# 目次

# 改訂情報

| 変更年月日 | 変更内容 |
| --- | --- |
| 2016-02-01 | 初版 |
| 2016-04-01 | 第2版 下記を追加・変更しました。<br>■ 「*応用編 - より高度なフロー*」へ以下の新規チュートリアルを追加。<br>　■ 「*複雑なフローの定義*」-「*変数を利用したフロー*」を追加。<br>　■ 「*ユーザ定義の作成*」-「*SELECTを用いたユーザ定義（SQL）の作成*」に「*応用：取得範囲を指定したユーザ定義（SQL）の作成*」を追加。<br>　■ 「*ユーザ定義の作成*」-「*参考：より高度なREST APIの呼び出し*」に「*JSONによる複雑なレスポンスを受け取る*」を追加。<br>　■ 「*ロジックフローのデバッグ*」を追加。<br>　■ 「*ロジックフローのエラーハンドリング*」を追加。<br>■ 「*付録*」へ以下の付録事項を追加。<br>　■ 「*作成したロジックフローの設計書出力*」を追加。 |
| 2016-08-01 | 第3版 下記を追加・変更しました。<br>■ 「*フローカテゴリを作成する*」の画像を差し替え。<br>■ 「*フロールーティングを設定する*」の画像を差し替え。<br>■ 「*暗黙的な変数の利用*」-「*セッション情報*」に新しいプロパティを追加。<br>■ 「*応用編 - より高度なフロー*」へ以下の新規チュートリアルを追加。<br>　■ 「*ユーザ定義の作成*」-「*ユーザ定義 - CSV Fetch*」を追加。<br>　■ 「*ユーザ定義の作成*」-「*ユーザ定義 - Database Fetch*」を追加。<br>　■ 「*フロートリガを利用する*」を追加。<br>■ 「*付録*」へ以下の付録事項を追加。<br>　■ 「*作成したロジックフローの設計書出力*」へ新規に追加された項目を追記。<br>　■ 「*チュートリアルデータのアーカイブファイル*」に新規チュートリアルのアーカイブファイルを追加。 |
| 2017-04-01 | 第4版 下記を追加・変更しました。<br>■ 「*応用編 - より高度なフロー*」へ以下の新規チュートリアルを追加。<br>　■ 「*ユーザ定義の作成*」-「*ユーザ定義 - テンプレート定義*」を追加。<br>　■ 「*柔軟なマッピング情報の定義*」-「*マッピングのデバッグ*」を追加。<br>　■ 「*ユーザ定義の作成*」-「*独自アイコンの追加方法*」を追加。 |
| 2017-08-01 | 第5版 下記を追加・変更しました。<br>■ 「*応用編 - より高度なフロー*」へ以下の新規チュートリアルを追加。<br>　■ 「*ユーザ定義の作成*」-「*ユーザ定義 - Excel入力*」を追加。<br>　■ 「*ユーザ定義の作成*」-「*ユーザ定義 - Excel出力*」を追加。 |
| 2017-12-01 | 第6版 下記を追加・変更しました。<br>■ 「*階層化された入力値・出力値の定義（Object型の定義）*」-「*階層化された値のマッピング*」にマッピングの仕様について説明を追加。<br>■ 「*応用編 - より高度なフロー*」へ以下の新規チュートリアルを追加。<br>　■ 「*ユーザ定義の作成*」-「*ユーザ定義 - XML解析*」を追加。<br>　■ 「*ユーザ定義の作成*」-「*ユーザ定義 - HTML解析*」を追加。<br>■ 「*暗黙的な変数の利用*」-「*セッション情報*」に baseUrl を追加。<br>■ 「*暗黙的な変数の利用*」-「*外部ユーザコンテキスト*」を追加。 |

| 変更年月日 | 変更内容 |
| --- | --- |
| 2018-04-01 | 第7版 下記を追加・変更しました。<br>■ 「*応用編 - より高度なフロー*」へ以下の新規チュートリアルを追加。<br>　■ 「*ユーザ定義の作成*」-「*ユーザ定義 - BIS申請/承認*」を追加。 |
| 2020-12-01 | 第8版 下記を追加・変更しました。<br>■ 「*応用編 - より高度なフロー*」 - 「*ユーザ定義の作成*」 - 「*ユーザ定義 - BIS申請/承認*」へ新規に追加された項目を追記。 |
| 2021-04-01 | 第9版 下記を追加・変更しました。<br>■ IM-LogicDesignerの「ロジックフロー定義一覧」画面の項目名変更に伴いドキュメント内の文言、および、キャプチャを変更 |

# はじめに

## 本書の目的

本書は、IM-LogicDesignerを利用してビジネスロジックの開発を初める開発者のみなさまの支援を目的としたドキュメントです。

## 対象読者

本書では次の開発者を対象としています。

- IM-LogicDesignerによる開発の一連の流れを知りたい
- IM-LogicDesignerを利用してビジネスロジックを開発したい

なお、本書では次の内容を理解していることが必須です。

- intra-mart Accel Platformを理解している。

また、次のドキュメントを読了していると、より理解が深まります。

- IM-LogicDesigner仕様書

## サンプルコードについて

本書に掲載されているサンプルコードは可読性を重視しており、性能面や保守性といった観点において必ずしも適切な実装ではありません。
開発においてサンプルコードを参考にされる場合には、上記について十分に注意してください。

## 本書の構成

本書は次の章で構成されています。

- *概要*

  本書、および、IM-LogicDesignerの概要について説明します。

- *基礎編 - ファースト・ステップ*

  基礎編として、シンプルなビジネスロジックの実装を行うチュートリアルです。
  このチュートリアルを通していただくことで、IM-LogicDesignerを利用する上で必要な作業の流れや、機能についてご理解いただけます。

- *応用編 - より高度なフロー*

  応用編として、基礎編のロジックを元に、より複雑なロジックを実装する方法を説明します。

- *付録*

  基礎編、応用編では触れていない補足事項です。

## IM-LogicDesignerとは

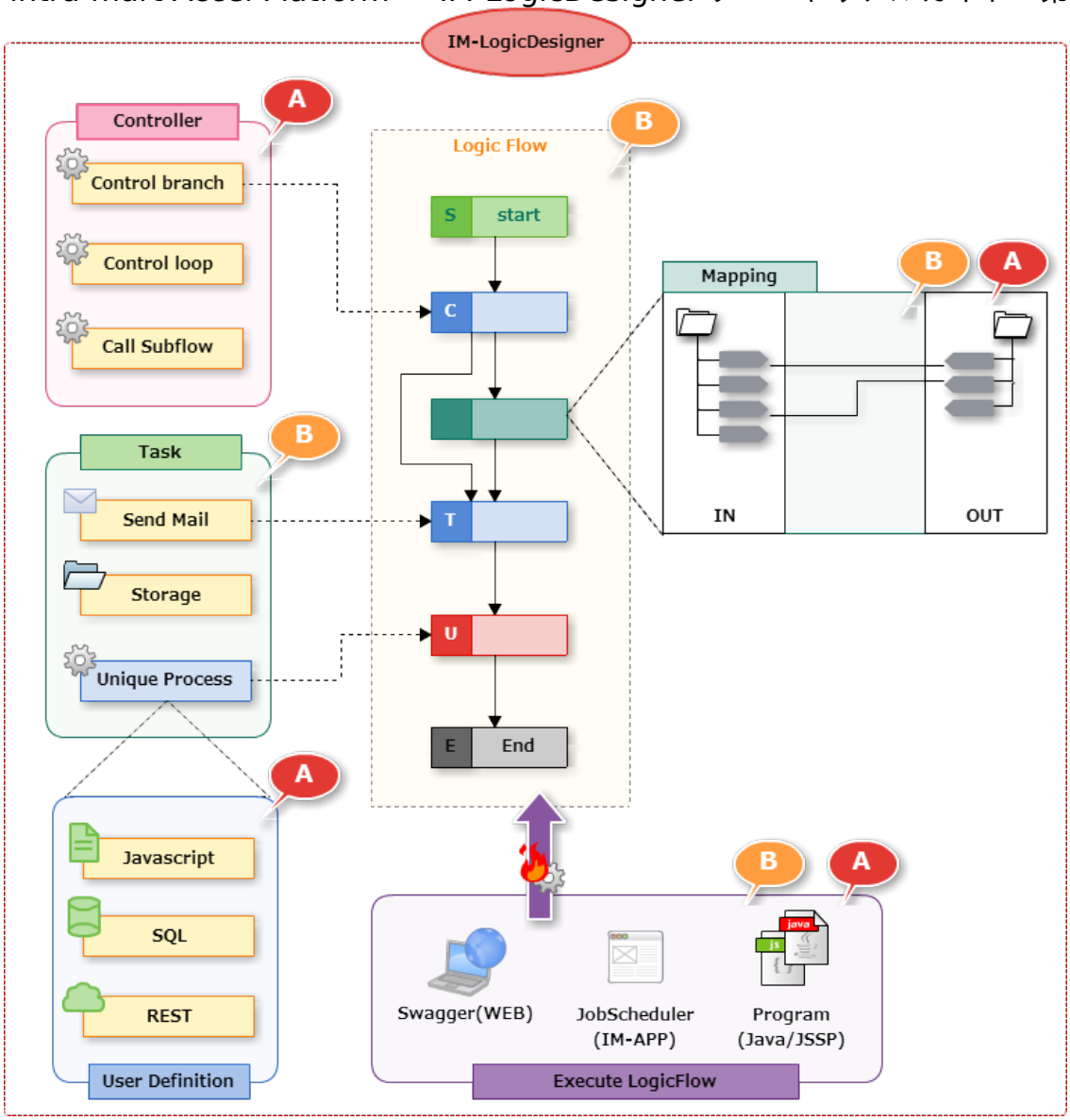
IM-LogicDesignerについて、ここでは「IM-LogicDesigner仕様書」 - 「概要」から一部引用して説明します。

> IM-LogicDesigner仕様書 - 3.1 IM-LogicDesignerとは
>
> IM-LogicDesignerとは、intra-mart Accel Platform上でビジネスロジックを簡単に作成することができるアプリケーションです。
> IM-LogicDesignerの特徴は以下の通りです。
>
> - プログラミングの知識がない人でもGUI上で処理を簡単に作成することができます。
> - 面倒なデータの変換や受け渡しなどは、IM-LogicDesignerが全て自動で行います。
> - Web画面上のみで、SQLや独自処理をサーバサイドJavaScriptで作成、定義でき、ビジネスロジックで利用することが可能です。
> - IM-LogicDesignerで作成したビジネスロジックは、以下に挙げるアプリケーションなどから呼び出すことできます。
>   - IM-BIS for Accel Platform
>   - IM-FormaDesigner for Accel Platform
>   - ジョブスケジューラ
> - TERASOLUNA Global Framework、スクリプト開発モデルから、IM-LogicDesignerで作成した独自のビジネスロジックを直接呼び出すことができます。
> - ビジネスロジックはREST APIとして利用することができるため、外部から呼び出すことも可能です。また、ビジネスロジック内で、外部のREST APIを呼び出すことも可能です。 さらに、REST APIに対しての認可、セキュリティ設定も行うことができるため、API GATEWAY機能として利用することが可能です。

本チュートリアルガイドは上記概要をベースとして、簡単なロジックフローの作成方法から、一歩先の利用方法まで、実際に開発者の皆様へ紹介します。

## IM-LogicDesignerの全体像と、本チュートリアルガイドの説明範囲

IM-LogicDesignerの全体像と、各機能に対する本チュートリアルガイドの説明範囲は以下の通りです。




図：全体像と説明範囲

- 「**B(Basic)**」の吹き出し - 「*基礎編 - ファースト・ステップ*」にて説明を行っている箇所です。
  一例として以下の内容を説明しています。
  - ロジックフローの定義方法
  - 標準で提供されるタスクの利用方法
  - マッピング
  - ロジックフローの実行方法
- 「**A(Advanced, Appendix)**」の吹き出し - 「*応用編 - より高度なフロー*」、および、「*付録*」にて説明を行っている箇所
  一例として以下の内容を説明しています。
  - 分岐や繰り返しといった制御処理の利用方法
  - ユーザ定義の作成、および、利用方法
  - 柔軟なマッピング情報の定義方法
  - ジョブや外部プログラムからのロジックフローの実行方法

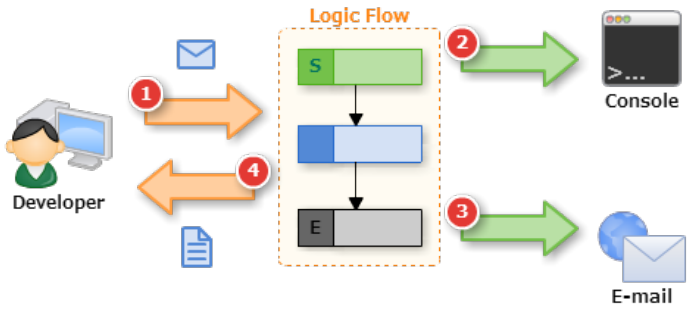
## チュートリアルの概要（作成物のイメージ）

本チュートリアルでは、

**「開発者の設定した入力情報を、コンソールへのログ、および、メールに発信し、その結果を出力情報として返す」**

という処理の実装を通して、IM-LogicDesignerの基本的な操作方法や作業の流れを説明します。

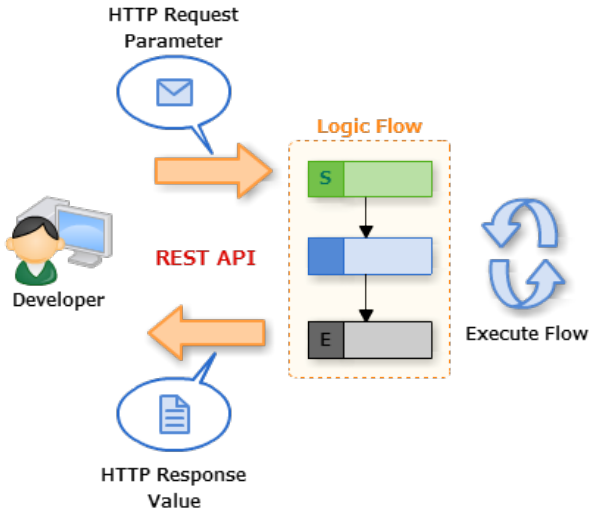
作成する処理の動作イメージは以下の通りです。




図： 基礎編 チュートリアル動作概要図

1. ロジックフロー実行者が、発信したいメッセージとともにフローを実行する。
2. メッセージを元にコンソールへログとして出力する。
3. メッセージを元にメールをロジックフロー実行者へ送信する。
4. ロジックフローの実行結果を、ロジックフロー実行者へ返却する。

本チュートリアルで作成したロジックフローの実行には、IM-LogicDesignerが備えるロジックフローをREST APIとして扱う機能を利用します。
上記動作概要における、フローの実行、および、実行結果の返却の詳細は以下の通りです。




図：基礎編 チュートリアルフロー実行概要図

## 事前準備

本チュートリアルを進めるにあたり、以下の事前準備が行われていることが前提です。

### 環境セットアップ

本チュートリアルを進める上で必要なintra-mart Accel Platformの環境セットアップ情報は以下の通りです。

1. 以下のモジュールを含んだ形で、intra-mart Accel Platformのwarファイルを作成していること
   - IM-LogicDesignerモジュール
   - IM-共通マスタモジュール
2. intra-mart Accel Platformのテナント環境セットアップが完了していること

3. サンプルデータのインポートが完了していること

また以下の事前準備は任意です。開発環境に合わせて準備が可能な場合、実施してください。

1. 以下のモジュールを含んだ形で、intra-mart Accel Platformのwarファイルを作成していること
   - メールモジュール
2. メール設定が完了しており、メールの送信が可能であること。

> **コラム**
>
> メール設定について
>
> メール設定の詳細は「設定ファイルリファレンス」 - 「メール設定」を参照してください。

### チュートリアル実行ユーザ

本チュートリアルでは、全て「テナント管理者」ロールを持つユーザで実施します。
必要に応じて、ロールの付与を実施してください。

## フローカテゴリを作成する

ここから本格的なチュートリアルを開始します。
一番初めに、今回作成するロジックフローの分類情報として、フローカテゴリを作成します。



### ロジックフローカテゴリを新規作成する

本チュートリアルで利用するフローカテゴリを作成します。
フローカテゴリの作成は、ロジックフロー定義一覧画面から行います。

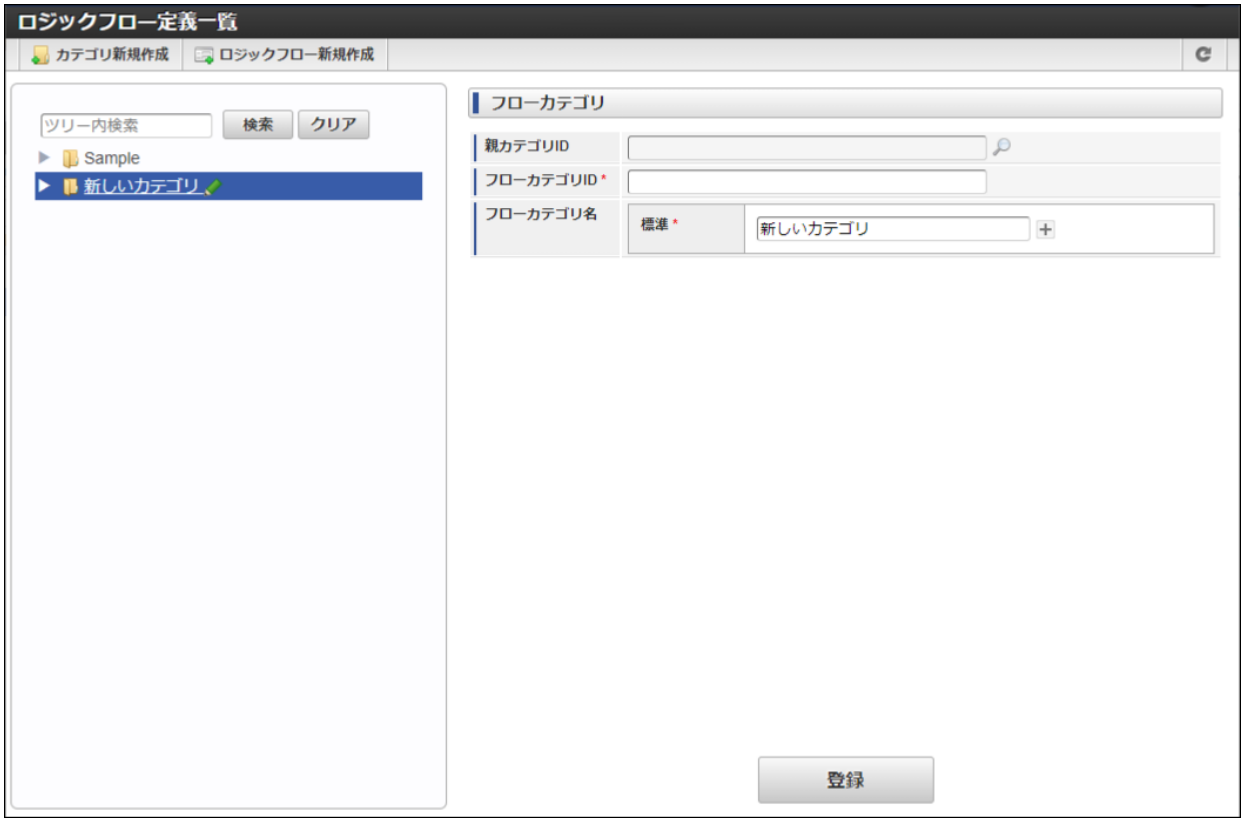
1. 「サイトマップ」→「LogicDesigner」→「フロー定義一覧」から、ロジックフロー定義一覧を開きます。


図：サイトマップ

2. ロジックフロー定義一覧画面左上の「カテゴリ新規作成」をクリックします。

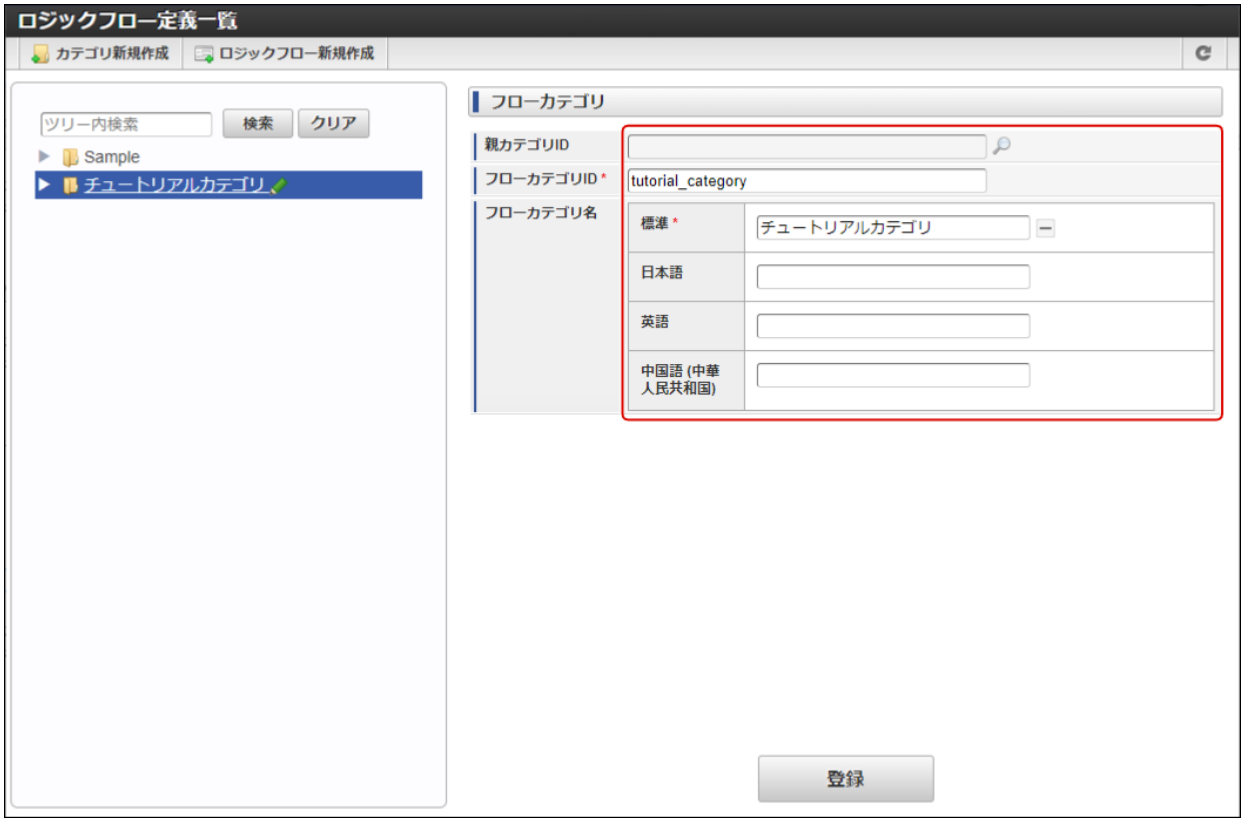

図：ロジックフロー定義一覧画面 - カテゴリ新規作成リンク

3. ロジックフローカテゴリ編集エリアが表示されます。




図：ロジックフロー定義一覧画面 - フローカテゴリ情報入力

4. フローカテゴリ情報の各項目に以下の値を入力します。
   - 親カテゴリID「（指定なし）」
   - フローカテゴリID「tutorial_category」
   - フローカテゴリ名
     - 標準 -「チュートリアルカテゴリ」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし

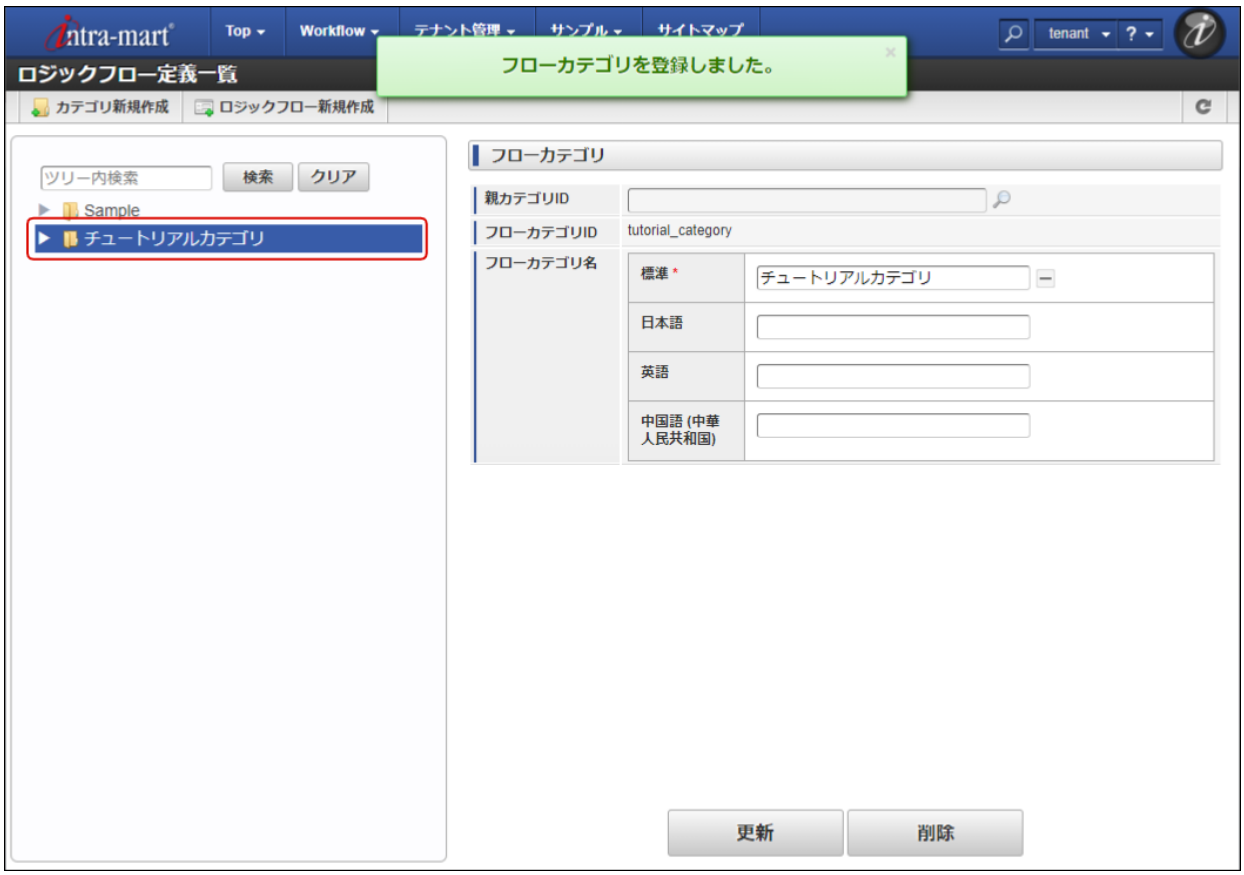



図：ロジックフロー定義一覧画面 - フローカテゴリ情報入力

5. 「登録」をクリックします。

### 作成したフローカテゴリを確認する

正常にフローカテゴリが新規作成された場合、正常に作成された旨のメッセージが表示されます。




図：フローカテゴリ作成後の一覧画面

次章「*ロジックフロー定義編集画面を開く*」では、ロジックフローの作成を行う編集画面を見ていきます。

## ロジックフロー定義編集画面を開く

次に、ロジックフローを作成するための編集画面を確認していきます。

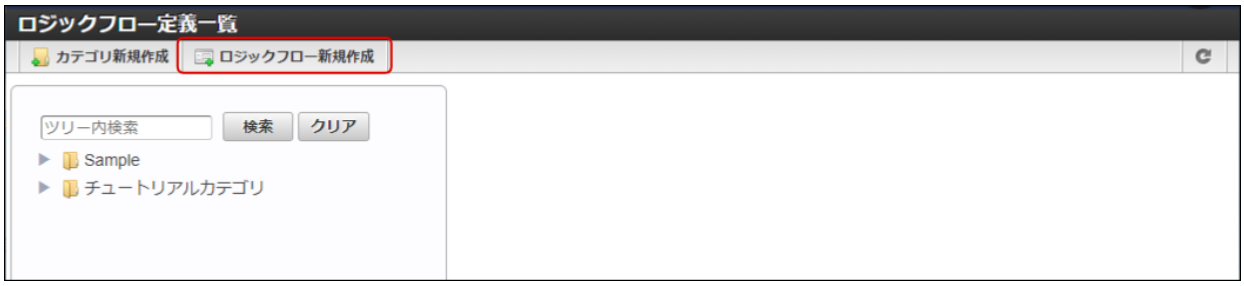
- ロジックフロー定義編集画面を開く
- ロジックフロー定義編集画面の詳細

### ロジックフロー定義編集画面を開く

1. 「サイトマップ」→「LogicDesigner」→「フロー定義一覧」から、ロジックフロー定義一覧を開きます。


図：サイトマップ

2. 「ロジックフロー定義一覧」画面左上の「ロジックフロー新規作成」をクリックします。




図：ロジックフロー定義一覧画面

3. 「ロジックフロー定義編集」画面が表示されます。

## ロジックフロー定義編集画面の詳細

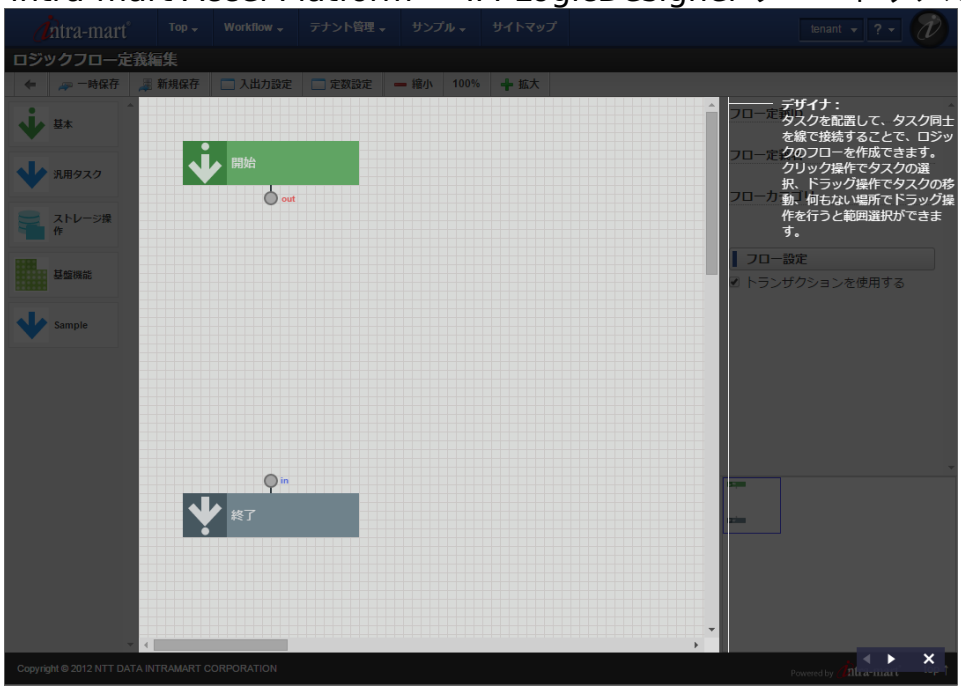
「ロジックフロー定義編集」画面は、用途に応じて複数のペイン（区画）に分かれています。
各ペインの詳細は以下の通りです。


図：ロジックフロー定義編集画面 - 初期表示

1. ロジックフローに対する基本的な操作を提供するヘッダです。
   一時保存や新規保存といったフローに対するアクションや、各種値の設定・フロー編集画面の拡大・縮小といったフロー内の設定などが行えます。
2. ロジックフローを構成するエレメント一覧です。この一覧から利用する制御要素やタスクを選択し、フロー上に配置します。
3. エレメントを配置しロジックフローを構成するフロー編集画面です。
   フロー編集画面には、予めフローの開始（緑色のコンポーネント）と、終了（灰色のコンポーネント）が配置された状態で表示されます。
4. フロー編集画面の全体図を表します。青い枠に囲まれた部分が現在の描写範囲で、ドラッグすることで描写範囲を変更できます。

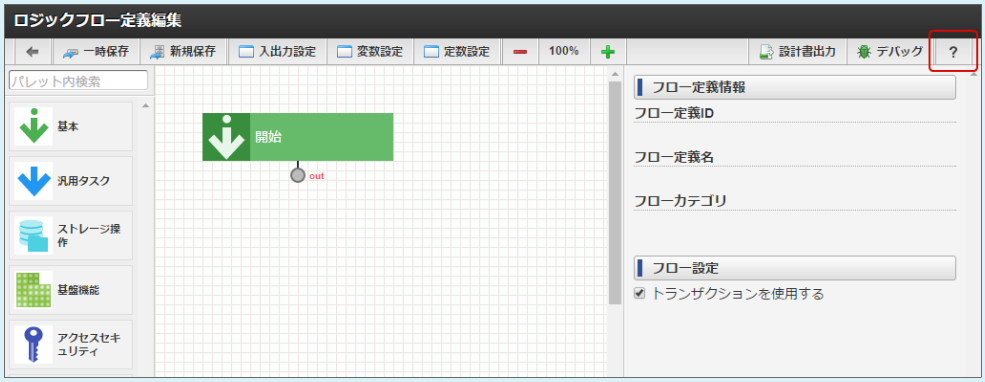
### 参考：ロジックフロー定義編集画面のサイトツアー

「ロジックフロー定義編集」画面には、サイトツアーによるナビゲーションが含まれています。
本ドキュメントと共に、合わせて確認してください。




図：ロジックフロー定義編集画面 - サイトツアー

> **コラム**
>
> サイトツアー機能について
>
> サイトツアーの詳細は「一般ユーザ操作ガイド」-「サイトツアーを呼び出す」を参照してください。

> **コラム**
>
> 2016 Spring(Maxima)以降でのサイトツアーの開始方法
>
> IM-LogicDesignerでは2016 Spring(Maxima)以降、「ロジックフロー定義編集」画面のレイアウト変更を行いました。
> そのため、サイトツアーの開始は以下のとおり、グローバルナビからでなく「ロジックフロー定義編集」画面から行ってください。
>
> - サイトツアーの開始 - ロジックフロー定義編集画面のヘッダ内、画面右の「?」をクリックする。
>
> 
> 
>
> 図：サイトツアーの開始

次章「*入出力設定を定義する*」では、作成するロジックフローの入力値と出力値を設定します。

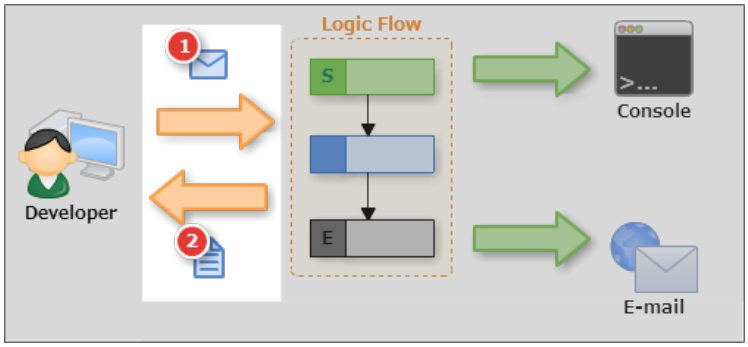
## 入出力設定を定義する

次に、作成するロジックフローの入力値/出力値の設定を行います。

- 入出力設定について
- 入出力設定画面を開く
- 入出力設定を行う

### 入出力設定について

ロジックフローの入出力設定では、作成したロジックフローを実際に呼び出す際の入力値と、呼び出しが完了した際に返却する出力値を定義します。
これは「*チュートリアルの概要（作成物のイメージ）*」で提示した動作イメージの以下にあたります。



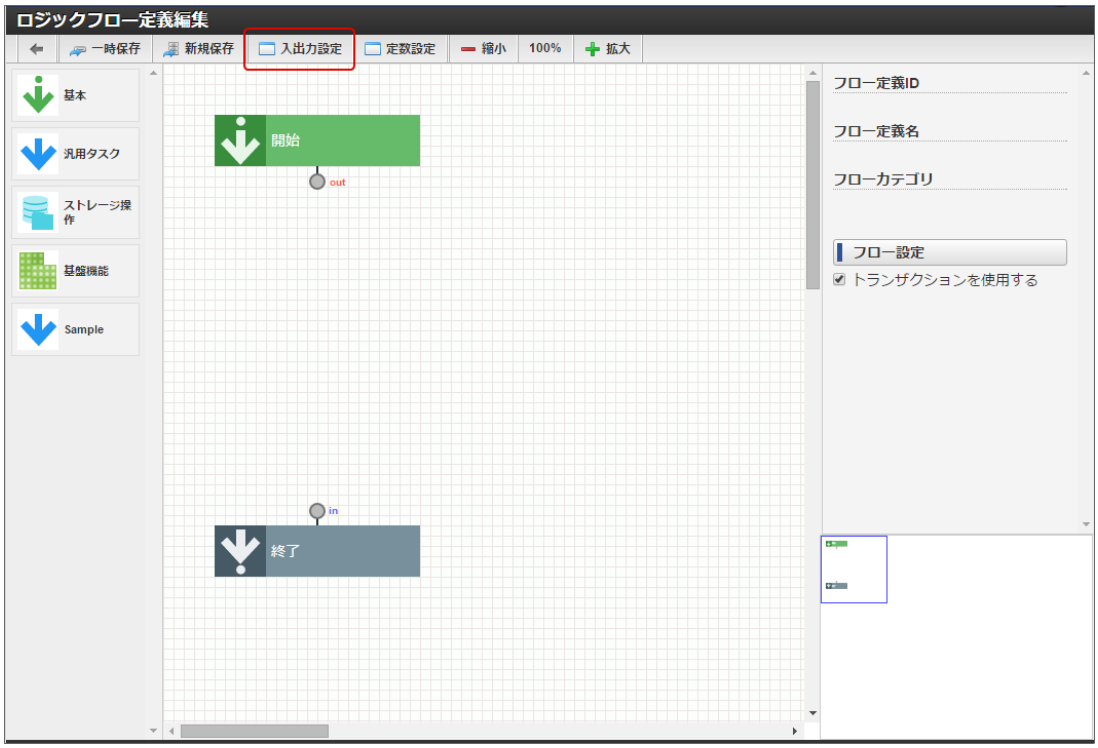

図：入出力設定

1. 入力値設定によって設定された内容で呼び出しを行う。
   - 例： REST APIのリクエストパラメータに入力値として設定して呼び出しを行う。
2. 出力値設定によって設定された内容が返却される。
   - 例： REST APIのレスポンスに設定した出力が返ってくる。

## 入出力設定画面を開く

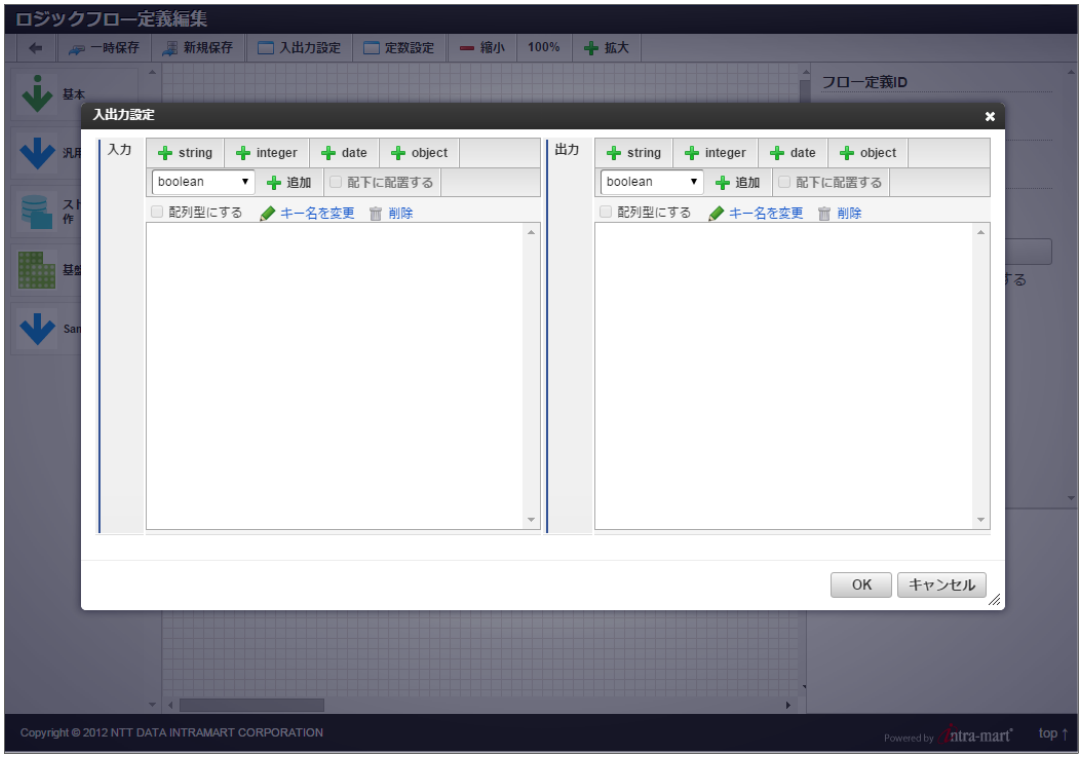
入出力設定画面は、ロジックフロー定義編集画面から開きます。

1. ロジックフロー定義編集画面上部、ヘッダ内の「入出力設定」をクリックします。




図：ロジックフロー定義編集画面ヘッダ

2. 「入出力設定」画面が表示されます。



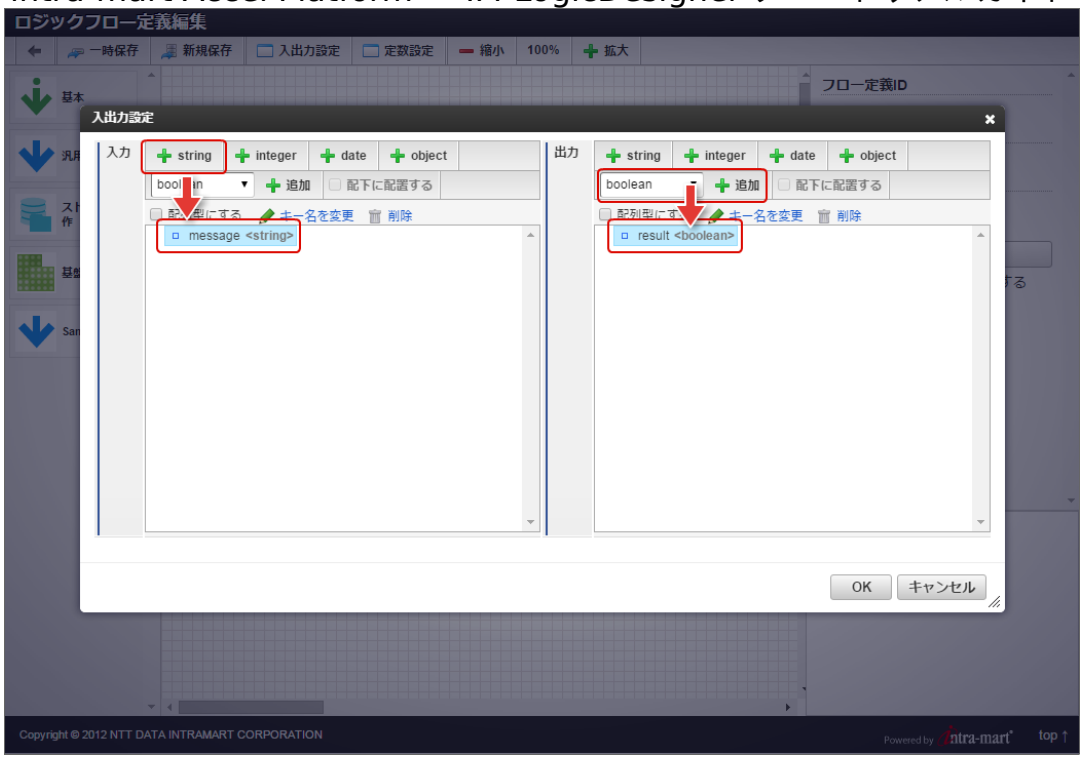

図：入出力設定画面

## 入出力設定を行う

本チュートリアルで作成するロジックフローの入出力設定を行います。
今回定義する入出力の詳細は以下の通りです。

- 入力値
  - 概要
    - 今回実装するビジネスロジックにおける「コンソールへのログ」、および、「メール」の本文となる文章を入力値として定義します。
  - パラメータ名
    - `message`
  - 型
    - `string`
- 出力値
  - 概要
    - 今回実装するビジネスロジックが正常に実行し終えたことを表すフラグを出力値として定義します。
  - パラメータ名
    - `result`
  - 型
    - `boolean`

定義内容をもとに、入出力設定画面で実際に設定を行います。




図：入力値、および、出力値の設定

- 入力値の設定
    1. 入出力設定画面の、入力ペイン上部にある「**+string**」をクリックします。
    2. 入力ペイン下部に新しく値が設定されたのを確認し、名称を「message」とします。
- 出力値の設定
    1. 入出力設定画面の、出力ペイン上部にあるセレクトボックスから「**boolean**」を選択します。
    2. セレクトボックス右にある、「**+追加**」をクリックします。
    3. 出力ペイン下部に新しく値が設定されたのを確認し、名称を「result」とします。

入出力の設定が完了したら、入出力設定画面右下のOKをクリックします。

以上で、ロジックフローへの入出力設定が完了しました。
なお、ロジックフローの入出力設定はフローの編集中でも変更可能です。

次章「*エレメントを配置する*」では、実際の処理を表すエレメントを配置していきます。

## エレメントを配置する

次に、フローにエレメントを配置し、実際の処理を定義します。



### エレメントとは

エレメントとはロジックフローを構成する各処理の総称で、フロー上では長方形のコンポーネントとして表現されます。
ロジックフロー定義編集画面を開いた際、初めから配置されていた「開始」、および、「終了」もエレメントの一つです。

エレメントはその性質により「制御要素」「タスク」の2つに更に分類されます。
初期から配置されている「開始」、および、「終了」は制御要素に分類され、これから配置する「コンソールへのログを出力する」、および、「メールを送信する」というエレメントはどちらもタスクに分類されます。

エレメント、制御要素、タスクの詳細については「IM-LogicDesigner仕様書」を参照してください。

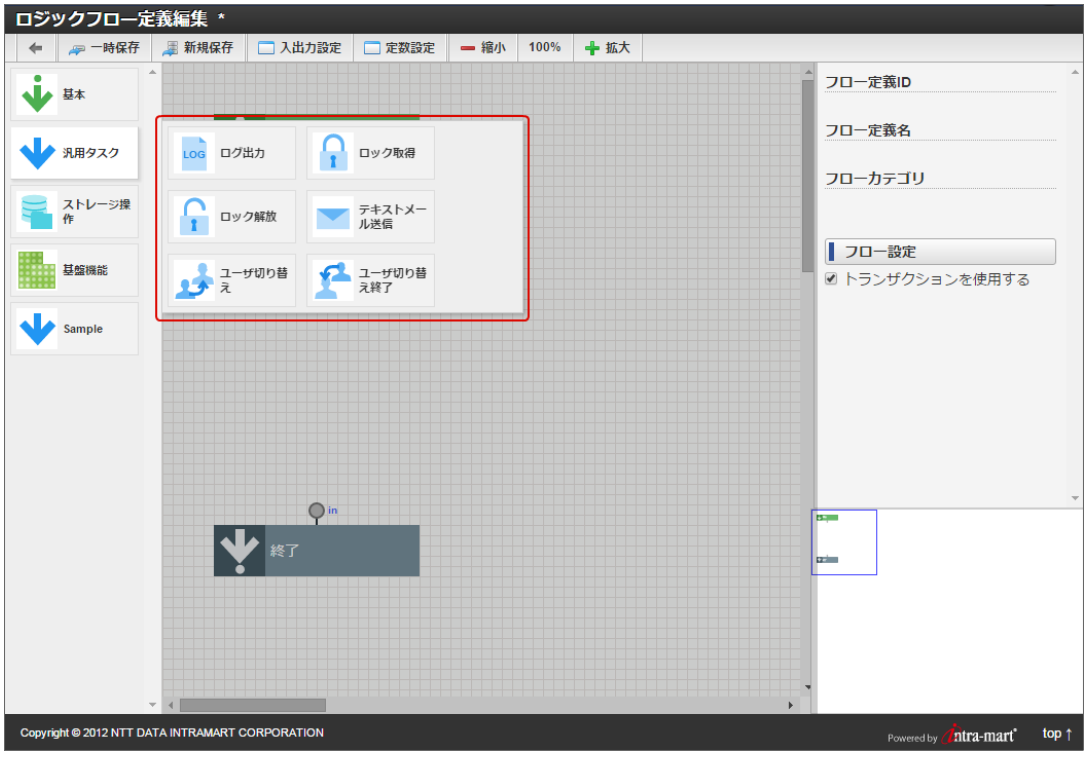
### エレメントを配置する

本チュートリアルで作成するロジックフローが実行する処理を、タスクとして配置します。
タスクはロジックフロー定義編集画面左部にあるパレットから選択することで、配置できます。

1. ロジックフロー定義編集画面左部、パレット内の「汎用タスク」へカーソルを合わせます。

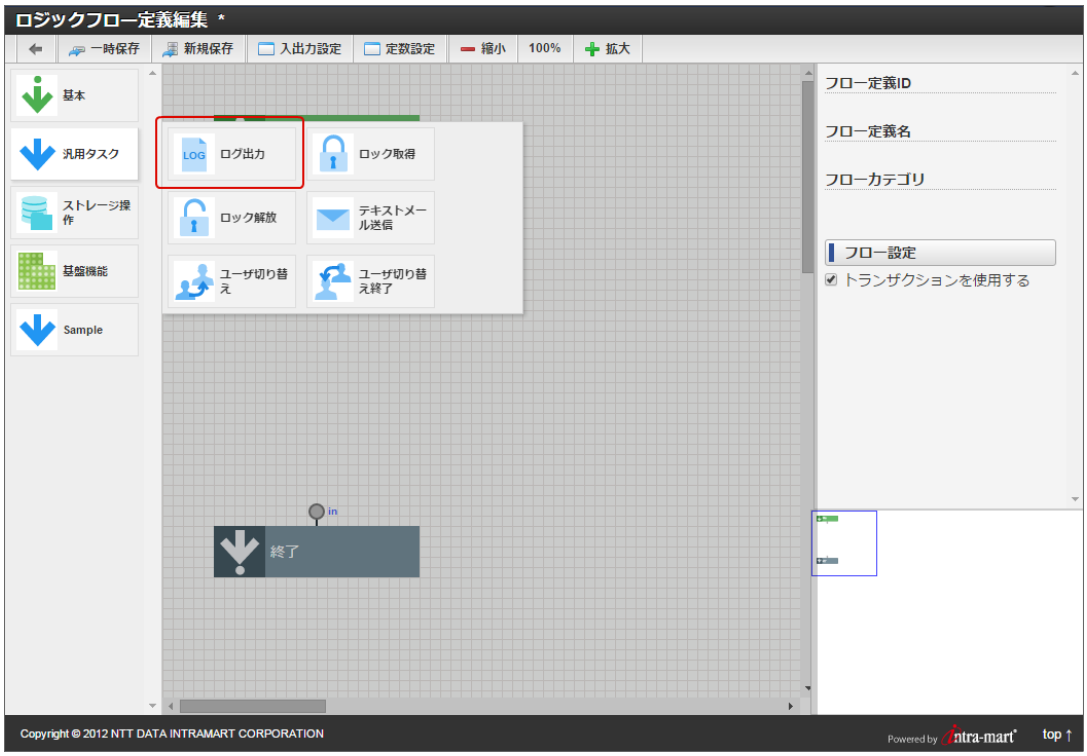

図：パレット内の「汎用タスク」

2. 「汎用タスク」にカテゴライズされるタスク一覧がスライドインします。



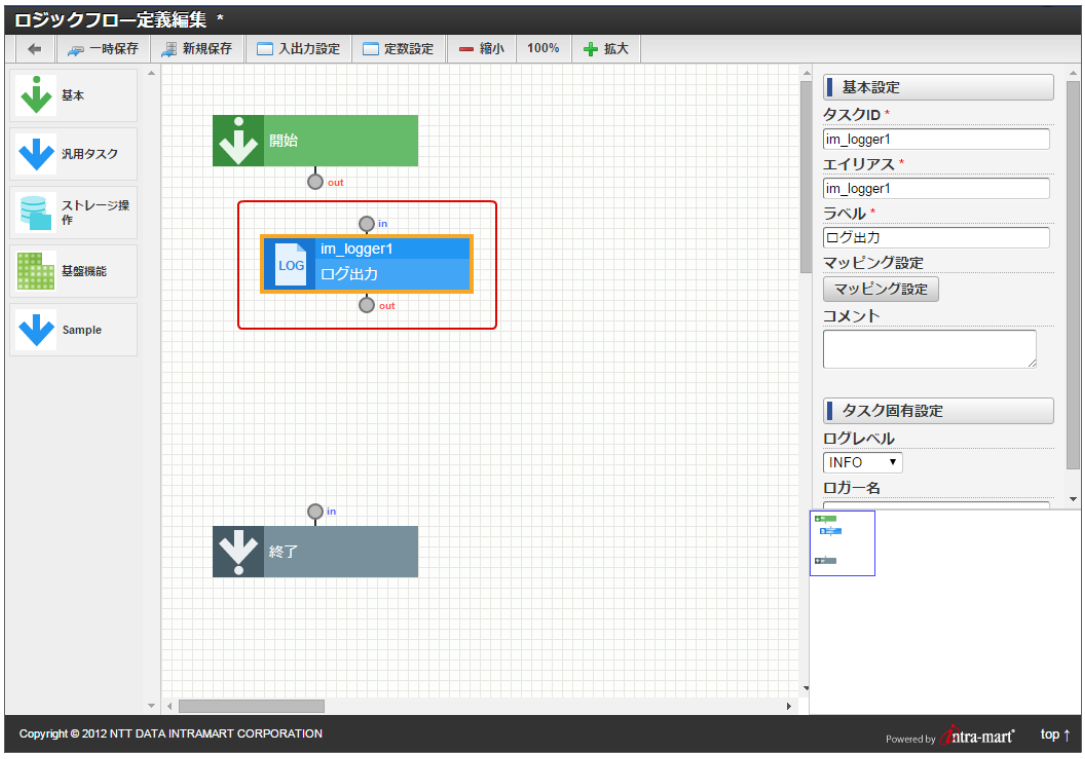

図：「汎用タスク」タスク一覧

3. スライドインした汎用タスクの一覧から、「ログ出力」をクリックします。



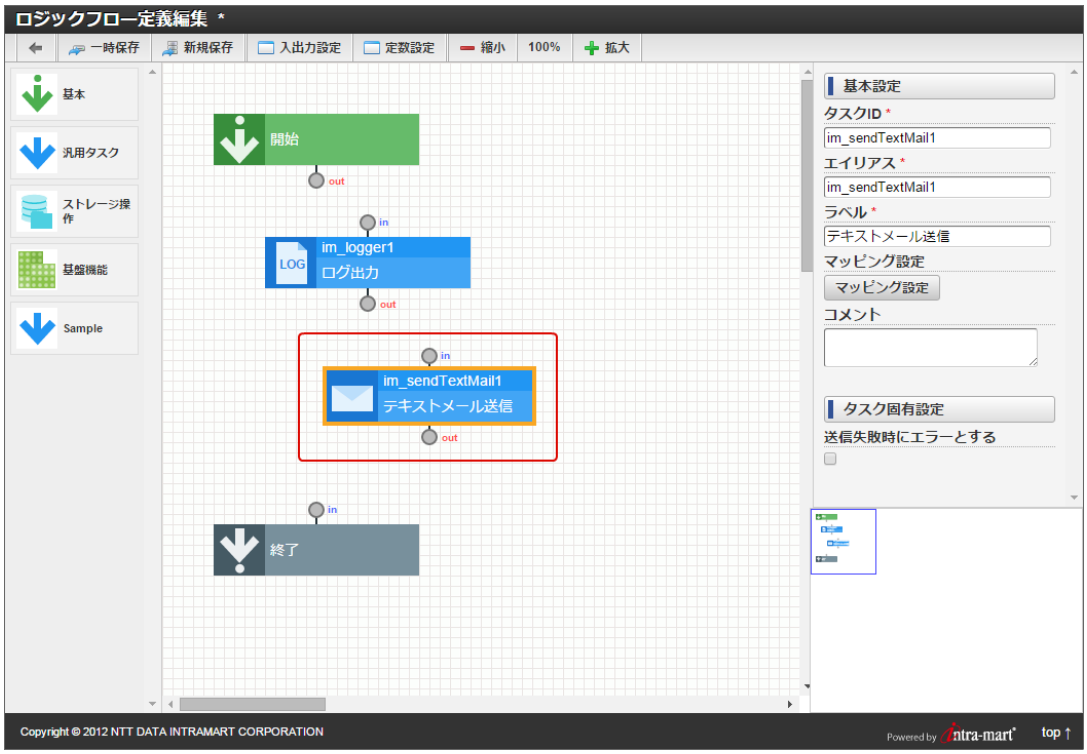

図：「ログ出力」タスクの選択

4. ログ出力を行うタスクがフロー編集画面上に追加されます。




図：タスクの配置

5. 同様に汎用タスクの一覧から、「テキストメール送信」をクリックし、フロー編集画面上に追加します。




図：「テキストメール送信」タスクの配置

以上で、ロジックフローへのエレメントの配置が完了しました。

次章「*線を引く（シーケンスを定義する）*」では、配置したエレメント同士を繋げ、処理の流れを定義します。

# 線を引く（シーケンスを定義する）

次に、配置したエレメント同士を繋げ、ビジネスロジックの処理の流れを定義します。



## 線を引く（シーケンスを定義する）

IM-LogicDesignerでは、処理の流れをエレメント同士を線で繋ぐことで定義します。
開発者は利用したい処理（エレメント）を配置し、それぞれをつなぎ合わせるだけで、任意の処理の流れを作ることが可能です。

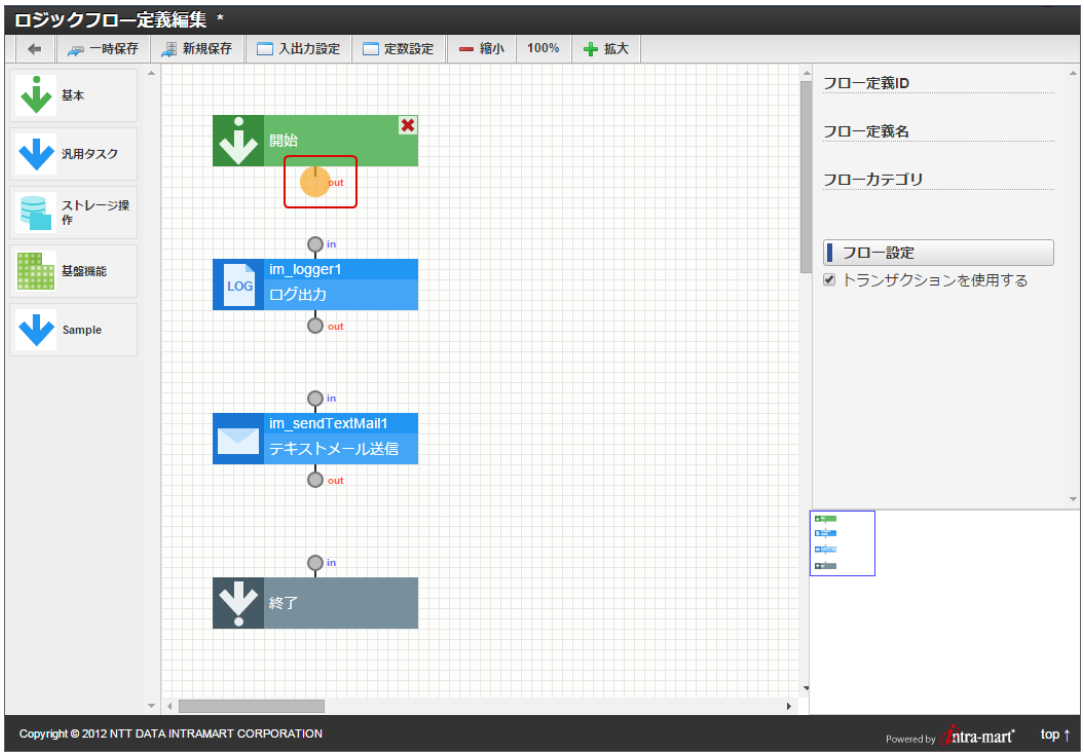
> **コラム**
>
> 線について
>
> IM-LogicDesignerでは、エレメント同士を繋ぐ線を「シーケンス」と呼びます。
> シーケンスには本章で上げたエレメント間での処理順序以外にも、いくつかの情報が定義されています。
>
> シーケンスの詳細は「IM-LogicDesigner仕様書」を参照してください。

本チュートリアルでは処理順を以下のように定義します。

1. 開始
2. ログ出力
3. メール送信
4. 終了

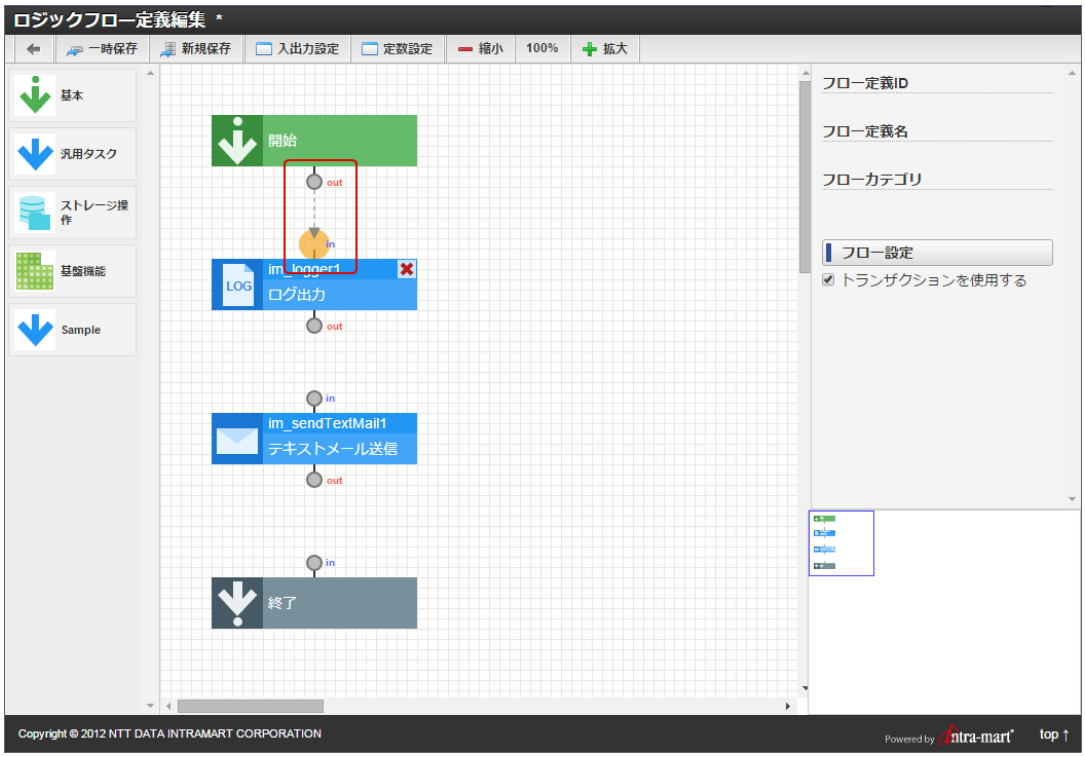
この処理順を元にフロー編集画面で線を引きます。

1. 「開始」の下部にある`out`と書かれた端子にカーソルをあわせます。



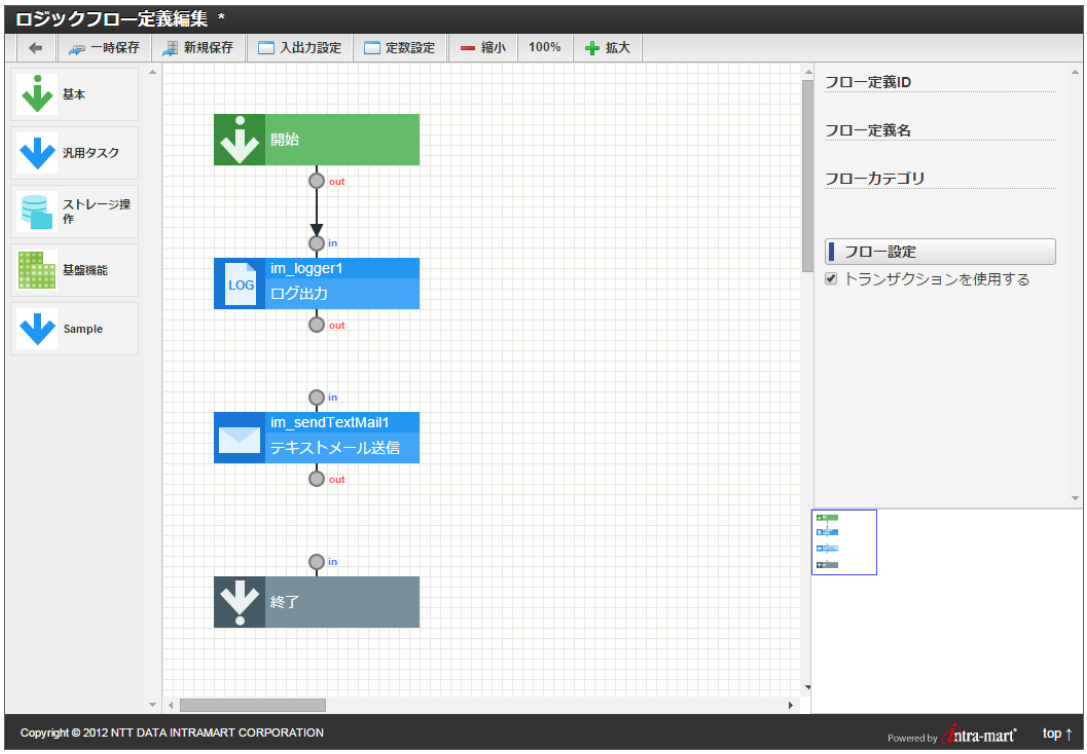

図：「開始」下部のout端子

2. そのままドラッグし、「ログ出力」の上部にあるinと書かれた端子にドロップします。



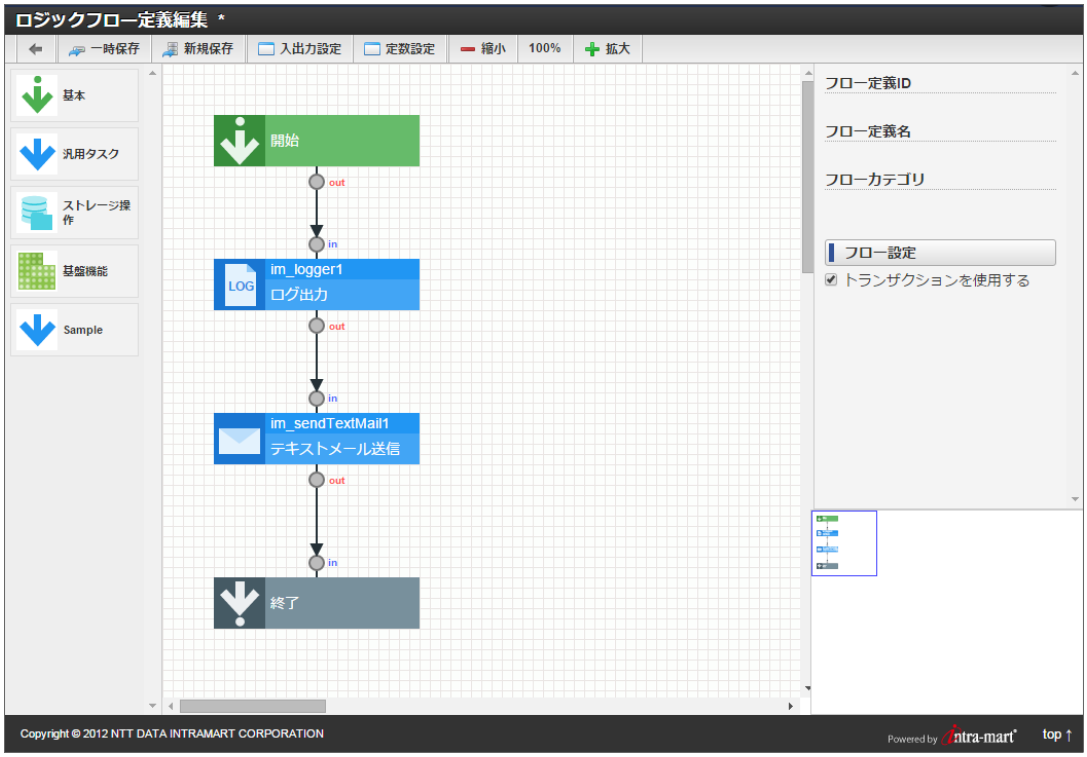

図：「ログ出力」上部のin端子

3. 「開始」と「ログ出力」が繋がったことが確認できます。




図：「開始」と「ログ出力」の接続

4. 同様の手順で、以下のエレメント間に線を引きます。
   - 「ログ出力」から「テキストメール送信」
   - 「テキストメール送信」から「終了」




図：全てのエレメントの接続

> **コラム**
>
> エレメントの移動
>
> 線を引く際、エレメント間が狭い場合や重なっている場合、エレメントをドラッグ&ドロップすることで移動することができます。

コラム

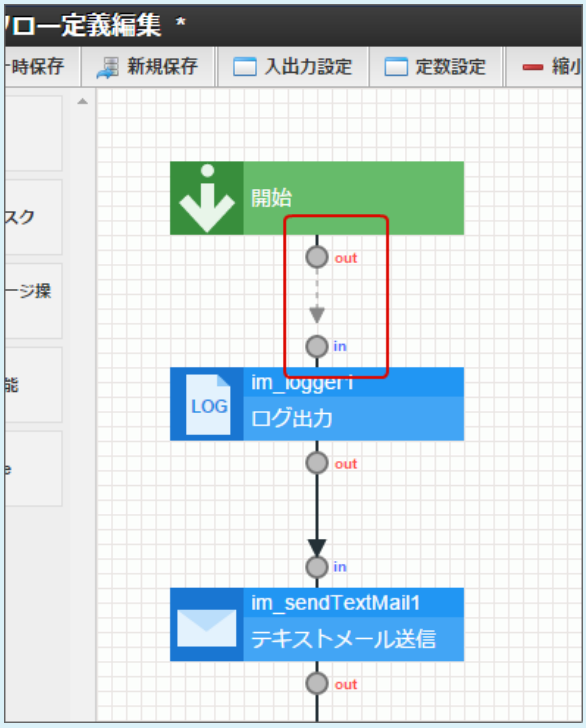
線の引き始め、および、引き終わり

各エレメントの端子にカーソルを合わせた際、線の引き始め、および、引き終わりが可能な場合に端子がオレンジ色にハイライトされます。

コラム

有効な線の定義

引いた線が正しくエレメント間を接続している場合、線は黒色で表されます。
逆に、引いた線が途中で途切れているなど、正しくエレメント間を接続できていない場合、線は灰色で表されます。




図：正しく接続できていない例

以上で、エレメント間に処理の流れを表す線を引くことができました。

次章「*一時保存する*」では、これまで定義した内容を一時保存する方法を説明します。

## 一時保存する

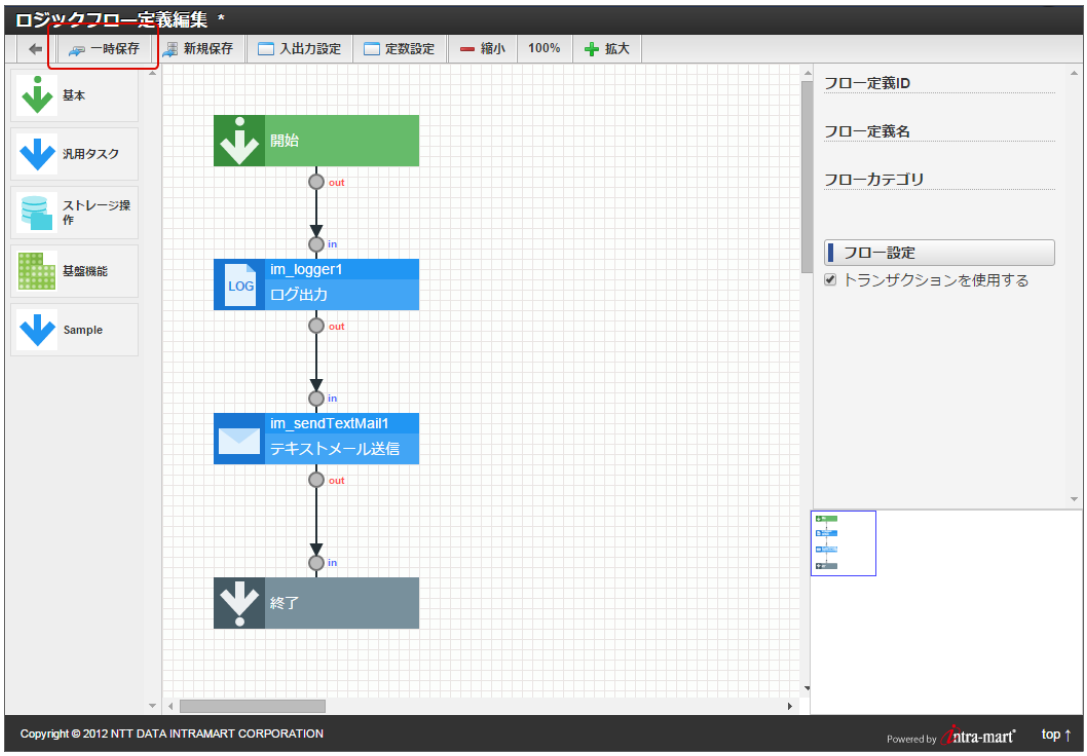
次に、これまで実施してきた内容を一時的に保存する方法を説明します。



### 一時保存する

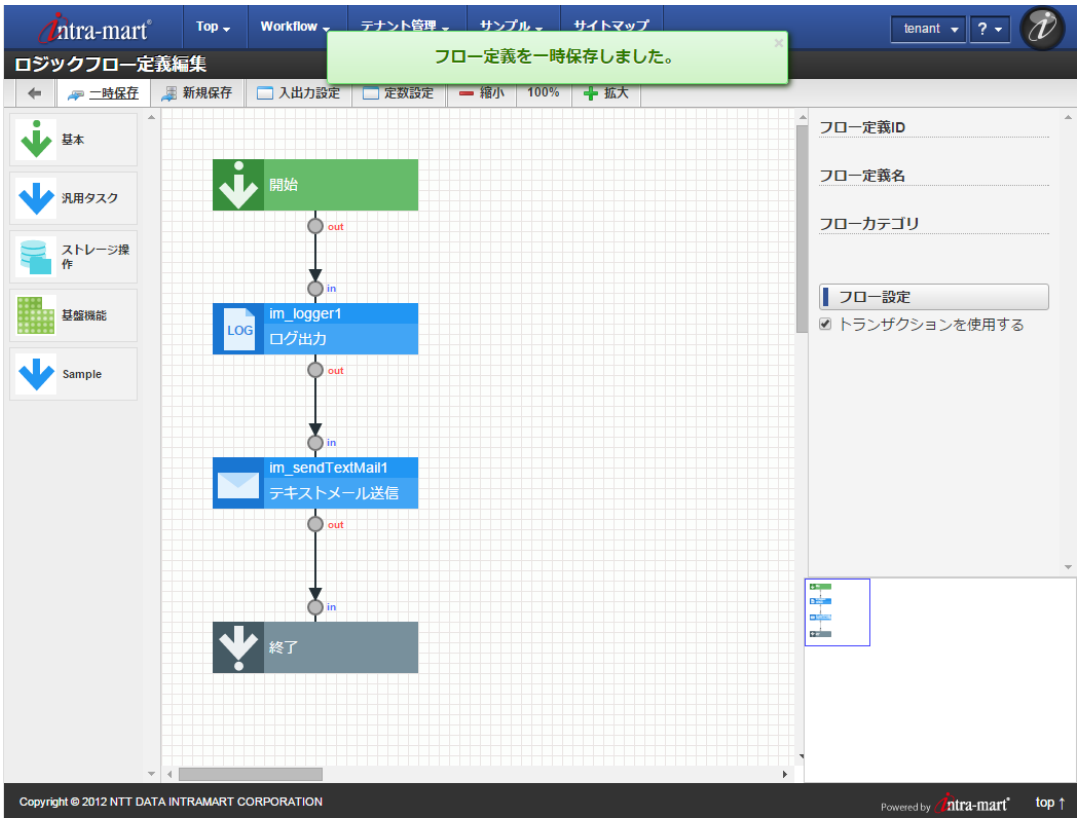
ロジックフロー定義編集画面では、編集内容を一時的に保存する事ができます。

1. ロジックフロー定義編集画面上部、ヘッダ内の「一時保存」をクリックします。




図：ロジックフロー定義編集画面ヘッダ

2. 正常に保存が行われた場合、その旨のメッセージが表示されます。




図：一時保存成功のメッセージ

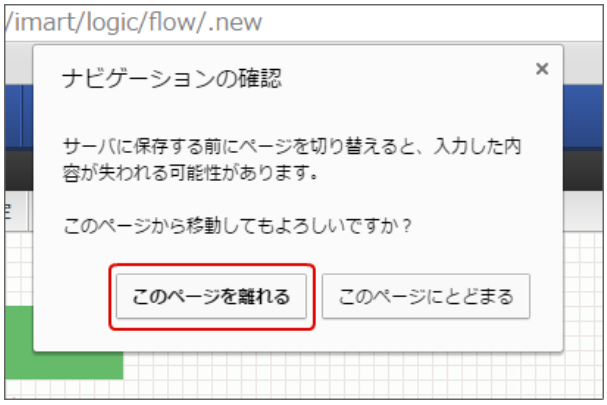
保存される範囲は、編集画面で行った全ての作業内容です。

> **注意**
>
> 一時保存可能な数
>
> ロジックフロー定義編集画面の一時保存機能が保存できる編集内容は、一度に１つまでです。
> 既に一時保存が行われた状態で再度一時保存を行った場合、古い内容は破棄されます。

## 一時保存された情報をロードする

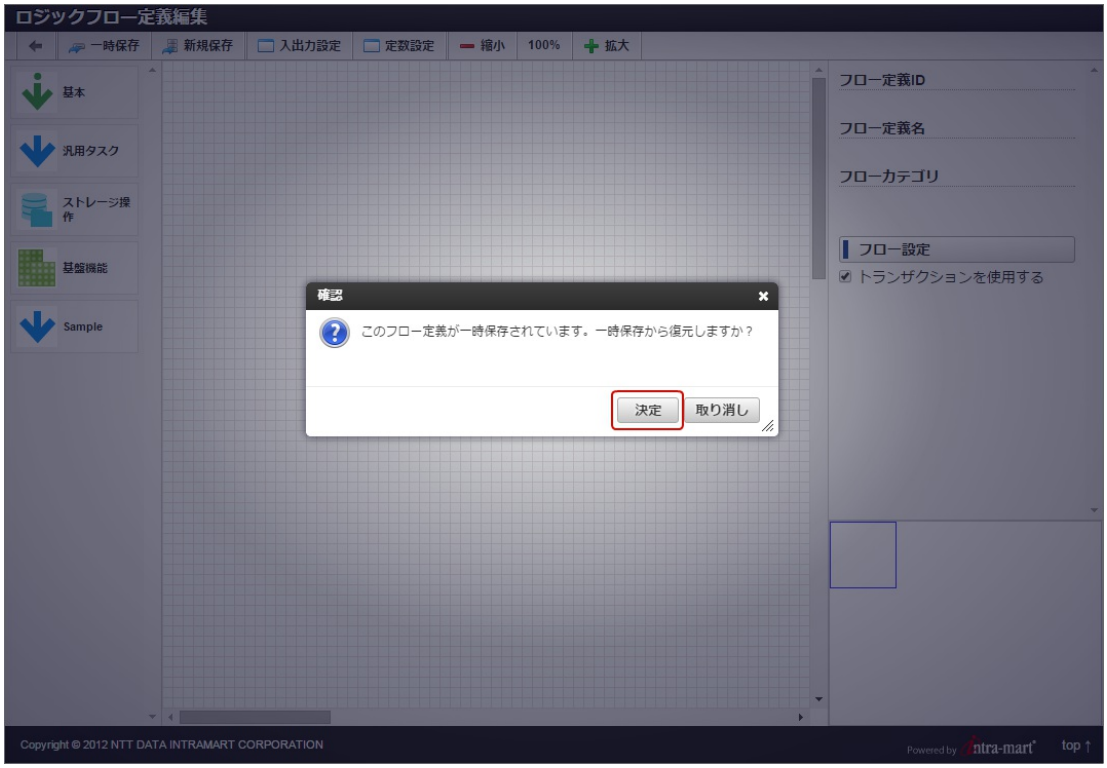
一時保存された情報は、新規作成開始時にロードすることができます。

1. 一時保存を行った後、一度ロジックフロー定義編集画面を閉じます
2. ページを閉じる際、ページ遷移の可否を問うダイアログが出た場合は、「遷移する」に該当する方をクリックしてください
   （画像はGoogle Chromeの例です）



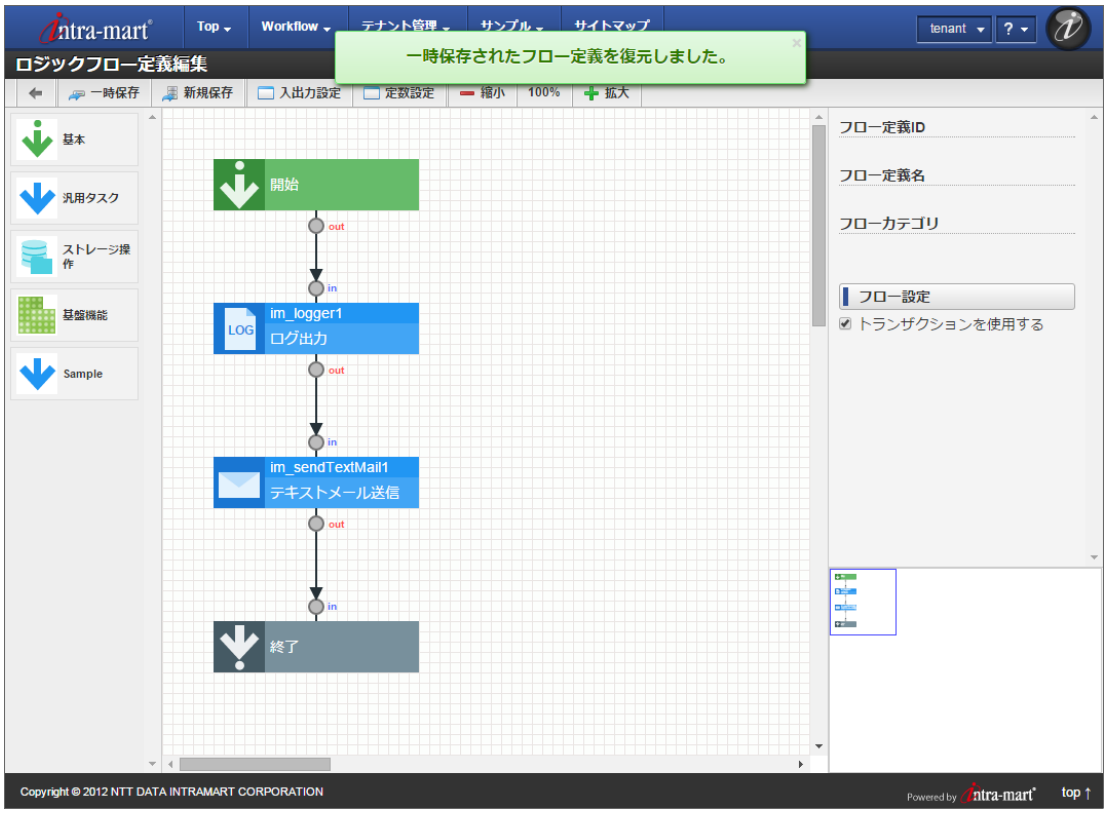

図：ページ遷移の際に表示されるダイアログ（Google Chrome）

3. 「サイトマップ」→「LogicDesigner」→「フロー定義一覧」から、ロジックフロー定義一覧を開き、定義一覧からロジックフロー新規作成をクリックし、ロジックフロー定義編集画面を開きます。
4. 一時保存から復元するかを問うダイアログが表示されるので、「決定」をクリックします。




図：復元確認ダイアログ

5. 一時保存した内容がロードされます。




図：保存内容のロード

> **注意**
>
> 一時保存された情報の有効範囲
>
> 一時保存機能によって保存された情報は、保存を行ったブラウザ単位で保持されます。
> そのため、Google Chromeで一時保存した情報は、Internet Explorerからロードすることはできません。

次章「*タスクのプロパティを設定する*」では、「ログ出力」、および、「テキストメール送信」タスクにプロパティを設定します。

## タスクのプロパティを設定する

次に、これまで配置してきた「ログ出力」、および、「テキストメールの送信」タスクに、プロパティを設定します。

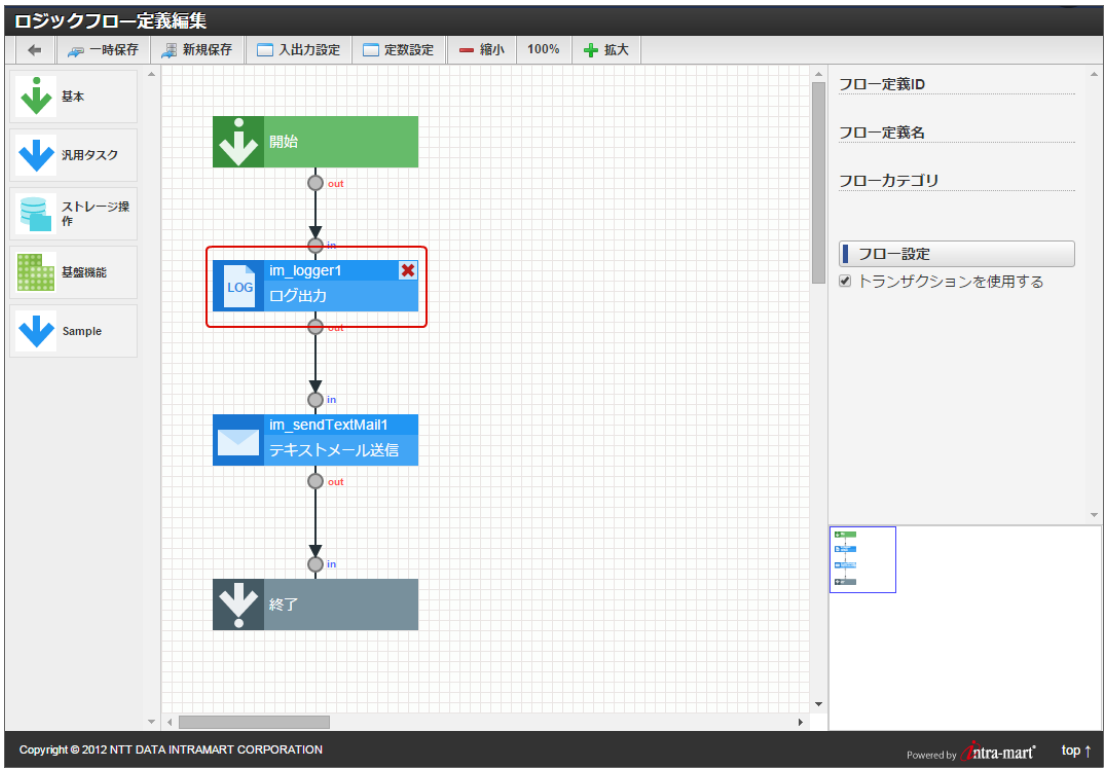


### プロパティを設定する

プロパティは、フロー編集画面上の設定を行いたいエレメントをクリックすることで、右側に選択したエレメントの処理内容にあわせたプロパティ入力画面が表示されます。
ここでは「ログ出力」タスク、および、「テキストメール送信」タスクに対して設定を行います。

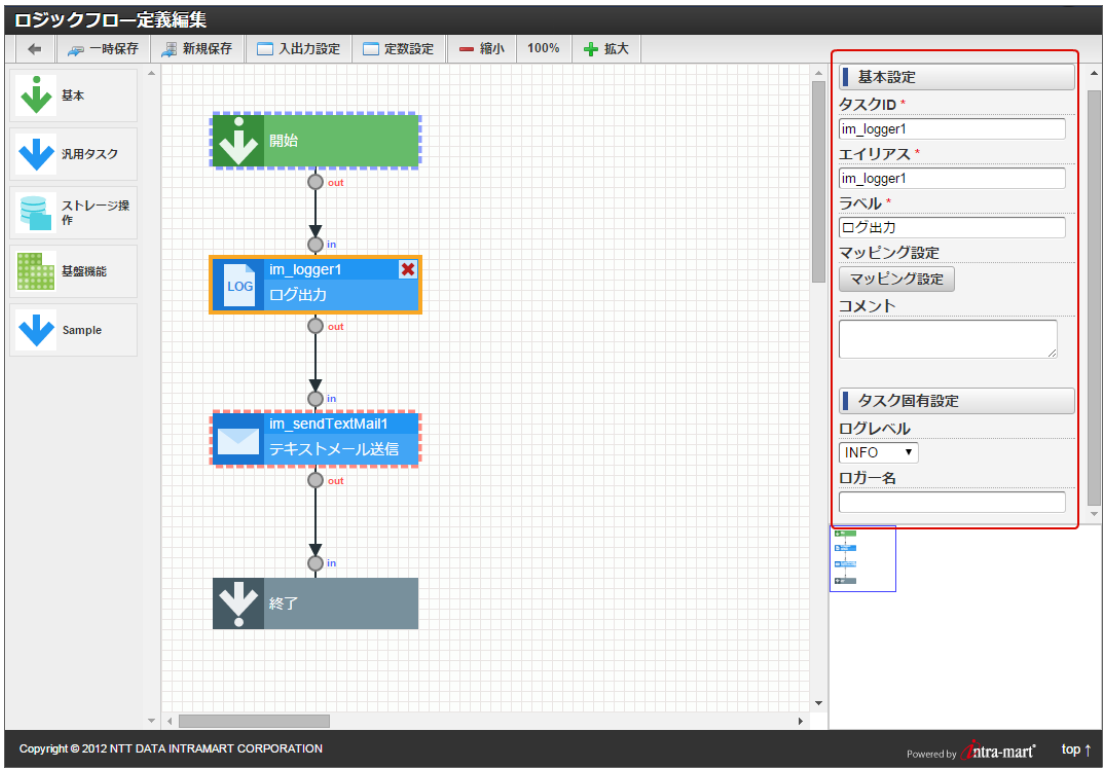
はじめに、「ログ出力」タスクのプロパティを設定します。

1. フロー編集画面上の「ログ出力」タスクをクリックします。



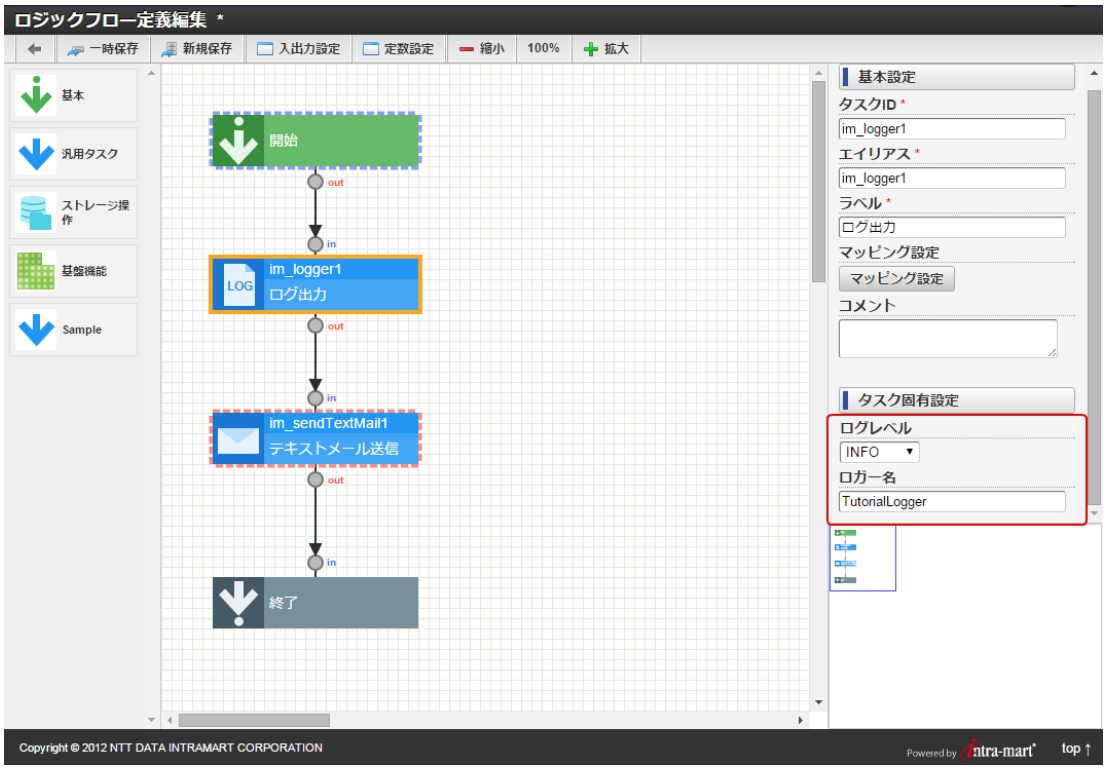

図：「ログ出力」タスク

2. 画面右側に「ログ出力」タスクに関するプロパティ画面が表示されます。




図：「ログ出力」タスクのプロパティ画面

3. タスク固有設定の項目を以下のとおりに変更します。
   - ログレベル - INFO
   - ロガー名 - TutorialLogger

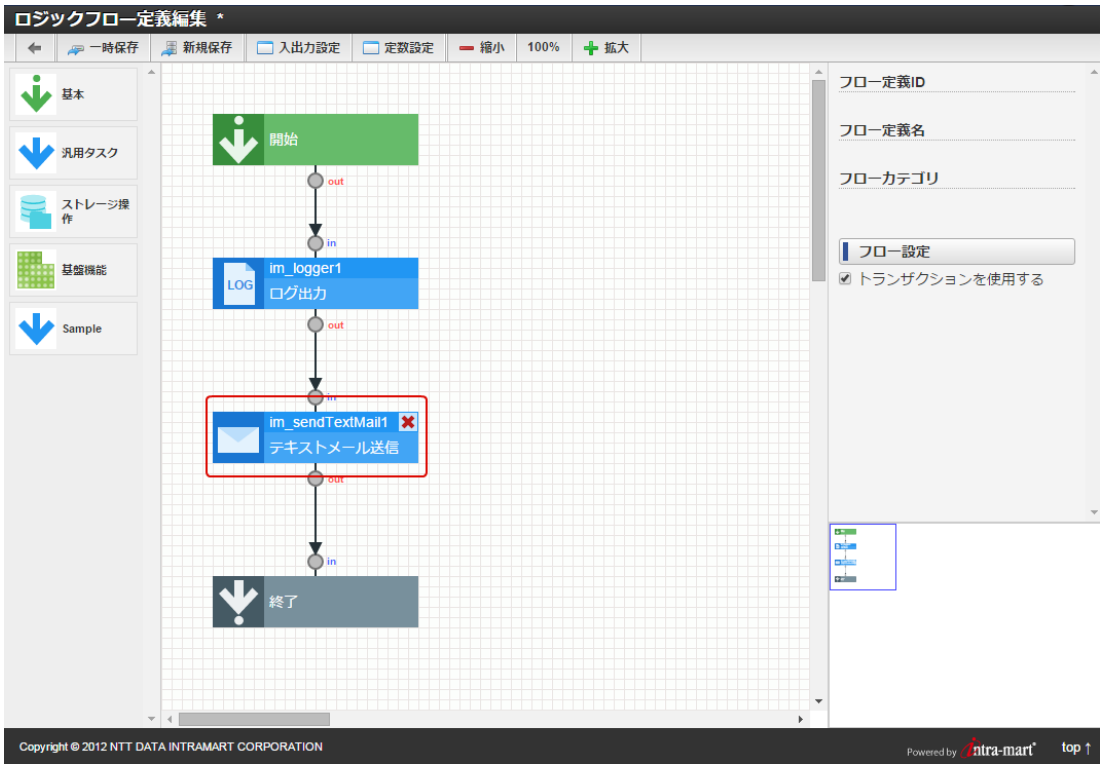



図：プロパティの設定

以上で、「ログ出力」タスクのプロパティ設定が完了しました。
プロパティに設定した内容は自動で保持されます（プロパティごとに保存アクションを行う必要はありません）。

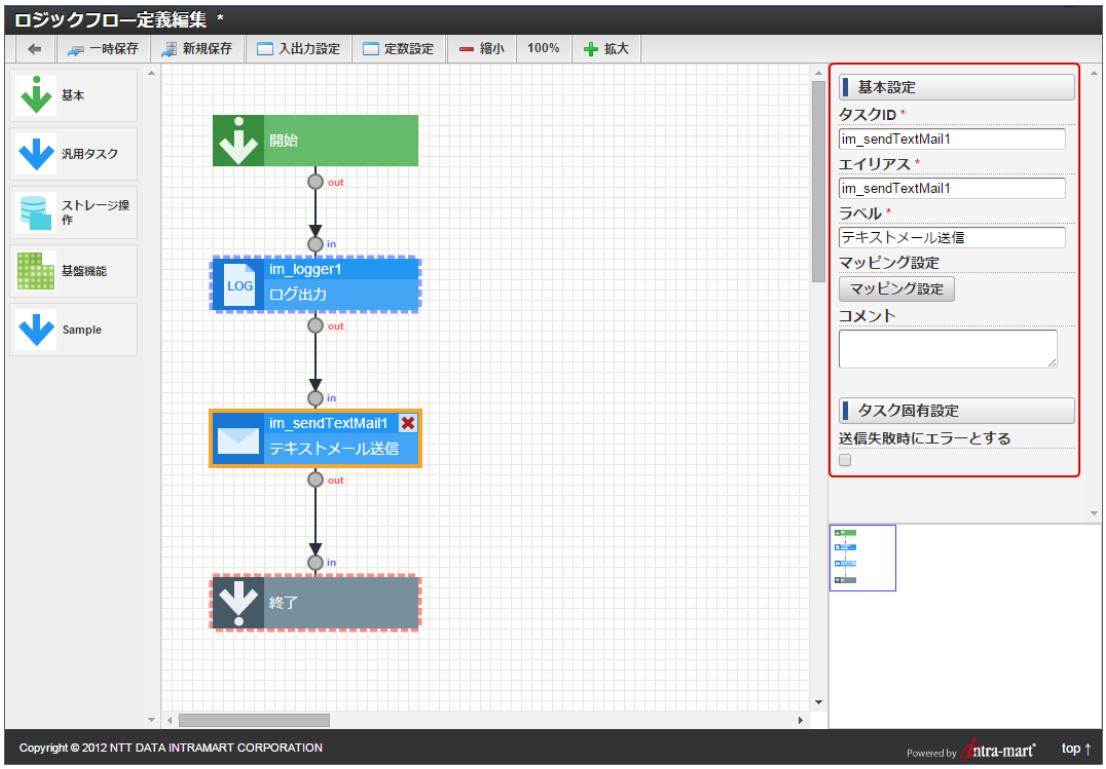
次に、同様の手順で「テキストメール送信」タスクのプロパティを設定します。
なお、「テキストメール送信」のプロパティはデフォルトのままとするため、ここではプロパティの確認のみ行います。

1. フロー編集画面上の「テキストメール送信」タスクをクリックします。




図：「テキストメール送信」タスク

2. 画面右側に「テキストメール送信」タスクに関するプロパティ画面が表示されます。




図：「テキストメール送信」タスクのプロパティ画面

3. タスク固有設定の項目が以下のとおりであることを確認してください。
   - 送信失敗時にエラーとする - チェックなし

以上で、プロパティ設定が全て完了しました。

> **コラム**
>
> プロパティ設定項目について
>
> 各エレメントの持つプロパティの設定項目の詳細は以下を参照してください。
>
> - **基本設定**
>   - 「*ロジックフローに共通する設定詳細*」
> - **タスク固有設定**
>   - 「IM-LogicDesigner仕様書」 - 「付録」 - 「タスク一覧」

次章「*定数値を定義する*」では、作成しているロジックフロー内で参照可能な定数値を定義します。

## 定数値を定義する

次に、作成するロジックフローの定数値の設定を行います。



### 定数値とは

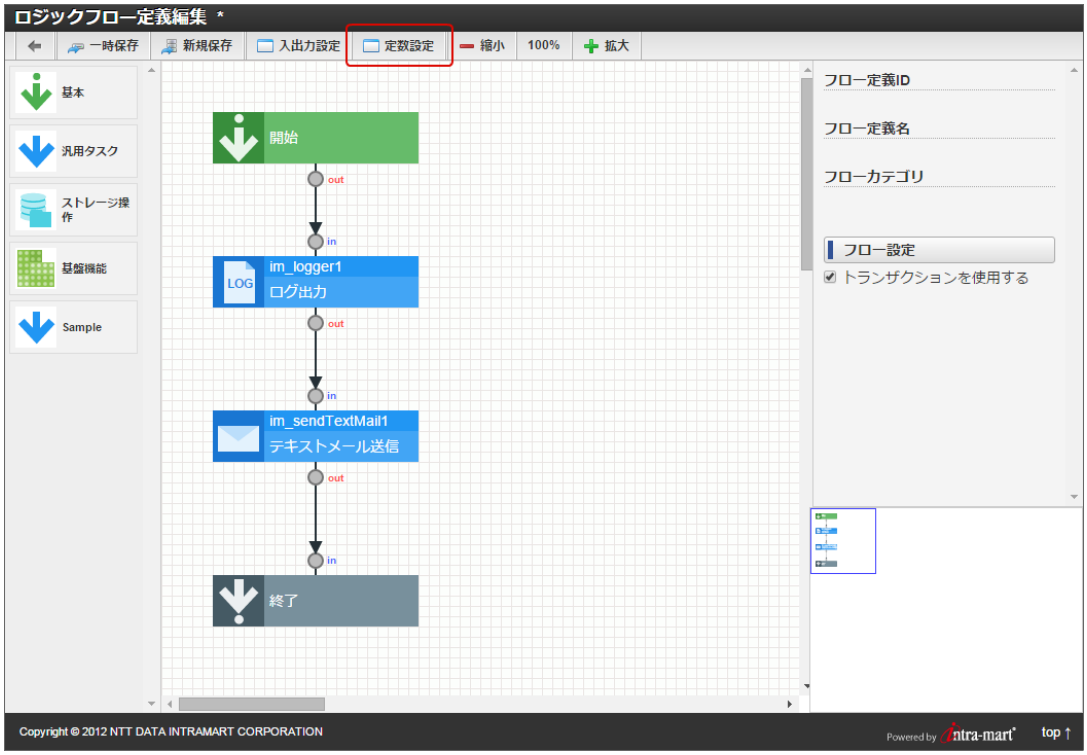
ロジックフローの定数値とは、ロジックフロー全体を通して共通して利用可能な固定値を定義したものです。
定数値として定義した値は、ロジックフロー内のどこからでも参照可能です。
作成するフローの一連の流れの中で、変更の無い値は定数値として定義することを推奨します。

> **コラム**
>
> 定数値の型について
>
> 定数値として定義した値は、全て文字列（string）として扱われます。

## 定数設定画面を開く

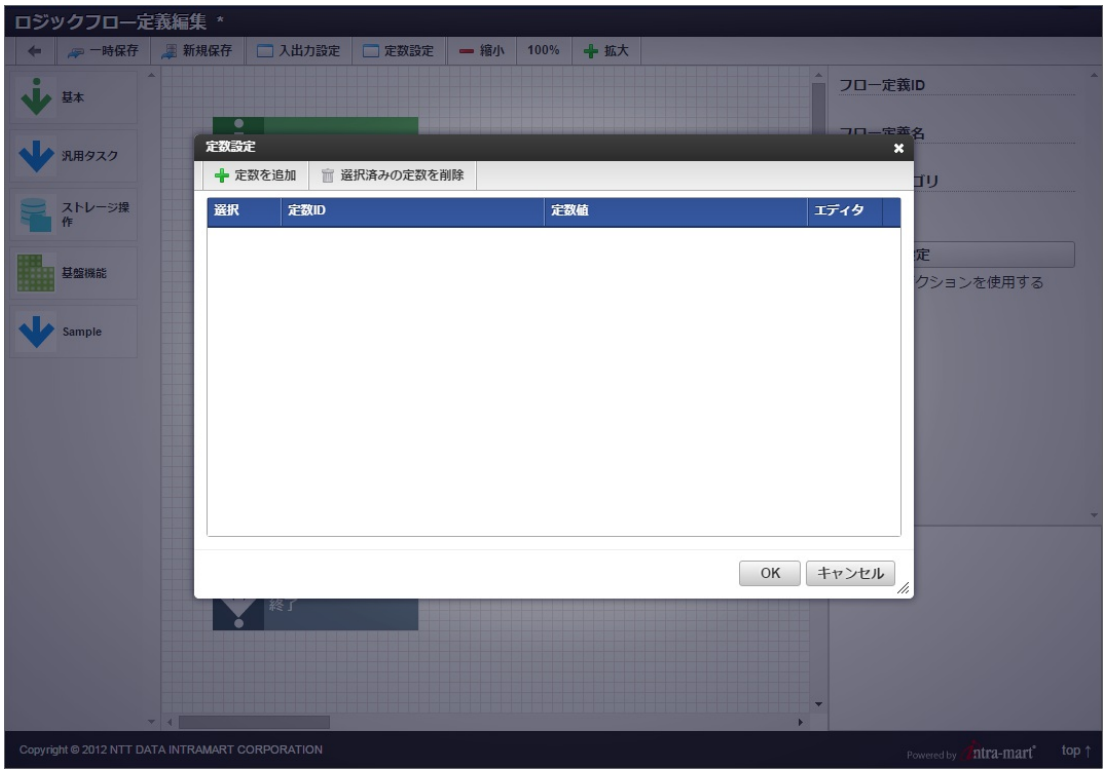
定数設定画面は、ロジックフロー定義編集画面から開きます。

1. ロジックフロー定義編集画面上部、ヘッダ内の「定数設定」をクリックします。




図：ロジックフロー定義編集画面ヘッダ

2. 定数設定画面が表示されます。




図：定数設定画面

## 定数値の定義

本チュートリアルで作成するロジックフローの定数値を定義します。

本チュートリアルで利用する定数値の詳細

本チュートリアルでは「テキストメール送信」タスク、および、「終了」制御要素に利用する値を、定数値として定義していきます。
定義する定数値は以下の通りです。

**「テキストメール送信」タスク**

| パラメータ名（定数ID） | 定数値 | 説明 |
|---|---|---|
| subject | IM-LD Tutorial | メールの題名です。 |
| from | aoyagi@intra-mart.jp | メールの送信者として送信可能なメールアドレスです。<br>本チュートリアルでは「aoyagi@intra-mart.jp」としています。 |
| to | ueda@intra-mart.jp | メールの宛先として送信可能なメールアドレスです。<br>先に設定したメールの送信者と同一のメールアドレスでも可です。<br>本チュートリアルでは「ueda@intra-mart.jp」としています。 |

> **注意**
>
> 宛先に設定するメールアドレスについて
>
> テスト環境などでチュートリアルを進める場合、宛先に設定するメールアドレスは必ず試験用のアドレスを用いてください。
> ロジックフローの実行テストを行う際、大量のメールが送信される可能性があります。

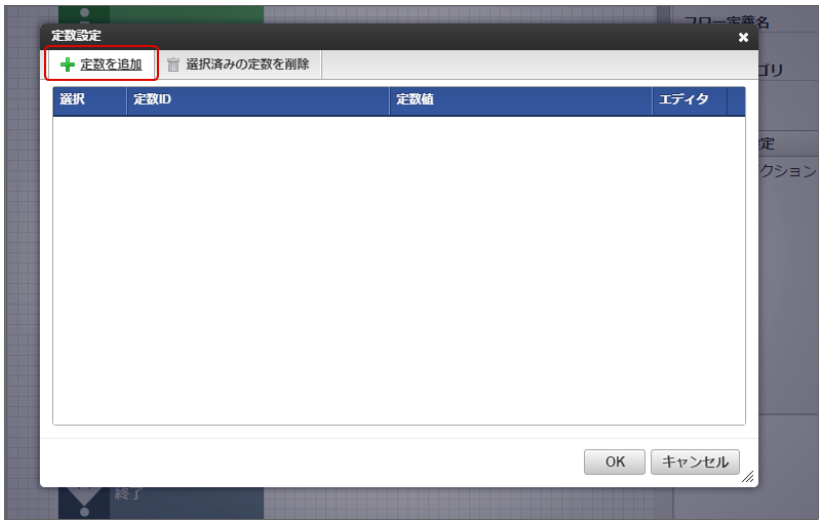
**「終了」制御要素**

| パラメータ名（定数ID） | 定数値 | 説明 |
|---|---|---|
| result | true | 正常実行されたことを表すフラグです。<br>今回のチュートリアルでは、処理が最後まで完了した場合、必ずtrueを返すものとします。 |

## 定数値の定義

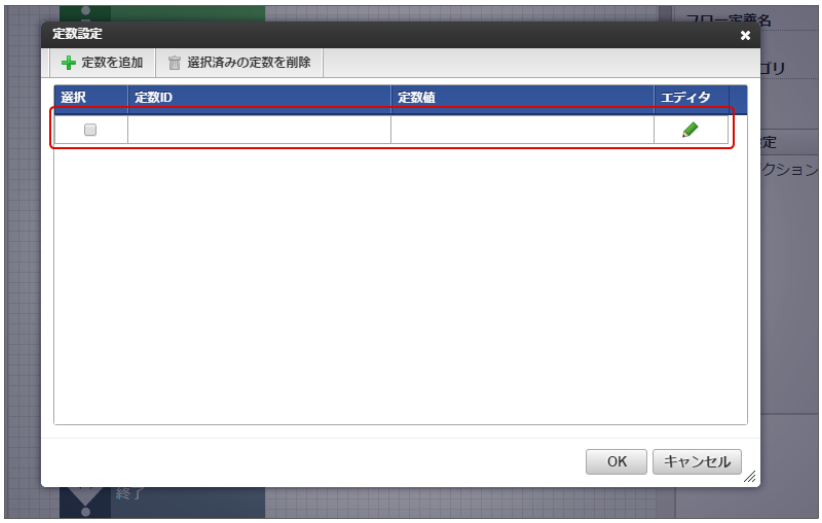
定義内容をもとに、定数設定画面で実際に設定を行います。

1. 定数設定画面のヘッダ内、「定数を追加」をクリックします。



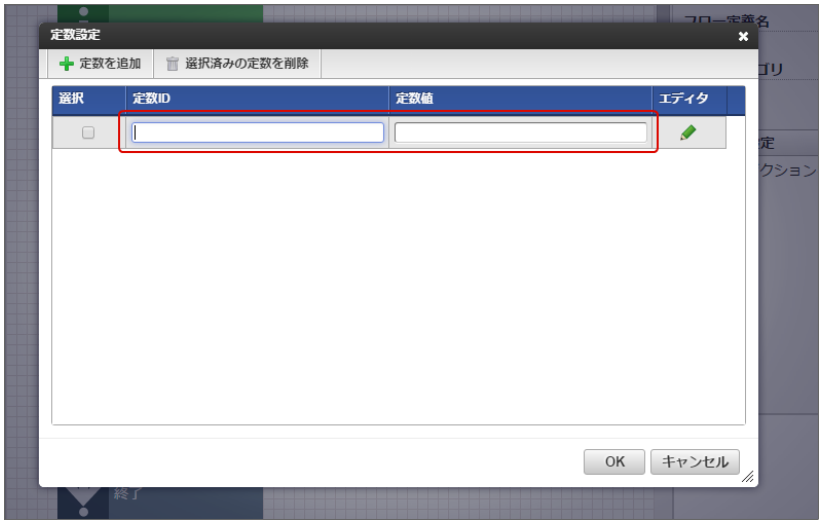

図：定数の追加

2. 未定義の定数フィールドが追加されます。



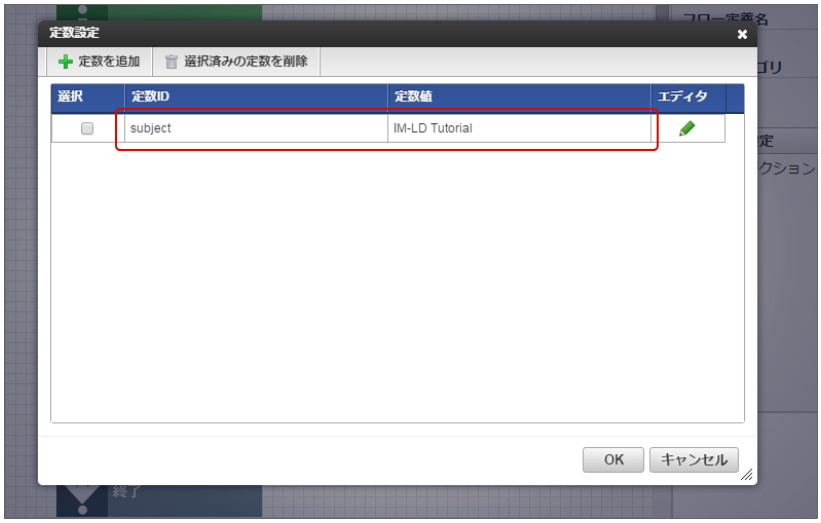

図：未定義の定数フィールド

3. 未定義の定数フィールドの、定数ID、または、定数値の空白部分をクリックし、編集可能な状態にします。



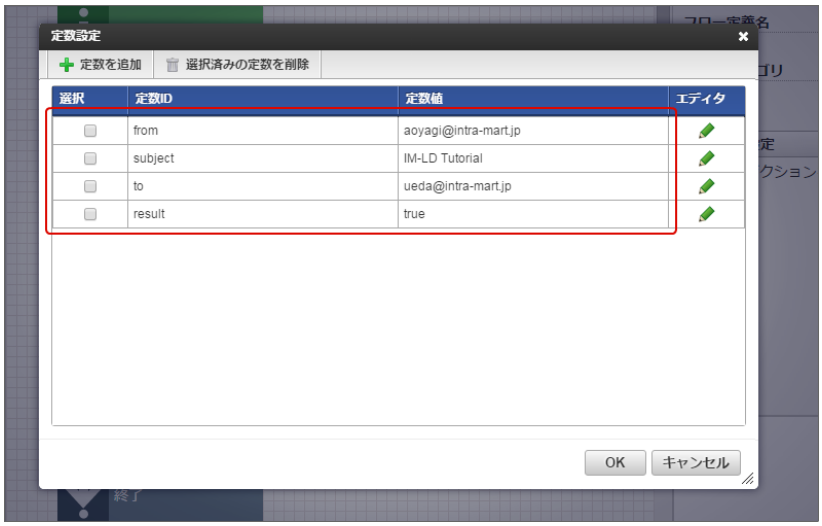

図：未定義の定数フィールドの編集

4. 「メールの題名」の項で定義した内容を設定します。




図：「メールの題名」定数の設定

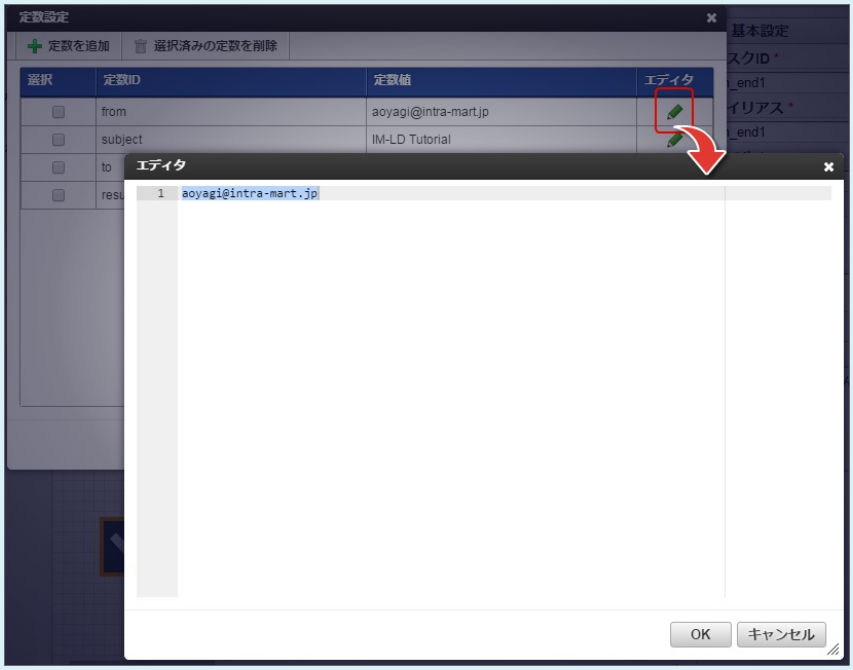
5. 同様の手順で、「メールの送信者」、「メールの宛先」、および、「正常実行を表すフラグ」についても設定します。




図：全ての定数の設定

6. 定数設定画面右下のOKをクリックします。

以上で、ロジックフローへの定数値設定が完了しました。
設定した定数値は、新規入力時と同様に定数ID、または、定数値のフィールドをクリックすることで編集可能です。

> **コラム**
>
> 定数値入力用エディタ
>
> 定義部分の右側にあるエディタアイコンをクリックすると、定数値入力専用の編集画面を開くことができます。
>
> 
> 
>
> 図：定数値入力専用エディタ
>
> 改行を含む定数値を定義する場合や、既存の定数値フィールドでは文字長が足らず入力がしづらい場合などにご利用ください。

次章「*マッピング設定を行う*」では、作成しているロジックフロー内における入出力値、および、各エレメント同士でのデータの受け渡しについて定義します。

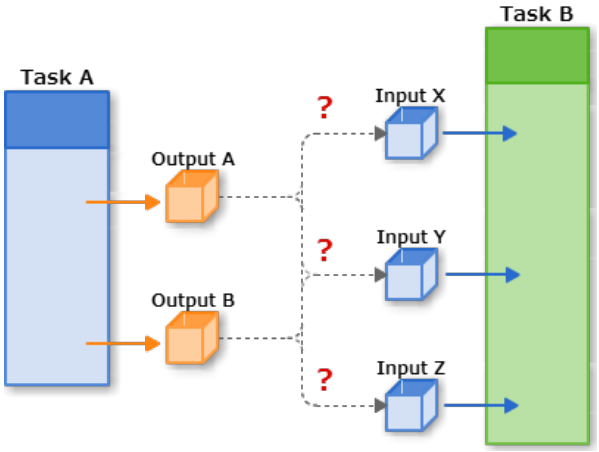
## マッピング設定を行う

次に、作成するロジックフローのマッピング設定を行います。

## マッピング設定とは

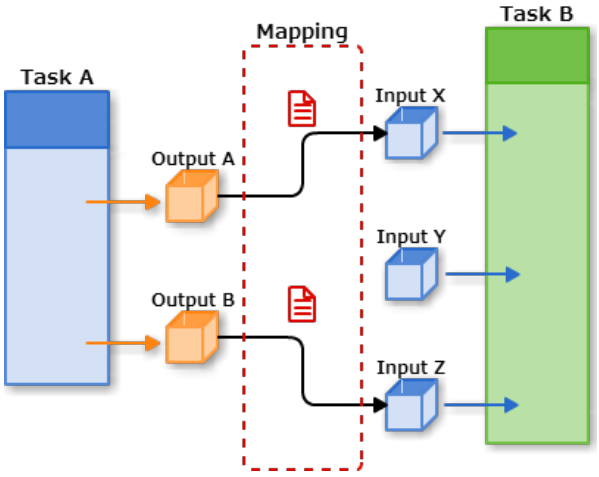
IM-LogicDesignerが提供するエレメント群は、（一部を除き）それぞれ自身が処理を行うために必要な入力値、および、その結果となる出力値が定義されています。
エレメントは、定義された入力値に合った値が渡されて、はじめて処理を行うことができます。

これまでの手順では、こうした「入力値へどのような値を渡すか？」「出力値をどうするか？」といったことが定義されていません。
また、「*入出力設定を定義する*」で定義したフロー自体の入出力情報も、現時点ではどのように扱うかが未定義のままです。




図：エレメントを並べただけでは、どのように値を受け渡していいか不明なため、処理が行えない。

ロジックフローのマッピング設定では、そうしたエレメント同士や入出力値、定数値などのデータのやり取りを定義します。



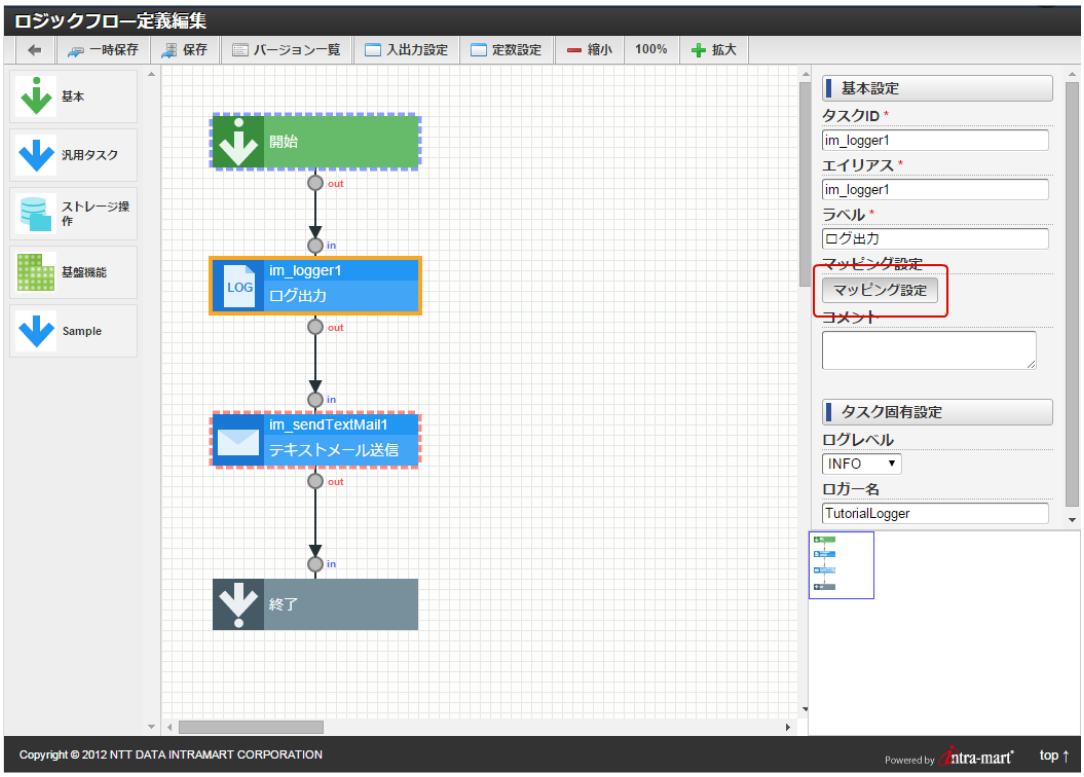

図：マッピング設定により、TaskAの出力値とTaskBの入力値がどのように紐づくか決定し、処理が行える。

> **コラム**
>
> タスクに定義された入出力値について
>
> IM-LogicDesignerが提供するエレメント群のうち、タスクに該当するものに定義されている入力値、および、出力値の詳細は、「IM-LogicDesigner仕様書」 - 「付録」- 「タスク一覧」を参照してください。

## マッピング設定画面を開く

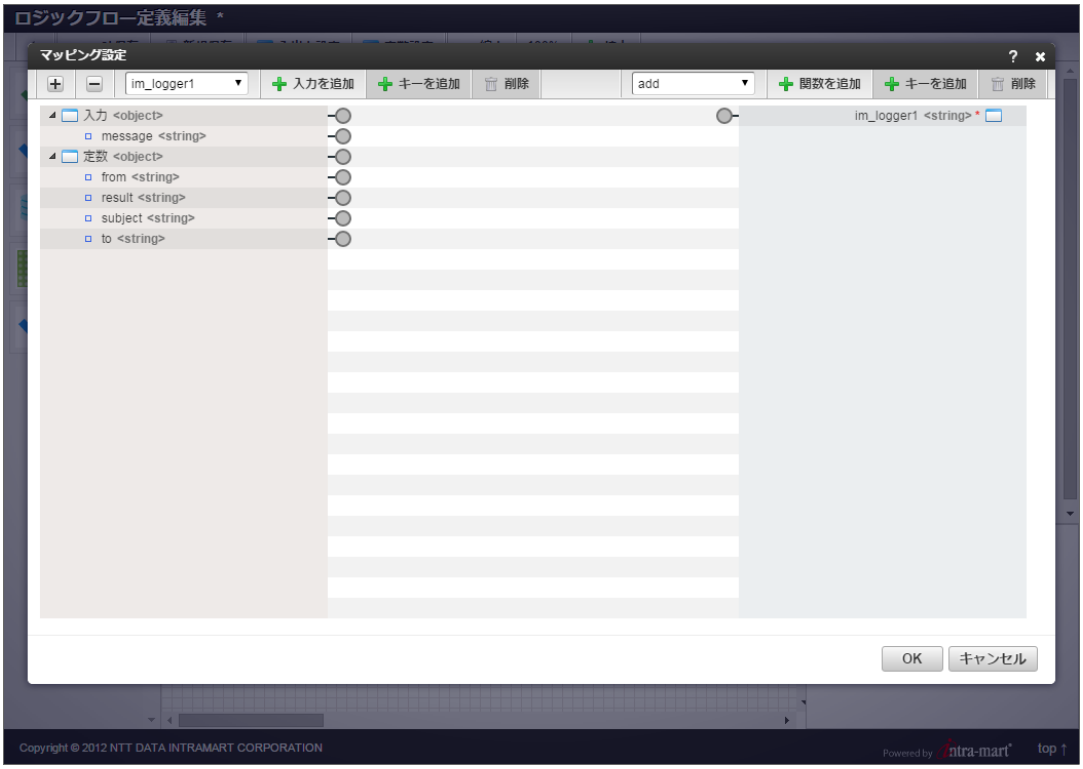
マッピングは、各エレメントのプロパティ画面からマッピング設定画面を開いて行います。
ここでは「ログ出力」タスクを例に説明します。

1. フロー編集画面上の「ログ出力」タスクをクリックし、プロパティ画面を表示します。
2. 基本設定の「マッピング設定」をクリックします。




図：「ログ出力」の基本設定

3. 「ログ出力」タスクのマッピング設定画面が開きました。




図：「ログ出力」マッピング設定画面

> **コラム**
>
> マッピング設定のないエレメント
>
> 「開始」制御要素など一部のエレメントは、マッピング設定を行う要素を持っていません。
> その場合、プロパティ画面上にマッピング設定の項目は表示されません。

## マッピング設定を行う

本チュートリアルで作成するロジックフローのマッピング設定を行います。

本チュートリアルでは、「ログ出力」、「テキストメール送信」タスク、および、「終了」制御要素のマッピングを以下のとおりに設定します。

- **ログ出力**

| 入力（始点） | 出力（終点） |
|---|---|
| 入力<object> - message<string> | im_logger1<string> |

- **テキストメール送信**

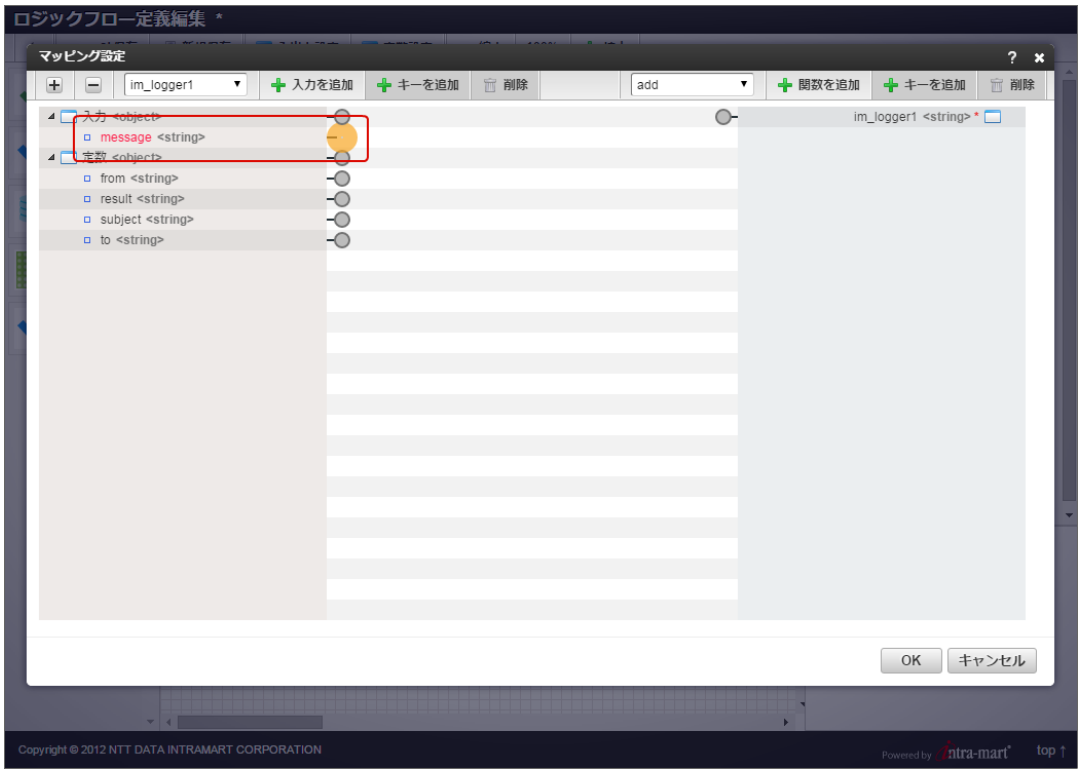
| 入力（始点） | 出力（終点） |
|---|---|
| 入力<object> - message<string> | im_sendTextMail1<object> - body<string> |
| 定数<object> - from<string> | im_sendTextMail1<object> - from<string> |
| 定数<object> - subject<string> | im_sendTextMail1<object> - subject<string> |
| 定数<object> - to<string> | im_sendTextMail1<object> - to<string[]> |

- **終了**

| 入力（始点） | 出力（終点） |
|---|---|
| 定数<object> - result<string> | 出力<object> - result<boolean> |

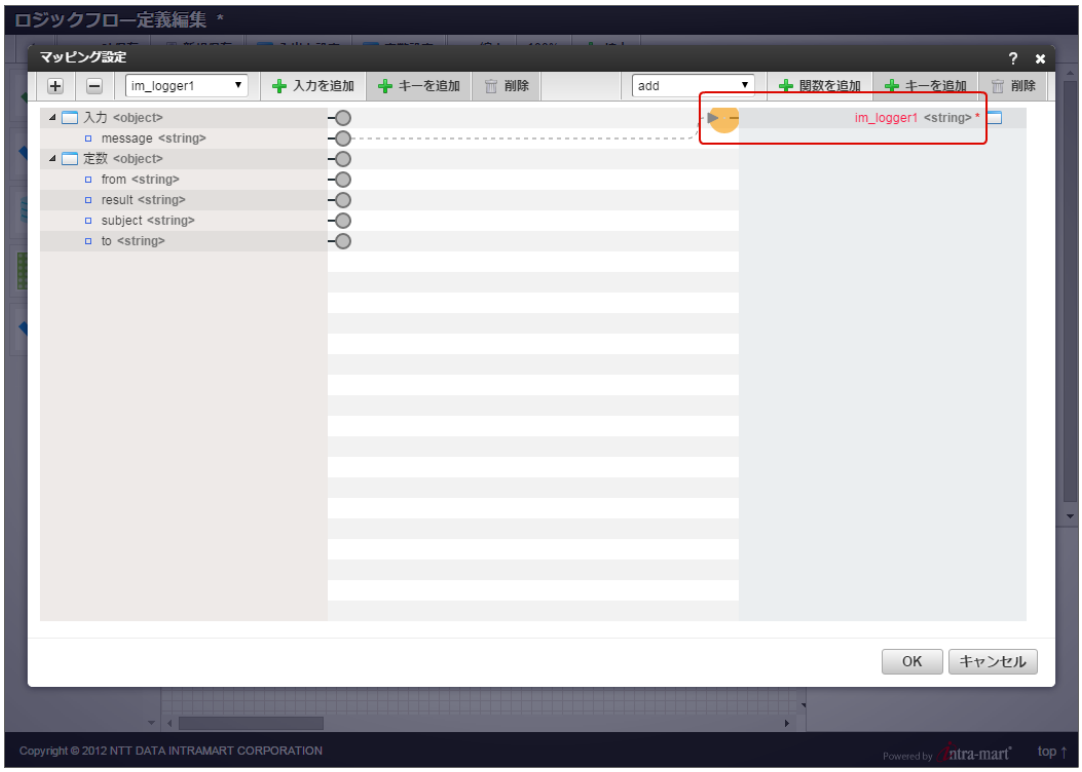
設定内容をもとに、マッピング設定画面で実際に設定を行います。

1. 「ログ出力」タスクのマッピング設定画面を開きます。
2. 設定画面左部、「入力<object>」要素の下にある「message<string>」から出ている端子にカーソルをあわせます。



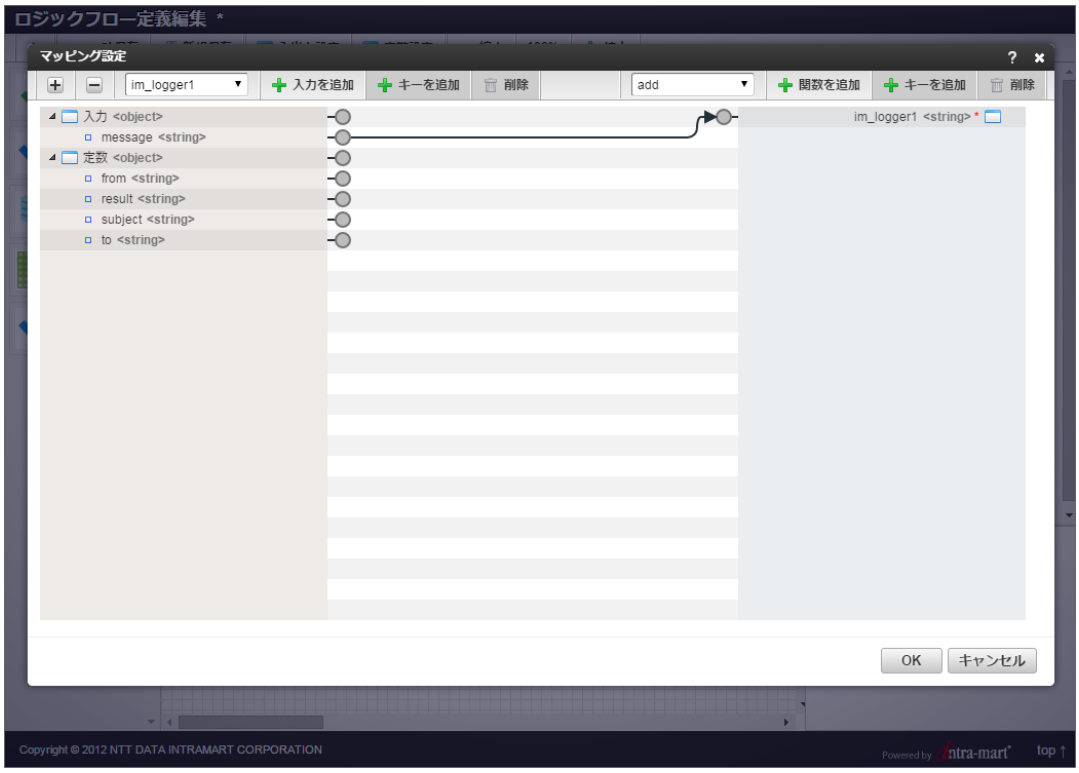

図：message<string>にカーソルをあわせる

3. そのままドラッグし、設定画面右部、「im_logger1<string>」から出ている端子にドロップします。



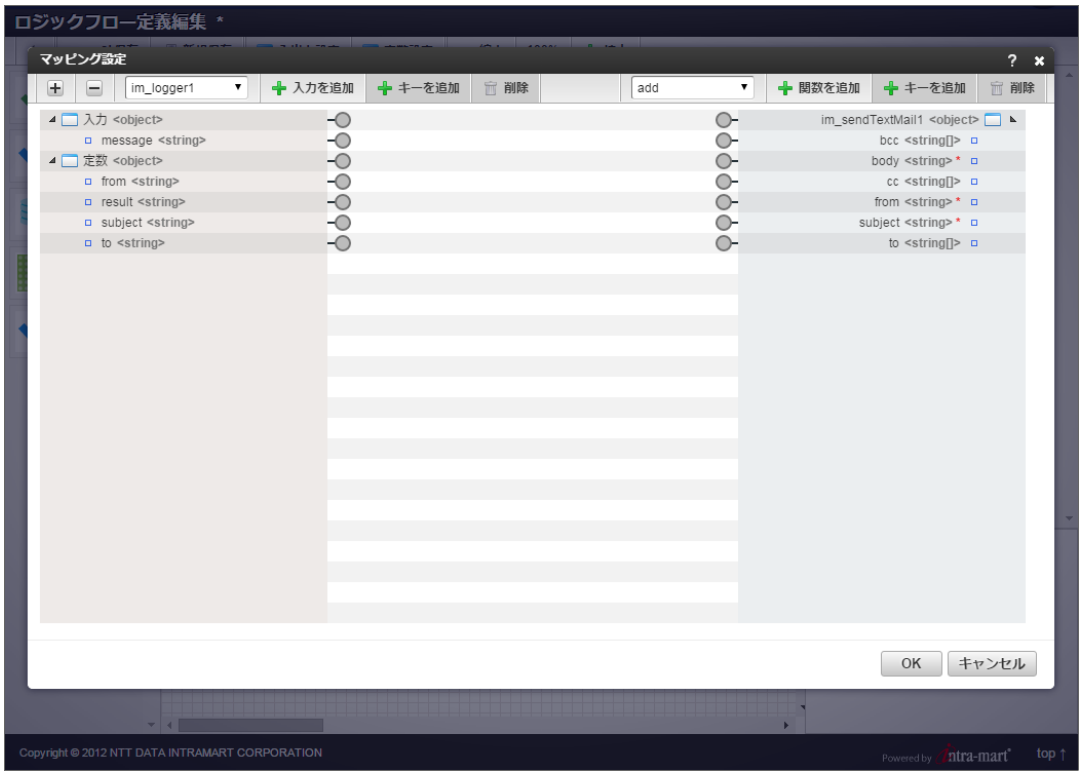

図：im_logger1<string>でドロップ

4. これにより、フロー実行時に入力される情報と、ログ出力を行う本文を表す情報がマッピングされました。



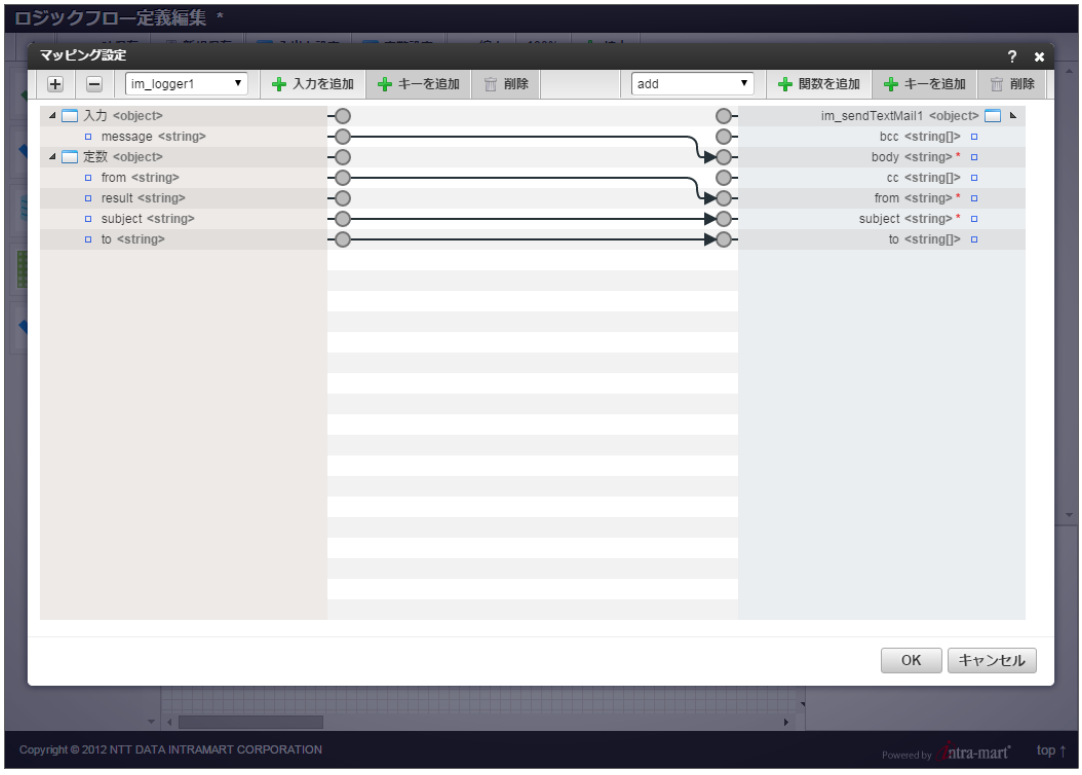

図：マッピング情報の作成

5. 設定画面右下のOKをクリックし、「ログ出力」タスクのマッピング設定を終了します。
6. 同様の手順で、「テキストメール送信」タスクのマッピング設定画面を開きます。



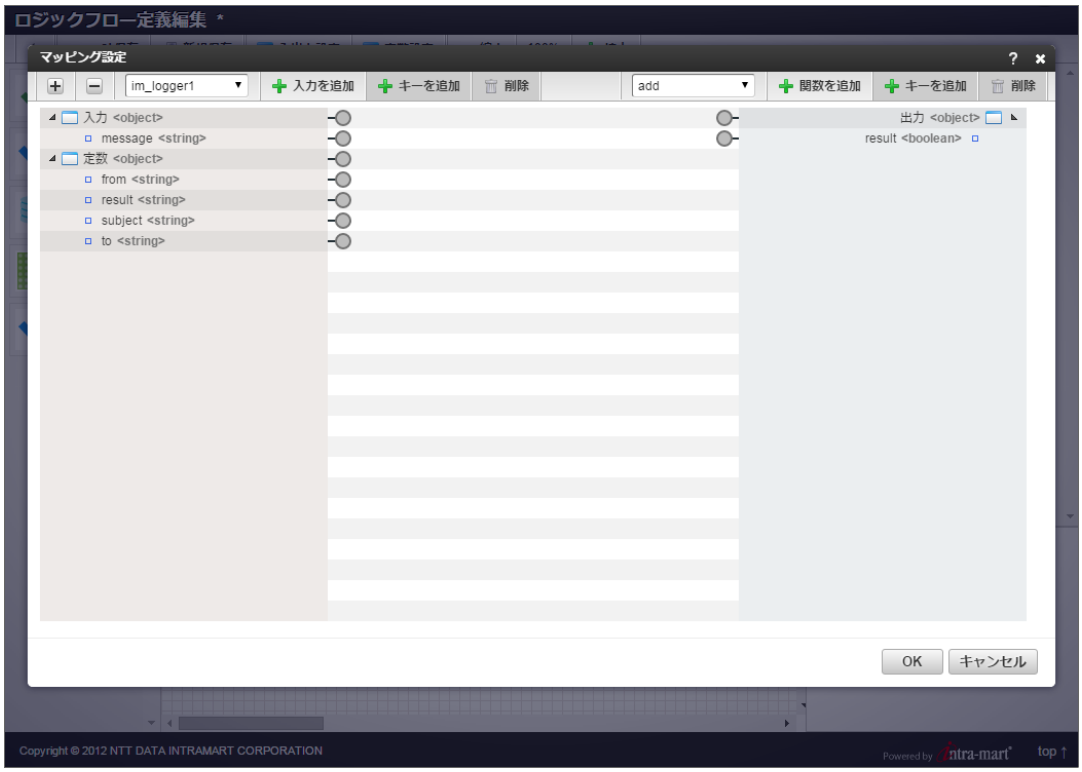

図：「テキストメール送信」マッピング設定画面

7. 設定情報を元に、「テキストメール送信」タスクのマッピングを行います。



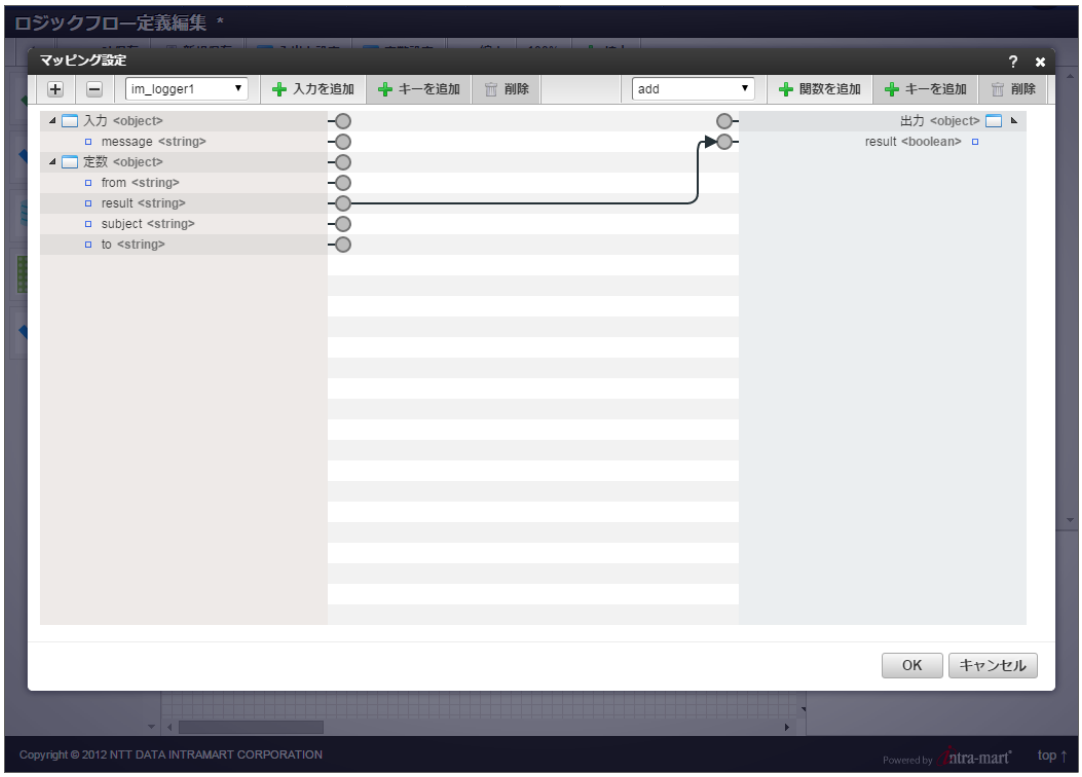

図：マッピング情報の作成

8. 設定画面右下のOKをクリックし、「テキストメール送信」タスクのマッピング設定を終了します。
9. 同様の手順で、「終了」制御要素のマッピング設定画面を開きます。




図：「終了」マッピング設定画面

10. 設定情報を元に、「終了」制御要素のマッピングを行います。




図：マッピング情報の作成

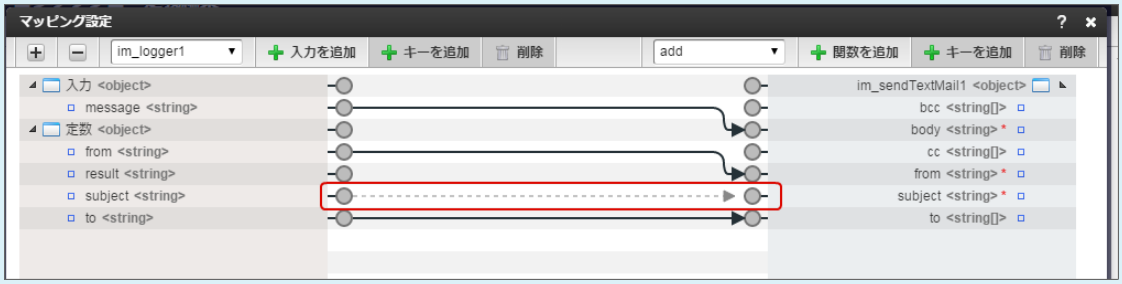
11. 設定画面右下のOKをクリックし、「終了」制御要素のマッピング設定を終了します。

> **コラム**
>
> 型の異なる値のマッピング
>
> 「テキストメール送信」のtoや、「終了」のresultのマッピングに関して、型の違うもの同士を繋げています。
> IM-LogicDesignerでは、こうした型の違うマッピングが行われた場合、（可能な限り）自動で型を変換し、解決します。
> この機能により多くの場合開発者は、入力値と出力値の型を気にすること無くロジックフローを定義できます。
>
> 変換可能、および、不可能なパターンについての詳細は、「IM-LogicDesigner データ型変換 仕様書」を参照してください。

コラム

有効な線の定義

入力（始点）と出力（終点）の間を正しく接続している場合、接続している線は黒色で表されます。
逆に、線が正しく接続できておらず、途中で途切れているなどしている場合、線は灰色で表されます。



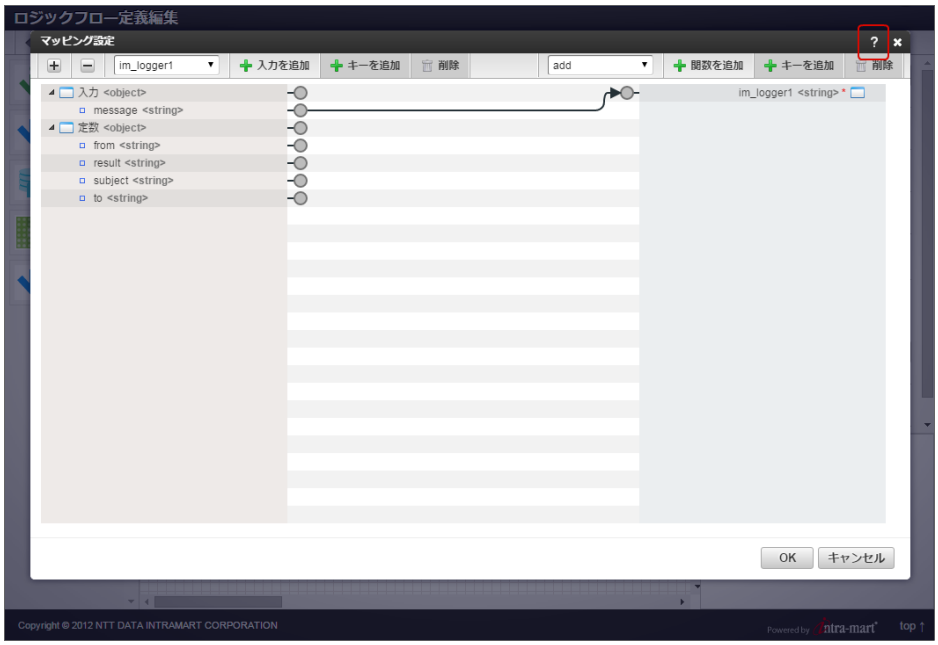

図：正しく接続できていない例

以上で、ロジックフローへのマッピング設定が完了しました。
そして、このマッピング設定をもって、ロジックフローを動作させるために必要な設定が全て完了しました。

### 参考：マッピング設定画面のサイトツアー

マッピング設定画面には、「*参考：ロジックフロー定義編集画面のサイトツアー*」と同様に、サイトツアーによるナビゲーションが含まれています。
マッピング設定画面のサイトツアーの開始方法は以下の通りです。

1. マッピング設定画面を開き、画面右上の「?」をクリックします。




図：マッピング設定画面のサイトツアーの開始ボタン

2. サイトツアーが開始します。


図：マッピング設定画面のサイトツアー

次章「*保存する*」では、これまで定義した内容を保存する方法を説明します。

## 保存する

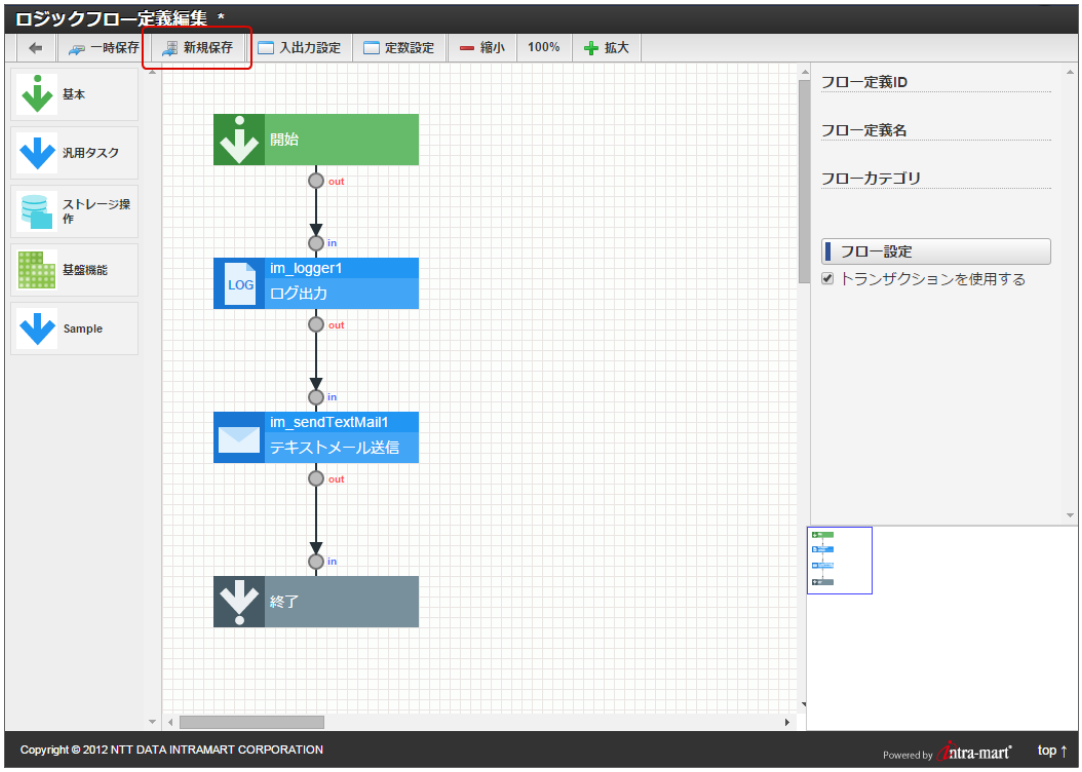
次に、これまで実施してきた内容を保存する方法を説明します。



### 保存する

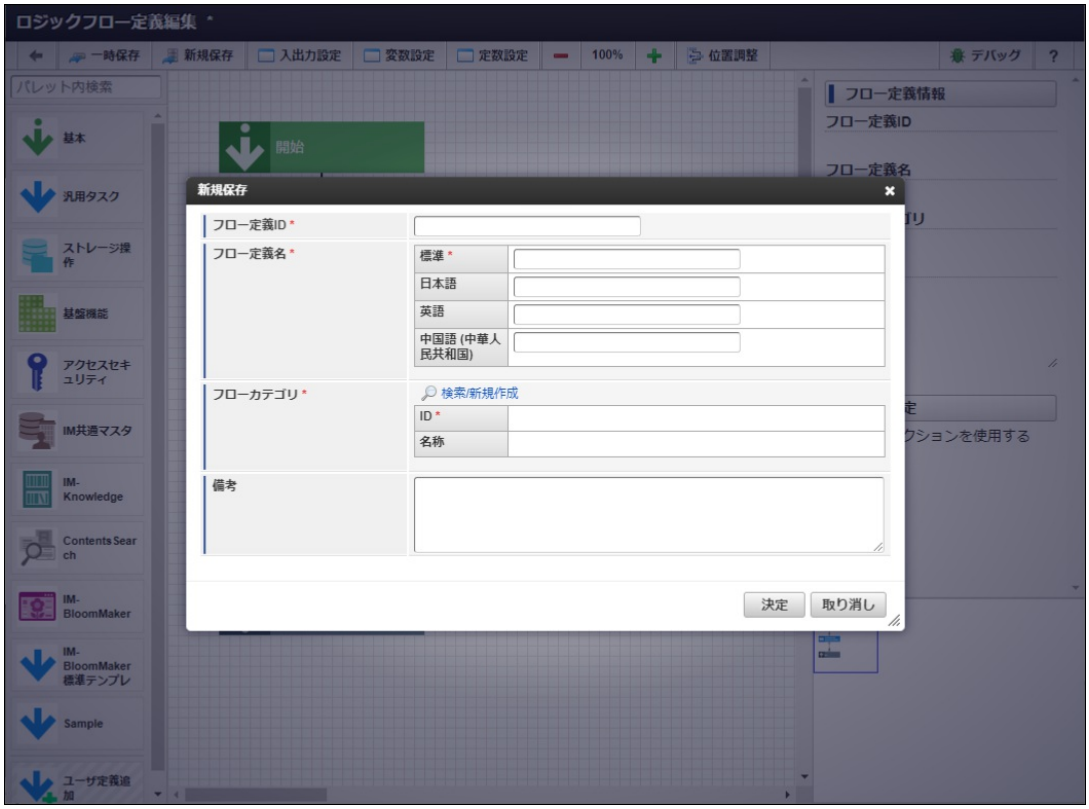
ロジックフローを作成する最後の手順として、これまで作成してきた内容を保存します。

1. ロジックフロー定義編集画面上部、ヘッダ内の「新規保存」をクリックします。



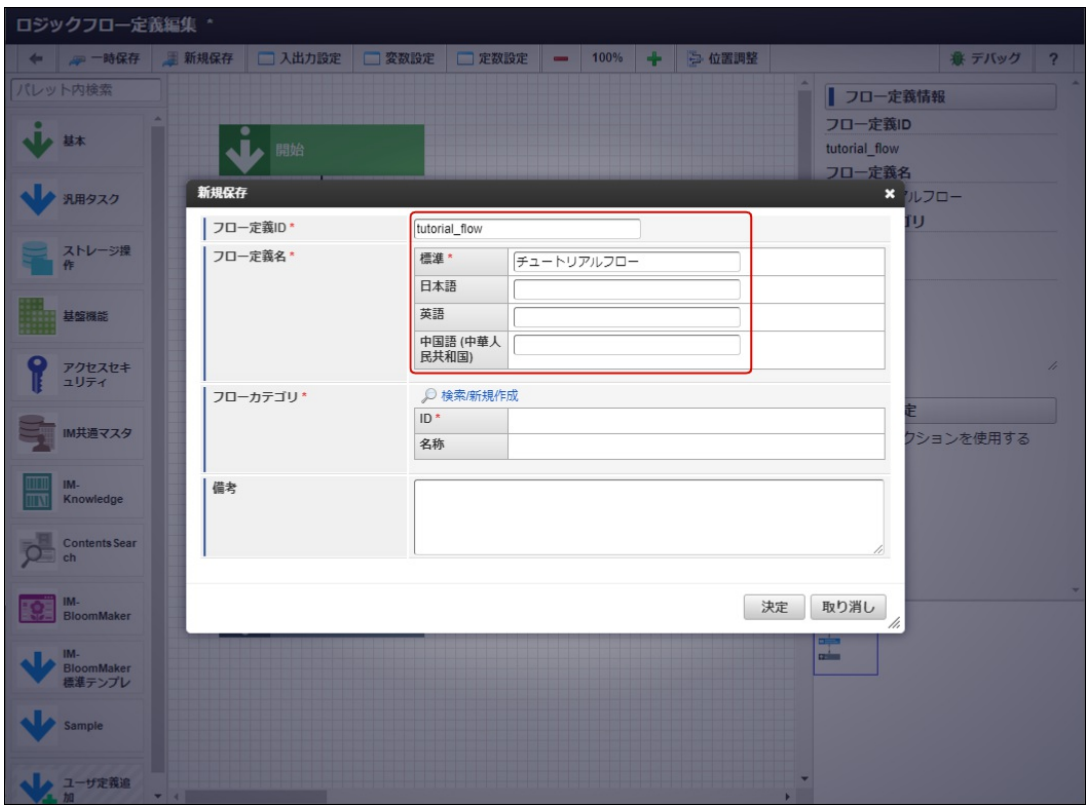

図：ロジックフロー定義編集画面ヘッダ

2. 新規保存画面が表示されます。



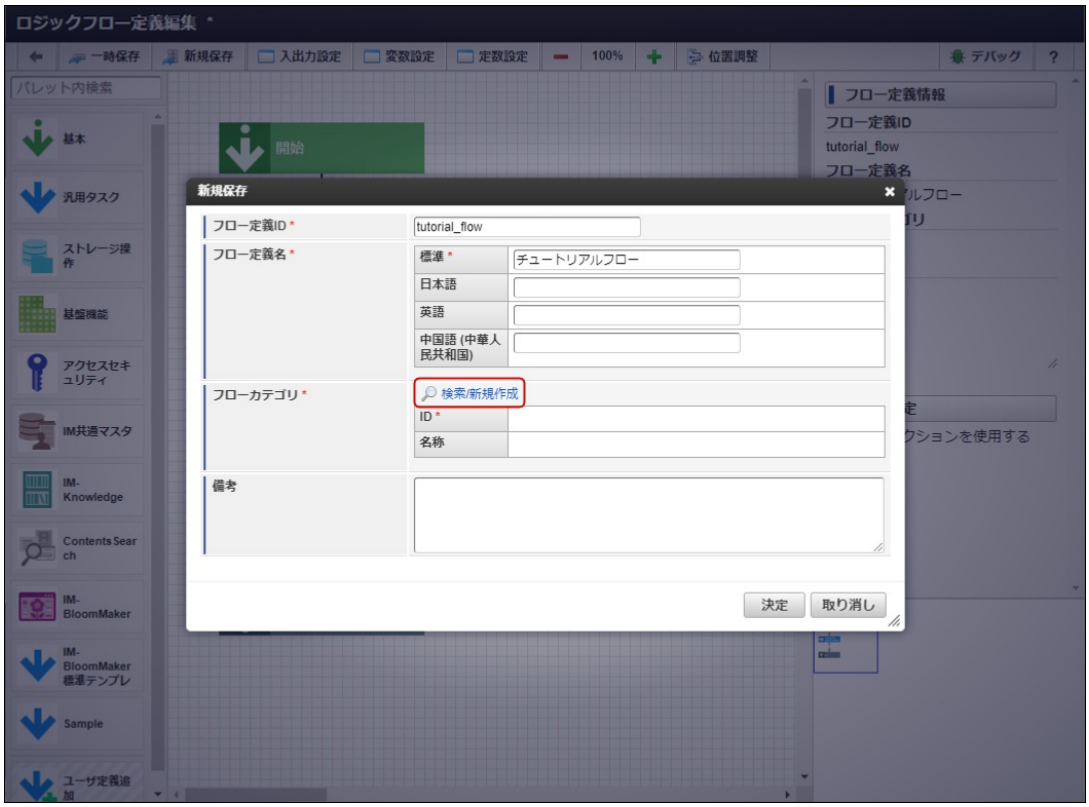

図：新規保存画面

3. 各項目に以下の値を入力します。
   - フロー定義ID「tutorial_flow」
   - フロー定義名
     - 標準 -「チュートリアルフロー」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし




図：保存情報の定義

4. 作成するロジックフローが属するカテゴリを設定するために、フローカテゴリの検索/新規作成リンクをクリックします。

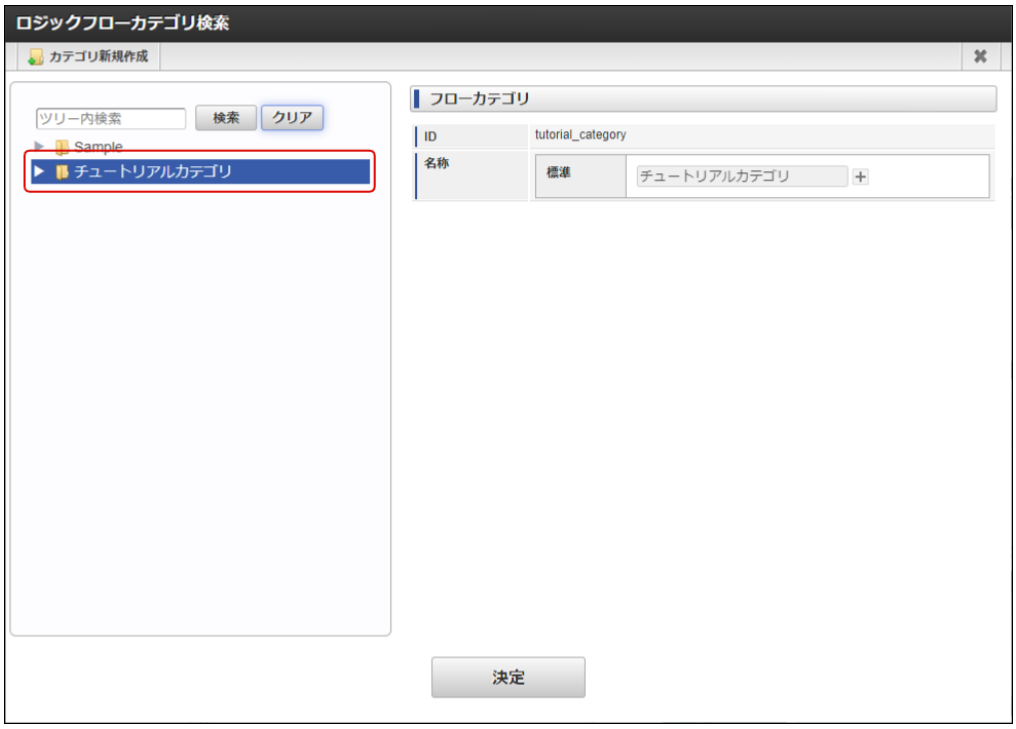



図：フローカテゴリの検索

5. ロジックフローカテゴリ検索画面がポップアップします。

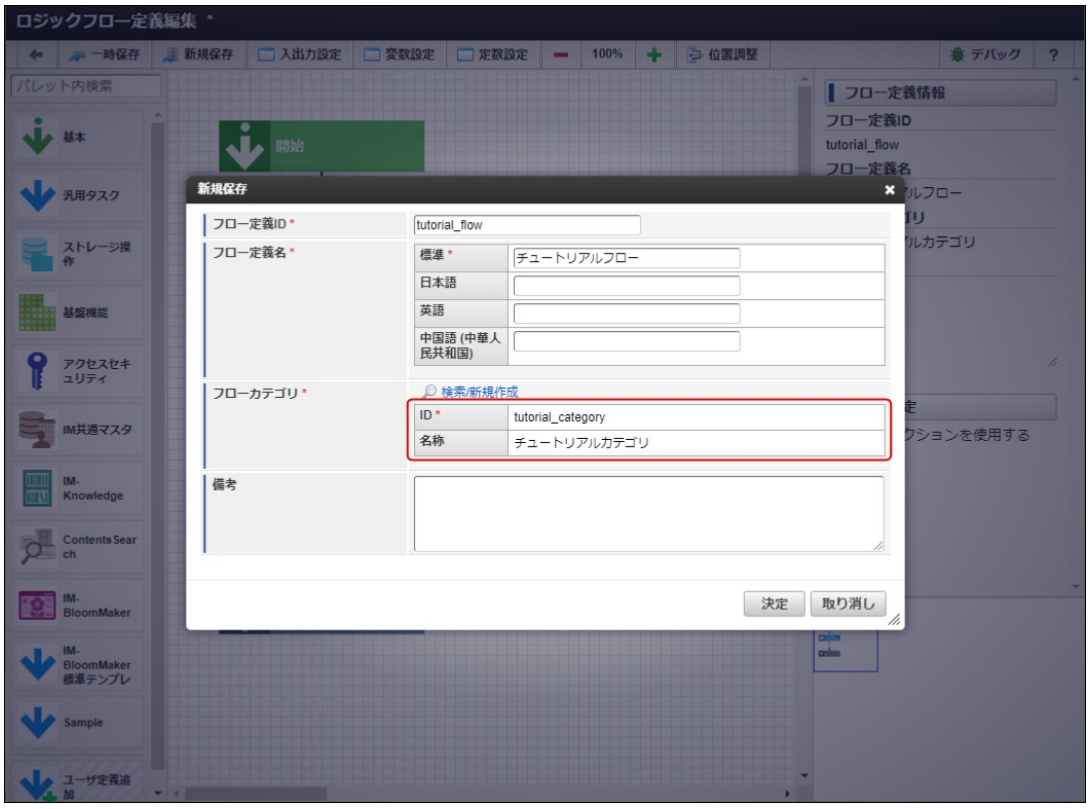

図：ロジックフローカテゴリー覧画面

6. 左ペインのツリーから「チュートリアルカテゴリ」を選択します。




図：フローカテゴリの選択

7. 一覧画面下部の「決定」をクリックします。
8. 一覧画面が閉じられ、新規保存画面のフローカテゴリの項目に、選択したフローカテゴリの情報が入力されます。




図：フローカテゴリ情報の設定

9. 新規保存画面右下のOKをクリックします。
10. 保存を確認するダイアログが表示されるので、OKをクリックします。
11. 保存が完了した旨のメッセージと共に、ロジックフロー定義一覧画面に遷移します。


図：ロジックフローの保存

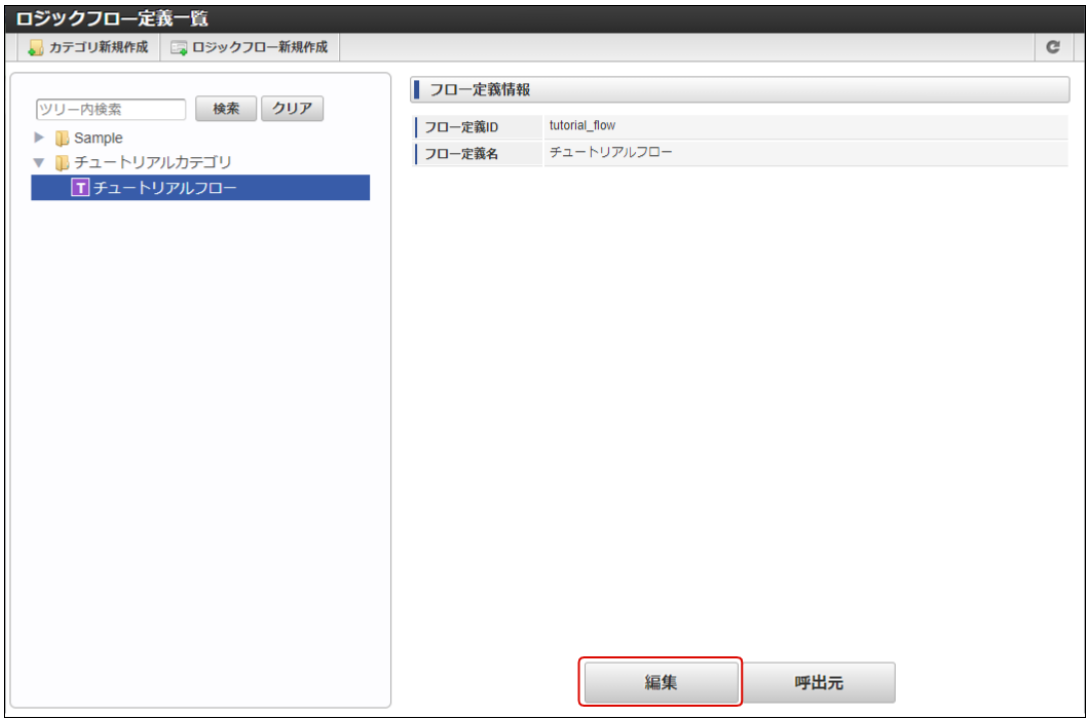
以上で、ロジックフローの保存が完了しました。

> **コラム**
>
> 保存時に行われる一時保存情報の削除
>
> IM-LogicDesignerでは「*一時保存する*」で扱った一時保存処理で保存したデータを、以下の保存タイミングで削除します。
>
> - 新規に作成したロジックフローの一時保存データの場合、その一時保存データが利用されたかを問わず、ロジックフローの新規保存時に削除されます。
> - 既に保存済みのロジックフローの一時保存データの場合、元となったロジックフローの保存（更新）時に削除されます。

## 保存されたロジックフローを確認する

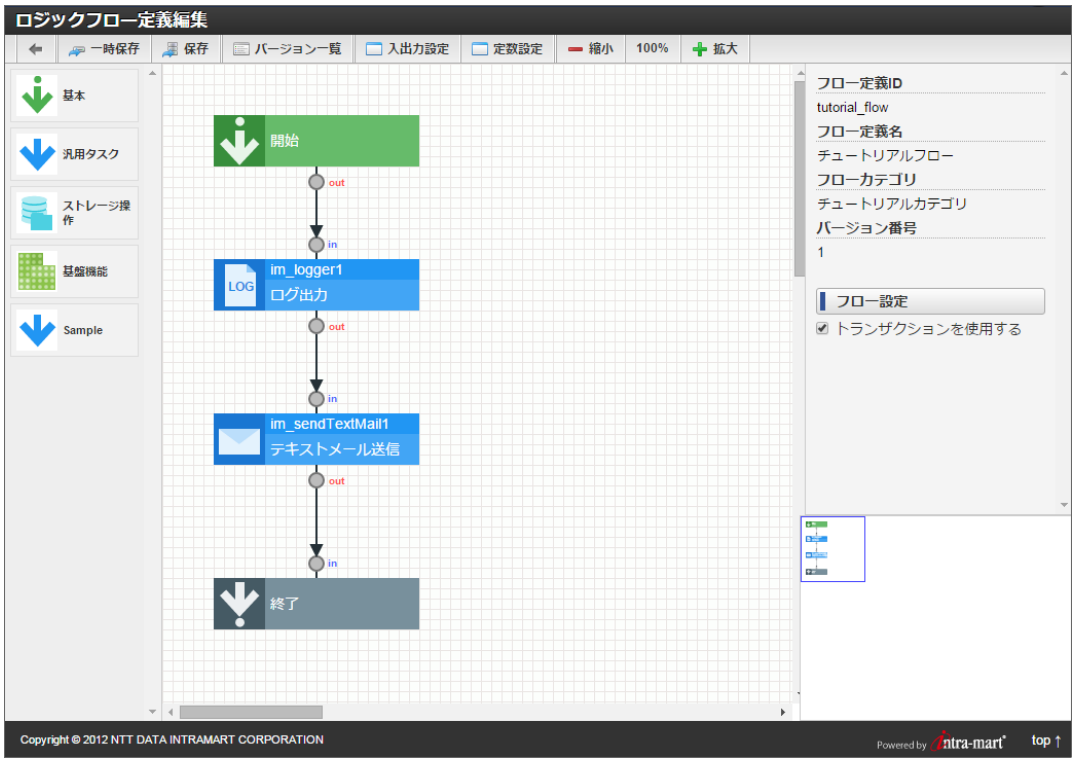
これまでの作業内容が正しく保存されているか確認します。

1. 「サイトマップ」→「LogicDesigner」→「フロー定義一覧」から、ロジックフロー定義一覧を開きます。
2. 左ペインのツリーから「チュートリアルカテゴリ」の開閉アイコンをクリックしカテゴリ配下を情報を表示します。
3. 本チュートリアルで作成したフロー定義「チュートリアルフロー」を選択し編集ボタンをクリックします。




図：編集ボタン

4. ロジックフロー定義編集画面が表示され、これまでの作業内容が全てロードされます。




図：ロジックフロー情報のロード

ロジックフローのフロー図や、入出力設定、定数値設定、各タスクのマッピング設定が正しく保存されていることを確認してください。

お疲れ様でした。
以上で「*チュートリアルの概要（作成物のイメージ）*」の章の初めに掲げた、

**「開発者の設定した入力情報を、コンソールへのログ、および、メールに発信し、その結果を出力情報として返す」**

という処理を行うロジックフローが完成しました。
送信したいメッセージを受け取り、コンソールとメールで出力するといったシンプルな処理内容ではありますが、IM-LogicDesignerを利用すればノンコーディングで実装できることがご理解いただけたかと思います。

次章「*フロールーティングを設定する*」では、今回作成したロジックフローをREST APIとして呼び出せるようにルーティング情報を定義します。

## フロールーティングを設定する

次に、ロジックフローをREST APIとして利用するため、ルーティング情報を定義します。

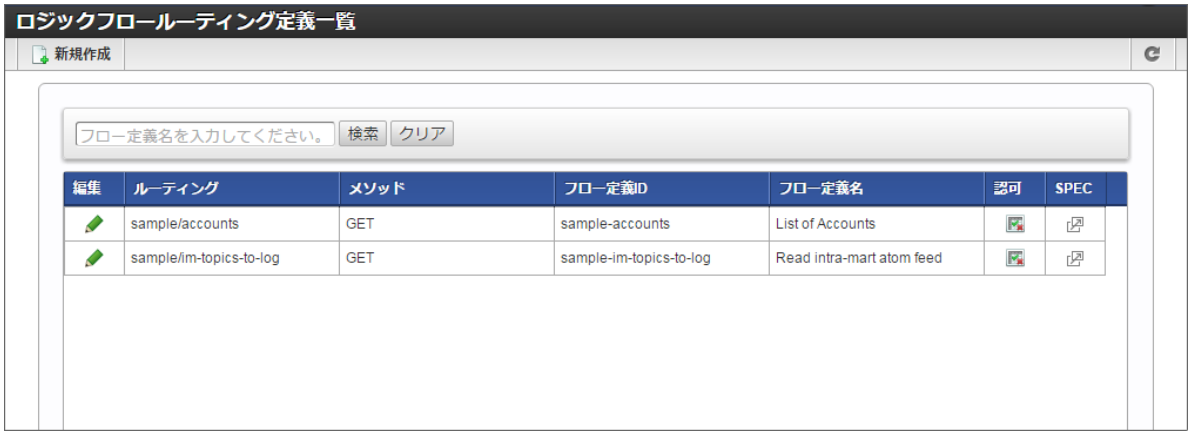
- ロジックフロールーティング定義一覧画面を表示する
- フロールーティングを新規作成する
- 作成したフロールーティングを確認する

### ロジックフロールーティング定義一覧画面を表示する

1. 「サイトマップ」→「LogicDesigner」→「ルーティング定義一覧」をクリックします。

図：サイトマップ

2. ロジックフロールーティング定義一覧画面が表示されます。

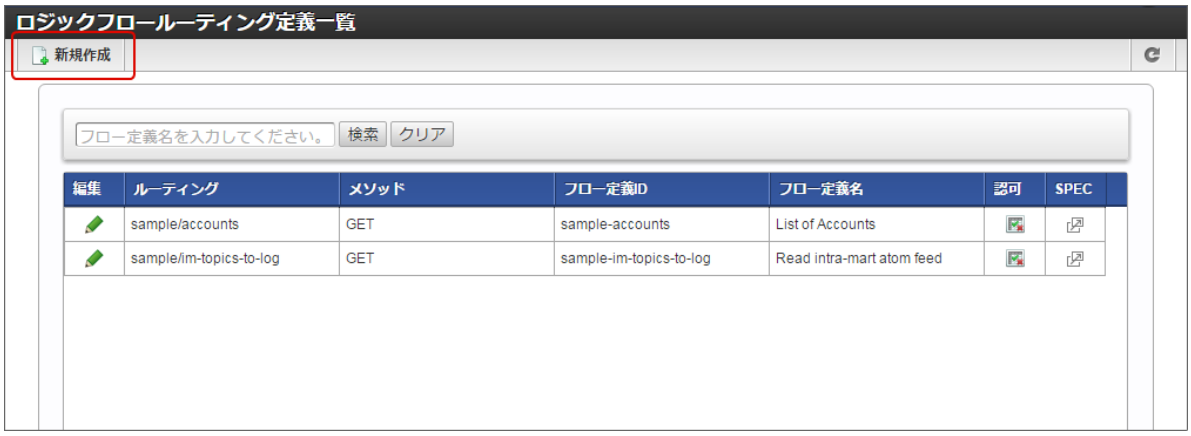



図：ロジックフロールーティング定義一覧画面

## フロールーティングを新規作成する

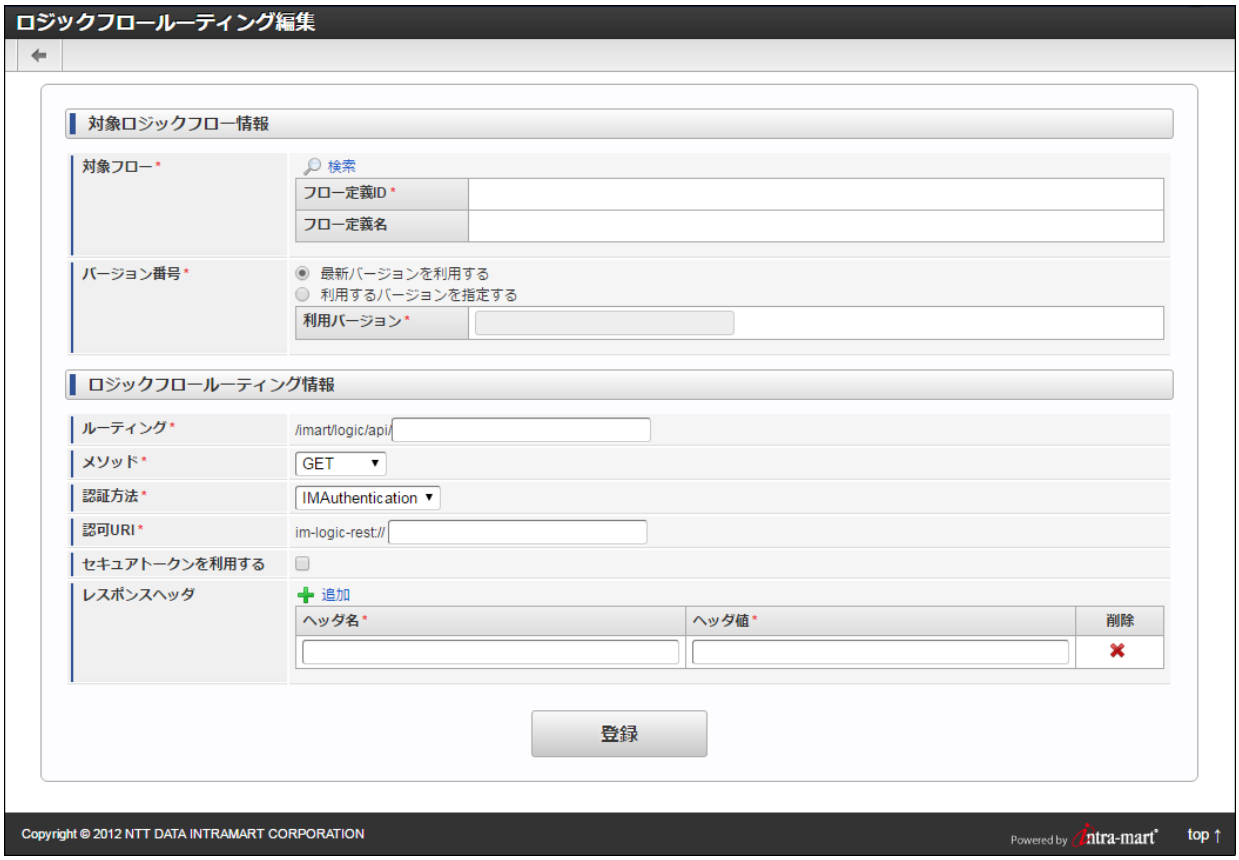
本チュートリアルで作成したロジックフローのルーティング情報を作成します。
ルーティングの作成は、ロジックフロールーティング編集画面から行います。

1. ロジックフロールーティング定義一覧画面左上の「新規作成」をクリックします。



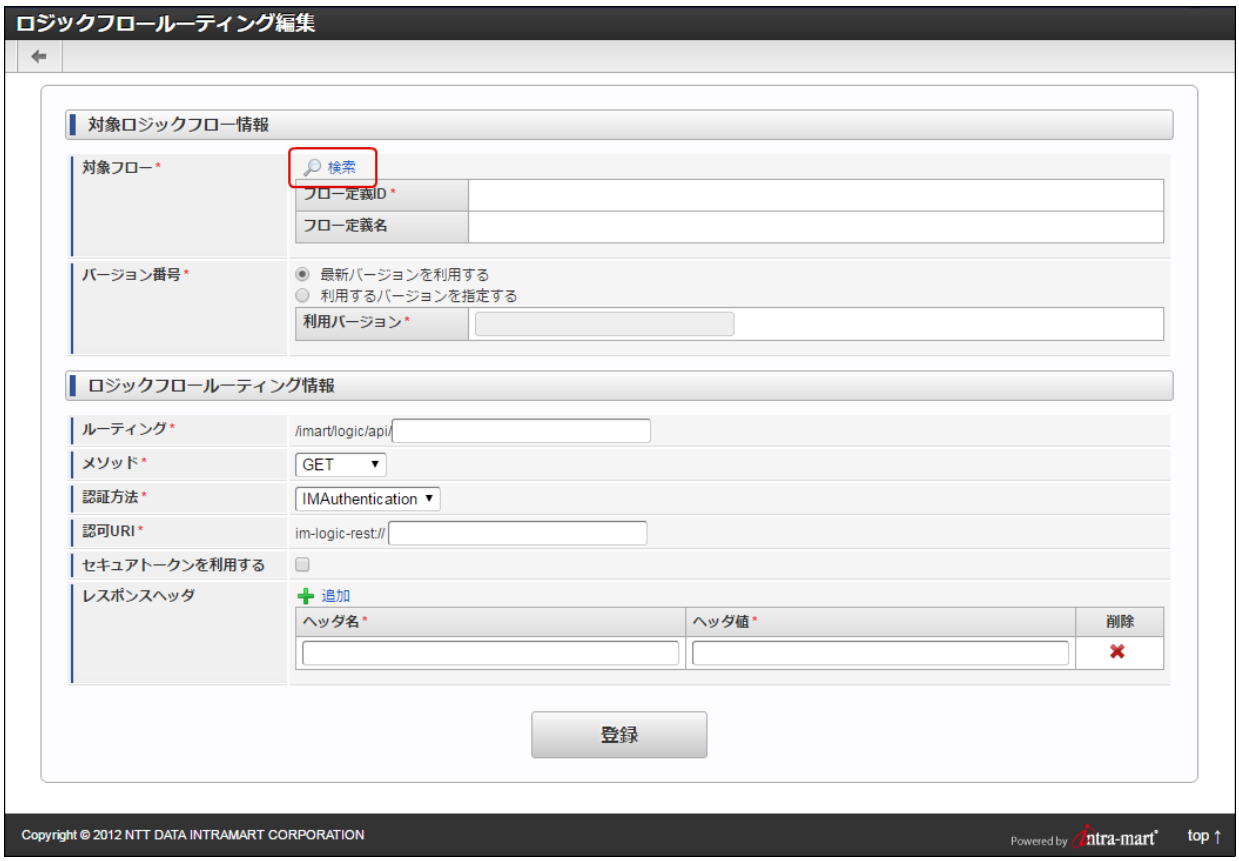

図：フロールーティング新規作成

2. ロジックフロールーティング編集画面が表示されます。




図：ロジックフロールーティング編集画面

3. 作成するフロールーティングが対象とするロジックフローを設定するために、対象フローの検索リンクをクリックします。




図：ロジックフローの検索

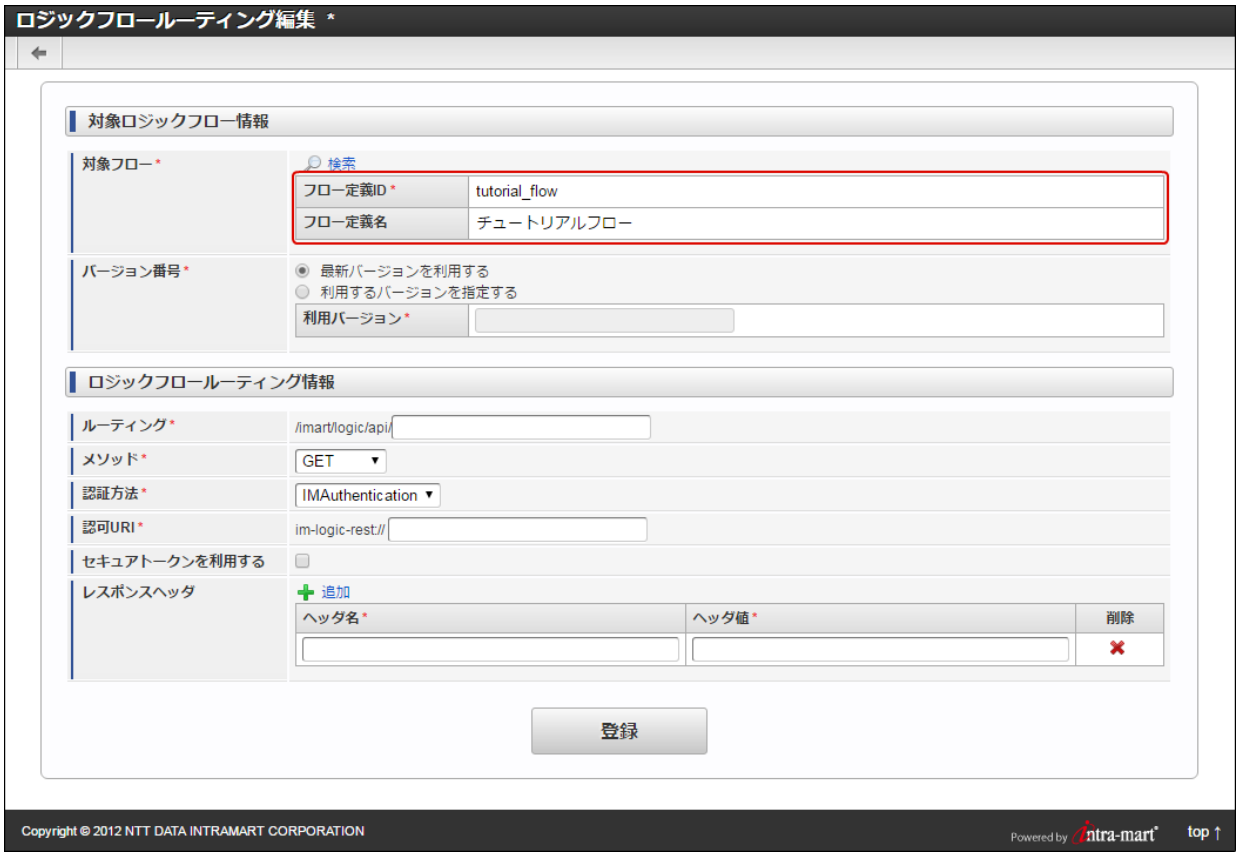
4. ロジックフロー定義検索画面がポップアップします。


図：ロジックフロー定義検索画面

5. 一覧の中から、フロー定義IDが「tutorial_flow」の行の「選択」チェックボックスにチェックをいれます。

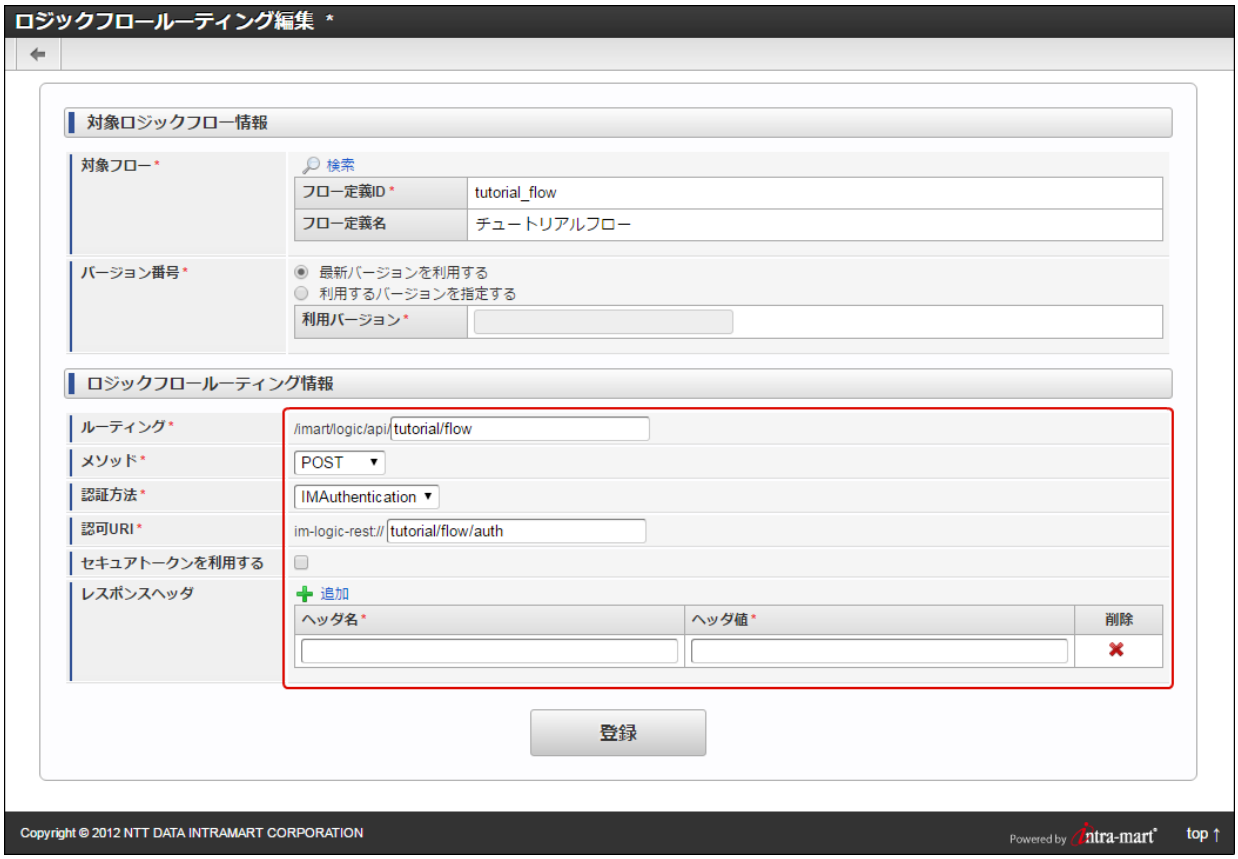

図：ロジックフローの選択

6. 検索画面下部の「決定」をクリックします。
7. 検索画面が閉じられ、対象フローの項目に、選択したフローカテゴリの情報が入力されます。




図：ロジックフロー情報の設定

8. ロジックフロールーティング情報の各項目に以下の値を入力します。
   - ルーティング「tutorial/flow」
   - メソッド「POST」
   - 認証方法「IMAuthentication」
   - 認可URI「tutorial/flow/auth」




図：ロジックフロールーティング情報の定義

9. 「登録」をクリックします。

### 作成したフロールーティングを確認する

正常にフロールーティングが新規作成された場合、正常に作成された旨のメッセージと共に、ロジックフロールーティング一覧画面へ遷移します。
ロジックフロールーティング一覧画面から、作成したフロールーティングが確認できます。


図：作成したフロールーティングの確認

次章「*ルーティングの認可を設定する*」では、作成したルーティングに対して認可を設定します。

## ルーティングの認可を設定する

次に、作成したフロールーティングに対する認可を設定します。

- フロールーティングの認可設定画面を開く
- フロールーティングの認可設定を行う

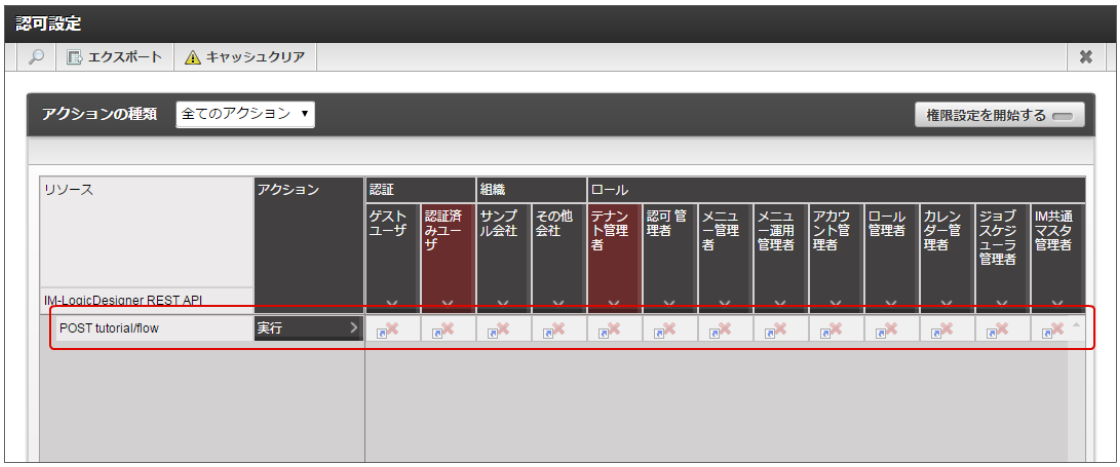
### フロールーティングの認可設定画面を開く

フロールーティングの認可設定画面は、ロジックフロールーティング一覧画面から開いて行います。

1. 「サイトマップ」→「LogicDesigner」→「ルーティング定義一覧」から、ロジックフロールーティング定義一覧を開きます。
2. 一覧の中から、本チュートリアルで作成したルーティング「tutorial/flow」の行の認可アイコンをクリックします。


図：認可アイコン

3. ルーティング「tutorial/flow」に対する認可設定画面がポップアップで表示されます。
   認可の表示名は、設定したメソッド＋ルーティングによって自動で決定されます。




図：認可設定画面

コラム

認可設定が行えない場合

実際の運用環境において、フロールーティングの認可を行う権限がない場合があります。
そのような状況で認可アイコンをクリックした場合、以下の様な警告メッセージが表示されます。

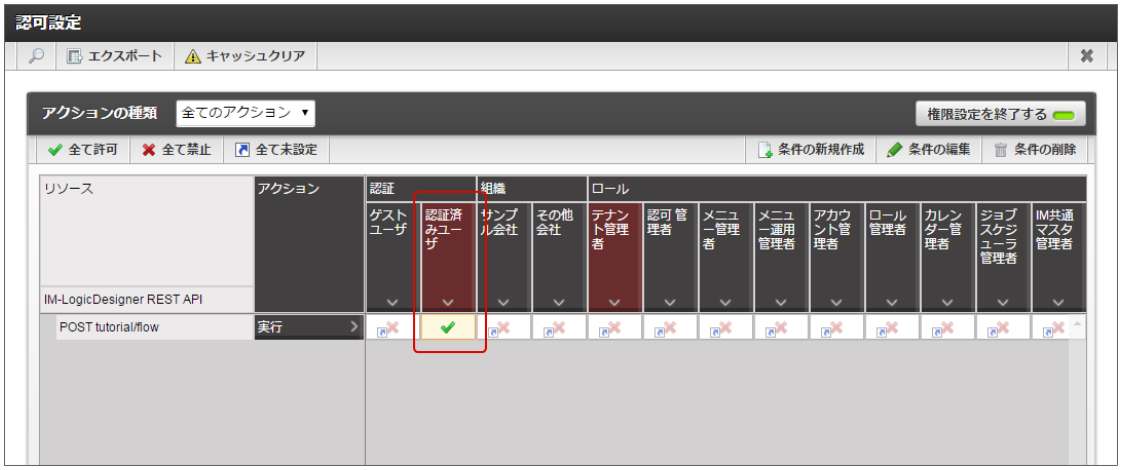

図：対象のフロールーティングの認可が行えない場合の警告メッセージ

警告メッセージが表示された場合、フロールーティングの認可を行う権限が操作を行ったユーザに対して付与されているかを確認してください

## フロールーティングの認可設定を行う

作成したフロールーティングに対して認可設定を行います。

初期状態では、フロールーティングの認可設定は全ての対象者条件に対して「禁止」が設定されます。
本チュートリアルでは対象者条件の「認証済みユーザ」に対して、「許可」を設定します。



図：認証済みユーザへ認可設定

認可設定画面の基本的な利用方法は「テナント管理者操作ガイド」 - 「認可を設定する」を参照してください。

> **注意**
>
> フロールーティングの認可と、ロジックフローの実行に対する認可
>
> 本章で説明した「フロールーティングの認可」は、ロジックフロー自体への認可ではありません。
>
> 本チュートリアルで行った認可設定は、フロールーティングで定義した「http://<HOST>:<PORT>/<CONTEXT_PATH>/logic/api/tutorial/flow」というURLへのアクセスに対する認可設定です。
> フロールーティングの対象となる ロジックフロー「tutorial_flow」の実行に対する認可設定ではありません。

以上で、フロールーティングに対する認可の設定が完了しました。

次章「*Swagger(SPEC)から実行する*」では、作成したルーティングをSwaggerを利用して、実際にロジックフローを実行します。

# Swagger(SPEC)から実行する

次に、Swaggerを利用して作成したフロールーティングを呼び出し、ロジックフローを実行します。



## Swaggerとは

Swaggerとは、REST APIに対するドキュメントの標準仕様であり、ドキュメントフォーマッター、および、ビューワーを提供するOSSです。
IM-LogicDesignerでは、作成したフロールーティングの仕様情報をSwaggerに対応した形式で出力します。

> **コラム**
>
> Swaggerについて
>
> Swaggerについての詳細は、以下のリンクを参照してください。
>
> - swagger.io
> - swagger(github)

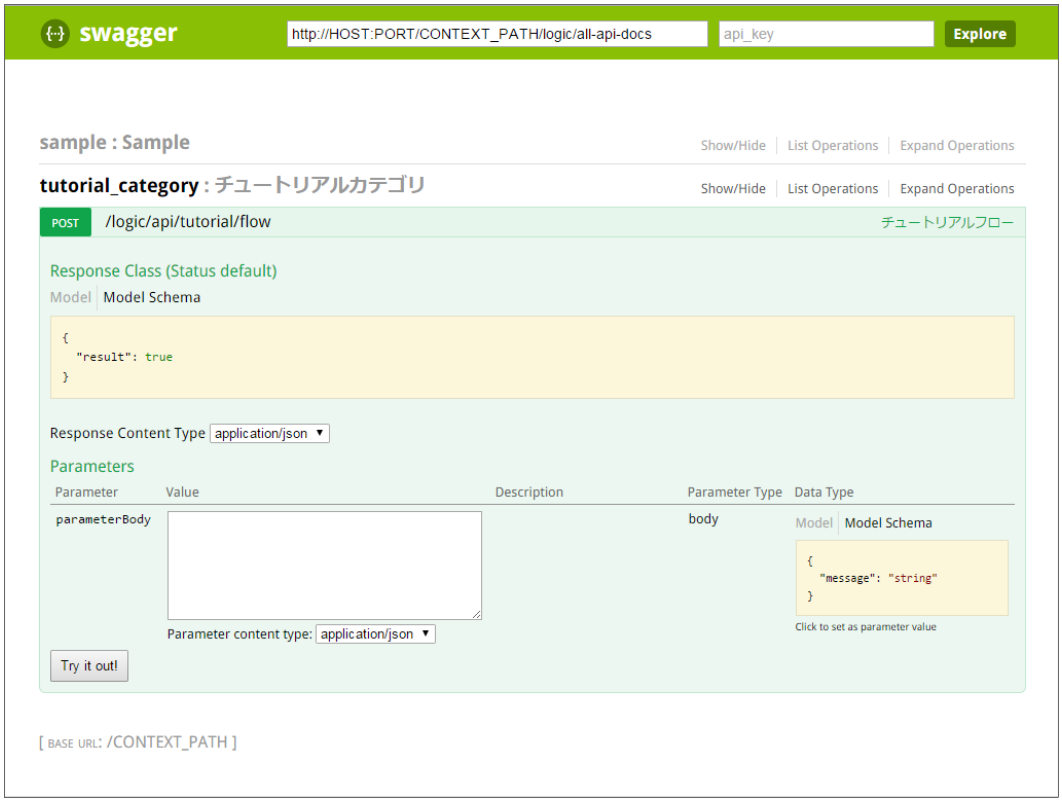
## Swaggerを開く

ロジックフロールーティング一覧画面では、ルーティング定義をもとに、Swagger上で対象のルーティングを実行するためのページへのリンクを提供します。

1. 「サイトマップ」→「LogicDesigner」→「ルーティング定義一覧」から、ロジックフロールーティング定義一覧を開きます。
2. 一覧の中から、本チュートリアルで作成したルーティング「tutorial/flow」の行のSPECアイコンをクリックします。

図：SPECアイコン

3. Swaggerの提供するREST APIビューワーの画面が新しいウィンドウで表示されます。

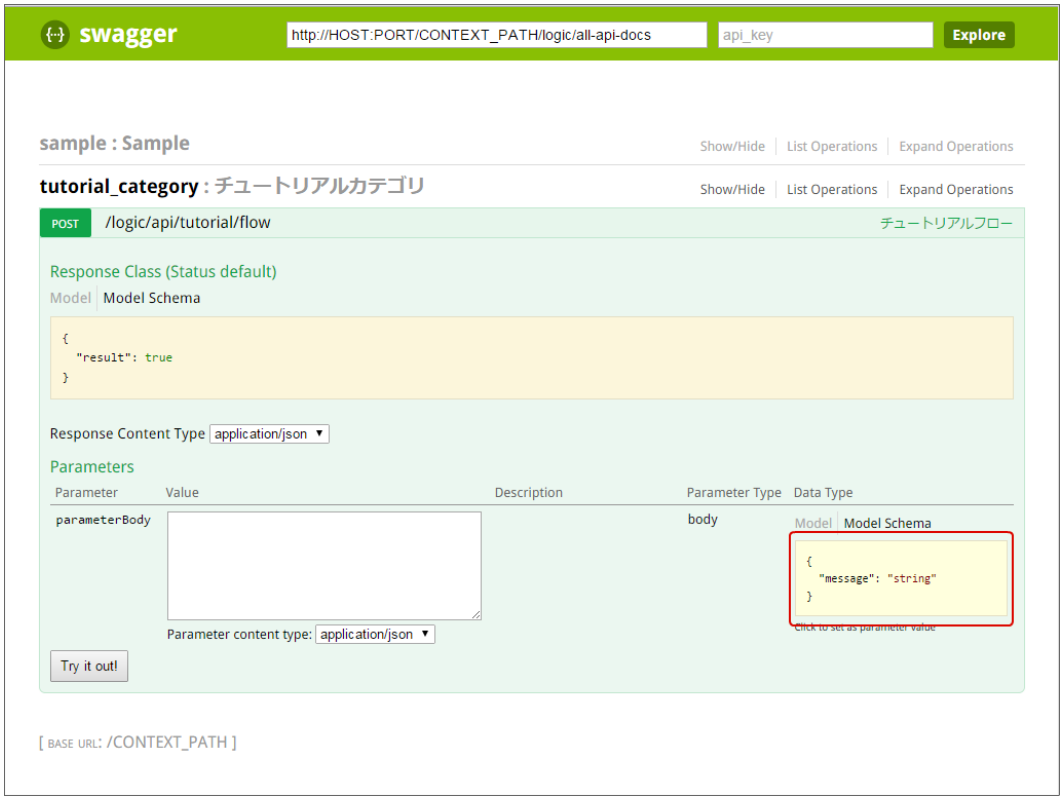



図：Swagger REST APIビューワー

## Swaggerからロジックフローを実行する

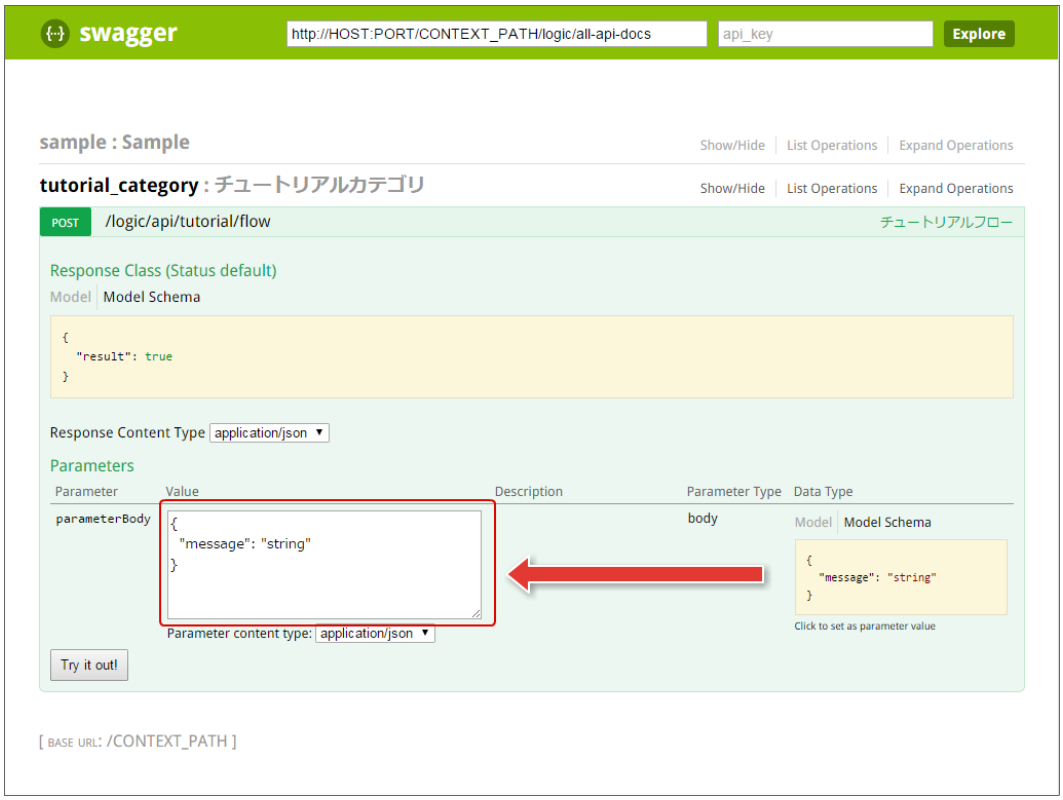
表示されたSwaggerの画面より、フロールーティングによって利用可能となったREST APIを介して、作成したロジックフローを実行します。

1. 画面右下のModel Schemaのテキストフィールドをクリックします。



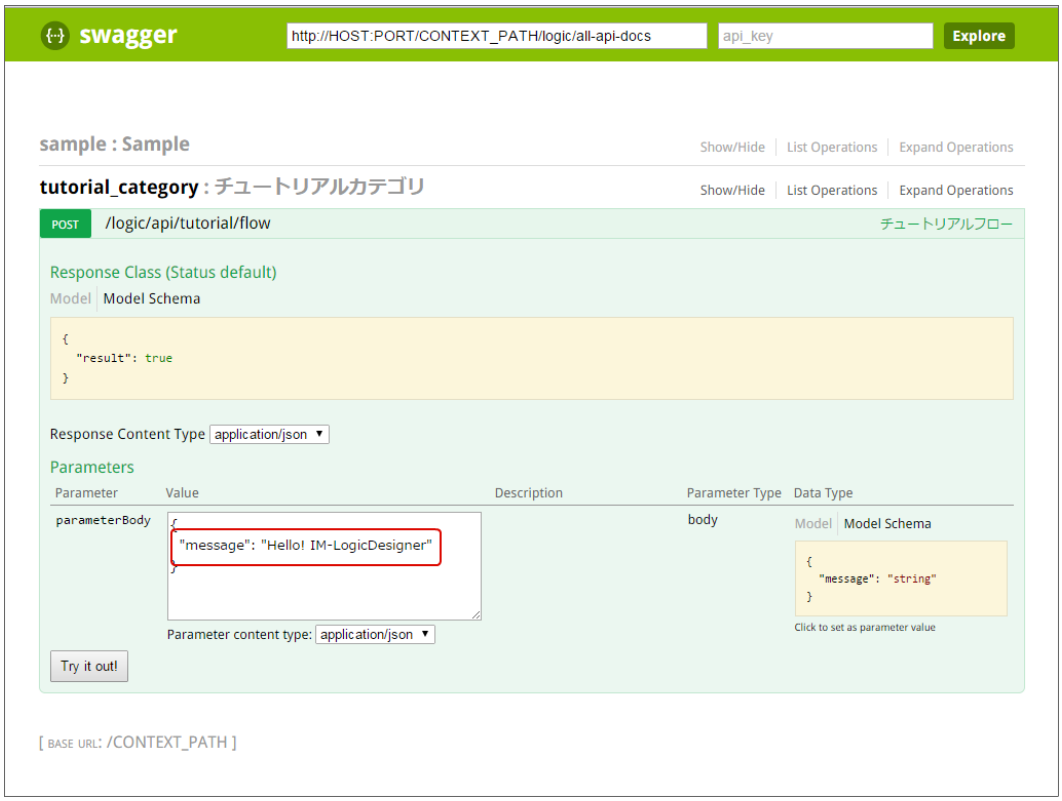

図：Model Schemaフィールド

2. 自動で、このREST APIを実行するために必要なparameterBodyのテンプレートが挿入されます。



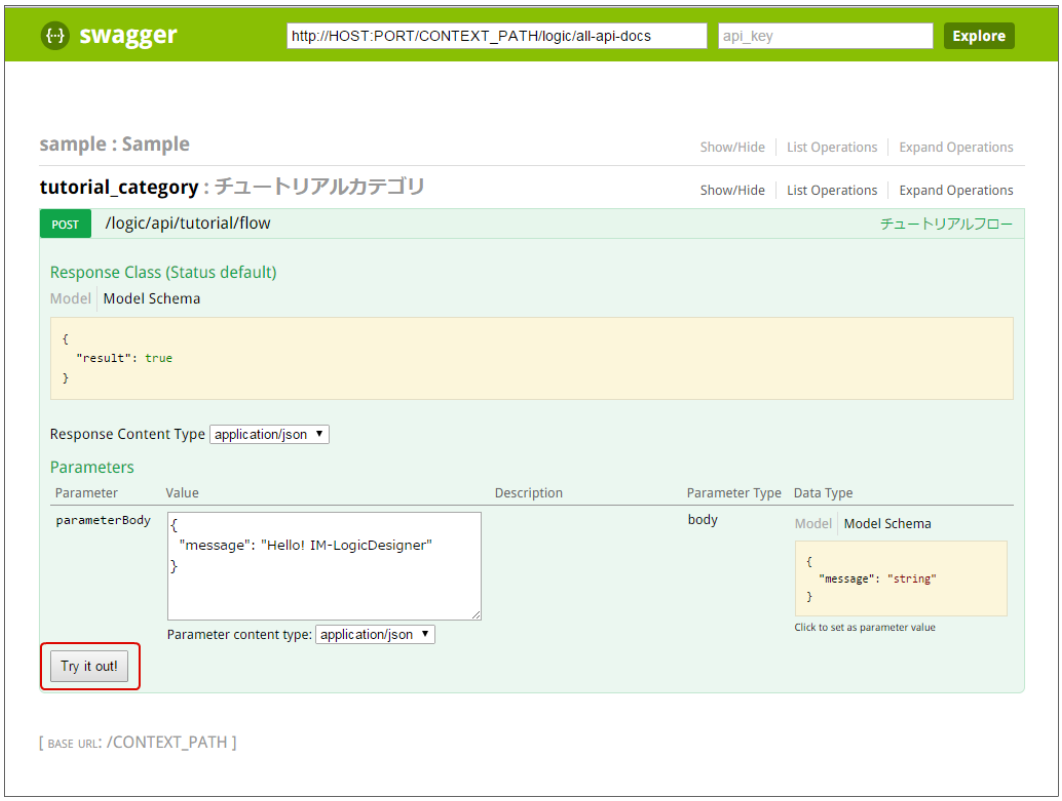

図：parameterBodyへの反映

3. 挿入されたテンプレートの以下の項目を編集します。
   - message「"Hello! IM-LogicDesigner"」



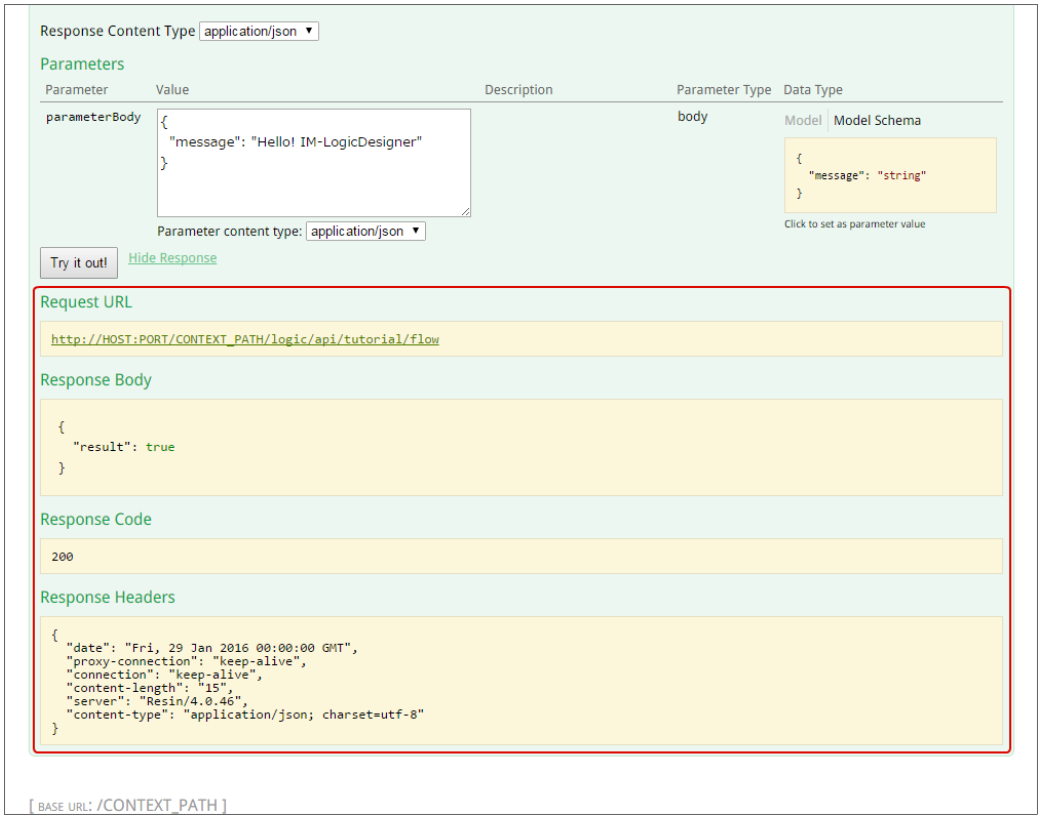

図：Valueの編集

4. 画面左下の「Try it out!」をクリックします。




図：「Try it out」のクリック

5. 実行結果が表示されます。




図：実行結果

以上で、Swaggerを利用したフロールーティングの呼び出し、および、ロジックフローを実行が完了しました。

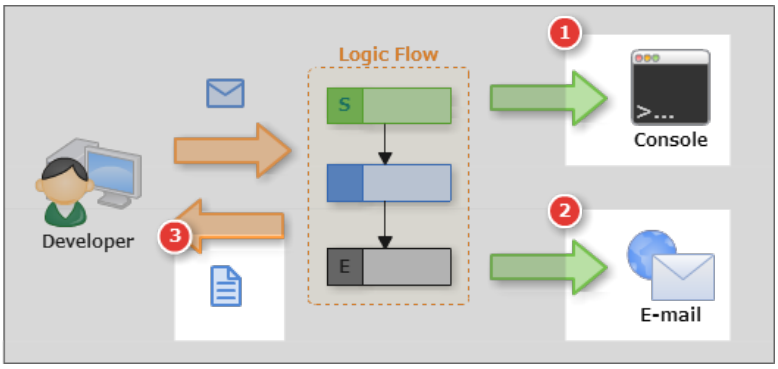
次章「*結果を確認する*」では、本章で実行したロジックフローの実行結果を確認します。

## 結果を確認する

次に、実行したロジックフローの実行結果を確認するとともに、その結果が想定するものであるかを検証します。

## 実行結果として確認する内容

実行結果を確認する前に、本チュートリアルで作成したロジックフローの**実行結果**にあたるものが何であるかを「*チュートリアルの概要（作成物のイメージ）*」で提示した動作イメージをもとに再確認します。
動作イメージにおける、ロジックフローの実行結果は以下にあたります。




図：ロジックフローの結果に該当する部分

1. 「ログ出力」タスクによる、コンソールへ出力されるログ情報
2. 「テキストメール送信」タスクによる、送信メール
3. ロジックフローが「出力値」として定義した出力情報

実行結果について、それぞれ確認していきます。

## 「ログ出力」タスクの結果

はじめに「ログ出力」タスクによる、コンソールへのログ出力結果を確認します。

コンソールへのログ出力結果として、stdout.logに以下の様なログが出力されていることを確認してください。

```
...
[INFO] TutorialLogger - [] Hello! IM-LogicDesigner
...
```

具体的には、出力されたログについて以下を確認してください。

- ログレベル
  - 「*タスクのプロパティを設定する*」-「*プロパティを設定する*」の「ログ出力」タスクで設定した「ログレベル」
- ロガー名
  - 「*タスクのプロパティを設定する*」-「*プロパティを設定する*」の「ログ出力」タスクで設定した「ロガー名」
- 出力内容
  - 「*Swagger(SPEC)から実行する*」-「*Swaggerからロジックフローを実行する*」の中のmessageパラメータに定義した文字列

これにより、出力されたログが「*タスクのプロパティを設定する*」で定義したプロパティの値であり、「*マッピング設定を行う*」において、「ログ出力」タスクにマッピングした値であることが確認できました。

> **コラム**
>
> コンソールへのログが出力されない
>
> コンソールへのログが出力されていない場合、これまでの実施内容の見直しと共に、ログ設定の見直しを行ってください。
> 「ログ出力」タスクは、intra-mart Accel Platformのログ設定に準拠します。
>
> ログ設定の詳細は、「ログ仕様書」を参照してください。

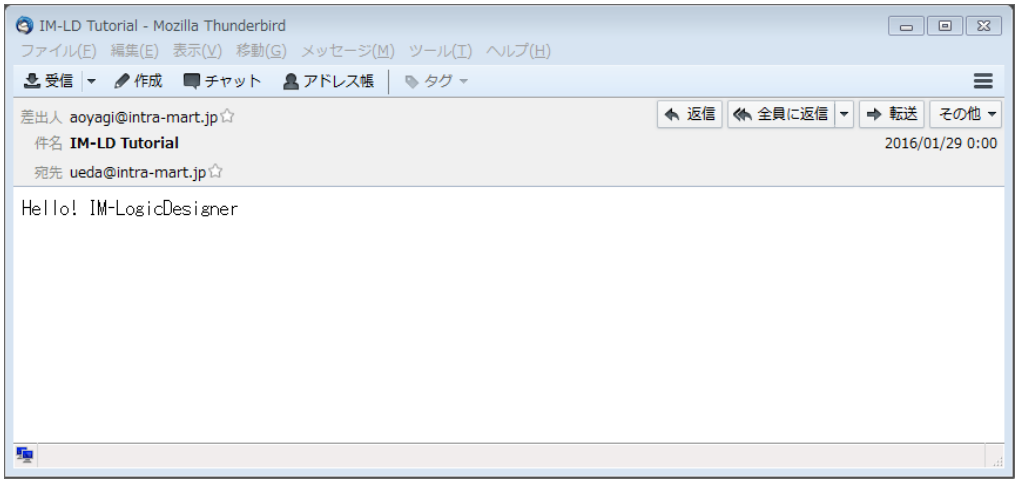
## 「テキストメール送信」タスクの結果

次に「テキストメール送信」タスクによる、送信メールについて実行結果を確認します。

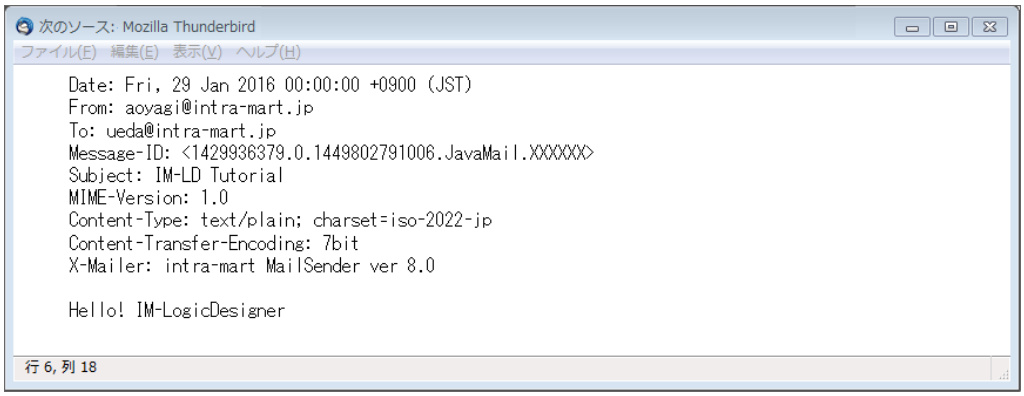
なお、「*チュートリアルの概要（作成物のイメージ）*」にてメールの送信環境については任意としています。
実行結果の確認は、環境に応じて「*メール送信環境が整備済み*」「*メール送信環境が未整備*」のどちらか一方を参照してください。

### メール送信環境が整備済み

メール送信環境が整っている場合、「*定数値を定義する*」-「*定数値の定義*」の中の「メールの宛先」として定義したユーザの、 メールアドレスに対してメールが送信されていることを確認します。
ロジックフローを実行した結果として、送信されたメールの例は以下の通りです。




図：送信されたメール（Thunderbird）




図：送信されたメール - ソース（Thunderbird）

送信されたメールの以下の項目を確認してください。

- 差出人（from）
  - 「*定数値を定義する*」-「*定数値の定義*」の中の「メール送信者」として定義したユーザのメールアドレス
- 件名（subject）
  - 「*定数値を定義する*」-「*定数値の定義*」の中の「メールの題名」として定義した文字列
- 宛先（to）
  - 「*定数値を定義する*」-「*定数値の定義*」の中の「メールの宛先」として定義したユーザのメールアドレス
- 本文（body）
  - 「*Swagger(SPEC)から実行する*」-「*Swaggerからロジックフローを実行する*」の中のmessageパラメータに定義した文字列

これにより、送信されたメールの詳細が「*マッピング設定を行う*」において、「テキストメール送信」タスクにマッピングした値であることが確認できました。

### メール送信環境が未整備

メール送信環境が整っていない場合、ロジックフローを実行した結果として「正常にロジックフローが実行完了」していることを確認してください。

通常のビジネスロジックにおいて、指定したアドレスへのメール送信が失敗した場合は、それに伴う例外が発生し、処理が中断されます。
IM-LogicDesignerでは、そうしたメールの送信処理の例外をエラーとして処理を中断するか否かを制御することができます。

具体的には、「*タスクのプロパティを設定する*」-「*プロパティを設定する*」で実施した「テキストメールの送信」タスクにおける、「送信失敗時

にエラーとする」プロパティが該当します。
本チュートリアルではこの項目のチェックボックスをオフにすることで、メール送信が失敗した場合でもフローが正常に進むようにしています。

> **注意**
>
> メール送信失敗時の取り扱いについて
>
> 実際の業務や、それに伴うビジネスロジック実装に際しては、要件に合わせてメール送信の失敗の取り扱いを決定してください。

### ロジックフローの「出力値」

最後にロジックフローの「出力値」として定義した出力情報を確認します。
出力値の出力情報は、「*Swagger(SPEC)から実行する*」-「*Swaggerからロジックフローを実行する*」で説明したSwaggerの実行結果から確認できます。

具体的には、Swaggerの実行結果のうち、Response Bodyが「出力値」の出力情報にあたります。


図：実行結果 - Response Body

Response Bodyの値が以下であることを確認してください。

```
{
  "result": true
}
```

これは、resultというキー名のパラメータが、trueという値であることを示しています。

この出力情報から、「*入出力設定を定義する*」で定義した出力値の設定に従ったフォーマットであり、格納された値が「*マッピング設定を行う*」において「終了」制御要素にマッピングした値であることが確認できました。

> **コラム**
>
> Swaggerの実行結果のその他の項目
>
> Swaggerの実行結果について、今回触れていないその他の項目についての詳細は以下の通りです。
>
> - **Request URL**
>  - 実行されたREST APIのアドレス（リクエストが行われたアドレス）
> - **Response Code**
>  - Request URLへのリクエスト結果として返されたHTTPレスポンスコード
> - **Response Header**
>  - Request URLへのリクエスト結果として返されたHTTPレスポンスヘッダ

以上で、ロジックフローの実行結果とその検証が完了しました。

次章「*データをエクスポートする*」では、最後にこれまで作成してきたロジックフローおよびフロールーティングのエクスポートを行います。

## データをエクスポートする

次に、ロジックフロー、および、フロールーティングのエクスポート方法を説明します。

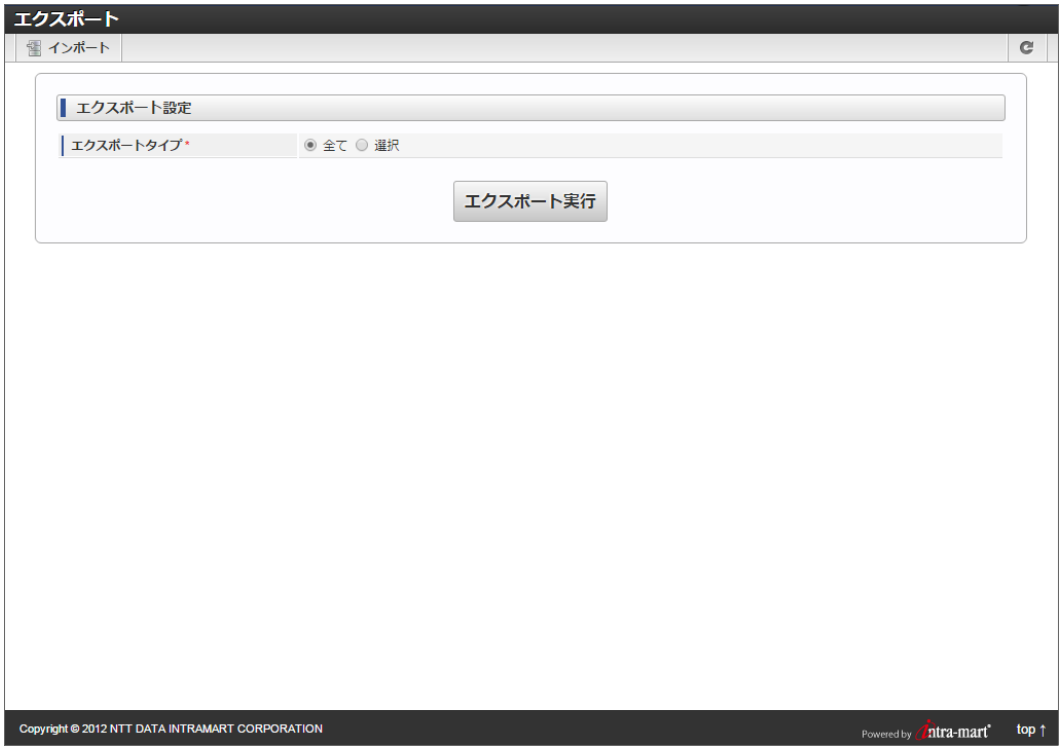


## エクスポート画面を表示する

1. 「サイトマップ」→「LogicDesigner」→「エクスポート」をクリックします。


図：サイトマップ

2. エクスポート画面が表示されます。




図：エクスポート画面

## エクスポートを実行する（全て）

IM-LogicDesignerでは、二種類のエクスポート方法を提供しています。
初めに、現在のintra-mart Accel Platform上にある全てのIM-LogicDesignerに関する情報をエクスポートする方法を説明します。

1. 「サイトマップ」→「LogicDesigner」→「エクスポート」から、エクスポート画面を表示します。
2. エクスポート設定のエクスポートタイプについて、「全て」を選択します。


図：エクスポートタイプの選択

3. 「エクスポート実行」をクリックします。


図：エクスポートの実行

4. ZIP形式に圧縮されたエクスポートデータが出力されます。

## エクスポートを実行する（対象を選択する）

次に、エクスポート対象を選択してエクスポートする方法を説明します。
ここでは、本チュートリアルで作成したロジックフローおよびフロールーティングを対象としてエクスポートを行います。

1. 「サイトマップ」→「LogicDesigner」→「エクスポート」から、エクスポート画面を表示します。
2. エクスポート設定のエクスポートタイプについて、「選択」を選択します。


図：エクスポートタイプの選択

3. ロジックフロー、フロールーティング、ユーザ定義の選択領域が表示されます。


図：エクスポート対象選択画面の表示

4. ロジックフローのエクスポート対象の選択において、フロー定義IDが「tutorial_flow」の行の「選択」チェックボックスにチェックをいれます。


図：選択（ロジックフロー）

5. フロールーティングのエクスポート対象の選択において、ルーティングが「tutorial/flow」の行の「選択」チェックボックスにチェックをいれます。


図：選択（フロールーティング）

6. 「エクスポート実行」をクリックします。


図：エクスポートの実行

7. ZIP形式に圧縮されたエクスポートデータが出力されます。

以上でロジックフローおよびフロールーティングのエクスポートが完了しました。

> **コラム**
>
> エクスポートされるデータ
>
> エクスポートされるデータの詳細については、「IM-LogicDesigner仕様書」-「インポート・エクスポート」を参照してください。

お疲れ様でした。そして、おめでとうございます。
以上で「*基礎編 - ファースト・ステップ*」のチュートリアルは全て完了です。

これまでのチュートリアルを通して、以下のことを学びました。

- ロジックフローの作成方法
- フロールーティングの定義方法
- Swaggerを利用したロジックフローの実行方法
- 作成したデータのエクスポート方法

次章「*応用編 - より高度なフロー*」では、これまでのチュートリアルをベースとして、より複雑で高度なフローを定義するための手助けとなるチュートリアルを用意しています。

基礎編のチュートリアルとは違い、それぞれが独立した小さなチュートリアルのため、作成したいロジックフローの要件に合わせた章を選んでください。

> **コラム**
>
> 応用編を進める前に
>
> 応用編では、各章のチュートリアルに「*基礎編 - ファースト・ステップ*」で作成したロジックフロー、および、フロールーティングの資材を利用します。
> 基礎編のチュートリアル資材が無い場合は、「*付録*」-「*チュートリアルデータのアーカイブファイル*」-「*基礎編*」を参照し資材の準備を行ってください。

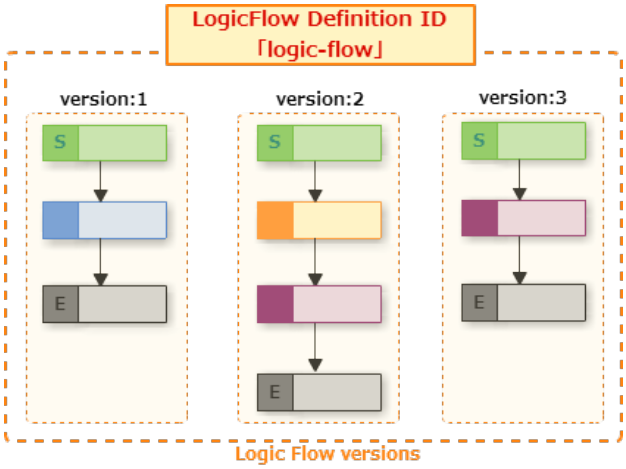
## ロジックフローのバージョン管理

この章では、ロジックフローのバージョン管理（履歴管理）について説明します。



### バージョンとは

ロジックフローのバージョンとは、ロジックフローの編集履歴を表す1単位です。
ロジックフローは、内部でバージョンの形式により編集履歴情報を複数保持する構造です。




図：ロジックフローは内部に複数のバージョンを保持している

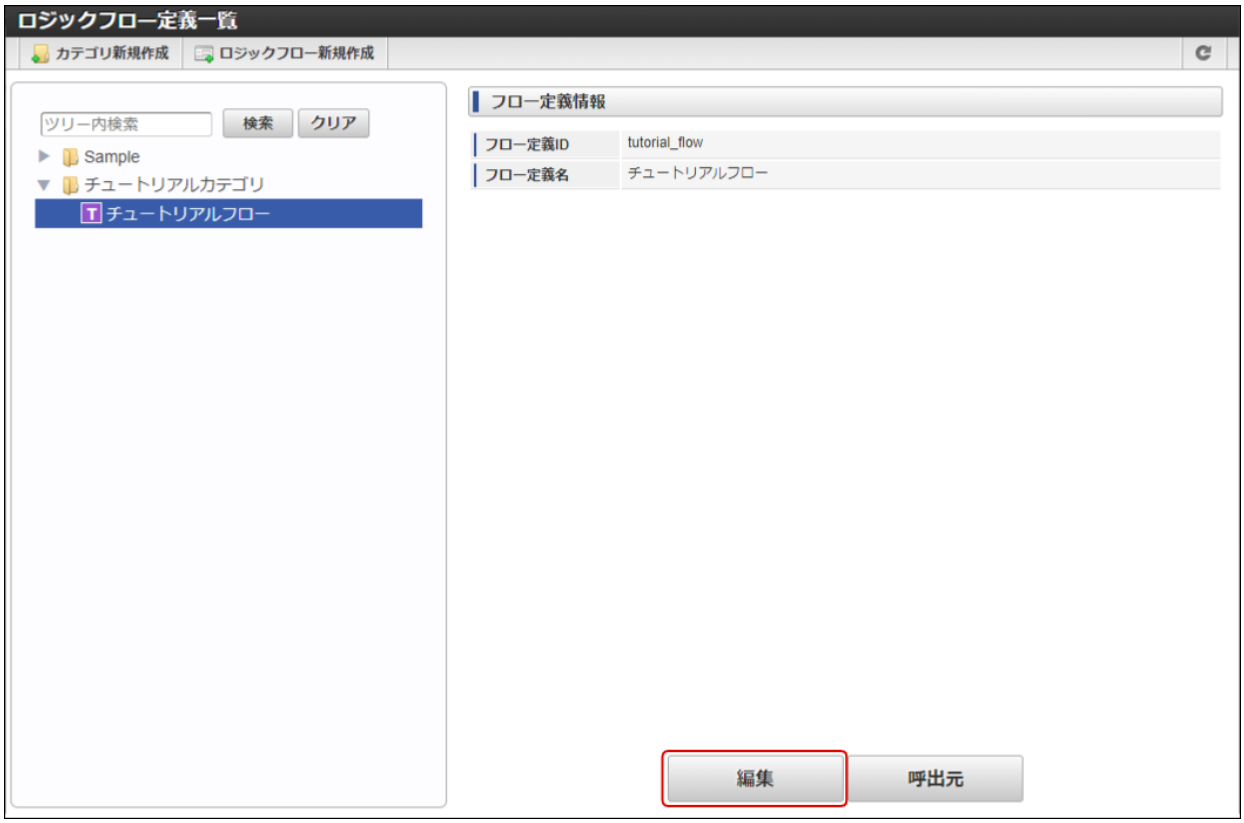
注意点として、ロジックフローを含むIM-LogicDesignerで扱うバージョンには「期間」という概念はありません。
他のintra-mart Accel Platformの機能が提供するバージョン管理のような、有効期間の管理といった用途での利用には適しません。

> **コラム**
>
> バージョンについての詳細
>
> ロジックフローを含むIM-LogicDesignerで扱うバージョンの詳細については、「IM-LogicDesigner仕様書」を参照してください。

### バージョン一覧画面を表示する

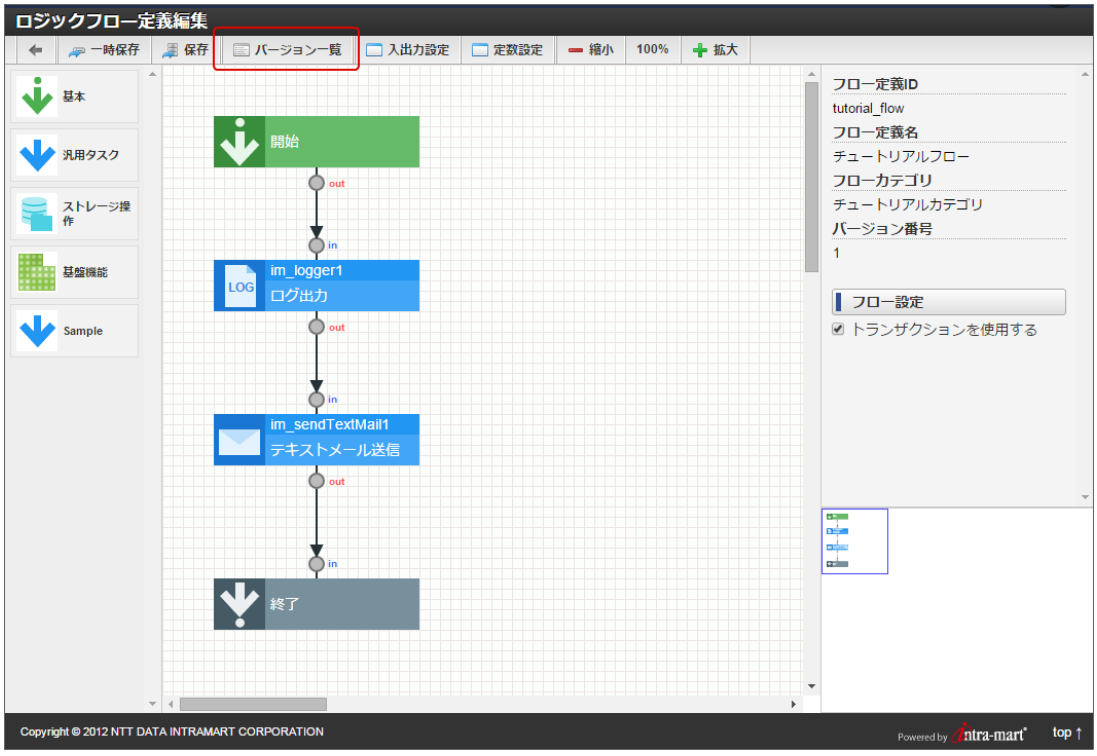
バージョン一覧画面は、ロジックフロー定義編集画面から開きます。

1. 「サイトマップ」→「LogicDesigner」→「フロー定義一覧」から、ロジックフロー定義一覧を開きます。
   左ペインのツリーから「チュートリアルカテゴリ」の開閉アイコンをクリックしカテゴリ配下を情報を表示します。
   フロー定義「チュートリアルフロー」を選択し編集ボタンをクリックします。



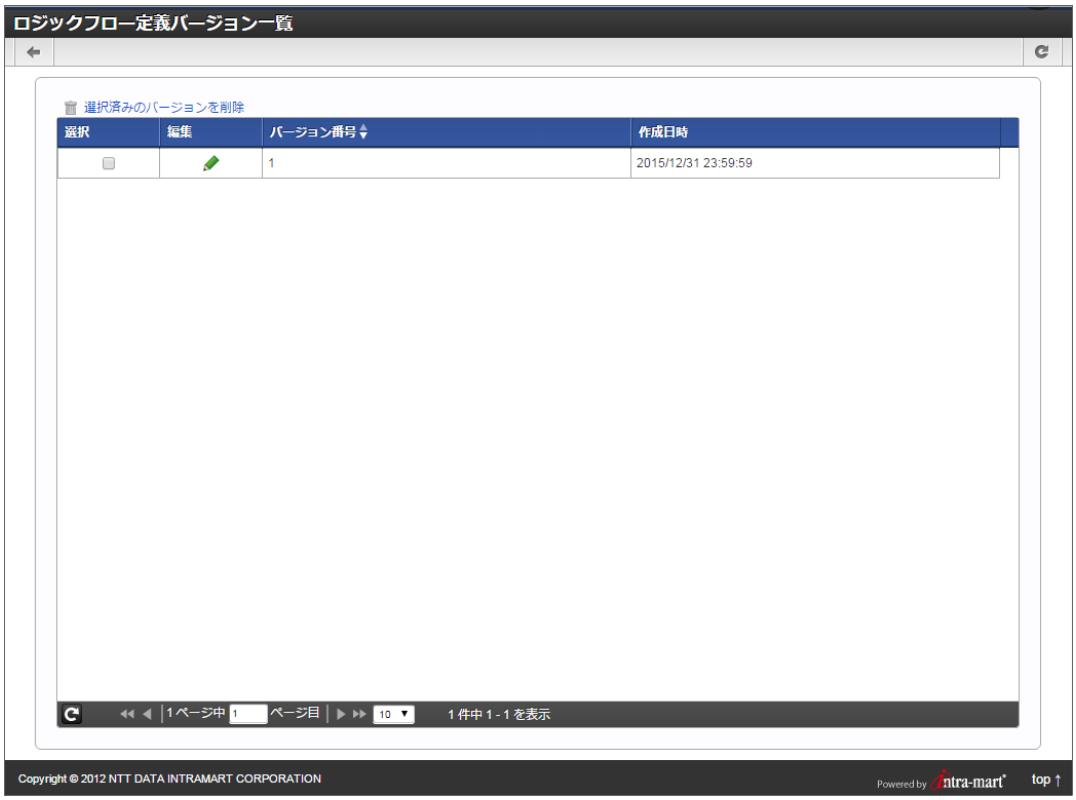

図：「チュートリアルフロー」編集ボタン

2. ロジックフロー定義編集画面上部、ヘッダ内の「バージョン一覧」をクリックします。




図：バージョン一覧をクリック

3. ロジックフロー定義バージョン一覧画面が表示されます。

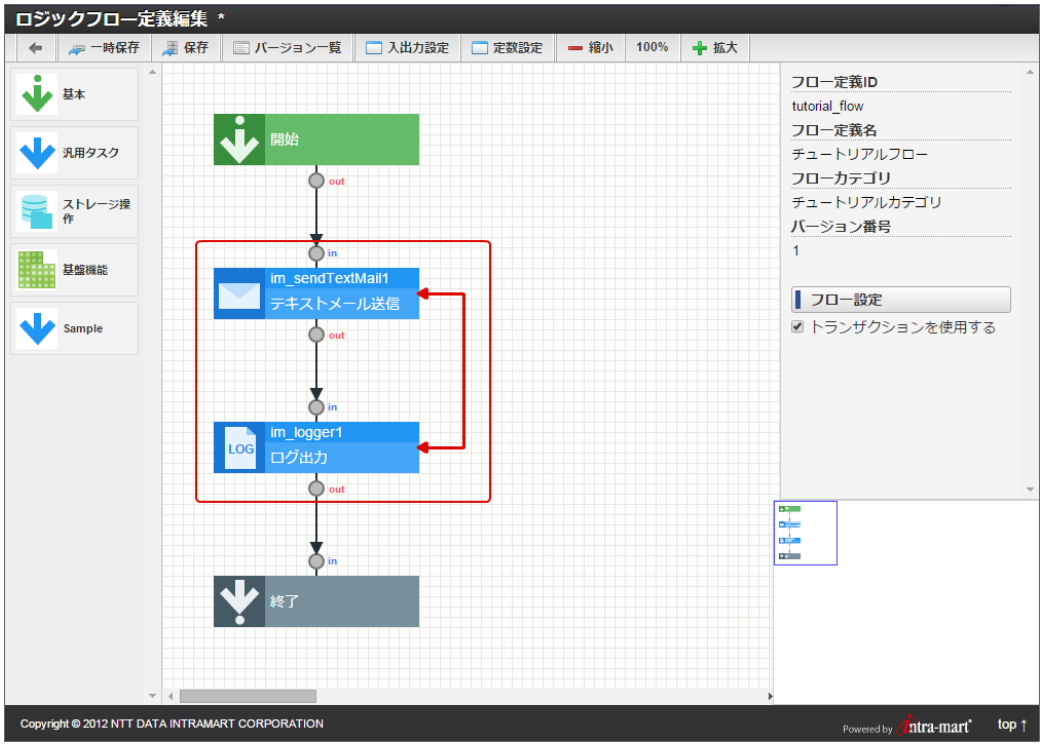



図：ロジックフロー定義バージョン一覧画面

## 新規にバージョンを作成する

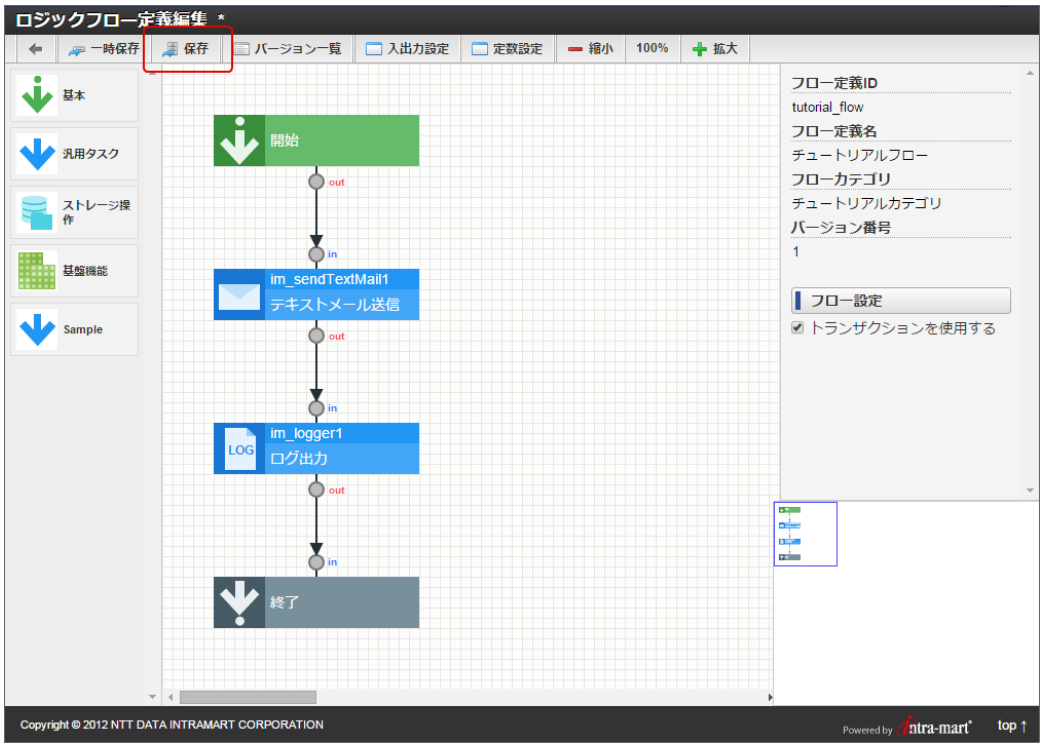
新しいバージョンを作成するには、ロジックフローを保存する際に新規バージョンを作成するオプションを指定します。

1. フロー定義「チュートリアルフロー」のロジックフロー定義編集画面を表示します。
2. 今回、バージョン間での違いを明確にするため、フローの一部を編集します。
   この例では、「ログ出力」タスクと「テキストメール送信」タスクの処理順を入れ替えています。



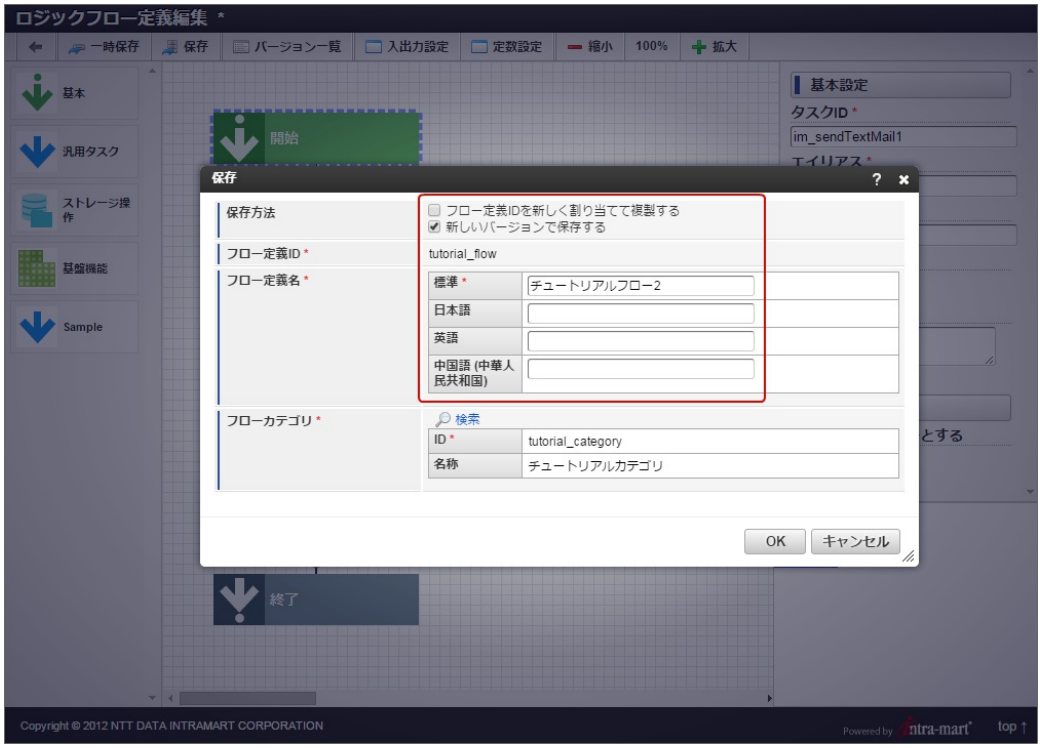

図：タスクの処理順の入れ替え

3. ロジックフロー定義編集画面上部、ヘッダ内の「保存」をクリックします。




図：保存をクリック

4. 各項目を以下のように入力します。
   - 保存方法
     - フロー定義IDを新しく割り当てて複製する - 「チェックボックス：オフ」
     - 新しいバージョンで保存する - 「チェックボックス：オン」
   - フロー定義名
     - 標準 - 「チュートリアルフロー2」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし



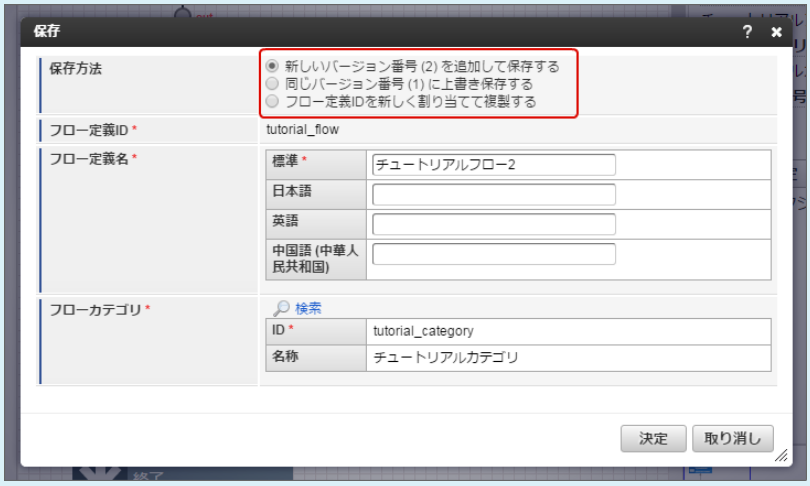

図：新しい保存内容の設定

コラム

2016 Spring(Maxima)以降での新規バージョン保存

IM-LogicDesignerでは2016 Spring(Maxima)以降、保存時の設定項目が一部変更されています。
本チュートリアルの「保存方法」と同様の保存を行うためには各項目を以下のように入力してください。

- 保存方法 - 「新しいバージョン番号（2）を追加して保存する」を選択する。

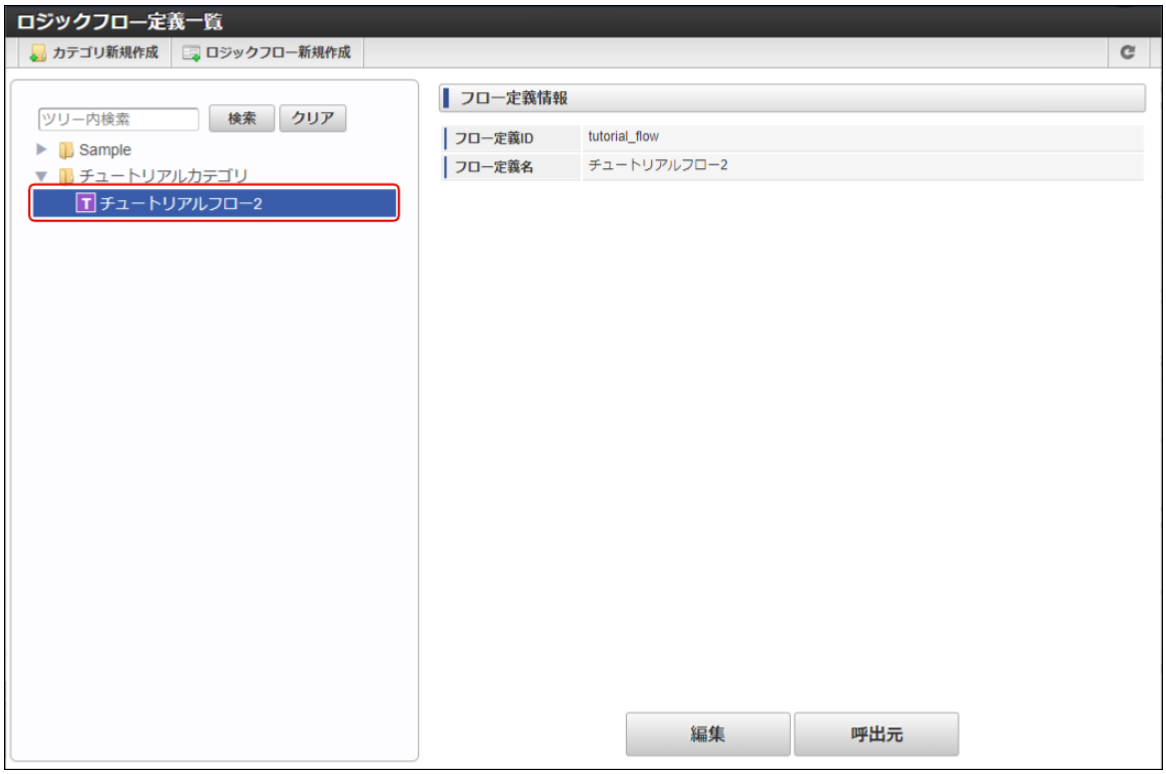



図：新しい保存設定

5. 保存画面右下のOKをクリックします。
6. 保存を確認するダイアログが表示されるので、OKをクリックします。
7. 保存が完了した旨のメッセージと共に、ロジックフロー定義一覧画面に遷移します。


図：保存完了後、ロジックフロー定義一覧画面へ遷移

8. 保存したフローの情報が更新されていることが確認できます。



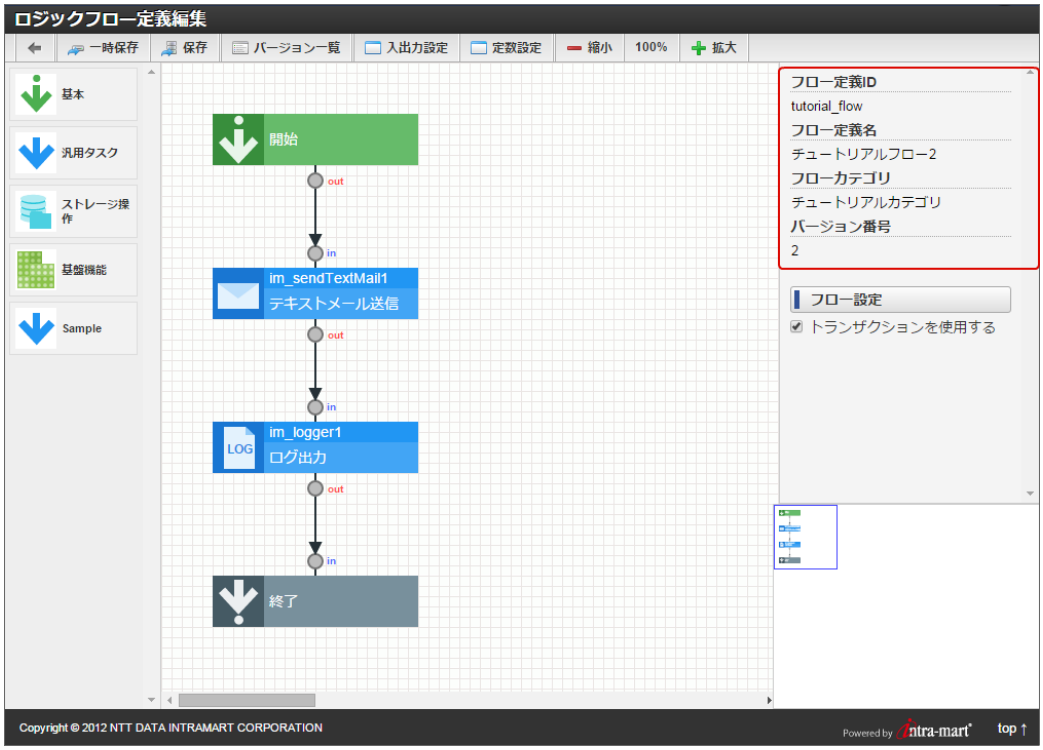

図：保存完了のロジックフロー定義

以上で、新規バージョンの作成が完了しました。

## バージョンを切り替える（任意のバージョンを編集する）

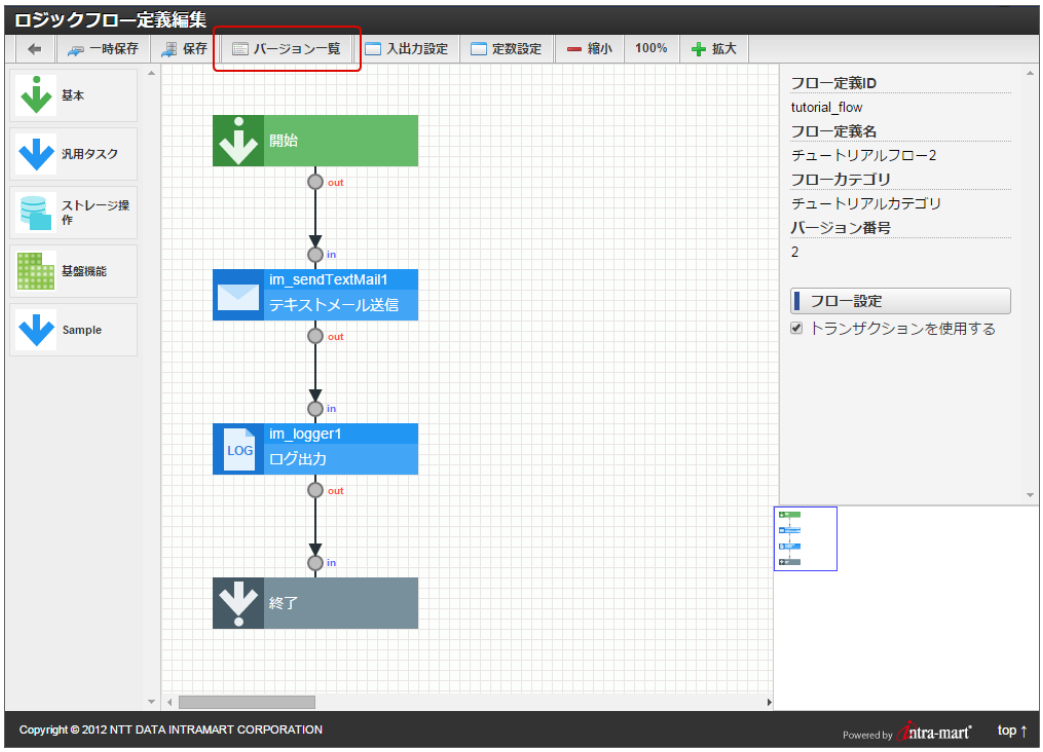
「*バージョン一覧画面を表示する*」から、新規に作成されたバージョンの確認と共に、バージョン単位で編集を行います。

1. フロー定義ID「tutorial_flow」のロジックフロー定義編集画面を表示します。
   ロジックフロー定義編集画面では、初期表示として対象フローの最新バージョンを表示することを確認してください。



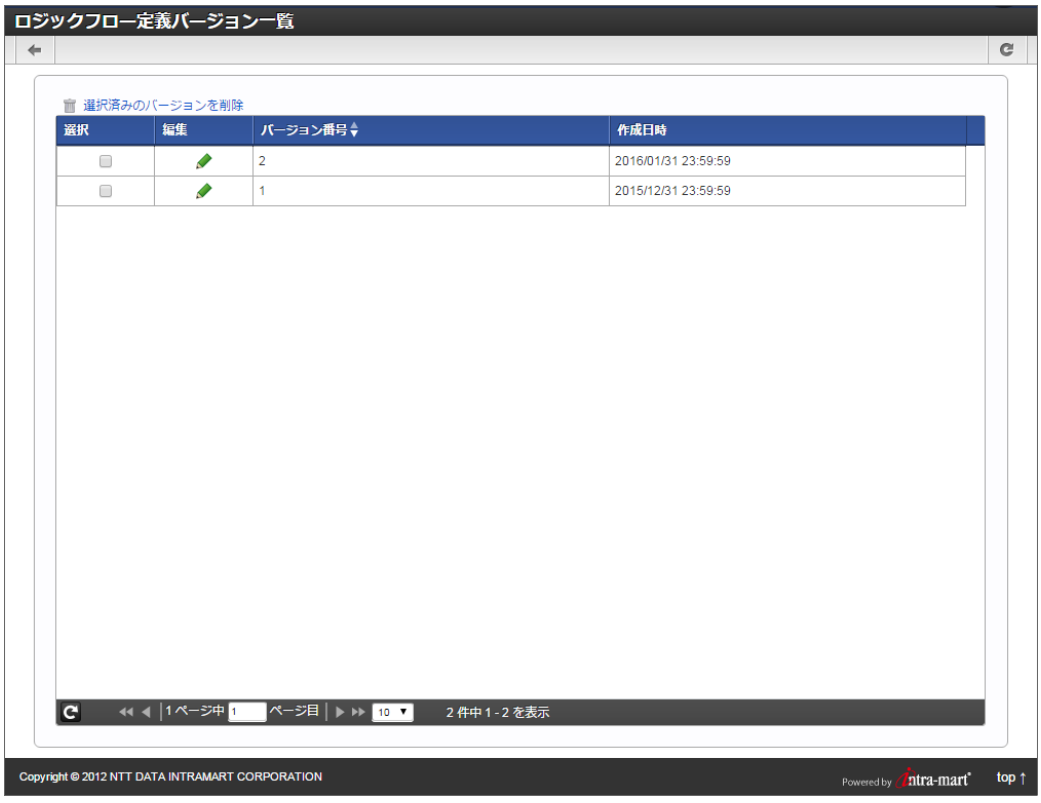

図：ロジックフロー情報の確認

2. ロジックフロー定義編集画面上部、ヘッダ内の「バージョン一覧」をクリックします。



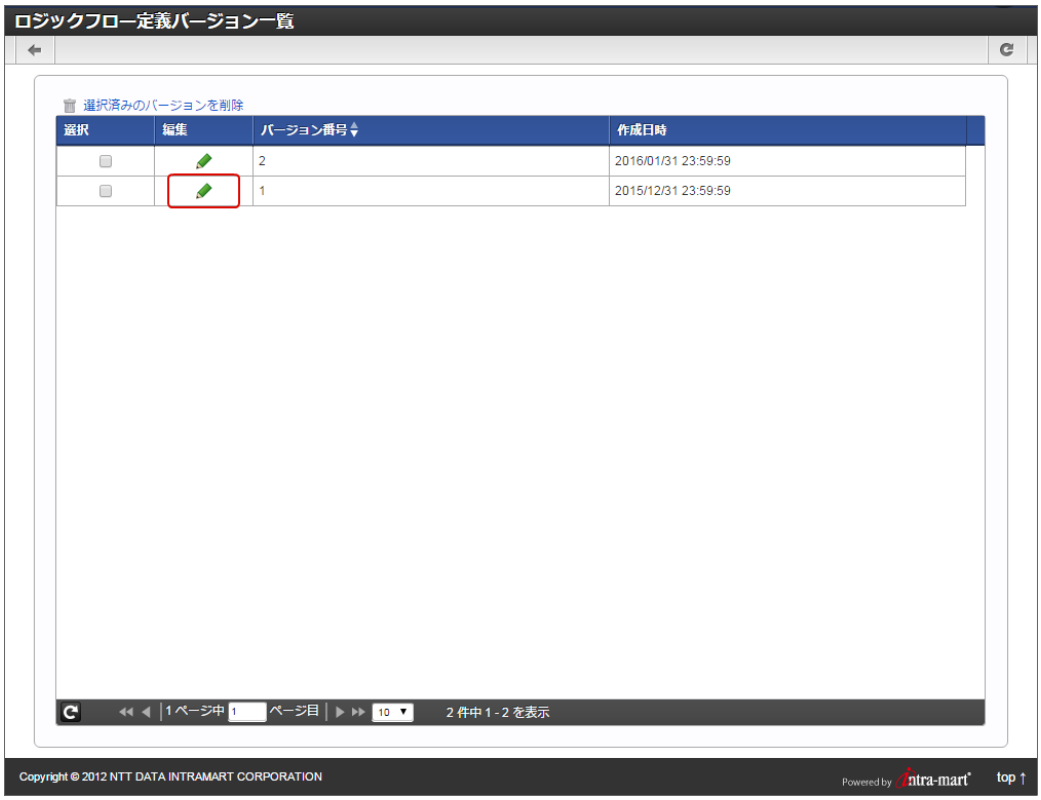

図：バージョン一覧をクリック

3. ロジックフロー定義バージョン一覧画面が表示されます。
   バージョン一覧画面に、新規作成時に割り振られたバージョン（1）と、先ほど新しく作成したバージョン（2）の2つがあることを確認してください。



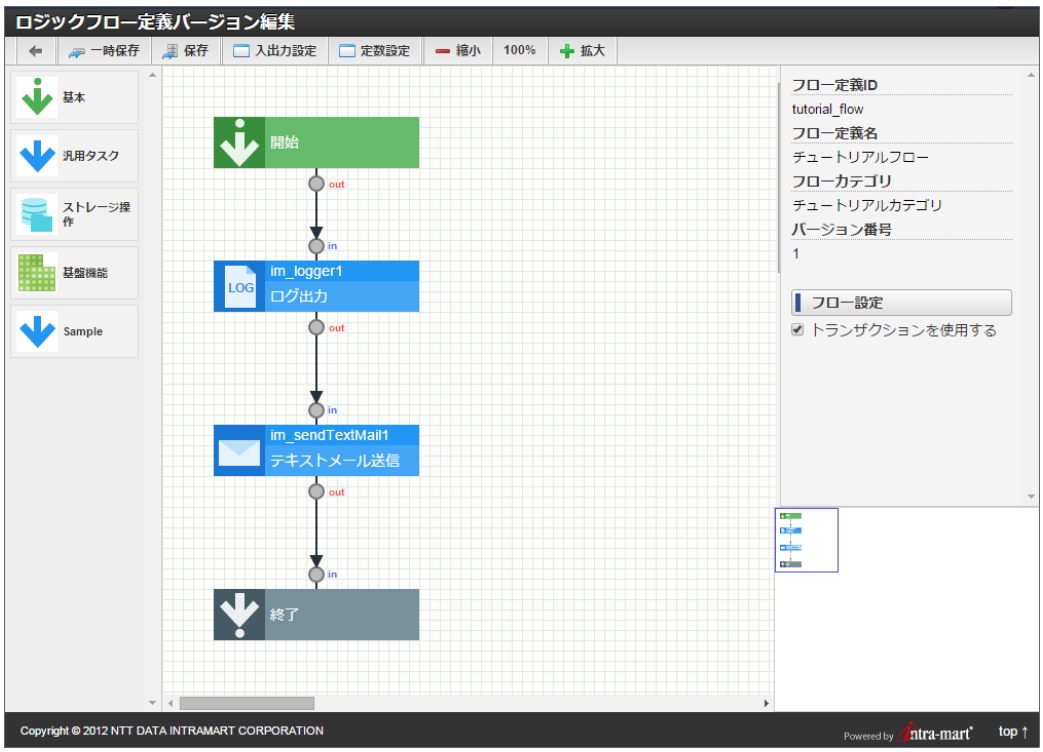

図：ロジックフロー定義バージョン一覧画面

4. バージョン番号「1」の行の編集アイコンをクリックします。



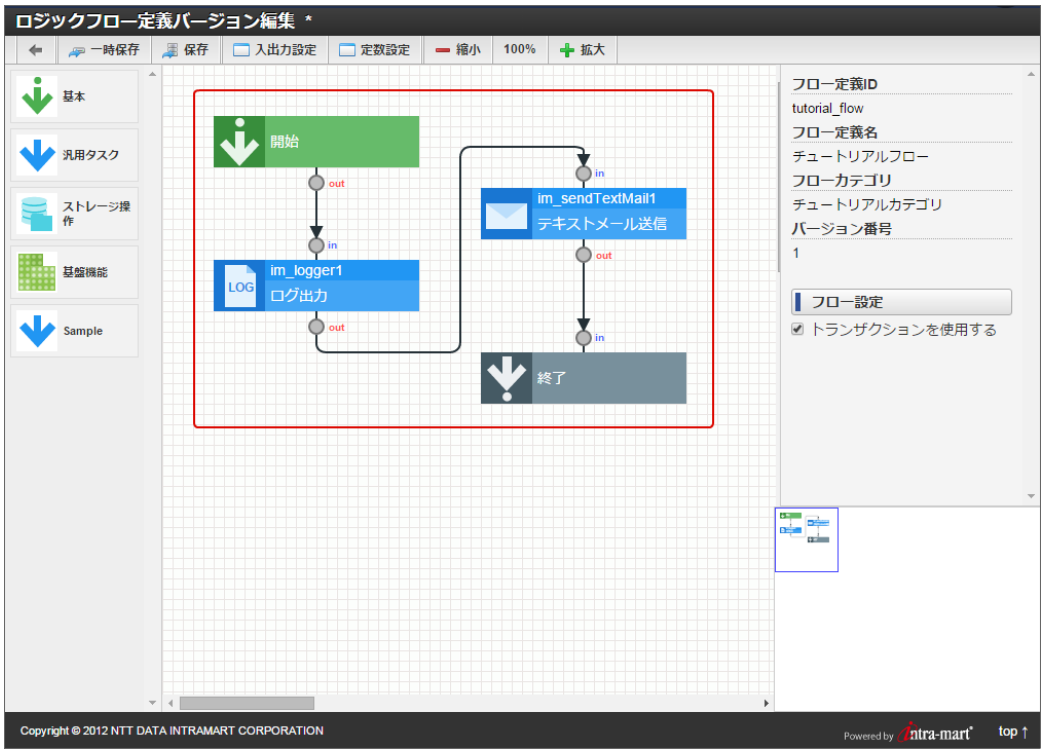

図：編集アイコンをクリック

5. ロジックフロー定義バージョン編集画面へ遷移します。
   編集対象のロジックフローのバージョンが「1」であり、フローの順序が「*新規にバージョンを作成する*」で編集したフローと異なることを確認してください。



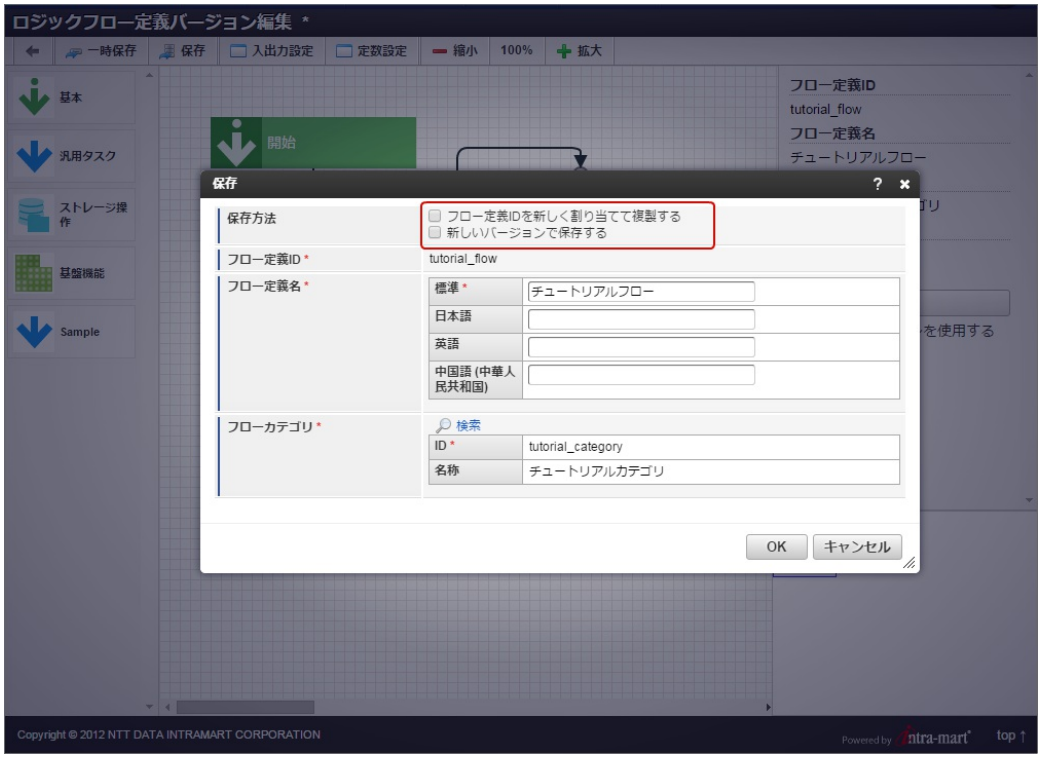

図：編集バージョンの確認

6. バージョンの編集と反映を確認するため、フローの一部を編集します。
   この例では、「テキストメール送信」タスクと「終了」制御要素の配置位置を変えています。




図：配置位置の変更

7. ロジックフロー定義編集画面上部、ヘッダ内の「保存」をクリックします。
8. 各項目を以下のように入力します。
   - 保存方法
     - フロー定義IDを新しく割り当てて複製する - 「チェックボックス：オフ」
     - 新しいバージョンで保存する - 「チェックボックス：オフ」



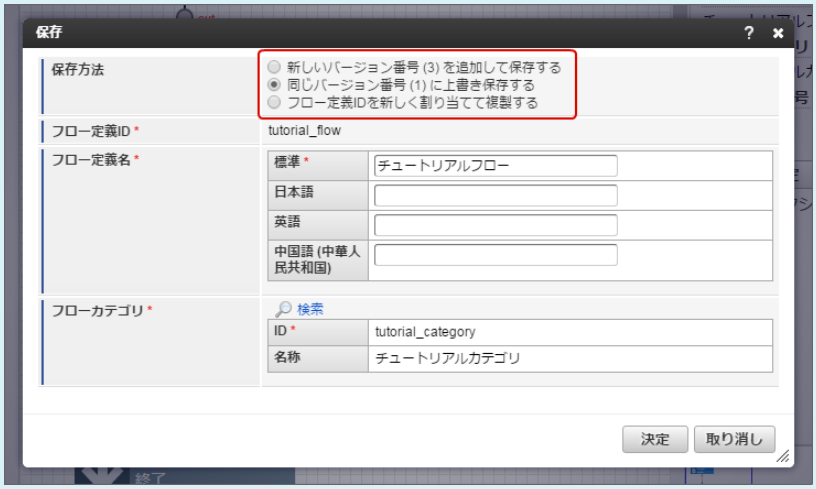

図：変更したバージョンの保存

コラム

2016 Spring(Maxima)以降での既存バージョン保存

IM-LogicDesignerでは2016 Spring(Maxima)以降、保存時の設定項目が一部変更されています。
本チュートリアルの「保存方法」と同様の保存を行うためには各項目を以下のように入力してください。

- 保存方法 - 「同じバージョン番号（1）に上書き保存する」を選択する。




図：新しい保存設定

9. 保存画面右下のOKをクリックします。
10. 保存を確認するダイアログが表示されるので、OKをクリックします。
11. 保存が完了した旨のメッセージと共に、ロジックフロー定義バージョン一覧画面に遷移します。


図：保存完了後、ロジックフロー定義バージョン一覧へ遷移

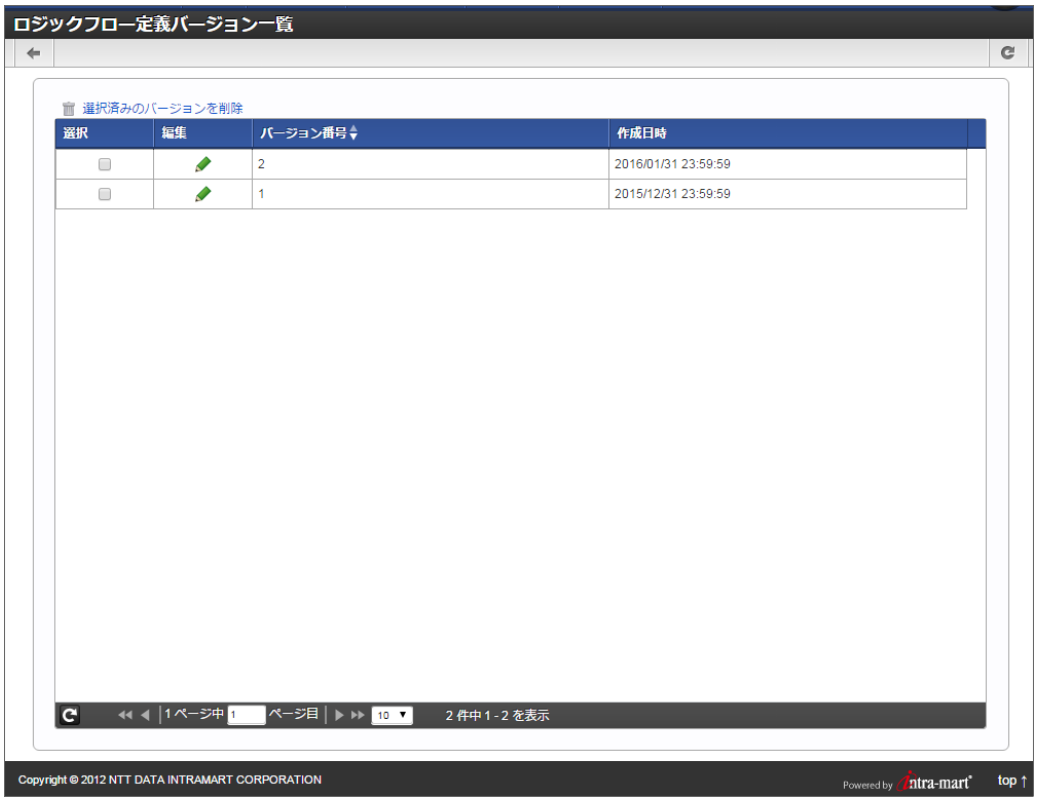
以上で、バージョンの切り替え、および、バージョンの編集が完了しました。

全ての作業が完了後、ロジックフロー定義バージョン一覧画面より、再度バージョン「1」および「2」を選択してください。
バージョン「1」が編集済みであること、および、バージョン「2」のフローに変更がないことが確認できます。

## 既存のバージョンを削除する

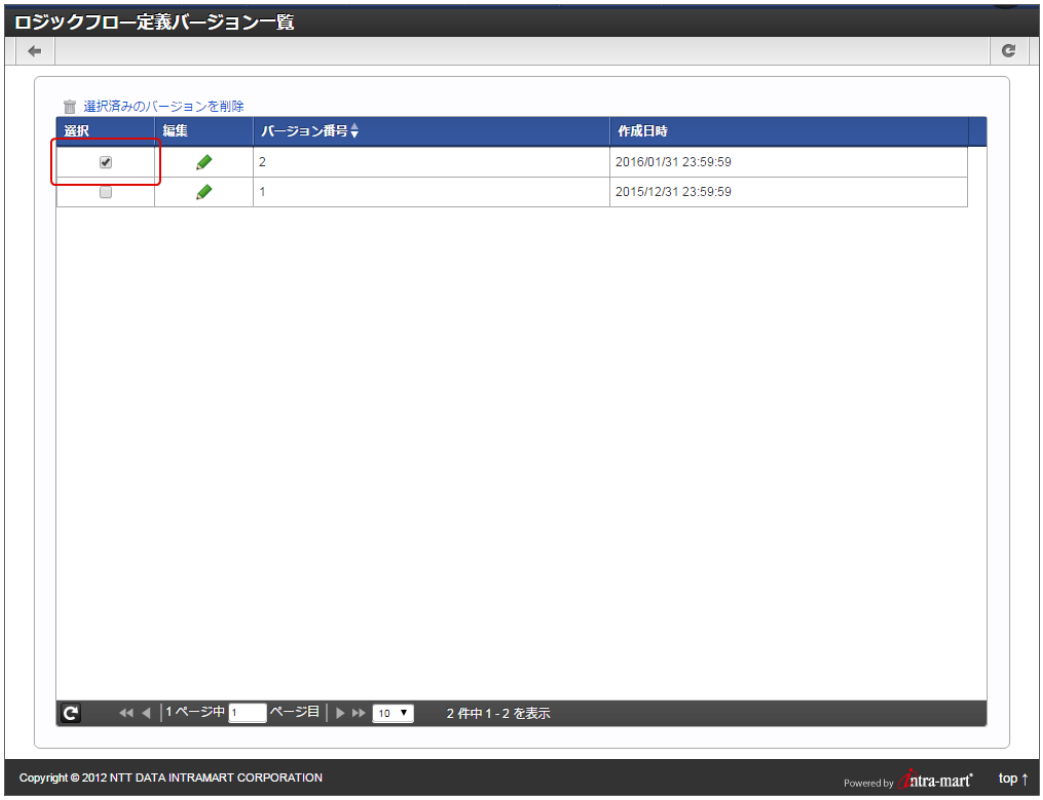
「*バージョン一覧画面を表示する*」から、作成したバージョンの削除を行います。

1. フロー定義ID「tutorial_flow」のロジックフロー定義バージョン一覧画面を表示します。



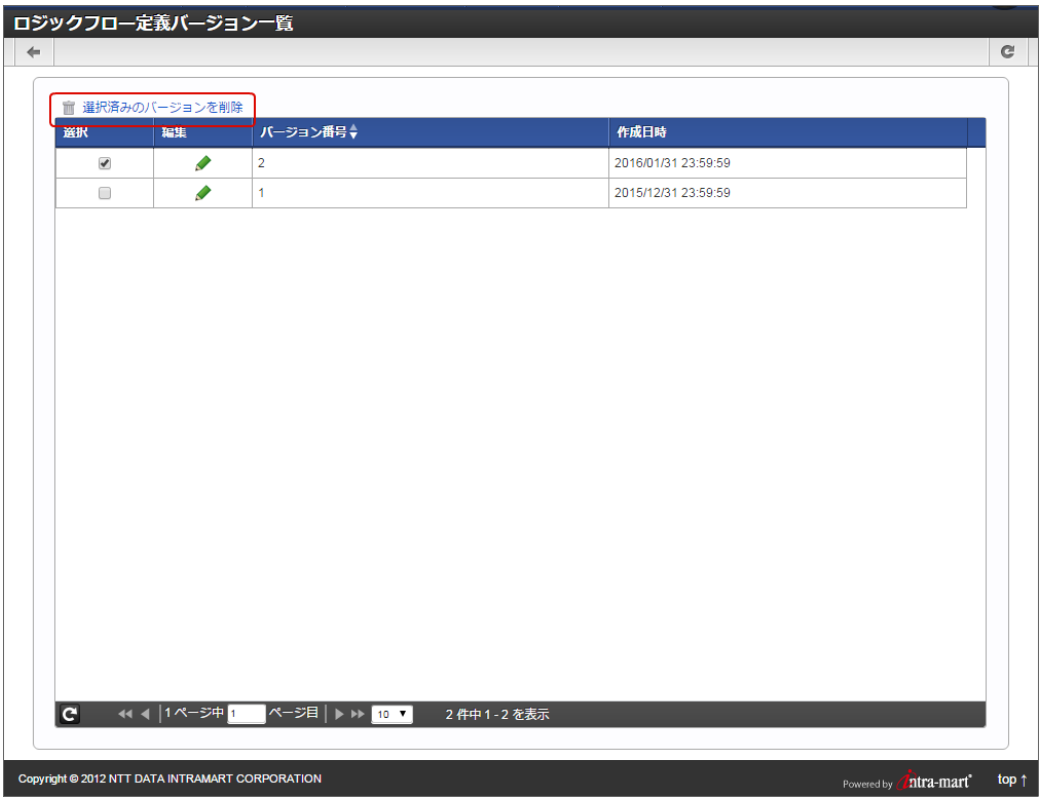

図：ロジックフロー定義バージョン一覧画面

2. バージョン番号「2」の行の選択チェックボックスにチェックを入れます




図：バージョン「2」へチェック

3. 一覧の左上の「選択済みのバージョンを削除」をクリックします。




図：選択済みのバージョンを削除をクリック

4. 削除を確認するダイアログが表示されるので、決定をクリックします。
5. 削除が完了した旨のメッセージが表示され、一覧から「2」のバージョンが無くなります。

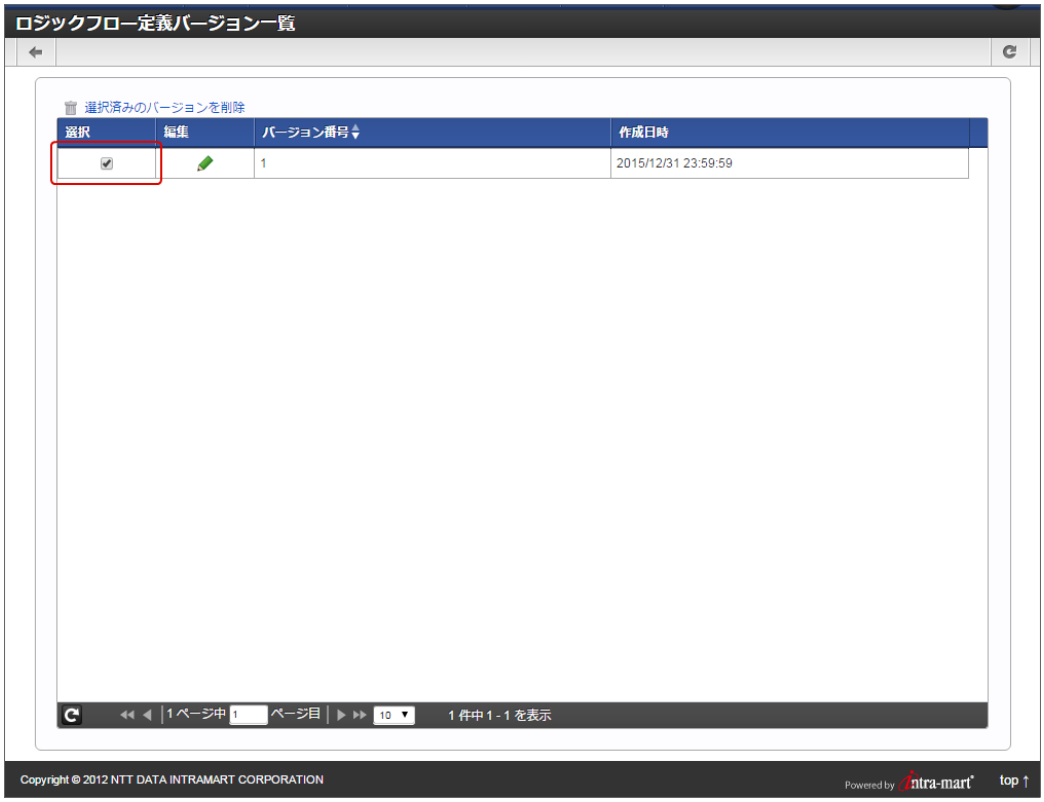

図：バージョン削除の確認

以上で、バージョンの削除が完了しました。

## 全てのバージョンを削除する（ロジックフローの削除）

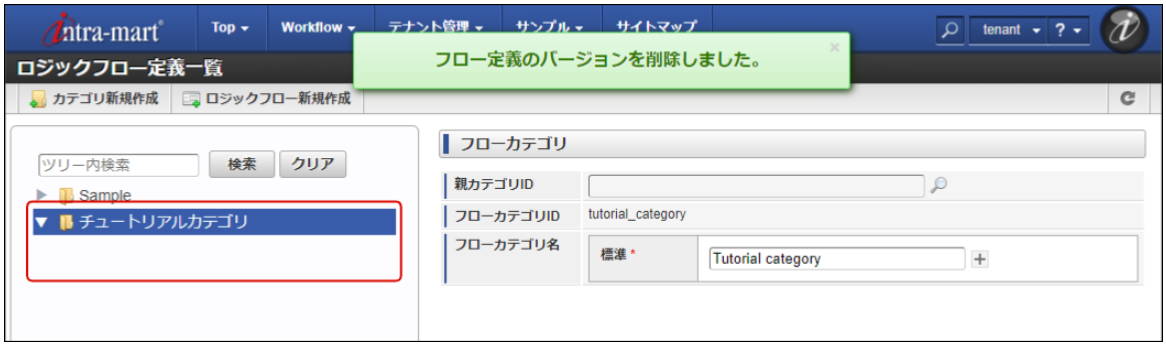
IM-LogicDesignerには、直接ロジックフローを削除するオペレーションは存在しません。
ロジックフローを削除するためには、ロジックフロー定義バージョン一覧画面より全てのバージョンを削除する必要があります。

1. フロー定義ID「tutorial_flow」のロジックフロー定義バージョン一覧画面を表示します。
2. バージョン番号「1」の行の選択チェックボックスにチェックを入れ、「選択済みのバージョンを削除」をクリックします。




図：バージョン「1」へチェック

3. 削除を確認するダイアログが表示されるので、決定をクリックします。
4. 削除が完了した旨のメッセージが表示され、ロジックフロー定義一覧画面へ遷移します。
   遷移先が「*既存のバージョンを削除する*」とは違い、ロジックフロー定義一覧画面であること、および、フロー定義ID「tutorial_flow」のロジックフローが一覧から無くなっていることを確認してください。




図：ロジックフローが削除されていることを確認

以上で、全てのバージョンの削除が完了しました。

## 複雑なフローの定義

この章では、「*基礎編 - ファースト・ステップ*」をベースに、もう一段踏み込んだフローの定義方法を説明します。
開発者は、より広い業務領域に対してIM-LogicDesignerの適用を試みることができます。

### 階層化された入力値・出力値の定義（Object型の定義）

この章では、ロジックフローの入出力値の階層的な定義の仕方について説明します。

- 階層化された値とは
- 階層化された値の定義方法
- 階層化された値のマッピング

## 階層化された値とは

階層化された値とは、値の定義に親子関係を含むものを指します。

「*基礎編 - ファースト・ステップ*」-「*入出力設定を定義する*」では、入出力ともに1つずつの非常にシンプルな構成です。
IM-LogicDesignerでは、入出力定義において上記のようなシンプルな構成の他に、JavaScriptにおけるObjectのような親子関係の存在する値の定義が可能です。

この章では、階層化された値の定義方法と、マッピングについて説明します。

### 本章で作成するフロー

本章では、階層化された値の定義方法の説明のため、「*基礎編 - ファースト・ステップ*」で作成したロジックフローを以下のように変更します。

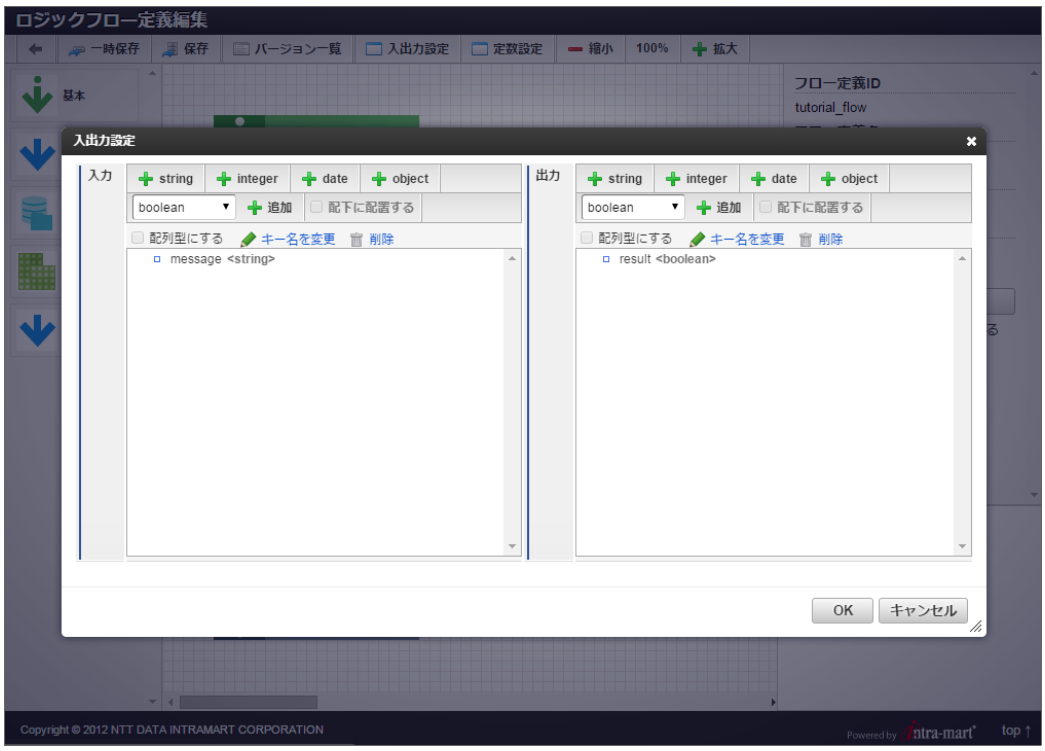
- ロジックフローの「入力値」を変更
  - 「*入出力設定を定義する*」にて定義していた入力値を変更します。変更内容は以下の通りです。
    - 「メールの題名（`subject`）」を入力として受け取るようにします。
    - 定義済みである本文（`message`)について、名称および定義位置を変更します。
- 「テキストメール送信」タスクの定数値の一部削除
  - 「*定数値を定義する*」にて定義していた「テキストメール送信」タスクの定数値のうち「メールの題名（`subject`）」を削除します。

ロジックフローの処理順の変更はありません。

## 階層化された値の定義方法

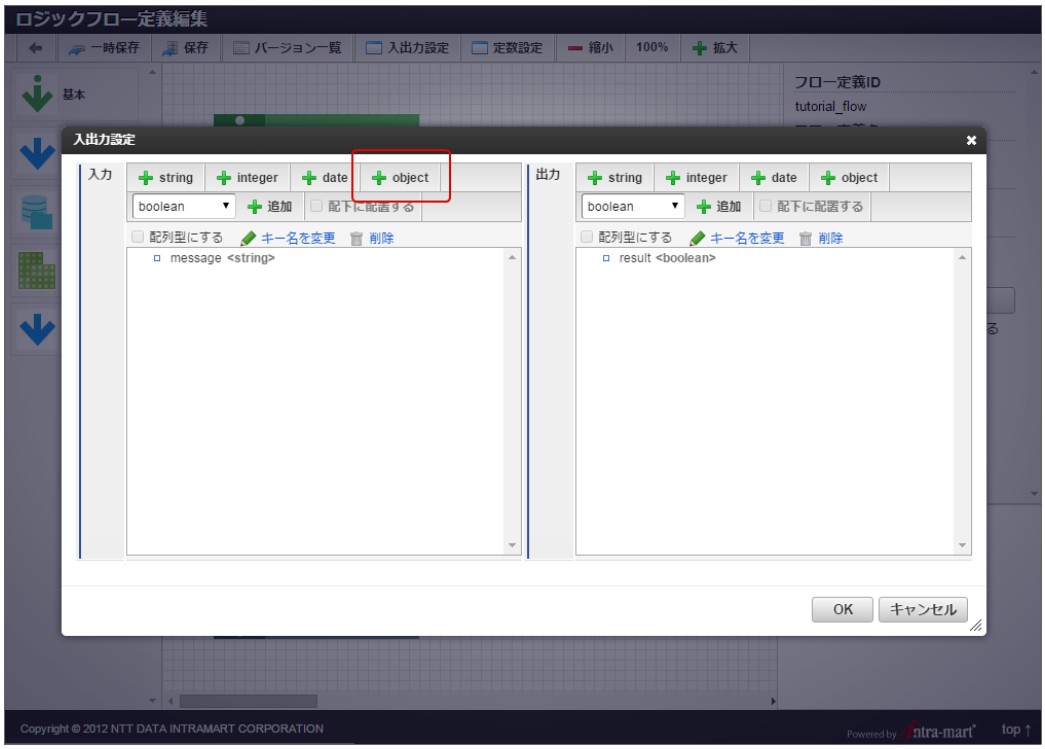
ロジックフローの入力値を本チュートリアルで作成するフローに合わせて、再定義します。

1. 「サイトマップ」→「LogicDesigner」→「フロー定義一覧」から、ロジックフロー定義一覧を開きます。
   左ペインのツリーから「チュートリアルカテゴリ」の開閉アイコンをクリックしカテゴリ配下を情報を表示します。
   フロー定義「チュートリアルフロー」を選択し編集ボタンをクリックし、ロジックフロー定義編集画面上部、ヘッダ内の「入出力設定」をクリックします。



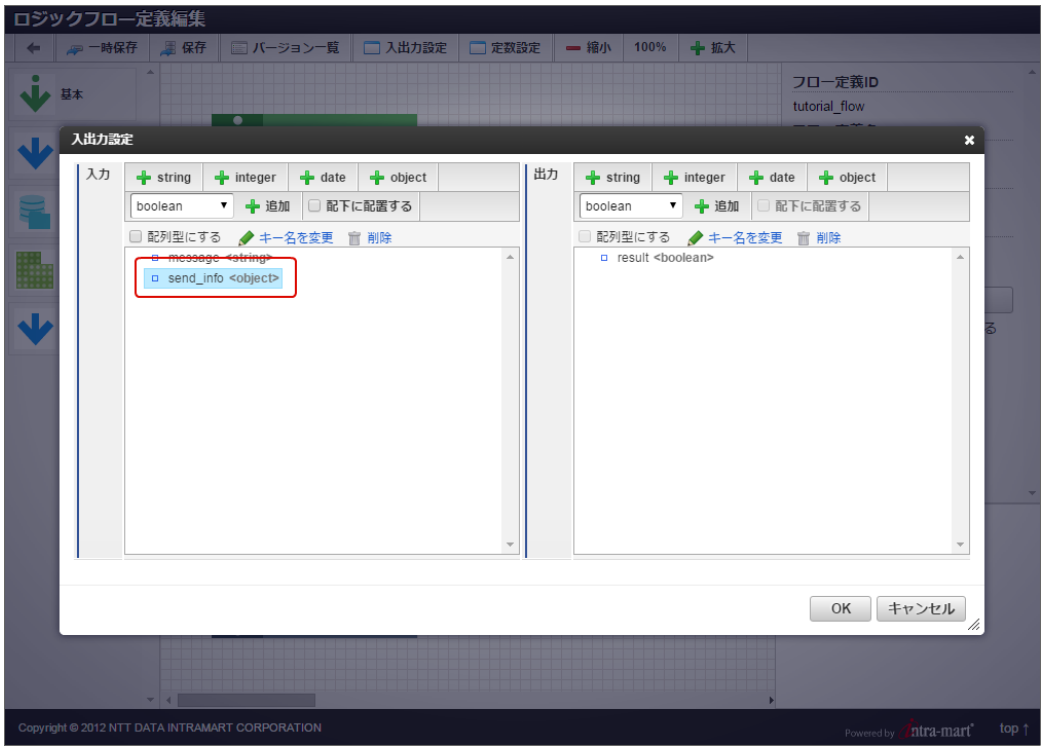

図：入出力設定をクリック

2. 入出力設定画面の、入力ペイン上部にある「`+object`」をクリックします。



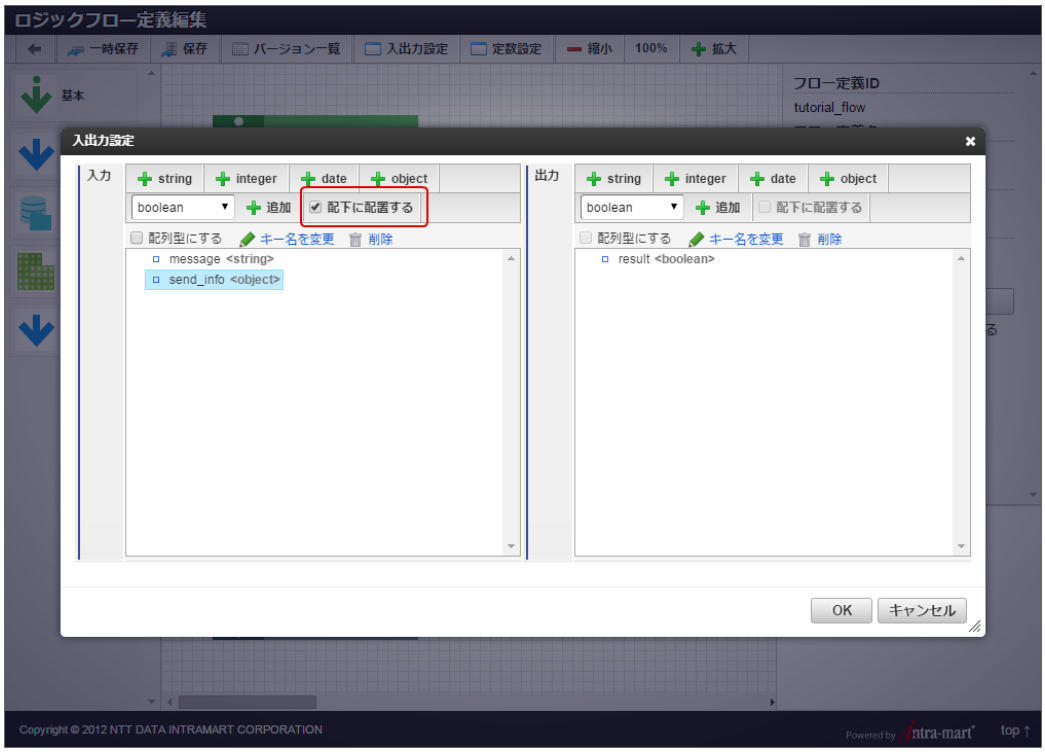

図：object要素の追加

3. 入力ペイン下部に、新しい値が設定されるので、名称を「send_info」とします。



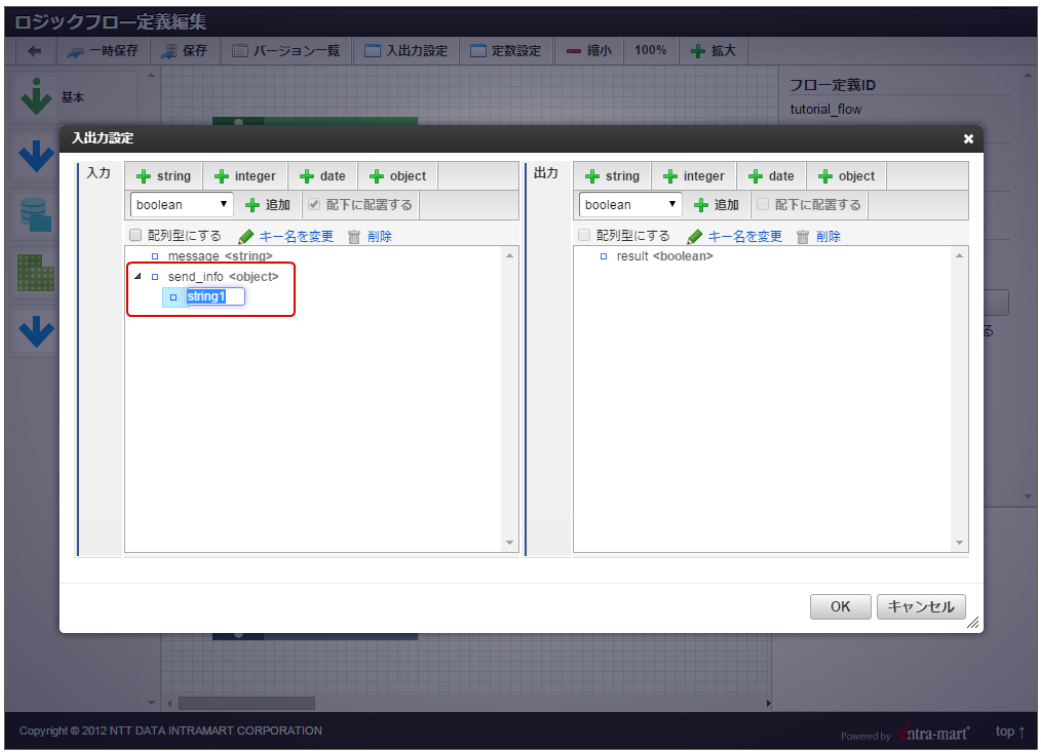

図：object要素の名称変更

4. 「send_info」を選択した状態で、入力ペイン上部にある「配下に配置する」にチェックをいれます。



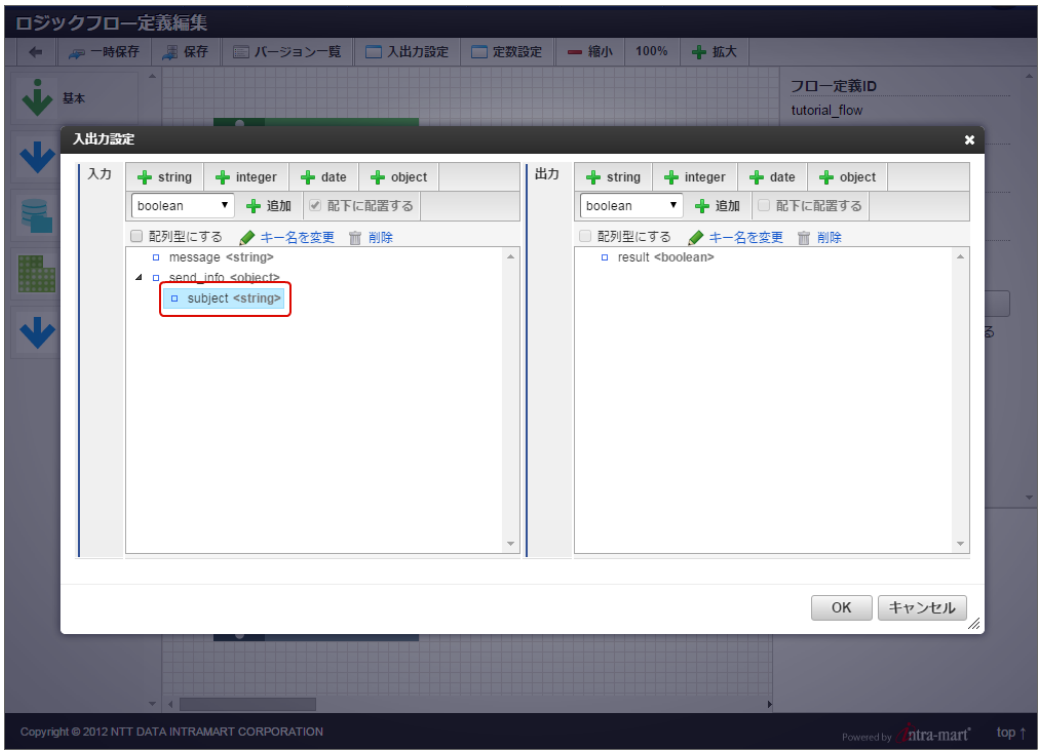

図：「配下に配置する」をチェック

5. 入力ペイン上部にある「+string」をクリックします。
   新しい値が「send_info」の配下に配置されることが確認できます。



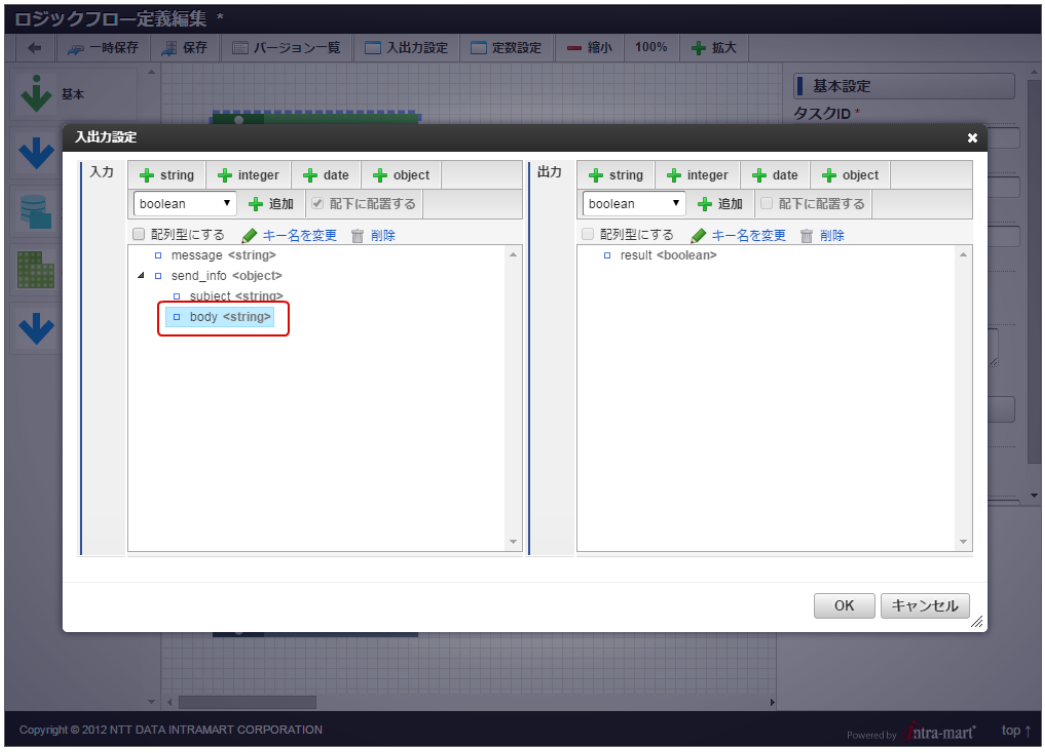

図：string要素を配下に追加

6. 配置された値の名称を「subject」とします。



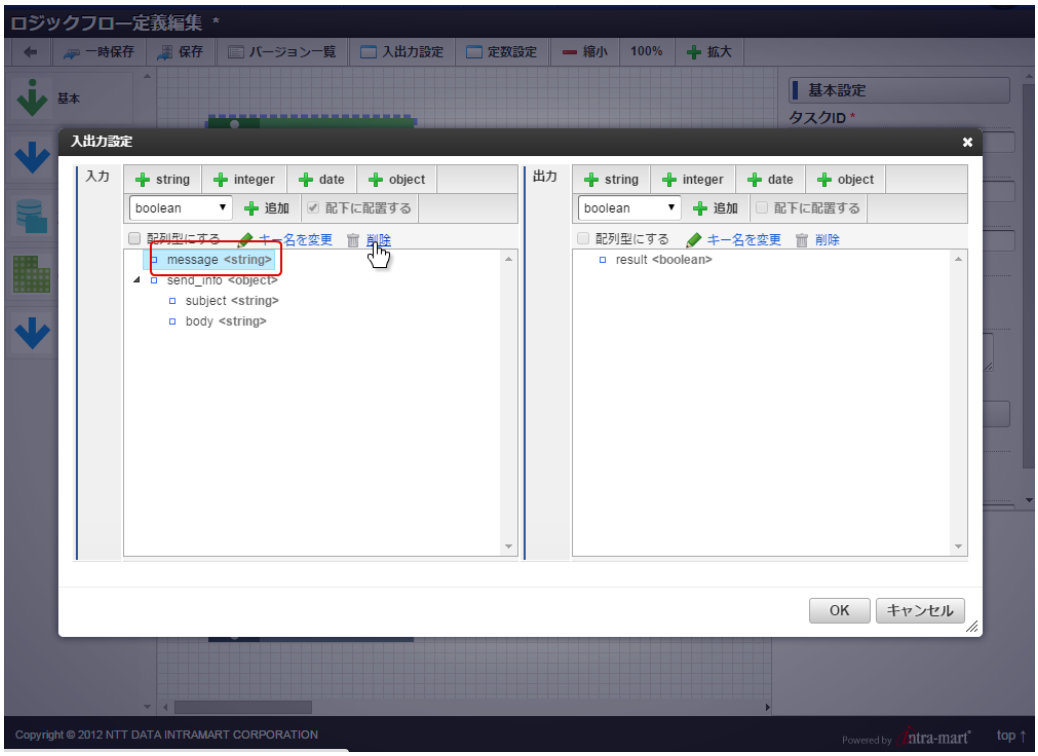

図：string要素の名称変更

7. 同様の手順で「+string」をクリックし、配下に配置された値を「body」とします。




図：新しいstring要素を配下に追加

8. 定義済みの本文（message）を削除します。今後本文情報は「body」で管理するものとします。




図：既存のstring要素の削除

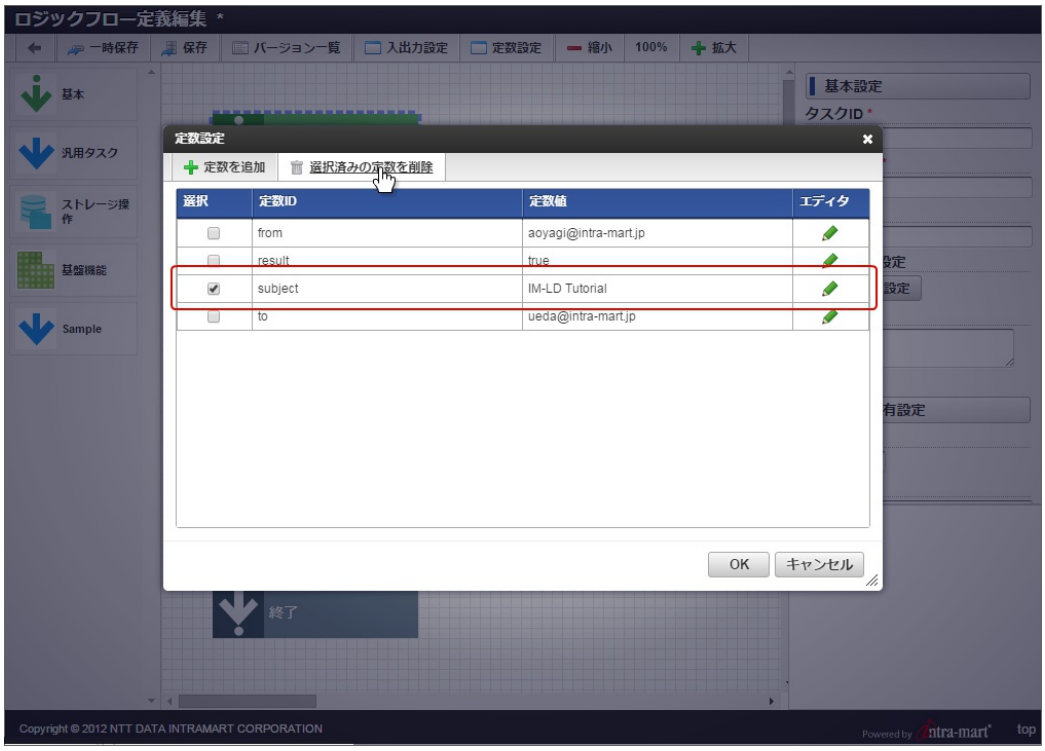
最後に、右下のOKをクリックし、設定を完了します。

以上で、階層化された入力値の設定が完了しました。

## 階層化された値のマッピング

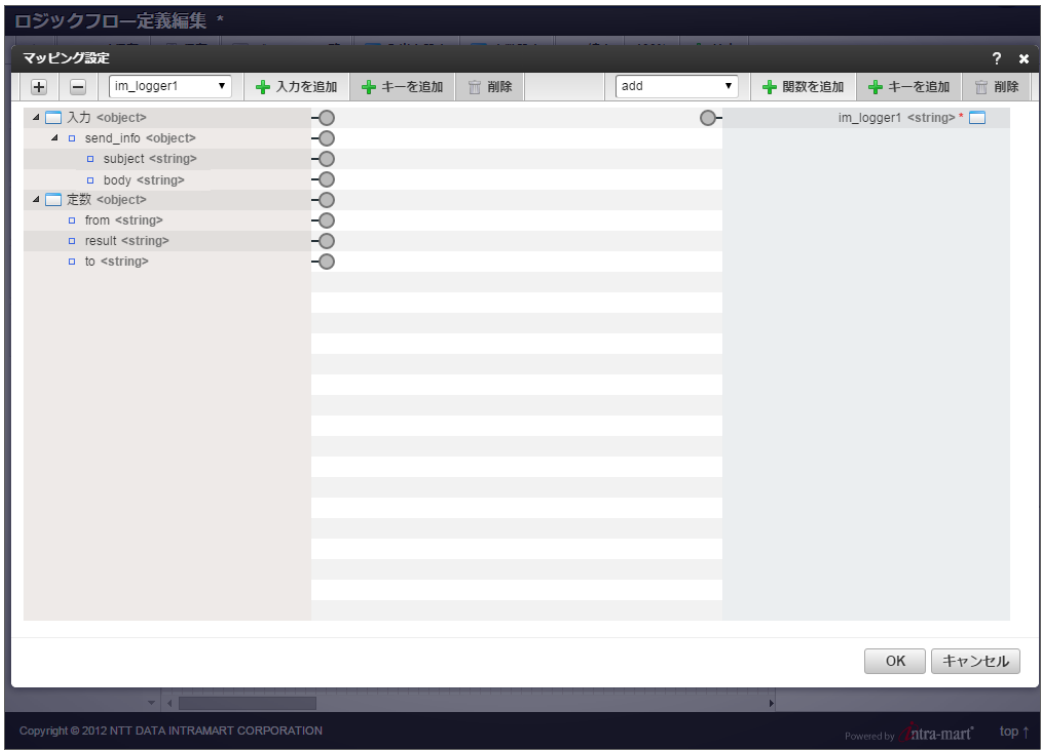
再定義した入力値を「テキストメール送信」タスクへマッピングします。

1. マッピング設定の事前準備として、定数設定画面からメールの題名（subject）を削除します。




図：定数値の削除

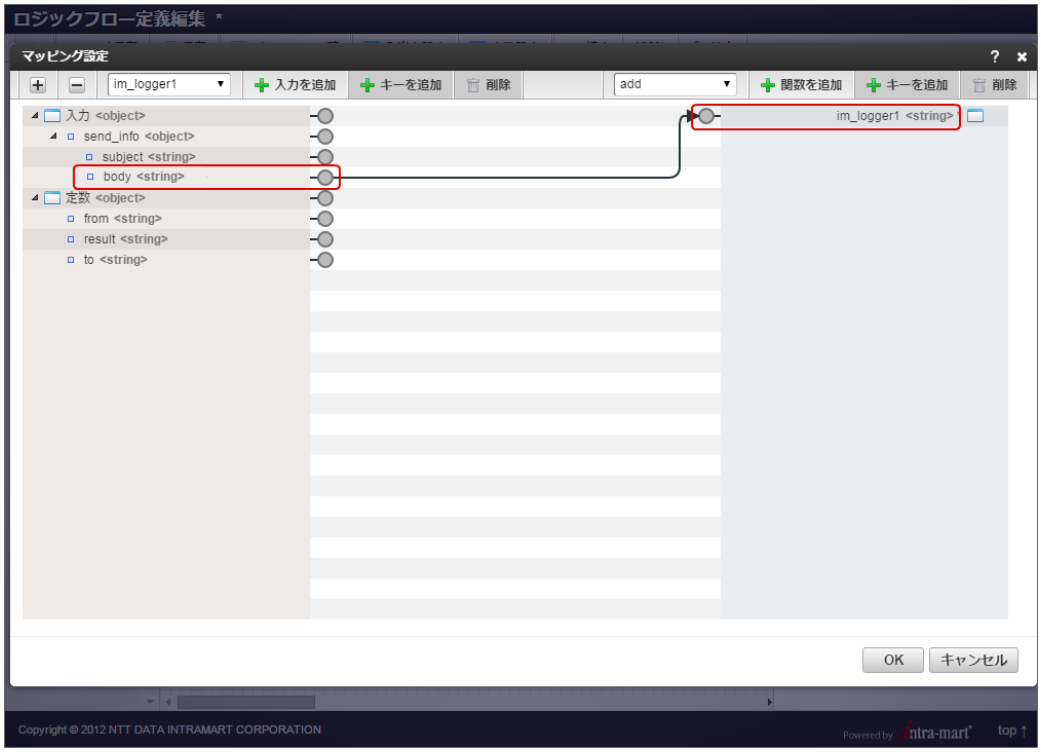
2. ロジックフロー定義編集画面上の「ログ出力」タスクをクリックし、プロパティからマッピング設定画面を開きます。
（事前に定義されたマッピング内容は、入力値の変更によって自動で消えます）




図：「ログ出力」タスクのマッピング設定画面

3. マッピングを以下のとおりに設定します。

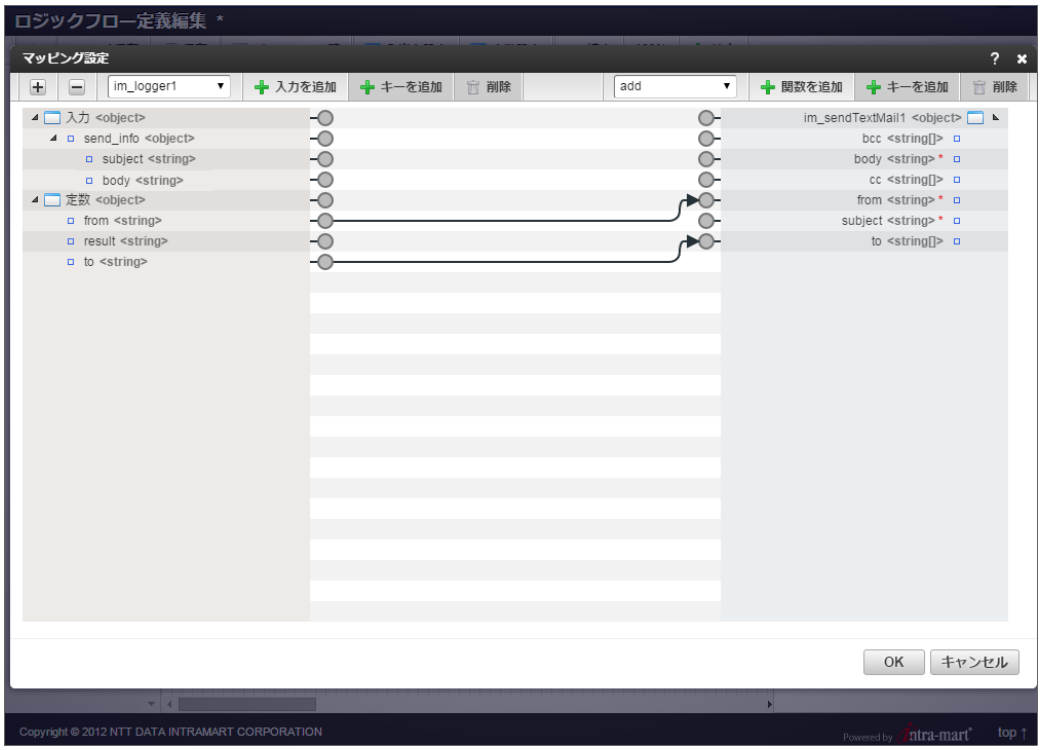
| 入力（始点） | 出力（終点） |
| --- | --- |
| 入力<object> - send_info<object> - body<string> | im_logger1<string> |




図：「ログ出力」タスクのマッピング設定

4. 設定画面右下のOKをクリックし、「ログ出力」タスクのマッピング設定を終了します。

5. ロジックフロー定義編集画面上の「テキストメール送信」タスクをクリックし、プロパティからマッピング設定画面を開きます。
（事前に定義された一部のマッピング内容は、入力値の変更によって自動で消えます）

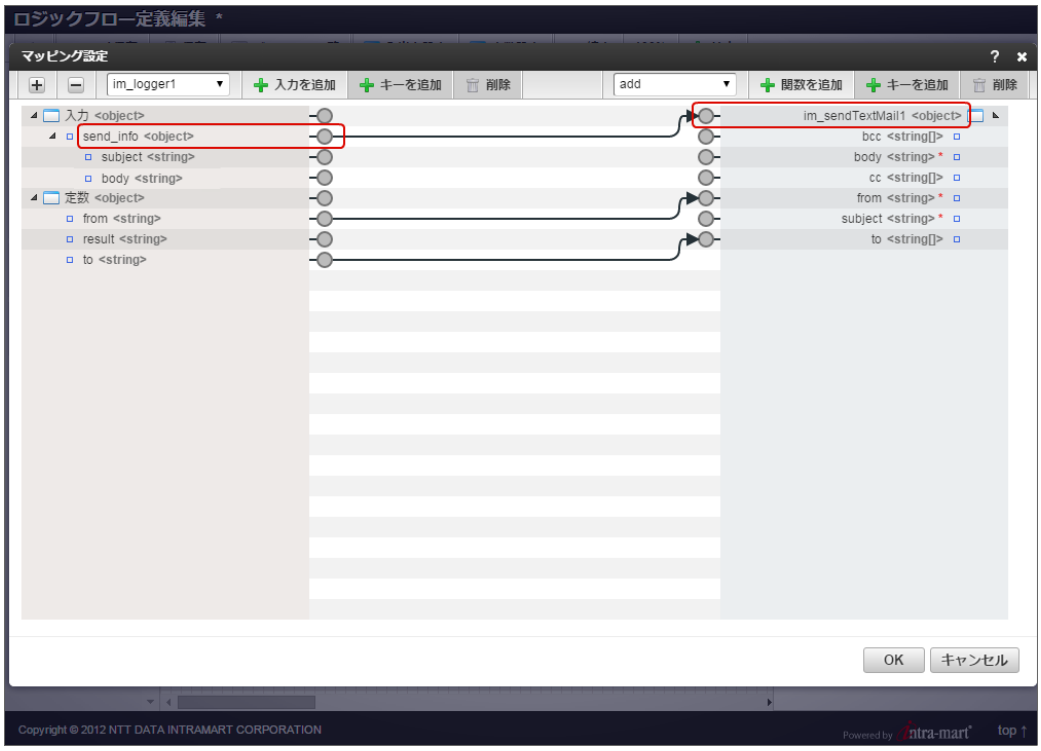



図：「テキストメール送信」タスクのマッピング設定画面

6. マッピングを以下のとおりに設定します。

| 入力（始点） | 出力（終点） |
| --- | --- |
| 入力<object> - send_info<object> | im_sendTextMail1<object> |

> **注意**
>
> マッピングの仕様について
>
> <object>型のマッピングとプリミティブ型のマッピングが混在した場合、マッピングを行った（線を引いた）順に処理が行われるため、一見、同一のマッピングに見えても動作が違うといったパターンが存在します。
>
> チュートリアルの流れ上、「from」と「to」に最初からマッピングがされている状態ですが、マッピングを一度外し、
> 「send_info」、「from」、「to」の順にマッピングの設定を行ってください。
> 「send_info」のマッピングが先に行われていない場合、処理が正常に行われず「from」と「to」の値がnullに設定されます。




図：「テキストメール送信」タスクのマッピング設定

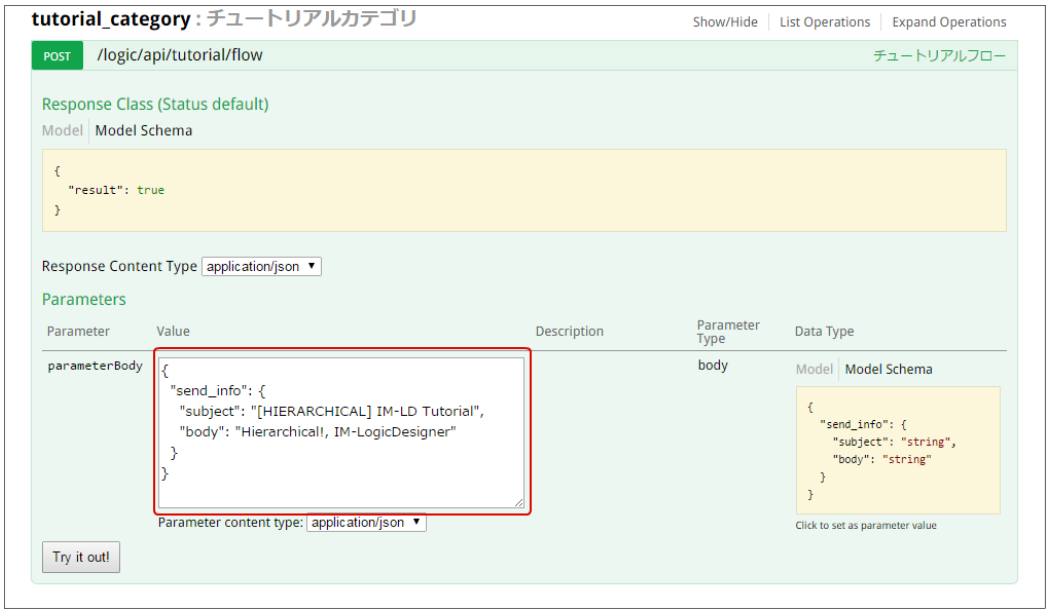
7. 設定画面右下のOKをクリックし、「テキストメール送信」タスクのマッピング設定を終了します。
8. ロジックフローを保存します。

階層化された値のマッピングの結果の確認

保存が完了した後、作成したロジックフローを実行し、結果を確認します。
実行方法は「*Swagger(SPEC)から実行する*」を、結果の確認方法は「*結果を確認する*」を参照してください。

今回、新しくフローの入力値を定義したため、実行時に指定するパラメータが追加されます（subject）。
追加されたパラメータを含め、以下の内容でロジックフローを実行します。

- subject - [HIERARCHICAL] IM-LD Tutorial
- body - Hierarchical!, IM-LogicDesigner




図：Swagger上での設定例

実行結果として「*基礎編 - ファースト・ステップ*」で作成したロジックフローで得られた結果と、同じ出力が得られることを確認してください。

この結果は、階層化された値同士をマッピングし、マッピング元とマッピング先で配下に同名の値を持っている場合、IM-LogicDesignerは自動でその値をマッピングすることを示しています。
今回、マッピング元のsend_info配下に存在するsubject、および、bodyが、自動的に「テキストメール送信」タスクのsubject、および、bodyに

紐付けられました。

以上で、ロジックフローの入出力値の階層的な定義が完了しました。

## 条件分岐を利用したフロー

この章では、ロジックフローの定義に条件分岐を利用する方法を説明します。



### 「分岐」制御要素

IM-LogicDesignerでは、フロー内で定義した条件による単純分岐処理を行う「分岐」制御要素を提供しています。
開発者は「分岐」制御要素を利用することで、指定した条件による分岐処理を実現することができます。

> **コラム**
>
> 分岐できるパターン数
>
> IM-LogicDesignerが提供する「分岐」制御要素では、分岐先は2つまで設定できます。
> 3つ以上の分岐については、「分岐」制御要素を複数組み合わせることで実現可能です。

本章で作成するフロー

本章では、「分岐」制御要素によって指定した条件の分岐処理が行われることを確認するため、「*基礎編 - ファースト・ステップ*」で作成したロジックフローを以下のように変更します。

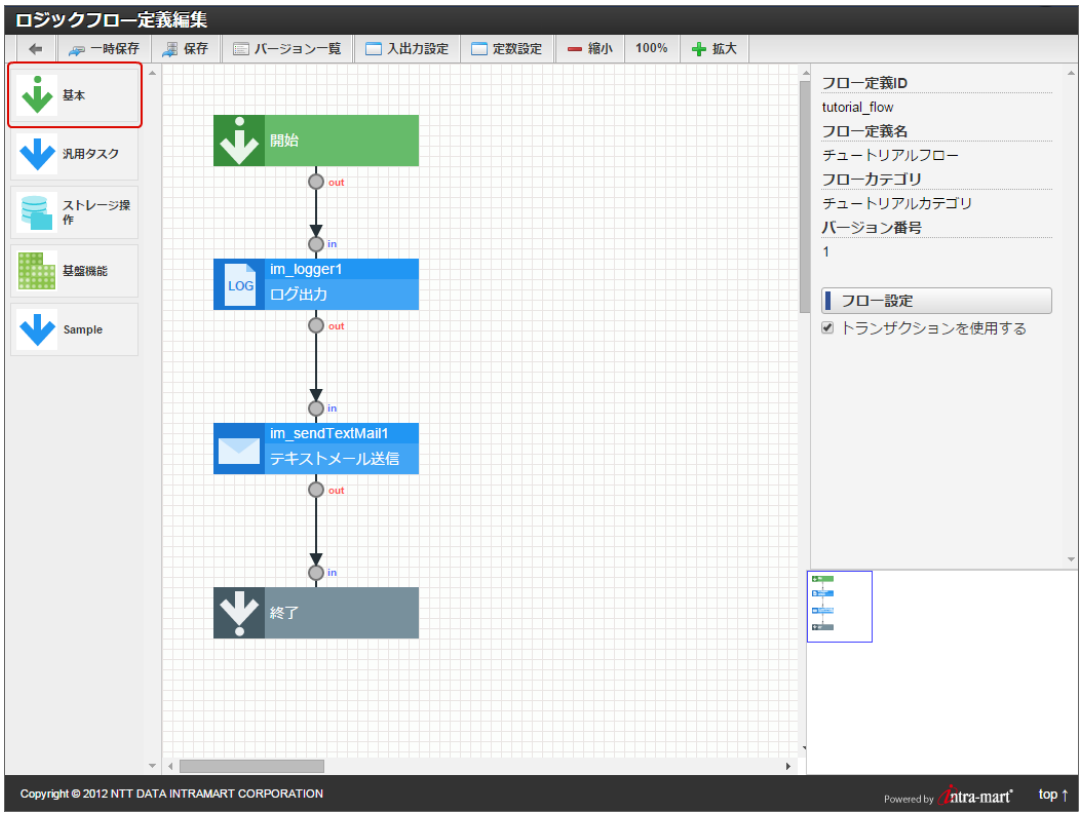
- 「ログ出力」タスクと「テキストメール送信」タスクの間に、「分岐」制御要素を組み込みます。
- 分岐条件としてロジックフローの入力値に分岐を決定するフラグを受け取ります。
- 「分岐」制御要素では、入力値のフラグを条件に分岐処理を行います。
  - フラグが`true`の場合、「テキストメール送信」タスクを実行する。
  - フラグが`false`の場合、「テキストメール送信」タスクの実行をスキップする。

### 条件分岐を利用したフローの作成

条件分岐を利用したフローを実現する最初のステップとして、ロジックフローに条件分岐のエレメントを配置します。

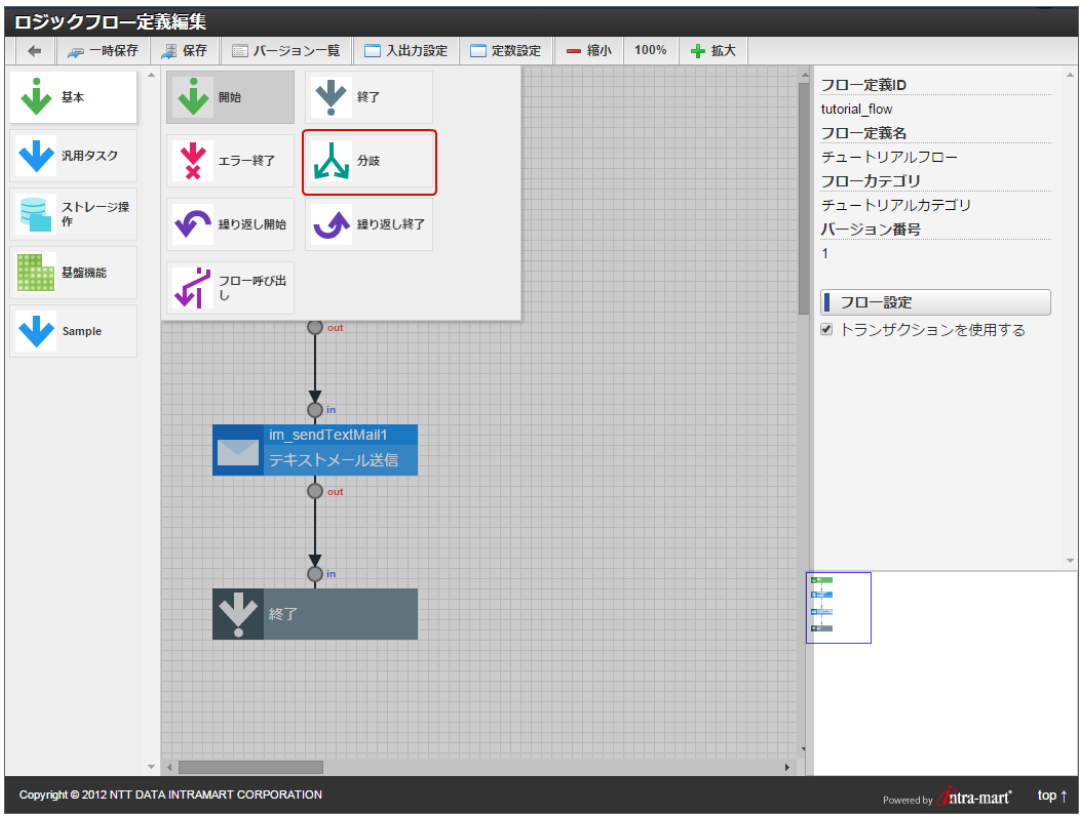
1. 「サイトマップ」→「LogicDesigner」→「フロー定義一覧」から、ロジックフロー定義一覧を開きます。
   左ペインのツリーから「チュートリアルカテゴリ」の開閉アイコンをクリックしカテゴリ配下を情報を表示します。
   フロー定義「チュートリアルフロー」を選択し編集ボタンをクリックし、ロジックフロー定義編集画面を開きます。

2. ロジックフロー定義編集画面左部、パレット内の「基本」へカーソルを合わせます。



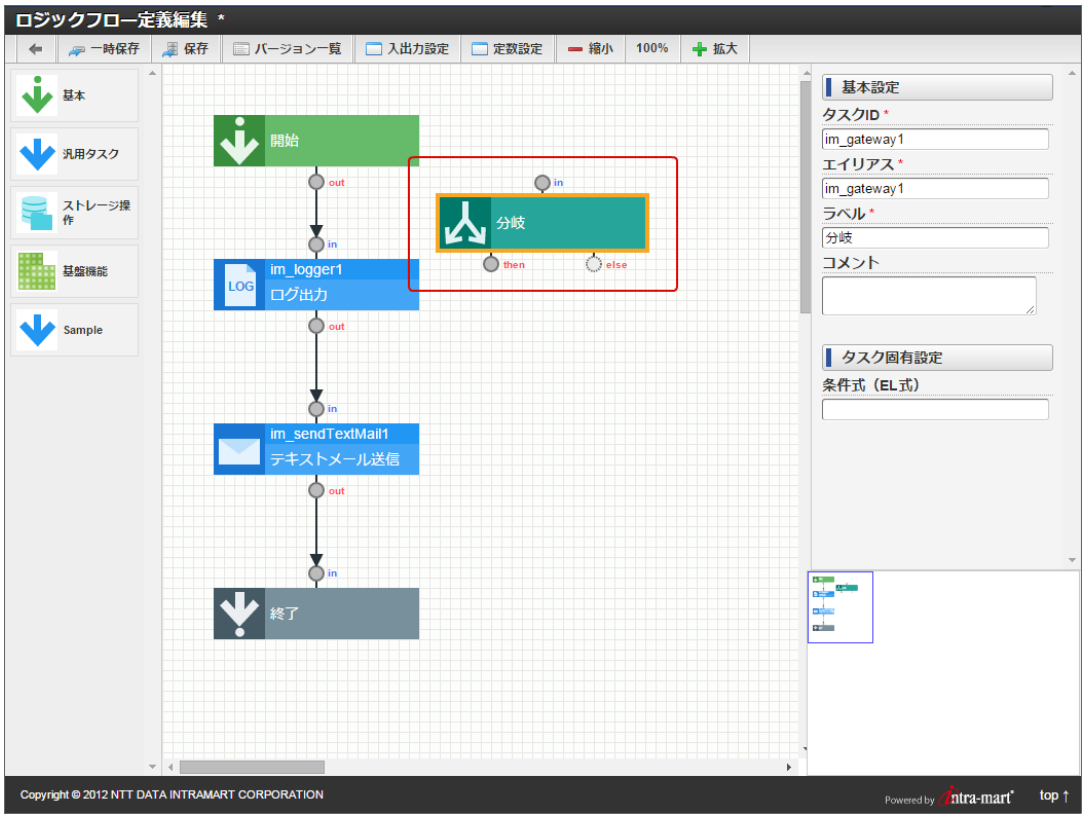

図：パレット内の「基本」

3. 「基本」にカテゴライズされる制御要素一覧がスライドインします。
   スライドインした一覧から、「分岐」をクリックします。




図：「分岐」制御要素をクリック

4. 分岐処理を行う制御要素がフロー編集画面上に追加されます。

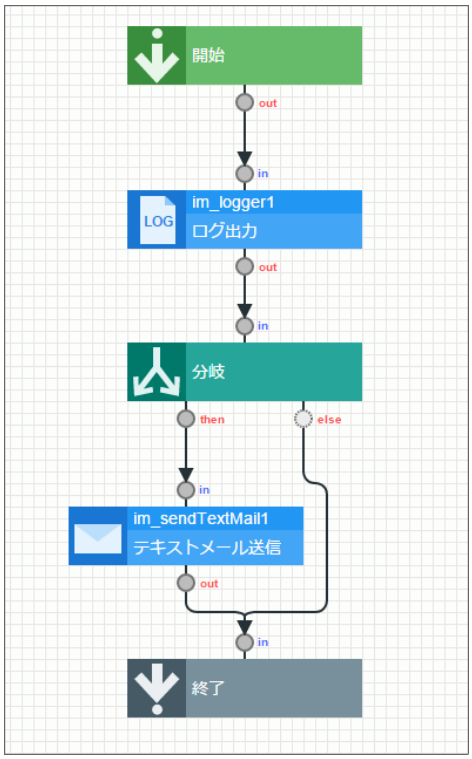



図：「分岐」制御要素の追加

追加された「分岐」制御要素を、既存のフローに組み込みます。
具体的な線（シーケンス）の定義内容は以下の通りです。

| 入力（始点） | 変更前の出力（終点） | 変更後の出力（終点） |
|---|---|---|
| ログ出力 - out | テキストメール送信 - in | 分岐 - in |
| 分岐 - then | （なし） | テキストメール送信 - in |
| 分岐 - else | （なし） | 終了 - in |

ロジックフローの完成例は以下の通りです。

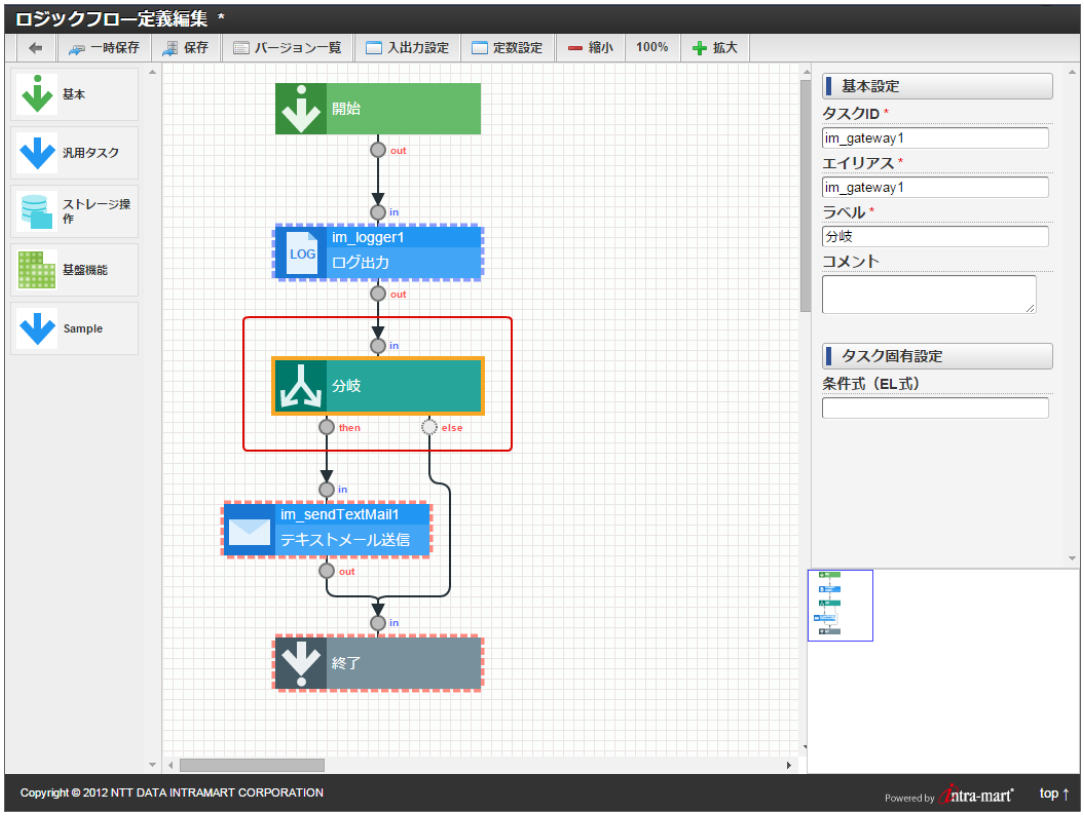



図：ロジックフロー（分岐）完成例

## 条件式の定義

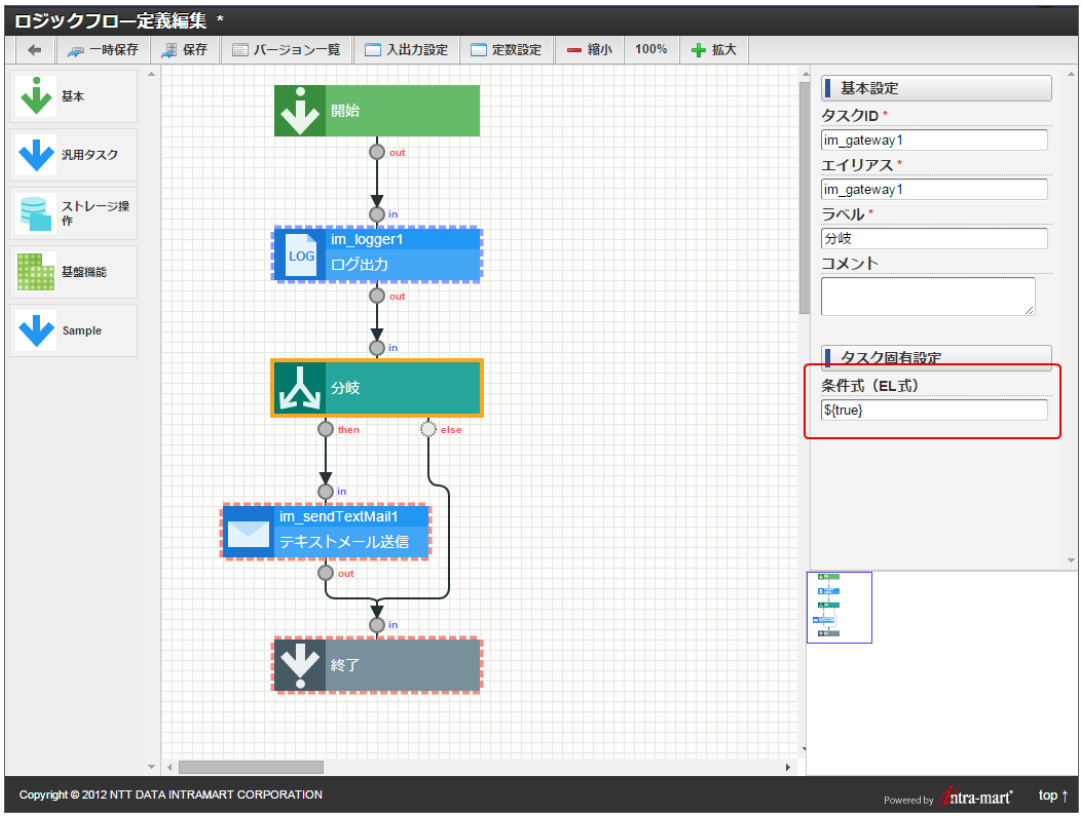
次に、実際に条件を定義し、その条件のもとロジックフローが動作することを確認します。
ロジックフローにおける条件分岐の条件設定は、設定を行う「分岐」制御要素のプロパティから行います。

1. フロー編集画面上の「分岐」制御要素をクリックします。




図：「分岐」制御要素を選択

2. タスク固有設定の項目を以下のとおりに変更します。
   - 条件式（EL式）-「`${true}`」

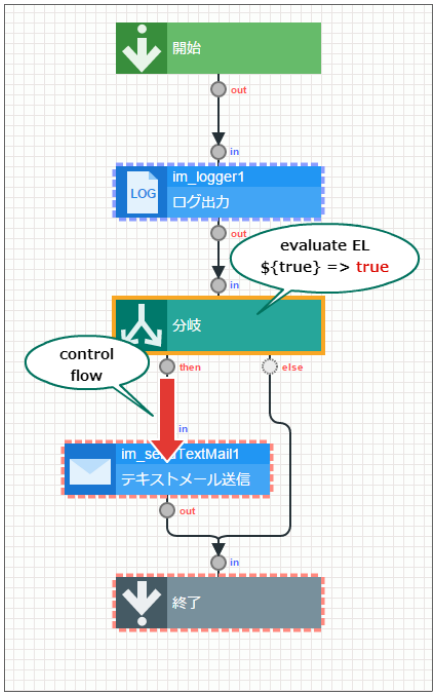



図：条件式の定義

3. ロジックフローを保存します。

保存が完了した後、作成したロジックフローを実行し、結果を確認します。
実行方法は「*Swagger(SPEC)から実行する*」を、結果の確認方法は「*結果を確認する*」を参照してください。
そして、実行結果として「*基礎編 - ファースト・ステップ*」で作成したロジックフローで得られた結果と、同じ出力が得られることを確認してください。

これは、「分岐」制御要素が定義した条件式を評価した結果として、常にthen端子が接続された方へ処理を制御したためです。




図：「分岐」制御要素が条件式を元に処理を制御するイメージ

> **コラム**
>
> EL式について
>
> EL式の概要や記述方法の詳細は「IM-LogicDesigner仕様書」-「付録」-「EL式」を参照してください。

「分岐」制御要素の動作を更に確認するためには、タスク固有設定の項目を以下のとおりに変更し、再度ロジックフローを実行してください。

- 条件式（EL式）-「${false}」

実行結果として「ログ出力」タスクの出力結果のみが得られ、「テキストメール送信」タスクの出力結果が得られないことを確認できます。

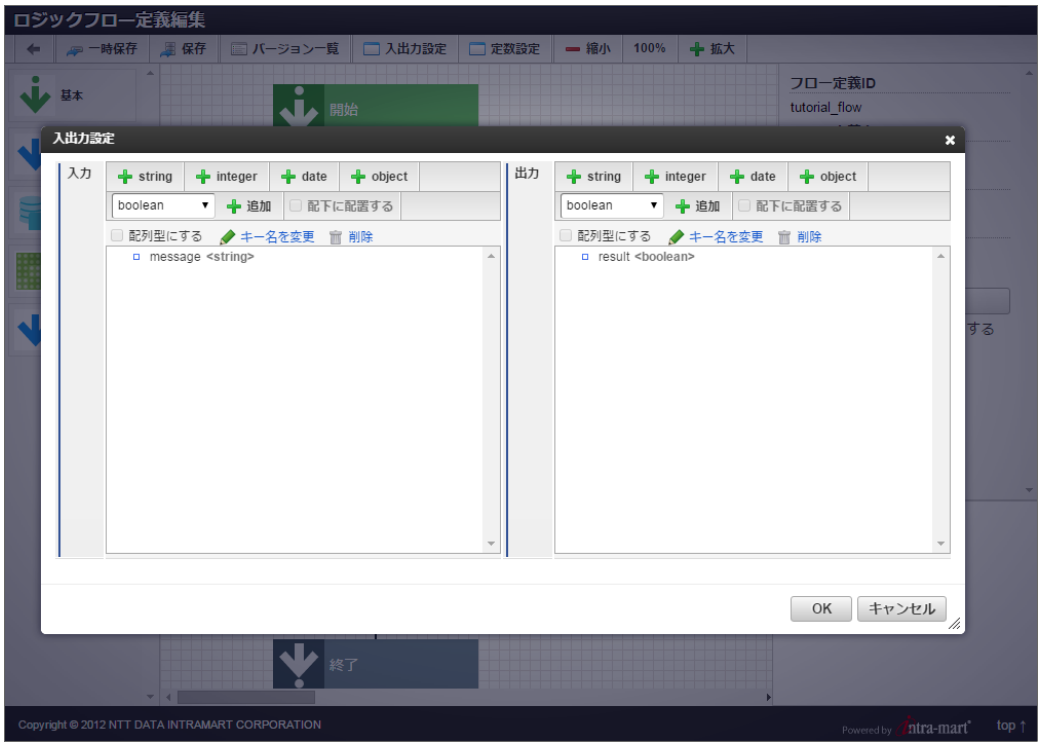
以上で、分岐に設定された条件をもとにロジックフローが動作することが確認できました。

## より高度な条件式の定義

「*条件式の定義*」で定義した条件式では常に同じ結果を返すため、条件分岐としては不十分です。
最後に、ロジックフローの入力値として分岐を決定するフラグ（boolean）を定義し、その値をもとに処理を行う方法を説明します。

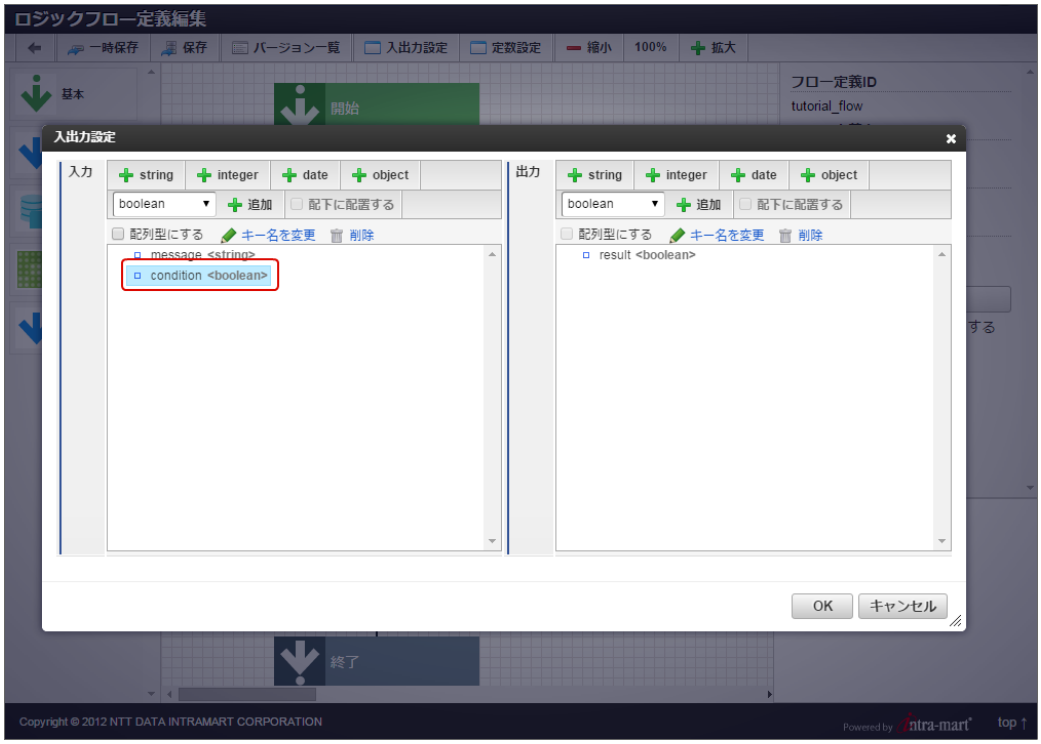
1. 「サイトマップ」→「LogicDesigner」→「フロー定義一覧」から、ロジックフロー定義一覧を開きます。
   一覧の中から、フロー定義ID「tutorial_flow」の行の編集アイコンをクリックし、ロジックフロー定義編集画面を開きます。

2. ロジックフロー定義編集画面上部、ヘッダ内の「入出力設定」をクリックし、入出力設定画面を表示します。



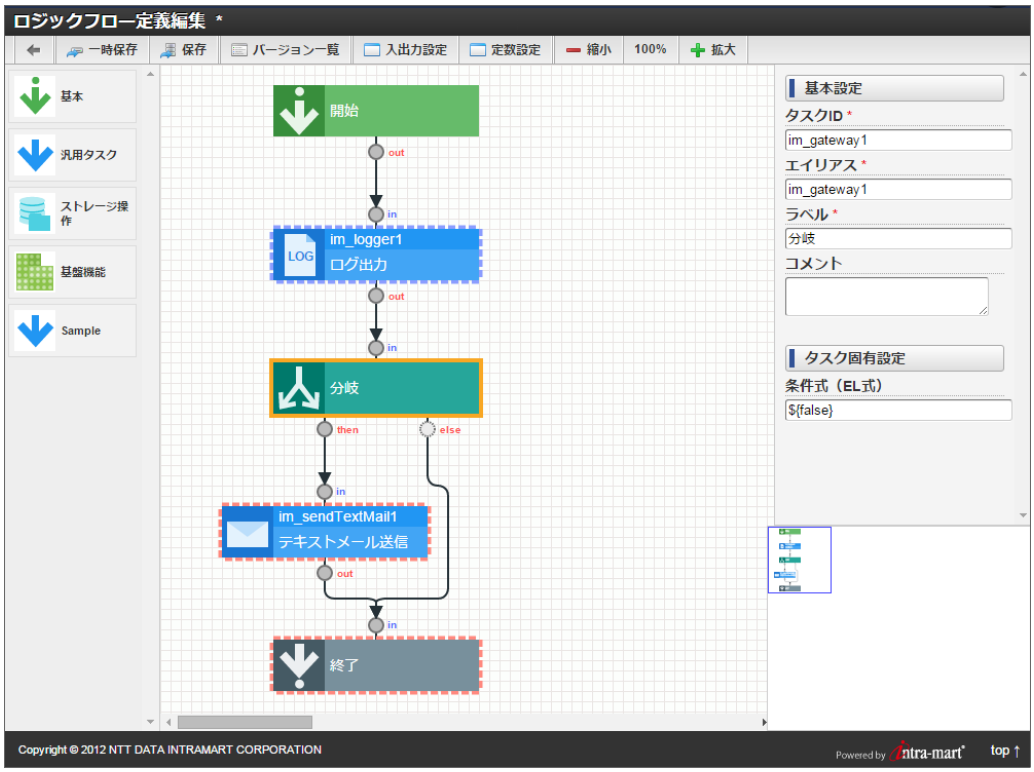

図：入出力設定画面の表示

3. 入力値として分岐を決定するフラグを以下のとおりに設定します。
   - パラメータ名「condition」
   - 型「boolean」



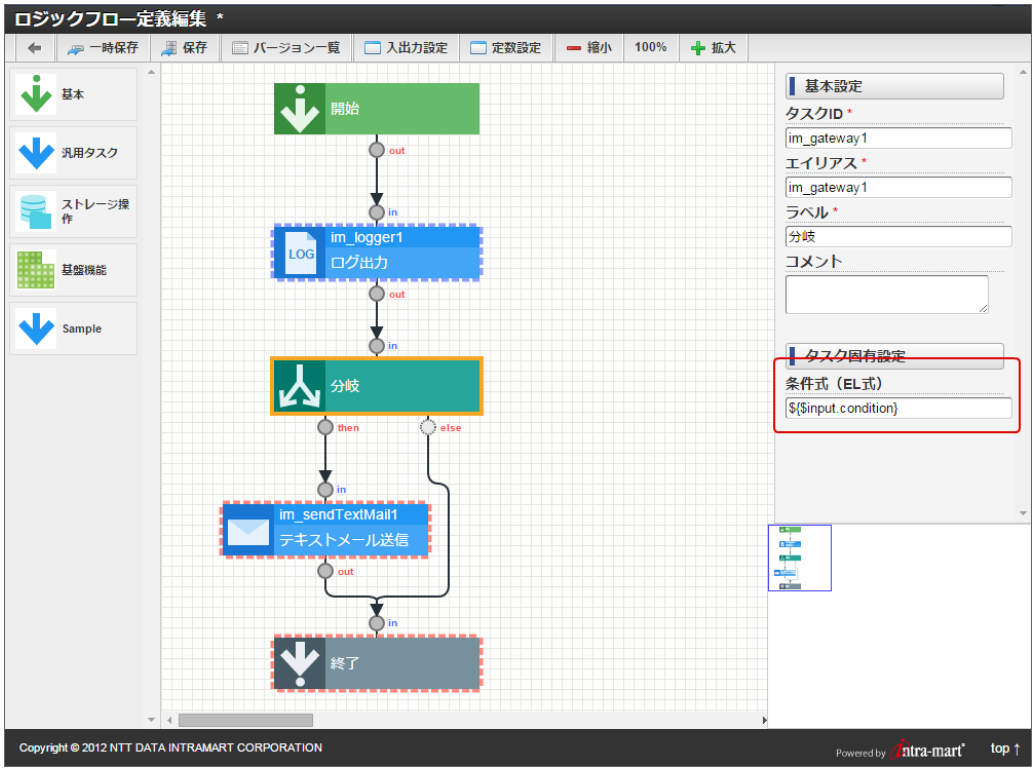

図：新規パラメータの追加（boolean）

4. 入力設定後、フロー編集画面上の「分岐」制御要素をクリックします。




図：「分岐」制御要素をクリック

5. タスク固有設定の項目を以下のとおりに変更します。
   - 条件式（EL式）-「`${$input.condition}`」




図：条件式の設定

6. ロジックフローを保存します。

保存が完了した後、作成したロジックフローを実行し、結果を確認します。
はじめに、今回新しく定義した「分岐を決定するフラグ」の値に`true`を設定し、ロジックフローを実行してください。


図：Swagger上での設定例

実行結果として「*基礎編 - ファースト・ステップ*」で作成したロジックフローで得られた結果と、同じ出力が得られることを確認してください。

続けて、「分岐を決定するフラグ」の値にfalseを設定し、ロジックフローを実行してください。
実行結果として「ログ出力」タスクの出力結果のみが得られ、「テキストメール送信」タスクの出力結果が得られないことを確認できます。
この結果は、「分岐」制御要素が定義した条件式を評価する際、ロジックフローの入力値として渡されたconditionの値をもとに処理を制御していることを表しています。

以上で、ロジックフローの入力値を条件として、処理を制御する定義が完了しました。

> **コラム**
>
> EL式で利用可能な暗黙的な変数
>
> 分岐条件の中で利用した$inputという変数は、IM-LogicDesignerが提供する、EL式で利用可能な暗黙的な変数の1つです。
> 暗黙的な変数の詳細は「IM-LogicDesigner仕様書」-「付録」-「EL式」-「利用可能な暗黙的な変数」を参照してください。

## 繰り返し処理を利用したフロー

この章では、ロジックフローの定義で繰り返し処理を利用する方法を説明します。



### 「繰り返し」制御要素

IM-LogicDesignerでは、フロー内で指定した条件、ないしは、回数の繰り返し処理を行う「繰り返し」制御要素を提供しています。
開発者は「繰り返し」制御要素を利用することで、同一処理の複数回実行を実現できます。

#### 本章で作成するフロー

本章では、「繰り返し」制御要素によって処理が複数回行われることを確認するため、「*基礎編 - ファースト・ステップ*」で作成したロジックフローを以下のように変更します。

- 「ログ出力」タスク、および、「テキストメール送信」タスクを、「繰り返し」制御要素で囲みます。
- 繰り返し実行する回数は、ロジックフローの入力値をもとに決定します。
  - この仕様に伴い、入力値に新しく実行回数を指定する値を定義します。

コラム

メール送信環境が未整備の場合

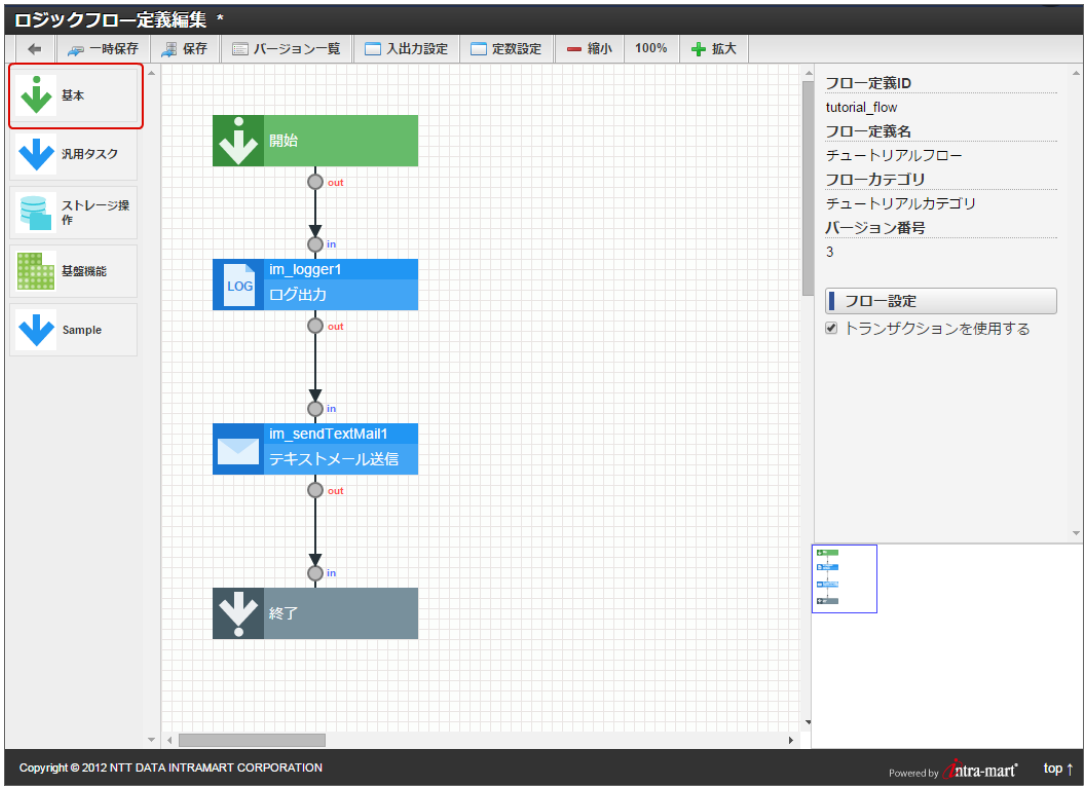
本チュートリアルは性質上、「テキストメール送信」タスクの処理が複数回実行されます。
メール送信環境が未整備の場合、一度のロジックフロー実行で多くの例外が出力されるので、ログ確認等の実行結果の確認が煩雑になることが考えられます。
必要に応じて「テキストメール送信」タスクを除外してください（動作確認を行う上で問題はありません）。

## 繰り返しを利用したフローの作成

繰り返しを利用したフローを実現する最初のステップとして、ロジックフローに繰り返し処理のエレメントを配置します。

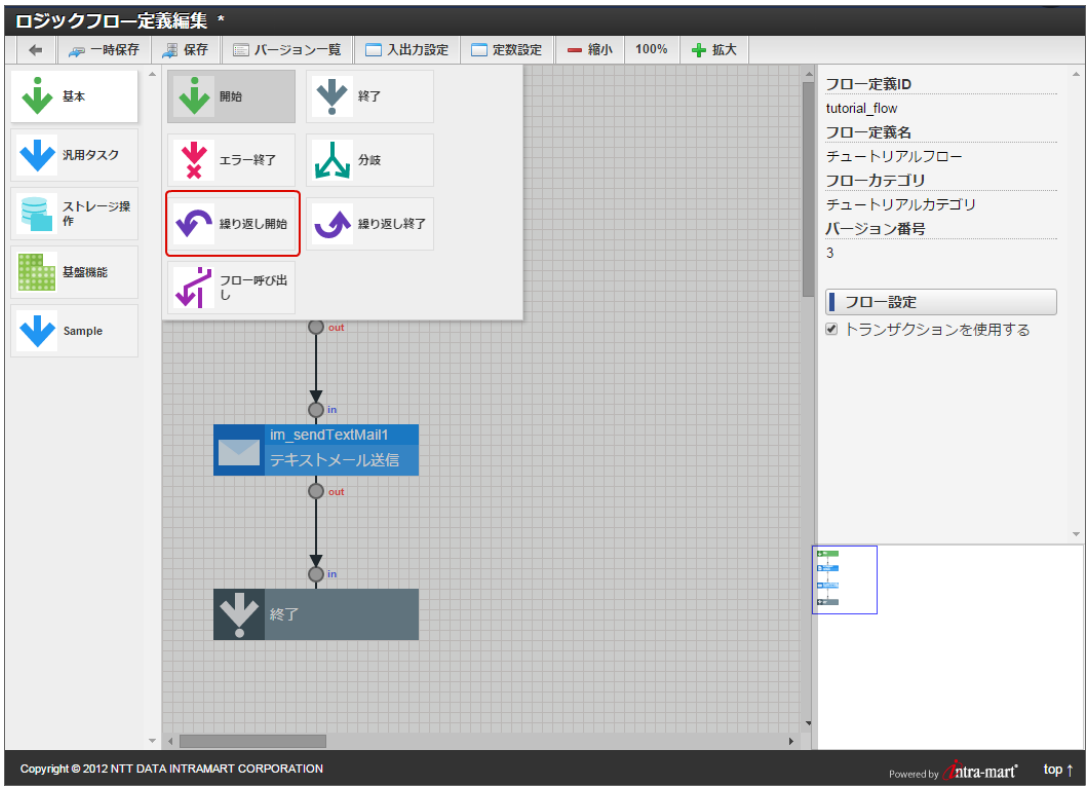
1. 「サイトマップ」→「LogicDesigner」→「フロー定義一覧」から、ロジックフロー定義一覧を開きます。
   左ペインのツリーから「チュートリアルカテゴリ」の開閉アイコンをクリックしカテゴリ配下を情報を表示します。
   フロー定義「チュートリアルフロー」を選択し編集ボタンをクリックし、ロジックフロー定義編集画面を開きます。

2. ロジックフロー定義編集画面左部、パレット内の「基本」へカーソルを合わせます。



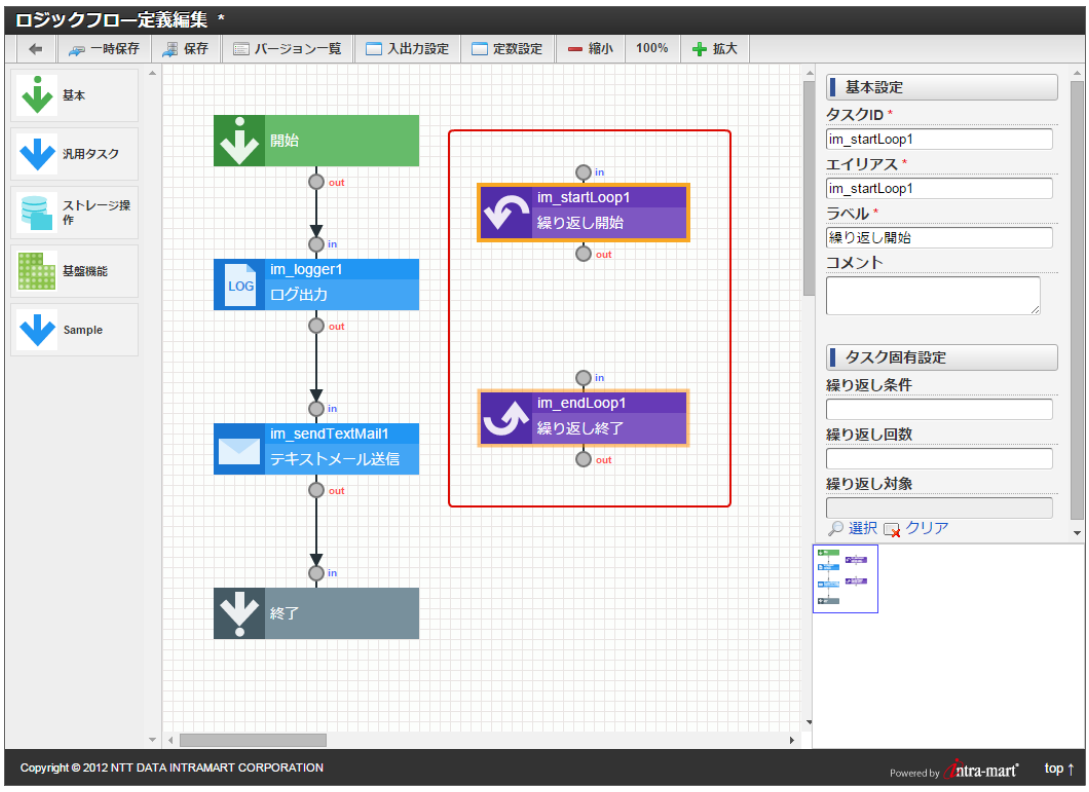

図：パレット内の「基本」

3. 「基本」にカテゴライズされる制御要素一覧がスライドインします。
   スライドインした一覧から、「繰り返し開始」をクリックします。




図：「繰り返し開始」制御要素をクリック

4. 繰り返し処理を行う制御要素（繰り返し開始、および、繰り返し終了）がフロー編集画面上に追加されます。



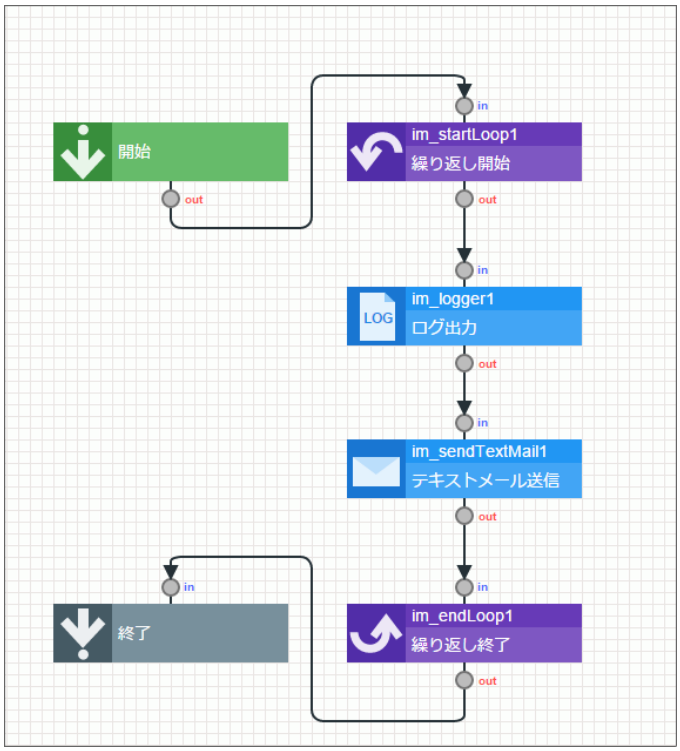

図：「繰り返し開始」「繰り返し終了」制御要素の追加

追加された「繰り返し開始」「繰り返し終了」制御要素を、既存のフローに組み込みます。
具体的な線（シーケンス）の定義内容は以下の通りです。

| 入力（始点） | 変更前の出力（終点） | 変更後の出力（終点） |
|---|---|---|
| 開始 - out | ログ出力 - in | 繰り返し開始 - in |
| 繰り返し開始 - out | （なし） | ログ出力 - in |
| テキストメール送信 - out | 終了 - in | 繰り返し終了 - in |

| 入力（始点） | 変更前の出力（終点） | 変更後の出力（終点） |
|---|---|---|
| 繰り返し終了 - out | （なし） | 終了 - in |

ロジックフローの完成例は以下の通りです。

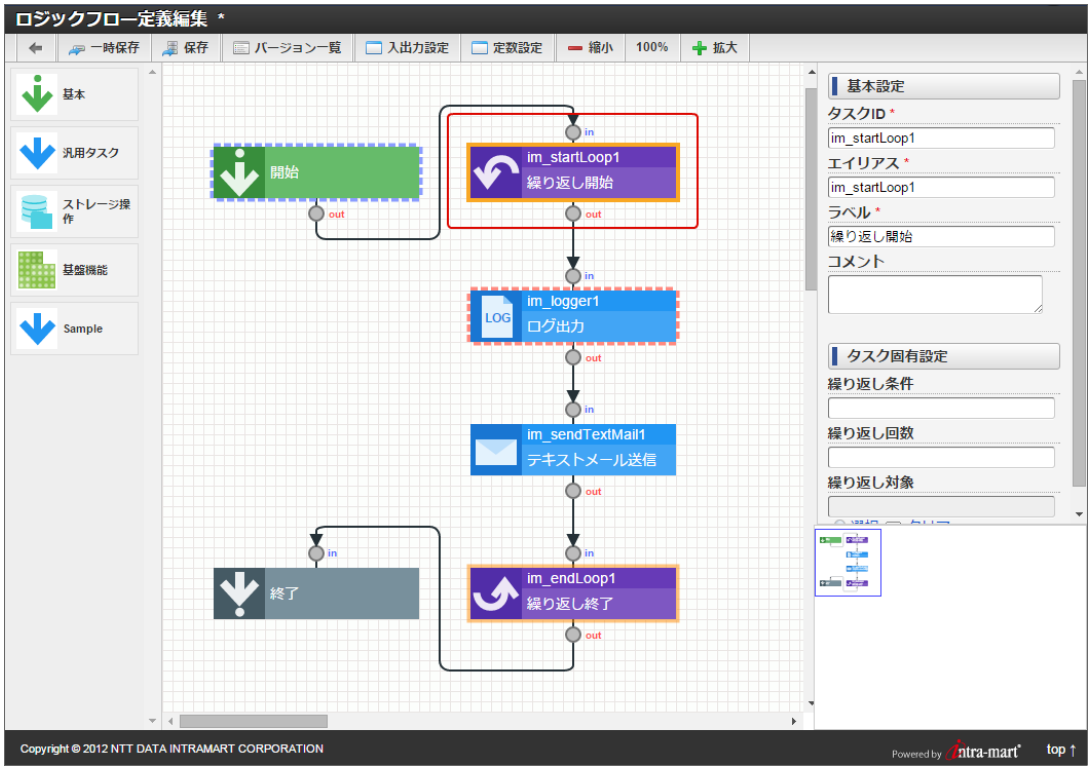



図：ロジックフロー（繰り返し）完成例

## 繰り返し条件の定義（回数）

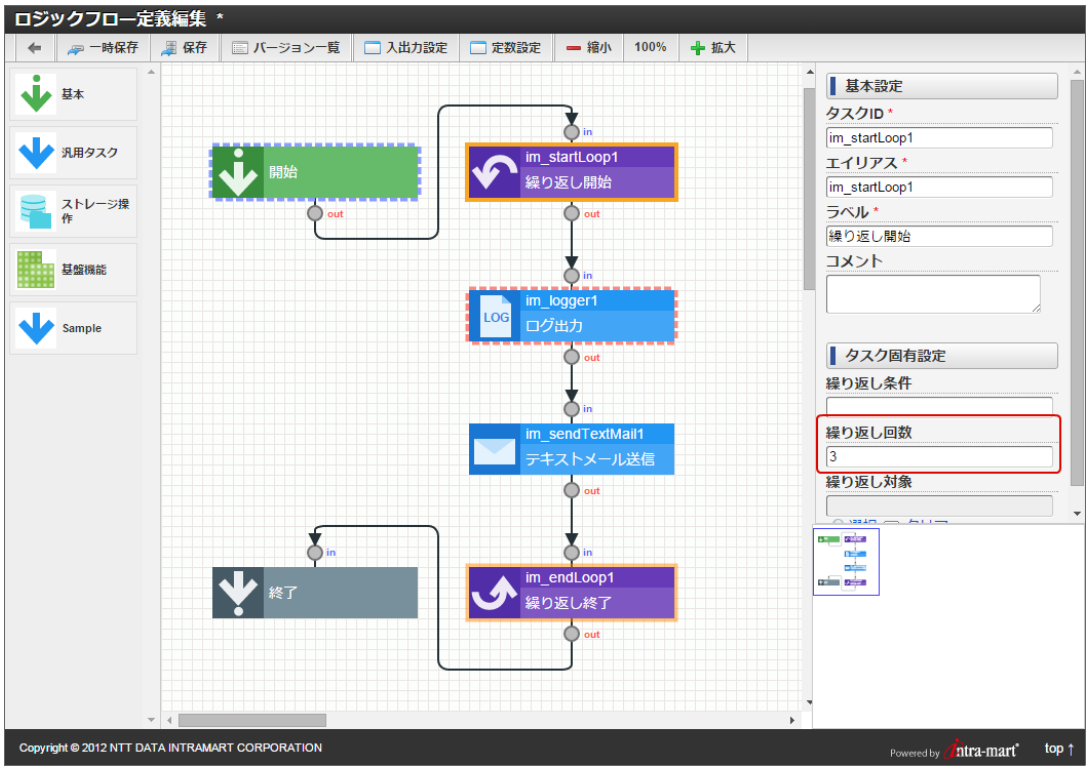
次に、実際に繰り返し条件を定義し、その条件のもとロジックフローが動作することを確認します。
ロジックフローにおける繰り返しの条件設定は、設定を行う「繰り返し開始」制御要素のプロパティから行います。
はじめに「繰り返し回数指定」での定義方法を説明します。

1. ロジックフロー定義編集画面上の「繰り返し開始」制御要素をクリックします。




図：「繰り返し開始」制御要素を選択

2. タスク固有設定の項目を以下のとおりに変更します。
   - 繰り返し回数 - 「3」




図：繰り返し回数の設定

3. ロジックフローを保存します。

以上で、繰り返し条件の定義（回数）が完了しました。

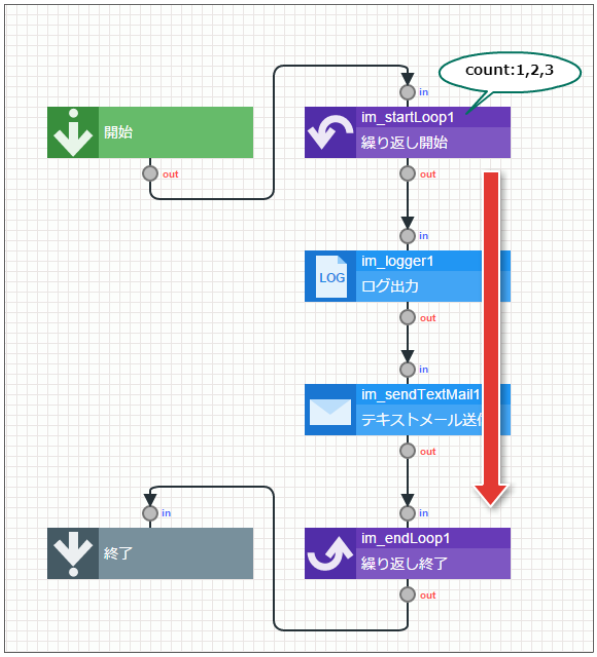
### 繰り返し条件の定義（回数）の結果の確認

保存が完了した後、作成したロジックフローを実行し、結果を確認します。
実行方法は「*Swagger(SPEC)から実行する*」を、結果の確認方法は「*結果を確認する*」を参照してください。

実行結果として「*基礎編 - ファースト・ステップ*」で作成したロジックフローの以下の内容が、3回出力されていることを確認してください。

- 「*「ログ出力」タスクの結果*」で確認したログ
- 「*「テキストメール送信」タスクの結果*」で確認したメール

これは「繰り返し」制御要素が、定義された繰り返し回数に応じて処理を制御していることを表しています。




図：「繰り返し」制御要素が回数を元に処理を制御するイメージ

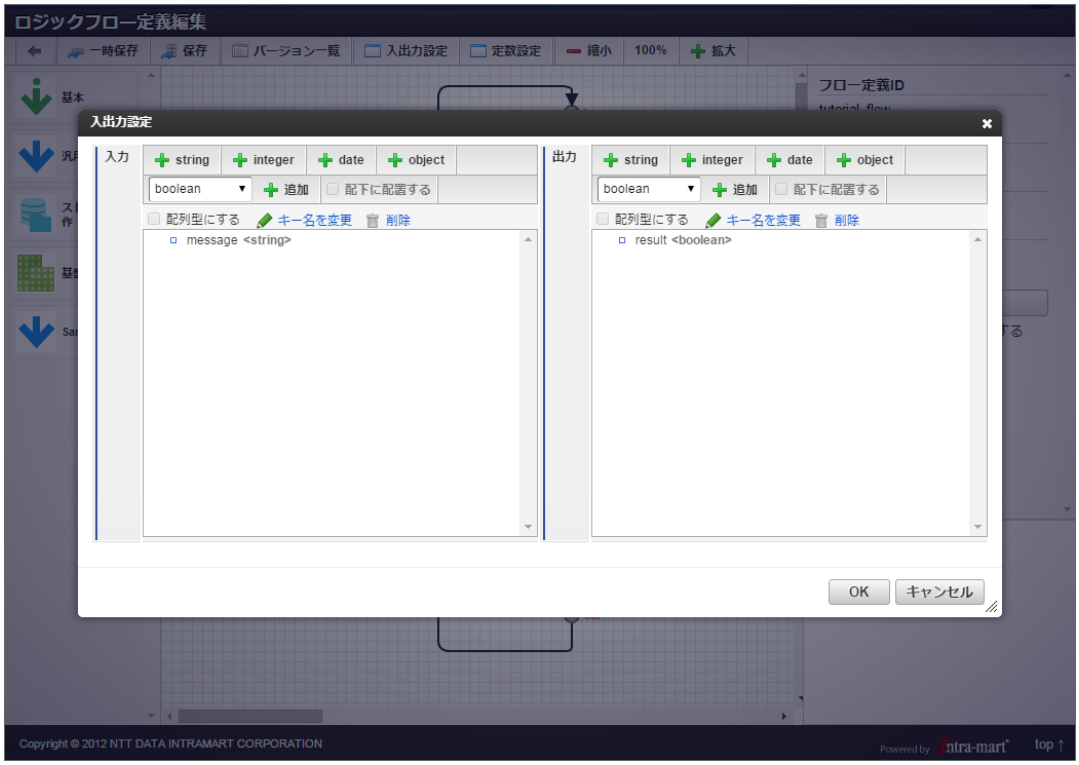
以上で、「繰り返し回数指定」を利用したロジックフローの繰り返し処理が確認できました。

## 繰り返し条件の定義（条件）

次に「繰り返し終了条件指定」での定義方法を説明します。
ここでは、繰り返す回数をロジックフローの入力値として受け取り、その条件を元に繰り返し処理が行われることを確認します。

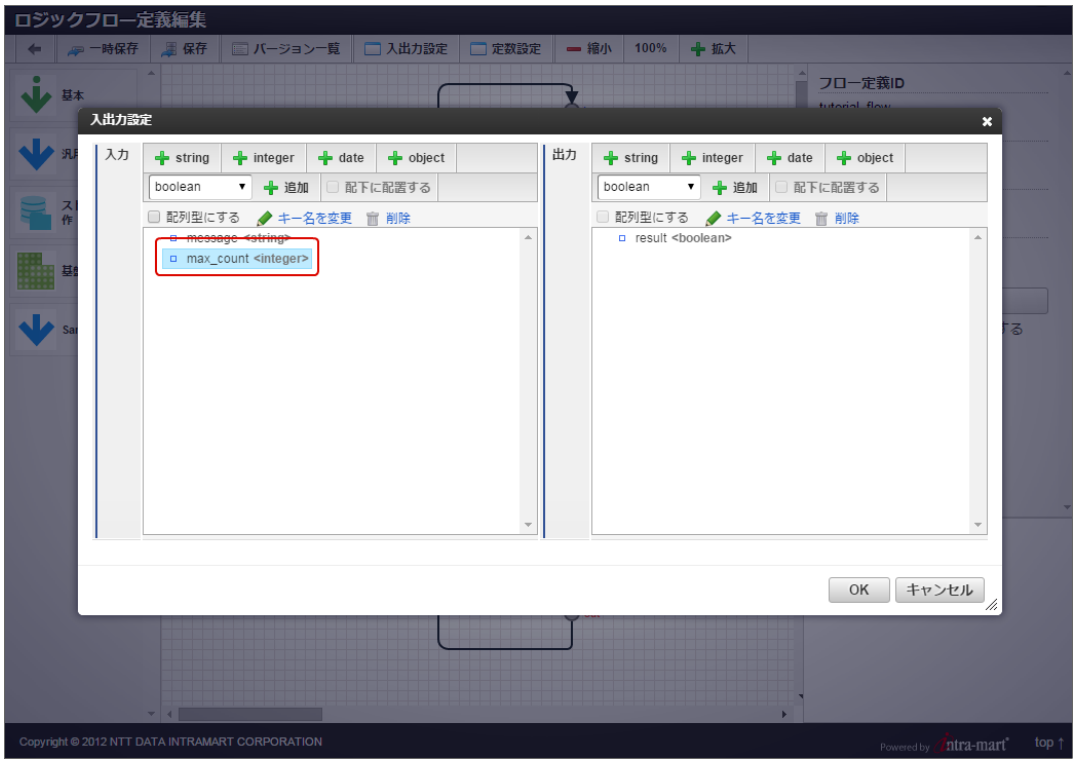
繰り返し終了条件を設定する前に、ロジックフローの入力値として新しく「繰り返しを行う回数」を表すパラメータを定義します。

1. ロジックフロー定義編集画面上部、ヘッダ内の「入出力設定」をクリックし、入出力設定画面を開きます。




図：入出力設定画面の表示

2. 入力として以下のとおりにパラメータを追加します。
   - パラメータ名
     - max_count
   - 型
     - integer

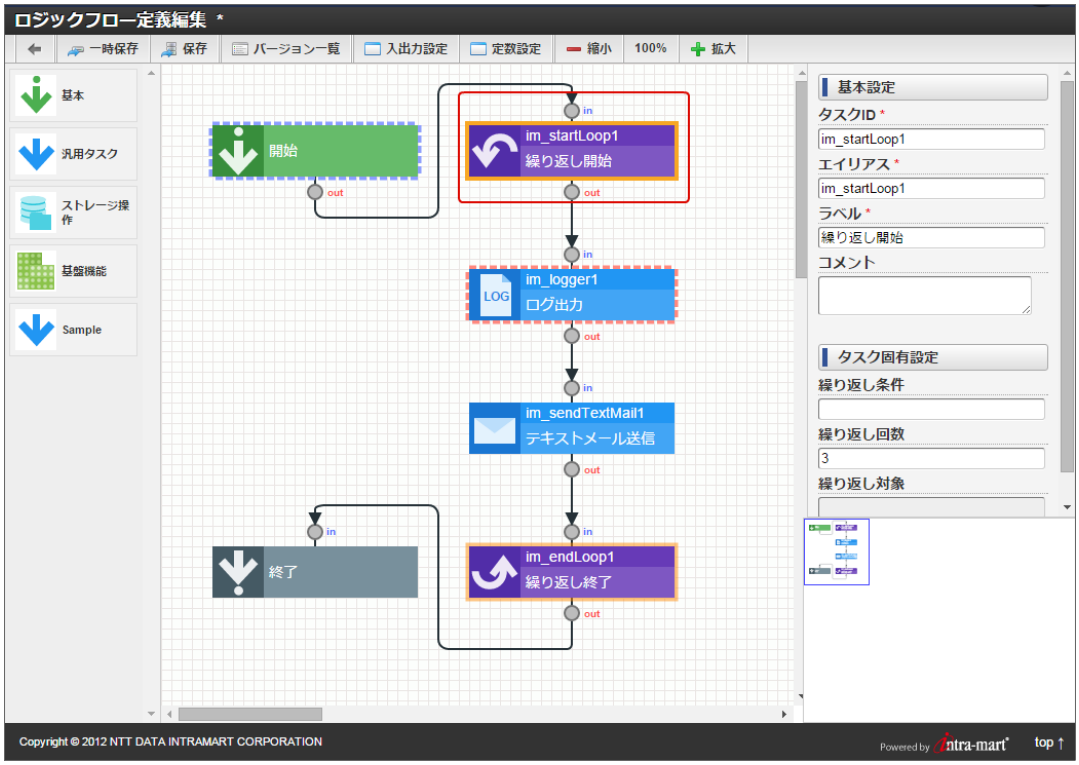



図：新規パラメータの追加（integer）

3. 入出力設定画面右下のOKをクリックします。

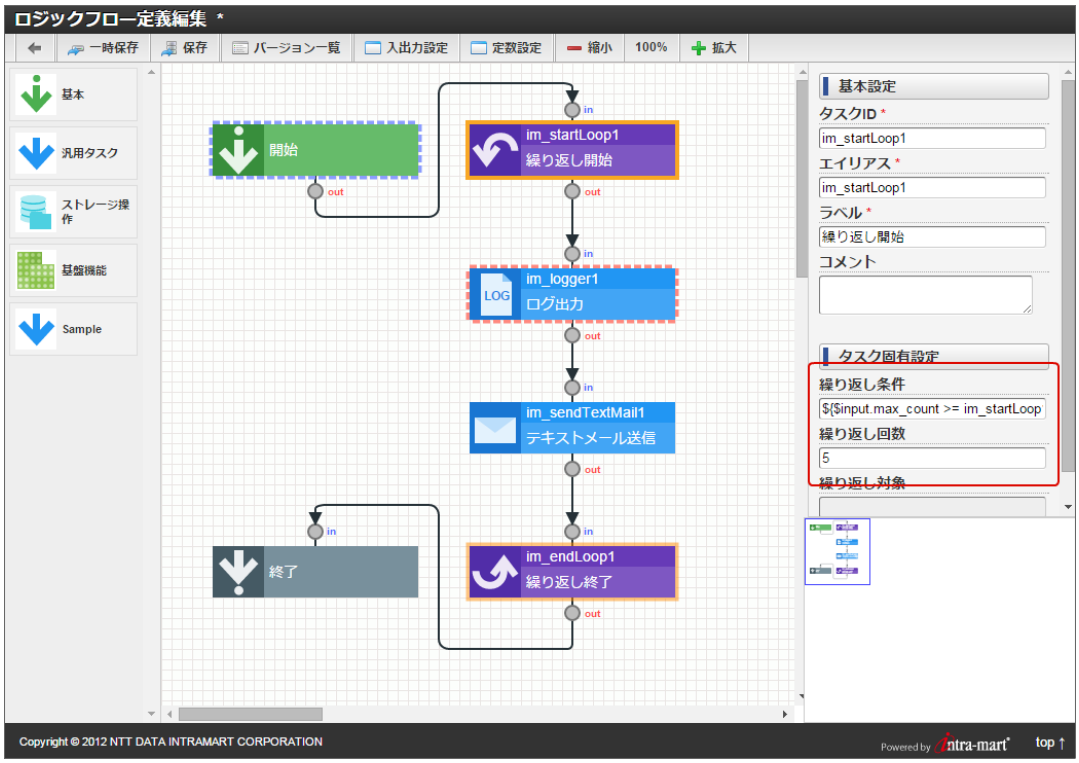
次に実際の「繰り返し終了条件」を指定します。

1. ロジックフロー定義編集画面上の「繰り返し開始」制御要素をクリックします。




図：「繰り返し開始」制御要素の選択

2. タスク固有設定の項目を以下のとおりに変更します。
   - 繰り返し条件 - 「${$input.max_count >= im_startLoop1.index}」
   - 繰り返し回数 - 5




図：繰り返し条件、および、繰り返し回数の設定

3. ロジックフローを保存します。

以上で、繰り返し条件の定義（条件）が完了しました。

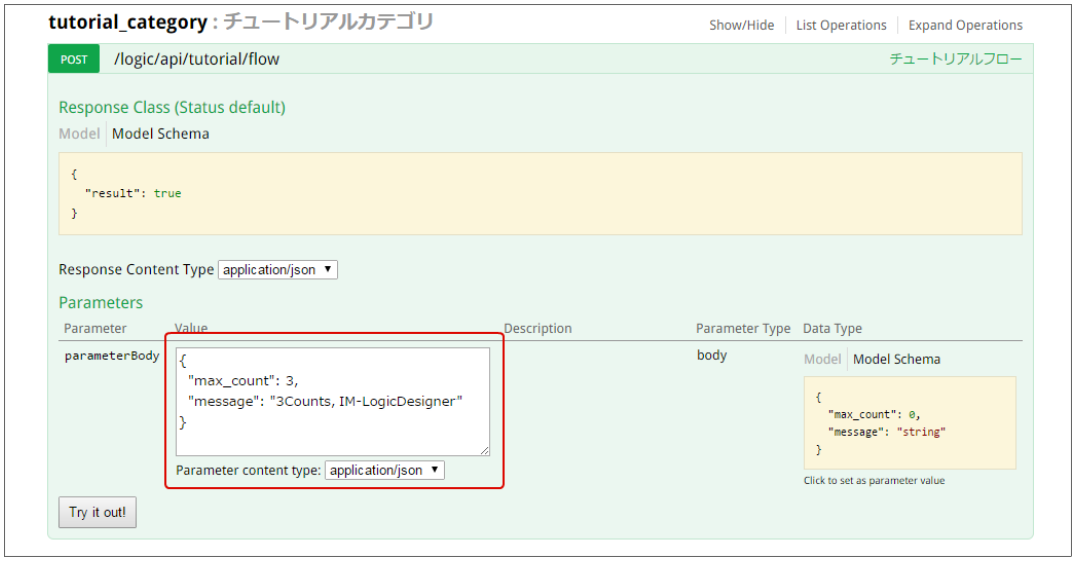
繰り返し条件の定義（条件）の結果の確認

保存が完了した後、作成したロジックフローを実行し、結果を確認します。
実行方法は「*Swagger(SPEC)から実行する*」を、結果の確認方法は「*結果を確認する*」を参照してください。

今回、新しくフローの入力値を定義したため、実行時に指定するパラメータが追加されます。
ここでは条件の定義が有効であることと、条件と回数の関係性を確認するために、２パターンの実行結果を確認します。

はじめに以下の内容でロジックフローを実行します。

- max_count - 3
- message - 3Counts, IM-LogicDesigner

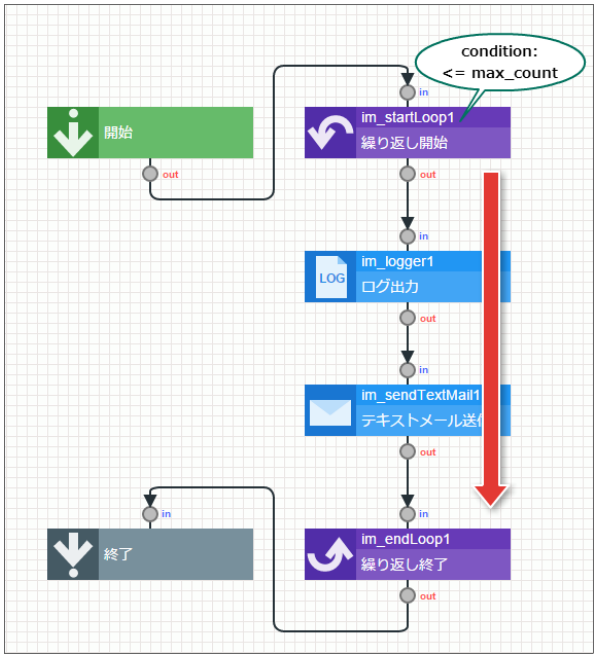



図：パラメータ設定例（その１）

実行結果として「*基礎編 - ファースト・ステップ*」で作成したロジックフローの以下の内容が、3回出力されていることを確認してください。

- 「*「ログ出力」タスクの結果*」で確認したログ
- 「*「テキストメール送信」タスクの結果*」で確認したメール

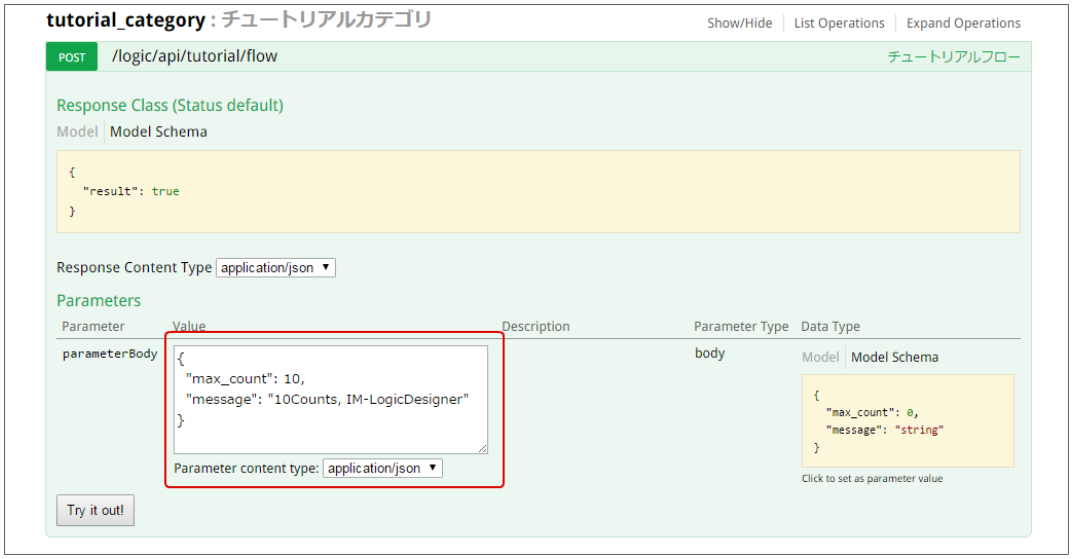
これは「繰り返し」制御要素が、定義された繰り返し条件に従い、現在の繰り返し回数（im_startLoop1.index）が繰り返しを行う回数（$input.max_count）以上になるまで繰り返し処理を制御していることを表しています。




図：「繰り返し」制御要素が条件を元に処理を制御するイメージ

次に以下の内容でロジックフローを実行します。

- max_count - 10
- message - 10Counts, IM-LogicDesigner

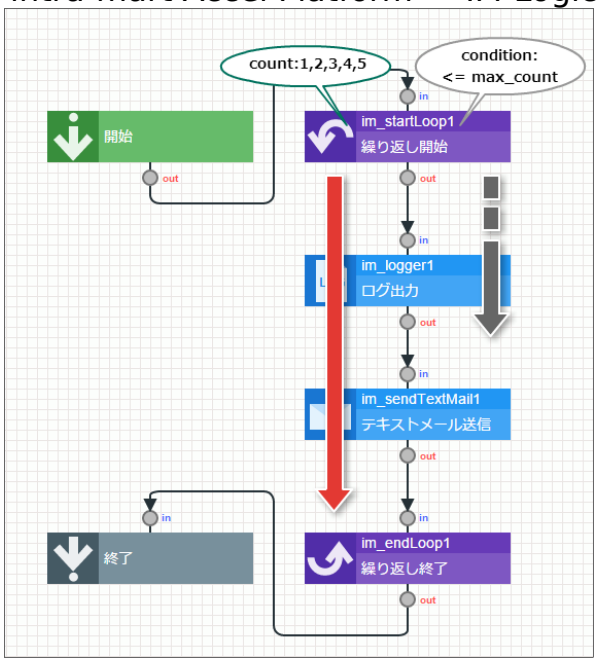



図：パラメータ設定例（その２）

実行結果として「*基礎編 - ファースト・ステップ*」で作成したロジックフローの以下の内容が、5回出力されていることを確認してください。

- 「*「ログ出力」タスクの結果*」で確認したログ
- 「*「テキストメール送信」タスクの結果*」で確認したメール

これは「繰り返し」制御要素が、定義された繰り返し条件と繰り返し回数の両方を精査し、先に繰り返し回数（5回）の条件を満たしたため、 繰り返し処理を制御したことを表しています。




図：「繰り返し」制御要素が条件と回数を元に処理を制御するイメージ

この動作仕様は、例えば繰り返し条件に意図せず長大な数を指定されてしまった場合のストッパーとして、 繰り返し回数にフローとして許容できる最大繰り返し回数を定義しておくといった運用が可能です。

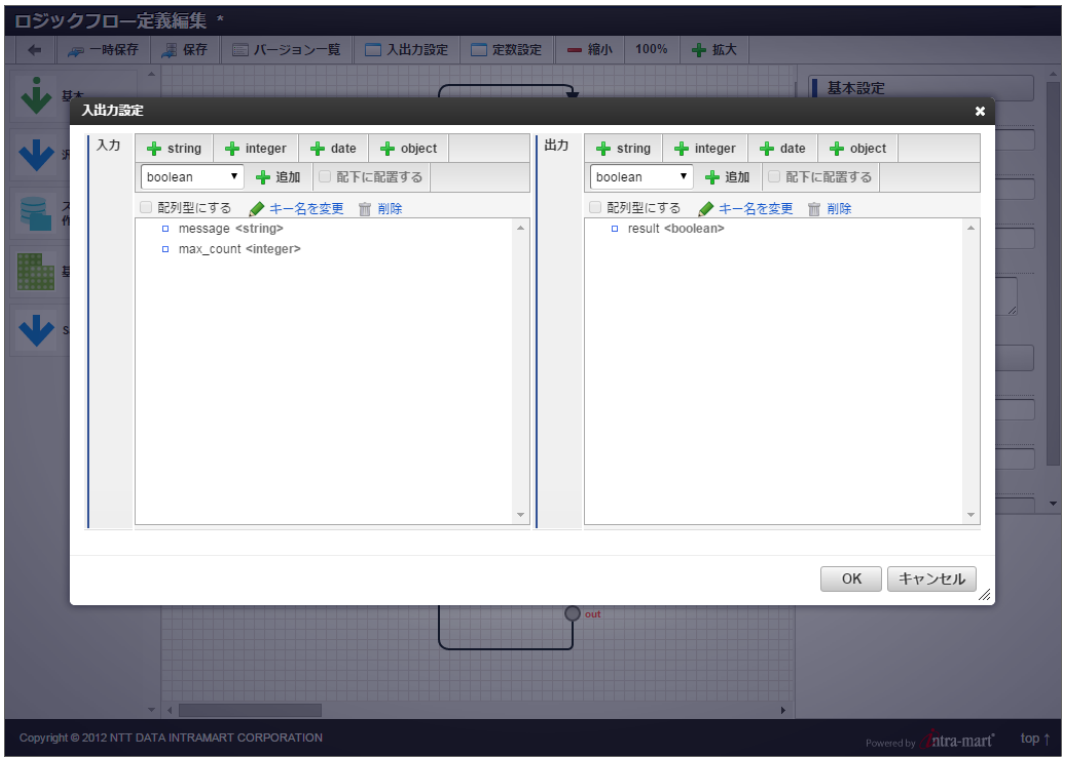
以上で、「繰り返し終了条件指定」を利用したロジックフローの繰り返し処理が確認できました。

## 繰り返し処理対象の指定と値の利用

最後に、繰り返し処理を行う対象を指定し、処理中にその値を利用する方法を説明します。
IM-LogicDesignerでは、一般的なプログラミング言語のように繰り返し処理中の値を利用できます。
ここでは、ロジックフローの入力値として設定済みの本文（`message<string>`）を複数指定可能に変更し、指定された本文の数だけ繰り返し処理を行う方法を説明します。

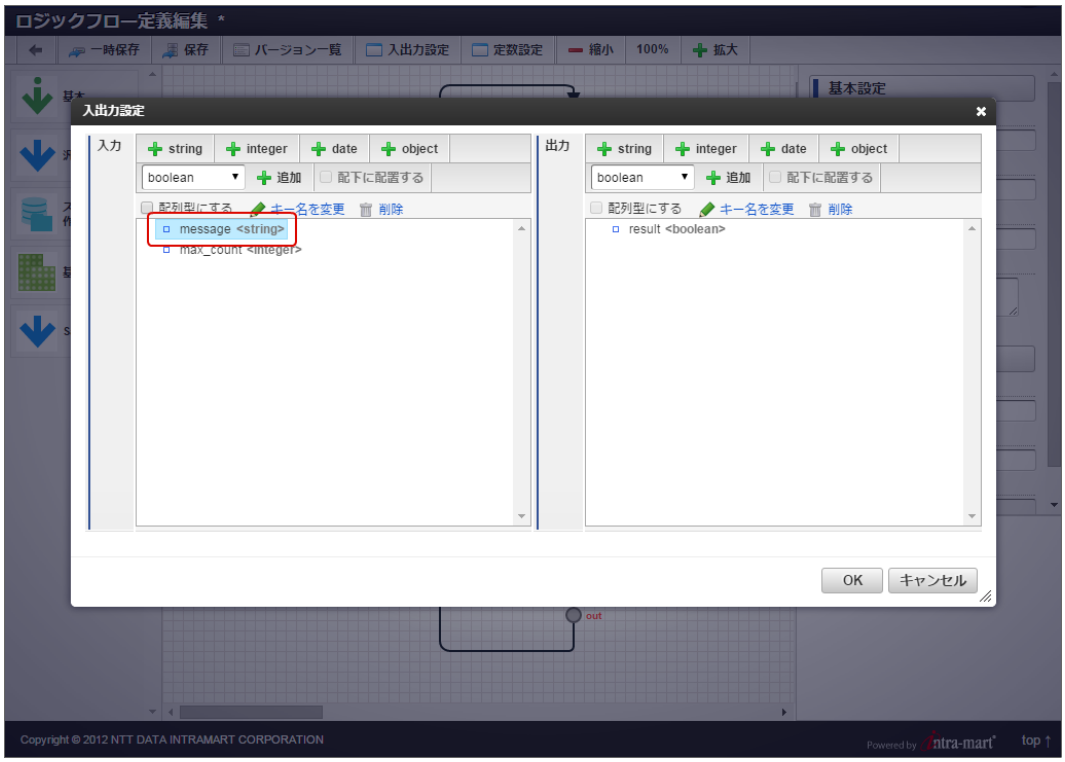
はじめに、ロジックフローの入力値を編集します。

1. ロジックフロー定義編集画面上部、ヘッダ内の「入出力設定」をクリックし、入出力設定画面を開きます。



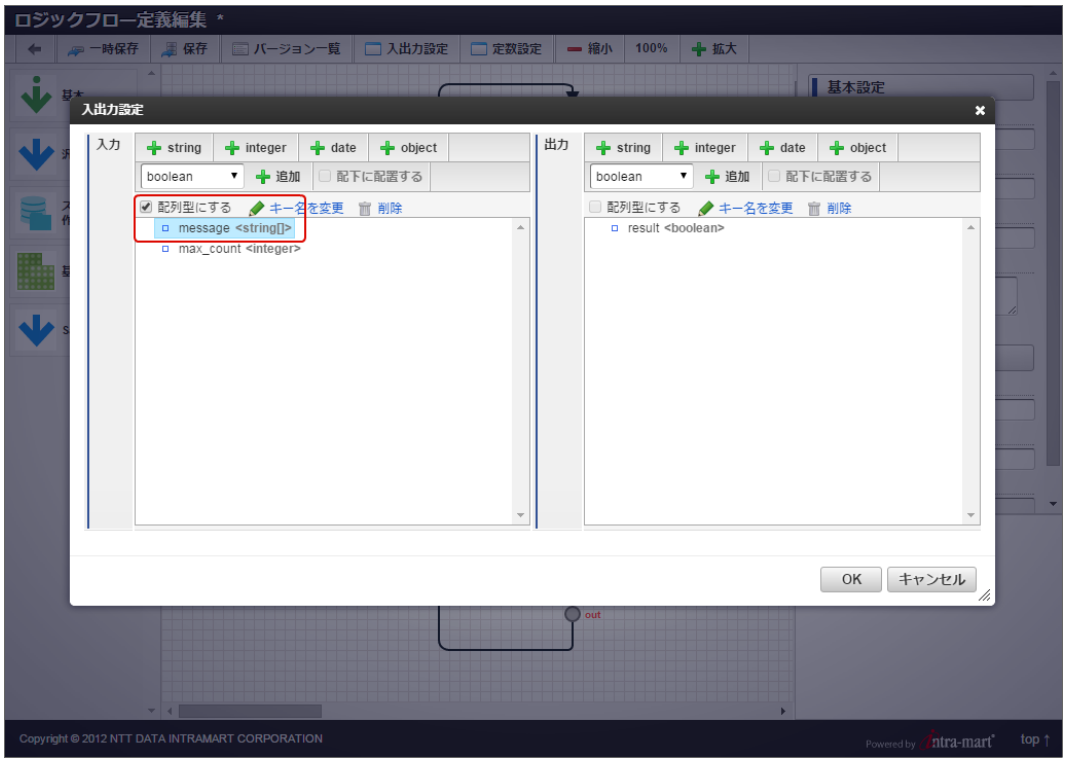

図：入出力設定画面の表示

2. 入力として設定済みの「`message<string>`」をクリックし、選択状態とします。




図：messageプロパティの選択

3. 入力ペイン上部にある「配列型にする」チェックボックスにチェックを入れます。
   チェックを入れたタイミングで、選択していたmessageの型が配列型（string[]）に変更されたことを確認してください。

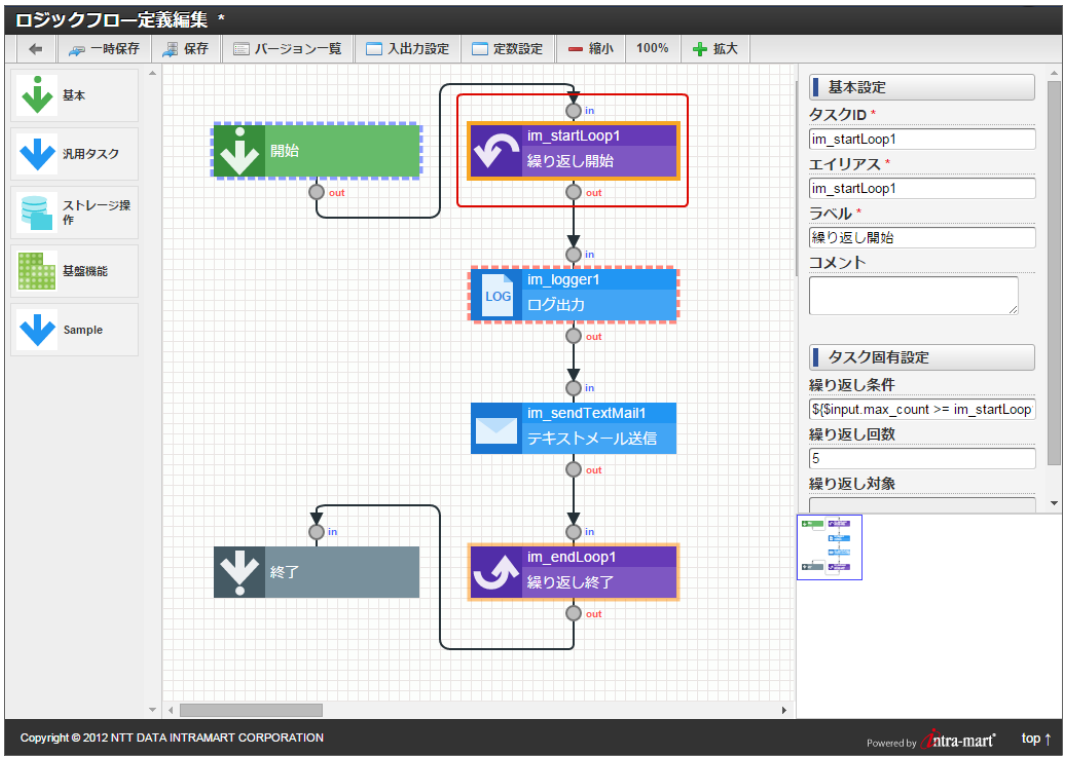



図：配列型への変更

4. 入出力設定画面右下のOKをクリックします。

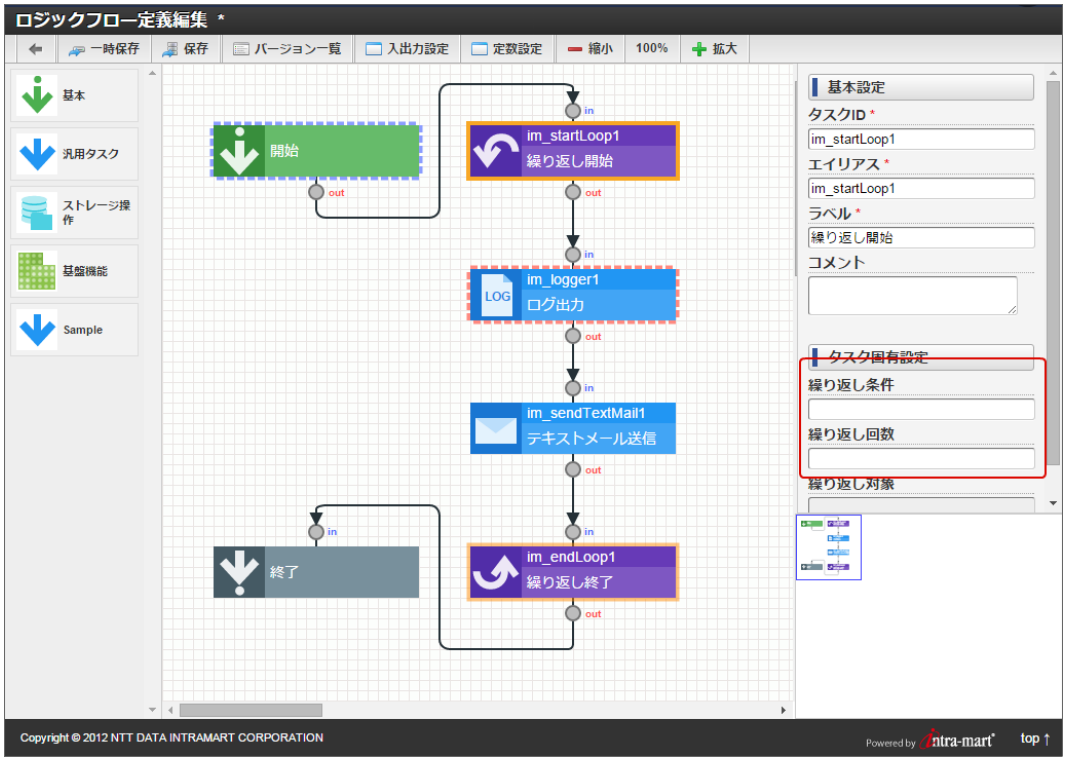
次に、変更を行った本文（message<string[]>）を繰り返しの対象に設定します。

1. ロジックフロー定義編集画面上の「繰り返し開始」制御要素をクリックします。



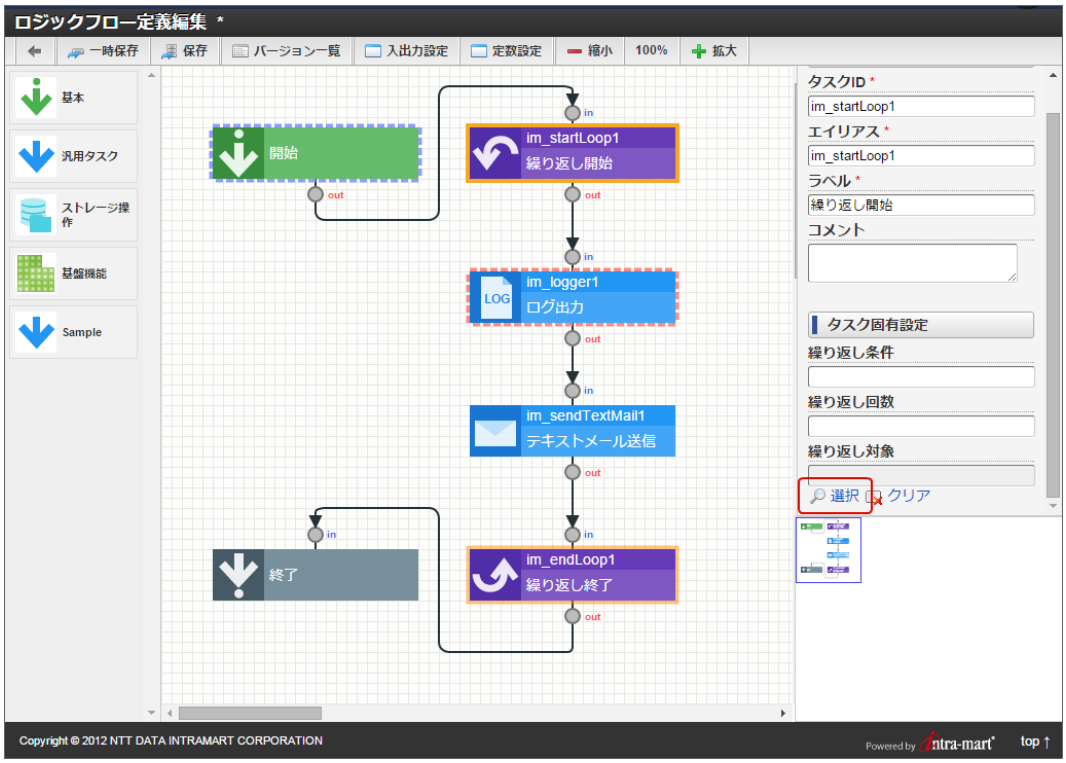

図：「繰り返し開始」制御要素の選択

2. タスク固有設定の項目のうち、以下の項目に設定された内容を消去します。
   - 繰り返し条件
   - 繰り返し回数



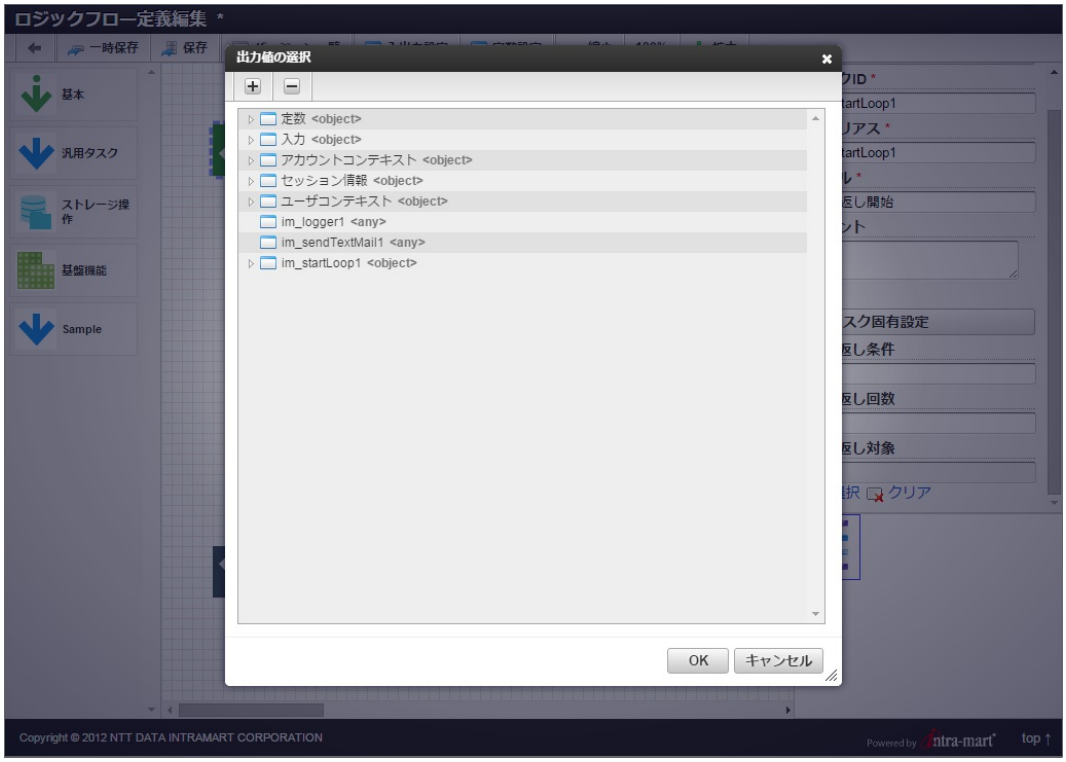

図：既存の条件、および、回数の削除

3. 繰り返し対象を設定するため、タスク固有設定の項目のうち「繰り返し対象」の下部にある検索リンクをクリックします。



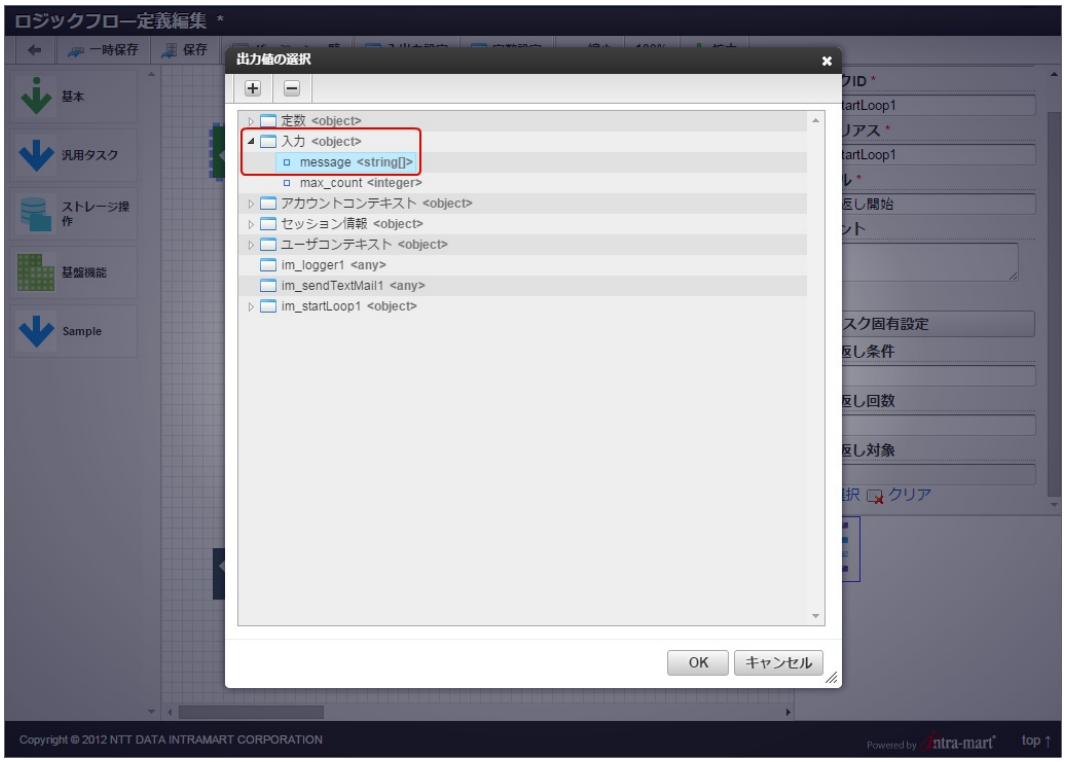

図：繰り返し対象の検索

4. 繰り返し対象を選択する、出力値の設定画面が表示されます。



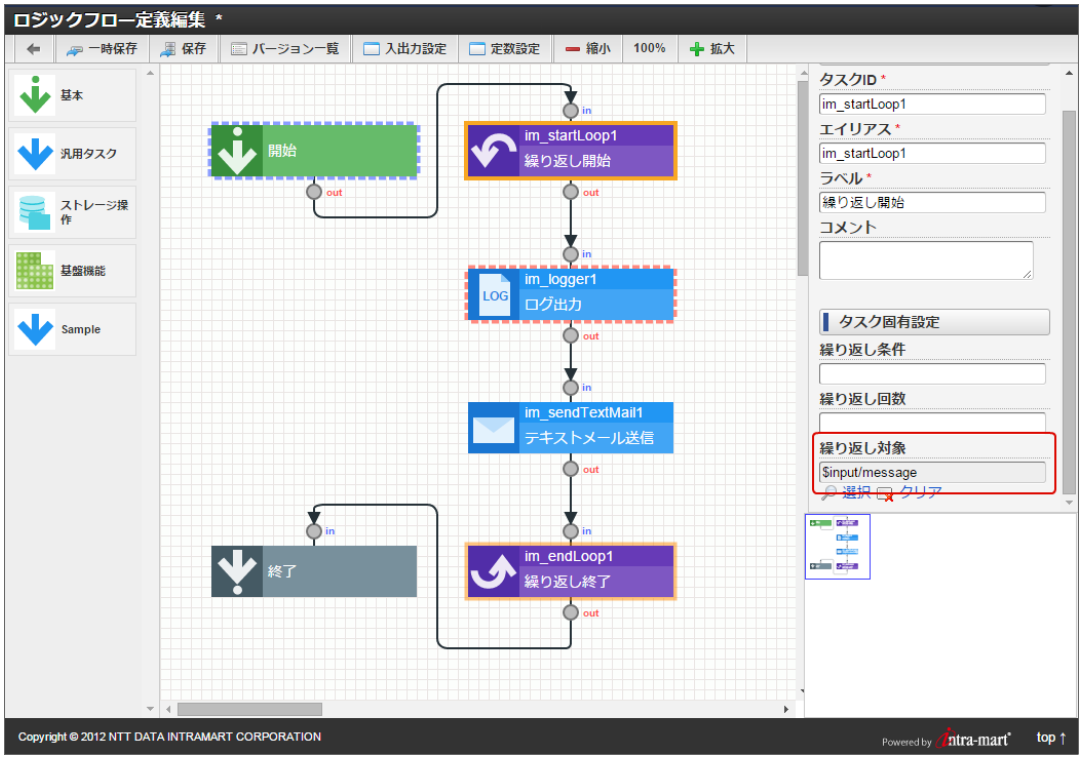

図：繰り返し対象選択画面

5. 繰り返しの対象として以下の項目を選択します。
   - 入力<object> - message<string[]>




図：繰り返し対象の選択

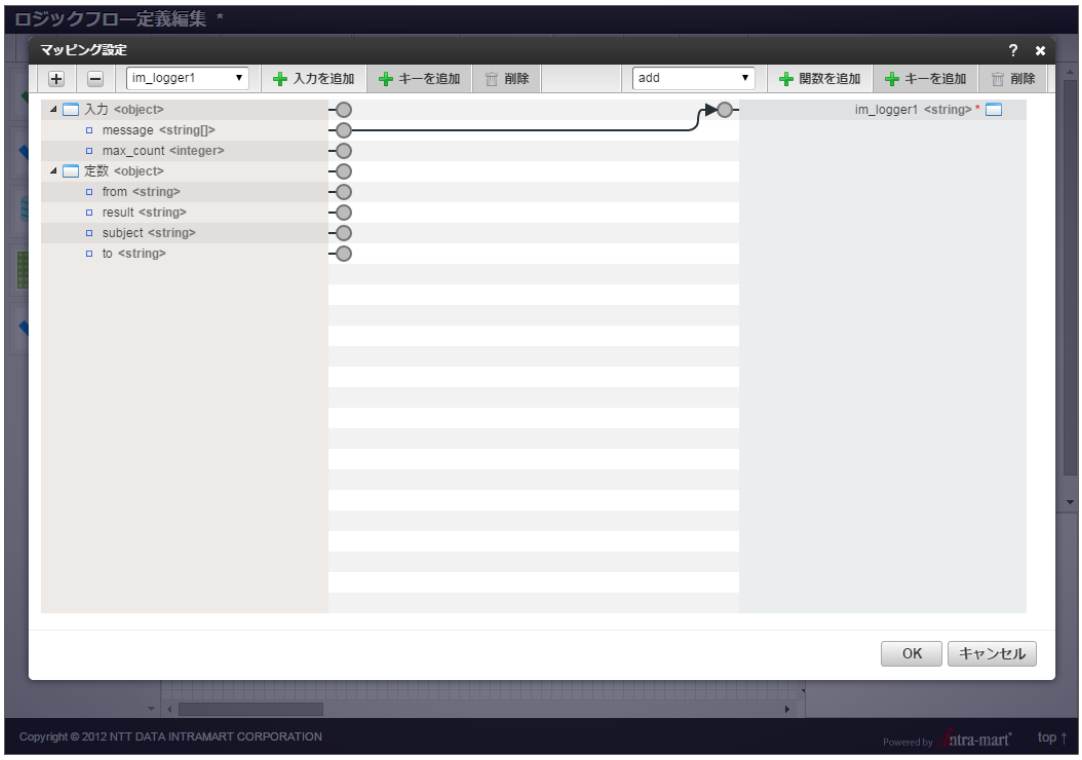
6. 設定画面下部の「OK」をクリックします。
7. 出力値の設定画面が消え、繰り返し対象の項目に以下の値が入力されます。
   - $input/message




図：選択した繰り返し対象の確認

最後に、設定した繰り返し対象の値のマッピング設定を行います。

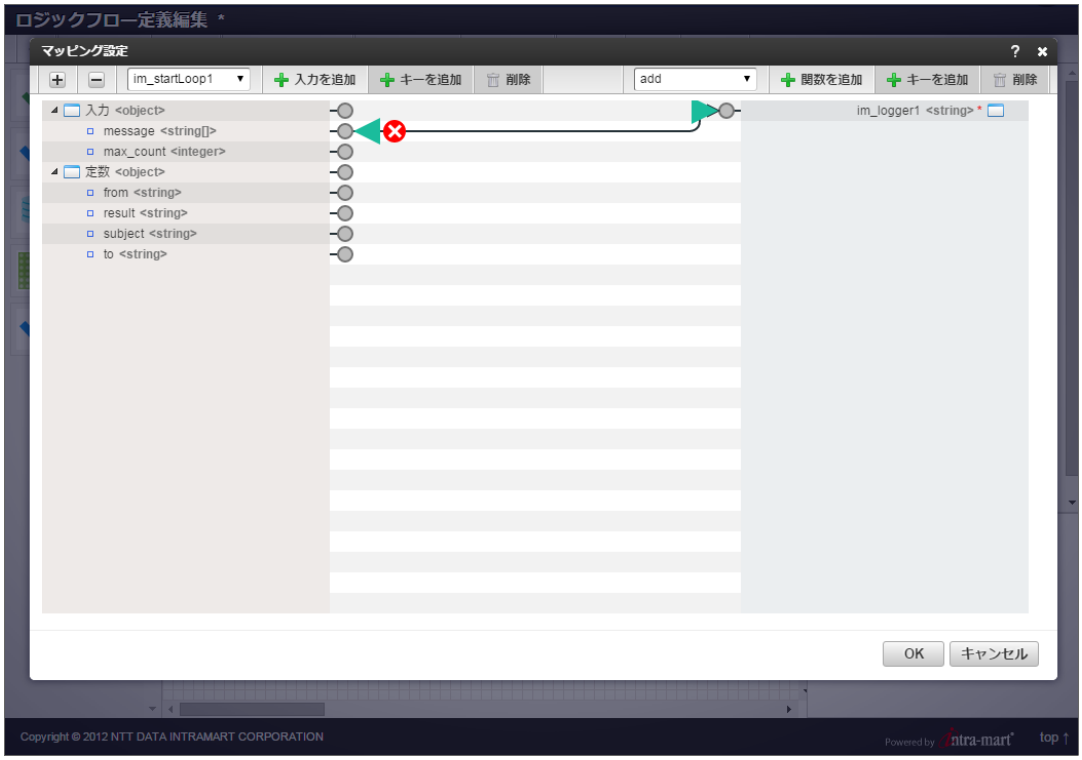
1. フロー編集画面上の「ログ出力」タスクをクリックした上で、「マッピング設定」をクリックし、マッピング設定画面を開きます。




図：「ログ出力」マッピング設定画面

2. 設定済みの以下のマッピング設定を削除します。

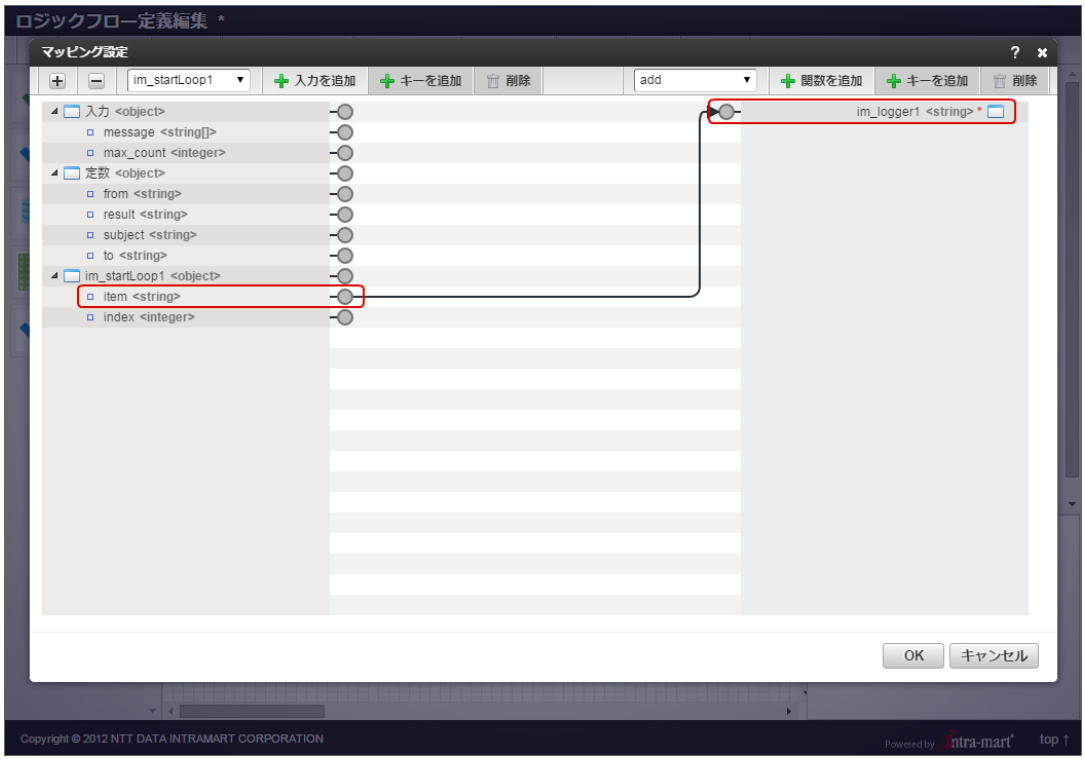
| 入力（始点） | 出力（終点） |
| --- | --- |
| 入力<object> - message<string[]> | im_logger1<string> |




図：既存のマッピング設定の削除

3. 新たにマッピング設定を以下のとおりに設定します。

| 入力（始点） | 出力（終点） |
| --- | --- |
| im_startLoop1<object> - item<string> | im_logger1<string> |




図：マッピング設定の追加

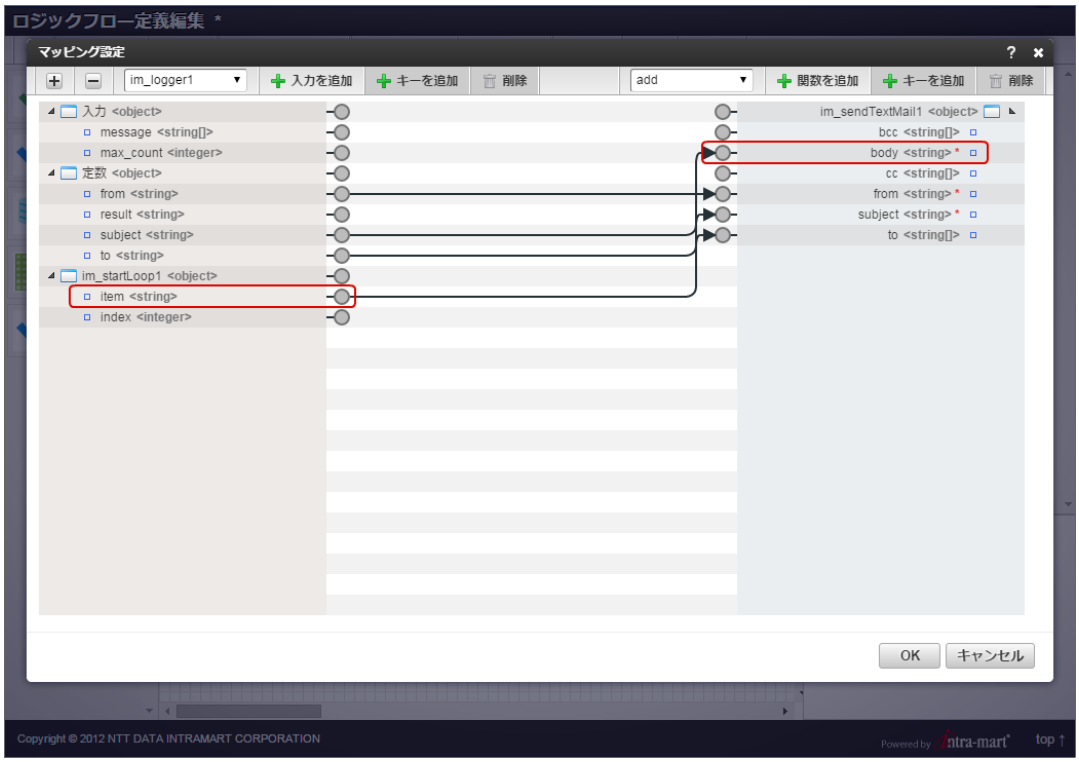
> **コラム**
>
> 入力として繰り返し開始（im_startLoop1<object>）を利用する方法
>
> 繰り返し開始の出力値を、「ログ出力」タスクの入力値として利用する方法の詳細は「*応用編 - より高度なフロー*」 - 「*様々な入力情報の利用*」 - 「*他のエレメントの出力値の利用*」を参照してください。

4. 設定画面右下のOKをクリックし、「ログ出力」タスクのマッピング設定を終了します。
5. 同様の手順で、「テキストメール送信」タスクのマッピング設定画面を開き、以下のとおりに設定します。
   - 設定の削除

| 入力（始点） | 出力（終点） |
|---|---|
| 入力<object> - message<string[]> | im_sendTextMail1<object> - body<string> |

   - 設定の追加

| 入力（始点） | 出力（終点） |
|---|---|
| im_startLoop1<object> - item<string> | im_sendTextMail1<object> - body<string> |




図：「テキストメール送信」タスクのマッピング設定変更

6. 設定画面右下のOKをクリックし、「テキストメール送信」タスクのマッピング設定を終了します。
7. ロジックフローを保存します。

以上で、繰り返し対象の指定と、その値の利用の設定が完了しました。

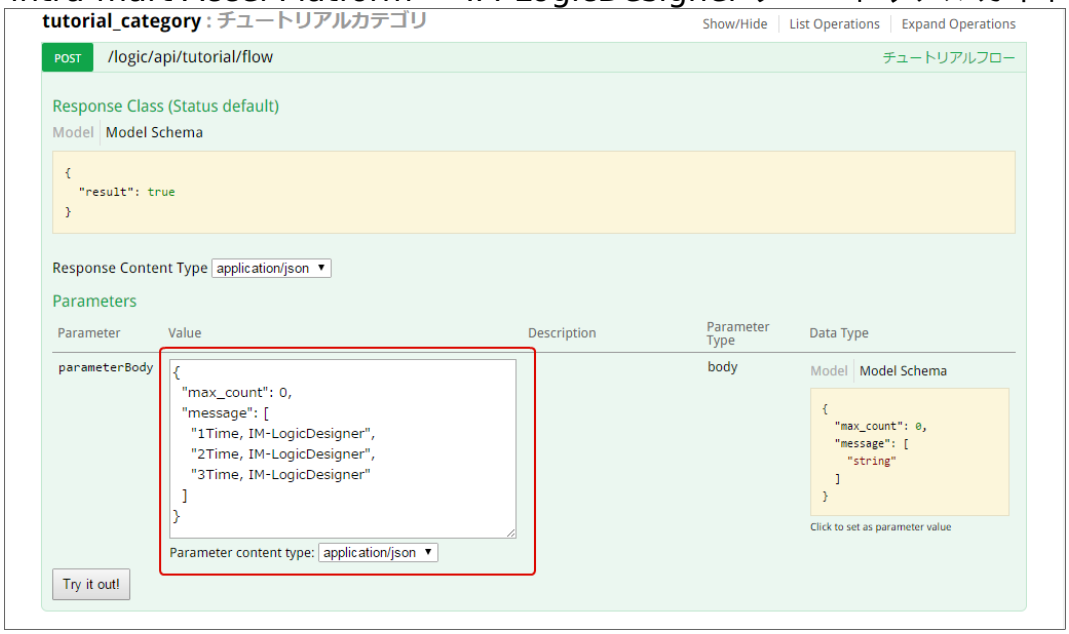
### 繰り返し処理対象の指定と値の利用の結果の確認

保存が完了した後、作成したロジックフローを実行し、結果を確認します。
実行方法は「*Swagger(SPEC)から実行する*」を、結果の確認方法は「*結果を確認する*」を参照してください。

今回、フローの入力値を一部変更したため、実行時に指定するパラメータが基礎編のチュートリアルとは異なります。
ここでは本文として以下の３つの文章を設定し、それを元に繰り返し処理が行われることを確認します。

Swagger画面を開き、以下の内容でロジックフローを実行します。

- max_count - 0
- message
  - 1Time, IM-LogicDesigner
  - 2Time, IM-LogicDesigner
  - 3Time, IM-LogicDesigner




図：複数の本文をロジックフローに渡して実行

> **コラム**
>
> パラメータの設定について
>
> 今回のパラメータ（JSON）の設定方法は複雑なため、定義方法がわからない場合は以下のテキストをコピーして利用してください。
>
> ```
> {
>  "max_count": 0,
>  "message": [
>  "1Time, IM-LogicDesigner",
>  "2Time, IM-LogicDesigner",
>  "3Time, IM-LogicDesigner"
>  ]
> }
> ```

実行結果として「*基礎編 - ファースト・ステップ*」で作成したロジックフローの以下の内容が、3回出力されていること、および、設定した3つの文章がそれぞれ1回ずつ出力されていることを確認してください。

- 「*「ログ出力」タスクの結果*」で確認したログ
- 「*「テキストメール送信」タスクの結果*」で確認したメール

これは「繰り返し」制御要素が、定義された繰り返し対象の数だけ繰り返し処理を制御したことを表しています。
今回は3つの文章を設定したため、三度繰り返し実行されました。

以上で、繰り返し処理対象の指定と値の利用の結果の確認ができました。

> **コラム**
>
> 繰り返し対象との複合条件
>
> 本チュートリアルでは説明を省きましたが、繰り返し対象の設定は「*繰り返し条件の定義（回数）*」および「*繰り返し条件の定義（条件）*」との複合条件の設定が可能です。
> 複数の条件を設定した場合、それらの内一番最初に条件から外れたタイミングで繰り返し処理は終了します。
>
> 例：繰り返し回数に3を指定し、繰り返し対象が5つあった場合、繰り返し回数が3回の段階で処理は終了します。

## サブフロー呼び出し

この章では、ロジックフローの定義で別のロジックフロー（サブフロー）を呼び出す方法を説明します。

## 「フロー呼び出し」制御要素

IM-LogicDesignerでは、あるフロー内で別のフローの呼び出し処理を行う「フロー呼び出し」制御要素を提供しています。
開発者は「フロー呼び出し」制御要素を利用することで、他のフローで実現した機能を容易に別フローに組み込む事ができます。

本章で作成するフロー

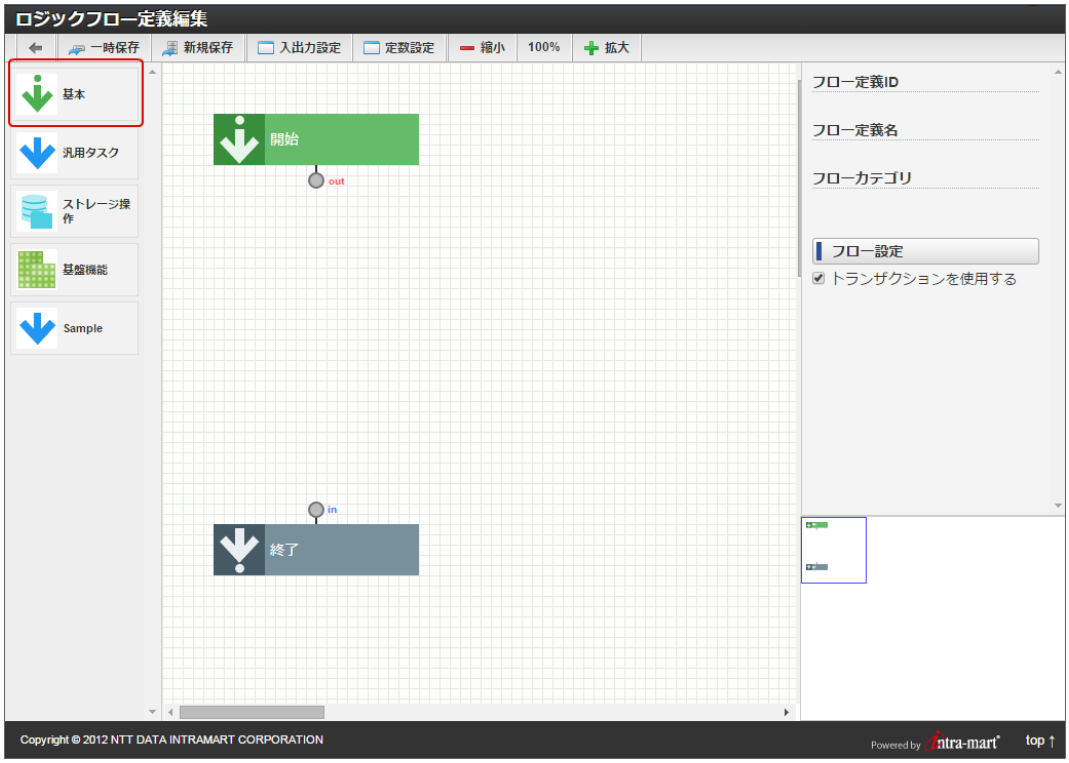
本章では、「フロー呼び出し」制御要素によって、作成済みの別ロジックフロー処理を呼び出せることを確認するため、以下のようなフローを作成します。

- 「*基礎編 - ファースト・ステップ*」で作成したロジックフローを「呼び出し先」ロジックフローとします。
  - 本章のチュートリアルでは、基礎編のチュートリアルで作成したロジックフローに対する編集は行いません。
- 「*基礎編 - ファースト・ステップ*」で作成したロジックフローとは別に新たにロジックフローを作成します。
- 新しく作成したロジックフローを「呼び出し元」ロジックフローとし、「呼び出し先」ロジックフローを呼び出す処理のみ定義します。

### フロー呼び出しを利用したフローの作成

フロー呼び出しを利用したフローを新しく作成します。
フロー呼び出しを行うためには、ロジックフローにフロー呼び出しを行うエレメントを配置します。

1. 「サイトマップ」→「LogicDesigner」→「フロー定義一覧」から、ロジックフロー定義一覧を開きます。
   ロジックフロー定義一覧画面左上の「ロジックフロー新規作成」をクリックします。
2. ロジックフロー定義編集画面左部、パレット内の「基本」へカーソルを合わせます。

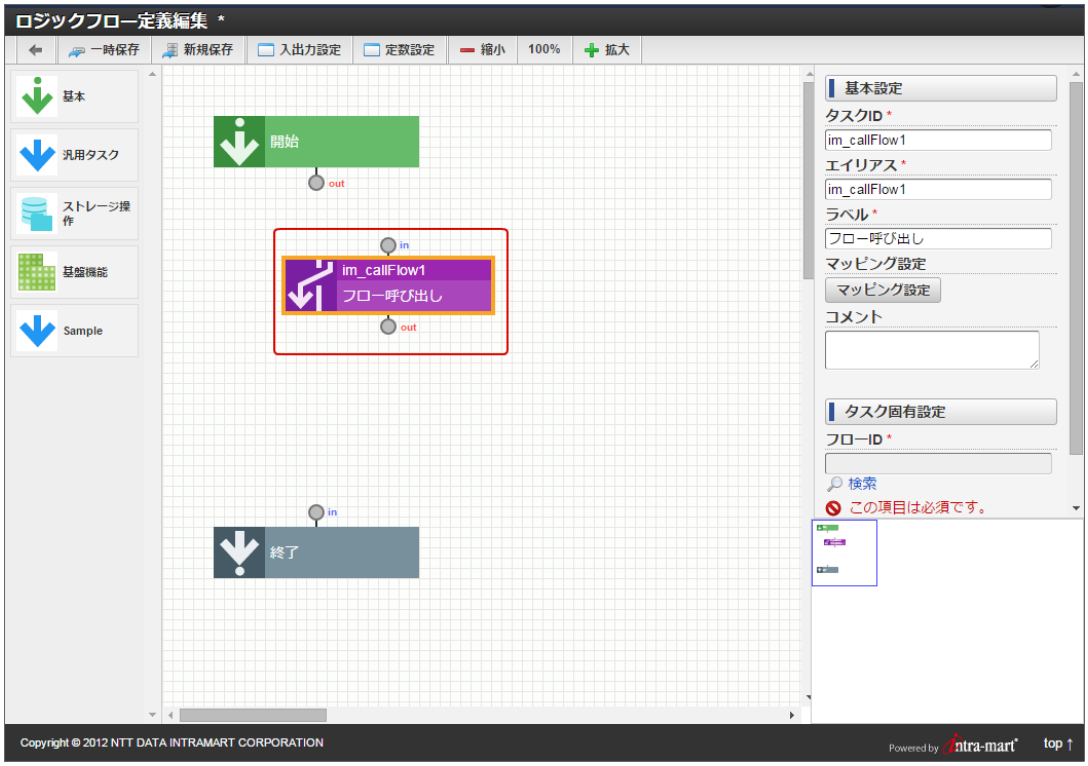



図：パレット内の「基本」

3. 「基本」にカテゴライズされる制御要素一覧がスライドインします。
   スライドインした一覧から、「フロー呼び出し」をクリックします。


図：「フロー呼び出し」制御要素をクリック

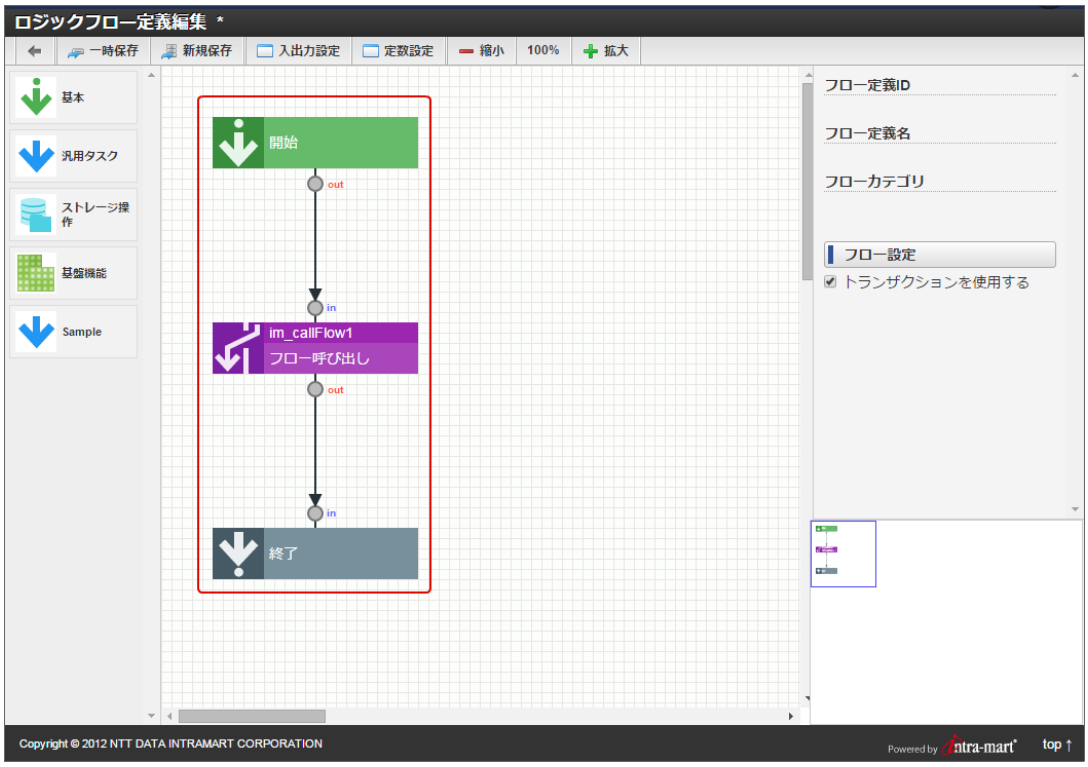
4. フロー呼び出しを行う制御要素がフロー編集画面上に追加されます。




図：「フロー呼び出し」制御要素の追加

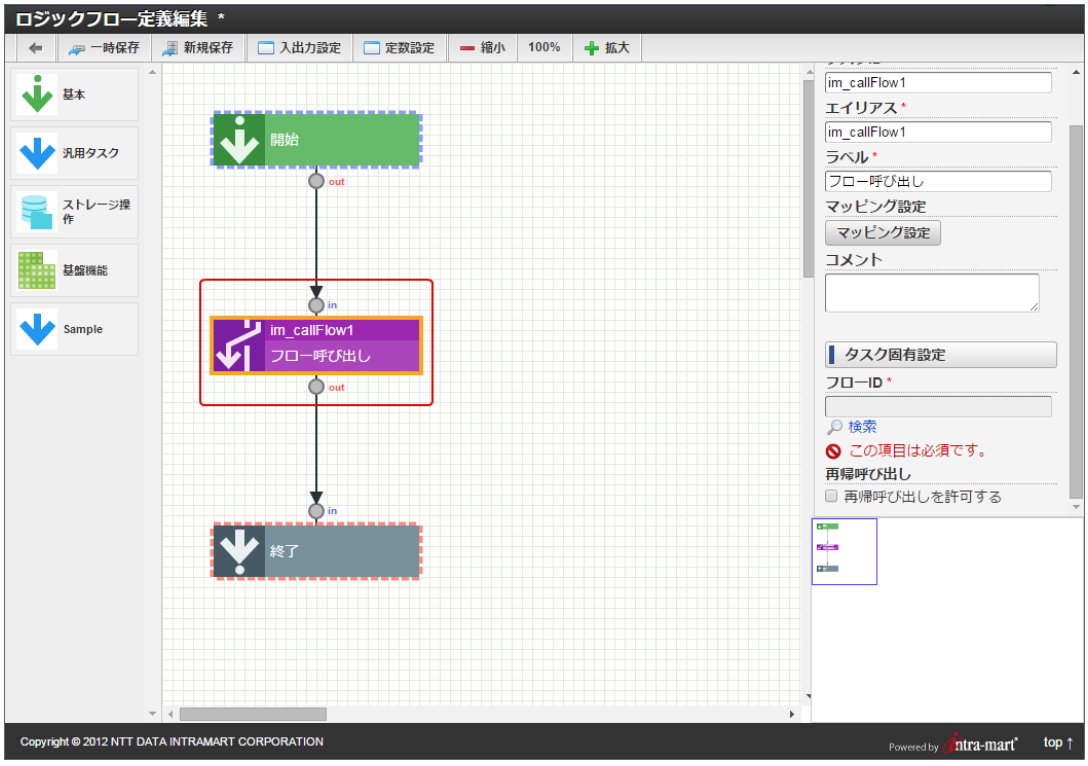
5. フロー呼び出しを行う制御要素を、以下の具体的な線（シーケンス）の定義内容に従って接続します。

| 入力（始点） | 出力（終点） |
|---|---|
| 開始 - out | フロー呼び出し - in |
| フロー呼び出し - out | 終了 - in |



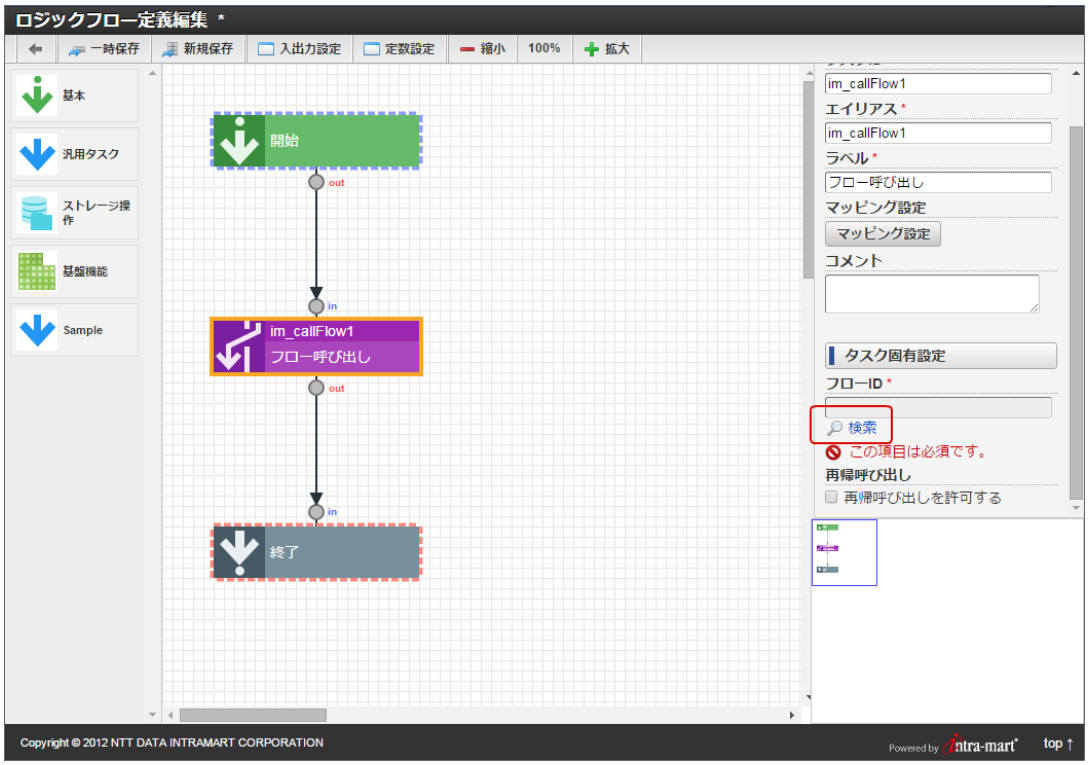

図：シーケンスの定義

6. フロー編集画面上の「フロー呼び出し」タスクをクリックします。




図：「フロー呼び出し」タスクをクリック

7. タスク固有設定の「フローID」項目下にある「検索」をクリックし、呼び出しを行うフローの検索ポップアップを表示します。

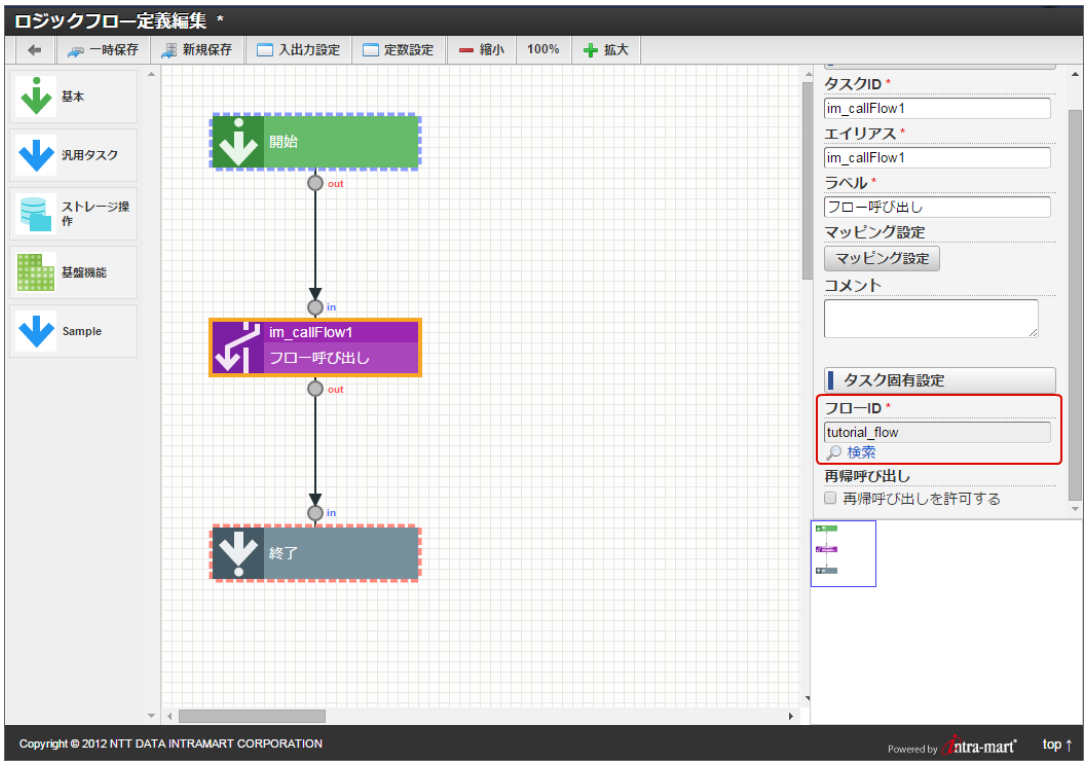



図：検索ポップアップの表示

8. ロジックフロー定義検索ポップアップから、フロー定義ID「tutorial_flow」の行の選択チェックボックスにチェックを入れ、「決定」をクリックします。


図：「tutorial_flow」行のチェック

9. タスク固有設定の「フローID」項目に、「tutorial_flow」が表示されていることを確認してください。

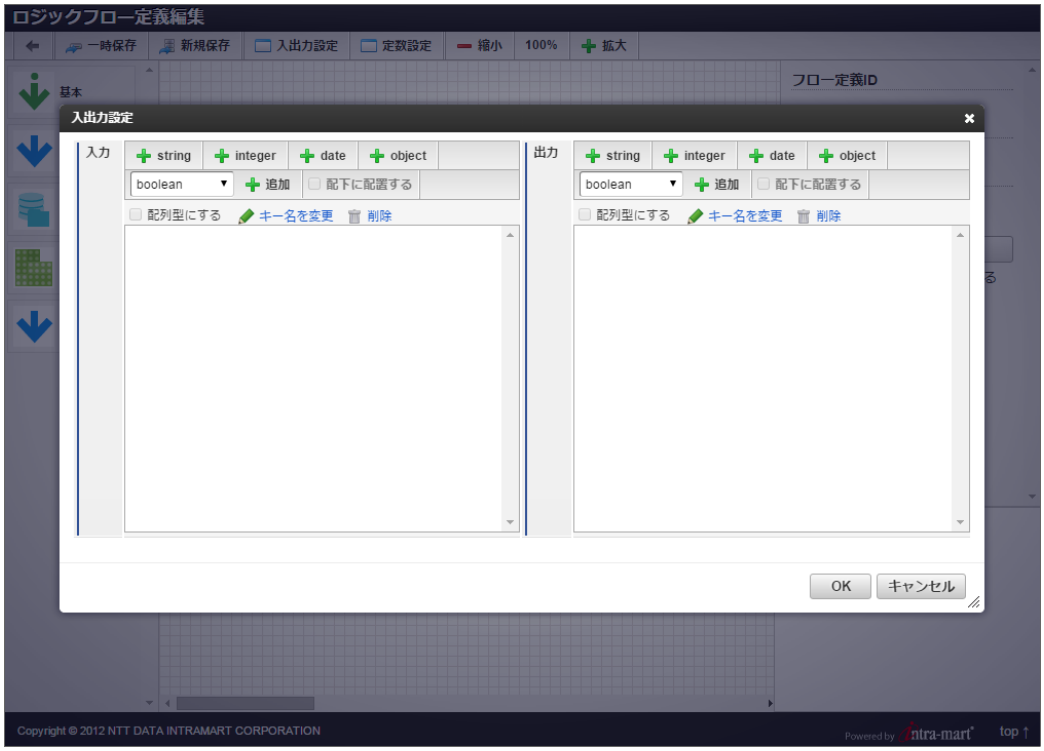



図：呼び出すフローIDの設定内容を確認

以上で、フロー呼び出しを含むロジックフローの定義が完了しました。

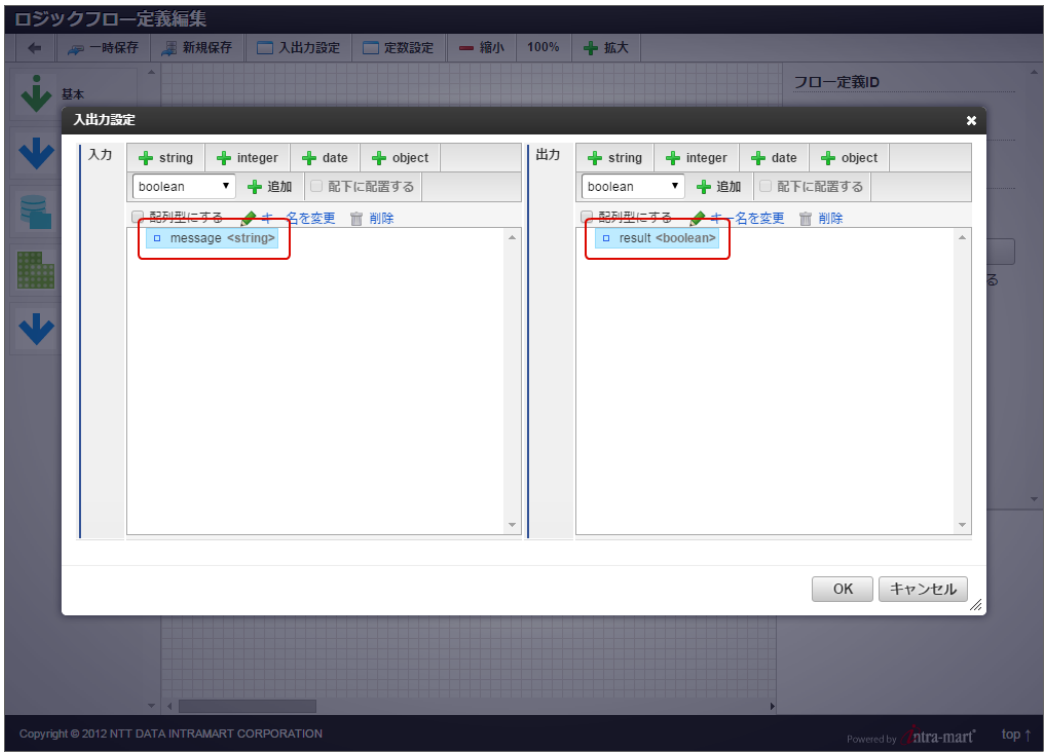
続いて、新しく作成したロジックフローの入出力値設定、および、マッピング設定を行います。
「*基礎編 - ファースト・ステップ*」-「*入出力設定を定義する*」で定義したとおり、呼び出し先のロジックフローには必要とする入力値、および、その結果の出力値が定義されています。
「フロー呼び出し」制御要素は、呼び出し先のロジックフローの入力値と出力値から、動的に自身の入力値・出力値を決定します。

1. ロジックフロー定義編集画面上部、ヘッダ内の「入出力設定」をクリックし、入出力設定画面を開きます。



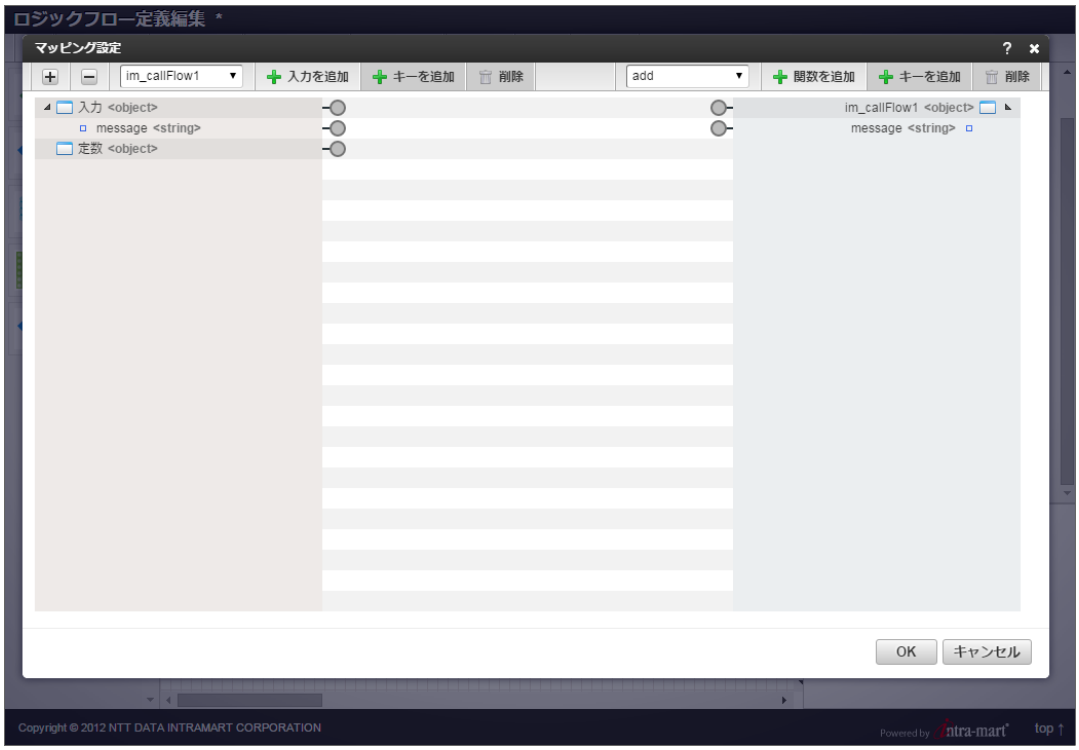

図：入出力設定画面の表示

2. 「*基礎編 - ファースト・ステップ*」-「*入出力設定を定義する*」と同様の入力値・出力値を設定します。




図：基礎編と同様の入出力値の設定

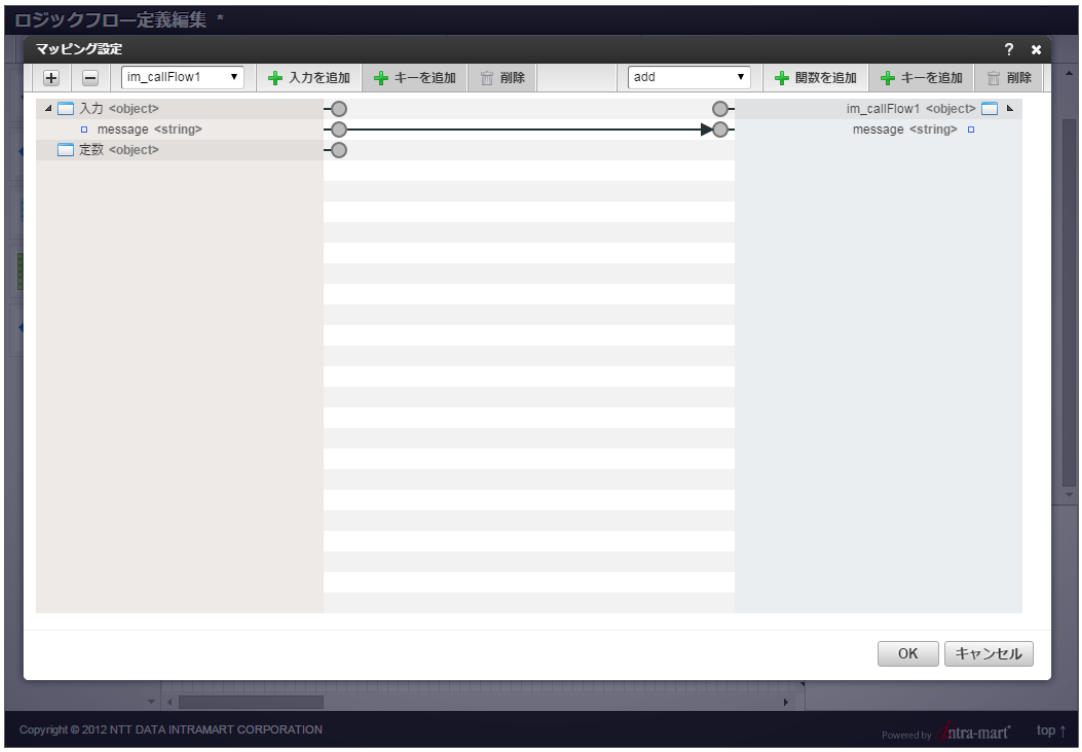
3. フロー編集画面上の「フロー呼び出し」制御要素をクリックした上で、「マッピング設定」をクリックし、マッピング設定画面を開きます。




図：「フロー呼び出し」制御要素のマッピング設定画面

4. マッピング設定を以下のとおりに設定します。

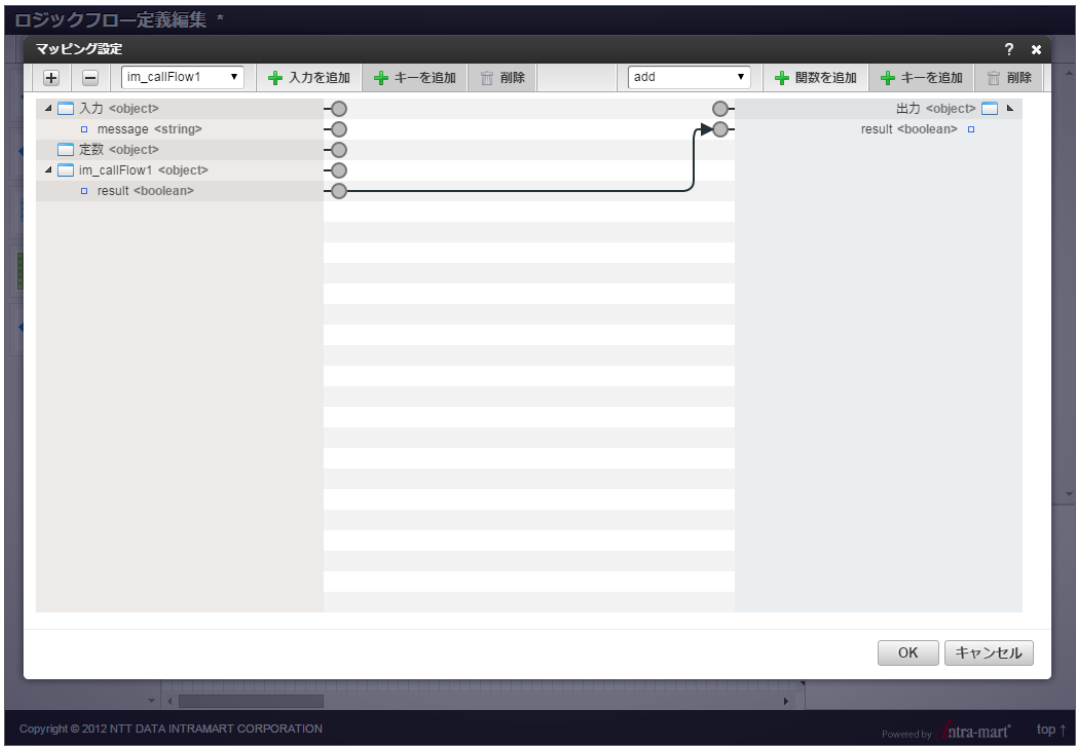
| 入力（始点） | 出力（終点） |
| --- | --- |
| 入力<object> - message<string> | im_callFlow1<object> - message<string> |




図：「フロー呼び出し」制御要素のマッピング設定

5. 設定画面右下のOKをクリックし、「フロー呼び出し」制御要素のマッピング設定を終了します。
6. 同様の手順で、「終了」制御要素のマッピング設定画面を開き、以下のとおりに設定します。

| 入力（始点） | 出力（終点） |
|---|---|
| im_callFlow1<object> - result<boolean> | 出力<object> - result<boolean> |

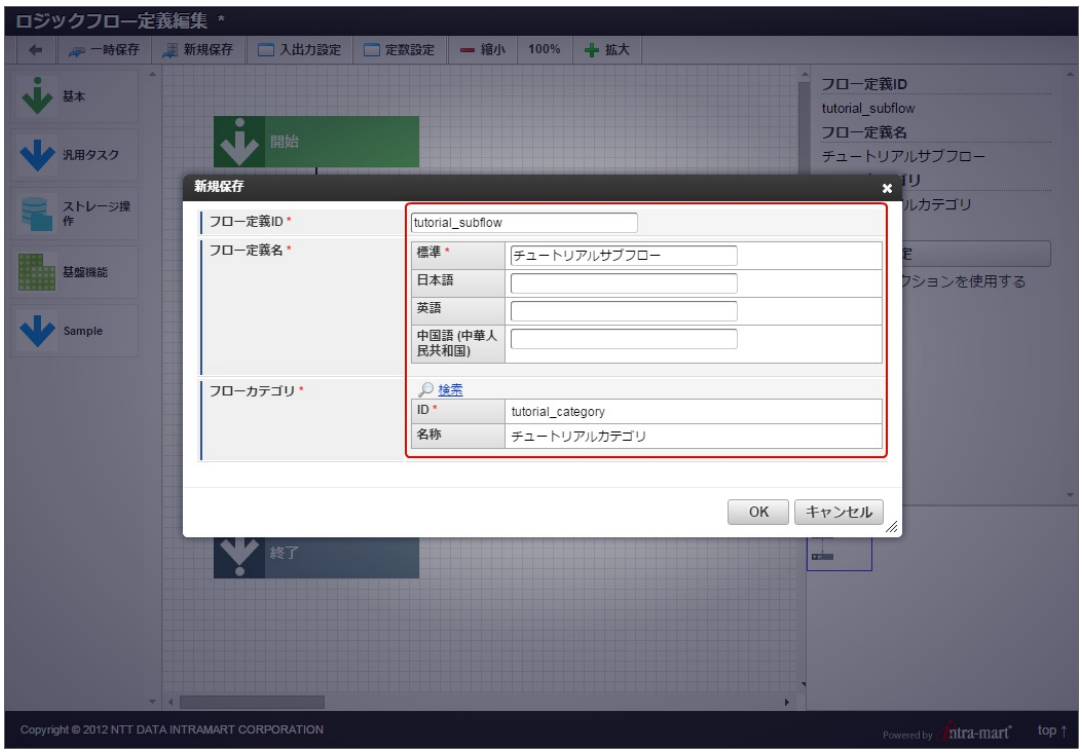



図：「終了」制御要素のマッピング設定

> **コラム**
>
> 入力としてフロー呼び出しの出力（im_callFlow1<object>）を利用する方法
>
> フロー呼び出しの出力値を、「終了」制御要素の入力値として利用する方法の詳細は「*応用編 - より高度なフロー*」 - 「*様々な入力情報の利用*」 - 「*他のエレメントの出力値の利用*」を参照してください。

7. 設定画面右下のOKをクリックし、「終了」制御要素のマッピング設定を終了します。

8. ロジックフローを以下のとおりに保存します。
   - フロー定義ID「tutorial_subflow」
   - フロー定義名
     - 標準 - 「チュートリアルサブフロー」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし
   - フローカテゴリ
     - ID - 「tutorial_category」
     - 名称 - 「チュートリアルカテゴリ」




図：サブフロー呼び出しの保存

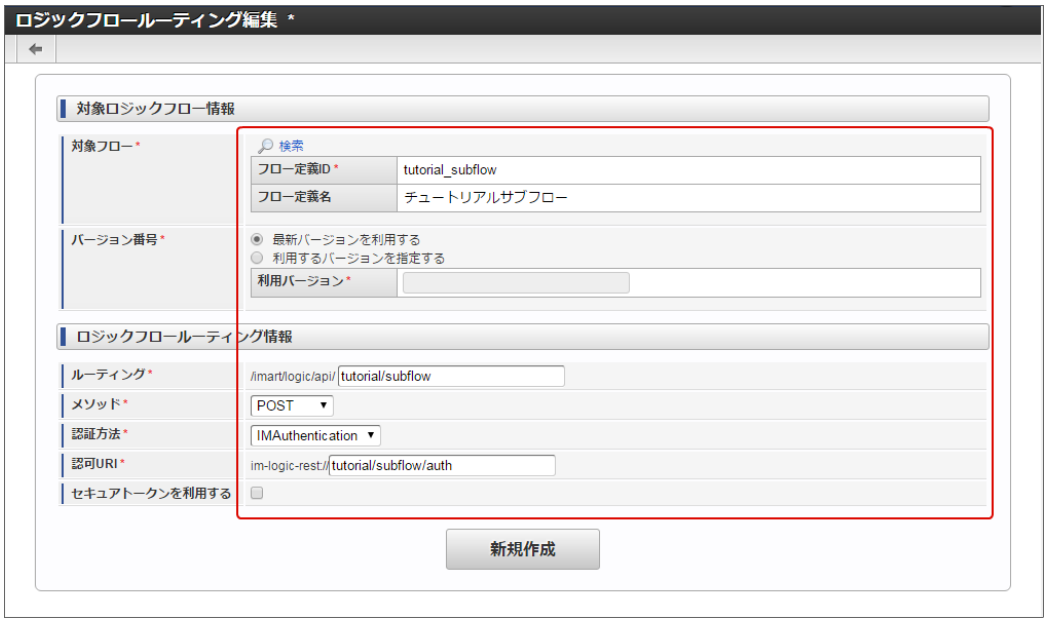
以上で、フロー呼び出しを利用したフローの作成が完了しました。

## サブフロー呼び出し結果の確認

次に、作成したフローを実行し、その結果を確認します。
フローの実行、および、結果の確認には「*基礎編 - ファースト・ステップ*」-「*フロールーティングを設定する*」と同様に、作成したフローのフロールーティングを定義し、Swaggerを利用して行います。

1. 「サイトマップ」→「LogicDesigner」→「ルーティング定義一覧」をクリックし、「ロジックフロールーティング定義一覧」画面を開きます。
   「ロジックフロールーティング定義一覧」画面左上の「新規作成」をクリックし、「ロジックフロールーティング編集」画面を表示します。

2. ロジックフロールーティング情報の各項目に以下の値を入力します。
   - 対象フロー
     - フロー定義ID - 「tutorial_subflow」
   - ルーティング「tutorial/subflow」
   - メソッド「POST」
   - 認証方法「IMAuthentication」
   - 認可URI「tutorial/subflow/auth」

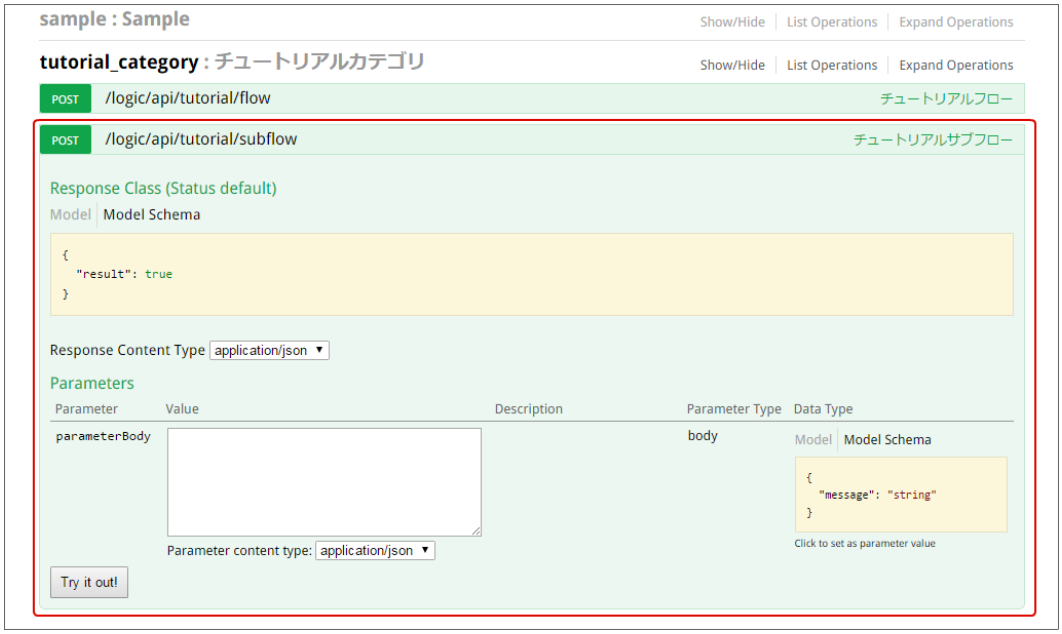



図：サブフロー呼び出しを行うロジックフロールーティング設定

3. 「新規作成」をクリックし、フロールーティング定義を作成します。

4. 作成したフロールーティング定義の認可を設定します。
   認可の設定内容として、「*基礎編 - ファースト・ステップ*」-「*ルーティングの認可を設定する*」と同様に、「認証済みユーザ」に対して「許可」を設定します。

5. 作成したフロールーティング「tutorial/subflow」の行のSPECアイコンをクリックし、Swaggerの画面を表示します。




図：Swagger画面の表示

6. 「*基礎編 - ファースト・ステップ*」-「*Swagger(SPEC)から実行する*」と同様の手順で、フローを実行します。

実行結果として「*基礎編 - ファースト・ステップ*」で作成したロジックフローで得られた結果と、同じ出力が得られることを確認してください。
これは、正しくフローの呼び出しが行われ、基礎編で作成したロジックフローが実行されたことを表します。

以上で、サブフロー呼び出し結果の確認が完了したました。

コラム

呼び出されるフロー（サブフロー）の制約

「フロー呼び出し」制御要素によって呼び出されるフローは以下の制約のもと呼びだされます。

- 呼び出されるフローのバージョンは、常に最新のバージョンで呼び出しが行われます。
- 「フロー呼び出し」制御要素を含むフローを保存後、呼び出し先のフローの最新バージョンが変更され、入出力情報が変更されていた場合、呼び出し元の入出力設定もその影響を受けます。

## 変数を利用したフロー

この章では、ロジックフローの定義で変数を利用する方法を説明します。



### 変数とは

一般的なプログラミングにおける変数とは、プログラムのソース上で扱われるデータの記憶領域として固有の名称を与えたものです。
IM-LogicDesignerにおける変数もそうした一般的なプログラミングにおける変数と同様に、ロジックフローの中で扱われるデータの記憶領域として固有の名称を与えたものです。

この章では、変数の定義方法と利用方法を説明します。

本章で作成するフロー

本章では変数の作成、および、その利用方法を確認するため、以下のようなフローを作成します。

- 変数として、ロジックフロー実行ユーザ名を格納する変数を定義します。
- ロジックフローの中で、ユーザコンテキストからフロー実行ユーザのユーザ名を取得し、変数に代入します。
- 変数に代入されたロジックフロー実行ユーザ名を「ログ出力」タスクを利用して、コンソールに出力します。

なお、本章では「*基礎編 - ファースト・ステップ*」で作成したロジックフローを利用していないことに注意してください。

### 変数設定画面を開く

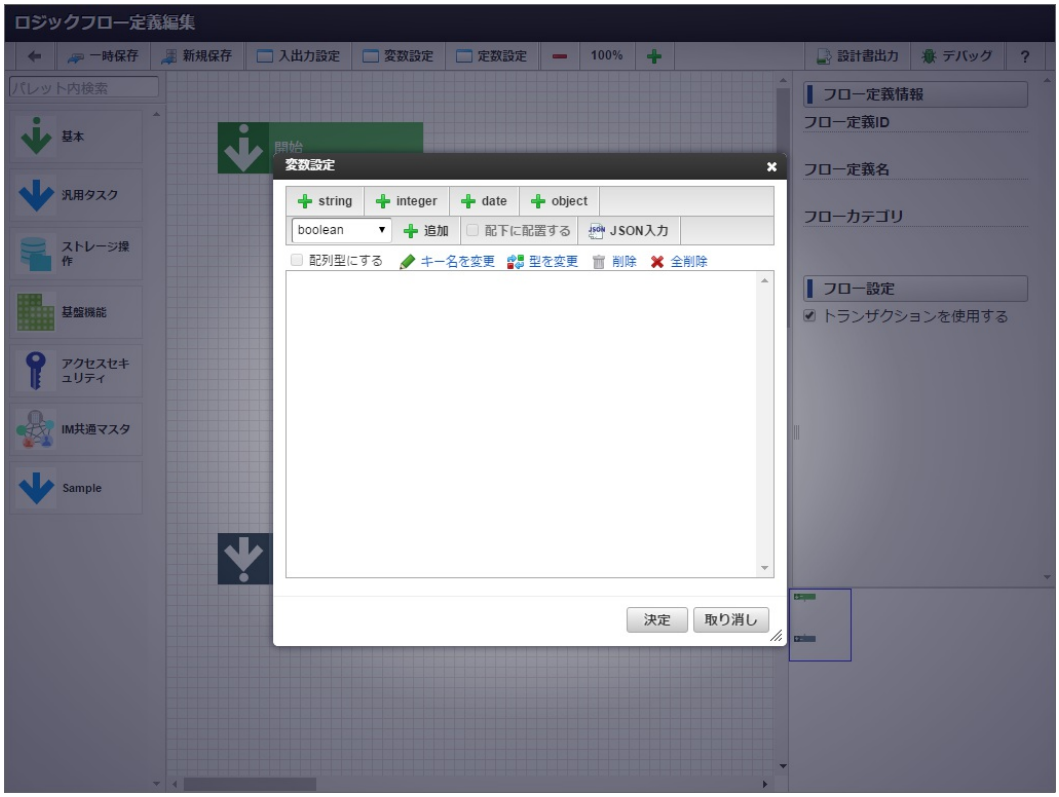
変数を利用するためには、予め変数を定義する必要があります。
変数の定義は変数設定画面から行います。

変数設定画面はロジックフロー定義編集画面から開きます。

1. 「サイトマップ」→「LogicDesigner」→「フロー定義一覧」から、ロジックフロー定義一覧を開きます。
   ロジックフロー定義一覧画面左上の「ロジックフロー新規作成」をクリックします。
2. ロジックフロー定義編集画面上部、ヘッダ内の「変数設定」をクリックします。


図：「変数設定」をクリック

3. 変数設定画面が表示されます。

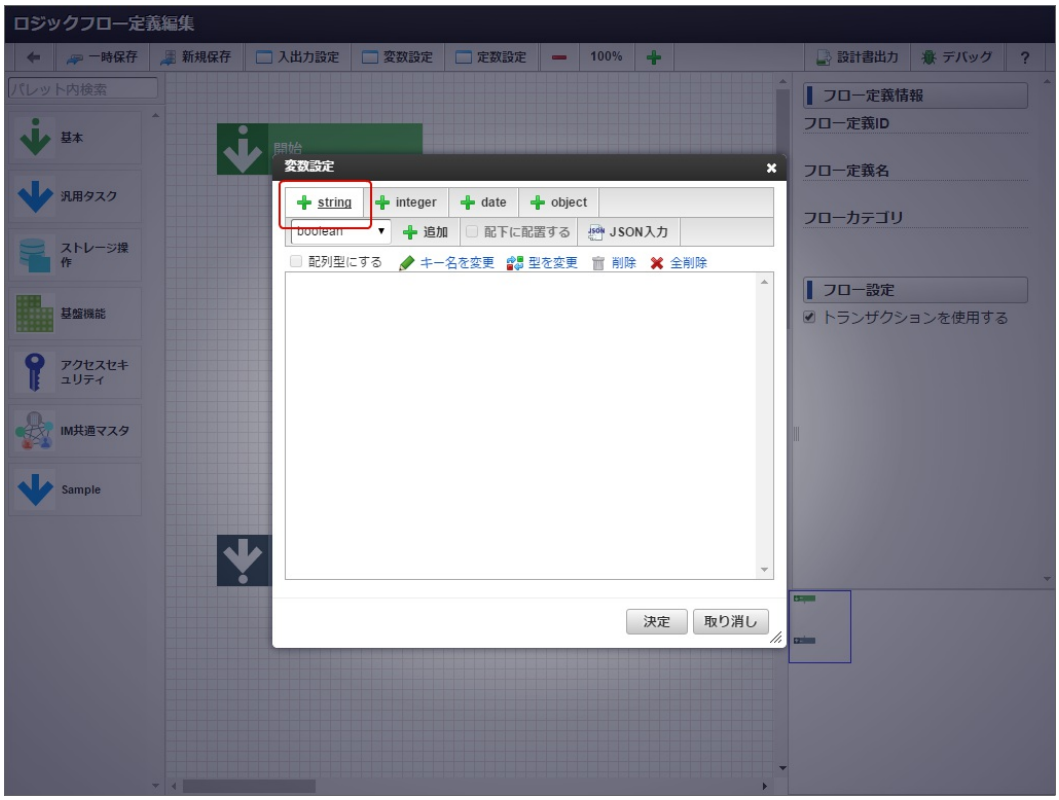



図：変数設定画面

## 変数を定義する

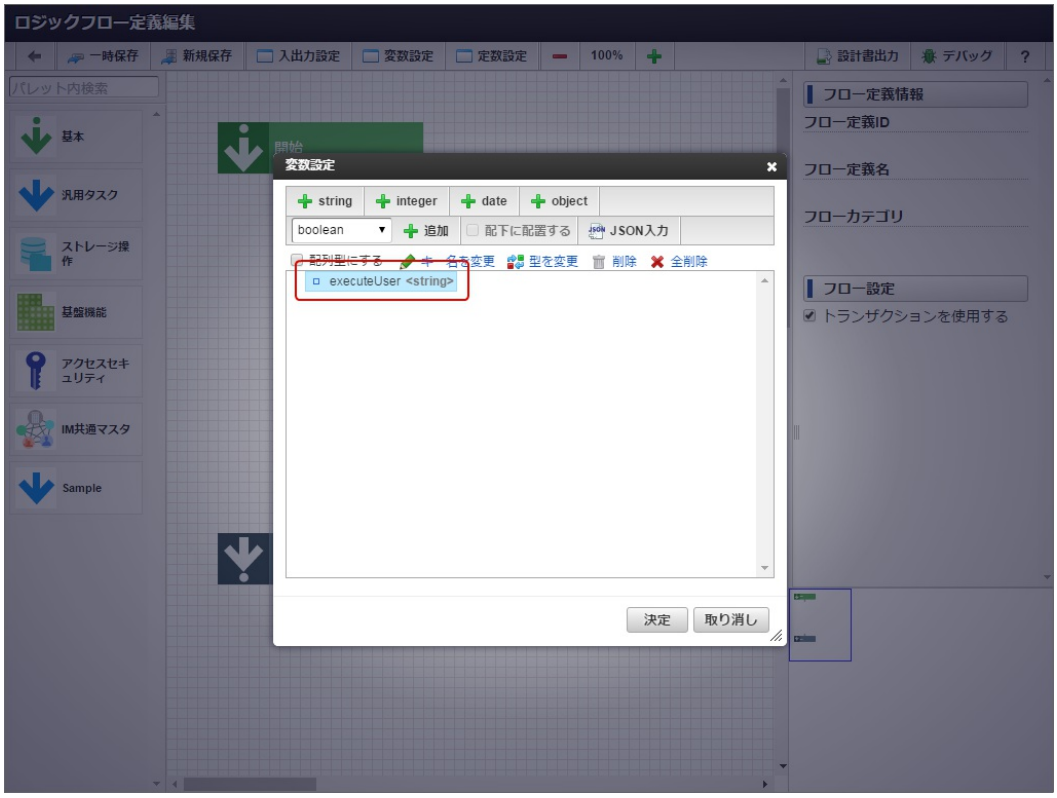
次に、新しく変数を定義します。
変数の定義は「*入出力設定を定義する*」で説明を行った入出力値の定義と同様の方法で利用可能です。

1. ヘッダ内の「変数設定」をクリックし、変数設定画面を表示します。
2. 変数設定画面上部にある「**+string**」をクリックします。




図：変数の追加

3. 画面中央に新しく値が設定されたのを確認し、名称を「executeUser」とします。




図：変数の名称変更

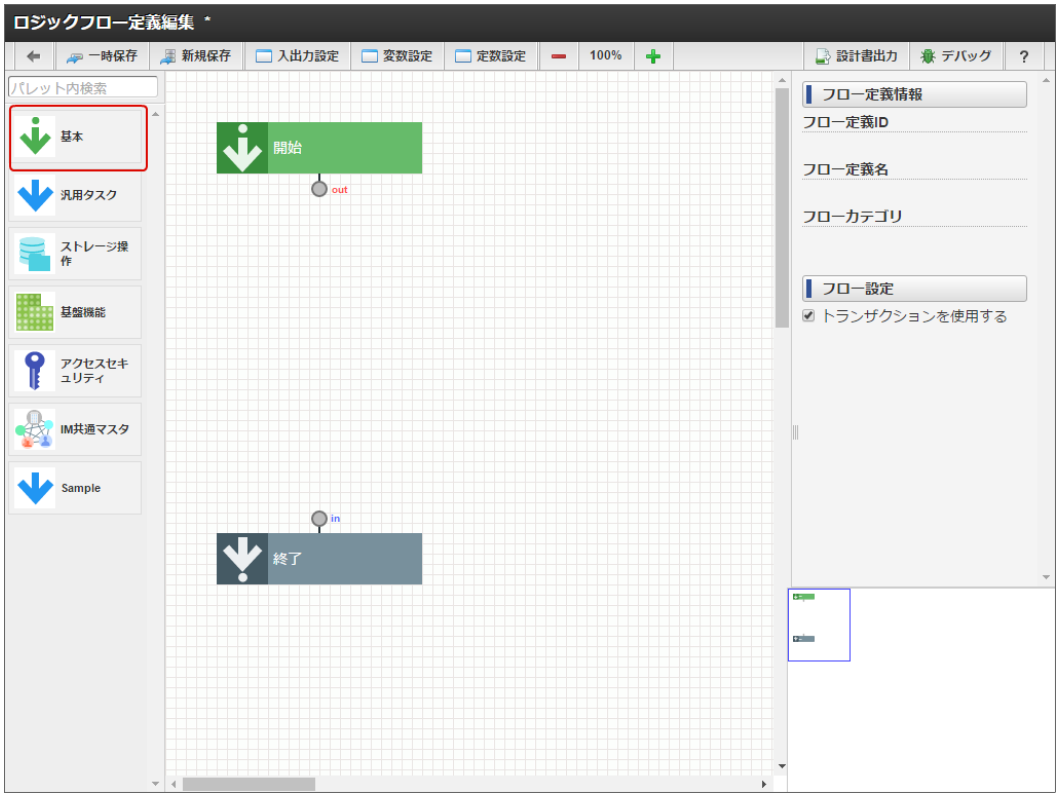
4. 変数設定画面下部の「決定」をクリックします。

以上で、変数の定義が完了しました。

## 変数へ値を代入する

次に、定義した変数へ実際に値を代入します。
IM-LogicDesignerでは、変数へ値を代入するための「変数操作」制御要素を提供しています。

1. ロジックフロー定義編集画面左部、パレット内の「基本」へカーソルを合わせます。

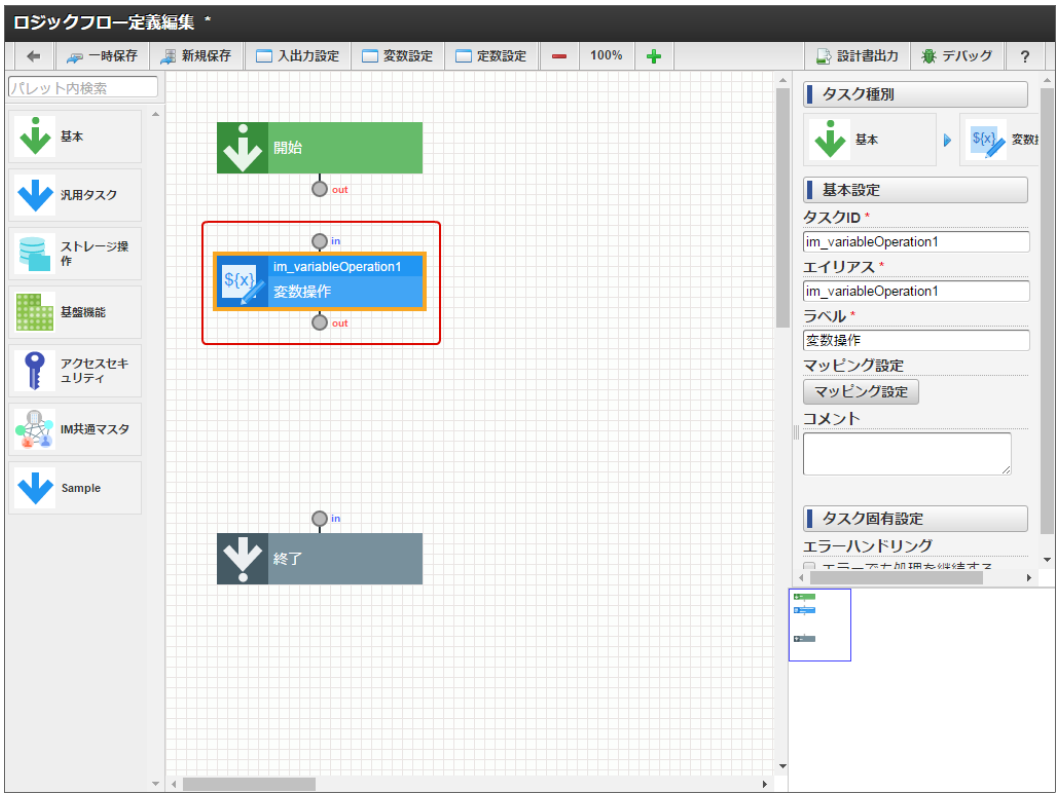



図：パレット内の「基本」

2. 「基本」にカテゴライズされる制御要素一覧がスライドインします。
スライドインした一覧から、「変数操作」をクリックします。

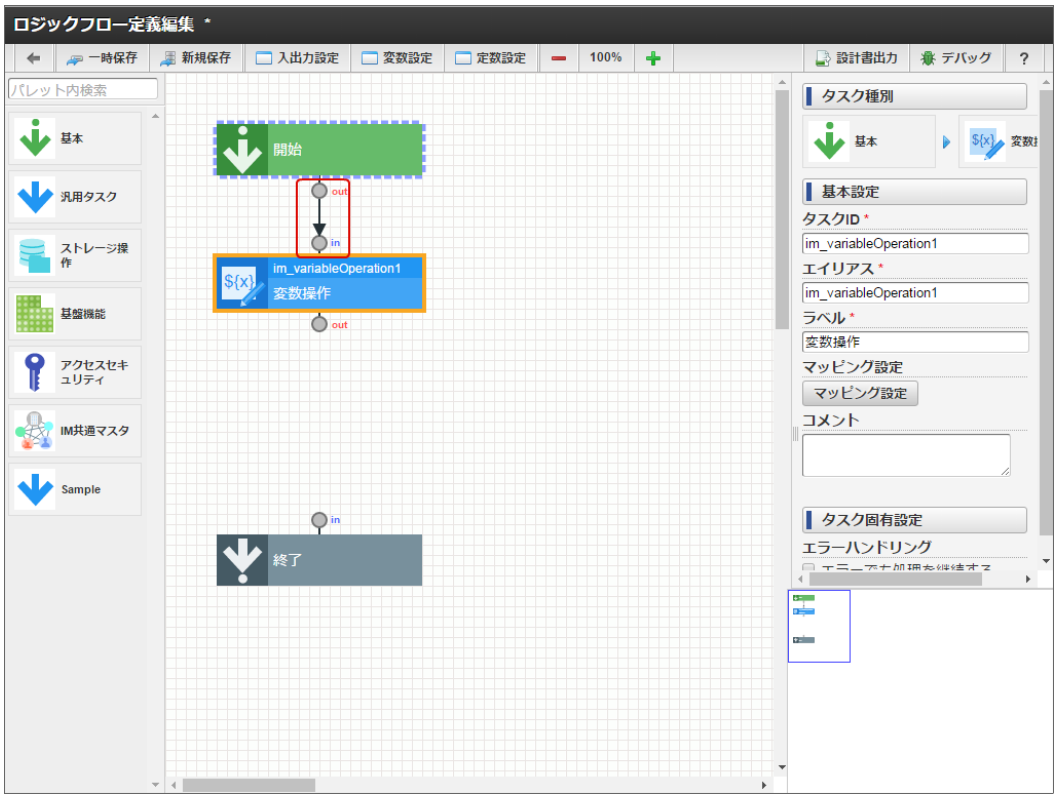

図：「変数操作」制御要素をクリック

3. 変数操作を行う制御要素がフロー編集画面上に追加されます。




図：「変数操作」制御要素の追加

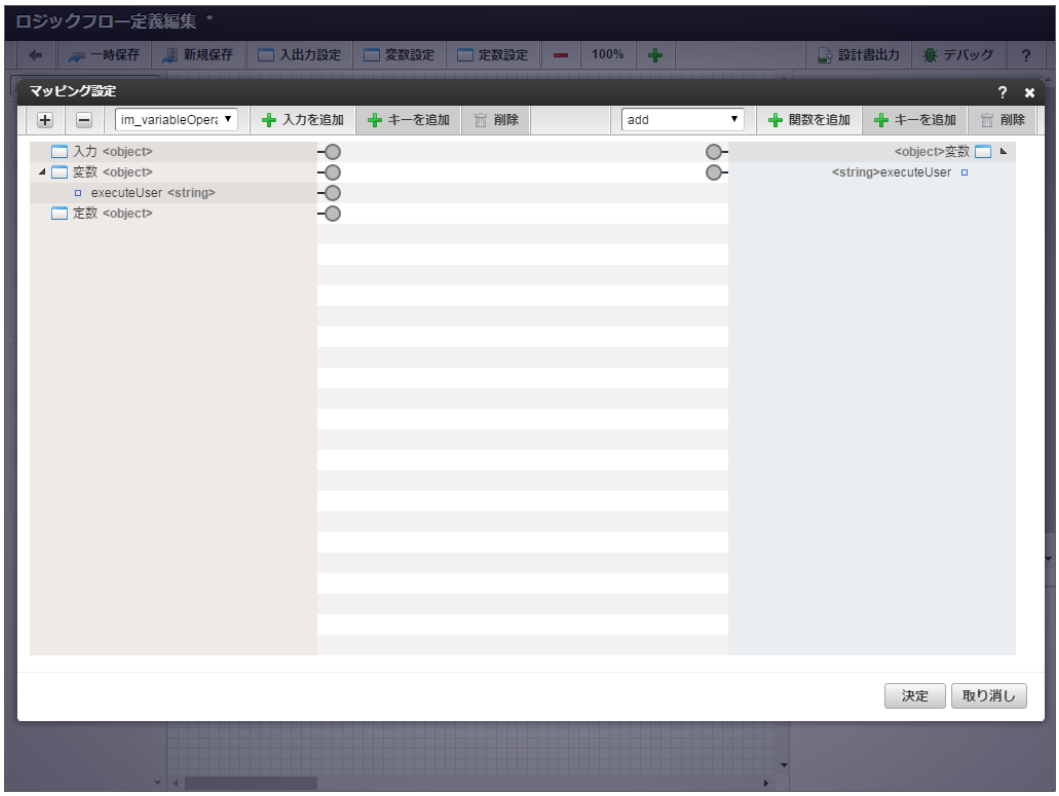
4. 「開始」制御要素と、「変数操作」制御要素を接続します。
   （正しく動作させるためのフローの接続は、次章で行います）




図：シーケンスの定義（変数操作）

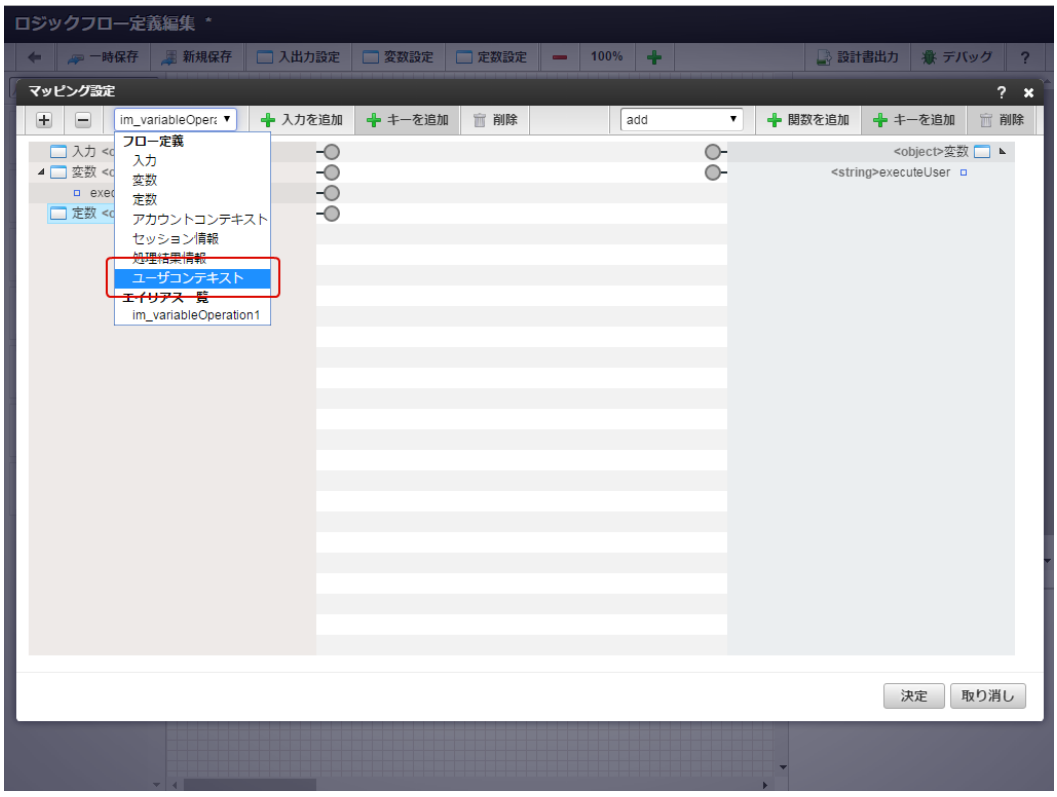
具体的な変数の操作は、「変数操作」制御要素のマッピング設定を用いて行います。

1. 追加した「変数操作」制御要素をクリックした上で、「マッピング設定」をクリックし、マッピング設定画面を開きます。



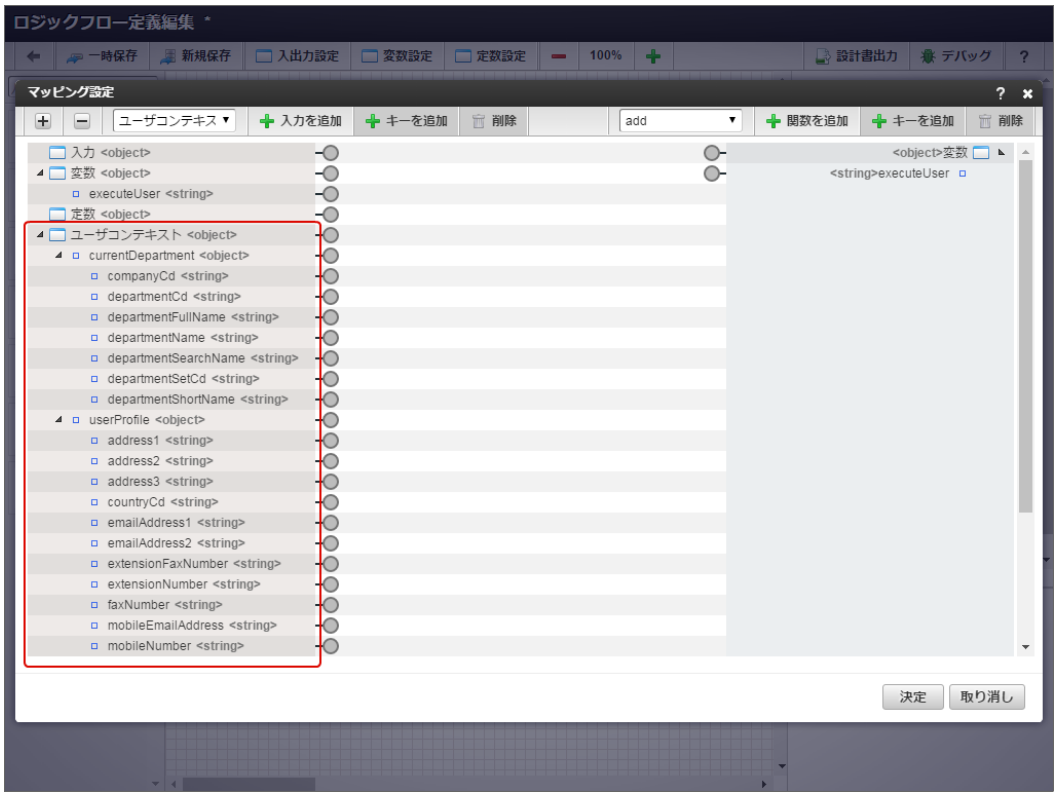

図：「変数操作」制御要素のマッピング設定

2. マッピング設定画面上部、ヘッダ内の左上に位置するセレクトボックスをクリックし、以下の項目を選択します。
   - エイリアス一覧 - ユーザコンテキスト




図：入力 - ユーザコンテキストの追加

3. セレクトボックスの中身が変更されたことを確認し、右側にある「入力を追加」をクリックし、ユーザコンテキストを新しく入力として追加します。

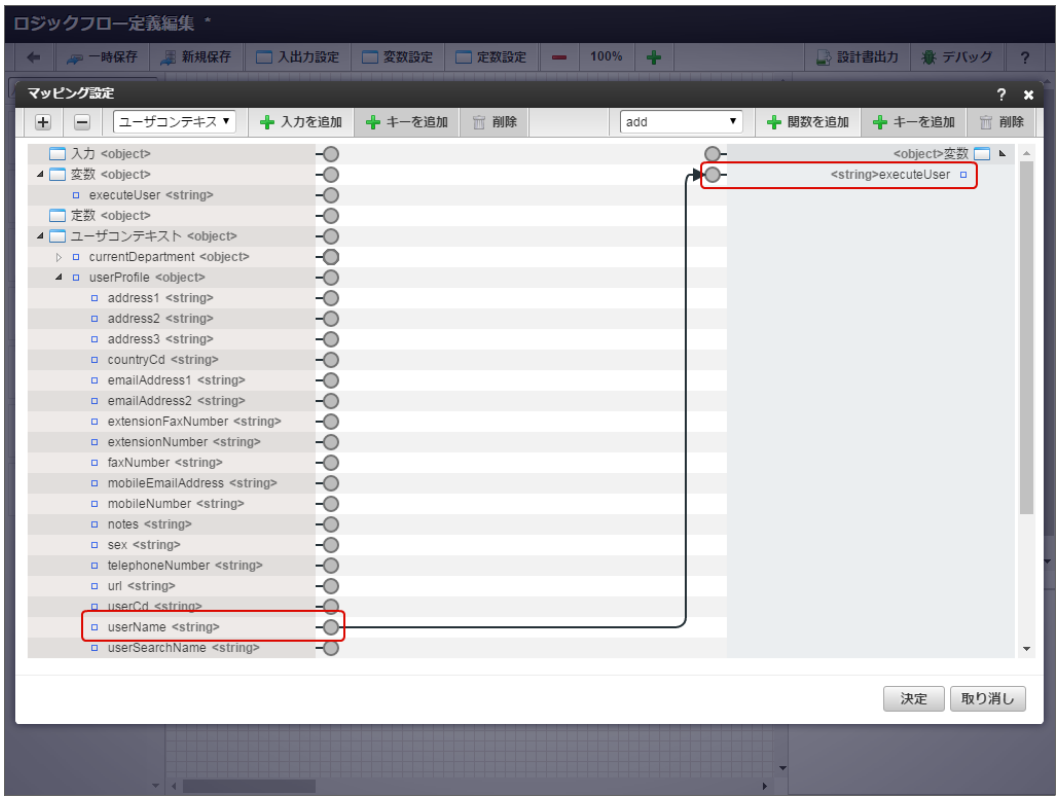



図：入力：ユーザコンテキストの追加

> **コラム**
>
> 入力としてユーザコンテキストを利用する
>
> 「変数操作」制御要素の入力値として利用しているユーザコンテキストの詳細は「*応用編 - より高度なフロー*」 - 「*様々な入力情報の利用*」 - 「*暗黙的な変数の利用*」を参照してください。

4. 作成した変数（executeUser）へ、ユーザコンテキストのユーザ名を代入するため、マッピング設定を以下のとおりに設定します。

| 入力（始点） | 出力（終点） |
|---|---|
| ユーザコンテキスト<object> - userProfile<object> - userName<string> | <object>変数 - <string>executeUser |




図：マッピング設定の追加

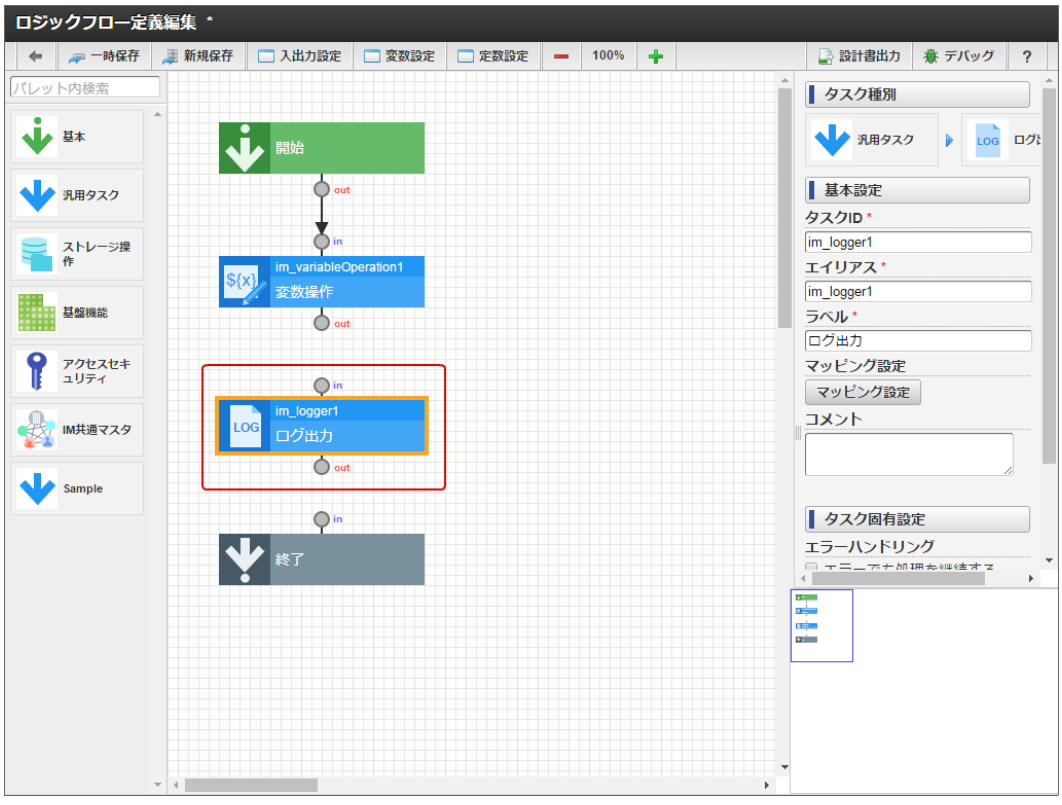
5. 設定画面右下のOKをクリックし、「変数操作」制御要素のマッピング設定を終了します。

以上で、変数への値の代入が完了しました。

## 変数を入力として利用する

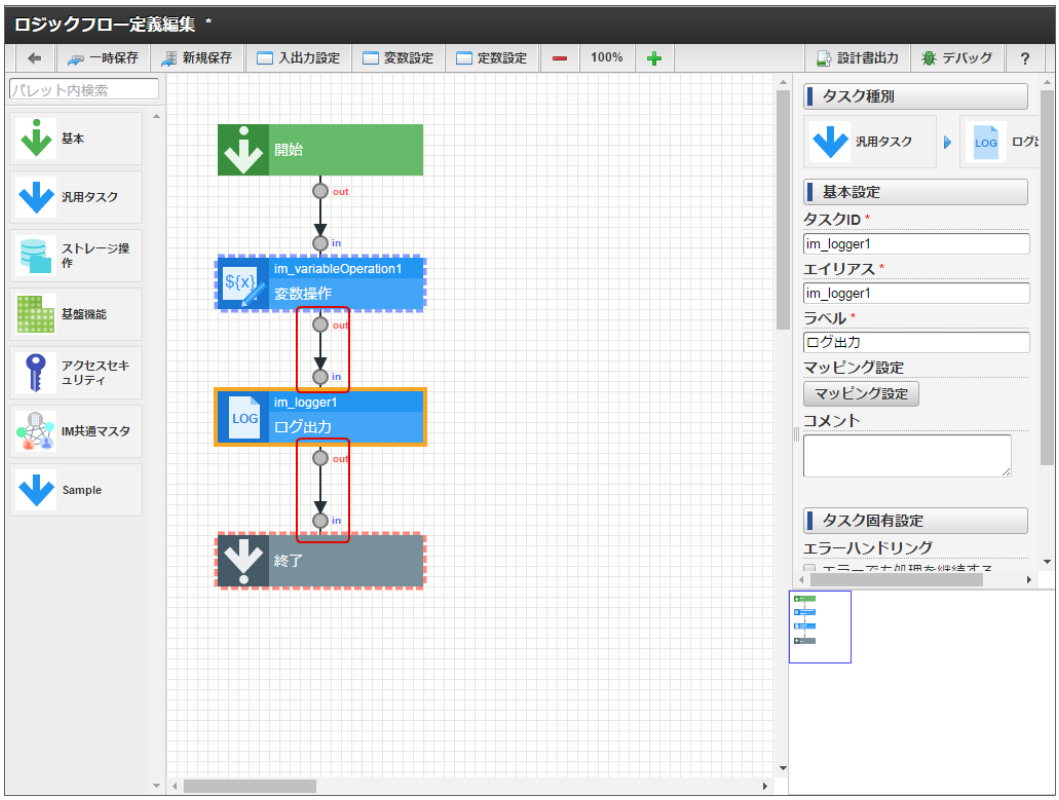
最後に、変数を実際に他のエレメントの入力として利用する方法を説明します。

1. ロジックフロー定義編集画面左部、パレット内の「汎用タスク」-「ログ出力」をクリックし、「ログ出力」タスクをフロー編集画面上に追加します。




図：「ログ出力」タスクの追加

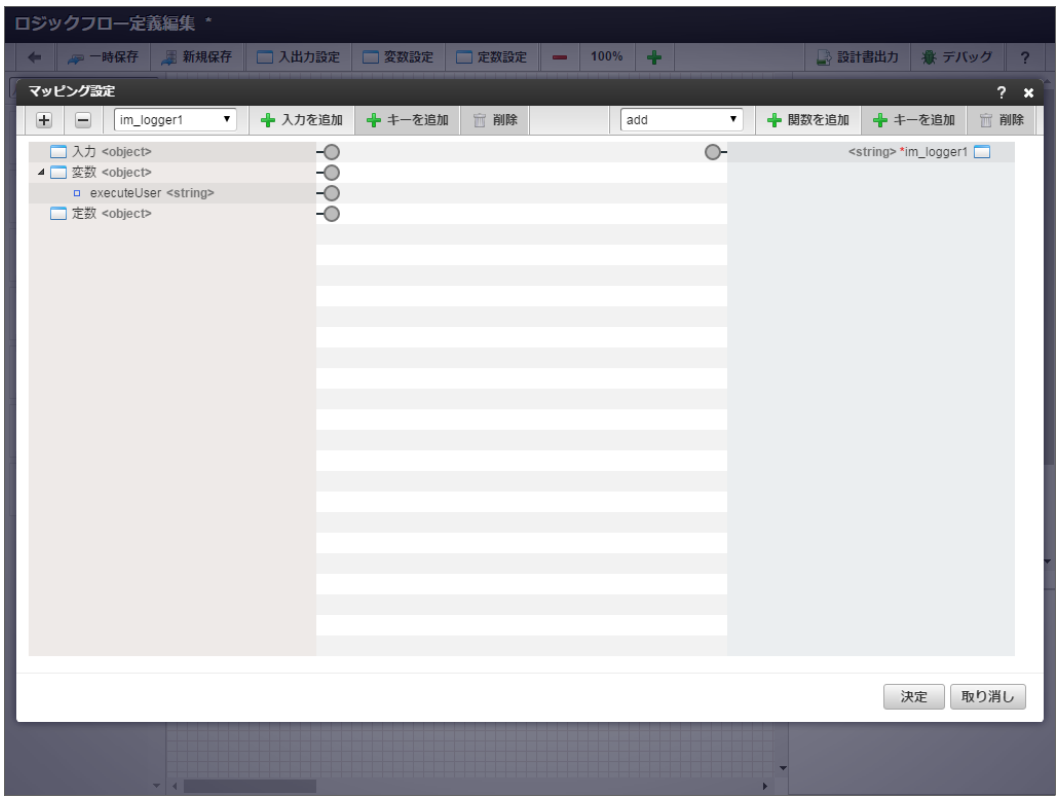
2. 以下のエレメント間に線を引きます。
   - 「変数操作」制御要素から「ログ出力」タスク
   - 「ログ出力」タスクから「終了」制御要素




図：シーケンス定義（完成）

3. 「ログ出力」タスクをクリックした上で、「マッピング設定」をクリックし、マッピング設定画面を開きます。

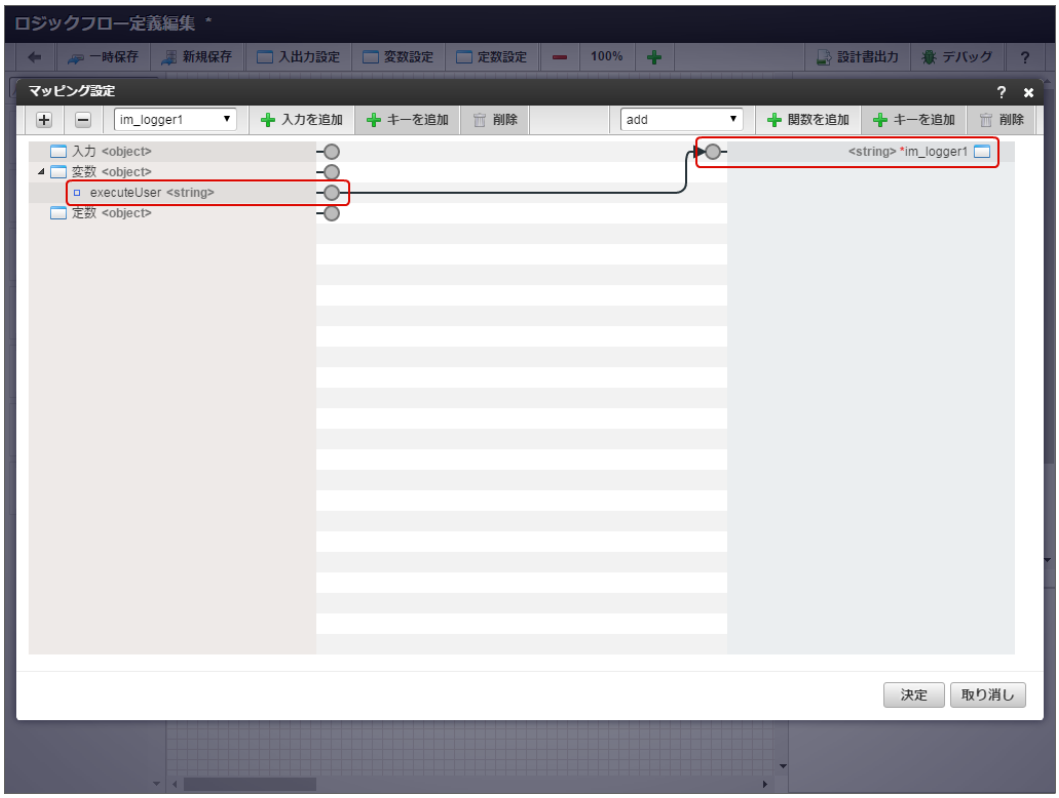
4. マッピング設定画面の入力としてデフォルトで「変数<object>」、および、その配下に「executeUser<string>」が定義されていることを確認してください。




図：「ログ出力」タスクのマッピング設定

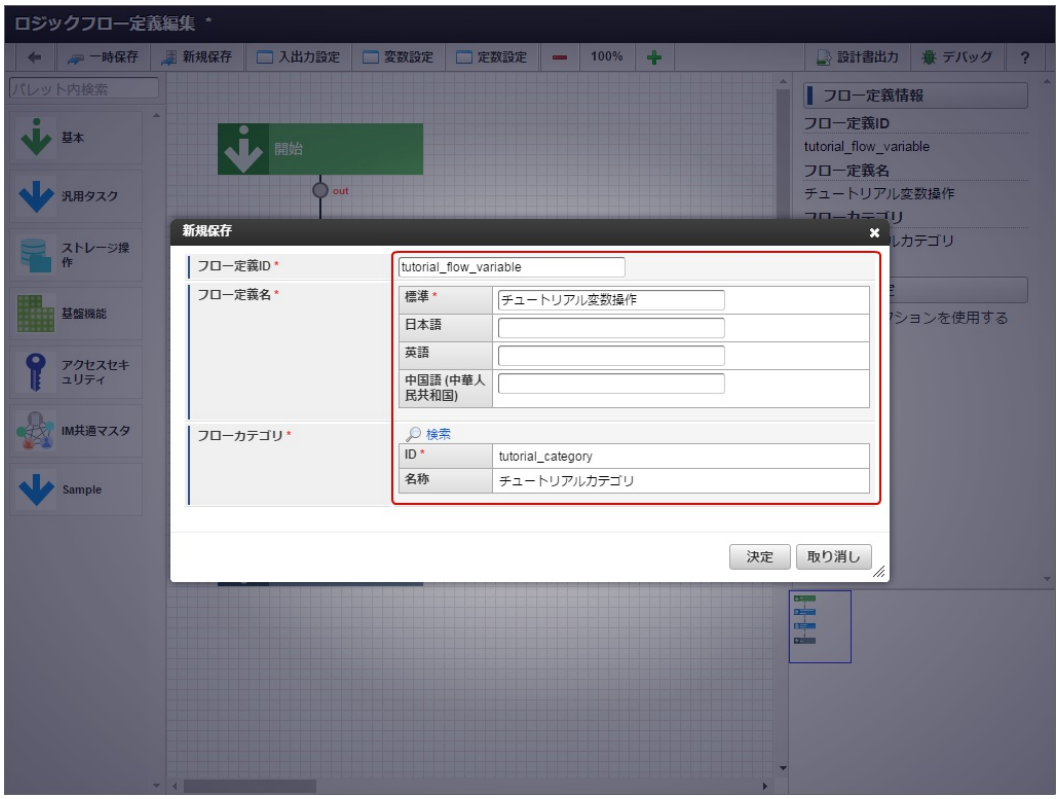
5. マッピング設定を以下のとおりに設定します。

| 入力（始点） | 出力（終点） |
| --- | --- |
| 変数<object> - executeUser<string> | <string>im_logger1 |




図：マッピング設定の追加

6. 設定画面右下のOKをクリックし、「ログ出力」タスクのマッピング設定を終了します。
7. ロジックフローを以下のとおりに保存します。
   - フロー定義ID「tutorial_flow_variable」
   - フロー定義名
     - 標準 - 「チュートリアル変数操作」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし
   - フローカテゴリ
     - ID - 「tutorial_category」
     - 名称 - 「チュートリアルカテゴリ」

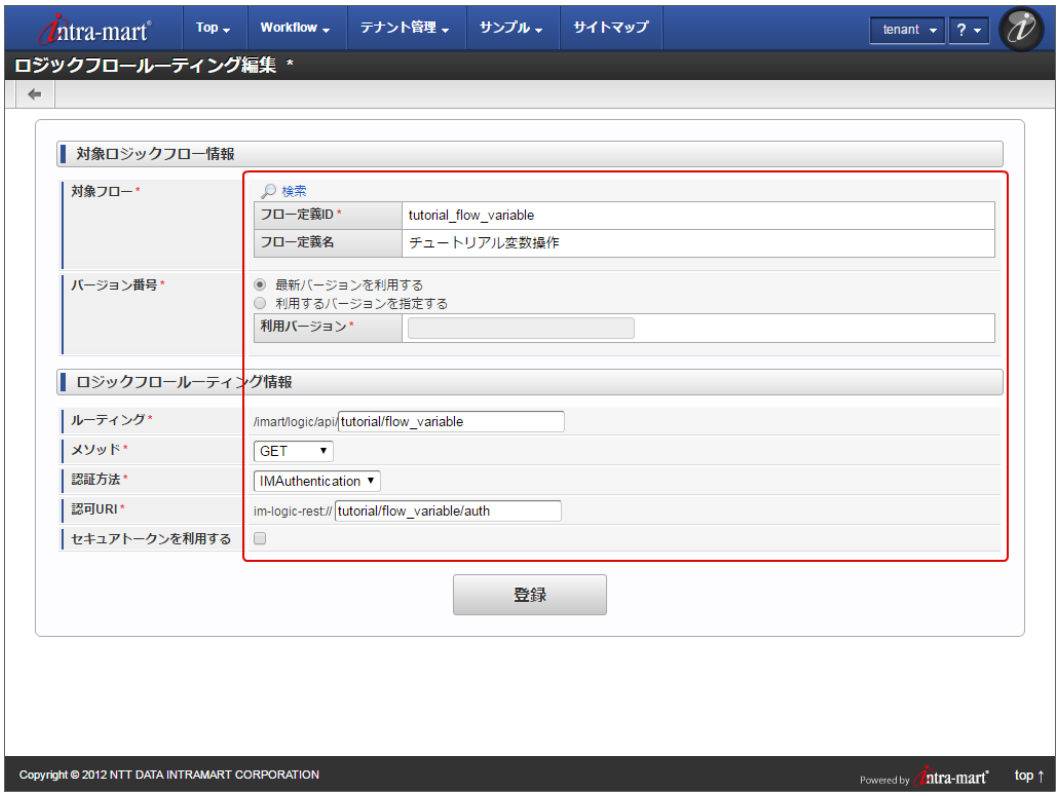



図：ロジックフローの保存

以上で、変数を入力として利用する準備が完了しました。

続いて、作成したフローを実際に実行し、変数が正しく動作することを確認します。
フローの実行、および、結果の確認には「*基礎編 - ファースト・ステップ*」-「*フロールーティングを設定する*」と同様に、作成したフローのフロールーティングを定義し、Swaggerを利用して行います。

1. ロジックフロールーティング情報の各項目に以下の値を入力します。
   - 対象フロー
     - フロー定義ID - 「tutorial_flow_variable」
   - ルーティング「tutorial/flow_variable」
   - メソッド「GET」
   - 認証方法「IMAuthentication」
   - 認可URI「tutorial/flow_variable/auth」

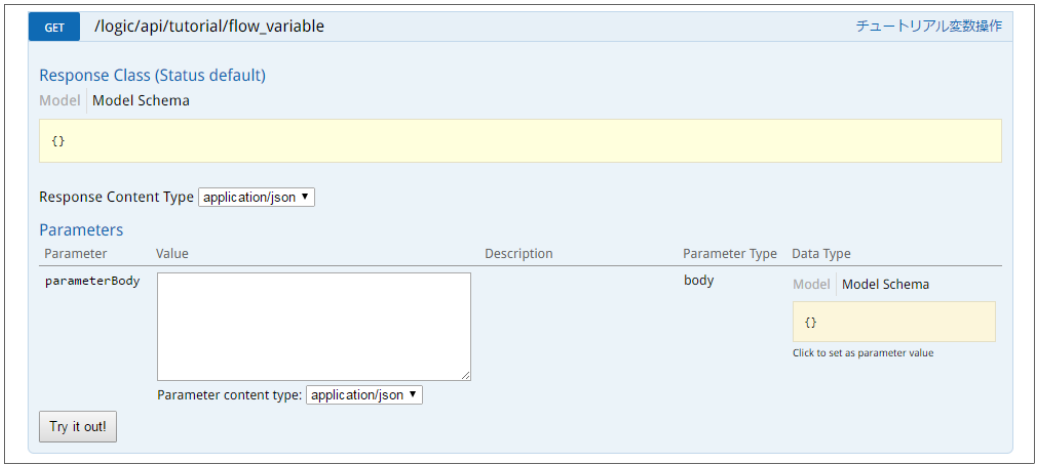



図：ロジックフロールーティングの作成

2. 「新規作成」をクリックし、フロールーティング定義を作成します。

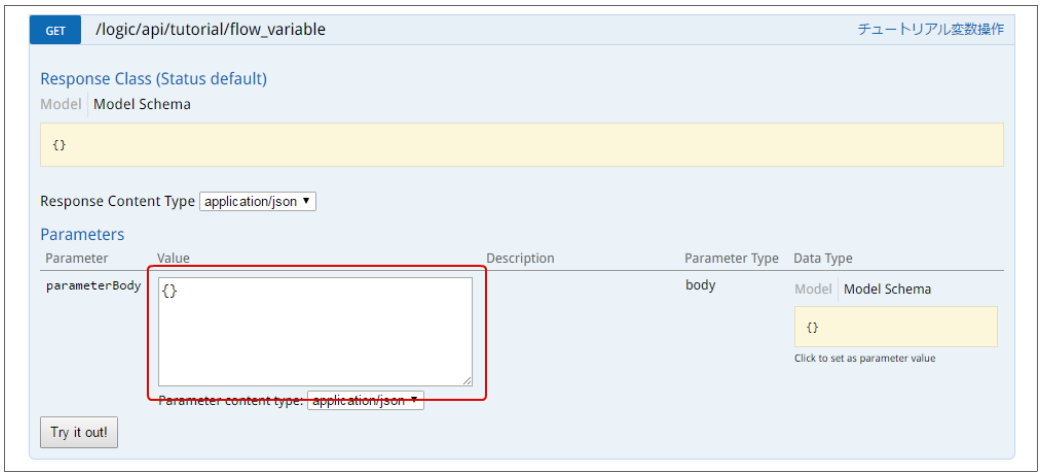
3. 作成したフロールーティング定義の認可を設定します。
   認可の設定内容として、「*基礎編 - ファースト・ステップ*」-「*ルーティングの認可を設定する*」と同様に、「認証済みユーザ」に対して「許可」を設定します。

4. 作成したフロールーティング「tutorial/flow_variable/auth」の行のSPECアイコンをクリックし、Swaggerの画面を表示します。




図：Swaggerの表示

5. 以下の内容でロジックフローを実行します。
   - parameterBody - 設定なし




図：パラメータ設定例

6. 実行結果として、Swaggerを介してロジックフローを実行したユーザ名（本チュートリアルではテナント管理者名）がコンソール上に出力されていることを確認してください。

得られた実行結果は「ユーザコンテキストから取得したユーザ名が正しく変数に格納され、それを参照してログが出力された」ことを表しています。

以上で、変数を利用したフローの作成が完了しました。

## 応用：変数を活用する

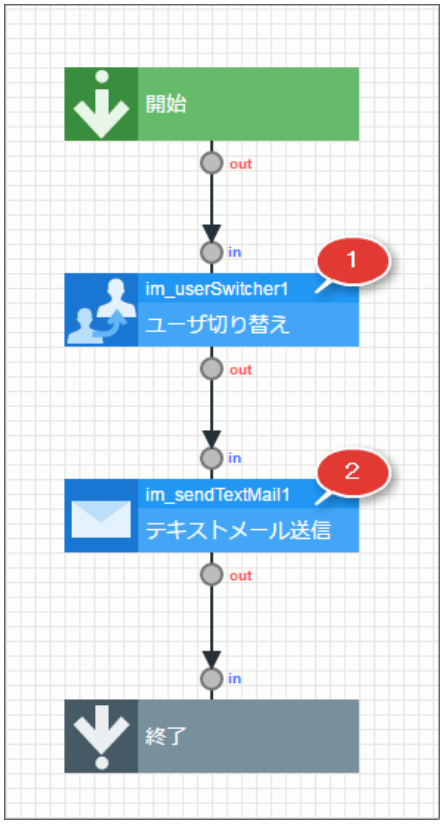
ここでは、これまで説明を行ってきた変数について、より具体的な活用方法の例を紹介します。
ロジックフローを実際に運用していくにあたって、変数の利用を検討する際の一助としてください。

なお、この章で紹介するロジックフローについて、詳細な設定は解説していません。
紹介しているロジックフローの実際の動作や設定項目を確認したい場合は、「*付録*」-「*チュートリアルデータのアーカイブファイル*」から該当するロジックフローをインポートしてください。

### Pattern1. ユーザ切り替え

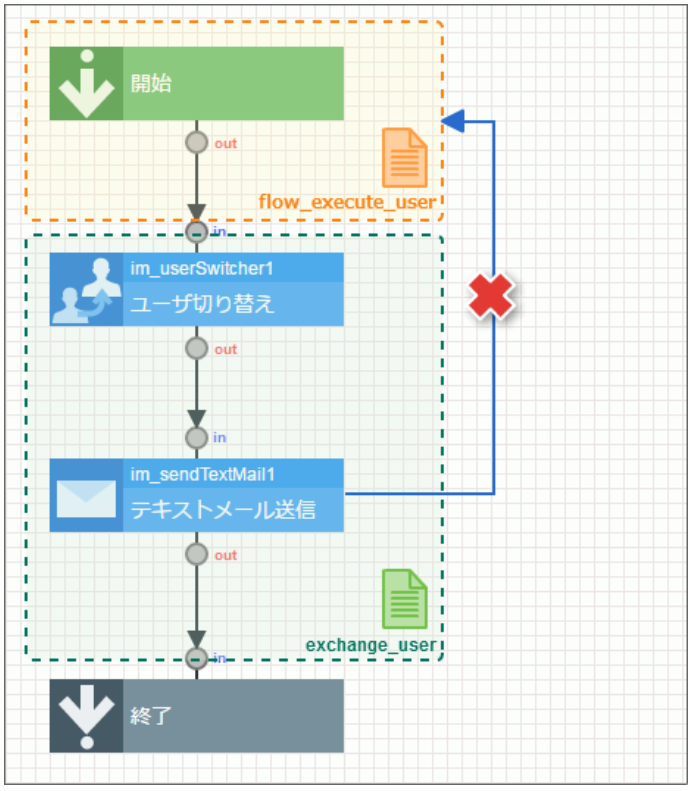
変数の活用方法として、ロジックフローが進行していく上で取得できなくなる値を格納する利用方法があります。
例えば、「メール送信をある特定のユーザで行う（Fromに指定する）ためにユーザ切り替えを利用するが、送信時には必ずCcにロジックフロー実行者のメールアドレスを指定する」ロジックフローを想定します。

以下は「ユーザ切り替え」タスクを利用し送信ユーザへ切り替え（１）、切り替えたユーザのメールアドレスをユーザコンテキストから取得しメールを送信する（２）フロー例です。



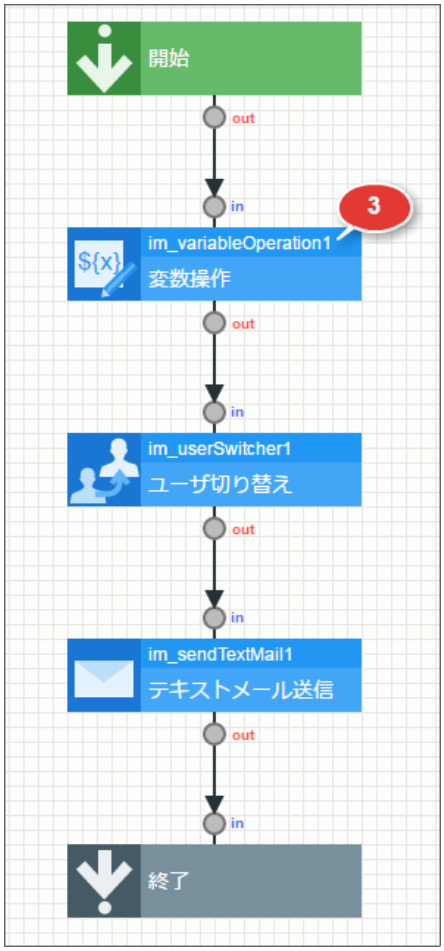

図：「ユーザ切り替え」タスクを用いたフロー例

しかし、このフローではロジックフロー実行ユーザの情報（メールアドレス）をユーザコンテキストから取得することができません。
（ユーザコンテキストから取得される値の仕様は「*様々な入力情報の利用*」-「*暗黙的な変数の利用*」を参照してください）



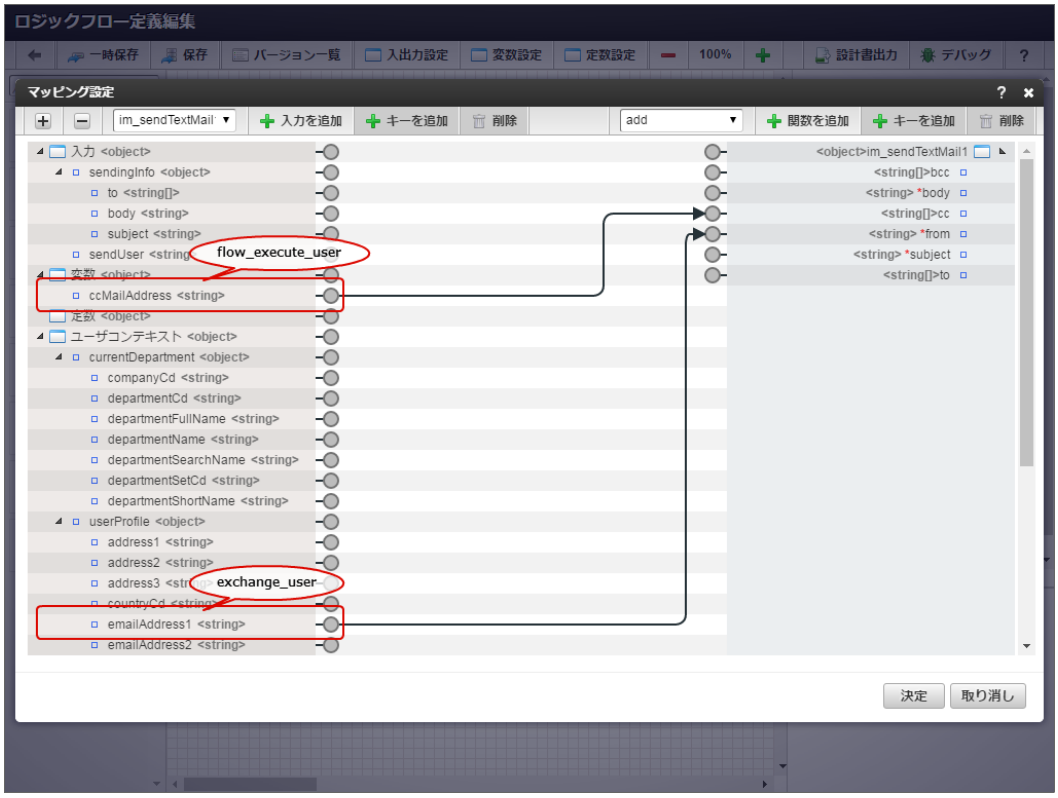

図：対象ユーザが切り替わっているため、実行ユーザ情報が取得できない。

ここで変数としてロジックフロー実行ユーザの情報を保持する変数を定義し、ユーザ切り替え前にこの変数へ情報を格納しておく（3）ように、フローを変更します。




図：ユーザ切り替え前に、実行ユーザの情報を格納しておくフロー例

ユーザ切り替え後のメール送信タスクのCcには、変数に保存したアドレスを利用します。




図：「テキストメール送信」タスクのマッピング例

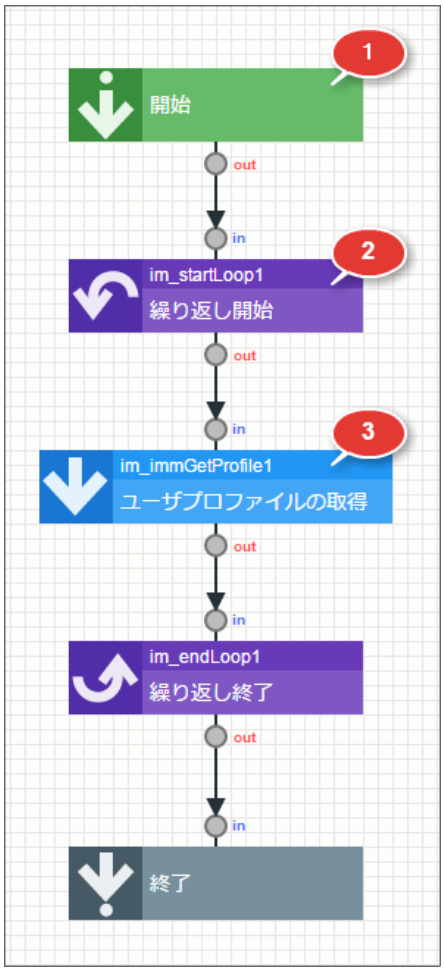
これにより、メール送信時に切り替えたユーザとロジックフロー実行ユーザそれぞれのアドレスを指定することが可能になりました。

Pattern2. 繰り返し処理

変数と「*繰り返し処理を利用したフロー*」で説明した繰り返し処理の相性は非常に良好です。

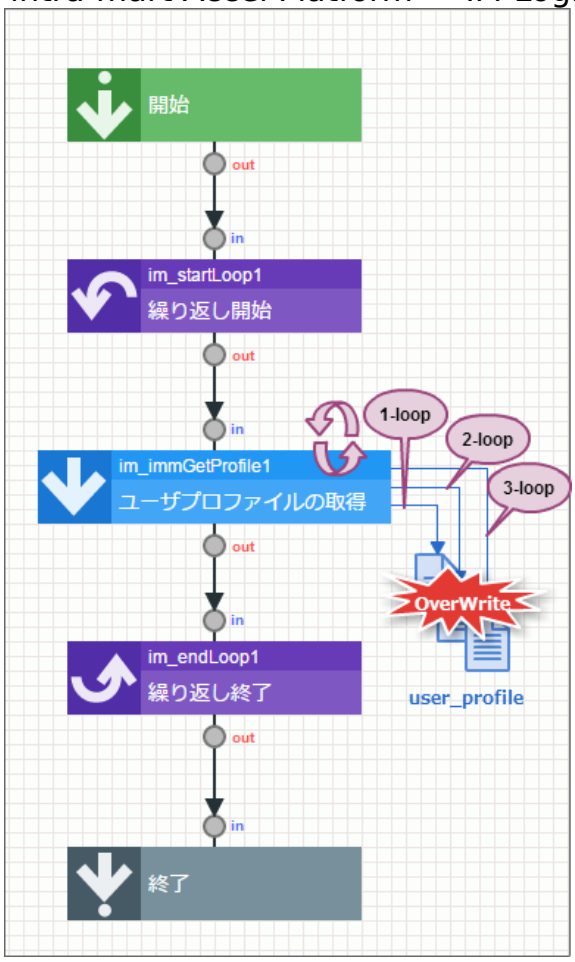
ここでは、「入力としてユーザコードの配列を受け取り、ユーザ名をIM-共通マスタから取得し、ユーザ名の配列として出力する」ロジックフローを想定します。

入力としてユーザコードの配列が与えられる（1）ので、繰り返し処理を利用し（2）、一つずつユーザコードを処理します（3）。



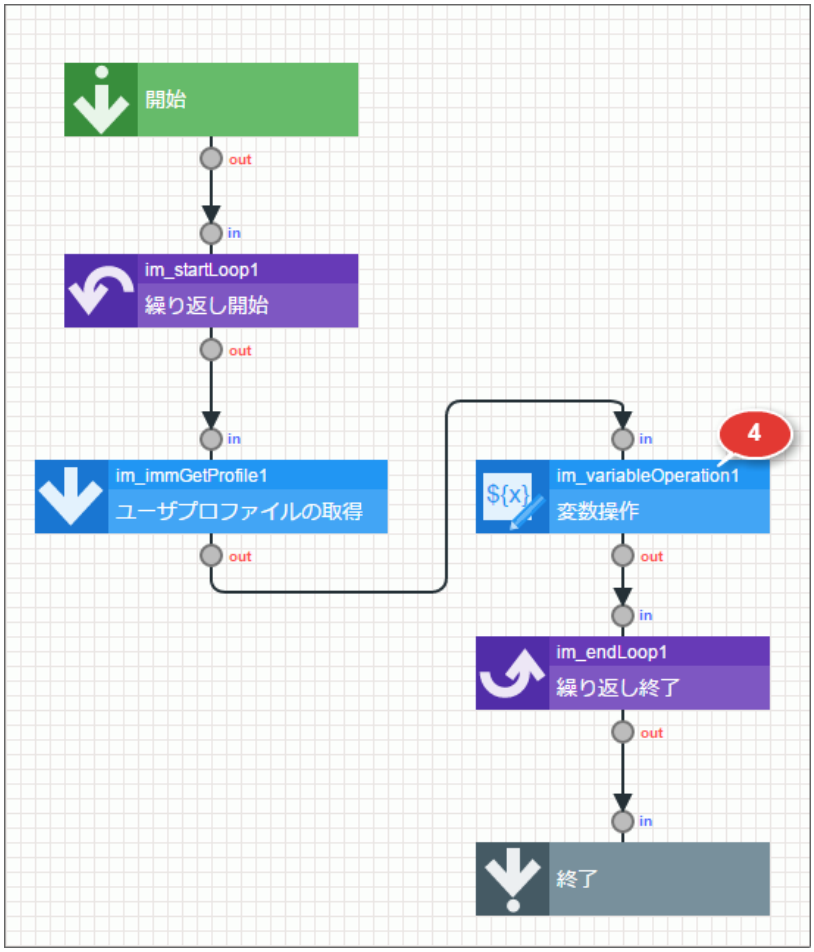

図：「繰り返し」制御要素を用いたフロー例

しかし、このままでは取得したユーザ名は次の繰り返し処理が行われた際に上書きされ、保持しておく事ができません。

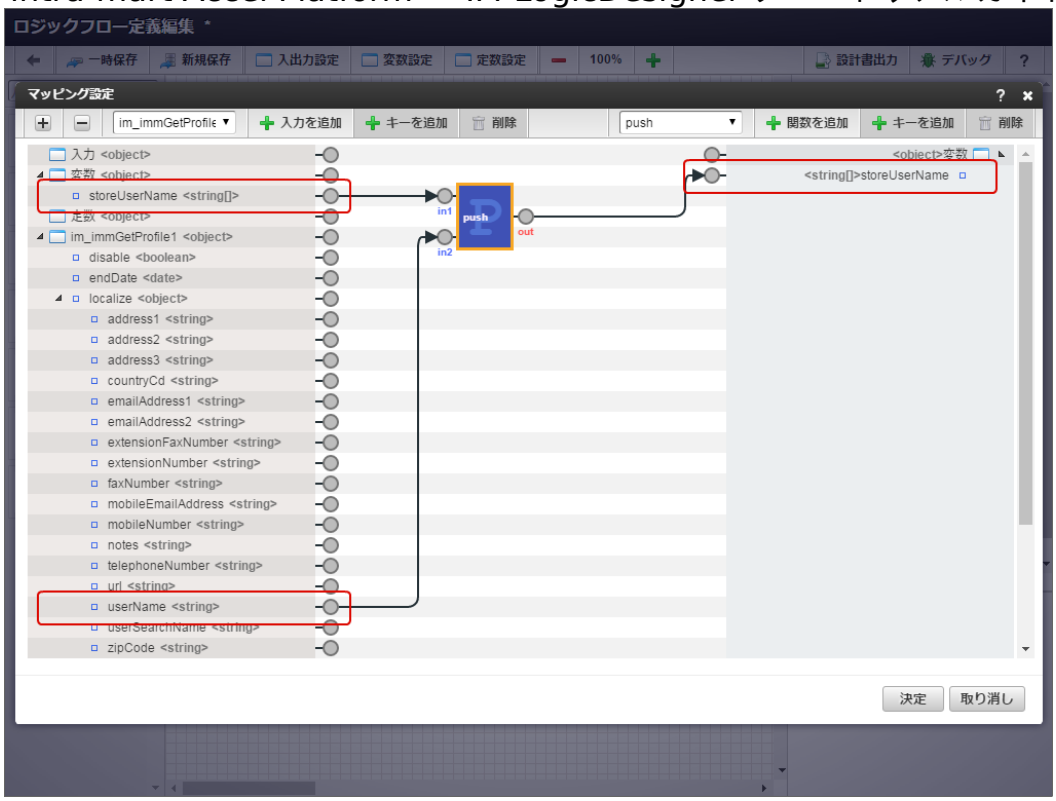



図：「ユーザプロファイルの取得」で取得された情報が、次の繰り返しでは保持できない

ここで変数として取得したユーザ名を保持しておく配列を定義し、取得したユーザ名を都度この変数に格納していく（４）フローに変更します。




図：取得したユーザ名を「変数操作」で格納していくフロー例




図：「変数操作」のマッピング例

これにより、入力されたユーザコードに対応するユーザ名は変数の配列に格納され、出力へユーザ名の配列（変数）を渡すことが可能になりました。

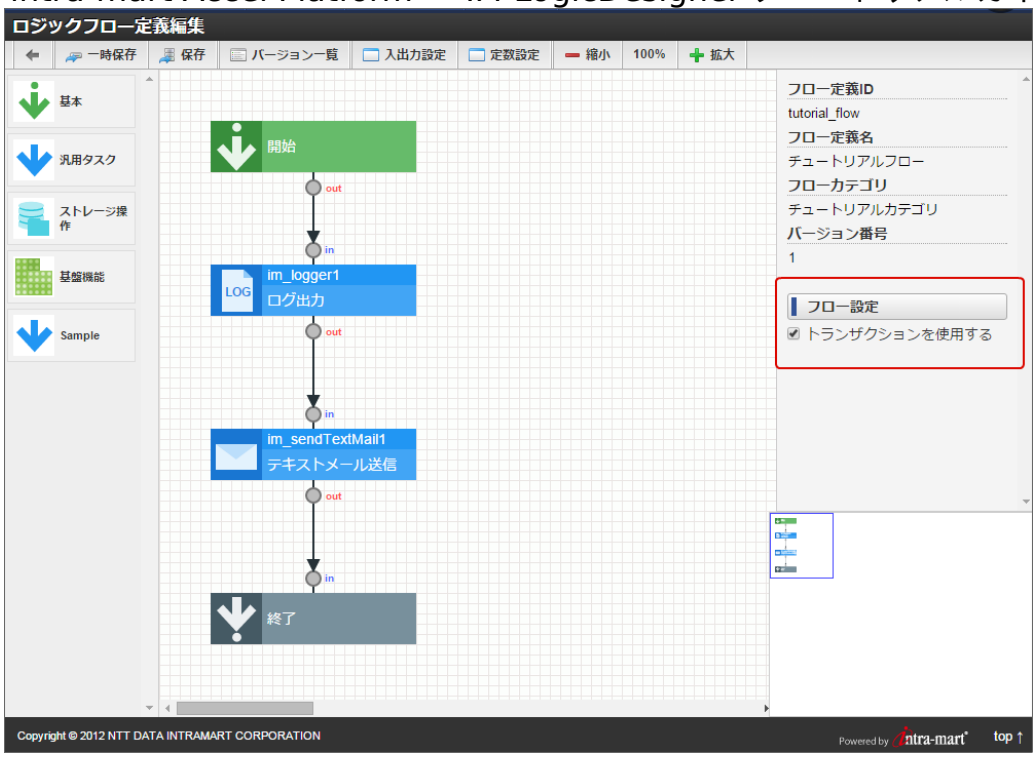
## ロジックフローに共通する設定詳細

この章では、ロジックフロー、および、各種エレメントに共通する設定項目についての詳細を説明します。



### ロジックフローのフロー設定

ロジックフロー定義編集画面では、フロー定義やマッピングの他に、ロジックフロー全体に対する設定（フロー設定）を行う事ができます。
ロジックフローのフロー設定画面はロジックフロー定義編集画面に遷移した際の初期状態として表示されている他、フロー編集画面上で配置されている何れのエレメントにもフォーカスが当たっていない際に表示されます。




図：ロジックフローのフロー設定

フロー設定で設定可能なプロパティは以下の通りです。

- トランザクションを使用する

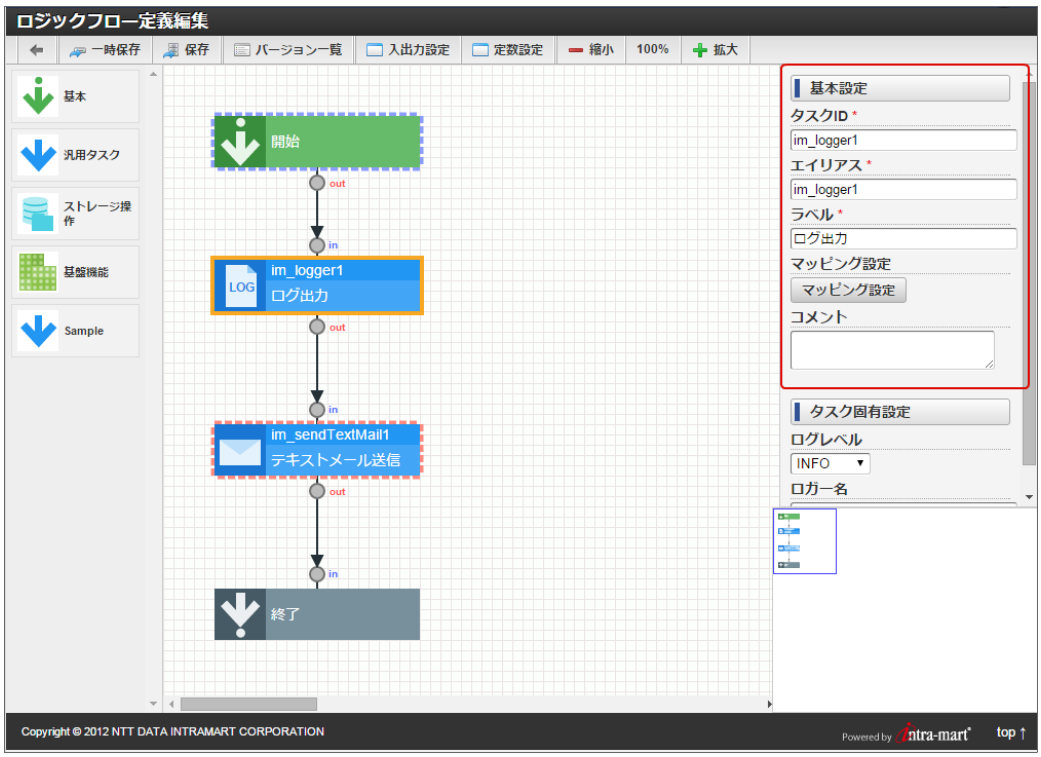
トランザクションを使用する（ロジックフローのトランザクション制御）

フロー設定画面から、ロジックフローのトランザクション制御を指定できます。

「トランザクションを使用する」プロパティにチェックを入れた場合、ロジックフロー開始時にトランザクションの開始も同時に行います。
「トランザクションを使用する」プロパティのチェックを外した場合、または、ロジックフローの呼び出し元で既にトランザクションが開始されていた場合、トランザクションは開始されません。

## エレメントの基本設定

エレメントには、各エレメントで独自に定義された「タスク固有設定」と、エレメントの処理内容に寄らず共通的に定義されている「基本設定」があります。



図：エレメントの「基本設定」と「タスク固有設定」

「タスク固有設定」についてはこれまで以下の章で触れてきました

- 「*基礎編 - ファースト・ステップ*」 - 「*タスクのプロパティを設定する*」では、エレメントのプロパティのうち「タスク固有設定」に属するプロパティを編集しました。
- 「*条件分岐を利用したフロー*」、「*繰り返し処理を利用したフロー*」、および、「*サブフロー呼び出し*」では、タスク固有設定にそれぞれの制御を行う条件を定義しました。

ここでは、エレメントの「基本設定」について、その設定項目の説明を行います。
「基本設定」で設定可能なプロパティは以下の通りです。

- タスクID
- エイリアス
- ラベル
- コメント

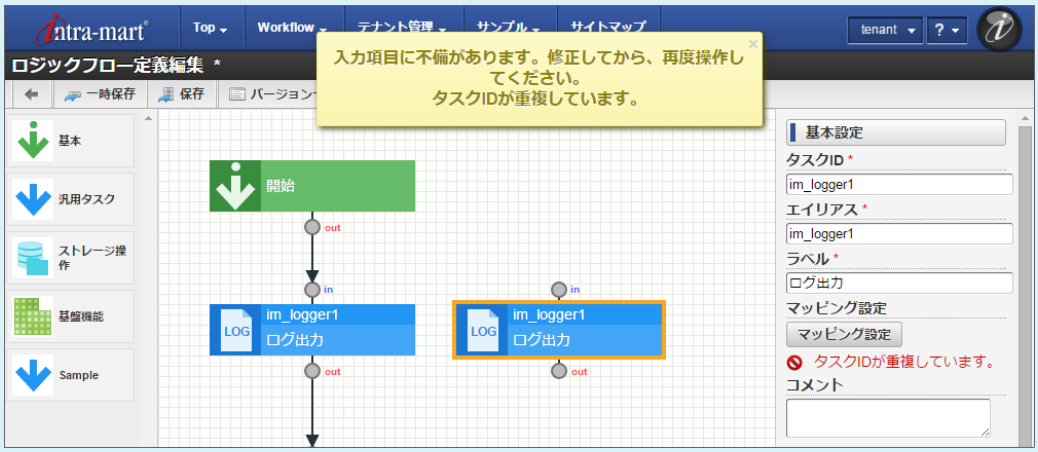
タスクID

エレメントに対して一意となるIDを設定します。
タスクIDは、フロー編集画面上に配置されているエレメント間で一意となるIDを指定する必要があります。

コラム

タスクIDの重複

複数のエレメントで同一のタスクIDが設定されていた場合、保存時に以下の警告メッセージが表示されます。




図：タスクIDが重複していた場合の警告メッセージ

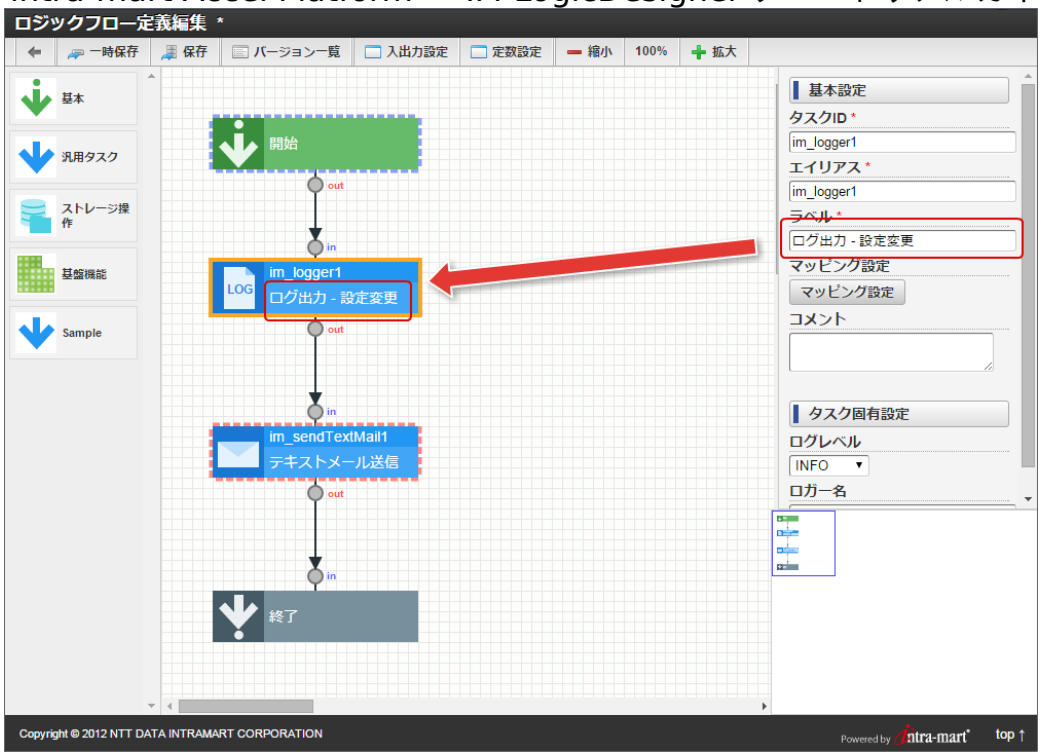
エイリアス

エレメントに対する別名を設定します。
エイリアスは、フロー編集画面上に配置されているエレメント間で、共通した名称を定義可能です。

エイリアスの詳細、および利用例は「IM-LogicDesigner仕様書」 - 「ロジックフロー」- 「エイリアス」を参照してください。

ラベル

エレメントの表示名を設定します。
ラベルに設定された内容は、フロー編集画面上に配置されたエレメント内の下部に表示されます。




図：ラベルの反映先

同じエレメントを複数配置した場合など、そのエレメントの具体的な処理情報をラベルとして定義することで、フローの視認性を上げることができます。

> **コラム**
>
> ラベルの多言語対応について
>
> ラベルに定義された文言は多言語対応されません。
> フロー編集画面上のエレメント上に表示される文言は、全てのロケール環境において同じラベルが利用されます。

コメント

エレメントに関するコメントや備考を設定します。
コメントに設定された内容はフロー編集画面上には表示されません。

一般的な備考の記載のほか、「*ユーザ定義の作成*」で作成した独自タスクへの補足事項の記載等に有用です。

> **コラム**
>
> コメントの多言語対応について
>
> コメントに定義された文言は多言語対応されません。

## 柔軟なマッピング情報の定義

この章では、「*基礎編 - ファースト・ステップ*」-「*マッピング設定を行う*」で扱ったマッピング設定について、より多様な定義方法を説明します。
開発者は、エレメントへのマッピングについて、より深い表現力を習得できます。

### 様々な入力情報の利用

この章では、マッピング設定で利用可能な入力情報の詳細について説明します。



#### 他のエレメントの出力値の利用

マッピング設定画面の初期状態では、入力値として以下の値が利用可能でした。

- 入力<object> - ロジックフローの入出力設定で設定された入力値
- 定数<object> - ロジックフローの定数設定で設定された値

これらに加えマッピング設定では、エレメントの出力値を、他のエレメントの入力値として設定することが可能です。

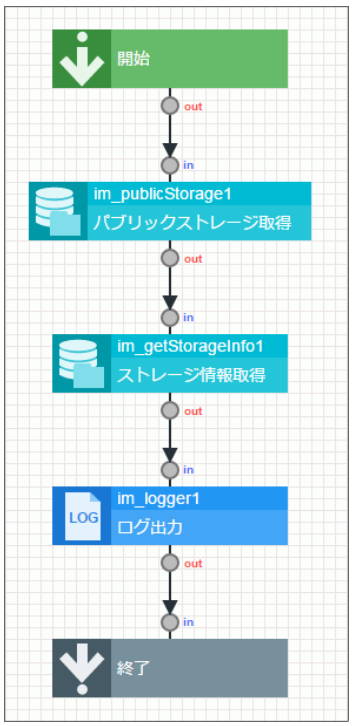
> **コラム**
>
> 出力値を持つエレメント/持たないエレメント
>
> IM-LogicDesignerが提供するエレメントの、出力値の有無についての詳細は、「IM-LogicDesigner仕様書」 - 「付録」 - 「タスク一覧」を参照してください。

本章で利用するフロー

本章では、以下のタスクを説明のために利用します。

1. 「パブリックストレージ取得」タスク（ストレージ操作）
2. 「ストレージ情報取得」タスク（ストレージ操作）
3. 「ログ出力」タスク（汎用タスク）

本章の追加方法を実施する場合、上記項番の順に接続されたロジックフローを準備してください。
以下は作成するロジックフローの接続例です。



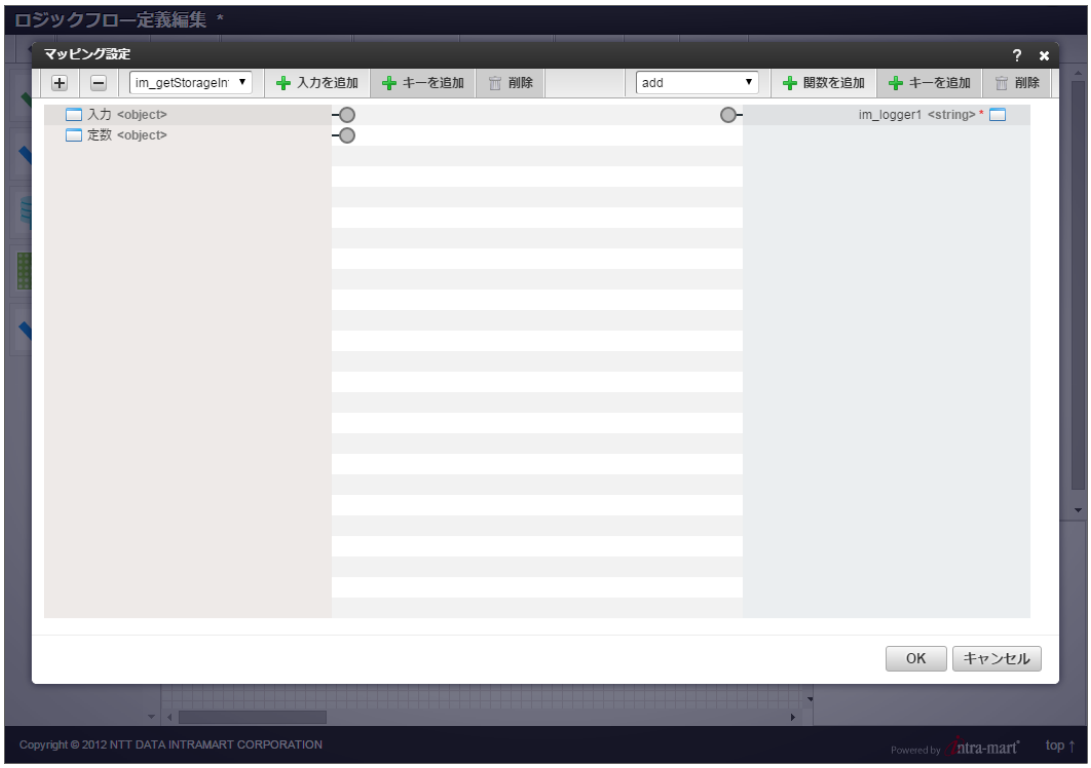

図：ロジックフロー作成例

> **コラム**
>
> その他の設定について
>
> 本章では、マッピング設定後の動作確認は行いません。
> そのため、本章で利用するフローに関する入力/出力設定や、フロールーティング設定等は行いません。

入力値として他のエレメントの出力値の追加

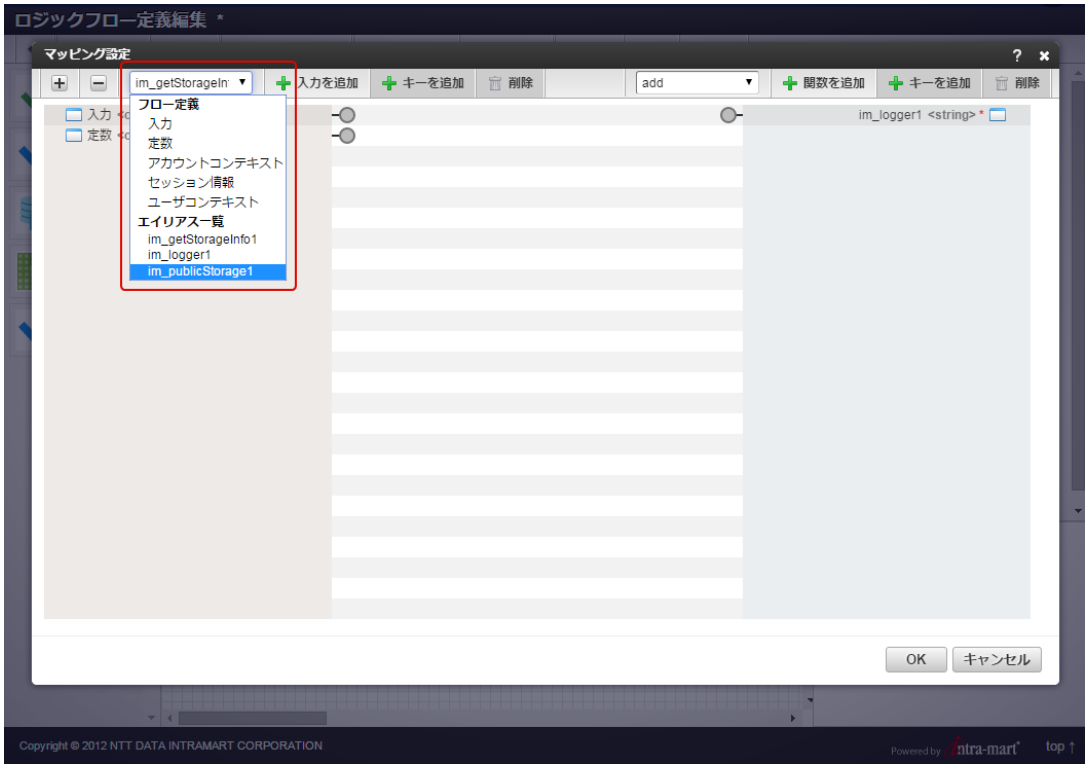
マッピング設定で、他のエレメントの出力値を、入力値として追加する方法を説明します。
本チュートリアルでは、「ログ出力」タスクのマッピング設定において、「パブリックストレージ取得」タスク、および、「ストレージ情報取得」タスクの出力値を、入力値として追加します。

1. フロー編集画面上の「ログ出力」タスクをクリックした上で、「マッピング設定」をクリックし、マッピング設定画面を開きます。



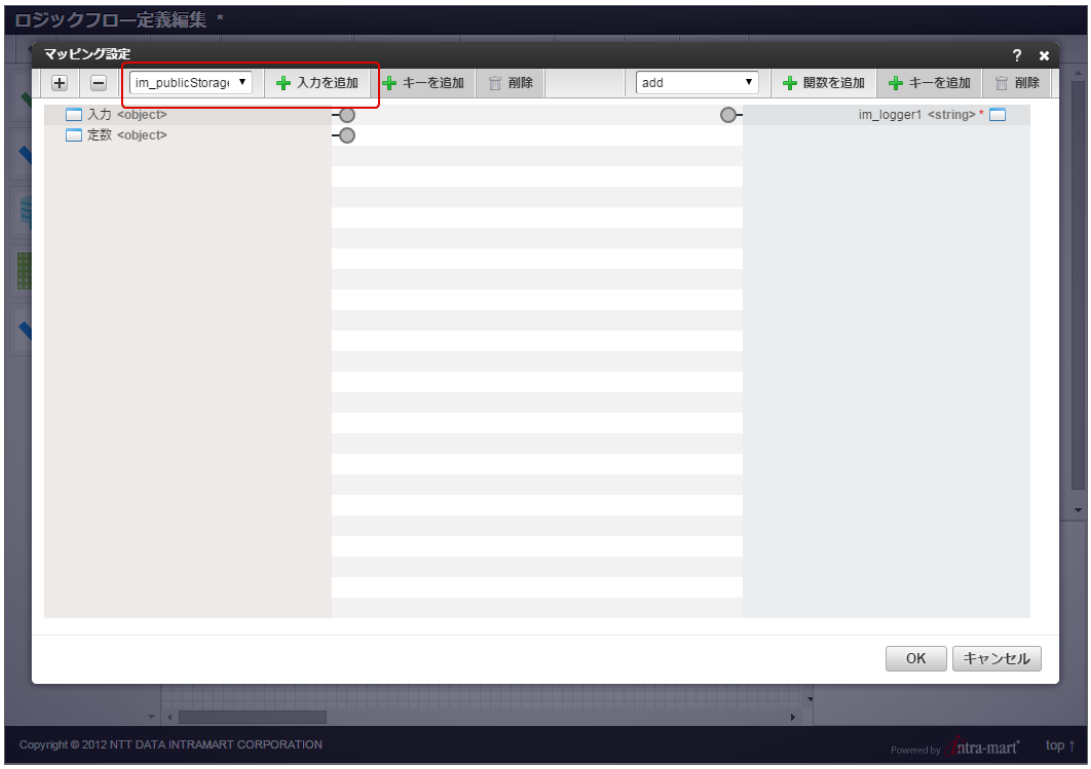

図：マッピング設定画面

2. マッピング設定画面上部、ヘッダ内の左上に位置するセレクトボックスをクリックし、以下の項目を選択します。
   - エイリアス一覧 - im_publicStorage1



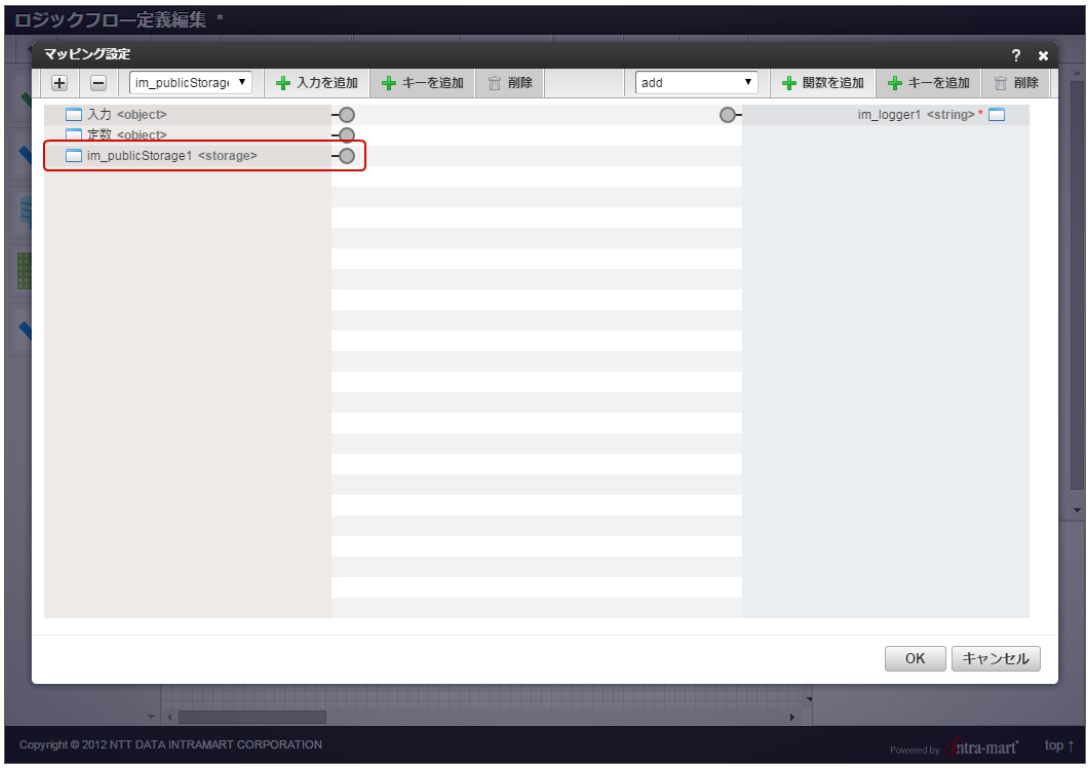

図：追加する入力のエイリアスを選択

3. セレクトボックスの中身が変更されたことを確認し、右側にある「入力を追加」をクリックします。



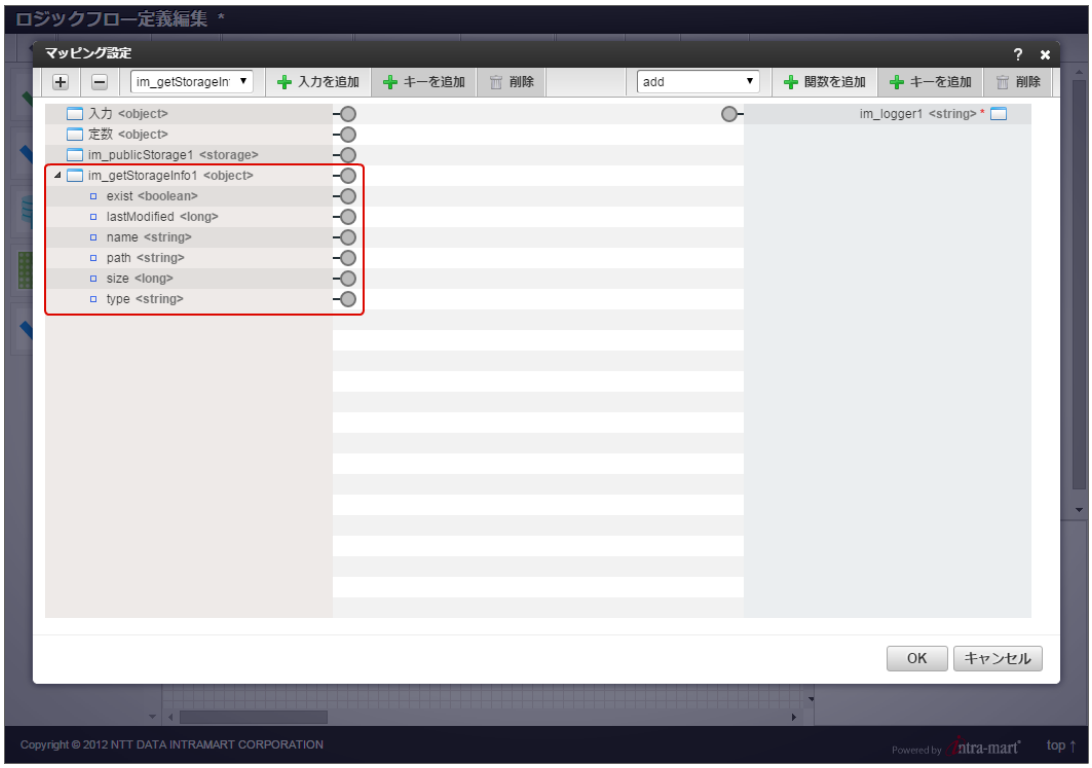

図：「入力を追加」をクリック

4. 入力値として、「パブリックストレージ取得」タスク（im_publicStorage1）の出力値が追加されました。




図：新しい入力値の追加（パブリックストレージ取得）

5. 同様の手順で、「ストレージ情報取得」タスク（im_getStorageInfo1）の出力値を追加します。
「ストレージ情報取得」タスクのような、出力値として階層化された値を持つ場合、その構造をもとに入力へ追加されます。




図：新しい入力値の追加（ストレージ情報取得）

以上で、他のエレメントの出力値を、入力値として追加できました。

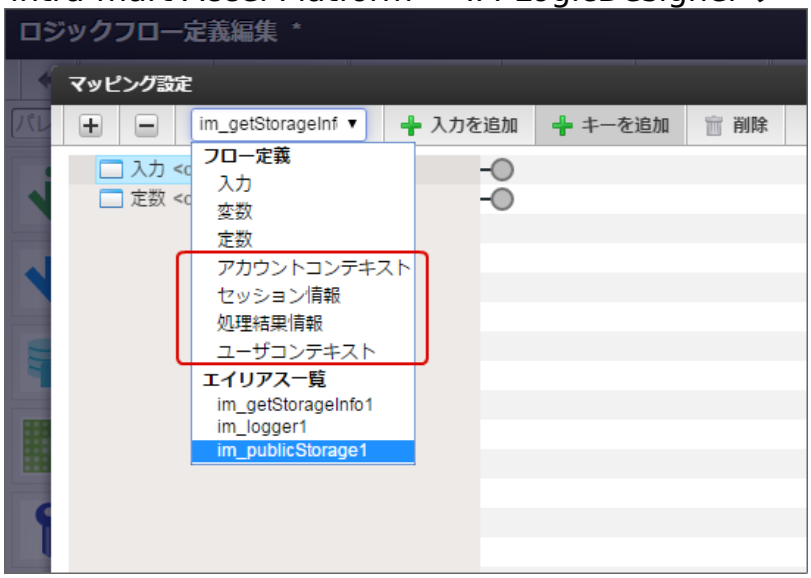
> **注意**
>
> 他のエレメントの出力値の利用位置と実行タイミング
>
> 他のエレメントの出力値は、対象となるエレメントが実行されてはじめて利用が可能です。
> 例えば、エレメントAの出力値をエレメントBで利用したい場合、エレメントBはエレメントAよりも後に実行される必要があります。

### 暗黙的な変数の利用

IM-LogicDesignerでは、ユーザの定義した「*定数値を定義する*」以外にも、intra-mart Accel Platformの扱う以下の情報を暗黙的な変数、定数の一部として利用できます。

- アカウントコンテキスト
- セッション情報
- 処理結果情報
- ユーザコンテキスト
- 外部ユーザコンテキスト

これらの暗黙的な変数は、「*他のエレメントの出力値の利用*」と同様の方法で、入力値として追加して利用できます。
より具体的には入力を選択するセレクトボックスから、追加を行いたい暗黙的な変数を選択します。




図：システムの提供する暗黙的な変数群

アカウントコンテキスト

マッピング設定では、アカウントコンテキストのプロパティから取得可能な情報を入力として利用可能です。
アカウントコンテキストの持つプロパティの詳細は以下のドキュメントを参照してください。

- 「アクセスコンテキスト仕様書」 - 「アカウントコンテキスト」
- API Document
  - AccountContext(Java)
  - AccountContextオブジェクト(JSSP)

> **注意**
>
> 対象となるアカウント
>
> アカウントコンテキストから取得されるアカウント情報は、ロジックフローを実行したユーザが対象です。
> ただし、「ユーザ切り替え」制御要素を利用した場合、切り替え以降のフローでは、切り替えたユーザのアカウントコンテキストが取得されます。

セッション情報

マッピング設定では、IM-LogicDesignerがロジックフロー実行中にセッション内に保持している以下の値を入力として利用可能です。

| プロパティ名 | 詳細 |
|---|---|
| flowId<string> | 実行中のロジックフローのフローIDです。 |
| startTime<date> | ロジックフローが実行された日時に関する日付情報です。 |
| systemDate<date> | システムロケールに対応したシステム日付情報です。<br>startTimeとは異なり、このプロパティが利用された時点の日付情報が取得されます。 |
| version<integer> | 実行中のロジックフローのフローバージョンです。 |
| fileSeparator<string> | ファイル区切り文字を表す定数値です。<br>このプロパティが実際に返す値はIM-LogicDesignerが動作しているOSに依存します。 |
| lineSeparator<string> | 改行コードを表す定数値です。<br>このプロパティが実際に返す値はIM-LogicDesignerが動作しているOSに依存します。 |
| baseUrl<string> | Webサーバのベースurlを表す定数値です。<br>この値は、テナント情報や設定ファイルに定義された値を取得します。<br>ベースURLが定義されていない場合は、リクエストオブジェクトからベースURLを解決します。<br>（ジョブや非同期タスクといったバックエンド処理で実行されており、リクエストオブジェクトが取得できない場合は値は取得できません。） |

処理結果情報

マッピング対象となるエレメントの一つ前のエレメントの処理結果情報を入力として利用可能です。

詳細は「*複雑なフローの定義*」-「*ロジックフローのエラーハンドリング*」を参照してください。

ユーザコンテキスト

マッピング設定では、ユーザコンテキストのプロパティから取得可能な以下の情報を入力として利用可能です。

- プロファイル情報
- 組織所属情報

ユーザコンテキストの持つプロパティの詳細は以下のドキュメントを参照してください。

- 「アクセスコンテキスト仕様書」 - 「ユーザコンテキスト」
- API Document
  - UserContext(Java)
  - UserContextオブジェクト(JSSP)

> **注意**
>
> ユーザコンテキストの利用について
>
> ユーザコンテキストをマッピング設定の入力として利用するには、ユーザコンテキストのモジュールがintra-mart Accel Platformに含まれている必要があります。

> **注意**
>
> 対象となるユーザ
>
> ユーザコンテキストから取得されるユーザ情報は、ロジックフローを実行したユーザが対象です。
> ただし、「ユーザ切り替え」制御要素を利用した場合、切り替え以降のフローでは、切り替えたユーザのユーザコンテキストが取得されます。

外部ユーザコンテキスト

マッピング設定では、外部ユーザコンテキストのプロパティから取得可能な以下の情報を入力として利用可能です。

- 外部ユーザかどうか

外部ユーザコンテキストの持つプロパティの詳細は以下のドキュメントを参照してください。

- 「アクセスコンテキスト仕様書」 - 「外部ユーザコンテキスト 」
- API Document
  - ExternalUserContext (Java)
  - ExternalUserContextオブジェクト (JSSP)

## マッピング関数の利用

この章では、マッピング設定で利用可能な関数の詳細について説明します。



### マッピング関数とは

マッピング関数とは、マッピング設定で利用可能な小規模な演算処理（四則演算や簡易な値の操作）の総称です。

「*基礎編 - ファースト・ステップ*」 - 「*マッピング設定を行う*」では、ロジックフローの入力値や定数値をそのまま出力先へ渡しました。
しかし、実際の業務や要件の中では、入力値をそのまま出力へ渡すのではなく、以下の様な処理結果を出力先へ渡したい場合があります。

- 数値形式の入力値に対する四則演算、および、四捨五入
- 文字列形式の入力値に対する文字列操作
- 特定のフォーマットへの整形

そうしたシチュエーションに合わせIM-LogicDesignerでは、いくつかの入力を組み合わせて目的の値を生成するための演算を行うマッピング関数を提供しています。
開発者はマッピング関数を利用することで、簡易な演算処理であればマッピング設定内で完結して定義することができます。

> **コラム**
>
> 提供するマッピング関数一覧
>
> IM-LogicDesignerが提供するマッピング関数の一覧、および、それらの詳細は「IM-LogicDesigner仕様書」 -「付録」- 「マッピング関数一覧」を参照してください。

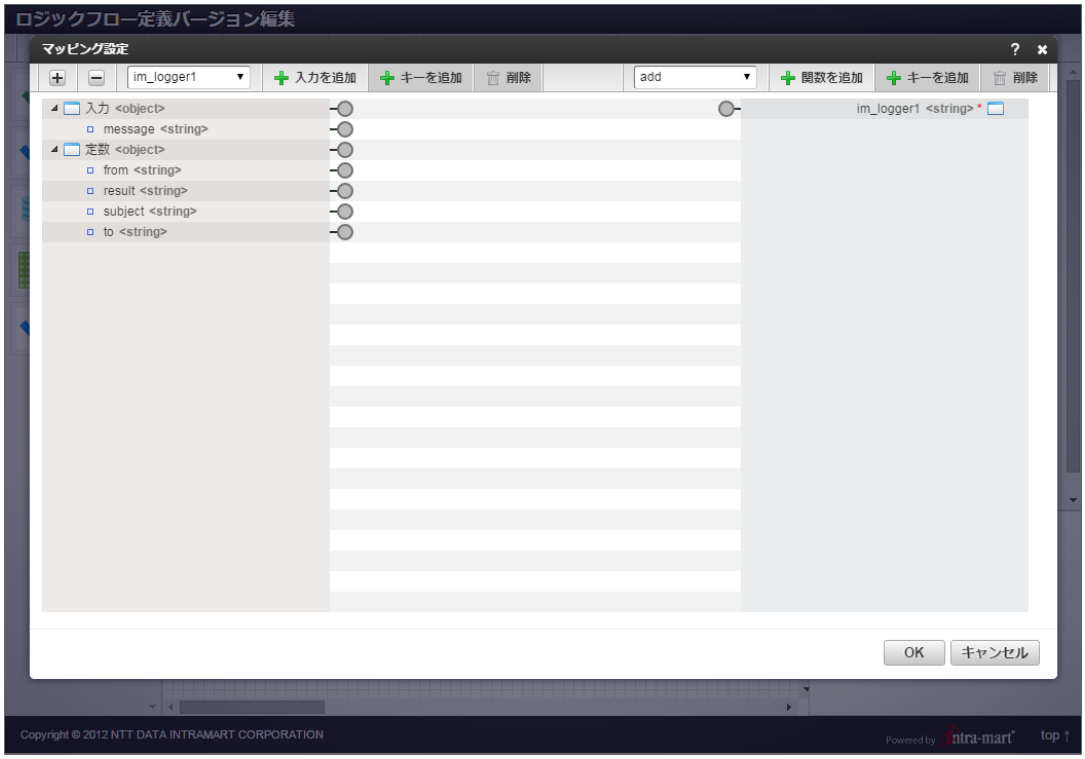
本章の実施環境について

本章では、マッピング関数の追加、および、マッピングの方法を説明する上で「*基礎編 - ファースト・ステップ*」で作成したロジックフローのうち、「ログ出力」タスクのマッピング設定を利用しています。
また説明を行う都合上、基礎編のチュートリアル内で定義を行ったマッピング設定を一部削除しています。

## マッピング関数の追加

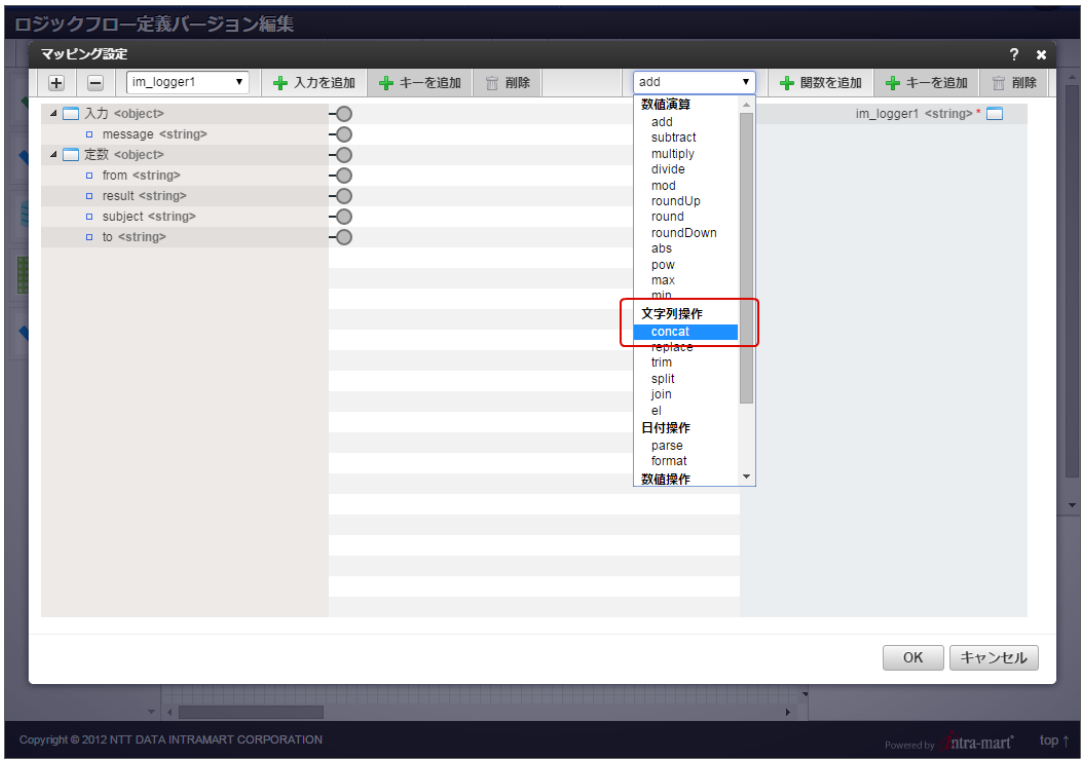
マッピング設定でマッピング関数を追加する方法を説明します。

1. フロー編集画面上の「ログ出力」タスクをクリックした上で、「マッピング設定」をクリックし、マッピング設定画面を開きます。



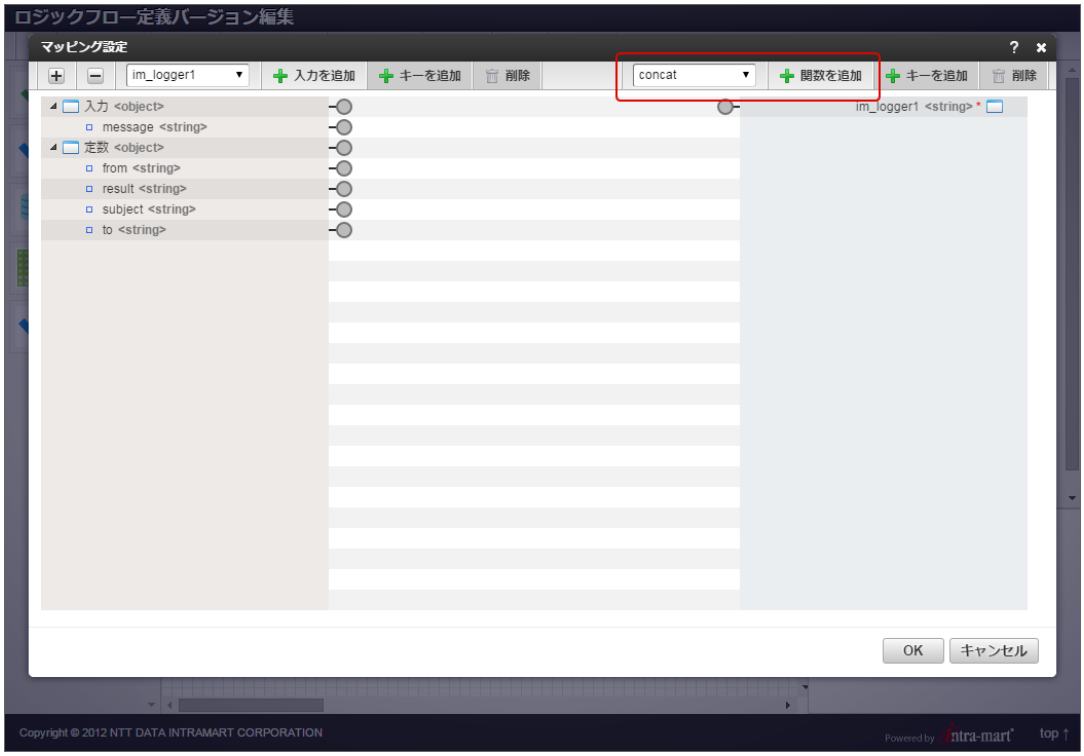

図：マッピング設定画面

2. マッピング設定画面上部、ヘッダ内の右上に位置するセレクトボックスをクリックし、以下の項目を選択します。
   - 文字列操作 - `concat`



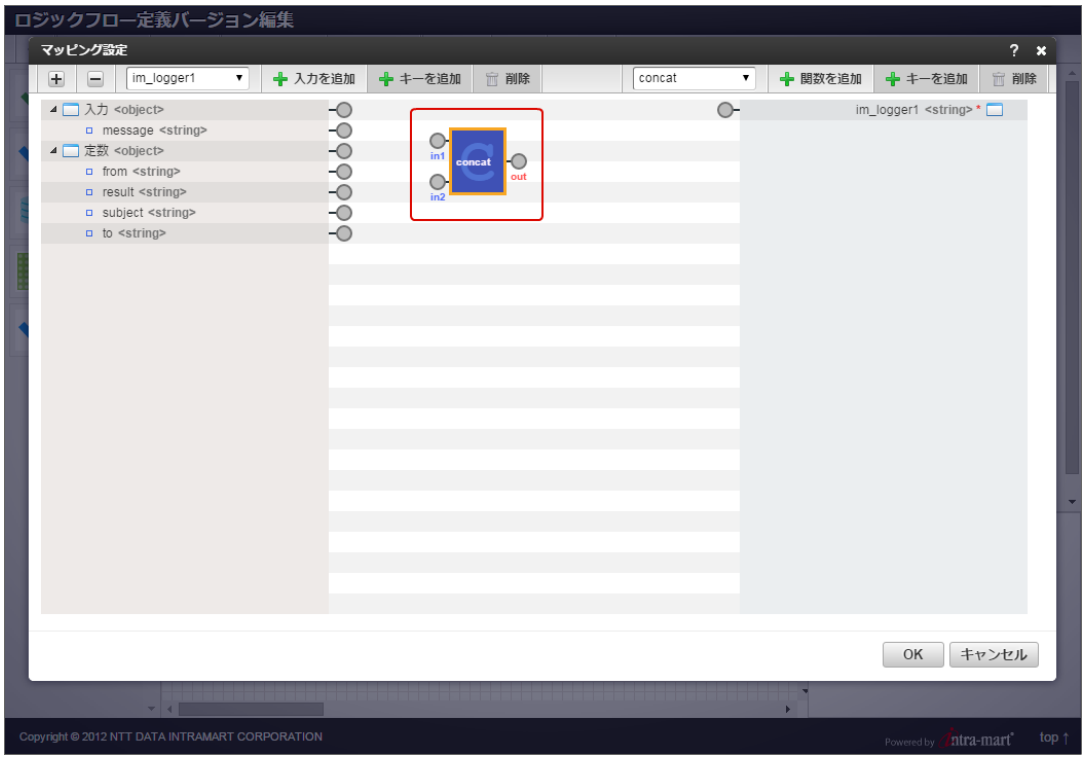

図：マッピング関数「concat」の選択

3. セレクトボックスの中身が変更されたことを確認し、右側にある「関数を追加」をクリックします。




図：「関数を追加」をクリック

4. マッピング関数として、「concat」関数が追加されました。



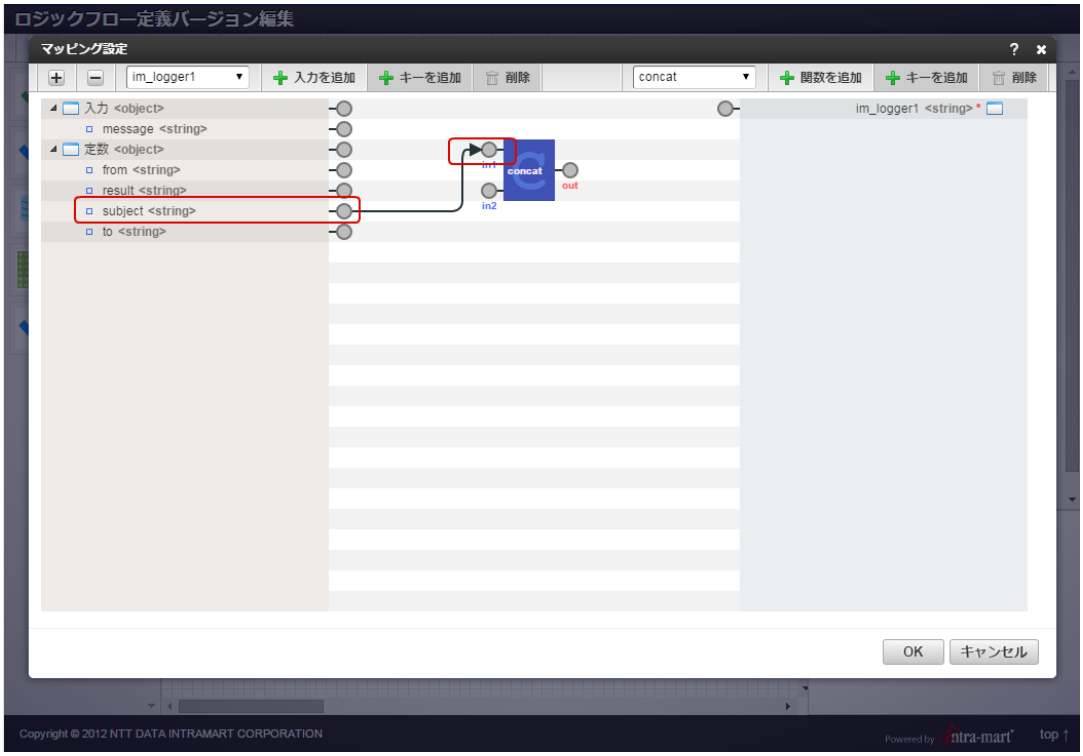

図：関数（concat）の追加

以上で、マッピング関数を追加できました。

## 関数を利用したマッピング

マッピング設定でマッピング関数を利用したマッピングを行う方法を説明します。
ここでは、追加した「concat」関数を利用して、ロジックフローの入力値である「message<string>」と、定数値である「subject」を結合した値を出力先の「ログ出力」タスクへ渡します。

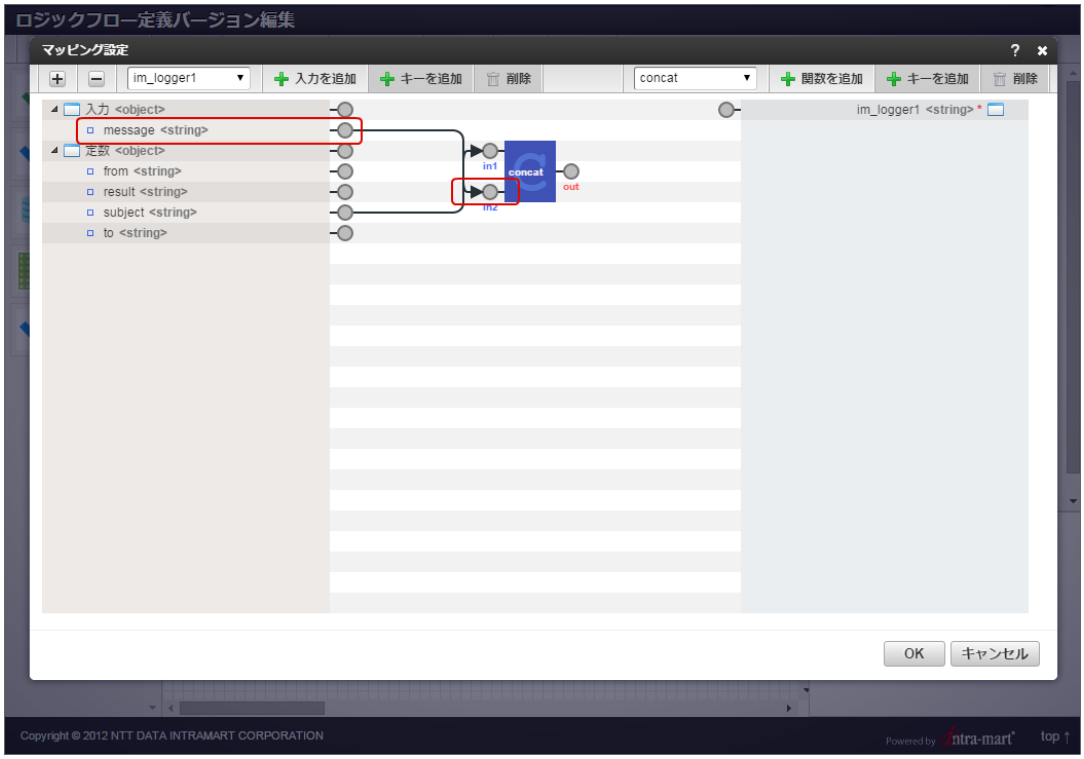
1. マッピング関数として、「concat」関数が追加された状態から開始します。
2. 設定画面左部、「定数<object>」要素の下にある「subject<string>」から出ている端子をドラッグし、「concat」関数の左部から出ている端子のうち「in1」へドロップします。




図：関数と入力値の接続（その１）

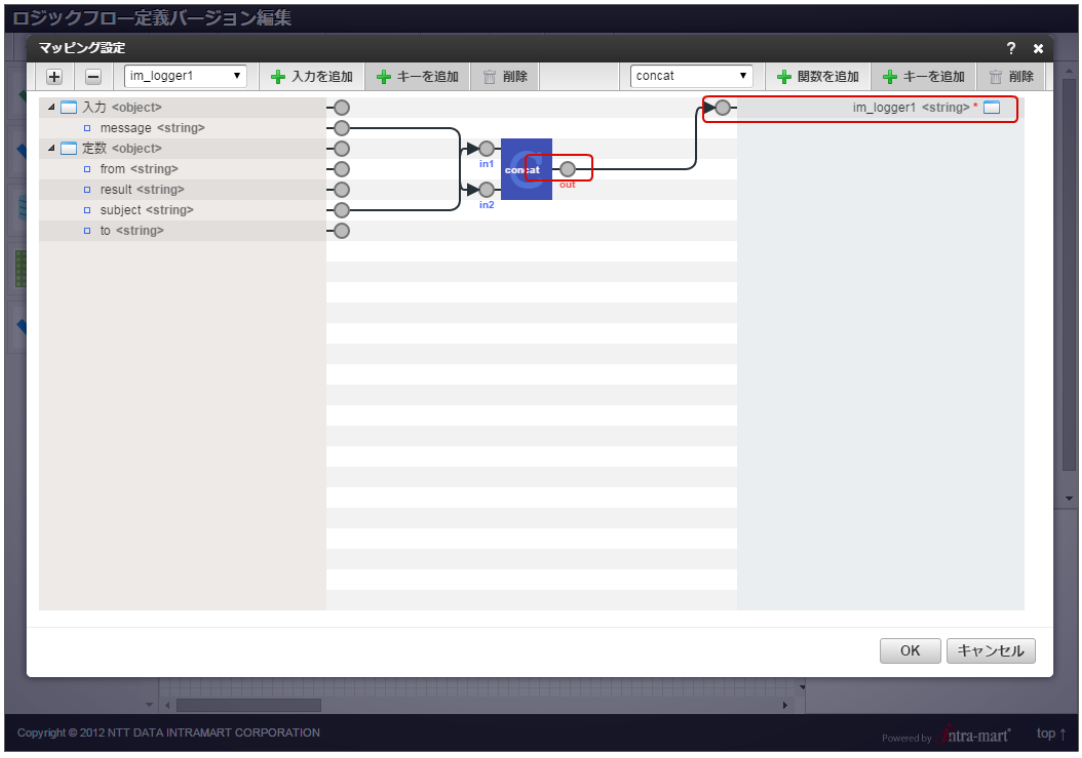
3. 同様に、「入力<object>」要素の下にある「message<string>」から出ている端子をドラッグし、「concat」関数の左部から出ている

端子のうち「in2」へドロップします。




図：関数と入力値の接続（その２）

4. 「concat」関数の右部から出ている端子「out」をドラッグし、設定画面右部、「im_logger1<string>」から出ている端子にドロップします。




図：関数と出力値の接続

5. これにより、マッピング関数を利用して、結合した値がマッピングされました。

以上で、関数を利用したマッピングが完了しました。

コラム

マッピング関数と型変換

以下のマッピング設定において相互の型が異なる場合、IM-LogicDesignerは通常のマッピング設定と同様に自動で型を変換し、解決を行います。

- 「エレメントの入力」と「マッピング関数の引数」
- 「マッピング関数の返却値」と「エレメントの出力」

変換可能、および、不可能なパターンについての詳細は、「IM-LogicDesigner データ型変換 仕様書」を参照してください。

## マッピングのデバッグ

この章では、マッピングのデバッグ機能について説明します。



### マッピングのデバッグ機能とは

マッピングのデバッグ機能とは、設定したマッピングを実際に実行して、入力値に対する出力値が期待通りの値になるか確認を行える機能です。デバッグ機能を用いることで、以下のようなマッピング上の問題を解決することが可能です。

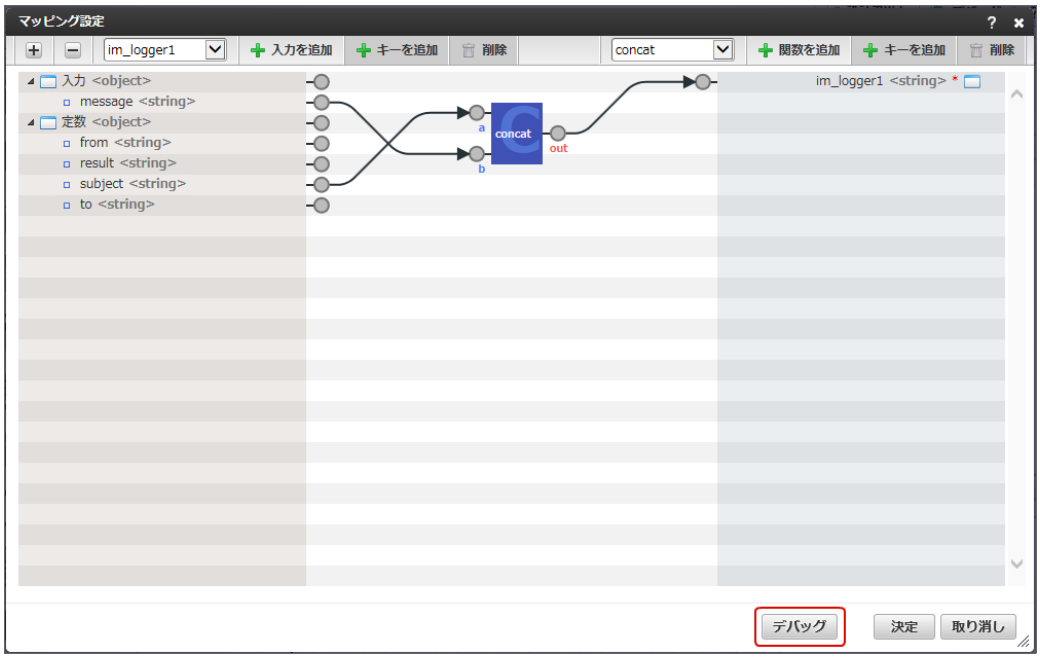
- 作成したマッピングで期待する値が出力されない（どの接続/関数で失敗しているのか特定したい）。
- 複雑なマッピング設定の場合、期待通りに接続されているのかわからない。
- 配列の有無や異なる型同士の接続において、最終的に期待する値に変換されるかどうかわからない。

また、こうした問題解決以外にも、作成したマッピングの簡単なテスト実行にも利用できます。

### デバッグ画面を表示する

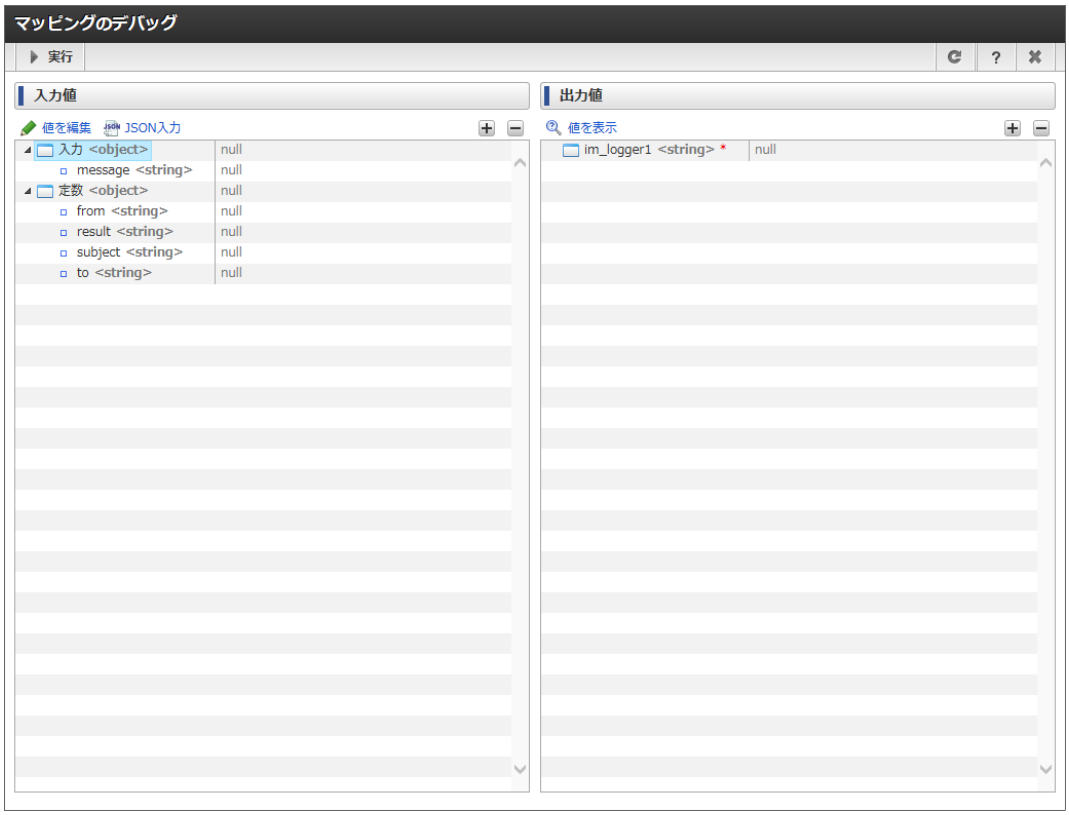
マッピングのデバッグ画面は、マッピング設定画面から開きます。

1. マッピング設定画面下部の「デバッグ」をクリックします。




図：ダイアログ内の「デバッグ」

2. マッピングのデバッグ画面が新規にポップアップして表示されます。

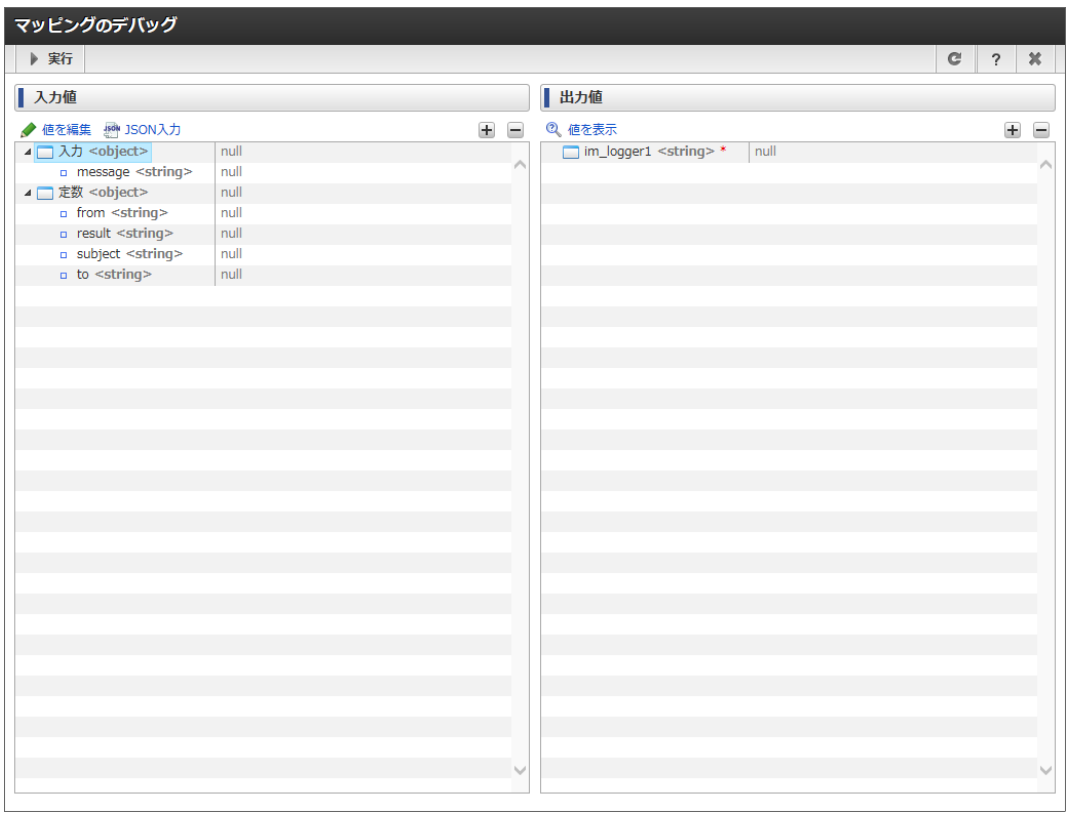



図：デバッグ画面

## マッピングをデバッグ実行する

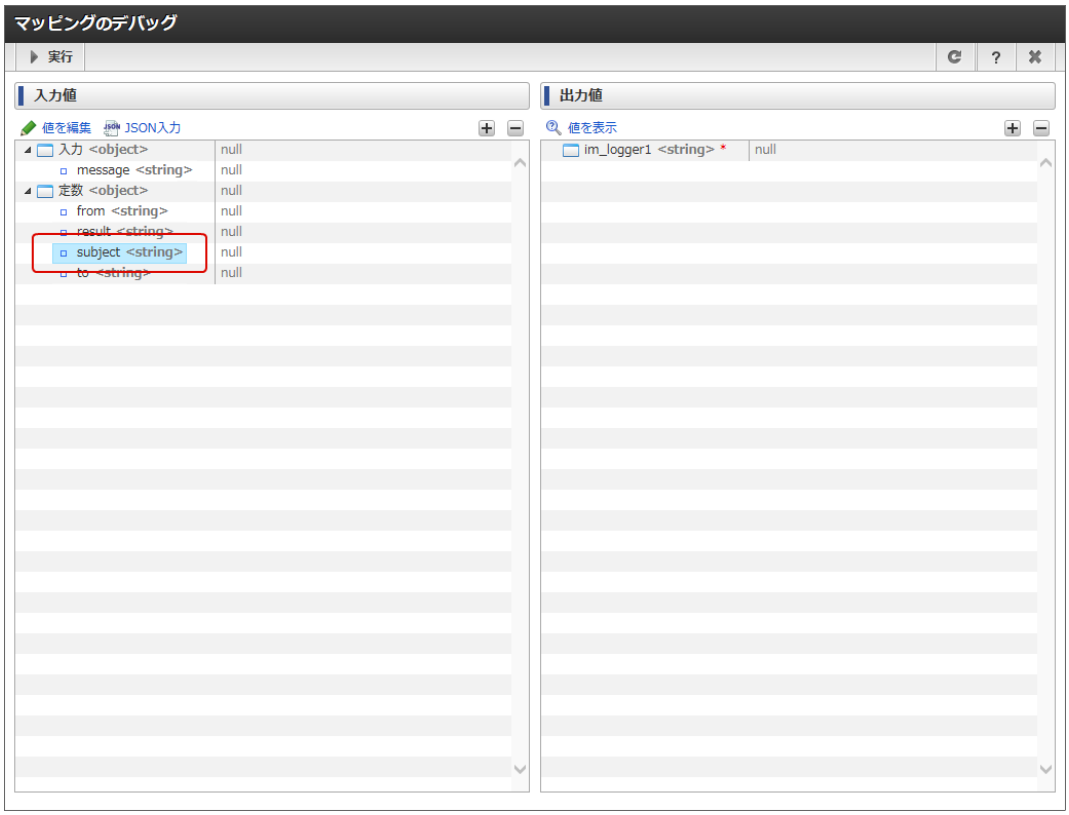
マッピングをデバッグ実行する方法を説明します。

1. デバッグ画面を開きます。
2. デバッグ実行を行う上で必要な「入力値の設定」項目が表示されます。



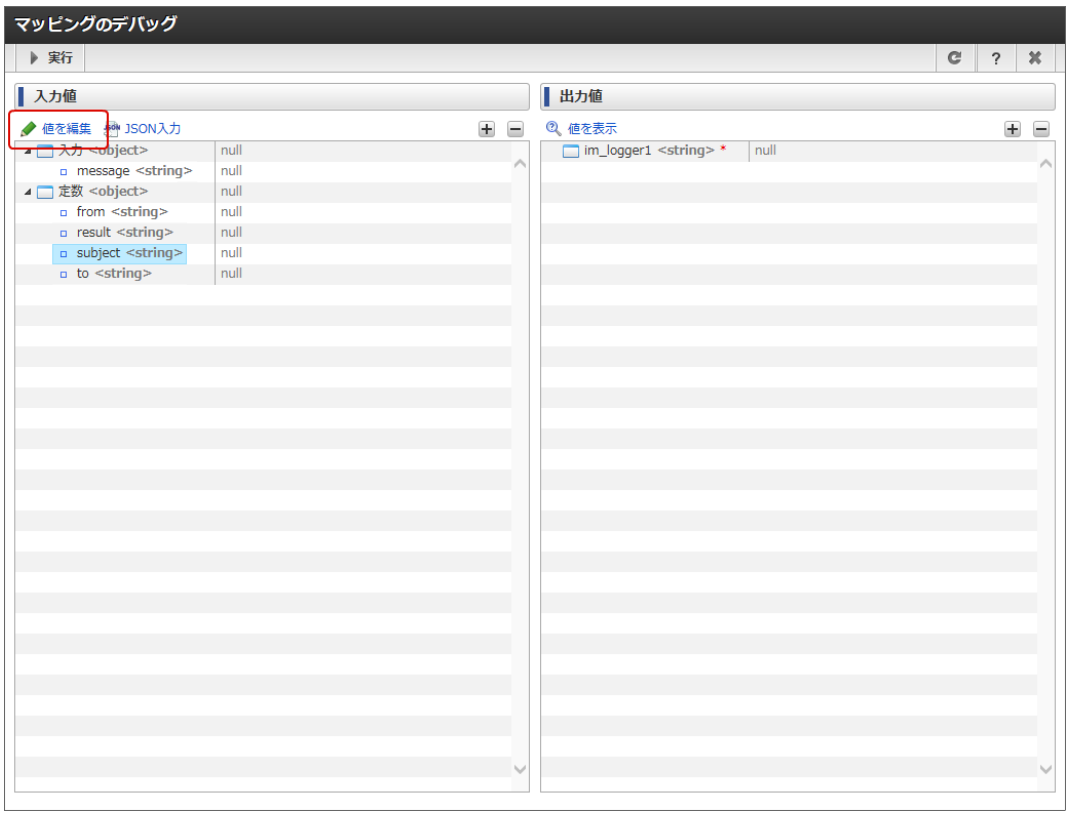

図：デバッグの実行設定画面

3. 定数<object>配下の`subject<string>`をクリックし、選択状態にします。



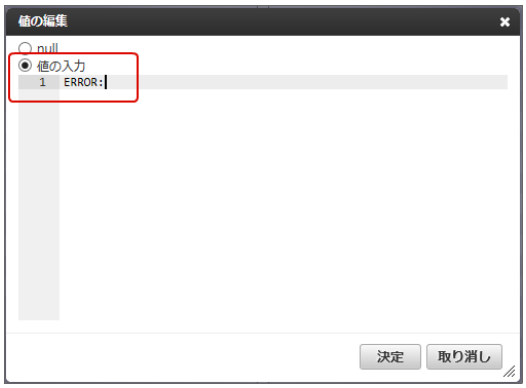

図：デバッグの実行設定画面 - 入力のフォーカス

4. 値一覧の左上の「値の編集」をクリックします。



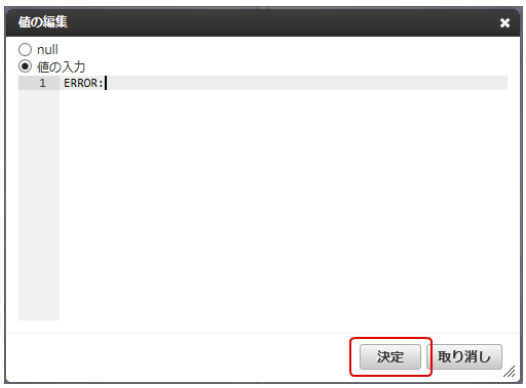

図：値の編集

5. 値の編集を行う画面が表示されるので、入力欄に以下の値を設定します。
   （値の設定を行うことで、自動で画面上部の「値の入力」ラジオボタンがオンになることを確認してください）
   - 設定値 - 「ERROR:」



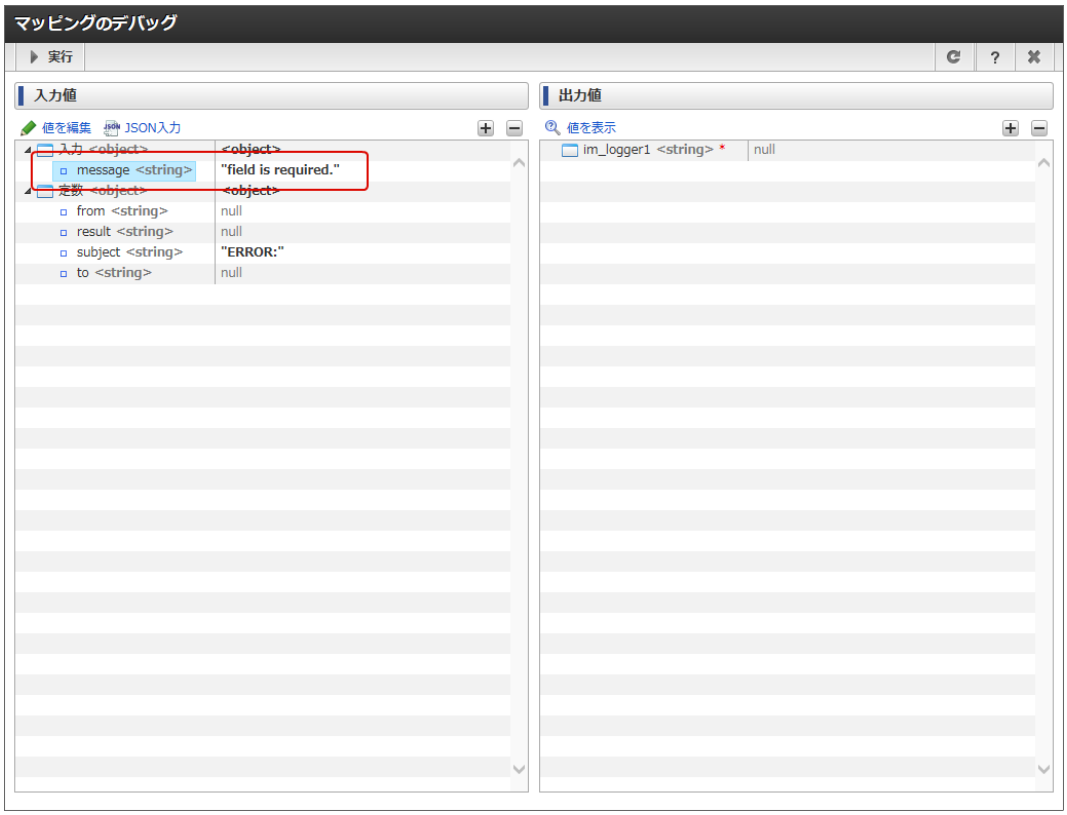

図：デバッグ用設定値の入力

6. 値の編集画面下部の「決定」をクリックします。



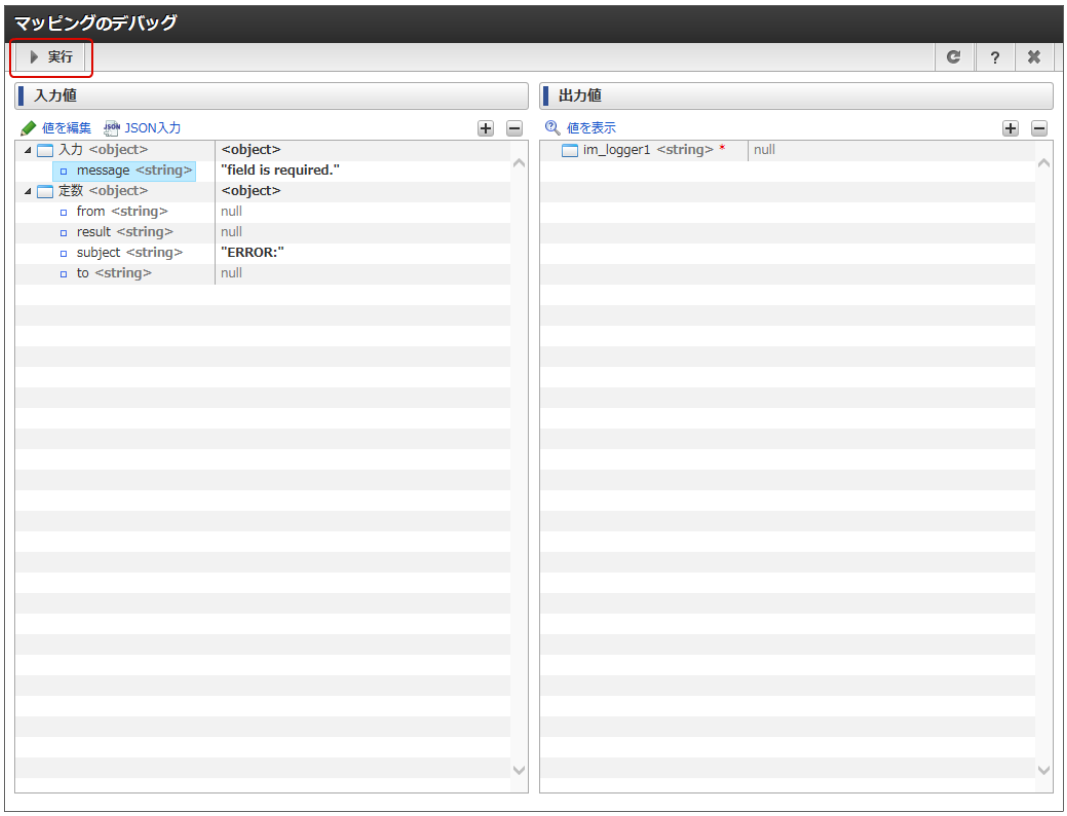

図：設定値の決定

7. 同様に、入力<object>配下のmessage<string>をクリックし、選択状態にします。
値を編集し、入力欄に以下の値を設定します。
   - 設定値 -「field is required.」



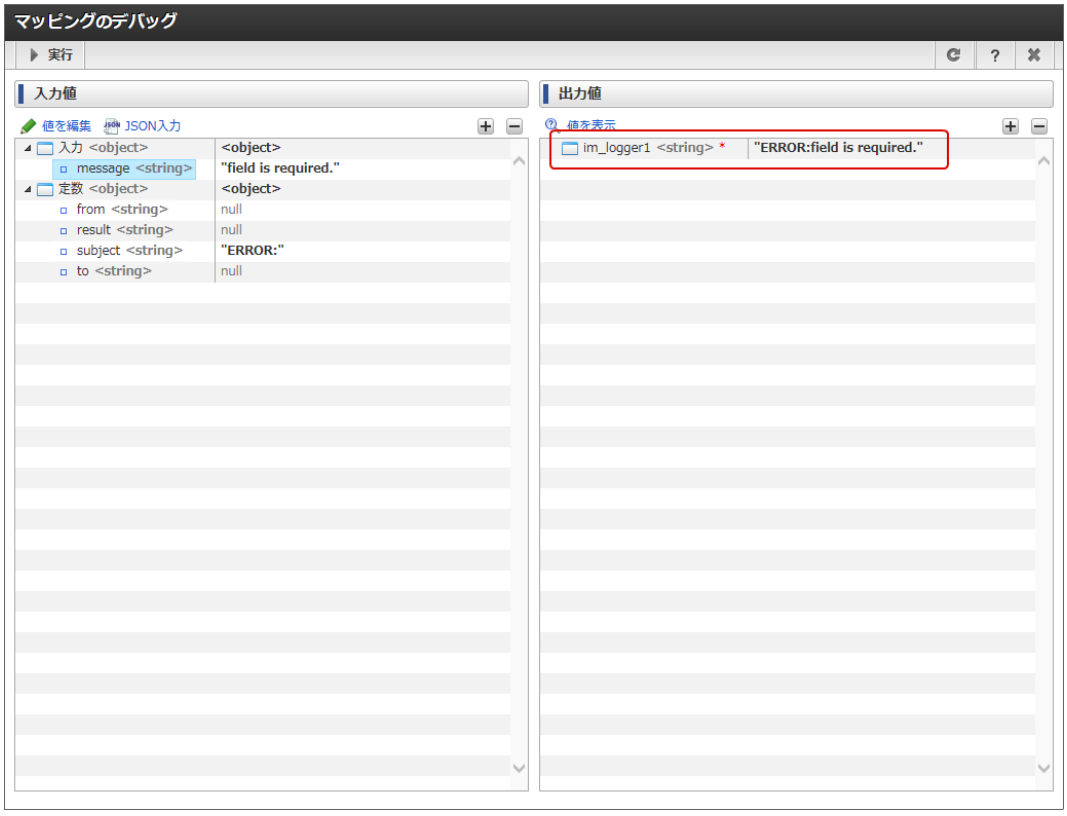

図：デバッグ用設定値の入力

8. ツールバーの「実行」をクリックします。




図：設定の反映と実行

9. デバッグの開始を確認するダイアログが表示されるので、「決定」をクリックします。
10. マッピングの実行が完了した旨のメッセージが表示され、実行結果が「出力値」に反映されます。




図：デバッグ実行の完了

以上で、マッピングのデバッグ実行が完了しました。
デバッグ実行した結果、出力値が期待する値になることを確認します。

- 出力値 - 「ERROR:field is required.」

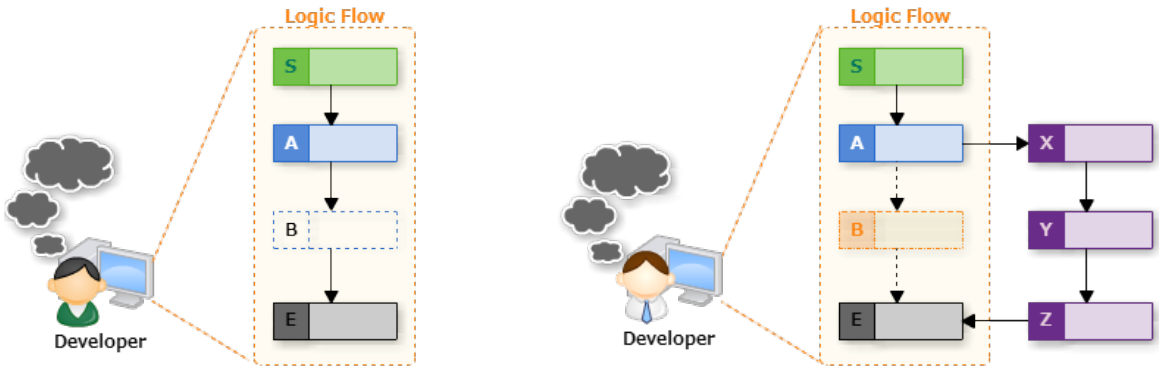
## ユーザ定義の作成

この章では、IM-LogicDesignerが標準で提供しているタスクとは別に、独自に処理内容を定義可能なユーザ定義（ユーザ定義タスク）について説明します。
開発者はユーザ定義を利用することで、直面している要件や課題に対してよりスマートなフローを作成することができます。

## ユーザ定義（ユーザ定義タスク）とは

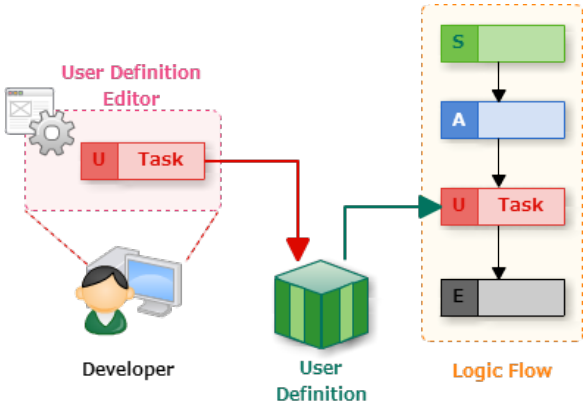
IM-LogicDesignerでは、intra-mart Accel Platform、および、その上で動作する機能群に関するタスクを数多く提供しています。
しかしながら、解決したい課題や要件によっては、以下のようにフローの作成がスムーズに行えない場合があります。

- 必要となる処理に適したタスクが提供されていない。
- 必要となる処理を実現するためには、タスクの冗長な利用をしなければならない。




左図：必要な処理タスクBが標準で提供されていない。右図：必要な処理タスクBの代替タスク（X、Y、Z）は存在するが冗長である。

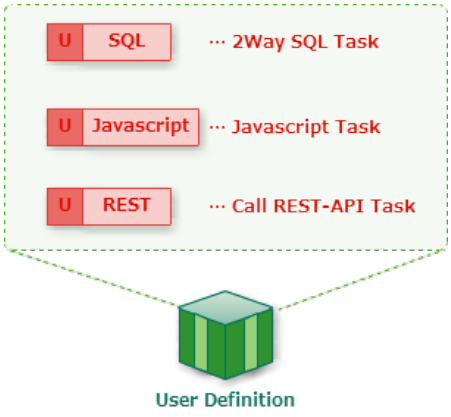
そうした場合に対し、ユーザが独自に直接処理を定義可能なユーザ定義（ユーザ定義タスク）をIM-LogicDesignerでは提供しています。
ユーザ定義を利用することで、標準で提供されているタスク以外の複雑な処理を実現することができます。




図：目的にあったタスクを独自に定義し、フローに組み込める

ユーザ定義の処理定義には、作成したい処理に応じて以下の三通りの方法が用意されています。

- SQL(2WaySQL)
- JavaScript
- REST




図：三通りのユーザ定義実装方法

この章ではユーザ定義について、上述した三通りの定義方法（SQL, JavaScript, REST）についての詳細と、ユーザ定義の利用方法について説明します。

次章「ユーザカテゴリ」では、ユーザ定義を分類するユーザカテゴリの作成方法について説明します。

## ユーザカテゴリ

この章では、ユーザ定義の分類情報として、フローカテゴリを作成方法を説明します。

> **コラム**
>
> ユーザカテゴリの詳細について
>
> ユーザカテゴリの詳細は「IM-LogicDesigner仕様書」を参照してください。

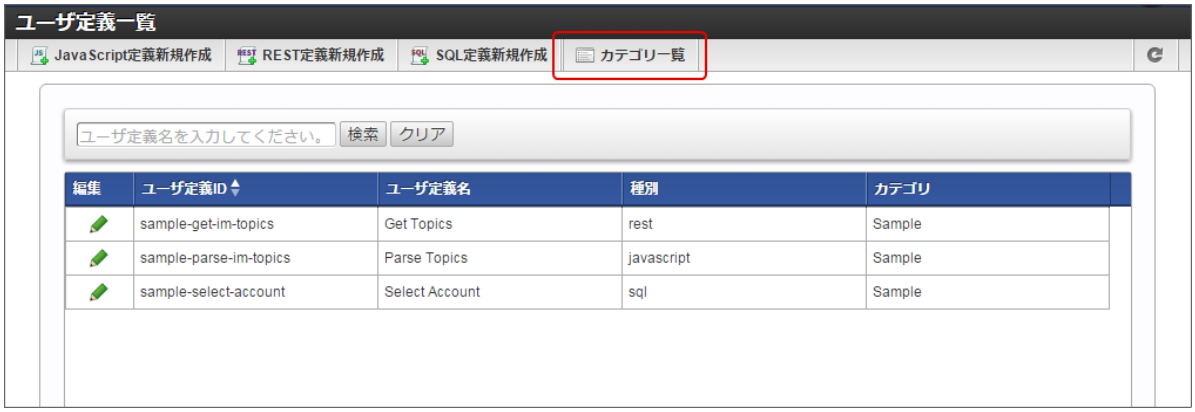
- ユーザカテゴリー覧画面を表示する
- ユーザカテゴリを新規作成する
- 作成したユーザカテゴリを確認する

### ユーザカテゴリー覧画面を表示する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義一覧」から、ユーザ定義一覧を開きます。

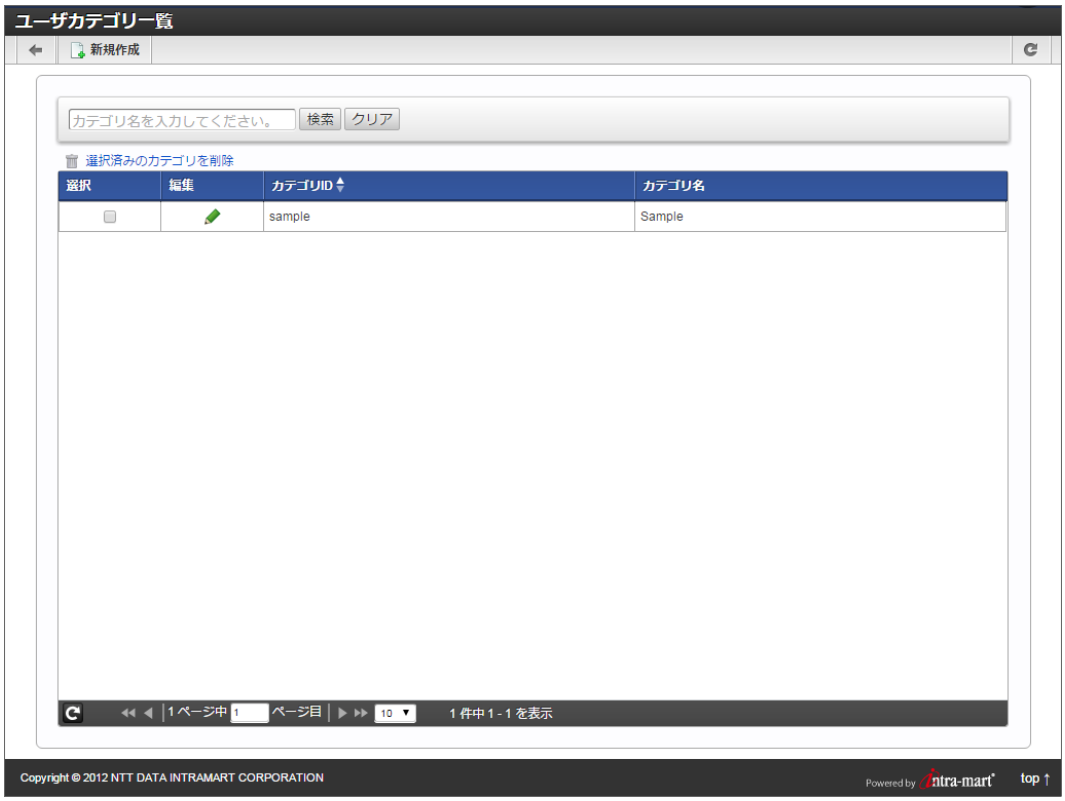

図：サイトマップ

2. 画面上部のヘッダ内、「カテゴリー覧」をクリックします。




図：ユーザ定義一覧画面

3. ユーザカテゴリー覧画面が表示されます。

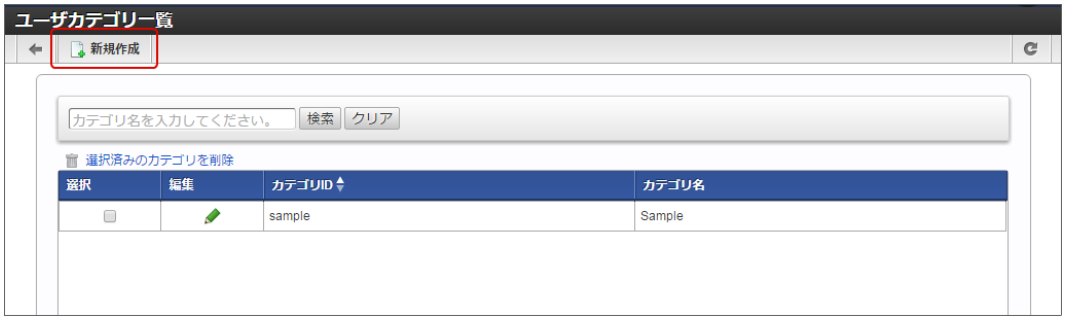



図：ユーザカテゴリー覧画面

## ユーザカテゴリを新規作成する

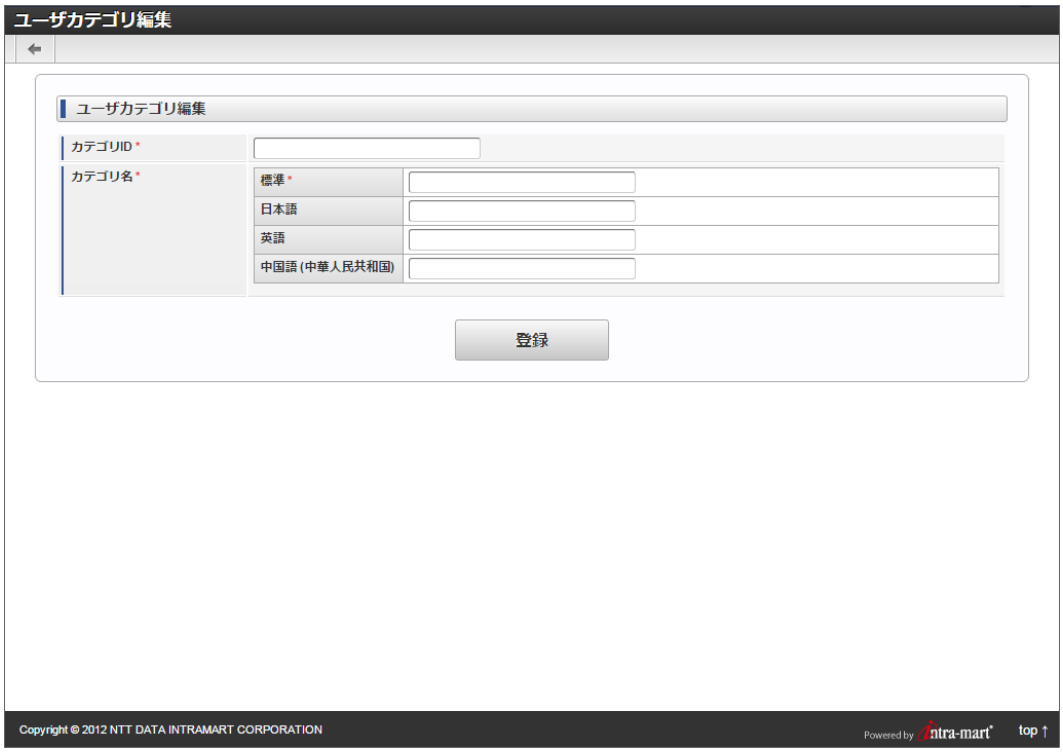
本チュートリアルで利用するユーザカテゴリを作成します。
ユーザカテゴリの作成は、ユーザカテゴリ編集画面から行います。

1. ユーザカテゴリー覧画面左上の「新規作成」をクリックします。



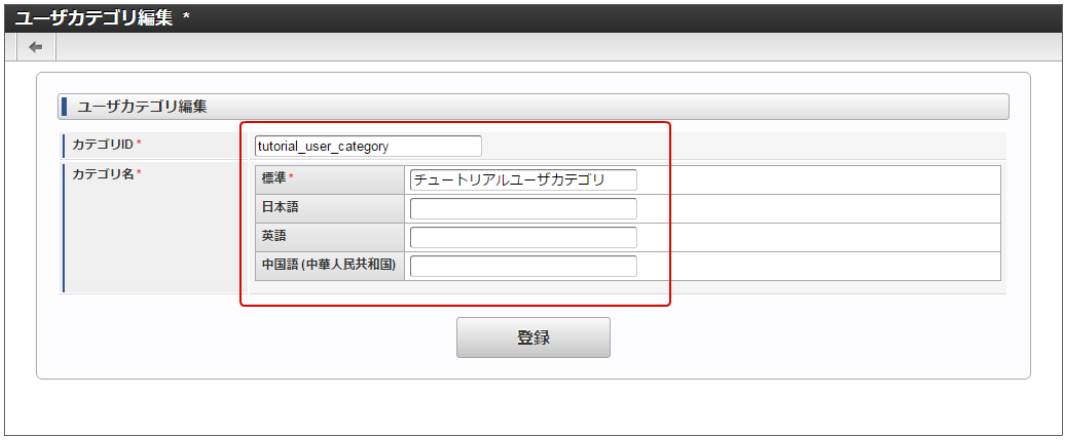

図：ユーザカテゴリー覧画面 - 新規作成

2. ユーザカテゴリ編集画面が表示されます。




図：ユーザカテゴリ編集画面

3. ユーザカテゴリ情報の各項目に以下の値を入力します。
   - フローカテゴリID「tutorial_user_category」
   - フローカテゴリ名
     - 標準 -「チュートリアルユーザカテゴリ」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし




図：ユーザカテゴリ情報の入力

4. 「登録」をクリックします。

以上で、ユーザカテゴリの作成が完了しました。

## 作成したユーザカテゴリを確認する

正常にユーザカテゴリが新規作成された場合、正常に作成された旨のメッセージと共に、ユーザカテゴリー覧画面へ遷移します。
ユーザカテゴリー覧画面から、作成したユーザカテゴリが確認できます。


図：ユーザカテゴリ作成後の一覧画面

次章から三章に渡り、ユーザ定義の定義方法を説明します。

- 「*ユーザ定義 - SQL(2WaySQL)*」では、SQL（2WaySQL）を利用した定義方法について説明します。
- 「*ユーザ定義 - JavaScript*」では、JavaScriptを利用した定義方法について説明します。
- 「*ユーザ定義 - REST*」では、REST APIを利用した定義方法について説明します。

## ユーザ定義 - SQL(2WaySQL)

### ユーザ定義（SQL）

ユーザ定義(SQL）とは、SQLを利用してタスクの処理内容を定義することが可能なユーザ定義の一つです。
タスクの入力値と2Way-SQLを組み合わせた柔軟な処理定義が行え、出力値として実行結果を返します。

> **コラム**
>
> 2Way-SQLとは
>
> ユーザ定義（SQL）では、2Way-SQLに対応しています。
> 2Way-SQLとはコメントとしてマッピングを定義することで、コメント内の定義情報を元としたプログラムで利用するテンプレートとしての利用、および、SQL*Plusなどのツールで実行の双方を両立したSQL文です。
>
> 2Way-SQLの詳細は以下をリンクを参照してください。
>
> - S2Dao - Top
> - Seasar2 - S2JDBC - JdbcManager - SQLファイルによる操作

#### データベース種別とクエリ種別

ユーザ定義（SQL）では、以下のデータベースへの問い合わせをサポートしています。

- テナントデータベース
- シェアードデータベース

また、クエリの種別として以下の構文をサポートしています。

- SELECT
- INSERT
- UPDATE
- DELETE

それぞれの定義方法については、次章以降の各構文に対応するチュートリアルを参照してください。

#### ユーザ定義（SQL）作成画面への遷移手順

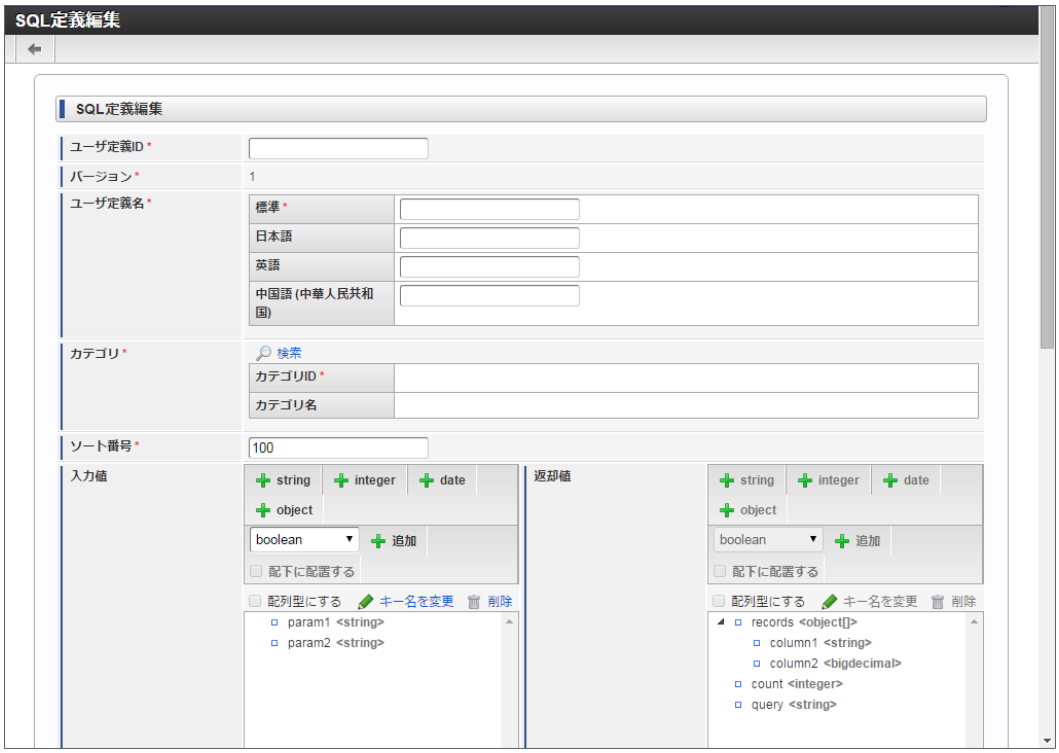
ユーザ定義（SQL）作成画面へは、ユーザ定義一覧画面から遷移します。

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義一覧」から、ユーザ定義一覧を開きます。

2. 画面上部のヘッダ内、「SQL定義新規作成」をクリックします。


図：ユーザ定義一覧 - SQL定義新規作成

3. 「SQL定義編集」画面が表示されます。




図：SQL定義編集画面

## SELECTを用いたユーザ定義（SQL）の作成

この章では、SELECT文を利用したユーザ定義の作成方法とその詳細について説明します。

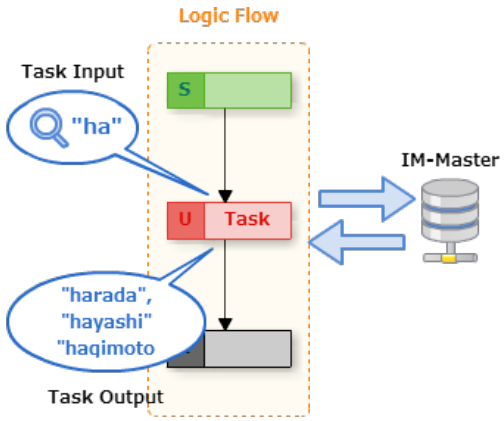


### 本チュートリアルで作成するユーザ定義の概要

本チュートリアルでは、

**「IM-共通マスタ上のユーザのうち、タスクの入力値として与えられた半角英数で始まるユーザコードを持つユーザを取得する」**

というタスクの処理の実装を通して、SELECT文を用いたユーザ定義の作成方法とその詳細を説明します。
作成するユーザ定義の処理イメージは以下の通りです。




図： 処理イメージ

図の例では、入力値として「ha」を与えた場合、IM-共通マスタのサンプルユーザ内で該当する以下のユーザ情報（ユーザコード）が出力値として取得されます

- harada
- hagimoto
- hayashi

なお、今回の取得処理では処理内容の簡略化のため、期間情報や多言語情報を加味しません。

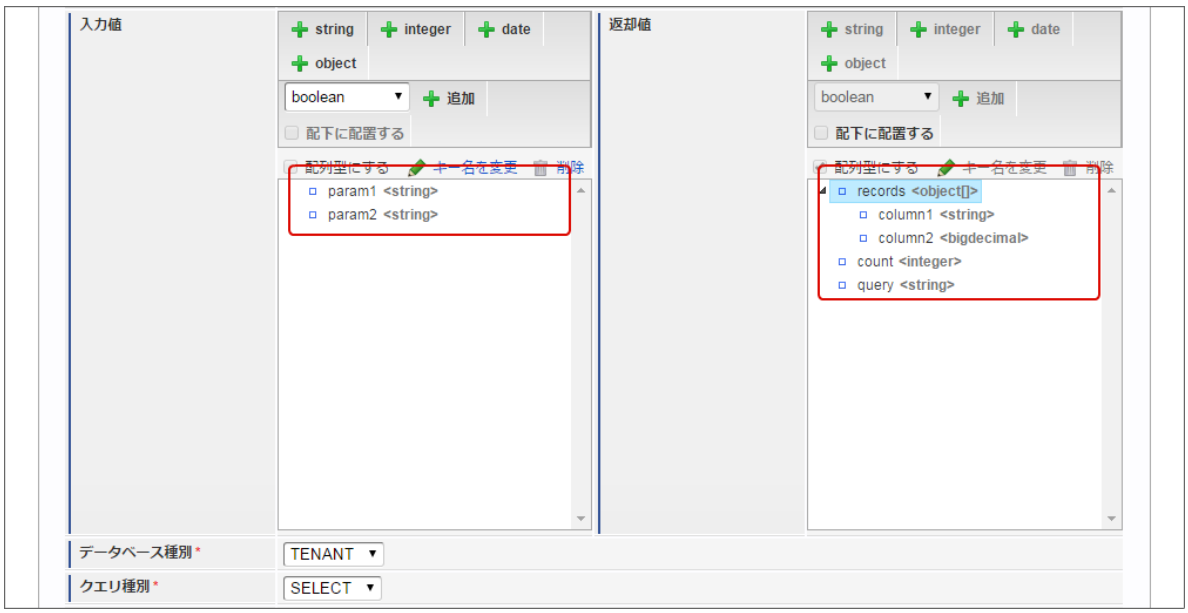
### データベース種別とクエリ種別

はじめに、今回作成するユーザ定義（SQL）が対象とするデータベース種別、および、クエリ種別を設定します。
本チュートリアルでは以下の値を設定します。

- データ種別 - 「TENANT」
- クエリ種別 - 「SELECT」

### 入力値/出力値

次に、作成するユーザ定義を呼び出す際の入力値と、呼び出しが完了した際に返却する出力値を定義します。
ユーザ定義（SQL）の作成において入力値/出力値は、クエリ種別を選択したタイミングで自動的に初期値が設定されます。
クエリ種別「SELECT」設定時の入力値/出力値の初期値は以下の通りです。




図：クエリ種別「SELECT」選択時の入力値/出力値の初期値

ここで留意すべき点は、出力値がシステムによって固定であることです。
IM-LogicDesignerではクエリ種別、および、後述するクエリ設定の内容を元に出力値を自動で決定します。
クエリ種別「SELECT」選択時に設定される出力値の詳細は以下の通りです。

| 出力値 | 説明 |
|---|---|

| 出力値 | 説明 |
| --- | --- |
| records<object[]> | 定義したSELECT文によって取得された値が格納されます。<br>格納される値の詳細は「*データ定義取得と、出力値への反映*」によって決定されます。 |
| count<integer> | 取得された値の総数が格納されます。 |
| query<string> | 実行されたクエリが格納されます。 |

なお出力値とは違い、入力値は開発者が決定する必要があります。
本チュートリアルでは、入力値として「ユーザ検索用の文字列」を以下のように定義します。

| 入力値 | 説明 |
| --- | --- |
| search_word<string> | ユーザ検索用の文字列を格納します。 |

### クエリ設定

次に、実際に取得処理を行うクエリ（SELECT文）の設定を行います。

想定するSELECT文

本チュートリアルで想定する処理は、以下のSQLによって表現されます。

```
SELECT
  user_cd
FROM
  imm_user
/*BEGIN*/
WHERE
  /*IF search_word != null*/
  user_cd LIKE /*search_word*/'ha%'
  /*END*/
/*END*/
```

コラム

重複を除去する場合

今回利用するSELECT文では、多言語情報や期間情報を加味しないため、一人の該当ユーザに対して複数の結果（ユーザコード）が返る場合があります。
本チュートリアルのユーザ定義を実際に作成、動作確認を行う場合、必要に応じてdistinctなどの重複を除外する指定を行ってください。

以下は指定例です。

```
SELECT
  distinct user_cd -- distinctの指定
FROM
/* 以降は同様 */
```

データ定義取得と、出力値への反映

「*想定するSELECT文*」の定義に合わせて、出力値の設定を行います。
IM-LogicDesignerでは、設定されたクエリを検証し、クエリの記述内容にあった出力値を自動で設定する機能を提供しています。

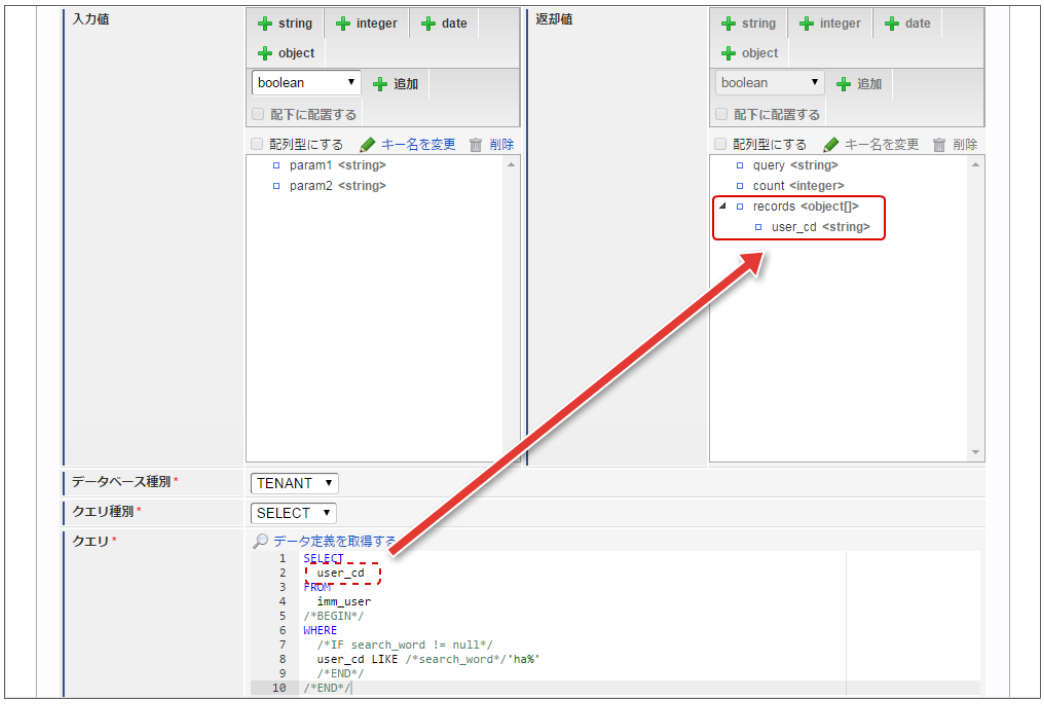
1. 「SQL定義編集」画面の「クエリ」にSQLを定義します。


図：クエリへSELECT文の定義

2. 「クエリ」入力欄の上部にある「データ定義を取得する」をクリックします。


図：データ定義を取得

3. 「クエリ」に定義されたSQLの検証結果が、「出力値」に反映されます。




図：クエリをもとに出力値へ反映

**注意**

出力値を反映される場合の注意点

定義されたSQLがシンタックスエラー等実行不可なものであった場合、以下のエラーメッセージが表示され、出力値への反映は行われません。


図：SQLが実行不可であった場合のエラーメッセージ

**コラム**

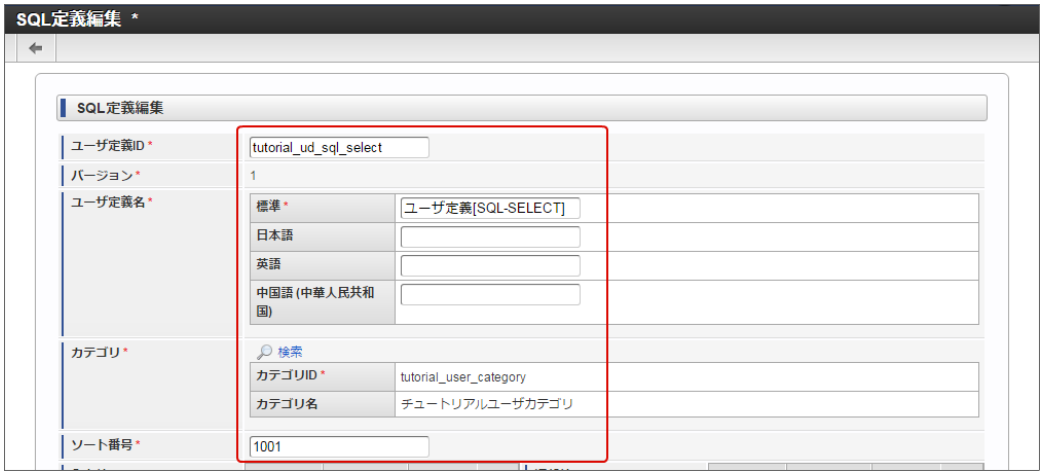
ワイルドカード（アスタリスク）使用時の出力値

定義されたSQLが、取得対象にワイルドカード（アスタリスク）を利用した場合、IM-LogicDesignerは出力値として対象テーブルに定義された全てのカラムを出力値対象とします。

## ユーザ定義（SQL）を作成する。

最後に、これまでの内容を踏まえてユーザ定義（SQL）を作成します。

1. 「SQL定義編集」画面を表示します。
2. ユーザ定義の基本情報となる各項目に以下の値を入力します。
   - ユーザ定義ID「tutorial_ud_sql_select」
   - バージョン「1」（固定）
   - ユーザ定義名
     - 標準 - 「ユーザ定義[SQL-SELECT]」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし
   - カテゴリ
     - カテゴリID - 「tutorial_user_category」
   - ソート番号「1001」

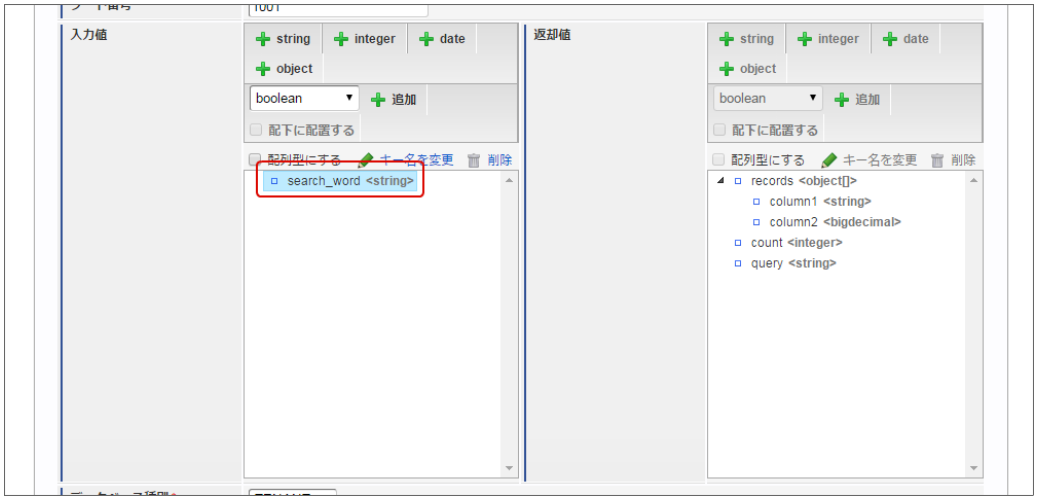



図：基本情報の定義

3. データベース種別とクエリ種別を「*データベース種別とクエリ種別*」をもとに設定します。


図：データベース種別、および、クエリ種別の定義

4. 入力値を「*入力値/出力値*」をもとに値を設定します。

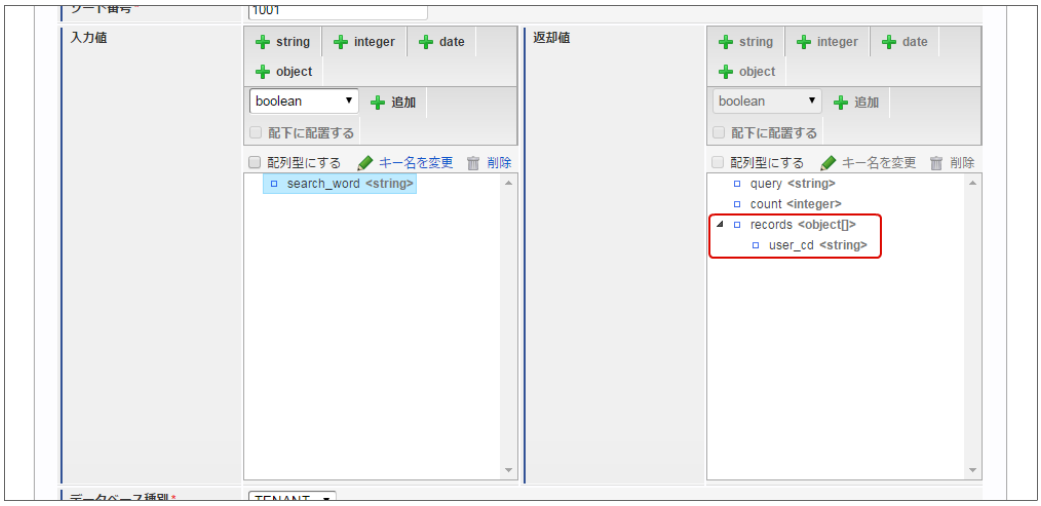



図：入力値の定義

5. クエリに「*想定するSELECT文*」で提示したSQLを設定します。


図：クエリの定義

6. 「*データ定義取得と、出力値への反映*」をもとに、出力値を設定します。




図：出力値の反映

7. 「登録」をクリックします。


図：登録

以上で、SELECTを用いたユーザ定義（SQL）の作成が完了しました。

### 応用：取得範囲を指定したユーザ定義（SQL）の作成

ここでは本チュートリアルで作成したユーザ定義（SQL）をベースとして、取得範囲を指定する方法を説明します。

取得範囲の指定

本チュートリアルで作成したユーザ定義は、検索条件に合致するユーザを全て取得します。
取得処理には問題はありませんが、実際の要件や課題の解決において検索条件に合致するユーザの一部を取得したい場合が多く存在します。
ユーザ定義（SQL）では、これに対して取得範囲を指定したSQLの定義をサポートしています。

取得範囲の指定を行うSQLを定義するには、SQL定義編集の項目を以下のとおりに設定します。

- 範囲を指定する - チェックボックス：オン


図：取得範囲の設定

取得範囲の指定を行うことによって、ユーザ定義（SQL）は自動的に以下の設定を行います。

- 「入力値」へ、範囲を指定するために必要なパラメータの追加
- 「クエリ」に定義されたSQLへ、取得範囲を指定する構文の追加

範囲指定を行うパラメータの詳細は「IM-LogicDesigner仕様書」 - 「ユーザ定義タスク」 - 「SQL(2WaySQL)」を参照してください。

> **注意**
>
> 追加される取得範囲を指定する構文について
>
> 取得範囲の指定を行うことによって追加される構文は、暗黙的に行われます。
> ユーザ定義（SQL）は取得範囲の指定がされていた場合、定義された「クエリ」に取得範囲の指定を行う構文を追加したうえで最終的な処理を行います。
>
> そのため、取得範囲の指定を行った場合でも、「SQL定義編集」画面の「クエリ」に定義されたSQLは変更されません。
> また、「クエリ」に直接取得範囲の指定を行う構文を定義していた場合はSQL不正でエラーが発生します。

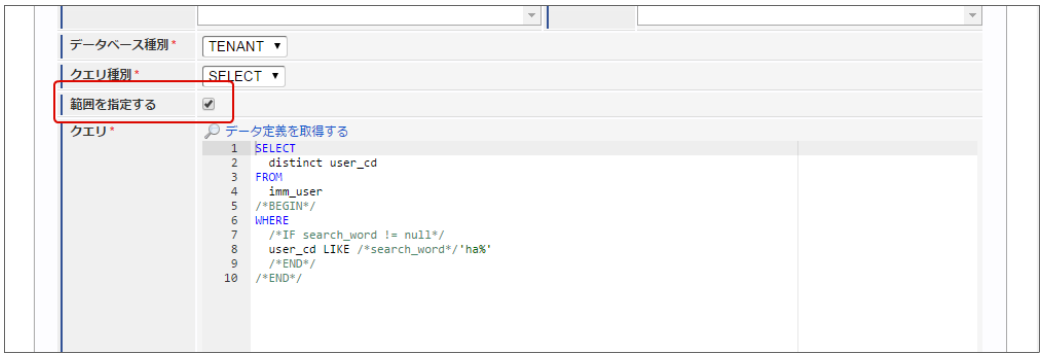
範囲指定を元に、ユーザ定義（SQL）を更新する。

上記の内容を踏まえてユーザ定義（SQL）を更新します。

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義一覧」から、ユーザ定義一覧を開きます。
   一覧の中から、ユーザ定義ID「tutorial_ud_sql_select」の行の編集アイコンをクリックします。

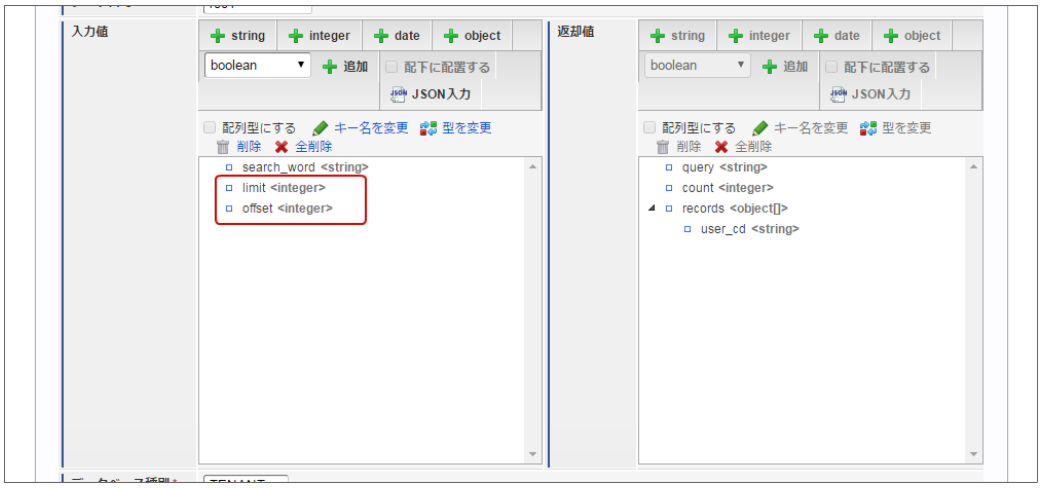

   図：「tutorial_ud_sql_select」編集アイコン

2. 取得範囲を指定するため、「*取得範囲の指定*」をもとに設定します。




図：取得範囲の設定

3. 入力値に以下のパラメータが自動で追加されることを確認してください。
   - 取得数（限定度） - limit<integer>
   - 取得位置 - offset<integer>




図：追加されたパラメータ

4. 「新バージョン登録」をクリックします。


図：新バージョン登録

以上で、取得範囲を指定したユーザ定義（SQL）の作成が完了しました。

## INSERTを用いたユーザ定義（SQL）の作成

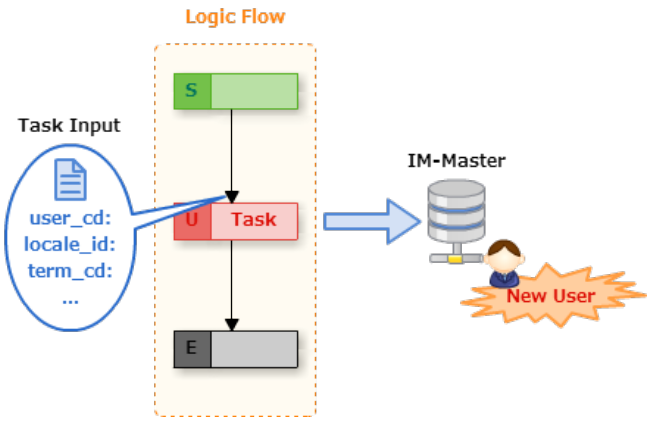
この章では、INSERT文を利用したユーザ定義の作成方法とその詳細について説明します。

## 本チュートリアルで作成するユーザ定義の概要

本チュートリアルでは、

**「IM-共通マスタのユーザテーブル(imm_user）へ、タスクの入力値として与えられた情報を元に新しく情報を追加する」**

というタスクの処理の実装を通して、INSERT文を用いたユーザ定義の作成方法とその詳細を説明します。
作成するユーザ定義の処理イメージは以下の通りです。




図：処理イメージ

なお、本チュートリアルでは簡略化のため、ユーザテーブル(imm_user）のうち必須（NOT NULL）である値のみを登録するものとしています。

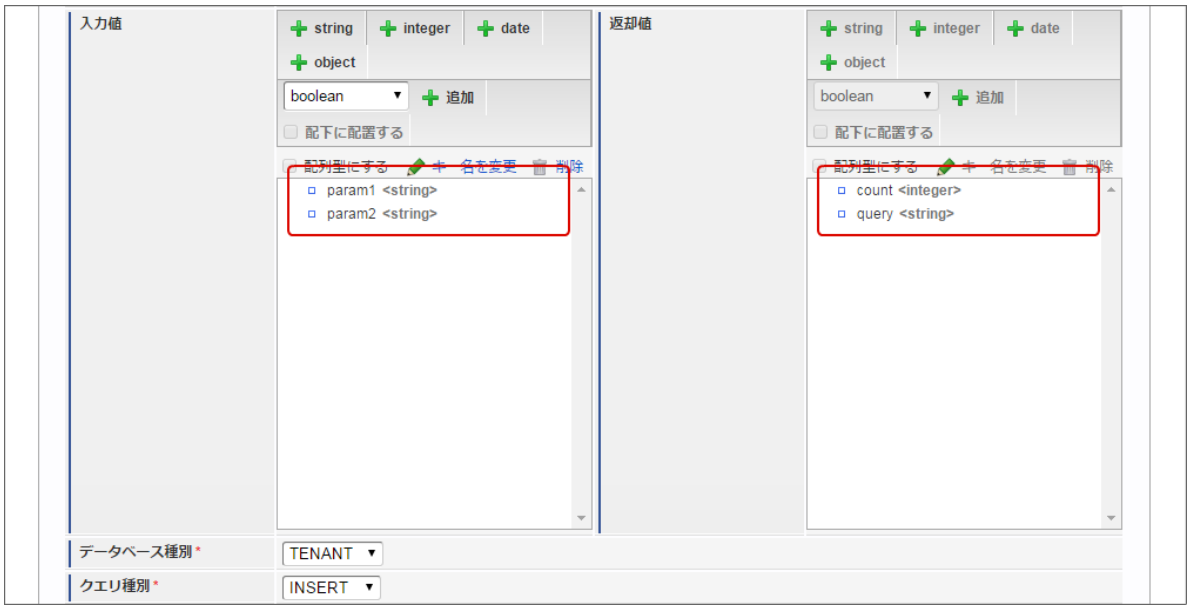
## データベース種別とクエリ種別

はじめに、今回作成するユーザ定義（SQL）が対象とするデータベース種別、および、クエリ種別を設定します。
本チュートリアルでは以下の値を設定します。

- データ種別 - 「TENANT」
- クエリ種別 - 「INSERT」

## 入力値/出力値

次に、作成するユーザ定義を呼び出す際の入力値と、呼び出しが完了した際に返却する出力値を定義します。
ユーザ定義（SQL）の作成において入力値/出力値は、クエリ種別を選択したタイミングで自動的に初期値が設定されます。
クエリ種別「INSERT」設定時の入力値/出力値の初期値は以下の通りです。



図：クエリ種別「INSERT」選択時の入力値/出力値の初期値

ここで留意すべき点は、出力値がシステムによって固定であることです。
IM-LogicDesignerではクエリ種別を元に出力値を自動で決定します。
クエリ種別「INSERT」選択時に設定される出力値の詳細は以下の通りです。

| 出力値 | 説明 |
| --- | --- |
| count<integer> | 処理総数が格納されます。 |
| query<string> | 実行されたクエリが格納されます。 |

なお出力値とは違い、入力値は開発者が決定する必要があります。
本チュートリアルでは、入力値として「ユーザテーブル(imm_user）において必須(NOT NULL）である値」をもとに定義します。

| 入力値 | 説明 |
| --- | --- |
| user_cd<string> | ユーザコード |
| locale_id<string> | ロケールID |
| term_cd<string> | 期間コード |
| start_date<sqldate> | 開始日 |
| end_date<sqldate> | 終了日 |
| user_name<string> | ユーザ名 |
| delete_flag<string> | 削除フラグ |
| sort_key<integer> | ソートキー |
| create_user_cd<string> | 作成者 |
| create_date<sqltimestamp> | 作成日 |
| record_user_cd<string> | 最終更新者 |
| record_date<sqltimestamp> | 最終更新日 |

### クエリ設定

次に、実際に追加処理を行うクエリ（INSERT文）の設定を行います。

想定するINSERT文

本チュートリアルで想定する処理は、以下のSQLによって表現されます。

```
INSERT INTO imm_user (
  user_cd,
  locale_id,
  term_cd,
  start_date,
  end_date,
  user_name,
  delete_flag,
  sort_key,
  create_user_cd,
  create_date,
  record_user_cd,
  record_date
) VALUES (
  /*user_cd*/'tutorial',
  /*locale_id*/'ja',
  /*term_cd*/'tutorial_term',
  /*start_date*/'2000-01-01',
  /*end_date*/'3000-01-01',
  /*user_name*/'チュートリアル',
  /*delete_flag*/0,
  /*sort_key*/100,
  /*create_user_cd*/'tenant',
  /*create_date*/'2016-01-31 00:00:00',
  /*record_user_cd*/'tenant',
  /*record_date*/'2016-01-31 00:00:00'
)
```

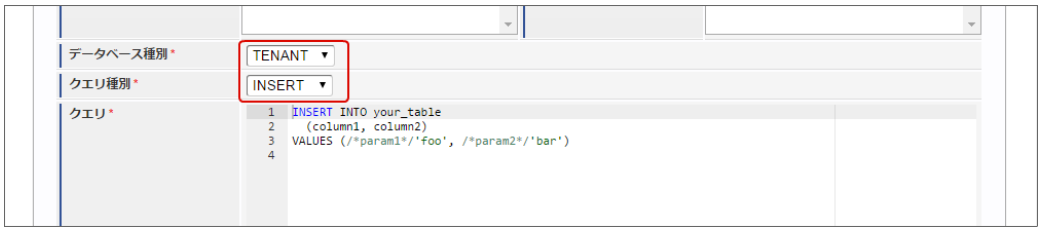
## ユーザ定義（SQL）を作成する。

最後に、これまでの内容を踏まえてユーザ定義（SQL）を作成します。

1. 「SQL定義編集」画面を表示します。
2. ユーザ定義の基本情報となる各項目に以下の値を入力します。
   - ユーザ定義ID「tutorial_ud_sql_insert」
   - バージョン「1」（固定）
   - ユーザ定義名
     - 標準 - 「ユーザ定義[SQL-INSERT]」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし
   - カテゴリ
     - カテゴリID - 「tutorial_user_category」
   - ソート番号「1002」

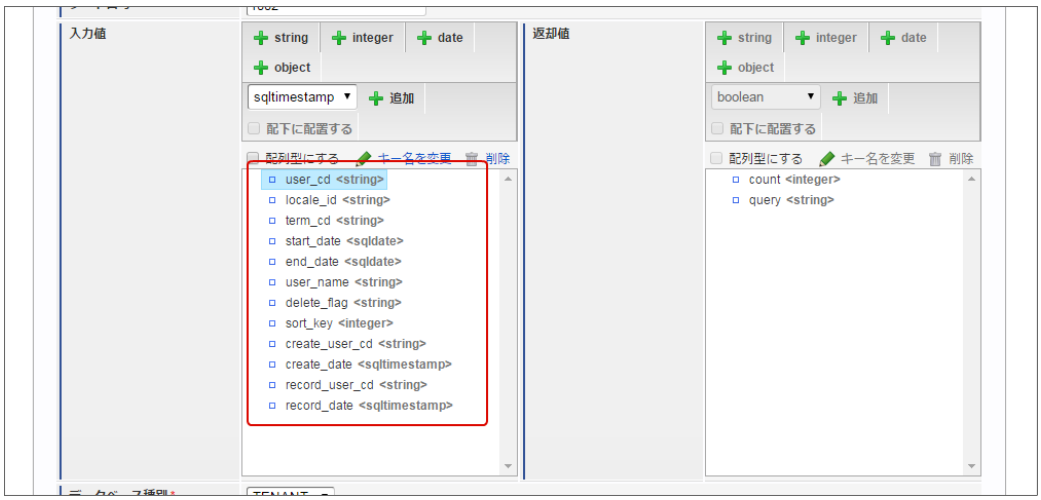

図：基本情報の定義

3. データベース種別とクエリ種別を「*データベース種別とクエリ種別*」をもとに設定します。




図：データベース種別、および、クエリ種別の定義

4. 入力値を「*入力値/出力値*」をもとに値を設定します。




図：入力値の定義

5. クエリに「*想定するINSERT文*」で提示したSQLを設定します。


図：クエリの定義

6. 「登録」をクリックします。


図：登録

以上で、INSERTを用いたユーザ定義（SQL）の作成が完了しました。

## UPDATEを用いたユーザ定義（SQL）の作成

この章では、UPDATE文を利用したユーザ定義の作成方法とその詳細について説明します。

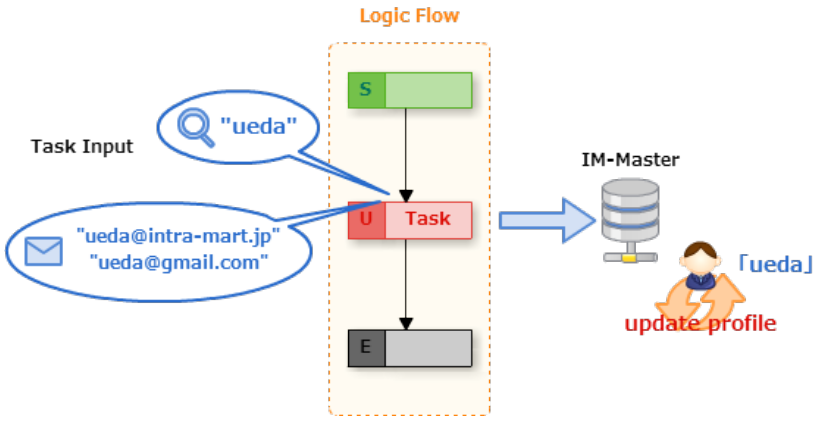


### 本チュートリアルで作成するユーザ定義の概要

本チュートリアルでは、

**「IM-共通マスタ上のユーザのうち、タスクの入力値として指定されたユーザコードを持つユーザのメールアドレスを、同じく入力値として指定されたメールアドレスで更新する」**

というタスクの処理の実装を通して、UPDATE文を用いたユーザ定義の作成方法とその詳細を説明します。
作成するユーザ定義の処理イメージは以下の通りです。




図：処理イメージ

本チュートリアルの具体的な処理として、入力値のユーザコードは完全一致とし、メールアドレスは最大二つ指定可能としています。
また、今回の更新処理では処理内容の簡略化のため、期間情報や多言語情報を加味しません。

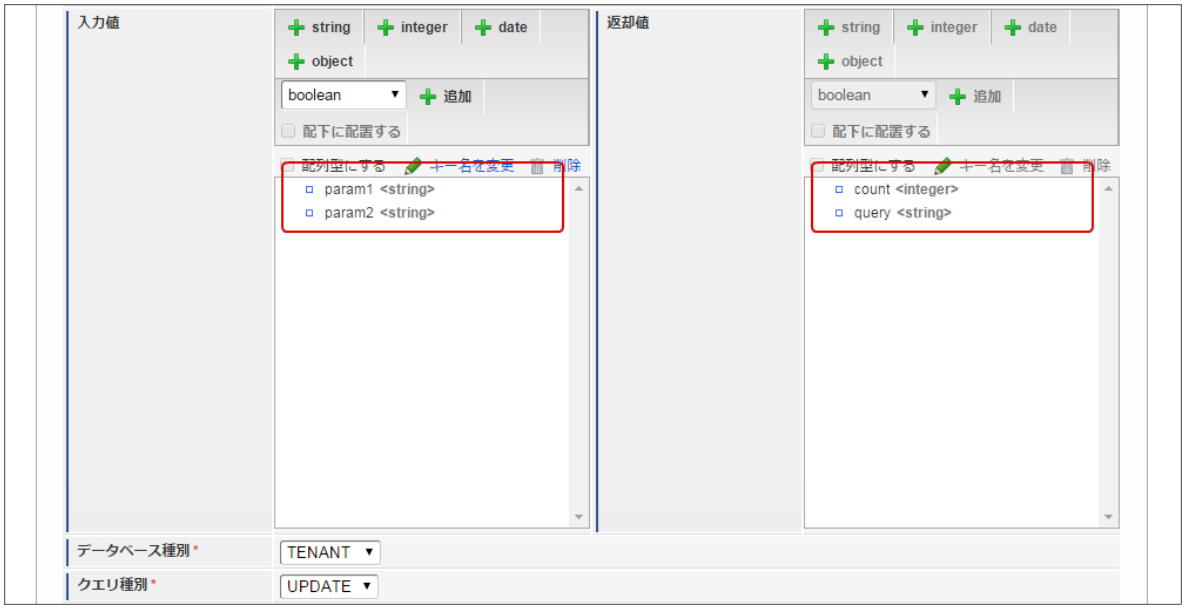
### データベース種別とクエリ種別

はじめに、今回作成するユーザ定義（SQL）が対象とするデータベース種別、および、クエリ種別を設定します。
本チュートリアルでは以下の値を設定します。

- データ種別 - 「TENANT」
- クエリ種別 - 「UPDATE」

### 入力値/出力値

次に、作成するユーザ定義を呼び出す際の入力値と、呼び出しが完了した際に返却する出力値を定義します。
ユーザ定義（SQL）の作成において入力値/出力値は、クエリ種別を選択したタイミングで自動的に初期値が設定されます。
クエリ種別「UPDATE」設定時の入力値/出力値の初期値は以下の通りです。




図：クエリ種別「UPDATE」選択時の入力値/出力値の初期値

ここで留意すべき点は、出力値がシステムによって固定であることです。
IM-LogicDesignerではクエリ種別を元に出力値を自動で決定します。
クエリ種別「UPDATE」選択時に設定される出力値の詳細は以下の通りです。

| 出力値 | 説明 |
|---|---|
| count<integer> | 処理総数が格納されます。 |
| query<string> | 実行されたクエリが格納されます。 |

なお出力値とは違い、入力値は開発者が決定する必要があります。
本チュートリアルでは、入力値として「更新対象ユーザのユーザコード」、および、「更新メールアドレス」を以下のように定義します。

| 入力値 | 説明 |
| --- | --- |
| update_user_cd<string> | 更新対象のユーザコードを格納します。 |
| email_address1<string> | 一つ目のメールアドレス情報が格納されます。 |
| email_address2<string> | 二つ目のメールアドレス情報が格納されます。 |

### クエリ設定

次に、実際に更新処理を行うクエリ（UPDATE文）の設定を行います。

想定するUPDATE文

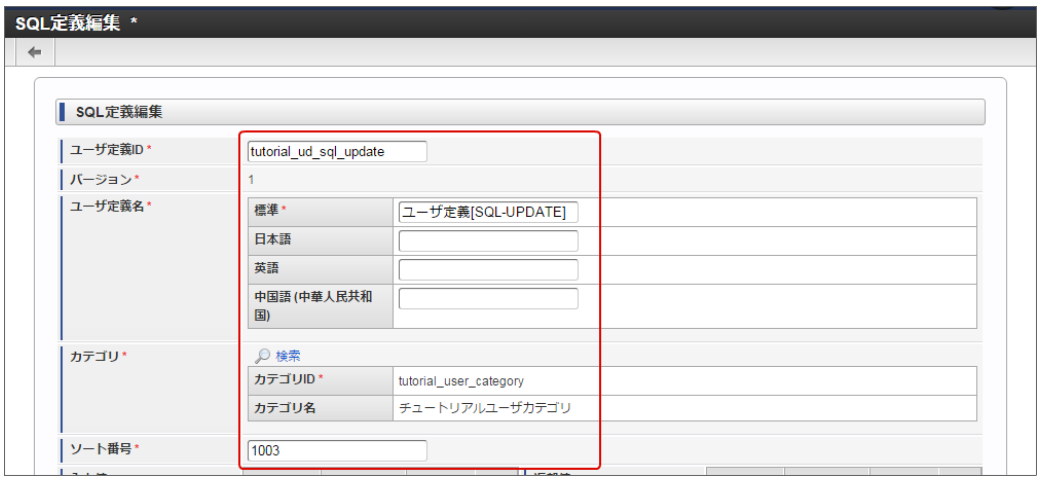
本チュートリアルで想定する処理は、以下のSQLによって表現されます。

```
UPDATE imm_user
SET email_address1 = /*email_address1*/'foo@intra-mart.jp',
   email_address2 = /*email_address2*/'bar@gmail.com'
WHERE
 user_cd = /*update_user_cd*/'tutorial'
```

### ユーザ定義（SQL）を作成する。

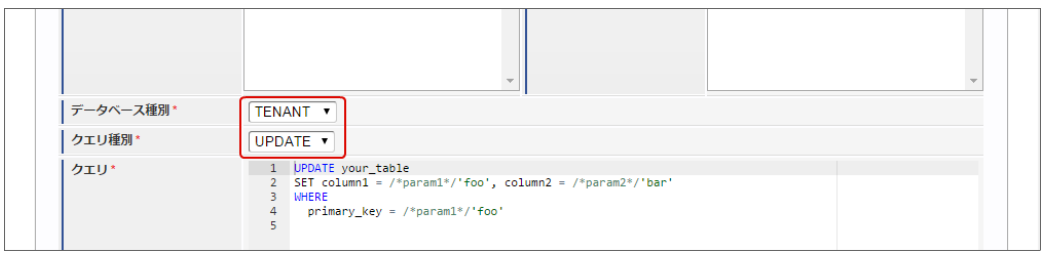
最後に、これまでの内容を踏まえてユーザ定義（SQL）を作成します。

1. 「SQL定義編集」画面を表示します。
2. ユーザ定義の基本情報となる各項目に以下の値を入力します。
   - ユーザ定義ID「tutorial_ud_sql_update」
   - バージョン「1」（固定）
   - ユーザ定義名
     - 標準 - 「ユーザ定義[SQL-UPDATE]」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし
   - カテゴリ
     - カテゴリID - 「tutorial_user_category」
   - ソート番号「1003」



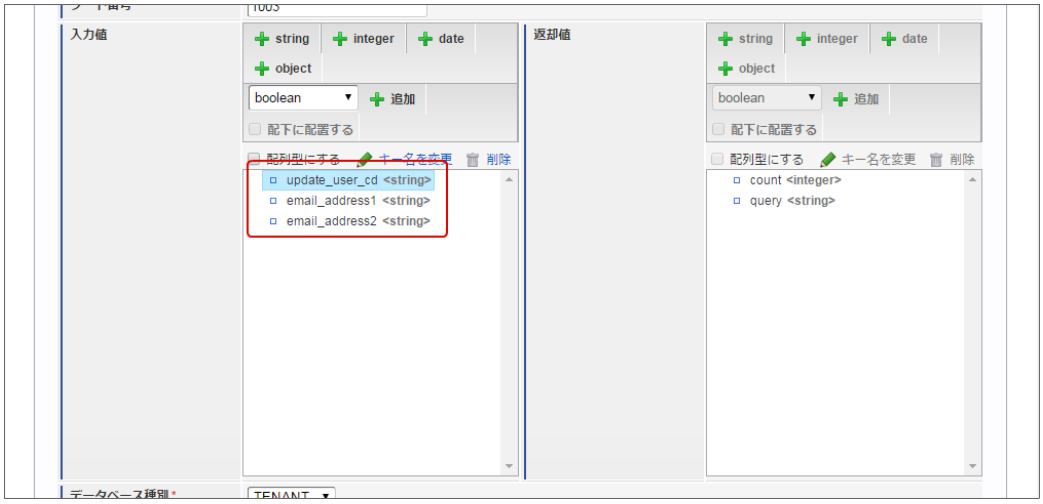

図：基本情報の定義

3. データベース種別とクエリ種別を「*データベース種別とクエリ種別*」をもとに設定します。




図：データベース種別、および、クエリ種別の定義

4. 入力値を「*入力値/出力値*」をもとに値を設定します。




図：入力値の定義

5. クエリに「*想定するUPDATE文*」で提示したSQLを設定します。


図：クエリの定義

6. 「登録」をクリックします。


図：登録

以上で、UPDATEを用いたユーザ定義（SQL）の作成が完了しました。

DELETEを用いたユーザ定義（SQL）の作成

この章では、DELETE文を利用したユーザ定義の作成方法とその詳細について説明します。

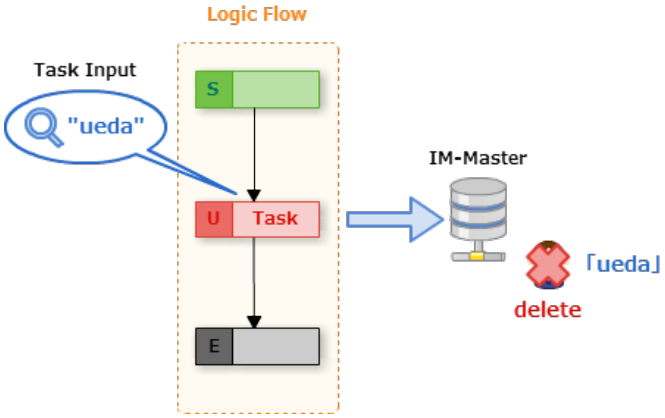


## 本チュートリアルで作成するユーザ定義の概要

本チュートリアルでは、

**「IM-共通マスタ上のユーザのうち、タスクの入力値として指定されたユーザコードを持つユーザを削除する」**

というタスクの処理の実装を通して、DELETE文を用いたユーザ定義の作成方法とその詳細を説明します。
作成するユーザ定義の処理イメージは以下の通りです。




図：処理イメージ

なお本チュートリアルでは、処理内容の簡略化のため期間情報や多言語情報を加味しません。

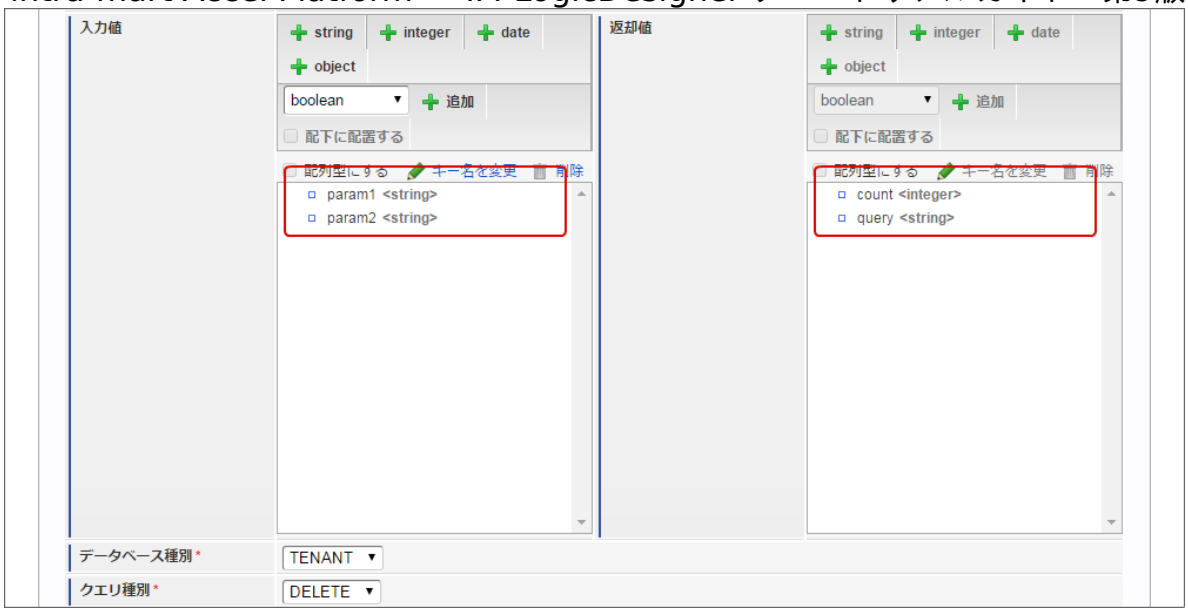
## データベース種別とクエリ種別

はじめに、今回作成するユーザ定義（SQL）が対象とするデータベース種別、および、クエリ種別を設定します。
本チュートリアルでは以下の値を設定します。

- データ種別 - 「TENANT」
- クエリ種別 - 「DELETE」

## 入力値/出力値

次に、作成するユーザ定義を呼び出す際の入力値と、呼び出しが完了した際に返却する出力値を定義します。
ユーザ定義（SQL）の作成において入力値/出力値は、クエリ種別を選択したタイミングで自動的に初期値が設定されます。
クエリ種別「DELETE」設定時の入力値/出力値の初期値は以下の通りです。




図：クエリ種別「DELETE」選択時の入力値/出力値の初期値

ここで留意すべき点は、出力値がシステムによって固定であることです。
IM-LogicDesignerではクエリ種別を元に出力値を自動で決定します。
クエリ種別「DELETE」選択時に設定される出力値の詳細は以下の通りです。

| 出力値 | 説明 |
|---|---|
| count<integer> | 処理総数が格納されます。 |
| query<string> | 実行されたクエリが格納されます。 |

なお出力値とは違い、入力値は開発者が決定する必要があります。
本チュートリアルでは、入力値として「削除対象ユーザのユーザコード」を以下のように定義します。

| 入力値 | 説明 |
|---|---|
| delete_user_cd<string> | 削除対象ユーザのユーザコードを格納します。 |

### クエリ設定

次に、実際に削除処理を行うクエリ（DELETE文）の設定を行います。

想定するDELETE文

本チュートリアルで想定する処理は、以下のSQLによって表現されます。

```
DELETE FROM imm_user
WHERE
   user_cd = /*delete_user_cd*/'tutorial'
```

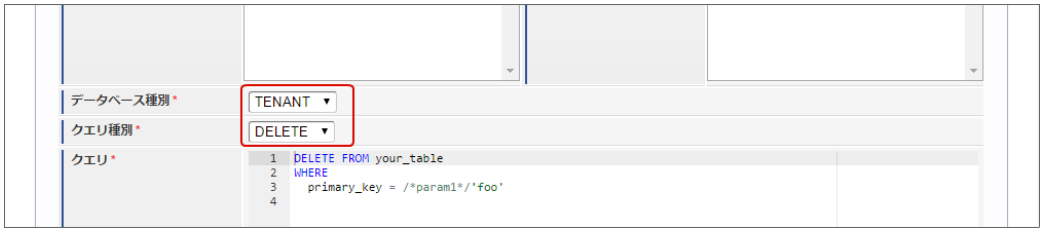
### ユーザ定義（SQL）を作成する。

最後に、これまでの内容を踏まえてユーザ定義（SQL）を作成します。

1. 「SQL定義編集」画面を表示します。
2. ユーザ定義の基本情報となる各項目に以下の値を入力します。
   - ユーザ定義ID「tutorial_ud_sql_delete」
   - バージョン「1」（固定）
   - ユーザ定義名
     - 標準 - 「ユーザ定義[SQL-DELETE]」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし
   - カテゴリ
     - カテゴリID - 「tutorial_user_category」
   - ソート番号「1004」

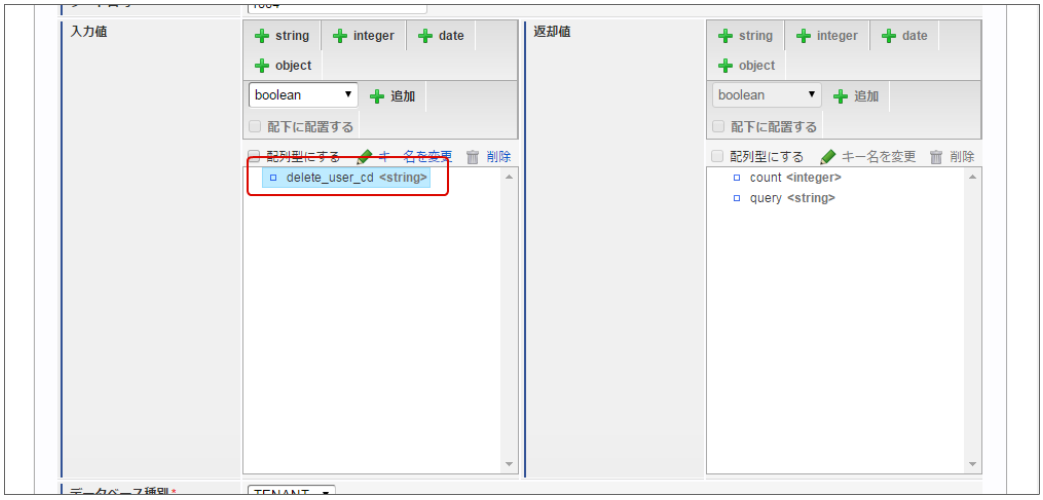

図：基礎情報の定義

3. データベース種別とクエリ種別を「*データベース種別とクエリ種別*」をもとに設定します。



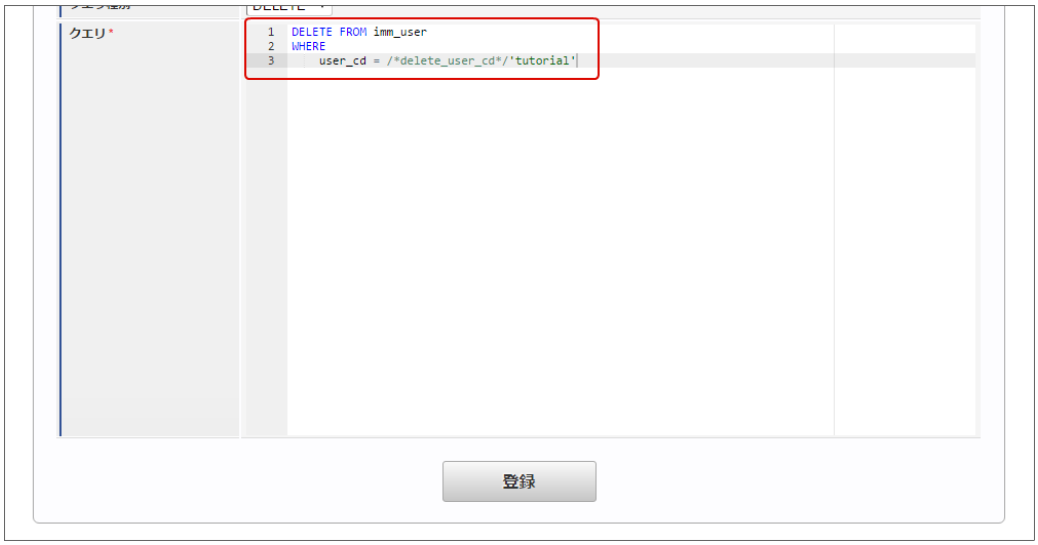

図：データベース種別、および、クエリ種別の定義

4. 入力値を「*入力値/出力値*」をもとに値を設定します。




図：入力値の定義

5. クエリに「*想定するDELETE文*」で提示したSQLを設定します。




図：クエリの定義

6. 「登録」をクリックします。


図：登録

以上で、DELETEを用いたユーザ定義（SQL）の作成が完了しました。

## ユーザ定義 - JavaScript

### ユーザ定義（JavaScript）

ユーザ定義(JavaScript）とは、JavaScriptを利用してタスクの処理内容を定義することが可能なユーザ定義の一つです。
タスクの入力値/出力値とJavaScriptの関数（function）の引数、および、戻り値を紐付けたプログラマブルな処理定義が行えます。
新規にロジックを構築することは勿論として、既存のintra-mart Accel Platform上のJavaScriptの資産も手軽にロジックフロー上で扱える様にできます。

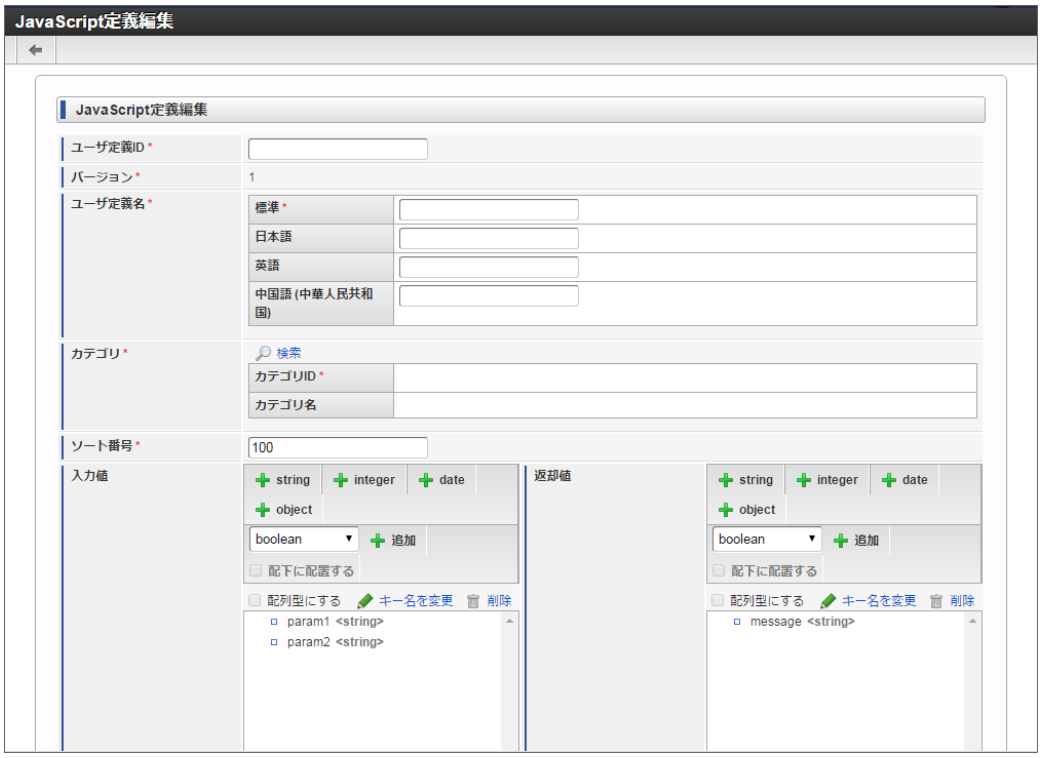
#### ユーザ定義（JavaScript）の作成画面への遷移手順

ユーザ定義（JavaScript）作成画面へは、ユーザ定義一覧画面から遷移します。

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義一覧」から、ユーザ定義一覧を開きます。
2. 画面上部のヘッダ内、「JavaScript定義新規作成」をクリックします。


図：ユーザ定義一覧 - JavaScript定義新規作成

3. 「JavaScript定義編集」画面が表示されます。




図：JavaScript定義編集画面

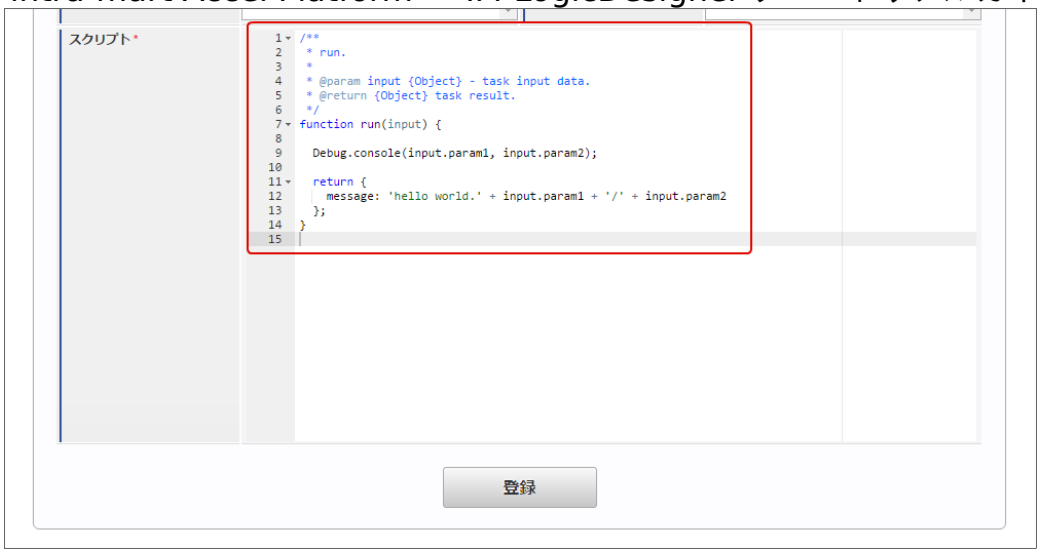
## 初期サンプルから見るユーザ定義（JavaScript）の詳細

この章では、JavaScriptを利用したユーザ定義の作成方法を、IM-LogicDesignerが提供する初期サンプルをもとに説明します。



### 初期サンプルについて

本章で扱う「初期サンプル」とは、「*ユーザ定義（JavaScript）の作成画面への遷移手順*」に基いて「JavaScript定義編集」画面に遷移した際、初めから定義されているJavaScriptコード、および、入力値/出力値のことを指します。




図：初期表示時に、サンプルとしてJavaScriptが定義されている

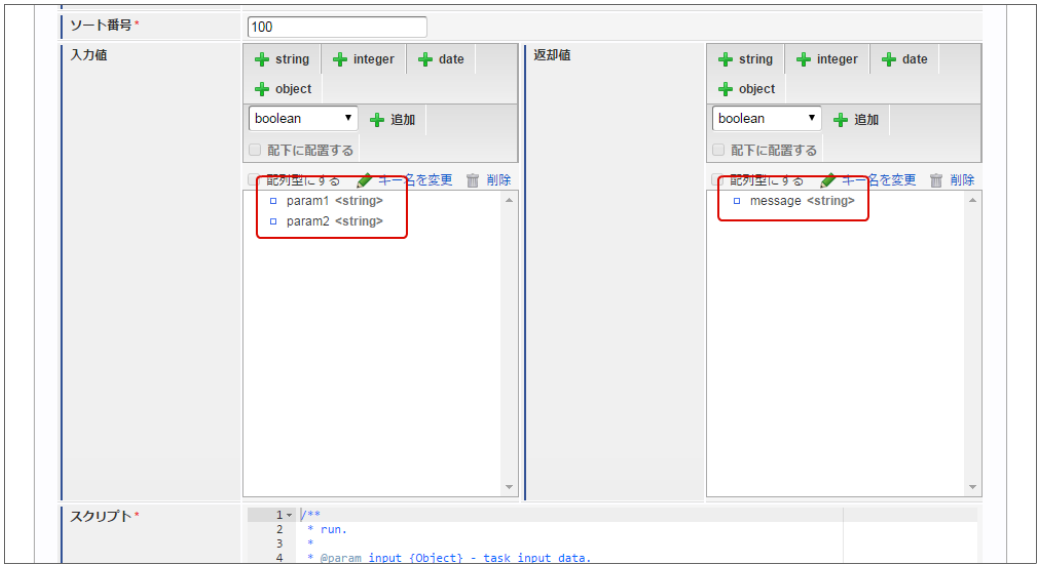
実際に定義されている初期サンプルのJavaScriptコードは以下の通りです。

```
/**
 * run.
 *
 * @param input {Object} - task input data.
 * @return {Object} task result.
 */
function run(input)  {

   Debug.console(input.param1, input.param2);

   return {
      message: 'hello world.' + input.param1 + '/' + input.param2
   };
}
```

初期サンプルのJavaScriptは、非常に単純なものですが、その中にはこれからユーザ定義（JavaScript）を作成する開発者にとって抑えるべきポイントが全て含まれています。

### 入力値/出力値

初めに、作成するユーザ定義を呼び出す際の入力値と、呼び出しが完了した際に返却する出力値の定義について説明します。
ユーザ定義（JavaScript）の作成において入力値/出力値は、開発者が定義したJavaScriptに紐づく形に手動で定義する必要があります。




図：初期サンプルの入力値、および、出力値

入力値

ユーザ定義（JavaScript）の入力値は、定義したJavaScriptの関数(function）の引数と紐付きます。

初期サンプルでは入力値として以下の値を定義しています。

- param1<string>
- param2<string>

定義された入力値は、「*スクリプト*」に定義されている「*エントリーポイント*」となる関数の引数に紐付きます。
（初期サンプルの7行目）

```
/**
 * ...
 */
function run(input) {
  //...
}
```

入力値は引数の直下に階層構造を保って紐づきます。利用方法は通常のJavaScriptのプロパティアクセスと同じです。
（初期サンプルの9行目）

```
function run(input) {
  // ...
  Debug.console(input.param1, input.param2);
  // ...
}
```

出力値

ユーザ定義（JavaScript）の出力値は、定義したJavaScriptの関数(function）の戻り値と紐付きます。

初期サンプルでは出力値として以下の値を定義しています。

- message<string>

定義された出力値は、「*スクリプト*」に定義されている「*エントリーポイント*」となる関数の戻り値と紐付きます。
（初期サンプルの11行目～13行目）

```
function run(input) {

  // ...

  return {
    message: 'hello world.' + input.param1 + '/' + input.param2
  };
}
```

出力値を扱う上で注意すべき点として、必ず出力値のキー名に合わせた形で戻り値を定義してください。
以下の様に定義した場合、出力値との紐付けは正しく行われません。

```
function run(input) {

  // ...

  return 'hello world.' + input.param1 + '/' + input.param2;
  // ここで返される文字列は、出力値として定義した"message<string>"とは紐付かない。
}
```

## スクリプト

次に、実際に処理を行うJavaScriptの定義方法と留意点を説明します。

JavaScript

ユーザ定義（JavaScript）では、通常のスクリプト開発モデルと同様にJavaScript、および、スクリプト開発向けim-BizAPIが利用可能です。

初期サンプルでは、im-BizAPIのうちデバッグ情報の表示を行う「Debugオブジェクト」を利用しています。
（初期サンプルの9行目）

```
function run(input) {
  // ...
  Debug.console(input.param1, input.param2);
  // ...
}
```

エントリーポイント

ユーザ定義（JavaScript）が実際に処理を行う際に、一番初めに実行する関数（function）はシステムで決まっています。
この処理開始時に一番最初に呼び出される関数をIM-LogicDesignerでは **エントリーポイント** と呼びます。
IM-LogicDesignerは定義されたスクリプトの中から、名称が「run」である関数を自動的に検出し、エントリーポイントとします。

初期サンプルでは、唯一存在する関数の名称に「run」を指定しています。
（初期サンプルの7行目）

```
/**
 * ...
 */
function run(input) {
  //...
}
```

コラム

エントリーポイントが無い場合

ユーザ定義（JavaScript）のスクリプト上に、エントリーポイントに該当する関数が定義されていない場合でもエラーとは判断されません。

入力値/出力値

「*入力値/出力値*」で説明したとおり、ユーザ定義（JavaScript）のスクリプトと入力値/出力値はエントリーポイントを介して紐付いています。
留意点としてエントリーポイントの引数、および、戻り値をIM-LogicDesignerでは以下のように扱います。

- 入力値に定義された値は、全てエントリーポイントの第一引数と紐付きます。
  - エントリーポイントに複数の引数が定義されていた場合、第二引数以降はundefinedとして扱われます。
  - 入力値を必要としないユーザ定義（JavaScript）の場合、引数のないエントリーポイントの定義も可能です。

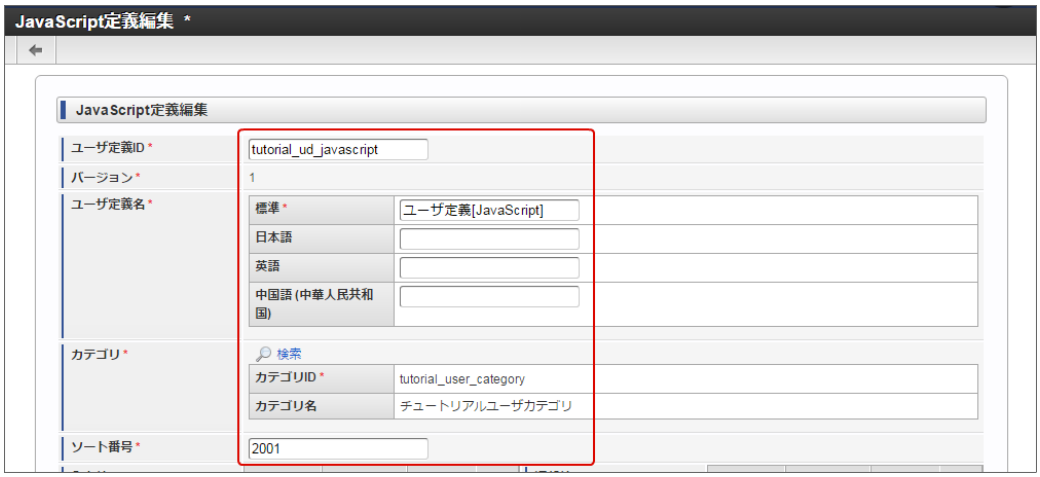
利用が制限されているJavaScriptについて

ユーザ定義（JavaScript）では、いくつかのim-BizAPIについて利用制限を行っています
詳細は「*参考：ユーザ定義（JavaScript）における制限*」を参照してください。

### ユーザ定義（JavaScript）を作成する。

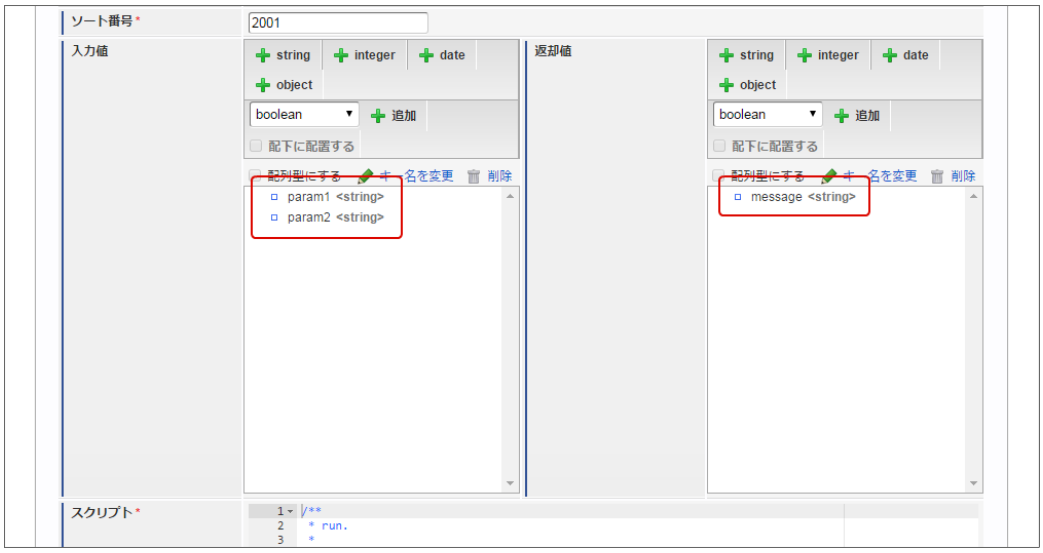
最後に、これまでの内容を踏まえてユーザ定義（JavaScript）を作成します。

1. 「JavaScript定義編集」画面を表示します。
2. ユーザ定義の基本情報となる各項目に以下の値を入力します。
   - ユーザ定義ID「tutorial_ud_javascript」
   - バージョン「1」（固定）
   - ユーザ定義名
     - 標準 - 「ユーザ定義[JavaScript]」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし
   - カテゴリ
     - カテゴリID - 「tutorial_user_category」
   - ソート番号「2001」



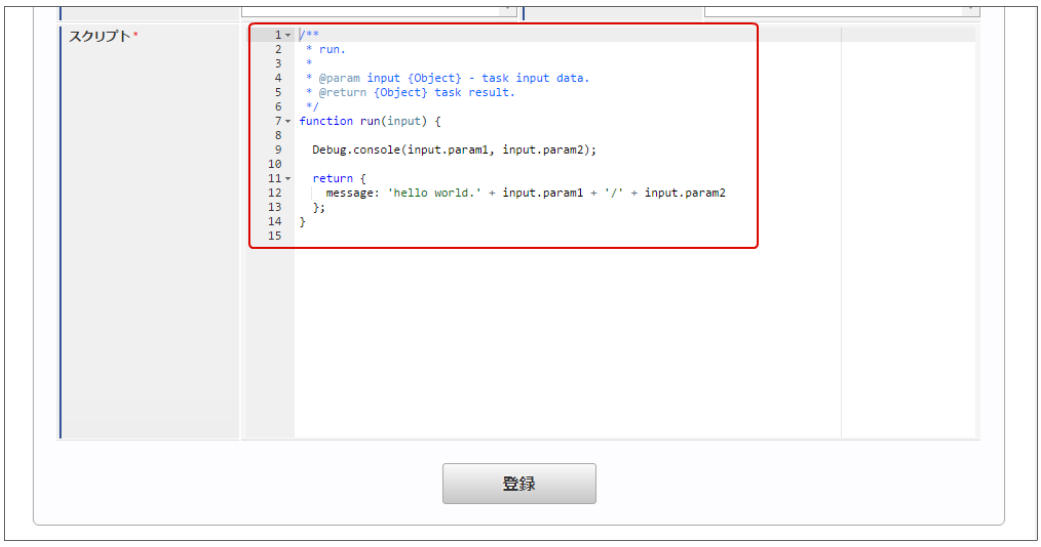

図：基本情報の入力

3. 入力値、および、出力値を「*入力値/出力値*」をもとに確認します。




図：入出力値の確認

4. スクリプトを「*スクリプト*」で提示されているJavaScriptをもとに確認します。




図：スクリプトの確認

5. 「登録」をクリックします。


図：登録

以上で、ユーザ定義（JavaScript）の作成が完了しました。

## 参考：ユーザ定義（JavaScript）における制限

この章では、ユーザ定義（JavaScript）のスクリプト定義時の制限について説明します。



### 制限されているim-BizAPIの概要

「*初期サンプルから見るユーザ定義（JavaScript）の詳細*」 - 「*スクリプト*」では原則全てのim-BizAPIの利用が可能ですが、その中で以下の観点に基づき利用を制限しているAPIがあります。

- システム管理系API、テナント管理系APIといったクリティカルな処理を行うことが可能なAPI。
- システムプロセスの呼び出しや、外部スクリプトの呼び出しといった、IM-LogicDesigner外のリソースの呼び出し、実行を行うAPI。
- 画面遷移を伴うAPI。

要件や課題の問題解決の際に上記のような処理が必要な場合は、IM-LogicDesignerを利用せず、従来通りの開発手法を採用することを考えてください。

> **コラム**
>
> 制限されているAPIを利用した場合
>
> 利用が制限されているAPIを利用した場合、該当のAPIが存在しないという旨のエラーメッセージと共に、ユーザ定義（JavaScript）の実行は失敗します。

im-BizAPIの利用制限設定

標準で、IM-LogicDesignerがシステムとして利用を制限しているAPIの一覧は以下のファイルに設定されています。

`WEB-INF/classes/META-INF/logic/jssp/blacklist`

> **警告**
>
> im-BizAPI制限の解除について
>
> IM-LogicDesignerでは、標準で制限しているim-BizAPIの制限を解除することは推奨していません。
> 標準で制限しているAPIの制限を解除して利用した場合、APIを利用したフローに起因する全ての動作内容について、一切のサポートが行われない点に留意してください。

コラム

制限するAPIの追加

標準で制限されたAPI以外で新しくIM-LogicDesignerでの利用を制限したいAPIがある場合、以下のドキュメントを参照してください。

- 「IM-LogicDesigner仕様書」 - 「ユーザ定義タスク」 - 「JavaScript」 - 「一部APIの利用制限設定」

## ユーザ定義 - REST

### ユーザ定義（REST）

ユーザ定義(REST）とは、RESTを利用した処理をタスクとして定義することが可能なユーザ定義の一つです。
タスクの入力値/出力値とRESTを呼び出すパラメータ/レスポンスを紐付け、フローのータスクとしてRESTの呼び出しを扱う事ができます。

ユーザ定義(REST）を利用することで、外部で提供されているREST APIの利用はもとより、イントラネット内に存在する別パッケージ製品の処理呼び出し、別環境に存在するintra-mart Accel Platformが提供するIM-LogicDesignerのフロールーティングの実行まで、多岐に渡る活用が可能です。

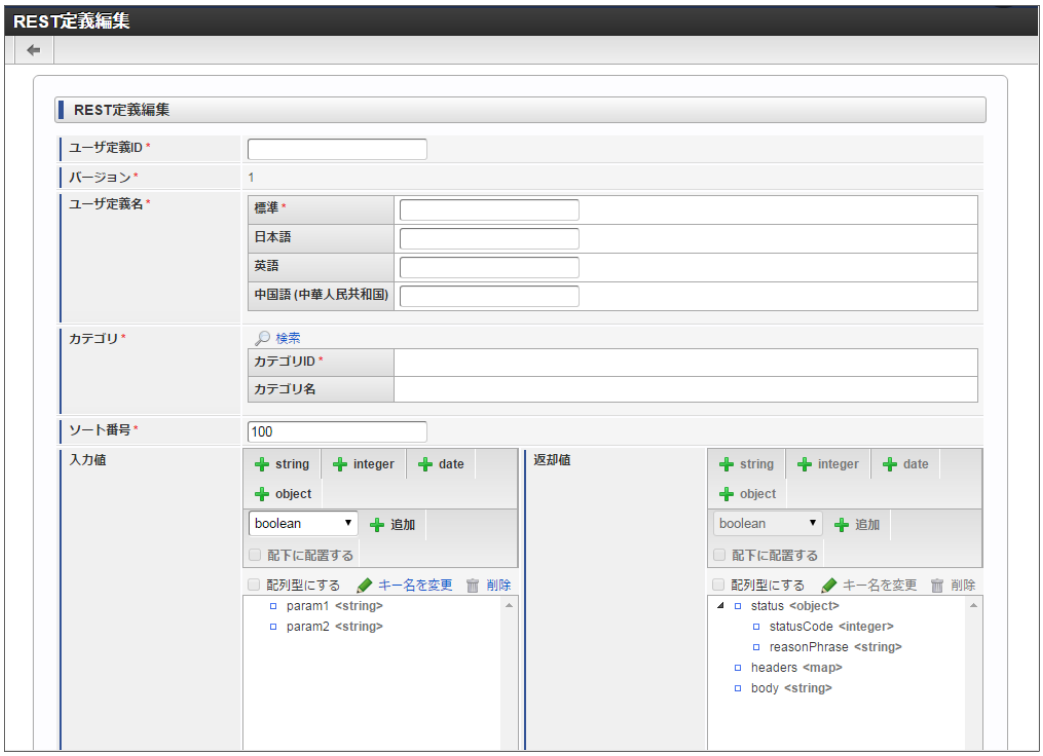
#### ユーザ定義（REST）の作成画面への遷移手順

ユーザ定義（REST）作成画面へは、ユーザ定義一覧画面から遷移します。

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義一覧」から、ユーザ定義一覧を開きます。
2. 画面上部のヘッダ内、「REST定義新規作成」をクリックします。


図：ユーザ定義一覧 - REST定義新規作成

3. 「REST定義編集」画面が表示されます。




図：REST定義編集画面

## ユーザ定義（REST）の詳細

この章では、RESTを利用したユーザ定義の作成方法を、仮想サービス（REST API）をもとに説明します。

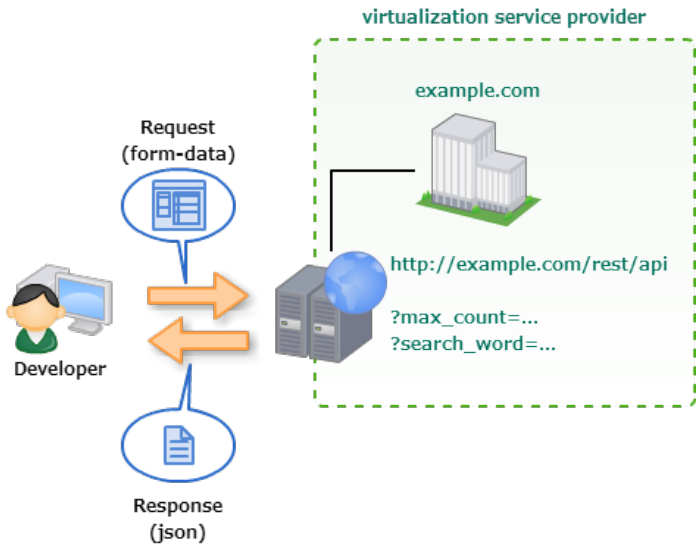


### 本チュートリアルで作成する概要

本チュートリアルでは、仮想のサービス提供者と提供REST APIを定義します。
その仮想サービスへアクセスを行うタスクの処理の実装を通して、RESTを利用したユーザ定義の作成方法とその詳細を説明します。

本チュートリアルで定義する仮想のサービス提供者とその詳細

本チュートリアルで定義する仮想のサービス提供者のイメージは以下の通りです。




図：仮想のサービス提供者のイメージ

詳細な設定項目、および、想定する値/設定内容については以下の通りです。

| 設定項目 | 想定する値/設定内容 |
| --- | --- |
| エンドポイント | http://example.org/rest/api |
| メソッド | POST |
| リクエストパラメータ | 以下の二つのパラメータを受け取ることができるものとします。<br>- max_count<br>  - 取得データの上限個数を指定します。<br>  - 型は数値型を取るものとします。<br>- search_word<br>  - 取得データを部分一致で検索する単語を指定します。<br>  - 型は文字列型を取るものとします。 |
| 入力様式 | x-www-form-urlencoded |
| 出力様式 | json |
| 認証方式 | OAuth2を利用した認証を利用します。 |

### リクエスト情報

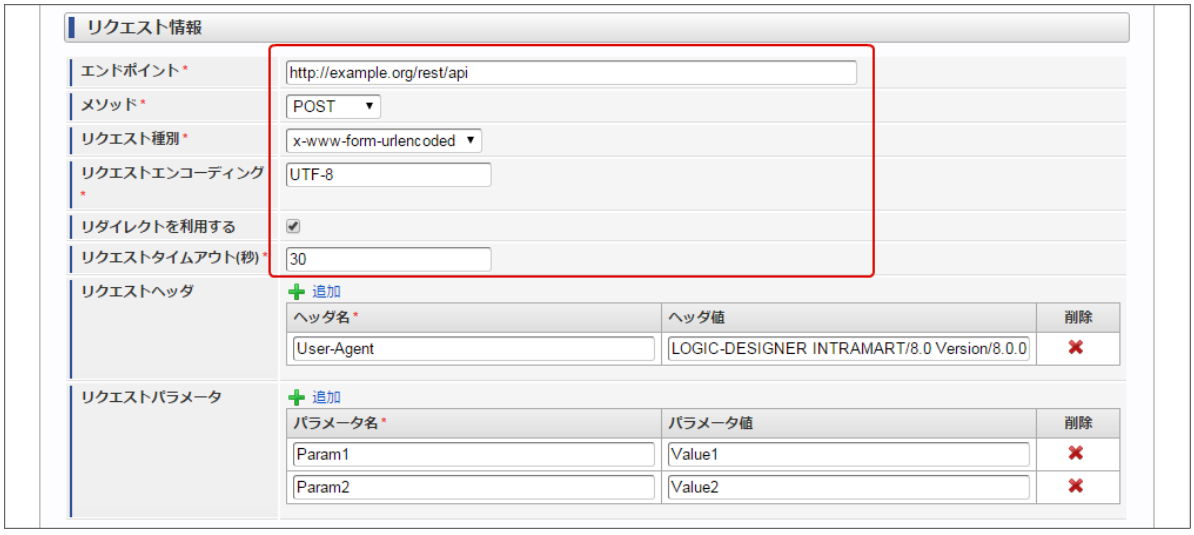
はじめに、今回作成するユーザ定義（REST）が対象とするサービスへのリクエスト情報の設定を行います。

基礎情報

リクエストの基礎情報を「*本チュートリアルで作成する概要*」を元に定義します。
ここでは「*リクエストヘッダ*」「*リクエストパラメータ*」を除くリクエスト情報を基礎情報としています。

定義内容は以下の通りです。

- エンドポイント
  - http://example.org/rest/api
- メソッド
  - POST
- リクエスト種別
  - x-www-form-urlencoded
- リクエストエンコーディング
  - UTF-8
- リダイレクトを利用する
  - チェックあり
- リクエストタイムアウト（秒）
  - 30



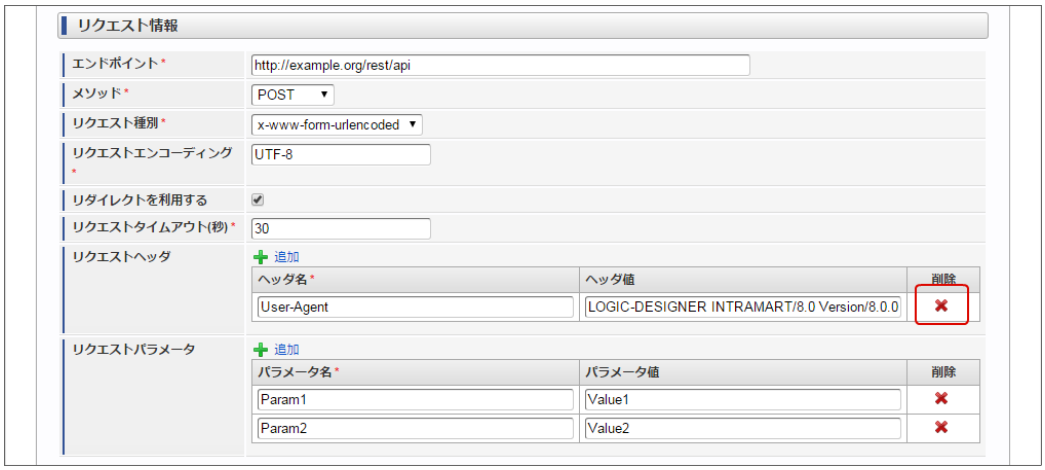

図：リクエスト情報 - 基礎情報

以上で、基礎情報の設定が完了しました。

リクエストヘッダ

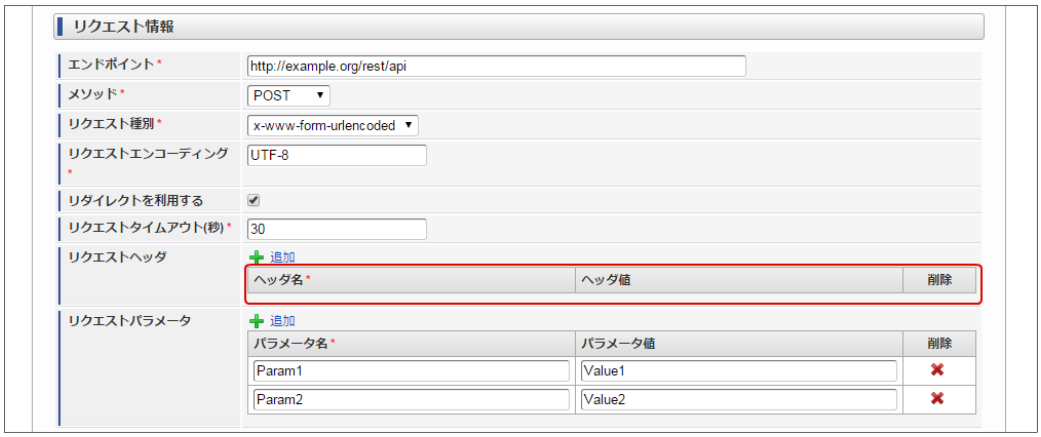
リクエストのヘッダ情報を「*本チュートリアルで作成する概要*」を元に定義します。
今回、OAuth2認証を利用するため、ヘッダにAuthorizationヘッダを定義します。

1. リクエストヘッダの初期値であるUser-Agentヘッダの行の右にある削除アイコンをクリックします。



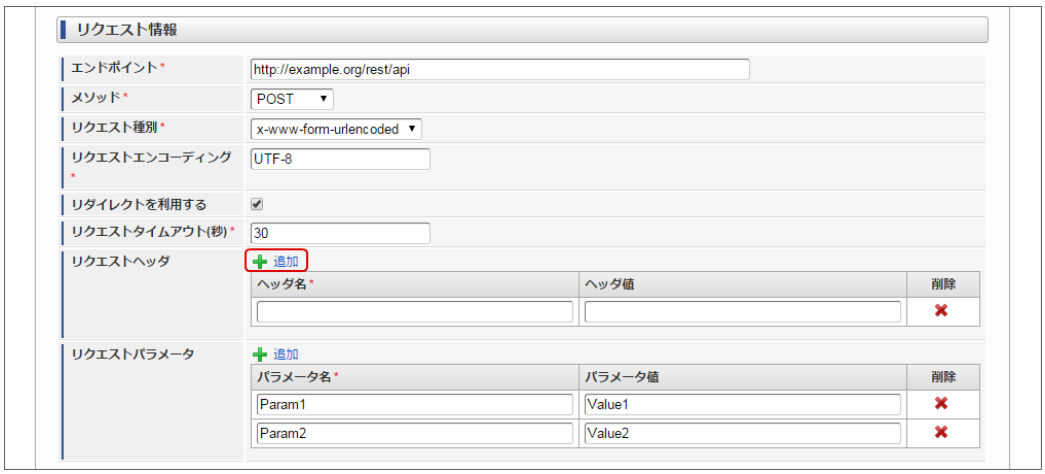

図：ヘッダの削除その1

2. User-Agentヘッダが削除されます。



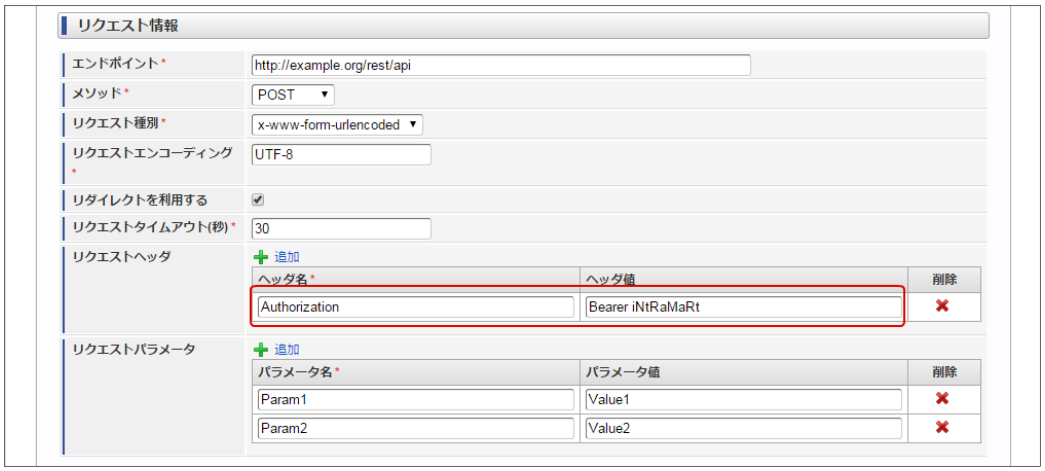

図：ヘッダの削除その2

3. ヘッダ情報を扱うテーブルの左上にある追加アイコンをクリックし、新しい行を追加します。




図：ヘッダの追加その1

4. 追加された行に以下の値を入力します。
   - ヘッダ名 「Authorization」
   - ヘッダ値 「Bearer iNtRaMaRt」



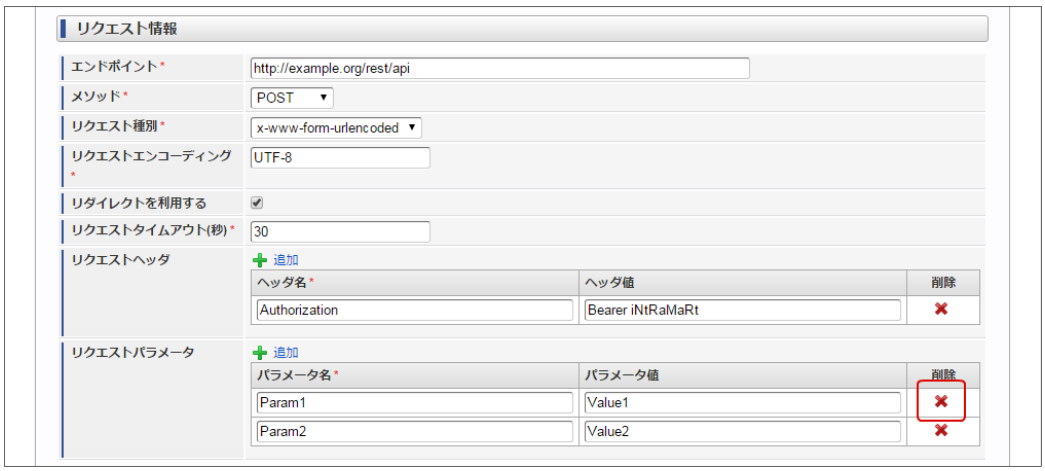

図：ヘッダの追加その2

以上で、リクエストヘッダの設定が完了しました。

リクエストパラメータ

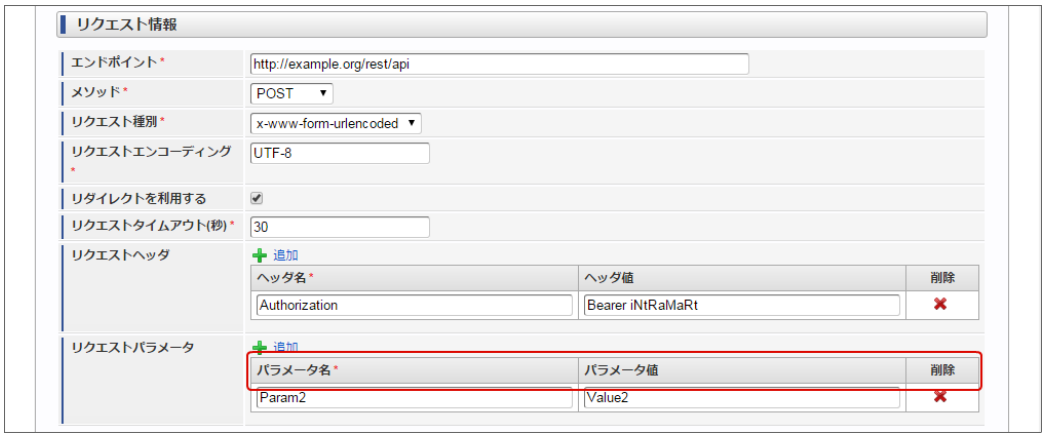
リクエストのパラメータ情報を「*本チュートリアルで作成する概要*」を元に定義します。
今回想定するサービスでは、二種類のパラメータの指定を許容するため、それぞれを定義します。

1. リクエストパラメータの初期値である`Param1`パラメータ名の行の右にある削除アイコンをクリックします。



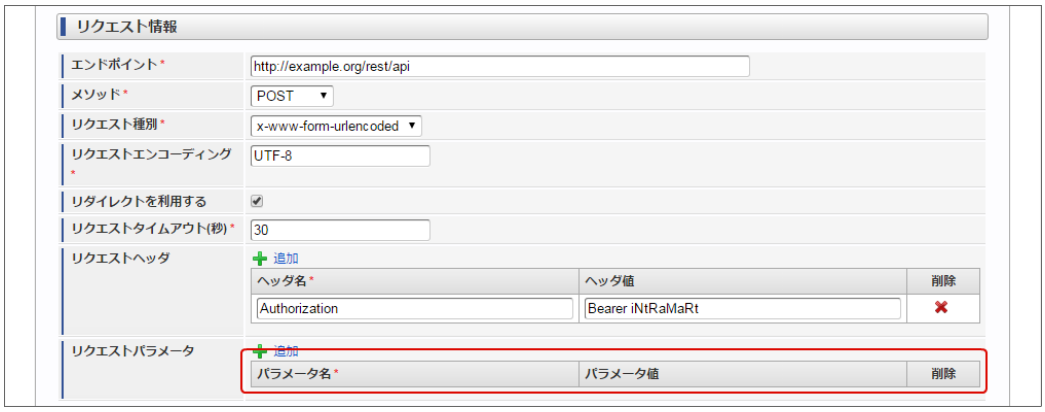

図：パラメータの削除その1

2. `Param1`パラメータが削除されます。



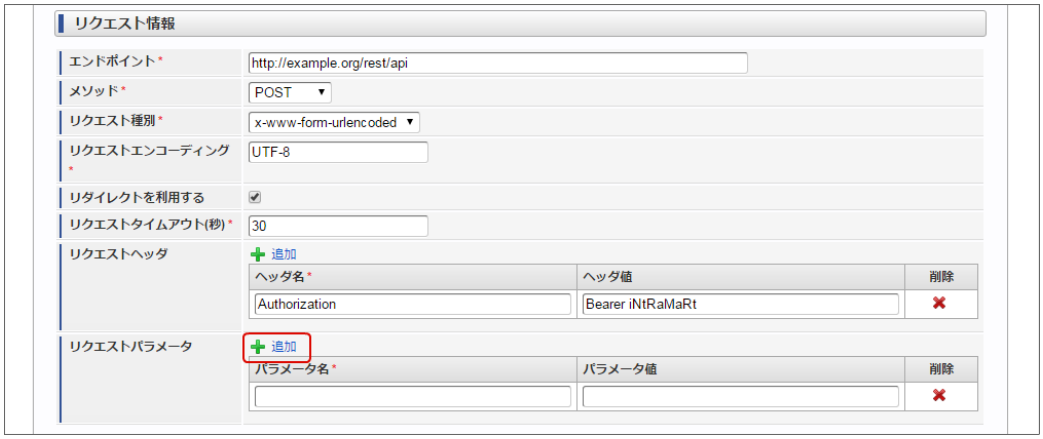

図：パラメータの削除その2

3. 同様の方法で`Param2`パラメータの行も削除します。



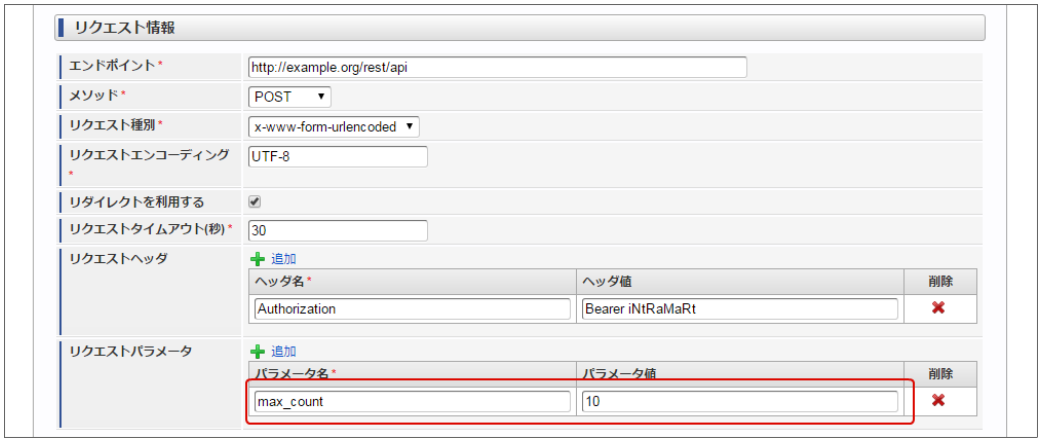

図：パラメータの削除その3

4. パラメータ情報を扱うテーブルの左上にある追加アイコンをクリックし、新しい行を追加します。



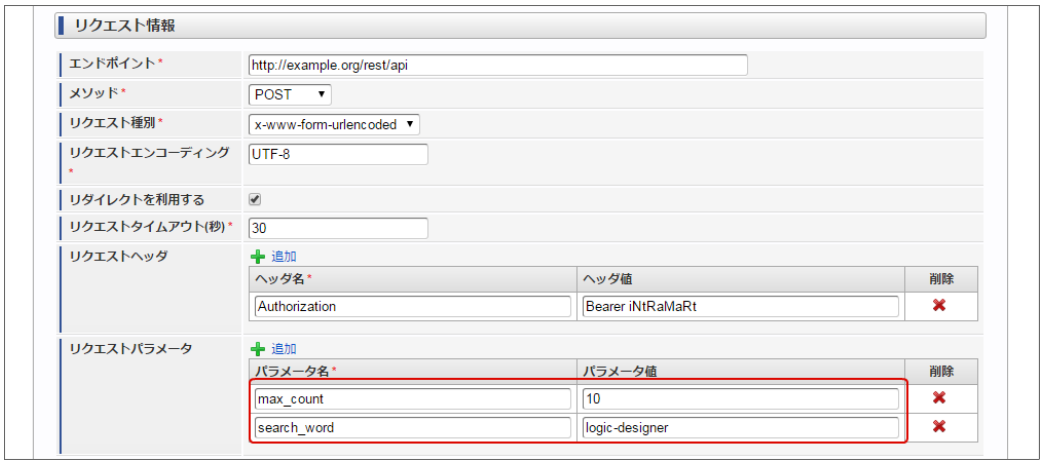

図：パラメータの追加その1

5. 追加された行に以下の値を入力します。
   - パラメータ名 「max_count」
   - パラメータ値 「10」




図：パラメータの追加その2（max_count）

6. 同様の手順で新しく行を追加し、以下の値を入力します。
   - パラメータ名 「search_word」
   - パラメータ値 「logic-designer」




図：パラメータの追加その3（search_word）

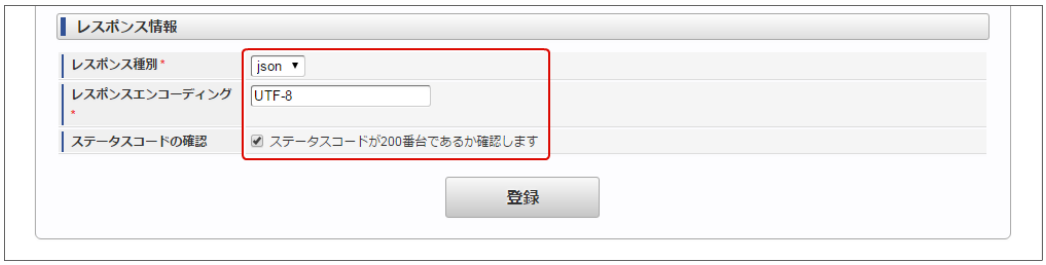
以上で、リクエストパラメータの設定が完了しました。

### レスポンス情報

次に、今回作成するユーザ定義（REST）が対象とするサービスが返すレスポンス情報の設定を行います。
レスポンス情報を「*本チュートリアルで作成する概要*」を元に定義します。

定義内容は以下の通りです。

- レスポンス種別
  - json
- レスポンスエンコーディング
  - UTF-8
- ステータスコードの確認
  - チェックボックス：オン




図：レスポンス情報

以上で、レスポンスの設定が完了しました。

### 入力値/出力値

次に、作成するユーザ定義（REST）を呼び出す際の入力値と、呼び出しが完了した際に返却する出力値を定義します。

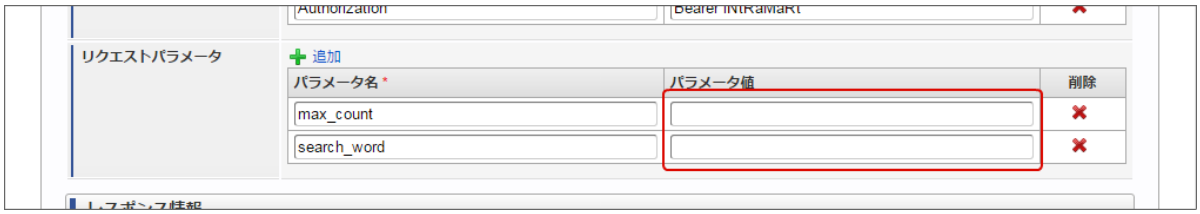
入力値

ユーザ定義（REST）の入力値は、「*リクエスト情報*」に動的設定を行う場合に定義します。
ここでは、「*リクエストパラメータ*」で定義したパラメータを入力値によって動的に変更するよう変更します。

1. 入力値として以下の項目を設定します。

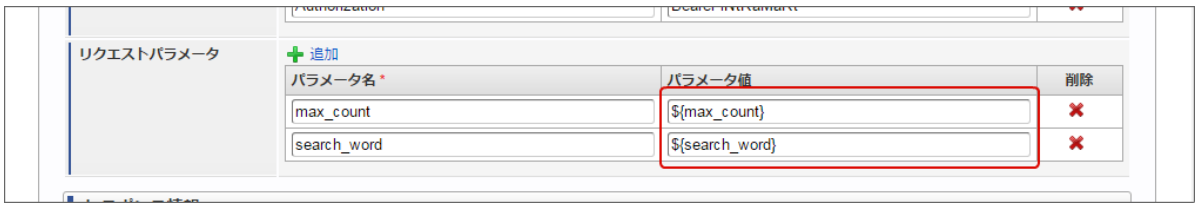
| 入力値 | 説明 |
| --- | --- |
| max_count<integer> | リクエストパラメータの「max_count」に紐づける入力値。 |
| search_word<string> | リクエストパラメータの「search_word」に紐づける入力値。 |

2. リクエストパラメータに定義した「max_count」、および、「search_word」のパラメータ値を削除します。




図：定義済みのパラメータ値の削除

3. 新しく以下のパラメータ値を定義します。
   - パラメータ名 「max_count」- パラメータ値 「${max_count}」
   - パラメータ名 「search_word」- パラメータ値 「${search_word}」




図：新規パラメータ（EL式）の追加

入力値を利用した動的設定についての詳細は「*エンドポイントやヘッダへのEL式を介した動的指定*」を参照してください。

出力値

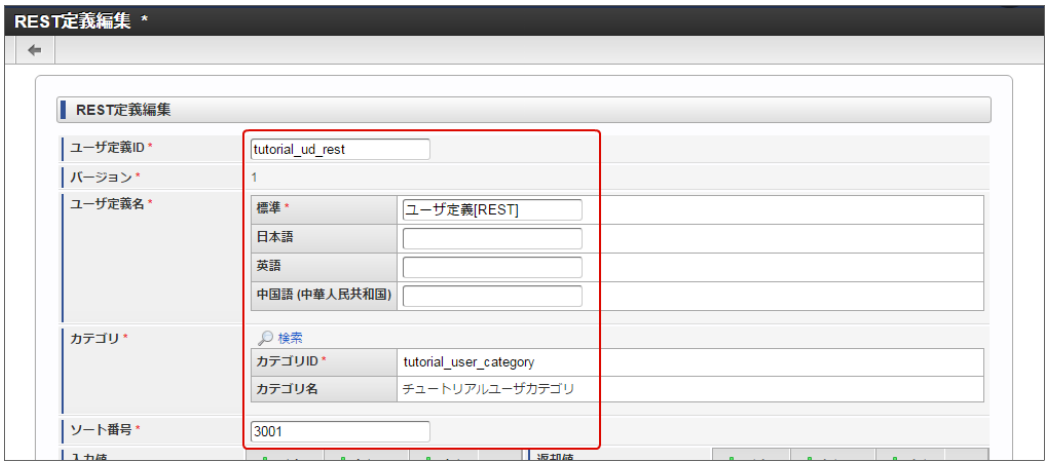
ユーザ定義（REST）の出力値は、入力値とは異なりシステムによって固定です。
IM-LogicDesignerではREST APIの実行結果として返されたレスポンスをもとに、以下の内容をユーザ定義（REST）の汎用的な出力値として定義します。

| 出力値 | 説明 |
|---|---|
| status<object> - statusCode<integer> | HTTPステータスコードが格納されます。 |
| status<object> - reasonPhrase<string> | HTTPステータスコードに基づく原因を表す文字列が格納されます |
| headers<map> | ヘッダ情報が格納されます。 |
| body<object> | 実行結果（HTTPレスポンスの本体） |

### ユーザ定義（REST）を作成する。

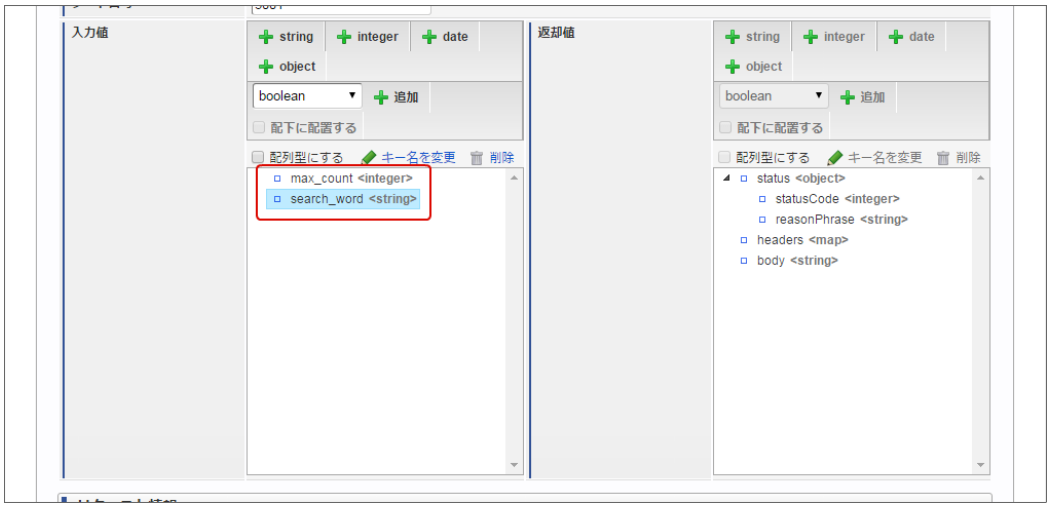
最後に、これまでの内容を踏まえてユーザ定義（REST）を作成します。

1. 「REST定義編集」画面を表示します。
2. ユーザ定義の基本情報となる各項目に以下の値を入力します。
   - ユーザ定義ID「tutorial_ud_rest」
   - バージョン「1」（固定）
   - ユーザ定義名
     - 標準 - 「ユーザ定義[REST]」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし
   - カテゴリ
     - カテゴリID - 「tutorial_user_category」
   - ソート番号「3001」



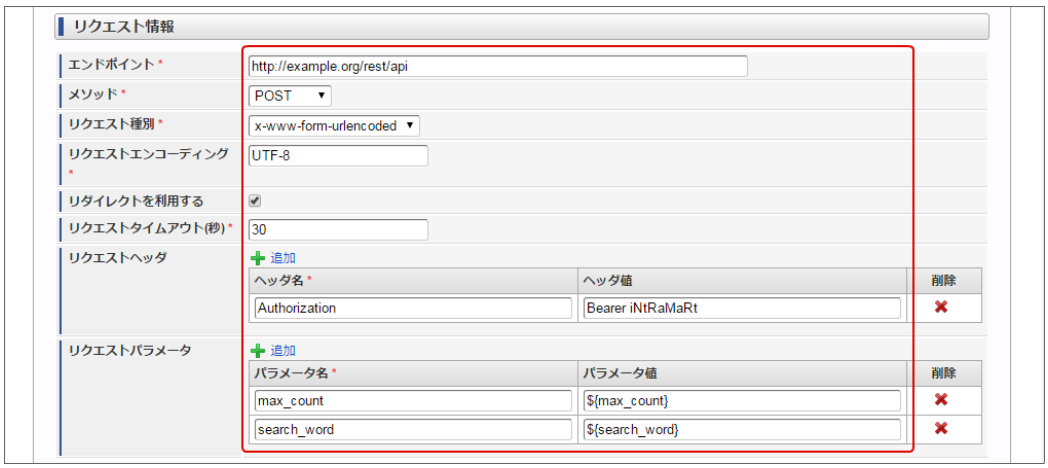

図：基礎情報の入力

3. 入力値、および、出力値を「*入力値/出力値*」をもとに定義します。



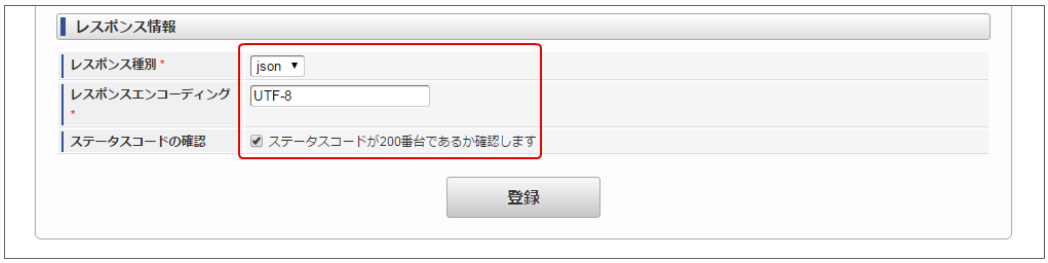

図：入出力値の定義

4. リクエスト情報を「*リクエスト情報*」をもとに定義します。




図：リクエスト情報の定義

5. レスポンス情報を「*レスポンス情報*」をもとに定義します。



図：レスポンス情報の定義

6. 「登録」をクリックします。


図：登録

以上で、ユーザ定義（REST）の作成が完了しました。

## 参考：より高度なREST APIの呼び出し

この章では、ユーザ定義（REST）を利用したREST APIの呼び出しについて、より高度な設定値、および、設定方法の説明を行います。



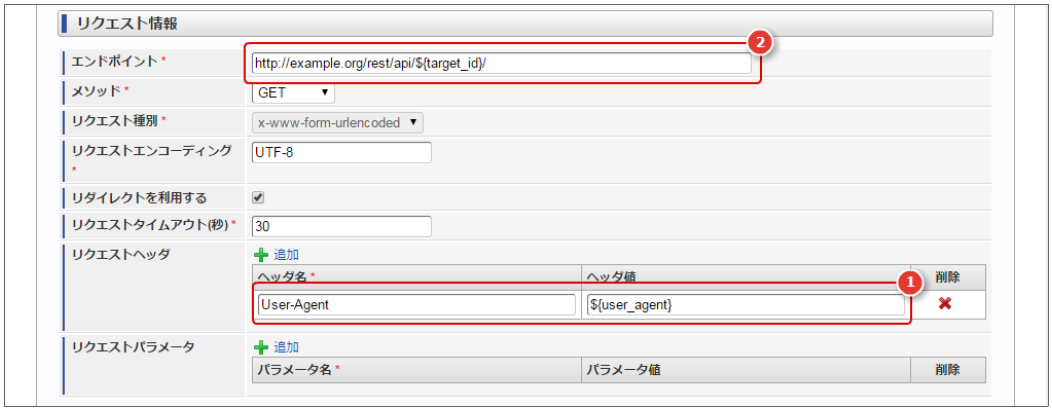
### エンドポイントやヘッダへのEL式を介した動的指定

「*入力値/出力値*」では、ユーザ定義（REST）の入力値と、リクエストパラメータの値とを紐付け、動的な変更が可能な定義方法を説明しました。IM-LogicDesignerでは、リクエストパラメータにかぎらず、リクエスト情報の全ての項目に関してEL式を利用することで動的な変更を定義可能です。

> **コラム**
>
> EL式について
>
> EL式の詳細については、「IM-LogicDesigner仕様書」 - 「付録」 - 「EL式」を参照してください。

動的指定の例：その1

リクエストパラメータと同様に、エンドポイントやヘッダに指定する値を入力値から動的に定義することが可能です。

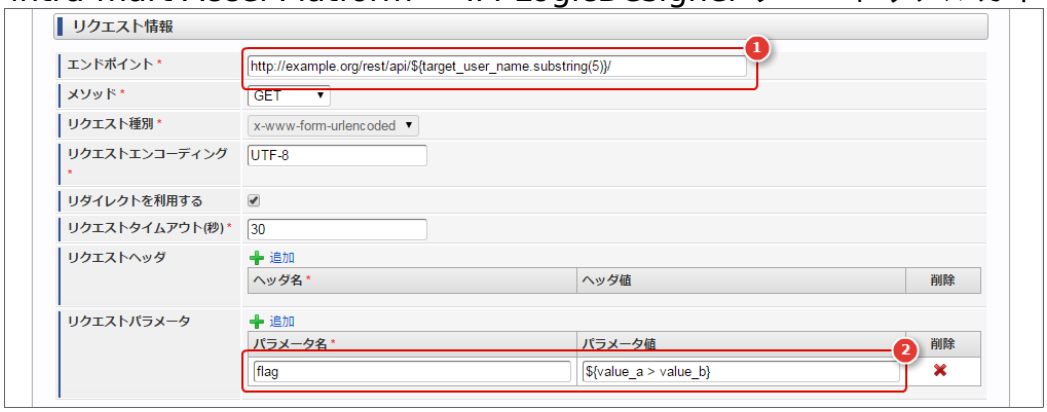



図：動的変更可能なパラメータ（EL式）の定義その1

1. 入力値「user_agent」として定義された値が、User-Agentヘッダの値として設定されます。
2. 動的指定は部分的な適応も可能です。
   入力値「target_id」を元に、URLの特定の部分（PathVariable）を動的に設定しています。

動的指定の例：その2

EL式を利用し、演算結果を最終的な設定値として定義することが可能です。




図：動的変更可能なパラメータ（EL式）の定義その2

1. 入力値「target_user_name」として定義された値の、6文字目以降を最終的な値として設定しています。
2. 入力値「value_a」、および、「value_b」の比較結果（真偽値）を最終的な値として設定しています。

注意点として、全ての演算結果は文字列（ないしは、文字列として取り扱い可能）である必要があります。

### リクエストおよびレスポンスの種別とその詳細

リクエスト情報、および、レスポンス情報で設定可能な以下の項目について詳細を説明します。

- リクエスト種別
- レスポンス種別

リクエスト種別

リクエスト種別で設定可能な値は以下の通りです。

| リクエスト種別 | 詳細 |
|---|---|
| x-www-form-urlencoded | リクエストをx-www-form-urlencoded形式で送信します。<br>パラメータはkey=value形式で送信されます。<br>リクエストのパラメータは自動的にURLエンコードされるため、明示的にエンコードを行う必要はありません。 |
| application/json | リクエストをjson形式で送信します。<br>application/jsonリクエスト種別を利用する場合は、必ず入力値にbodyを定義する必要があります。<br>IM-LogicDesignerは入力値に定義されたbodyをJSONに変換し、リクエストボディに含めて送信を行います。 |
| form-data | リクエストをmultipart/form-data形式で送信します。<br>パラメータはkey=value形式で送信されます。<br>リクエストのパラメータは自動的にURLエンコードされるため、明示的にエンコードを行う必要はありません。<br><br>なお、form-dataリクエスト種別を利用する場合、パラメータにバイナリ形式の値を指定可能です。<br>主にファイルのアップロード等の処理が含まれるREST APIに対して有用です。 |
| raw | 上記以外の形式で送信したい場合に選択してください。<br>rawリクエスト種別を利用する場合は、必ず入力値にbodyを定義する必要があります。<br>IM-LogicDesignerは入力値に定義されたbodyをそのままリクエストボディに含めて送信を行います。<br>また、raw リクエスト種別を利用する場合は、「*リクエストヘッダ*」に明示的なContent-Typeの指定を行ってください。 |

レスポンス種別

レスポンス種別で設定可能な値は以下の通りです。

| レスポンス種別 | 詳細 |
|---|---|
| json | REST APIのレスポンスをJSONとして受け取ります。<br>IM-LogicDesignerは、レスポンスのbody部をobjectとして取り出し、JSONとして解釈します。<br>取り出し結果は文字コードが加味されます。 |

| レスポンス種別 | 詳細 |
| --- | --- |
| text | REST APIのレスポンスをTextとして受け取ります。<br>IM-LogicDesignerは、レスポンスのbody部をstringとして取り出し、HTMLとして解釈します。<br>取り出し結果は文字コードが加味されます。 |
| raw | REST APIのレスポンスをRaw-Dataとして受け取ります。<br>IM-LogicDesignerは、レスポンスのbody部をバイナリとして取り出します。<br>主にダウンロード等の処理が含まれるREST APIに対して有用です。 |

### ステータスコードのチェック

レスポンス情報の「ステータスコードの確認」を利用することで、「2xx Success」に準ずるステータスコード以外のレスポンスが返却された場合の動作を制御することができます。

具体的には、「ステータスコードの確認」を設定（チェックを付ける）した場合、200番代以外のステータスコードが返って来た場合、タスクの実行エラーとして例外をスローします。
設定をしなかった場合、200番代以外のステータスコードが返ってきた場合（例として400番台（処理の失敗）、500番台（サーバエラー））でも正常に処理が行われたと判断し、次のタスクへ処理をすすめます。

### JSONによる複雑なレスポンスを受け取る

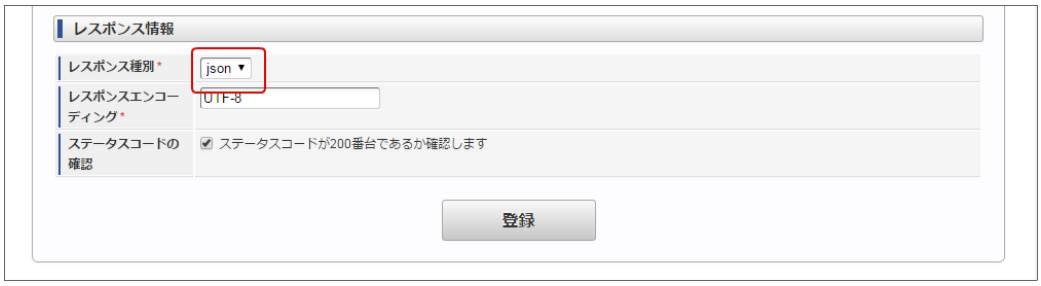
ユーザ定義（REST）においてレスポンスをJSONで受け取る際に、取得するJSONの構造が複雑な場合があります（特に外部サービスなどを利用する場合）。
複雑な構造のJSONを正しく受け取るためには、返却値のbody<object>配下に取得対象のJSONの構造を正確に定義する必要があります。

しかしながら取得するJSONの複雑さが増すほど、定義ミスによって正しくレスポンスが受け取れない事態が起こりやすくなります。
そして何より、そうした複雑なJSONの構造を一つ一つ定義していく作業はとても煩雑です。
そこでIM-LogicDesignerでは、レスポンスの定義にJSONを直接利用する手段を提供しています。

ここでは以下の様なJSONをレスポンスを返すサービスを想定します。

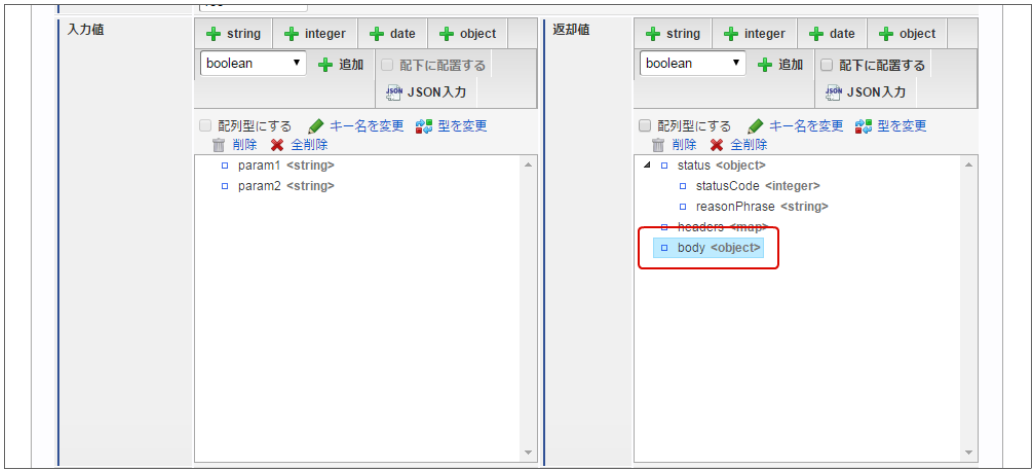
```
{
  "resultCd" : "56afc9a9-2988-4d13-ace5-1af3e21782f8",
  "basic" : {
    "id" : "BS-000001",
    "name" : "Basic Result Pattern",
    "error" : false,
    "sortKey" : 101,
    "parameter" : ["BS-1", "BS-2", "BS-3"]
  },
  "advanced" : {
    "id" : "ADV-10000101",
    "name" : "Advanced Result Pattern",
    "error" : false,
    "sortKey" : 1001,
    "dependency" : [
      {
        "dependId" : "DP-100",
        "dependName" : "DP-ADV-300",
        "dependVersion" : "1.0.0",
        "enable" : true
      },
      {
        "dependId" : "DP-200",
        "dependName" : "DP-ADV-500",
        "dependVersion" : "2.1.0",
        "enable" : false
      }
    ]
  }
}
```

1. 「REST定義編集」画面を表示します。
2. レスポンス情報の項目を以下のとおりに設定します。
   - レスポンス種別 - json



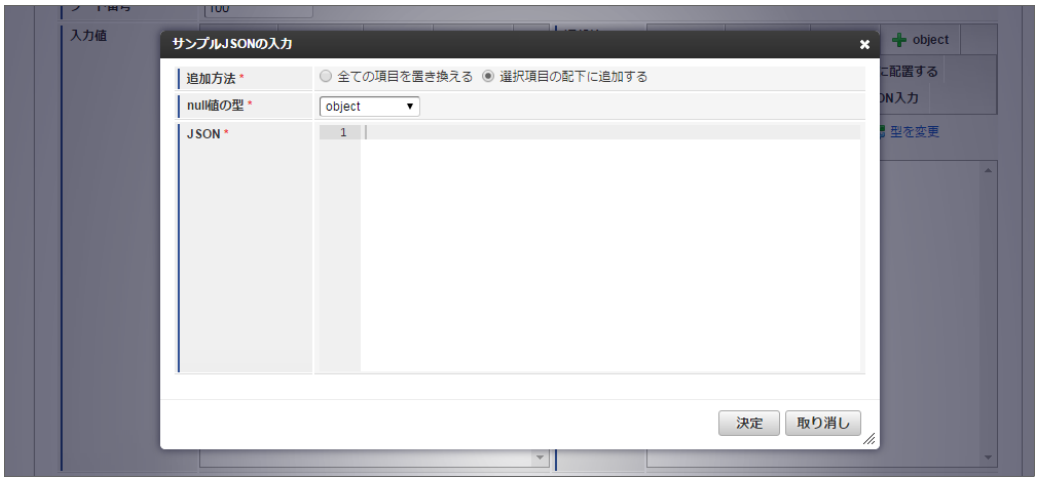

図：JSONレスポンスの設定

3. 返却値が自動で更新され、`body`パラメータの型が`object`になっていることを確認し、`body<object>`パラメータをクリックして選択状態にします。



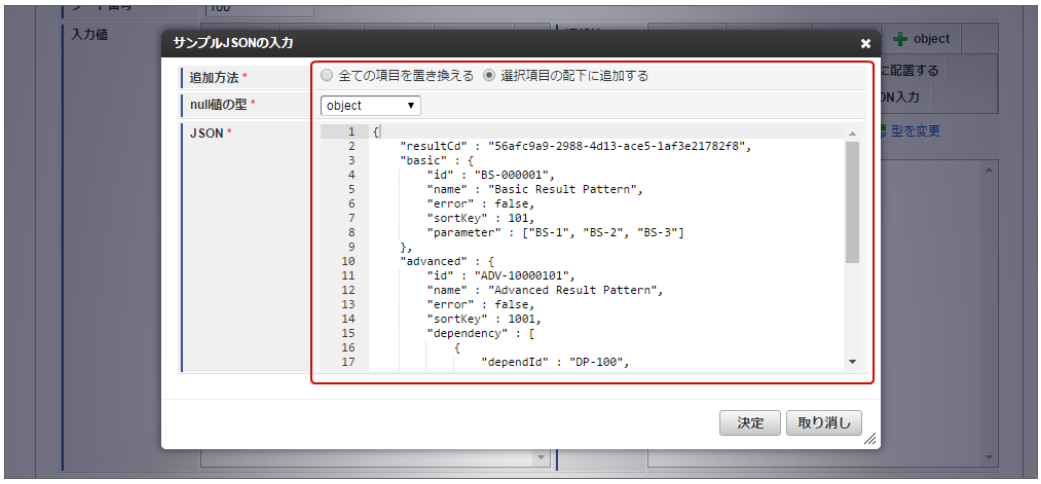

図：自動設定の確認、および、選択

4. 返却値の上部にある「JSON入力」をクリックします。
5. 「サンプルJSONの入力」画面が表示されます。




図：「サンプルJSONの入力」画面

6. 画面項目を以下のとおりに設定します。
   - 追加方法 - `選択項目の配下に追加する`
   - null値の型 - `object`
   - JSON - 上記で定義した想定するJSONレスポンス




図：「サンプルJSONの入力」の設定

7. 画面下部の決定をクリックします。
8. 返却値のbody<object>配下に、レスポンスのJSON構造が自動生成されたことを確認してください。


図：JSON構造の自動生成

## ユーザ定義 - Database Fetch

### ユーザ定義（Database Fetch）

ユーザ定義（Database Fetch）とは、データベース上に格納されている大量のデータを効率的に取得・処理することが可能なユーザ定義の一つです。
タスクの入力値と2Way-SQLを用いたSELECT文とを組み合わせた取得処理定義を行い、実行結果（出力値）を繰り返しの形で扱うことができます。

ユーザ定義（Database Fetch）と、「*ユーザ定義 - SQL(2WaySQL)*」を比較した場合、以下に示す違いがあります。

- ユーザ定義（Database Fetch）で定義可能なSQLはSELECT文のみです。
- ユーザ定義（Database Fetch）は「繰り返し」制御要素と同じ、開始/終了をペアで持つタスクを提供します。

#### ユーザ定義（Database Fetch）の作成画面への遷移手順

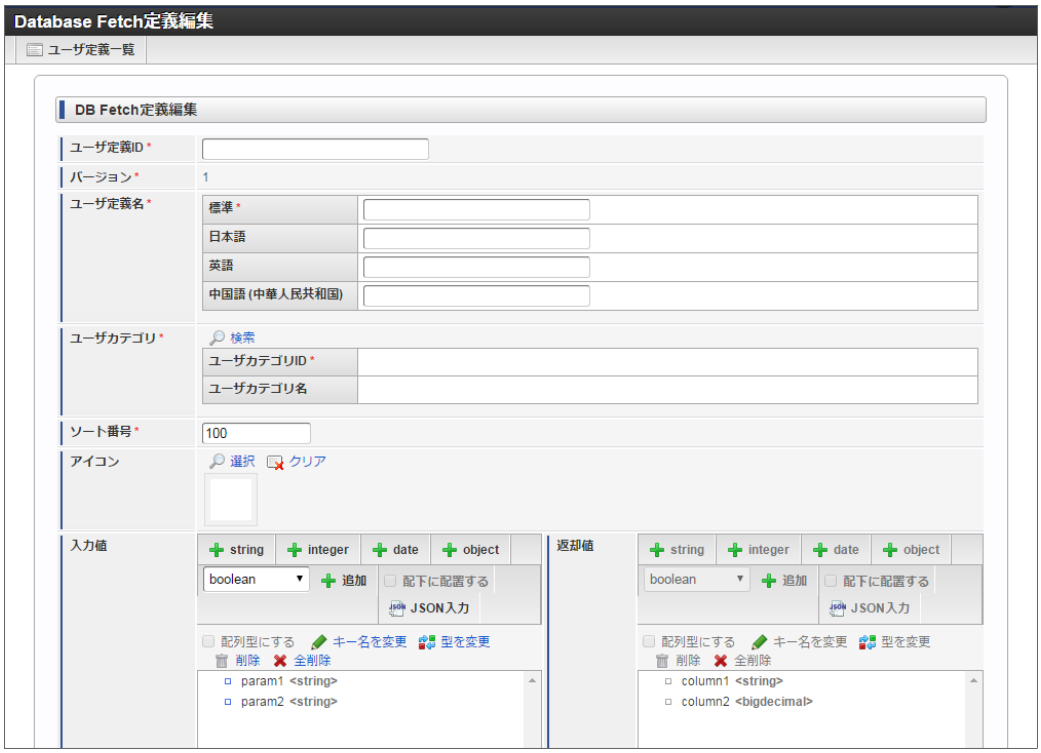
ユーザ定義（Database Fetch）作成画面へは、サイトマップメニューとユーザ定義一覧画面から遷移できます。

メニューから作成画面へ遷移する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義」配下から、「Database Fetch定義新規作成」をクリックします。


図：サイトマップ - ユーザ定義

2. 「Database Fetch定義編集」画面が表示されます。




図：Database Fetch定義編集画面

ユーザ定義一覧画面から作成画面へ遷移する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義一覧」から、ユーザ定義一覧を開きます。
2. 画面上部のヘッダ内、「新規作成」をクリックし、ポップアップしたメニューから「Database Fetch定義新規作成」をクリックします。


図：ユーザ定義一覧 - Database Fetch定義新規作成

3. 「Database Fetch定義編集」画面が表示されます。

## ユーザ定義（Database Fetch）の詳細

この章では、Database Fetchを利用したユーザ定義の作成方法とその詳細について説明します。

> **コラム**
>
> ユーザ定義（Database Fetch）の詳細を進める上での留意点
>
> ユーザ定義（Database Fetch）の作成方法は、多くの点で「*ユーザ定義 - SQL(2WaySQL)*」-「*SELECTを用いたユーザ定義（SQL）の作成*」と内容が共通しています。
> そのため、本章で行う説明の多くは「*SELECTを用いたユーザ定義（SQL）の作成*」と類似していることに留意してください。
>
> なお、ユーザ定義（SQL）との相違点はコラムにて追記しています。
> ユーザ定義（SQL）の説明を読了済みの開発者の方は、コラムを中心に確認していただくことで効率よく要点を理解いただけます。

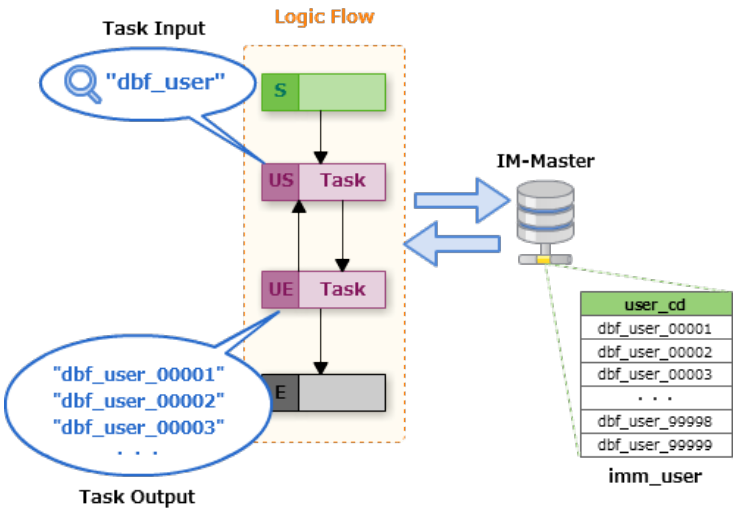


### 本チュートリアルで作成するユーザ定義の概要

本チュートリアルでは、

**「IM-共通マスタ上のユーザのうち、タスクの入力値として与えられた半角英数で始まるユーザコードを持つユーザを取得する」**

というタスクの処理の実装を通して、ユーザ定義（Database Fetch）の作成方法とその詳細を説明します。
作成するユーザ定義の処理イメージは以下の通りです。




図： 処理イメージ

図の例では、入力値として「dbf_user」を与えた場合、IM-共通マスタのユーザ内で該当する以下のようなユーザ情報（ユーザコード）が出力値として取得されます

- dbf_user_00001
- dbf_user_00002
- dbf_user_00003

なお、今回の取得処理では処理内容の簡略化のため、期間情報や多言語情報を加味しません。

### データベース種別の選択

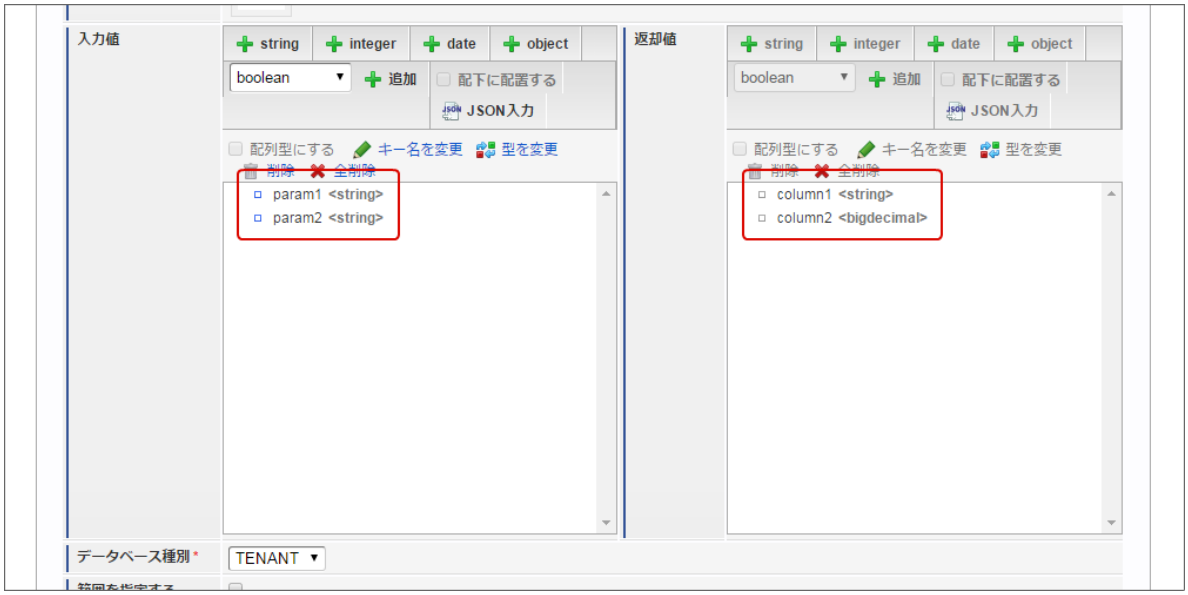
はじめに、今回作成するユーザ定義（Database Fetch）が対象とするデータベース種別を設定します。
本チュートリアルでは以下の値を設定します。

- データ種別 - 「TENANT」

> **コラム**
>
> ユーザ定義（SQL）との相違点　その１
>
> ユーザ定義（Database Fetch）では必ずSELECT文を定義するため、「クエリ種別」の設定項目はありません。

### 入力値/出力値

次に、作成するユーザ定義を呼び出す際の入力値と、呼び出しが完了した際に返却する出力値を定義します。
ユーザ定義（Database Fetch）の入力値/出力値の初期値は以下の通りです。




図：ユーザ定義（Database Fetch）の入力値/出力値の初期値

入力値

ユーザ定義（Database Fetch）の入力値は開発者が決定する必要があります。
本チュートリアルでは、入力値として「ユーザ検索用の文字列」を以下のように定義します。

| 入力値 | 説明 |
|---|---|
| search_word<string> | ユーザ検索用の文字列を格納します。 |

出力値

ユーザ定義（Database Fetch）の出力値はシステムによって固定です。
IM-LogicDesignerではユーザ定義（Database Fetch）の出力値を後述するクエリ設定の内容を元に自動で決定します。
詳細は「*データ定義取得と、出力値への反映*」を参照してください。

> **コラム**
>
> ユーザ定義（SQL）との相違点　その２
>
> ユーザ定義（Database Fetch）では以下の出力値は提供されません。
>
> - count<integer> - 取得された値の総数。
> - query<string> - 実行されたクエリ。

### クエリ設定

次に、実際に取得処理を行うクエリ（SELECT文）の設定を行います。

想定するSELECT文

本チュートリアルで想定する処理は、以下のSQLによって表現されます。

```
SELECT
  user_cd
FROM
  imm_user
/*BEGIN*/
WHERE
  /*IF search_word != null*/
  user_cd LIKE /*search_word*/'dbf_user%'
  /*END*/
/*END*/
```

コラム

重複を除去する場合

今回利用するSELECT文では、多言語情報や期間情報を加味しないため、一人の該当ユーザに対して複数の結果（ユーザコード）が返る場合があります。
本チュートリアルのユーザ定義を実際に作成、動作確認を行う場合、必要に応じてdistinctなどの重複を除外する指定を行ってください。

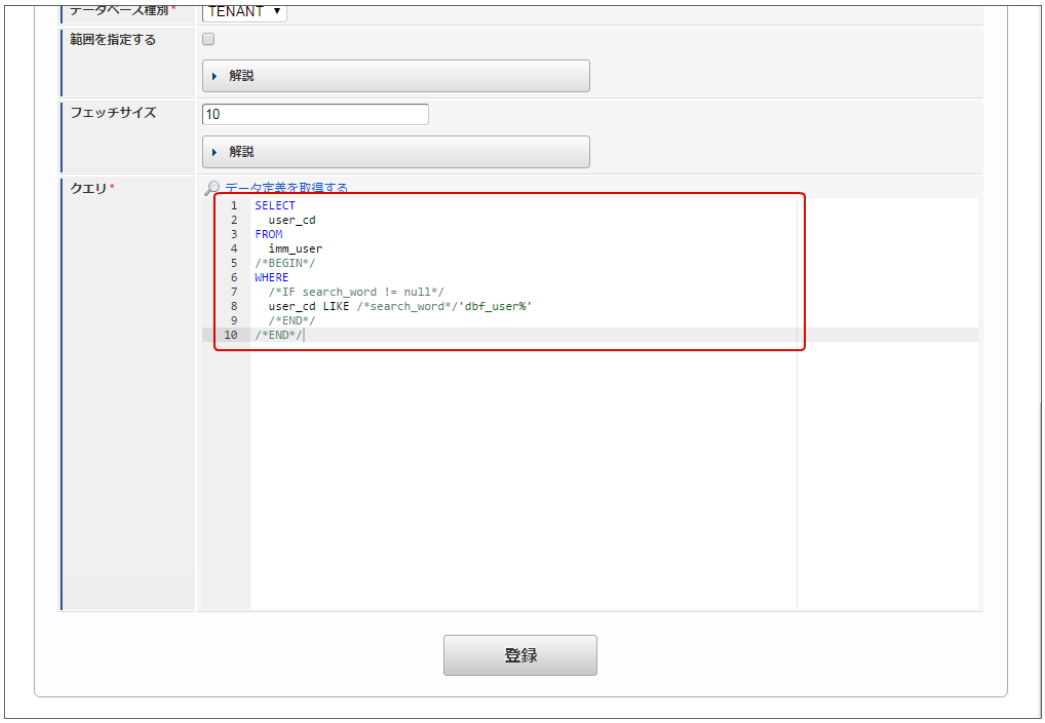
以下は指定例です。

```
SELECT
  distinct user_cd -- distinctの指定
FROM
/* 以降は同様 */
```

データ定義取得と、出力値への反映

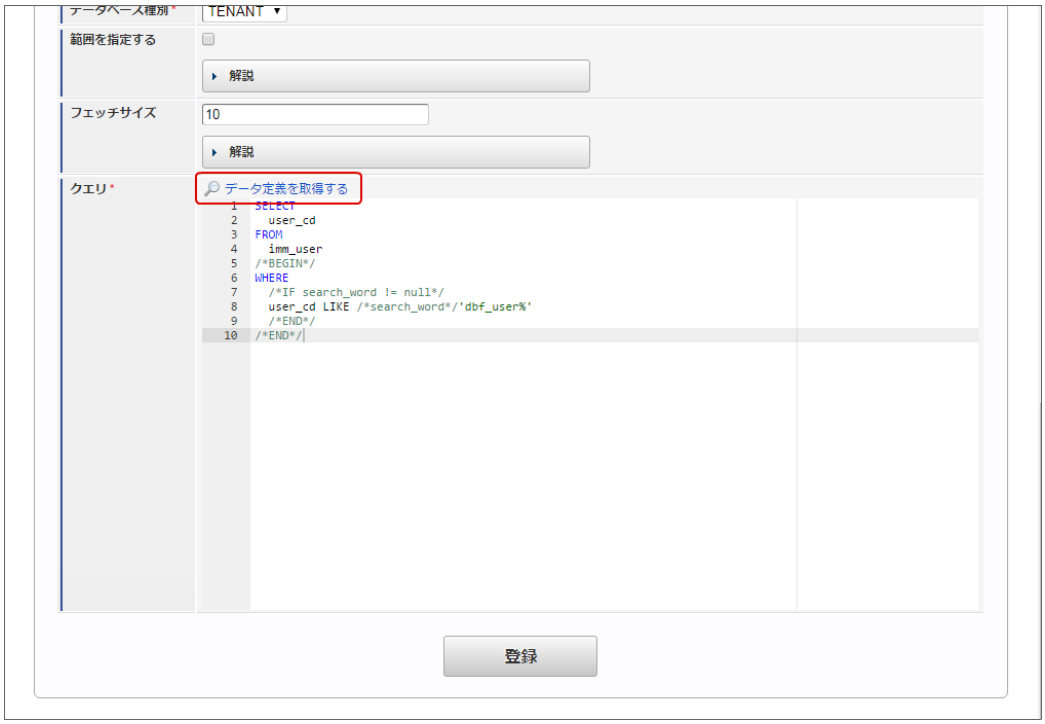
「*想定するSELECT文*」の定義に合わせて、出力値の設定を行います。
IM-LogicDesignerでは、設定されたクエリを検証し、クエリの記述内容にあった出力値を自動で設定する機能を提供しています。

1. 「Database Fetch定義編集」画面の「クエリ」にSQLを定義します。



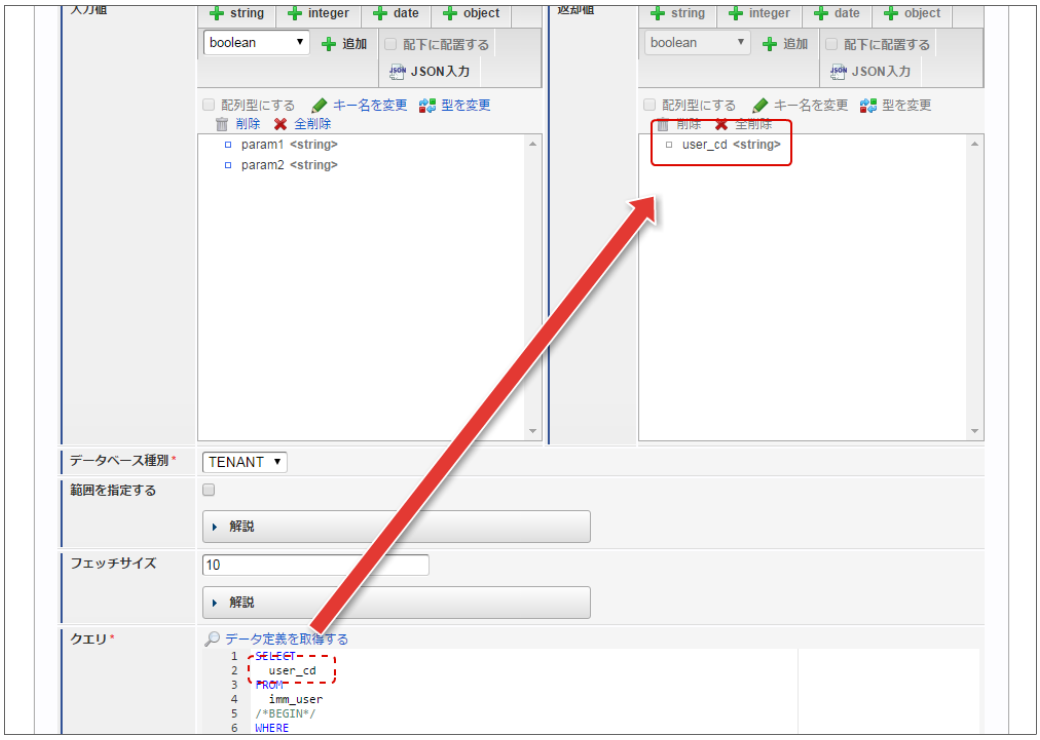

図：クエリへSELECT文の定義

2. 「クエリ」入力欄の上部にある「データ定義を取得する」をクリックします。




図：データ定義を取得

3. 「クエリ」に定義されたSQLの検証結果が、「出力値」に反映されます。

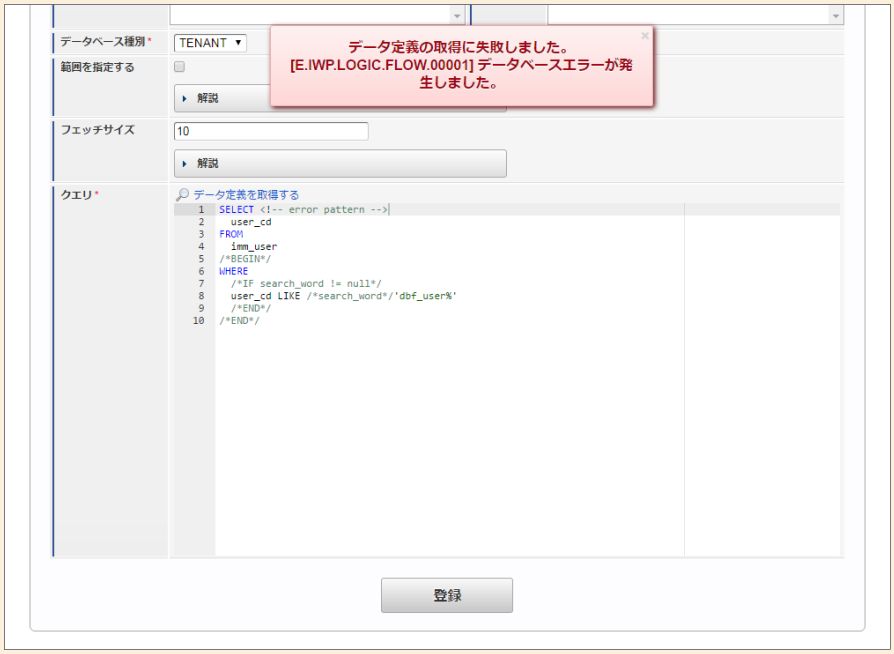



図：クエリをもとに出力値へ反映

注意

出力値を反映される場合の注意点

定義されたSQLがシンタックスエラー等実行不可なものであった場合、以下のエラーメッセージが表示され、出力値への反映は行われません。




図：SQLが実行不可であった場合のエラーメッセージ

コラム

ワイルドカード（アスタリスク）使用時の出力値

定義されたSQLが、取得対象にワイルドカード（アスタリスク）を利用した場合、IM-LogicDesignerは出力値として対象テーブルに定義された全てのカラムを出力値対象とします。

### 取得範囲の指定

次に、取得範囲の指定を必要に応じて行います。

本チュートリアルで作成したユーザ定義は、検索条件に合致するユーザを全て取得します。
取得処理には問題はありませんが、実際の要件や課題の解決において検索条件に合致するユーザの一部を取得したい場合が多く存在します。
ユーザ定義（Database Fetch）では、これに対して取得範囲を指定したSQLの定義をサポートしています。

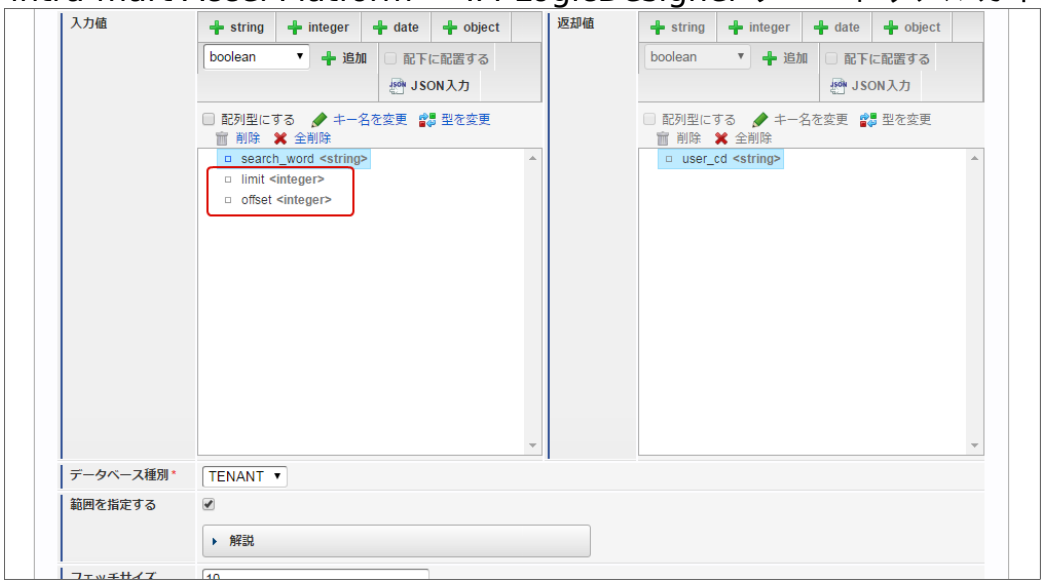
取得範囲の指定を行うSQLを定義するには、Database Fetch定義編集の項目を以下のとおりに設定します。

- 範囲を指定する - `チェックボックス：オン`


図：取得範囲の設定

取得範囲の指定を行うことによって、ユーザ定義（Database Fetch）は自動的に以下の設定を行います。

- 「入力値」へ、範囲を指定するために必要なパラメータの追加
- 「クエリ」に定義されたSQLへ、取得範囲を指定する構文の追加




図：追加されたパラメータ

範囲指定を行うパラメータの詳細は「IM-LogicDesigner仕様書」 - 「ユーザ定義タスク」 - 「Database Fetch」を参照してください。

> **注意**
>
> 追加される取得範囲を指定する構文について
>
> 取得範囲の指定を行うことによって追加される構文は、暗黙的に行われます。
> ユーザ定義（Database Fetch）は取得範囲の指定がされていた場合、定義された「クエリ」に取得範囲の指定を行う構文を追加したうえで最終的な処理を行います。
>
> そのため、取得範囲の指定を行った場合でも、「Database Fetch定義編集」画面の「クエリ」に定義されたSQLは変更されません。
> また、「クエリ」に直接取得範囲の指定を行う構文を定義していた場合はSQL不正でエラーが発生します。

### フェッチサイズの指定

次に、フェッチサイズの指定を行います。

> **コラム**
>
> ユーザ定義（SQL）との相違点　その３
>
> フェッチサイズの指定はユーザ定義（Database Fetch）でだけ設定可能な項目です。

パフォーマンス向上などを目的とした実際の要件や課題に合わせてフェッチサイズの変更を行う場合、この値を変更してください。
なお、ユーザ定義（Database Fetch）では、デフォルトのフェッチサイズを`10`としています。


図：フェッチサイズの指定（デフォルト値）

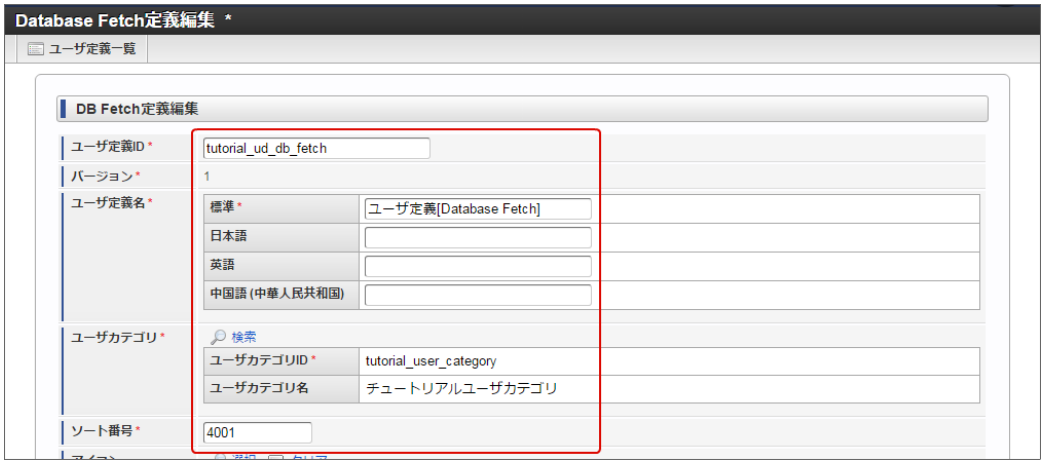
本チュートリアルではフェッチサイズにデフォルト値である`10`をそのまま指定します。

> **注意**
>
> 適切なフェッチサイズの指定について
>
> フェッチサイズを大きく設定すると、データベースサーバとの通信回数が削減され、パフォーマンス向上が期待できますが、メモリ使用量が増大します。
> フェッチサイズの最適値は、サーバ環境や取得対象となるテーブルのレコード件数、または、カラムサイズ等によって異なることに留意してください。

ユーザ定義（Database Fetch）を作成する。

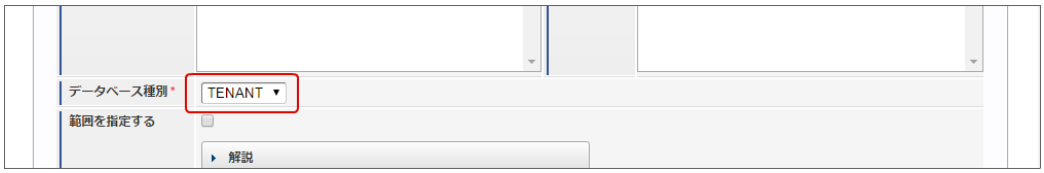
最後に、これまでの内容を踏まえてユーザ定義（Database Fetch）を作成します。

1. 「Database Fetch定義編集」画面を表示します。
2. ユーザ定義の基本情報となる各項目に以下の値を入力します。
   - ユーザ定義ID「tutorial_ud_db_fetch」
   - バージョン「1」（固定）
   - ユーザ定義名
     - 標準 - 「ユーザ定義[Database Fetch]」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし
   - カテゴリ
     - カテゴリID - 「tutorial_user_category」
   - ソート番号「4001」



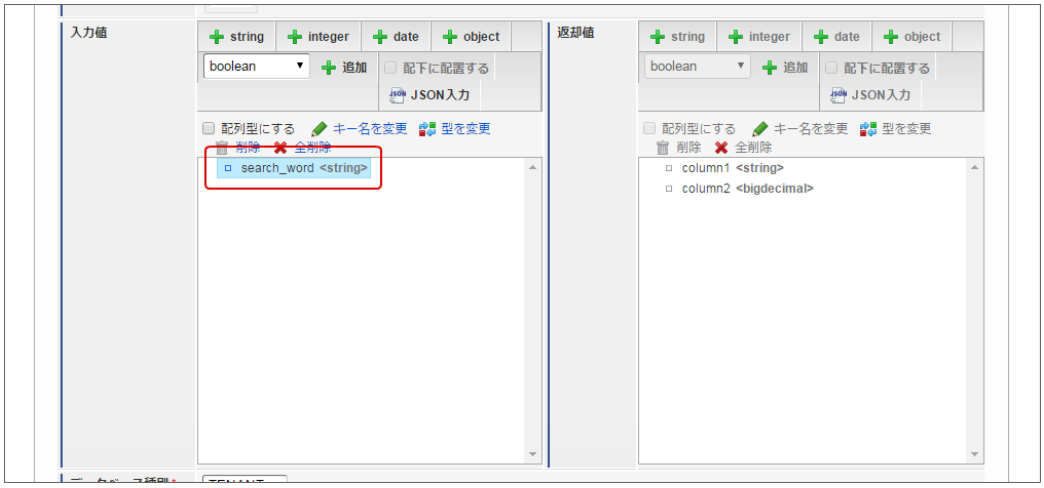

図：基本情報の定義

3. データベース種別を「*データベース種別の選択*」をもとに設定します。



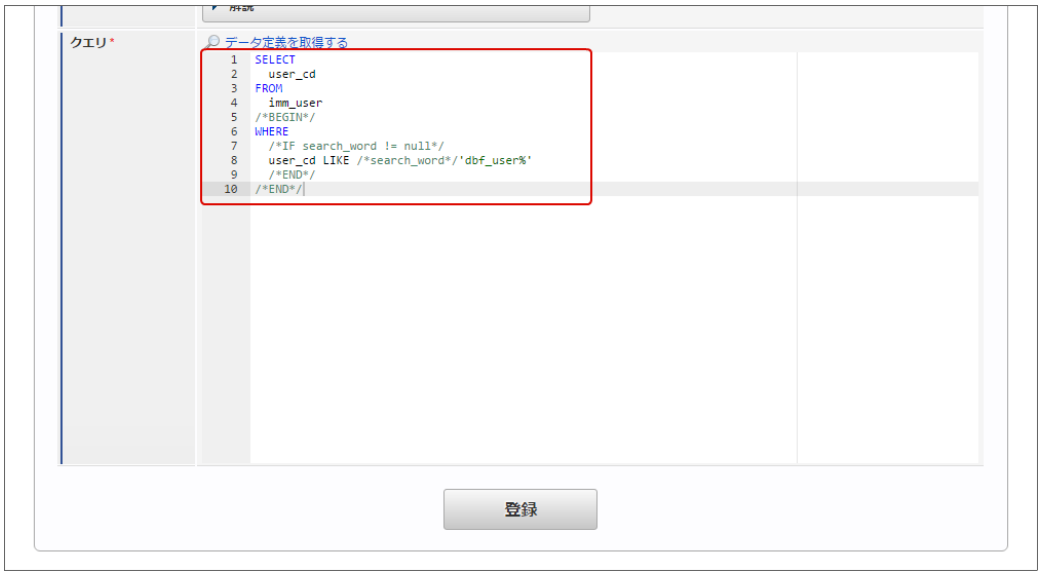

図：データベース種別の定義

4. 入力値を「*入力値/出力値*」をもとに値を設定します。



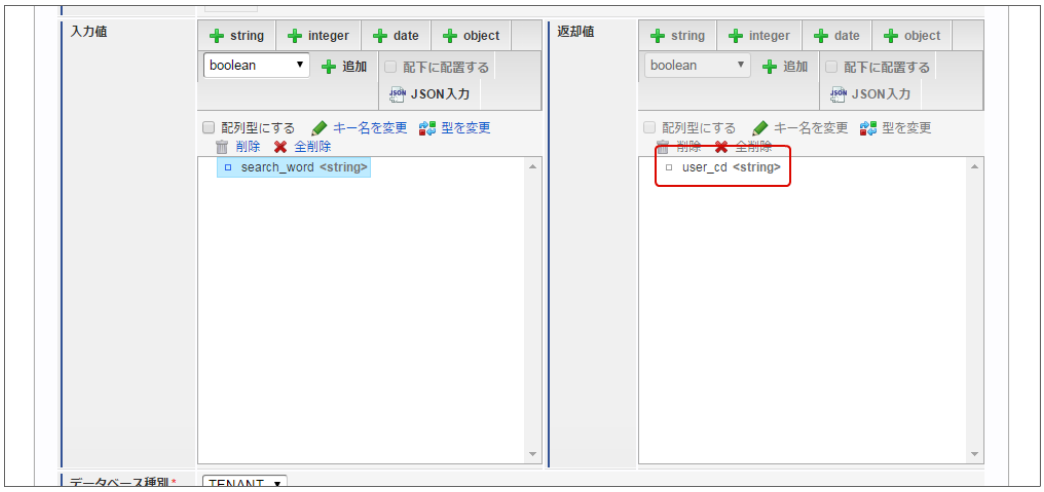

図：入力値の定義

5. クエリに「*想定するSELECT文*」で提示したSQLを設定します。



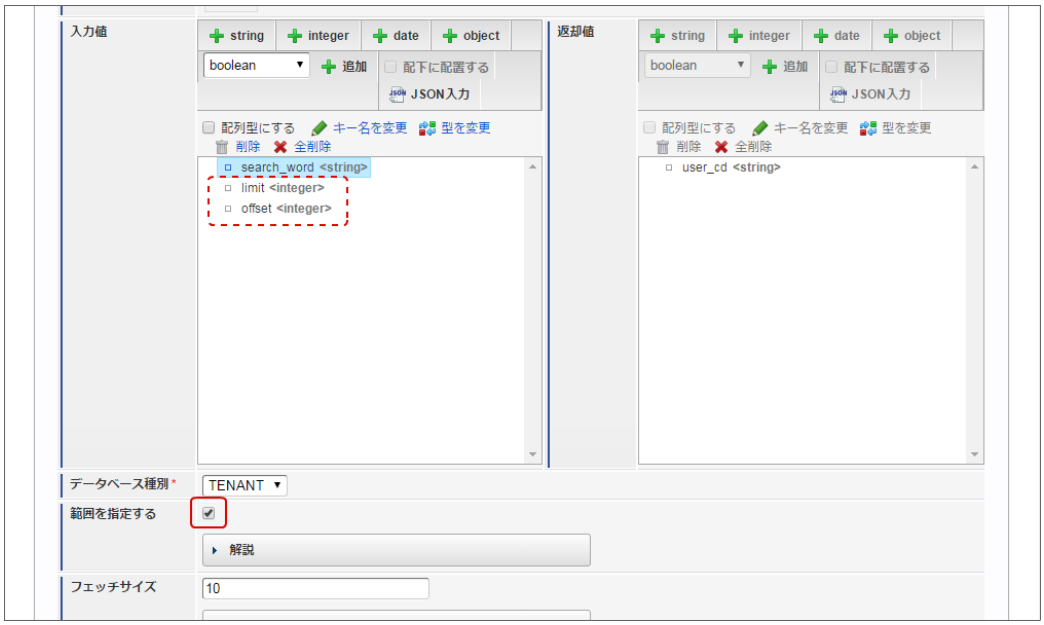

図：クエリの定義

6. 「*データ定義取得と、出力値への反映*」をもとに、出力値を設定します。



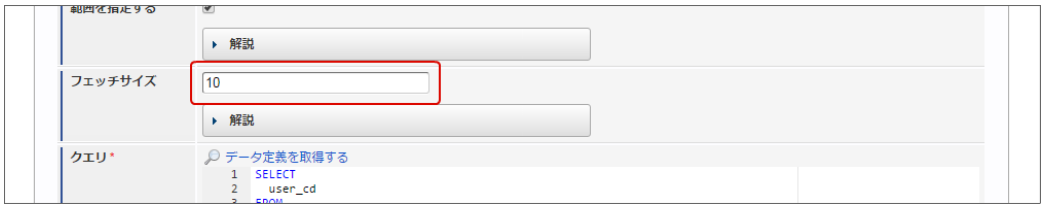

図：出力値の反映

7. 「*取得範囲の指定*」をもとに、取得範囲の指定を行います。




図：取得範囲の指定

8. 「*フェッチサイズの指定*」をもとに、フェッチサイズの指定を行います。




図：フェッチサイズの指定

9. 「登録」をクリックします。


図：登録

以上で、ユーザ定義（Database Fetch）の作成が完了しました。

## ユーザ定義 - CSV Fetch

### ユーザ定義（CSV Fetch）

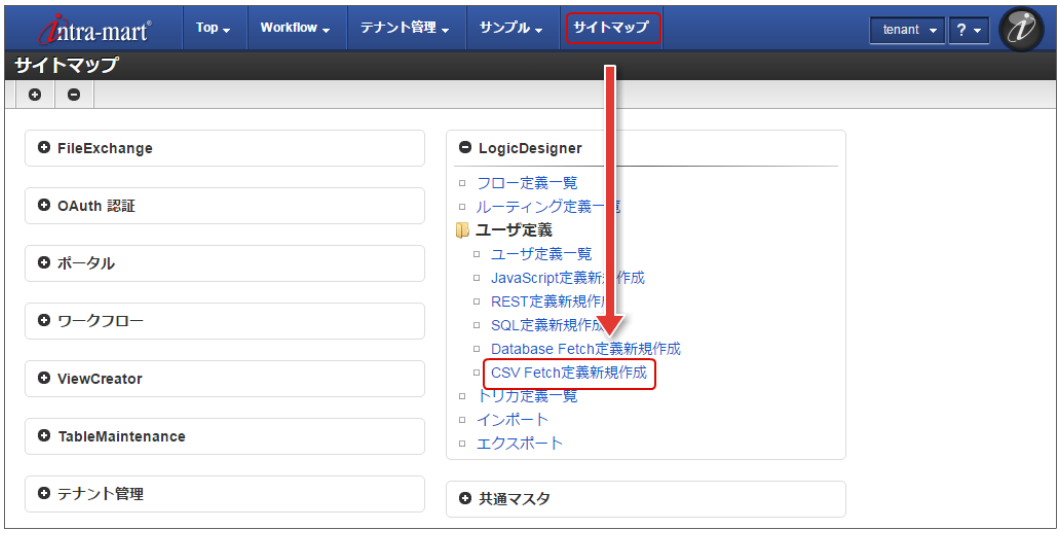
ユーザ定義（CSV Fetch）とは、CSVファイル上の大量のレコードを効率的に読み込み・処理することが可能なユーザ定義の一つです。
タスクの入力値と処理対象のCSVファイルを紐付け、出力値として読み込んだレコードの各要素を繰り返しの形で扱うことができます。
ユーザ定義（CSV Fetch）は「繰り返し」制御要素と同じ、開始/終了をペアで持つタスクを提供します。

#### ユーザ定義（CSV Fetch）の作成画面への遷移手順

ユーザ定義（CSV Fetch）作成画面へは、サイトマップメニューとユーザ定義一覧画面から遷移できます。

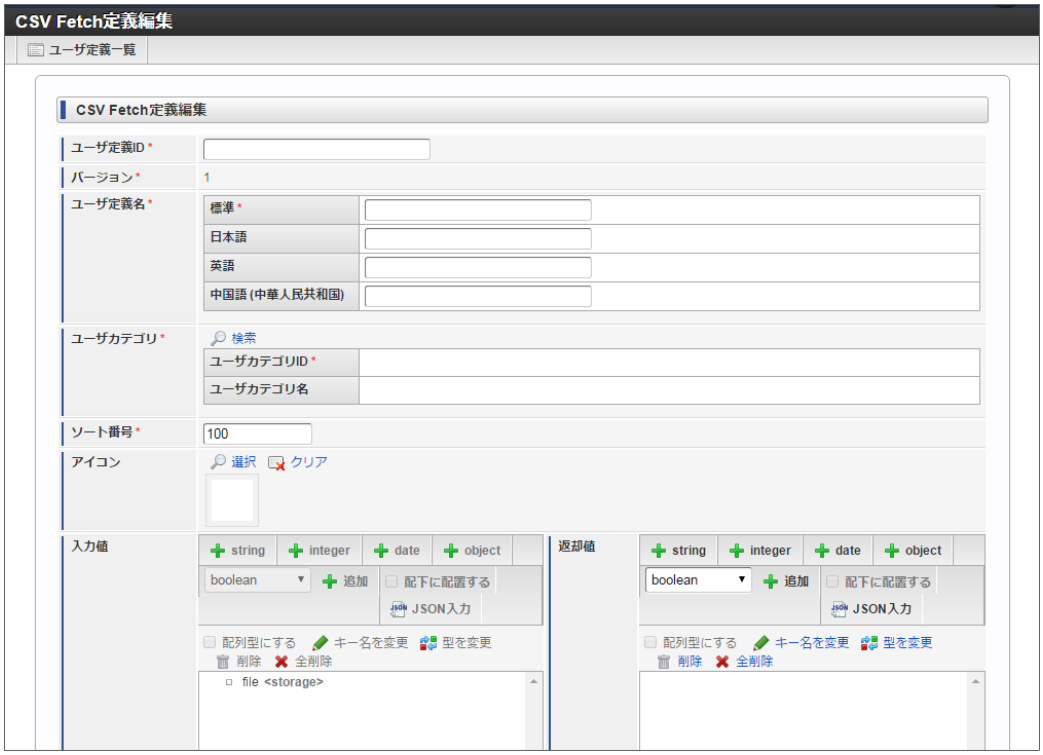
メニューから作成画面へ遷移する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義」配下から、「CSV Fetch定義新規作成」をクリックします。



図：サイトマップ - ユーザ定義

2. 「CSV Fetch定義編集」画面が表示されます。




図：CSV Fetch定義編集画面

ユーザ定義一覧画面から作成画面へ遷移する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義一覧」から、ユーザ定義一覧を開きます。
2. 画面上部のヘッダ内、「新規作成」をクリックし、ポップアップしたメニューから「CSV Fetch定義新規作成」をクリックします。


図：ユーザ定義一覧 - CSV Fetch定義新規作成

3. 「CSV Fetch定義編集」画面が表示されます。

## ユーザ定義（CSV Fetch）の詳細

この章では、CSV Fetchを利用したユーザ定義の作成方法とその詳細について説明します。



### 本チュートリアルで作成する概要

本チュートリアルでは、シンプルなユーザ情報を列挙したCSVファイルを定義し、そのCSVファイルを取り扱うユーザ定義（CSV Fetch）の実装を通して、CSV Fetchを利用したユーザ定義の作成方法とその詳細を説明します。

本チュートリアルで定義するCSVファイルの詳細

本チュートリアルで扱うユーザは、以下の情報（プロパティ）を持つものと定義します。

- ユーザ名（user_name）
- 年齢（age）
- 誕生日（birth_date）
- 備考（description）

加えて、本チュートリアルで扱うCSVファイルは、以下の仕様に沿って定義します。

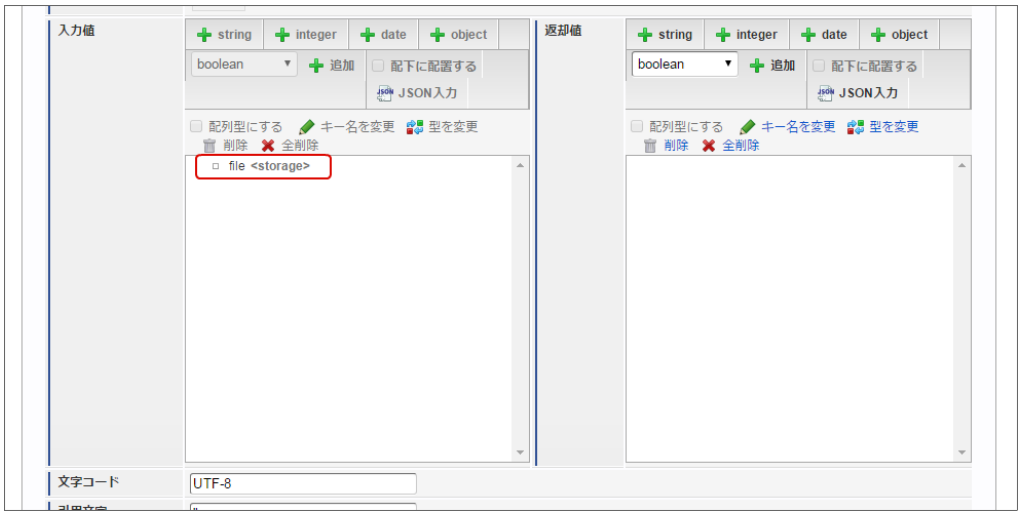
- 1行目はヘッダ行とする。
- 各フィールドの値はダブルクォーテーション（"～"）で括る。
- フィールド間はカンマ（,）で区切る。

上記の定義を踏まえて、本チュートリアルで定義するユーザ情報のCSVファイルのサンプルは以下の通りです。

```
user_name, age, birth_date, description
"csv_user00001","20","2016-08-01T00:00:00Z","CSV User No. 00001"
"csv_user00002","21","2016-08-01T00:00:00Z","CSV User No. 00002"
"csv_user00003","22","2016-08-01T00:00:00Z","CSV User No. 00003"
"csv_user00004","23","2016-08-01T00:00:00Z","CSV User No. 00004"
"csv_user00005","24","2016-08-01T00:00:00Z","CSV User No. 00005"
...
"csv_user99998","28","2016-08-01T00:00:00Z","CSV User No. 99998"
"csv_user99999","29","2016-08-01T00:00:00Z","CSV User No. 99999"
```

### 入力値/出力値

はじめに、作成するユーザ定義を呼び出す際の入力値と、呼び出しが完了した際に返却する出力値を定義します。
ユーザ定義（CSV Fetch）の入力値/出力値の初期値は以下の通りです。




図：入力値、および、出力値の初期値

入力値

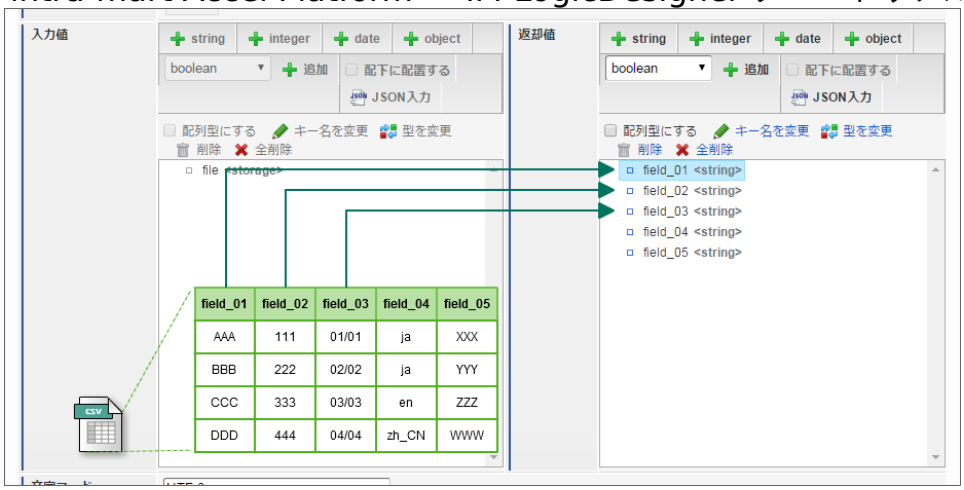
ユーザ定義（CSV Fetch）の入力値はシステムによって固定です。
IM-LogicDesignerではCSVファイルを取り扱うにあたり、以下の内容を入力値として定義しています。

| 入力値 | 説明 |
| --- | --- |
| file<storage> | データを読み込むCSVファイルのアドレスを示すストレージ情報 |

出力値

ユーザ定義（CSV Fetch）では出力値と、読み込んだCSVファイルの各フィールド値と紐付けます。

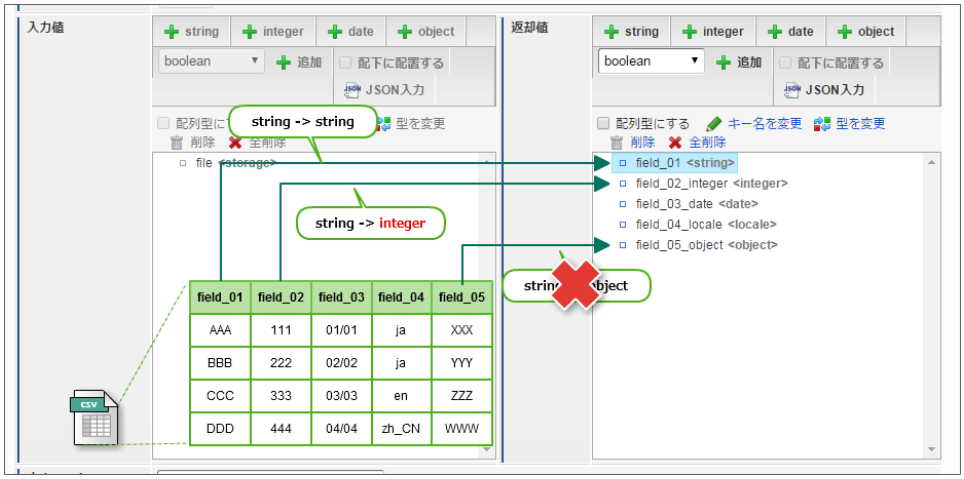
より具体的にはユーザ定義（CSV Fetch）では、CSVファイルの1番目のフィールドと、出力値の1番目の定義値とを紐付けます。
以後同様にCSVファイルのN番目のフィールドと出力値のN番目の定義値が紐づけます。




図：CSVファイルのフィールドと出力値の紐付け

> **コラム**
>
> CSVファイルのヘッダ名と出力値名
>
> 出力値との紐付けを行う際に、CSVファイルにヘッダが指定されていても動作は変わりません。
> CSVファイルのヘッダ名と出力値名が同一であった場合でも、紐付けの順番は変わらないことに注意してください。

またユーザ定義（CSV Fetch）では、読み込んだCSVファイルの各フィールドの値を全て文字列（string）として扱います。
出力値と紐付ける際に、紐付け先の出力値の型がstring以外だった場合、IM-LogicDesignerは自動的に型変換を行います。
（そのため、stringから変換することの出来ない型を出力値に定義した場合は、エラーとして扱います）




図：CSVファイルのフィールドと出力値の型変換

本チュートリアルでは、「*本チュートリアルで作成する概要*」を元に出力値を以下のように定義します。

| 出力値 | 説明 |
|---|---|
| name<string> | ユーザ名（user_name）フィールドを格納します。 |
| age<integer> | 年齢（age）フィールドを格納します。 |
| birth<date> | 誕生日（birth_date）フィールドを格納します。 |
| desc<string> | 備考（description）フィールドを格納します。 |

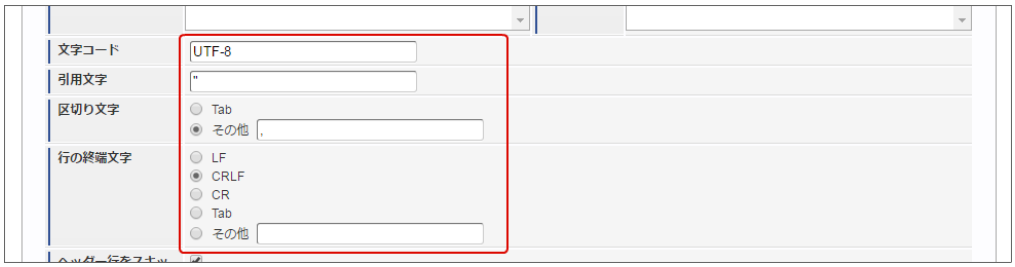
以上で、出力値の設定が完了しました。

### フォーマット設定

次に、作成するユーザ定義（CSV Fetch）が読み込むCSVファイルのフォーマット情報を設定します。
フォーマット設定を「*本チュートリアルで作成する概要*」を元に以下のとおりに定義します。

- **文字コード**
  - UTF-8
- **引用文字**

    - "（ダブルクォート）
- **区切り文字**
    - ,（カンマ）
- **行の終端文字**
    - CRLF



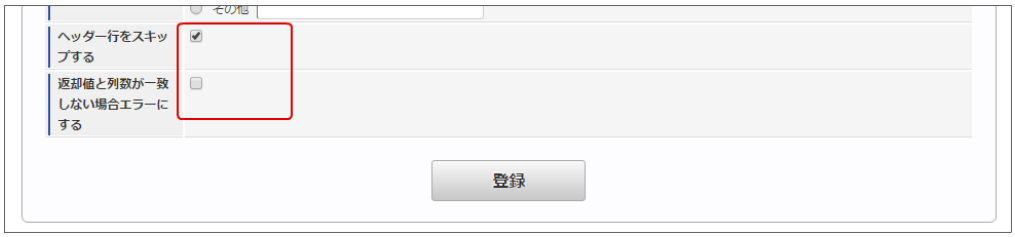

図：フォーマット設定

以上で、フォーマット設定が完了しました。

### 処理設定

次に、作成するユーザ定義（CSV Fetch）の処理時の振る舞いを設定します。
処理設定の定義内容は以下の通りです。

- **ヘッダ行をスキップする**
    - チェックボックス：オン
- **返却値と列数が一致しない場合エラーにする**
    - チェックボックス：オフ




図：処理設定

以上で、処理設定が完了しました。

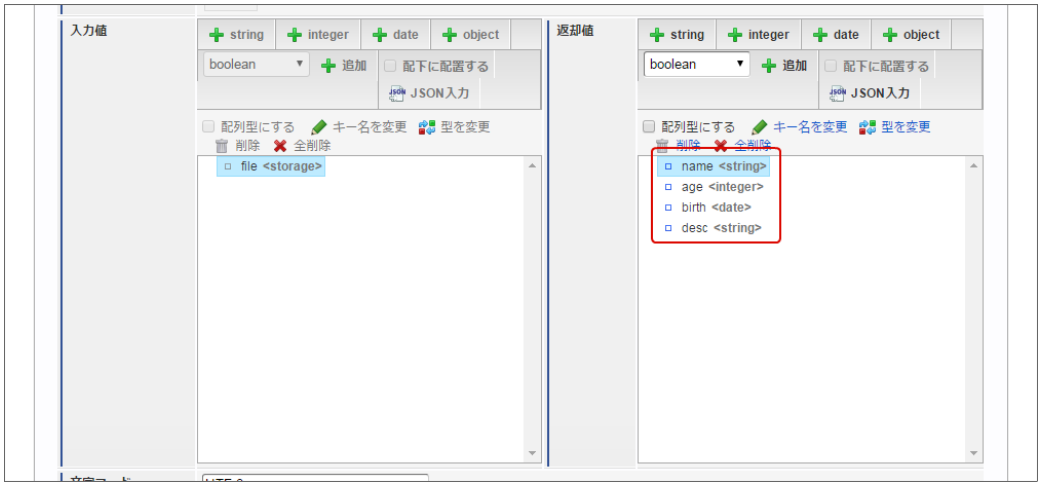
### ユーザ定義（CSV Fetch）を作成する。

最後に、これまでの内容を踏まえてユーザ定義（CSV Fetch）を作成します。

1. 「CSV Fetch定義編集」画面を表示します。
2. ユーザ定義の基本情報となる各項目に以下の値を入力します。
    - ユーザ定義ID「tutorial_ud_csv_fetch」
    - バージョン「1」（固定）
    - ユーザ定義名
        - 標準 - 「ユーザ定義[CSV Fetch]」
        - 日本語、英語、中国語（中華人民共和国） - 入力なし
    - カテゴリ
        - カテゴリID - 「tutorial_user_category」
    - ソート番号「5001」

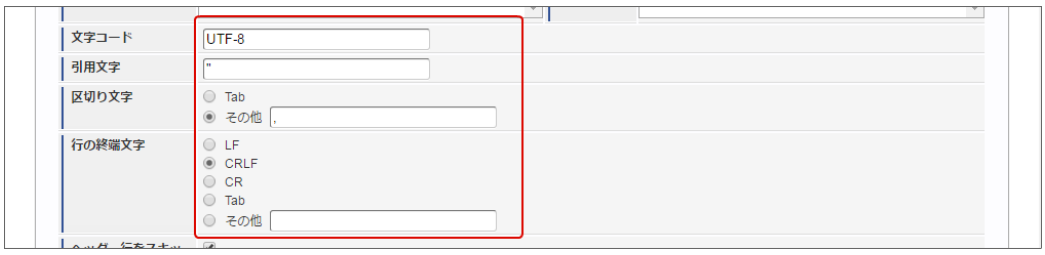

図：基本情報の定義

3. 入力値、および、出力値を「*入力値/出力値*」をもとに定義します。



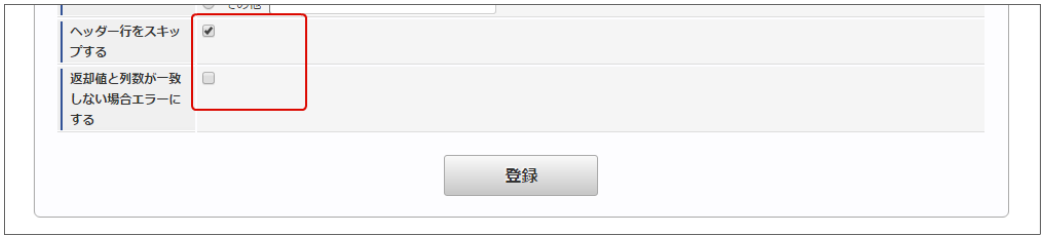

図：入出力値の定義

4. フォーマット設定を「*フォーマット設定*」をもとに定義します。




図：フォーマット設定の定義

5. 処理設定を「*処理設定*」をもとに定義します。




図：処理設定の定義

6. 「登録」をクリックします。


図：登録

以上で、ユーザ定義（CSV Fetch）の作成が完了しました。

## ユーザ定義 - テンプレート定義

### ユーザ定義（テンプレート）

ユーザ定義（テンプレート）とは、文字列テンプレートを用い、入力に応じて動的に文書を生成するタスクを定義することが可能なユーザ定義の一つです。
タスクの入力値と文字列テンプレートを用いて文書の生成を行い、生成結果の文字列を出力値として扱うことができます。
また、多言語（日本語・英語・中国語（中華人民共和国））に標準で対応しています。

#### ユーザ定義（テンプレート）の作成画面への遷移手順

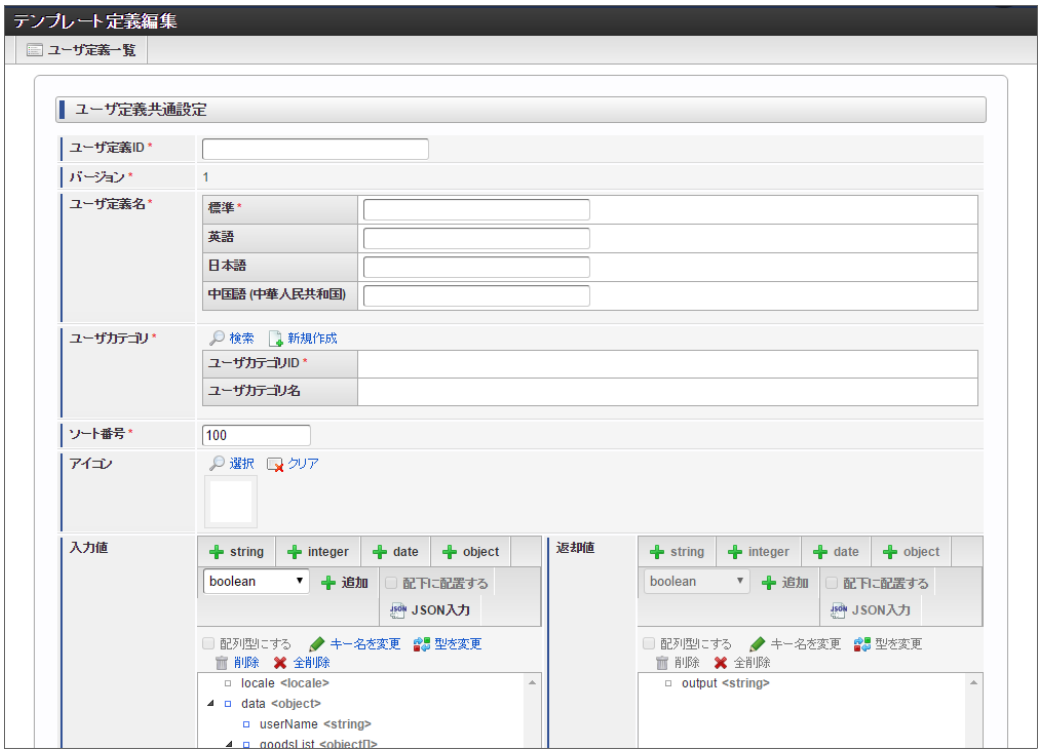
ユーザ定義（テンプレート）作成画面へは、サイトマップメニューとユーザ定義一覧画面から遷移できます。

メニューから作成画面へ遷移する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義」配下から、「テンプレート定義新規作成」をクリックします。


図：サイトマップ - ユーザ定義

2. 「テンプレート定義編集」画面が表示されます。




図：テンプレート定義編集画面

ユーザ定義一覧画面から作成画面へ遷移する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義一覧」から、ユーザ定義一覧を開きます。
2. 画面上部のヘッダ内、「新規作成」をクリックし、ポップアップしたメニューから「テンプレート定義新規作成」をクリックします。


図：ユーザ定義一覧 - テンプレート定義新規作成

3. 「テンプレート定義編集」画面が表示されます。

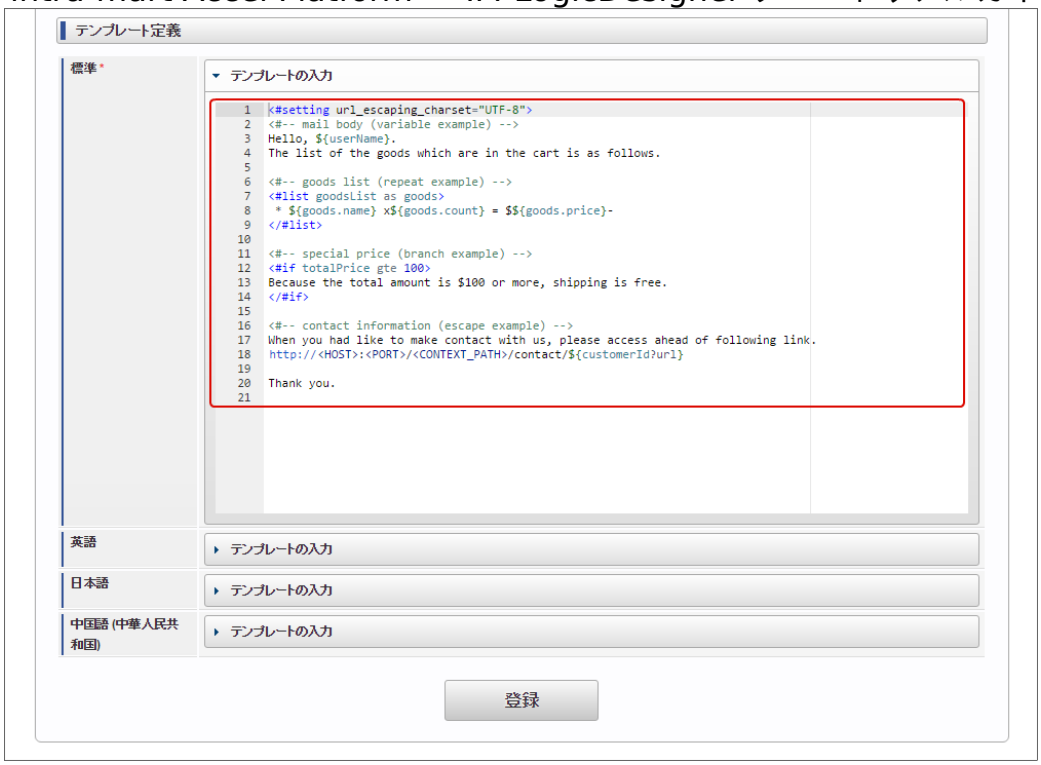
## 初期サンプルから見るユーザ定義（テンプレート）の詳細

この章では、テンプレートを利用したユーザ定義の作成方法を、IM-LogicDesignerが提供する初期サンプルをもとに説明します。



### 初期サンプルについて

本章で扱う「初期サンプル」とは、「*ユーザ定義（テンプレート）の作成画面への遷移手順*」に基いて「テンプレート定義編集」画面に遷移した際、初めから定義されているテンプレート、および、入力値/出力値のことを指します。




図：初期表示時、サンプルテンプレートが定義されている

初期表示時に定義されているテンプレートは以下の通りです。

```
<#setting url_escaping_charset="UTF-8">
<#-- mail body (variable example) -->
Hello, ${userName}.
The list of the goods which are in the cart is as follows.

<#-- goods list (repeat example) -->
<#list goodsList as goods>
 * ${goods.name} x${goods.count} = $${goods.price}-
</#list>

<#-- special price (branch example) -->
<#if totalPrice gte 100>
Because the total amount is $100 or more, shipping is free.
</#if>

<#-- contact information (escape example) -->
When you had like to make contact with us, please access ahead of following link.
http://<HOST>:<PORT>/<CONTEXT_PATH>/contact/${customerId?url}

Thank you.
```

初期表示時のテンプレートには、利用頻度の高い以下の構文が含まれています。

- 文字列の置換
- 繰り返し
- 条件分岐
- エスケープ処理

### テンプレート定義

はじめに、テンプレートの定義方法と留意点を説明します。

テンプレートの記法/構文

ユーザ定義（テンプレート）では、テンプレートの記法として**FreeMarker Template Language（FTL）**を利用します。

> **コラム**
>
> FTLの詳細について
>
> FTLの詳細については、「IM-LogicDesigner仕様書」 - 「ユーザ定義タスク」 - 「テンプレート」を参照ください。
> また、仕様書で説明している以外の式、および、ディレクティブについては「Template Language Reference」を参照してください。

ユーザ定義（テンプレート）の初期サンプルでは、物品購入に関するメール本文を想定したテンプレートを定義しています。
ここでは、各段落についてそれぞれどの様な記法/構文が利用されているかを説明します。

式の利用

テンプレートの記法/構文のベースとなるのは式（補間式）です。
式では、「*${XXX}*」という構文で表現され、*XXX*で指定された変数や、後述する記法に応じて、内容を補間します。

以下の例では、式を利用して*userName*の値をテンプレートに埋め込んでいます。

```
<#setting url_escaping_charset="UTF-8">
<#-- mail body (variable example) -->
Hello, ${userName}.
The list of the goods which are in the cart is as follows.
```

繰り返しの利用

テンプレートでは複数の値を順次取り出して処理を行う、繰り返し処理が利用可能です。
繰り返し処理を行うには「*<#list LIST as ITEM></#list>*」というタグ形式を用います。
このタグ形式を、FTLではディレクティブと呼びます。

以下の例では、繰り返しを利用して*goodsList*に含まれる要素を*goods*という名称で扱っています。
そして*goods*に含まれる値を式（補間式）を利用してテンプレートに埋め込んでいます。

```
<#-- goods list (repeat example) -->
<#list goodsList as goods>
 * ${goods.name} x${goods.count} = $${goods.price}-
</#list>
```

条件文の利用

条件文は繰り返しと同様にディレクティブを利用します。
条件文のディレクティブは「*<#if CONDITION>* ～ *</#if>*」という形で表現します。

以下の例では、*totalPrice*の値が100以上（*gte* : greater than or equal）という条件文を定義しています。
そして、*totalPrice*が100以上の場合には、送料無料である文言が追加でテンプレートに埋め込まれます。

```
<#-- special price (branch example) -->
<#if totalPrice gte 100>
Because the total amount is $100 or more, shipping is free.
</#if>
```

ビルトイン式の利用

「*式の利用*」で説明した式（補間式）では、利用する値を内部でさらに処理を加えるビルトイン式の定義が可能です。
ビルトイン式は通常の式で指定する変数の最後に*?*を付与し、その後ろに具体的な処理を示す内容を指定します。

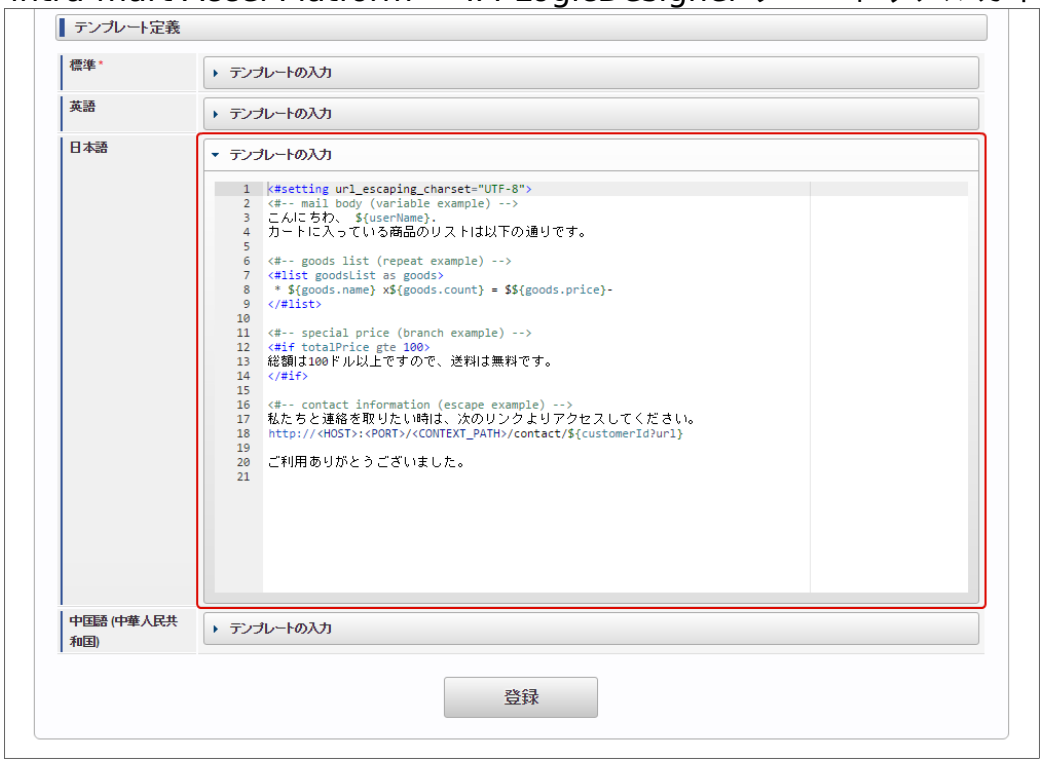
以下の例では、*customerId*に「*url*」というビルトイン式を定義しています。
*url*ビルトイン式は指定した値をURLエンコードします。

```
<#-- contact information (escape example) -->
When you had like to make contact with us, please access ahead of following link.
http://<HOST>:<PORT>/<CONTEXT_PATH>/contact/${customerId?url}
```

テンプレートの多言語化

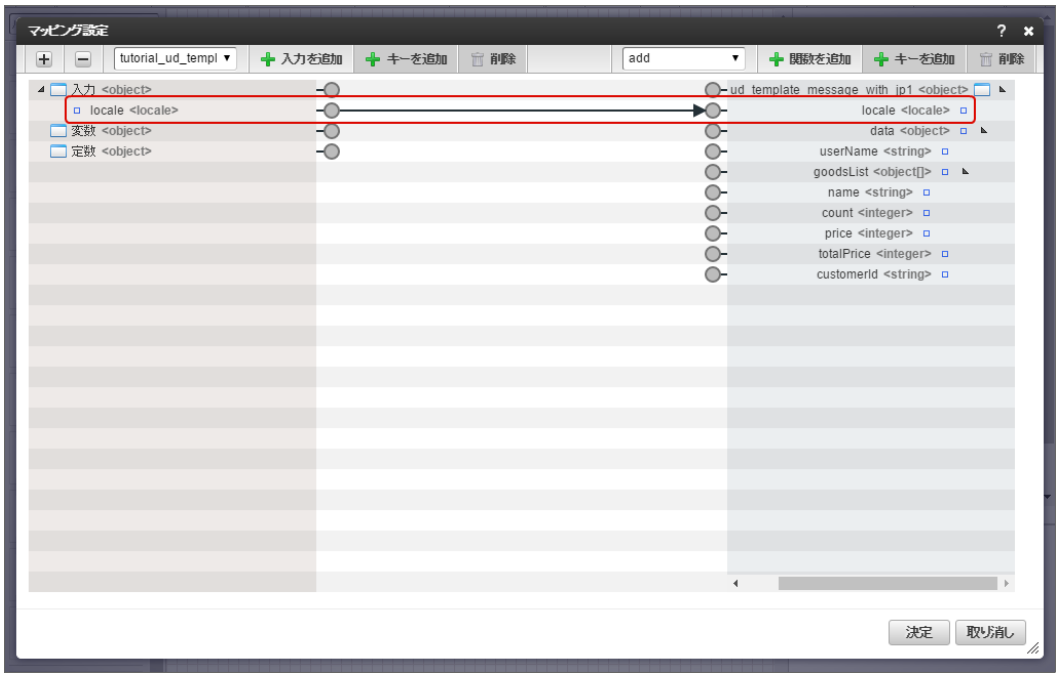
ユーザ定義（テンプレート）では、標準で多言語対応を行うための仕組みが用意されています。
多言語対応を行うには、はじめに各言語フィールドに言語に応じたテンプレートを定義します。

言語フィールドは「テンプレートの入力」をクリックすることで展開/格納されます。




図：日本語フィールドへの定義例

次に実際にユーザ定義（テンプレート）をロジックフロー上で利用する際に、多言語化の基準となるロケール情報をユーザ定義に渡します。
具体的には、ユーザ定義（テンプレート）が標準で提供する入力値である「*locale*」に多言語化の基準となるロケール情報をマッピングします。

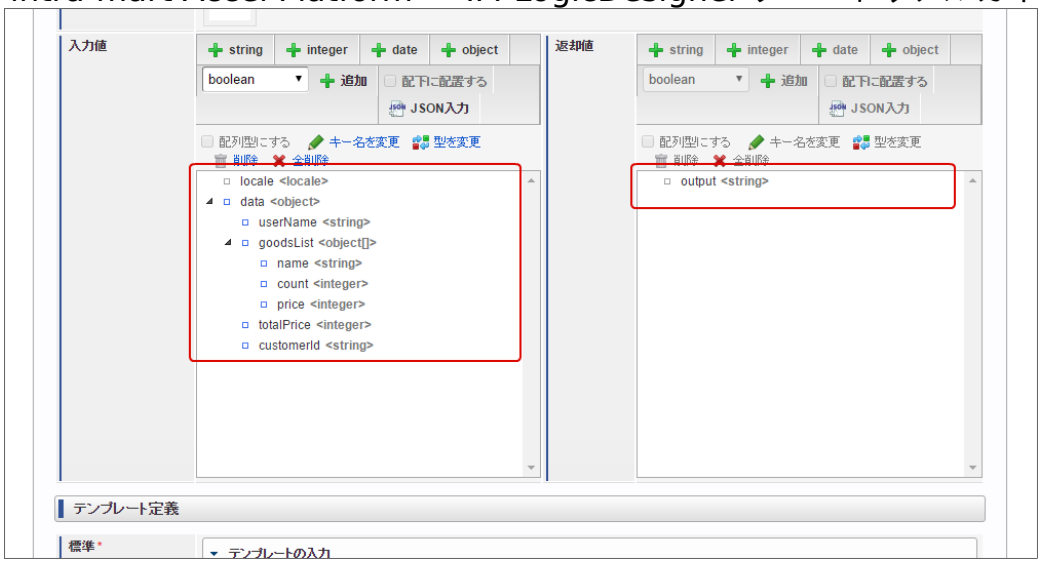



図：ロジックフローの入力値を利用する場合のロケール情報のマッピング例

対応するロケールの言語テンプレートが定義されていない場合、または、ロケールが指定されていない場合、テンプレートは「標準」が選択されます。

### 入力値/出力値

次に、作成するユーザ定義（テンプレート）を呼び出す際の入力値と、呼び出しが完了した際に返却する出力値の定義について説明します。




図：初期表示時の入力値、および、出力値

入力値

ユーザ定義（テンプレート）の入力値は、システムによって以下の二つが固定で定義されています。

| 入力値 | 説明 |
|---|---|
| locale<locale> | 利用するテンプレートのロケール（言語）の指定を行う入力値 |
| data<object> | テンプレートを定義する記法/構文内で利用する値の、親となる入力値 |

定義したテンプレートに「*テンプレートの記法/構文*」で紹介した方法で値を埋め込む場合には、data<object>配下に具体的な入力値を定義します。
初期サンプルではdata<object>配下に、以下の要素をテンプレートへの入力値として定義しています。

- userName<string> - ユーザ名
- goodsList<object[]> - 商品リスト
  - name<string>
  - count<integer>
  - price<integer>
- totalPrice<integer> - 合計金額
- customerId<string> - カスタマーID

注意点として、data<object>の配下以外に定義された値はテンプレート上での利用はできません。

出力値

ユーザ定義（テンプレート）の出力値は、システムによって固定です。
IM-LogicDesignerでは定義されたテンプレート、および、「*入力値*」をもとに、以下の内容をユーザ定義（テンプレート）の出力値として定義します。

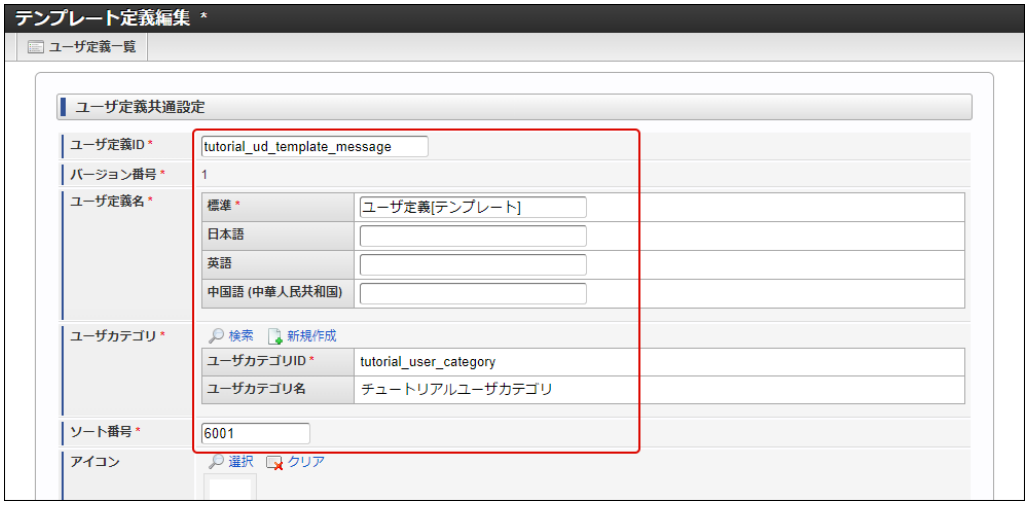
| 出力値 | 説明 |
|---|---|
| output<string> | テンプレート、および、入力値から生成された文字列 |

### ユーザ定義（テンプレート）を作成する。

最後に、これまでの内容を踏まえてユーザ定義（テンプレート）を作成します。

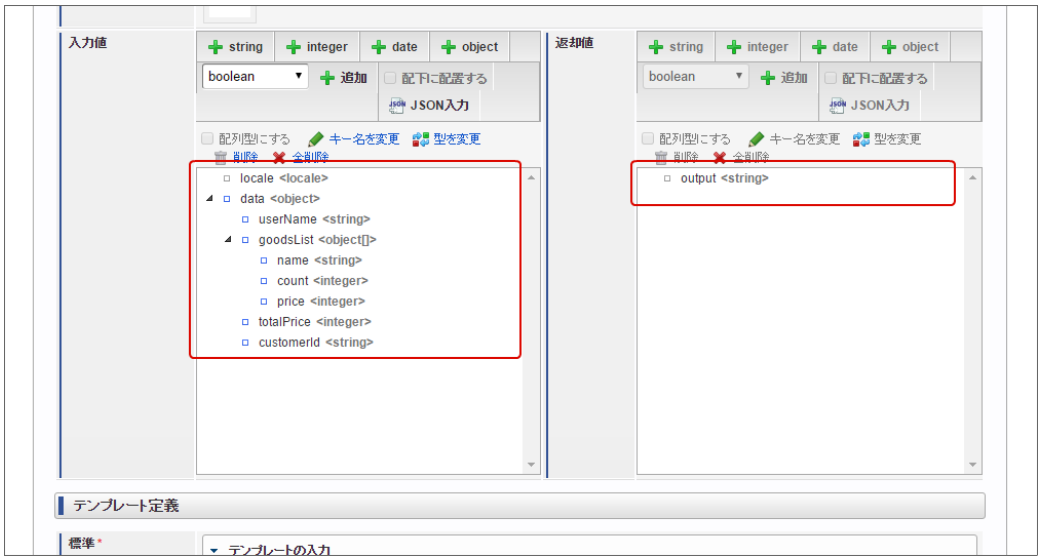
1. 「テンプレート定義編集」画面を表示します。
2. ユーザ定義の基本情報となる各項目に以下の値を入力します。
   - ユーザ定義ID「tutorial_ud_template_message」
   - バージョン「1」（固定）
   - ユーザ定義名
     - 標準 - 「ユーザ定義[テンプレート]」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし

- カテゴリ
  - カテゴリID - 「tutorial_user_category」
- ソート番号「6001」



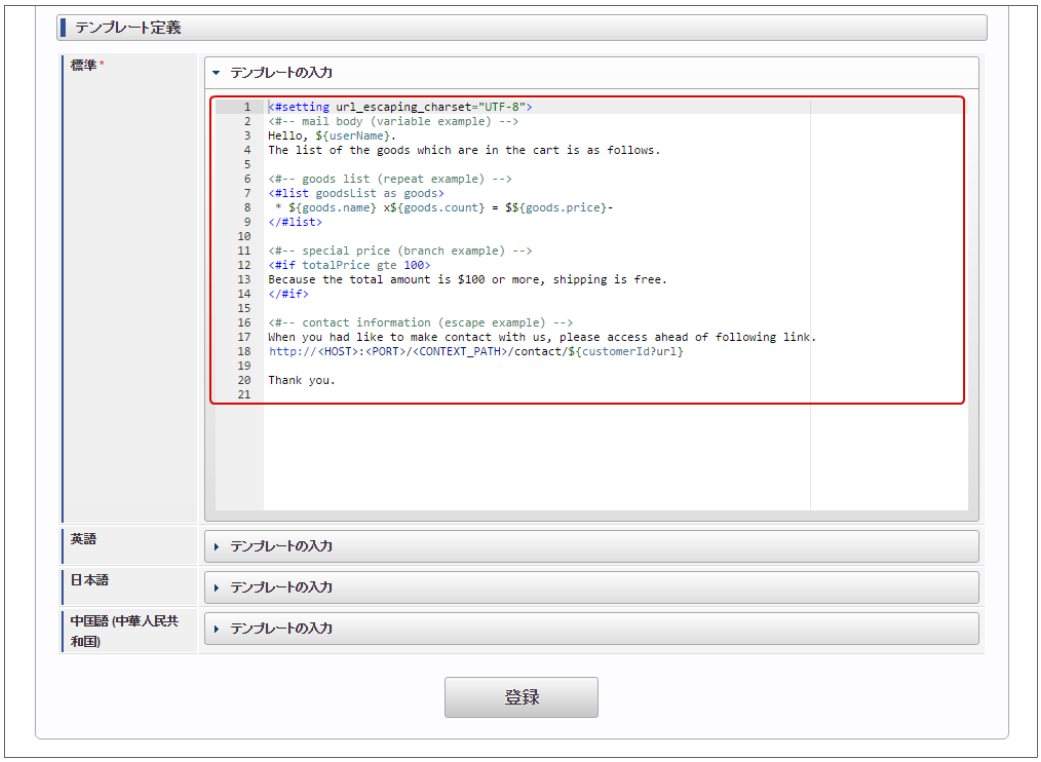

図：基本情報の入力

3. 入力値、および、出力値を「*入力値/出力値*」をもとに確認します。




図：入出力値の確認

4. テンプレートを「*テンプレート定義*」で提示されているテンプレート定義を元に確認します。




図：テンプレート定義の確認

5. 「登録」をクリックします。


図：登録

以上で、ユーザ定義（テンプレート）の作成が完了しました。

## ユーザ定義 - Excel入力

### ユーザ定義（Excel入力）

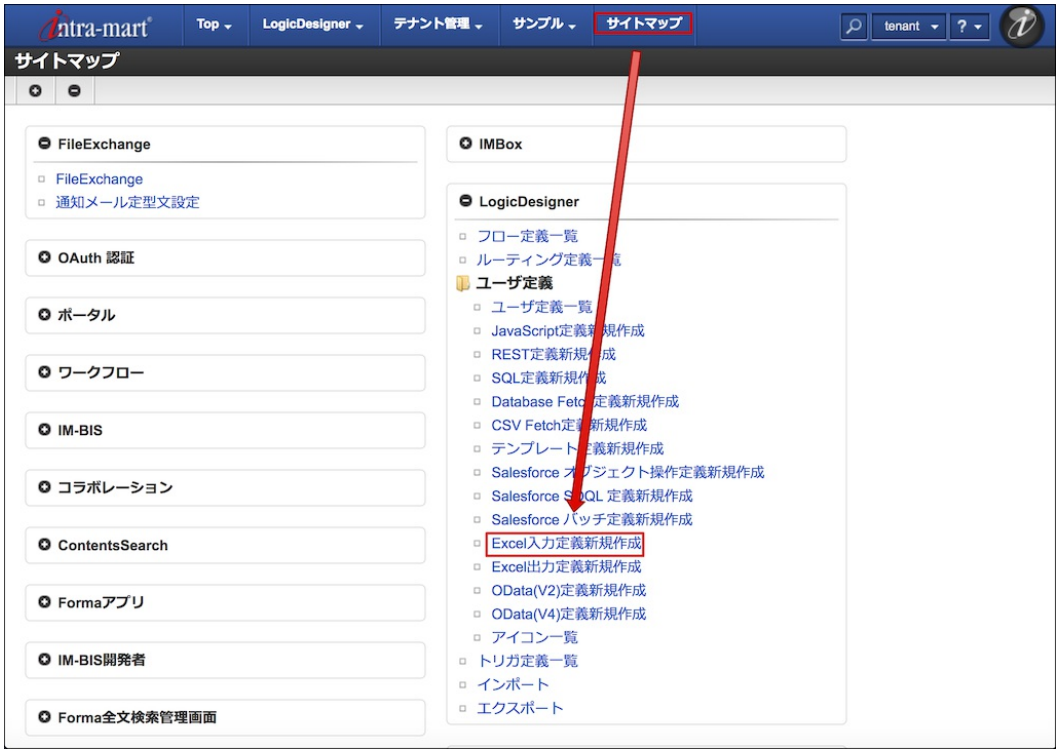
ユーザ定義（Excel入力）とは、Excelファイル上のレコードを効率的に読み込み・処理することが可能なユーザ定義の一つです。
タスクの入力値と処理対象のExcelファイルを紐付け、出力値として読み込んだレコードを出力値として扱うことができます。

ユーザ定義（Excel入力）の作成画面への遷移手順

ユーザ定義（Excel入力）作成画面へは、サイトマップメニューとユーザ定義一覧画面から遷移できます。

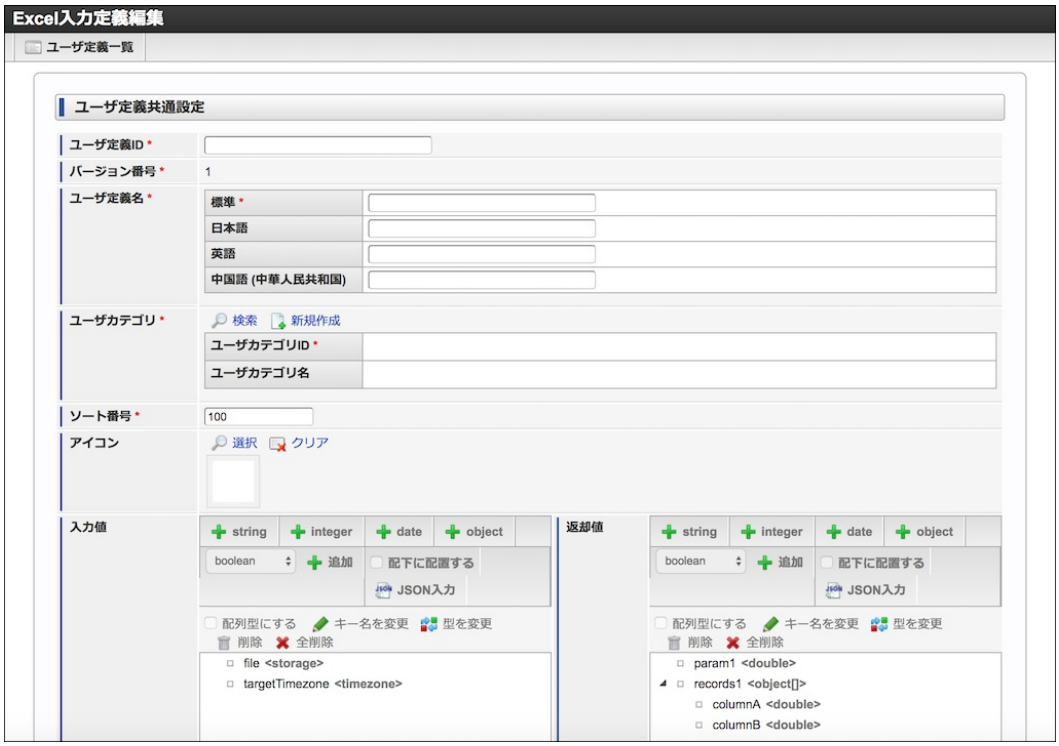
メニューから作成画面へ遷移する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義」配下から、「Excel入力定義新規作成」をクリックします。




図：サイトマップ - ユーザ定義

2. 「Excel入力定義編集」画面が表示されます。




図：「Excel入力定義編集」画面

ユーザ定義一覧画面から作成画面へ遷移する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義一覧」から、ユーザ定義一覧を開きます。
2. 画面上部のヘッダ内、「新規作成」をクリックし、ポップアップしたメニューから「Excel入力定義新規作成」をクリックします。


図：ユーザ定義一覧 - Excel入力定義新規作成

3. 「Excel入力定義編集」画面が表示されます。

## ユーザ定義（Excel入力）の詳細

この章では、Excel入力を利用したユーザ定義の作成方法とその詳細について説明します。



### 本チュートリアルで作成する概要

本チュートリアルでは、シンプルなユーザ情報を列挙したExcelファイルを読み込みます。
チュートリアルを通して、ユーザ定義（Excel入力）の作成方法とその詳細を説明します。

本チュートリアルで定義するExcelファイルの詳細

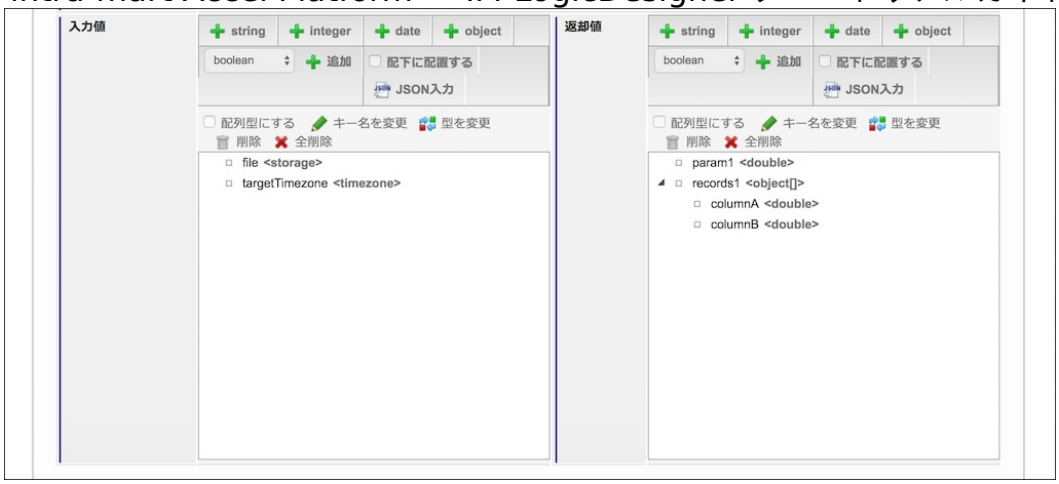
本チュートリアルで使用するExcelファイルのサンプルは以下の通りです。

| | A | B | C | D |
|---|---|---|---|---|
| 1 | Customer Master | | | |
| 2 | user_name | age | birth_date | description |
| 3 | user001 | 20 | 2016-08-01 | User No. 001 |
| 4 | user002 | 21 | 2016-08-02 | User No. 002 |
| 5 | user003 | 22 | 2016-08-03 | User No. 003 |
| 6 | user004 | 23 | 2016-08-04 | User No. 004 |
| 7 | user005 | 24 | 2016-08-05 | User No. 005 |
| 8 | | | | |

図：使用するExcelファイルのサンプル

### 入力値/出力値

はじめに、作成するユーザ定義を呼び出す際の入力値と、呼び出しが完了した際に返却する返却値を定義します。
ユーザ定義（Excel入力）の入力値/返却値の初期値は以下の通りです。




図：入力値、および、返却値の初期値

入力値

ユーザ定義（Excel入力）の入力値はシステムによって固定です。
IM-LogicDesignerではExcelファイルを取り扱うにあたり、以下の内容を入力値として定義しています。

| 入力値 | 説明 |
|---|---|
| file<storage> | データを読み込むExcelファイルのアドレスを示すストレージ情報 |
| targetTimezone<timezone> | セルの値を読み込む際に基準とするタイムゾーン targetTimezone が指定されていない場合、「アカウントコンテキストのタイムゾーン」が指定されます。 |

出力値

「*セル入力定義*」、「*範囲指定入力定義*」で設定した返却パラメータ名とデータ型が定義されます。
本チュートリアルでは以下のように定義します。

| 返却パラメータ | 説明 |
|---|---|
| table_name<string> | 指定したセルの値を格納します。 |
| records<object []> - userName<string> | 指定した範囲のセルの値を格納します。 |
| records<object []> - age<double> | 指定した範囲のセルの値を格納します。 |
| records<object []> - birthDate<date> | 指定した範囲のセルの値を格納します。 |
| records<object []> - description<string> | 指定した範囲のセルの値を格納します。 |

Excel入力共通定義

読み込むシートを指定する方法を選択します。
IM-LogicDesignerではExcelファイルを読み込むにあたり、以下の方法で読み込むシートを指定します。

| 指定方法 | 説明 |
|---|---|
| シート名で指定 | 読み込むシートを、シート名で指定します。 |
| シートの順番で指定 | 読み込むシートをシートの順番で指定します。 最初のシートを指定する場合は、シート番号に 0 を入力してください。 |

セル入力定義

セルの入力定義を以下のように定義します。

| 定義項目 | 説明 |
|---|---|
| シート名 | シート名を指定します。 |
| シート番号 | シートの順番で指定します。 |
| セル | 読み込むセルを指定します。 |
| 返却パラメータ名 | パラメータ名を指定します。 |

| 定義項目 | 説明 |
| --- | --- |
| データ型 | データ型を指定します。 |

> **注意**
>
> セルのデータ型と、ユーザ定義で指定したデータ型について
>
> セルのデータ型と、ユーザ定義で指定したデータ型が異なる場合、値を読み込むことができません。
> 後述の範囲指定入力でも同様です。

範囲指定入力定義

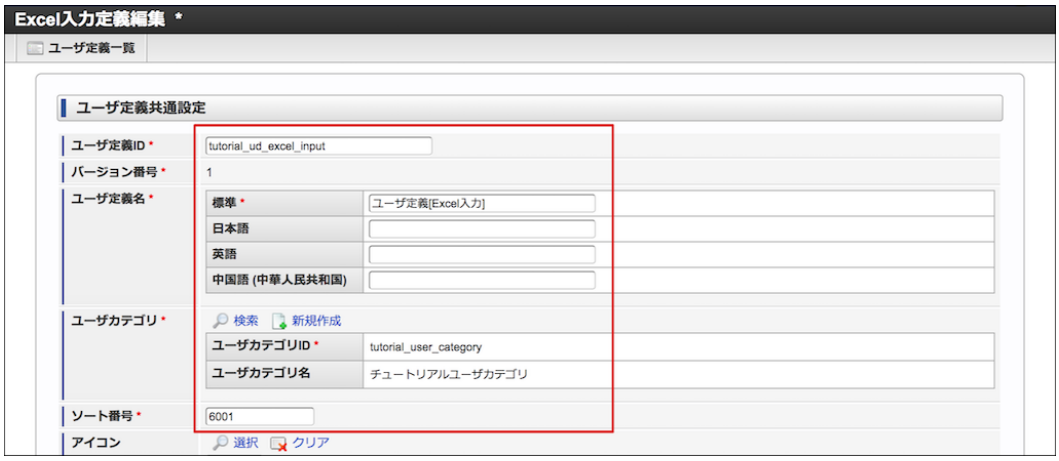
範囲指定入力定義を以下のように定義します。

| 定義項目 | 説明 |
| --- | --- |
| シート名 | シート名を指定します。 |
| シート番号 | シートの順番で指定します。 |
| 対象列 | 読み込む対象列を指定します。 |
| 開始行 | 読み込みを開始する行を指定します。 |
| 終了条件 | 読み込みを終了する条件を指定します。 |
| その他 | 読み込み時のその他動作を指定します。 |
| 返却パラメータ名 | パラメータ名を指定します。 |
| データ型 | データ型を指定します。 |

## ユーザ定義（Excel入力）を作成する

最後に、これまでの内容を踏まえてユーザ定義（Excel）を作成します。

1. 「Excel定義編集」画面を表示します。
2. ユーザ定義の基本情報となる各項目に以下の値を入力します。
   - ユーザ定義ID「tutorial_ud_excel_input」
   - バージョン「1」（固定）
   - ユーザ定義名
     - 標準 - 「ユーザ定義[Excel入力]」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし
   - カテゴリ
     - カテゴリID - 「tutorial_user_category」
   - ソート番号「7001」

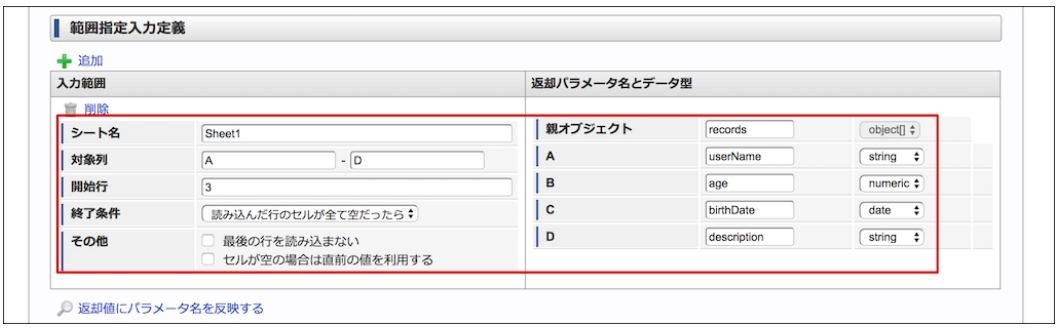



図：基本情報の定義

3. セル入力定義を定義します。

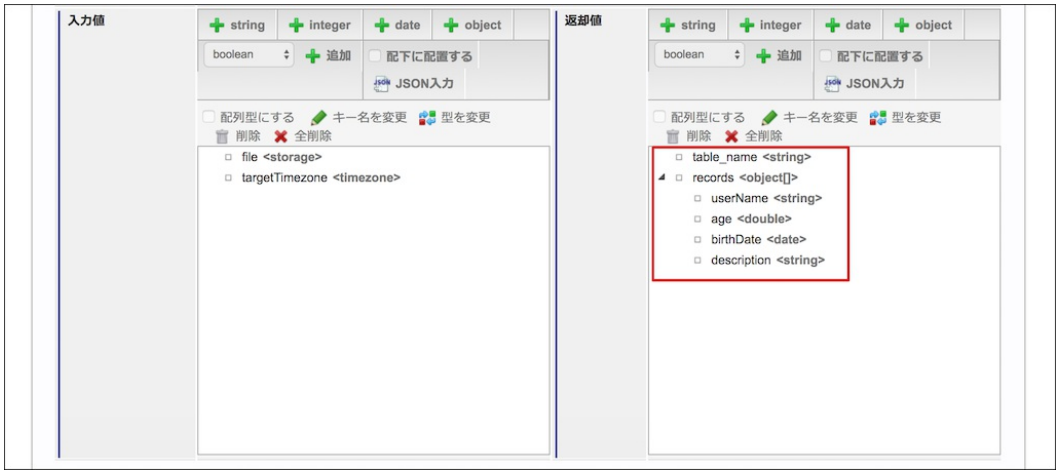

図：セル入力定義の定義

4. 範囲指定入力を使用する場合、以下のように定義します。




図：範囲指定入力の定義

5. セル入力定義、範囲指定入力定義の定義後、「返却値にパラメータ名を反映する」をクリックすると返却値が以下のように定義されます。




図：返却値の定義

6. 「登録」をクリックします。


図：登録

以上で、ユーザ定義（Excel入力）の作成が完了しました。

## ユーザ定義 - Excel出力

### ユーザ定義（Excel出力）

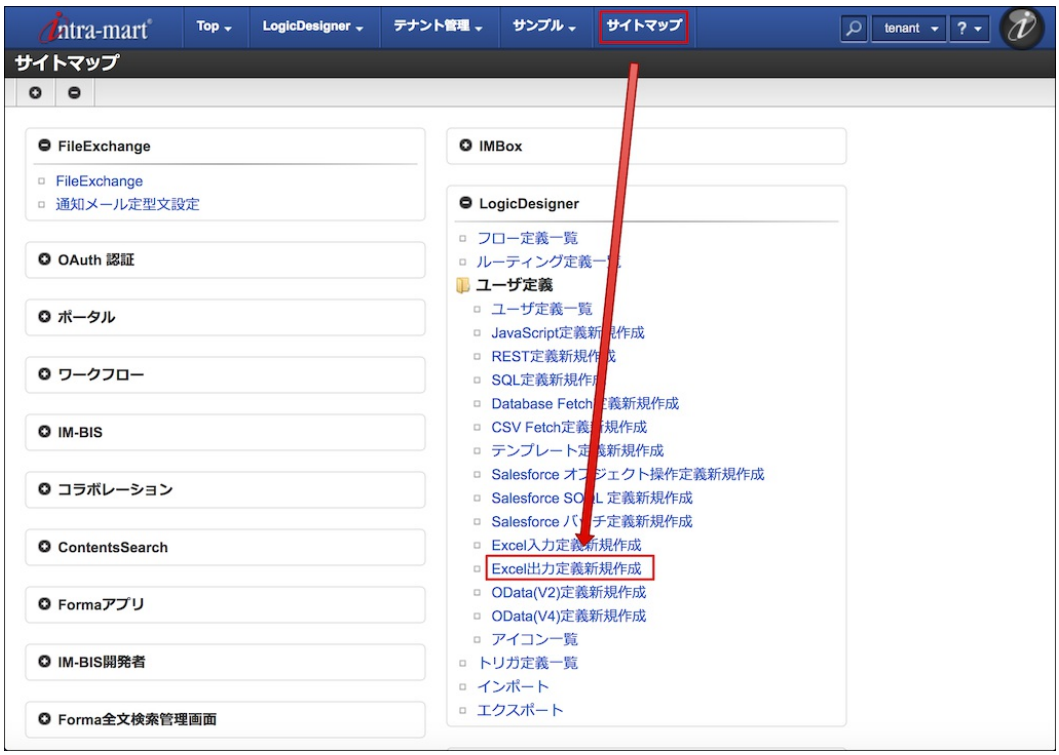
ユーザ定義（Excel出力）とは、入力値として渡された値をExcelファイルへ書き出すことが可能なユーザ定義の一つです。

### ユーザ定義（Excel出力）の作成画面への遷移手順

ユーザ定義（Excel出力）作成画面へは、サイトマップメニューとユーザ定義一覧画面から遷移できます。

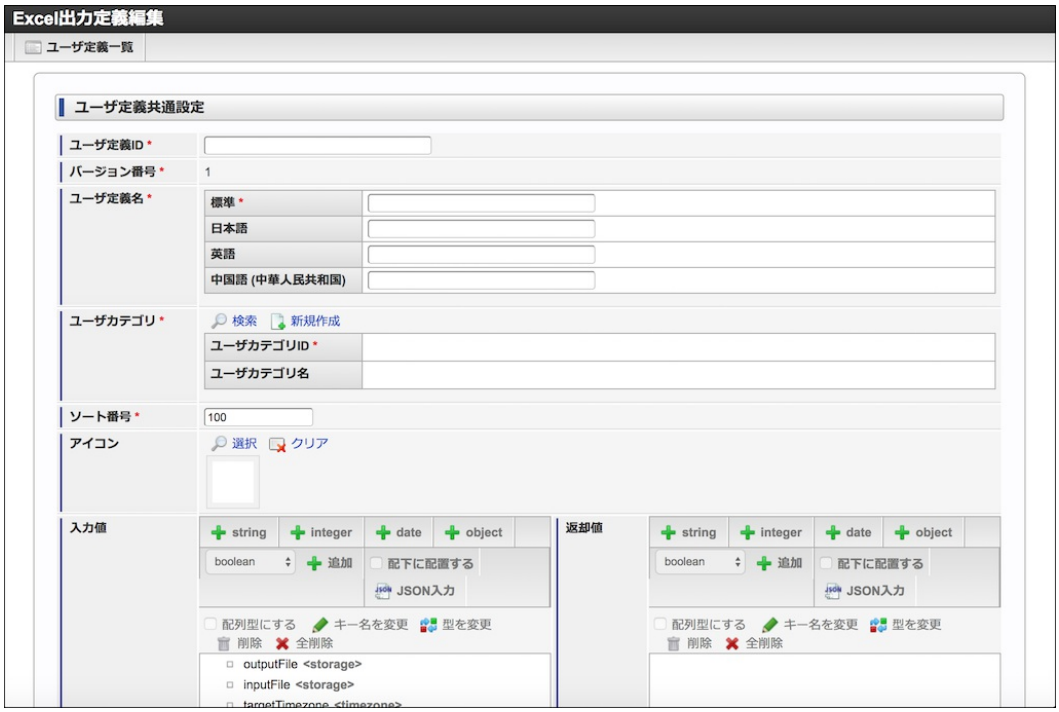
メニューから作成画面へ遷移する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義」配下から、「Excel出力定義新規作成」をクリックします。




図：サイトマップ - ユーザ定義

2. 「Excel出力定義編集」画面が表示されます。




図：「Excel出力定義編集」画面

ユーザ定義一覧画面から作成画面へ遷移する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義一覧」から、ユーザ定義一覧を開きます。
2. 画面上部のヘッダ内、「新規作成」をクリックし、ポップアップしたメニューから「Excel出力定義新規作成」をクリックします。


図：ユーザ定義一覧 - Excel出力定義新規作成

3. 「Excel出力定義編集」画面が表示されます。

## ユーザ定義（Excel出力）の詳細

この章では、Excel出力を利用したユーザ定義の作成方法とその詳細について説明します。



### 本チュートリアルで作成する概要

本チュートリアルでは、入力値として与えられたデータをExcelファイルに出力するタスクの実装を通して、Excel出力を利用したユーザ定義の作成方法とその詳細を説明します。

### 入力値/出力値

はじめに、作成するユーザ定義を呼び出す際の入力値を定義します。
ユーザ定義（Excel出力）の入力値/返却値の初期値は以下の通りです。


図：入力値、および、返却値の初期値

入力値

ユーザ定義（Excel出力）の入力値はシステムによって固定なものと、セル出力定義、範囲指定出力定義で入力した情報が定義されます。
IM-LogicDesignerではExcelファイルを取り扱うにあたり、以下の内容を固定の入力値として定義しています。

| 入力値 | 説明 |
|---|---|
| outputFile<storage> | データを出力するExcelファイルのアドレスを示すストレージ情報 |
| inputFile<storage> | データを出力するExcelファイルのテンプレートとなるファイルのアドレスを示すストレージ情報 |
| targetTimezone<timezone> | セルの値を読み込む際に基準とするタイムゾーン targetTimezone が指定されていない場合、「アカウントコンテキストのタイムゾーン」が指定されます。 |

出力値

ユーザ定義（Excel出力）に返却値はありません。

Excel出力共通定義

書き込むシートを指定する方法を選択します。
IM-LogicDesignerではExcelファイルへ書き込みを行うにあたり、以下の方法で書き込むシートを指定します。

| 定義項目 | 説明 |
|---|---|
| シート名で指定 | 出力するシートを、シート名で指定します。 |
| シートの順番で指定 | 出力するシートをシートの順番で指定します。 最初のシートを指定する場合は、シート番号に 0 を入力してください。 |
| 関数の再計算 | チェックボックスをオンにすると、出力されたExcelファイルを開いた際に関数を再計算させることができます。 |

セル出力定義

セルの出力定義を以下のように定義します。

| 定義項目 | 説明 |
|---|---|
| シート名 | シート名を指定します。 |
| シート番号 | シートの順番で指定します。 |
| セル | 出力するセルを指定します。 |
| 入力パラメータ名 | パラメータ名を指定します。 |
| データ型 | データ型を指定します。 |

範囲指定出力定義

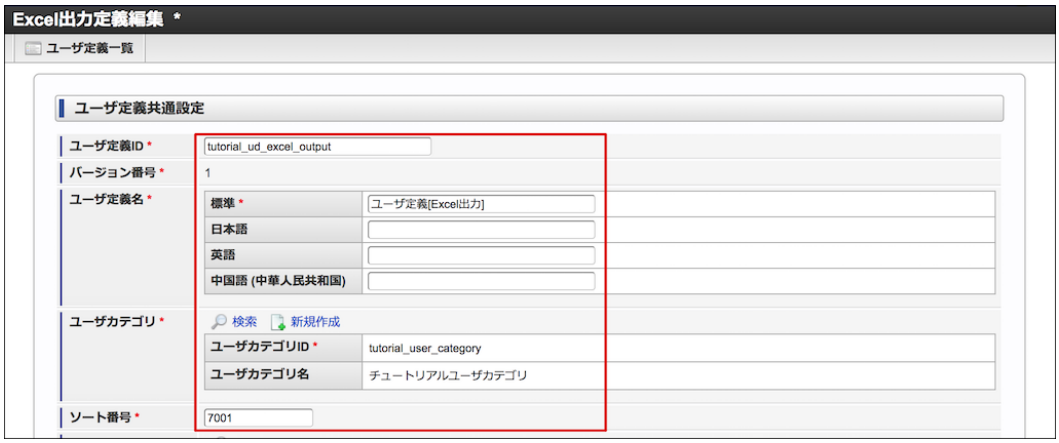
範囲指定出力定義を以下のように定義します。

| 定義項目 | 説明 |
|---|---|
| シート名 | シート名を指定します。 |
| シート番号 | シートの順番で指定します。 |
| 対象列 | 出力する対象列を指定します。 |
| 開始行 | 出力を開始する行を指定します。 |
| 終了条件 | 出力を終了する条件を指定します。 |
| 入力パラメータ名 | パラメータ名を指定します。 |
| データ型 | データ型を指定します。 |

### ユーザ定義（Excel）を作成する

最後に、これまでの内容を踏まえてユーザ定義（Excel）を作成します。

1. 「Excel定義編集」画面を表示します。
2. ユーザ定義の基本情報となる各項目に以下の値を入力します。
   - ユーザ定義ID「tutorial_ud_excel_output」
   - バージョン「1」（固定）
   - ユーザ定義名
     - 標準 - 「ユーザ定義[Excel出力]」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし
   - カテゴリ
     - カテゴリID - 「tutorial_user_category」
   - ソート番号「7101」

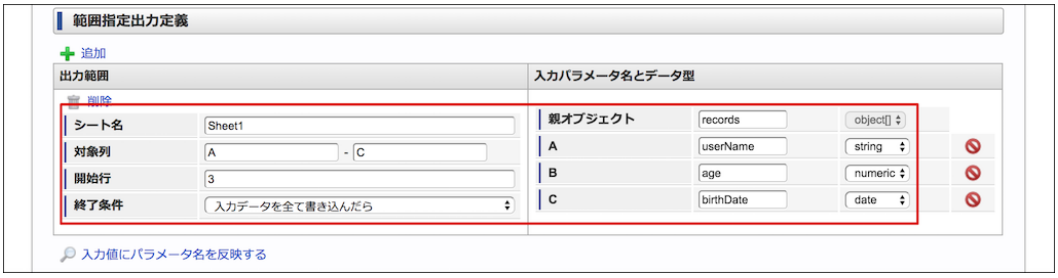



図：基本情報の定義

3. セル出力定義を定義します。


図：セル出力定義の定義

4. 範囲指定出力を使用する場合、以下のように定義します。




図：範囲指定出力の定義

5. セル出力定義、範囲指定出力定義の定義後、「入力値にパラメータ名を反映する」をクリックすると入力値が以下のように定義されます。


図：入力値の定義

6. 「登録」をクリックします。


図：登録

以上で、ユーザ定義（Excel出力）の作成が完了しました。

## ユーザ定義 - XML解析

### ユーザ定義（XML解析）

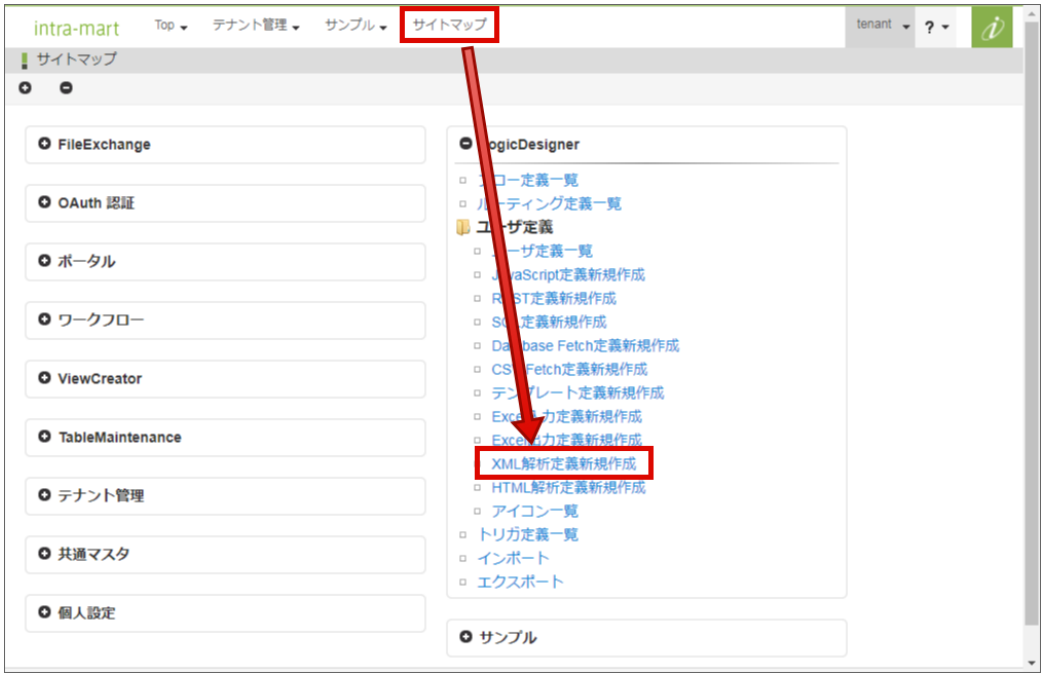
ユーザ定義（XML解析）とは、XMLを読み込み・任意の要素の値を取得することが可能なユーザ定義の一つです。
タスクの入力値に指定したXMLから、出力値に設定した要素の値を取得することができます。

### ユーザ定義（XML解析）の作成画面への遷移手順

ユーザ定義（XML解析）作成画面へは、サイトマップメニューとユーザ定義一覧画面から遷移できます。

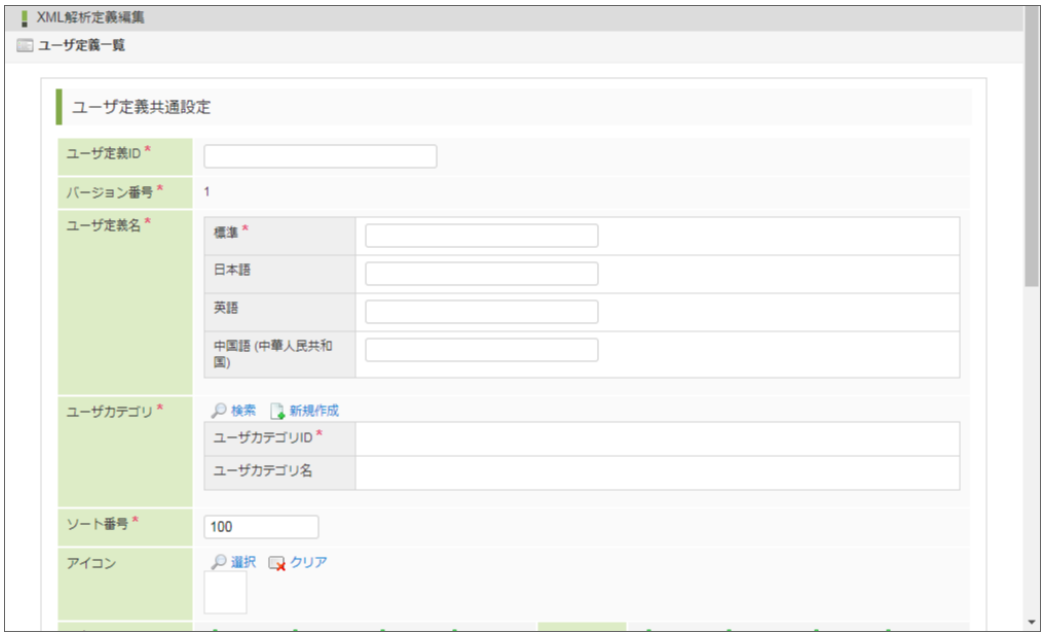
メニューから作成画面へ遷移する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義」配下から、「XML解析定義新規作成」をクリックします。




図：サイトマップ - ユーザ定義

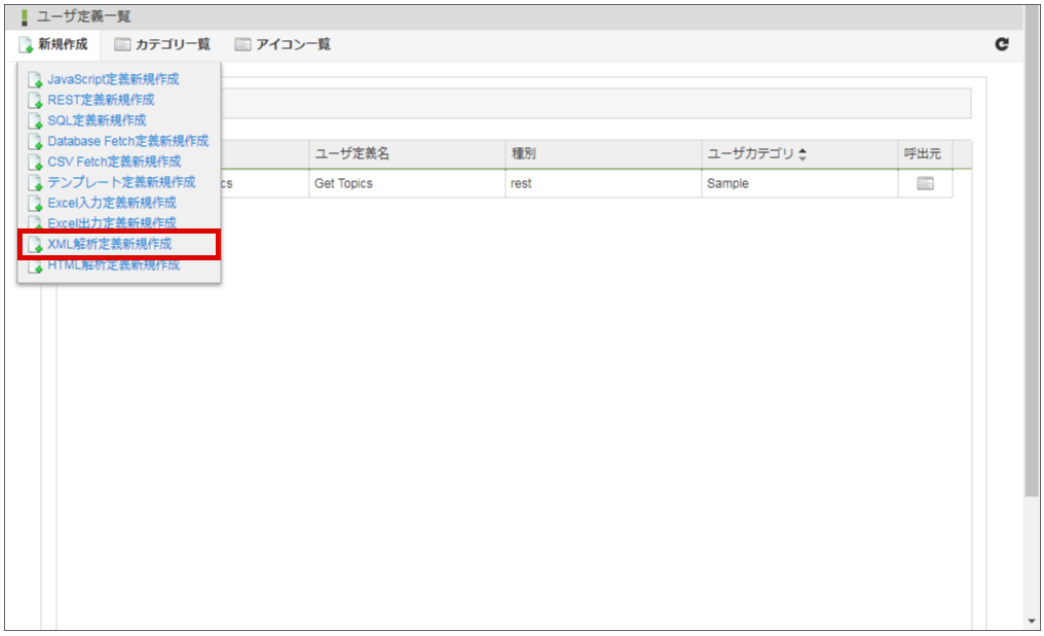
2. 「XML解析定義編集」画面が表示されます。




図：「XML解析定義編集」画面

ユーザ定義一覧画面から作成画面へ遷移する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義一覧」から、ユーザ定義一覧を開きます。
2. 画面上部のヘッダ内、「新規作成」をクリックし、ポップアップしたメニューから「XML解析定義新規作成」をクリックします。




図：ユーザ定義一覧 - XML解析定義新規作成

3. 「XML解析定義編集」画面が表示されます。

## ユーザ定義（XML解析）の詳細

この章では、XML解析を利用したユーザ定義の作成方法とその詳細について説明します。



### 本チュートリアルで作成する概要

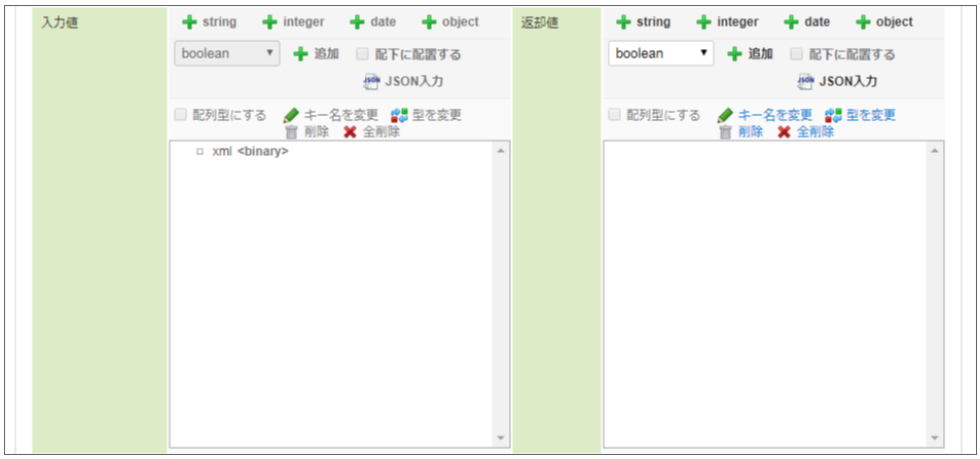
本チュートリアルでは、アカウント情報のXMLを読み込みアカウントデータを取得します。
チュートリアルを通して、ユーザ定義（XML解析）の作成方法とその詳細を説明します。

本チュートリアルで使用するXMLの詳細

本チュートリアルではアカウントエクスポートでエクスポートされたアカウント情報XMLを使用します。
XMLの内容は以下の通りです。

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="http://intra-mart.co.jp/system/admin/account/account-data">
  <account-data cd="aoyagi">
    <password>aoyagi</password>
    <first-day-of-week>-1</first-day-of-week>
    <login-failure-count>0</login-failure-count>
    <valid-start-date>1900-01-01</valid-start-date>
    <valid-end-date>3000-01-01</valid-end-date>
    <account-license>true</account-license>
    <account-roles>
      <account-role id="im_workflow_user">
        <role-valid-start-date>1900-01-01</role-valid-start-date>
        <role-valid-end-date>3000-01-01</role-valid-end-date>
      </account-role>
    </account-roles>
  </account-data>
  <account-data cd="hagimoto">
    <password>hagimoto</password>
    <first-day-of-week>-1</first-day-of-week>
    <login-failure-count>0</login-failure-count>
    <valid-start-date>1900-01-01</valid-start-date>
    <valid-end-date>3000-01-01</valid-end-date>
    <account-license>true</account-license>
    <account-roles>
      <account-role id="im_workflow_user">
        <role-valid-start-date>1900-01-01</role-valid-start-date>
        <role-valid-end-date>3000-01-01</role-valid-end-date>
      </account-role>
    </account-roles>
  </account-data>
  <account-data cd="harada">
    <password>harada</password>
    <first-day-of-week>-1</first-day-of-week>
    <login-failure-count>0</login-failure-count>
    <valid-start-date>1900-01-01</valid-start-date>
    <valid-end-date>3000-01-01</valid-end-date>
    <account-license>true</account-license>
    <account-roles>
      <account-role id="im_workflow_user">
        <role-valid-start-date>1900-01-01</role-valid-start-date>
        <role-valid-end-date>3000-01-01</role-valid-end-date>
      </account-role>
    </account-roles>
  </account-data>
  <account-data cd="hayashi">
    <password>hayashi</password>
    <first-day-of-week>-1</first-day-of-week>
    <login-failure-count>0</login-failure-count>
    <valid-start-date>1900-01-01</valid-start-date>
    <valid-end-date>3000-01-01</valid-end-date>
    <account-license>true</account-license>
    <account-roles>
      <account-role id="im_workflow_user">
        <role-valid-start-date>1900-01-01</role-valid-start-date>
        <role-valid-end-date>3000-01-01</role-valid-end-date>
      </account-role>
    </account-roles>
  </account-data>
  <account-data cd="ikuta">
    <password>ikuta</password>
    <first-day-of-week>-1</first-day-of-week>
    <login-failure-count>0</login-failure-count>
    <valid-start-date>1900-01-01</valid-start-date>
    <valid-end-date>3000-01-01</valid-end-date>
    <account-license>true</account-license>
    <account-roles>
      <account-role id="im_workflow_user">
        <role-valid-start-date>1900-01-01</role-valid-start-date>
        <role-valid-end-date>3000-01-01</role-valid-end-date>
      </account-role>
    </account-roles>
  </account-data>
</root>
```

ユーザ定義を呼び出す際の入力値と、呼び出しが完了した際に返却する返却値を定義します。
ユーザ定義（XML解析）の入力値/返却値の初期値は以下の通りです。




図：入力値、および、返却値の初期値

入力値

ユーザ定義（XML解析）の入力値はシステムによって固定です。
IM-LogicDesignerではXMLを取り扱うにあたり、以下の内容を入力値として定義しています。

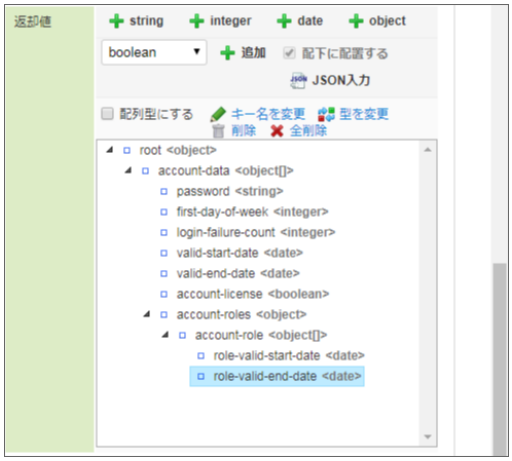
| 入力値 | 説明 |
| --- | --- |
| xml<binary> | データを読み込むXML |

出力値

ユーザ定義（XML解析）の出力値はXMLから取得する要素を定義します。
出力値の定義方法には、「*ルートから要素を定義する方法*」 と「*XPath で要素を定義する方法*」があります。

ルートから要素を定義する方法

返却値にXML内のタグ名にあわせて要素を定義することで、定義された名前の要素を取得することができます。
アカウント情報XMLの場合、以下のように定義します。




図：ルートから要素を定義した例

XMLから属性の値を取得したい場合は、要素を Object として定義し、配下の要素に「@属性名」の要素を追加します。
またテキスト要素を取得したい場合は「#text」の要素を追加します。


図：属性、テキスト取得例

XPathで要素を定義する方法

XPathを使用して取得したい要素を指定して定義を行う方法です。XMLから特定の要素のみ取得したい場合に利用します。
たとえば、アカウント情報からユーザコードのみを取得したい場合は属性「cd」へのXPathを定義します。

- XPath

  `/root/account-data/@cd`

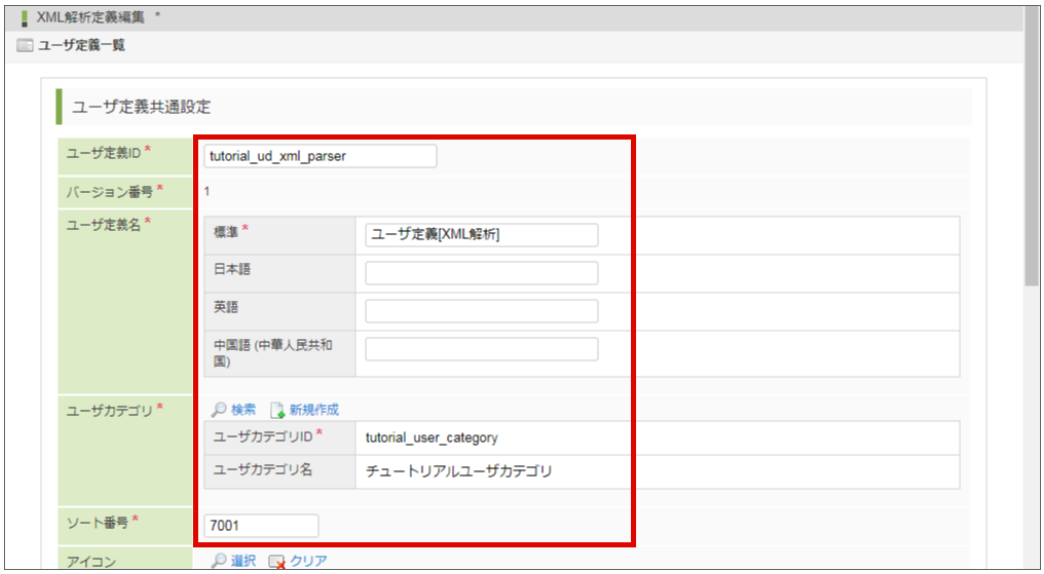
XML解析定義のパラメータ名には、XPathで指定した要素を関連付ける返却値の要素名を指定します。


図：ユーザコード取得例

### ユーザ定義（XML解析）を作成する

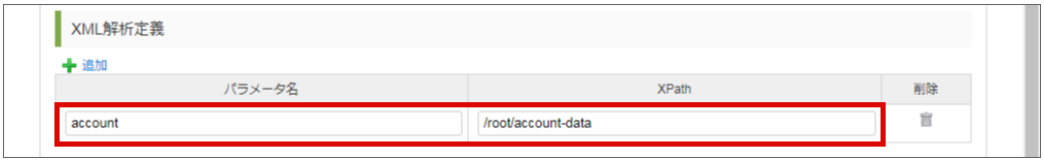
最後に、これまでの内容を踏まえてユーザ定義（XML解析）を作成します。

1. 「XML解析定義編集」画面を表示します。
2. ユーザ定義の基本情報となる各項目に以下の値を入力します。
   - ユーザ定義ID「`tutorial_ud_xml_parser`」
   - バージョン「`1`」（固定）
   - ユーザ定義名
     - 標準 - 「`ユーザ定義[XML解析]`」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし
   - カテゴリ
     - カテゴリID - 「`tutorial_user_category`」
   - ソート番号「`8001`」



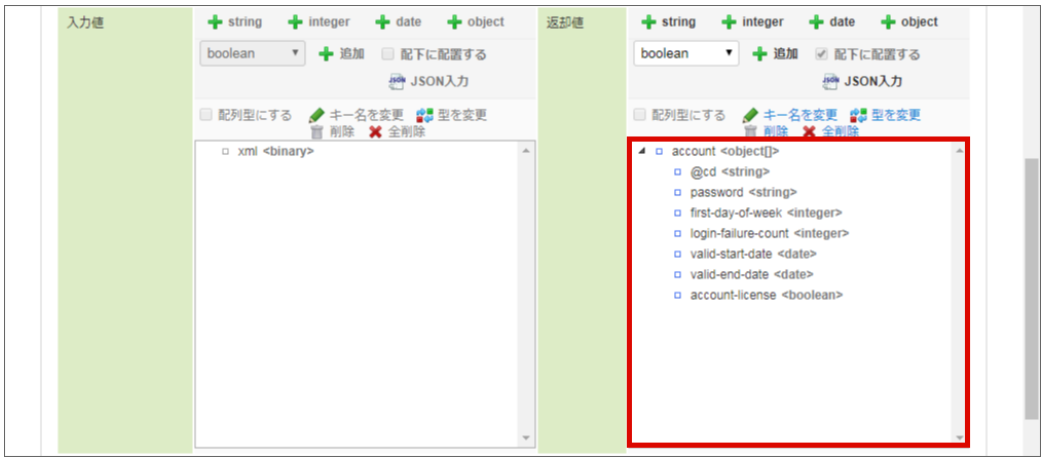

図：基本情報の定義

3. XPathを定義します。




図：XPathの定義

4. パラメータ名に入力した値をルートにして取得する要素を返却値に定義します。




図：返却値の定義

## ユーザ定義 - HTML解析

### ユーザ定義（HTML解析）

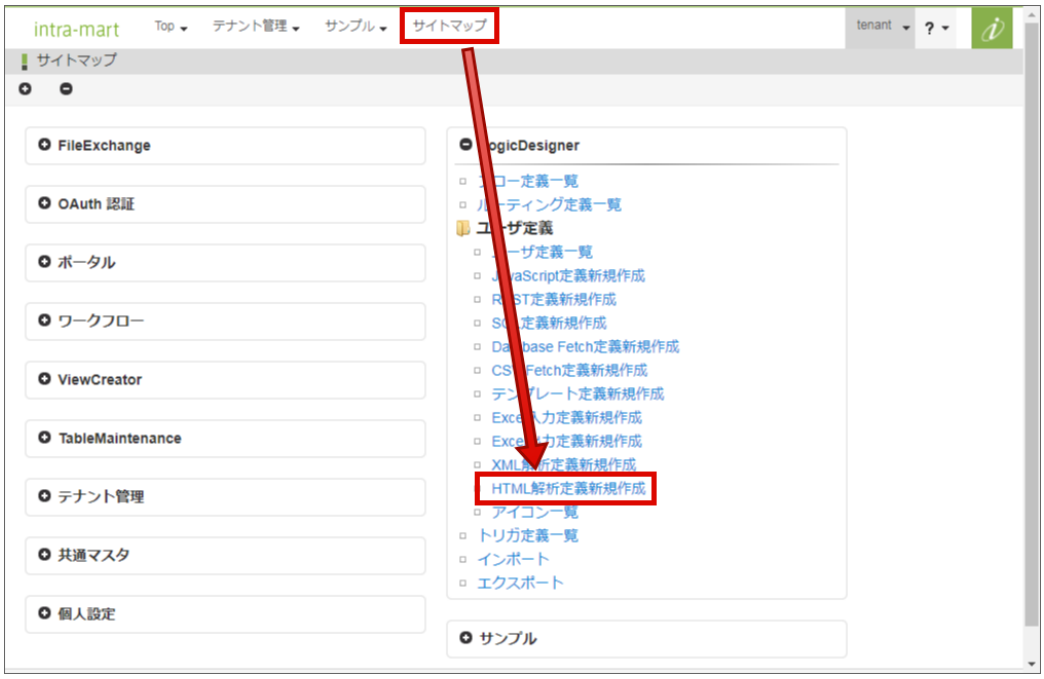
ユーザ定義（HTML解析）とは、HTMLを読み込み・任意の要素の値を取得することが可能なユーザ定義の一つです。
タスクの入力値に指定したHTMLから、出力値に設定した要素の値を取得することができます。

#### ユーザ定義（HTML解析）の作成画面への遷移手順

ユーザ定義（HTML解析）作成画面へは、サイトマップメニューとユーザ定義一覧画面から遷移できます。

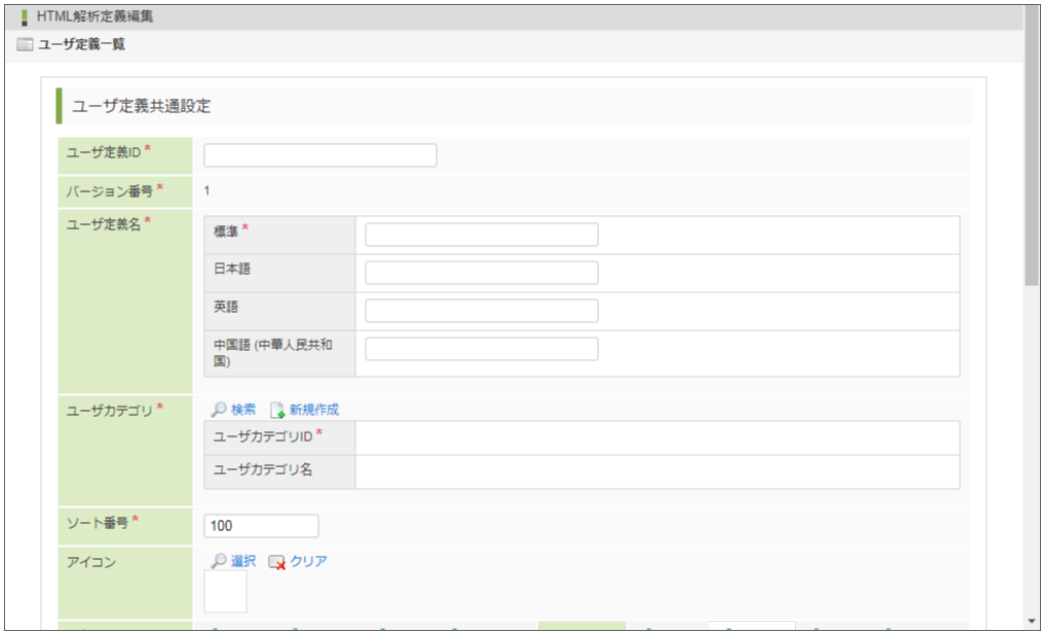
メニューから作成画面へ遷移する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義」配下から、「HTML解析定義新規作成」をクリックします。




図：サイトマップ - ユーザ定義

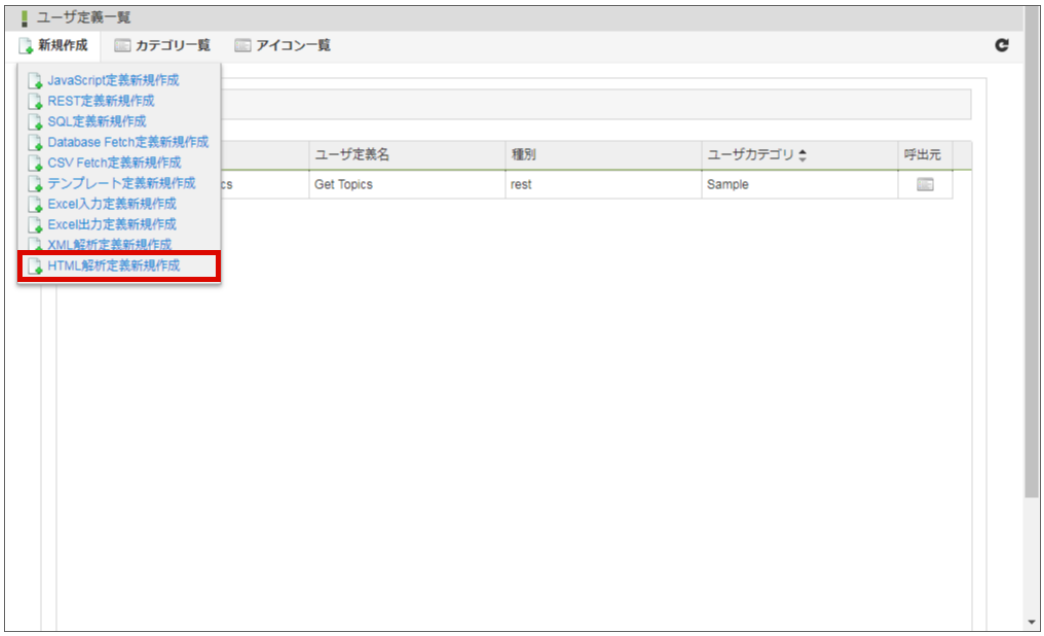
2. 「HTML解析定義編集」画面が表示されます。




図：「XML解析定義編集」画面

ユーザ定義一覧画面から作成画面へ遷移する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義一覧」から、ユーザ定義一覧を開きます。
2. 画面上部のヘッダ内、「新規作成」をクリックし、ポップアップしたメニューから「HTML解析定義新規作成」をクリックします。




図：ユーザ定義一覧 - HTML解析定義新規作成

3. 「HTML解析定義編集」画面が表示されます。

## ユーザ定義（HTML解析）の詳細

この章では、HTML解析を利用したユーザ定義の作成方法とその詳細について説明します。



### 本チュートリアルで作成する概要

本チュートリアルでは、サンプルのHTMLを読み込み要素を取得します。
チュートリアルを通して、ユーザ定義（HTML解析）の作成方法とその詳細を説明します。

本チュートリアルで使用するHTMLの詳細

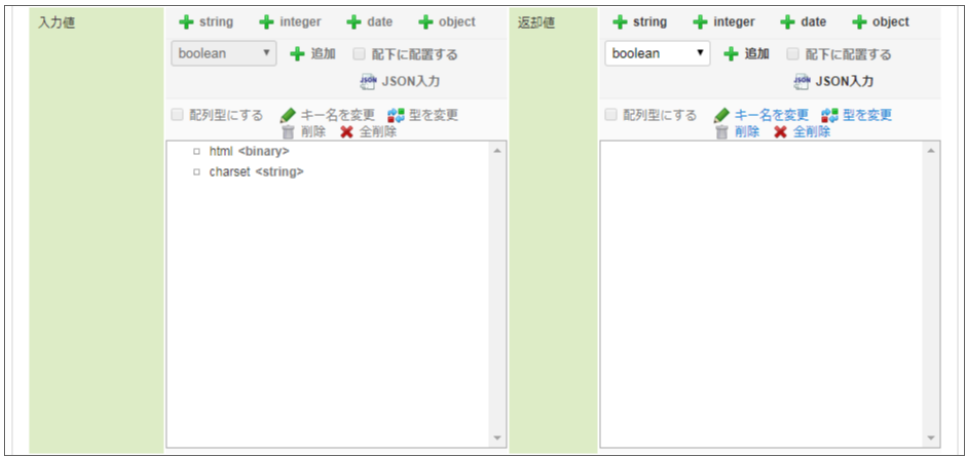
本チュートリアルで使用するHTMLのサンプルは以下の通りです。

```
<html>
 <body>
  <h1>Sample HTML</h1>
  <div class="contents">
   <table>
    <tr>
     <th class="header">param1</th>
     <th class="header">param2</th>
    </tr>
    <tr>
     <td class="data">record1-a</td>
     <td class="data">recoed1-b</td>
    </tr>
    <tr>
     <td class="data">record2-a</td>
     <td class="data">recoed2-b</td>
    </tr>
   </table>
  </div>
 </body>
</html>
```

### 入力値/出力値

ユーザ定義を呼び出す際の入力値と、呼び出しが完了した際に返却する返却値を定義します。

ユーザ定義（HTML解析）の入力値/返却値の初期値は以下の通りです。




図：入力値、および、返却値の初期値

入力値

ユーザ定義（HTML解析）の入力値はシステムによって固定です。
IM-LogicDesignerではHTMLを取り扱うにあたり、以下の内容を入力値として定義しています。

| 入力値 | 説明 |
| --- | --- |
| html<binary> | データを読み込むHTML |
| charset<string> | HTMLの文字コード |

出力値

ユーザ定義（HTML解析）の出力値はHTMLから取得する要素を定義します。
出力値の定義方法には、「*ルートから要素を定義する方法*」 と「*CSSセレクタで要素を定義する方法*」があります。

ルートから要素を定義する方法

返却値にHTML内のタグ名にあわせて要素を定義することで、定義された名前の要素を取得することができます。
サンプルのHTMLの場合、以下のように定義します。


図：ルートから要素を定義した例

HTMLから属性の値を取得したい場合は、要素を Object として定義し、配下の要素に「@属性名」の要素を追加します。
またテキスト要素を取得したい場合は「#text」の要素を追加します。


図：属性、テキスト取得例

CSSセレクタで要素を定義する方法

CSSセレクタを使用して取得したい要素を指定して定義を行う方法です。HTMLから特定の要素のみ取得したい場合に利用します。
たとえば、サンプルのHTMLから `param1` の列の値を取得したい場合は以下のように定義します。

- CSSセレクタ

  `div.contents table td:nth-of-type(1)`

HTML解析定義のパラメータ名には、CSSセレクタで指定した要素を関連付ける返却値の要素名を指定します。

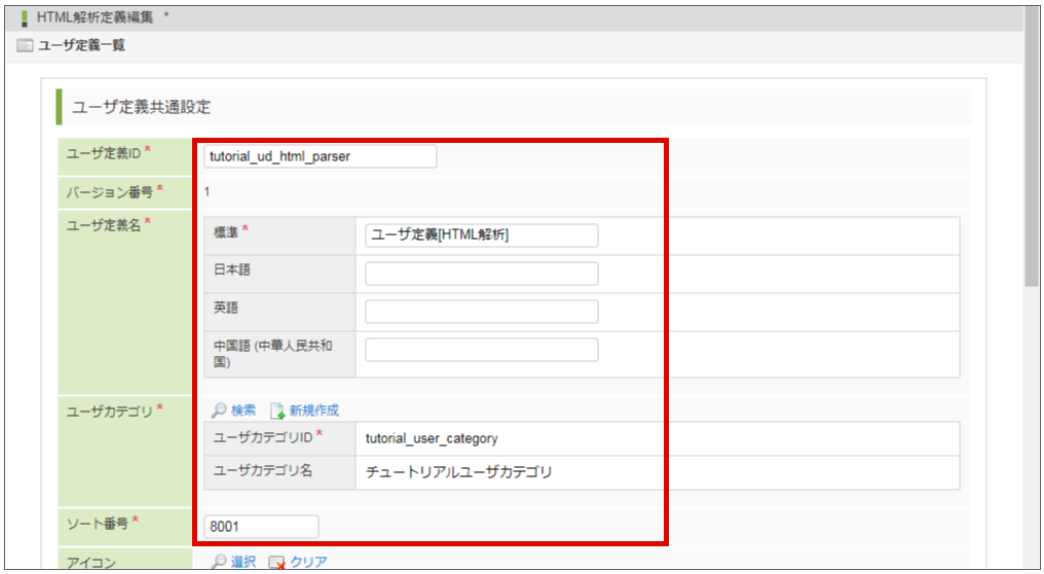

図：CSSセレクタ 定義例

使用できるCSSセレクタについては「API ドキュメント Selector」を参照してください。

### ユーザ定義（HTML解析）を作成する

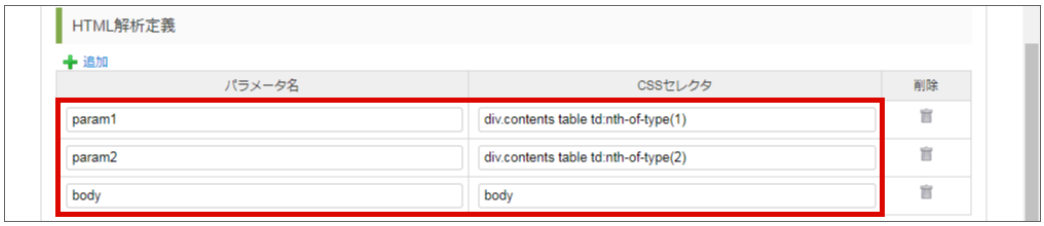
最後に、これまでの内容を踏まえてユーザ定義（HTML解析）を作成します。

1. 「HTML解析定義編集」画面を表示します。
2. ユーザ定義の基本情報となる各項目に以下の値を入力します。
   - ユーザ定義ID「tutorial_ud_html_parser」
   - バージョン「1」（固定）
   - ユーザ定義名
     - 標準 - 「ユーザ定義[HTML解析]」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし
   - カテゴリ
     - カテゴリID - 「tutorial_user_category」
   - ソート番号「9001」



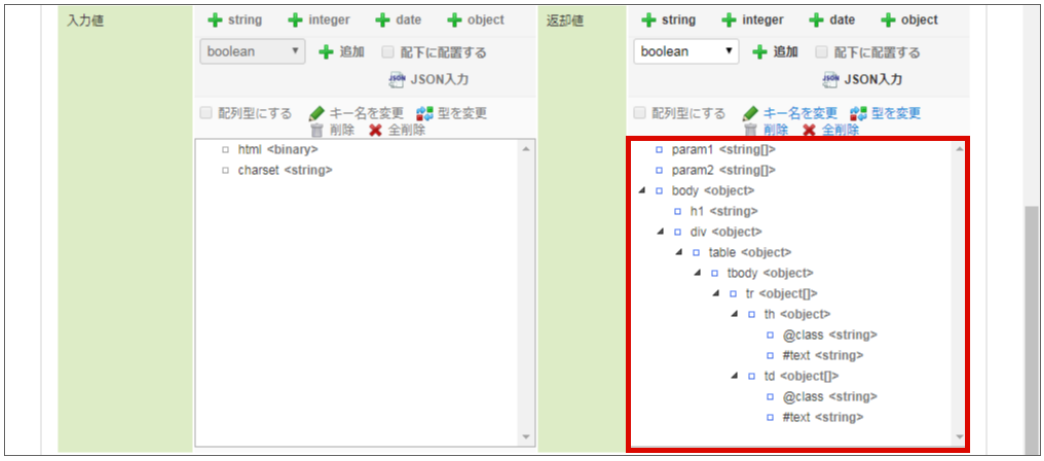

図：基本情報の定義

3. 取得する要素を表すCSSセレクタを定義します。




図：CSSセレクタの定義

4. パラメータ名に入力した値をルートにして取得する要素を返却値に定義します。




図：返却値の定義

## ユーザ定義 - BIS申請/承認

### ユーザ定義（BIS申請/承認）

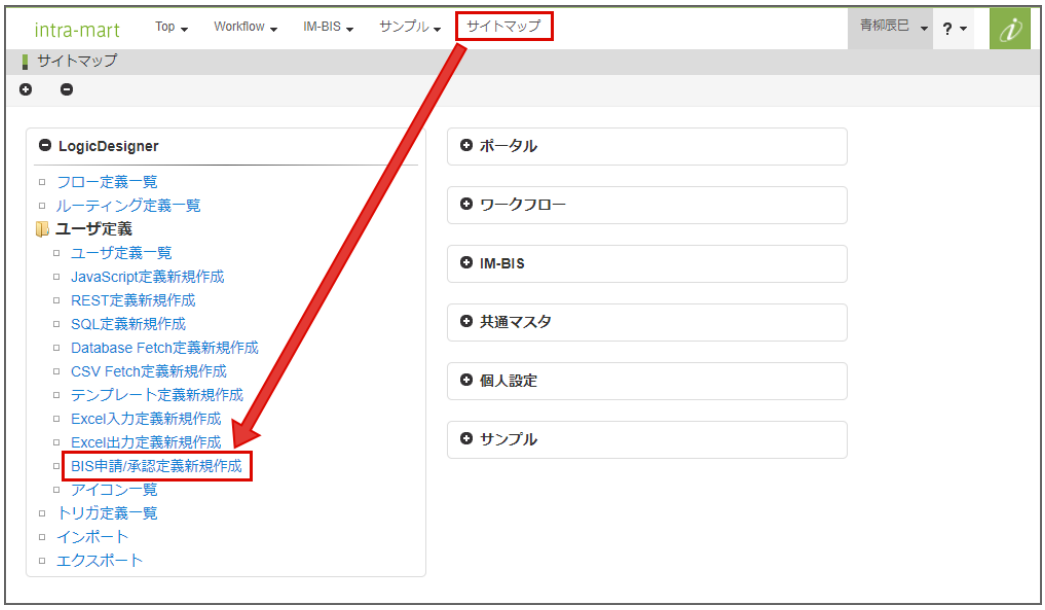
ユーザ定義（BIS申請/承認）とは、ワークフローを処理することが可能なユーザ定義の一つです。
タスクの入力値と処理対象のフロー・案件を紐付け、出力値として申請した案件の情報を扱うことができます。（※）
※ 出力値は処理種別が「申請」「起票」の場合のみ利用できます。

#### ユーザ定義（BIS申請/承認）の作成画面への遷移手順

ユーザ定義（BIS申請/承認）作成画面へは、サイトマップメニューとユーザ定義一覧画面から遷移できます。

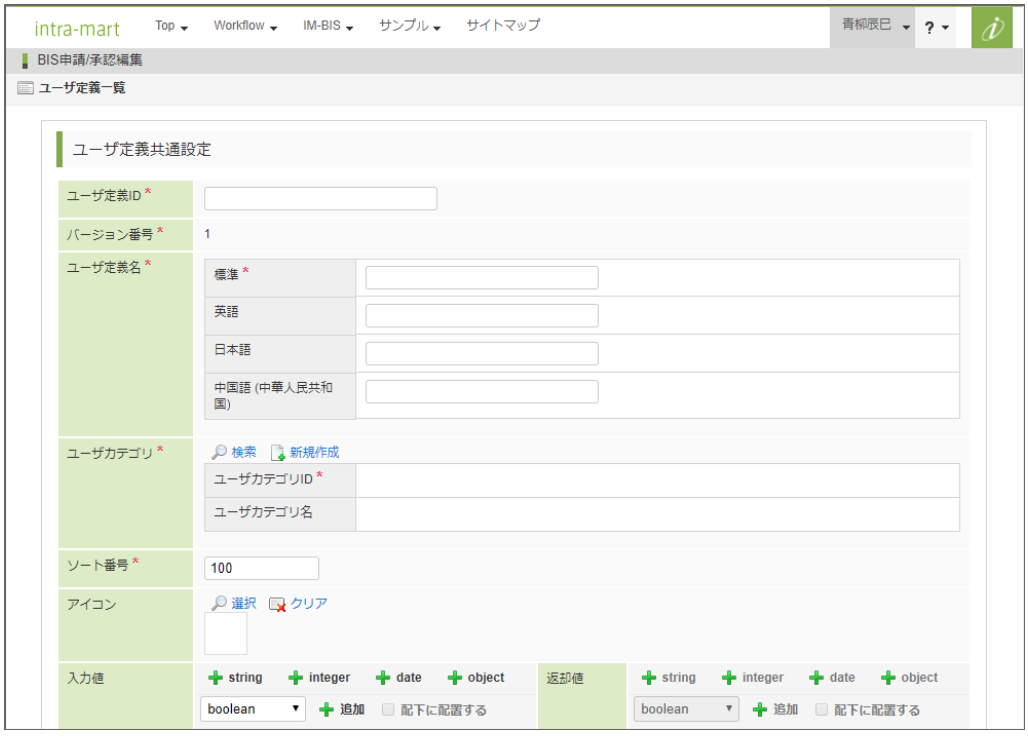
メニューから作成画面へ遷移する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義」配下から、「BIS申請/承認定義新規作成」をクリックします。




図：サイトマップ - ユーザ定義

2. 「BIS申請/承認編集」画面が表示されます。




図：「BIS申請/承認編集」画面

ユーザ定義一覧画面から作成画面へ遷移する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義一覧」から、ユーザ定義一覧を開きます。
2. 画面上部のヘッダ内、「新規作成」をクリックし、ポップアップしたメニューから「BIS申請/承認定義新規作成」をクリックします。


図：ユーザ定義一覧 - BIS申請/承認定義新規作成

3. 「BIS申請/承認編集」画面が表示されます。

## 申請を行うユーザ定義（BIS申請/承認）の作成

この章では、申請を実行するユーザ定義の作成方法とその詳細について説明します。

> **注意**
> ユーザ定義（BIS申請/承認）のチュートリアルは、 IM-LogicDesigner 2017 Winter（8.0.18）以降のバージョンで実行できます。

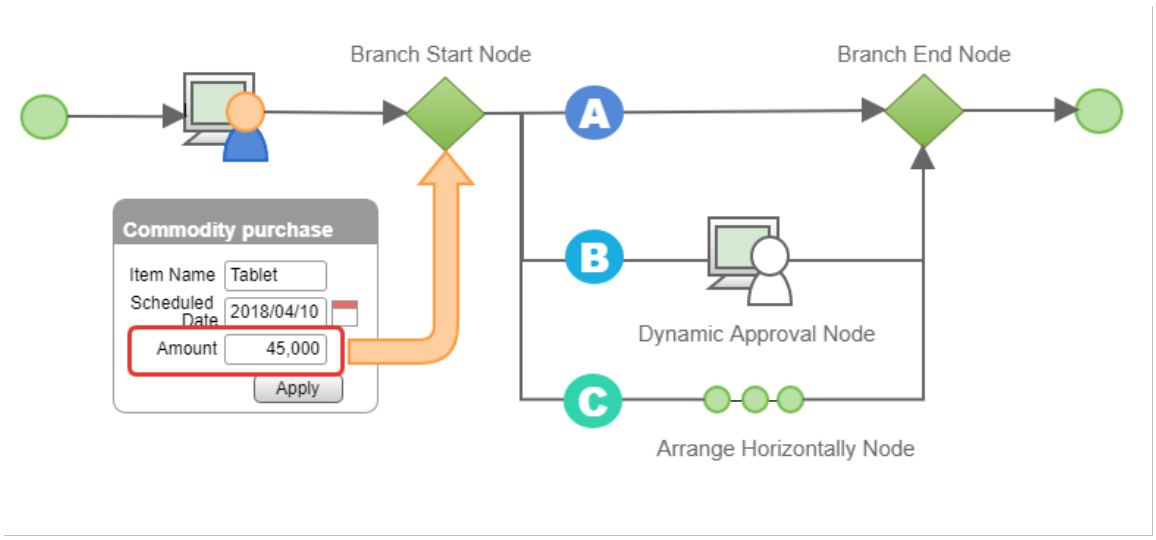


### 本チュートリアルで作成するユーザ定義の概要

本チュートリアルでは、

**「特定のワークフローに対する一括申請を実行する」**

チュートリアルでは、申請画面で入力した金額に応じて、分岐先・処理対象者が変わるワークフローを一括実行します。




図：申請画面と承認ルート

分岐先は以下のとおりです。

- **A**
  入力した金額が10,000円未満の場合のルートです。
  分岐終了ノードに接続しています。

- **B**
  入力した金額が10,000円以上、かつ50,000円以下の場合のルートです。
  動的承認ノードに接続しています。
- **C**
  入力した金額が50,000円を超える場合のルートです。
  横配置ノードに接続しています。

なお、今回のチュートリアルでは処理内容の簡略化のため、一部の設定内容については説明を省略しています。
省略した内容については、 *応用編* からダウンロードしたファイルをインポートして確認してください。

以降のチュートリアルを行う前に、以下のファイルをインポートしてください。

- ユーザ定義（JavaScript）

  分岐開始ノードに対して、分岐先と動的承認ノード・横配置ノードの処理対象者を設定する処理を実装しています。
  インポートの手順は *インポート方法* を参照してください。

  - **ユーザ定義[BIS申請/承認-申請]ルール アーカイブ（ZIP形式）**
- IM-BIS ワークフローの定義情報

  IM-LogicDesigner の実行対象のワークフローに関する定義情報です。
  インポートの手順は「 IM-BIS システム管理者操作ガイド 」-「 一括インポート・エクスポートを行う 」を参照してください。

  - **ユーザ定義[BIS申請/承認-申請]ワークフロー アーカイブ（ZIP形式）**

> **注意**
>
> 本チュートリアルに付属するサンプルでは、以下の事項については説明を割愛しています。
> 実際の開発においては、必要に応じて適切な実装を行ってください。
>
> - 例外処理
> - バリデーションチェック
> - 「ボタン（イベント）」の二度押し防止
> - 処理結果の通知
> - 画面遷移・タブ遷移

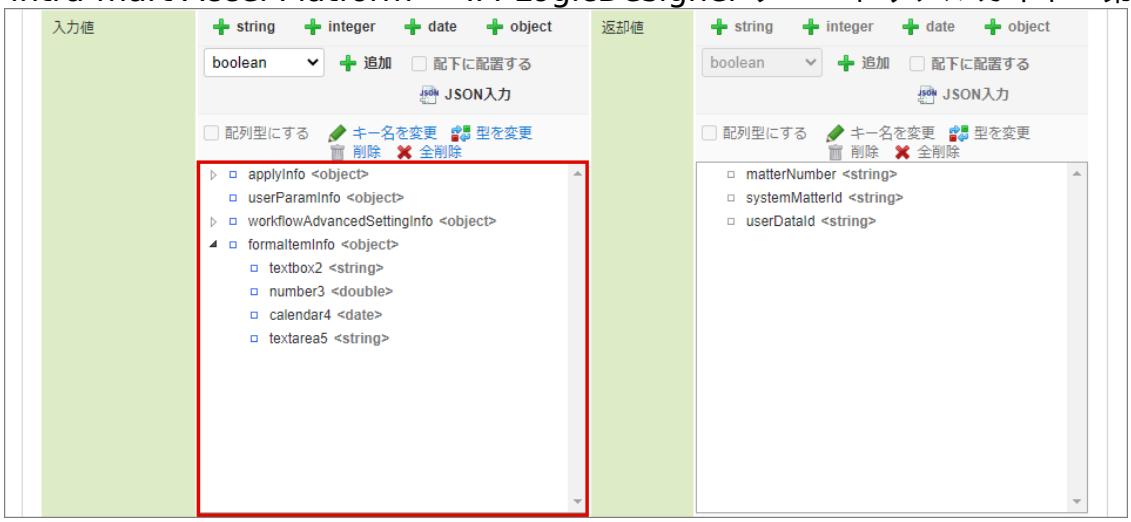
### 処理種別とBIS定義

はじめに、今回作成するユーザ定義（BIS申請/承認）が対象とするアプリケーション種別、処理種別、対象のフローを設定します。
本チュートリアルでは以下の値を設定します。

- アプリケーション種別 - 「BIS」
- 処理種別 - 「申請」
- 定義情報 - 「BISワークフロー（LD：申請）」

### 入力値/出力値

次に、作成するユーザ定義を呼び出す際の入力値と、呼び出しが完了した際に返却する返却値を定義します。
ユーザ定義（BIS申請/承認）の作成において入力値/返却値は、アプリケーション種別・処理種別・BIS/Forma定義を選択したタイミングで自動的に初期値が設定されます。
アプリケーション種別「BIS」、処理種別「申請」、定義情報「BISワークフロー（LD：申請）」設定時の入力値/返却値の初期値は以下の通りです。




図：アプリケーション種別「BIS」・処理種別「申請」・定義情報「BISワークフロー（LD：申請）」選択時の入力値/返却値の初期値

入力値

ユーザ定義（BIS申請/承認）の入力値はシステムによって自動的に設定された内容を実行対象のフロー・案件に基づいて変更する必要があります。
IM-LogicDesignerではIM-BIS、Formaのフローの処理を取り扱うにあたり、以下の内容を入力値として定義しています。

| 入力値 | 説明 |
|---|---|
| applyUserPlugin<object> | 処理対象者情報（プラグイン情報）です。<br>処理種別「起票」時に指定します。 |
| draftInfo<object> | 起票情報です。<br>処理種別「起票」時に指定します。 |
| applyFromUnapplyInfo<object> | 起票案件の申請情報です。<br>処理種別「起票案件の申請」時に指定します。 |
| applyInfo<object> | 申請情報です。<br>処理種別「申請」時に指定します。 |
| reapplyInfo<object> | 再申請情報です。<br>処理種別「再申請」時に指定します。 |
| approveInfo<object> | 承認情報です。<br>処理種別「承認」時に指定します。 |
| workflowAdvancedSettingInfo<object> | ワークフロー設定情報（高度な設定）です。<br>対象のフローの処理時に以下の設定が必要な場合に指定します。<br>▪ 分岐先選択情報<br>▪ 確認ノード設定情報<br>▪ 動的承認ノード設定情報<br>▪ 横配置ノード設定情報<br>▪ 縦配置ノード設定情報 |
| userParamInfo<object> | ユーザパラメータ設定情報です。<br>ユーザパラメータが存在する場合に指定します。 |
| formaItemInfo<object> | 画面アイテム入力情報です。<br>メインフォームに対象のアイテムが存在する場合に指定します。 |
| formaTableItemInfo<object> | テーブル系画面アイテム入力情報です。<br>メインフォームに対象のアイテムが存在する場合に指定します。 |

入力値の詳細は「 IM-LogicDesigner仕様書 」-「タスク一覧」-「IM-BIS」-「申請」を参照してください。

出力値

IM-LogicDesignerでは、処理種別に基づいて返却値を自動で決定します。
処理種別「起票」「申請」選択時に設定される返却値の詳細は以下の通りです。

| 返却値 | 説明 |
|---|---|

| 返却値 | 説明 |
| --- | --- |
| matterNumber<string> | 入力値に定義したmatterNumberに設定した値が格納されます。<br>入力値で値を設定しない場合、Nullが格納されます。 |
| systemMatterId<string> | 案件の申請実行時にシステムで採番された案件を識別するためのIDが格納されます。 |
| userDataId<string> | 案件の申請実行時にシステムで採番された案件に紐づくユーザデータを識別するためのIDが格納されます。 |

### BIS申請/承認

申請の実行対象のワークフローと処理種別を選択します。
IM-LogicDesignerでは「BIS定義検索」「Forma定義検索」を利用して申請対象のフローを指定します。

| アプリケーション種別 | 説明 |
| --- | --- |
| BIS | 実行対象のフローがBISで作成したワークフローの場合に選択します。 |
| Forma（Workflow） | 実行対象のフローがForma（WF連携）で作成したワークフローの場合に選択します。 |

| 処理種別 | 説明 |
| --- | --- |
| 起票 | フローに対する処理として、起票を行う場合に選択します。<br>詳細は「起票」を参照してください。 |
| 起票案件の申請 | フローに対する処理として、起票案件からの申請を行う場合に選択します。<br>詳細は「起票案件からの申請」を参照してください。 |
| 申請 | フローに対する処理として、新規に申請を行う場合に選択します。 |
| 再申請 | フローに対する処理として、再申請を行う場合に選択します。<br>詳細は「再申請」を参照してください。 |
| 承認 | フローに対する処理として、承認を行う場合に選択します。<br>詳細は「承認」を参照してください。 |

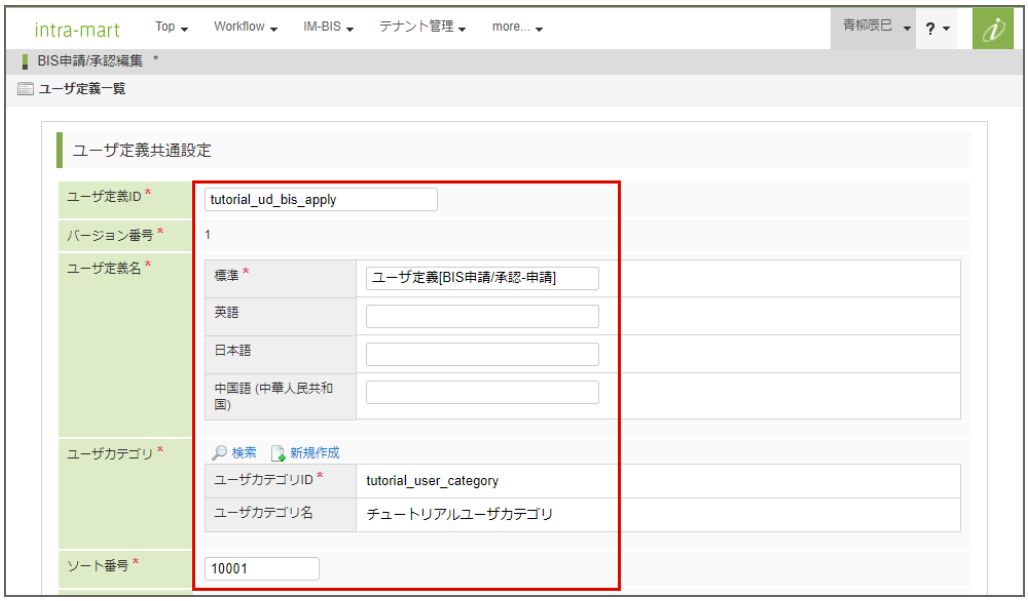
| 定義情報 | 説明 |
| --- | --- |
| BIS ID | 「BIS定義検索」から選択したフローを識別するためのIDが表示されます。 |
| BIS 名 | 「BIS定義検索」から選択したフローの名称が表示されます。 |
| バージョン開始日 | 「BIS定義検索」時の検索基準日に合致するフローのバージョンの開始日が表示されます。 |
| バージョン終了日 | 「BIS定義検索」時の検索基準日に合致するフローのバージョンの終了日が表示されます。 |
| 説明 | 「BIS定義検索」から選択したフローの説明が表示されます。 |
| アプリケーションID | 「Forma定義検索」から選択したフローを識別するためのIDが表示されます。 |
| アプリケーション名 | 「Forma定義検索」から選択したフローの名称が表示されます。 |
| アプリケーション履歴番号 | 「Forma定義検索」から選択したフローのアプリケーション履歴番号が表示されます。 |
| 説明 | 「Forma定義検索」から選択したフローの説明が表示されます。 |

### ユーザ定義（BIS申請/承認）を作成する

最後に、これまでの内容を踏まえてユーザ定義（BIS申請/承認）を作成します。

1. 「BIS申請/承認定義編集」画面を表示します。
2. ユーザ定義の基本情報となる各項目に以下の値を入力します。
   - ユーザ定義ID「tutorial_ud_bis_apply」
   - バージョン「1」（固定）
   - ユーザ定義名
     - 標準 - 「ユーザ定義[BIS申請/承認-申請]」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし

- カテゴリ
  - カテゴリID - 「tutorial_user_category」
- ソート番号「10001」




図：基本情報の定義

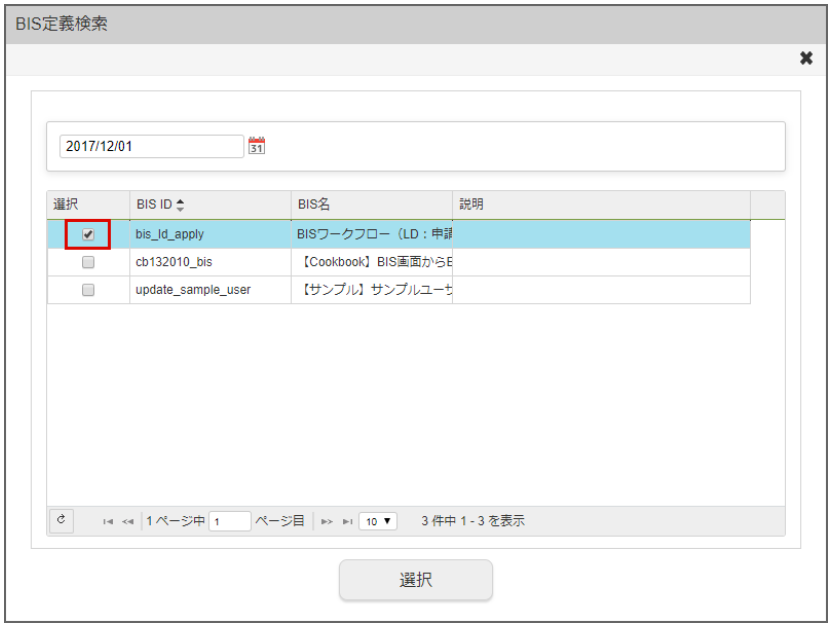
3. ユーザ定義「BIS申請/承認」を定義します。
   アプリケーション種別は「BIS」とします。
   処理種別は「申請」とします。


図：アプリケーション種別処理種別の選択

4. 対象のフローを選択するには、「検索」をクリックします。

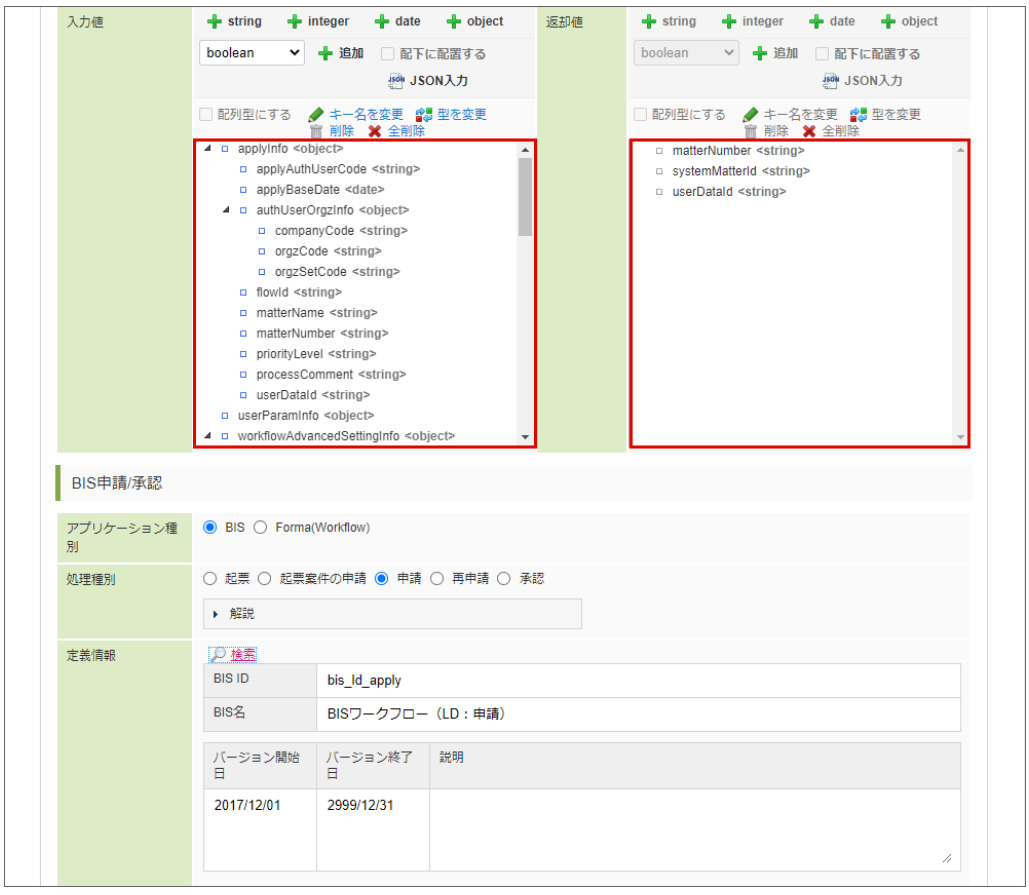
図：BIS定義の選択

5. 「BIS定義検索」で「BISワークフロー（LD：申請）」のチェックボックスをオンにし、「選択」をクリックします。



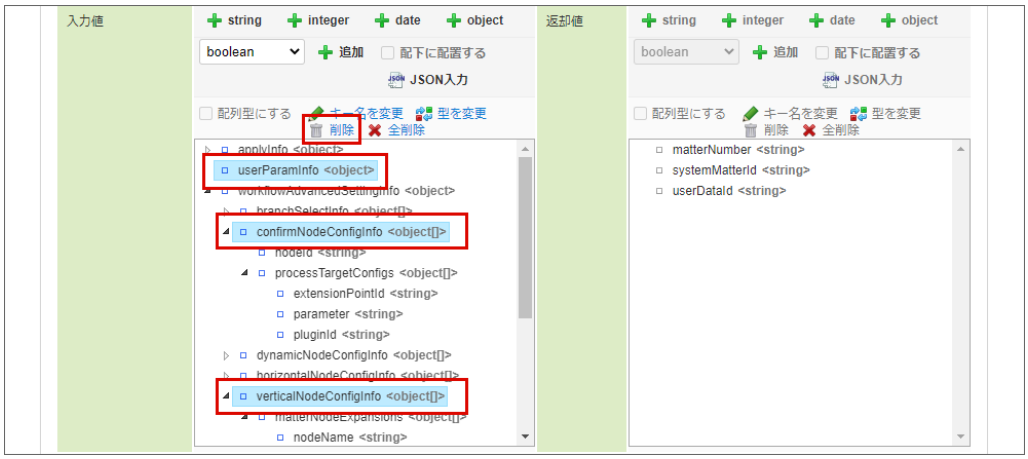

図：フローの検索

6. アプリケーション種別、処理種別、BIS定義の選択後、入力値・返却値が以下のように定義されます。




図：入力値・返却値の定義

7. 入力値については、選択したフローに基づいて取捨選択する必要があります。
選択したフローでは、ユーザパラメータ、確認ノード、縦配置ノードの設定は不要なため、「userParamInfo」、「workflowAdvancedSettingInfo」の配下から「confirmNodeConfigInfo」、「verticalNodeConfigInfo」を削除します。




図：入力値・返却値の定義

8. 「登録」をクリックします。


図：登録

以上で、ユーザ定義（BIS申請/承認）の作成が完了しました。
引き続き、作成したユーザ定義を実行するためにロジックフローを作成していきましょう。

## ロジックフローを作成する

ロジックフローを新規に登録するために、 *ロジックフロー定義編集画面を開く* に基づいてロジックフロー定義編集画面を表示後、以降の手順を実施してください。

入出力設定を定義する

入出力設定では、申請対象のワークフローの申請者に関する情報と処理結果メッセージに関する情報を定義します。

入出力設定には、以下のとおりに設定してください。

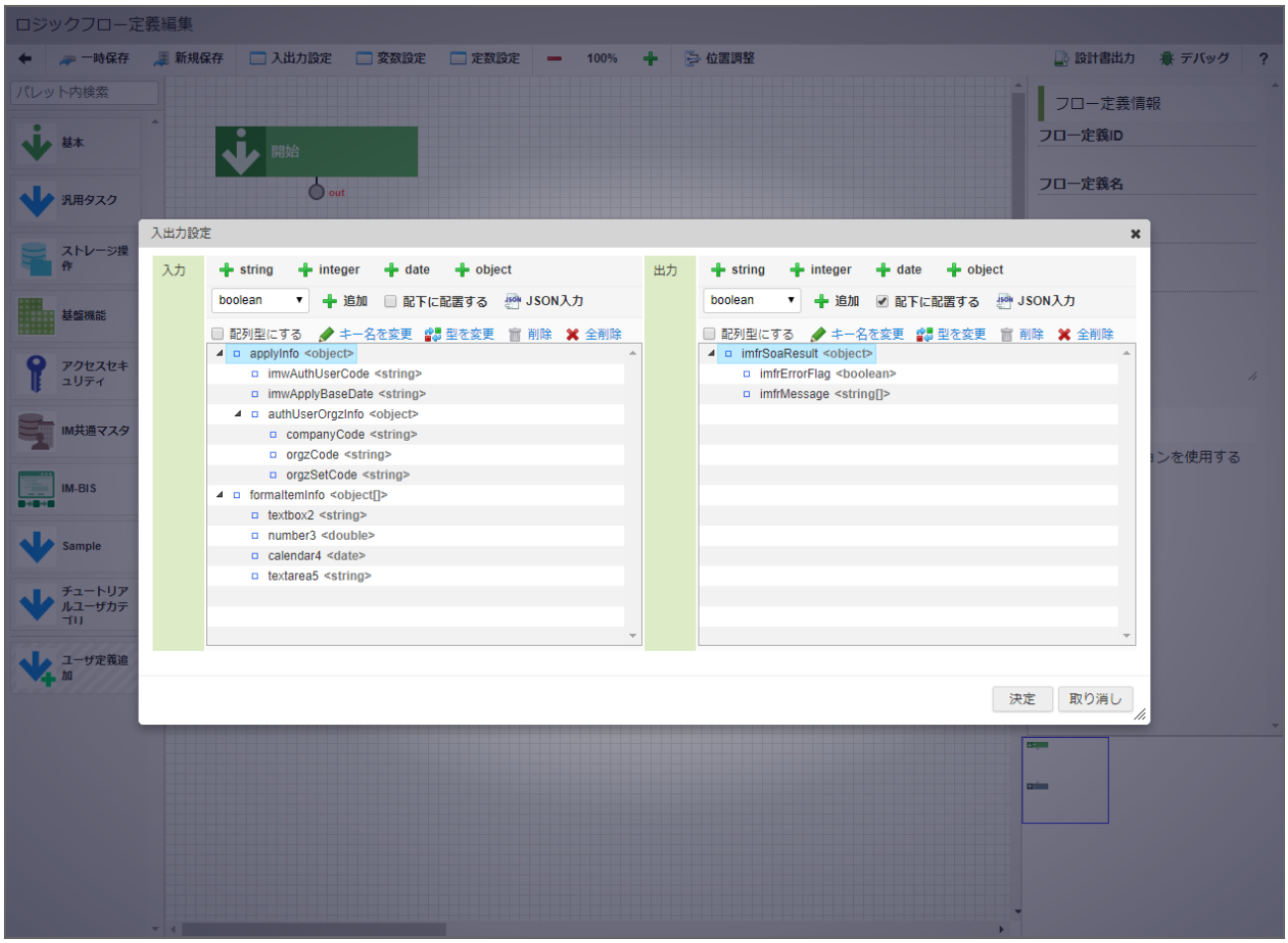
- 入力値
  - 概要
    申請情報 applyInfo 、 画面アイテム入力情報 formaItemInfo を入力値として定義します。
    申請権限者のユーザコード・申請基準日は、IM-BISが暗黙的に連携するシステムパラメータの値を利用できるように、システムパラメータのキー名に基づいて定義します。
    暗黙的に連携するシステムパラメータの詳細は「 IM-BIS 仕様書 」-「 暗黙的に連携するリクエストパラメータの仕様 」を参照してください。
  - パラメータ名・型

| | | |
|---|---|---|
| applyInfo <object> | | |
| | imwAuthUserCode <string> | |
| | imwApplyBaseDate <string> | |
| | authUserOrgzInfo <object> | |
| | | companyCode <string> |
| | | orgzCode <string> |
| | | orgzSetCode <string> |
| formaItemInfo <object[]> | | |
| | textbox2 <string> | |
| | number3 <double> | |
| | calendar4 <date> | |
| | textarea5 <string> | |

- 出力値
  - 概要
    処理結果メッセージ ImfrSoaResult を出力値として定義します。
    処理結果メッセージの詳細は「 IM-BIS 仕様書 」-「 暗黙的に連携するレスポンスパラメータの仕様 」を参照してください。
  - パラメータ名・型

| imfrSoaResult <object> |
| --- |
| imfrErrorFlag <boolean> |
| imfrMessage <string[]> |

上記の内容に基づく入出力設定の内容は以下のとおりです。

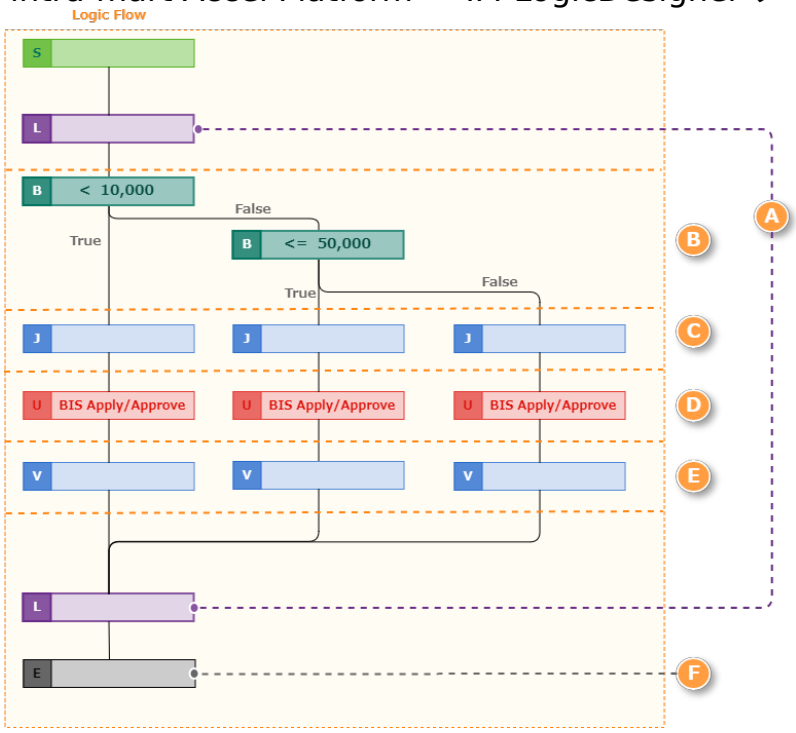



図：入力値および出力値の設定

エレメントを配置する / シーケンスを定義する

フローに必要なエレメントを配置し、実際の処理を定義します。

今回作成するロジックフローは、以下の図のように配置したエレメントで構成しています。




| 記号 | 説明 |
|---|---|
| **A** | 入力画面のテーブルの行数分の回数の処理を実行させるためのループを定義しています。 |
| **B** | 入力画面の金額に基づく分岐条件を定義しています。 |
| **C** | ワークフロー設定情報（高度な設定） workflowAdvancedSettingInfo のうち、以下の設定を定義しています。<br>各設定内容の詳細は「 IM-LogicDesigner仕様書 」-「 IM-BISタスク・申請 」を参照してください。<br>▪ 分岐先選択情報 branchSelectInfo<br>▪ 動的承認ノード設定情報 branchSelectInfo - 処理対象設定（プラグイン情報） processTargetConfigs<br>▪ 横配置ノード設定情報 horizontalNodeConfigInfo - 案件ノード展開情報 matterNodeExpansions |
| **D** | ユーザ定義（BIS申請/承認）を定義しています。 |
| **E** | 処理結果として出力するメッセージの内容を定義しています。 |
| **F** | 処理結果として呼び出し元に返却する返却値を定義しています。<br>本チュートリアルでは、呼び出し元の画面にメッセージを表示するための imfrSoaResult を定義します。 |

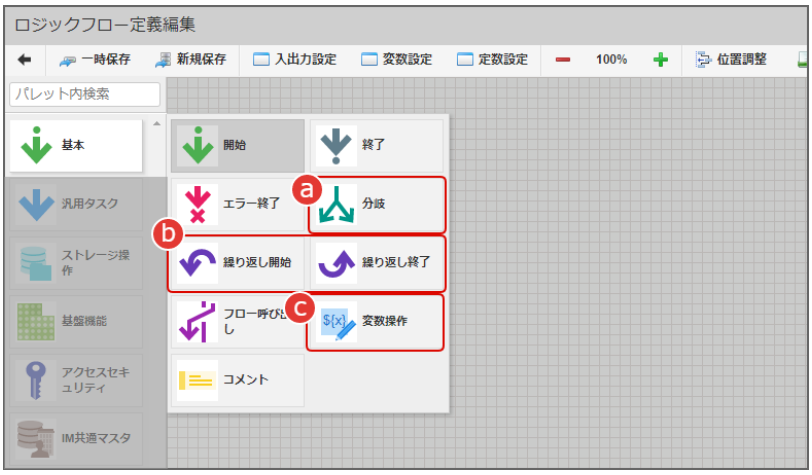
以降の手順でエレメントを配置していきます。
エレメントの配置方法の詳細は *エレメントを配置する* を参照してください。

1. パレット内「基本」から以下のエレメントをそれぞれクリックして追加してください。
   名前の横に数字がある場合は、その数字の個数分追加してください。

   a. 分岐（2）

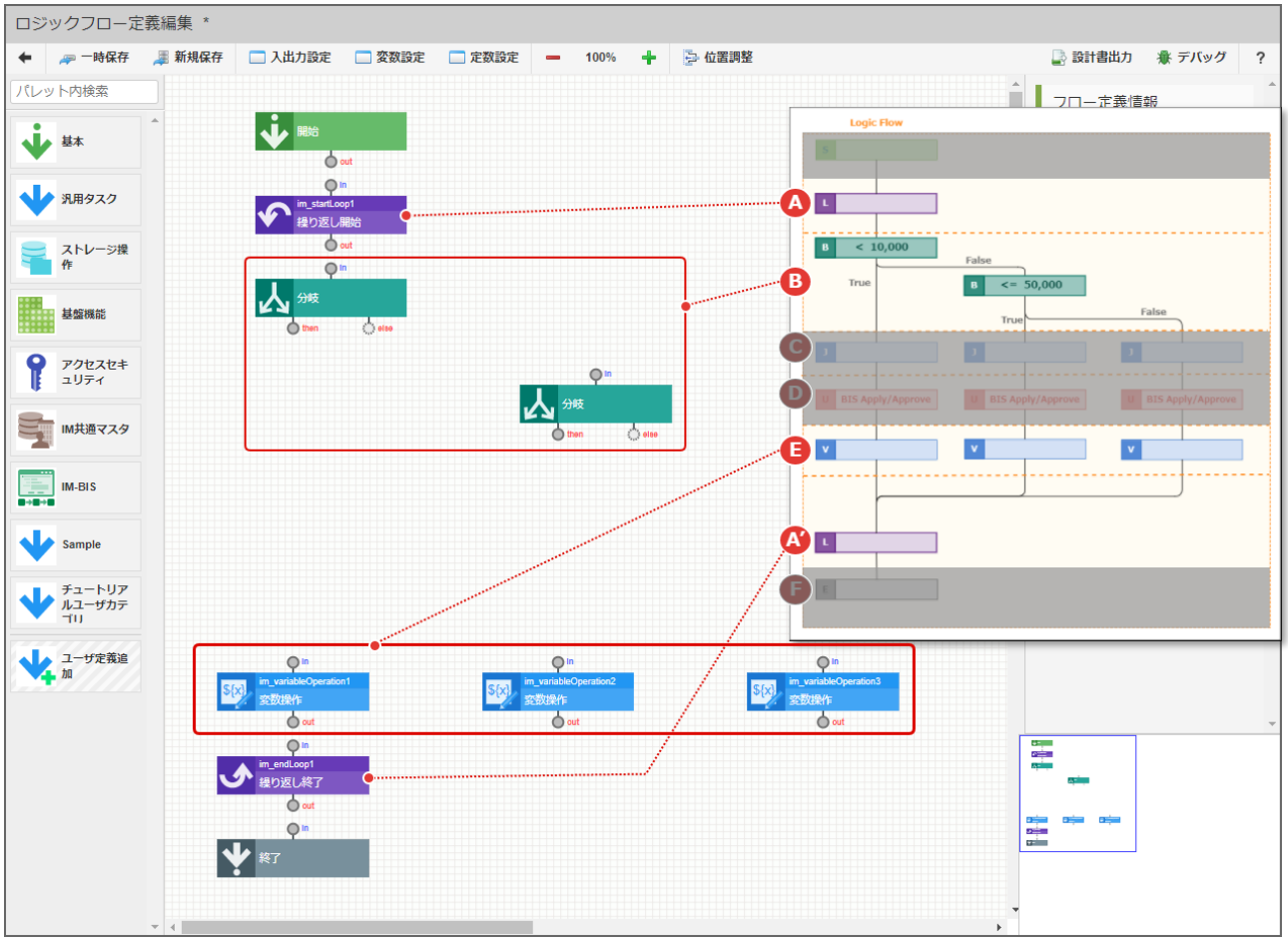
   b. 繰り返し開始・繰り返し終了

   c. 変数操作（3）



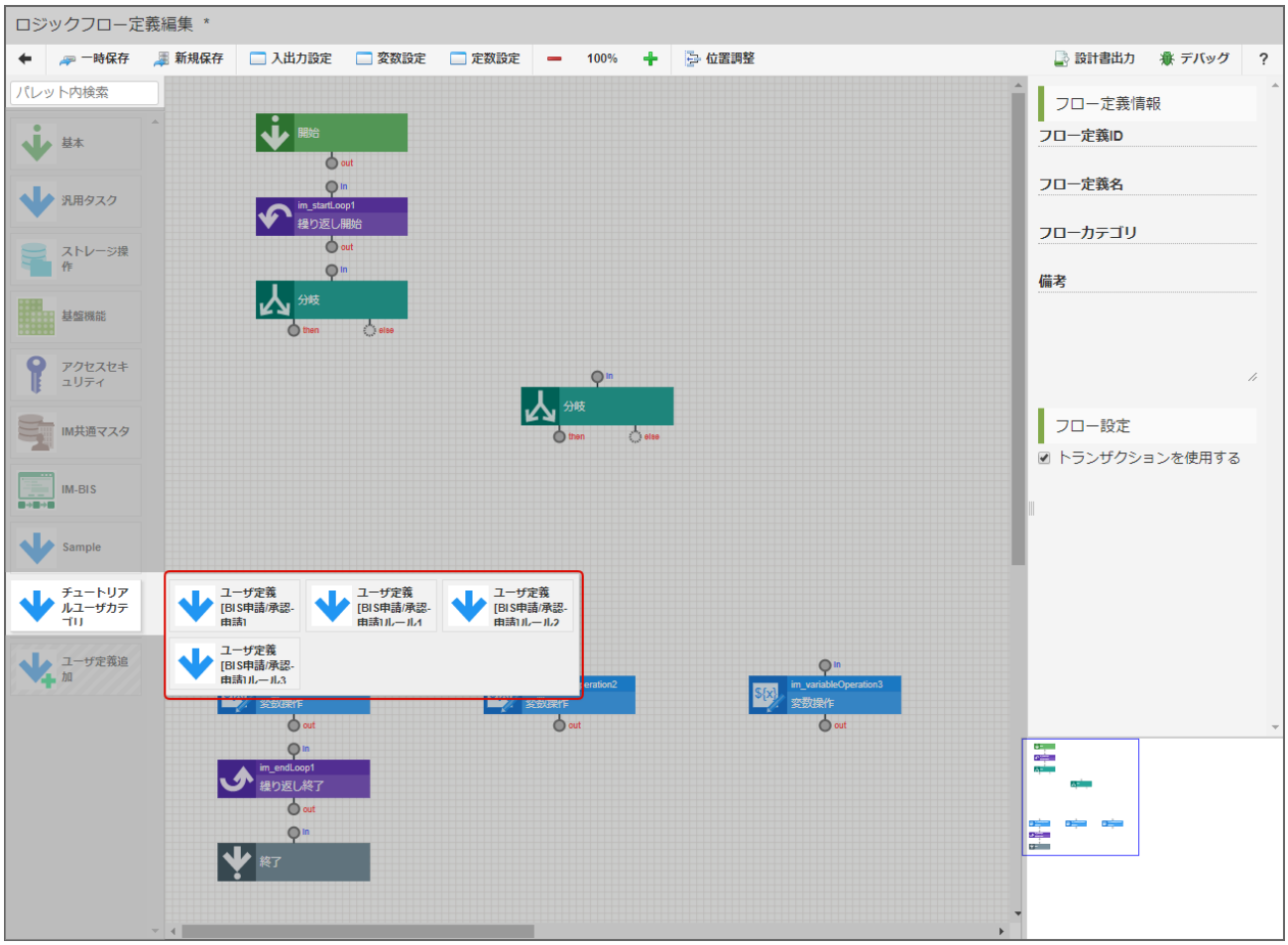

図：「基本」タスク一覧

これらは、エレメントの構成で示した図のA、B、Eに該当します。
図を参考に追加したエレメントを適切な位置に配置してください。
上記の内容に基づいた現時点のエレメントの配置は以下のとおりです。



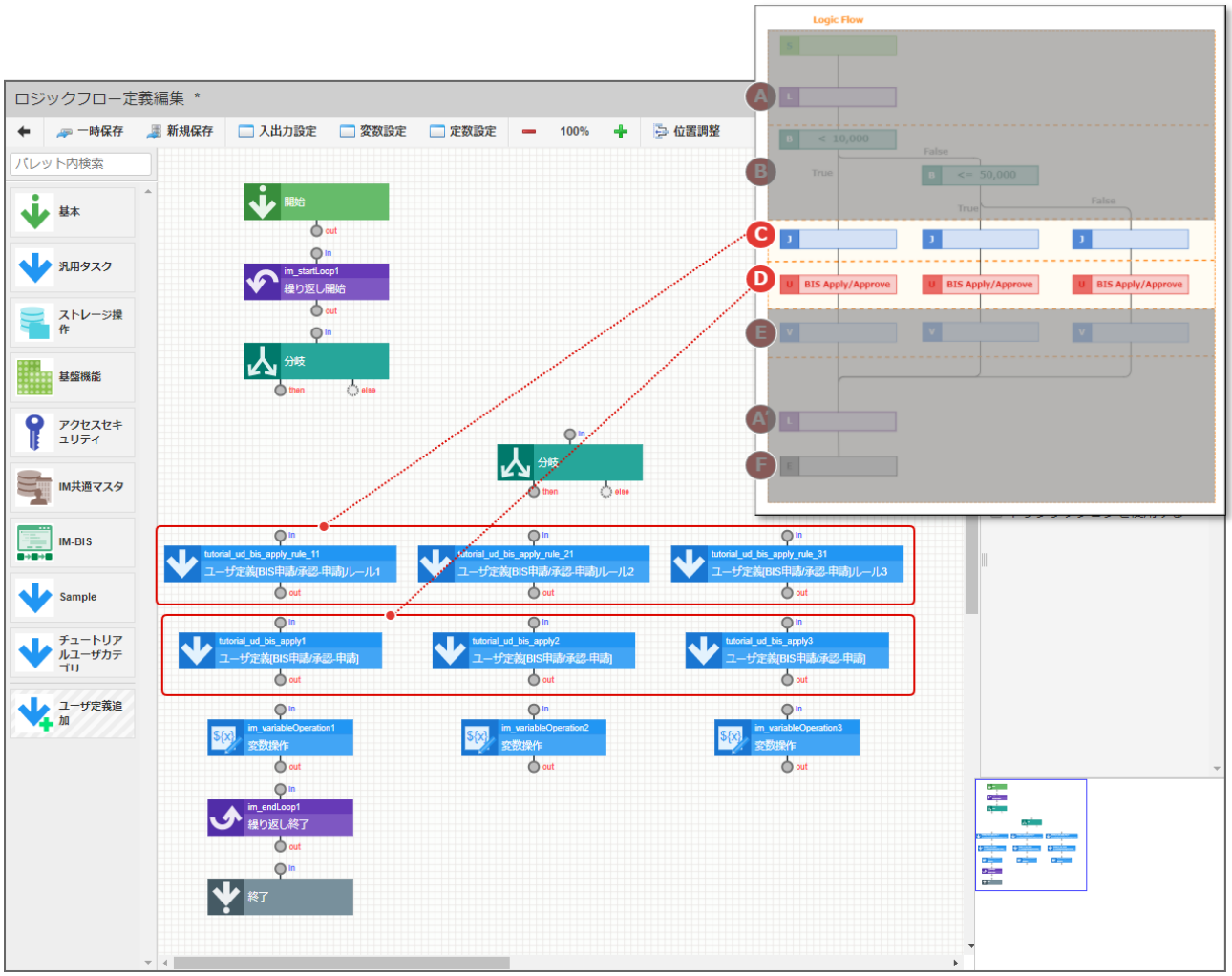

図：基本タスクの配置

2. パレット内「チュートリアルユーザカテゴリ」から以下のエレメントをそれぞれクリックして追加してください。
   名前の横に数字がある場合は、その数字の個数分追加してください。
   - ユーザ定義[BIS申請/承認-申請]（３）
   - ユーザ定義[BIS申請/承認-申請]ルール1
   - ユーザ定義[BIS申請/承認-申請]ルール2
   - ユーザ定義[BIS申請/承認-申請]ルール3




図：「チュートリアルユーザカテゴリ」タスク一覧

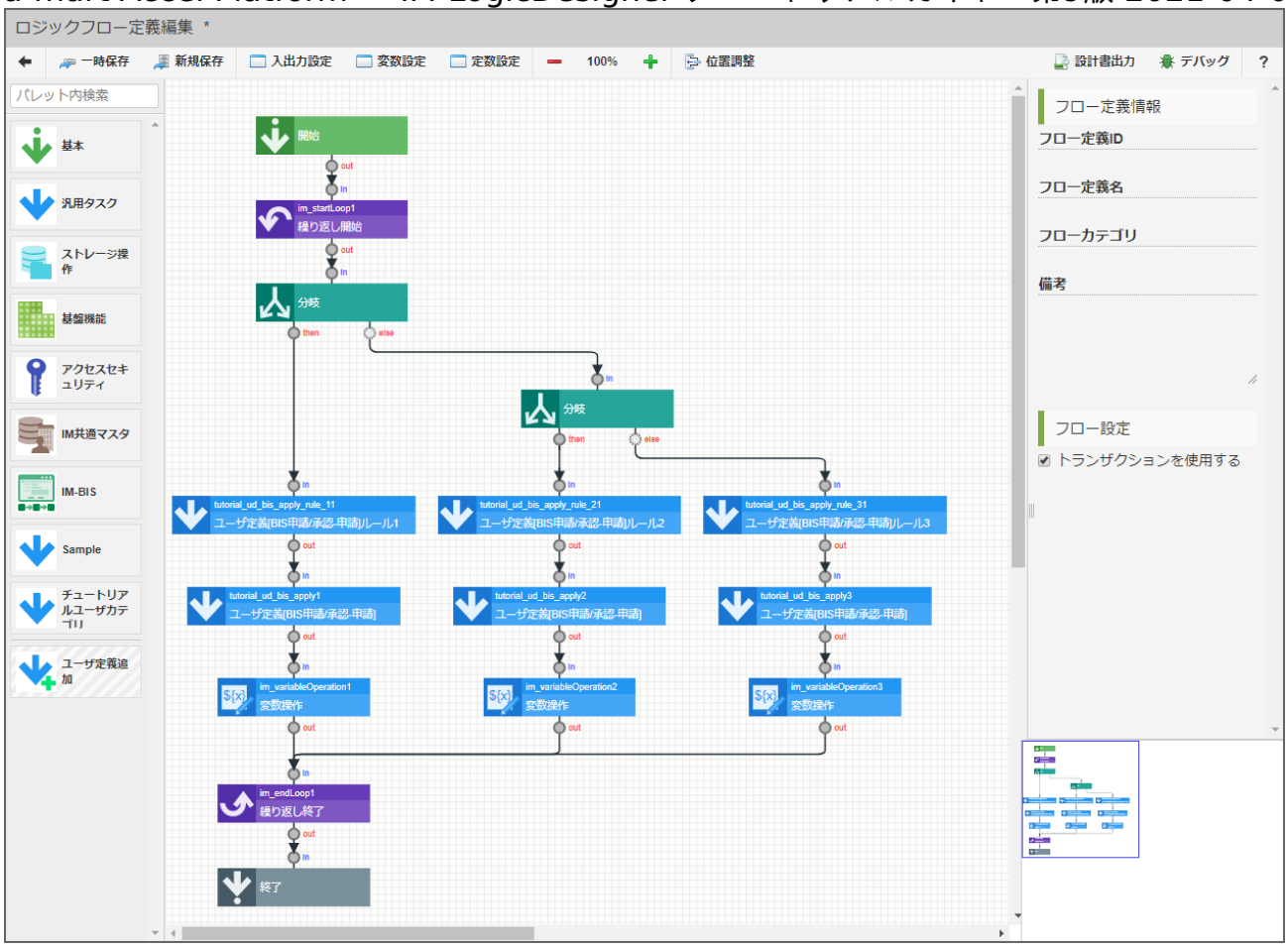
これらは、エレメントの構成で示した図のC、Dに該当します。
図を参考に追加したエレメントを適切な位置に配置してください。
上記の内容に基づいた現時点のエレメントの配置は以下のとおりです。




図：ユーザ定義タスクの配置

以上で、ロジックフローへのエレメントの配置が完了しました。
配置したエレメントに対し、 *線を引く（シーケンスを定義する）* の手順に基づいて、最初に示した図のとおりにエレメントを接続してください。

シーケンスの定義まで完了している状態は以下のとおりです。



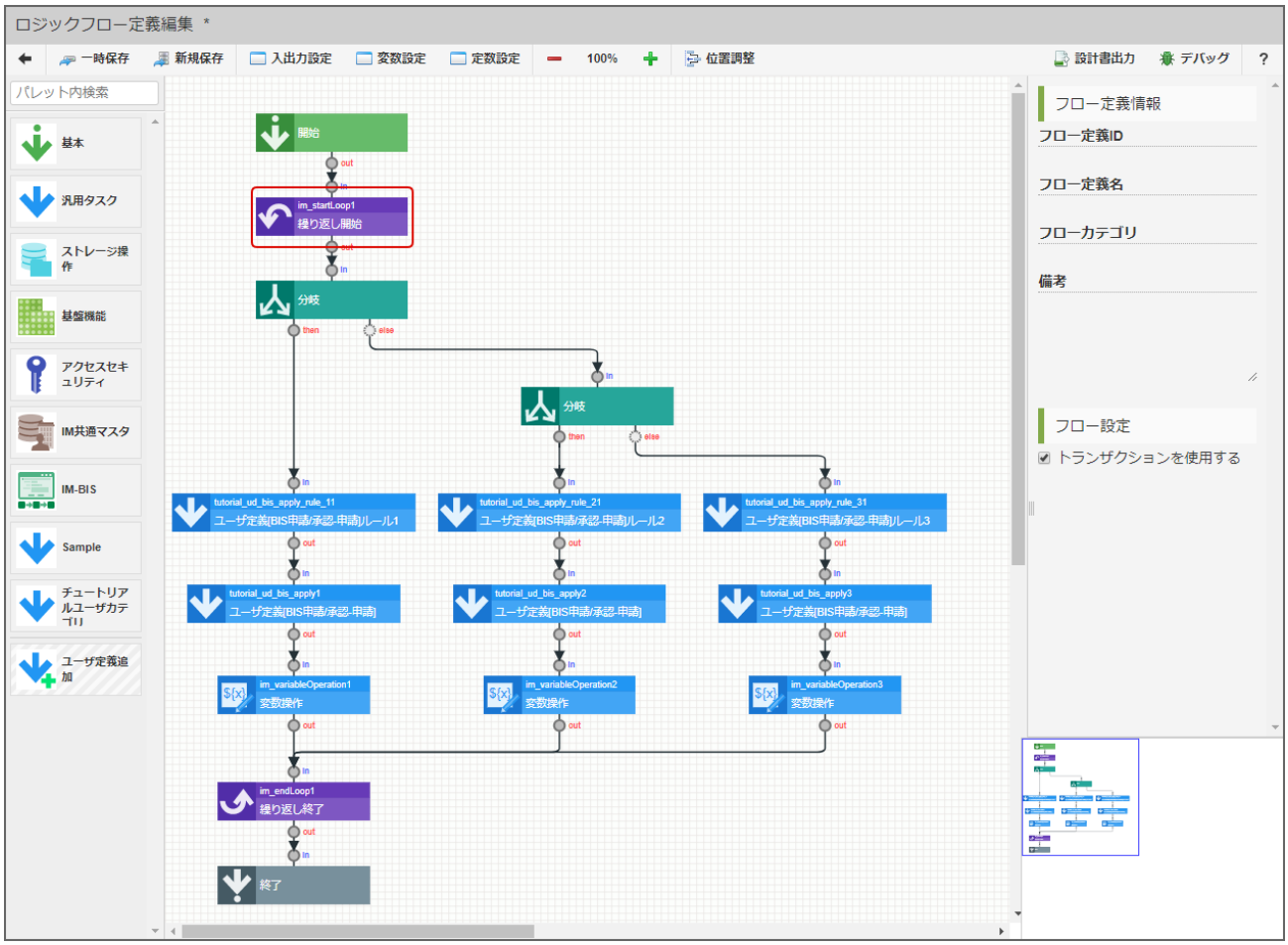

図：全てのエレメントの接続

プロパティを設定する

配置したエレメントのプロパティを設定します。

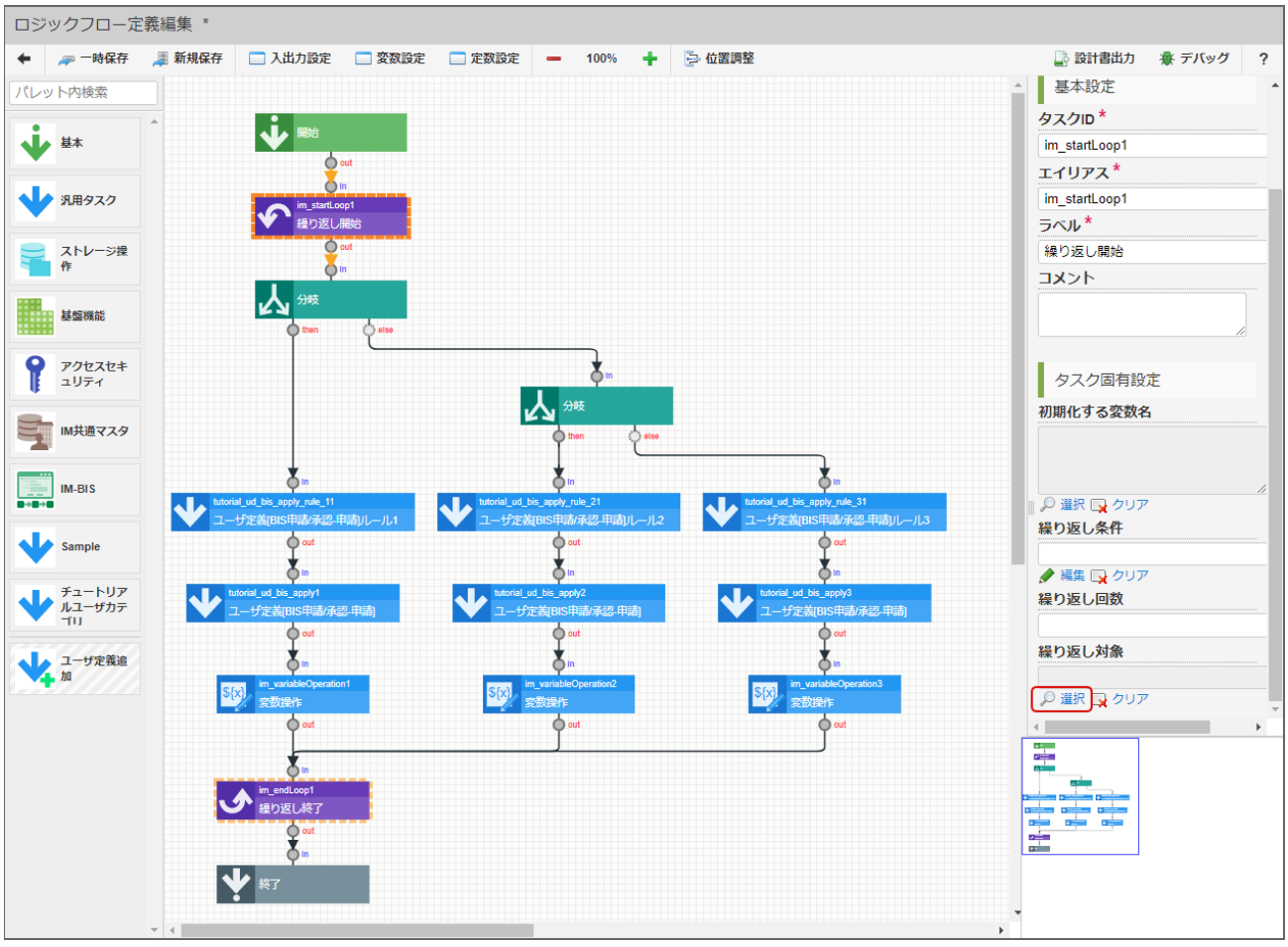
はじめに、「繰り返し開始」タスクのプロパティを設定します。

1. フロー編集画面上の「繰り返し開始」タスクをクリックします。



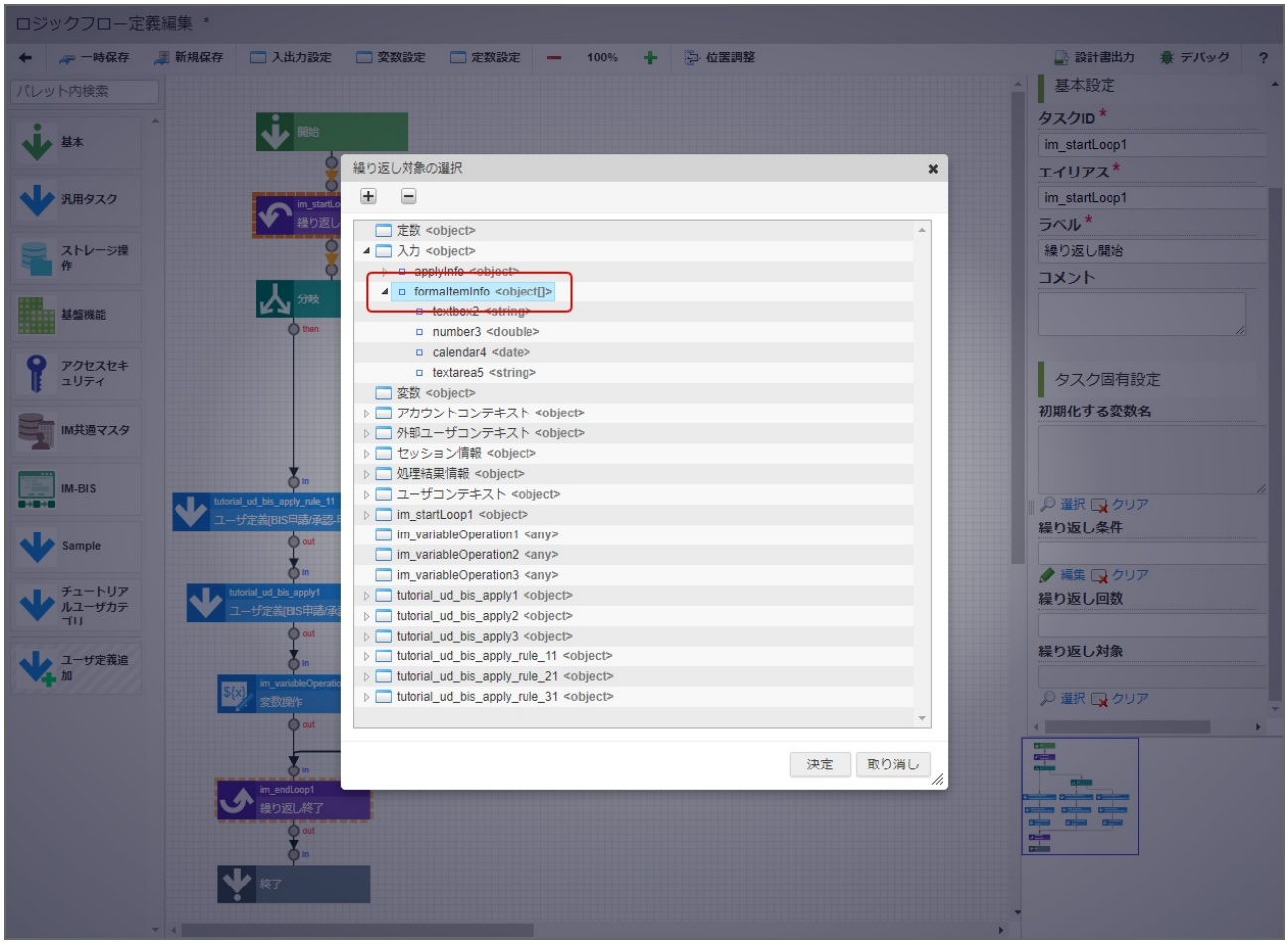

図：「繰り返し開始」タスク

2. 画面右側に「繰り返し開始」タスクに関するプロパティ画面が表示されます。
タスク固有設定の「繰り返し対象」の「選択」をクリックします。

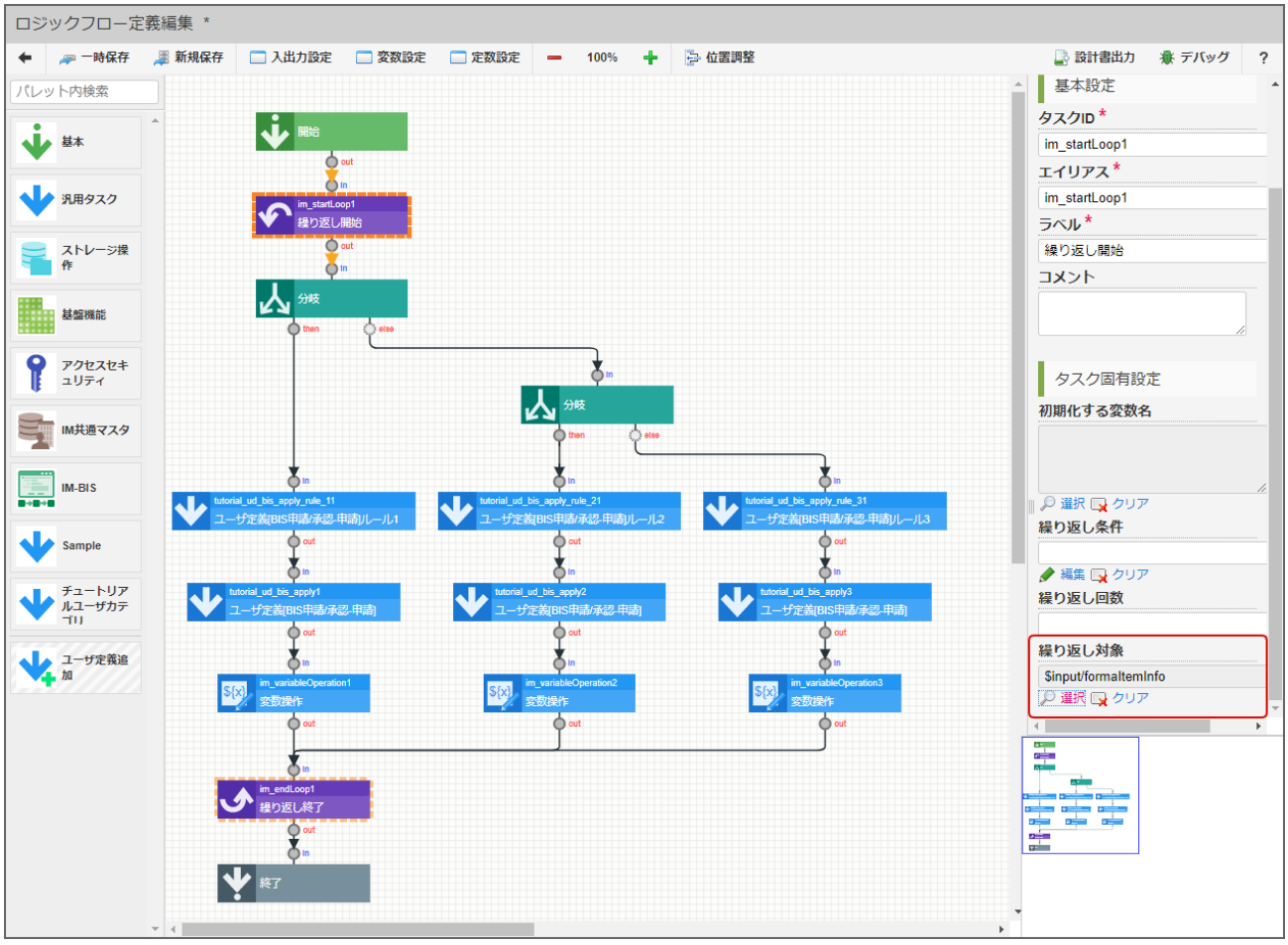


図：「繰り返し開始」タスクのプロパティ画面

3. 入力 - formaItemInfoを選択し、「決定」をクリックします。




図：繰り返し対象の選択

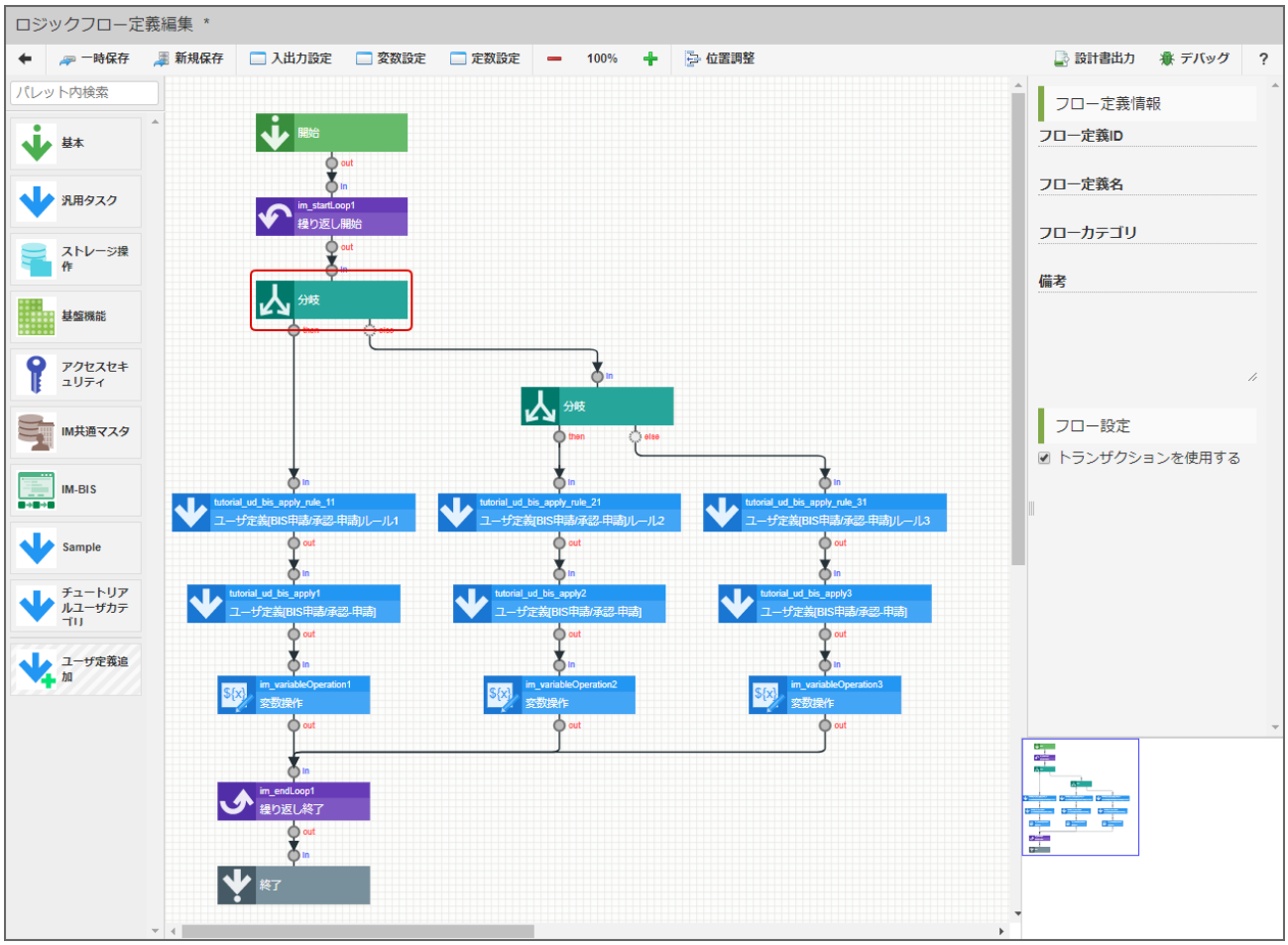


図：「繰り返し開始」タスクのプロパティの設定

4. 以上で、入力画面のテーブルの行数分の繰り返し処理が設定できました。

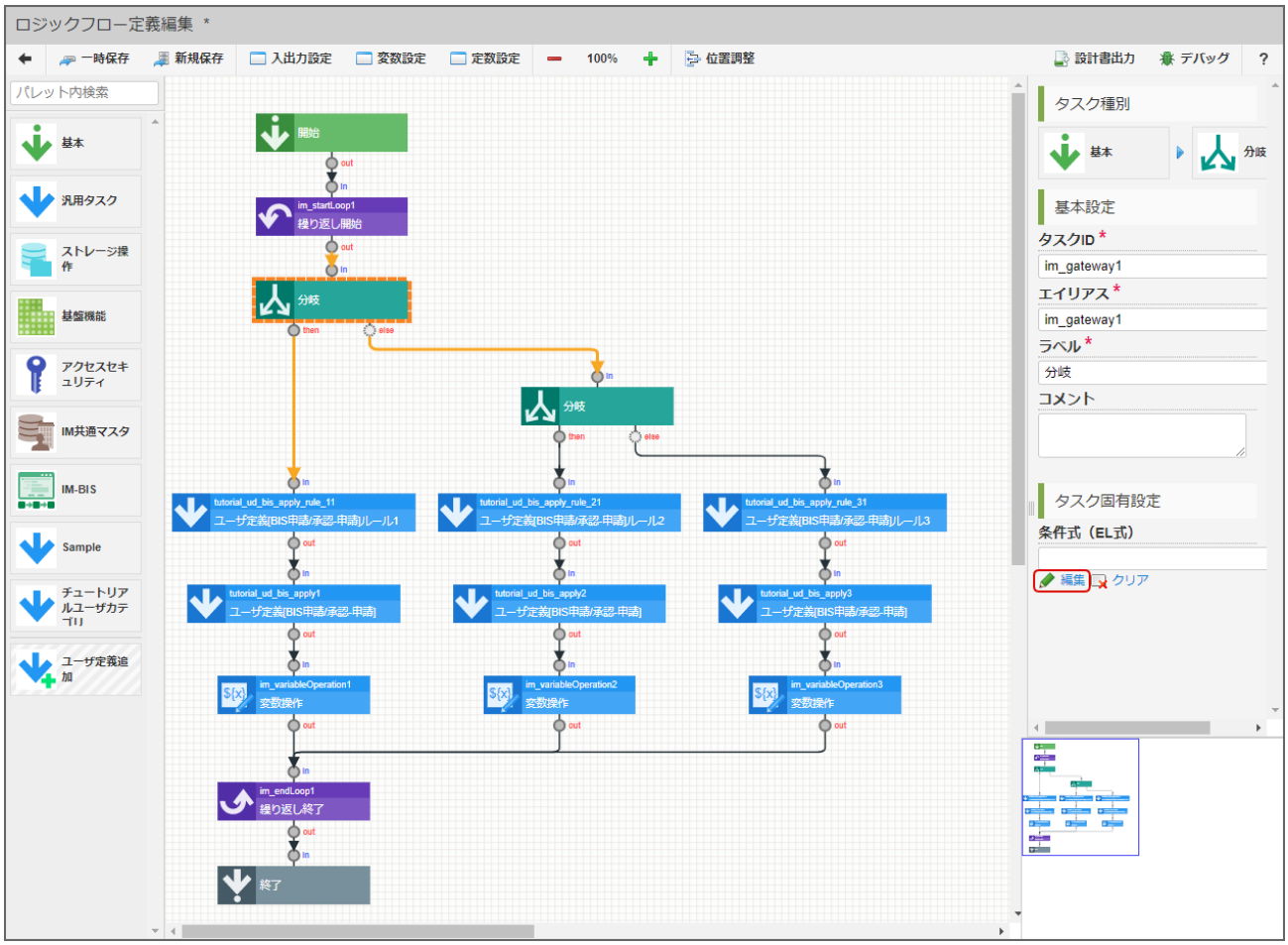
続いて、「分岐」制御要素のプロパティを設定します。
2つの分岐のうち、ユーザ定義[BIS申請/承認-申請]ルール1に接続する分岐条件を「10,000円未満の場合に実行」とするように設定します。

1. フロー編集画面上の「分岐」タスクをクリックします。



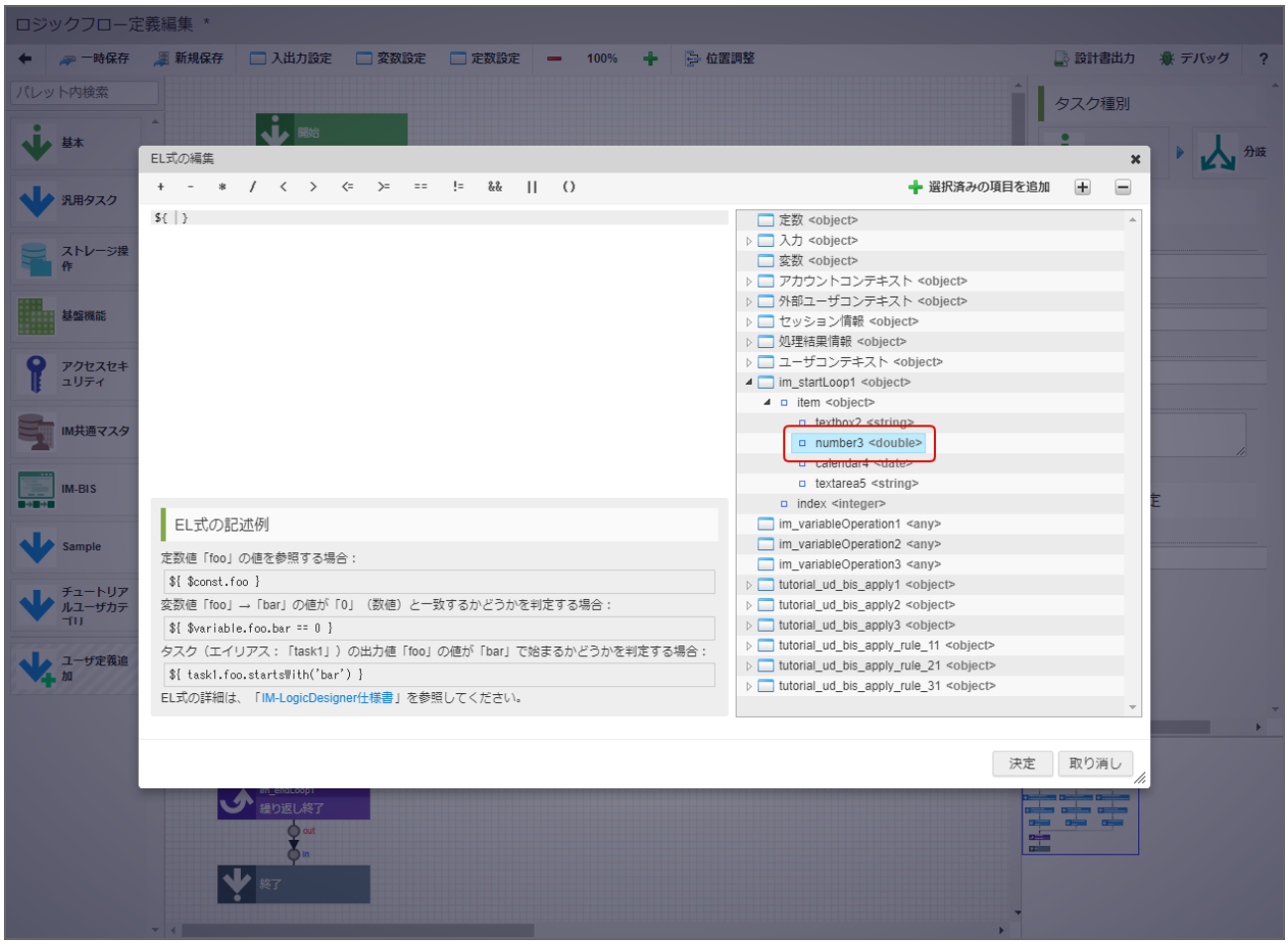

図：「分岐」制御要素を選択

2. タスク固有設定の「条件式（EL式）」の「編集」をクリックします。



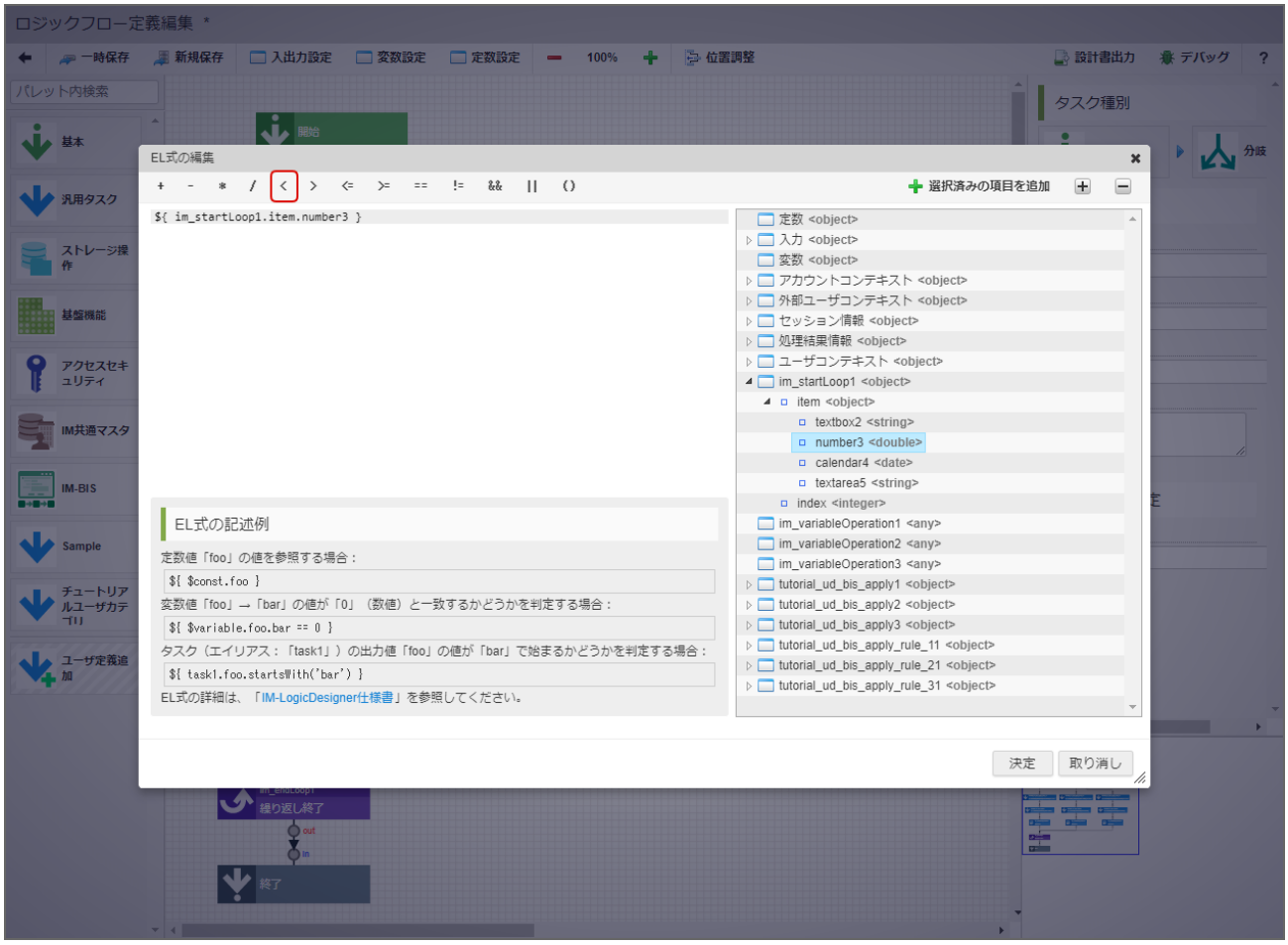

図：「分岐」制御要素のタスク固有設定

3. im_startLoop1<object> （繰り返し開始タスクのタスクID） - item - number3 をダブルクリックします。
式に ${ im_startLoop1.item.number3 } と表示されます。



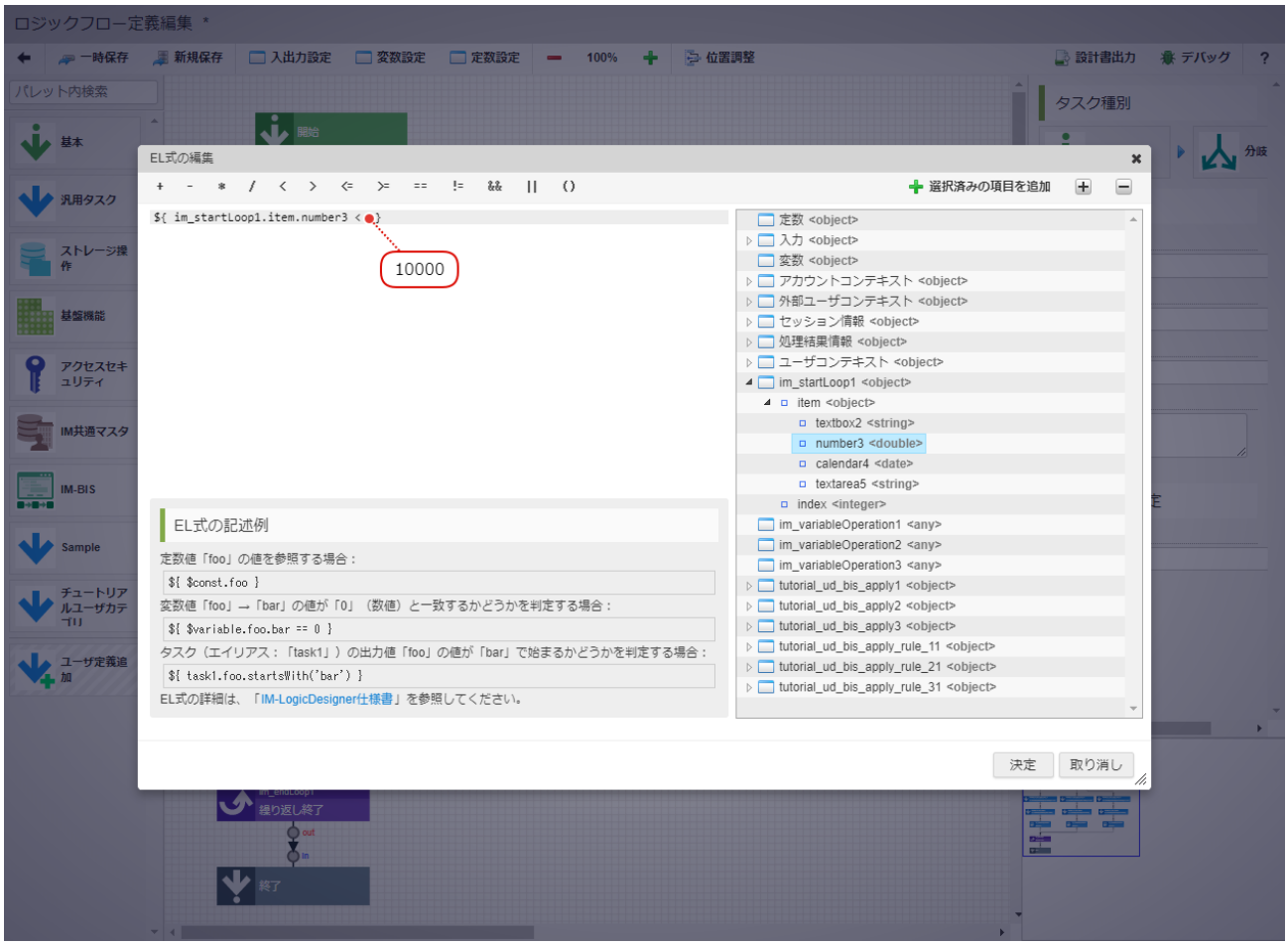
図：EL式の編集

4. 続けて、不等号の `<` をクリックします。



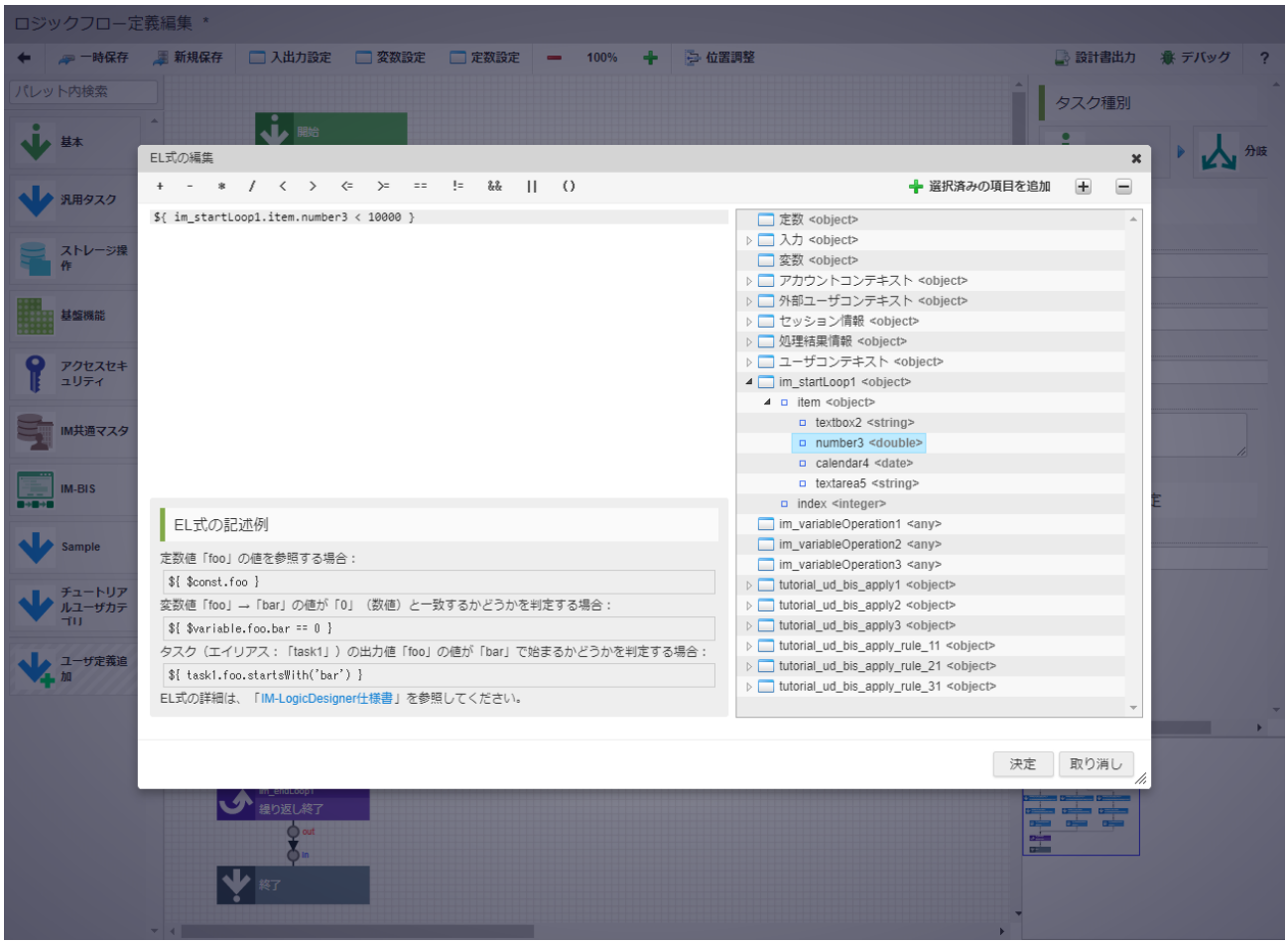

図：EL式に不等号を追加

5. 挿入した不等号の次に、比較する値の `10000` を入力します。



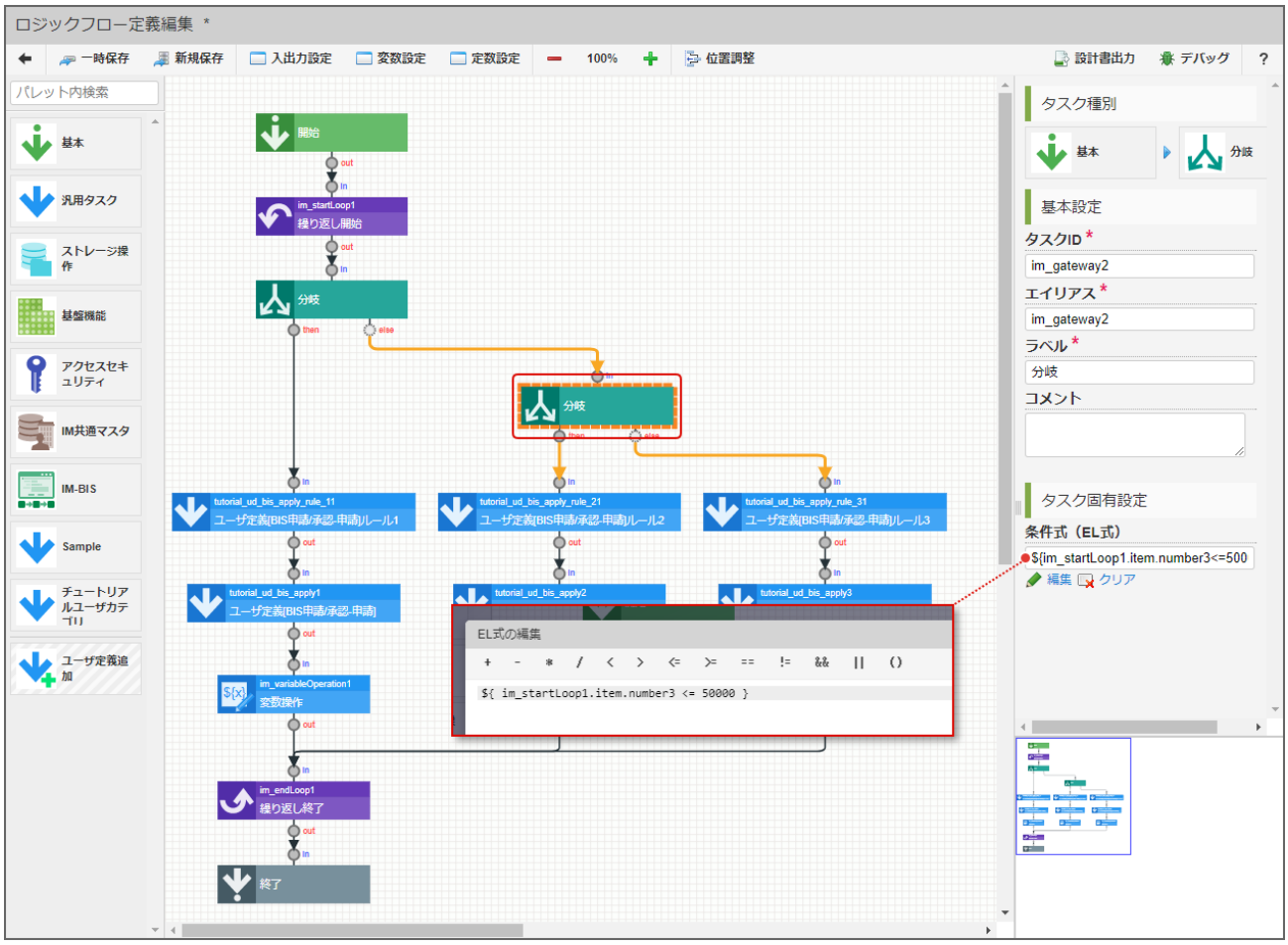

図：EL式に値を追加

6. 設定画面右下の 決定 をクリックし、「分岐」制御要素の条件式の定義を終了します。




図：「分岐」制御要素の条件式の定義（その1）

同様の手順で、もう一方の分岐の条件を「金額が10000円以上かつ50000円以下」と設定します。
（先に設定した分岐の条件で「10000円以上であること」は設定されているため、「50000円以下であること」のみを条件に定義します。
）



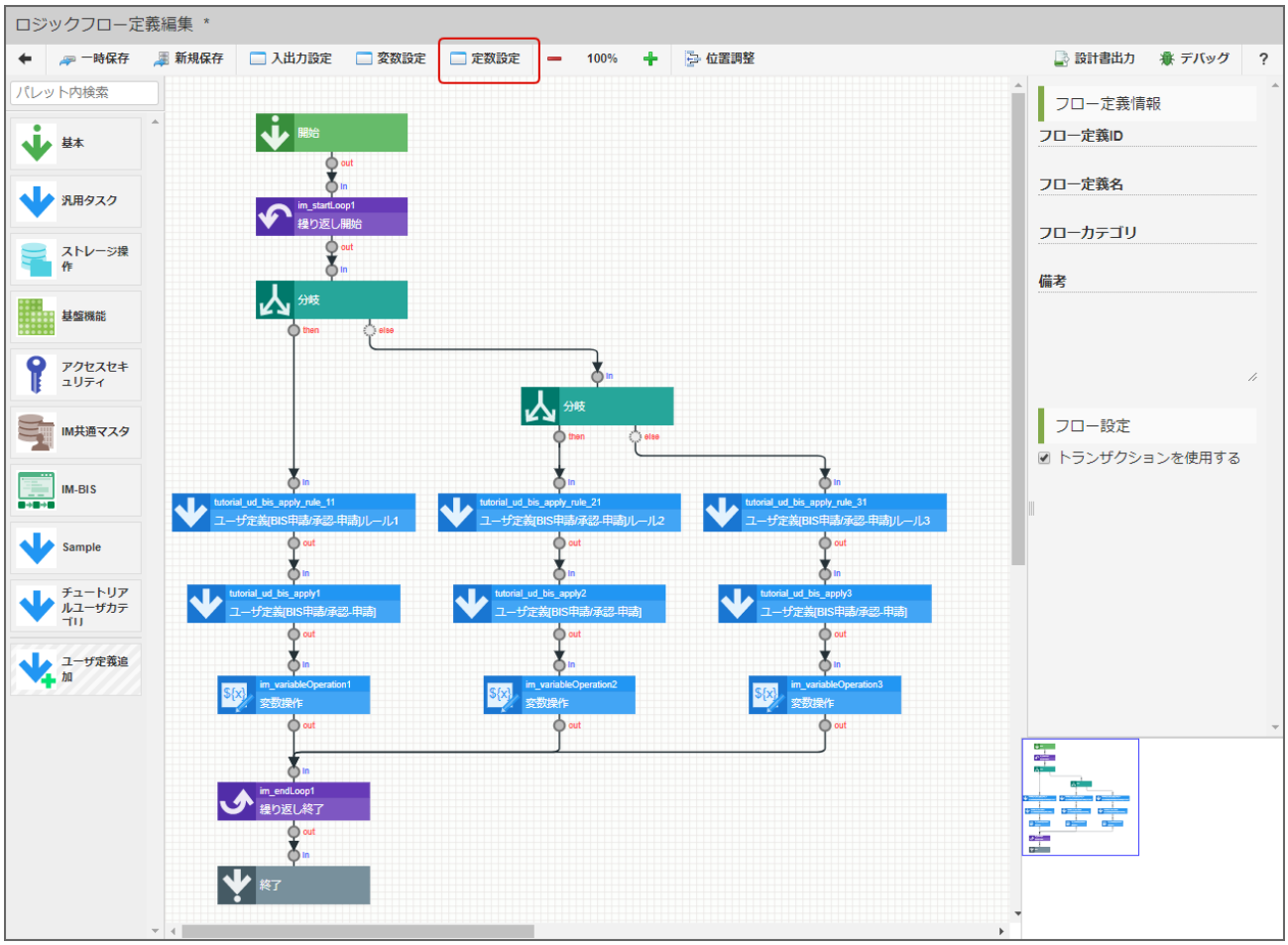

図：「分岐」制御要素の条件式の定義（その2）

以上で、プロパティ設定が全て完了しました。
続いて、ロジックフロー内で利用する定数値・変数を定義します。

定数値・変数を設定する

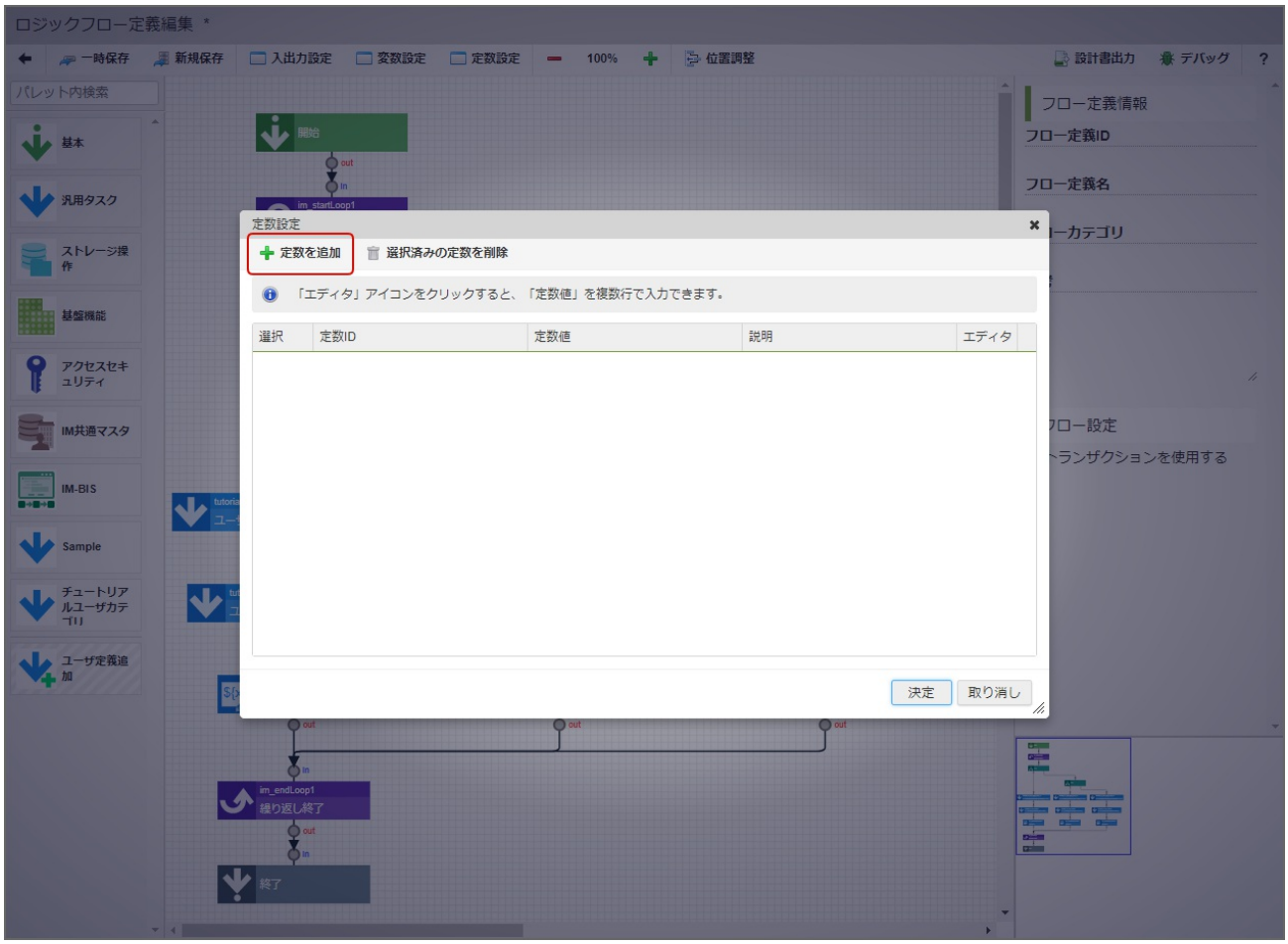
定数では、ユーザ定義（BIS申請/承認）の実行に必要な情報のうち、フローIDやノードIDなどの固定の値を定義します。
チュートリアルでは、そのほかに処理結果メッセージの固定の文言なども定数に定義していきます。

1. ロジックフロー定義編集画面上部、ヘッダ内の「定数設定」をクリックします。




図：「定数設定」をクリック

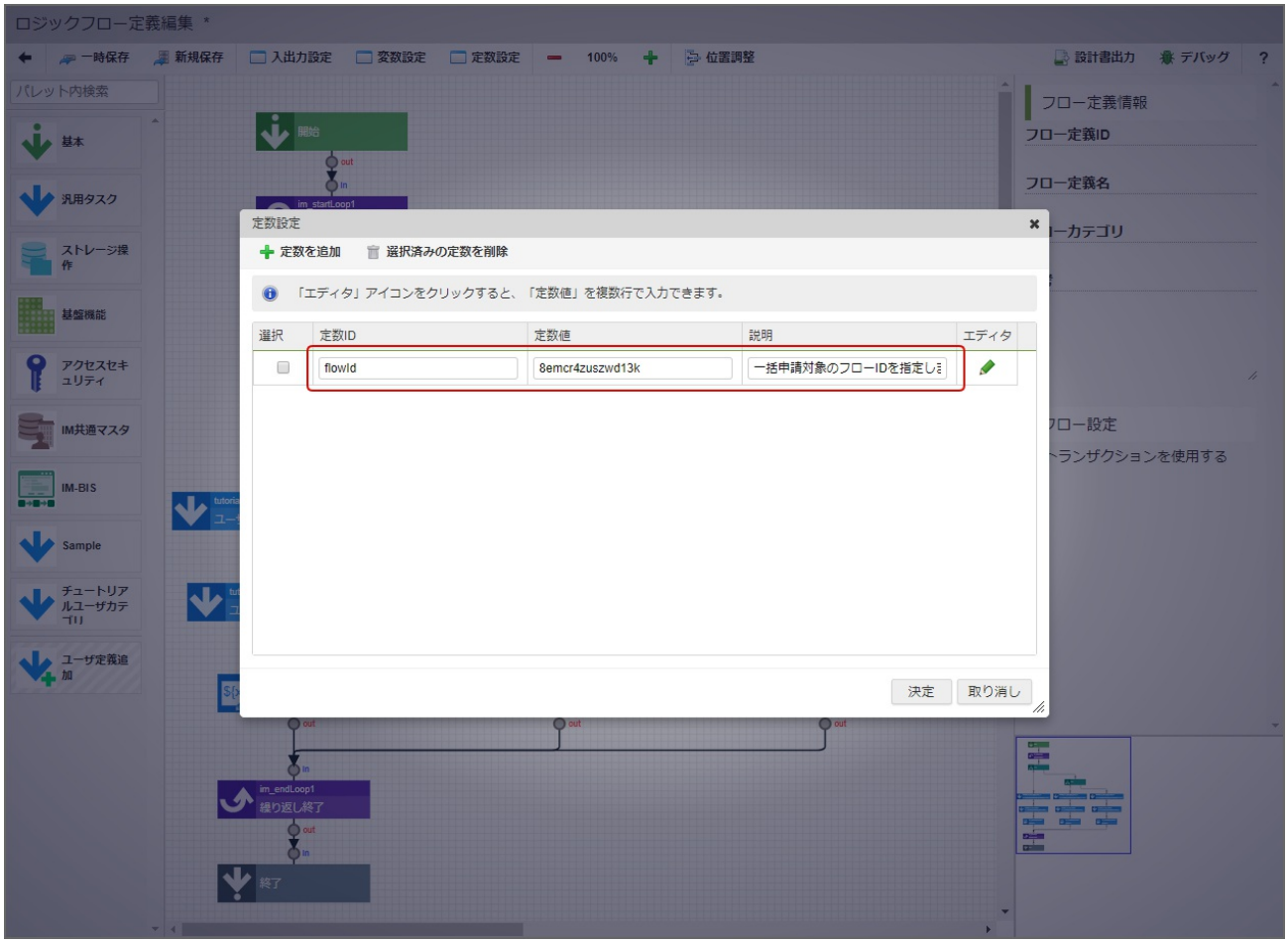
2. 「定数を追加」をクリックします。




図：定数設定画面

3. 定数に「フローID」を登録するために、追加された行には以下のとおりに入力します。

| パラメータ名（定数ID） | 定数値 | 説明 |
|---|---|---|
| flowId | 8emcr4zuszwd13k | 一括申請対象のフローIDを指定します。 |

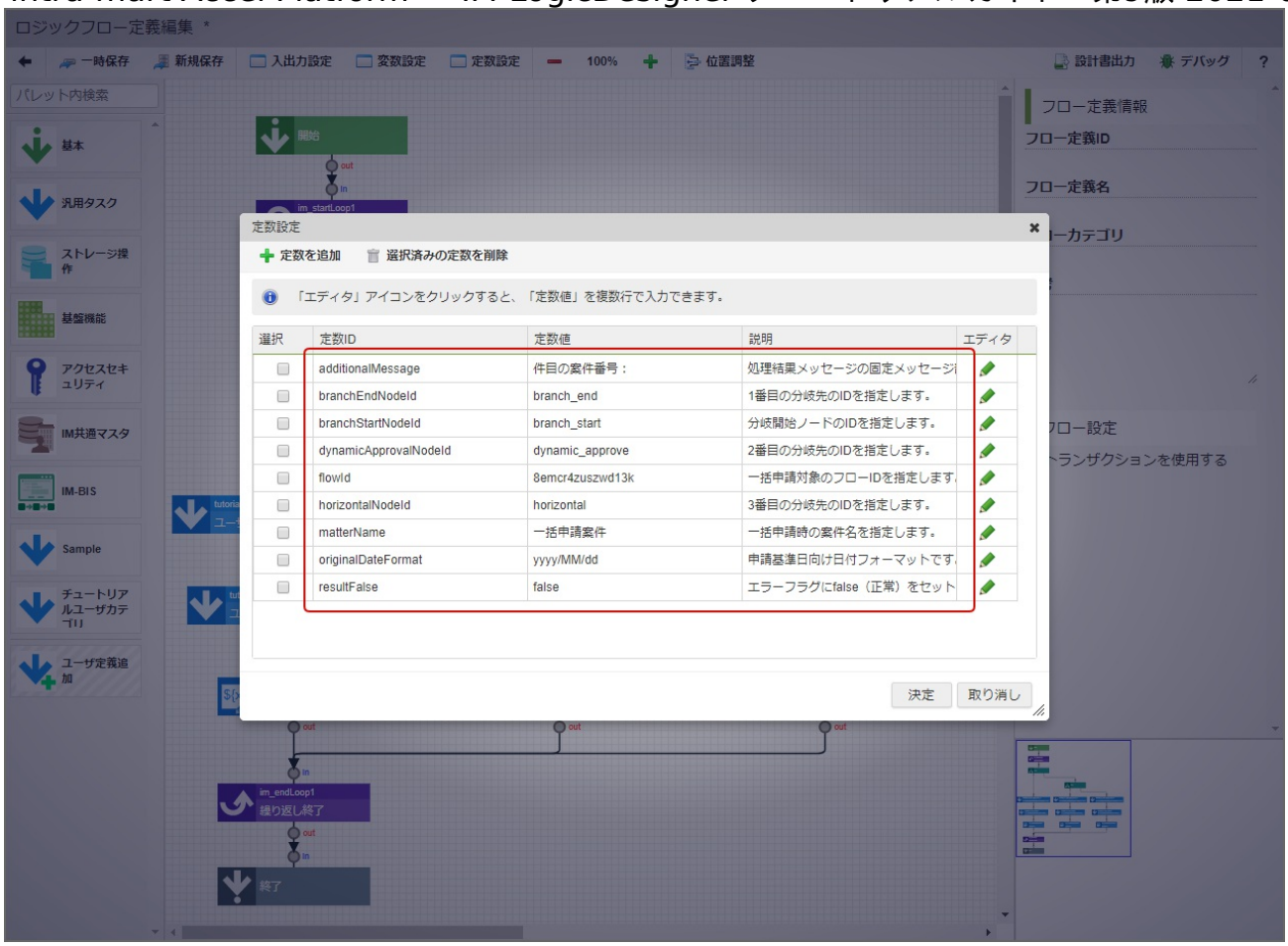



図：定数「フローID」の設定

4. 同様の手順で、以下の定数についても定義します。

| パラメータ名（定数ID） | 定数値 | 説明 |
|---|---|---|
| additionalMessage | 件目の案件番号： | 処理結果メッセージの固定メッセージ部分を指定します。 |
| branchEndNodeId | branch_end | 1番目の分岐先のIDを指定します。 |
| branchStartNodeId | branch_start | 分岐開始ノードのIDを指定します。 |
| dynamicApprovalNodeId | dynamic_approve | 2番目の分岐先のIDを指定します。 |
| horizontalNodeId | horizontal | 3番目の分岐先のIDを指定します。 |
| matterName | 一括申請案件 | 一括申請時の案件名を指定します。 |
| originalDateFormat | yyyy/MM/dd | 申請基準日向け日付フォーマットです。 |
| resultFalse | false | エラーフラグにfalse（正常）をセットします。 |

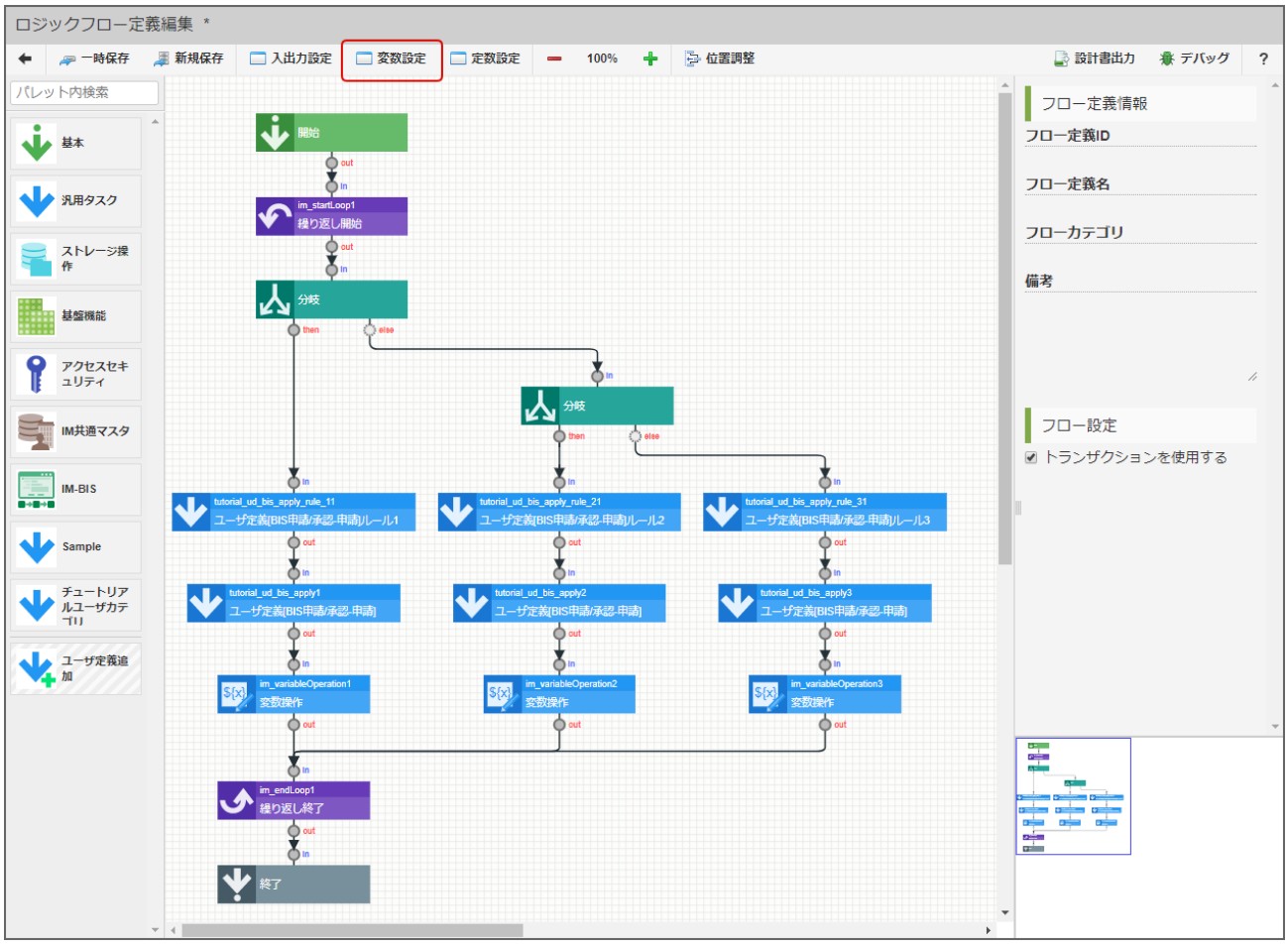
上記の内容に基づく定数設定の内容は以下のとおりです。




図：全ての定数の設定

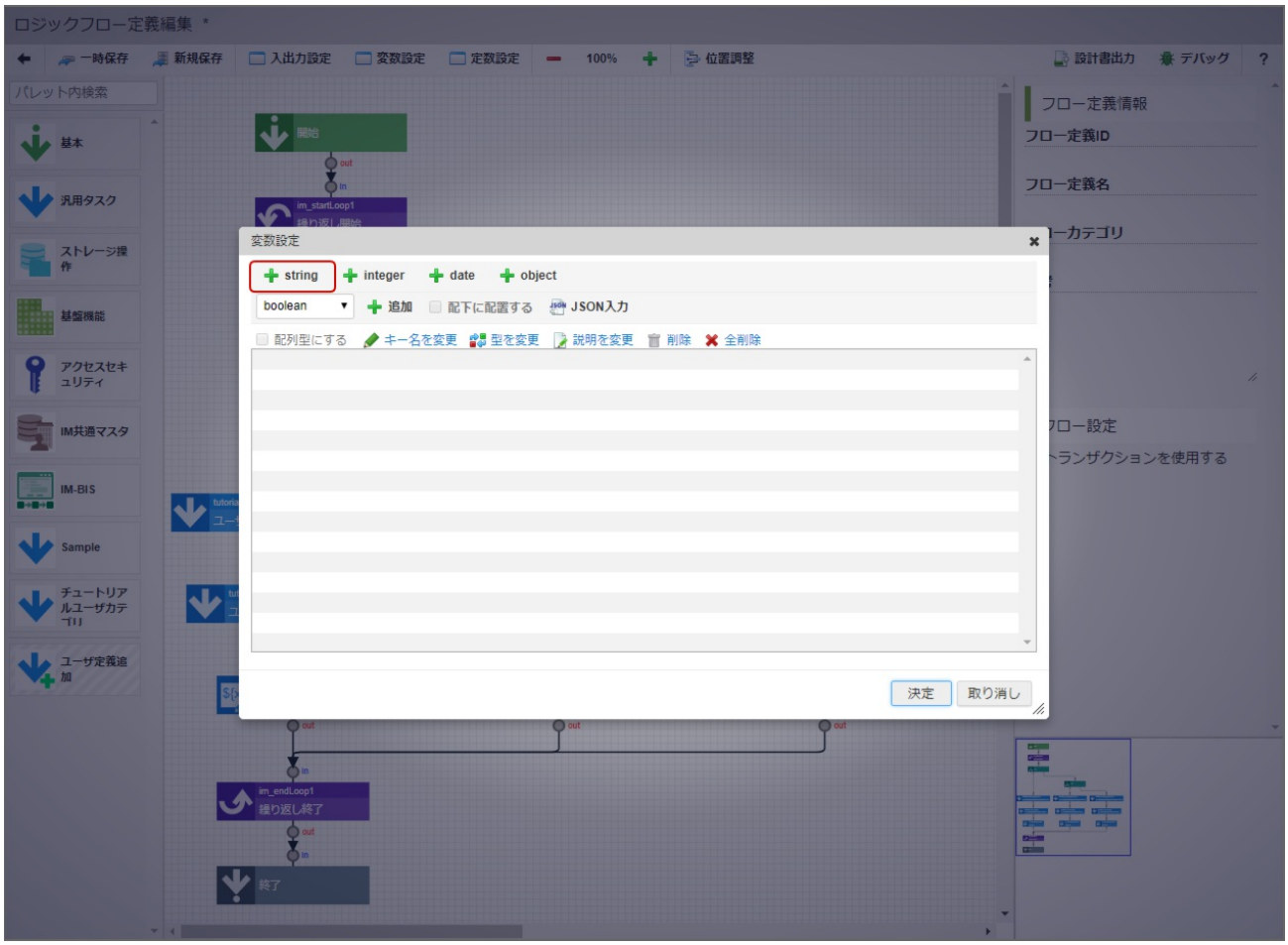
変数では、出力値に設定している処理結果メッセージを格納する変数を定義します。

1. ロジックフロー定義編集画面上部、ヘッダ内の「変数設定」をクリックします。



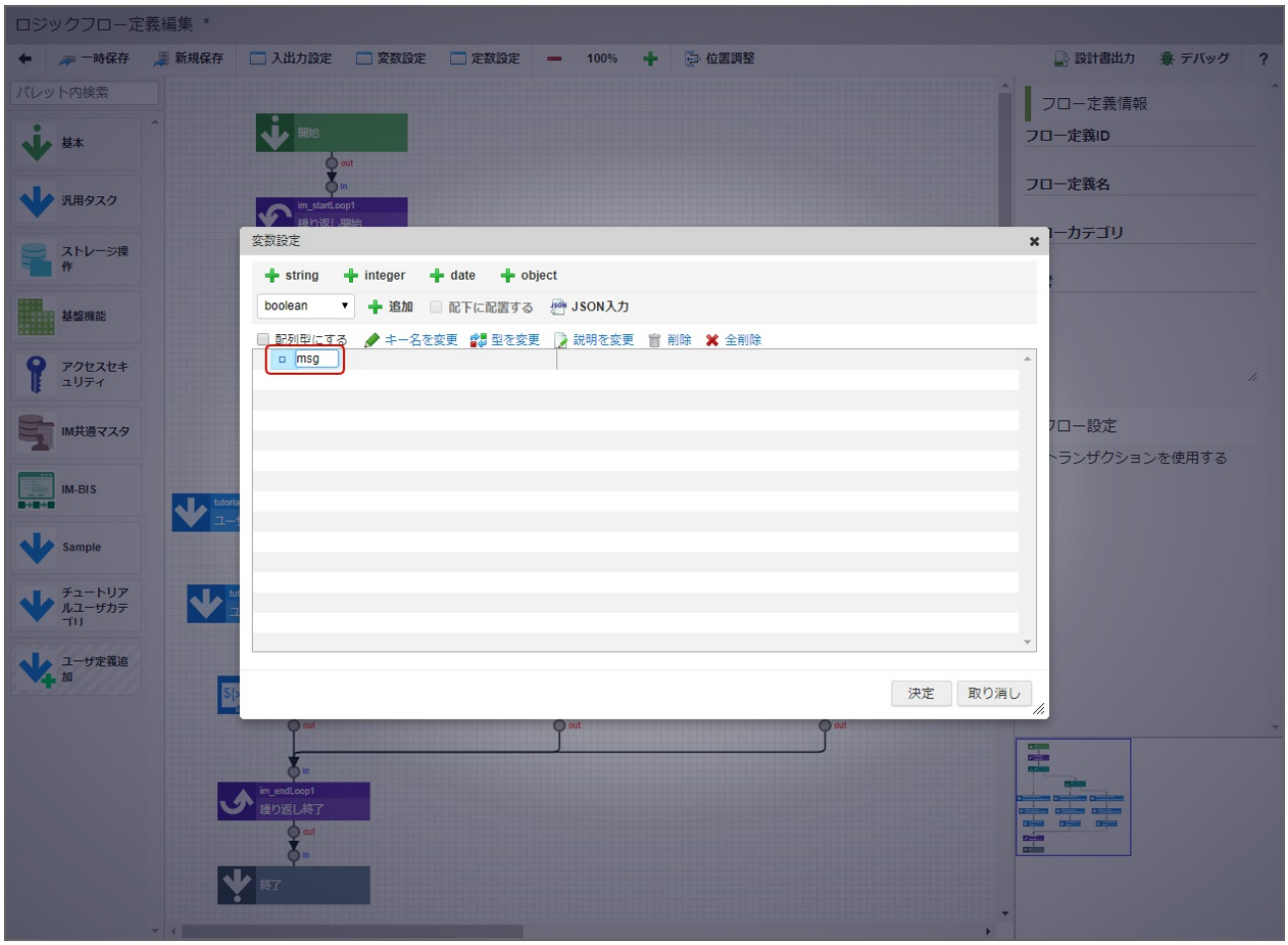
図：「変数設定」をクリック

2. 変数設定画面上部にある「**+string**」をクリックします。



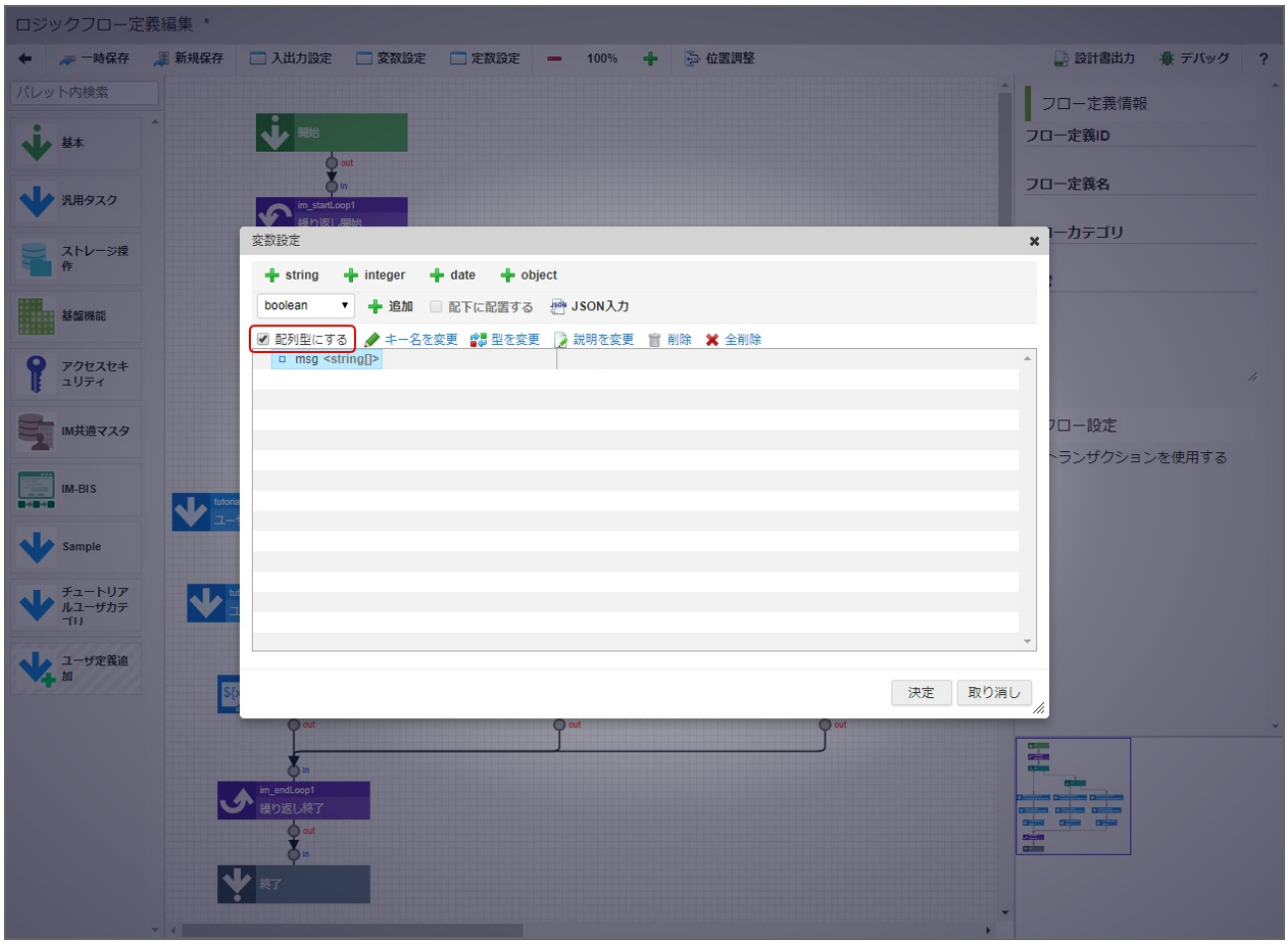

図：変数の追加

3. 画面中央に新しく値が設定されたことを確認し、名称を「 msg 」とします。




図：変数の名称変更

4. 変数はString型の配列とする必要があるため、「配列型にする」のチェックボックスをオンにします。



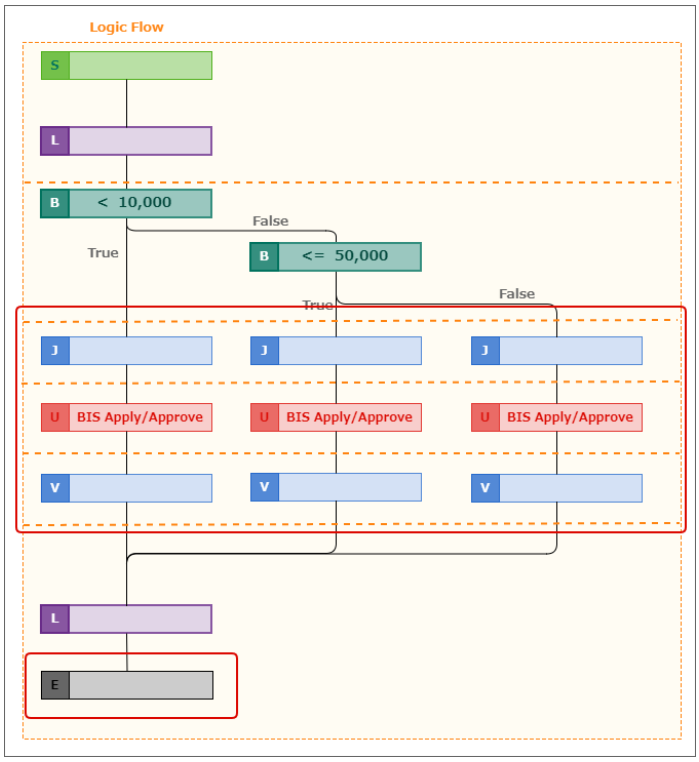

図：変数を配列に設定

5. 変数設定画面下部の「決定」をクリックします。

以上で、変数の定義が完了しました。

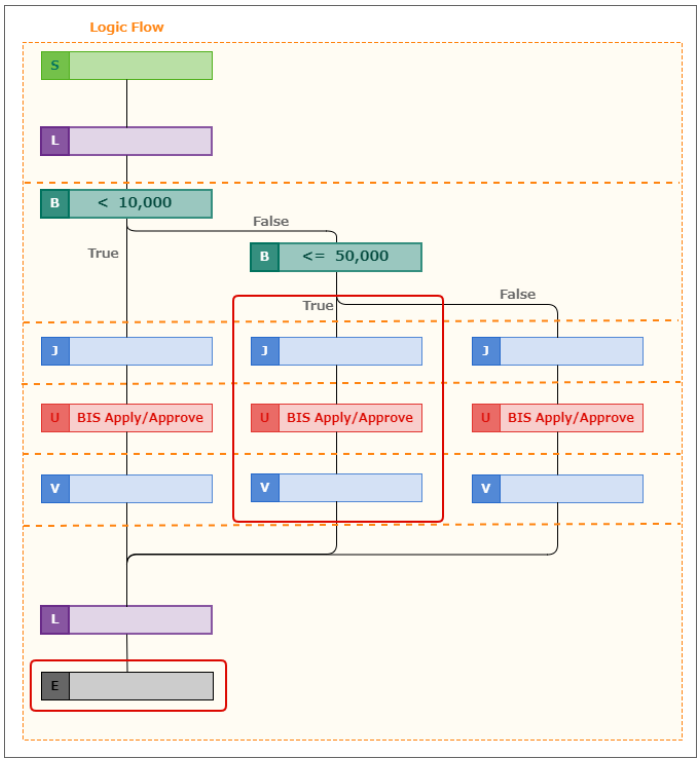
マッピング設定を行う

マッピングは、各エレメントのプロパティ画面からマッピング設定画面を開いて行います。
ここまで作成したロジックフローでマッピング設定が必要なエレメントは以下のとおりです。



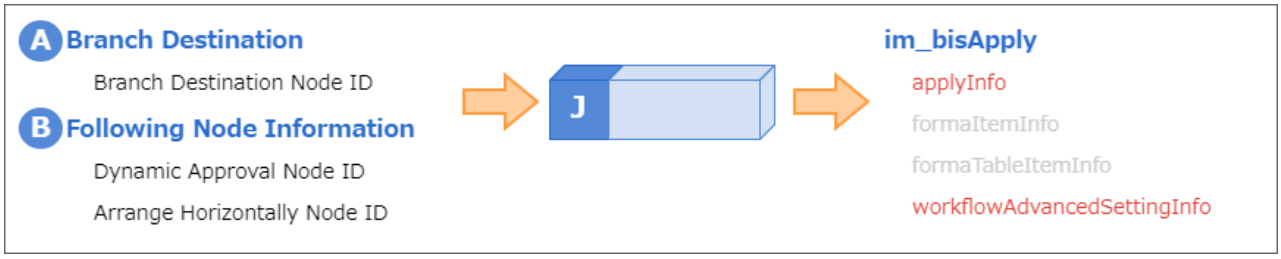

本チュートリアルでは、以下のエレメントへのマッピングの設定を例に説明します。

- 「金額が10000円以上、かつ50000円以下に合致」した場合に実行されるタスク
  - ユーザ定義[BIS申請/承認-申請]ルール2
  - ユーザ定義[BIS申請/承認-申請]
  - 変数操作
- 「終了」制御要素




図： 設定対象のタスク

はじめに、分岐のエレメントに接続しているユーザ定義（JavaScript）のマッピング設定を行います。
このユーザ定義では、後続の処理に関するノード情報を入力値に渡すと、ユーザ定義（BIS申請/承認）の申請情報 applyInfo 、ワークフロー設定情報（高度な設定） workflowAdvancedSettingInfo を返却するように実装しています。




**A. Branch Destination** （分岐先ノード情報）　分岐開始ノードの次の遷移先を指定します。

- Branch Destination Node ID （分岐先ノードID）

**B. Following Node Information** （後続のノード情報）　分岐開始ノードに接続している後続のノードに関する情報です。

- Dynamic Approval Node ID （動的承認ノードID）
- Arrange Horizontally Approval Node ID （横配置ノードID）

分岐ルート別の設定内容は以下のとおりです。

- **ユーザ定義[BIS申請/承認-申請]ルール1** （ユーザ定義（JavaScript））

  「分岐終了ノード」に遷移するルートです。

| 入力（始点） | 出力（終点） |
|---|---|
| 定数<object> - branchEndNodeId<string> | tutorial_ud_bis_apply_rule_11<object> - branchEndId<string> |
| 定数<object> - branchStartNodeId<string> | tutorial_ud_bis_apply_rule_11<object> - branchStartId<string> |
| 定数<object> - dynamicApprovalNodeId<string> | tutorial_ud_bis_apply_rule_11<object> - dynamicApprovalNodeId<string> |
| 定数<object> - horizontalNodeId<string> | tutorial_ud_bis_apply_rule_11<object> - horizontalNodeId<string> |

- **ユーザ定義[BIS申請/承認-申請]ルール2** （ユーザ定義（JavaScript））

  「動的承認ノード」に遷移するルートです。

| 入力（始点） | 出力（終点） |
|---|---|
| 定数<object> - branchStartNodeId<string> | tutorial_ud_bis_apply_rule_21<object> - branchStartId<string> |
| 定数<object> - dynamicApprovalNodeId<string> | tutorial_ud_bis_apply_rule_21<object> - dynamicApprovalNodeId<string> |
| 定数<object> - horizontalNodeId<string> | tutorial_ud_bis_apply_rule_21<object> - horizontalNodeId<string> |

- **ユーザ定義[BIS申請/承認-申請]ルール3** （ユーザ定義（JavaScript））

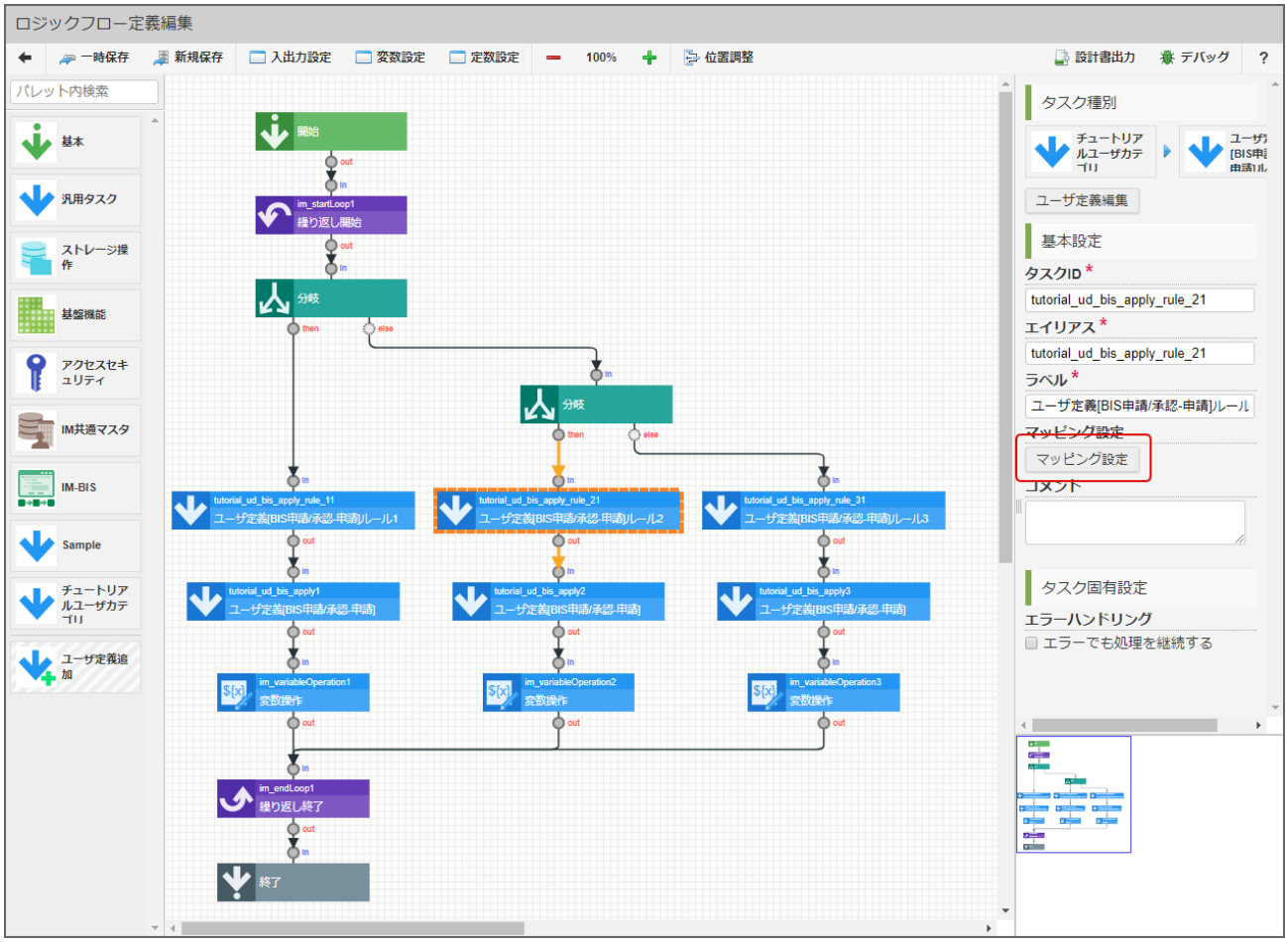
  「横配置ノード」に遷移するルートです。

| 入力（始点） | 出力（終点） |
|---|---|
| 定数<object> - branchStartNodeId<string> | tutorial_ud_bis_apply_rule_31<object> - branchStartId<string> |
| 定数<object> - dynamicApprovalNodeId<string> | tutorial_ud_bis_apply_rule_31<object> - dynamicApprovalNodeId<string> |
| 定数<object> - horizontalNodeId<string> | tutorial_ud_bis_apply_rule_31<object> - horizontalNodeId<string> |

設定内容をもとに、マッピング設定画面で実際に設定を行います。

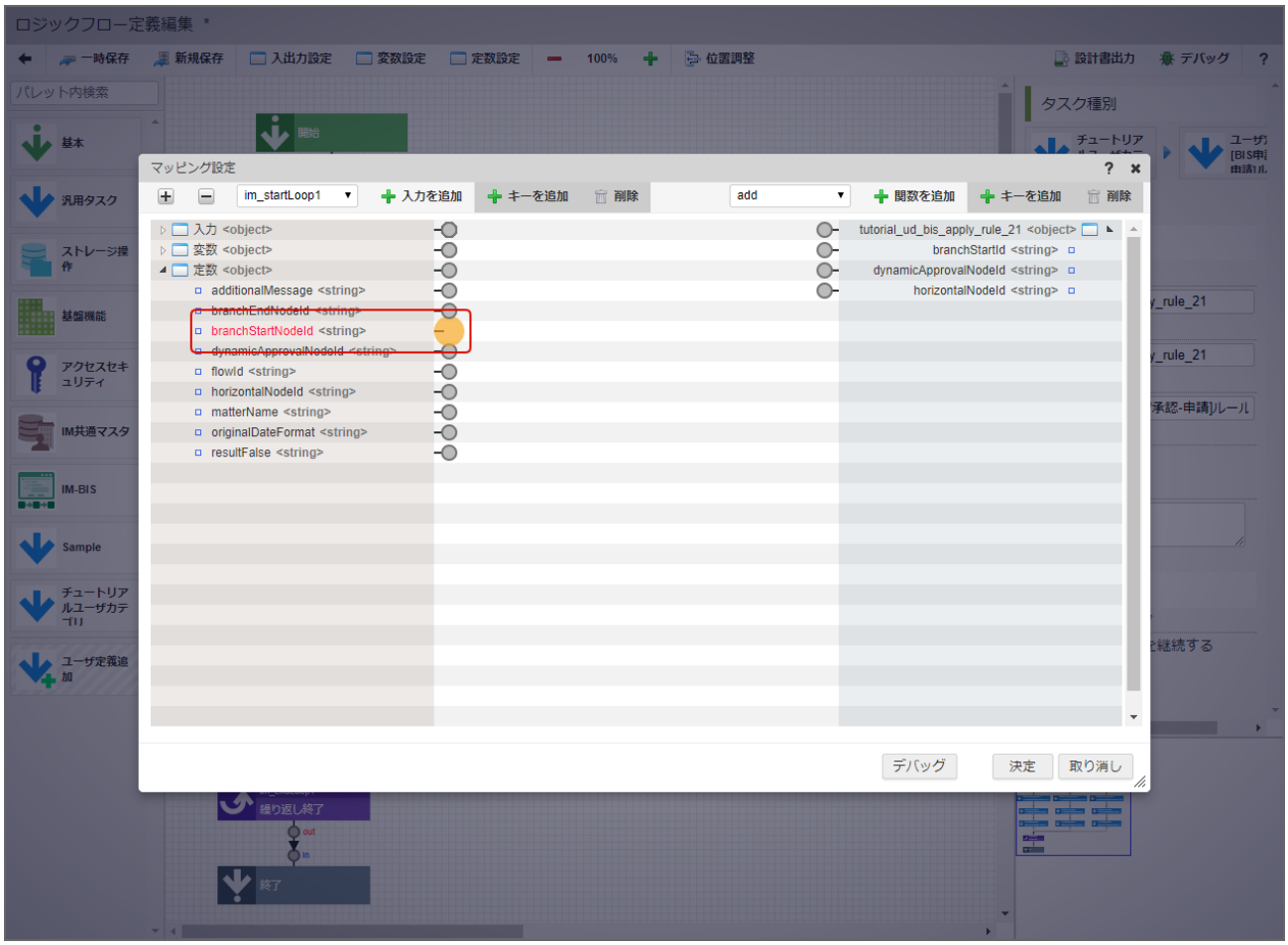
要素同士を直接接続する方法については、 *マッピング設定を行う* を参照してください。

1. 2番目の「分岐」制御要素の then のルートに接続している「ユーザ定義[BIS申請/承認-申請]ルール2」タスクのマッピング設定画面を開きます。



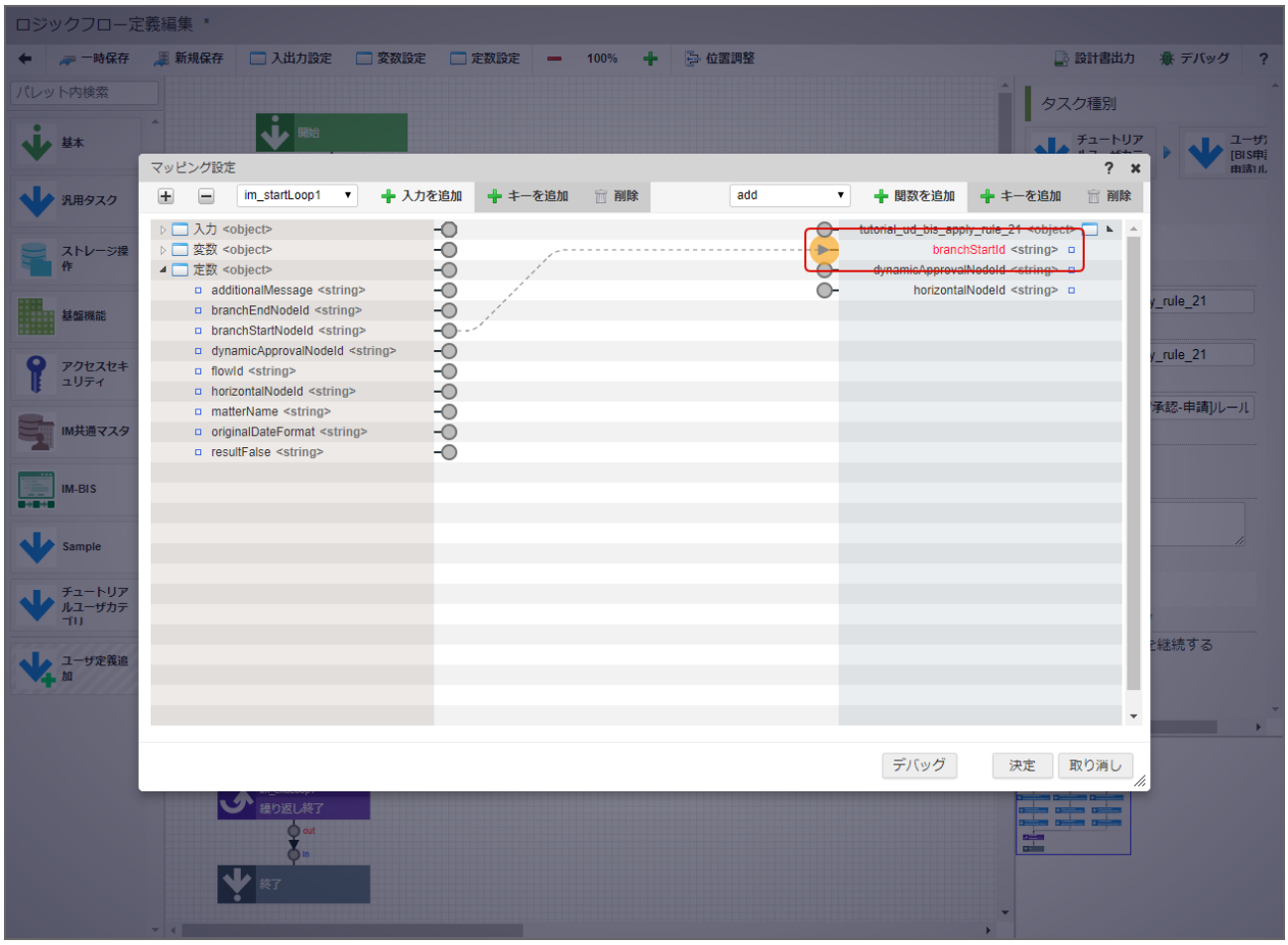

図：「ユーザ定義[BIS申請/承認-申請]ルール2」をクリック

2. 設定画面左部、「定数<object>」要素の下にある「 branchStartNodeId<string> 」から出ている端子にカーソルをあわせます。



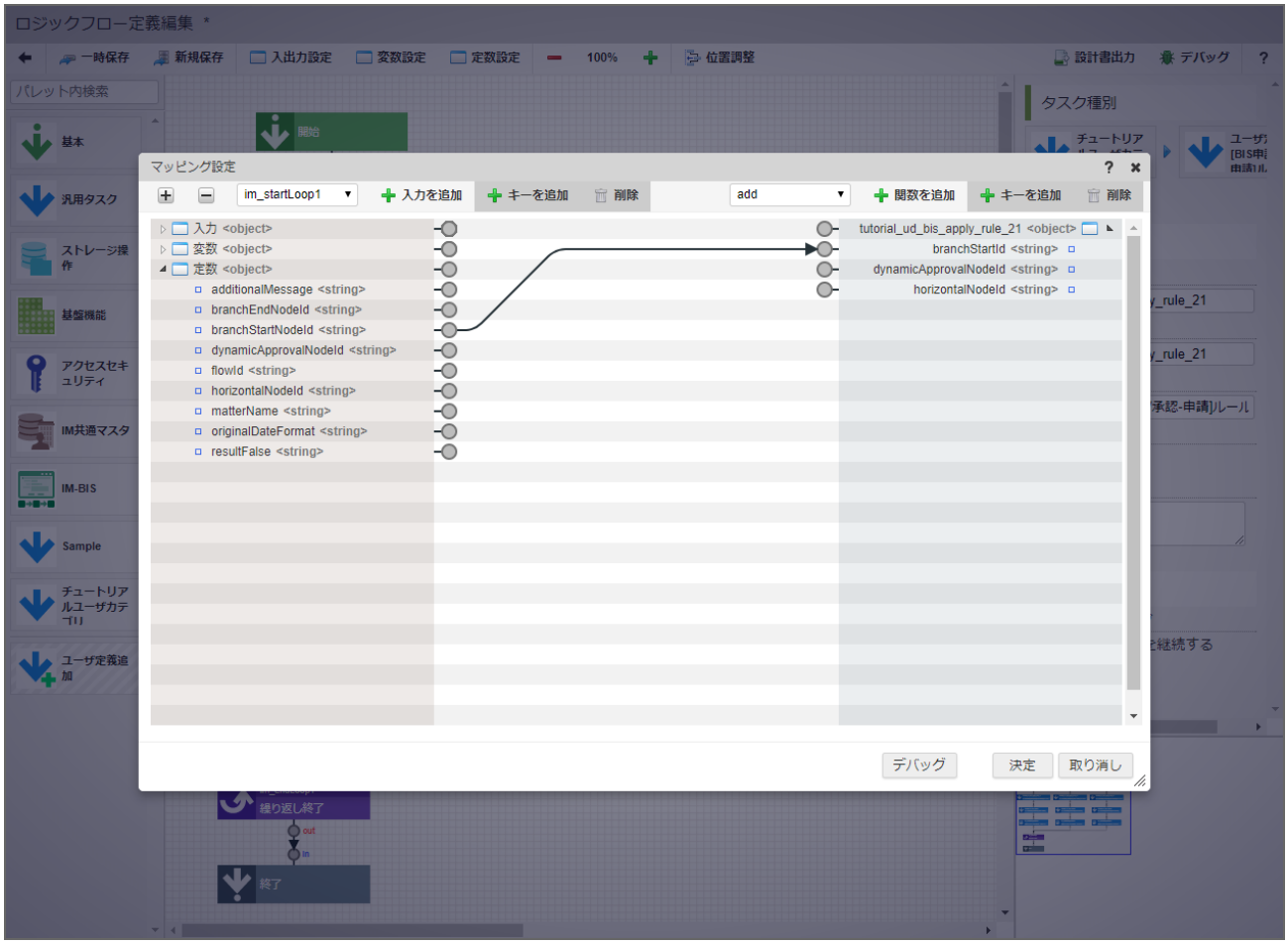

図： branchStartNodeId<string> にカーソルをあわせる

3. そのままドラッグし、設定画面右部、「tutorial_ud_bis_apply_rule_21<object>」要素の下にある「 branchStartId<string> 」から出ている端子にドロップします。



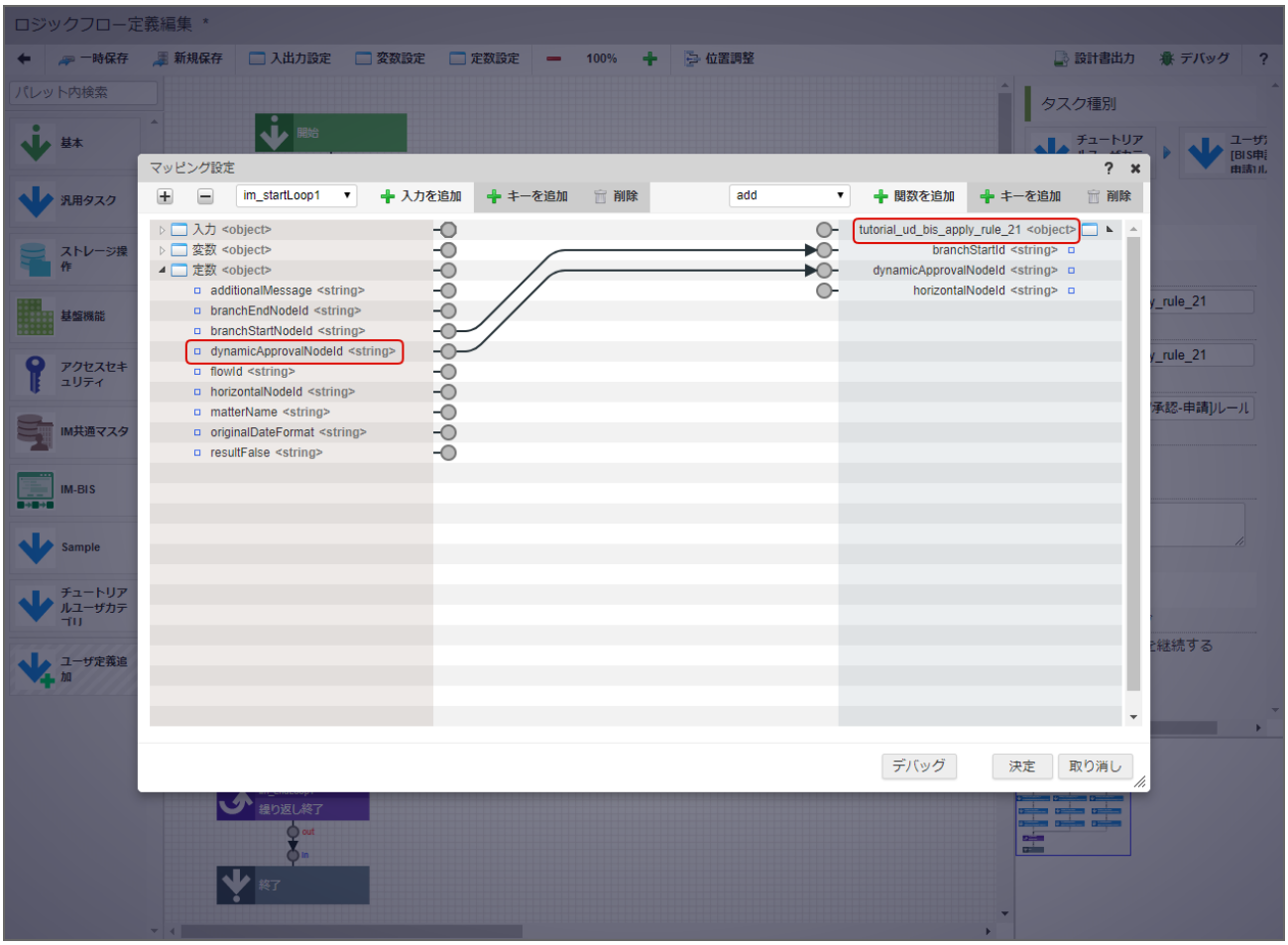
図： branchStartId<string> でドロップ

4. これにより、分岐開始ノードの情報がマッピングされました。



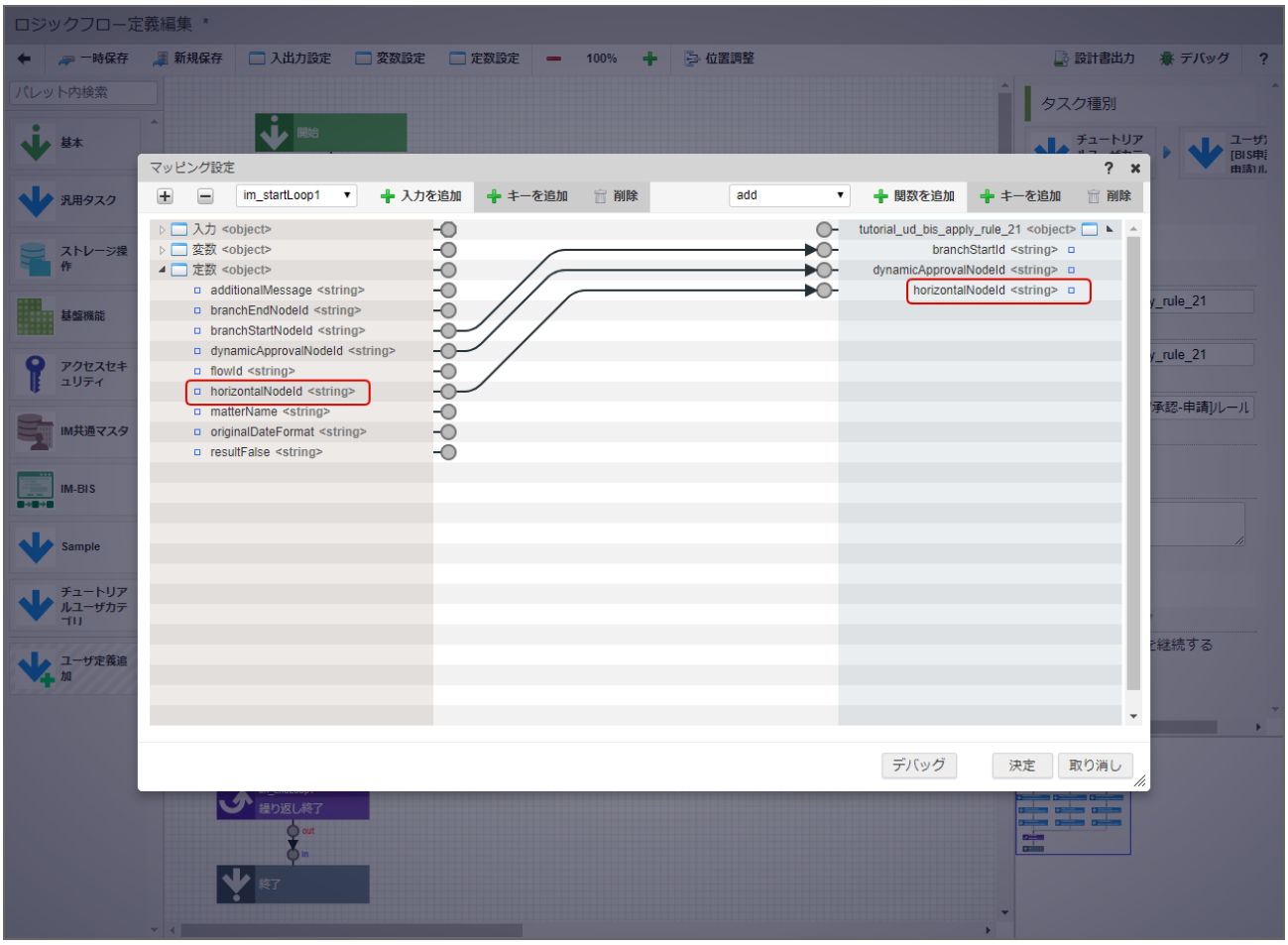

図：分岐開始ノードのマッピング情報の作成

5. 同様に、動的承認ノード・横配置ノードの情報をマッピングします。
   - 動的承認ノードの情報のマッピング




図：動的承認ノードのマッピング情報の作成

- 横配置ノードの情報のマッピング

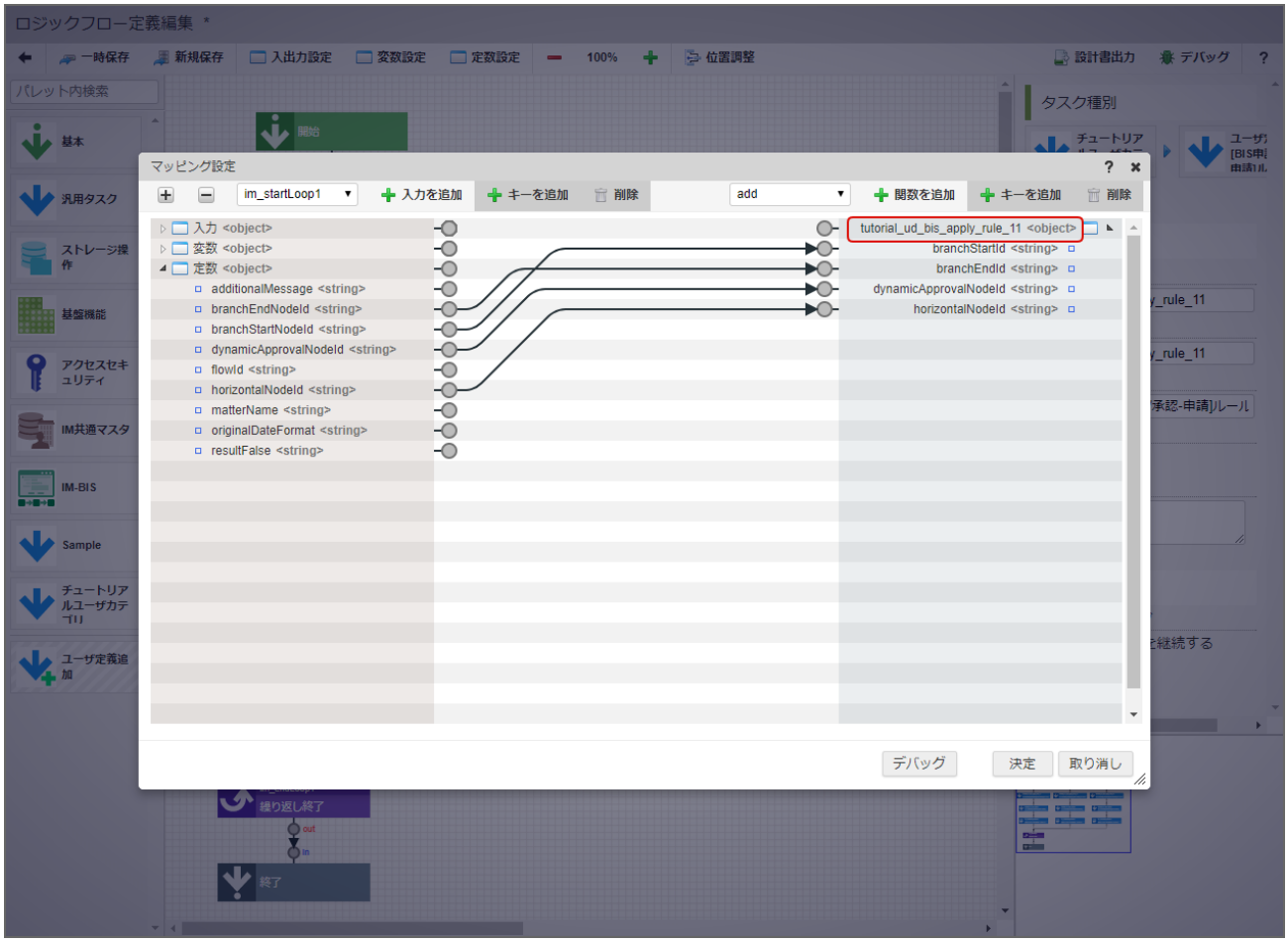



図：横配置ノードのマッピング情報の作成

6. これで、「ユーザ定義[BIS申請/承認-申請]ルール2」に対するマッピングが設定できました。

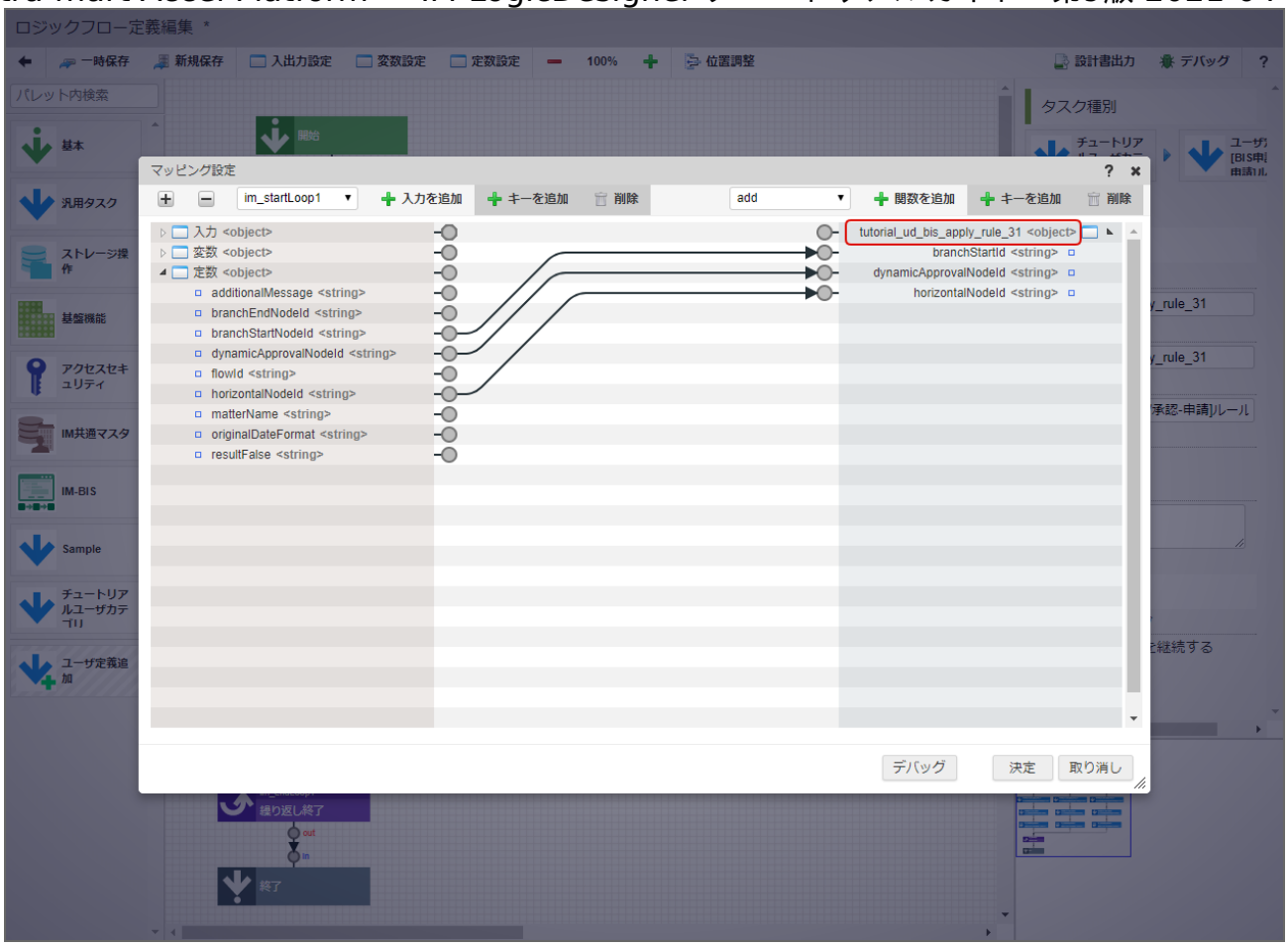
「ユーザ定義[BIS申請/承認-申請]ルール1」、「ユーザ定義[BIS申請/承認-申請]ルール3」についても、同様に「定数<object>」要素の下から対応する項目同士をマッピングします。

- ユーザ定義[BIS申請/承認-申請]ルール1のマッピング
  このルートのみ遷移先が分岐終了ノードのため、分岐終了ノードのマッピングが必要です。




図：ユーザ定義[BIS申請/承認-申請]ルール1のマッピング情報の作成

- ユーザ定義[BIS申請/承認-申請]ルール3のマッピング




図：ユーザ定義[BIS申請/承認-申請]ルール3のマッピング情報の作成

続いて、「ユーザ定義[BIS申請/承認-申請]」タスクのマッピングを行います。
設定内容は以下のとおりです。

- **ユーザ定義[BIS申請/承認-申請]**

  「ユーザ定義[BIS申請/承認-申請]」については、基本的に同じマッピングの設定を行います。
  金額によって分岐先や次の処理対象者が異なるため、該当の処理に関連する部分の接続が異なる点に注意してください。

  - 金額が10000円未満に合致するルートの場合

| 入力（始点） | 出力（終点） |
|---|---|
| 入力<object> - applyInfo<object> - imwAuthUserCode<string> | tutorial_ud_bis_apply1<object> - applyInfo<object> - applyAuthUserCode<string> |
| 入力<object> - applyInfo<object> - imwApplyBaseDate<string><br>定数<object> - originalDateFormat<string><br>（ parse 関数を利用） | tutorial_ud_bis_apply1<object> - applyInfo<object> - applyBaseDate<date> |
| 入力<object> - applyInfo<object> - authUserOrgzInfo<object> | tutorial_ud_bis_apply1<object> - applyInfo<object> - authUserOrgzInfo<object> |
| 定数<object> - flowId<string> | tutorial_ud_bis_apply1<object> - applyInfo<object> - flowId<string> |
| 定数<object> - matterName<string> | tutorial_ud_bis_apply1<object> - applyInfo<object> - matterName<string> |
| tutorial_ud_bis_apply_rule_11<object> | tutorial_ud_bis_apply1<object> - workflowAdvancedSettingInfo<object> |
| im_startLoop1<object> - item<object> | tutorial_ud_bis_apply1<object> - formaItemInfo<object> |

  - 金額が10000円以上かつ50000円以下に合致するルートの場合

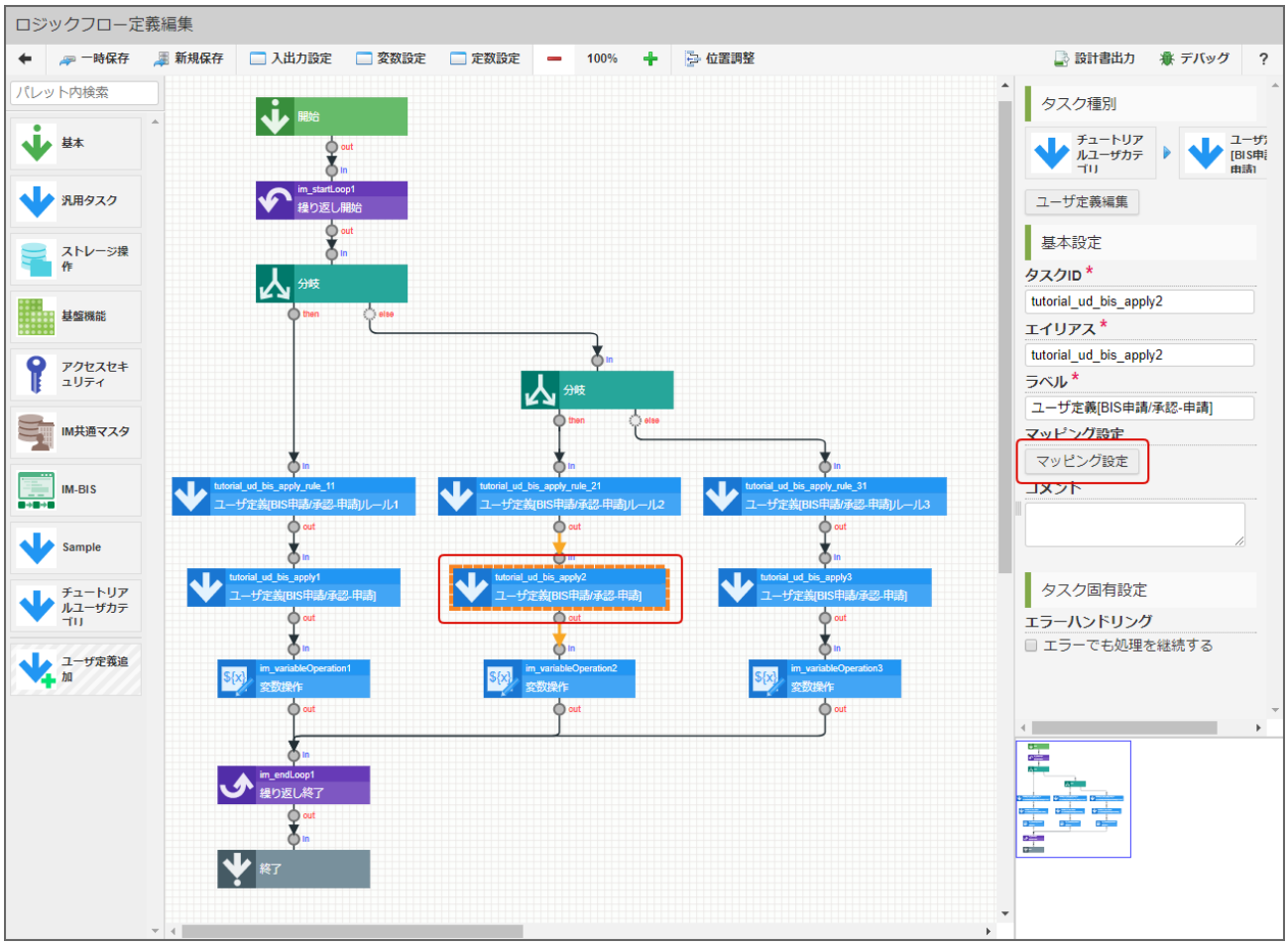
| 入力（始点） | 出力（終点） |
| --- | --- |
| 入力<object> - applyInfo<object> - imwAuthUserCode<string> | tutorial_ud_bis_apply2<object> - applyInfo<object> - applyAuthUserCode<string> |
| 入力<object> - applyInfo<object> - imwApplyBaseDate<string><br>定数<object> - originalDateFormat<string><br>（ parse 関数を利用） | tutorial_ud_bis_apply2<object> - applyInfo<object> - applyBaseDate<date> |
| 入力<object> - applyInfo<object> - authUserOrgzInfo<object> | tutorial_ud_bis_apply2<object> - applyInfo<object> - authUserOrgzInfo<object> |
| 定数<object> - flowId<string> | tutorial_ud_bis_apply2<object> - applyInfo<object> - flowId<string> |
| 定数<object> - matterName<string> | tutorial_ud_bis_apply2<object> - applyInfo<object> - matterName<string> |
| tutorial_ud_bis_apply_rule_21<object> | tutorial_ud_bis_apply2<object> - workflowAdvancedSettingInfo<object> |
| im_startLoop1<object> - item<object> | tutorial_ud_bis_apply2<object> - formaItemInfo<object> |

- 金額が50000円超に合致するルートの場合

| 入力（始点） | 出力（終点） |
| --- | --- |
| 入力<object> - applyInfo<object> - imwAuthUserCode<string> | tutorial_ud_bis_apply3<object> - applyInfo<object> - applyAuthUserCode<string> |
| 入力<object> - applyInfo<object> - imwApplyBaseDate<string><br>定数<object> - originalDateFormat<string><br>（ parse 関数を利用） | tutorial_ud_bis_apply3<object> - applyInfo<object> - applyBaseDate<date> |
| 入力<object> - applyInfo<object> - authUserOrgzInfo<object> | tutorial_ud_bis_apply3<object> - applyInfo<object> - authUserOrgzInfo<object> |
| 定数<object> - flowId<string> | tutorial_ud_bis_apply3<object> - applyInfo<object> - flowId<string> |
| 定数<object> - matterName<string> | tutorial_ud_bis_apply3<object> - applyInfo<object> - matterName<string> |
| tutorial_ud_bis_apply_rule_31<object> | tutorial_ud_bis_apply3<object> - workflowAdvancedSettingInfo<object> |
| im_startLoop1<object> - item<object> | tutorial_ud_bis_apply3<object> - formaItemInfo<object> |

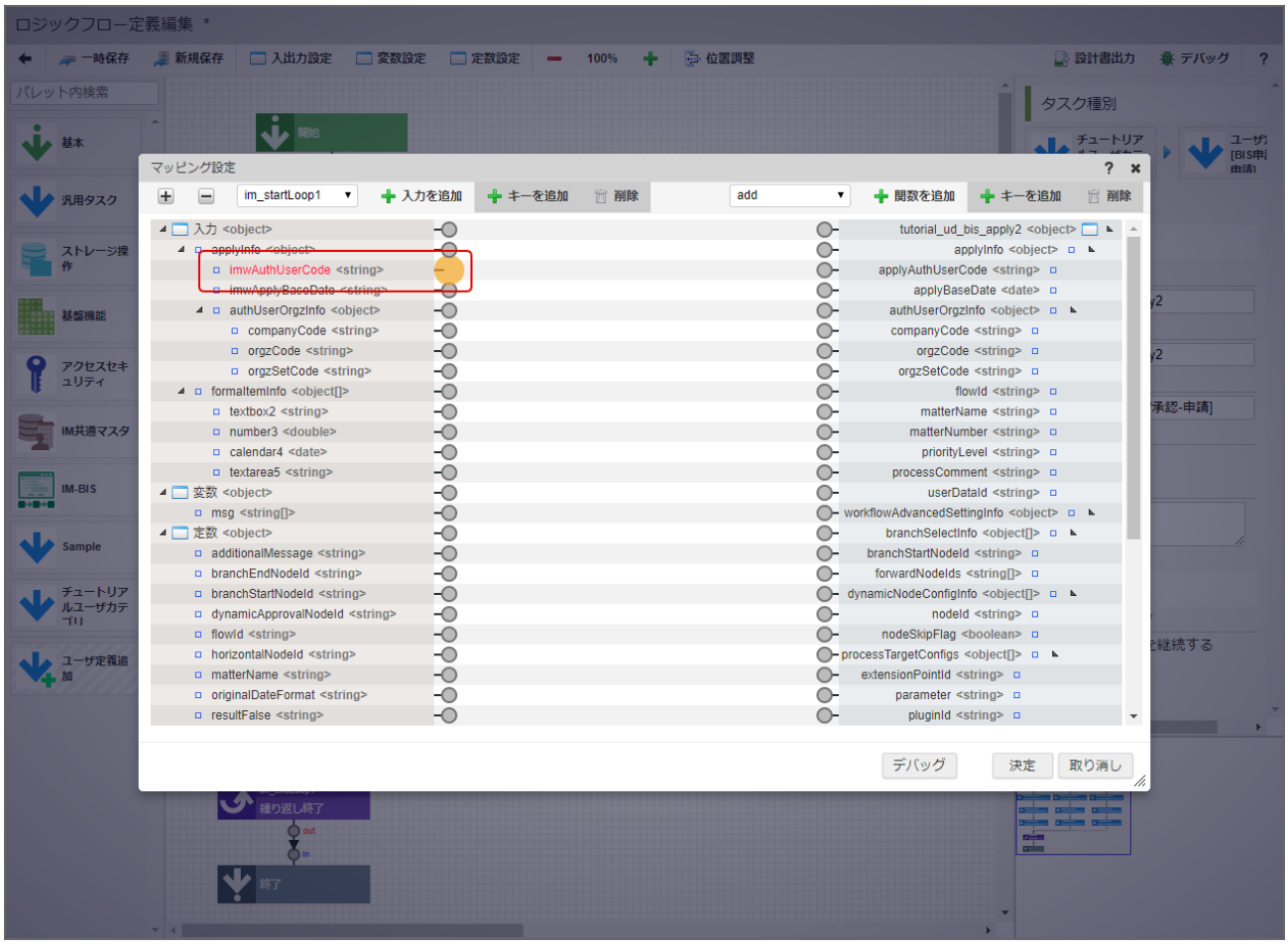
実際に「ユーザ定義[BIS申請/承認-申請]」タスクのマッピングを設定します。

1. 中央のルート上の「ユーザ定義[BIS申請/承認-申請]」タスクのマッピング設定画面を開きます。



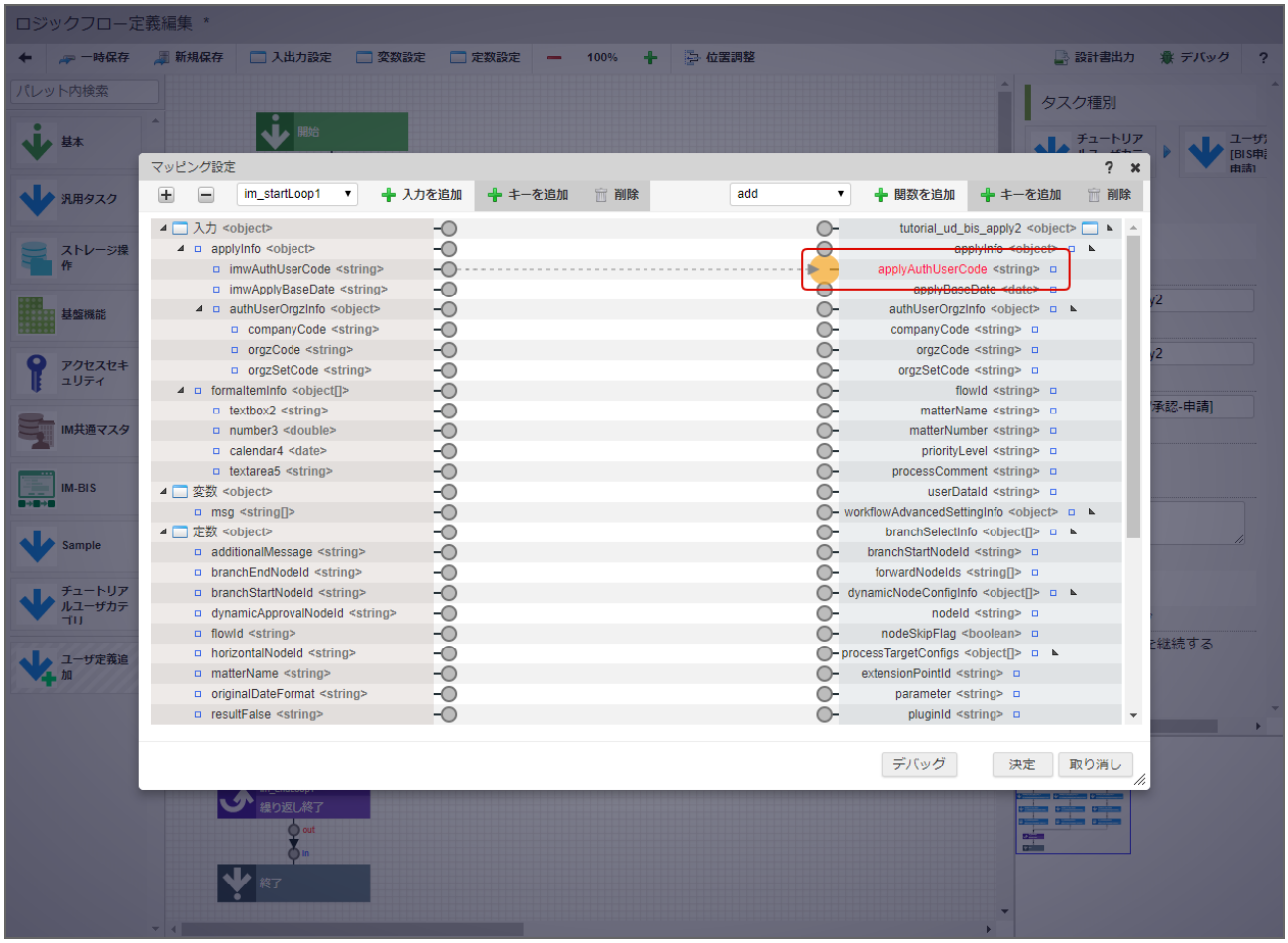

図：「ユーザ定義[BIS申請/承認-申請]」をクリック

2. 設定画面左部、「入力<object>」-「applyInfo<object>」要素の下にある「 imwAuthUserCode<string> 」から出ている端子にカーソルをあわせます。



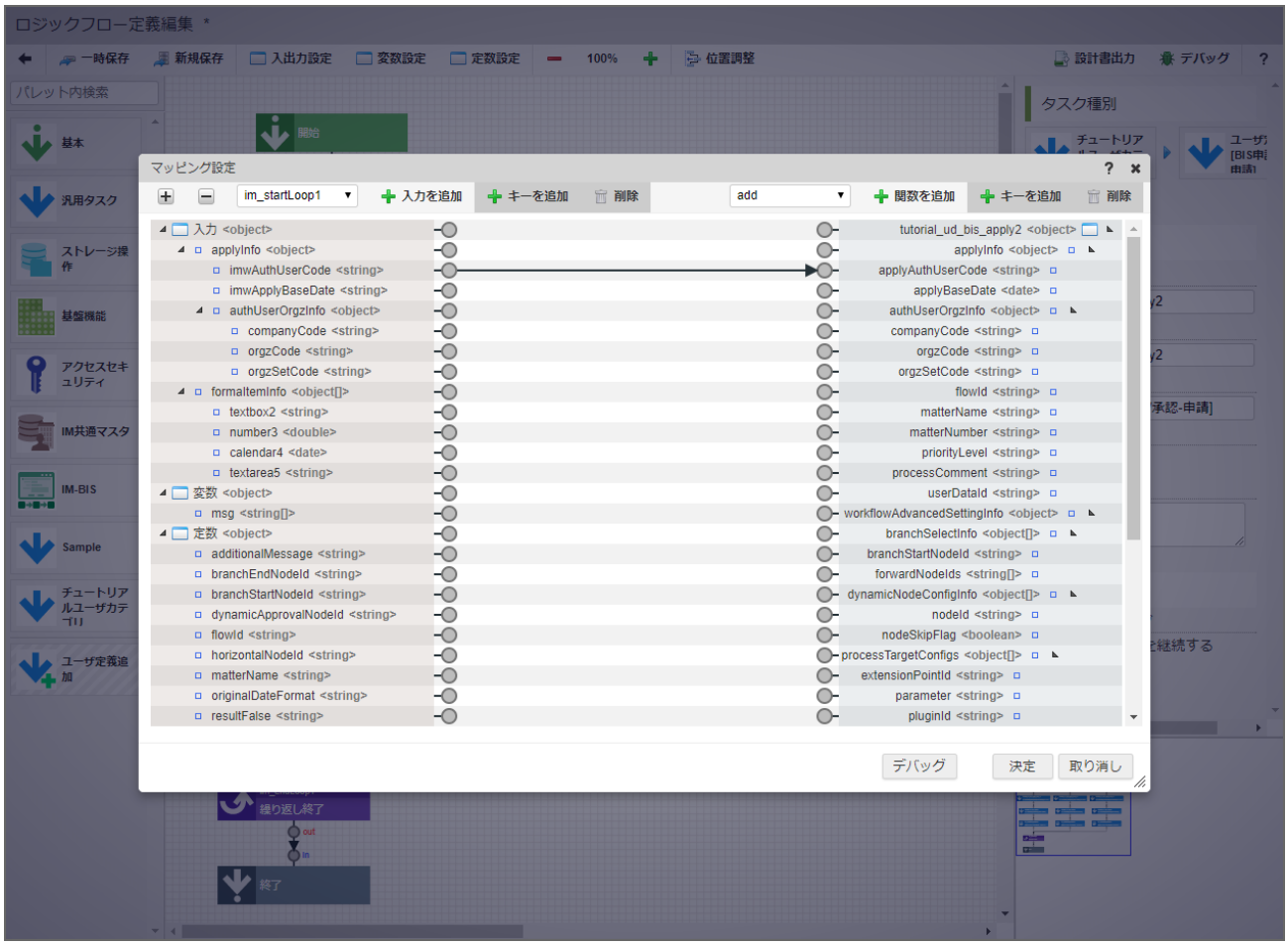
図： imwAuthUserCode<string> にカーソルをあわせる

3. そのままドラッグし、設定画面右部、「tutorial_ud_bis_apply2<object>」-「applyInfo<object>」要素の下にある「applyAuthUserCode<string>」から出ている端子にドロップします。



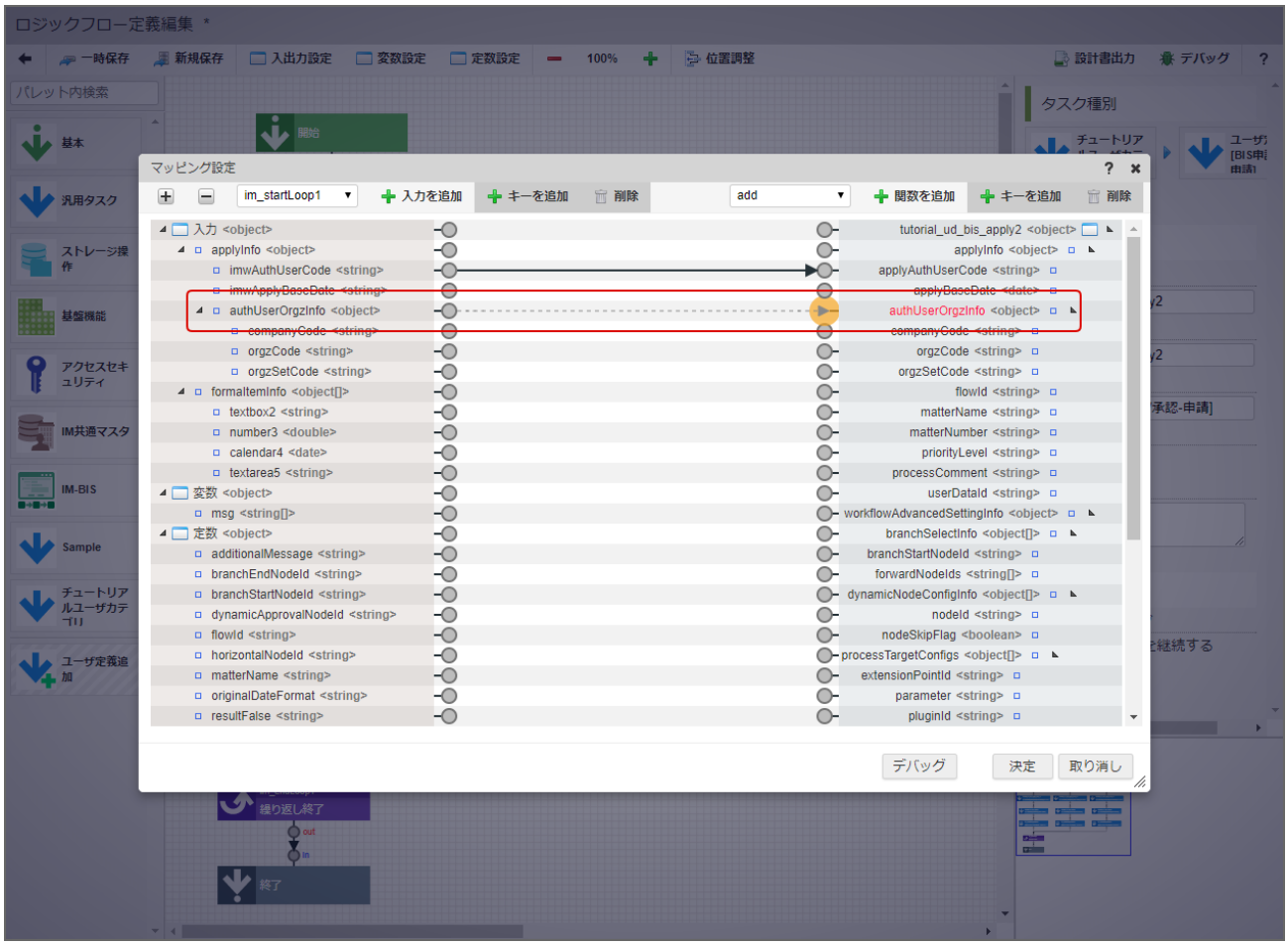

図： applyAuthUserCode<string> でドロップ

4. これにより、フロー実行時の申請者情報と、 BIS申請/承認の申請権限者を表す情報がマッピングされました。



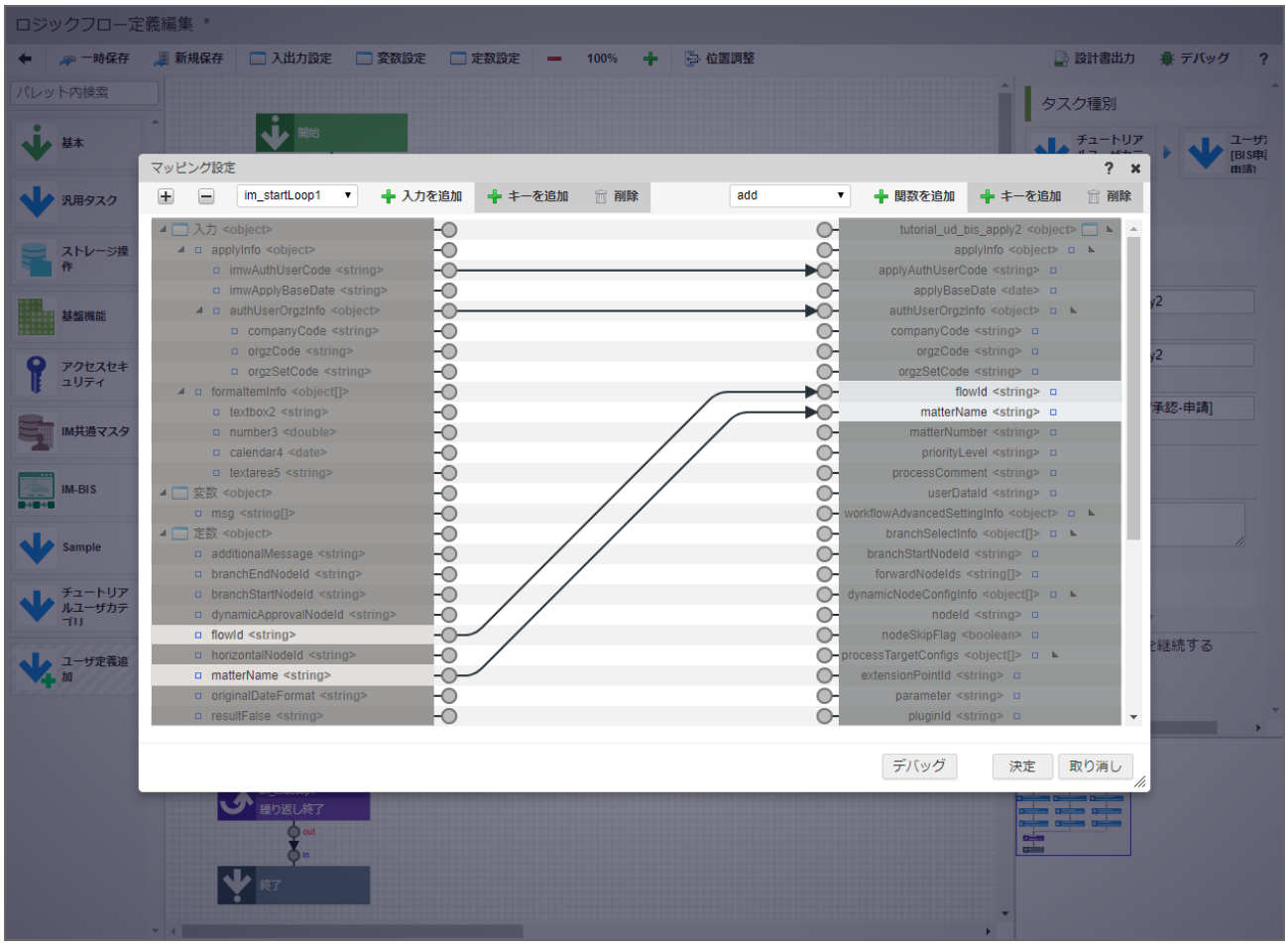

図：マッピング情報の作成

5. 「入力<object>」-「applyInfo<object>」要素の下にある「 authUserOrgzInfo<object> 」と「tutorial_ud_bis_apply2<object>」-「applyInfo<object>」要素の下にある「 authUserOrgzInfo<object> 」（権限者所属組織情報）をマッピングします。




図：権限者所属組織情報のマッピング

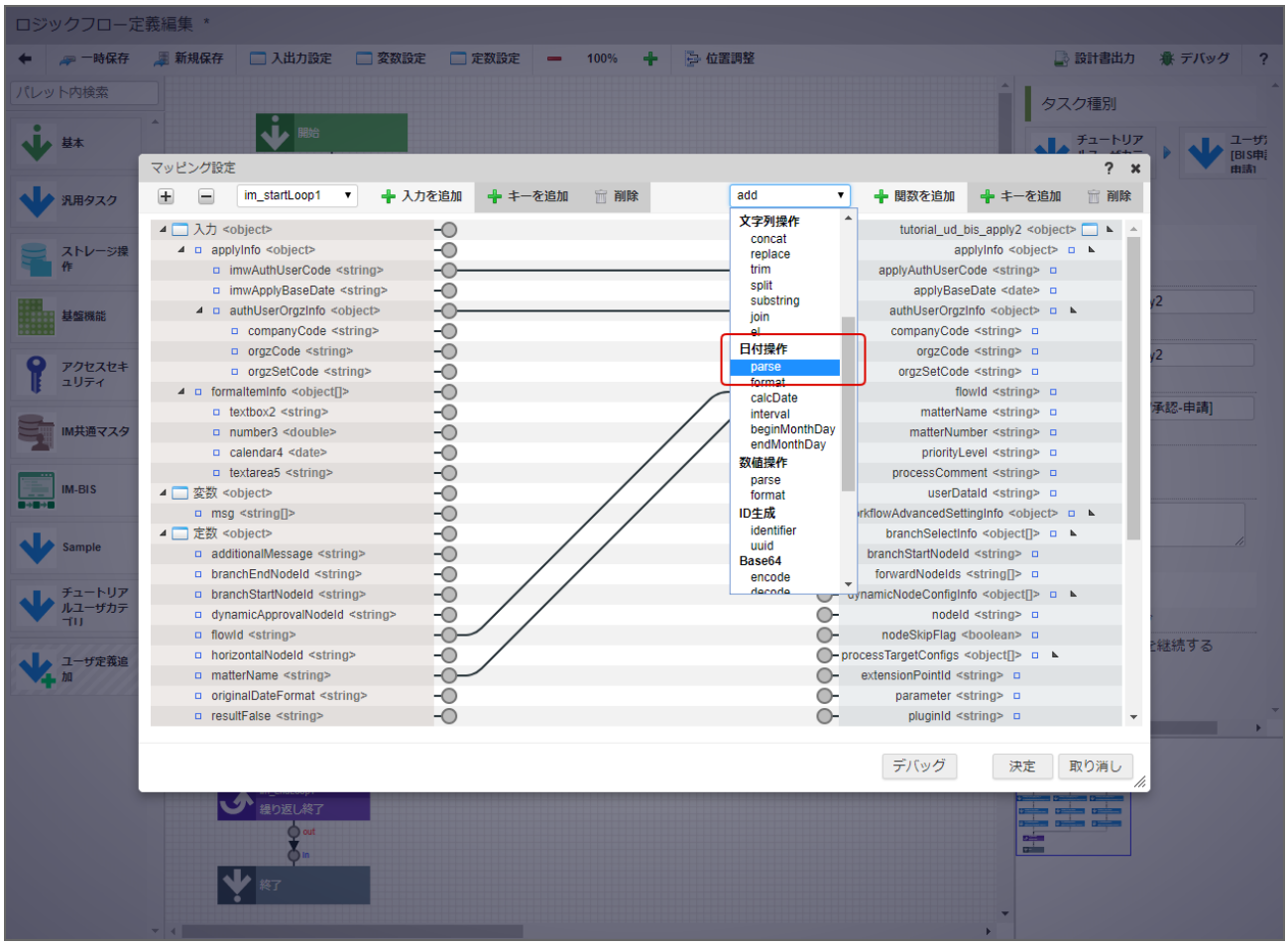
6. 同様の手順で、設定情報に基づいた以下の要素をマッピングします。
   - 定数<object> - flowId<string>
     - tutorial_ud_bis_apply2<object> - applyInfo<object> -flowId<string>
   - 定数<object> - matterName<string>
     - tutorial_ud_bis_apply2<object> - applyInfo<object> -matterName<string>




図：定数と実行対象のフロー情報のマッピング

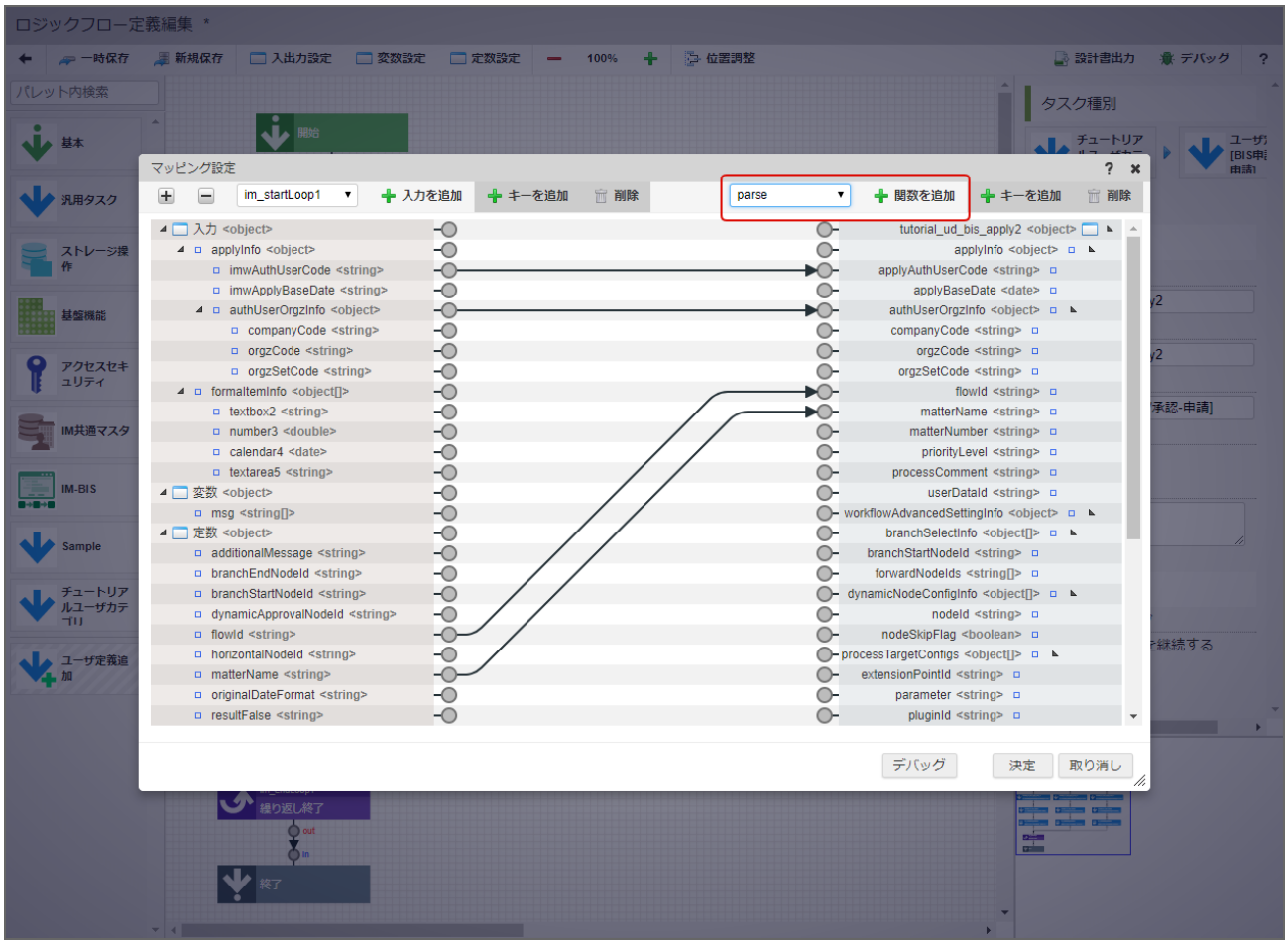
続いて、関数による日付フォーマットの変換を行いながら「ユーザ定義[BIS申請/承認-申請]」タスクの申請基準日の設定方法を説明します。

1. マッピング設定画面上部、ヘッダ内の中央右寄りに位置するセレクトボックスをクリックし、以下の項目を選択します。
   - 日付操作 - `parse`



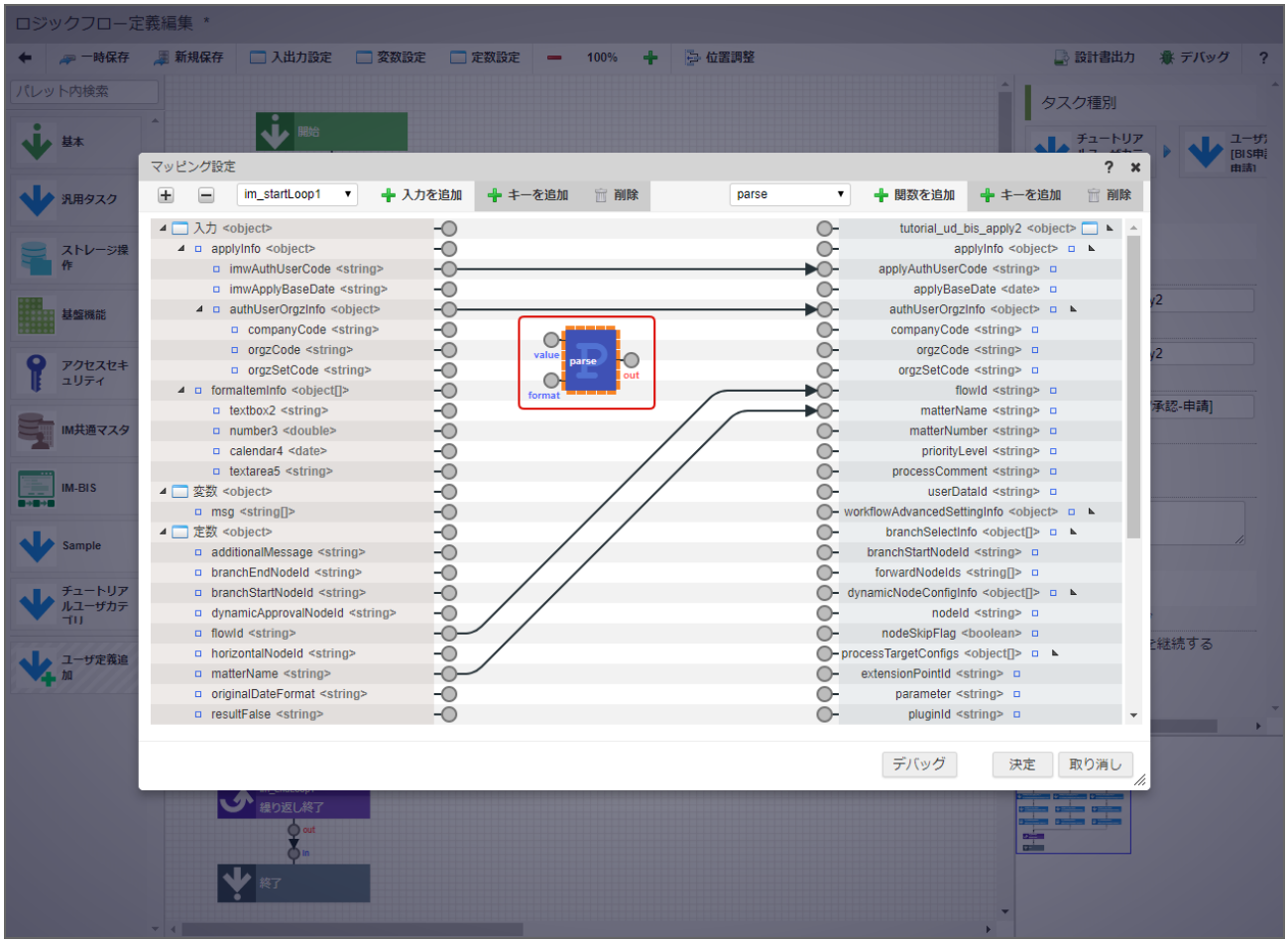

図：マッピング関数「parse」の選択

2. セレクトボックスの中身が変更されたことを確認し、右側にある「関数を追加」をクリックします。



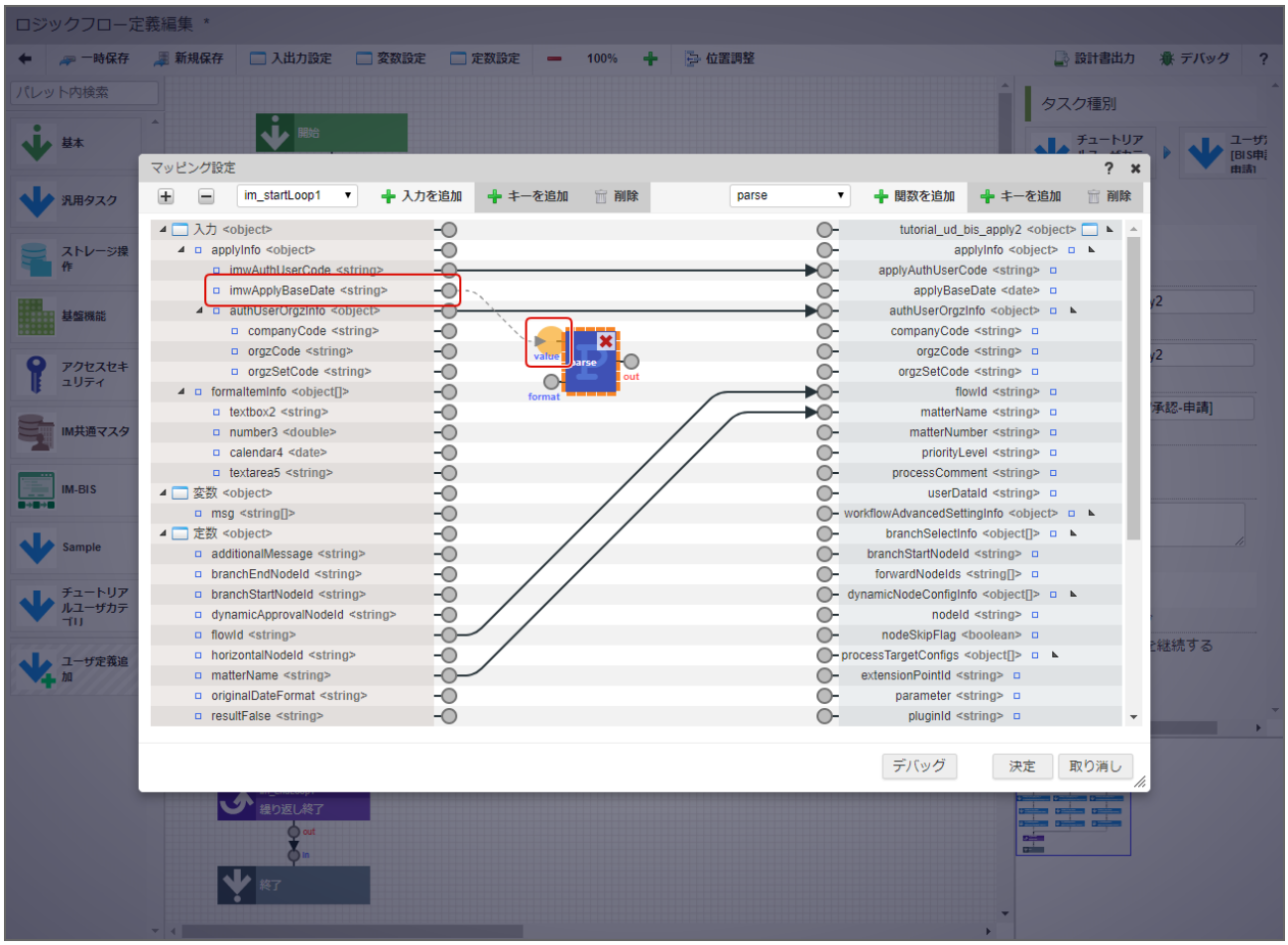

図：「関数を追加」をクリック

3. マッピング関数として、「parse」関数が追加されました。



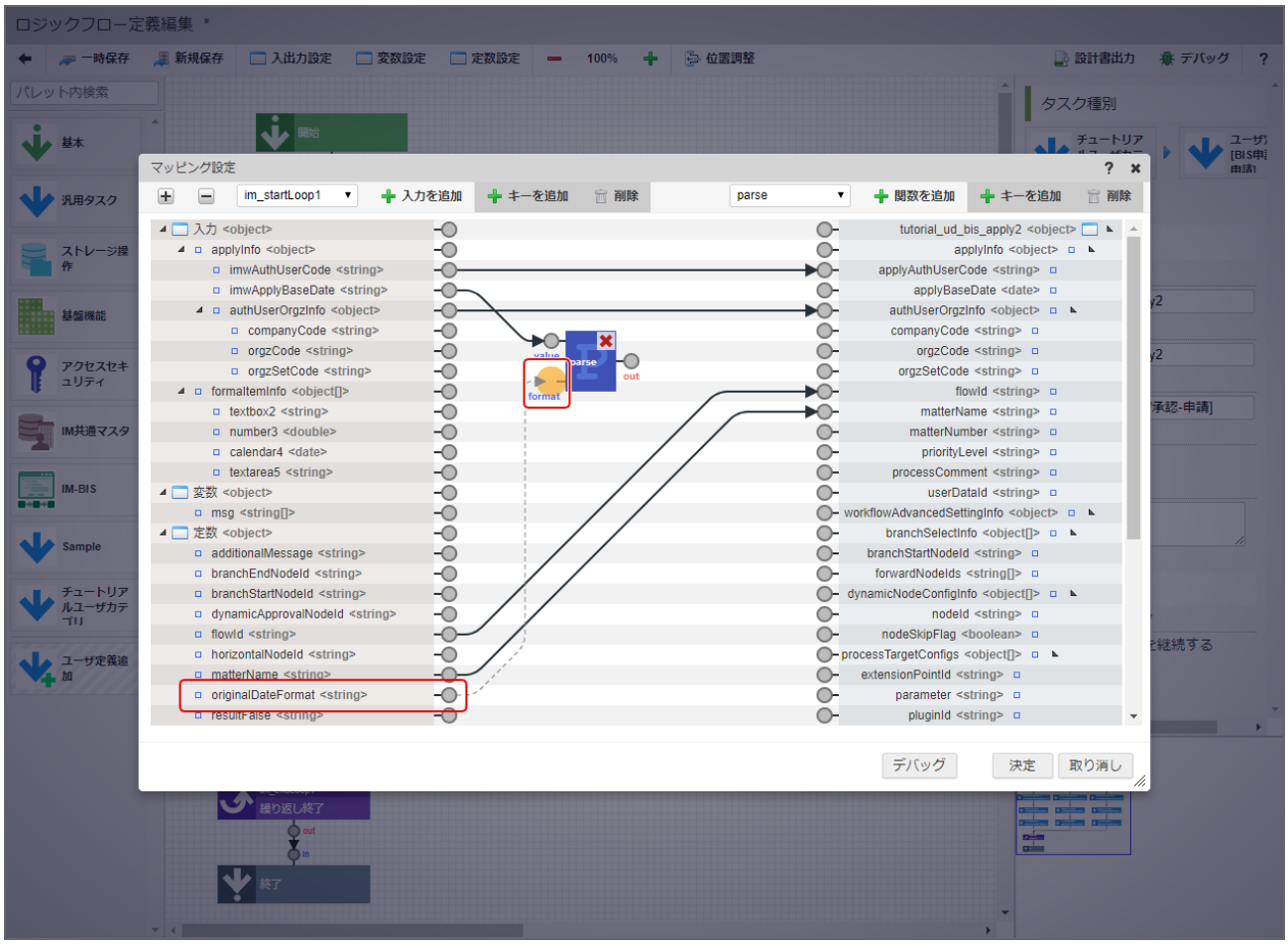

図：関数（parse）の追加

4. 設定画面左部、「入力<object>」-「applyInfo<object>」要素の下にある「 imwApplyBaseDate<string> 」から出ている端子をドラッグし、「parse」関数の左部から出ている端子のうち「 value 」へドロップします。



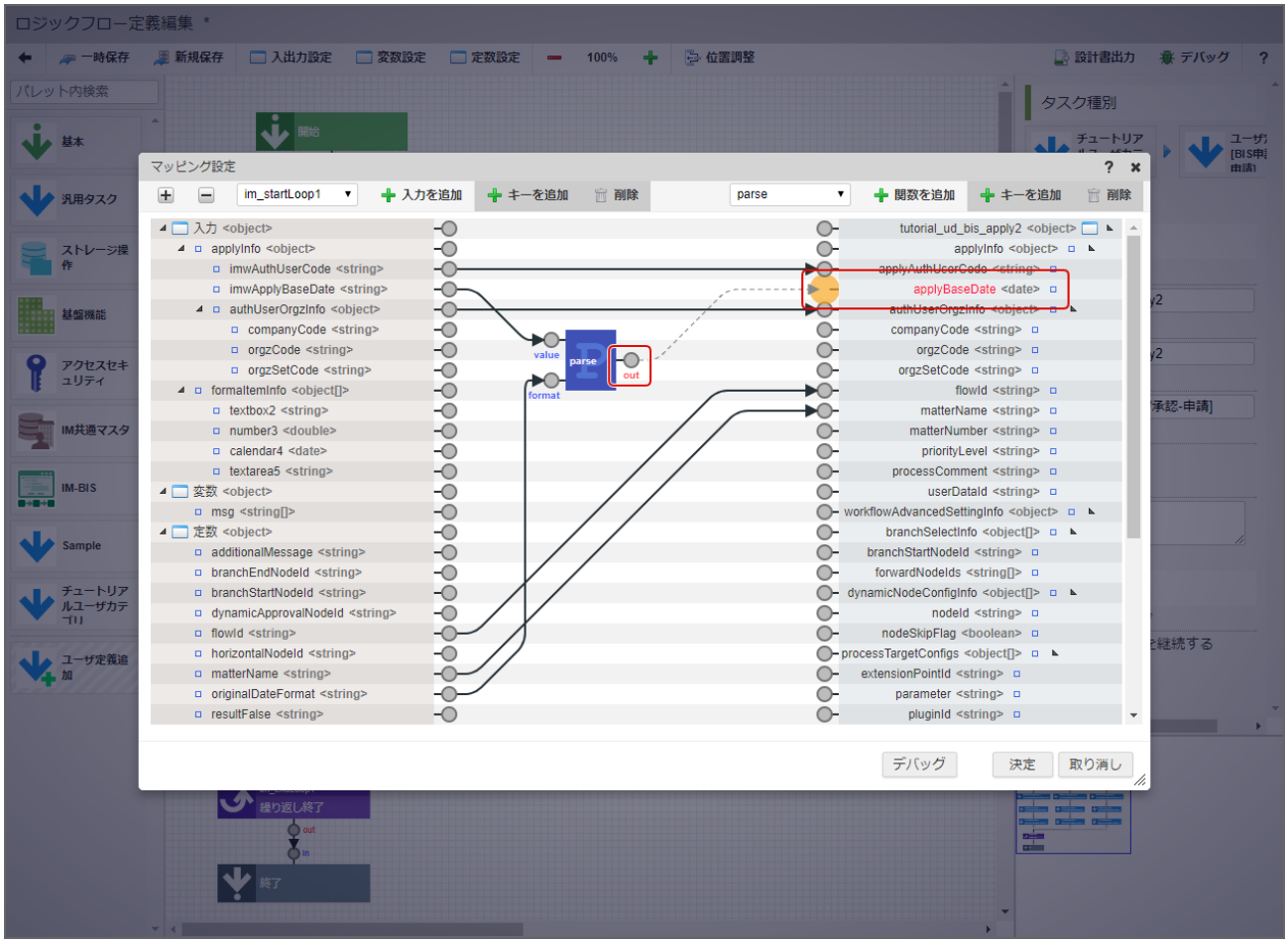

図：関数と入力値の接続

5. 同様に、「定数<object>」要素の下にある「 originalDateFormat<string> 」から出ている端子をドラッグし、「parse」関数の左部から出ている端子のうち「 format 」へドロップします。



図：関数とフォーマットの接続

6. 「parse」関数の右部から出ている端子「 out 」をドラッグし、設定画面右部、「applyBaseDate<date>」から出ている端子にドロップします。

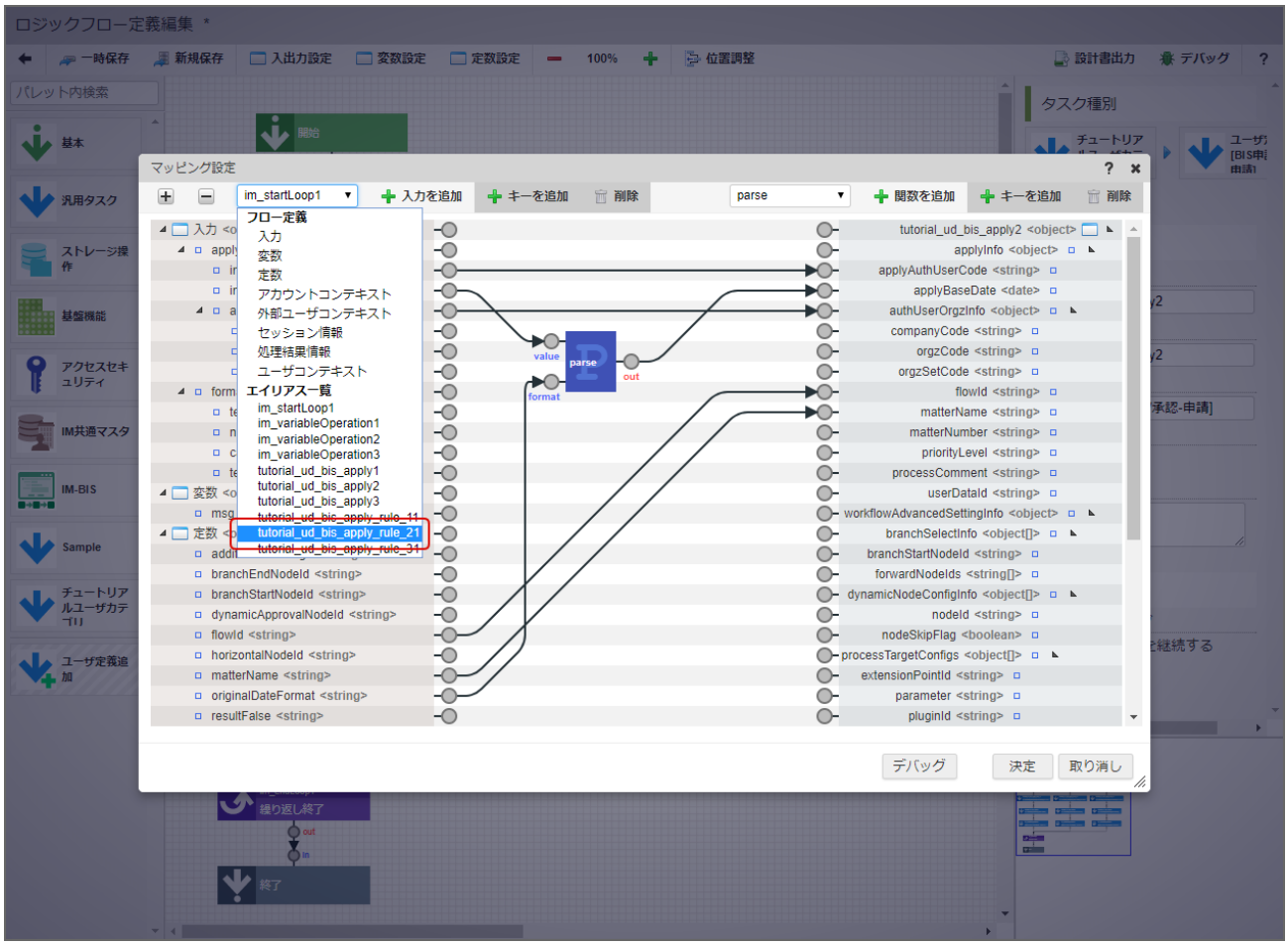



図：関数と出力値の接続

7. これにより、マッピング関数を利用して、日付フォーマットを変換した値がマッピングされました。

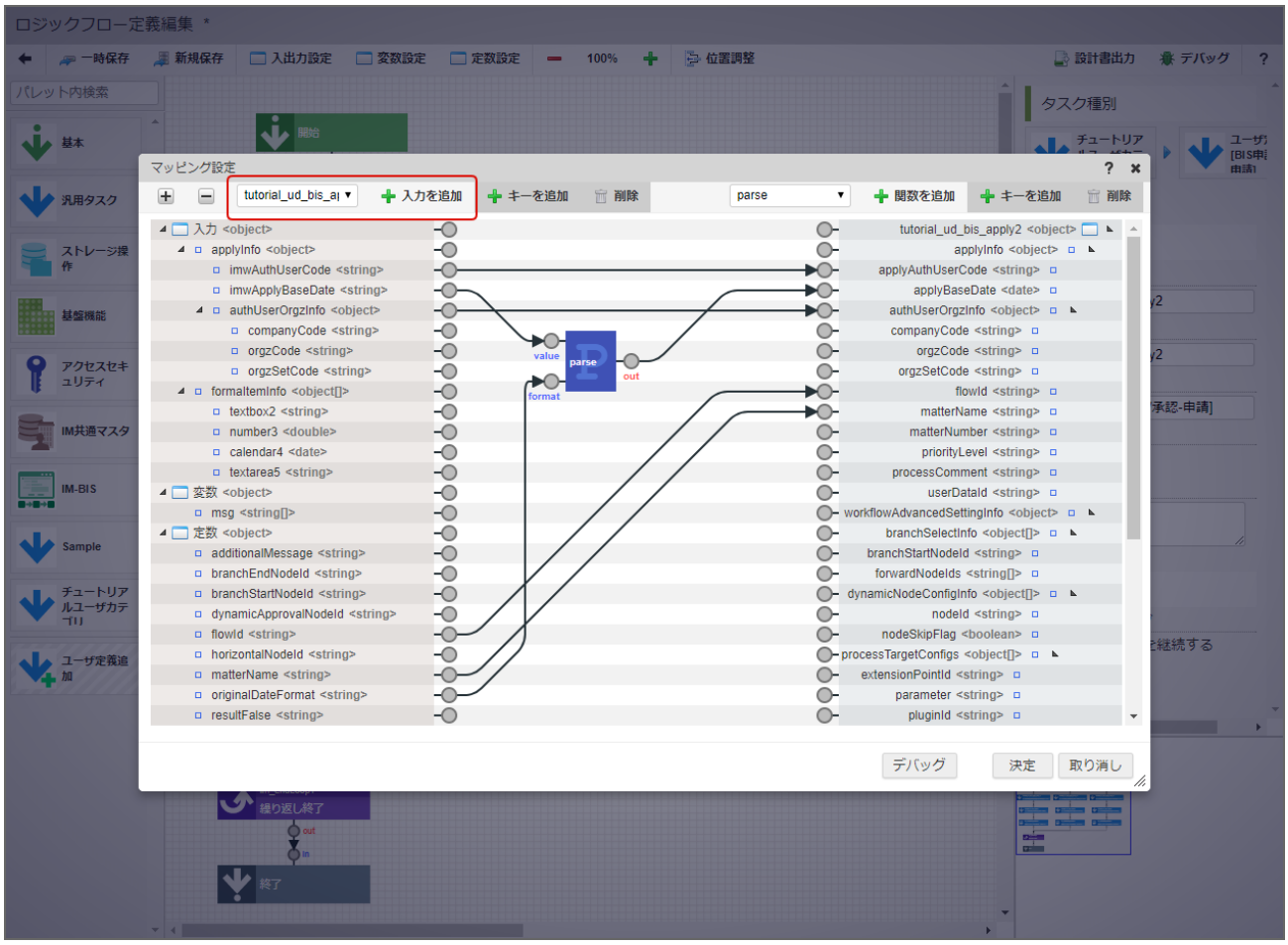
続いて、「ユーザ定義[BIS申請/承認-申請]」タスクのワークフロー設定情報（高度な設定）の設定方法を説明します。

1. マッピング設定画面上部、ヘッダ内の左側に位置するセレクトボックスをクリックし、以下の項目を選択します。
   - エイリアス一覧 - tutorial_ud_bis_apply_rule_21<object> （ユーザ定義[BIS申請/承認-申請]ルール2）



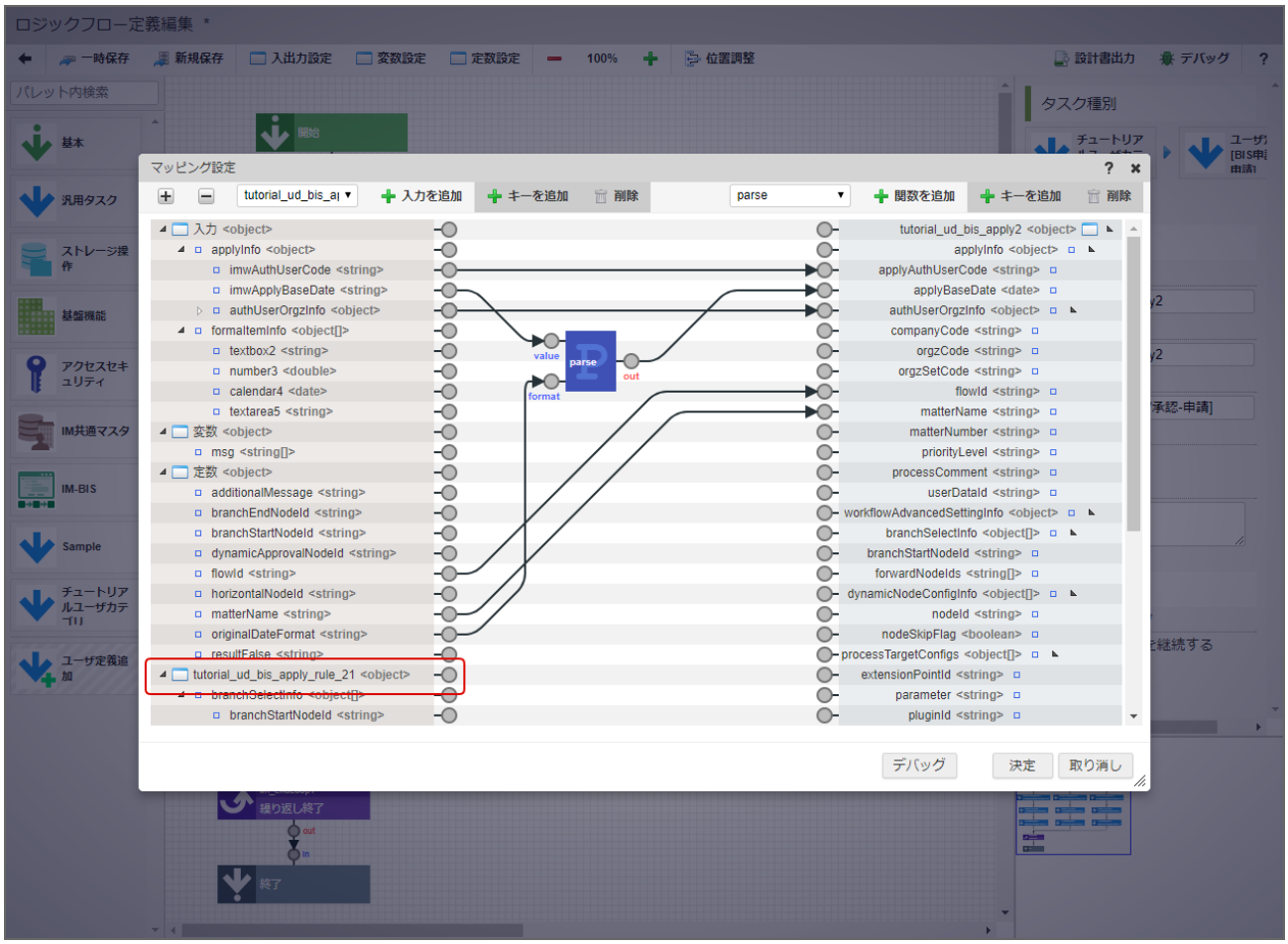

図：入力に追加するエイリアスを選択

2. セレクトボックスの中身が変更されたことを確認し、右側にある「入力を追加」をクリックします。



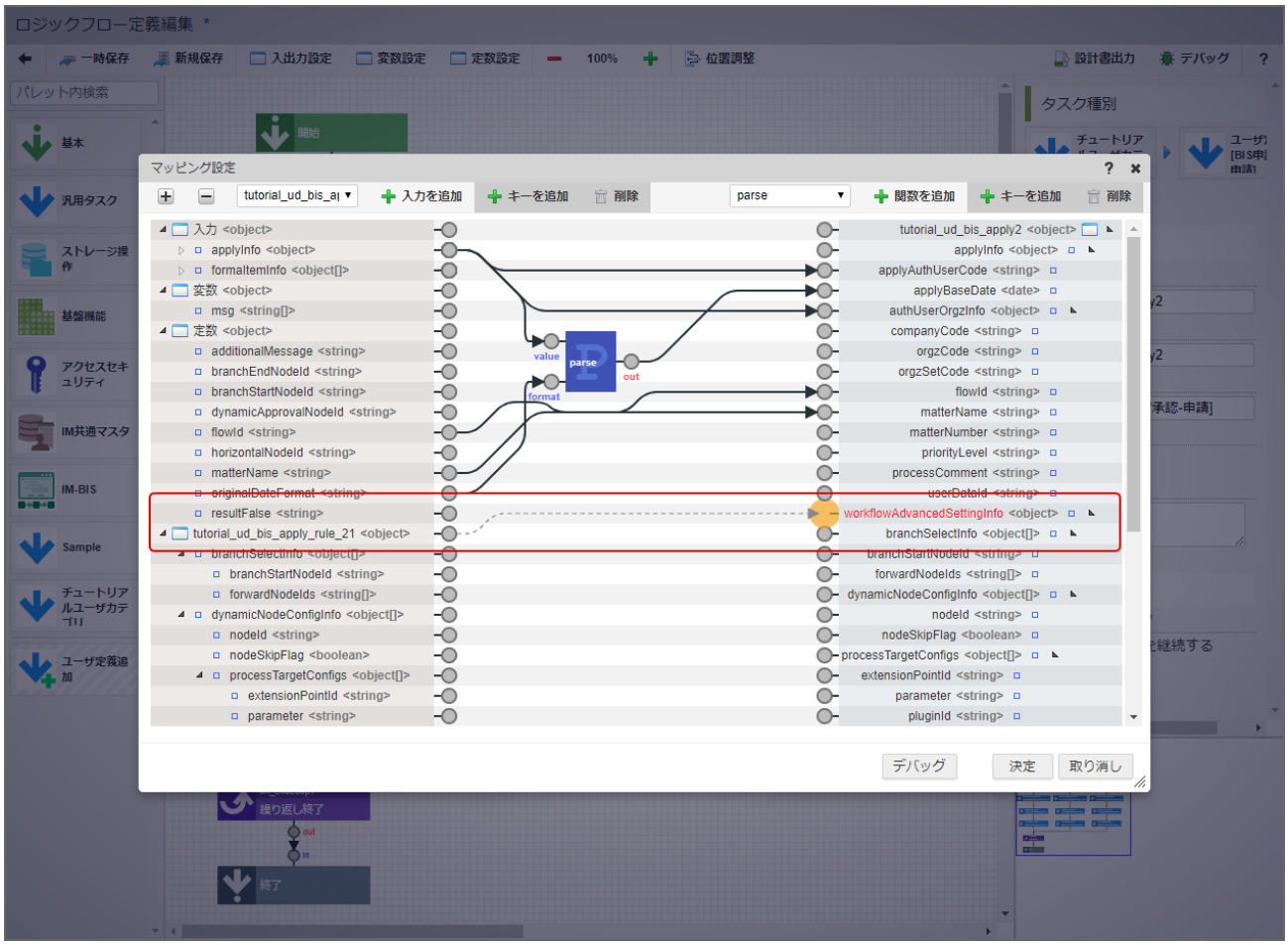

図：「入力を追加」をクリック

3. 入力値として、「ユーザ定義[BIS申請/承認-申請]ルール2」タスク（tutorial_ud_bis_apply_rule_21<object>）の出力値が追加されました。




図：新しい入力値の追加（ユーザ定義[BIS申請/承認-申請]ルール2）

4. 設定画面左部、追加した「tutorial_ud_bis_apply_rule_21<object>」から出ている端子をドラッグし、設定画面右部、「workflowAdvancedSettingInfo<object>」から出ている端子にドロップします。

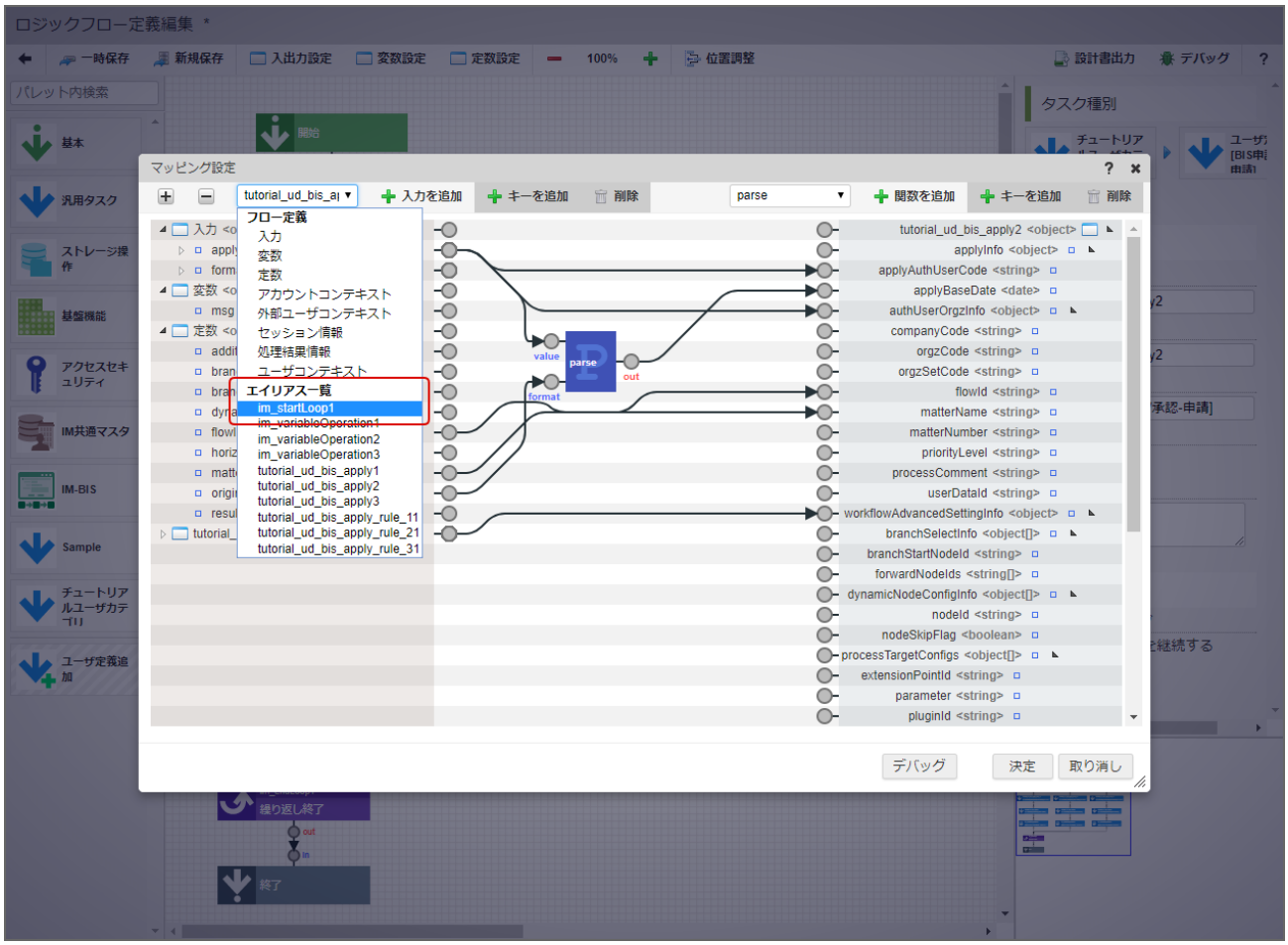



図：新しい入力値（ユーザ定義[BIS申請/承認-申請]ルール2）と出力値の接続

5. これにより、ワークフロー実行情報のマッピングが設定できました。

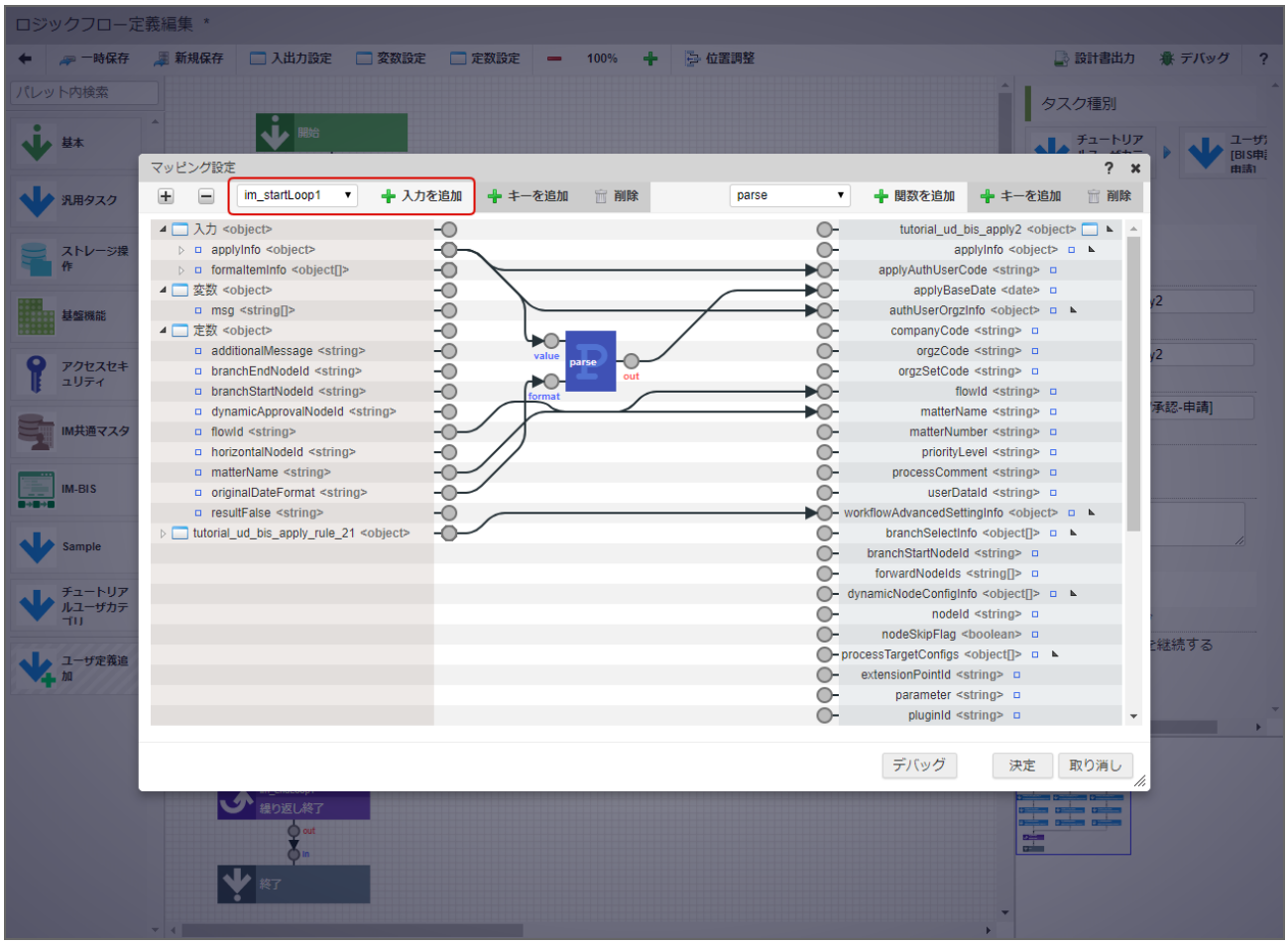
続いて、「ユーザ定義[BIS申請/承認-申請]」タスクの画面アイテム入力情報の設定方法を説明します。

1. マッピング設定画面上部、ヘッダ内の左側に位置するセレクトボックスをクリックし、以下の項目を選択します。
   - エイリアス一覧 - `im_startLoop1<object>` （繰り返し開始）



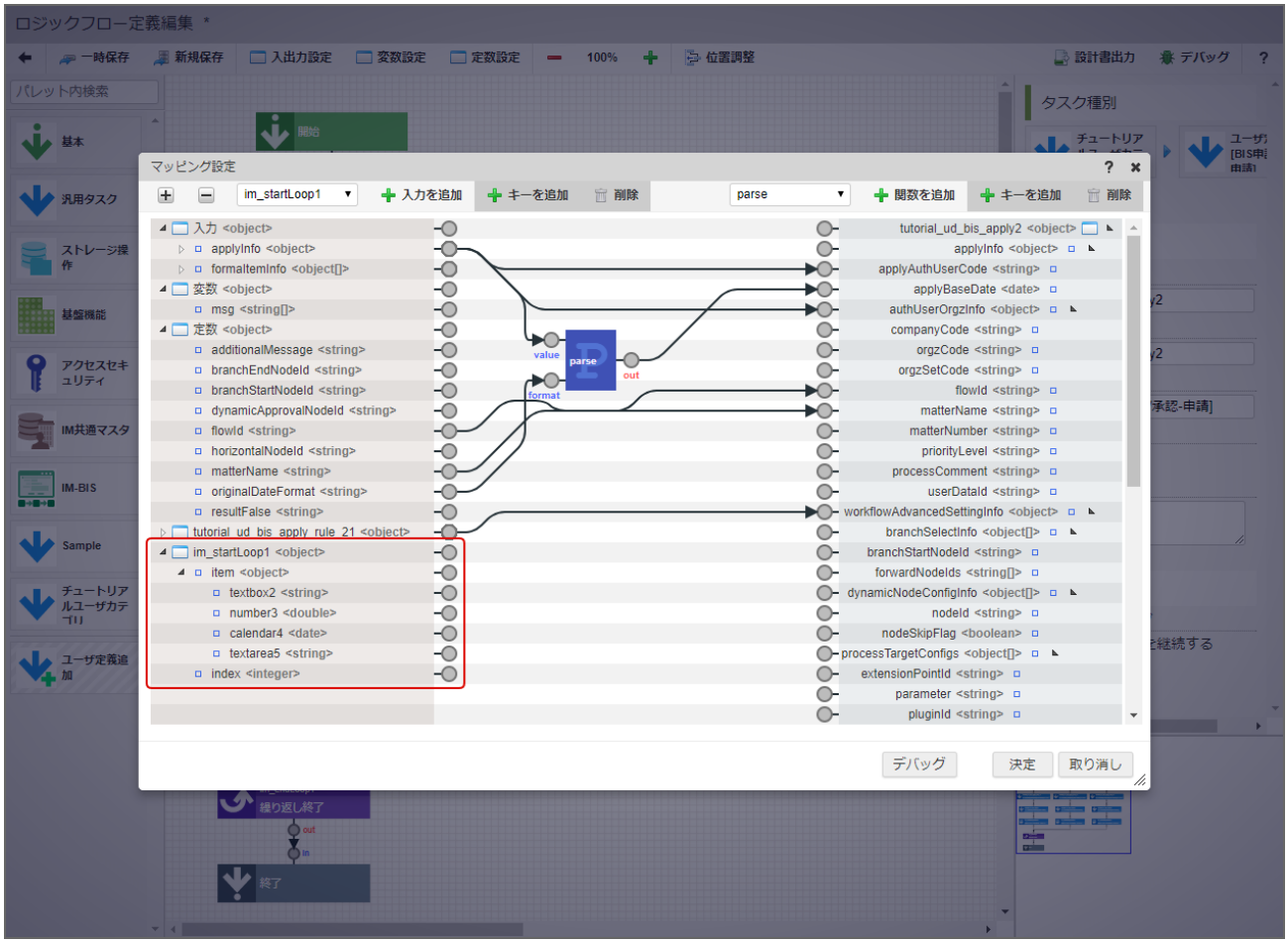

図：入力に追加するエイリアスを選択

2. セレクトボックスの中身が変更されたことを確認し、右側にある「入力を追加」をクリックします。



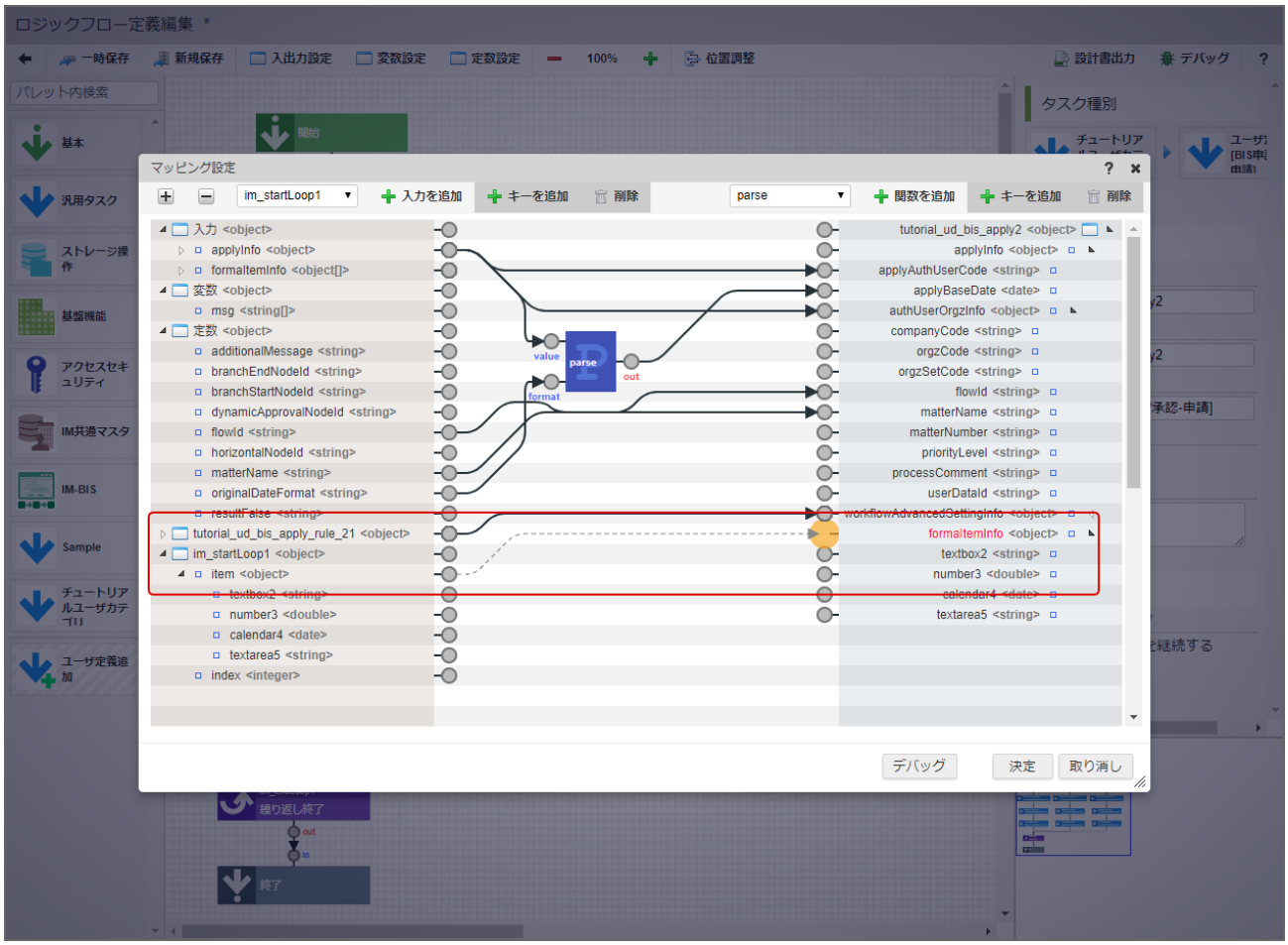

図：「入力を追加」をクリック

3. 入力値として、「繰り返し開始」制御要素（im_startLoop1<object>）の出力値が追加されました。



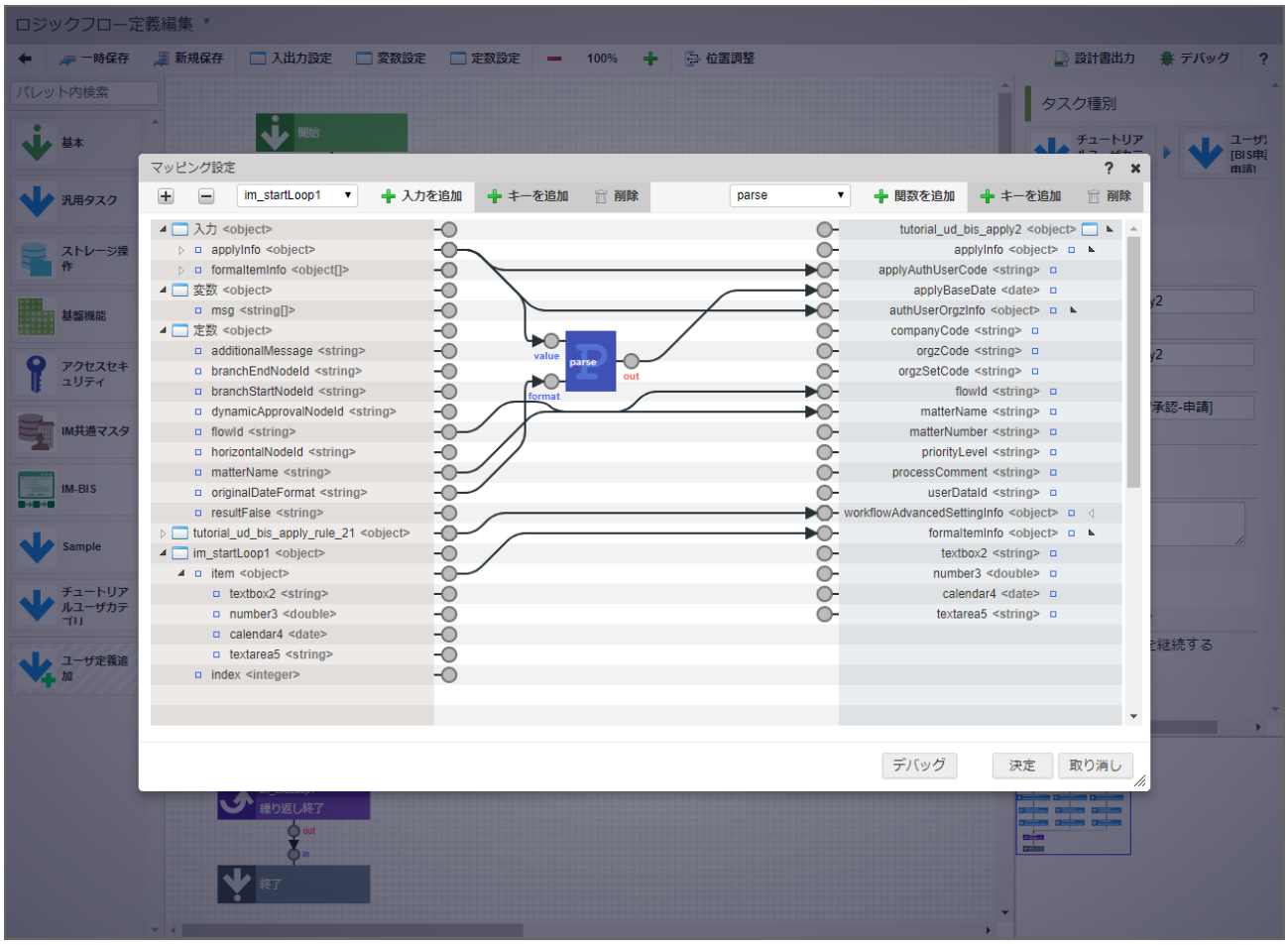

図：入力値の追加（繰り返し開始）

4. 設定画面左部、追加した「im_startLoop1<object>」要素の下にある「item<object>」から出ている端子をドラッグし、設定画面右部、「formaItemInfo<object>」から出ている端子にドロップします。




図：入力値（繰り返し開始）と出力値の接続

5. これにより、繰り返し対象のグリッドテーブルのレコード（行）を1つの申請情報に設定するためのマッピングが設定できました。

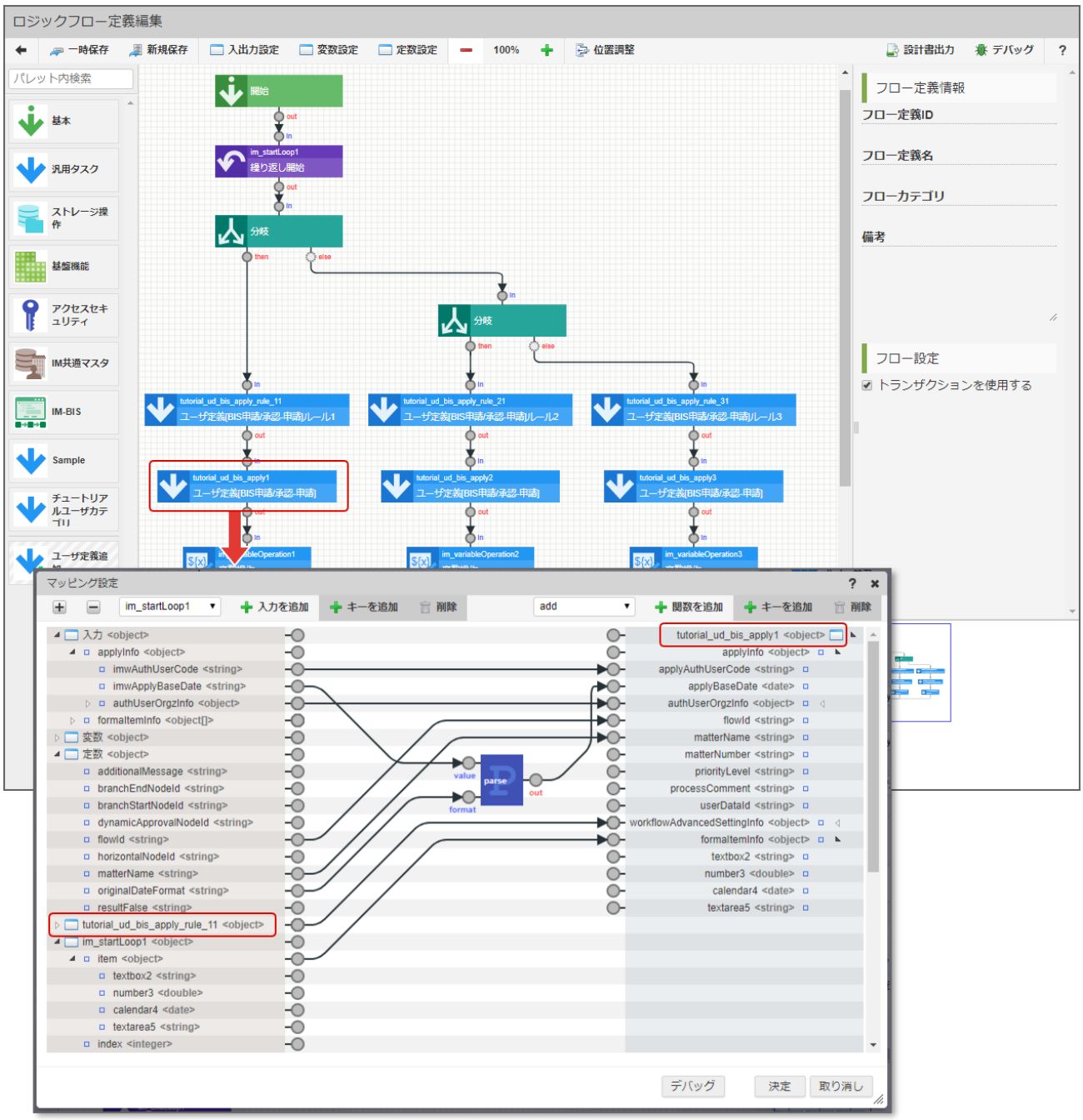



図：入力値（繰り返し開始）と出力値の接続

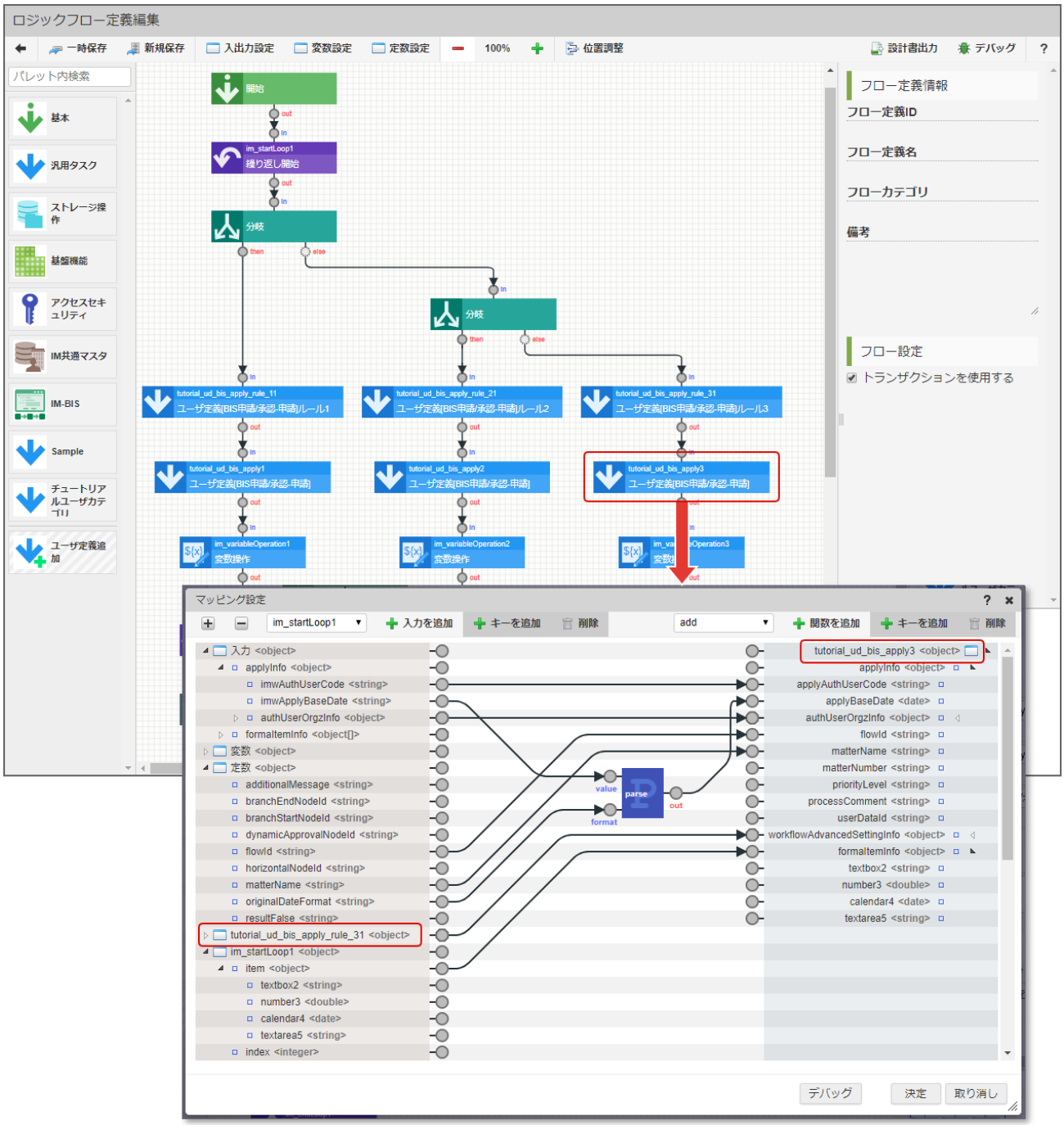
6. ここまでの手順で、「ユーザ定義[BIS申請/承認-申請]」タスクに必要なマッピングが設定できました。
   設定画面右下の決定をクリックし、「ユーザ定義[BIS申請/承認-申請]」タスクのマッピング設定を終了します。

同様の手順で、残り2つの「ユーザ定義[BIS申請/承認-申請]」タスクのマッピングを設定します。
マッピングはほぼ同じですが、ワークフロー設定情報（高度な設定）に接続するユーザ定義が異なる点のみ注意してください。

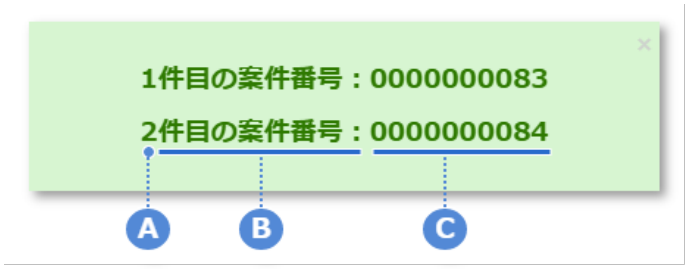
- 1番目の「分岐」制御要素の then に接続する「ユーザ定義[BIS申請/承認-申請]」タスク




- 2番目の「分岐」制御要素の else に接続する「ユーザ定義[BIS申請/承認-申請]」タスク




ここまでのマッピングにより、ワークフローの申請を実行できますが、ユーザに正常に処理が行われたことを伝えるためのメッセージを以下のように表示するためのメッセージを「変数操作」制御要素で設定します。
以下のようにメッセージを画面に表示するための設定を行います。




| 記号 | 説明 |
|---|---|
| A | 「繰り返し」制御要素の出力値を利用し、申請を実行した案件の順番を表示します。 |
| B | 「定数」を利用し、固定のメッセージを表示します。 |
| C | ユーザ定義「BIS申請/承認-申請」の出力値を利用し、申請を行った案件の案件番号を表示します。 |

分岐ルート別の設定内容は以下のとおりです。

- **変数操作** （左のルート）

| 入力（始点） | 出力（終点） |
|---|---|
| im_startLoop1<object> - index<integer><br>定数<object> - additionalMessage<string> | concat (1) - a<br>concat (1) - b |
| concat (1) - out<br>tutorial_ud_bis_apply1<object> - matterNumber<string> | concat (2) - a<br>concat (2) - b |
| 変数<object> - msg<string[]><br>concat (2) - out | push - array<br>push - value |
| push - out | 変数<object> - msg<string[]> |

- **変数操作** （中央のルート）

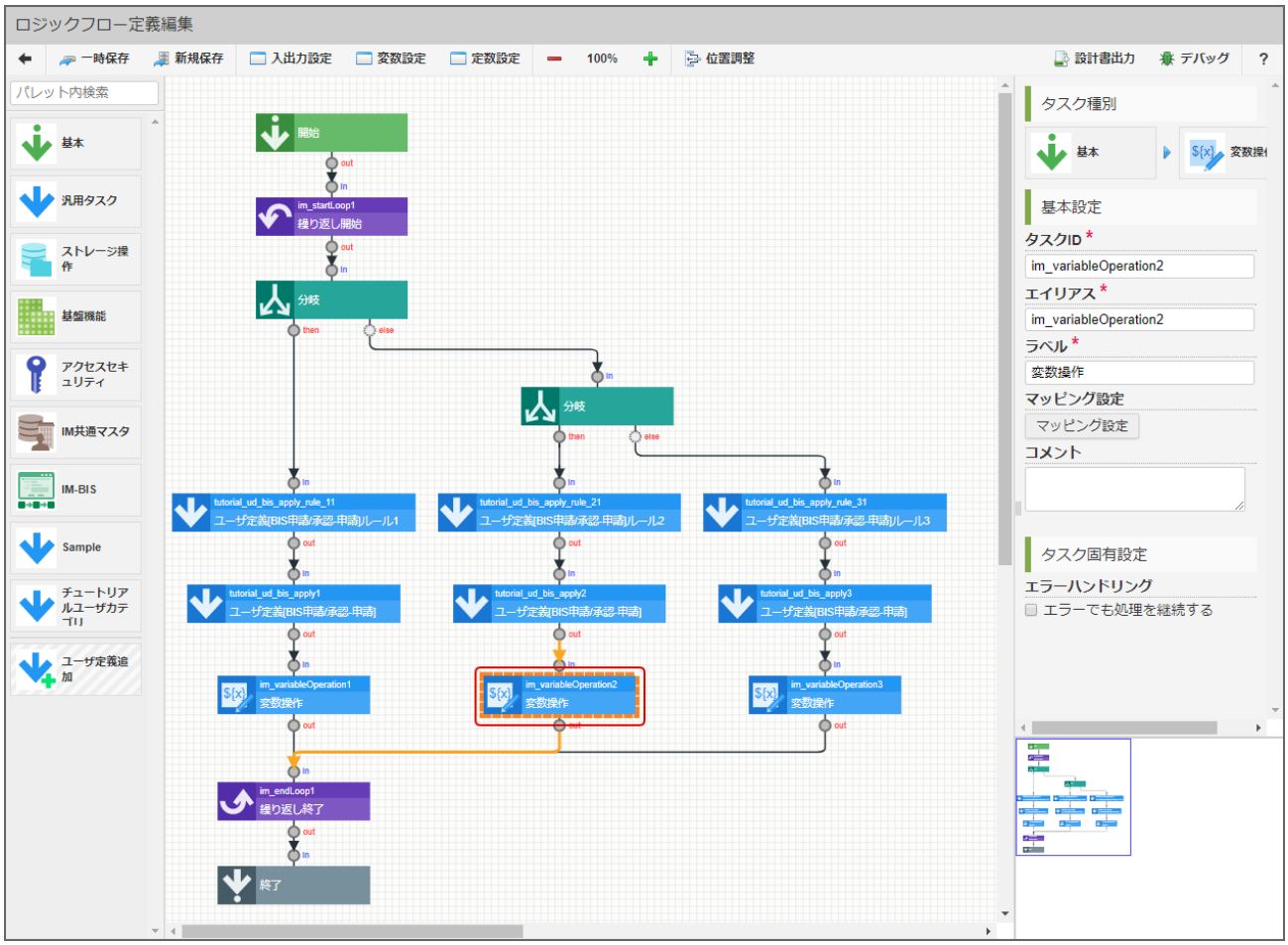
| 入力（始点） | 出力（終点） |
|---|---|
| im_startLoop1<object> - index<integer><br>定数<object> - additionalMessage<string> | concat (1) - a<br>concat (1) - b |
| concat (1) - out<br>tutorial_ud_bis_apply2<object> - matterNumber<string> | concat (2) - a<br>concat (2) - b |
| 変数<object> - msg<string[]><br>concat (2) - out | push - array<br>push - value |
| push - out | 変数<object> - msg<string[]> |

- **変数操作** （右のルート）

| 入力（始点） | 出力（終点） |
|---|---|
| im_startLoop1<object> - index<integer><br>定数<object> - additionalMessage<string> | concat (1) - a<br>concat (1) - b |
| concat (1) - out<br>tutorial_ud_bis_apply3<object> - matterNumber<string> | concat (2) - a<br>concat (2) - b |
| 変数<object> - msg<string[]><br>concat (2) - out | push - array<br>push - value |
| push - out | 変数<object> - msg<string[]> |

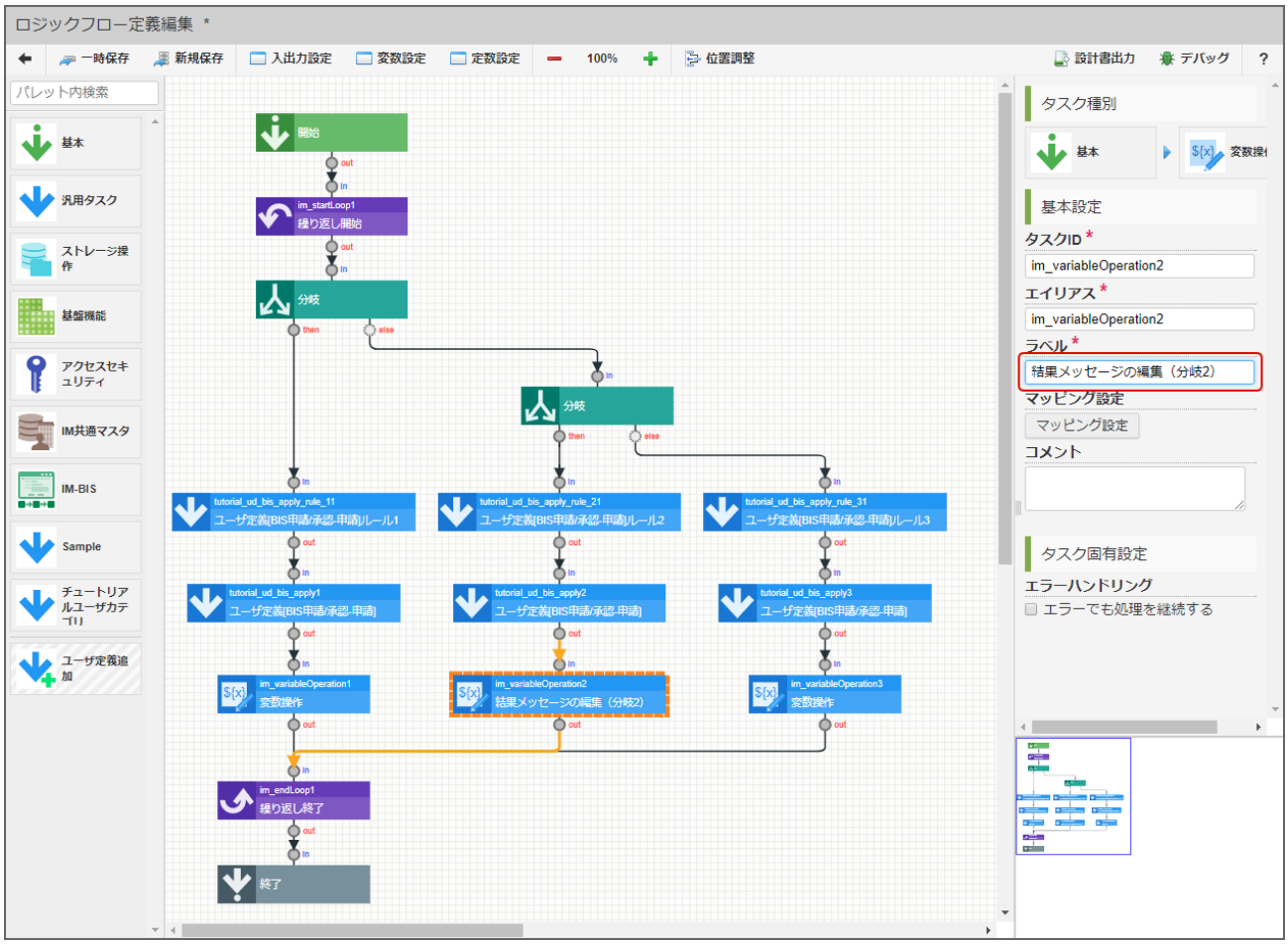
実際に「変数操作」制御要素のマッピングを設定します。

1. 中央のルート上の「変数操作」制御要素をクリックします。



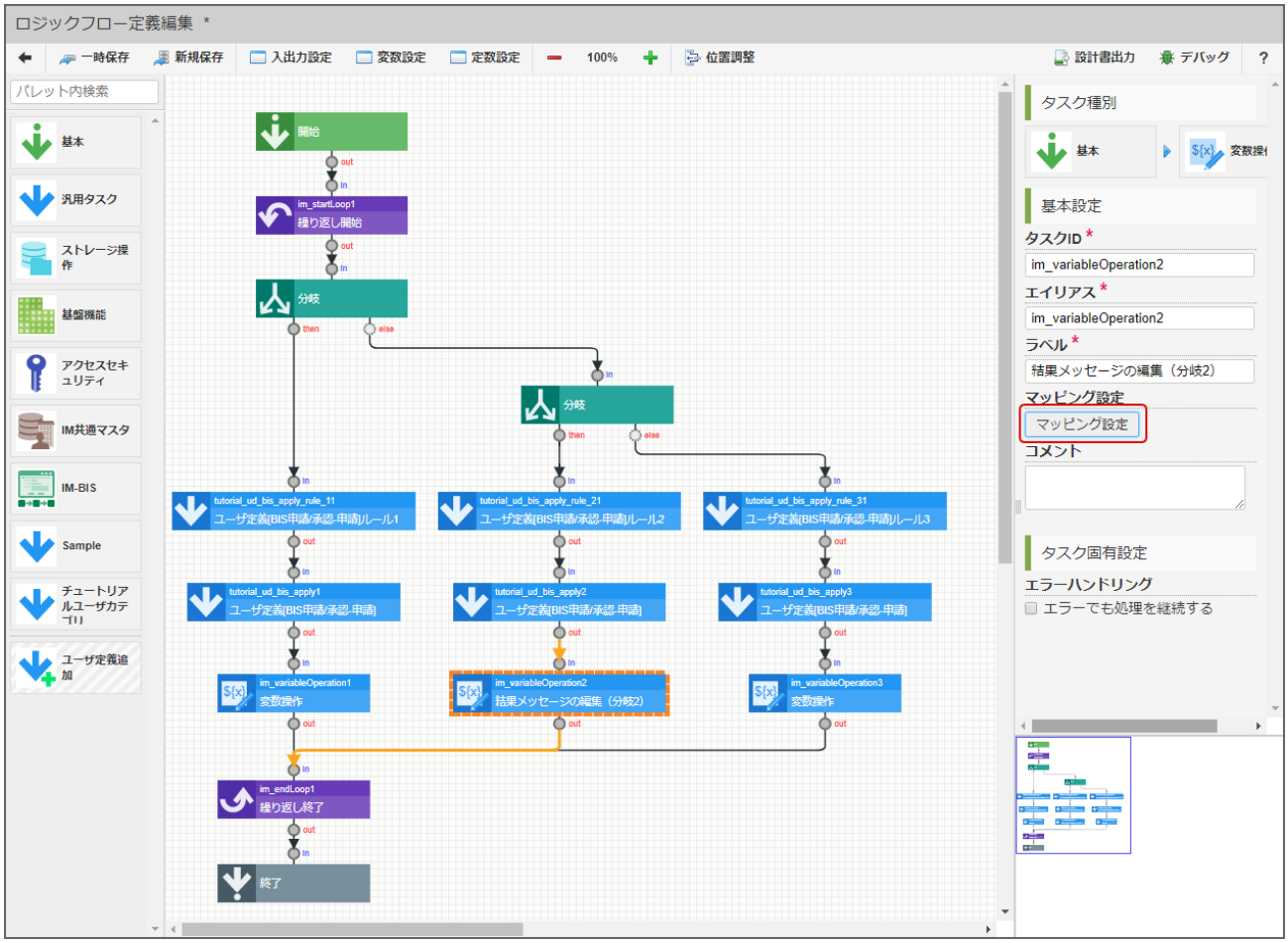

図：「変数操作」をクリック

2. ラベルを「結果メッセージの編集（分岐2）」に変更します。



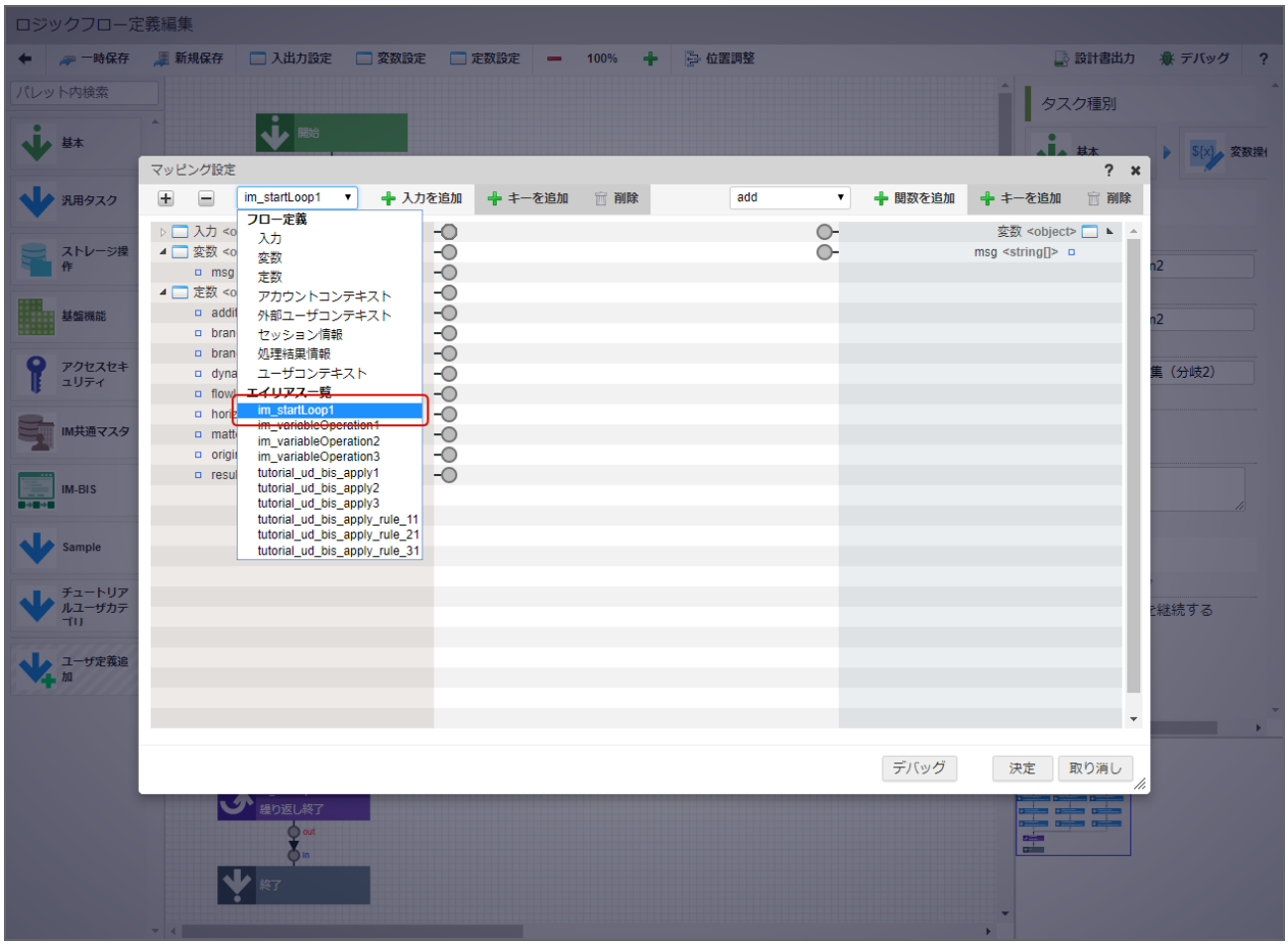

図：「変数操作」制御要素のラベルの変更

3. 「マッピング設定」をクリックします。



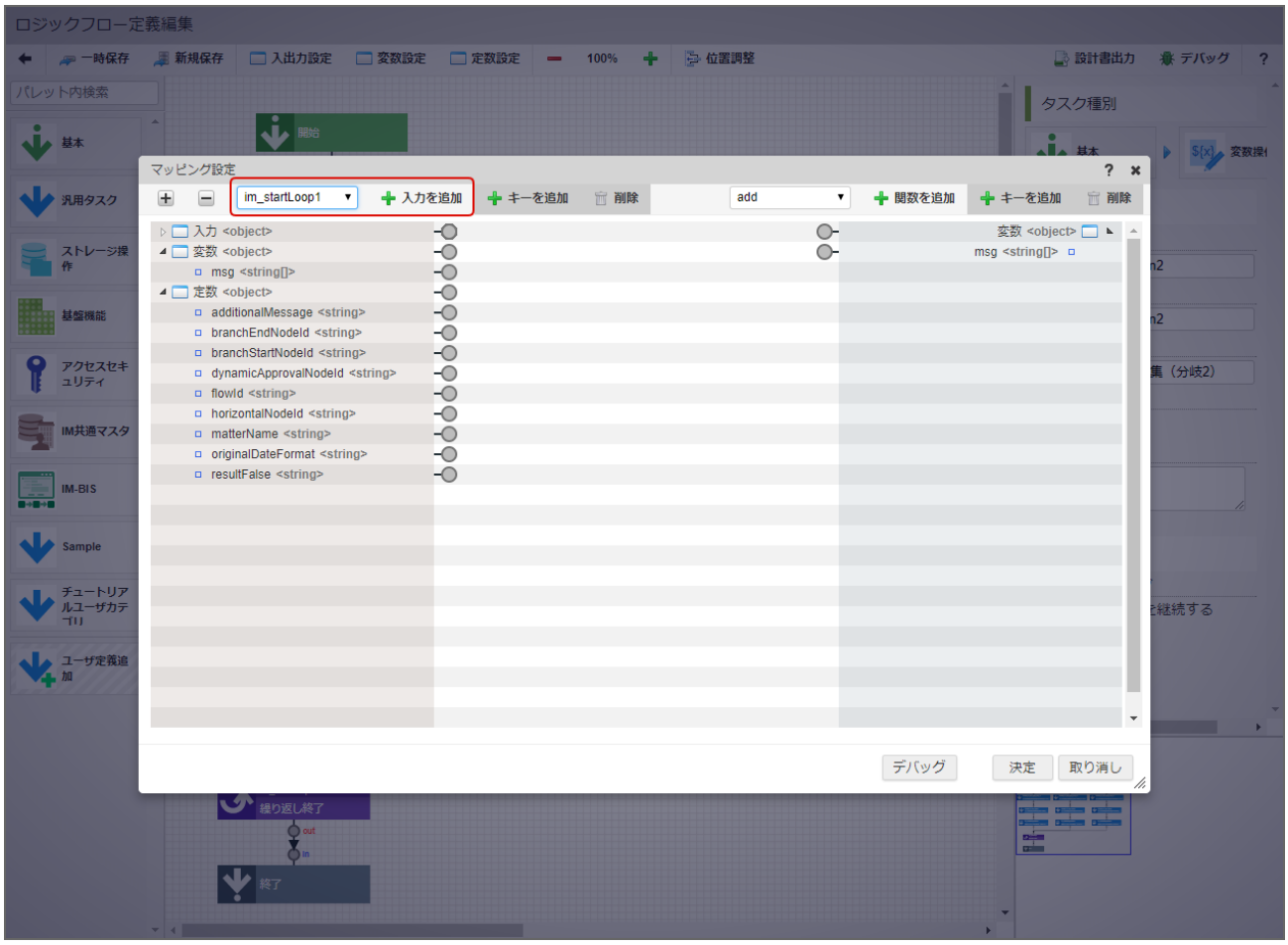

図：「変数操作」制御要素のマッピング設定の変更

4. マッピング設定画面上部、ヘッダ内の左側に位置するセレクトボックスをクリックし、以下の項目を選択します。
   - エイリアス一覧 - im_startLoop1<object> （繰り返し開始）



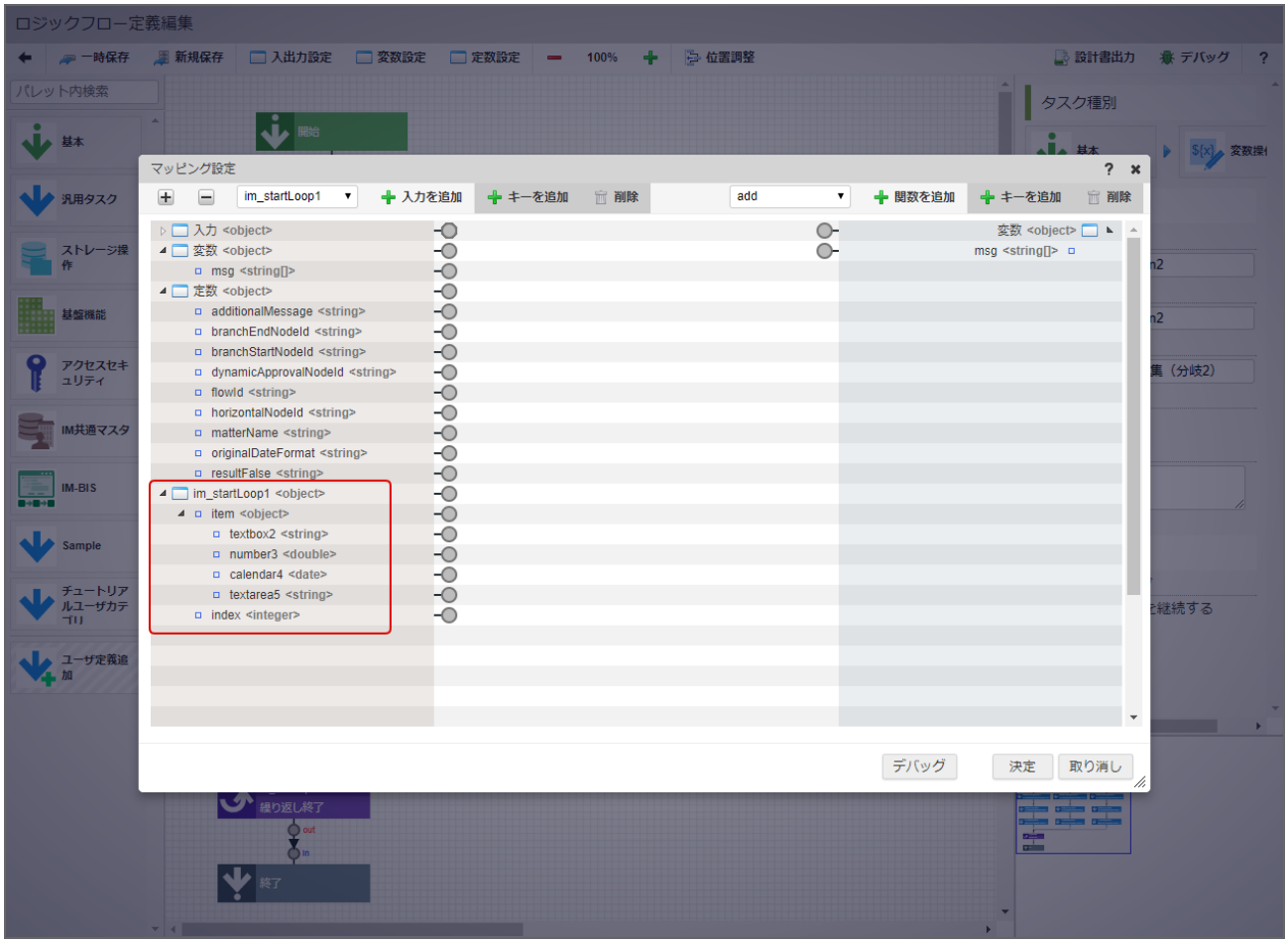

図：「繰り返し」制御要素をプルダウンから選択

5. セレクトボックスの中身が変更されたことを確認し、右側にある「入力を追加」をクリックします。



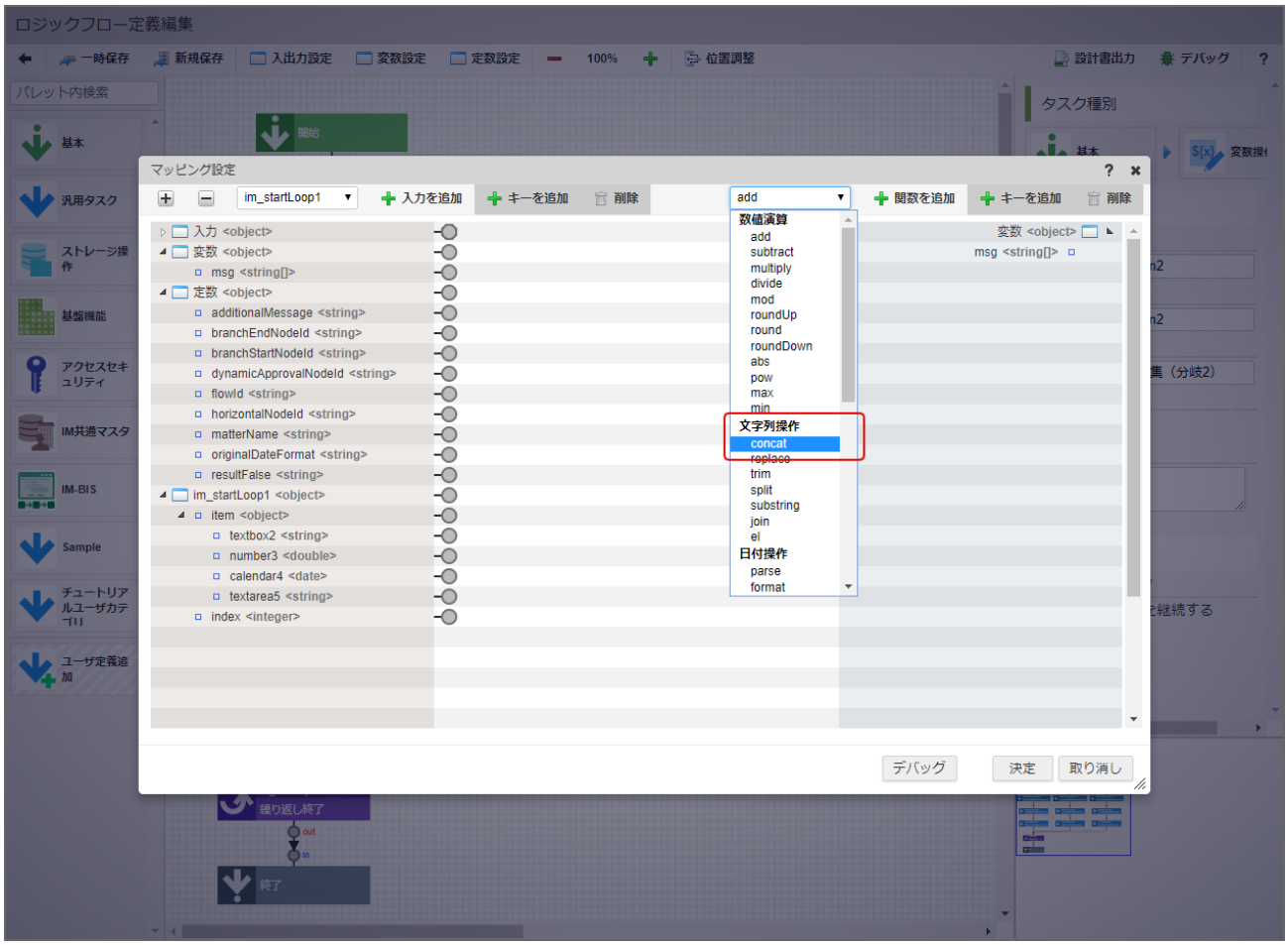

図：入力に「繰り返し」制御要素を追加

6. 入力値として、「繰り返し開始」制御要素（im_startLoop1<object>）の出力値が追加されました。



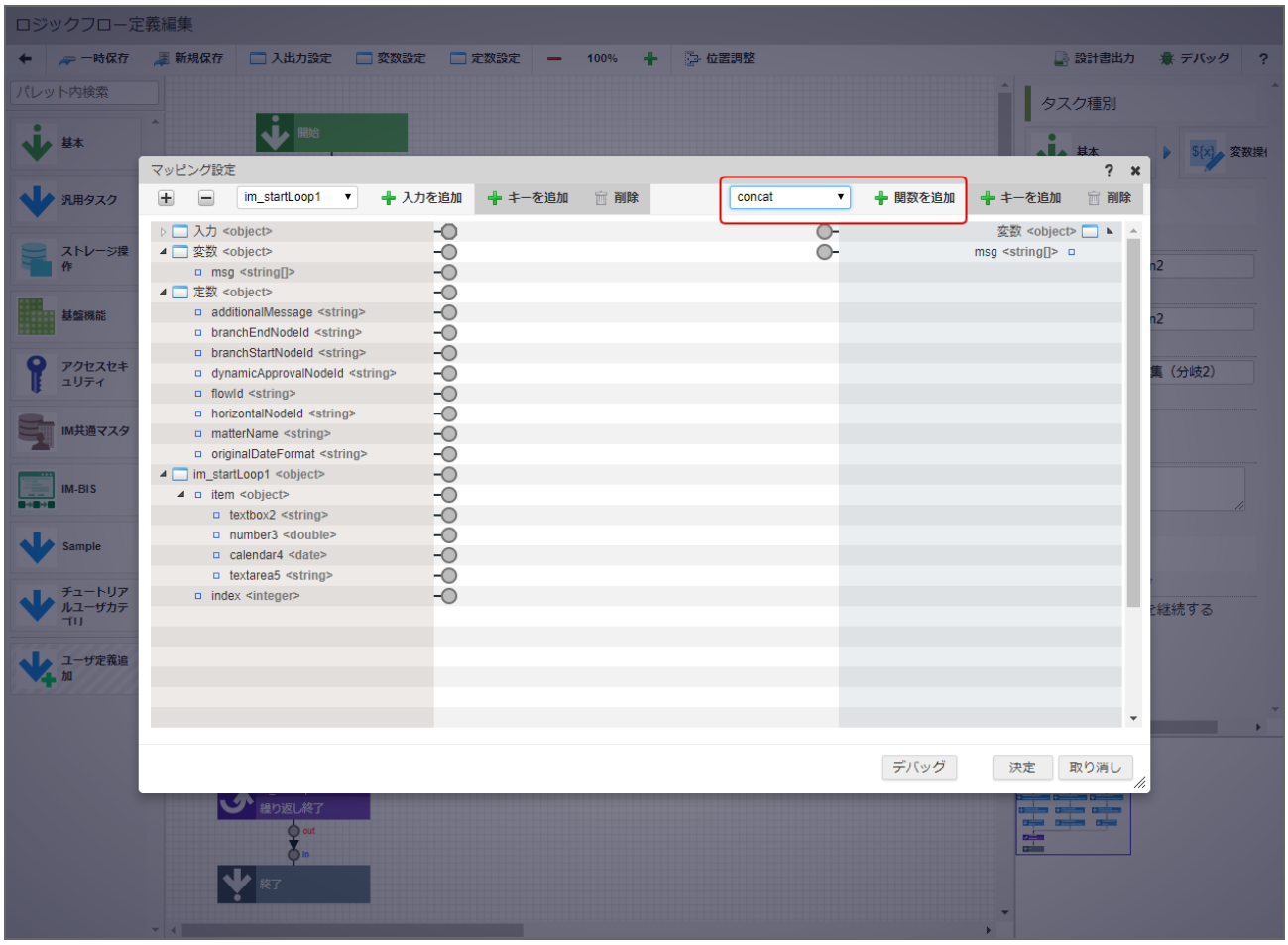

図：入力値の追加（繰り返し開始）

7. マッピング設定画面上部、ヘッダ内の中央右寄りに位置するセレクトボックスをクリックし、以下の項目を選択します。
   - 文字列操作 - concat



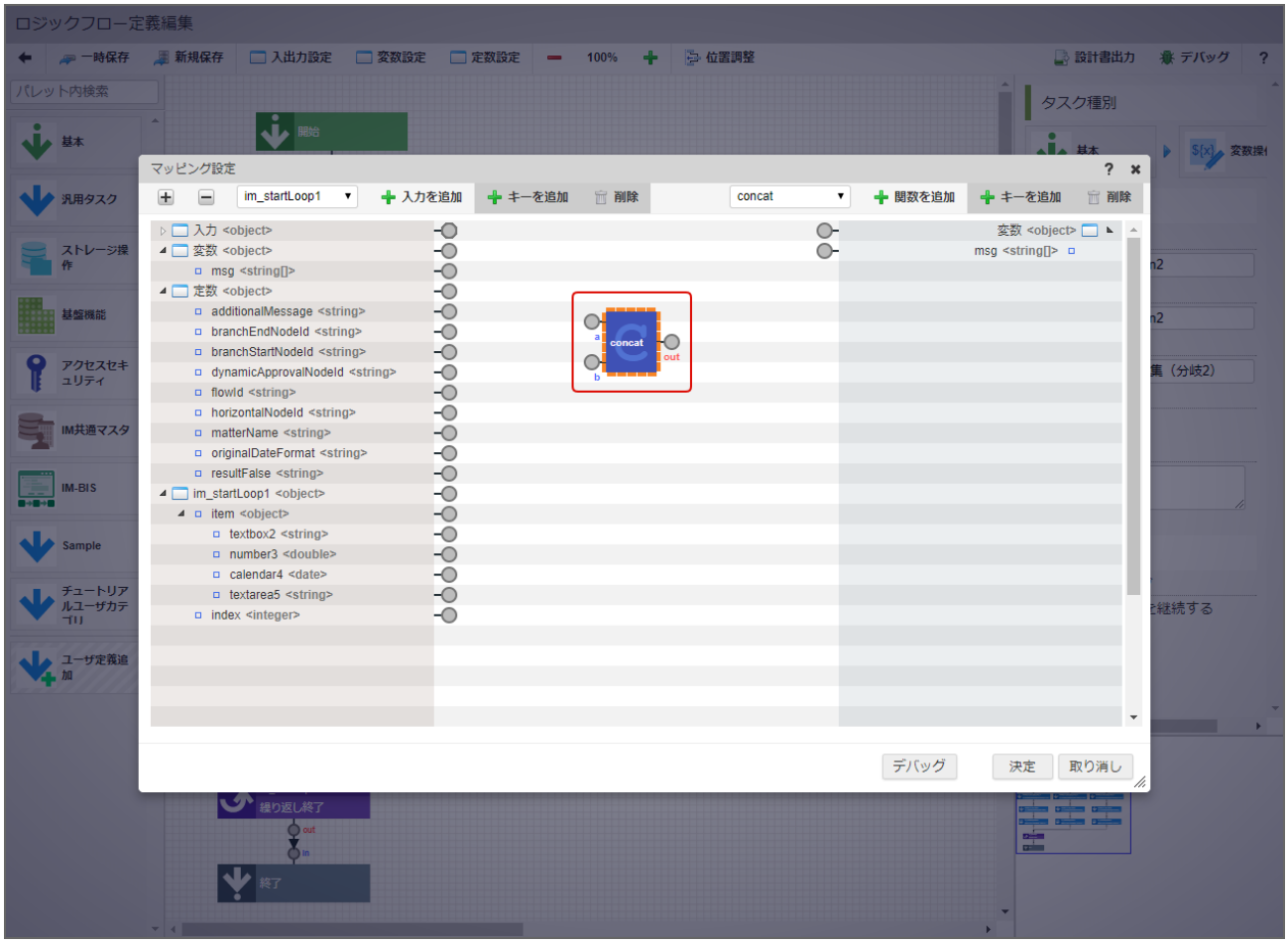

図：マッピング関数「concat」の選択

8. セレクトボックスの中身が変更されたことを確認し、右側にある「関数を追加」をクリックします。



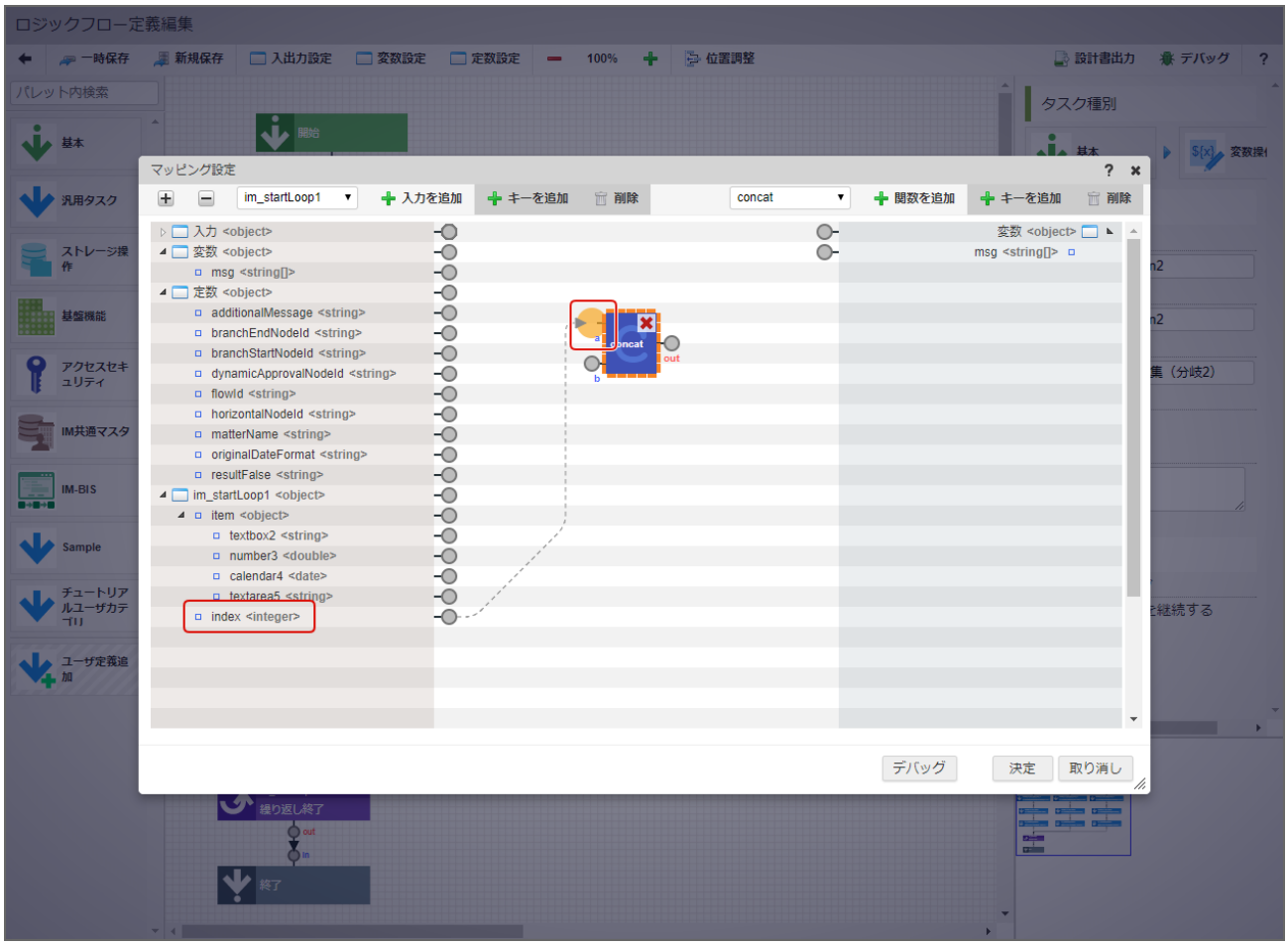

図：「関数を追加」をクリック

9. マッピング関数として、「concat」関数が追加されました。



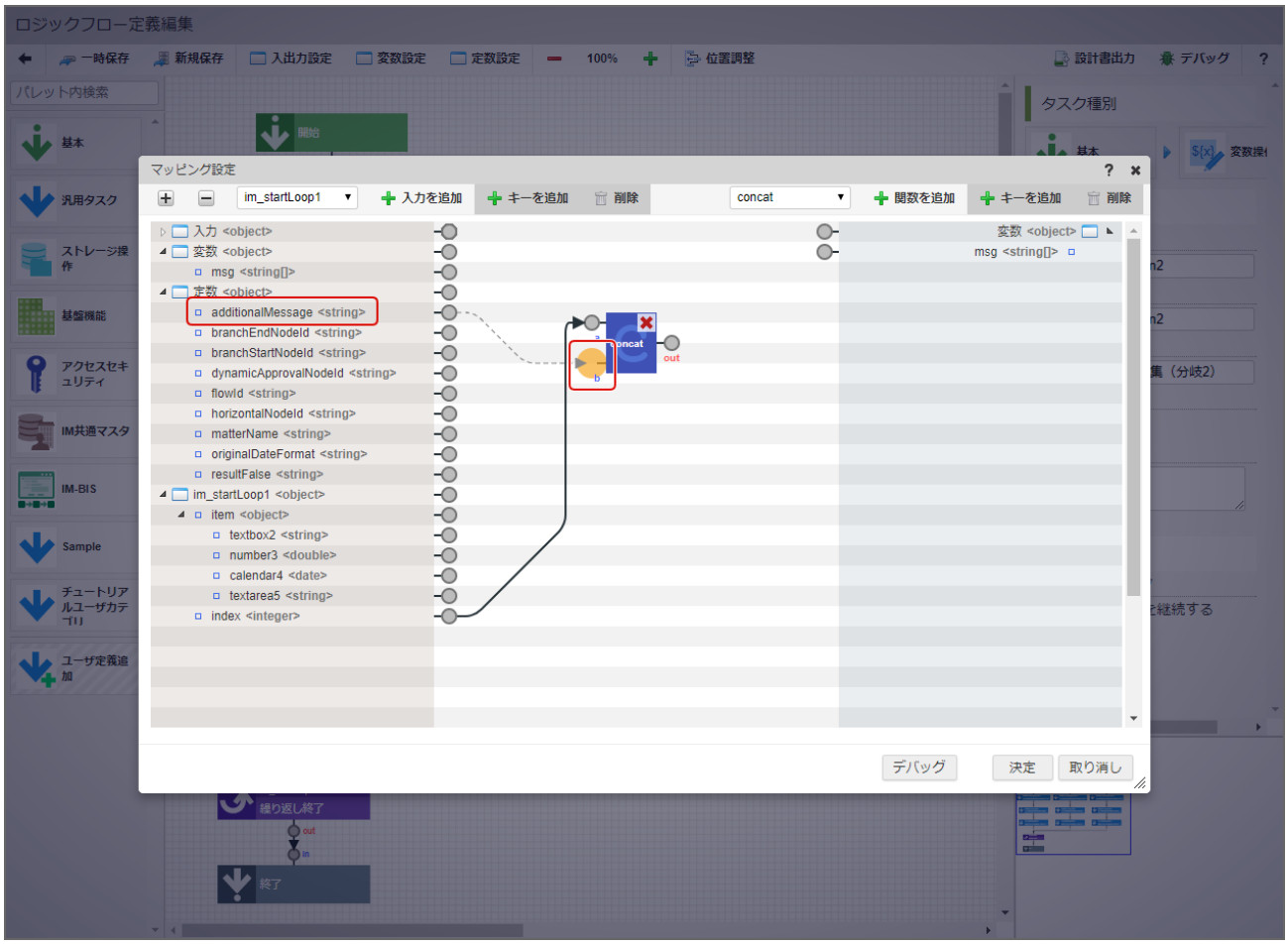

図：関数（concat）の追加

10. 設定画面左部、「im_startLoop1<object>」要素の下にある「index<integer>」から出ている端子をドラッグし、「concat」関数の左部から出ている端子のうち「a」へドロップします。



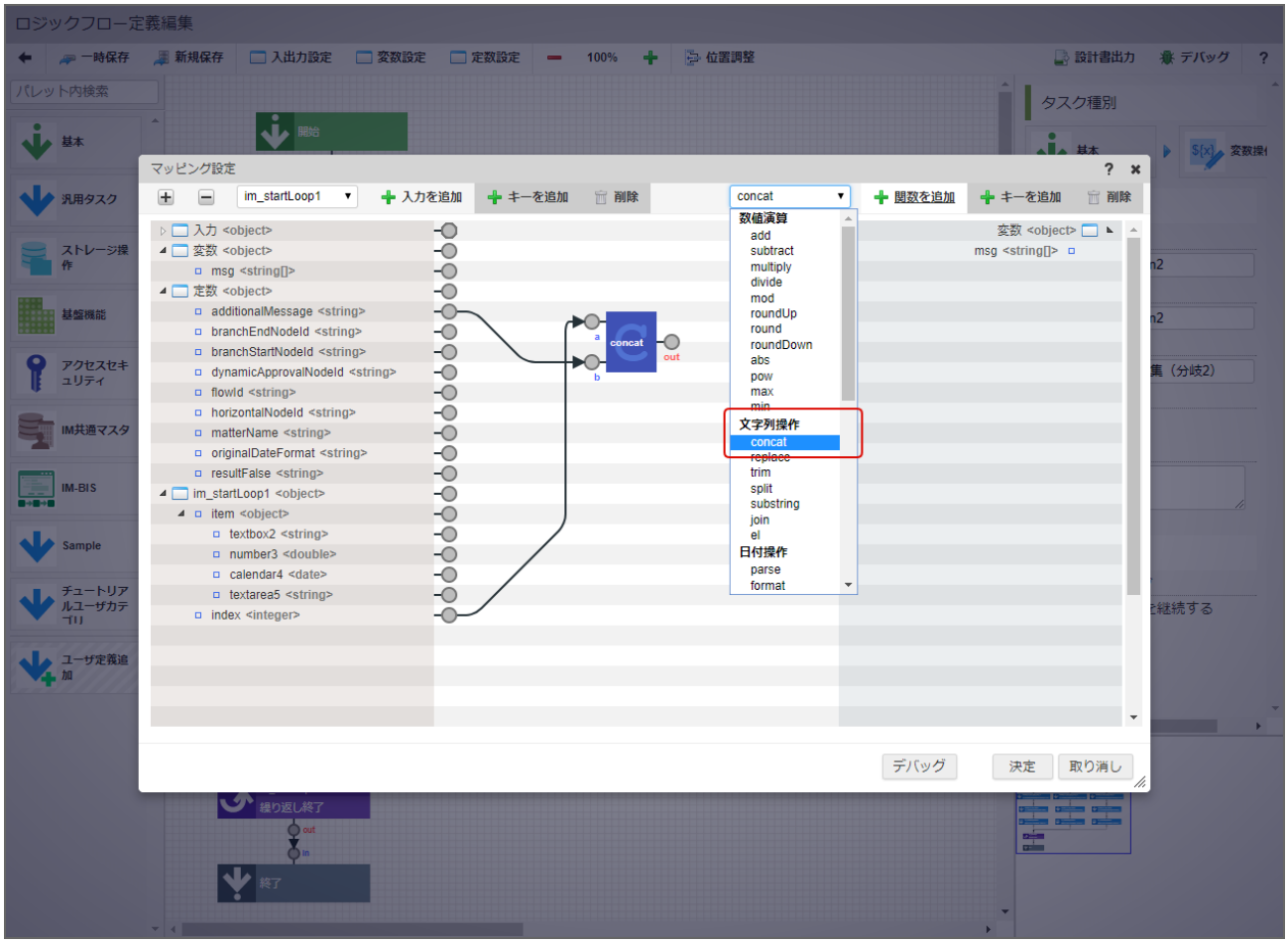

図：関数と入力値（「繰り返し」制御要素）の接続

11. 同様に、「定数<object>」要素の下にある「 additionalMessage<string> 」から出ている端子をドラッグし、「concat」関数の左部から出ている端子のうち「 b 」へドロップします。



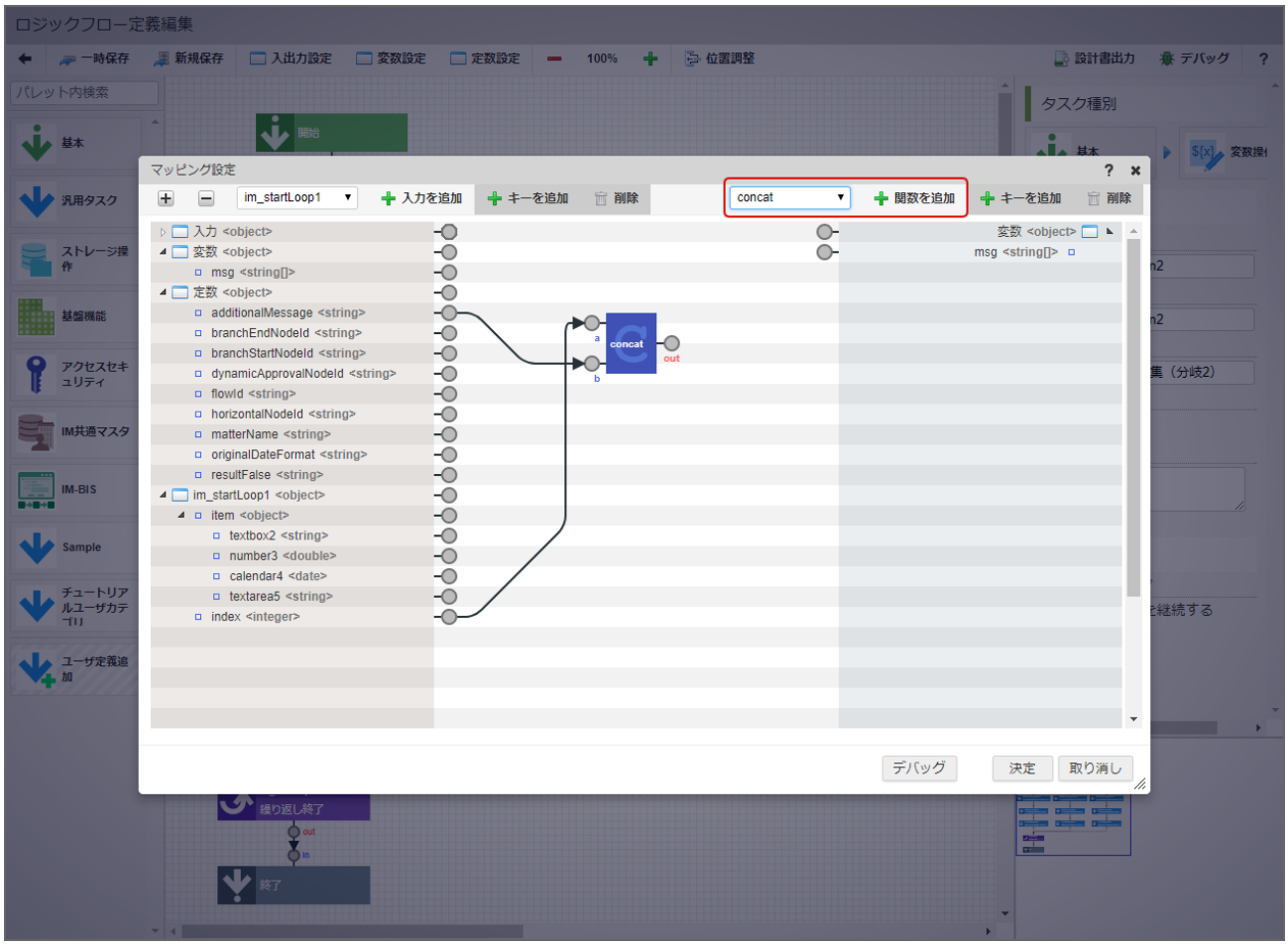
図：関数と定数の接続

12. マッピング設定画面上部、ヘッダ内の中央右寄りに位置するセレクトボックスをクリックし、以下の項目を選択します。
    - 文字列操作 - concat



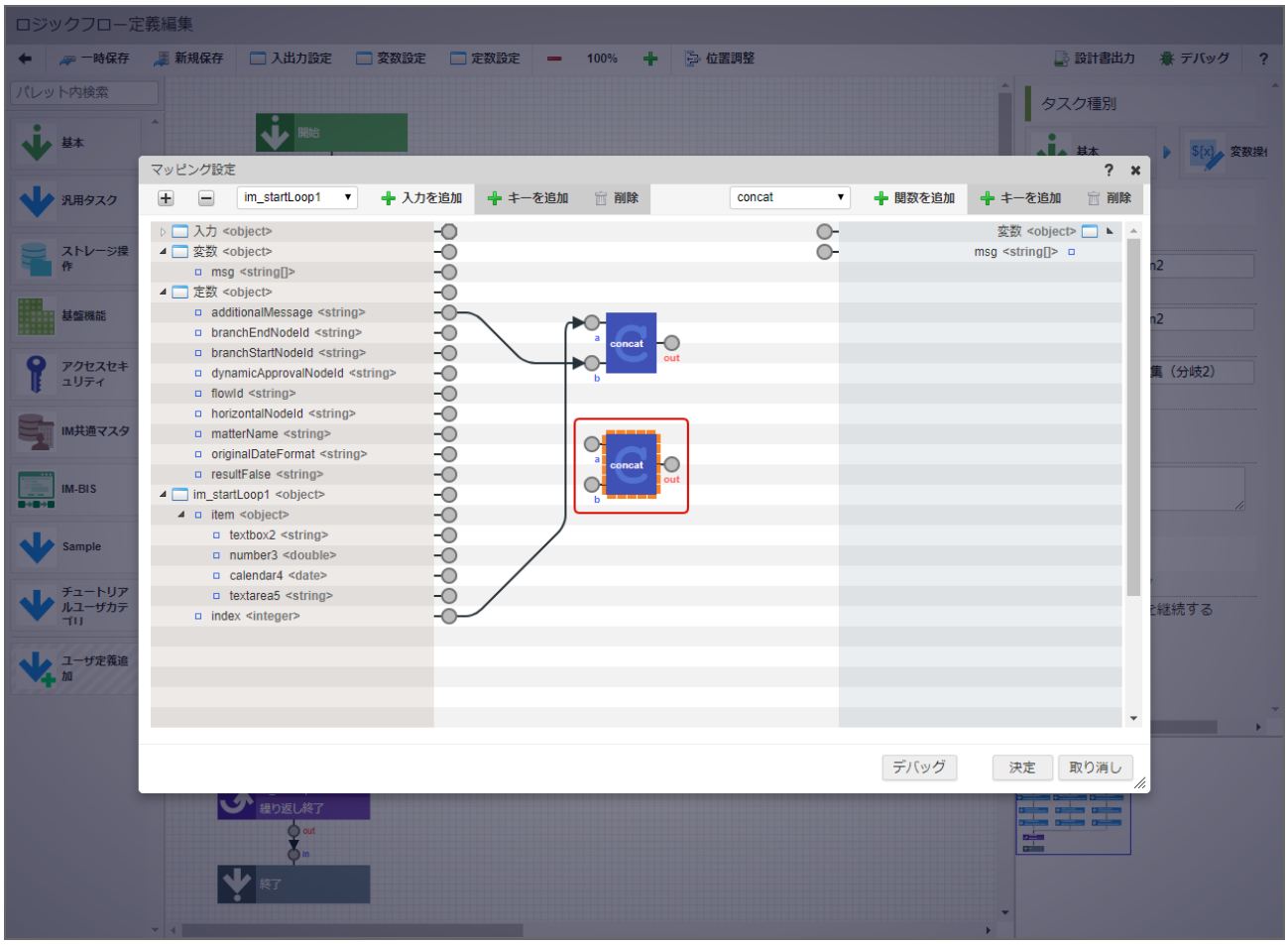

図：マッピング関数「concat」の選択

13. セレクトボックスの中身が変更されたことを確認し、右側にある「関数を追加」をクリックします。



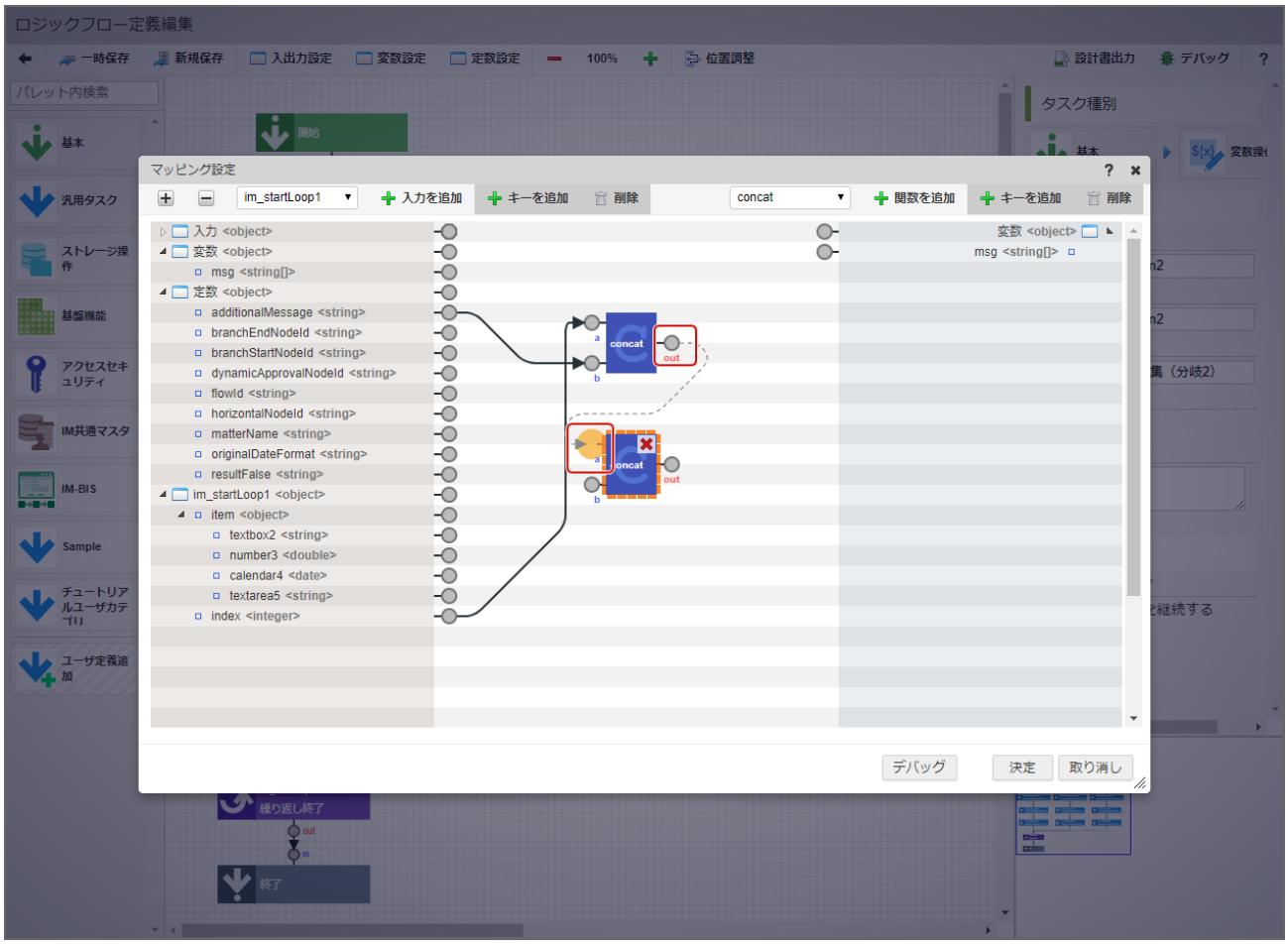

図：「関数を追加」をクリック

14. マッピング関数として、「concat」関数が追加されました。



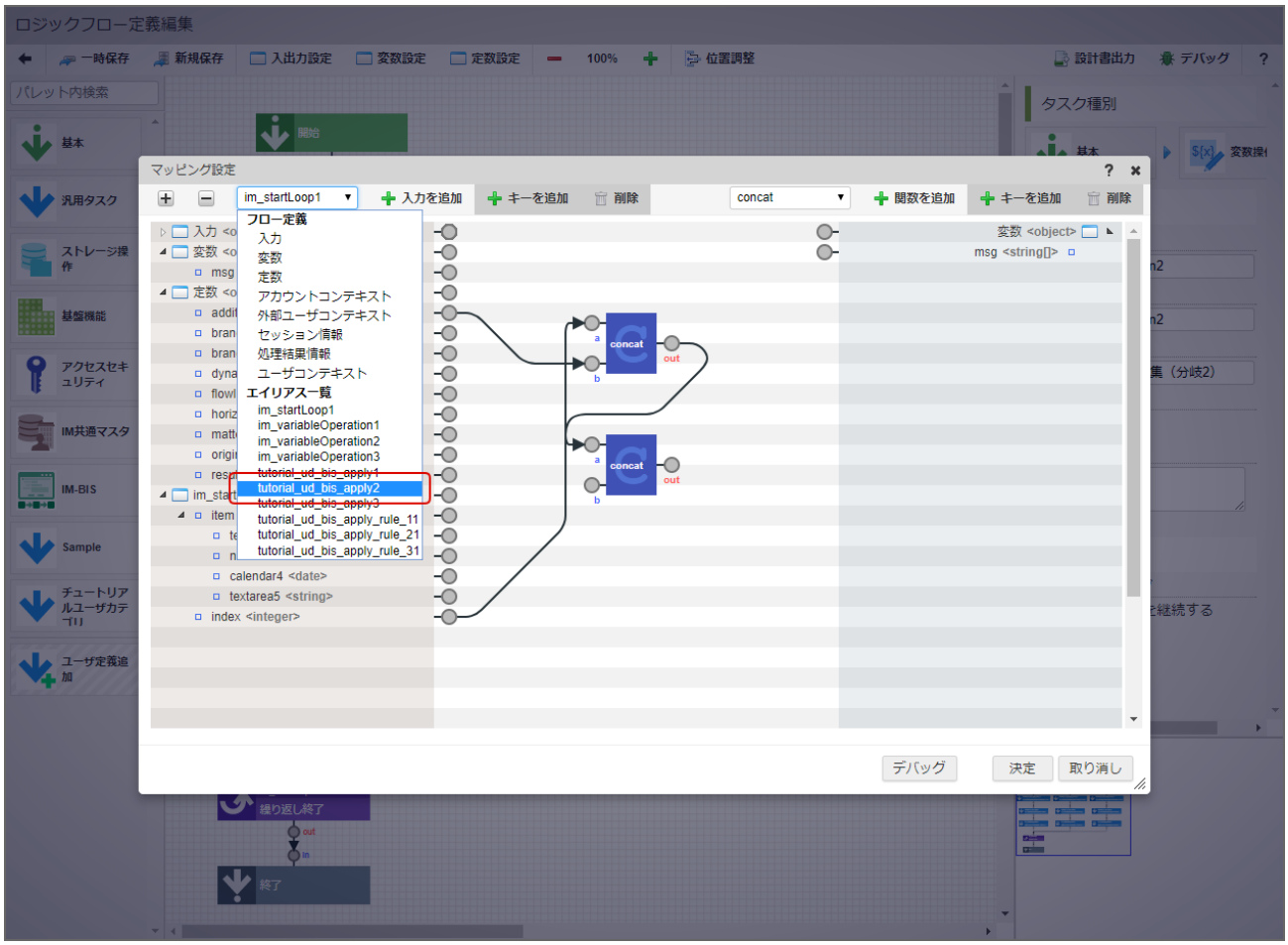

図：関数（concat）の追加

15. 先に配置した関数「concat」から出ている端子をドラッグし、後に配置した関数「concat」の左部から出ている端子のうち「a」へドロップします。



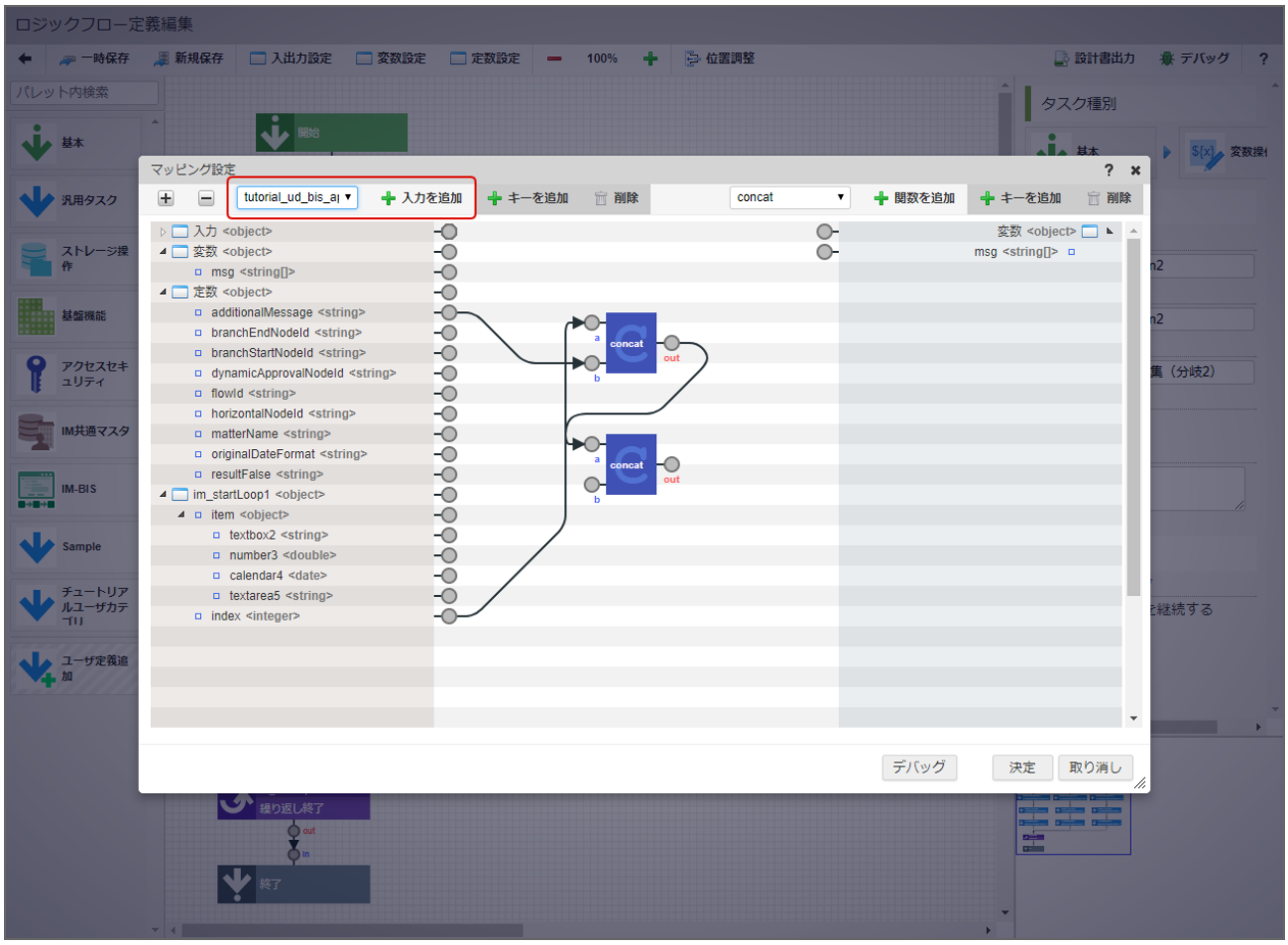

図：関数（concat）同士の接続

16. マッピング設定画面上部、ヘッダ内の左側に位置するセレクトボックスをクリックし、以下の項目を選択します。
    - エイリアス一覧 - tutorial_ud_bis_apply2<object> （「ユーザ定義[BIS申請/承認-申請]」）



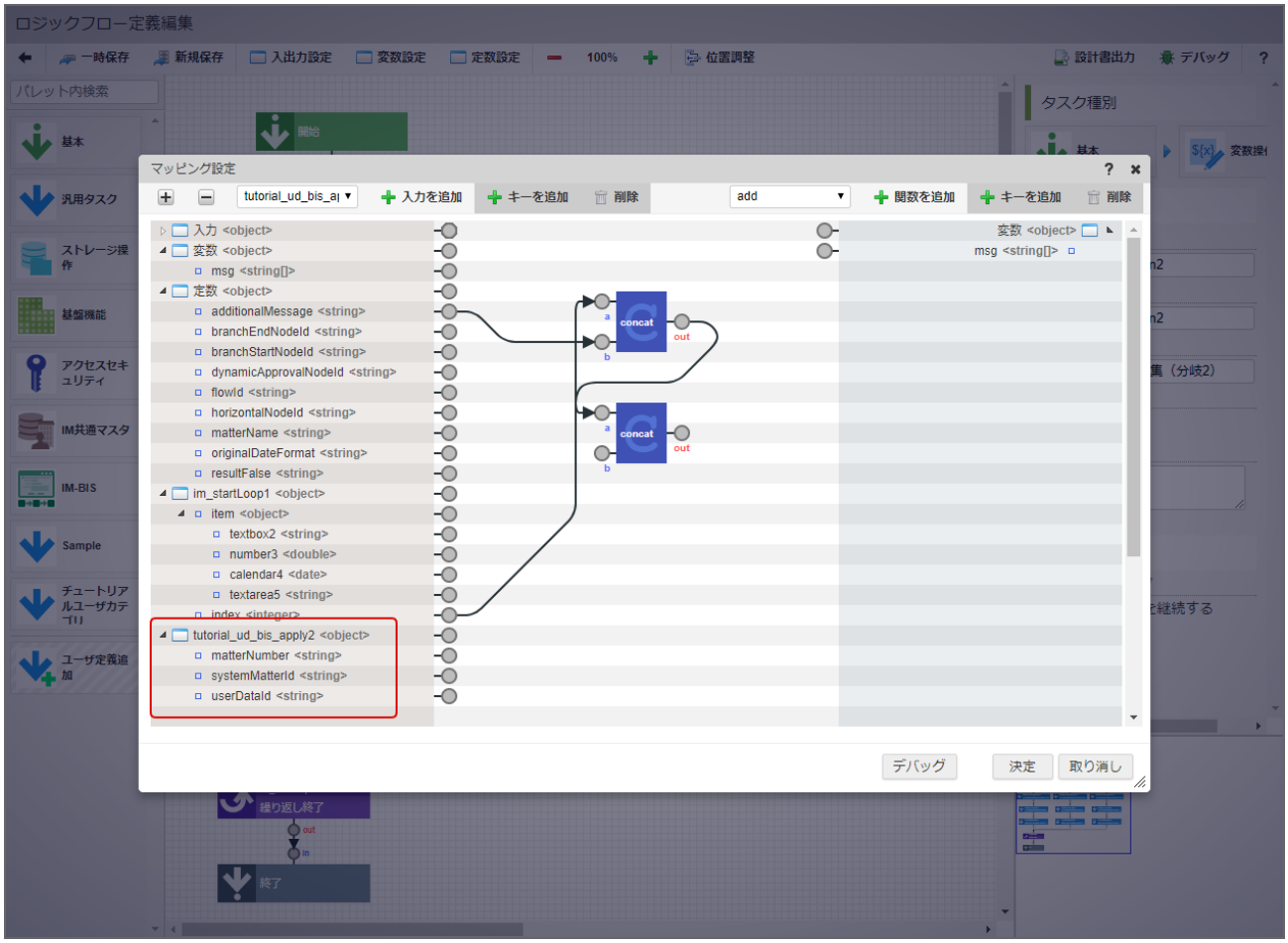

図：入力に追加するエイリアスを選択

17. セレクトボックスの中身が変更されたことを確認し、右側にある「入力を追加」をクリックします。



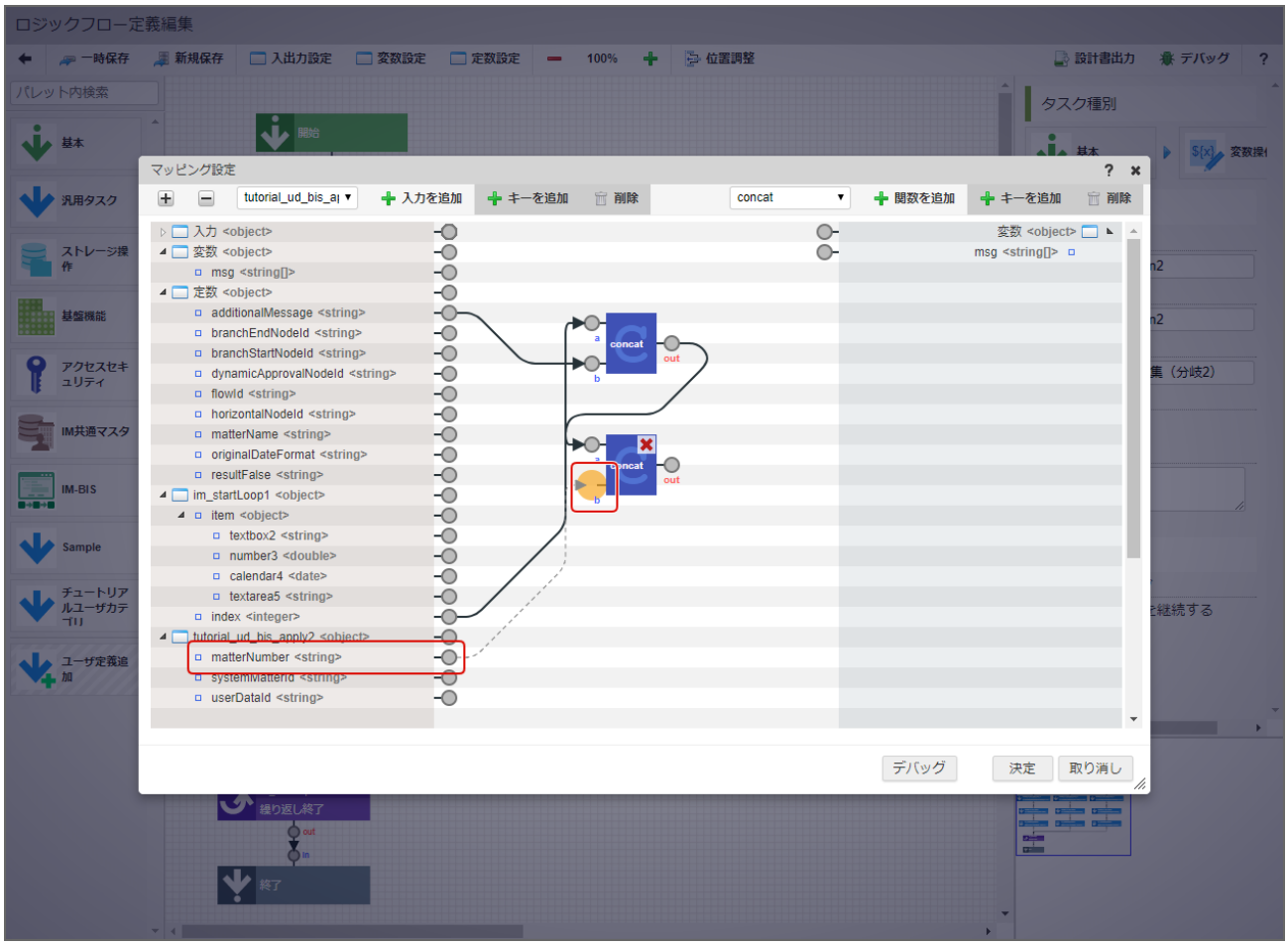

図：「入力を追加」をクリック

18. 入力値として、「ユーザ定義[BIS申請/承認-申請]」タスク（tutorial_ud_bis_apply2<object>）の出力値が追加されました。



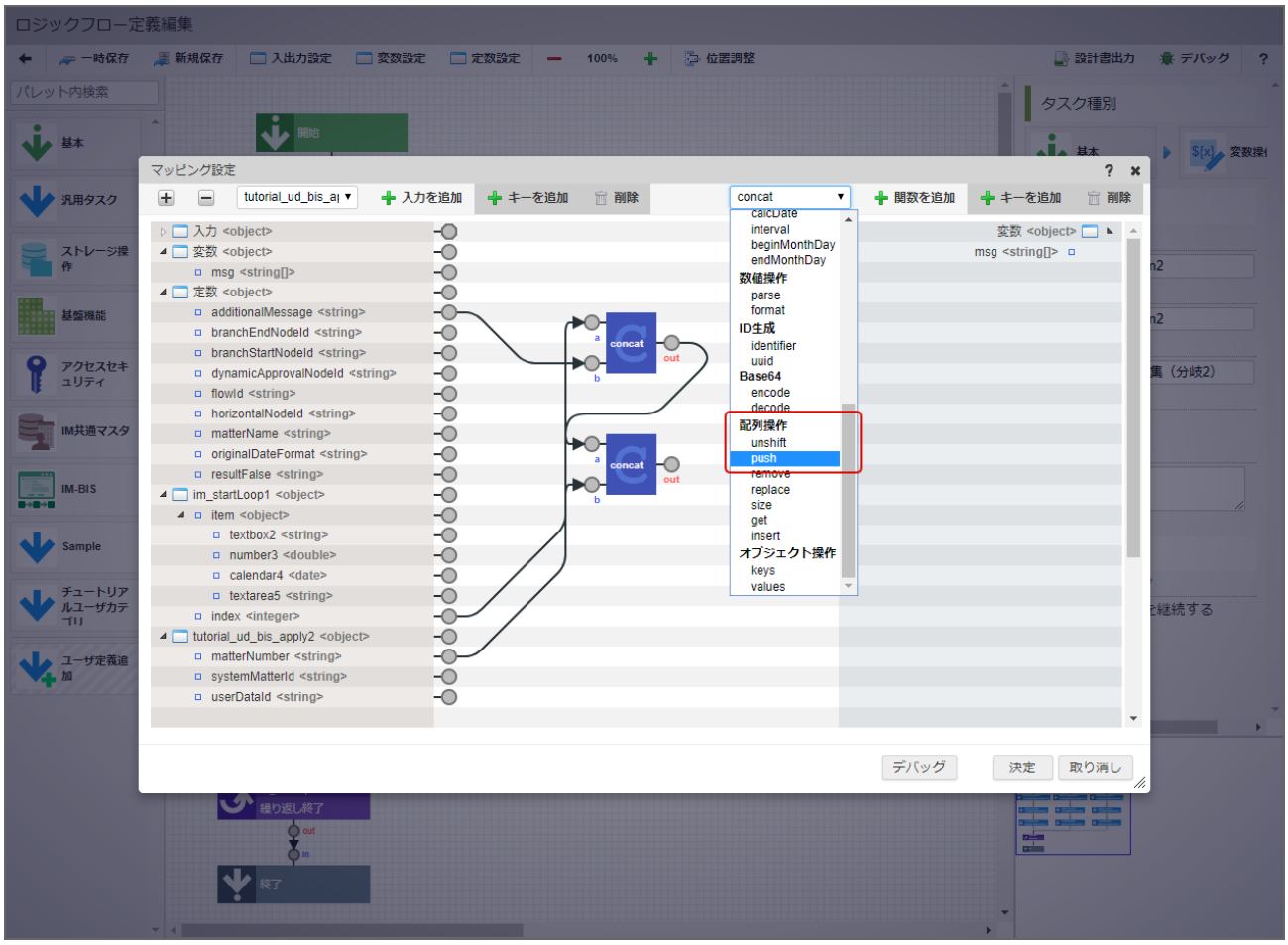

図：入力値の追加（ユーザ定義[BIS申請/承認-申請]）

19. 設定画面左部、「tutorial_ud_bis_apply2<object>」要素の下にある「 matterNumber<string> 」から出ている端子をドラッグし、「concat」関数の左部から出ている端子のうち「 b 」へドロップします。



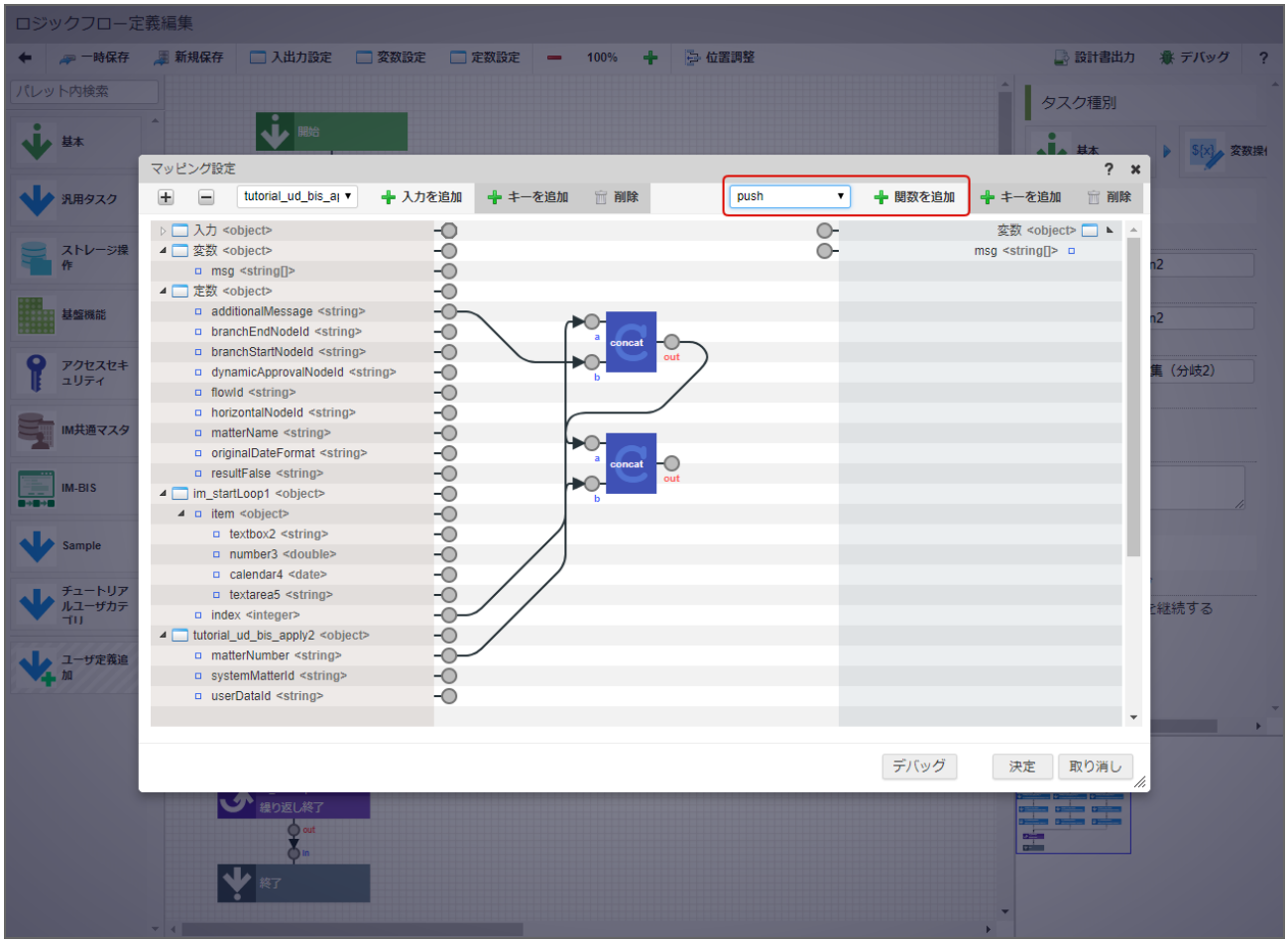

図：関数と入力値（「ユーザ定義[BIS申請/承認-申請]」の出力値）の接続

20. マッピング設定画面上部、ヘッダ内の中央右寄りに位置するセレクトボックスをクリックし、以下の項目を選択します。
    - 配列操作 - push



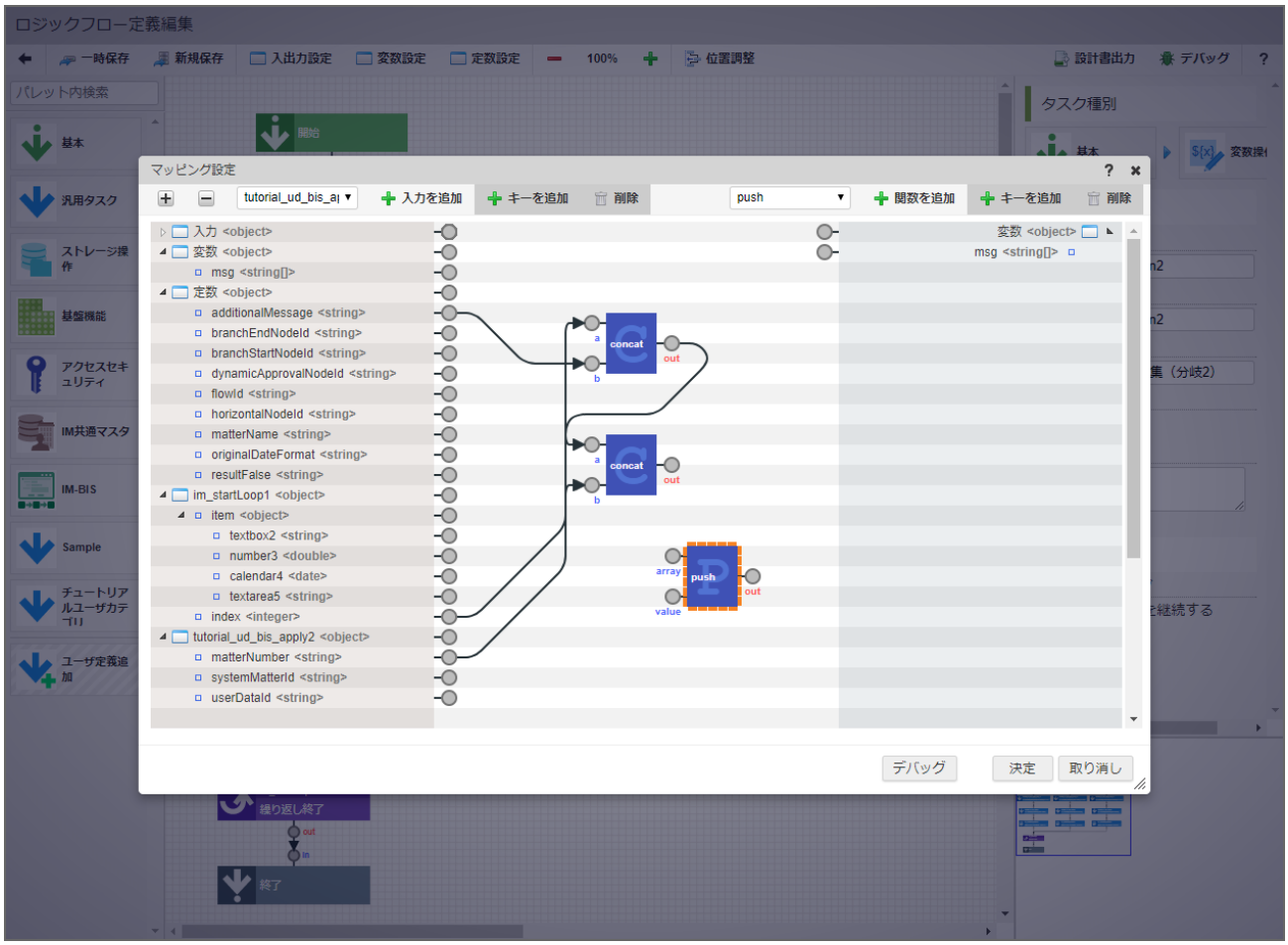
図：マッピング関数「push」の選択

21. セレクトボックスの中身が変更されたことを確認し、右側にある「関数を追加」をクリックします。



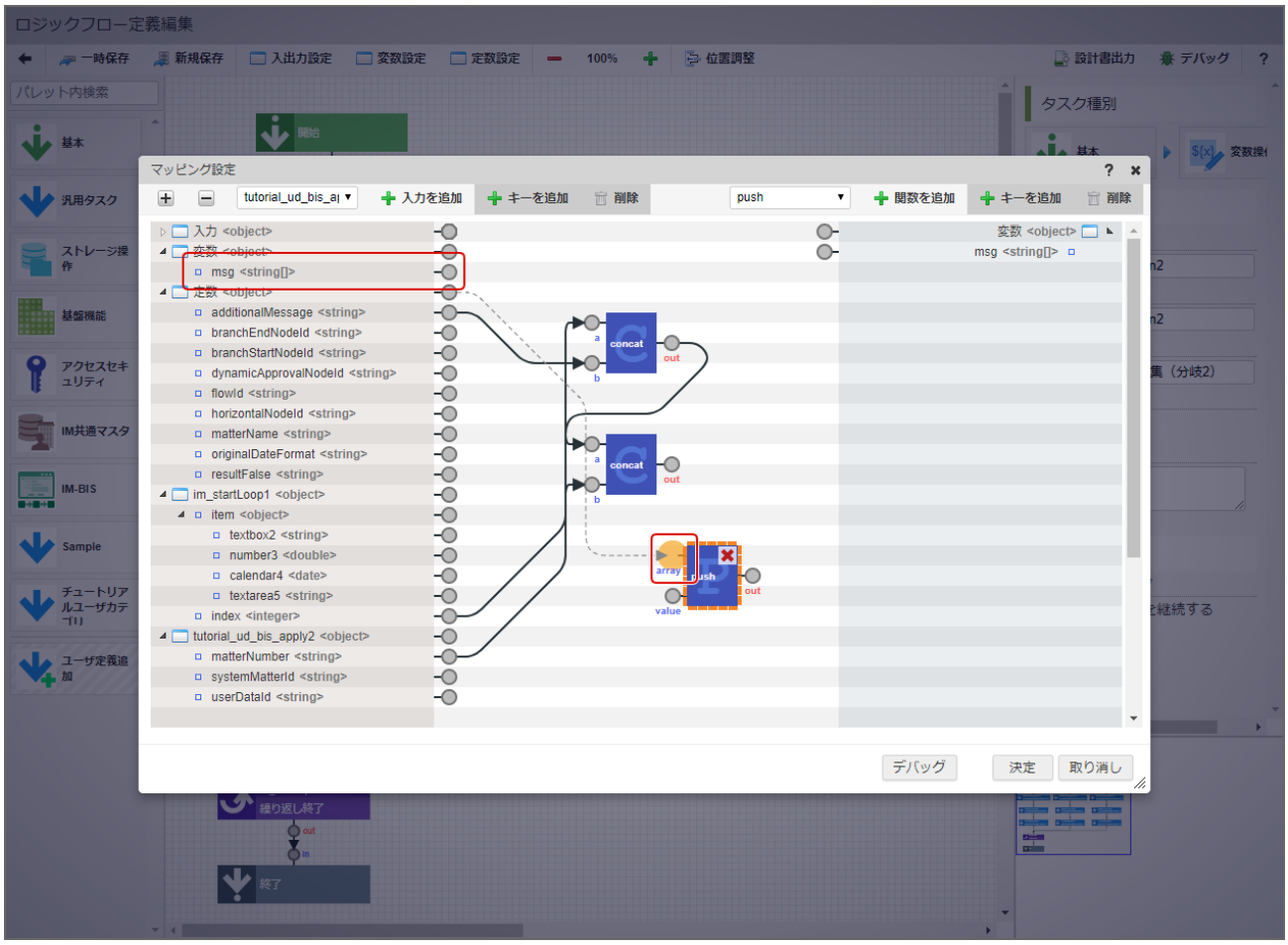

図：「関数を追加」をクリック

22. マッピング関数として、「push」関数が追加されました。



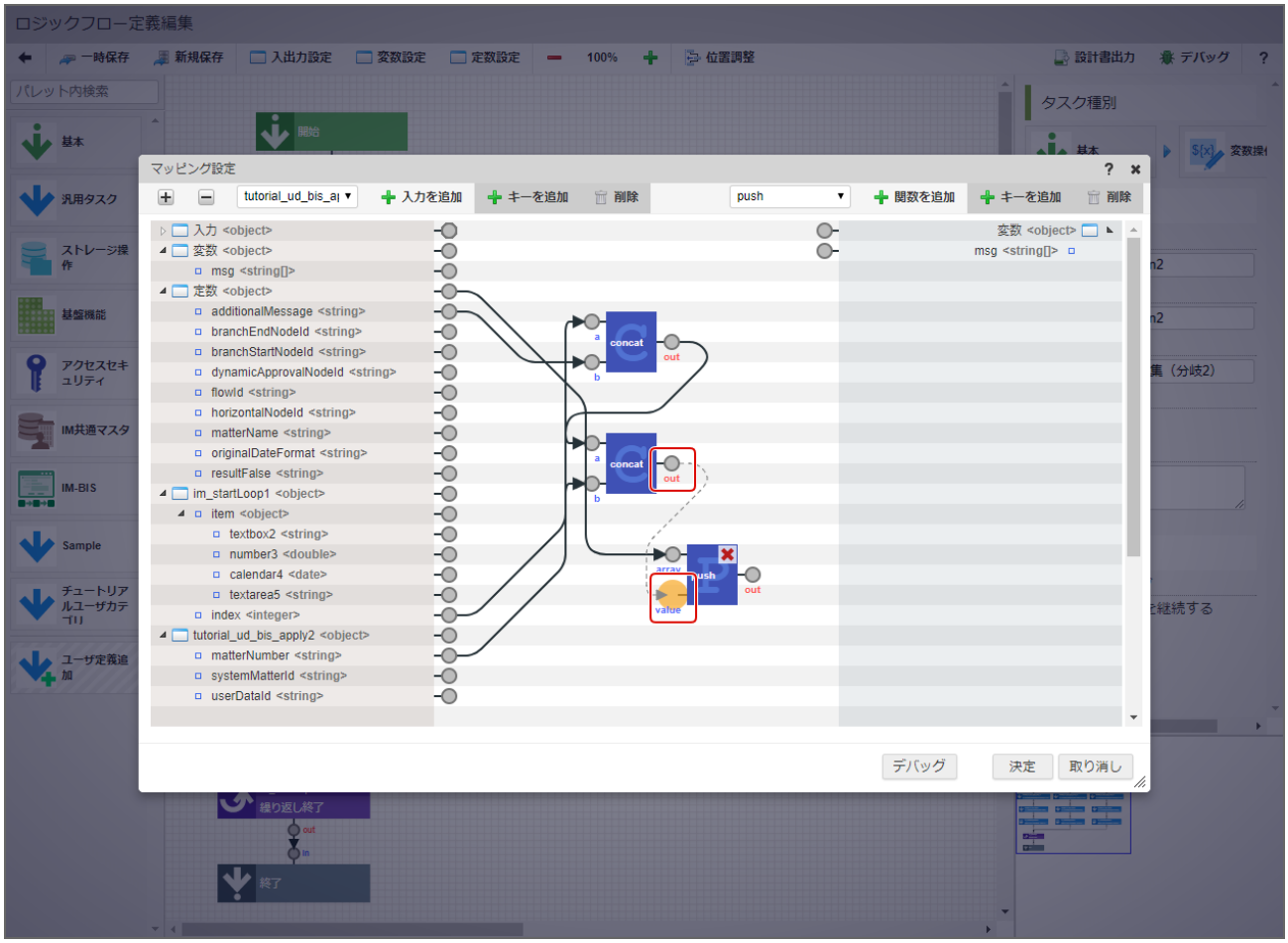

図：関数（push）の追加

23. 「変数<object>」要素の下にある「 msg<string[]> 」から出ている端子をドラッグし、「push」関数の左部から出ている端子のうち「 array 」へドロップします。



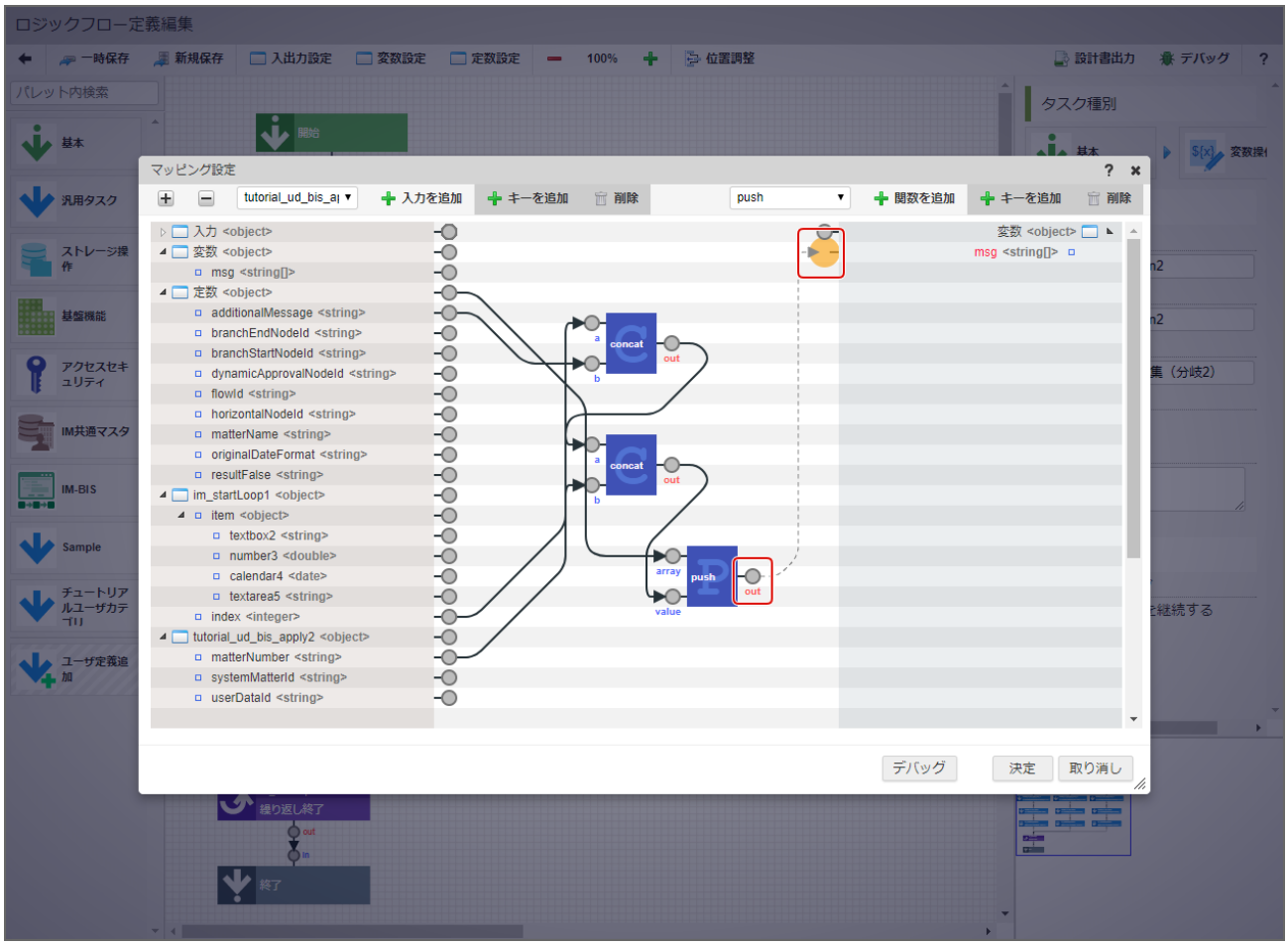
図：関数と入力値の接続

24. 後に配置した関数「concat」から出ている端子をドラッグし、関数「push」の左部から出ている端子のうち「value」へドロップします。




図：関数（concat）と関数（push）の接続

25. 関数「push」から出ている端子をドラッグし、設定画面右部、「変数<object>」要素の下にある「msg<string[]>」から出ている端子にドロップします。

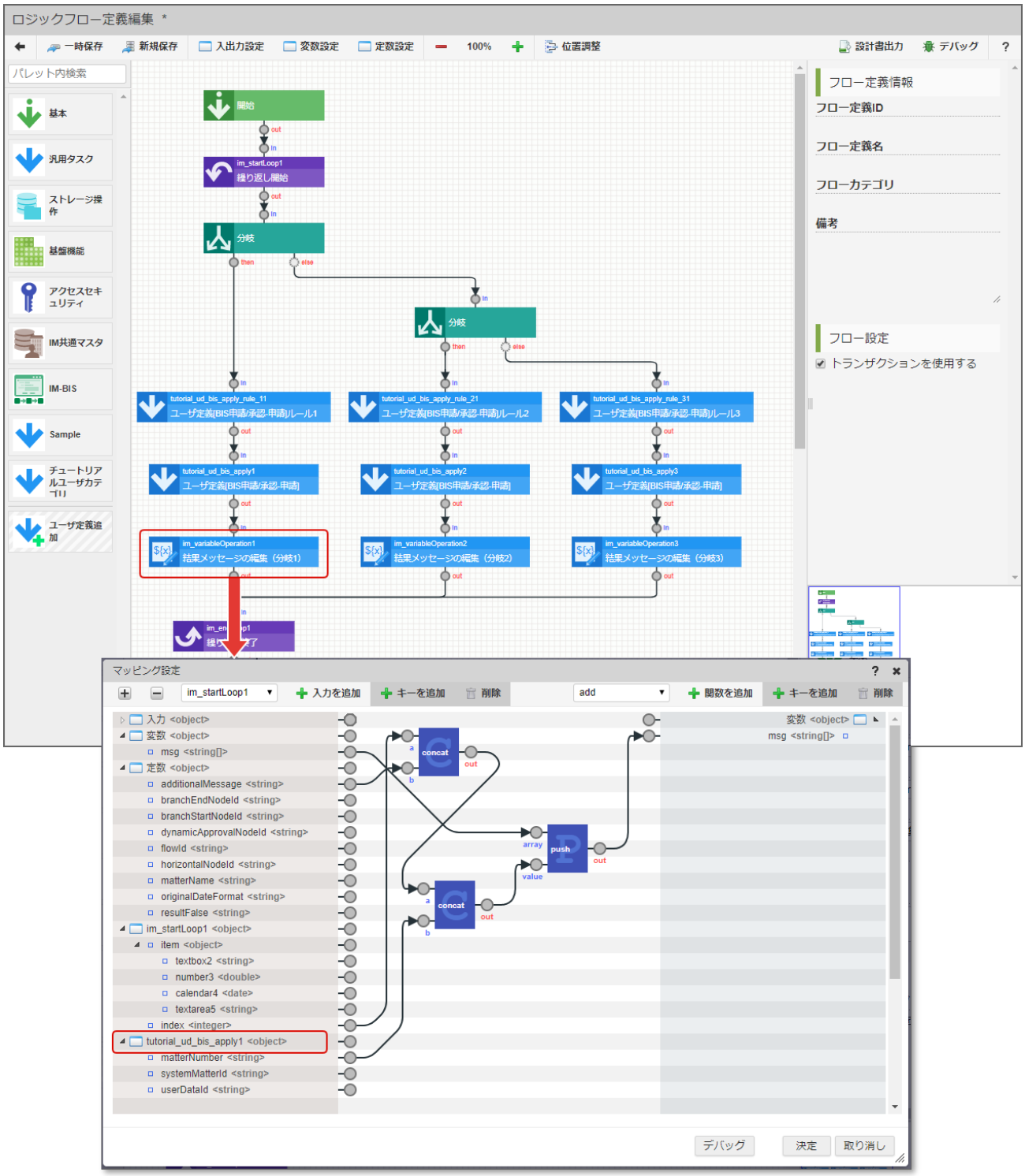



図：関数（push）と変数の接続

26. ここまでの手順で、「変数操作」制御要素を利用したメッセージの編集に必要なマッピングが設定できました。
    設定画面右下の決定をクリックし、「変数操作」制御要素のマッピング設定を終了します。

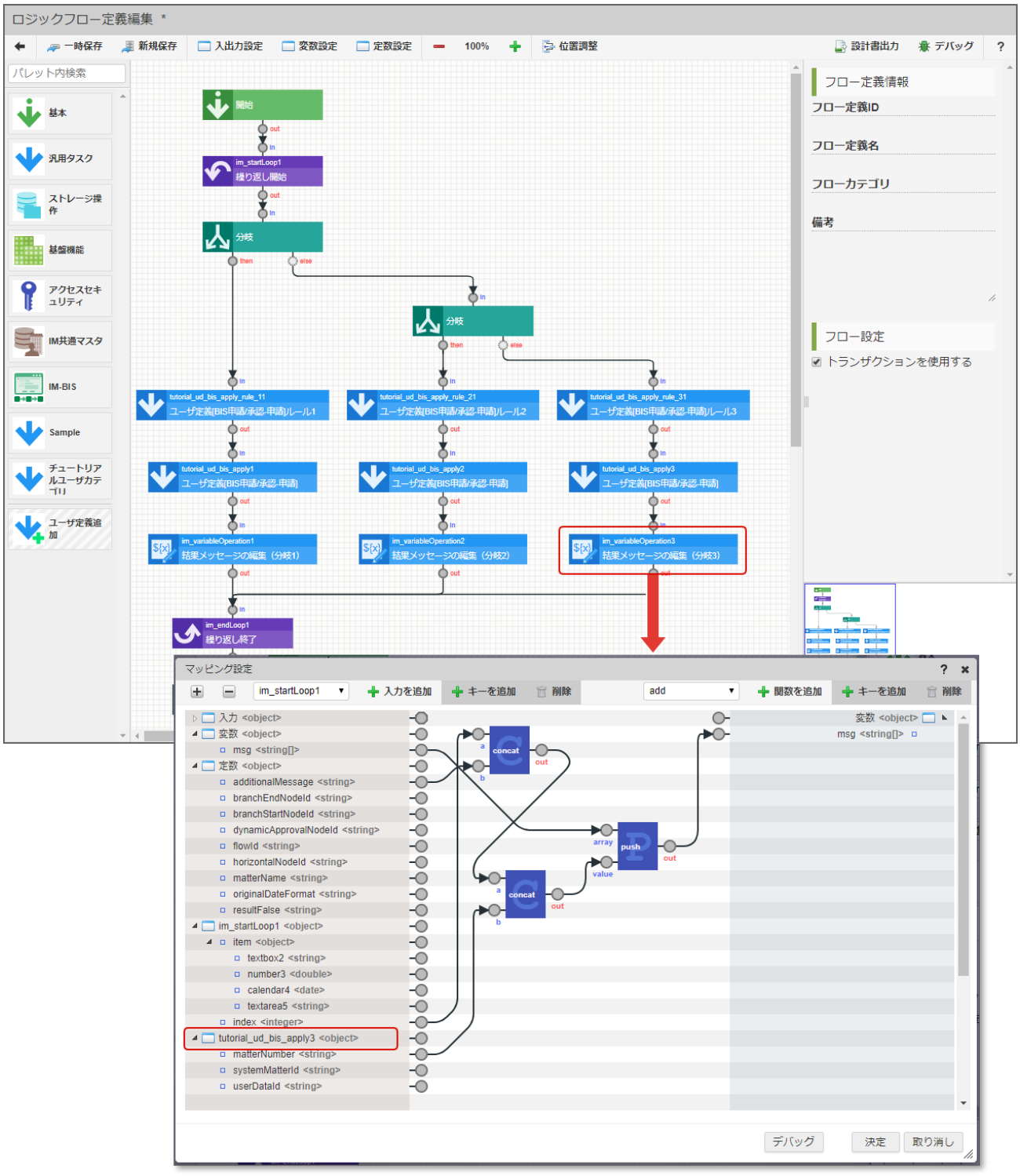
残りの左のルートと右のルート上の「変数操作」制御要素に対し、同様の手順でマッピングを設定します。
このマッピングでは、案件番号の参照元のエイリアスがそれぞれ異なっている点に注意してください。

- 左のルート上の「変数操作」制御要素のマッピング




図： 左のルート上の「変数操作」制御要素のマッピング

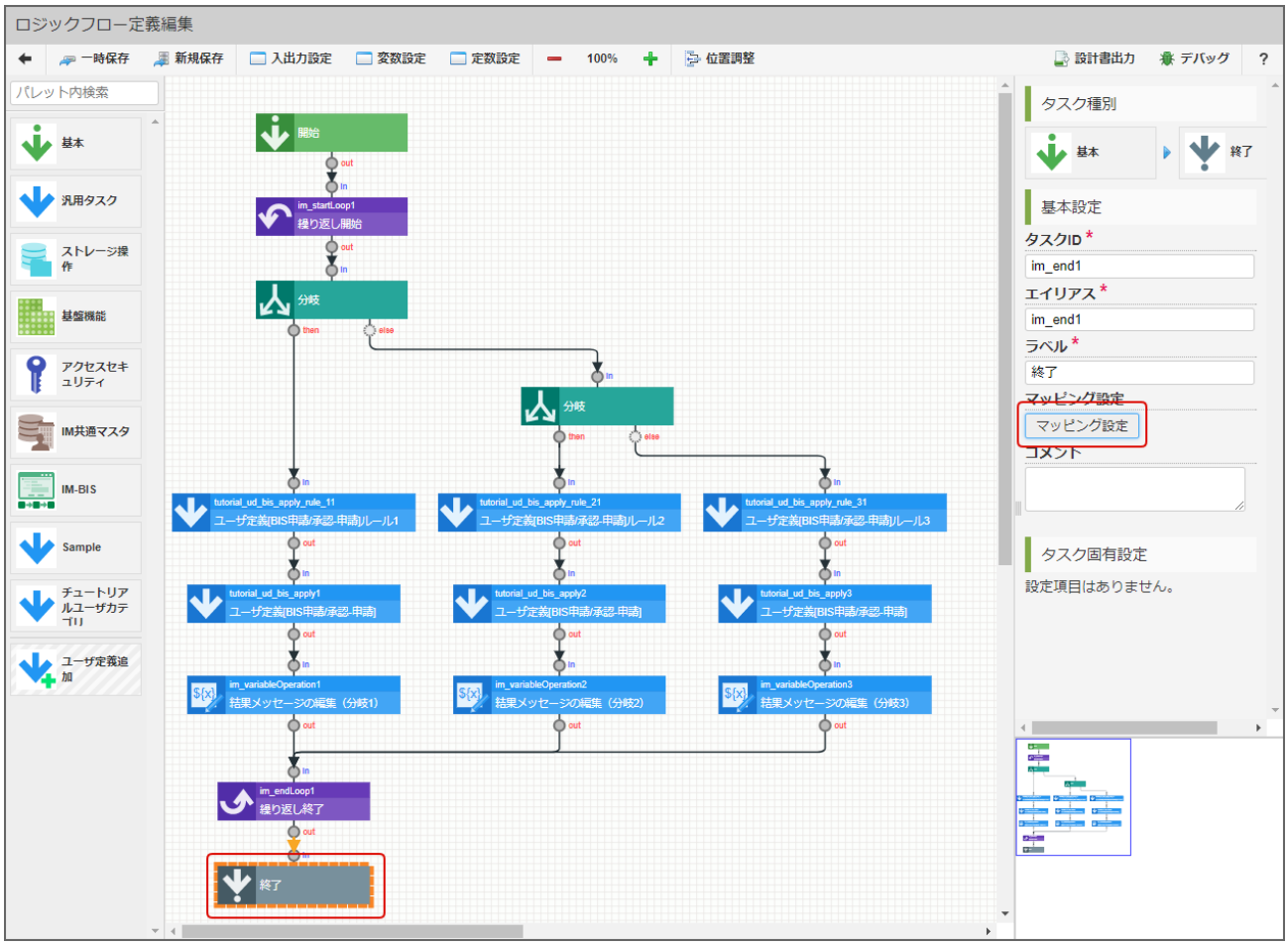
- 右のルート上の「変数操作」制御要素のマッピング




図： 右のルート上の「変数操作」制御要素のマッピング

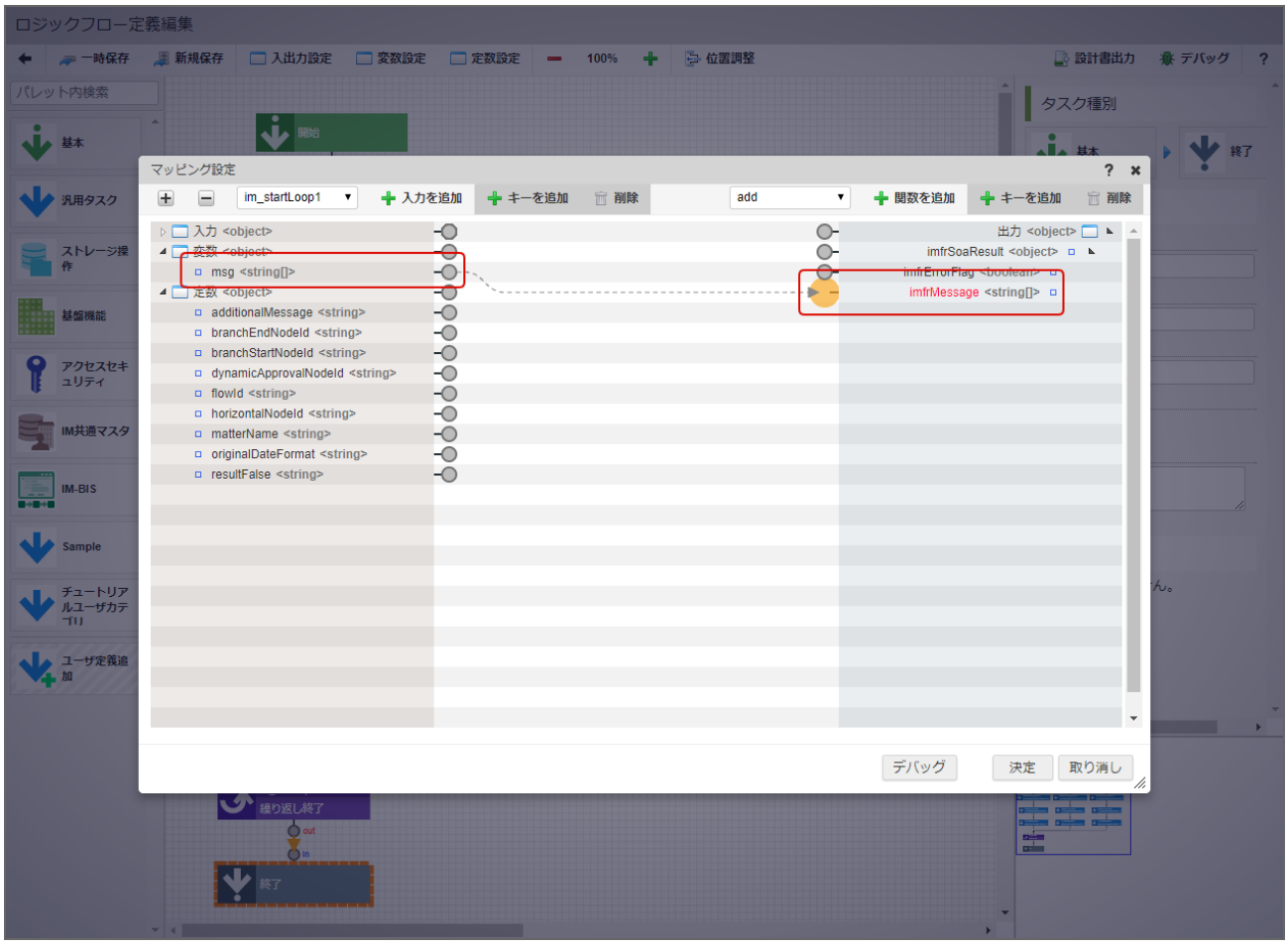
最後に「終了」制御要素で、編集したメッセージを出力値にマッピングします。

1. 「終了」制御要素のマッピング設定画面を開きます。



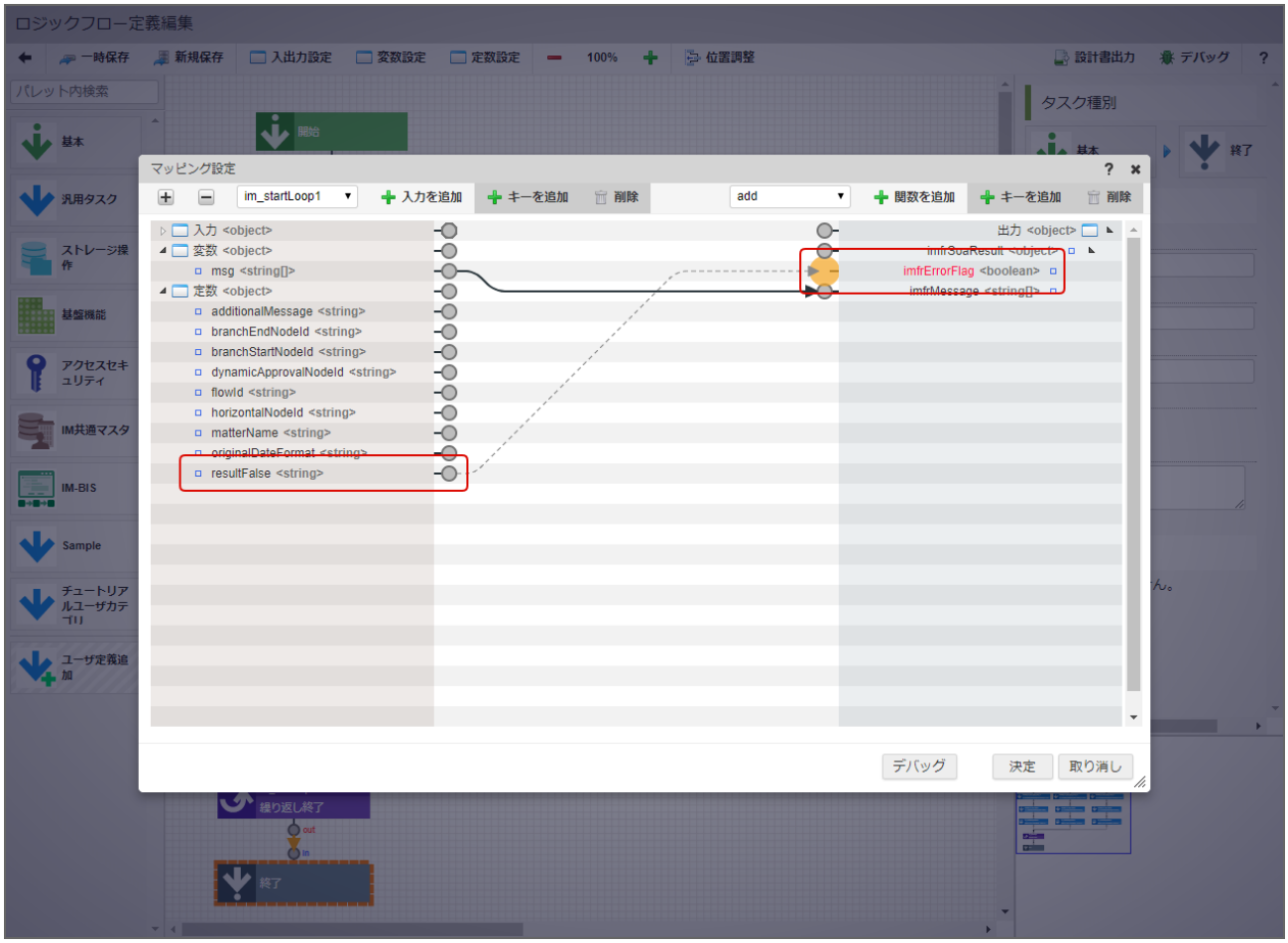

図：「終了」制御要素のマッピング設定をクリック

2. 設定画面左部、「変数<object>」要素の下にある「 msg<string[]> 」から出ている端子をドラッグし、設定画面右部、「 imfrMessage<string[]> 」から出ている端子にドロップします。



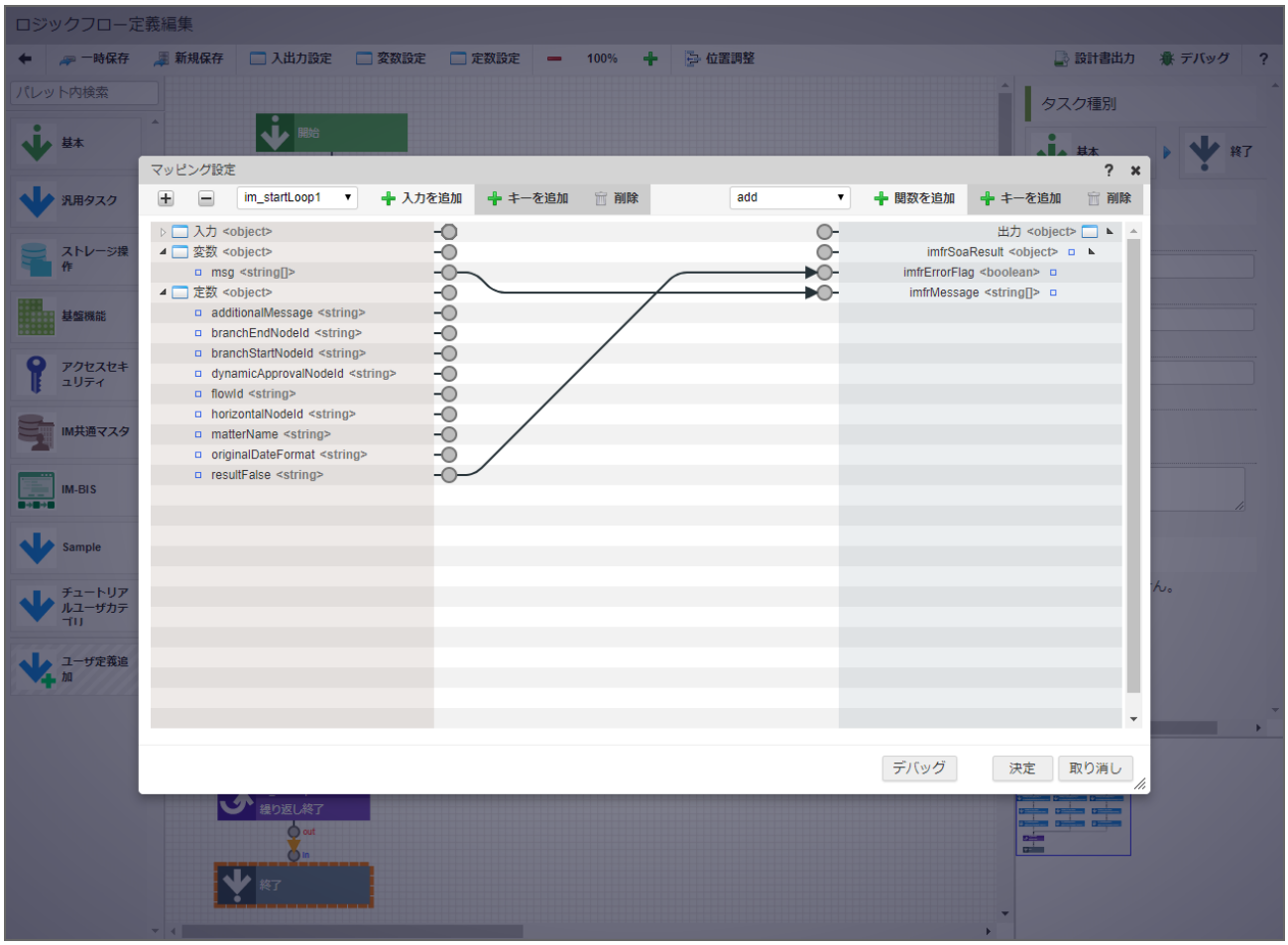
図： メッセージ情報のマッピング

3. 設定画面左部、「定数<object>」要素の下にある「 resultFalse<string> 」から出ている端子をドラッグし、設定画面右部、「 imfrErrorFlag<boolean> 」から出ている端子にドロップします。




図： エラーフラグのマッピング

4. ここまでの手順で、「終了」制御要素を利用したメッセージの編集に必要なマッピングが設定できました。
設定画面右下の 決定 をクリックし、「終了」制御要素のマッピング設定を終了します。

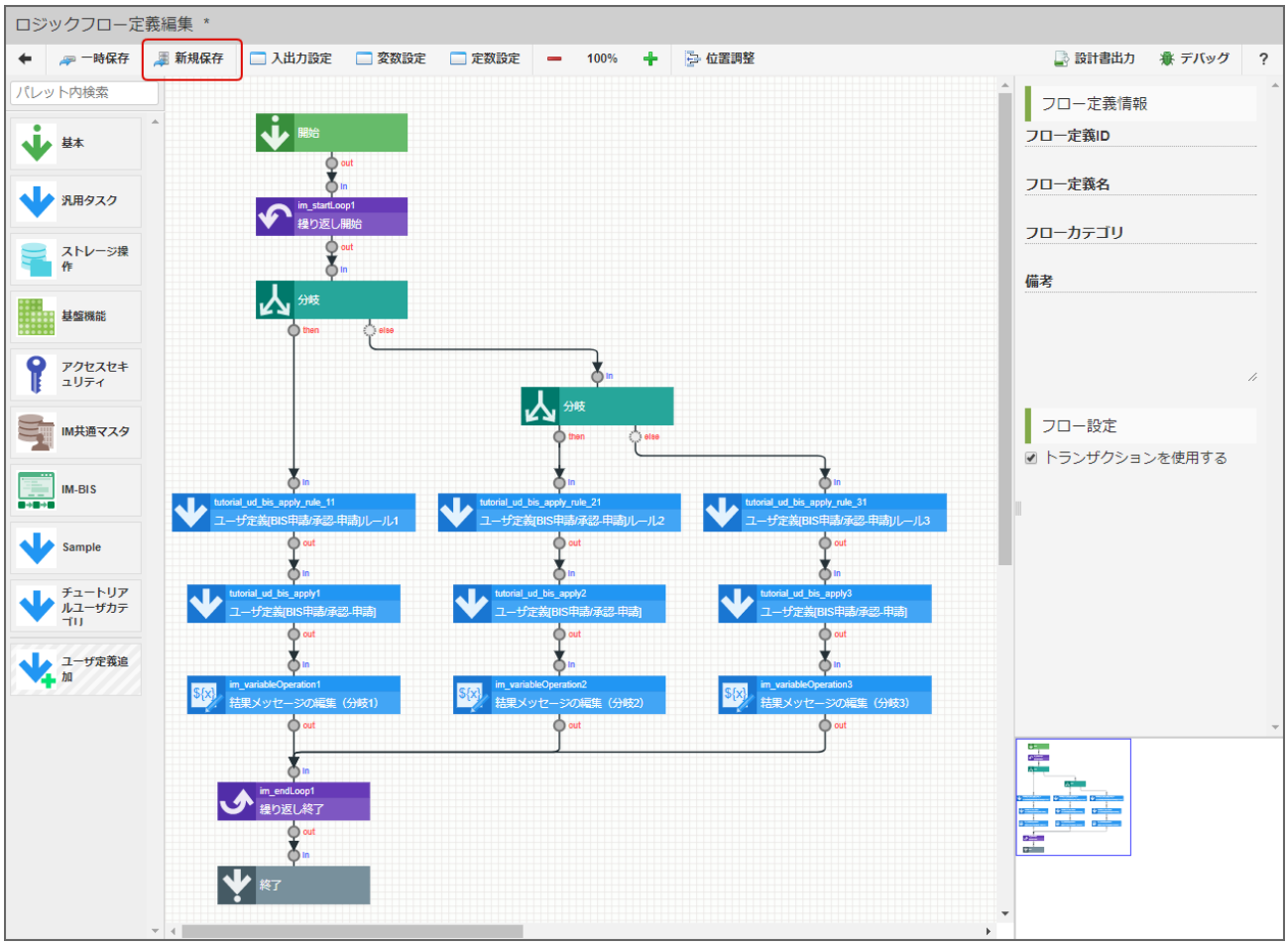



図：「終了」制御要素のマッピング設定

保存する

ロジックフローを作成する最後の手順として、これまで作成してきた内容を保存します。
ロジックフローの保存方法の詳細は *保存する* を参照してください。

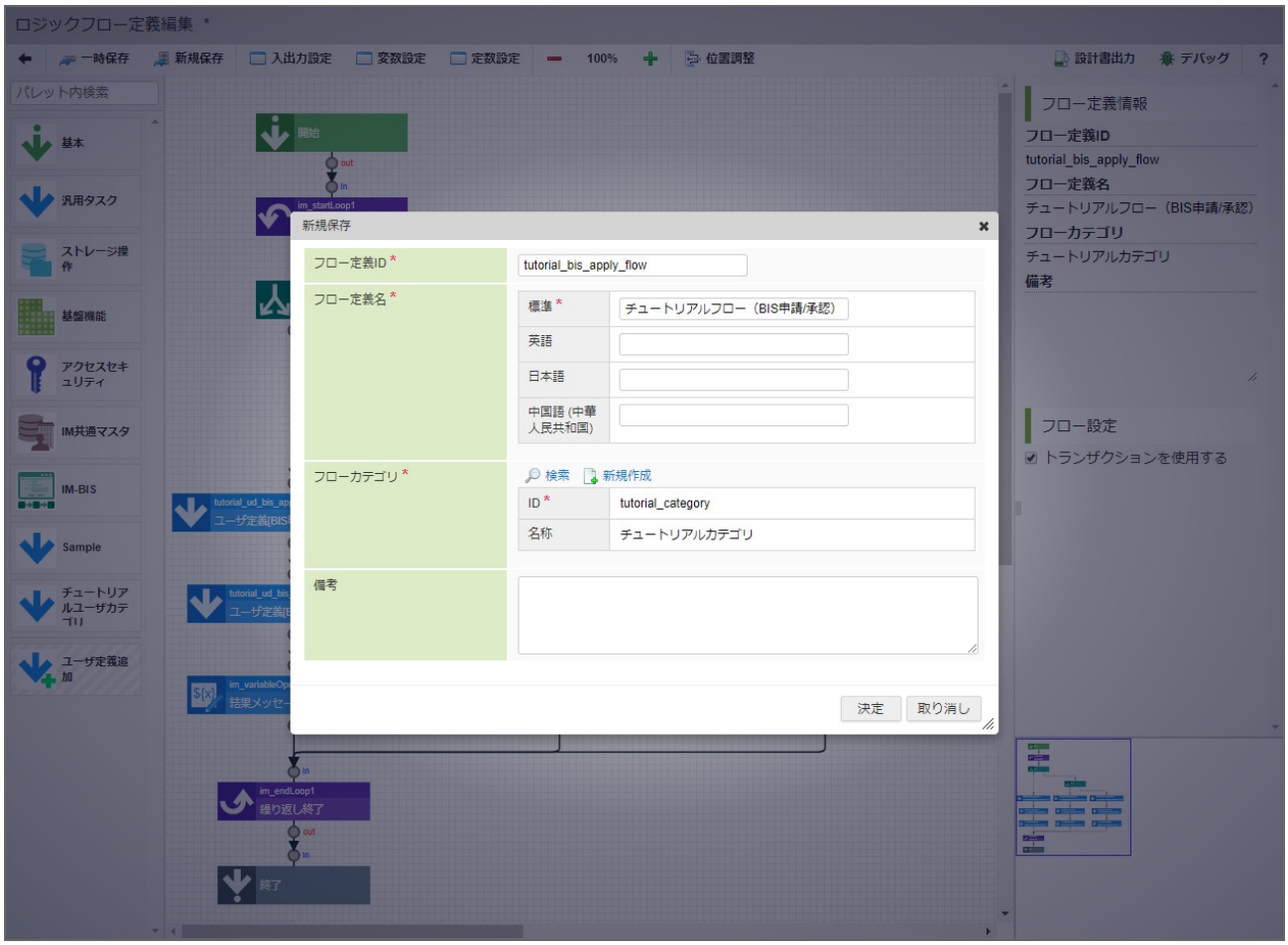
1. ロジックフロー定義編集画面上部、ヘッダ内の「新規保存」をクリックします。




図： ロジックフロー定義編集画面ヘッダ

2. 各項目に以下の値を入力し、新規保存画面右下の 決定 をクリックします。

| 設定項目 | 設定値 |
|---|---|
| フロー定義ID | tutorial_bis_apply_flow |
| フロー定義名 | ▪ 標準 - チュートリアルフロー（BIS申請/承認）<br>▪ 日本語、英語、中国語（中華人民共和国） - 入力なし |
| フローカテゴリ | ▪ ID - tutorial_category<br>▪ 名称 - チュートリアルカテゴリ |




図： 保存情報の定義

以上で、ロジックフローの保存が完了しました。

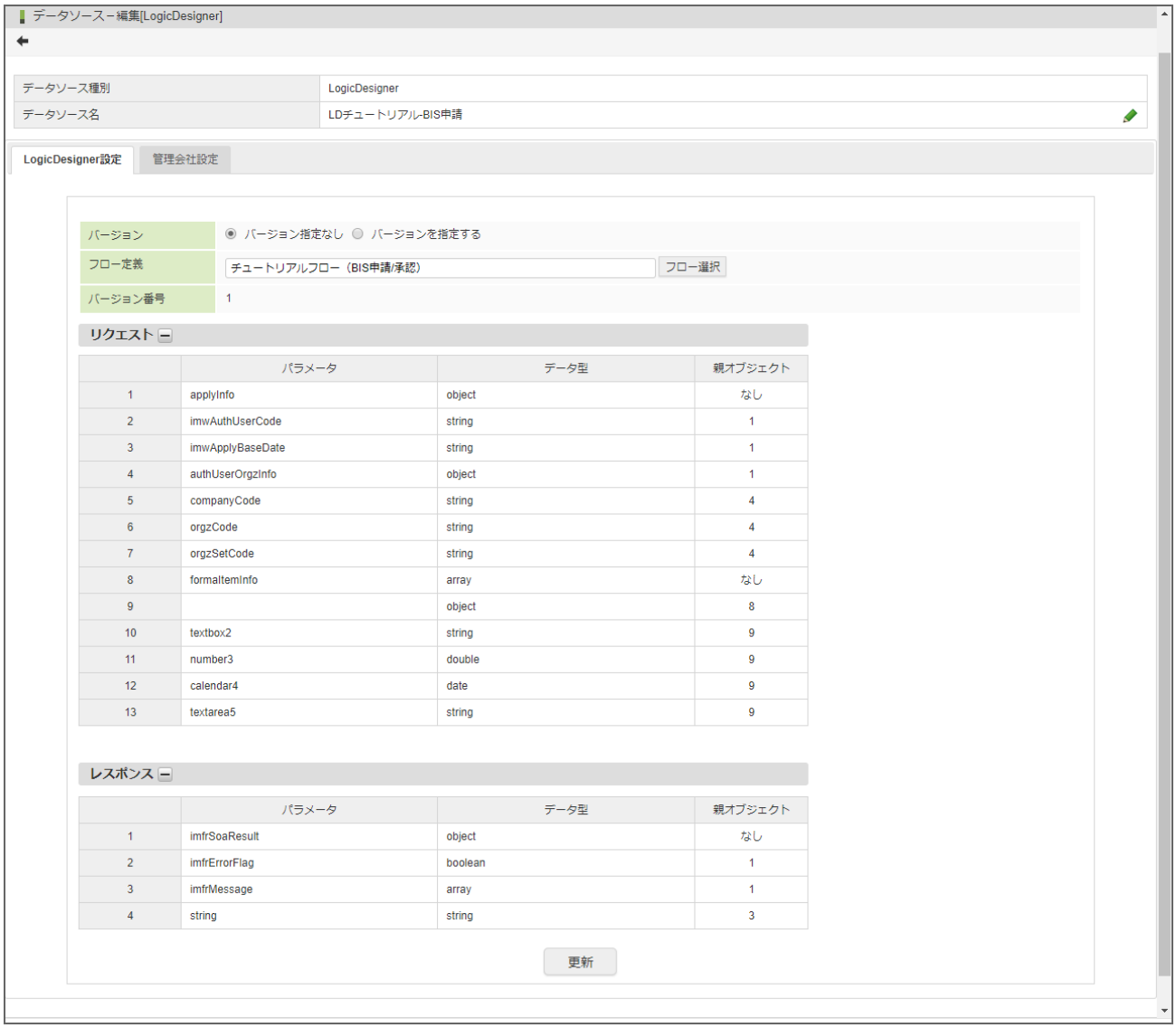
## BISのアクションイベントを設定する

ロジックフローをBISのワークフロー画面から実行するために、アクションイベントにロジックフローを設定します。

データソース定義を登録する

作成したロジックフローをデータソース定義（LogicDesigner）に登録します。
データソース定義の登録手順の詳細は「 IM-BIS システム管理者操作ガイド 」-「 データソース定義を設定する 」を参照してください。

- 以下は設定例です。

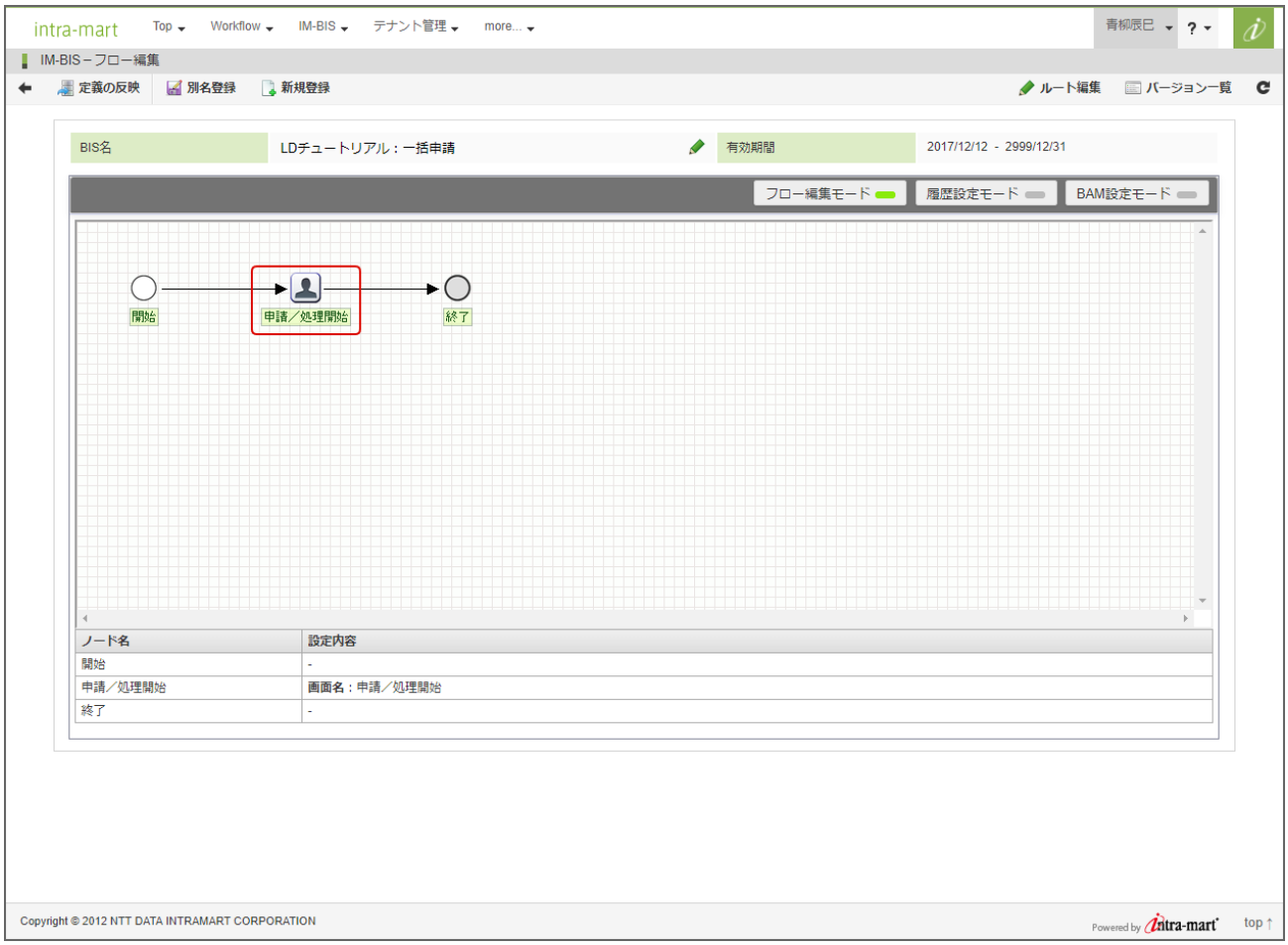



図： LogicDesignerのデータソース定義の設定

アクションイベントに外部連携を設定する

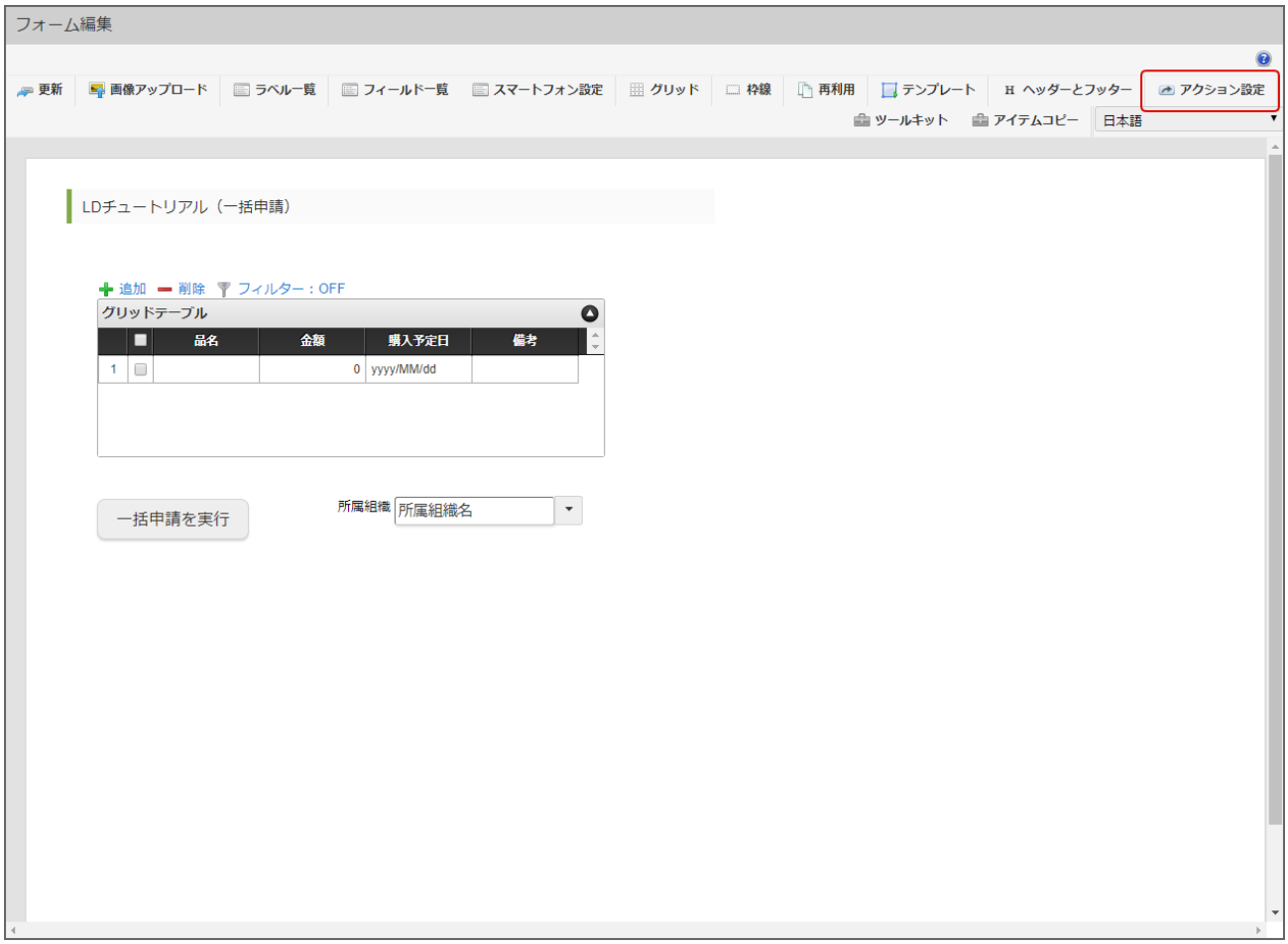
登録したデータソース定義を利用して、BISの画面上のボタンがクリックされたタイミングでロジックフローによる申請が実行されるように設定します。

1. 以下のBIS定義のフロー編集画面を開きます。
   - BISフロー - LDチュートリアル：一括申請
2. 「申請／処理開始」をダブルクリックして、フォーム編集画面を開きます。



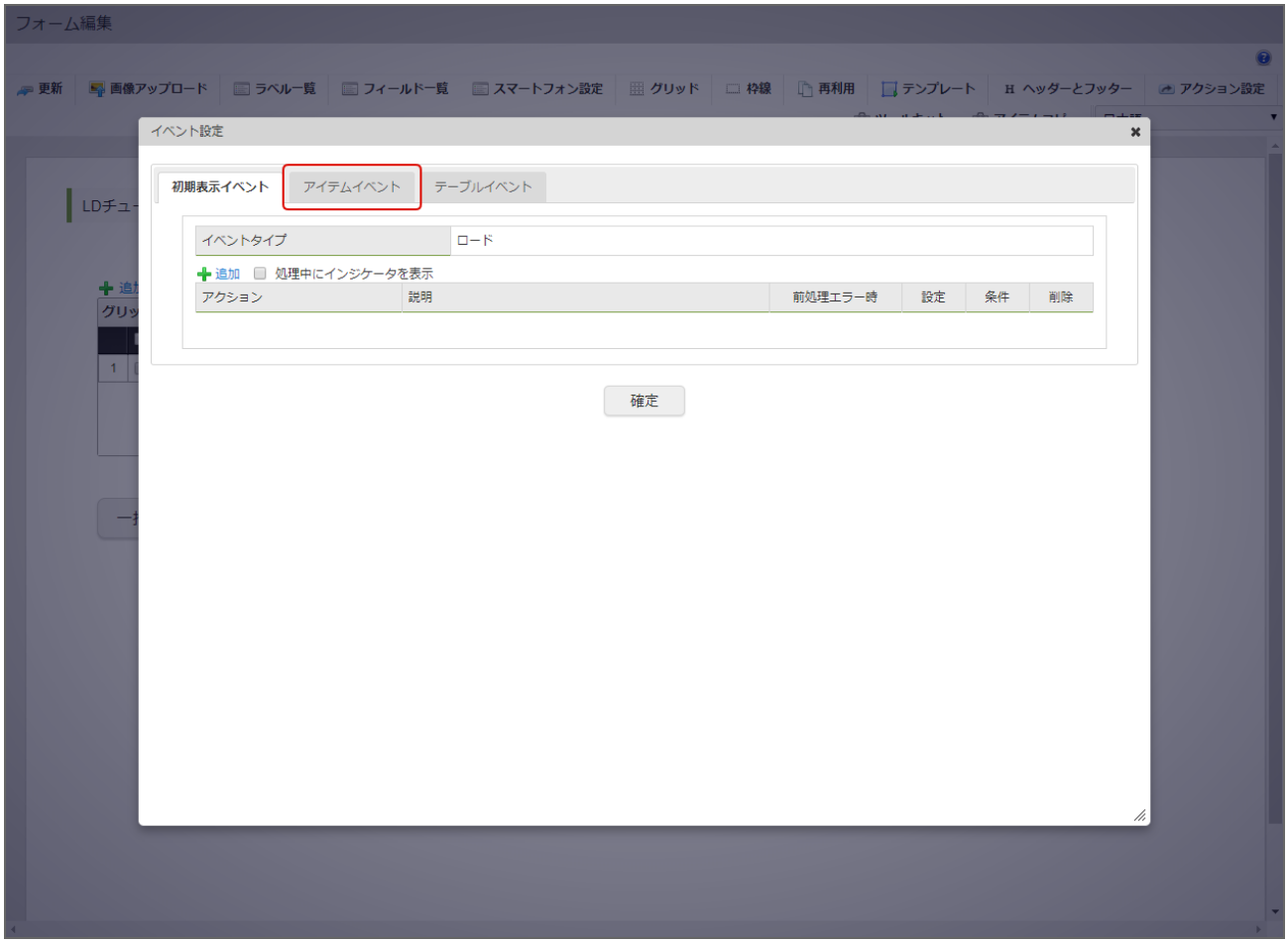

図：フロー編集

3. 画面右上の「アクション設定」をクリックします。



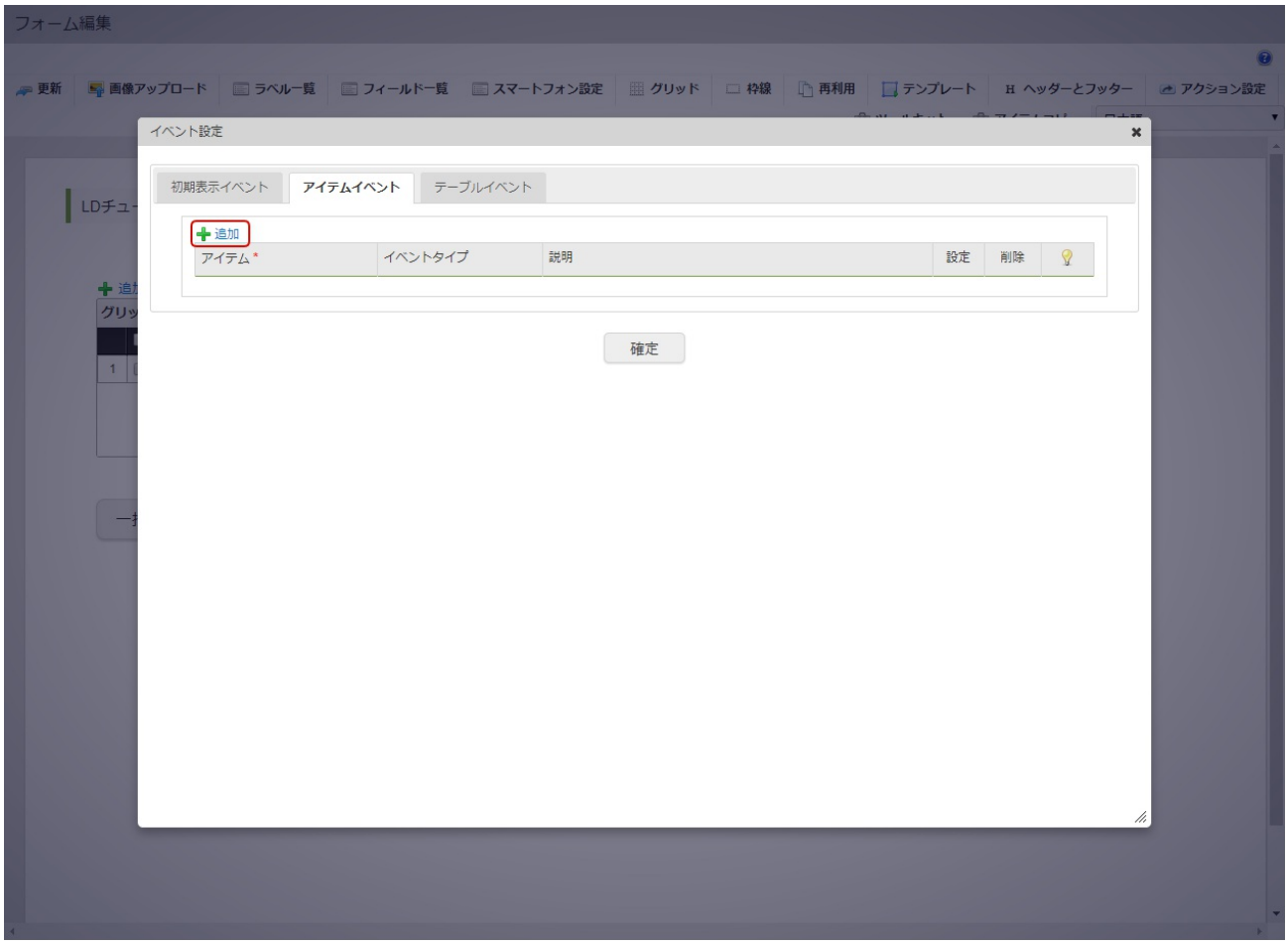

図：フォーム編集

4. 「アイテムイベント」タブに切り替えます。




図：イベント設定

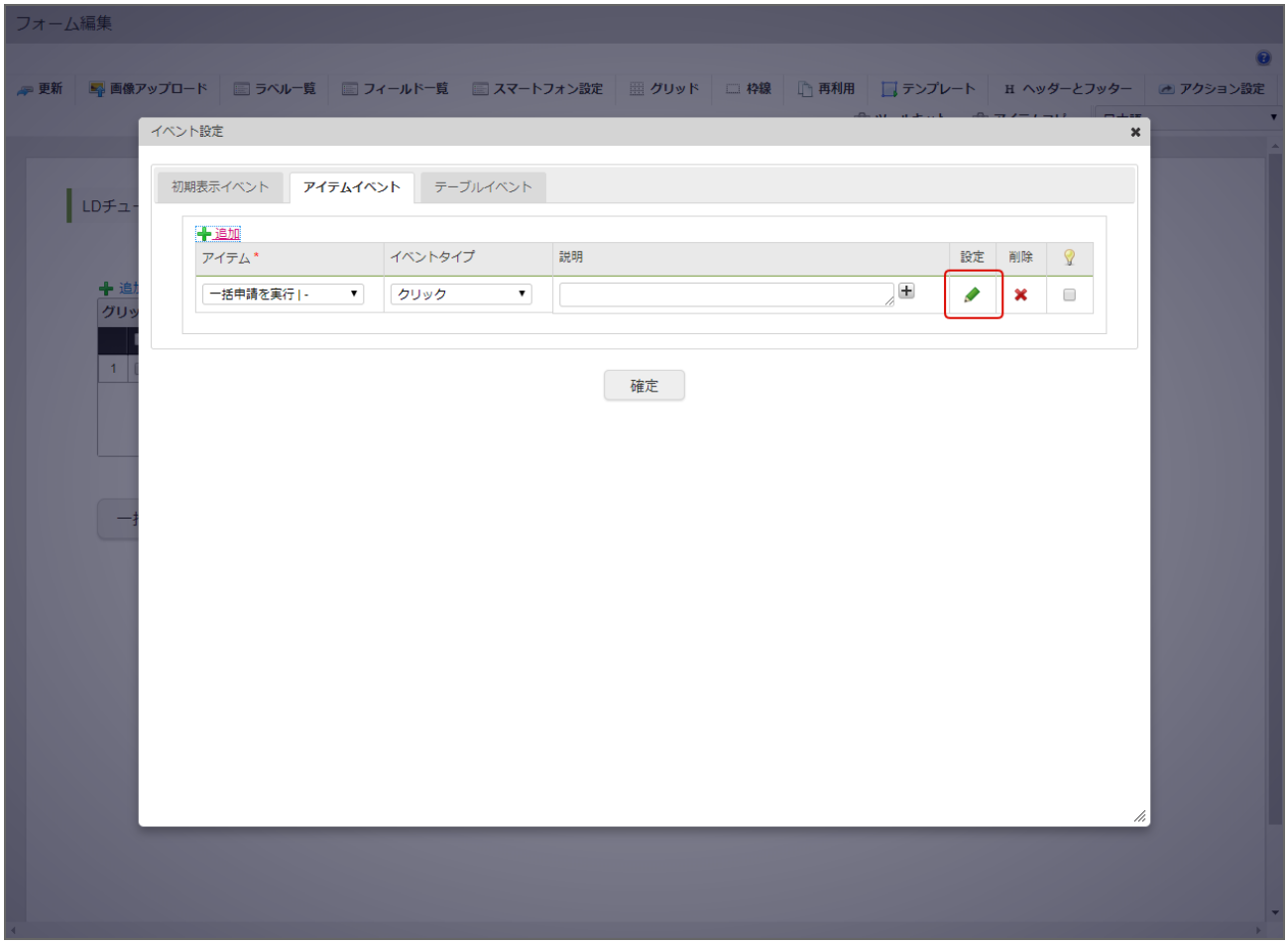
5. 「**+追加**」をクリックして、1行追加します。



図：アクションイベントの追加

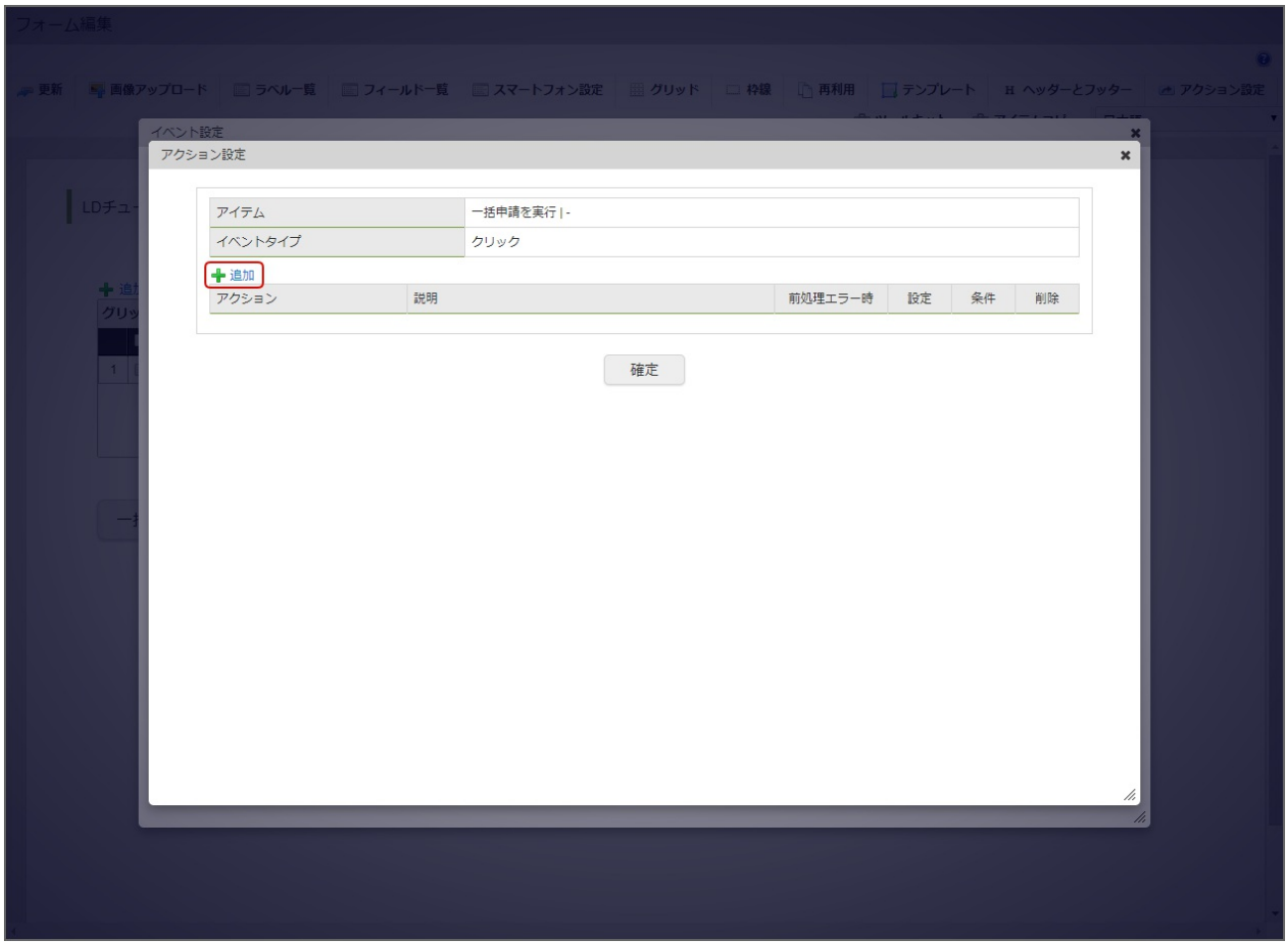
6. 以下のとおりに入力し、「設定」をクリックして「アクション設定」を開きます。

| 設定項目 | 設定値 |
| --- | --- |
| アイテム | 一括申請を実行\|- |
| イベントタイプ | クリック |




図：アイテムイベントの登録

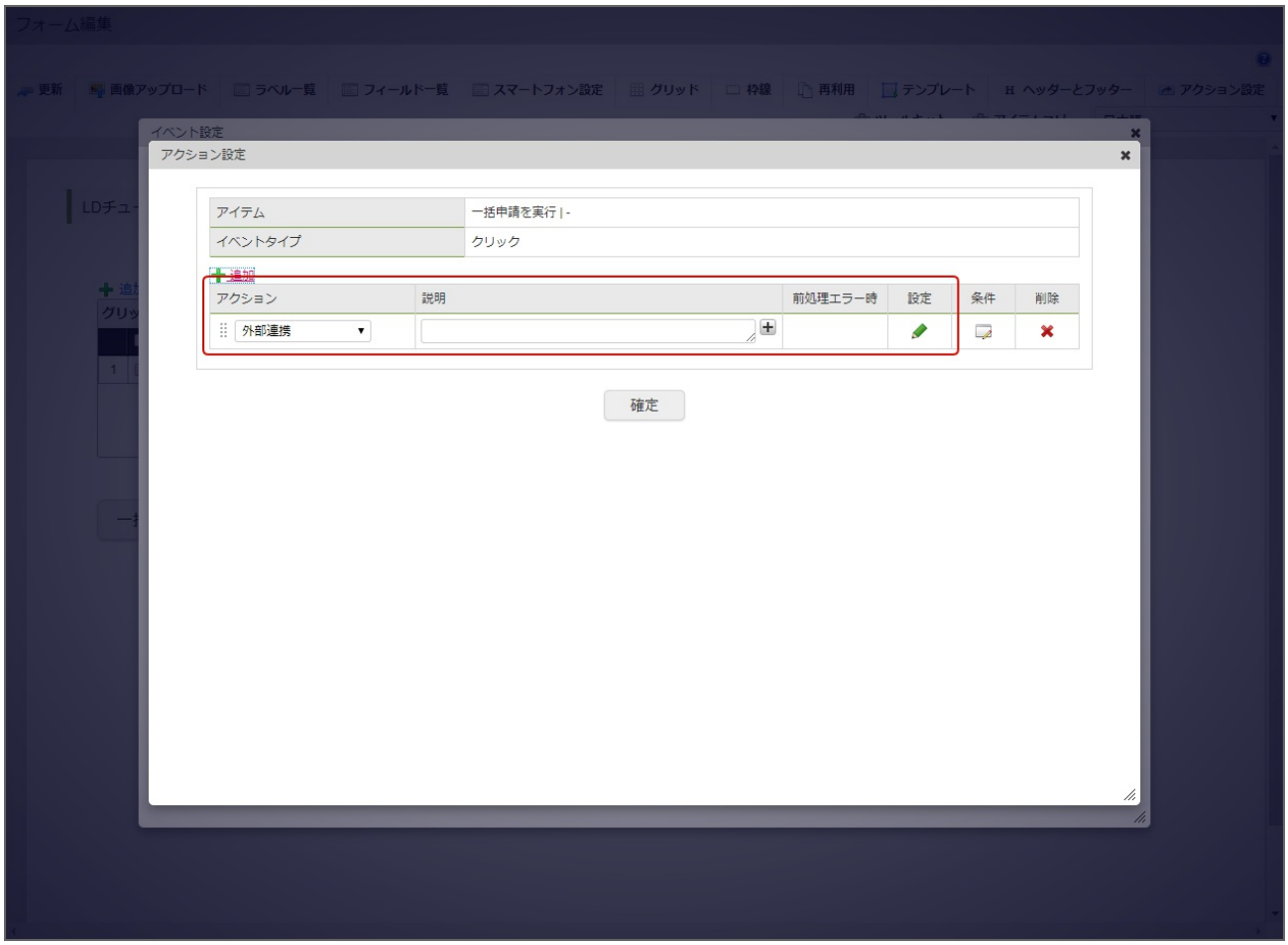
7. 「**+追加**」をクリックして、1行追加します。




図：アクションの追加

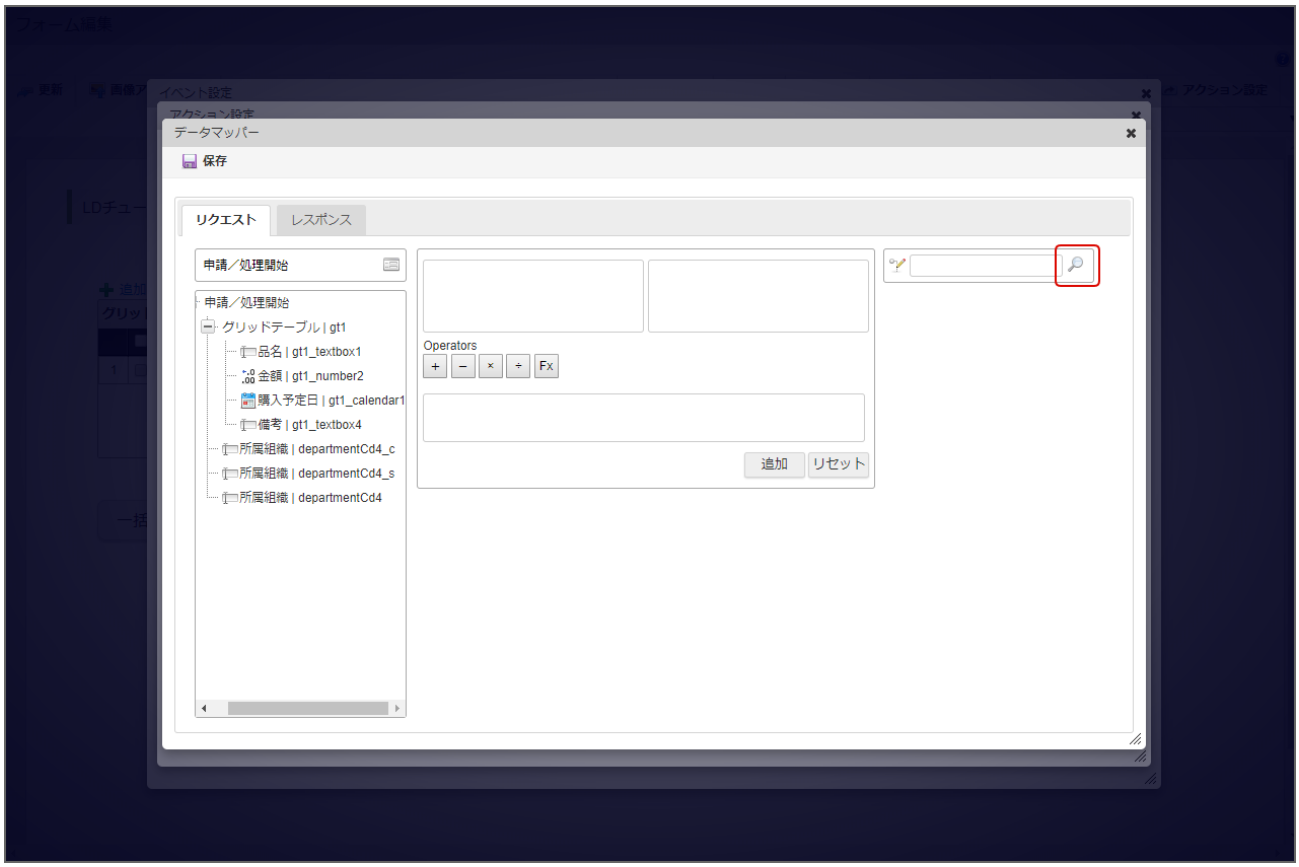
8. 以下のとおりに入力し、「設定」をクリックして「データマッパー」画面を開きます。

| 設定項目 | 設定値 |
| --- | --- |
| アクション | 外部連携 |



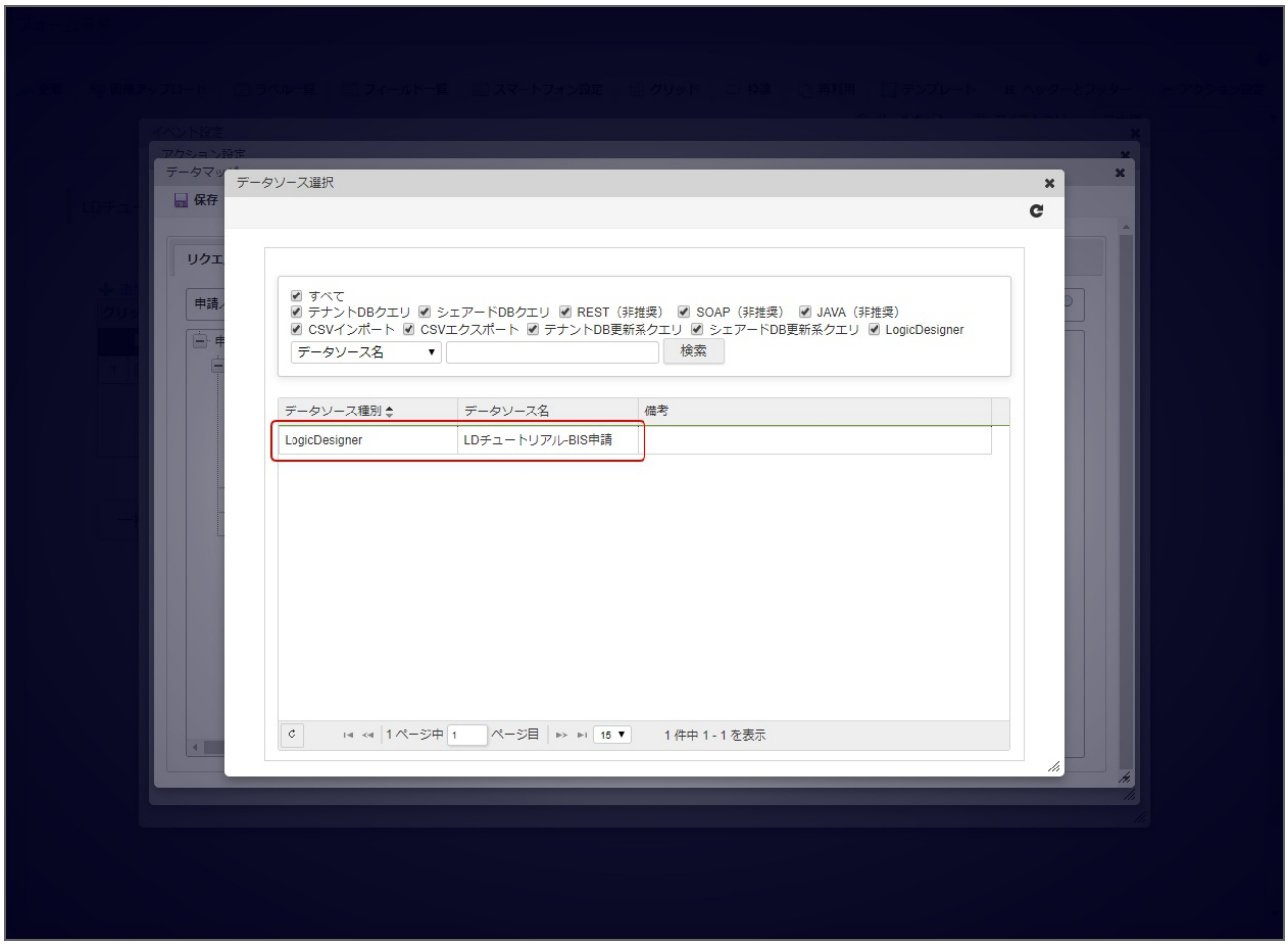

図：アクション設定の登録

9. 右の「検索」をクリックして「データソース選択」画面を開きます。



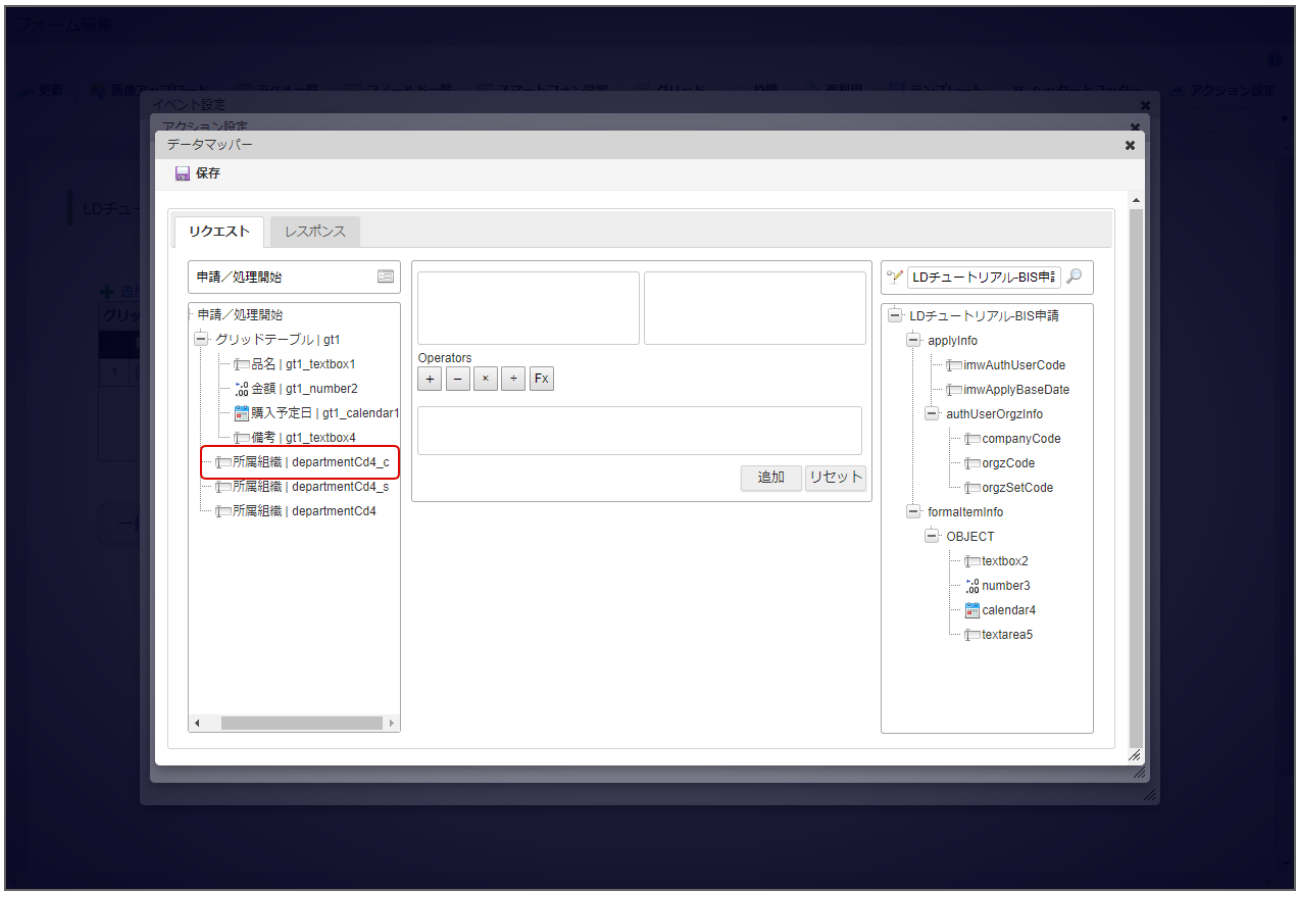

図：データマッパーの設定

10. データソース選択から登録したLogicDesignerのデータソース定義をクリックします。



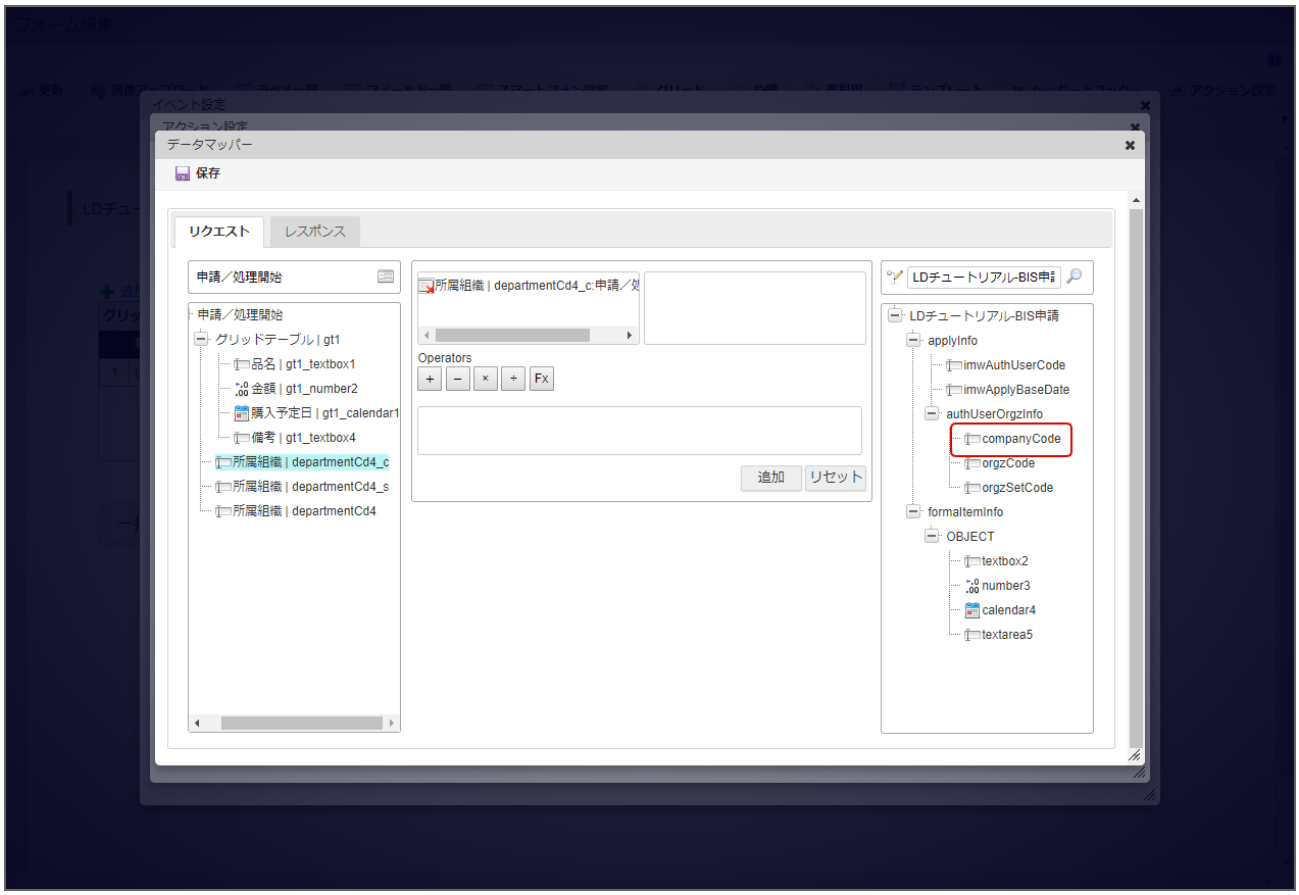

図：データソース定義の選択

11. 左の欄から 所属組織|departmentCd4_c をクリックします。



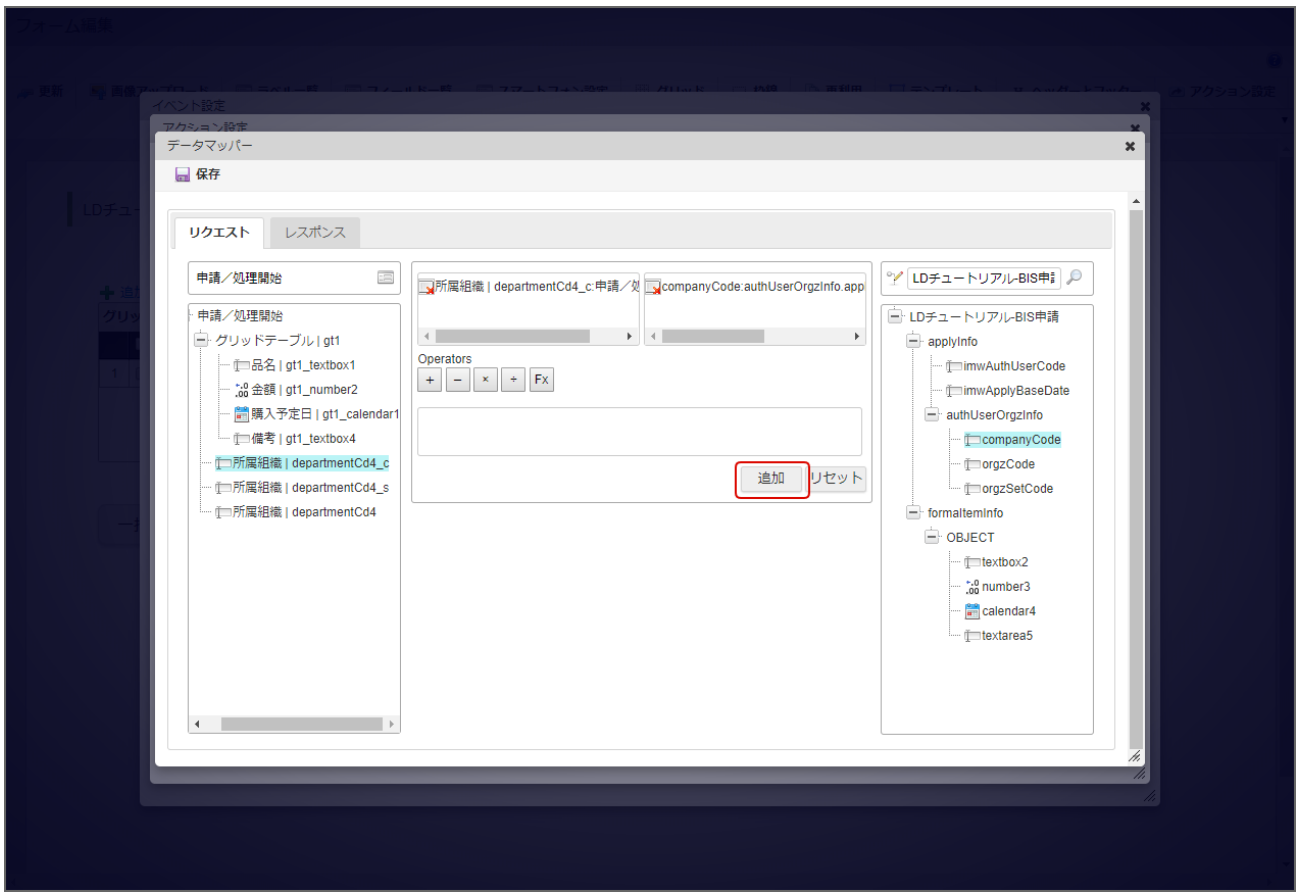

図：データマッパーでのマッピングの設定（1）

12. 右の欄の「LDチュートリアル-BIS申請」配下の companyCode をクリックします。



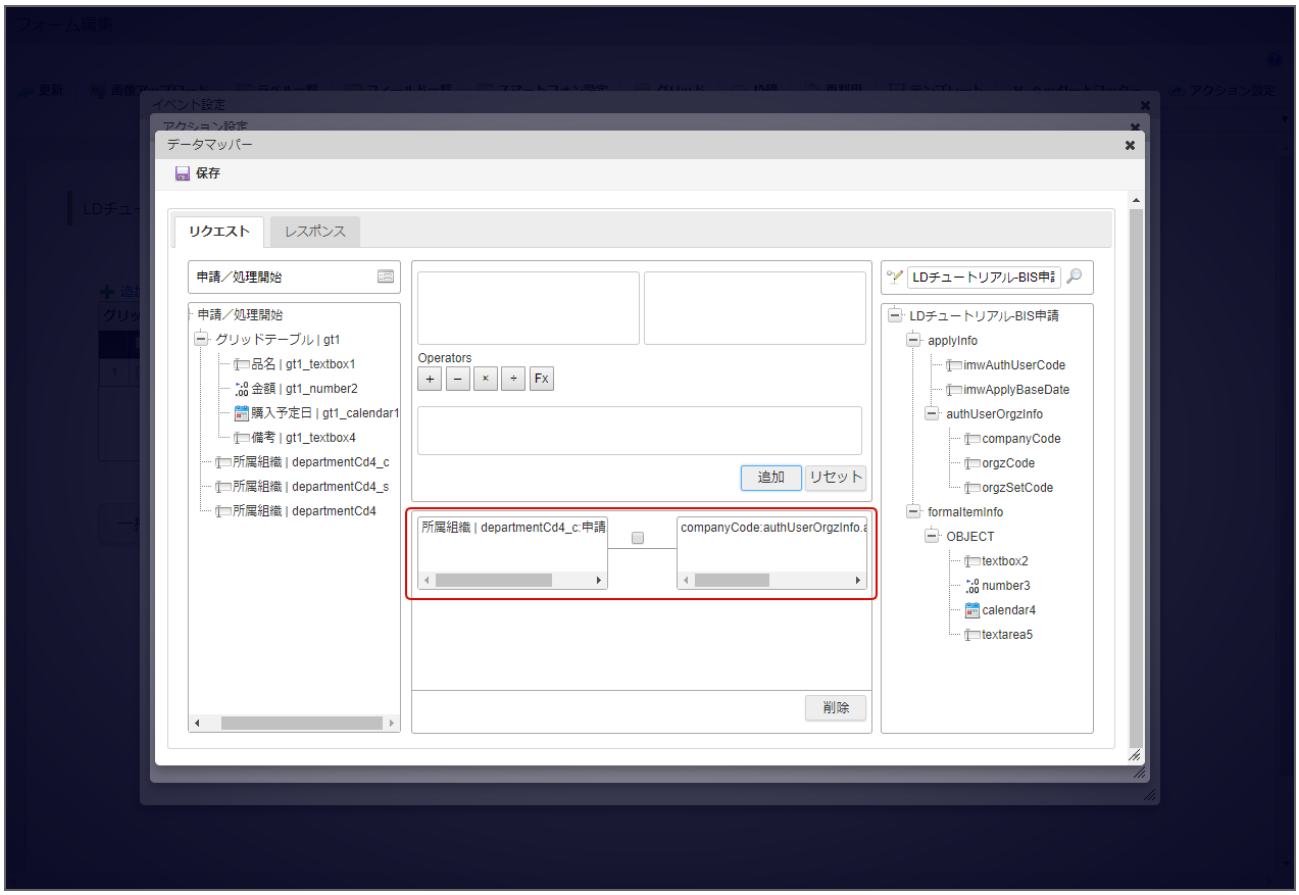

図：データマッパーでのマッピングの設定（2）

13. 「追加」をクリックしてマッピングを追加します。




図：データマッパーでのマッピングの設定（3）

14. 会社コードのマッピングが設定できました。



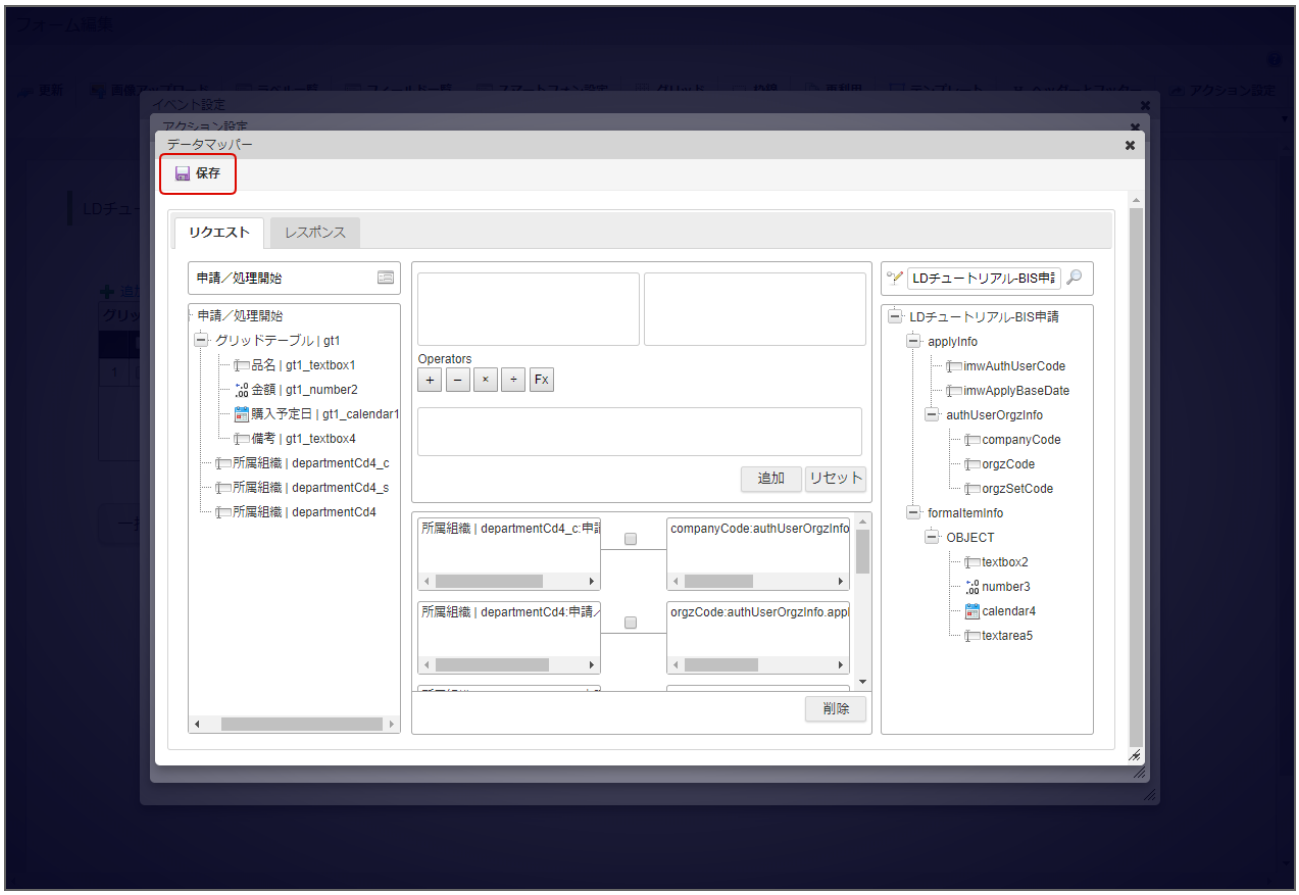

図：データマッパーでのマッピングの設定（4）

15. 同様にして、以下のマッピングを追加してください。

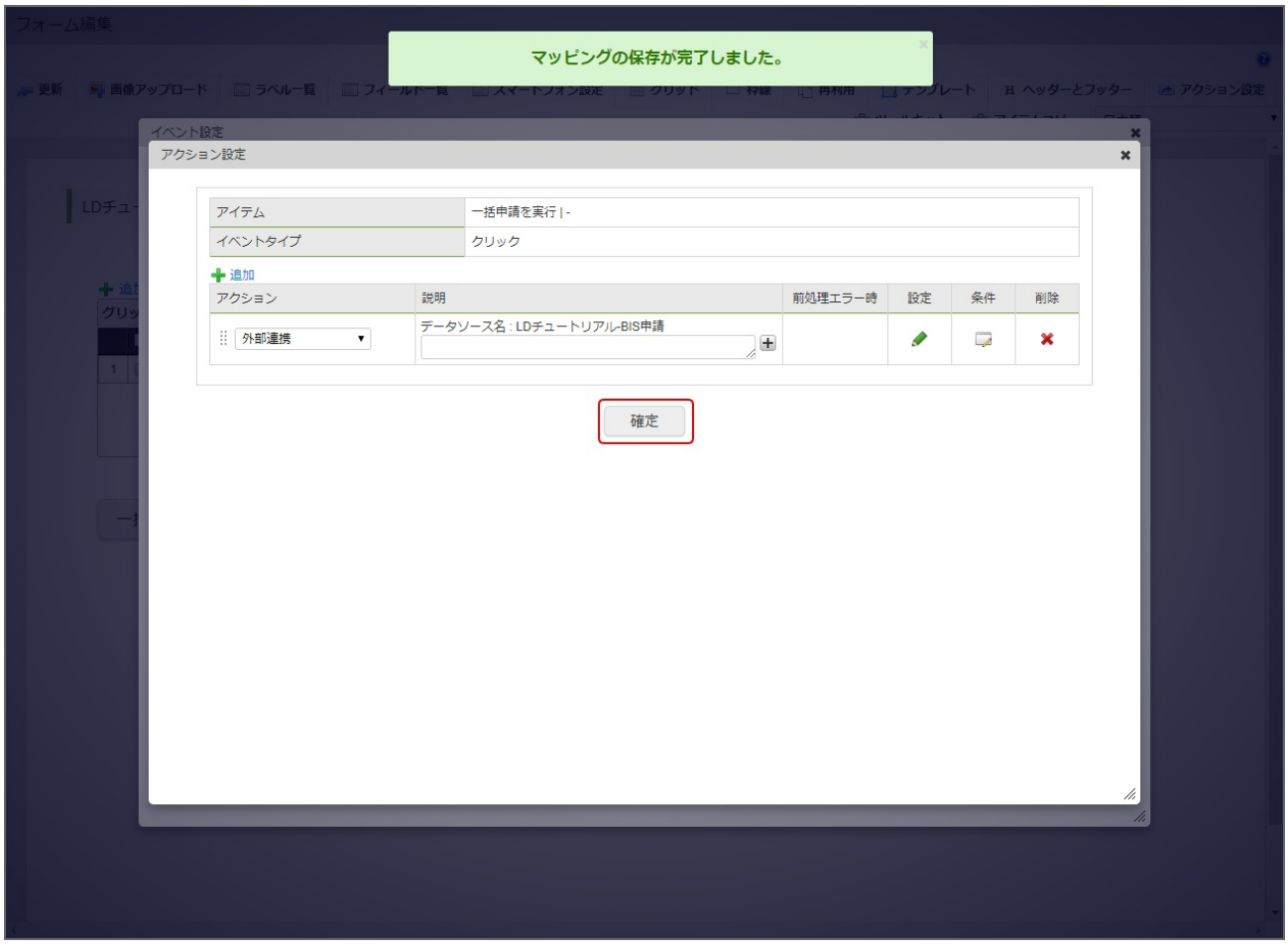
| 左の欄（画面の項目） | 右の欄（データソース定義の項目） |
|---|---|
| 所属組織\|departmentCd4_s | authUserOrgzInfo - orgzSetCode |
| 所属組織\|departmentCd4 | authUserOrgzInfo - orgzCode |
| 品名\|gt1_textbox1 | formalItemInfo - OBJECT - textbox2 |
| 金額\|gt1_number2 | formalItemInfo - OBJECT - number3 |
| 購入予定日\|gt1_calnedar1 | formalItemInfo - OBJECT - calendar4 |
| 備考\|gt1_textbox4 | formalItemInfo - OBJECT - textarea5 |

> **コラム**
>
> 以下の項目については、暗黙的に連携されるため、明示的にマッピングする必要はありません。
>
> - データソース定義「LDチュートリアル-BIS申請」配下
>  - 「リクエスト」タブ
>  - imwAuthUserCode
>  - imwApplyBaseDate
>  - 「レスポンス」タブ
>  - imfrSoaResult 配下全て
>
> 暗黙的に連携するパラメータの詳細は、以下のドキュメントを参照してください
>
> - 「IM-BIS 仕様書」-「暗黙的に連携するリクエストパラメータの仕様」、「暗黙的に連携するレスポンスパラメータの仕様」

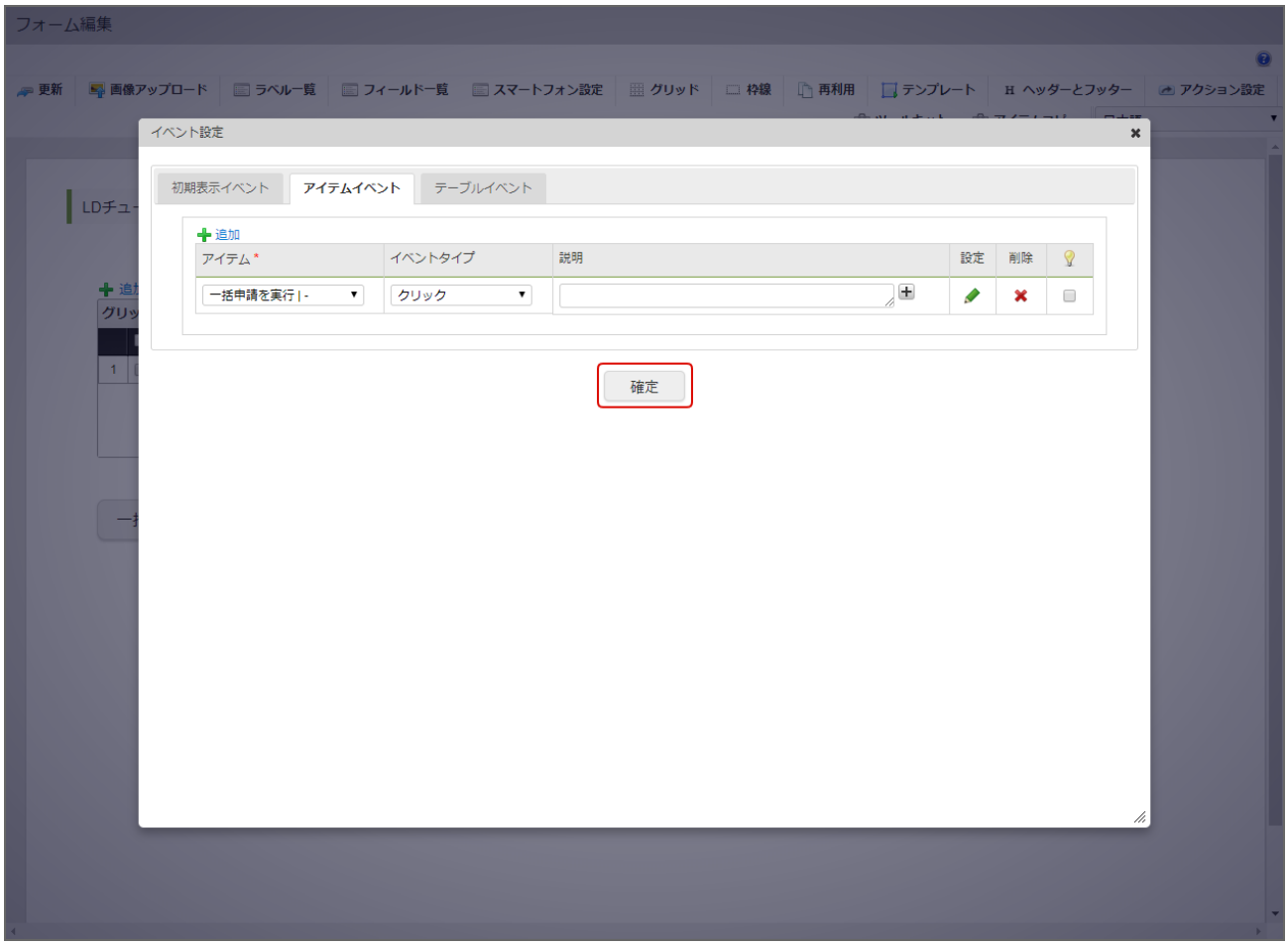
16. 必要なマッピングが設定できたら「保存」をクリック後、右上の×から「データマッパー」画面を閉じます。



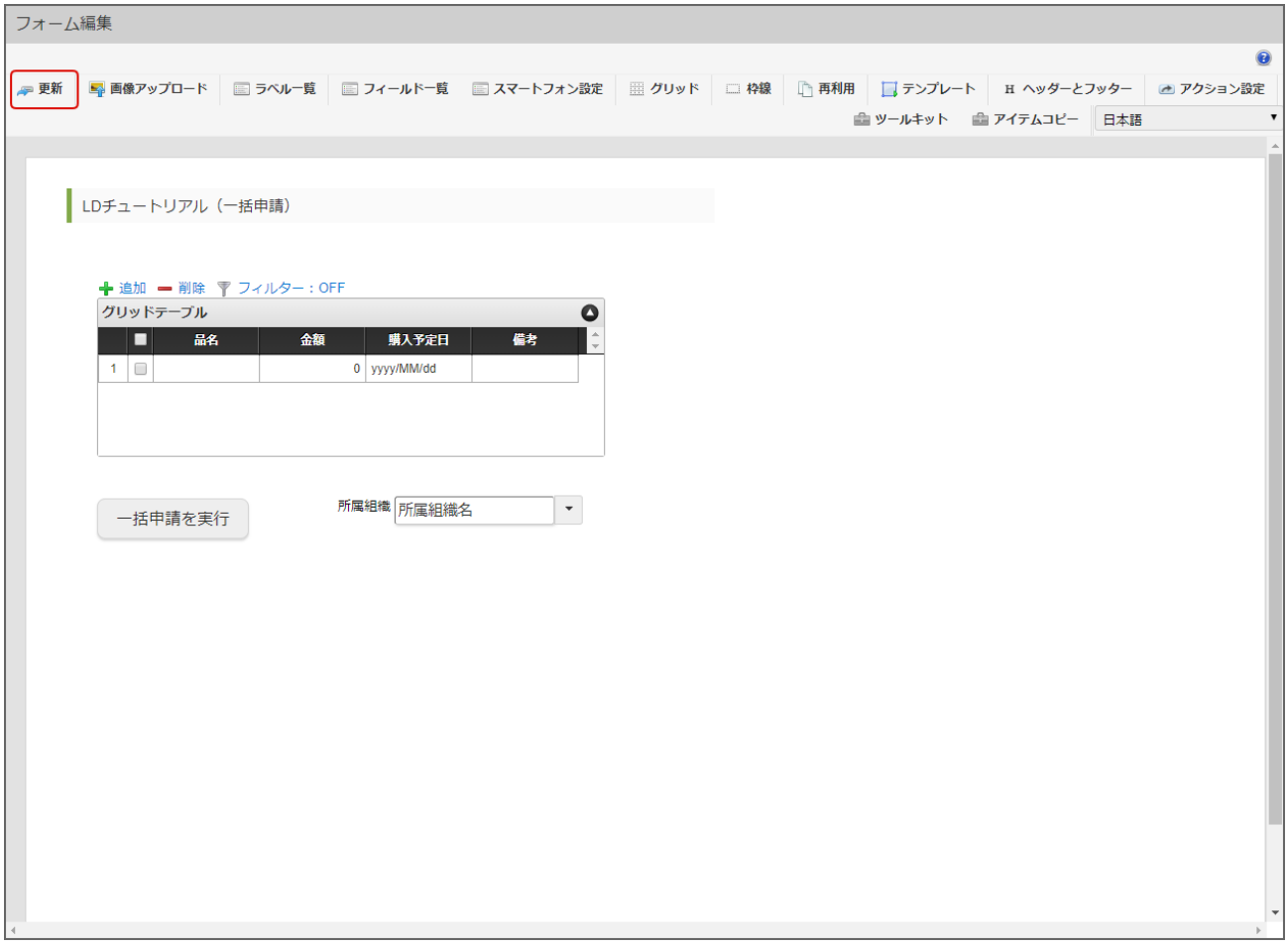

17. 「アクション設定」で「確定」をクリックします。



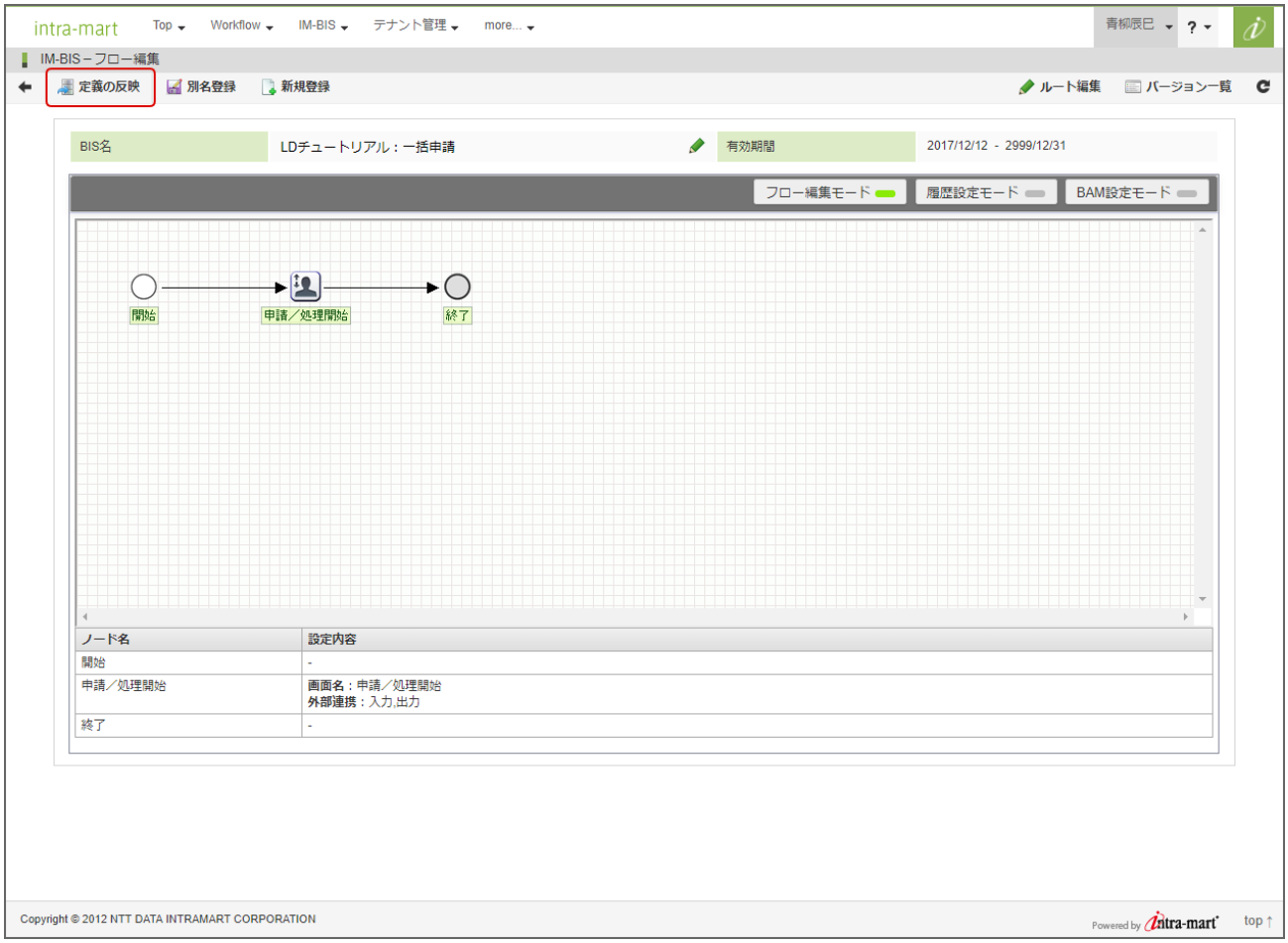

18. 「イベント設定」で「確定」をクリックします。




19. 「フォーム編集」画面の「更新」をクリックして保存します。
保存後はフォーム編集の画面を閉じてください。




20. 最後に「定義の反映」をクリックすると、作成したロジックフローの設定が完了します。
ロジックフローを実行して確認するには、「IM-BIS」配下の「申請一覧」から「LDチュートリアル：一括申請」を表示してください。




以上で、ユーザ定義（BIS申請/承認）とユーザ定義（BIS申請/承認）を利用したロジックフローの作成が完了しました。

## ユーザ定義の利用方法

この章では、作成したユーザ定義をロジックフローで利用する方法を説明します。



### ユーザ定義の配置

作成したユーザ定義を実際にロジックフローに配置する方法を説明します。

ユーザ定義は、標準で提供されているエレメントと同様の方法でロジックフロー上に配置可能です。
ここでは前章までで作成してきたユーザ定義、および、ユーザカテゴリを例としています。

1. 「サイトマップ」→「LogicDesigner」→「フロー定義一覧」から、ロジックフロー定義一覧を開きます。
   「ロジックフロー定義一覧」画面左上の「ロジックフロー新規作成」をクリックし、「ロジックフロー定義編集」画面を開きます。

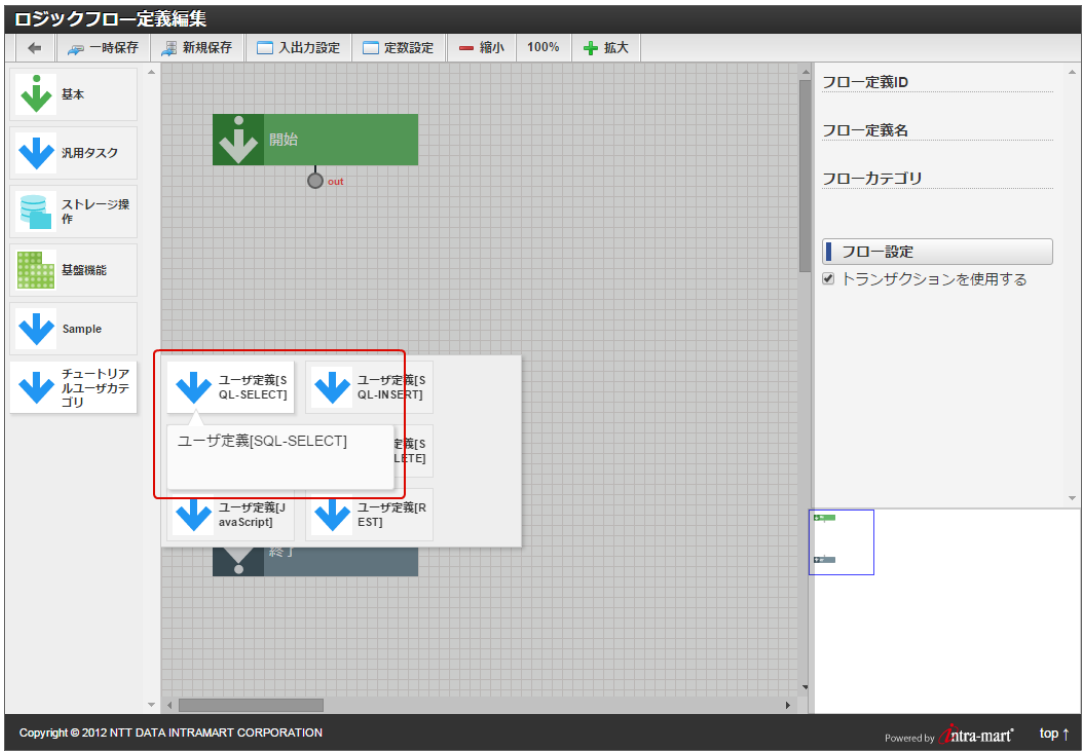
2. 「ロジックフロー定義編集」画面左部のパレットに、「*ユーザカテゴリ*」で作成したカテゴリが新しく配置されることが確認できます。


図：パレット上に作成したユーザカテゴリが表示される

3. パレット内の「チュートリアルユーザカテゴリ」へカーソルを合わせます。
   タスク一覧がスライドインし、前章までで作成してきたユーザ定義が一覧表示されます。

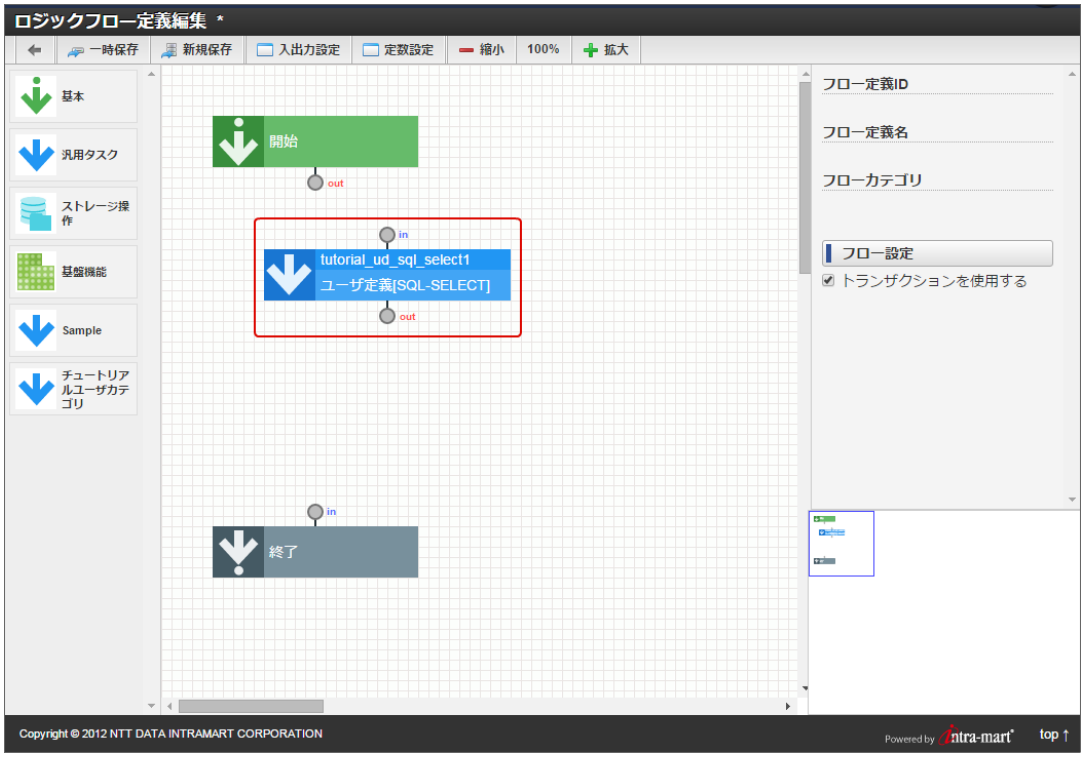

図：作成したユーザ定義一覧

4. 「ユーザ定義SQL-SELECT」をクリックします。




図：「ユーザ定義SQL-SELECT」をクリック

5. 「ユーザ定義SQL-SELECT」に対応するタスクがフロー編集画面上に追加されます。




図：「ユーザ定義SQL-SELECT」の追加

以上で、ロジックフローへのユーザ定義の配置が完了しました

## ユーザ定義の管理

作成したユーザ定義を管理する方法を説明します。

### ユーザ定義のバージョン管理

ユーザ定義は「*ロジックフローのバージョン管理*」で説明を行ったロジックフローと同様の仕様に則ってバージョン管理が行われます。

ユーザ定義のバージョン管理で留意すべき点は、ロジックフローで利用する場合ユーザ定義は常に最新のバージョンが選択される点です。
ユーザ定義の最新バージョンが更新されたタイミングで、対象とするユーザ定義を利用している全てのロジックフローの動作は変更されます。

ユーザ定義の削除

ユーザ定義は「*ロジックフローのバージョン管理*」 - 「*全てのバージョンを削除する（ロジックフローの削除）*」で説明を行ったロジックフローと同様、全てのバージョンを削除することでユーザ定義自体も削除されます。

ただし、ユーザ定義の削除は、削除対象のユーザ定義がどのロジックフローからも利用されていない必要があります。
もし何れかのロジックフローから利用されている場合、以下の様なエラーメッセージが表示され削除を行うことはできません。


図：削除失敗を表すエラーメッセージ

## ユーザカテゴリ、および、ユーザ定義へのアイコン設定方法

この章ではユーザカテゴリ、および、ユーザ定義へアイコンを設定する方法を説明します。



### アイコンについて

2016 Summer(Nirvana)から、ユーザカテゴリ、および、ユーザ定義へアイコンを設定することができるようになりました。
ユーザカテゴリ、および、ユーザ定義へ用途に合わせたアイコンを設定することで、以下の事が見込めます。

- ロジックフロー編集画面上でフロー作成を行う際の視認性が向上し、オペレーションミスを防ぎます。
- 他の人が作成したロジックフローなどを確認する際、配置されたユーザ定義の目的等がわかりやすくなります。


図：アイコン設定概要

ここではアイコンの設定方法、および、設定したアイコンが反映されているかの確認を行います。

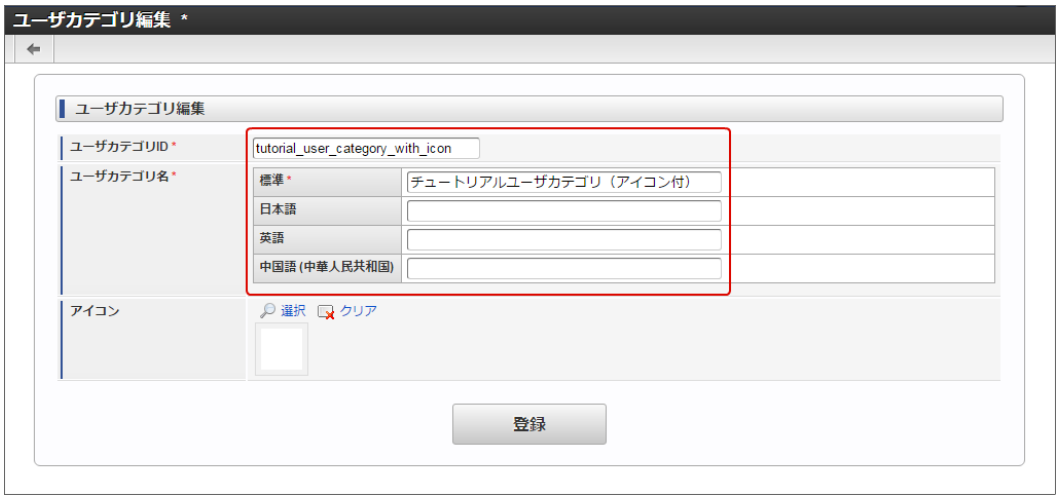
### アイコンの設定

ユーザカテゴリ、および、ユーザ定義それぞれについてアイコンの設定方法を説明します。

ユーザカテゴリにアイコンを設定する

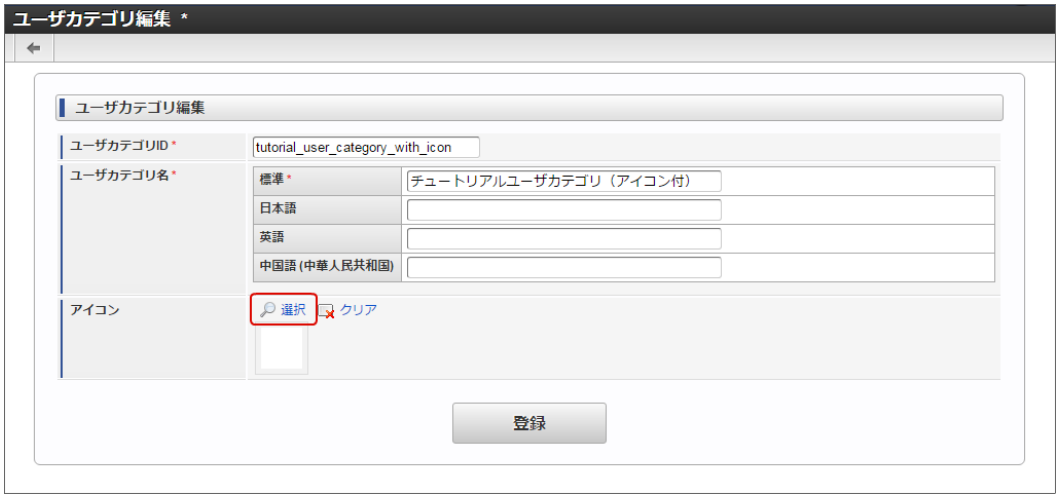
1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義一覧」から、ユーザ定義一覧を開きます。
   ユーザ定義一覧上部のヘッダ内、「カテゴリー覧」をクリックします。
2. ユーザカテゴリー覧画面左上の「新規作成」をクリックします。
3. ユーザカテゴリ情報の各項目に以下の値を入力します。
   - フローカテゴリID「tutorial_user_category_with_icon」

- フローカテゴリ名
  - 標準 - 「チュートリアルユーザカテゴリ（アイコン付）」
  - 日本語、英語、中国語（中華人民共和国） - 入力なし



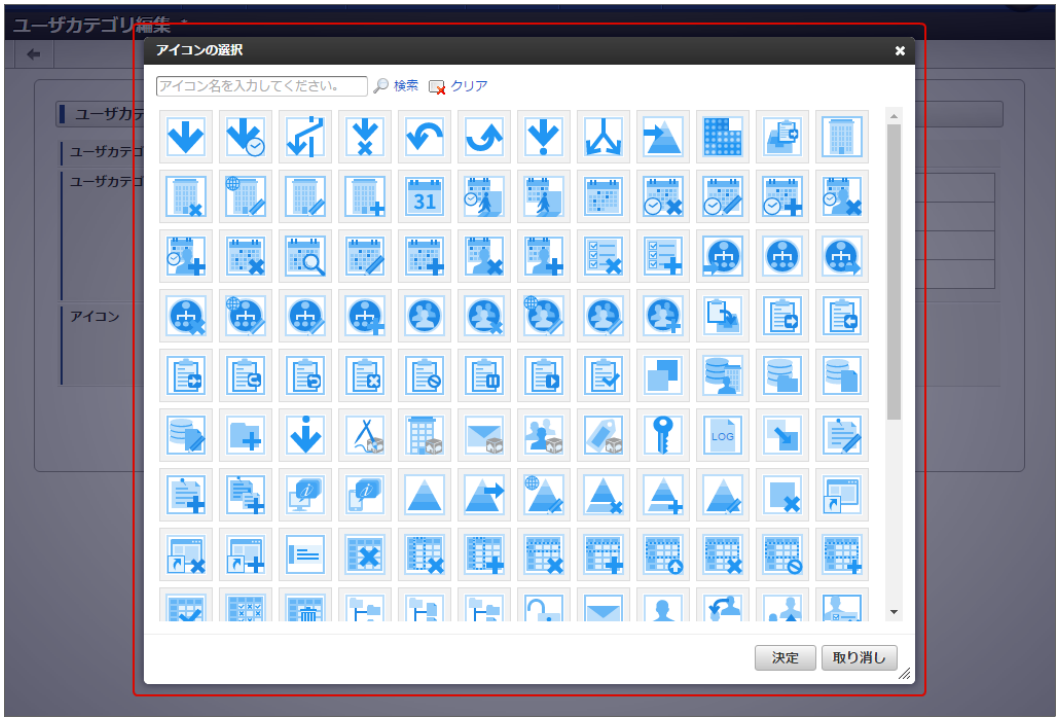

図：ユーザカテゴリ情報の入力

4. ユーザカテゴリ情報のアイコンの項目内の「選択」をクリックします。



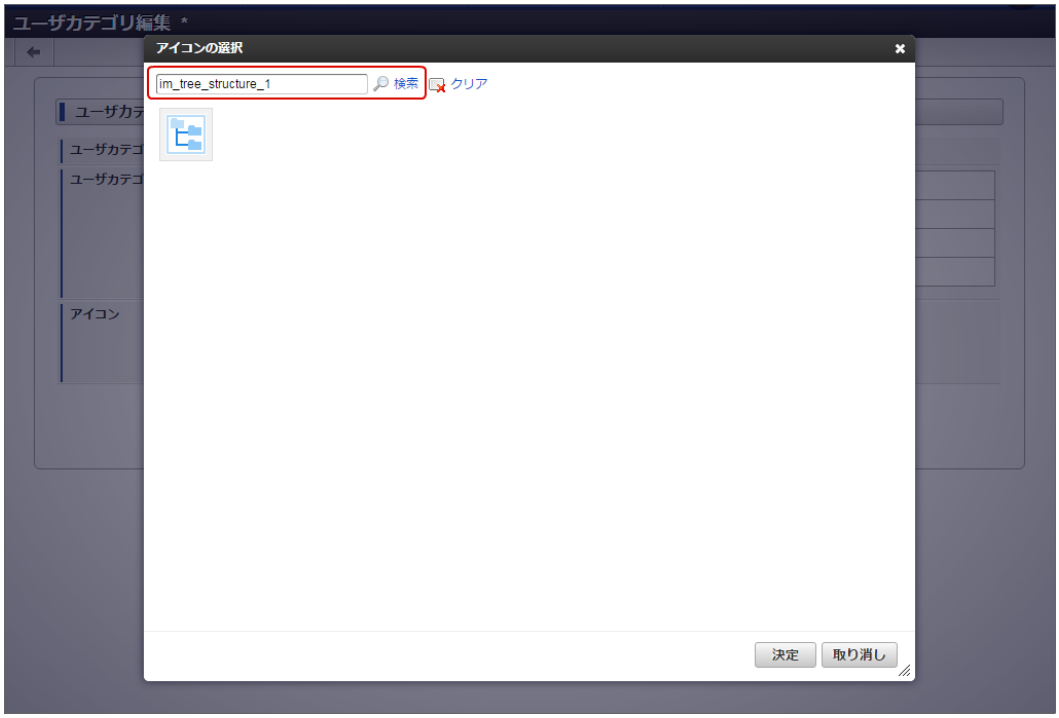

図：アイコンの選択

5. アイコンの選択ダイアログが表示されます。



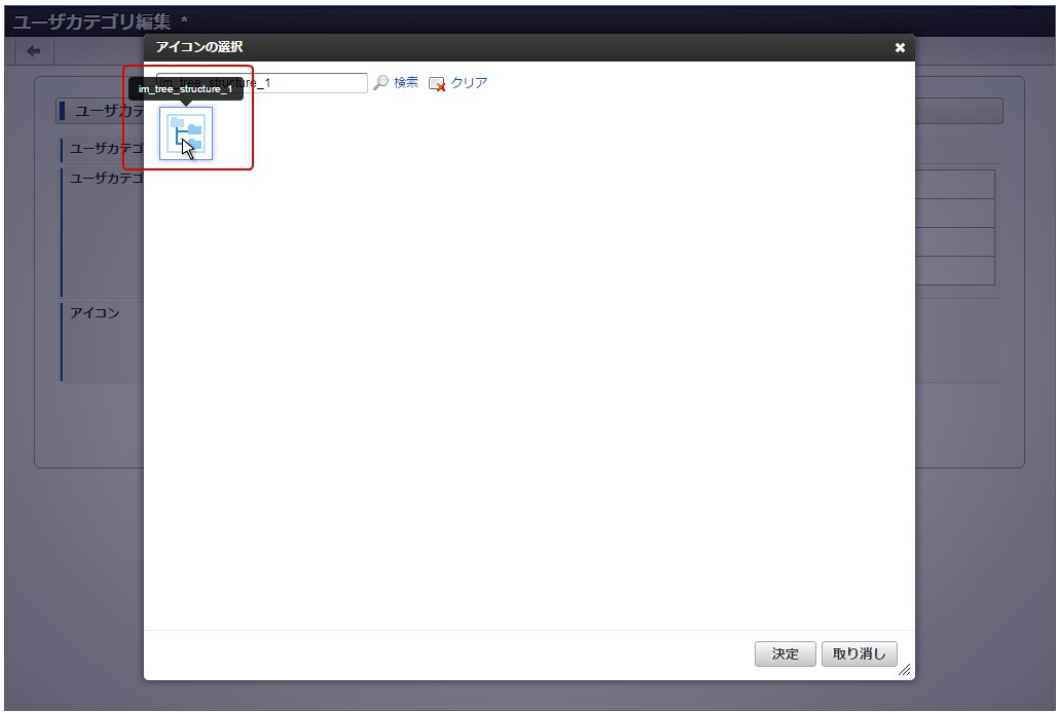
図：アイコン選択ダイアログ

6. ダイアログ上部の検索欄に「im_tree_structure_1」と入力し、「検索」をクリックします。



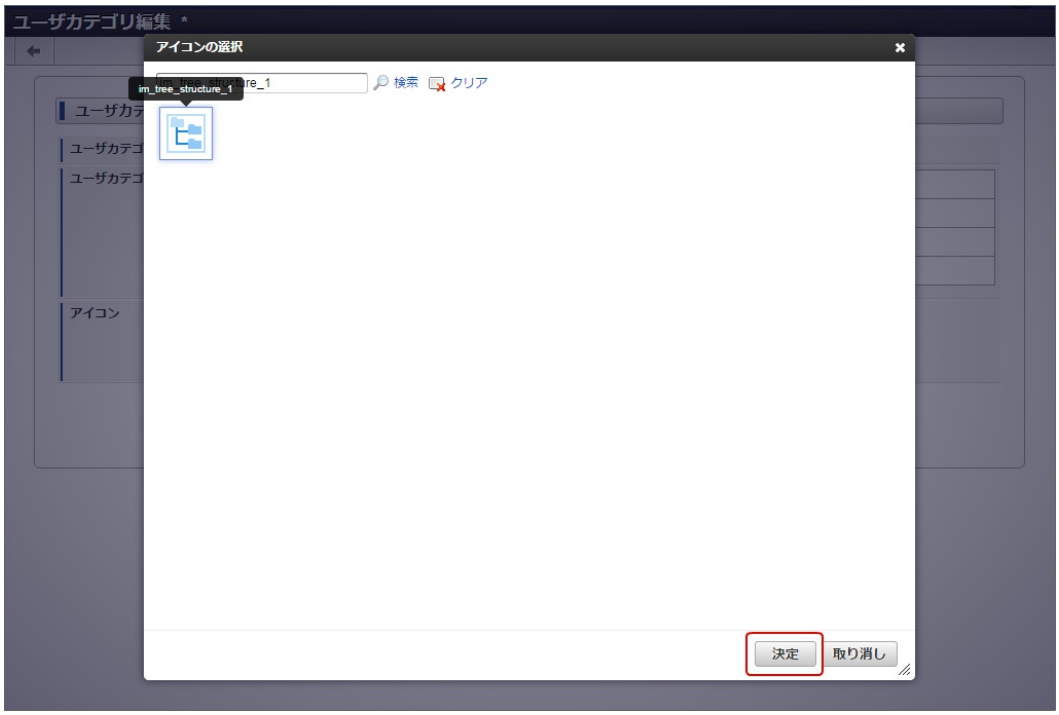

図：検索

7. 検索結果の候補で表示されたアイコンをクリックします。




図：アイコンの選択

8. 「決定」をクリックします。




図：決定

9. 「登録」をクリックします。

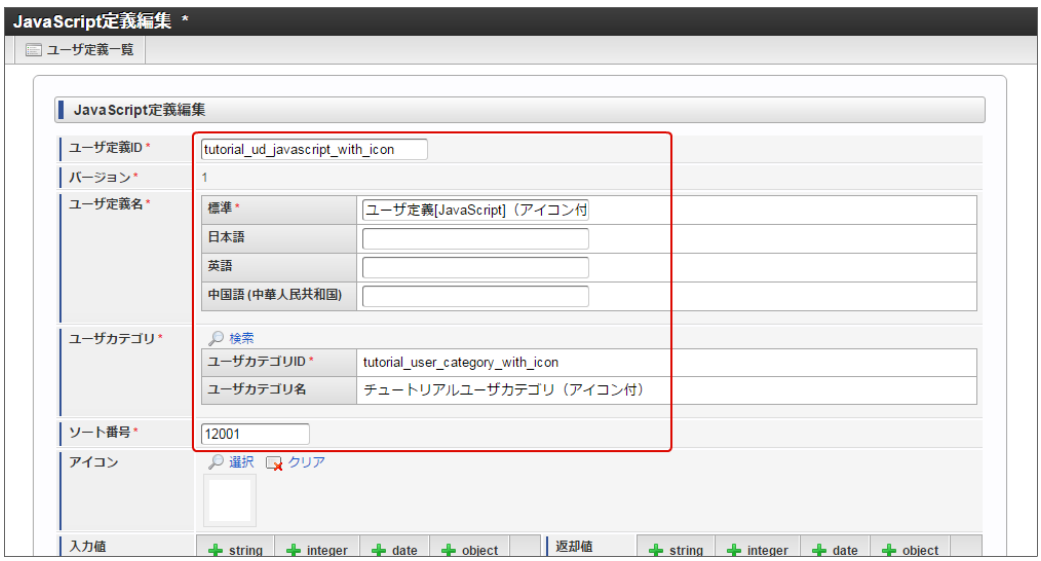
以上で、ユーザカテゴリへアイコンを設定できました。

ユーザ定義にアイコンを設定する

1. 「サイトマップ」→「LogicDesigner」→「JavaScript定義新規作成」から、JavaScript定義新規作成を開きます。

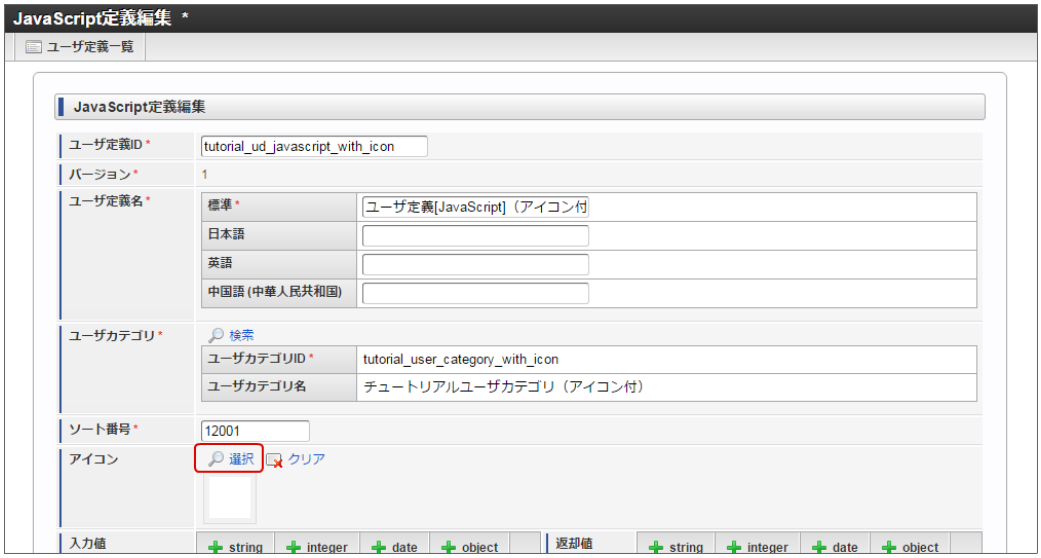
> **コラム**
> 本チュートリアルではユーザ定義（JavaScript）をベースに説明を進めますが、他のユーザ定義でも設定方法は同じです。

2. ユーザ定義の基本情報となる各項目に以下の値を入力します。
   - ユーザ定義ID「tutorial_ud_javascript_with_icon」
   - バージョン「1」（固定）
   - ユーザ定義名
     - 標準 -「ユーザ定義[JavaScript]（アイコン付）」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし
   - カテゴリ
     - カテゴリID -「tutorial_user_category_with_icon」
   - ソート番号「12001」



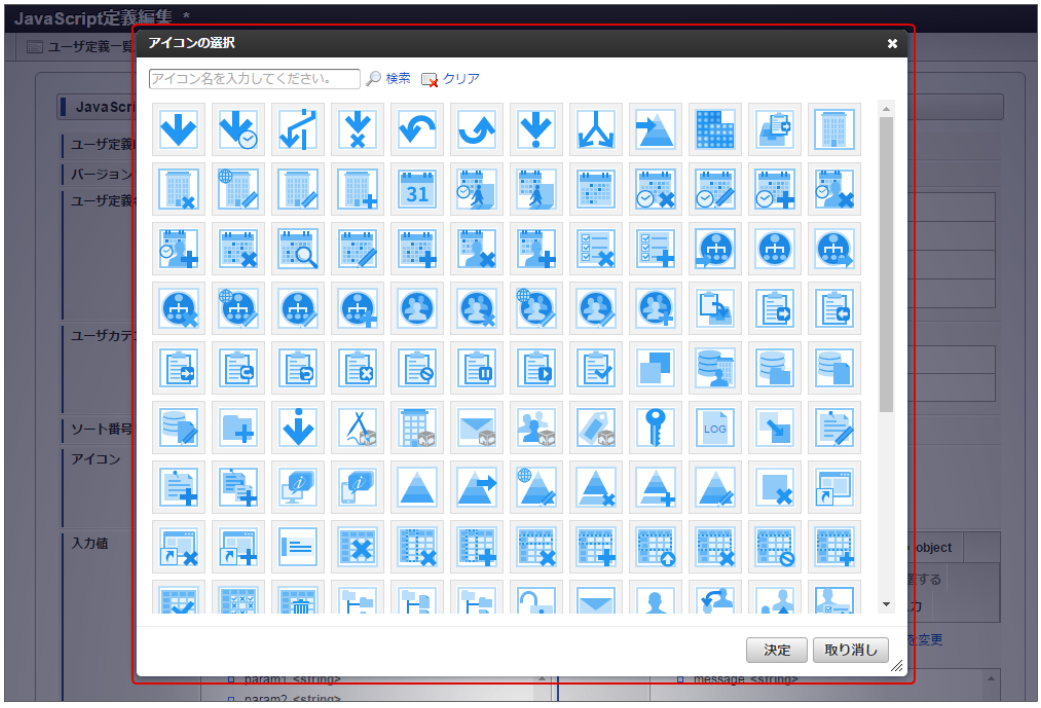
図：ユーザ定義情報の入力

3. JavaScript定義編集情報のアイコンの項目内、「選択」をクリックします。



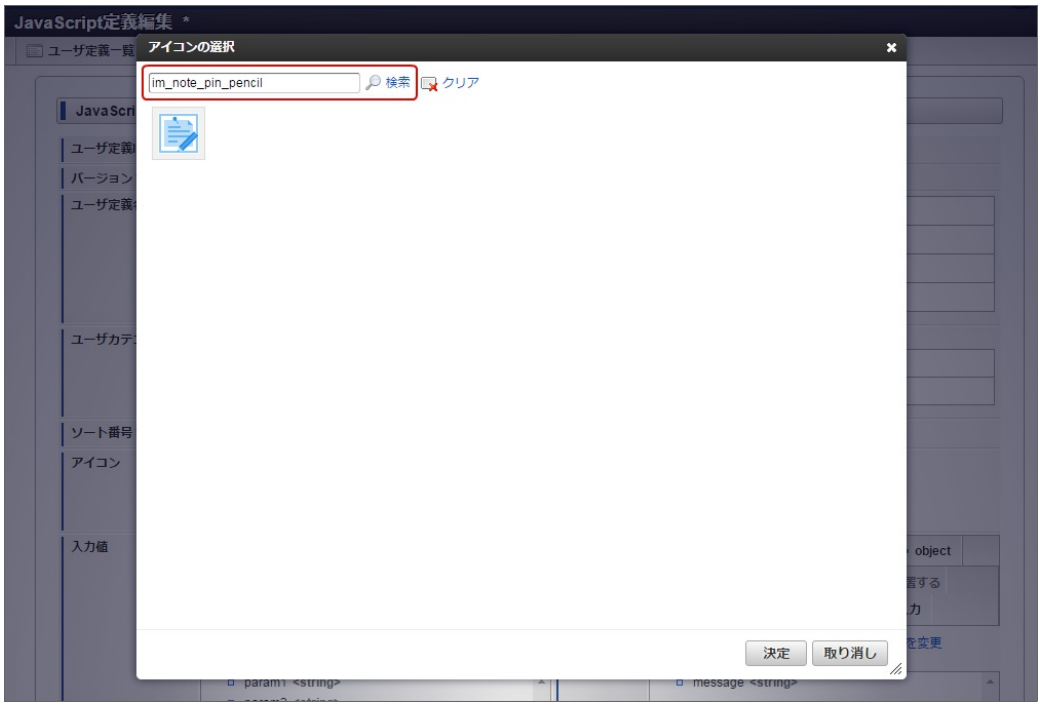

図：アイコンの選択

4. アイコンの選択ダイアログが表示されます。



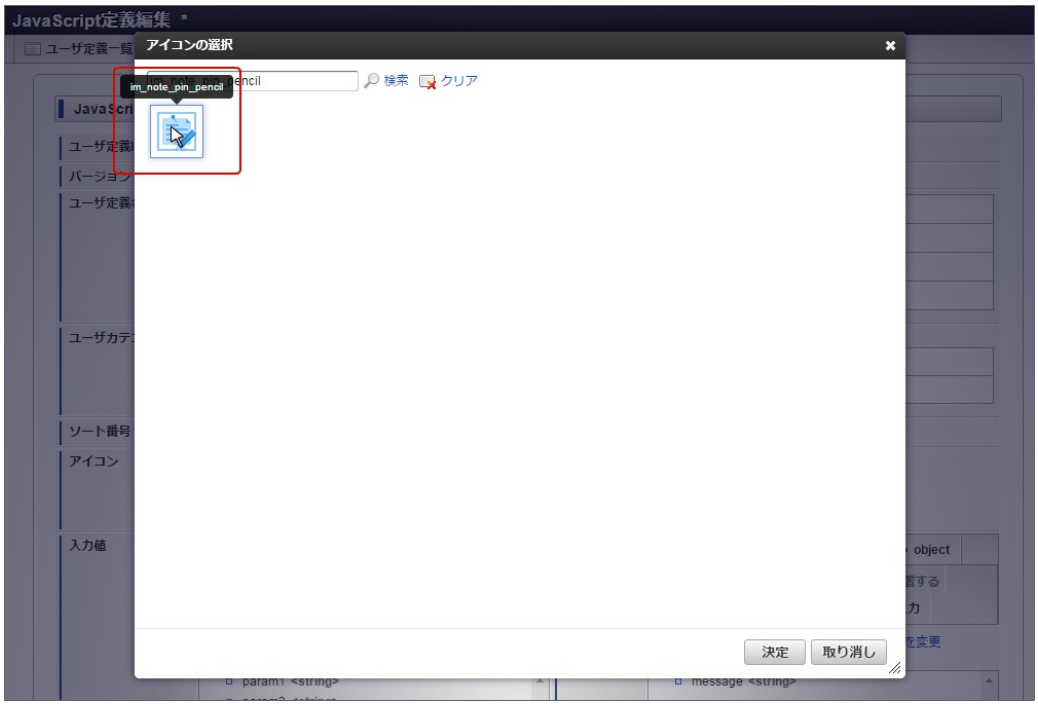

図：アイコン選択ダイアログ

5. ダイアログ上部の検索欄に「im_note_pin_pencil」と入力し、「検索」をクリックします。



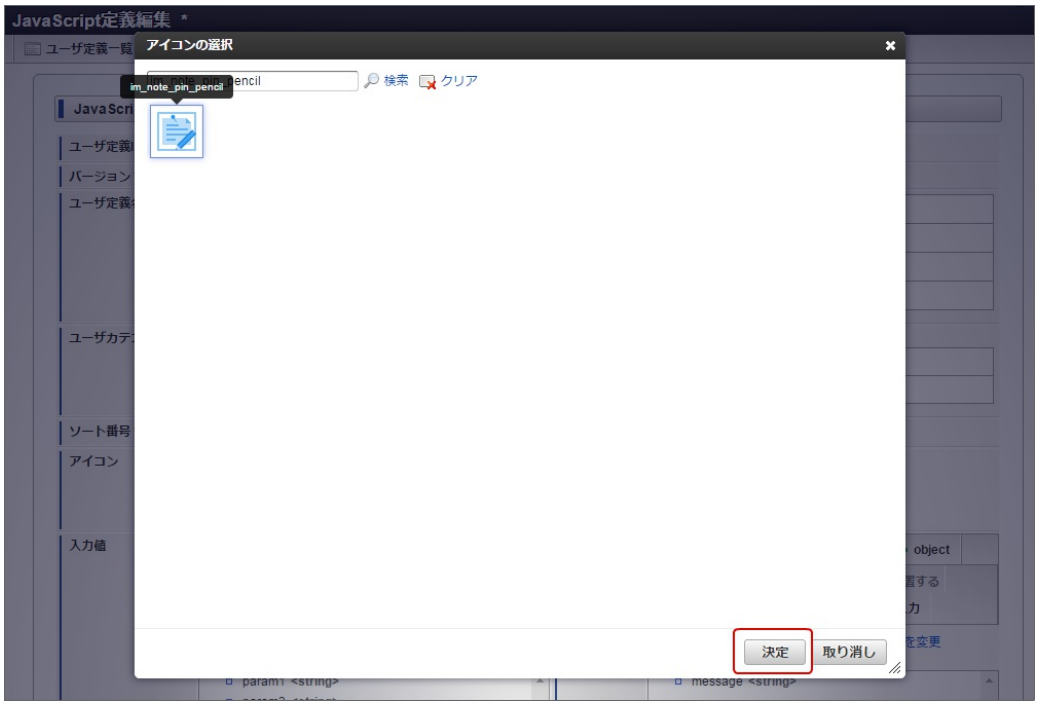

図：検索

6. 検索結果の候補で表示されたアイコンをクリックします。




図：アイコンの選択

7. 「決定」をクリックします。




図：決定

8. 「登録」をクリックします。

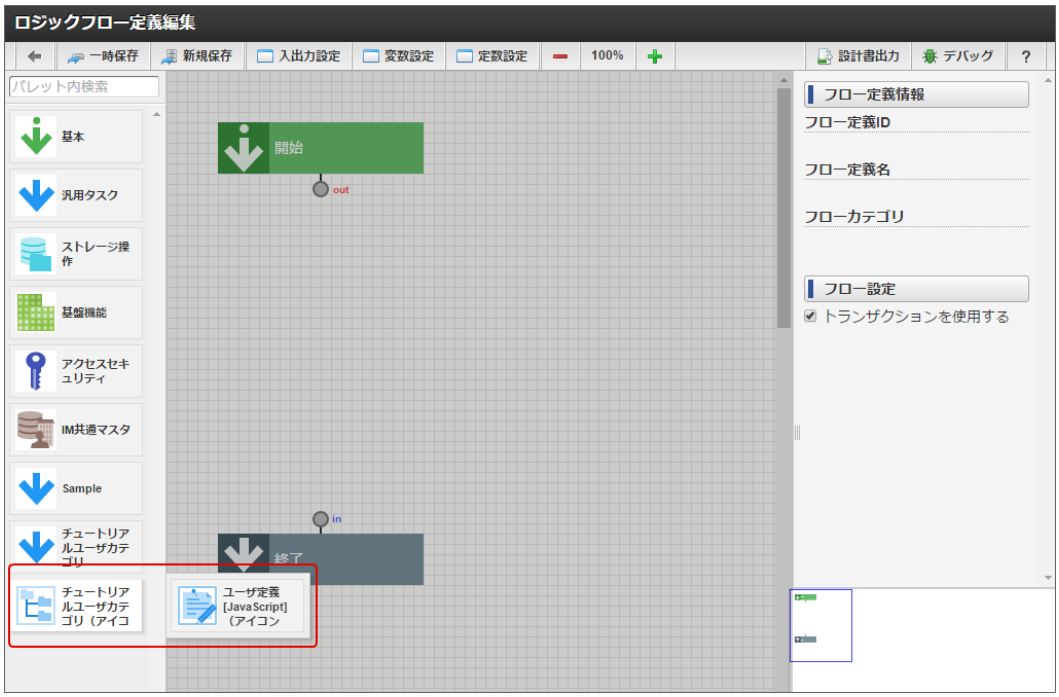
以上で、ユーザ定義へアイコンを設定できました。

## アイコンが設定されたことを確認する

ユーザカテゴリ、および、ユーザ定義それぞれに設定したアイコンが反映されていることを確認します。

1. 「サイトマップ」→「LogicDesigner」→「フロー定義一覧」から、ロジックフロー定義一覧を開きます。
   ロジックフロー定義一覧画面左上の「ロジックフロー新規作成」をクリックします。

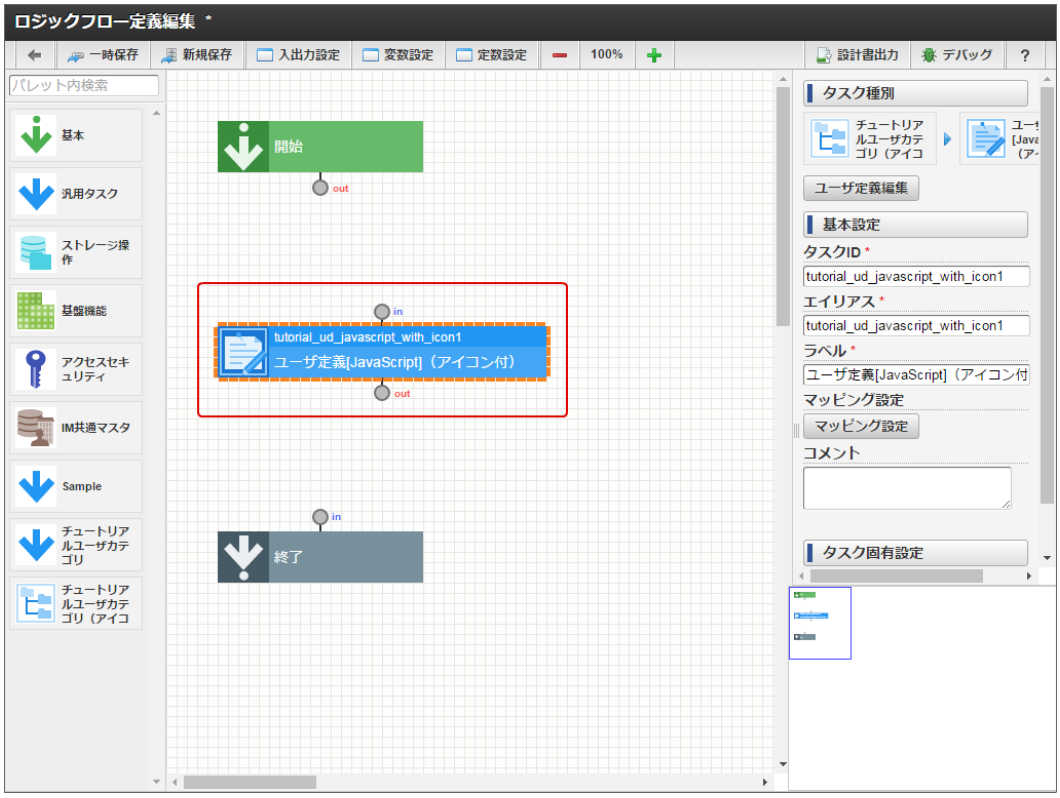
2. 「ロジックフロー定義編集」画面から、以下を確認します。
   - 画面左部、エレメント一覧の「チュートリアルユーザカテゴリ（アイコン付）」ユーザカテゴリのアイコン
   - チュートリアルユーザカテゴリ（アイコン付）ユーザカテゴリにカテゴライズされている「ユーザ定義[JavaScript]（アイコン付）」ユーザ定義のアイコン




図：アイコンの確認　その 1

3. 「ユーザ定義[JavaScript]（アイコン付）」ユーザ定義をクリックします。

4. フロー編集画面上に追加されたユーザ定義に、設定を行ったアイコンが表示されていることを確認します。




図：アイコンの確認　その2

以上で、ユーザカテゴリ、および、ユーザ定義へアイコンが設定されたことを確認できました。

> **コラム**
>
> アイコンが未設定のユーザカテゴリ、および、ユーザ定義について
>
> アイコンが未設定の場合、および、2016 Summer(Nirvana)以前に作成したユーザカテゴリやユーザ定義をインポートした場合、アイコンはデフォルトで「im_arrow_270」が表示されます。
>
> 
> 
>
> 図：「im_arrow_270」アイコン

## 独自アイコンの追加方法

この章ではユーザカテゴリ、および、ユーザ定義へ設定可能なアイコン一覧に、独自のアイコンを追加する方法を説明します。



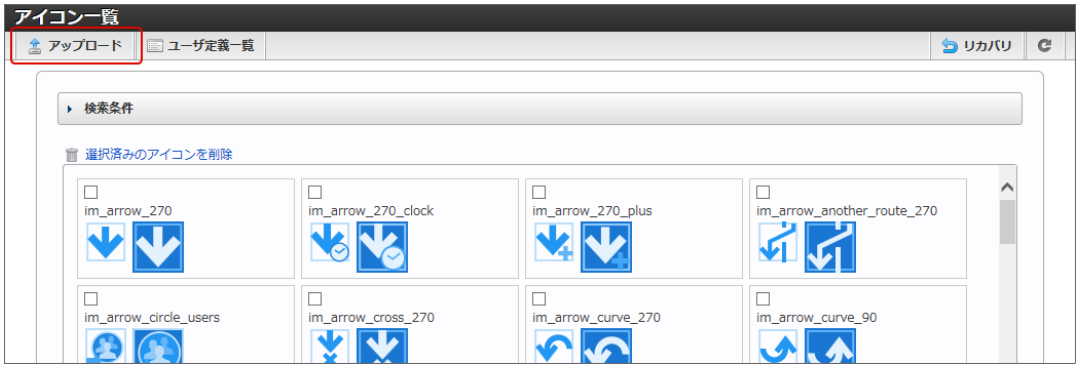
### アイコンのアップロード

2017 Spring(Portland)から、ユーザカテゴリ、および、ユーザ定義のアイコンに、独自のアイコンを追加できるようになりました。
ユーザカテゴリ、および、ユーザ定義それぞれについてアイコンの設定方法を説明します。

アイコン一覧に独自アイコンを追加する

1. 「サイトマップ」→「LogicDesigner」→「ユーザ定義」-「アイコン一覧」から、アイコン一覧を開きます。

2. アイコン一覧画面左上の「アップロード」をクリックします。




図：アイコン一覧画面

3. アイコン情報の各項目に以下の値を入力します。
   - アイコン名「tutorial_icon」
   - アイコンファイル（パレット用）
     - 40x40 の画像ファイルを選択します。
   - アイコンファイル（デザイナ用）
     - 50x50 の画像ファイルを選択します。


図：アイコン情報の入力

4. 「決定」をクリックします。

以上で、アイコン一覧へ独自のアイコンを追加できました。

### 独自アイコンをユーザ定義に設定する

「*ユーザカテゴリ、および、ユーザ定義へのアイコン設定方法*」-「*アイコンの設定*」を参照して、アイコンの設定を行います。
ユーザカテゴリ、および、ユーザ定義に対して、追加した独自アイコンを設定できます。

## ロジックフローのデバッグ

この章では、ロジックフローのデバッグ機能について説明します。



### ロジックフローのデバッグ機能とは

ロジックフローのデバッグ機能とは、ブラウザ上からロジックフローのステップ実行や、実行中のフローに流れるパラメータの参照などが行える機能です。
デバッグ機能を用いることで、以下のようなロジックフロー上の問題を解決することが可能です。

- 作成したロジックフローが特定の入力で必ず失敗する（どのエレメントで失敗しているのか特定したい）。

- あるタスクが想定とは違う処理を行う（そのタスクへ渡される実際の入力値と出力値を確認したい）。
- 条件分岐や繰り返し処理が正しく動作していない（制御結果を確認したい）。

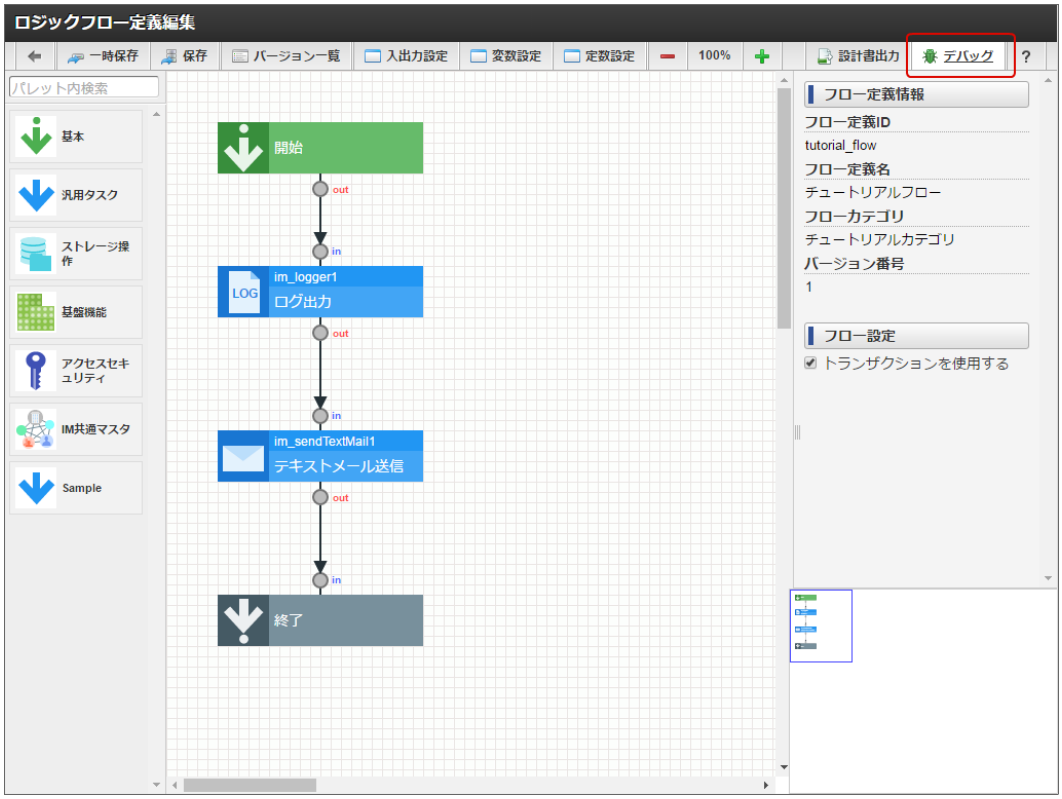
また、こうした問題解決以外にも、作成したロジックフローの簡単なテスト実行にも利用することができます。

## デバッグ画面を表示する

ロジックフローのデバッグ画面は、ロジックフロー定義編集画面から開きます。

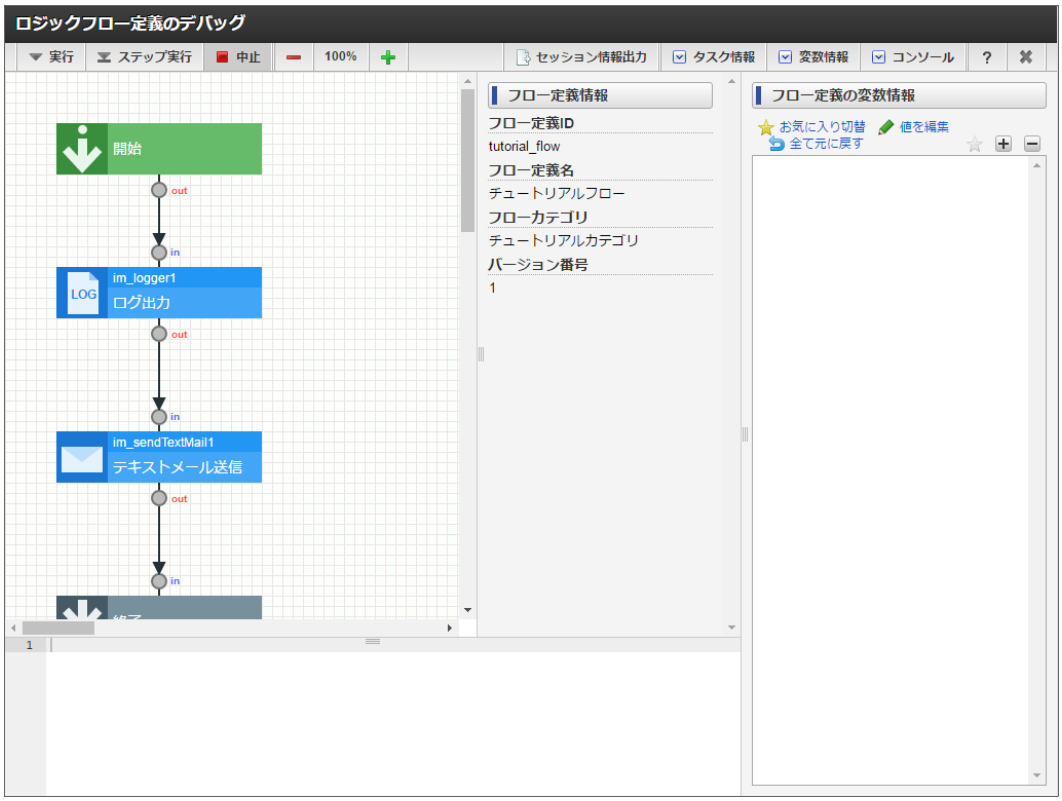
1. 「サイトマップ」→「LogicDesigner」→「フロー定義一覧」から、ロジックフロー定義一覧を開きます。
   左ペインのツリーから「チュートリアルカテゴリ」の開閉アイコンをクリックしカテゴリ配下を情報を表示します。
   フロー定義「チュートリアルフロー」を選択し編集ボタンをクリックします。

2. ロジックフロー定義編集画面上部、ヘッダ内の「デバッグ」をクリックします。




図：ヘッダ内の「デバッグ」

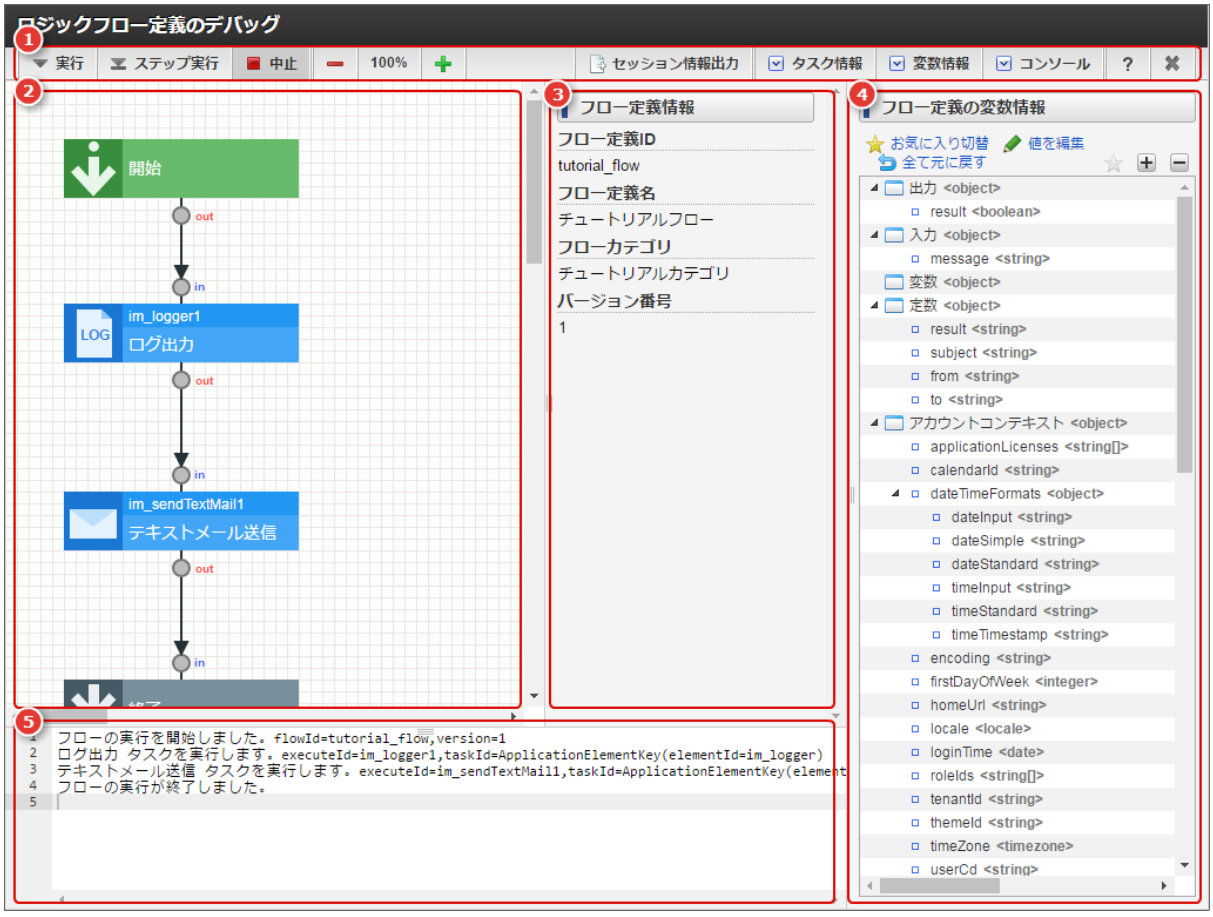
3. ロジックフロー定義のデバッグ画面が新規にポップアップして表示されます。




図：デバッグ画面

## デバッグ画面の詳細

ロジックフローのデバッグ画面は、用途に応じて複数の区画（ペイン）に分かれています。
各ペインの詳細は以下の通りです。




図：デバッグ画面の全体図

1. フローの実行・中断や、各種ペインの表示・非表示を切り替えることができるメニューヘッダです。

2. デバッグを行うロジックフローの確認と、デバッグ実行時に処理をサスペンドするためのブレイクポイントの設定ができる「ロジックフロー確認」ペインです。
   なお、あくまでフローの確認が主であるため、この画面上でのロジックフローの編集は行えません。

3. 「ロジックフロー確認」ペイン上で選択したエレメントの詳細を表示する「タスク情報」ペインです。
   「ロジックフロー確認」ペインと同様に確認が主であるため、タスク情報ペイン内での設定変更は行えません。
   ただし、後述の「*応用：ブレイクポイントを利用したデバッグ実行*」で説明しているブレイクポイントの設定のみ、タスク情報ペイン内でも可能です。

4. ロジックフローをデバッグ実行する上で関わる全ての変数情報の表示、および、編集ができる「フロー定義の変数情報」ペインです。
   このペインで確認可能な変数情報は以下の通りです。
   - デバッグ実行時に渡す入出力値
   - 各エレメントの入出力の返す値
   - 定数値・変数といった事前に定義されている値

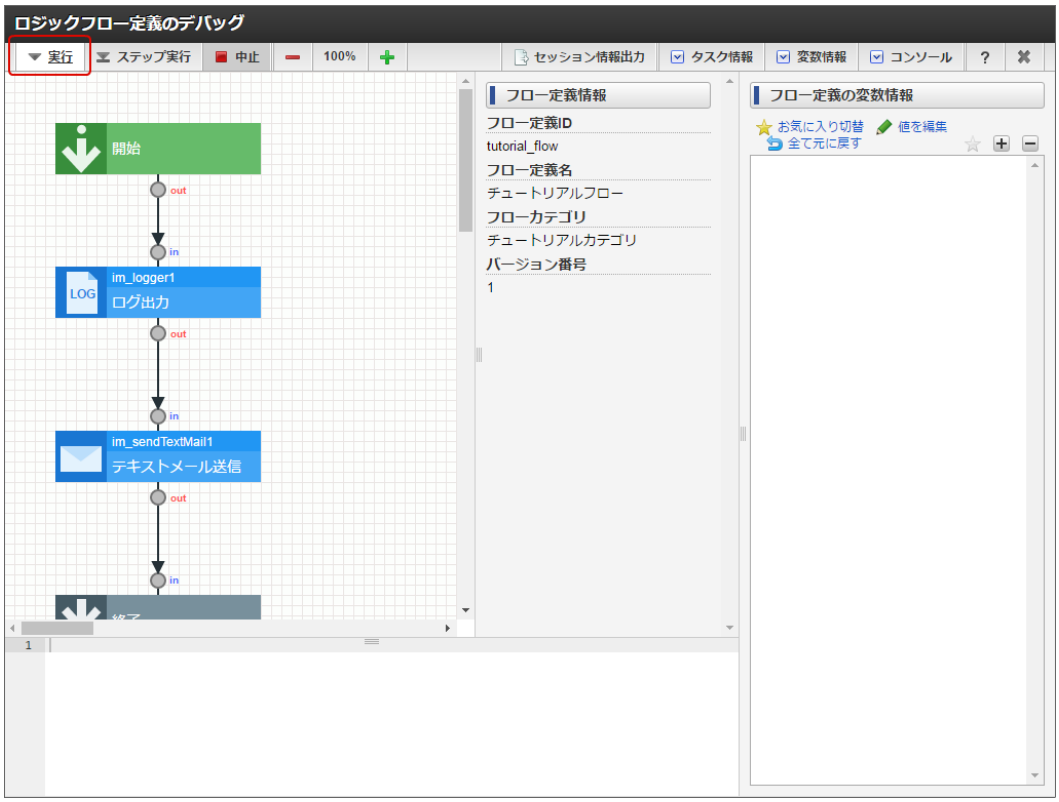
5. デバッグ画面上の処理情報がログとして出力される「コンソール」ペインです。

## ロジックフローをデバッグ実行する

ロジックフローをデバッグ実行する方法を説明します。

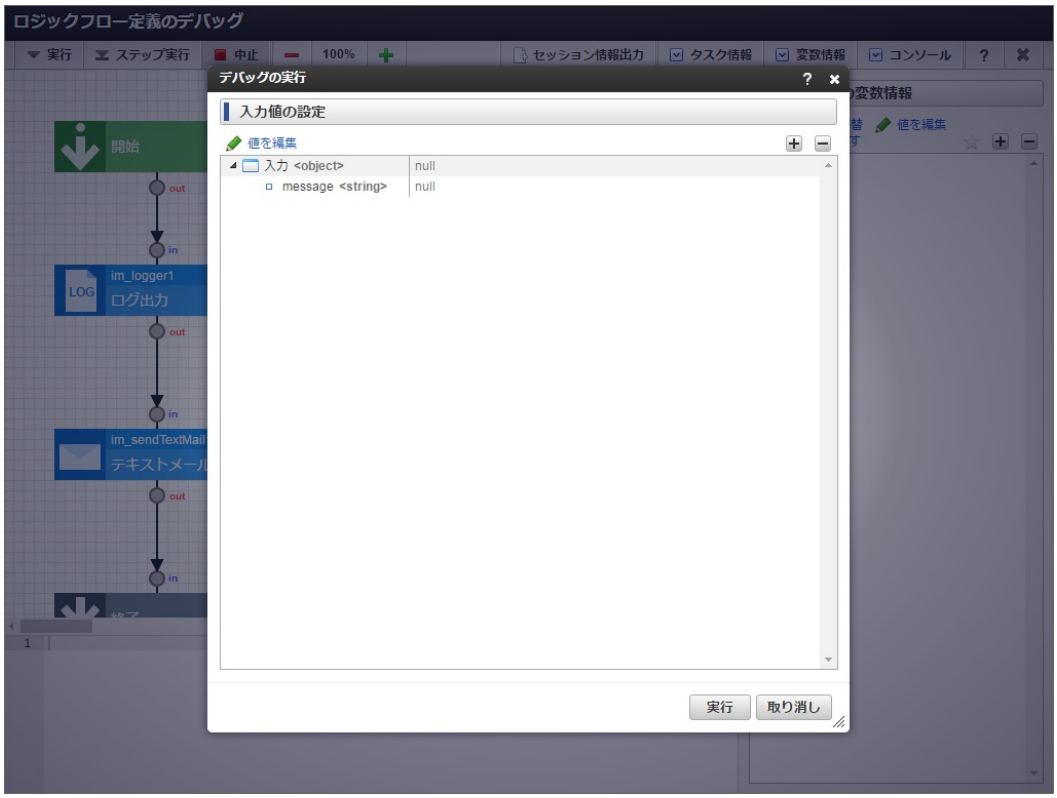
1. デバッグ画面を開きます。

2. メニューヘッダ内の「実行」をクリックします。



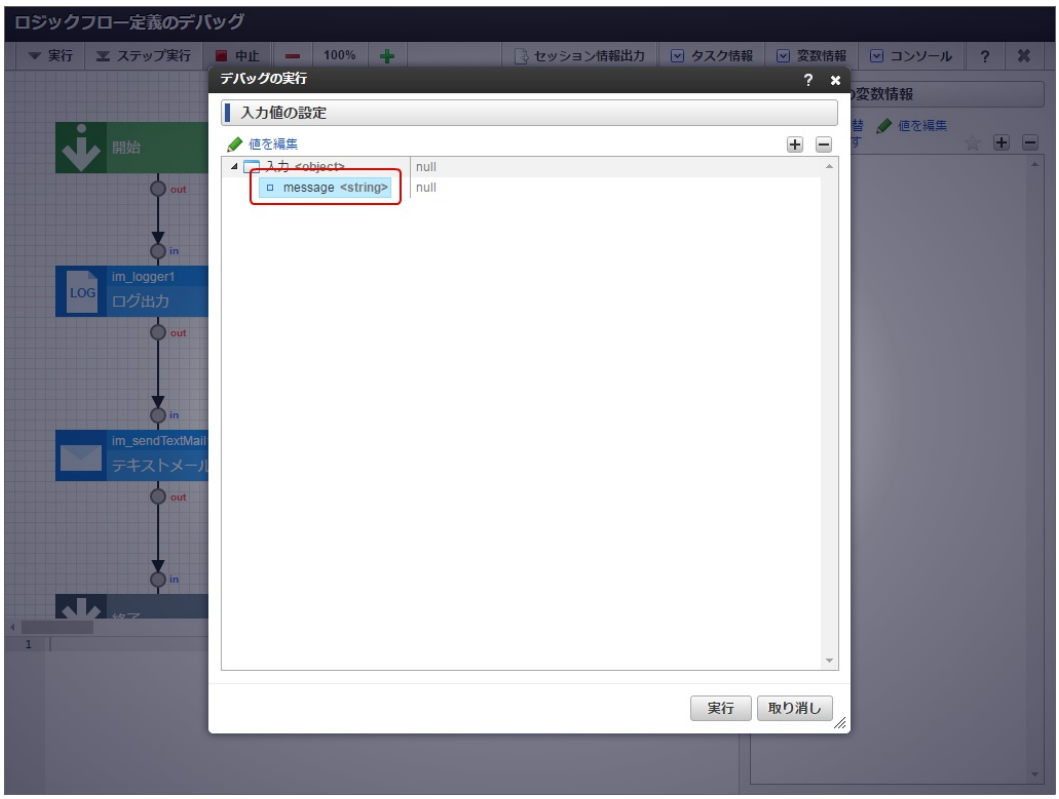

図：ヘッダ内の「実行」

3. デバッグ実行を行う上で必要な「入力値の設定」項目が表示されます。



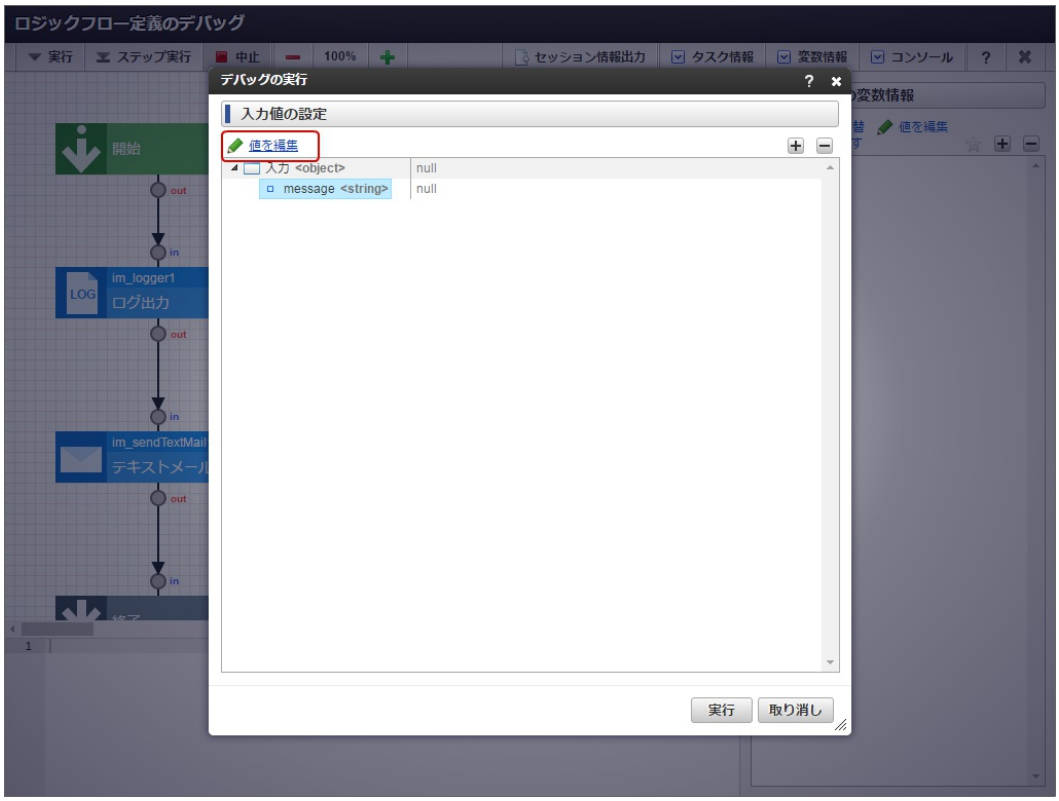

図：デバッグの実行設定画面

4. 入力<object>配下のmessage<string>をクリックし、選択状態にします。



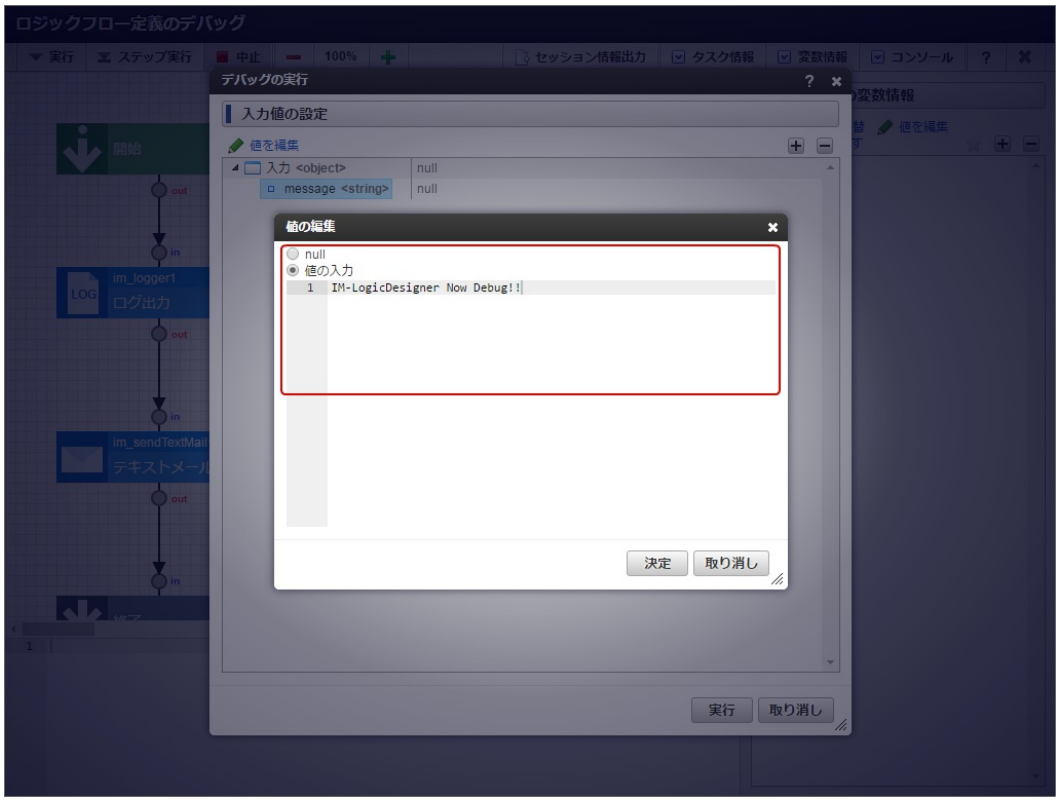

図：デバッグの実行設定画面 - 入力のフォーカス

5. 値一覧の左上の「値の編集」をクリックします。



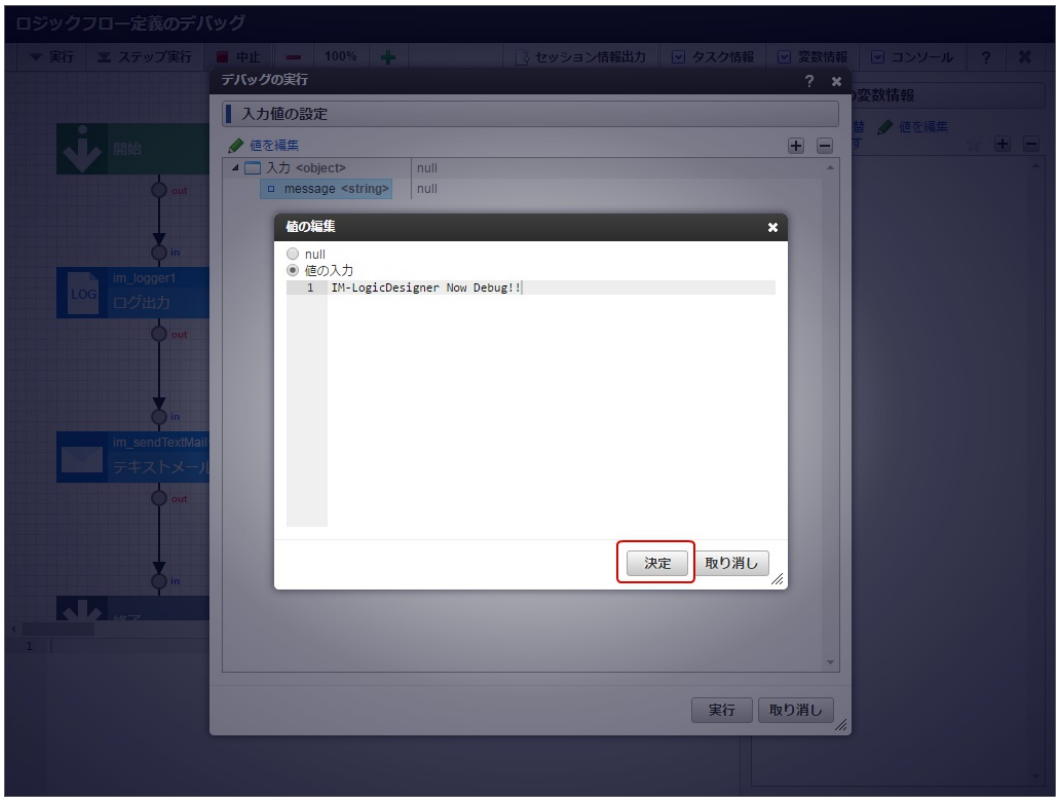

図：値の編集

6. 値の編集を行う画面が表示されるので、入力欄に以下の値を設定します。
   （値の設定を行うことで、自動で画面上部の「値の入力」ラジオボタンがオンになることを確認してください）
   - 設定値 - 「IM-LogicDesigner Now Debug!!」



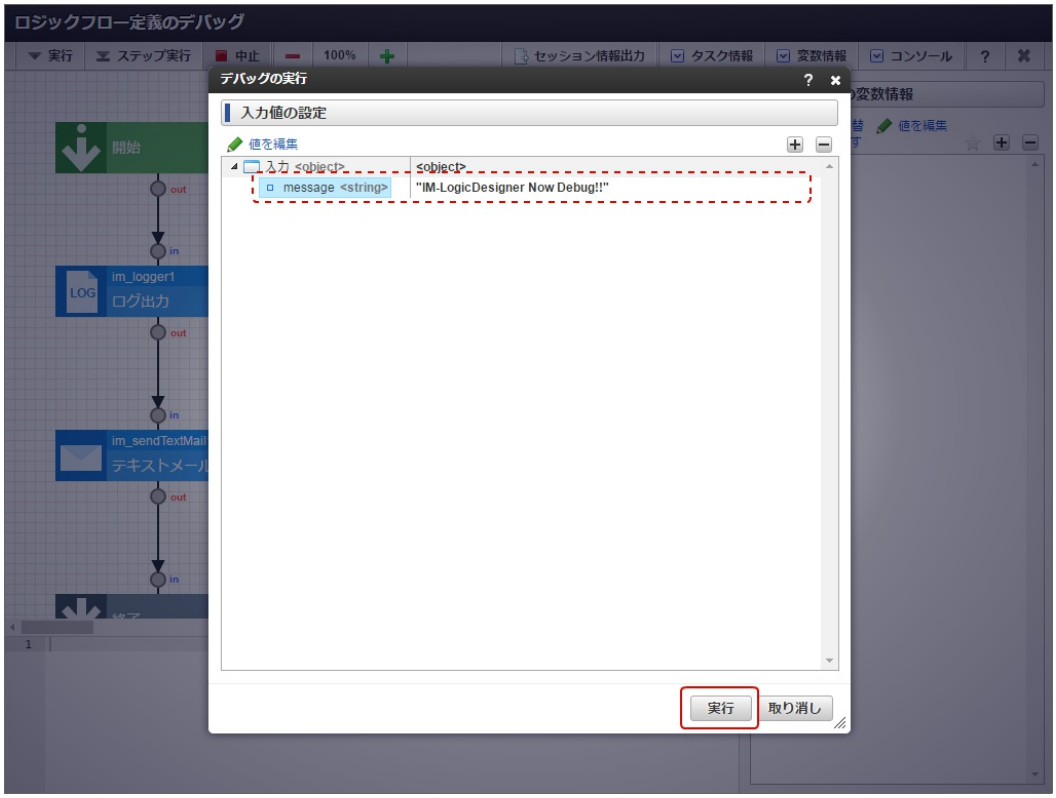

図：デバッグ用設定値の入力

7. 値の編集画面下部の「決定」をクリックします。



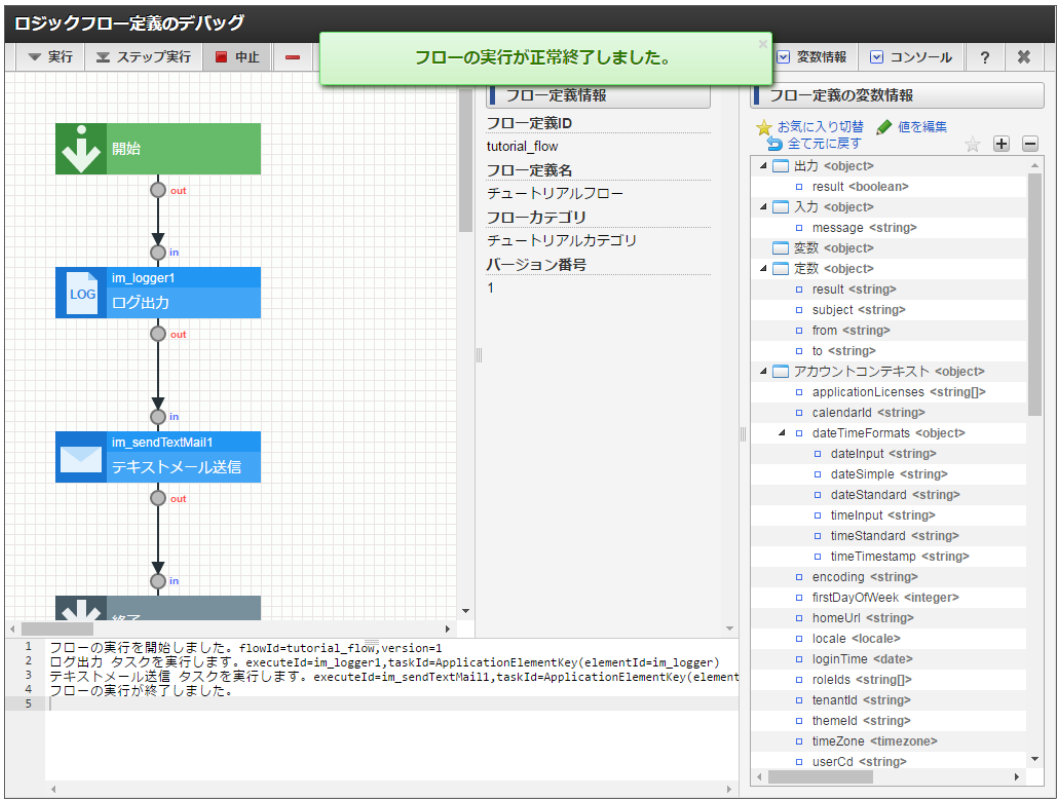

図：設定値の決定

8. message<string>へ、設定した値が反映されていることを確認し、画面下部の「実行」をクリックします。




図：設定の反映と実行

9. デバッグの開始を確認するダイアログが表示されるので、「決定」をクリックします。
10. ロジックフローが実行が完了した旨のメッセージが表示され、実行結果が「フロー定義の変数情報」、および、「コンソール」に反映されます。




図：デバッグ実行の完了

以上で、ロジックフローのデバッグ実行が完了しました。

## デバッグ実行の結果を確認する

「*ロジックフローをデバッグ実行する*」でデバッグ実行した結果を確認します。
デバッグ実行を行った場合、通常のフロー実行時に得られる結果以外に、以下の実行結果が確認できます。

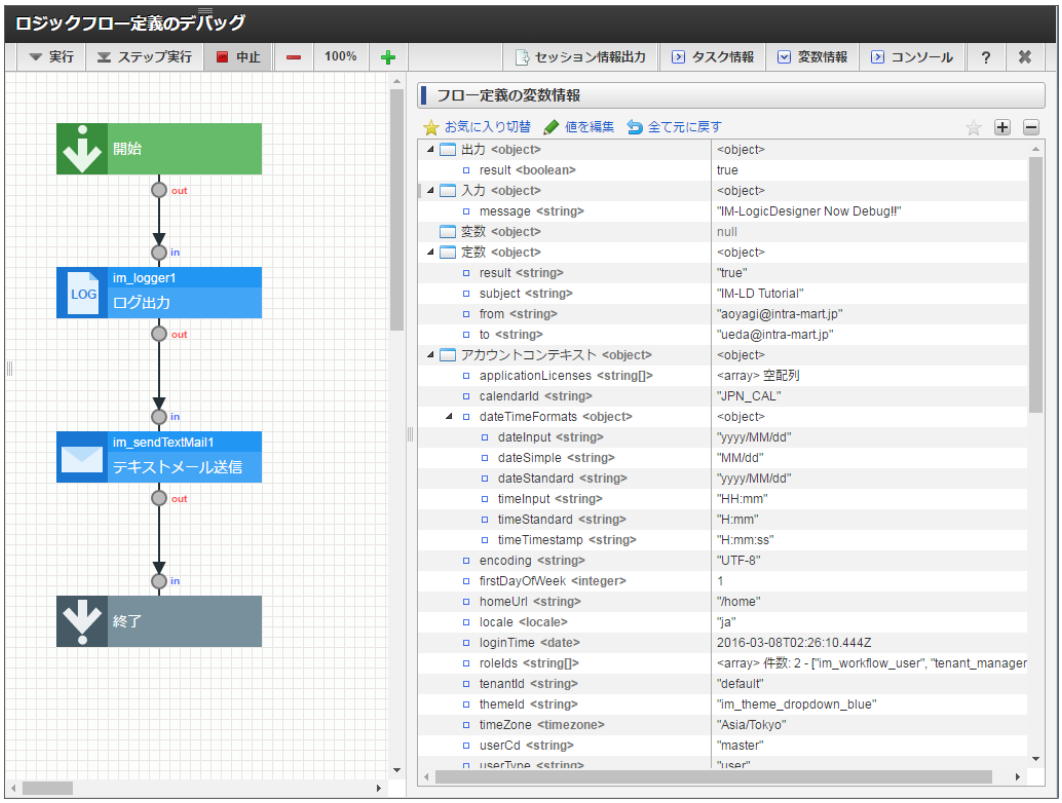
- ロジックフローデバッグ実行時に各種変数へ割り当てられた値
- ロジックフローの実行情報

ここでは、上記二点の確認を行います。

### デバッグ実行時に各種変数へ割り当てられた値の確認

はじめに、実行時に各種変数へ割り当てられた値を確認します。
変数の確認は「フロー定義の変数情報」ペインから行います。

1. フロー定義の変数情報を確認しやすくするため、以下を実施します。
   - 「タスク情報」ペイン、および、「コンソール」ペインの非表示化
   - 「フロー定義の変数情報」ペインの表示領域の拡大

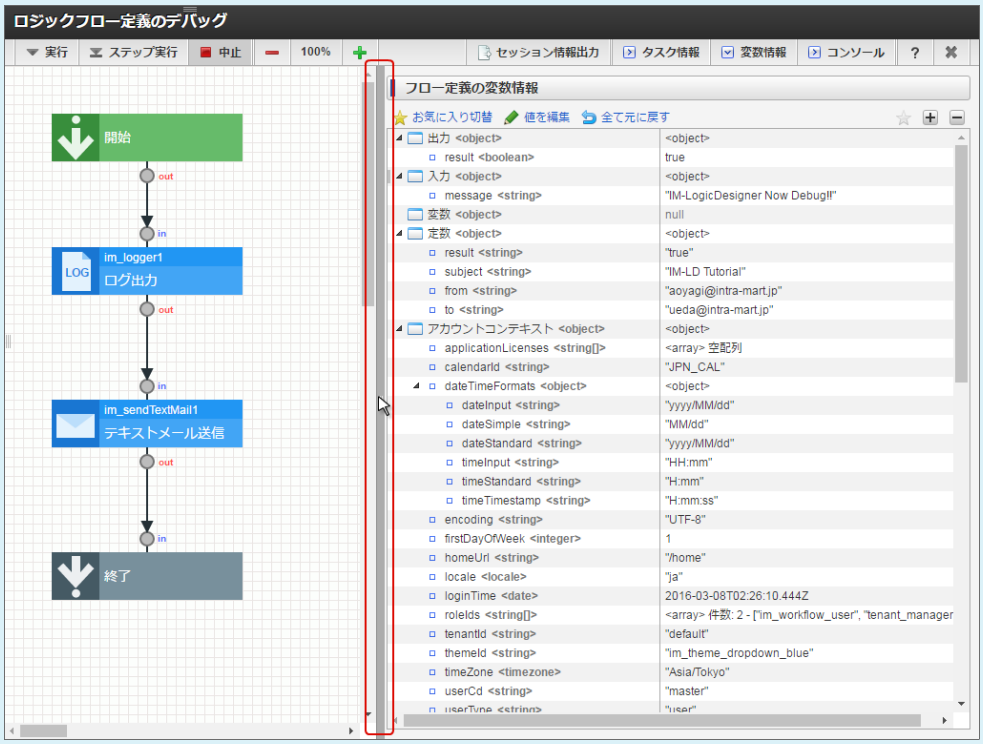



図：フロー定義の変数情報ペインの表示

コラム

表示領域の拡大

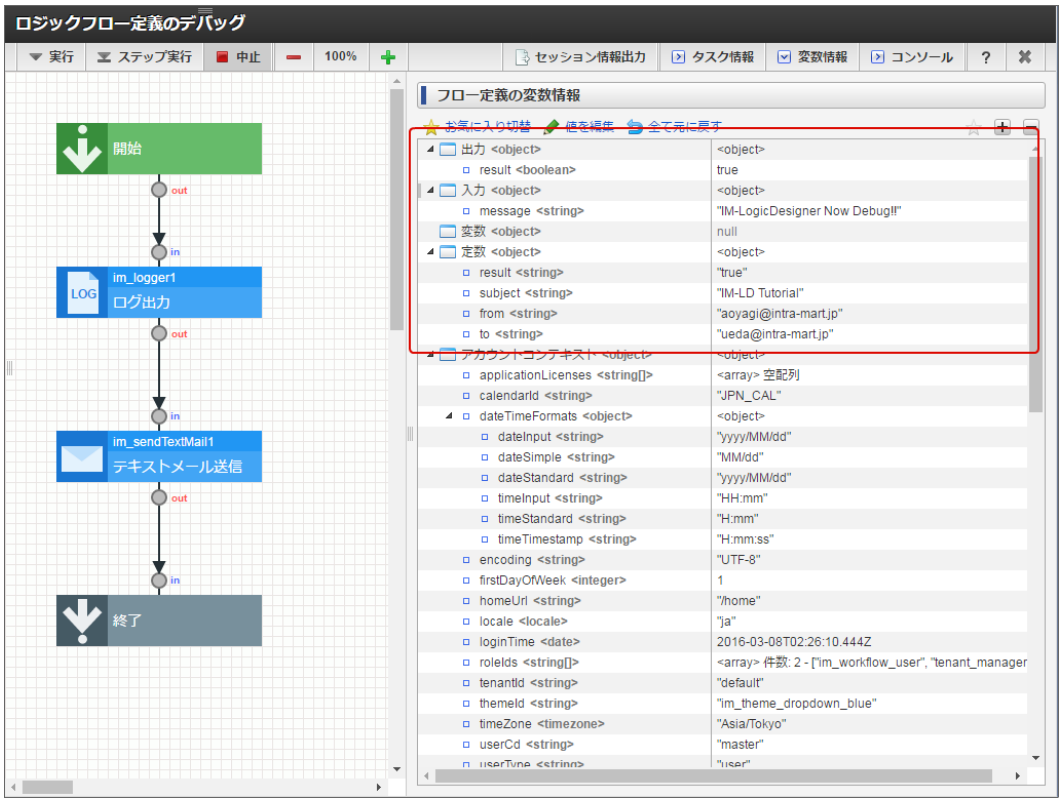
「タスク情報」ペイン、「フロー定義の変数情報」ペイン、および、「コンソール」ペインは各ペインの端をドラッグ&ドロップすることで表示領域を拡大・縮小することができます。
作業環境の解像度によって表示情報が確認しづらい場合などに活用してください。




図：表示域拡大 - 「フロー定義の変数情報」ペインの例

2. 「フロー定義の変数情報」ペイン内において、変数と値が以下のとおり設定されていることを確認してください。
   - 「入力」 - 「*入出力設定を定義する*」で設定した入力値（message）と、「*ロジックフローをデバッグ実行する*」で実行時に指定した設定値。

- 「出力」 - 「*入出力設定を定義する*」で設定した出力値（result）と、正常実行されたことを表すフラグ（true）。
- 「定数」 - 「*定数値を定義する*」で設定した定数値、および、その値。



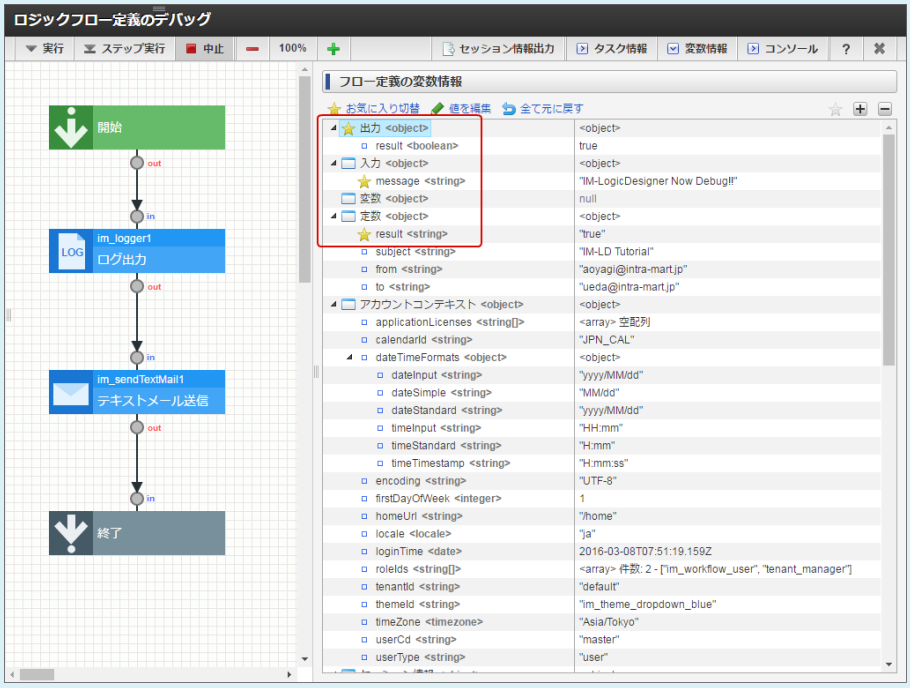

図：変数情報の確認

「フロー定義の変数情報」ペインでは、上記以外に「*暗黙的な変数の利用*」で紹介している暗黙的な変数についても値を確認できます。

コラム

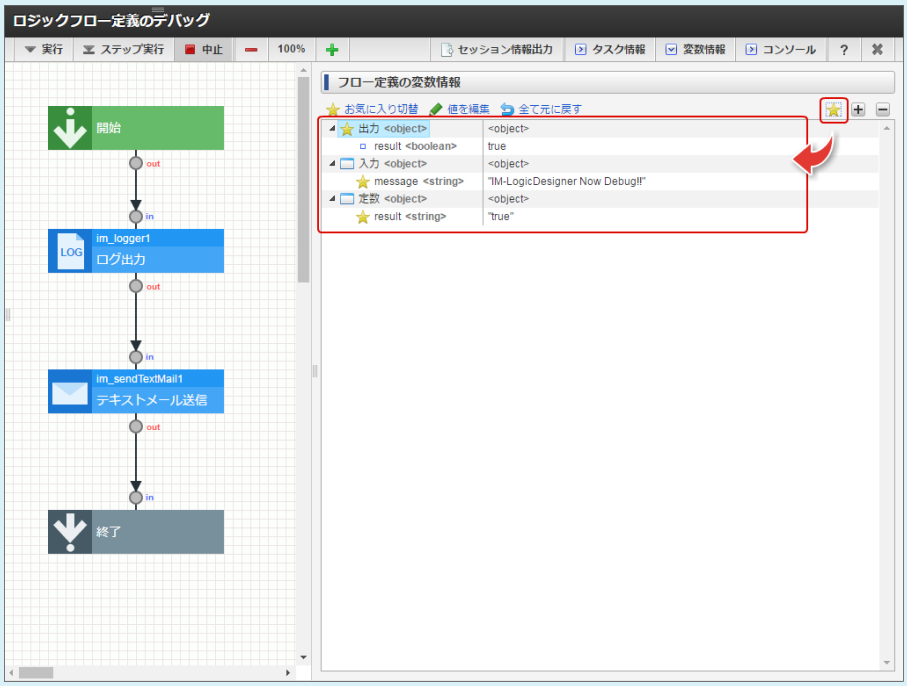
変数情報のお気に入り機能

「フロー定義の変数情報」ペイン内の変数は左端に表示されているアイコンをクリックすることでお気に入りとして設定することが可能です。




図：お気に入り設定

お気に入り設定された変数は、「フロー定義の変数情報」ペインの右上の星アイコンをクリックすることでお気に入り設定された変数のみに絞り込んで表示することができます。




図：お気に入りの絞り込み表示

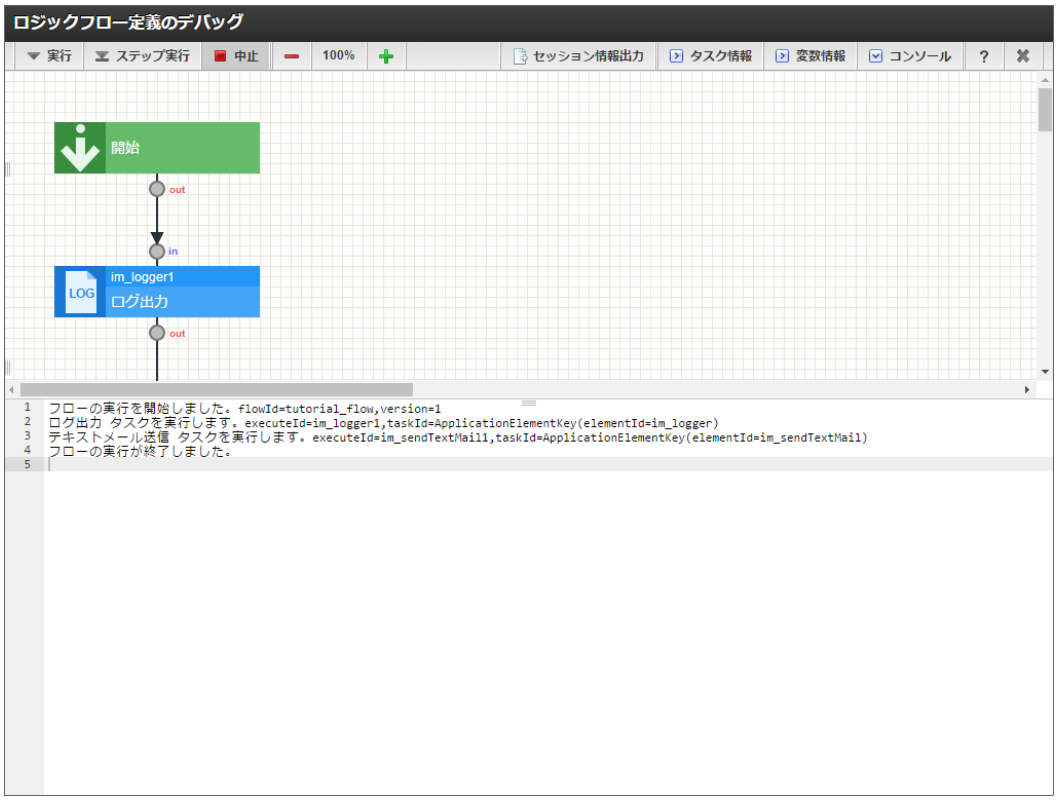
お気に入り設定を利用することで、特に注意して確認したい変数の変化を見逃しづらくなります。

## 実行情報の確認

次に、ロジックフローの実行情報を確認します。
実行情報の確認は「コンソール」ペインから行います。

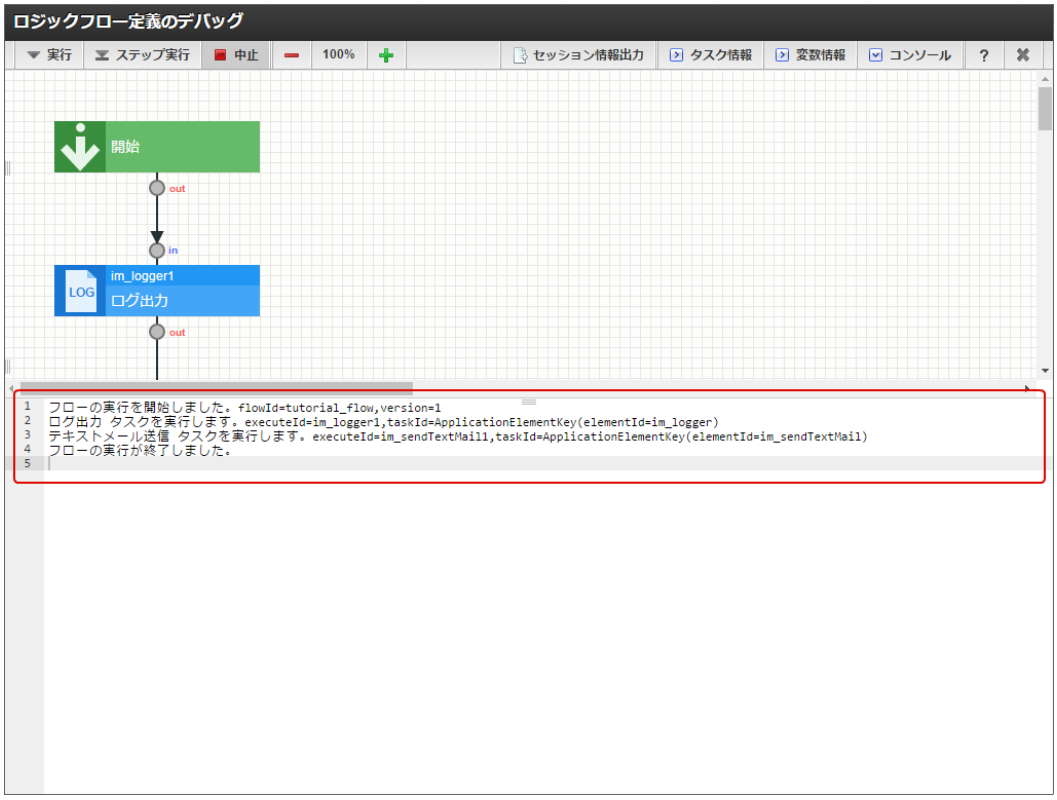
1. フロー定義の実行情報を確認しやすくするため、以下を実施します。
   - 「タスク情報」ペイン、および、「フロー定義の変数情報」ペインの非表示化

- 「コンソール」ペインの表示領域の拡大




図：コンソールペインの表示

2. 「コンソール」ペイン内において、以下の内容が出力されていることを確認してください。
   - 実行されたフローのフローIDとバージョン情報のログ
   - 実行された順に、エレメントの実行IDとタスクIDのログ
   - フローの実行が正常に終了した旨のログ



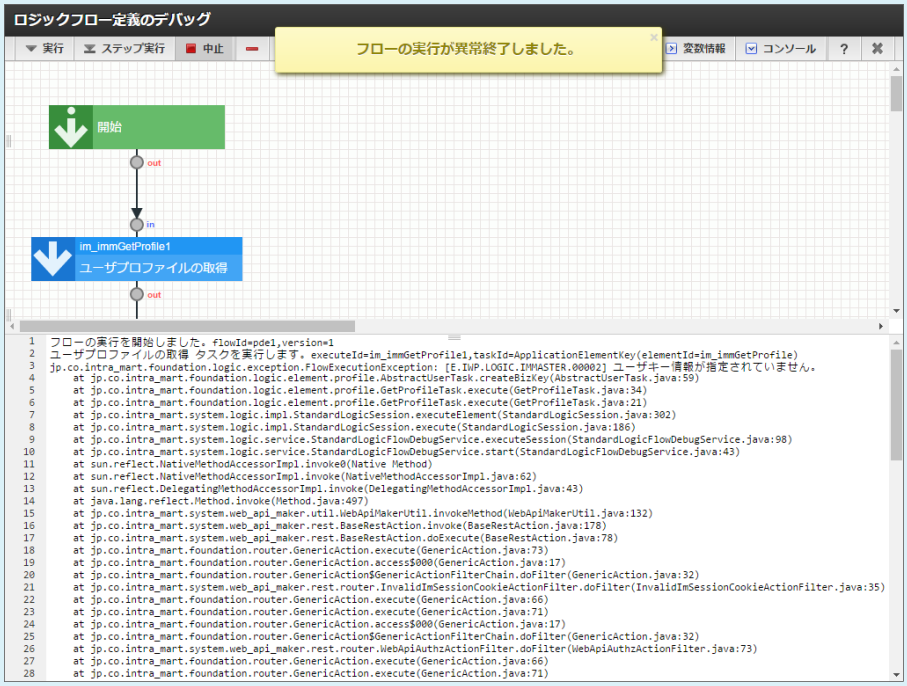

図：実行情報の表示

「コンソール」ペインにはエレメントの情報ログが実行順に出力されるため、特に繰り返し処理や条件分岐と言った複雑な処理経路を辿るフローのデバッグ時に有用です。

コラム

エラーが発生した場合のコンソール

デバッグ実行時にエラーが発生した場合、発生したエラーに関するスタックトレースがコンソールに表示されます。




図：コンソールにスタックトレースが表示される例

上記画像の例は、ユーザプロファイルを取得するためのユーザコードが指定されていなかった場合のスタックトレースです。

以上で、ロジックフローのデバッグ実行の結果が確認できました。

## ロジックフローを順番にデバッグ実行する（ステップ実行）

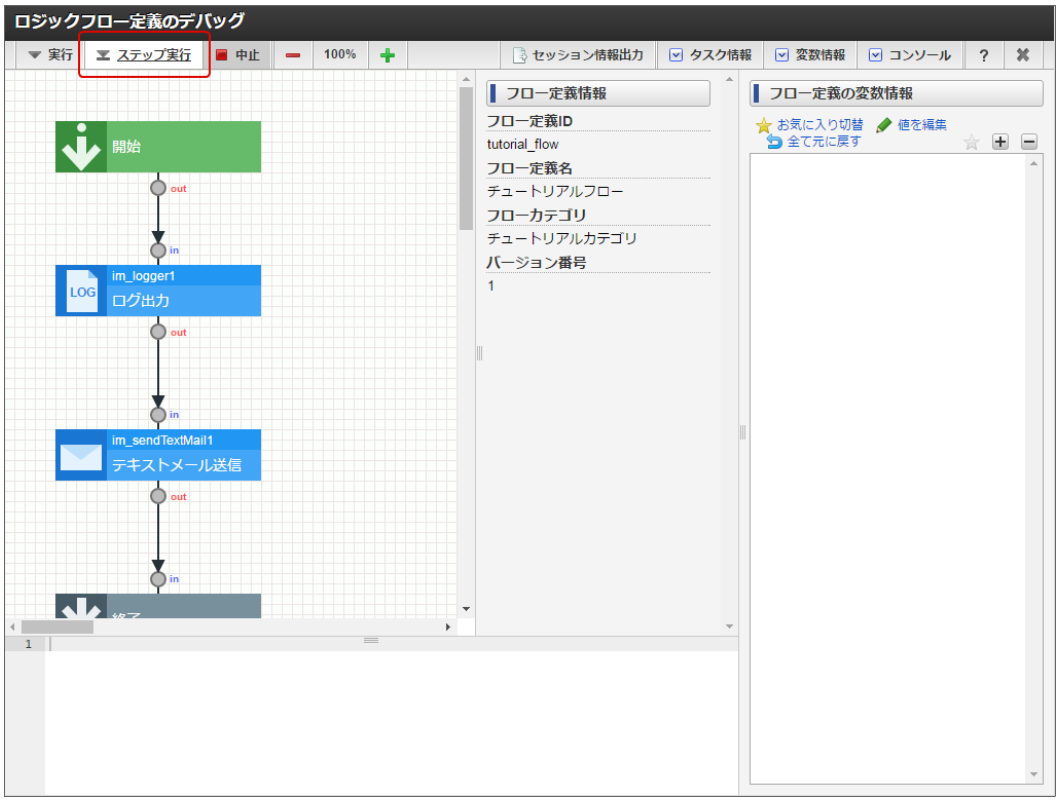
ロジックフローのデバッグ機能では通常実行の他に、処理をエレメントごとにサスペンドしながらデバッグを行うステップ実行をサポートしています。

ステップ実行を利用することで、配置した各エレメントに想定した値が入力/出力されているか 1 つずつ確認しながらデバッグを行う事が可能です。
これは、これまで利用したことのないタスクやユーザ定義についての、エレメントレベルでの処理結果も踏まえた検証作業等に有用です。

### ロジックフローのステップ実行

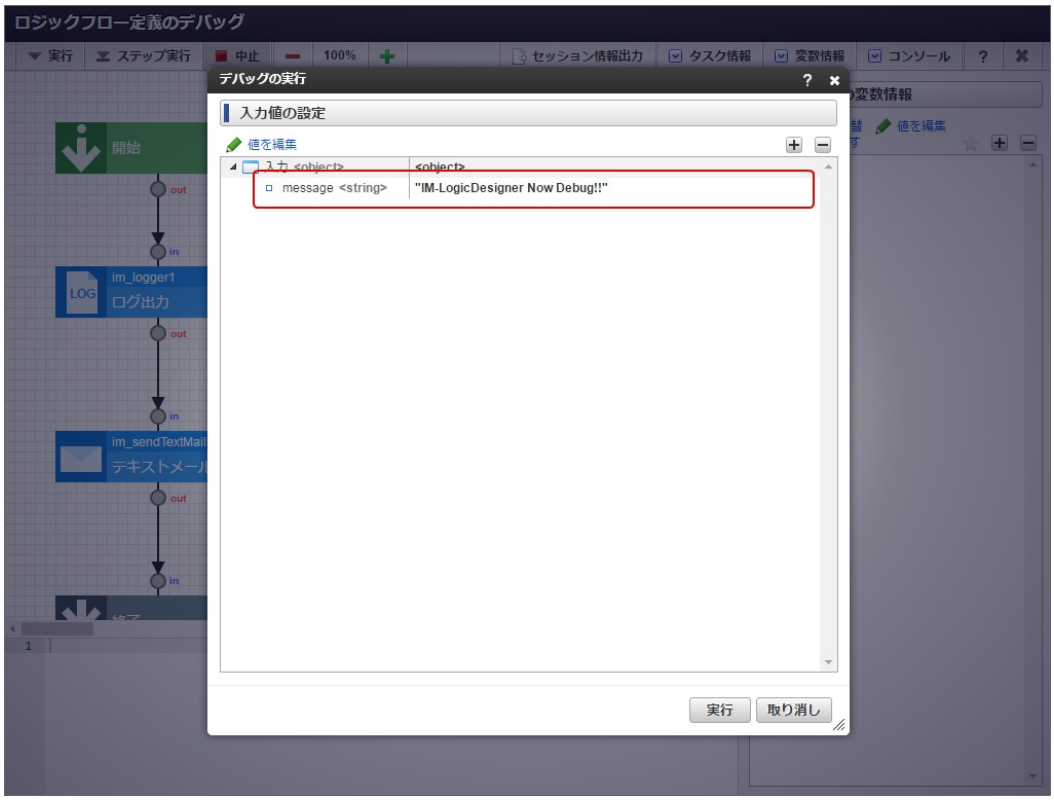
実際にロジックフローをステップ実行を利用してデバッグする方法を説明します。

1. デバッグ画面を開きます。
2. メニューヘッダ内の「ステップ実行」をクリックします。



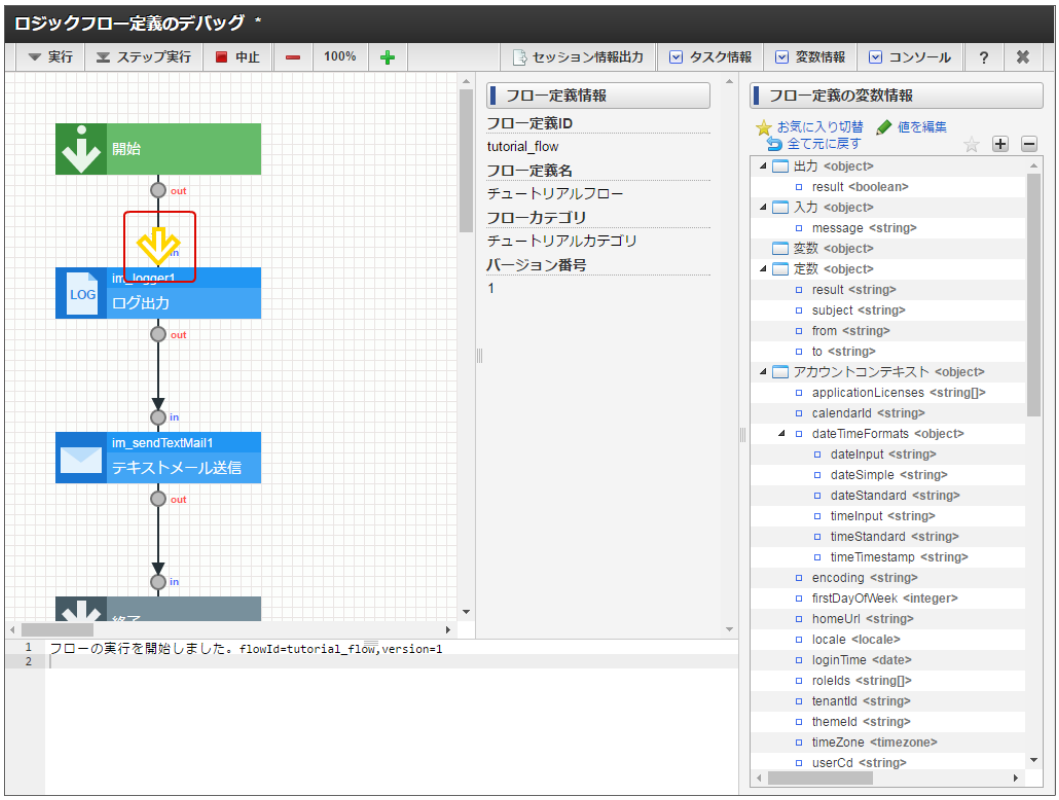

図：ヘッダ内の「ステップ実行」

3. デバッグ実行を行う上で必要な「入力値の設定」項目が表示されます。
   ここでは、「*ロジックフローをデバッグ実行する*」と同様の手順で値を設定します。



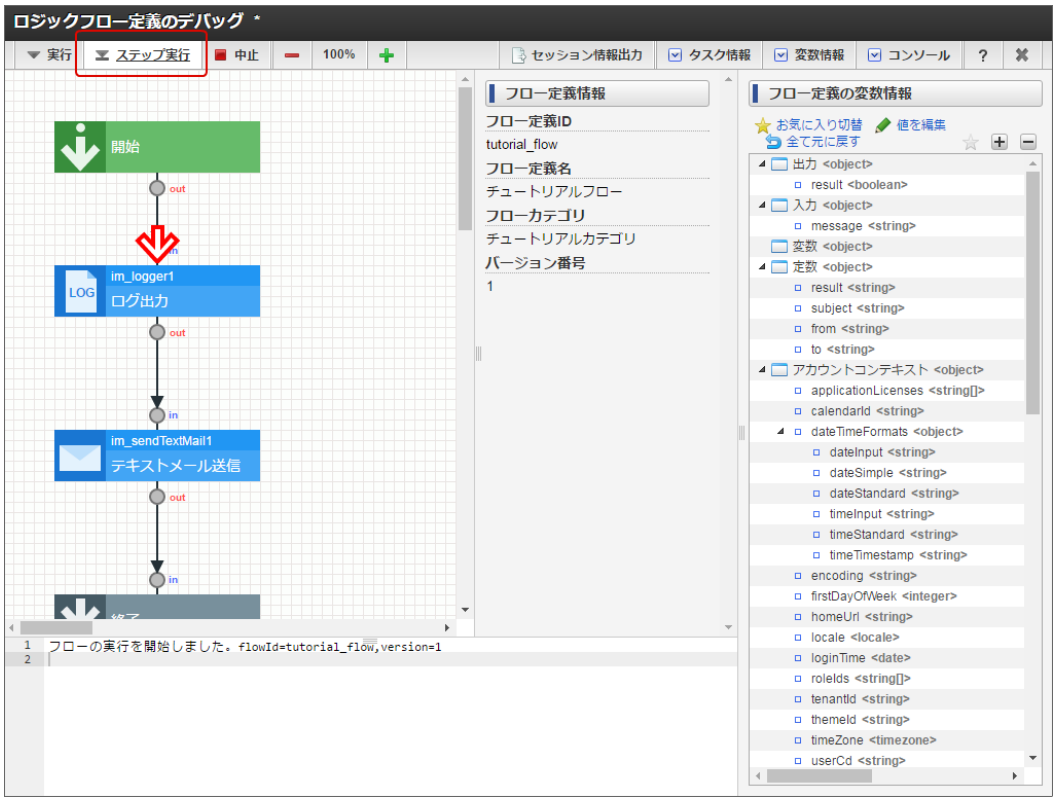

図：デバッグ用設定値の入力

4. 画面下部の「実行」をクリックします。
5. デバッグの開始を確認するダイアログが表示されるので、「決定」をクリックします。
6. ロジックフローが実行され、「開始」制御要素と「ログ出力」タスクの間で処理がサスペンドしたことを表す矢印が表示されます。



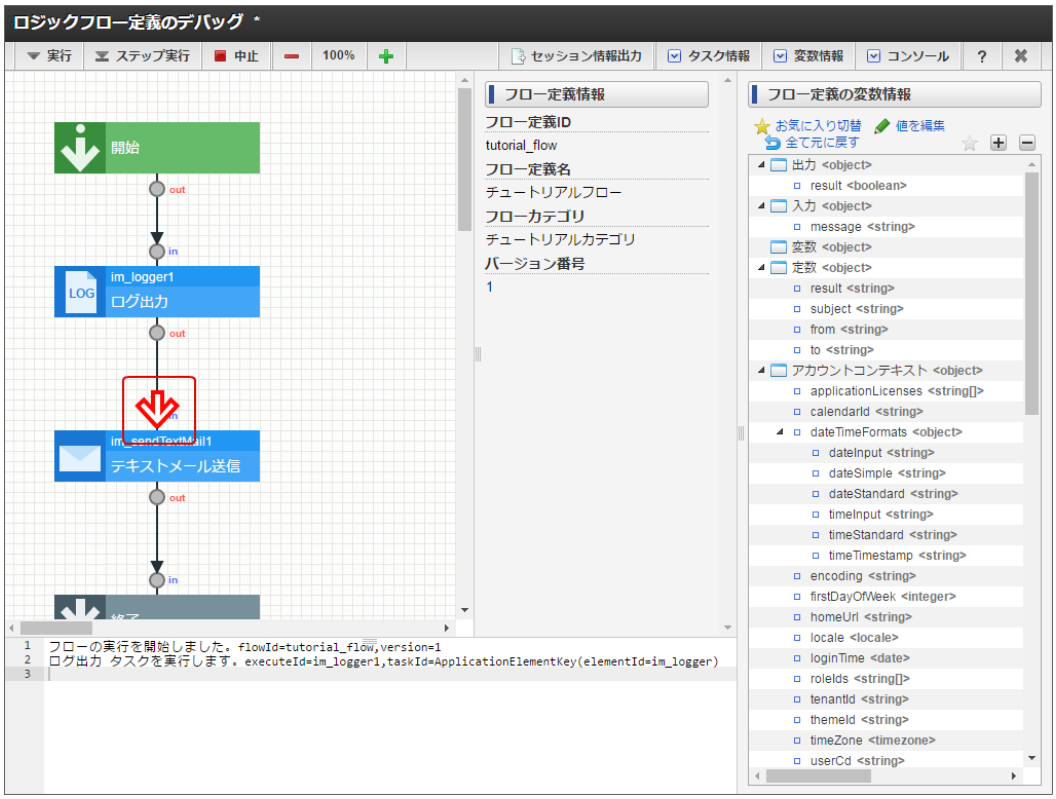

図：現在サスペンドしている位置の表示（矢印）

7. 処理が「ログ出力」タスクの直前でサスペンドしていることを以下より確認してください。
   - 「コンソール」ペインには、フローが実行されたことを示すログのみが出力されている。
   - サーバのコンソールには、「ログ出力」タスクによるログが出力されていない。
8. 再度「ステップ実行」をクリックします。




図：「ステップ実行」のクリック（２回目）

9. ロジックフローが再実行され、「ログ出力」タスクと「テキストメール送信」タスクの間に処理がサスペンドしたことを表す矢印が移動します。



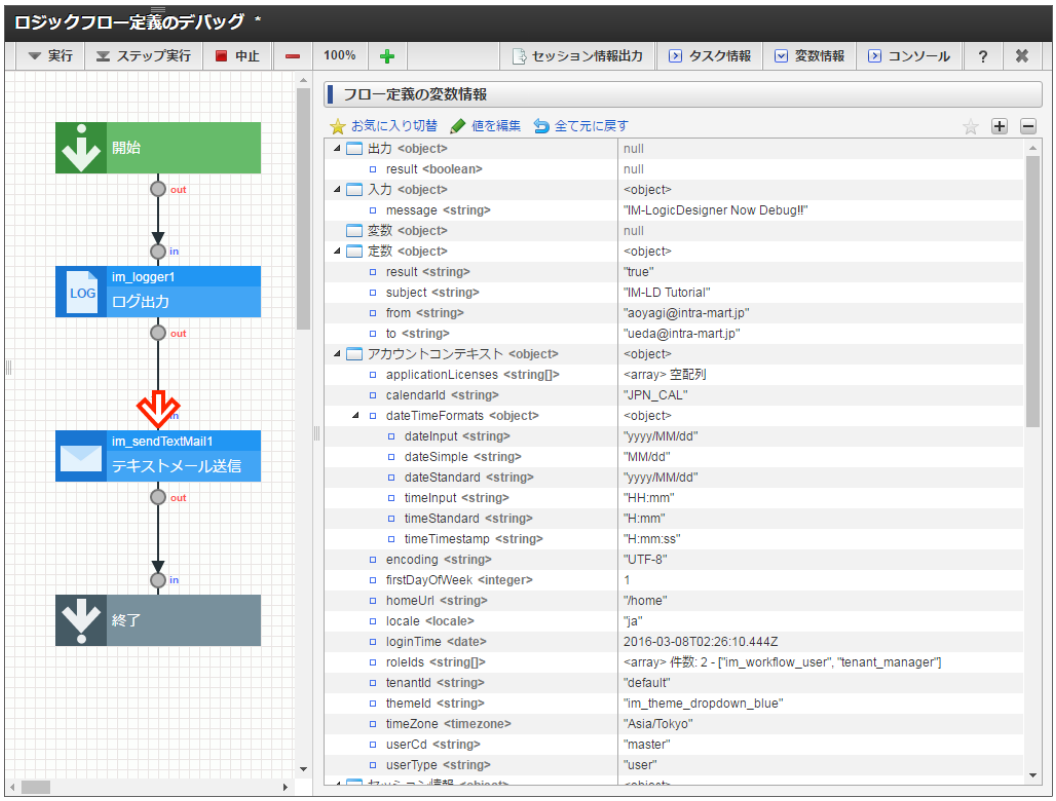

図：現在サスペンドしている位置の表示（矢印）

10. 処理が「テキストメール送信」タスクの直前でサスペンドしていることを以下より確認してください。
    - 「コンソール」ペインには、「ログ出力」タスクの実行までのログが出力されている。
    - サーバのコンソールに、入力値として設定した「IM-LogicDesigner Now Debug!!」が表示されている。
    - 「テキストメール送信」タスクに関する処理結果が確認されていない。

### 実行途中での変数の編集

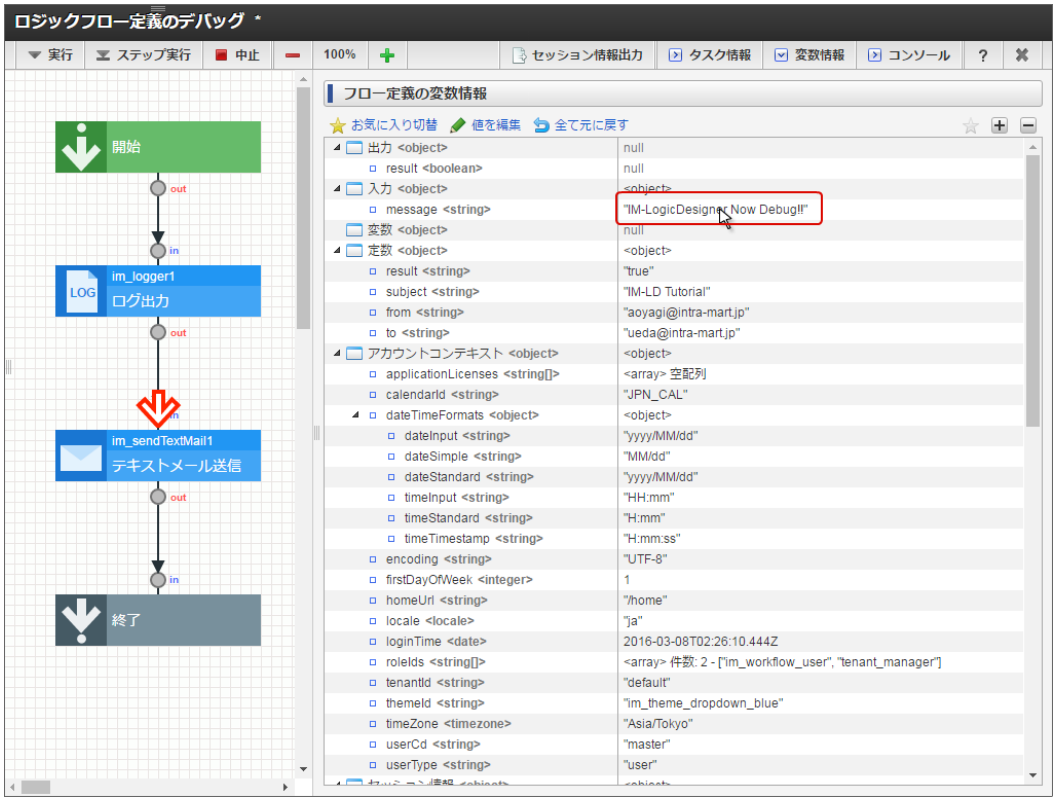
ロジックフローのデバッグ機能では、実行途中で特定の変数の値を編集することが可能です。
ここでは、現在ロジックフロー上に流れている入力値（message）の値を変更し、それが後続の処理に反映されることを確認します。

1. 「フロー定義の変数情報」ペインを表示します。



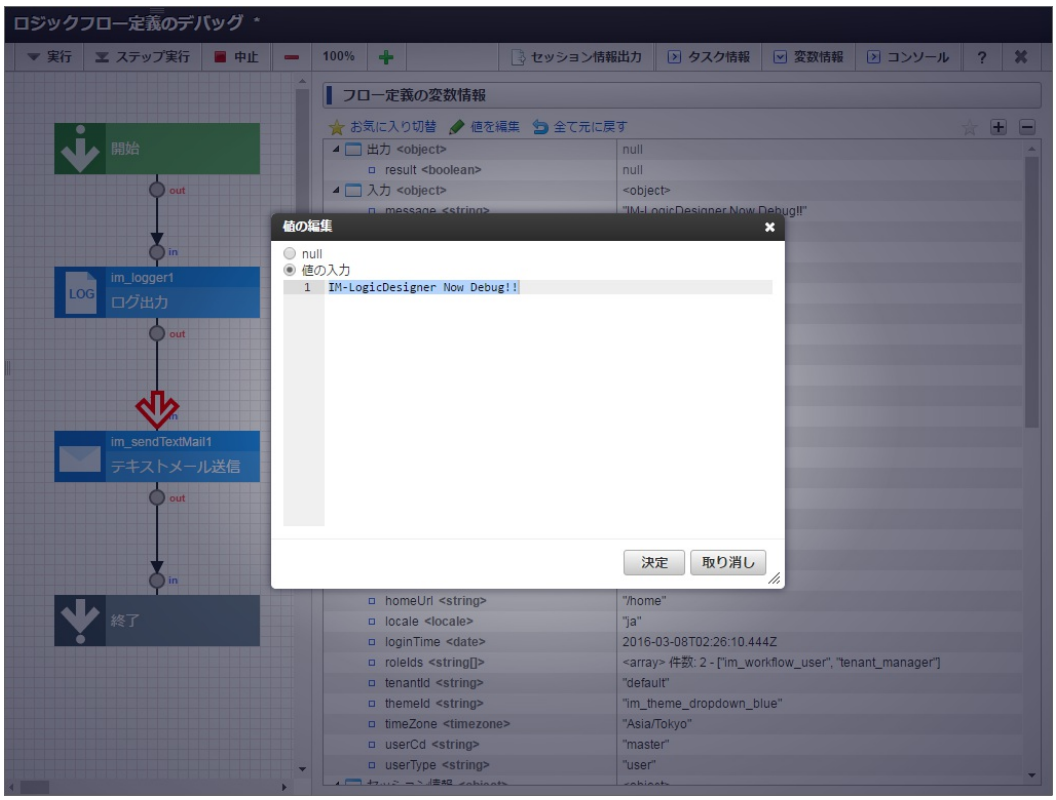
図：ヘッダ内の「ステップ実行」

2. 入力に設定されている、「IM-LogicDesigner Now Debug!!」文字列をダブルクリックします。



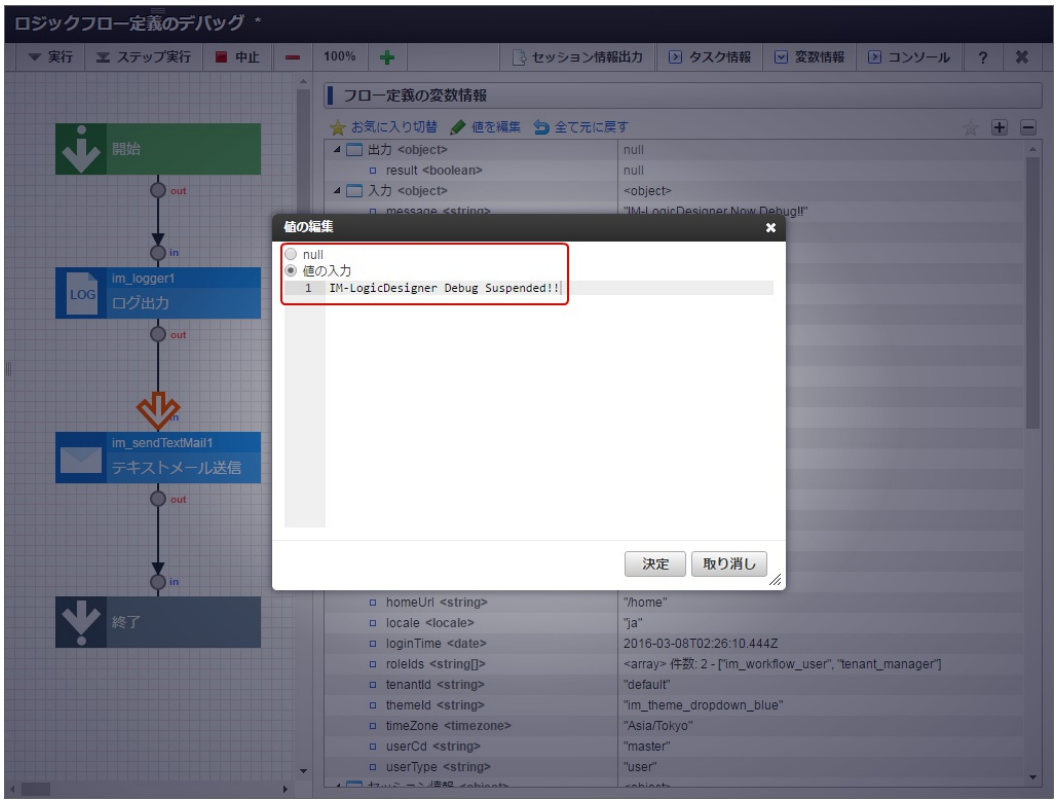

図：値の編集操作

3. 「値の編集」画面が表示されます。




図：値の編集画面

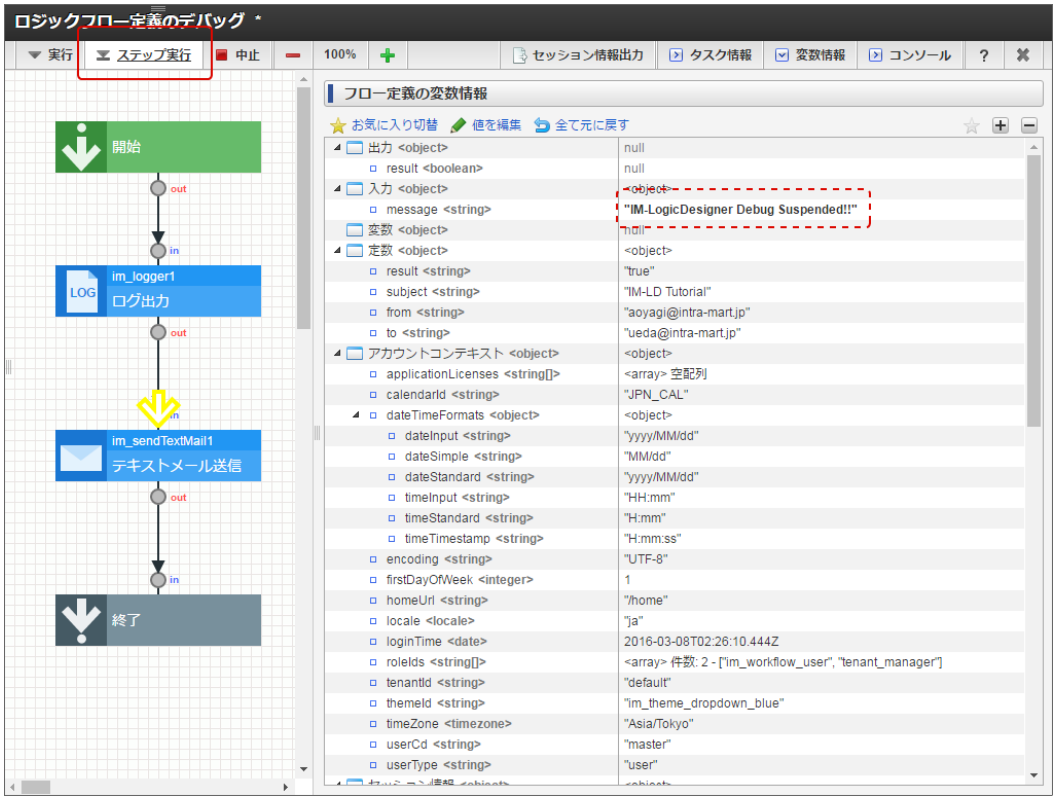
4. 入力欄に以下の値を設定します。
   - 設定値 - 「IM-LogicDesigner Debug Suspended!!」




図：変更する値の設定

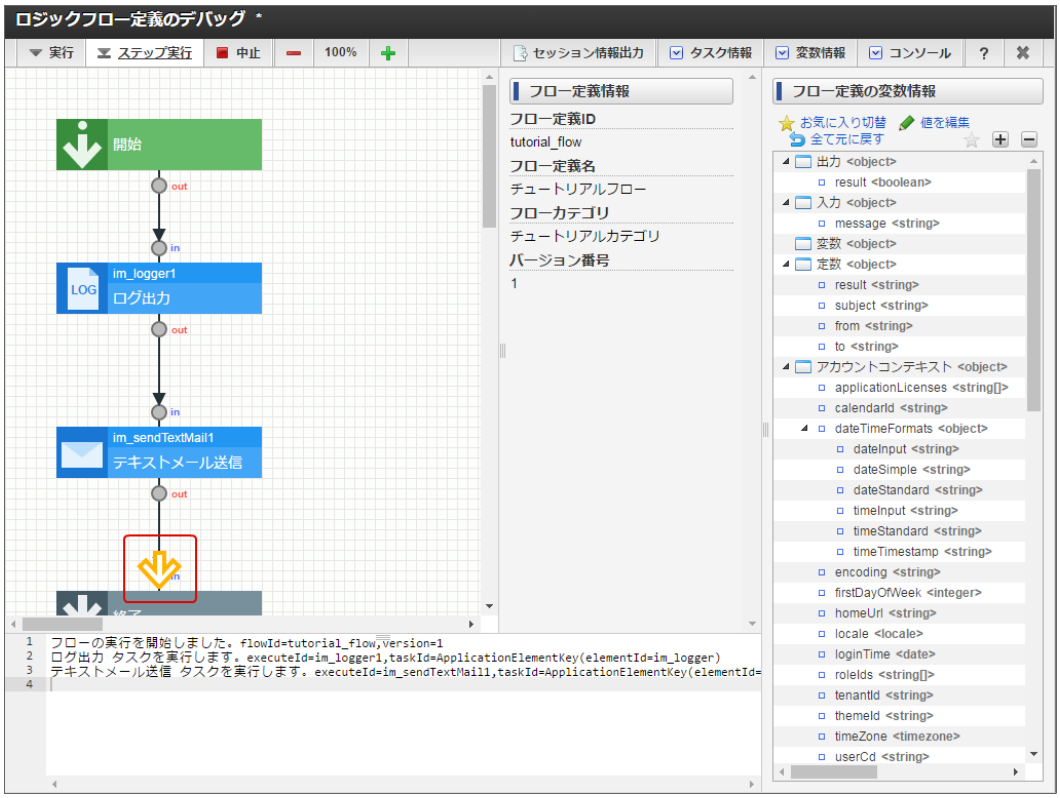
5. 値の編集画面下部の「決定」をクリックします。

6. message<string>へ、設定した値が反映されていることを確認し、再度「ステップ実行」をクリックします。



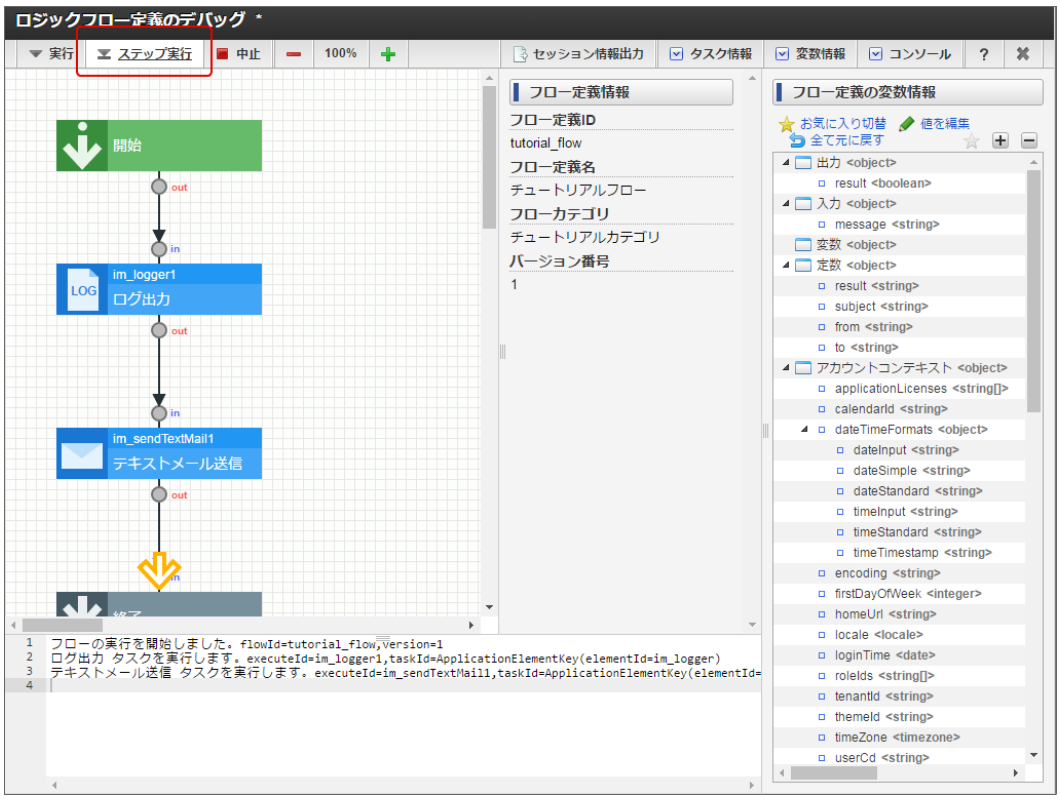

図：「ステップ実行」のクリック（３回目）

7. ロジックフローが再実行され、「テキストメール送信」タスクと「終了」制御要素の間に処理がサスペンドしたことを表す矢印が移動します。



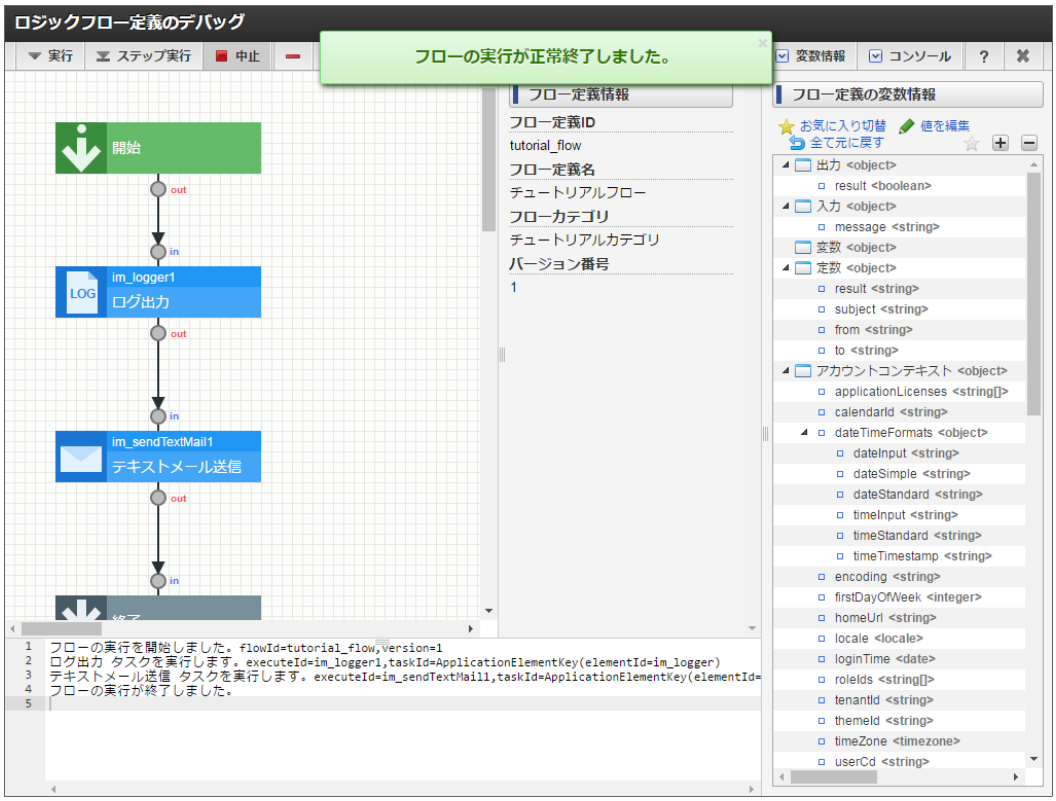

図：現在サスペンドしている位置の表示（矢印）

8. 処理が「終了」制御要素の直前でサスペンドしていることを以下より確認してください。
   - 「コンソール」ペインには、「テキストメール送信」タスクの実行までのログが出力されている。
   - テキストメールが送信され、その本文には「IM-LogicDesigner Debug Suspended!!」の文字列が設定されていること。
9. 再度「ステップ実行」をクリックします。




図：「ステップ実行」のクリック（４回目）

10. 全ての実行処理が完了し、画面上部にロジックフローが実行が完了した旨のメッセージが表示されます。




図：ステップ実行の完了

> **コラム**
>
> メールの送信環境が無い場合
>
> メールの送信環境が無い場合は、「ログ出力」タスクの実行前に値を変更して動作の確認を行ってください。

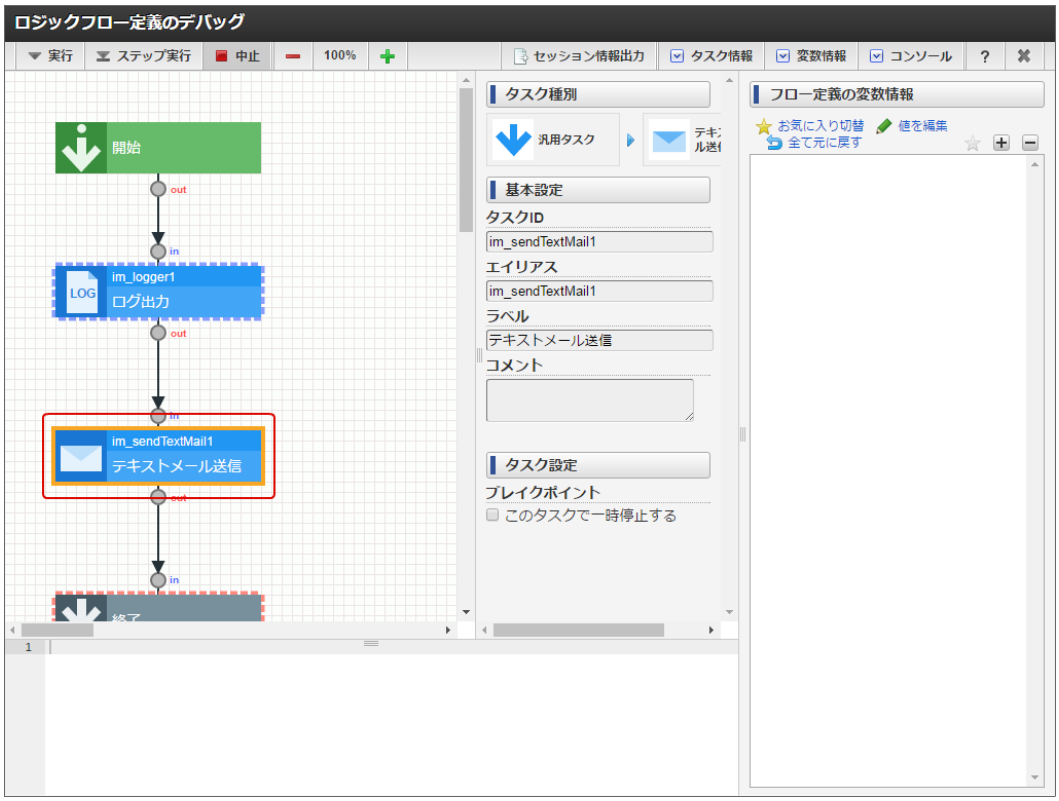
以上で、ステップ実行を用いたロジックフローのデバッグ実行が完了しました。

## 応用：ブレイクポイントを利用したデバッグ実行

長大なロジックフローにおける一部のエレメントの動作を検証したい場合、ブレイクポイントを利用したデバッグ実行を検討してください。
ブレイクポイントを利用することで、任意の位置でデバッグをサスペンドすることができます。

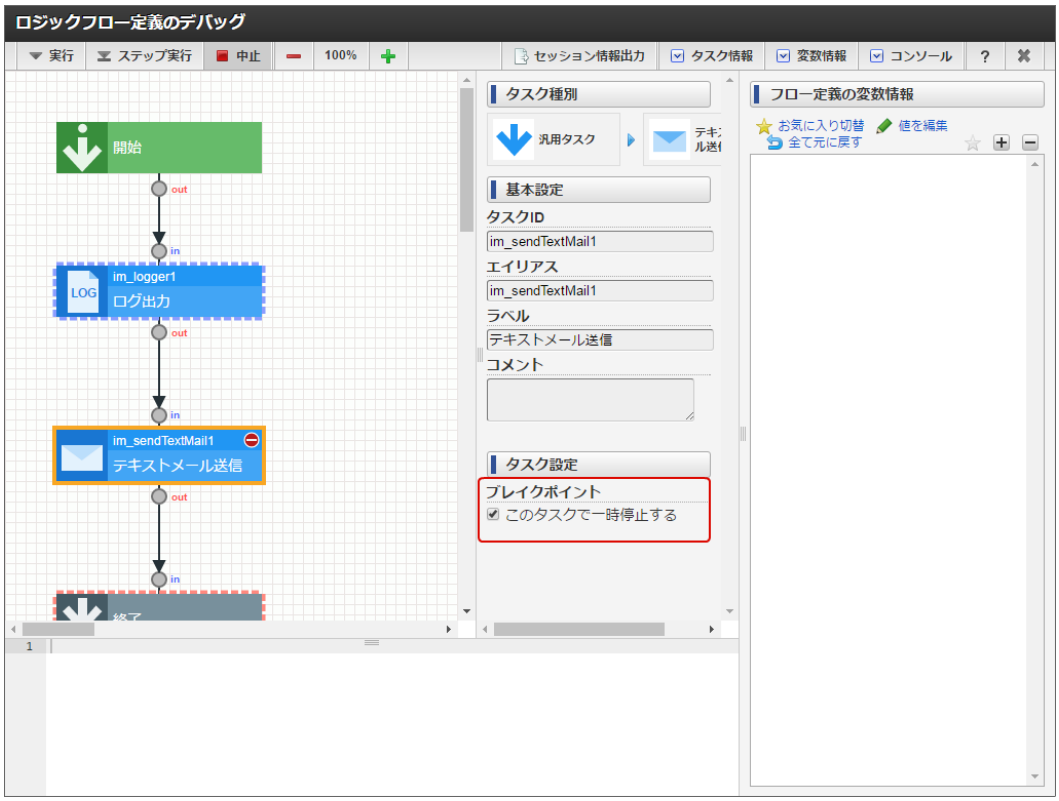
ここでは、ロジックフロー内の「テキストメール送信」タスクにブレイクポイントを設定し、デバッグ実行時に処理が「テキストメール送信」タスク実行直前でサスペンドすることを確認します。

1. デバッグ画面を開きます。
2. 「ロジックフロー確認」ペイン上の「テキストメール送信」タスクを選択します。



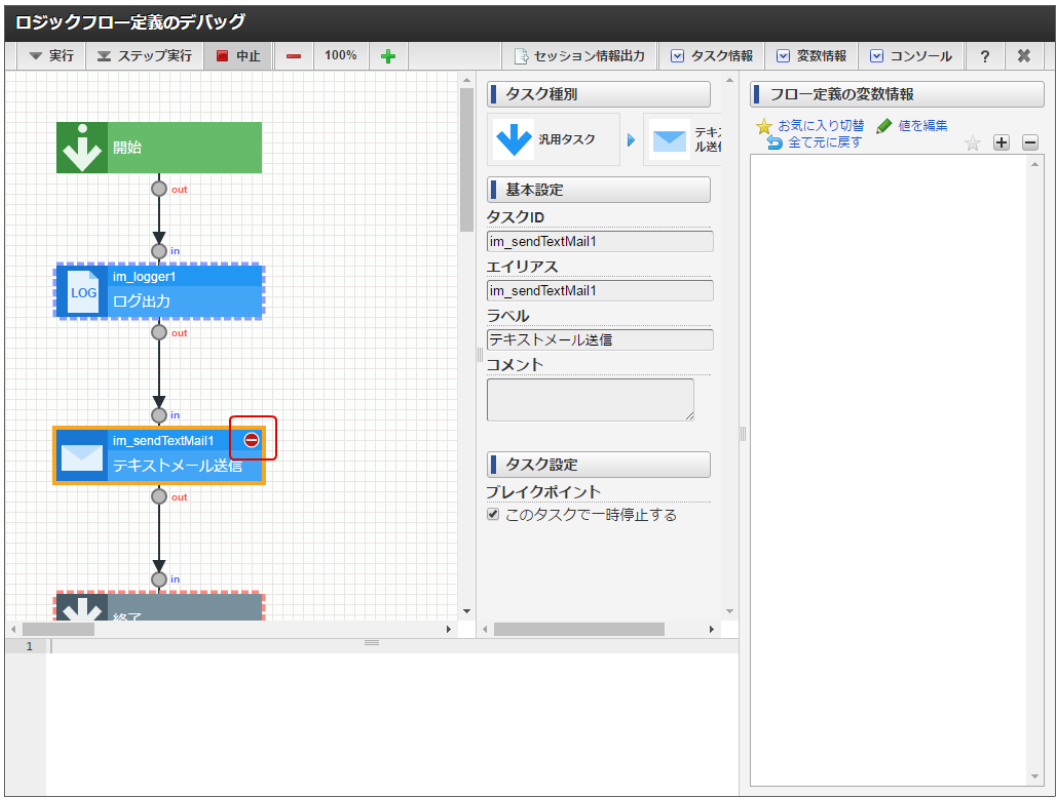

図：「テキストメール送信」タスクの選択

3. 「タスク情報」ペイン下部、タスク設定の項目を以下のとおりに変更します。
   - ブレイクポイント - このタスクで一時停止する - チェックボックスをオンにする




図：ブレイクポイントの設定

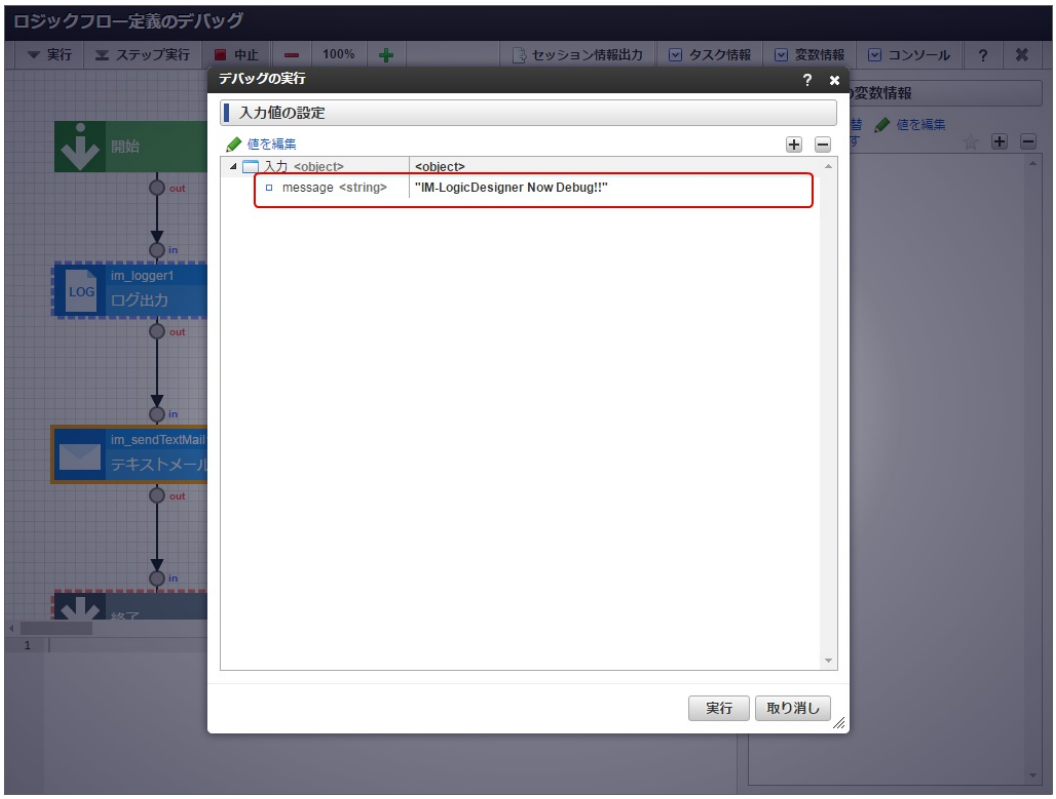
4. 「テキストメール送信」タスクの右上に、ブレイクポイントが設定されたことを表す停止マークが設定されたことを確認してください。




図：停止マーク

> **コラム**
>
> ブレイクポイントの設定
>
> ブレイクポイントは設定を行いたいエレメントをダブルクリックすることでも設定可能です。

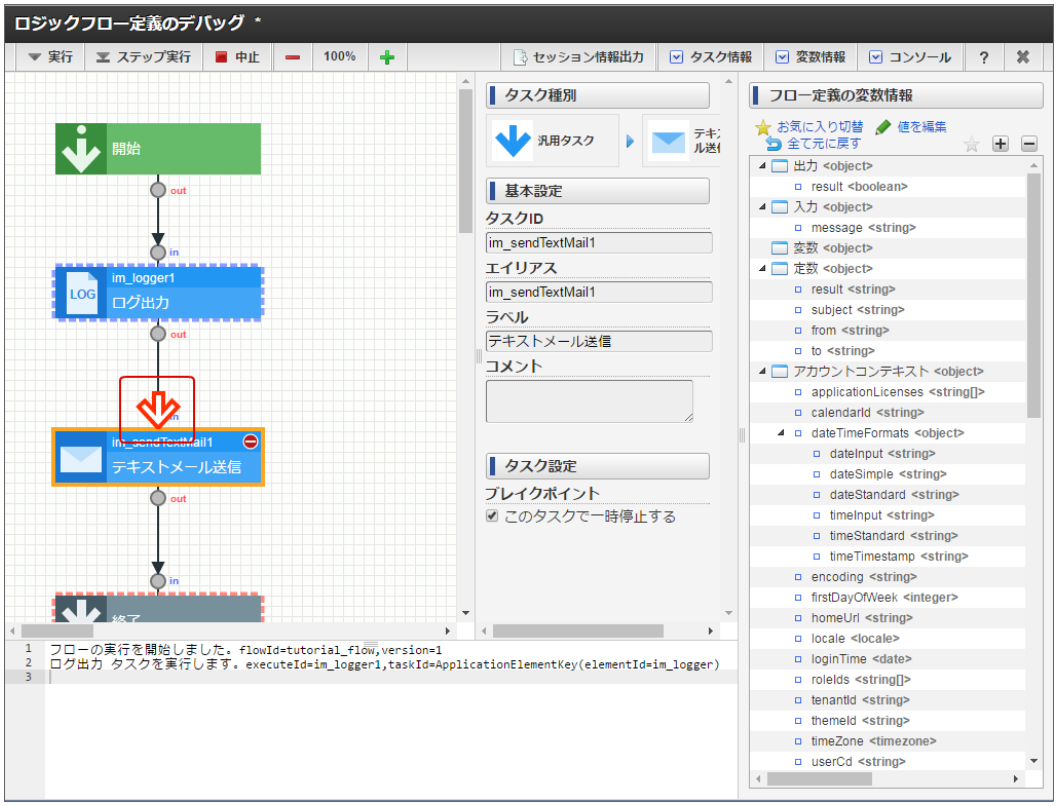
5. メニューヘッダ内の「実行」をクリックし、「*ロジックフローをデバッグ実行する*」と同様の手順で値を設定します。




図：デバッグ用設定値の入力

6. 画面下部の「実行」をクリックします。
7. デバッグの開始を確認するダイアログが表示されるので、「決定」をクリックします。

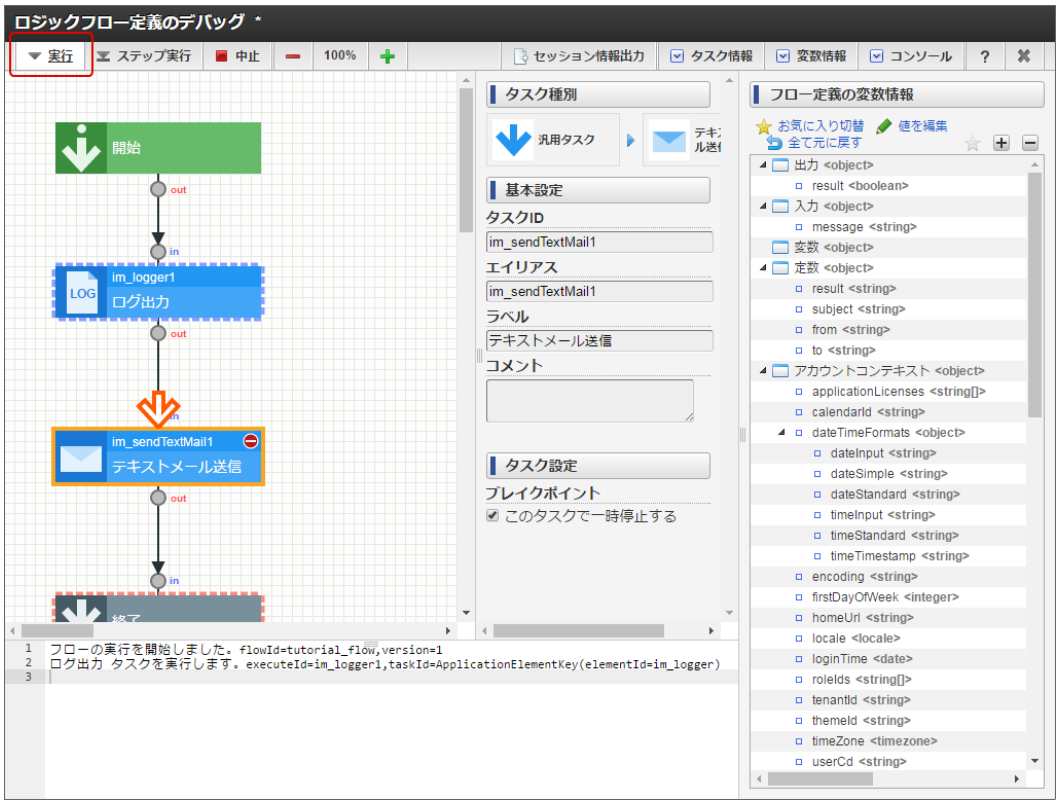
8. ロジックフローが実行され、「ログ出力」タスクと「テキストメール送信」タスクの間に処理がサスペンドしたことを表す矢印が表示されます。




図：ブレイクポイントにより処理がサスペンド

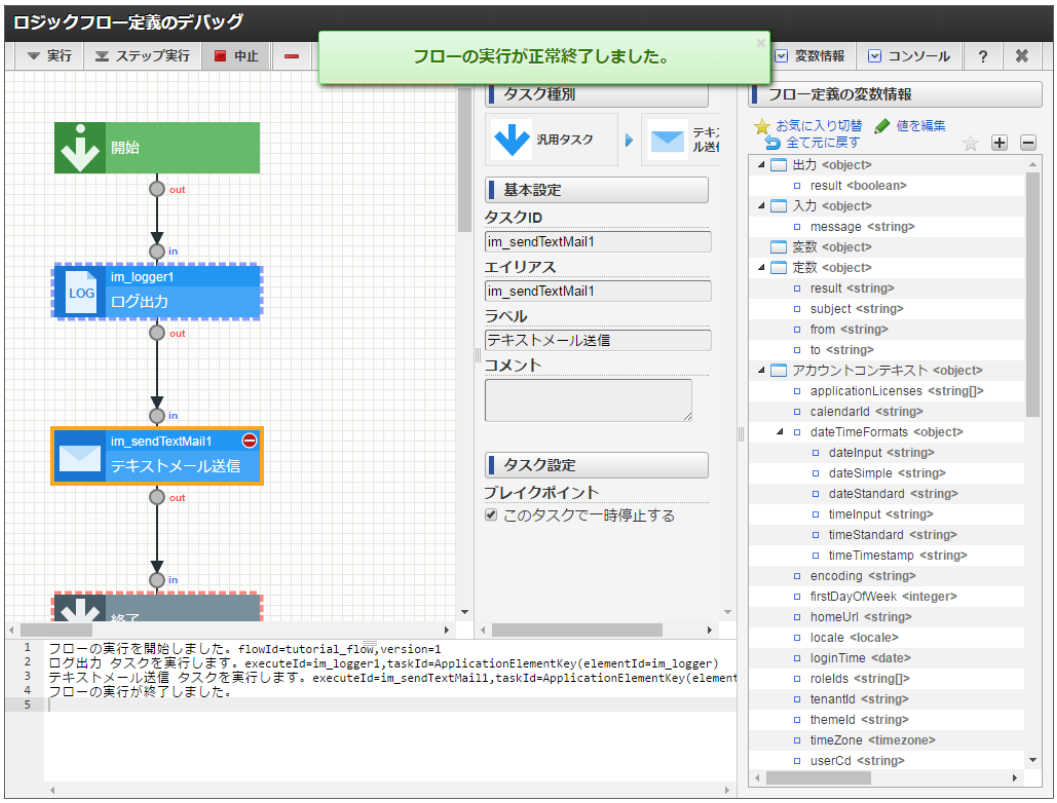
9. 処理が「テキストメール送信」タスクの直前でサスペンドしていることを以下より確認してください。
   - 「コンソール」ペインには、「ログ出力」タスクの実行までのログが出力されている。
   - サーバのコンソールに、入力値として設定した「IM-LogicDesigner Now Debug!!」が表示されている。
   - 「テキストメール送信」タスクに関する処理結果が確認されていない。

10. 再度「実行」をクリックします。




図：「実行」をクリック（2回目）

11. 全ての実行処理が完了し、画面上部にロジックフローが実行が完了した旨のメッセージが表示されます。




図：ブレイクポイントを用いたフロー実行の完了

以上で、ブレイクポイントを用いたロジックフローのデバッグ実行が完了しました。

## ロジックフローのエラーハンドリング

この章では、ロジックフローにおいてエラーハンドリングを行う方法を説明します。



### ロジックフローのエラーハンドリングとは

ロジックフローのエラーハンドリングは大別して以下の二つに分類されます。

- エレメントでエラーが発生した場合、以降の処理を継続するかを決定する（エラー発生時のハンドリング）。
- 自身より前に処理が行われたエレメントのエラー状況をもとに、自身の振る舞いを制御する（エラー発生後のハンドリング）。

この章では、上記の分類をもとに以下の説明を行います。

- エラー発生時のハンドリング - エレメントに対するエラーハンドリングの設定方法。
- エラー発生後のハンドリング - 受け取ったエラー状況（IM-LogicDesignerではこれを**処理結果情報**と呼びます）をもとに振る舞いを変更する方法。

### エラーハンドリングの設定、および、動作確認

ここでは、エラー発生時のハンドリングの設定方法の説明と、動作の違いの確認を行います。

具体的にはエレメントに対して、処理中にエラーが発生した場合に以下の二通りのエラー処理のどちらを選択するかを設定します。

- 処理を継続する
- フロー全体のエラーとして処理を中断する

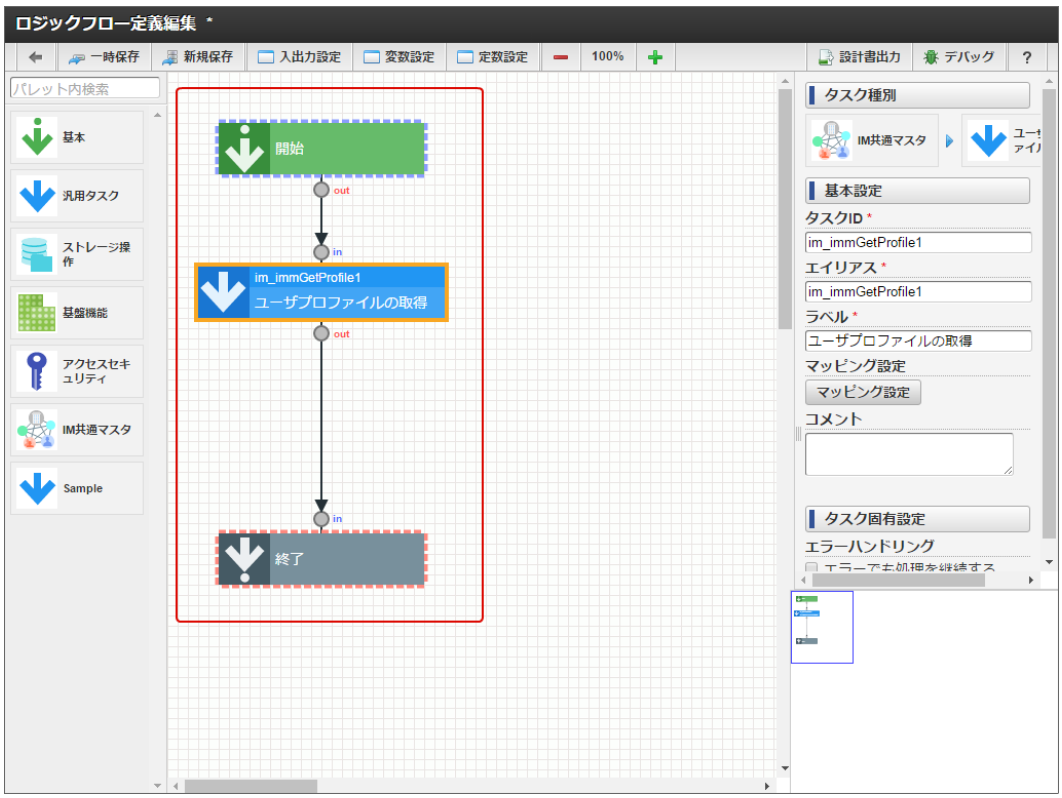
そして、それぞれにおいてどのように動作が異なるかを実際にエラーを発生させて確認します。

フローの準備

はじめに、エラーハンドリングが行われることを確認するため、エラーが発生する簡単なロジックフローを準備します。

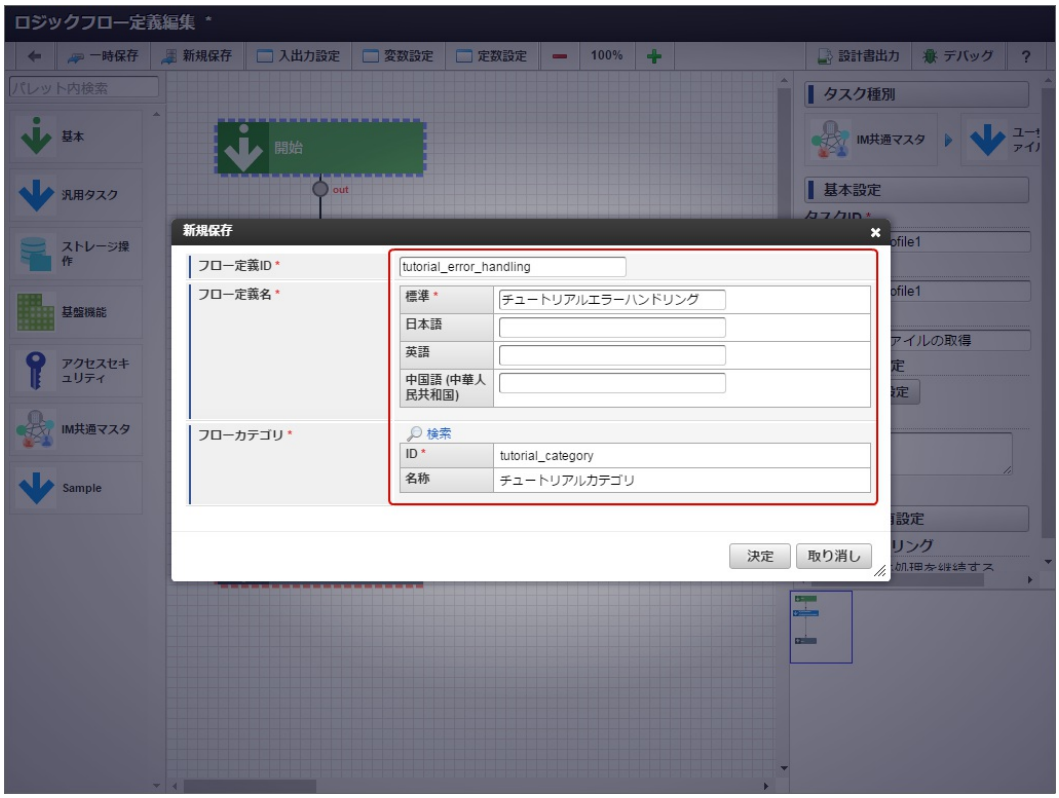
1. 「サイトマップ」→「LogicDesigner」→「フロー定義一覧」から、ロジックフロー定義一覧を開きます。
   ロジックフロー定義一覧画面左上の「ロジックフロー新規作成」をクリックします。

2. 「ユーザプロファイルの取得」タスク（IM共通マスタ）をフロー編集画面に追加し、以下のエレメント間に線を引きます。
   - 「開始」制御要素から「ユーザプロファイルの取得」タスク
   - 「ユーザプロファイルの取得」タスクから「終了」制御要素




図：ロジックフロー配線例

3. ロジックフローを以下のとおりに保存します。
   - フロー定義ID「tutorial_error_handling」
   - フロー定義名
     - 標準 - 「チュートリアルエラーハンドリング」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし
   - フローカテゴリ
     - ID - 「tutorial_category」
     - 名称 - 「チュートリアルカテゴリ」




図：ロジックフローの保存

以上で、フローの準備が完了しました。

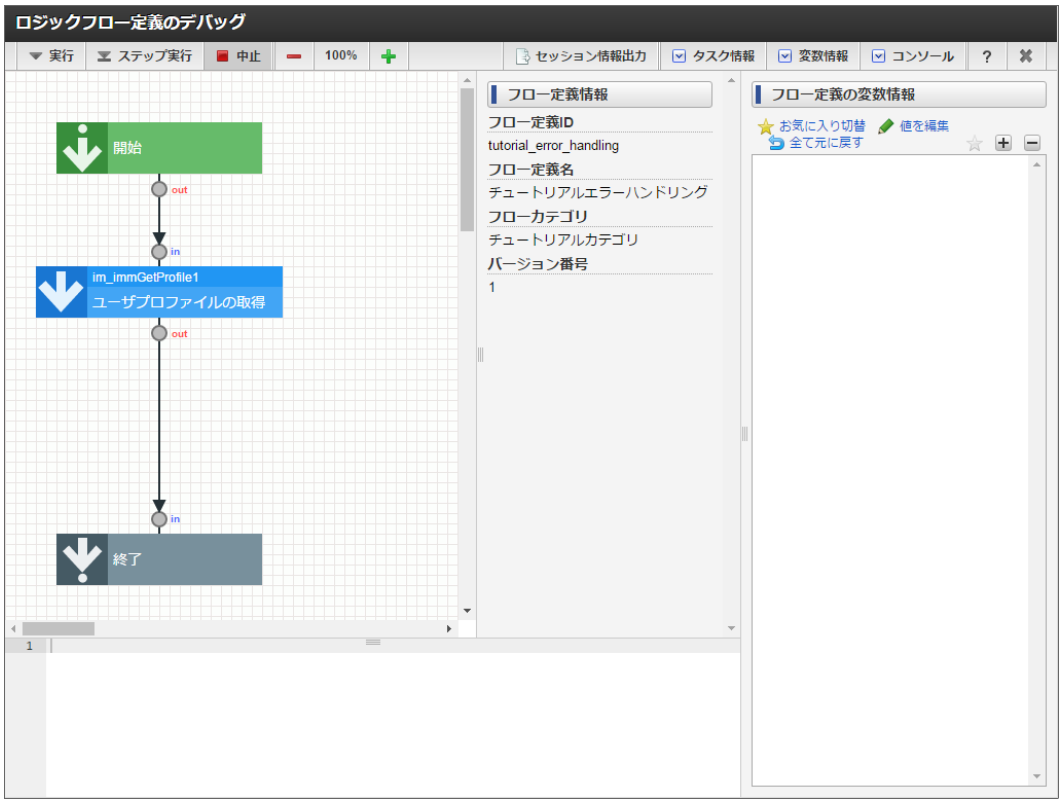
> **コラム**
>
> 今回準備したフローについて
>
> 今回準備したフローでは、「ユーザプロファイルの取得」タスクを実行する上で必須パラメータであるユーザコードを指定していません。
> そのため、このフローを実行した場合必ずエラーが発生します。

### デフォルトの動作を確認する

次に、デフォルトでのエラー発生時のハンドリングを確認します。
IM-LogicDesignerではエレメントの処理中にエラーが発生した場合、デフォルトの動作として「フロー全体のエラーとして処理を中断」します。

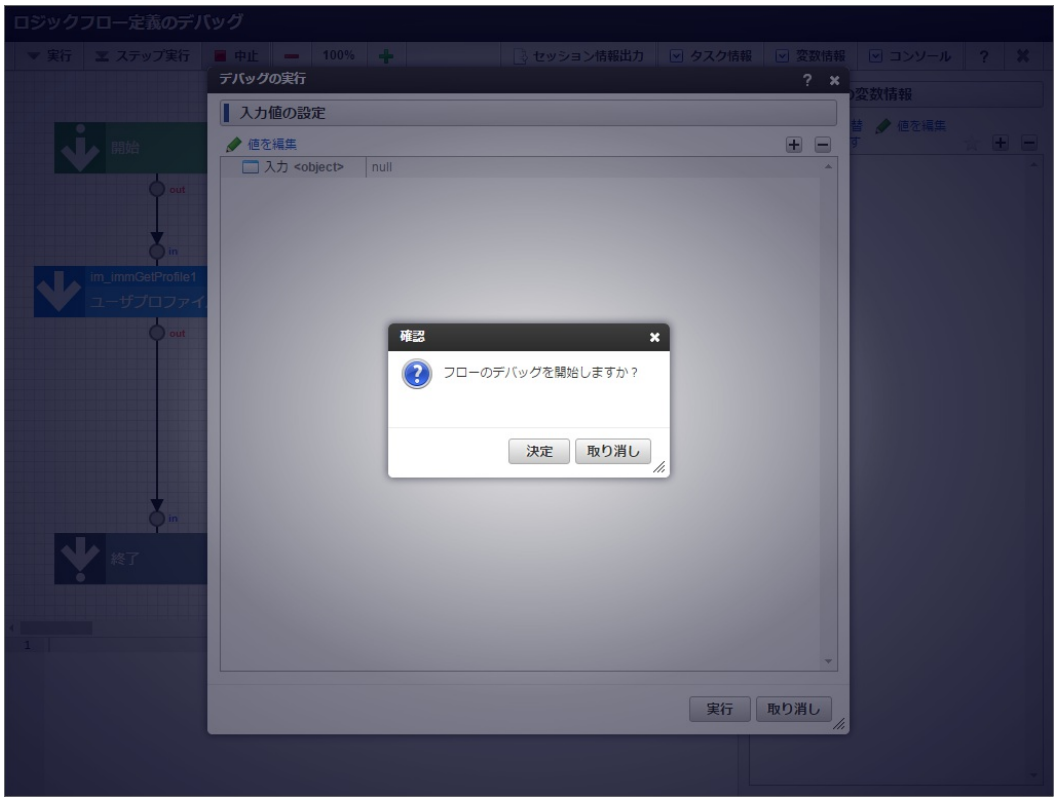
今回の動作確認は発生したエラーも含めて確認していくため、デバッグ機能を利用します。
デバッグ機能の詳細な利用方法については「*ロジックフローのデバッグ*」を参照してください。

1. ロジックフロー定義編集画面上部、ヘッダ内の「デバッグ」をクリックし、デバッグ画面を表示します。



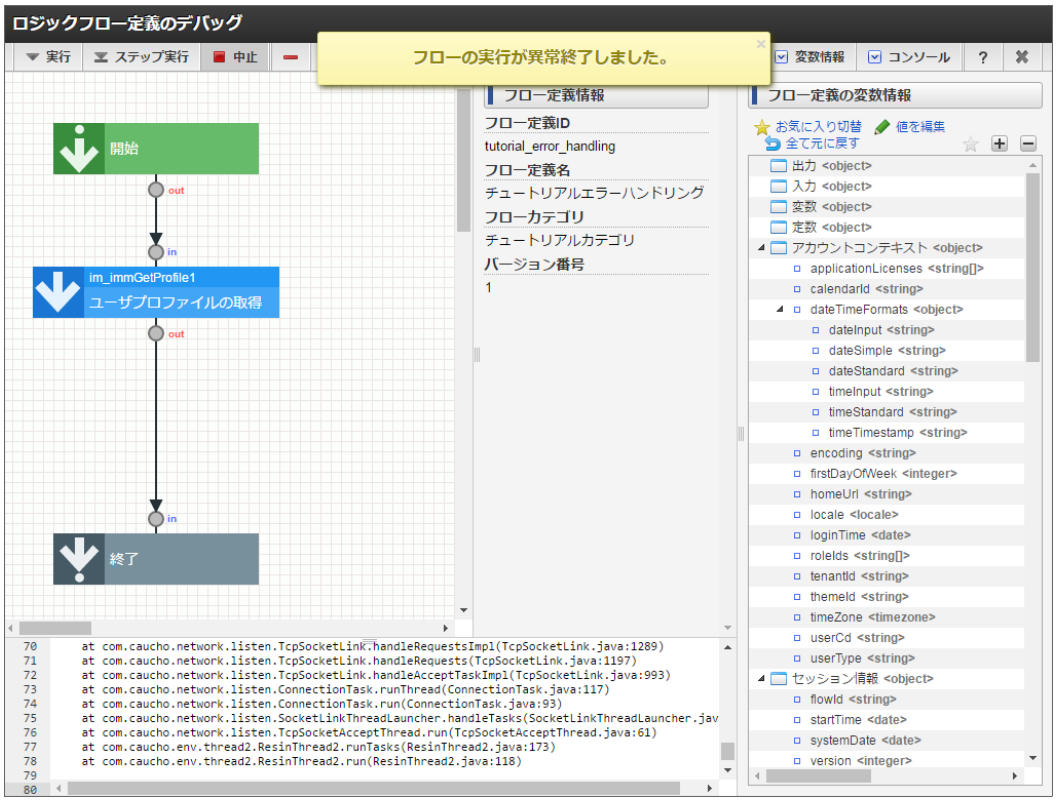

図：デバッグ画面

2. デバッグ画面上部、ヘッダ内の「実行」をクリックし、デバッグ実行を行います。




図：デバッグ実行

3. 「ユーザプロファイルの取得」タスク実行時にエラーが発生し、ロジックフローが異常終了することを確認してください。



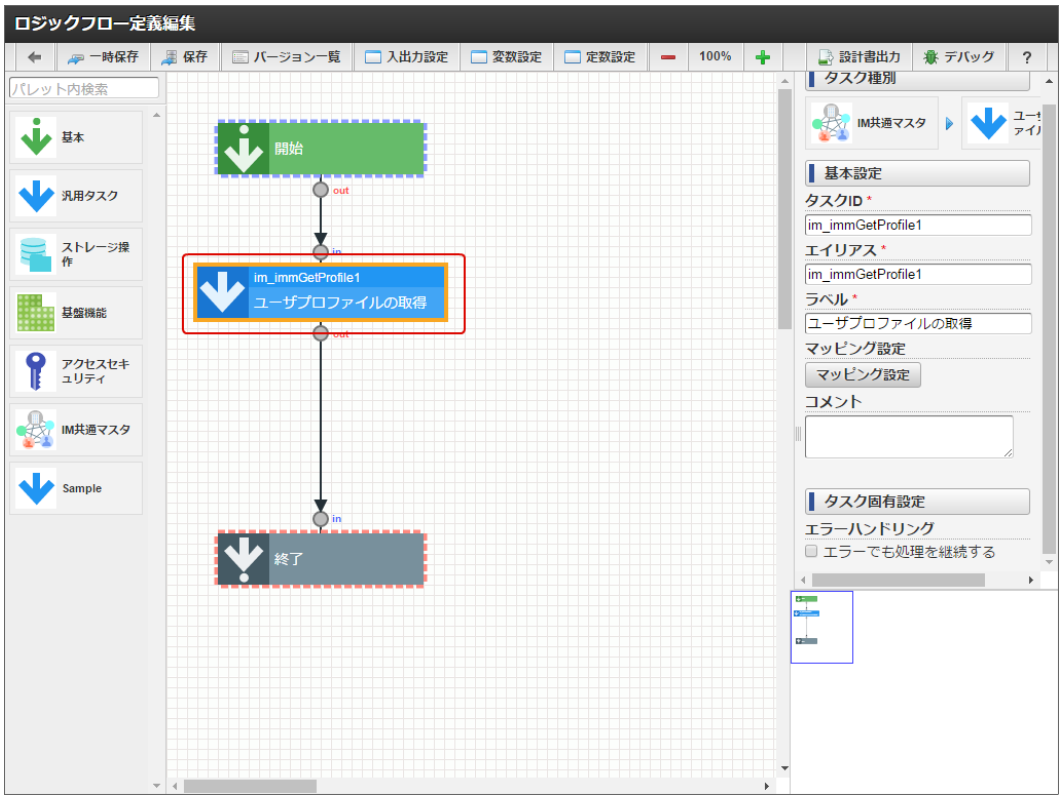

図：デバッグ実行結果（異常終了）

以上で、エラー発生時のデフォルトのハンドリングが確認できました。

## エラーハンドリングの設定を行う

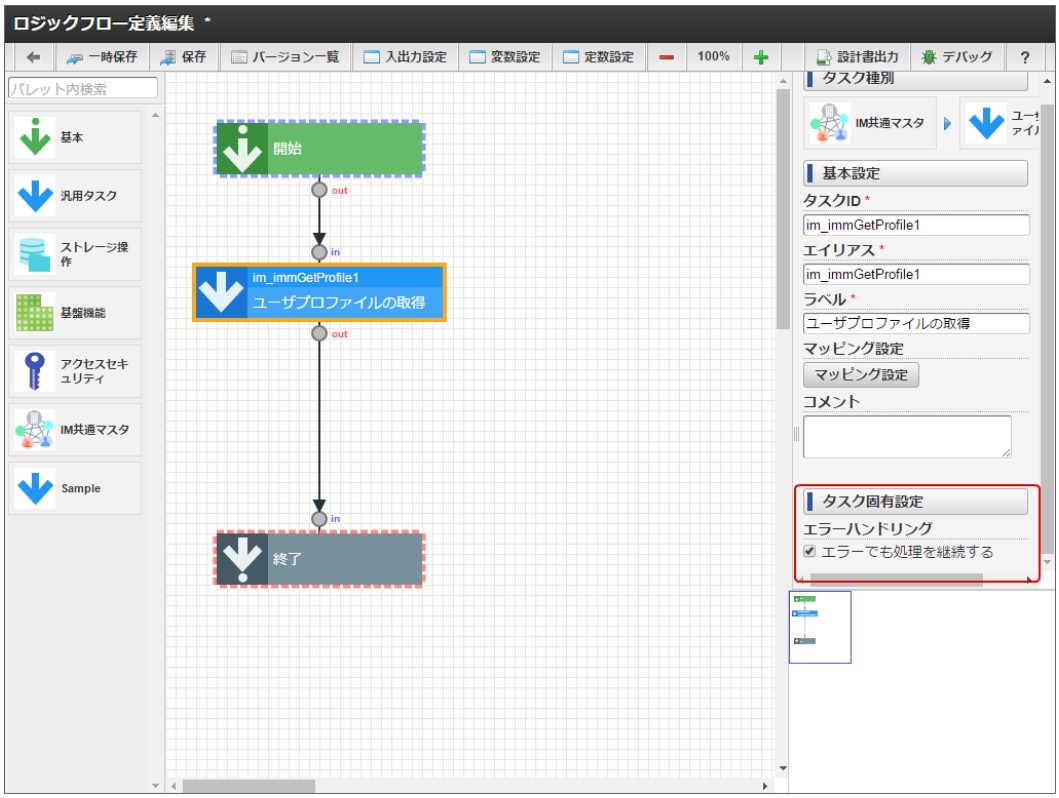
次に、エラーハンドリングの設定を行います。
エラーハンドリングの設定は、エレメントのプロパティから行います。

1. デバッグ画面を開いている場合は、デバッグ画面を閉じます。
2. 「ユーザプロファイルの取得」タスクをクリックし、プロパティ画面を表示します。




図：「ユーザプロファイルの取得」タスクのプロパティ画面

3. タスク固有設定の項目を以下のとおりに変更します。
   - エラーでも処理を継続する - チェックボックス：オン




図：エラーハンドリングの設定

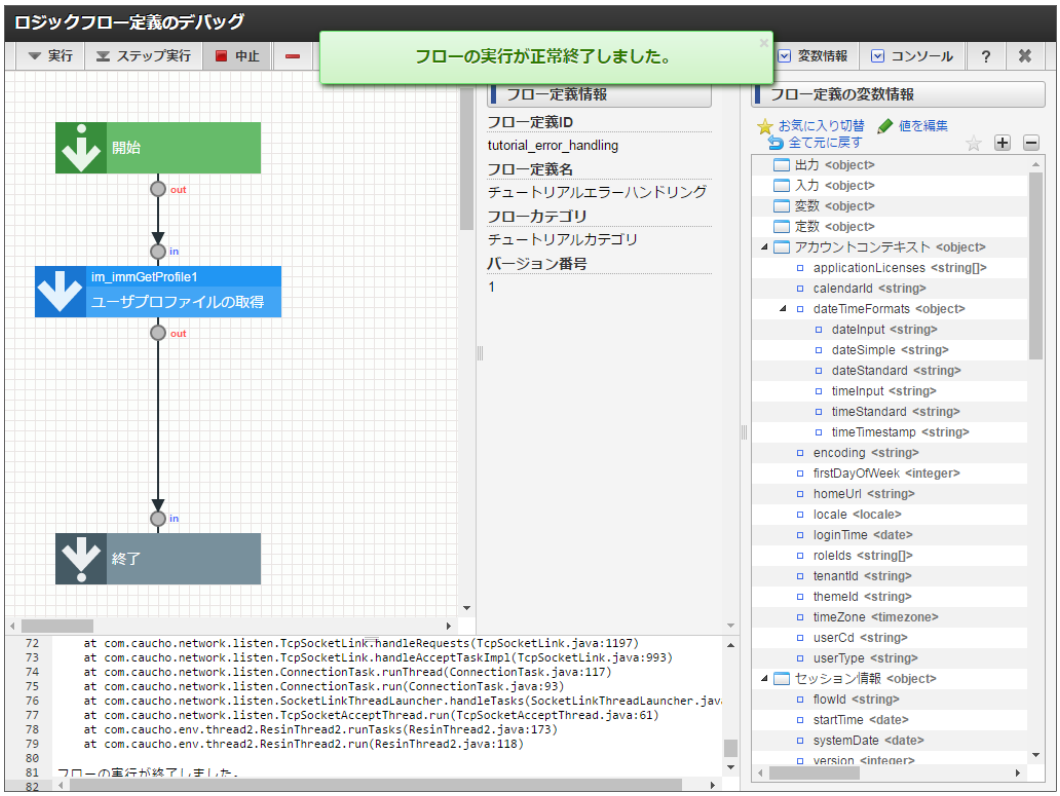
4. ロジックフローを保存します。

以上で、エラーハンドリングの設定が完了しました。
この設定により、「ユーザプロファイルの取得」タスクで処理中にエラーが発生した場合にも処理が継続されるようになりました。

## エラーハンドリング設定後の動作を確認する。

最後に、設定したエラーハンドリングの動作を確認します。
確認は「*デフォルトの動作を確認する*」と同じくデバッグ機能から行います。

1. ロジックフロー定義編集画面上部、ヘッダ内の「デバッグ」をクリックし、デバッグ画面を表示します。
2. デバッグ画面上部、ヘッダ内の「実行」をクリックし、デバッグ実行を行います。
3. 「ユーザプロファイルの取得」タスク実行時にエラーが発生しますが、ロジックフローが「正常に終了」したことを確認してください。




図：デバッグ実行結果（正常終了）

これは、「ユーザプロファイルの取得」タスクでエラーが発生した場合でも、次のエレメント（「終了」制御要素）へ処理が渡ったことを表しています。

以上で、エラーハンドリングの動作が確認できました。

> **注意**
>
> エラー発生後の動作について
>
> IM-LogicDesignerのエラー発生時のハンドリング（エラーでも処理を継続するプロパティ）は、あくまで「エラーが発生した場合でも次に処理を進めるハンドリング」であることに注意してください。
> エラーの発生したエレメントから本来得られるはず情報（今回であればユーザプロファイル）を以降のタスクで利用していた場合、**以降のタスクでも連鎖的にエラーが発生する可能性が考えられます。**
>
> そのため本章で説明したエラーハンドリングは、エレメントにおいてエラーが発生しても後続の処理に影響を与えない場合のみ設定するか、次章の「*処理結果情報の利用方法、および、動作確認*」を参照し、適切な処理の振り分けを行うフローを定義した上で利用してください。

## 処理結果情報の利用方法、および、動作確認

ここでは、エラー発生後のハンドリングを行うために、処理結果情報の詳細と利用方法を説明します。

具体的にはエレメントの処理結果情報にはどの様な内容が含まれているかを説明します。
そして、処理結果情報を利用した異なる振る舞いを行うフローを作成、動作確認します。

### 処理結果情報の詳細

はじめに、エレメントの処理結果情報の詳細について説明します。

エレメントの処理結果情報は**暗黙的な変数**の一つとして利用可能です。

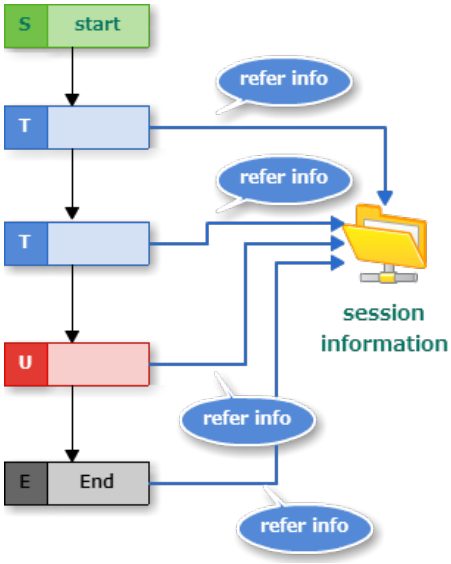
> **コラム**
>
> 暗黙的な変数について
>
> 暗黙的な変数についての詳細、および、利用方法は「*暗黙的な変数の利用*」を参照してください。

処理結果情報は以下の要素から構成されます。

| プロパティ名 | 詳細 |
| --- | --- |
| error<boolean> | エレメントの処理がエラーだった場合trueが指定されます。<br>エレメントの処理が正常に終了した場合はfalseが指定されます。 |
| errorMessage<string> | エラー発生時のエラーメッセージが指定されます。<br>エレメントの処理が正常に終了した場合はnullが指定されます。 |
| executeId<string> | この処理結果情報が紐づくエレメントの実行IDが指定されます。 |

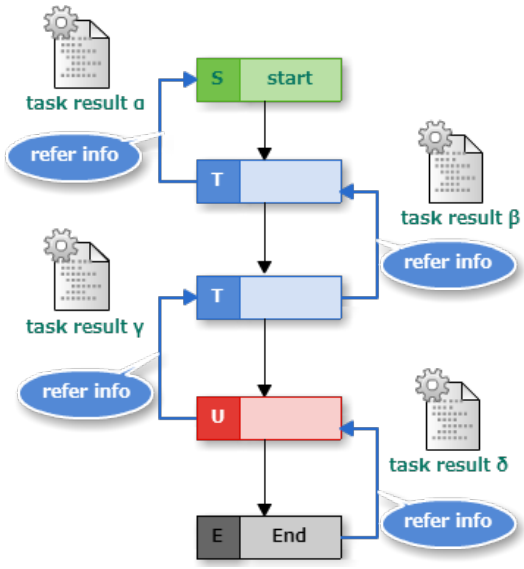
また、処理結果情報には利用するにあたって考慮すべき仕様があります。
それは、処理結果情報は「参照するエレメントの一つ前のエレメントの情報」を返すという点です。

例えば、同じ暗黙的な変数である「セッション情報」は、どのエレメントから参照しても同じ振る舞いの値を返します。




図：セッション情報は全エレメントで同じ振る舞いの値を参照する。

しかし、処理結果情報は参照するエレメントによって取得できる値の振る舞いが変わります。




図：処理結果情報は各エレメントの一つ前のエレメント情報を、それぞれ参照する。

ロジックフローを作成する上で、離れたエレメントの処理結果を参照したい場合は、「*変数を利用したフロー*」の利用を検討してください。

## エラー発生後のハンドリングを行うフローの作成

次に、処理結果情報を用いてエラー発生後のハンドリングを行うフローを作成します。
作成するフローは「*フローの準備*」をベースとして、以下のとおりに編集するものとします。

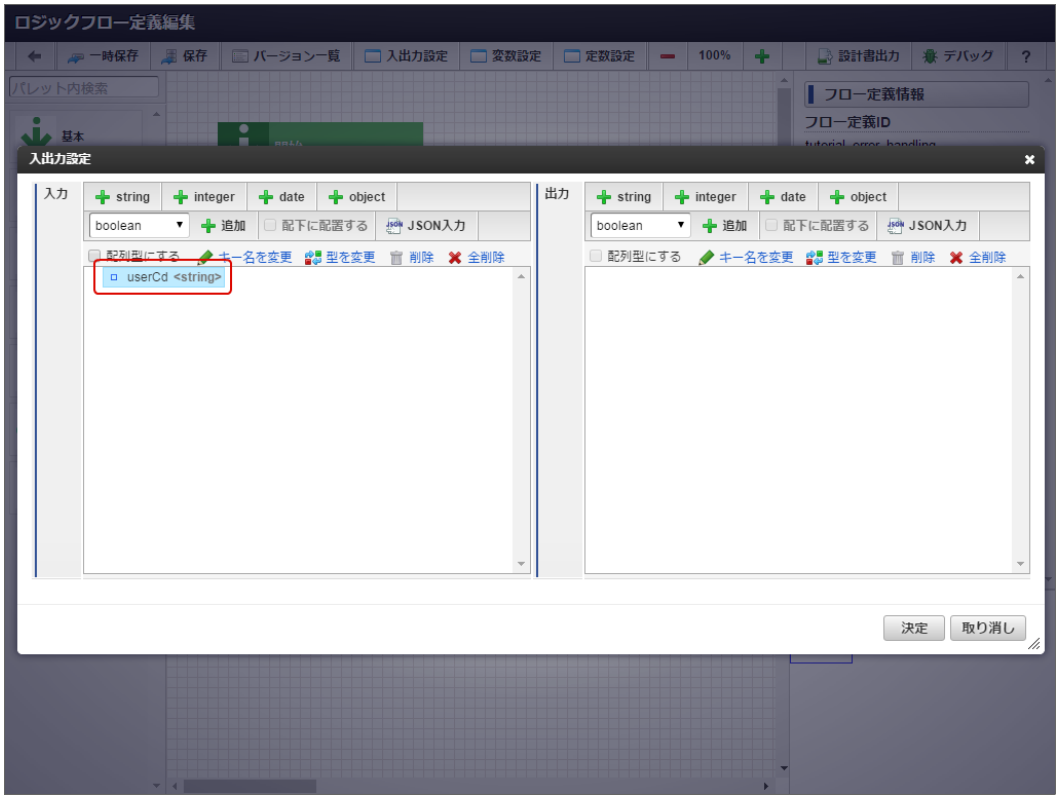
- 新しくロジックフローの入力として、ユーザコードを受け取る。
- 「ユーザプロファイルの取得」タスクは、入力のユーザコードを利用してユーザプロファイルを取得する。
- 「ユーザプロファイルの取得」タスクの処理結果に応じて以下のように振る舞うフローを定義する。
  - 正常に処理出来た場合は、取得結果からユーザ名を「ログ出力」タスクを利用して出力する。

- 処理がエラーとなった場合は、処理がエラーになった旨を「ログ出力」タスクを利用して出力する。

「*フローの準備*」で作成したフローのデバッグ画面を閉じ、編集画面を再表示した後に新しくフローを編集していきます。

1. ロジックフロー定義編集画面上部、ヘッダ内の「入出力設定」をクリックし、入出力設定画面を開きます。
2. 入力に以下のパラメータを定義します。

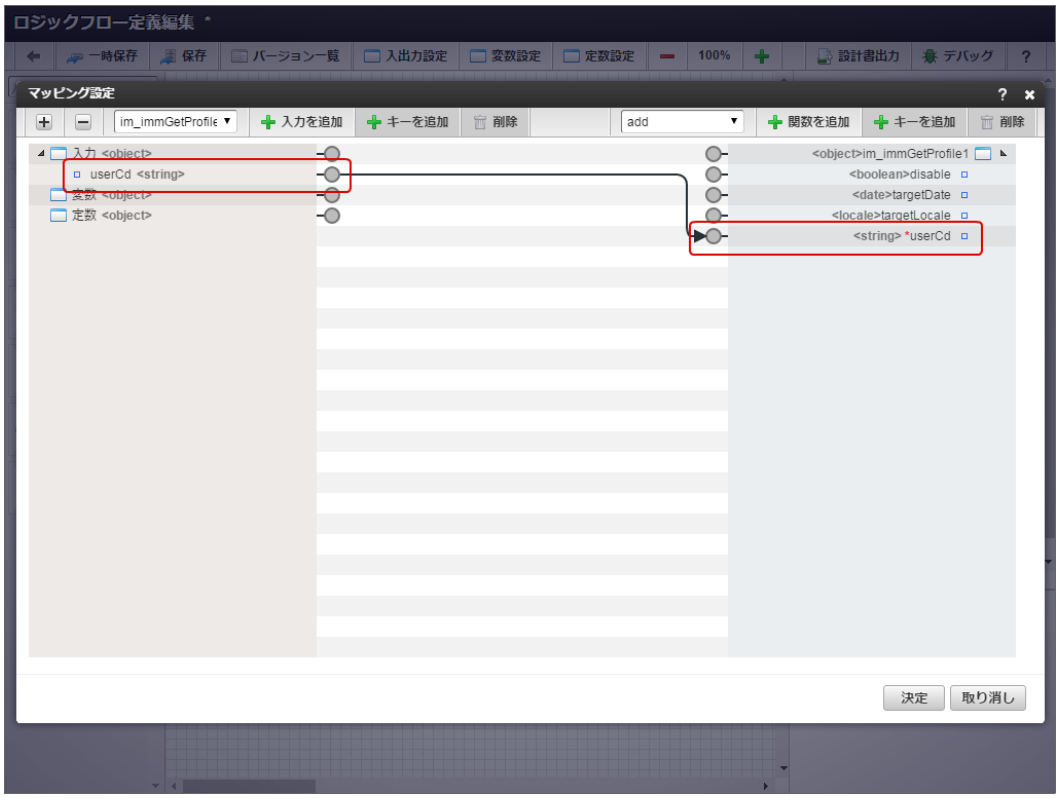
| パラメータ名 | 型 |
| --- | --- |
| userCd | string |




図：新しい入力である「ユーザコード」の定義

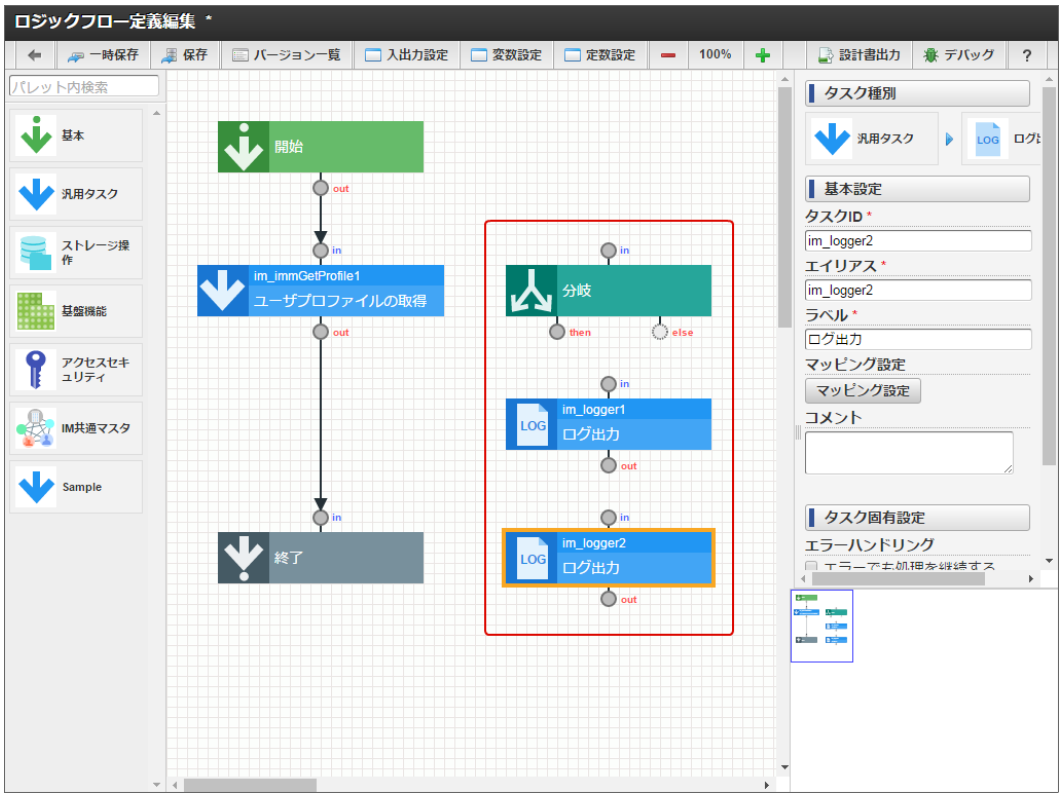
3. 「ユーザプロファイルの取得」タスクをクリックした上で、「マッピング設定」をクリックし、マッピング設定画面を開きます。
4. マッピング設定を以下のとおりに設定します。

| 入力（始点） | 出力（終点） |
| --- | --- |
| 入力<object> - userCd<string> | <object>im_immGetProfile1 - <string>userCd |




図：「ユーザプロファイルの取得」タスクのマッピング設定

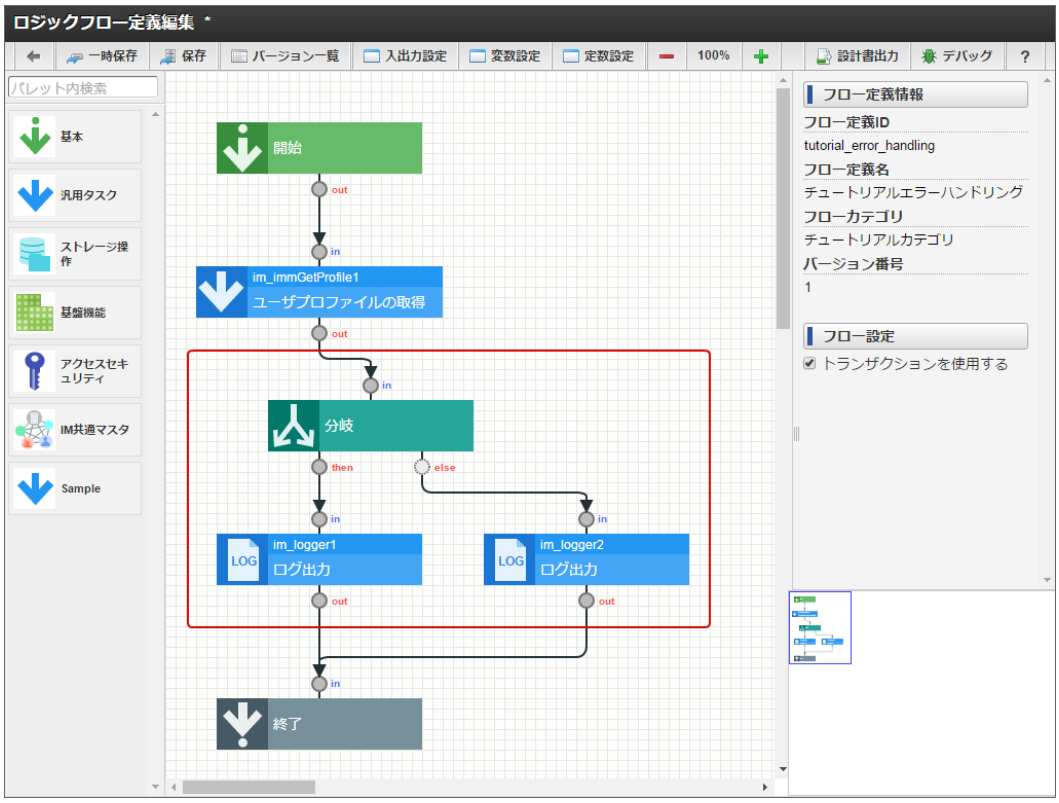
5. 以下のエレメントをフロー編集画面上に追加します。
   - 「基本」-「分岐」制御要素
   - 「汎用タスク」-「ログ出力」タスク（エラー発生時用）
     - タスクID、および、エイリアスは「im_logger1」とします。
   - 「汎用タスク」-「ログ出力」タスク（正常処理用）
     - タスクID、および、エイリアスは「im_logger2」とします。




図：新しいエレメントの追加

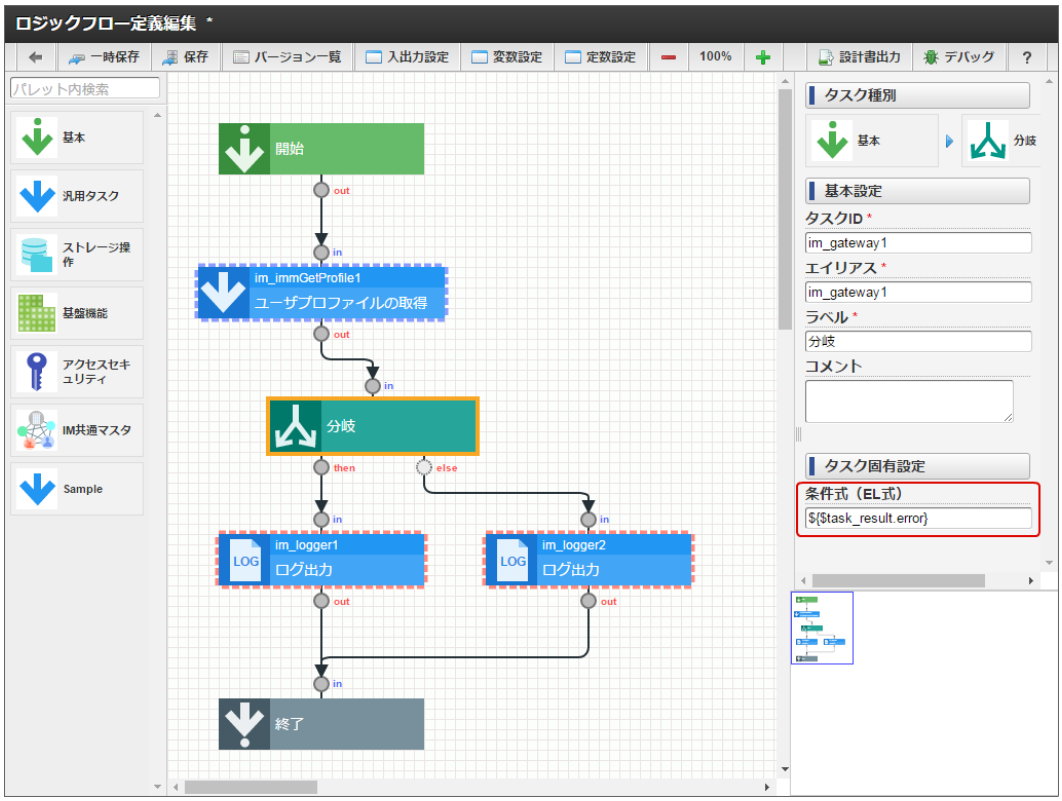
6. 追加したエレメント、および、既存のフローについて 以下のエレメント間に線を引きます。
   - 「ユーザプロファイルの取得」タスクから「分岐」制御要素

- 「分岐」制御要素（then）から「ログ出力」タスク（エラー発生時用）
- 「分岐」制御要素（else）から「ログ出力」タスク（正常処理用）
- 「ログ出力」タスク（正常処理/エラー発生時）から「終了」制御要素




図：ロジックフロー配線例

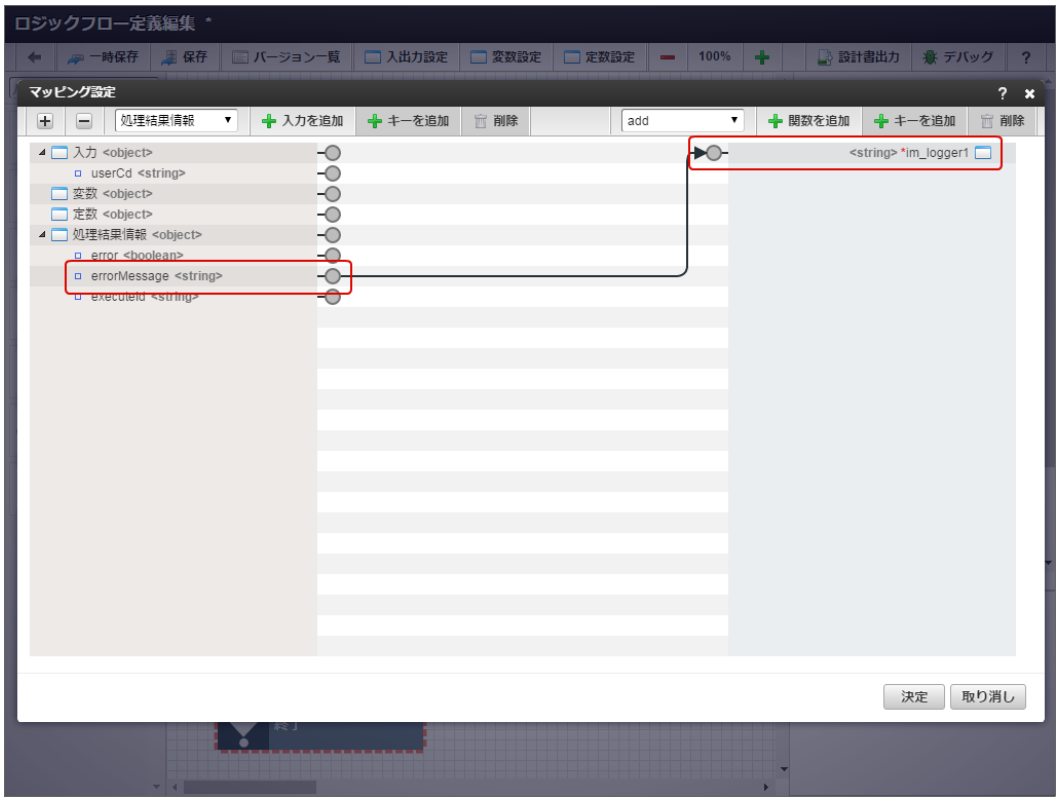
7. 「分岐」制御要素をクリックし、プロパティのタスク固有設定の項目を以下のとおりに変更します。
   - 条件式（EL式）-「${$task_result.error}」




図：条件の定義

8. 「ログ出力」タスク（エラー発生時用）をクリックした上で、「マッピング設定」をクリックし、マッピング設定画面を開きます。
9. マッピング設定を以下のとおりに設定します。

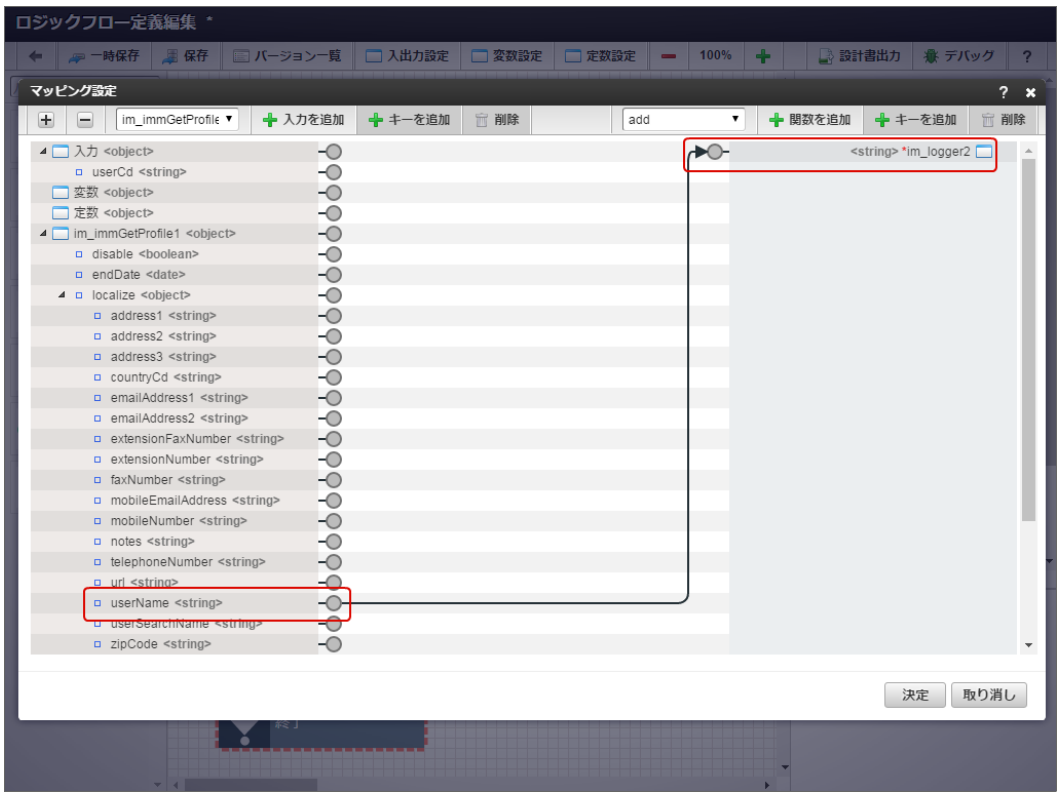
| 入力（始点） | 出力（終点） |
| --- | --- |
| 処理結果情報<object> - errorMessage<string> | <string>im_logger1 |




図：「ログ出力」タスク（エラー発生時用）のマッピング設定

10. 「ログ出力」タスク（正常処理用）をクリックした上で、「マッピング設定」をクリックし、マッピング設定画面を開きます。
11. マッピング設定を以下のとおりに設定します。

| 入力（始点） | 出力（終点） |
| --- | --- |
| im_immGetProfile1<object> - localize<object> - userName<string> | <string>im_logger2 |



図：「ログ出力」タスク（正常処理用）のマッピング設定

12. ロジックフローを保存します。

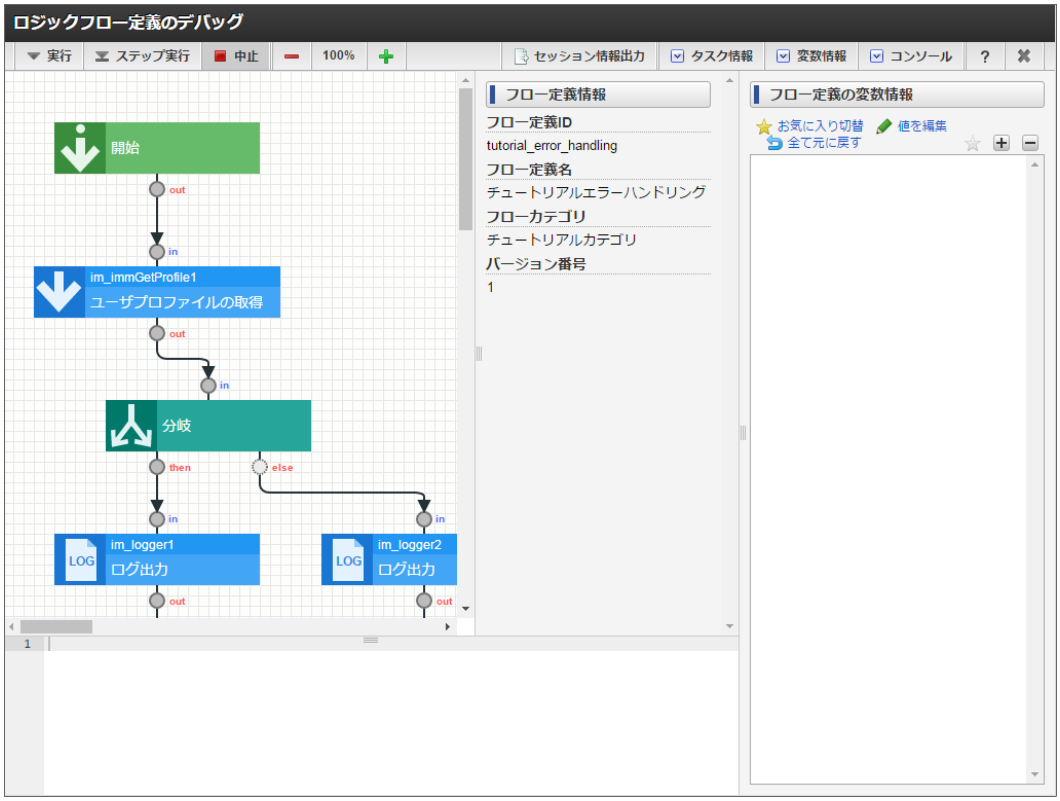
以上で、フローの編集が完了しました。

## エラー発生後のハンドリングを行うフローの動作確認

最後に、編集したロジックフローしたエラーハンドリングの動作を確認します。
確認は「*エラーハンドリングの設定、および、動作確認*」と同じくデバッグ機能から行います。

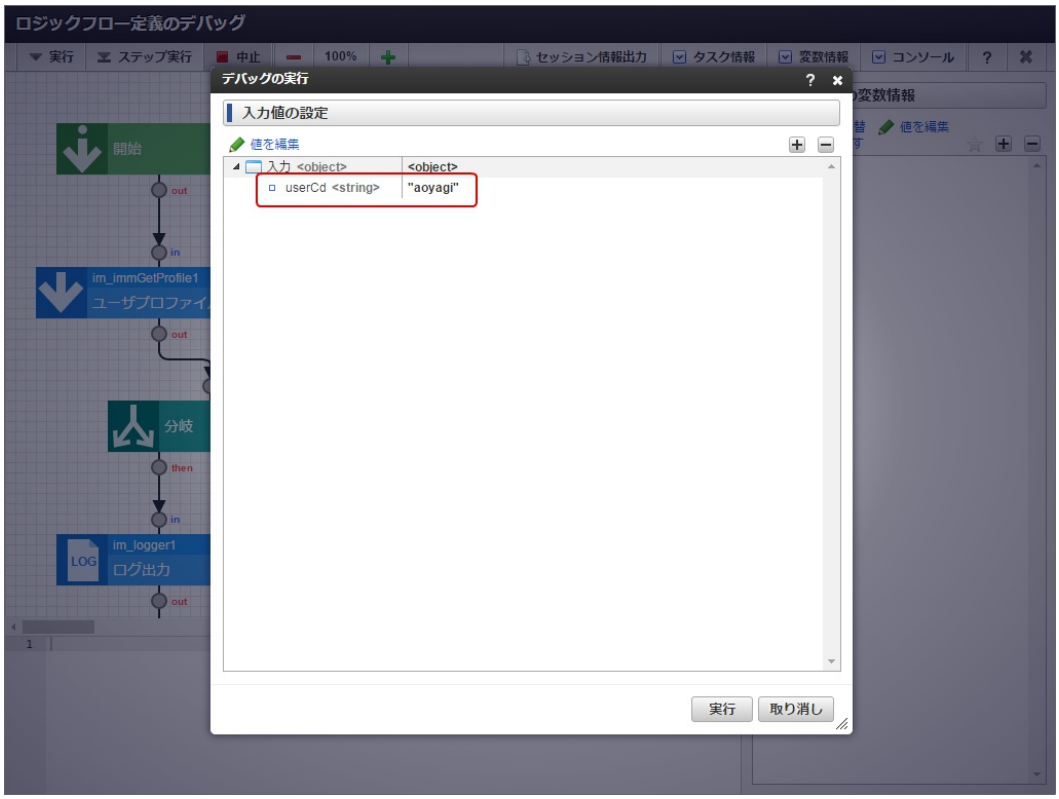
先に正常に処理できた場合の動作を確認します。

1. ロジックフロー定義編集画面上部、ヘッダ内の「デバッグ」をクリックし、デバッグ画面を表示します。




図：デバッグ画面

2. デバッグ画面上部、ヘッダ内の「実行」をクリックします。
3. デバッグ実行の入力値として以下を定義し、実行します。
   - 入力<object> - `userCd<string>` - 「`aoyagi`」

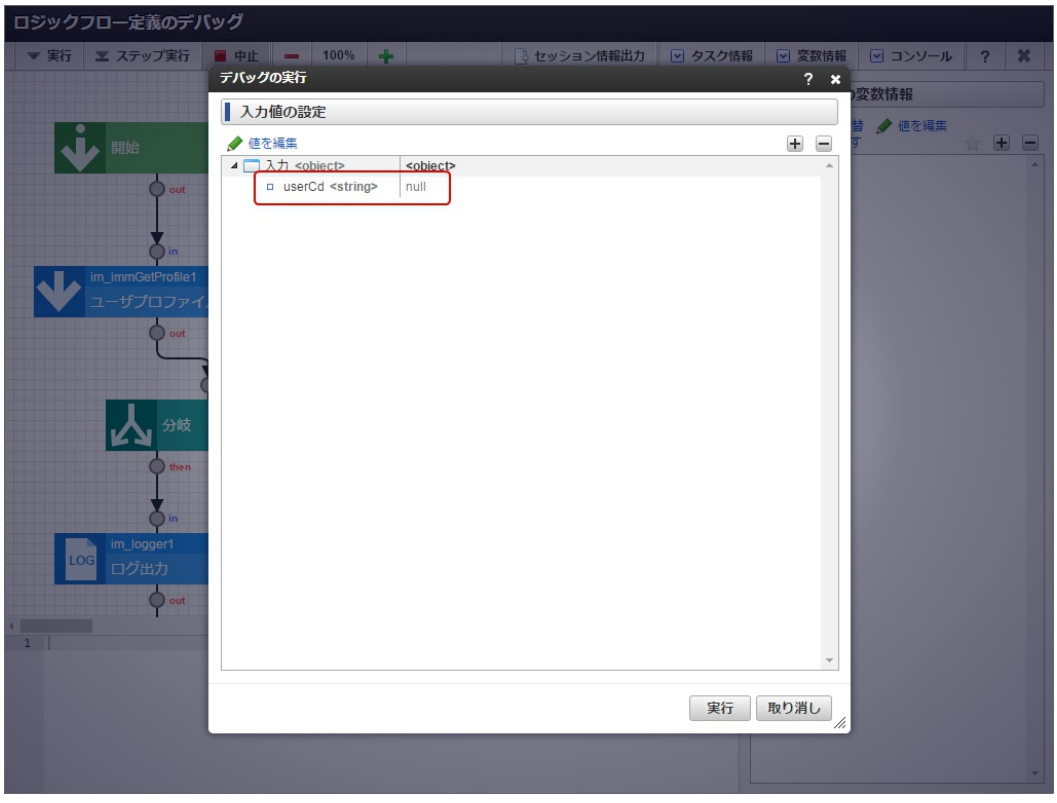



図：デバッグ実行（正常）

4. 実行結果として以下を確認してください。
   - ロジックフローがエラーを発生せずに正常終了していること。
   - 「ログ出力」タスクからサーバのコンソールに「青柳辰巳」が出力されていること。

次に処理の途中でエラーが発生した場合の動作を確認します。

1. デバッグ画面上部、ヘッダ内の「実行」をクリックします。
2. デバッグ実行の入力値として以下を定義し、実行します。
   - 入力<object> - userCd<string> - 「null」




図：デバッグ実行（エラー発生）

3. 実行結果として以下を確認してください。
   - ロジックフローはエラーが発生しているが、正常に終了していること。
   - 「ログ出力」タスクからサーバのコンソールに「[E.IWP.LOGIC.IMMASTER.00003] ユーザコードが指定されていません。」というエラーメッセージが出力されていること。

これは、「ユーザプロファイルの取得」タスクでエラーが発生したか否かの情報（$task_result.error）を利用して分岐処理が行われていることを表しています。

以上で、エラー発生後のハンドリングの動作が確認できました。

## ジョブを利用したロジックフローの実行

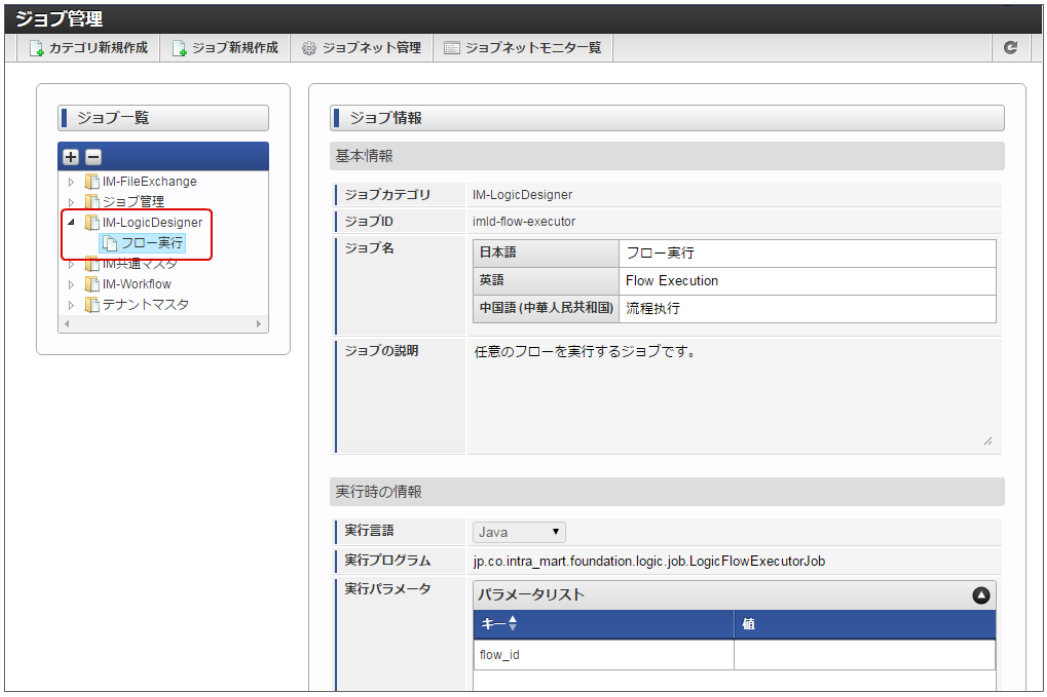
この章では、ロジックフローをジョブから実行する方法について説明します。

> **コラム**
>
> ジョブスケジューラについて
>
> ジョブ、ジョブネットといったジョブスケジューラの基本的な概要や利用方法についての詳細は「ジョブスケジューラ仕様書」を参照してください。



### ロジックフローを実行するジョブの確認

はじめに、ロジックフローを実行するためのジョブを確認します。
IM-LogicDesignerは標準で、任意のロジックフローを実行するジョブを提供しています。
具体的には、ジョブ一覧の「IM-LogicDesigner」 - 「フロー実行」がそれに該当します。




図：ロジックフロー実行ジョブ

フロー実行ジョブの詳細は「ジョブ・ジョブネットリファレンス」 - 「フロー実行」を参照してください。

### ロジックフローを実行するジョブネットの作成

次に、ロジックフローを実行するためのジョブネットを作成します。

#### カテゴリの作成

作成するジョブネットの属するカテゴリを作成します。

今回、以下の内容でジョブネットカテゴリを作成するものとします。

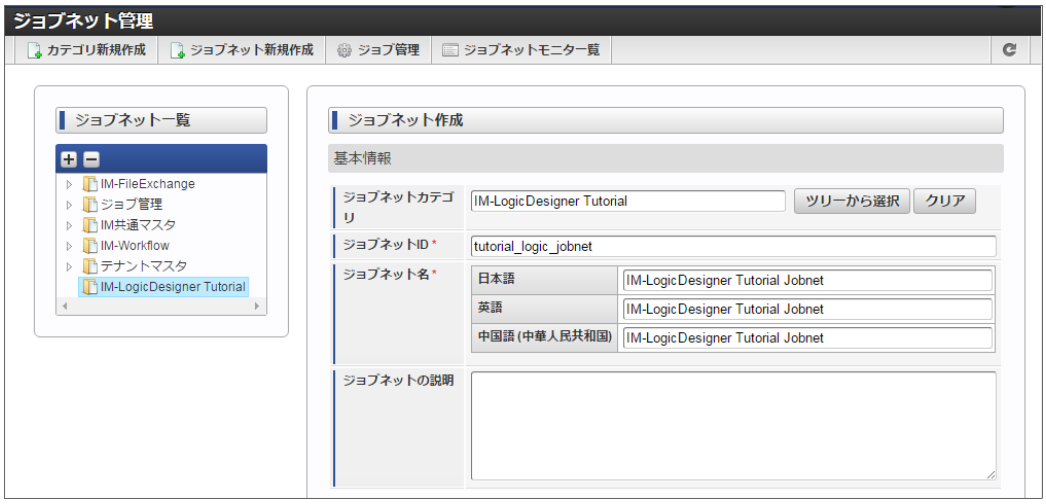
- カテゴリID 「tutorial_logic_category」
- カテゴリ名
  - 日本語、英語、中国語 （中華人民共和国） - 「IM-LogicDesigner Tutorial」


図：作成カテゴリ例

## ジョブネットの作成

ロジックフローを実行するためのジョブネットを作成します。
ジョブネットの基礎情報を以下の内容で設定します。

- ジョブネットカテゴリ 「IM-LogicDesigner Tutorial」
- ジョブネットID 「tutorial_logic_jobnet」
- ジョブネット名
  - 日本語、英語、中国語（中華人民共和国） - 「IM-LogicDesigner Tutorial Jobnet」

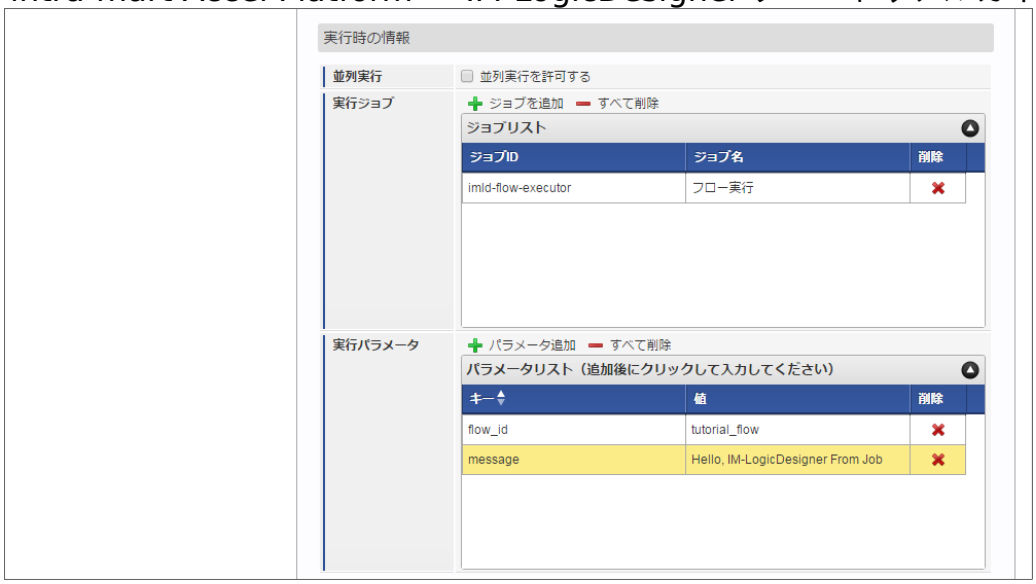



図：基礎情報の定義

次に、ジョブネットの実行時の情報を以下の内容で設定します。

- 並列実行 「チェックボックス：オフ」
- 実行ジョブとして以下のジョブを設定。
  - 「IM-LogicDesigner」 - 「フロー実行」
- 実行パラメータとして以下の内容を設定

| キー | 値 |
| --- | --- |
| flow_id | tutorial_flow |
| message | Hello, IM-LogicDesigner From Job |




図：実行時の情報の定義

> **コラム**
>
> 動作対象とするロジックフローについて
>
> ここで指定するフローID「tutorial_flow」は、「*基礎編 - ファースト・ステップ*」終了後のロジックフローを想定しています。
> 応用編などを元にフロー内容の改変、または、複数バージョンを定義している場合はバージョン追加やインポート等を利用し、フローの調整を行ってください。

次にトリガ設定を以下の内容で設定します。

- トリガー「繰り返し指定」
  - 回数 -「3」
  - 間隔 -「60」
  - 有効 -「チェックボックス：オン」


図：トリガ設定の定義

### ロジックフローの実行の確認

最後に、作成したジョブネットの動作を確認します。

本章で作成したジョブネットは、「ジョブネット作成後、一分間隔で三回」実行されます。
実行結果として「*基礎編 - ファースト・ステップ*」で作成したロジックフローで得られた結果と同じ出力が、三回得られることを確認してください。

以上で、ジョブを利用したロジックフローの実行が完了しました。

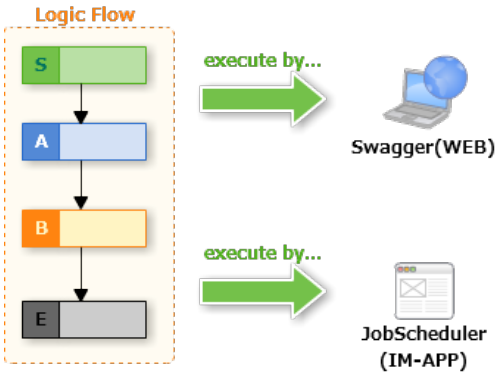
## フロートリガを利用する

この章では、フロートリガの概要、および、利用方法について説明します。

## フロートリガとは？

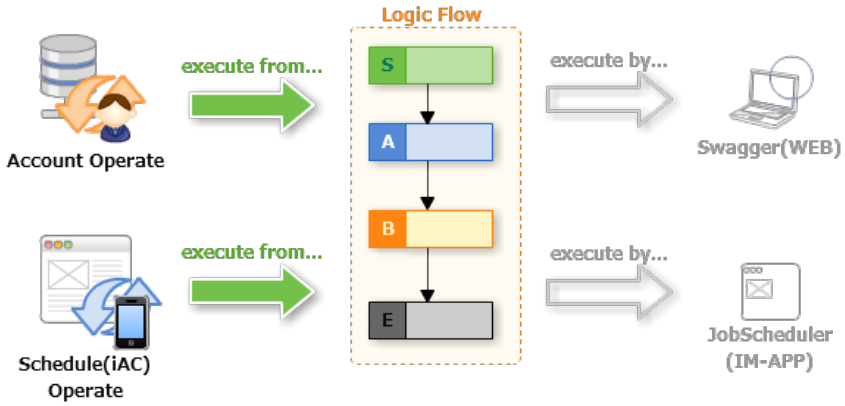
フロートリガとは、intra-mart Accel Platform上で行われる特定の操作に連動して、ロジックフローを実行するための設定情報です。

「*フロールーティングを設定する*」や「*ジョブを利用したロジックフローの実行*」などで説明した方法でロジックフローを実行する場合、開発者が能動的にどのタイミングで実行するかを決定する必要があります。




図：フローの能動的な実行

フロートリガを利用することで、フローを実行したいタイミング（トリガ発生条件）や実行するフローを設定し、受動的にロジックフローを実行することが可能です。




図：フローの受動的な実行

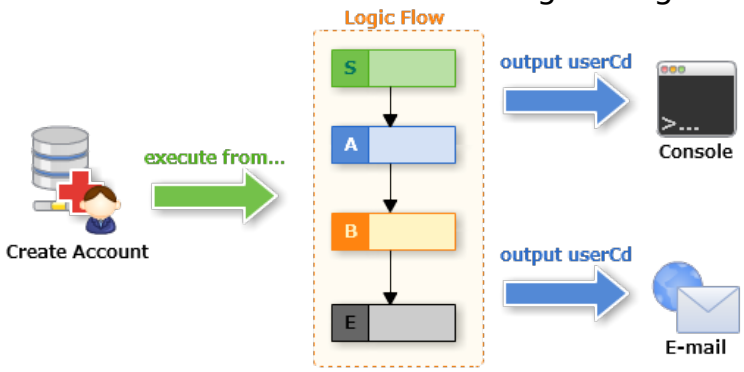
> **コラム**
>
> フロートリガの内部動作
>
> フロートリガはIM-Propagationを利用して実現されています。
> IM-Propagationの詳細については「IM-Propagation 仕様書」を参照してください。

### 本チュートリアルについて

本チュートリアルでは、「*基礎編 - ファースト・ステップ*」で作成したロジックフローを用いて、

**「アカウント情報が新規に作成されたタイミングで、作成されたアカウント情報のユーザコードをロジックフローに渡し、実行する」**

という処理の実装を行います。




図：チュートリアルイメージ

## トリガ定義一覧画面を表示する

フロートリガの一覧画面は、サイトマップより遷移します。

1. 「サイトマップ」→「LogicDesigner」→「トリガ定義一覧」をクリックします。


図：サイトマップ

2. トリガ定義一覧画面が表示されます。


図：トリガ定義一覧画面

## 新規にフロートリガを作成する

フロートリガの作成は、トリガ定義編集画面から行います。

1. トリガ定義一覧画面左上の「新規作成」をクリックします。


図：新規作成リンク

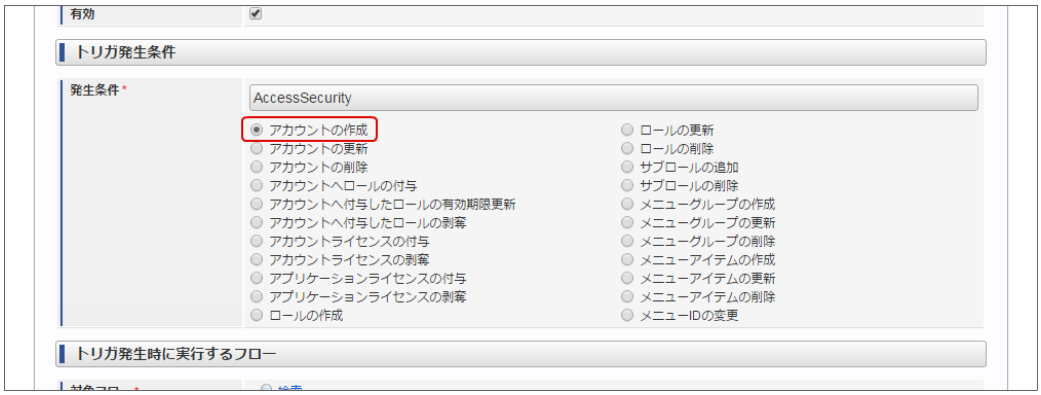
2. トリガ定義編集画面が表示されます。


図：トリガ定義編集画面

3. トリガ定義の基本情報となる各項目に以下の値を入力します。
   - トリガ定義ID「flow_trigger_account_create」
   - トリガ定義名
     - 標準 -「トリガ定義[Account Create]」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし
   - 有効「チェックボックス：オン」

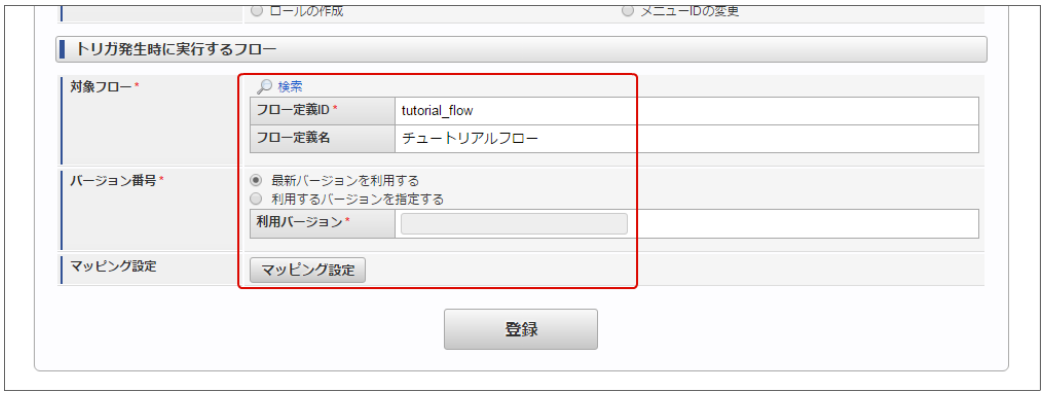

図：基本情報

4. トリガ定義の発生条件について、本チュートリアルが想定するタイミング（トリガ発生条件）である「アカウントの作成」を選択します。




図：トリガ発生条件

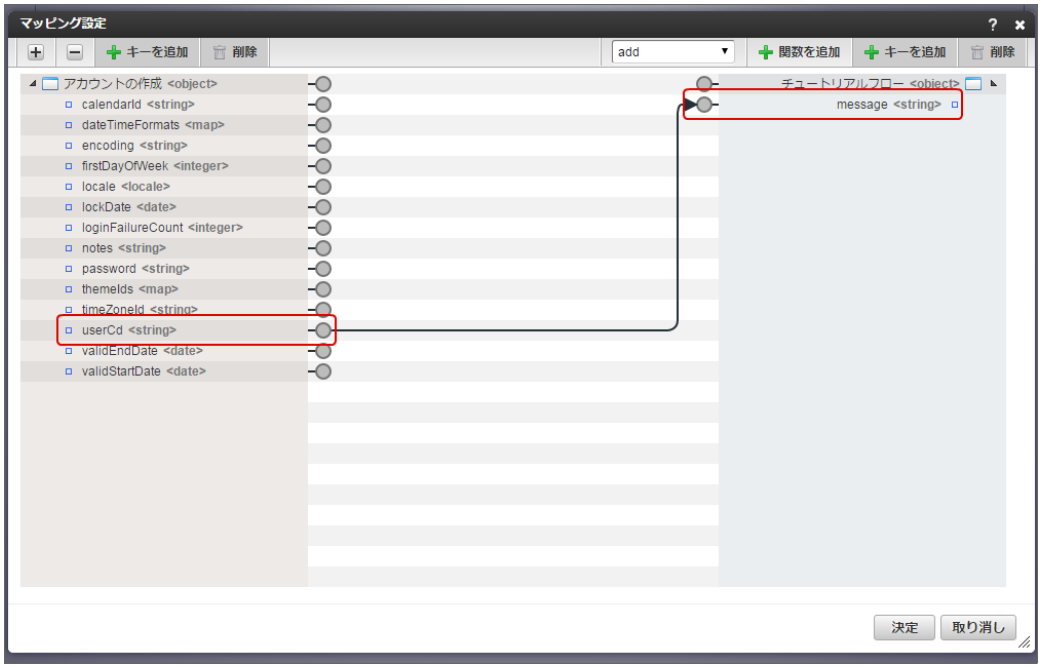
5. トリガ発生時に実行するフローについて、各項目に以下の値を入力します。
   - 対象フロー
     - フロー定義ID - 「tutorial_flow」
     - フロー定義名 - 「チュートリアルフロー」




図：実行フロー情報

6. 作成されたアカウント情報から、ユーザコードをロジックフローへ渡すためのマッピング設定を以下のように行います。

| 入力（始点） | 出力（終点） |
|---|---|
| アカウントの作成<object> - userCd<string> | チュートリアルフロー<object> - message<string> |




図：マッピング設定

7. 「登録」をクリックします。


図：登録

以上で、アカウントの作成をタイミング（トリガ発生条件）としたフロートリガの作成が完了しました。

## 作成したフロートリガの動作を確認する

「*フロートリガとは？*」で説明したとおり、フロートリガは外部の動作をタイミング（トリガ発生条件）に動作します。
本チュートリアルでは「アカウントの作成」をタイミング（トリガ発生条件）としたので、実際にアカウントの作成を行いフローが想定する動作をするか確認します。

1. 「サイトマップ」→「共通マスタ」→「ユーザ」をクリックします。
2. ユーザ検索が表示されます。
   ユーザ検索左上の「ユーザの新規登録」をクリックします。


図：ユーザの新規登録 その 1

3. ユーザ詳細（新規）について、各項目に以下の値を入力します。
   - 基本 - ユーザコード「tutorial_trigger_user」
   - プロファイル - ユーザ名「チュートリアルトリガユーザ」


図：ユーザの新規登録 その２

4. 「登録」をクリックします。

アカウントの作成後、続いてロジックフローの実行結果を確認します。
実行結果として「*基礎編 - ファースト・ステップ*」で作成したロジックフローの以下の内容が、作成したユーザコード（tutorial_trigger_user）を入力として出力されていることを確認してください。

- 「*「ログ出力」タスクの結果*」で確認したログ
- 「*「テキストメール送信」タスクの結果*」で確認したメール

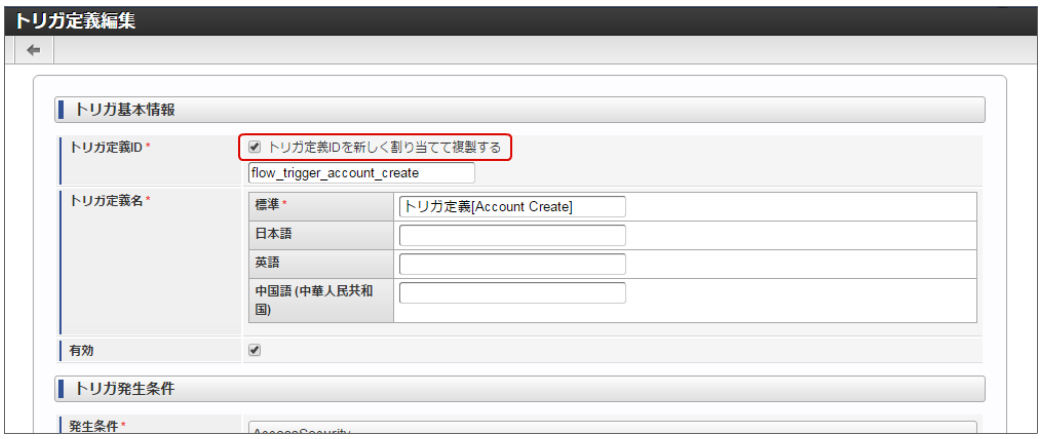
以上で、作成したフロートリガの動作が確認できました。

## 同じトリガ発生条件に複数のロジックフローを設定する

フロートリガでは、同じトリガ発生条件に複数のロジックフローを設定することができます。
ここでは「*新規にフロートリガを作成する*」で説明した方法をベースに、二つ目のフロートリガを設定します。

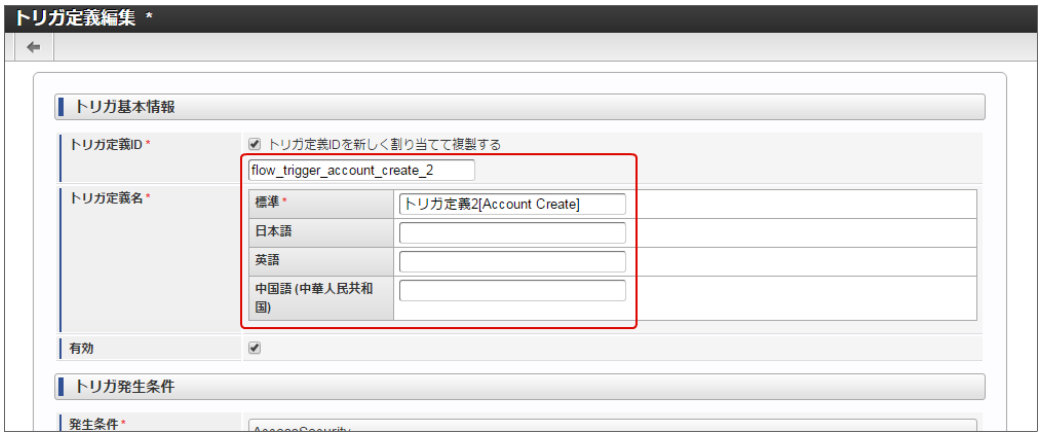
1. 「サイトマップ」→「LogicDesigner」→「トリガ定義一覧」から、トリガ定義一覧画面を開きます。
   一覧の中から、トリガ定義ID「flow_trigger_account_create」の行の編集アイコンをクリックします。

2. トリガ定義の基本情報の「トリガ定義IDを新しく割り当てて複製する」にチェックを入れます。




図：トリガ定義IDを新しく割り当てて複製する

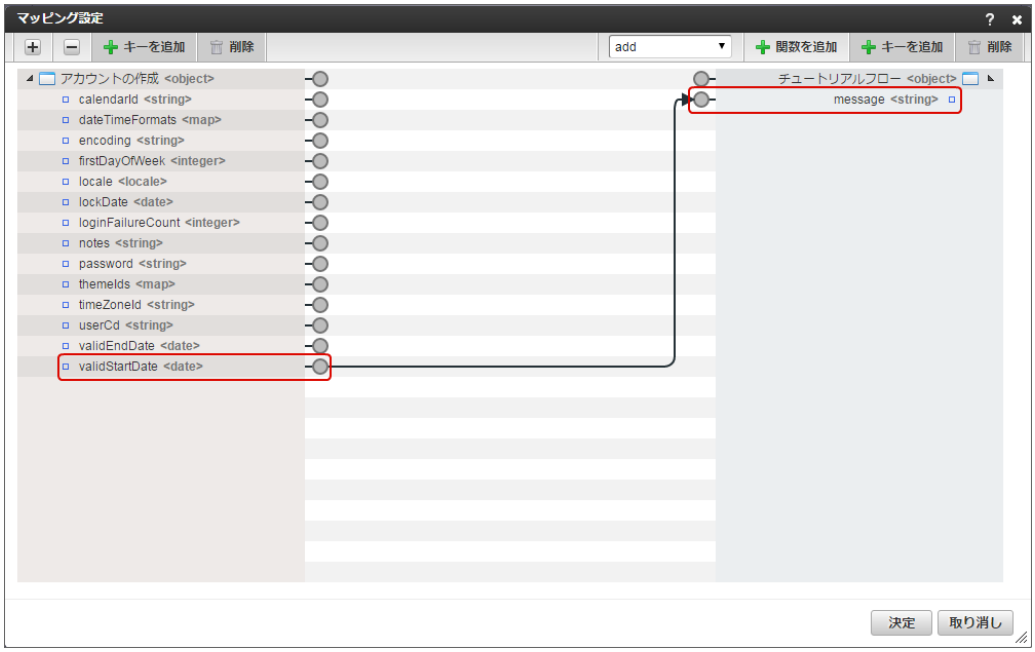
3. トリガ定義IDが入力可能になったことを確認した上で、トリガ定義の基本情報となる各項目について以下の値を変更します。
   - トリガ定義ID「flow_trigger_account_create_2」
   - トリガ定義名
     - 標準 -「トリガ定義2[Account Create]」
     - 日本語、英語、中国語（中華人民共和国） - 入力なし




図：二つ目のフロートリガの設定

4. 今回作成するフロートリガでは、作成したアカウントの有効開始日を入力としてマッピング設定を行います。
   作成されたアカウント情報から、有効開始日をロジックフローへ渡すためのマッピング設定を以下のように行います。
   （もとから定義されているマッピング情報は削除してください）

| 入力（始点） | 出力（終点） |
| --- | --- |
| アカウントの作成<object> - validStartDate<date> | チュートリアルフロー<object> - message<string> |




図：二つ目のフロートリガのマッピング設定

5. 「登録」をクリックします。

二つ目のフロートリガ作成後、「*作成したフロートリガの動作を確認する*」で説明した方法と同様に新しいユーザを作成します。
ユーザ作成後の実行結果として、以下の二つの出力結果が得られることを確認してください。

- 「*新規にフロートリガを作成する*」で作成したフロートリガから得られる結果（ユーザコードの出力）。
- 本項で作成したフロートリガから得られる結果（有効開始日の出力）。

> **注意**
>
> フロートリガの実行順序について
>
> 同じトリガ発生条件を設定したフロートリガが複数あった場合、**フロートリガの実行順序は不定**であることに注意してください。
> あるフロートリガの動作以前、または、以後に動くことを想定する別のフロートリガの設定を行った場合、正常に動作しない場合があります。

## 作成したフロートリガの利用を停止する

フロートリガの利用を停止する場合、以下の二つの方法があります。

- フロートリガを無効にする。
- フロートリガを削除する。

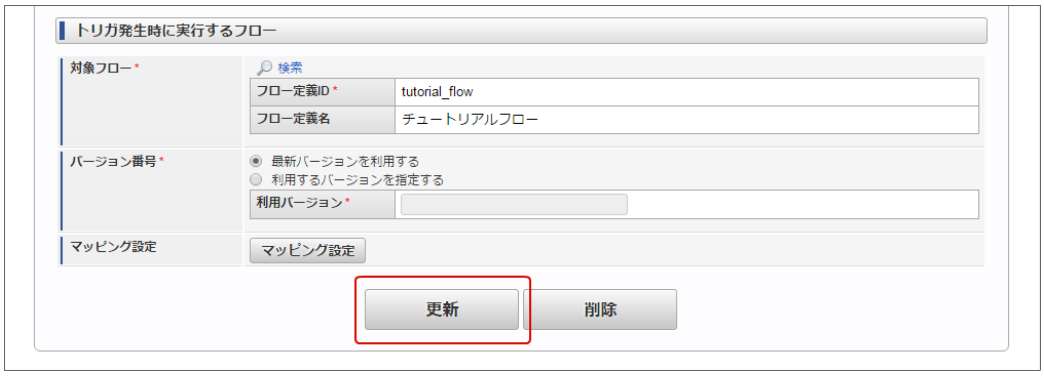
### 有効/無効による制御

一時的にフロートリガの利用を停止したい場合、フロートリガを無効にする方法が有用です。

1. 「サイトマップ」→「LogicDesigner」→「トリガ定義一覧」から、トリガ定義一覧画面を開きます。
   一覧の中から、トリガ定義ID「flow_trigger_account_create」の行の編集アイコンをクリックします。
2. トリガ定義の基本情報から、以下の項目を変更します。
   - 有効「チェックをはずす」


図：有効/無効の設定

3. 「更新」をクリックします。




図：更新

更新後、「*作成したフロートリガの動作を確認する*」で説明した方法と同様に新しいユーザを作成します。
ユーザ作成後の実行結果として、「*新規にフロートリガを作成する*」で作成したフロートリガは動作していないことを確認してください。

### フローを削除する

対象のフロートリガが不要になった等、今後利用するケースが無い場合、フロートリガを削除する方法が有用です。

1. 「サイトマップ」→「LogicDesigner」→「トリガ定義一覧」から、トリガ定義一覧画面を開きます。
   一覧の中から、トリガ定義ID「flow_trigger_account_create_2」の行の編集アイコンをクリックします。
2. トリガ定義編集下部、「削除」をクリックします。


図：削除

3. 削除を確認するダイアログが表示されるので、決定をクリックします。
4. 削除が完了した旨のメッセージと共に、トリガ定義一覧画面に遷移します。


図：削除後、一覧画面へ遷移

削除後、「*作成したフロートリガの動作を確認する*」で説明した方法と同様に新しいユーザを作成します。
ユーザ作成後の実行結果として、「*同じトリガ発生条件に複数のロジックフローを設定する*」で作成したフロートリガは動作していないことを確認してください。

# ログを用いたデバッグ



## ログを用いたデバッグについて

IM-LogicDesignerでは、ロジックフロー実行時の内部的に管理されている実行フェーズと、それに伴う入出力値の詳細を、デバッグログとして出力することが可能です。
以下に、「*基礎編 - ファースト・ステップ*」で作成したロジックフローを例に、出力内容の違いを提示します。

### ログ設定無し

ログ設定が無い場合、出力内容としてフロー内に定義された「ログ出力」タスクの実行内容のみが確認できます。

```
...
[INFO] TutorialLogger - [] Hello, IM-LogicDesignerLog
...
```

### ログ設定有り

ログ設定がある場合、出力内容として実行フェーズや、その際にやりとりされている実際の値が確認できます。

```
...
[DEBUG] LOGIC_FLOW_LOG - [] create session. (flowId=tutorial_flow)
[DEBUG] LOGIC_FLOW_LOG - [] [BEGIN_FLOW] execute session. (flowId=tutorial_flow, inputData={message=Hello, IM-
LogicDesignerLog})
[DEBUG] LOGIC_FLOW_LOG - [] [BEFORE_EXECUTION] execute task. (executeId=im_logger1,
taskId=ApplicationElementKey(elementId=im_logger), inputData=Hello, IM-LogicDesignerLog)
[INFO] TutorialLogger - [] Hello, IM-LogicDesignerLog
[DEBUG] LOGIC_FLOW_LOG - [] [AFTER_EXECUTION] task result. (executeId=im_logger1,
taskId=ApplicationElementKey(elementId=im_logger), result=null)
[DEBUG] LOGIC_FLOW_LOG - [] [BEFORE_EXECUTION] execute task. (executeId=im_sendTextMail1,
taskId=ApplicationElementKey(elementId=im_sendTextMail), inputData=TextMailInfo(from=aoyagi@intra-mart.jp, to=[ueda@intra-
mart.jp], cc=null, bcc=null, subject=IM-LD Tutorial, body=Hello, IM-LogicDesignerLog))
[DEBUG] LOGIC_FLOW_LOG - [] [AFTER_EXECUTION] task result. (executeId=im_sendTextMail1,
taskId=ApplicationElementKey(elementId=im_sendTextMail), result=null)
[DEBUG] LOGIC_FLOW_LOG - [] [END_FLOW] session result. (flowId=tutorial_flow, executionTime=138ms, result={result=true})
...
```

## ログの設定について

ログの設定方法の詳細は「ログ仕様書」-「IM-LogicDesignerログ」を参照してください。

# プログラムからロジックフローの呼び出し

ロジックフローの呼び出し（実行）に関してはこれまでに、以下の2種類の方法を説明してきました。

- Swaggerを利用したロジックフローの実行
  - 「*基礎編 - ファースト・ステップ*」 -「*Swagger(SPEC)から実行する*」
- ジョブを利用したロジックフローの実行
  - 「*応用編 - より高度なフロー*」 -「*ジョブを利用したロジックフローの実行*」

これ以外に、ロジックフローはプログラム中から呼び出しを行うことが可能です。
プログラム中からの呼び出し方法の詳細は以下を参照してください。

- Java開発モデルから呼びだす
  - 「IM-LogicDesigner拡張プログラミングガイド」 -「付録」 -「独自のビジネスロジックからロジックフローを呼び出す方法 - Java開発モデル」
- スクリプト開発モデルから呼びだす

- 「IM-LogicDesigner拡張プログラミングガイド」 - 「付録」 - 「独自のビジネスロジックからロジックフローを呼び出す方法 - スクリプト開発モデル」

# 作成したロジックフローの設計書出力

IM-LogicDesignerでは作成したロジックフローの定義内容を元に、フローに関する設計書を出力することが可能です。
ここでは、設計書の出力方法と、出力される設計書の詳細を説明します。



## 設計書出力機能を利用する

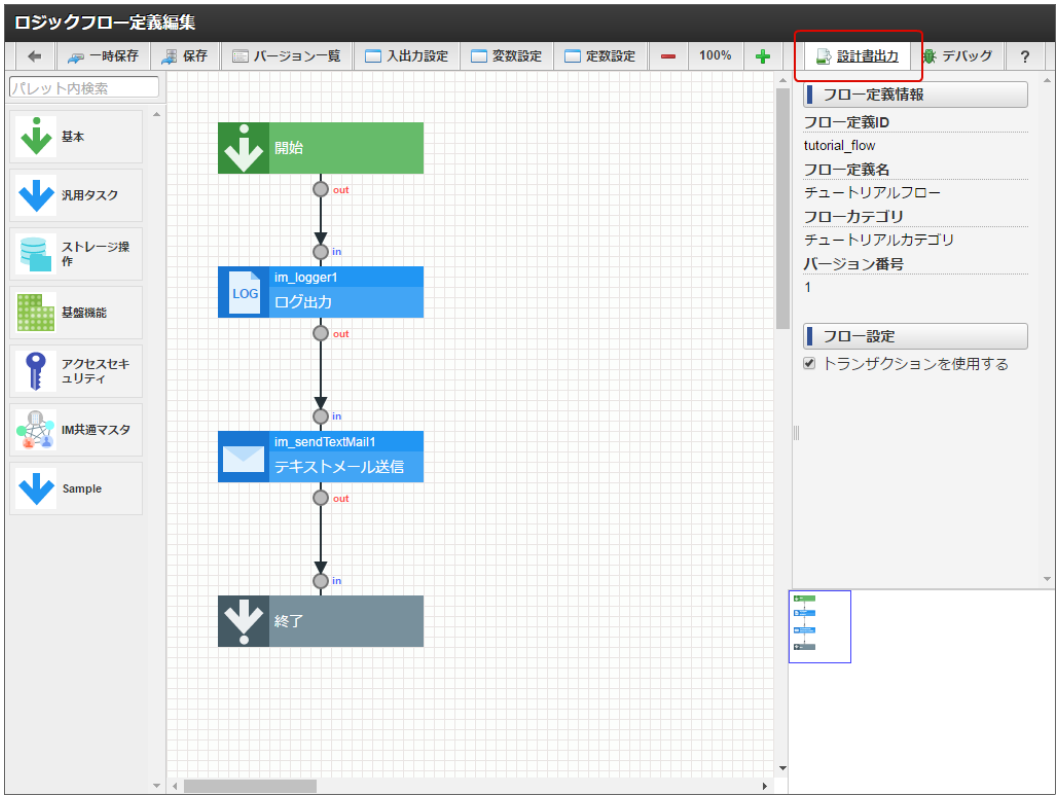
設計書出力機能は、IM-LogicDesignerの機能拡張モジュールとして提供されています。
intra-mart Accel Platformのデフォルト構成には含まれないため、以下のモジュールを構成に含めてwar作成を行ってください。

| モジュールID | モジュール名 | バージョン |
|---|---|---|
| jp.co.intra_mart.im_logic_development | IM-LogicDesigner 設計書出力 | 8.0.0 |

## 設計書を出力する

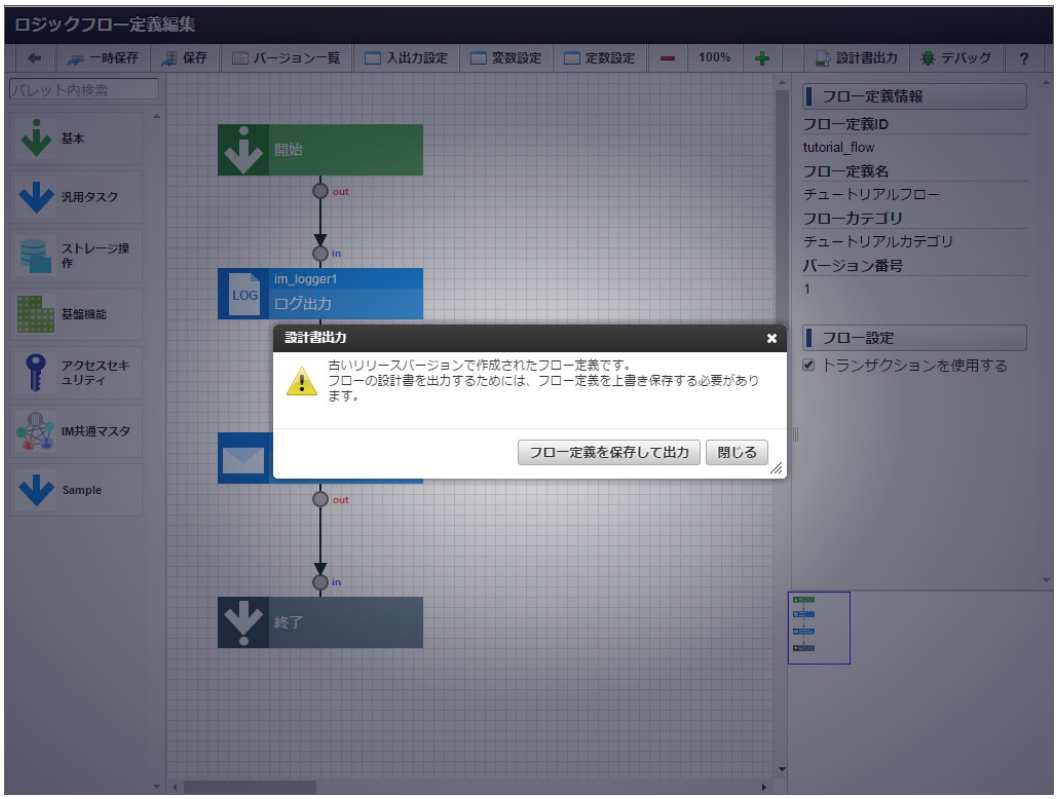
1. 「サイトマップ」→「LogicDesigner」→「フロー定義一覧」から、ロジックフロー定義一覧を開きます。
   左ペインのツリーから「チュートリアルカテゴリ」の開閉アイコンをクリックしカテゴリ配下を情報を表示します。
   フロー定義「チュートリアルフロー」を選択し編集ボタンをクリックします。

2. ロジックフロー定義編集画面上部、ヘッダ内の「設計書出力」をクリックします。




図：ヘッダ内の「設計書出力」

3. 2015 Winter(Lydia)以前のバージョンで保存されたフローの場合、一度保存を行う必要があります。




図：以前のバージョンの場合、一度保存を行うことを促すダイアログが表示

4. 設計書の出力を確認するダイアログが表示されるので、「決定」をクリックします。
5. 設計書が出力されます。

> **コラム**
>
> 設計書のファイルフォーマット
>
> IM-LogicDesignerが出力する設計書はXLSX形式で出力されます。

## 設計書の詳細

出力される設計書は以下の内容から構成されます。

| シート名 | 詳細 |
|---|---|
| 表紙 | 設計書の表紙です。<br>フローカテゴリ、フロー名、バージョン等が記載されます。 |
| フロー概要 | フローの詳細な情報が記載されます。<br>主に保存処理時に定義した内容、作成者/作成日時、更新者/更新日時などが記載されます。 |
| フロー図 | 作成したフローの全体図が記載されます。 |
| 入出力値 | フローに定義された入出力値の一覧が記載されます。 |
| 変数 | フローに定義された変数の一覧が記載されます。 |
| 定数値 | フローに定義された定数値の一覧が記載されます。 |
| エレメント一覧 | フローに配置されたエレメントの一覧が記載されます。<br>各エレメントについて、プロパティで設定された内容や、マッピングの有無などが記載されます。 |
| データマッピング | フローで定義されたマッピング情報の一覧が記載されます。 |
| 関連 | このフローが呼び出しているフロー、および、呼び出されているフローの一覧が記載されます。 |
| REST API | このフローに対するフロールーティング定義の一覧が記載されます。 |
| REST API レスポンスヘッダ | ※ このシートは2016 Summer(Nirvana)から追加されました。<br>このフローに対するフロールーティング定義のレスポンスヘッダの一覧が記載されます。 |

| シート名 | 詳細 |
| --- | --- |
| トリガ定義 | ※ このシートは2016 Summer(Nirvana)から追加されました。<br>このフローを利用しているトリガ定義の一覧が記載されます。 |
| トリガマッピング | ※ このシートは2016 Summer(Nirvana)から追加されました。<br>このフローを利用しているトリガ定義で定義されたマッピング情報の一覧が記載されます。 |

## チュートリアルデータのアーカイブファイル

この章では、これまでのチュートリアルで作成してきたロジックフローについて、そのアーカイブデータを提供します。
作成したロジックフローが動作しない場合などの参考用、「*応用編 - より高度なフロー*」を進める上での作業データ用などにご利用ください。



### 基礎編

- 基礎編 - チュートリアルロジックフローアーカイブ
  - **基礎編アーカイブ（ZIP形式）**

### 応用編

- 応用編
  - 「*複雑なフローの定義*」
    - 「*階層化された入力値・出力値の定義（Object型の定義）*」アーカイブ
      - **階層化された入力値・出力値の定義 アーカイブ（ZIP形式）**
    - 「*条件分岐を利用したフロー*」アーカイブ
      - **条件分岐を利用したフロー アーカイブ（ZIP形式）**
    - 「*繰り返し処理を利用したフロー*」アーカイブ
      - **繰り返し処理を利用したフロー アーカイブ（ZIP形式）**
    - 「*サブフロー呼び出し*」アーカイブ
      - **サブフロー呼び出し アーカイブ（ZIP形式）**

      > **コラム**
      > 呼び出し先の「*基礎編 - ファースト・ステップ*」ロジックフローは含まれません。

    - 「*変数を利用したフロー*」アーカイブ
      - **変数を活用する アーカイブ（ZIP形式）**
  - 「*ユーザ定義の作成*」
    - 「*SELECTを用いたユーザ定義（SQL）の作成*」アーカイブ
      - **ユーザ定義（SQL) - SELECT アーカイブ（ZIP形式）**
    - 「*INSERTを用いたユーザ定義（SQL）の作成*」アーカイブ
      - **ユーザ定義（SQL) - INSERT アーカイブ（ZIP形式）**
    - 「*UPDATEを用いたユーザ定義（SQL）の作成*」アーカイブ
      - **ユーザ定義（SQL) - UPDATE アーカイブ（ZIP形式）**
    - 「*DELETEを用いたユーザ定義（SQL）の作成*」アーカイブ
      - **ユーザ定義（SQL) - DELETE アーカイブ（ZIP形式）**
    - 「*ユーザ定義 - JavaScript*」アーカイブ
      - **ユーザ定義（JavaScript) アーカイブ（ZIP形式）**
    - 「*ユーザ定義 - REST*」アーカイブ
      - **ユーザ定義（REST) アーカイブ（ZIP形式）**
    - 「*ユーザ定義 - Database Fetch*」アーカイブ（※ 2016 Summer(Nirvana)追加。）
      - **ユーザ定義（DB Fetch) アーカイブ（ZIP形式）**
    - 「*ユーザ定義 - CSV Fetch*」アーカイブ（※ 2016 Summer(Nirvana)追加。）

        - **ユーザ定義（CSV Fetch）アーカイブ（ZIP形式）**
    - 「*申請を行うユーザ定義（BIS申請/承認）の作成*」アーカイブ（※ 2017 Winter(Rebecca)追加。）
        - **ユーザ定義（BIS申請/承認） - 申請 アーカイブ（ZIP形式）**
          *ユーザ定義 - BIS申請/承認* で利用する他のユーザ定義も含まれています。
        - **BIS定義（BIS申請/承認） - 申請 アーカイブ（ZIP形式）**
          *ユーザ定義 - BIS申請/承認* でのチュートリアル完成版のBIS定義です。
        - **ロジックフロー定義（BIS申請/承認） - 申請 アーカイブ（ZIP形式）**
          *ユーザ定義 - BIS申請/承認* でのチュートリアル完成版のロジックフロー定義です。

> **注意**
>
> 同じフローIDを持つロジックフローのインポートについて
>
> チュートリアルの関係上、以下のロジックフローは全て同じフロー定義IDで定義されています。
>
> - **基礎編**
>     - チュートリアルロジックフローアーカイブ
> - **応用編**
>     - 「*階層化された入力値・出力値の定義（Object型の定義）*」アーカイブ
>     - 「*条件分岐を利用したフロー*」アーカイブ
>     - 「*繰り返し処理を利用したフロー*」アーカイブ
>
> 上記アーカイブを続けて複数インポートした場合、正しくインポートが行われない場合があります。
> その際は、一度インポートを行ったロジックフローを削除した上で、再度インポートを行ってください。
> IM-LogicDesignerのインポート仕様の詳細は、「IM-LogicDesigner仕様書」 -「インポート・エクスポート」 を参照してください。

## インポート方法

ロジックフロー、フロールーティング、および、ユーザ定義のインポート方法を説明します。

1. 「サイトマップ」→「LogicDesigner」→「インポート」をクリックします。


図：

2. 「インポート」画面が表示されます。


図：

3. 「インポート設定」 -「インポートファイル」 -「ファイルを選択」をクリックし、インポート用のアーカイブファイルを選択します。


図：

4. 「インポート実行」をクリックします。


図：

以上でインポートが完了しました。