Experience Cloud 開発者ガイド

バージョン 60.0, Spring '24

最終更新日: 2024/01/02

本書の英語版と翻訳版で相違がある場合は英語版を優先するものとします。

目次

第 1 章　エクスペリエンスビルダーサイトを使いこなすには

トピック:

エクスペリエンスビルダーテンプレートにより、従業員、顧客、パートナーをつなげるブランド設定されたサイトを作成できます。Lightning コンポーネントフレームワークに基づいたエクスペリエンスビルダーテンプレートには、すぐに使える多くの機能や Lightning コンポーネントが用意されています。ですが、Lightning コンポーネントフレームワークの本領は、それぞれのビジネスニーズに合わせてカスタム Lightning コンポーネントや機能を開発して、サイトの外観を完全にカスタマイズできる点にあります。

Spring '19 (API バージョン 45.0) 以降、Lightning Web コンポーネントモデルと従来の Aura コンポーネントモデルの 2 つのプログラミングモデルを使用して Lightning コンポーネントを作成できます。Lightning Web コンポーネントは、HTML と最新の JavaScript を使用して作成されたカスタム HTML 要素です。

ヒント: Spring '21 (API バージョン 50.0) の Experience Cloud への名称変更により、新しい用語がいくつか取り入れられました。それぞれの内容を把握するのは大変かもしれません。概要は次のとおりです。

エクスペリエンスビルダーサイト (旧称「Lightning コミュニティ」) は、エクスペリエンスビルダーでカスタマイズするテンプレートベースのサイトです。Lightning Web Runtime (LWR) の使用開始に伴い、さらに明確にするため 2 つの新しい用語が追加されました。

- *LWR* サイトは、Build Your Own (LWR) テンプレートなど、最新の LWR ベースのテンプレートを使用して構築されます。LWR サイトは、Lightning Web コンポーネントでのみ使用できます。Aura コンポーネントでは使用できません。「LWR Sites for Experience Cloud (Experience Cloud の LWR サイト)」を参照してください。
- *Aura* サイトは、カスタマーサービス、Partner Central、カスタマー取引先ポータルなど、Aura で実行されるオリジナルのテンプレートを使用して構築されます。Aura サイトの場合、Lightning Web コンポーネントと Aura コンポーネントは 1 つのページで共存および相互運用できます。

Lightning Web コンポーネントと Aura コンポーネントの両方をエクスペリエンスビルダーのドラッグアンドドロップコンポーネントとして設定できます。システム管理者とエンドユーザーは、コンポーネントの開発にどのプログラミングモデルが使用されたかを知らず、単に Lightning コンポーネントとして認識します。

このガイドは、開発者、パートナー、ISVなど、あらゆるユーザーを対象として、カスタム Aura サイトおよびコンポーネント、テーマレイアウト、そして Lightning Bolt ソリューションを作成する方法を説明します。

ご利用になる前に

カスタムエクスペリエンスビルダーサイトの開発を始めるには、Lightning での開発に慣れていることが必要です。

エクスペリエンスビルダーサイトと Lightning コンポーネントを作成可能なエディションは、**Enterprise** Edition、**Performance** Edition、**Unlimited** Edition、**Developer** Edition、または Sandbox です。

このガイドを正しく活用するためには、以下が必要です。

- デジタルエクスペリエンスが有効になっている組織
- エクスペリエンスビルダーテンプレートまたは Lightning Bolt ソリューションに基づいた新規または既存のサイト
- エクスペリエンスビルダーの使用についての知識
- Lightning コンポーネントの開発経験と CSS の使用経験

Lightning 開発のリソース

Lightning 開発に慣れていない方は、次のリソースを参考にしてください。

Lightning Aura コンポーネント開発者ガイド
Aura のすべてを網羅したガイドです。このガイドに記載されている基礎的なコンセプトやアプローチが、本ガイドの基盤となっています。『*Experience Cloud 開発者ガイド*』は、開発シリーズのパート 2 のようなガイドですので、パート 1 の内容を習得するまでは使用する必要はありません。

Lightning Web Components 開発者ガイド
Lightning Web コンポーネント (HTML と最新の JavaScript 使用して作成されたカスタム HTML 要素) の開発方法について学びます。

LWR Sites for Experience Cloud Developer Guide (Experience Cloud の LWR サイト開発者ガイド)
Experience Cloud の Build Your Own (LWR) テンプレートを使用して、読み込み時間が短く拡張性の高いサイトを開発できます。このテンプレートは、最新の Lightning Web Runtime (LWR) と Lightning Web コンポーネント (LWC) プログラミングモデルに基づいています。

Aura コンポーネントの基本 (Trailhead モジュール)
Aura コンポーネントを使用し、再利用可能な UI コンポーネントによって最新 Web アプリケーションを構築します。Lightning コアコンポーネントの概念について学び、スタンドアロンアプリケーション、Salesforce モバイルアプリケーション、または Lightning Experience で実行できる単純な経費追跡アプリケーションを作成します。

Lightning Web コンポーネントの作成 (Trailhead トレイル)
JavaScript と HTML を使用して再利用可能な Lightning Web コンポーネントを開発します。

「クイックスタート: Aura コンポーネント」 (Trailhead プロジェクト)
組織の取引先責任者のリストを表示する最初のコンポーネントを作成します。

Build a Custom Theme Layout Component for Experience Builder Sites (エクスペリエンスビルダーサイト用のカスタムテーマレイアウトコンポーネントの構築) (Trailhead プロジェクト)
テーマレイアウトコンポーネントを使用して、エクスペリエンスビルダーサイトをカスタマイズします。

「Lightning Components Performance Best Practices (Lightning コンポーネントのパフォーマンスのベストプラクティス)」 (ブログ投稿)
コンポーネントのパフォーマンスに影響する Lightning の特性について解説し、コンポーネントを最適化するためのベストプラクティスを紹介します。

Experience Cloud のリソース

Experience Cloud に慣れていない方は、次のリソースを参考にしてください。

Experience Cloud サイトの設定および管理 (ヘルプ)
テンプレートを使用して、ブランド設定されたサイトを作成し、顧客やパートナーとオンラインで直接やりとりします。

Experience Cloud を使用したリーチの拡大 (Trailhead トレイル)
Salesforce Experience Cloud の使用開始に必要なツールについて学びます。

Experience Cloud の概要 (ヘルプ)
他の Experience Cloud リソースの最新情報を提供します。

Salesforce Lightning とは?

Salesforce Lightning は、あらゆるデバイス向けの応答性の高いアプリケーションの開発を容易にし、開発者に Lightning コンポーネントフレームワークと便利なツールを提供します。

Lightning には次のテクノロジーがあります。

- Lightning コンポーネントは、開発とアプリケーションのパフォーマンスを加速します。カスタムコンポーネントを開発すると、他の開発者やシステム管理者がそれを再利用可能なビルディングブロックとして使用し、エクスペリエンスビルダーサイト、Salesforce モバイルアプリケーション、Lightning Experience をカスタマイズできます。
- Lightning アプリケーションビルダーでは、システム管理者は標準およびカスタムの Lightning コンポーネントを使用して、コードを使用せずに視覚的に Lightning ページを構築できます。システム管理者がコードを使用せずにカスタムユーザーインターフェースを構築できるように、Lightning アプリケーションビルダーで Lightning コンポーネントを使用できるようにします。
- エクスペリエンスビルダーでは、システム管理者はテンプレートとコンポーネントを使用して、視覚的にサイトを構築できます。システム管理者がコードを使用せずにサイトページを構築できるように、エクスペリエンスビルダーで Lightning コンポーネントを使用できるようにします。

Lightning フレームワークで構築されている Salesforce 製品を次に示します。

- エクスペリエンスビルダーテンプレート
- Lightning Bolt ソリューション
- Lightning Experience
- Salesforce モバイルアプリケーション

メモ: Lightning Experience を有効化しなくても、エクスペリエンスビルダーテンプレートを使用したり、Lightning コンポーネントを開発したりできます。エクスペリエンスビルダーサイトでは Lightning Experience と同じ基盤技術を使用しますが、お互いに独立しています。

関連トピック:

Salesforce ヘルプ: Experience Cloud による Lightning の使用

Lightning Aura コンポーネント開発者ガイド

使用するエクスペリエンスビルダーテンプレートは?

使用するエクスペリエンスビルダーテンプレートに応じて Lightning Web コンポーネントモデルと従来の Aura コンポーネントモデルの 2 つのプログラミングモデルを使用して、エクスペリエンスビルダーサイトを作成できます。Build Your Own (LWR) テンプレートは新しい Lightning Web Runtime (LWR) に基づいており、Lightning Web コンポーネントでのみ使用できます。Aura コンポーネントでは使用できません。他のテンプレートは Aura コンポーネントモデルに基づいており、Lightning Web コンポーネントと Aura コンポーネントの両方を使用できます。

Build Your Own

すべてのエクスペリエンスビルダーサイトで必要とされる基本ページが提供されます (ホーム、レコードの作成、エラー、レコードの詳細、レコードリスト、関連レコードリスト、検索、パスワードを確認、パスワードを忘れた場合、ログイン、ログインエラー、登録)。必要に応じて、構築している環境にページやコンポーネントを追加します。サイトのデザインを調整するには、ブランドやテーマをカスタマイズします。

Build Your Own (LWR)

新しい Lightning Web Runtime (LWR) プラットフォームにより提供されるこのカスタマイズ可能なテンプレートでは、比類のないページパフォーマンスが提供され、開発者の生産性が向上します。ピクセル単位まで完璧なページをすばやく構築し、独自のブランドに合った Lightning Web コンポーネントおよびテーマを開発します。

このテンプレートはカスタム Lightning Web コンポーネントの開発と Salesforce DX、ユーザーインターフェース API、Apex の操作に慣れている開発者、コンサルティングパートナー、および ISV に最適です。

「LWR Sites for Experience Cloud (Experience Cloud の LWR サイト)」ガイドを参照してください。

カスタマー取引先ポータル

顧客が取引先情報にアクセスし、情報を更新できる非公開の安全な場所。顧客がポータルで作業できるようにすれば、カスタマーリレーションが向上し、コストが削減します。顧客が請求書を参照して支払を行ったり、取引先情報を更新したり、よくある質問に対する回答を知識ベースで検索したりすることができます。

カスタマーサービス

複数の事前作成済みテーマオプションが用意された、強力で反応型のセルフサービステンプレート。カスタマーサービステンプレートでは、ユーザーがコミュニティへの質問の投稿、記事の検索および表示、コラボレーション、ケースを作成しサポートエージェントへの問い合わせを行うことができます。ナレッジ、Chatter の質問、およびケースをサポートします。

Partner Central

チャネル販売のワークフロー用に設計された柔軟で応答性の高いテンプレートです。パートナーネットワークを採用、構築、拡大して、オンラインのブランド空間でチャネル販売とチャネルマーケティングを共に

促進します。リードの配布、商談の登録、およびマーケティングキャンペーンを簡単に設定します。また、トレーニングおよび販促用の資料を中央リポジトリで共有し、レポートを使用してパイプラインを追跡します。

ヘルプセンター
: 知識ベースから利用できるようにした記事が公開されるセルフサービスコミュニティ。カスタマーサポートチームの負担を軽減でき、ユーザーは自分でソリューションを見つけられたという満足感を得ることができます。

関連トピック:

Salesforce ヘルプ: 使用すべき Experience Cloud テンプレートは?

Experience Cloud 開発者リリースノート

Experience Cloud 開発者エクスペリエンスの最新の更新や変更点については、Salesforce のリリースノートを参照してください。

新しい機能と変更された機能については、Salesforce リリースノートの Experience Cloud のセクションにある「開発者の生産性」を参照してください。また、Salesforce リリースノートの「開発者向けの新規および変更された項目」にも Experience Cloud に関する情報が記載されています。

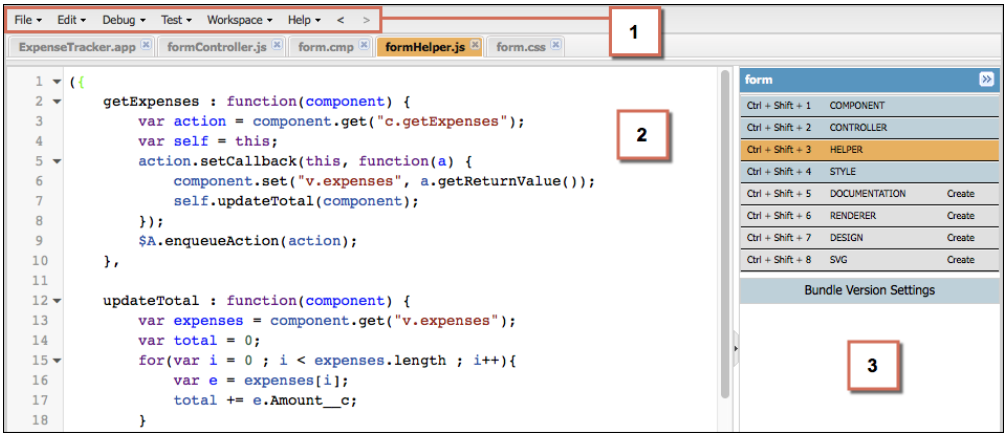
第 2 章　エクスペリエンスビルダーサイトの開発: 基本

トピック:

開発者コンソール開発ツール、基本的なドラッグアンドドロップ Aura コンポーネントの作成方法、およびその過程で考慮すべきヒントについて説明します。

開発者コンソールの使用

開発者コンソールには、Auraコンポーネントおよびアプリケーションを開発するためのツールが用意されています。



開発者コンソールでは、次の機能を実行できます。

- メニューバー (1) を使用して、次の Lightning リソースを作成したり、開いたりする。
  - アプリケーション
  - コンポーネント
  - インターフェース
  - 行動
  - トークン
- ワークスペース (2) を使用して、Lightning リソースを操作する。
- サイドバー (3) を使用して、特定のコンポーネントのバンドルに含まれるクライアント側のリソースを作成したり、開いたりする。
  - コントローラー
  - ヘルパー
  - スタイル
  - ドキュメント
  - レンダラー
  - 設計
  - SVG

関連トピック:

Salesforce ヘルプ: 開発者コンソールを開く

Lightning Aura コンポーネント開発者ガイド: 開発者コンソールでの Aura コンポーネントの作成

Lightning Aura コンポーネント開発者ガイド: コンポーネントのバンドル

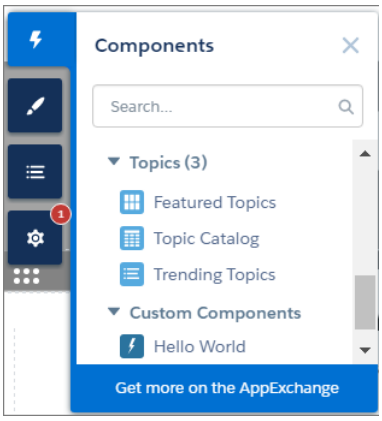
エクスペリエンスビルダーのドラッグアンドドロップ Aura コンポーネントの設定

エクスペリエンスビルダーでカスタム Aura コンポーネントを使用する前に、いくつかの設定ステップを実行する必要があります。

1.インターフェースをコンポーネントに追加する

エクスペリエンスビルダーにドラッグアンドドロップコンポーネントとして表示するには、コンポーネントに `forceCommunity:availableForAllPageTypes` インターフェースを実装する必要があります。

Aura コンポーネントを作成すると、そのコンポーネントは、組織のすべての Aura サイトの [コンポーネント] パネルに表示されます。



シンプルな「Hello World」コンポーネントのサンプルコードを次に示します。コンポーネントの名前は *componentName*.cmp にする必要があります。

メモ: コンポーネントなどのリソースを自分の組織外で使用できるようにするには、`access="global"` でマークします。たとえば、インストール済みパッケージで、または他の組織のエクスペリエンスビルダーユーザーがコンポーネントを使用できるようにするには、`access="global"` を使用します。

```
<aura:component implements="forceCommunity:availableForAllPageTypes" access="global">
    <aura:attribute name="greeting" type="String" default="Hello" access="global" />
    <aura:attribute name="subject" type="String" default="World" access="global" />

    <div>{!v.greeting}, {!v.subject}!</div>
</aura:component>
```

警告: サイトにカスタムコンポーネントを追加すると、ゲストユーザープロファイルに設定したオブジェクトレベルセキュリティと項目レベルセキュリティ (FLS) をスキップできます。Lightning コンポーネントでは、オブジェクトを参照したり、Apex コントローラーからオブジェクトを取得したりするときに、CRUD および FLS が自動的に適用されることはありません。つまり、このフレームワークでは、ユーザーに CRUD 権限および FLS 表示権限がないレコードと項目は引き続き表示されます。CRUD と FLS は、Apex コントローラーで手動によって適用する必要があります。

2.デザインリソースをコンポーネントバンドルに追加する

デザインリソースは、エクスペリエンスビルダーで公開するコンポーネント属性を制御します。デザインリソースは .cmp リソースと同じフォルダーに存在します。このリソースには、Aura コンポーネントの設計時の動作(ページまたはアプリケーションでコンポーネントを表示するためにビジュアルツールが必要とする情報)が記述されます。

たとえば、属性のデフォルト値を設定したり、システム管理者が Aura コンポーネントの属性をエクスペリエンスビルダーで編集できるようにしたりするには、コンポーネントバンドルでデザインリソースが必要になります。

「Hello World」コンポーネントと一緒にバンドルするデザインリソースを次に示します。デザインリソースの名前は、*componentName*.design にする必要があります。

```
<design:component label="Hello World">
    <design:attribute name="greeting" label="Greeting" />
    <design:attribute name="subject" label="Subject" description="Name of the person you
want to greet" />
</design:component>
```

省略可能。SVG リソースをコンポーネントバンドルに追加する

コンポーネントのカスタムアイコンを定義するには、コンポーネントバンドルに SVG リソースを追加します。エクスペリエンスビルダーの [コンポーネント] パネル内のコンポーネントの横にアイコンが表示されます。

SVG リソースを含めないと、デフォルトのアイコン ( ) が使用されます。

「Hello World」コンポーネントと一緒に表示するシンプルな赤い円の SVG リソースの例を次に示します。SVG リソースの名前は *componentName*.svg にする必要があります。

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg xmlns="http://www.w3.org/2000/svg"
     width="400" height="400">
  <circle cx="100" cy="100" r="50" stroke="black"
    stroke-width="5" fill="red" />
</svg>
```

省略可能。CSS リソースをコンポーネントバンドルに追加する

カスタムコンポーネントのスタイルを定義するには、コンポーネントバンドルに CSS リソースを追加します。

「Hello World」コンポーネントと一緒に表示するシンプルなクラスの CSS を次に示します。CSS リソースの名前は *componentName*.css にする必要があります。

```
.THIS .greeting {
    color: #ffe4e1;
    font-size: 20px;
}
```

クラスを作成したら、コンポーネントに適用します。

```
<aura:component implements="forceCommunity:availableForAllPageTypes" access="global">
    <aura:attribute name="greeting" type="String" default="Hello" access="global" />
    <aura:attribute name="subject" type="String" default="World" access="global" />

    <div class="greeting">{!v.greeting}, {!v.subject}!</div>
</aura:component>
```

関連トピック:

Lightning Aura コンポーネント開発者ガイド: Aura コンポーネントのブラウザーサポート

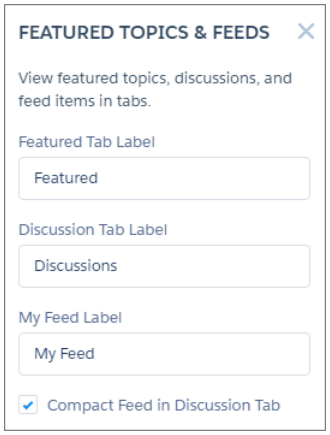
エクスペリエンスビルダーでのコンポーネント属性の公開

デザインリソースを使用して、エクスペリエンスビルダーで公開する属性を制御します。デザインリソースは、コンポーネントと同じフォルダーにあります。デザインリソースには、Auraコンポーネントの設計時の動作(ページまたはアプリケーションにコンポーネントを表示するためにビジュアルツールが必要とする情報)が記述されます。

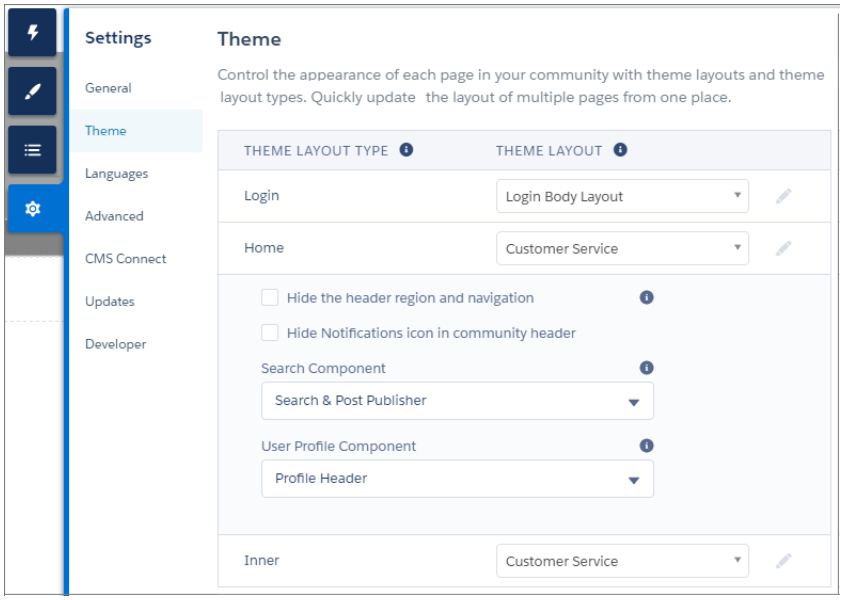
Auraコンポーネントの属性をシステム管理者がエクスペリエンスビルダーで編集できるようにするには、属性の `design:attribute` ノードをデザインリソースに追加します。属性を必須とマークすると、デフォルト値が割り当てられている場合を除き、エクスペリエンスビルダーに自動的に表示されます。

コンポーネント定義内のデフォルト値が設定された必須属性と必須とマークされていない属性をユーザーに表示するには、デザインリソースでそれらを指定する必要があります。デザインリソースでは、`int`、`string`、または `boolean` 型の属性のみがサポートされます。

ドラッグアンドドロップコンポーネントの場合、公開された属性はコンポーネントのプロパティパネルに表示されます。



テーマレイアウトコンポーネントの場合、公開された属性は [設定] > [テーマ] 領域でそのテーマレイアウトが選択されている場合に表示されます。



関連トピック:

Lightning Aura コンポーネント開発者ガイド: Aura コンポーネントバンドルのデザインリソース

エクスペリエンスビルダーの Aura コンポーネントの設定に関するヒントおよび考慮事項

Aura サイトの Aura コンポーネントおよびコンポーネントのバンドルを作成する場合、次のガイドラインを参考にしてください。

コンポーネント

- `<design:component label="foo">` などのデザインファイル要素の `label` 属性を使用して、コンポーネントにわかりやすい名前を付けます。
- 表示領域の 100% の幅 (余白を含む) を占めるようにコンポーネントを設計します。
- ユーザー操作を必要とする場合、宣言型ツールでコンポーネントの適切なプレースホルダー動作を指定します。
- コンポーネントに空白のボックスが表示されないようにしてください。他のサイトがどのように動作するかを考えます。たとえば、Facebook では、フィード項目がサーバーから返されるまでフィードの概要が表示されます。これにより、UI 応答のユーザーの認識が向上します。
- 起動されたイベントにコンポーネントが連動する場合は、イベントが起動される前に表示するデフォルトの状態を指定します。
- 標準設計トークンを使用してコンポーネントのスタイルを設定し、Salesforce Design System との一貫性を保ちます。
- 組織とサイトで Lightning Locker が有効になっている場合、Lightning Locker が、Summer '17 (API バージョン 40.0) 以降で作成されたすべての Aura コンポーネントに適用されます。組織レベルでは、Lightning Web Security が

有効化されていない場合、Lightning Locker が使用されます。「セキュアサイトの開発: CSP、LWS、および Lightning Locker」を参照してください。

- カスタムコンポーネントの新しいプロパティをエクスペリエンスビルダーで編集できるようにする場合は、サイトの翻訳に関する次の考慮事項に留意してください。エクスペリエンスビルダーのページでコンポーネントが使用されている場合、そのコンポーネントをページから削除して、更新済みバージョンに置き換えます。置き換えない場合、サイトのコンテンツを翻訳用にエクスポートすると、エクスポートしたファイル内の該当のコンポーネントインスタンスに追加されたプロパティは省略されます。すでに翻訳されたコンテンツがコンポーネントに含まれている場合は、まずサイトのコンテンツをエクスポートして既存の翻訳を保持します。次に、コンポーネントを更新済みバージョンに置き換えます。

属性

- デザインファイルを使用して、エクスペリエンスビルダーに公開する属性を制御します。
- システム管理者にとって使いやすくわかりやすい属性にします。SOQL クエリ、JSON オブジェクト、Apex クラス名は公開しません。
- ユーザーの操作性が低下しないように、必須属性にはデフォルト値を指定します。デフォルト値のない必須属性を持つコンポーネントをエクスペリエンスビルダーに追加すると、無効と表示されます。
- 公開される属性には、サポートされる基本のデータ型 (`string`、`integer`、`boolean`) を使用します。
- `<design:attribute>` 要素の整数属性に最小値と最大値を指定して、有効な値範囲を制御します。
- 文字列属性では、事前定義された一連の値を持つデータ取得元を指定して、属性の設定を選択リストとして公開できます。
- 属性に、わかりやすい表示名を使用した表示ラベルを指定します。
- 説明を含めて、データ形式や予期される値範囲など、予期されるデータおよびガイドラインを説明します。説明テキストは、プロパティパネルにツールチップとして表示されます。

- `forceCommunity:availableForAllPageTypes` インターフェースを実装するコンポーネントの設計属性を削除するには、まずコンポーネントからインターフェースを削除した後、設計属性を削除します。その後でインターフェースを再実装します。コンポーネントがサイトページで参照されている場合は、変更する前にページからコンポーネントを削除する必要があります。

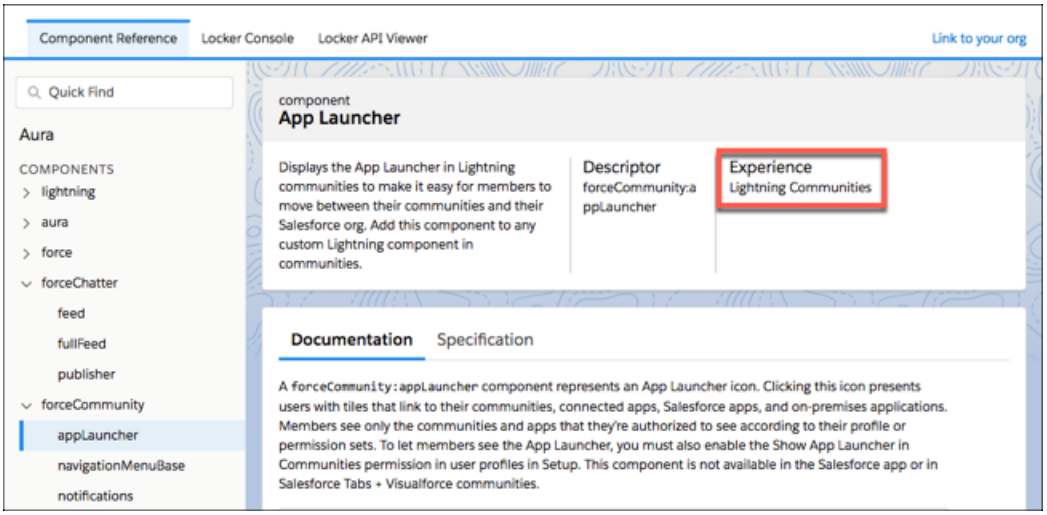
関連トピック:

Lightning Aura コンポーネント開発者ガイド: アプリケーションでの Salesforce Lightning Design System の使用
Lightning Aura コンポーネント開発者ガイド: 設計トークンを使用したスタイル設定
Lightning Aura コンポーネント開発者ガイド: Aura コンポーネントバンドルのデザインリソース

サポートされる Aura コンポーネント、インターフェース、イベント

すべての Aura コンポーネント、インターフェース、イベントが Aura ベースのエクスペリエンスビルダーサイトでサポートされているわけではありません。一部は、Salesforce モバイルアプリケーションまたは Lightning Experience でのみ使用できます。サイトをカスタマイズする前に、何が使用可能かを確認してください。

Aura コンポーネント、インターフェース、イベントは、コンポーネントライブラリに記載されています。各コンポーネント、インターフェース、イベントには、サポート対象のエクスペリエンスが示されます。



関連トピック:

- *コンポーネントの参照*
- *インターフェースの参照*
- *イベントの参照*

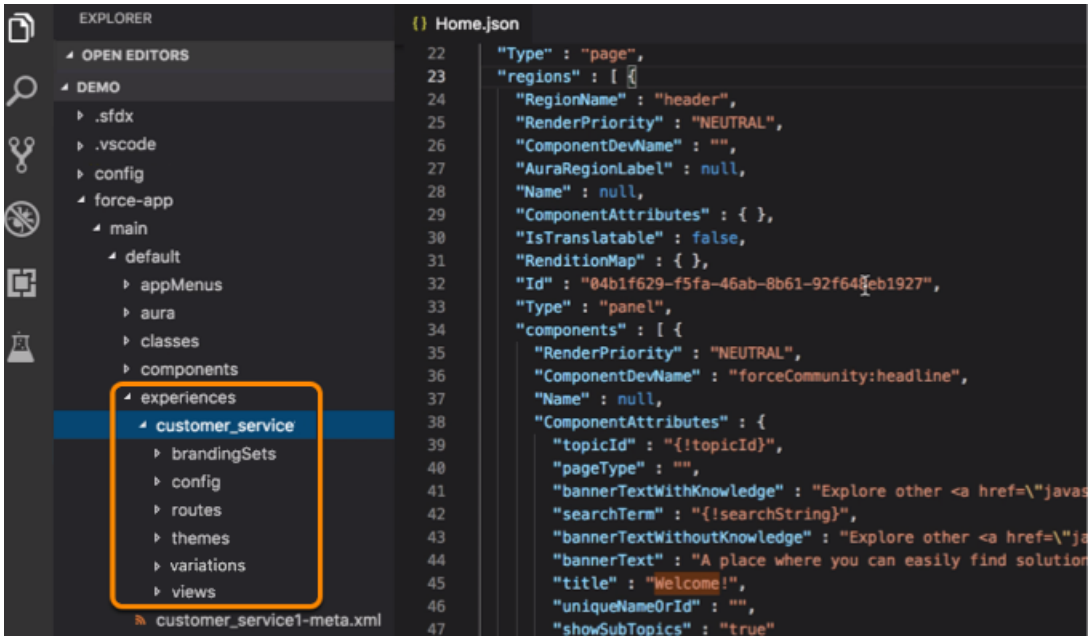
パーソナライズ対象の API 参照名とグループ名

Connect REST API またはメタデータ API を使用してエクスペリエンスビルダーサイトをパーソナライズする場合、エクスペリエンスバリエーション対象の API 参照名とグループ名を決定します。

Audience メタデータ型の Connect Rest API または PersonalizationTargetInfo サブタイプの Target Input リクエストボディの場合、以下を指定する必要があります。

- `groupName` プロパティの対象のグループ名
- `targetValue` プロパティの対象の API 参照名

これらの名前を確認するには、サイトの ExperienceBundle フォルダーのいくつかの JSON ファイルからプロパティ値をコピーする必要があります。



これらの値を確認する方法は、対象 (ページのバリエーション、ブランドセット、コンポーネントの表示、コンポーネントの属性) によって異なります。

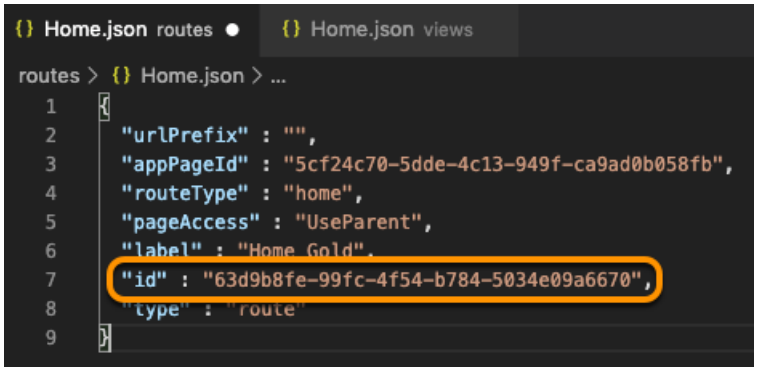
ページのバリエーション

ページのバリエーションのグループ名とAPI参照名を確認するには、関連するルートファイルと対応するビューファイルを開きます。たとえば、routes フォルダーの `Home.json` ファイルと views フォルダーの `Home.json` ファイルを開きます。

グループ名

形式: ***`route.id`***。*`route.id`* は、ルート JSON ファイルの `id` です。

例: `63d9b8fe-99fc-4f54-b784-5034e09a6670`

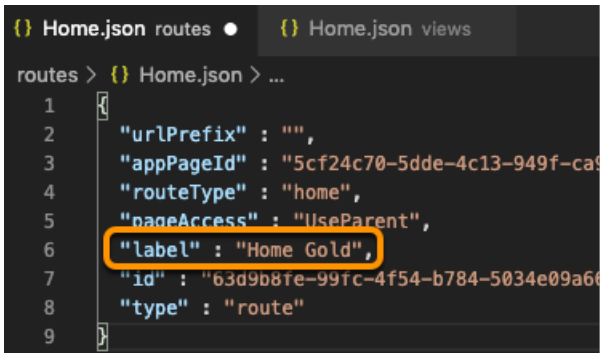


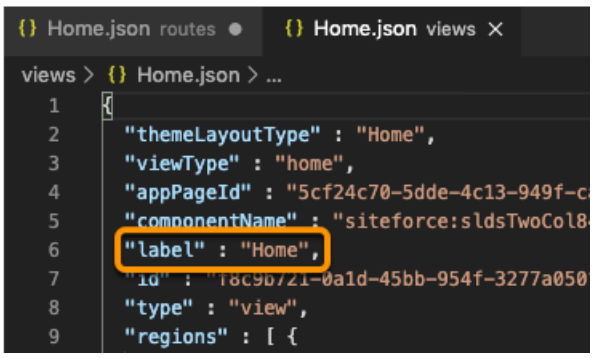
開発者名

形式: ***`route.label_view.label`***_Page。

- *`route.label`* は、ルート JSON ファイルの `label` プロパティです。
- *`view.label`* は、ビュー JSON ファイルの `label` プロパティです。

例: `Home_Gold_Home_Page`





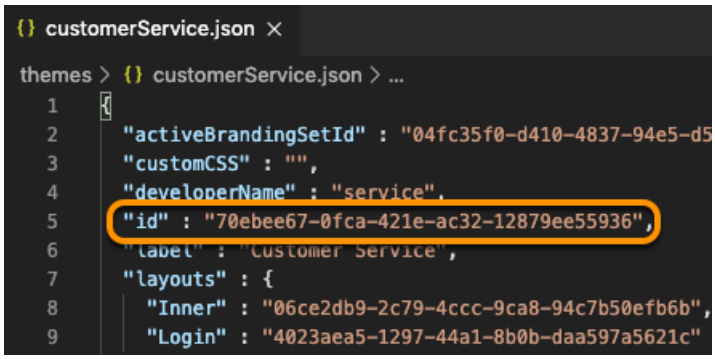
ブランドセット

ブランドセットのグループ名と API 参照名を確認するには、関連するブランドセットファイルと対応するテーマファイルを開きます。たとえば、カスタマーサービスサイトの場合、themes フォルダーの `customerService.json` ファイルと brandingSets フォルダーの `customerService.json` ファイルを開きます。

グループ名

形式: ***`theme.id`***`$#$Branding`。*`theme.id`* は、テーマ JSON ファイルの `id` プロパティです。

例: `70ebee67-0fca-421e-ac32-12879ee55936$#$Branding`

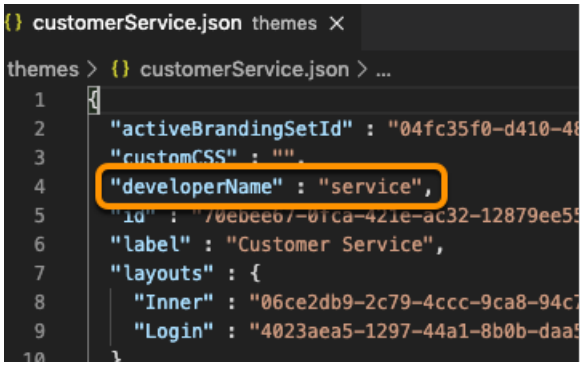


開発者名

形式: ***`theme.developerName_brandingSet.label`***`_Branding`。

- *`theme.developerName`* は、テーマ JSON ファイルの `developerName` プロパティです。
- *`brandingSet.label`* は、ブランドセット JSON ファイルの `label` プロパティです。

例: `service_Customer_Service_Branding`



コンポーネントの表示

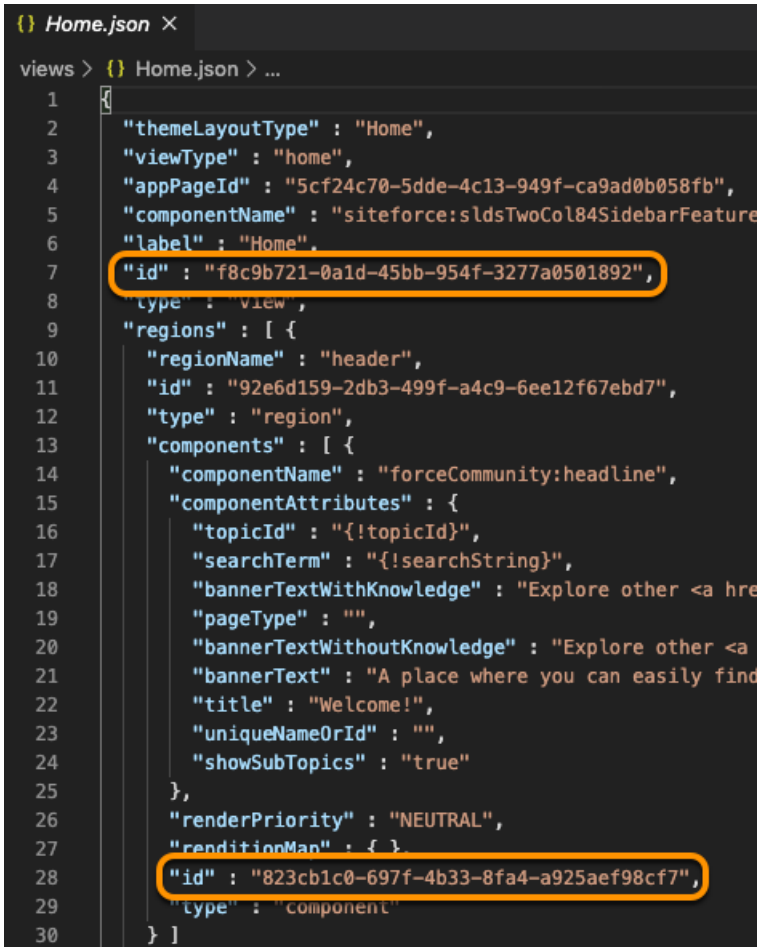
コンポーネントの表示のグループ名と API 参照名を確認するには、コンポーネントが含まれているビューファイルを開きます。たとえば、views フォルダーの `Home.json` ファイルを開きます。

グループ名

形式: ***`view.id`*`$#$`*`component.id`**。

- *`view.id`* は、ビュー JSON ファイルの `id` プロパティです。
- *`component.id`* は、ビュー JSON ファイルのコンポーネントの `id` プロパティです。

例: `f8c9b721-0a1d-45bb-954f-3277a0501892$#$823cb1c0-697f-4b33-8fa4-a925aef98cf7`

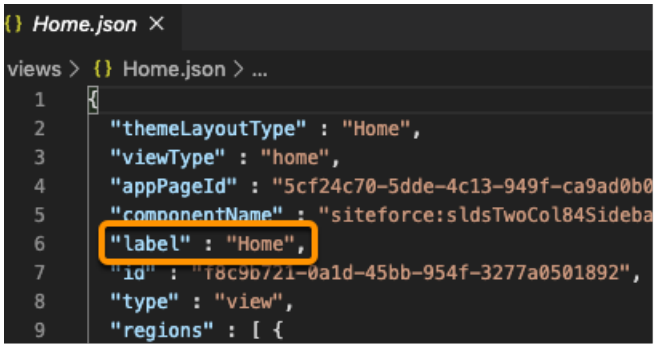


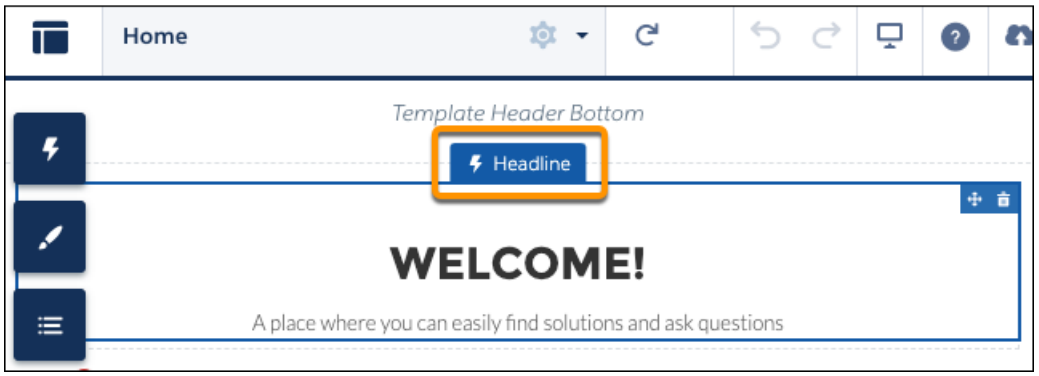
開発者名

形式: ***view.label_componentName***_Component。

- *view.label* は、ビュー JSON ファイルの `label` プロパティです。
- *componentName* は、JSON ファイルではなくエクスペリエンスビルダーのコンポーネントの名前です。

例: `Home_Headline_Component`





メモ: 必要に応じて、一意になるように数値を API 参照名に追加することもできます。例: `Home_Page_Rich_Content_Editor_Component1`。

コンポーネントの属性

コンポーネントの属性の場合、グループ名と API 参照名は、コンポーネントがビュー本体にあるのか、テーマレイアウトのヘッダーまたはフッターにあるのかによって異なります。

ビュー本体のコンポーネント

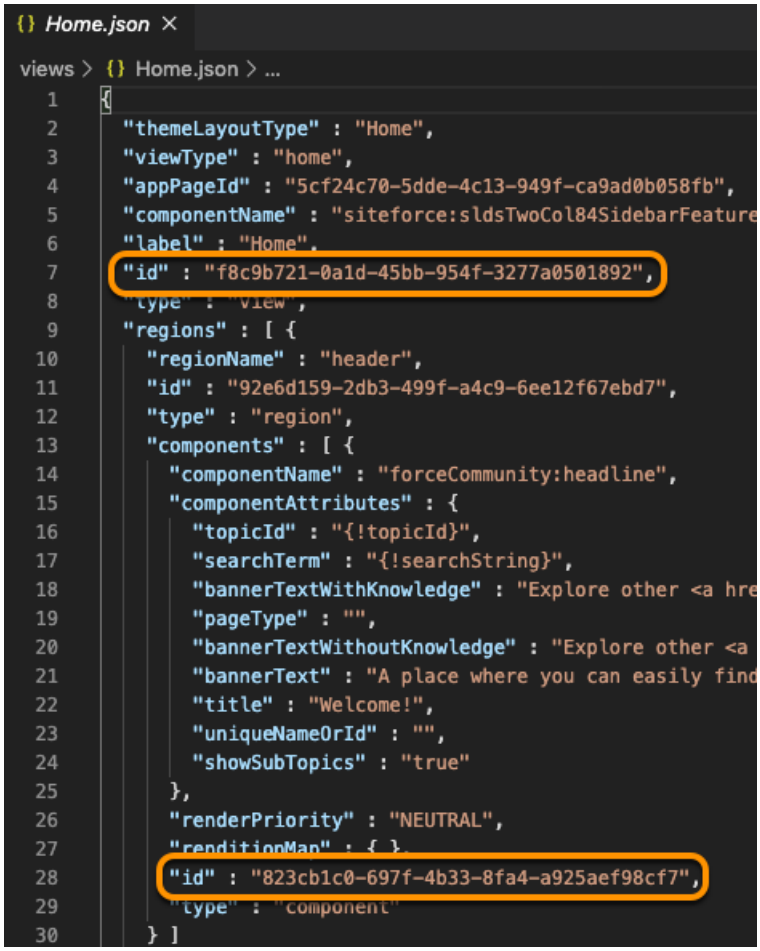
ビュー本体のコンポーネントのグループ名とAPI参照名を確認するには、コンポーネントが含まれているビューファイルを開きます。たとえば、views フォルダーの `Home.json` ファイルを開きます。

グループ名

形式: ***`view.id`*`$#$`*`component.id`***。

- *`view.id`* は、ビュー JSON ファイルの `id` プロパティです。
- *`component.id`* は、ビュー JSON ファイルのコンポーネントの `id` プロパティです。

例: `f8c9b721-0a1d-45bb-954f-3277a0501892$#$823cb1c0-697f-4b33-8fa4-a925aef98cf7`

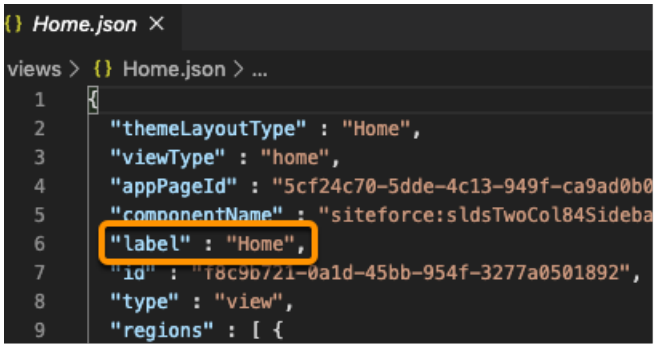


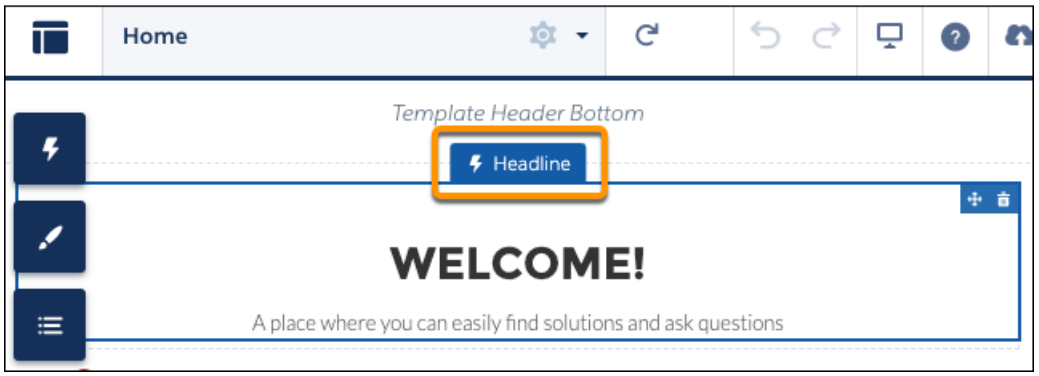
開発者名

形式: ***view.label_componentName***_Component_Properties。

- *view.label* は、ビュー JSON ファイルの `label` プロパティです。
- *componentName* は、JSON ファイルではなくエクスペリエンスビルダーのコンポーネントの名前です。

例: `Home_Headline_Component_Properties`





メモ: 必要に応じて、一意になるように数値を API 参照名に追加することもできます。例; `Home_Page_Rich_Content_Editor_Component1`。

ヘッダーまたはフッターのコンポーネント

テーマレイアウトのヘッダーまたはフッターのコンポーネントのグループ名と API 参照名を確認するには、コンポーネントが含まれているテーマファイルを開きます。たとえば、カスタマーサービスサイトの場合、themes フォルダーの `customerService.json` ファイルを開きます。

グループ名

形式: ***`themeLayout.id`*****`$#$`*****`component.id`**。

- *`themeLayout.id`* は、コンポーネントが含まれているレイアウトの `id` プロパティです。
- *`component.id`* は、レイアウトのコンポーネントの `id` プロパティです。

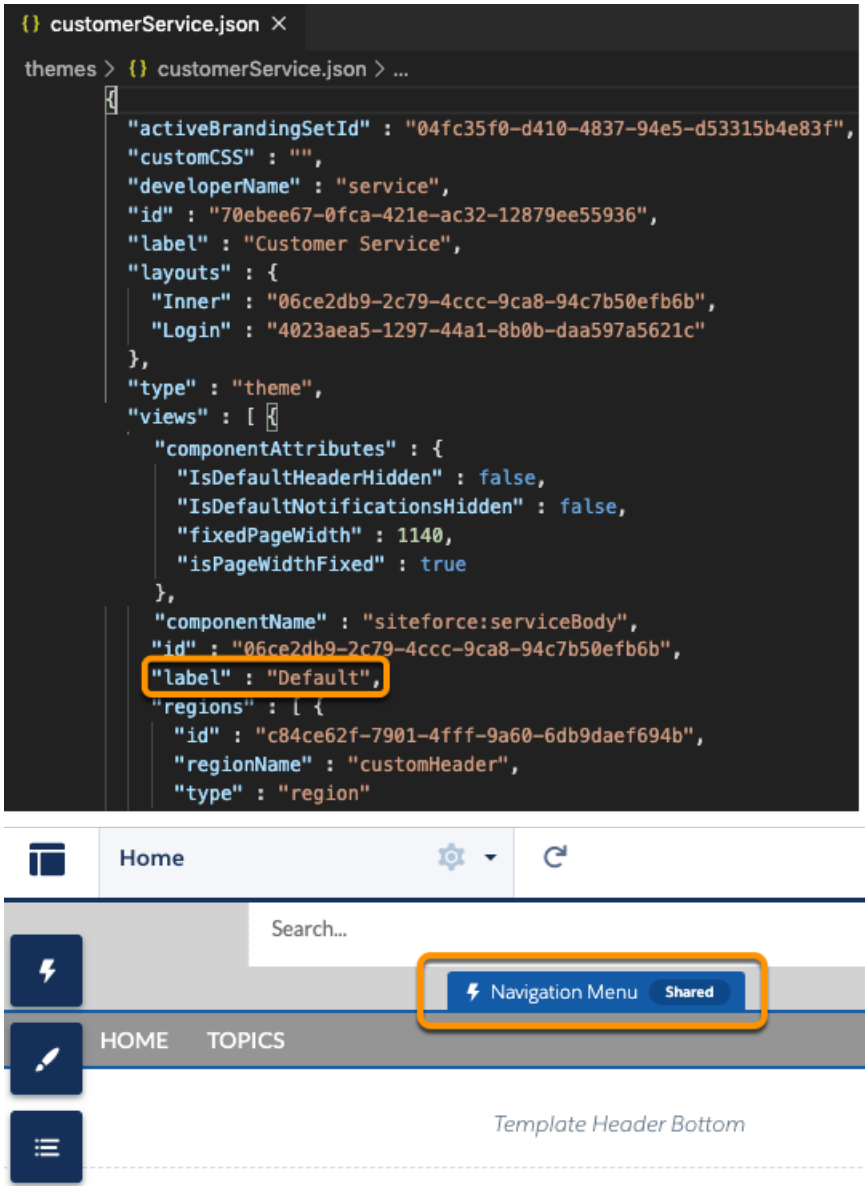
例: `06ce2db9-2c79-4ccc-9ca8-94c7b50efb6b$#$c55d1908-fe6b-47e8-b41e-70ad05aeb490`

開発者名

形式: ***themeLayout.label_componentName***_Component_Properties。

- *themeLayout.label* は、コンポーネントが含まれているレイアウトの `label` プロパティです。
- *componentName* は、JSON ファイルではなくエクスペリエンスビルダーのコンポーネントの名前です。

例: `Default_Navigation_Menu_Component_Properties`



関連トピック:

メタデータ API 開発者ガイド: 利用者

Connect REST API 開発者ガイド: Target Input

ユーザーの個人情報表示設定への準拠

ポータルやサイトがある組織では、ユーザーの個人識別情報や連絡先情報が他のユーザーに表示されないようにする固有の設定が提供されます。これらの設定は、`WITH SECURITY_ENFORCED` 句や `stripInaccessible` メソッドなどの Apex セキュリティ機能があっても、Apex では適用されません。特定の項目がゲストや外部の承認済みユーザーに表示されないようにするには、以下のサンプルコードに従います。

ユーザーオブジェクトのユーザーの個人情報を非表示にする方法:

```
public User[] fetchUserDetail(Set userIds) {
    // Query all the fields of user which we are expected in user record to show that on
UI or to
    // perform some business logic.
    User[] userRecords = [SELECT id, username, communitynickname, firstname, lastname,
title
    FROM User WHERE id IN :userIds];

    for (User userRecord : userRecords) {
        // User is not fetching his own record and is not standard user.
       if(userRecord.id != UserInfo.getUserId() && !Auth.CommunitiesUtil.isInternalUser())
 {
            // clear-out all PII fields form user record which we have queried above.
            userRecord.username = '';
            userRecord.title = '';
        }
    }
    return userRecords;
}
```

コミュニティまたはポータル内のユーザーの連絡先情報表示設定に準拠するために、特定の項目に関連付けられた設定がチェックされ、それに応じてデータの表示/非表示が決まります。Experience Cloud サイト内のユーザーの連絡先表示設定:

```
public User[] fetchUserRecordRespectingFLVPreferences(Set<Id> userIds) {

    //Fetch users records along with fields specific user preferences.
    User[] userRecords = [SELECT email, UserPreferencesShowEmailToExternalUsers,
UserPreferencesShowEmailToGuestUsers FROM User WHERE id IN :userIds];

    // If context user is internal user then return result without any restriction.
    if (Auth.CommunitiesUtil.isInternalUser()) {
        return userRecords;
    }

    // If user is guest user then return result as per the user's UserPreference for the
fields related to the Guest user visibility.
    if (Auth.CommunitiesUtil.isGuestUser()){
        return fetchUserRecordForGuestUser(userRecords);
    }

   // Return result as per the user's UserPreference for the fields related to the External
 user visibility
    return fetchUserRecordForExternalUser(userRecords);

}

// Apply Field level visibilty logic by checking user's UserPreferences for the fields
related to the External user visibility.
public User[] fetchUserRecordForExternalUser(User[] userRecords) {

    for(User userRecord : userRecords) {
```

```
        //Clear field of user record when context user fetching other user's record and
Field Level Visibility for that field is set to Restricted.
        if(userRecord.id != UserInfo.getUserId() &&
!userRecord.UserPreferencesShowEmailToExternalUsers)
        {
            userRecord.email = '';
        }

    }

    return userRecords;
}

// Apply Field level visibilty logic by checking user's UserPreferences for the fields
related to the Guest user visibility.
public User[] fetchUserRecordForGuestUser(User[] userRecords) {

    for(User userRecord : userRecords) {

        //Clear field of user record when context user fetching other user's record and
user preference for that field is NOT set to public.
        if(!userRecord.UserPreferencesShowEmailToGuestUsers)
        {
            userRecord.email = '';
        }

    }

    return userRecords;

}
```

関連トピック:

Salesforce ヘルプ: 外部ユーザーへの個人ユーザー情報の表示の管理

Salesforce ヘルプ: Experience Cloud サイト内での個人連絡先情報の共有

第 3 章　エクスペリエンスビルダーテンプレートの外観のカスタマイズ

トピック:

エクスペリエンスビルダーテンプレートの外観は、複雑さと粒度がそれぞれ異なるいくつかの方法で制御できます。

エクスペリエンスビルダーでは、テンプレート特有の(そのためサイト間での共有はできない)スタイルを編集できます。エクスペリエンスビルダーのオプションは最もシンプルで、コーディングは不要です。

- [テーマ] パネルは、シンプルなポイント & クリックプロパティでテンプレートを更新します。この手法は、システム管理者が使用するのに理想的です。

  メモ: 個別のコンポーネントプロパティパネル (ヘッダーや検索など) では、それぞれに特有の外観の調整が可能です。

- CSS エディターを使用すると、テンプレート要素の基本スタイルを上書きするカスタム CSS を作成できます。このオプションは、CSS に慣れていて、標準のコンポーネントやテンプレート要素を少しだけ変更したい場合に適しています。

  メモ: テーマの交換が導入されたため、作成したカスタム CSS は有効なテーマに直接関連付けることができるようになりました。既存のサイトでは、現在使用されている必要なカスタムCSSが新たに選択したテーマにコピーされるようにしてください。カスタムCSSが必要な場合は、テーマレイアウトの適切な場所 (`brandLogoImage`、`action color` など) でエクスペリエンスビルダーサイト用のデザイントークンを使用して、テンプレートやテーマを更新しやすいようにしてください。

ただし、テンプレートの外観を完全にカスタマイズするには、自身のコンポーネントを作成する必要があります。

- カスタム Aura コンポーネントでは CSS リソースがコンポーネントバンドルの一部としてカプセル化されるため、サイト間でコンポーネントを再利用できます。
- コンテンツレイアウトコンポーネントは、ページのコンテンツ領域を定義し、コンテンツを含みます。
- テーマレイアウトコンポーネントは、テンプレートの構造上のレイアウト (ヘッダーやフッターなど) をカスタマイズして、デフォルトのスタイルを上書きします。

エクスペリエンスビルダーテンプレートの外観のカスタマイズ

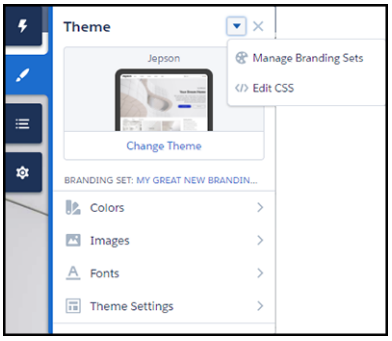
ポーネントのセキュリティの確保

[テーマ] パネルでのテンプレートの更新

エクスペリエンスビルダーでテンプレートの外観を変更するのに最もシンプルな方法は[テーマ]パネルを使用することです。システム管理者は、[テーマ] パネルを使用して色を適用し、フォントを指定し、ロゴを追加し、全体的なページ構造とデフォルトを調整することで、サイト全体のスタイルを簡単に設定できます。

[テーマ]パネルで設定されたプロパティは、テンプレートのページとほとんどの既製コンポーネントに適用されます。ブランドセットを使用すると、色、画像、フォントのバンドルを簡単に適用できます。

標準デザイントークンを使用するカスタム Lightning コンポーネントに [テーマ] パネルのプロパティを適用することで、コンポーネントの外観を制御できます。



メモ: Spring '17 のテンプレートを引き続き使用している場合、ログインページのテーマプロパティをサイトの残りのページと統合するには、[設定] > [更新] でテンプレートを更新します。そうでない場合は、ログインページのテーマを別に設定する必要があります。

関連トピック:

Salesforce ヘルプ: エクスペリエンスビルダーの事前作成済みテーマ

カスタム CSS でのテンプレート要素の上書き

エクスペリエンスビルダーで CSS エディターを使用して、デフォルトのテンプレートと [テーマ] パネルのスタイルを上書きするカスタム CSS を追加します。また、パディングの調整など、標準コンポーネントの外観を少し変更することもできます。

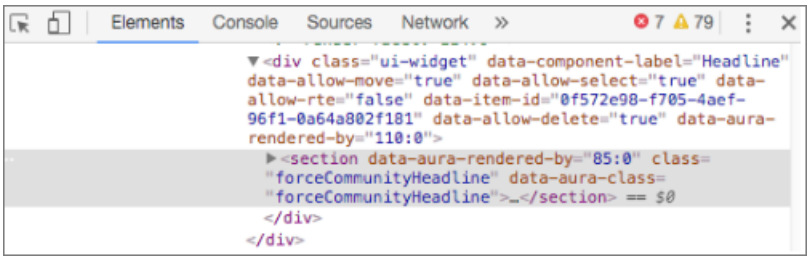
重要: カスタム CSS は慎重に使用してください。テンプレートコンポーネントの今後のリリースで、使用中の CSS カスタマイズがサポートされない可能性があります。また、Salesforce カスタマーサポートがカスタム CSS の問題の解決をサポートできない可能性があります。

ヒント: カスタム CSS を使用するのではなく、テンプレートそのものをカスタマイズするには、カスタム Lightning Web コンポーネントまたは Aura コンポーネントとカスタムテーマレイアウトコンポーネントで CSS リソースを使用します。グローバルな上書きを使用する場合は、各リリースで更新されるたびに必ず Sandbox でサイトをテストしてください。

テンプレート項目の CSS に若干の変更を加える場合は、Chrome DevTools を使用してページを調査し、項目の完全修飾名と CSS クラスを見つけます。そして、この情報を使用して、項目の標準 CSS をカスタム CSS で上書き

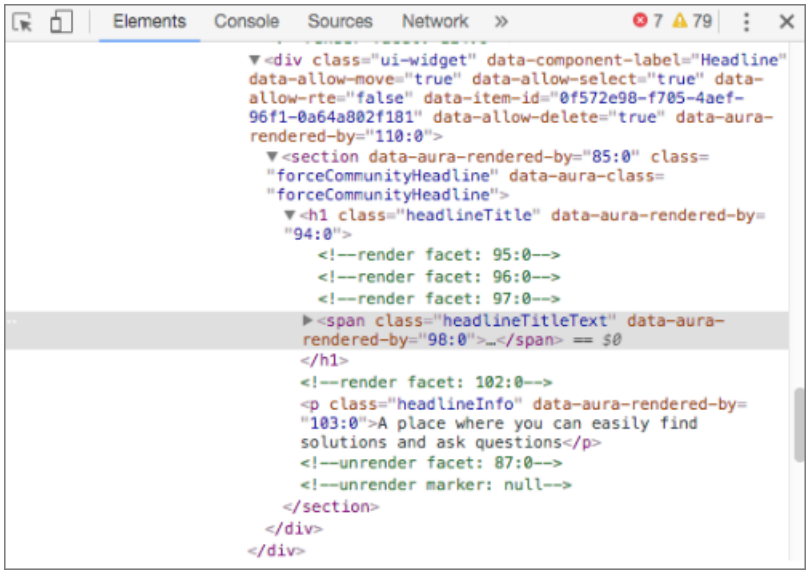
します。ページとスタイルの調査と編集についての詳細は、「Google Chrome DevTools」の Web サイトを参照してください。

コンポーネントを調査するのに最も簡単な方法は、プレビューモードでページを表示することです。次の例では、[見出し] コンポーネントを調査して、コンポーネントの完全修飾名 (`forceCommunityHeadline`) を特定しています。



メモ: コンポーネントの最上位の CSS クラスが定義されていない場合は、このオプションは表示されないため、コンポーネントを正確に特定することはできません。

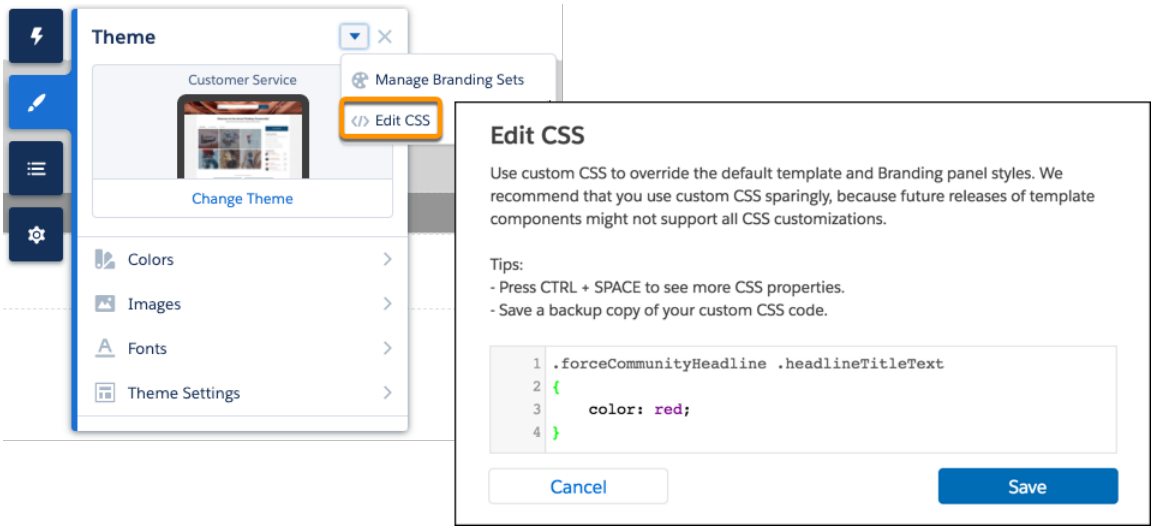
次に、スタイルを適用する要素 (例: `headlineTitleText`) を探します。要素にクラス名がない場合は、要素を対象とした特定のセレクターを作成する必要があります。



この情報を元に、デフォルトのタイトル色を上書きするためのカスタムスタイルを作成できます。

```
.forceCommunityHeadline .headlineTitleText
{
    color: red;
}
```

次に、このスタイルを CSS エディターに追加します。



同様に、カスタム CSS を使用して、コンポーネント全体を非表示することもできます。

```
.forceCommunityHeadline
{
    display: none;
}
```

ヒント: [設定] > [詳細] のヘッドマークアップで、静的リソースまたは外部リソースとして CSS スタイルシートにリンクできます。ただし、ヘッドマークアップやCSS上書きではグローバル値プロバイダーはサポートされないため、`$resource` を使用して静的リソースを参照することはできません。代わりに、`/sfsites/c/resource/resource_name` の構文で相対 URL を使用します。

たとえば、画像を静的リソースとして「Headline (見出し)」という名前でアップロードした場合は、CSS エディターでは次のように参照します。

```
.forceCommunityHeadline
{
    background-image: url('/sfsites/c/resource/headline')
}
```

ヘッドマークアップは、お気に入りアイコン、SEO メタタグ、および他の項目を追加するのにも便利です。ただし、デフォルトの厳格なCSPセキュリティレベルを使用するとコードに影響が出る場合があることを忘れないでください。

CSS 上書きの移行

Spring '17 から Winter '19 の間に、いくつかのエクスペリエンスビルダーコンポーネントの CSS セレクターが更新されました。それ以降テンプレートを更新しておらず、サイトでカスタム CSS を使用してデフォルトのテンプレートとテーマパネルスタイルを上書きしている場合、新しいセレクターに移行する必要があります。

関連トピック:

Salesforce ヘルプ: 静的リソース
Salesforce ヘルプ: ページの <head> へのマークアップの追加によるエクスペリエンスビルダーサイトのカスタマイズ

CSS 上書きの移行

Spring '17 から Winter '19 の間に、いくつかのエクスペリエンスビルダーコンポーネントの CSS セレクターが更新されました。それ以降テンプレートを更新しておらず、サイトでカスタム CSS を使用してデフォルトのテンプレートとテーマパネルスタイルを上書きしている場合、新しいセレクターに移行する必要があります。

Salesforce ヘルプの「エクスペリエンスビルダーサイトのテンプレートの更新」を参照してください。

メモ:

- テンプレートの更新では、カスタマイズがサポートされるとは限らないため、カスタム CSS は慎重に使用してください。
- 現在、カスタム CSS は、すべてのサイトページで共有されています。ログインページでカスタム CSS を使用していた場合は、コピーしてから CSS エディターを閉じます。次に、ログイン以外のページに移動し、エディターを再び開いて、カスタム CSS を追加します。

ナビゲーションメニューの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

パネルコンテナの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

レコードバナーコンポーネントの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

レコード詳細コンポーネントの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

レコードレイアウトコンポーネントの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

レコードリストコンポーネントの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

レコード関連リストコンポーネントの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

関連記事コンポーネントの CSS 上書きの移行
カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

評価ランキング表コンポーネントの CSS 上書きの移行
カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

組み込みサービスサイドバーヘッダーコンポーネントの CSS 上書きの移行
カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

トピック別トレンド記事コンポーネントの CSS 上書きの移行
カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

関連トピック:

カスタム CSS でのテンプレート要素の上書き

ナビゲーションメニューの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

このトピックでは、ナビゲーションメニューのセレクターの変更点について説明します。

メモ:

- テンプレートの更新では、カスタマイズがサポートされるとは限らないため、カスタム CSS は慎重に使用してください。
- 現在、カスタム CSS は、すべてのサイトページで共有されています。ログインページでカスタム CSS を使用していた場合は、コピーしてから CSS エディターを閉じます。次に、ログイン以外のページに移動し、エディターを再び開いて、カスタム CSS を追加します。

完全なナビゲーションメニュー

以前のセレクター	新しいセレクター

`.forceCommunityNavigationMenu`
`.forceCommunityNavigationMenu #navigationMenu`
`.forceCommunityNavigationMenu .navigationMenu`
`.forceCommunityNavigationMenu .navigationMenuWrapper`
	`.comm-navigation`

モバイルメニューカーテン

以前のセレクター	新しいセレクター
`.forceCommunityNavigationMenu .navigationMenuWrapperCurtain`	`.comm-navigation nav.slds-is-fixed`

ホームメニュー項目

以前のセレクター	新しいセレクター
`.forceCommunityNavigationMenu .homeLink` `.forceCommunityNavigationMenu .homeButton`	`.comm-navigation .slds-list__item a[data-type="home"]`

ホームメニュー項目のリンク

以前のセレクター	新しいセレクター

`.forceCommunityNavigationMenu`
`.homeLink.forceCommunityNavigationMenu`
`.homeButton.comm-navigation .slds-list__item a[data-type="home"]`
	`.comm-navigation .comm-navigation__item a[data-type="home"]`

モバイルメニュートグルボタン

以前のセレクター	新しいセレクター
`.forceCommunityNavigationMenu .toggleNav`	`.siteforceServiceBody .cHeaderPanel .cAltToggleNav`

最上位メニュー項目

サブメニュートリガーが含まれます。

以前のセレクター	新しいセレクター
`.forceCommunityNavigationMenu .menuItem` `.forceCommunityGlobalNavigation .navigationMenuNode`	`.comm-navigation .comm-navigation__list > .slds-list__item`
`.forceCommunityNavigationMenu .menuItem`	`.comm-navigation .comm-navigation__list > .slds-list__item > .comm-navigation__item`

以前のセレクター	新しいセレクター
.forceCommunityGlobalNavigation .navigationMenuNode .comm-navigation .comm-navigation__list > .slds-list__item	

現在の最上位メニュー項目

以前のセレクター	新しいセレクター
.forceCommunityNavigationMenu .current .forceCommunityGlobalNavigation .menuItem.current	.comm-navigation .comm-navigation__list > .slds-list__item > .slds-is-active
.forceCommunityNavigationMenu .current .forceCommunityGlobalNavigation .menuItem.current .comm-navigation .comm-navigation__list > .slds-list__item > .slds-is-active	.comm-navigation .comm-navigation__list > .slds-list__item> .comm-navigation__item > .slds-is-active

最上位メニュー項目のリンク

以前のセレクター	新しいセレクター
.forceCommunityNavigationMenu .menuItemLink .forceCommunityNavigationMenu a.menuItemLink .forceCommunityNavigationMenu .menuItem .menuItemLink .forceCommunityNavigationMenu .menuItem a .forceCommunityNavigationMenu .menuItem a.menuItemLink	.comm-navigation .comm-navigation__list > .slds-list__item > a .comm-navigation .comm-navigation__list > .slds-list__item > button
.forceCommunityNavigationMenu .menuItemLink .forceCommunityNavigationMenu a.menuItemLink .forceCommunityNavigationMenu .menuItem .menuItemLink .forceCommunityNavigationMenu .menuItem a	.comm-navigation .comm-navigation__list > .slds-list__item> .comm-navigation__item > a .comm-navigation .comm-navigation__list > .slds-list__item> .comm-navigation__item > button

以前のセレクター	新しいセレクター

`.forceCommunityNavigationMenu .menuItem a.menuItemLink`
`.comm-navigation .comm-navigation__list > .slds-list__item > a`
`.comm-navigation .comm-navigation__list > .slds-list__item > button`	

サブメニュー項目

以前のセレクター	新しいセレクター
`.forceCommunityNavigationMenu .subMenuItem`	`.comm-navigation .slds-list_vertical.slds-is-nested .slds-list__item`
`.forceCommunityNavigationMenu .subMenuItem` `.comm-navigation .slds-list_vertical.slds-is-nested .slds-list__item`	`.comm-navigation .slds-list_vertical.slds-is-nested .comm-navigation__item`

現在の/有効なサブメニュー項目

以前のセレクター	新しいセレクター
`.forceCommunityNavigationMenu .subMenuItem.current`	`.comm-navigation .slds-list_vertical.slds-is-nested .slds-list__item .slds-is-active`
`.forceCommunityNavigationMenu .subMenuItem.current` `.comm-navigation .slds-list_vertical.slds-is-nested .slds-list__item .slds-is-active`	`.comm-navigation .slds-list_vertical.slds-is-nested .comm-navigation__item .slds-is-active`

サブメニュートリガーのリンク

以前のセレクター	新しいセレクター
`.forceCommunityNavigationMenu .triggerLink` `.forceCommunityNavigationMenu .triggerLabel`	`.comm-navigation .slds-list__item button:enabled`

以前のセレクター	新しいセレクター

`.forceCommunityNavigationMenu .triggerLink`
`..forceCommunityNavigationMenu .triggerLabel`
`.comm-navigation .slds-list__item button:enabled`
	`.comm-navigation .comm-navigation__item button:enabled`

サブメニュートリガーのリンクアイコン

以前のセレクター	新しいセレクター
`.forceCommunityNavigationMenu .triggerLink .forceIcon`	`.comm-navigation .slds-list__item button:enabled .slds-icon_container`
`.forceCommunityNavigationMenu .triggerLink .forceIcon` `.comm-navigation .slds-list__item button:enabled .slds-icon_container`	`.comm-navigation .comm-navigation__item button:enabled .slds-icon_container`

メニュー項目

最上位およびサブメニューの項目が含まれます。

以前のセレクター	新しいセレクター
`.forceCommunityNavigationMenu .navigationMenu li`	`.comm-navigation .slds-list__item`
`.forceCommunityNavigationMenu .navigationMenu li` `.comm-navigation .slds-list__item`	`.comm-navigation .comm-navigation__item`

メニュー項目のリンク

最上位およびサブメニューの項目が含まれます。

以前のセレクター	新しいセレクター
`.forceCommunityNavigationMenu a` `.forceCommunityNavigationMenu a.menuItemLink`	`.comm-navigation .slds-list__item a` `.comm-navigation .slds-list__item button`

以前のセレクター	新しいセレクター
.forceCommunityNavigationMenu a .forceCommunityNavigationMenu a.menuItemLink .comm-navigation .slds-list__item a .comm-navigation .slds-list__item button	.comm-navigation .comm-navigation__item a.comm-navigation .comm-navigation__item button

サブメニュー

以前のセレクター	新しいセレクター
.forceCommunityNavigationMenu .subMenu	.comm-navigation .slds-list_vertical.slds-is-nested

サブメニュー項目

以前のセレクター	新しいセレクター
.forceCommunityNavigationMenu .subMenuItem	.comm-navigation .slds-list_vertical.slds-is-nested .slds-list__item
.forceCommunityNavigationMenu .subMenuItem .comm-navigation .slds-list_vertical.slds-is-nested .slds-list__item	.comm-navigation .slds-list_vertical.slds-is-nested .comm-navigation__item

サブメニュー項目のリンク

以前のセレクター	新しいセレクター
.forceCommunityNavigationMenu .subMenuItem a .forceCommunityNavigationMenu .subMenu a	.comm-navigation .slds-list_vertical.slds-is-nested .slds-list__item a
.comm-navigation .slds-list_vertical.slds-is-nested .comm-navigation__item	.comm-navigation .slds-list_vertical.slds-is-nested .comm-navigation__item a

パネルコンテナの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

このトピックでは、セレクターの変更点について説明します。

メモ:

- テンプレートの更新では、カスタマイズがサポートされるとは限らないため、カスタム CSS は慎重に使用してください。
- 現在、カスタム CSS は、すべてのサイトページで共有されています。ログインページでカスタム CSS を使用していた場合は、コピーしてから CSS エディターを閉じます。次に、ログイン以外のページに移動し、エディターを再び開いて、カスタム CSS を追加します。

パネルコンテナ

以前のセレクター	新しいセレクター

`.uiPanelManager2`
`.onePanelManager`
`.siteforcePanelManager`
	`.comm-panels-container`

レコードバナーコンポーネントの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

このトピックでは、セレクターの変更点について説明します。

メモ:

- テンプレートの更新では、カスタマイズがサポートされるとは限らないため、カスタム CSS は慎重に使用してください。
- 現在、カスタム CSS は、すべてのサイトページで共有されています。ログインページでカスタム CSS を使用していた場合は、コピーしてから CSS エディターを閉じます。次に、ログイン以外のページに移動し、エディターを再び開いて、カスタム CSS を追加します。

詳細項目の表示ラベル

以前のセレクター	新しいセレクター
`.forceCommunityRecordHeadline` `.slds-text-heading-label-normal`	`.forceCommunityRecordHeadline` `.slds-form-element__label`

詳細項目の値

以前のセレクター	新しいセレクター
`.forceCommunityRecordHeadline .slds-text-body--regular`	`.forceCommunityRecordHeadline .slds-form-element__static`

レコード詳細コンポーネントの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。このトピックでは、セレクターの変更点について説明します。

メモ:

- テンプレートの更新では、カスタマイズがサポートされるとは限らないため、カスタム CSS は慎重に使用してください。
- 現在、カスタム CSS は、すべてのサイトページで共有されています。ログインページでカスタム CSS を使用していた場合は、コピーしてから CSS エディターを閉じます。次に、ログイン以外のページに移動し、エディターを再び開いて、カスタム CSS を追加します。

フォーム要素の区切り文字

以前のセレクター	新しいセレクター
`.forceCommunityRecordDetail .slds-form-element__control .slds-form-element_separator`	`.forceCommunityRecordDetail .slds-form-element_separator`

レコードレイアウトコンポーネントの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。このトピックでは、セレクターの変更点について説明します。

メモ:

- テンプレートの更新では、カスタマイズがサポートされるとは限らないため、カスタム CSS は慎重に使用してください。
- 現在、カスタム CSS は、すべてのサイトページで共有されています。ログインページでカスタム CSS を使用していた場合は、コピーしてから CSS エディターを閉じます。次に、ログイン以外のページに移動し、エディターを再び開いて、カスタム CSS を追加します。

完全なレコードレイアウト

以前のセレクター	新しいセレクター
`.forceRecordLayout`	`force-record-layout2`
`.forceRecordLayout.forcePageBlock`	`force-record-layout-block`

セクション

以前のセレクター	新しいセレクター
`.forceRecordLayout .full.forcePageBlockSectionView`	`force-record-layout2 force-record-layout-section`
`.forceRecordLayout .full.forcePageBlockSection`	
`.forceRecordLayout .full.forcePageBlockSectionView.forcePageBlockSection`	
`.forceRecordLayout .forcePageBlockSectionView`	
`.forceRecordLayout .forcePageBlockSection`	
`.forceRecordLayout .forcePageBlockSectionView.forcePageBlockSection`	

セクションタイトル

以前のセレクター	新しいセレクター
`.forceRecordLayout .full.forcePageBlockSection h3`	`force-record-layout2 force-record-layout-section h3`

セクション行

以前のセレクター	新しいセレクター
`.forceRecordLayout .full.forcePageBlockSectionRow`	`force-record-layout2 force-record-layout-row`
`.forceRecordLayout .forcePageBlockSectionRow`	

セクション項目

以前のセレクター	新しいセレクター

`.forceRecordLayout .full.forcePageBlockItem.forcePageBlockItemView`
`.forceRecordLayout .full.forcePageBlockItem`
`.forceRecordLayout .full.forcePageBlockItemView`
`.forceRecordLayout .forcePageBlockItem`
`.forceRecordLayout .forcePageBlockItemView`
`.forceRecordLayout .forcePageBlockItem.forcePageBlockItemView`
	`force-record-layout2`
	`force-record-layout-item`

セクション項目表示ラベル

以前のセレクター	新しいセレクター
`.forceRecordLayout .slds-form-element__label`	`force-record-layout2 .slds-form-element__label`

セクション項目値

以前のセレクター	新しいセレクター
`.forceRecordLayout .itemBody`	`force-record-layout2 .slds-form-element__static`

セクション項目値リンク

以前のセレクター	新しいセレクター
`.forceRecordLayout a`	`force-record-layout2 a`

レコードリストコンポーネントの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。このトピックでは、セレクターの変更点について説明します。

メモ:

- テンプレートの更新では、カスタマイズがサポートされるとは限らないため、カスタム CSS は慎重に使用してください。

- 現在、カスタム CSS は、すべてのサイトページで共有されています。ログインページでカスタム CSS を使用していた場合は、コピーしてから CSS エディターを閉じます。次に、ログイン以外のページに移動し、エディターを再び開いて、カスタム CSS を追加します。

リストビューボタンバー

以前のセレクター	新しいセレクター
`.forceListViewManagerHeader .forceListViewManagerButtonBar`	`.forceListViewManagerHeader force-list-view-manager-button-bar`

リストビューボタンバーのボタン

以前のセレクター	新しいセレクター
`.forceListViewManagerHeader .forceListViewManagerButtonBar button`	`.forceListViewManagerHeader force-list-view-manager-button-bar button`

リストビュー状況情報

以前のセレクター	新しいセレクター
`.forceListViewManagerHeader .test-listViewStatusInfo`	`.forceListViewManagerHeader force-list-view-manager-status-info`

ピッカートリガーのリンク

以前のセレクター	新しいセレクター
`.forceListViewManagerHeader .triggerLink`	`.forceListViewManagerHeader .triggerLink .slds-button`

レコード関連リストコンポーネントの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

このトピックでは、セレクターの変更点について説明します。

メモ:

- テンプレートの更新では、カスタマイズがサポートされるとは限らないため、カスタム CSS は慎重に使用してください。
- 現在、カスタム CSS は、すべてのサイトページで共有されています。ログインページでカスタム CSS を使用していた場合は、コピーしてから CSS エディターを閉じます。次に、ログイン以外のページに移動し、エディターを再び開いて、カスタム CSS を追加します。

詳細項目の表示ラベル

以前のセレクター	新しいセレクター
.forceCommunityRelatedRecords .slds-card__header-link .slds-text-heading--small	.forceCommunityRelatedRecords .slds-card__header-title

関連記事コンポーネントの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

このトピックでは、セレクターの変更点について説明します。

メモ:

- テンプレートの更新では、カスタマイズがサポートされるとは限らないため、カスタム CSS は慎重に使用してください。
- 現在、カスタム CSS は、すべてのサイトページで共有されています。ログインページでカスタム CSS を使用していた場合は、コピーしてから CSS エディターを閉じます。次に、ログイン以外のページに移動し、エディターを再び開いて、カスタム CSS を追加します。

完全な関連記事

以前のセレクター	新しいセレクター
.selfServiceSimilarArticles .base-items .uiAbstractList .selfServiceBaseSimpleItems	.comm-related-articles

コンポーネントタイトル

以前のセレクター	新しいセレクター
.selfServiceSimilarArticles h2 .selfServiceSimilarArticles .base-items-header	.comm-related-articles h2

記事リスト

以前のセレクター	新しいセレクター
.selfServiceSimilarArticles ul	.comm-related-articles ul

以前のセレクター	新しいセレクター
`.selfServiceSimilarArticles .base-items`	

記事リスト項目

以前のセレクター	新しいセレクター
`.selfServiceSimilarArticles .base-simple-item`	`.comm-related-articles li`
`.selfServiceSimilarArticles .base-simple-item` `.comm-related-articles li`	`.comm-related-articles` `.comm-related-articles__item`

記事リンク

以前のセレクター	新しいセレクター
`.selfServiceSimilarArticles .item-title` `.selfServiceSimilarArticles .item-title-link`	`.comm-related-articles li a`

`.selfServiceSimilarArticles .item-title`
`.selfServiceSimilarArticles .item-title-link`
`.comm-related-articles li a`
	`.comm-related-articles`
	`.comm-related-articles__item a`

評価ランキング表コンポーネントの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

このトピックでは、セレクターの変更点について説明します。

メモ:

- テンプレートの更新では、カスタマイズがサポートされるとは限らないため、カスタム CSS は慎重に使用してください。
- 現在、カスタム CSS は、すべてのサイトページで共有されています。ログインページでカスタム CSS を使用していた場合は、コピーしてから CSS エディターを閉じます。次に、ログイン以外のページに移動し、エディターを再び開いて、カスタム CSS を追加します。

完全な評価ランキング表

以前のセレクター	新しいセレクター
.forceCommunityReputationLeaderboard.leaderboard	.comm-leaderboard

評価ランキング表の行

以前のセレクター	新しいセレクター
.forceCommunityReputationLeaderboardRow	.comm-leaderboard li
.forceCommunityReputationLeaderboardRow .comm-leaderboard li	.comm-leaderboard .comm-leaderboard__item

ユーザー情報列

以前のセレクター	新しいセレクター
.forceCommunityReputationLeaderboard .pointsAndLevels .forceCommunityReputationLeaderboard .userInfoCol	.comm-leaderboard .slds-media__body

評価ポイント列

以前のセレクター	新しいセレクター
.forceCommunityReputationLeaderboard .reputationCol	.comm-leaderboard__points-column

タイトル

以前のセレクター	新しいセレクター
.forceCommunityReputationLeaderboard .title	.comm-leaderboard h2

ユーザーレベル名

以前のセレクター	新しいセレクター
.forceCommunityReputationLeaderboard .reputationLevelName	.comm-leaderboard__level-name

ユーザー評価レベル画像

以前のセレクター	新しいセレクター
`.forceCommunityReputationLeaderboard .reputationLevelImage`	`.comm-leaderboard .slds-media__body .slds-icon_small`

ユーザーのユーザー名

以前のセレクター	新しいセレクター
`.forceCommunityReputationLeaderboard .userName`	`.comm-leaderboard__user-name`

ユーザーのユーザー画像

以前のセレクター	新しいセレクター
`.forceCommunityReputationLeaderboard .userPhoto`	`.comm-leaderboard .slds-media__figure`

評価レベルのユーザーポイント

以前のセレクター	新しいセレクター
`.forceCommunityReputationLeaderboard reputationPointsNumber。`	`.comm-leaderboard__points`

評価レベルのユーザーポイント語

以前のセレクター	新しいセレクター
`.forceCommunityReputationLeaderboard .reputationPointsWord。`	`.comm-leaderboard__points-word`

組み込みサービスサイドバーヘッダーコンポーネントの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。このトピックでは、セレクターの変更点について説明します。

メモ:

- テンプレートの更新では、カスタマイズがサポートされるとは限らないため、カスタム CSS は慎重に使用してください。

- 現在、カスタム CSS は、すべてのサイトページで共有されています。ログインページでカスタム CSS を使用していた場合は、コピーしてから CSS エディターを閉じます。次に、ログイン以外のページに移動し、エディターを再び開いて、カスタム CSS を追加します。

ヘッダー背景色

以前のセレクター	新しいセレクター
`.sidebarHeader { background-color: #aaa; }`	`.embeddedServiceSidebar.sidebarHeader { background-color:#aaa; }`

トピック別トレンド記事コンポーネントの CSS 上書きの移行

カスタム CSS 上書きを継続して使用する場合は、テンプレートの更新後に CSS 上書きを移行してください。

このトピックでは、セレクターの変更点について説明します。

メモ:

- テンプレートの更新では、カスタマイズがサポートされるとは限らないため、カスタム CSS は慎重に使用してください。
- 現在、カスタム CSS は、すべてのサイトページで共有されています。ログインページでカスタム CSS を使用していた場合は、コピーしてから CSS エディターを閉じます。次に、ログイン以外のページに移動し、エディターを再び開いて、カスタム CSS を追加します。

コミュニティのトレンド記事

以前のセレクター	新しいセレクター

`.selfServiceTopicTrendingArticles`
`.base-items`
`.uiAbstractList`
`.selfServiceBaseSimpleItems`
	`.comm-topic-trending-articles`

コンポーネントタイトル

以前のセレクター	新しいセレクター
`.selfServiceTopicTrendingArticles h2` `.selfServiceTopicTrendingArticles .base-items-header`	`.comm-topic-trending-articles h2`

記事リスト

以前のセレクター	新しいセレクター
`.selfServiceTopicTrendingArticles ul` `.selfServiceTopicTrendingArticles .base-items`	`.comm-topic-trending-articles ul`

記事リスト項目

以前のセレクター	新しいセレクター
`.selfServiceTopicTrendingArticles .base-simple-item`	`.comm-topic-trending-articles li`
`.selfServiceTopicTrendingArticles .base-simple-item` `.comm-topic-trending-articles li`	`.comm-topic-trending-articles .comm-topic-trending-articles__item`

記事リンク

以前のセレクター	新しいセレクター
`.selfServiceTopicTrendingArticles .item-title-link` `.selfServiceTopicTrendingArticles a`	`.comm-topic-trending-articles li a`

`.selfServiceTopicTrendingArticles .item-title-link`
`.selfServiceTopicTrendingArticles a`
`.comm-topic-trending-articles li a`
	`.comm-topic-trending-articles .comm-topic-trending-articles__item a`

エクスペリエンスビルダーサイトでのカスタムフォントの使用

カスタムフォントを静的リソースとしてアップロードし、サイト全体のプライマリフォントおよびヘッダーフォントとして使用します。複数のフォントファイルをアップロードする場合は、.zipファイルを使用します。

1. [設定] で、[クイック検索] ボックスに「*静的リソース*」と入力し、[静的リソース] を選択します。
2. [新規] をクリックし、ファイルをアップロードして、静的リソースに名前を付けます。リソース名を書き留めます。

   サイトに公開ページがある場合は、[キャッシュコントロール] 設定で [公開] を選択します。フォントリソースを公開しないと、ページでは代わりにブラウザーのデフォルトフォントが使用されます。

3. エクスペリエンスビルダーで [テーマ] > > [CSS を編集] をクリックして CSS エディターを開きます。
4. `@font-face` CSS ルールを使用して、アップロードしたフォントを参照します。
   次に例を示します。

```
@font-face {
    font-family: 'myFirstFont';
    src: url('/myPartnerSite/s/sfsites/c/resource/MyFonts/bold/myFirstFont.woff')
format('woff');
}
```

   メモ: `format()` は、URL で参照されているフォントの形式を記述する省略可能なヒントです。

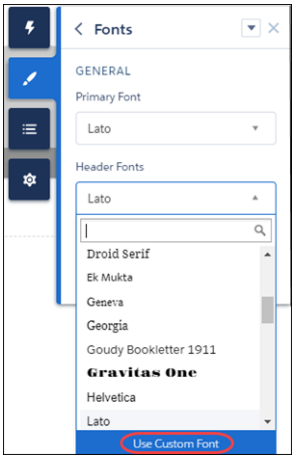
   - 1 つのフォントファイルを参照する場合の構文は */path_prefix*`/s/sfsites/c/resource/`*resource_name* で、*path_prefix* はサイトの作成時に追加した URL 値 (例: `myPartnerSite`) です。

     たとえば、`myFirstFont.woff` というファイルをアップロードし、リソースの名前が MyFonts であれば、URL は `/myPartnerSite/s/sfsites/c/resource/MyFonts` になります。

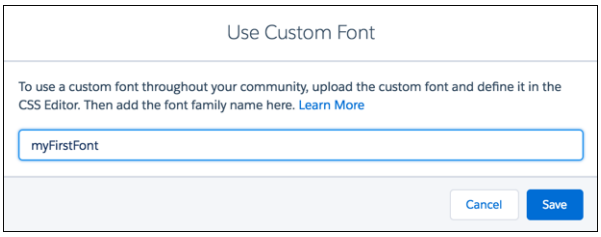
   - .zip ファイル内のファイルを参照するには、フォルダー構造を含めて .zip ファイル名を省略します。構文は */path_prefix*`/s/sfsites/c/resource/`*resource_name*`/`*font_folder*`/`*font_file* です。

     たとえば、`bold/myFirstFont.woff` を含む `fonts.zip` をアップロードし、リソースの名前が MyFonts であれば、URL は `/myPartnerSite/s/sfsites/c/resource/MyFonts/bold/myFirstFont.woff` になります。

5. [テーマ] パネルで [フォント] を選択し、[プライマリフォント] または [ヘッダーフォント] ドロップダウンリストを選択して、[カスタムフォントを使用] をクリックします。



6. CSS エディターに入力したフォントファミリー名 (例: `myFirstFont`) を追加して、変更内容を保存します。



テンプレートのテーマレイアウトのカスタマイズ

テンプレートテーマに独自のスタンプを配置して外観を変換するには、カスタムテーマレイアウトコンポーネントを作成します。テンプレートの構造上のレイアウト (ヘッダーやフッターなど) をカスタマイズして、デフォルトのスタイルを上書きできます。

テーマレイアウトコンポーネントは、サイトのテンプレートページの最上位のテンプレート (1) です。テーマレイアウトコンポーネントは、テーマレイアウトを使用してページに整理および適用されます。テーマレイアウトコンポーネントには、共通のヘッダーとフッター (2) が含まれ、多くの場合、ナビゲーション、検索、およびユーザープロファイルメニューが含まれます。一方、コンテンツレイアウト (3) では、ページのコンテンツ範囲を定義します。2 列のコンテンツレイアウトを次に示します。

関連トピック:

Trailhead: Build a Custom Theme Layout Component for Experience Builder Sites (エクスペリエンスビルダーサイト用のカスタムテーマレイアウトコンポーネントの構築)

カスタムテーマレイアウトのしくみ

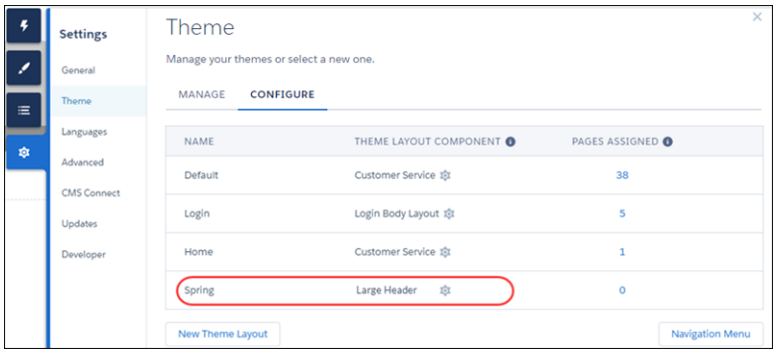
テーマレイアウトの仕組みを理解するため、エクスペリエンスビルダーの視点から物事を見てみましょう。エクスペリエンスビルダーでは、テーマレイアウトとテーマレイアウトコンポーネントを組み合わせることで、サイトの各ページの外観と構造をより詳細に制御できます。会社のブランドとスタイルに合わせてレイアウトのヘッダーとフッターをカスタマイズし、テーマプロパティを設定するか、カスタム検索バーとユーザープロファイルメニューを使用できます。その後テーマレイアウトを使用して、1 か所から、テーマレイアウトコンポーネントを個々のページに適用したり、すばやくレイアウトを変更したりすることができます。

テーマレイアウトは、サイト内で同じテーマレイアウトコンポーネントを共有するページを分類します。既存のテーマレイアウトに、テーマレイアウトコンポーネントを割り当てることができます。その後、ページのプロパティでテーマレイアウト (つまりテーマレイアウトコンポーネント) を適用します。

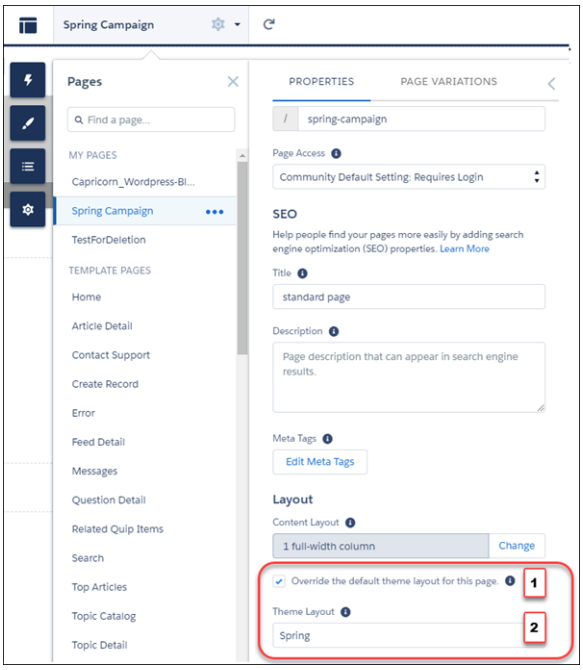
たとえば、[カスタマーサービス]テンプレートには次のテーマレイアウトとコンポーネントが含まれていますが、必要に応じてカスタムコンポーネントを作成したり、レイアウトを切り替えたりすることができます。

- [デフォルト]では、ログインページ以外のすべてのページに[カスタマーサービス]テーマレイアウトが適用されます。
- [ログイン]では、ログインページに[ログインのボディレイアウト]テーマレイアウトコンポーネントが適用されます。

例: たとえば、近日予定されている春のキャンペーンのために 3 つのページを作成するとします。`forceCommunity:themeLayout` インターフェースを使用して、開発者コンソールでカスタム [Large Header (大きなヘッダー)] テーマレイアウトを作成します。[設定] > [テーマ] 領域で、キャンペーンページを分類するために「Spring (春)」というカスタムテーマレイアウトを追加して、それに [Large Header (大きなヘッダー)] レイアウトコンポーネントを割り当てます。



次に、[Spring (春)] テーマレイアウトを各ページのプロパティで適用します。各ページに [Large Header (大きなヘッダー)] レイアウトがすぐに適用されます。[このページのデフォルトのテーマレイアウトを上書きします。](1)を選択してテーマレイアウトを表示します。使用可能なオプションから新しいレイアウト (2) を選択します。

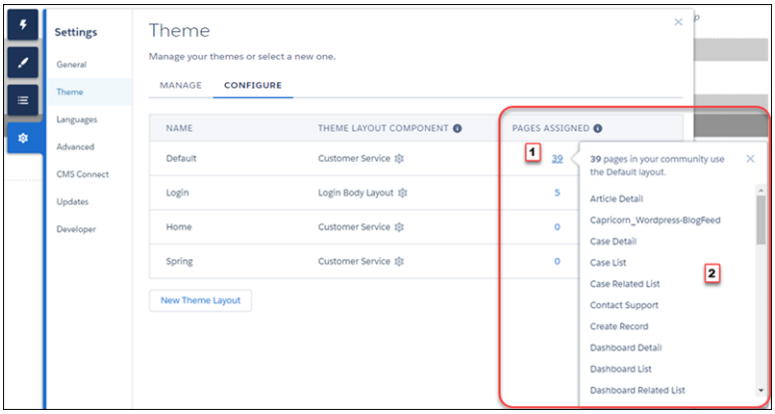


うまくいっているように見えましたが、マーケティングの統括責任者がヘッダーが場所を取りすぎると判断しました。テーマレイアウトを変更するために各ページのプロパティを更新する必要がないため、簡単に修正できます。[テーマ]領域で1回クリックするだけで、[Spring(春)]を[Small Header(小さなヘッダー)]レイアウトに切り替えて、3つのページすべてをすぐに更新できます。

例: ここで、[Small Header(小さなヘッダー)]レイアウトに2つのカスタムプロパティ、[Blue Background(青い背景)]と[Small Logo(小さいロゴ)]が含まれているとします。これらのプロパティは有効になっていてすべてのキャンペーンページに適用されています。ただし、1つのページには、小さいロゴプロパティのみを適用する必要があります。

この場合、「Spring B(春B)」というテーマレイアウトを作成し、それに[Small Header(小さなヘッダー)]レイアウトコンポーネントを割り当て、小さいロゴを有効にすることができます。その後、[Spring B(春B)]テーマをページに適用します。

テーマレイアウトにどのページが関連付けられているか不明な場合は、どうしたらよいでしょう。



テーマレイアウトに関連付けられているページ数とページを 1 クリックで瞬時に把握できます。[設定] > [テーマ] から、テーマレイアウトの列に表示されている [割り当てられたページ] の合計数 (1) をクリックします。この値をクリックすると、そのテーマレイアウトに関連付けられているページのリスト (2) が開きます。

テーマレイアウトを使用すると、必要なだけ詳細な制御を維持しながら、同じテーマレイアウトコンポーネントを異なる方法で簡単に再利用することができます。

関連トピック:

Trailhead: Build a Custom Theme Layout Component for Experience Builder Sites (エクスペリエンスビルダーサイト用のカスタムテーマレイアウトコンポーネントの構築)

カスタムテーマレイアウトコンポーネントの設定

開発者コンソールでカスタムテーマレイアウトコンポーネントを作成して、[カスタマーサービス] テンプレートのページの外観および全体的な構造を変換する方法を見てみましょう。

1.インターフェースのテーマレイアウトコンポーネントへの追加

テーマレイアウトコンポーネントをエクスペリエンスビルダーの [設定] > [テーマ] 領域に表示するには、テーマレイアウトコンポーネントに `forceCommunity:themeLayout` インターフェースを実装する必要があります。

コードで `{!v.body}` を明示的に宣言して、テーマレイアウトにコンテンツレイアウトが含まれていることを確認します。ページのコンテンツをテーマレイアウト内に表示したい場合は必ず、`{!v.body}` を追加します。

`Aura.Component[]` として宣言された属性を追加して、ページのコンポーネントを含む領域をテーマレイアウトに含めます。コンポーネントをマークアップ内の領域に追加することも、ユーザーがコンポーネントをドラッグアンドドロップできるように領域をオープンにしておくこともできます。`Aura.Component[]` として宣言された属性や、マークアップに含まれる属性はすべて、ユーザーがコンポーネントを追加できるテーマレイアウトのオープン領域として表示されます。次に例を示します。

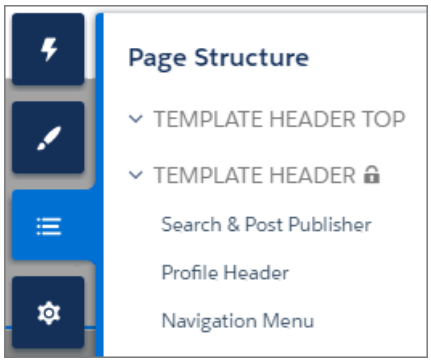
```
<aura:component implements="forceCommunity:themeLayout">
<aura:attribute name="myRegion" type="Aura.Component[]"/>

{!v.body}
```

```
</aura:component>
```

カスタマーサービスでは、テンプレートヘッダーが次のロック済みの領域で構成されます。

- `search` ([検索パブリッシャー] コンポーネントを含む)
- `profileMenu` ([プロファイルヘッダー] コンポーネントを含む)
- `navBar` ([ナビゲーションメニュー] コンポーネントを含む)



[テンプレートのヘッダー] 領域で既存のコンポーネントを再利用するカスタムテーマレイアウトを作成するには、必要に応じて、`search`、`profileMenu`、`navBar` などを属性名の値として宣言します。次に例を示します。

```
<aura:attribute name="navBar" type="Aura.Component[]" required="false" />
```

ヒント: 交換可能なカスタムプロファイルメニューまたは検索コンポーネントを作成する場合、属性名の値として `search` または `profileMenu` を宣言すると、エクスペリエンスビルダーでテーマレイアウトを使用するときにユーザーもカスタムコンポーネントを選択できます。

マークアップに領域を追加して、テーマレイアウト本体のどこに表示するかを定義します。

以下は、シンプルなテーマレイアウトのサンプルコードです。

```
<aura:component implements="forceCommunity:themeLayout" access="global" description="Sample
 Custom Theme Layout">
    <aura:attribute name="search" type="Aura.Component[]" required="false"/>
    <aura:attribute name="profileMenu" type="Aura.Component[]" required="false"/>
    <aura:attribute name="navBar" type="Aura.Component[]" required="false"/>
    <aura:attribute name="newHeader" type="Aura.Component[]" required="false"/>
    <div>
        <div class="searchRegion">
            {!v.search}
        </div>
        <div class="profileMenuRegion">
            {!v.profileMenu}
        </div>
        <div class="navigation">
            {!v.navBar}
        </div>
        <div class="newHeader">
            {!v.newHeader}
```

```
        </div>
        <div class="mainContentArea">
            {!v.body}
        </div>
    </div>
</aura:component>
```

2.テーマプロパティを含めるためのデザインリソースの追加

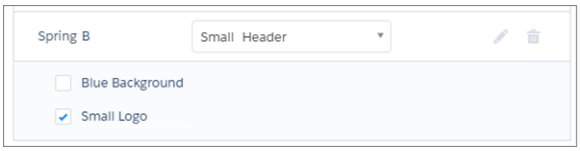
エクスペリエンスビルダーでテーマレイアウトプロパティを公開するには、デザインリソースをバンドルに追加します。

まず、コンポーネントでプロパティを実装します。

```
<aura:component implements="forceCommunity:themeLayout" access="global" description="Small
 Header">
    <aura:attribute name="blueBackground" type="Boolean" default="false"/>
    <aura:attribute name="smallLogo" type="Boolean" default="false" />
    ...
```

設計リソースでテーマプロパティを定義して、UI でプロパティを表示します。次の例では、[Small Header (小さなヘッダー)] テーマレイアウトに表示ラベルと 2 つのチェックボックスを追加しています。

```
<design:component label="Small Header">
    <design:attribute name="blueBackground" label="Blue Background"/>
    <design:attribute name="smallLogo" label="Small Logo"/>
</design:component>
```



3.CSS リソースの追加による重複の問題の回避

必要に応じて、できれば標準デザイントークンを使用して、CSS リソースをバンドルに追加してテーマレイアウトのスタイルを設定します。

ダイアログボックスやフロート表示など、位置付けられた要素の重複の問題を回避する手順は、次のとおりです。

- CSS スタイルを適用します。

  ```
  .THIS {
      position: relative;
      z-index: 1;
  }
  ```

- 要素をカスタムテーマレイアウトの `div` タグでラップします。

  ```
  <div class="mainContentArea">
      {!v.body}
  </div>
  ```

メモ: テーマレイアウトは、レイアウト内のすべてのスタイルを制御し、領域やコンポーネントに影などのスタイルを追加します。カスタムテーマレイアウトの場合、デフォルトで SLDS が読み込まれます。

関連トピック:

Trailhead: Build a Custom Theme Layout Component for Experience Builder Sites (エクスペリエンスビルダーサイト用のカスタムテーマレイアウトコンポーネントの構築)

数式を使用した Aura サイトへの動的データの追加

式を使用することで、コンポーネントの属性に渡されるプロパティ値などの情報にアクセスできます。

式はリテラル値、変数、サブ式、演算子などで構成され、1 つの値に解決されます。メソッドコールは式に使用できません。

式の構文は、`{!expression}` です。ここで `expression` は式を表すプレースホルダーです。

次の式は、認証されたユーザー情報、データカテゴリに関連付けられた画像、レコード情報をサイトページに表示する場合に使用します。

式	表示
`{!CurrentUser.name}`	ユーザー詳細ページに表示されるユーザーの姓と名を合わせたもの。
`{!CurrentUser.firstName}`	ユーザー編集ページに表示されるユーザーの名。
`{!CurrentUser.lastName}`	ユーザー編集ページに表示されるユーザーの姓。
`{!CurrentUser.userName}`	ユーザーのログインを定義する管理項目。
`{!CurrentUser.id}`	ユーザーの Salesforce ID。
`{!CurrentUser.email}`	ユーザーのメールアドレス。
`{!CurrentUser.communityNickname}`	サイトでユーザーの識別に使用される名前。
`{!CurrentUser.accountId}`	ユーザーに関連付けられている取引先ID。この式は、パートナーユーザーや顧客ユーザーの有効な取引先 ID を表示します。その他の場合、「000000000000000」を表示します。
`{!CurrentUser.effectiveAccountId}`	有効なアカウントに関連付けられている取引先ID。この式は、パートナーユーザーや顧客ユーザーの有効な取引先 ID を表示します。その他の場合、「000000000000000」を表示します。
`{!Global.PathPrefix}/{!DataCategory.Name}.jpg`	検索コンポーネントのデータカテゴリに関連付けられている画像。
`{!Global.PathPrefix}/<Name of the Subfolder>/{!DataCategory.Name}.jpg`	検索コンポーネントのサブフォルダー内のデータカテゴリに関連付けられている画像。

式	表示
`{!recordId}`	オブジェクトページの 15 桁のレコード ID。
`{!term}`	Aura ベースの標準の検索ページで、HTML エンコードされた検索語を返す式。

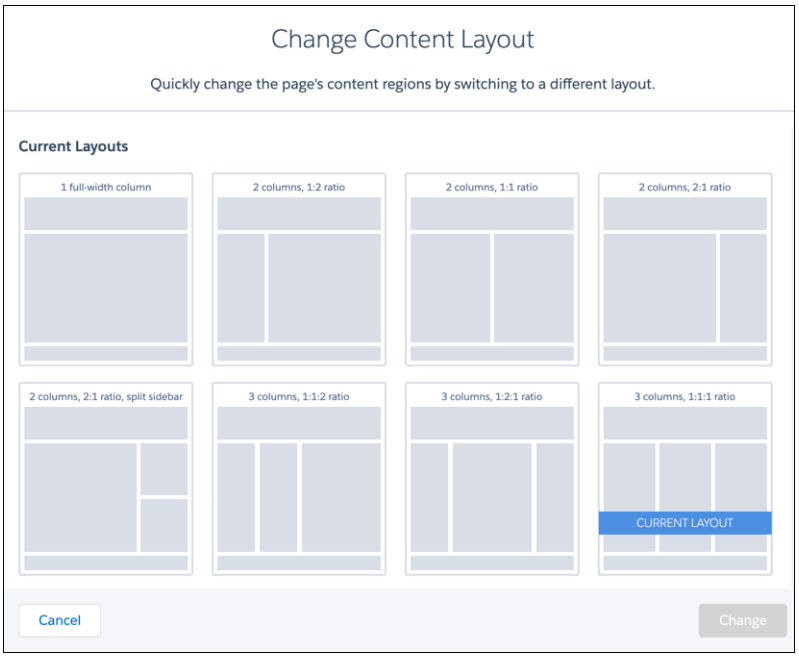
関連トピック:

LWR Sites for Experience Cloud (Experience Cloud の LWR サイト): Use Expressions in LWR Sites (LWR サイトでの式の使用)

エクスペリエンスビルダーのカスタムコンテンツレイアウトコンポーネントの作成

エクスペリエンスビルダーには、比率 2:1 の 2 列レイアウトなど、ページのコンテンツ領域を定義するレイアウトがいくつか用意され、すぐに使用できます。ただし、サイト用にカスタマイズされたレイアウトが必要な場合は、カスタムコンテンツレイアウトコンポーネントを作成し、それをエクスペリエンスビルダーで新規ページを作成するときに使用します。サイトテンプレートに付属するデフォルトページのコンテンツレイアウトを更新することもできます。

開発者コンソールでカスタムコンテンツレイアウトコンポーネントを作成すると、そのコンポーネントはエクスペリエンスビルダーの [新規ページ] および [レイアウトを変更] ダイアログボックスに表示されます。



1.新規インターフェースをコンテンツレイアウトコンポーネントに追加する

エクスペリエンスビルダーの[新規ページ]および[レイアウトを変更]ダイアログボックスに表示するには、コンテンツレイアウトコンポーネントに `forceCommunity:layout` インターフェースを実装する必要があります。

シンプルな2列のコンテンツレイアウトのサンプルコードを次に示します。

```
<aura:component implements="forceCommunity:layout" description="Custom Content Layout"
access="global">
    <aura:attribute name="column1" type="Aura.Component[]" required="false"></aura:attribute>

    <aura:attribute name="column2" type="Aura.Component[]" required="false"></aura:attribute>


    <div class="container">
        <div class="contentPanel">
            <div class="left">
                {!v.column1}
            </div>
            <div class="right">
                {!v.column2}
            </div>
        </div>
    </div>
</aura:component>
```

メモ: コンポーネントなどのリソースを `access="global"` としてマークし、リソースを自分の組織外で使用できるようにします。たとえば、インストール済みパッケージで、または他の組織のLightningアプリケーションビルダーユーザーまたはエクスペリエンスビルダーユーザーが、コンポーネントを使用できるようにする場合などです。

また、`access="global"` とマークされているコンポーネント、イベント、インターフェースのドキュメントを作成できます。このドキュメントは、パッケージを使用またはインストールする組織のコンポーネントライブラリに自動的に表示されます。

2.CSS リソースをコンポーネントバンドルに追加する

次に、必要に応じてCSSリソースを追加してコンテンツレイアウトのスタイルを設定します。

シンプルな2列のコンテンツレイアウトのサンプルCSSを次に示します。

```
.THIS .contentPanel:before,
.THIS .contentPanel:after {
    content: " ";
    display: table;
}
.THIS .contentPanel:after {
    clear: both;
}
```

```
.THIS .left {
    float: left;
    width: 50%;
}
.THIS .right {
    float: right;
    width: 50%;
}
```

CSS リソースの名前は *componentName*.css にする必要があります。

3.省略可能: SVG リソースをコンポーネントバンドルに追加する

SVG リソースをコンポーネントバンドルに追加して、エクスペリエンスビルダーに表示されるときのコンテンツレイアウトコンポーネントのカスタムアイコンを定義できます。

エクスペリエンスビルダーのコンテンツレイアウトコンポーネントで推奨される画像サイズは、170 x 170 ピクセルです。ただし、画像のサイズが異なる場合、エクスペリエンスビルダーが適合するように画像を拡大縮小します。

SVG リソースの名前は *componentName*.svg にする必要があります。

関連トピック:

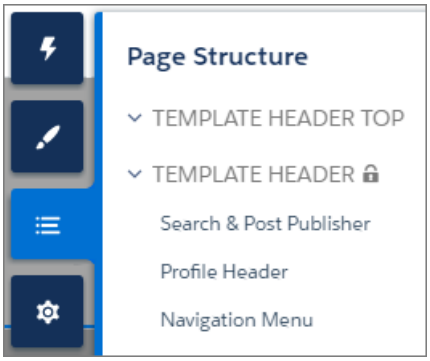
Salesforce ヘルプ: エクスペリエンスビルダーでのコンテンツレイアウトの変更

交換可能な検索およびプロファイルメニューコンポーネントの設定

エクスペリエンスビルダーでカスタムコンポーネントを作成し、テンプレートの標準の[プロファイルヘッダー]コンポーネントや [検索パブリッシャーと投稿パブリッシャー] コンポーネントを置き換えます。

たとえば、[カスタマーサービス] では、テンプレートヘッダーが次のロック済みの領域で構成されます。

- search ([検索パブリッシャー] コンポーネントを含む)
- profileMenu ([プロファイルヘッダー] コンポーネントを含む)
- navBar ([ナビゲーションメニュー] コンポーネントを含む)



これらの指定領域名により、以下を簡単に行うことができます。

- デフォルトのテーマレイアウトコンポーネントまたはカスタムテーマレイアウトコンポーネントで、検索コンポーネントとプロファイルコンポーネントを交換する。
- 既存のカスタマイズ (選択中の検索コンポーネントなど) を維持したまま、テーマレイアウトコンポーネントを交換する。

コンポーネントが正しいインターフェース (このケースでは `forceCommunity:searchInterface` または `forceCommunity:profileMenuInterface`) を実装すると、これらの領域の候補として識別されます。そのため、これらのインターフェースはテーマレイアウトコンポーネント (デフォルトの [カスタマーサービス] テーマレイアウトコンポーネントなど) に交換可能なコンポーネントとして表示され、`search` または `profileMenu` が属性名値として宣言されます。

```
<aura:attribute name="search" type="Aura.Component[]" required="false" />
```

forceCommunity:profileMenuInterface

`forceCommunity:profileMenuInterface` インターフェースを Aura コンポーネントに追加して、テンプレートのカスタムプロファイルメニューコンポーネントとして使用できるようにします。作成したカスタムプロファイルメニューコンポーネントは、システム管理者がエクスペリエンスビルダーの [設定] > [テーマ] で選択して、テンプレートの標準 [プロファイルヘッダー] コンポーネントと交換できます。

シンプルなプロファイルメニューコンポーネントのコードを示します。

```
<aura:component implements="forceCommunity:profileMenuInterface" access="global">
    <aura:attribute name="options" type="String[]" default="Option 1, Option 2"/>
    <ui:menu >
        <ui:menuTriggerLink aura:id="trigger" label="Profile Menu"/>
        <ui:menuList class="actionMenu" aura:id="actionMenu">
            <aura:iteration items="{!v.options}" var="itemLabel">
                <ui:actionMenuItem label="{!itemLabel}" click="{!c.handleClick}"/>
            </aura:iteration>
        </ui:menuList>
    </ui:menu>
</aura:component>
```

forceCommunity:searchInterface

`forceCommunity:searchInterface` インターフェースを Aura コンポーネントに追加して、テンプレートのカスタム検索コンポーネントとして使用できるようにします。作成したカスタム検索コンポーネントは、システム管理者がエクスペリエンスビルダーの [設定] > [テーマ] で選択して、テンプレートの標準 [検索パブリッシャーと投稿パブリッシャー] コンポーネントと交換できます。

シンプルな検索コンポーネントのコードを示します。

```
<aura:component implements="forceCommunity:searchInterface" access="global">
    <div class="search">
        <div class="search-wrapper">
            <form class="search-form">
                <div class="search-input-wrapper">
                    <input class="search-input" type="text" placeholder="My Search"/>
```

```
                </div>
                <input type="hidden" name="language" value="en" />
            </form>
        </div>
    </div>
</aura:component>
```

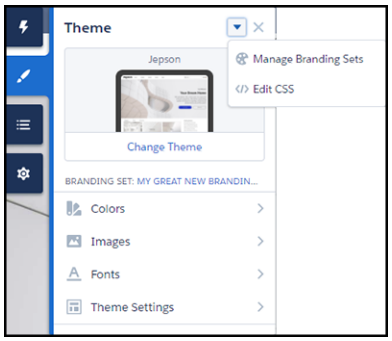
関連トピック:

Trailhead: Build a Custom Search Component (カスタム検索コンポーネントの作成)

エクスペリエンスビルダーサイトの標準デザイントークン

Salesforceでは、コンポーネントスタイルリソースでアクセスできる一連の基本トークンが公開されます。これらの標準トークンを使用すると、独自のカスタムコンポーネントでSalesforce Lightning Design System (SLDS)の外観を模倣することができます。SLDSの変化に伴って、標準デザイントークンを使用してスタイル設定されたコンポーネントも変化します。標準デザイントークンのサブセットを使用して、エクスペリエンスビルダーで[テーマ]パネルと互換性のあるコンポーネントを作成します。

[テーマ]パネルでは、システム管理者がブランドプロパティを使用してサイト全体のスタイル設定をすばやく行うことができます。[テーマ]パネルの各プロパティは、1つ以上の標準デザイントークンと対応付けられます。システム管理者が[テーマ]パネルのプロパティを更新すると、そのプロパティに関連付けられたトークンを使用するLightningコンポーネントが更新されます。



エクスペリエンスビルダーサイトで使用可能なトークン

エクスペリエンスビルダーサイトでは、`force:base`から拡張した場合に次の標準トークンを使用できます。

重要: 標準トークン値はSLDSに合わせて変化します。使用可能なトークンとその値は、通知なしで変更される可能性があります。

[テーマ]パネルのプロパティ	対応付けられる標準デザイントークン
テキストの色	`colorTextDefault`
詳細テキストの色	• – `colorTextActionLabel` • `colorTextLabel`

[テーマ] パネルのプロパティ	対応付けられる標準デザイントークン
	• colorTextPlaceholder • colorTextWeak
アクションの色	• colorBackgroundButtonBrand • colorBorderBrand • colorBorderButtonBrand • colorBrand • colorTextBrand • colorTextActionLabelActive • colorTextTabLabelSelected メモ: Summer '18 以降、colorBackgroundHighlight はアクションの色には対応付けられません。
リンクの色	colorTextLink
会社のロゴ	brandLogoImage
フロート表示テキストの色	• colorTextButtonBrand • colorTextButtonBrandHover • colorTextInverse
境界線の色	• colorBorder • colorBorderButtonDefault • colorBorderInput • colorBorderSeparatorAlt
主要フォント	fontFamily
テキストの大文字/小文字	textTransform

また、次の標準トークンはテンプレートの派生テーマプロパティに使用できます。[テーマ] パネルでプロパティを更新すると、派生ブランドプロパティに間接的にアクセスできます。たとえば、[テーマ] パネルで [アクションの色] プロパティを変更した場合、新しい値に基づいて [アクションの濃い色] 値が再適用されます。

派生ブランドプロパティ	対応付けられる標準デザイントークン
アクションの濃い色 ([アクションの色] から派生)	• colorBackgroundButtonBrandActive • colorBackgroundButtonBrandHover
フロート表示の色 ([アクションの色] から派生)	• colorBackgroundButtonDefaultHover • colorBackgroundRowHover

派生ブランドプロパティ	対応付けられる標準デザイントークン
	• colorBackgroundRowSelected • colorBackgroundShade
リンクの濃い色 ([リンクの色] から派生)	• colorTextLinkActive • colorTextLinkHover

SLDS で使用可能なデザイントークンの完全なリストについては、Lightning Design System サイトの「Design Tokens (デザイントークン)」を参照してください。

メモ: 一部の標準コンポーネントは標準デザイントークンを使用しません。そのため、トークンを使用してテーマレイアウトのスタイルを設定している場合は、設定したスタイルを一部のコンポーネントがスタイルを継承しないことがあります。

関連トピック:

Lightning Aura コンポーネント開発者ガイド: 設計トークンを使用したスタイル設定
Lightning Aura コンポーネント開発者ガイド: アプリケーションでの Salesforce Lightning Design System の使用

Experience Cloud サイトを使用する組織のカスタムコンポーネントのセキュリティの確保

開発者はカスタムコンポーネントを使用して Experience Cloud サイトの機能やビジネスロジックをカスタマイズできます。他のカスタムソリューションと同様に、開発者は潜在的なセキュリティ関連の落とし穴に注意する必要があります。組み込み防御策をスキップすると、サイトと組織はセキュリティリスクにさらされる可能性があります。

たとえば、開発者が機密データをカスタムコンポーネントの定義にテキストとして保存すると、そのデータは露出する可能性があります。このような露出は、組織でデジタルエクスペリエンスが有効化され、組織にカスタムコンポーネントがあり、カスタムコンポーネントの API 参照名が認識されている場合に発生する可能性があります。露出は、サイトが公開か非公開かに関係なく発生する可能性があります。

露出したデータには次の情報が含まれている可能性があります。

- コンポーネント定義にテキストとして保存された機密情報
- HTML、JavaScript、CSS ファイルを含む、コンポーネントの完全なコンポーネント定義
- コンポーネント定義に含まれるその他のコンポーネントの名前
- コンポーネント定義で使用されている Apex コントローラーおよびメソッドの名前

このようなデータは、Salesforce 組織で使用されているか Experience Cloud サイトで使用されているかにかかわらず、また使用されていない場合も、組織のあらゆるカスタムコンポーネントで露出する可能性があります。

カスタムコンポーネントでのデータ露出のリスクを低減するには、次の手順に従います。

- 組織のすべてのカスタムコンポーネントのコンポーネント定義を確認する

- 機密データをコンポーネント定義に保存しない。機密データには、個人識別情報、会社の機密情報、またはビジネスや顧客にとって機密と思われる情報が含まれる可能性があります
- すべてのカスタムコントローラーを確認し、必要なユーザープロファイルのみにカスタムコントローラーへのアクセス権があることを確認する
- `@AuraEnabled` を使用して必要なメソッドのみが公開されていることを確認する
- 複雑で組織固有の、カスタムコンポーネントの命名規則を使用する

第 4 章 セキュアサイトの開発: 認証済みユーザーとゲストユーザー

トピック:

外部ユーザーや認証されていないゲストユーザーがアクセスできる Experience Cloud サイトを実装する場合、次のセキュリティの考慮事項に留意してください。外部ユーザーには Experience Cloud サイトへのログイン権限がありますが、内部 Salesforce 組織にはアクセスできません。ゲストユーザーは、Experience Cloud サイトの公開されているページおよびコンポーネントにアクセスできるインターネット上のユーザーです。

宣言型アクセスの制限

オブジェクトを参照するための権限を付与すると、外部ユーザーは標準コントローラーを使用してオブジェクトを参照できます。標準コントローラーは、Lightning機能が有効になっているエクスペリエンスビルダーサイトおよびSalesforceタブ + Visualforceサイトで使用できます。これらのコントローラーは、プラットフォームの宣言型権限のみに基づいてアクセス権を付与します。

外部ユーザーが仲介なしでコントローラーを介してアクセスできるオブジェクトのみを作成、参照、変更、削除するための宣言型アクセス権を付与します。Salesforceプラットフォームには、データの作成、参照、更新、削除に使用できる標準コントローラーが含まれています。標準UIコントローラーは、プラットフォームの共有ルール、作成、参照、更新、削除(CRUD)権限、項目レベルセキュリティ(FLS)でエンコードされる宣言型アクセスポリシーを適用します。オブジェクトを参照または更新するための権限を外部ユーザーに付与すると、外部ユーザーはその操作を実行できるようになります。実行されたくないオブジェクトに対する権限を必要以上に付与しないでください。

セキュリティモデルの決定

使用事例ごとに、カスタムアクセスコントロールモデルを実装するのか、宣言型プラットフォームアクセスコントロールモデルを使用するのかを決定します。可能な場合はプラットフォーム宣言型アクセスコントロールモデルを使用することをお勧めします。ただし、カスタムアクセスコントロールモデルが必要な場合もあります。

カスタムアクセスコントロールモデルが必要な場合:

1. コントローラーがアクセスするオブジェクトに対する宣言型データ権限(作成、参照、更新、削除(CRUD)、項目レベルセキュリティ(FLS)、共有など)を該当するユーザープロファイルおよび権限セットから削除します。共有を使用せずにコントローラーを宣言します。
2. 共有を使用せずにコントローラーに手続き型アクセスコントロールを実装します。営業を受けるコントローラーごとに、セキュリティポリシーで必要な手続き型アクセスコントロールロジックを実装します。

プラットフォームの宣言型アクセスコントロールモデルを使用できる場合:

1. 共有を使用してコントローラーを宣言する場合、プロファイルおよび権限セットごとに共有、オブジェクトのCRUD権限、FLSを適切に設定します。

コントローラーのセキュリティモデルの選択: 例

リードを作成するコントローラーを考えます。カスタムアクセスコントロールの例は次のとおりです。

- リードの作成前にCAPTCHAを要求する。
- リードの作成前に紹介コードを要求する。
- リードの作成前のライセンス契約に同意するようにユーザーに要求する。

これらの各例では、該当するポリシーで宣言型の共有、CRUD権限、FLSでは適用できない手続き型ステップが必要になります。これらのケースでは、Apexコントローラーでカスタムロジックを作成して手続き型ルールを適用します。コントローラーを呼び出して、ユーザーが基盤となるデータにのみアクセスできるようにするには、宣言型アクセス権(リードオブジェクトに対するCRUD、FLS、共有など)を削除する必要があります。

ただし、セキュリティポリシーをプラットフォームのCRUD権限、FLS、共有ロジックに対応付けることができる場合、適切な共有設定およびオブジェクトのCRUD権限を設定してそのロジックを実装します。その後、共有を使用してコントローラーを宣言します。

宣言型アクセス権を削除して、共有を使用しない手続き型ロジックルールを使用することには一定のリスクがあることに注意してください。

- 手続き型Apexコードを介して組織のセキュリティロジックを実装します。適切なプロファイル、レコード、ステートフルアクセスチェックを正しく実装できない場合、実装エラーとなり、データの不正アクセスにつながります。
- セキュリティポリシーでステートフルロジックが求められている場合、カスタムセッション管理ロジックを実装して要求間で状態を保持します。
- 手続き型アクセスコントロールロジックを作成する場合、組織のシステム管理者がすばやく管理および変更することが困難になります。

ユーザーが必ず標準コントローラーではなく独自のコントローラーを使用して基盤となるオブジェクトにアクセスするようにするには、CRUD権限およびFLSアクセス権を削除する必要があります。プラットフォームでは、認証されていないゲストユーザーを区別できないため、ほとんどの場合、カスタムアクセスコントロールを実装し、ゲストユーザーによるすべてのオブジェクトへの宣言型アクセスを拒否することが必要になります。

認証されていないゲストユーザーのガイドライン

宣言型アクセスコントロールモデルまたはカスタムアクセスコントロールモデルを選択する前に、レコードIDの暗号化と、認証されていないゲストユーザーへのさまざまなレベルのアクセス権の付与に関する次のガイドラインを考慮してください。

ゲストユーザーのレコード ID の暗号化

セキュリティ上の理由により、レコードを公開する場合を除き、ゲストユーザーがレコード ID でレコードを検索できないようにしてください。ゲストユーザーがレコードを作成して、後でそのレコードにアクセスする場合、レコード ID、レコード作成タイムスタンプ、現在のタイムスタンプの組み合わせを使用する暗号化された文字列を作成します。暗号化された文字列は、レコード作成者のみが持つレコードの一意の識別子として機能します。後日、ゲストユーザーは、要求を処理する Apex コードから暗号化された文字列を送信するように要求されます。その Apex コードで、文字列を復号化してレコード ID や他のレコード識別子を取得し、要求されたレコードを取得または更新します。

レコードを参照するためのアクセス権をゲストユーザーに付与

レコードデータを参照するためのアクセス権をゲストユーザーに付与するには、データを公開します。ガイドラインを確認し、データを損なうことなく必要なアクセス権をゲストユーザーに付与するように実装を設計します。

レコードを作成するためのアクセス権をゲストユーザーに付与

ゲストユーザーがオブジェクトレコードを作成できるようにするには、必要なオブジェクトへの作成アクセス権を含めるようにゲストユーザープロファイルを設定します。

レコードを更新するためのアクセス権をゲストユーザーに付与

ゲストユーザーがレコードを更新できるようにするには、共有を使用しないシステムコンテキストでアクションを実行します。ユーザーがレコードを更新できるようにするう前に、以前にユーザーに提供された暗号化されたトークンを確認することをお勧めします。正しいレコードであることを確認するには、レコードに関する情報 (作成者など) を確認します。

ゲストユーザーのレコード ID の暗号化

セキュリティ上の理由により、レコードを公開する場合を除き、ゲストユーザーがレコード ID でレコードを検索できないようにしてください。ゲストユーザーがレコードを作成して、後でそのレコードにアクセスする場合、レコード ID、レコード作成タイムスタンプ、現在のタイムスタンプの組み合わせを使用する暗号化された文字列を作成します。暗号化された文字列は、レコード作成者のみが持つレコードの一意の識別子として機能します。後日、ゲストユーザーは、要求を処理する Apex コードから暗号化された文字列を送信するように要求されます。その Apex コードで、文字列を復号化してレコード ID や他のレコード識別子を取得し、要求されたレコードを取得または更新します。

ヒント: User Encryption Decryption AppExchange パッケージでは、暗号化および復号化に `System.Crypto` Apex ライブラリを使用して、関連データを保存し、2つのテンプレートフローを提供する `UserCryptoHelper` クラスが提供されます。この管理パッケージを使用して、カスタマイズされたレコード ID の暗号化を実装したり、類似する独自の管理パッケージを作成したりします。

関連トピック:

- 共有なしのサンプルコード: レコードを作成して後で参照するためのアクセス権をゲストユーザーに付与
- 共有なしのサンプルコード: レコードを作成して後で更新するためのアクセス権をゲストユーザーに付与
- *Apex 開発者ガイド*
- *Apex リファレンスガイド*: Crypto クラス

レコードを参照するためのアクセス権をゲストユーザーに付与

レコードデータを参照するためのアクセス権をゲストユーザーに付与するには、データを公開します。ガイドラインを確認し、データを損なうことなく必要なアクセス権をゲストユーザーに付与するように実装を設計します。

警告: Summer '20 リリースでは、新しい設定が追加され、ゲストユーザーのレコードアクセス権に関するガイドラインが更新されています。Winter '21 以降、Summer '20 で導入されたガイドラインの更新が適用されます。Winter '21 リリース後は、以前の方法が機能しなくなるため、このドキュメントに記載されているいずれかの方法を使用します。

ゲストユーザーがレコードデータの参照を要求するたびに、共有ルールが要求に適用されるかどうかを判断するモードで応答が実行されます。モードごとにセキュリティ実装が異なるため、データの機密性を考慮して、ビジネスニーズに合った最も安全な方法を決定します。個別のフローや複数の Apex クラスを使用して、モードごとに実行する要求を分けることもできます。

機密情報の処理方法

データを認証されていないユーザーに返す前に、すべての機密情報をレコードから削除します。セキュリティ上の理由により、機密情報が含まれるレコードを取得するためにレコード ID などの推測可能な情報は使用してしないでください。

共有を使用する場合

共有を使用して実行されるレコード要求では、共有ルールでレコードへのアクセス権がゲストユーザーに付与されていない限りレコードにアクセスできません。次のシナリオに該当する場合は、参照のみアクセス権の共有ルールを検討してください。

- レコードを公開して誰でもアクセスできるようにする。
- 他のレコードを公開せずに共有ルールを使用して対象レコードを選択できる。

共有を使用しない場合

警告: 共有を使用せずに要求を実装する場合、組織の機密データが意図せず公開されないように慎重に要求および応答データを設計してください。

共有を使用しないシステムモードで実行されるレコード要求では、システムレベルアクセス権でアクションを実行し、共有ルールをスキップします。実行されるアクションや実行方法について注意を払わないと、共有を使用せずに実行されたときに要求でレコードデータが公開または変更される可能性があります。共有を使用せずに実行されるクエリでは、選択されたすべてのレコードが公開されます。

次のいずれかのシナリオに該当する場合は、共有を使用しないシステムモードを検討してください。

- レコードに対する参照のみ以上のアクセス権がゲストユーザーに必要である。
- レコードを公開したくない。
- 他のレコードを公開せずにレコードを選択することができない。
- 対象レコードが親子リレーションに含まれていて、子レコードへのアクセス権が親レコードの更新アクセス権によって制限されている。共有ルールでは、ゲストユーザーに更新アクセス権を付与できないため、このシナリオでは共有を使用せずに要求を実行します。

レコードを選択するための暗号化されたレコード ID

ゲストユーザーがレコードを作成して、後でそのレコードにアクセスする必要がある場合、レコード作成タイムスタンプを使用してレコード ID を暗号化し、暗号化された文字列をクライアントに返します。ゲストユーザーが長い文字列を入力しないで済むように、暗号化された文字列が含まれる URL をゲストユーザーに提供します。ゲストユーザーがレコードに対する参照アクセス権を要求すると、URL の暗号化された文字列を取得します。レコードを選択するには、復号化されたレコード ID を使用します。

Lightning のコンポーネント

オブジェクトの項目に直接リンクする Lightning コンポーネントは、オブジェクトの作成、参照、更新、削除 (CRUD) 権限と項目レベルセキュリティ (FLS) のチェックを自動的に実行し、コンポーネントがユーザーに表示されるかどうかを判断します。ゲストユーザーと共有しないレコードの場合、CRUD と FLS のチェックは失敗し、コンポーネントは表示されません。Lightning コンポーネントを使用してそれらのレコードを表示するには、変数に値を設定し、Apex コードでそれらの変数をオブジェクトの項目に個別に割り当てます。

この方法では、CRUD と FLS の自動チェックが回避されるため、次のガイドラインに従って Lightning コンポーネントを実装してください。

- 必要なレコード項目のみをクエリに含めます。
- 機密項目をクライアント側のコードに渡さないでください。
- クライアントで必要な項目のみをクライアントに渡します。クライアントに送信されたデータはすべて公開されます。

コードサンプル

- 共有を使用するサンプルコード: レコードを参照するためのアクセス権をゲストユーザーに付与
- 共有なしのサンプルコード:同じトランザクションでレコードを作成して参照するためのアクセス権をゲストユーザーに付与
- 共有なしのサンプルコード: レコードを作成して後で参照するためのアクセス権をゲストユーザーに付与

フローのサンプル

- 共有を使用するサンプルフロー: レコードを参照するためのアクセス権をゲストユーザーに付与
- 共有なしのサンプルフロー:1つのフローでレコードを作成して参照するためのアクセス権をゲストユーザーに付与

関連トピック:

Salesforce ヘルプ: ゲストユーザーの共有設定とレコードアクセスの保護

Salesforce ヘルプ: 共有ルール

Salesforce セキュリティガイド: 共有ルール

レコードを作成するためのアクセス権をゲストユーザーに付与

ゲストユーザーがオブジェクトレコードを作成できるようにするには、必要なオブジェクトへの作成アクセス権を含めるようにゲストユーザープロファイルを設定します。

オブジェクトへの作成アクセス権を付与するには、オブジェクトへの参照アクセス権を付与する必要があります。該当のオブジェクトに対する参照アクセス権が不要な場合は、そのオブジェクトのすべての権限を削除して、共有なしコントローラーで作成ロジックを実行することをお勧めします。

ヒント: ゲストが作成したデータで検証ルールを実行し、そのデータが自動化プロセスに影響しないことを確認します。

レコード ID とゲストユーザー

レコードの作成後にクライアントへの応答にレコード ID を含めないでください。以後のアクセス用に一意のレコード識別子を作成するには、レコード作成タイムスタンプを使用してレコード ID を暗号化し、暗号化した文字列をクライアントに返します。

レコードを取得すると、レコード ID がオブジェクトに自動的に含まれます。オブジェクトからレコード ID を削除して、クライアントにレコード ID を渡さないようにします。

Apex メソッドでのレコードの作成とアクセス

ゲストユーザー共有ルールはトランザクションの完了後に有効になります。共有ありの Apex コードでレコードを作成し、新しく作成したレコードを要求した場合、ゲストユーザーが共有ルールを使用してレコードにアクセスしていると、ゲスト共有ルールはまだ有効になっていないため参照要求は失敗します。ゲストユーザーがレコードを作成し、その新しく作成したレコードを同じメソッド内で参照できるようにするには、`without sharing` キーワードを使用してクラスを作成します。

フローでのレコードの作成とアクセス

フローでは、インタビューで画面、ローカルアクション、または一時停止要素が実行されるまでレコードの作成要素でレコードは作成されません。フローでレコードを作成し、同じフローで同じレコードを参照するには、レコードの作成と取得の間に画面を挿入するか、レコードを Apex アクションで参照します。

サンプル

- サンプルフロー: レコードを作成するためのアクセス権をゲストユーザーに付与
- 共有なしのサンプルフロー:1つのフローでレコードを作成して参照するためのアクセス権をゲストユーザーに付与
- 共有なしのサンプルコード:同じトランザクションでレコードを作成して参照するためのアクセス権をゲストユーザーに付与

関連トピック:

Salesforce ヘルプ: ゲストユーザープロファイルの設定

ゲストユーザーのレコード ID の暗号化

レコードを更新するためのアクセス権をゲストユーザーに付与

ゲストユーザーがレコードを更新できるようにするには、共有を使用しないシステムコンテキストでアクションを実行します。ユーザーがレコードを更新できるようにするう前に、以前にユーザーに提供された暗号化されたトークンを確認することをお勧めします。正しいレコードであることを確認するには、レコードに関する情報 (作成者など) を確認します。

警告: Summer '20 リリースでは、新しい設定が追加され、ゲストユーザーのレコードアクセス権に関するガイドラインが更新されています。Winter '21 以降、Summer '20 で導入されたガイドラインの更新が適用されます。Winter '21 リリース後は、共有ルールを使用してレコードに対する更新アクセス権をゲストユーザーに付与できなくなるため、without sharing モードを使用する必要があります。

Lightning コンポーネントとゲストユーザー

オブジェクトの項目に直接リンクする Lightning コンポーネントは、オブジェクト権限と項目レベルセキュリティ (FLS) のチェックを自動的に実行し、コンポーネントがユーザーに表示されるかどうかを判断します。[ゲストユーザーのレコードアクセス権を保護] 設定が有効になっている場合、レコードを更新するためのアクセス権をゲストユーザーに付与できません。その場合、更新権限を必要とするオブジェクト権限チェックは失敗し、コンポーネントは表示されません。

Lightning コンポーネントを使用してゲストユーザーの入力を処理するには、変数を使用して Lightning コンポーネントの値を設定します。次に、Apex コードでそれらの変数をレコードの項目に個別に関連付けます。この方法では、オブジェクト権限と FLS の自動チェックが回避されるため、次のガイドラインに従って Lightning コンポーネントを実装してください。

- 更新を実行する前に、サーバー側のコードを使用してレコードを取得し、更新するレコードであることを確認します。
- ユーザーに表示しないレコード項目をクライアントに渡さないでください。クライアントに送信するデータは公開されます。
- レコード ID をクライアントに渡さないでください。クライアントからレコード ID を受け入れないでください。レコードの一意の識別子を作成するには、レコード ID を文字列として暗号化します。
- サーバー側のロジックを使用して、目的の項目のみを更新するように制限します。クライアント側のコードを使用して、サーバー側の動作を決定しないでください。
- コードで更新を実行する前にサーバー側のロジックを使用してクライアントのデータを検証します。

関連トピック:

共有なしのサンプルコード: レコードを作成して後で更新するためのアクセス権をゲストユーザーに付与

Salesforce Developers Wiki (Salesforce 開発者 Wiki): Enforcing CRUD and FLS (CRUD および FLS の適用)

ゲストユーザーのレコード ID の暗号化

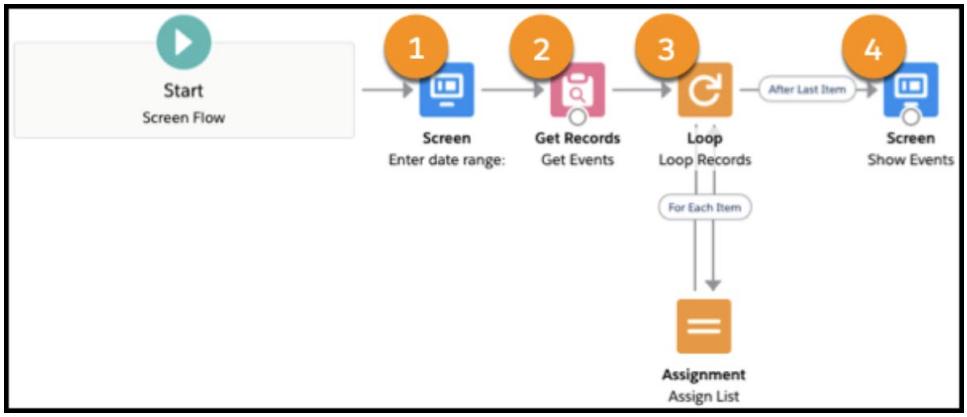
宣言型アクセスコントロールモデルの例

次のコードおよびフローの例では、宣言型アクセスコントロールモデルを使用して、レコードを参照するためのアクセス権を認証されていないゲストユーザーに付与します。

共有を使用するサンプルフロー: レコードを参照するためのアクセス権をゲストユーザーに付与

このサンプルフローでは、ゲストユーザーが日付範囲を入力し、その範囲内のイベントを表示します。ゲストユーザーには、共有ルールを介してレコードを参照するためのアクセス権があるため、ゲストユーザープロファイルによって、フローでアクセスできる項目が決まります。

重要: 参照アクセス権をゲストユーザーに付与する前に、「レコードを参照するためのアクセス権をゲストユーザーに付与」を参照してください。



フロー設定

ゲストユーザーは共有ルールを介してレコードにアクセスできるため、[フローの実行方法] を [ユーザーまたはシステムコンテキスト — フローの起動方法に依存します] に設定します。

日付範囲の入力 (1)

フローの最初の要素は、開始日と終了日の入力項目を表示する画面です。この要素は、入力日付を `Start_Date` および `End_Date` 変数に保存します。

イベントの取得 (2)

次の要素は、次の条件に一致するイベントを選択する [レコードを取得] クエリです。

- イベントの StartDateTime が Start_Date 変数よりも大きい。
- イベントの EndDateTime が End_Date 変数よりも小さい。
- イベントの isPrivate 値が False である。
- イベントの isArchived 値が False である。

この要素は、選択されたイベントを GetEvents 変数に保存します。

レコードのループ (3)

[ループ] 要素は、GetEvents 変数の各イベントをループします。

ループ内で、[割り当て] 要素が各イベントの StartDateTime、EndDateTime、Subject、Location を文字列に追加します。

イベントの表示 (4)

最後の要素は、すべてのイベントが含まれる文字列を表示する画面です。

関連トピック:

Salesforce ヘルプ: ゲストユーザーの共有設定とレコードアクセスの保護
Salesforce ヘルプ: 共有ルール
Salesforce ヘルプ: フローが実行されるコンテキスト
Salesforce ヘルプ: フロー要素

共有を使用するサンプルコード: レコードを参照するためのアクセス権をゲストユーザーに付与

この一連のコードサンプルでは、ゲストユーザーが日付範囲を入力し、その範囲内のイベントを表示します。ゲストユーザーには、共有ルールを介してレコードを参照するためのアクセス権があります。

重要: 参照アクセス権をゲストユーザーに付与する前に、「レコードを参照するためのアクセス権をゲストユーザーに付与」を参照してください。

Aura コンポーネント; DisplayEvents.cmp

このサンプル Aura コンポーネントでは、ユーザーがイベントを表示するための開始日と終了日を入力する 2 つの lightning:input コンポーネントが表示されます。lightning:card コンポーネントには、各イベントの StartDateTime、EndDateTime、Subject、Location が表示されます。

```
<aura:component controller="GuestUserEventsAuraController">

    <aura:attribute name="events" type="Event[]"/>
    <aura:attribute name="StartDate" type="String" default=""/>
    <aura:attribute name="EndDate" type="String" default=""/>

    <lightning:input type="datetime" name="StartDate" value="{!v.StartDate}"
aura:id="StartDate" label="Start after: " required="true"/>
```

```
    <lightning:input type="datetime" name="EndDate" value="{!v.EndDate}" aura:id="EndDate"
 label="End before: " required="true"/>
    <lightning:button name="Submit" variant="brand" label="Find events" title="Find events"
 onclick="{!c.handleSearch}"/>

    <lightning:card title="Events">
        <p class="slds-p-horizontal--small">
            <aura:iteration items="{!v.events}" var="event">
                {!event.Subject} ({!event.Location}) starts at {!event.StartDateTime} and
 ends at {!event.EndDateTime} <br/>
            </aura:iteration>
        </p>
    </lightning:card>
</aura:component>
```

コンポーネントコントローラー: DisplayEventsController.js

このサンプル JavaScript コントローラーは、Aura コンポーネントのイベントを処理し、ヘルパーファイルのメソッドをコールします。

```
({
    handleSearch : function(component, event, helper) {
        helper.doSearch(component, event, helper);
    }
})
```

JavaScript ヘルパー: DisplayEventsHelper.js

この JavaScript ヘルパーは、ユーザーが送信した 2 つのタイムスタンプ内のイベントを検索する非同期要求を作成し、要求の完了時に実行されるアクションを定義します。

```
({
        doSearch : function(component, event, helper) {
            var start_date = component.find("StartDate").get("v.value");
            var end_date = component.find("EndDate").get("v.value");
            var action = component.get("c.searchEvents");
            action.setParams({
                "start_date": start_date,
                "end_date": end_date
            });
            action.setCallback(this, function(response){
                component.set("v.events", response.getReturnValue());
            });
            $A.enqueueAction(action);
        }
})
```

Apex コントローラー: GuestUserEventsAuraController.cls

このサンプル Apex コントローラーは、JavaScript ヘルパーからレコードを検索するコールを受信します。次の条件と一致するイベントを選択します。

- イベントの `StartDateTime` が `Start_Date` パラメーターよりも大きい。
- イベントの `EndDateTime` が `End_Date` パラメーターよりも小さい。
- イベントの `isPrivate` 値が `False` である。
- イベントの `isArchived` 値が `False` である。

クエリから各イベントの次の項目が返されます。

- `StartDateTime`
- `EndDateTime`
- `Location`
- `Subject`
- `Id`

ゲストユーザーにはレコード ID は必要ないため、`for` ループで他のすべての項目が新しいイベントオブジェクトにコピーされます。次に、新しいオブジェクトが新しいリストに追加され、リストがクライアントに返されます。

ゲストユーザーは共有ルールを介してレコードにアクセスできるため、`with sharing` キーワードでクラスが定義されます。

警告: `@AuraEnabled` メソッドは、インターネット上の任意のシステムまたは個人が呼び出すことができます。手続き型アクセスチェックを実装して、メソッドの実行を保護します。クエリで目的のレコードと必須項目のみが選択されることを確認します。

```
public with sharing class GuestUserEventsAuraController {

    @AuraEnabled
    public static List<Event> searchEvents(Datetime start_date, Datetime end_date){
        List<Event> results = [SELECT Event.Subject,
                                      Event.StartDateTime,
                                      Event.EndDateTime,
                                      Event.Location
                FROM Event
                WHERE Event.EndDateTime<:end_date AND
                      Event.StartDateTime>:start_date AND
                      Event.isPrivate=False AND
                      Event.isArchived=False];

        List<Event> filtered_events = new List<Event>();
        for (Event event : results) {
            Event new_event = new Event(Subject = event.Subject,
                                                                  StartDateTime =
event.StartDateTime,
                                                                  EndDateTime =
event.EndDateTime,
                                                                  Location = event.Location);

            filtered_events.add(new_event);
        }
        return filtered_events;
```

```
    }
}
```

関連トピック:

Salesforce Developers Wiki (Salesforce 開発者 Wiki): Enforcing CRUD and FLS (CRUD および FLS の適用)

Salesforce ヘルプ: ゲストユーザーの共有設定とレコードアクセスの保護

Salesforce ヘルプ: 共有ルール

Apex 開発者ガイド

Lightning Aura コンポーネント開発者ガイド

カスタムアクセスコントロールモデルの例

次のコードおよびフローの例では、カスタムアクセスコントロールモデルを使用して、レコードを作成するためのアクセス権を認証されていないゲストユーザーに付与します。

共有なしのサンプルコード: レコードを作成して後で参照するためのアクセス権をゲストユーザーに付与

これらのコードサンプルでは、2つの個別のインタラクションがサポートされています。最初のインタラクションで、ゲストユーザーはケースを作成します。以降のアクセスを許可するために、Apex メソッドはレコード ID を暗号化された文字列に置き換えます。ゲストユーザーが後でケースを参照する場合、暗号化された文字列を入力します。Apex メソッドは、文字列を復号化し、それを使用してケースを取得します。

サンプルフロー: レコードを作成するためのアクセス権をゲストユーザーに付与

このサンプルフローでは、ゲストユーザーがフィードバックを入力し、フローでそのフィードバックをカスタムオブジェクトレコードに保存します。ゲストユーザーには、作成後にレコードを参照するためのアクセス権はありません。

共有なしのサンプルコード:同じトランザクションでレコードを作成して参照するためのアクセス権をゲストユーザーに付与

この一連のコードサンプルでは、ゲストユーザーが詳細を入力してサポート問題を報告し、Apex コードでケースを作成します。作成後、Apex メソッドは新しいレコードを取得し、Aura コンポーネントでレコードの一部をゲストユーザーに表示します。レコードへのアクセス権をゲストユーザーに付与するためにオブジェクト権限やプラットフォーム共有は使用されないため、Apex コードは共有を使用せずに実行されます。

共有なしのサンプルフロー:1つのフローでレコードを作成して参照するためのアクセス権をゲストユーザーに付与

このサンプルフローでは、ゲストユーザーが詳細を入力してサポート問題を報告し、フローでケースを作成します。ゲストユーザーがレコードを作成すると、デフォルトの有効ユーザーがレコードの所有者になり、ゲストユーザーはレコードに直接アクセスできなくなります。その後、フローで新規ケースを取得してケースの `CaseNumber` および `Status` 項目を取得し、こられの項目をゲストユーザーに表示します。レコードの作成後、ゲストユーザーはレコードを所有しておらず、フローでレコードを取得する必要があるため、フローは共有を使用せずに実行されます。

共有なしのサンプルコード: レコードを作成して後で更新するためのアクセス権をゲストユーザーに付与
これらのコードサンプルでは、2つの個別のインタラクションがサポートされています。最初のインタラクションで、ゲストユーザーはケースを作成します。セキュリティ上の理由により、Apex メソッドはレコード ID を暗号化された文字列に置き換えます。ゲストユーザーが後でケースをクローズする場合、暗号化された文字列を入力します。Apex メソッドは、文字列を復号化してレコード ID を取得します。次に、レコード ID を使用してケースを選択し、ケースの状況を更新します。

共有なしのサンプルコード: レコードを作成して後で参照するためのアクセス権をゲストユーザーに付与

これらのコードサンプルでは、2 つの個別のインタラクションがサポートされています。最初のインタラクションで、ゲストユーザーはケースを作成します。以降のアクセスを許可するために、Apex メソッドはレコード ID を暗号化された文字列に置き換えます。ゲストユーザーが後でケースを参照する場合、暗号化された文字列を入力します。Apex メソッドは、文字列を復号化し、それを使用してケースを取得します。

Aura コンポーネント: CreateCase.cmp

このサンプル Aura コンポーネントでは、ゲストユーザーが新規ケースに関する詳細や既存のケースのトークンを入力できるいくつかのコンポーネントが表示されます。レコードの作成後、`lightning:card` コンポーネントには、新規ケースの暗号化されたトークンや、トークンと一致するケースの状況が表示されます。

デモのために、このサンプルではゲストユーザーがケースのトークンを入力できる項目を使用します。このシナリオを実装するには、トークンが含まれるリンクをゲストユーザーに提供し、URL からトークンを取得します。

```
<aura:component controller="GuestUserCreateForLater">
    <aura:attribute name="caseID" type="String"/>
    <aura:attribute name="case_status" type="String"/>
    <aura:attribute name="subject" type="String"/>
    <aura:attribute name="description" type="String"/>
    <aura:attribute name="email" type="String"/>

     Enter details to create a new case
    <lightning:input type="email" name="email" required="true" value="{!v.email}"
aura:id="email" label="Where should we send email updates?"/>
    <lightning:input name="subject" label="Subject" required="true" value="{!v.subject}"
aura:id="subject"/>
    <lightning:textarea name="description" required="true" label="Description"
value="{!v.description}" aura:id="description"/>
   <lightning:button name="submit" variant="brand" label="Create case" title="Create case"
 onclick="{!c.submitCase}"/>

    <aura:if isTrue="{!v.caseID}">
        <lightning:card title="Case">
            <p class="slds-p-horizontal--small">
                New case created:
                <p>{!v.caseID}</p>
            </p>
        </lightning:card>
```

```
    </aura:if>

    Or enter an existing case token to view the status of the case
    <lightning:textarea name="existing_case" required="false" label="Existing case token"
 aura:id="existing_case"/>
 <lightning:button name="submit" variant="brand" label="Lookup case" title="Lookup case"
onclick="{!c.lookupCase}"/>
    <aura:if isTrue="{!v.case_status}">
        <lightning:card title="Case">
            <p class="slds-p-horizontal--small">
                Case status:
                <p>{!v.case_status}</p>
            </p>
        </lightning:card>
    </aura:if>
</aura:component>
```

コンポーネントコントローラー: CreateCaseController.js

このサンプル JavaScript コントローラーは、Aura コンポーネントのイベントを処理し、ヘルパーファイルのメソッドをコールします。

```
({
    submitCase : function(component, event, helper) {
        helper.makeCase(component, event, helper);
    }
    lookupCase : function(component,event,helper){
        helper.getCase(component,event,helper);
    }
})
```

JavaScript ヘルパー: DisplayCaseHelper.js

JavaScript ヘルパーには、次の 2 つのメソッドがあります。

`makeCase()`
`makeCase()` メソッドは、送信されたデータを使用してケースを作成する非同期要求を作成します。要求が完了すると、コールバックで新規ケースの一意のトークンが Aura コンポーネントで使用される変数の `caseID` 項目に保存されます。

`getCase()`
`getCase()` メソッドは、ゲストユーザーが入力したトークンを使用して、トークンと一致するケースを非同期に取得します。メソッドのコールバックで Apex メソッドからの応答がキャッシュされ、値が `case_status` 変数に保存されます。

```
({
    makeCase : function(component, event, helper) {
        var subject = component.find("subject").get("v.value");
        var description = component.find("description").get("v.value");
        var email = component.find("email").get("v.value");

        var action = component.get("c.CreateCase");
        action.setParams({
```

```
            "subject": subject,
            "description": description,
            "email": email
        });
        action.setCallback(this, function(response){
            component.set("v.caseID", response.getReturnValue());
        });
        $A.enqueueAction(action);
    },
    getCase : function(component,event,helper){
        var case_token = component.find("existing_case").get("v.value");
        var action = component.get("c.GetCase");
        action.setParams({
            "token":case_token
        });
        action.setCallback(this, function(response){
            component.set("v.case_status", response.getReturnValue());
        });
        $A.enqueueAction(action);
    }
})
```

Apex コントローラー: GuestUserCreateForLater.cls

このサンプル Apex コントローラーは、ケースを作成および取得するコールを受信します。User Encryption Decryption AppExchange パッケージを使用して、データを暗号化および復号化します。

CreateCase()

`CreateCase()` Apex メソッドは、ゲストユーザーの入力を使用してケースを作成します。レコードの作成後、レコード ID、レコードの `CreatedDate` 項目、現在のタイムスタンプから暗号化された文字列を生成します。Apex メソッドは、暗号化された文字列を返します。

GetCase()

`GetCase()` メソッドは、提供された文字列を復号化して結果を検証し、復号化されたレコード ID と作成されたタイムスタンプをヘルパーメソッドに渡し、元のレコードを取得します。応答はレコードの状況になります。

レコードの作成やレコードへのアクセスのためにオブジェクト権限やプラットフォーム共有は使用されないため、`without sharing` キーワードを使用してクラスを定義します。

警告: `@AuraEnabled` メソッドは、インターネット上の任意のシステムまたは個人が呼び出すことができます。クエリで新規作成されたレコードのみを取得でき、必須項目のみが選択されることを確認します。

```
public class without sharing GuestUserCreateForLater {

    @AuraEnabled
    public static String CreateCase(String subject,
                                     String description,
                                     String email){
  Case new_case = new Case(Subject=subject,
                           Description=description,
                           SuppliedEmail=email);
        insert new_case;
```

```
        List<Case> results = getCase(new_case.Id);


        String encryptedID = ued.UserCryptoHelper.doEncrypt(results[0].Id+'|'+
results[0].CreatedDate.getTime() +'|'+System.DateTime.now().getTime());
        return encryptedID;
    }

    public static final Long validTimestampMinutes = 10;

    @AuraEnabled
    public static String GetCase(String token){
        String status = 'Case not found';
        String decrypted_token = '';
        try {
            decrypted_token = ued.UserCryptoHelper.doDecrypt(token);
        } catch(Exception e) {
            return status;
        }

        String[] decrypted_parts = decrypted_token.split('\\|');
        String decryptedRecordId = decrypted_parts[0];
        String created_timestamp = decrypted_parts[1];
        String original_request_timestamp = decrypted_parts[2];



        if( isTimestampValid(System.Long.valueOf(original_request_timestamp)) ){
         List<Case> caseList = getCase(decryptedRecordId, created_timestamp);
         if(caseList.size() == 1){
             status = caseList[0].Status;
      }else{
                 status = 'Case not found';
         }
     }
        return status;

    }

    private static List<Case> getCase(String caseID, Datetime created_date)
    {
        List<Case> results = [SELECT Case.CaseNumber, Case.CreatedDate, Case.Status
        FROM Case
        WHERE Case.Id=:caseID AND Case.CreatedDate=:created_date];
        return results;
    }

    private static Boolean isTimestampValid(Long timestamp)
    {
        return ((System.now().getTime() - timestamp) / 60000) < validTimestampMinutes;
    }
}
```

メモ: 機密性の高い情報を読み込む場合、セキュリティの向上のために次のいずれかの追加の対策を検討してください。

- 参照または変更しようとしているデータに関連する、本人しかしらない追加情報を入力するようにユーザーに求める。
- データを参照または変更するときにログインするようにユーザーに求める。

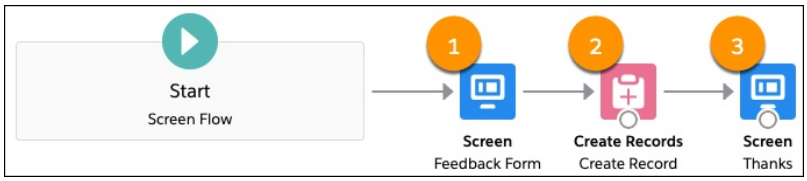
関連トピック:

Salesforce ヘルプ: ゲストユーザーの共有設定とレコードアクセスの保護
Apex 開発者ガイド
Lightning Aura コンポーネント開発者ガイド

サンプルフロー: レコードを作成するためのアクセス権をゲストユーザーに付与

このサンプルフローでは、ゲストユーザーがフィードバックを入力し、フローでそのフィードバックをカスタムオブジェクトレコードに保存します。ゲストユーザーには、作成後にレコードを参照するためのアクセス権はありません。

重要: レコードを作成するためのアクセス権をゲストユーザーに付与する前に、「レコードを作成するためのアクセス権をゲストユーザーに付与」を参照してください。



Feedback__c カスタムオブジェクト

このシナリオでは、Feedback__cカスタムオブジェクトを使用して、ゲストユーザーからのフィードバックを保存します。Feedback__c カスタムオブジェクトには、次の項目があります (アルファベット順)。

`Email__c`
必須。ゲストユーザーのメールアドレス。データ型: メール

`Score__c`
必須。ゲストユーザーが入力したフィードバックスコア。可能な値は、`0`、`1`、`2`、`3`、`4`、`5` です。

`Additional_comments__c`
ゲストユーザーが入力した追加のフィードバック。データ型: ロングテキストエリア

フロー設定

フローでレコードに対する参照アクセス権は必要なく、オブジェクト権限は使用されないため、[フローの実行方法] を [システムコンテキスト共有なし — すべてのデータにアクセス] に設定します。

フィードバックフォーム (1)

フローの最初の要素は、次のコンポーネントを表示する画面です。

- ユーザーのメールアドレスのメールコンポーネント。
- 0～5の整数を許可するように設定されたユーザーのフィードバックスコアのスライダーコンポーネント。
- 追加コメントのロングテキストエリアコンポーネント。

レコードの作成 (2)

次の要素は、Feedback__c レコードを作成する [レコードを作成] 要素です。

終了画面 (3)

最後の画面要素は、フィードバックについてユーザーに感謝の意を表すテキストを表示します。

関連トピック:

Salesforce ヘルプ: フローが実行されるコンテキスト
Salesforce ヘルプ: フロー要素
Salesforce ヘルプ: ゲストユーザーによるフローへのアクセスの許可
Salesforce ヘルプ: フローが実行されるコンテキスト
Salesforce ヘルプ: フロー要素
Salesforce ヘルプ: ゲストユーザーによるフローへのアクセスの許可

共有なしのサンプルコード: 同じトランザクションでレコードを作成して参照するためのアクセス権をゲストユーザーに付与

この一連のコードサンプルでは、ゲストユーザーが詳細を入力してサポート問題を報告し、Apex コードでケースを作成します。作成後、Apex メソッドは新しいレコードを取得し、Aura コンポーネントでレコードの一部をゲストユーザーに表示します。レコードへのアクセス権をゲストユーザーに付与するためにオブジェクト権限やプラットフォーム共有は使用されないため、Apex コードは共有を使用せずに実行されます。

Aura コンポーネント: CreateCase.cmp

このサンプル Aura コンポーネントでは、ユーザーがケースに関する詳細を入力するいくつかのコンポーネントが表示されます。作成後、`lightning:card` コンポーネントには、新しいケースのケース番号と状況が表示されます。

```
<aura:component controller="GuestUserCreateCase">

    <aura:attribute name="caseNumber" type="String"/>
    <aura:attribute name="status" type="String"/>
    <aura:attribute name="subject" type="String" default=""/>
    <aura:attribute name="description" type="String" default=""/>
    <aura:attribute name="email" type="String" default=""/>
    <aura:attribute name="name" type="String" default=""/>
```

```
    <aura:attribute name="reason" type="String"/>
    <aura:attribute name="type" type="String" default=""/>

    <lightning:select name="select" label="Reason" required="true" value="{!v.reason}"
aura:id="reason">
        <option value="installation">Installation</option>
        <option value="equipmentcomplexity">Equipment Complexity</option>
        <option value="performance">Performance</option>
        <option value="breakdown">Breakdown</option>
        <option value="equipmentdesign">Equipment Design</option>
        <option value="feedback">Feedback</option>
        <option value="other">Other</option>
    </lightning:select>

    <lightning:select name="type" label="Type" required="true" value="{!v.type}"
aura:id="type">
        <option value="mechanical">Mechanical</option>
        <option value="electrical">Electrical</option>
        <option value="electronic">Electronic</option>
        <option value="structural">Structural</option>
        <option value="other">Other</option>
    </lightning:select>

    <lightning:input type="email" name="email" required="true" value="{!v.email}"
aura:id="email" label="Where should we send email updates?"/>
    <lightning:input name="name" label="Name" required="true" value="{!v.name}"
aura:id="name"/>


    <lightning:input name="subject" label="Subject" required="true" value="{!v.subject}"
aura:id="subject"/>
    <lightning:textarea name="description" required="true" label="Description"
value="{!v.description}" aura:id="description"/>

   <lightning:button name="submit" variant="brand" label="Submit case" title="Submit case"
 onclick="{!c.submitCase}"/>

    <aura:if isTrue="{!v.caseNumber}">
        <lightning:card title="Case">
            <p class="slds-p-horizontal--small">
                {!v.caseNumber} has status {!v.status}.
            </p>
        </lightning:card>
    </aura:if>
</aura:component>
```

コンポーネントコントローラー: CreateCaseController.js

このサンプル JavaScript コントローラーは、Aura コンポーネントのイベントを処理し、ヘルパーファイルのメソッドをコールします。

```
({
    submitCase : function(component, event, helper) {
        helper.makeCase(component, event, helper);
```

```
    }
})
```

JavaScript ヘルパー: DisplayCaseHelper.js

このJavaScriptヘルパーは、送信されたデータを使用してケースを作成する非同期要求を作成します。要求が完了すると、コールバックでケース番号とケース状況がAuraコンポーネントで使用される変数に保存されます。

```
({
        makeCase : function(component, event, helper) {
            var subject = component.get("v.subject");
            var description = component.get("v.description");
            var email = component.get("v.email");
            var name = component.get("v.name");
            var reason = component.get("v.reason");
            var type = component.get("v.type");


            var action = component.get("c.CreateCase");
            action.setParams({
                "subject": subject,
                "description": description,
                "email": email,
                "name": name,
                "reason": reason,
                "caseType": type
            });
            action.setCallback(this, function(response){
                component.set("v.caseNumber", response.getReturnValue()[0]);
                component.set("v.status", response.getReturnValue()[1]);
            });
            $A.enqueueAction(action);
        }
})
```

Apex コントローラー: GuestUserCreateCase.apxc

このサンプル Apex コントローラーは、レコードを作成して、新しいレコードを取得し、新しいレコードの必須項目をクライアントに返します。オブジェクト権限とプラットフォーム共有は使用されないため、このコントローラーは共有なしで実行されます。

レコードデータが意図せず公開されないように、`CreateCase` メソッドは `CaseNumber` および `Status` 項目のみを返します。

警告: `@AuraEnabled` クラスは、インターネット上の任意のシステムまたは個人が呼び出すことができます。メソッドから新しいレコードの必須項目のみが返されていることを確認します。

```
public without sharing class GuestUserCreateCase {

    @AuraEnabled
    public static List<String> CreateCase(String subject,
                                           String description,
```

```
                                        String email,
                                        String name,
                                        String reason,
                                        String caseType,
                                        String phone){
    Case new_case = new Case(Subject=subject,
                                    Description=description,
                                    SuppliedEmail=email,
                                    SuppliedName=name,
                                    Reason=reason,
                                    Type=caseType,
                                    SuppliedPhone=phone);
            insert new_case;

            List<Case> results = getCase(new_case.Id);

            List<String> response = new List<String>();
            response.add(results[0].CaseNumber);
            response.add(results[0].Status);
            return response;

        }

        private static List<Case> getCase(String caseID)
        {
            List<Case> results = [SELECT Case.CaseNumber, Case.CreatedDate
            FROM Case
            WHERE Case.Id=:caseID];
            return results;
        }

}
```

関連トピック:

Salesforce **ヘルプ: ゲストユーザーの共有設定とレコードアクセスの保護**

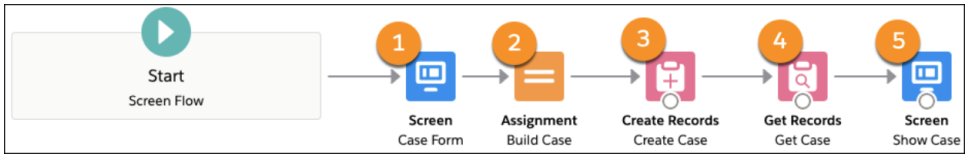
Apex **開発者ガイド**

Lightning Aura **コンポーネント開発者ガイド**

共有なしのサンプルフロー:1つのフローでレコードを作成して参照するためのアクセス権をゲストユーザーに付与

このサンプルフローでは、ゲストユーザーが詳細を入力してサポート問題を報告し、フローでケースを作成します。ゲストユーザーがレコードを作成すると、デフォルトの有効ユーザーがレコードの所有者になり、ゲストユーザーはレコードに直接アクセスできなくなります。その後、フローで新規ケースを取得してケースの `CaseNumber` および `Status` 項目を取得し、これらの項目をゲストユーザーに表示します。レコードの作成後、ゲストユーザーはレコードを所有しておらず、フローでレコードを取得する必要があるため、フローは共有を使用せずに実行されます。

重要: 作成および参照アクセス権をゲストユーザーに付与する前に、「レコードを参照するためのアクセス権をゲストユーザーに付与」および「レコードを作成するためのアクセス権をゲストユーザーに付与」を参照してください。



フロー設定

フローではレコードを作成し、共有を使用せずにレコードを取得するため、[フローの実行方法] を [システムコンテキスト共有なし — すべてのデータにアクセス] に設定します。

ケースフォーム (1)

フローの最初の要素は、次の入力コンポーネントを表示する画面です。

- 会社の名前のテキストコンポーネント
- 申請者の名前の名前コンポーネント
- 申請者のメールアドレスのメールコンポーネント
- 申請者の電話番号の電話コンポーネント
- レコードタイプの `Type_Options` 項目のオプション値が含まれる選択リストコンポーネント
- レコードタイプの `Reason_Options` 項目のオプション値が含まれる選択リストコンポーネント
- ケースの件名のテキストコンポーネント
- ケースの説明のロングテキストエリアコンポーネント

割り当て (2)

2 番目の要素は、入力コンポーネントのデータを新しい `Case` レコード変数に割り当てます。

レコードの作成 (3)

次の要素は、`Case` レコード変数を使用してケースレコードを作成する [レコードを作成] 要素です。ゲストユーザーが入力した情報に加えて、要素を設定して、ケースの元の項目を `Web` として定義します。

レコードの取得 (4)

[レコードを取得] 要素は、[レコードを作成] 要素で自動的に定義される `Id` 項目を使用して新規レコードを取得します。取得されたレコードは、新しい `Case` レコード変数に保存されます。

終了画面 (5)

最後の画面要素は、[レコードを取得] 要素の `Case` レコード変数のケースの `CaseNumber` および `Status` 項目を表示します。

関連トピック:

- *Salesforce ヘルプ*: フローが実行されるコンテキスト
- *Salesforce ヘルプ*: フロー要素
- *Salesforce ヘルプ*: ゲストユーザーによるフローへのアクセスの許可
- *Salesforce ヘルプ*: Apex アクション
- *Salesforce ヘルプ*: トランザクションのフロー
- *Apex 開発者ガイド*: InvocableMethod アノテーション

共有なしのサンプルコード: レコードを作成して後で更新するためのアクセス権をゲストユーザーに付与

これらのコードサンプルでは、2 つの個別のインタラクションがサポートされています。最初のインタラクションで、ゲストユーザーはケースを作成します。セキュリティ上の理由により、Apex メソッドはレコード ID を暗号化された文字列に置き換えます。ゲストユーザーが後でケースをクローズする場合、暗号化された文字列を入力します。Apex メソッドは、文字列を復号化してレコード ID を取得します。次に、レコード ID を使用してケースを選択し、ケースの状況を更新します。

Aura コンポーネント: CreateCase.cmp

このサンプル Aura コンポーネントでは、レコードの作成およびクローズのためのコンポーネントが表示されます。

ケースを作成するために、ゲストユーザーはコンポーネントを使用してケースの詳細を入力します。レコードの作成後、`lightning:card` コンポーネントには、新規ケースの暗号化されたトークンや、トークンと一致するケースの状況が表示されます。

デモのために、このサンプルではゲストユーザーがケースのトークンを直接入力するコンポーネントを表示します。このシナリオを実装するには、トークンが含まれるリンクをゲストユーザーに提供し、URL からトークンを取得します。

```
<aura:component controller="GuestUserCreateForLater">
    <aura:attribute name="caseID" type="String"/>
    <aura:attribute name="case_status" type="String"/>
    <aura:attribute name="subject" type="String"/>
    <aura:attribute name="description" type="String"/>
    <aura:attribute name="email" type="String"/>

     Enter details to create a new case
    <lightning:input type="email" name="email" required="true" value="{!v.email}"
aura:id="email" label="Where should we send email updates?"/>
    <lightning:input name="subject" label="Subject" required="true" value="{!v.subject}"
aura:id="subject"/>
    <lightning:textarea name="description" required="true" label="Description"
```

```
value="{!v.description}" aura:id="description"/>
    <lightning:button name="submit" variant="brand" label="Create case" title="Create case"
 onclick="{!c.submitCase}"/>

    <aura:if isTrue="{!v.caseID}">
        <lightning:card title="Case">
            <p class="slds-p-horizontal--small">
                New case created:
                <p>{!v.caseID}</p>
            </p>
        </lightning:card>
    </aura:if>

    Or enter an existing case token to close the case
    <lightning:textarea name="existing_case" required="false" label="Existing case token"
 aura:id="existing_case"/>
 <lightning:button name="submit" variant="brand" label="Close case" title="Close case"
onclick="{!c.updateCase}"/>
    <aura:if isTrue="{!v.case_status}">
        <lightning:card title="Case">
            <p class="slds-p-horizontal--small">
                Case status:
                <p>{!v.case_status}</p>
            </p>
        </lightning:card>
    </aura:if>
</aura:component>
```

コンポーネントコントローラー: CreateCaseController.js

このサンプル JavaScript コントローラーは、Aura コンポーネントのイベントを処理し、ヘルパーファイルのメソッドをコールします。

```
({
    submitCase : function(component, event, helper) {
        helper.makeCase(component, event, helper);
    },
    updateCase : function(component,event,helper){
        helper.updateCase(component,event,helper);
    }
})
```

JavaScript ヘルパー: CaseHelper.js

JavaScript ヘルパーには、次の 2 つのメソッドがあります。

`makeCase()`

`makeCase()` メソッドは、送信されたデータを使用してケースを作成する非同期要求を作成します。要求が完了すると、コールバックで新規ケースの一意のトークンが Aura コンポーネントで使用される変数の `caseID` 項目に保存されます。

updateCase()

`updateCase()` メソッドは、ゲストユーザーが入力したトークンを使用して、トークンと一致するケースを非同期に更新します。メソッドのコールバックで Apex メソッドからの応答がキャッシュされ、値が `case_status` 変数に保存されます。

```
({
    makeCase : function(component, event, helper) {
        var subject = component.find("subject").get("v.value");
        var description = component.find("description").get("v.value");
        var email = component.find("email").get("v.value");

        var action = component.get("c.CreateCase");
        action.setParams({
            "subject": subject,
            "description": description,
            "email": email
        });
        action.setCallback(this, function(response){
            component.set("v.caseID", response.getReturnValue());
        });
        $A.enqueueAction(action);
    },
    updateCase : function(component,event,helper){
        var case_token = component.find("existing_case").get("v.value");
        var action = component.get("c.UpdateCase");
        action.setParams({
            "token":case_token
        });
        action.setCallback(this, function(response){
            component.set("v.case_status", response.getReturnValue());
        });
        $A.enqueueAction(action);
    }
})
```

Apex コントローラー: GuestUserCreateForLater.cls

このサンプル Apex コントローラーは、ケースを作成および更新するコールを受信します。User Encryption Decryption AppExchange パッケージを使用して、データを暗号化および復号化します。

CreateCase()

`CreateCase()` Apex メソッドは、ゲストユーザーの入力を使用してケースを作成します。レコードの作成後、レコード ID、レコードの `CreatedDate` 項目、現在のタイムスタンプから暗号化された文字列を生成します。Apex は、新規ケースの ID を暗号化された文字列に置き換えます。

UpdateCase()

`UpdateCase()` メソッドは、提供された文字列を復号化して結果を検証し、その情報を使用して元のレコードの状況を更新します。応答は、レコードの状況またはエラーメッセージ (エラーが発生した場合) になります。

レコードに直接アクセスしないため、`with sharing` キーワードを使用してクラスを定義します。

警告: `@AuraEnabled` クラスは、インターネット上の任意のシステムまたは個人が呼び出すことができます。クエリで適切なレコードのみを更新できることを確認します。

```
public with sharing class GuestUserCreateForLater {

    @AuraEnabled
    public static String CreateCase(String subject,
                                        String description,
                                        String email){
  Case new_case = new Case(Subject=subject,
                                Description=description,
                                SuppliedEmail=email);
        insert new_case;

       List<Case> results = GuestUserCaseHelperWS.getCase(new_case.Id);


        String encryptedID = ued.UserCryptoHelper.doEncrypt(results[0].Id+'|'+
results[0].CreatedDate.getTime() +'|'+System.DateTime.now().getTime());
        return encryptedID;
    }

    public static final Long validTimestampMinutes = 10;

    @AuraEnabled
    public static String UpdateCase(String token){
        String status = 'Case not found';
        String decrypted_token = '';
        try {
            decrypted_token = ued.UserCryptoHelper.doDecrypt(token);
        } catch(Exception e) {
            return status;
        }
        String[] decrypted_parts = decrypted_token.split('\\|');
        String decryptedRecordId = decrypted_parts[0];
        String created_timestamp = decrypted_parts[1];
        String original_request_timestamp = decrypted_parts[2];


        if( isTimestampValid(System.Long.valueOf(original_request_timestamp))) {

         List<Case> caseList = GuestUserCaseHelperWS.getCase(decryptedRecordId,
created_timestamp);
            if(caseList.size() == 1){
            Case case_to_update = caseList[0];
                        case_to_update.Status = 'Closed';
                try {
                         GuestUserCaseHelperWS.updateCase(case_to_update);
                         status = 'Closed';
                } catch(DmlException e){
                      System.debug('An unexpected error has occurred: ' + e.getMessage());

                }
            }else{
```

```
                status = 'Case not found';
            }
        }
        return status;
    }

    private static Boolean isTimestampValid(Long timestamp)
    {
        return ((System.now().getTime() - timestamp) / 60000) < validTimestampMinutes;
    }
}
```

Apex ヘルパークラス: GuestUserCaseHelperWS.apxc

このサンプル Apex ヘルパークラスは、ID でレコードを取得してレコードを更新するメソッドを定義します。Apex コントローラーはこのメソッドをコールします。

共有を使用せずにレコードを取得および更新できるように、`without sharing` キーワードを使用してクラスを定義します。

```
public without sharing class GuestUserCaseHelperWS {

    public static List<Case> getCase(String caseID, Datetime created_date)
    {
        List<Case> results = [SELECT Case.CaseNumber, Case.CreatedDate, Case.Status
        FROM Case
        WHERE Case.Id=:caseIDAND Case.CreatedDate=:created_date];
        return results;
    }

    public static Case updateCase(Case case_to_update)
    {
        update case_to_update;
        return case_to_update;
    }
}
```

関連トピック:

Salesforce ヘルプ: ゲストユーザーの共有設定とレコードアクセスの保護

Apex 開発者ガイド

Lightning Aura コンポーネント開発者ガイド

Apex クラスへのアクセスの制限

ゲストユーザーや外部ユーザーのアクセスをコールする必要のあるクラスに制限します。

Apex クラスに公開されているメソッド (`@InvocableMethod`、`@AuraEnabled`、`@RestResource`、`webservice` などを使用するメソッド) が含まれている場合、ゲストユーザーや外部ユーザーは任意のパラメーターでこれらのメソッドを呼び出すことができます。ただし、Apex クラスを実行するための権限が必要です。Apex クラ

スへのアクセスを特定の権限セットまたはプロファイルを持つユーザーに制限することをお勧めします。ゲストユーザーや外部ユーザーにApexクラスへのフルアクセス権を付与することは安全ではありません。どのユーザーがどのApexクラスをコールする必要があるのかについて十分に考慮し、それらのロールの権限セットを作成して、必要な権限セットでApexクラスを有効にしてください。

フローセキュリティ

ゲストユーザーや外部ユーザーがフローを実行する必要がある場合、ユーザーがすべてのフローを実行できるようにせずに、フロー権限を上書きして特定の外部ユーザープロファイル、権限セット、サイトゲストユーザープロファイルにのみアクセス権を付与します。可能な場合はシステムコンテキストでフローが実行されないようにして、サブフローへのアクセスを制限します。そのようにしない場合は、それらのフローおよびサブフローの手続き型アクセスコントロールを実装します。

フローは、オブジェクトおよびApexクラスへのアクセスに対するプラットフォームセキュリティ設定を上書きできる強力な機能です。フローを使用して、権限セットを有効化および無効化できます。画面フローは、ユーザー制御の入力パラメーターを使用してブラウザーで制御されます。そのため、「フローを実行」権限を上書きし、ゲストユーザーや外部ユーザーのプロファイルまたは権限セットに基づいてアクセス権を特定のフローに割り当てることをお勧めします。ゲストユーザーの場合、適切なサイトのゲストユーザープロファイルでフローアクセスポリシーを設定します。

また、ユーザーが個別にサブフローを実行する場合でもサブフローの実行権限を削除することは優れたセキュリティプラクティスになります。セキュリティの観点から、2つの個別のフローを作成し、サブフローとして実行されるフローではなく、ユーザーが直接実行するフローにのみアクセス権を付与することをお勧めします。サブフローではなく、最上位レベルの親フローにのみフローアクセス権を付与します。フローによってコールされる呼び出し可能なApexメソッドにも同様の推奨事項が適用されます。それらのメソッドがコールされるべきフローでのみコールされるように、それらのクラスへのアクセス権をユーザーに付与しないようにします。

ユーザーに画面フローを実行するための権限が付与されている場合、次の操作を実行できます。

- 選択したパラメーターを使用して任意のタイミングでフローを呼び出す。
- 任意のタイミングでフローをキャンセルする。

これらの考慮事項は、サブフローや、他のフローからコールされるフローにも適用されます。

フローユーザーが実行できる具体的な操作は次のとおりです。

- 画面フローの入力(開始)変数を表示および変更する。
- 画面サブフローから親フローに返される出力変数を表示する。
- サブフローの入力変数を変更する(サブフローを実行する権限がある場合)。

これらのいずれかの機能がセキュリティポリシーに違反する場合、サブフローを使用しないでください。たとえば、機密にしておく必要がある請求情報やその他の機密情報がサブフローで処理される場合、メインフローでビジネスロジックを保持します。

SOQLインジェクション

動的SOQLクエリに渡されるユーザー制御データをサニタイズします。

SOQL または SOSL インジェクションは、入力されたユーザー制御データが Apex コードで適切にサニタイズされずに動的 SOQL または SOQL クエリに挿入されると発生します。次の 2 つのシナリオを考えます。

- クエリの全体的な構造の変更
- クエリパラメーターの値の変更

コール元が権限のないデータにアクセスできないようにする Apex コードを呼び出して、これらのシナリオを制御します。

次のコードを考えます。

```
@AuraEnabled
public static List<Account> getAccountName(string userId) {
  if (FeatureManagement.checkPermission('readAccount')) {
      string query='SELECT Name FROM Account WHERE Id=\''+ userId + '\'';
      return database.query(query);
  }
}
```

ユーザーはクエリを制御して、権限を超える情報にアクセスできる可能性があります。

ユーザーがアクセスできる情報は、クエリの戻り値で制限されません。ユーザーは次のような文字列を送信できます。

```
userId = '0035Y00003pPJiNQAW\' OR AnnualRevenue>100000.00 OR Name=\'a'
// 0035Y00003pPJiNQAW is any id to any object that is not an account
```

クエリの戻り値として年間売上が $100,000 を超えるすべての取引先が返されます。これは、開発者が意図したコール元に返される内容ではありません。

SOQL または SOSL インジェクションを修正するには、コンテキストに応じた適切なエンコードが必要になります。`String.escapeSingleQuotes` をすべてのユーザー入力に適用するのではなく、SOQL または SOSL クエリのどこにあるのかに基づいて項目をサニタイズします。

- WHERE (SOQL)、ORDER BY (SOQL)、WITH (SOSL)、FIND (SOSL) 句の変数の場合、束縛変数を使用します。

  ```
  string query='SELECT Name FROM Account WHERE Id=:userId';
  ```

- 項目名やテーブル名の場合、項目の `describeResult` で `isAccessible` をコールします。また、宣言型ポリシーを適用するには、独自の手続き型ロジックを使用して、項目名やテーブル名をセキュリティポリシーで許可されている名前に制限します。
- 引用符で囲まれた文字列のパラメーターの場合、バインド変数を使用します。バインド変数または `String.escapeSingleQuote` を使用して、引用符で囲まれたコンテキストにないテーブル名、項目名、パラメーターをサニタイズしないでください。
- 他のプリミティブ型の場合、ユーザー入力をブール型、整数、ID、または他のプリミティブ (非文字列) 型にキャストします。

WITH SECURITY_ENFORCED キーワードでは SELECT および FROM 句のみがサニタイズされ、WHERE 句はサニタイズされないため、このキーワードは SOQL または SOSL インジェクション攻撃のサニタイザーにはなりません。

第 5 章 セキュアサイトの開発: CSP、LWS、および Lightning Locker

トピック:

Experience Cloud の Aura および LWR サイトでは、コンテンツセキュリティポリシー (CSP) と Lightning Web セキュリティ (LWS) または Lightning Locker のいずれかを使用して、悪意のある攻撃やカスタムコードの脆弱性からサイトを保護します。独自のカスタムコンポーネントを開発したり、サードパーティコンポーネントを使用したり、カスタムコードを `head` マークアップに追加したりする場合、これらのセキュリティ機能の潜在的な影響を考慮します。

CSP

CSP は、ページに読み込むコンテンツのソースを制御するための W3C 標準です。CSP ルールはページレベルで機能し、すべてのサードパーティコンポーネントとカスタムコードに適用されます。デフォルトでは、フレームワークのヘッダーに読み込むことができるコンテンツは安全な (HTTPS) URL からのみで、JavaScript からの XHR 要求は禁止されています。

エクスペリエンスビルダーでは、さまざまなレベルの CSP スクリプトセキュリティを使用できます。CSP レベルはサイトごとに固有です。

Lightning Locker と Lightning Web セキュリティ

Lightning Locker アーキテクチャレイヤーでは、個々の Lightning コンポーネントの名前空間が各自のコンテナで分離され、コーディングのベストプラクティスが適用されるため、セキュリティが向上します。Lightning Locker は、Lightning コンポーネントおよび Experience Cloud の Aura サイトのデフォルトのセキュリティアーキテクチャです。

LWS は、Lightning コンポーネントでセキュリティ保護のコーディング手法を簡単に使用できるように設計されており、Lightning Locker を置き換えることを目的としています。LWS の目的は、Lightning Locker と同様に、Lightning コンポーネントが他の名前空間のプラットフォームコードやコンポーネントに属するデータに干渉したり、アクセスしたりするのを防ぐことです。ただし、Lightning Web セキュリティのアーキテクチャは、異なる手法で Lightning Web コンポーネントを保護します。

組織およびサイトレベルでの LWS の適用方法

システム管理者は、LWS を組織レベルで有効化して、Lightning Locker の代わりに組織全体で使用することができます。LWS は、[設定] の [セッションの設定] の [Lightning Web コンポーネント用および Aura コンポーネント用 Lightning Web セキュリティの使用] 設定で有効化します。

この組織レベルの設定は Aura サイトに影響します。これは LWS が組織で有効になると、サイトレベルで LWS が Lightning Locker に置き換わるためです。次に、エクスペリエンスビルダーで Aura サイトの Lightning Locker 設定を無効にした場合、実際には LWS が無効になります。

LWR サイトには独自の LWS インスタンスがあるため、LWS の組織設定は LWR サイトには影響しません。LWR サイトで Lightning Locker を無効にすると、LWS が組織で有効になっていても、サイトの LWS インスタンスが無効になります。

メモ: デフォルトでは、すべての新しいエクスペリエンスビルダーサイトで厳格な CSP が有効になります。これは Lightning Locker または LWS も有効であることを意味します。エクスペリエンスビルダーで Lightning Locker 設定にアクセスするには、[緩和された CSP] を選択します。

Commerce Cloud の B2B ストアと B2C ストアの LWR テンプレートでは、LWS はデフォルトで有効化されません。

次の表では、Aura または LWR サイトでの組織レベルの設定とサイトレベルの設定の影響をまとめています。

Experience Cloud サイトのフレームワーク	サイトレベルの設定	組織レベルの設定	**LWS** と **Locker** のどちらがサイトで使用されるか
Aura	✗	✗	✗
	✗	✓	✗
	✓	✗	**Lightning Locker**
	✓	✓	**LWS**
LWR	✗	✗	✗
	✗	✓	✗
	✓	✗	**LWS** (サイトのインスタンス)

Experience Cloud サイトのフレームワーク	サイトレベルの設定	組織レベルの設定	LWS と Locker のどちらがサイトで使用されるか
	✔	✔	LWS (サイトのインスタンス)

関連トピック:

- *Salesforce* ヘルプ: エクスペリエンスビルダーサイトの CSP および Lightning Locker
- *Salesforce* ヘルプ: エクスペリエンスビルダーサイトでのセキュリティレベルの選択
- *Salesforce* ヘルプ: CSP および Lightning Locker の設計に関する考慮事項
- *Lightning Web* コンポーネント開発者ガイド: Lightning Locker のセキュリティ
- *Lightning Aura* コンポーネント開発者ガイド: セキュアなコードの開発

Aura サイトでの Lightning Locker の競合の解決

Lightning Locker は、すべての新規 Aura サイトで、デフォルトで有効化されます。ただし、ページのサードパーティコンポーネントや `head` マークアップのカスタムコードが Lightning Locker との競合のために期待どおりに動作しないことがあります。そのような場合、ここに記載されているいずれかの回避策を使用することをお勧めします。

JavaScript カスタムイベントの使用

Lightning Locker は、`head` マークアップではなく、他の名前空間のリソースとのやりとりからサードパーティコンポーネントやカスタムコードを保護します。この制限は、Lightning Locker をバイパスしてセキュリティに脆弱性をもたらすカスタムコードを `head` マークアップに含めることができることを意味します。

この制限に対応するには、`head` マークアップで `CustomEvent` コンストラクターを使用して、サードパーティ Aura コンポーネント、Lightning Web コンポーネント、カスタムコードを分離します。これにより、サードパーティコンポーネントやカスタムコードは、そのリソースを直接読み込んだり参照したりせずにリソースとやりとりできます。

イベントを介してリスナーに渡される必要があるデータは、イベントの初期化時に作成される `detail` プロパティで渡されます。`detail` プロパティは、`head` マークアップリスナーの dataLayer に対応付けられます。これにより、`EventTarget` を拡張する任意のリソースにカスタムイベントがディスパッチされます。カスタムイベントの使用例については、「Adobe Analytics および Lightning Locker」を参照してください。

警告: JavaScript `CustomEvent` コンストラクターで渡すデータに注意して、使用方法が安全であることを確認します。ページで実行される JavaScript (使用しているサードパーティ App Exchange コンポーネントなど) でイベント名がリスンされて、このデータが読み取られる可能性があります。

API 39.0 への Aura コンポーネントの設定

サードパーティコンポーネントやカスタムコードと Aura コンポーネントのやりとりが期待どおりに行われない場合、Aura コンポーネントを Salesforce API バージョン 39.0 に設定します。これにより、コンポーネントの Lightning Locker が無効になります。『*Lightning Aura コンポーネント開発者ガイド*』の「コンポーネントの Lightning Locker の無効化」を参照してください。

警告: Aura コンポーネントの Lightning Locker を無効にすると、サイトのセキュリティが低下し、設計時にコンポーネントを使用したり、実行時に表示したりできなくなる可能性があります。

一貫性を保ちデバッグを容易にするため、親 Aura コンポーネントと子コンポーネントの API バージョンが同じになるようにします。そのため、コンポーネント階層内で使用する Aura コンポーネント (コンポーネント内のコンポーネントや、別のコンポーネントを拡張するコンポーネントなど) を API バージョン 39.0 に設定しないでください。

組織で LWS が有効な場合、コンポーネントで API バージョン 39.0 を設定しても、そのコンポーネントに対して LWS は無効化されません。ただし、Lightning Locker でブロックされるコンポーネントの動作も、LWS では許可される可能性が高いため、無効化する必要性がなくなります。

Lightning Locker の無効化

警告: この回避策は、最後の手段としてのみ使用してください。

サイトで Lightning Locker を無効にすると、すべてのサイトのサードパーティコンポーネントとカスタムコードで無効になります。この影響は、広範囲で予想外の結果になる可能性があります (サイトのセキュリティの低下など)。サードパーティコンポーネントが Lightning Locker なしで動作するように設定されていない場合、そのコンポーネントは設計時に使用したり、実行時に表示したりできない可能性があります。Lightning Locker が無効になっている場合、異なる名前空間のコンポーネントがやりとりしたり、お互いのドキュメントオブジェクトモデル (DOM) にアクセスしたり、サイトとやりとりするカスタムリソースに関する制限が緩和されたりします。

Lightning Locker の無効化についての詳細は、Salesforce ヘルプの「エクスペリエンスビルダーサイトでのセキュリティレベルの選択」を参照してください。

メモ: 組織で LWS が有効になっている場合、Aura サイトで Lightning Locker を無効にすると、実際にはサイト内の LWS が無効になります。LWR サイトで Lightning Locker を無効にすると、LWS が組織で有効になっていても、サイトの LWS インスタンスが無効になります。

関連トピック:

- エクスペリエンスビルダーサイトの ExperienceBundle
- *Experience Cloud* ヘルプ: エクスペリエンスビルダーサイトでのセキュリティレベルの選択
- *Salesforce* ヘルプ: ページの `<head>` へのマークアップの追加によるエクスペリエンスビルダーサイトのカスタマイズ
- *Lightning Web コンポーネント開発者ガイド*: イベントを使用した通信
- *Lightning Aura コンポーネント開発者ガイド*: イベントとの通信

Lightning Locker が無効になっている場合のサードパーティコンポーネントの実行

エクスペリエンスビルダーサイトで Lightning Locker を無効にする場合、管理パッケージからインストールされたサードパーティコンポーネントが設計時に使用できて実行時に表示されるように設定する必要があります。

[緩和された CSP] セキュリティレベルから Lightning Locker を無効にできます。

組織で LWS が有効になっている場合、Aura サイトで Lightning Locker を無効にすると、実際にはサイト内の LWS が無効になります。LWR サイトで Lightning Locker を無効にすると、LWS が組織で有効になっていても、サイトの LWS インスタンスが無効になります。LWR サイトでは、Lightning Locker や LWS を無効にすることなく、サードパーティライブラリを組み込むことができます。詳細は、『*LWR Sites for Experience Cloud (Experience Cloud 向け LWR サイト)*』の「Integrate Third-Party Libraries Using the Privileged Script Tag (特権 Script タグを使用したサードパーティライブラリの統合)」を参照してください。

警告: Lightning Locker を無効にすると、サイトのセキュリティが低下する可能性があります。Lightning Locker の無効化は、最後の手段としてのみ行ってください。

Lightning Locker なしにサードパーティ Aura コンポーネントを実行するための設定

サードパーティ Aura コンポーネントの場合、管理パッケージ開発者は、コンポーネントの `lightningcommunity:allowInRelaxedCSP` インターフェースを設定する必要があります。

Locker なしにサードパーティ Lightning Web コンポーネントを実行するための設定

サードパーティ Lightning Web コンポーネントの場合、管理パッケージ開発者は、コンポーネントの設定ファイルの `capability` タグの `lightningCommunity__RelaxedCSP` 値を設定する必要があります。

関連トピック:

- エクスペリエンスビルダーサイトの ExperienceBundle
- *Experience Cloud* ヘルプ: エクスペリエンスビルダーサイトでのセキュリティレベルの選択
- *Salesforce* ヘルプ: エクスペリエンスビルダーサイトのサードパーティホストを許可リストに追加する場所
- *Lightning Web Component Reference (Lightning Web* コンポーネントリファレンス*)*: Allow In Relaxed Csp
- *Lightning Web* コンポーネント開発者ガイド: XML 設定ファイルの要素

例: Aura サイトの Adobe Analytics および Lightning Locker

Adobe Analytics は、Aura サイトのコンポーネントとやりとりするため、Lightning Locker で予期しない結果が生じる可能性があります。推奨される回避策は、`head` マークアップで JavaScript カスタムイベントを使用して Adobe Analytics を分離することです。これにより、Adobe Analytics は、そのリソースを直接読み込んだり参照したりせずにコンポーネントとやりとりできます。

ヒント: LWR サイトでは、別の方法を用いることで、アナリティクスを組み込むことができます。詳細は、『*LWR Sites for Experience Cloud (Experience Cloud* 向け *LWR* サイト*)*』の「Integrate Third-Party Libraries Using the Privileged Script Tag (特権 Script タグを使用したサードパーティライブラリの統合)」を参照してください。

Aura サイトへの Adobe Analytics の追加

`script` タグを使用して、Adobe Analytics スクリプトおよび該当するイベントリスナーをサイトの `head` マークアップに追加します。

```
<script>
    document.addEventListener('analyticsEvent', function(e) {
        //add logic here to tell your dataLayer about the event
        //dataLayer.action = e.detail.action;
        //dataLayer.label = e.detail.label;
        //or map payload to an AA library event
    });
```

```
    document.addEventListener('analyticsViewChange', function() {
    });
</script>
<script src="full-url-to-your-adobe-script" async></script>
```

カスタムイベントの使用

Adobe Analytics とやりとりするコンポーネントで、`detail` プロパティを使用してカスタムイベントを実装します。このプロパティにより、データがイベントを介してリスナーに渡されて、`head` マークアップリスナーの `dataLayer` に対応付けられます。これにより、`EventTarget` を拡張する任意のリソースにカスタムイベントをディスパッチできます。

```
document.dispatchEvent(new CustomEvent('analyticsEvent', {'detail': {action: 'click',
label: 'Submitted Case'}}));
```

警告: JavaScript `CustomEvent` コンストラクターで渡すデータに注意して、使用方法が安全であることを確認します。ページで実行される JavaScript (Adobe Analytics など) でイベント名がリスンされて、このデータが読み取られる可能性があります。

Aura コンポーネントの追加イベントの実装

Adobe Analytics が Aura コンポーネントとやりとりする場合、`forceCommunity:routeChange` および `aura:locationChange` イベントも実装する必要があります。

`forceCommunity:routeChange` は、Lightning コンポーネントフレームワーク内のビューの変更を追跡します。

```
<aura:component implements="forceCommunity:availableForAllPageTypes">
    <aura:handler event="forceCommunity:routeChange" action="{!c.handleRouteChange}" />
</aura:component>
```

```
handleRouteChange : function(component, event, helper) {
    document.dispatchEvent(new Event('analyticsViewChange'));
}
```

`aura:locationChange` は、ブラウザーのロケーションバーの URL のハッシュ部分が変更されたことを示します。ただし、場所 URL のハッシュ部分は、タブコンポーネントのタブの変更を実装するためなどに使用されますが、ほとんど使用されることはありません。

関連トピック:

- *Salesforce ヘルプ*: ページの `<head>` へのマークアップの追加によるエクスペリエンスビルダーサイトのカスタマイズ
- *Lightning Web Component Reference (Lightning Web コンポーネントリファレンス)*: Route Change
- *Lightning Web Component Reference (Lightning Web コンポーネントリファレンス)*: Location Change
- *Lightning Aura コンポーネント開発者ガイド*:Lightning コンポーネントフレームワークとは?

第 6 章　エクスペリエンスビルダーサイトパフォーマンスの分析と向上

Salesforce Page Optimizer は、サイトを分析してパフォーマンスに影響する問題を識別します。この情報を使用して設計を調整し、メンバーのサイトパフォーマンスを向上させます。コミュニティページオプティマイザーは、Chrome ウェブストアから入手できる無料のプラグインです。他の Chrome 拡張機能と同様にこのプラグインをダウンロードしてインストールします。

Page Optimizer をダウンロードするには、エクスペリエンスビルダーで、左サイドバーの  をクリックし、[詳細] をクリックします。

エディション

使用可能なインターフェース: Salesforce Classic (使用できない組織もあります) および Lightning Experience の両方

使用可能なエディション: **Enterprise** Edition、**Performance** Edition、**Unlimited** Edition、および **Developer** Edition

ユーザ権限

Experience Cloud サイトをカスタマイズする

- サイトのメンバーであること、「エクスペリエンスの作成および設定」

  または

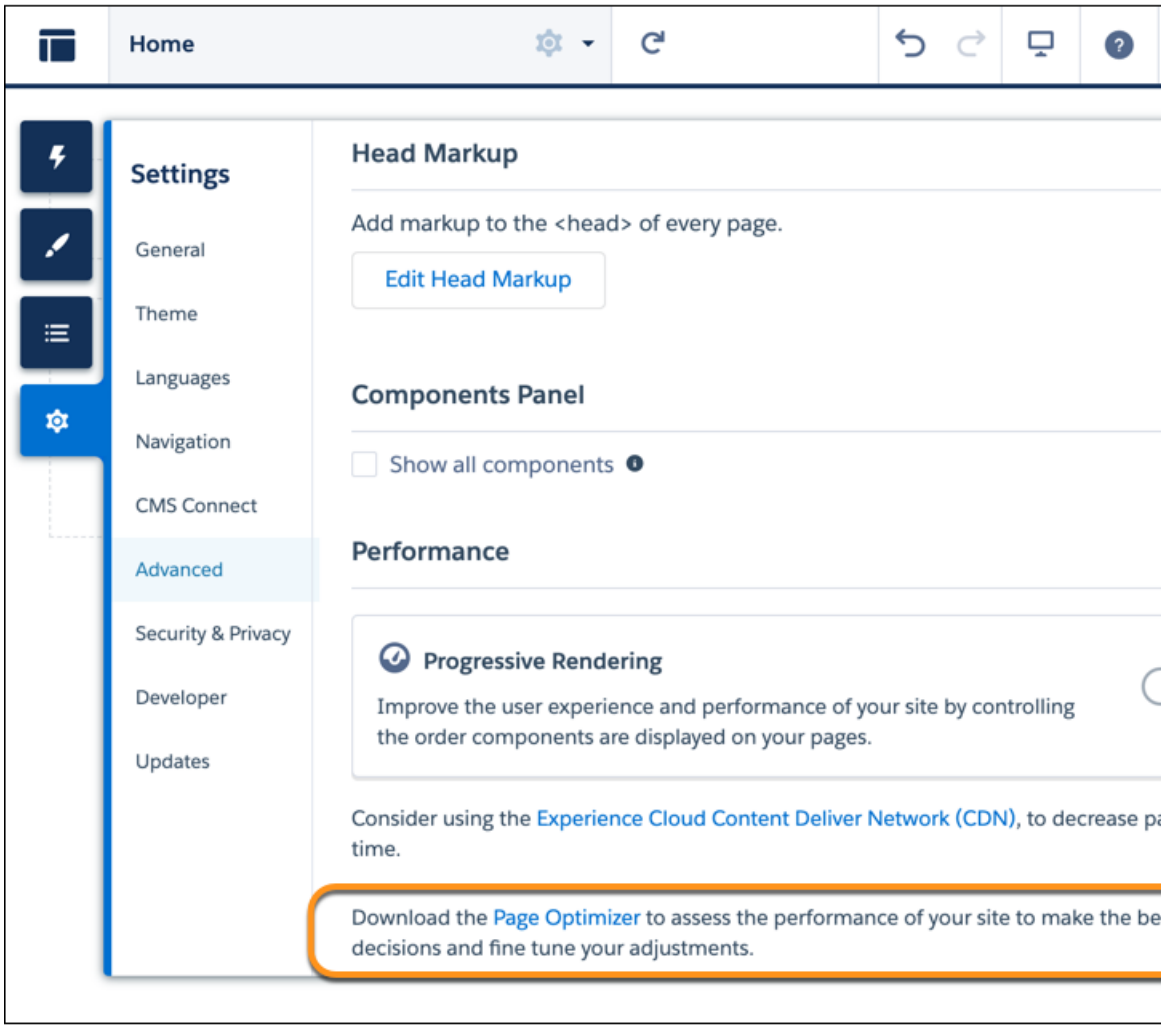
- サイトのメンバーであること、「設定・定義を参照する」、およびそのサイトのエクスペリエンス管理者、公開者、または作成者であること

Experience Cloud サイトを公開する

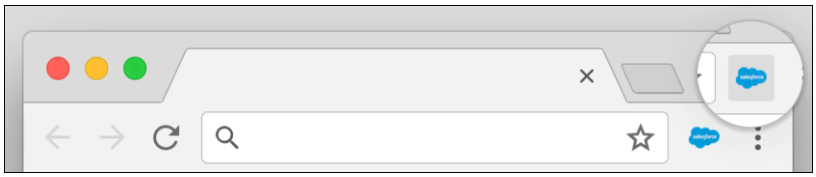
- サイトのメンバーであること、「エクスペリエンスの作成および設定」

  または

- サイトのメンバーであること、そのサイトのエクスペリエンス管理者または公開者であること

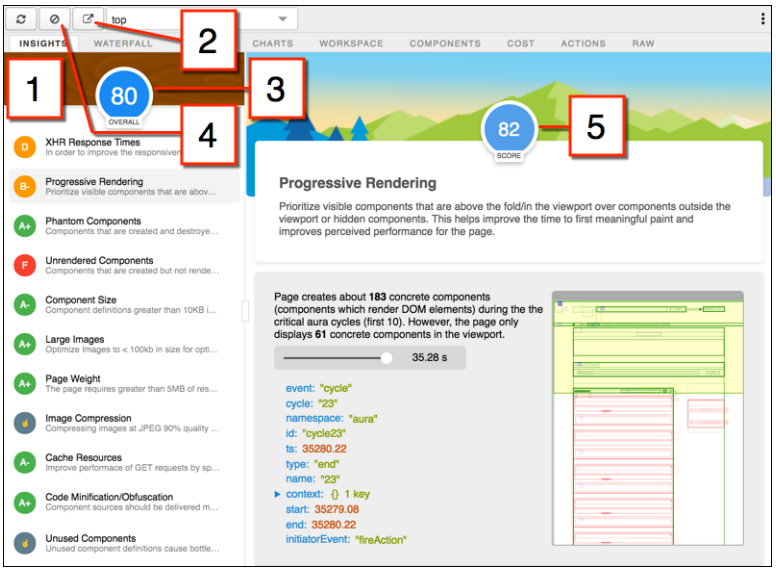


インストールされた Page Optimizer は、他の Chrome 拡張機能と同じ場所にあります。



Insights (インサイト)

サイトを分析するには、公開サイトに移動し、ページを読み込んでから、PageOptimizer を起動します。



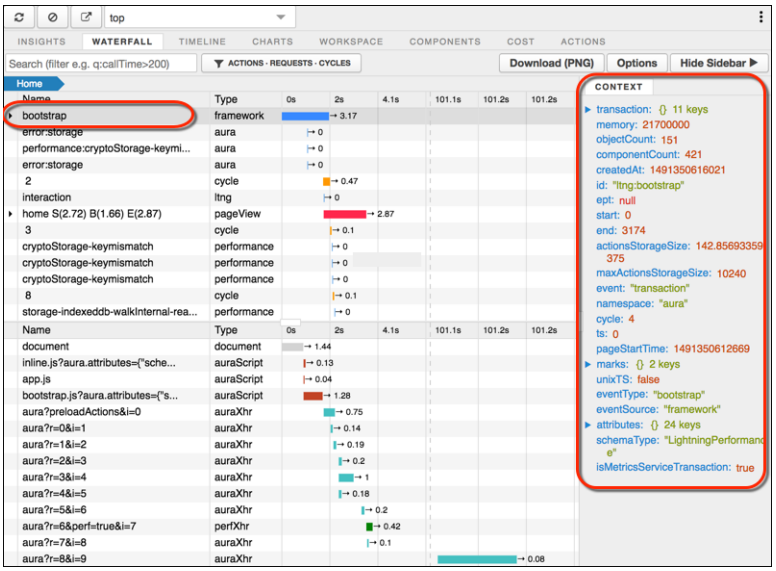
[Insights (インサイト)] タブ (1) は、Lightning フレームワークを使用して開発された Web アプリケーションのベストプラクティスに基づき、ページを評価します。このタブには、さまざまな分析ルールでの全体的なパフォーマンススコア (3) と個々のスコア (5) が表示されます。詳細と推奨アクションを表示するには、各ルールをクリックします。作業領域を広げるには、ポップアウト (2) をクリックします。

[Insights (インサイト)] タブには、具体的な推奨事項は表示されません。詳しいインサイトは、[Waterfall (ウォーターフォール)]、[Timeline (タイムライン)]、[Charts (グラフ)]、[Cost (コスト)]、[Actions (アクション)] タブに表示される未加工データを確認してください。

収集した総計値を削除するには、[クリア] (4) をクリックします。ページでいくつかのユーザーアクションを実行して新しい総計値を収集してから、Page Optimizer を開き直します。たとえば、フィード項目のいいね!に関するパフォーマンス総計値を収集するには、パフォーマンス総計値をクリアして、[Like (いいね!)] をクリックし、Page Optimizer を開き直します。

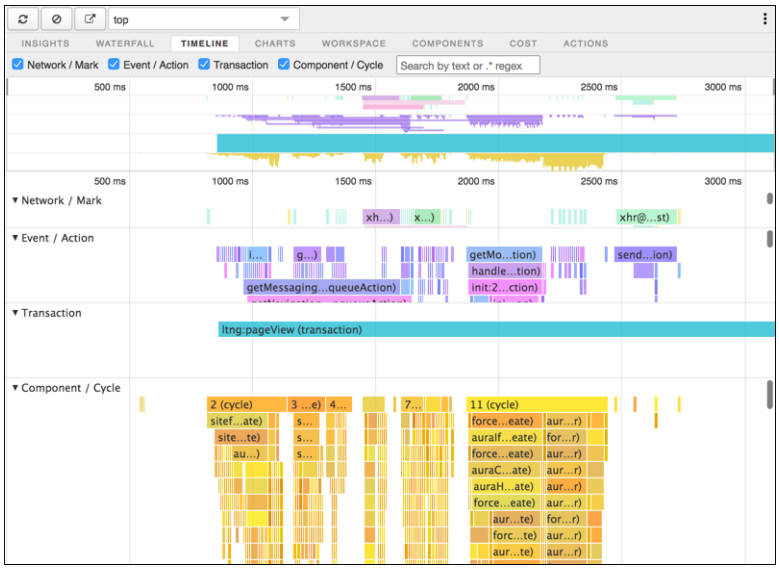
Waterfall (滝)

[Waterfall (滝)] タブには、すべてのネットワーク要求とパフォーマンス計測データが表示されます。行をクリックすると、サイドバーにコンテキスト情報が表示されます。各行の左にある矢印をクリックすると、各行の情報が展開されます。



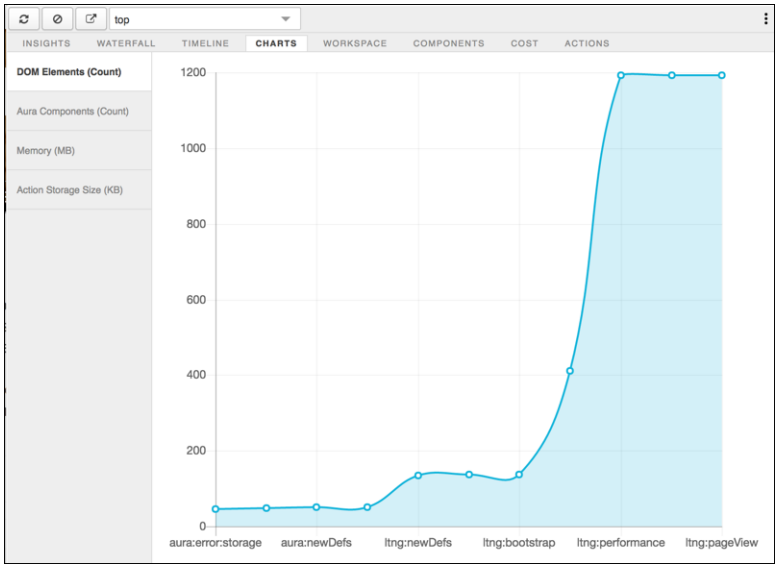
Timeline (タイムライン)

[Timeline (タイムライン)] タブには、各コンポーネントの表示ライフサイクルのプロファイルが表示されます。このタイムラインビューは Lightning フレームワーク総計値の表示用に最適化されているため、Chrome デベロッパーツールよりもわかりやすくなっています。



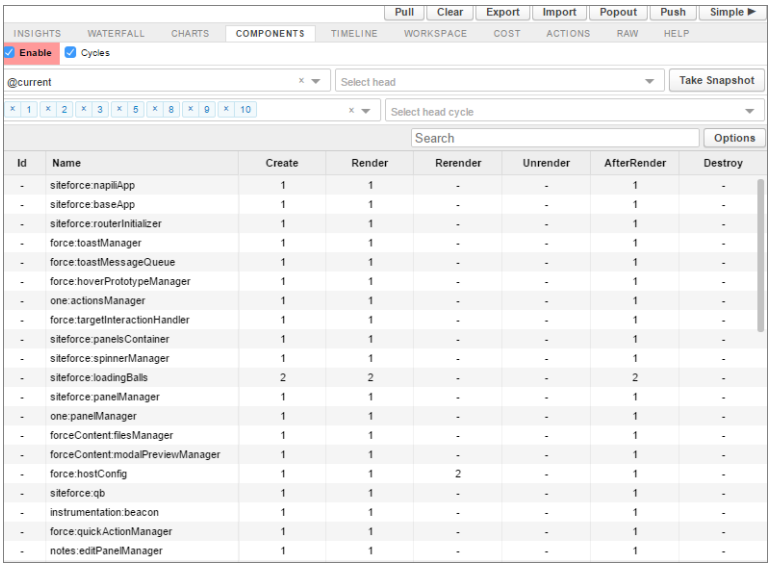
Charts (グラフ)

[Charts (グラフ)] タブには、顧客がページを使用したときのメモリとコンポーネントに関するトレンド情報が表示されます。



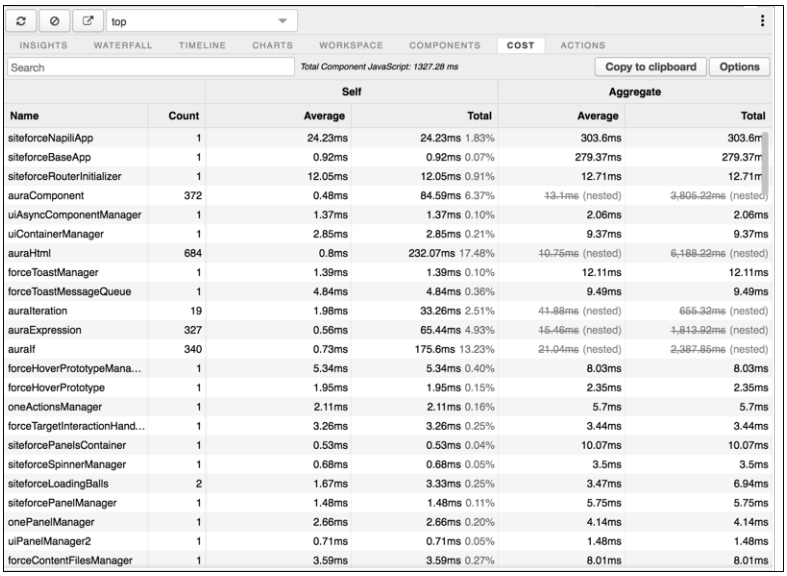
Components (コンポーネント)

[Components (コンポーネント)] タブには、ページ内の各コンポーネントのライフサイクルの計数が表示されます。このビューは、潜在的なコンポーネントの漏洩と予期しない表示動作の識別に役立ちます。コンポーネントのパフォーマンスを全体的に確認するには、[Components (コンポーネント)] タブを [Cost (コスト)] タブと併用します。



Cost (コスト)

[Cost (コスト)] タブには、各コンポーネントがロジックの処理にかかりきりだった時間が表示されます。この時間が短いほど、パフォーマンスが優れています。

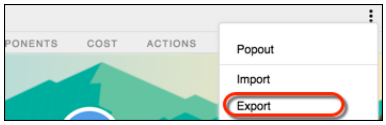


Actions (アクション)

[Actions (アクション)] タブには、ページで実行されたすべてのアクションとそのタイミング情報のリストが表示されます。

エクスポート

分析をファイルにエクスポートし、開発チームやサポートチームと共有できます。



フィードバックの送信

皆様から意見をお待ちしております。コメント、質問、要求、および見つかった問題を共有してください。フィードバックの送信。

関連トピック:

Salesforce 開発者ブログ: Lightning Components Performance Best Practices (Lightning コンポーネントのパフォーマンスのベストプラクティス)

第 7 章　エクスペリエンスビルダーサイトへの Pardot トラッキングの追加

訪問者がまだプロスペクトに変換されていなくても、Pardot では、サイトでの訪問者との対話と活動を追跡できます。トラッキングが有効化されたら、Pardot を使用して訪問者のエンゲージメントに関するレポートを表示し、サイトの活動に基づいてリードを自動的にスコアリングします。

1. Pardot で、追跡するキャンペーンに移動します。
2. [トラッキングコードを表示] をクリックしてコードをコピーします。
3. トラッキングの追加先のサイトのエクスペリエンスビルダーにアクセスします。
4. [設定] > [詳細] で [ヘッドマークアップを編集] をクリックし、Pardot トラッキングコードを貼り付けます。

エクスペリエンスビルダーサイトは単一ページアプリケーション (SPA) のため、ユーザーがサイトの別のページに移動すると、ページ全体ではなくコンテンツ領域のみが再読み込みされます。Pardot スクリプトでは、サイトの最初の読み込みがページビューとして記録されます。ページ内のアプリケーション内ナビゲーションを Pardot 内でより正確に取得できるようにスクリプトを変更します。

1. [ヘッドマークアップを編集] ウィンドウで Pardot トラッキングコードを変更し、ページ状態の変更がセッション履歴に追加されるようにします。

   変更内容が記述されたサンプルコードスニペットを次に示します。

```
<script type='text/javascript'>
piAId = '{{%pardot-id-for-your-org%}}'; //no change from OOTB code
 (format: 123456)
piCId = '';
piHostname = '{{%pardot-hostname-for-your-org%}}'; //no change
from OOTB code (format: www.yourpardottrackerdomain.com)

(function() {
  //patching the history push state function to include calling
  // the async_load function that sends data to Pardot
  var pushState = history.pushState;
  history.pushState = function() {
    pushState.apply(history, arguments);
    async_load();
  };

    function async_load(){0
        var s = document.createElement('script'); s.type =
```

```
'text/javascript';
       s.src = ('https:' == document.location.protocol ? 'https://'
 : 'http://') + piHostname + '/pd.js';
        var c = document.getElementsByTagName('script')[0];
c.parentNode.insertBefore(s, c);
    }
    if(window.attachEvent)
    {
        window.attachEvent('onload', async_load);
         //attach event listener for browser history changes
         // for browsers that support attachEvent
         window.attachEvent('onpopstate', async_load);
    }
    else
    {
        window.addEventListener('load', async_load, false);
        //add eventlistener for browser history changes
        // for all other browsers
        window.addEventListener('popstate', async_load, false);
    }
})();


</script>
```

ヘッドマークアップが予期したとおりに動作するようにサイトのコンテンツセキュリティポリシー (CSP) 設定を更新します。

1. エクスペリエンスビルダー > [設定] > [セキュリティ] にアクセスします。
2. [緩和された CSP: インラインスクリプトと許可されたホストへのアクセスを許可] を選択し、確認ウィンドウで [許可] をクリックします。
3. [CSP エラー] の下に、ブロックされたサイトのリストが表示されます。Pardot トラッカードメインとして許可する各サイトで [URL を許可] をクリックします。

Pardot と Experience Cloud 間の正常なインテグレーションが有効化されたら、ページビューを正確に追跡し、訪問者の Experience Cloud サイトナビゲーションに基づいてリードをスコアリングできます。

関連トピック:

- *Salesforce* ヘルプ: トラッキングコードの実装
- *Salesforce* ヘルプ: ページの <head> へのマークアップの追加によるエクスペリエンスビルダーサイトのカスタマイズ
- *Salesforce* ヘルプ: エクスペリエンスビルダーサイトのサードパーティホストを許可リストに追加する場所

第 8 章　エクスペリエンスビルダーサイトでの CMS の使用

コンテンツ管理システム (CMS) は、コンテンツを複製することなく再利用できる機能を提供します。CMS を使用すると、コンテンツを複数のサイトにフィードできるだけでなく、一元的に更新して一度にすべての場所でコンテンツを最新の状態にすることができます。

Experience Cloud には、ニーズに応じて 2 つの CMS オプションがあります。Salesforce CMS は、組織の複数のチャネルを使用してコンテンツを作成、管理、整理するために組織に組み込まれています。CMS Connect は、サードパーティ CMS のコンテンツをサイトに組み込むためのツールです。これらのオプションについての詳細は、『CMS Developer Guide (CMS 開発者ガイド)』を参照してください。

第 9 章 デフレクションの報告: デフレクションシグナルフレームワーク

トピック:

ユーザーがケースを開始し、問題に対処する (その結果としてケースが破棄される) デフレクション項目を表示すると `lightningcommunity:deflectionSignal` イベントが起動します。

たとえば、ユーザーがカスタマーケースを作成するためのフォームに記入しているときに、ページ上で役に立つ記事を見つけたとします。その記事に有用な情報が記載されていたので、ユーザーはケースの作成は不要だと判断します。`lightningcommunity:deflectionSignal` イベントが起動します。このイベントには、ユーザーと記事のインタラクションに関する情報が含まれます。ユーザーはケースを作成しなかったため、ユーザーのアクションは成功したデフレクションとして報告されます。

これらのイベントは、対象オブジェクトのコミュニティケースデフレクション総計値を使用したカスタムレポートタイプで報告できます。レポートでは、成功したデフレクション、失敗したデフレクション、または潜在的なデフレクションとしてシグナルが表示されます。

メモ: 認証済みユーザーによってトリガーされた `lightningcommunity:deflectionSignal` イベントのみが報告されます。

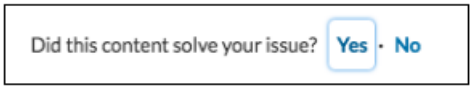
関連トピック:

ケース作成デフレクション信号

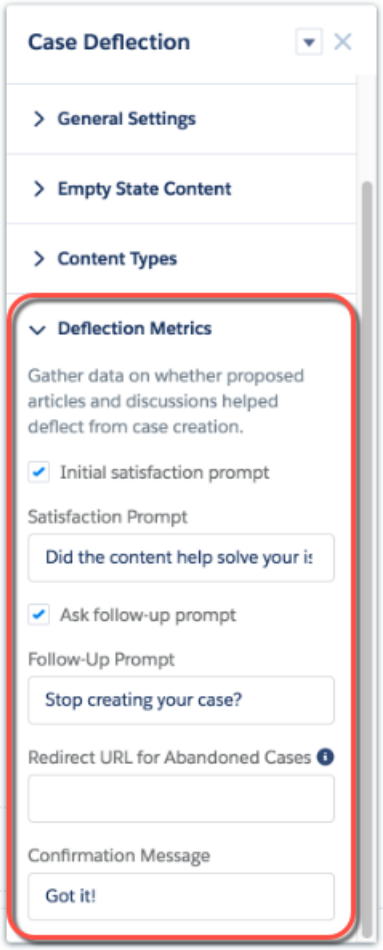
ケース作成デフレクション信号

`lightningcommunity:deflectionSignal` イベントは、ユーザーが顧客のケースの作成を回避したときにAuraサイトで起動します。ユーザーが記事またはディスカッションを表示した後に、ユーザーはやりとりが役に立ったかどうか、およびケースを破棄するかどうかを尋ねられます。

メモ: 認証済みユーザーからトリガーされた `lightningcommunity:deflectionSignal` イベントのデータのみが収集されます。



エクスペリエンスビルダーでコンポーネントの[デフレクション総計値]プロパティを使用して、このイベントを自動的に起動するようにケースデフレクションコンポーネントを設定できます。ケースデフレクションコンポーネントは【カスタマーサポートへの連絡】フォームと連携して、デフレクションのやりとりを登録します。



属性

ケースデフレクションコンポーネントからのデフレクション信号の `sourceType` は、`caseCreateDeflectionModal` です。

`source` は、ユーザーが件名項目またはケース作成フォームの説明に入力した内容です。`destination` は、記事またはディスカッションデフレクション項目の ID です。

`payload` は、JavaScript オブジェクトのキーと値の対応付けです。この種別の信号では、次のプロパティが使用されます。

ペイロードプロパティ	型	説明	サポートされている値	必須
`deflectionAnswer`	string	デフレクション項目が役に立ったかどうかを尋ねる最初の質問に対するユーザーの回答。	• `YES` • `NO` • `null` — ユーザーは投票しなかった	いいえ
`confirmationAnswer`	string	ケースの作成を停止するかどうかを尋ねる 2 つ目の質問に対するユーザーの回答。	• `YES` • `NO` • `null` — ユーザーは投票しなかった	いいえ
`state`	string	ポップアップウィンドウが閉じられる前の最後の状態。	• `MeasureDeflectionState` — ユーザーは最初の質問に回答しなかった • `ConfirmationQuestionState` — ユーザーは 2 つ目の質問に回答しなかった • `ConfirmationMessageState` — ユーザーは両方の質問に回答した	いいえ
`caseCreated`	boolean	ユーザーがケースを作成したかどうかを示します。	• `true` — ユーザーはケースを作成した • `false` — ユーザーはケースを作成しなかった	いいえ

例

カスタムAuraコンポーネントは、このシステムイベントをリスンし、必要に応じて処理できます。たとえば、ユーザーがコンテンツを役立たないと判断した場合に、コンポーネントで別のプロセスを開始できます。

システムイベントをリスンするサンプルコンポーネントを次に示します。

```
<aura:component implements="forceCommunity:availableForAllPageTypes">
    <aura:attribute name="message" type="String" required="false"/>
    <aura:handler event="lightningcommunity:deflectionSignal" action="{!c.handleSignal}"/>

    <lightning:formattedText value="{!v.message}"/>
</aura:component>
```

このクライアント側コントローラーの例では、システムイベントを処理して、失敗したケースデフレクションがあるかどうかを確認します。つまり、ユーザーがデフレクション項目を役立たないと判断したやりとりがあるかどうかをコントローラーで確認します。

```
({
    handleSignal: function(component, event, helper) {
        var signal = event.getParams() || {},
            sourceType = signal.sourceType,
            payload = signal.payload;
        // Process case create deflection signals
        if (sourceType && sourceType === "caseCreateDeflectionModal") {
            if (payload && payload.deflectionAnswer === "NO") {
                component.set("v.message", "Sorry you didn't find that helpful.");
            }
            if (payload && payload.caseCreated === true) {
                component.set("v.message", "We Apologize For The Inconvenience. We'll get
 in touch with you shortly about your case.");
            }
        }
    }
})
```

ケース作成フォームおよびケースデフレクションコンポーネントとして機能するカスタムAuraコンポーネントも、このイベントを起動できます。有効なパラメーターが指定されると、レポートを作成するためにイベントが自動的に処理されます。この例では、コンポーネントの属性の値を使用して `lightningcommunity:deflectionSignal` イベントを起動します。

```
fireCaseDeflectionSignal : function(component, shouldSubmitSourceTypeSignals) {
    var evt = $A.get("e.lightningcommunity:deflectionSignal");
    evt.setParams({
        sourceType: "caseCreateDeflectionModal",
        source: cmp.get("v.deflectionTerm"),
        destinationType: component.get("v.deflectionEntityType"),
        destination: component.get("v.deflectionEntityId"),
        payload: {
            deflectionAnswer: component.get("v.deflectionAnswer"),
            confirmationAnswer: component.get("v.confirmationAnswer"),
            state: component.get("v.deflectionState"),
            caseCreated: component.get("v.caseCreated")
        },
```

```
        shouldSubmitSourceTypeSignals: shouldSubmitSourceTypeSignals
    });
    evt.fire();
}
```

ユーザーは最終的にケースを作成するか破棄するかを決定する前に、複数のデフレクション項目を連続して表示できます。それぞれの表示で `lightningcommunity:deflectionSignal` イベントが起動します。すべてのイベントを 1 つのバッチで処理する場合は、ユーザーがケースを破棄または作成する最後のイベントで `shouldSubmitSourceTypeSignals=true` を設定します。この例では、ケースが作成されたかどうかに基づいて、最後のデフレクション信号イベントを起動します。

```
fireCaseCreatedSignal : function(component, caseCreated) {
    // Send all accumulated signals to the server to be processed
    var evt = $A.get("e.lightningcommunity:deflectionSignal");
    evt.setParams({
        sourceType: "caseCreateDeflectionModal",
        payload: {
            caseCreated: caseCreated
        },
        shouldSubmitSourceTypeSignals: true
    });
    evt.fire();
}
```

第 10 章　Sandbox から本番組織への Experience Cloud サイトのリリース

トピック:

本番組織にリリースする前に、テスト環境 (Sandbox など) で Experience Cloud サイトを作成、カスタマイズ、テストすることをお勧めします。テストが完了したら、変更セットかメタデータ API を使用して、ある組織から別の組織へサイトを移行できます。いくつかの要因に応じて、変更セットを使用するかメタデータ API を使用するかを決定します。移行する変更の複雑さ、開発者ツールの使いやすさ、使用しているアプリケーションライフサイクル管理 (ALM) モデルなどを考慮します。

エディション

使用可能なインターフェース: Salesforce Classic (使用できない組織もあります) および Lightning Experience の両方

使用可能なエディション: **Enterprise** Edition、**Performance** Edition、**Unlimited** Edition、および **Developer** Edition

ヒント: 使用可能な ALM モデルと開発オプションについての詳細は、Trailhead の「自分に適したアプリケーションライフサイクル管理モデルの判断」を参照してください。

メモ: Lightning Bolt ソリューションは、組織間で Experience Cloud サイトをリリースする場合には適していません。AppExchange でソリューションを共有または販売したり、すぐに利用できるソリューションや新しいデザインをサイトに実装したりする場合に Lightning Bolt ソリューションを使用します。

変更セット

ポイント & クリックツールを使用するほうが慣れている場合は、変更セットを使用してリリースします。変更セットは、接続組織にリリースできる組織のカスタマイズ (またはメタデータコンポーネント) のセットを表します。

宣言型ツールを使用してアプリケーションを管理できます。カスタマイズのニーズに対応するために、コマンドラインインターフェースやバージョン管理システムを使用する必要はありません。[設定] メニューを使用して、開発環境で変更を作成します。その後 ALM に取り組みながら、環境間で変更を移行します。

リリースアーティファクトは、本番組織に存在するメタデータからの変更部分の集合体です。リリースされるのは、メタデータに追加または変更される部分のみで、変更がなければリリースもありません。

メタデータ API

メタデータ API の最新情報に詳しく、コードの世界のほうが慣れている場合は、メタデータ API を使用して、変更をプログラムでリリースします。Experience Cloud サイト、カスタムオブジェクト定義、ページレイアウトなどの組織のカスタマイズ情報を取得、リリース、作成、更新、削除できます。

メタデータ API の使用は、変更が複雑な場合、または複数の作業ストリームを管理するために、より厳格な変更管理プロセスと監査プロセス(またはバージョン管理システム) が必要な場合に最適です。

変更セットプロセスの場合と同様、作成するリリースアーティファクトは、本番組織に存在するメタデータからの変更部分の集合体です。

ヒント: 一部の Experience Cloud サイト設定と機能は、まだメタデータ API でサポートされていないため、別の環境に手動で移行する必要があります。これらの変更を追跡し、移行し忘れないようにしてください。

関連トピック:

Trailhead: パッケージ開発を使用したアプリケーションの共同構築

Trailhead: Develop an App with Salesforce CLI and Source Control (Salesforce CLI とソース制御を使用したアプリケーションの開発)

変更セットを使用した Experience Cloud サイトのリリース

変更セットを使用すると、Experience Cloud サイトのすべてまたは一部をリリース接続のある関連組織間で移行できます。[ネットワーク] コンポーネントの種類を使用して、任意の Experience Cloud フルサイトを移行できます。または、[デジタルエクスペリエンス] コンポーネントの種類を使用して、拡張 LWR サイトの部分的なサイトコンテンツを移行できます。

エディション

使用可能なインターフェース: Salesforce Classic (使用できない組織もあります) および Lightning Experience

使用可能なエディション: **Enterprise** Edition、**Performance** Edition、および **Unlimited** Edition

変更セットを使用した完全な Experience Cloud サイトのリリース

変更セットを使用すると、リリース接続のある関連組織 (Sandbox 組織と本番組織など) 間で Experience Cloud サイトを移行できます。テスト環境でサイトを作成、カスタマイズ、およびテストし、テストが完了したらそのサイトを本番環境に移行します。

変更セットを使用した Experience Cloud サイトの部分的なコンテンツのリリース

変更セットを使用すると、リリース接続のある関連組織 (Sandbox 組織と本番組織など) 間で Experience Cloud の個々のコンポーネントおよびコンテンツを移行できます。テスト環境で変更を作成してテストし、準備が整ったらそれらの変更のみを本番に移行します。

変更セットを使用した Experience Cloud サイトのリリースの考慮事項

変更セットを使用してエクスペリエンスビルダーまたは Salesforce タブ + Visualforce サイトを移行するときは、次の考慮事項と制限に留意してください。

変更セットを使用した完全な Experience Cloud サイトのリリース

ユーザ権限

Experience Cloud サイトをカスタマイズまたは公開する	「エクスペリエンスの作成および設定」
リリース接続を変更し、受信変更セットを使用する	「変更セットのリリース」および「すべてのデータの編集」 メモ: リリースのためにユーザーがメタデータのみにアクセスする必要がある場合は、「メタデータ API 関数を使用したメタデータを変更」権限を有効にできます。この権限は、これらのユーザーに組織データへのアクセスを提供せずに、リリースに必要なアクセス権を付与します。詳細は、Salesforce ヘルプの「Modify Metadata Through Metadata API Functions Permission (「メタデータ API 関数を使用した

エディション

使用可能なインターフェース: Salesforce Classic (使用できない組織もあります) および Lightning Experience

使用可能なエディション: **Enterprise** Edition、**Performance** Edition、および **Unlimited** Edition

メタデータを変更」権限)」を参照してください。

送信変更セットを使用する	「変更セットの作成とアップロード」、「AppExchange パッケージの作成」、および「AppExchange パッケージのアップロード」

変更セットを使用すると、リリース接続のある関連組織 (Sandbox 組織と本番組織など) 間で Experience Cloud サイトを移行できます。テスト環境でサイトを作成、カスタマイズ、およびテストし、テストが完了したらそのサイトを本番環境に移行します。

変更セットでコンポーネントの種類に [ネットワーク] を使用して、完全なエクスペリエンスビルダーサイトと Salesforce タブ + Visualforce サイトを移行できます。

1. Sandbox などの任意のテスト組織でサイトを作成してテストします。
2. テスト組織で [設定] から、[クイック検索] ボックスに「*送信変更セット*」と入力し、[送信変更セット] を選択します。
3. 変更セットを作成し、[変更セットコンポーネント] セクションで [追加] をクリックします。
4. サイトの種類として [ネットワーク] を選択し、コミュニティを選択して、[変更セットに追加] をクリックします。
5. 連動する項目を追加するには、[連動関係を参照/追加] をクリックします。表示されたすべての連動関係を選択することをお勧めします。

   ヒント:

   - 標準オブジェクトにリンクするナビゲーションメニューの場合、カスタムリストビューは連動関係として含まれません。手動でカスタムリストビューを変更セットに追加します。
   - [管理] > [メンバー] で参照される新規または変更済みのプロファイルまたは権限セットを手動で追加します。
   - 連動関係のリストには 2 つの Site.com 項目 (`MySiteName` と `MySiteName1`) が含まれます。`MySiteName` には、エクスペリエンスワークスペースの [管理] で設定可能なさまざまな Visualforce ページが保持されます。`MySiteName1` には、エクスペリエンスビルダーからのページが含まれます。

6. [アップロード] をクリックし、本番など、対象組織を選択します。

   対象組織で着信接続が許可されていることを確認します。着信および発信組織にはリリース接続が必要です。

7. [設定] から [受信変更セット] を選択し、リリース元組織からアップロードした変更セットを見つけます。
8. 変更セットを検証およびリリースして対象組織で使用できるようにします。

   警告: 受信変更セットをリリースすると、対象組織のサイトが上書きされます。

9. 対象組織のサイトにサポートされていない項目がある場合は手動で再設定します。

10. サイトのデータを追加し、テストしてすべてが期待どおりに機能することを確認します。次に、変更を公開して稼働を開始します。

関連トピック:

Salesforce ヘルプ: 変更セットのベストプラクティス
Salesforce ヘルプ: 送信変更セットのアップロード
Salesforce ヘルプ: 受信変更セットのリリース

変更セットを使用した Experience Cloud サイトの部分的なコンテンツのリリース

ユーザ権限	
Experience Cloud サイトをカスタマイズまたは公開する	「エクスペリエンスの作成および設定」
リリース接続を変更し、受信変更セットを使用する	「変更セットのリリース」および「すべてのデータの編集」 リリースのためにユーザーがメタデータのみにアクセスする必要がある場合は、「メタデータ API 関数を使用したメタデータを変更」権限を有効にできます。この権限は、これらのユーザーに組織データへのアクセスを提供せずに、リリースに必要なアクセス権を付与します。詳細は、*Salesforce* ヘルプの「「メタデータ API 関数を使用したメタデータを変更」権限」を参照してください。
送信変更セットを使用する	「変更セットの作成とアップロード」、「AppExchange パッケージの作成」、および「AppExchange パッケージのアップロード」

エディション

使用可能なインターフェース: Salesforce Classic (使用できない組織もあります) および Lightning Experience

使用可能なエディション: **Enterprise** Edition、**Performance** Edition、および **Unlimited** Edition

変更セットを使用すると、リリース接続のある関連組織 (Sandbox 組織と本番組織など) 間で Experience Cloud の個々のコンポーネントおよびコンテンツを移行できます。テスト環境で変更を作成してテストし、準備が整ったらそれらの変更のみを本番に移行します。

変更セットを使用した部分的なリリースは、Winter '23 以降で作成された拡張 LWR サイトで使用できます。

変更セットでは、[デジタルエクスペリエンス]コンポーネントの種類を使用して、エクスペリエンスビルダーサイトの部分的なコンテンツを移行できます。

1. Sandbox などの任意のテスト組織でサイトを作成してテストします。

2. テスト組織の [設定] から、[クイック検索] ボックスに「*送信変更セット*」と入力し、[送信変更セット] を選択します。
3. 変更セットを作成し、[変更セットコンポーネント] セクションで [追加] をクリックします。
4. [カスタムデジタルエクスペリエンス] コンポーネントの種類を選択します。
5. コンポーネントのリストからリリースするコンテンツを選択し、[変更セットに追加] をクリックします。

   ヒント:

   - リストされた各コンポーネントが属する拡張 LWR サイトを特定するには、[種別] 列を使用します。[種別] 列では、site/*MySiteName* という命名規則を使用します。
   - リストされた各コンポーネントが表すコンテンツを特定するには、[名前] 列を使用します。[名前] 列では、sfdc_cms__<contentType>/<contentName> という命名規則を使用します。たとえば、sfdc_cms__brandingSet/Build_Your_Own_LWR という名前は、Build Your Own LWR という名前のブランドセットを表します。

6. [アップロード] をクリックし、本番など、対象組織を選択します。

   入力先組織で受信接続が許可されていることを確認します。受信および送信組織にはリリース接続が必要です。

7. [設定] から [受信変更セット] を選択し、ソース組織からアップロードした変更セットを見つけます。
8. 変更セットを検証およびリリースして対象組織で使用できるようにします。

   警告: 受信変更セットをリリースすると、対象組織の対応するサイトコンテンツが上書きされます。

9. 対象組織のサイトにサポートされていない項目がある場合は手動で再設定します。
10. すべてが期待どおり動作することを確認するために、サイトをテストします。次に、変更を公開して稼働を開始します。

変更セットを使用した Experience Cloud サイトのリリースの考慮事項

変更セットを使用してエクスペリエンスビルダーまたは Salesforce タブ + Visualforce サイトを移行するときは、次の考慮事項と制限に留意してください。

エディション

使用可能なインターフェース: Salesforce Classic (使用できない組織もあります) および Lightning Experience

使用可能なエディション: **Enterprise** Edition、**Performance** Edition、および **Unlimited** Edition

一般情報

- 受信変更セットをリリースすると、対象組織の Experience Cloud サイトが上書きされます。変更セットを使用してコンポーネントを削除することはできませんが、エクスペリエンスビルダーサイト内のページは削除できます。たとえば、Sandbox でエクスペリエンスビルダーサイトからページを削除し、更新された送信変更セットを作成したとします。対象組織 (本番組織など) にその変更セットを再リリースすると、そこでもページが削除されます。
- リリース元組織で Experience Cloud サイトテンプレートを更新する場合、変更セットをリリースする前に、リリース先組織でもテンプレートを更新します。

- 以前のリリースバージョンを使用している対象組織にリリースすることはできません。たとえば、リリース元組織が Summer '19 (API バージョン 46.0) の場合、Spring '19 (API バージョン 45.0) の対象組織にはリリースできません。

管理

[管理] 設定は、エクスペリエンスワークスペースにあります。

- [管理] > [メンバー] で参照される、新規または変更済みのプロファイルまたは権限セットを送信変更セットに追加します。これらが連動関係として自動的に含まれることはありません。
- Summer '17 リリースより前に Sandbox 組織で作成された Experience Cloud サイトの場合、正常に転送するには、移行の前に管理設定を再保存する必要があります。
- リリース先組織でサイトを公開するまで、パスワードの変更、パスワードを忘れた場合、ホーム、セルフ登録、およびログインページの設定がデフォルト値に戻って表示されます。
- [メンバー] 領域および [ログイン & 登録] 領域の設定を更新するには、別の変更セットで変更をリリースする必要があります。最初に [メンバー] 領域の設定を更新およびリリースしてから、[ログイン & 登録] 設定を更新およびリリースします。

ナビゲーションメニュー

[ナビゲーションメニュー] コンポーネントは、エクスペリエンスビルダーサイトで使用できます。

- オブジェクトにリンクするメニュー項目では、リストビューはデフォルトのリストビューにリセットされます。また、標準オブジェクトのカスタムリストビューは連動関係として含まれません。
- メニュー項目を追加したナビメニューをリリースすると、対象環境の既存のメニュー項目に適用されているすべての翻訳が削除されます。

おすすめ

- おすすめの名前に対する更新はサポートされていません。以前に移行されたことがあるリリース元組織でおすすめの名前を変更した場合、対象組織では新しいおすすめとして扱われます。
- おすすめの画像はサポートされません。
- 受信変更セットをリリースすると、リリース先組織のスケジュール済みおすすめが、リリース元組織からのおすすめで上書きされます。

拡張 LWR サイトの部分的リリース

- Winter '23 以降に作成された拡張 LWR Experience Cloud サイトでは、サイトコンテンツの部分的なリリースが可能です。
- 部分的なサイトコンテンツをリリースするには、リリース先組織にリリース元サイトと同じ名前の既存のサイトを含めます。
- デジタルエクスペリエンスコンポーネントを送信変更セットに追加すると、そのコンポーネントのバリエーションも変更セットに追加されます。たとえば、ホームビューコンポーネントを追加すると、ホームビューのすべての翻訳が自動的に追加されます。

サポート対象外の設定と機能

次の項目はサポートされません。受信変更セットをリリースした後に手動で追加してください。

- ナビゲーショントピックおよび主要トピック
- 利用者ターゲティング
- ダッシュボードとエンゲージメント
- おすすめの画像
- エクスペリエンスビルダーの [ブランド] パネルの画像
- エクスペリエンスワークスペースにある次の [管理] 設定:
  - [ログイン & 登録] 領域の [登録] セクションの [取引先] 項目
  - [ログイン & 登録] 領域の [登録] セクションの [表示するログインオプションを選択します] オプション
  - [設定] 領域
  - リッチパブリッシャーアプリケーション領域

関連トピック:

Salesforce ヘルプ: 変更セットのベストプラクティス

Salesforce ヘルプ: 変更セットの実装のヒント

メタデータ API を使用した Experience Cloud サイトのリリース

エディション

使用可能なインターフェース: Salesforce Classic (使用できない組織もあります) および Lightning Experience の両方

使用可能なエディション: **Enterprise** Edition、**Performance** Edition、**Unlimited** Edition、および **Developer** Edition

メタデータ API を使用して、Salesforce 組織間で Experience Cloud サイトを移行します。テスト環境でサイトの設定とテストを行ってから、サイトのデータを取得して本番組織にリリースします。

サイトのフレームワークに応じて、次のメタデータ型を組み合わせてサイトを定義します。サイトを移行するには、メタデータ API の `retrieve` コールを使用して、組織のコンポーネントの XML ファイル表現を取得します。

Network
Experience Cloud サイトを表します。ページの上書き、メール、メンバーシップの設定などの管理設定が含まれています。

CustomSite
`indexPage`、`siteAdmin`、URL 定義など、ドメインとページの設定情報が含まれています。

DigitalExperienceBundle と DigitalExperienceConfig または ExperienceBundle あるいは SiteDotCom
必要なメタデータ型は、サイトの種別によって異なります。メタデータ型は、サイトを構成するさまざまな設定や、ページ、ブランドセット、テーマなどのコンポーネントを表します。

23 年冬に導入され機能強化された LWR サイト (API バージョン 56.0) では、DigitalExperienceBundle と DigitalExperienceConfig を組み合わせて、サイトの要素や設定がテキストベースの表現で提供されますが、よりコンテンツを重視した構造に刷新されています。編集可能なサイトメタデータを取得し、サイトをプロ

グラムですばやく作成、更新、公開、リリースできます。また、機能強化された LWR サイトを部分的にリリースすることも可能です。

機能強化されていない LWR サイトの場合、ExperienceBundle は、サイトの設定、ページ、およびコンポーネントのテキストベースの表現を提供します。編集可能なサイトメタデータを取得し、サイトをプログラムですばやく作成、更新、公開、および部分的にリリースできます。

Aura サイトの場合、ExperienceBundle と SiteDotCom のいずれかを選択して使用できますが、ExperienceBundle の使用をお勧めします。Summer '19 以前 (API バージョン 45.0 以前) は、Network、CustomSite、SiteDotCom メタデータ型を組み合わせて Aura サイトを定義していました。ただし、SiteDotCom 型を取得すると、人間が判読できないバイナリの `.site` ファイルが生成されました。「ExperienceBundle for Experience Builder Sites (エクスペリエンスビルダーサイトの ExperienceBundle)」を参照してください。

Visualforce サイトの場合、Network でサイトを表します。

これらのメタデータ型についての詳細とデータ移行の手順は、『メタデータ API 開発者ガイド』と『Salesforce CLI コマンドリファレンス』を参照してください。

必要なメタデータ型の一覧

サイトのリリースに使用するメタデータ型は、サイトの種別によって異なります。

必要なメタデータ型	機能強化された LWR サイト	LWR サイト	Aura サイト	Visualforce サイト
Network	✅	✅	✅	✅
CustomSite	✅	✅	✅	✅
DigitalExperienceBundle	✅			
DigitalExperienceConfig	✅			
ExperienceBundle		✅	✅	
SiteDotCom			ExperienceBundle または SiteDotCom	✅

ヒントと検討事項

- データを別の組織に移行する前に、エラーの発生を避けるため、移行先の組織でデジタルエクスペリエンスを有効化し、Sandbox 組織で使用したものと同じドメイン名を入力します。
- 各 Experience Cloud サイトのネットワークコンポーネントには、一意の名前と URL パスプレフィックスがあります。ネットワークコンポーネントを取得すると生成される XML ファイル名は、ネットワークの名前に基づきます。移行時に API でファイル名が参照され、ファイル名が存在している場合はサイトが更新されます。存在していない場合、API でサイトが作成されます。誰かが Sandbox でサイト名を変更して移行しようとすると、エラーが表示されます。API は、既存のパスプレフィックスでサイトの作成を試みます。
- すべての連動関係が取り込まれていることを CustomSite の XML ファイルで確認します。いずれかがない場合、XML ファイルで明示的に指定します。

- 前述の必須コンポーネントに加えて、使用するサイトに必要な他のコンポーネントもすべて組み込みます。コンポーネントには、カスタムオブジェクト、カスタム項目、カスタム Lightning コンポーネント、Apex クラスなどの項目を含めることができます。
- ロック解除済みパッケージを使用して Network コンポーネントと Profile コンポーネントをリリースするには、コンポーネントごとに個別のロック解除済みパッケージを作成して個別にリリースします。
- ExperienceBundle を使用して Aura サイトをリリースする場合、SiteDotCom 型がマニフェストファイルに含まれていないことを確認します。
- [管理]>[設定]でサイトの名前を変更する場合は、リリース元サイトとリリース先サイトで、`Network` コンポーネントの `picassoSite` 属性と `site` 属性の値が一致することを確認します。
- ゲストユーザープロファイルに変更があった場合は、サイト移行の一部としてプロファイルを含めます。
- ユーザープロファイルを移行すると、ユーザーが本番組織のサイトに追加されます。その後、新しいサイトの場合と同じようにメールがメンバーに送信されます。
- リリース中、リリース先組織の `NavigationMenu` API 参照名がリリース元組織の API 参照名と同じであることを確認します。
- containerType が CommunityTemplateDefinition の場合、メタデータ API を介して既存の `NavigationMenu` を更新することはできません。
- カスタムテンプレートによって Aura サイトをリリースするには、まずに CommunityTemplateDefinition と関連するメタデータタイプ (CommunityThemeDefinition など) を取得してリリースします。次に、ExperienceBundle または SiteDotCom と関連するメタデータ型を取得してリリースします。
- メニュー項目を追加したナビメニューをリリースすると、リリース先環境の既存のメニュー項目に適用されているすべての翻訳が削除されます。
- サイトを移動する際にナビゲーションメニューを含めるには、NavigationMenu メタデータ型を使用します。
- 以前のリリースバージョンを使用しているリリース先組織にリリースすることはできません。たとえば、リリース元組織が Summer '19 (API バージョン 46.0) の場合、Spring '19 (API バージョン 45.0) のリリース先組織にはリリースできません。
- `NavigationLinkSet` は、Winter '20 (API バージョン 47.0) で廃止され、`NavigationMenu` に置き換えられました。
- ExperienceBundle では、異なる API バージョンでの取得およびリリースはサポートされません。ExperienceBundle メタデータを古い API バージョンから新しいバージョン (API バージョン 48.0 から 49.0 など) にアップグレードする場合は、次の手順を実行します。
  1. package.xml マニフェストファイルの API バージョンを 48.0 に設定して、パッケージをリリースします。
  2. 次に、package.xml の API バージョンを 49.0 に設定します。
  3. 最新の ExperienceBundle の更新を取得するには、パッケージを取得します。
- サイトのリリース時には、無効な ID 値に関する警告メッセージが表示されることがあります。例: 「コンポーネント `9b8a4e98-e724-4292-bd3c-0813adf9ddc2` の `topicId` プロパティは `ID` 値が `0TO4R000000EGPEWA4` のオブジェクトを参照しています。場合によっては、対象組織にリリースされたときに、`ID` 値が無効になることがあります。たとえば、参照される `ID` が対象組織に存在しない場合があります。対象組織でコンポーネントの問題が発生した場合は、`ID` 値が正しいことを確認してください」。

  この状況では、警告が表示されてもサイトを正常にリリースできます。ただし、コンポーネントが参照しているオブジェクト ID がまだ有効であることを、対象組織で確認することをおすすめします。ID が正しく

ない場合は、手動で ID を更新し、対象組織内でのコンポーネントの問題を解決してください。また、自分で成したカスタムコンポーネントの場合は、オブジェクト ID をオブジェクトの API 名に置き換えて、今後この問題が発生しないようにすることを検討してください。

サンプルテンプレート

次のサンプルには、メタデータ API で移行できるすべての項目が含まれています。

```
<?xml version="1.0" encoding="UTF-8"?>
<Network xmlns="http://soap.sforce.com/2006/04/metadata">
    <allowInternalUserLogin>true</allowInternalUserLogin>
    <allowMembersToFlag>true</allowMembersToFlag>
    <allowedExtensions>txt,png,jpg,jpeg,pdf,doc,csv</allowedExtensions>
   <caseCommentEmailTemplate>unfiled$public/ContactFollowUpSAMPLE</caseCommentEmailTemplate>


<changePasswordTemplate>unfiled$public/CommunityChangePasswordEmailTemplate</changePasswordTemplate>

    </communityRoles>
    <disableReputationRecordConversations>true</disableReputationRecordConversations>
    <emailSenderAddress>admin@myorg.com</emailSenderAddress>
    <emailSenderName>MyCommunity</emailSenderName>
    <enableCustomVFErrorPageOverrides>true</enableCustomVFErrorPageOverrides>
    <enableDirectMessages>true</enableDirectMessages>
    <enableGuestChatter>true</enableGuestChatter>
    <enableGuestFileAccess>false</enableGuestFileAccess>
    <enableInvitation>false</enableInvitation>
    <enableKnowledgeable>true</enableKnowledgeable>
    <enableNicknameDisplay>true</enableNicknameDisplay>
    <enablePrivateMessages>false</enablePrivateMessages>
    <enableReputation>true</enableReputation>
    <enableShowAllNetworkSettings>true</enableShowAllNetworkSettings>
    <enableSiteAsContainer>true</enableSiteAsContainer>
    <enableTalkingAboutStats>true</enableTalkingAboutStats>
    <enableTopicAssignmentRules>true</enableTopicAssignmentRules>
    <enableTopicSuggestions>true</enableTopicSuggestions>
    <enableUpDownVote>true</enableUpDownVote>

<forgotPasswordTemplate>unfiled$public/CommunityForgotPasswordEmailTemplate</forgotPasswordTemplate>

    <gatherCustomerSentimentData>false</gatherCustomerSentimentData>
    <lockoutTemplate>unfiled$public/CommunityLockoutEmailTemplate</lockoutTemplate>
    <maxFileSizeKb>51200</maxFileSizeKb>
    <networkMemberGroups>
        <permissionSet>MyCommunity_Permissions</permissionSet>
        <profile>Admin</profile>
    </networkMemberGroups>
    <networkPageOverrides>
       <changePasswordPageOverrideSetting>VisualForce</changePasswordPageOverrideSetting>

        <forgotPasswordPageOverrideSetting>Designer</forgotPasswordPageOverrideSetting>
        <homePageOverrideSetting>Designer</homePageOverrideSetting>
        <loginPageOverrideSetting>Designer</loginPageOverrideSetting>
```

```
        <selfRegProfilePageOverrideSetting>Designer</selfRegProfilePageOverrideSetting>
    </networkPageOverrides>
    <picassoSite>MyCommunity1</picassoSite>
    <selfRegistration>true</selfRegistration>
    <sendWelcomeEmail>true</sendWelcomeEmail>
    <site>MyCommunity</site>
    <status>Live</status>
    <tabs>
        <defaultTab>home</defaultTab>
        <standardTab>Chatter</standardTab>
    </tabs>
    <urlPathPrefix>mycommunity</urlPathPrefix>
    <welcomeTemplate>unfiled$public/CommunityWelcomeEmailTemplate</welcomeTemplate>
</Network>
```

package.xml マニフェストファイルのサンプル

マニフェストファイルでは、取得するコンポーネントを定義します。次のサンプルに、エクスペリエンスビルダーサイトのすべてのコンポーネントを取得するための package.xml マニフェストファイルを示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>*</members>
        <name>Network</name>
    </types>
    <types>
        <members>*</members>
        <name>CustomSite</name>
    </types>
    <types>
        <members>*</members>
        <name>ExperienceBundle</name>
    </types>
    <types>
        <members>*</members>
        <name>CustomTab</name>
    </types>
    <types>
        <members>*</members>
        <name>CustomObject</name>
    </types>
    <types>
        <members>*</members>
        <name>ApexClass</name>
    </types>
    <types>
        <members>*</members>
        <name>ApexPage</name>
    </types>
    <types>
        <members>*</members>
        <name>ApexComponent</name>
```

```
    </types>
    <types>
        <members>*</members>
        <name>Portal</name>
    </types>
    <types>
        <members>*</members>
        <name>Profile</name>
    </types>
    <types>
        <members>*</members>
        <name>Document</name>
    </types>
    <version>46.0</version>
</Package>
```

エクスペリエンスビルダーサイトの ExperienceBundle

ExperienceBundle メタデータ型は、エクスペリエンスビルダーサイトを構成するさまざまな設定およびコンポーネント(ページ、ブランドセット、テーマなど)のテキストベースの表現を提供します。サイトが独自の組織用であるかどうかにかかわらず、またはあなたがコンサルティングパートナーまたは ISV であるかどうかにかかわらず、VS Code 向け Salesforce 拡張機能、Salesforce CLI、お気に入りの IDE またはテキストエディターなど、好みの開発ツールを使用してサイトをプログラムですばやく更新し、リリースします。

Summer '19 リリース以前 (API バージョン 45.0 以前) は、Network、CustomSite、SiteDotCom メタデータ型を組み合わせてエクスペリエンスビルダーサイトを定義していました。ただし、SiteDotCom 型を取得すると、人間が判読できないバイナリの `.site` ファイルが生成されました。SiteDotCom ではなく ExperienceBundle 型を取得すると、3 レベルのフォルダー構造に含まれる詳細なサイトメタデータを、人間が判読可能な形式で抽出して編集できます。

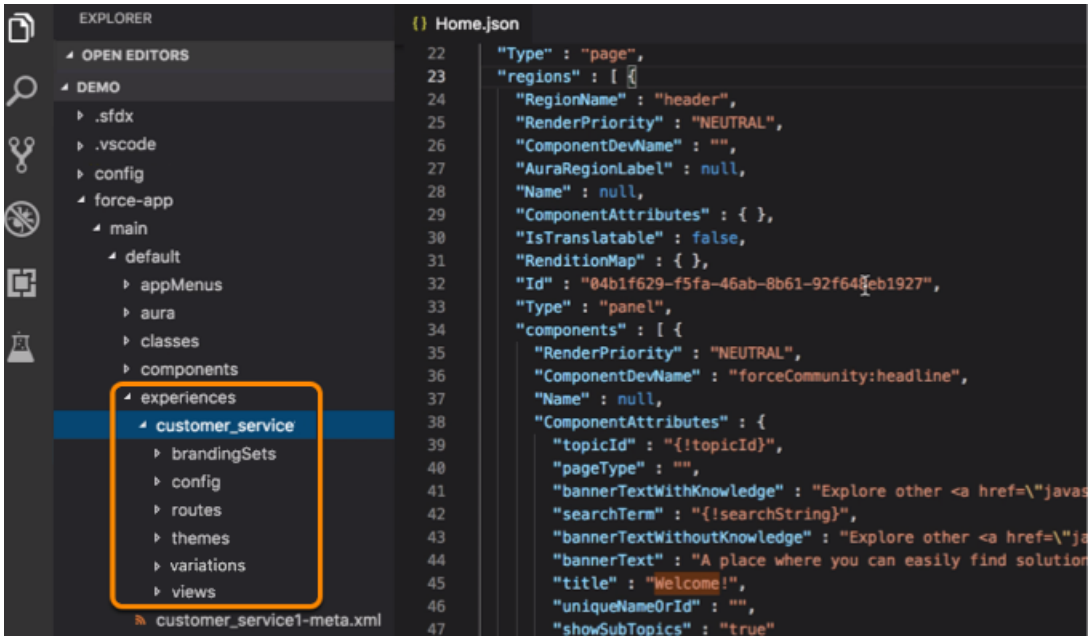
制限事項

- 管理パッケージはサポートされません。

ExperienceBundle の構造

ExperienceBundle を取得すると、データは 3 レベルのフォルダー構造に保存されます。

`experiences` フォルダーには、組織の各エクスペリエンスビルダーサイト用のフォルダーが含まれます。各 *`site_name`* フォルダー(この例では `customer_service`)には、サイトを定義し、エクスペリエンスビルダーでアクセスするさまざまな要素を表すサブフォルダーが含まれます。各サブフォルダーは、ローカルマシンやスクラッチ組織で編集してからリリースできるプロパティが入った .json ファイルを持っています。



各エクスペリエンスビルダーサイトを定義するファイルを詳しく見てみましょう。

フォルダー	内容
brandingSets	*branding_set_name*.json は、サイトのブランドセットプロパティを定義します。
config	• *site_name*.json は、公開アクセスや順次表示などのサイト設定を定義します。 • languages.json は、サポートされる言語を定義します。 • loginAppPage.json と mainAppPage.json は、単一ページアプリケーション (SPA) です。loginAppPage.json は、ログインを必要とするサイトページで使用します。mainAppPage は、他のすべてのページで使用します。 SPA は、1 つの HTML ページを読み込む Web アプリケーションです。ユーザーが移動する複数のページで構成される従来の Web サイトとは異なり、SPA は、ユーザーが操作するときにページを動的に更新するための複数のビューで構成されています。
routes	URL や他のルート関連情報を定義する *page_name*.json という名前のファイルがページごとに 1 つ含まれます。
themes	テーマを定義する *theme_name*.json という名前のファイルがテーマごとに 1 つ含まれます。
variations	*experienceVariation_name*.json という名前のファイルがバリエーションごとに 1 つ含まれます。 環境のバリエーションを使用することで、利用者ごとにエクスペリエンスビルダーサイトにおけるブランド設定、ページバリエーション、コンポーネントの表示、コンポーネントの属性などのデフォルト動作を変更できます。

フォルダー	内容
views	*view_name*.json という名前のファイルがビューごとに 1 つ含まれます。各ファイルは、エンドユーザーに表示するページに相当する SPA ビューを定義します。ビューは、表示されるページ内の他の領域やコンポーネントを含む領域から構成されます。

ヒント: エクスペリエンスビルダーサイトの .json ファイルを更新する前に、バックアップとしてサイトのフォルダーをコピーすることをお勧めします。

ExperienceBundle の完全な定義と、含まれるファイルについては、『メタデータ API 開発者ガイド』を参照してください。

ExperienceBundle のメタデータ型の有効化

Aura サイトの ExperienceBundle を有効化すると、メタデータ API コール (`retrieve` および `deploy`) と Salesforce DX 操作 (`pull`、`push`、`status`) は SiteDotCom 型ではなく ExperienceBundle 型を使用します。

変更セットを使用してサイトをリリースすると、連動関係のリストには Site.com 型の 2 つの項目 (*MySiteName* and *MySiteName*1) が含まれます。*MySiteName*1 は、SiteDotCom ではなく ExperienceBundle を表すようになります。

メモ: LWR サイトの ExperienceBundle メタデータ型を有効にする必要はありません。LWR サイトでは、デフォルトで ExperienceBundle が使用されます。

1. [設定] から、[クイック検索] ボックスに「デジタルエクスペリエンス」と入力し、[設定] を選択します。
2. **[ExperienceBundle メタデータ API を有効化]** を選択します。
3. 変更内容を保存します。

あるいは、スクラッチ組織定義ファイルを使用してスクラッチ組織を作成するときにこの機能を有効にすることもできます。(「Metadata Coverage (対象メタデータ)」レポート」を参照)。

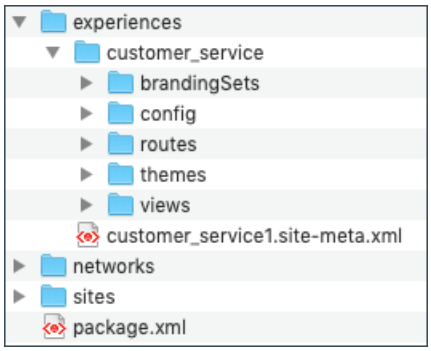
```
{
    "orgName": "Sample Org",
    "edition": "developer",
    "features": [
        "COMMUNITIES"
    ],
    "settings": {
        "experienceBundleSettings": {
            "enableExperienceBundleMetadata": true
        },
        "communitiesSettings": {
            "enableNetworksEnabled": true
        }
    }
}
```

メタデータ API を使用した ExperienceBundle の取得とリリース

メタデータ API では、マニフェストファイルは、取得するコンポーネントを定義します。次の例は、SiteDotCom ではなく ExperienceBundle を使用してエクスペリエンスビルダーサイトを取得する `package.xml` マニフェストファイルを示しています。

```
<?xml version="1.0" encoding="UTF-8"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>*</members>
        <name>CustomSite</name>
    </types>
    <types>
        <members>*</members>
        <name>ExperienceBundle</name>
    </types>
    <types>
        <members>*</members>
        <name>Network</name>
    </types>
    <version>46.0</version>
</Package>
```

.zip ファイルを取得したら、展開して取り出したファイルを開いて編集してください。



「zip ファイルを使用したメタデータのリリースと取得」を参照してください。

Salesforce DX を使用した ExperienceBundle の取得とリリース

Salesforce Developer Experience (DX) は、開発ライフサイクル全体を合理化するツールセットです。Salesforce DX によって、チーム開発とコラボレーションが向上し、自動テストと継続的インテグレーションが促進され、リリースサイクルがより効率的で俊敏になります。

Salesforce DX 環境を設定してある場合は、以下をすばやく行うことができます。

- `sf project retrieve start` を使用して組織内のすべてのエクスペリエンスビルダーサイトを取得する
- `sf project deploy start` を使用して更新をリリースする
- `sf project deploy preview` または `sf project retrieve preview` を使用してサーバー上の競合や変更を確認する

- `sf community list template` を使用して使用可能なテンプレートのリストを取得する
- `sf community create` を使用してサイトを作成する
- `sf community publish` を使用してエクスペリエンスビルダーサイトを公開する

Salesforce DX に不慣れな場合は、以下の役に立つリソースを参照してください。

- Salesforce DX 開発者ガイド
- 「クイックスタート: Salesforce DX」 (Trailhead)
- 「パッケージ開発を使用したアプリケーションの共同構築」 (Trailhead)
- Salesforce CLI コマンドリファレンス

関連トピック:

ブログ投稿: ExperienceBundle & Salesforce DX: A Developer's Dream for Coding Lighting Communities (ExperienceBundle & Salesforce DX: Lightning コミュニティのコーディングに関する開発者の夢)

メタデータ API 開発者ガイド: ExperienceBundle

Salesforce CLI コマンドリファレンス: community コマンド

拡張 LWR サイトへの移行時のリリースの問題の回避

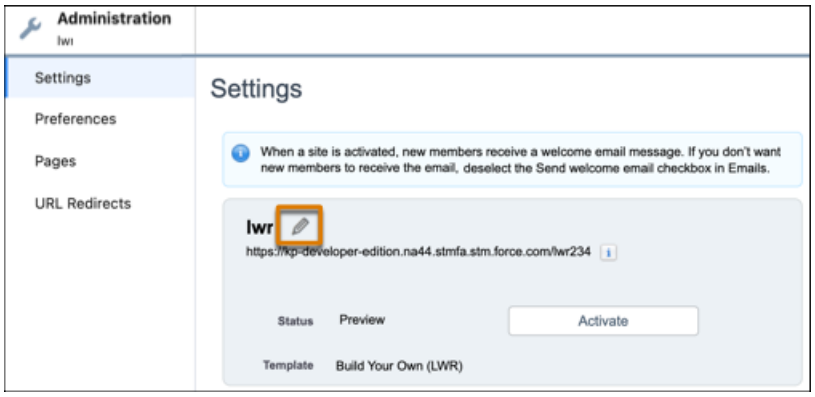
Winter '24 からは、Winter '23 に初めて導入された拡張サイトとコンテンツプラットフォームを無効にすることができなくなります。その結果、LWR テンプレートから作成されるすべてのサイトは、デフォルトで拡張 LWR サイトとなります。ExperienceBundle メタデータ型を使用する非拡張 LWR サイトとは異なり、拡張 LWR サイトでは DigitalExperienceBundle 型と DigitalExperienceConfig 型を使用します。ただし、Winter '24 以前に Sandbox などのソース組織で作成してある非拡張 LWR サイトを、本番組織などの対象組織に初めてリリースする場合は、このメタデータ型の違いによってリリースの問題が発生する場合があります。

この場合は、リリース*前に*対象組織でサイトを作成するのではなく、リリースプロセスを使用してサイトを作成することをお勧めします。そうせずに最初に対象組織でサイトを作成した場合、新しいサイトは拡張 LWR サイトとなり、DigitalExperienceBundle 型と DigitalExperienceConfig 型が使用されます。そしてソース組織上の非拡張 LWR サイトでは ExperienceBundle 型を使用しているため、そのサイトを対象組織にリリースしようとすると、メタデータ型の不一致のためにリリースが失敗します。

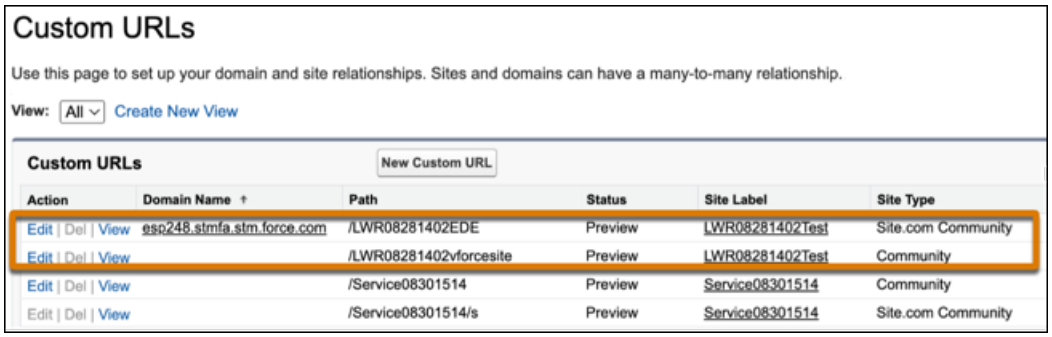
もし、リリース前に対象組織でサイトを作成してあるとしたら、リリースエラーを解決するにはどうすればよいでしょうか。

Experience Cloud サイトを削除することはできないため、この場合にはサイトの名前を変更して、対象組織のサイト URL を更新することで、ソース組織の値と一致しないようにすることをお勧めします。これによってデプロイメントプロセスが解放され、ソース組織の値を使用して対象組織でサイトを作成し直せるようになります。

1. 対象組織では、サイトの [管理] ワークスペースの [設定] ページで、サイト名をソース組織のサイトとは違う名前に変更します。



2. [設定] から [クイック検索] ボックスに「カスタム *URL*」と入力します。
3. [カスタム URL] で、サイトの 2 つの URL を見つけます。各サイトには次の URL があります。
   - 必要に応じて ExperienceBundle 型または DigitalExperienceBundle および DigitalExperienceConfig 型に対応付けられる Site.com コミュニティ URL
   - CustomSite 型に対応付けられるコミュニティ URL



4. ソース組織のサイト URL とは同じにならないように両方のサイト URL を変更します。
5. サイト名と URL を更新したら、好きなリリースツールを使用して、変更セットまたはメタデータ API でサイトを再びリリースします。
6. Metadata API を使用する場合は、システムが対象組織で非拡張 LWR サイトを自動的に作成できるようにするため、サイトを取得して Network 型と CustomSite 型を含めてください。変更セットを使用する場合は、リリースする前に変更セットを再作成してください。

関連トピック:

Salesforce ヘルプ: 拡張サイトおよびコンテンツプラットフォームとは?

メタデータ API 開発者ガイド: DigitalExperienceBundle

認証済み LWR サイトのリリースに関する考慮事項

Winter '23 より、エクスペリエンスビルダーまたは Connect API で作成された新しい LWR サイトの URL の末尾に /s が含まれなくなりました。Winter '23 以前に作成された認証済み LWR サイトの URL は、依然として /s を含んでおり、この URL 構造の更新は、Sandbox と本番 URL が一致しない場合にリリースに影響します。サポートされるリリースシナリオと、/s に関係するサポートされていないリリースエラーの解決方法について説明します。

メモ: このシナリオでは、Sandbox と本番が同じバージョンの Salesforce であることを前提としています。サイトを新しいバージョンから古いバージョンへリリースすることはサポートされていません。

エディション

使用可能なインターフェース: Salesforce Classic (使用できない組織もあります) および Lightning Experience

使用可能なエディション: **Enterprise** Edition、**Developer** Edition、**Performance** Edition、および **Unlimited** Edition

メタデータ API リリース

- Metadata API リリースによるサイトの作成は、リリース元サイトの URL に /s が含まれているかどうかに関係なくサポートされます。リリース元サイトの URL が /s を含む場合、新しいリリース先サイトの URL にも /s が含まれます。リリース元サイトの URL が /s が含まれていない場合、新しいリリース先サイトの URL にも /s は含まれません。
- メタデータ API リリースでサイトを更新するには、リリース元サイトの URL と更新先サイトの URL が一致している必要があります。両方の URL に /s が含まれているか、または両方とも /s が含まれていない必要があります。
- /s に関係するメタデータ API デプロイメントエラーを解決するには、リリース元サイトの URL で /s を追加または削除してメタデータバンドルを更新します。リリース元サイトの URL とリリース先サイトの URL を一致させます。リリース先サイトの URL で /s を追加したり削除したりすることはできません。

変更セットリリース

- 変更セットリリースによるサイトの作成は、リリース元サイトの URL に /s が含まれているかどうかに関係なくサポートされます。リリース元サイトの URL が /s を含む場合、新しいリリース先サイトの URL にも /s が含まれます。リリース元サイトの URL が /s が含まれていない場合、新しいリリース先サイトの URL にも /s は含まれません。
- 変更セットリリースでサイトを更新するには、リリース元サイトの URL と更新先サイトの URL が一致している必要があります。両方の URL に /s が含まれているか、または両方とも /s が含まれていない必要があります。
- /s に関係する変更セットリリースエラーを解決するには、API またはエクスペリエンスビルダーを使用してリリース元サイトまたはリリース先サイトのどちらかの名前を変更します。いずれかのサイトの名前を変更すると、リリース先サイトが更新されるのではなく、サイトが作成されます。リリース元サイトやリリース先サイトの URL で /s を追加したり、削除したりすることはできません。