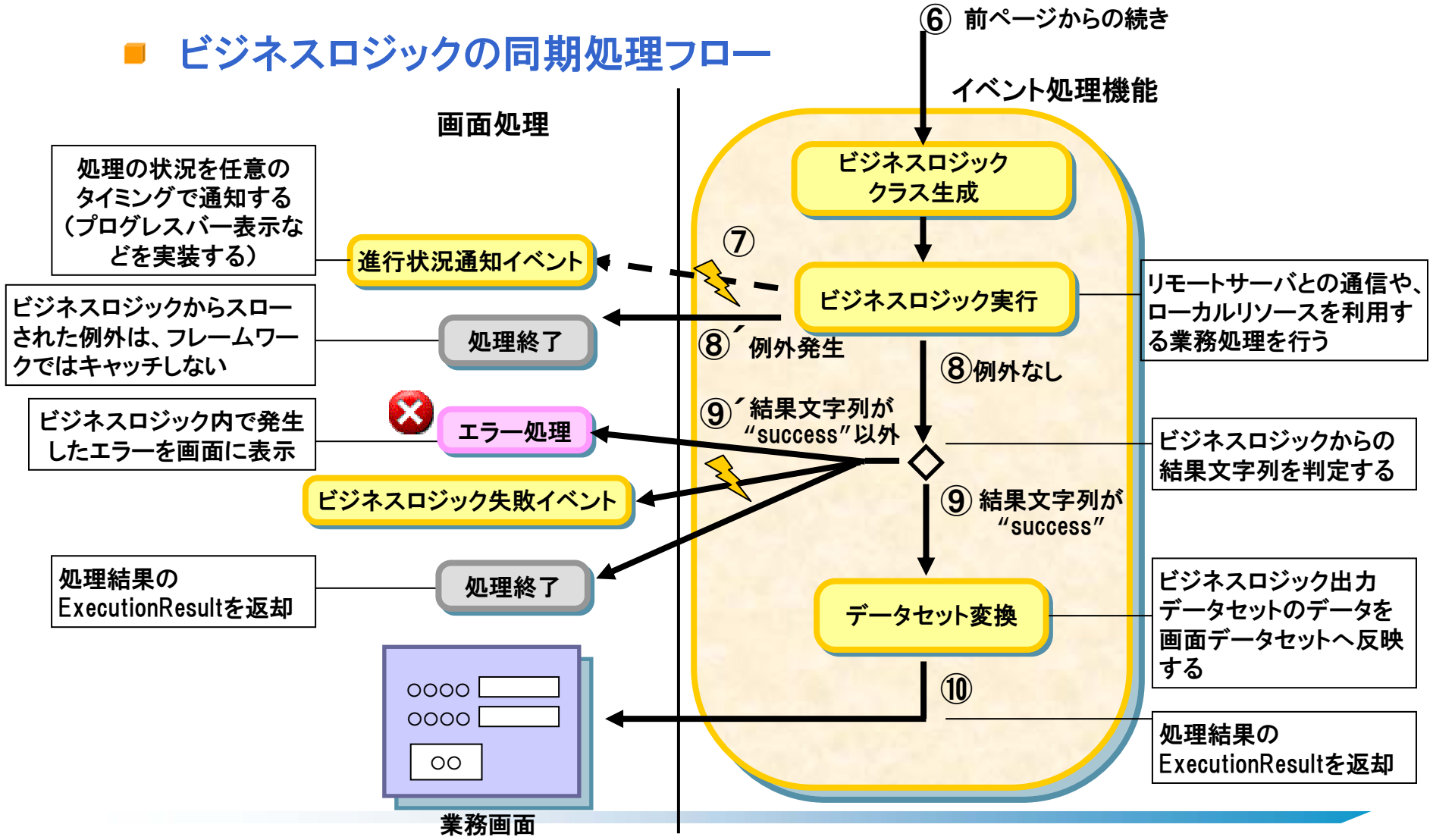
FB-01 イベント処理機能 – 同期処理の動作イメージ（1/2）

ビジネスロジックの同期処理フロー



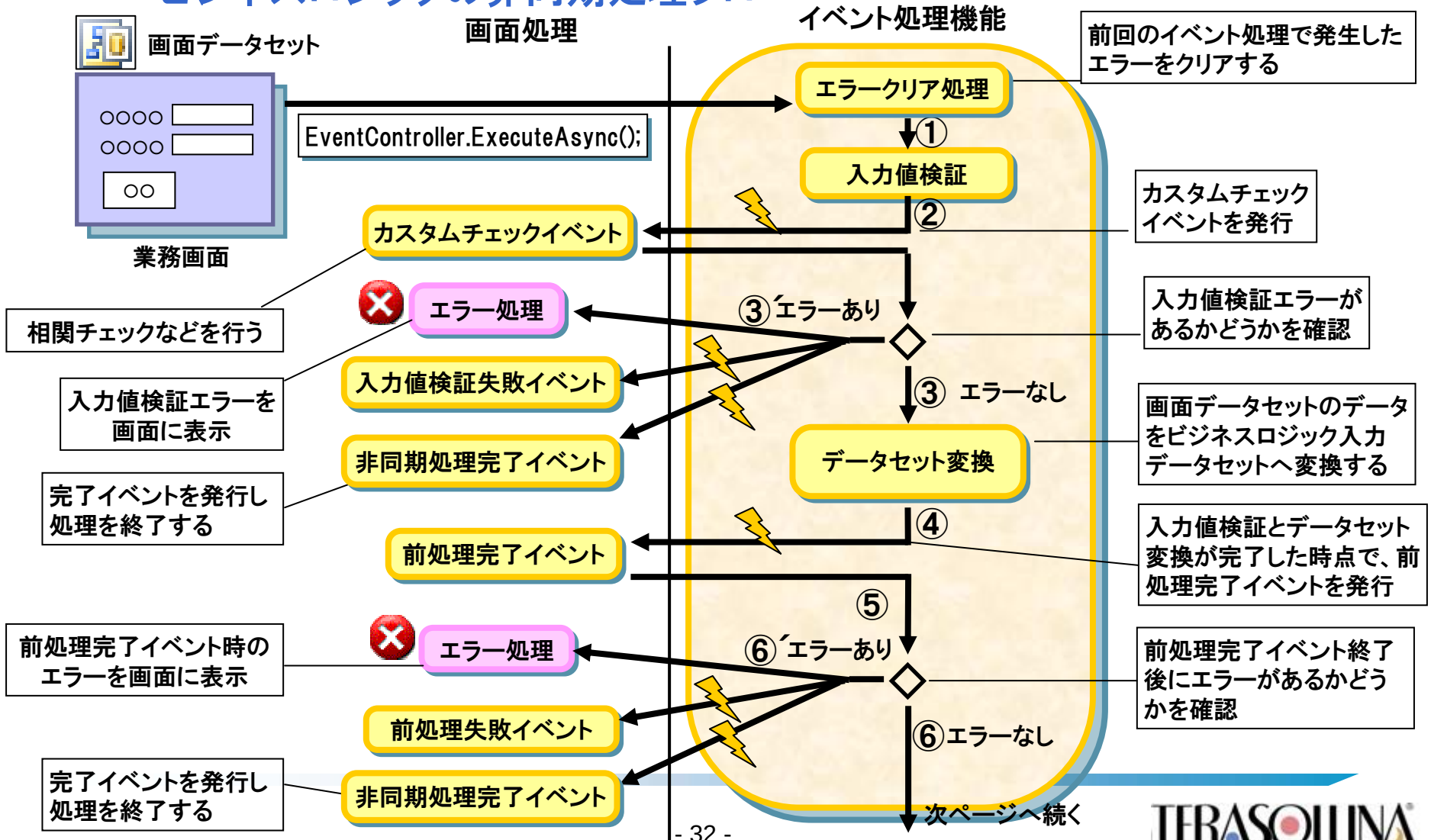
FB-01 イベント処理機能 – 同期処理の動作イメージ(2/2)

ビジネスロジックの同期処理フロー



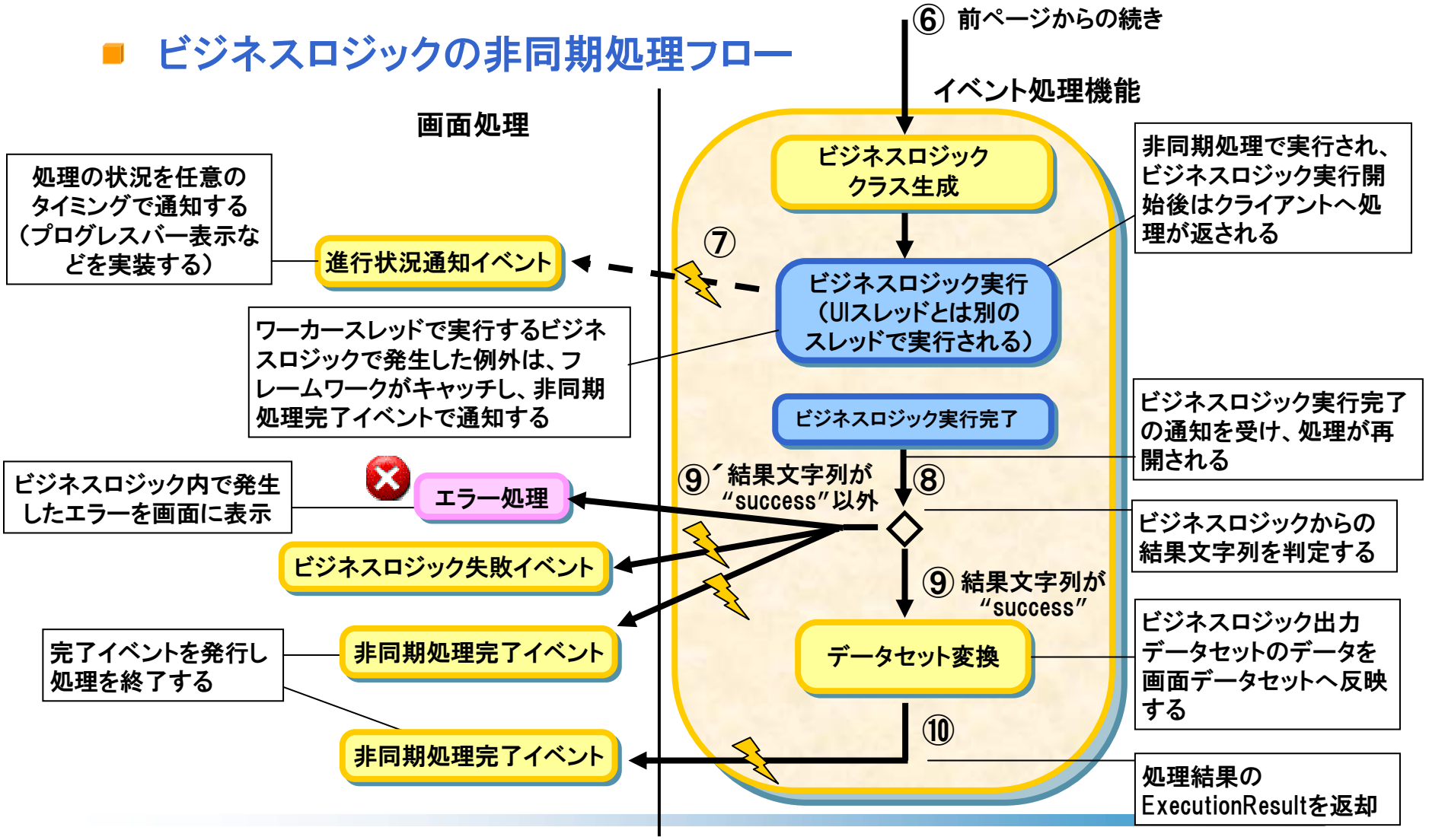
FB-01 イベント処理機能 – 非同期処理の動作イメージ(1/2)

ビジネスロジックの非同期処理フロー



FB-01 イベント処理機能 – 非同期処理の動作イメージ(2/2)

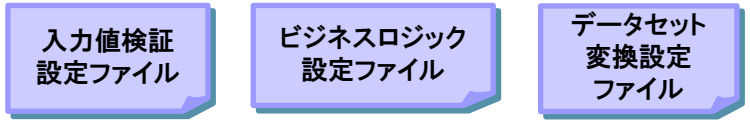
ビジネスロジックの非同期処理フロー



FB-01 イベント処理機能 – 使用方法(1/6)

- アプリケーション構成ファイルの設定
  - 各種設定ファイルのパスを記述する
    - ビジネスロジック設定ファイルのパス
    - データセット変換設定ファイルのパス

- 各設定ファイルの設定
  - 機能ごとの設定ファイルを記述する
    - 入力値検証設定ファイル
    - ビジネスロジック設定ファイル
    - データセット変換設定ファイル



※設定方法の詳細は以下を参照のこと

CM-02 入力値検証機能
CM-04 ビジネスロジック生成機能
} TERASOLUNA Server/Client Framework for .NET 2.1 アーキテクチャ説明書(共通編)

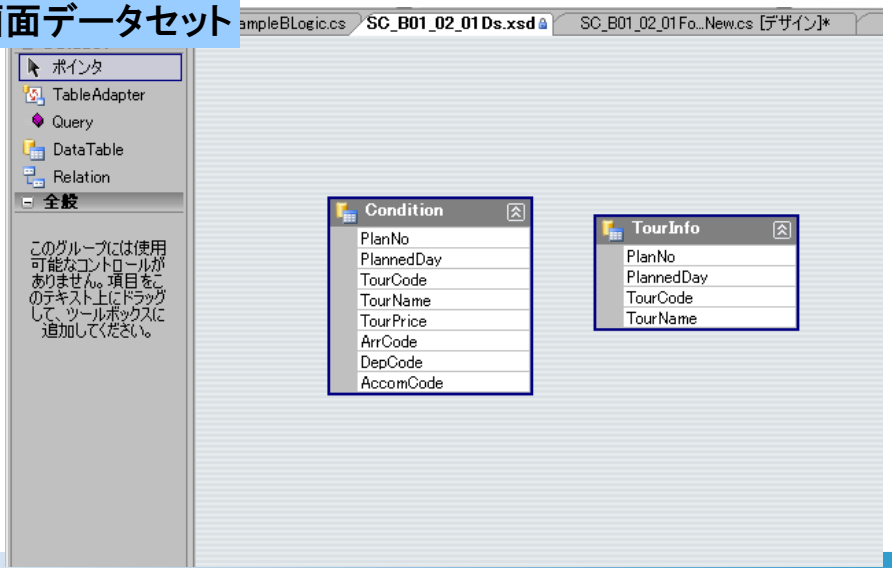
FB-02 データセット変換機能 → 本アーキテクチャ説明書

FB-01 イベント処理機能 – 使用方法(2/6)

- 画面クラスの作成
  - 画面クラスを作成し、作成したフォームにツールボックスからテキストボックスやボタンなどの画面項目を配置する
  - 通常は、拡張フォーム機能が提供するFormBase(画面クラスの標準実装)を継承して画面を作成する
- 画面データセットの作成
  - 1つの画面に対して1つのデータセットを用意する

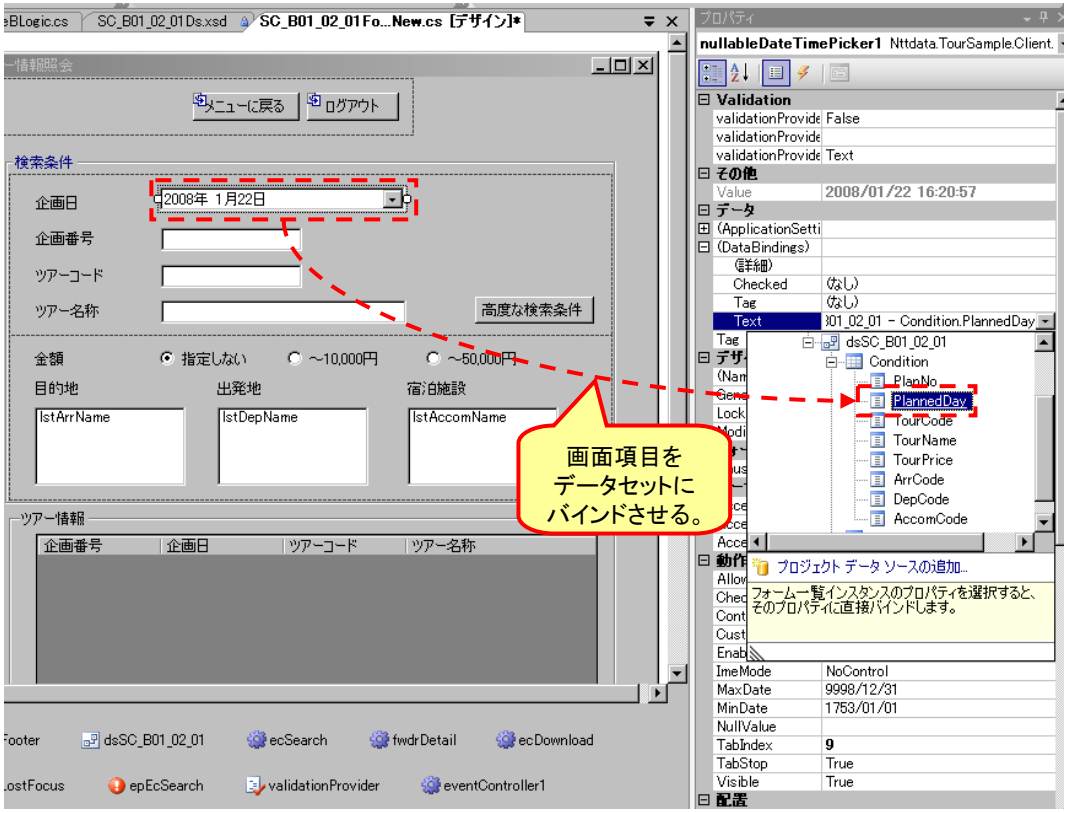
画面クラス

画面データセット



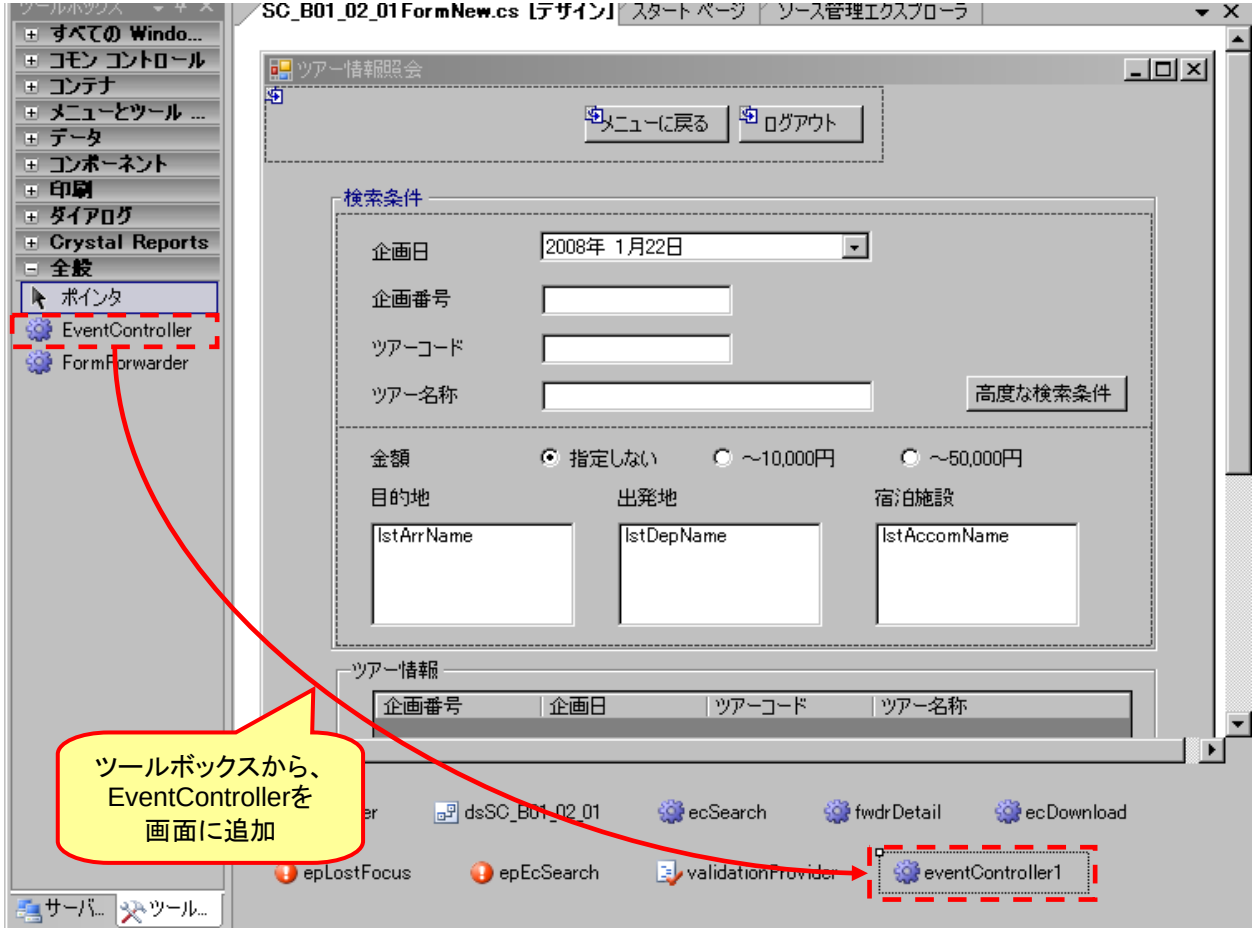
FB-01 イベント処理機能 – 使用方法(2/6)

- 画面項目とデータセットのバインド
  - Visual Studioのデザイナを使って、画面項目を対応するデータセットのカラムにバインドさせる。



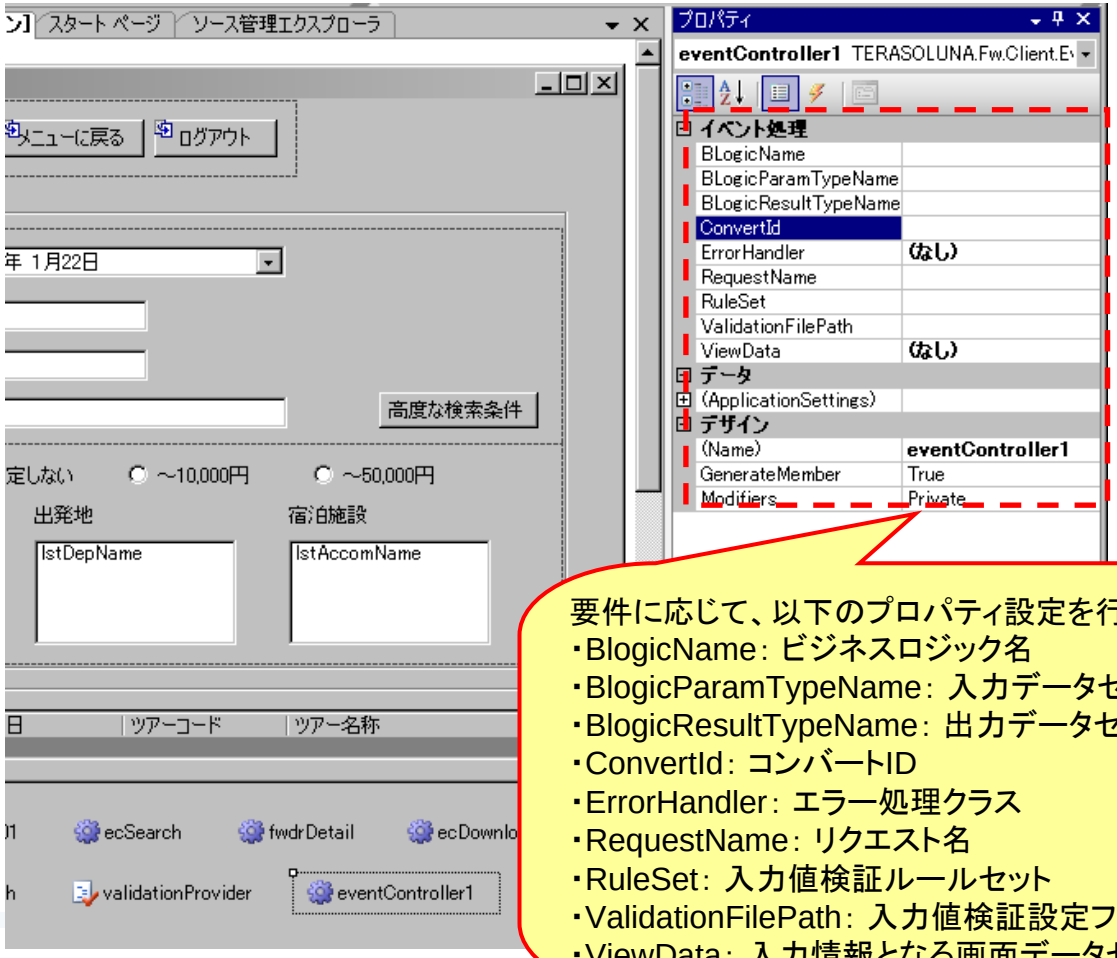
FB-01 イベント処理機能 – 使用方法(2/6)

画面へのイベントコントローラの追加



FB-01 イベント処理機能 – 使用方法(3/6)

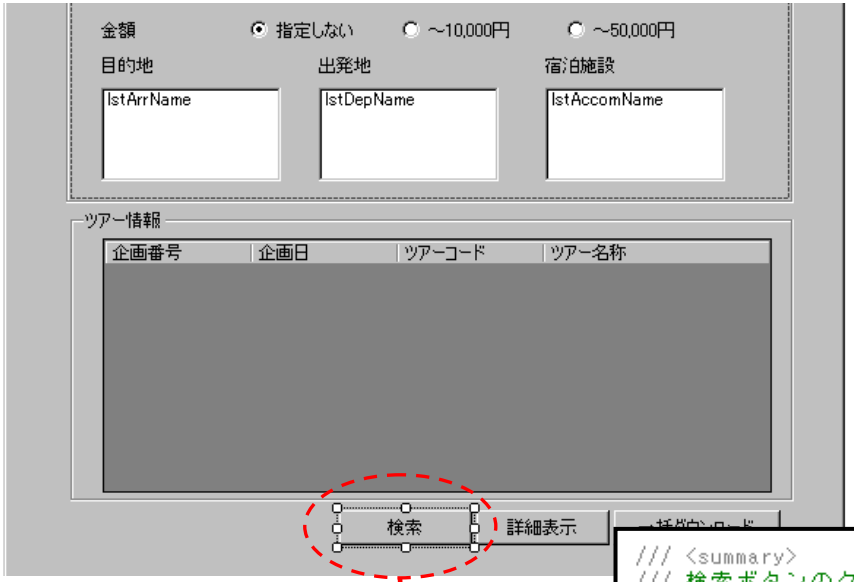
- イベントコントローラのプロパティ設定



要件に応じて、以下のプロパティ設定を行う。
- BlogicName: ビジネスロジック名
- BlogicParamTypeName: 入力データセット型名
- BlogicResultTypeName: 出力データセット型名
- ConvertId: コンバートID
- ErrorHandler: エラー処理クラス
- RequestName: リクエスト名
- RuleSet: 入力値検証ルールセット
- ValidationFilePath: 入力値検証設定ファイル名
- ViewData: 入力情報となる画面データセット

FB-01 イベント処理機能 – 使用方法(4/6)

イベント処理の実行



同期処理の場合

```
ExecutionResult executionResult = eventController.Execute();
```

非同期処理の場合

```
eventController.ExecuteAsync();
```

イベントコントローラの Execute／ExecuteAsync メソッドを実行する。

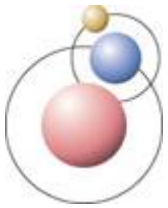
```
/// <summary>
/// 検索ボタンのクリックイベント。
/// </summary>
private void search_Click(object sender, EventArgs e)
{

    //イベントコントローラを用いてイベント処理を実行する
    ExecutionResult result = eventController1.Execute();

    if (result.Success)
    {
        // 成功時の処理
        MessageBox.Show("イベント処理が成功しました");
    }
    else
    {
        // 失敗時の処理
        MessageBox.Show("イベント処理が失敗しました");
    }
```

Clickイベントなどにイベント処理の実行を実装する。

イベントが正常に実行された場合、Successプロパティにtrueが設定される。



FB-01 イベント処理機能 – 使用方法(5/6)

- 処理結果の受け取り方
  - 同期処理
    - 戻り値としてExecutionResultが返却される
  - 非同期処理
    - 非同期処理完了イベントハンドラで処理結果を受け取る

```
// 同期処理の例
private void btnGetTourCode_Click
        (object sender, EventArgs e)
{
    // 戻り値から結果オブジェクトを取得する。
    ExecutionResult result =  eventController1.Execute();
}
```

```
/// <summary>
/// 非同期処理完了イベント。
/// </summary>
private void eventController1_ExecuteCompleted
    (object sender, ExecuteCompletedEventArgs e)
{
    // ワーカスレッドで実行したビジネスロジックで例外が発生
    if (e.Error != null)
    {
        // 例外に対する処理
    }
    else if (e.Cancelled)
    {
        // キャンセルされた時の処理
    }
    else if (e.ExecutionResult.Success)
    {
        // 成功時の処理
    }
    else
    {
        // 失敗時の処理
    }
}
```

FB-01 イベント処理機能 – 使用方法(6/6)

処理結果の判定方法

- イベントが正常に実行されたかどうかを判別する場合は、ExecutionResultのSuccessプロパティを確認する
  - 正常に完了した場合はSuccessプロパティにtrueが設定される
- イベントの実行に失敗した場合のエラー種別を判別する場合は、ExecutionResultのResultStringプロパティを確認する
  - ResultStringプロパティには、エラー内容に応じた文字列(イベント処理を実行した結果)を表す文字列が格納されているので、エラーハンドリング処理が可能となる

ExecutionResultのResultString値一覧

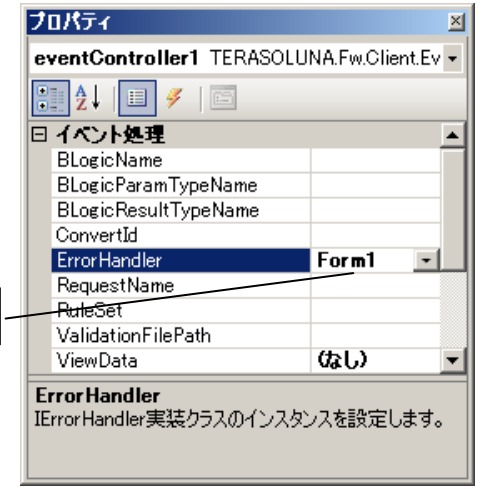
項番	状態	値
1	単項目チェックエラー	“validationError”
2	カスタムチェックエラー	“validationError”
3	前処理エラー	“preprocessedError”または前処理イベントで設定するResultStringの値
4	ビジネスロジック実行失敗	ビジネスロジック実行結果オブジェクトのResultStringの値
5	ビジネスロジック実行成功	“success”

FB-01 イベント処理機能 – 使用方法(6/6)

エラー処理の設定方法

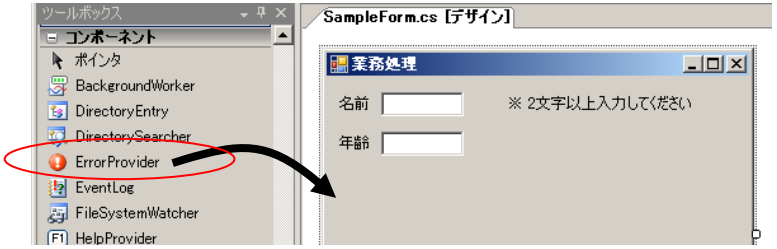
入力値検証エラーが発生したとき、画面にアイコンで通知する方法

① イベントコントローラのErrorHandlerプロパティにIErrorHandler実装クラスのインスタンスを設定する。通常は、FormBase派生クラスである画面クラスのインスタンスを設定する。



画面のインスタンス

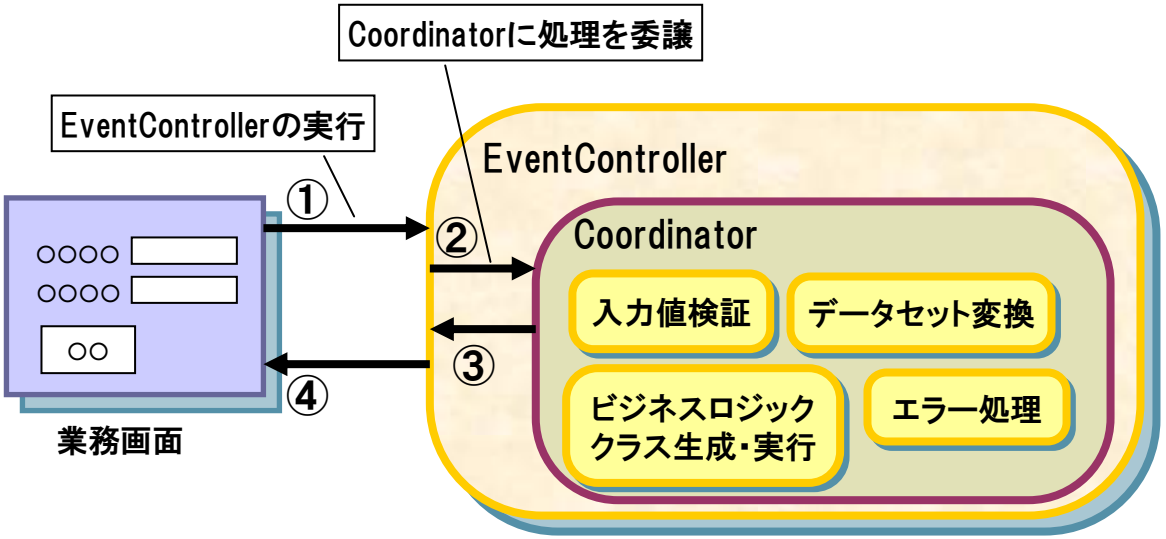
② デザイナでErrorProviderを画面に追加する



入力値検証エラーが発生するとアイコンが表示される

FB-01 イベント処理機能 – 拡張方法(1/2)

- EventControllerの内部処理
  - 画面から呼び出されたEventControllerは、内部ではCoordinatorを生成し、処理を委譲している。
  - Coordinatorを拡張することで内部処理を変更することができる



FB-01 イベント処理機能 – 拡張方法(2/2)

- EventControllerの内部処理の差し替え
  - アプリケーション構成ファイルのappSettingsセクションにCoordinatorを拡張したクラスの完全修飾名を指定することで、利用するCoordinatorの実装を差し替えることができる。

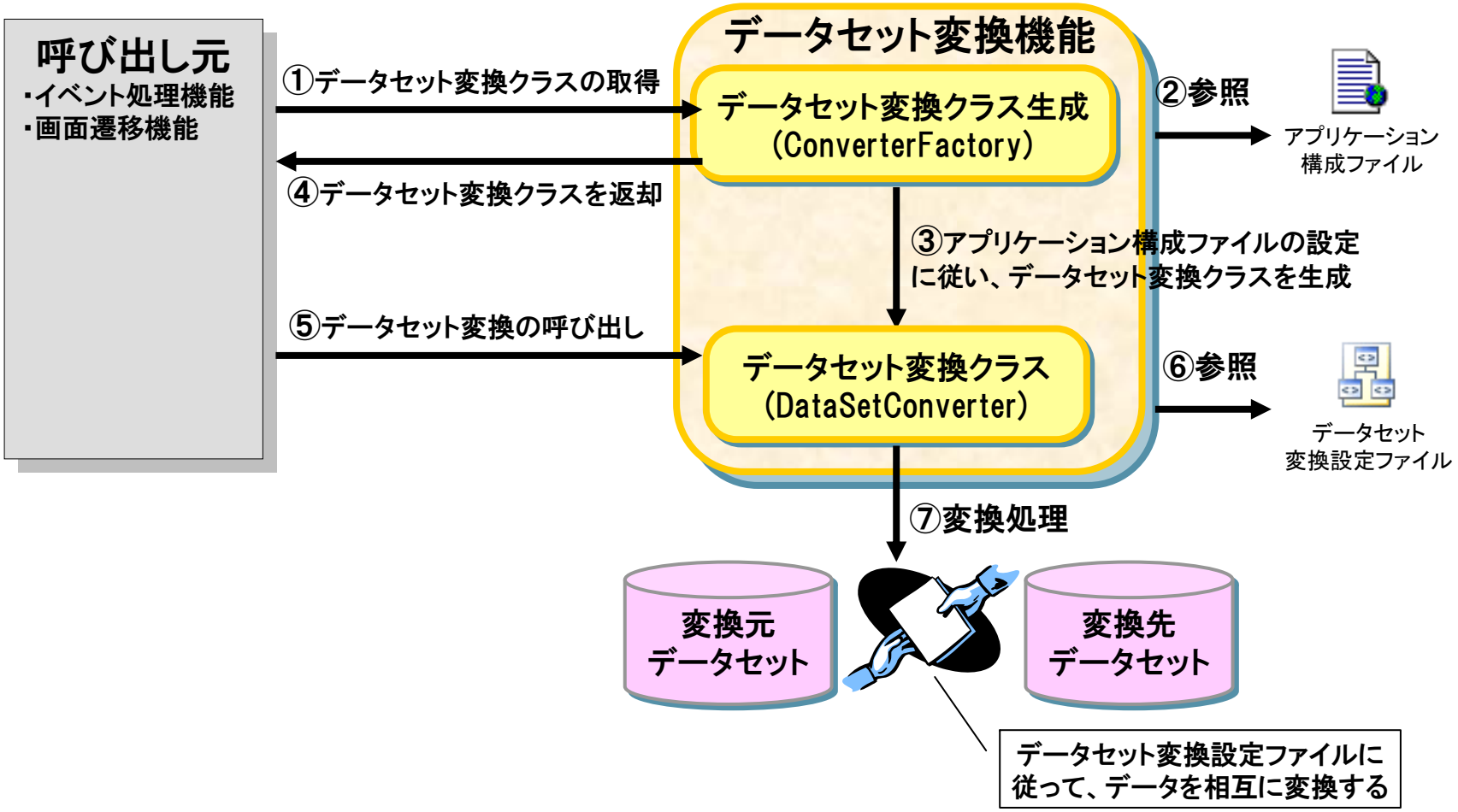
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="CoordinatorTypeName" value="Sample.CoordinatorEx, Sample"/>
  </appSettings>
</configuration>
```

FB-02 データセット変換機能 – 概要

- スキーマが異なる2つのデータセット間で、相互にデータを変換する機能を提供する
  - XML形式のデータセット変換設定ファイルに従って、変換処理を行う
  - この機能はイベント処理機能、画面遷移機能から呼び出される

FB-02 データセット変換機能 – 動作イメージ(1/5)

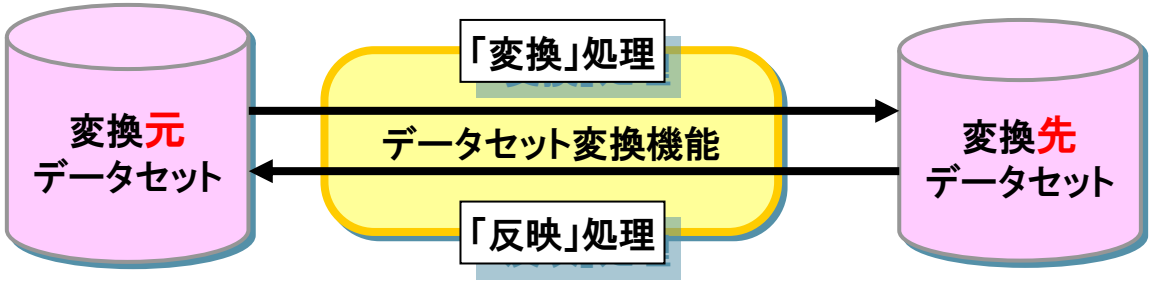
データセット変換機能を利用した際の処理フロー



FB-02 データセット変換機能 – 動作イメージ(2/5)

変換と反映

- 「変換元から変換先」を往路、「変換先から変換元」を復路としてデータの流れを区別している
- 往路を「変換」、復路を「反映」と呼ぶ
  - 「変換」処理の場合
    - 変換先データセットのテーブルは空でなければならない
  - 「反映」処理の場合
    - 変換元データセットのデータの扱いが2通りある
      - 変換元のデータを1度削除してから変換先のデータをコピーする
      - 変換先から変換元へデータを上書きする



FB-02 データセット変換機能 – 動作イメージ(3/5)

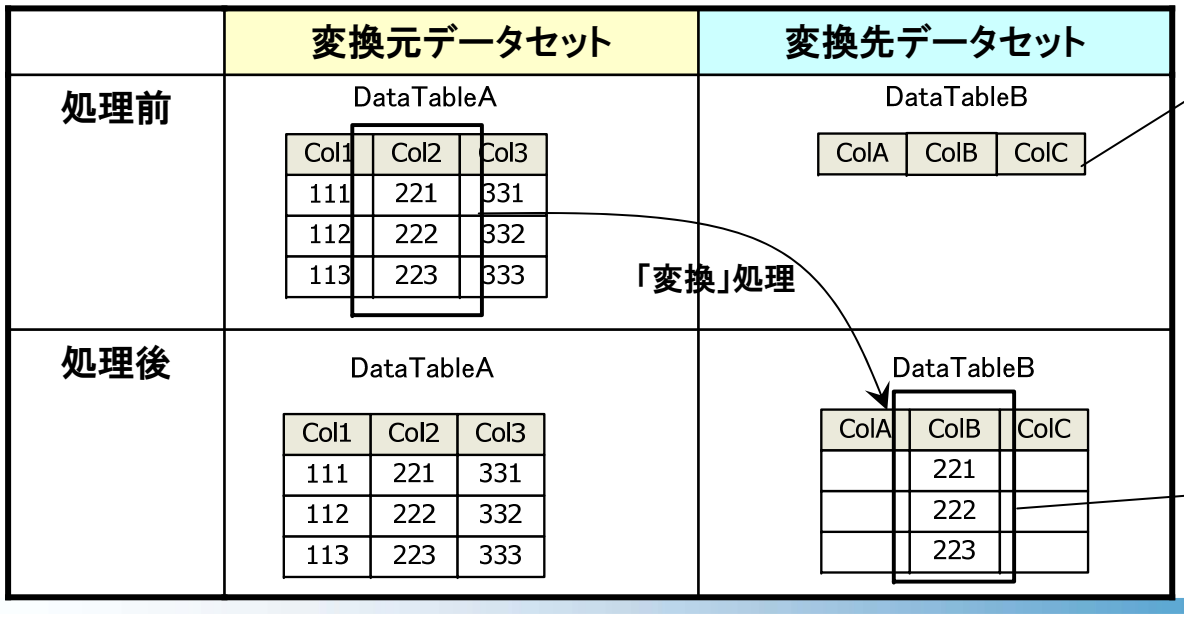
「変換」処理

変換元から変換先へデータをコピーする

データセット変換設定ファイル設定例

```
<convert id="foo">
  <param>
    <column src="DataTableA.Col2" dest="DataTableB.ColB"/>
  </param>
  <result/>
</convert>
```

DataTableAテーブルのCol2カラムのデータを
DataTableBテーブルのColBカラムにコピーする



FB-02 データセット変換機能 – 動作イメージ(4/5)

「反映」処理(書き換え)

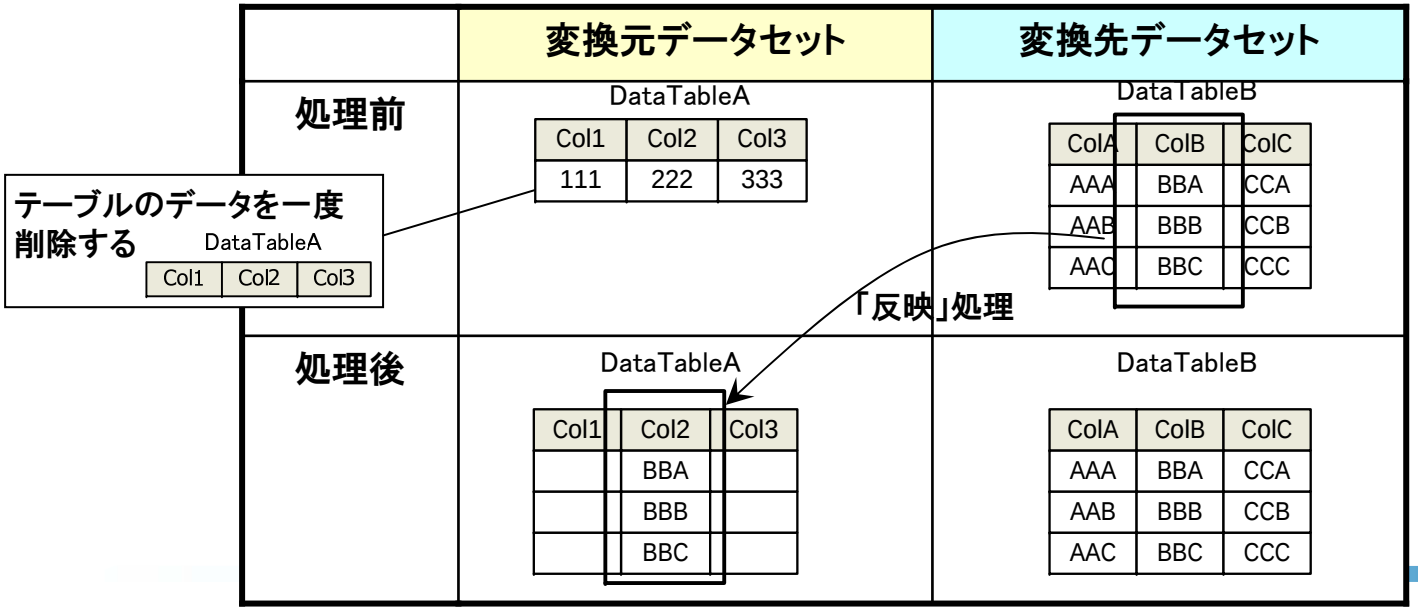
- 変換元のテーブルのデータを一度削除してから変換先のデータをコピーする
  - 変換元のテーブルと変換先のテーブルの行数が異なる場合はclear-view要素を指定する
  - データセット変換設定ファイルのclear-view要素に削除するテーブルを設定する

データセット変換設定ファイル設定例

```
<convert id="foo">
  <param/>
  <result>
    <clear-view table="DataTableA" />
    <column src="DataTableA.Col2" dest="DataTableB.ColB" />
  </result>
</convert>
```

データを削除するテーブルを clear-view要素で設定する

DataTableAテーブルのデータを一度削除してから、DataTableBテーブルのColBカラムのデータをDataTableAテーブルのCol2カラムにコピーする



FB-02 データセット変換機能 – 動作イメージ(5/5)

「反映」処理(上書き)

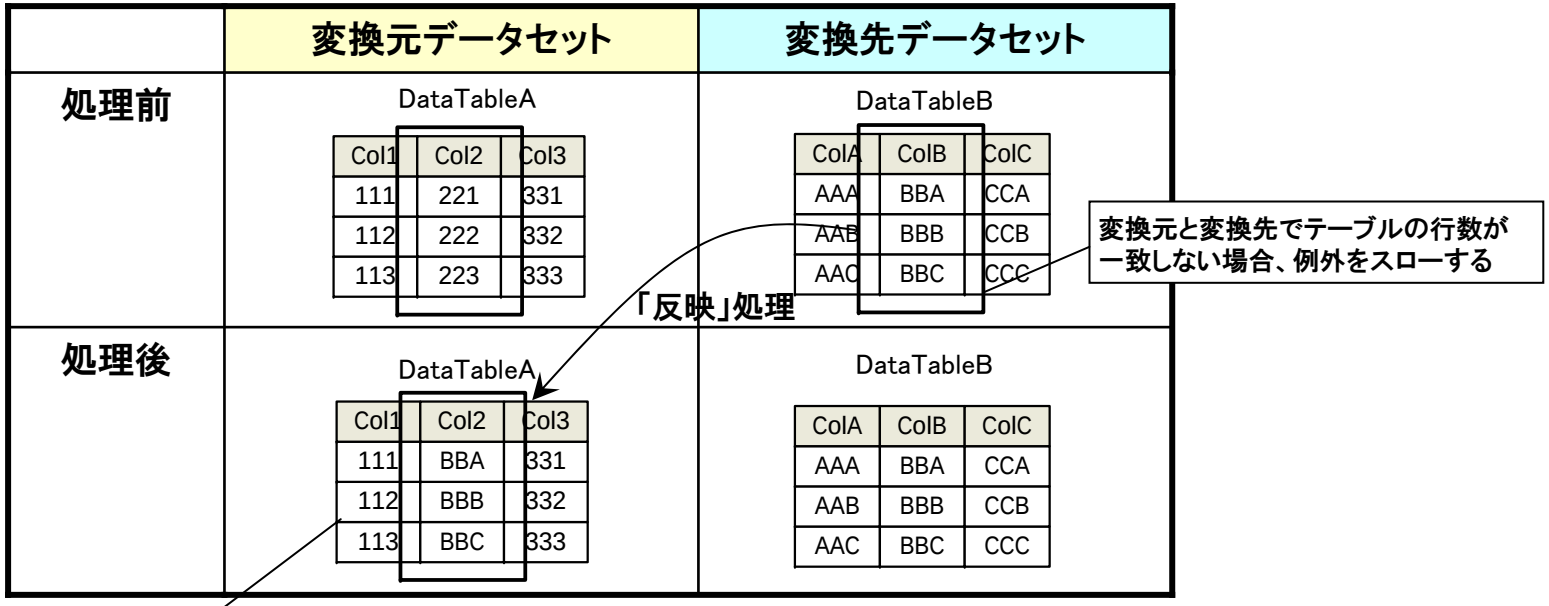
変換先から変換元へデータを上書きする

- 行数が同じ場合で、上書きしてよい場合は、clear-view要素を指定しない

データセット変換設定ファイル設定例

```
<convert id="foo">
  <result>
    <column src="DataTableA.Col2" dest="DataTableB.ColB" />
  </result>
</convert>
```

DataTableBテーブルのColBカラムのデータを
DataTableAテーブルのCol2カラムに上書きする



FB-02 データセット変換機能 – 使用例(1/4)

アプリケーション構成ファイルの設定

利用するデータセット変換設定ファイルのパスを設定する

- データセット変換設定ファイルは複数定義することができる

アプリケーション構成ファイル設定例

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="conversionConfiguration"
             type="TERASOLUNA.Fw.Client.Configuration.Conversion.ConversionConfigurationSection, TERASOLUNA.Fw.Client"/>
  </configSections>
  <conversionConfiguration>
    <files>
      <file path="config¥Converter1.config"/>
      <file path="config¥Converter2.config"/>
    </files>
  </conversionConfiguration>
</configuration>
```

データセット変換設定ファイルのパス

FB-02 データセット変換機能 – 使用例 (2/4)

データセット変換設定ファイルの設定

- 「変換」処理と「反映」処理の設定を定義する

コンバートIDを定義する。
全てのデータセット変換設定ファイル
の中で一意でなければならない

データセット変換設定ファイル設定例

```
<?xml version="1.0" encoding="utf-8" ?>
<conversionConfiguration xmlns="http://www.terasoluna.jp/schema/ConversionSchema.xsd">
  <convert id="getInfo">
    <param>
      <column src="InfoA.name" dest="InfoB.name" />
      <column src="InfoA.birth" dest="InfoB.birth" />
    </param>
    <result>
      <clear-view table ="InfoA"/>
      <column src="InfoA.name" dest="InfoB.name" />
      <column src="InfoA.birth" dest="InfoB.birth" />
    </result>
  </convert>
</conversionConfiguration>
```

param要素には「変換」処理の設定を定義する

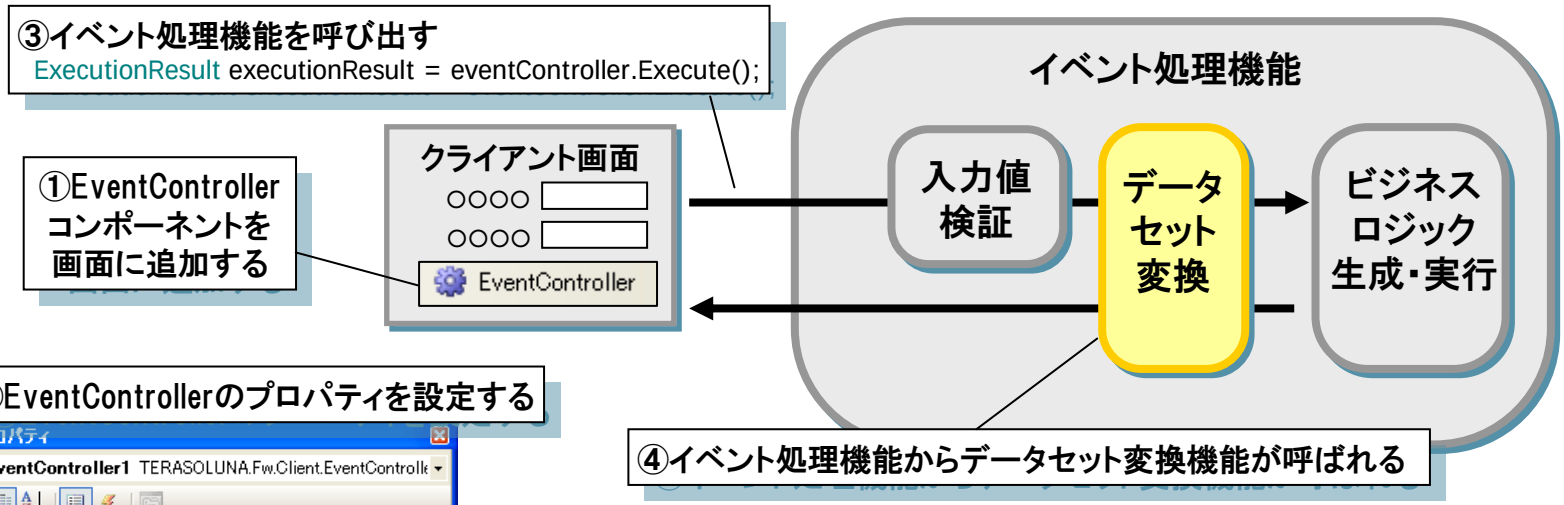
result要素には「反映」処理の設定を定義する

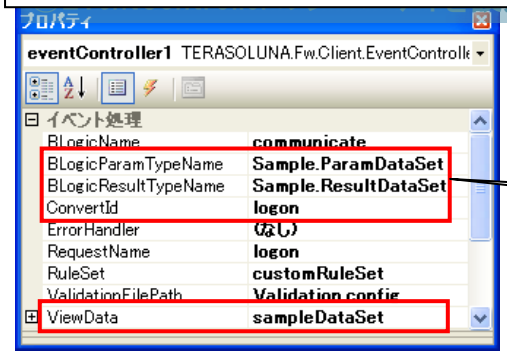
column要素
src属性 ：変換元データセットの[テーブル名.カラム名]
dest属性：変換先データセットの[テーブル名.カラム名]

FB-02 データセット変換機能 – 使用例 (3/4)

- 「イベント処理機能」からの呼び出し例

  EventControllerコンポーネントの ConvertId プロパティにデータセット変換設定ファイルに定義したコンバートIDを設定することで、データセット変換機能を利用できる

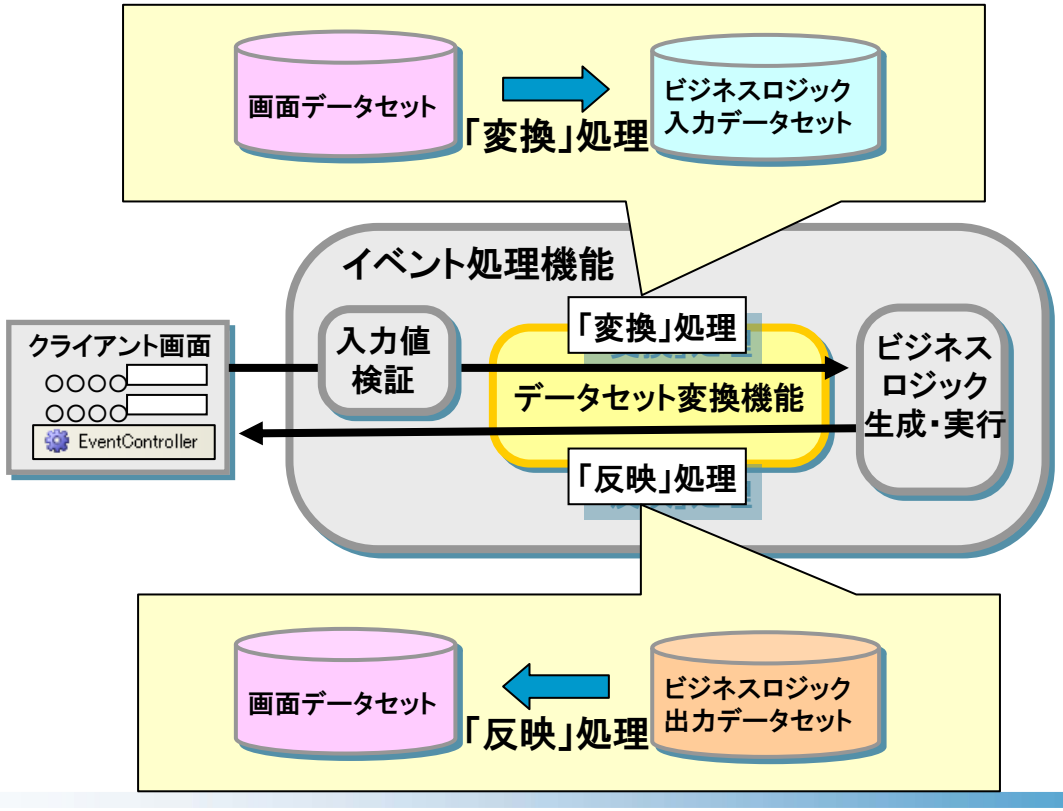


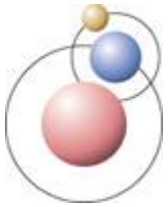


BLogicParamTypeName…ビジネスロジック入力情報として用いるデータセットの完全修飾名
BLogicResultTypeName…ビジネスロジック出力情報として用いるデータセットの完全修飾名
ConvertId…データセット変換設定ファイルに定義したコンバートID
ViewData…画面データセットのインスタンス

FB-02 データセット変換機能 – 使用例(4/4)

- 「イベント処理機能」からの呼び出し例
  - イベント処理機能ではデータセット変換が2回行われる
    1. 「変換」処理 【変換元】画面データセット 【変換先】ビジネスロジック入力データセット
    2. 「反映」処理 【変換元】画面データセット 【変換先】ビジネスロジック出力データセット





FB-02 データセット変換機能 – 拡張方法(1/2)

- 独自のデータセット変換クラスを利用したい場合は、データセット変換インターフェイス(IConverter)を実装したクラスを作成する
  - Convert(string id, DataSet srcData, DataSet destData)メソッド
    - コンバートID(id)に従って、変換元データセット(srcData)と変換先データセット(destData)に対する「変換」処理を実装する
  - Reverse(string id, DataSet srcData, DataSet destData)メソッド
    - コンバートID(id)に従って、変換元データセット(srcData)と変換先データセット(destData)に対する「反映」処理を実装する

FB-02 データセット変換機能 – 拡張方法(2/2)

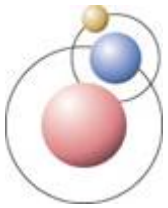
データセット変換クラスの差し替え

- アプリケーション構成ファイルのappSettingsセクションにIConverterを実装したクラスの完全修飾名を指定することで、利用するデータセット変換クラスの実装を差し替えることができる。

アプリケーション構成ファイルの設定例

```
<appSettings>
  <add key="ConverterTypeName"
       value="Sample.Converter, Sample"/>
 </appSettings>
```

appSettings要素に“ConverterTypeName”をキーとして、
データセット実装クラスの完全修飾名を記述する

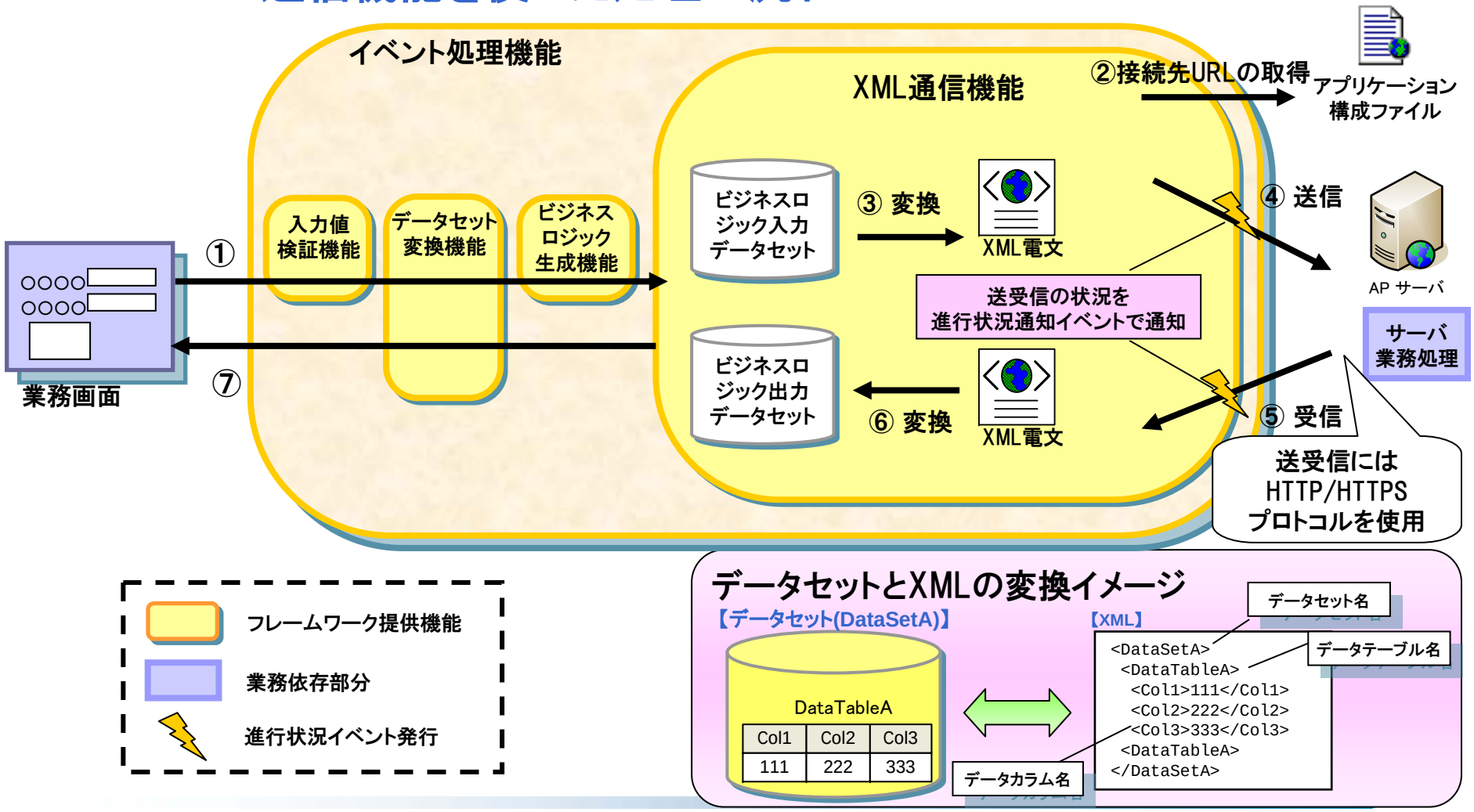


FC-01 XML通信機能 - 概要

- HTTP通信を行い、TERASOLUNA Server Framework for Java/.NETで開発されたサーバAPと連携してXML電文の送受信する機能を提供する
  - 本機能はイベント処理機能から呼び出される
  - 本機能が提供するXML通信クラスを直接利用して、XML通信処理を実行することも可能である

FC-01 XML通信機能 - 動作イメージ

XML通信機能を使った処理の流れ



FC-01 XML通信機能 – 使用例(1/4)

アプリケーション構成ファイルの設定

- 接続先URLとリクエストタイムアウト時間を設定する
  - ファイルアップロード機能・ファイルダウンロード機能も、同じ設定となる

アプリケーション構成ファイル設定例

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="BaseUrl" value="http://terasoluna.local/index.aspx"/>
    <add key="RequestTimeout" value="60000"/>
  </appSettings>
</configuration>
```

接続先URL

リクエストタイムアウト時間(ミリ秒)
設定を省略した場合の既定値は100,000(100秒)

- ビジネスロジック設定ファイルとデータセット変換設定ファイルのパスを設定する
  - 設定方法は以下を参照
    - CM-04 ビジネスロジック生成機能
    - FB-02 データセット変換機能

FC-01 XML通信機能 – 使用例(2/4)

ビジネスロジック設定ファイルの設定

- XML通信を行うビジネスロジックの「名前」と「クラスの完全修飾名」を設定する
  - 設定方法の詳細は「CM-04 ビジネスロジック生成機能」を参照

ビジネスロジック設定ファイル設定例

```
<?xml version="1.0" encoding="utf-8"?>
<blogicConfiguration xmlns="http://www.terasoluna.jp/schema/BLogicSchema.xsd">
  <blogic name="communicate"
          type="TERASOLUNA.Fw.Client.BLogic.DataSetXmlCommunicateBLogic`1, TERASOLUNA.Fw.Client" />
</blogicConfiguration>
```

ビジネスロジック名

ビジネスロジッククラスの完全修飾名

FC-01 XML通信機能 – 使用例(3/4)

データセット変換設定ファイルの設定

- 画面データセットからビジネスロジック入出力データセットへの変換設定を記述する
  - 設定方法の詳細は「FB-02 データセット変換機能」を参照

データセット変換設定ファイル設定例

```
<?xml version="1.0" encoding="utf-8" ?>
<conversionConfiguration xmlns="http://www.terasoluna.jp/schema/ConversionSchema.xsd">
  <convert id="getInfo">
    <param>
      <column src="InfoA.name" dest="InfoB.name" />
      <column src="InfoA.birth" dest="InfoB.birth" />
    </param>
    <result>
      <clear-view table ="InfoA"/>
      <column src="InfoA.name" dest="InfoB.name" />
      <column src="InfoA.birth" dest="InfoB.birth" />
    </result>
  </convert>
</conversionConfiguration>
```

画面データセットからXMLとしてサーバへ送信したいデータのみを、ビジネスロジック入力データセットに設定

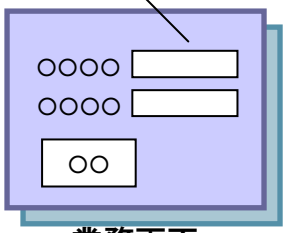
サーバから受信したXMLの中から、画面データセットへ反映したいデータのみを指定

FC-01 XML通信機能 – 使用例(4/4)

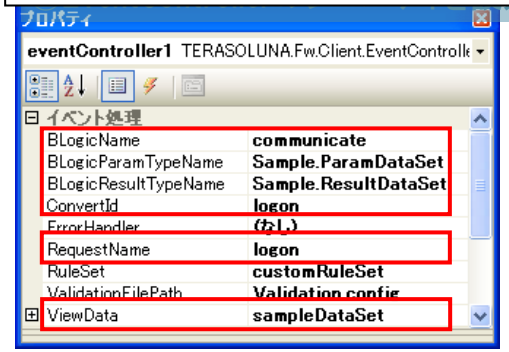
「イベント処理機能」からの呼び出し例

- EventControllerコンポーネントのプロパティにXML通信機能を利用するための設定を行う

① コンポーネントを画面に追加する



②EventControllerのプロパティを設定する

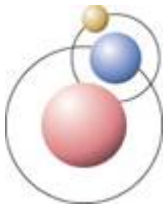


③ イベント処理機能を呼び出す

```
ExecutionResult executionResult = eventController1.Execute();
```

<< XML通信機能に関わるプロパティ >>

プロパティ名	説明
BLogicName	ビジネスロジック設定ファイルに定義したXML通信を行うビジネスロジックの名前
BLogicParamTypeName	ビジネスロジック入力データセットの完全修飾名
BLogicResultTypeName	ビジネスロジック出力データセットの完全修飾名
ConvertId	データセット変換設定ファイル定義したコンバートID
RequestName	サーバで動作する処理を特定するための識別子
ViewData	画面データセットのインスタンス

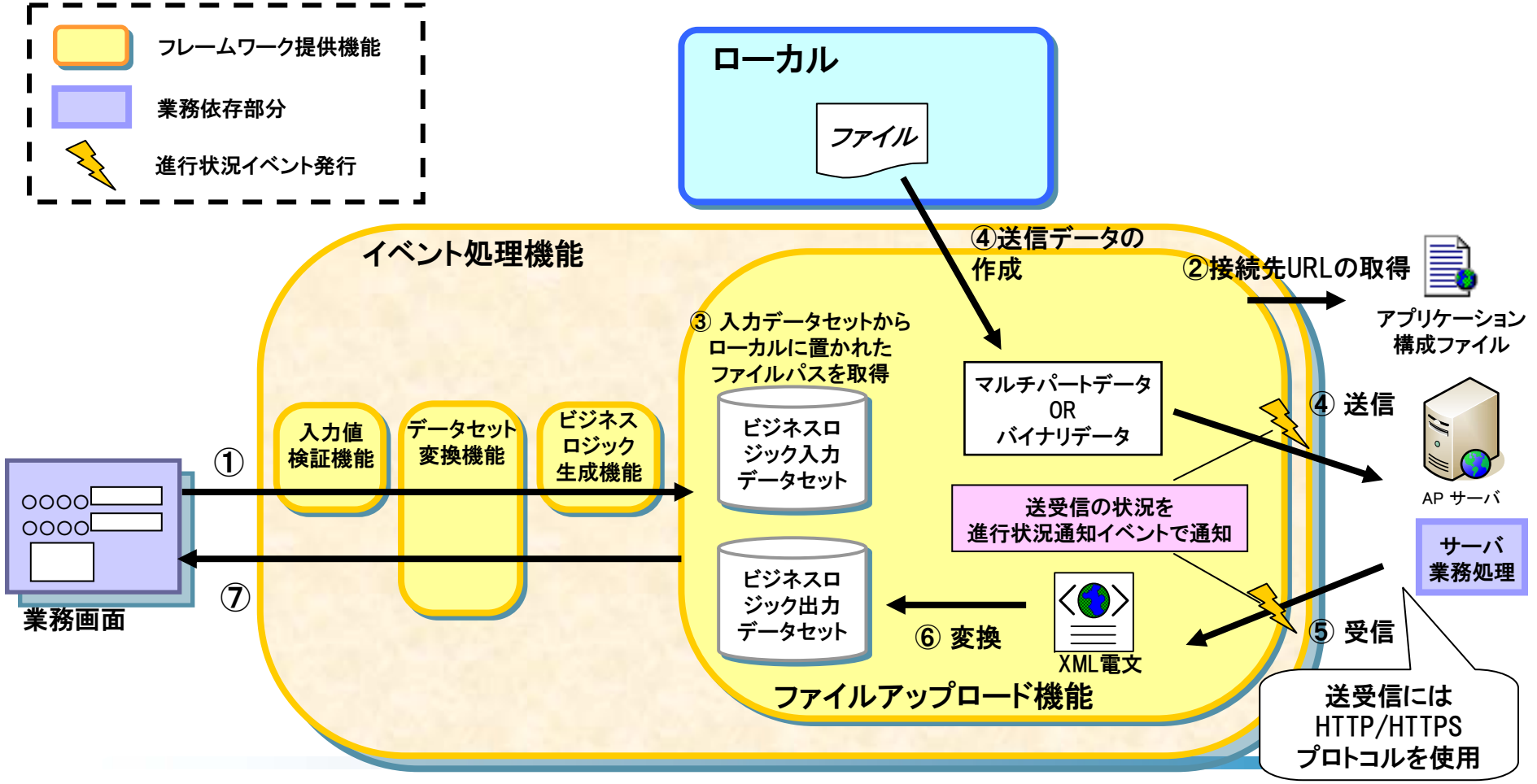


FC-02 ファイルアップロード機能 - 概要

- HTTP通信を行い、TERASOLUNA Server Framework for Java/.NETで開発されたサーバAPと連携してファイルアップロードを行うための仕組みを提供する
  - ローカルに置かれたファイルを、マルチパート形式またはバイナリ形式でアップロードし、レスポンスとしてXML形式のデータを受け取る
  - マルチパートアップロード機能では、複数ファイルのアップロードができる
  - 本機能はイベント処理機能から呼び出される
  - 本機能が提供するファイルアップロード用通信クラスを直接利用して、処理を実行することも可能である

FC-02 ファイルアップロード機能 - 動作イメージ

ファイルアップロード機能を使った処理の流れ



FC-02 ファイルアップロード機能 – 使用例 (1/5)

アプリケーション構成ファイルの設定

- 接続先URLとリクエストタイムアウト時間を設定する
  - XML通信機能・ファイルダウンロード機能も、同じ設定となる

アプリケーション構成ファイル設定例

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="BaseUrl" value="http://terasoluna.local/index.aspx"/>
    <add key="RequestTimeout" value="60000"/>
  </appSettings>
</configuration>
```

接続先URL

リクエストタイムアウト時間(ミリ秒)
設定を省略した場合の既定値は100,000(100秒)

- ビジネスロジック設定ファイルとデータセット変換設定ファイルのパスを設定する
  - 設定方法は以下を参照のこと
    - CM-04 ビジネスロジック生成機能
    - FB-02 データセット変換機能

FC-02 ファイルアップロード機能 – 使用例 (2/5)

ビジネスロジック設定ファイルの設定

- ファイルアップロードを行うビジネスロジックの「名前」と「クラスの完全修飾名」を設定する
  - 設定方法の詳細は「CM-04 ビジネスロジック生成機能」を参照

ビジネスロジック設定ファイル設定例(マルチパート形式)

```
<?xml version="1.0" encoding="utf-8"?>
<blogicConfiguration xmlns="http://www.terasoluna.jp/schema/BLogicSchema.xsd">
  <blogic name="multipartUpload"
          type="TERASOLUNA.Fw.Client.BLogic.MultipartUploadBLogic`1, TERASOLUNA.Fw.Client" />
</blogicConfiguration>
```

ビジネスロジック名

ビジネスロジッククラスの完全修飾名

ビジネスロジック設定ファイル設定例(バイナリ形式)

```
<?xml version="1.0" encoding="utf-8"?>
<blogicConfiguration xmlns="http://www.terasoluna.jp/schema/BLogicSchema.xsd">
  <blogic name="binaryUpload"
          type="TERASOLUNA.Fw.Client.BLogic.BinaryFileUploadBLogic`1, TERASOLUNA.Fw.Client" />
</blogicConfiguration>
```

ビジネスロジック名

ビジネスロジッククラスの完全修飾名

FC-02 ファイルアップロード機能 – 使用例 (3/5)

データセット変換設定ファイルの設定(マルチパート形式)

- 画面データセットとビジネスロジック入出力データセットへの変換設定を記述する
  - ビジネスロジック入力データセットに必要な情報の詳細は「FC-02 ファイルアップロード機能」を参照
  - 設定方法の詳細は「FB-02 データセット変換機能」を参照

データセット変換設定ファイル設定例

```
<?xml version="1.0" encoding="utf-8" ?>
<conversionConfiguration xmlns="http://www.terasoluna.jp/schema/ConversionSchema.xsd">
 <convert id="multipartUpload">
  <param>
   <column src="UploadFile.Name" dest="File.Name" />
   <column src="UploadFile.FilePath" dest="File.FilePath" />
   <column src="UploadText.Name" dest="Text.Name" />
   <column src="UploadText.Value" dest="Text.Value" />
  </param>
  <result>
   <column src="ResultTable.Result" />
  </result>
 </convert>
</conversionConfiguration>
```

マルチパート形式のアップロードに必要な情報をビジネスロジック入力データセットに設定

アップロード結果を格納するビジネスロジック出力データセットの設定

FC-02 ファイルアップロード機能 – 使用例 (4/5)

データセット変換設定ファイルの設定(バイナリ形式)

- 画面データセットとビジネスロジック入出力データセットへの変換設定を記述する
  - ビジネスロジック入力データセットに必要な情報の詳細は「FC-02 ファイルアップロード機能」を参照
  - 設定方法の詳細は「FB-02 データセット変換機能」を参照

データセット変換設定ファイル設定例

```
<?xml version="1.0" encoding="utf-8" ?>
<conversionConfiguration xmlns="http://www.terasoluna.jp/schema/ConversionSchema.xsd">
 <convert id="binaryUpload">
  <param>
   <column src="UploadFile.FilePath" dest="File.FilePath" />
  </param>
  <result>
   <column src="ResultTable.Result" />
  </result>
 </convert>
</conversionConfiguration>
```

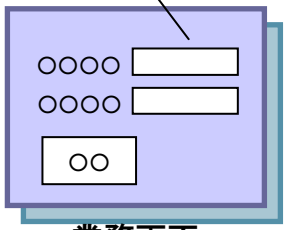
バイナリ形式のアップロードに必要な情報をビジネスロジック入力データセットに設定

アップロード結果を格納するビジネスロジック出力データセットの設定

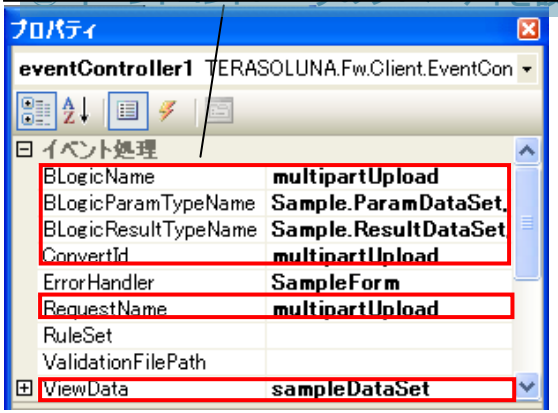
FC-02 ファイルアップロード機能 – 使用例 (5/5)

「イベント処理機能」からの呼び出し例

① コンポーネントを画面に追加する



② イベントコントローラのプロパティを設定する

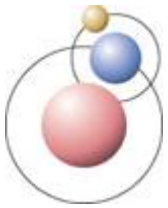


③ イベント処理機能を呼び出す

```
ExecutionResult executionResult = eventController1.Execute();
```

<< ファイルアップロード機能に関わるプロパティ >>

プロパティ名	説明
BLogicName	ビジネスロジック設定ファイルに定義したマルチパート形式またはバイナリ形式でアップロードを行うビジネスロジックの名前
BLogicParamTypeName	ビジネスロジック入力データセットの完全修飾名（省略可能）
BLogicResultTypeName	ビジネスロジック出力データセットの完全修飾名（省略可能）
ConvertId	データセット変換設定ファイル定義したコンバートID
RequestName	サーバで動作する処理を特定するための識別子
ViewData	画面データセットのインスタンス

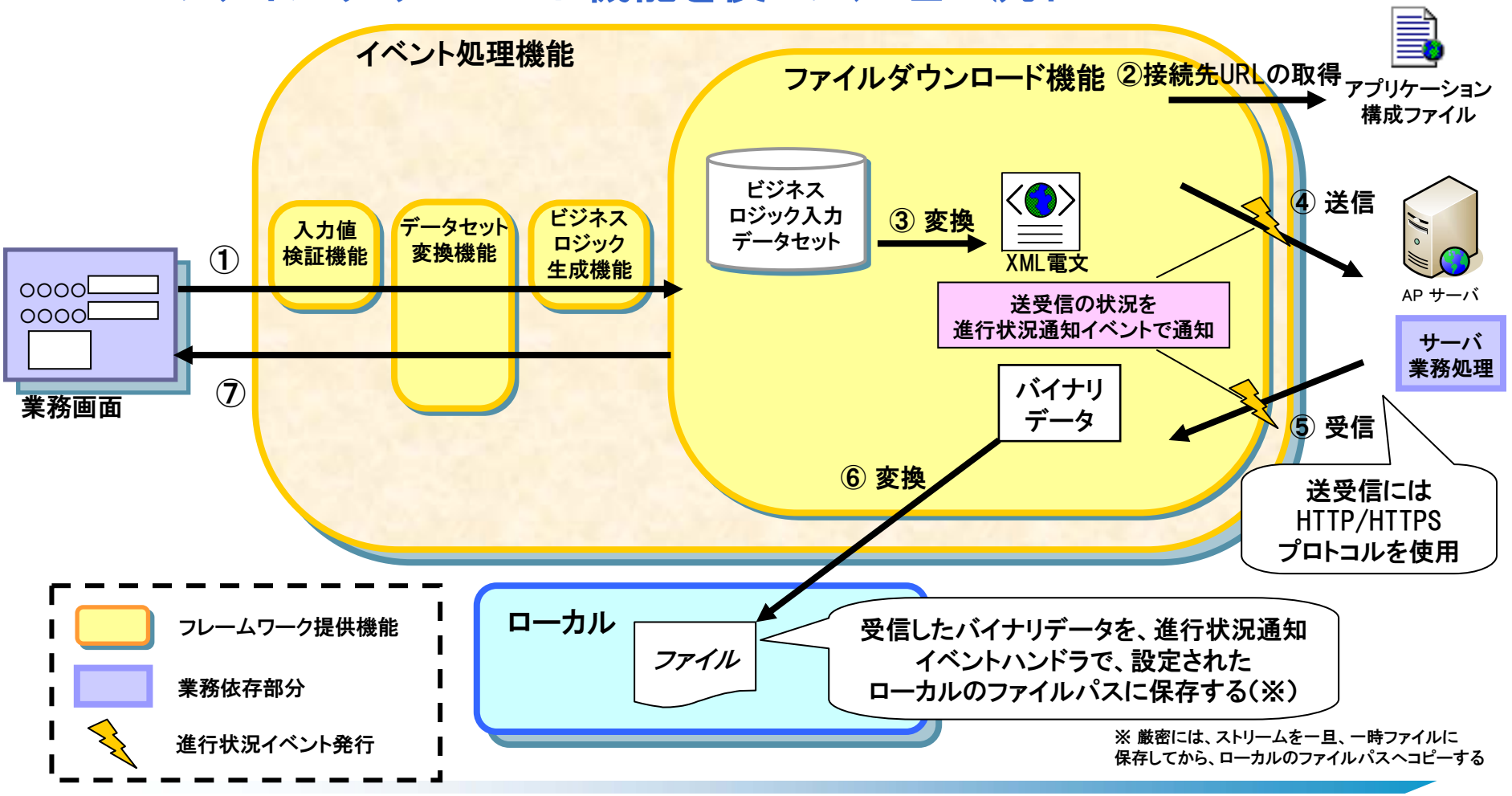


FC-03 ファイルダウンロード機能 - 概要

- HTTP通信を行い、TERASOLUNA Server Framework for Java/.NETで開発されたサーバAPと連携してファイルダウンロードを行うための仕組みを提供する
  - リクエストはXML形式であり、ファイルダウンロードした際の受信データはファイルとして保存する
  - 本機能はイベント処理機能から呼び出される
  - 本機能が提供するファイルダウンロード用通信クラスを直接利用して、処理を実行することも可能である

FC-03 ファイルダウンロード機能 - 動作イメージ

ファイルダウンロード機能を使った処理の流れ



FC-03 ファイルダウンロード機能 – 使用例(1/5)

アプリケーション構成ファイルの設定

- 接続先URLとリクエストタイムアウト時間を設定する
  - XML通信機能・ファイルアップロード機能も、同じ設定となる

アプリケーション構成ファイル設定例

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <appSettings>
    <add key="BaseUrl" value="http://terasoluna.local/index.aspx"/>
    <add key="RequestTimeout" value="60000"/>
  </appSettings>
</configuration>
```

接続先URL

リクエストタイムアウト時間(ミリ秒)
設定を省略した場合の既定値は100,000(100秒)

- ビジネスロジック設定ファイルとデータセット変換設定ファイルのパスを設定する
  - 設定方法は以下を参照のこと
    - CM-04 ビジネスロジック生成機能
    - FB-02 データセット変換機能

FC-03 ファイルダウンロード機能 – 使用例 (2/5)

ビジネスロジック設定ファイルの設定

- XML通信を行うビジネスロジックの「名前」と「クラスの完全修飾名」を設定する
  - 設定方法の詳細は「CM-04 ビジネスロジック生成機能」を参照

ビジネスロジック設定ファイル設定例

```
<?xml version="1.0" encoding="utf-8"?>
<blogicConfiguration xmlns="http://www.terasoluna.jp/schema/BLogicSchema.xsd">
  <blogic name="download"
          type="TERASOLUNA.Fw.Client.BLogic.BinaryFileDownloadBLogic, TERASOLUNA.Fw.Client" />
</blogicConfiguration>
```

ビジネスロジック名

ビジネスロジッククラスの完全修飾名

FC-03 ファイルダウンロード機能 – 使用例 (3/5)

データセット変換設定ファイルの設定

- 画面データセットからビジネスロジック入力データセットへの変換設定を記述する
  - 設定方法の詳細は「FB-02 データセット変換機能」を参照

データセット変換設定ファイル設定例

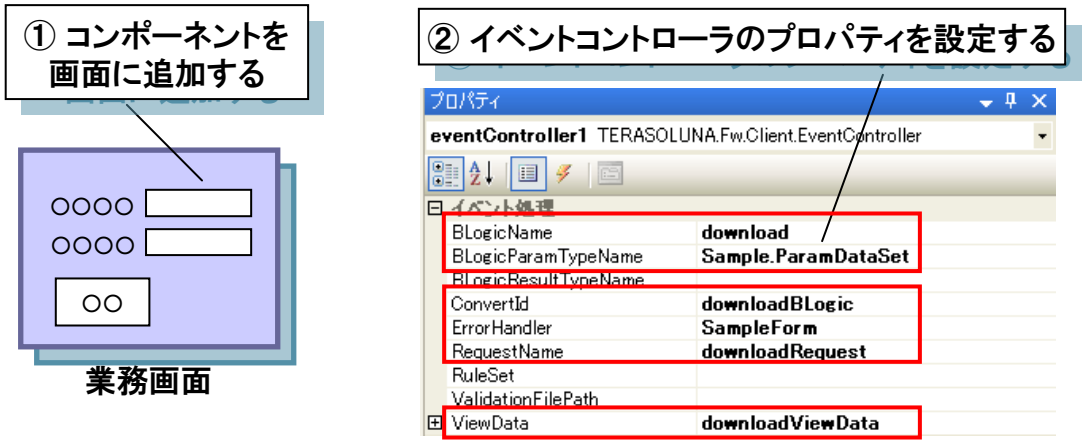
```
<?xml version="1.0" encoding="utf-8" ?>
<conversionConfiguration xmlns="http://www.terasoluna.jp/schema/ConversionSchema.xsd">
  <convert id="getInfo">
    <param>
      <column src="InfoA.name" dest="InfoB.name" />
      <column src="InfoA.birth" dest="InfoB.birth" />
    </param>
    <result>
    </result>
  </convert>
</conversionConfiguration>
```

result要素は空要素

画面データセットからXMLとしてサーバへ送信したいデータのみを、ビジネスロジック入力データセットに設定

FC-03 ファイルダウンロード機能 – 使用例(4/5)

「イベント処理機能」からの呼び出し例



<< XML通信機能に関わるプロパティ >>

プロパティ名	説明
BLogicName	ビジネスロジック設定ファイルに定義したファイルダウンロードを行うビジネスロジックの名前
BLogicParamTypeName	ビジネスロジック入力データセットの完全修飾名(省略可能)
ConvertId	データセット変換設定ファイル定義したコンバートID
RequestName	サーバで動作する処理を特定するための識別子
ViewData	画面データセットのインスタンス

FC-03 ファイルダウンロード機能 – 使用例(5/5)

「イベント処理機能」からの呼び出し例

③ 進行状況通知イベントハンドラを実装する

- ファイルダウンロード機能を利用するときは、進行状況通知イベントハンドラを利用して、ダウンロードファイルパスを設定する
  - イベントオブジェクトのItemsプロパティのキーに“DownloadReady”が含まれていた場合“DownloadFilePath”キーの値へダウンロードファイルパスを設定する

```
private void eventController1_ExecuteProgressChanged(object sender, ExecuteProgressChangedEventArgs e)
{
    if (e.Items.Contains("DownloadReady")
    {
        saveFileDialog.Title = "ダウンロードファイルパスの指定";
        saveFileDialog.FileName = e.Items["ContentFileName"] as string;
        if (DialogResult.OK == saveFileDialog.ShowDialog())
        {
            e.Items.Add("DownloadFilePath", saveFileDialog.FileName);
        }
    }
}
```

DownloadReadyキーが含まれていた場合、ダウンロードファイルパスを設定する

サーバから返却されたファイル名を取得することもできる

DownloadFilePathキーの値へダウンロードファイルパスを設定する

④ イベント処理機能を呼び出す

```
ExecutionResult executionResult = eventController1.Execute();
```