Sitecore CMS 7 以降

開発者のための、ページ エディターの推奨プラクティス

ページ エディターの構築およびエディター エクスペリエンスの向上のためのガイド

目次

Chapter 1

イントロダクション

このガイドは、ページ エディターの構築と、編集者のユーザー エクスペリエンスの向上を実現するために役立ててください。このガイドは主にバックエンド デベロッパーを対象とします。環境設定に携わる関係者全員が、プロジェクト ライフ サイクルの早い段階でページ エディターの構築について検討することが重要です。

開発者は編集および設計の機能をサポートするために、ソリューションを設計する必要があります (「編集のサポート」および「デザインのサポート」 を参照してください)。さらに、開発者とプロジェクト マネージャーは、ページ エディターの構成および拡張における予算を立て、ユーザー エクスペリエンスを向上する必要があります。最後に、テストにおいてソリューションのユーザーによるページ エディターのユーザビリティ テストを行うことが理想的です。

このガイドには次の章があります。

- Chapter 1 – イントロダクション
  この章では、コンテンツ、目的、このマニュアルにおける対象者について説明します
- Chapter 2 – 編集のサポート
  インライン編集のサポート手順について説明します
- Chapter 3 – デザインのサポート
  ページ エディターのデザイン変更のサポート手順について説明します
- Chapter 4 – ページ エディターの設定
  基本設定でページ エディターのユーザー エクスペリエンスを向上する方法の概要について説明します
- Chapter 5 – ページ エディターの拡張
  ユーザー エクスペリエンスを向上するための、ページ エディターのカスタマイズおよび拡張方法の概要について説明します

- Chapter 6 – ページ エディターの検出
  JavaScript および C# を使用したページ エディター モードの検出と、ページ モードに基づいたサイトの挙動の修正方法について説明します

- Chapter 7 – ヒントと秘訣
  様々なヒントおよび秘訣と、Sitecore モジュールにおけるページ エディターの役割についての概要を説明します

Chapter 2

編集のサポート

この章では、ページ エディターのコンテンツ編集機能を使用した、コンテンツのインライン編集をサポートする、 基本的な必須事項について説明します。

この章には次のセクションがあります。

- 常に FieldRenderer パイプラインを使用する

2.1 常に FieldRenderer パイプラインを使用する

FieldRenderer パイプラインからフィールドをレンダリングする場合、フィールドは自動的にページ エディターで編集可能になります。すべての Sitecore コントロールとヘルパーは、最終的にこのパイプラインを通過します。

- XSLT レンダリングおよびサブレイアウト用の Sitecore コントロール

```
<sc:FieldRenderer FieldName="Text" runat="server" />
```

- Sitecore は、日付、テキスト、リンク、画像のフィールドに専用のコントロールを提供します。

```
<sc:Text Field="Text" runat="server" />
```

- XSLT レンダリングの sc:field

```
<xsl:value-of select="sc:field('Text')" />
```

- Sitecore MVC ヘルパー

```
@Html.Sitecore().Field("Text")
```

- コード ビハインドでの FieldRenderer.Render() メソッドの使用

```
FieldRenderer.Render("Text")
```

Sitecore MVC のヒント

FieldRenderer.Render() メソッドは、文字列を返します。コンテンツ編集モードでは、この文字列は追加の HTML を含んでインライン編集をサポートします。FieldRenderer.Render() によって設定されている MVC モデルにプロパティがある場合、HtmlString を返すことを確認してください。HtmlString を返さければ、ページ エディターの HTML は回避され、プロパティを表示で使用する場合にプレーン テキストとして表示されます。これによってバリュー インラインを編集することができなくなります。

以下の方法により、フィールドの RAW 値をブラウザにレンダリングする場合、その値はページ エディターで編集できません。

- Sitecore.Context.Item.Fields["Text"].Value
- Sitecore.Context.Item["Text"]
- sc:fld("Text")

2.1.1 編集の無効化

時として、コンテンツ編集者による特定のフィールドのインライン編集を避けたい場合があります。ページ タイトルやメタ情報が良い例です。これらはどちらもページにレンダリングされないので、アクセスすることはできません。このような場合、フィールドの RAW 値を表示するのではなく、Sitecore コントロールまたはヘルパーの特定のインスタンスのインライン編集を無効化にすることができます。

- Sitecore コントロールを使用する場合:

```
<sc:Text Field="Text" DisableWebEditing="true" runat="server" />
```

- Sitecore MVC ヘルパーを使用する場合:

```
@Html.Sitecore().Field("Text", new { disable-web-editing=true })
```

- `FieldRenderer.Render()` メソッドを直接使用する場合:

```
FieldRenderer.Render("Text", "disable-web-editing=true")
```

General Link、Rich Text、Image などのフィールド タイプの RAW 値には、カスタム XML が含まれます。これらを直接ブラウザに表示することはできません。

Chapter 3

デザインのサポート

この章では、ページ エディターのデザイン変更を使用し、ページをコンポーネントのライブラリから生成する機能をサポートするための、基本的な必須事項について説明します。

この章には次のセクションがあります。

- ページ エディターでデザインをサポートする理由
- ページの分解
- 命名規則
- プレースホルダーの設定アイテムの作成
- データ ソースの使用
- ページ エディターのセキュリティ
- レスポンシブ デザイン

3.1 ページ エディターでデザインをサポートする理由

Sitecore ページは様々なコンポーネントにより生成されます。これらのコンポーネントの中には、現在表示されているアイテム (コンテキスト アイテム) からフィールドの値を表示するものもありますが、多くの場合フィールドの値はコンテンツ ツリー上の他のアイテムから出力されます。たとえば、ホームページのバナーは、世界各地で異なる *Banner* アイテムとして表される場合があります。バナーはホームページ上に表示されていますが、実際のコンテンツは *Home* アイテムに属していません。

コンテンツ エディターを使用する場合、モジュール化されたコンテンツおよびレイアウト詳細を編集する際の難易度があがります。

- ページを表すアイテム (たとえば *Home* アイテムなど) には、ホームページで訪問者に表示するすべてコンテンツは含まれません。
- コンテンツ編集者は、コンテンツが取得される場所を検索してコンテンツ ツリーに配置するために、アイテムのレイアウト詳細を確認する必要があります。
- コンテンツ編集者がアイテムの外観を変更する場合、直接アイテムのレイアウト詳細を編集する必要があります。
- アイテムのレイアウト詳細には、コンポーネントの配列方法に関する指示は含まれません。

対象的に、ページ エディターは非常に視覚的なツールです。

- ページ エディターを使用してインライン編集することによって、コンテンツ編集者は訪問者から見たページを表すインスタント プレビューを確認することができます。
- ページ エディターを使用することによって、コンポーネントのパーソナライゼーションおよび多変量テストを設定することが容易になります。

ページ エディターをサポートするように構築することは、Sitecore のマーケティングおよびアナリティクス機能へのサポートも構築することになります。これは、これらが同じコンポーネント ベースの仕組みに基づくためです。

重要
ページ エディターへのサポートは、ソリューションの設計と構築方法に影響するため、プロジェクトの開始時に考慮する必要がります。

3.2 ページの分解

ページのデザインは、個々にページ の種類を設計するのではなく、小さく再使用可能なコンポーネントに分解します。そうすることによって、コンテンツ編集者は小さいパーツから様々なページを作成することができます。

メモ
デフォルトのコンポーネント一式をデータ テンプレートのスタンダード バリューで指定する方法がお勧めです。後になってスタンダード バリューに変更を行うことで、そのデータ テンプレートに基づくすべてのアイテムに変更を反映できます。またページをデフォルトの外観にリセットする場合、コンテンツ編集者はスタンダード バリューにリセットすることができます。

次の例では、シンプルなページ デザインをコンポーネントに分解する方法を説明します。

メモ
デザインの整合性を保ち、コンテンツ編集者のページ エディターの操作をシンプルにするために、ページ エディター上の自由度を制限したい場合があります。これは、ページ コンポーネントの数を少なくするか、またはページ エディター内で制限を設定することによって実現することができます。

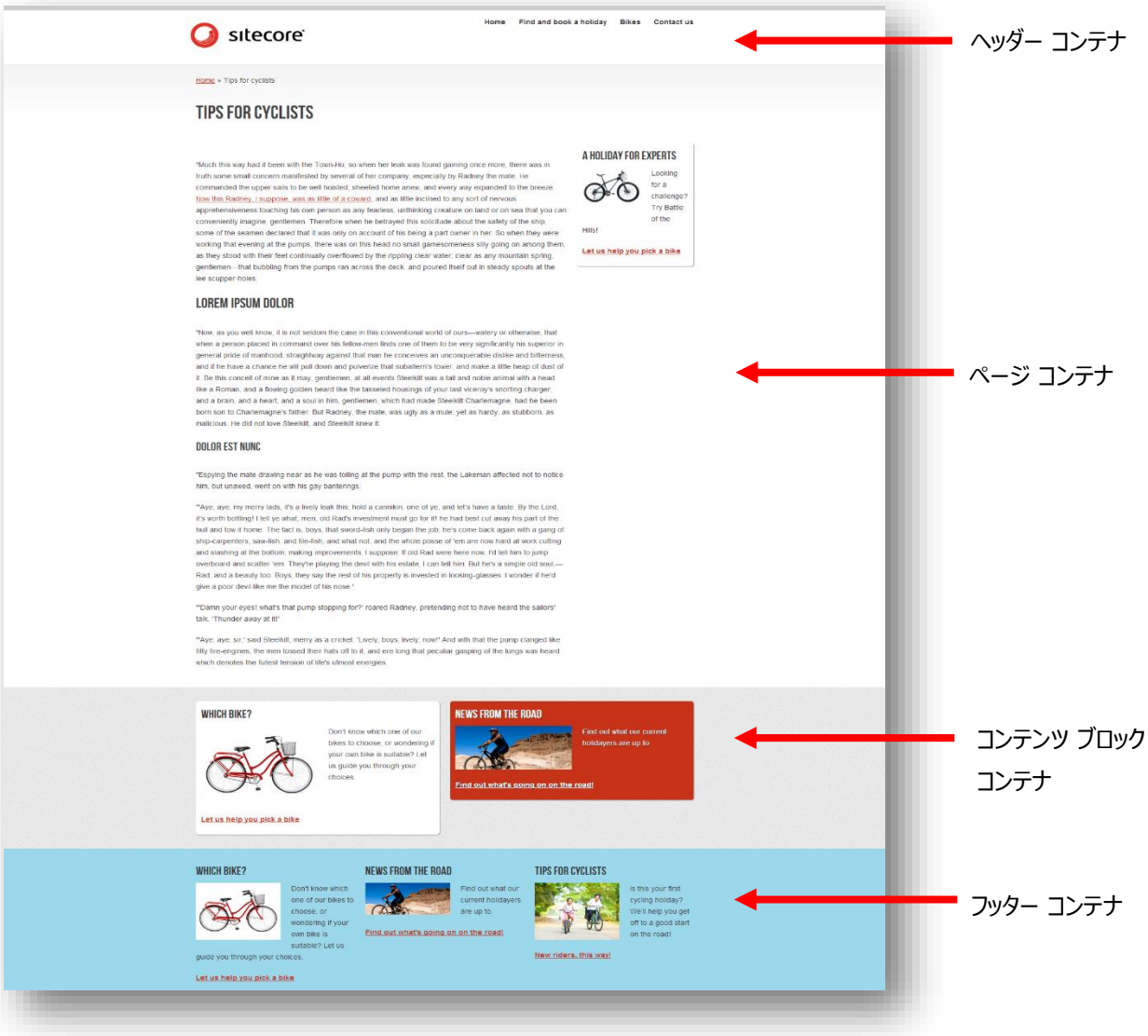
ページをコンポーネントに分解することによって、特定の訪問者の属性に基づくコンテンツまたはデザインを変更し、ページを部位別にパーソナライズまたはテストすることが可能になります。

3.2.1 水平方向のコンテナ

サンプル デザインは、ヘッダー、フッター、ページ、コンテンツ ブロックのコンテナの、主に 4 つの水平方向のコンテナによって構成されます。これらのコンテナには、スタイルが指定された`<div>` 要素と、ネストされたプレースホルダーのみが含まれます。追加のコンポーネントは、これらのプレースホルダー内でネスト化されます。ヘッダーとフッターは、すべての単一ページに表示される傾向があります。したがって、静的なバインドを使用し、ヘッダーとフッターをメイン サイトのレイアウトに含むことを検討してください。

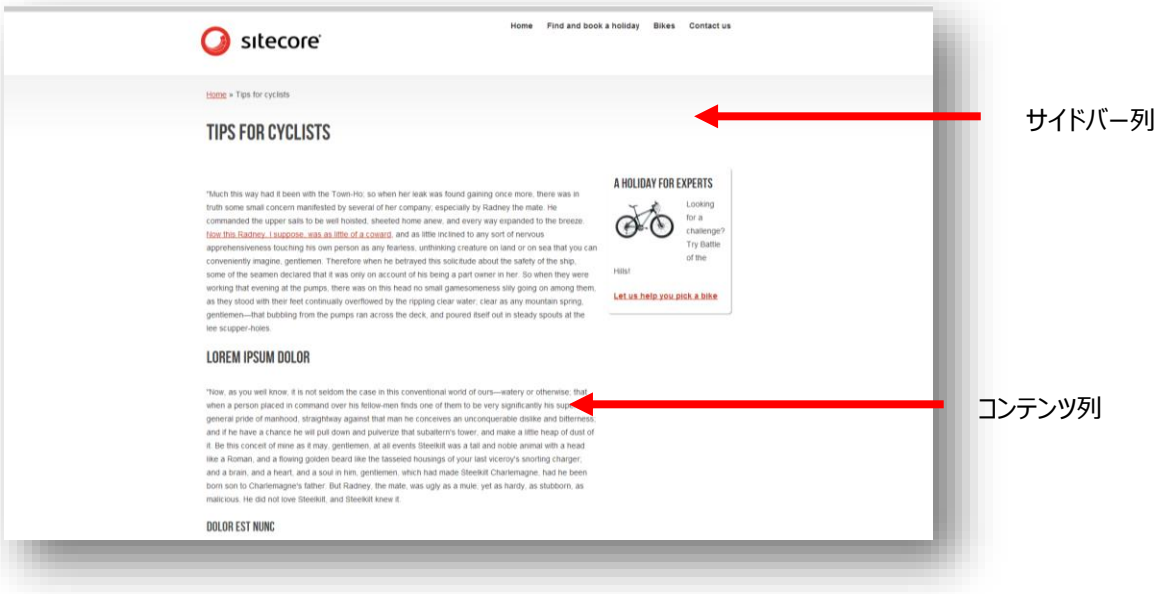
重要

静的にバインドされたコンポーネントは、ページ エディターでパーソナライズまたはテストすることはできません。



3.2.2 列

コンテナ内で、ページを縦向きの列に細分することができます。次の例では、ページ コンテナ内で 2 列のグリッドがネストされています。これは、将来的にデザインの選択肢として、1 列または 3 列構成が導入され、ページ コンテナと 2 列のグリッドが別々のコンポーネントになることが考えられるためです。



3.2.3 機能ブロック

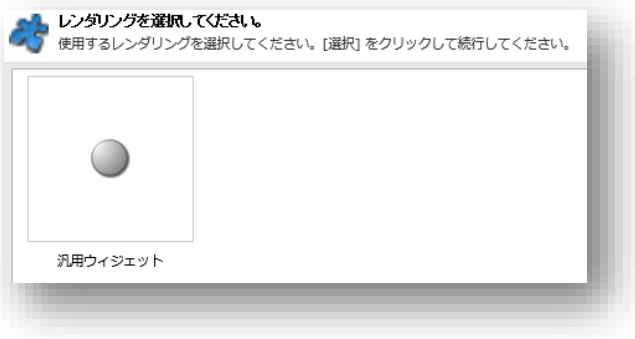
コンテナと縦向きのグリッドには、Sitecore コンテンツを生成する機能ブロックが含まれます。次の例では 4 つの異なる種類を使用します。

- メイン ナビゲーション – これは静的にヘッダー コンテナにバインドすることができます。
- ブレッドクラム – これは、すべてのページで表示する場合は、静的にページ コンテナにバインドすることができます。
- ページ コンテンツ – 今表示しているアイテムのコンテンツを直接左側の列に表示することができます。またはフォームまたはギャラリーのような、別のアイテムのコンテンツを表示するコンポーネントを作成することによって、2 列グリッド上でコンテンツを再利用することができます。
- 汎用コンテンツ ブロック – タイトル、テキスト、画像、リンクによって構成されるコンテンツの小さなかたまりです。それぞれ異なるデザインを持ち、別のロケーションに表示されますが、機能的にはまったく同じです。

3.3 命名規則

3.3.1 命名規則

コンテンツ編集者が新しいコンポーネントを挿入する場合、コンポーネントの特定方法としてコンポーネントの定義アイテムの名前または表示名を使用します。



したがって、コンポーネントには簡単に識別可能で読みやすい名前または表示名を付ける必要があります。

メモ
Sitecore では、コンポーネントの定義アイテムに選択したアイコンの拡大版がデフォルトで表示されます。推奨プラクティスとしては、コンテンツ編集者に挿入中のコンポーネントのプレビューを提供する、コンポーネントの縮小版を選択してください (「コンポーネントのプレビュー」 を参照してください。)

3.3.2 データ テンプレートとフィールドの名前付け

ページを表すデータ テンプレートには、そのページのデフォルトの外観にちなんだ名前を付けることはできません。たとえば、常にデータ テンプレートが 2 列または 3 列構成のレイアウトで使用されると仮定することはできないので、データテンプレートに *Two Column Page* または *Three Column Page* という名前を付けることは不適切です。

フィールドには、データが表示される場所ではなく、所有するデータにちなんだ名前を付ける必要があります。たとえば、*Sidebar* という名前のフィールドを作成した場合、この名前には問題点が 2 つあります。

- このフィールドに含まれるデータが常にサイドバーに表示される保証はありません。
- Sidebar という名前には、このフィールドに含まれるデータの種類について関連性がありません。

サイドバーにどのような情報を表示するのかを考慮してください。たとえば、旅行の料金についての情報は旅行アイテムに属します。この場合は、*Price Information* という名前のフィールドを作成する必要があります。情報が旅行に関連するけれど一部ではない場合は (たとえば旅行の販売促進など) 、データ ソースを使用してその情報をコンテンツ ツリーから取得する必要があります。

3.4 プレースホルダーの設定アイテムの作成

ページは、コンポーネントをプレースホルダーにバインドすることによって生成されます。プレースホルダーは、一意な Key プロパティを指定する Sitecore コントロールです。Web フォームでは、コントロールによって表されます。

```
        <sc:Placeholder Key="main" runat="server" />
```

Sitecore MVC では、プレースホルダーは Sitecore HTML ヘルパーを使用して定義されます。

```
        @Html.Sitecore().Placeholder("main")
```

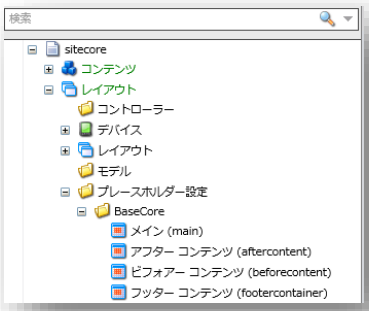
ページ エディターでプレースホルダーを表示するために、プレースホルダーの設定アイテムを作成する必要があります。プレースホルダーの設定アイテムが作成されているかどうかに関わらずプレースホルダーを表示する場合は、`web.config` で次の設定を `true` に変更します。

```
        <setting name="WebEdit.PlaceholdersEditableWithoutSettings" value="false" />
```

メモ

プレースホルダーの設定アイテムを作成しない場合、レイアウト詳細ダイアログを使用して、引き続きコンポーネントをプレースホルダーにバインドすることが可能です。

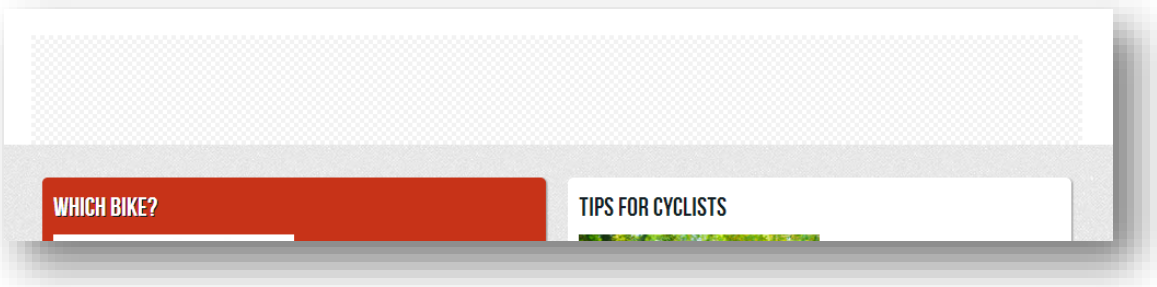
プレースホルダーの設定アイテムは、`/sitecore`/レイアウト/プレースホルダー設定　の配下にあります。



メモ

プレースホルダー キーとは、プレースホルダー コントロールの Key 属性を意味します。このフィールドでは大文字・小文字による区別はされず、さらにプレースホルダーの設定アイテムの名前およびその名前が表すキーが同じ大文字・小文字を使用しなくても問題ではありません。

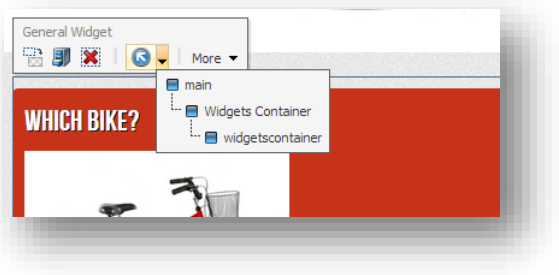
一度プレースホルダーの設定アイテムが作成されると、プレースホルダーはページ エディターでチェックの背景を持つボックスとして表示されます。



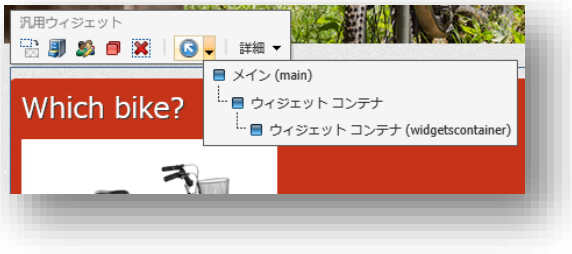
既にプレースホルダーにコンポーネントが含まれる場合、リボンの [コンポーネント] ボタンをクリックすると、[ここに追加する] コントロールが表示されます。

コンポーネントはプレースホルダー間で移動させることができます。

グッド プラクティスは、プレースホルダーの設定アイテムに、それらが表すキーと同じ名前を付けることです。小文字を使用し、スペースを削除することによって、コンテンツ編集者はプレゼンテーションのネスト構造においてプレースホルダーとコンポーネントの区別が簡単にできるようになります。次の例では、**main** と **widgetscontainer** はプレースホルダーです。



プレースホルダーの設定アイテムに表示名を定義する場合は、日本語の表示名の後ろにプレースホルダーのキーの値を含めておくとプレースホルダーとコンポーネントの区別だけでなく、プレースホルダーのキーの値を簡単に確認することが可能です。



メモ

デフォルトでは、プレースホルダーは対応するプレースホルダーの設定アイテムがない場合、ページ エディターに表示されません。ただし、`web.config` で次の設定を `true` に変更することによって、挙動をオーバーライドすることができます。

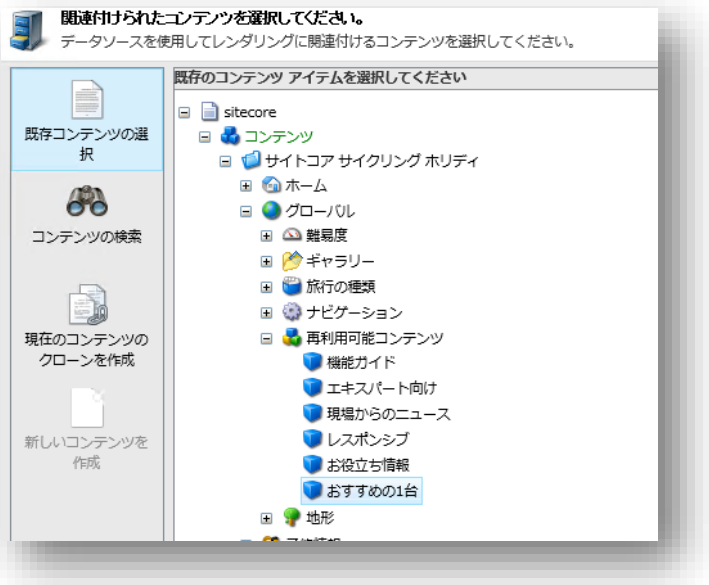
```
<setting name="WebEdit.PlaceholdersEditableWithoutSettings" value="false" />
```

3.5 データ ソースの使用

ページに表示されているすべてのデータが独占的にそのページに属しているわけではありません。コンポーネントなどのコンテンツは、複数のページにわたって再利用することができます。次の例では、Tips for cyclists というタイトルと、メイン コンテンツのみがコンテキスト アイテムに属しています。汎用コンテンツ ブロックはそれぞれ、コンテンツ ツリー上の別のアイテムから取得されたデータを表示しています。これは、コンポーネントごとに別のデータ ソースを指定することによって行うことができます。

コンテンツ編集者は、ページ エディターでコンポーネントを選択し、 [**関連するコンテンツを設定します**] ボタンをクリックすることによって、データ ソースを変更することができます。

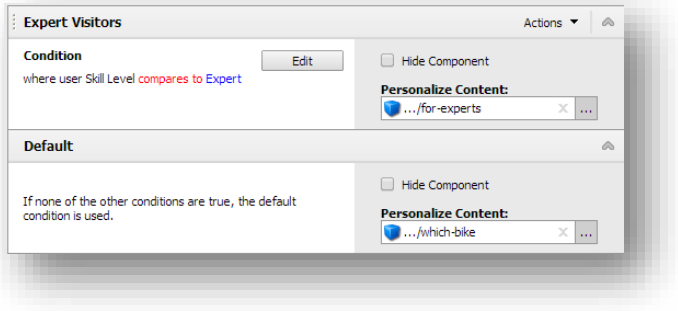
アイテムを選択し、データ ソースとして取り扱うことができます。



重要

コンテンツ ツリーのアイテムはデータ ソースとして取り扱うことができます。これは、ページを表すアイテムや、再利用可能なコンテンツの小さなかたまりのみを表すアイテムになります。上記の例では、選択されたデータ ソースはメイン サイトのツリーの外部にあるアイテムで、それ自体のレイアウト詳細はありません。

コンポーネントのパーソナライゼーションおよびテストは、コンポーネントのデータ ソースを変更する機能にある程度依存しています。次の例では、特定の条件を満たす場合に、コンポーネントがデータソースを [**which-bike**] から [**for-experts**] に変更するように設定されます。



同様に、テストを行うことによって、様々なユーザーに異なるデータ ソースを表示し、最も効果的な訪問をもたらすコンテンツを測定します。

3.5.1 データソースのコーディング

デフォルトでは、コンテンツを出力するコントロールおよびヘルパーは、コンテキスト アイテムをターゲットします。次の各例は、親のレンダリングで指定されたアイテムではなく、`Sitecore.Context.Item` の[テキスト]フィールドの値を出力します。

サブレイアウトと XSLT

```
        <sc:Text Field="Text" runat="server" />
```

MVC レンダリング

```
        @Html.Sitecore().Field("Text")
```

両方の場合において、リクエストは最終的に `FieldRenderer` パイプラインに渡されます。

```
        FieldRenderer.Render("Text")
```

データ ソースに指定されたアイテムを対象にするために、そのアイテムをコントロールまたはヘルパーから `FieldRenderer` パイプラインに渡す必要があります。

3.5.2 サブレイアウトでのデータソースの取得

サブレイアウトのコード ビハインドで、`this.Parent` を `Sublayout` としてキャストすることによって、`Sublayout` クラスからデータ ソース アイテムを取得します。

```
        string datasource = String.Empty;

        if (this.Parent is Sublayout)
        {
```

```
            datasource = ((Sublayout)Parent).DataSource;
        }
```

`Sublayout` オブジェクトは、アイテム `GUID` の文字列です。データ ソースが設定されていない場合は、このプロパティは空の接続文字列を返します。この文字列を使用し、コントロールの `DataSource` プロパティを設定します。

```
        <sc:Text Field="Text" ID="TextField" runat="server" />
```

他の ASP.NET コントロールで行うように、コントロールをその ID によってターゲットします。

```
        string datasource = String.Empty;

        if (this.Parent is Sublayout)
        {
            datasource = ((Sublayout)Parent).DataSource;

            if (!String.IsNullOrEmpty(datasource))
            {
               TextField.DataSource = datasource;
            }
        }
```

または、Sitecore ID に変換し、実際の Sitecore `Item` を取得し、コントロールの `Item` プロパティを設定します。

```
        string datasource = String.Empty;

        if (this.Parent is Sublayout)
        {
            datasource = ((Sublayout)Parent).DataSource;

            if (!String.IsNullOrEmpty(datasource))
            {
               Item item = Sitecore.Context.Database.GetItem(new ID(datasource));
               TextField.Item = item;
            }
        }
```

`Attributes` コレクションを使用し、データ ソース ID を文字列として取得することもできます。

```
        string datasourceID = Attributes["sc_datasource"];

        if (!String.IsNullOrEmpty(datasourceID))
        {
           Item datasourceItem = Sitecore.Context.Database.GetItem(new ID(datasourceID));
        }
```

ヒント

セレクションではなく、クエリに基づくアイテムのコレクションを返すことができます。追加情報は、開発チームのブログ投稿を参照してください:
http://www.sitecore.net/Community/Technical-Blogs/Sitecore-7-Development-Team/Posts/2013/04/Sitecore-7-Datasources.aspx

3.5.3 XSLT でのデータソースの取得

データ ソースが設定されている場合、`$sc_item` の変数はデータ ソースを返します。設定されていない場合は、コンテキスト アイテムを返します。

```
<sc:Text Field="Text" Select="$sc_item" />
```

3.5.4 ビュー レンダリングでのデータソースの取得

Sitecore のデフォルト `RenderingModel` を使用する場合、データ ソースはプロパティとして有効です。

```
@Model.Item
```

次も有効です。

```
@Model.Rendering.Item
```

このオブジェクトをフィールドのヘルパーに渡します。

```
@Html.Sitecore().Field("Text", Model.Item)
```

3.5.5 コントローラー レンダリングでのデータソースの取得

コントローラー レンダリング アクション内で、データ ソース アイテムにアクセスするには、`RenderingContext` を使用します。

```
RenderingContext.Current.Rendering.Item
```

メモ
直接ビューに RenderingContext を使用しないでください。ASP.NET MVC の推奨プラクティスでは、すべての表示データは、モデルまたはビュー モデル オブジェクトを通してビューに渡されることを指示します。

3.6 ページ エディターのセキュリティ

3.6.1 必須のロール

ページ エディターを使用してページをデザインするために、コンテンツ編集者は **sitecore¥Sitecore Client Designing** グループに属している必要があります。**sitecore¥Designer** グループは、このロールと **sitecore¥Sitecore Client Users** を結合します。この許可を所有するコンテンツ編集者は、 [コンテンツ編集] および [デザイン変更] モードを使用することができます。

通常の読み取り/書き取り ルールを適用します。コンテンツ エディターで行う際と同じ様に、コンテンツ編集者はページ エディターで同じコンテンツを変更することができます。**sitecore¥Sitecore Client Designing** グループに属していないコンテンツ編集者は、引き続きページ エディターを使用してコンテンツを入力することができますが、コンポーネントの追加または削除、もしくはそれらのプロパティを変更することはできません。

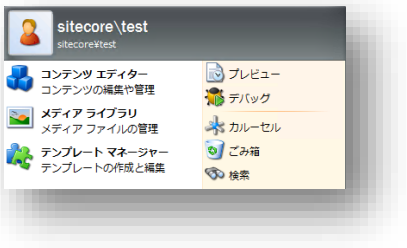
3.6.2 ページ エディターへのアクセスの無効化

コンテンツ編集者によるページ エディターへのアクセスを一切許可しない場合は、他のアイテムで行う際と同様に、Core データベースのページ エディター ボタンへのアクセスを制限します。

デフォルトでは、次のロケーションにページ エディター ボタンがあります。

- `/sitecore/content/Documents` and `settings/All users/Start menu/Right/Page Editor`
- `/sitecore/content/Applications/Content Editor/Ribbons/Chunks/Publish/Page Editor`

これらのアイテムの読取り権限が許可されていない場合、リボンまたはデスクトップ メニューに表示されません。

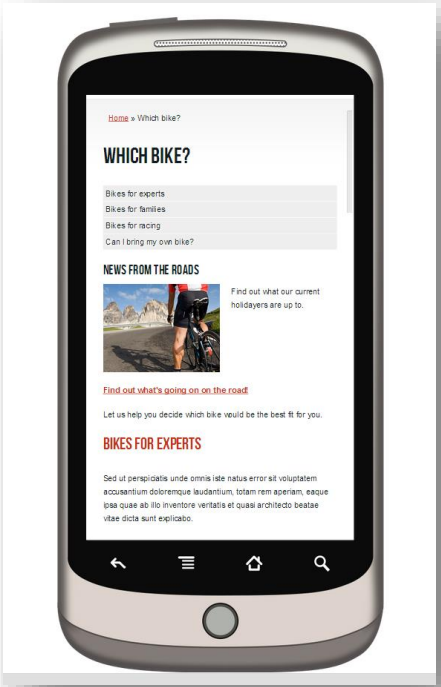


ページ エディター ボタンを `Sitecore` ログイン ページから削除するには、`<root>\Website\sitecore\login.aspx` からリンクを編集します。

3.7 レスポンシブ デザイン

サイトがレスポンシブである場合は、コンテンツ編集者がデスクトップ マシン上でページを構築するために使用するコンポーネントの組合せがどうであれ、小さいデバイスでも機能する必要があります。

- プレースホルダー制限が小さいデバイスに対応することを確認します。特定のウィジェットは、引き続き小さいスクリーン上の特定のロケーションで機能するでしょうか？
- コンテンツ編集者には、[**エクスペリエンス**] タブの [**プレビュー**] モードで**デバイス シミュレーター**を使用することを推奨します。そうすることで、小さめの画面におけるページの外観のイメージを確認することができます。



メモ

現在、ページ エディター インターフェースはモバイルまたはタッチスクリーン向けに最適化されていません。

Chapter 4

ページ エディターの設定

この章では、Sitecore コンテンツ編集者のユーザー エクスペリエンスを向上するために、ページ エディターの設定方法について説明します。

この章には次のセクションがあります。

- 有効なコントロール
- プレースホルダーのロック
- プレースホルダーの設定のオーバーライド
- 編集可能なコンポーネント
- データソースの場所とテンプレート
- 互換性のあるレンダリング
- リッチ テキスト エディター フィールド ボタン
- コンポーネントのプレビュー

4.1 有効なコントロール

プレースホルダーの設定アイテムを使用することによって、特定のプレースホルダーに追加可能なコンポーネントのリストを指定することができます。

メモ
これらの制限はページ エディターにのみ適用されます。アイテムのレイアウト詳細にアクセス可能なユーザーであれば、引き続き任意のコンポーネントを任意のプレースホルダーに追加することができます。

有効なコンポーネントのリストを特定する場合、コンテンツ編集者は、特定のプレースホルダーで機能することが保障されているコンポーネントのみを確認することができます。

4.2 プレースホルダーのロック

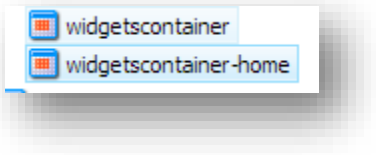
場合によって、コンテンツ編集者がコンポーネントを削除することや、コンポーネントを特定のプレースホルダーの外部に移動することを許可することなく、コンテンツ編集者によるコンポーネントのデータ ソースの変更を許可する場合があります。プレースホルダーの設定アイテムの [**編集可能**] チェックボックスをオフにすることによって、コンテンツ編集者によるコンポーネントの追加または削除を回避することができます。

[**ここに追加する**] ラベルは非表示になり、[**配置を変更する**] および [**削除**] アイコンはグレーアウトされます。

4.3 プレースホルダーの設定のオーバーライド

プレースホルダーは複数のコンテキストで使用される傾向があります。たとえば、**WidgetContainer** プレースホルダーは、ホームページとホリディ ページにも表示される場合があります。しかし、表示場所に応じて同じプレースホルダーに別の設定を指定したい場合があります。これは、そのプレースホルダーにプレースホルダー設定のオーバーライドを作成することによって実現できます。プレースホルダー設定のオーバーライドは、特定のアイテムに、またはデータ テンプレートのスタンダード バリューに次の手順を使用して適用することができます。

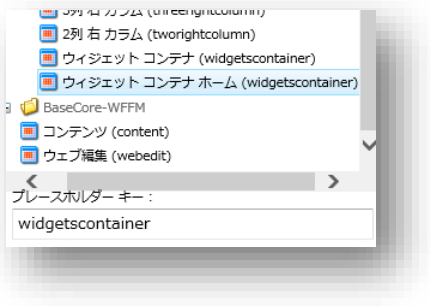
1. オーバーライドしたいプレースホルダーと類似した名前を使用し、通常のプレースホルダーの設定アイテムを作成します。



2. [**プレースホルダー キー**] フィールドが空白であることを確認し、必要な制限を設定します。

3. オーバーライドを適用したいアイテムまたはスタンダード バリューを選択します。
4. [レイアウト詳細] ダイアログを開き、[編集] をクリックします。
5. 左側のメニューで、[**プレースホルダー設定**] をクリックします。
6. [**追加**] をクリックし、新しいオーバーライドを作成します。

7. 作成したプレースホルダー設定アイテムを選択し、既存のプレースホルダー キーにマッピングします。



8. アイテムを保存します。

プレースホルダー キーが特定のアイテムのレイアウト詳細 (オーバーライドがデータ テンプレートのスタンダード バリューに設定されている場合は、アイテム の種類) に設定されると、その設定はプレースホルダーの設定アイテムをオーバーライドします。次の例では、オーバーライドを使用することによって、ホームページの **WidgetsContainer** プレースホルダー キーのインスタンスは、**汎用ウィジェット** コンポーネントと、**予約ウィジェット** コンポーネントを受け入れることができます。

4.4 編集可能なコンポーネント

プレースホルダーの設定アイテムのように、コンポーネントにも [**編集可能**] チェックボックスがあります。

このチェックボックスをオフにした場合:

- コンテンツ編集者は、たとえコンポーネントが有効なコントロールとして指定されている場合でも、このコンポーネントをプレースホルダーに追加することはできません。
- コンテンツ編集者は、ページ エディターでコンポーネントを選択することはできません。

コンポーネントによって表示されるフィールドは、編集不可として設定されない限り編集可能なままです。この設定は、たとえば予約フォームなど、特定のコンポーネントのプレースホルダーを開発者のみが制御する場合において特に有用です。

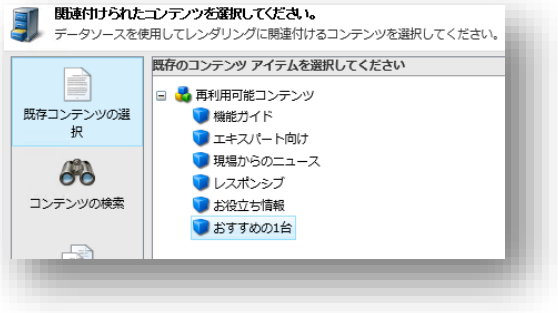
4.5 データソースの場所とテンプレート

多くのコンポーネントは機能的にデータ ソースの指定が必要です。デフォルトでは、コンテンツ編集者はコンポーネントのツールバーの [**関連するコンテンツを設定します**] ボタンをクリックし、アイテムを選択する必要があります。これにはいくつかのデメリットがあります。

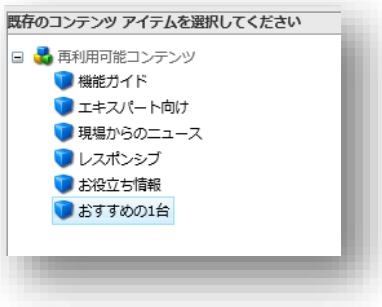
- 数回クリックしなければなりません。
- コンテンツ編集者は正しい種類のアイテムを大規模なコンテンツ ツリーから探す必要があります。
- コンテンツ編集者にデータ ソースの選択を指示するダイアログが表示されないため、設定することを忘れてしまう可能性があります。

データ ソース テンプレートおよびコンポーネントの場所を指定することには、次のメリットがあります。

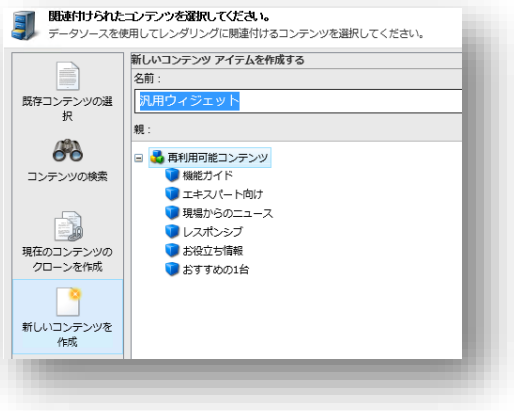
- データ ソースの場所を設定することは、コンテンツ編集者をコンテンツ ツリーの特定エリアに制限することができます。たとえ`/sitecore/Content` アイテムでも可能です。



- データ ソースを設定することによって、新しいコンポーネントを挿入する場合にコンテンツ編集者に強制的にデータ ソースを選択させます。即座に [**関連付けられたコンテンツを選択してください。**] ダイアログがポップアップされます。
- データ ソースのテンプレートを設定することによって、コンテンツ編集者がデータ ソースとして選択できるアイテムの種類を制限します。使用できないアイテムの種類はグレーアウト表示されます。



- データ ソースの場所およびデータ ソースのテンプレートの両方をコンポーネントに設定することによって、[**新しいコンテンツを作成**] ボタンが有効になります。これによって、コンテンツ編集者はコンテンツ エディターを開くことなく、正しい種類の正しい場所にある新しいデータ ソース アイテムを作成することができます。



[**データソースの場所**] および [**データソース テンプレート**] フィールドは、コンポーネントの定義アイテムに配置されます。

4.6 互換性のあるレンダリング

2 列および 3 列構成のコンテナなど、特定のコンポーネントには互換性があります。これは次のことを意味します。

- 非常に類似した機能を実装しているが、少しだけデザインが異なる
- 同じデータ ソースを受け入れる。これは必須ではありませんが、互換性のあるレンダリングはコンポーネントのデータ ソースを変更する場合、コンテンツを変更する必要がないことを示唆します。
- 同じレンダリング パラメーターを受け入れる。これは必須ではありませんが、互換性のあるレンダリングはレンダリング パラメーターの選択肢にも互換性があることを示唆します。

Sitecore では、現在選択済みのコンポーネントと互換性のあるコンポーネントのリストを構成することができます。これは、コンポーネントの定義アイテムで**互換性のあるレンダリング**のマルチリストを使用することによって行うことができます。

メモ
互換性のあるレンダリング間で切り替えを行うために、リストの各コンポーネントは、互換性のあるレンダリング フィールドに他のコンポーネントをリストする必要があります。そうしない場合は、最初に選択されたコンポーネントへ戻すことができない場合があります。

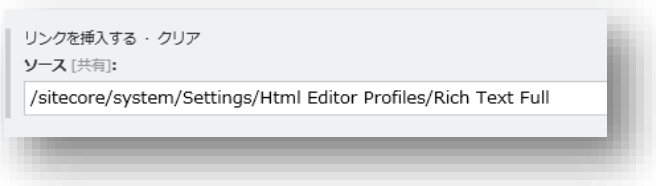
互換性のあるレンダリングを持つコンポーネントには、ツールバーに別の **レンダリングに置き換えます** と示されたボタンがあります。

重要

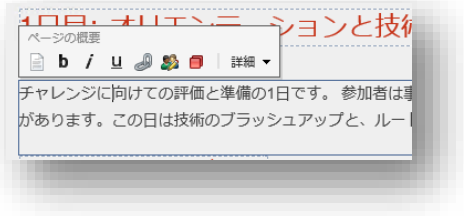
互換性のあるレンダリングとしてリストされたコンポーネントも同じプレースホルダーの[**関連付けられたコントロール**]リストに設定します。そうでなければ、選択リストには表示されません。

4.7 リッチ テキスト エディター フィールド ボタン

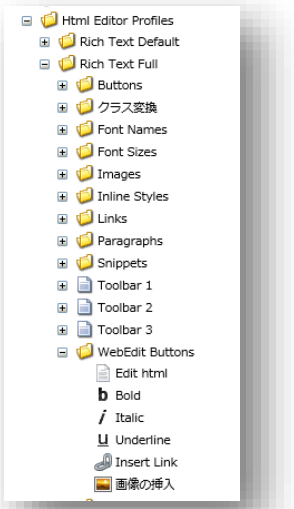
リッチ テキスト エディター フィールドは、リッチ テキスト エディター プロファイル (または RTE プロファイル) を使用してカスタマイズ可能です。RTE プロファイルは、Core データソースで設定されるボタンおよびツールバーのフォルダーです。特定のフィールドをプロファイルと関連付けるには、そのソースをプロファイルのパスに設定します。



一般的に、開発者は既存の RTE プロファイルを複製し、その名前を変更します。たとえば、Sample Rich Text Default など。そして要求どおりにボタンを追加／削除します。すべての RTE プロファイルは、 Core データベースの `/sitecore/system/Settings/Html Editor Profiles` の配下に保存されます。ページ エディターで表示する場合、リッチ テキスト エディター フィールドには、ツールバーにタイプ固有のボタンがいくつかあります。



これらのボタンもフィールドに関連付けられた RTE プロファイルによって制御され、*Webedit Buttons* というフォルダーから取得されます。



この特定のインスタンスにおけるコンテンツ編集者の操作を制限するには、[**Edit html**] ボタンを削除することを検討してください。このボタンは、ページ エディターでは有効ではないオプションの完全なセットを備えた、別のリッチ テキスト エディター ダイアログを開きます。

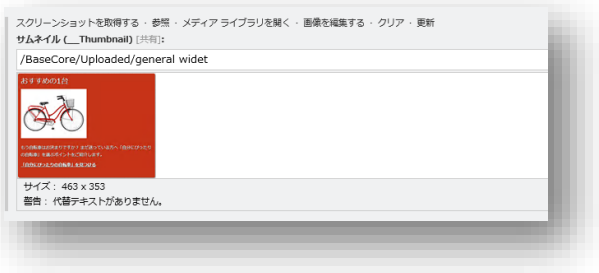
4.8 コンポーネントのプレビュー

[**レンダリングを選択してください。**] ダイアログは、新しいコンポーネントを挿入する際に表示され、デフォルトではコンポーネント アイテムのアイコンの拡大版が表示されます。

アイコンだけの情報ではコンポーネントの外観を確認することができません。コンテンツ編集者に有用なプレビューを表示するには、コンポーネントのスクリーンショットを撮り、コンポーネントの[**サムネイル**] フィールドにアップロードします。

メモ

サムネイル フィールドにアクセスするには、コンテンツ エディターで**スタンダード フィールド**を有効にする必要があります。



[**レンダリングを選択してください。**] ダイアログにはコンポーネントのプレビューが表示されます。

Chapter 5

ページ エディターの拡張

この章では、カスタム ボタンおよびツールバーを使用した、ページ エディターの拡張方法について説明します。

この章には次のセクションがあります。

- カスタム エクスペリエンス ボタン
- エディット フレーム
- リボンのカスタマイズ

5.1 カスタム エクスペリエンス ボタン

フィールドおよびコンポーネントを囲むフリー ツールバーを修正し、いくつかのカスタム エクスペリエンス ボタンを含むことができます。これらのボタンの機能を追加することによって、ページエディターの拡張と改善が行え、結果としてコンテンツ エディターおよびデスクトップへの移動を削減することができます。

- すべてのカスタム エクスペリエンス ボタンは、Core データベースの `/sitecore/content/Applications/WebEdit/Custom Experience Buttons` の配下で定義されます。
- 一度作成し設定すると、カスタム エクスペリエンス ボタンは、フィールドの定義アイテムまたは Master データベースのコンポーネントの定義アイテムに割り当てられます。
- 割り当てられたカスタム エクスペリエンス ボタンは、ページ エディターにフィールドまたはコンポーネントが表示されている場合はいつでもフリー ツールバーに表示されます。

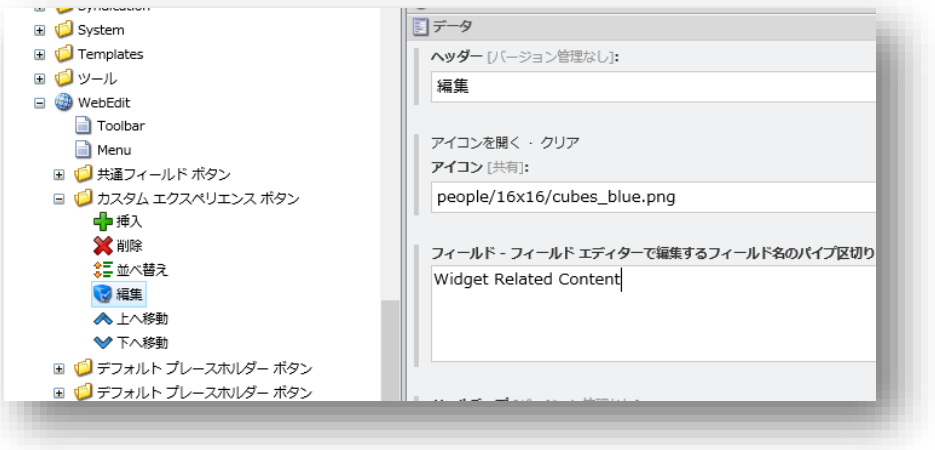
メモ
コンポーネントに割り当てられたカスタム エクスペリエンス ボタンはデザイン モードでのみ表示されます。

5.1.1 フィールド エディター ボタン

フィールド エディター ボタンは、最も一般的に使用されるカスタム エクスペリエンス ボタンの種類です。これらは追加のフィールドをダイアログに表示するために使用されます。これは次の場合に特に役立ちます。

- フィールドはページには表示されません。たとえば、メタデータまたはスタイルなど。
- フィールドの種類はインラインを編集できません。たとえば、Multilist または Treelist など。

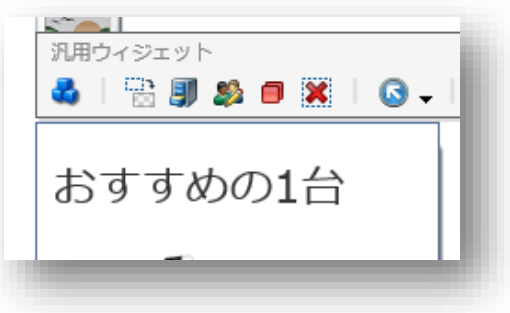
次の例では、[**編集**] フィールド エディター ボタンは、[**Widget Related Content**] フィールドをダイアログに開くように設定されています。



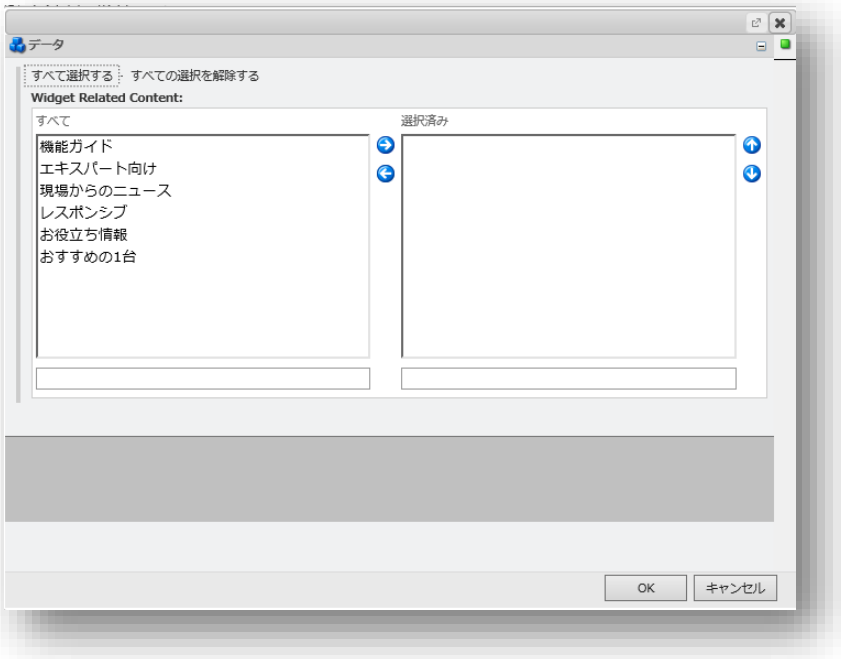
メモ

このリストには、| (パイプ) 記号で区切られた任意の数のフィールドを含むことができます。

この結果、新しいボタンが汎用ウィジェットのフリー ツールバーに表示されます。



ボタンをクリックすると、指定したフィールドを持つダイアログが開かれます。

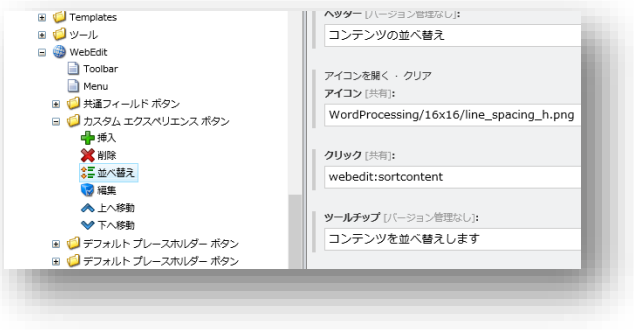


重要

フィールド エディター ボタンはコンテキストのデータ ソースに対して働きます。前の例では、**Widget Related Content** フィールドには、バックグラウンドの General Widget コンポーネントで選択されたデータ ソースに対して動作します。

5.1.2 Webedit ボタン

WebEdit Buttons ボタンはプログラミングが必要で、カスタムの挙動をトリガーするために使用できます。フィールド エディター ボタンとは違って、特異的にフィールドのセットを開き、webedit ボタンはコマンドをトリガーします。次の例では、**webedit:sortcontent** コマンドがトリガーされています。

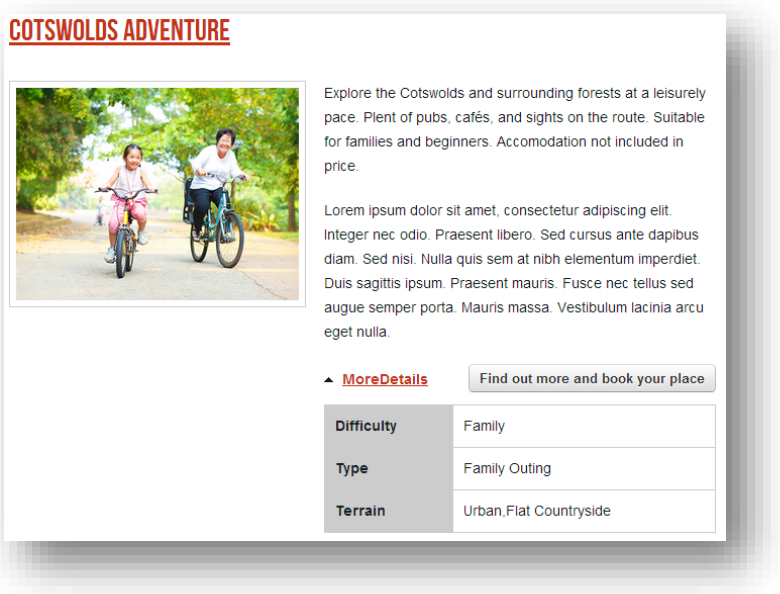


次は webedit ボタンの使用方法についての例です。

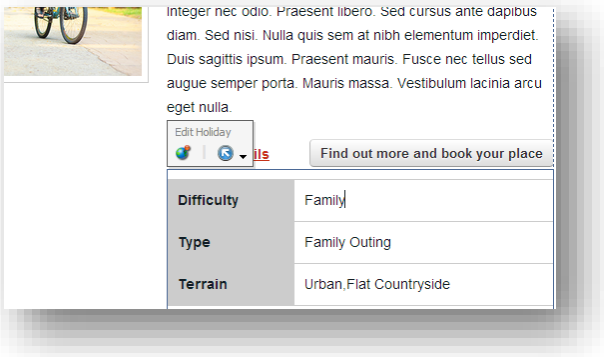
- スペルチェック ボタンをリッチ テキスト フィールドに追加します。
- コンポーネントを上下に移動します。
- コンポーネントのデータ ソースをパブリッシュします。

5.2 エディット フレーム

いくつかのインスタンスでは、カスタム エクスペリエンスを特定のコンポーネントの一部またはページに追加したい場合があります。次のコンポーネントには、いくつかのリストを出力する表が含まれます。これらはマルチリスト フィールドとして Sitecore に保存されます。このフィールドを表示する最適な方法は、フィールド エディター ボタンを使用することです (「フィールド エディター ボタン」を参照してください。)

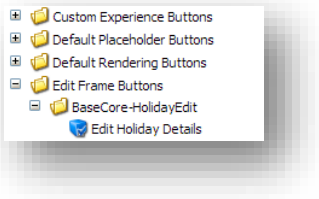


ただし、コンポーネントは大きいので、フィールド エディター ボタンは修正するコンテンツの近くに配置する必要があります。エディット フレームを使用することによって、フリーツールバーでマークアップの特定箇所を囲むことができます。このフリー ツールバーは、Sitecore コンポーネントおよびフィールドを囲むフリーツールバーと全く同じような外観と機能を持ち、コンポーネントおよびフィールドに追加できるものと全く同じようなカスタム エクスペリエンス ボタンのリストを表示するために使用できます。次の例は、'Edit Holiday' フィールド エディター ボタンを有効にするエディット フレームを使用した表の囲み方を示します。



各検索結果の周辺にエディット フレームを作成するには、`<sc:EditFrame />` コントロールを使って検索結果のマークアップを囲み、次の 2 つを指定します。

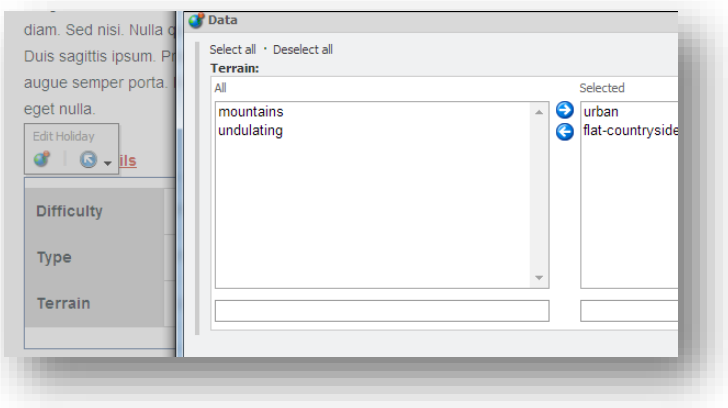
- Core データベースのカスタム エクスペリエンス ボタンのフォルダーへのパスを指定します。各エディット フレームは、ひとつ以上のカスタム エクスペリエンス ボタンを含む **Edit Frame Buttons** の下にサブフォルダーを持つ必要があります。この例では、**Edit Holiday Details** だけがあります。



- エディット フレームを使用してターゲットするデータ ソース アイテム

```
        <div class="sectionItem searchResult">
            <sc:EditFrame Buttons="/sitecore/content/Applications/WebEdit/Edit Frame
Buttons/BaseCore-HolidayEdit" DataSource="<%# Item.ItemId.ToString() %>" runat="server">
               <!-- My search result -->
            </sc:EditFrame>
        </div>
```

ボタンをクリックすると、holiday アイテムのエディター ダイアログが開きます。



メモ

エディット フレームが参照中のアイテムを修正する許可をコンテンツ編集者が持たない場合、ボタンは一切表示されません。同様に、コンテンツ編集者が特定のフィールドに制限されている場合、編集が許可されていないフィールドはグレーアウト表示になります。

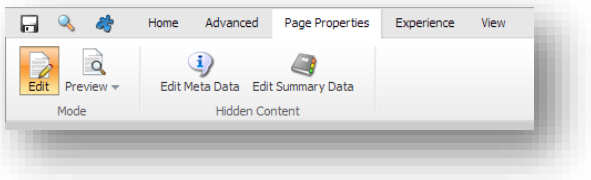
次の場合にもエディット フレームを使用することができます。

- ページ ロードの `FieldRenderer` パイプラインからではなく、AJAX またはインデックスから提供されたコンテンツを囲むので、デフォルトでは編集不可です (`FieldRenderer` パイプラインから提供されたコンテンツの編集に関するメソッドは、「動的なインライン編集またはインデックスされたコンテンツ」を参照してください。)
- コンポーネントのグループ全体を囲み、挿入されているコンポーネントに一括変更を行うことが可能なカスタムの webedit ボタンを作成できます。

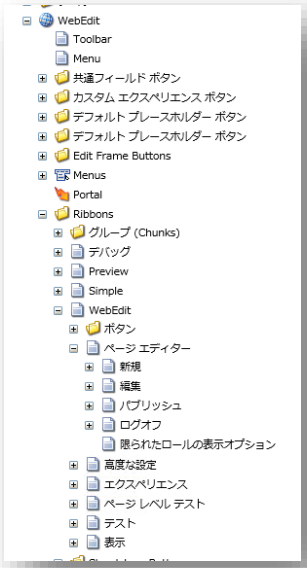
重要
現在エディット フレームは Sitecore MVC でサポートされていません。

5.3 リボンのカスタマイズ

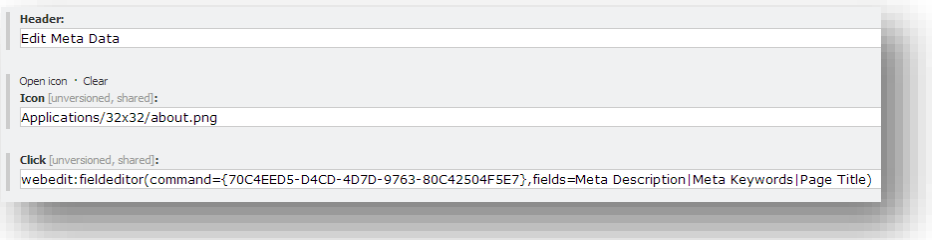
リボンに追加のボタンを挿入することができます。これは、たとえばページ タイトルやメタ記述など、表示されることがないフィールドがあり、特定のコンポーネントにも属していない場合に特に有用です。以下は、ページ エディターのカスタムのタブの例です。



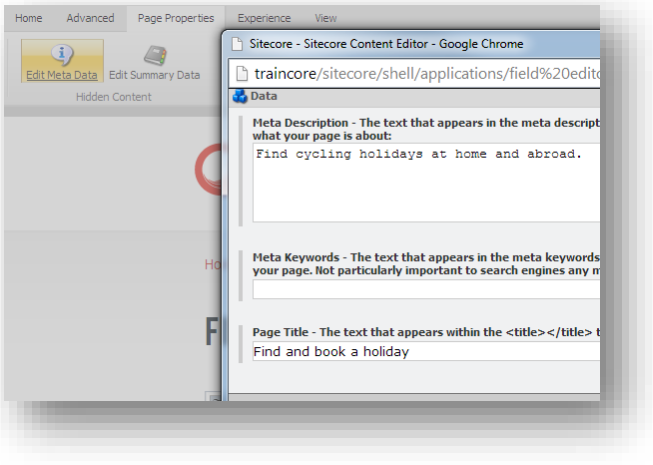
カスタム エクスペリエンス ボタンやエディット フレームのように、これらの追加ボタンは Core データベースの `/sitecore/content/Applications/WebEdit` で定義されます。



(`/sitecore/templates/System/Ribbon/Large Button` の種類の) ボタンは、カスタム エクスペリエンス ボタンと非常に似ていて、[クリック] フィールドでパラメーター化されたコマンドを取得します。



上記の例では、[**Edit Meta Data**] ボタンをクリックすると、フィールドが表示されているフィールド エディターが開きます。



これらをリボンに追加することで、コンテンツ編集者は、非表示のページ レベル フィールドを修正するためにコンテンツ エディターを開く必要がなくなります。

Chapter 6

ページ エディターの検出

この章では、C# および Javascript を使ったページ エディターの検出方法と、コンテンツ編集者のユーザー エクスペリエンスを向上するために、検出をどのように使用するのかについて説明します。

この章には次のセクションがあります。

- ページ モードの検出
- ページ モードに応じたコンポーネントの挙動の変更

6.1 ページ モードの検出

サイトの訪問者に表示されているビューは常にインライン編集に適切とは限りません。これは特にアニメーションの要素に該当します。開発者はサイトがページ エディターの編集モードで表示されているか否かを検出し、それに合わせて挙動を修正することができます。

メモ
ページ エディター モードの検出を可能にする同じクラスは、デバッグ、プレビュー、通常モードの検出にも使用できます。

6.1.1 C# を使用したページ モードの検出

`Sitecore.Context.PageMode` クラスによって、開発者はコードを使ってページ エディターをターゲットすることができます。一般的な `IsPageEditor` ブール式に加え、`IsPageEditorEditingMode` を使ってコンテンツ編集およびデザイン変更のモードをターゲットすることもできます。

```
If (Sitecore.Context.PageMode.IsPageEditor)
{
      // Page Editor-specific behavior
}
```

有効なオプションには次があります。

- `IsDebugging`
- `IsNormal`
- `IsPageEditor`
- `IsPageEditorEditing` (編集およびデザイン モードを含みます。)
- `IsPreview`
- `IsProfiling`
- `IsSimulatedDevicePreviewing`

6.1.2 Sitecore MVC でのページ モードの検出

ページ エディター モードは、Sitecore MVC でまったく同じクラスを使用して検出することができます。

```
@Sitecore.Context.PageMode.IsPageEditor
```

ただし、ASP.NET MVC 内における推奨プラクティスは、表示に必要なすべてのデータをビュー モデルから取得することです。したがって、直接 `Sitecore.Context.PageMode.IsPageEditor` を呼び出すのではなく、`IsPageEditor` プロパティをカスタム ビュー モデルに組み込むことが好ましいです。

```
public class CustomModel
      public bool IsPageEditor { get; set; }
```

```
        }
```

6.1.3 Javascript を使用したページ モードの検出

Sitecore は 、Javascript をプレビューおよびページ エディターで有効にします。プレビュー モードでは、次のオブジェクトが有効です。

```
        Sitecore
```

ページ エディター モードでは、次のオブジェクトが有効です。

```
        Sitecore.PageModes.PageEditor
```

これらのオブジェクトは、Javascript の挙動を変更する際に使用できます。

```
        if (Sitecore) {
            if (Sitecore.PageModes.PageEditor) {
                return 'pageeditor';
            }
            return 'preview';
        }

        return 'visitor';
        };
```

6.2 ページ モードに応じたコンポーネントの挙動の変更

次は、ページ エディター エクスペリエンスを向上するための、ページモードの検出に関する使用方法のリストです。これは完全なリストではありません。

メモ
デバッグおよびプレビュー モードは、訪問者から見たサイトを表示します。したがって、コンテンツ編集者向けに、具体的にサイトの挙動を変更することを推奨します (`Sitecore.Context.PageModes.IsPageEditor` を確認することによって行うことができます) 。たとえば、`Sitecore.Context.PageModes.IsNormal` が false の場合にコンポーネントを非表示または表示する場合、これはデバッグおよびプレビュー モードにも反映されます。

6.2.1 サンプル データ

コンポーネントのデータソース ロケーションとデータソース テンプレートを設定することによって、強制的にコンテンツ編集者にデータソースを選択させることができます。ただし、コンテンツ編集者は過ってデータ ソースを削除してしまう場合があります。次の例は、データ ソースが指定されていない場合にページ エディター モードに表示されるサンプル コンテンツです。このコンテンツはコンテンツ編集者にダイアログを表示し、データ ソースを選択させます。

これを行うには、データ ソースが指定されているかどうかを確認します。データ ソースが存在せず、ページ エディター モードにいる場合は、各コントロール (例えば `<sc:Text />`) の `Item` または `DataSource` プロパティを、サンプル アイテムに設定します。プレビューおよび通常のモードでは、訪問者にコンポーネントを非表示にします。

```
        string datasource = (Sublayout)this.Parent.Datasource;

        if (datasource == null)
        {
             if (Sitecore.Context.PageMode.IsPageEditor)
             {
                Item sampleItem = MyReferences.SampleWidgetItem;

                HeadingTextControl.Item = item;
                HeadingTextControl.DisableWebEditing = true;
             }
```

```
            else
            {
              this.Visible = false;
            }
        }
```

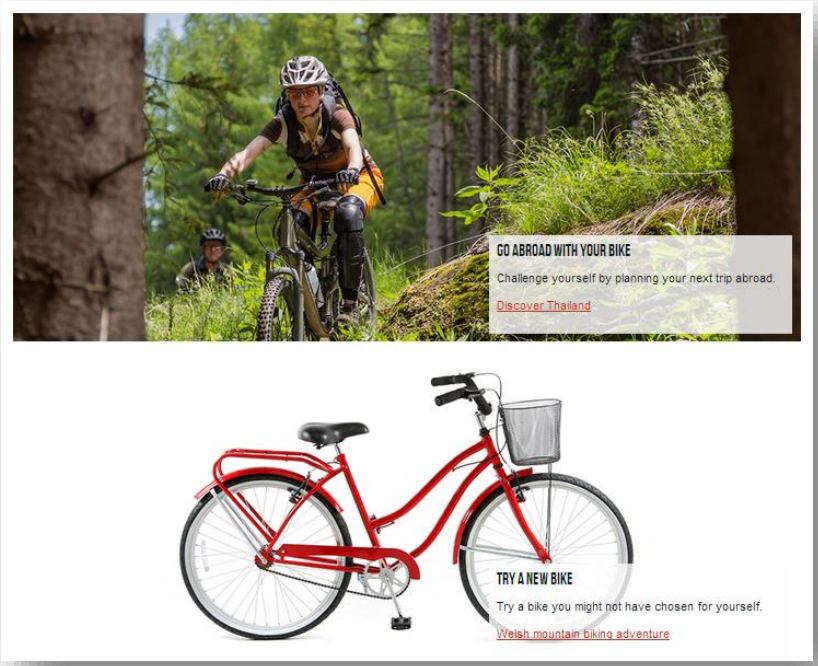
上記のコードについて、注意すべき点がいくつかあります。

- `MyReferences.SampleWidget` は、サンプル アイテムを参照します。サンプル アイテムは、実際のデータ ソースとまったく同じデータ テンプレートから作成されます。サンプル アイテムを使用する理由は、翻訳が可能で、データテンプレートでの変更が 伝達されるためです。コンテンツもまた翻訳可能です。
- コンポーネントの `Datasource` プロパティを設定するのではなく (機能しません) 、それぞれの個々のコントロールの `Item` プロパティを設定します。
- サンプルのデータ ソースが表示されている場合、各 Sitecore コントロールの `DisableWebEditing` を `ture` に設定します。このように設定しなかった場合、コンテンツ編集者は新しいデータ ソースを選択するのではなく、サンプル テキストを編集しようとする場合があります。

または、コンテンツ編集者にデータ ソースを選択するようダイアログを表示し、コンポーネント全体を置換えることができます。

6.2.2 動的なコンポーネント

スクローリング ギャラリーなどのアニメーション コンポーネントは編集が困難です。フロントエンド デベロッパーまたはバックエンド デベロッパーは、編集エクスペリエンスを向上させるために、アニメーションを停止したり、コンテンツを別の方法で表示することができます。次の例では、ギャラリーはページ エディター モードでは画像をフラットに表示します。



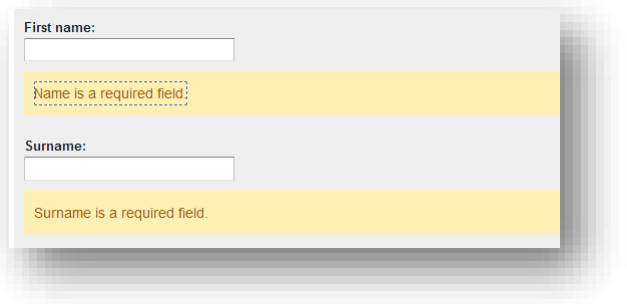
プレビューまたは通常モードで、ギャラリーは通常通り、画像をスライドさせて表示させます。

6.2.3 非表示コンテンツ

特定のコンテンツは、訪問者がページでインタラクトした場合にのみ表示されます。このようなコンテンツは、コンテンツ編集者によるコンテンツ インラインの編集を困難または不可能にし、コンテンツ エディターを使用する必要性がでてきます。このような種類のコンテンツには次があります。

- フォーム エラーと承認メッセージ
- モーダル ウィンドウ
- ホバーでのみ表示されるコンテンツ

ページ エディター モードでページが表示されている場合は、開発者はこのコンテンツをコンテキストに表示することによってヘルプできます。この例では、フォームのエラー メッセージはページがロードする際に表示されます。



非表示コンテンツを表示することによって、特に翻訳中においてコンテンツが見落とされるリスクを軽減します。

6.2.4 外部コンテンツ

コンテンツ編集者は、Twitter フィードまたはニュース ストリームなどの外部コンテンツを編集することはできません。また、外部コンテンツはページ ロード時間を増加させ、挿入されている Javascript はページ エディターと衝突する可能性があります。

したがって、コンテンツ編集者が完全に表示されたページを確認する場合は、外部コンテンツをプレビュー モードに切り替えることを指示する静的なメッセージと置換えることを検討してください。

Chapter 7

ヒントと秘訣

この章では、ページ エディター エクスペリエンスを向上させるための、様々なヒントと秘訣を紹介します。

この章には次のセクションがあります。

- 動的なインライン編集またはインデックスされたコンテンツ
- ページ エディターと Sitecore モジュール

7.1 動的なインライン編集またはインデックスされたコンテンツ

検索インデックスから直接表示されたコンテンツは (Sitecore では 初期状態で Lucence および Solr をサポートします)、ページ エディターで編集することはできません。次の例は、外部インデックスから出力された 3 つの検索結果を示します。

コンテンツ編集者によるこのコンテンツの管理を可能にする方法がいくつかあります。

- エディット フレームを使って各検索結果を囲みます。このエディット フレームは、編集可能にしたいフィールド (たとえば検索結果タイトルや説明) を表示する、フィールド エディター ボタンのフォルダーをポイントします。エディット フレームのデータ ソースを検索結果によって表される アイテム ID に設定します (Sitecore の `SearchResultItem` には、`ID` プロパティがあります)。コンテンツ編集者は、アイテムへのアクセスを持つ場合、フィールド エディター ボタンをクリックしてコンテンツをアップデートすることができます。
- ページ エディター モードの場合、検索結果のコンテンツは、インデックスからではなくデータベースから表示します。これによって、`SearchResultItem` (カスタム クラスを使用する場合は同等のもの) からの `ID` プロパティを使ってアイテムを取得し、そのフィールド値はインデックスされた値ではなく、`FieldRenderer` を使って出力する必要があります。

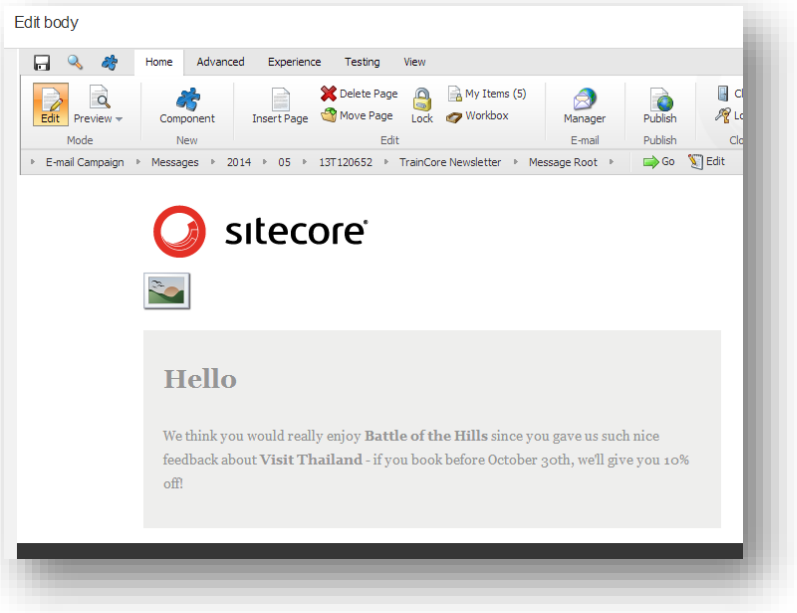
AJAX からページにもたらされたコンテンツにも同じ原則を適用します。このコンテンツが元は Sitecore から取得した場合でも (たとえばカスタムの Web API など) 、`FieldRenderer` パイプラインからはレンダーされないので、ページ エディターでは編集することはできません。ページ エディター モードでのみデータをページ ロードに表示し、編集を有効にすることを検討してください。

7.2 ページ エディターと Sitecore モジュール

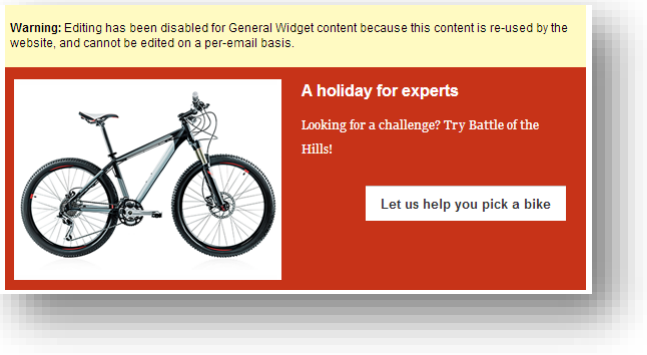
7.2.1 Email Campaign Manager (ECM)

Email Campaign Manager の電子メールは、ページ エディター インターフェースを使ってコンポーネントから構築されます。独自の ECM 電子メールを構築したい場合は、通常の Sitecore ページとまったく同じ原則を使って電子メールを構築する必要があります。

- HTML 電子メールのデザインがコンポーネント化をサポートすることを確認します
- デザインと編集の両方をサポートしていることを確認します
- コンポーネントの定義アイテムとプレースホルダーの設定アイテムを要求通りに設定します



電子メール コンポーネントは、メイン Web サイトと同じデータ ソースを使用することができます。ただし、コンテンツ編集者はこのコンテンツが共有されていることを認識していない場合があります。電子メールのコンテキストでコンテンツを編集した場合、これらの変更はメインサイトにも反映されます。これは、その変更が特定の電子メールに特有の場合に問題になります。この問題を回避するには、共有コンテンツを使用するコンポーネントの編集を無効にするか、またはコンテンツ編集者がページ エディター モードの場合に警告を表示します。



7.2.2 Web Forms for Marketers (WFFM)

Web Forms for Marketers は、モジュールをインストールする場合に、[**Form**] コンポーネントを特定のプレースホルダーに追加するようにダイアログを表示します。コンポーネントを他のプレースホルダーで有効にする場合は、プレースホルダーの設定アイテムの **関連付けられたコントロール** リストに追加します。

7.2.3 Sitecore のための JQuery Mobile Component Library

Sitecore Marketplace で有効な 、ページ エディターと互換性のある JQuery モバイル コンポーネント ライブラリがあります。
https://marketplace.sitecore.net/en/Modules/Sitecore_jQuery_Mobile_Component_Library.aspx