HCLSoftware

Unica Journey V12.1.8 管理者ガイド

Contents

Chapter 1. Unica Journey の概要

Unica Journey は、コンテキスト主導のパーソナライズされたマルチステップのオムニチャネル カスタマー エクスペリエンスを作成、実行、視覚化するための目標ベースのオーケストレーション ソリューションです。

Unica Journey にマーケターが使用できる:

- カスタマー エクスペリエンスの目標を定義する
- ジャーニーをリアルタイムで簡単に調整して達成する
- 洗練された直感的な Journey キャンバスを使用して、チャネル/タッチポイントおよびイベントのカスタマー・ジャーニー全体を作成および視覚化します

カスタマー ジャーニーは完全に自動化され、顧客のブランド エンゲージメントのすべてのステップと同期されます。Journey内でリアルタイムのインサイトを使用して、Journey で起きていることを反映する顧客の行動を理解する。

Unica Journey の特長

Unica Journey の特徴以下の通り:

- **目標主導のエクスペリエンス**: カスタマー・エクスペリエンスの目標を定義し、ジャーニーをリアルタイムで簡単に調整して達成します。
- **オーケストレーション・キャンバス**: 洗練された直感的な Journey キャンバスを使用して、チャネル/タッチポイントおよびイベントを横断するカスタマー・ジャーニー全体を作成および視覚化します。
- **Always on Engagement**: 顧客のブランド・エンゲージメントのすべてのステップと同期する、完全に自動化された実行です。
- **リアルタイムのインサイト**: ジャーニーでの出来事を反映したインサイトを使用して、顧客の行動を理解します。
- **タッチポイントの選択**: デジタル・チャネル用のすぐに使用できるネイティブ・タッチポイントを活用するか、カスタム・タッチポイントを作成して、エコ・システム全体のジャーニーをシームレスに調整します。
- **動的データ・フレームワーク**: 柔軟なデータ定義とエントリー・ソースにより、複数のタッチポイントからのさまざまな形式 (ファイル、API など) のコンテキスト・データとイベントでカスタマー・ジャーニーを強化します。

Note: 分散型 (クラスタ) Journey セットアップでは、Journey Web とエンジンを同じマシンにインストールしないことをお勧めします。Web とエンジンを別々のマシンにインストールすることをお勧めします。

Unica Journeyの利点

Unica Journeyの利点ガ以下の通り:

- **ブランド ロイヤルティの向上**: ターゲットを絞り自動化されたジャーニーで顧客を獲得、育成、変換、維持することにより、ブランド フォローを強化します。
- **オムニ チャネル エンゲージメントの強化**: アウトバウンド (Unica Campaign) およびインバウンド エンゲージメント (Unica Interact、Unica Discover、および Unica Deliver) のネイティブ統合により、チャネル全体で一貫したカスタマー エクスペリエンスを提供します。

- **顧客コンバージョン サイクルの短縮**: 一歩先を行き、タイムリーなネクスト ベスト アクションで顧客を目標に導きます。
- **瞬間的な反応**: 顧客がジャーニーのどこにいるかを知る機会を逃さず、適切な体験で顧客を喜ばせます。
- **マーケティングの TCO の削減**: Unica Link を備えたオープンで柔軟性あるフレームワークを通して、自動化されたフローとプラグ アンド プレイにより MarTech エコシステムへ統合することにより、マーケティングの TCO を削減します。

Chapter 2. 新しいナビゲーション UI

Unica は新しい UI を導入しました。

1. 管理者は、**「ユーザー設定」 > 「設定の編集」**から、任意のユーザーの UI を切り替えることができるようになります。
2. 画面上部にあるトグル・ボタンを使用すると、従来型 UI (古い UI) と新型 UI (新しい UI) を切り替えることができます。

NPN

Unica Platform は、Journey 用の共通のアクセス・ポイントとユーザー・インターフェースを備えています。共通インターフェースには以下の機能があります。

- 複数の Unica 製品がインストールされている場合、新規ウィンドウを起動せずに製品間で移動できます。
- **「最新」**メニューを使用して、最近訪問したページのリストを表示し、それらの任意のページに移動できます。
- **お気に入り**アイコンをクリックしてページをお気に入りに設定すると、ページが**お気に入り**リストに表示されます。
- フォルダ、ジャーニー、エントリー・ソース、データ定義、テンプレートの検索機能にアクセスできます。すべてのフォルダには検索フィールドが使われています。例えば、Journey 内部でエントリー・ソースのリストを表示している場合、検索はすべてのエントリー・ソースを対象に実行されます。

Chapter 3. Journey マルチ・パーティション

V12.1.7 以降、Journey はマルチ・パーティションをサポートしています。12.1.7 にアップグレードするユーザーの場合、システムに複数のパーティションが存在すると、Journey アプリケーション内のさまざまなエンティティー間でパーティションを横断した参照が発生する可能性があります。

パーティションを横断する参照の例:

partition1 のユーザーによって作成されたエントリー・ソース ES001 は、Partition2 のユーザーによって作成された Journey JS0001 で使用されています。

アップグレードの前に、管理者は次のクエリーを実行する必要があります。

select * from organisations

このクエリーで返されるレコードが 1 つだけで、id 列の値が 1 の場合、このようなシステムは 12.1.7 に直接アップグレードできます。そうでない場合、ユーザーはサポート・チームに連絡して、システム内にパーティションが存在する場合は、パーティションを横断する参照をチェックおよび修正する必要があります。

12.1.7 より前のバージョンでは、ユーザーは他のパーティションに属しているエンティティーにアクセスできました。

- 12.1.7 以降、ユーザーのビューは自分のパーティションのエンティティーに制限されます。たとえば、partition1 のユーザーは、自分のパーティションに属するエントリー・ソース、データ定義、テンプレート、ジャーニー、フォルダのみを表示できます。

  他のパーティションのエンティティーに何らかの方法でアクセスすると、エラー 404 (not found) が発生します。これは、悪意のあるユーザーが Journey の REST API を呼び出し、パーティションを横断してリソースにアクセスしないようにするためです。
- Journey は、別のパーティションに属するエントリー・ソースを介して送信されるデータを受け入れません。この場合、システムはエラーをログに記録します。ただし、特定のエントリー・ソースが複数のジャーニーで使用され、その一部がエントリー・ソースのパーティションに属している場合、そのデータはこれらのジャーニーにのみ送信されます。
- ジャーニーは、公開アクション・タッチポイントを使用して、異なるパーティションに属するエントリー・ソースにデータを公開することはできません。このような場合、システムはエラーをログに記録し、オーディエンスは次のタッチポイントに移動します。
- クライアント ID とシークレットのペアを使用して、/api/entry sources/rest/data を介してオーディエンス・データを入力する場合、そのペアはエントリー・ソースのパーティションに属している必要があります。それ以外の場合、システムは 404 Not Found エラーを表示します。このような場合、管理者はエントリー・ソースのパーティションのユーザーとしてログを記録し、新しいクライアント ID とシークレットのペアを生成する必要があります。ダウンストリーム・アプリケーションの既存のクライアント ID/シークレット・ペアのすべての参照も修正する必要があります。
- ユーザーは、他のパーティションのエントリー・ソース参照またはデータ定義の参照を持つジャーニーをコピーできません。それ以外の場合は、404 Not Found エラーがユーザーに表示されます。

プラットフォーム構成における構成変更。

12.1.7 のクリーン・インストールまたは 12.1.7 へのアップグレードでは、新しいノード・**パーティション**が Journey アプリケーションのノードに追加されます。管理者は、パーティション・テンプレート・ノードをクリックし、パーティション名を指定して、1 つ以上のパーティションを作成する必要があります。パーティションの数とその名前は、Campaign、Deliver など、他のインストール済み Unica 製品に存在するパーティションと一致する必要があります。

パーティション・ノード (partition1 など) を追加する場合、ユーザーがパーティション・ノード (`Affinium|Journey|partitions|partition1` など) をクリックすると、次の属性が表示されます。

- Link_Configured (デフォルト No)
- Deliver_Configured (デフォルト No)
- Contact_Central_Configured (デフォルト No)
- Dedup_End_Audience (デフォルト No)
- Goal_Max_Allowed (デフォルト 15)
- Mask_Exported_Audience_Data (デフォルト No) - このプロパティーはエンジンに固有です。このプロパティーが **Yes** に設定されている場合、オーディエンスの詳細のマスク可能なフィールドがマスクされ、公開アクション・タッチポイントでオーディエンスの詳細が CSV または Kafka トピックに公開されます。Full_Mask_Exported_Audience_Data プロパティーが **Yes** に設定されている場合、オーディエンス詳細のマスク可能なフィールドは完全にマスクされます。
- Full_Mask_Exported_Audience_Data (デフォルト No) - このプロパティーは、Web とエンジンの両方に適用されます。Journey Web の場合、オーディエンス詳細ポップアップが表示され、ジャーニー・キャンバスの最初の場所でクリック可能な場合、このプロパティーが **Yes** に設定されているときは、マスク可能なフィールドは完全にマスクされ、このプロパティーが **No** に設定されているときは、部分的にマスクされます。

  Journey エンジンの場合、公開アクション・タッチポイントでオーディエンス・データが CSV ファイルまたは Kafka トピックに公開され、Full_Mask_Exported_Audience_Data と Mask_Exported_Audience_Data の両方のプロパティーが **Yes** に設定されている場合、そのデータ内のマスク可能なフィールドの値は全部がマスクされ、プロパティーが **No** に設定されている場合は一部がマスクされます。

次のノードが各パーティション・ノードに追加されます

Journey 構成

- Journey_Waittime_Configurations
- Sync_Contact_and_Response_Tables
- Validations_On_Journey_Records
- Validation_Headers - `Affinium|Journey|partition|partition1|Journey_Configurations|Validation_Headers` の下の Journey 構成で構成されているヘッダーとその値に応じて、オーディエンス詳細ポップアップを有効/無効にします。

  前述のすべてのヘッダーが http リクエストで見つかり、各ヘッダーに構成で指定されたサポート対象値のうち少なくとも 1 つの値が含まれている場合は、ユーザーにオーディエンス詳細ポップアップが表示されます。ヘッダー名と値は大文字と小文字を区別しません。つまり、大文字と小文字を考慮せずに一致します。ヘッダー名、サポート

されている値、およびフィールド区切り文字 (複数の値の場合) で構成される各トリプレットでは、管理者はフィールド区切り文字を自由に選択できます。値の区切り文字フィールドを空白のままにすると、値全体が単一のサポートされる値と見なされます。

    - Validation_Header_Template
- リンク構成
- Deliver 構成
- 統合

Note: application.properties の設定、およびプラットフォーム構成のグローバル設定 (すなわちパーティション外の設定) は、お客様の参照のために保持されています。

Journey_Waittime_Configurations

- Engagement_Split_Waittime
- Decision_Split_Waittime
- Max_No_Of_Days_In_Delay_Evaluation
- Max_No_Of_Days_In_Delay_Expression

Sync_Contact_and_Response_Tables

- Sync_Contact_and_Response_Tables

Validations_On_Journey_Records

- Validations_On_Journey_Records - [デフォルト値は Yes]

リンク構成

- Link_URL - [change_me]
- Link_Data_Source_User - [change_me]
- Link_Data_Source_Name - [change_me]
- LinkProjectName - **_app_journey** [デフォルト - すべてのパーティションでそのまま維持する必要があります]
- Application - **journey**[デフォルト - すべてのパーティションでそのまま維持する必要があります]

Note: 12.1.8 リリース以降、Journey はマルチ・パーティション・リンク・サポートに対応しています。ユーザーは、複数の Journey パーティションで同じ Link インスタンスを構成できます。

各 Journey パーティション・ノードで、ユーザーは次のようにリンク情報を設定する必要があります。

1. Journey <Partition1> ノードの場所 -> (Affinium|Journey|partitions|<partition1>|Link_Configurations) に移動して次のパラメーターを設定します。

   Link_URL: https://Linkhostname.com

Link_Data_Source_User: <partition1_user>

Link_Data_Source_Name: <LinkDS>

LinkProjectName: _app_journey [デフォルト-すべてのパーティションでそのまま維持する必要があります]

Application: journey [デフォルト-すべてのパーティションでそのまま維持する必要があります]

次に、**「設定」 > 「ユーザー」 > <Partition1_user> > 「データ・ソースの編集」** に移動します。

Partition1 ユーザーの <LinkDS> データ・ソースを追加します

2. Journey<Partition2>ノードの場所 -> (Affinium|Journey|partitions|Partition2|Link_Configurations) に移動して次のパラメーターを設定します。

   Link_URL:https://Linkhostname.com

   Link_Data_Source_User: <partition2_user>

   Link_Data_Source_Name: <LinkDS1>

   LinkProjectName: _app_journey [デフォルト-すべてのパーティションでそのまま維持する必要があります]

   Application: journey [デフォルト-すべてのパーティションでそのまま維持する必要があります]

   次に、**「設定」 > 「ユーザー」 > <Partition2_user> > 「データ・ソースの編集」** に移動します。

   Partition2 ユーザーの <LinkDS1> データソースを追加します

Deliver 構成

- Deliver_URL：有効な Deliver TMS URL [change_me] を構成する必要があります。

Note: `Affinium|Journey|Deliver_Configurations|Unsubscription_Attribute` 構成はグローバルなままです。つまり、パーティション・レベルには存在しません。

統合

- preferredCognitiveServiceProvider

Note:

V12.1.7 以降、Journey エンジンは起動時にプラットフォームからパーティション設定も読み取ります。パーティション・ノード内の設定を変更した場合は、Journey Web および Journey エンジンを再起動する必要があります。

12.1.7 にアップグレードするには、プラットフォームを 12.1.7 にアップグレードする必要があります。

Chapter 4. ジャーニー オーディエンスのクリーンアップ

V12.1.7 以降では、ジャーニー オーディエンスをライブで実行しているジャーニーから削除することができます。cron ジョブがスケジュールされます (顧客は必要に応じて cron 式を構成できます)。このジョブは、ステータスが JOURNEY_COMPLETED および JOURNEY_ENDED のすべてのオーディエンスを検索し、それらのオーディエンスとその関連データ (オーディエンス応答、マイルストーン、目標などのデータ) をそれぞれのテーブルから削除します。この削除されたデータは回復できません。これは、ジャーニー エンジンの全体的なパフォーマンスの向上に役立ちます。これをサポートするために、エンジンの application.properties ファイルに次のプロパティーが導入されました。

1. cleanup.audience.enable=true/false - ジャーニー・データのクリーンアップを、それぞれ有効/無効にします。

   Note: この設定は、削除されたジャーニーのオーディエンスと、その関連データを、ジャーニーのシステム・データベース・テーブルから削除するために使用されます。

   cleanup.audience.enable=true の場合、ジャーニーのオーディエンス数に対する副次的影響は次のとおりです。

   a. **ジャーニー統計のカウント**

      (archival.audience.ttl=x) 日よりも前にジャーニーにオーディエンスが存在していた場合、これらのオーディエンスのデータは削除されるため、ジャーニー開始ノードのカウントはそれに応じて減少します。

      オーディエンス・レコードが削除されると、それぞれの応答 (ある場合) も削除されるため、タッチポイント (E メール、SMS、プッシュ、WhatsApp、REST、JDBC、ループ、Salesforce など) もそれに応じて減少します。

      Note:

      - オーディエンスの公開タッチポイント統計は、上記の削除による影響を受けず、変わりません。
      - オーディエンスの Adtech タッチポイント統計は、上記の削除による影響を受けず、変わりません。
      - ジャーニー・キャンバス・ブランチを横断したジャーニー・オーディエンス・フロー・カウントは、影響を受けません。
      - **(archival.audience.ttl=x)** 日以内にジャーニーに入ったオーディエンスは影響を受けず、ジャーニーの統計と開始ノードにそのまま残ります。

   b. **ジャーニーの目標と目標の分析**

      - **(Archival.audience.ttl=x)** 日よりも前に存在していたオーディエンスにより目標が達成されたジャーニー - 削除後、目標達成数は、それに応じて減少します。
      - **頻度に基づく目標と頻度に基づかない目標** - オーディエンスの応答が削除されるため、達成された目標もそれに応じて減少します。したがって、設定された目標は再度達成する必要があります。

- **頻度に基づかない目標** - 特定の目標に対してマークされた目標達成基準がある場合、ジャーニーは設定された目標を達成した後に完了します。
- **ジャーニー目標分析ページ** - すべての目標のグラフは、ジャーニー・オーディエンスの削除後の新しい目標達成数に従って表示されます。

c. **「ジャーニー・オーディエンス分析」タブ**

- **(archival.audience.ttl=x)** 日よりも前にジャーニーにオーディエンスが存在していた場合、これらのオーディエンスのデータは削除されるため、「ジャーニー・オーディエンス分析」タブにおける数が減少し、それに応じてグラフが表示されます。

d. **ジャーニー・キャンバス・ページのマイルストーン**

オーディエンスの削除後、ジャーニー・キャンバス・ページのマイルストーン・カウントは、新しいオーディエンスがジャーニー・マイルストーンに到達したときにのみ更新され、マイルストーン・キャンバス・ページでは減少したカウントのみが表示されます。

e. **「マイルストーン分析」タブ**

(archival.audience.ttl=x) 日よりも前にジャーニーにオーディエンスが存在していた場合、「マイルストーン分析」ページのカウントは**「マイルストーン・キャンバス」**ページと同様に減少します。したがって、**マイルストーン達成者 (%)** が影響を受けます。

f. **[ジャーニー・パフォーマンス] タブ**

オーディエンスの削除後、E メール、SMS、プッシュの各タイルのパフォーマンス・タブにおけるカウントは影響を受けず、そのまま維持されます。

g. **ループ**

(archival.audience.ttl=x) 日よりも前にジャーニーにオーディエンスが存在していた場合、ループの詳細リンクに表示される応答数も、それに応じて影響を受けます。

2. archival.audience.enable=true/false - **journeyaudiences** から **endjourneyaudience** へのジャーニー・オーディエンスのアーカイブが、それぞれ有効/無効になります。

3. archival.audience.ttl=X - cleanup.audience.enable=true の場合、ジャーニーにおいて X 日より長く経過したジャーニー・オーディエンスおよびその関連データが **archival.audience.cron** で設定された頻度で削除され、ジャーニーにおいて X 日が経過していないジャーニー・オーディエンスおよびその関連データはジャーニー内に維持されます。

Note: archival.audience.ttl のデフォルト値は 365 日です。最小で 2 日に設定できます。2 日未満の値を追加した場合、このプロパティーでは 2 日に相当します。

4. archival.audience.cron=0 0/5 * * * ? -> ジャーニー・オーディエンスのクリーンアップは、この cron ジョブに設定された頻度に従って実行されます。

   例:

   cleanup.audience.enable=true

   archival.audience.enable=true

   archival.audience.ttl=3

   archival.audience.cron=0 0/5 * * * ?

   上記の例では、archival.audience.enable=true であるため、ステータスが Journey_Completed/Journey_Ended のオーディエンスは **journeyaudiences** テーブルから **endjourneyaudience** テーブルに移動します。また、cleanup.audience.enable=true および archival.audience.ttl=3 と設定されているため、設定された 3 日より長く経過した Journey_completed または Journey_ended のオーディエンスは、データベース・テーブル内に存在する関連データとともに、設定された頻度「archival.audience.cron=0 0/5 * * * ?」でクリーンアップされます。

Journey エンジンがクラスター・モードで展開されている場合:

1. 1 つのノードでアーカイブとクリーンアップの両方のプロセスを実行するには、その 1 つのノードで両方のフラグを true にします。
2. アーカイブとクリーンアップの両方のプロセスを無効にするには、すべてのノードで false に設定します。
3. 異なるエンジン・ノードでアーカイブとクリーンアップのプロセスを実行するには、これらのプロパティーを 2 つのいずれかのエンジン・サーバー・ノードで個別に true にします。

   例: クラスターに 4 つのエンジン・ノードがあると仮定します。2 つの異なるエンジン・ノードでクリーンアップ・プロセスを実行する場合は、次の設定を行う必要があります。

   cleanup.audience.enable=true server 1

   archival.audience.enable=false server 1

   cleanup.audience.enable=false server 2

   archival.audience.enable=true server 2

   cleanup.audience.enable=false server 3

   archival.audience.enable=false server 3

cleanup.audience.enable=false server 4

archival.audience.enable=false server 4

Note:

何百万ものジャーニー・オーディエンスとその関連データを削除している間に、QueryTimeOut 例外が発生する場合があります。これを回避するには、Journey 12.1.7 をインストールした直後、またはこのバージョンにアップグレードした直後、あるいは Journey エンジンを開始する前に、以下のインデックスを作成することが推奨されます。

- LoopAudienceFlow(audienceId) にインデックス MY_UJR_IDX_LAF を作成します
- BatchIDAudiencedataMap(audienceId) にインデックス MY_UJR_IDX_BIDADM を作成します
- JourneyAudienceGoal(audienceId) にインデックス MY_UJR_IDX_JAG を作成します
- AudienceResponseHTTPDetail(audienceResponseId) にインデックス MY_UJR_IDX_ARHD を作成します
- AudienceResponseExtended(audienceResponseId) にインデックス MY_UJR_IDX_ARE を作成します
- AudienceResponseMetaData(audienceResponseId) にインデックス MY_UJR_IDX_ARMD を作成します。
- AudienceResponseInteraction(audienceResponseId) にインデックス MY_UJR_IDX_ARI を作成します
- AudienceResponse(audienceId) にインデックス MY_UJR_IDX_AR を作成します。
- JourneyAudienceFlow(audienceId) にインデックス MY_UJR_IDX_JAF を作成します
- AudienceWaitState(audienceId) にインデックス MY_UJR_IDX_AWS を作成します
- JourneyDeliverResponseMaster(audienceId) にインデックス MY_UJR_IDX_JDRM を作成します。
- JourneyAudienceMilestone(audienceId) にインデックス MY_UJR_IDX_JAM を作成します

Chapter 5. Unica Journey 統合

電子メール用 Unica Journey 実行エンジン

Unica Journey は、電子メール配信のために Unica Deliver と Unica Link をサポートしています。Journey との統合には、どちらを使ってもかまいません。

Unica Journey と Unica Link の統合

Unica Link は、電子メール、SMS、CRM、ADTECH、および JDBC の各チャネルを介してコミュニケーションを送信する機能を提供します。Unica Link には、電子メール、SMS、CRM、ADTECH、および JDBC の各チャネルへコミュニケーションを配信するために、次のリファレンスコネクタが用意されています。

以下のリファレンスコネクタをお好みで取り付けてください。

- **MailChimp** - 電子メール用
- **Mandrill** - 電子メール用
- **Twilio** - SMS 用
- **Salesforce** - CRM 向け

Unica Link との統合により、Journey は、電子メール、SMS、CRM、ADTECH、JDBC 実行でのみ、あらゆるサードパーティ・ベンダーと統合することができます。

Table 1. Unica Link のインストールと構成

タスク	ドキュメンテーション
Unica Link のインストールと設定	『*Unica Link V12.1* インストール・ガイド』を参照してください。
Journey の Unica Link コネクタアプリをインストールします。	『*Unica Link V12.1* インストール・ガイド』を参照してください。
Unica Link コネクタのインストール - MailChimp	『*Unica Link Mailchimp Connector* ユーザー・ガイド』を参照してください。
Unica Link コネクタのインストール - Mandrill	『*Unica Link Mandrill Connector* ユーザー・ガイド』を参照してください。
Unica Link コネクタのインストール - Twilio	『*Unica Link Twilio Connector* ユーザー・ガイド』を参照してください。
Unica Link コネクタのインストール - Salesforce	『*Unica Link Salesforce Connector* ユーザー・ガイド』を参照してください。

Note: HCLは、これらのデリバリーチャネルベンダーのアカウントやアクセスを提供するものではありません。あなたの好みに基づいて、これらのベンダーから権利やアカウントを取得することができます。

Unica Journey と Unica Deliver の統合

Unica Journey は、Unica Deliver の機能を利用して、電子メール・コミュニケーションを送信します。これは、リアルタイムで電子メールの応答をキャプチャーし、オーディエンス Journey を処理するのにも役立ちます.Unica Deliver と Unica Journey, の統合の有効化について詳しくは、『*Unica Journey* インストール・ガイド』を参照してください。

Unica Journey の Unica Campaign および Unica Interact との統合

Unica Journey は、Unica Campaign および Unica Interact とシームレスに統合します。Unica Campaign および Unica Interact は、特定の Kafka トピックについてオーディエンス・データを Unica Journey に送信します。視聴者データはKafkaのエントリーソースを通じて送信され、これらのエントリーソースからデータを消費するすべてのジャーニーにプッシュされます。

Unica Journey と Unica Campaign および Unica Interact の連携については、以下のドキュメント・マップに記載されているガイドを参照してください。

Journeyは、Campaignの複数のパーティションからのデータをサポートします。

ジャーニーサポートデータを複数のパーティションに分割してキャンペーンを実施。

1. Journeyアプリケーションは、マルチパーティションに対応していません。
2. ジャーニーで処理できるのは、Campaign/Interact/Deliverの複数のパーティションのデータのみです。このジャーニーでは、1つのパーティションで実行されます。

構成プラットフォームとユーザーの役割と権限を変更する必要があります。

- エントリーソースの下に表示されるキャンペーンフローチャートの詳細は、複数のパーティションからのものです。
- パーティションに基づき、メール/SMS/WhatsAppのタッチポイントにDeliverテンプレートが表示されます。

Table 2. Unica Campaign と他の HCL 製品との統合

タスク	ドキュメンテーション
Unica Campaign と Unica Journey の統合	『*Unica Campaign* 管理ガイド』および『*Unica Campaign* ユーザー・ガイド』を参照してください。
Unica Campaign と Unica Interact の統合	『*Unica Interact* 管理ガイド』を参照してください。

Unica Journey とUnica Discover の連携

Unica Journey は、Unica Discover とシームレスに統合します。Unica Discoverは、オーディエンスの闘争データをUnica Journeyに送信します。視聴者データはRESTの入力ソースを通じて送信され、これらの入力ソースからデータを消費するすべてのジャーニーにプッシュされます。4つのスクリプトが提供されますので、Journeyをインストールした後、すぐにス

クリプトを実行する必要があります。これにより、2つのエントリー・ソースと、CART用Discover Entryソースとフォーム用Discover Entryソースという2つのデータ定義が作成されます。スクリプトは次のとおりです。

- Discover-MariaDB.sql
- Discover-MS-SQL.sql
- Discover-OneDB.sql
- Discover-Oracle.sql

DD 名	カート
説明	顧客がカートや選択した商品セットを放棄した場合、このイベントをトリガーすることができます。

Table 3. 送信する属性

名前	タイプ	長さ	注記
メール*	TEXT	200	これは必須フィールドです。
名前	TEXT	200	
DiscoverSessionId	TEXT	50	Discover Session IDはリンクバックするために必要です。
CartId	TEXT	50	カートを識別するための一意なID。
カートバリュー	NUMBER		
イベント日時	TIMESTAMP		イベントの日付と時刻（UTC） 経度
EventType	TEXT		イベントタイプはCART_ABANDONEDにすることができます。
クッキーID	TEXT	1024	
TLT_BROWSER	TEXT	50	ブラウザの詳細
TLT_MODEL	TEXT	50	デバイスの詳細
HTTP_ACCEPT_LANGUAGE	TEXT	50	言語

DD 名	フォーム
説明	顧客がWebフォームに入力すると、このイベントを公開することができます。

Table 4. 送信する属性

名前	タイプ	長さ	注記
メール*	TEXT	200	これは必須フィールドです。
名前	TEXT	200	
DiscoverSessionId	TEXT	50	Discover Session IDはリンクバックするために必要です。
FormId	TEXT	50	フォームを識別するための一意なID
フォーム名	TEXT	100	
イベント日時	TIMESTAMP		イベントの日付と時刻（UTC） 経度
クッキーID	TEXT	1024	
TLT_BROWSER	TEXT	50	ブラウザの詳細
TLT_MODEL	TEXT	50	デバイスの詳細
HTTP_ACCEPT_LANGUAGE	TEXT	50	言語
EventType	TEXT		イベントタイプはFORM_SUBMITTED, FORM_ABANDONEDのいずれかになります。

Note: Fixpack 3以降では、Unica JourneyとUnica Discover機能の連携が可能になります。

Unica Deliver の概要

Unica Deliver は、Web ベースのエンタープライズ規模のマーケティング・メッセージ・ソリューションです。これを使用することで、アウトバウンド一括メッセージングおよびトランザクション・メッセージング・キャンペーンを実行できます。Deliverは、Unica Campaign と統合されており、さらに Unica でホストされるセキュアなメッセージ作成、送信、および追跡リソースと統合されています。

Deliver を使用すると、パーソナライズされた電子メール通信の作成、送信、および追跡ができます。Deliver は Campaign をインストールして操作するので、Campaign フローチャートを使って受信者情報を正確に選択およびセグメント化し、各メッセージをカスタマイズすることができます。

オーディエンスを選択します

Campaign を使用して、各メッセージをパーソナライズするために使用できるメッセージ受信者と各人に関するデータを選択します。

Deliver を使用すると、多数の電子メール受信者にすばやく個別に連絡できます。ただし、トランザクションに応答して単一の電子メール メッセージを自動的に送信するようにメーリングを構成することもできます。

メッセージを作成します

Deliver Document Composer には、パーソナライズされたメッセージ・コンテンツの設計、プレビュー、および公開に使用できる編集ツールが用意されています。Document Composer にアップロードしたコンテンツを含むメッセージを作成したり、Deliver がメッセージをビルドおよび送信するときに外部コンテンツにリンクすることができます。Deliver には、各受信者の個人データに基づいて、条件付きでコンテンツを表示するメッセージを設計する方法がいくつか用意されています。

メッセージを送信して応答を追跡する

目標に応じて、メッセージング キャンペーンをできるだけ早く実行するようにスケジュールしたり、後で実行するようにスケジュールすることができます。Deliver は、メッセージ配信を監視し、受信者の応答を追跡します。システムは、Campaign データベース・スキーマの一部としてインストールされる Deliver システム・テーブルにコンタクトおよび応答データを返します。

使用方法

開始するには、Campaign をインストールし、ホストされたメッセージング・アカウントを用意する必要があります。

システム管理者は、ホストされたメッセージング・アカウントを要求し、Unica を操作して、リモートメッセージングおよび追跡システムへのセキュアなアクセスを構成する必要があります。一部のメッセージング機能は、Unica への要求があった場合にのみ利用できます。ホストされたメッセージング・アカウントの確立と Unica がホストするメッセージングへのアクセスの構成について詳しくは、『*Unica Deliver* スタートアップおよび管理者ガイド』を参照してください。

Unica Deliver 統合

Unica Deliver と Unica Journey を統合するには、Unica Platform で次の構成を行う必要があります。

1. Unica Platform で、**「設定」 > 「構成」**に移動します。
   Result
   「構成カテゴリー」ページが表示されます。
2. **Journey** を選択します。
   Result
   「'Journey' の設定」ページが表示されます。
3. **「設定の編集」**を選択します。
   Result
   「(Journey)」ページが表示されます。
4. 次の手順を実行します。

a. **「Deliver_Configured」**フィールドで**「はい」**を選択します。
b. **「変更を保存」**をクリックします。

5. 展開されたジャーニー ノードで、**「Deliver_Configurations」**を選択します。
   Result
   「'Deliver_Configurations' の設定」ページが表示されます。
6. **「設定の編集」**を選択します。
   Result
   [(Deliver_Configurations)]ページが表示されます。
7. 次の手順を実行します。

a. 以下のフィールドに値を指定します。
- **Deliver_URL**: Deliver に対して構成された URL。

Note:

i. ユーザーは、Journey の構成で TMS URL を更新する必要があります。SOAP TMS URL はサポートされておらず、REST URL のみがサポートされているため、TMS URL を更新します。**「Platform」 > 「設定」 > 「構成」 > 「Journey」 > 「Deliver_URL」**に移動します。

ii. V12.1.6 以降では、古い **Deliver API URL** を新しい **Deliver REST API URL** に更新する必要があります。アップグレード後は、Deliver タッチポイントを含むすべてのジャーニーを再公開してください。

場所: `(Affinium|Journey|Deliver_Configurations)`

新しい Deliver URL すなわち http://<host-name>/delivertms/api/deliver/rest/v2/tms

- **Deliver_Partition**: **Deliver_URL** にアクセスする資格証明が格納されているパーティションです。
- ジャーニーに Deliver タッチポイントが構成されている場合、ユーザーはジャーニーを一時停止して再公開する必要があります。その後、オーディエンスのみが処理を開始します

b. **「変更を保存」**をクリックします。

Kafkaの統合

Journey ノードに関して、Unica Platform で Kafka を設定する必要があります。

Unica Platform でKafka_Configurationsにアクセスする

Kafka_Configurationsにアクセスするには、次の手順を実行します。

1. Unica Platform で**「設定」 > 「構成」**に移動します。
2. **Journey** ノード を展開します。
3. **Kafka_Configurations** を選択します。
4. **「設定の編集」**を選択します。

CommunicationMechanism 値に基づく必須構成。

(Kafka_Configurations) ページで、CommunicationMechanism フィールドに次の値のいずれかを選択できます。

- NO_SASLPLAINTEXT_SSL
- SASL_PLAINTEXT
- SSL
- SASL_PLAINTEXT_SSL

選択した内容に応じて、以下の項目が必須となります。

フィールド名	NO_SASLPLAINTEXT_SSL	SASL_PLAINTEXT	SSL	SASL_PLAINTEXT_SSL
KafkaBrokerURL	あり	あり	あり	あり
TopicName	あり	あり	あり	あり
sasl.mechanism		あり		あり
ユーザーForKafkaData		あり	あり	あり
sasl.jaas.config.dataSource		あり		あり
truststore.location			あり	あり
truststore.password.dataSource			あり	あり
keystore.location			あり	あり
keystore.password.dataSource			あり	あり
key.password.dataSource			オプション	オプション
ssl.endpoint.identification.algorithm			あり	あり

必要な設定を行い、**「変更を保存」**をクリックします。

Note: Kafkaログファイルのサイズが大きいため、ディスクストレージが不足し、Kafkaサーバーを突然シャットダウンしたこと。

Note: 一度オーディエンス タイプを構成に使用すると、他の構成には再度使用できません。CIF を構成しているときに 1 つのオーディエンス タイプを使用すると、同じオーディエンス タイプを同じエントリー ソースで再度使用することはできません。

Kafka トピック

Kafka の下には、Journey エンジンが始動するとトピックが作成されます。

Kafka トピック	説明
DATA_CLEAN	受信データのフィルタリングに関するクリーンアップ・ルールを適用するためにデータ・クリーンアップ・サービスで使用されます。
JOURNEY_EVENT	ジャーニー・キャンバス・フローをオーケストレーションするためにオーケストレーション・サービスで使用されます。
Kafkalog	非同期ロギングに使用されます。
JOURNEY_PARSER	ジャーニー・キャンバス・デザインを解析するためにジャーニー・パーサー・サービスで使用されます。
DATA_FETCH	12.1.3 より後では使用されていません。
DELAYV2	公開されたオーディエンスに遅延を適用するために遅延サービスによって消費されます。
END	オーディエンスのジャーニーが終了したときに Journey によって使用されます。
ASYNC_DATABASE_STREAM_DATALOG	
ASYNC_DATABASE_AUDIENCE_RESPONSE	audienceresponse テーブルにオーディエンス応答データを挿入するためにエンジンによって使用されます。
ASYNC_DATABASE_DISCARD_DATA	廃棄されたオーディエンス・データを処理するために Journey によって使用されます。
ASYNC_DATABASE_AUDIENCE_FLOW	journeyAudienceFlow テーブルに統計レコードを挿入するためにエンジンによって使用されます。

Kafka トピック	説明
ASYNC_DATABASE_DATA_ERRORS	エラーを処理するために Journey で使用されます。
ASYNC_DATABASE_AUDIENCE_BULK_REPONSE	オーディエンス応答データを Adtech 用のオーディエンス一括応答テーブルに挿入するために、エンジンによって使用されます。
ASYNC_DATABASE_UNSUBSCRIBE_EMAIL	Journey からの電子メールの購読登録を挿入するためにエンジンによって使用されます。
SMS_SEND	設定されたチャネルを介して、公開されたオーディエンスに SMS を送信するために SMS サービスによって使用されます。
SALESFORCE_SEND	Link を介して Salesforce にデータを送信するために使用します。
JOIN	このトピックはエンジンによって使用され、結合コントロールを介してあるノード (先行ノード) と別のノード (後続ノード) を接続することで、これらのノード間でオーディエンスを送るためにだけに使用されます。
LOOP	このトピックは、任意のループ・スタート ノードの先行ノードからオーディエンスを受信するためにエンジンによって使用されます。ループ制限の構成に従って、オーディエンスを反復します。
ENGAGEMENT_SPLIT	オーディエンスを利用するためにエンゲージメント・スプリット・サービスによって使用されます。
JOURNEY_GOAL	データを消費するために目標サービスによって使用されます。
INCOMING_RESPONSES	エンジンによって消費される、Link によって公開された応答
PUBLISH_ACTION_POINT	ジャーニー、ノード、およびオーディエンスの詳細を取得するために、エンジン publishactionpointservice によって使用されます。
DELIVER_RESPONSES	エンジンによって消費される、Deliver RCT モジュールによって公開された応答。
ADTECH_SEND	Adtech 用にオーディエンスを処理するために Adtech サービスによって消費されます。
ADD_MILESTONE	

Kafka トピック	説明
PROCESS_MILESTONE	
WHATSAPP_SEND	設定されたチャネルを介して、公開されたオーディエンスにメッセージを送信するために WhatsApp サービスによって使用されます。
JDBC_SEND	公開されたオーディエンス ID に対して、JDBC タッチポイント用にアクションを実行するためにエンジンによって使用されます。
REST_INIT	使用されていません
REST_API	ジャーニーで REST タッチポイントを設定した後に、エンジン RestAPIService によって使用されます。
CHRH	12.1.5 より後では使用されていません
DELAY_UPDATE	遅延しているオーディエンス・データを更新するためにエンジンによって使用されます。
CALLABLE	
NOTIFICATION	
RESTREQUEST	REST リクエストを送信するためにジャーニーで使用されます。
RESTEXECUTE	セキュアおよび非セキュアのサードパーティ API を呼び出すために、エンジン RestExecutionservice によって使用されます。
RESTRESPONSE	REST サービスから受信した応答を消費するために Journey によって使用されます。
RESTUPDATE	REST サービスから受け取った応答を処理し、ジャーニーのオーディエンス詳細を更新するために、Journey によって使用されます。
RESTAUDIENCERESPONSE	オーディエンスの応答やデータを保存するために、エンジン RestAudienceResponseService で使用されます。
CIFINTEGRATION	外部システムから CIF によって送信されたデータを消費するために、Journey によって使用されます。
EMAILWAIT	エンジン EmailCapacityWaitService で使用されます。非コミュニケーション時間に要求が受信

Kafka トピック	説明
	されると EMAILWAIT になります (12.1.7 ユーザー・ガイド 31 ページを参照)。
SMSWAIT	エンジンの SmsCapacityWaitService で使用されます。非コミュニケーション時間に要求が受信されると SMSWAIT になります (12.1.7 ユーザー・ガイド 31 ページを参照)。
WHATSAPPWAIT	エンジン WhatsAppCapacityWaitService で使用されます。非コミュニケーション時間に要求が受信されると WHATSAPPWAIT になります (12.1.7 ユーザー・ガイド 31 ページを参照)。
NOTIFICATIONWAIT	エンジン PushNotificationCapacityWaitService で使用されます。非コミュニケーション時間に要求が受信されると NOTIFICATIONWAIT になります (12.1.7 ユーザー・ガイド 31 ページを参照)。
DELIVER_READER	公開されたオーディエンスに Deliver を介して通知を送信するために Deliver ヘルパー・サービスにより使用されます。
DEDUP	重複排除されたデータを消費するために Journey で使用されます。
ESEVENT	
RESPONSE_ACTION_EVENT	
PAUSE_AUDIENCE_CALC_COUNT	このトピックは、特定の一時停止ルールに対して一時停止するオーディエンス数を計算するように Journey エンジンに要求するために Journey Web で使用されます。
PAUSE_AUDIENCE_EXEC	このトピックは、特定のルールの一時停止実行プロセスを開始するように Journey エンジンに要求するために Journey Web で使用されます。
PAUSE_AUDIENCE_PROC	このトピックは、一時停止プロセスの 2 つの構成要素間で内部的に使用されます。一時停止処理のデータ抽出サービスは、一時停止するオーディエンスの情報を読み取り、このトピックを通じてこの情報を一括してライター・サービスに送信します。
JOURNEY_CONTROL	
DELIVER_RESPONSES_STAGE	CH/RH 応答を開始するためにエンジン サービスで使用されます。

パーティションへの Kafka レプリケーション

BATCH_IMPORT

バッチ・データを発行するすべてのエントリー・ソースは、CSV、TSC などのフラット・ファイルを介してこのトピックにメッセージを発行します。各メッセージは、単一のレコード群に対応している必要があります。

メッセージ属性

- エントリー・ソース ID
- ジャーニー ID
- データ
    - タイムスタンプ - メッセージがトピックに追加されたときのタイムスタンプ
    - エントリー・ソース・タイプ
    - FQN

オブジェクト・タイプ

- entrySourceID:
- journeyID:
- entrySourceType: enum
- データ: データを含んでいるファイルの FQN
- timestamp:

STREAMING_IMPORT

ストリーミング・データを発行するすべてのエントリー・ソースは、一連の個別レコードを介して、このトピックにメッセージを発行します。各メッセージは、単一のレコードに対応している必要があります。

メッセージ属性

- エントリー・ソース ID (存在する場合)
- データ (JSON 形式)
- メタデータ
    - タイムスタンプ - メッセージがトピックに追加されたときのタイムスタンプ
    - エントリー・ソース・タイプ
    - その他のメタデータ

オブジェクト・タイプ

- entrySourceID: (これはオプションです)
- entrySourceType: enum
- データ: (単一レコードの JSON)
- timestamp:
- metaData: (他のデータの JSON)

DATA_MAP

このトピックには、受信データが消去され、データ定義にマッピングする必要が生じるとメッセージが追加されます。

メッセージ属性

- エントリー・ソース ID
- ジャーニー ID
- データ
    - タイムスタンプ - トピックにメッセージが入力されたときのタイムスタンプ
    - データ (JSON)
    - レコード数 (バッチの場合)

オブジェクト・タイプ

- entrySourceID:
- journeyID
- entrySourceType: enum
- timeStamp
- データ: <JSON 文字列>
- numRecords:

RAW_DATA_FETCH

このトピックには、マップされていないデータを DATA ARCHIVE テーブルから取得するためのメッセージが含まれます。

メッセージ属性

- エントリー・ソース ID
- タイムスタンプ - トピックにメッセージが入力されたときのタイムスタンプ
- 状態 - 取得が必要なデータの状態 (RAW または CLEANED)
- レコード数 - 取得するレコードの数。指定されていない場合は、すべてのレコードです。

オブジェクト・タイプ

- entrySourceID:
- timeStamp
- state: enum
- numberRecords:

END_JOURNEY

このトピックには、一部のオーディエンスが定義されたジャーニーの論理的な終点に達している、さまざまな実行中のジャーニーに関するすべてのメッセージが保持されます。

JOURNEY_EVENTS

このトピックには、実行中の Journey インスタンスの有効期間中のさまざまなイベントに関するすべてのメッセージが保持されます。

メッセージ属性

- ID - これは、ジャーニー ID とオプションのノード ID の組み合わせです。
- データ
    - 状態 - ジャーニーの状態
    - Timestamp
    - メタデータ

オブジェクト・タイプ

- ID: journeyID
- state: enum:
- timeStamp
- データ:

PUBLISHED

- ID - ジャーニー ID
- データ
    - 状態 - PUBLISHED
    - タイムスタンプ - トピックにメッセージが入力されたときのタイムスタンプ
    - メタデータ - 完全なジャーニー構造を持つ JSON ファイルの FQN

オブジェクト・タイプ

- ID: journeyID
- state: enum:PUBLISHED
- timeStamp
- データ: ジャーニー構造の JSON (JSON のシリアル化形式にすることも可能)

PAUSE

- ID - ジャーニー ID
- データ
    - タイムスタンプ - トピックにメッセージが入力されたときのタイムスタンプ
    - 状態 - PAUSE

オブジェクト・タイプ

- ID: journeyID
- state: enum:PAUSE
- timestamp
- データ: <空>

JOURNEY_PAUSED

- ID - ジャーニー ID
- データ
    - タイムスタンプ - トピックにメッセージが入力されたときのタイムスタンプ
    - 状態 - PAUSED

オブジェクト・タイプ

- ID: journeyID
- state: enum:PAUSED
- timestamp
- データ: <空>

PARSED

- ID - ジャーニー ID
- データ
    - タイムスタンプ - トピックにメッセージが入力されたときのタイムスタンプ
    - 状態 - PARSED

オブジェクト・タイプ

- ID: journeyID
- state: enum:PARSED
- timestamp
- データ: <空>

MAPPED

- ID - ジャーニー ID
- データ
- タイムスタンプ - トピックにメッセージが入力されたときのタイムスタンプ
- 状態 - PARSED

オブジェクト・タイプ

- ID: journeyID
- state: enum:MAPPED

- timestamp
- データ: <空>

FETCHED

- ID - ジャーニー ID
- データ
    - タイムスタンプ - トピックにメッセージが入力されたときのタイムスタンプ
    - レコード ID のリスト

オブジェクト・タイプ

- ID: journeyID#nodeID
- state: enum:FETCHED
- timestamp
- numberRecords:
- listRecords:

DELAY_EXECUTED

- ID - この ID は、ジャーニー ID と次のノードのノード ID です。
- データ
    - 状態 - DELAY_EXECUTED
    - タイムスタンプ - トピックにメッセージが入力されたときのタイムスタンプ
    - レコード ID のリスト

オブジェクト・タイプ

- ID: journeyID#nodeID
- state: enum:DELAY_EXECUTED
- timestamp
- numberRecords:
- listRecords:

EMAIL_SENT

- ID - この ID は、ジャーニー ID と次のノードのノード ID です。
- データ
    - 状態 - EMAIL_SENT
    - タイムスタンプ - トピックにメッセージが入力されたときのタイムスタンプ
    - レコード ID のリスト

オブジェクト・タイプ

- ID: journeyID#nodeID
- state: enum:EMAIL_SENT
- timestamp
- numberRecords:
- listRecords:

JOURNEY_PARSE

このトピックには、検証および解析が行われるジャーニーのすべてのメッセージが保持されます。

メッセージ属性

- ID - ジャーニー ID
- データ
    - タイムスタンプ - トピックにメッセージが入力されたときのタイムスタンプ
    - 状態 - PARSE
    - データ

オブジェクト・タイプ

- ID: journeyID
- state: enum:PARSE
- timestamp
- データ: <空>

JOURNEY_CACHE

このトピックには、キャッシュに追加する必要があるジャーニー・データのすべてのメッセージが保持されます。

メッセージ属性

- ジャーニー ID
- エントリー・ソース ID
- データ
    - タイムスタンプ - トピックにメッセージが入力されたときのタイムスタンプ
    - データ

オブジェクト・タイプ

- ID: journeyID
- state: enum:PARSE
- timestamp
- データ: <空>

ASYNC_DATABASE_INSERTS

このトピックには、データベース・テーブルに非同期で挿入されるすべてのメッセージが保持されます。

メッセージ属性

- エンティティー
- 名前
- メッセージ
- Timestamp

JOURNEY_MAP

このトピックには、ジャーニー・マップ・サービスに関するすべてのメッセージが保持されます。このデータは、ジャーニーに関連する各エントリー・ソースのすべての詳細で構成されます。

メッセージ属性

- ジャーニー ID
- データ
    - ID、タイプ、FQN を含む、ジャーニーのエントリー・ソースのリスト

JOURNEY_DATA

このトピックには、ログまたは状態情報のデータベース・テーブルにプッシュされるすべてのメッセージが保持されます。これらのメッセージは、ジャーニーの実行には影響しません。

メッセージ属性

- サービス ID
- データ
    - クエリー
    - Timestamp

EMAIL_SEND

このトピックには、電子メールを送信する必要があるジャーニーの実行中インスタンスからのすべてのメッセージが保持されます。

メッセージ属性

- ID - これは、ジャーニー ID とノード ID の組み合わせです。
- オーディエンス ID - 電子メールが送信される先のオーディエンス ID のリスト

OUTGOING_EMAIL

このトピックには、電子メールとして送信する必要がある Journey アプリケーションからのすべてのメッセージが保持されます。これらは、適切なアダプター・サービスまたはユニバーサル・コネクタによって動作します。

メッセージ属性

- sourceAppId - アプリケーションの名前 - Journey
- sourceInstanceId - タッチポイント ID
- Audiences - オーディエンス・データのリスト

EMAIL_RESPONSES

このトピックでは、構成済みコネクタを介して、送信された電子メールに対するオーディエンスの応答イベントとして配信エンジンから受信したすべてのメッセージが保持されます。

メッセージ属性

- タッチポイント ID
- オーディエンス ID
- イベント ID
- Timestamp

OUTGOING_SMS

このトピックでは、SMS として送信する必要がある Journey アプリケーションからのすべてのメッセージが保持されます。これらは、適切なアダプター・サービスまたはユニバーサル・コネクタによって動作します。

メッセージ属性

- sourceAppId - アプリケーションの名前 - Journey
- sourceInstanceId - タッチポイント ID
- Audiences - オーディエンス・データのリスト

SMS_RESPONSES

このトピックでは、構成済みコネクタを介して、送信された SMS に対するオーディエンスの応答イベントとして配信エンジンから受信したすべてのメッセージが保持されます。

メッセージ属性

- タッチポイント ID
- オーディエンス ID
- イベント ID
- Timestamp

DELAY

このトピックでは、次のタッチポイントを実行する前に遅延を追加する必要があるジャーニーの実行中インスタンスからのすべてのメッセージが保持されます。

メッセージ属性

- ID - これは、ジャーニー ID とノード ID の組み合わせです。
- 構成 - これは、遅延タッチポイントの構成データです。
- 構造 - これは、ジャーニー JSON から抽出された JSON 構造です。

DECISION_SPLIT

このトピックでは、一部のフィルターに基づいて受信データを分割する必要があるジャーニーの実行中インスタンスからのすべてのメッセージが保持されます。

メッセージ属性

- ID - これは、ジャーニー ID とノード ID の組み合わせです。
- 構成 - これは、決定の分割タッチポイントの構成データです。
- 構造 - これは、ジャーニー JSON から抽出された JSON 構造です。

JOURNEY_PAUSE

このトピックでは、特定のジャーニーを一時停止するためのオーケストレーション・サービスからのメッセージが保持されます。

メッセージ属性

- ID - 一時停止する必要があるジャーニー ID

DATA_PAUSED

このトピックでは、一時停止されたジャーニーの実行中インスタンスからのすべてのメッセージが保持されます。

メッセージ属性

- 名前 - ソース・トピックの名前
- ID - ジャーニー ID
- メッセージ - 元のメッセージ

JOURNEY_RESUME

このトピックでは、特定のジャーニーを再開するためのオーケストレーション・サービスからのメッセージが保持されます。

メッセージ属性

- ID - 再開する必要があるジャーニー ID

DATA_RESUMED

このトピックでは、一時停止状態にあったが、現在は個別のサービスにより再開する必要があるメッセージがすべて保持されます。

メッセージ属性

- 名前 - ソース・トピックの名前
- ID - ジャーニー ID
- メッセージ - 元のメッセージ

JOURNEY_ENGINE_ERRORS

このトピックでは、エラーが発生し、それを必要に応じて管理コンソールの Web 上に表示することができたエンジンからの、すべてのメッセージが保持されます。

メッセージ属性

- ID - エラーの原因となったエンティティーを識別する ID (ジャーニー ID、オーディエンス ID)
- 状態 - エラーの状態 (MEDIUM/HIGH/CRITICAL)
- カテゴリー - エラーを検出したサービスの名前
- テキスト - 実際のエラー・テキスト

JOURNEY_GOAL_VERIFICATION

このトピックでは、ジャーニーの目標が達成されたかどうかを検証する必要があるジャーニー目標検証サービスのすべてのメッセージが保持されます。

メッセージ属性

- ID - ジャーニー ID
- EVENT_TYPE - EMAIL RESPONSE、SMS_REPOSNE、PUSH_RESPONSE
- データ
- Timestamp

JOURNEY_GOAL_ACHIEVED

このトピックでは、ジャーニーが設定された目標を達成したときの通知を処理するサービスのメッセージが保持されます。

メッセージ属性

- ID - ジャーニー ID
- EVENT_TYPE - TIME、COUNT
- Timestamp

JOURNEY_ENGINE_MONITORING

このトピックは、監視データを Web に送信するためにエンジンによって使用されます。

メッセージ属性

- 名前 - サービスの名前は、サービスを参照している、またはメッセージが全体としてジャーニー・エンジンのものであることを示しているエンジンを参照しているメッセージです。
- 状態 - STARTING、GOING DOWN、NOT RESPONDING
- スレッド ID
- Timestamp
- IP アドレス

状態、タイプ、およびステータス

1. **エントリー・ソース・タイプ**
    - フラット・ファイル
    - REST
2. **データ・パイプラインのステータス**
    - インポート済み
    - 消去済み
    - マッピング
3. **ログのタイプ**
    - ファイル
4. **ログのサブ・タイプ**
5. **データ・パイプラインの状態**
6. **ジャーニーの状態**
    - PUBLISHED - ジャーニーがフロント・エンドから公開済み状態にプッシュされたとき
    - PARSED - ジャーニーが正常に解析されたとき
    - RUNNING - ジャーニーがデータをマッピングし、実行を開始したとき
    - PAUSED - ジャーニーが明示的または暗黙的に一時停止されたとき。現在のジャーニーの実行が完了します
    - STOPPED - ジャーニーが明示的または暗黙的に停止されたとき。現在のジャーニーの実行が中断されます
    - COMPLETED - ジャーニーが論理的な実行を完了したとき
    - GOAL ACHIEVED - ジャーニーが定義された目標を達成したとき
7. **オーディエンス・テーブルの状態**
    - NEW - 処理されていない、指定したジャーニー ID の新しいレコード
    - PROCESSED - 処理が行われた、指定されたジャーニー ID のレコード
    - COMPLETED - ジャーニーを完了 (論理的に終了) した、指定されたジャーニー ID のオーディエンス
    - GOAL ACHEIVED - 目標を達成したこれらのオーディエンスに関連付けられている ジャーニー

Chapter 6. Journeyの始まりと終わりのプロセス

About this task

プロセスを開始

1. プロセスウェブの開始
    a. KafkaとZookeeperの設定
        i. IP - Zookeeper/Kafkaが動作しているIP。
        ii. PORT- Kafka（デフォルト9092）、Zookeeperデフォルトポート2181
        iii. ログのパス
        iv. auto.create.topic.enable = true, このプロパティは、Engine Publishサービスを動作させるためにtrueに設定する必要があります。
    b. Zookeeperを起動し、10秒待つ
    c. kafkaを起動する
    d. configure Journey.xml --(Doc、Doc2参照)
    e. configure フォルダ下の Log4j2.xml を設定する。
    f. ウェブサーバー（JBOSS/TOMCAT/WebSphere）の起動
    g. Start Journey Web アプリケーション
2. エンジン始動
    a. application.portiesの設定
        i. DBの詳細を追加
        ii. Kafkaの詳細を追加する（例：spring.kafka.bootstrap-servers=127.0.0.1:9092, 127.0.0.2:9092)
            - Ignite Storageのパス、spring.ignite.storage.path、エンジンを実行するユーザーがIgniteフォルダのパスを読み書きできること。
        iii. configure フォルダ下の Log4j2.xml を設定する。
        iv. プロパティ spring.ignite.ipFinder.List を以下のように設定します。
            - ```
              spring.ignite.ipFinder.List=127.0.0.1:63501,127.0.0.1:63502,
              127.0.0.1:63503,127.0.0.1:63504
              ```
        v. エンジンの起動 (java -jar journeyEngine.jar)

停止処理（データを失わないための手順)

a. ウェブサーバーを停止する
b. エンジンを停止する - Director を使用します。または、Journey エンジン・プロセス ID を特定して、以下のコマンドを実行します
    - **Linux** → kill -SIGINT [engine-process-id] または kill -2 [engine-process-id]
    - **Windows** → taskkill /PID [engine-process-id]
c. Kafkaを停止する
d. Zookeeperを止める
              ```

Chapter 7. Journey ユーザーのロールと権限

Unica Journey を使い始める前に、ユーザーにロールと権限を割り当てる必要があります。

Note: 構成の変更には、Unica Journey の再起動が必要です。セキュリティー構成について詳しくは、『*Unica Platform* 管理者ガイド』を参照してください。

Journey ロールへの権限の割当

ユーザーに役割を割り当てる前に、使用可能な役割に権限を割り当てる必要があります。

About this task

Journey には、次の 2 つのユーザー・ロールが用意されています。

- **JourneyAdmin**
- **Journeyユーザー**

両方のロールにアクセス許可を割り当てるには、次の手順を実行します。

1. Unica Platform のホームページから **「設定」 > 「ユーザーのロールと権限」** を選択します。
   Result
   [ユーザーの役割と権限]ページが表示されます。
2. 左のパネルで、**「Unica Journey」 > 「partition1」** を展開します。
   Result
   partition1ページが表示されます。
3. **[権限の割り当て]**を選択します。
   Result
   (管理役割のプロパティ)ページが表示されます。
4. **「権限の保存および編集」**をクリックします。
   Result
   (パーティション 1 のアクセス許可)ページが表示されます。
5. **アプリケーション**を展開します。
6. 次のフィールドに値を設定します。

操作	JourneyAdmin のデフォルト設定	JourneyUser のデフォルト設定
データ定義の作成	あり	なし
データ定義の編集	あり	なし

操作	JourneyAdmin のデフォルト設定	JourneyUser のデフォルト設定
データ定義の削除	あり	なし
エントリ ソースの作成	あり	なし
エントリ ソースの編集	あり	なし
エントリ ソースの削除	あり	なし
作成 Journey	あり	あり
Journey の編集	あり	あり
Journey の削除	あり	なし
Journey の公開	あり	あり
Journey の完了	あり	あり
Journey の一時停止	あり	あり
目標の追加/変更/削除	あり	なし
目標ビュー	あり	あり
設定の追加・変更・削除	あり	なし
ビューの設定	あり	あり

Note:

- **JourneyAdmin** ロールの場合、権限を減らさず、デフォルトの権限を保持することをお勧めします。デフォルトでは、**JourneyAdmin** はすべての権限を持ちます。
- **JourneyUser** ロールには、適切と思われる権限を付与します。**JourneyUser** にはすべての権限を付与することができますが、これは推奨されません。

7. 権限を付与したら、**[変更を保存]** をクリックします。

ユーザーへの JourneyAdmin ロールを割り当てします

ユーザーに**JourneyAdmin**ロールを割り当てるには、次の手順を実行します。

1. Marketing Platform のホームページから**「設定」 > 「ユーザーの役割と権限」**を選択します。
   Result
   [ユーザーの役割と権限]ページが表示されます。
2. 左パネルで**Unica Journey**を展開します。
3. **「partition1」 > 「JourneyAdmin」**を選択します。
   Result
   JourneyAdminページが表示されます。
4. **[ユーザー]**セクションで、ユーザーを選択します。たとえば、 `asm_admin`です。

Result

asm_admin (asm_admin)ユーザーの詳細ページが表示されます。

5. **[ロールの編集]** を選択します。

   Result

   [ロールの編集]ページが表示されます。

6. [使用可能なロール] リストから、[**JourneyAdmin (Unica Journey)**] を選択し、[**>>**] ボタンをクリックしてロールを **[選択されたロール]** リストに移動します。
7. **「変更を保存」**をクリックします。

JourneyUserロールをユーザーに割り当てる

JourneyUserロールをユーザーに割り当てるには、次の手順を実行します。

1. Marketing Platform のホームページから**「設定」 > 「ユーザーの役割と権限」**を選択します。

   Result

   [ユーザーの役割と権限]ページが表示されます。

2. 左パネルで**Unica Journey**を展開します。
3. **「partition1」 > 「JourneyUser」**を選択します。

   Result

   JourneyUserページが表示されます。

4. **[ユーザー]**セクションで、ユーザーを選択します。たとえば、 `journey_example`です。

   Result

   journey_example (journey_example)ユーザーの詳細ページが表示されます。

5. **[ロールの編集]** を選択します。

   Result

   [ロールの編集]ページが表示されます。

6. [使用可能なロール] リストから**JourneyUser (Unica Journey)**を選択し、[**>>**] ボタンをクリックしてロールを **[選択されたロール]** リストに移動します。
7. **「変更を保存」**をクリックします。

ContactCentralAdmin ロールをユーザーに割り当てる

Unica Journey 管理者は、ContactCentralAdmin ロールを Journey ユーザーに割り当てて、これらのユーザーが Contact Central にアクセスできるようにする必要があります。Contact Central を Journey に対して有効にするには、Platform から Contact_Central_Configured の値を「Yes」に設定する必要があります。デフォルトでは、この値は「いいえ」に設定されています。ユーザーは、Platformで、パスAffinium|JourneyからContact_Central_Configuredの値「はい」/「いいえ」を選択できます。詳しくは、『*Unica Contact Central* 管理ガイド』を参照してください。

Chapter 8. Journey 対話ログ

Journey の対話ログは、スケジュールされたジョブとして実行されます。スケジューリング・パラメーターは、Journey エンジンの `application.properties` で設定します。以下に設定例を示します。

```
engine.logging.cron=0 15 3 * * ?
```

スケジュールされたジョブは、Journey エンジンの `application.properties` ファイルで再定義された代替スキーマにデータをエクスポートします。

```
journey.report.datasource.url =
journey.report.datasource.username =
journey.report.datasource.password =
journey.report.datasource.driver-class-name=
```

対話ログは、Journey アプリケーションに入ったすべてのコンタクトが、公開済みまたは完了の各 Journey で移動したときに、その動きをキャプチャーします。公開されているが一時停止されているジャーニーも、対話ログの対象と見なされます。

すべてのタッチポイント、電子メール、SMS、または CRM は、それぞれのチャネルを介して構成された統合を使用してオーディエンス データが送信されるため、Interaction Logging の対象と見なされます。すべての連絡先から受信した応答も取得されます。

Log4j2

Journey Web および Journey エンジンは、この標準をログ記録に使用します。Journey Web と Journey エンジンの両方の `log4j2.xml` ファイルは、インストール場所の `conf` フォルダ内に配置されます。

Journey Web と Journey エンジンの両方は、通常のアプリケーション・ログに加えパフォーマンス・ログを生成します。Journey Web の場合、デフォルトのログの場所は `logs` フォルダ内です。Journey エンジンの場合、ログはデフォルトで /Journeys/Engine/logs の下に JourneyEngine.log として作成されます。Journey Web と Journey エンジンの両方とも、前述のフォルダはインストール場所内に配置されます。

Chapter 9. Journey のログ記録

Journey バージョン 12.1.2 以前では、`journey_master_config.properties` ファイル内の **JOURNEY_LOGGING** プロパティーは、オーディエンスの追跡にのみ使用されていました。

バージョン 12.1.3 以降では、**JOURNEY_LOGGING** 機能は次のように拡張されています。

- デフォルトの Journey エンジン・ログ・レベルで Journey が開始されたかどうかを示します。
- ログを簡単にカスタマイズして、次のログを表示できます。`FATAL`、`ERROR`、`WARN`、`INFO`、`DEBUG`、`TRACEAUDIENCE`、または `TRACE`。
- Kafka からの読み取りと Kafka への書き込みをトレース・レベルでログに記録します。
- ジャーニーごとに複数のログ・ファイルを作成します。

JOURNEY_LOGGING プロパティーの構成

JOURNEY_LOGGING プロパティーを構成するためのルールは次のとおりです。

1. `journey_master_config.properties` ファイルにアクセスして、**JOURNEY_LOGGING** を見つけます。
2. プロパティーのコメントを外して編集し、値を指定します。
3. 値は CSV 文字列である必要があり、各文字列の形式は *J_ID:LEVEL* である必要があります。ここで、
   - *J_ID* はジャーニー ID です
   - *LEVEL* には、以下のいずれかの値を指定できます。`FATAL`、`ERROR`、`WARN`、`INFO`、`DEBUG`、`TRACEAUDIENCE`、または `TRACE`。*LEVEL* 値の順序は、次のように特殊性の高いものから低いものになっています。`FATAL`、`ERROR`、`WARN`、`INFO`、`DEBUG`、`TRACEAUDIENCE`、または `TRACE`
4. *J_ID:LEVEL* のデフォルト値は、空ストリングです。*J_ID:LEVEL* の値を変更するには、コメントを外して必要な値を指定します。
5. ロガー `com.hcl.journeyLogger` のデフォルト値を調整するには、`log4j2.xml` にアクセスして変更を行います。構成するログ・レベルは、`log4j2.xml` 内で固有にする必要があります。
6. *J_ID:LEVEL* の構成後、システムにより *J_ID.log* ファイルが作成されます。このログ・ファイルは、ジャーニー ID に固有のものです。複数のジャーニーのログを確認する場合は、複数の *J_ID:LEVEL* をカンマで区切って構成します。システムによって *J_ID.log* ファイルが作成されます。これらのファイルは、それぞれのジャーニー専用です。
7. `JOURNEY_LOGGING` で指定した 1 つを除くすべてのジャーニーに *DEFAULT:<LEVEL>* を適用する場合:
   a. `DEFAULT` に値を指定しない場合、`OFF` と見なされます。
   b. *J_ID:LEVEL* が「ジャーニーのログ」で構成されていない場合、`log4j2.xml` のログ・レベルが考慮されます。
8. Ignite キャッシュ (Delay/ES/DS/Wait4Capacity) のログをキャプチャーするには、次の形式で値を指定します。

   ```
   JOURNEY_LOGGING=181#361:DEBUG,<J_ID>#<NODE_ID>:DEBUG
   ```

   ここで

   <J_ID> はジャーニー ID で、*<NODE_ID>* はノード ID です。

Note:

- 変更は、60 秒以内に自動的に再ロードされます。
- log4j2 でパッケージ/クラス・レベルごとにログ・レベルを設定する場合は、`JOURNEY_LOGGING` が空であるか、コメント化されていることを確認します。ジャーニー・ログ・レベルとクラス固有のログ構成の組み合わせはサポートされていません。

JOURNEY_LOGGING=Archival:WARN - このパラメーターは、Journey エンジンに必要なログ・レベルを設定するために使用されます。

- spring.jpa.properties.hibernate.jdbc.batch_size=900 - このパラメーターには、JDBC 接続を使用した挿入および更新のためのバッチ内のレコードの最大数を指定します。この値を変更すると、エンジンのパフォーマンスに影響を与える可能性があります。
- spring.jpa.properties.hibernate.order_inserts=true - このパラメーターには、挿入ステートメントをバッチ処理するかどうかを指定します。
- spring.jpa.properties.hibernate.order_updates=true - このパラメーターには、挿入ステートメントをバッチ処理するかどうかを指定します。
- journey.kafka.producer.linger.ms=500 - Kafka プロデューサーがネットワーク・ディスパッチを行う前に 16kb (デフォルト) のデータ収集を待機する時間 (ミリ秒単位)
- audience.archive.cron - ジャーニー・オーディエンス・テーブルのアーカイブ cron ジョブについて言及するために使用されます。

Log4j

log4j2.xml クラスまたはパッケージ構成の場合、log4j2.xml ファイルにアクセスし、コメントを解除して次の例を使用します。

```
<Logger name="<<class name>> or <<package name>>" level="<<log level>>" additivity="false">
<appender-ref ref="Routing"/>
<appender-ref ref="console"/>
</Logger>
```

ここで

- *<<class name>>* は、完全修飾クラス名です。
- *<<package name>>* は、完全修飾パッケージ名です。
- *<<log level>>* は、JOURNEY_LOGGING プロパティーの構成 on page 42 に記載されているログ・レベルの 1 つです。

Interact

```
interact.startSession.enabled=false/true
```

これは、Journey エンジンの application.properties ファイルの設定で、Journey と Interact 間の通信のデバッグを有効にします。

値を **true** に設定すると、デバッグ・メッセージが有効になります

Chapter 10. ジャーニー・レコードの検証

レコードがジャーニーに入ると、ユーザーはレコードに検証を設定できます。

Validations_On_Journey_Records プロパティーが `Yes` に設定されると、電子メール形式、必須フィールドのデータ型、および必須フィールドのデータ定義の最小長と最大長についての検証が実行されます。

デフォルトで、**Validations_On_Journey_Records** プロパティーは `Yes` に設定されています。

このプロパティーは、Platform 側の Platform 構成から操作できます。Platform 構成設定の下で次のように移動できます: `Affinium|Journey|Journey_Configurations` の設定。

Validations_On_Journey_Records: このフラグには、ジャーニー・レコードで検証を実行するかどうかを定義します。

使用可能な値: `Yes` / `No`.

Note:

- **Validations_On_Journey_Records** プロパティーが `No` に設定されていて、ユーザーがデータ重複解消設定にいくつかのフィールドを追加した場合、空白および `NULL` データは重複排除フィールドでは受け入れられません。
- フラグを設定したら、変更を反映させるために、Platform と Journey からログアウトし、両方のアプリケーションに再度ログインします。
- どのような理由であれ、Platform から Journey 内の **Validations_On_Journey_Records** プロパティーにアクセスできない場合は、代替として Journey Webから `application.properties` ファイルの `journey.journeyrecords.validationrules` プロパティーを設定できます。

Chapter 11. Journey GDPR

Journey GDPR へのアクセス

GDPR ツールは、Journey アプリケーション・フォルダーからアクセスできます。場所は以下の通りです。

```
<Journey_Home>\Journey\tools\GDPR\
```

「GDPR のサポート対象」 > 「MariaDB」、Microsoft SQL Server、OneDb データベースおよび **Oracle**

Journey GDPRの実行

Journey GDPR を実行するには、次の手順を実行します。

1. `gdpr.properties` ファイル内の以下のプロパティーを変更します。

プロパティー名	値の例	注記
`Journey.audience.DBType`	`ORACLE`	現在、Journey は Oracle のみをサポートしています。
`Journey.audience.Db. Schema.Name`	`Journeyユーザー`	Journey データベースで使用されるスキーマ名。
`Journey.audience.Field`	`電子メール/携帯電話番号`	入力 CSV ファイルのフィールド名。
`Journey.audience.Csv`	`<GDPR_HOME>/sample/JourneyAudiences.csv`	<GDPR_HOME> をカレント・ディレクトリーのパスに置き換えます。 これは、Journey からのオプトアウトに必要なレコードを含む入力 `csv` ファイルです。
`Journey.audience.Output`	`<GDPR_HOME>/JourneyAudiences.sql`	`JourneyAudiences.sql` は、Journey アプリケーションからすべてのレコードを削除するために使用されるすべての SQL クエリーを含む出力ファイル名です。<GDPR_HOME> をカレント・ディレクトリーのパスに置き換えます。
`Journey.audience.Output. FileSizeLimit`	`10`	数値はMBs単位です。ファイル・サイズが入力した値を超えると、次のサフィックスを持つ複数のファイルが生成されます。`JourneyAudiences`

プロパティー名	値の例	注記
		_0、JourneyAudiences _1 など。

2. **Note:** エラーが表示された場合は、このログファイルを使用して追跡できます。

3. ファイルを実行するには、次のいずれかの手順を実行します。
    a. Windows の場合、gdpr_purge.bat ファイルを探して実行します。たとえば、gdpr_purge.bat ファイルが D:\workspace\HCL_GDPR\dist\journey\ にある場合、gdpr_purge.bat ファイルを実行します。
    b. UNIX ベースのシステムの場合、gdpr_purge.sh ファイルを探して実行します。たとえば、gdpr_purge.sh ファイルが \workspace\HCL_GDPR\dist\journey\ にある場合、./gdpr_purge.sh コマンドを実行します。
4. gdpr_purge.bat (Windows) または gdpr_purge.sh (Linux) を実行すると、上記手順で指定した <GDPR_HOME> に 「JourneyAudiences 0」、「JourneyAudiences _1」、「JourneyAudiences _2」 などの出力ファイルが作成されます。生成されるファイル数は、指定されたファイルサイズに依存します。
5. 「*JourneyAudiences_x*」ファイルには、JourneyAudiences.csv に記載されたレコードの削除クエリーが含まれます。
6. これらのクエリーは、「Journey」データベースで必要に応じて手動で実行し、「Journeyaudiences」テーブルからレコードを削除させる必要があります。

GDPR ユーティリティは、次のテーブルからレコードを削除します。JourneyAudiences、AudienceResponse、AudienceResponseMetaData、AudienceResponseInteraction、JourneyAudienceMile および JourneyAudienceGoal。ただし、集計結果が格納されている各テーブルからのデータ削除は行いません。たとえば、journeyFlow、journeyAudienceFlow、JourneyGoalContactTransaction などのテーブルです。したがって、UI でカウントの不一致が発生します。

GDPRツールでは、Publish Kafkaトピックからも、ファイルシステム上のファイルからも、顧客データを削除することはできません。このデータは、ユーザーが必要に応じて手動で削除する必要があります。

GDPRツールでは、JDBCコネクタからエクスポートされた顧客データを削除することはできません。

Chapter 12. SSL を使用した Kafka 認証

組織の Kafka インスタンスを使用している場合は、その Kafka インスタンス用に構成された証明書を使用できます。SSL キーと証明書を生成し、Journey アプリケーションのプロパティで構成するためのクライアント証明書を取得する必要はありません。

証明書がない場合は、自己署名認証局 (CA) を生成できます。これは、単なる公開鍵と秘密鍵のペアと証明書です。

各なKafka クライアントとブローカーの信頼ストアに, 同じ CA 証明書を追加する必要があります。

各なKafka ブローカーに SSL キーと証明書を生成します

Kafka サーバーの自己署名証明書を生成するには、次の手順を実行します。

前提条件

- 証明書と信頼ストアを生成するには、Java keytool と OpenSSL が必要です。
- 必要に応じて、OpenSSL の代わりに任意の SSL 証明書生成ユーティリティを使用できます。

1. SSL をデプロイするには、クラスター内の各マシンのキーと証明書を生成します。最初にキーを一時キーストアに生成して、後で CA でエクスポートして署名できるようにします。

   ```
   keytool -keystore kafka.server.keystore.jks -alias localhost -validity 365 -genkey
   ```

   - キーストア: 証明書を格納するキーストア・ファイル。キーストア ファイルには、証明書の秘密鍵が含まれているため、安全に保管する必要があります。
   - 有効性: 証明書の有効期間 (日数)。
2. 独自の CA (認証局) を作成する

   ```
   openssl req -new -x509 -keyout ca-key -out ca-cert -days 365
   ```

   生成された CA は、単に公開鍵と秘密鍵のペアと証明書であり、他の証明書に署名することを目的としています。
3. 生成された CA をクライアントの信頼ストアに追加して、クライアントがこの CA を信頼できるようにします。
   - `keytool -keystore kafka.server.truststore.jks -alias CARoot -import -file ca-cert`
   - `keytool -keystore kafka.client.truststore.jks -alias CARoot -import -file ca-cert`
4. 生成された CA を使用して、キーストア内のすべての証明書に署名します。
   a. キーストアから証明書をエクスポートします。

      ```
      keytool -keystore kafka.server.keystore.jks -alias localhost -certreq -file cert-file
      ```
5. CA で署名します。

   ```
   openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days 365 -CAcreateserial -passin
   pass:<password>
   ```
6. CA の証明書と署名付き証明書の両方をキーストアにインポートします。

   ```
   keytool -keystore kafka.server.keystore.jks -alias CARoot -import -file ca-cert
   ```

```
keytool -keystore kafka.server.keystore.jks -alias localhost -import -file cert-signed
```

7. クライアント キーストアを作成し、CA の証明書と署名付き証明書の両方をクライアント キーストアにインポートします。これらのクライアント証明書は、アプリケーション プロパティで使用されます。

```
keytool -keystore kafka.client.keystore.jks -alias localhost -validity 365 -genkey

keytool -keystore kafka.client.keystore.jks -alias localhost -certreq -file cert-file

openssl x509 -req -CA ca-cert -CAkey ca-key -in cert-file -out cert-signed -days 365 -CAcreateserial -passin
pass:<password>

keytool -keystore kafka.client.keystore.jks -alias CARoot -import -file ca-cert

keytool -keystore kafka.client.keystore.jks -alias localhost -import -file cert-signed
```

Kafka サーバー、Journey、および Link コンポーネントに SSL を構成する

Kafka サーバーとクライアント証明書に使用するサーバー証明書は、Journey Web、Journey エンジン、Unica Link (Kafka リンク) など、Kafka サーバーに接続するすべてのアプリケーション、またはこの Kafka サーバーに接続するために必要なその他のツールで使用する必要があります。

Kafka サーバー、Journey コンポーネント、および Link コンポーネントに SSL 認証を構成するには、以下のセクションで説明する手順を実行します。

Kafka サーバーをSSL 認証で 構成する

次のサーバー証明書は、Kafka サーバーのみに使用する必要があります。これらの証明書を必要なマシンで共有し、パスワードをメモします。

- `kafka.server.keystore.jks`
- `Kafka.server.truststore.jks`

Kafka サーバーの config ディレクトリーにある次の server.properties を更新します。

```
listeners=SSL://<KAFKA_HOST>:<KAFKA_PORT>
ssl.keystore.location=/PATH/kafka.server.keystore.jks
ssl.keystore.password= password
ssl.key.password= password
ssl.truststore.location= /PATH/kafka.server.truststore.jks
ssl.truststore.password= password
ssl.endpoint.identification.algorithm=
ssl.client.auth=required
security.inter.broker.protocol=SSL
```

Journey エンジンに Kafka SSL を構成する

次のクライアント証明書を使用し、必要なマシンでこれらの証明書を共有し、パスワードをメモします。

- `Kafka.client.keystore.jks`
- `kafka.client.truststore.jks`

1. <JOURNEY_HOME>/Engine/ ディレクトリーから journey_engine_master.config を更新します。
2. 次のプロパティ値を更新します。

```
kafka.security.enabled=Y
kafka.security.protocols.enabled=SSL
security.protocol=SSL
ssl.truststore.location= /PATH/kafka.client.truststore.jks
ssl.truststore.password=<ENCYPTED PASSWORD WITH JOURNEY ENCRYPTION
TOOL>
ssl.keystore.location= /PATH/kafka.client.keystore.jks
ssl.keystore.password=<ENCYPTED PASSWORD WITH JOURNEY ENCRYPTION TOOL>
ssl.key.password=<ENCYPTED PASSWORD WITH JOURNEY ENCRYPTION TOOL>
ssl.endpoint.identification.algorithm=
```

Journey Web に Kafka SSL を構成する

1. <JOURNEY_HOME>/Web/properties/ ディレクトリーから Journey Web application.properties ファイルを更新します。
2. 次のプロパティ値を更新します。

```
kafka.security.enabled=Y
kafka.security.protocols.enabled=SSL
ssl.truststore.location= /PATH/kafka.client.truststore.jks
ssl.truststore.password= <ENCYPTED PASSWORD WITH JOURNEY ENCRYPTION
TOOL>
ssl.keystore.location= /PATH/kafka.client.keystore.jks
ssl.keystore.password= <ENCYPTED PASSWORD WITH JOURNEY ENCRYPTION
TOOL>
ssl.key.password= <ENCYPTED PASSWORD WITH JOURNEY ENCRYPTION TOOL>
ssl.endpoint.identification.algorithm=
```

Unica Link コンポーネントに SSL を構成する

Unica Link インストール済み環境の kafkalink.properties ファイルで次のプロパティー値を更新します。

```
security.ssl=true
security.protocol=SSL
ssl.truststore.location= /PATH/kafka.client.truststore.jks
ssl.truststore.password=password
security.authentication=username
ssl.keystore.location= /PATH/kafka.client.keystore.jks
ssl.keystore.password=password
ssl.key.password=passwordssl.endpoint.identification.algorithm=
```

Kafka サーバー、Journey および Link コンポーネントに SSL を構成する

Kafka サーバー、Journey コンポーネント、および Link コンポーネントに SASL 認証を構成するには、以下のセクションで説明する手順を実行します。

Kafka サーバーをSASL 認証で 構成する

1. kafka-run-class.bat/sh.で JVM パラメーターを指定します。

   set `JAVA_OPTS=%JAVA_OPTS%`

   `-Djava.security.auth.login.config=/PATH/kafka_server_jaas.conf`

   `set COMMAND=%JAVA% %JAVA_OPTS% %KAFKA_HEAP_OPTS%`

   `%KAFKA_JVM_PERFORMANCE_OPTS% %KAFKA_JMX_OPTS% %KAFKA_LOG4J_OPTS% -cp`

   `"%CLASSPATH%" %KAFKA_OPTS% %*`

   サンプル jaas.config ファイル:

   ```
   KafkaServer {
      org.apache.kafka.common.security.plain.PlainLoginModule required
      username="admin"
      password="admin-secret"
      user_admin="admin-secret"
      user_alice="alice-secret";
   };
   ```

   ```
   KafkaClient {
     org.apache.kafka.common.security.plain.PlainLoginModule required
     username="alice"
     password="alice-secret";
   };
   ```

2. `KAFKA_SERVER/config/server.properties` から次の Kafka サーバー プロパティ ファイルを更新します。

   ```
   listeners=SASL_PLAINTEXT:// <KAFKA_HOST>:<KAFKA_PORT>
   security.inter.broker.protocol=SASL_PLAINTEXT
   sasl.mechanism.inter.broker.protocol=PLAIN
   sasl.enabled.mechanisms=PLAIN
   ```

Journey エンジンに Kafka SASL を構成する

1. <JOURNEY_HOME>/Engine/conf/ ディレクトリーから Journey エンジン log4j2.xml ファイルを更新します。log4j2.xml の次の行のコメントを外します。

   ```
    <!-- Kafka SASL configuration -->
   <Property name="security.protocol">${sys:security.protocol}</Property>
   <Property name="sasl.mechanism">${sys:sasl.mechanism}</Property>
   ```

2. <JOURNEY_HOME>/Engine/ ディレクトリーから journey_engine_master.config を更新します。次のプロパティ値を更新します。

   ```
   kafka.security.enabled=Y
   kafka.security.protocols.enabled=SASL_PLAINTEXT
   security.protocol=SASL_PLAINTEXT
   sasl.mechanism=PLAIN
   java.security.auth.login.config=./kafka_client_jaas.conf
   ```

Journey Web に Kafka SASL を構成する

<JOURNEY_HOME>/Web/properties/ ディレクトリーから Journey Web application.properties ファイルを更新します。

```
kafka.security.enabled=Y
kafka.security.protocols.enabled=SASL_PLAINTEXT
 java.security.auth.login.config=/PATH/kafka_client_jaas.conf
```

Unica Link コンポーネントに Kafka SASL を構成する

Unica Link インストール済み環境の kafkalink.properties ファイルで次のプロパティー値を更新します。

```
security.sasl =true
security.protocol=SASL_PLAINTEXT
security.sasl.auth.login.config =/PATH/kafka_client_jaas.conf
sasl.mechanism=PLAIN
```

Kafka サーバーおよび Journey コンポーネントに SASL_SSL を構成する

Kafka サーバーやその他の Journey コンポーネントに SASL 認証を構成するには、次のセクションで説明する手順を完了してください。

Note: Unica Link は、SASL_SSL 認証メカニズムを使用した Kafka-link への接続をサポートしていません。SASL または SSL 認証メカニズムを使用する必要があります。

Kafka SASL_SSL でKafka サーバーを設定する

Kafka サーバー構成ディレクトリーで以下の server.properties を更新します。

```
listeners=SASL_SSL:// <KAFKA_HOST>:<KAFKA_PORT>
security.inter.broker.protocol=SASL_PLAINTEXT
sasl.mechanism.inter.broker.protocol=PLAIN
sasl.enabled.mechanisms=PLAIN
ssl.keystore.location=/PATH/kafka.server.keystore.jks
ssl.keystore.password=password
ssl.key.password= password
ssl.truststore.location=/PATH/kafka.server.truststore.jks
ssl.truststore.password= password
ssl.endpoint.identification.algorithm=
ssl.client.auth=required
security.inter.broker.protocol=SSL
```

Journey エンジンに Kafka SASLを構成する

1. <JOURNEY_HOME>/Engine/conf/ ディレクトリーから Journey エンジン log4j2.xml ファイルを更新します。

   log4j2.xml の次の行のコメントを外します。

```
<Property name="sasl.mechanism">${sys:sasl.mechanism}</Property>
<Property name="security.protocol" >${sys:security.protocol}</Property>
<Property name="ssl.truststore.location" >${sys:ssl.truststore.location}</Property>
<Property name="ssl.truststore.password">${sys:ssl.truststore.password}</Property>
<Property name="ssl.keystore.location">${sys:ssl.keystore.location}</Property>
<Property name="ssl.keystore.password">${sys:ssl.keystore.password}</Property>
<Property name="ssl.key.password">${sys:ssl.key.password}</Property>
<Property
 name="ssl.endpoint.identification.algorithm">${sys:ssl.endpoint.identification.algorithm}</Property>
```

2. <JOURNEY_HOME>/Engine/ ディレクトリーから次の journey_engine_master.config を更新します。

   次のプロパティ値を更新します。

```
kafka.security.enabled=Y
kafka.security.protocols.enabled=SASL_SSL
ssl.truststore.location=/PATH/kafka.client.truststore.jks
ssl.truststore.password=<ENCYPTED PASSWORD WITH JOURNEY ENCRYPTION TOOL>
ssl.keystore.location=/PATH/kafka.client.keystore.jks
ssl.keystore.password=<ENCYPTED PASSWORD WITH JOURNEY ENCRYPTION TOOL>
ssl.key.password=<ENCYPTED PASSWORD WITH JOURNEY ENCRYPTION TOOL>
ssl.endpoint.identification.algorithm=
java.security.auth.login.config=/PATH/kafka_client_jaas.conf
```

Journey Web に Kafka SASL_SSL を構成する

<JOURNEY_HOME>/Web/properties/ ディレクトリーから、次の Journey Web application.properties ファイルを更新します。

```
kafka.security.enabled=Y
kafka.security.protocols.enabled=SASL_SSL
ssl.truststore.location=/PATH/kafka.client.truststore.jks
ssl.truststore.password=<ENCYPTED PASSWORD WITH JOURNEY ENCRYPTION TOOL>
ssl.keystore.location=/PATH/kafka.client.keystore.jks
ssl.keystore.password=<ENCYPTED PASSWORD WITH JOURNEY ENCRYPTION TOOL>
ssl.key.password=<ENCYPTED PASSWORD WITH JOURNEY ENCRYPTION TOOL>
ssl.endpoint.identification.algorithm=
java.security.auth.login.config=/PATH/kafka_client_jaas.conf
```

Affinium|Journey|Kafka_Configurations

KafkaBrokerURL

説明

Zookeer または Kafka を実行している IP とポートを定義するには、このプロパティーを使用します。

デフォルト値

デフォルト値が定義されていません。

有効な値

有効な Kafka ブローカー URL

CommunicationMechanism

説明

Kafka クライアント認証の構成を指定します。

デフォルト値

デフォルト値が定義されていません。

有効な値

Kafka 構成ページでは、組織の Kafka サーバーのストリーム・セキュリティーに応じて、CommunicationMechanism フィールドに次のいずれかの値を選択できます。

- NO_SASLPLAINTEXT_SSL
- SASL_PLAINTEXT
- SSL
- SASL_PLAINTEXT_SSL

sasl.mechanism

説明

Kafka クライアント認証を指定します。

デフォルト値

デフォルト値が定義されていません。

有効な値

Kafka サーバーの認証構成に応じて、次のいずれかの値を選択できます。

- SASL_PLAINTEXT
- SASL_PLAINTEXT_SSL
- SSL

UserForKafkaDataSource

説明

Kafka サービス・アクセス資格情報を含むデータ・ソースを参照する HCL Unica ユーザーを指定します。この値は、システム・ユーザーを作成するときに構成します。

デフォルト値

デフォルト値が定義されていません。

有効な値

Kafka データ・ソースを参照する有効なユーザー

sasl.jaas.config.dataSource

説明

Kafka は SASL の構成に Java 認証および承認サービス (JAAS) を使用します。すべての SASL 認証メカニズムに JAAS 構成を提供する必要があります。

デフォルト値

デフォルト値が定義されていません。

有効な値

Kafka_home/server.properties の「listener.name.sasl_ssl.plain.sasl.jaas.config」を参照してください

truststore.location

説明

「kafka.server.truststore.jks」のパスを指定します

デフォルト値

デフォルト値が定義されていません。

有効な値

Kafka_home/server.properties/ssl.truststore.location に記載されている「kafka.server.truststore.jks」ファイルのパス。

truststore.password.dataSource

説明

Kafka トラストストアのログイン認証情報を含む Platform データ・ソースを指定します。この値は、システム・ユーザーを作成するときに構成します。

デフォルト値

デフォルト値が定義されていません。

有効な値

`kafka_home/server.properties/ssl.keystore.location` に記載されている「kafka.server.keystore.jks」のパス

keystore.password.dataSource

説明

Kafka キーストアのログイン認証情報を含む Platform データ・ソースを指定します。この値は、システム・ユーザーを作成するときに構成します。

デフォルト値

デフォルト値が定義されていません。

有効な値

Kafka キーストアのログイン資格情報を含むデータ・ソース

key.password.dataSource

説明

Kafka キーのログイン認証情報を含む Platform データ・ソースを指定します。この値は、システム・ユーザーを作成するときに構成します。

デフォルト値

デフォルト値が定義されていません。

有効な値

Kafka キーストアのログイン資格情報を含むデータ・ソース

ssl.endpoint.identification.algorithm

説明

クライアントがサーバーのホスト名を検証するために使用するエンドポイント識別アルゴリズムを指定します。ssl.endpoint.identification.algorithm を空の文字列に設定すると、サーバー・ホスト名の検証が無効になります。

デフォルト値

空

有効な値

Kafka_home/server.properties に記載されている ssl.endpoint.identification.algorithm を参照してください。

Note: オンプレミスの Confluent Kafka のサポート - Journey は、セキュリティ・メカニズムを使用せずにオンプレミスの Confluent Kafka と連携します。Google で管理する Confluent Kafka - Journey は、Google が管理する Confluent Kafka と連携します。その際、セキュリティ・メカニズムには PLAIN メカニズムによる SASL_SSL を使用し、SSL 証明書は使用しません。

Chapter 13. Kerberos を使用した Kafka 認証

クライアント (プロデューサー、コンシューマー、コネクト ワーカーなど) は、独自のプリンシパル (通常はクライアントを実行しているユーザーと同じ名前) を使用してクラスターに対して認証するため、必要に応じてこれらのプリンシパルを取得または作成します。次に、各クライアントの JAAS 構成プロパティーを設定します。JVM 内の異なるクライアントは、異なるプリンシパルを指定することで、異なるユーザーとして実行できます。

Kafka Kerberos 構成を Journey で開始する前に、次の操作を実行します。

Kerberos をホストするマシンの `/var/kerberos/krb5kdc` に存在するすべてのファイルを、次の Journey のインストール場所にコピーします。 `<Journey-HOME>/Journey/Web/properties`

Kerberos をホストするマシンの /etc にある **krb5.conf** ファイルを、次の Journey のインストール場所にコピーします。 `<Journey-HOME>/Journey/Web/properties`

Note: `kafka_Hosted` マシンのホスト名を、Unica アプリケーションをホストするマシンに追加し、その逆も同様に実行します。Kafka 通信が 2 台のマシン間で正常に動作することを確認してください。

1. **Journey エンジンおよび Journey Web 用の Kafka セキュリティーを有効にする**
   - **Journey エンジンの場合** - krb5 ファイルの場所を `<JOURNEY_HOME>/Engine/journey_master_config.properties` に置き換えます。

     kafka.security.enabled=Y

     kafka.security.protocols.enabled=GSSAPI
   - **Journey Web の場合** - krb5 ファイルの場所を `<JOURNEY_HOME>/WEB/properties/application.properties` に置き換えます。

     kafka.security.enabled=Y

     kafka.security.protocols.enabled=GSSAPI
2. JAAS 構成で構成されている keytab が、Kafka クライアントを起動しているオペレーティング・システムのユーザーによって読み取り可能であることを確認します
   - **Journey エンジンの場合** - kafka_server_jaas.conf ファイルの場所を `<JOURNEY_HOME>/Engine/journey_master_config.properties` に置き換えます。

     `java.security.auth.login.config = <jass LOCATION> example - /etc/kafka_server_jaas.conf`
   - **Journey Web の場合** - kafka_server_jaas.conf ファイルの場所を `<JOURNEY_HOME>/WEB/properties/application.properties` に置き換えます。

     `java.security.auth.login.config = <jass LOCATION> example - /etc/kafka_server_jaas.conf`
3. krb5 ファイルの場所を JVM パラメーターとして各クライアントの JVM に渡します。

- **Journey エンジンの場合** - krb5 ファイルの場所を `<JOURNEY_HOME>/Engine/journey_master_config.properties` に置き換えます。

  ```
  java.security.krb5.conf = <krb5 LOCATION> example - /etc/krb5.conf
  ```

- **Journey Web の場合** - krb5 ファイルの場所を `<JOURNEY_HOME>/WEB/properties/application.properties` に置き換えます。

  ```
  java.security.krb5.conf = <krb5 LOCATION> example - /etc/krb5.conf
  ```

4. Kafka で Kerberos を構成する場合、次のファイルで以下の変更を追加または更新します。

   Location - `/<Journey-HOME>/Journey/KafkaStandalone/config/server.properties`

   ```
   sasl.enabled.mechanisms=GSSAPI sasl.mechanism.inter.broker.protocol=GSSAPI
    security.inter.broker.protocol=SASL_PLAINTEXT listeners=SASL_PLAINTEXT://0.0.0.0:9092
    advertised.listeners=SASL_PLAINTEXT://<Kafka_Kerberos_principle_host_name>:9092 #E.g.
    advertised.listeners=SASL_PLAINTEXT://
   ip-10-10-10-100.ap-south-1.compute.internal.nonprod.hclpnp.com:9092
    listener.name.sasl_plaintext.gssapi.sasl.jaas.config=com.sun.security.auth.module.Krb5LoginModule
    required useKeyTab=true storeKey=true keyTab="/var/kerberos/krb5kdc/kfkserver.keytab"
    principal="kafka/<Kafka_Kerberos_principle_host_name>"; sasl.kerberos.service.name=kafka #e.g.
    keyTab="/var/kerberos/krb5kdc/kfkserver.keytab"
    principal="kafka/ip-10-10-10-100.ap-south-1.compute.internal.nonprod.hclpnp.com";
    sasl.kerberos.service.name=kafka
   ```

   Note: この値は、Kafka のプリンシパルに応じて変化します

   ```
   ip-10-10-10-100.ap-south-1.compute.internal.nonprod.hclpnp.com@HCLPNP.COM
   ```

5. Journey エンジンおよび Journey Web の application.properties ファイルに Kafka Kerberos ホスト名を追加します。

   ```
   spring.kafka.bootstrap-servers=<Kafka_Kerbersos_Host_Name>:9092
   ```

6. 次の名前でフォルダを作成します -> **krb5.conf.d** および **log**。作成場所: `/<Journey-HOME>/Journey/Web/properties`
7. `/<Journey-HOME>/Journey/Web/properties` にある `kafka_server_kerberos_jaas.conf` または `equivalent Kafka Kerberos client conf` を、適用される kerberos の詳細と Journey のインストール場所に応じた最新のパスで更新します。

   ```
   keyTab="/var/kerberos/krb5kdc/zkserver.keytab"
   principal="zookeeper/ip-10-10-10-10.ec2.internal.nonprod.hclpnp.com";
   keyTab="/var/kerberos/krb5kdc/kfkserver.keytab"
   principal="kafka/ip-10-10-10-10.ec2.internal.nonprod.hclpnp.com"
   ```

8. `/<Journey-HOME>/Journey/Web/properties` にある **krb5.conf** を、適用される kerberos の詳細と Journey のインストール場所に応じた最新のパスで更新します。

   ```
   includedir /<Journey-HOME>/Journey/Web/properties/krb5.conf.d/
    [logging]
    default = FILE:/<Journey-HOME>/Journey/Web/properties/log/krb5libs.log
    kdc = FILE:/<Journey-HOME>/Journey/Web/properties/log/krb5kdc.log
    admin_server = FILE:/<Journey-HOME>/Journey/Web/properties/log/kadmind.log
   ```

Kerberos Kafka 認証の JBoss 設定

次の変更は、JBoss standalone.xml ファイルで、Journey Web を開始する前に行われています。

1. に移動してください。 `<JBOSS_HOME>\standalone\configuration\`
2. standalone.xml を開きます。
    a. **servlet-container** ノードに以下の値がすべて含まれていることを確認します。そうでない場合は、ノードを次の一連の行に置き換えてください

    ```
    <servlet-container name="default" disable-caching-for-secured-pages="false">
    <jsp-config/>
    <websockets/>
    </servlet-container>
    ```

    b. **kafka_server_kerberos_jaas.conf および krb5.conf のファイル・パスを、次に示すように standalone.xml で構成します。**

    ```
    <system-properties>
     <property name="java.security.auth.login.config"
     value="<Journey-HOME>/Journey/Web/properties/kafka_server_kerberos_jaas.conf"/>
     <property name="java.security.krb5.conf"
     value="<Journey-HOME>/Journey/Web/properties/krb5.conf"/>
     <property name="java.security.krb5.debug" value="true"/>
     <property name="java.security.disable.secdomain.option" value="true"/>
     </system-properties>
    ```

    c. **次に示すように、standalone.xml で KafkaClient および KafkaServer conf を追加します。**

```
<security-domain name="KafkaClient" cache-type="default">
 <authentication>
 <login-module code="com.sun.security.auth.module.Krb5LoginModule" flag="required">
 <module-option name="storeKey" value="true"/>
 <module-option name="useKeyTab" value="true"/>
 <module-option name="refreshKrb5Config" value="true"/>
 <module-option name="principal" value="<CLIENT-PRINCIPLE>"/> example -
 "kafka/ip-10-10-10-10.ec2.nonprod.hclpnp.com"
 <module-option name="keyTab" value="<Journey-HOME>/Journey/Web/properties/kfkserver.keytab"/>
 <module-option name="doNotPrompt" value="true"/>
 </login-module>
 </authentication>
</security-domain>
<security-domain name="KafkaServer" cache-type="default">
 <authentication>
 <login-module code="com.sun.security.auth.module.Krb5LoginModule" flag="required">
 <module-option name="storeKey" value="true"/>
 <module-option name="useKeyTab" value="true"/>
 <module-option name="refreshKrb5Config" value="true"/>
 <module-option name="principal" value="<Server-PRINCIPLE>"/> example -
 "zookeeper/ip-10-10-10-10.ec2.nonprod.hclpnp.com"
 <module-option name="keyTab" value="<Journey-HOME>/Journey/Web/properties/zkserver.keytab"/>
 <module-option name="doNotPrompt" value="true"/>
 </login-module>
 </authentication>
</security-domain>
```

Note: 前述のパラメーター keyTab とプリンシパルの値は、Kerberos サーバーのセットアップ時に生成されたものと同じである必要があります。

3. `Jboss <JBOSS_HOME>\standalone\configurationstandalone.xml` に前記の変更を保存した後、Journey Web を開始できます。Journey に Kafka Kerberos を構成した後、Journey で REST エントリー・ソースを作成している場合に、アプリケーションで次のようなエラーが発生したときは、次のように **servlet-container** ノードのノード構成を次のように確認する必要があります。

   Errors: REST API を取得できないか、Kafka を取得できません

   - **servlet-container** ノードに以下の値がすべて含まれていることを確認します。含まれていない場合は、ノードを以下の位置の次の一連の行で置き換えてください。

     ```
     <JBOSS_HOME>\standalone\configuration\standalone.xml
     <servlet-container name="default" disable-caching-for-secured-pages="false">
     <jsp-config/>
     <websockets/>
     </servlet-container>
     ```

Kerberos Kafka 認証の WAS 設定

WebSphere 環境では、2 つのファイルを設定する必要があります。

1. システム環境で krb5 を設定します。

   krb5 サーバーの場合 > 「サーバー・タイプ」 > 「サーバーを選択」 > 「Java およびプロセス管理」 > 「プロセス定義」 > 「Java 仮想マシン」 > 「汎用 JVM 引数」 に移動して、下記の引数を引数リストに追加します。

   -Djava.security.krb5.conf=<KRB File Location>\krb5.conf

2. JAAS エントリー

   <WebSphere Home>\AppServer\profiles\<Profile>\properties\wsjaas.conf に移動します。

   次のエントリーをファイルに追加します。

   KafkaServer

   ```
   { com.ibm.security.auth.module.Krb5LoginModule required useKeytab="<Zookeper Key Tab Path>
   \zkserver.keytab" storeKey=true principal= <Zookeper Principle> eg- "zookeeper/ip-10-10-10-10.ap-
   south-1.compute.internal.nonprod.hclpnp.com" credsType = both; }
   ```

   KafkaClient

   ```
   { com.ibm.security.auth.module.Krb5LoginModule required useKeytab="<Kafka Server Kay
   Tab Path>\kfkserver.keytab" principal= <Kafka Principle> eg - "kafka/ip-10-10-10-10.ap-
   south-1.compute.internal.nonprod.hclpnp.com" credsType = both debug=true; }
   ```

   アプリケーション・サーバーを再起動します。

Deliver 応答のための Kafka Kerberos の設定:

Unica アプリケーションにログインし、次の場所に移動します。

1. **「場所」 > 「設定」 > 「構成」 > 「HCL Unica」 > 「Journey」 > 「Kafka」 > 「構成」**

   次のように設定します。

   ```
   KafkaBrokerURL:     <Priciple_Hostname>:9092
   CommunicationMechanism:     GSSAPI
   sasl.mechanism: GSSAPI
   sasl.jass.config.location:
    <JOURNEY_HOME>/WEB/properties/application.properties/kafka_server_Kerberose_jaas.conf
   java.security.krb5.conf.location: <JOURNEY_HOME>/WEB/properties/application.properties/krb5.conf
   ```

   .
2. 場所: **「設定」 > 「構成」 > 「HCL Unica」 > 「Deliver」 > 「Kafka」 > 「RCT」**

   次のように設定します。

   ```
   Set
   KafkaBrokerURL:     <Priciple_Hostname>:9092
   CommunicationMechanism:     SASL_PLAINTEXT
   sasl.mechanism: GSSAPI
   ```

外部リソースを Journey で構成するための Kafka の構成 (AssetPicker および Journey の構成)

Unica アプリケーションにログインし、次の場所に移動します。

「場所」 > 「設定」 > 「構成」 > 「HCL Unica」 > 「Journey」 > 「統合」 > 「データ・ソース」 > <システム構成テンプレート名> > 「Kafka 構成」

```
Bootstrap servers (comma separated list of hosts)          <Karberos_Kafka_Principle_Host:>7001
(e.g. ip-10-10-10-100.nonprod.hclpnp.com:7001){}
Security protocol             SASL_PLAINTEXT
SASL mechanism             KERBEROS
Kerberos - Configuration file path (e.g. /etc/krb5.conf, C:/Windows/krb5.ini)
 <JOURNEY_HOME>/WEB/properties/application.properties/krb5.conf
Kerberos - Keytab file path               <JOURNEY_HOME>/WEB/properties/kfkserver.keytab
Kerberos - Principal        kafka/<Karberos_Kafka_Principle_Host>@HCLPNP.COM
e.g. kafka/ip-10-10-10-100.nonprod.hclpnp.com@HCLPNP.COM
Kerberos - Service Name              kafka
```

Chapter 14. SSL 用に Web アプリケーション サーバー Tomcat を構成する

Unica アプリケーションが配置されているすべてのアプリケーション・サーバーで、採用を決定した証明書を使用するように Web アプリケーション・サーバーを構成します。

これらの手順の実行について詳しくは、アプリケーション・サーバーの資料を参照してください。

Cookie のセキュリティーの確認

いくつかの cookie は、クライアント・ブラウザーで適切に保護されない場合があります。cookies を保護しないと、アプリケーションは中間者攻撃およびセッション・ハイジャック攻撃に対してぜい弱なままになります。この問題を修正するには、以下の予防手段を取ります。

- 常に SSL の使用を強制して、ワイヤー上で代行受信されている cookie のリスクを減らします。
- Web アプリケーション・サーバーで、`secure` フラグおよび `httponly` フラグをすべての cookie に設定します。
  - `secure` フラグは、ブラウザーが HTTPS 接続のみを使用して cookie を送信するよう指示します。このフラグを設定する場合、相互に通信するすべてのアプリケーションで SSL を有効にする必要があります。
  - `httponly` フラグは、cookie がクライアント・サイドのスクリプトを介してアクセスされないようにします。

Tomcat での SSL のフラグの設定

Tomcat で `secure` フラグと `httponly` フラグを設定するには 、Tomcat の `.xml` サーバーで以下の変更を行います。

このタスクについて

```
<Connector port="7003" protocol="org.apache.coyote.http11.Http11NioProtocol"
maxThreads="150" SSLEnabled="true" scheme="https" acceptCount="100" clientAuth="false"
disableUploadTimeout="true" enableLookups="false" secure="true" sslProtocol="TLS"
 keystoreFile="/opt/v12.1/v12.1.0.1.1/Campaign/SSL_NEW/PlatformClientIdentity.jks" keystorePass="password" >
  </Connector>
```

Unica Journey に SSL を構成する

SSL を使用するように Unica Journey を構成するには、いくつかの構成プロパティーを設定する必要があります。このセクションで、Unica Journey のインストール済み環境、および SSL を使用して保護する通信に適した手順を使用してください。

About this task

Unica インストール済み環境にセキュア接続でアクセスするとき、また、以下の手順で説明するようにアプリケーションのナビゲーション・プロパティを設定する場合は、URL で`https`とセキュア・ポート番号を使用する必要があります。Tomcat のデフォルトの SSL ポートは `8443` です。

この手順に従って Journey with SSL を設定します

1. Unica にログインし、**「設定」>「構成」**をクリックします。
2. `Affinium | Journey | navigation` プロパティーの値を Unica Journey URL に設定します。

例: `https://host.domain:SSL_port/unica`

各部の意味は以下のとおりです。

- *host* は Unica Journey がインストールされているマシンの名前または IP アドレスです。
- *domain* は Unica 製品がインストールされているお客様の企業ドメインです。
- *SSL_Port* は Unica Journey が配置されているアプリケーション・サーバーの SSL ポートです。

URL が `https` で始まることに注意してください。

Chapter 15. Mailchimp 構成

Mailchimp を外部ソースとして Journey で構成します。

About this task

「Unica Platform」 > 「設定」 > 「構成」に移動します

「構成カテゴリー」ページから、**「Journey」 > 「統合」 > 「データ・ソース」**に移動します

Journey データ・ソースを追加する場合

1. **「systemconfigurationTemplates」**をクリックします
   Result
   「systemConfigurationTemplates」ページが表示されます。
2. 以下の情報を提供します。
   - 新規カテゴリー名: <Journey>
   - **systemIdentifier**: ジャーニー
   - **userCredentials**: デフォルトのユーザー
   - **defaultUserCredentials**: asm_Admin
   - **dataSourceNameForCredentials**: <JOURNEY_DS_1>
   - **AdditionalParameters**:
   - **event-publisher-service.kafka.topics**: CIFINTEGRATION
   - **event-publisher-service.kafka.topics.CIFINTEGRATION.value.format**: Json
3. **「保存」**をクリックします。
4. 「Journey」ノードを展開し、**「httpGateway」**をクリックします
   Result
   「'httpGateway' の設定」ページが表示されます。
5. 以下の情報を提供します。

   baseUrl : http:<hostname>:<port>/journey
6. **「Kafka 構成」**リンクをクリックして、以下の情報を追加します

   ブートストラップ・サーバー (ホストのカンマ区切りリスト): <Kafkahost>:<port>

   例: <IP またはホスト名>:9092

Mailchimp データ・ソースを追加する場合

1. **「systemconfigurationTemplates」**をクリックします
   Result
   「systemConfigurationTemplates」ページが表示されます。
2. 以下の情報を提供します。
   - 新規カテゴリー名: <Mailchimp>
   - **systemIdentifier**: Mailchimp

- **userCredentials**: デフォルトのユーザー
- **defaultUserCredentials**: asm_Admin
- **dataSourceNameForCredentials**: <Mailchimp_DS>

3. **「保存」**をクリックします。
4. 「Mailchimp」ノードを展開し、**「httpGateway」**をクリックします

   Result

   「'httpGateway' の設定」ページが表示されます。
5. 以下の情報を提供します

   baseUrl : https://<MailchimpHostname>/<Version>/

   ユーザーはこれらのデータ・ソースを Journey に追加する必要があります。ユーザーは以下の情報を収集する必要があります。

   a. Journey データ・ソースのユーザー ID とパスワード。**「Journey アプリケーション」 > 「設定」 > 「REST」**に移動し、新しい REST 統合を作成するか、既存の REST 統合を使用します。clientid と Client Secret をコピーします。
   b. Mailchimp データ・ソースのユーザー ID とパスワード。Mailchimp アプリケーションにログインし、**「プロファイル」 > 「追加」 > 「API キー」**に移動します。API キーを取得し、ユーザー列名は user になります

   mailchimp a/c で、**「Webhook の追加」** URL は次の通りです。

https://<AssetpickerHostname>:<Port>/asset-viewer/api/AssetPicker/webhook/Mailchimp/events/webhook_listener

「プラットフォーム設定」 > 「ユーザー」に移動します

「必須ユーザー」 (asm_Admin など) をクリックします

「dataSources リンクの編集」をクリックし、以下のデータ・ソースを追加します

Mailchimp_DS - ユーザー: <user> およびパスワード <API キー>

JOURNEY_DS_1 - ユーザー: <clientid> およびパスワード <Client Secret>

「Unica Platform」 > 「設定」 > 「構成」 > 「Unica Platform」 > 「セキュリティー」 > 「API 管理」 > 「Unica Content Integration」に移動します

「API 構成テンプレート」をクリックし、以下の情報を追加します

- **新しいカテゴリー名**: <Mailchimp>
- **API URI**: /webhook/Mailchimp/events/*
- **API アクセスに認証を要求する** - 未選択

保存] をクリックします。

これらの手順を実行すると、Mailchimp が外部ソースとして Journey に追加されます。

Note:

- ユーザーが Journey で外部ソースを設定した場合は、以下のプロパティーを `False` のままにします。これにより、ユーザーは、Journey データの重複解消設定で構成された重要なフィールドごとに、外部ソースからデータを取得できます。
  - `<rule-enabled>false</rule-enabled>\<JourneyEngine>\conf\data-validation-rules.xml`
  - プラットフォームで**「設定」 > 「構成」 > 「Journey」 > 「Journey の構成」**に移動し、**Validation_On_Journey_Records** プロパティーの値を `False` に設定します。
- 外部ソース・ユーザーを構成するには、Unica Platform をインストールまたはアップグレードする際に、Unica Content Integration コンポーネントを前提条件としてインストールする必要があります。

Chapter 16. 設定

設定メニューを使用して、Eメール コネクタ、SMS コネクタ、CRM 接続、REST 統合などの Journey 統合。

デフォルトの電子メール接続の設定する

複数のコネクタがある Unica Link に電子メールを送信する場合、**[設定]** メニューでデフォルトの電子メール接続を設定できます。

About this task

デフォルトの電子メール接続を設定するには、次の手順を実行します。

1. **> Link >「電子メール」**を選択します。
   Result
   [電子メール] ページが表示されます。
2. **[使用可能な接続]**リストから、接続を選択します。
   利用可能な接続には、Mandril、Mailchimp などが含まれます。
3. **「保存」**をクリックします。
   既存の接続を選択解除して、**[保存]** をクリックすることもできます。これにより、デフォルトの接続が設定されていないことが保証されます。

デフォルトの SMS 接続の設定

複数のコネクタがある Unica Link へ SMS を送信する場合は、**設定** メニューでデフォルトの SMS 接続を設定できます。

About this task

デフォルトの SMS 接続を設定するには、次の手順を実行します。

1. **> Link >「SMS」**を選択します。
   Result
   SMS ページが表示されます。
2. **[使用可能な接続]**リストから、接続を選択します。

   Note:

   電話番号の形式は、配信チャネルの仕様に従って記述する必要があります。Journey は、同じ形式で電話番号を配信チャネルに送信します。たとえば、Journey でサポートされている Twilio 接続の電話番号形式は以下のとおりです:

   - *<plus sign><country-code><10-digit phone number>* - `+15403241212`。
   - *<plus sign> <country-code <(area-code)> <three-digit number><four-digit number>* - `+1 (540) 324 1212`。

- *<plus sign>-<country-code>-<area-code>-<three-digit number>-<four-digit number>* - `+1-540-324-1212`。
- *<plus sign> <country-code>-<area-code>-<three-digit number>-<four-digit number>* - `+1 540-324-1212`。

入力した電話番号の形式に関係なく、Unica Journey は以下の形式で番号を保存します。*<plus sign><country-code><10-digit phone number>*。たとえば、電話番号を `+1 540-324-1212` として指定すると、Unica Journey は電話番号を `+15403241212` として保存します。

デフォルトの SMS 接続として Twilio を選択すると、以下の形式の電話番号のみが受け入れられます。*<plus sign><country-code><10-digit phone number>*。たとえば、`+15403241212` です。

3. **「保存」**をクリックします。

デフォルトの CRM 接続の設定

複数の CRM 接続がある場合は、**[設定]** メニューで既定の CRM 接続を設定できます。

About this task

デフォルトの CRM 接続を設定するには、次の手順を実行します。

1. **> Link >「CRM」**を選択します。
   Result
   CRM ページが表示されます。
2. **[使用可能な接続]**リストから、接続を選択します。
3. **「保存」**をクリックします。

デフォルトの ADTECH 接続の設定

複数の ADTECH 接続がある場合は、**[設定]** メニューで既定の ADTECH 接続を設定できます。

About this task

デフォルトの ADTECH 接続を設定するには、次の手順を実行します。

1. **>「リンク」 >「ADTECH」**を選択します。
   Result
   「ADTECH」 ページが表示されます。
2. **[使用可能な接続]**リストから、接続を選択します。
3. **「保存」**をクリックします。

デフォルトのデータベース接続の設定

複数のデータベース接続がある場合、**「設定」** メニューでデフォルトのデータベース接続を設定することができます。

About this task

データベース接続をデフォルトで設定するには、次の手順を実行します。

1. **>「リンク」 >「データベース」**を選択します。
   Result
   「データベース」ページが表示されます。
2. **[使用可能な接続]**リストから、接続を選択します。
3. **「保存」**をクリックします。

デフォルトの PUSH 接続の設定

Unica Link への複数のコネクタで PUSH 送信をする場合、**「設定」**メニューでデフォルトの電子メール接続を設定できます。

About this task

デフォルトの PUSH 接続を設定するには、次の手順を実行します。

1. **> Link >「プッシュ」**を選択します。
   Result
   「プッシュ」ページが表示されます。
2. **[使用可能な接続]**リストから、接続を選択します。
   使用可能な接続には、`Batch_Android` と `Batch_iOS` があります。詳しくは、「接続を管理する on page 68」を参照してください。
3. **「保存」**をクリックします。
   既存の接続を選択解除して、**[保存]** をクリックすることもできます。これにより、デフォルトの接続が設定されていないことが保証されます。

接続を管理する

このメニューから Unica Link 接続を管理できます。

About this task

Mailchimp、Mandrill、Salesforce、Twilio などの Unica Link コネクタを接続を作成できます。すべての既存の接続を「**既存の接続** (*n*)」パネルで表示できます。*n* は接続の数です。

1. Mailchimp 接続を作成するには、次の手順を実行します。

   a. **> Link >「接続の管理」 >「新規作成」**を選択します。
      Result
      [新しい接続の作成]ページが表示されます。

   b. 以下のフィールドに値を指定します。
      - **名前**-必須
      - **説明**-オプション

c. **「次へ」**をクリックします。

d. **[接続の選択]** パネルから、[**Mailchimp**] を選択します。

e. **[接続プロパティ]** パネルで、次の必須フィールドに値を指定します。

Note: 入力するフィールドと値については、『*Unica Link Mailchimp Connector* ユーザー・ガイド』を参照してください。

- **ベース URL**
- **ユーザー ID**
- **API キー**
- **アクティビティー・フェッチ頻度**
- **アクティビティー・フェッチ単位**

f. **[テスト]**をクリックして、接続をテストします。指定された値が正しい場合は、成功メッセージが表示されます。指定した値が正しくない場合は、エラー メッセージが表示されます。

g. 接続を保存するには、**[保存]** をクリックします。
Result
新しい接続が正常に保存され、**[既存の接続]** パネルに表示されます。

2. Mandril 接続を作成するには、次の手順を実行します。

a. **> Link > 「接続の管理」 > 「新規作成」**を選択します。
Result
[新しい接続の作成]ページが表示されます。

b. 以下のフィールドに値を指定します。
- **名前**-必須
- **説明**-オプション

c. **「次へ」**をクリックします。

d. **[接続の選択]** パネルから、 **Mandrill**を選択します。

e. **[接続プロパティ]** パネルで、次の必須フィールドに値を指定します。

Note: 入力するフィールドと値については、*Unica Link* マンドリル ユーザー ガイドを参照してください。

- **API キー**
- **アクティビティー・フェッチ頻度**
- **アクティビティー・フェッチ単位**

f. **[テスト]**をクリックして、接続をテストします。指定された値が正しい場合は、成功メッセージが表示されます。指定した値が正しくない場合は、エラー メッセージが表示されます。

g. 接続を保存するには、**[保存]** をクリックします。

Result

新しい接続が正常に保存され、**[既存の接続]** パネルに表示されます。

3. Salesforce 接続を作成するには、次の手順を実行します。

a. **> Link > 「接続の管理」 > 「新規作成」**を選択します。

Result

[新しい接続の作成]ページが表示されます。

b. 以下のフィールドに値を指定します。

- **名前**-必須
- **説明**-オプション

c. **「次へ」**をクリックします。

d. **[接続の選択]** パネルから、 **Salesforce**を選択します。

e. **[接続プロパティ]** パネルで、次の必須フィールドに値を指定します。

Note: 入力するフィールドと値については、*Unica Link Salesforce* ユーザー ガイドを参照してください。

- **インスタンス URL**
- **アクセストークン**
- **バージョン**

f. **[テスト]**をクリックして、接続をテストします。指定された値が正しい場合は、成功メッセージが表示されます。指定した値が正しくない場合は、エラー メッセージが表示されます。

g. 接続を保存するには、**[保存]** をクリックします。

Result

新しい接続が正常に保存され、**[既存の接続]** パネルに表示されます。

4. Twilio 接続を作成するには、次の手順を実行します。

a. **> Link > 「接続の管理」 > 「新規作成」**を選択します。

Result

[新しい接続の作成]ページが表示されます。

b. 以下のフィールドに値を指定します。

- **名前**-必須
- **説明**-オプション

c. **「次へ」**をクリックします。

d. **[接続の選択]** パネルから、[**Twilio**] を選択します。

e. **[接続プロパティ]** パネルで、次の必須フィールドに値を指定します。

Note: 入力するフィールドと値については、*Unica Link Twilio* ユーザー ガイドを参照してください。

- **ベース URL**
- **アカウント SID**
- **認証トークン**
- **番号から**
- **再試行間隔**
- **再試行回数**

f. **[テスト]**をクリックして、接続をテストします。指定された値が正しい場合は、成功メッセージが表示されます。指定した値が正しくない場合は、エラー メッセージが表示されます。

g. 接続を保存するには、**[保存]** をクリックします。
Result
新しい接続が正常に保存され、**[既存の接続]** パネルに表示されます。

5. Android または iOS 用のバッチ接続を作成するには、以下の手順を実行します。

a. **> Link > 「接続の管理」 > 「新規作成」**を選択します。
Result
[新しい接続の作成]ページが表示されます。

b. 以下のフィールドに値を指定します。
- **名前**- 必須。例えば、`Batch_Android` / `Batch_iOS`
- **説明**-オプション

c. **「次へ」**をクリックします。

d. **「接続の選択」**パネルから、**「バッチ」**を選択します。

e. **[接続プロパティ]** パネルで、次の必須フィールドに値を指定します。
- **ベース URL**
- **ライブ・キー**
- **REST キー**
- **アクティビティー・フェッチ頻度**
- **アクティビティー・フェッチ単位**

f. **[テスト]**をクリックして、接続をテストします。指定された値が正しい場合は、成功メッセージが表示されます。指定した値が正しくない場合は、エラー メッセージが表示されます。

g. 接続を保存するには、**[保存]** をクリックします。
Result
新しい接続が正常に保存され、**[既存の接続]** パネルに表示されます。

Note: Android と iOS の両方にバッチ接続を使用する場合は、それぞれにバッチ接続を作成する必要があります。

REST 統合

REST キーは、アプリケーションへのサードパーティ ログインに使用されます。キーと値のペアを生成し、キーと値のペアを使用して、サードパーティ製アプリケーションを使用して Journey にログインできます。

新しい REST 統合の作成

新しい REST 統合キー ペアを作成するには、次の手順を実行します。

1. **>「REST」**を選択します。
Result
RESTページが表示されます。
2. **+ REST 統合**をクリックします。
Result
新しい REST 統合ページが表示されます。
3. 以下のフィールドに値を指定します。
   - **アプリ名**- 必須。
   - **説明**- オプション。
4. **[キーの生成]** をクリックします。
Result
システムはClientIDと**ClientSecret**を生成します。
5. トグル バーを使用して、**[ステータス]**を `[Active]`または [非アクティブ] に変更します。デフォルトでは、**ステータス**は`Active`です。
6. REST 統合を保存するには、**[保存]** をクリックします。
Journey にオーディエンス データを送信するには、REST エンドポイントの構成に使用される REST エントリ ソースに記載されている詳細に従います。手順 (4) を実行したときに受け取ったClientIDと**ClientSecret**を使用して、エントリ ソースで REST エンドポイントを構成します。
7. 次の URL 例を使用して認証トークンを生成します。「http://comp-4946-.nonprod.hclpnp.com:80/journey/api/thirdpartylogin」。この認証トークンとエントリー ソース コードを使用して、データを送信します。

Note: エントリー・ソース・コードは REST エントリー・ソースでデータを送るには必須です。

8. 8.REST API エンドポイント (REST エントリー ソース作成ページで使用可能) 例 - http://comp-4946-1.nonprod.hclpnp.com:80/journey/api/entrysources/rest/data
9. /journey/api/entrysources/rest/data が入力として使用される JSON のサンプル形式は次のとおりです。

{

"entrySourceCode": "ES-00000125",

"data": [

{ "Email": "pooja_sharma@hcl.com", "FirstName": "Pooja", "LastName": "Sharma", "Age": 30, "Address": 125, "CreatedDate": "15 09 22", "PHONE_NUMBER": "+919623444160", "DeviceID":"ac79649c-ca1b-4c3f-99ce-56d5ad69fbba" }

]

}

Note: json フィールドは、ジャーニーに関連付けられているデータ定義に従って変更する必要があります。

10. REST エントリー ソースのデータをプッシュする前に、ジャーニーを公開状態にする必要があります。

REST 統合リストを表示します

Unica Journey は、作成された REST 統合のリストを維持します。

About this task

REST 統合のリストを表示するには、次の手順を実行します。

1. **>「REST」**を選択します。
   Result
   REST ページが表示されます。
2. 次の操作のいずれかを実行します。
   a. **[名前]** フィールドで REST 統合のリストを昇順または降順で表示するには、[名前] をクリックします。
   b. [説明] フィールドで REST 統合のリストを昇順または降順で表示するには、**[説明]** をクリックします。

既存の REST 統合の変更

既存の REST 統合を説明とステータスのみを変更できます。

About this task

既存の REST 統合を変更するには、次の手順を実行します。

1. **>「REST」**を選択します。
   Result
   REST ページが表示されます。
2. 残りの統合を変更するには、次のいずれかを実行できます。
   Choose from:

- リストから必要な REST 統合を選択します
- 選択 **>**

Result

REST 統合の更新ページが表示されます。

3. 次のフィールドのみを更新できます。

Choose from:

- **説明**
- **ステータス**

4. 変更を保存するには、**[保存]** をクリックします。

REST 統合の削除

使用されなくなった、または不要になった非アクティブな REST 統合のみを削除できます。

Before you begin

REST 統合エントリのステータスを変更するには、既存の REST 統合の変更 on page 73 を参照してください。

About this task

既存の非アクティブな REST 統合を削除するには、次の手順を実行します。

1. **>「REST」**を選択します。

Result

RESTページが表示されます。

2. 次のいずれかの手順を実行します。

Choose from:

- REST 統合を削除するには、 **>** を選択して、リスト内の REST 統合を成功させます。
- 複数の REST 統合を削除するには、リストで削除する REST 統合の前にあるチェックボックスを選択し、**[削除]** をクリックします。

3. 確認のダイアログ・ボックスが表示されます。削除を続行するには、**[OK]** をクリックします。

カスタム・キー・ストアと信頼ストアの使用

セキュアな REST API、つまり、[https://|https:] で始まる URL の API 用に REST タッチポイントを構成する場合、サードパーティ REST API の実装によっては、SSL キー証明書を構成しなければならない場合があります。

通常、SSL 証明書は Java インストール・フォルダで構成することも、まったく異なるフォルダで構成することもできます。後者の場合、システム管理者は、Journey エンジン、ファイル `application.properties` で、以下の設定を行う必要があります。 `<installation location of journey application>\Engine\application.properties`

- **ssl.restclient.custom.store**

  true: Journey アプリケーションの信頼ストアとキーストアが、あるカスタム・ロケーション (またはフォルダ) に存在し、ジャーニーとサードパーティ・システムの証明書を Java ランタイムのインストール・ディレクトリーではなく、そのロケーションにインポートする場合。

  false: 証明書を Java ランタイムのインストール・ディレクトリーにインポートする場合。

- **ssl.restclient.truststore.defaultalgorithm**

  要求された信頼管理アルゴリズムの標準名。

  標準アルゴリズム名について詳しくは、https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html を参照してください。

  値の例: SunX509

- **ssl.restclient.keystore.defaultalgorithm**

  要求されたアルゴリズムの標準名。標準アルゴリズム名について詳しくは、https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html を参照してください。

  値の例: SunX509

- **ssl.restclient.truststore.type**

  信頼ストアのタイプを表します (デフォルトは JKS)。

  信頼ストアとは、デジタル証明書 (特に公開鍵証明書) のリポジトリーまたは保管場所であり、セキュアな通信の信頼を確立するために使用されます。

  信頼ストアには、認証局 (CA) や信頼できるサーバーなど、信頼できるエンティティーの公開鍵証明書が格納されます。

  SSL/TLS ハンドシェイク・プロセス中にリモート・エンティティー (Web サイトやサーバーなど) の信頼性を検証するために使用されます。

- **ssl.restclient.truststore.location**

  ファイル・システム上のフォルダまたはディレクトリー。Journey アプリケーションの信頼ストアを表します。

- **ssl.restclient.truststore.password**

  通常、信頼ストアはパスワードで保護されます。このパラメーターは、信頼ストアのパスワードを表します。

- **ssl.restclient.keystore.type**

  キーストアのタイプを表します (デフォルトは JKS)

  キーストアは、暗号キーと証明書の保管場所を表します。通常、キーストアには、データの暗号化と復号化、デジタル署名の作成、および安全な通信の確立に不可欠な秘密鍵が格納されます。

- **ssl.restclient.keystore.location**

ファイル・システム上のフォルダまたはディレクトリー。Journey アプリケーションのキーストアを表します。

- **ssl.restclient.keystore.password**

  通常、キーストアはパスワードで保護されています。このパラメーターは、キーストアのパスワードを表します。

- **ssl.restclient.key.password**

  通常、キーストア内の鍵もパスワードで保護されます。このパラメーターは、鍵のパスワードを表します。

- **ssl.restclient.protocols**

  ネットワーク上で安全な通信を提供するように設計された、サポートされている暗号プロトコルのコンマ区切りリスト

  推奨値: TLSv1.2

- **ssl.restclient.truststore.password.encrypted**

  true または missing に設定した場合は、ssl.restclient.truststore.password にパスワードの暗号化された値を指定します。

  false に設定した場合は、ssl.restclient.truststore.password にパスワードのプレーン・テキスト値を指定します。

  Note: パスワードを空白にする場合は、このフラグを false に設定します。

- **ssl.restclient.keystore.password.encrypted**

  true または missing に設定した場合は、ssl.restclient.keystore.password にパスワードの暗号化された値を指定します。

  false に設定した場合は、ssl.restclient.keystore.password にパスワードのプレーン・テキスト値を指定します。

  Note: パスワードを空白にする場合は、このフラグを false に設定します。

- **ssl.restclient.key.password.encrypted**

  true または missing に設定した場合は、ssl.restclient.key.password にパスワードの暗号化された値を指定します。

  false に設定した場合は、ssl.restclient.key.password にパスワードのプレーン・テキスト値を指定します。

  Note: パスワードを空白にする場合は、このフラグを false に設定します。

  パスワードを暗号化するには、<Journey install location>/tools フォルダにある次のユーティリティーを実行してください。`JourneyEncryptionUtility.bat(or .sh)`

  使用法: JourneyEncryptionUtility.bat (または .sh) <plaintext password>

  例

プレーン・テキストのパスワードが「abcd」の場合は、JourneyEncryptionUtility.bat (または.sh) abcd となります。

次のような出力が表示されます。

Entered String is : abcd

暗号化された文字列: MIuFcm7mkspvIMEx7XywAA==

対象のパラメーターの application.properties に、暗号化された値 MIuFcm7mkspvIMEx7XywAA== を設定します。前記の 3 つのパラメーターすべてに異なるパスワード値を使用している場合は、このユーティリティーを 3 回実行し、それに応じて値を使用します。

Journey Proxy 統合

Proxyサーバーは、旅ウェブとエンジンのプロジェクトに統合され、これにより、ユーザーはセキュリティを追加し、アプリケーションサーバーをProxyサーバーの背後に保つことができるようになりました。Proxyサーバーは、Deliver、Link、Platformの各サーバーとやり取りを行います。

Journey Web - Deliver、Link、Platformサーバーと通信し、構成の詳細を取得したり、Journeyにメール/SMS/AdTech Pointを統合したりします。

Journey Engine - Proxyを使用してDeliver/Link Serverとの通信を行い、Eメール/SMS/Adtechの詳細をエンドサーバーに送信します。

Journey Webでサポートされているproxy

1. SOCKS
2. HTTP
3. HTTPS

Journeyエンジンでサポートされているproxy

1. HTTP

Note: EngineがDeliverと通信するために使用するSOAP（Apache Axis2）では、SOCKSおよびHTTPS Proxyはサポートされていません。

エンジンの application.properties ファイルでエンジン用に設定するプロパティ

- journey.proxy.type=NONE
- spring.proxy.host=[IP]
- spring.proxy.port=[PORT]

- spring.proxy.username=[username]
- spring.proxy.password=[password]

Web application.properties ファイルで Web 用に設定されるプロパティ

- journey.proxy.type=NONE
- spring.proxy.host=[IP]
- spring.proxy.port=[PORT]
- spring.proxy.username=[username]
- spring.proxy.password=[password]
- server.use-forward-headers=true

Note: journey.proxy.typeプロパティのデフォルト値はNONEで、NONEに設定するとProxyは無効となる。

Journey エンジン接続プールの設定

- journey.datasource.maxpool.size=[MAX_POOL_SIZE] – DB接続プールのサイズを設定します。
- journey.datasource.minIdle.size=[MIN_IDLE_SIZE] – 最小アイドル接続のサイズを設定します。

Developer Tools

デベロッパーツールの一覧を表示します。

API 文書

ユーザーは、JourneyのRESTAPIのリストを見つけることができます。