Sitecore CMS 6.6

セグメント ビルダー デベロッパー ガイド

デベロッパー向けセグメント ビルダー API 利用ガイド

イントロダクション

セグメント ビルダー コンポーネントは、Analytics データベースから一定の条件を満たす訪問者のリストを取得するために使用します。このコンポーネントは、エンゲージメント オートメーションで、オートメーション プランの状態にセグメントを追加するために使用します。詳細については、『エンゲージメント オートメーション クックブック』の、「状態へのセグメントの追加」のセクションを参照してください。

本ガイドは、API リファレンスとデベロッパー ガイドを兼ねています。セグメント ビルダーの、Sitecore のエンゲージメント オートメーション機能との連携機能について記載しています。

デベロッパーは、セグメント ビルダー コンポーネントを使って以下のことができます。

- ダイアログを使用して、選択したセグメントから訪問者を取得する。
- 新規セグメント ビルダー ルールを作成する。

セグメント ビルダー ダイアログの使用

この章では、以下を説明します。

- プログラム的にダイアログを開く方法。
- ルール コレクション (ダイアログの結果として得られます) で定義されたセグメントから訪問者を取得する方法。これは、本文書の最初の部分です。

セグメント ビルダーへのアクセス

エンド ユーザーがセグメント ビルダー ダイアログを開き、一定の条件を満たす訪問者のリストを Analytics データベースから取得する手順を以下に示します。

1. 以下のコードを使用して、ダイアログを開きます。

```
var url = new UrlString(Sitecore.Context.Site.XmlControlPage);
url["xmlcontrol"] = "Sitecore.Shell.Applications.Analytics.SegmentBuilder";
var handle = new UrlHandle();
handle["Value"] = <rules value>; //when updating an existing rule, pass the XML
created previously, otherwise - use string.Empty.
handle.Add(url);
SheerResponse.ShowModalDialog(url.ToString(), "800px", "600px", string.Empty, true);
args.WaitForPostBack();
```

2. ダイアログ上で各種条件を選択し、**[OK]** をクリックします。

3. 次に、ポストバック アクションを実装することで、ダイアログ結果を取得します。取得結果は、ルールを表す文字列となります。セグメント ビルダーを使用して、これらのルールを条件セットを定義するものに変換、また後に SQL ステートメントに変換することができます。以下のコード スニペットはこのアクションを記述したものです。

```
if (args.IsPostBack)
{
    if (args.HasResult)
    {
        var value = args.Result == "-" ? string.Empty : args.Result;
        //processing the value code here. Use SegmentBuilder class to get number of
          visitors of list of Visitor IDs:
        //var segmentBuilderHelper = new
          Sitecore.Shell.Applications.Analytics.SegmentBuilder.SegmentBuilder();
        //int count = segmentBuilderHelper.GetVisitorCount(value);
    }
    return;
}
```

4. この結果に基づき、所定のユーザー ルールを満たす訪問者の数または ID を取得するコードを実装できます。

以下に、`SegmentBuilderRules` クラスの例を示します。このクラスは、セグメント ビルダー ダイアログを開き、またその結果を保存するものです。結果は、通常の *Rules* 形式で保存されます。

```
namespace Sitecore.Shell.Applications.ContentEditor
{
    using Sitecore.Diagnostics;
    using Sitecore.Text;
    using Sitecore.Web;
    using Sitecore.Web.UI.Sheer;

    /// <summary>
    /// The segment builder rules.
    /// </summary>
    public class SegmentBuilderRules : Rules
```

```
        {
            /// <summary>
            /// Edits the text.
            /// </summary>
            /// <param name="args">The arguments.</param>
            protected override void Edit(Sitecore.Web.UI.Sheer.ClientPipelineArgs args)
            {
                Assert.ArgumentNotNull(args, "args");

                if (this.Disabled)
                {
                    return;
                }

                if (args.IsPostBack)
                {
                    if (args.HasResult)
                    {
                        this.Value = args.Result == "-" ? string.Empty : args.Result;
                        SheerResponse.SetAttribute(this.ID, "value", this.Value);
                        SheerResponse.SetModified(true);
                        this.Refresh();
                    }

                    return;
                }

                var value = this.Value;
                if (value == EditorConstants.NoValue)
                {
                    value = string.Empty;
                }

                Assert.IsNotNull(Sitecore.Context.Site, "site");
                var url = new UrlString(Sitecore.Context.Site.XmlControlPage);
                url["xmlcontrol"] =
                 "Sitecore.Shell.Applications.Analytics.SegmentBuilder";
                var handle = new UrlHandle();
                handle["Value"] = this.Value;
                handle.Add(url);

                SheerResponse.ShowModalDialog(url.ToString(), "800px", "600px",
                 string.Empty, true);

                args.WaitForPostBack();
            }
        }
    }
```

ルール コレクションから訪問者を取得する

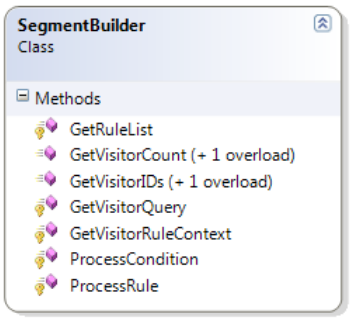
このセクションでは、セグメント ビルダーの主要なクラス、機能、親クラス、実装、メソッド、プロパティについて説明します。各セクションには、API 内の 1 つの重要なモジュールについて記述されます。また、各セクションには、クラスの説明と、そのクラスのプロパティやメソッドの情報が表形式でまとめられています。

以下のクラスを使用しても、訪問者の情報を取得できます。

Sitecore.Shell.Applications.Analytics.SegmentBuilder.SegmentBuilder

`SegmentBuilder` クラスのメソッドを使用して、選択したルールを満たす `VisitorIds` を取得します。

これは、セグメント ビルダー ダイアログで取得した結果から、訪問者 ID リストを取得するクラスヘルパーです。ダイアログでの結果は文字列として取得され、ダイアログで定義されたセグメントをシリアル化したものです。各セグメントは、セグメント ビルダー ルールの組み合わせです。



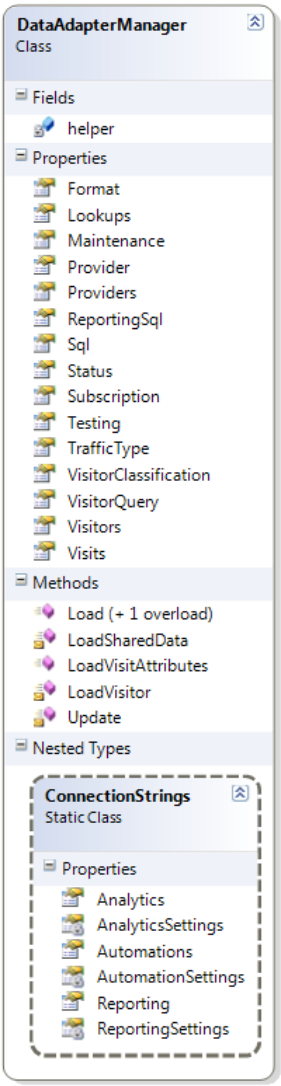
以下の表に、SegmentBuilder クラスのパブリック メソッド、およびそれらのオーバーロード メソッドを示します。

メソッド	説明
`int GetVisitorCount( string rules, Database database)`	サイト訪問者数を取得します。 rules パラメーターは、セグメント ビルダー ルールを表す文字列です。 database パラメーターには、ルール アイテムの定義が格納されています。 例: `<visitorFilters> <rule uid=\"{F3C60029-4911-4EDA-B8E4-732B81E E17C9}\"> <conditions> <condition id=\"{CB7EAA5E-3D4D-41DF-8453- 34DB6FCE2F9E}\" uid=\"2EE7B61586224E01AC71CAB9C5E819DF\" automationids=\"{D036CAB9-B2F5-4176-A67F- 82081463B78E}|{1A9E3116-8D42-46C5-BAD1- FD4D9B342826}\" /> </conditions> </rule> </visitorFilters>` ルールを処理するために、メソッドは指定されたデータベースからアイテムを取得し、ルールを実行します。 以下のパスに従ってコンテンツ ツリーをたどることで、セグメント ビルダーのルール アイテムにアクセスできます。 Master/sitecore/system/Settings/Rules/Segment Builder ルール エンジンの詳細については、『ルール エンジン クックブック』を参照してください。
`int GetVisitorCount( string rules)`	前述のメソッドのオーバーロードです。訪問者数を取得します。 データベースの値をコンテキスト データベースに等しい Client.ContentDatabase に割り当てます。
`Enumerable<Guid> GetVisitorIDs( string rules, Database database)`	このメソッドは、前述の 2 つのメソッドに類似したものです。唯一の違いは、指定したルールを満たす訪問者の ID のリストを返すということです。
`IEnumerable<Guid> GetVisitorIDs( string rules)`	これは、前述のメソッドのオーバーロードです。ルールに基づいて、訪問者 ID を取得します。 データベースの値をコンテキスト データベースと等しい Client.ContentDatabase に割り当てます。

メモ
訪問者に関連したその他の情報を取得するには、Engagement Analytics API の以下のクラスを使用してください。
Sitecore.Analytics.Data.DataAccess.VisitorFactory.GetVisitor

Sitecore.Analytics.Data.DataAccess.DataAdapters.DataAdapterManager

`DataAdapterManager` クラスは、`DataAdapterProvider` クラスにメソッド コールをリダイレクトするデータ アダプター マネージャーを定義します。



以下の表に、セグメント ビルダーの `DataAdapterManager` クラスのプロパティを示します。その他のプロパティの詳細については、『Engagement Analytics API リファレンス ガイド』を参照してください。

プロパティ	説明
`VisitorQuery` VisitorQuery	このクラスには、2 つの機能があります。 • `GetVisitorIDsSql` メソッド、`GetVisitorCountSql` メソッドの基本クラスです。これらのメソッドを使って、訪問者のクエリを、訪問者の数と ID を取得する、DBMS で実行可能なステートメントに変換できます。 • 訪問者のクエリを取得します。Analytics データベース内のルール アイテムのうち、ルール クラスを参照するものにアクセスします。これらのクラスを使って、クエリを組み立てます。 **メモ** `SqlVisitorQuery` クラスのような継承クラスを使用して、このオブジェクトに格納されたデータを適切な SQL または Oracle ステートメントに変換することもできます。
VisitorBase Visitors	訪問者を取得します。このプロパティを使って、構築した VisitorQuery オブジェクトから、訪問者の数または ID を取得することもできます。

このクラスのメソッドは、セグメント ビルダーではなく、Engagement Analytics API に属するものです。

セグメント ビルダー ルールを作成する

この章では、新規のルール クラスの作成について説明します。`Sitecore.Analytics.Data.DataAccess.DataAdapters` 名前空間内のすべての関連するクラスについて記述します。

セグメント ビルダー ルール API

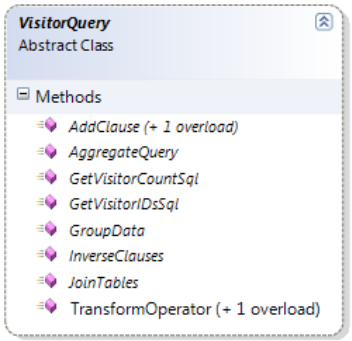
以降のセクションでは、`DataAdapters` 名前空間内のクラスについて記述します。この名前空間は、Engagement Analytics API に属するものですが、セグメント ビルダー ルールの作成にも使用します。

Sitecore.Analytics.Data.DataAccess.DataAdapters.VisitorQuery

`VisitorQuery` は、訪問者のクエリを操作するための主要なメソッドを持つ抽象クラスです。実装されたルールが条件クエリを構築する際に呼び出すメソッドが属するクラスです。このクエリは、Analytics データベース上で実行されます。

このクラスは、以下のことを可能にします。

- 訪問者の選抜条件を統合し、セグメントにグループ化します。これには、`AddClause` メソッド、`AggregateQuery` メソッド、`JoinTables` メソッドを使います。
- 統合した条件に基づき、SQL ステートメントを生成します。このステートメントは、Analytics データベース上で実行できます。例えば、以下のことが可能です。
    - `GetVisitorCountSql` を使って、DBMS で訪問者数を取得するための SQL ステートメントを取得する。
    - `GetVisitorIDsSql` を使って、DBMS で訪問者の ID のリストを取得するための SQL ステートメントを取得する。



以下の表に、VisitorQuery クラスのメソッドを示します。

メソッド	説明
void AddClause(ClauseBase left, ClauseBase right, ClauseOption option, string group)	句の左側と右側の引数、論理演算子、グループ名から句を構築し、クエリに追加します。 group パラメーターはルールごとに一意に決定される名前です。同じグループに属するクエリは、かっこで囲んで連結されます。 例: AddClause("a=b", "c=d", "and" , "group1"); AddClause("e=f", "g=h", "and" , "group2"); この例では、異なる 2 つのグループがあるので、SQL クエリは次のようになります。 Select… from… where (a=b and c=d) or (e=f and g=h) 例: AddClause("a=b", "c=d", "and" , "group1"); AddClause("e=f", "g=h", "and" , "group1"); この例では、同じグループなので、SQL クエリは次のようになります。 Select… from… where (a=b and c=d or e=f and g=h)
void AddClause(ClauseBase clause, string group)	句とグループ名に基づき、句を追加します。
void AggregateQuery(VisitorQuery query, ClauseOption option)	訪問者のクエリの句と論理演算子を使って、操作に関連したすべての登録された句を含む句を生成します。
string GetVisitorCountSql(out object[] parameters)	parameters パラメーターを受け取り、訪問者の数を求める SQL ステートメントを返します。 Parameters は、結果を得るために SQL ステートメントとあわせてデータベースに渡されるパラメーターの配列です。
string GetVisitorIDsSql(out object[] parameters)	パラメーターを受け取り、SQL ステートメントを含む文字列を返します。 この SQL ステートメントは、訪問者の ID のリストを返します。これらの ID は、指定されたルールにしたがって統合された条件を満たす訪問者のものとなります。
void GroupData(AnalyticsTable table, string column)	Analytics テーブル、列名を受け取り、この列名に基づいてデータをグループに分けます。

メソッド	説明
`void InverseClauses()`	`VisitorQuery` オブジェクト内のすべての句を統合し、ネゲートします。「ネゲートする」とは、つまり論理演算子「Not」を付加するという意味です。 例: それぞれが数を表す 4 つの文字、`A`、`B`、`C`、`D` があったとします。句「`letter > B`」は、`C` と `D` を返します。 この句の逆は、「`Not(letter > B)`」または「`letter <= B`」となり、その戻り値は `A` と `B` となります。 例: 上記と同様に、`A`、`B`、`C`、`D` に対し、「`letter = A OR letter > C.`」の句を考えます。この句の逆は、「`Not(letter = A OR letter > C)`」となり、この戻り値は `B` と `C` になります。 これは、「`except where`」ルールに使用します。例えば、「`except where visitor from UK`」(英国のユーザー以外)というルールは、「NOT (user from UK)」と同義です。

`void JoinTables(`
`  AnalyticsTable tableLeft,`
`  string columnLeft,`
`  AnalyticsTable tableRight,`
`  string columnRight)`
	Analytics の左のテーブル、左の列、Analytics の右のテーブル、右の列を受け取り、これらのテーブルを結合します。

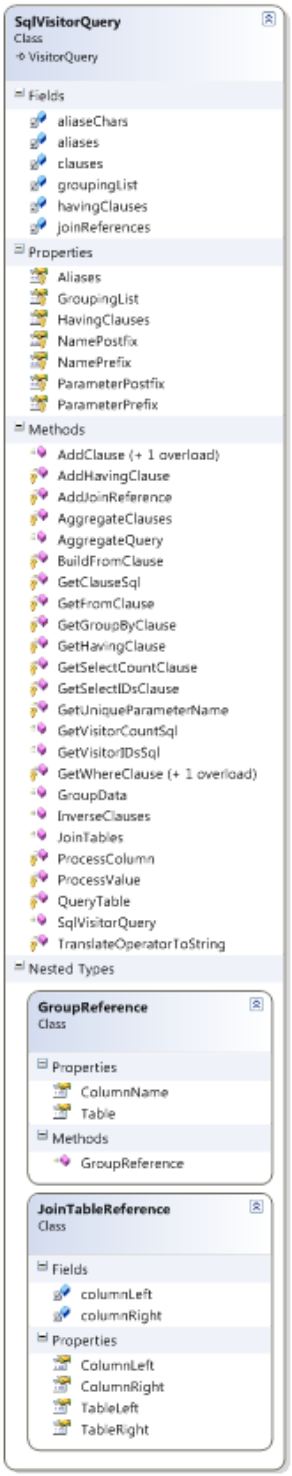
`QueryConditionOperator TransformOperator(` `  ConditionOperator @operator)`	ルールに関連した論理演算子を受け取り、`VisitorQuery` がサポートする演算子に変換します。 例えば、ルール エンジンは、以下の 2 つの論理演算子をサポートしています。 `greater 、 equal`
`QueryConditionOperator TransformOperator(` `StringConditionOperator @operator)`	ルールに関連した文字列演算子を受け取り、`VisitorQuery` がサポートする演算子に変換します。 例えば、ルール エンジンは、以下の 2 つの文字列演算子をサポートしています。 `contains、startwith` このメソッドは、デベロッパーが自分が策定したルールで使用している演算子を、クエリ操作に変換する際に使用します。

Sitecore.Analytics.Data.DataAccess.DataAdapters.Sql.SqlVisitorQuery

`SqlVisitorQuery` クラスを使うことで、SQL ステートメントとしてクエリを生成できます。

基本クラスの `VisitorQuery` は抽象クラスなので、常に `SqlVisitorQuery` または `OracleVisitorQuery` クラスを使用することになります。

`OracleVisitorQuery` クラスは、`SqlVisitorQuery` と同等の機能を持ちますが、Oracle データベース用となっています。

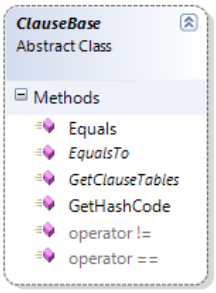


`SqlVisitorQuery` クラスのプロパティは、プライベートで、読み取り専用となっています。

このクラスのメソッドについては、`VisitorQuery` クラスを説明したセクションに記載してあります。

Sitecore.Analytics.Data.DataAccess.DataAdapters.ClauseBase

このクラスは、ルール句を処理するすべてのクラスの基本クラスです。句の共通のメソッドおよびプロパティを定義します。



以下の表に、ClauseBase クラスのメソッドを示します。

メソッド	説明
`bool Equals( object obj)`	C# の Equal メソッドと同様で、指定された句オブジェクトと、カレントの句が同一のものかどうかを判定します。
`bool EqualsTo( ClauseBase clause)`	指定された句とカレントの句が同一のものかどうかを判定します。前述のメソッドをより限定的にしたもので、ClauseBase オブジェクトのみを比較します。
`AnalyticsTable[] GetClauseTables()`	句で使用している表の配列を返します。
`int GetHashCode()`	カレントの句のハッシュ コードを返します。ハッシュ コードは、ハッシュテーブルのハッシュ アルゴリズムやデータ構造体に適合するものです。

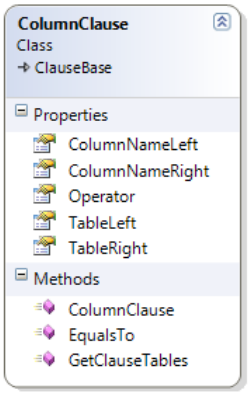
Sitecore.Analytics.Data.DataAccess.DataAdapters.ColumnClause

ColumnClause 句は、2 つの表の、2 つの列を比較するクエリで使用できます。

例:

```
Viritors.VisitorId = AutomationStates.VisitorId
```

このクラスでは、表の列を表す、左と右 2 つの引数と、論理演算子を使います。



以下の表に、ColumnClause クラスのプロパティを示します。

プロパティ	説明
`string ColumnNameLeft`	句の左側の引数の、列名を表します。 上記の例では、ColumnNameLeft は VisitorId となります。
`string ColumnNameRight`	句の右側の引数の、列名を表します。
`QueryConditionOperator Operator`	論理演算子を表します。
`AnalyticsTable TableLeft`	左側の引数の、表名を表します。 前述の例では、以下のように割り当てられます。 TableLeft は Visitors
`AnalyticsTable TableRight`	右の引数の表名を表します。

以下の表に、ColumnClause クラスのメソッドを示します。

メソッド	説明
`ColumnClause( AnalyticsTable tableLeft, string columnNameLeft, AnalyticsTable tableRight, string columnNameRight, QueryConditionOperator @operator)`	ColumnClause クラスの新規インスタンスを初期化します。
`bool EqualsTo( ClauseBase obj)`	句が同一かどうかを判定します。 **メモ** このメソッドは使わないでください。内部ロジック用のものです。
`AnalyticsTable[] GetClauseTables()`	句で使用している表の配列を返します。

Sitecore.Analytics.Data.DataAccess.DataAdapters.ComplexClause

and や or などの論理演算子で結合された 2 つ以上の論理式を含む句を、複合句と言います。

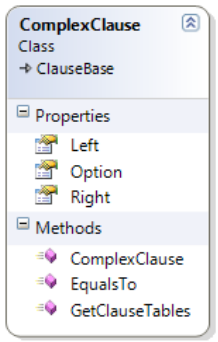
例:

```
select VisitorId from Visitors as v where v.ExternalUser= 'someuser' and v.Value > 5
```

このクエリでは、2 つの句が and 演算子で結合されています。このクエリ内の複合句を構築するには、以下のコード スニペットを使います。

```
var clauseLeft = new ValueClause(AnalyticsTable.Visitors, "ExternalUser", "someuser",
QueryConditionOperator.Equal);
var clauseRight = new ValueClause(AnalyticsTable.Visitors, "Value", 5,
QueryConditionOperator.GreaterThan);
var clause = new ComplexClause(clauseLeft, ClauseOption.And, clauseRight);
query.AddClause(clause, "somegroupname");
```

これはほんの一例です。他にも、2 つの句と演算子を扱うことのできる、VisitorQuery クラスの AddClause メソッドを使う方法もあります。



以下の表に、ComplexClause クラスのプロパティを示します。

プロパティ	説明
`ClauseBase Left`	ステートメントの左側の句を表します。
`ClauseOption Option`	句を結合するのに使用する論理演算子を表します。
`ClauseBase Right`	ステートメントの右側の句を表します。

以下の表に、ComplexClause クラスのメソッドを示します。

メソッド	説明
`ComplexClause( ClauseBase left, ClauseBase right, ClauseOption option)`	左右の句と論理演算子を受け取り、複合句を初期化します。

メソッド	説明
`bool EqualsTo( ClauseBase obj)`	指定された句とカレントの複合句が同一のものかどうかを判定します。 **メモ** このメソッドは使わないでください。内部ロジック用のものです。
`AnalyticsTable[] GetClauseTables()`	複合句で使用している表の配列を返します。

Sitecore.Analytics.Data.DataAccess.DataAdapters.HavingClause

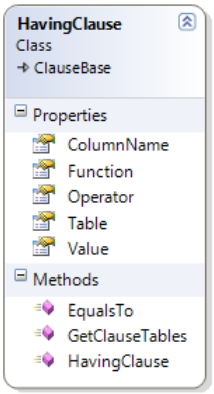
クエリで `Having` 句を表すのに使用します。詳細については、MSDN (Microsoft Developer Network) 内の Having (Transact-SQL) の記事を参照してください。

以下に `Having` 句の書式を示します。

```
HAVING <AggregateFunction>(<ColumnName>) <QueryConditionOperator> <Value>
```

例:

```
        HAVING Count(LineTotal) > 100
```



以下の表に、`HavingClause` クラスのプロパティを示します。

プロパティ	説明
`string ColumnName`	`Having` 句で使用する列の名前を表します。
`AggregateFunction Function`	`Having` 句で使用する関数を表します。 **メモ** `AggregateFunction` が現在サポートするのは、`Count` と `DistinctCount` です。
`QueryConditionOperator Operator`	`Having` で使用する演算子を表します。
`AnalyticsTable Table`	`Having` で使用する表を表します。
`object Value`	比較対象の値を表します。

以下の表に、`HavingClause` クラスのメソッドを示します。

メソッド	説明
`bool EqualsTo( ClauseBase obj)`	指定された句オブジェクトとカレントの `Having` 句が同一のものかどうかを判定します。 **メモ** このメソッドは使わないでください。内部ロジック用のものです。

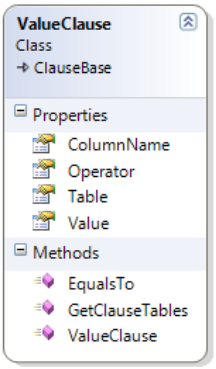
メソッド	説明
`AnalyticsTable[] GetClauseTables()`	`Having` 句で使用している表の配列を返します。
`HavingClause( AnalyticsTable table, string columnName, AggregateFunction function, object value, QueryConditionOperator @operator)`	`HavingClause` クラスの新規インスタンスを初期化します。

Sitecore.Analytics.Data.DataAccess.DataAdapters.ValueClause

この句は、表の列と値を比較するクエリで使用します。

例:

```
      Visitors.VisitorNumber = 5
```



以下の表に、`ValueClause` クラスのプロパティを示します。

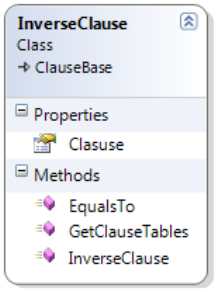
プロパティ	説明
`string ColumnName`	値と比較する列の名前を表します。
`QueryConditionOperator Operator`	比較演算子を表します。
`AnalyticsTable Table`	当該の列を含む表を表します。
`object Value`	列と比較する値を表します。

以下の表に、ValueClause クラスのメソッドを示します。

メソッド	説明
bool EqualsTo(ClauseBase obj)	指定された句オブジェクトとカレントの句が同一のものかどうかを判定します。 **メモ** このメソッドは使わないでください。内部ロジック用のものです。
AnalyticsTable[] GetClauseTables()	句で使用している表の配列を返します。
public ValueClause(AnalyticsTable table, string columnName, object value, QueryConditionOperator @operator)	ValueClause クラスの新規インスタンスを初期化します。

Sitecore.Analytics.Data.DataAccess.DataAdapters.InverseClause

InverseClause クラスは、指定された句をネゲートするのに使います。



以下の表に、InverseClause クラスの唯一のプロパティを示します。

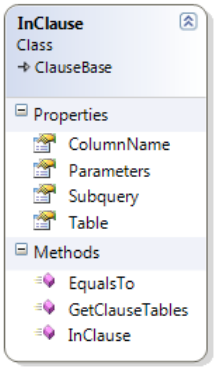
プロパティ	説明
ClauseBase Clause	ネゲートする句オブジェクトを表します。

以下の表に、InverseClause クラスのメソッドを示します。

メソッド	説明
bool EqualsTo(ClauseBase obj)	指定された句オブジェクトとカレントの inverse 句が同一のものかどうかを判定します。 **メモ** このメソッドは使わないでください。内部ロジック用のものです。
AnalyticsTable[] GetClauseTables()	句で使用している表の配列を返します。
InverseClause(ClauseBase clause)	InverseClause クラスの新規インスタンスを初期化します。

Sitecore.Analytics.Data.DataAccess.DataAdapters.InClause

クエリで IN 句を表すのに使用します。詳細については、MSDN (Microsoft Developer Network) 内の In (Transact-SQL) の記事を参照してください。



以下の表に、InClause クラスのプロパティを示します。

プロパティ	説明
`string ColumnName`	IN 句の左側の引数として使う列の名前を表します。
`object[] Parameters`	subquery のパラメーターを表します。
`string Subquery`	IN 句の右側の引数として使う subquery を表します。
`AnalyticsTable Table`	ColumnName 列を含む表を表します。

以下の表に、InClause クラスのメソッドを示します。

メソッド	説明
`bool EqualsTo( ClauseBase obj)`	指定された句オブジェクトとカレントの IN 句が同一のものかどうかを判定します。 **メモ** このメソッドは使わないでください。内部ロジック用のものです。
`AnalyticsTable[] GetClauseTables()`	句で使用している表の配列を返します。
`InClause( AnalyticsTable table, string columnName, string subquery, object[] parameters)`	InClause クラスの新規インスタンスを初期化します。

セグメント ビルダー ルールをクラスとして生成する

このセクションでは、セグメント ビルダーの API を使ってルールを作成する手順について説明します。このルールは、特定の国に所属する訪問者を取得します。

入力パラメーターは以下のとおりです。

- 国の名前
- 文字列比較演算子。リストから演算子を選択するルールを作成します。

メモ
セグメント ビルダーは、存在するすべての種類のルール演算子をサポートします。
これらの演算子は、Master データベースで定義されています。これらのルールへのコンテンツ ツリーのパスは、以下のとおりです。

```
/sitecore/system/Settings/Rules/Common/Operators 、
/sitecore/system/Settings/Rules/Common/String Operators
```

コンテンツ エディターでは、セグメント ビルダー ルールの生成は、通常の Sitecore ルールの生成と同様であり、この文書では割愛します。

訪問者のクエリを作成する

ルール クラスを実装するには、まず生成する SQL ステートメントの書式の理解が必要です。どの表や列を使うのかを把握していなくてはなりません。

例えば、以下の SQL は、ある一定の条件を満たす訪問者を検索します。

```
select v.VisitorId
from Visitors as v left join Visits as vi on v.VisitorId=vi.VisitorId
where vi.Country group by v.VisitorId
```

`select` 句やエイリアスについては、`VisitorQuery` クラスが自動的に生成するので、考慮する必要はありません。表と列だけ指定すれば十分です。

ルール クラスを作成する

作成したルールは国名を表す文字列を照合し、選択した比較演算子をサポートします。ルール クラスは、`Sitecore.Rules.Conditions.StringOperatorCondition<T>` クラスの派生クラスでなくてはなりません。ここで、「`T`」はルールのコンテキストを表します。

```
public class CountryCondition<T> : StringOperatorCondition<T> where T :
VisitorRuleContext
{
}
```

`StringOperatorCondition` クラスは`GetQueryOperator` メソッドを持ち、このメソッドは `StringConditionOperator` 型の比較演算子を返します。この比較演算子をさらに、`QueryConditionOperator` に変換する必要があります。

ルール パラメーターを作成する

ルール パラメーターは、ルール クラスのプロパティとして表す必要があります。

```
public string Value
{
  get;
  set;
}
```

例えば、`Country` をルール パラメーター アイテムとして指定、ルールの句に追加することができます。

```
string value = this.GetValue(ruleContext);
     var clause = new ValueClause(AnalyticsTable.Visits, "Country", value, @operator);
     ruleContext.VisitorQuery.AddClause(clause,
     ruleContext.GetUniqueConditionGroupName("CountryCondition"));
```

ルール ロジックを作成する

`CountryCondition<T>` ルール クラスでは、`StringOperatorCondition<T>` がベースとなっている、`Execute` メソッドをオーバーライドする必要があります。

ルール ロジックを実装するには、`Execute` メソッドをオーバーライドする必要があります。

```
protected override bool Execute([NotNull] T ruleContext)
{
  Assert.ArgumentNotNull(ruleContext, "ruleContext");
}
```

まとめ

ここまでは、セグメント ビルダー ルールを表すクラスを作成する要件について説明しました。

このルールは、SQL ステートメントとしてクエリを生成します。

以下にクラスの例を示します。このクラスは、次の機能を持ちます。

1. `StringConditionOperator` 型の比較演算子を、`QueryConditionOperator` 型の演算子に変換します。
2. 国名を取得し、ルール パラメーターとして渡します。
3. 新規のクエリ コンテキストと、適切な句を生成します。
4. 生成した句を、クエリに追加します。

```
public class CountryCondition<T> : StringOperatorCondition<T> where T :
VisitorRuleContext
{
  public string Value { get; set; }
  protected override bool Execute([NotNull] T ruleContext)
  {
    Assert.ArgumentNotNull(ruleContext, "ruleContext");
    //converting string operator to query one.
    QueryConditionOperator @operator = this.GetQueryOperator(ruleContext);
    if (@operator == QueryConditionOperator.Unknown)
    {
      Log.Warn("Cannot evaluate visit condition. Condition operator is not defined.",
      this);
      return true;
    }

    //creating rule context with clause
    var innerContext = VisitorRuleContext.Create(ruleContext.Item) as T;
    Assert.IsNotNull(innerContext, "right context");
    //join the necessary tables
    innerContext.VisitorQuery.JoinTables(AnalyticsTable.Visitors, "VisitorId",
    AnalyticsTable.Visits, "VisitorId");
    //grouping data by the visitor's ID.
    innerContext.VisitorQuery.GroupData(AnalyticsTable.Visitors, "VisitorId");
    //Add the country clause.
    this.AddClause(innerContext);
    object[] parameters;
    //Retrieving the SQL from the temporary query. This is a part of the method and
    should be the same for almost any rule. The final version of the SQL should look
    like "select VisitorId from Visitors where VisitorId IN (<select SQL from rule1>)
    AND VisitorId IN (<select SQL from rule2>) ...". In this way, you ensure that you
    don't break the whole statement by our rule.
    string subquery = innerContext.VisitorQuery.GetVisitorIDsSql(out parameters);
    var inClause = new InClause(AnalyticsTable.Visitors, "VisitorId", subquery,
    parameters);
    ruleContext.VisitorQuery.AddClause(inClause,
    ruleContext.GetUniqueConditionGroupName("VisitCondition"));
    //return "true" at the end of the rule. This tells to the system that everything
    is fine and your rule should be considered.
    return true;
  }
```

```
        protected virtual void AddClause([NotNull] T ruleContext)
        {
          Assert.ArgumentNotNull(ruleContext, "ruleContext");
          QueryConditionOperator @operator = this.GetQueryOperator(ruleContext);
          if (@operator == QueryConditionOperator.Unknown)
          {
            Log.Warn("Cannot evaluate country condition. Condition operator is not
            defined.", this);
            return;
          }
          string value = this.GetValue(ruleContext);
          var clause = new ValueClause(AnalyticsTable.Visits, "Country", value, @operator);
          ruleContext.VisitorQuery.AddClause(clause,
          ruleContext.GetUniqueConditionGroupName("CountryCondition"));
        }

        protected virtual QueryConditionOperator GetQueryOperator([NotNull] T ruleContext)
        {
          Assert.ArgumentNotNull(ruleContext, "ruleContext");
          StringConditionOperator stringOprtator = this.GetOperator();
          return ruleContext.VisitorQuery.TransformOperator(stringOprtator);
        }

        protected virtual string GetValue([NotNull] T ruleContext)
        {
          Assert.ArgumentNotNull(ruleContext, "ruleContext");
          return this.Value ?? string.Empty;
        }
      }
```