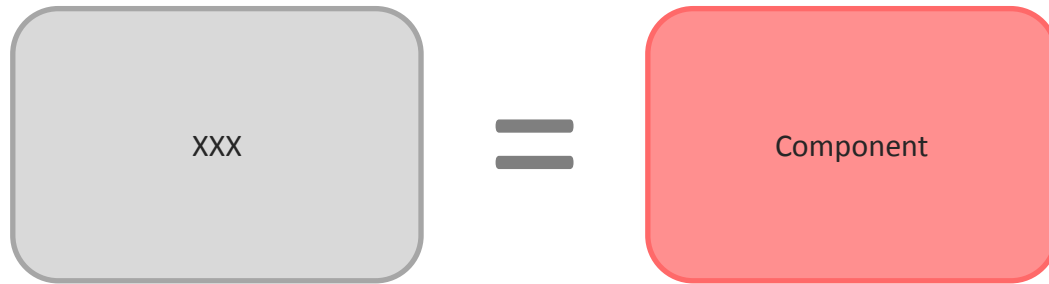


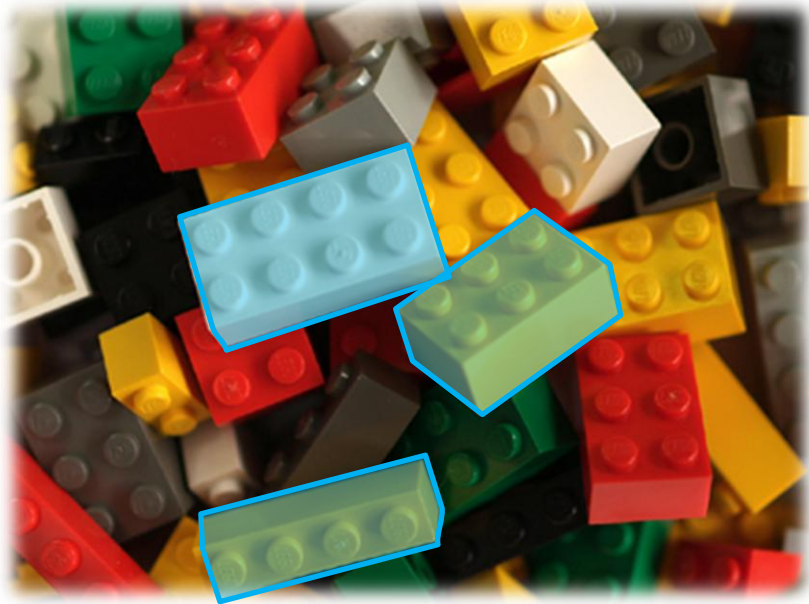
Custom XXX のすすめ

脱中級者をめざして

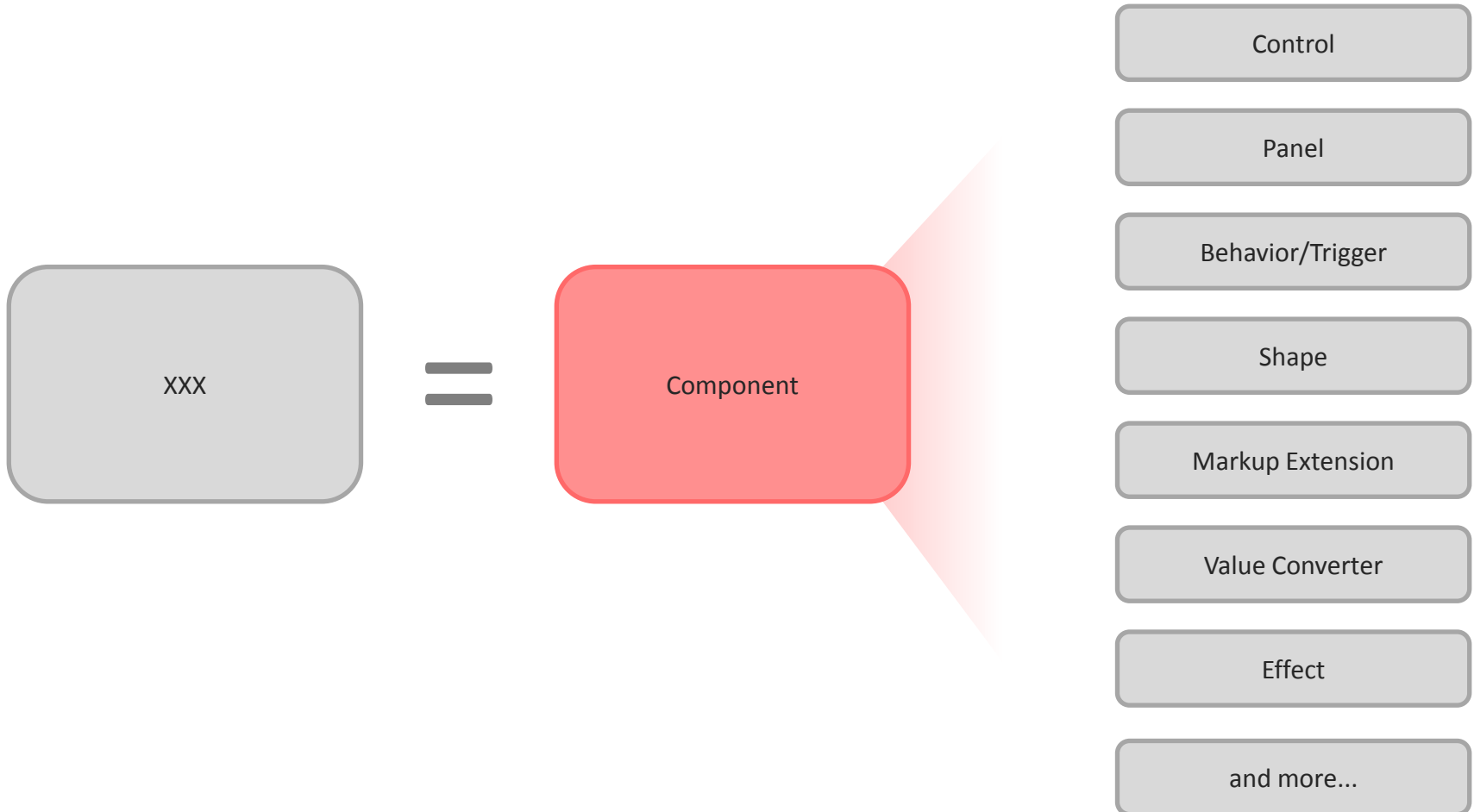
Custom XXX

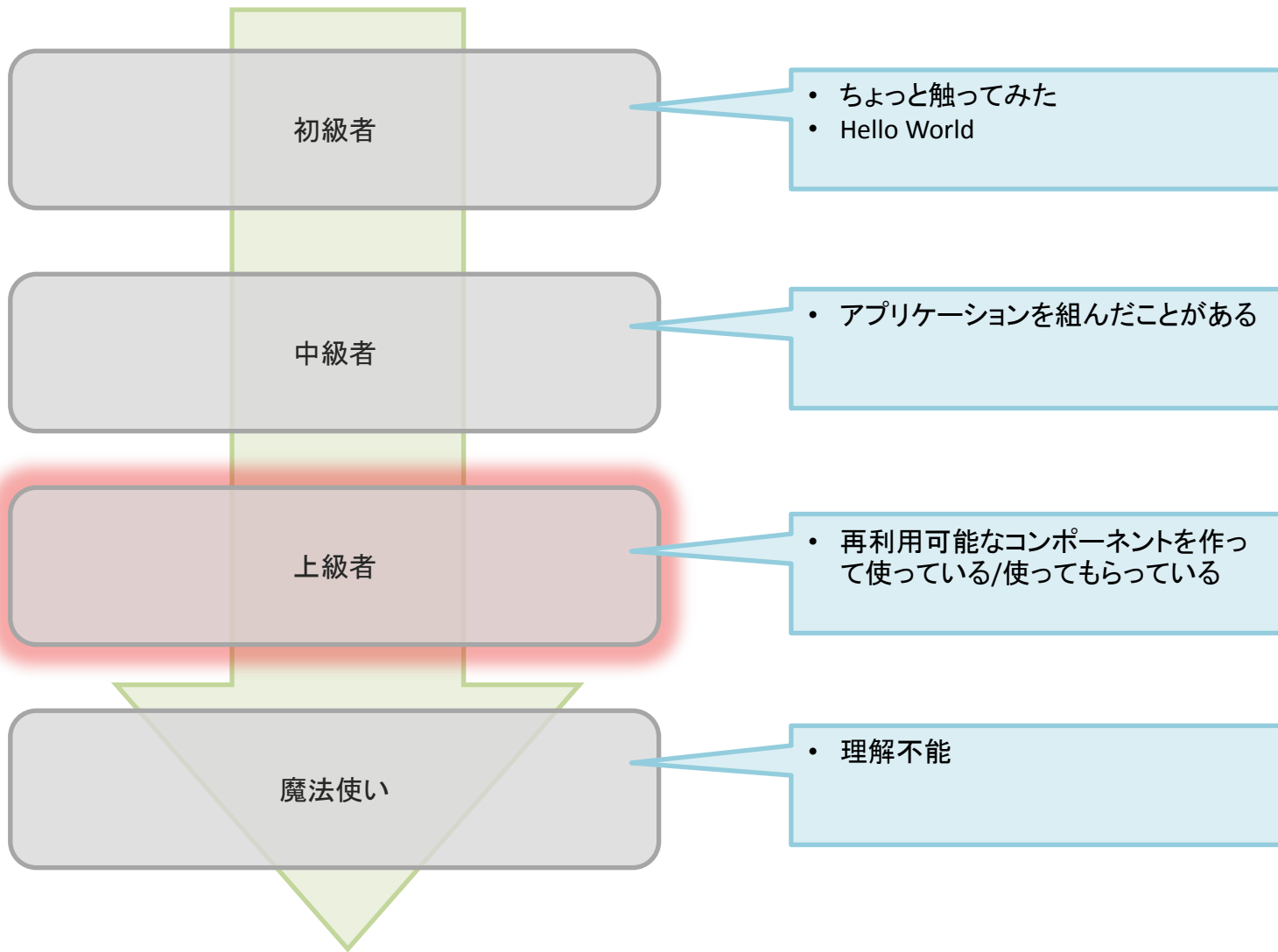


Component

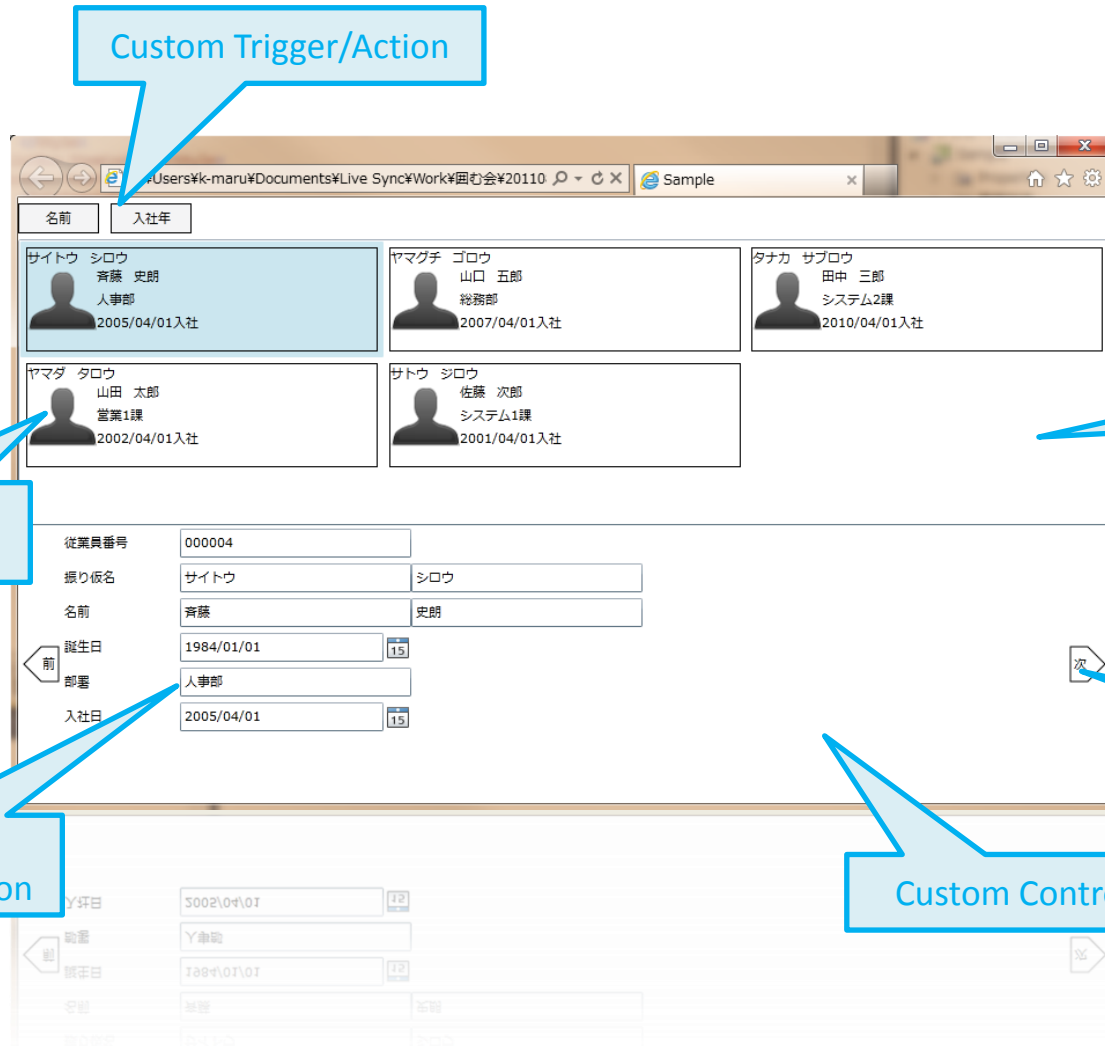


Custom XXX for Silverlight



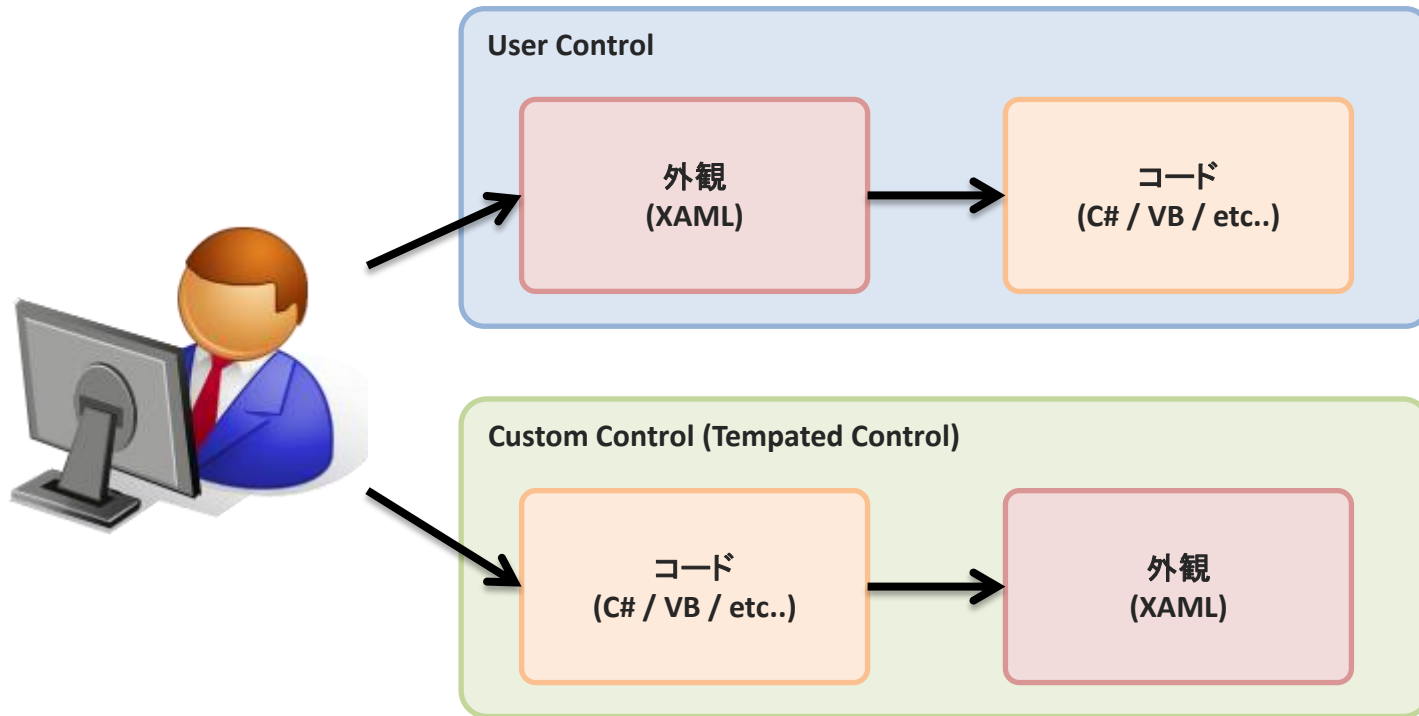


Example



CUSTOM CONTROL

CUSTOM CONTROL



※あくまでイメージです

CUSTOM CONTROL

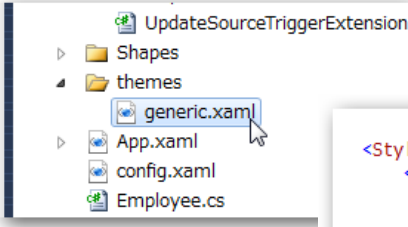
- つくりどころ/なにがうれしいのか
 - 使うときにTemplateを差し替えられるので、“画面部品としての機能”を再利用できる
 - 特定の要件に従った機能を複数の画面、プロジェクトで使いまわしたい
 - 見た目は利用個所ごとに変更したい
 - Toolkit等でコントロールは豊富にあるので、あまり単機能のコントロールを作る機会が少ないが...
- 関連する技術
 - Template/Style
 - Visual State Manager
 - Data Binding
 - etc...

作り方

1. themesフォルダ以下にgeneric.xamlを配置して、
その中にTemplateを定義
 - お約束
 - MergedDictionaryでgeneric.xamlに取り込むようにすることでコントロール単位でTemplate定義のxamlファイルをわけることができる
2. Controlクラスを継承する
3. コンストラクターでDefaultStyleKeyを指定する
4. OnApplyTemplateでTemplateから、Template内に配置されている要素を取得し、操作準備を整える

作り方

① Themesフォルダ以下にgeneric.xamlを配置して、その中にTemplateを定義



```
<Style TargetType="local:PagingContentControl">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="local:PagingContentControl">
        <Grid>
          <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="Auto" />
          </Grid.ColumnDefinitions>
          <Button x:Name="PreviousButton" Content="前" Grid.Column=
            <Button.Style>
              <Style TargetType="Button">
```

```
public PagingContentControl() {
  this.DefaultStyleKey = typeof(PagingContentControl);
  this.Pages = new DependencyObjectCollection<UIElement>();
}
```

② コンストラクターで DefaultStyleKeyを指定する

③ OnApplyTemplateでTemplateから、Template内に配置されている要素を取得し、操作準備を整える

```
public override void OnApplyTemplate() {
  if (templateApplied) {
    return;
  }
  base.OnApplyTemplate();

  this.previousButton = this.GetTemplateChild("PreviousButton") as ButtonBase;
  this.nextButton = this.GetTemplateChild("NextButton") as ButtonBase;
  this.contentHolder = this.GetTemplateChild("ContentHolder") as ContentControl;
```

注意点

- 必ずNULLチェック
 - Templateを使う側で書き換え可能なため、Template内で使おうとしてる要素が定義されていない可能性がある
 - 定義されていない場合は、例外を投げるのではなく、「動かない(処理を抜ける)」ようにする
- 利用する要素の型はできる限り基底を利用する
 - 例えばButtonBaseを利用するようにしておくと、使う側はButton, HyperLink, RadioButton, etc をTemplateに指定できる

CUSTOM PANEL

CUSTOM PANEL

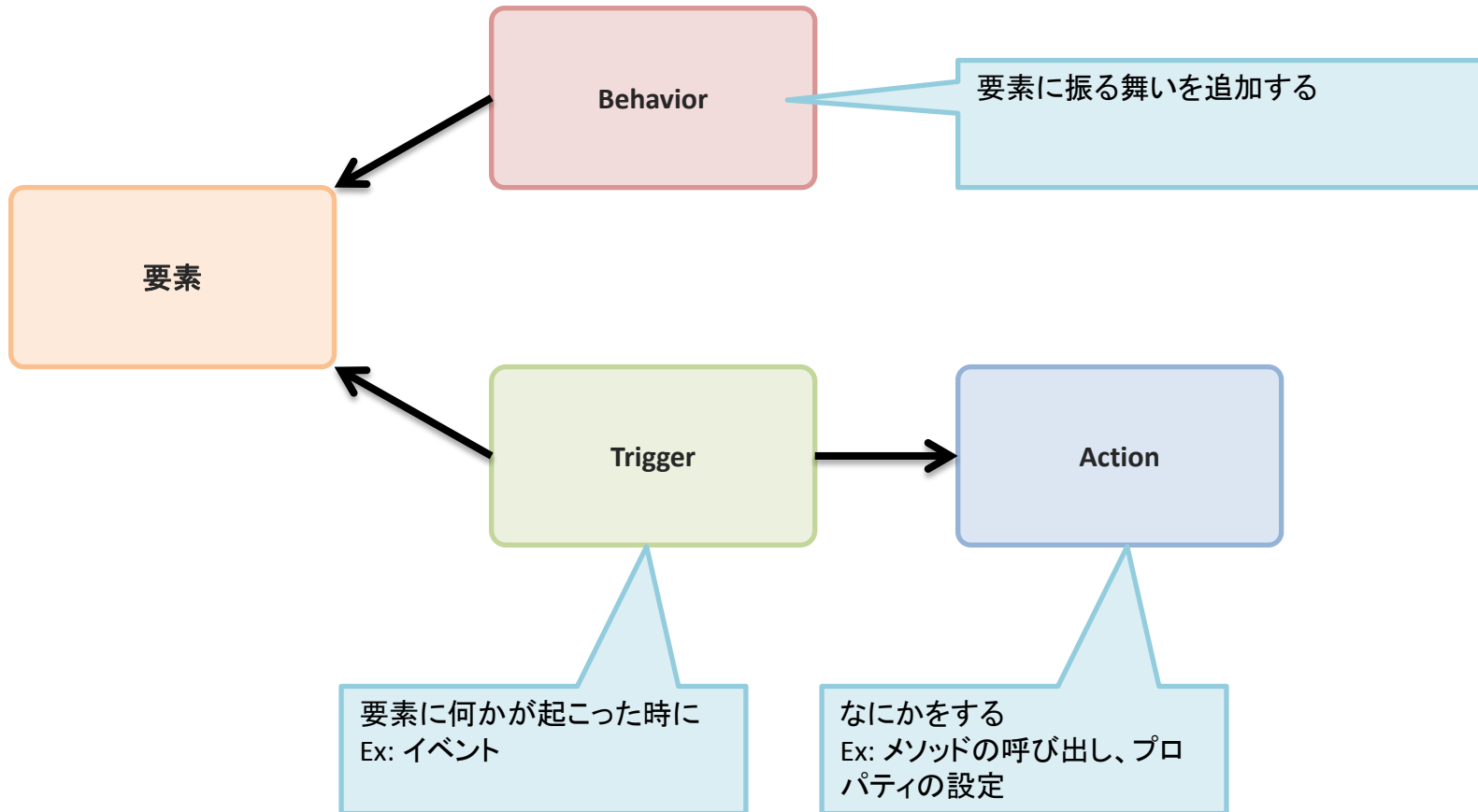
- つくりどころ/なにがうれしいのか
 - 標準で提供されているパネルで実現できない配置を行いたい

作り方

1. Panelを継承する
2. MeasureOverride をオーバーライドして、サイズを測定する
 - 子要素のサイズ、位置の指定
 - パネルが必要とするサイズ
3. ArrangeOverrideをオーバーライドして、サイズを決定する
 - 子要素のサイズ、位置の指定
 - パネルの最終的なサイズ

CUSTOM TRIGGER / ACTION

CUSTOM TRIGGER / ACTION



CUSTOM TRIGGER / ACTION

- つくりどころ/なにがうれしいのか
 - TRIGGER を作ることはほとんどない
 - 個人的にはEventTrigger以外ほとんど使ったことがない
 - Actionはよく作る
 - XAML上で宣言的に処理を追加することができる
 - MVVM実現するには必須

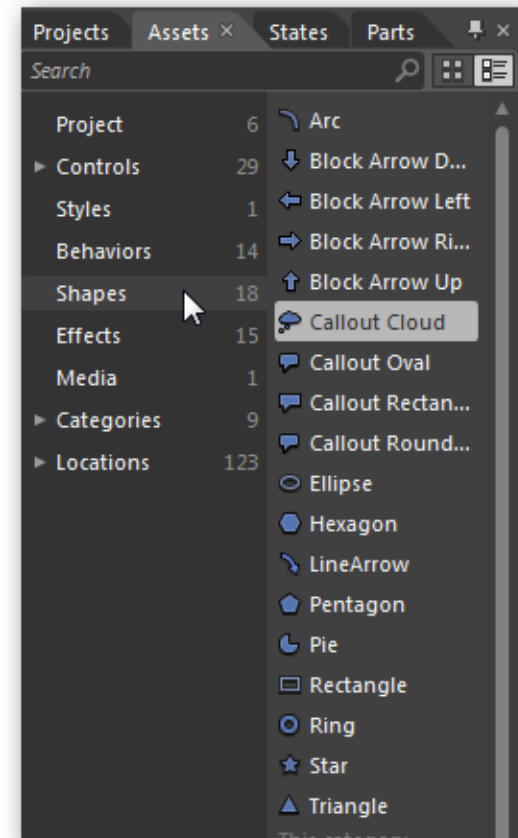
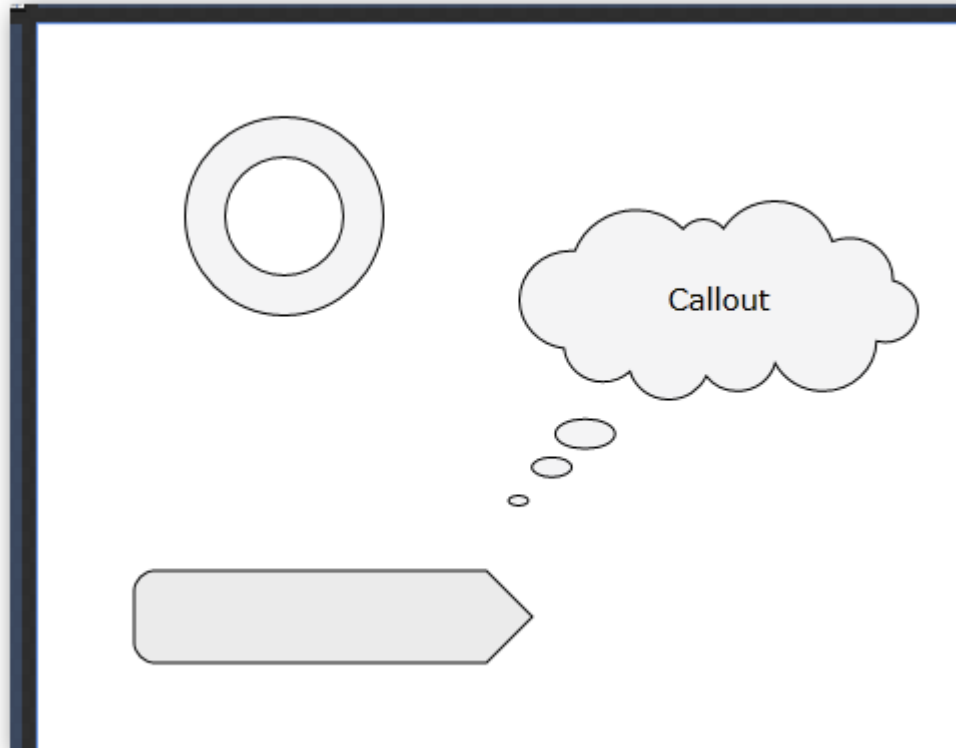
Action の作り方

- TriggerActionを継承する
 - よく使うのはTargetedTriggerAction<T>
- Invokeメソッドをオーバーライドして、Triggerが実行された時の処理を記述する
- アクションが要素に関連付けられたときの処理を記述する場合
 - OnAttachedメソッドをオーバーライドする
 - 関連付けられた要素に対しての処理
 - イベントの紐付など
 - OnDetachedメソッドをオーバーライドする
 - OnAttachedで処理した内容の解放
 - イベント購読の解除等

CUSTOM SHAPE

CUSTOM SHAPE

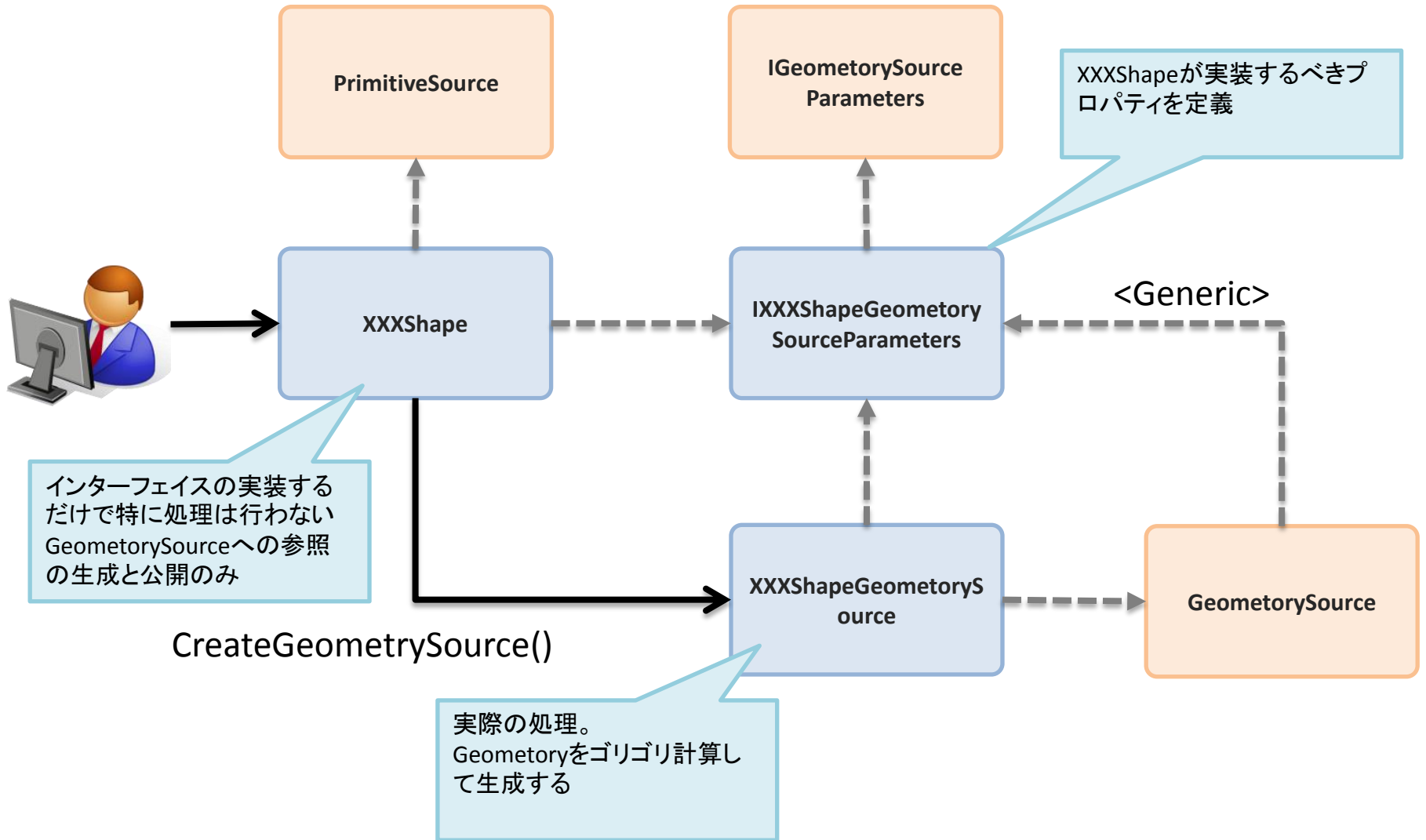
- Expressoin Blend SDKに含まれる
 - 作り方やサンプルはいまだネット上に見当たらず・・・。



CUSTOM SHAPE

- つくりどころ/なにがうれしいのか
 - サイズを変えても意図したとおりの形状を維持できる
 - 個別でPathで書くと、サイズ変更できない
 - パンくずリストのように図形のなかに画面名が含まれていて、画面名はそれぞれに長さが違う場合など

CUSTOM SHAPE



作り方

1. IXXXShapeGeometorySourceParametersインターフェイスを作成する
 - IGeometrySourceParameters を継承する
 - 利用者から利用できるプロパティを定義する
2. XXXShapeGeometorySourceクラスを作成する
 - GeometrySource を継承し、Genericには上記で作成した IXXXShapeGeometorySourceParametersを指定する
 - UpdateCachedGeometryメソッドをオーバーライドしてGeometoryをゴリゴリ記述する
3. XXXShapeクラスを作成する
 - PrimitiveShapeを継承する
 - IXXXShapeGeometorySourceParametersを継承する
 - IXXXShapeGeometorySourceParametersで定義されているプロパティを依存関係プロパティとして実装する
 - PrimitiveShapeのCreateGeometorySourceメソッドをオーバーライドしてXXXShapeGeometorySourceのインスタンスを返す

CUSTOM MARKUP EXTENSION

CUSTOM MARKUP EXTENSION

```
tionChanged">  
  setObject="{Binding ElementName=EmployeeContent}"  
  propertyName="DataContext"  
  value="{Binding SelectedValue, ElementName=EmployeesList}" />
```

```
source="{StaticResource EmployeesViewSource}"  
DataContext, ElementName=Self}" />
```

```
property="localData:BindingHelper.UpdateSourceTrigger"  
value="{localMarkup:UpdateSourceTrigger TargetProperty=Text, TargetEvent=TextChanged}" /  
... localData:BindingHelper, BasedOn="{StaticResource InputStyle}" />
```

CUSTOM MARKUP EXTENSION

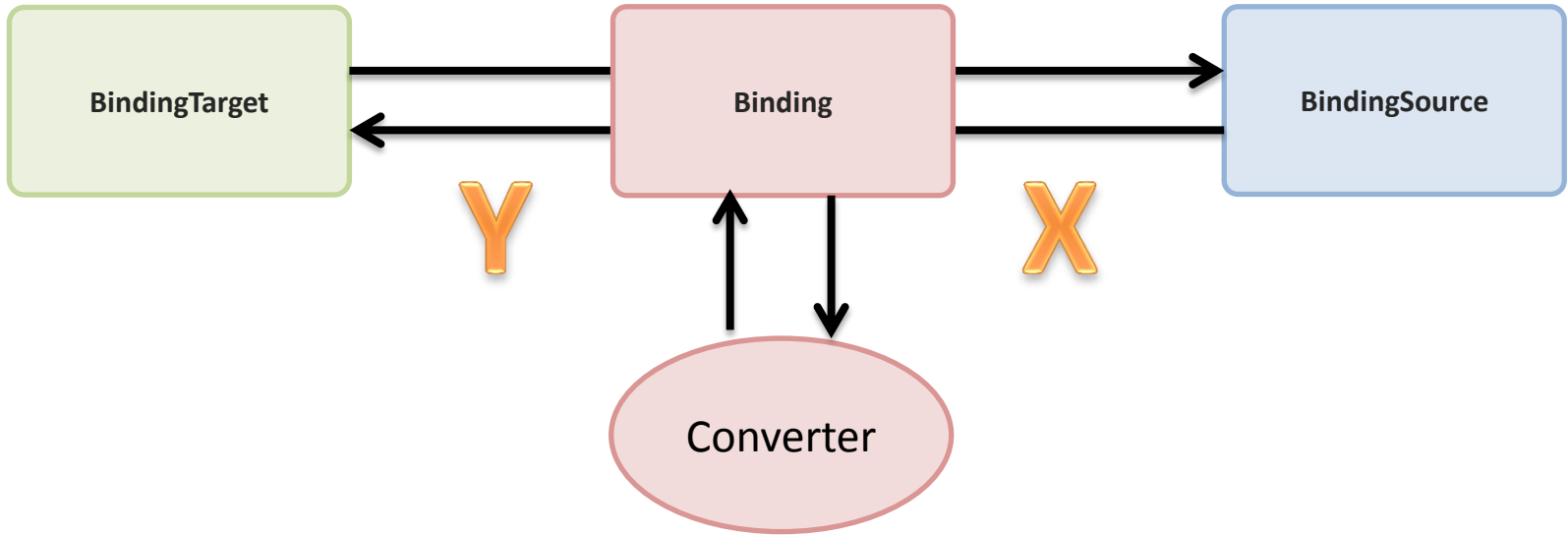
- つくりどころ/なにがうれしいのか
 - 断定的に値を計算できて、XAML上で属性に値を設定したい場合
 - 繰り返し多く定義するときにXMLをチマチマ書いてられない場合
 - これは邪道

作り方

1. `IMarkupExtension<T>`または`MarkupExtension`を継承したクラスを作る
 - クラス名のサフィックスは“Extension”にするお約束
 - XAML記述上は省略可能
2. `ProvideValue`メソッドをオーバーライドし、返したい値(属性に設定したい値)を生成し、返す
 - `IMarkupExtension<T>`を継承している場合は、戻り値の型はTになる
 - `ProvideValue`の引数の `IServiceProvider` の `GetService`からは以下を取得可
 - `IRootObjectProvider` (定義されているXAML上のルート要素を取得する)
 - `IXAMLTypeResolver`(XAML上で指定する要素名からTypeを取得する)
 - `IProvideValueTarget`(`MarkupExtension`が設定されているインスタンス、およびプロパティ名を取得する)

CUSTOM VALUE CONVERTER

CUSTOM VALUE CONVERTER



CUSTOM VALUE CONVERTER

- つくりどころ/なにがうれしいのか
 - データの表現方法が違うプロパティにバインドしたい場合
 - Boolean から Visivility
 - String から BitmapImage

作り方

1. `ICustomValueConverter` インターフェイスを継承する
2. `Convert` メソッドを実装する
 - `BindingSource` から `BindingTarget` に渡される時の変換
3. `ConvertBack` メソッドを実装する
 - `BindingTarget` から `BindingSource` に戻される時の変換
 - `Convert` のみの一方通行の場合、例外の `NotSupportedException` を発行する

全体的な注意点

- 機能を絞り込む
 - 多機能は作りが複雑になり、不具合が埋まりやすい
 - 多機能は使う側のパラメーター指定も多くなりがちで、使いにくくなりがち
- チェック処理はしつこいくらいでOK
 - Publicメソッドには必ず引数チェック
- 動ける状態と動けない状態を明確に