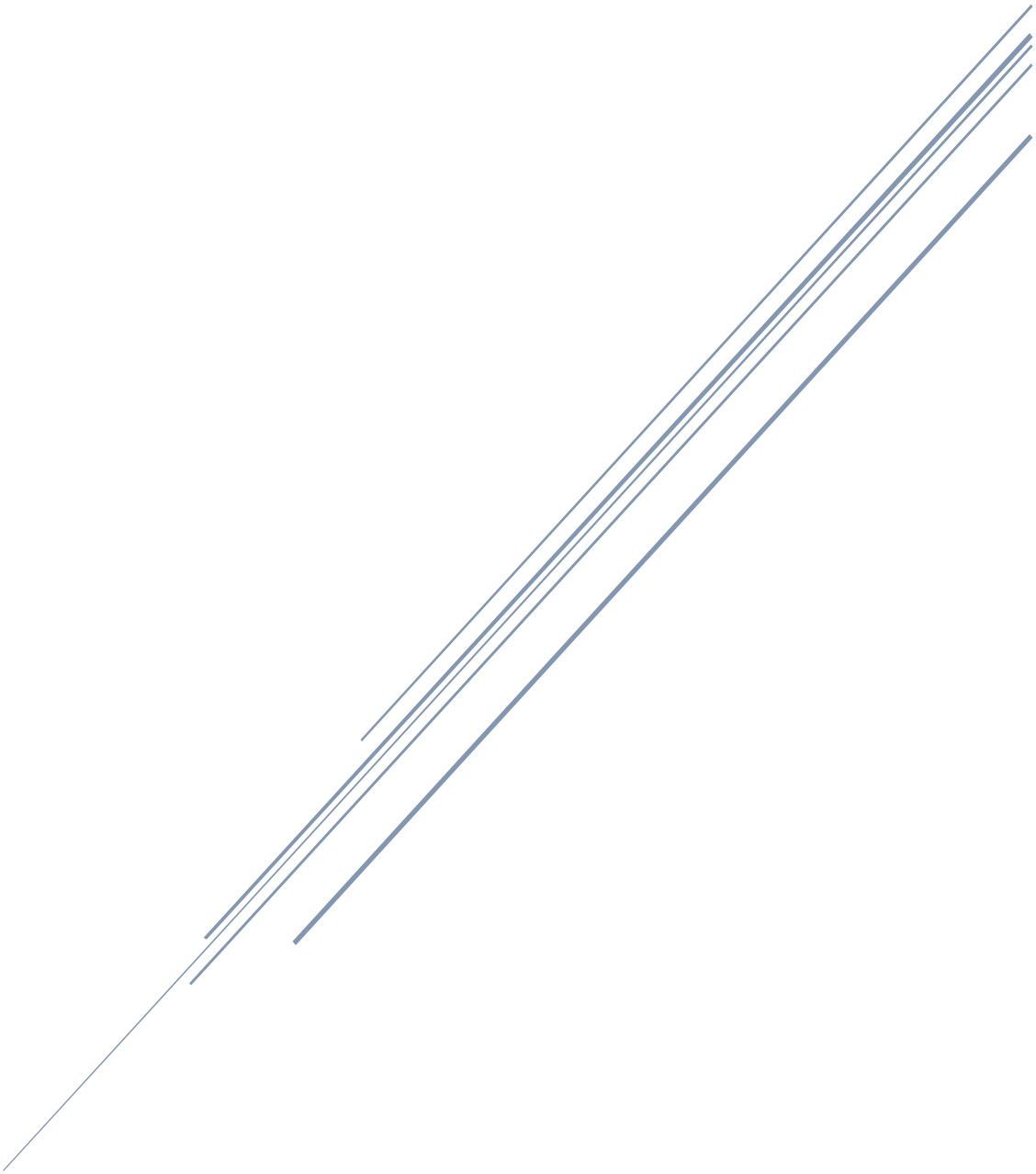
CIOF システム API 利用手順書

2022 年 1 月 27 日　Ver2.12



目次

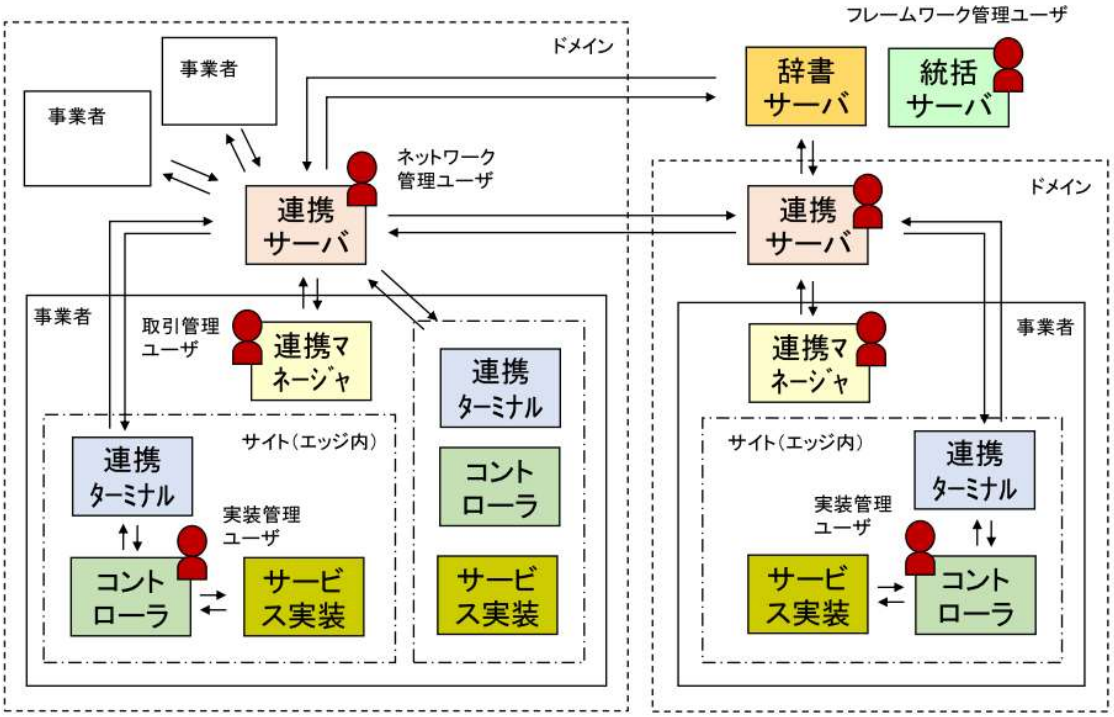
1 本書の目的

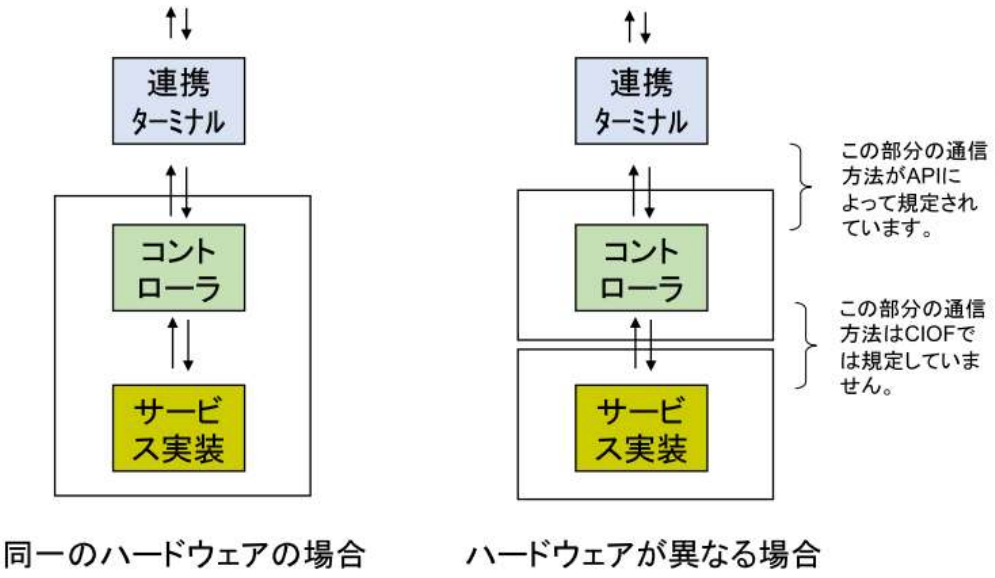
本書は、CIOF システムコントローラ用 API リファレンスマニュアルで規定されている API の実際の使用方法について、例を用いて説明しています。

2 システム構成

2.1 アーキテクチャ

CIOF システム全体アーキテクチャおよび今回説明する対象であるコントローラの位置付けおよび API の規定範囲は、次の図で示す通りです。





2.2 用語

CIOF システム外部仕様書より用語説明を抜粋しています。各用語の詳細については、CIOF システム外部仕様書を参考にしてください。

2.2.1 データ取引（Data Trading）

データ取引は、異なるサイト間でデータを伝送する場合、あらかじめ、データの提供者とデータの利用者の双方で、データに関する権利および義務を定め、それらを履行することをいう。１つの取引は、その契約の締結から実際のデータの伝送、そしてそれに伴う権利と義務の履行、そして契約の失効までの一連の行為を含む。

2.2.2　取引データ（Trading Data）

データ取引の対象となる個々のデータのこと。連携サーバが固有の ID と付与する。連携サーバ ID と取引データ ID でグローバルにユニークとなる。取引データは、ハッシュが生成され、それを取引データの実体の代わりに保存することで、取引データ間の突合せを可能とする。また、取引データのトレーサビリティは、このハッシュを用いる。

2.2.3　連携サーバ（Hyper Connection Server）

複数の異なるサイト間でデータ流通を仲介するサーバ。履歴を保持し、要求に応じてその内容を照会可能とする。連携サーバは複数存在する場合があり、それぞれが管理するドメイン間でのデータ流通も併せて行う。

2.2.4　連携ターミナル（Hyper Connection Terminal）

取引データを実際に利活用するサイトの単位となり、そこに含まれるコントローラに対して、取引データ提供、データ利用に対する要求に対応する。各サイトのプライベートなネットワーク内に配置され、ローカルな IP アドレスを持つ。インターネットとは、あらかじめ設定した連携サーバとのみ通信する。

2.2.5　辞書サーバ（Hyper Dictionary Server）

取引契約等で他の事業者との間でデータ取引の内容について意思決定する際に用いる用語について、あらかじめその意味や用法について記述するための個別辞書、共通辞書、外部辞書を管理する。辞書の登録や修正、検索などに対応する。

2.2.6　連携マネージャ（Hyper Connection Manager）

事業者がデータ取引に関する定義や管理を行うためのシステムであり、クラウド上で提供される。辞書の管理、契約の権利、個別の実装の管理、および取引内容や履歴の管理などを扱う。事業者ごとに設定されたユーザがログインし、それぞれの権限に応じた管理業務を行うことができる。

2.2.7　データ変換サーバ（Data Translation Server）

データ取引において、データの提供時または利用時に、取引契約にあるデータプロファイルの定義にもとづいて、取引データの内容を変換するサーバ。連携ターミナルから呼ばれるリクエストに対応して変換後のデータを返す。

2.2.8　コントローラ（Edge Control Unit）

連携ターミナルと直接通信し、データの受け渡しを行うソフトウェア。各連携ターミナルが管轄するサイトごとに最低１つ存在し、データ取引（提供または利用）のための処理を実施する。各サイト内で、それぞれの目的に応じて利用するデータを管理し

データ処理制御する。エッジ内部に配置される場合は特に、エッジコントローラと呼ぶ場合もある。

2.2.9　サービス構成モデル（Service Component Model）

ユーザからみて業務として意味を把握できる単位。業務プロセスの１ステップとして、そこで得られた結果は、他の業務で利用可能となる。プロセス構成モデルを要素として持つ。

2.2.10　データ構成モデル（Data Component Model）

データ取引におけるデータのクラス定義。リレーショナルデータベースでは、テーブルまたはビューに対応する。１つ以上のデータ項目定義によって構成される。定義された内容は、メタデータとして、辞書サーバに登録される。データ構成モデルに対応して具体的なデータレコードが設定される。

2.2.11　事業者（Trade Account Party）

データ流通を実施するに際して、データ提供またはデータ利用を行うそれぞれのサイトの所有者。取引契約の当事者となる。プラットフォーム事業者である場合は、共通辞書の登録ができる。また、コンポーネント事業さである場合は、外部辞書が登録できる。

2.2.12 取引契約（Trade Contract）

取引データを２つのサイト間で流通させる場合に、それぞれの事業者間であらかじめ交わすデータ取引に関する契約の内容を示す。この内容は双方の事業者で共有する。取引契約は、契約プロファイル、データプロファイル、サービスプロファイルで構成される。

3 連携ターミナルの準備

連携ターミナルの準備については、別紙「CIOF システム操作手順書」を参照してください。

4 事業者・ユーザ情報を更新する

事業者・ユーザ情報を更新する方法については、別紙「CIOF システム操作手順書」を参照してください。

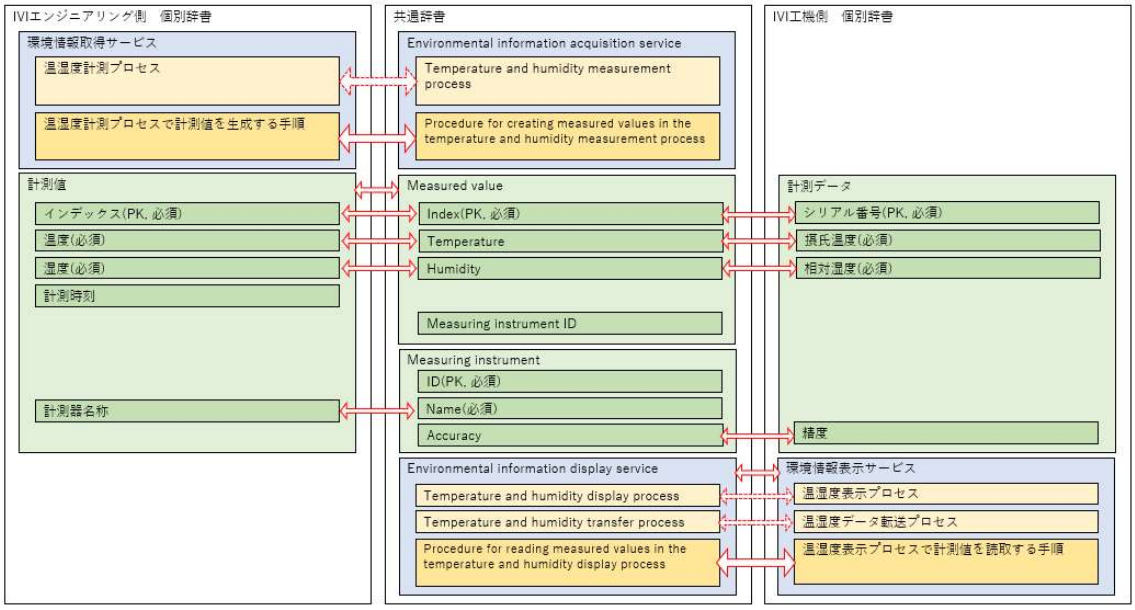
5 前提となる辞書および実装

本例は、次のような辞書データおよび実装データに基づいて説明しています[1]。辞書データおよび実装データの登録方法については、別紙「CIOF システム操作手順書」を参照してください。

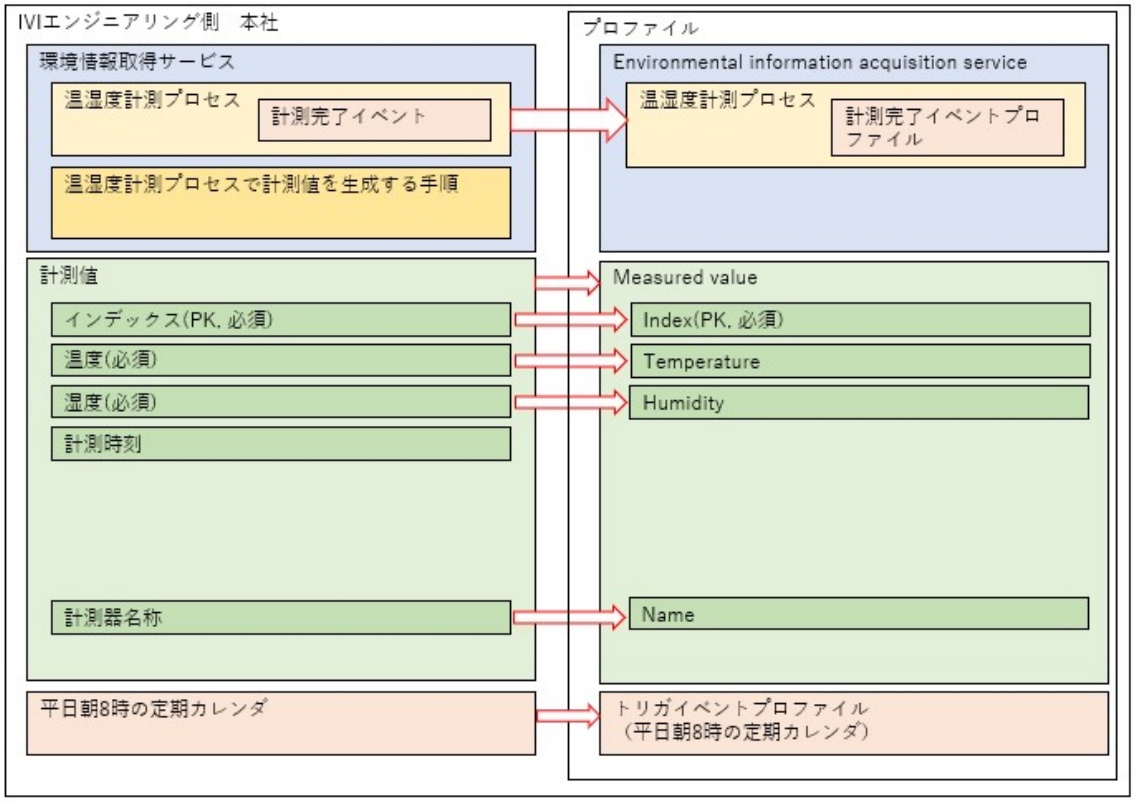
IVI エンジニアリング、IVI 工機、共通辞書の間での各種モデルの関係を、次に示します。

[1] パブサブ通信例で登場する IVI テクノの辞書データについては省略

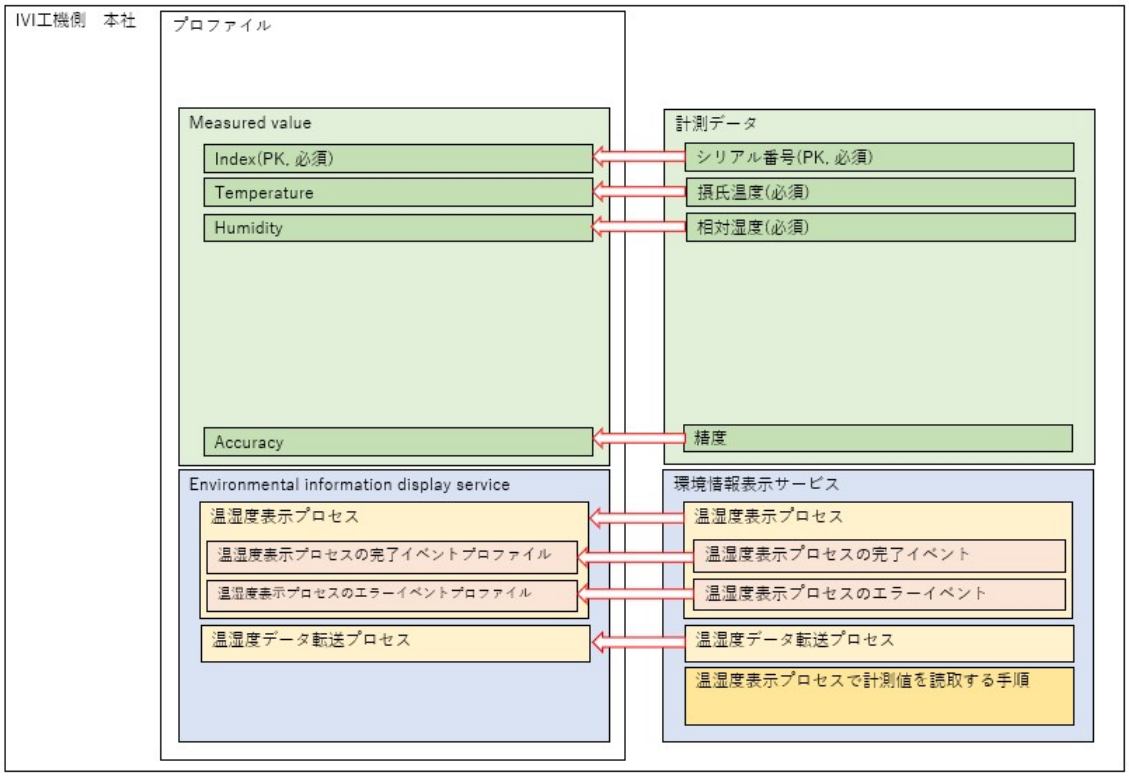
辞書の関係



IVI エンジニアリング側の実装とプロファイルを、以下に示します。



IVI 工機側の実装とプロファイルを、以下に示します。



6 コントローラ用 API の利用方法

コントローラが使用する API は、以下のように全部で 12 種類あります。

No	名称	説明	メソッド
1	コントローラの状態通知	コントローラの起動および終了を通知します。	PUT
2	サービス実装の取得	サービス実装、プロセス実装、イベント実装の構造を取得します。	GET
3	サービス実装の状態通知	サービス実装、プロセス実装、イベント実装の状態を通知します。	PUT
4	データ実装の取得	データ実装、データ項目実装の構造を取得します。	GET
5	データ実装の状態通知	データ実装、データ項目実装の状態を通知します。	PUT
6	取引契約の取得	取引契約の内容を取得します。	GET
7	カレンダの取得	カレンダの定義を取得します。	GET
8	取引データの取得	取引データを取得します。	GET
9	取引データの送信	契約内容に基づき取引データを送信します。	POST
10	リクエストパラメータの取得	リクエストパラメータを取得します。	GET
11	リクエストパラメータの送信	リクエストパラメータを送信します。	POST
12	サービス記録の通知	サービス記録を送信します。	POST

コントローラ用 API の詳細仕様については、別紙「CIOF システムコントローラ用 API リファレンスマニュアル」を参照してください。本章では、コントローラ用 API の使用手順について、利用場面を例にして説明します。

6.1 事前準備

コントローラを CIOF システムに接続するための事前準備について、説明します。

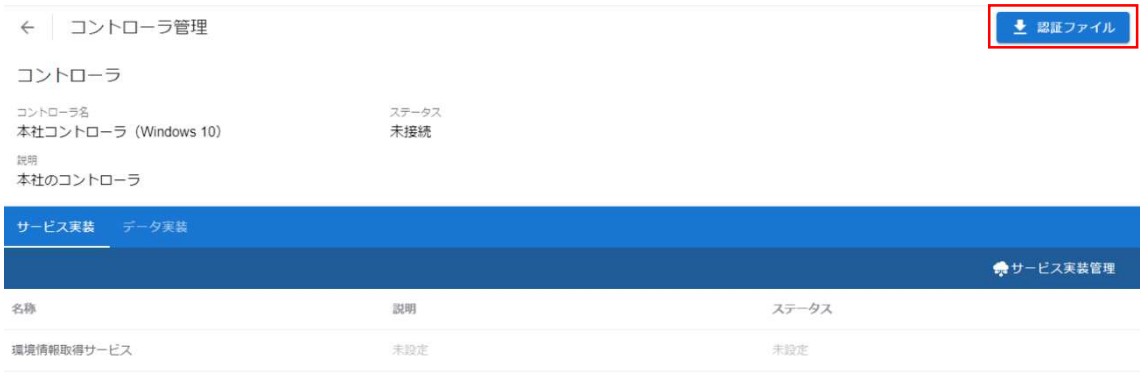
6.1.1　認証キーの設定

コントローラから連携ターミナルの API を利用する際には、HTTP ヘッダに認証キーを設定する必要があります。本節では、認証キーの設定方法について、説明します。

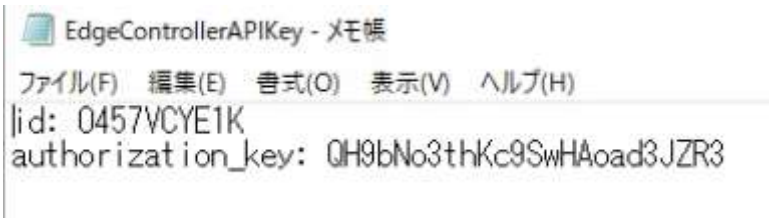
初めに、コントローラの認証ファイルを入手します。連携マネージャのホーム画面から「サイト実装管理」を選択します。

連携ターミナルと通信したいコントローラの右側にある編集ボタンを押下します。

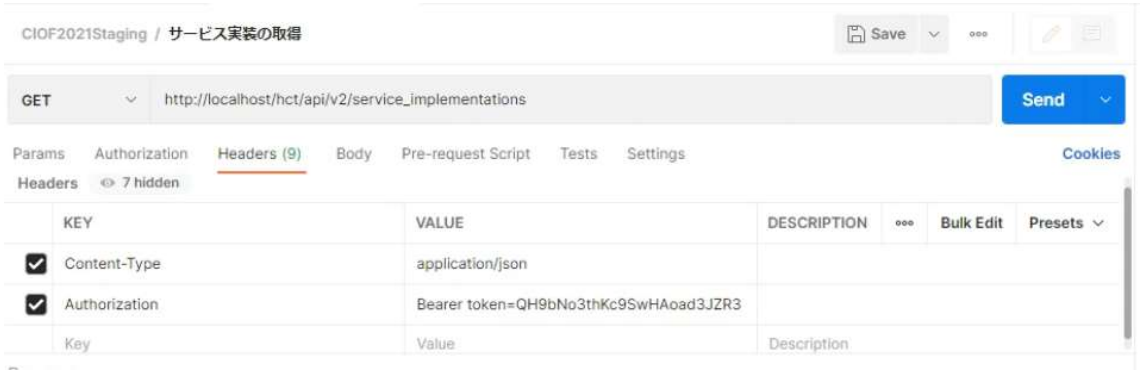
赤枠内の「認証ファイル」を押下します。



認証ファイルがダウンロードされますので、その中身をメモ帳などで確認します。



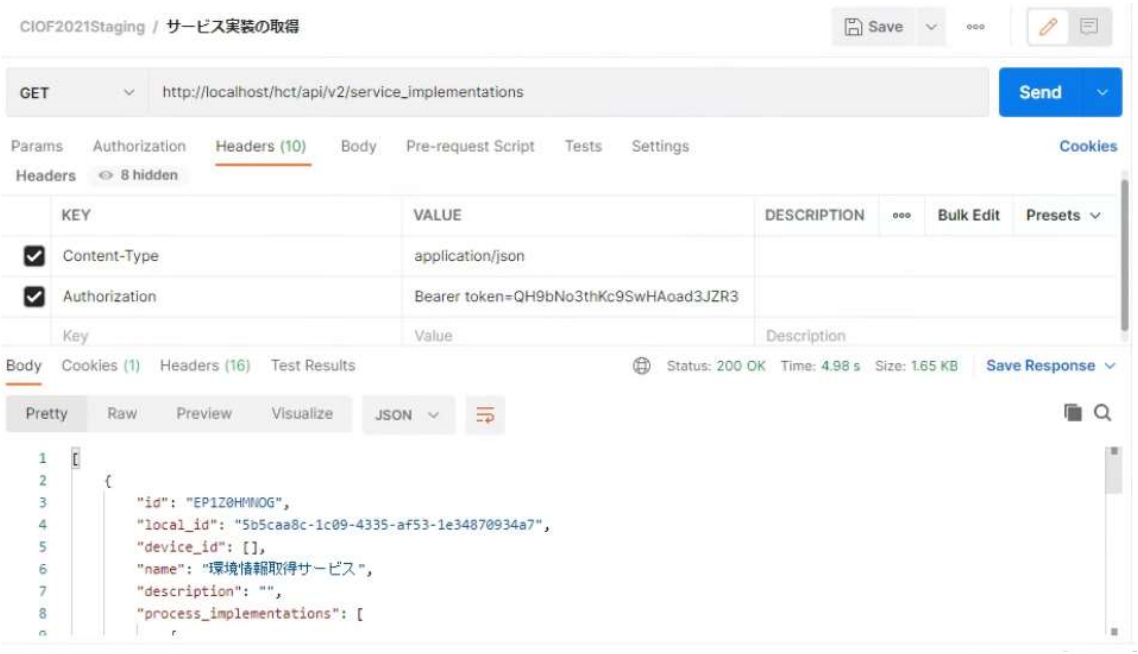
authorization_key を HTTP のヘッダに記載します。



※本資料においては、コントローラとして連携ターミナルと同じ PC にインストールした Postman と呼ばれるツールを用います。Postman のインストールは次の URL から行ってください。無償利用の範囲で動作確認をすることができます。

https://www.postman.com/

Send ボタンを押下することで、HTTP リクエストを連携ターミナルに送信し、サービス実装が取得されることが確認できました。



6.1.2　内部 ID（実際のソフトウェア）と CIOF システムで作成した実装との紐づけ

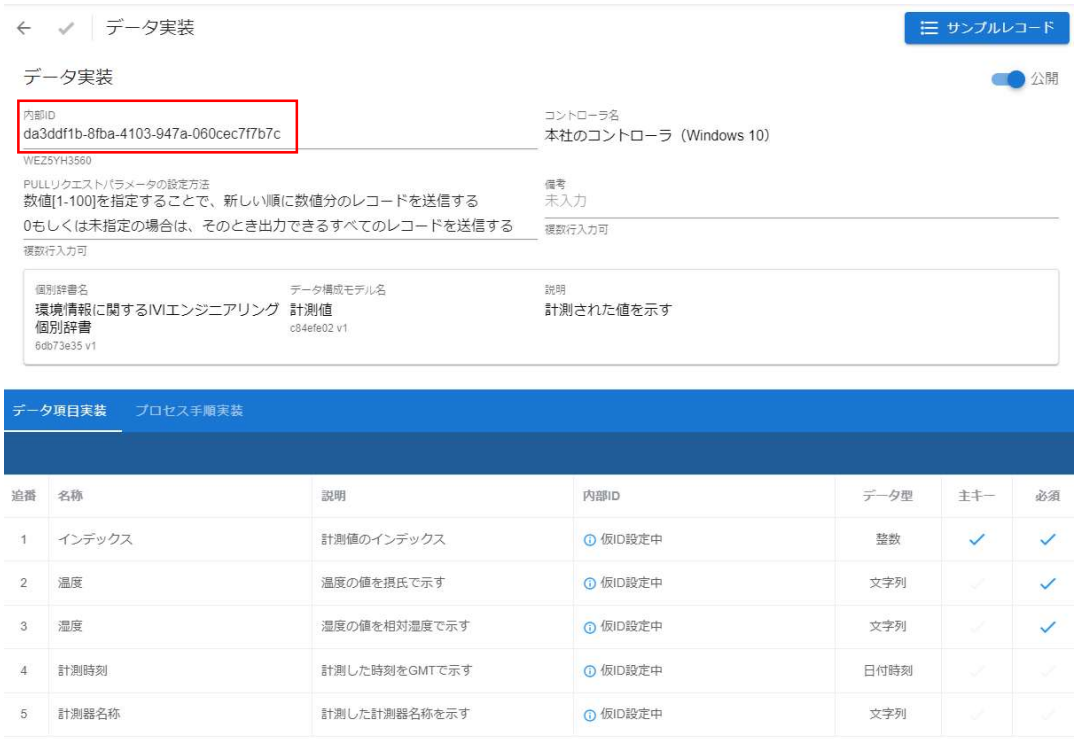
内部 ID と CIOF システムで作成した実装との紐づけ方法には、以下の 2 パターンがあります。

6.1.2.1　コントローラ側のファイルで紐づける場合

コントローラが管理するファイルなど（SDK にサンプル実装があります）を用いて、コントローラ側のインターフェースによって内部 ID を入力させ、内部で CIOF システムが発番した ID と紐づけます。

6.1.2.2　連携マネージャへ内部 ID を入力することで紐づける場合

連携マネージャには、各実装を定義する際に内部 ID を入力する欄があります。その欄に CIOF システムが発番した ID と紐づけたい内部 ID を入力します。例えば、データ実装の場合は、次の赤枠内に入力します。



6.2　コントローラ起動時処理

コントローラが起動したら、初めに CIOF システムと同期を行う必要があります。ここでは、API を用いてコントローラと CIOF システムとの同期のとりかたについて、実行すべき内容を順に説明します。

6.2.1　コントローラ設定および接続通知

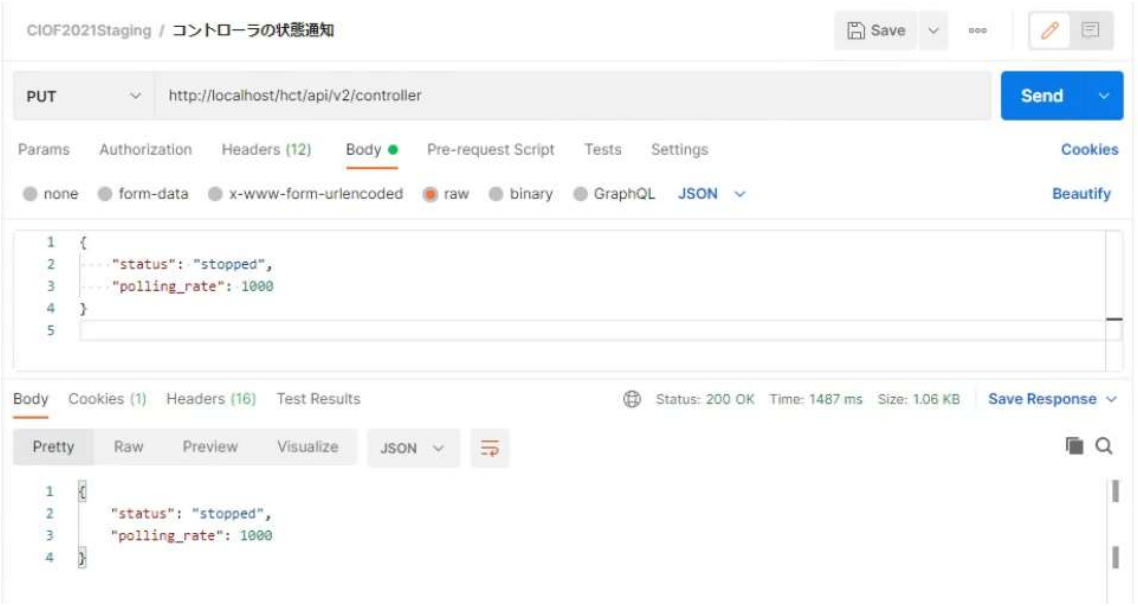
CIOF システムに接続されるすべてのコントローラは、自身の設定および接続状態について、CIOF システム側へ通知する必要があります。

コントローラの電源が ON し、連携ターミナルとの通信が可能となったタイミングで、コントローラ状態通知 API にて、status を disconnected から stopped へ変更すると同時に、自身で決めた polling_rate (=取引データの取得 API およびリクエストパラメータの取得 API を呼び出す間隔)を設定します。

コントローラは、ここで設定した polling_rate 通りの間隔で連携ターミナルに対して API を用いて取引データおよびリクエストパラメータを取得しなければなりません[2]。

以下に Postman による実行例を示します。

[2] polling_rate を大きく超える期間、メッセージ取得 API が呼び出されていない場合、CIOF システムが status を stopped にする仕様が、将来的に追加される見込みです



この対応によって、CIOF システムにコントローラの状態およびポーリング周期が通知されますが、それによって通信に制限がかかることはありません。一方で、管理者は連携マネージャから、コントローラの状態を確認することができます。次に、コントローラの状態確認手順を示します。

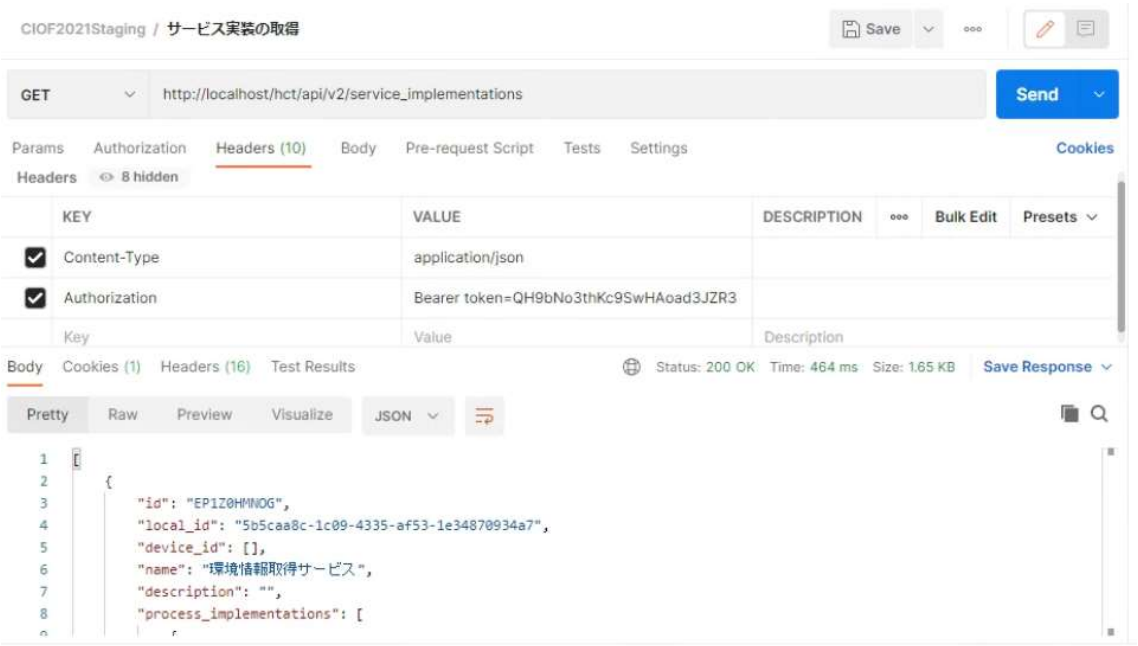
連携マネージャのホーム画面から「サイト実装管理」を選択します。

確認したいコントローラ名称の行に、コントローラのステータスとポーリング周期が表示されています。

6.2.2　コントローラ配下の各実装状態の通知

CIOF システムに接続されるすべてのコントローラは、自身の配下における各実装の状態を確認し、CIOF システム側へ通知する必要があります。ここでは、API を用いた状態通知の手順を説明します。

初めに、サービス実装の取得 API とデータ実装の取得 API を用いて、コントローラ配下のサービス実装、プロセス実装、イベント実装[3]、データ実装、データ項目実装を取得します。以下に Postman による実行例を示します。

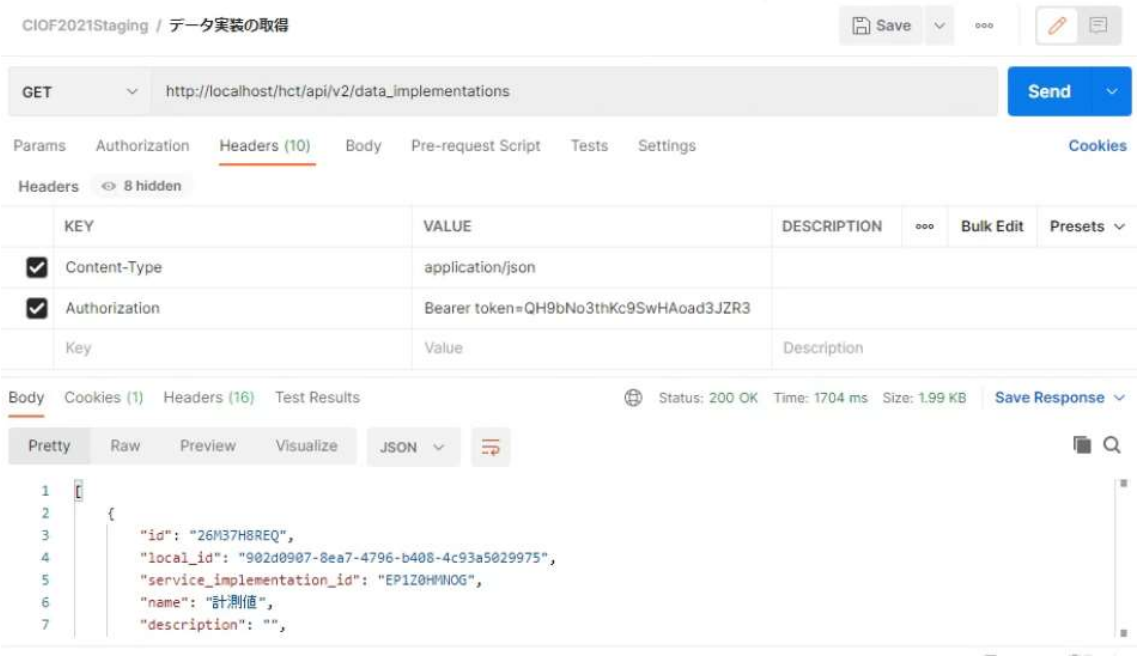


[3] ここで取得されるイベント実装は、該当するサービス実装に紐づくイベント実装のうち、以下の条件を満たすものに限られますので注意してください。

①イベントプロファイルが存在し、記録可能であるもの(モニタ)

②イベントプロファイルが存在し、トリガ実装が存在するトリガとなるイベント実装であるもの(トリガ)

なお、②の場合においては、カレンダも応答含まれます

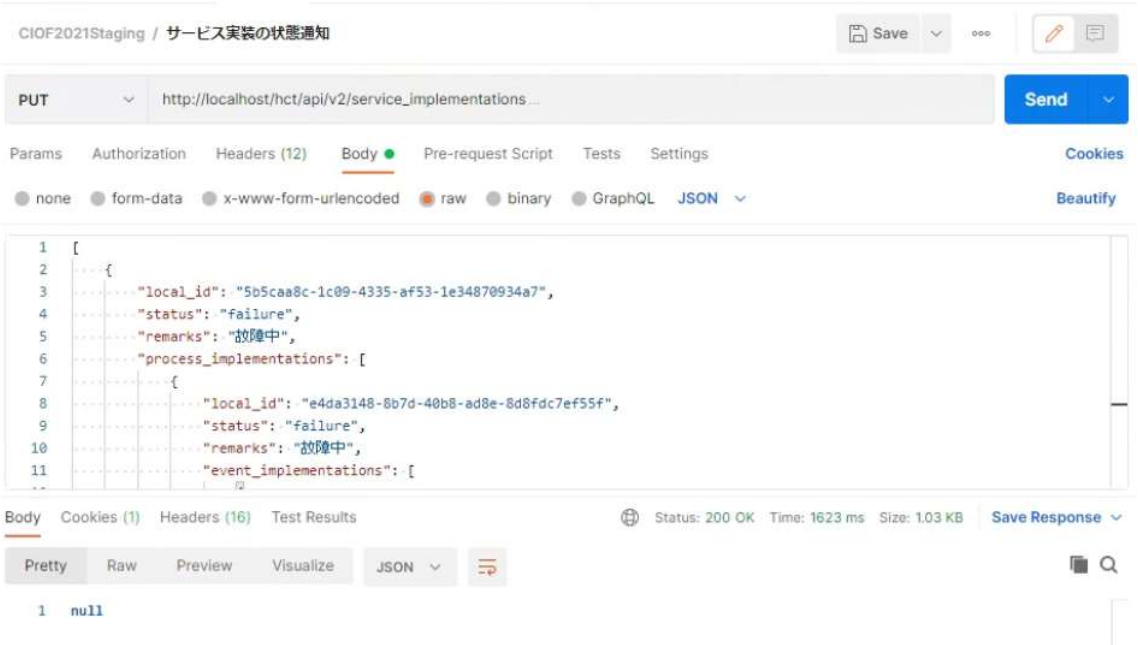


取得した各実装に対して現物（リアル世界に存在するソフトウェアモジュール）がどのような状態であるかを確認し、状態を設定します。

サービス実装・プロセス実装・イベント実装は、正常”ready”、停止“halt”、故障“failure”、例外“exception”の 4 つの状態を持ち、データ実装は、正常”valid”，異常“invalid”，例外“exception”の 3 つの状態を持ち、データ項目実装は、正常”valid”，例外“exception”の 2 つの状態を持ちます。

取得した各実装に対して現物（リアル世界に存在するソフトウェアモジュール）が存在していない場合は、例外“exception”を通知します。現物として存在しているにもかかわらず、取得した各実装の中に存在しない場合は、そのサービスやデータに関する

情報について、CIOF システムを用いて送受信したり操作したりすることはできません。以下に Postman による実行例を示します。



6.2.3　内部 ID の通知

内部 ID を各サービス実装およびデータ実装に設定することで、既存システムやエンドユーザが機器購入時点において割り当てられていた ID を活用することができます。内部 ID の導入は必須ではありません。ここでは、内部 ID を用いる場合に API を用いて通知する手順を説明します。

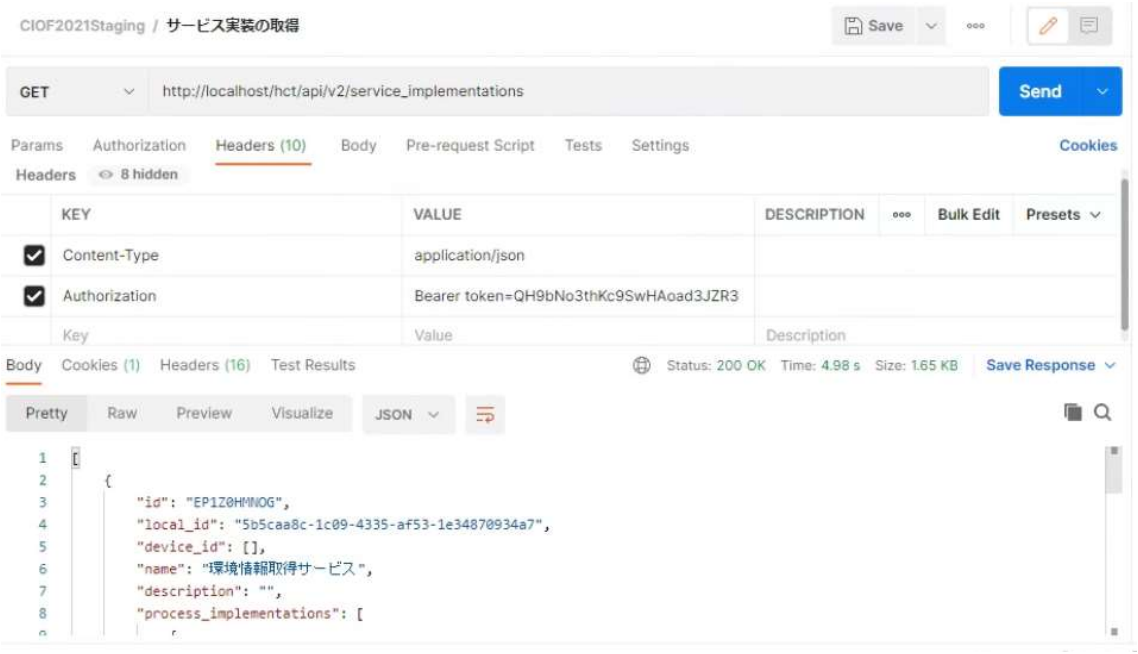
内部 ID を設定するためには、サービス実装の状態通知 API およびデータ実装の状態通知 API を用います。HTTP リクエスト時に、"id"を指定した各実装の"local_id"に指定された文字列が内部 ID として CIOF システムへ登録されます。以後、サービス実

装を通知する場合には、"id"を指定せずに"local_id"のみで該当の各実装にアクセスすることができるようになります。内部 IDは、登録時に次のような範囲でのユニーク制約がありますので、注意してください。

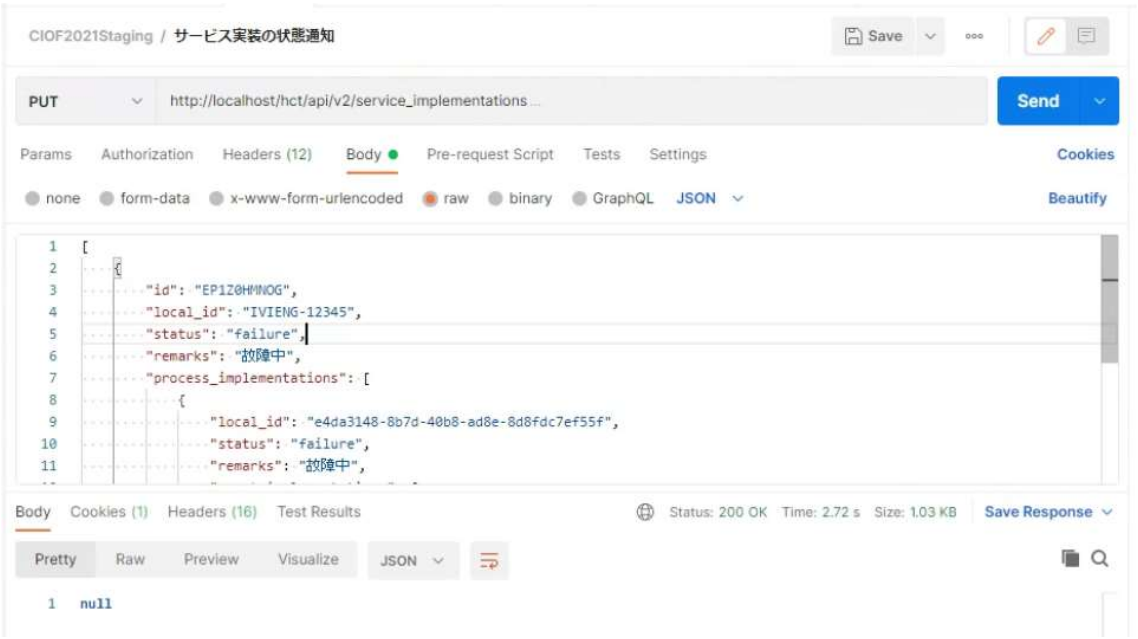
名称	ユニーク制約範囲
サービス実装内部 ID	コントローラ（サイト）
プロセス実装内部 ID	サービス実装
イベント実装内部 ID	プロセス実装
データ実装内部 ID	コントローラ（サイト）
データ項目実装内部 ID	データ実装

以下に Postman によるサービス実装の状態通知 API を用いた実行例を示します。（データ実装の状態通知 API を用いた登録方法も同様の手順ですので、省略します）

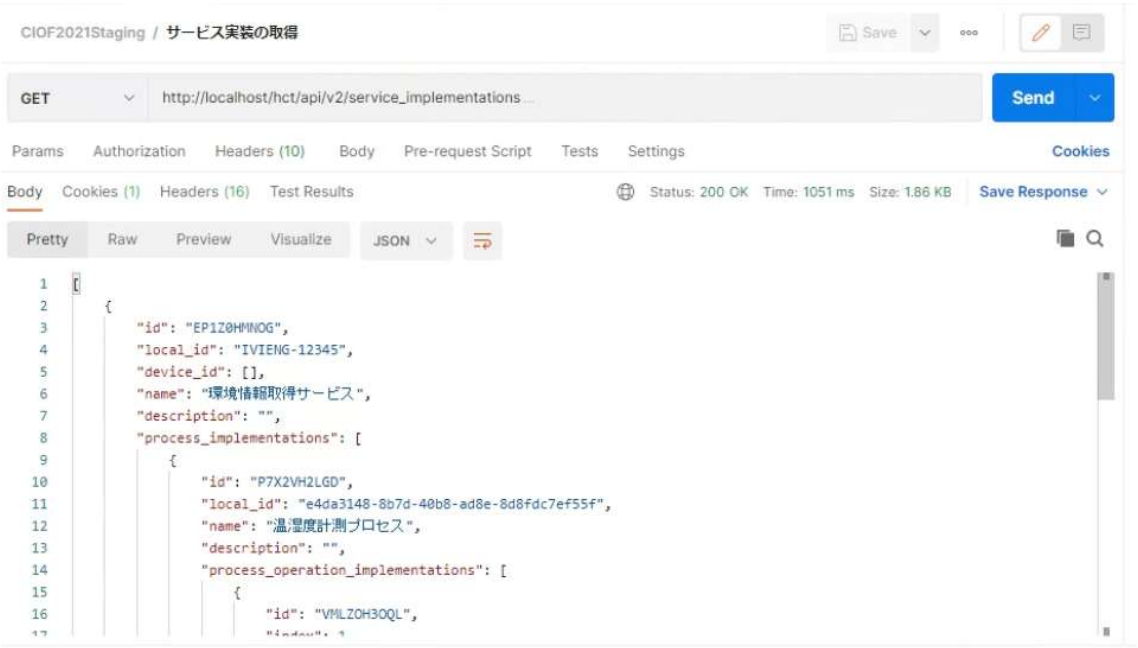
初めにサービス実装取得 API によって、CIOF システムが管理する id を確認します。



id と local_id を紐づけ、サービス実装の状態通知 API で登録します。ここでは、id が EP1Z0HMNOG であるサービス実装の local_id を IVIENG-12345 と設定します。以下に Postman による実行例を示します。



最後に、もう一度、サービス実装取得 API を用いることで、CIOF システムが管理する id と local_id が紐づいたことを確認します。以下に Postman による実行例を示します。

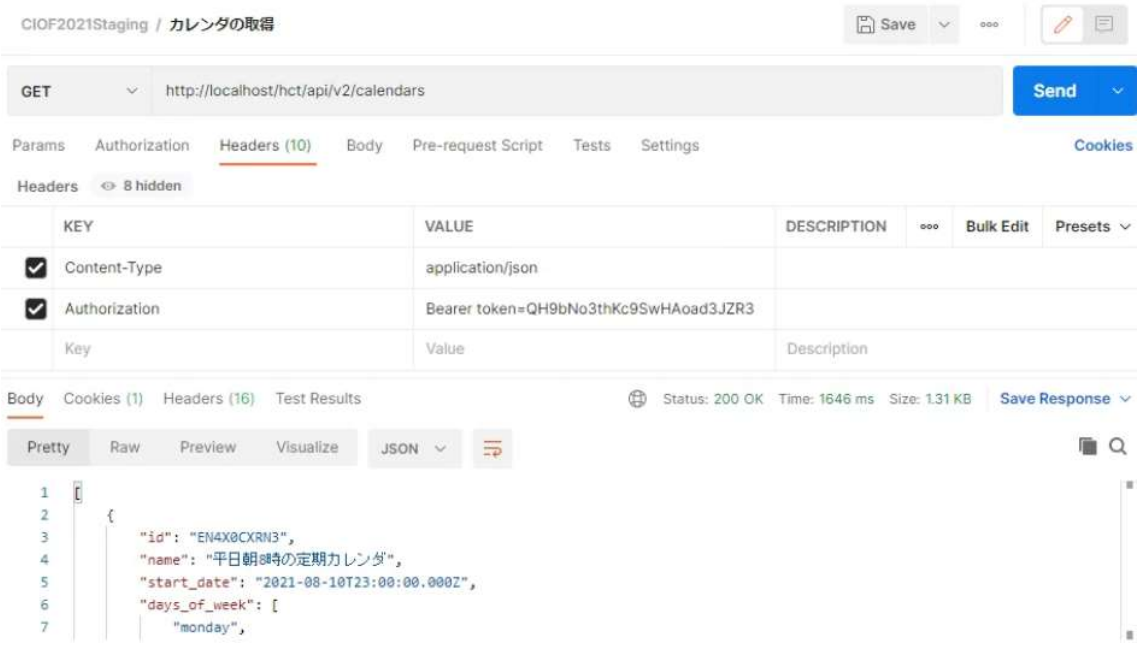


内部 ID は、連携マネージャからも操作することができます。そのため、連携マネージャにて id と local_id を紐づけておいて、コントローラ側からは、それら情報を読み取るという運用も認められています。

6.2.4　カレンダの取得

CIOF システムでは、時刻に起因するイベントをカレンダとして取り扱います。カレンダは、予め連携マネージャを経由して登録されています。コントローラは、起動時

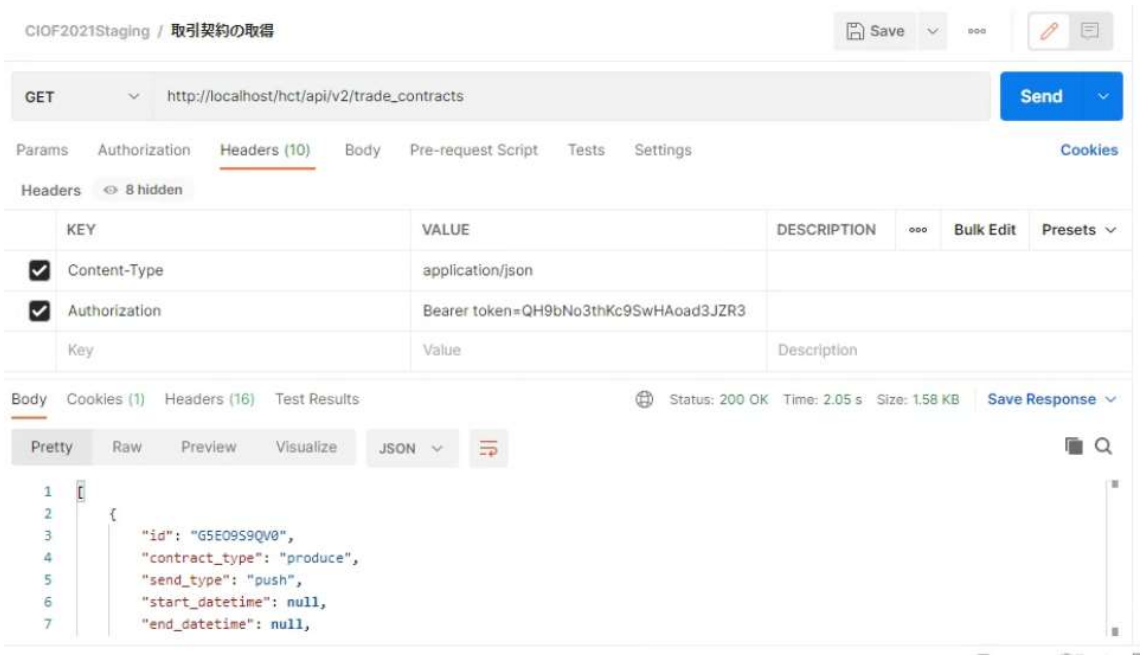
に登録されているカレンダを読み込み、それらカレンダに従った動作が求められます。以下に Postman によるカレンダ取得の実行例を示します。



カレンダの扱い方に関する詳細は、6.4 カレンダの取得と扱い方に記載されています。

6.2.5　取引契約の取得

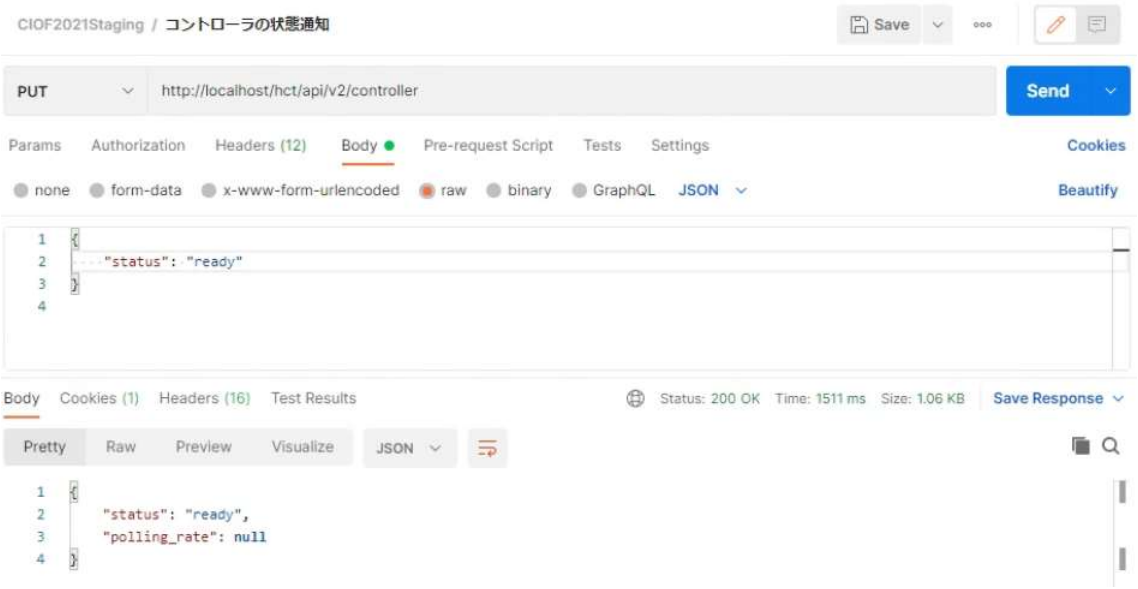
CIOF システムでは、データ送受信の前提として、取引契約が存在します。取引契約は、連携マネージャを通してデータ送受信に関する各種の取り決めを事業者間（厳密には IVI との間も含む）で規定するものです。コントローラは、起動時に登録されている取引中の取引契約を読み込み、それら契約に従ったデータ送受信が求められます。以下に Postman による取引契約取得の実行例を示します。



取引契約の扱い方に関する詳細は、6.5 取引契約の取得と扱い方に記載してあります。

6.2.6　コントローラの正常動作開始通知

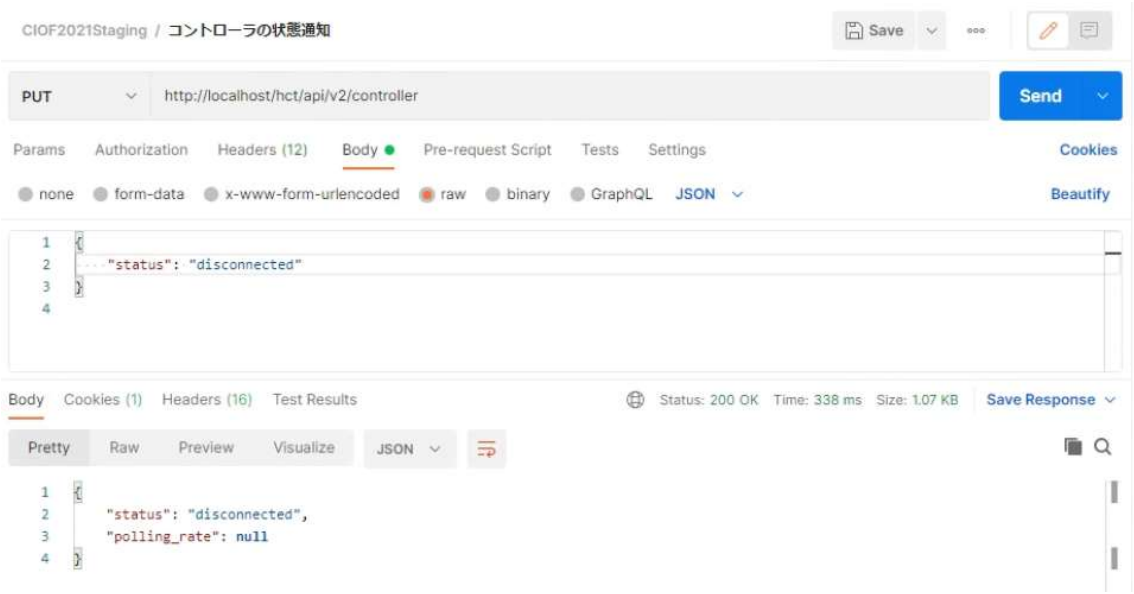
ここまでの手順を実行し、すべてが正常に完了した場合は、コントローラ状態通知 API にて、コントローラの status を ready にします。以下に Postman による実行例を示します。



6.3 コントローラ終了時処理

コントローラの終了時においても、コントローラの状態を CIOF システムへ通知する必要があります。ここでは、API を用いたコントローラ状態の通知の手順について説明します。

コントローラが電源 OFF する場合は、その直前に、コントローラ状態通知 API にて、status を disconnected とします。polling_rate の設定についての規定はありません。以下に Postman による実行例を示します。

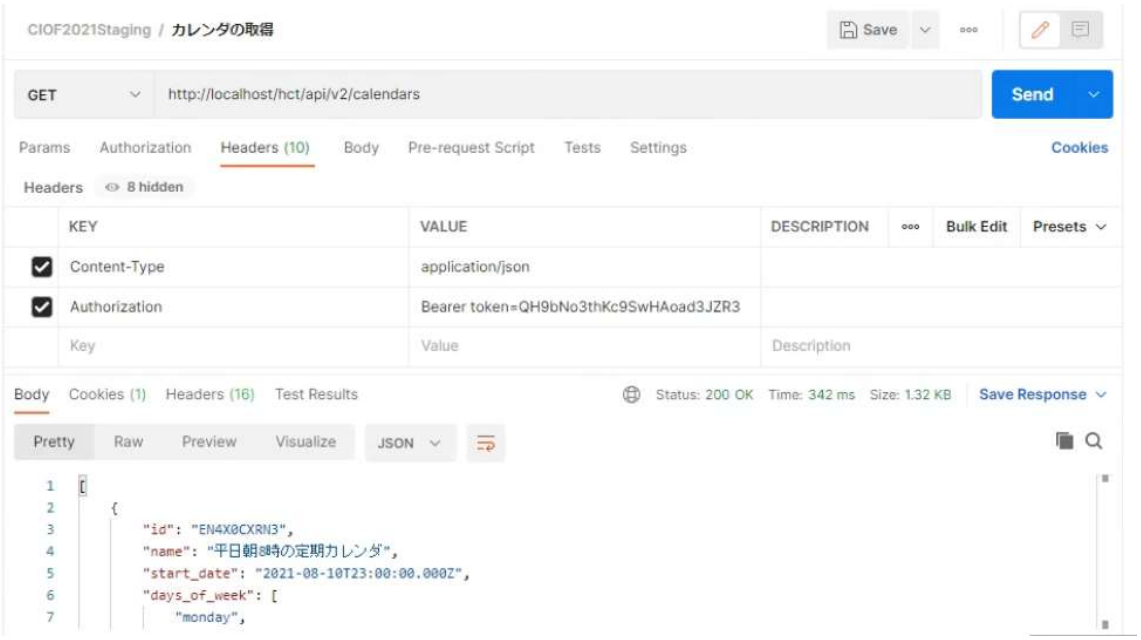


6.4　カレンダの取得と扱い方

CIOF システムでは、時刻に起因するイベントをカレンダとして取り扱います。カレンダは、予め連携マネージャを経由して、登録されています。コントローラは、連携ターミナルから、カレンダ情報を取得し、その情報を基に自分が管理するプロセス実装を起動します。カレンダの取得と、その情報をサービス実装および起動するプロセス実装と紐づけるためには、2 つの API を用います。

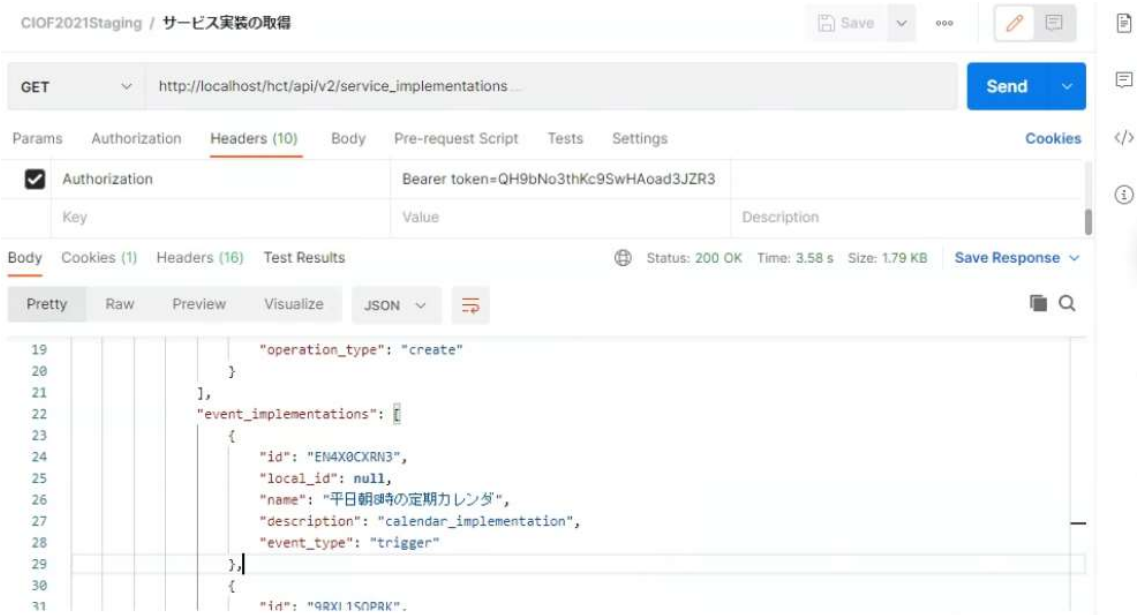
6.4.1　カレンダの取得

初めにカレンダの取得 API を用いてカレンダ情報を取得します。以下に Postman による実行例を示します。



6.4.2　カレンダとイベント実装の紐づけ

次にサービス実装の取得 API にてトリガとなるイベント実装として登録されたカレンダを取得します。以下に Postman による実行例を示します。



カレンダは、連携マネージャを用いて設定したサービス実装配下の"event_implementations"に登録されています。通常のイベント実装と異なり、"description": "calendar_implementation"となっていることが分かります。

calendar_implementation となっているイベント実装における"id"をキーにして、先に取得したカレンダの"id"と一致していることを確認し、情報を紐づけます。この場合は、EN4X0CXRN3 が該当します。サービス実装の取得 API で取得できるカレンダは、イベントプロファイルが存在し、トリガ実装が存在する「トリガとなるイベント実装」として見ることのできるカレンダのみであることに注意してください。

6.4.3　カレンダにおける各項目の設定方法

カレンダにおける各項目の設定には、大きな自由度が与えられていましたが、各事業者間で同じ意味にもかかわらず、設定方法が異なる可能性もあるため、連携マネージャによって入力制限を加えました。また、以下に各項目の意味合いと制限事項について記載します。

プロパティ名	型	概要	制約など
name	String	カレンダの名称	
decsription	String?	カレンダの説明	
startDate	DateTime	基準日時	
recurrentTimeZone	IanaTimeZoneIds	基準日時のタイムゾーン名	IANA — Time Zone Database Noda Time | Time zones
intervalType	IntervalTypeType?	間隔タイプ `minute hour day week month year` がそれぞれ「分ごと」「時間ごと」「日ごと」「週間ごと」「ヶ月ごと」「年ごと」に対応する。 繰り返しを使用しない場合は `null`	
interval	Integer?	間隔の数値	• `intervalType != null` の場合は `interval != null` でなければならない • `intervalType == null` の場合は `interval` を無視して良い • `interval != null` の場合は `1 <= interval <= 10000` でなければならない

daysOfWeek	DayOfWeekType[]?	曜日
		間隔タイプが「分ごと」「時間ごと」「週ごと」「ヶ月ごと」の場合のみ使用されるプロパティ。
		間隔タイプが「ヶ月ごと」の場合、曜日の週プロパティと合わせて使用する事で「第 2 月曜日」の様な表現を行う。
		`monday tuesday wednesday thursday friday saturday sunday` がそれぞれ「月」「火」「水」「木」「金」「土」「日」の曜日に対応する。
			• `intervalType == null || intervalType == 'day' || intervalType == 'year'` の場合 `daysOfWeek` を無視してよい
			• `intervalType == 'minute' || intervalType == 'hour' || intervalType == 'week'` の場合
			  ◦ `daysOfWeek != null && daysOfWeek.length > 0` でなければならない
			  ◦ `daysOfWeek` に基準日時の曜日が含まれていなければならない
			• `intervalType == 'month'` の場合
			  ◦ `weeksOfMonth != null && weeksOfMonth.length > 0` の場合
			  ▪ `daysOfWeek != null && daysOfWeek.length > 0` でなければならない
			  ▪ `daysOfWeek` に基準日時の曜日が含まれていなければならない
			  ◦ `weeksOfMonth == null || weeksOfMonth.length == 0` の場合
			  ▪ `daysOfWeek == null || daysOfWeek.length == 0` でなければならない

weeksOfMonth	Integer[]?	曜日の週
		間隔タイプが「ヶ月ごと」の場合に、曜日プロパティと合わせて使用する事で「第 2 月曜日」の様な表現を行う。
		例えば、2021年9月13日は 2021年9月の2回目の月曜日だが、その何度目の曜日か (この場合は2) を指している
		`weeksOfMonth == 5` は、第 5 曜日もしくは、その月の最終曜日を指す
			• `intervalType == 'month'` の場合
			  ◦ `daysOfWeek != null && daysOfWeek.length > 0` の場合
			  ▪ `weeksOfMonth != null && weeksOfMonth.length == 1` でなければならない
			  ▪ `weeksOfMonth` に基準日時の曜日の週が含まれていなければならない
			  ▪ `1 <= weeksOfMonth[0] <= 5` でなければならない
			  ◦ `daysOfWeek == null || daysOfWeek.length == 0` の場合
			  ▪ `weeksOfMonth == null || weeksOfMonth.length == 0` でなければならない
			• `intervalType != 'month'` の場合
			  ◦ `daysOfWeek` を無視してよい

endDate	DateTime?	終了日時
		終了日時以降 (この日時を含む) はスケジュールがキャンセルされる。
		繰り返し回数と排他的で、同時に設定された場合の動作は不定とする。
			• 終了条件として終了日時を使用する場合は以下を全て満たす必要がある
			  ◦ `endDate != null` でなければならない
			  ◦ `startDate < endDate` でなければならない
			  ◦ `numberOfOccurences == null` でなければならない

numberOfOccurences	Integer?	繰り返し回数
		指定回数を実行するとスケジュールがキャンセルされる。
		終了日時と排他的で、同時に設定された場合の動作は不定とする。
			• 終了条件として繰り返し回数を使用する場合は以下を全て満たす必要がある
			  ◦ `numberOfOccurences != null` でなければならない
			  ◦ `1 <= numberOfOccurences <= 10000` でなければならない
			  ◦ 終了日時と排他的のため `endDate === null` でなければならない

IntervalTypeType

間隔タイプの列挙型

値	説明
minute	分ごと
hour	時間ごと
day	日ごと
week	週ごと
month	ヶ月ごと
year	年ごと

DayOfWeekType

曜日の列挙型

値	説明
monday	月曜日
tuesday	火曜日
wednesday	水曜日
thursday	木曜日
friday	金曜日
saturday	土曜日
sunday	日曜日

IanaTimeZoneIds

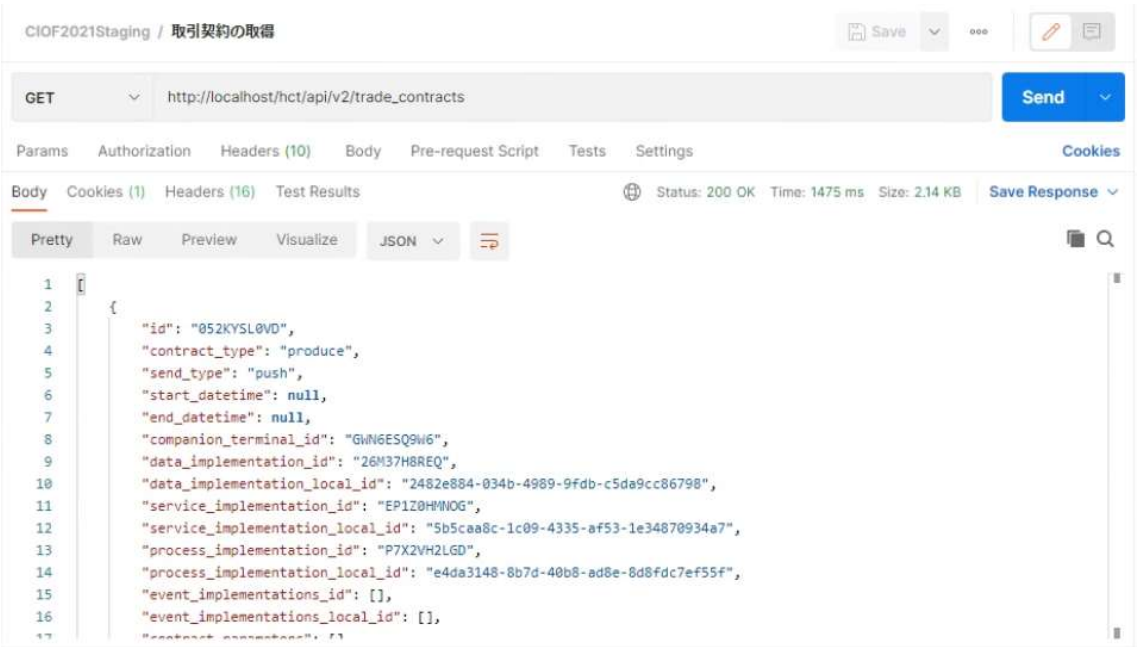
以下のコマンドで取得したタイムゾーンIDの列挙型

```
curl -s -o - 'https://nodatime.org/TimeZones?version=2021a&format=json' | jq -r '.Zones[].Id' -
```

6.5 取引契約の取得と扱い方

6.5.1 取引契約の取得

データ送受信の前提として、取引契約が存在します。取引契約は、連携マネージャを通してデータ送受信に関する各種の取り決めを事業者間（厳密には IVI との間も含む）で規定するものです。連携マネージャを通して作成された取引契約には、取引契約 ID が CIOF システムによって発行されています。コントローラが各データの送受信する際には、取引契約 ID が必要となります。コントローラから取引契約 ID を取得するためには、取引契約の取得 API を用います。これにより、現在取引中となっている取引契約の一覧を取得できます。以下に Postman による取引契約取得の実行例を示します。



6.5.2　取引契約による通信方法

取引契約を取得したコントローラは、各取引契約 ID に紐づいたデータを送受信する場合に、どのサービス実装（ソフトウェアモジュール）と通信する必要があるかを把握しておく必要があります。

例えば、次に示すような、取引契約 ID "052KYSL0VD"においては、サービス実装 ID "EP1Z0HMNOG"に該当するサービス実装がデータを利用する契約となっています。したがって、該当する取引契約と紐づいたデータを連携ターミナルから受信した場合、コントローラは、"EP1Z0HMNOG"に該当するサービス実装に受信したデータを転送する必要があります。

```
    {
     "id": "052KYSL0VD",
     "contract_type": "consume",
     "send_type": "push",
     "start_datetime": null,
     "end_datetime": null,
     "companion_terminal_id": "N5GYVS3E58",
     "data_implementation_id": "YPQ49H906W",
     "data_implementation_local_id": "265d529e-55bc-4015-8e9c-fa90e132ad8b",
     "service_implementation_id": "V7X93HMZPJ",
     "service_implementation_local_id": "04ea5d71-455d-4452-a2c1-82901f456876",
     "process_implementation_id": "9GOQWHOK7R",
     "process_implementation_local_id": "6f0f8299-5fa8-46c7-aaa9-b33e17d09af4",
     "event_implementations_id": [
         "4R4N1S4L8P",
         "9JOQ5SDMRO"
     ],
     "event_implementations_local_id": [
         "b020ceaa-b8f1-4e1b-a124-4d5c6edadb99",
         "c102f896-1ebd-4c73-ae32-460d83d7c520"
     ],
     "contract_parameters": []
  }
```


6.5.3　取引契約における契約事項(contract_parameters)の扱い

取引契約のパラメータのひとつに契約事項(contract_parameters)があります。契約事項では、マシンリーダブルな表現によって、データの取り扱い方についての取り決めを行います。CIOF システムでは、次のような取り決め方法を定めています。

契約項目	契約内容	意味

use	<イベント実装 ID>	モニタイベントが発火した場合は、サービス記録の通知 API により"利用"を報告しなければならない。
delete	required	削除リクエストがあった場合に取引データを削除しなければならない。
delete	reporting	取引データを削除した場合に、サービス記録の通知 API により"削除"を報告しなければならない。
revise, store, duplicate のいずれか１つ	prohibited	改変、保存、複製の禁止
revise, store, duplicate のいずれか１つ	reporting	改変、保存、複製した場合に、報告しなければならない。

6.5.4　取引契約における事業者の立場

取引契約においては、契約締結時およびデータ送受信時に、それぞれの事業者に対して役割が定義されています。ここでは、それぞれの役割と名称について記載します。

6.5.4.1　取引契約のデータの提供側・利用側

取引契約によって送受信される実データの流れの向きを表す概念です。コントローラは、取引データの送信 API を利用する際には、指定する取引契約 ID に記載されたデータの向きに従う必要があります。

- 提供側（producer）：生成された実データを送信する側
- 利用側（consumer）：提供側が生成した実データを受信する側

6.5.4.2　取引契約のデータプロファイルの提示側・受入側

データプロファイルがどちら側から提示されるかを表す概念です。これは、連携マネージャでの操作方法に影響を受けますが、コントローラが意識する必要はありません。

- 提示側（proposer）：データプロファイルの提示側であり、マッピングされる側
- 受入側（responder）：データプロファイルの提案に対する受入側であり、マッピングする側

6.5.4.3　取引契約の発行側・受領側

取引契約がどちら側から開始されるかを表す概念です。連携マネージャを用いて取引契約の締結を開始する際に決まる立場です。コントローラが意識する必要はありません。

- 発行側（issuer）：取引契約プロファイルを作成した側
- 受領側（recipient）：発行側によって指定された取引相手となる側

6.6　取引契約に基づいたデータ送受信

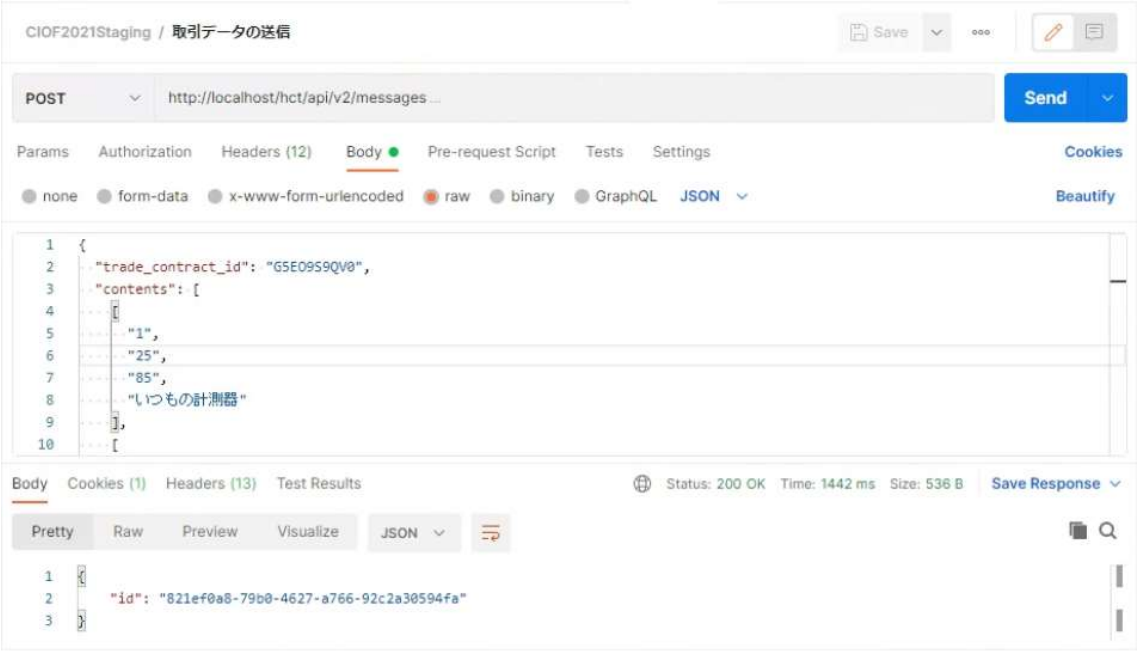
取引契約に基づいたデータ送受信の形態には、プッシュ通信、プル通信、パブサブ通信、コレクト通信の 4 種類があります。本節では、それぞれ 4 種類の形態に対して API をどのように用いれば良いかを説明します。

6.6.1　プッシュ通信

プッシュ通信は、取引契約における提供側から利用側へ一方的にデータを送信する契約形態です。ここでは、プッシュ通信をするために利用する API の流れについて説明します。前提として、コントローラは、現在取引中となっているすべての取引契約に関する情報を、取引契約の取得 API を用いて取得済みであるものとします。

初めに、取引契約におけるデータ提供側コントローラ配下のサービス実装から送信したいデータがコントローラに送られてくることになります（この点について CIOF システムは通信方法を規定していません）。

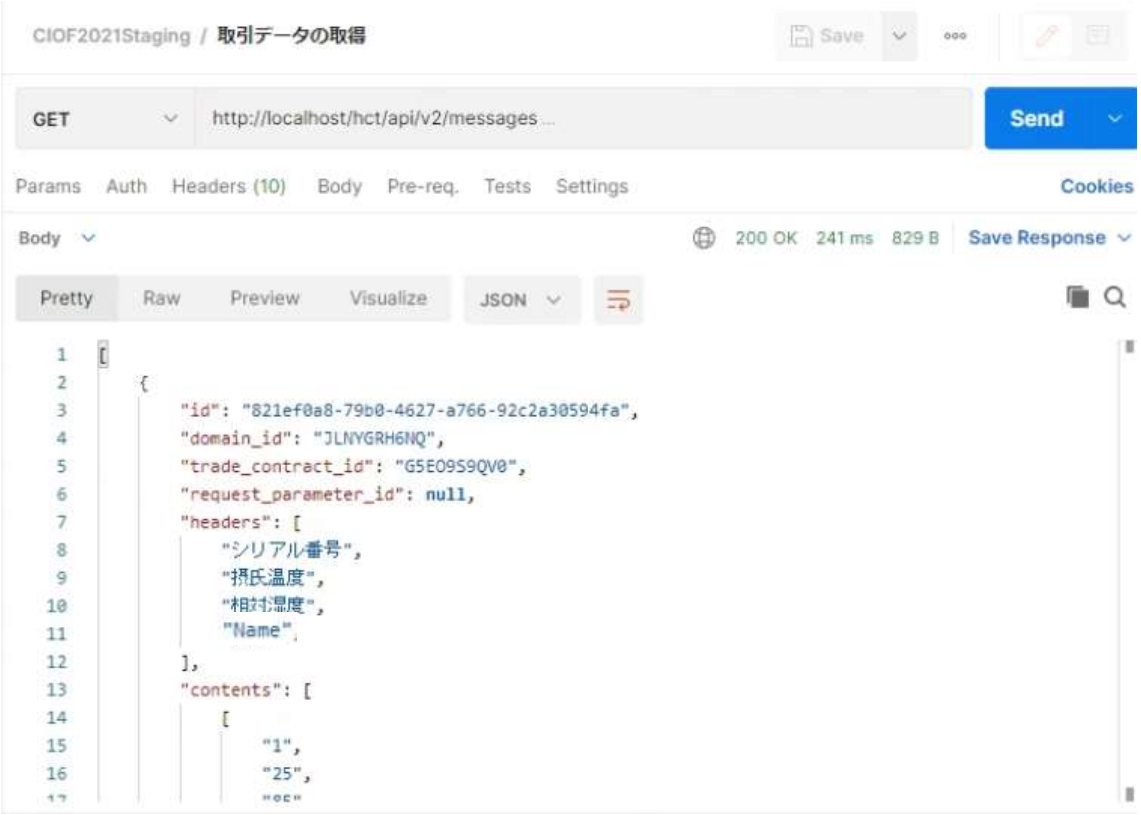
コントローラは、データ送信元のサービス実装および実際に送信するデータプロファイルの構造を確認し、内部で保持している取引契約一覧と比較することで、trade_contract_id を定めます。コントローラは取引データの送信 API を用いて、trade_contract_id と contents（サービス実装から送信されてきた取引データ）を対にして送信します。同じ内容のデータに関して、複数相手と取引契約を結んでいた場合は、trade_contract_id が複数存在することになります。その場合は、取引データの送信 API を trade_contract_id ごとに複数回呼び出します。以下に Postman による実行例を示します。



※contents については、契約に基づいたデータのバリデーションチェックが行われ、取引契約にて取り決めをしたものと項目数が一致していない場合は、エラーとなります。

送信が完了すると、取引データ ID の応答があります。応答された取引データ ID は、今回 API で送信したデータ一式に対して与えられています。例えば、送信したレコード数が 1 の場合でも 10 の場合でも 1 つの取引データ ID が発行されます。取引データ ID は、送信後の取引データが利用側でどのように操作されたかという履歴情報を確認するためのキーとなりますので、十分に気を付けて保存しておく必要があります。

続けて、**取引相手方**においてデータを受信します。データの受信はポーリング周期に従って取引データの取得 API を繰り返し呼び出しているはずです。以下に Postman による実行例を示します。



API を用いて受信される contents は、前回の受信以後、今回の API によるリクエストまでの時点で自身のコントローラ宛（コントローラ配下のサービス実装宛）に送信されているすべての取引データです。取引データの取得 API は、URL パラメータを指定することで内容を限定して受信することもできます。宛先となるサービス実装での限定、取引データに該当する取引契約での限定が可能です。複数の ID を指定することもできます。詳細は、CIOF システムコントローラ用 API リファレンスマニュアルを参考にしてください。

一度受信したメッセージは、サーバ上から削除されますので、再度受信することはできません。内容を限定したことによって受信されなかったメッセージが削除されることはありません[5]。

送受信データの対応関係は、取引データ ID 「821ef0a8-79b0-4627-a766-92c2a30594fa」によって確認できます。連携マネージャからも同様の ID をキーにデータ送受信の履歴を確認することができます。履歴の確認方法については、6.8 サービス記録の通知にて説明します。

6.6.2　プル通信

プル通信は、プッシュ通信と異なり、データリクエストを基にデータの送受信を行うことを指します。プル通信を行うことができるのは、取引契約を結ぶ際にプル通信を許可している契約である場合に限られます。プル通信の詳細説明は、6.7.1 データリクエストを参照してください。

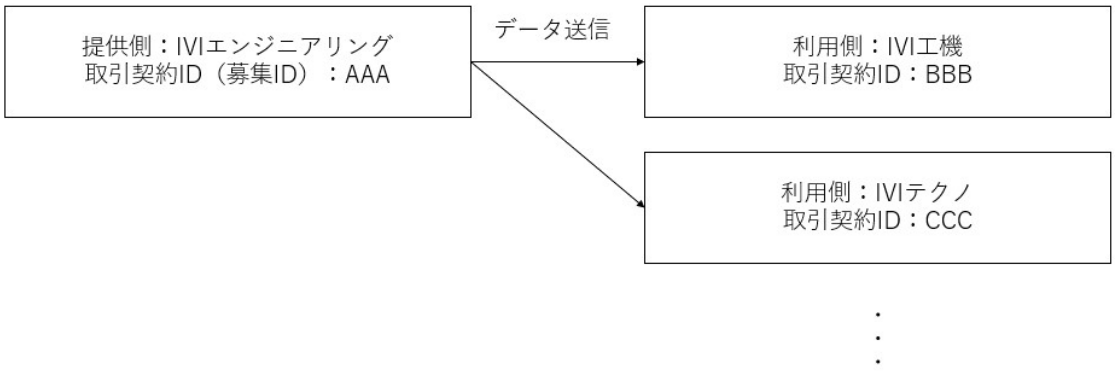
[5] 今後のアップデートにより、長期間にわたり受信されなかったメッセージが削除される可能性はあります。これは、サーバ上に蓄積されるデータが無限に増えることを防ぐための措置です。

6.6.3　パブサブ通信

パブサブ通信では、取引契約における提供側（パブリッシュ側）が 1 度の API 呼び出しにより、複数の利用側（サブスクライブ側）へ同一内容の取引データを送信することができます。提供側のデータ送信方法および、利用側の受信方法においては、プッシュ通信と同様の手順で同様の API を用いるため、コントローラ側で意識する必要はありません。

ただし、提供側と利用側で用いる取引契約 ID が異なります。これは、提供側と利用側が 1 対 N の関係で紐づいており、それぞれの事業者ごとに取引契約 ID が設定されるためです。

以下に概念図を示します。IVI エンジニアリングのコントローラがデータ送信を行う場合は、AAA の取引契約 ID（特に、募集 ID と呼びます）を指定します。



具体例として、連携マネージャの画面を示します。2 者へ送信する場合は、自分の ID を含めて取引契約 ID が 3 つ存在することが分かります。

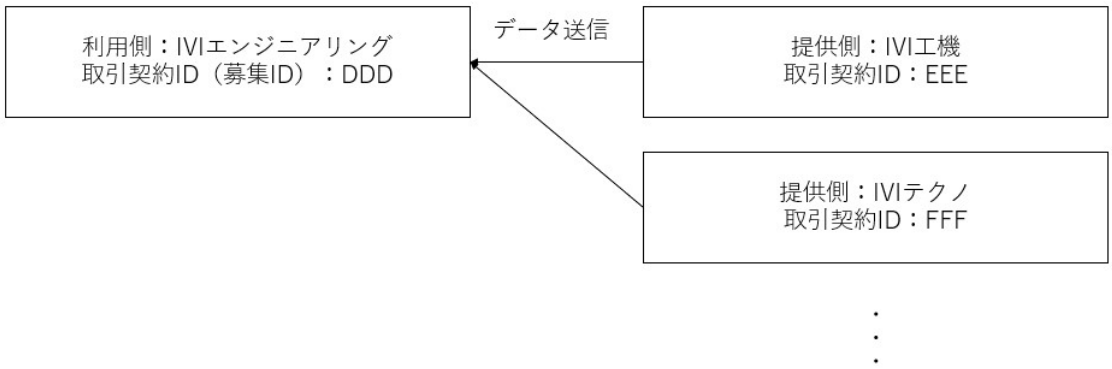
なお、パブサブ通信では、プル通信のようにリクエストパラメータを用いることはできません。

6.6.4　コレクト通信

コレクト通信では、利用側（コレクト側）が 1 度に複数の提供側から同一取引条件での取引データを受信することができます。提供側のデータ送信方法および、利用側の受信方法においては、プッシュ通信と同様の手順で同様の API を用いるため、コントローラ側で意識する必要はありません。

ただし、提供側と利用側で用いる取引契約 ID は異なります。これは、提供側と利用側が N 対 1 の関係で紐づいており、それぞれの事業者ごとに取引契約 ID が設定されるためです。

以下に概念図を示します。IVI 工機および IVI テクノのコントローラがデータ送信を行う場合は、それぞれ EEE および FFF の取引契約 ID を指定します。IVI エンジニアリングのコントローラがデータ受信を行う場合は、DDD の取引契約 ID（募集 ID）が付されます。



なお、コレクト通信では、プル通信のようにリクエストパラメータを用いることはできません。

6.6.5　通信区分によりできることのまとめ

ここまでで説明した通信区分によって、提供側および利用側からできることは、次の通りです。

	提供側			利用側		
	データ送信	データリクエスト	削除リクエスト	データ送信	データリクエスト	削除リクエスト
プッシュ	OK	NG	OK	NG	NG	NG
プル	OK	NG	OK	NG	OK	NG
パブサブ	OK	NG	NG	NG	NG	NG

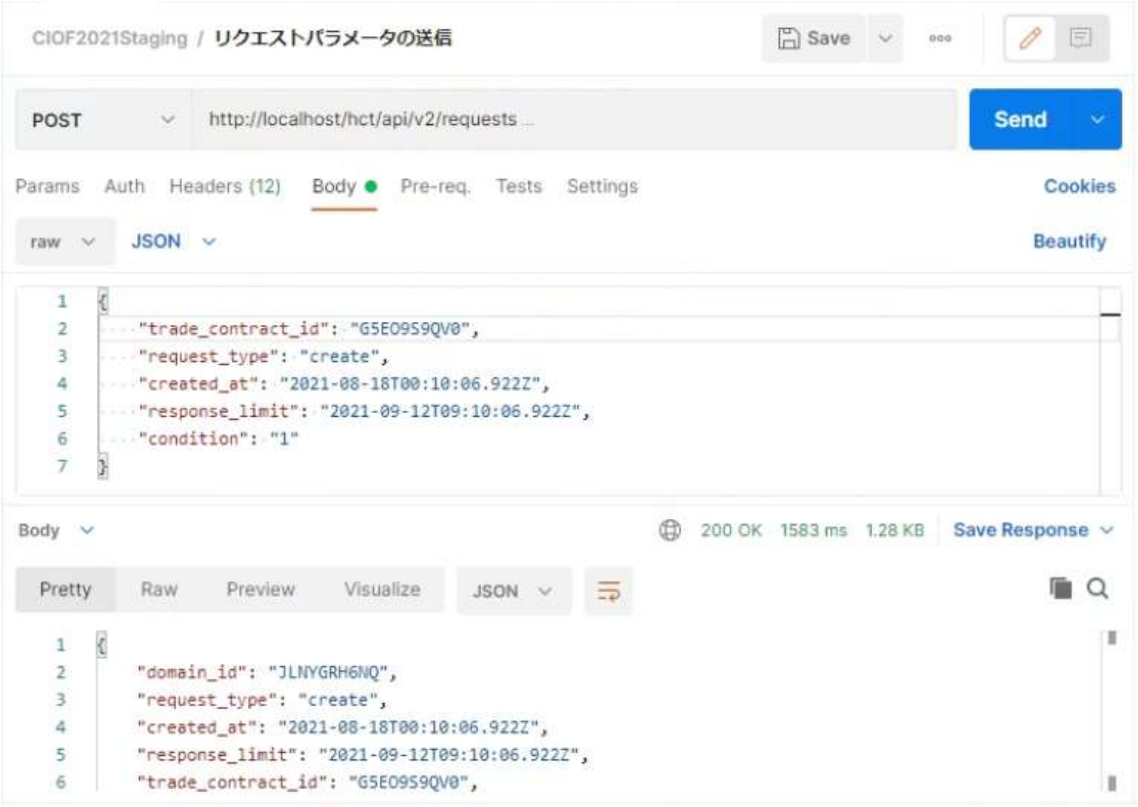
コレクト	OK	NG	NG	NG	NG	NG

6.7 リクエストパラメータの送受信

6.7.1 データリクエスト

プル通信が認められている契約においては、取引契約における利用側が提供側に対してデータリクエストを送ることができます。データリクエストとは、取引契約における利用側が提供側に対してデータ送信を促すための要求です。データリクエストでは、契約（データプロファイル）通りにすべてのデータについての送信要求をすることに加えて、予め取引契約の発行者が決めた条件で限定したデータについての送信要求をすることもできます。ここでは、API を用いたデータリクエストの方法について説明します。

初めに、取引契約における利用側でリクエストパラメータを送信します。リクエストパラメータの送信には、リクエストパラメータの送信 API を用います。以下に Postman による実行例を示します。

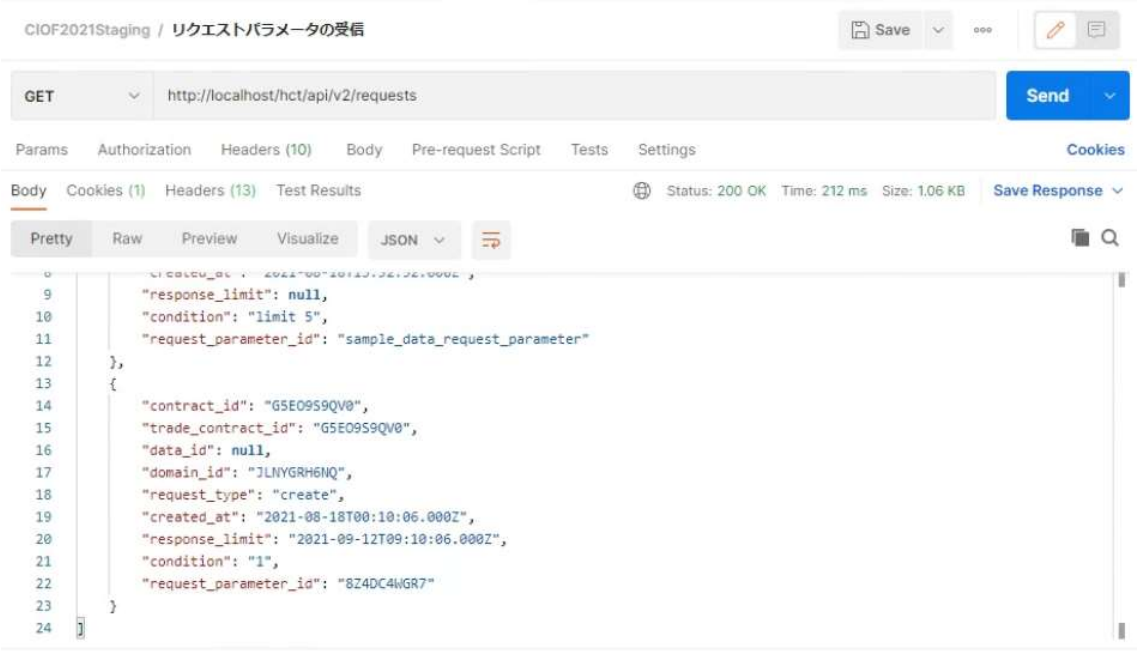


本例において、"condition"は、"1"というパラメータで依頼していますが、本パラメータは、予め取引契約の発行者が決めた条件にしたがって記述します。発行者は、連携マネージャのデータ実装画面内「PULL リクエストパラメータの設定方法」に、取引契約にて扱いたいデータプロファイルの元となるデータ実装に対して、条件を自由記述することで設定します。以下に、連携マネージャの画面を示します。

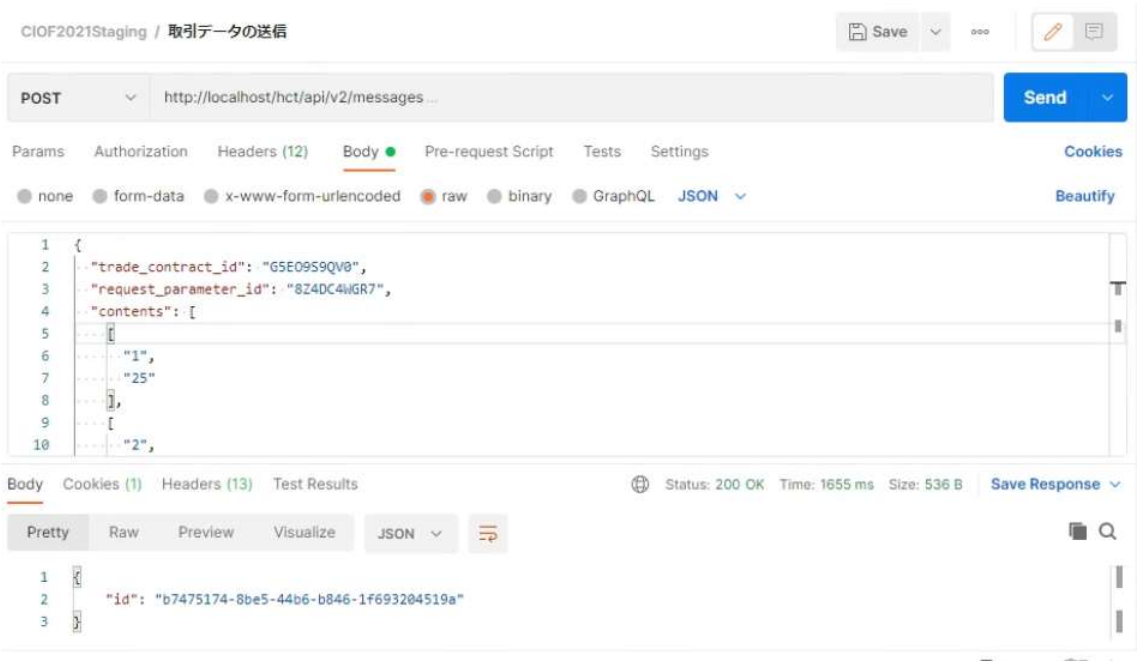
取引契約を扱う際に赤枠内に先の条件が転記されます。

リクエストパラメータの送信 API のレスポンス内に request_parameter_id が発行されているので、記録しておきます。以下に Postman による実行例を示します。

次に、**取引契約における提供側**でリクエストパラメータを受信します。リクエストパラメータの受信には、リクエストパラメータの取得 API を用います。以下に Postman による実行例を示します。

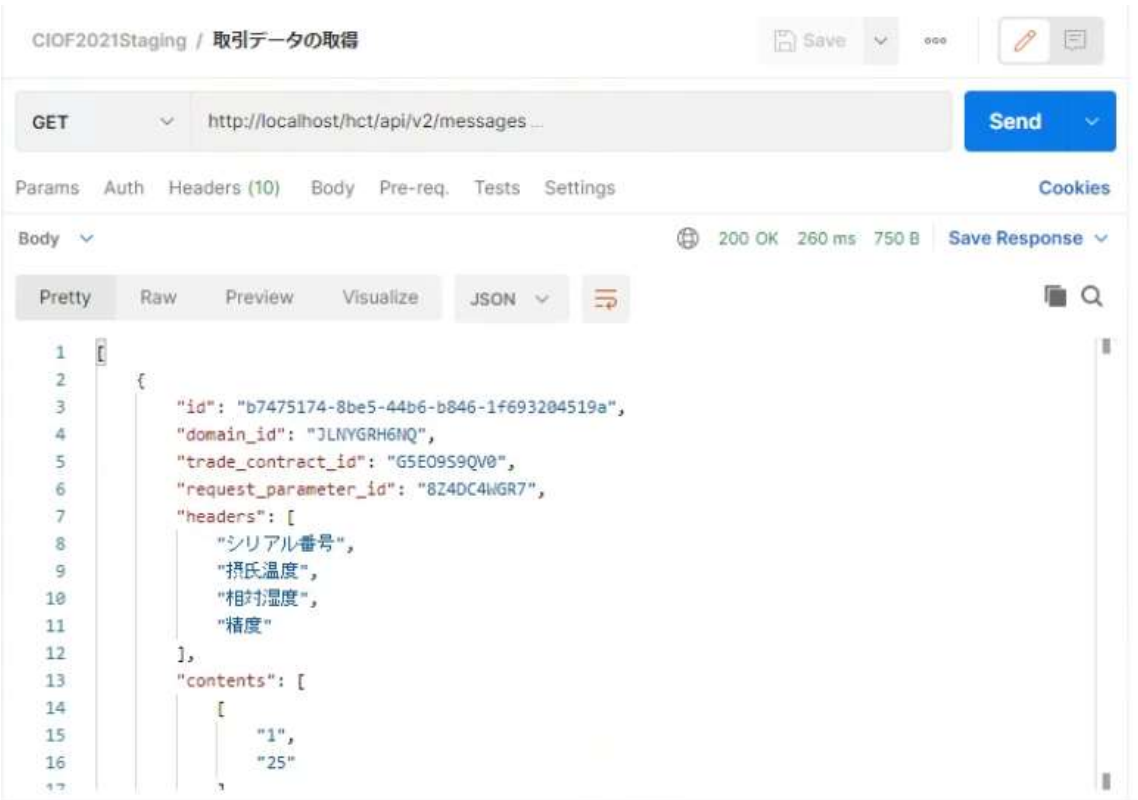


リクエストパラメータを取得後は、リクエストパラメータの内容を確認します。この場合は、"condition"が"1"というパラメータで要求がきています。data_id が null なのは、これがデータ要求であるためです。データ削除リクエストの場合は、data_id に削除要求対象となる取引データ ID が指定されます。受信したリクエストパラメータ ID"8Z4DC4WGR7"を request_parameter_id に加えて、送信します。以下に Postman による実行例を示します。



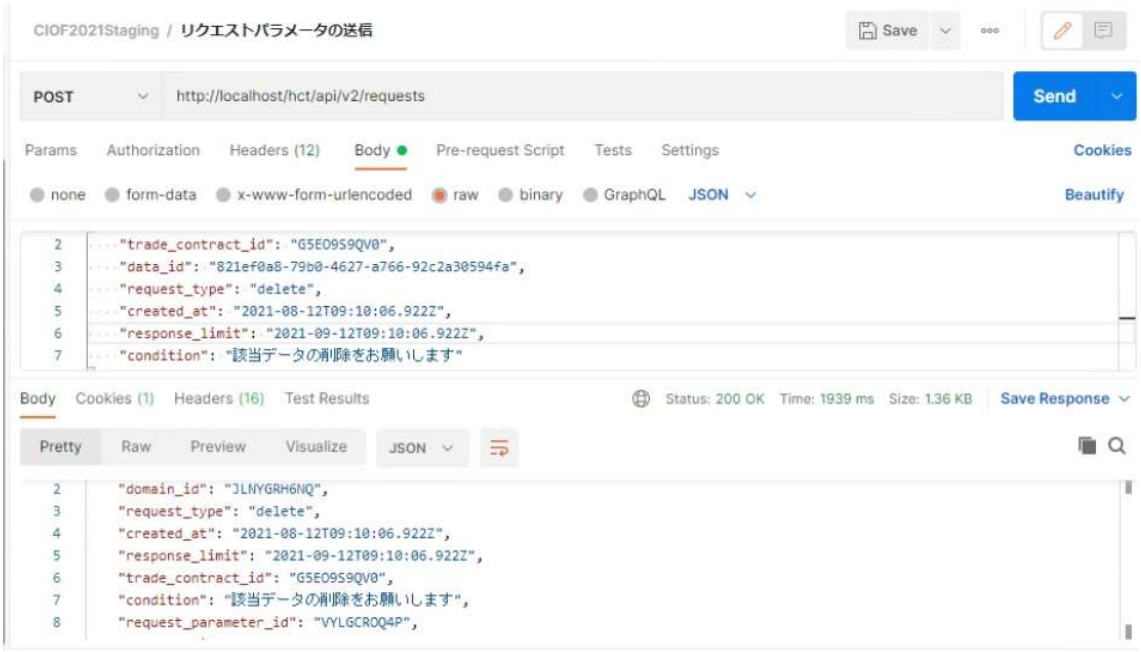
取引契約における利用側でメッセージを取得します。予め記録しておいた

request_parameter_id と照らし合わせて、どのリクエストへの応答かを確認します。

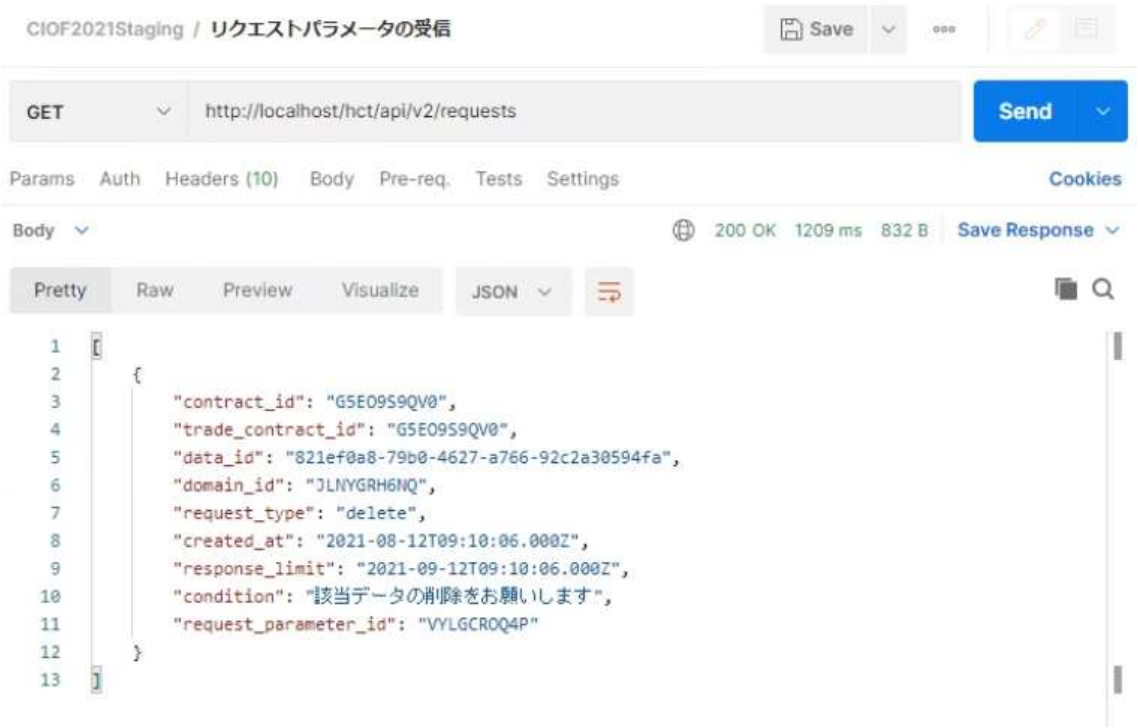


6.7.2　削除リクエスト

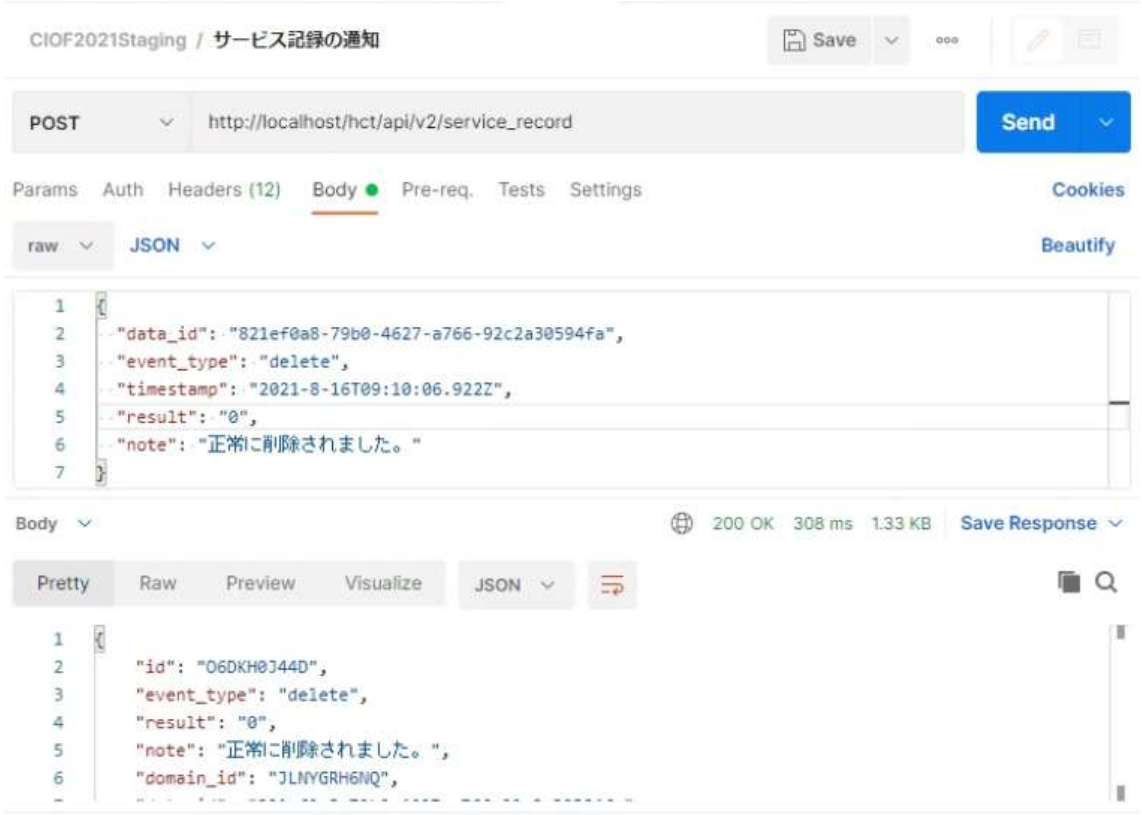
CIOF システムで通信したデータに対して、取引契約における提供側が利用側に対して削除リクエストを送ることができます。削除リクエストの送信には、リクエストパラメータの送信 API を用います。"request_type": "delete"を指定し、削除対象のデータを data_id に指定します。以下に Postman による実行例を示します。



次に、**取引契約における利用側**で削除リクエストを受信します。削除リクエストの受信には、リクエストパラメータの取得 API を用います。以下に Postman による実行例を示します。



リクエストパラメータを取得後は、削除リクエストの対象データ("data_id")を確認して、該当データを削除します。該当データを削除後は、その内容をサービス記録の通知を用いて報告します。以下に Postman による実行例を示します。



取引契約における提供側において、削除依頼した取引データが本当に削除されたこと

を連携マネージャの取引実績管理画面より確認します。

なお、削除リクエストは、連携マネージャからも送ることができます。前述の画面（取引実績管理→取引契約一覧から該当契約の目のマーク→該当取引データ ID の目のマーク）上部から、「データ削除リクエスト」ボタンを押下します。次に連携マネージャの画面を示します。

6.8 サービス記録の通知と確認方法

6.8.1 サービス記録の通知

CIOF システムでは、送受信された取引データがどのように操作されたのかを追跡することができます。そのために、コントローラは自身の配下にあるサービス実装がどのようにデータを操作したのかを連携ターミナルを通して CIOF システムに報告する必要があります。これをサービス記録の通知と呼びます。

あらかじめ連携マネージャを用いて締結された取引契約において、取引データの操作記録が必要な場合、サービス記録の通知 API を用いてサービス記録を通知します。サービス記録の通知内容は次の通りの区分を持ちます。生成（create）、送信（send）、受信（receive）、転送（forward）、読出（read）、保管（store）、複製（duplicate）、改変（revise）、削除（delete）、利用（use）。ただし、生成（create）、送信（send）、転送（forward）は連携サーバが設定しますのでコントローラから API で設定することができません。

生成（create）は、取引データ送信 API によって連携ターミナルへ送信された取引データが連携サーバに到達し、取引データ ID が割り当てられたタイミング、送信（send）は、生成（create）後に連携サーバから他の連携サーバへ送信されるタイミング、受信（receive）は、連携サーバから相手方の連携ターミナルへ取引データが到達したタイミング、転送（forward）は、相手方の連携ターミナルから相手方のコントローラへ取引データが到達したタイミングでそれぞれ記録されます。

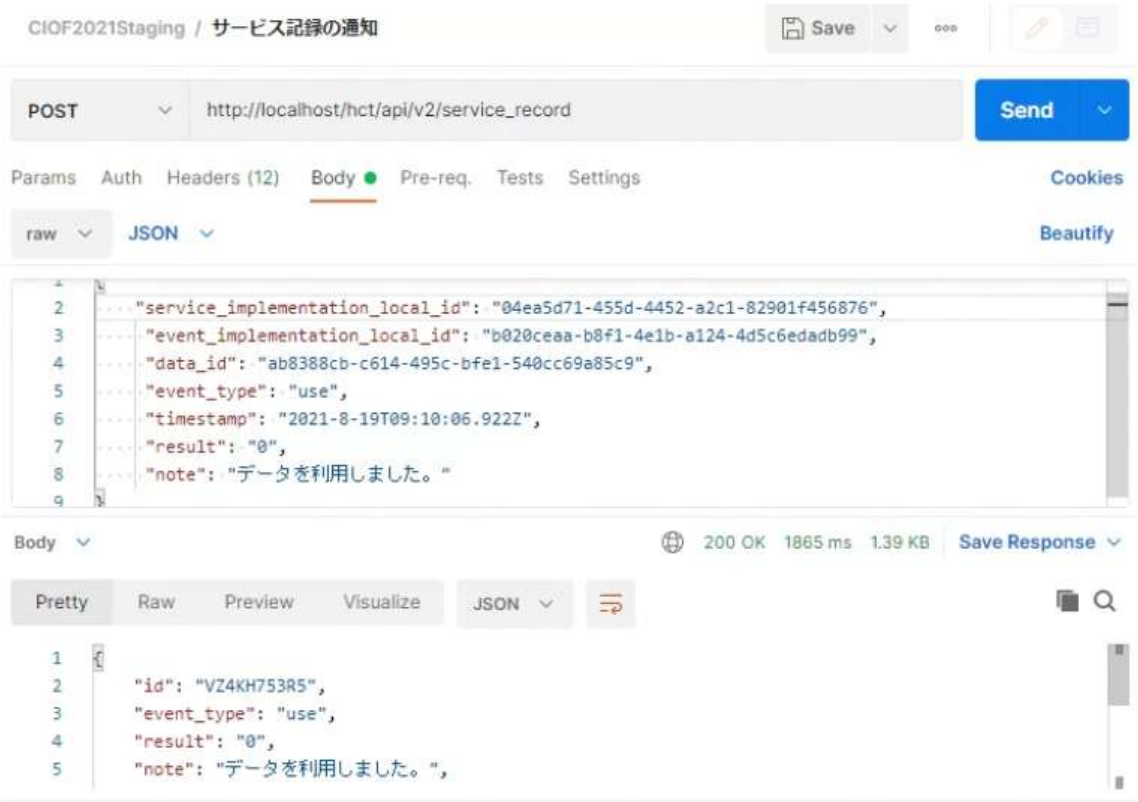
読出（read）は、コントローラから、その配下のサービス実装に該当する取引データが到達したタイミング、保管（store）は、サービス実装が該当する取引データを永続化したタイミング、複製（duplicate）は、サービス実装が該当する取引データを複製したタイミング、改変（revise）は、サービス実装が該当する取引データを改変したタ

イミング、削除（delete）は、サービス実装が該当する取引データを削除（複製している場合は、複製したすべてのデータを削除）したタイミング、利用（use）は、サービス実装が該当する取引データを利用したタイミングで通知します。

区分名	記録者	記録タイミング
生成（create）	連携サーバ	取引データ送信 API によって連携ターミナルへ送信された取引データが連携サーバに到達し、取引データ ID が割り当てられたタイミング
送信（send）	連携サーバ	生成（create）後に連携サーバから他の連携サーバへ送信されるタイミング
受信（receive）	連携サーバ	連携サーバから相手方の連携ターミナルへ取引データが到達したタイミング
転送（forward）	連携サーバ	相手方の連携ターミナルから相手方のコントローラへ取引データが到達したタイミング
読出（read）	コントローラ	コントローラから、その配下のサービス実装に該当する取引データが到達したタイミング
保管（store）	コントローラ	サービス実装が該当する取引データを永続化したタイミング

複製（duplicate）	コントローラ	サービス実装が該当する取引データを複製したタイミング
改変（revise）	コントローラ	サービス実装が該当する取引データを改変したタイミング
削除（delete）	コントローラ	サービス実装が該当する取引データを削除（複製している場合は、複製したすべてのデータを削除）したタイミング
利用（use）	コントローラ	サービス実装が該当する取引データを利用したタイミング

以下に Postman による実行例を示します。



区分名が利用（use）の場合、サービス記録の通知 API を用いる際に、イベント実装を通知する必要があります。イベント実装の通知は、ID もしくは内部 ID にて行います。内部 ID で通知する場合には、通知するイベント実装を一意に定めるために、サービス実装 ID もしくはサービス実装内部 ID やプロセス実装 ID もしくはプロセス実装内部 ID を指定する必要があります。指定パターンは次の通りです。

ID 名称	指定方法 1	指定方法 2	指定方法 3	指定方法 4
service_implementation_id			○	
service_implementation_local_id				○
process_implementation_id		○		
process_implementation_local_id			○	○
event_implementation_id	○			
event_implementation_local_id		○	○	○

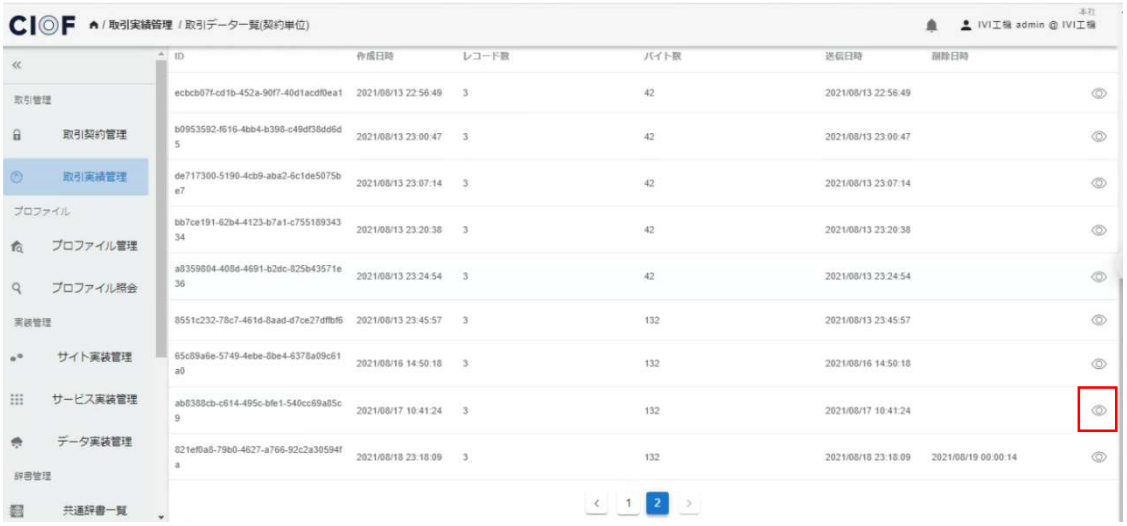
区分名が利用（use）以外の通知区分においては、イベント実装 ID の通知は不要です。

6.8.2　サービス記録の確認方法

サービス記録の確認は、連携マネージャから行います。連携マネージャのホーム画面から「取引実績管理」を選択します。

ここでは、契約毎、データ実装毎に取引データを送受信・各種操作した履歴を確認することができます。確認したい取引行における目の形をしたボタンを押下します。

続けて、「取引データ ID：ab8388cb-c614-495c-bfe1-540cc69a85c9」に対する赤枠内の目の形をしたボタンを押下します。



該当する取引データに関するデータ操作記録を確認できました。データ操作記録とは、データを生成（create）、送信（send）、転送（forward）、読出（read）、保管（store）、複製（duplicate）、改変（revise）、削除（delete）した場合の記録を指します。ここでは、連携サーバが記録した内容を確認することができています。

「サービス実行記録」タブを選択すると、先ほど API を用いて登録したサービスの実行記録が確認できました。サービスの実行記録とは、データが相手方のサービス実装にて利用(use)された場合の記録を指します。

6.9 サンプルレコードの送信

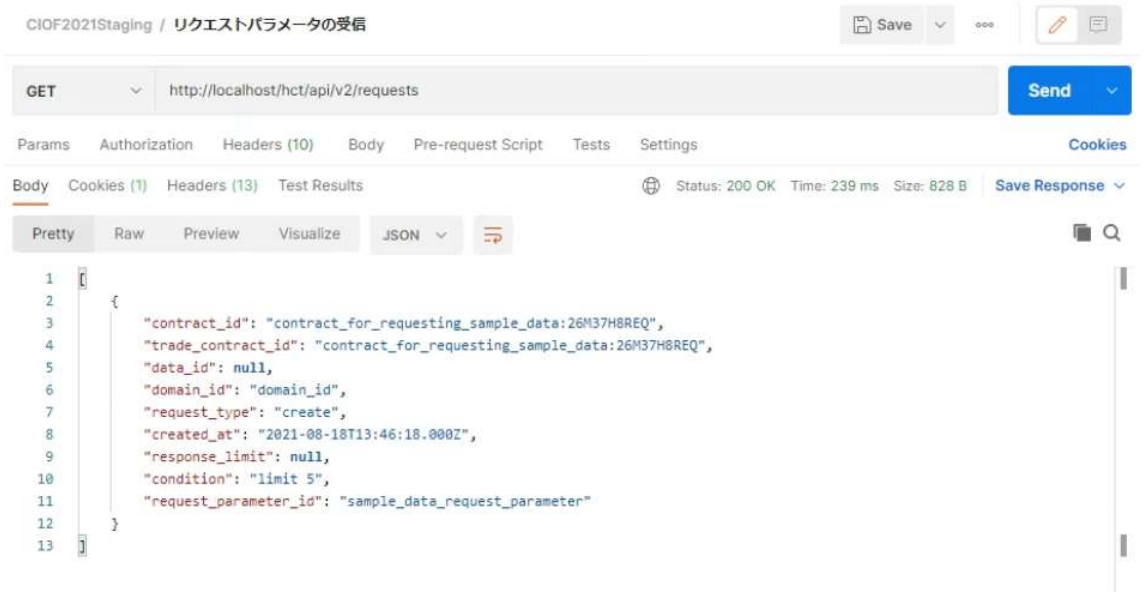
登録した各データ実装には、サンプルレコードを登録することができます。サンプルレコードとは、取引相手に対して該当のデータ実装が実際にはどのようなデータ列を

持つのかを示すために準備された一連のデータのことを指します。コントローラは、あるデータ実装を持つサービス実装に対して、サンプルレコードの登録依頼が行われた場合、その依頼を該当するサービス実装に伝え、該当するデータ実装を連携ターミナルへ提供する必要があります。ここでは、サンプルレコードの送信までの手順を示します。

初めに連携マネージャから、データ実装管理→データ実装を表示し、右上の「サンプルレコード」ボタンを押下します。

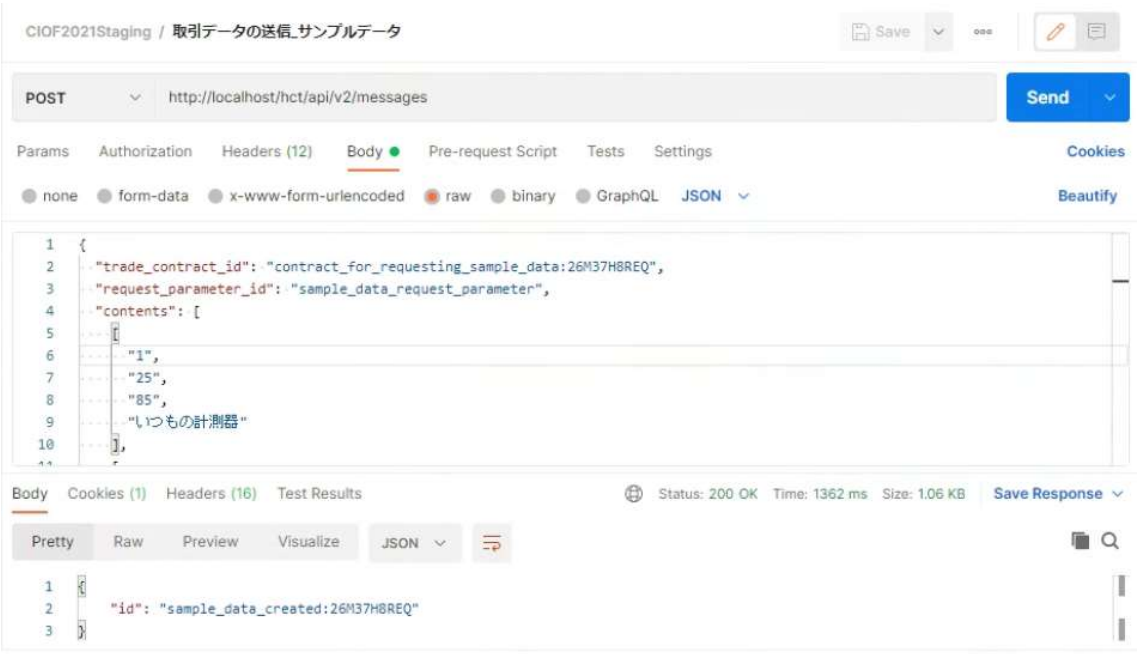
リクエストボタンを押下します。

これにより、該当するデータ実装が紐づくコントローラに、以下のようなリクエストパラメータが送信されます。



※ contract_id 内の contract_for_requesting_sample_data は固定文字列で、そのあとの ID は該当するデータ実装 ID になっていますので、コントローラからの識別用に利用できます。condition 内の limit 5 は、CIOF システムが与える固定値です。実際に送信データを 5 件以内に制限するものではありませんが、データ数は概ね 5 件前後を想定しています。

リクエストパラメータを受けたコントローラは、該当データを送信します。



該当データが送信されると、ブラウザの更新に伴い、サンプルレコード画面に送信されたデータが表示されます。

6.10 定常動作時の処理例

コントローラ用 API を用いた定常動作の一例を、次のように示します。

連携ターミナルへ設定したポーリング周期で以下の処理を行う。

1. 取引データの取得 API により、連携ターミナルから取引データを受信

   (ア)受信した取引データを宛先のサービス実装へ転送

2. リクエストパラメータの受信 API により、連携ターミナルからリクエストパラメータを受信

(ア)受信したデータリクエストを該当するデータ実装を生成するサービス実装へ転送

(イ)受信した削除リクエストを該当するデータ実装を管理するサービス実装へ転送

1 日 1 回程度、以下の処理を行う。

1. サービス実装の取得 API により、連携ターミナルからサービス実装等を取得する

(ア)取得した各種実装と現実に存在するソフトウェアモジュールとの間に差異がある場合は、サービス実装の状態通知 API により、存在しないサービス実装の状態を exception とする

2. データ実装の取得 API により、連携ターミナルからデータ実装を取得する

(ア)取得したデータ実装と現実に存在するデータとの間に差異がある場合は、データ実装の状態通知 API により、存在しないデータ実装の状態を exception とする

3. 取引契約の取得 API により、連携ターミナルから取引中の取引契約一覧を受信

4. カレンダの取得 API により、連携ターミナルからカレンダ一覧を取得

配下のサービス実装からの要求により、以下の対応を行う。

1. 取引データの送信 API により、連携ターミナルへ取引データを送信する

2. リクエストパラメータの送信 API により、連携ターミナルへリクエストパラメータを送信する

3. サービス実装の状態通知 API により、連携ターミナルへサービス実装の状態変化を通知する

4. データ実装の状態通知 API により、連携ターミナルへデータ実装の状態変化を通知する

5. サービス記録の通知 API により、連携ターミナルへサービス記録を通知する

7 変更履歴

版数	作成日	作成者	改版内容
2.1	2021.8.19	IVI 坂	2021 年度向けプレ公開版
2.11	2021.9.16	IVI 坂	正式版
2.12	2022.1.27	IVI 坂	誤植修正、機能追加に対応 カレンダ実装という名称を削除し、カレンダに統一