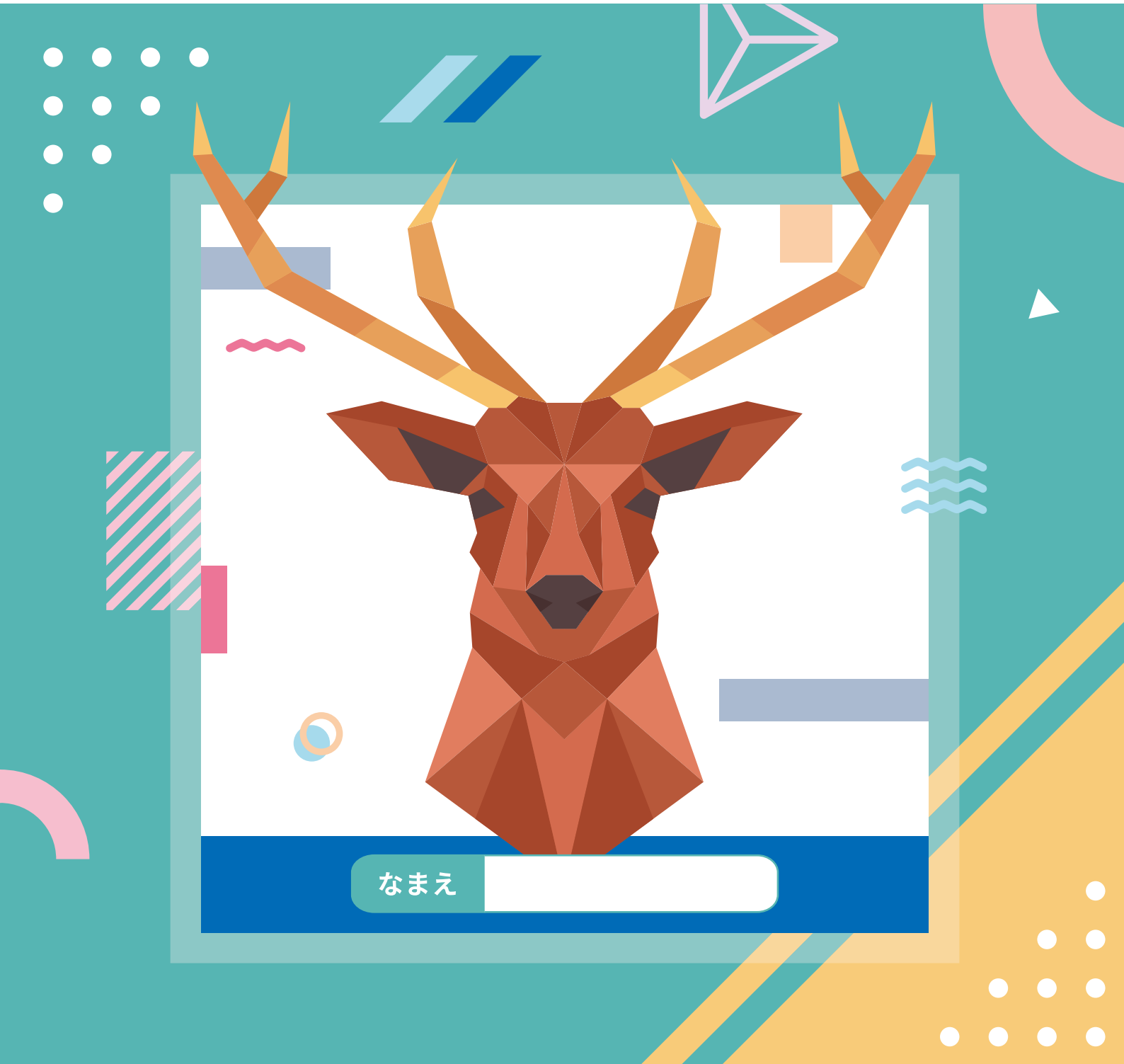
gusuku Customine

カスタマインの歩き方

Job Runner 編



はじめに（この冊子の目的）

使いこなすととても便利な gusuku Customine（以降カスタマインと表記）。しかし、実現できることが多すぎるがゆえに使っていただいているお客様より「難しい」というお声をいただくこともあります。そこで、実際に役立つカスタマイズを作成しながらカスタマインの使い方にも慣れていただくための教材を用意しました。

この冊子では、「Job Runner」を取り上げます。

「Job Runner」は、ユーザーの操作を起点とする「kintone アプリのカスタマイズ」とは異なり、毎時・毎月など決まった時間に実行するカスタマイズ「定期実行」や、kintone の Webhook の通知を起点として動くカスタマイズ「kintone アプリの Webhook」を作成できます。

また、Job Runner は複数のレコードを一括処理することを得意とします。カスタマイズによっては「kintone アプリのカスタマイズ」では実現できなかった、または複雑な方法を取る必要があったカスタマイズが、Job Runner ではスマートに実現できます。

「kintone アプリのカスタマイズ」とは考え方が異なる点もありますので、それぞれの向き・不向きも含め、Job Runner のカスタマイズでできることを知り、活用していただきたいと思います。

この冊子の習得目標

1. Job Runner の特徴を理解する
2. 定期実行タスクの基本的なカスタマイズの作成方法を理解する
3. kintone アプリの Webhook の基本的なカスタマイズの作成方法を理解する
4. kintone アプリのカスタマイズと Job Runner のカスタマイズのどちらで作成すべきか判断できるようになる

この冊子は、2022 年 10 月 01 日時点の仕様をもとに作成しています。

目次

1. はじめに理解していただきたいこと

1-1. 本書で使用する用語

本書ではカスタマインの画面や用語を以下のように記載します。

アクション

　カスタマインの設定における 1 単位です。「やること」と「条件」を組み合わせた 1 セットが 1 つのアクションとなります。

やること

　カスタマイン設定画面の左側で選択する「やること」は、本書では [やること：テーブル行をレコードとして取得する] のように記載します。

条件

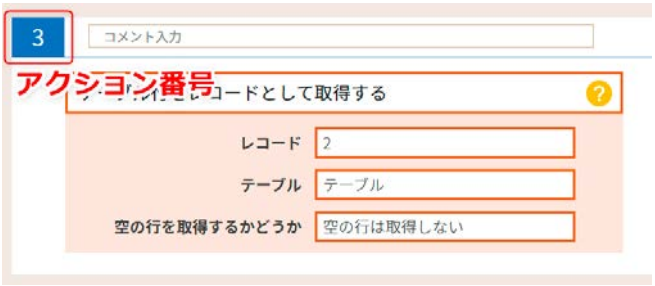
　カスタマイン設定画面の右側で選択する「条件」は、本書では＜条件：レコード 1 行が準備できた時＞のように記載します。

追加条件

　「条件」の下部で「条件を追加」を選択して設定する「追加条件」は、本書では＜追加条件：フィールド値が特定の値ならば＞のように記載します。

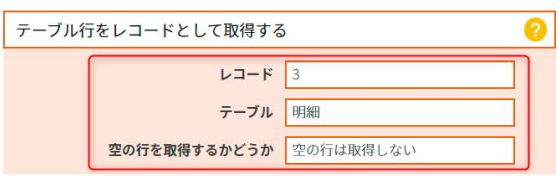
アクション番号

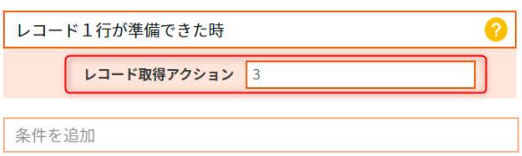
　カスタマイン設定画面で各アクション（設定）の左上の数字です。本書内で【アクション：1】と記載した場合には、アクション番号 1 番の設定を指します。



パラメーター

　やること や 条件 の設定項目です。

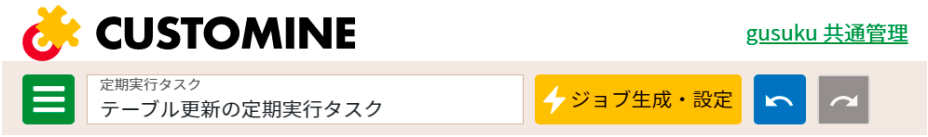




ジョブ

Job Runner（kintone アプリの Webhook/ 定期実行）の「ジョブ生成・設定」ボタンを押すと、カスタマイズを元に作成されます。実行時間になった時（定期実行タスク）や、Webhook の通知を kintone から受け取った時（kintone アプリの Webhook）に実行されます。

カスタマイズを変更した際は「ジョブ生成・設定」ボタンを再度押して、カスタマイズの変更をジョブに反映する必要があります。



Webhook

Web アプリケーション同士が連携するときの考え方の一つで、Web アプリケーションでイベントが実行された際、外部サービスに HTTP/HTTPS で通知する仕組みのことを Webhook と呼びます。（出典：Cybozu Developer Network）

Webhook という言葉自体は kintone の専門用語ではなく、広く Web サービスにおいて利用される用語です。

kintone の Webhook 通知

kintone では、アプリの設定 > Webhook > Webhook の編集画面で Webhook の設定が可能です。

kintone アプリ上で「通知を送信する条件」でチェックした操作が行われた際に、「Webhook URL」に指定した URL に Webhook の通知が送信されます。下記の例では、kintone アプリにレコードの追加が行われた際に、カスタマインに Webhook の通知を送る設定を行っています。

kintone の Webhook の設定については、kintone ヘルプをご確認ください。
「Webhook を設定する」
https://jp.cybozu.help/k/ja/user/app_settings/set_webhook/webhook.html

kintone アプリの Webhook

カスタマインの Job Runner で作成することのできるカスタマイズの種類です。

カスタマイズを作成すると URL が作成され、生成された URL を kintone の Webhook の設定で登録することで、kintone の Webhook 通知が発生すると、登録したカスタマインの URL に通知が届き、カスタマイズが実行されます。

1-2. それぞれの特徴

「kintone アプリのカスタマイズ」

ユーザーの画面操作を「条件」にアクションが実行されます。「条件」となるのは、画面表示、レコード保存、ボタン操作などです。ただ、kintone の仕様上 CSV 読み込みなどカスタマイズの起点とならない操作もあります。

どのような「条件」があるかについてはこちらのページをご確認ください。

『「条件」一覧』

https://docs-customine.gusuku.io/ja/conditions/

具体例としては、アプリにタブや色を付けて見やすくする、または操作しやすくするカスタマイズや、フィールドの値を変更した時に他のフィールドに値をコピーするといった、ユーザーの操作に基づいた処理に適しています。

「kintone アプリの Webhook」

kintone の Webhook 通知をカスタマインが受け取った事を起点にカスタマイズが実行されます。

具体例としては、トヨクモ社のフォームブリッジのような外部サービスからレコードが追加された時に、レコード追加に連動して処理をすることなどがあります。

「定期実行タスク」

設定した時間に実行されます。時間は自由に設定できますが、最短の実行単位は 1 時間ごとです。

詳細はこちらのページをご確認ください。

「Job Runner ジョブ生成・設定ダイアログの使い方」
https://support.gusuku.io/ja-JP/support/solutions/articles/36000266684

具体例としては、年齢計算のように毎日決まった時間に処理をしたい場合や、基幹システムから定時でデータ連携された後に更新作業をする場合などに適しています。

kintone アプリのカスタマイズとの組み合わせ

■ kintone アプリの Webhook の場合
kintone アプリのカスタマイズの [やること：**kintone Webhook を起動する**] を使うと、「kintone アプリのカスタマイズ」から kintone アプリの Webhook で作成したカスタマイズを呼び出すことも可能です。

通常の Webhook は 1 レコードのみですが、[やること：**kintone Webhook を起動する**] を使ってカスタマイズを呼び出した場合は複数レコードを対象にすることが出来ます。この仕組みは一覧画面から複数レコードを指定して一括して処理するのにも適しています。

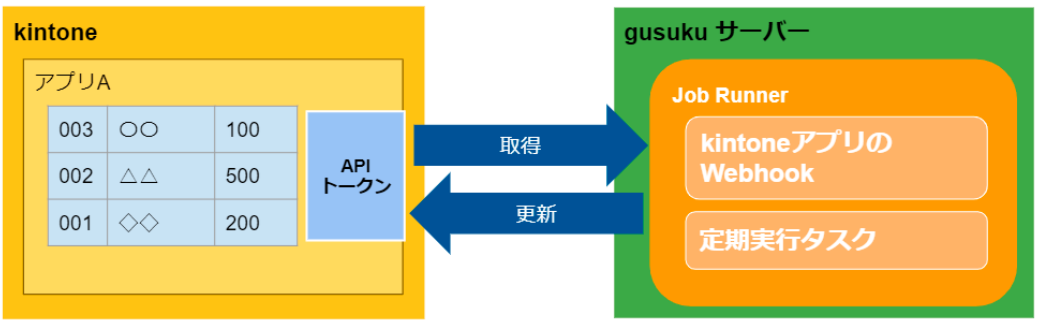
※ [やること：**kintone Webhook を起動する**] は内部的にカスタマイズを呼び出す仕組みであり、kintone 上で Webhook を発生させる仕組みではありません。

■定期実行 の場合
kintone アプリのカスタマイズの [やること：**定期実行タスクを直ちに起動する**] を使うと、「kintone アプリのカスタマイズ」から定期実行で作成したカスタマイズを呼び出すことも可能です。

例えば、CSV ファイルを読み込んだ後にボタンを押したら一括で年齢計算を行うなどのカスタマイズで有効です。

1-3.Job Runner のカスタマイズが動く仕組み

Job Runner のカスタマイズは、gusuku サーバー上で動作します。
gusuku サーバーは kintone とは別の場所に存在しているため、レコードの取得や更新は「API トークン」の仕組みを使って行います。



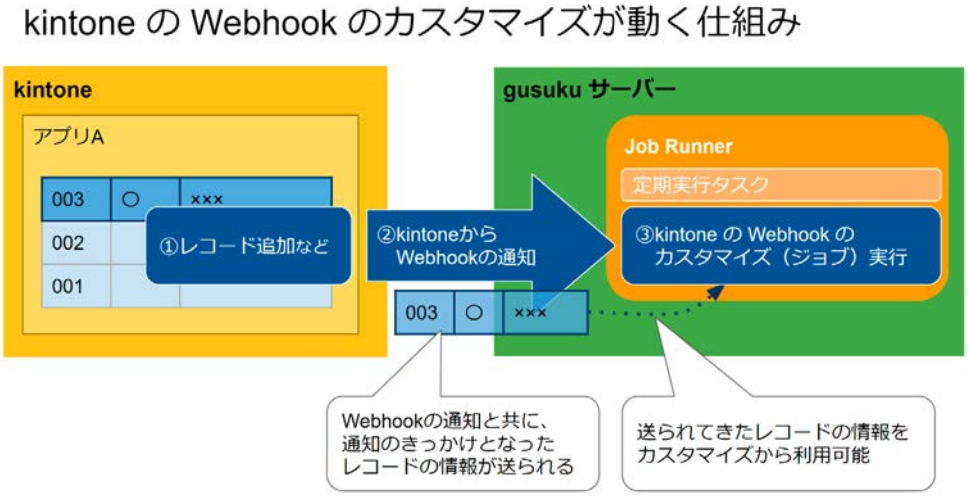
APIトークンは、kintoneのアプリの設定画面で生成が可能です。
各APIトークンに対しては個別にアクセス権の指定が可能となっており、Job Runnerのカスタマイズでは、カスタマイズで使用するAPIトークンのアクセス権に基づいてkintoneにアクセスします。

下記は、kintoneのアプリの設定画面のAPIトークンの設定画面です。

kintoneアプリのWebhook

kintoneアプリのWebhookのカスタマイズは、kintoneから送信されたWebhookの通知の受信を起点にジョブが実行されます。

Webhookの通知には、通知のきっかけとなったkintoneのレコードの情報が含まれるため、送られてきたレコードの情報をカスタマイズから利用可能です。
Webhookの通知に含まれるレコード以外の情報をkintoneから取得したり更新を行う場合は、対象となるkintoneアプリに接続して処理を行います。



1-4. kintoneアプリのカスタマイズとJob Runnerの向き不向き

Job Runnerは、kintoneアプリのカスタマイズでは実現できなかった処理を可能にします。

なぜかというとkintoneアプリのカスタマイズは、基本的にユーザーの操作を起点に処理が実行されます。そのため、ユーザーが一覧画面を開く、ボタンを押す、プロセス管理のアクションを実行する……など、ユーザーが操作を行わなければカスタマイズを実行することができません。ですがJob Runnerであれば、例えば定期実行タスクのカスタマイズを使用することで、毎日午前1時になったら処理を実行する、毎週金曜日に実行する……のように、決まった時間に自動的にカスタマイズが動き、ユーザーが操作していない間にkintoneのレコードを更新する……といった事が可能です。

また、Job Runner の kintone アプリの Webhook のカスタマイズを使用すれば、フォームブリッジなどの外部サービスからレコードが追加された時にカスタマイズを実行できます。

このように kintone アプリのカスタマイズではできなかった処理を、Job Runner では行う事ができます。

また、Job Runner は kintone をデータの塊……データベースのように扱う事ができます。” 今開いている画面 ” や ” ユーザーが選択したレコード ” のようなユーザーの画面操作に関連する機能や条件を持たない代わりに、複数のレコードに対する一括処理が得意です。

さらに、Job Runner による処理は API トークンの権限によって行われるため（※後述）、ゲストスペースでゲストによるレコードの変更をポータルのアプリに反映するといった、kintone アプリのカスタマイズでは権限上の制約によって行えないような処理を実現する事もできます。

さまざまな用途で使用できる Job Runner ですが、制約もあります。

たとえば、Job Runner はジョブが開始したら画面に読み込み中画面を表示し、終わったら画面にダイアログを表示する……といった処理は実現できません。kintone アプリのカスタマイズであれば、処理に長い時間がかかる場合、[やること：**読み込み中画面を表示する**] で読み込み中画面を表示し、処理が終了したら [やること：**読み込み中画面を終了する**] で読み込み中画面を消し、更にダイアログを表示して処理の終了をユーザーに知らせることができます。

ですが Job Runner は、このような ” 処理が終わったら〇〇する ” といったことがとても苦手です。基本的に、Job Runner の処理がいつ終わったかを知る術はありません。そのため、対話的に処理の開始と完了をユーザーに知らせるような処理は作成できず、定期実行タスクのようにユーザーが知らない内に処理が行われ、いつのまにか終わっている…といった処理に向いています。

1-5.Job Runner を使用する際の注意点

Job Runner のカスタマイズは、kintone アプリのカスタマイズと異なる点がいくつかあります。

また、Job Runner 独自の注意点もあります。

アプリスロットの考え方

カスタマインは、アプリスロットを課金対象としています。

kintone アプリのカスタマイズは、1 つのアプリスロットにつき 1 つの kintone アプリをカスタマイズできます。

それに対して Job Runner では、1 つのアプリスロット割り当てで Job Runner のカスタマイズを 1 か月につき 3 時間実行できます。3 時間の中で実行されるカスタマイズは、1 つのカスタマイズでも複数のカスタマイズでもかまいません。また、1 つのカスタマイズの中で複数の kintone アプリのレコードを取得・更新していてもかまいません。

実行時間は 1 つの kintone サブドメイン内で実行された Job Runner のカスタマイズ実行時間の累計で計算されます。

Job Runner で使用するアプリスロットは、あらかじめ Job Runner でいくつ使うかを割り当てておく必要があります。

1 か月につき 3 時間以内の実行であれば 1 アプリスロット、3 時間では足りず、6 時間以内の実行

であれば 2 アプリスロット……のように、契約中のプランで使用可能なアプリスロットの中から、必要な数のアプリスロットを割り当ててください。

なお、1 か月あたりの実行時間がどの程度になるかはレコード数やフィールド数、テーブルの有無や行数など様々な要因によって変わるため、ご自身の環境で実際に実行してどの程度実行時間がかかるかを検証し、判断してください。

「**Job Runner を利用するとき、初めにすること：ジョブへのスロット割り当てについて**」
https://support.gusuku.io/ja-JP/support/solutions/articles/36000267055

なお、契約中のプランによって Job Runner に割り当て可能な上限アプリスロット数は異なるため、その点も注意する必要があります。

「**アプリスロットの考え方・数え方** 」
https://support.gusuku.io/ja-JP/support/solutions/articles/36000077664

Job Runner のカスタマイズは、アプリスロットの割り当てを超えて実行されることはありません。

例えば、Job Runner にアプリスロットを 1 個割り当てた状態で 1 か月中のジョブの実行時間が 3 時間を超えた場合、3 時間を超えるきっかけとなったジョブは最後まで実行されますが (※注 1)、次のジョブはエラーとなり起動しません。

※注 1：ジョブの実行時間が 15 分を超えた場合は、Job Runner の仕様によりアプリスロットの割り当て残り時間に関わらずその時点でジョブは強制終了します。

カスタマイズの動作テストの際も実行時間は消費する

カスタマインではテスト用と本番用のジョブ実行の区別はないため、動作テストの際もジョブを実行した時間ぶんは常に割り当て時間が消費されます。

エラーが発生した際の通知

Job Runner のジョブを実行時にエラーが発生した場合、カスタマインの画面での確認 (※後述) も可能ですが、それとは別に自動的にメールでの通知が行われます。

下記はエラー発生時のメール例です。

メールの通知先は、gusuku の組織メンバーのうち、権限がオーナーに指定されたユーザーのメールアドレスです。

カスタマイズの実行権限

前述の通り、Job Runner は API トークンを使用して kintone にアクセスするため、使用する API トークンに指定した権限に基づいてカスタマイズは実行されます。

対して kintone アプリのカスタマイズでは、kintone の画面を開いて操作しているログインユーザーの権限でカスタマイズは実行されます。

そのため、カスタマイズが権限のエラーによって動作しない場合は、それぞれで確認する場所が異なるので注意してください。

kintone アプリのカスタマイズ：

kintone にログインしているユーザーが、kintone アプリのアプリ / レコード / フィールドのアクセス権を有しているか確認する。

Job Runner のカスタマイズ：

カスタマイズに関わる kintone アプリの API トークンの権限を確認する。

レコードの操作者

また、API トークンを使用した操作は、kintone 上では Administrator による操作として記録される点にも注意が必要です。（出典：Cybozu Developer Network）

そのため、Job Runner によって kintone のレコードの更新を行った場合や、レコードにコメントを投稿するなどした際は、更新履歴には Administrator による操作として記録されます。

詳細は Cybozu Developer Network のページをご確認ください。

「kintone REST API の共通仕様」

https://developer.cybozu.io/hc/ja/articles/201941754

kintone サブドメインをまたぐ処理は作成できない

これは kintone アプリのカスタマイズでも同様ですが、kintone サブドメインをまたいだ処理は Job Runner でもできません。

このため、例えば aaa.cybozu.com のレコードを bbb.cybozu.com にコピーする、といった処理はできません。

コラム：フィールド名とフィールドコード

kintone の仕組みとして、フィールド値を参照する場合は「フィールドコード」を指定します。kintone のフィールドには「フィールド名」と「フィールドコード」があるのでご注意ください。

簡単な見分け方としては、フィールドの個別設定画面で上にあるのが「フィールド名」で下にあるのが「フィールドコード」です。「フィールドコード」を明示的に指定しないと「文字列 _1 行 _1」のように自動でフィールドコードが設定されるため、フィールドコードがどのフィールドを指すのか判別しづらくなります。そのため、フィールド名と同じ名称をつけたり、「フィールド名 _ アプリ名」のようにフィールドコードからフィールド名を類推しやすい名前にしておくことをおすすめします。

2.Job Runner の基本的なカスタマイズ

Job Runner のカスタマイズでは、下記のような流れで処理を行っていきます。

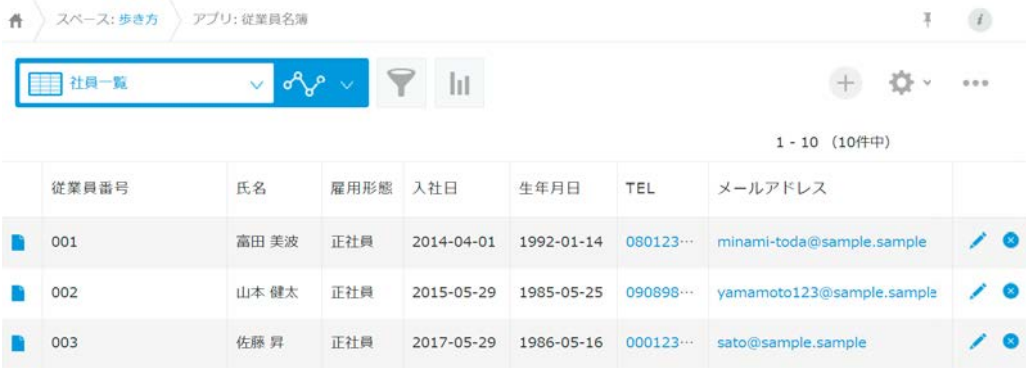
1. 処理対象となる kintone アプリに接続する
2. 処理対象となるレコードを取得する
3. 取得したレコードに対して処理を行う
4. 処理結果を kintone に反映する
5. s カスタマイズの実行

この章では「定期実行タスク」「kintone アプリの Webhook」それぞれにおけるカスタマイズの流れを説明するとともに、両機能で共通する「取得したレコードの使い方」についても説明します。

なお、本書で使用するアプリは、kintone アプリストアで提供されているアプリ以外は、アプリテンプレートとして弊社サポートサイトで配布しています。必要に応じて 以下 URL よりダウンロードして使用してください。
https://support.gusuku.io/ja-JP/support/solutions/folders/36000240808

2-1. 定期実行タスク

例として使用するアプリは、kintone アプリストアの人事労務パックの従業員名簿アプリを使用します。



処理対象とする『従業員名簿』アプリは下記の状態です。

・計算結果の年齢を保存する数値フィールド「年齢」を新たに追加する

カスタマイズでは『従業員名簿』アプリの全レコードに対して、『生年月日』をもとに処理を行った日時点の年齢を計算し、『年齢』フィールドにセットします。

このカスタマイズが毎日自動的に実行されるように設定することで、ユーザーの操作なしで『従業員名簿』アプリの年齢を最新にすることができます。

2-1-1. 処理対象となる kintone アプリに接続する

Job Runner のカスタマイズは、kintone アプリのカスタマイズと違い、” カスタマイズ実行時に画面に表示されているレコード ” がありません。

このため値を取得したい、更新したい、コメントを投稿したいなど何か処理を行いたいレコードについては、あらかじめ取得しておく必要があります。

また、レコードの取得には前述の通り API トークンが必要です。処理対象としたいレコードの取得を行うための kintone アプリへの接続は、この API トークンを使って行います。

kintone アプリへの接続は、[やること：**kintone 接続設定を行う**] を使用します。

[やること：**kintone 接続設定を行う**] を指定したアクションは、レコードの取得・更新を行う kintone アプリの数だけ作成する必要があります。今回はレコードの取得・更新を行うアプリは従業員名簿アプリのみですので、アクションは一つだけ作成します。

「kintone アプリの API トークン」には事前に kintone アプリで設定・取得した API トークンを指定します。各 API トークンに設定すべきアクセス権は以下の通りです。

アプリ名	アクセス権
従業員名簿	レコード閲覧 レコード編集

定期実行タスクのカスタマイズの起点となる条件は＜条件：**実行予定時刻になった時**＞です。これ以外の条件を指定することはありません。

2-1-2. 処理対象となるレコードを取得する

レコードの取得を行います。今回の処理では、従業員名簿アプリの全レコードを対象に年齢の計算を行いたいので、[やること：**全レコードを取得する**] を使用します。

[やること：**全レコードを取得する**] の他にも、「やること：**キーを指定してレコードを取得する**」や「やること：**条件を組み立ててレコードを取得する**」などのやることを使用してレコードを取得することもできます。

これらのやることはキーなどの条件となる値を指定する必要がありますが、前述したとおり Job Runner には " カスタマイズ実行時に画面に表示されているレコード " がないため、条件となる値には固定値を指定するか、または別のアクションで条件としたい値を持つレコードを取得し、そのレコード内のフィールドの値を指定する必要があります。

2-1-3. 取得したレコードに対して処理を行う

2-1-2 で取得したレコードに対して、[やること：**年齢を計算する**] で処理を行います。ここで指定する条件は＜条件：**レコード 1 行が準備できた時**＞です。＜条件：**レコード 1 行が準備できた時**＞は、

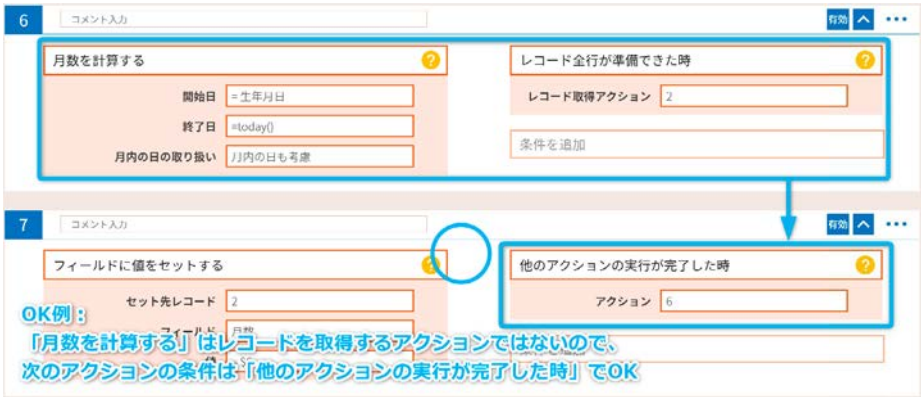
2-1-2 で取得したレコードから 1 レコードずつ取り出し、処理を行います。

Job Runner のカスタマイズでは、レコードを取得したのち、取得したレコードに対しての処理の起点となるひとつ目のアクションの条件は原則＜条件：**レコード 1 行が準備できた時**＞または＜条件：**レコード全行が準備できた時**＞を使用します。

取得したレコードに対しての処理の起点となるアクションで＜条件：**他のアクションの実行が完了した時**＞を指定すると、取得したレコードを正常に扱う事ができないため使用しないように気を付けてください。

NG 例：

OK 例：



なお、Job Runner のカスタマイズではアクションを設定する際に やること より先に 条件 を指定するのがおすすめです。Job Runner のアクションは、条件に指定されたアクションを辿って処理対象となるレコードおよびレコードが所属しているアプリを識別する仕組みとなっています。このため、条件が指定されていない状態ではやることのパラメーターでフィールドを使用する設定ができません。

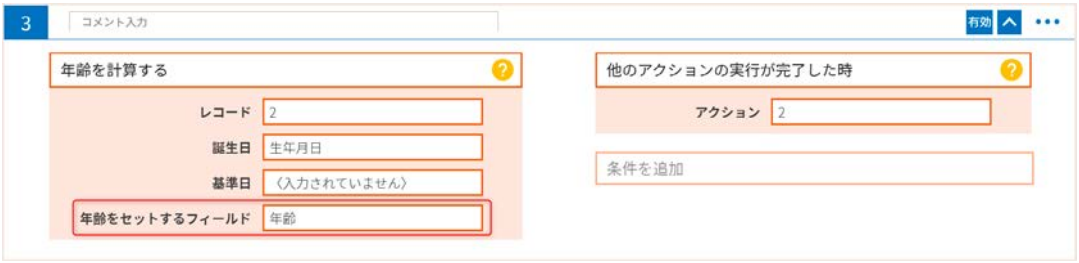
下記は、条件を指定せずにやることのパラメーターで「フィールド選択」ボタンをクリックした際のエラーメッセージです。条件が指定されていないため、どのアプリのフィールドを使用可能なのか

を判別できず、下記のようなエラーが表示されます。

このようなエラーメッセージが表示された場合は、まず条件を指定した上でやることの設定を行ってください。

2-1-4. 処理結果を kintone に反映する

処理結果を kintone のレコードに反映します。[やること：**年齢を計算する**] は、パラメーター「結果をセットするフィールド」に処理結果を保存したいフィールドを指定することで処理結果をフィールドに反映します。



フィールドカテゴリーのやることは、パラメーター「結果をセットするフィールド」で処理結果をフィールドに保存したり、パラメーター「フィールド」に指定したフィールドの値を直接置換することで処理結果を反映できます。

それ以外のやることの結果をレコードに反映する場合は、[やること:**フィールドに値をセットする**] や [やること：**キーの値をもとにレコードを更新する**] などのやることを使って反映します。

それぞれの使い分けについては、3. 実例に学ぶ Job Runner のカスタマイズで説明します。

2-1-5. カスタマイズの実行

カスタマイズができたら、「ジョブ生成・設定」ボタンをクリックし、カスタマイズをもとにジョブを作成します。

作成したカスタマイズを**実行する前**に、下記カスタマイズ実行時の注意点を確認し、準備を行ってください。

カスタマイズ実行時の注意点

Job Runner のカスタマイズは複数のレコードに対する一括処理が可能なため、カスタマイズの誤りによって意図せず大量のレコードが更新される危険性があります。

そういった事故を防ぐため、カスタマイズの実行時は下記のような予防措置をとることをおすすめします。

・誤って想定しないレコードが更新された場合にデータを復旧できるように、CSV への書き出しやレコードのバックアップを行う連携サービスを使用してレコードのバックアップを取得しておく (※注 2)

・更新対象とするレコードを [やること：**クエリで条件を指定してレコードを取得する**] などで限定し、まずは少ないレコードで試してみる

※注 2：gusuku シリーズの 1 つである gusuku Deploit のバックアップオプションでもレコードのバックアップは可能です。

gusuku Deploit については下記の製品ページをご覧ください。

https://deploit.gusuku.io/

カスタマイズの実行

定期実行タスクのカスタマイズは、下記の 3 パターンの実行が可能です。

・直ちに実行

・定期実行

・画面から呼び出して実行

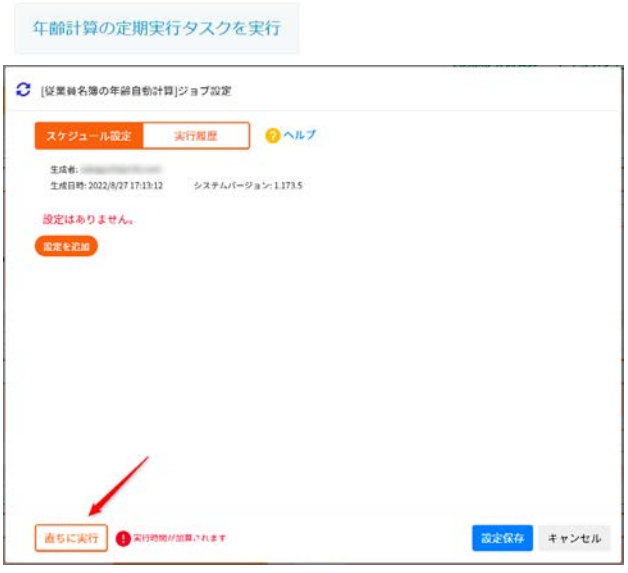
それぞれ主に下記のような用途で使用します。

直ちに実行：

・カスタマイズ作成中のテストで実行する

・複数のレコードに対して行う処理を、1 回だけ Job Runner で処理を行う

例：新しく作った検索用文字列フィールドに対して、既存レコードのフィールドには Job Runner で一括で検索用文字列をセットし、その後は kintone アプリのカスタマイズでレコードが保存された際に検索用文字列の作成を行う　など



定期実行：

・毎日、毎月など決まったタイミングで繰り返し処理を実行する

画面から呼び出して実行：

・ボタンクリックなどユーザーの操作によって Job Runner の処理を呼び出す

例：ユーザーが CSV から kintone にレコードを読み込んだ後で、ボタンを押して Job Runner の処理を呼び出す　など

カスタマイズの実行確認では**直ちに実行**を使用します。ボタンをクリックするとすぐにジョブが実行されます。

実行結果はジョブ設定画面の「実行履歴」タブに表示されます。まだ表示されていない場合は、ジョブ設定画面左上のボタンをクリックしてください。エラーが表示されている場合や、うまく動かないときの対処方法は **うまく動かないときは？** を参照してください。

スケジュール設定

　カスタマイズが想定通り動作することを確認したら、定期実行タスクを実行したいタイミングでスケジュール設定します。

　スケジュールは、下記の間隔で指定ができます。

・1 年毎
・1 ヶ月毎
・1 週間毎
・1 日毎
・1 時間毎

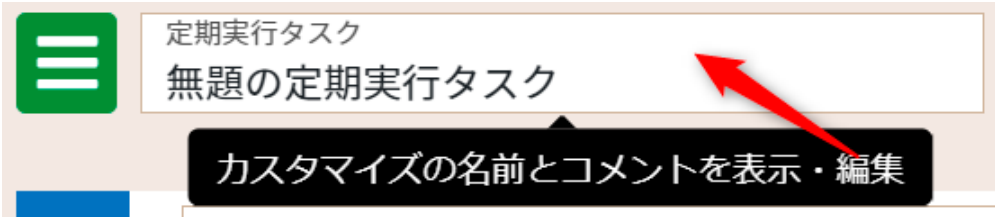
以上が、定期実行タスクの設定です。

コラム：定期実行タスクのカスタマイズ名は必ず指定しましょう

定期実行タスクのカスタマイズは特定の kintone アプリに紐づかないため、カスタマイズ名は既定の名称では kintone のアプリ名が含まれない「無題の定期実行タスク」になっています。

もしカスタマイズ名をこのままにしておくと、ドライブ画面に表示されるカスタマイズが無題の定期実行タスクだらけになり、カスタマイズの識別ができなくなってしまいます。

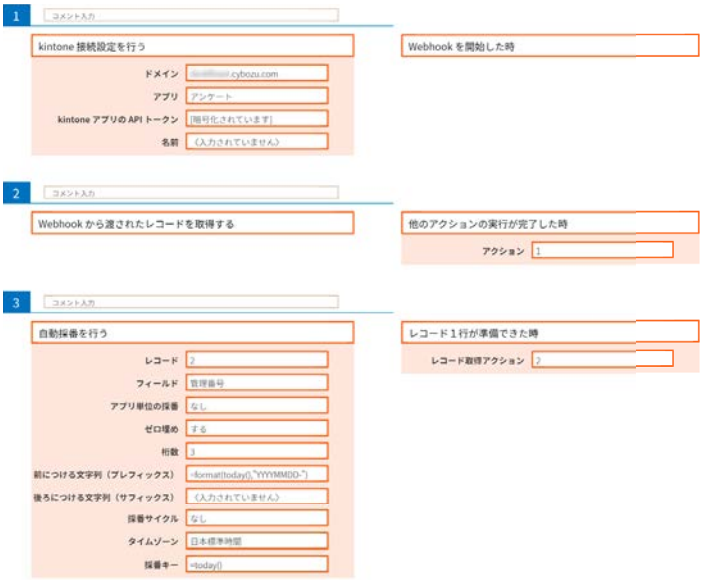
このような状況を防ぐため、定期実行タスクのカスタマイズを作成したら、まずカスタマイズ名を分かりやすい名前に変更することを強くおすすめします。
カスタマイズ名をクリックすると名前の変更とコメントを記載する画面が出てきますので、こちらから変更してください。



2-2.kintone アプリの Webhook

例として使用するアプリは、kintone アプリストアのアンケートアプリを使用します。
処理対象とする従業員名簿アプリは下記の状態です。
・アンケートごとに重複しない採番結果を保存するための「管理番号」フィールド（文字列（1 行））を追加しました。

カスタマイズの仕様は下記です。
・アンケートアプリにレコードが追加されると、管理番号フィールドに下記の形式で採番された管理番号がセットされる。
20220827-001
・ユーザーが kintone の画面でレコードを追加した時に加え、フォームブリッジなどの連携サービスでレコードが追加された時も同様に採番される。



2-2-1. 処理対象となる kintone アプリに接続する

定期実行タスクのカスタマイズと同様に、カスタマイズで使用する kintone アプリは [やること：**kintone 接続設定を行う**] を使用して接続する必要があります。

ただ、例外として下記のレコードだけは [やること：**kintone 接続設定を行う**] で接続しなくても使用することができます。

kintone の Webhook の通知のきっかけとなったレコードの閲覧

レコードの閲覧が可能な理由は、kintone の Webhook の通知に通知の発生元となったレコードの情報が含まれるためです。そのため [やること：**kintone 接続設定を行う**] で接続しなくても [やること：**Webhook から渡されたレコードを取得する**] でレコードの情報を取得し、フィールドの値をカスタマイズで利用できます。

ただ可能なのはあくまで通知の発生元となったレコードの閲覧だけですので、レコードの更新を行う場合は [やること：**kintone 接続設定を行う**] で接続する必要があります。発生元となった以外のレコードを取得・更新する場合や、もちろん他のアプリのレコードを扱う場合も同様に [やること：**kintone 接続設定を行う**] で接続が必要です。

今回は、発生元となったレコードの取得だけでなく自動採番を行った結果でレコードの更新を行いたいので、『アンケート』アプリに接続します。

この時使用する条件は＜追加条件：**Webhook を開始した時**＞を指定します。kintone アプリの Webhook のカスタマイズの起点は、必ずこの条件になります。

今回のカスタマイズを実行するために必要な、kintone アプリの API トークン の権限は下記です。

アプリ名	アクセス権
アンケート	レコード閲覧 レコード編集

そして、[やること：**Webhook から渡されたレコードを取得する**] で kintone アプリの Webhook の通知と共に送信されてきた、通知の発生元となったレコードを取得します。

2-2-2. 取得したレコードに対して処理を行う

ここからは、定期実行タスクのカスタマイズと変わりません。今回の例では、[やること：**自動採番を行う**] で『管理番号』フィールドに採番を行います。条件は＜条件：**レコード 1 行が準備できた時**＞です。

2-2-3. 処理結果を kintone に反映する

[やること：**自動採番を行う**] は、パラメーター「フィールド」に指定したフィールドに採番結果をセットしますので、やることひとつで処理結果の反映も完了します。

2-2-4. カスタマイズの実行

カスタマイズが確認可能なところまで完成したら、「ジョブ生成・設定」ボタンを押し、ジョブを作成します。

なお、kintone アプリの Webhook のカスタマイズはジョブ設定画面に「テスト実行」ボタンがありますが、定期実行タスクのカスタマイズと異なり、基本的に「テスト実行」では実行できません。

試しに「テスト実行」で実行してみると、「(2) Webhook から URL を渡されていません。」とのエラーが出力されます。

これは [やること：**Webhook から渡されたレコードを取得する**] が、kintone から送信された Webhook の通知からレコードを取得するやることだからです。「テスト実行」では kintone から Webhook の通知が送られてこないため、レコードの取得を試みた結果、レコードを得られず「(2) Webhook から URL を渡されていません。」といったエラーになるのです。

そのため、kintone アプリの Webhook のカスタマイズを実行確認するためには、下記のいずれかの方法でカスタマイズにレコードを渡す必要があります。

- kintone アプリの設定で kintone に Webhook の通知を発生させる設定を行い、kintone からの Webhook の通知を起点にカスタマイズを実行する
- kintone の Webhook の通知は使わずに、kintone アプリのカスタマイズから Job Runner の「kintone アプリの Webhook のカスタマイズ」を直接呼び出して実行

kintone アプリの Webhook の通知設定

カスタマインの「ジョブ生成・設定」ボタンを押し、ジョブ設定画面に表示された URL をコピーします。

kintone の アプリの設定 >Webhook>Webhook の追加 から、kintone アプリに Webhook の設定を追加します。このとき、Webhook URL に先ほどコピーした URL を貼り付けます。Webhook の設定は、Webhook URL に指定した宛先に、kintone から Webhook の通知を送るための設定です。Webhook URL の他に、通知を送信する条件には適切な設定を行い、有効化にもチェックを入れて保存し、アプリを更新してください。

カスタマイズの実行

今回の例では kintone アプリにレコードが追加された時に kintone から Webhook の通知をする設定をしたので、kintone アプリにレコードを追加すれば kintone から Webhook URL に設定した URL に向けて通知が行われ、その結果カスタマイズが実行されます。

kintone アプリにレコードを追加したのち、実行結果は下記の方法で確認します。

・kintone から正常に Webhook の通知が行われたかを確認する

・kintone の アプリの設定 >Webhook>Webhook の画面で、「ログを確認」から確認できます。

もしこの Webhook ログ の画面に記録がない場合は、kintone の Webhook の設定が誤っているか、または kintone の仕様により Webhook の通知が発生しない操作を行ったため、発生しなかった可能性があります。

kintone の Webhook の通知については、下記のサイボウズ様のページをご確認ください。
https://jp.cybozu.help/k/ja/user/app_settings/set_webhook/webhook.html

・カスタマインのカスタマイズの実行結果を確認する
カスタマインのメイン画面から「ジョブ生成・設定」ボタンを押し、実行履歴タブから確認してください。

2-3. 取得したレコードの使い方

2-3-1.1 レコードずつ処理を行う

取得したレコードに対して、1 レコードずつ個別の処理を行いたい場合は、＜条件：**レコード 1 行が準備できた時**＞ を使用します。

・レコード内のフィールドの値によって個別に処理を行いたい
・レコード内のテーブルを扱いたい
・レコード内の関連レコード一覧に表示されたレコードの集計を行いたい

下記の例は、『顧客マスタ』アプリの『顧客名』フィールドの値をもとに、全レコード一括で検索用文字列を作成し、『顧客名検索用』フィールドにセットするカスタマイズです。

『顧客マスタ』アプリに下記のようなレコードが登録されている場合を例にとります。
今回のカスタマイズを実行するために必要な、kintone アプリの API トークン の権限は下記です。

アプリ名	アクセス権
顧客マスタ	レコード閲覧 レコード編集

レコード番号	顧客コード	顧客名	顧客名検索用
4	C0004	ヴィクトリーフィールド株式会社	
3	C0003	山田商事株式会社	
2	C0002	田中マーケティングテクノロジー	
1	C0001	アールスリーインスティテュート	

アクション２ [やること:**全レコードを取得する**] で『顧客マスタ』アプリの全レコードを取得します。
次のアクション３ [やること：**検索用文字列を作成する**] でアクション２で取得したレコードに対して検索用文字列を作成するのですが、この時の条件は＜条件：**レコード１行が準備できた時**＞を指定します。

＜条件：**レコード１行が準備できた時**＞は、パラメーター「レコード取得アクション」に指定したアクション（ [やること：**全レコードを取得する**] などの、レコードを取得するやることが指定されたアクション番号を指定します）で取得したレコードの中から、１レコードが準備できる度に発生します。

そのため、アクション３ [やること：**検索用文字列を作成する**] は、レコードの数だけ実行されることになります。
また、アクション３ [やること：**検索用文字列を作成する**] のやることでは、常に１レコードだけを扱う事ができます。

他のレコードの事は見えないため、集計や絞り込みには使えません。

その代わり、取得した 1 レコードを 1 つずつ扱う事ができるため、レコードごとの処理を行う事ができます。

今回は、取得したレコード内のフィールドの値を元に、検索用文字列を作成し、フィールドにセットしました。ジョブを実行すると、下記のような結果が得られます。

レコード番号	顧客コード	顧客名	顧客名検索用
4	C0004	ヴィクトリーフィールド株式会社	ヴ ヴィ ヴィク ヴィクト ヴィクトリ ヴィクトリー ヴ…
3	C0003	山田商事株式会社	山 山田 山田商 山田商事 山田商事株 山田商事株式 山…
2	C0002	田中マーケティングテクノロジー	田 田中 田中マ 田中マー 田中マーケ 田中マーケテ 田…
1	C0001	アールスリーインスティテュート	ア アー アール アールス アールスリ アールスリー ア…

＜条件：**レコード 1 行が準備できた時**＞を使用した実例は、実例に学ぶ Job Runner のカスタマイズを参照してください。

コラム：どちらの条件でも処理が可能なやること

やることによっては＜条件：**レコード 1 行が準備できた時**＞でも＜条件：**レコード全行が準備できた時**＞でもどちらを使っても処理が可能なものも存在します。その場合、特に全行をまとめて処理したい理由がなければ＜条件：**レコード 1 行が準備できた時**＞を使用する方がおすすめです。

理由は、レコードを取得するアクションで取得したレコードの数が多い場合に、＜条件：**レコード全行が準備できた時**＞の方が比較的メモリエラーが発生しやすいためです。メモリエラーは、Job Runner のシステム上のメモリに収まりきらないほどの大量のレコードを処理しようとした際に起きるエラーで、下記のようなメッセージが表示されます。

ジョブは強制終了しました。メモリを使い切った可能性があります。**The job has crashed. Probably out of memory.**

＜条件：**レコード全行が準備できた時**＞は、取得したすべてのレコードをまとめて扱うため、このメモリエラーが起きやすいのです。

＜条件：**レコード 1 行が準備できた時**＞は、1 行準備できたら処理、1 行準備できたら処理…というように、全てのレコードが取得されるのを待たずに、準備できたレコードから 1 レコードずつ取得され、処理が実行されます。そのため、メモリに大量のレコードが存在する状況になりにくく、比較的メモリエラーが起きづらくなります。

※あくまで比較的メモリエラーになりにくいだけで、ならないわけではありません。

参考までに、＜条件：**レコード 1 行が準備できた時**＞でも＜条件：**レコード全行が準備できた時**＞でもどちらを使っても処理が可能な代表的なやることとしては、[やること：**年齢を計算する**] や、今回例でも使用した [やること：検索用文字列を作成する] のような、処理結果をセットするフィールドをパラメーターとして持つやることが挙げられます。

2-3-2. 全レコードをまとめて処理する

取得したレコードに対して、下記のような全レコードが揃ってからでなければ実行できないような処理を行いたい場合は＜条件：**レコード全行が準備できた時**＞を使用します。

・集計
・絞り込み

下記の例は、『売上』アプリから全レコードを取得し、レコード内の『合計金額』フィールドの値を合計するカスタマイズです。

『売上』アプリに下記のようなレコードが登録されている場合を例にとります。

勘定月	売上日	顧客コード	顧客名	合計金額	税込金額(消費税率8%)
202209	2022-09-23	C0002	田中マーケティングテクノロジー	648,000	699,840
202209	2022-09-15	C0002	田中マーケティングテクノロジー	1,320,000	1,425,600
202209	2022-09-07	C0004	ヴィクトリーフィールド株式会社	816,000	881,280

今回のカスタマイズを実行するために必要な、kintone アプリの API トークン の権限は下記です。

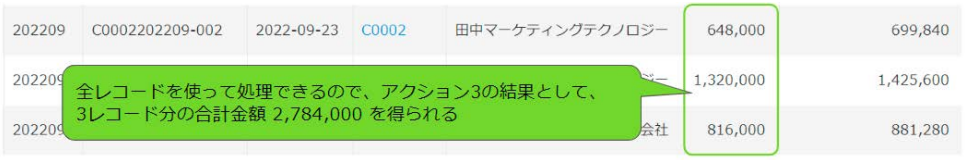
アプリ名	アクセス権
売上	レコード閲覧

※集計した結果を kintone のレコードに保存する場合は、レコード閲覧の他にレコード追加やレコード更新の権限が必要です。

アクション２[やること：**全レコードを取得する**] で『売上』アプリの全レコードを取得します。
次のアクション３[やること：**レコード中のフィールド合計値を計算する**] でアクション２で取得したレコードを集計するのですが、この時の条件は＜条件：**レコード全行が準備できた時**＞を指定します。

＜条件：**レコード全行が準備できた時**＞ は、パラメーター「レコード取得アクション」に指定したアクション（［やること：**全レコードを取得する**］などの、レコードを取得するやることが指定されたアクション番号を指定します）で取得したレコードが全て取得完了し、カスタマイズで使用可能な状態になったときに発生します。

そのため、＜条件：**レコード全行が準備できた時**＞が指定されたアクション３のやること［やること：**レコード中のフィールド合計値を計算する**］では、アクション２で取得した全レコードを使って処理できます。ですので、アクション２で取得した全レコードの合計金額を集計できます。



もしこのとき、誤って＜条件：**レコード１行が準備できた時**＞を指定してしまうと、アクション３では取得したレコードの中から１レコードしか扱う事ができないため、合計金額は１レコード分の金額になってしまいます。

誤った条件の例：

なお、このカスタマイズ例では集計した結果をレコードに反映する処理は省略しています。集計した結果を kintone のレコードに反映する方法については、次章の実例に学ぶ Job Runner のカスタマイズを参照ください。

3. 実例に学ぶ Job Runner のカスタマイズ

3-1.1 レコードと全レコードの組み合わせ

『売上』アプリのレコードについて、『勘定月』フィールド別に合計金額を集計し、『月別売上合計』アプリに勘定月ごとのレコードを作成する処理を例にとります。

下記は、『売上』アプリのレコードです。勘定月が「202209」と「202210」のレコードが存在しています。

勘定月	売上番号	売上日	顧客コード	顧客名	合計金額	税込金額(消費税率8%)
202210	C0003202210-001	2022-10-20	C0003	山田商事株式会社	48,000	51,840
202210	C0001202210-001	2022-10-07	C0001	アールスリーインスティテュート	1,440,000	1,555,200
202209	C0002202209-002	2022-09-23	C0002	田中マーケティングテクノロジー	648,000	699,840
202209	C0002202209-001	2022-09-15	C0002	田中マーケティングテクノロジー	1,320,000	1,425,600
202209	C0004202209-001	2022-09-07	C0004	ヴィクトリーフィールド株式会社	816,000	881,280

カスタマイズ例は下記です。

Job Runner のカスタマイズは、” このアクションで扱っているレコードはどのレコードなのか ” を意識して作成する必要があります。ここからは、アクションごとにどのレコードを扱っているのかを図示してご紹介します。

まず、[やること：**kintone 接続設定を行う**] で集計元となるレコードを持つ『売上』アプリと、集計後の値を保存するための『月別売上合計』アプリに接続します。

今回のカスタマイズを実行するために必要な、kintone アプリの API トークン の権限は下記です。

アプリ名	アクセス権
売上	レコード閲覧
月別売上合計	レコード閲覧 レコード追加 レコード編集

次に、『売上』アプリから全レコードを取得します。今回の処理は勘定月ごとの合計金額を集計したいので、集計の軸となるのは『勘定月』フィールドです。ただし、kintone に登録されたレコードは以下のように同じ勘定月のレコードが複数存在します。

勘定月	売上番号	売上日	顧客コード	顧客名	合計金額	税込金額(消費税率8%)
202210	C0003202210-001	2022-10-20	C0003	山田商事株式会社	48,000	51,840
202210	C0001202210-001	2022-10-07	C0001	アールスリーインスティテュート	1,440,000	1,555,200
202209	C0002202209-002	2022-09-23	C0002	田中マーケティングテクノロジー	648,000	699,840
202209	C0002202209-001	2022-09-15	C0002	田中マーケティングテクノロジー	1,320,000	1,425,600
202209	C0004202209-001	2022-09-07	C0004	ヴィクトリーフィールド株式会社	816,000	881,280

そのため、上記のレコードを以下のように勘定月ごとに 1 件ずつになるように加工する必要があります。

勘定月
202210
202209

今回のように特定のフィールドを集計軸として集計のカスタマイズを作成する際によく使う方法として、下記のような方法があります。

1. 集計対象となるレコードを取得する
2. 1で取得したレコードを、集計軸となるフィールドで重複を除去する
3. 2で重複がなくなった値ごとに集計を行う

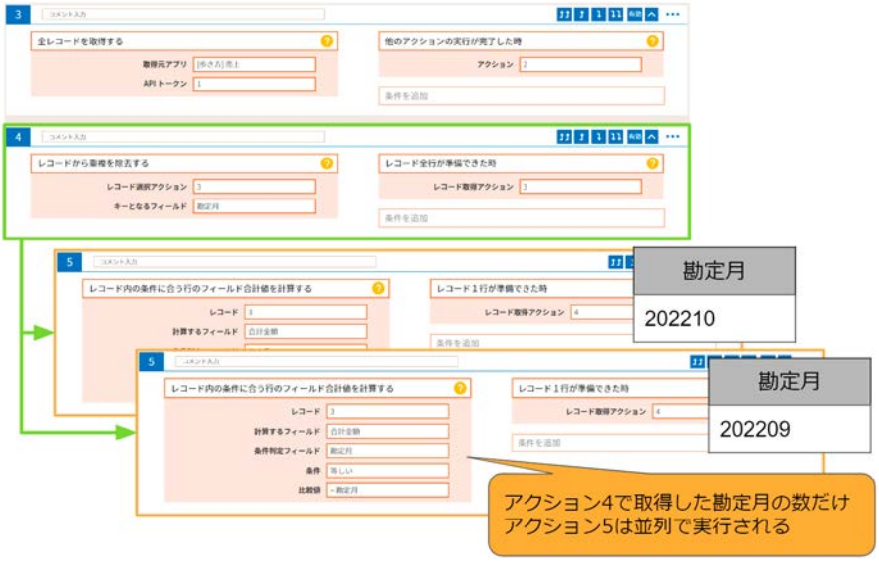
このようなカスタマイズをすることで、集計対象となるレコードの中から必要な集計軸を自動的に抽出して集計できます。

実際のカスタマイズでは、[やること：**全レコードを取得する**] で『売上』アプリのレコードを取得し、[やること：**レコードから重複を除去する**] で、集計軸としたい『勘定月』フィールドを「キーとなるフィールド」に指定して重複を除去します。

このとき指定する条件は＜条件：**レコード全行が準備できた時**＞です。[やること：**レコードから重複を除去する**] は、全レコードの中に存在する『勘定月』から重複を除去したいので、この条件を使用します。

集計軸となる『勘定月』を重複のない状態で取得できたら、集計を行っていきます。

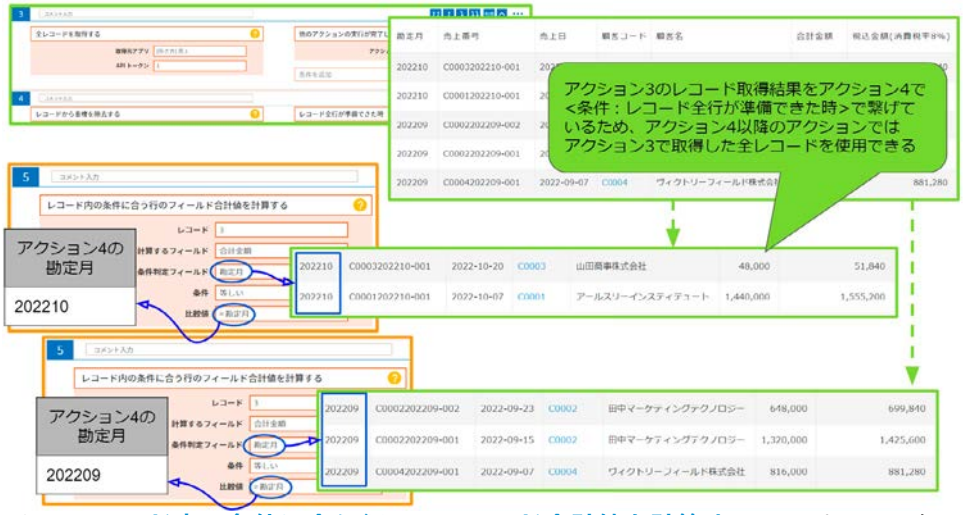
アクション５のやることは [やること：**レコード内の条件に合う行のフィールド合計値を計算する**]、条件は＜条件：**レコード１行が準備できた時**＞です。＜条件：**レコード１行が準備できた時**＞のパラメーター「レコード取得アクション」に「4」を指定し、アクション４で重複を除去した『勘定月』ごとの処理を行う事ができます。アクション５は、アクション４で取得した『勘定月』の数だけ並列で実行されるため、今回の例では「202210」と「202209」の２回実行されます。



[やること：**レコード内の条件に合う行のフィールド合計値を計算する**] の設定についても見ていきます。[やること：**レコード内の条件に合う行のフィールド合計値を計算する**] のパラメーター「レコード」には「3」を指定していますので、集計対象となるレコードはアクション 3 で指定した『売上』アプリの全レコードが対象となります。

Job Runner の処理では、＜条件：**レコード全行が準備できた時**＞や＜条件：レコード 1 行が準備できた時＞で繋がった、前のアクションで取得したレコードを後のアクションで利用できます。今回の例では、アクション 3 の取得結果をアクション 4 で＜条件：**レコード全行が準備できた時**＞で繋げているため、アクション 4 以降のアクション 5、アクション 7 からは、レコードを選択するパラメーターでアクション 3 を選択すると、アクション 3 で取得した『売上』アプリの全レコードを使用できます。

下記の例では、アクション 5 の集計対象となるレコードに「3」を指定することで、” アクション 3 で取得されたレコード内で、条件と一致するレコードを集計対象にする ” ことができます。



[やること：**レコード内の条件に合う行のフィールド合計値を計算する**] のパラメーターについても見ていきます。

パラメーター	指定する値
レコード	集計対象のレコード
計算するフィールド	集計対象のレコード内の、集計したいフィールド
条件判定フィールド	集計対象のレコード内の、集計軸となるフィールド
比較値	集計軸の値。今回はアクション4の結果を指定

同じフィールド名、フィールドコードなので分かりづらいですが、「条件判定フィールド」に指定しているフィールドは、アクション 3 のレコード内のフィールド『勘定月』、「比較値」に指定しているのはアクション 4 で重複を除去した結果得られた『勘定月』（つまり、「202210」と「202209」）です。

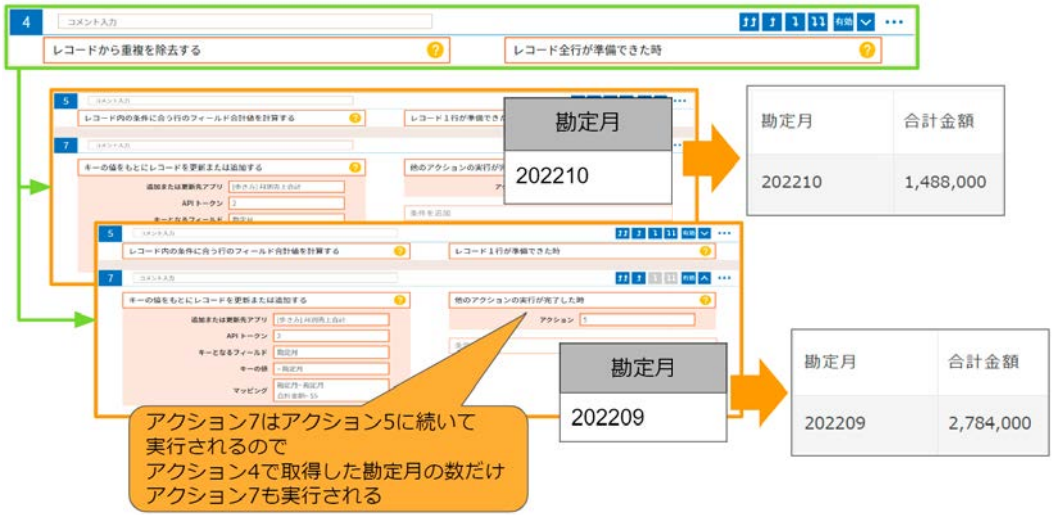
アクション 5 は、アクション 5 の条件が＜条件：**レコード 1 行が準備できた時**＞、「レコード取得アクション」が「4」なので、アクション 4 の結果として得られる「202210」と「202209」の値 2 つ分、つまり 2 回実行されます。

kintone アプリのカスタマイズ（画面のカスタマイズ）ですと、他のアクションの結果をパラメーターに指定する場合、「= $1. フィールドコード」のような記述が基本的に必要となりますが、Job

Runner の場合は＜条件：**レコード１行が準備できた時**＞が指定されたアクション以降では、＜条件：**レコード１行が準備できた時**＞のパラメーター「レコード取得アクション」で指定したレコード内のフィールドについて、「= フィールドコード」のようなアクション番号なし、フィールドコードのみでの指定が可能です。

最後に、集計した結果を『月別売上合計』アプリに反映します。今回は、『月別売上合計』アプリにレコードがなければレコード追加、あれば更新を行います。レコードがある・ないの判断は、『月別売上合計』アプリの『勘定月』フィールドの値で判断します。

アクション７の条件は＜条件：**他のアクションの実行が完了した時**＞です。アクション５はレコードを取得する やること ではないので、アクション５の次に実行したいアクションの条件は＜条件：**他のアクションの実行が完了した時**＞で問題ありません。アクション７はアクション５に続いて実行されますので、アクション５と同様に、今回は「202210」と「202209」の２回実行されます。



以上のカスタマイズを実行すると、『売上』アプリの全レコードを『勘定月』フィールドを集計軸に集計した結果を、『月別売上合計』アプリに追加・更新できます。

＜条件：**レコード１行が準備できた時**＞と＜条件：**レコード全行が準備できた時**＞の二つの条件の動きを理解し、” このアクションではどのレコードを扱っているか ” を意識しながらカスタマイズを行ってください。

【参考】日付やフィールドの値ごとの集計を行う別の方法

　日付やフィールドごとの集計を行う方法として、[やること：**レコードをフィールド値毎に束ねる**] と [やること：**レコードを日付毎に束ねる**] があります。これらの やること を使用すれば、前述したようなフィールド値によって集計する処理のほか、日付フィールドの値を月単位・週単位などで集計するといった様々な単位での集計を比較的簡単に行う事ができますので、こちらの方法もお試しください。

今回のカスタマイズを実行するために必要な、kintone アプリの API トークン の権限は下記です。

アプリ名	アクセス権
売上	レコード閲覧
月別売上合計	レコード閲覧 レコード追加 レコード編集

3-2. テーブルを扱う

Job Runner は、レコードと同様にテーブル行についても 1 行ずつ処理を行う事が得意です。例として、『日報』アプリのテーブルに記録された対応履歴を、『顧客別対応履歴』アプリのレコードと同期するカスタマイズを紹介します。

今回のカスタマイズの要件は下記です。

・『日報』アプリのテーブルに記録された顧客ごとの対応履歴を、『顧客別対応履歴』アプリにレコードとして登録し、同期する

・『日報』アプリのテーブルには、顧客別の対応履歴以外の明細も存在するが、『顧客別対応履歴』アプリと同期するのは『顧客コード』フィールドに値が入っている行のみ

・『顧客別対応履歴』アプリの『顧客コード』フィールドは、ルックアップフィールドを使用している

『日報』アプリのテーブルに『明細番号』フィールドを作成し、報告日と報告者のログイン名と連番を結合した「20220927customine01」のような値を設定しています。この値は、『日報』アプリ内で一意になるように設定しており、『顧客別対応履歴』のレコードと同期をとる際のキー（お互いを結び付け、識別するための値）として使用します。

『顧客別対応履歴』は下記のような結果となります。『日報』アプリの明細番号が「20220927customine01」の行は、顧客コードフィールドが空欄のため『顧客別対応履歴』アプリには同期されません。

レコード番号	日報明細番号	対応日付	顧客コード	対応内容	顧客名
5	20220927customine03	2022-09-27	C0002	問い合わせ	田中マーケティングテクノロジー
4	20220927customine02	2022-09-27	C0004	見積書提出	ヴィクトリーフィールド株式会社

それではカスタマイズを見ていきます。

まずは、[やること:**kintone 接続設定を行う**] で『日報』『顧客別対応履歴』、そして『顧客マスタ』アプリに接続します。

ここでなぜ『顧客マスタ』アプリの接続が必要かについては、コラム：ルックアップフィールドに値をセットするときはをご確認ください。

今回のカスタマイズを実行するために必要な、kintone アプリの API トークン の権限は下記です。

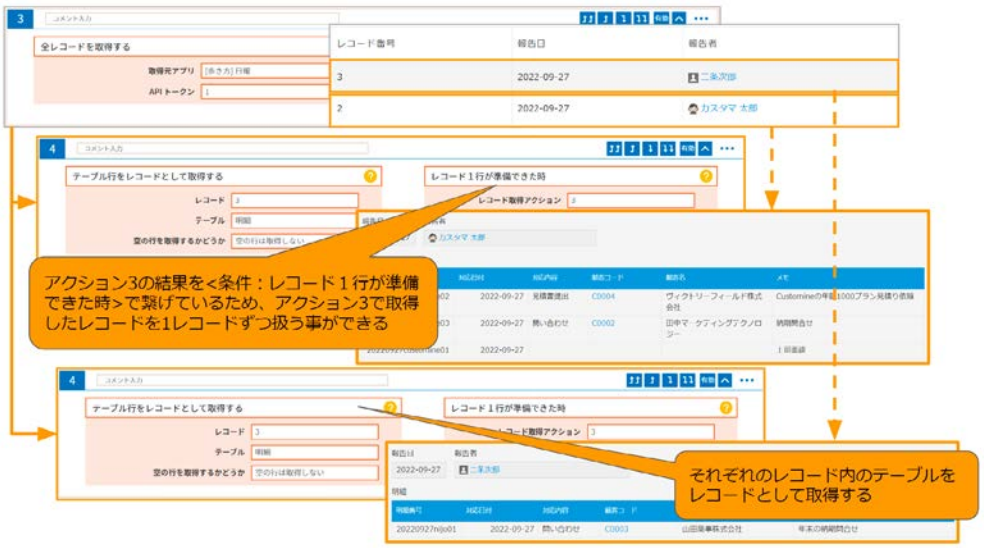
アプリ名	アクセス権
日報	レコード閲覧
顧客別対応履歴	レコード閲覧 レコード追加 レコード編集
顧客マスタ	レコード閲覧 レコード編集

次に、『日報』アプリから全レコードを取得します。

日報アプリのレコードは下記のようになっていますので、アクション 3 ではレコード番号 2、3 の全レコードが取得されます。

次のアクションでは、[やること：**テーブル行をレコードとして取得する**] を < 条件：**レコード 1 行が準備できた時** > で実行します。

パラメーター「レコード取得アクション」には「3」を指定し、アクション３で取得した『日報』アプリの全レコードから１レコードずつ取り出して処理を行う事ができます。



次に、レコードとして取得したテーブル行ごとの処理を行っていきます。

[やること：**レコードをもとに別のレコードを更新または追加する**] を ＜条件：**レコード１行が準備できた時**＞ で使用して、テーブル行を １ 行ずつ『顧客別対応履歴』アプリのレコードと同期をとります。[やること：**レコードをもとに別のレコードを更新または追加する**] は、パラメーター「追加または更新先アプリ」に指定したアプリに、パラメーター「キーとなる更新先のフィールド」の値が一致するレコードがあればレコード更新、なければレコード追加を行うやることです。

今回は、『日報』アプリの『明細番号』フィールドと、『顧客別対応履歴』アプリの『日報明細番号』フィールドの値をキーに同期を行うよう設定しています。

アクション５の設定でポイントとなるのは、＜条件：**フィールド値が空でないならば**＞です。アクション５では、条件に＜条件：**レコード１行が準備できた時**＞を指定しているため、アクション４で取得したテーブル行を １ 行ずつ扱う事ができます。そのため、アクション５の条件の意味は、アクション４で取得したテーブル行の『顧客コード』フィールドの値が空でない場合のみ、アクションを実行する、となります。

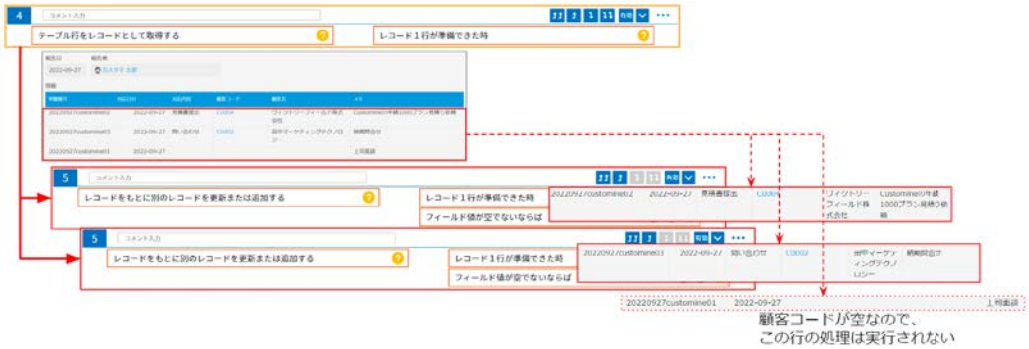
報告日　報告者

2022-09-27　カスタマ 太郎

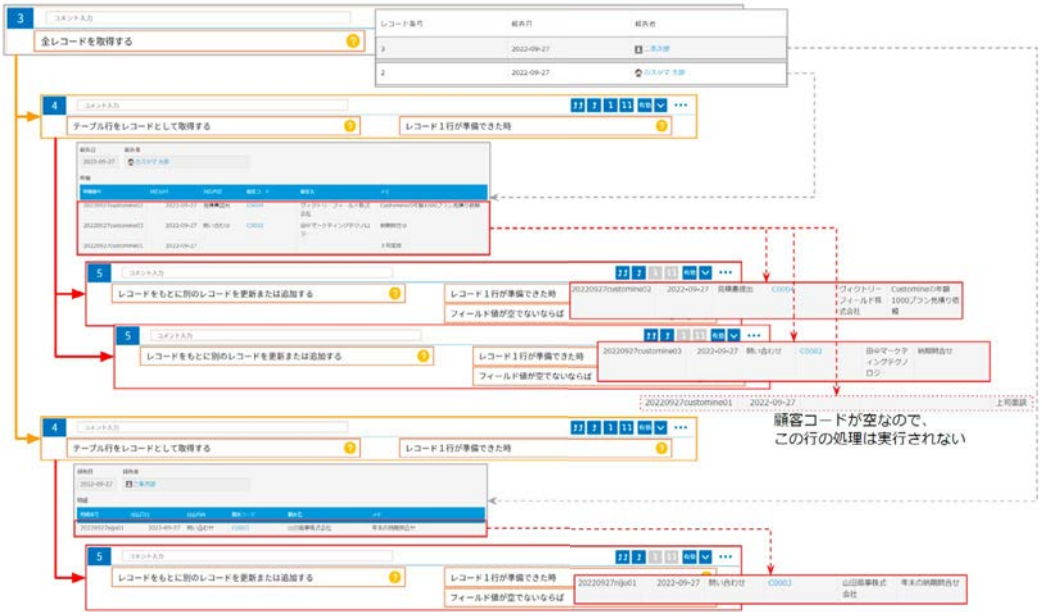
明細

明細番号	対応日付	対応内容	顧客コード	顧客名	メモ
20220927customine02	2022-09-27	見積書提出	C0004	ヴィクトリーフィールド株式会社	Customineの年額1000プラン見積り依頼
20220927customine03	2022-09-27	問い合わせ	C0002	田中マーケティングテクノロジー	納期問合せ
20220927customine01	2022-09-27				上司面談

今回の例では、「20220927customine01」の上司面談の行は顧客コードが空ですので、この行についてはアクション 5 は実行されません。



全体の各アクションの流れと、アクションで扱うレコード (テーブル行) は下記の通りです。



以上が、テーブル行と別のアプリのレコードの情報を同期するカスタマイズです。今回は定期実行タスクのカスタマイズで作成したため、たとえば夜のうちに活動履歴を作成し、翌朝には最新化されている、といった運用が想定されます。

Job Runner の処理では、テーブル行は基本的に [やること：**テーブル行をレコードとして取得する**] でレコードとして取得し、続くアクションではレコードとほぼ同じように扱う事ができます。Job Runner ではテーブルに関するやることが kintone アプリの画面のカスタマイズと比較して少ないのですが、これは [やること:**テーブル行をレコードとして取得する**] でレコードとして取得すれば、その後の処理はレコード用のやることで対応可能であるためです。

コラム：ルックアップフィールドに値をセットするときは

Job Runner の処理でルックアップフィールドに対して値をセットする際は、ルックアップフィールドの「関連付けるアプリ」に指定したアプリについても、[やること：**kintone 接続設定を行う**] で接続を行う必要があります。kintone はルックアップフィールドに値がセットされると「関連付けるアプリ」に指定したアプリからフィールドの値の取得を行うため、その際に使用するための接続設定が必要となります。

カスタマイズでは下記のように指定します。

kintone 接続設定を行った上で、[やること：**レコードをもとに別のレコードを更新または追加する**] などのレコードを追加・更新するやることのパラメーター「追加または更新先アプリの API トークン」に、レコードの更新･追加先アプリの kintone 接続設定に加え、ルックアップの「関連付けるアプリ」に対する kintone 接続設定も共に指定します。

下記のように、パラメーター「追加または更新先アプリの API トークン」の選択で複数の [やること：**kintone 接続設定を行う**] のアクションを指定できます。

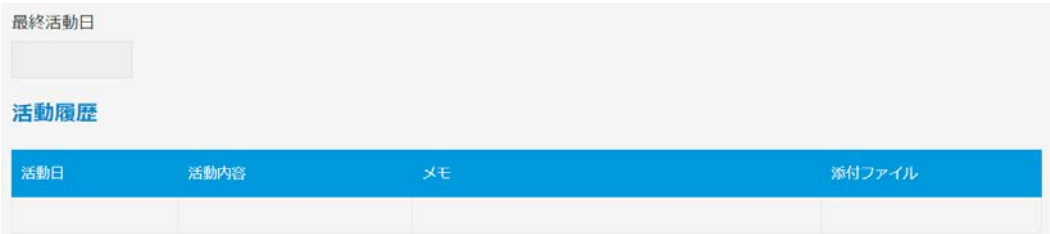
3-3. レコードが取得された時、取得されなかった時のアクションの分岐

[やること：**全レコードを取得する**] などのレコードを取得するアクションの結果を使用するアクションの条件には < 条件：**レコード 1 行が準備できた時** > か < 条件：**レコード全行が準備できた時** > を使用する必要がある事を、これまで紹介してきました。

ですが例外として、レコードを取得するアクションを実行した結果、1 件もレコードを取得できなかったときに実行したいアクションに限り、< 条件：**他のアクションの実行が完了した時** > を使用する必要があります。理由は、< 条件：**レコード 1 行が準備できた時** > と < 条件：**レコード全行が準備できた時** > は、レコードが取得されたときにしか発生しないためです。

カスタマイズ例は、『案件管理』アプリの『活動履歴』テーブルの『活動日』フィールドの値が最大の日付を、テーブル外の『最終活動日』フィールドにセットする処理です。

また、『活動履歴』テーブルに空行しかなければ、最終活動日の値をクリアする処理にて、レコードが取得されなかった場合のカスタマイズを行っています。



まずは、『案件管理』アプリから全レコードを取得し、アクション 21 で取得したレコードを 1 レコードずつ取り出し、更にそのレコード内のテーブル行を [やること：**テーブル行をレコードとして取得する**] で取り出します。

アクション 22、アクション 24 は、アクション 21 で空ではないテーブル行が存在し、レコードとして取得できた場合の処理です。

アクション 23 は、アクション 21 で空でないテーブル行が存在せず、レコードとして取得できなかった場合の処理です。

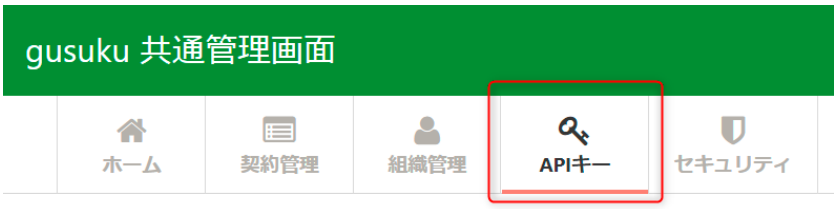
＜条件：**他のアクションの実行が完了した時**＞は、レコードが取得される・されないに関わらず、パラメーター「アクション」に指定したアクションの実行が完了すれば発生します。ですので、＜条件：**他のアクションの実行が完了した時**＞と＜条件：**レコード件数が 0 件ならば**＞を組み合わせることで、”レコードを取得した結果、レコードが 1 件も取得されなかったとき ” のアクションを作成できます。

3-4.kintone アプリの画面のカスタマイズから定期実行タスクをダイレクトに呼び出す

kintone アプリのカスタマイズで、[やること：**定期実行タスクを直ちに起動する**] を使用すると、Job Runner の定期実行タスクのカスタマイズを kintone の画面から呼び出すことができます。

[やること：**定期実行タスクを直ちに起動する**] を使用する際は、「定期実行タスクのカスタマイズ」で作成済みのカスタマイズを指定し、「gusuku API キー」に、gusuku のサービスを使用するための gusuku API キーを指定します。

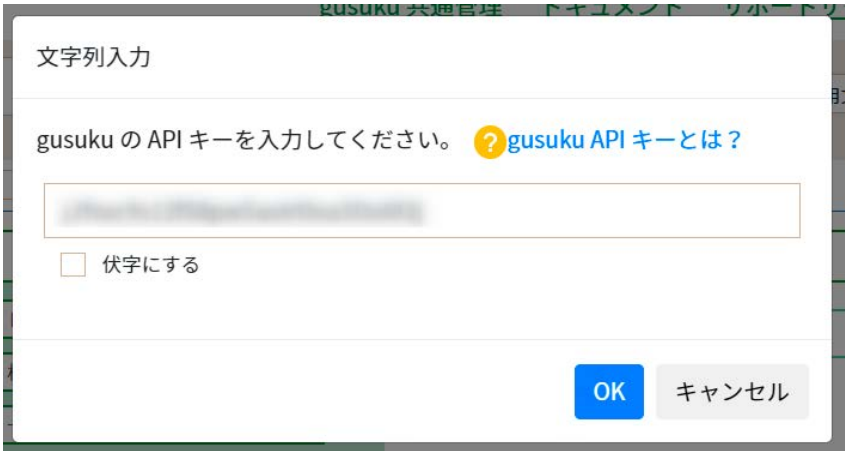
gusuku API キーは、gusuku 共通管理画面の「API キー」タブで作成・確認が可能です。
https://admin.gusuku.io/apikey



新しい gusuku API キーを作成する場合は、画面下部の「API キー作成」にて、キー名称を指定し、「API キーを新規に作成」ボタンをクリックします。

ボタンクリック後に、正常に API キーが作成されれば「API キー一覧」に作成した gusuku API キーが表示されますので、錠マークをクリックして展開し、API キーをコピーしてください。

コピーした gusuku API キーを [やること：**定期実行タスクを直ちに起動する**] のパラメーター「gusuku API キー」に貼り付ければ設定完了です。



なお、定期実行タスクのジョブの終了を kintone の画面から知るすべはありません。従って、たとえばボタンを押すと [やること：**読み込み中画面を表示する**] で読み込み中と表示し、定期実行タスクのジョブが終了したら [やること：**読み込み中画面を終了する**] で読み込み中画面を消す、といったことはできません。

定期実行タスクのジョブが終わったかどうかはジョブ実行履歴画面や定期実行タスクのカスタマイズの実行履歴から確認してください。

参考までに、kintone 画面の完成例は下記となります。

3-5.kintone の Webhook の通知を介さず Customine の kintone アプリの画面のカスタマイズからダイレクトに kintone アプリの Webhook のカスタマイズを呼び出す

kintone アプリのカスタマイズで、[やること：**kintone Webhook を起動する**] を使用すると、Job Runner の kintone アプリの Webhook のカスタマイズを kintone の画面から呼び出すことができます。

この方法で Job Runner の kintone アプリの Webhook のカスタマイズを呼び出した場合、下記のような特徴があります。

・kintone の Webhook の通知を介さずに、直接 Job Runner の kintone アプリの Webhook のカスタマイズが呼び出される
⇒そのため、この方法で呼び出す場合は kintone アプリの「アプリの設定 >Webhook」での設定は必要ありません。

・kintone の Webhook の通知は、1 レコード単位で発生するため Job Runner の kintone アプリの Webhook のカスタマイズでひとつのジョブで受け取る Webhook のレコードは 1 レコードですが、この方法で呼び出した場合は複数レコードを受け取る（渡す）ことができる
⇒画面で選択した複数のレコードをまとめて Job Runner の kintone アプリの Webhook のカスタマイズに渡し、処理することができます。

下記のカスタマイズでは、一覧画面にチェックボックスを表示し、選択されたレコードを Job Runner の kintone アプリの Webhook のカスタマイズに渡しています。

[やること：**kintone Webhook を起動する**] のパラメーターは、下記のように設定します。

・kintone Webhook のカスタマイズ：
Job Runner であらかじめ作成しておいた kintone アプリの Webhook のカスタマイズを選択します。

・gusuku API キー：
4-5.kintone アプリの画面のカスタマイズから定期実行タスクをダイレクトに呼び出すと同様です

・操作の種類：
レコードに対してどのような操作を行ったとみなして kintone アプリの Webhook のカスタマイズを呼び出すのかを指定します。
これは、kintone の Webhook の設定や Job Runner の < 条件：Webhook の発生がレコードの追加によるならば > などの条件と対応していますので、詳しくはドキュメントをご確認ください。

Webhook を設定する (kintone ヘルプ)

https://jp.cybozu.help/k/ja/id/040600.html#set_webhook_webhook_10

「Webhook」（追加条件）

https://docs-customine.gusuku.io/ja/job/conditions/condition_webhook/

・アプリ：

どのアプリを Webhook の発生元とみなして kintone アプリの Webhook のカスタマイズを呼び出すのかを指定します。呼び出す kintone アプリの Webhook のカスタマイズでカスタマイズの作成時に選択した kintone アプリと同じアプリを指定する必要があります。

・レコード：

kintone アプリの Webhook のカスタマイズに渡すレコードを指定します。例では、一覧で選択したレコードを指定しています。

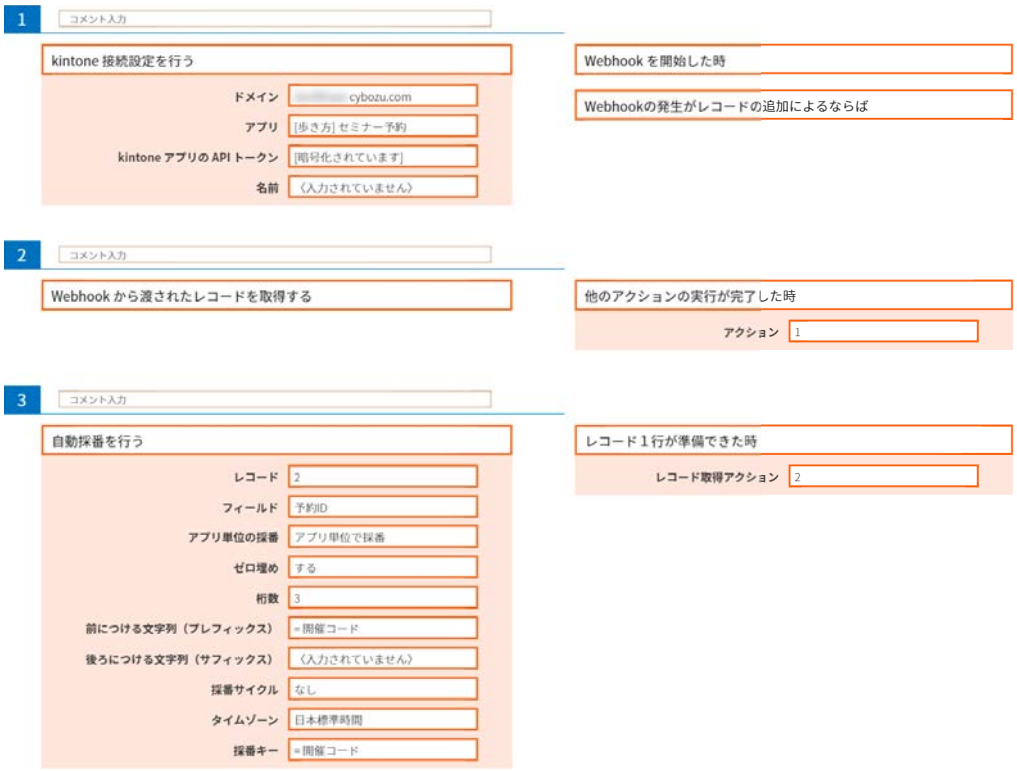
・同時実行：

同じカスタマイズから作成したジョブの同時実行を許可するかどうかを指定します。

kintone アプリのカスタマイズを登録すると、このような画面になります。

「選択したレコードを一括で自動採番する」ボタンをクリックすると、チェックボックスで選択したレコードが kintone アプリの Webhook のカスタマイズに渡され、一括で自動採番されます。

【参考】セミナー予約の Webhook のカスタマイズ



なお、定期実行タスクと同様に、kintone アプリの Webhook のカスタマイズのジョブの終了を kintone の画面から知るすべはありません。

4. うまく動かないときは？

4-1. カスタマイズは少し作っては確認、少し作っては確認の繰り返しで育てよう

カスタマインのカスタマイズは複数のアクションを組み合わせて作成することが多く、慣れてくると多くのアクションによって構成される大きなカスタマイズを、一度も kintone アプリで試して動かさずに作ってしまいがちです。

このようにして一度も実行結果を試さずに作成した大きなカスタマイズは、うまく動かなかった場合にどのアクションに原因があるかを見つけるのが非常に大変です。

複数のアクションを組み合わせたカスタマイズでは、少し作ったら実行して動きを確認する→問題があれば直す→動きを確認する→問題なければ次のカスタマイズを追加する、という手順の繰り返しで作成することをおすすめします。

この章では、うまく動かない時に原因を見つけるための方法と、便利なカスタマインの機能について紹介します。

4-2. まずは状況把握

Job Runner を使っている場合は、主に次の動きを実現したいケースが多いと思います。

・フィールド値を変更する
・レコードを追加・更新する

実行したけど「うまく動かない」と思ったということは、それらの結果が確認できなかったのだと思います。しかし、実際には処理自体は正常に完了しているものの、確認するべき場所を間違っている（思い違いをしている）というケースもよくあります。

次の点に注意して確認するようにしてください。

・アプリ
  ◦開発用・本番用に同じ内容の別アプリがある
  ◦過去のアプリを再利用するなどで、同名のアプリがある

  信用できるのはアプリ ID のみです。Job Runner で設定したアプリ ID と動作確認をしたアプリ ID が一致しているか確認しましょう

・レコード
  ◦一覧の条件指定でレコードが表示されていない
    ■ 表示しているのが、(作業者が自分) など
  ◦レコードの権限で表示されていない
    ■ Job Runner は Administrator の権限でレコード操作を動作するので、アプリを見ているログインユーザーとは別の権限

処理対象のレコードを閲覧する権限があるか、確認する方法は適切であるか確認しましょう

・フィールド
  ◦同値で更新されているが気づいていない
    ■「フィールドに値をセットする」で同値の場合は Job Runner 側で判断してレコード更新しない場合があります（例外的に更新する場合もあります）
    ■「キーの値をもとにレコードを更新する」などでは同値でもレコード更新しますが、値が変わっていないためアプリのレコード変更履歴には記載されません
  ◦更新対象ではないフィールドである
    ■ ルックアップで取得するフィールドや文字列の自動計算などは、フィールドに直接値をセットしても変わりません

これらの確認をしても結果を確認できなかった場合は、以下の点を確認してください。

・なにをしたいか整理する
  ◦なにができたら正しいかを書き出してみる
  ◦どのアプリの、どのレコードのどのフィールドが、どんな値になったら想定通りか

それらが整理できたら次のステップに移ります。

4-3. 動作確認用の便利ツールの使い方

動作確認時に便利なのは以下の 2 つです。まずはこれらの内容を確認しましょう。

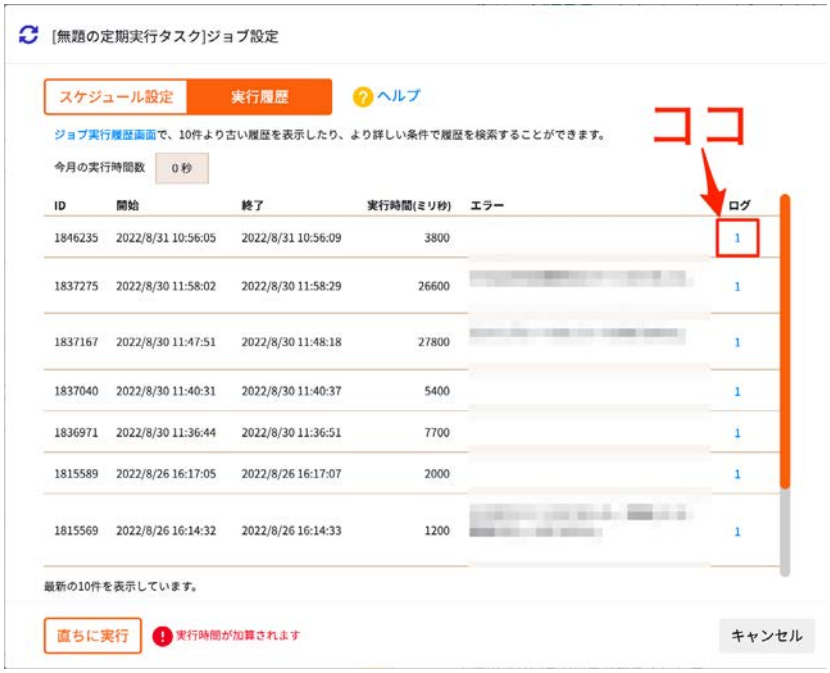
・ログ
・アクショングラフ

ログ

動作時のアクションの実行結果や処理内容などが出力されています。

「ジョブ生成・設定」内の「実行履歴」にあるログのリンクをクリックすると詳細なログが表示されます。「実行履歴」に一覧表示されている内容ではなく、リンクをクリックして表示されるログをご確認ください。

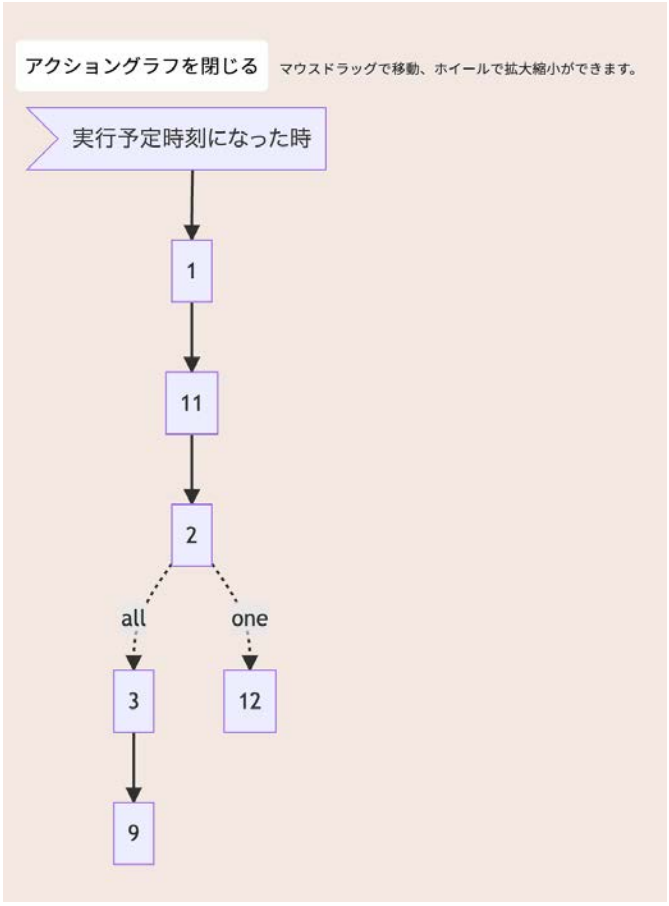
ログに表示されるエラーについては後ほど解説します。



アクショングラフ

画面右上のページメニューの中に「アクショングラフ」というボタンを押すとアクショングラフが表示されます。

アクションの繋がりがわかるので、つながっていないアクションの値を参照していたり、＜条件：レコード 1 行が準備できた時＞と＜条件：レコード全行が準備できた時＞の違いなどが確認できます。



< 条件：**レコード 1 行が準備できた時** > は one で繋がります。
< 条件：**レコード全行が準備できた時** > は all で繋がります。

　このようにツリーが分岐している場合には、9 番では 12 番の結果を利用することはできません。

　ここまでで、確認すべきポイントとその時に使うツールを紹介しました。
　これらの情報をもとに意図した動きになるように修正していきましょう

4-4. 問題の切り分け手順

　エラーが記録されている場合は、まずはそれに対処していきます。
エラー表示はないのに意図した動きではない場合は【エラーが出力されていない場合】に進んでください。

カスタマイズ作成画面にエラーが表示されている場合

　以下のようにカスタマイズ作成画面に表示されているエラーメッセージには、エラーがあるアクション番号や原因になっているアクション番号が表示されています。

```
ページ(2):アクション(17) :セット先レコード:流れのつながっていないアクションは見ることはできません。(15)
ページ(2):アクション(17) :レコード:流れのつながっていないアクションは見ることはできません。(15)
```

エラーメッセージの例

> ページ (2): アクション (17) : セット先レコード : 流れのつながっていないアクションは見ることはできません。(15)

解説

文字通り、アクションの繋がりが適切でない場合に表示されるメッセージです。
最初の「ページ (2): アクション (17) 」はエラーとなっているアクション番号です。この部分はリンクになっているので、クリックすると問題のアクションにジャンプします。
メッセージの最後の「(15)」は原因となっているアクション番号です。

このエラーの場合は、例えば、レコードを取得するアクションとフィールドに値をセットするアクションが適切に繋がっていないケースなどで発生します。
このエラーが表示されるときは、指定できないアクションを選択している場合が多いため、アクショングラフでアクションごとの繋がりを確認するとともに、正しいアクションを指定してください。

エラーメッセージの例

> ページ (2): アクション (17) : レコード : 流れのつながっていないアクションは見ることはできません。(15)

解説

「追加条件」で同じような状態になるとメッセージが少し変わります。原因が「やること」なのか「追加条件」なのかを注意してご確認ください。

エラーメッセージの例

> ページ (2): アクション (17) : フィールド : フィールドコード [文字列] がなくなっているので選択を外しました。

解説

設定後にフィールドコードが変わっている場合に表示されます。
この場合はフィールド選択ダイアログでリロードした後に、フィールドを再選択してください。

ログにエラーが出力されている場合

エラーメッセージにはエラーの原因が書かれているので、まずはその内容を確認します。不明な用語があった場合には、まずはカスタマインのサポートサイトで検索してみてください。また、kintone 自体がエラーを出しているケースもあるため、kintone のヘルプページも確認してください。

カスタマインのサポートサイト
https://support.gusuku.io/
kintone のヘルプページ
https://jp.cybozu.help/k/ja/

ここではエラーの例を紹介しながら確認ポイントを解説していきます。
() 内のある数値はアクション番号です。このアクションでエラーになっているので、まずは設定内容を確認しましょう。

エラーメッセージの例

> (3) 式がエラーになりました。[数値 _5 + 1]
> 数値が正しくありません。

解説

エラーメッセージに書いてあるように数値が正しくありません。
このエラーは、式に使用しているフィールドやアクションの値が空欄などの数値ではない場合に発生します。
まずは、[やること : **ログにメッセージを出力する**] で、式に使っているそれぞれの値を確認しましょう。
数値以外の値と確認できた場合は、必ず数値になるような修正をしてください。（フィールドに初期値を設定する、集計対象が 0 件の場合に空欄になるケースはアクションを分ける、などです）

エラーメッセージの例

> HTTP Error Code: 520
> {'code': 'GAIA_LO04', 'id': 'Ss9Ln9GAIP7bNRXMScDN', 'message': ' フィールド「既存ルックアップ」の値「A10001」が、ルックアップの参照先のフィールドにないか、またはアプリやフィールドの閲覧権限がありません。'}

解説

権限がないというエラーの場合は、「kintone 接続設定を行う」に指定している API トークンの権限が足りていないか、別のアプリの API トークンを入力しているか、「kintone 接続設定を行う」を指定していないか、がほとんどです。

ごく稀に、アプリで設定後に「アプリを更新」を押していないために、API トークンの設定がアプリに反映されていないという場合もあります。

注意点としては、ルックアップの更新の場合は更新するレコードの編集権限とルックアップコピー元のアプリの閲覧権限が必要なので、「kintone 接続設定を行う」は 2 つ指定が必要な事です。
コピー元のアプリは忘れがちなので、設定が最新で更に漏れなく設定されていることをご確認ください。

「**関連レコード一覧の条件でレコードを取得する**」を使うときはルックアップと同じように取得先アプリの閲覧権限が必要です。同様にご注意ください。

エラーメッセージの例

> [action 2] record_api.get_webhook_records starting.
> (2) Webhook から URL を渡されていません。

解説

「kintone アプリの Webhook」で作成したカスタマイズを、アプリのレコード操作ではなく Customine のジョブ設定にある「テスト実行」を押して呼び出した場合にエラーになります。
「kintone アプリの Webhook」の場合は、レコード操作で動作確認してください。

エラーメッセージの例

> (17) フィールド [文字列] が見つかりません。

解説

選択した「レコード」にないフィールドが指定されている場合のエラーです。
カスタマイズを何度も修正している場合などに、指定のレコードを変更したけどフィールドはそのままであるとか、「条件」が「**他のアクションの実行が完了した時**」である場合に発生することがあります。

・フィールドを再選択する
・「条件」を「**レコード 1 行が準備できた時**」に変更する
　などの修正対応をしてください。

エラーが出力されていない場合

ログを見てもエラーらしき出力がない場合は、順に確認を進めていきます。

アクションが最後まで実行されているか

レコード取得後のアクションで「条件」が＜条件：**レコード 1 行が準備できた時**＞や＜条件：**レコード全行が準備できた時**＞の場合は、取得したレコードが 0 件の場合は実行され**ません。**
また、アクションの繋ぎ先を間違えている場合も、意図したアクションが実行されないことがあります。
この場合はアクショングラフでアクションの繋がりを確認してください。

アクションはレコード数だけ実行されているか

キーを指定してレコードを取得した時など、想定外のレコード数が取得されることがあります。
その場合、同一フィールドに何回も更新するような動きになることがあり、その時に想定していない値で更新されるケースがあります。どのレコードを元に処理をして、どのレコード（のフィールド）を更新するのかをご確認ください。

レコード更新のデータが出力されているか

更新前の値と更新後の値が同値の場合など、更新が不要な場合にはデータは作成されません。

カスタマイズを実行した結果レコードの更新がある場合は、
ログの最後に表示されている
ended.
の前に、
BEGIN PUT
で始まる行があります。

レコード更新されていないという場合は、以下の例を参考にこのデータが作成されているかをご確認ください。

更新がない場合

```
2022-09-27T03:32:08.600Z [action 17] field_api.set_field_value done.
2022-09-27T03:32:08.600Z ended.
```

「**値が変更されているレコードを保存する**」のように処理の途中でレコード更新がある場合

```
2022-09-22T10:51:40.563Z [action 10] record_api.save_dirty_record starting.
2022-09-22T10:51:40.563Z BEGIN PUT https://r3it-dev.cybozu.com/k/v1/records.
json
2022-09-22T10:51:40.563Z {'app': 287, 'records': [{'id': '1309', 'revision': -1, 'record':
{'$id': {'value': '1309'}, 'Job_Runner ステータス ': {'type': 'SINGLE_LINE_TEXT', 'value': ' 処
理完了 '}}}]}
2022-09-22T10:51:40.808Z END    PUT https://r3it-dev.cybozu.com/k/v1/records.
json
2022-09-22T10:51:40.809Z [action 10] record_api.save_dirty_record done.
2022-09-22T10:51:40.809Z ended.
```

処理完了時に値が変更されたフィールドがあった場合

```
022-09-21T00:49:46.172Z [action 11] table_api.add_table_row done.
2022-09-21T00:49:46.173Z BEGIN PUT https://r3it-dev.cybozu.com/k/guest/26/
v1/records.json
2022-09-21T00:49:46.173Z {'app': 1239, 'records': [{'id': '13', 'revision': -1, 'record':
{'$id': {'value': '13'}, ' テーブル _7': {'type': 'SUBTABLE', 'value': [{'value': {' 単価 ': {'type':
'SINGLE_LINE_TEXT', 'value': None}, ' 数値 ': {'type': 'NUMBER', 'value': None}, ' 画像 ':
{'type': 'FILE', 'value': []}}}]}, ' テーブル _4': {'type': 'SUBTABLE', 'value': [{'value': {' 行番号 ':
{'type': 'NUMBER', 'value': ''}}}, {'value': {' 行番号 ': {'type': 'NUMBER', 'value': ''}}}, {'value':
{' 行番号 ': {'type': 'NUMBER', 'value': ''}}}, {'value': {' 行番号 ': {'type': 'NUMBER', 'value':
''}}}}}}}}}
2022-09-21T00:49:46.321Z END    PUT https://r3it-dev.cybozu.com/k/guest/26/v1/
records.json
2022-09-21T00:49:46.322Z ended.
```

4-5. テスト用アプリ作成

　カスタマイズをガンガン試してと言われても、既にユーザーが使っている kintone アプリ (以下 本番用アプリ) にカスタマイズすると、誤ったレコードを更新してしまうリスクがあったり、フィールドの値やステータスなど、カスタマイズを試したい状態のレコードがないなど実行確認が難しいことがよくあります。

　そういった時のために、実際にユーザーが使うアプリとは別に、そのアプリをコピーした**テスト用アプリ**を作成することをおすすめします。


gusuku
CUSTOMINE
Job Runner 編