

# Reveal 開発者ガイド

---

このガイドでは、Reveal SDK と API メソッドに関する技術情報 (アーキテクチャ情報を含む) を提供します。

## 作業の開始

[Reveal SDK の概要](#) - SDK の主な概念と機能の紹介。

[インストールと要件](#) - インストール方法と必須条件。

[Desktop SDK の概要](#) Desktop SDK の主な概念とアーキテクチャについて。

- [セットアップと構成](#) Desktop SDK を使用できるようにするために必要な手順。

[WEB .NET SDK の概要](#) Web .NET SDK の主な概念とアーキテクチャについて。

- [セットアップと構成](#) Web .NET SDK を使用できるようにするために必要な手順。
- [はじめてのアプリ作成](#) このページは、Web ページ/アプリケーションにはじめてダッシュボードを表示する際の手順について説明します。

[WEB JAVA SDK の概要](#) Web JAVA SDK の主な概念とアーキテクチャについて。

- [セットアップと構成](#) Web JAVA SDK を使用できるようにするために必要な手順。
- [UpMedia サンプルの実行](#) 提供されている UpMedia サンプルの実行ガイド。

# Reveal SDK の概要

---

- **Desktop SDK** - Reveal Desktop SDK は、Reveal を外部 (ホスト) の Windows アプリケーション (WPF または WinForms) に組み込むことができます。
- **Web SDK** - Reveal Web SDK は、Reveal を外部 (ホスト) Web アプリケーションに組み込みます。Reveal Web ビューアの組み込みが可能な SDK には、2 つのコンポーネントが含まれています。
  - Reveal Web クライアント SDK
  - 2 つの異なるプラットフォーム (.NET と JAVA) でサポートされている Reveal Web サーバー SDK

Reveal の SDK をインストールするときは、.NET Web SDK と Desktop SDK を同時にインストールします。JAVA SDK は、[Maven \(英語\)](#) モジュールのセットとして配布されます。

## 主要機能

Reveal SDK は、アプリケーションに Reveal を組み込むことができるため、ユーザーがダッシュボードを表示したり修正したりすることができます。

Reveal SDK は、Web、Windows WPF および Windows フォームなどの複数のプラットフォームおよびテクノロジーで開発されたアプリケーションに Reveal を統合するために使用できます。

含まれるアプリは Reveal SDK を使用して以下を行うことができます。

- ダッシュボードにインメモリデータを提供します。データが含まれるアプリに既に読み込まれている場合、ダッシュボードを開く前にそれを保存する必要はありません。Reveal は組み込みの InMemory データプロバイダーを使用して、そのデータをダッシュボードの入力として使用できます。
- ダッシュボードが描画される前に設定します。たとえば、現在のユーザーに基づいてデータを取得するためにはデータベースまたはテーブルの名前を変更します。
- ダッシュボードが描画される前、または表示されている間でも、ダッシュボード フィルターを変更します。含まれるアプリは、この機能を使用して、アプリ内のフィルターまたは選択をダッシュボードで可視化されたデータと同期させることができます。
- ダッシュボードでデータポイントが選択されたとき (チャート内のバーがクリックされたときなど) に通知を受け取ります。たとえば、追加情報を表示することまたは詳細を含むページに移動するなどのアクションを発生します。
- その他の多数の機能。

# インストールと要件 (Web)

## Desktop SDK 要件

- Reveal SDK は、.NET バージョン 4.6.2 以降および Visual Studio 2015 以降が必要です。

## ウェブ SDK .NET 要件

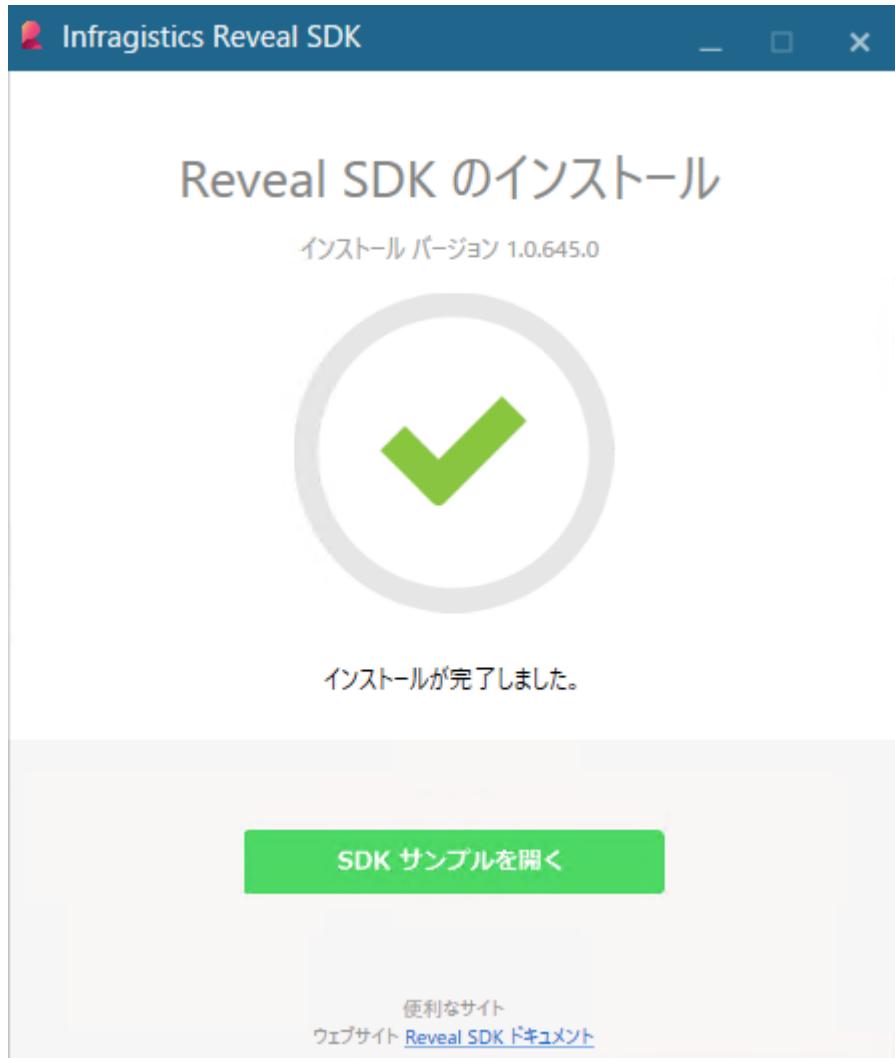
- Reveal Server SDK は、.NET Framework 4.6.2 以降をターゲットとする.NET Core 2.2 以降のサーバーサイドプロジェクトが必要です。

## Reveal デスクトップとウェブ .NET SDK のインストール

ウェブとデスクトップの両方のプラットフォーム用の Reveal SDK は、[こちら](#)からダウンロードしてください。準備ができたら、インストーラーの手順に従います。



インストール完了後、[SDK サンプルを開く] リンクをクリックしてサンプルを表示できます。



## サンプル

サンプルは、%public%\Documents\Infragistics\Reveal\SDK\ にあります。

ここにソリューション ファイル (Reveal.Sdk.Samples.sln) があります。このプロジェクトは、すべての Web、WPF、および WinForms サンプルを組み合わせたものです。

Web の場合、IIS でサンプルを実行し、StartUp プロジェクトを変更するには、ノード パッケージを復元する必要があります。復元するには、ソリューション エクスプローラーでソリューションを右クリックし、[パッケージを復元] を選択します。

## ウェブ SDK JAVA 要件

- [Java SDK \(英語\)](#) 11.0.10 以降を推奨します。
- [Maven \(英語\)](#) 3.6.3 以降を推奨します。

## JAVA SDK のインストール

Reveal Java SDK は、[Maven \(英語\)](#) モジュールのセットとして配布されます。SDK ライブラリを操作するには、Reveal の Maven リポジトリへの参照と、Maven pom.xml ファイルの依存関係を追加する必要があります。詳細については、[セットアップと構成](#)を参照してください。

## サンプル

JAVA SDK の使用方法を示す **UpMedia サンプル**は [GitHub \(英語\)](#) から取得できます。

UpMedia サンプルの実行方法の詳細については、[このリンク](#)を参照してください。

# SDK 用のダッシュボードの取得

---

## 概要

Reveal は、アプリに埋め込むように意図的に設計された ビジネス インテリジェンス プラットフォームです。埋め込み Reveal ダッシュボードは、ビジネスのステータス、メトリクス、またはパフォーマンスなどの情報を迅速かつ簡単に表示します。

Reveal SDK と Reveal アプリの違いを見てみましょう。

**Reveal アプリは**、データに基づく意思決定を迅速に行うことができるビジネス インテリジェンス ツールです。ワークスペースでダッシュボードを作成、表示、共有できます。また、使用するプラットフォーム (Web、デスクトップ、iOS、Android) に関係なく、同じエクスペリエンスを提供します。Reveal アプリの詳細については、[オンライン デモ](#)にアクセスするか、[ヘルプ ドキュメント](#)を参照してください。

**Reveal SDK** を使用することにより、複数のプラットフォームおよびテクノロジーで開発されたアプリに Reveal を埋め込むことができます。また、エンドユーザーが Reveal ダッシュボードの表示や修正を行うことができます。

## 概要

アプリケーションにダッシュボードを表示するには、はじめにプラットフォーム上の Reveal アプリケーションを使用してダッシュボードを作成する必要があります。

Reveal SDK を使用して最初からダッシュボードを作成することができますが、最初に SDK を評価するときに推奨されるアプローチは、アプリで作成されたダッシュボードから始める方法です。

## ダッシュボード ファイルの取得

[ダッシュボードを作成した後](#)、取得するには以下の手順に従ってください。

### 1. Reveal アプリでダッシュボードを開く

任意のプラットフォームに Reveal アプリケーションをインストールした後は、独自のダッシュボードを作成するか、アプリに付属のサンプル ダッシュボードのいずれかを使用できます。

### 2. エクスポート オプションにアクセス

オーバーフロー メニューに移動し、[エクスポート]、[ダッシュボード] の順に選択します。これにより、後で SDK を統合するときにアプリケーションで使用する拡張子 .rdash のファイルが生成されます。

# Campaigns

Date Filter  
過去 12か月 ▼

CampaignID  
すべて ▼

## Spend vs Bu...

48,624 +4.09%▲  
Budget: 年度実行

## Website Traf...

252,327 -9.83%▼  
前年比

## Conversions

35,163 -7.43%▼  
前年比

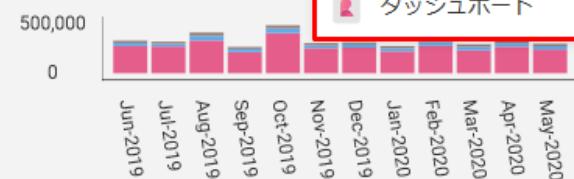
## Actual Spend vs Budget

Spend    Budget



## Website Traffic Break

Paid Traffic    Organic Traffic

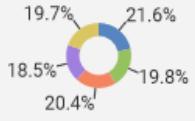


## Conversions

## Conversions



## Conversions...



ダッシュボード ファイルは、電子メール (Android および iOS) を介してエクスポートすることも、コンピュータ上の .rdash ファイル (デスクトップおよび Web) としてエクスポートすることもできます。

← エクスポート形式

画像

PowerPoint

PDF

Excel

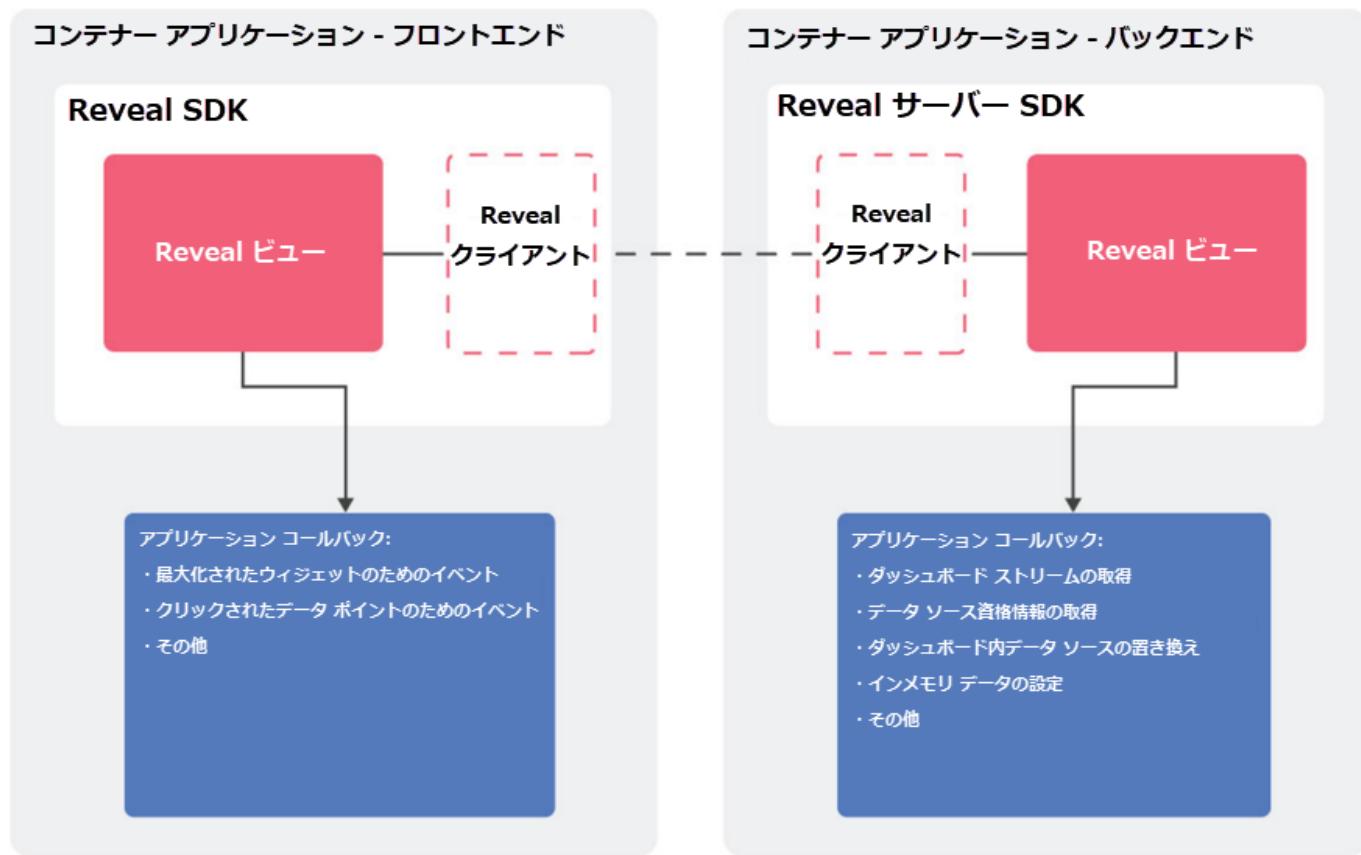
ダッシュボード

# Web SDK の概要

Web アプリケーションに Reveal を組み込む場合、2 つのコンポーネントが常に関係しているため、アーキテクチャはネイティブ アプリよりも少々複雑になります。

- **Reveal クライアント SDK:** Web アプリケーションに統合する必要がある JavaScript ライブラリのセット。現在サポートされているフレームワークは、jQuery、Angular、および React です。
- **Reveal サーバー SDK:** サーバー アプリケーションに統合されるサーバー側のコンポーネント。現在、これは .NET Core 3.1 以降を対象とする ASP.NET Core アプリケーションです。

以下の図では、Reveal Web SDK を埋め込んだ Web アプリケーションのアーキテクチャを表示しています。



上記のように、SDK はネイティブ アプリケーションとほとんど同様に機能します。違いは、一部のコールバックがクライアント側で呼び出され (データポイントがクリックされたときに送信されるイベントなど)、その他はサーバー側で呼び出される (ダッシュボードのロードまたはメモリ内データの提供のためのコールバックなど) 点です。

## クライアント側とサーバー側の部分のホスト (異なるサーバーを利用)

クライアント側とサーバー側のパートを個別に、たとえば異なる URL でホストできます。

これには、以下のようにウィンドウ オブジェクトのプロパティを設定します。

```
$.ig.RevealSdkSettings.setBaseUrl("{back-end base url}");
```

プロパティを正しく設定するには、URL の末尾にスラッシュ記号が必要です。

このプロパティは、[\*.ig.RevealView\* のインスタンス化](#)の前に設定します。

# セットアップと構成 (Web)

## 前提条件

Reveal Server SDK には、.NET Core 2.2+ または .NET Framework 4.6.2 以降の ASP MVC アプリケーションプロジェクトが必要です。

NET Framework 4.6.2 以降をターゲットとする場合、Reveal Server SDK には win7-x64 ランタイム環境がサポートされます。Web プロジェクトをデバッグするには、win7-x64 互換の *RuntimeIdentifier* プラットフォームを追加する必要があります。

```
<PropertyGroup>

  <TargetFramework>net462</TargetFramework>

  <RuntimeIdentifier>win7-x64</RuntimeIdentifier>

</PropertyGroup>
```

## セットアップと構成の概要

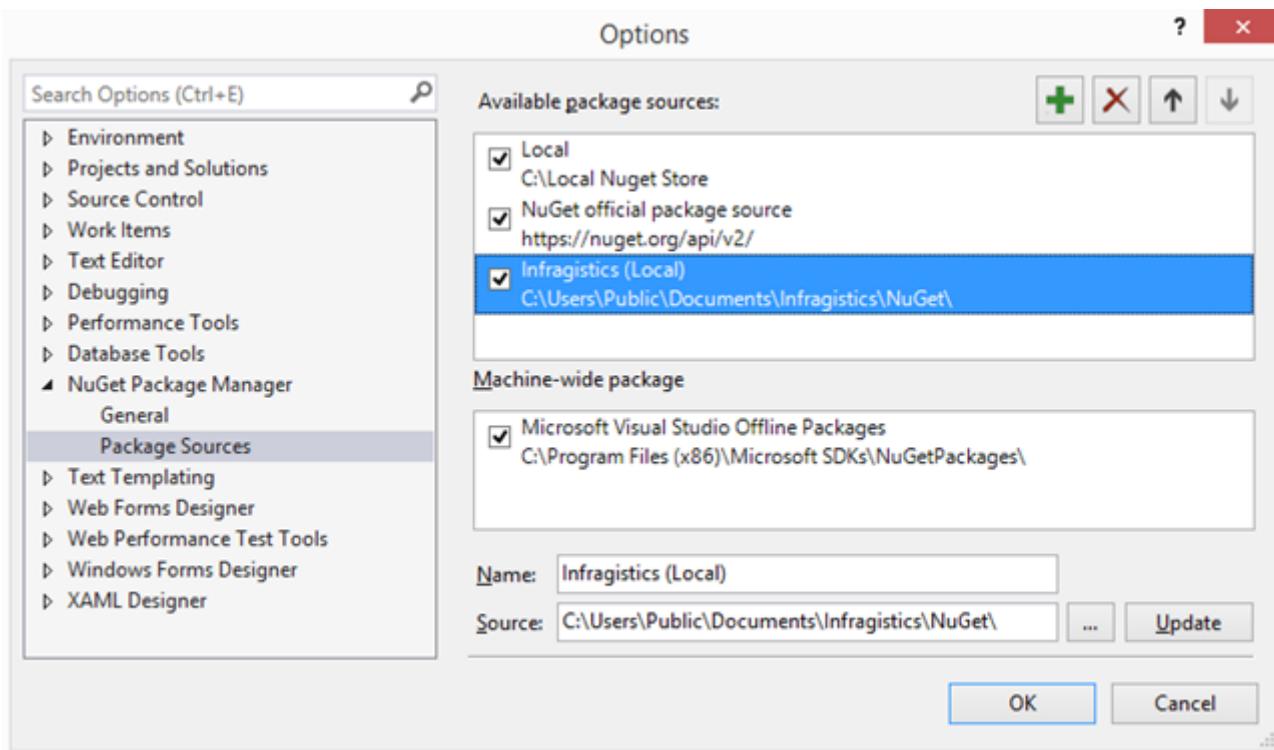
以下は、Reveal Web Server SDK 設定の手順です。

1. [アセンブリへの参照を追加し、依存関係パッケージをインストール](#)
2. [サーバー コンテキストを定義](#)
3. [サーバー SDK を初期化](#)
4. [サーバー側画像生成の有効化.](#)

### 1. アセンブリと依存関係パッケージの準備

アセンブリへの参照を追加して依存関係パッケージをインストールするには、**NuGet** パッケージ マネージャの使用をお勧めします。プロジェクトのセットアップには、**Reveal.Sdk.Web.AspNetCore** (トライアル NuGet パッケージをインストールする方法が最も簡単です)。

Reveal SDK をインストールすると、%public%\Documents\Infragistics\NuGet を指す *Infragistics (Local)* と呼ばれる新しい NuGet パッケージソースが **nuget.config** に追加されます。



Infragistics (Local) フィードがインストーラーによって正しく設定されていることを確認後:

- **Reveal.Sdk.Web.AspNetCore** NuGet パッケージをプロジェクトにインストールします。
- NuGet パッケージ参照を System.Data.SQLite バージョン 1.0.111 以降に追加します。

ビルドに問題がある場合は、この[リンク](#)を参照してください。

## 2. サーバー コンテキストの定義

必要な DLL を参照したら、**RevealSdkContextBase** 抽象クラスを継承するクラスを作成する必要があります。このクラスは、Reveal SDK をアプリケーション内で実行できるようにし、SDK を操作するためのコールバックを提供します。

```
using Reveal.Sdk;
public class RevealSdkContext : RevealSdkContextBase
{
    public override IrvDataSourceProvider DataSourceProvider => null;

    public override IrvDataProvider DataProvider => null;

    public override IrvAuthenticationProvider AuthenticationProvider => null;

    public override Task<Dashboard> GetDashboardAsync(string dashboardId)
    {
        var fileName = $"C:\\Temp\\{dashboardId}.rdash";
        var fileStream = new FileStream(fileName, FileMode.Open, FileAccess.Read);
        return Task.FromResult(new Dashboard(fileStream));
    }

    //このコールバックは、RevealView オブジェクトのクライアント側に onSave イベントがインストールされていない場合のみ使用されます。
}
```

```
// 詳細については、Web クライアント SDK のドキュメントをご覧ください。
public override Task SaveDashboardAsync(string userId, string dashboardId,
Dashboard dashboard)
{
    return Task.CompletedTask;
}
```

上記の実装は、C:\Temp フォルダからダッシュボードをロードし、*dashboardId* 変数に依存する .rdash ファイルを検索します。アプリケーションでは、他のディレクトリやデータベースから、あるいは組み込みリソースからもダッシュボードを読み込めるよう変更します。

[!NOTE] **null を返すプロパティ**: 最初の 3 つのプロパティ *DataSourceProvider*、*DataProvider*、および *AuthenticationProvider* はすべて *null* を返すように実装されています。このガイドでは、これらのプロパティのためにすべてのインターフェイスをインストールする方法について説明します。

### 3. サーバー SDK の初期化

**Startup.cs** のアプリケーションの **ConfigureServices** メソッド *RevealEmbedSettings* クラスを渡し、サービス拡張メソッド *AddRevealServices* を呼び出します。

*AddRevealServices* 拡張メソッドは **Reveal.Sdk** 名前空間で定義されているため、ユーザーを追加する必要があります。また、以下に示すように **CachePath** プロパティも設定してください。

```
services.AddRevealServices(new RevealEmbedSettings
{
    LocalFilePath = @"C:\Temp\Reveal\DataSource",
    CachePath = @"C:\Temp"
}, new RevealSdkContext());
```

[!NOTE] **LocalFilePath** は、ダッシュボードデータソースとしてローカルの Excel ファイルまたは CSV ファイルを使用しており、*RevealSdkContext* クラスが前述のように *RevealSdkContextBase* を継承している場合にのみ必要です。

MVC サービスを追加するときに **AddReveal** 拡張メソッドを呼び出すことによって、Reveal エンドポイントを追加できます。以下はコードスニペットです。

```
services.AddMvc().AddReveal();
```

*AddRevealServices* と同様に、*AddReveal* メソッドは *Reveal.Sdk* で定義されているため、ディレクティブを使用してください。

### 4. サーバー側画像生成の有効化

**画像エクスポート**機能 (プログラム上およびユーザー操作の両方により) を使用するには、以下の手順を実行する必要があります。

1. <InstallationDirectory>\SDK\Web\JS\Server から以下の 3 つのファイルを取得します。

- package.json
- packages-lock.json
- screenshoteer.js

2. ファイルをプロジェクトのルート レベル (「wwwroot」の親フォルダー) にコピーします。

3. **npm** (Node.js のパッケージ マネージャー) がインストールされていることを確認してください。

画像エクスポート機能が必要ない場合は、ファイルをプロジェクトにコピーする必要はありません。ただし、プロジェクトをビルドする場合、*npm* が見つからない警告メッセージが表示され、プロジェクトが正しく動作しません。

このエラーを解決するには、以下のプロパティをプロジェクトに追加します。

```
<PropertyGroup>
  <DisableRevealExportToImage>true</DisableRevealExportToImage>
</PropertyGroup>
```

## NuGet 使用時のビルドの問題

**SQLite.Interop.dll** に関するデプロイメントの問題を処理するために、NuGet パッケージでカスタムの .targets ファイルが使用されています。

ビルドに問題がある場合は、プロジェクトに次のプロパティを追加してこの動作を無効にできます。

```
<DisableSQLiteInteropFix>true</DisableSQLiteInteropFix>
```

## セットアップと構成 (クライアント)

以下は、Reveal Web Client SDK を設定するための手順です。

1. [依存関係の確認](#)
2. [Web Client SDK の参照](#)
3. [Web Client SDK のインスタンス化](#)

### 1. 依存関係の確認

Reveal Web Client SDK には、サードパーティの参照が 2 つあります。

- [jQuery](#) 2.2 またはそれ以上
- [Day.js](#) 1.8.15 またはそれ以上
- [Quill RTE](#) 1.3.6 またはそれ以上
- [Marker Clusterer](#) 3 またはそれ以上
- [Google Maps](#) 3 またはそれ以上

## 2. Web Client SDK の参照

Web ページで **\$.ig.RevealView** コンポーネントを有効にするには、いくつかの クリプトを含める必要があります。これらのスクリプトは Reveal Web Client SDK の一部として提供されます。

```
<script src="~/Reveal/infragistics.reveal.js"></script>
```

JavaScript ファイルは <InstallationDirectory>\SDK\Web\JS\Client にあります。

## 3. Web Client SDK のインスタンス化

ダッシュボードのプレゼンテーションは、Web Client SDK を介してネイティブに処理されます。

以下の手順に従って作業を開始します。

1. id を指定して `<div />` 要素を定義し、**\$.ig.RevealView** コンストラクターを呼び出します。

[!NOTE] サーバー側とクライアント側のパートを個別にホスト 個別のサーバーでクライアント 側とサーバー側のパートをホストする場合は、次の手順を続行する前に[こちら](#)を参照してください。

2. **\$.ig.RVDashboard.loadDashboard** を呼び出して `dashboardId` と成功およびエラー ハンドラを指定します。
3. 成功ハンドラーで、ダッシュボードが描画される DOM 要素のセレクターを渡すことにより、**\$.ig.RevealView** コンポーネントをインスタンス化します。最後に、取得したダッシュボードを使用し、**\$.ig.RevealView** のダッシュボード プロパティに設定する必要があります。

### サンプルコード

```
<!DOCTYPE html>
<html>
  <head>
    :
    <script type="text/javascript">
      var dashboardId = "dashboardId";

      $.ig.RVDashboard.loadDashboard(
        dashboardId,
        function (dashboard) {
          var revealView = new $.ig.RevealView("#revealView");
          revealView.dashboard = dashboard;
        },
        function (error) {
          //ここで発生する可能性があるエラーを処理します。
        }
      );
    </script>
  </head>
  <body>
```

```
<div id="revealView" style="height:500px;" />
</body>
</html>
```

# Web SDK をはじめて使用する

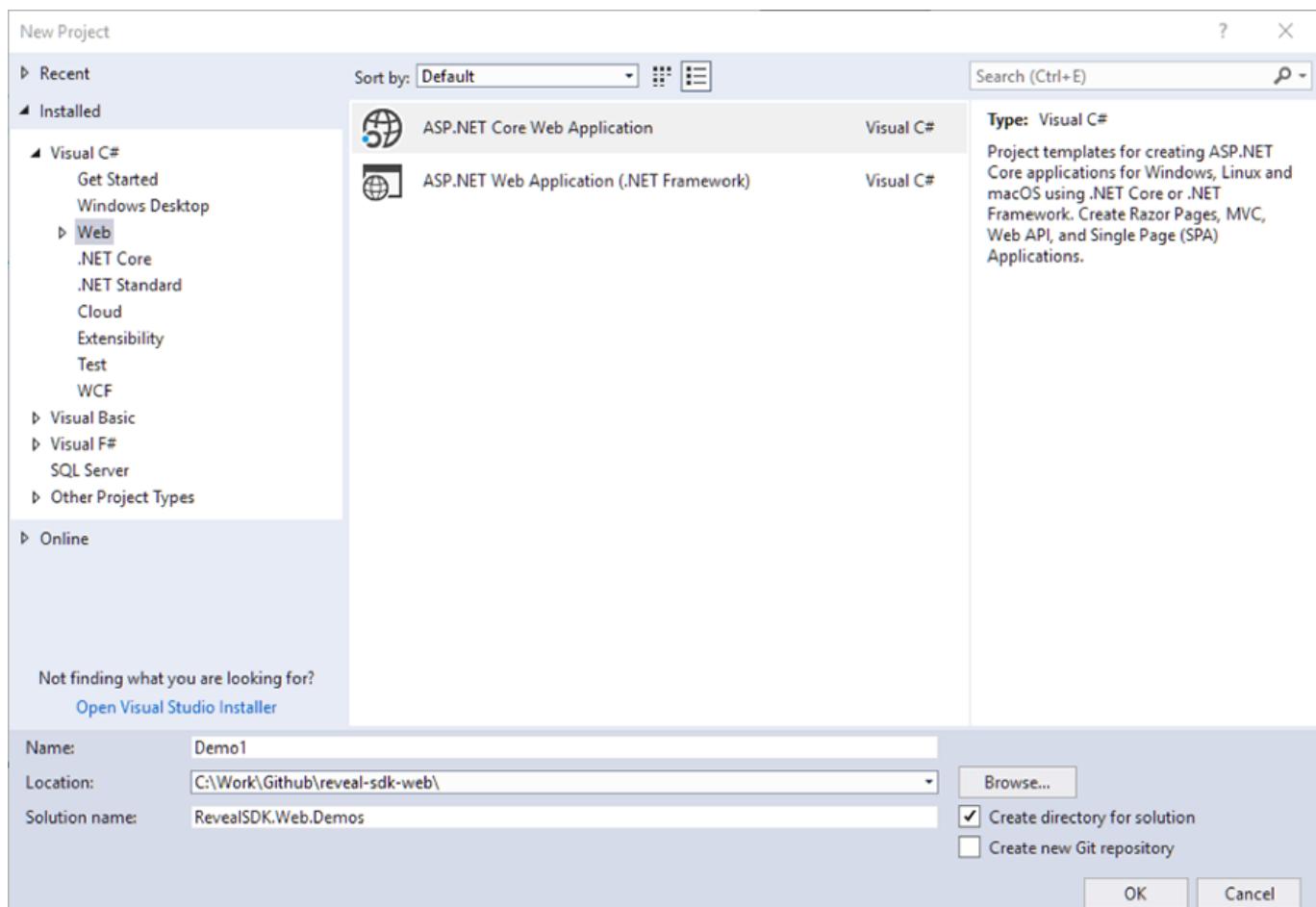
このページは、Web ページ/アプリケーションにはじめてダッシュボードを表示する際の手順について説明します。

## 手順

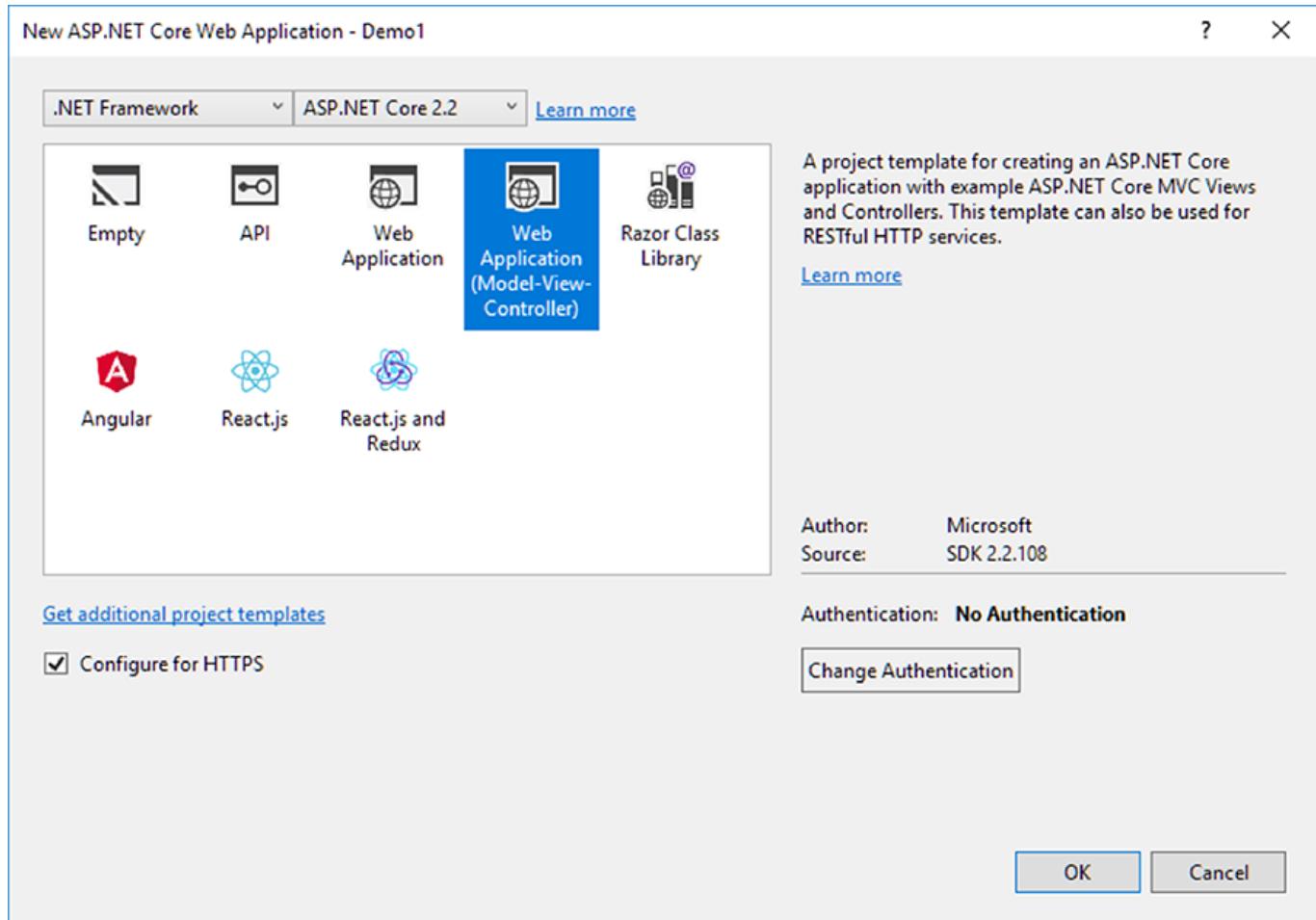
1. プロジェクトを作成する
2. Reveal SDK のインストール
3. サーバー構成の設定
4. クライアントアプリケーションに Reveal を埋め込む
5. Reveal フォントの使用
6. クライアントアプリケーションのスタイル設定

## 手順 1 - プロジェクトを作成する

Visual Studio 2017 を開き、ASP.NET Core Web アプリケーションの新しいプロジェクトを作成します。

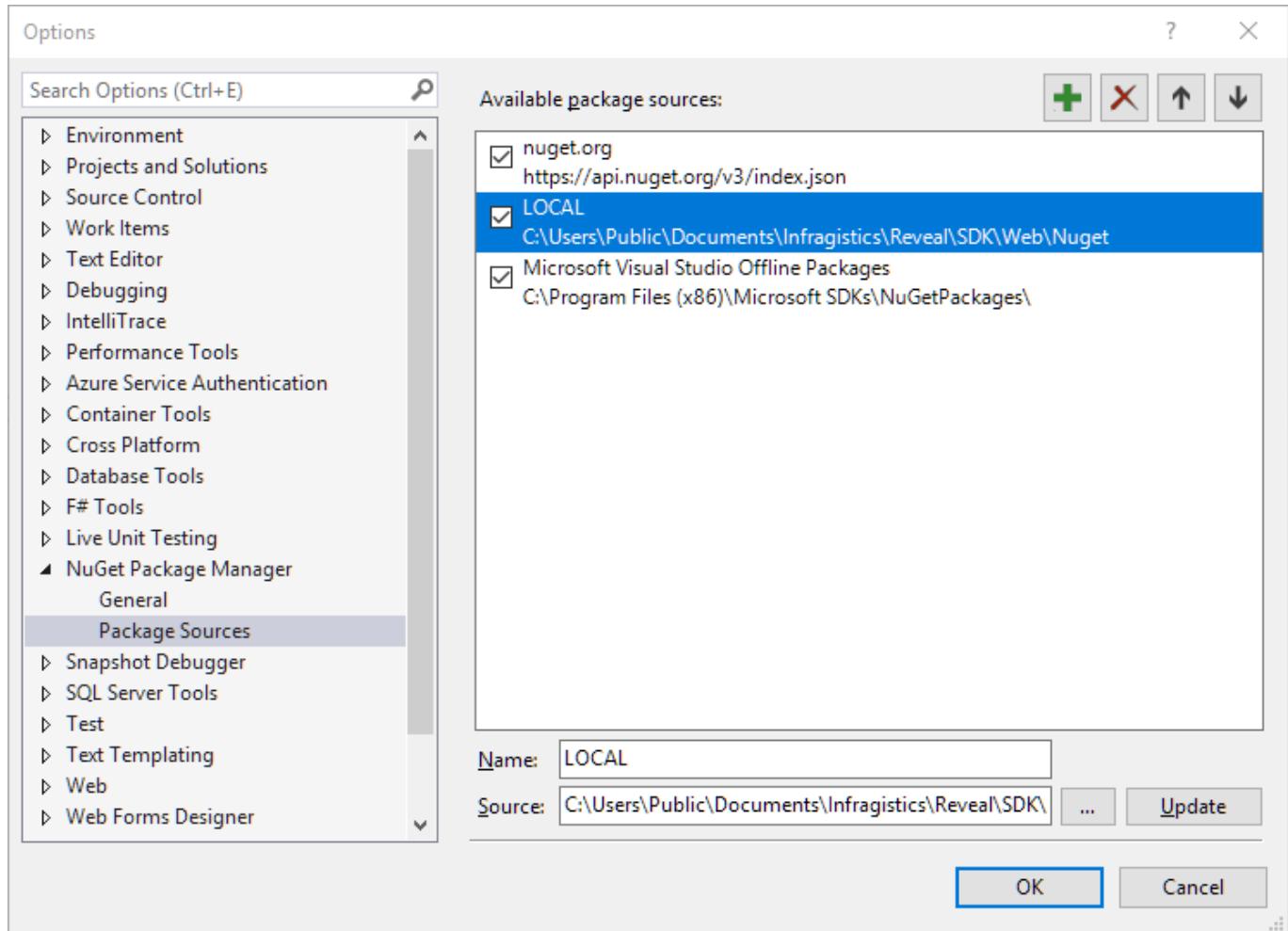


以下のように .NET Framework および ASP.NET Core 2.2 を選択します。



## 手順 2 - Reveal SDK のインストール

<https://www.revealbi.io/jp> から Infragistics Reveal SDK をダウンロードし、システムにインストールします。Visual Studio で ツール > オプション > Nuget パッケージマネージャー > パッケージ ソースを選択します。インストールされた SDK の Nuget フォルダーを指す新しいソースを追加します。



その後、パッケージ ソースを追加したものに変更して Nuget をインストールできます。

NuGet - Solution X

Browse    Installed    Updates    Consolidate

Search (Ctrl+L) ↻ ⟳  Include prerelease

Manage Packages for Solution  
Package source: LOCAL ⚙

**Infragistics.Reveal.Sdk.Web.AspNetCore.Trial** by Reveal Team v1.0.706  
Infragistics.Reveal.Sdk.Web.AspNetCore

Versions - 0

<input type="checkbox"/> Project
<input type="checkbox"/> Demo1

Installed: not installed Uninstall

Version: Latest stable 1.0.706 Install

Options

Description  
Reveal SDK for Web - ASP .NET Core [Trial]

Version: 1.0.706  
Owner(s): Infragistics Inc.  
Author(s): Reveal Team

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

Do not show this again

Preview Changes X

Visual Studio is about to make changes to this solution. Click OK to proceed with the changes listed below.

Demo1

**Updates:**  
Newtonsoft.Json.11.0.2 -> Newtonsoft.Json.12.0.1

**Installing:**

- Antlr4.Runtime.Standard.4.7.1
- Infragistics.Reveal.Sdk.Web.AspNetCore.1.0.661
- Microsoft.Azure.Amqp.2.3.0
- Microsoft.Azure.ServiceBus.3.1.0
- Microsoft.Azure.Services.AppAuthentication.1.0.1
- Microsoft.EntityFrameworkCore.2.0.3
- Microsoft.EntityFrameworkCore.Design.2.0.3
- Microsoft.EntityFrameworkCore.Relational.2.0.3
- Microsoft.EntityFrameworkCore.SqlServer.2.0.3
- Microsoft.EntityFrameworkCore.Tools.2.0.3
- Microsoft.IdentityModel.Clients.ActiveDirectory.3.17.2
- Microsoft.IdentityModel.Logging.5.2.2
- Microsoft.IdentityModel.Tokens.5.2.2
- Remotion.Linq.2.1.1
- System.Data.SqlClient.4.4.3
- System.Data.SQLite.Core.1.0.108
- System.IdentityModel.Tokens.Jwt.5.2.2

Do not show this again OK Cancel

## 手順 3 - サーバー構成の設定

プロジェクトに新しい Reveal SDK フォルダーを作成し、**RevealSdkContextBase** 抽象クラスを実装する **RevealSdkContext.cs** クラスを追加します。

```
using Reveal.Sdk;
using System;
using System.IO;
using System.Reflection;
using System.Threading.Tasks;

namespace Demo1.RevealSDK
{
    public class RevealSdkContext : RevealSdkContextBase
    {
        public override IRVDataSourceProvider DataSourceProvider => null;

        public override IRVDataProvider DataProvider => null;

        public override IRVAuthenticationProvider AuthenticationProvider =>
null;

        public override Task<Dashboard> GetDashboardAsync(string dashboardId)
        {
            var dashboardFileName = dashboardId + ".rdash";
            var resourceName = $"Demo1.Dashboards.{dashboardFileName}";
            var assembly = Assembly.GetExecutingAssembly();
            return Task.FromResult(new
Dashboard(assembly.GetManifestResourceStream(resourceName)));
        }

        public override Task SaveDashboardAsync(string userId, string
dashboardId, Dashboard dashboard)
        {
            return Task.CompletedTask;
        }
    }
}
```

上記のコードでは、**Demo1.Dashboards** がダッシュボード ファイルが含まれる場所を示しているため、プロジェクトに新しいダッシュボード フォルダーを作成し、ここでは空のままにします。

これを行うには、**Startup.cs** の **ConfigureServices** メソッドに以下のコードを追加します。

```
services.AddRevealServices(new RevealEmbedSettings
{
    CachePath = @"C:\Temp"
```

```
}, new RevealSdkContext());  
  
services.AddMvc().AddReveal();
```

同じファイルに必要な参照を追加します。

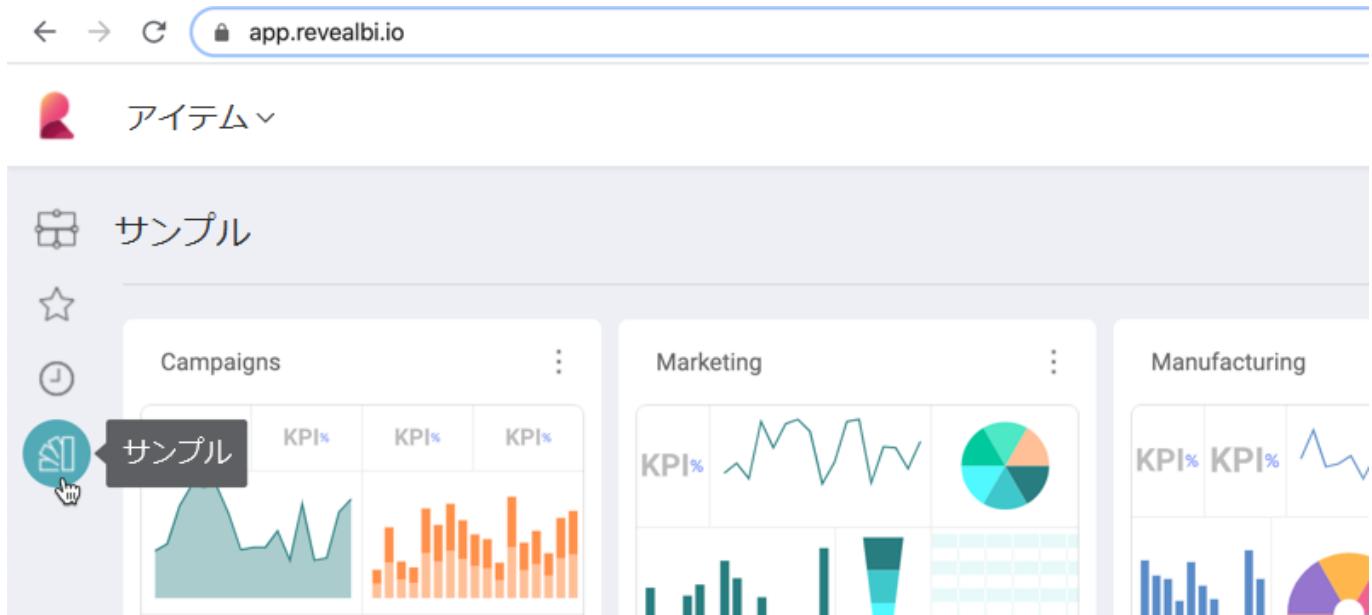
```
using Demo1.RevealSDK;  
using Reveal.Sdk;
```

問題が発生した場合は、サーバー SDK - [セットアップと構成](#) トピックを参照してください。

## 手順 4 - クライアント アプリケーションに Reveal を埋め込む

はじめにダッシュボードを準備します。このデモでは、Reveal のサンプル セクションの **Marketing ダッシュボード**を使用できますが、テーマは異なります。ダッシュボードを開き、編集モードに入ります。

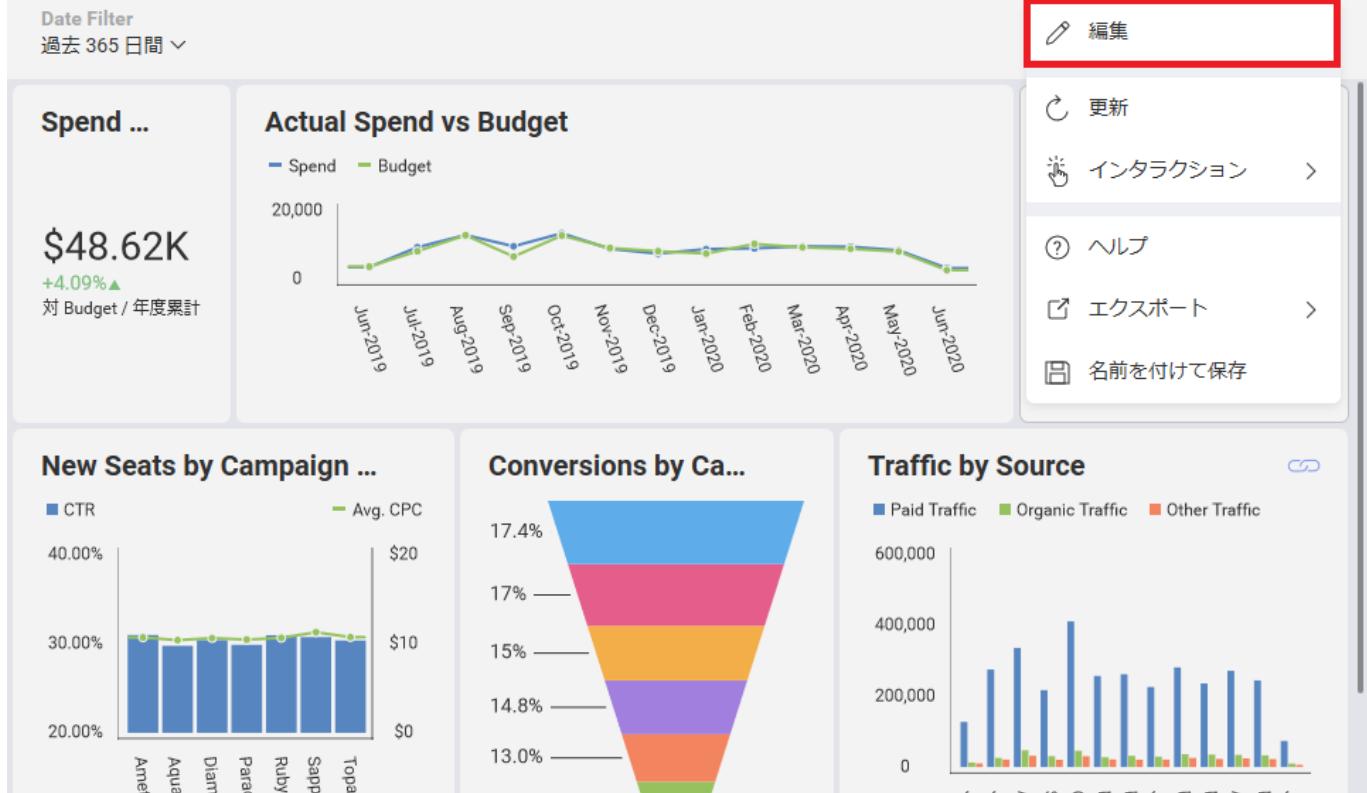
Reveal アプリ (<https://app.revealbi.io>) を開き、**サンプル**に移動します。



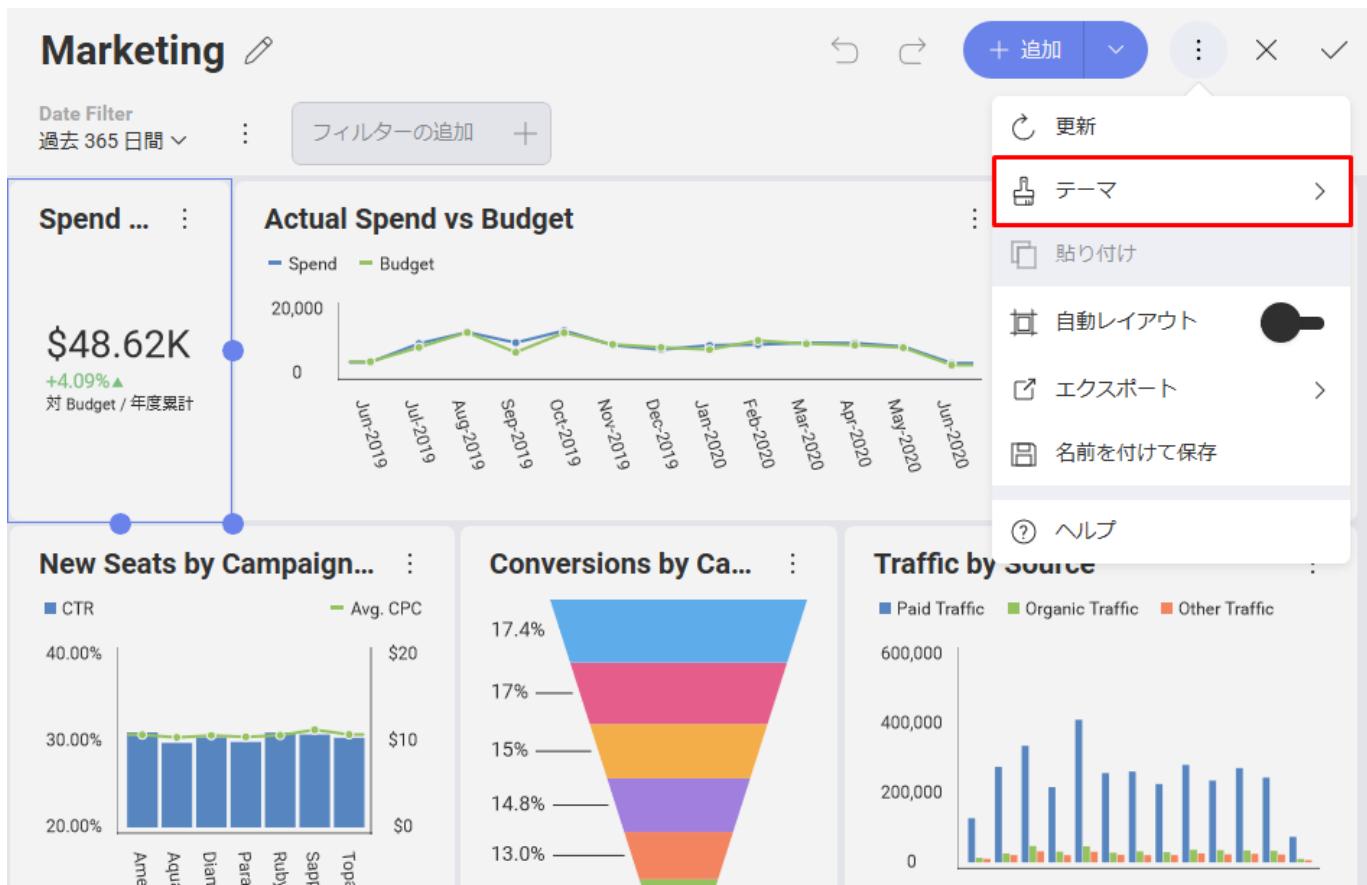
The screenshot shows the Reveal BI application interface. At the top, there is a browser-style address bar with a lock icon and the URL [app.revealbi.io](https://app.revealbi.io). Below the address bar, there is a navigation menu with icons for profile, items, sample, star, and refresh. The main area is titled "サンプル". It displays three dashboards: "Campaigns", "Marketing", and "Manufacturing". The "Marketing" dashboard is currently selected and shown in detail. It features a line chart with a green line, a bar chart with orange bars, and a pie chart. A tooltip "サンプル" is visible over the line chart area. A hand cursor icon is positioned over the "Marketing" dashboard title.

Marketing ダッシュボードを選択し、**編集モード**に入ります。

# Marketing

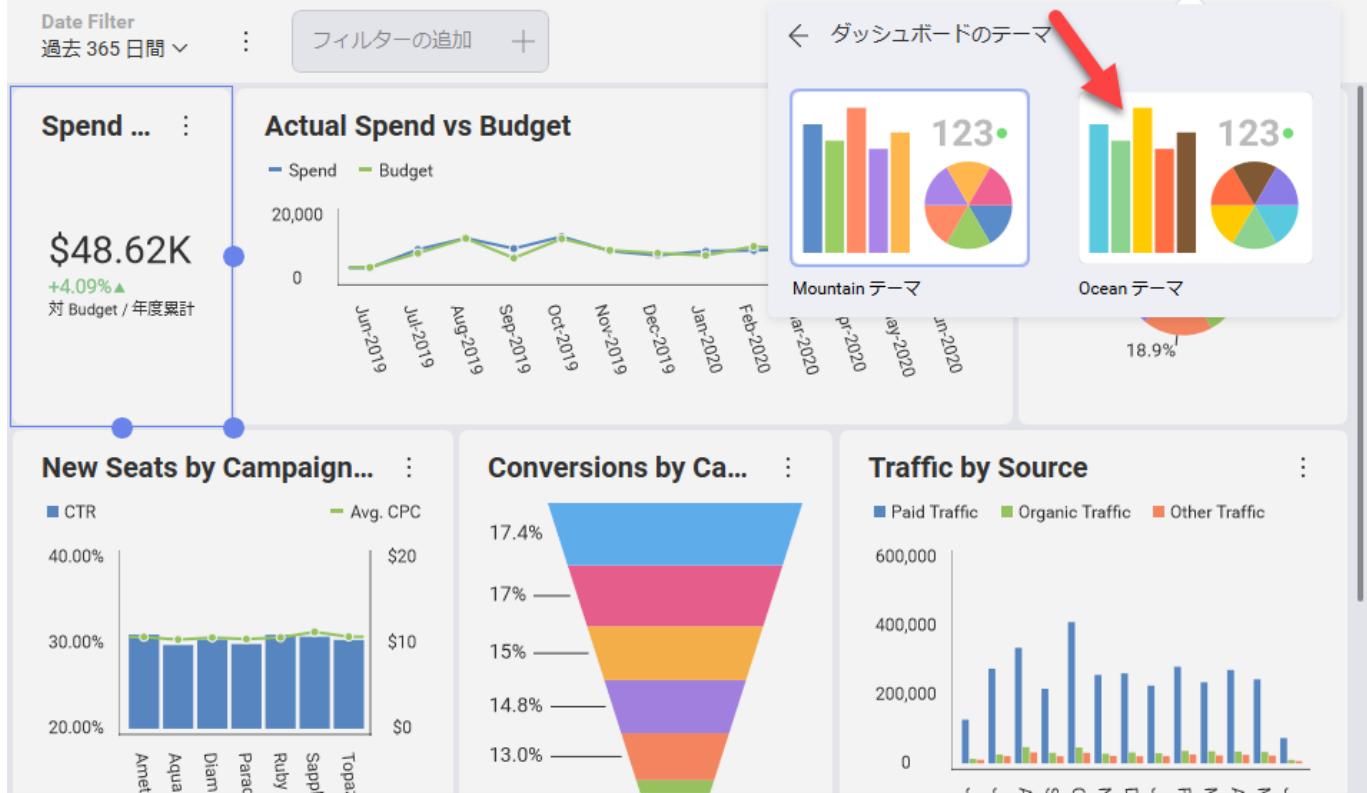


編集モードに入った後に [テーマ] ボタンをクリックします。



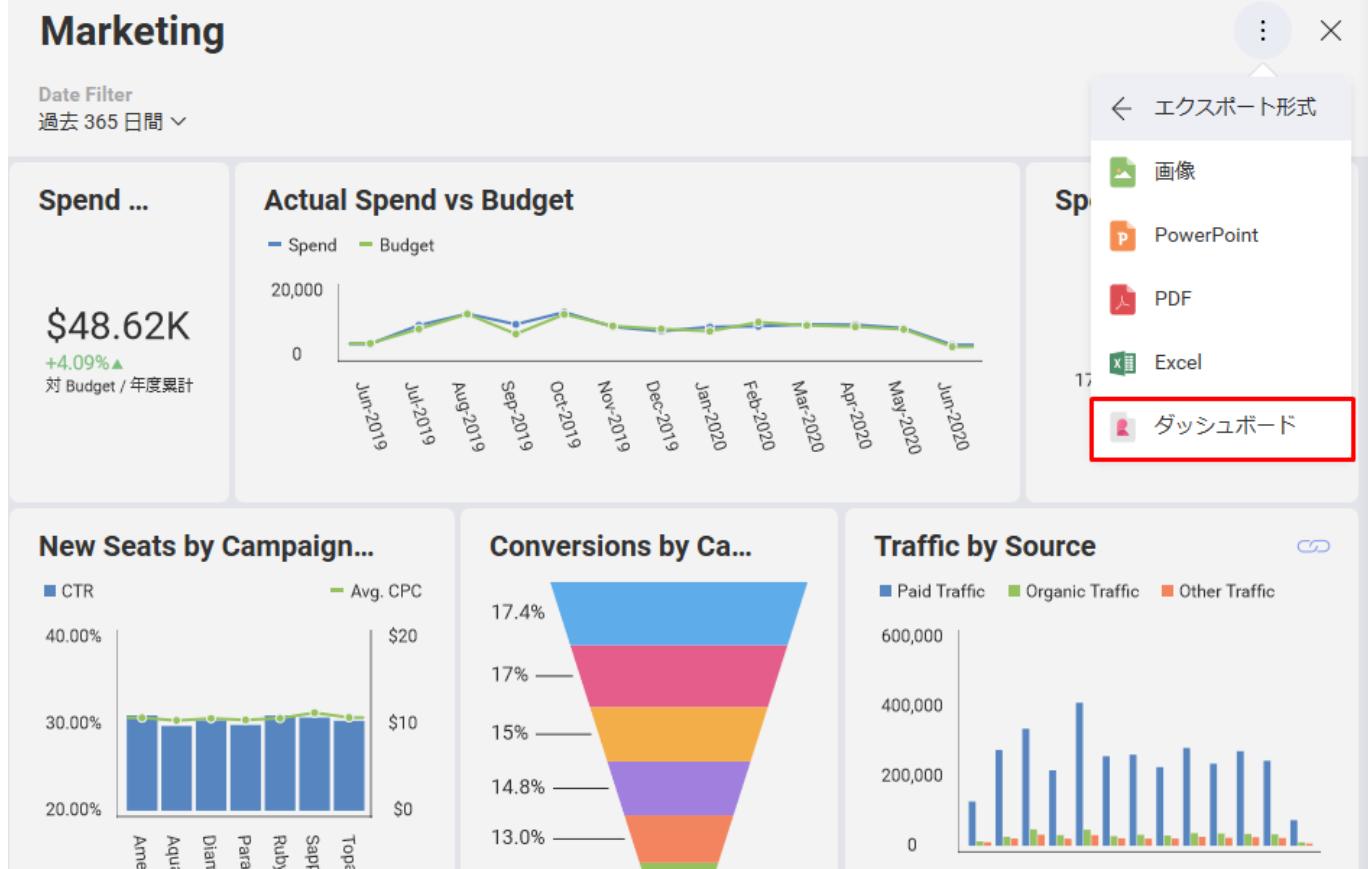
[Ocean テーマ] を選択します。

## Marketing

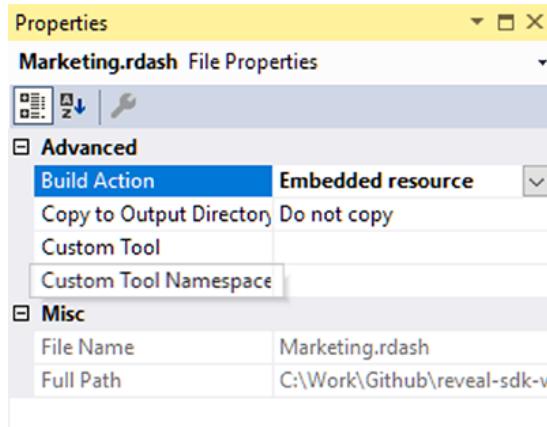


変更したダッシュボードを保存し、エクスポートします。

[!NOTE] Marketing ダッシュボードは、Reveal アプリのサンプルの一部であるため、通常のダッシュボードと同じ方法で保存することはできません。代わりに、[名前を付けて保存] を使用して場所を選択する必要があります。



**Marketing.rdash** ダッシュボード ファイルを手順 3 で作成したダッシュボード フォルダーに移動し、Visual Studio でこのアイテムのビルド アクションを埋め込みリソースに設定します。



#### Build Action

How the file relates to the build and deployment processes.

次に、新しいページ *Marketing.cshtml* を追加して、ダウンロードしたダッシュボードを可視化します。

```
@{
    ViewData["Title"] = "Marketing";
}

@section Scripts
{
    <script type="text/javascript">
        // #revealView 要素にダッシュボードを読み込む
    </script>
}

<section>
    <div id="revealView" style="height:800px;"></div>
</section>
```

次に、**HomeController.cs** に新しいアクション メソッドを追加します。

```
public IActionResult Marketing()
{
    return View();
}
```

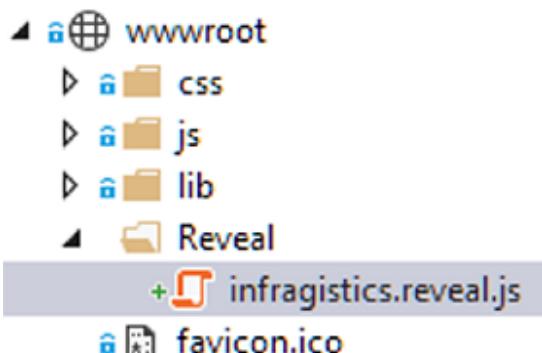
**Layout.cshtml** にある Reveal のサードパーティの依存関係について、スクリプトと css ファイルへの参照をいくつか追加しましょう:

```
<script src="https://unpkg.com/dayjs"></script>
<link rel="stylesheet"
      href="https://code.jquery.com/ui/1.10.2/themes/smoothness/jquery-ui.css" />
```

```
<script type="text/javascript">
src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.2.1.min.js"></script>
<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.min.js"></script>
<script src="https://cdn.quilljs.com/1.3.6/quill.js"></script>
<link href="https://cdn.quilljs.com/1.3.6/quill.snow.css" rel="stylesheet">
```

続行するには、プロジェクトの wwwroot フォルダーに新しい Reveal フォルダーを作成します。

**infragistics.reveal.js** をコピーします。このファイルは、Reveal SDK の  
<InstallationDirectory>\SDK\Web\JS\Client にあります。



そして、Day.js のスクリプトの後に **\_Layout.cshtml** でこのライブラリを参照します。

```
<script src="~/Reveal/infragistics.reveal.js"></script>
```

同じファイル内のフッター セクションも削除し、新しいページのナビゲーションにリンクを追加します。

```
<li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Marketing">Marketing</a>
</li>
```

Marketing.cshtml のスクリプトを、ダッシュボードをロードするためのロジックで更新しましょう。

```
var dashboardId = "Marketing.rdash";

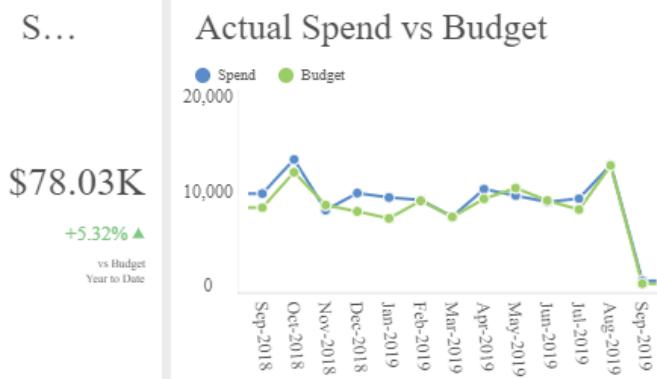
$.ig.RVDashboard.loadDashboard(dashboardId, function (dashboard) {
    var revealView = new $.ig.RevealView("#revealView");
    revealView.dashboard = dashboard;
}, function (error) {
    //ここで発生する可能性があるエラーを処理します。
});
```

最後に、Web ページを実行すると、ダッシュボードが表示されます。

## Marketing

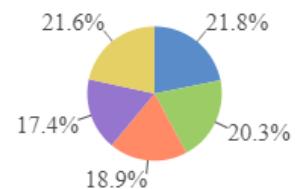
Date Filter  
Last 365 days ▾

S...



Spend by Ter...

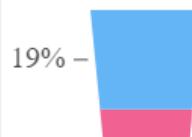
Americas APAC EMEA  
India Japan



Traffic by Source



Con...



New Seats b...

CampaignID	↑↓
Amethyst	🔗

問題が発生した場合は、クライアント SDK [セットアップと構成](#)トピックを参照してください。

## 手順 5 - Reveal フォントの使用

Reveal アプリは Roboto フォントを使用します。アプリと同じ外観を実現するには、

<https://fonts.google.com/specimen/Roboto> からフォントをダウンロードし、次の TTF ファイルをプロジェクトの `wwwroot/css` フォルダーへコピーします。

- Roboto-Regular.ttf
- Roboto-Bold.ttf
- Roboto-Light.ttf
- Roboto-Medium.ttf

次に、`site.css` に以下のように参照を追加します。

```
@font-face {
    font-family: "Roboto-Regular";
    src: url("Roboto-Regular.ttf");
}

@font-face {
    font-family: "Roboto-Bold";
    src: url("Roboto-Bold.ttf");
}
```

```
@font-face {
    font-family: "Roboto-Light";
    src: url("Roboto-Light.ttf");
}

@font-face {
    font-family: "Roboto-Medium";
    src: url("Roboto-Medium.ttf");
}
```

フォントの読み込みを改善するには、infragistics.reveal.js 参照の横にある **\_Layout.cshtml** で Google Web Font Loader への参照を追加します。

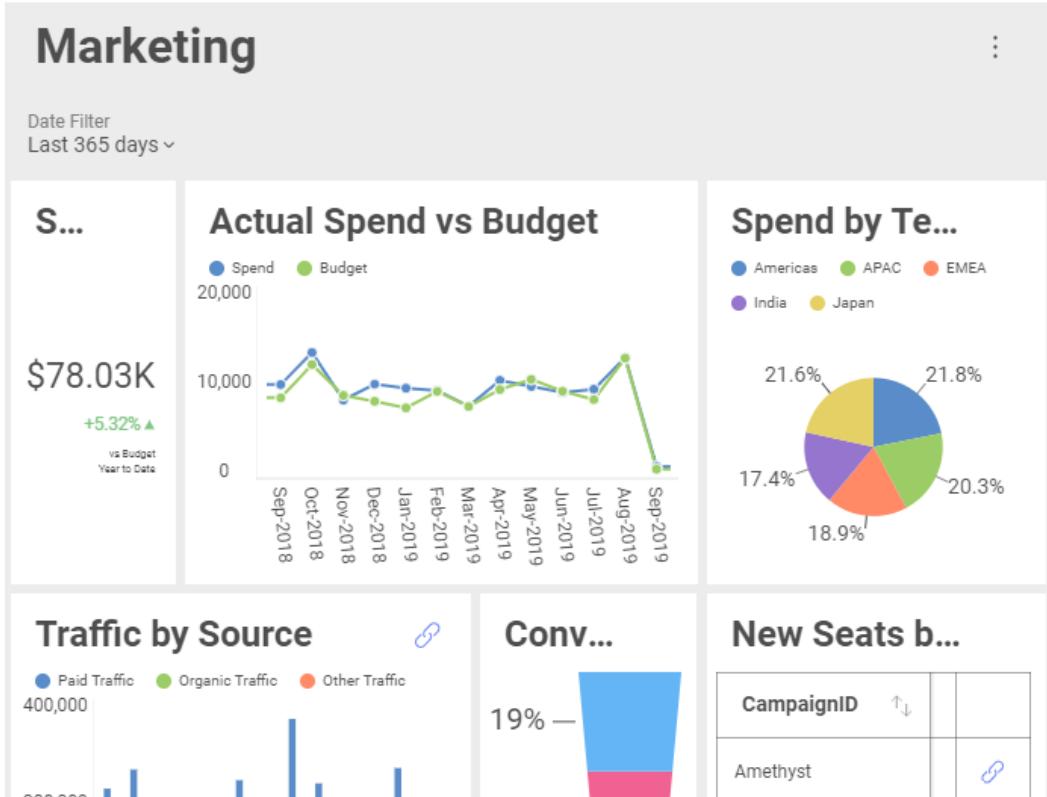
```
<script src="https://ajax.googleapis.com/ajax/libs/webfont/1/webfont.js"></script>
```

最後に、**Marketing.cshtml** ページのスクリプト セクションを変更して、フォントローダーを利用します。

```
WebFont.load({
    custom: {
        families: ['Roboto-Regular', 'Roboto-Bold', 'Roboto-Light', 'Roboto-Medium'],
        urls: ['/css/site.css']
    },
    active: function () {
        var dashboardId = "Marketing.rdash";

        $.ig.RVDashboard.loadDashboard(dashboardId, function (dashboard) {
            var revealView = new $.ig.RevealView("#revealView");
            revealView.dashboard = dashboard;
        }, function (error) {
            //ここで発生する可能性があるエラーを処理します。
        });
    },
});
```

結果は以下のようになります。



## 手順 6 - クライアントアプリケーションのスタイル設定

デフォルトのテンプレートを使用する代わりに、クライアントアプリケーションのスタイルを設定できます。

**HomeController.cs** からプライバシーを削除し、Marketing にリダイレクトするようにインデックスを変更します。

```
public IActionResult Index()
{
    return RedirectToAction("Marketing");
}
```

また、Index.cshtml および Privacy.cshtml ファイルは使用されないため削除してください。Marketing.cshtml の <div> 要素のスタイル設定を削除します。

wwwroot に新しい img フォルダーを作成し、そこへ logo.png をコピーします。これは、[こちら](#)からダウンロードできます。

[Layout.cshtml](#) で、以下の変更を行います。

- タイトルを Demo1 から Overview へ変更します。
- ヘッダーの後の div を削除します。
- ロゴ、セパレータ、タイトルを追加してヘッダーを変更します。

```
<header>
  <div class="header">
    <img class="logo" src "~/img/logo.png" alt="logo" />
    <span class="line" />
    <h1>Overview</h1>
  </div>
</header>
```

**site.css** で、Roboto フォント用に追加したものを除くすべてのスタイルを削除し、ヘッダーにスタイルを追加します。

```
/* Header
----- */

header {
  display: flex;
  width: 100%;
  height: 70px;
  box-shadow: 0 4px 12px 0 rgba(0, 0, 0, 0.2);
  background-color: #37405a;
}

img.logo {
  width: 50px;
  height: 50px;
  margin: 10px;
  float: left;
}

span.line {
  float: left;
  width: 1px;
  height: 50px;
  margin-top: 10px;
  border: solid 1px #2b2e40;
}

h1 {
  float: left;
  padding-top: 12px;
  padding-left: 20px;
  height: 24px;
  font-family: Roboto-Regular;
  font-size: 20px;
  font-weight: 400;
  color: #ffffff;
}
```

ボディのスタイル:

```

/* Body
----- */
body {
    display: flex;
    flex-direction: column;
    background-image: linear-gradient(to bottom, #30365a, #2b2e40);
}

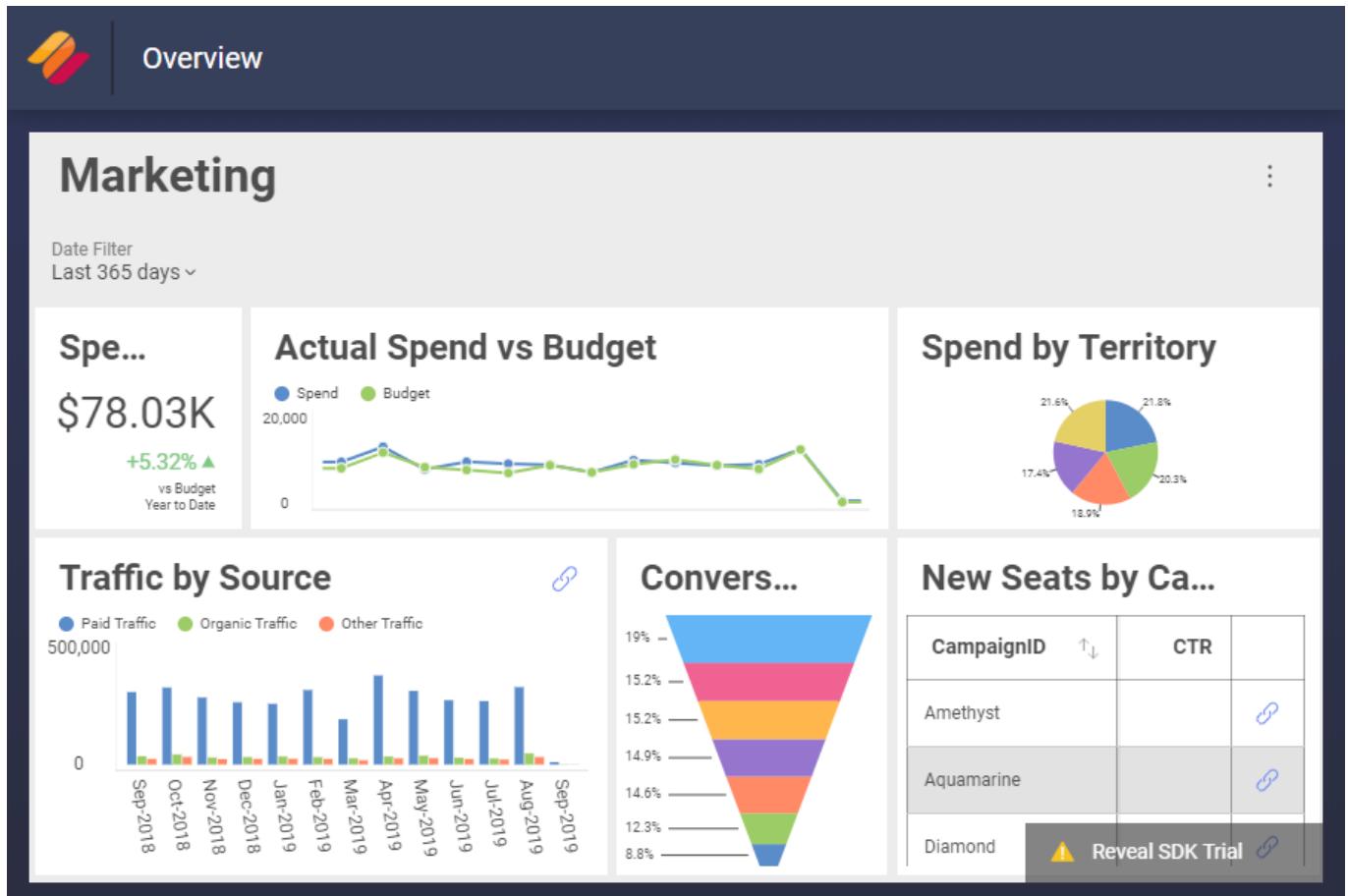
html, body {
    width: 100%;
    height: 100%;
}

body section {
    display: block;
    width: 100%;
    height: 100%;
    padding: 15px;
}

#revealView {
    height: 100%;
}

```

結果は以下のようになります。



# ダッシュボード ファイルの読み込み

## 概要

SDK でダッシュボードを開く/保存するには 2 つの方法があります

- **サーバー側:** はじめにクライアントページでダッシュボード ID を指定します。次に、サーバー上で以下に詳述するコールバック メソッドを使用して、指定された ID を持つダッシュボードのコンテンツと共にストリームを返します。

この方法が最も簡単な方法で、SDK をはじめて評価するときに推奨される方法です。

- **クライアント側:** 完全な制御と高い柔軟性が提供されます。カスタム サーバーからコンテンツを取得しながら、クライアントページでダッシュボードのコンテンツをストリームに提供します。

この方法を使用すると、たとえばユーザーのアクセス許可を確認したり、カスタムなユーザーインターフェイスを表示してダッシュボードを選択したり、ユーザーが使用する .rdash ファイルをアップロードしたりすることができます。クライアント側での方法の詳細については、[セットアップと構成 \(クライアント\)](#)を参照してください。

## サーバー側の作業

ダッシュボードを視覚化するために、SDK に Dashboard クラスのインスタンスを提供できます。これにより、ストリームを rdash または rdash の json 文字列表現にインスタンス化できます。

以下のコードスニペットは、プロジェクトに組み込みリソースとして追加された Rdash ファイルを読み込む方法を示しています。このメソッドは、**RevealSdkContextBase.GetDashboardAsync** のための実装です。

## コード

```
public override Task<Dashboard> GetDashboardAsync(string dashboardId)
{
    var dashboardFileName = dashboardId + ".rdash";
    var resourceName = $"Demo1.Dashboards.{dashboardFileName}";
    var assembly = Assembly.GetExecutingAssembly();
    var rdashStream = assembly.GetManifestResourceStream(resourceName);
    var dashboard = new Dashboard(rdashStream);

    return Task.FromResult(dashboard);
}
```

**RevealSdkContextBase.GetDashboardAsync** のこのコードは、クライアントで **RVDashboard.loadDashboard** 関数を使用するとサーバー上で呼び出されます。そして最初のパラメーターとしてクライアント側で指定された dashboardId を取得します。

# データ ソースの置き換え (MS SQL サーバー)

## 概要

ダッシュボードのデータを (Reveal Server SDK) ロードして処理する前に、ダッシュボードの各視覚化に使用される構成またはデータをオーバーライドできます。

以下は **RevealSdkContextBase** で実装するプロパティの 1 つです。

```
IRVDataSourceProvider DataSourceProvider { get; }
```

インターフェイス **IRVDataSourceProvider** を実装するクラスは、特定の可視化またはダッシュボード フィルターによって使用されるデータ ソースを置換または変更することができます。

## 使用事例

以下は、一般的なユース ケースです。

- 使用するデータベースの名前は、現在のユーザーやアプリが取得する userId、division、company、customer などその他の属性に応じて変更できます。これにより、マルチテナントデータベースからデータを取得する単一のダッシュボードを持つことができます。
- 使用されているテーブルの名前、ロードするファイルのパスなどを変更することができます。ユース ケースは上記のものと似ています。
- データ ソースをメモリ内のデータ ソースに置き換えることができます。Reveal App はインメモリ データ ソースをサポートしていないため、CSV ファイルを使用してダッシュボードを設計し、このコード バックを使用して CSV データ ソースをインメモリデータ ソースに置き換えることができます。このシナリオでは、データは実際にメモリからロードされます (またはカスタムデータローダを使用して)。インメモリ データ ソースの使用方法の詳細については、[インメモリ データのサポート](#)をご覧ください。

## コード

以下のコード スニペットは、ダッシュボードの可視化のためにデータ ソースを置き換える方法の例です。**ChangeVisualizationDataSourceItemAsync** メソッドは、開かれているすべてのダッシュボードで、すべての可視化に対して呼び出されます。

```
public class SampleDataSourceProvider : IRVDataSourceProvider
{
    public Task<RVDataSourceItem>
    ChangeDashboardFilterDataSourceItemAsync(string userId, string dashboardId,
    RVDashboardFilter globalFilter, RVDataSourceItem dataSourceItem)
    {
        return Task.FromResult<RVDataSourceItem>(null);
    }
}
```

```

public Task<RVDataSourceItem> ChangeVisualizationDataSourceItemAsync(
    string userId, string dashboardId, RVVisualization visualization,
    RVDataSourceItem dataSourceItem)
{
    var sqlServerDsi = dataSourceItem as RVSqlServerDataSourceItem;
    if (sqlServerDsi != null)
    {
        // SQL サーバー ホストとデータベースの変更
        var sqlServerDS = (RVSqlServerDataSource)sqlServerDsi.DataSource;
        sqlServerDS.Host = "10.0.0.20";
        sqlServerDS.Database = "Adventure Works";

        // SQL サーバー テーブル/ビューの変更
        sqlServerDsi.Table = "Employees";
        return Task.FromResult((RVDataSourceItem)sqlServerDsi);
    }

    // データ ソース アイテムを新しいアイテムと置き換える
    if (visualization.Title == "Top Customers")
    {
        var sqlDs = new RVSqlServerDataSource();
        sqlDs.Host = "salesdb.local";
        sqlDs.Database = "Sales";

        var sqlDsi = new RVSqlServerDataSourceItem(sqlDs);
        sqlServerDsi.Table = "Customers";

        return Task.FromResult((RVDataSourceItem)sqlServerDsi);
    }

    return Task.FromResult(dataSourceItem);
}
}

```

上記の例では、次の 2 つの置換が実行されます。

- MS SQL Server データベースを使用するすべてのデータソースは、ハードコードされたサーバー 10.0.0.20、Adventure Works データベース、および Employees テーブルを使用するように変更されます。
- 注:** これは単純化されたシナリオで、同じテーブルからデータを取得するためにすべての可視化を置き換えた場合も、現実のシナリオとしては意味がありません。実際のアプリケーションでは、userId、dashboardId、データソース自体の値 (サーバー、データベースなど) などの追加情報を使用して新しい値を推測します。
- Top Customers というタイトルのすべてのウィジェットは、Customers テーブルを使用して salesdb.local サーバーの Sales データベースからデータを取得する新しい SQL Server データソースに設定されます。

**IRVDataSourceProvider** を実装することに加えて、それを返すために **RevealSdkContextBase.DataSourceProvider** の実装を変更する必要があります。

```
IRVDataSourceProvider DataServiceProvider => new SampleDataSourceProvider();
```

# Excel および CSV ファイルのデータ ソースの置き換え

## 概要

Reveal アプリでダッシュボードを作成するとき、クラウドに保存されている Excel または CSV ファイルを使用してデータを入力することがよくあります。

ダッシュボードをエクスポートしてカスタム アプリケーションに組み込んだ後、これらのファイルをローカルディレクトリに移動し、**Reveal SDK** を使用してアクセスし、ローカルデータ ソースとして設定できます。

## 手順

ローカルの Excel ファイルと CSV ファイルを使用してエクスポートされた ダッシュボードにデータを入力するには、次の手順に従う必要があります。

1. [SDK 用のダッシュボードの取得](#)の説明に従って、ダッシュボード ファイルをエクスポートします。
2. [初めてのアプリ作成](#)の説明に従って、アプリケーションにダッシュボードを読み込みます。
3. ダッシュボードの作成に使用したファイルをクラウドストレージからダウンロードし、ローカル フォルダーにコピーします。
4. ローカル フォルダ名を *LocalStoragePath* プロパティの値として設定します。これについての詳細には、[セットアップと構成 \(サーバー\) - サーバー SDK を初期化](#)をご覧ください。
5. プロジェクトに新しい *CloudToLocalDatasourceProvider* クラスを追加します。
6. 以下のコード セクションの関連するスニペットから実装コードをコピーします。
7. *RevealSdkContext* クラスの *DataSourceProvider* プロパティを *CloudToLocalDatasourceProvider* に設定します:

```
public override IRVDataSourceProvider DataSourceProvider => new CloudToLocalDatasourceProvider();
```

## コード

```
public class CloudToLocalDatasourceProvider : IRVDataSourceProvider
{
    public Task<RVDataSourceItem>
    ChangeDashboardFilterDataSourceItemAsync(string userId, string dashboardId,
                                              RVDashboardFilter filter, RVDataSourceItem dataSourceItem)
    {
        return ProcessDataSourceItem(dataSourceItem);
    }

    public Task<RVDataSourceItem>
    ChangeVisualizationDataSourceItemAsync(string userId, string dashboardId,
                                           RVVisualization visualization, RVDataSourceItem
    dataSourceItem)
    {
```

```

        return ProcessDataSourceItem(dataSourceItem);
    }

    protected Task<RVDataSourceItem> ProcessDataSourceItem(RVDataSourceItem
dataSourceItem)
{
    // Return data source unless it is an excel or csv file.
    if (dataSourceItem is RVExcelDataSourceItem == false &&
        dataSourceItem is RVCsvDataSourceItem == false)
    {
        return Task.FromResult(dataSourceItem);
    }

    var resourceBased = dataSourceItem as RVResourceBasedDataSourceItem;
    var resourceItem = resourceBased?.ResourceItem as RVDataSourceItem;
    var title = resourceItem?.Title;

    if (string.IsNullOrEmpty(title))
    {
        return Task.FromResult(dataSourceItem);
    }

    var localItem = new RVLocalFileDataSourceItem();

    // The SDK uses the local folder set as LocalStoragePath.
    localItem.Uri = @"local:/" + title;

    // The title assigned here is the original data source name.
    localItem.Title = title;
    resourceBased.ResourceItem = localItem;

    return Task.FromResult(dataSourceItem);
}
}

```

[!NOTE] *CloudToLocalDatasourceProvider* は、Excel ファイルと CSV ファイルのみを自動的に置き換えます。他のファイルタイプまたはデータ ソースは変更されません。置換ファイルは、ダッシュボードの作成に使用したものと同じであるか、**同じスキーマ**を共有しているがデータが異なる新しいファイルである必要があります。

# インメモリ データのサポート (WEB)

## 概要

アプリケーション状態の一環として、ユーザーが要求したレポートの結果や Reveal でまだサポートされていないデータソースからの情報 (カスタムデータベースや特定のファイル形式など) にすでにメモリ内にあるデータを使用します。

インメモリは特別なタイプのデータソースで、SDK でのみ使用でき、Reveal アプリケーションでそのまま使用することはできません。そのため、[インメモリ データソース] を直接使用することはできません。以下のように、別の方に従ってください。

## インメモリ データソースの使用

推奨される方法は、**インメモリ データと一致するスキーマを持つデータファイルを定義する** 方法です。データファイルは、たとえば CSV または Excel ファイルにすることができ、スキーマは基本的にフィールドのリストと各フィールドのデータ型です。

以下の例では、特定のスキーマを使用してデータファイルを作成し、データベースから情報を取得する代わりにメモリ内のデータを使用する方法について詳しく説明します。

## コード例

以下の例では、人事システムに人事メトリクスを表示するダッシュボードを埋め込むために、社内の Employee のリストでインメモリデータを使用します。データベースから Employee のリストを取得するのではなく、メモリ内のデータを使用します。

これらすべてを実現するには、ダミー データを使用して、Reveal アプリでダッシュボードを作成してエクスポートする必要があります。

### Reveal アプリについて

Reveal アプリは、ダッシュボードを作成し、表示させ、およびチームと共有できるビジネス インテリジェンス ツールです。Reveal アプリの詳細については、[オンラインデモ](#)にアクセスするか、[ヘルプ ドキュメント](#) を参照してください。

### データファイルとサンプル ダッシュボードの準備

簡略化した Employee には以下のプロパティがあります。

- *EmployeeID*: string
- *Fullname*: string
- *Wage*: numeric

### 手順:

1. 同じスキーマで CSV ファイルを作成

```
EmployeeID,Fullname,Wage  
23,John Smith,345.67  
45,Emma Thompson,432.23
```

2. Dropbox や Google Drive など、しファイル共有システムにファイルをアップロード
3. ダミーデータを使用してダッシュボードを作成 (実際の生産データは後でアプリケーションで提供する予定です)
4. ダッシュボードをエクスポート (ダッシュボード → エクスポート → ダッシュボード)

ダッシュボードの可視化と実際のデータの返却

ダミーのデータではなくカスタムのデータを使ってダッシュボードを可視化する必要があります。

1. [データソースの置き換え](#)で説明したように、**IRVDataSourceProvider** を実装し、**RevealSdkContextBase** の **DataSourceProvider** プロパティとして返します。

次に、メソッド **ChangeVisualizationDataSourceItemAsync** の実装では、次のようなコードを追加する必要があります。

```
public Task<RVDataSourceItem> ChangeVisualizationDataSourceItemAsync(string  
userId, string dashboardId, RVVisualization visualization, RVDataSourceItem  
dataSourceItem)  
{  
    var csvDsi = dataSourceItem as RVCsvDataSourceItem;  
    if (csvDsi != null)  
    {  
        var inMemDsi = new RVInMemoryDataSourceItem("employees");  
        return Task.FromResult((RVDataSourceItem)inMemDsi);  
    }  
    return Task.FromResult((RVDataSourceItem)null);  
}
```

このようにして、ダッシュボード内の CSV ファイルへのすべての参照を、基本的に employees で識別されるインメモリデータソースに置き換えます。この ID は後でデータを返すときに使用されます。

2. 以下のように **IRVDataProvider** を実装するために、実際のデータを返すメソッドを実装します。

```
public class EmbedDataProvider : IRVDataProvider  
{  
    public Task<IRVInMemoryData> GetData(string userId,  
    RVInMemoryDataSourceItem dataSourceItem)  
    {  
        var datasetId = dataSourceItem.DatasetId;  
        if (datasetId == "employees")  
        {  
            var data = new List<Employee>()
```

```

        {
            new Employee(){ EmployeeID = "1", Fullname="John Doe",
Wage = 80325.61 },
            new Employee(){ EmployeeID = "2", Fullname="Doe John",
Wage = 10325.61 },
        };
        return Task.FromResult<IRVInMemoryData>(new
RVInMemoryData<Employee>(data));
    }
    else
    {
        throw new Exception("Invalid data requested");
    }
}
}

```

Employee クラスのプロパティは CSV ファイルの列とまったく同じ名前であり、データ型も同じです。フィールド名、フィールド ラベル、またはプロパティのデータ型を変更したい場合は、クラス宣言で属性を使用できます。

- RVSchemaColumn 属性を使用してフィールド名やデータ型を変更できます。
- DisplayName 属性を使用してフィールドラベルを変更できます。

```

public class Employee
{
    [RVSchemaColumn("EmployeeID", RVSchemaColumnType.Number)]
    public string EmployeeID { get; set; }

    [DisplayName("EmployeeFullscreen")]
    public string Fullname { get; set; }

    [RVSchemaColumn("MonthlyWage")]
    public double Wage { get; set; }
}

```

更に **IRVDataProvider** の実装には、それを返すために **RevealSdkContextBase.DataProvider** の実装を変更する必要があります。

```
IRVDataProvider DataProvider => new EmbedDataProvider();
```

# データ ソースへ資格情報を提供

## 概要

Server SDK では、データ ソースにアクセスするときに使用される一連の資格情報を渡すことができます。

## コード

最初の手順は、以下に示すように、**IRVAuthenticationProvider** を実装し、それを **RevealSdkContextBase** の **AuthenticationProvider** プロパティとして返します。

```
public class EmbedAuthenticationProvider : IRVAuthenticationProvider
{
    public Task<IRVDataSourceCredential> ResolveCredentialsAsync(string userId,
RVDashboardDataSource dataSource)
    {
        IrvDataSourceCredential userCredential = null;
        if (dataSource is RVPostgresDataSource)
        {
            userCredential = new
RVUsernamePasswordDataSourceCredential("postgresuser", "password");
        }
        else if (dataSource is RVSqlServerDataSource)
            // 「domain」パラメーターは必ずしも必要ではなく、これは SQL Server の構成によって異なりま
す。
        {
            userCredential = new
RVUsernamePasswordDataSourceCredential("sqlserveruser", "password", "domain");
        }
        else if (dataSource is RVGoogleDriveDataSource)
        {
            userCredential = new
RVBearerTokenDataSourceCredential("fhJhbUci0mJSUzi1nIiSint....",
"user@company.com");
        }
        else if (dataSource is RVRESTDataSource)
        {
            userCredential = new RVUsernamePasswordDataSourceCredential(); // 
Anonymous
        }
        return Task.FromResult<IRVDataSourceCredential>(userCredential);
    }
}
```

## 実装するクラスの選択

使用できるクラスは 2 つあり、どちらも **IRVDataSourceCredential** インターフェイスを実装します。以下に詳述するように、データ ソースに応じてクラスを選択する必要があります。

- クラス **RVBearerTokenDataSourceCredential** は以下で動作します。
  - アナリティクス ツール (Google アナリティクス)。
  - コンテンツ マネージャーとクラウド サービス (Box、Dropbox、Google Drive、OneDrive、SharePoint Online)。
- クラス **RVUsernamePasswordDataSourceCredential** は以下と動作します。
  - カスタマー リレーションシップ マネージャ- (Microsoft Dynamics CRM オンプレミスおよびオンライン)。
  - データベース (Microsoft SQL Server、Microsoft Analysis Services サーバー、MySQL、PostgreSQL、Oracle、Sybase)。
- **両クラス**は以下と動作します。
  - その他のデータ ソース (OData フィード、Web Resources、REST API)。

## 認証なし

認証なしで匿名のリソースで作業することができます。この場合、空のコンストラクタを持つ **RVUsernamePasswordDataSourceCredential** を使用できます。これは、そのクラスで機能するすべてのデータ ソースに対して実行できます。

上記のサンプルで使用したコード スニペット:

```
else if (dataSource is RVRESTDataSource)
{
    userCredential = new RVUsernamePasswordDataSourceCredential();
}
```

# 新しい可視化とダッシュボードの作成 (Web)

## 概要

[ダッシュボードの編集と保存](#)で説明したように、ダッシュボードの変更を保存する方法を処理する方法は 2 つ(クライアントサイドとサーバーサイド)あります。これらのシナリオは、ユーザーが既存のダッシュボードに以下のような変更を加える際には正しく機能します。

- フィルターの追加/編集
- 視覚化のタイプを変更する(チャート、ゲージ、グリッドなど)
- テーマの変更

ただし、新しい視覚化を追加するには、ユーザーは使用するデータソースを選択する必要があります。そのためには、含まれているアプリケーションが SDK に情報を提供する必要があるので、新しい可視化に使用できるデータソースのリストを表示できます。

## データソースのリストを表示

データソースのリストを表示するために使用する必要があるコールバックは、**onDataSourcesRequested** です。このコールバックに独自の関数を設定しない場合、新しい表示形式が作成されると、Reveal はダッシュボードで使用されているすべてのデータソースを表示します(存在する場合)。

コード:

以下のコードは、インメモリ項目と SQL Server データソースを表示するようにデータソース選択画面を構成する方法を示しています。

```
window.revealView.onDataSourcesRequested = function (callback) {
    var inMemoryDSI = new $.ig.RVInMemoryDataSourceItem("employees");
    inMemoryDSI.title = "Employees";
    inMemoryDSI.description = "Employees";

    var sqlDs = new $.ig.RVSqlServerDataSource();
    sqlDs.title = "Clients";
    sqlDs.id = "SqlDataSource1";
    sqlDs.host = "db.mycompany.local";
    sqlDs.port = 1433;
    sqlDs.database ="Invoices";

    callback(new $.ig.RevealDataSources([sqlDs], [inMemoryDSI], false));
};
```

3 番目のパラメータの `false` 値は、ダッシュボード上の既存のデータソースが表示されないようにします。そのため、[+] ボタンを使用して新しいウィジェットを作成すると、以下の画面が表示されます。

データ項目

Employees

データストア

Clients  
Invoices @ db.mycompany.local

RVInMemoryDataSourceItem コンストラクタに渡される employees パラメーターは、[インメモリデータのサポート](#)で使用されているコンストラクタと同じデータセット ID で、サーバー側で返されるデータセットを識別します。

## 新しいダッシュボードの作成

以下の手順でダッシュボードを簡単に作成できます。

- ダッシュボードプロパティを `$.ig.RevealView` に設定せず、更に `$.ig.RVDashboard.loadDashboard` を使用せずに `$.ig.RevealView` オブジェクトを初期化します。
- ダッシュボードを編集モードで起動するには、`isEditing` を `true` に設定します。
- ダッシュボード プロパティを、新しく作成された `$.ig.RVDashboard` のインスタンスに設定します。

```
var revealView = new $.ig.RevealView("#revealView");
revealView.startEditMode = true;
revealView.dashboard = new $.ig.RVDashboard();
```

SDK とともに配布されている UpMedia ウェブ アプリケーションに、`CreateDashboard.cshtml` の実用的な例が含まれます。

# ダッシュボードの編集と保存

## ダッシュボードの編集

**RevealView** の **Dashboard** プロパティ (タイプ `$.ig.RVDashboard`) は、エンドユーザーがダッシュボードの編集を開始すると更新されます。たとえば、可視化またはフィルターを追加または削除すると、`$.ig.RVDashboard` のコレクションが自動的に更新されます。

さらに、`$.ig.RVDashboard` クラスには、ダッシュボードに未保存の変更があるかどうかを確認するのに非常に役立つ **onHasChangesChanged** プロパティが含まれています。

コードサンプル:

```
dashboard.onHasChangesChanged = function (hasChanges) {
    console.log("Has Changes: " + hasChanges);
};
```

ユーザーが可視化の編集を終了した後、可視化工ディターを閉じると、`$.ig.RevealView` の **VisualizationEditorClosed** イベントが発生します:

```
revealView.onVisualizationEditorClosed = function (args) {
    if (args.isCancelled) {
        console.log("Visualization editor cancelled " + (args.isNewVisualization
? "creating a new visualization" : "editing " + args.visualization.title));
        return;
    }
    if (args.isNewVisualization) {
        console.log("New Visualization created: " + args.visualization.title);
    } else {
        console.log("Visualization modified: " + args.visualization.title);
    }
};
```

新しい可視化を追加する方法を制御する必要がある場合は、[新しい可視化とダッシュボードの作成](#)を参照してください。

## ダッシュボードの保存

[ダッシュボードファイルの読み込み](#)でダッシュボードに変更を保存する方法が 2 つ紹介されています。

- **クライアントサイド:** このメソッドを使用するには **revealView** オブジェクトの **onSave** 属性に関数を設定する必要があります。この方法は、操作 (保存と名前を付けて保存) がどのように実行されるかについて、包含アプリケーションの柔軟性が高まるため、推奨される方法です。

コードサンプル:

```
revealView.onSave = function(rv, saveEvent) {
    saveEvent.serialize(function(blobValue) {
        //TODO: XMLHttpRequest オブジェクトを使用してサーバーに POST するなど、
        //BLOB 値を保存します。
    });
};
```

保存操作を処理したくない場合は、次の設定でダッシュボードを編集するオプションをオフにすることができます。

```
revealView.canSaveAs = false;
```

この方法は、ユーザーが変更を加えることを想定していない場合などに便利です。

- **サーバー側:** `onSave` イベントが `$.ig.RevealView` オブジェクトに設定されていない場合、デフォルトのサーバー側の保存方法が使用されます。エンドユーザーが変更されたダッシュボードを保存した後、HTTP POST リクエストが呼び出されます。その結果、現在定義されている SDK コンテキストの `SaveDashboardAsync` メソッドが呼び出されます。次に `_dashboardId` を文字列として取得し、`dashboardStream` にダッシュボードの Stream 表現を取得します。

サーバー側のアプローチでは、コードをクライアントサイドに実装するだけで済みますが、クライアントサイドの柔軟性は失われます。これは、たとえば、ダッシュボードを保存する最終的な場所をユーザーが選択できないことを意味します。SDK コンテキストの詳細については、[サーバーコンテキストの定義](#)を参照してください。

# ダッシュボードまたは画像として可視化データをエクスポート (WEB)

---

## 概要

ダッシュボードまたは特定の可視化の画像を生成してそれをエクスポートする場合、以下の オプションがあります。

- **画像**として;
- **PDF** として;
- **PowerPoint** プrezentationとして;
- **Excel** データ形式として;

ダッシュボードまたは視覚化を有効にするには、次のことをします。

- `$.ig.RevealView` でエクスポート設定を有効にする、あるいは
- `$.ig.RevealView` 以外でコードによるエクスポートを開始

## 前提条件

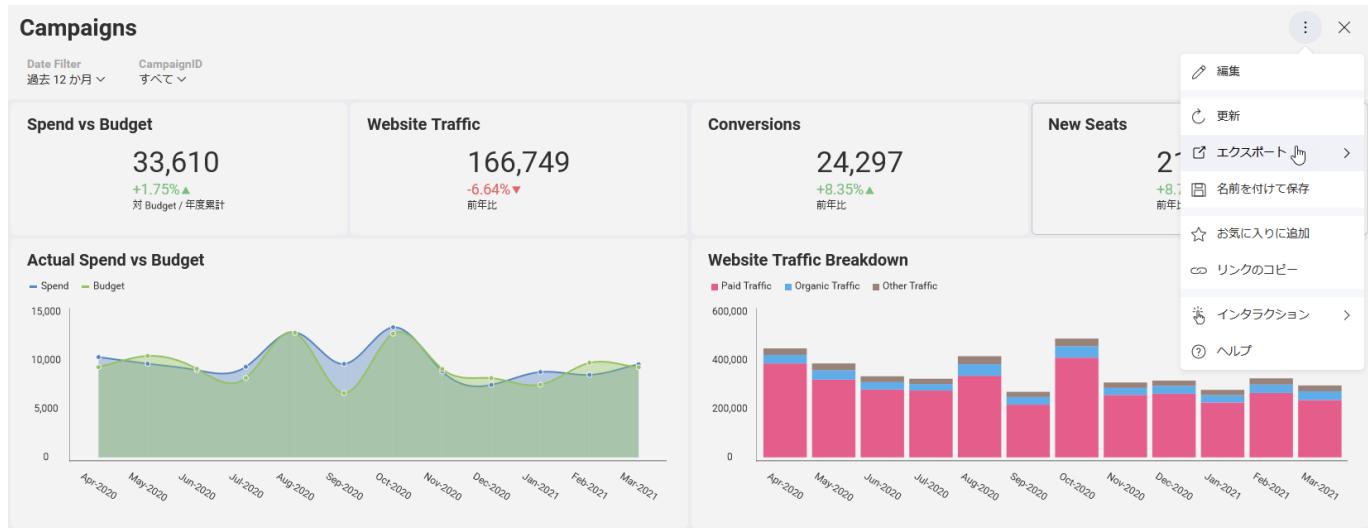
サーバー側で**画像エクスポート**機能を有効にする必要があります。これを行うには、[サーバー側のスクリーンショット生成を有効にする](#)を参照してください。

## エクスポートの設定の使用

ユーザー側でダッシュボードから特定の可視化画像やドキュメントなどを生成可能にするには、関連するプロパティを `true` に設定するだけでできるようになります。

- `revealView.showExportImage` - **画像**としてエクスポートする場合;
- `revealView.showExportToPDF` - **PDF** としてエクスポートする場合;
- `revealView.showExportToPowerpoint` - **PowerPoint** プrezentationとしてエクスポートする場合;
- `revealView.showExportToExcel` - **Excel** データ形式としてエクスポートする場合;

これにより、ダッシュボードが開かれたとき、または特定の可視化が最大化されたときに、オーバーフローメニューで**エクスポート** ボタンが使用可能になります。



ユーザーが [エクスポート] ボタンをクリックすると、有効なエクスポート オプションの 1 つを選択できます。

### 画像エクスポート オプションを使用する場合の詳細

ユーザーがエクスポート オプションから [画像のエクスポート] を選択すると、[画像のエクスポート] ダイアログが開きます。ここでは、ユーザーは [クリップボードへコピー] と [画像としてエクスポート] の 2 つのオプションから選択することができます。右下の [画像のエクスポート] ボタンをクリックすると、画像がンドユーザーに送信されます。

含まれているアプリがエクスポートされた画像を別の方法で処理する必要がある場合、出力画像にアクセスできる **onImageExported** コールバックを提供できます。以下は **onImageExported** コールバックのサンプル実装です。

```
revealView.onImageExported = function (img) {
  var body = window.open("about:blank").document.body;
  body.appendChild(img);
}
```

### コードによって開始されたエクスポート

コードで RevealView の画像を取得するには、**toImage** メソッドを呼び出す必要があります。このメソッドを呼び出しても [画像としてエクスポート] ダイアログは表示されません。これにより、ユーザーが RevealView の外側にあるボタンをクリックしたときにスクリーンショットを取得できます。このメソッドは、RevealView コンポーネントが画面に表示されていると同じスクリーンショットを作成します。

```
var image = revealView.toImage();
```

**toImage** メソッドの呼び出し時にユーザーがダイアログを開いている場合、ダッシュボードと一緒にそのダイアログのスクリーンショットが取得されます。

これは、Reveal アプリ側では Reveal ビューのエクスポート ボタンを非表示にしておき、Reveal ビューの外部で発生する別の操作から画像へのエクスポートを起動するシナリオで役立ちます。

# ダッシュボード リンク

---

## 概要

Reveal アプリケーションはダッシュボードのリンクをサポートしているため、ユーザーはダッシュボードをナビゲートできます。ダッシュボードからダッシュボードに移動することで、業務上のハイレベルな概要からより詳細なビューに進むことができます。

## 一般的なユースケース

たとえば、各部門 (HR、Sales、Marketing) の主要業績評価指標を表示する Company 360 ダッシュボードを作成できます。ユーザーが表示形式の 1 つを最大化すると、ナビゲーションがトリガーされ、ユーザーはその部門に関するより詳細な情報を含む別のダッシュボードに移動します。

あるいは、ダッシュボード リンクを使用して、より具体的なダッシュボードに移動することもできます。たとえば、上位 25 の顧客を表示する可視化を含むダッシュボードで、顧客の1人を選択すると、ユーザーは新しいダッシュボードに移動します。この新しいダッシュボードには、最近の購入、連絡先情報、売れ筋商品など、選択した顧客に関する詳細情報が表示されます。

ダッシュボードへリンクする機能の詳細については、Reveal のユーザーガイドの[ダッシュボードのリンク](#)を参照してください。

## コード例

SDK とのダッシュボードリンクを使用できますが、ナビゲーションがトリガーされているときには包含アプリケーションが関与する必要があります。SDK はダッシュボードが格納されている場所を処理しないため、ターゲット ダッシュボード用のダッシュボードファイルを提供するには、それを含むアプリケーションが必要です。

実用的な例として、SDK とともに配布されている *UpMedia* サンプルの Marketing.cshtml ページを使用できます。

基本的にこの 2 項目を完了すると後は SDK によって処理されます。

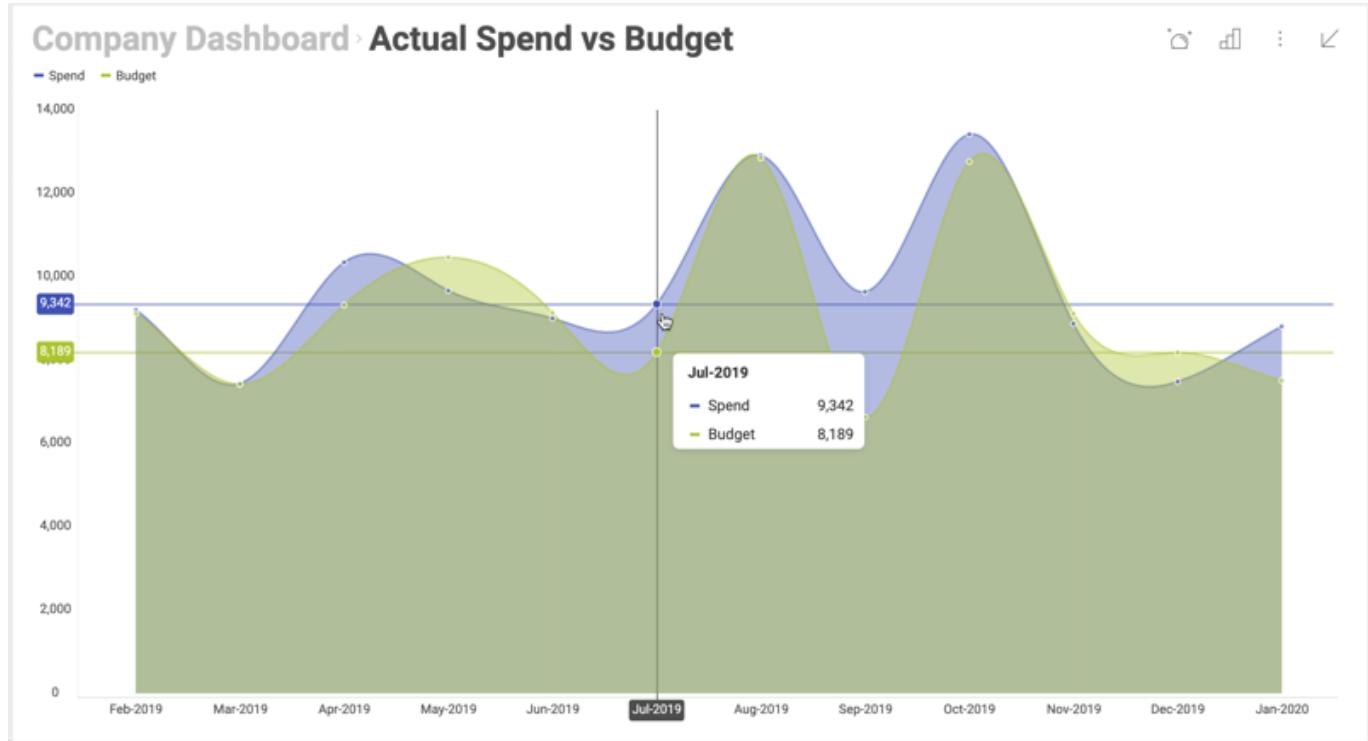
1. **onVisualizationLinkingDashboard** イベントを処理します。
2. ターゲット ダッシュボードの ID を使用してコールバックを呼び出します

```
revealView.onVisualizationLinkingDashboard = function (title, url, callback) {  
    // リンクのターゲットのダッシュボード ID を提供します。  
    callback("Campaigns");  
};
```

# 可視化の最大化と単一可視化モード

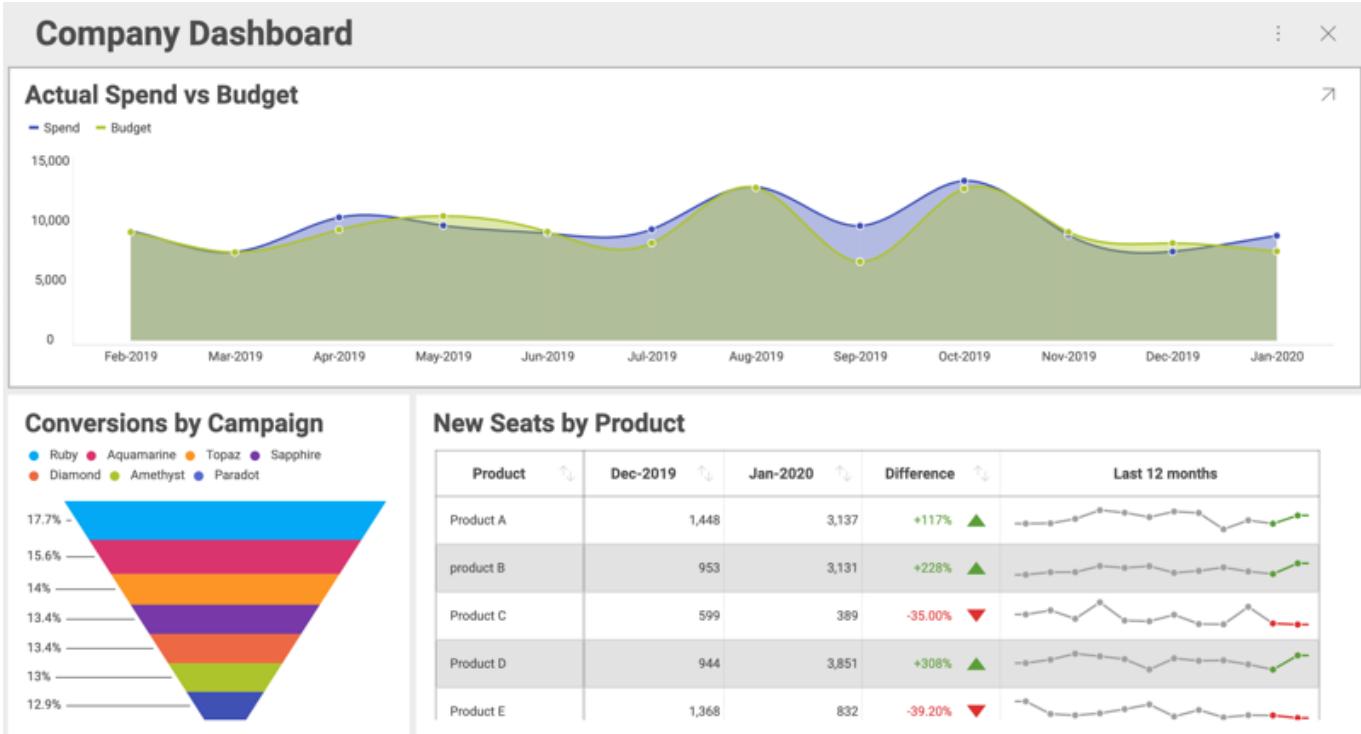
## 概要

Web Client SDK は、ダッシュボードをユーザーに表示する際に最大化した可視化表示を 1 つだけ表示し、更に最初の可視化表示をロックしてユーザーがダッシュボード全体にアクセスできないようにすることができます。



## 例の詳細

可視化した 3 つのダッシュボードがあり、それぞれの可視化に会社の異なる部門のデータが表示されているとします。



この例では、業務アプリケーションでこれらの可視化を使用します。各部署のホームページに表示される情報の一部として含めます。

## 可視化の最大化

最大化された可視化でダッシュボードを開くには、最初に **revealView** のダッシュボード プロパティを設定する必要があります。次に、最大化する可視化を **\$.ig.RevealView** インスタンスに渡し、**maximizedVisualization** プロパティを設定します。この属性に視覚化を設定しない場合、ダッシュボード全体が表示されます。

**\$.ig.RevealView** オブジェクトの設定に示すように、ページに特定のダッシュボードを表示できます。今回は、**maximizedVisualization** プロパティを設定する必要があります。以下のコードスニペットに示すように、ID が AllDivisions のダッシュボードから可視化した Sales が表示されています。

```

<script type="text/javascript">
...
var dashboardId = 'AllDivisions';

$.ig.RVDashboard.loadDashboard(dashboardId, function (dashboard) {
    var revealView = new $.ig.RevealView("#revealView");
    revealView.dashboard = dashboard;
    revealView.maximizedVisualization =
        dashboard.visualizations.getByTitle('Sales');

}, function (error) {
    console.log(error);
});
</script>

```

```
<div id="revealView" style="height:500px;" />
```

最初に最大化した可視化表示は Sales というタイトルの可視化になりますが、それでもエンドユーザーはダッシュボードに戻って残りの可視化を表示できます。

## 単一可視化モード

また、最初の可視化をロックして、常に可視化を 1 つのみ表示するようにすることもできます。これにより、ダッシュボードは単一の視覚化ダッシュボードのように機能します。これが [単一可視化モード] の概念です。

[単一可視化モード] をオンにするには、以下のように **singleVisualizationMode** を true に設定します。

```
revealView.singleVisualizationMode = true;
```

この 1 行を追加すると、ダッシュボードは単一の視覚化ダッシュボードとして機能します。各部門のホームページでも同じことができます。`dashboard.visualizations.getTitle()` の表示形式のタイトルを正しいものに置き換えてください。

## ロックされた可視化を動的に変更

ページを再ロードせずに、表示されている単一の表示形式を動的に変更することもできます。ユーザーの観点から見ると、アプリは部門のセレクターと最大化された視覚化を備えた単一ページのアプリケーションになります。ユーザーがリストから 1 つの部門を選択すると、最大化された視覚化が更新されます

以下では、`$.ig.RevealView` の **maximizeVisualization** メソッドを使用します。

```
<script type="text/javascript">
    var dashboardId = 'AllDivisions';

    $.ig.RVDashboard.loadDashboard(dashboardId, function (dashboard) {
        var revealView = window.revealView = new $.ig.RevealView("#revealView");
        revealView.singleVisualizationMode = true;
        revealView.dashboard = dashboard;
        revealView.maximizedVisualization =
            dashboard.visualizations.getTitle('Sales');

    }, function (error) {
        console.log(error);
    });
    function maximizeVisualization(title) {
        var dashboard = window.revealView.dashboard;
        window.revealView.maximizedVisualization =
            dashboard.visualizations.getTitle(title);
    }
</script>

<section style="display:grid;grid-template-rows:30px auto;">
    <section style="display:grid;grid-template-columns:auto auto auto;">
        <button onclick="maximizeVisualization('Sales')">Sales</button>
```

```
<button onclick="maximizeVisualization('HR')">HR</button>
<button onclick="maximizeVisualization('Marketing')">Marketing</button>
</section>
<div id="revealView" style="height:500px;" />
</section>
```

注意事項:

- **\$ig.RevealView** オブジェクトは window.revealView に設定し、後で **maximizeVisualization** プロパティが設定されたときに使用できます。
- div の前のセクションに追加されたボタンは、例として使用しています。最大化された可視化を切り替える手段として追加されました。ここでは、アプリケーションで同様のコードを使用する必要があります。
- この例では、サンプルダッシュボードの表示形式と一致するようにボタンがハードコードされていますが、ダッシュボードの表示形式のリストを繰り返すことでボタンのリストを動的に生成することができます。詳細については、**\$ig.RVDashboard.visualizations** をご覧ください。

# カスタム テーマの作成

---

## 概要

分析を既存のアプリケーションに埋め込む場合、それらのダッシュボードがアプリのルック アンド フィールと一致することが重要です。そのため、SDK を通じて Reveal ダッシュボードを完全に制御できます。

カスタム テーマで達成できる主なカスタマイズ:

- **カラー パレット**: 表示形式でシリーズを表示するために使用される色。色の数に制限はありません。すべての色が表示形式で使用されると、Reveal はこれらの色の新しい色合いを自動生成します。これにより、色が重複せず、各値に独自の色が設定されます。
- **アクセント色**: Reveal のデフォルトのアクセント色は、[+ ダッシュボード] ボタンやその他のインタラクティブなアクションで見つけることができる青の色合いです。アプリケーションで使用するのと同じアクセント色に一致するように色を変更できます。
- **条件付き書式の色**: 条件付き書式を使用するときに設定できる境界のデフォルトの色を変更します。
- **フォント**: Reveal は、アプリケーションで 3 種類のテキストを使用します: 標準、中、太字。これらの各テキストグループのフォントの使用を指定できます。
- **表示形式とダッシュボードの背景色**: ダッシュボードの背景色と表示形式の背景色を個別に構成できます。

## 一般的なユース ケース: 新しいカスタム テーマ

Reveal で独自のテーマを作成するのは、新しい **\$.ig.RevealTheme()** クラスのインスタンスを作成するのと同じように簡単です。このクラスには、概要にリストされているすべてのカスタマイズ可能な設定が含まれています。

新しい **\$.ig.RevealTheme** インスタンスを作成すると、各設定のデフォルト値が取得され、必要に応じてそれらを変更できます。

次に、テーマ インスタンスを **\$.ig.RevealSdkSettings** のクラスの静的テーマ プロパティに渡します。画面にダッシュボードまたは別の Reveal コンポーネントがすでに表示されている場合は、適用された変更を表示するために、再度描画する (ダッシュボード プロパティを再度設定する) 必要があります。

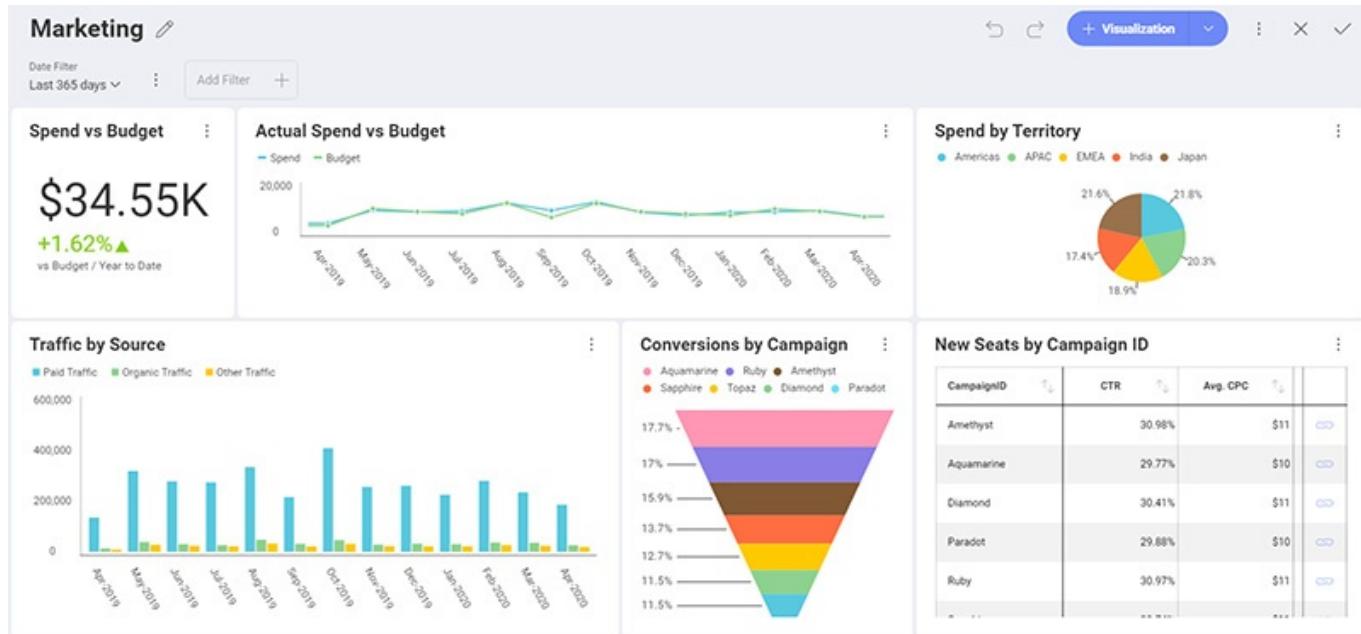
## 一般的なユース ケース: カスタム テーマの変更

すでに独自のテーマを適用しているが、他に加えた変更を失わずに一部の設定を変更したい場合があります。

この場合、**\$.ig.RevealSdkSettings** からテーマの静的プロパティを取得する必要があります。このメソッドを使用すると、RevealTheme 設定に最後に設定した値を取得できます。RevealTheme の新しいインスタンスを最初から作成する場合とは異なり、変更を適用してテーマを再度更新すると、デフォルト値ではなく、変更していない各設定の最新の値が取得されます。

## コード例

まず、以下は変更する前のサンプル ダッシュボードです。



次のコード スニペットでは、`revealTheme` クラスの新しいインスタンスを作成し、必要な設定に変更を適用し、Reveal Web でテーマを更新する方法を確認できます。

```
var revealTheme = new $.ig.RevealTheme();
revealTheme.chartColors = ["rgb(192, 80, 77)", "rgb(101, 197, 235)", "rgb(232, 77, 137)"];

revealTheme.mediumFont = "Gabriola";
revealTheme.boldFont = "Comic Sans MS";
revealTheme.fontColor = "rgb(31, 59, 84)";
revealTheme.accentColor = "rgb(192, 80, 77)";
revealTheme.dashboardBackgroundColor = "rgb(232, 235, 252)";

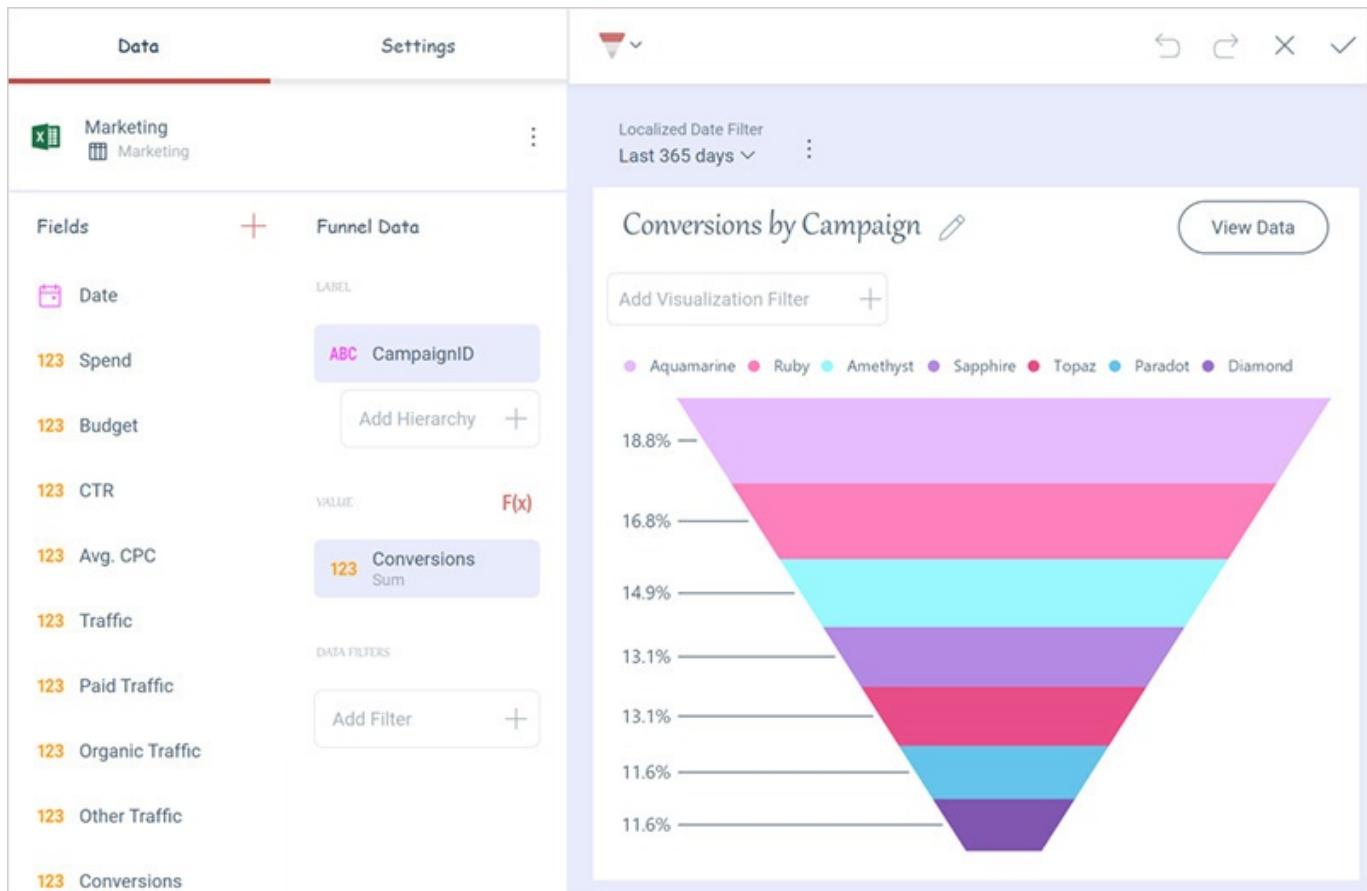
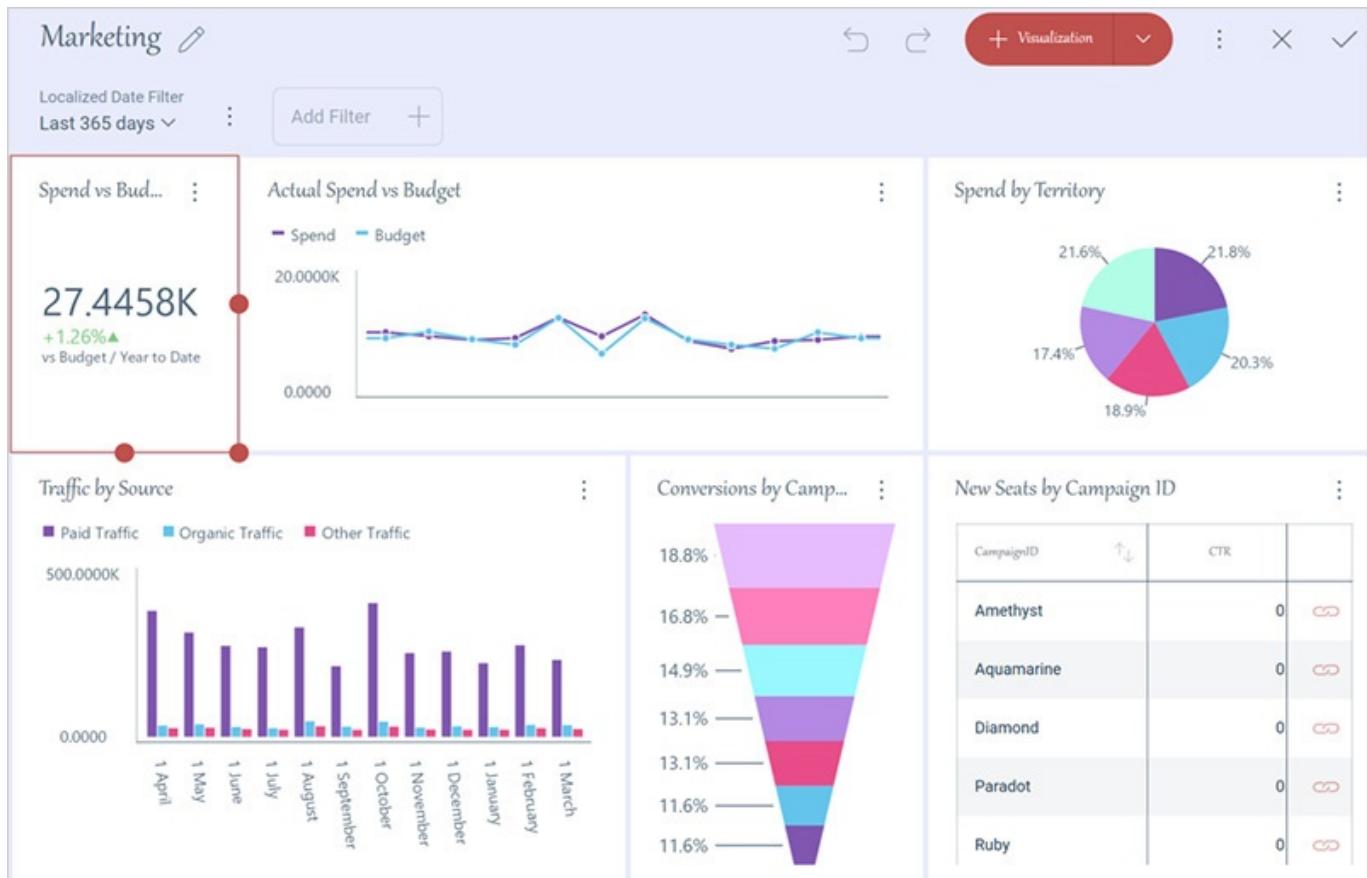
$.ig.RevealSdkSettings.theme = revealTheme;
```

[!NOTE] Reveal テーマの `boldFont`、`regularFont`、または `mediumFont` の設定を定義する場合、正確なフォントファミリ名を渡す必要があります。太さは、名前ではなくフォント自体の定義によって定義されます。`@font-face` (CSS プロパティ) を使用して、`$.ig.RevealTheme` フォント設定で指定されたフォントフェイス名が使用可能であることを確認する必要がある場合があります。

さらに、フォントをカスタマイズするには、ページの CSS に次の行を追加する必要があります。

```
<link href="https://fonts.googleapis.com/css?family=Righteous" rel="stylesheet">
<link href="https://fonts.googleapis.com/css?family=Domine" rel="stylesheet">
<link href="https://fonts.googleapis.com/css?family=Caveat" rel="stylesheet">
```

テーマの変更を実装すると、ダッシュボードと表示形式エディターの両方の結果を確認できます。



## カラー タイプの使用

RGB (red, green, blue) または HEX カラーのいずれかを使用して、カラー設定を指定できます。

```
revealTheme.dashboardBackgroundColor = "rgb(232, 235, 252)";  
revealTheme.dashboardBackgroundColor = "#E8EBFC";
```

## 定義済みのテーマ

Reveal SDK には、*Mountain Light*、*Mountain Dark*、*Ocean Light*、*Ocean Dark* の 4 つのビルド済みテーマが付属しています。アプリケーションのデザインに最適なものを設定することも、カスタムテーマのベースとして使用することもできます。

*UpdateCurrentTheme* メソッドを使用して、選択したビルド済みテーマの設定を適用します。

### Mountain Light テーマ

```
$.ig.RevealSdkSettings.theme = new $.ig.MountainLightTheme();
```

[!NOTE] Mountain Light には、カスタマイズ可能なすべての設定のデフォルト値が含まれています。つまり、Mountain Light と Reveal テーマは基本的に同じように見えます。

### Mountain Dark テーマ

```
$.ig.RevealSdkSettings.theme = new $.ig.MountainDarkTheme();
```

### Ocean Light テーマ

```
$.ig.RevealSdkSettings.theme = new $.ig.OceanLightTheme();
```

### Ocean Dark テーマ

```
$.ig.RevealSdkSettings.theme = new $.ig.OceanDarkTheme();
```

## 定義済みのテーマはどのように見えますか？

以下は、各ビルド済みテーマが適用されたときの表示形式エディターとダッシュボード エディターの外観を示すテーブルです。

テーマ	ダッシュボード エディター	表示形式エディター
-----	---------------	-----------

テーマ	ダッシュボードエディター	表示形式エディター
Mountain Light (デフォルト)	<p>Marketing</p> <p>This dashboard displays marketing performance across different dimensions. It includes a line chart for Actual Spend vs Budget, a pie chart for Spend by Territory, a bar chart for Traffic by Source, and a funnel chart for Conversions. A table titled 'New Seats by Campaign ID' provides detailed data for each campaign's CTR.</p>	<p>Conversions by Campaign</p> <p>A funnel chart visualizing the conversion rates for various campaigns. The total conversion rate is 17.4%.</p>
Mountain Dark	<p>Marketing</p> <p>The same marketing dashboard as above, but displayed in dark mode. The overall layout and data points remain identical.</p>	<p>Conversions by Campaign</p> <p>The same funnel chart as above, displayed in dark mode. The total conversion rate is 17.4%.</p>
Ocean Light	<p>Marketing</p> <p>The same marketing dashboard as above, displayed in light mode. The data and visual style are consistent with the Mountain Light version.</p>	<p>Conversions by Campaign</p> <p>The same funnel chart as above, displayed in light mode. The total conversion rate is 17.4%.</p>
Ocean Dark	<p>Marketing</p> <p>The same marketing dashboard as above, displayed in dark mode. The data and visual style are consistent with the Mountain Dark version.</p>	<p>Conversions by Campaign</p> <p>The same funnel chart as above, displayed in dark mode. The total conversion rate is 17.4%.</p>

# ユーザー クリック イベントの処理

---

## 概要

SDK は、ユーザーが可視化内のデータを含むセルをクリックしたときに処理できます。たとえば独自のナビゲーションを提供したり、アプリの既存の選択を変更したりする場合などに非常に便利です。

## コード

**onVisualizationDataPointClicked** イベントに登録することで、ユーザー クリック イベントを処理できます。

```
window.revealView.onVisualizationDataPointClicked = function (visualization, cell, row) {
    alert('Visualization clicked: ' + visualization.title() + ", cell: " +
cell.value);
};
```

コールバック関数では、クリックの場所に関する情報を受け取ります。

- クリックされた可視化の名前。
- クリックされたセルの値 (値、書式設定された値、列の名前を含む)。
- 同じセルの残りの値。

アプリケーションで選択したカスタマーを変更する場合は、カスタマー ID などの特定の属性を検索するためにはセル内の残りの値を使用できます。カスタマーの売上高のように、ユーザーが別のセルをクリックした場合も必要な情報を取得できます。

# ツールチップの作業

## 概要

エンドユーザーが表示形式でシリーズの上をホバーするか、シリーズをクリックするたびにトリガーされるイベントがあります(以下の画像を参照)。このイベントは `.onTooltipShowing` と呼ばれ、表示形式でツールチップを表示する方法をより柔軟に指定できます。



## 一般的なユースケース

Tooltip イベントをキャンセルするか、もしくはユーザーに表示される項目を変更できます。最も一般的な例:

- ツールチップをすべて無効にするか、特定の表示形式でのみ表示したい場合。
- ビューアーに役立つ RevealView コンポーネント以外のツールチップにデータを表示したい場合。

このイベントは、グリッドやゲージなどのツールチップをサポートしない表示形式ではトリガーされないことに注意してください。

## コード例

以下のコード スニペットでは、表示形式のツールチップを無効にし、エンドユーザーがこの表示形式をホバーまたはクリックするときにイベント引数から追加情報を取得する方法を示します。

```
revealView.onTooltipShowing = function (args) {
```

```
if (args.visualization.title == "NoNeedForToolips") {
    args.Cancel = true;
}
console.log("onTooltipShowing: visualization: " + args.visualization.title() +
",cell: " + args.cell.value + ", row:" + args.row.length);
};
```

イベント引数には、ホバーまたはクリックされている表示形式、ホバーまたはクリックされたデータの正確なセル、このセルの行全体 (他の列の情報が必要な場合)、および Cancel ブール値に関する情報が含まれています。

# ユーザーインターフェイス要素の表示/非表示

**\$.ig.RevealView** コンポーネントを使用して、エンドユーザーに対するさまざまな機能または UI 要素を有効または無効にすることができます。使用可能なプロパティの多くはブール型で簡単に使用できますが、他のプロパティはそれほど多くありません。

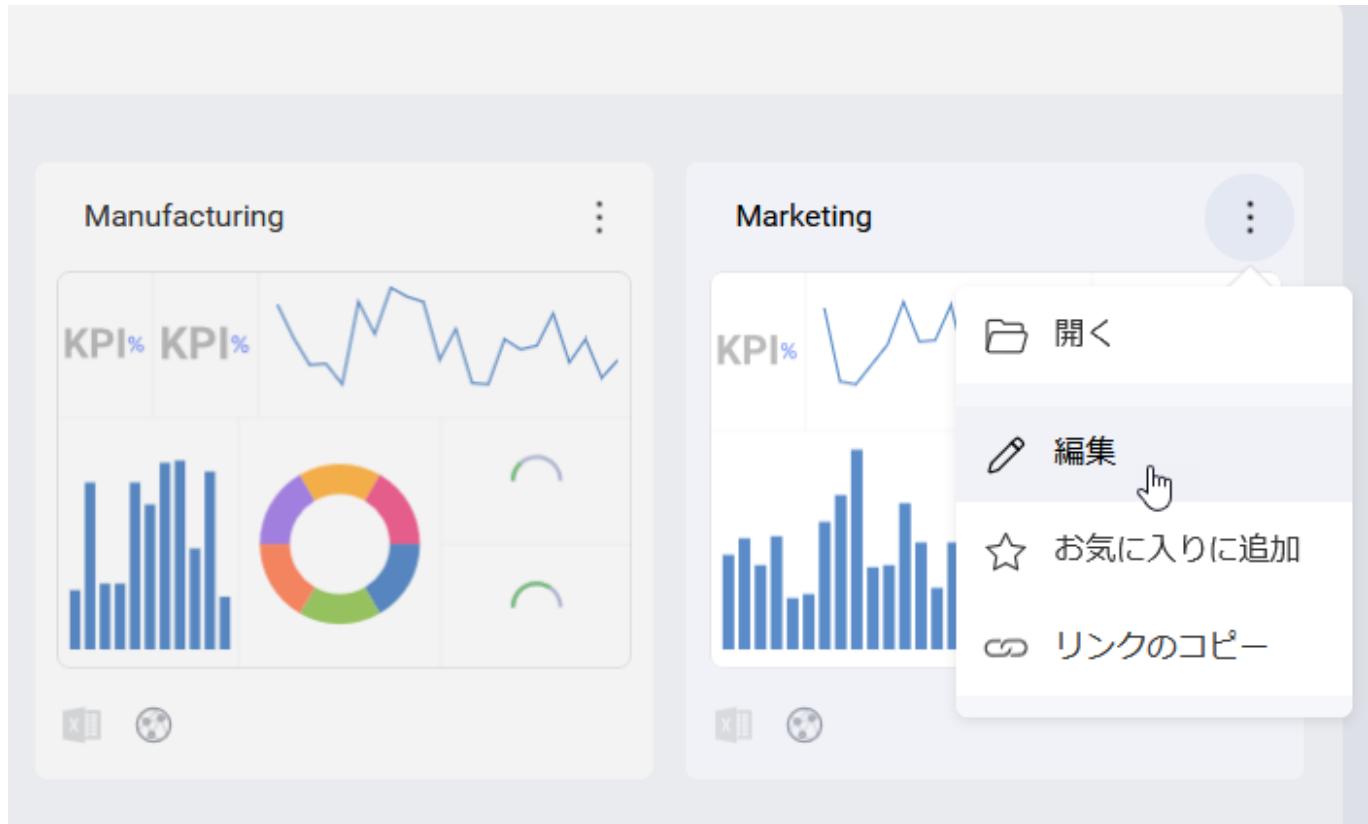
以下に作成する `revealView` インスタンスおよび DOM 要素は、このトピックのすべてのコード スニペットで使用されます。

```
var revealView = new $.ig.RevealView("#revealView");
...
<div id="revealView" style="height:500px;" />
```

[!NOTE] CSS レイアウト アプローチによっては、RevealView をホストする要素を静的ではない位置属性 (相対または絶対など) を設定して配置する必要がある場合があります。

## canEdit

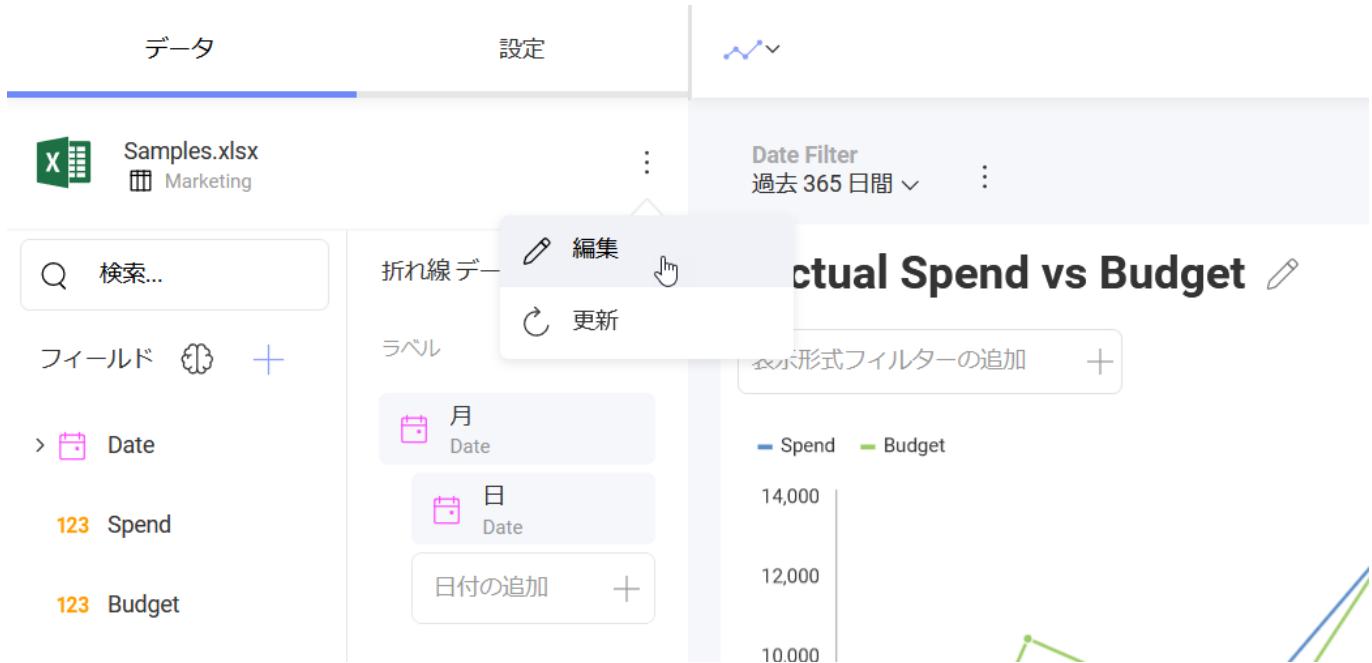
このプロパティは、ダッシュボードを編集するユーザー機能を無効にするために使用できます。



```
revealView.setEditable(false);
```

## showEditDataSource

このプロパティを使用して、ダッシュボードデータソースの編集を無効にすることができます。

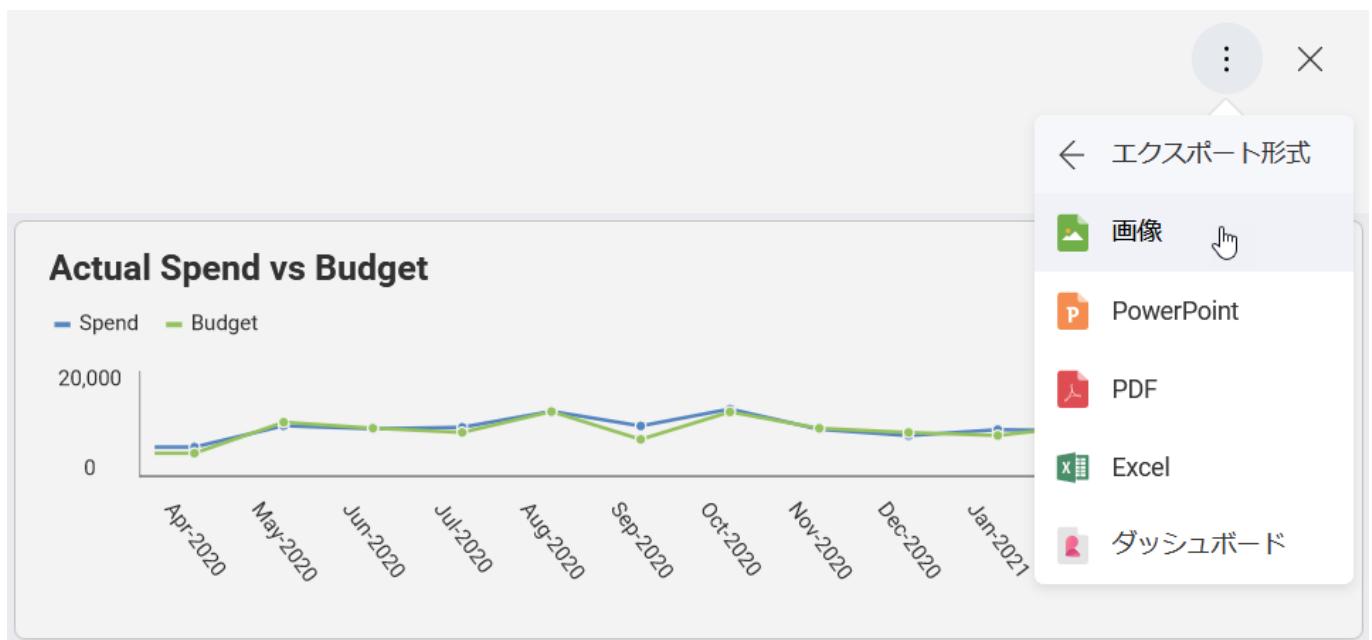


The screenshot shows a dashboard editor interface. At the top, there are tabs for 'データ' (Data) and '設定' (Settings). Below the tabs, there's a file selection area showing 'Samples.xlsx' and 'Marketing'. A date filter is set to '過去 365 日間' (Last 365 days). On the right, a chart titled 'Actual Spend vs Budget' displays two lines: 'Spend' (blue) and 'Budget' (green). The chart has a legend at the bottom left and a date range from April 2020 to January 2021 on the x-axis. A tooltip is shown over the chart area, indicating options like '編集' (Edit), '更新' (Update), and '小小形形式フィルターの追加' (Add small form filter). On the left side, there's a sidebar with a search bar, a field selection section ('フィールド') with 'Date', 'Spend', and 'Budget', and a date range selector ('月' and '日').

```
revealView.showEditDataSource = false;
```

## showExportImage

このプロパティを使用して、ダッシュボードの画像へのエクスポートを無効にすることができます。



The screenshot shows a dashboard titled 'Actual Spend vs Budget' with a line chart comparing 'Spend' and 'Budget' from April 2020 to January 2021. To the right of the chart, a dropdown menu titled 'エクスポート形式' (Export Format) is open, showing options: '画像' (Image), 'PowerPoint', 'PDF', 'Excel', and 'ダッシュボード'. The 'Image' option is selected.

```
revealView.showExportImage = false;
```

## showExportToPowerpoint

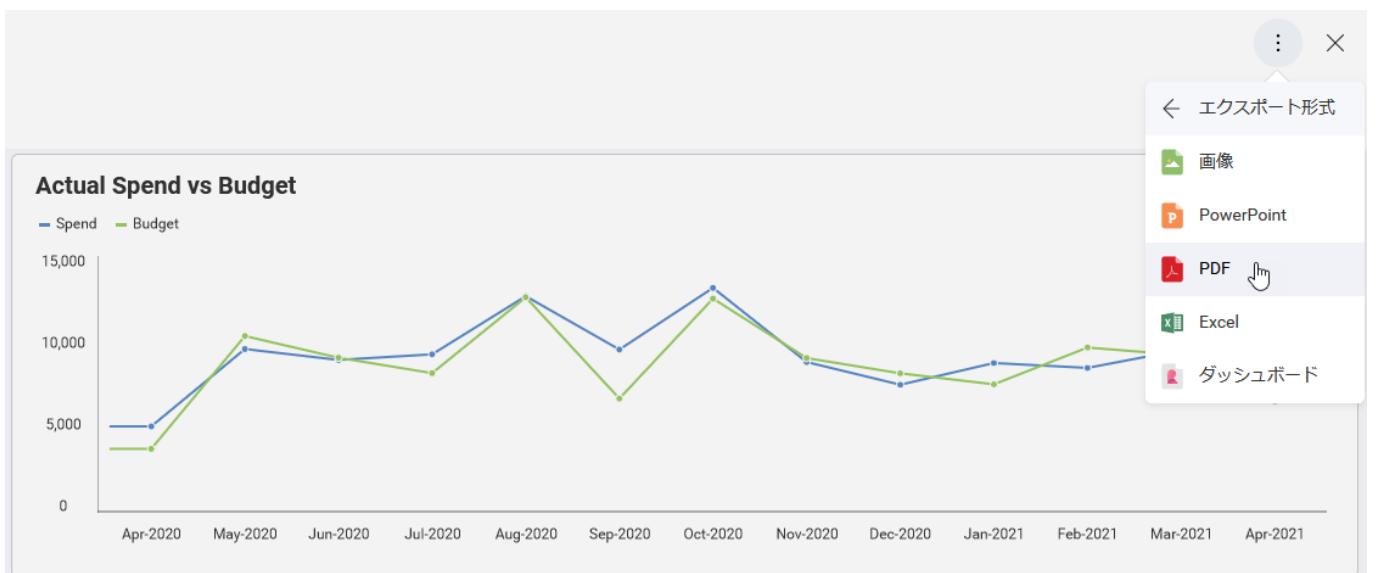
このプロパティを使用して、ダッシュボードのPowerPointへのエクスポートを無効にすることができます。



```
revealView.showExportToPowerpoint = false;
```

## showExportToPDF

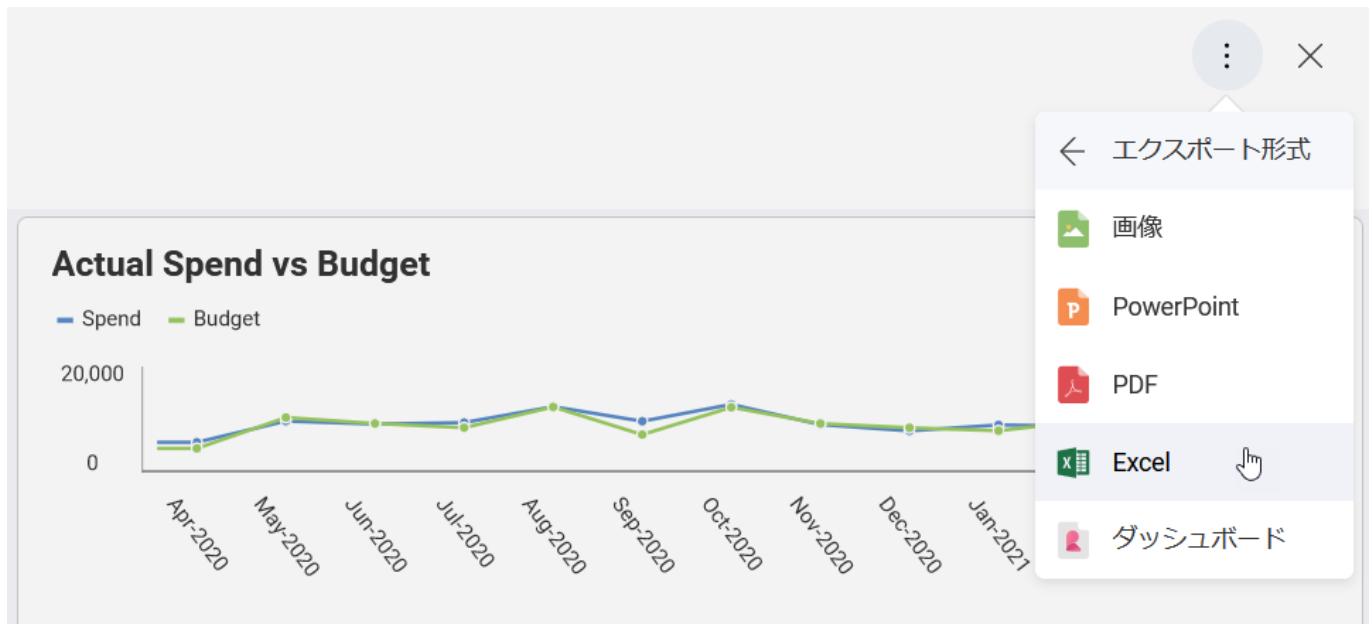
このプロパティを使用して、ダッシュボードの PDF へのエクスポートを無効にすることができます。



```
revealView.showExportToPDF = false;
```

## showExportToExcel

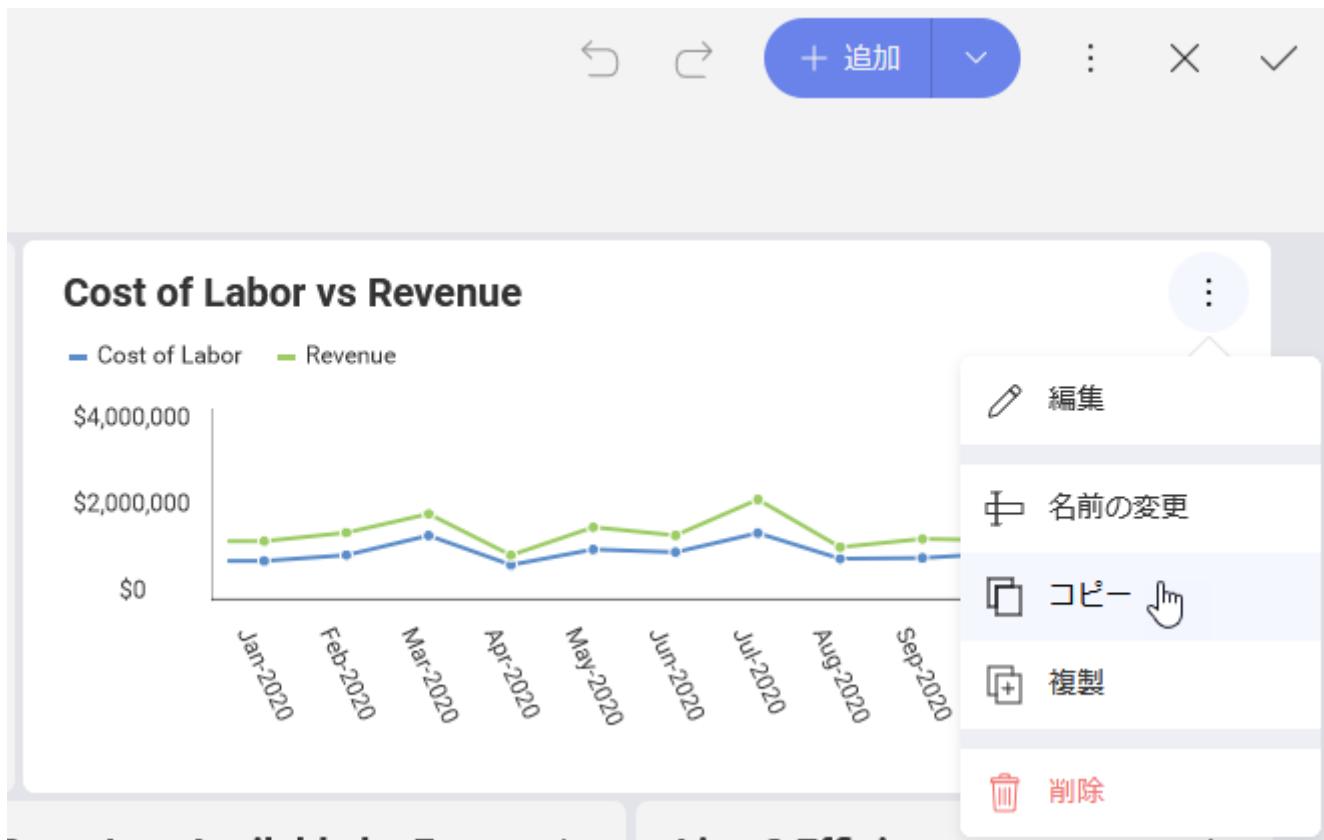
このプロパティを使用して、ダッシュボードの Excel へのエクスポートを無効にすることができます。



```
revealView.showExportToExcel = false;
```

## canCopyVisualization

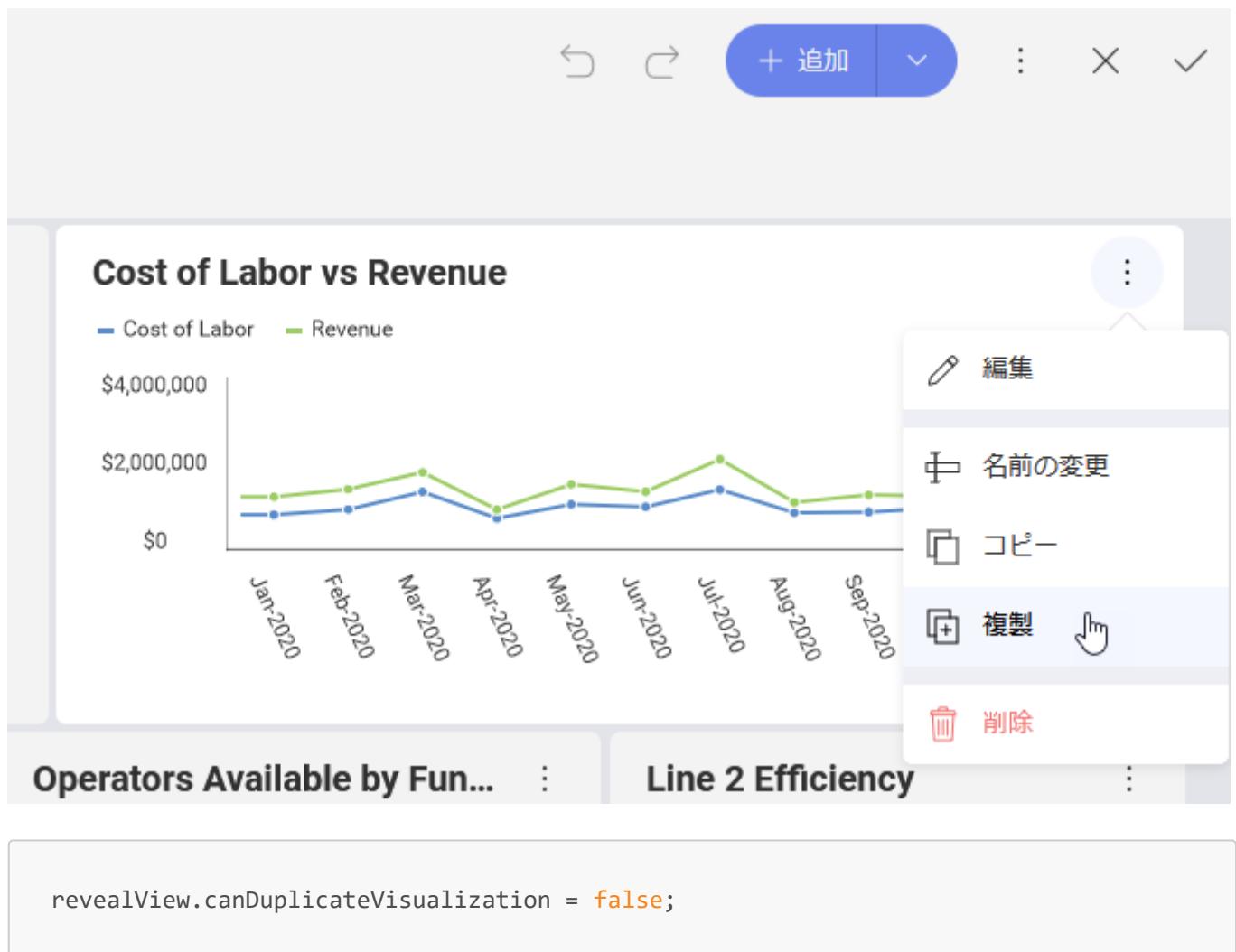
このプロパティは、表示形式をコピーし、後で現在のダッシュボードまたは別のダッシュボードに貼り付ける機能を無効にするために使用できます。



```
revealView.canCopyVisualization = false;
```

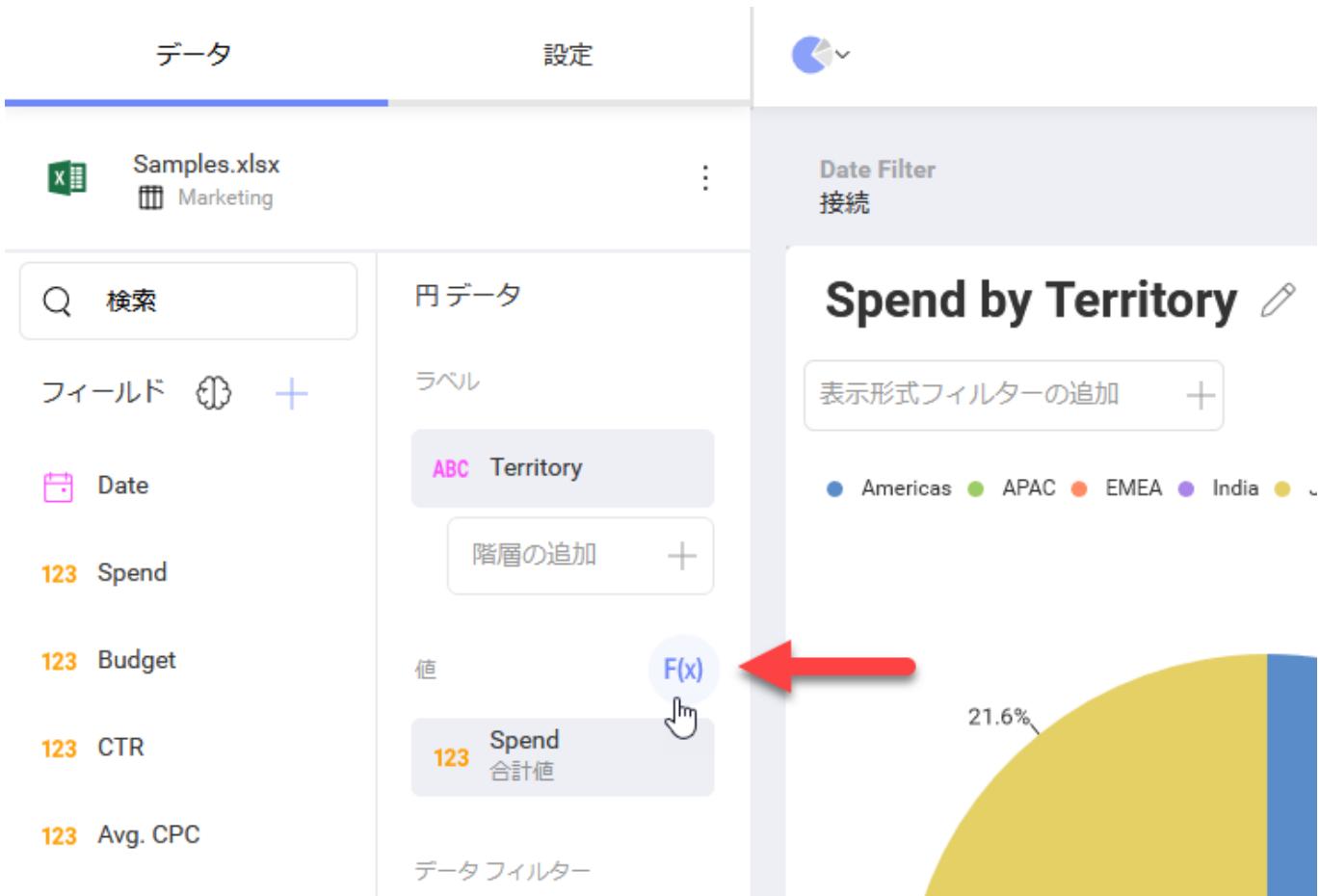
## canDuplicateVisualization

このプロパティは、現在のダッシュボードで表示形式を複製する機能を無効にするために使用できます。



## canAddPostCalculatedFields

このプロパティを使用して、現在のダッシュボードに新しい事後計算フィールドを追加する機能を無効にできます。

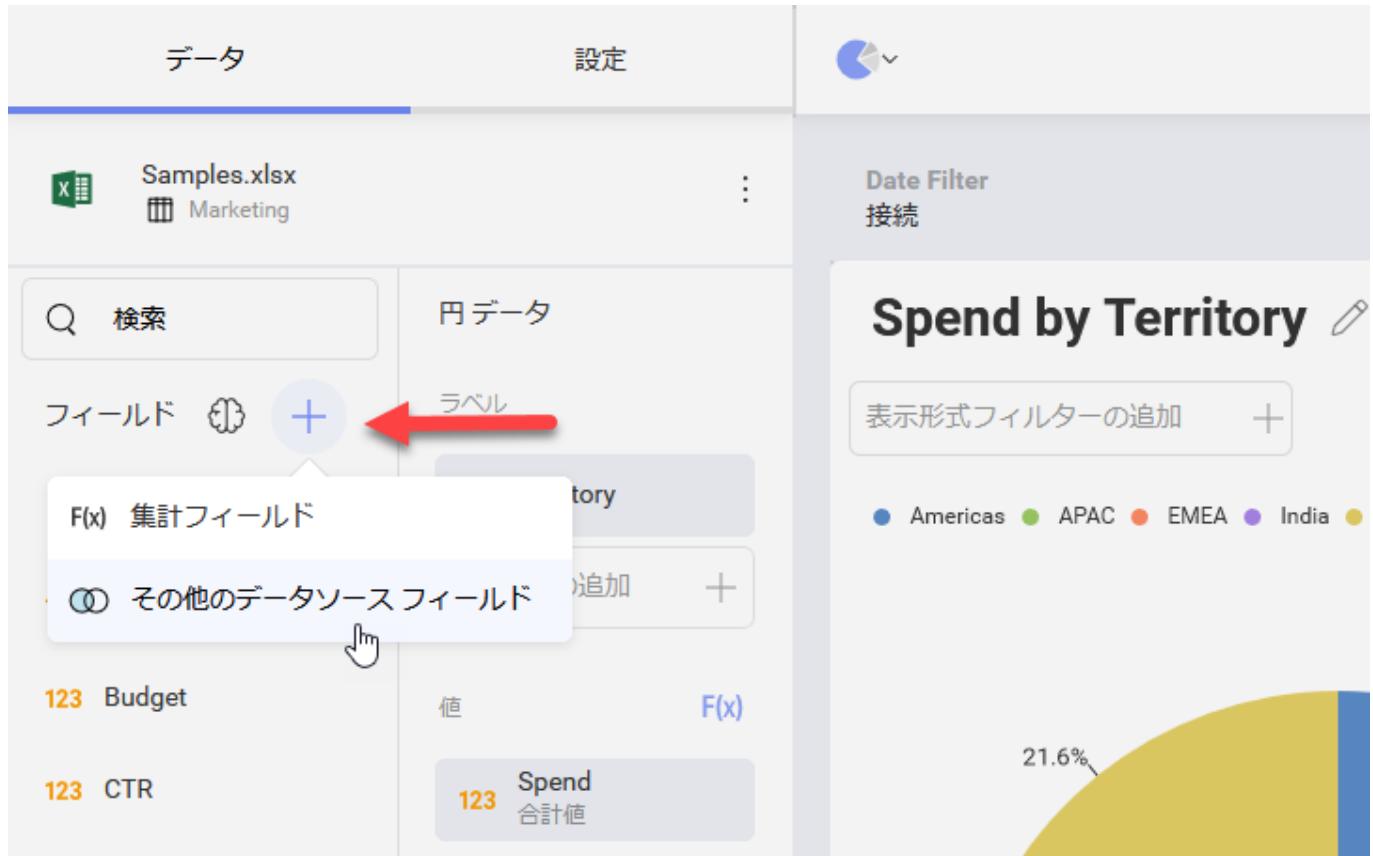


事後計算フィールドはデータセットの新しいフィールドで、すでに集計された値に数式を適用して作成されます。 詳細については、[Reveal ヘルプ](#)をご覧ください。

```
revealView.canAddPostCalculatedFields = false;
```

## canAddCalculatedFields

このプロパティを使用して、現在のダッシュボードに新しい事前計算フィールドを追加する機能を無効にできます。



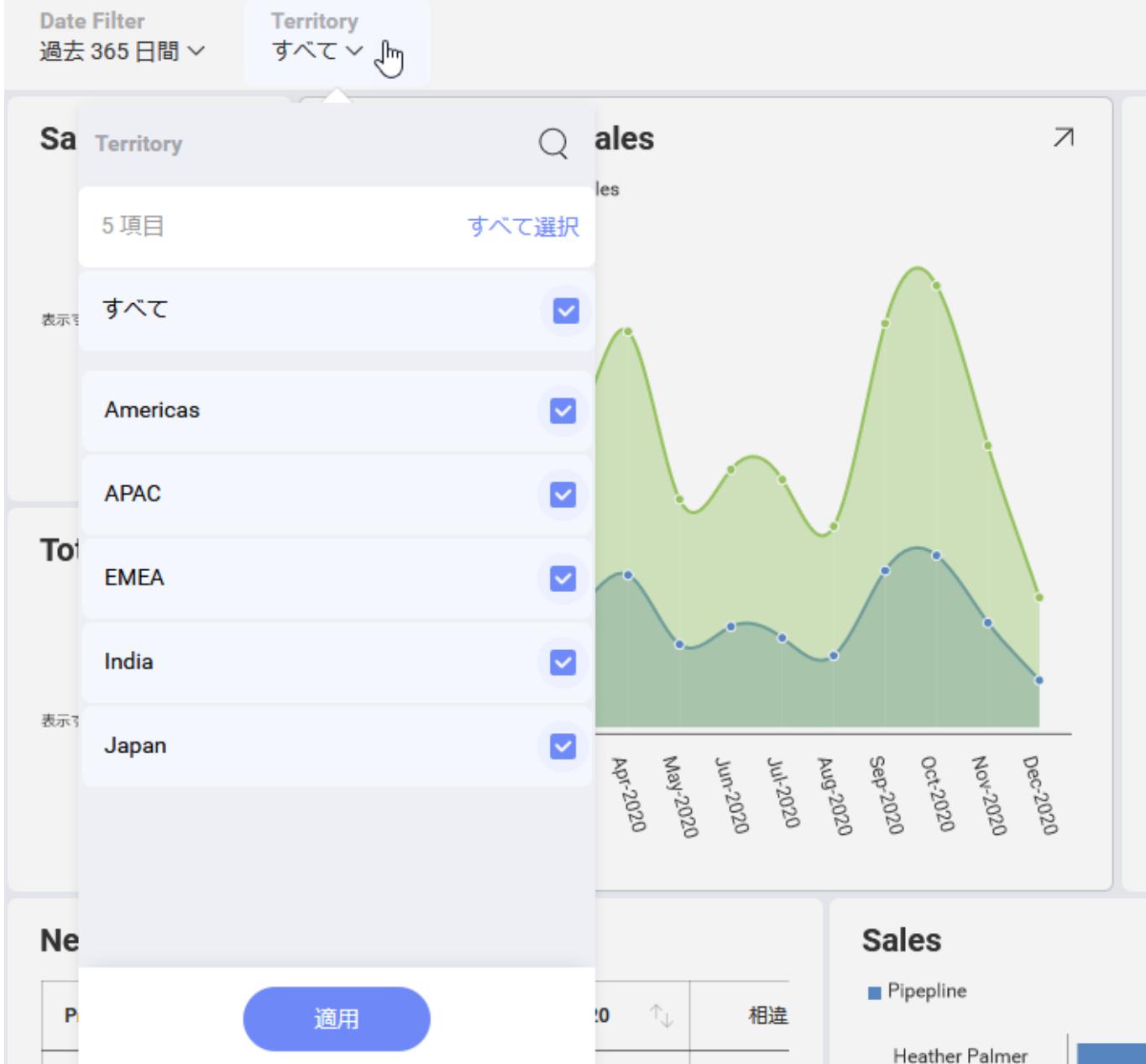
事前計算フィールドはデータセットの新しいフィールドで、データ エディター集計を実行する前に評価されます。 詳細については、[Reveal ヘルプ](#)をご覧ください。

```
revealView.canAddCalculatedFields = true;
```

## showFilters

このプロパティは、ユーザーにダッシュボード フィルター UI を表示または非表示にするために使用できます。

# Sales



ダッシュボード フィルターを使用すると、ダッシュボードの全ての表示形式のコンテンツを一度にフィルター適用できます。

```
revealView.showFilters = true;
```

## canAddDashboardFilter

このプロパティを使用して、[ダッシュボード フィルターの追加] メニュー項目を表示または非表示にすることができます。

# Manufacturing

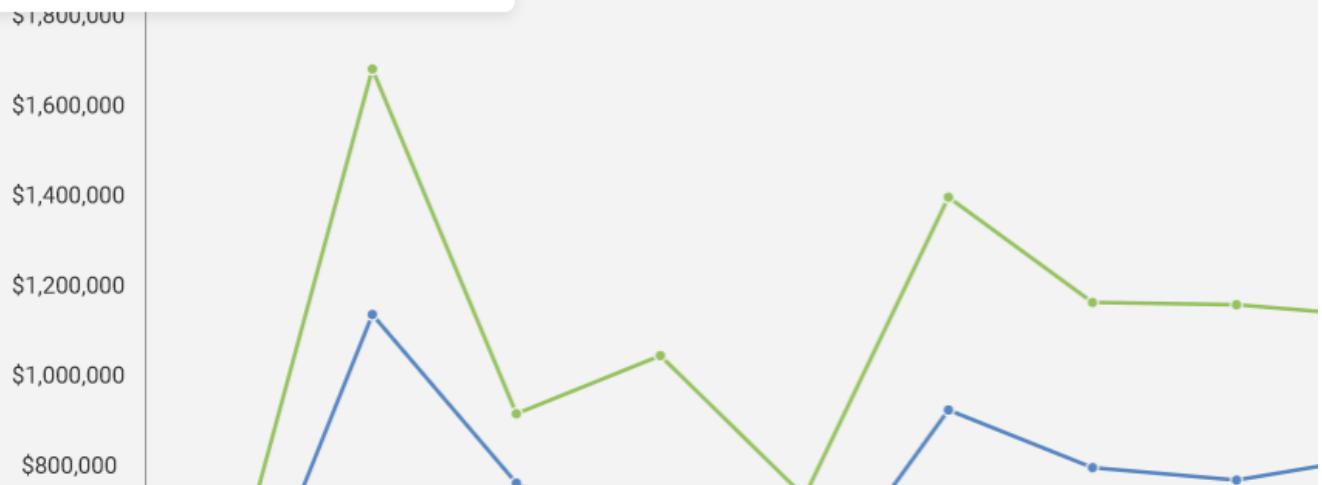


フィルターの追加 +

▽ ダッシュボード フィルターの追加



▽ 日付 フィルターの追加



```
revealView.canAddDashboardFilter = false;
```

## canAddDateFilter

このプロパティを使用して、[日付 フィルターの追加] メニュー項目を表示または非表示にすることができます。

# Manufacturing



フィルターの追加 +

▽ ダッシュボード フィルターの追加



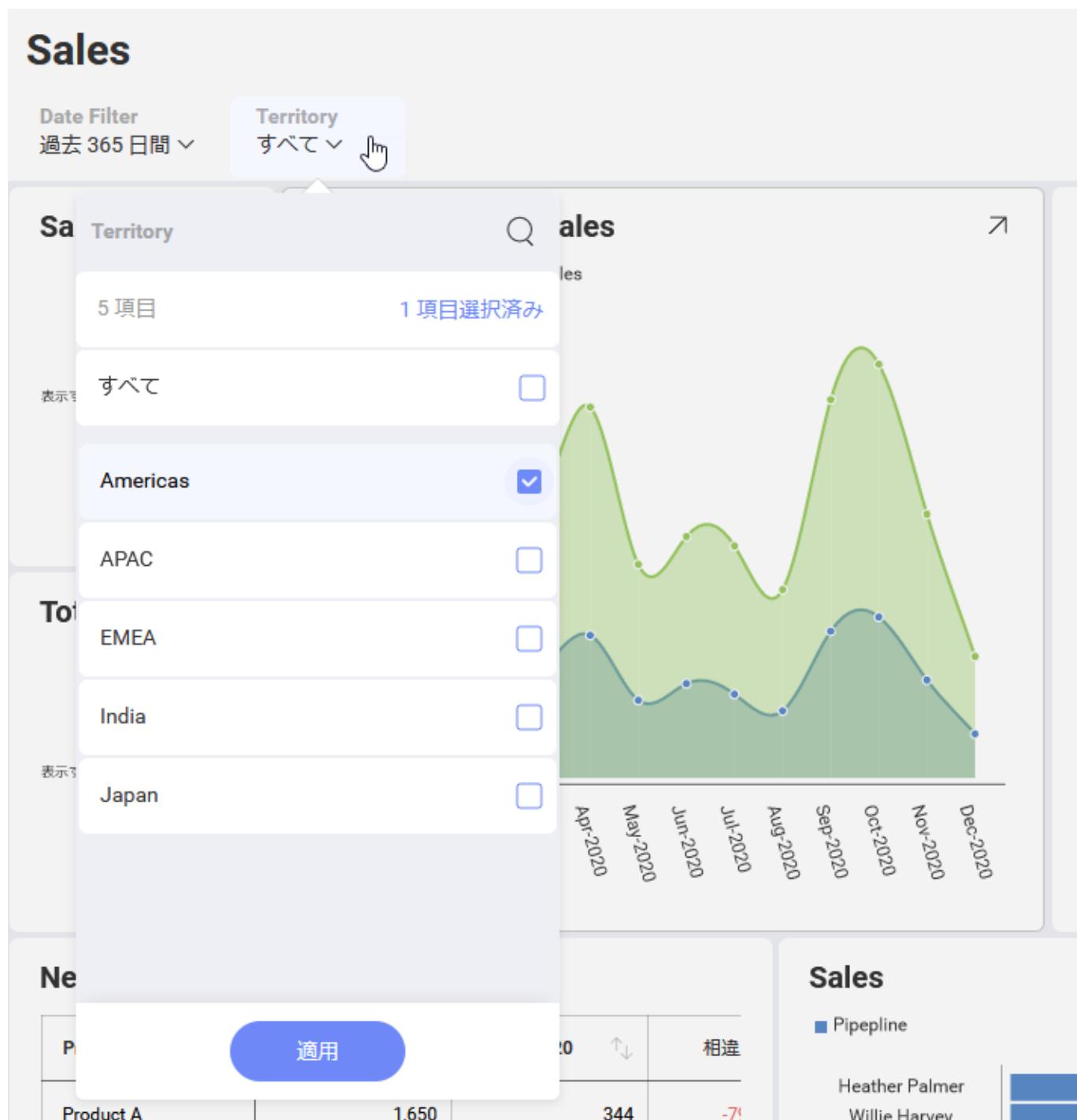
▽ 日付 フィルターの追加



```
revealView.canAddDateFilter = false;
```

## 選択済みフィルター

ダッシュボードを読み込む時に既存のダッシュボード フィルターから最初に選択される値を指定できます。



次のコードスニペットは、ダッシュボード「AppsStats」を読み込む方法を示しています。「Territory」ダッシュボード フィルターの選択値を「Americas」に設定して、ダッシュボードには「Americas」でフィルタリングされたデータが表示されます。

```

var dashboardId = "AppsStats";

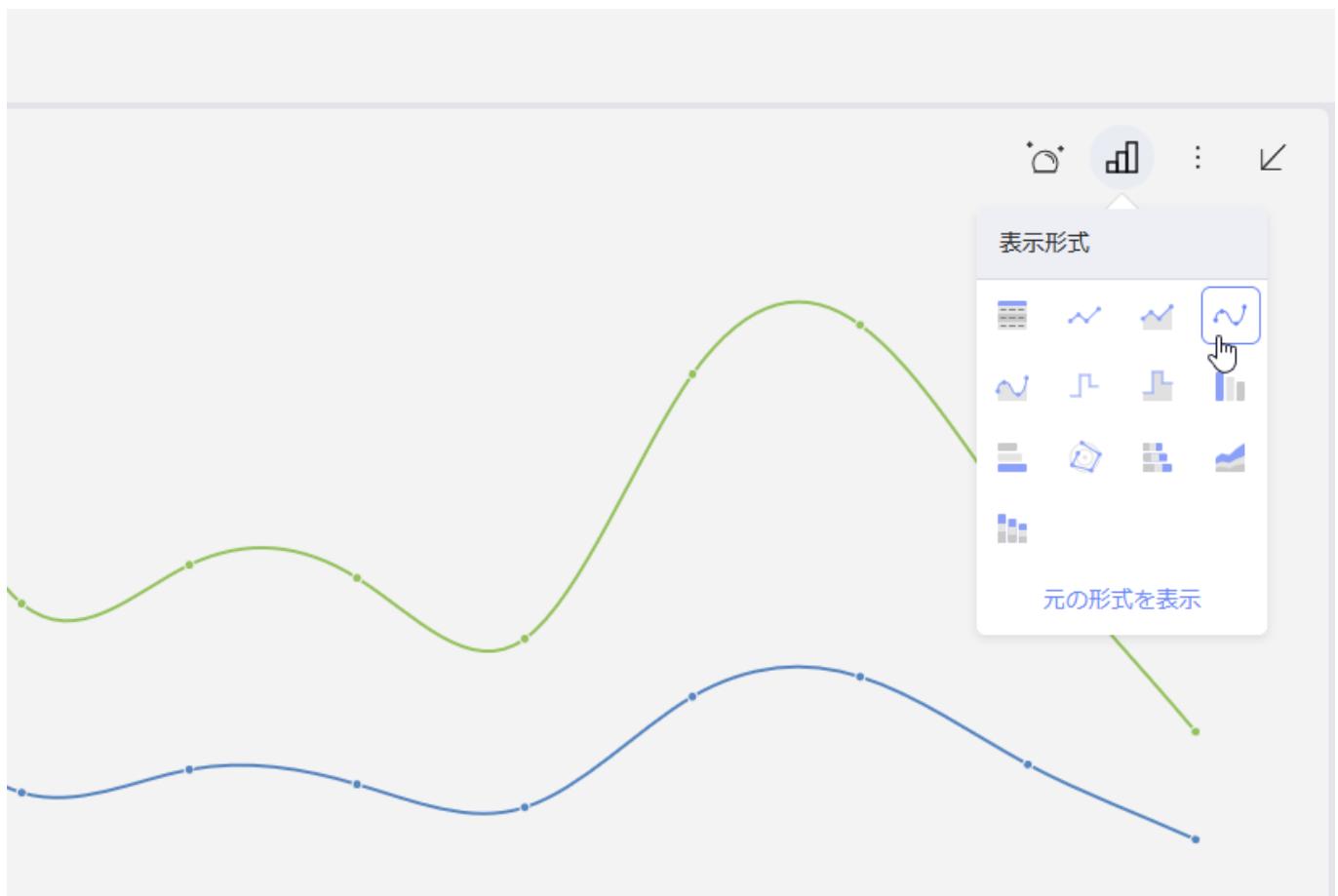
$.ig.RVDashboard.loadDashboard(dashboardId, function (dashboard) {
    dashboard.filters.getByName("Territory").selectedValues = ["Americas"];

    var revealView = new $.ig.RevealView("#revealView");
    revealView.dashboard = dashboard;
}, function (error) {
});

```

## availableChartTypes

このプロパティは、ユーザーが使用できる表示形式タイプをフィルターするために使用できます。



たとえば、以下のように表示形式を追加または削除できます。

```

revealView.availableChartTypes.add($.ig.RVChartType.bulletGraph);
revealView.availableChartTypes.remove($.ig.RVChartType.choropleth);

```

さらに、使用可能な表示形式のみを含む新しい配列を使用できます。

```

revealView.AvailableChartTypes = [$.ig.RVChartType.bulletGraph,
$.ig.RVChartType.choropleth];

```

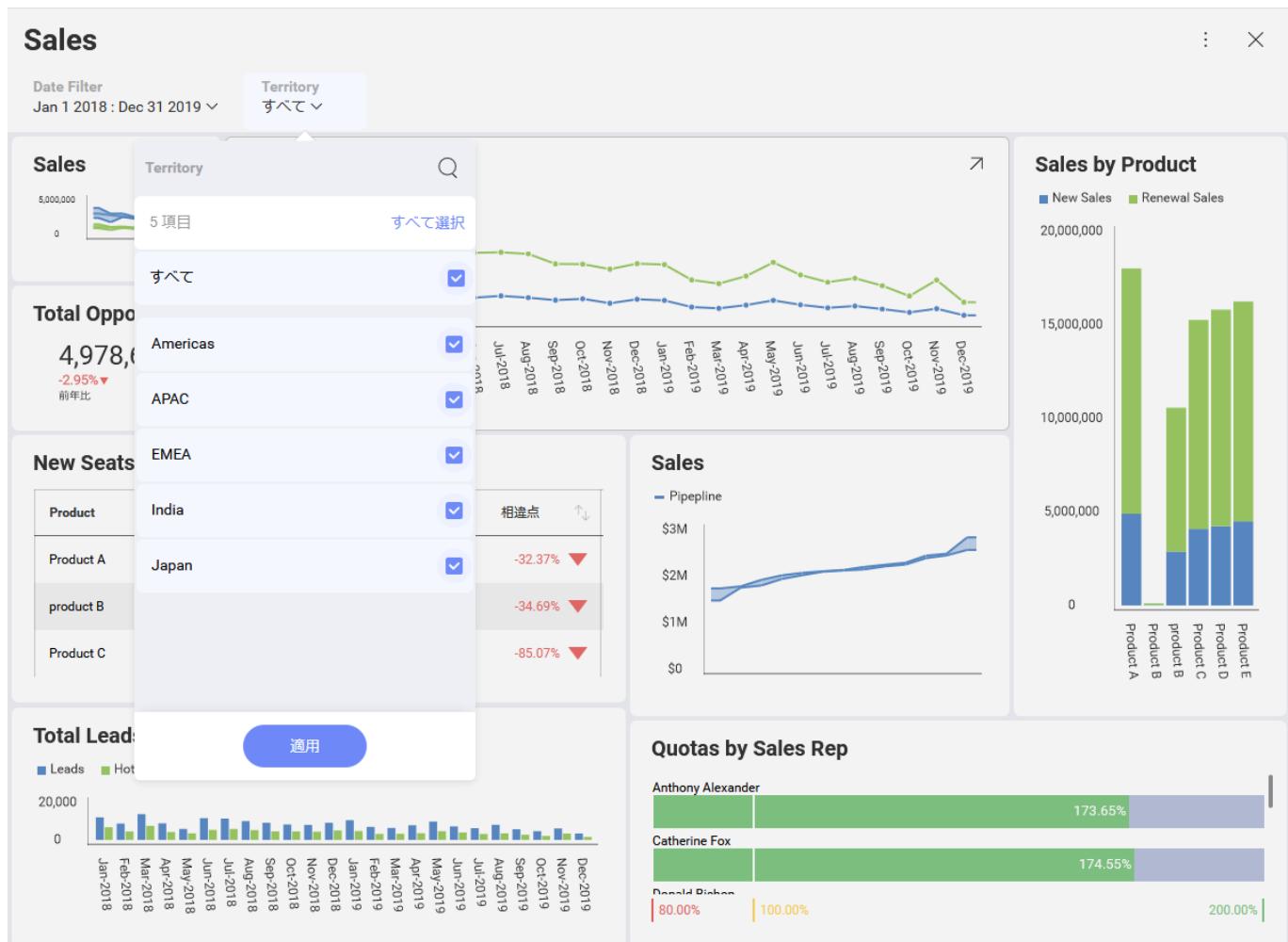


# 動的フィルター選択の設定

## 概要

アプリケーションにカスタム UI を統合し、選択した値の一覧をユーザーに表示する際に、そのユーザー選択をダッシュボードのフィルターと同期させたりすることもできます。

たとえば、現在の地域に基づいて数字を変更する Sales ダッシュボードと、地域を選択するためのカスタム UI を作成する場合、ユーザーが選択を変更後、Sales ダッシュボードにその変更を反映させる必要があります。ほとんどの場合、ダッシュボードに通常表示されているフィルター選択を非表示にします。これにより、ユーザーが画面領域を変更する 2 つの異なる方法を混同することはありません。



次のコードスニペットでは、説明したシナリオを実現する方法について詳しく説明しています。

```
<script type="text/javascript">
    var dashboardId = 'Sales';

    $.ig.RVDashboard.loadDashboard(dashboardId, function (dashboard) {
        var revealView = window.revealView = new $.ig.RevealView("#revealView");
        revealView.showFilters = false;
        revealView.dashboard = dashboard;
    }, function (error) {
```

```

        console.log(error);
    });
    function setSelectedTerritory(territory) {
        window.revealView.dashboard.filters.getByTitle('Territory').selectedValues
= [territory];
    }
</script>

<section style="display:grid;grid-template-rows:30px auto;">
    <section style="display:grid;grid-template-columns:auto auto auto auto auto;">
        <button onclick="setSelectedTerritory('Americas')">Americas</button>
        <button onclick="setSelectedTerritory('APAC')">APAC</button>
        <button onclick="setSelectedTerritory('EMEA')">EMEA</button>
        <button onclick="setSelectedTerritory('India')">India</button>
        <button onclick="setSelectedTerritory('Japan')">Japan</button>
    </section>
    <div id="revealView" style="height:500px;" />
</section>

```

上記でダッシュボードの上部に 5 つのボタン (アメリカ、APAC、EMEA、インド、日本) が追加されました。ダッシュボードの各地域に 1 つずつあります。

## 動的リストの使用

アメリカ大陸、アジア太平洋地域、インドなどの地域は時間の経過とともに変化しませんが、他の値の一覧は変化する可能性があります。この場合、新しい地域がリストに追加されても、新しいボタンは自動的に追加されません。

**\$.ig.RVDashboardFilter.getFilterValues** メソッドを使用して特定のフィルター値の一覧を取得することができます。この場合、次の呼び出しは `_window.territories` に 5 つの **\$.ig.RVFilterValue** オブジェクトを含む配列を残します。

```

var filter = window.revealView.dashboard.getByTitle('Territory');
filter.getFilterValues(function (values) {
    window.territories = values;
}, function (error) {
    console.log(error);
});

```

**\$.ig.RVFilterValue** の `label` 属性を使用して地域の名前を表示し、`values` 属性を使用してフィルターに選択を設定できます。以下のコード スニペットは、自動的に地域を選択するための ComboBox を設定する方法を示します。

```

<script type="text/javascript">
    var dashboardId = 'Sales';

    $.ig.RVDashboard.loadDashboard(dashboardId, function (dashboard) {
        var revealView = window.revealView = new $.ig.RevealView("#revealView");

```

```

revealView.showFilters = false;
revealView.dashboard = dashboard;

var filter = revealView.dashboard.filters.getByName('Territory');
filter.getFilterValues(function (values) {
    window.territories = values;
    var buttonsPanel = $('#buttonsPanel')[0];
    for (var i = 0; i < values.length; i++) {
        var button = $('<button
onclick="setSelectedTerritory(window.territories[' + i + '].values)">' +
values[i].label + '</button>');
        buttonsPanel.append(button[0]);
    }
}, function (error) {
    console.log(error);
});
}, function (error) {
    console.log(error);
});
function setSelectedTerritory(territory) {
    var filter = window.revealView.dashboard.getByName('Territory');
    filter.selectedValues = [territory]);
}
</script>

<section style="display:grid;grid-template-rows:30px auto;">
    <section style="display:grid;grid-template-columns:auto auto auto auto auto;" id="buttonsPanel">
        </section>
        <div id="revealView" style="height:500px;" />
    </section>

```

上の図のように、ボタンを含むセクションには、buttonsPanel という ID が割り当てられています。次に、JQuery を使用してボタンを動的に作成し、それらを DOM 文書に追加します。

変数 window.territories は地域のリストを保持し、後で各ボタンの onclick コードの値を選択するときに使用されます。

# 初期フィルター選択の設定

## 概要

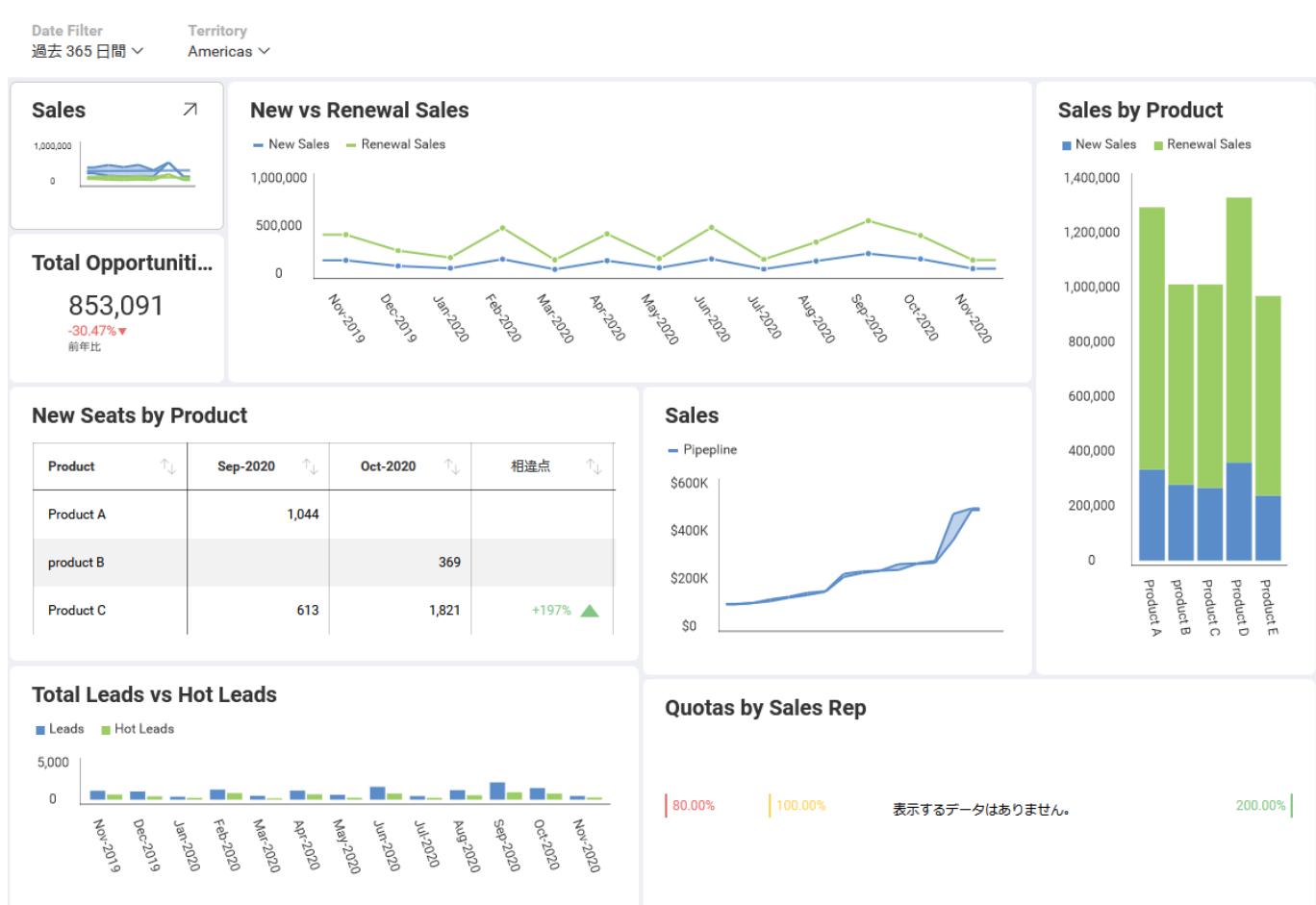
すでに適用されているフィルターを含むダッシュボードを表示したい場合は、ダッシュボード フィルターですべての可視化のコンテンツを一度にスライスすると非常に便利です。これにより、SDK を使用して、すべてのダッシュボードの可視化に対してコンテキスト内に留まる最初のダッシュボード フィルター選択を設定できます。

## 例の詳細

この例では、Sales データを表示するダッシュボードには以下のフィルターがあります。

- 一定期間 (過去 365 日、年度累計など)
- 地域 (南北アメリカ、ヨーロッパ、アジアなど)

## Sales



## コード例

この場合、初期フィルター選択を次のように設定します。

- [年度累計] ([過去 365 日]) の代わりとなる、このダッシュボードのデフォルト設定)。
- 現在のユーザーの地域に関する付けられている売上。

初期化プロセスの一部として、ダッシュボードがロードされたら、ダッシュボード内のフィルターのリストを取得し、これらのフィルターを使用して、ダッシュボード オブジェクトを介して選択した値を設定し、最後にそれを revealView のダッシュボード プロパティに割り当てることができます。

```
<script type="text/javascript">
    var dashboardId = 'Sales';

    $.ig.RVDashboard.loadDashboard(dashboardId, function (dashboard) {

        dashboard.filters.getTitle("Territory").selectedValues =
        [getCurrentUser().territory];
        dashboard.dateFilter = new
        $.ig.RVDateDashboardFilter($.ig.RVDateFilterType.YearToDate);

        var revealView = new $.ig.RevealView("#revealView");
        revealView.dashboard = dashboard;

    }, function (error) {
        console.log(error);
    });
</script>

<div id="revealView" style="height:500px;" ></div>
```

[!NOTE] 上記のコードは、**getCurrentUser().territory** が現在のユーザーの地域を返すことを前提としています。

## フィルターの非表示化

ユーザーが自分の地域以外のデータにアクセスしたくない場合があります。このような場合、ダッシュボード フィルターを含むパネルを非表示にするように **\$.ig.RevealView** オブジェクトを設定することで、フィルターへのアクセスを制限できます。

```
revealView.showFilters = false;
```

この設定では、関連する地域のデータのみを表示するようにユーザーを制限します。

最後に、それでもユーザーに日付フィルターの選択を変更させたい場合は、[動的フィルター選択の設定](#) トピックをご覧ください。ユーザーが日付フィルターを変更できるようにするための独自の UI を作成する方法についての情報があります。

## Web サーバー .NET API リファレンス

ここでは、Reveal SDK、特に Web サーバー .NET API に関する詳しい情報を見つけることができます。完全なリファレンスについては、[このリンク](#)をご覧ください。

最も一般的に使用されるクラスとインターフェース:

## **SDK の主な概念と機能:**

[RevealSdkContextBase](#)

[RevealEmbedSettings](#)

[Dashboard クラス](#)

## **データ ソース**

[IRVDataSourceProvider](#)

[RVSqIIServerDataSource](#)

[RVSqIIServerDataSourceItem](#)

[RVRESTDataSource](#)

[RVRESTDataSourceItem](#)

[RVJsonDataSource](#)

[RVJsonDataSourceItem](#)

[IRVDataProvider](#)

[RVInMemoryData](#)

[RVInMemoryDataSource](#)

[RVInMemoryDataSourceItem](#)

## **認証**

[IRVAuthenticationProvider](#)

[IRVDataSourceCredential](#)

[RVBearerTokenDataSourceCredential](#)

[RVUsernamePasswordDataSourceCredential](#)

## **Web クライント JS API リファレンス**

ここでは、Reveal SDK、特に Web クライント JavaScript API に関する詳しい情報を見つけることができます。完全なリファレンスについては、[このリンク](#)をご覧ください。

## **最も一般的に使用されるクラスとインターフェース**

### **SDK の主な概念と機能:**

[RevealView](#)

[RVDashboard](#)

[RevealSdkSettings](#)

[RVVisualization](#)

[RevealTheme](#)

[RevealUtility](#)

## **データ ソース**

[RVSqIIServerDataSource](#)

[RVSqIIServerDataSourceItem](#)

[RVRESTDataSource](#)

[RVRESTDataSourceItem](#)

[RVJsonDataSource](#)

[RVJsonDataSourceItem](#)

[RVInMemoryDataSource](#)

[RVInMemoryDataSourceItem](#)

## フィルタリング

[RVDateDashboardFilter](#)

[RVDashboardFilter](#)

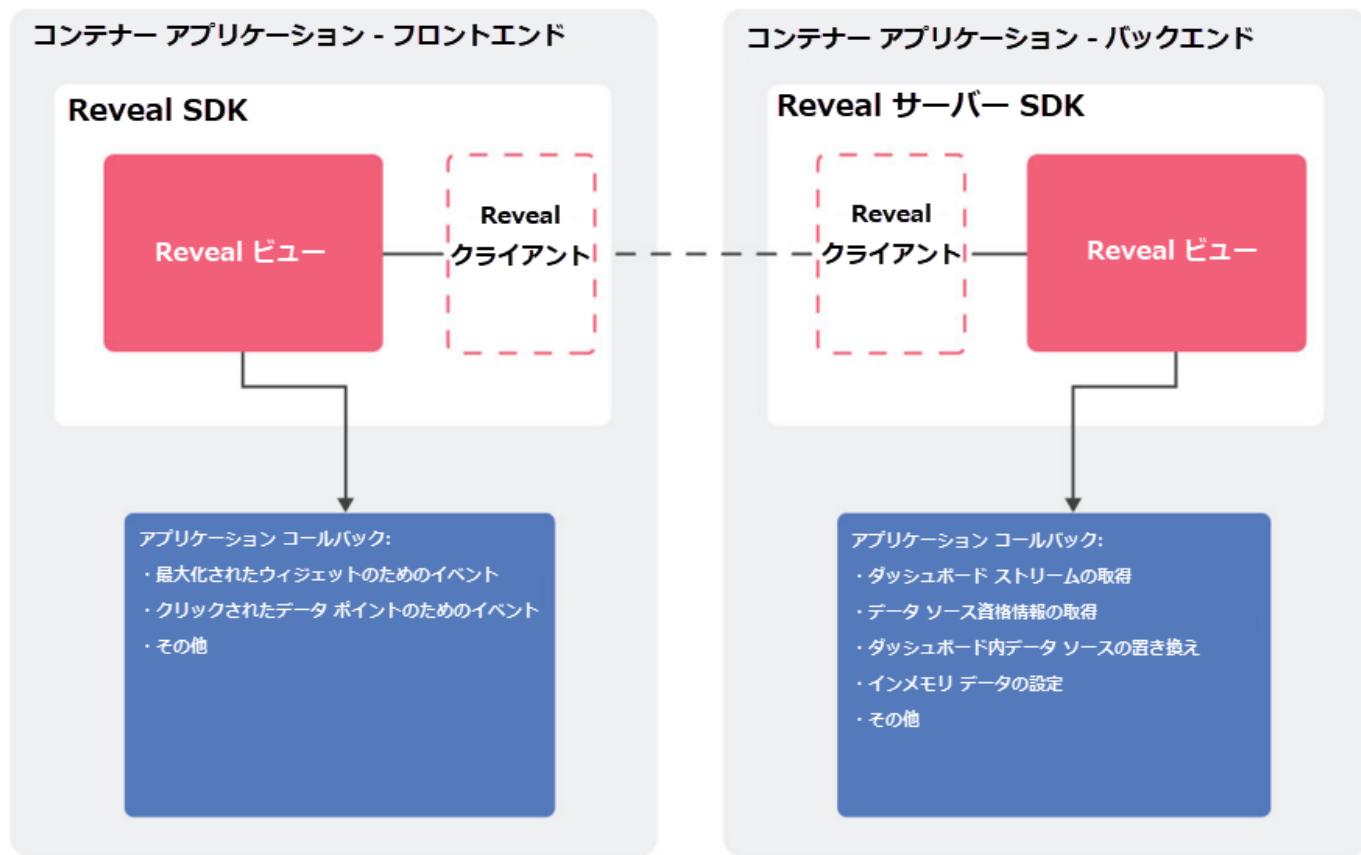
[RVFilterValue](#)

# JAVA SDK の概要

Web アプリケーションに Reveal を埋め込む場合、2 つのコンポーネントが常に関係しているため、アーキテクチャはネイティブ アプリよりも少々複雑になります:

- **Reveal クライアント SDK:** ウェブ アプリケーションに統合される必要がある JavaScript ライブラリのセット。現在サポートされているフレームワークは、jQuery、Angular、および React です。
- **Reveal サーバー SDK:** サーバー アプリケーションに統合されるサーバー側コンポーネント。現在のライブラリは JAVA SDK 11+ を必要とし、[Maven \(英語\)](#) モジュールのセットとして配布されます。

次の図では、Reveal Web SDK を埋め込んだ Web アプリケーションのアーキテクチャを可視化しています:



上記の示したように、SDK はネイティブ アプリケーションとほとんど同じように機能します。違いは、一部のコールバックはクライアント側で呼び出され (データポイントがクリックされたときに送信されるイベントなど)、その他はサーバー側で呼び出される (ダッシュボードのロードまたはメモリ内データの提供のためのコールバックなど) 点です。

## クライアント側とサーバー側の部分のホスト (異なるサーバーを利用)

クライアント側とサーバー側のパートを個別に、たとえば異なる URL でホストできます。

これには、以下のようにウィンドウ オブジェクトのプロパティを設定します:

```
$.ig.RevealSdkSettings.setBaseUrl("http://localhost:8080/upmedia/reveal-api");
```

使用される形式は **http://[server]:[port]/[application]/reveal-api** であることに注意してください。プロパティを正しく設定するには、**/reveal-api** を含める必要があります。

このプロパティは、[\\$.ig.RevealView のインスタンス化の前に設定します。](#)

# UpMedia サンプルの実行

---

GitHub から [UpMedia サンプル](#) を取得した後、実行に役立つ詳細情報を以下に示します。

## Tomcat の UpMedia サンプル アプリケーション

### 要件

- [Java SDK \(英語\)](#) 11.0.10 以降を推奨します。
- [Tomcat \(英語\)](#) 9.0.41 以降を推奨します。
- [Eclipse for Enterprise Java Developers \(英語\)](#) バージョン 2020-12 以降を推奨します。
- Maven リポジトリと依存関係を追加する必要があります。 詳細については、[セットアップと構成](#) を参照してください。

### 手順

#### 1. プロジェクトを読み込みます。

1. Eclipse で、*File > Import > Existing Projects into Workspace* に移動します。
2. *Select root directory* で、Reveal の SDK パスを選択し、UpMedia プロジェクトも選択します。
3. *Copy projects into workspace* オプションをチェックします。
4. [Finish] をクリックします。

プロジェクトが読み込まれ、検査する準備ができました。

#### 2. Tomcat でプロジェクトを実行する

1. UpMedia プロジェクトで *Run on Server* を右クリックします。
2. Tomcat 9 を選択し、Tomcat のインストールパスへのルートパスを設定します。
3. デフォルト設定でプロジェクトを実行します。

#### 3. サンプルをブラウザーで可視化します。

- Eclipse の内部ブラウザーには Windows に関する既知の問題があります。代わりに Google Chrome または別のブラウザーを使用してください (デフォルトの URL は <http://localhost:8080/upmedia> です)。

WAR ファイルを使用する代替手順:

1. *Run As -> Maven build* に移動します。
2. *package* ゴールを使用します。
3. 作成した WAR ファイルを手動で Tomcat に配備します。

## Spring Backend を使用した UpMedia React のサンプル

### 要件

- [Java SDK \(英語\)](#) 11.0.10 以降を推奨します。
- [NodeJS \(英語\)](#) 14.15.4 以降を推奨します。 NPM バージョン 6.14.10 を推奨します。

- Maven リポジトリと依存関係を追加する必要があります。詳細については、[セットアップと構成](#)を参照してください。

## 手順

### 1. Spring アプリケーションを実行します。

1. *upmedia-backend-spring* フォルダーで Spring Boot アプリケーションを特定します。
2. 以下のコマンドを実行してサンプルアプリケーションを実行します。

```
mvn spring-boot:run  
npm start
```

3. <http://localhost:8080/upmediabackend/reveal-api/DashboardFile/Sales> にアクセスしてサーバーを確認します。その結果、Sales サンプルダッシュボードの JSON ドキュメントを取得します。Reveal サービス [/upmediabackend/reveal-api/](http://upmediabackend/reveal-api/) にあります。

### 2. React アプリケーションを実行します。

1. *upmedia-react* フォルダーで React アプリケーションを特定します。
2. React アプリケーションを通常通りに実行します。

```
npm install  
npm start
```

3. <http://localhost:3000> で React アプリケーションにアクセスします。

以下のコンポーネントをお試しください。

- **Filters:** 既存のダッシュボードを開き、フィルタリング エクスペリエンスをカスタマイズする方法を示します。
- **Linking:** 別のダッシュボードへのリンクを持つ既存のダッシュボードを開く方法、およびリンクを正しく構成する方法を示します。
- **CreateDashboard:** ダッシュボード エディターを開いて新しいダッシュボードをゼロから作成する方法を示します。また、新しい表示形式を作成するときにユーザーに表示されるデータ ソースのリストを設定する方法も示します。

Reveal の Web クライアント SDK の詳細については、[こちら](#)を参照してください。

## Tomcat Backend を使用した UpMedia React のサンプル

### 要件

- [Java SDK \(英語\)](#) 11.0.10 以降を推奨します。
- [Tomcat \(英語\)](#) 9.0.41 以降を推奨します。
- [Eclipse for Enterprise Java Developers \(英語\)](#) バージョン 2020-12 以降を推奨します。

- Maven リポジトリと依存関係を追加する必要があります。詳細については、[セットアップと構成](#)を参照してください。

## 手順

### 1. プロジェクトを読み込みます。

1. Eclipse で、*File > Import > Existing Projects into Workspace* に移動します。
2. *Select root directory* で、Reveal の SDK パスを選択し、*upmedia-backend-tomcat* プロジェクトも選択します。
3. *Copy projects into workspace* オプションをチェックします。
4. [Finish] をクリックします。

プロジェクトが読み込まれ、検査する準備ができました。

### 2. Tomcat でプロジェクトを実行する

1. *upmedia-backend-tomcat* プロジェクトで *Run on Server* を右クリックします。
2. Tomcat 9 を選択し、Tomcat のインストールパスへのルートパスを設定します。
3. デフォルト設定でプロジェクトを実行します。
4. <http://localhost:8080/upmediabackend/reveal-api/DashboardFile/Sales> にアクセスしてサーバーを確認します。その結果、Sales サンプルダッシュボードの JSON ドキュメントを取得します。

### 3. React アプリケーションを実行します。

1. *upmedia-react* フォルダーで React アプリケーションを特定します。
2. React アプリケーションを通常通りに実行します。

```
npm install  
npm start
```

3. <http://localhost:3000> で React アプリケーションにアクセスします。

以下のコンポーネントをお試しください。

- **Filters:** 既存のダッシュボードを開き、フィルタリング エクスペリエンスをカスタマイズする方法を示します。
- **Linking:** 別のダッシュボードへのリンクを持つ既存のダッシュボードを開く方法、およびリンクを正しく構成する方法を示します。
- **CreateDashboard:** ダッシュボード エディターを開いて新しいダッシュボードをゼロから作成する方法を示します。また、新しい表示形式を作成するときにユーザーに表示されるデータ ソースのリストを設定する方法も示します。

Reveal の Web クライアント SDK の詳細については、[こちら](#)を参照してください。

# セットアップと構成 (JAVA SDK)

## 前提条件 (Maven)

Reveal Java SDK は、[Maven \(英語\)](#) モジュールのセットとして配布されます。SDK ライブラリを操作するには、Reveal の Maven リポジトリへの参照と、Maven pom.xml ファイルの依存関係を追加する必要があります。

以下のリポジトリを追加します:

```
<repositories>
  <repository>
    <id>reveal.public</id>
    <url>https://maven.revealbi.io/repository/public</url>
  </repository>
</repositories>
```

以下の依存関係を追加します:

```
<dependency>
  <groupId>com.infragistics.reveal.sdk</groupId>
  <artifactId>reveal-sdk</artifactId>
  <version>version_number</version>
</dependency>
```

version\_number を **0.9.6** のような番号に置き換えます。

Maven についてご不明な点がございましたら、次の[リンク \(英語\)](#) を参照してください。

[!NOTE] Oracle データベースを使用している場合は、アプリケーションにドライバーを追加する必要があります。

## セットアップと構成 (汎用サーバー)

Reveal を既存のアプリケーションと統合するには、次の一般的な手順に従う必要があります:

1. 既存のアプリの実装に依存関係を追加します。
2. Reveal SDK に依存関係を追加します。
3. Reveal を初期化します。
4. サーバー側エクスポートを有効にします。

Tomcat または Spring の構成については、以下のリンクを参照してください。

- [Tomcat サーバー](#)
- [Spring サーバー](#)
- [Oracle Server](#)

手順 1 – アプリケーションに依存関係を追加します。

必要な手順に従って、既存のアプリケーションに依存関係を追加します。

手順 2 - Reveal SDK に依存関係を追加します。

reveal-sdk に依存関係を追加し、SDK のバージョンを指定します。

```
<dependency>
    <groupId>com.infragistics.reveal.sdk</groupId>
    <artifactId>reveal-sdk</artifactId>
    <version>version_number</version>
</dependency>
```

version\_number を **1.0.1821** のような番号に置き換えます。

手順 3 - Reveal を初期化します。

Reveal を初期化するには、**RevealEngineInitializer.initialize** を使用します。

初期パラメーターなしでメソッドを呼び出すことが可能です。

```
RevealEngineInitializer.initialize();
```

ただし、ほとんどの場合、以下の例のようにパラメーターを使用します。

```
RevealEngineInitializer.initialize(
    new InitializeParameterBuilder()
        .setAuthProvider(new RevealAuthenticationProvider())
        .setUserContextProvider(new RevealUserContextProvider())
        .setDashboardProvider(new RevealDashboardProvider())
        .setDataSourceProvider(new UpMediaDataSourceProvider())
        .setDataProvider(new UpMediaInMemoryDataProvider())
        .setMaxConcurrentImageRenderThreads(2)
        .setLicense("SERIAL_KEY_TO_BE_USED")
        .build());
```

これらのパラメーターは Reveal のカスタマイズに使用される**プロバイダー**です。Reveal をアプリケーションに統合する場合は、独自のプロバイダーを作成する必要があります。

**RevealEngineInitializer.initialize** に渡される利用可能なパラメーターは次のとおりです:

- *setAuthProvider*。ここで、認証を解決し、IRVAuthenticationProvider を実装するカスタム クラスを含める必要があります。
- *setUserContextProvider*。IRVUserContextProvider を実装するユーザーに関する情報を提供するカスタム クラス。

- *setDashboardProvider*。ダッシュボードを置換または変更するカスタム クラス。  
IRVDashboardProvider を実装します。
- *setDataSourceProvider*。データソースを置換または変更するカスタム クラス。IRVDataProvider を実装します。
- *setDataProvider*。ダッシュボードのインメモリ データを返すカスタム クラス。IRVDataProvider を実装します。
- *setLicense*。ここでは、シリアルキーを含めて SDK ライセンスを構成できます。

ダッシュボード プロバイダーを実装する方法の詳細については、GitHub の [UpMedia サンプル \(英語\)](#) を参照してください。

#### 手順 4 - サーバー側エクスポートを有効にします。

Java SDK は、ダッシュボードをさまざまな形式 (Image、PDF、PPT、Excel) にエクスポートするためにいくつかのネイティブ コンポーネントを使用します。

これらの形式の 1 つ以上にサーバー側をエクスポートする場合は、[サーバー側のエクスポート構成](#)を参照してください。

### セットアップと構成 (クライアント)

以下は、Reveal Web クライアント SDK を設定するための手順です。

1. [依存関係を確認します。](#)
2. [Web クライアント SDK を参照します。](#)
3. [Web クライアント SDK をインスタンス化します。](#)

#### 1. 依存関係の確認

Reveal Web クライアント SDK には、次のサードパーティの参照があります:

- [jQuery](#) 2.2 またはそれ以上
- [Day.js](#) 1.8.15 またはそれ以上
- [Quill RTE](#) 1.3.6 またはそれ以上
- [Marker Clusterer](#) 3 またはそれ以上
- [Google Maps](#) 3 またはそれ以上

#### 2. Web クライアント SDK の参照

Web ページで **\$.ig.RevealView** コンポーネントを有効にするには、いくつかのスクリプトを含める必要があります。これらのスクリプトは Reveal Web クライアント SDK の一部として提供されます。

```
<script src="~/Reveal/infragistics.reveal.js"></script>
```

JavaScript ファイルは "<InstallationDirectory>\SDK\Web\JS\Client" にあります。

#### 3. Web クライアント SDK のインスタンス化

ダッシュボードのプレゼンテーションは、Web クライアント SDK を介してネイティブに処理されます。

以下の手順に従って作業を開始します:

1. "id" を指定して <div /> 要素を定義し、**\$.ig.RevealView** コンストラクターを呼び出します。

[!NOTE] サーバー側とクライアント側のパートを個別にホストする 別々のサーバーでクライアント側とサーバー側のパートをホストする場合は、次の手順を続行する前に[こちら](#)を参照してください。

2. *dashboardId* と成功およびエラー ハンドラーを指定して **\$.ig.RVDashboard.loadDashboard** を呼び出します。

3. 成功ハンドラーで、ダッシュボードが描画される DOM 要素のセレクターを渡すことにより、**\$.ig.RevealView** コンポーネントをインスタンス化します。

最後に、取得したダッシュボードを使用し、**\$.ig.RevealView** のダッシュボード プロパティに設定する必要があります。

## サンプルコード

```
<!DOCTYPE html>
<html>
  <head>
    :
    <script type="text/javascript">
      var dashboardId = "dashboardId";

      $.ig.RVDashboard.loadDashboard(
        dashboardId,
        function (dashboard) {
          var revealView = new $.ig.RevealView("#revealView");
          revealView.dashboard = dashboard;
        },
        function (error) {
          //Process any error that might occur here
        }
      );
    </script>
  </head>
  <body>
    <div id="revealView" style="height:500px;" />
  </body>
</html>
```

## Oracle データベースの操作

上記のように、Reveal Java SDK は、[Maven](#) モジュールのセットとして配布されます。SDK ライブラリを操作するには、Maven pom.xml ファイルに 2 つの参照と依存関係を追加する必要があります。

以下のリポジトリを追加します:

```
<repositories>
  <repository>
    <id>reveal.public</id>
    <url>http://revealpackages.eastus.cloudapp.azure.com/repository/public</url>
  </repository>
  <repository>
    <id>jeecg</id>
    <url>http://maven.jeecg.org/nexus/content/repositories/jeecg/</url>
  </repository>
</repositories>
```

そして、次の依存関係を追加します:

```
<dependencies>
  <dependency>
    <groupId>com.infragistics.reveal.sdk</groupId>
    <artifactId>reveal-sdk</artifactId>
    <version>version_number</version>
  </dependency>
  <dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc14</artifactId>
    <version>10.2.0.5.0</version>
  </dependency>
</dependencies>
```

# Tomcat サーバーのセットアップと構成

## 前提条件 (Maven)

Reveal Java SDK は、[Maven \(英語\)](#) モジュールのセットとして配布されます。SDK ライブラリを操作するには、Reveal の Maven リポジトリへの参照と、Maven pom.xml ファイルの依存関係を追加する必要があります。

以下のリポジトリを追加します:

```
<repositories>
  <repository>
    <id>reveal.public</id>
    <url>https://maven.revealbi.io/repository/public</url>
  </repository>
</repositories>
```

以下の依存関係を追加します:

```
<dependency>
  <groupId>com.infragistics.reveal.sdk</groupId>
  <artifactId>reveal-sdk</artifactId>
  <version>version_number</version>
</dependency>
```

version\_number を **0.9.6** のような番号に置き換えます。

Maven についてご不明な点がございましたら、次の[リンク \(英語\)](#) を参照してください。

## セットアップと構成

既存の Tomcat アプリケーションまたはその他の JEE コンテナーを使用して Reveal を設定するには、次のことを行う必要があります:

1. JAX-RS 実装に依存関係を追加します。
2. Reveal SDK に依存関係を追加します。
3. Reveal を初期化します。
4. サーバー側エクスポートを有効にします。

手順 1 - JAX-RS 実装に依存関係を追加します。

Jakarta RESTful Web サービス (JAX-RS) 実装に依存関係を追加します。Jersey、RESTEasy、Apache CXF などの複数のオプションから選択できます。ご希望のプロバイダーが説明する手順に従ってください。

例として、Jersey に追加する必要のある依存関係を次に示します:

```
<dependency>
    <groupId>org.glassfish.jersey.containers</groupId>
    <artifactId>jersey-container-servlet</artifactId>
    <version>2.32</version>
</dependency>
<dependency>
    <groupId>org.glassfish.jersey.inject</groupId>
    <artifactId>jersey-cdi2-se</artifactId>
    <version>2.32</version>
</dependency>
```

手順 2 - Reveal SDK に依存関係を追加します。

*reveal-sdk* に依存関係を追加し、SDK のバージョンを指定します。

```
<dependency>
    <groupId>com.infragistics.reveal.sdk</groupId>
    <artifactId>reveal-sdk</artifactId>
    <version>version_number</version>
</dependency>
```

version\_number を **1.0.1821** のような番号に置き換えます。

手順 3 - Reveal を初期化します。

Reveal を初期化するため、**ServletContextListener** クラスを追加します。これを行うには、パッケージ *com.company.analytics.upmedia.reveal* 内にある *upmedia-backend-tomcat* ソースコードからクラス **WebAppListener** をコピーできます。

Reveal を初期化するには、**RevealEngineInitializer.initialize** を使用します。

初期パラメーターなしでメソッドを呼び出すことが可能です。

```
RevealEngineInitializer.initialize();
```

ただし、ほとんどの場合、以下の例のようにパラメーターを使用します。

```
RevealEngineInitializer.initialize(
    new InitializeParameterBuilder()
        .setAuthProvider(new RevealAuthenticationProvider())
        .setUserContextProvider(new RevealUserContextProvider())
        .setDashboardProvider(new RevealDashboardProvider())
        .setDataSourceProvider(new UpMediaDataSourceProvider())
        .setDataProvider(new UpMediaInMemoryDataProvider())
        .setMaxConcurrentImageRenderThreads(2)
```

```
.setLicense("SERIAL_KEY_TO_BE_USED")
.build());
```

これらのパラメーターは Reveal のカスタマイズに使用されるプロバイダーです。Reveal をアプリケーションに統合する場合は、独自のプロバイダーを作成する必要があります。

**RevealEngineInitializer.initialize** に渡される利用可能なパラメーターは次のとおりです:

- *setAuthProvider*。ここで、認証を解決し、IRVAuthenticationProvider を実装するカスタム クラスを含める必要があります。
- *setUserContextProvider*。IRVUserContextProvider を実装するユーザーに関する情報を提供するカスタム クラス。
- *setDashboardProvider*。ダッシュボードを置換または変更するカスタム クラス。IRVDashboardProvider を実装します。
- *setDataSourceProvider*。データソースを置換または変更するカスタム クラス。IRVDataSourceProvider を実装します。
- *setDataProvider*。ダッシュボードのインメモリ データを返すカスタム クラス。IRVDataProvider を実装します。
- *setLicense*。ここでは、シリアルキーを含めて SDK ライセンスを構成できます。

ダッシュボード プロバイダーを実装する方法の詳細については、GitHub の [UpMedia サンプル \(英語\)](#) を参照してください。

#### 手順 4 - サーバー側エクスポートを有効にします。

Java SDK は、ダッシュボードをさまざまな形式 (Image、PDF、PPT、Excel) にエクスポートするためにいくつかのネイティブ コンポーネントを使用します。

これらの形式の 1 つ以上にサーバー側をエクスポートする場合は、[サーバー側のエクスポート構成](#)を参照してください。

# Spring サーバーのセットアップと構成

## 前提条件 (Maven)

Reveal Java SDK は、[Maven \(英語\)](#) モジュールのセットとして配布されます。SDK ライブラリを操作するには、Reveal の Maven リポジトリへの参照と、Maven pom.xml ファイルの依存関係を追加する必要があります。

以下のリポジトリを追加します:

```
<repositories>
  <repository>
    <id>reveal.public</id>
    <url>https://maven.revealbi.io/repository/public</url>
  </repository>
</repositories>
```

以下の依存関係を追加します:

```
<dependency>
  <groupId>com.infragistics.reveal.sdk</groupId>
  <artifactId>reveal-sdk</artifactId>
  <version>version_number</version>
</dependency>
```

version\_number を **0.9.6** のような番号に置き換えます。

Maven についてご不明な点がございましたら、次の[リンク \(英語\)](#) を参照してください。

## セットアップと構成

既存の Spring Boot アプリケーションで Reveal をセットアップするには、次のことを行う必要があります:

1. spring-starter-jersey の実装に依存関係を追加します。
2. Reveal SDK に依存関係を追加します。
3. Reveal を初期化します。
4. サーバー側エクスポートを有効にします。

手順 1 - spring-starter-jersey の実装に依存関係を追加します。

まだ追加されていない場合は、spring-starter-jersey に依存関係を追加します。

```
<dependency>
  <groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-jersey</artifactId>
</dependency>
```

手順 2 - Reveal SDK に依存関係を追加します。

reveal-sdk に依存関係を追加し、SDK のバージョンを指定します。

```
<dependency>
  <groupId>com.infragistics.reveal.sdk</groupId>
  <artifactId>reveal-sdk</artifactId>
  <version>version_number</version>
</dependency>
```

version\_number を **1.0.1821** のような番号に置き換えます。

手順 3 - Reveal を初期化します。

Reveal リソースを使用して Jakarta RESTful Web サービス (JAX-RS) アプリケーションを初期化する **JerseyConfig** コンポーネントを追加します。これを行うには、パッケージ *com.pany.analytics.upmedia.reveal* 内にある *upmedia-backend-spring* ソースコードからクラス **RevealJerseyConfig** をコピーできます。

**@ApplicationPath** 注釈は、Reveal サービスを利用するパスを構成します。これを変更する場合は、クライアント側のパスも変更する必要があります。React アプリケーションの場合、これは index.html で構成されます。

```
$.ig.RevealSdkSettings.setBaseUrl("http://localhost:8080/upmedia-backend/reveal-
api/");
```

Reveal を初期化するには、**RevealEngineInitializer.initialize** を使用します。

初期パラメーターなしでメソッドを呼び出すことが可能です。

```
RevealEngineInitializer.initialize();
```

ただし、ほとんどの場合、以下の例のようにパラメーターを使用します。

```
RevealEngineInitializer.initialize(
  new InitializeParameterBuilder()
    .setAuthProvider(new RevealAuthenticationProvider())
    .setUserContextProvider(new RevealUserContextProvider())
    .setDashboardProvider(new RevealDashboardProvider())
    .setDataSourceProvider(new UpMediaDataSourceProvider())
    .setDataProvider(new UpMediaInMemoryDataProvider())
```

```
.setMaxConcurrentImageRenderThreads(2)
.setLicense("SERIAL_KEY_TO_BE_USED")
.build());
```

これらのパラメーターは Reveal のカスタマイズに使用される**プロバイダー**です。Reveal をアプリケーションに統合する場合は、独自のプロバイダーを作成する必要があります。

**RevealEngineInitializer.initialize** に渡される利用可能なパラメーターは次のとおりです:

- *setAuthProvider*。ここで、認証を解決し、IRVAuthenticationProvider を実装するカスタム クラスを含める必要があります。
- *setUserContextProvider*。IRVUserContextProvider を実装するユーザーに関する情報を提供するカスタム クラス。
- *setDashboardProvider*。ダッシュボードを置換または変更するカスタム クラス。  
IRVDashboardProvider を実装します。
- *setDataSourceProvider*。データソースを置換または変更するカスタム クラス。IRVDataSourceProvider を実装します。
- *setDataProvider*。ダッシュボードのインメモリ データを返すカスタム クラス。IRVDataProvider を実装します。
- *setLicense*。ここでは、シリアルキーを含めて SDK ライセンスを構成できます。

ダッシュボード プロバイダーを実装する方法の詳細については、GitHub の [UpMedia サンプル \(英語\)](#) を参照してください。

#### 手順 4 - サーバー側エクスポートを有効にします。

Java SDK は、ダッシュボードをさまざまな形式 (Image、PDF、PPT、Excel) にエクスポートするためにいくつかのネイティブ コンポーネントを使用します。

これらの形式の 1 つ以上にサーバー側をエクスポートする場合は、[サーバー側のエクスポート構成](#)を参照してください。

# Oracle サーバーのセットアップと構成

## 前提条件 (Maven)

Reveal Java SDK は、[Maven](#) モジュールのセットとして配布されます。SDK ライブラリを操作するには、Reveal の Maven リポジトリへの参照と、Maven pom.xml ファイルの依存関係を追加する必要があります。

以下のリポジトリを追加します:

```
<repositories>
  <repository>
    <id>reveal.public</id>
    <url>http://revealpackages.eastus.cloudapp.azure.com/repository/public</url>
  </repository>
  <repository>
    <id>jeecg</id>
    <url>http://maven.jeecg.org/nexus/content/repositories/jeecg/</url>
  </repository>
</repositories>
```

そして、次の依存関係を追加します:

```
<dependencies>
  <dependency>
    <groupId>com.infragistics.reveal.sdk</groupId>
    <artifactId>reveal-sdk</artifactId>
    <version>version_number</version>
  </dependency>
  <dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc14</artifactId>
    <version>10.2.0.5.0</version>
  </dependency>
</dependencies>
```

version\_number を **0.9.6** のような番号に置き換えます。

Maven についてご不明な点がございましたら、次の[リンク](#)を参照してください。

## セットアップと構成

Reveal を既存のアプリケーションと統合するには、次の手順に従う必要があります:

1. 既存のアプリの実装に依存関係を追加します。
2. Reveal SDK に依存関係を追加します。
3. Reveal を初期化します。

4. サーバー側エクスポートを有効にします。

手順 1 – アプリケーションに依存関係を追加します。

Oracle サーバーに必要な手順に従って、既存のアプリケーションに依存関係を追加します。

手順 2 - Reveal SDK に依存関係を追加します。

*reveal-sdk* に依存関係を追加し、SDK のバージョンを指定します。

```
<dependency>
    <groupId>com.infragistics.reveal.sdk</groupId>
    <artifactId>reveal-sdk</artifactId>
    <version>version_number</version>
</dependency>
```

version\_number を **1.0.1821** のような番号に置き換えます。

手順 3 - Reveal を初期化します。

Reveal を初期化するには、**RevealEngineInitializer.initialize** を使用します。

初期パラメーターなしでメソッドを呼び出すことが可能です。

```
RevealEngineInitializer.initialize();
```

ただし、ほとんどの場合、以下の例のようにパラメーターを使用します。

```
RevealEngineInitializer.initialize(
    new InitializeParameterBuilder()
        .setAuthProvider(new RevealAuthenticationProvider())
        .setUserContextProvider(new RevealUserContextProvider())
        .setDashboardProvider(new RevealDashboardProvider())
        .setDataSourceProvider(new UpMediaDataSourceProvider())
        .setDataProvider(new UpMediaInMemoryDataProvider())
        .setMaxConcurrentImageRenderThreads(2)
        .setLicense("SERIAL_KEY_TO_BE_USED")
    .build());
```

これらのパラメーターは Reveal のカスタマイズに使用される**プロバイダー**です。Reveal をアプリケーションに統合する場合は、独自のプロバイダーを作成する必要があります。

**RevealEngineInitializer.initialize** に渡される利用可能なパラメーターは次のとおりです:

- *setAuthProvider*。ここで、認証を解決し、IRVAuthenticationProvider を実装するカスタム クラスを含める必要があります。

- *setUserContextProvider*。IRVUserContextProvider を実装するユーザーに関する情報を提供するカスタム クラス。
- *setDashboardProvider*。ダッシュボードを置換または変更するカスタム クラス。IRVDashboardProvider を実装します。
- *setDataSourceProvider*。データソースを置換または変更するカスタム クラス。IRVDataSourceProvider を実装します。
- *setDataProvider*。ダッシュボードのインメモリ データを返すカスタム クラス。IRVDataProvider を実装します。
- *setLicense*。ここでは、シリアルキーを含めて SDK ライセンスを構成できます。

ダッシュボード プロバイダーを実装する方法の詳細については、GitHub の [UpMedia サンプル \(英語\)](#) を参照してください。

#### Step 4 - 手順 4 - サーバー側エクスポートを有効にします。

Java SDK は、ダッシュボードをさまざまな形式 (Image、PDF、PPT、Excel) にエクスポートするためにいくつかのネイティブ コンポーネントを使用します。

これらの形式の 1 つ以上にサーバー側をエクスポートする場合は、[サーバー側のエクスポート構成](#)を参照してください。

### セットアップと構成 (クライアント)

以下は、Reveal Web Client SDK を設定するための手順です。

1. [依存関係を確認します](#).
2. [Web Client SDK を参照します](#).
3. [Web Client SDK をインスタンス化します](#).

#### 1. 依存関係の確認

Reveal Web Client SDK には、次のサードパーティの参照があります:

- [jQuery](#) 2.2 またはそれ以上
- [Day.js](#) 1.8.15 またはそれ以上
- [Quill RTE](#) 1.3.6 またはそれ以上
- [Marker Clusterer](#) v3 またはそれ以上
- [Google マップ](#) v3 またはそれ以上

#### 2. Web Client SDK の参照

Web ページで **\$ig.RevealView** コンポーネントを有効にするには、いくつかのスクリプトを含める必要があります。これらのスクリプトは Reveal Web Client SDK の一部として提供されます。

```
<script src="~/Reveal/infragistics.reveal.js"></script>
```

JavaScript ファイルは "<InstallationDirectory>\SDK\Web\JS\Client" にあります。

### 3. Web Client SDK のインスタンス化

ダッシュボードのプレゼンテーションは、Web Client SDK を介してネイティブに処理されます。

以下の手順に従って作業を開始します:

1. "id" を指定して `<div />` 要素を定義し、`$.ig.RevealView` コンストラクターを呼び出します。

[!NOTE] サーバー側とクライアント側のパートを個別にホストする 別々のサーバーでクライアント側とサーバー側のパートをホストする場合は、次の手順を続行する前に[こちら](#)を参照してください。

2. `dashboardId` と成功およびエラー ハンドラーを指定して `$.ig.RVDashboard.loadDashboard` を呼び出します。
3. 成功ハンドラーで、ダッシュボードが描画される DOM 要素のセレクターを渡すことにより、`$.ig.RevealView` コンポーネントをインスタンス化します。最後に、取得したダッシュボードを使用し、`$.ig.RevealView` のダッシュボード プロパティに設定する必要があります

#### サンプルコード

```
<!DOCTYPE html>
<html>
  <head>
    :
    <script type="text/javascript">
      var dashboardId = "dashboardId";

      $.ig.RVDashboard.loadDashboard(
        dashboardId,
        function (dashboard) {
          var revealView = new $.ig.RevealView("#revealView");
          revealView.dashboard = dashboard;
        },
        function (error) {
          //Process any error that might occur here
        }
      );
    </script>
  </head>
  <body>
    <div id="revealView" style="height:500px;" />
  </body>
</html>
```

# サーバー側エクスポートの構成

---

Java SDK は、ダッシュボードをさまざまな形式 (Image、PDF、PPT、Excel) にエクスポートするためにいくつかのネイティブコンポーネントを使用します。

- **画像のエクスポート**には [Playwright for Java \(英語\)](#) を使用します。
- **PDF、PPT、および Excel ドキュメントのエクスポート**には、ExportTool (ネイティブアプリケーション) を使用します。

## エクスポートの準備

ダッシュボードを初めて開くと、**Playwright と ExportTool の両方が必要なダウンロードを自動的にトリガ一します**。ただし、プラットフォームによっては、事前にインストールする必要がある依存関係があり、サーバー環境によって外部ダウンロードが制限される場合があり、これらのツールを手動でセットアップする必要があります。

## Playwright の構成

Playwright は必要なバイナリを自動的にダウンロードします。ただし、手動による構成が必要な場合や、その構成 (または調整) の詳細については、[Playwright のドキュメント \(英語\)](#)を参照してください。

## macOS の依存関係

macOS に必要なライブラリは **libgdiplus** のみです。インストール情報は[こちら](#)を参照してください。

## Linux の依存関係

Linux には複数のネイティブライブラリへの依存関係があります。インストールする必要がある依存関係の正確なリストは、使用するディストリビューション、バージョン、および以前にインストールしたパッケージのリストによって異なります。

以下は、基本的な Ubuntu 18.0.4 ディストリビューションに必要なライブラリのリストです。

```
sudo apt-get update

sudo apt-get install -y libgdiplus\
    libatk1.0-0\
    libatk-bridge2.0-0\
    libxkbcommon0\
    libcomposite1\
    libxdamage1\
    libxfixes3\
    libxrandr2\
    libgbm1\
    libgtk-3-0\
    libpango-1.0-0\
    libcairo2\
    libgdk-pixbuf2.0-0\
```

```
libatspi2.0-0  
sudo apt-get install -y --no-install-recommends xvfb
```

必要に応じて、ログ ファイルに含まれるエラーから、見つからないライブラリに関する詳細情報を取得できます。

その他の環境では、以下もインストールする必要があります。

```
sudo apt-get install -y --allow-unauthenticated libc6-dev  
sudo apt-get install -y --allow-unauthenticated libx11-dev
```

## ExportTool の手動設定

以下の手順は、以下のシナリオでのみ必要です。

- 自動ダウンロード メカニズムに問題がある場合
- すべてを事前にインストールしておく必要がある場合

### 手順

1. ご使用のプラットフォームに必要なバイナリ ([Windows](#)、[Linux](#)、または [macOS](#)) をダウンロードします。
2. Web アプリケーションが実行されているサーバーのディレクトリにファイルを解凍します (ユーザーはそのディレクトリにアクセスできる必要があります)。
3. zip ファイルを抽出した後、**ExportTool** を以下の場所で取得できます:  
`<dir>/<version>/<arch>/ExportTool`。例:

```
<dir>/1.0.0/linux-x64/ExportTool.
```

4. Reveal の初期化時に、zip ファイルを抽出したディレクトリを設定します。以下のコード スニペットに似ています。

```
String exportToolDir = "<dir>";  
RevealEngineInitializer.initialize(  
    new InitializeParameterBuilder()  
        .setAuthProvider(new UpmediaAuthenticationProvider())  
        .setUserContextProvider(new UpmediaUserContextProvider())  
        .setDashboardProvider(new UpmediaDashboardProvider())  
        .setLicense("SERIAL_KEY_TO_BE_USED")  
        .setExportToolContainerPath(exportToolDir)  
        .build());
```

または、以下に示すように **reveal.exportToolContainerPath** システム プロパティでディレクトリを指定できます。

```
java -Dreveal.exportToolContainerPath=<dir> -jar target/upmedia-backend-spring.war
```

# ダッシュボード ファイルの読み込み

## 概要

SDK でダッシュボードを開く/保存するには 2 つの方法があります。

- **サーバー側:** Reveal のクライアント側コンポーネントは、サーバー側コンポーネントを使用してダッシュボードの定義を取得し、定義された各視覚化とフィルターのデータも取得します。

これが最も簡単な方法であり、最初に SDK を評価するときに推奨される方法です。

- **クライアント側:** 完全な制御と高い柔軟性が提供されます。カスタム サーバーからコンテンツを取得しながら、クライアントページでダッシュボードのコンテンツをストリームに提供します。

この方法を使用すると、たとえば、ユーザーのアクセス許可を確認したり、カスタムなユーザーインターフェイスを表示してダッシュボードを選択したり、ユーザーが使用する .rdash ファイルをアップロードしたりすることができます。クライアント側アプローチの詳細については、[セットアップと構成 \(クライアント\)](#) を参照してください。

## サーバー側アプローチ

まず、クライアント側からダッシュボード ID とダッシュボードを要求しているユーザーの ID の両方を取得します。次に、サーバーで、ダッシュボードの定義を取得し、定義された各視覚化とフィルターのデータを取得します。

ダッシュボードの保存方法 (ファイルシステム、データベースなど) とどのユーザーに対してダッシュボードへのアクセスを許可するか、SDK の一部ではないため、ご自身で処理する必要があります。

インターフェイス IRVDashboardProvider を実装する独自のダッシュボード プロバイダーが必要です:

```
public interface IRVDashboardProvider {  
    InputStream getDashboard(String userId, String dashboardId) throws  
    IOException;  
    void saveDashboard(String userId, String dashboardId, InputStream  
    dashboardStream) throws IOException;  
}
```

ダッシュボードの保存を許可しない場合は、*saveDashboard* の実装を空のままにしておくことができます。

*getDashboard* メソッドはユーザー ID とダッシュボード ID を受け取り、それらを保存するために選択したダッシュボード (ファイルシステム、データベースなど) を見つける必要があります。最後に、ダッシュボードの内容を含む *InputStream* を .rdash 形式 (基本的には JSON ドキュメントを含む ZIP ファイル) で返すことが期待されています。

詳細については、[GitHub](#) の UpMedia サンプルの実装 (upmedia、upmedia-backend-tomcat、upmedia-backendspring の *UpmediaDashboardProvider*) を参照してください。

# データ ソースへ資格情報を提供

## 概要

Server SDK では、データ ソースにアクセスするときに使用される一連の資格情報を渡すことができます。

## コード

最初の手順は、以下に示すように、**IRVAuthenticationProvider** を実装し、それを **RevealSdkContextBase** の **AuthenticationProvider** プロパティとして返します。

```
public class EmbedAuthenticationProvider : IRVAuthenticationProvider
{
    public Task<IRVDataSourceCredential> ResolveCredentialsAsync(string userId,
RVDashboardDataSource dataSource)
    {
        IrvDataSourceCredential userCredential = null;
        if (dataSource is RVPostgresDataSource)
        {
            userCredential = new
RVUsernamePasswordDataSourceCredential("postgresuser", "password");
        }
        else if (dataSource is RVSqlServerDataSource)
            // 「domain」パラメーターは必ずしも必要ではなく、これは SQL Server の構成によって異なりま
す。
        {
            userCredential = new
RVUsernamePasswordDataSourceCredential("sqlserveruser", "password", "domain");
        }
        else if (dataSource is RVGoogleDriveDataSource)
        {
            userCredential = new
RVBearerTokenDataSourceCredential("fhJhbUci0mJSUzi1nIiSint....",
"user@company.com");
        }
        else if (dataSource is RVRESTDataSource)
        {
            userCredential = new RVUsernamePasswordDataSourceCredential(); // 
Anonymous
        }
        return Task.FromResult<IRVDataSourceCredential>(userCredential);
    }
}
```

## 実装するクラスの選択

使用できるクラスは 2 つあり、どちらも **IRVDataSourceCredential** インターフェイスを実装します。以下に詳述するように、データ ソースに応じてクラスを選択する必要があります。

- クラス **RVBearerTokenDataSourceCredential** は以下で動作します。
  - アナリティクス ツール (Google アナリティクス)。
  - コンテンツ マネージャーとクラウド サービス (Box、Dropbox、Google Drive、OneDrive、SharePoint Online)。
- クラス **RVUsernamePasswordDataSourceCredential** は以下と動作します。
  - カスタマー リレーションシップ マネージャー (Microsoft Dynamics CRM オンプレミスおよびオンライン)。
  - データベース (Microsoft SQL Server、Microsoft Analysis Services サーバー、MySQL、PostgreSQL、Oracle、Sybase)。
- **両クラス**は以下と動作します。
  - その他のデータ ソース (OData フィード、Web Resources、REST API)。

## 認証なし

認証なしで匿名のリソースで作業することができます。この場合、空のコンストラクタを持つ **RVUsernamePasswordDataSourceCredential** を使用できます。これは、そのクラスで機能するすべてのデータ ソースに対して実行できます。

上記のサンプルで使用したコード スニペット:

```
else if (dataSource is RVRESTDataSource)
{
    userCredential = new RVUsernamePasswordDataSourceCredential();
}
```

## Java SDK 再頒布物 (OSS, サードパーティ ソフトウェアと Reveal バイナリ)

サーバー/クライアント	アセンブリ/使用されたコード	ライセンス
サーバー側 SDK	reveal-sdk-<sdk_version_number>.jar	<a href="#">Reveal ライセンス</a>
サーバー側 SDK	reveal-rest-service-<sdk_version_number>.jar	<a href="#">Reveal ライセンス</a>
サーバー側 SDK	reveal-engine-service-api-<sdk_version_number>.jar	<a href="#">Reveal ライセンス</a>
サーバー側 SDK	reveal-sdk-api-<sdk_version_number>.jar	<a href="#">Reveal ライセンス</a>
サーバー側 SDK	reveal-engine-service-<sdk_version_number>.jar	<a href="#">Reveal ライセンス</a>
サーバー側 SDK	connectors-<sdk_version_number>.jar	<a href="#">Reveal ライセンス</a>
サーバー側 SDK	datalayer-<sdk_version_number>.jar	<a href="#">Reveal ライセンス</a>
サーバー側 SDK	core-<sdk_version_number>.jar	<a href="#">Reveal ライセンス</a>

サーバー側 SDK	requests-< sdk_version_number >.jar	Reveal ライセンス
サーバー側 SDK	ExportTool(.exe)	Reveal ライセンス
クライアント側 SDK	infragistics.reveal.js	Reveal ライセンス
クライアント側 SDK	infragistics.langpack.<country_name>.js	Reveal ライセンス
クライアント側 SDK	reveal-webComponent.js	Reveal ライセンス
サーバー側 SDK	driver-1.10.0.jar, driver-bundle-1.10.0.jar	Apache 2.0
サーバー側 SDK	Java-WebSocket-1.5.1.jar	MIT
サーバー側 SDK	kxml2-2.3.0.jar	MIT
サーバー側 SDK	ojdbc14-10.2.0.5.0.jar	OTN
サーバー側 SDK	postgresql-9.4.1212.jre6	PostgreSQL ライセンス
サーバー側 SDK	okhttp-3.8.0.jar	Apache 2.0
サーバー側 SDK	jtds-1.3.1.jar	LGPL-2.1
サーバー側 SDK	gson-2.8.6.jar	Apache 2.0
サーバー側 SDK	jxl-2.6.12.jar	LGPL-2.0
サーバー側 SDK	antlr4-runtime-4.7.jar	BSD 3-Clause
サーバー側 SDK	joda-time-2.10	Apache 2.0
サーバー側 SDK	jackson-core-2.9.6.jar	Apache 2.0
サーバー側 SDK	commons-text-1.1.jar	Apache 2.0
サーバー側 SDK	Utility class from Spring	Apache 2.0
サーバー側 SDK	jcifs-1.3.17.jar	LGPL-2.1
サーバー側 SDK	json-20171018.jar	Custom
サーバー側 SDK	log4j-1.2.14	Apache 2.0
サーバー側 SDK	mariadb-java-client-1.5.9.jar	LGPL-2.1
サーバー側 SDK	okio-1.13.0.jar	Apache 2.0
サーバー側 SDK	commons-lang3-3.5	Apache 2.0
サーバー側 SDK	snowflake JDBC driver	Apache 2.0
クライアント側 SDK	Bootstrap	MIT
クライアント側 SDK	jQuery	MIT
クライアント側 SDK	jQuery validation	MIT
クライアント側 SDK	jQuery validation unobtrusive	Apache 2.0

クライアント側 SDK	<a href="#">Roboto Google Font</a>	Apache 2.0
クライアント側 SDK	<a href="#">Google Maps Marker Clusterer</a>	Apache 2.0
クライアント側 SDK	<a href="#">Google Maps Platform</a>	カスタム
クライアント側 SDK	<a href="#">Babel JS Compiler</a>	MIT
クライアント側 SDK	<a href="#">LitElement</a>	BSD 3-Clause
クライアント側 SDK	<a href="#">Simple JavaScript Inheritance</a>	MIT
クライアント側 SDK	<a href="#">Day.js</a>	MIT
クライアント側 SDK	<a href="#">Quill Rich Text Editor</a>	BSD 3-Clause
クライアント側 SDK	<a href="#">snowflake connector .NET</a>	Apache 2.0

## JAVA API リファレンス

ここでは、Reveal SDK、特に JAVA サーバー API に関する詳しい情報を見つけることができます。完全なリファレンスについては、[このリンク](#)をご覧ください。

最も一般的に使用されるクラスとインターフェース

### SDK の主な概念と機能:

[RevealEngineInitializer](#)

[InitializeParameter](#)

[InitializeParameterBuilder](#)

### データ ソース

[IRVDashboardProvider](#)

[IRVDataSourceProvider](#)

[RVSqIIServerDataSource](#)

[RVSqIIServerDataSourceItem](#)

[RVExcelDataSource](#)

[RVExcelDataSource](#)

[RVRESTDataSource](#)

[RVRESTDataSourceItem](#)

[RVJsonDataSource](#)

[RVJsonDataSourceItem](#)

[RVInMemoryDataSource](#)

[RVInMemoryDataSourceItem](#)

### 認証

[IRVAuthenticationProvider](#)

[IRVDataSourceCredential](#)

[RVUsernamePasswordDataSourceCredential](#)

[IRVUserContextProvider](#)

# Desktop SDK の使用

---

Reveal を別のアプリケーション (Windows WPF、Windows フォーム、iOS、または Android) に埋め込むと、Reveal SDK はアプリに統合されたライブラリまたはフレームワークとして提供されます (統合手順はプラットフォームごとに異なります)。

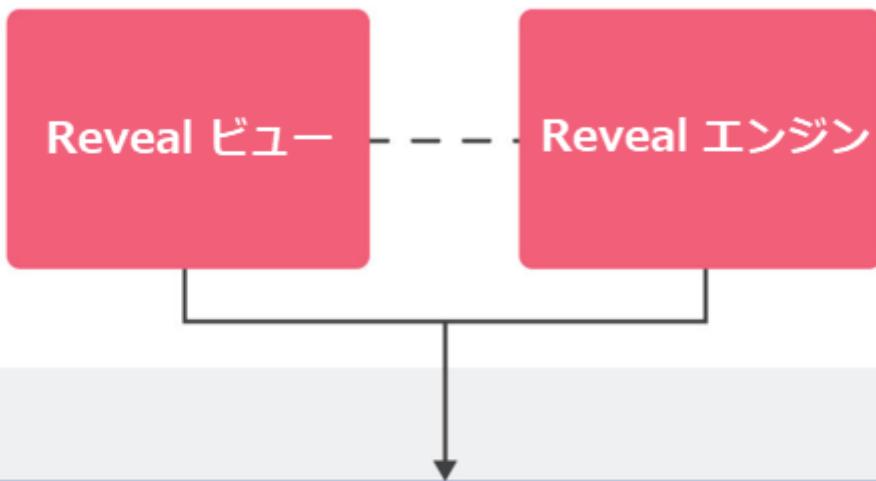
## RevealView コンポーネント

内包アプリは、レンダリングするダッシュボードで構成された RevealView オブジェクトを作成します。このビューは内包アプリに表示され、一連のコールバックを使用してダッシュボードのレンダリング方法と使用されるデータをカスタマイズできます。

RevealView コンポーネントはレンダリングおよびデータ変換機能を Reveal エンジンを通じて自動的に提供しますが、ダッシュボードまたは資格情報の保管は処理しません。ダッシュボードの定義を保持するバイナリコンテンツ (.rdash ファイル) は、内包アプリケーションによって提供される必要があります。これにより、コンテナ アプリケーションは、ダッシュボードの使用方法やエンドユーザーによる共有方法を処理できます (たとえば、ダッシュボードは内部サーバーからダウンロードしたり、アプリケーションのバイナリにリソースとしてまとめたり、ファイルシステムに保存したりできます)。

# クライアント アプリケーション

## Reveal SDK



### アプリケーション コールバック:

- ・ダッシュボードストリームの取得
- ・データソース資格情報の取得
- ・ダッシュボードデータソースの置き換え
- ・インメモリデータの設定
- ・最大化されたウィジェットのためのイベント
- ・クリックされたデータポイントのためのイベント
- ・その他

## 資格情報の詳細

データベース (または認証を必要とするその他のデータソース) からデータを取得する場合、通常、内包アプリは既にこれらの資格情報を構成ファイルからロードするか、安全な記憶域に保存することによって処理します。これは、Reveal がこれらの資格情報の保管と処理をコンテナに委任するためです。必要に応じて、アプリは内部の認証情報を返すか、ユーザーに入力を求めるかを決定します。

# セットアップと構成 (Desktop)

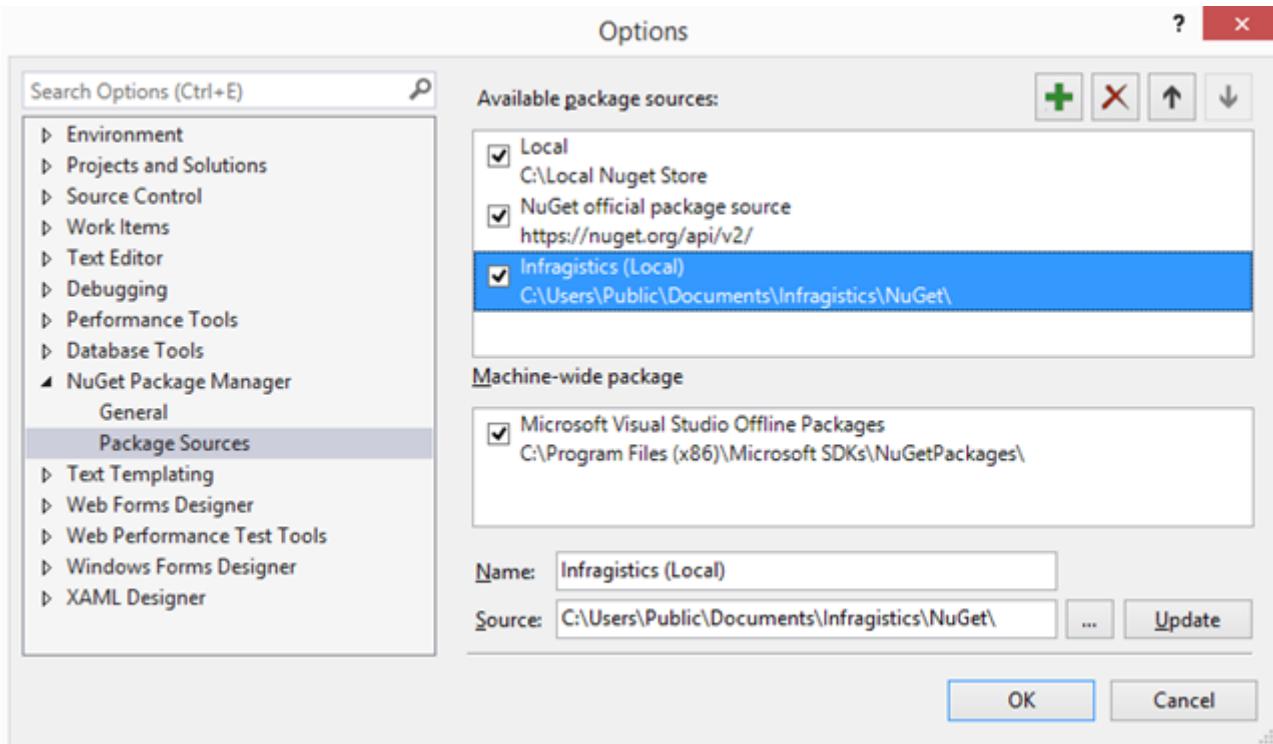
Reveal Desktop SDK の設定は、次のいずれかを選択できます。

- NuGet パッケージマネージャの使用
- プロジェクトを手動でセットアップ

## NuGet の使用 (推奨)

WPF または WinForms アプリケーションプロジェクトのセットアップには、**Reveal.Sdk.Wpf** パッケージをインストールする方法が最も簡単です。

Reveal SDK をインストールすると、%public%\Documents\Infragistics\NuGet を指す Infragistics (Local) と呼ばれる新しい NuGet パッケージソースが nuget.config に追加されます。



Infragistics (Local) フィードがインストーラーによって正しく設定されていることを確認後、**Reveal.Sdk.Wpf** パッケージをアプリケーションプロジェクトにインストールできます。

NuGet パッケージをインストールすると、以下の依存関係パッケージもインストールされます。

- CefSharp.Wpf (87.1.132 以降)
- SkiaSharp (1.68.0 以降)
- System.Data.SQLite.Core (1.0.108 以降)

[!NOTE] Microsoft SQL Server データを視覚化できるようにするには、プロジェクトに Microsoft.Data.SqlClient (1.1.3) パッケージを手動でインストールする必要があります。

## 手動設定の使用

プロジェクトを手動で設定するには、以下の手順に従ってください。

- インストーラーによってドロップされたアセンブリへの参照を <InstallationDirectory>\SDK\WPF\Binaries に追加します。
- RevealView コントロールが依存する以下の NuGet パッケージをインストールします。
  - CefSharp.Wpf (87.1.132 以降)
  - SkiaSharp (1.68.0 以降)
  - System.Data.SQLite.Core (1.0.108 以降)
  - Microsoft.Data.SqlClient (1.1.3 以降)

以下は、**CefSharp.Wpf** の既知の問題を処理する方法です。

## CefSharp 依存パッケージの処理

CefSharp 依存関係パッケージのインストールすると、ビルド (AnyCPU をターゲットにしている場合) が失敗する問題。

[!NOTE] **エラーの詳細:** エラーの説明は次のようになります。エラー: platform(x86/x64)を指定した場合、CefSharp.Common はそのまま使用できます。AnyCPU サポートの場合は、この[リンク](#)を参照してください。

このエラーを修正するためには、エラーの URL で説明されているように、CefSharpAnyCpuSupport プロパティをプロジェクト ファイルに追加する必要があります。アプリケーションの **.csproj** ファイルに以下のプロパティ グループを追加するのみです。

```
<PropertyGroup>
  <CefSharpAnyCpuSupport>true</CefSharpAnyCpuSupport>
</PropertyGroup>
```

エラーの修正については以上です。CefSharp の GitHub Issues で示されている の手順を適用する必要はありません。Reveal コンポーネントは必要に応じて CefBrowser クラスを初期化します。

以上でプロジェクトで Reveal ダッシュボードを表示する設定が完了しました。

# 新しい可視化とダッシュボードの作成 (Desktop)

## 概要

既存のダッシュボードに新しい可視化を追加するには、ユーザーが使用するデータ ソースを選択する必要があります。そのためには、含まれているアプリケーションが SDK に情報を提供する必要があるので、新しい可視化に使用できるデータ ソースのリストを表示できます。

## データ ソースのリストを表示

データ ソースのリストを表示するために使用する必要があるコールバックは、**DataSourcesRequested** です。このコールバックに独自のメソッドを設定しない場合、新しい可視化が作成されると、Reveal はダッシュボードで使用されているデータ ソースがある場合はすべて表示します。

コード:

以下のコードは、インメモリ項目と SQL Server データ ソースを表示するようにデータ ソース選択画面を構成する方法を示しています。

```
private void RevealView_DataSourcesRequested(object sender,
DataSourcesRequestedEventArgs e)
{
    var inMemoryDSI = new RVInMemoryDataSourceItem("employees");
    inMemoryDSI.Title = "Employees";
    inMemoryDSI.Description = "Employees";

    var sqlDs = new RVSqlServerDataSource();
    sqlDs.Title = "Clients";
    sqlDs.Id = "SqlDataSource1";
    sqlDs.Host = "db.mycompany.local";
    sqlDs.Port = 1433;
    sqlDs.Database = "Invoices";

    e.Callback(new RevealDataSources(
        new List<RVDashboardDataSource> { sqlDs },
        new List<RVDataSourceItem> { inMemoryDSI },
        false));
}
```

上記のコードでは、DataSourcesRequested イベントを処理するために次のメソッドをアタッチしたと想定しています。

```
revealView.DataSourcesRequested += RevealView_DataSourcesRequested
```

3 番目のパラメータの `false` 値は、ダッシュボード上の既存のデータ ソースが表示されないようにします。そのため、[+] ボタンを使用して新しいウィジェットを作成すると、以下の画面が表示されます。



RVInMemoryDataSourceItem コンストラクタに渡される employees パラメーターは [インメモリデータのサポート](#) で使用されているコンストラクタと同じデータセット ID で、返されるデータセットを識別します。

## 新しいダッシュボードの作成

以下の手順でダッシュボードを簡単に作成できます。 **RevealView** を初期化するだけです。通常、ユーザーにダッシュボードを最初から作成する機能を提供する場合は、空のダッシュボードを編集モードで直接開いて、ユーザーがすぐに編集を開始できるようにします。

```
revealView = new RevealView();
revealView.StartEditMode = true;
revealView.DataSourcesRequested += RevealView_DataSourcesRequested;

revealView.Dashboard = new RVDashboard();
```

SDK とともに配布されている UpMedia WPF アプリケーションに、*EmptyDashboard.xaml.cs* の実用的な例があります。

# ダッシュボードの読み込み

---

アプリケーションに埋め込まれた [RevealView](#) コントロールに既存の Reveal ダッシュボードを表示する場合は、4 つのオプションから選択できます。

- ファイルパスからダッシュボードを読み込み
- ファイルストリームからダッシュボードを読み込み
- 埋め込まれたリソースからダッシュボードを読み込み
- json からダッシュボードを読み込み

ダッシュボードを [RevealView](#) に読み込むには、**.rdash** ファイル (.rdash は Reveal によって作成されたダッシュボードのファイル拡張子) を取得し、それを [RVDashboard](#) オブジェクトとして逆シリアル化し、[RevealView.Dashboard](#) プロパティを RVDashboard オブジェクトインスタンスに割り当てます。

**.rdash** ダッシュボード ファイルは次の方法で作成できます:

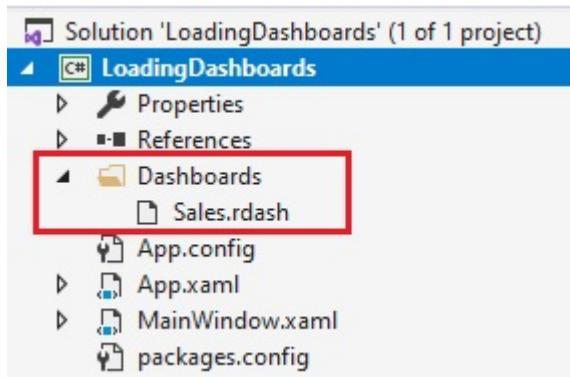
- ダッシュボードを [Reveal BI Web サイト](#) から .rdash ファイルとしてエクスポートします。
- ダッシュボードをネイティブの Reveal アプリケーションの 1 つから .rdash ファイルとしてエクスポートします。
- Reveal SDK を使用してアプリケーションで作成されたダッシュボードを保存またはエクスポートします。

詳しくは [SDK 用のダッシュボードの取得](#)をご覧ください。

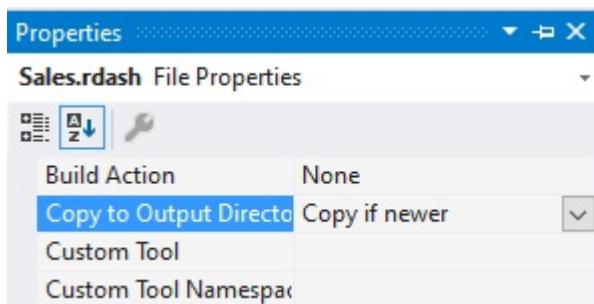
# ファイルパスから読み込み

ダッシュボード ファイルをアプリケーションと一緒に出荷することは非常に一般的です。これらのファイルは通常、アプリケーションの実行中にディスクからファイルを読み込みできるように、既知のディレクトリのクライアントディスク ドライブにコピーされます。ファイルパスを使用してこれらのダッシュボードを読み込むには、**.rdash** ファイルへのファイルパスを知っておく必要があります。

この例では、**[Dashboards]** と呼ばれる Visual Studio ソリューションにディレクトリを作成しました。このディレクトリには、アプリケーションのすべての .rdash ファイルが含まれます。



各 .rdash ファイルのプロパティで、**[Copy to Output Directory]** の値を **[Copy if Newer]** に設定していることを確認することが重要です。これにより、プロジェクトのビルド時にダッシュボード ファイルがディスクにコピーされます。



最初の手順は、四方込みしたい .rdash ファイルのファイルの場所を取得することです。ダッシュボードへのファイルパスを取得したら、**RVDashboard** の新しいインスタンスを作成し、ファイルパスを **RVDashboard** クラスのコンストラクターに渡します。

この例では、**Environment.CurrentDirectory** を使用して、アプリケーションの現在の実行ディレクトリを取得しています。次に、**Path.Combine** メソッドを使用して、**Dashboards** ディレクトリにある **Sales.rdash** ダッシュボードの場所を追加します。**Sales.rdash** ダッシュボードへの正しいファイルパスを得したら、コンストラクター引数としてファイルパスを使用して、**RevealView.Dashboard** プロパティを **RVDashboard** オブジェクトの新しいインスタンスに設定します。

```
var filePath = Path.Combine(Environment.CurrentDirectory,
    "Dashboards/Sales.rdash");
_revealView.Dashboard = new RVDashboard(filePath);
```

`RVDashboard.LoadDashboardAsync` メソッドを使用して、ダッシュボードをファイルパスから `RevealView` に非同期で読み込むこともできます。

```
var filePath = Path.Combine(Environment.CurrentDirectory,  
    "Dashboards/Sales.rdash");  
_revealView.Dashboard = await RVDashboard.LoadDashboardAsync(filePath);
```

[!NOTE] このサンプルのソース コードは [GitHub](#) にあります。

# ファイルストリームから読み込み

ファイルストリームからの Reveal ダッシュボードの読み込みは、ファイルパスからのダッシュボードの読み込みと非常によく似ています。この場合、ダッシュボード ファイルのファイルパスを取得したら、`RVDashboard` オブジェクトインスタンスを作成する前に、それを `FileStream` に読み込みます。

この例では、`File.OpenRead` メソッドを使用して、`Sales.rdash` ファイルをファイルストリームに読み込んでいます。次に、コンストラクター引数としてファイルストリームを渡すことにより、新しい `RVDashboard` オブジェクトを作成し、新しく作成された `RVDashboard` インスタンスを `RevealView.Dashboard` プロパティに割り当てます。

```
var filePath = Path.Combine(Environment.CurrentDirectory,
    "Dashboards/Sales.rdash");
using (var stream = File.OpenRead(filePath))
{
    _revealView.Dashboard = new RVDashboard(stream);
}
```

`RVDashboard.LoadDashboardAsync` メソッドを使用して、ダッシュボードをファイルストリームから `RevealView` に非同期で読み込むこともできます。

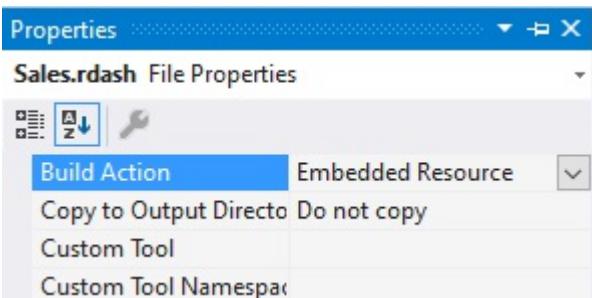
```
var filePath = Path.Combine(Environment.CurrentDirectory,
    "Dashboards/Sales.rdash");
using (var stream = File.OpenRead(filePath))
{
    _revealView.Dashboard = await RVDashboard.LoadDashboardAsync(stream);
}
```

[!NOTE] このサンプルのソース コードは [GitHub](#) にあります。

# リソースから読み込み

アプリケーションでファイルを配布するためのもう 1 つのオプションは、ファイルをリソースとしてアプリケーションに埋め込むことです。これにより、クライアントのディスク ドライブにファイルが配置されるのではなく、ファイルがアプリケーションのアセンブリに直接埋め込まれます。

Reveal ダッシュボード .rdash ファイルをリソースとしてアプリケーションに埋め込むには、Visual Studio でダッシュボード ファイルのプロパティを開き、.rdash ファイルの [Build Action] を [EmbeddedResource] に設定します。



ダッシュボードが **EmbeddedResource** として定義されたら、`Assembly.GetManifestResourceStream` メソッドを使用してダッシュボードを読み込むことができます。このメソッドは、`RevealView` に読み込むために使用できる `Stream` オブジェクトを返します。

`Assembly.GetManifestResourceStream` メソッドで指定するリソースの名前には、.rdash ファイルの名前空間とファイル名が含まれている必要があることに注意してください。

この例では、リソースの名前は、アプリケーションのルート名前空間 `LoadingDashboards` と、ダッシュボード ファイルを含むディレクトリである `Dashboards` で始まり、その後に .rdash ファイル `Sales.rdash` の名前が続けます。これにより、`LoadingDashboards.Dashboards.Sales.rdash` の完全なリソース名が得られます。

```
var resource =
Assembly.GetExecutingAssembly().GetManifestResourceStream($"LoadingDashboards.Dashboards.Sales.rdash");
using (resource)
{
    _revealView.Dashboard = new RVDashboard(resource);
}
```

`RVDashboard.LoadDashboardAsync` メソッドを使用して、ダッシュボードをリソース ストリームから `RevealView` に埋め込みリソースとして非同期に読み込むこともできます。

```
var resource =
Assembly.GetExecutingAssembly().GetManifestResourceStream($"LoadingDashboards.Dashboards.Sales.rdash");
using (resource)
{
```

```
_revealView.Dashboard = await RVDashboard.LoadDashboardAsync(resource);  
}
```

[!NOTE] このサンプルのソース コードは [GitHub](#) にあります。

# JSON から読み込み

---

上級ユーザー、または Reveal ダッシュボードを .rdash ファイルではなく .json ファイルにシリアル化するユーザーの場合、`RVDashboard.LoadFromJsonAsync` メソッドを使用してこれらの JSON ベースのファイルを読み込みできます。

最初の手順は、Reveal ダッシュボードを json 文字列にシリアル化することです。文字列を取得したら、JSON をディスクに保存するか、rdata ストアに保存できます。

Reveal ダッシュボードを JSON にシリアル化するには、`RVDashboard.ExportToJson` メソッドを呼び出すだけです。

```
var json = dashboard.ExportToJson();
```

ダッシュボードが JSON 形式にシリアル化されたら、その JSON ファイルをディスクに保存するか、`RevealView` に直接読み込みできます。

ディスクからダッシュボード JSON ファイルを読み込む場合、コードは次のようにになります:

```
var filePath = Path.Combine(Environment.CurrentDirectory,
    "Dashboards/Sales.json");
var json = File.ReadAllText(filePath);
```

JSON 文字列を取得したら、`RVDashboard.LoadFromJsonAsync` メソッドの結果を `RevealView.Dashboard` プロパティに設定して、JSON 文字列をメソッド引数として渡すことでダッシュボードを読み込むことができます。

```
_revealView.Dashboard = await RVDashboard.LoadFromJsonAsync(json);
```

[!WARNING] JSON にシリアル化された後に Reveal ダッシュボードのコンテンツを操作または変更すると、ダッシュボードの完全性が損なわれ、ダッシュボードのコンテンツに取り返しのつかない損傷が生じる可能性があります。これにより、エラーやダッシュボードの読み込みの失敗により、アプリケーションで実行時に例外がスローされる可能性があります

[!NOTE] このサンプルのソース コードは [GitHub](#) にあります。

# ダッシュボードの編集と保存

## 概要

ダッシュボードを読み込むには、ダッシュボード ファイルを含むストリームを **RevealView** コンポーネントに提供する必要があります。その後、ユーザーがダッシュボードを変更した後に、変更されたダッシュボード ファイルを処理したい場合があります。

## ダッシュボードの編集

**RevealView** の **Dashboard** プロパティ (タイプ **RVDashboard**) は、エンドユーザーがダッシュボードの編集を開始すると更新されます。たとえば、可視化またはフィルターを追加または削除すると、**RVDashboard** のコレクションが自動的に更新されます。

**RVDashboard** の **PropertyChanged** イベントに添付して、**HasPendingChanges** などのプロパティへの変更の通知を受け取ります:

### コード サンプル:

```
dashboard.PropertyChanged += Dashboard_PropertyChanged;
```

次に、イベントハンドラーを実装します:

```
private void Dashboard_PropertyChanged(object sender,  
System.ComponentModel.PropertyChangedEventArgs e)  
{  
    if (e.PropertyName == "HasPendingChanges")  
    {  
        Console.Out.WriteLine("HasPendingChanges: " +  
((RVDashboard)sender).HasPendingChanges);  
    }  
}
```

ユーザーが可視化の編集を終了した後、可視化工ディターを閉じると、**Revealview** の **VisualizationEditorClosed** イベントが発生します。次のコードを使用して、イベントに添付することができます:

```
revealView.VisualizationEditorClosed += RevealView_VisualizationEditorClosed;
```

次にイベントハンドラーを実装します:

```
private void RevealView_VisualizationEditorClosed(object sender,  
VisualizationEditorClosedEventArgs e)
```

```

{
    if (e.IsCancelled)
    {
        Console.Out.WriteLine("Visualization editor cancelled " +
(e.IsNewVisualization ? "creating a new visualization" : "editing " +
e.Visualization.Title));
        return;
    }
    if (e.IsNewVisualization)
    {
        Console.Out.WriteLine("New visualization created: " +
e.Visualization.Title);
    } else
    {
        Console.Out.WriteLine("Visualization modified: " + e.Visualization.Title);
    }
}

```

新しい可視化を追加する方法を制御する必要がある場合は、[新しい可視化とダッシュボードの作成](#)を参照してください。

## ダッシュボードの保存

次のコードを使用して **SaveDashboard** イベントにアタッチできます。

```
_revealView.SaveDashboard += RevealView_SaveDashboard;
```

次にイベントハンドラーを実装します。

```

private async void RevealView_SaveDashboard(object sender, DashboardSaveEventArgs
args)
{
    var data = await args.Serialize();
    using (var output = File.Open($"{args.Name}.rdash", FileMode.Create))
    {
        output.Write(data, 0, data.Length);
    }
    args.SaveFinished();
}

```

上記の例は、Save イベントを処理するための単純化された実装を示しています。実際のシナリオでは、カスタム UI をユーザーに表示し、ユーザーがダッシュボードの場所と名前を選択できます。

ユーザーがダッシュボードの名前を変更した場合は、メソッド **SerializeWithNewName** を使用して、ダッシュボードの Title 属性に正しく反映された名前を取得できます。

保存操作を処理したくない場合は、次の設定でダッシュボードを編集するオプションをオフにすることができます。

```
revealView.CanEdit = false;
```

この方法は、たとえば、ユーザーが変更を加えることを想定していない場合に便利です。

# ダッシュボードまたは可視化データをエクスポートする

---

## 概要

ダッシュボードまたは特定の可視化の画像を生成してそれをエクスポートする場合は、以下のオプションがあります。

- **画像**として;
- **PDF** として;
- **PowerPoint** プrezentationとして;
- **Excel** データ形式として;

ダッシュボードまたは視覚化を有効にするには、次のことができます。

- [RevealView](#) のエクスポート設定を使用するか、
- **画像**としてエクスポートする場合は、コードによってエクスポートを開始します [RevealView 以外](#)。

## 前提条件

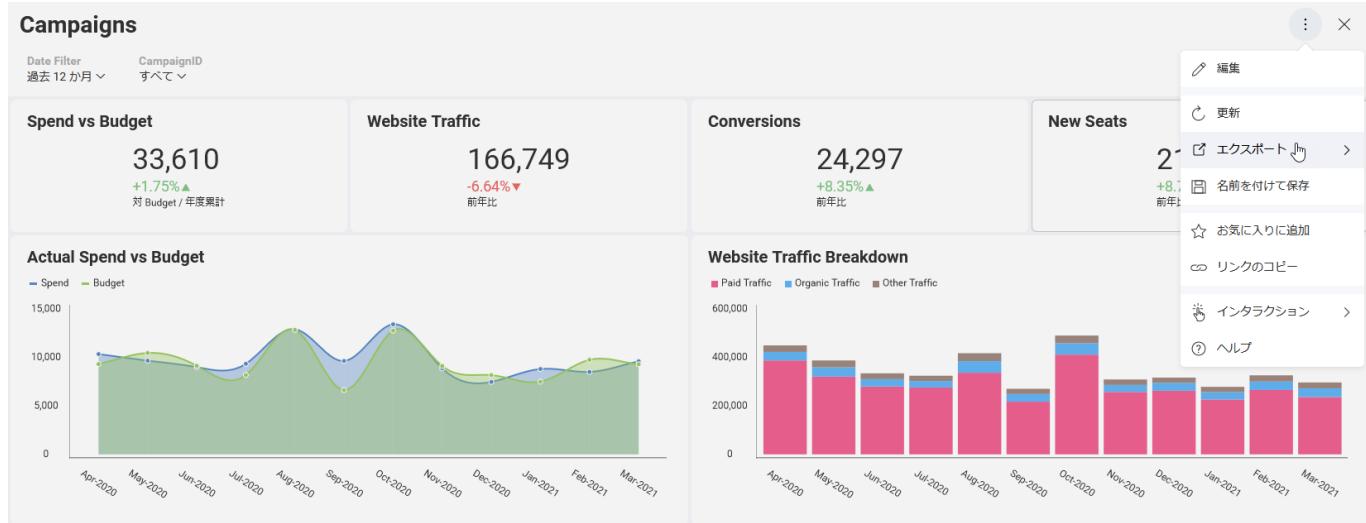
画像へのエクスポート機能を使用するには、[CefSharp.Wpf NuGet package \(>= 83.4.20\)](#) への参照をプロジェクトに追加する必要があります。

## エクスポート設定を有効にする

ユーザーがダッシュボードから画像、ドキュメント、プレゼンテーションを生成できるようにするためにダッシュボードを読み込む際に関連するプロパティをtrueに設定してください。

- **revealView.showExportImage** - **画像**としてエクスポートする場合;
- **RevealView.ShowExportToPDF** - **PDF** ドキュメントとしてエクスポートする場合;
- **RevealView.ShowExportToPowerpoint** - **PowerPoint** プrezentationとしてエクスポートする場合;
- **RevealView.ShowExportToExcel** - **Excel** データ形式でエクスポートする場合;

これにより、ダッシュボードが開かれたとき、または特定の可視化が最大化されたときに、オーバーフローメニューで [エクスポート] ボタンが使用可能になります。



ユーザーが [エクスポート] ボタンをクリックすると、利用可能なエクスポートオプションを選択できます。

### 画像エクスポートオプションを使用する場合の詳細

ユーザーが [エクスポート] ボタンをクリックすると、[画像としてエクスポート] ダイアログが開きます。ユーザーは 2 つのオプションから選択することができます。[クリップボードへコピー] と [画像としてエクスポート]。

右下の [画像としてエクスポート] ボタンをクリックすると、RevealView は **ImageExported** イベントを発生します。 **ImageExportedEventArgs** の **ImageProperty** で画像にアクセスするには、イベントハンドラーを通してこのイベントにすでにサブスクライブしている必要があります。イベントをサブスクライブしていない場合は、ファイルの保存ダイアログが開き、ユーザーは画像を保存する場所を指定できます。

以下は **ImageExported** イベントハンドラーのサンプル実装です。

```
private void RevealView_ImageExported(object sender, ImageExportedEventArgs e)
{
    var image = e.Image;
    if (image == null) return;
    // アプリで開くためだけにディスクに保存します。
    var imageFile = Path.GetTempFileName() + ".png";
    using (var fileStream = new FileStream(imageFile, FileMode.Create))

    {
        BitmapEncoder encoder = new PngBitmapEncoder();
        encoder.Frames.Add(BitmapFrame.Create(image));
        encoder.Save(fileStream);
    }

    System.Diagnostics.Process.Start(imageFile);
}
```

**ImageExported** イベント引数の他のプロパティは、**CloseExportDialog** です。デフォルト値は `true` です。`false` に設定した場合、イベントハンドラーの呼び出しが終了した後に Export Image ダイアログは閉じられません。

エンドユーザーに保存ダイアログを表示してイメージの保存場所を選択できるようなシナリオでは、`CloseExportDialog` を `false` に設定すると便利な場合があります。ユーザーが場所とファイル名を選択せずに保存ダイアログを閉じた場合は、`ExportImage` ダイアログを開いたままにしておくことをお勧めします。

## コードによって開始されたエクスポート

コードで `RevealView` の画像を取得するには、**ToImage** メソッドを呼び出す必要があります。このメソッドを呼び出しても [画像としてエクスポート] ダイアログは表示されません。これにより、ユーザーが `RevealView` の外側にあるボタンをクリックしたときにスクリーンショットを取得できます。このメソッドは、`RevealView` コンポーネントが画面に表示されていると同じスクリーンショットを作成します。

`ToImage` メソッドの呼び出し時にユーザーがダイアログを開いている場合、ダッシュボードと一緒にそのダイアログのスクリーンショットが取得されます。

# ダッシュボード リンク

---

## 概要

Reveal アプリケーションはダッシュボードのリンクをサポートしているため、ユーザーはダッシュボードをナビゲートできます。ダッシュボードからダッシュボードに移動することで、業務上のハイレベルな概要からより詳細なビューに進むことができます。

## 一般的なユースケース

たとえば、各部門 (HR、Sales、Marketing) の主要業績評価指標を表示する Company 360 ダッシュボードを作成できます。ユーザーが表示形式の 1 つを最大化すると、ナビゲーションがトリガーされ、ユーザーはその部門に関するより詳細な情報を含む別のダッシュボードに移動します。

あるいは、ダッシュボード リンクを使用して、より具体的なダッシュボードに移動することもできます。たとえば、上位 25 の顧客を表示する可視化を含むダッシュボードで、顧客の1人を選択すると、ユーザーは新しいダッシュボードに移動します。この新しいダッシュボードには、最近の購入、連絡先情報、売れ筋商品など、選択した顧客に関する詳細情報が表示されます。

ダッシュボードへリンクする機能の詳細については、Reveal のユーザーガイドの[ダッシュボードのリンク](#)を参照してください。

## コード例

SDK とのダッシュボードリンクを使用できますが、ナビゲーションがトリガーされているときには包含アプリケーションが関与する必要があります。SDK はダッシュボードが格納されている場所を処理しないため、ターゲット ダッシュボード用のダッシュボードファイルを提供するには、それを含むアプリケーションが必要です。

実用的な例として、SDK とともに配布されている UpMedia サンプルの **Marketing.xaml.cs** ページを使用できます。

基本的にこの 2 項目を完了すると後は SDK によって処理されます。

1. **VisualizationLinkingDashboard** イベントを処理します。
2. ターゲット ダッシュボードの ID を使用してコールバックを呼び出します。

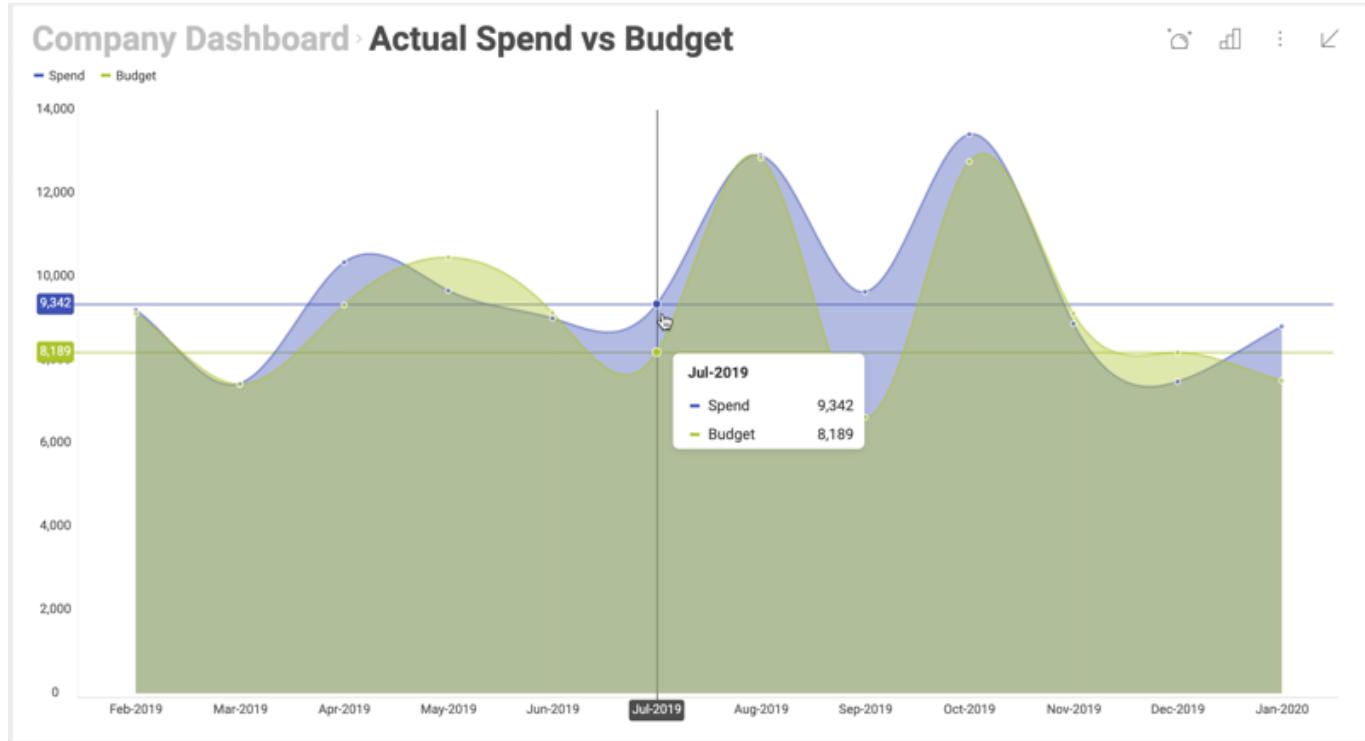
```
//VisualizationLinkingDashboard イベントに添付します。
revealView.VisualizationLinkingDashboard += RevealView_VisualizationLinking;

//イベント ハンドラーを実装します。
private void RevealView_VisualizationLinking(object sender,
    VisualizationLinkingDashboardEventArgs e)
{
    e.Callback("Campaigns", GetDashboardStream("Campaigns"));
}
```

# 可視化の最大化と単一可視化モード

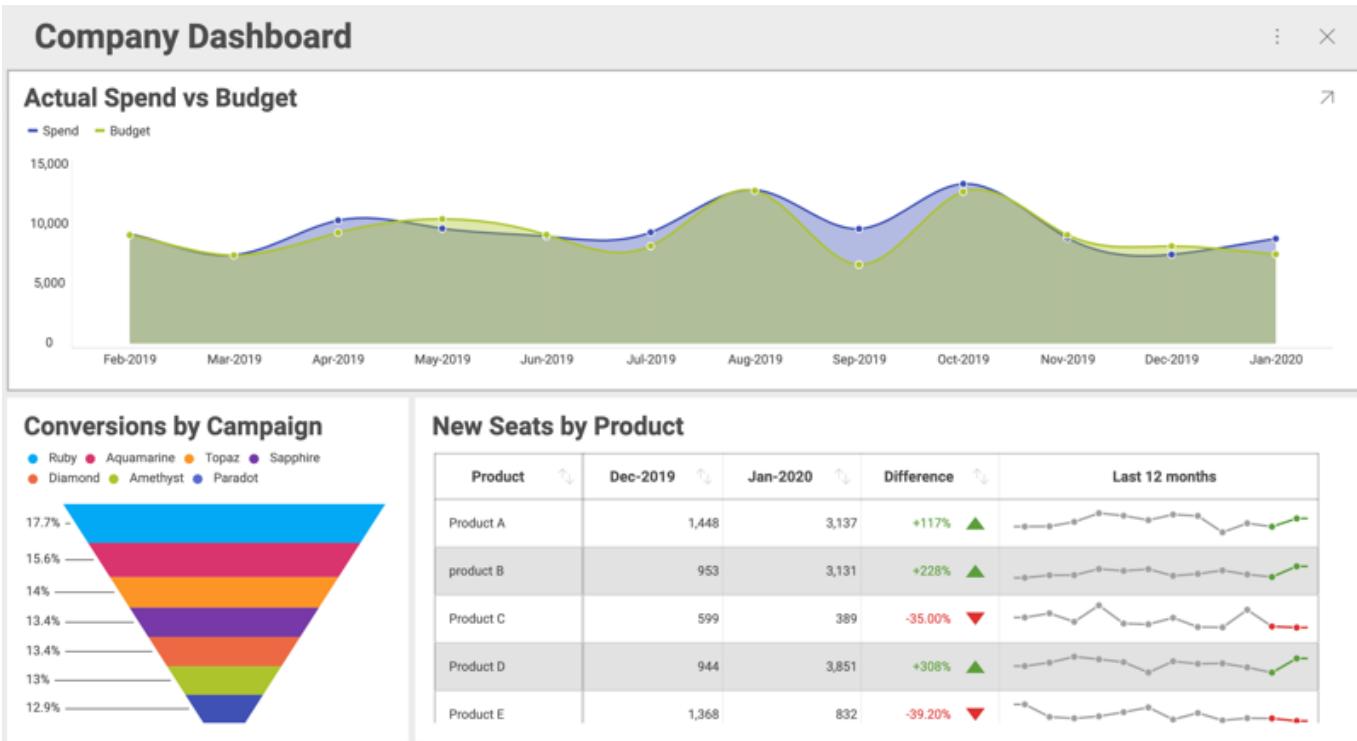
## 概要

Desktop SDK は、ダッシュボードをユーザーに表示する際に最大化した可視化表示を 1 つだけ表示し、更に最初の可視化表示をロックしてユーザーがダッシュボード全体にアクセスできないようにすることができます。



## サンプル詳細

可視化した 3 つのダッシュボードがあり、それぞれの可視化で会社の異なる部門のデータが表示されていると仮定します (例えば、Marketing、Sales、HR)。



この例では、業務アプリケーションでこれらの可視化を使用します。各部署のホームページに表示される情報の一部として含めます。

## 可視化の最大化

最大化された可視化でダッシュボードを開くには、`revealView.Dashboard` プロパティを割り当てた後、**RevealView の MaximizedVisualization** プロパティを使用する必要があります。この属性に視覚化を設定しないと、ダッシュボード全体が表示されます。

**RevealView オブジェクトの設定**に示すように、ページに特定のダッシュボードを表示できます。今回は、**MaximizedVisualization** 属性を設定する必要があります。以下のコード スニペットに示すように、可視化 Sales を使用してください。

```
var revealView = new RevealView();
using (var fileStream = File.OpenRead(path))
{
    var dashboard = new RVDashboard(fileStream);
    revealView.Dashboard = dashboard;
    revealView.MaximizedVisualization =
        dashboard.Visualizations.GetByTitle("Sales");
}
```

最初に最大化した可視化表示は Sales というタイトルの可視化になりますが、それでもエンドユーザーはダッシュボードに戻って残りの可視化を表示できます。

## 単一可視化モード

また、最初の可視化をロックして、常に可視化を 1 つのみ表示するようにすることもできます。これにより、ダッシュボードは単一の視覚化ダッシュボードのように機能します。これが [単一可視化モード] の概念

です。

单一可視化モードをオンにするには、以下に示すように、**SingleVisualizationMode** プロパティを **true** に設定します。

```
revealView.SingleVisualizationMode = true;
```

この 1 行を追加すると、ダッシュボードは単一の視覚化ダッシュボードとして機能します。各部門のホームページでも同じことができます。**dashboard.Visualizations.GetByTitle()** の可視化のタイトルを適切なタイトルに置き換えてください。

## ロックされた可視化を動的に変更

ページを再ロードせずに、表示されている単一の表示形式を動的に変更することもできます。ユーザーの観点から見ると、アプリは部門のセレクターと最大化された視覚化を備えた単一ページのアプリケーションになります。ユーザーがリストから 1 つの部門を選択すると、最大化された視覚化が更新されます。

以下では、**RevealView** の **MaximizeVisualization** メソッドを使用するか、**MaximizedVisualization** プロパティを設定してこのシナリオを実現できます。

```
private void MaximizeVisualization(string title)
{
    revealView.MaximizeVisualization(revealView.Dashboard.Visualizations.GetTitle(title));
    //or set the property
    revealView.MaximizedVisualization =
    revealView.Dashboard.Visualizations.GetTitle(title);
}
```

最後に、カスタム コントロールを上記の方法で接続します。それにより、アプリケーションの選択が変わったときに視覚化が最大化されます。

注意事項:

- ダッシュボードで可視化のリストを繰り返すことで、ボタンのリストを動的に生成できます。詳細については、**RVDashboard.Visualizations** をご覧ください。
- SDK とともに配布されている UpMedia WPF アプリケーションの UpMedia に、**Manufacturing.xaml.cs** の動作を確認できる例があります。このサンプルビューでは、画面下部にすべての可視化がトグル ボタンのリストとして表示されます。

# インメモリ データのサポート (Desktop)

## 概要

アプリケーション状態の一環として、ユーザーが要求したレポートの結果や Reveal でまだサポートされていないデータソースからの情報 (カスタムデータベースや特定のファイル形式など) にすでにメモリ内にあるデータを使用します。

インメモリは特別なタイプのデータソースで、SDK でのみ使用でき、Reveal アプリケーションでそのまま使用することはできません。このため、[インメモリ データ ソース] を直接使用することはできません。以下で説明するように、別のアプローチをとる必要があります。

## インメモリ データ ソースの使用

推奨される方法は、**インメモリ データと一致するスキーマを持つデータ ファイルを定義する**方法です。データ ファイルは、たとえば CSV または Excel ファイルにすることができ、スキーマは基本的にフィールドのリストと各フィールドのデータ型です。

以下の例では、特定のスキーマを使用してデータファイルを作成し、データベースから情報を取得する代わりにメモリ内のデータを使用する方法について詳しく説明します。

## コード例

次の例では、人事システムに人事メトリクスを表示するダッシュボードを埋め込むために、会社内の従業員のリストでインメモリデータを使用します。データベースから従業員のリストを取得するのではなく、メモリ内のデータを使用します。

これらすべてを実現するには、ダミー データを使用して、Reveal アプリでダッシュボードを作成してエクスポートする必要があります。

## Reveal アプリについて

Reveal アプリは、ダッシュボードを作成して表示し、チームと共有できるビジネス インテリジェンス ツールです。Reveal アプリの詳細については、[オンライン デモ](#)にアクセスするか、[ヘルプ ドキュメント](#)を参照いただけます。

## データ ファイルとサンプル ダッシュボードの準備

簡略化 Employee には以下のプロパティがあります。

- *EmployeeID*: string
- *Fullname*: string
- *Wage*: numeric

## 手順

1. 同じスキーマで CSV ファイルを作成します。

```
EmployeeID,Fullname,Wage  
23,John Smith,345.67  
45,Emma Thompson,432.23
```

2. Dropbox や Google Drive など、ファイル共有システムにファイルをアップロードしてください。
3. ダミーデータを使用してダッシュボードを作成します。実際の生成データは後でアプリケーションで提供します。
4. ダッシュボードをエクスポートして (ダッシュボード -> エクスポート -> ダッシュボード)。

## ダッシュボードの可視化と実際のデータの返却

ダミーのデータではなくカスタムのデータを使ってダッシュボードを可視化する必要があります。

1. [データソースの置き換え](#)を参照して、**IRVDataSourceProvider** を実装し、**RevealSdkSettings** の **DataSourceProvider** プロパティに設定します。

次に、メソッド **ChangeVisualizationDataSourceItemAsync** の実装では、次のようなコードを追加する必要があります。

```
public Task<RVDataSourceItem>  
ChangeVisualizationDataSourceItemAsync(RVVisualization visualization,  
RVDataSourceItem dataSourceItem)  
{  
    var csvDsi = dataSourceItem as RVCsvDataSourceItem;  
    if (csvDsi != null)  
    {  
        var inMemDsi = new RVInMemoryDataSourceItem("employees");  
        return Task.FromResult((RVDataSourceItem)inMemDsi);  
    }  
    return Task.FromResult((RVDataSourceItem)null);  
}
```

このようにして、ダッシュボード内の CSV ファイルへのすべての参照を、基本的に employees で識別されるインメモリ データ ソースに置き換えます。この ID は後でデータを返すときに使用されます。

2. 以下のように **IRVDataProvider** を実装するために、実際のデータを返すメソッドを実装します。

```
public class EmbedDataProvider : IRVDataProvider  
{  
    public Task<RVInMemoryData> GetData(RVInMemoryDataSourceItem  
dataSourceItem)  
    {  
        var datasetId = dataSourceItem.DatasetId;  
        if (datasetId == "employees")  
        {  
            var data = new List<Employee>()
```

```

    {
        new Employee(){ EmployeeID = "1", Fullname="John Doe",
Wage = 80325.61 },
        new Employee(){ EmployeeID = "2", Fullname="Doe John",
Wage = 10325.61 },
    };
    return Task.FromResult<IRVInMemoryData>(new
RVInMemoryData<Employee>(data));
}
else
{
    throw new Exception("Invalid data requested");
}
}
}

```

Employee クラスのプロパティは CSV ファイルの列とまったく同じ名前であり、データ型も同じです。フィールド名、フィールド ラベル、またはプロパティのデータ型を変更したい場合は、クラス宣言で次の属性を使用できます。

- RVSchemaColumn 属性を使用してフィールド名やデータ型を変更できます。
- DisplayName 属性を使用してフィールドラベルを変更できます。

```

public class Employee
{
    [RVSchemaColumn("EmployeeID", RVSchemaColumnType.Number)]
    public string EmployeeID { get; set; }

    [DisplayName("EmployeeFullName")]
    public string Fullname { get; set; }

    [RVSchemaColumn("MonthlyWage")]
    public double Wage { get; set; }
}

```

最後に、**IRVDataProvider** から **RevealView.DataProvider** のプロパティの実装を設定してください。

```
revealView.DataProvider = new SampleDataProvider();
```

# データ ソースへの資格情報の提供

## 概要

Server SDK では、データ ソースにアクセスするときに使用される一連の資格情報を渡すことができます。

## コード

最初の手順は、以下に示すように、**IRVAuthenticationProvider** を実装し、それを **RevealSdkSettings** の **AuthenticationProvider** プロパティに設定します。

```
public class EmbedAuthenticationProvider : IRVAuthenticationProvider
{
    public Task<IRVDataSourceCredential>
ResolveCredentialsAsync(RVDashboardDataSource dataSource)
    {
        IrvDataSourceCredential userCredential = null;
        if (dataSource is RVPostgresDataSource)
        {
            userCredential = new
RVUsernamePasswordDataSourceCredential("postgresuser", "password");
        }
        else if (dataSource is RVSqlServerDataSource)
        {
            userCredential = new
RVUsernamePasswordDataSourceCredential("sqlserveruser", "password", "domain");
        }
        else if (dataSource is RVGoogleDriveDataSource)
        {
            userCredential = new
RBearerTokenDataSourceCredential("fhJhbUci0mJSUzi1nIiSint....",
"user@company.com");
        }
        else if (dataSource is RVRestDataSource)
        {
            userCredential = new RVUsernamePasswordDataSourceCredential(); //匿名
        }
        return Task.FromResult<IRVDataSourceCredential>(userCredential);
    }
}
```

## 実装するクラスの選択

使用できるクラスは 2 つあり、どちらも **IRVDataSourceCredential** インターフェイスを実装します。以下に詳述するように、データ ソースに応じてクラスを選択する必要があります。

- クラス **RVBearerTokenDataSourceCredential** は以下と動作します。

- アナリティクス ツール (Google アナリティクス)
- コンテンツ マネージャーとクラウド サービス (Box、Dropbox、Google Drive、OneDrive、SharePoint Online)。
- クラス **RVUsernamePasswordDataSourceCredential** は以下と動作します。
  - カスタマー リレーションシップ マネージャ- (Microsoft Dynamics CRM オンプレミスおよびオンライン)
  - データベース (Microsoft SQL Server、Microsoft Analysis Services サーバー、MySQL、PostgreSQL、Oracle、Sybase)
- **両クラス** は以下と動作します。
  - その他のデータ ソース (OData フィード、Web Resources、REST API)

## 認証なし

認証なしの匿名のリソースで作業する場合は、空のコンストラクタを持つ

**RVUsernamePasswordDataSourceCredential** を使用できます。これは、そのクラスで機能するすべてのデータ ソースに対して実行できます。

上記のサンプルで使用するコード スニペット:

```
else if (dataSource is RVRESTDataSource)
{
    userCredential = new RVUsernamePasswordDataSourceCredential();
}
```

# データ ソースの置き換え (MS SQL サーバー) (Desktop)

## 概要

ダッシュボードのデータを (Reveal SDK によって) ロードして処理する前に、ダッシュボードの各視覚化に使用される構成またはデータをオーバーライドできます。

**RevealSdkSettings** の `DataSourceProvider` プロパティを設定する必要があります:

```
public IRVDataSourceProvider DataSourceProvider { get; set; }
```

インターフェイス **IRVDataSourceProvider** を実装するクラスは、特定の可視化またはダッシュボード フィルターによって使用されるデータ ソースを置換または変更することができます。

## 使用事例

以下に、一般的なユースケースのリストがあります。

- 現在のユーザー、またはアプリが `userId`、`division`、`company`、`customer` などのその他の属性に応じて、使用するデータベースの名前を変更できます。これにより、マルチテナント データベースからデータを取得する単一のダッシュボードを持つことができます。
- 使用されているテーブルの名前、ロードするファイルのパスなどを変更することができます。ユースケースは上記のものと似ています。
- データ ソースをメモリ内のデータ ソースに置き換えることができます。Reveal App はインメモリ データ ソースをサポートしていないため、CSV ファイルを使用してダッシュボードを設計し、このコードバックを使用して CSV データ ソースをインメモリデータ ソースに置き換えることができます。このシナリオでは、データは実際にメモリからロードされます (またはカスタムデータローダを使用して)。インメモリ データ ソースの使用方法の詳細については、[インメモリ データのサポート](#)を参照してください。

## コード

次のコードスニペットは、ダッシュボードの可視化のためにデータ ソースを置き換える方法の例を示しています。メソッド **ChangeVisualizationDataSourceItemAsync** は、開かれているすべてのダッシュボードで、すべての可視化に対して呼び出されます。

```
public class SampleDataSourceProvider : IRVDataSourceProvider
{
    public Task<RVDataSourceItem> ChangeDashboardFilterDataSourceItemAsync(
        RVDashboardFilter globalFilter, RVDataSourceItem dataSourceItem)
    {
        return Task.FromResult<RVDataSourceItem>(null);
    }
}
```

```

public Task<RVDataSourceItem> ChangeVisualizationDataSourceItemAsync(
    RVVisualization visualization, RVDataSourceItem dataSourceItem)
{
    var sqlServerDsi = dataSourceItem as RVSqlServerDataSourceItem;
    if (sqlServerDsi != null)
    {
        // SQL サーバー ホストとデータベースの変更
        var sqlServerDS = (RVSqlServerDataSource)sqlServerDsi.DataSource;
        sqlServerDS.Host = "10.0.0.20";
        sqlServerDS.Database = "Adventure Works";

        // SQL サーバー テーブル/ビュー
        sqlServerDsi.Table = "Employees";
        return Task.FromResult((RVDataSourceItem)sqlServerDsi);
    }

    // データ ソース アイテムを新しいアイテムと置き換える
    if (visualization.Title == "Top Customers")
    {
        var sqlDs = new RVSqlServerDataSource();
        sqlDs.Host = "salesdb.local";
        sqlDs.Database = "Sales";

        var sqlDsi = new RVSqlServerDataSourceItem(sqlDs);
        sqlServerDsi.Table = "Customers";

        return Task.FromResult((RVDataSourceItem)sqlServerDsi);
    }

    return Task.FromResult((RVDataSourceItem)dataSourceItem);
}
}

```

上記の例では、次の 2 つの置換が実行されます。

- MS SQL Server データベースを使用するすべてのデータソースは、ハードコードされたサーバー 10.0.0.20、Adventure Works データベース、および Employees テーブルを使用するように変更されます。

[!NOTE] これは単純化されたシナリオで、同じテーブルからデータを取得するためにすべての可視化を置き換えるても、現実のシナリオとしては意味がありません。実際のアプリケーションでは、userId、dashboardId、データソース自体の値(サーバー、データベースなど)などの追加情報を使用して新しい値を推測します。

- Top Customers というタイトルのすべてのウィジェットは、Customers テーブルを使用して salesdb.local サーバーの Sales データベースからデータを取得する新しい SQL Server データソースに設定されます。

**IRVDataSourceProvider** の実装に加え、**RevealSdkSettings** の **DataSourceProvider** プロパティに実装を設定します。

```
RevealSdkSettings.DataSourceProvider = new SampleDataSourceProvider();
```

# Excel および CSV ファイルのデータ ソースの置き換え

## 概要

Reveal アプリでダッシュボードを作成するとき、クラウドに保存されている Excel または CSV ファイルを使用してデータを入力することがよくあります。

ダッシュボードをエクスポートしてカスタム アプリケーションに組み込んだ後、これらのファイルをローカルディレクトリに移動し、Reveal SDK を使用してアクセスし、ローカルデータ ソースとして設定できます。

## 手順

ローカルの Excel ファイルと CSV ファイルを使用してエクスポートされた ダッシュボードにデータを入力するには、次の手順に従う必要があります

1. [SDK 用のダッシュボードの取得](#)の説明に従って、**ダッシュボード ファイルをエクスポートします。**
2. [ダッシュボード ファイルの読み込み](#)の説明に従って、アプリケーションに**ダッシュボードを読み込みます。**
3. ダッシュボードの作成に使用した**ファイルをクラウド ストレージからダウンロードし、ローカル フォルダーにコピーします。**
4. **ローカル フォルダーネ名**を *Reveal.SdkSettings.LocalFilesRootFolder* プロパティの値として**設定します**。  
*Datasources* をローカル フォルダーとして設定するための参照として、  
*Reveal.Sdk.Samples.UpMedia.Wpf* サンプル アプリケーションを使用できます。サンプル アプリケーションには、Reveal SDK のインストールが付随しています。
5. プロジェクトに**新しい CloudToLocalDatasourceProvider クラスを追加します**。
6. 以下のコード セクションの関連するスニペットから**実装コードをコピーします**。
7. *RevealSdkContext* クラスの **DataSourceProvider** プロパティを **CloudToLocalDatasourceProvider** に**設定します**:

```
public override IRVDataSourceProvider DataSourceProvider => new
CloudToLocalDatasourceProvider();
```

## コード

```
public class CloudToLocalDatasourceProvider : IRVDataSourceProvider
{
    public Task<RVDataSourceItem>
ChangeDashboardFilterDataSourceItemAsync(RVDashboardFilter filter,
RVDataSourceItem dataSourceItem)
    {
        return ProcessDataSourceItem(dataSourceItem);
    }
    public Task<RVDataSourceItem>
ChangeVisualizationDataSourceItemAsync(RVVisualization visualization,
RVDataSourceItem dataSourceItem)
```

```

    {
        return ProcessDataSourceItem(dataSourceItem);
    }
    protected Task<RVDataSourceItem> ProcessDataSourceItem(RVDataSourceItem
dataSourceItem)
    {
        // Return data source unless it is an excel or csv file.
        if (dataSourceItem is RVExcelDataSourceItem == false &&
            dataSourceItem is RVCsvDataSourceItem == false)
        {
            return Task.FromResult(dataSourceItem);
        }

        var resourceBased = dataSourceItem as RVResourceBasedDataSourceItem;
        var resourceItem = resourceBased?.ResourceItem as RVDataSourceItem;
        var title = resourceItem?.Title;

        if (string.IsNullOrEmpty(title))
        {
            return Task.FromResult(dataSourceItem);
        }

        var localItem = new RVLocalFileDataSourceItem();

        // The SDK uses the local folder set as
        Reveal.SdkSettings.LocalFilesRootFolder
        localItem.Uri = @"local:/" + title;

        // The title assigned here is the original data source name.
        localItem.Title = title;
        resourceBased.ResourceItem = localItem;

        return Task.FromResult(dataSourceItem);
    }
}

```

[!NOTE] *CloudToLocalDatasourceProvider* は、Excel ファイルと CSV ファイルのみを自動的に置き換えます。他のファイルタイプまたはデータソースは変更されません。置換ファイルは、ダッシュボードの作成に使用したものと同じであるか、**同じスキーマ**を共有しているがデータが異なる新しいファイルである必要があります。

## ローカリゼーションサービスの使用

ローカリゼーションサービスは、カスタムロジックに基づいてさまざまなダッシュボード要素をローカライズできます。また、フィールドのカスタム書式設定を設定する機能も提供します。

### ローカライズがサポートされる要素

ローカライズ可能なダッシュボード要素:

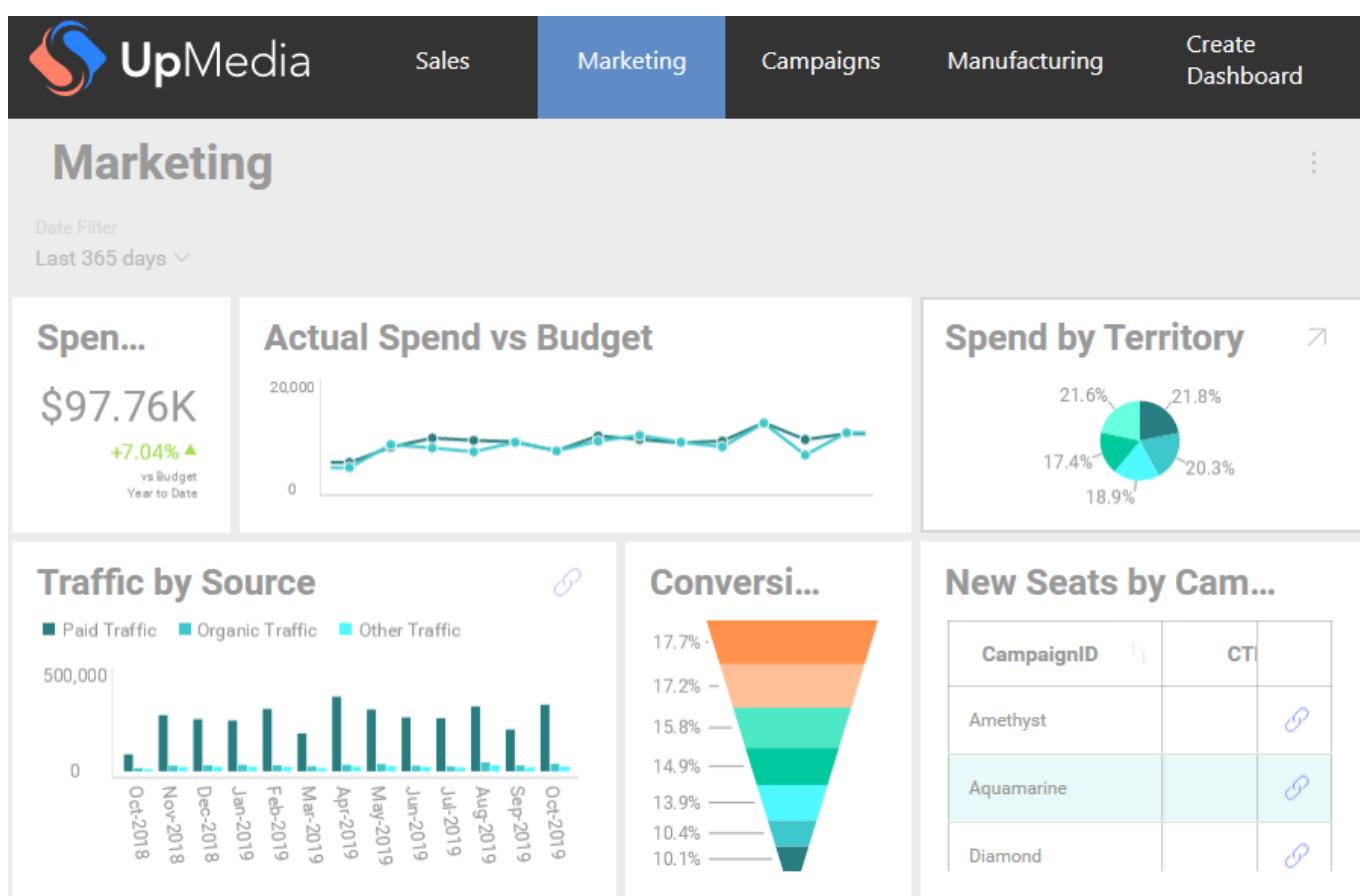
- ダッシュボードのタイトル
- ダッシュボード フィルター タイトル
- 可視化フィルター タイトル
- フィールド ラベル
- 集計フィールド ラベル

## ローカリゼーションサービスの使用

以下は、リンクする方法の 2 つの例で [ダッシュボード タイトルのローカライズ](#) およびカスタム ロジックを追加して同じダッシュボードで [フィールド ラベルをローカライズ](#) する方法があります。また、[数値フィールド](#) および [非集計日付フィールド](#) の書式設定を変更する方法の例も示します。この例で使用するダッシュボードは、Marketing サンプル ダッシュボードです。

### ダッシュボードのタイトルのローカライズの例

Marketing サンプルの初期状態:



以下の手順に従って、Marketing ダッシュボードのタイトルを Localized Marketing にローカライズします。

1. ダッシュボードのローカライズを許可するには、カスタム実装に **LocalizationProvider** プロパティを設定する必要があります。

```
RevealSdkSettings.LocalizationProvider = new UpMediaLocalizationProvider()
```

## 2. IRVLocalizationProvider の実装:

```
public class UpMediaLocalizationProvider : IRVLocalizationProvider
{
    public IRVLocalizationService GetLocalizationService()
    {
        return new UpMediaLocalizationService();
    }
}
```

## 3. ダッシュボード タイトルをローカライズするには、以下に示すようにIRVLocalizationService で GetLocalizedString メソッドを実装します。

```
public class UpMediaLocalizationService : IRVLocalizationService
{
    public RVFormattingSpec GetFormattingSettingsForField(string fieldName,
RVDashboardDataType dataType, RVFormattingSpec currentSettings, bool
isAggregated)
    {
        return null;
    }

    public string GetLocalizedString(string originalValue,
RVLocalizationElementType type)
    {
        if (type == RVLocalizationElementType.DashboardTitle &&
originalValue == "Marketing")
        {
            return "Localized Marketing";
        }

        return originalValue;
    }
}
```

アプリを再度実行すると、ローカライズされたダッシュボードのタイトル (Localized Marketing) が表示されます。



## Sales

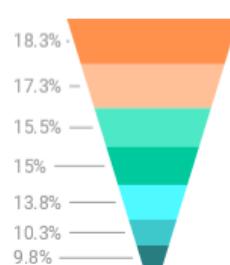
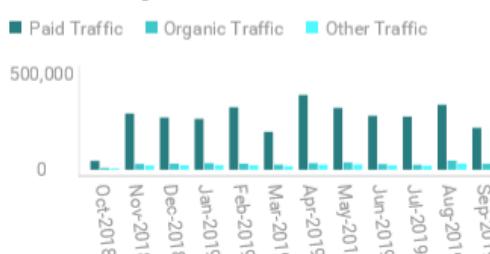
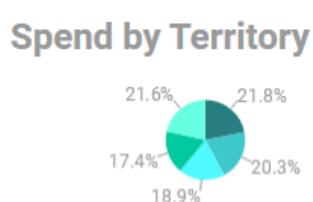
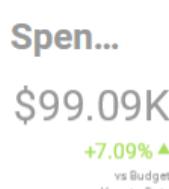
Marketing

## Campaigns

## Manufacturing

Create  
Dashboard

## Localized Marketing

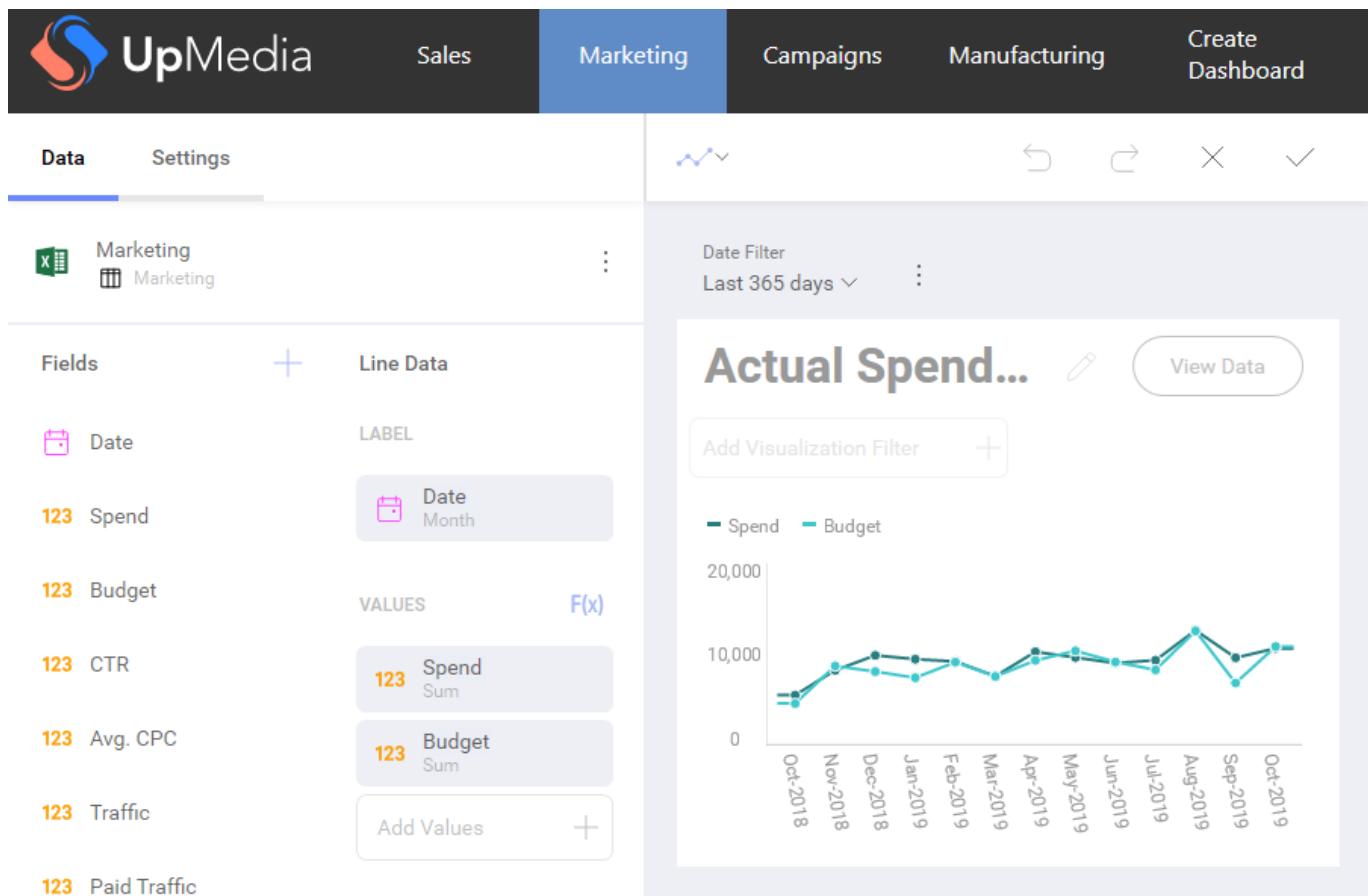


CampaignID	CTI
Amethyst	<a href="#">🔗</a>
Aquamarine	<a href="#">🔗</a>
Diamond	<a href="#">🔗</a>

#### フィールド ラベルのローカライズの例

以下に、同じダッシュボードの複数の要素をローカライズする方法の例を示します。

以下は、Marketing サンプルの可視化 (Actual Spend vs Budget) の初期状態の 1 つの例です。



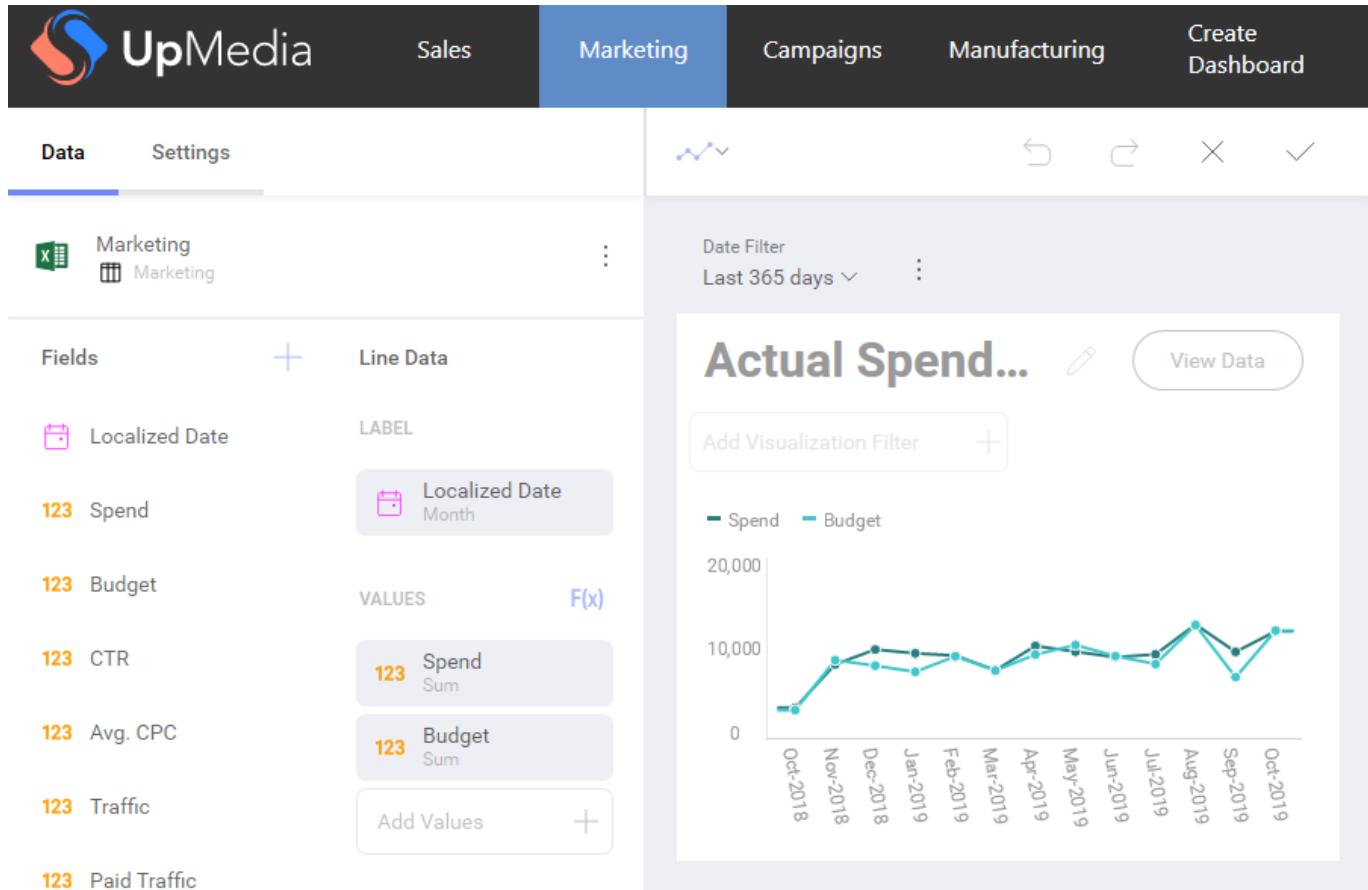
Date フィールド ラベルをローカライズするには、Date フィールドのローカライズを処理するロジックを **UpMediaLocalizationService** に追加する必要があります。

```
public class UpMediaLocalizationService : IRVLocalizationService
{
    public RVFormattingSpec GetFormattingSettingsForField(string fieldName,
    RVDashboardDataType dataType, RVFormattingSpec currentSettings, bool isAggregated)
    {
        return null;
    }

    public string GetLocalizedString(string originalValue,
    RVLocalizationElementType type)
    {
        if (type == RVLocalizationElementType.DashboardTitle && originalValue == "Marketing")
        {
            return "Localized Marketing";
        }
        else if (type == RVLocalizationElementType.FieldLabel && originalValue == "Date")
        {
            return "Localized Date";
        }

        return originalValue;
    }
}
```

Actual Spend vs Budget の Date フィールド ラベルが Localized Date に変更されました。



手順の例に従って、その他のダッシュボード要素をローカライズできます。

## ローカリゼーションサービスを使用した書式設定の変更

ローカリゼーションサービスを使用して数値フィールドと非集計日付フィールドの書式設定を変更できます。

### 数値フィールドの書式設定の変更の例

以下の Spend vs Budget 可視化の初期状態では、米ドル(\$) 通貨で書式設定された数値フィールドを示します。

## Localized Marketing &gt; Spend vs Budget

# : ↺

\$99.81K

+7.53%▲

vs Budget / Year to Date

通貨書式を変更するには、新しい書式設定を作成し、**IRVLocalizationService** の実装の **GetFormattingSettingsForField** メソッドでそれらを返す必要があります。

コード スニペットは、数値の書式設定を日本円(¥)に変更し、小数桁なしで表示する方法を示します。

```
public class UpMediaLocalizationService : IRVLocalizationService
{
    public RVFormattingSpec GetFormattingSettingsForField(string fieldName,
    RVDashboardDataType dataType, RVFormattingSpec currentSettings, bool isAggregated)
    {
        if (fieldName == "Spend" && dataType == RVDashboardDataType.Number &&
isAggregated == true)
        {
            var newSettings = new RVNumberFormattingSpec()
            {
                ApplyMkFormat = false,
                CurrencySymbol = "¥",
                DecimalDigits = 0,
                FormatType = RVDashboardNumberFormattingType.Currency,
                NegativeFormat = RVDashboardNegativeFormatType.MinusSign,
                ShowGroupingSeparator = true
            };

            return newSettings;
        }

        return null;
    }
}
```

```
public string GetLocalizedString(string originalValue,  
RVLocalizationElementType type)  
{  
    ...  
}  
}
```

現在、金額はその他の通貨で表示されます。



## 日付フィールドの書式設定の変更の例

現在、集計された日付フィールドの書式設定の変更はローカリゼーション サービスでは設定できず、ピボットの値には影響しません。変更には書式設定サービスを使用してください。

ローカリゼーション サービスでは、**非集計日付フィールド**の書式設定を変更できます。

はじめに、集計データを除外するために、Actual Spend vs Budget 可視化を **Grid** に変更します。



Sales

Marketing

Campaigns

Manufacturing

Create Dashboard

Date Filter

Last 365 days

Add Filter



## Localized Marketing > Actual Spend vs Budget



Localized Date	Spend	Budget
14-Mar-2019	¥536	341
21-Feb-2019	¥448	535
07-Sep-2019	¥403	264
31-May-2019	¥556	270
26-Mar-2019	¥769	607
13-Sep-2019	¥690	211
10-Nov-2018	¥398	254
...	...	...

Date フィールドの書式設定を変更するには、**GetFormattingSettingsForField** メソッドのロジックに設定を追加する必要があります。以下のコード スニペットは、January 01、2001 のように、日付形式を変更して月のフルネームを表示する方法を示します。

```
public RVFormattingSpec GetFormattingSettingsForField(string fieldName,
RVDashboardDataType dataType, RVFormattingSpec currentSettings, bool isAggregated)
{
    if (fieldName == "Spend" && dataType == RVDashboardDataType.Number &&
isAggregated == true)
    {
        var newSettings = new RVNumberFormattingSpec()
        {
            ApplyMkFormat = false,
            CurrencySymbol = "¥",
            DecimalDigits = 0,
            FormatType = RVDashboardNumberFormattingType.Currency,
            NegativeFormat = RVDashboardNegativeFormatType.MinusSign,
            ShowGroupingSeparator = true
        };

        return newSettings;
    }
    else if (fieldName == "Date" && dataType == RVDashboardDataType.Date &&
isAggregated == false)
    {
        var newSettings = new RVDateFormattingSpec()
        {
            DateFormat = "MMMM dd,yyyy"
        };
    }
}
```

```

    };

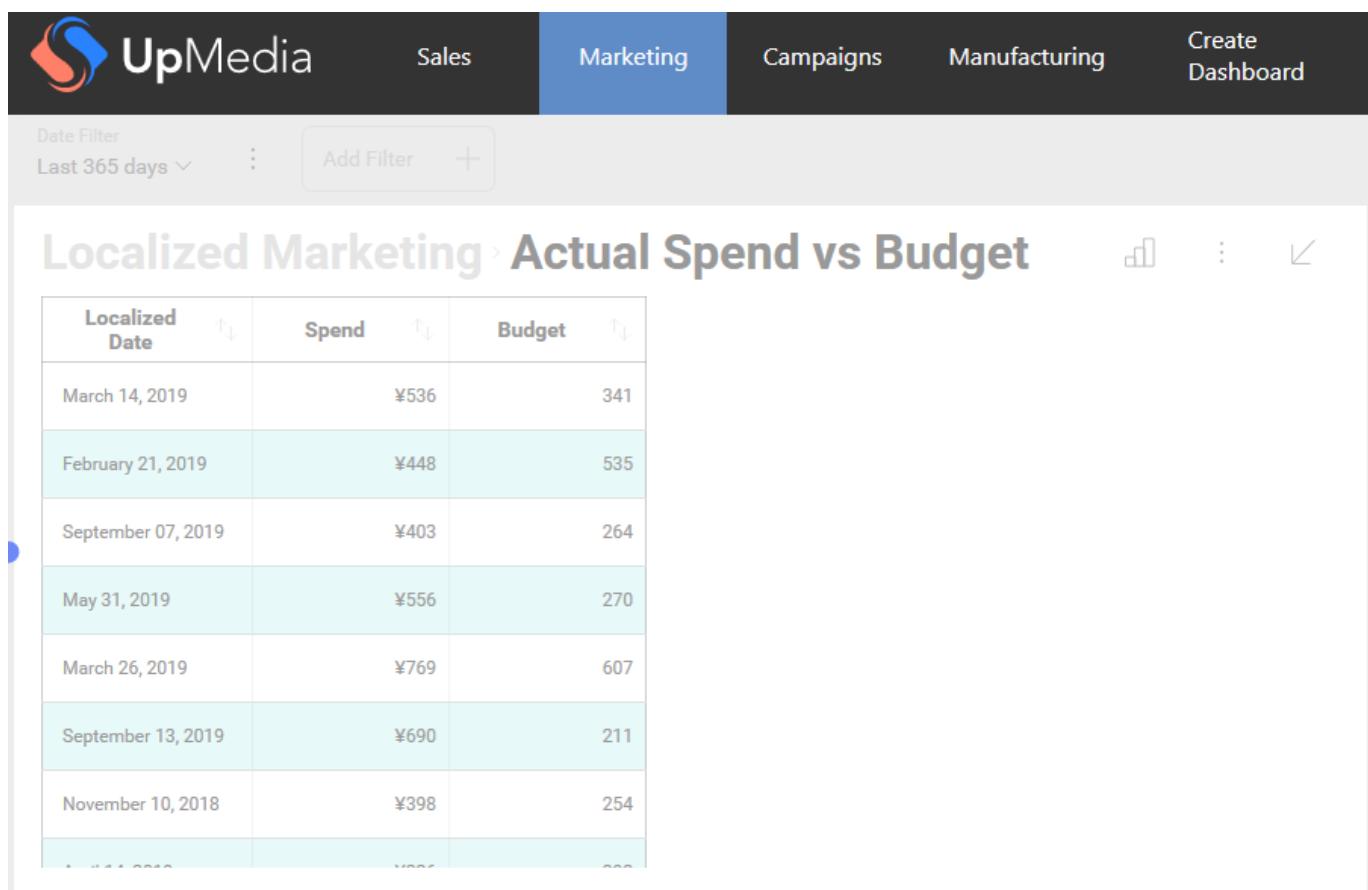
    return newSettings;
}

return null;
}

```

[!NOTE] フィールド名が Localized Date ではなく Date であるかどうかを確認する必要があることに注意してください。これは、フィールドの名前に基づいて書式設定が適用されるためです。この場合、Date はフィールドの名前で、Localized Date は表示されるラベルです。ダッシュボードを編集する場合、フィールド ラベルは変更できますが、フィールド名は元の名前のままでです。

アプリを再度実行し、可視化をグリッドに変更すると更新された日付形式が表示されます。



The screenshot shows the UpMedia dashboard interface. At the top, there is a navigation bar with tabs for Sales, Marketing (which is currently selected), Campaigns, Manufacturing, and a Create Dashboard button. Below the navigation bar, there is a date filter section with a dropdown set to "Last 365 days" and a "Add Filter" button. The main content area displays a grid titled "Localized Marketing > Actual Spend vs Budget". The grid has three columns: "Localized Date", "Spend", and "Budget". The data rows are:

Localized Date	Spend	Budget
March 14, 2019	¥536	341
February 21, 2019	¥448	535
September 07, 2019	¥403	264
May 31, 2019	¥556	270
March 26, 2019	¥769	607
September 13, 2019	¥690	211
November 10, 2018	¥398	254
Total	¥3,669	2,267

# 書式設定サービスの使用

書式設定サービスは、フィールドの書式設定を使用せずにダッシュボードデータをカスタムに書式設定できます。

## 書式設定がサポートされる要素

書式設定がサポートされる要素

- 数値データ
- 日付、時間、または日時のデータ
- 集計される日、時間、または日時のデータ

## 書式設定サービスの使用

以下は、書式設定の方法の 3 つの例を示します。[数値データ](#)、[集計した日付データ](#)、[\(未集計\) 日時データ](#)。この例で使用されるダッシュボードは、*Marketing* サンプル ダッシュボードです。

### 数値データの書式設定例

以下に、**Marketing** サンプル ダッシュボードの New Seats by Campaign ID 可視化の初期状態を示します。

The screenshot shows the UpMedia Marketing sample dashboard. At the top, there is a navigation bar with the UpMedia logo, Sales, Marketing (which is highlighted in blue), Campaigns, Manufacturing, and a 'Create Dashboard' button. Below the navigation bar, the main content area has a title 'Marketing > New Seats by Campaign ID'. The data is presented in a table:

CampaignID ↑↓	CTR ↑↓	Avg. CPC ↑↓	New Seats ↑↓	
Amethyst	30.98%	\$11	41,605	🔗
Aquamarine	29.77%	\$10	46,257	🔗
Diamond	30.41%	\$11	38,353	🔗
Paradot	29.88%	\$10	39,659	🔗
Ruby	30.97%	\$11	50,990	🔗
Sapphire	30.74%	\$11	38,866	🔗
Topaz	30.36%	\$11	41,571	🔗

以下の手順に従って、数値データを書式設定して 5 行の 10 進数を表示します。

1. ダッシュボード データの書式を許可するには、**FormattingProvider** プロパティをカスタム実装に設定します。

```
RevealSdkSettings.FormattingProvider = new UpMediaFormattingProvider();
```

## 2. **IRVFormattingProvider** プロバイダーを実装します。

```
public class UpMediaFormattingProvider : IRVFormattingProvider
{
    public RVBaseFormattingService GetFormattingService()
    {
        return new UpMediaFormattingService();
    }
}
```

## 3. **RVBaseFormattingService** の実装で **FormatNumber** メソッドをオーバーライドします。

```
public class UpMediaFormattingService : RVBaseFormattingService
{
    public override string FormatNumber(double value, RVNumberFormattingSpec
formatting, bool ignoreMkFormatting)
    {
        return string.Format("{0:0.00000}", value);
    }
}
```

アプリを再実行すると、すべての数値データが 5 衔の 10 進数で数値を表示するように書式設定され、他のすべての書式設定 (フィールドが通貨またはパーセンテージを表すかどうかなど) が無視されることがわかります。

## Marketing &gt; New Seats by Campaign ID

CampaignID ↑↓	CTR ↑↓	Avg. CPC ↑↓	New Seats ↑↓	
Amethyst	0.30984	10.69906	41605.00000	
Aquamarine	0.29771	10.40072	46257.00000	
Diamond	0.30412	10.62603	38353.00000	
Paradot	0.29880	10.46180	39659.00000	
Ruby	0.30966	10.64822	50990.00000	
Sapphire	0.30741	11.28615	38866.00000	
Topaz	0.30362	10.73305	41571.00000	

たとえば、パーセンテージを表示するフィールドのみの書式を変更する場合は、数値フィールドのタイプのチェックを追加する必要があります。

```
public class UpMediaFormattingService : RVBaseFormattingService
{
    public override string FormatNumber(double value, RVNumberFormattingSpec
formatting, bool ignoreMkFormatting)
    {
        if (formatting.FormatType == RVDashboardNumberFormattingType.Percent)
        {
            return string.Format("{0:0.00000%}", value);
        }

        return base.FormatNumber(value, formatting, ignoreMkFormatting);
    }
}
```

これで、書式設定されたパーセンテージ フィールドのみができました。

## Marketing > New Seats by Campaign ID

grid
⋮
left arrow

CampaignID ↑↓	CTR ↑↓	Avg. CPC ↑↓	New Seats ↑↓	
Amethyst	30.98438%	\$11	41,605	🔗
Aquamarine	29.77124%	\$10	46,257	🔗
Diamond	30.41176%	\$11	38,353	🔗
Paradot	29.87970%	\$10	39,659	🔗
Ruby	30.96552%	\$11	50,990	🔗
Sapphire	30.74074%	\$11	38,866	🔗
Topaz	30.36170%	\$11	41,571	🔗

ダッシュボードのその他の可視化では、数値データの書式が変更されていないことがわかります。書式設定はチャートによって制御されるため、フィールドの書式設定を変更するには、フィールドの書式設定を変更する必要があります。変更には、[ローカライゼーションサービス](#)を使用してください。

## Marketing

Date Filter  
 Last 365 days ▾

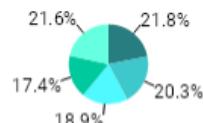
**Spen...** ↗

**\$101.21K**  
 +6.98%▲  
 vs Budget / Year to Date

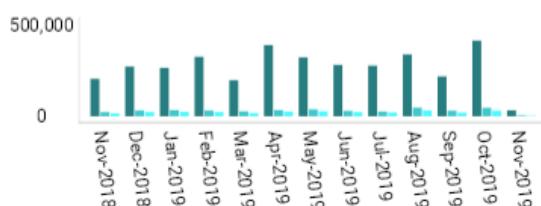
### Actual Spend vs Budget



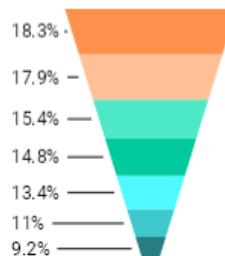
### Spend by Territory



### Traffic by Source

Paid Traffic   Organic Traffic   Other Traffic


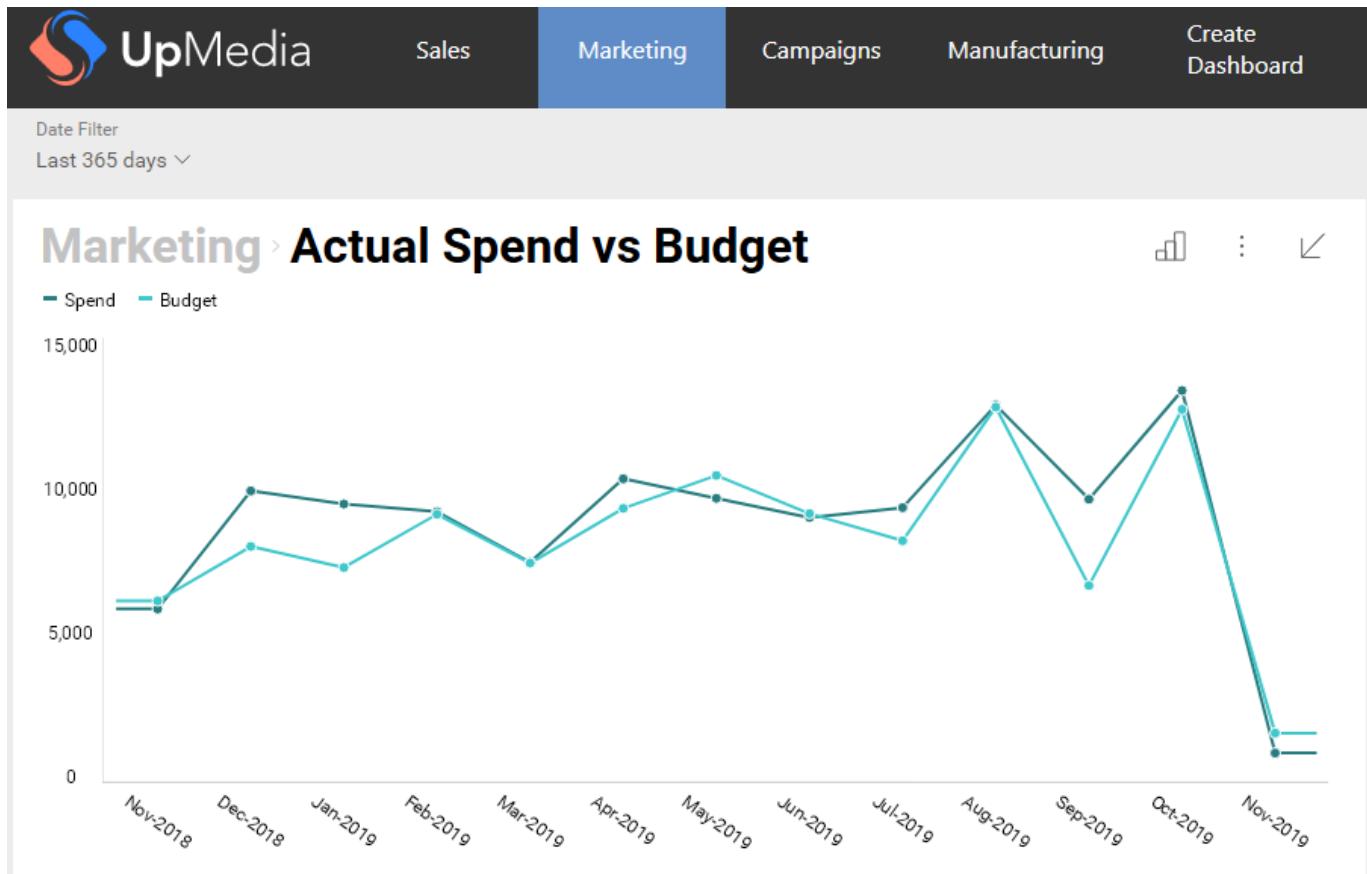
### Conversi...



### New Seats by Cam...

CampaignID ↑↓	CT	
Amethyst	🔗	
Aquamarine	🔗	
Diamond	🔗	

これは、Marketing サンプル **Actual Spend vs Budget** の可視化のうちの 1 つの初期状態です。



以下に、完全な月名を表示するために集約された日付データを表示する方法の例を示します (例: January 2001)。これを行うには、**RVBaseFormattingService** の実装で **FormatAggregatedDate** メソッドをオーバーライドする必要があります。

```
public class UpMediaFormattingService : RVBaseFormattingService
{
    public override string FormatAggregatedDate(DateTime value,
        RVDashboardDataType type, RVDashboardDateAggregationType aggregation,
        RVDateFormattingSpec formatting)
    {
        if (aggregation == RVDashboardDateAggregationType.Month)
        {
            return string.Format("{0:MMMM yyyy}", value);
        }

        return base.FormatAggregatedDate(value, type, aggregation);
    }
}
```

アプリを再度実行すると、更新された日付が表示されます。

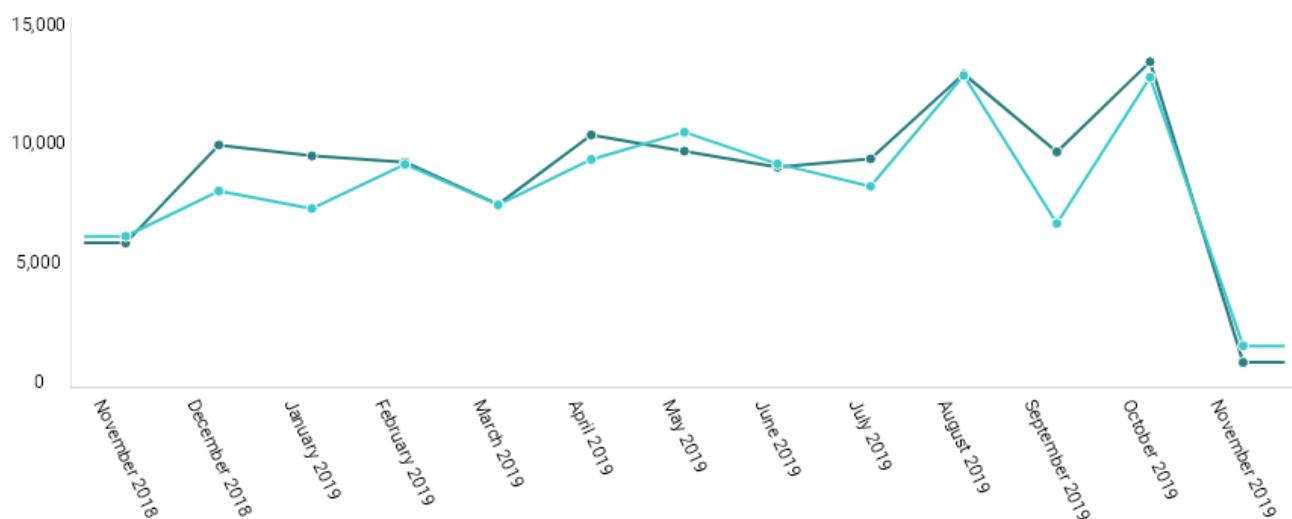
Date Filter

Last 365 days ▾

## Marketing > Actual Spend vs Budget



Spend    Budget



### 日時(非集計)データ書式設定の例

以下に、非集計の日付データを書式設定する方法の例を示します。まず、集計データを除外するために、**Actual Spend vs Budget** 可視化を **Grid** に変更します。

Date Filter

Last 365 days ▾



Add Filter



## Marketing > Actual Spend vs Budget



Date ↑↓	Spend ↑↓	Budget ↑↓
14-Mar-2019	536	341
21-Feb-2019	448	535
07-Sep-2019	403	264
31-May-2019	556	270
26-Mar-2019	769	607
13-Sep-2019	690	211
10-Nov-2018	398	254
...	...	...

日付の表示方法を変更するには、**RVBaseFormattingService** の実装で **FormatNumber** メソッドをオーバーライドする必要があります。Monday, 01 January 2001 のように、曜日と日付を含む日付を作成します。

```
public class UpMediaFormattingService : RVBaseFormattingService
{
    public override string FormatDate(DateTime value, RVDashboardDataType type,
    RVDateFormattingSpec formatting, bool localTimeZone)
    {
        return string.Format("{0:dddd, dd MMMM yyyy}", value);
    }
}
```

また、アプリを再実行してグリッドの可視化を変更すると、更新された日付が表示されます(キャッシュのためにデータの更新が必要な場合があります)。

The screenshot shows the UpMedia dashboard interface. At the top, there is a navigation bar with tabs for Sales, Marketing (which is currently selected), Campaigns, Manufacturing, and Create Dashboard. Below the navigation bar, there is a date filter section with a dropdown set to 'Last 365 days', an 'Add Filter' button, and a '+' button. The main content area displays a grid titled 'Marketing > Actual Spend vs Budget'. The grid has columns for Date, Spend, and Budget, with sorting icons for each. The data in the grid is as follows:

Date	Spend	Budget
Thursday, 14 March 2019	536	341
Thursday, 21 February 2019	448	535
Saturday, 07 September 2019	403	264
Friday, 31 May 2019	556	270
Tuesday, 26 March 2019	769	607
Friday, 13 September 2019	690	211
Saturday, 10 November 2018	398	254
Total	3,663	2,663

## カスタム テーマの作成

### 概要

分析を既存のアプリケーションに埋め込む場合、それらのダッシュボードがアプリのルック アンド フィールと一致することが重要です。そのため、SDK を通じて Reveal ダッシュボードを完全に制御できます。

Reveal で独自のテーマを作成するには、**Theme** プロパティを変更します。

```

var regularFont = new FontFamily(new Uri("pack://application:,,,/[Your
ProjectName];component/[pathToFonts]/"), "./#Verdana Italic");
var boldFont = new FontFamily(new Uri("pack://application:,,,/[Your
ProjectName];component/[pathToFonts]/"), "./#Verdana Bold");
var mediumFont = new FontFamily(new Uri("pack://application:,,,/[Your
ProjectName];component/[pathToFonts]/"), "./#Verdana Bold Italic");

var currentTheme = RevealSdkSettings.Theme;
currentTheme.ChartColors.Clear();
currentTheme.ChartColors.Add(Color.FromRgb(192, 80, 77));
currentTheme.ChartColors.Add(Color.FromRgb(101, 197, 235));
currentTheme.ChartColors.Add(Color.FromRgb(232, 77, 137));

currentTheme.BoldFont = new FontFamily("Gabriola");
currentTheme.MediumFont = new FontFamily("Comic Sans MS");
currentTheme.FontColor = Color.FromRgb(31, 59, 84);
currentTheme.AccentColor = Color.FromRgb(192, 80, 77);
currentTheme.DashboardBackgroundColor = Color.FromRgb(232, 235, 252);

RevealSdkSettings.Theme = currentTheme;

```

[!NOTE] 新しい色のセットを追加するには、まずチャートの色リストのデフォルト値をクリアする必要があります。

画面にダッシュボードまたは別の Reveal コンポーネントがすでに表示されている場合は、適用された変更を表示するために、再度レンダリングする必要があります。

## カスタマイズ可能なテーマ設定

テーマのカスタマイズに使用できる設定は、*RevealTheme()* クラスの一部です。*RevealTheme() class* には、すべてのダッシュボードとアプリの設定と現在の設定値が含まれています。以下のテーブルには、カスタマイズ可能なすべての設定とそれに関連する簡単な説明、タイプ、各設定のデフォルト値があります。

名前	タイプ	説明
<b>ChartColors</b>	List	表示形式でシリーズを示すために使用される色。色の数に制限はありません。すべての色が表示形式で使用されると、Reveal はこれらの色の新しい色合いを自動生成します。これにより、色が重複せず、各値に独自の色が設定されます。
<b>AccentColor</b>	Color	Reveal のデフォルトのアクセント色は、[+ ダッシュボード] ボタンやその他のインタラクティブなアクションで確認することができる青の色合いで、アプリケーションで使用するのと同じアクセント色に一致するように色を変更できます。

名前	タイプ	説明
<b>DashboardBackgroundColor</b>	Color	ダッシュボードの背景色を設定します。 これはメインの背景色です。
<b>VisualizationBackgroundColor</b>	Color	表示形式の背景色を設定します。これは 二番目のは背景色です。
<b>ConditionalFormatting</b>	RVConditionalFormatting	条件付き書式を使用するときに設定でき る境界のデフォルトの色を変更します。
<b>RegularFont</b>	FontFamily	通常のフォントスタイルを設定します。
<b>BoldFont</b>	FontFamily	太字のフォントスタイルを設定します。
<b>MediumFont</b>	FontFamily	中のフォントスタイルを設定します。
<b>FontColor</b>	Color	フォントの色を設定します。
<b>HighlightColor</b>	Color	特定のダッシュボード シナリオ (予測およ び外れ値の統計関数) の強調色を設定しま す。
<b>UseRoundedCorners</b>	bool	(デフォルト) ボタン、ツールチップ、コ ンテナ、表示形式などの隅が丸められて います。false に設定すると、コーナーは 四角になります。

## ビルド済みのテーマ

Reveal SDK には、*Mountain Light*、*Mountain Dark*、*Ocean Light*、*Ocean Dark* の 4 つのビルド済みテーマが付属しています。アプリケーションのデザインに最適なものを設定することも、カスタム テーマのベースとして使用することもできます。

新しいインスタンスを作成して、選択したビルド済みテーマの設定を適用します。

### **Mountain Light** テーマ

```
RevealSdkSettings.Theme = new MountainLightTheme();
```

[!NOTE] Mountain Light には、カスタマイズ可能なすべての設定のデフォルト値が含まれています。

### **Mountain Dark** テーマ

```
RevealSdkSettings.Theme = new MountainDarkTheme();
```

### **Ocean Light** テーマ

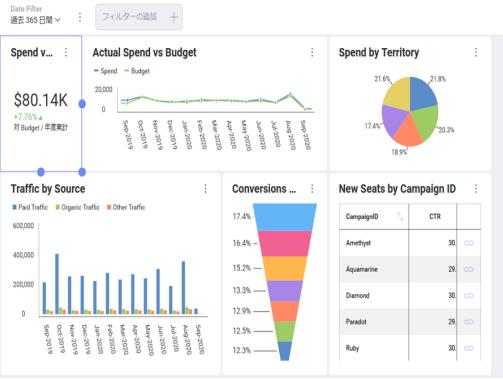
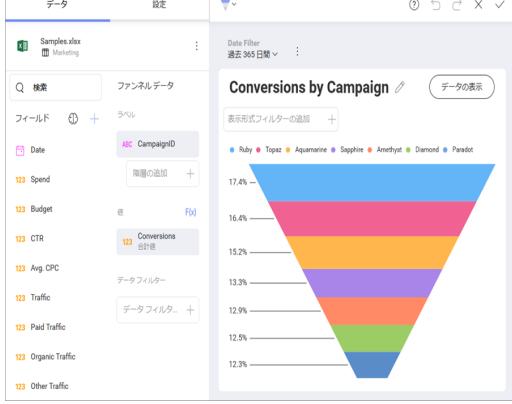
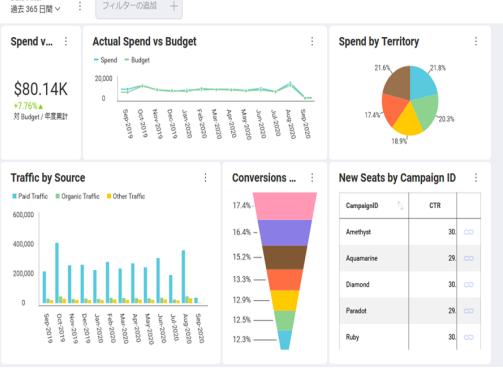
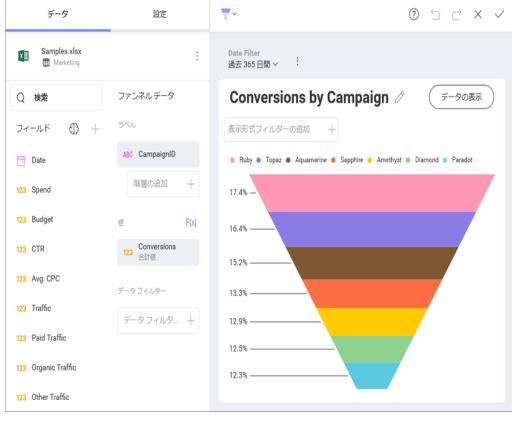
```
RevealSdkSettings.Theme = new OceanLightTheme();
```

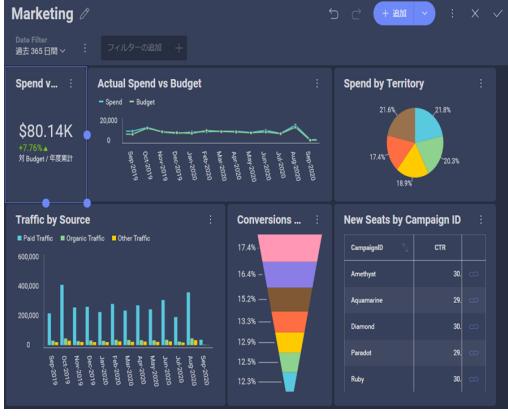
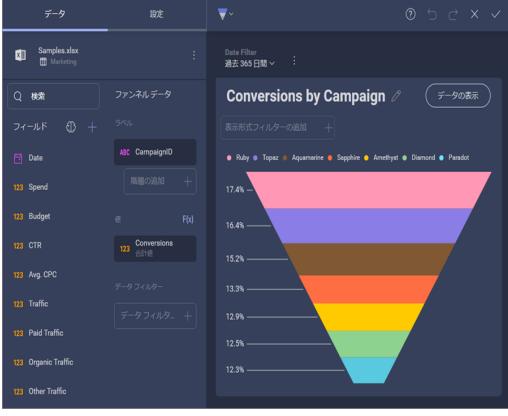
## Ocean Dark テーマ

```
RevealSdkSettings.Theme = new OceanDarkTheme();
```

ビルド済みのテーマはどのように見えますか?

以下は、各ビルド済みテーマが適用されたときの形式エディターとダッシュボードエディターの外観を示すテーブルです。

テーマ	ダッシュボードエディター	形式エディター
Mountain Light (デフォルト?)		
Mountain Dark		
Ocean Light		

テーマ	ダッシュボード エディター	形式エディター
Ocean Dark	 <p>The screenshot shows a Marketing dashboard with the following components:</p> <ul style="list-style-type: none"> <li><b>Actual Spend vs Budget</b>: A line chart comparing actual spend against budget over time. The total spend is \$80.14K.</li> <li><b>Spend by Territory</b>: A pie chart showing the distribution of spend across territories.</li> <li><b>Traffic by Source</b>: A bar chart showing traffic volume from Paid Traffic, Organic Traffic, and Other Traffic.</li> <li><b>Conversions ...</b>: A funnel chart showing conversion rates for different campaign IDs.</li> <li><b>New Seats by Campaign ID</b>: A table showing new seat counts for each campaign ID.</li> </ul>	 <p>The screenshot shows a Conversions by Campaign report with the following details:</p> <ul style="list-style-type: none"> <li><b>Data Filter</b>: Set to "過去 365 日間".</li> <li><b>Fields</b> (left sidebar):       <ul style="list-style-type: none"> <li>Marketing</li> <li>Date</li> <li>Spend</li> <li>Budget</li> <li>CTR</li> <li>Avg CPC</li> <li>Traffic</li> <li>Paid Traffic</li> <li>Organic Traffic</li> <li>Other Traffic</li> </ul> </li> <li><b>Report Content</b> (right sidebar):       <ul style="list-style-type: none"> <li><b>Conversions by Campaign</b>: Stacked area chart showing conversion rates for various campaigns.</li> </ul> </li> </ul>

# ユーザー クリックイベントの処理

## 概要

SDK は、ユーザーが可視化内のデータを含むセルをクリックしたときに処理できます。たとえば独自のナビゲーションを提供したり、アプリの既存の選択を変更したりする場合などに非常に便利です。

## コード

**VisualizationDataPointClicked** イベントに登録することで、ユーザー クリックイベントを処理できます。

```
//VisualizationDataPointClicked イベントに添付します。  
revealView.VisualizationDataPointClicked +=  
    RevealView_VisualizationDataPointClicked;
```

コールバック関数では、クリックの場所に関する情報を受け取ります。

- クリックされた可視化の名前。
- クリックされたセルの値 (値、書式設定された値、列の名前を含む)。
- 同じセルの残りの値。

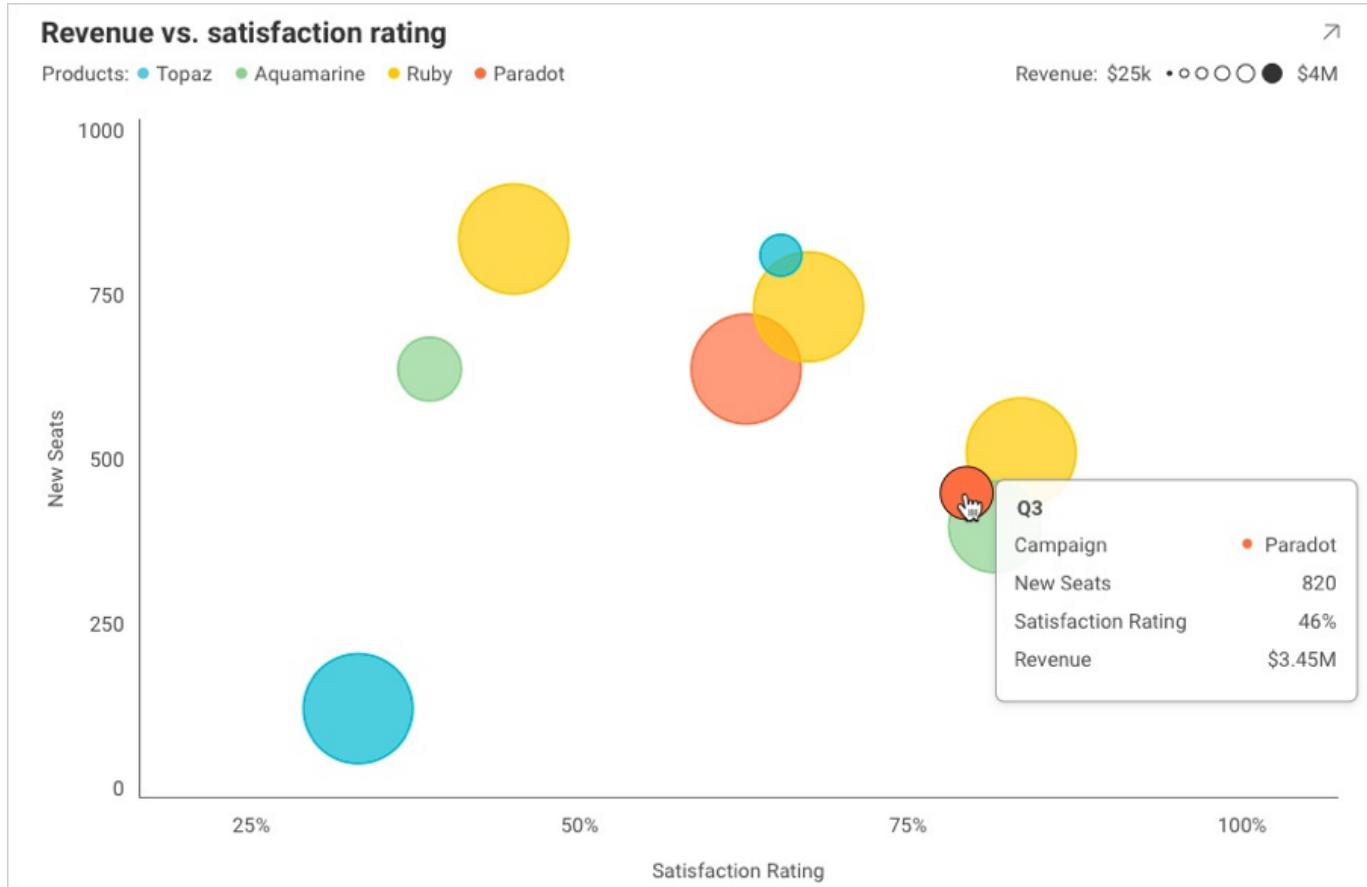
```
private void RevealView_VisualizationDataPointClicked(object sender,  
    VisualizationClickedEventArgs e)  
{  
    var vizTitle = e.Visualization.Title;  
    var column = e.Cell.ColumnName;  
    var formattedValue = e.Cell.FormattedValue;  
    var value = e.Cell.Value;  
    foreach (var c in e.Row)  
    {  
        var cValue = c.Value;  
        //列 c.ColumnNAme の値を使用します。  
    }  
}
```

アプリケーションで選択したカスタマーを変更する場合は、カスタマー ID などの特定の属性を検索するためにはセル内の残りの値を使用できます。カスタマーの売上高のように、ユーザーが別のセルをクリックした場合も必要な情報を取得できます。

# ツールチップの作業

## 概要

エンドユーザーが表示形式でシリーズの上をホバーするか、シリーズをクリックするたびにトリガーされるイベントがあります(以下の画像を参照)。このイベントは **revealView TooltipShowing** と呼ばれ、表示形式でツールチップを表示する方法をより柔軟に指定できます。



## 一般的なユースケース

Tooltip イベントをキャンセルするか、もしくはユーザーに表示される項目を変更できます。最も一般的な例:

- ツールチップをすべて無効にするか、特定の表示形式でのみ表示したい場合。
- ビューアーに役立つ RevealView コンポーネント以外のツールチップにデータを表示したい場合。

このイベントは、グリッドやゲージなどのツールチップをサポートしない表示形式ではトリガーされないことに注意してください。

## コード例

以下のコード スニペットでは、表示形式のツールチップを無効にし、エンドユーザーがこの表示形式をホバーまたはクリックするときにイベント引数から追加情報を取得する方法を示します。

```
private void RevealView_TooltipShowing(object sender, TooltipShowingEventArgs e)
```

```
{  
    if (e.Visualization.Title == "NoNeedForToolips")  
    {  
        e.Cancel = true;  
    }  
    Debug.WriteLine($"TooltipShowing: Visualization: {e.Visualization.Title},  
Cell: {e.Cell}, Row: {e.Row}");  
}
```

イベント引数には、ホバーまたはクリックされている表示形式、ホバーまたはクリックされたデータの正確なセル、このセルの行全体 (他の列の情報が必要な場合)、および Cancel ブール値に関する情報が含まれています。

# ユーザーインターフェイス要素の表示/非表示

**RevealView** コンポーネントを使用して、エンドユーザーに対するさまざまな機能または UI 要素を有効または無効にすることができます。使用可能なプロパティの多くはブール型で簡単に使用できますが、その他のプロパティはそれほど多くありません。

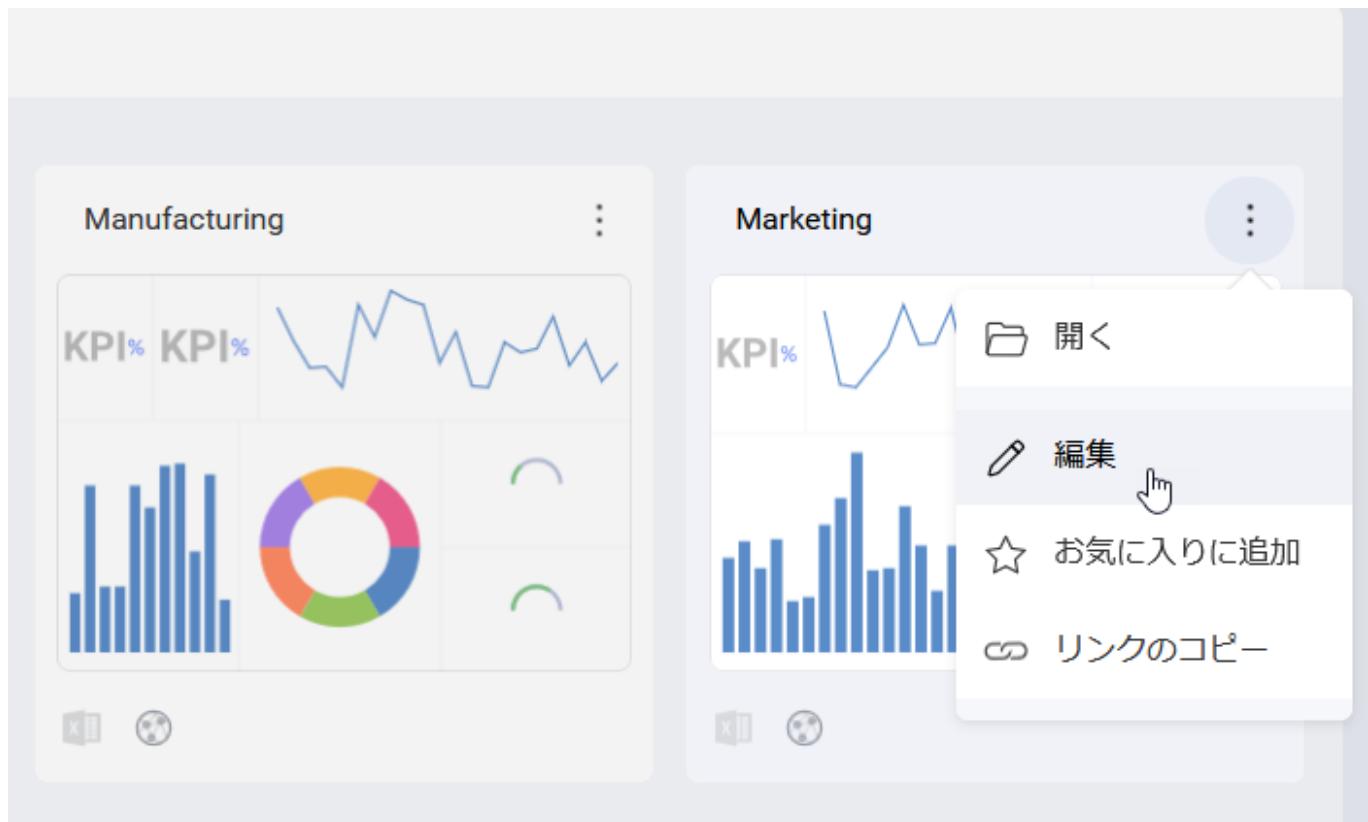
以下に作成する `revealView` インスタンスは、このトピックのすべてのコード スニペットで使用されます。

```
var revealView = new RevealView();
```

すべてのプロパティは、初期化時に **RevealView** によって読み取られ、その値に基づいて、ユーザーにさまざまな機能または UI 要素を表示または非表示にします。

## CanEdit

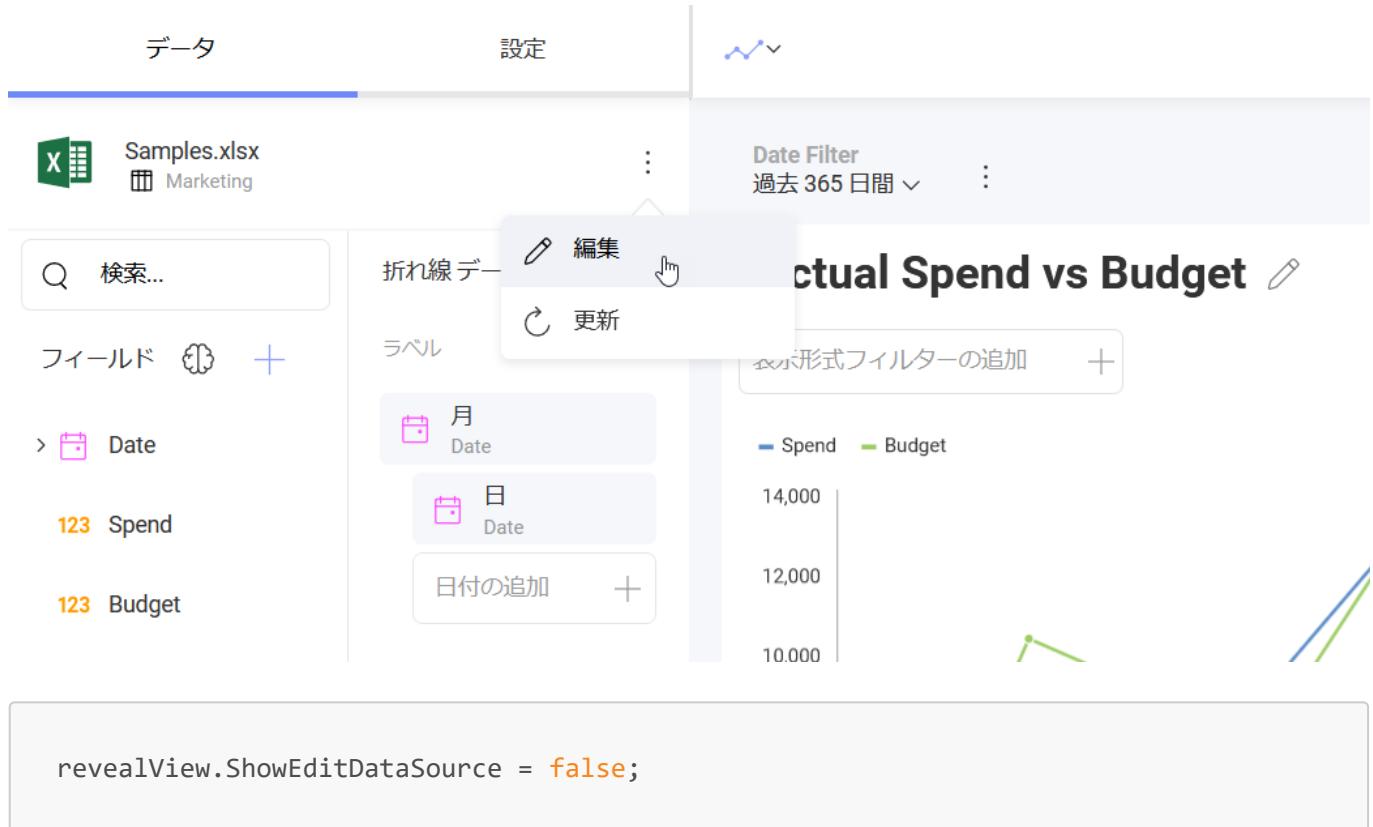
このプロパティは、ダッシュボードを編集するユーザー機能を無効にするために使用できます。



```
revealView.CanEdit = false;
```

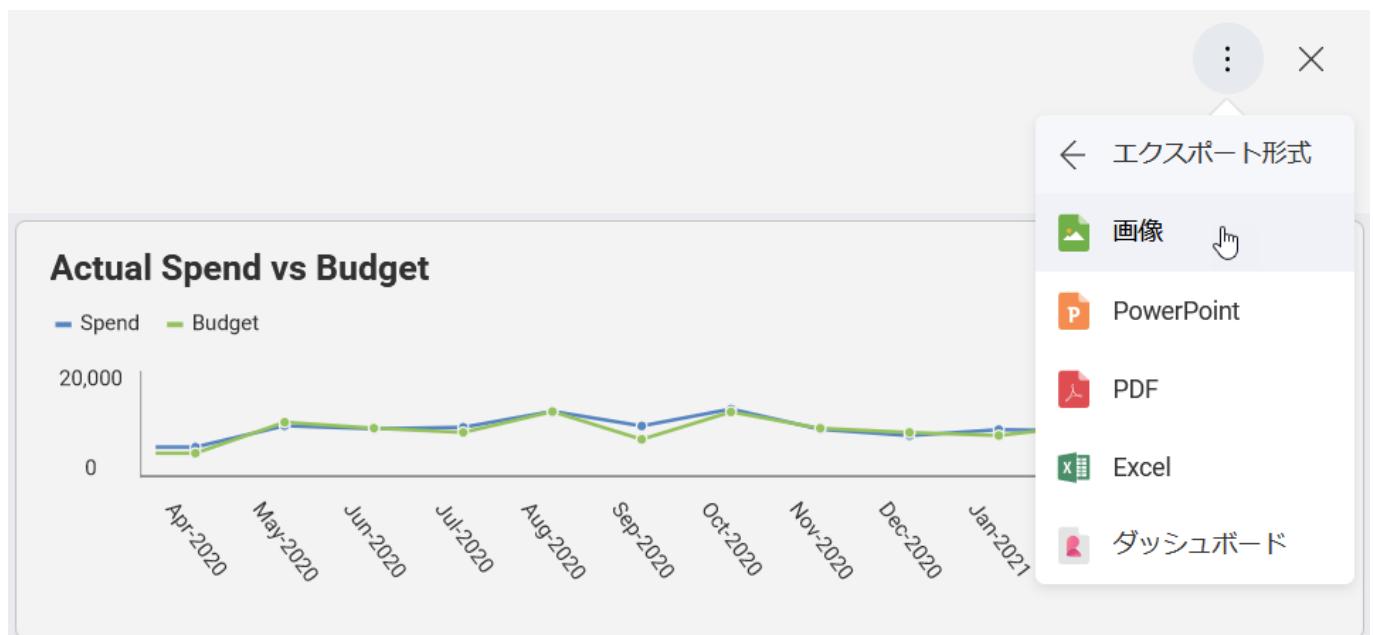
## ShowEditDataSource

このプロパティを使用して、ダッシュボードデータソースの編集を無効にすることができます。



## ShowExportImage

このプロパティを使用して、ダッシュボードの画像へのエクスポートを無効にすることができます。



```
revealView.ShowExportImage = false;
```

## ShowExportToPowerpoint

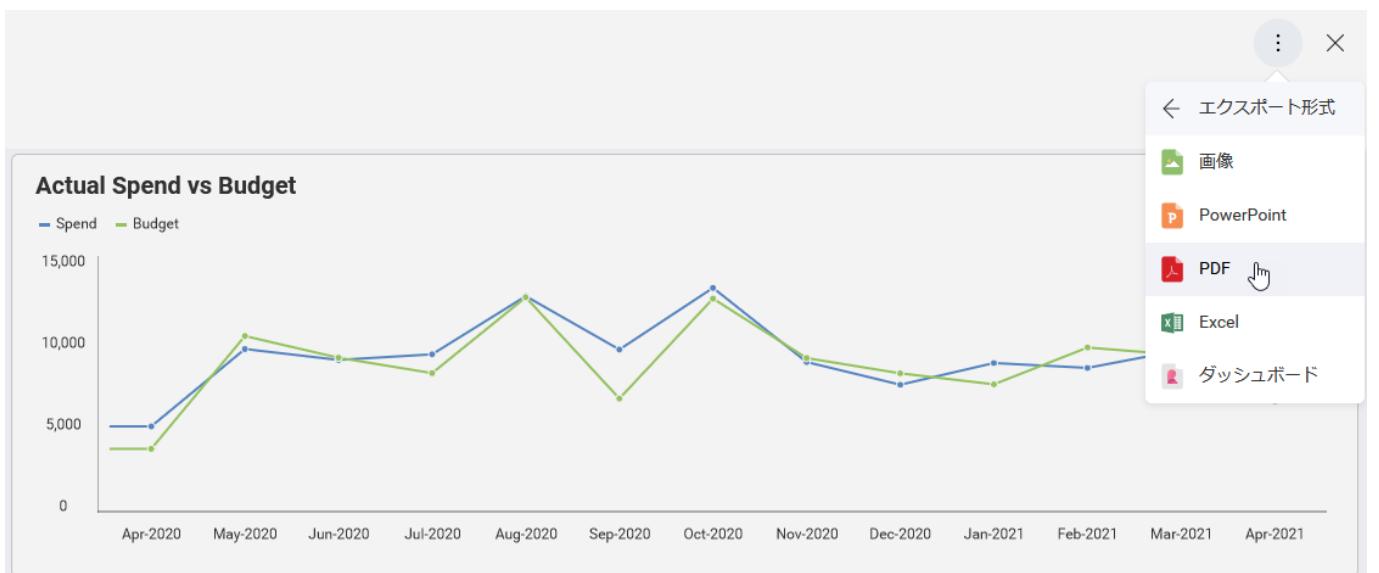
このプロパティを使用して、ダッシュボードの PowerPoint へのエクスポートを無効にすることができます。



```
revealView.ShowExportToPowerpoint = false;
```

## ShowExportToPDF

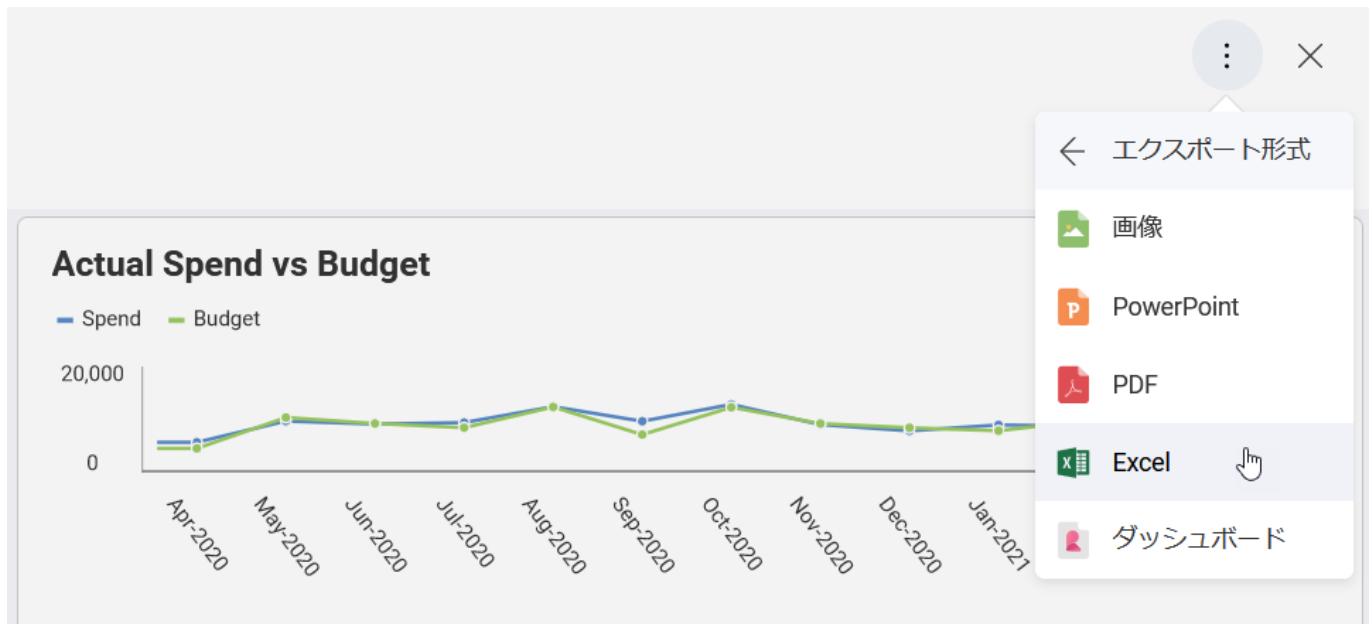
このプロパティを使用して、ダッシュボードの PDF へのエクスポートを無効にすることができます。



```
revealView.ShowExportToPDF = false;
```

## ShowExportToExcel

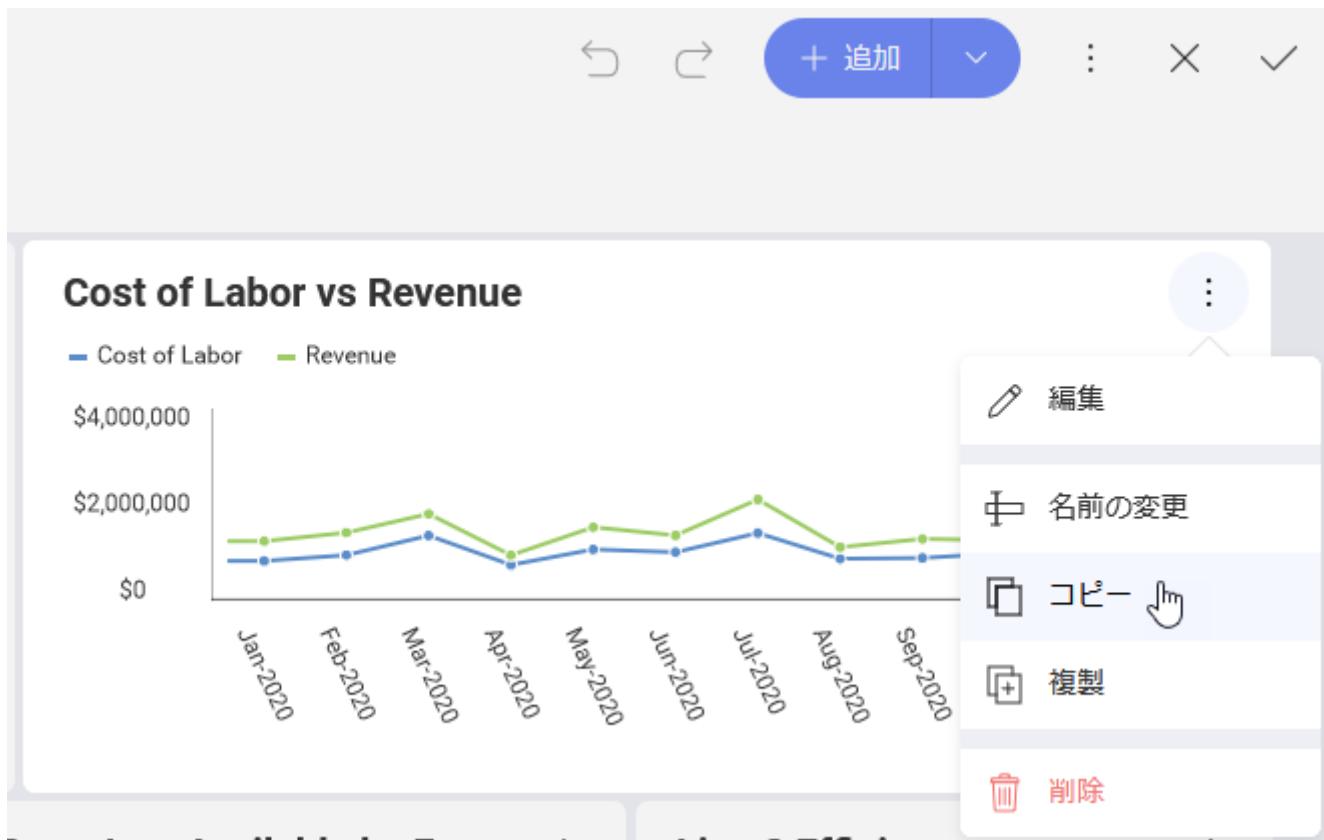
このプロパティを使用して、ダッシュボードの Excel へのエクスポートを無効にすることができます。



```
revealView.ShowExportToExcel = false;
```

## CanCopyVisualization

このプロパティは、表示形式をコピーし、後で現在のダッシュボードまたは別のダッシュボードに貼り付ける機能を無効にするために使用できます。



```
revealView.CanCopyVisualization = false;
```

## CanDuplicateVisualization

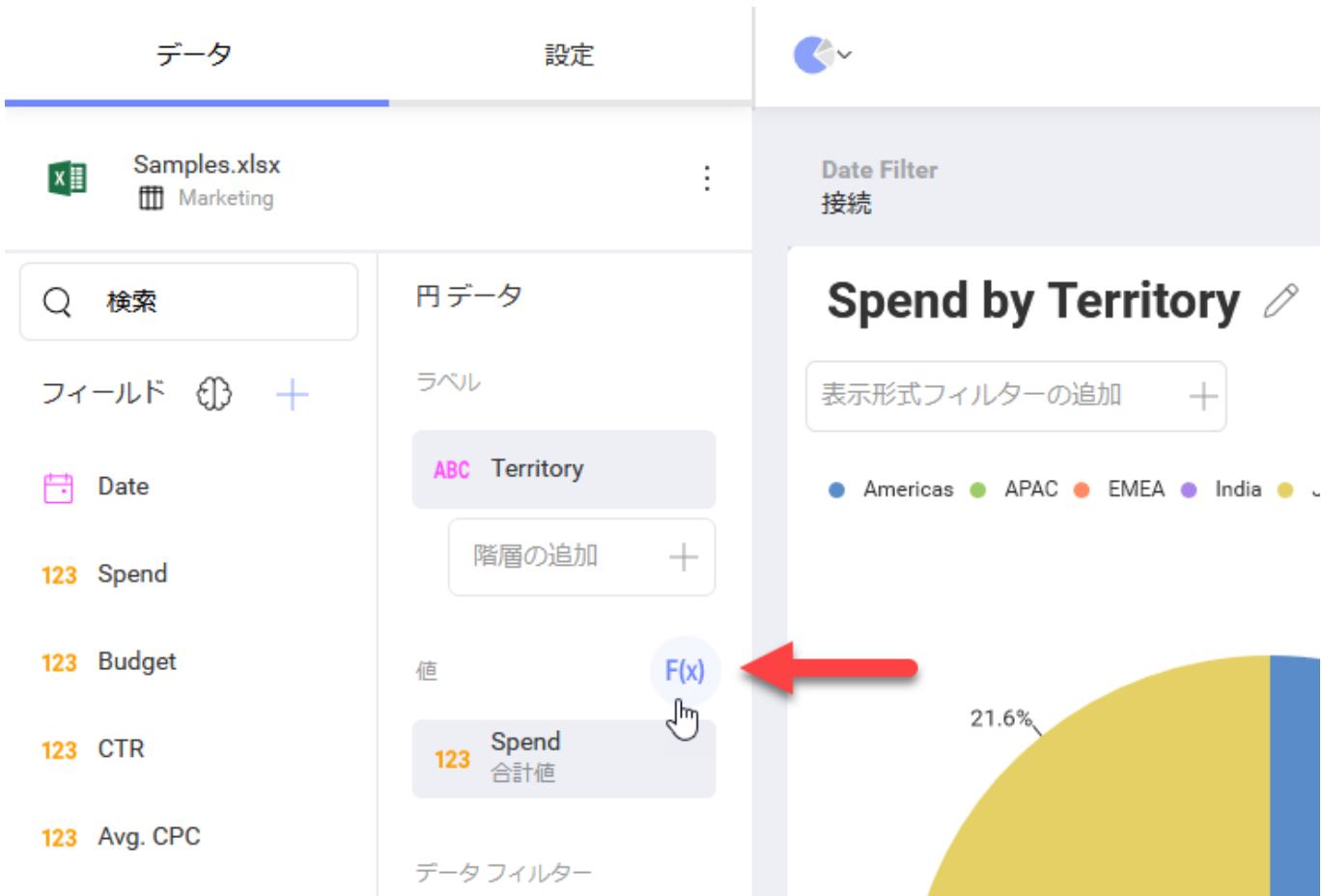
このプロパティは、現在のダッシュボードで表示形式を複製する機能を無効にするために使用できます。

The screenshot shows a Power BI dashboard with a line chart titled "Cost of Labor vs Revenue". The chart tracks two metrics over time from January to September 2020: "Cost of Labor" (blue line) and "Revenue" (green line). A context menu is open on the right side of the chart, listing options: "編集" (Edit), "名前の変更" (Change name), "コピー" (Copy), "複製" (Duplicate), and "削除" (Delete). The "複製" (Duplicate) option is highlighted with a cursor icon.

```
revealView.CanDuplicateVisualization = false;
```

## CanAddPostCalculatedFields

このプロパティを使用して、現在のダッシュボードに新しい事後計算フィールドを追加する機能を無効にできます。

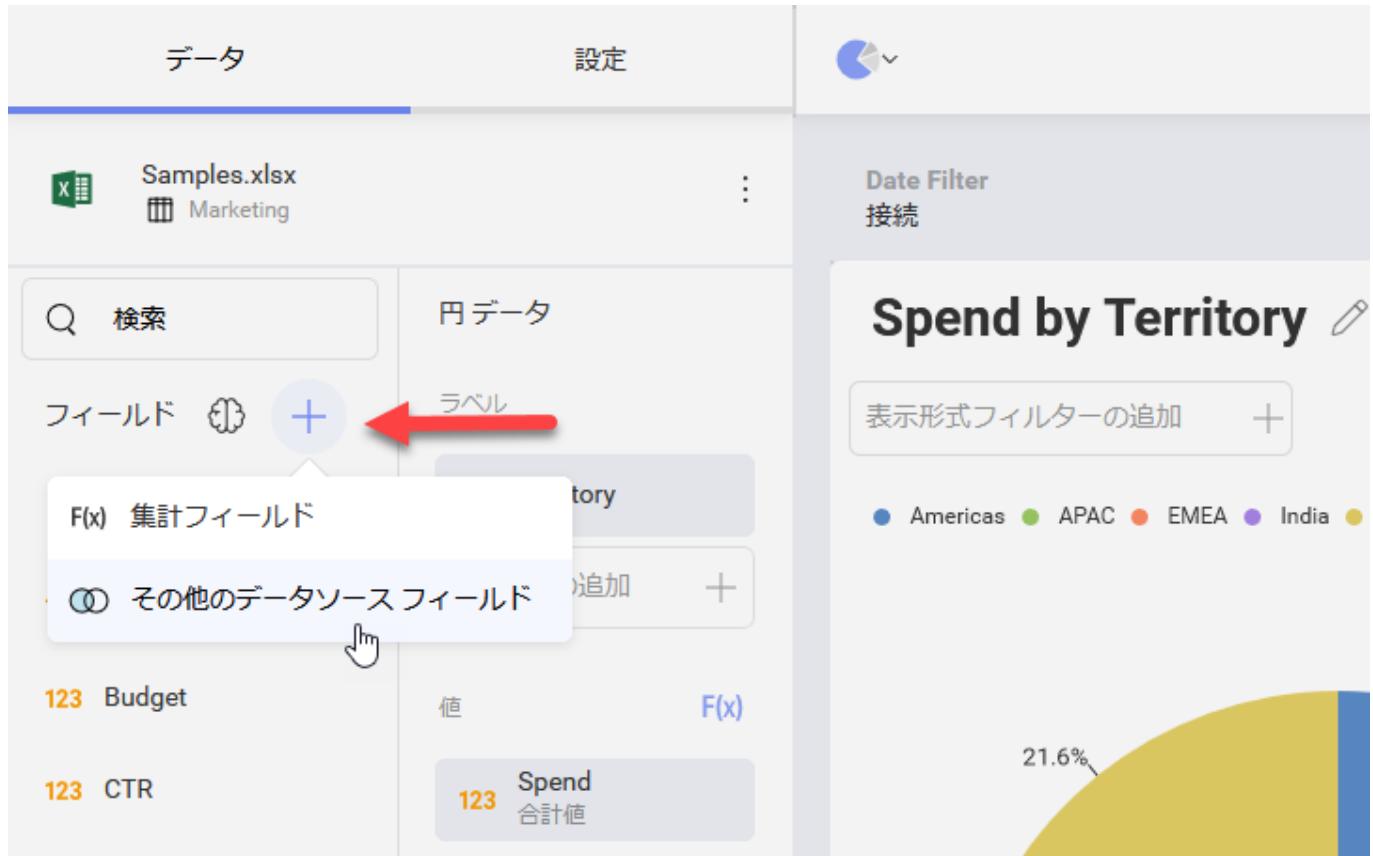


事後計算フィールドはデータセットの新しいフィールドで、すでに集計された値に数式を適用して作成されます。 詳細については、[Reveal ヘルプ](#)をご覧ください。

```
revealView.CanAddPostCalculatedFields = false;
```

## CanAddCalculatedFields

このプロパティを使用して、現在のダッシュボードに新しい事前計算フィールドを追加する機能を無効にできます。



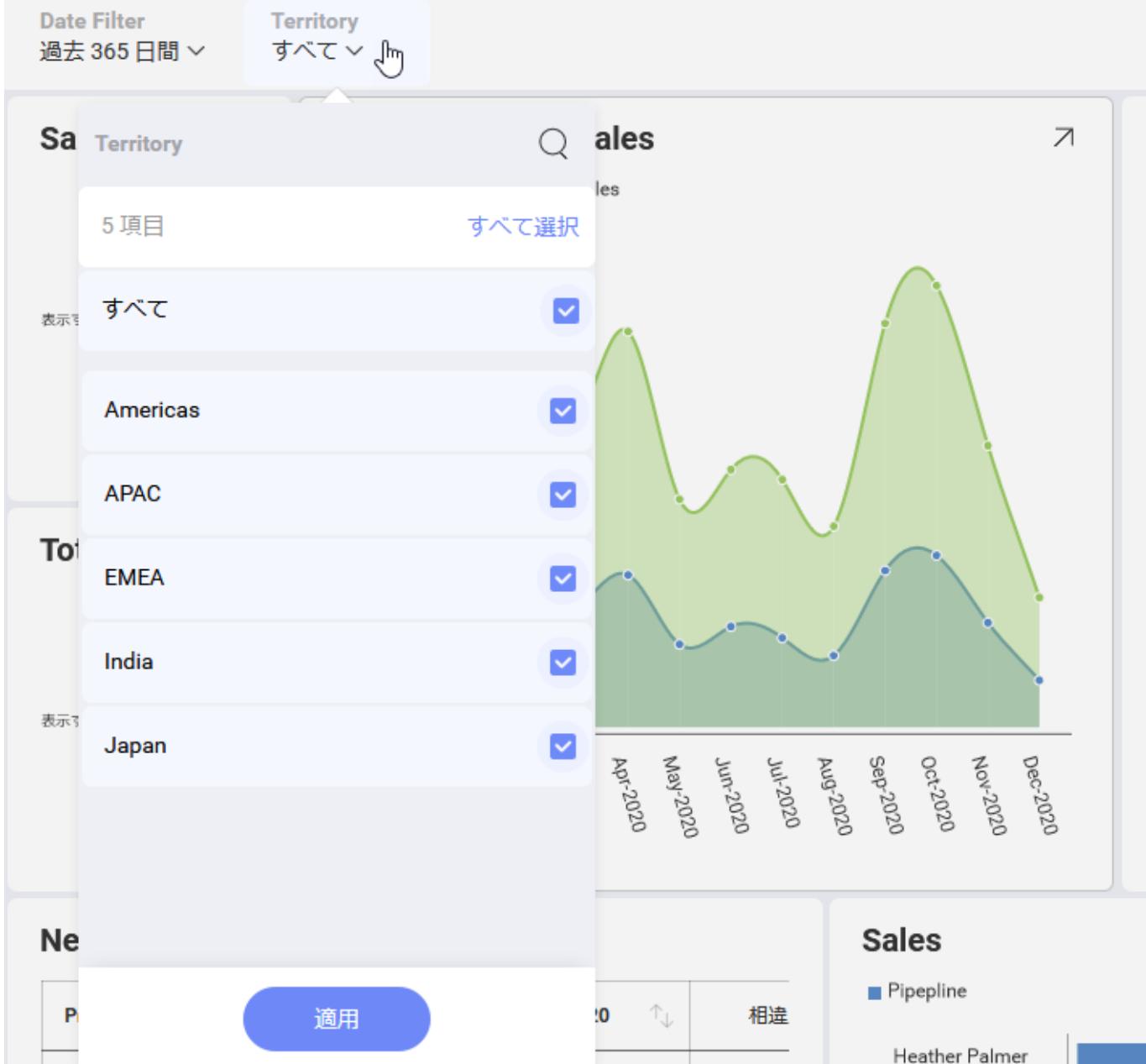
事前計算フィールドはデータセットの新しいフィールドで、データ エディター集計を実行する前に評価されます。 詳細については、[Reveal ヘルプ](#)をご覧ください。

```
revealView.CanAddCalculatedFields = false;
```

## ShowFilters

このプロパティは、ユーザーにダッシュボード フィルター UI を表示または非表示にするために使用できます。

# Sales



ダッシュボード フィルターを使用すると、ダッシュボードの全ての表示形式のコンテンツを一度にフィルター適用できます。

```
revealView.ShowFilters = false;
```

## CanAddDashboardFilter

このプロパティを使用して、[ダッシュボード フィルターの追加] メニュー項目を表示または非表示にすることができます。

# Manufacturing

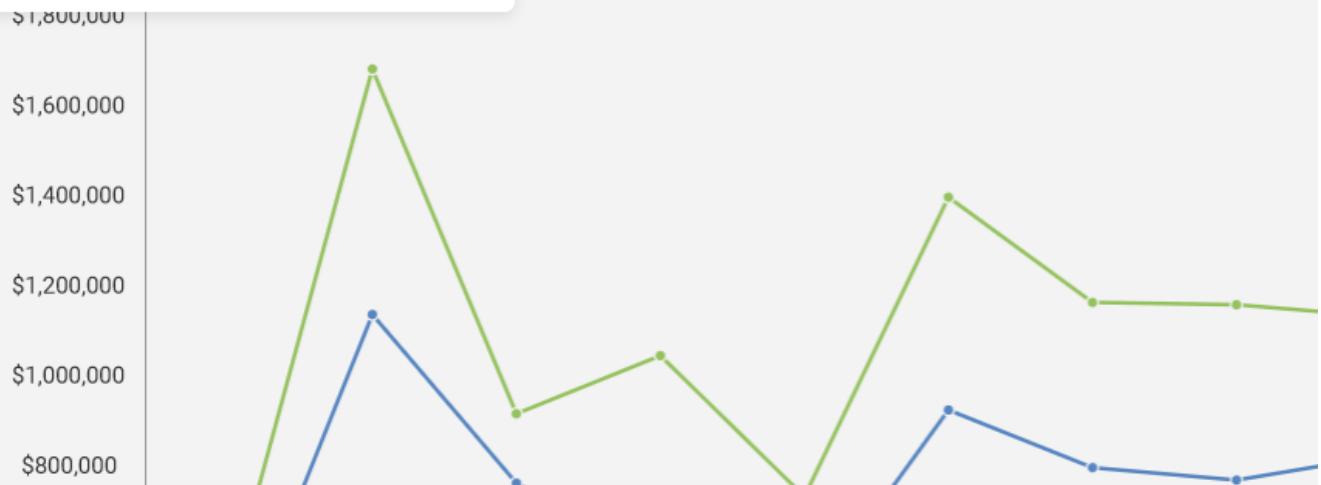


フィルターの追加 +

▽ ダッシュボード フィルターの追加



▽ 日付 フィルターの追加



```
revealView.CanAddDashboardFilter = false;
```

## CanAddDateFilter

このプロパティを使用して、[日付 フィルターの追加] メニュー項目を表示または非表示にすることができます。

# Manufacturing



フィルターの追加 +

▽ ダッシュボード フィルターの追加



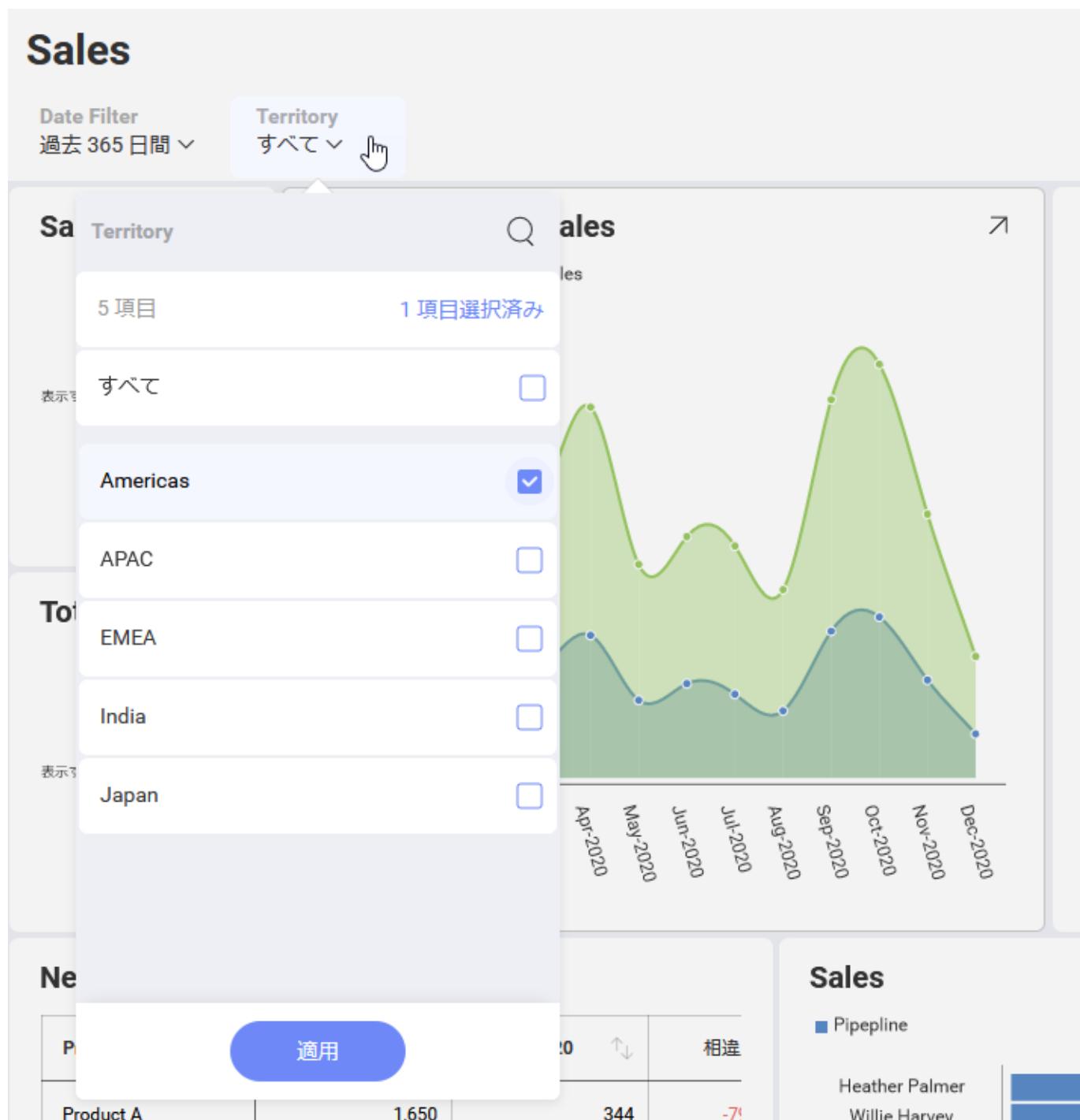
▽ 日付 フィルターの追加



```
revealView.CanAddDateFilter = false;
```

## 選択済みフィルター

ダッシュボードを読み込む時に既存のダッシュボード フィルターから最初に選択される値を指定できます。

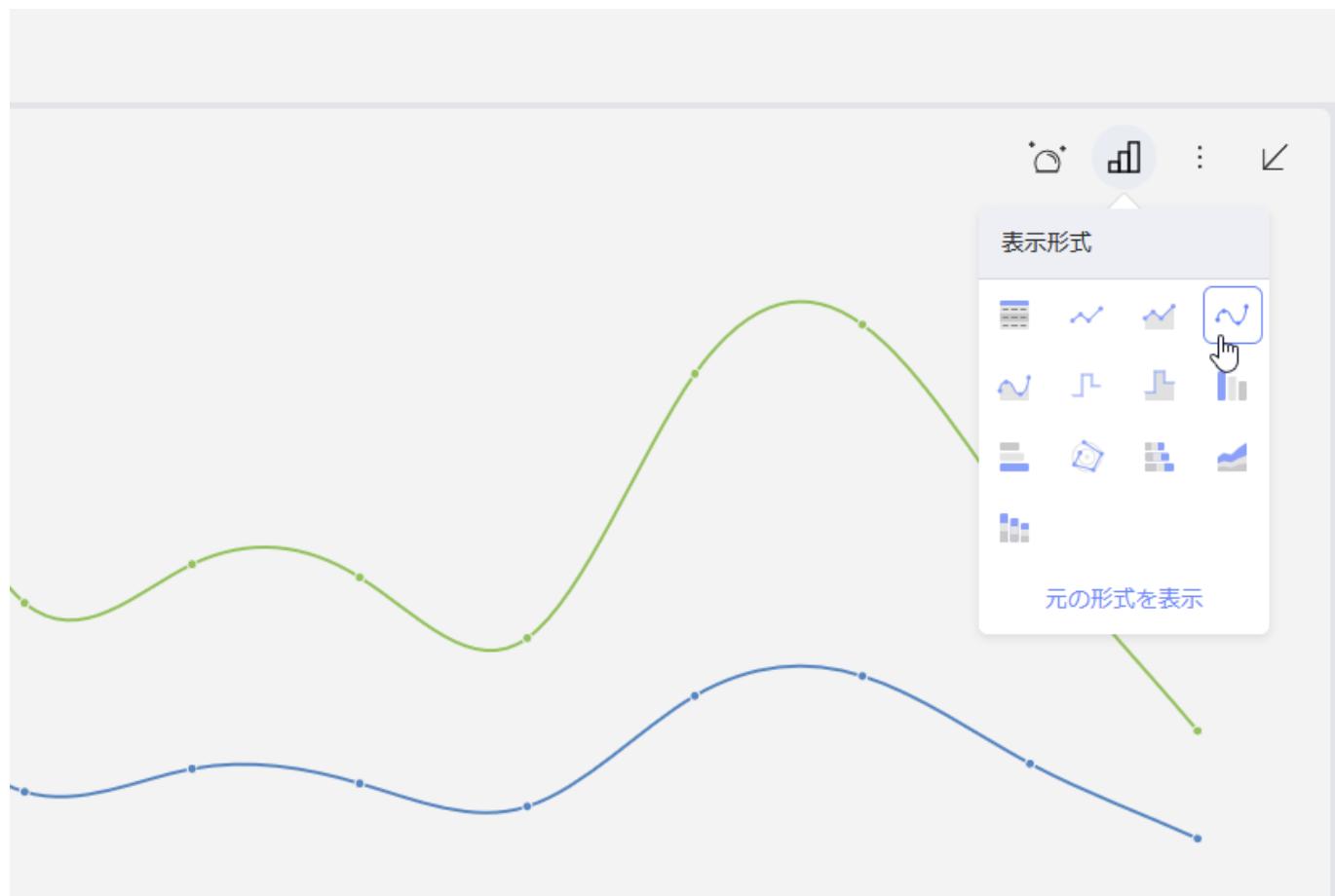


次のコードスニペットは、ダッシュボードを読み込む方法を示し、選択された「Territory」値を「Americas」に設定します。したがって、ダッシュボードには「Americas」でフィルタリングされたデータが表示されます。

```
var dashboard = new RVDashboard(path);
dashboard.filters.GetByTitle("Territory").selectedValues = new List<object>() {
    "Americas" };
revealView.Dashboard = dashboard;
```

## AvailableChartTypes

このプロパティは、ユーザーが使用できる表示形式タイプをフィルターするために使用できます。



たとえば、以下のように表示形式を追加または削除できます。

```
revealView.AvailableChartTypes.Add(RVChartType.BulletGraph);
revealView.AvailableChartTypes.Remove(RVChartType.Choropleth);
```

さらに、使用可能な表示形式のみを含む新しいリストを作成できます。

```
revealView.AvailableChartTypes = new List<RVChartType>() {
    RVChartType.BulletGraph, RVChartType.Choropleth };
```

# 初期フィルター選択の設定

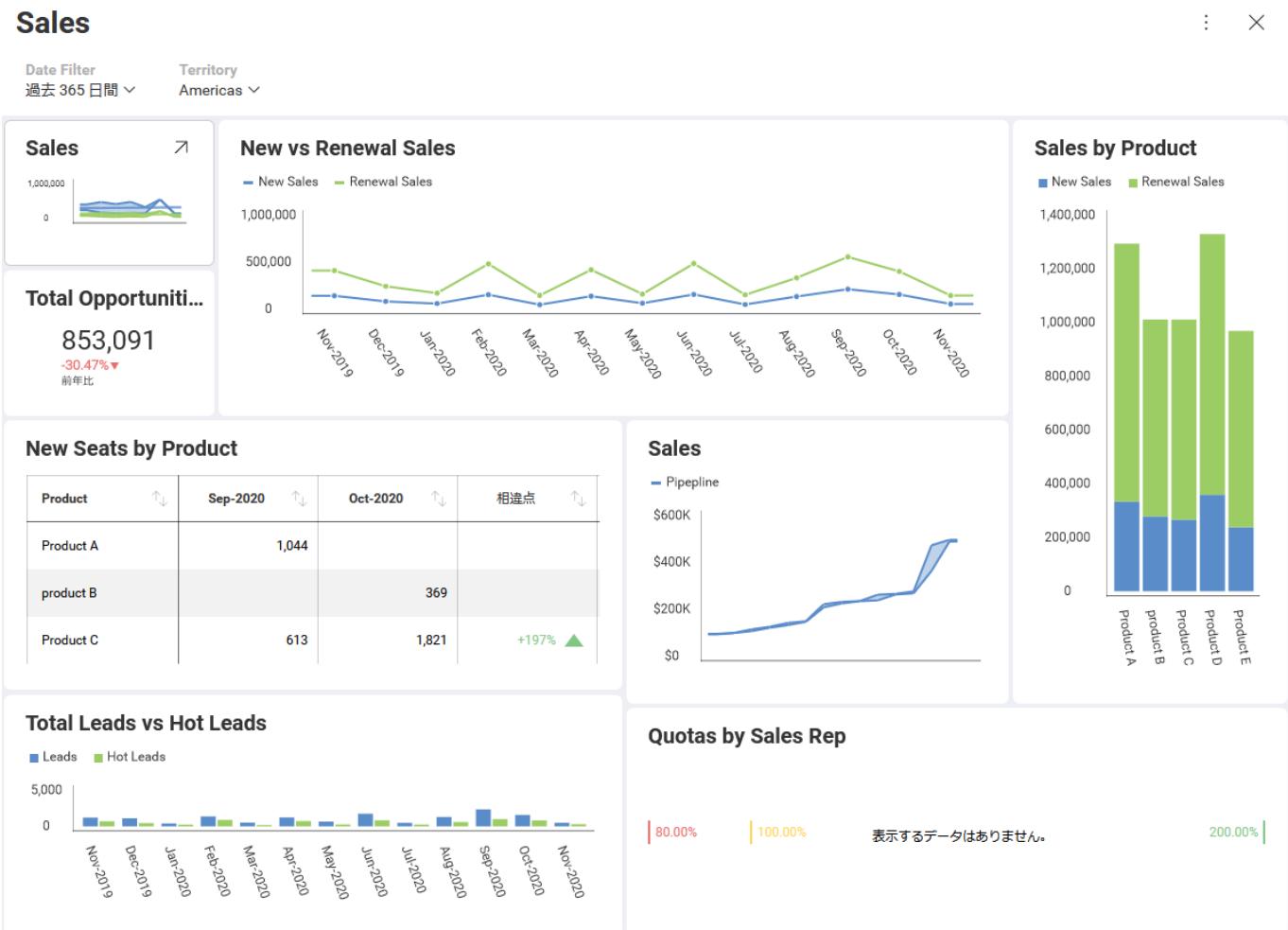
## 概要

すでに適用されているフィルターを含むダッシュボードを表示したい場合は、ダッシュボード フィルターですべてのウィジェットのコンテンツを一度にスライスすると非常に便利です。これにより、SDK を使用して、すべてのダッシュボードのウィジェットに対してコンテキスト内に留まる最初のダッシュボード フィルター選択を設定できます。

## 例の詳細

この例では、Sales データを表示するダッシュボードには以下のフィルターがあります。

- 定期間 (過去 365 日、年度累計など)。
- 地域 (南北アメリカ、ヨーロッパ、アジアなど)



## コード例

この場合、初期フィルター選択を次のように設定します。

- [年度累計] ([過去 365 日]) の代わりに、このダッシュボードのデフォルト設定)。
- 現在のユーザーの地域に関連付けられている売上。

初期化プロセスの一部として、そしてダッシュボードがロードされたら、ダッシュボード内のフィルターのリストを取得し、これらのフィルターを使用して **RevealView** で最初に選択された値を設定できます。

```
var revealView = new RevealView();

using (var fileStream = File.OpenRead(path))
{
    var dashboard = new RVDashboard(fileStream);

    dashboard.DateFilter = new RVDateDashboardFilter(RVDateFilterType.YearToDate);
    dashboard.Filters.GetByTitle("Territory").SelectedValues = new List<object>()
    { CurrentUser.Territory };

    revealView.Dashboard = dashboard;
}
```

[!NOTE] 上記のコードは、**CurrentUser.Territory** が現在のユーザーの地域を返すことを前提としています。

## フィルターを非表示にする

ユーザーが自分の地域以外のデータにアクセスしたくない場合があります。このような場合、ダッシュボード フィルターを含むパネルを非表示にするように **RevealView** オブジェクトを設定することで、フィルターへのアクセスを制限できます。

```
revealView.ShowFilters = false;
```

この設定では、関連する地域のデータのみを表示するようにユーザーを制限します。

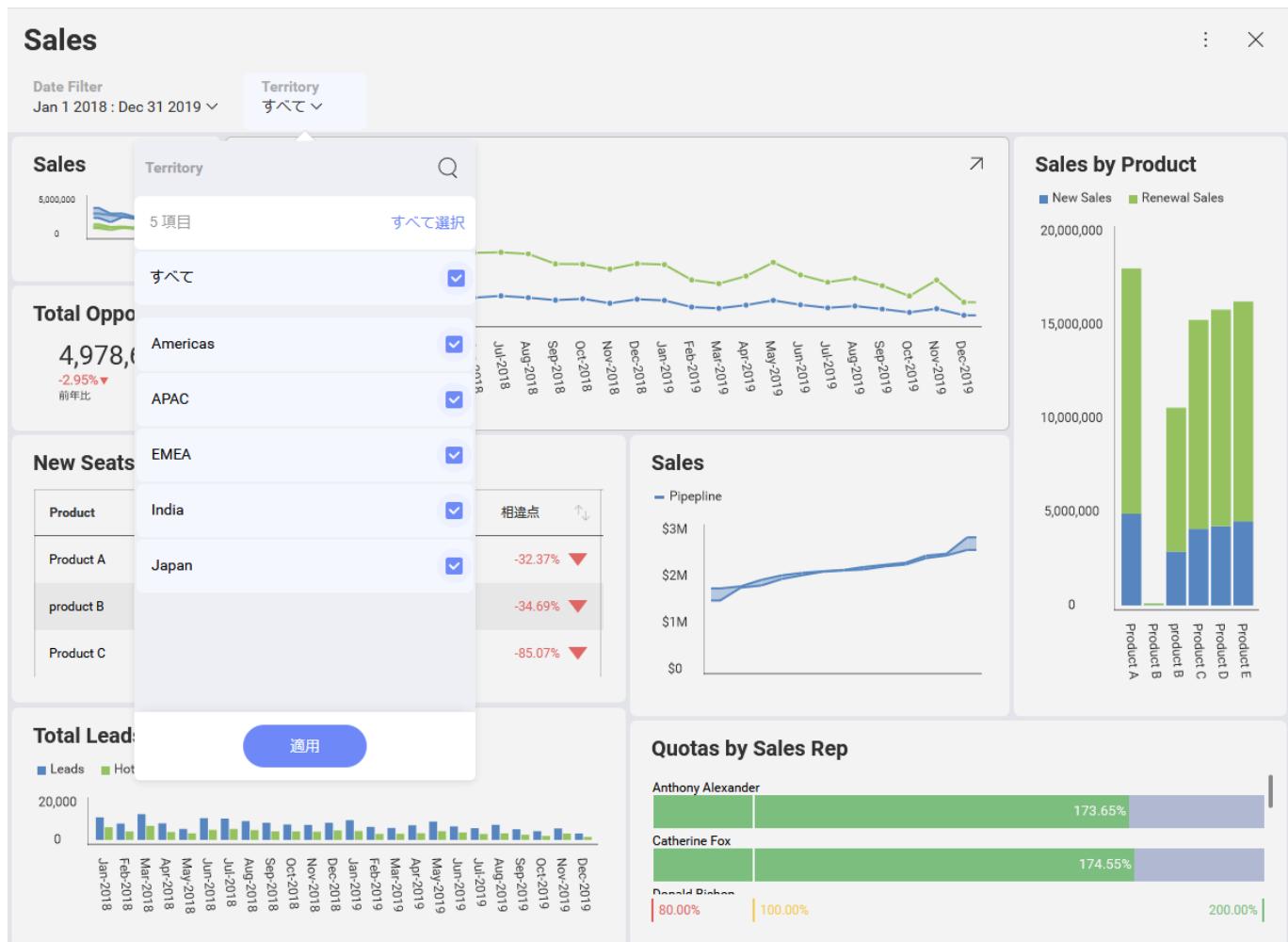
最後に、それでもユーザーに日付フィルターの選択を変更させたい場合は、[動的フィルター選択の設定](#) トップをご覧ください。ユーザーが日付フィルターを変更できるようにするための独自の UI を作成する方法についての情報があります。

# 動的フィルター選択の設定

## 概要

アプリケーションにカスタム UI を統合し、選択した値の一覧をユーザーに表示する際に、そのユーザー選択をダッシュボードのフィルターと同期させたりすることもできます。

たとえば、現在の地域に基づいて数字を変更する Sales ダッシュボードと、地域を選択するためのカスタム UI を作成する場合、ユーザーが選択を変更後、Sales ダッシュボードにその変更を反映させます。ほとんどの場合、ダッシュボードに通常表示されているフィルター選択を非表示にします。これにより、ユーザーが画面領域を変更する 2 つの異なる方法を混同することはありません。



次のコードスニペットでは、説明したシナリオを実現する方法について詳しく説明しています。

```
private void Americas_Click(object sender, RoutedEventArgs e)
{
    revealView.Dashboard.Filters.GetByTitle("Territory").SelectedValues = new
    List<object>() { "Americas" };
}
private void APAC_Click(object sender, RoutedEventArgs e)
{
    revealView.Dashboard.Filters.GetByTitle("Territory").SelectedValues = new
```

```
List<object>() { "APAC" };
```

上記でアメリカと APAC の間で選択した地域を変更するために 2 つのボタンが追加されました。コードにはクリックハンドラーのみが含まれています。

**SetFilterSelectedValues** を使用して新しい地域名を設定するだけで、セレクターで選択が変更されたときにも同じことができます。

## 動的リストの使用

アメリカ大陸、アジア太平洋地域、インドなどの地域は時間の経過とともに変化しませんが、他の値の一覧は変化する可能性があります。この場合、新しい地域がリストに追加されても、新しいボタンは自動的に追加されません。

**RevealUtility.GetFilterValues** を使用して、特定のフィルター値の一覧を取得できます。この場合、次の呼び出しへ territories 変数に **RVFilterValue** オブジェクトを含む配列を残し、その後すべての領域を持つ ComboBox の項目のリストをロードします。

```
using (var stream = File.OpenRead(@"..\..\Sales.rdash"))
{
    var dashboard = new RVDashboard(stream);

    var filterValues = await
        dashboard.Filters.GetByTitle("Territory").GetFilterValuesAsync();
    var territories = filterValues.ToList();

    revealView.Dashboard = dashboard;

    foreach (var t in territories)
    {
        cmbTerritories.Items.Add(t);
    }
    cmbTerritories.SelectionChanged += CmbTerritories_SelectionChanged;
}
```

その後、**RVFilterValue** の **label** 属性を使用して地域の名前を表示し、**Value** 属性を使用してフィルターに選択を設定できます。

次のコードスニペットは、コンボボックスの選択変更イベントを処理してダッシュボードのフィルターを更新する方法を示しています。

```
private void CmbTerritories_SelectionChanged(object sender,
    SelectionChangedEventArgs e)
{
    var selectedItems = new List<object>();
    var filterValue = cmbTerritories.SelectedItem as RVFilterValue;
```

```
if (filterValue != null)
{
    selectedItems.Add(filterValue.Value);
}

await dashboard.Filters.GetByTitle("Territory").SelectedItems = selectedItems;
}
```

## 日付フィルターでの作業

日付フィルターは、データが関連付けられていない特別な種類のフィルターです。選択できる値の一覧はありませんが、実際には時間間隔(日付 A から日付 B まで)を選択します。Year To Date、Previous Month などの定義済みフィルタのリストから選択、または任意の範囲(2019年1月12日から2019年1月30日まで)を指定できます。

### 定義済みフィルターでの作業

定義済みフィルターの1つを設定するには、次のようなコードを使用できます。

```
revealView.Dashboard.DateFilter = new
RVDateDashboardFilter(RVDateFilterType.YearToDate);
```

すべての定義済み日付フィルターのリストが必要な場合は、API リファレンスの **RVDateFilterType** を参照してください。

### カスタムな範囲での作業

たとえば過去15日間のカスタム範囲を設定する場合、以下のようなコードを使用できます。

```
revealView.Dashboard.DateFilter =
new RVDateDashboardFilter(
RVDateFilterType.CustomRange,
new RVDateRange(DateTime.UtcNow.AddDays(-15), DateTime.UtcNow)
));
```

SDKとともに配布されている UpMedia WPF アプリケーションの UpMedia に、**Sales.xaml.cs** の実用的な例があります。このサンプルビューには、日付範囲セレクタと、トグルボタンのリストとして表示される地域の2のカスタムフィルター コンポーネントが含まれています。

## デスクトップ .NET API リファレンス

ここでは、Reveal SDK、特に Desktop .NET API に関する詳しい情報を見つけることができます。完全なリファレンスについては、[このリンク](#)をご覧ください。

### 最も一般的に使用されるクラスとインターフェース

## **SDK の主な概念と機能:**

[RevealView クラス](#)

[RVDashboard クラス](#)

[RevealSdkSettings クラス](#)

## **データ ソース**

[IRVDataSourceProvider インターフェイス](#)

[RevealDataSources クラス](#)

[RVSqIIServerDataSource](#)

[RVSqIIServerDataSourceItem](#)

[RVRESTDataSource](#)

[RVRESTDataSourceItem](#)

[RVJsonDataSource](#)

[RVJsonDataSourceItem](#)

[IRVDataProvider](#)

[RVInMemoryData](#)

[RVInMemoryDataSource](#)

[RVInMemoryDataSourceItem](#)

## **認証**

[IRVAuthenticationProvider](#)

[IRVDataSourceCredential](#)

[RVBearerTokenDataSourceCredential](#)

[RVUsernamePasswordDataSourceCredential](#)

## **ローカライズ**

[IRVLocalizationProvider](#)

[IRVLocalizationService](#)

## **ダッシュボードの書式設定**

[IRVFormattingProvider](#)

[RVBaseFormattingService](#)

# 既知の問題

---

- ウェブとデスクトップ
- デスクトップのみ

## ウェブとデスクトップ

- NuGet パッケージを使用する際に、Reveal SDK のライセンス (SDK インストーラーに有効なキーを入力) 後もウォーターマークが表示されます。

**回避策:** プロジェクトから NuGet パッケージをアンインストールし、NuGet のキャッシュをクリアして、パッケージを再度インストールしてください。NuGet のすべてのキャッシュをクリアしたくない場合、キャッシュした場所を検索し、Infragistics Reveal アイテムのみをクリアできます。場所は NuGet のバージョンと、packages.config または PackageReferece のどちらが使用されているかによって異なります。

## デスクトップ

- package.config を使用してプロジェクトの Reveal SDK を新しいバージョンに更新後、古い NuGet バージョンのアンインストールや新しいバージョンへの更新ができません。

**回避策 1:** Reveal SDK を更新する前にプロジェクトから NuGet パッケージをアンインストールし、インストーラーを使用して Reveal SDK を更新すると、更新された NuGet パッケージを再インストールできます。

**回避策 2:** Reveal SDK を更新する前に既存の NuGet パッケージをバックアップします。これは、"%public%\Documents\Infragistics\Nuget" (インストーラーによって作成されたローカル NuGet パッケージストアの場所) に移動します。既存のパッケージを他のフォルダーにバックアップし、更新されたインストーラーを実行してから、バックアップしたパッケージを同じ場所にコピーします。これでプロジェクトで使用されている NuGet バージョンをアップグレードできます。

# リリースノート

---

以下は、Reveal SDK の各バージョンに含まれる新機能および拡張機能です。

日付	SDK バージョン	説明
9月2021年	1.0.2013	<p>[公開バグ修正] 計算フィールドを Excel にエクスポートすると、セルが空になる問題</p> <p>ゼロ除算を行う計算フィールドを Excel にエクスポートすると、結果には空のセルが含まれていました。</p>
9月2021年	1.0.2012	<p>[公開バグ修正] [SDK] スパークライン チャートが 100% を超えると小数点が非表示になる問題</p> <p>Web SDK では、ダッシュボードの設定後に変更された場合、canSaveAs プロパティが優先されませんでした。</p>
9月2021年	1.0.2012	<p>[公開バグ修正] [SDK] ウィンドウ サイズが小さいと、テキストチャートが判読できなくなる問題</p> <p>Web とデスクトップの両方で、ウィンドウのサイズが小さいと、テキストチャート フォントが読み取れなくなります。</p>
8月2021年	1.0.2008	<p>[公開バグ修正] [SDK] ダッシュボードをストリームとして保存する際に問題が発生する問題</p> <p>ダッシュボードをストリームとして保存するときに、特定のケースで <code>dashboard.Serialize.Async()</code> が null を返していました。</p>
6月2021年	1.0.2005	<p>散布図が OpenStreetMap をサポートするようになりました!</p> <p>デスクトップ (WPF) および Web クライアント (JS) で OpenStreetMap 画像タイルを構成して使用できるようになりました。</p>
		<p>新しいサムネイル コンポーネント!</p> <p><code>RevealDashboardThumbnailView</code> を使用してダッシュボードのサムネイルを描画できるようになりました。</p>

日付	SDK バージョン	説明
		<p>Web クライアントからサーバー側のデータ ソースへの資格情報新しいタイプの資格情報 <i>RVHeadersDataSourceCredentials</i> を使用すると、認証ヘッダーとクッキーを REST および Web リソースのデータ ソースに送信できます。 詳細については、GitHub で次の<a href="#">サンプル</a>をご覧ください。</p>
		<p>SDK AspNetCore サービス挿入</p> <p><i>RevealSdkContext</i> および <i>RevealUserContext</i> の実装をタイプのみ (インスタンスを渡さない) として登録できるようになり、これらのクラスがコンストラクターを介して挿入された他の AspNetCore サービスを取得できるようになりました。 詳細については、GitHub で次の<a href="#">サンプル</a>をご覧ください。</p>
		<p>[公開バグ修正] 計算フィールド フィルターがサーバー上のデータ プロセスで機能しない問題</p> <p>データのサーバー側集計を有効にすると、フィルターとして使用される計算フィールドが期待どおりにデータをフィルタリングしていなかった問題。</p>
		<p>[公開バグ修正] ダッシュボード フィルターに関する Google アナリティクスの問題</p> <p>Google アナリティクス データ ソースからデータを取得するときに、ダッシュボード フィルターを作成できなかった問題。</p>
		<p>散布図が OpenStreetMap をサポートするようになりました!</p> <p>SDK Web クライアント (JS) で OpenStreetMap 画像タイルを構成して使用できるようになりました。</p>
		<p>[公開 SDK バグ修正] コンポーネントを再マウントした後、テキストボックスのコンテンツが表示されない問題</p> <p>ダッシュボードとテキストボックスの視覚化で React を使用すると、コンポーネントの再マウント後にコンテンツが表示されませんでした。 ページの再読み込みが必要でした。</p>
6月 2021 年	1.0.7 JAVA	<p>[公開バグ修正] 計算フィールド フィルターがサーバー上のデータ プロセスで機能しない問題</p> <p>データのサーバー側集計を有効にすると、フィルターとして使用される計算フィールドが期待どおりにデータをフィルタリングしていなかった問題。</p>
		<p>[公開バグ修正] ダッシュボード フィルターに関する Google アナリティクスの問題</p> <p>Google アナリティクス データ ソースからデータを取得するときに、ダッシュボード フィルターを作成できなかった問題。</p>

日付	SDK バージョン	説明
6月2021年	1.0.6 JAVA	<p>[バグ修正] [SDK] Grizzly サーバーが例外をスローする問題            Grizzly で Reveal を実行すると、<i>javax.servlet.ServletContext</i> クラス (<i>javax.servlet:javaz.servlet-api</i> アセンブリ) の依存関係が間違っているため、<i>NoClassDefFoundError</i> 例外がスローされていました。</p>
		<p>JAVA SDK の新しいサンプルがリリースされました!  <a href="#">Grizzly</a> サーバーで Reveal を使用する方法を示す新しい <a href="#">GitHub サンプル</a> があります。</p>
6月2021年	1.0.5 JAVA	<p>Web クライアントからサーバー側のデータ ソースへの資格情報            新しいタイプの資格情報 <i>RVHeadersDataSourceCredentials</i> を使用すると、認証ヘッダーとクッキーを REST および Web リソースのデータ ソースに送信できます。詳細については、GitHub で次の<a href="#">サンプル</a>をご覧ください。</p>
5月2021年	1.0.1956 (1.0.4 JAVA)	<p>[公開バグ修正] [SDK] 誤って表示されたデータ ソースの完全なリストの問題            Desktop SDK で <i>DataSourcesRequested</i> コールバックを使用すると、明示的に追加されたデータ ソースの代わりに、データ ソースのリスト全体が表示されていました。</p> <p>[公開バグ修正] [SDK] Desktop SDK の Excel へのエクスポートが期待どおりに機能しない問題            ダッシュボードを歳読み込んでから単一の視覚化を Excel にエクスポートすると、ダッシュボードの最初の視覚化は常にエクスポートされたものでした。</p> <p>[公開バグ修正] [SDK] 動的ポートを使用する SQL データ ソースを含むダッシュボードが読み込まれていない問題            動的ポート (ホスト フィールドにインスタンスを提供) を使用して定義された SQL データ ソースを使用してダッシュボードを読み込むと、動的ポート構成の問題が原因でデータ ソース接続が機能しませんでした。</p> <p>[公開バグ修正] 視覚化フィルターとして設定された計算フィールドがエラーをスローしていた問題            別の計算フィールドに依存する計算フィールドに基づいて視覚化フィルターを構成すると、「無効な列名」というエラーが表示されました。</p> <p>[公開バグ修正] "sort by:" 構成が異なるドリルダウン シナリオが期待どおりに機能しない問題            階層内のフィールドが "sort by:" と降順の並べ替えの組み合わせで構成されている場合、結果としてダッシュボードが読み込まれませんでした。</p>

日付	SDK バージョン	説明
		<p>クロスドメイン アプリケーションでの Web クライアントからサーバー側への資格情報 バックエンドがフロントエンドと同じドメインになく、認証クッキーが必要な場合は、次の Web SDK 設定を使用して資格情報を要求できます: <b>\$ig.RevealSdkSettings.requestWithCredentialsFlag = true;</b></p>
5 月 2021 年	1.0.3 JAVA	<p>新しい Snowflake コネクタ! Reveal Java SDK は、Snowflake データ ソース コネクタをサポートするようになりました。これには、同じ Snowflake データベース内のテーブル間のデータブレンディングも含まれます。</p>
4 月 2021 年	1.0.0 JAVA	<p>Java 用の Reveal BI エンジンが強化されました。 Java プラットフォームは他のプラットフォームと同じくらい堅固になり、可視化が大量のデータをクライアントに送り返すときにサーバーがクラッシュするのを防ぐのに役立ちます。 <b>InitializeParameterBuilder</b> のいくつかの新しいプロパティがこれを制御します: <i>maxInMemoryCells</i>、<i>maxStorageCells</i>、<i>maxStringCellSize</i>、および <i>maxTotalStringSize</i>。</p>
3 月 2021 年	1.0.1866	<p>新しい JAVA SDK のリリース Reveal は、.NET 以外の別の Web サーバー オプションとして JAVA をサポートするようになりました。JAVA SDK には JAVA 11+ が必要であり、Maven モジュールのセットとして配布されます。詳細については、<a href="#">セットアップと構成</a>を参照してください。</p>
		<p>JAVA SDK サンプルのリリース 使用可能な JAVA SDK UpMedia サンプルは <a href="#">Github</a> に掲載されています。</p>
		<p>Web および Desktop SDK の新しいプロパティ: <i>showEditDataSource</i> - データ ソース オーバーフロー メニューで通常使用できる [編集] ボタンを無効にするために使用できます。 <i>canAddDashboardFilter</i> - このプロパティは、[フィルターの追加] メニューの [ダッシュボード フィルターの追加] オプションを非表示にすることができます。これらのオプションは、ダッシュボード編集モードで使用できます。 <i>canAddDateFilter</i> - このプロパティは、[フィルターの追加] メニューの [日付フィルターの追加] オプションを非表示にすることができます。これらのオプションは、ダッシュボード編集モードで使用できます。</p>
		<p>[公開バグ修正] [SDK] <i>revealView.canSaveAs</i> プロパティが正しく動作しない問題 Web SDK では、ダッシュボードの設定後に変更された場合、<i>canSaveAs</i> プロパティが優先されませんでした。</p>

日付	SDK バージョン	説明
		[公開バグ修正] [SDK] HttpContextAccessor.HttpContext プロパティが正しく動作しない問題 Web SDK では、ダッシュボードを保存する際に (SaveDashboardAsync メソッドからアクセスした場合)、HttpContextAccessor.HttpContext が null になります。
3月 2021 年	1.0.1821	[公開バグ修正] [SDK] SDK アプリが NRE 例外をスローすることがある問題 ユーザーが操作せずに SDK アプリケーションを 90 分以上開いた場合、操作を実行すると例外がスローされていました。
2月 2021 年	1.0.1772	[公開バグ修正] [SDK] packages.config で WPF NuGet パッケージのインストールが失敗する問題 ホストプロジェクトが packages.config を使用した場合、WPF NuGet パッケージのインストールが失敗していました。
		[公開バグ修正] [SDK] HasPendingChanges プロパティが正しく動作しない問題 Desktop SDK では、ダッシュボードを変更して保存した後、HasPendingChanges プロパティが false に設定されていませんでした。
		[公開バグ修正] [SDK] カスタム フィルタリングが正しく動作しない問題 Desktop SDK では、カスタム クエリが正しくデータをフィルタリングしていました。
2月 2021 年	1.0.1763	[公開バグ修正] [SDK] SQL Server テーブルを非表示にすると、ビューも非表示になる問題 RVDataSourceItemsFilter を使用してすべてのテーブルを非表示にし、ビューのみを表示すると、[ビュー] タブも非表示になりました。
		[公開バグ修正] [SDK] AzureSQL データ プロバイダーがエラーを発生する問題 AzureSQL 接続を追加すると、エラー メッセージが表示されました。
		[公開バグ修正] [SDK] LocalizationProvider が設定されている場合に日付フィルターが表示されない問題 LocalizationProvider が設定されている場合、表示形式エディターに日付フィルターの開始/終了は表示されません。
		[公開バグ修正] 日本語にローカライズされていない単語。 「Others」は、日本語の「その他」にローカライズされていませんでした。
1月 2021 年	1.0.1712	[公開バグ修正] [SDK] サーバー コンポーネントは Newtonsoft.Json シリアライザーに依存しています。 Reveal サーバー コンポーネントは、MVC アプリケーションのデフォルトの JSON シリアル化設定に依存していました。これで、ホスティング アプリは必要に応じて JSON シリアル化設定を構成できます。

日付	SDK バージョン	説明
12 月 2020 年	1.0.1669	<p>[公開バグ修正] [SDK] SQL Server フィルタリングが NVARCHAR 列で機能しない問題 フィルタリングされた値にマルチバイト文字が含まれている場合、Microsoft SQL Server のフィルタリングが NVARCHAR 列に対して機能しませんでした。</p>
		<p>[公開バグ修正] [サーバーでデータを処理] で SDK ピボット階層フィルタリングが機能しない問題 [サーバーでデータを処理] オプションがオンになっている場合、ピボットエディターでのドリルダウン階層はデータをフィルタリングしていかなかった問題。</p>
12 月 2020 年	1.0.1629	<p>[公開バグ修正] [サーバーでデータを処理] で SDK カスタム フィルタリングが機能しない問題 [サーバーでデータを処理] オプションがオンになっている場合、カスタム クエリは正しい行数を返しなかった問題。</p> <p>JSON ファイルを使用してダッシュボードを保存/ロード Reveal SDK を使用して、JSON ファイルからダッシュボードを保存/ロードできるようになりました。</p>
		<p>[公開バグ修正] カテゴリ フィールド ラベルが表示されない問題 カテゴリ チャートでは、ツールチップにフィールド ラベルではなく、カテゴリの元のフィールド名が表示されていました。</p>
12 月 2020 年	1.0.1629	<p>[公開バグ修正] ドリルダウン ブレッドクラムの日付が誤って表示される問題 日付フィールドをドリルダウンすると、ブレッドクラムに値が正しく表示されませんでした。これで、ブレッドクラムはドリルダウン レベルに関する明確な情報を提供することになりました。</p>
		<p>[公開バグ修正] ホバー ツールチップと十字線がデフォルトで表示されない問題 ダッシュボード ビュー モードでは、ユーザーが有効にするまで、ホバー ツールチップと十字線は表示されませんでした。現在、これらはデフォルトで有効になっています。</p>
9 月 2020 年	1.0.1422	<p>Amazon Athena コネクター (ベータ版) Amazon のサーバーレス インタラクティブな Athena クエリ サービスに接続できるようになりました。</p>

日付	SDK バージョン	説明
		<p>新しいビルド済みテーマ</p> <p>4つのビルド済みアプリ テーマを追加しました。いずれかを選択し、カスタマイズ可能な設定を使用して、表示形式およびダッシュボード エディターのレックアンドフィールをカスタマイズします。次のテーマから選択できます: MountainLightTheme (Desktop) / \$.ig.MountainLightTheme (Web); MountainDarkTheme (Desktop) / \$.ig.MountainDarkTheme (Web); OceanLightTheme (Desktop) / \$.ig.OceanLightTheme (Web); OceanDarkTheme (Desktop) / \$.ig.OceanDarkTheme (Web)。</p>
		<p>Marketo プロバイダーを利用できるようになりました。</p> <p>Marketo マーケティング プラットフォームに接続し、データを Reveal で使用します。</p>
		<p>Amazon Redshift を利用できるようになりました。</p> <p>Amazon Redshift クラウド データ ウェアハウスのデータを使用して、新しいインサイトを得ることができます。</p>
		<p>新しい「サーバーでデータを処理」機能</p> <p>MS SQL、MySQL、Postgres データ ソースからのデータをサーバー側で集計することができます。</p>
		<p>チャートの軸範囲を設定する新しい API</p> <p>特定の表示形式のためにランタイムで軸範囲をプログラム的に変更できるようになりました。</p>
		<p>Salesforce データ ソースの機能強化</p> <p>Reveal で Salesforce レポートを使用できます。</p>
		<p>新しい Quickbooks データ ソース</p> <p>Quickbooks アカウントに接続し、エンティティを使用して Reveal でデータ分析を実行します。</p>
7月 2020 年	1.0.1374	<p>新しい Hubspot データ ソース</p> <p>Hubspot に接続できます。</p>
		<p>Sharepoint リストとドキュメント ライブラリのサポート</p> <p>SharePoint ライブラリのすべてのファイルについて収集されたメタデータ (名前、タイプなど) を Reveal のデータ ソースとして使用できるようになりました。</p>
		<p>新しい階級区分図</p> <p>階級区分図の表示形式により、美しい主題図を作成できます。地理空間データを驚くほどわかりやすく表示できます。色によって、マップ上のパターン、トレンド、および異常をすばやく発見できます。</p>

日付	SDK バージョン	説明
5月 2020 年	1.0.1255	<p>新しい Azure Analysis Services データ ソース この新しいデータ ソースにより、Azure Analysis Services のデータ モデルを使用してダッシュボードを作成できます。</p>
		<p>Google スプレッドシート ファイルの新しいアイコン Google スプレッドシート ファイルのアイコンが変更されました。</p>
		<p>新しいホバー イベント API この新しいイベントは、WPF では revealView.ToolTipShowing、Web では .onTooltipShowing と呼ばれ、エンドユーザーが表示形式でシリーズをホバーするか、シリーズをクリックするたびに発生されます。</p>
		<p>新しい TreeMap 表示形式 この新しい表示形式タイプを使用して、大きな階層をネストされた四角形の集合で表示できます。四角形のサイズは、さまざまなメトリック間の部分と全体の関係を示し、同様のデータ間のパターンと関係を識別します。</p>
5月 2020 年	1.0.1222	<p>Excel へエクスポート機能拡張 エクスポートする際に複数の表示形式タイプをスプレッドシートに追加できます。散布図、バブルチャート、スパークラインチャートが利用できるようになりました。</p>
		<p>UI/UX の改善 表示形式、ダッシュボード、新しいデータ ソース ダイアログなどのユーザー エクスペリエンスを向上するために、小さな変更が追加されました。</p>
		<p>Google ドライブで共有ドライブのサポートを追加 G Suite Business アカウントをお持ちの場合、共有ドライブ データにアクセスし、それを使って Reveal で表示形式を構築できます。</p>
4月 2020 年	1.0.1136	<p>新しいカスタム テーマ 新しい RevealTheme (デスクトップ) / \$.ig.RevealTheme (Web) クラスでカスタマイズ可能な設定の一部またはすべてを構成することにより、Reveal で独自のテーマを作成できるようになりました。</p>
2月 2020 年	1.0.981	<p>RevealSettings の新しいプロパティ \$.ig.RevealSettings にさまざまな機能を制御するための複数の新しいプロパティを追加しました。ShowExportToPDF、ShowExportToPowerpoint、ShowExportToExcel、ShowStatisticalFunctions、ShowDataBlending、ShowMachineLearningModelsIntegration、StartWithNewVisualization、InitialThemeName。</p>
		<p>アクセント色のサポート SetAccentColor メソッドが \$.ig.RevealView に追加されました。</p>

日付	SDK バージョン	説明
		Trigger プロパティが DataSourceRequested イベントに追加されました。 DataSourcesRequestedTriggerType 型の Trigger プロパティを DataSourcesRequested イベント引数に追加しました。このイベントのユーザーは、DataSourcesRequested の目的について詳細なコンテキストを取得できます。
11月 2019 年	1.0.825	画像エクスポート機能が利用できるようになりました。 サーバー側の画像エクスポート(プログラム上およびユーザー操作の両方により)が再び有効になりました。修正の詳細については、以下のトピックを参照してください。 <a href="#">サーバー側画像生成の有効化</a>
		Reveal Desktop SDK のローカリゼーションサービス さまざまなダッシュボード要素のタイトルおよびラベルをローカライズすることができます。ローカリゼーションサービスでは、数値および非集計の日付フィールドの書式設定を変更することもできます。
9月 2019 年	1.0.80x	Reveal Desktop SDK の書式設定サービス 数値データ、集計および非集計の日付フィールドを好みに合わせて書式設定できます。デフォルトの書式設定を無視して、ダッシュボードデータを書式設定します。
		セットアップと構成の変更 (Server SDK) Reveal Server SDK には、.NET Core 2.2+ および .NET Framework 4.6.1+ ASP MVC アプリケーションプロジェクトがサポートされます。また、NuGet パッケージマネージャーのみを使用すると、アセンブリを参照し、依存関係パッケージをインストールします。
		ステップバイステップ ガイド Reveal SDK の前提条件、セットアップや構成に必要な手順全般に関するトピックを追加。
9月 2019 年	1.0.70x	ウィジェットデータソースの変更 エンドユーザーによってウィジェットのデータソースを変更する機能を有効または無効にできます。編集モードで [可視化データ] 画面を開いた際に、UI の [データソースの変更] ボタンを表示または非表示にできます
		ダッシュボードテーマの変更 エンドユーザーによってダッシュボードのテーマを変更する機能を有効または無効にできます。ダッシュボードの編集モードに入る際に、使用可能なテーマを表示するためのボタンを表示または非表示にできます。