# 目次

# 改訂情報

| 変更年月日 | 変更内容 |
| --- | --- |
| 2024-04-01 | 初版 |
| 2024-10-01 | 第2版 以下を追加・変更しました。<br>■ 「*はじめに*」に Amazon Bedrock に関する説明を追加<br>■ 「*はじめに*」に弊社検証済みモデルの表を追加<br>■ 「*Amazon Bedrockのセットアップ*」を追加<br>■ 「*生成AI連携ドライバ設定*」に Amazon Bedrock に関する説明を追加<br>■ 「*IM-LogicDesignerタスク*」に Amazon Bedrock に関する説明を追加<br>■ 「*はじめに*」-「*主要コンポーネント*」内にコラムを追加<br>■ 「*IM-LogicDesignerタスク*」内のコラムにパラメータ制限に関する説明を追加<br>■ 「*セットアップ（各種製品アシスタント）*」を追加<br>■ 「*共通アシスタント実行画面*」を追加 |
| 2025-04-01 | 第3版 以下を追加・変更しました。<br>■ 「*はじめに*」の弊社検証済みモデルの表を更新<br>■ 「*生成AI連携ドライバ設定*」にデフォルトパラメータに関する説明を追加<br>■ 「*生成AI連携ドライバ設定*」の APIバージョンに関する記載について、明示的な指定を推奨しない変更を実施したため、記載を削除<br>■ 「*Azure OpenAI Serviceのセットアップ*」のモデルデプロイに関するコラムを更新<br>■ 「*Wiki アシスタントのセットアップ*」に日本語・英語以外での利用に関する注釈を追加<br>■ 「*Wiki アシスタントのセットアップ*」に Amazon Bedrock に関する説明を追加<br>■ 「*Wiki アシスタントのセットアップ*」の Azure OpenAI Service に関する説明を更新<br>■ 「*Wiki アシスタントのセットアップ*」のテキスト抽出設定に関する説明を更新<br>■ 「*ViewCreator SQLビルダ アシスタント のセットアップ*」に Amazon Bedrock に関する説明を追加<br>■ 「*Accel Studio アプリケーション作成 アシスタント のセットアップ*」を追加<br>■ 「*IM-LogicDesignerタスク*」に「*ロジックフローアシスタントでの利用を想定したタスク*」を追加<br>■ 「*IM-LogicDesignerタスク*」に「*ベクトルデータベース操作を想定したタスク*」を追加<br>■ 「*IM-LogicDesignerタスク*」に「*データ処理を想定したタスク*」を追加<br>■ 「*アシスタント定義*」を追加<br>■ 「*ロジックフローアシスタントの作成例*」を追加 |

| 変更年月日 | 変更内容 |
| --- | --- |
| 2025-10-01 | 第4版 以下を追加・変更しました。<br>■ 「*はじめに*」に記載していたモデルに関する情報を「*サポートモデル*」に移動、更新<br>■ 「*Azure OpenAI Serviceのセットアップ*」-「*モデルデプロイ*」内のモデルに関する記載を移動<br>■ 「*Amazon Bedrockのセットアップ*」-「*モデルアクセス設定*」内のモデルに関する記載を移動<br>■ 「*セットアップ（各種製品アシスタント）*」内のモデルに関する記載を移動<br>■ 「*生成AI連携ドライバ設定*」の Amazon Bedrock 画像生成モデルのデフォルト値を更新<br>■ 「*Amazon Bedrockのセットアップ*」の「*モデルアクセス設定*」を更新<br>■ 「*ViewCreator SQLビルダ アシスタント のセットアップ*」に Azure OpenAI Service に関する説明を追加<br>■ 「*Azure OpenAI Serviceのセットアップ*」にモデルデプロイ時に関するコラムを追加<br>■ 「*開発者向けガイド*」を追加<br>■ 「*ロジックフローアシスタントの作成例*」を「*開発者向けガイド*」の「*独自アシスタントの作成*」に移動<br>■ 「*独自アシスタントの作成*」の「*ベクトルデータ構築ジョブの作成*」にベクトルデータの削除方法に関する注釈を追加<br>■ 「*Wiki アシスタントのセットアップ*」 -「*生成AIサービスとモデル*」に Wiki アシスタント 専用の個別パラメータ設定に関するコラムを追加<br>■ 「*Wiki アシスタントのセットアップ*」に「*運用時の注意事項*」を追加<br>■ 「*Accel Studio テスト機能 Copilot のセットアップ*」を追加<br>■ 「*ViewCreator SQLビルダ アシスタント のセットアップ*」に「独自のテーブルメタデータ情報を追加する」へのリンクを追加 |
| 2026-01-30 | 第5版 以下を追加・変更しました。<br>■ 「*サポートモデル一覧*」-「*OpenAI を使用する場合*」のチャット アクションのサポートモデルおよび備考を更新<br>■ 「*サポートモデル一覧*」-「*Azure OpenAI Service を使用する場合*」のチャット アクションのサポートモデルおよび備考を更新<br>■ 「*リージョン対応情報*」の Azure OpenAI Service におけるリージョンを更新 |

| 変更年月日 | 変更内容 |
| --- | --- |
| 2026-04-01 | 第6版 以下を追加・変更しました。<br>■ 「*Amazon Bedrockのセットアップ*」-「*前提条件*」に以下を追加<br>　■ グローバルクロスリージョン推論に関する注意事項<br>　■ AWSパートナー経由でご契約の場合のAnthropicモデル利用に関する注意事項<br>■ 「*関連情報*」-「*アップデート時の確認事項*」-「*確認ポイント*」に以下を追加<br>　■ Amazon Bedrock でグローバルクロスリージョン推論が使用されているかの確認項目<br>　■ Amazon Bedrock でAnthropicモデルを利用する場合の注意事項<br>■ 2025年10月からの Amazon Bedrock でのモデルアクセス自動有効化に伴い、「*モデルアクセス設定*」のモデルアクセスに関する説明を更新<br>■ 2025年10月からの Amazon Bedrock でのモデルアクセス自動有効化に伴い、「*モデルの有効化*」の Amazon Bedrock モデル有効化手順を更新<br>■ 「*Amazon Bedrockのセットアップ*」に「*ガードレールの作成（任意）*」以降のガードレール関連のセクションを追加<br>■ 「*サポートモデル一覧*」-「*OpenAI を使用する場合*」の Accel Studio テスト機能 アシスタント チャットのデフォルトモデルおよびサポートモデルを更新<br>■ 「*サポートモデル一覧*」-「*Azure OpenAI Service を使用する場合*」の Accel Studio テスト機能 アシスタント チャットのサポートモデルを更新<br>■ 「*サポートモデル一覧*」-「*Amazon Bedrock を使用する場合*」のデフォルトモデルおよびサポートモデルを更新<br>　■ チャット アクションのサポートモデルを更新<br>　■ ViewCreator SQLビルダ アシスタント チャットのデフォルトモデルを更新<br>　■ Accel Studio アプリケーション作成 アシスタント のデフォルトモデルを更新<br>　■ Accel Studio テスト機能 アシスタント チャットのデフォルトモデルおよびサポートモデルを更新<br>■ 「*サポートモデル一覧*」-「*Amazon Bedrock を使用する場合*」のコラムにデフォルトモデルにグローバルクロスリージョン推論が使用される場合の対応を追記<br>■ 「*リージョン対応情報*」-「*Amazon Bedrock におけるリージョン*」にクロスリージョン推論専用モデルが存在する旨の注意を追加<br>■ 「*リージョン対応情報*」-「*Amazon Bedrock におけるリージョン*」-「モデルIDにおける地域プレフィックスの指定」を「*クロスリージョン推論プロファイルの利用*」に変更し内容を更新<br>■ 「*モデル更新履歴*」の Amazon Bedrock における Accel Studio アプリケーション作成 アシスタント のデフォルトモデルを更新<br>■ 「*機能別モデル設定パラメータ名一覧*」に ViewCreator SQLビルダ アシスタント 向けの OpenAI と Amazon Bedrock 用のモデル設定、個別パラメータに関する説明を追加<br>■ 「*共通アシスタント実行画面*」-「*アシスタントの実行*」にアシスタント実行のキーボードショートカットに関する説明を追加<br>■ Java API「*チャットサンプル*」、JavaScript API「*チャットサンプル*」を更新<br>■ 「*ロジックフローアシスタントの作成例*」に「*運用上の考慮事項*」を追加<br>■ 「*スクリプト開発モデル「アシスタント実行 UI」実装*」のエラーメッセージの対処に関する説明を更新<br>■ 「*JavaEE開発モデル「アシスタント実行 UI」実装*」のエラーメッセージの対処に関する説明を更新 |

# はじめに

## IM-Copilotとは

IM-Copilotは intra-mart Accel Platform上で生成AIを利用した業務アプリケーション開発、および
intra-mart Accel Platformの各種製品で生成AIを活用するための基盤機能です。

IM-Copilotの特徴は以下の通りです。

- 生成AIサービスによる違いを利用者に意識させない汎用的なインタフェースを提供します。
- 生成AIを企業で利用するために求められる統制機能(ログ保存、不正利用の防止)を提供します。
- 業務アプリから生成AIを利用しやすいGUI開発部品、APIを提供します。
- intra-mart Accel Platformの各種製品から生成AIを利用するためのアシスタント機能を提供します。

## IM-Copilotの利用者、利用方法

IM-Copilotの利用者、利用方法は下記を想定しています。

- 業務アプリから手軽に生成AIを呼び出して活用したい利用者（汎用利用者）
  - GUI開発(IM-LogicDesigner)により、プログラミング知識がなくても生成AIを利用可能です。
  - スクリプト開発、Java EE開発から汎用的なAPIを介して生成AIを利用可能です。
  - いずれの開発方法でも、利用者は生成AIサービスの違いを意識する必要はありません。
- 生成AIをより高度に活用した業務アプリを開発したい利用者（専用利用者） **（今後提供予定）**
  - 汎用利用よりも踏み込んだ生成AI活用(チューニング、特定の生成AIサービス向けの実装)が可能です。
  - 利用者は生成AIサービスの違いを意識する必要があります。

## 主要コンポーネント

IM-Copilotは、以下の主要なコンポーネント（機能）で構成されています。

### ドライバ

各種生成AIサービスに接続するための設定・接続処理を担うモジュールです。
APIキーやエンドポイントの管理、リクエストの送信、レスポンスの受信などを統一的に扱います。

IM-Copilotで接続可能な生成AIサービスは、以下の3種類です：

- OpenAI
  - 高精度な自然言語処理と多様なモデル選択肢を提供しており、汎用的な用途に幅広く対応します。
- Azure OpenAI Service
  - Microsoft Azure 上で OpenAI モデルを利用でき、企業向けのセキュリティやガバナンス機能が充実しています。
- Amazon Bedrock
  - AWS環境との高い親和性を活かし、Anthropic Claude や Amazon Nova などの生成AIモデルを柔軟に選択・活用できます。

バーチャルテナントごとに、接続対象の生成AIサービスを管理・設定できます。

### アクション

生成AIの違いを意識することなく利用できる、汎用的な基本的処理を提供しています。
たとえば、ユーザはAIの種類に依存せず、チャットを通じて要約や翻訳、分類などの操作を一貫して行うことができます。

また、アクションには以下のような開発スタイルに対応した機能が含まれています：

- ローコード向け機能
  - IM-LogicDesignerタスクとして提供しています。
- プロコード向けAPI

- JavaEE開発モデル、スクリプト開発モデル向けの APIを提供しています。

これにより、さまざまな開発ユーザ層に対応した柔軟な構成が提供されています。

## 各種アシスタント

既存プロダクトの機能を補助・強化するための支援モジュールです。
たとえば、「Wiki アシスタント」、「 Accel Studio アプリケーション作成 アシスタント」、「 ViewCreator SQLビルダ アシスタント」など、特定の機能に特化したアシスタントが用意されています。

## ベクトルデータベース操作機能

RAG（Retrieval-Augmented Generation）を構築する際に使用される、ベクトルデータベースとの連携機能です。
ドキュメントの埋め込み、検索、類似度計算などを行い、生成AIの応答精度を高めます。

## 共通アシスタント実行画面およびチャットUI埋め込み部品

ユーザとの対話を行うための共通UIを提供します。
アプリケーションに組み込めるチャット部品も含まれており、柔軟なUI統合が可能です。

## 今後の追加予定コンポーネント、改善予定

IM-Copilotでは、さらなる利便性と拡張性の向上を目指し、追加コンポーネントの導入を検討しています。

- 追加コンポーネントの導入
  - アシスタントの追加、AIエージェントなどの新機能を候補として検討しています。
- 既存機能の改善
  - PoC等でいただいたご意見・ご要望を踏まえ、使い勝手や安定性の向上に継続的に取り組んでいます。

今後のアップデートで順次公開を予定しています。

# サポートモデル

## サポートモデル一覧

このページでは、IM-Copilotの各機能で利用可能なサポートモデルの一覧と、ユースケースに応じたモデル選択のポイントを紹介します。

目次



### サポートモデルとは

サポートモデルは、各機能において選択可能な追加モデルです。
目的に応じて選択、品質向上・コスト削減・応答速度の改善などを図ることができます。

サポートモデルには「デフォルトモデル」と「サポートモデル」が存在します。

- デフォルトモデルは、設定ファイル（IM-Copilot生成AI連携ドライバ設定）において基底のモデルとして定義されており、各機能でモデルが明示的に指定されていない場合に使用されます。
- サポートモデルは、ユーザが任意にデフォルトモデルとして指定できるほか、各機能ごとに個別に設定することも可能です。この場合、機能ごとに設定されたサポートモデルがデフォルトモデルよりも優先され、上書きされる形で使用されます。

この仕組みにより、柔軟かつ細やかなモデル選択が可能となり、用途や性能要件に応じた最適なモデル構成を実現できます。

> **コラム**
>
> デフォルトモデルの更新について:
> 各生成AIサービスのモデルリリースや弊社のアップデートタイミングに応じて更新される場合があります。
> アップデート更新に合わせてモデルデプロイを行ってください。（推奨）
> 以前のモデルを利用したい場合は各機能で個別にモデルを指定してください。（機能側で指定可能な場合）

> **コラム**
>
> intra-mart Accel Platform 2025 Spring(Kamille)より、デフォルトモデルを指定するためのパラメータ設定が追加されました。
> 詳細は 「設定ファイルリファレンス」 - 「IM-Copilot生成AI連携ドライバ設定」 を参照してください。

### ユースケースとモデル選択

モデル選択が必要になるユースケースとして、以下の3つがあります：

新規セットアップ時：

- 利用可能な機能や、各モデルの設定可否について確認します。
- 必要に応じて、使用するモデルを有効化する作業が発生する場合があります。

アップデート時：

- デフォルトモデルやサポート対象モデルに変更がないかを確認します。
- モデルの追加や変更があった場合は、必要に応じて新しいモデルを有効化します。

開発・運用中のチューニング：

- 応答品質の向上やコスト削減を目的として、使用するモデルの見直しや変更を行います。
- チャットタスクや API においては、入力値でモデルを指定することが可能であり、再起動を伴わず柔軟な切り替えができます。

## 機能別サポートモデル一覧

### OpenAI を使用する場合

| 機能 | デフォルトモデル | サポートモデル | 備考 |
|---|---|---|---|
| チャット アクション | gpt-4o-mini | gpt-4o<br>gpt-4-turbo-2024-04-09<br>gpt-4.1-mini<br>gpt-4.1<br>gpt-5.1<br>gpt-5.2 | 高速応答と低コストを重視する場合は gpt-4o-mini や gpt-4.1-mini を推奨<br>※モデルを gpt-5.1、gpt-5.2に指定する場合、モデルの仕様により、stops パラメータは利用できません。<br>※モデルを gpt-5.1、gpt-5.2に指定し、かつストリーミング出力を無効にする場合、モデルの仕様により、返答内容のトークン数より maxTokens の値が小さいと返答が空文字になる可能性があるため、maxTokens は十分大きな値を設定することを推奨します。 |
| 埋め込み アクション | text-embedding-3-small | text-embedding-ada-002 | |
| 画像生成 アクション | dall-e-2 | dall-e-3 | |
| 音声生成 アクション | tts-1 | tts-1-hd | |
| 文字起こし アクション | whisper-1 | | |
| Wiki アシスタント チャット | gpt-4o-mini | gpt-4.1-mini<br>gpt-4.1 | |
| Wiki アシスタント 埋め込み | text-embedding-3-small | | |
| ViewCreator SQLビルダ アシスタント チャット | gpt-4o-mini | | |
| ViewCreator SQLビルダ アシスタント 埋め込み | text-embedding-3-small | | |
| Accel Studio アプリケーション作成 アシスタント チャット | gpt-4o-mini | | |
| Accel Studio テスト機能 アシスタント チャット | gpt-5.1 | gpt-5-mini<br>gpt-5.2<br>gpt-5.4 | |

### Azure OpenAI Service を使用する場合

| 機能 | デフォルトモデル | サポートモデル | 備考 |
|---|---|---|---|

| 機能 | デフォルトモデル | サポートモデル | 備考 |
|---|---|---|---|
| チャット アクション | gpt-4o-mini | gpt-4o<br>gpt-4.1-mini<br>gpt-4.1<br>gpt-5.1<br>gpt-5.2 | 高速応答と低コストを重視する場合は gpt-4o-mini や gpt-4.1-mini を推奨<br>※モデルを gpt-5.1、gpt-5.2に指定する場合、モデルの仕様により、stops パラメータは利用できません。<br>※モデルを gpt-5.1、gpt-5.2に指定し、かつ ストリーミング出力を無効にする場合、モデルの仕様により、返答内容のトークン数より maxTokens の値が小さいと返答が空文字になる可能性があるため、maxTokens は十分大きな値を設定することを推奨します。 |
| 埋め込み アクション | text-embedding-ada-002 | text-embedding-3-small | |
| 画像生成 アクション | | | Azure OpenAI Service 側のサポート終了に伴い、現在利用できません。代替モデルへの対応は今後予定しています。詳細は FAQ を参照してください。 |
| 音声生成 アクション | | | |
| 文字起こし アクション | whisper | | |
| Wiki アシスタント チャット | gpt-4o-mini | gpt-4o<br>gpt-4.1-mini<br>gpt-4.1 | |
| Wiki アシスタント 埋め込み | text-embedding-ada-002 | text-embedding-3-small | |
| ViewCreator SQLビルダ アシスタント チャット | gpt-4o-mini | | |
| ViewCreator SQLビルダ アシスタント 埋め込み | text-embedding-3-small | text-embedding-ada-002 | |
| Accel Studio アプリケーション作成 アシスタント チャット | gpt-4o-mini | | |
| Accel Studio テスト機能 アシスタント チャット | gpt-5-mini | gpt-5.1<br>gpt-5.2<br>gpt-5.4 | |

> **コラム**
>
> Azure OpenAI Service においては、モデル名ではなくデフォルトのデプロイ名を表しています。
> 上記以外の名称でデプロイを行っている場合は、そのデプロイ名を指定してください。

### Amazon Bedrock を使用する場合

| 機能 | デフォルトモデル | サポートモデル | 備考 |
|---|---|---|---|

| 機能 | デフォルトモデル | サポートモデル | 備考 |
| --- | --- | --- | --- |
| チャット アクション | anthropic.claude-3-haiku-20240307-v1:0 | anthropic.claude-sonnet-4-6<br><br>anthropic.claude-sonnet-4-5-20250929-v1:0<br><br>anthropic.claude-sonnet-4-20250514-v1:0<br><br>anthropic.claude-3-7-sonnet-20250219-v1:0<br><br>anthropic.claude-3-5-sonnet-20240620-v1:0<br><br>anthropic.claude-3-5-haiku-20241022-v1:0<br><br>amazon.nova-pro-v1:0<br><br>amazon.nova-lite-v1:0 | Claude：精度重視・文脈理解に優れた設計<br><br>Nova：効率重視・軽量で高速な応答設計<br><br>※モデルを anthropic.claude-sonnet-4-5-20250929-v1:0 、anthropic.claude-sonnet-4-6 に指定する場合、モデルの仕様により、topP と temperature を同時に設定することはできません。 |
| 埋め込み アクション | amazon.titan-embed-text-v1 | | |
| 画像生成 アクション | amazon.nova-canvas-v1:0 | | |
| 音声生成 アクション | | | |
| 文字起こし アクション | | | |
| Wiki アシスタント チャット | anthropic.claude-3-haiku-20240307-v1:0 | anthropic.claude-3-7-sonnet-20250219-v1:0<br><br>anthropic.claude-3-5-sonnet-20240620-v1:0<br><br>anthropic.claude-3-5-haiku-20241022-v1:0<br><br>amazon.nova-pro-v1:0<br><br>amazon.nova-lite-v1:0 | Claude：精度重視・文脈理解に優れた設計<br><br>Nova：効率重視・軽量で高速な応答設計 |
| Wiki アシスタント 埋め込み | amazon.titan-embed-text-v1 | | |
| ViewCreator SQLビルダ アシスタント チャット | anthropic.claude-haiku-4-5-20251001-v1:0 | | ※ デフォルト設定ではグローバルクロスリージョン推論が使用されます。 データの処理リージョンを限定する場合は地域プレフィックスの変更が必要です。 詳細は「*クロスリージョン推論プロファイルの利用*」を参照してください。 |
| ViewCreator SQLビルダ アシスタント 埋め込み | amazon.titan-embed-text-v1 | | |
| Accel Studio アプリケーション作成 アシスタント チャット | anthropic.claude-haiku-4-5-20251001-v1:0 | | ※ デフォルト設定ではグローバルクロスリージョン推論が使用されます。 データの処理リージョンを限定する場合は地域プレフィックスの変更が必要です。 詳細は「*クロスリージョン推論プロファイルの利用*」を参照してください。 |

| 機能 | デフォルトモデル | サポートモデル | 備考 |
| --- | --- | --- | --- |
| Accel Studio テスト機能 アシスタント チャット | anthropic.claude-sonnet-4-6 | anthropic.claude-sonnet-4-20250514-v1:0<br><br>anthropic.claude-opus-4-6-v1 | ※ デフォルト設定ではグローバルクロスリージョン推論が使用されます。 データの処理リージョンを限定する場合は地域プレフィックスの変更が必要です。 詳細は「*クロスリージョン推論プロファイルの利用*」を参照してください。 |

コラム

Amazon Bedrock の一部のモデルはクロスリージョン推論プロファイル経由でのみ利用可能です。これらのモデルを呼び出す際は、モデルIDに地域プレフィックスの付与が必要です。

クロスリージョン推論プロファイルにはグローバルと地理的の2種類があります。
弊社が標準で指定しているデフォルトモデルの一部はグローバルクロスリージョン推論で動作します。
データの処理リージョンを特定の地理的範囲に限定する必要がある場合は、地理的クロスリージョン推論プロファイルへの変更をご検討ください。

詳細は「*クロスリージョン推論プロファイルの利用*」を参照してください。

### モデル選定のポイント

「サポートモデル」の観点だけでなく、目的に応じた性能（品質・コスト・応答速度）や技術的制約も考慮する必要があります。

- 品質重視
  - 高性能モデル（例：GPT-5.2、Claude Sonnet）を使用することで、高精度な応答が可能です。
- コスト重視
  - 軽量モデル（例：GPT-4o mini、Claude Haiku、Nova Lite）を選ぶことで、コストを抑えることができます。
- 技術的制約
  - モデルごとにパラメータ制限などがあります。選定時は公式ドキュメントの確認や試用による事前検証を推奨します。

### モデルの有効化について

一部のモデルは利用前に有効化が必要です。
詳細は「*関連情報*」-「*モデルの有効化*」を参照してください。

## モデル更新履歴

このページでは、各機能におけるモデルの更新履歴を記録しています。
デフォルトモデルの変更、新たなモデルの追加、サポート終了などの情報を確認できます。

目次



## 更新履歴一覧の凡例（用語の説明）

### 機能

IM-Copilotにおいて、生成AIモデルを活用して提供される各種機能を指します。
このページでは、**各アップデートリリース時点で提供されている機能** を対象として記録しています。

### デフォルトモデル

デフォルトモデルは、各機能において **明示的なモデル指定がない場合に使用される基準のモデル** です。
これは設定ファイル（IM-Copilot生成AI連携ドライバ設定）で定義されており、通常の利用時にはこのモデルが自動的に適用されます。

### サポートモデル

サポートモデルは、各機能で **利用可能なモデルの一覧** を指します。
ユーザはこの中から任意のモデルを選択して、デフォルトモデルとして上書き設定できます。
また、機能ごとに個別にモデルを指定することも可能で、その場合は **指定されたサポートモデルが優先されて使用** されます。

## 更新履歴一覧

### OpenAI モデル更新履歴

#### 2024 Spring(Iris)

2024 Spring(Iris) アップデートリリース時の 2024 年 4 月 1 日時点の OpenAI 利用におけるサポートモデルは以下の通りです。

| 機能 | デフォルトモデル | サポートモデル |
|---|---|---|
| チャット アクション | gpt-3.5-turbo-16k | |
| 画像生成 アクション | dall-e-2 | dall-e-3 |
| 埋め込み アクション | text-embedding-3-small | text-embedding-ada-002 |
| 文字起こし アクション | whisper-1 | |
| 音声生成 アクション | tts-1 | tts-1-hd |

### 2024 Autumn(Jasmine)

2024 Autumn(Jasmine) アップデートリリース時の 2024 年 10 月 1 日時点の OpenAI 利用におけるサポートモデルは以下の通りです。
新たに GPT-4 系のモデルが追加されました。
デフォルトモデルが「gpt-4o-mini」に変更されました。

| 機能 | デフォルトモデル | サポートモデル |
|---|---|---|
| チャット アクション | **gpt-4o-mini** | **gpt-4o**<br>**gpt-4-turbo-2024-04-09**<br>**gpt-4-vision-preview**<br>gpt-3.5-turbo-16k |
| 画像生成 アクション | dall-e-2 | dall-e-3 |
| 埋め込み アクション | text-embedding-3-small | text-embedding-ada-002 |
| 文字起こし アクション | whisper-1 | |
| 音声生成 アクション | tts-1 | tts-1-hd |
| Wiki アシスタント チャット | **gpt-4o-mini** | |
| Wiki アシスタント 埋め込み | text-embedding-3-small | |
| ViewCreator SQLビルダ アシスタント チャット | **gpt-4o-mini** | |
| ViewCreator SQLビルダ アシスタント 埋め込み | text-embedding-3-small | |

### 2025 Spring(Kamille)

2025 Spring(Kamille) アップデートリリース時の 2025 年 4 月 1 日時点の OpenAI 利用におけるサポートモデルは以下の通りです。
サポートモデルの追加、削除はありません。

| 機能 | デフォルトモデル | サポートモデル |
|---|---|---|
| チャット アクション | gpt-4o-mini | gpt-4o<br>gpt-4-turbo-2024-04-09<br>gpt-3.5-turbo-16k<br>gpt-4-vision-preview |
| 画像生成 アクション | dall-e-2 | dall-e-3 |
| 埋め込み アクション | text-embedding-3-small | text-embedding-ada-002 |
| 文字起こし アクション | whisper-1 | |
| 音声生成 アクション | tts-1 | tts-1-hd |
| Wiki アシスタント チャット | gpt-4o-mini | |
| Wiki アシスタント 埋め込み | text-embedding-3-small | |
| ViewCreator SQLビルダ アシスタント チャット | gpt-4o-mini | |
| ViewCreator SQLビルダ アシスタント 埋め込み | text-embedding-3-small | |

| 機能 | デフォルトモデル | サポートモデル |
| --- | --- | --- |
| Accel Studio アプリケーション作成 アシスタント チャット | gpt-4o-mini | |

### 2025 Autumn(Lilac)

2025 Autumn(Lilac) アップデートリリース時の 2025 年 10 月 1 日時点の OpenAI 利用におけるサポートモデルは以下の通りです。
GPT-4.1 系のモデルが追加されました。
「gpt-3.5-turbo-16k」および「gpt-4-vision-preview」は、 OpenAI 側のサポート終了に伴い削除されました。現在は新規利用ができなくなっています。
2026 年 1 月末に gpt-5.1 および gpt-5.2 がサポートモデルに追加されました。

| 機能 | デフォルトモデル | サポートモデル |
| --- | --- | --- |
| チャット アクション | gpt-4o-mini | gpt-4o<br>gpt-4-turbo-2024-04-09<br>**gpt-4.1-mini**<br>**gpt-4.1**<br>**gpt-5.1**<br>**gpt-5.2** |
| 画像生成 アクション | dall-e-2 | dall-e-3 |
| 埋め込み アクション | text-embedding-3-small | text-embedding-ada-002 |
| 文字起こし アクション | whisper-1 | |
| 音声生成 アクション | tts-1 | tts-1-hd |
| Wiki アシスタント チャット | gpt-4o-mini | **gpt-4.1-mini**<br>**gpt-4.1** |
| Wiki アシスタント 埋め込み | text-embedding-3-small | |
| ViewCreator SQLビルダ アシスタント チャット | gpt-4o-mini | |
| ViewCreator SQLビルダ アシスタント 埋め込み | text-embedding-3-small | |
| Accel Studio アプリケーション作成 アシスタント チャット | gpt-4o-mini | |
| Accel Studio テスト機能 アシスタント チャット | **gpt-5** | **gpt-5-mini** |

### 2026 Spring(Mimosa)

2026 Spring(Mimosa) アップデートリリース時の 2026 年 4 月 1 日時点の OpenAI 利用におけるサポートモデルは以下の通りです。

| 機能 | デフォルトモデル | サポートモデル |
| --- | --- | --- |
| チャット アクション | gpt-4o-mini | gpt-4o<br>gpt-4-turbo-2024-04-09<br>gpt-4.1-mini<br>gpt-4.1<br>gpt-5.1<br>gpt-5.2 |
| 画像生成 アクション | dall-e-2 | dall-e-3 |
| 埋め込み アクション | text-embedding-3-small | text-embedding-ada-002 |
| 文字起こし アクション | whisper-1 | |

| 機能 | デフォルトモデル | サポートモデル |
|---|---|---|
| 音声生成 アクション | tts-1 | tts-1-hd |
| Wiki アシスタント チャット | gpt-4o-mini | gpt-4.1-mini<br>gpt-4.1 |
| Wiki アシスタント 埋め込み | text-embedding-3-small | |
| ViewCreator SQLビルダ アシスタント チャット | gpt-4o-mini | |
| ViewCreator SQLビルダ アシスタント 埋め込み | text-embedding-3-small | |
| Accel Studio アプリケーション作成 アシスタント チャット | gpt-4o-mini | |
| Accel Studio テスト機能 アシスタント チャット | **gpt-5.1** | gpt-5-mini<br>**gpt-5.2**<br>**gpt-5.4** |

## Azure OpenAI Service モデル更新履歴

### 2024 Spring(Iris)

2024 Spring(Iris) アップデートリリース時の 2024 年 4 月 1 日時点の Azure OpenAI Service 利用におけるサポートモデルは以下の通りです。

| 機能 | デフォルトモデル | サポートモデル |
|---|---|---|
| チャット アクション | gpt-35-turbo | |
| 画像生成 アクション | dall-e-3 | |
| 埋め込み アクション | text-embedding-ada-002 | |
| 文字起こし アクション | whisper | |
| 音声生成 アクション | | |

### 2024 Autumn(Jasmine)

2024 Autumn(Jasmine) アップデートリリース時の 2024 年 10 月 1 日時点の Azure OpenAI Service 利用におけるサポートモデルは以下の通りです。
新たに GPT-4 系のモデルが追加されました。
デフォルトモデルが「gpt-4o」に変更されました。

| 機能 | デフォルトモデル | サポートモデル |
|---|---|---|
| チャット アクション | **gpt-4o** | gpt-35-turbo |
| 画像生成 アクション | dall-e-3 | |
| 埋め込み アクション | text-embedding-ada-002 | |
| 文字起こし アクション | whisper | |
| 音声生成 アクション | | |
| Wiki アシスタント チャット | **gpt-4o** | |
| Wiki アシスタント 埋め込み | text-embedding-ada-002 | |

### 2025 Spring(Kamille)

2025 Spring(Kamille) アップデートリリース時の 2025 年 4 月 1 日時点の Azure OpenAI Service 利用におけるサポートモデルは以下の通りです。
新たに GPT-4 系のモデルが追加されました。
デフォルトモデルが「gpt-4o-mini」に変更されました。

| 機能 | デフォルトモデル | サポートモデル |
|---|---|---|
| チャット アクション | **gpt-4o-mini** | gpt-4o<br>gpt-35-turbo |
| 画像生成 アクション | dall-e-3 | |
| 埋め込み アクション | text-embedding-ada-002 | |
| 文字起こし アクション | whisper | |
| 音声生成 アクション | | |
| Wiki アシスタント チャット | **gpt-4o-mini** | gpt-4o |
| Wiki アシスタント 埋め込み | text-embedding-ada-002 | |
| Accel Studio アプリケーション作成 アシスタント チャット | **gpt-4o-mini** | |

### 2025 Autumn(Lilac)

2025 Autumn(Lilac) アップデートリリース時の 2025 年 10 月 1 日時点の Azure OpenAI Service 利用におけるサポートモデルは以下の通りです。
GPT-4.1 系のモデルが追加されました。
「text-embedding-3-small」が追加されました。
「gpt-35-turbo」は Azure OpenAI Service 側のサポート終了に伴い削除されました。 Azure OpenAI Service では一部の「gpt-35-turbo」モデルが引き続き提供されていますが、利用可能なモデルは順次更新されており、最新モデルへの移行が推奨されています。

| 機能 | デフォルトモデル | サポートモデル |
|---|---|---|
| チャット アクション | gpt-4o-mini | gpt-4o<br>**gpt-4.1-mini**<br>**gpt-4.1** |
| 画像生成 アクション | dall-e-3 | |
| 埋め込み アクション | text-embedding-ada-002 | **text-embedding-3-small** |
| 文字起こし アクション | whisper | |
| 音声生成 アクション | | |
| Wiki アシスタント チャット | gpt-4o-mini | gpt-4o<br>**gpt-4.1-mini**<br>**gpt-4.1** |
| Wiki アシスタント 埋め込み | text-embedding-ada-002 | **text-embedding-3-small** |
| ViewCreator SQLビルダ アシスタント チャット | gpt-4o-mini | |
| ViewCreator SQLビルダ アシスタント 埋め込み | text-embedding-3-small | text-embedding-ada-002 |
| Accel Studio アプリケーション作成 アシスタント チャット | gpt-4o-mini | |
| Accel Studio テスト機能 アシスタント チャット | **gpt-5-mini** | |

### 2026 Spring(Mimosa)

2026 Spring(Mimosa) アップデートリリース時の 2026 年 4 月 1 日時点の Azure OpenAI Service 利用におけるサポートモデルは以下の通りです。
「dall-e-3」は、 Azure OpenAI Service 側のサポート終了に伴い削除されました。現在は新規利用ができなくなっています。詳細は FAQ を参照してください。

| 機能 | デフォルトモデル | サポートモデル |
|---|---|---|

| 機能 | デフォルトモデル | サポートモデル |
| --- | --- | --- |
| チャット アクション | gpt-4o-mini | gpt-4o<br>gpt-4.1-mini<br>gpt-4.1<br>**gpt-5.1**<br>**gpt-5.2** |
| 画像生成 アクション | | |
| 埋め込み アクション | text-embedding-ada-002 | text-embedding-3-small |
| 文字起こし アクション | whisper | |
| 音声生成 アクション | | |
| Wiki アシスタント チャット | gpt-4o-mini | gpt-4o<br>gpt-4.1-mini<br>gpt-4.1 |
| Wiki アシスタント 埋め込み | text-embedding-ada-002 | text-embedding-3-small |
| ViewCreator SQLビルダ アシスタント チャット | gpt-4o-mini | |
| ViewCreator SQLビルダ アシスタント 埋め込み | text-embedding-3-small | text-embedding-ada-002 |
| Accel Studio アプリケーション作成 アシスタント チャット | gpt-4o-mini | |
| Accel Studio テスト機能 アシスタント チャット | gpt-5-mini | **gpt-5.1**<br>**gpt-5.2**<br>**gpt-5.4** |

## Amazon Bedrock モデル更新履歴

### 2024 Autumn(Jasmine)

2024 Autumn(Jasmine) アップデートリリース時の 2024 年 10 月 1 日時点の Amazon Bedrock 利用におけるサポートモデルは以下の通りです。

| 機能 | デフォルトモデル | サポートモデル |
| --- | --- | --- |
| チャット アクション | anthropic.claude-3-haiku-20240307-v1:0 | anthropic.claude-3-sonnet-20240229-v1:0<br>anthropic.claude-v2<br>anthropic.claude-instant-v1 |
| 画像生成 アクション | | |
| 埋め込み アクション | amazon.titan-embed-text-v1 | |
| 文字起こし アクション | | |
| 音声生成 アクション | | |

### 2025 Spring(Kamille)

2025 Spring(Kamille) アップデートリリース時の 2025 年 4 月 1 日時点の Amazon Bedrock 利用におけるサポートモデルは以下の通りです。
「anthropic.claude-3-5-sonnet-20240620-v1:0」が追加されました。
リリース時の Accel Studio アプリケーション作成 アシスタント チャットのデフォルトモデルは「anthropic.claude-3-5-sonnet-20240620-v1:0」でしたが、 Amazon Bedrock 側のサポート終了に伴い、2026 年 4 月 1 日リリースのパッチにより「anthropic.claude-haiku-4-5-20251001-v1:0」に変更されました。

| 機能 | デフォルトモデル | サポートモデル |
| --- | --- | --- |
| チャット アクション | anthropic.claude-3-haiku-20240307-v1:0 | anthropic.claude-3-sonnet-20240229-v1:0<br>anthropic.claude-v2<br>anthropic.claude-instant-v1 |
| 画像生成 アクション | | |
| 埋め込み アクション | amazon.titan-embed-text-v1 | |
| 文字起こし アクション | | |
| 音声生成 アクション | | |
| Wiki アシスタント チャット | anthropic.claude-3-haiku-20240307-v1:0 | |
| Wiki アシスタント 埋め込み | amazon.titan-embed-text-v1 | |
| ViewCreator SQLビルダ アシスタント チャット | anthropic.claude-3-haiku-20240307-v1:0 | |
| ViewCreator SQLビルダ アシスタント 埋め込み | amazon.titan-embed-text-v1 | |
| Accel Studio アプリケーション作成 アシスタント チャット | **anthropic.claude-haiku-4-5-20251001-v1:0** | |

### 2025 Autumn(Lilac)

2025 Autumn(Lilac) アップデートリリース時の 2025 年 10 月 1 日時点の Amazon Bedrock 利用におけるサポートモデルは以下の通りです。
Claude 系の新しいモデルが追加されました。
Nova 系のモデルが追加されました。
「anthropic.claude-v2」および「anthropic.claude-instant-v1」は、 Amazon Bedrock 側のサポート終了に伴い削除されました。現在は新規利用ができなくなっています。
リリース時の Accel Studio アプリケーション作成 アシスタント チャットのデフォルトモデルは「anthropic.claude-3-5-sonnet-20240620-v1:0」でしたが、 Amazon Bedrock 側のサポート終了に伴い、2026 年 4 月 1 日リリースのパッチにより「anthropic.claude-haiku-4-5-20251001-v1:0」に変更されました。

| 機能 | デフォルトモデル | サポートモデル |
| --- | --- | --- |
| チャット アクション | anthropic.claude-3-haiku-20240307-v1:0 | **anthropic.claude-3-7-sonnet-20250219-v1:0**<br>**anthropic.claude-3-5-sonnet-20240620-v1:0**<br>**anthropic.claude-3-5-haiku-20241022-v1:0**<br>anthropic.claude-3-sonnet-20240229-v1:0<br>**amazon.nova-pro-v1:0**<br>**amazon.nova-lite-v1:0** |
| 画像生成 アクション | **amazon.nova-canvas-v1:0** | |
| 埋め込み アクション | amazon.titan-embed-text-v1 | |
| 文字起こし アクション | | |
| 音声生成 アクション | | |

| 機能 | デフォルトモデル | サポートモデル |
| --- | --- | --- |
| Wiki アシスタント チャット | anthropic.claude-3-haiku-20240307-v1:0 | **anthropic.claude-3-7-sonnet-20250219-v1:0**<br>**anthropic.claude-3-5-sonnet-20240620-v1:0**<br>**anthropic.claude-3-5-haiku-20241022-v1:0**<br>**amazon.nova-pro-v1:0**<br>**amazon.nova-lite-v1:0** |
| Wiki アシスタント 埋め込み | amazon.titan-embed-text-v1 | |
| ViewCreator SQLビルダ アシスタント チャット | anthropic.claude-3-haiku-20240307-v1:0 | |
| ViewCreator SQLビルダ アシスタント 埋め込み | amazon.titan-embed-text-v1 | |
| Accel Studio アプリケーション作成 アシスタント チャット | anthropic.claude-haiku-4-5-20251001-v1:0 | |
| Accel Studio テスト機能 アシスタント チャット | **anthropic.claude-sonnet-4-20250514-v1:0** | **anthropic.claude-3-7-sonnet-20250219-v1:0** |

### 2026 Spring(Mimosa)

2026 Spring(Mimosa) アップデートリリース時の 2026 年 4 月 1 日時点の Amazon Bedrock 利用におけるサポートモデルは以下の通りです。
「anthropic.claude-3-sonnet-20240229-v1:0」は、 Amazon Bedrock 側のサポート終了に伴い削除されました。現在は新規利用ができなくなっています。

| 機能 | デフォルトモデル | サポートモデル |
| --- | --- | --- |
| チャット アクション | anthropic.claude-3-haiku-20240307-v1:0 | **anthropic.claude-sonnet-4-6**<br>**anthropic.claude-sonnet-4-5-20250929-v1:0**<br>**anthropic.claude-sonnet-4-20250514-v1:0**<br>anthropic.claude-3-7-sonnet-20250219-v1:0<br>anthropic.claude-3-5-sonnet-20240620-v1:0<br>anthropic.claude-3-5-haiku-20241022-v1:0<br>amazon.nova-pro-v1:0<br>amazon.nova-lite-v1:0 |
| 画像生成 アクション | amazon.nova-canvas-v1:0 | |
| 埋め込み アクション | amazon.titan-embed-text-v1 | |
| 文字起こし アクション | | |
| 音声生成 アクション | | |

| 機能 | デフォルトモデル | サポートモデル |
| --- | --- | --- |
| Wiki アシスタント チャット | anthropic.claude-3-haiku-20240307-v1:0 | anthropic.claude-3-7-sonnet-20250219-v1:0<br>anthropic.claude-3-5-sonnet-20240620-v1:0<br>anthropic.claude-3-5-haiku-20241022-v1:0<br>amazon.nova-pro-v1:0<br>amazon.nova-lite-v1:0 |
| Wiki アシスタント 埋め込み | amazon.titan-embed-text-v1 | |
| ViewCreator SQLビルダ アシスタント チャット | **anthropic.claude-haiku-4-5-20251001-v1:0** | |
| ViewCreator SQLビルダ アシスタント 埋め込み | amazon.titan-embed-text-v1 | |
| Accel Studio アプリケーション作成 アシスタント チャット | **anthropic.claude-haiku-4-5-20251001-v1:0** | |
| Accel Studio テスト機能 アシスタント チャット | **anthropic.claude-sonnet-4-6** | anthropic.claude-sonnet-4-20250514-v1:0<br>**anthropic.claude-opus-4-6-v1** |

## サポートモデル変更の種類、影響

### サポートモデルの変更（デフォルトモデル設定の更新）

以下のような影響が考えられます：

- 出力結果が変化する可能性があります。
- 新しいモデルを利用するには、事前に有効化設定が必要な場合があります。
- モデルの特性（応答速度、精度、対応言語など）に応じて、パフォーマンスやコストが変動することがあります。

### サポートモデルの追加（新規モデルの導入）

以下のような利点が考えられます：

- ユースケースに応じて、より最適なモデルを選択できる柔軟性が向上します。
- 新モデルは、既存モデルよりも高速・高精度な応答を提供する場合があります。

※ 新規モデルはデフォルトでは有効化されていない場合があります。必要に応じて設定を確認してください。
※ モデルの有効化手順については「*関連情報*」-「*モデルの有効化*」を参照してください。

### サポートモデルの廃止

以下のような影響が考えられます：

- 対象モデルを利用していた機能でエラーが発生する可能性があります。モデルの切り替えが必要です。
- モデルの切り替えに伴い、出力品質や表現スタイルが変わる場合があります。

※ 廃止モデルの利用継続はできません。早めの移行を推奨します。

## 今後の更新予定

サポート対象モデルの追加やデフォルトモデルの変更については、生成AIモデルの進化に応じて、各アップデートのタイミングで順次対応・反映していく方針です。

※ 最新のサポートモデルおよび補足情報については「*サポートモデル*」を参照してください。

# リージョン対応情報

このページでは、 IM-Copilot でサポートしているモデルを中心に、どのリージョンで利用可能かを解説します。
特に Azure OpenAI Service と Amazon Bedrock に焦点を当て、各サービスのリージョン別の提供状況を整理しています。

## 生成AIにおけるリージョンの背景と制約

生成AIを活用するクラウドサービスでは、リージョンによって利用可能なモデルや機能、料金体系が異なります。
これは、各国の法規制、インフラの整備状況、データ主権などが影響しているためです。

### Azure OpenAI Service におけるリージョン

Azure OpenAI Service で利用可能なリージョンは以下の通りです：

| モデル | グローバル標準対応 | 主な利用可能リージョン |
|---|---|---|
| gpt-4o-mini | Yes | East US, Sweden Central など |
| gpt-4o | Yes | East US, Sweden Central, South Central US など |
| gpt-4.1-mini | Yes | East US, Sweden Central など |
| gpt-4.1 | Yes | East US, Sweden Central, South Central US など |
| gpt-5-mini | Yes | East US, Sweden Central, South Central US など |
| gpt-5 | Yes | East US, Sweden Central, South Central US など |
| gpt-5.1 | Yes | East US, Sweden Central など |
| gpt-5.2 | Yes | East US, Sweden Central など |
| text-embedding-ada-002 | No | Japan East, East US, West Europe, Australia East など |
| text-embedding-3-small | No | Japan East, East US, West Europe, Australia East など |
| dall-e-3 | No | East US など |
| whisper | No | East US など |

コラム

「グローバル標準対応」とは、 Azure OpenAI Service においてモデルが特定のリージョンに固定されず、Azure のグローバルインフラを通じて最適なリージョンにルーティングされるデプロイ方式を指します。
これにより、より高い可用性とスループットが期待できますが、推論処理がどのリージョンで行われるかは固定されません。
https://learn.microsoft.com/ja-jp/azure/ai-foundry/foundry-models/concepts/deployment-types

注意

モデルの利用可能リージョンは、変更される場合があります。最新の情報は公式ドキュメントを参考にしてください。
https://learn.microsoft.com/azure/ai-services/openai/concepts/models

### Amazon Bedrock におけるリージョン

Amazon Bedrock で利用可能なリージョンは以下の通りです：

| モデル | 主な利用可能リージョン |
|---|---|
| anthropic.claude-3-haiku-20240307-v1:0 | ap-northeast-1, us-east-1, us-west-2 など |
| anthropic.claude-3-5-haiku-20241022-v1:0 | us-east-1, us-west-2 など |
| anthropic.claude-haiku-4-5-20251001-v1:0 | ap-northeast-1, us-east-1, us-west-2 など |
| anthropic.claude-3-sonnet-20240229-v1:0 | ap-northeast-1, us-east-1, us-west-2 など |
| anthropic.claude-3-5-sonnet-20240620-v1:0 | ap-northeast-1, us-east-1, us-west-2 など |

| モデル | 主な利用可能リージョン |
|---|---|
| anthropic.claude-3-7-sonnet-20250219-v1:0 | ap-northeast-1, us-east-1, us-west-2 など |
| anthropic.claude-sonnet-4-20250514-v1:0 | ap-northeast-1, us-east-1, us-west-2 など |
| anthropic.claude-sonnet-4-5-20250929-v1:0 | ap-northeast-1, us-east-1, us-west-2 など |
| anthropic.claude-sonnet-4-6 | ap-northeast-1, us-east-1, us-west-2 など |
| anthropic.claude-opus-4-6-v1 | ap-northeast-1, us-east-1, us-west-2 など |
| amazon.nova-pro-v1:0 | ap-northeast-1, us-east-1, eu-west-1 など |
| amazon.nova-lite-v1:0 | ap-northeast-1, us-east-1, eu-west-1 など |
| amazon.titan-embed-text-v1 | ap-northeast-1, us-east-1, us-west-2 など |
| amazon.nova-canvas-v1:0 | ap-northeast-1, us-east-1, eu-west-1 など |

注意

モデルの利用可能リージョンは、変更される場合があります。最新の情報は公式ドキュメントを参考にしてください。
https://docs.aws.amazon.com/bedrock/latest/userguide/models-supported.html

注意

一部のモデルはクロスリージョン推論プロファイル経由でのみ利用可能です。シングルリージョンでの直接呼び出しに対応しているかどうかは、以下の公式ドキュメントを確認してください。
https://docs.aws.amazon.com/bedrock/latest/userguide/inference-profiles-support.html

クロスリージョン推論プロファイルの利用

Amazon Bedrock の一部のモデルは、クロスリージョン推論プロファイル経由でのみ利用可能です。
これらのモデルを呼び出す際は、モデルIDに地域プレフィックスを付与する必要があります。

クロスリージョン推論プロファイルには以下の2種類があります。

グローバルクロスリージョン推論（global.）

AWSが複数の商用リージョンにまたがってリクエストをルーティングします。
データの処理リージョンは固定されません。

地理的クロスリージョン推論（jp. / apac. / us. / eu. 等）

特定の地理的エリア内でのみルーティングします。
データの処理リージョンを一定の地理的範囲に限定できます。

例えば、東京リージョン（ap-northeast-1）から利用する場合、モデルIDは以下のように指定します：

| 種類 | モデルID指定例 |
|---|---|
| グローバル | global.anthropic.claude-haiku-4-5-20251001-v1:0 |
| 地理的（日本） | jp.anthropic.claude-haiku-4-5-20251001-v1:0 |
| 地理的（APAC） | apac.amazon.nova-pro-v1:0 |

注意

データの処理リージョンを特定の地理的範囲に限定する必要がある場合（データ主権・法規制等）は、地理的クロスリージョン推論プロファイルをご利用ください。

地域プレフィックスの形式や対応状況は今後変更される場合があります。
モデルIDの指定ミスは推論リクエストの失敗につながるため、以下の公式ドキュメントをご確認ください。
https://docs.aws.amazon.com/bedrock/latest/userguide/inference-profiles-support.html

## 関連情報

このページでは、モデル運用に関連する追加情報や、アップデート時の確認事項、トラブルシューティングをまとめています。

目次



## モデルの有効化

一部の生成AIサービスやモデルは、利用を開始する前に明示的な「有効化」手続きが必要です。
以下のようなケースに該当する場合は、モデルの有効化を行ってください。

- Azure OpenAI Service を利用する場合：新しいモデルを利用する際にデプロイが必要です。
- Amazon Bedrock を利用する場合：Anthropic（Claude）モデルの初回利用時のみフォーム送信が必要です。それ以外のモデルは追加手続き不要です。

### Azure OpenAI Service モデル有効化手順

1. Azure OpenAI リソースの作成（未作成の場合）

    1. Azure ポータルにログイン
    2. 「リソースの作成」→「Azure OpenAI」を選択
    3. 必要事項（サブスクリプション、リソースグループ、リージョンなど）を入力
    4. 「確認および作成」でリソースを作成

2. モデルの有効化（デプロイ）

    1. 作成した Azure OpenAI リソースにアクセス
    2. 左メニューから「モデルのデプロイ」を選択
    3. 「+ 新しいデプロイ」をクリック
    4. 以下を設定：
        ・ モデル名
        ・ デプロイ名（任意の識別名）
        ・ モデルのバージョン（選択可能な場合）
    5. 「作成」をクリックしてデプロイを完了

> **注意**
>
> Azure ポータルの UI は更新される可能性があります。
> 表示されるメニュー名やボタンの位置が変更されている場合は、「モデルのデプロイ」や「OpenAI」などのキーワードで検索するか、公式ドキュメントを参照してください。
> モデルの利用には、Microsoft による事前承認が必要な場合があります。

### Amazon Bedrock モデル有効化手順

Amazon Bedrock のモデルは、すべての商用 AWS リージョンでデフォルトで有効化されています。そのため、ほとんどのモデルは追加の有効化手続きなしに利用を開始できます。
ただし、Anthropic（Claude）モデル を利用する場合は、初回利用時に1回限りの使用フォームの送信が必要です。

1. AWS マネジメントコンソールにログインします。
   https://aws.amazon.com/jp/console/

2. 「すべてのサービス」より「Amazon Bedrock」を検索し、Amazon Bedrock サービスにアクセスします。

3. プレイグラウンド等から Anthropic モデルを選択し、表示されるフォームに必要事項を入力して送信してください。

> コラム
> - AWS Organizations の管理アカウントからフォームを送信すると、組織内の全メンバーアカウントで自動的に有効化されます。
> - Amazon Nova モデルは有効化手続き不要で即時利用可能です。

## モデルの変更

以下のような場合には、現在利用中のモデルを別のモデルに変更する必要があります。

- モデル変更が必要となる主なケース
  - サポート対象モデルの廃止
  - 品質やコストの最適化を目的としたモデルの再選定
  - 機能ごとに推奨されるモデルの更新

### 必須アクション（廃止モデルの場合）

廃止されたモデルを使用している場合は、代替モデルへの切り替えが必須です。以下の対応を行ってください。

- 代替モデルに切り替える
  - 設定ファイル（IM-Copilot生成AI連携ドライバ設定）のデフォルトパラメータ・個別パラメータを変更
    - 参考情報：*機能別モデル設定パラメータ名一覧*
  - 各機能の入力パラメータを変更（明示的にモデルを指定している場合）
    - IM-LogicDesignerタスク
    - JavaEE開発モデル、スクリプト開発モデル向けの API
- モデル変更後は、動作確認や品質評価を実施する

### 推奨アクション（再選定・推奨モデル更新の場合）

モデルの品質やコスト、機能面での最適化を目的とした変更の場合は、以下の対応を検討してください。

- 利用目的に応じたモデルの再選定を行う
- 推奨モデルに切り替える
  - 設定ファイル（IM-Copilot生成AI連携ドライバ設定）のデフォルトパラメータ・個別パラメータを変更
    - 参考情報：*機能別モデル設定パラメータ名一覧*
  - 各機能の入力パラメータを変更（明示的にモデルを指定している場合）
    - IM-LogicDesignerタスク
    - JavaEE開発モデル、スクリプト開発モデル向けの API
- モデル変更後は、動作確認や品質評価を実施する

### 機能別モデル設定パラメータ名一覧

設定ファイル（IM-Copilot生成AI連携ドライバ設定）により、デフォルトパラメータおよび個別パラメータの変更が可能です。
以下は、各機能に対応するモデルを設定するためのデフォルトパラメータ・個別パラメータ、および対応バージョンの一覧です。

OpenAI を使用する場合

| 機能 | デフォルト | 個別（機能専用、優先） | バージョン（以降利用可能） |
|---|---|---|---|
| チャット アクション | default-chat-model | | 2024 Autumn(Jasmine) |
| 埋め込み アクション | default-embeddings-model | | 2024 Autumn(Jasmine) |
| 画像生成 アクション | default-images-model | | 2024 Autumn(Jasmine) |
| 音声生成 アクション | default-speech-model | | 2024 Autumn(Jasmine) |
| 文字起こし アクション | default-transcription-model | | 2024 Autumn(Jasmine) |

| 機能 | デフォルト | 個別（機能専用、優先） | バージョン（以降利用可能） |
|---|---|---|---|
| Wiki アシスタント チャット | default-chat-model | im-wiki-copilot-query-generation-chat-model<br>im-wiki-copilot-response-generation-chat-model | 2024 Autumn(Jasmine)<br>個別：2025 Autumn(Lilac) |
| Wiki アシスタント 埋め込み | default-embeddings-model | im-wiki-copilot-embeddings-model | 2024 Autumn(Jasmine)<br>個別：2025 Autumn(Lilac) |
| ViewCreator SQLビルダ アシスタント チャット | default-chat-model | viewcreator-copilot-chat-model | 2024 Autumn(Jasmine)<br>個別：2026 Spring(Mimosa) |
| ViewCreator SQLビルダ アシスタント 埋め込み | default-embeddings-model | viewcreator-copilot-embeddings-model | 2024 Autumn(Jasmine)<br>個別：2026 Spring(Mimosa) |
| Accel Studio アプリケーション作成 アシスタント チャット | default-chat-model | accel-studio-copilot-chat-model | 2024 Autumn(Jasmine) |
| Accel Studio テスト機能 アシスタント チャット | default-chat-model | accel-studio-testing-copilot-chat-model | 2025 Autumn(Lilac) |

Azure OpenAI Service を使用する場合

| 機能 | デフォルト | 個別（機能専用、優先） | バージョン（以降利用可能） |
|---|---|---|---|
| チャット アクション | default-chat-deployment-id | | 2024 Autumn(Jasmine) |
| 埋め込み アクション | default-embeddings-deployment-id | | 2024 Autumn(Jasmine) |
| 画像生成 アクション | default-images-deployment-id | | 2024 Autumn(Jasmine) |
| 文字起こし アクション | default-transcription-deployment-id | | 2024 Autumn(Jasmine) |
| Wiki アシスタント チャット | default-chat-deployment-id | im-wiki-copilot-query-generation-chat-deployment-id<br>im-wiki-copilot-response-generation-chat-deployment-id | 2024 Autumn(Jasmine)<br>個別：2025 Autumn(Lilac) |
| Wiki アシスタント 埋め込み | default-embeddings-deployment-id | im-wiki-copilot-embeddings-deployment-id | 2024 Autumn(Jasmine)<br>個別：2025 Autumn(Lilac) |
| ViewCreator SQLビルダ アシスタント チャット | default-chat-deployment-id | viewcreator-copilot-chat-deployment-id | 2024 Autumn(Jasmine)<br>個別：2025 Autumn(Lilac) |
| ViewCreator SQLビルダ アシスタント 埋め込み | default-embeddings-deployment-id | viewcreator-copilot-embeddings-deployment-id | 2024 Autumn(Jasmine)<br>個別：2025 Autumn(Lilac) |
| Accel Studio アプリケーション作成 アシスタント チャット | default-chat-deployment-id | accel-studio-copilot-deployment-id | 2024 Autumn(Jasmine) |
| Accel Studio テスト機能 アシスタント チャット | default-chat-deployment-id | accel-studio-testing-copilot-deployment-id | 2025 Autumn(Lilac) |

Amazon Bedrock を使用する場合

| 機能 | デフォルト | 個別（機能専用、優先） | バージョン（以降利用可能） |
|---|---|---|---|
| チャット アクション | default-chat-model | | 2024 Autumn(Jasmine) |

| 機能 | デフォルト | 個別（機能専用、優先） | バージョン（以降利用可能） |
|---|---|---|---|
| 埋め込み アクション | default-embeddings-model | | 2024 Autumn(Jasmine) |
| 画像生成 アクション | default-images-model | | 2024 Autumn(Jasmine) |
| Wiki アシスタント チャット | default-chat-model | im-wiki-copilot-query-generation-chat-model<br>im-wiki-copilot-response-generation-chat-model | 2024 Autumn(Jasmine)<br>個別：2025 Autumn(Lilac) |
| Wiki アシスタント 埋め込み | default-embeddings-model | im-wiki-copilot-embeddings-model | 2024 Autumn(Jasmine)<br>個別：2025 Autumn(Lilac) |
| ViewCreator SQLビルダ アシスタント チャット | default-chat-model | viewcreator-copilot-chat-model | 2024 Autumn(Jasmine)<br>個別：2026 Spring(Mimosa) |
| ViewCreator SQLビルダ アシスタント 埋め込み | default-embeddings-model | viewcreator-copilot-embeddings-model | 2024 Autumn(Jasmine)<br>個別：2026 Spring(Mimosa) |
| Accel Studio アプリケーション作成 アシスタント チャット | default-chat-model | accel-studio-copilot-chat-model | 2024 Autumn(Jasmine) |
| Accel Studio テスト機能 アシスタント チャット | default-chat-model | accel-studio-testing-copilot-chat-model | 2025 Autumn(Lilac) |

## アップデート時の確認事項

アップデートにより、以下のような変更が発生する可能性があります。

- デフォルトモデルの変更
- サポートモデルの追加・削除
- モデルの挙動や性能の変化（例：デフォルトモデルの変更）
- 各機能の入力・出力内容の変更

### 確認ポイント

アップデートに伴い、以下の点について事前に確認しておくことを推奨します。

- アップデートにより、利用中の機能に影響があるか
- アップデートにより、モデルの再有効化が必要か

> **コラム**
>
> Amazon Bedrock をご利用で、アップデートにより新たに利用するモデルに Anthropicモデルが含まれる場合は、以下をご確認ください。
>
> - 初回利用フォームが未送信の場合は送信が必要です。詳細は「*モデルの有効化*」を参照してください。
> - AWSパートナー（AWSソリューションプロバイダー / AWSディストリビューター）経由でご契約の場合は、Anthropicモデルの利用可否をご契約先へご確認ください。
>
>   詳細は「*Amazon Bedrockのセットアップ*」の前提条件を参照してください。

- アップデートにより、新たに利用するモデルにグローバルクロスリージョン推論が使用されているか

> **コラム**
>
> Amazon Bedrock をご利用で、アップデートにより新たに利用するモデルにグローバルクロスリージョン推論が使用されている場合、データの処理リージョンが固定されません。
> データの処理リージョンを特定の地理的範囲に限定する必要がある場合は地理的クロスリージョン推論プロファイルへの変更をご検討ください。
>
> 対象の機能・モデルおよび変更方法については「*Amazon Bedrock を使用する場合*」および「*クロスリージョン推論プロファイルの利用*」を参照してください。

- アップデートにより、設定ファイル（IM-Copilot生成AI連携ドライバ設定）の変更があるか
- アップデートにより、各機能の入力パラメータに変更があるか

### 推奨アクション

影響を最小限に抑えるため、以下の対応を検討してください。

- アップデートの準備
  - アップデートに備えて、モデル変更観点の手順書を整備する。（例：モデル設定・有効化の手順など）
- 影響範囲の特定
  - 「*モデル更新履歴*」や「*サポートモデル一覧*」を確認し、変更点を把握する。
  - 使用中のモデルや機能との関連性を洗い出す。
- 設定・各機能の入力パラメータの見直し
  - 設定ファイル（IM-Copilot生成AI連携ドライバ設定）を確認、必要に応じて更新する。
  - IM-LogicDesignerタスクやAPIの入力パラメータを確認、必要に応じて更新する。（例：モデルID）
- テスト環境での検証、適用判断
  - 本番環境に適用する前に、ステージング環境で動作確認を行う。
  - 特にモデルの挙動や性能に関する観点で検証する。
  - 検証結果を基に本番環境においてモデル変更を実施する。

## トラブルシューティング

生成AIの利用に際して、問題が発生した場合は、以下のような対応を行うことで原因の特定と解決が可能です。

- 生成AIに接続できない場合
  - ネットワークやProxy設定で制限されていないか確認する
  - APIキーや認証情報が正しいか確認する
  - 接続先のエンドポイントURLが正しいか確認する
  - サービス側の障害情報（ステータスページなど）を確認する
  - 接続時のログ内容を確認し、エラーコードやメッセージをもとに原因を特定する
- モデルが利用できない場合
  - モデル名が正しいか確認する
  - モデルが有効化されているか確認する
  - 利用中のリージョンやプロファイルに制限がないか確認する
    - 例：Amazon Bedrock の場合は、推論プロファイルが正しく設定されているかを確認する
- モデルの応答が不安定・遅延する場合
  - 他のサポートモデルで再検証する
  - 入力形式やパラメータを見直す
  - 処理が同一リソース上で集中している場合、処理の分離やスケジューリングの工夫を検討する
    - 例：RAGなどの埋め込みジョブを深夜帯に実行するなど
- 意図しない応答・精度が低下した場合
  - 各生成AIサービスの仕様変更、アップデートによる挙動変化を確認する
  - 過去の応答との比較検証を行う
  - 入力プロンプトや前処理を見直す

# セットアップ（ 生成AI ）

IM-Copilotで利用可能な生成AIサービスのセットアップ手順を説明します。

## OpenAIのセットアップ

IM-Copilot を利用するための OpenAI のセットアップ方法について説明します。

項目



### 前提条件

当セットアップ手順は2024年3月末時点の OpenAI 公開情報をもとに、セットアップの主な流れを記載しています。
詳細な手順については OpenAI 側のドキュメント（ https://platform.openai.com/docs ）、および、最新情報も参照してください。

### セットアップ手順

#### OpenAI アカウント作成

1. https://platform.openai.com/docs にアクセスしてください。
2. 右上の「Sign Up」をクリックしてください。
3. 以降は表示された画面内容に従ってアカウントを作成してください。
4. アカウント作成に成功後、ログインしてください。

#### APIキー作成

1. https://platform.openai.com/api-keys にアクセスしてください。
2. 以降は表示された画面内容に従って「APIキー」を作成してください。
3. 作成した「APIキー」をメモ帳などに控えてください。後の手順で利用します。

#### Organization ID確認

1. https://platform.openai.com/account/organization にアクセスしてください。
2. 表示されている「Organization ID」をメモ帳などに控えてください。後の手順で利用します。

#### 課金設定（任意）

1. https://platform.openai.com/account/billing/overview にアクセスしてください。
2. 以降は表示された画面内容に従って課金設定を行ってください。

#### 使用上限設定（任意）

1. https://platform.openai.com/usage にアクセスしてください。
2. 以降は表示された画面内容に従って使用上限設定を行ってください。

## Azure OpenAI Serviceのセットアップ

IM-Copilot を利用するための Azure OpenAI Service のセットアップ方法について説明します。



## 前提条件

当セットアップ手順は2024年3月末時点の Azure OpenAI Service 公開情報をもとに、セットアップの主な流れを記載しています。
詳細な手順については Azure OpenAI Service 側のドキュメント（https://learn.microsoft.com/azure/ai-services/openai ）、および、最新情報も参照してください。

## セットアップ手順

### Azure OpenAI Service アカウント作成

1. https://azure.microsoft.com にアクセスしてください。
2. 「無料アカウント」をクリックしてください。
3. 「無料で始める」をクリックしてください。
4. 以降は表示された画面内容に従ってアカウントを作成してください。
5. アカウント作成に成功後、ログインしてください。

### リソースグループ作成

1. Azureポータル（ https://portal.azure.com/#home ） にアクセスしてください。
2. サイドメニューから「リソースグループ」を選択してください。
3. 「作成」をクリックしてください。
4. 以降は表示された画面内容に従ってリソースグループを作成してください。

### リソース作成

1. Azureポータル（ https://portal.azure.com/#home ） にアクセスしてください。
2. 「リソースの作成」をクリックしてください。
3. 検索欄に「OpenAI」を入力してください。
4. 検索結果の Azure OpenAI の「作成」をクリックしてください。
5. 以降は表示された画面内容に従ってリソースを作成してください。
6. 作成したリソースの名前（インスタンスの名前）をメモ帳などに控えてください。

> **注意**
>
> 作成中に以下メッセージが表示された場合、審査申請を行ってください。
> 「Azure OpenAIサービスへのアクセスを要求するには、ここをクリックしてください。」
>
> 審査完了後にリソース作成を再開してください。

### モデルデプロイ

1. Azureポータル（ https://portal.azure.com/#home ） にアクセスしてください。
2. 先ほど作成したリソースを選択してください。
3. サイドメニューから「モデル デプロイ」を選択してください。
4. 以降は利用想定の機能に応じて、 *サポートモデル* を参考に、表示された画面内容に従ってモデルをデプロイしてください。

> **コラム**
>
> リソースのリージョンによって選択できないベースモデルがある場合があります。

### APIキー作成

1. Azureポータル（ https://portal.azure.com/#home ） にアクセスしてください。
2. 先ほど作成したリソースを選択してください。
3. 「キーを管理するにはここをクリック」をクリックしてください。
4. 表示されている「APIキー」と「エンドポイント」をメモ帳などに控えてください。後の手順で利用します。

### 課金設定（任意）

1. Azureポータル（ https://portal.azure.com/#home ） にアクセスしてください。
2. 「コストの管理と請求」をクリックしてください。
3. 以降は表示された画面内容に従って課金設定を行ってください。

## Amazon Bedrockのセットアップ

IM-Copilot を利用するための Amazon Bedrock のセットアップ方法について説明します。

項目



### 前提条件

当セットアップ手順は2025年10月時点の Amazon Bedrock 公開情報をもとに、セットアップの主な流れを記載しています。
詳細な手順については Amazon Bedrock 側のドキュメント（ https://docs.aws.amazon.com/bedrock/latest/userguide/what-is-bedrock.html ）、および、最新情報も参照してください。

コラム

Amazon Bedrock をご利用の場合、弊社が標準で指定しているデフォルトモデルの一部はグローバルクロスリージョン推論で動作します。
データの処理リージョンを特定の地理的範囲に限定する必要がある場合は、地理的クロスリージョン推論プロファイルへの変更が必要です。

対象の機能・モデルおよび変更方法については「*Amazon Bedrock を使用する場合*」および「*クロスリージョン推論プロファイルの利用*」を参照してください。

注意

Amazon Bedrockを経由したAnthropic（Claude）モデルの利用に関して

Amazon BedrockはAWSパートナーによるリセール対象サービスですが、Anthropicモデルについてはリセール（再販）に制限が設けられています。

リセラー（AWSソリューションプロバイダー / AWSディストリビューター）経由で契約しているAWSアカウントでは、Anthropic社のリセール認証を受けていないパートナー経由の場合、Anthropicモデルを利用できない場合があります。

リセラー経由で発行されたAWSアカウントでAnthropicモデルを呼び出すと、以下のエラーが発生する場合があります。

```
ValidationException: Access to this model is not available for channel program accounts.
Reach out to your AWS Solution Provider or AWS Distributor for more information.
```

Anthropicモデルのリセール認証を取得済みかどうかは、ご契約先のリセラーへお問い合わせください。
AWS直接契約のアカウントを利用されている場合は、この制限は適用されません。

## セットアップ手順

### モデルアクセス設定

Amazon Bedrock のモデルは、すべての商用 AWS リージョンでデフォルトで有効化されています。 Anthropic（Claude）モデル を利用する場合は、初回利用時に1回限りの使用フォームの送信が必要です。

1. AWS マネジメントコンソールにログインします。
   https://aws.amazon.com/jp/console/
2. 「すべてのサービス」より「Amazon Bedrock」を検索し、Amazon Bedrock サービスにアクセスします。
3. プレイグラウンド等から Anthropic モデルを選択し、表示されるフォームに必要事項を入力して送信してください。

   コラム
   - AWS Organizations の管理アカウントからフォームを送信すると、組織内の全メンバーアカウントで自動的に有効化されます。

### ポリシーの作成

1. AWS マネジメントコンソールにログインします。
   https://aws.amazon.com/jp/console/
2. 「すべてのサービス」より「IAM」を検索し、IAMサービスにアクセスします。
3. メニューより「ポリシー」ページを開き、「ポリシーの作成」をクリックします。
4. 「サービスを選択」にて「Bedrock」を検索し、選択してください。
5. 「すべての Bedrock アクション (bedrock:*)」チェックボックスをオンにします。
6. 「リソース」の「すべて」を選択し、「次へ」をクリックします。
7. 「ポリシー名」に任意のポリシー名を入力し、「ポリシーの作成」をクリックしてください。
8. ポリシーが作成されました。
   作成されたポリシー名は *アクセスキーとシークレットキーの作成* や *ロールの作成* に使用しますので、控えておいてください。

### 認証方式の選択

IM-Copilot から Amazon Bedrock サービスへ接続する際の認証方式として、以下の3通りが利用可能です。
以下、ご利用の認証方式に合わせてセットアップしてください。

- *認証情報ファイルを利用して認証する場合*
- *設定ファイルに直接記述したアクセスキーとシークレットキーを利用して認証する場合*
- *インスタンスプロファイルを利用して認証する場合*（※Amazon EC2 インスタンス上で iAP が動作している場合のみ）

> **コラム**
>
> 各認証方式については、AWSのドキュメントも参考にしてください。
>
> - 設定ファイルと認証情報ファイルの設定
>
> https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html
>
> - IAM ユーザーのアクセスキーの管理
>
> https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html
>
> - インスタンスプロファイルの使用
>
> https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2_instance-profiles.html

#### 認証情報ファイルを利用して認証する場合

ユーザーの作成

1. IAMサービスにアクセスします。
2. メニューより「ユーザー」ページを開き、「ユーザーの作成」をクリックします。
3. 「ユーザー名」に任意のユーザー名を入力し、「次へ」をクリックしてください。
4. 「許可のオプション」にて「ポリシーを直接アタッチする」を選択してください。
   その後、*ポリシーの作成* で作成したポリシー名にチェックし、「次へ」をクリックしてください。
5. 「ユーザーの作成」をクリックしてください。
6. ユーザーが作成されました。

アクセスキーとシークレットキーの作成

1. IAMサービスにアクセスします。
2. メニューより「ユーザー」ページを開き、*ユーザーの作成* で作成したユーザー名をクリックしてください。
3. 「セキュリティ認証情報」タブをクリックし、「アクセスキー」にて「アクセスキーを作成」をクリックしてください。
4. 「コマンドラインインターフェイス (CLI)」を選択し、「上記のレコメンデーションを理解し、アクセスキーを作成します。」にチェックを入れ、「次へ」をクリックしてください。
5. 「アクセスキーを作成」をクリックしてください。
6. 「アクセスキーを取得」画面でアクセスキーとシークレットアクセスキーを控えるか、「.csv ファイルをダウンロード」をクリックしてください。その後、「完了」をクリックしてください。
7. アクセスキーとシークレットキーは、認証情報ファイルを利用して認証する場合には、*認証情報ファイルの作成* で使用します。
   設定ファイルに直接記述したアクセスキーとシークレットキーを利用して認証する場合には、*生成AI連携ドライバ設定* で使用します。

認証情報ファイルの作成

認証情報ファイルを利用して認証する場合、 iAP が動作している環境のファイルシステム内に、以下のような方法を用いて認証情報ファイル（credentials）を作成する必要があります。

- AWS コマンドラインインターフェイス (AWS CLI) を利用して作成
- 直接ファイルを用意し、内容を記述して作成

これらの詳細な手順については、AWSが提供するドキュメントを参照してください。
この過程において、*アクセスキーとシークレットキーの作成* で作成したアクセスキーとシークレットキーが要求されます。
また、指定したプロファイル名は、*生成AI連携ドライバ設定* で使用します。

### 設定ファイルに直接記述したアクセスキーとシークレットキーを利用して認証する場合

ユーザーの作成

1. IAMサービスにアクセスします。
2. メニューより「ユーザー」ページを開き、「ユーザーの作成」をクリックします。
3. 「ユーザー名」に任意のユーザー名を入力し、「次へ」をクリックしてください。
4. 「許可のオプション」にて「ポリシーを直接アタッチする」を選択してください。
   その後、 *ポリシーの作成* で作成したポリシー名にチェックし、「次へ」をクリックしてください。
5. 「ユーザーの作成」をクリックしてください。
6. ユーザーが作成されました。

アクセスキーとシークレットキーの作成

1. IAMサービスにアクセスします。
2. メニューより「ユーザー」ページを開き、*ユーザーの作成* で作成したユーザー名をクリックしてください。
3. 「セキュリティ認証情報」タブをクリックし、「アクセスキー」にて「アクセスキーを作成」をクリックしてください。
4. 「コマンドラインインターフェイス (CLI)」を選択し、「上記のレコメンデーションを理解し、アクセスキーを作成します。」にチェックを入れ、「次へ」をクリックしてください。
5. 「アクセスキーを作成」をクリックしてください。
6. 「アクセスキーを取得」画面でアクセスキーとシークレットアクセスキーを控えるか、「.csv ファイルをダウンロード」をクリックしてください。その後、「完了」をクリックしてください。
7. アクセスキーとシークレットキーは、認証情報ファイルを利用して認証する場合には、*認証情報ファイルの作成* で使用します。
   設定ファイルに直接記述したアクセスキーとシークレットキーを利用して認証する場合には、*生成AI連携ドライバ設定* で使用します。

### インスタンスプロファイルを利用して認証する場合

EC2インスタンスにアタッチされているロールの確認

この作業は、利用している iAP が Amazon EC2 インスタンス上で動作していることを前提としています。

1. AWS マネジメントコンソールにログインします。
   https://aws.amazon.com/jp/console/
2. サービスより「EC2」を検索し、EC2サービスにアクセスします。
3. メニューより「インスタンス」ページを開き、利用しているインスタンスをクリックします。
4. 「セキュリティ」タブをクリックし、「IAM ロール」に表示されているIAMロールをクリックします。
   もし「IAM ロール」にロールが表示されていない場合は、「アクション」＞「セキュリティ」＞「IAM ロールの変更」より、EC2インスタンスの利用用途に応じたIAMロールを割り当ててください。
5. IAMロールの詳細情報が表示されます。
   表示されている「ARN」の文字列を控えておいてください。

ロールの作成

この作業は、利用している iAP が Amazon EC2 インスタンス上で動作していることを前提としています。

1. IAMサービスにアクセスします。
2. メニューより「ロール」ページを開き、「ロールを作成」をクリックしてください。
3. 「信頼されたエンティティタイプ」にて「カスタム信頼ポリシー」を選択し、「カスタム信頼ポリシー」に以下を入力してください。
   `${EC2インスタンスにアタッチされたロールのARN}` の箇所は、*EC2インスタンスにアタッチされているロールの確認* で確認した「ARN」の文字列としてください。
   その後、「次へ」を押下してください。

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
        "Effect": "Allow",
        "Principal": {
          "AWS": "${EC2インスタンスにアタッチされたロールのARN}"
        },
        "Action": "sts:AssumeRole"
    }
  ]
}
```

4. 「許可ポリシー」にて、 *ポリシーの作成* で作成したポリシー名にチェックを入れ、「次へ」をクリックしてください。
5. 「ロール名」に任意のロール名を入力し、「ロールを作成」をクリックしてください。
6. ロールが作成されました。
   作成されたロールのARNは、 *生成AI連携ドライバ設定* で使用しますので、控えておいてください。

### ガードレールの作成（任意）

Amazon Bedrock Guardrails は、AI モデルへの入出力を監視し、不適切なコンテンツから保護する機能です。
有害なコンテンツや機密情報などを自動的に検出・ブロック・マスキングすることで、安全に生成AIを利用できます。
ガードレール機能を使用する場合は、以下の手順でガードレールを作成してください。

> **コラム**
>
> ガードレール機能の詳細については、 Amazon Bedrock のドキュメントを参照してください。
>
> Amazon Bedrock Guardrails
> https://docs.aws.amazon.com/bedrock/latest/userguide/guardrails.html

> **注意**
>
> **日本語対応について**
>
> 日本語コンテンツに対して Amazon Bedrock Guardrails を使用する場合は、以下の点に注意してください。
>
> - 日本語対応には **Standard Tier** の利用が必要です。 （従来の Classic Tier では日本語はサポートされていません）
> - Standard Tier では、ガードレール評価に **クロスリージョン推論** が使用されます。 クロスリージョン推論では、AWS が同一地理エリア内の複数リージョンに処理を分散します。
> - 日本語対応はガードレールの機能ごとに異なります。
>   例：
>   - コンテンツフィルタ、拒否トピック：日本語対応（Standard Tier）
>   - 単語フィルタ、コンテキストグラウンディングチェック：日本語での評価が制限される、または十分に機能しない場合がある
>
>   実際の対応状況は利用する機能や設定、実行環境により異なる場合があります。 Guardrails の Tier や言語サポートの詳細については、 Amazon Bedrock のドキュメントを参照してください。

1. AWS マネジメントコンソールにログインします。
   https://aws.amazon.com/jp/console/
2. サービスより「Amazon Bedrock」を検索し、Amazon Bedrock サービスにアクセスします。
3. メニューより「ガードレール」ページを開きます。
4. 「ガードレールを作成」をクリックします。
5. 「名前」に任意のガードレール名を入力してください。
6. 利用シナリオに応じて、ステップに従って必要なガードレールの設定を行ってください。
   主なポリシーとして、以下のような設定が可能です。
   - コンテンツフィルタ：有害または不適切なコンテンツを検出・フィルタリング
   - 機密情報フィルタ：個人情報（PII）などの機密情報を検出・ブロック／マスキング

- 拒否トピック（Denied topics）：特定のトピックに関する入力・出力を制限
- 単語フィルタ：特定の単語やフレーズをブロック

7. 「ガードレールを作成」をクリックしてください。

8. ガードレールが作成されました。
   作成されたガードレールの「ガードレール ID」は *生成AI連携ドライバ設定* で使用するため、控えておいてください。
   作成時点のガードレールのバージョンは「 DRAFT 」です。

> **コラム**
>
> ガードレールのバージョン管理について
>
> 作成直後の DRAFT バージョン で動作確認を行った後、本番環境では番号付きのバージョンを作成・指定して使用することを推奨します。

### ガードレールのテストと調整

ガードレールを作成した後は、実際の利用シナリオを想定して動作確認および調整を行うことを推奨します。
AWS マネジメントコンソールのガードレールテスト機能を使用することで、入力および出力が想定どおりにブロックまたは許可されるかを確認できます。

> **コラム**
>
> ガードレールのテスト機能の詳細については、 Amazon Bedrock のドキュメントを参照してください。
>
> Test your guardrail
> https://docs.aws.amazon.com/bedrock/latest/userguide/guardrails-test.html

特に以下の点を重点的に確認してください。

- 意図しない正常な入力・出力がブロックされていないか
- 各フィルタの **filter strength** （フィルタ強度）が適切に設定されているか
  フィルタ強度が高すぎる場合、正常なコンテンツがブロックされる可能性があります。
  一方、低すぎる場合は不適切なコンテンツを検出できない可能性があります。
  実際の使用条件に基づき、適切なバランスとなるよう調整してください。

### ガードレール違反時の応答メッセージのカスタマイズ

本節では、開発者向け機能（Java ChatAction API、JavaScript ChatAction API）を利用する場合に、ガードレール違反時の応答メッセージを多言語対応する方法について説明します。
開発者向け機能を利用しない場合は、本節の手順は不要です。
各 APIの利用方法の詳細は「*開発者向けガイド*」を参照してください。

IM-Copilot の標準機能（アシスタントの実行画面）では、ガードレール違反時に製品で用意した固定メッセージが表示されます。
一方、開発者向け機能を使用してチャットを実行する場合、ガードレールの違反種別により以下の挙動が適用されます。

- Java ChatAction API 非ストリーミング形式 または JavaScript ChatAction API
  入力違反時：エラーが発生します（本カスタマイズの対象外）
  出力違反時：ガードレール作成時に設定したメッセージ（多言語未対応）がアシスタントの回答として返されます
- Java ChatAction API ストリーミング形式
  入力違反、出力違反ともにガードレール作成時に設定したメッセージ（多言語未対応）がアシスタントの回答として返されます

出力違反時のメッセージをメッセージキーとして定義することで、利用者の言語設定に応じたメッセージを取得できます。

多言語対応が必要な場合は、以下の方法でメッセージをカスタマイズしてください。

1. ガードレール作成時に指定するメッセージとして、プレフィックス「`MSG.I.IWP.COPILOT`」で始まるメッセージキーを設定します。
   例：`MSG.I.IWP.COPILOT.GUARDRAILS.CONTENT.FILTER`
   本機能で利用できるデフォルトのメッセージキーをあらかじめ用意しています。

   ```
   MSG.I.IWP.COPILOT.GUARDRAILS.CONTENT.FILTER=安全性に関するポリシーにより回答できません。
   ```

2. 独自のメッセージキーを追加する場合は対応するメッセージキーをメッセージプロパティファイルに定義します。

この設定により、開発者向け機能でのガードレール出力違反時に利用者の言語設定に応じたメッセージが返されます。

コラム

IM-Copilotで利用できる生成AIサービスは、今後の製品アップデートに合わせて追加を予定しています。

- Amazon Bedrockは、 IM-Copilot Amazon Bedrockドライバモジュール を導入している場合のみ利用可能です。

# セットアップ（ iAP ）

IM-Copilot を利用するための intra-mart Accel Platform のセットアップ方法について説明します。

項目



## 前提条件

IM-Copilot は以下のエディションでご利用が可能です。

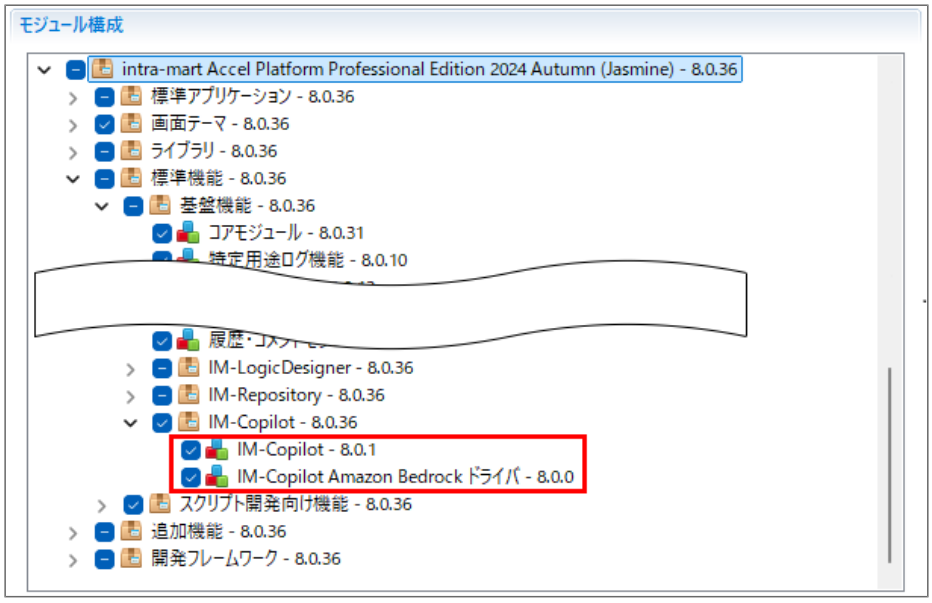
- パッケージライセンス
  - intra-mart Accel Platform Advanced Edition 2025 Spring(Kamille) 以降
- カスタマーサクセスライセンス
  - intra-mart Accel Platform Low-Code Edition 2025 Spring(Kamille) 以降
  - intra-mart Accel Platform Advance Edition 2024 Spring(Iris) 以降
  - intra-mart Accel Platform Professional Edition 2024 Spring(Iris) 以降

> **コラム**
>
> Standard Edition, Basic Edition, Pro Code Edition ではご利用いただけません。

## セットアップ手順

### IM-Juggling プロジェクトの編集

1. ご利用のIM-Jugglingプロジェクトに、IM-Copilotモジュールを追加してください。
   IM-Copilotモジュールは、[標準機能 - 基盤機能 - IM-Copilot] 配下に存在します。




> **コラム**
>
> Amazon Bedrock を生成AIサービスとして利用する場合は IM-Copilot Amazon Bedrock ドライバモジュールを選択してください。

2. エラーメッセージが表示された場合は、そのメッセージをクリックし、指示に従って以下の設定ファイルを追加してください。

- IM-Copilot生成AI連携ドライバ設定
- IM-Copilot生成AI連携アクション設定

> **コラム**
>
> IM-Jugglingの使用方法の詳細については、「intra-mart Accel Platform セットアップガイド」 -「プロジェクトの作成とモジュールの選択」 を参照してください。

## 生成AI連携ドライバ設定

1. 生成AI連携ドライバ設定ファイルを編集します。
   「ProjectNavigator」内の <（プロジェクト名）/conf/im-copilot-driver-config.xml> ファイルをダブルクリックで開き、「ソース」タブを選択してください。

   各生成AIサービスに対しての接続に関する設定を行います。

```
<?xml version="1.0" encoding="UTF-8"?>
<im-copilot-driver-config xmlns="https://www.intra-mart.jp/im-copilot/im-copilot-driver-config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="https://www.intra-mart.jp/im-copilot/im-
copilot-driver-config ../schema/im-copilot-driver-config.xsd ">
 <!-- Linkage Driver Setting for Common to Tenants -->
 <default-drivers>
  <driver type="open-ai">
   <api-key>sk-0000XXXXXX</api-key>
   <base-url>https://api.openai.com/v1</base-url>
   <organization>XXXXXXXX</organization>
   <retry-count>3</retry-count>
   <retry-wait>1</retry-wait>
   <parameters>
    <parameter>
     <parameter-name>default-chat-model</parameter-name>
     <parameter-value>gpt-4o-mini</parameter-value>
    </parameter>
    <parameter>
     <parameter-name>default-embeddings-model</parameter-name>
     <parameter-value>text-embedding-3-small</parameter-value>
    </parameter>
    <parameter>
     <parameter-name>default-images-model</parameter-name>
     <parameter-value>dall-e-2</parameter-value>
    </parameter>
    <parameter>
     <parameter-name>default-speech-model</parameter-name>
     <parameter-value>tts-1</parameter-value>
    </parameter>
    <parameter>
     <parameter-name>default-speech-voice</parameter-name>
     <parameter-value>echo</parameter-value>
    </parameter>
    <parameter>
     <parameter-name>default-transcription-model</parameter-name>
     <parameter-value>whisper-1</parameter-value>
    </parameter>
   </parameters>
  </driver>
  <driver type="azure-open-ai">
   <api-key>9999XXXXXX</api-key>
   <base-url>https://openai-service-foo.openai.azure.com/openai/</base-url>
   <retry-count>3</retry-count>
   <retry-wait>1</retry-wait>
   <parameters>
    <parameter>
     <parameter-name>default-chat-deployment-id</parameter-name>
     <parameter-value>gpt-4o-mini</parameter-value>
    </parameter>
    <parameter>
     <parameter-name>default-embeddings-deployment-id</parameter-name>
     <parameter-value>text-embedding-ada-002</parameter-value>
    </parameter>
```

```
    <parameter>
     <parameter-name>default-images-deployment-id</parameter-name>
     <parameter-value>dall-e-3</parameter-value>
    </parameter>
    <parameter>
     <parameter-name>default-transcription-deployment-id</parameter-name>
     <parameter-value>whisper</parameter-value>
    </parameter>
   </parameters>
  </driver>
  <driver type="amazon-bedrock">
   <aws-region>ap-northeast-1</aws-region>
   <aws-credentials-profile>default</aws-credentials-profile>
   <parameters>
    <parameter>
     <parameter-name>default-chat-model</parameter-name>
     <parameter-value>anthropic.claude-3-haiku-20240307-v1:0</parameter-value>
    </parameter>
    <parameter>
     <parameter-name>default-chat-max-tokens</parameter-name>
     <parameter-value>4096</parameter-value>
    </parameter>
    <parameter>
     <parameter-name>default-embeddings-model</parameter-name>
     <parameter-value>amazon.titan-embed-text-v1</parameter-value>
    </parameter>
    <parameter>
     <parameter-name>default-images-model</parameter-name>
     <parameter-value>amazon.nova-canvas-v1:0</parameter-value>
    </parameter>
   </parameters>
  </driver>
 </default-drivers>
</im-copilot-driver-config>
```

2. 以下のドライバ設定の項目を記述してください。
   設定内容の詳細については、 「設定ファイルリファレンス」 - 「IM-Copilot生成AI連携ドライバ設定」 を参照してください。

   - OpenAI を利用する場合
     <driver>タグの type属性に、`open-ai` を指定してください。
     <api-key>タグ 、<base-url>タグ が必須項目です。
     必要に応じて、 <organization>タグ 、<retry-count>タグ、<retry-wait>タグを設定してください。
   - Azure OpenAI Service を利用する場合
     <driver>タグの type属性に、`azure-open-ai` を指定してください。
     <api-key>タグ 、<base-url>タグ が必須項目です。
     必要に応じて、 <retry-count>タグ、<retry-wait>タグを設定してください。
   - Amazon Bedrock を利用する場合
     <driver>タグの type属性に、`amazon-bedrock` を指定してください。
     <aws-region>タグにAWSリージョンコードを設定してください。（例：us-east-1、ap-northeast-1）
     認証方式に応じて以下のいずれかの設定を行ってください。
     - 認証情報ファイルを利用して認証する場合
       <aws-credentials-profile>タグを記述し、*認証情報ファイルの作成* で得たプロファイル名を設定してください。
     - 設定ファイルに直接記述したアクセスキーとシークレットキーを利用して認証する場合
       <aws-credentials-static>タグを記述し、*アクセスキーとシークレットキーの作成* で得たアクセスキーとシークレットキーを設定してください。
     - インスタンスプロファイルを利用して認証する場合（※Amazon EC2 インスタンス上で iAP が動作している場合のみ）
       <aws-credentials-iam-role-arn>タグを記述し、*ロールの作成* で作成したロールのARNを設定してください。
     - ガードレール機能を利用する場合は、<parameters>タグ内に以下のパラメータを設定してください。
       - `default-guardrail-enabled` - ChatAction API を利用して動作する機能に対するガードレール機能の有効化フラグ（true/false）
       - `default-guardrail-id` - *ガードレールの作成（任意）* で作成したガードレールID
       - `default-guardrail-version` - 使用するガードレールのバージョン（番号または DRAFT）

       パラメータの詳細については、「設定ファイルリファレンス」 - 「Amazon Bedrock ガードレール デフォルトパラメータ設定」を参照してください。
   - <parameters>タグ内にデフォルトパラメータを指定してください。（モデル、デプロイ名）

## 生成AI連携アクション設定

1. 生成AI連携アクション設定ファイルを編集します。
   「ProjectNavigator」内の <（プロジェクト名）/conf/im-copilot-action-config.xml> ファイルをダブルクリックで開き、「ソース」タブを選択してください。

2. 利用するテナントについて、各アクションの利用ドライバ種別を記述してください。
   設定内容の詳細については、 「設定ファイルリファレンス」 - 「IM-Copilot生成AI連携アクション設定」 を参照してください。

# セットアップ（ 各種製品アシスタント ）

各種製品でアシスタント機能を利用するためのセットアップ方法について説明します。

## Wiki アシスタントのセットアップ

Wiki アシスタントのセットアップ方法について説明します。

項目



### Wiki アシスタントについて

Wiki アシスタントは IM-Wiki内の情報をチャット形式で問い合わせ可能にする機能です。

Wiki アシスタントの特徴は以下の通りです。

- IM-Wikiの任意のコンテンツに対してアシスタントを作成、利用できます。
- Wiki アシスタントの作成単位は「1 Wiki コンテンツ」＝「1 Wiki アシスタント」です。
- 各 Wiki アシスタントは認可設定を持っており利用可能なユーザを制御可能です。

機能の詳細については、「IM-Knowledge管理者操作ガイド」 - 「Wiki アシスタント」を参照してください。

> **注意**
>
> Wiki アシスタントは、intra-mart Accel Platform で提供しているベクトルデータベースアクセス機能を利用して検索拡張生成（RAG）による応答を行います。
> 応答生成時にはキーワード検索とベクトル類似性検索で情報収集を行いますが、標準のベクトルデータベースアクセス機能では、日本語と英語の検索キーワード抽出トークナイザのみ提供しています。
> そのため、日本語・英語以外の言語ではキーワード抽出ができず、キーワード検索による情報収集が不十分となり、回答精度が低下する可能性があります。

### セットアップ手順

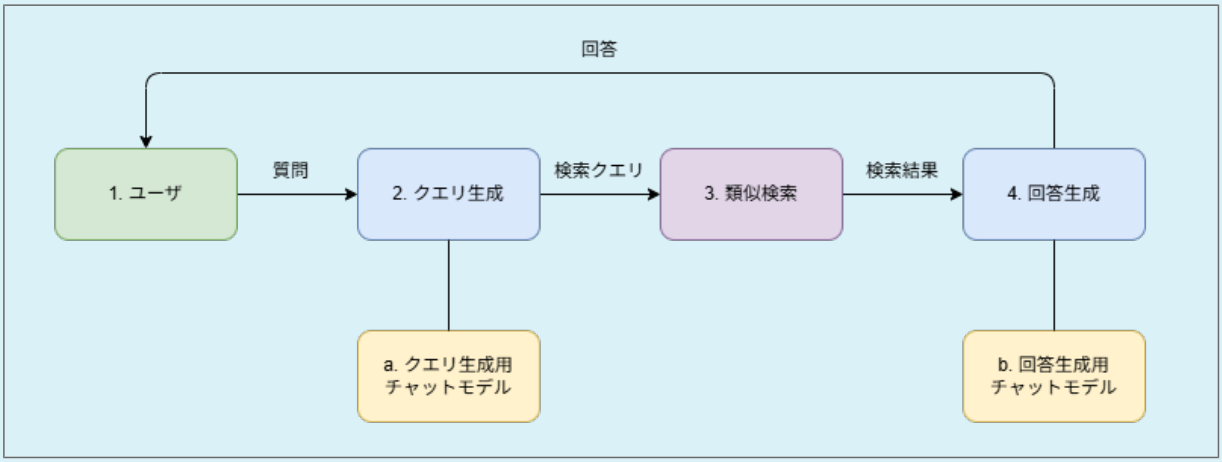
#### 生成AIサービスとモデル

Wiki アシスタントにおけるサポートモデルは、「*サポートモデル*」を参照してください。

> **コラム**
>
> Wiki アシスタント 専用の個別パラメータを使用することで、Wiki アシスタント 向けのモデル指定が可能です。
> 詳細は「*関連情報*」-「*機能別モデル設定パラメータ名一覧*」を参照してください。

コラム

Wiki アシスタント が呼び出される際には、下記の処理が実行されます。
1. 質問: ユーザが質問をします。
2. クエリ生成: ユーザの質問に基づき、「a. クエリ生成用チャットモデル」を呼び出して効果的な検索クエリを生成します。
3. 類似検索: クエリをベクトル化し、類似するベクトルを持つチャンクを検索します。
4. 回答生成: ユーザの質問と検索結果をコンテキストとして、「b. 回答生成用チャットモデル」を呼び出して最終的な回答を生成します。




「a. クエリ生成用チャットモデル」と「b. 回答生成用チャットモデル」は Wiki アシスタント 専用の個別パラメータを使用して別々で設定可能です。
詳細は「*関連情報*」-「*機能別モデル設定パラメータ名一覧*」を参照してください。

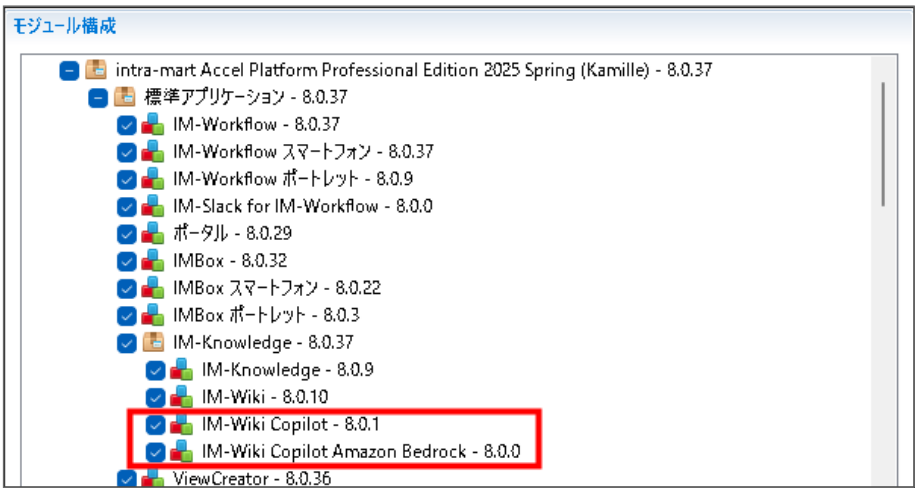
## 生成AI連携ドライバ設定

Wiki アシスタントは、*生成AI連携ドライバ設定* ファイルの一番初めに記載されているドライバ設定を基に生成AIサービスに接続します。

- *生成AI連携ドライバ設定* ファイルにテナントドライバ情報設定（drivers）の設定が存在する場合、その xml 内で一番初めに記載されているドライバ設定を基に生成AIサービスに接続します。
- テナントドライバ情報設定（drivers）の設定が存在しない場合は、デフォルトドライバ情報設定（default-drivers）の xml 内で一番初めに記載されているドライバ設定を基に生成AIサービスに接続します。

## IM-Jugglingプロジェクトの編集

ご利用のIM-Jugglingプロジェクトに、IM-Wiki Copilotモジュールを追加してください。
IM-Wiki Copilotモジュールは、[標準アプリケーション - IM-Knowledge] 配下に存在します。




## テキスト抽出設定

Wiki アシスタントは、添付ファイルのテキスト抽出にテキスト抽出機能（ND Universal Extractor）を利用します。
必要に応じて、テキスト抽出機能のテキスト抽出設定の調整を検討してください。
設定内容の詳細については、「設定ファイルリファレンス」 -「テキスト抽出設定」 を参照してください。

### ベクトルデータベース接続設定

Wiki アシスタントは、テナント環境セットアップでベクトルデータベース接続情報の設定が必要です。
設定内容の詳細については、「テナント環境セットアップ」 - 「ベクトルデータベース接続情報」 を参照してください。

### テナントセットアップ後の設定

テナントセットアップ後、運用に合わせて Wiki アシスタントに関する設定が必要です。
設定内容の詳細については、「IM-Knowledge管理者操作ガイド」 - 「Wiki アシスタント」を参照してください。

## 運用時の注意事項

### 生成AIサービス・埋め込みモデル変更時の対応

Wiki アシスタント の運用中に、生成AIサービスまたは埋め込みモデルを変更した場合は、既存のベクトルデータの再構築が必要です。
詳細は「ジョブ・ジョブネットリファレンス」-「Wiki アシスタント同期」を参照してください。

> **注意**
>
> 埋め込みモデルが変更されると、同じテキストでも異なるベクトル値が生成されるため、既存のベクトルデータと新しいクエリのベクトルが適切に比較できなくなります。
> そのため、変更後の埋め込みモデルを使用してすべてのコンテンツを再度ベクトル化し、ベクトルデータベースを更新する必要があります。

# ViewCreator SQLビルダ アシスタント のセットアップ

SQL生成アシスタントのセットアップ方法について説明します。

項目



## ViewCreator SQLビルダ アシスタント について

ViewCreator SQLビルダ アシスタント は、テーブルのデータを参照するためのSQL（SELECT文）作成を、生成AIがサポートする機能です。
「ユーザの一覧を取得するSQLを作成してください」のような、自然言語による指示からSQLを生成できます。
ViewCreator SQLビルダ アシスタント は、製品標準で作成されるテーブル情報について回答できます。
具体的な利用方法は「SQLビルダによるクエリの作成」を参照してください。
また、独自で追加したテーブルについても、生成AIが参照可能な学習データを追加できます。
具体的な追加方法は「独自のテーブルメタデータ情報を追加する」を参照してください。

> **コラム**
>
> 製品標準で用意されるテーブルであっても、ViewCreator SQLビルダ アシスタント が利用できないテーブルがあります。
> 利用可能なテーブルについて、質問できます。
> 　例）「xxxテーブルやxxxテーブルは利用できますか？」

## セットアップ手順

### 生成AIサービスとモデル

ViewCreator SQLビルダ アシスタント におけるサポートモデルは、「*サポートモデル*」を参照してください。

コラム

ViewCreator SQLビルダ アシスタント 専用の個別パラメータを使用することで、 ViewCreator SQLビルダ アシスタント 向けのモデル指定が可能です。
詳細は「*関連情報*」-「*機能別モデル設定パラメータ名一覧*」を参照してください。

注意

ViewCreator SQLビルダ アシスタント における Azure OpenAI Service の標準埋め込みモデルは*text-embedding-3-small* です。
*Azure OpenAI Serviceのセットアップ*で intra-mart Accel Platform の標準埋め込みモデル*text-embedding-ada-002* をデプロイした場合は、ドライバ設定の個別パラメータ「Azure OpenAI Service ViewCreator SQLビルダ アシスタント 埋め込みモデルデプロイ名 パラメータ設定」で *text-embedding-ada-002* を設定してください。
ViewCreator SQLビルダ アシスタント では intra-mart Accel Platform の標準埋め込みモデル*text-embedding-ada-002* も動作検証しています。

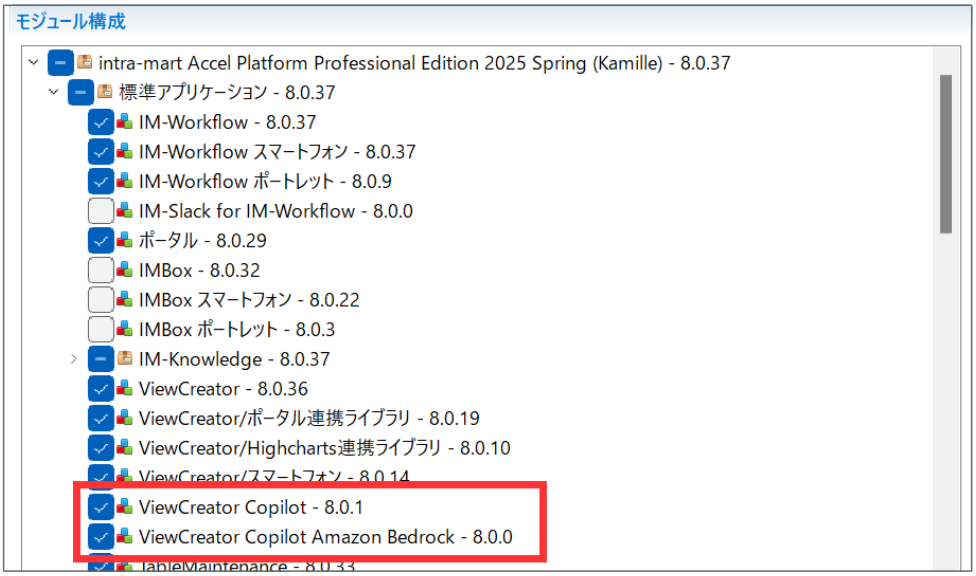
## 生成AI連携ドライバ設定

ViewCreator SQLビルダ アシスタント は、*生成AI連携ドライバ設定* ファイルの一番初めに記載されているドライバ設定を基に生成AIサービスに接続します。

- *生成AI連携ドライバ設定* ファイルにテナントドライバ情報設定（drivers）の設定が存在する場合、その xml 内で一番初めに記載されているドライバ設定を基に生成AIサービスに接続します。
- テナントドライバ情報設定（drivers）の設定が存在しない場合は、デフォルトドライバ情報設定（default-drivers）の xml 内で一番初めに記載されているドライバ設定を基に生成AIサービスに接続します。
- Azure OpenAI Service はドライバ設定の個別パラメータで ViewCreator SQLビルダ アシスタント で使用するモデルを指定できます。
  個別パラメータについては、 「設定ファイルリファレンス」 - 「IM-Copilot生成AI連携ドライバ設定」を参照してください。

## IM-Jugglingプロジェクトの編集

ご利用のIM-Jugglingプロジェクトに、ViewCreator Copilotモジュールを追加してください。
ViewCreator Copilotモジュールは、[標準アプリケーション] 配下に存在します。




## ベクトルデータベース接続設定

ViewCreator SQLビルダ アシスタント は、テナント環境セットアップでベクトルデータベース接続情報の設定が必要です。
設定内容の詳細については、 「テナント環境セットアップ」 - 「ベクトルデータベース接続情報」 を参照してください。

## テナントセットアップ後の設定

テナントセットアップ後、アシスタント使用前の準備として以下の操作が必要です。

- ViewCreator SQLビルダ アシスタント の認可設定

- ジョブネットの実行

詳細については、「ViewCreator 管理者操作ガイド」 を参照してください。

# Accel Studio アプリケーション作成 アシスタント のセットアップ

Accel Studio アプリケーション作成 アシスタント のセットアップ方法について説明します。

項目



## Accel Studio アプリケーション作成 アシスタント について

Accel Studio アプリケーション作成 アシスタント は、Accel Studioのアプリケーションを作成する際、自然言語による指示でアプリケーションの入力項目の自動補完を行う機能です。
例えば、「書籍管理アプリケーションを作成してください」といった指示に対して、アプリケーションの入力項目を自動補完します。
具体的な利用方法は 「Accel Studio アプリケーション管理機能 仕様書」 - 「アプリケーション作成 アシスタント」を参照してください。

## セットアップ手順

### 生成AIサービスとモデル

Accel Studio アプリケーション作成 アシスタント におけるサポートモデルは、「*サポートモデル*」を参照してください。

> **コラム**
>
> Accel Studio アプリケーション作成 アシスタント 専用の個別パラメータを使用することで、 Accel Studio アプリケーション作成 アシスタント 向けのモデル指定が可能です。
> 詳細は「*関連情報*」-「*機能別モデル設定パラメータ名一覧*」を参照してください。

### 生成AI連携ドライバ設定

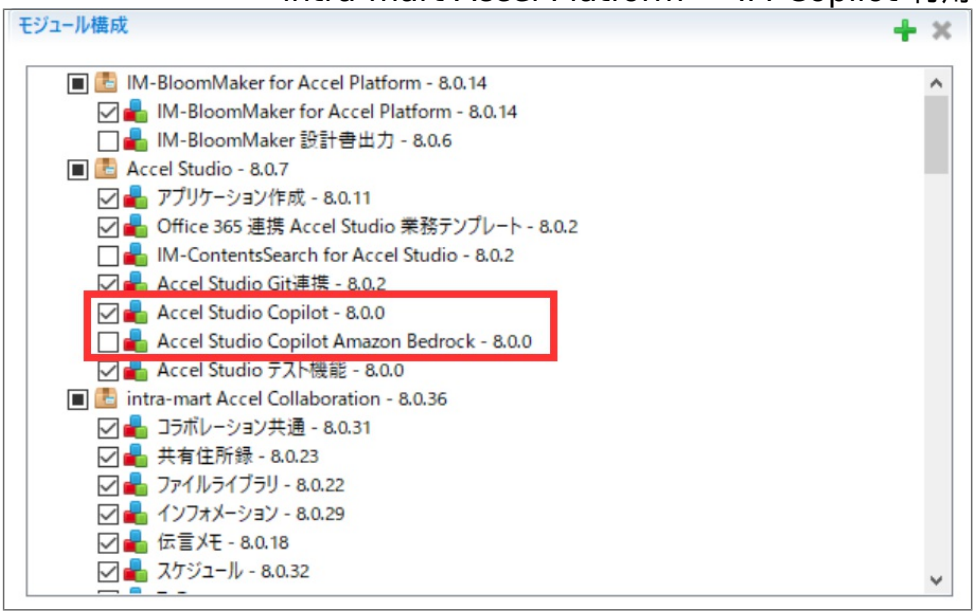
Accel Studio アプリケーション作成 アシスタント は、*生成AI連携ドライバ設定* ファイルの一番初めに記載されているドライバ設定を基に生成AIサービスに接続します。

- *生成AI連携ドライバ設定* ファイルにテナントドライバ情報設定（drivers）の設定が存在する場合、その xml 内で一番初めに記載されているドライバ設定を基に生成AIサービスに接続します。
- テナントドライバ情報設定（drivers）の設定が存在しない場合は、デフォルトドライバ情報設定（default-drivers）の xml 内で一番初めに記載されているドライバ設定を基に生成AIサービスに接続します。

> **コラム**
>
> Accel Studio アプリケーション作成 アシスタント で利用するモデルを変更する場合は、*生成AI連携ドライバ設定* ファイルのドライバ設定を変更してください。

### IM-Juggling プロジェクトの編集

ご利用のIM-Juggling プロジェクトに、Accel Studio Copilotモジュールを追加してください。
Accel Studio Copilotモジュールは、[アプリケーション作成] 配下に存在します。




# Accel Studio テスト機能 Copilot のセットアップ

Accel Studio テスト機能 Copilot のセットアップ方法について説明します。

項目



## Accel Studio テスト機能 Copilot について

Accel Studio テスト機能 Copilot は、Accel Studio テスト機能 のテスト定義を生成する機能です。
IM-BloomMaker などで作成された画面から画面情報を取得、解析し、テスト定義を生成します。
Accel Studio テスト機能 Copilot の具体的な利用方法は 「Accel Studio テスト機能 利用ガイド」 - 「6.4. テスト生成」を参照してください。

## セットアップ手順

### 生成AIサービスとモデル

Accel Studio テスト機能 Copilot におけるサポートモデルは、「*サポートモデル*」を参照してください。

### 生成AI連携ドライバ設定

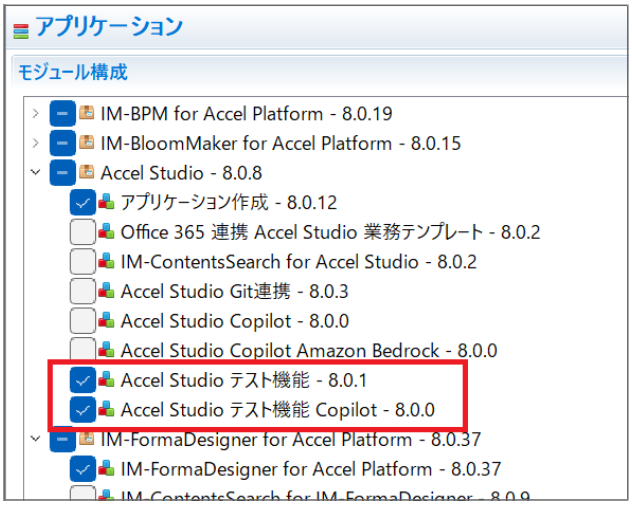
Accel Studio テスト機能 Copilot は、*生成AI連携ドライバ設定* ファイルの一番初めに記載されているドライバ設定を基に生成AIサービスに接続します。

- *生成AI連携ドライバ設定* ファイルにテナントドライバ情報設定（drivers）の設定が存在する場合、その xml 内で一番初めに記載されているドライバ設定を基に生成AIサービスに接続します。
- テナントドライバ情報設定（drivers）の設定が存在しない場合は、デフォルトドライバ情報設定（default-drivers）の xml 内で一番初めに記載されているドライバ設定を基に生成AIサービスに接続します。

> **コラム**
>
> Accel Studio テスト機能 Copilot で利用するモデルを変更する場合は、*生成AI連携ドライバ設定* ファイルのドライバ設定を変更してください。

### IM-Jugglingプロジェクトの編集

ご利用のIM-Jugglingプロジェクトに、Accel Studio テスト機能 Copilotモジュールを追加してください。
Accel Studio テスト機能 Copilotモジュールは、Accel Studio 配下に存在します。
Accel Studio テスト機能 Copilotを利用するには、Accel Studio テスト機能モジュールが必要です。




> **コラム**
>
> 各種製品で利用できるアシスタントは、今後の製品アップデートに合わせて追加を予定しています。

# 共通アシスタント実行画面



## 概要

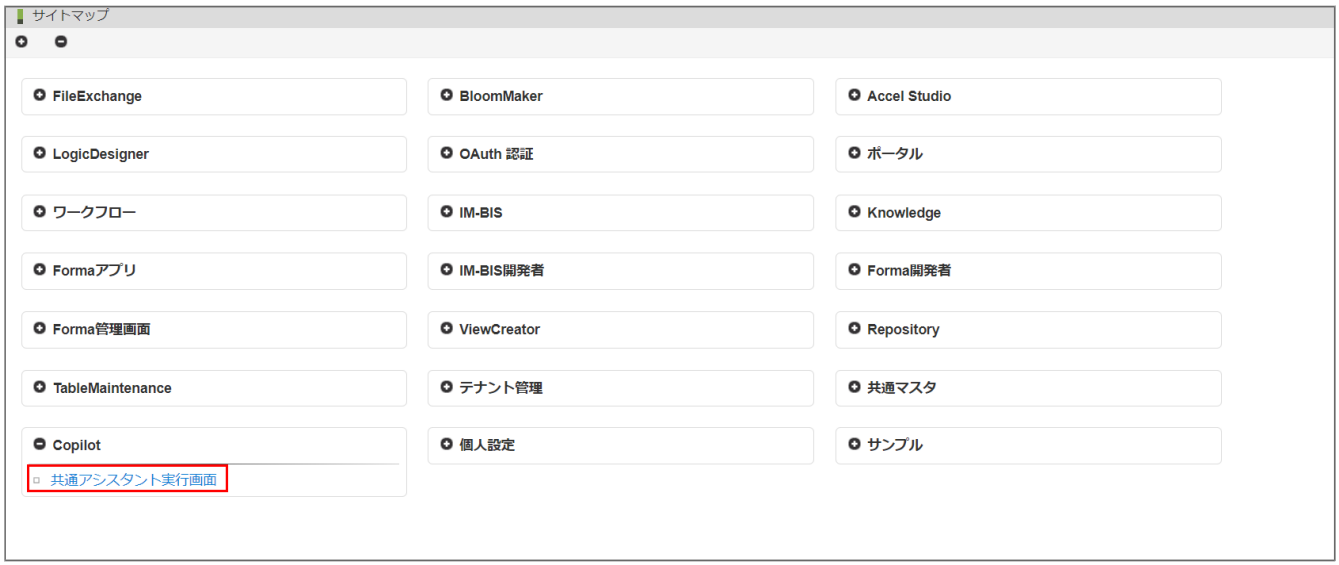
共通アシスタント実行画面は生成AIアシスタントの実行を行うための共通画面です。
作成した Wiki アシスタントの実行結果を確認したい場合等に利用します。

> **コラム**
>
> Wiki アシスタントの登録方法については、「IM-Knowledge管理者操作ガイド」 - 「Wiki アシスタント」を参照してください。

アシスタントの実行方法は以下のとおりです。

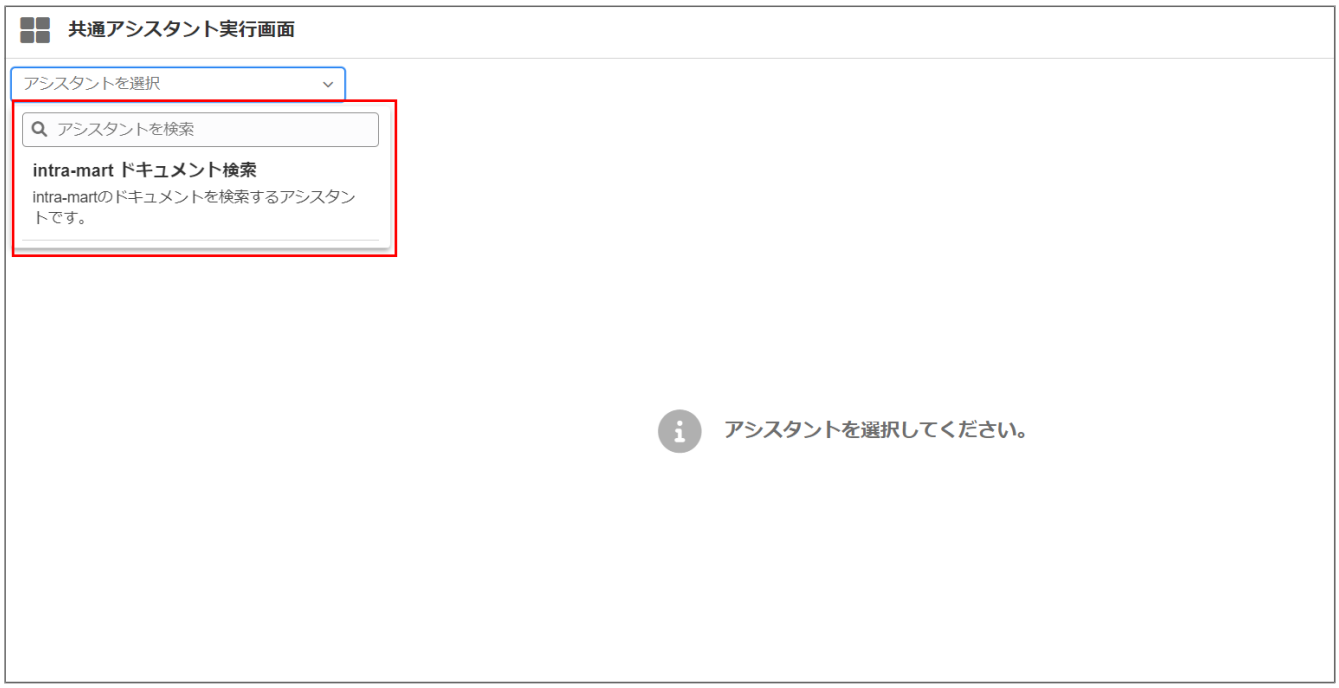
## アシスタントの実行

1. 「サイトマップ」→「Copilot」→「共通アシスタント実行画面」をクリックします。




2. 共通アシスタント実行画面の「アシスタントを選択」セレクトボックスをクリックします。


3. アシスタントの一覧から実行するアシスタントを選択します。

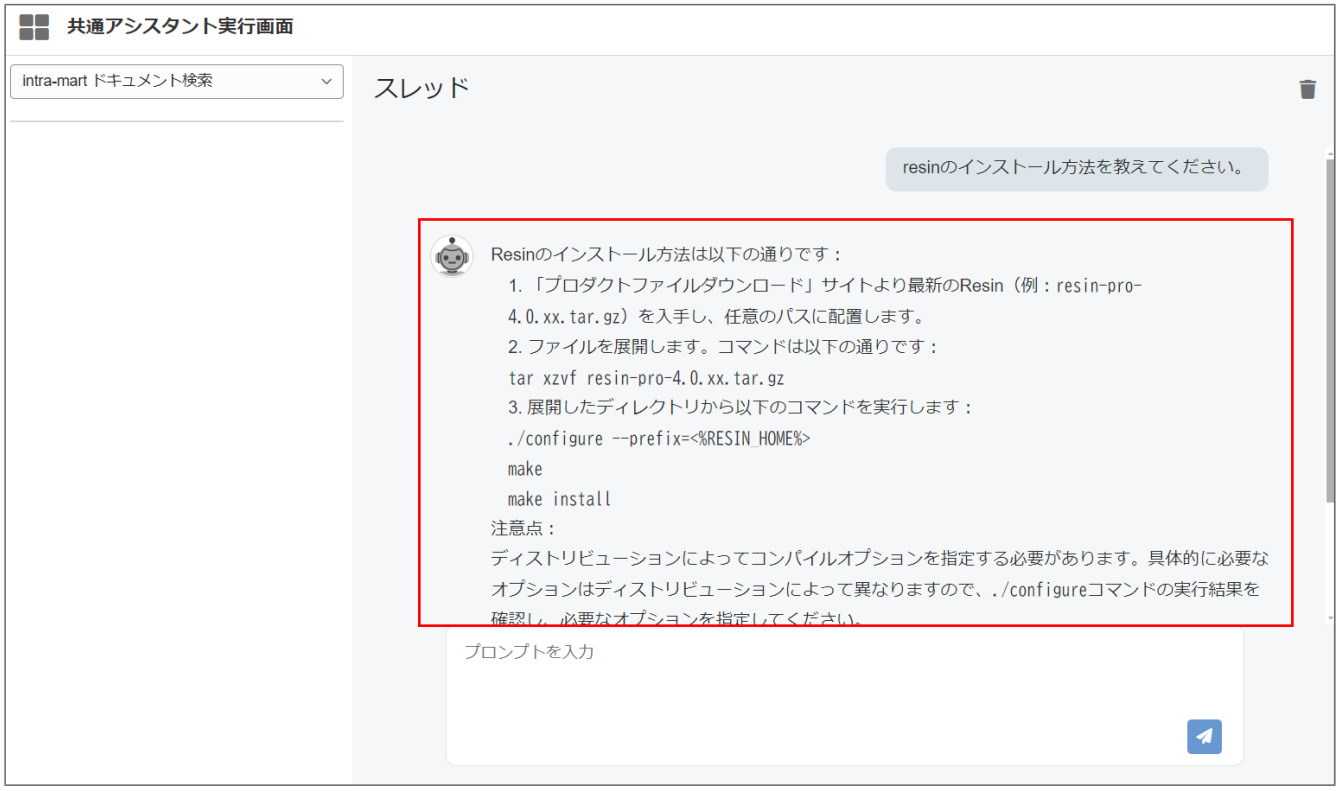



4. プロンプトの入力欄に問い合わせ内容を入力します。


5. アシスタントの実行ボタンをクリックします。または、以下のキーボードショートカットをご利用いただけます。
   - Windows: `Ctrl + Enter`
   - Mac: `Command + Enter`


6. アシスタントの実行が開始され、「処理中」と表示されます。


7. 問い合わせ結果が表示されます。




## メッセージ履歴の削除

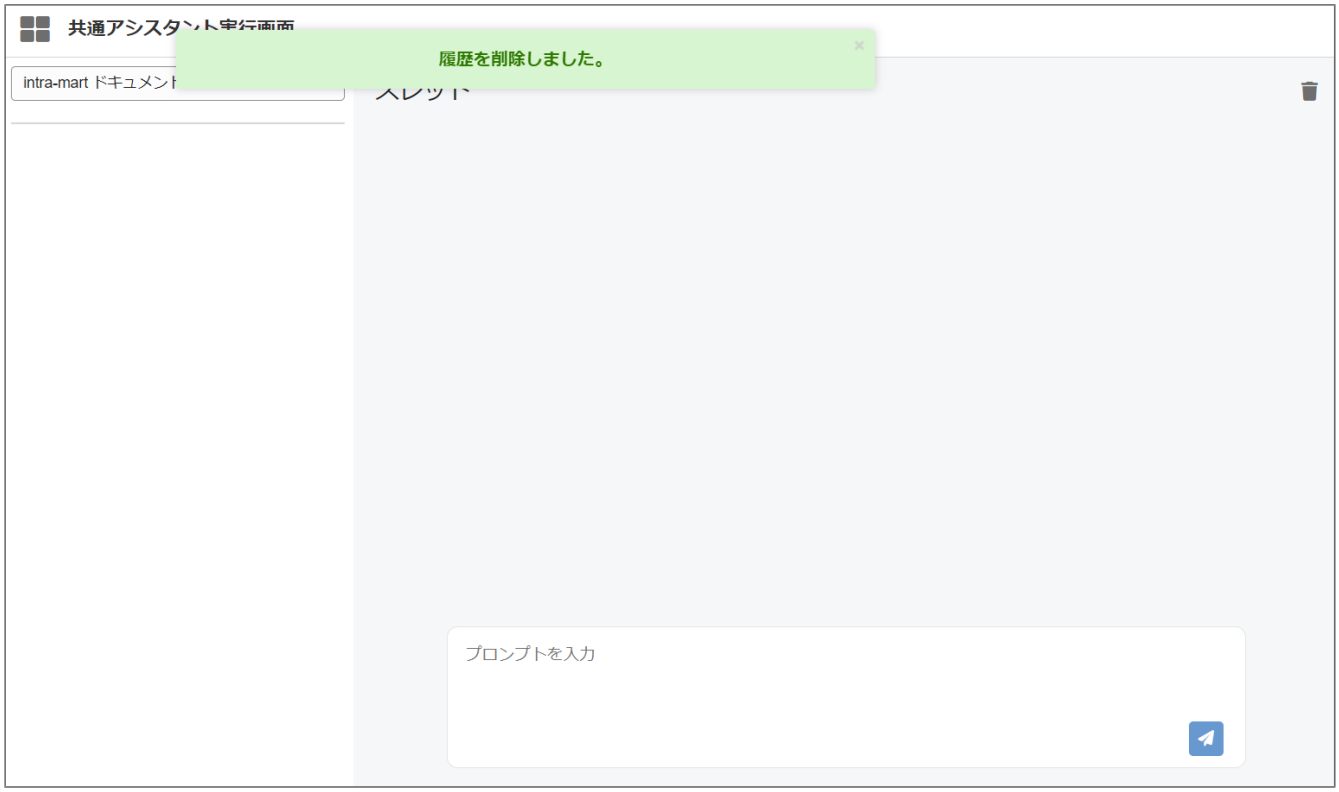
メッセージ履歴を利用するアシスタントの場合、問い合わせ内容とその結果がメッセージ履歴として保存されます。
保存された履歴はアシスタント実行時に生成AIサービスへのリクエスト送信時等に利用されます。
そのため、メッセージ履歴が多い場合にトークン数上限に到達してしまいエラーが発生する場合があります。

不要なメッセージ履歴は以下の手順で削除できます。

1. 共通アシスタント実行画面の「削除」アイコンをクリックします。


2. 確認ダイアログの「確定」ボタンをクリックします。


3. メッセージ履歴が削除されます。


履歴を削除しました。
プロンプトを入力

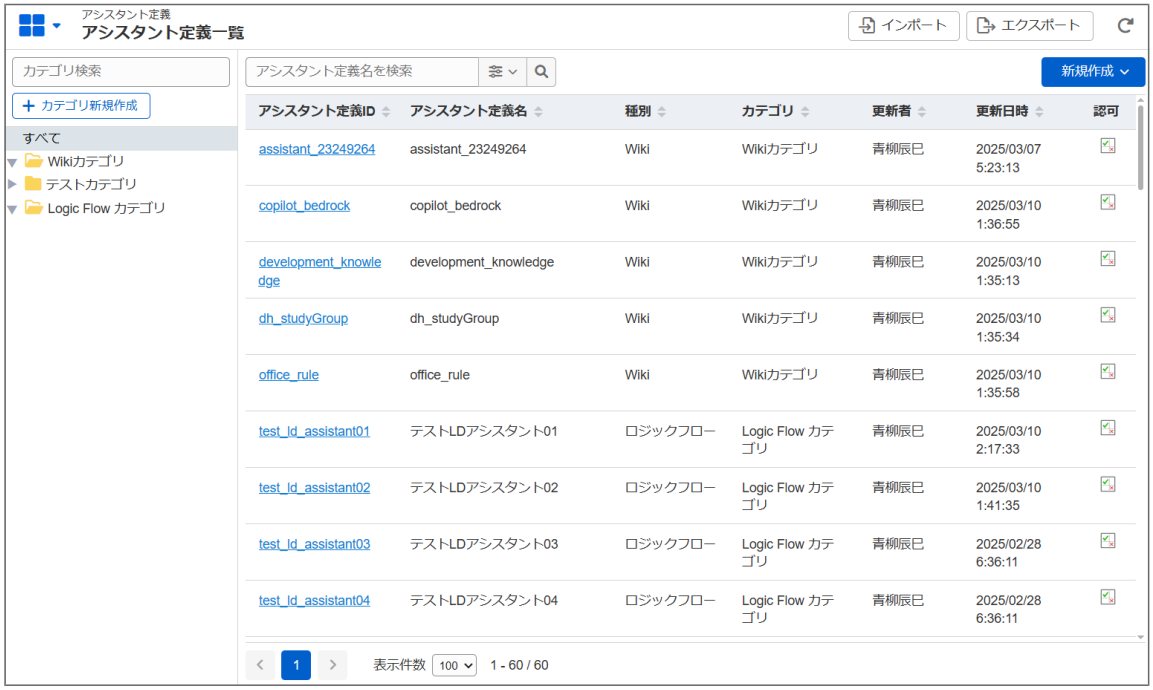
# アシスタント定義

アシスタント定義は、 IM-LogicDesignerのロジックフローやIM-Wikiのコンテンツから生成AIアシスタントを作成・管理する機能です。 ここではアシスタント定義を扱う画面の機能について説明します。

## アシスタント定義一覧

1. 「サイトマップ」→「Copilot」→「アシスタント定義」→「アシスタント定義一覧」をクリックし、「アシスタント定義一覧」画面を表示します。


2. 「アシスタント定義一覧」画面が表示されます。




＜画面項目（ヘッダ）＞

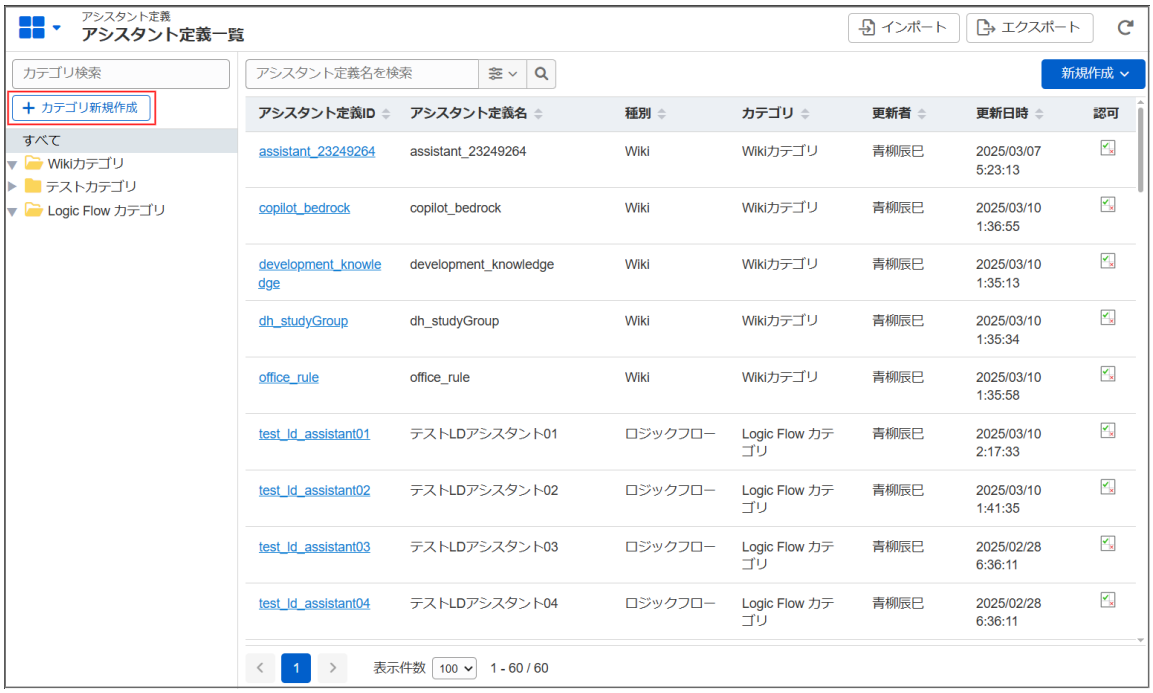
| 項目 | 説明 |
|---|---|
| 「関連リンク」アイコン | 共通アシスタント実行、インポート、エクスポート画面へのリンクが表示されます。 |
| 「インポート」ボタン | アシスタント定義 インポート画面に遷移します。 |
| 「エクスポート」ボタン | アシスタント定義 エクスポート画面に遷移します。 |
| 「更新」アイコン | ページを再読み込みします。 |

＜画面項目（コンテンツ）＞

| 項目 | 説明 |
| --- | --- |
| カテゴリ検索窓 | 検索するカテゴリ名を表す文字列（の一部）を入力します。 |
| 「カテゴリ新規作成」ボタン | カテゴリ新規作成ダイアログが表示されます。 |
| アシスタント定義名検索窓 | 検索するアシスタント定義名を表す文字列（の一部）を入力します。 |
| 「詳細検索」アイコン | 検索するアシスタント定義ID、アシスタント定義名を表す文字列（の一部）を入力、またはアシスタント定義種別を指定して検索します。 |
| 「虫眼鏡」アイコン | 検索を実行します。 |
| 「新規作成」ボタン | アシスタント定義種別リストが表示されます。 |
| 「認可」アイコン | アシスタント利用ユーザがアシスタントを利用するための認可設定画面が表示されます。 |
| 「ページャ」アイコン | 一覧の表示ページを切り替えます。 |
| 「表示件数」プルダウン | エンティティログの表示件数を設定します。<br>設定可能は表示件数は、50, 100, 150, 200 件です。 |

## カテゴリを登録する

1. 「カテゴリ新規作成」アイコンをクリックします。



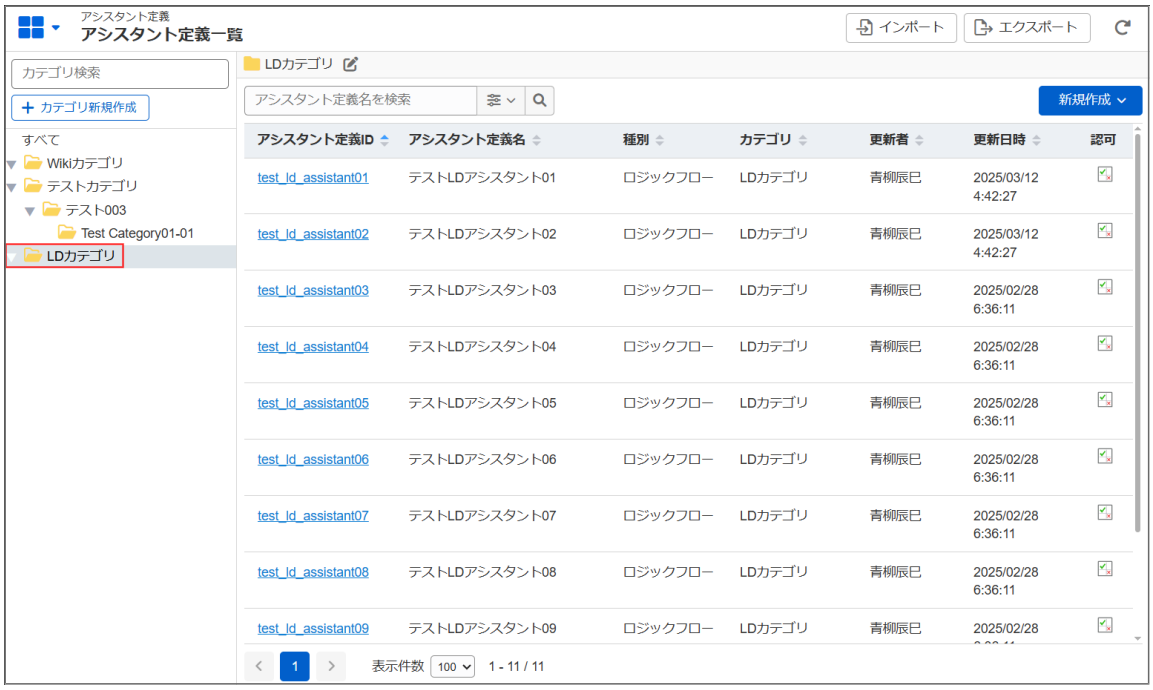

2. カテゴリ新規作成ダイアログが表示されます。
   カテゴリ情報を入力し「登録」ボタンをクリックします。

＜画面項目＞

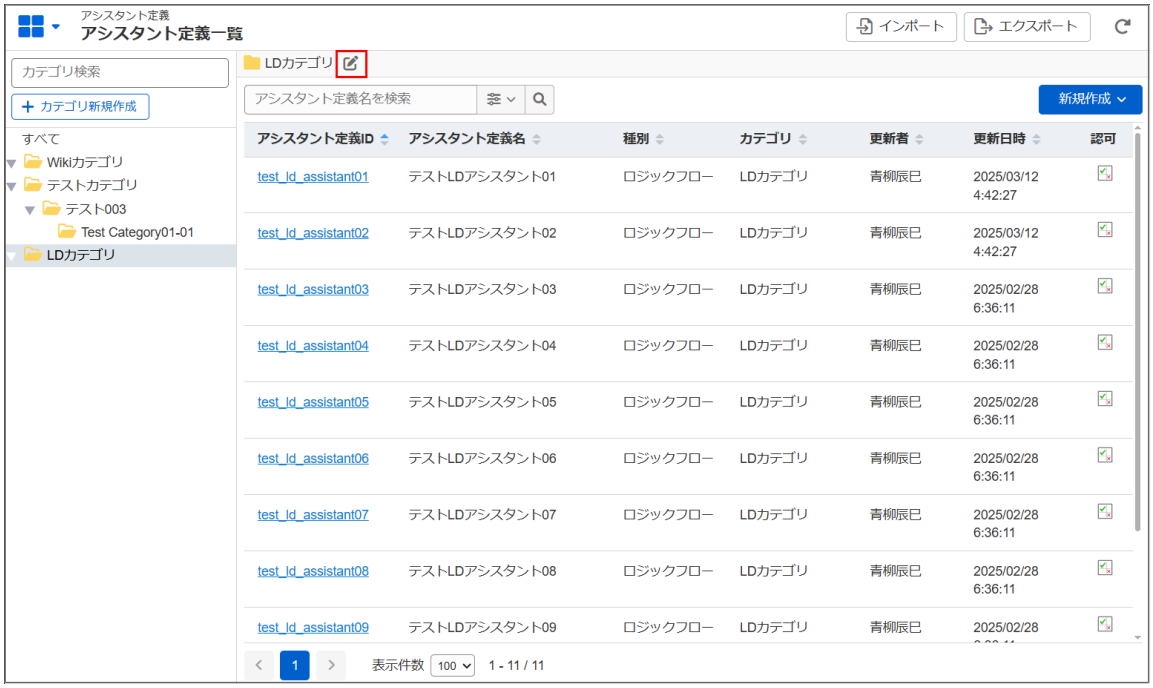
| 項目 | 説明 |
| --- | --- |
| カテゴリID | カテゴリを一意に表す文字列を設定します。<br>この項目は必須項目です。 |
| カテゴリ名 | カテゴリを表す名称を設定します。<br>名称には各言語で利用するものと、言語情報が指定されていない場合に標準で利用するものを設定します。<br>この項目は標準のみ必須項目です。 |
| ソート番号 | カテゴリ表示時のソート順を設定します。<br>この項目は必須項目です。 |
| 親カテゴリ | 親カテゴリを指定します。 |
| 「キャンセル」ボタン | 操作をキャンセルしてダイアログをクローズします。 |
| 「登録」ボタン | カテゴリを登録します。 |

## カテゴリを編集する

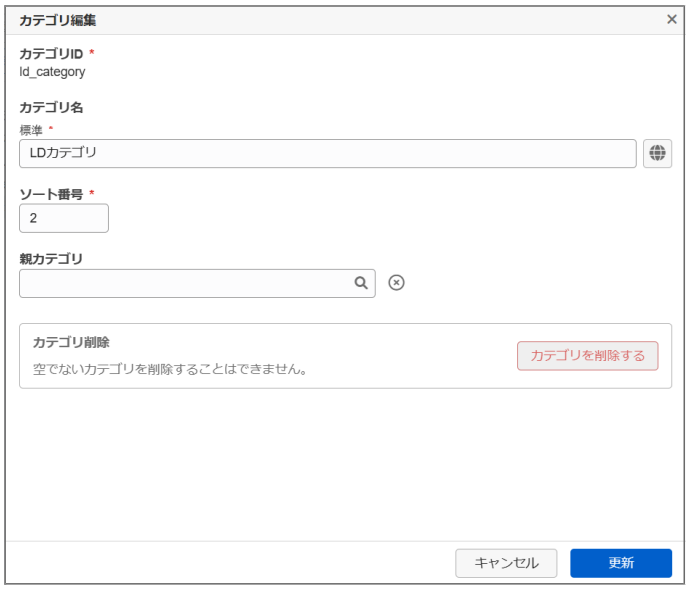
1. 「*アシスタント定義一覧*」の手順をもとに、アシスタント定義一覧を表示して、カテゴリツリーから編集するカテゴリをクリックします。




2. 「カテゴリ編集」アイコンをクリックします。

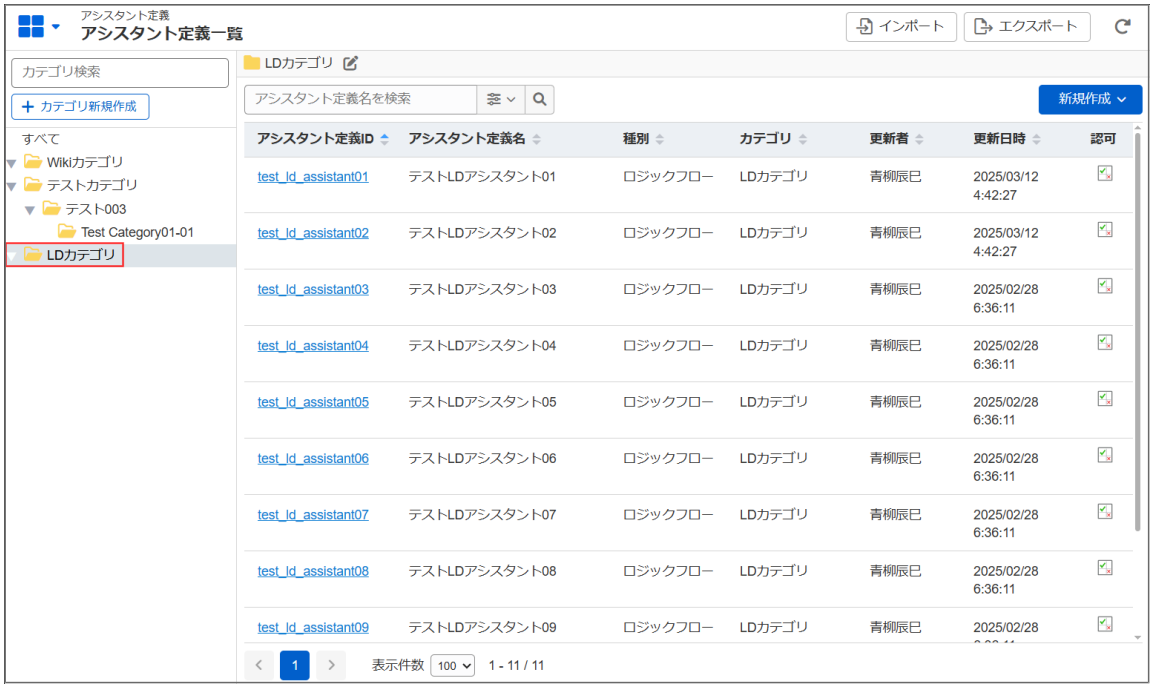



3. カテゴリ編集ダイアログが表示されます。
   カテゴリ情報を入力し「更新」ボタンをクリックします。

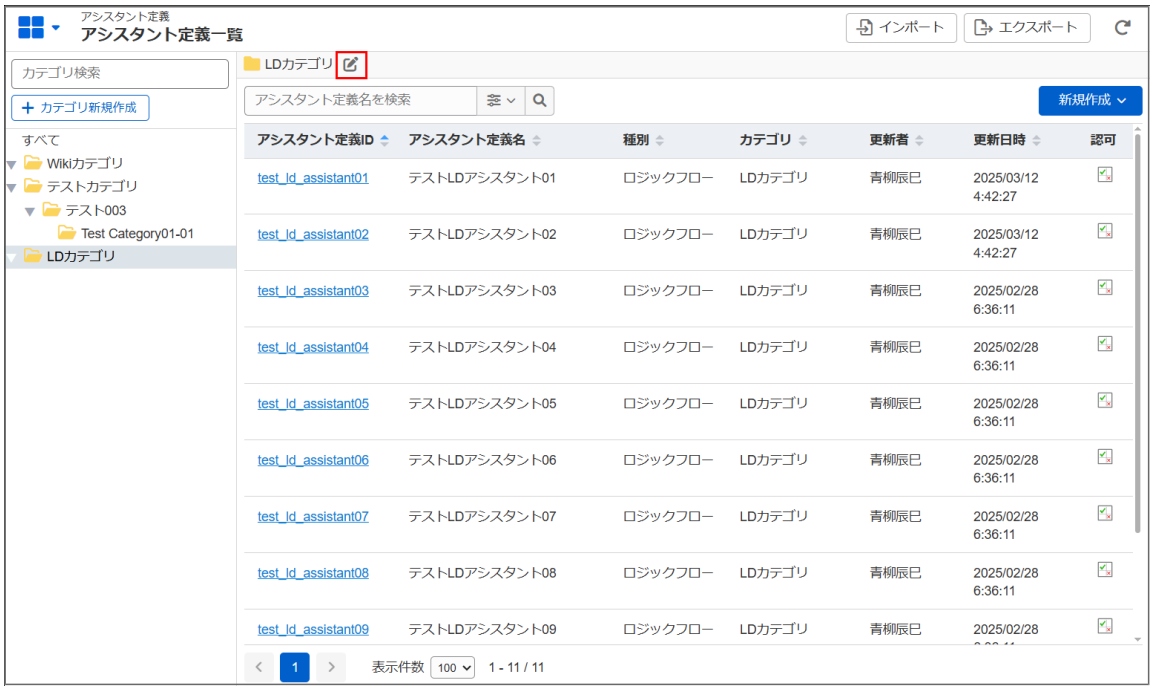



## カテゴリを削除する

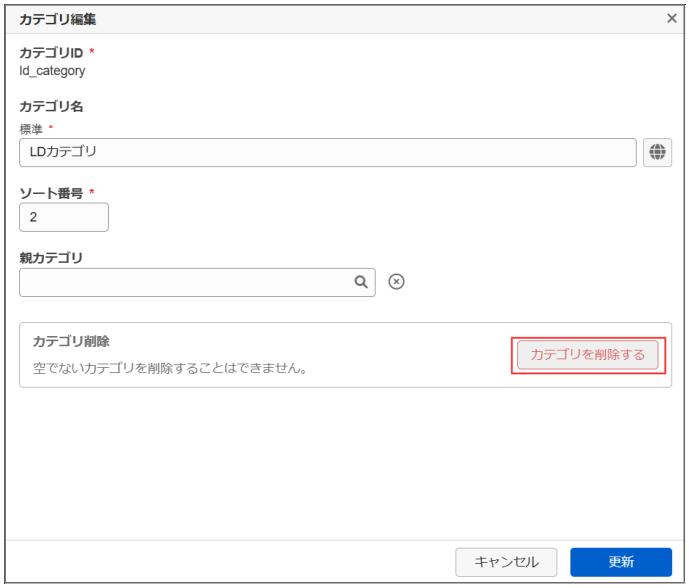
1. 「*アシスタント定義一覧*」の手順をもとに、アシスタント定義一覧を表示して、カテゴリツリーから削除するカテゴリをクリックします。




2. 「カテゴリ編集」アイコンをクリックします。

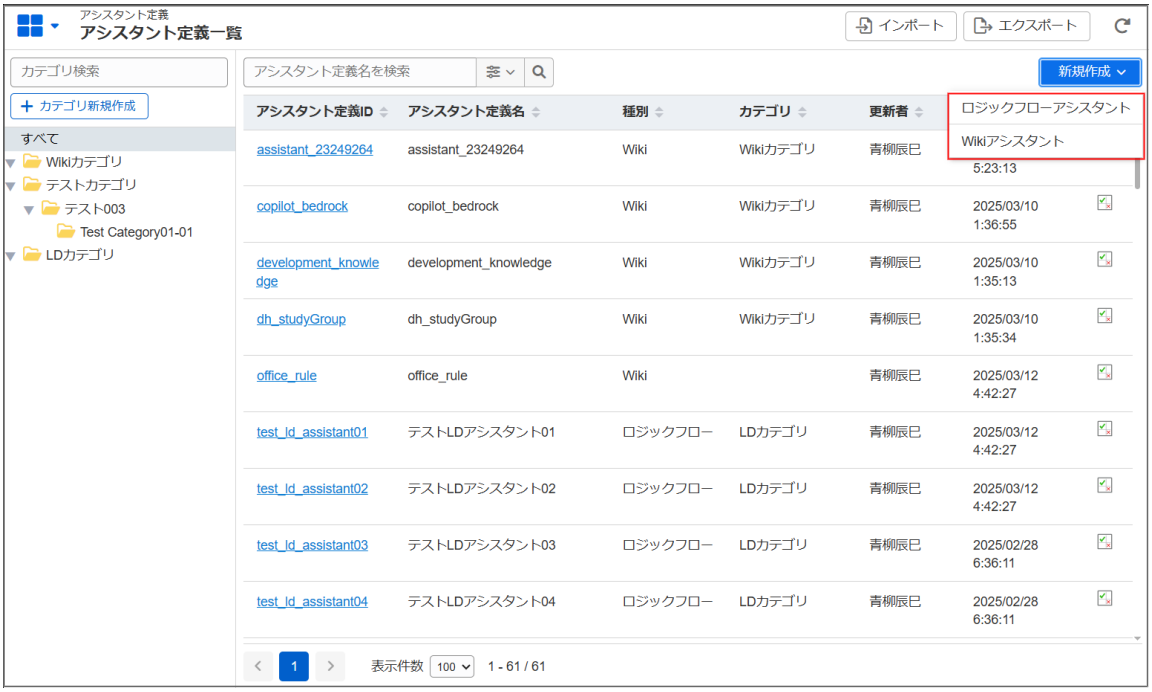



3. 「カテゴリを削除する」ボタンをクリックします。確認ダイアログの「削除」ボタンをクリックします。




## アシスタント定義を登録する

1. 「新規作成」ボタン、または「アシスタント定義を新規作成」ボタンをクリックします。
   アシスタント定義種別リストから作成対象のアシスタント定義種別を選択します。
   - 「新規作成」ボタンから作成




   - 「アシスタント定義を新規作成」ボタンから作成


2. アシスタント定義 新規作成画面に遷移します。
   アシスタント情報を入力し「新規作成」ボタンをクリックします。


＜画面項目（基本情報）＞

| 項目 | 説明 |
|---|---|
| アシスタントID | アシスタントを一意に表す文字列を設定します。<br>この項目は必須項目です。 |
| アシスタント名 | アシスタントを表す名称を入力します。<br>名称には各言語で利用するものと、言語情報が指定されていない場合に標準で利用するものを設定します。<br>この項目は標準のみ必須項目です。 |
| カテゴリ | アシスタントが属するカテゴリを指定します。 |
| 説明 | アシスタントの説明を入力します。<br>説明には各言語で利用するものと、言語情報が指定されていない場合に標準で利用するものを設定します。 |

＜画面項目（詳細情報 - ロジックフロー）＞

| 項目 | 説明 |
|---|---|

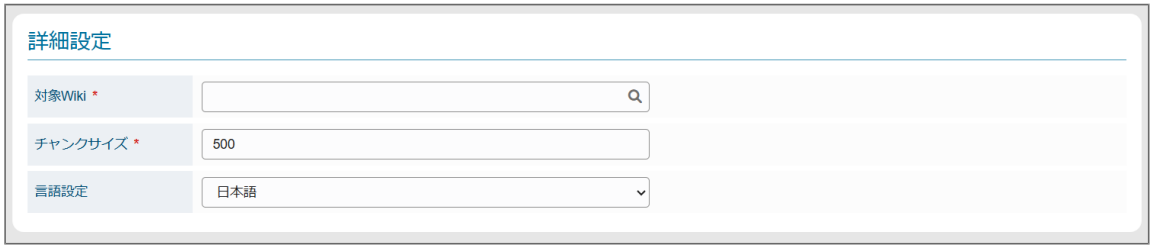
| 項目 | 説明 |
| --- | --- |
| 対象フロー | アシスタントを作成するロジックフローを指定します。<br>この項目は必須項目です。 |
| バージョン | アシスタントを作成するロジックフローのバージョンを設定します。<br>最新バージョンまたは利用するバージョンを指定します。 |

コラム

ロジックフローアシスタントについて

ロジックフローアシスタントの構築手順については以下を参照してください。
「*ロジックフローアシスタントの作成例*」

＜画面項目（詳細情報 - Wikiアシスタント）＞




| 項目 | 説明 |
| --- | --- |
| 対象Wiki | アシスタントを作成するWikiを指定します。<br>この項目は必須項目です。 |
| チャンクサイズ | コンテンツの内容をベクトルデータベースへ格納する際に行うテキスト分割の単位となる文字数を設定します。<br>設定可能なチャンクサイズの最小値は100 、最大値は4000 です。<br>この項目は必須項目です。 |
| 言語設定 | 対象のWiki コンテンツの言語を指定します。 |

コラム

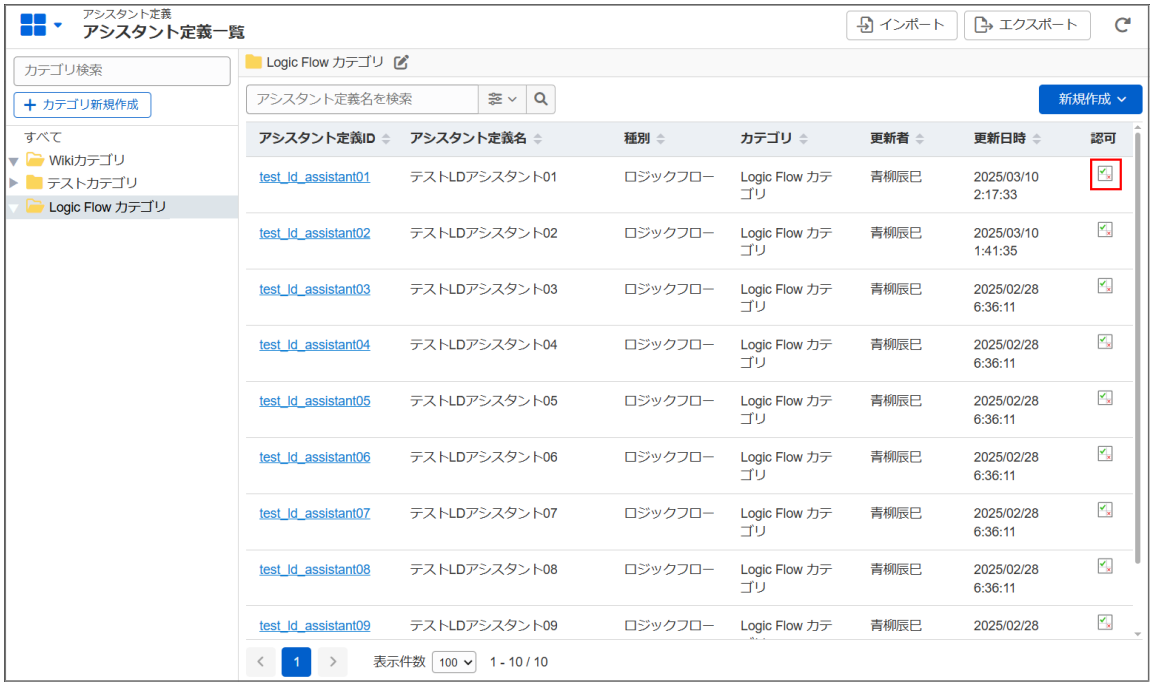
Wikiアシスタントについて

- IM-Jugglingプロジェクトに、IM-Wiki Copilotモジュールが含まれていない場合は、アシスタント定義種別リストでWikiアシスタントを選択できません。
- Wikiアシスタントは「 Knowledge 」のメニューの Wiki 画面でも作成できます。
  Wiki 画面での登録方法については、「IM-Knowledge管理者操作ガイド」 - 「Wiki アシスタント」を参照してください。
- 作成したWikiアシスタントの認可設定の初期状態は、対象Wikiのナレッジグループの認可設定が引き継がれています。

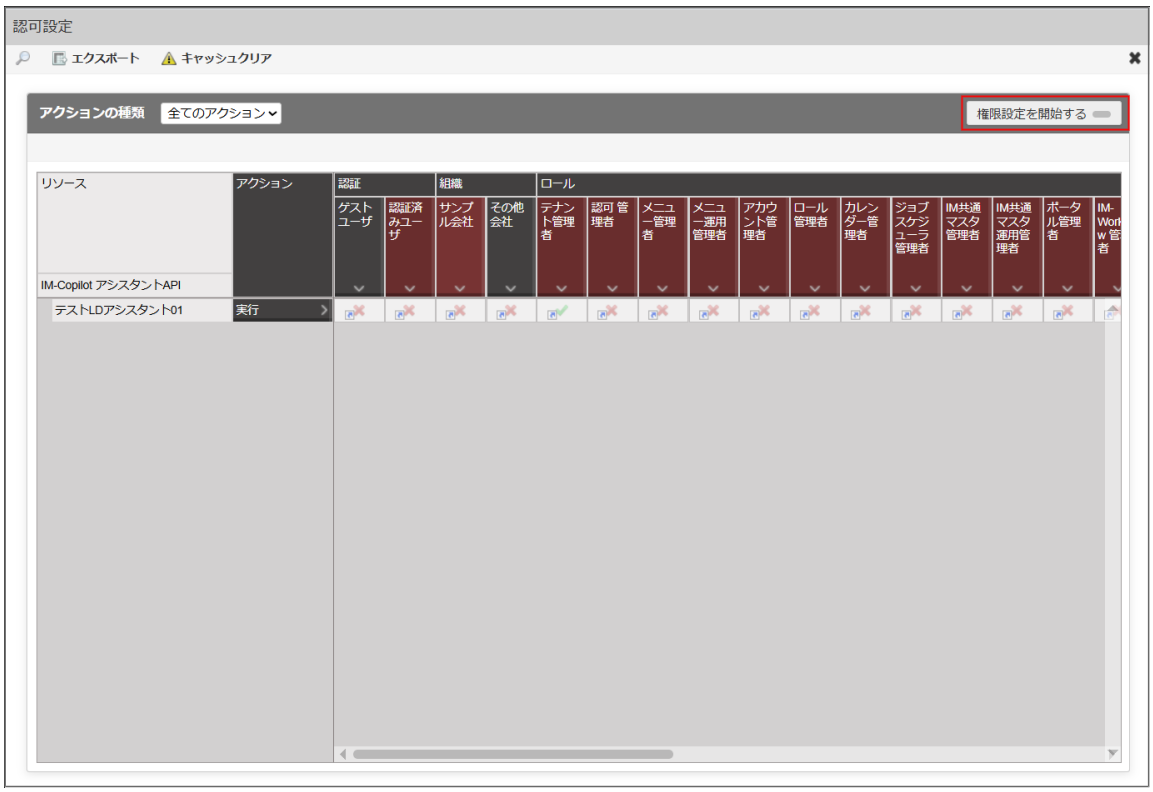
## アシスタント定義の認可設定をする

アシスタント定義の作成後は、必要に応じてアシスタント利用ユーザがアシスタントを利用するための認可設定を行います。

1. アシスタント定義一覧から認可設定を行うアシスタント定義の「認可設定」アイコンをクリックします。

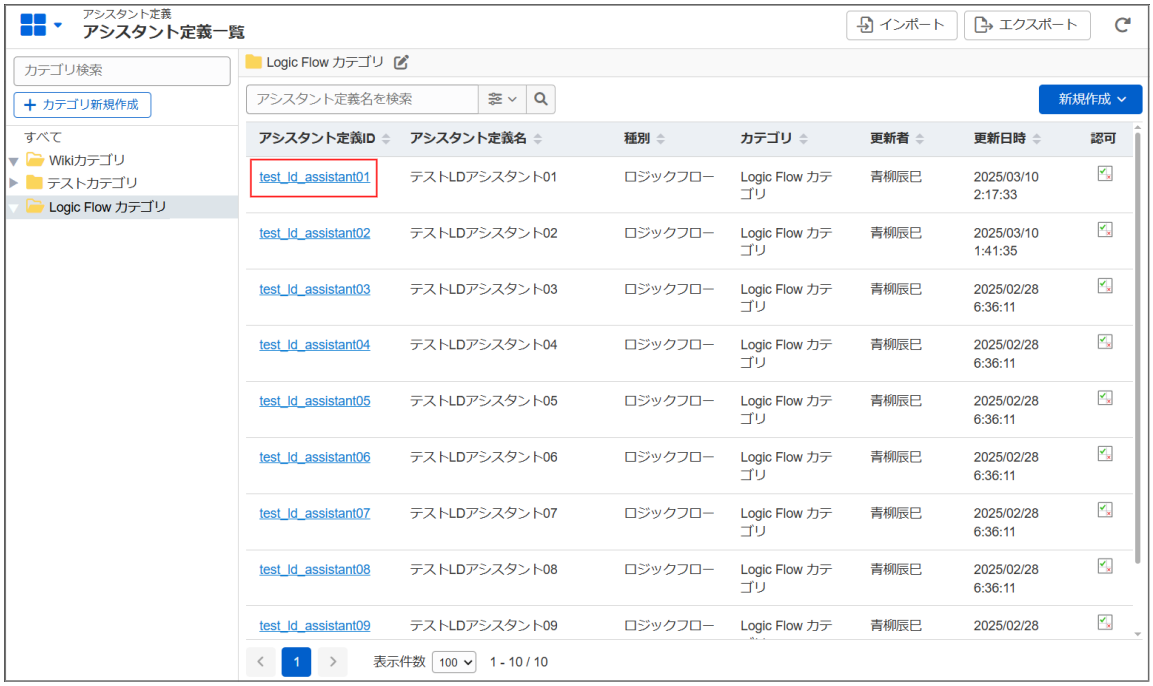



2. 別画面に「アシスタントの認可設定」画面が表示されます。「権限設定を開始する」をクリックし、認可設定を行います。
   認可設定画面の基本的な利用方法は「テナント管理者操作ガイド」 - 「認可を設定する」を参照してください。

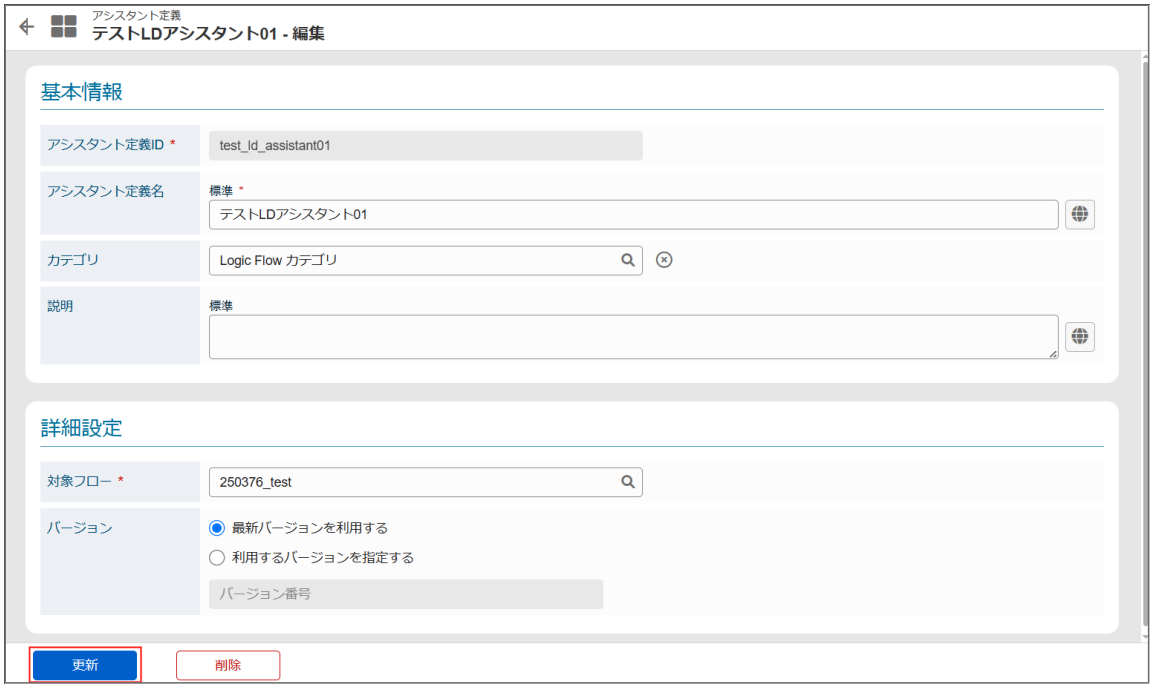



## アシスタント定義を編集する

1. 「*アシスタント定義一覧*」の手順をもとに、アシスタント定義一覧を表示します。
   一覧から編集するアシスタント定義のリンクをクリックします。

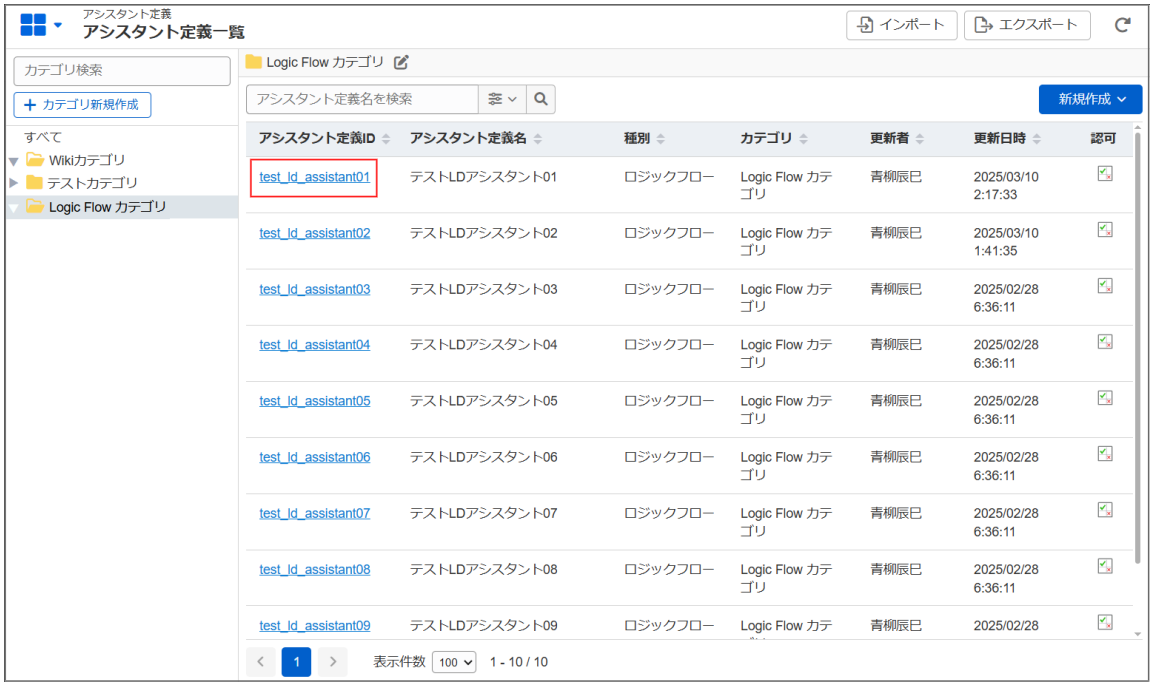



2. アシスタント情報を編集して「更新」ボタンをクリックします。確認ダイアログの「更新」ボタンをクリックします。

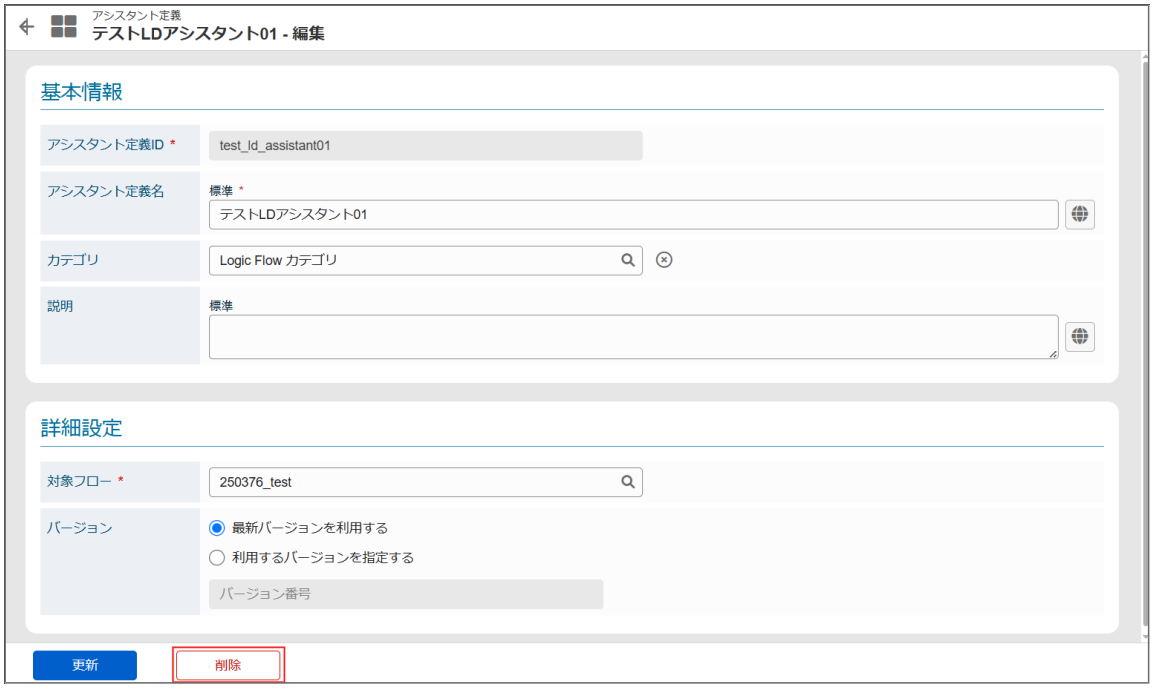



## アシスタント定義を削除する

1. 「*アシスタント定義一覧*」の手順をもとに、アシスタント定義一覧を表示します。
   一覧から削除するアシスタント定義のリンクをクリックします。

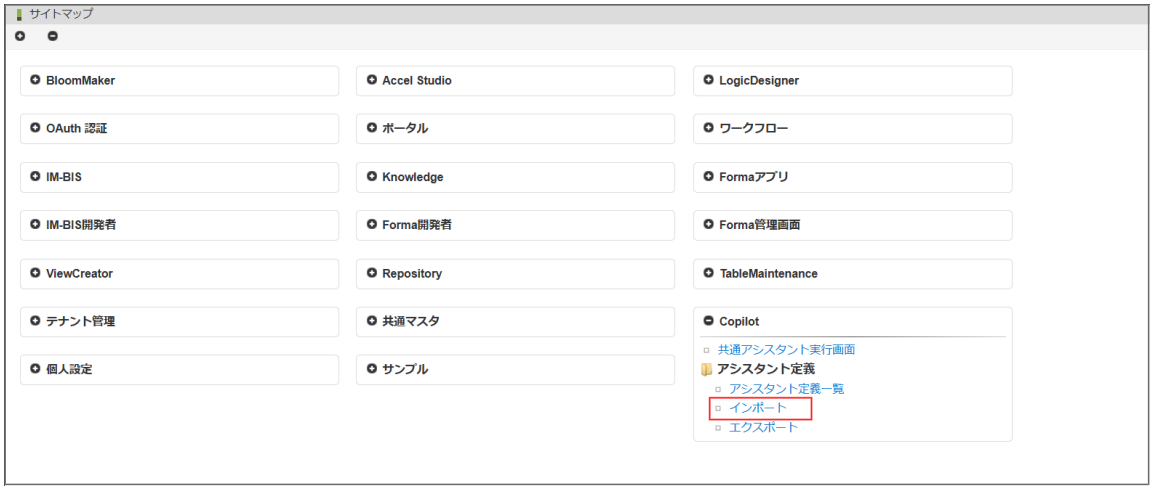



2. 「削除」ボタンをクリックします。確認ダイアログの「削除」ボタンをクリックします。




## アシスタント定義のインポート

1. 「サイトマップ」→「Copilot」→「アシスタント定義」→「インポート」をクリックし、「インポート」画面を表示します。

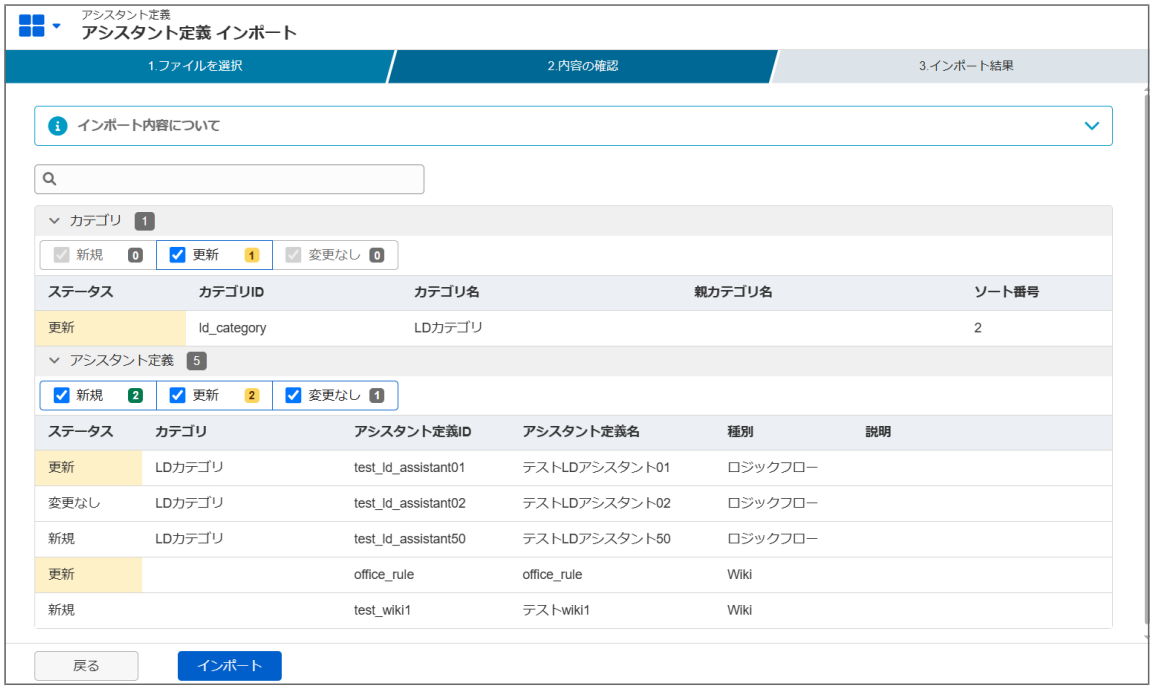



2. 「インポート」画面が表示されます。


3. インポートファイルをドラッグ＆ドロップまたは「ファイルを選択」ボタンをクリックしてファイルを選択します。
インポートできるファイル形式は、jsonのみです。


4. 「内容の確認へ」ボタンをクリックします。

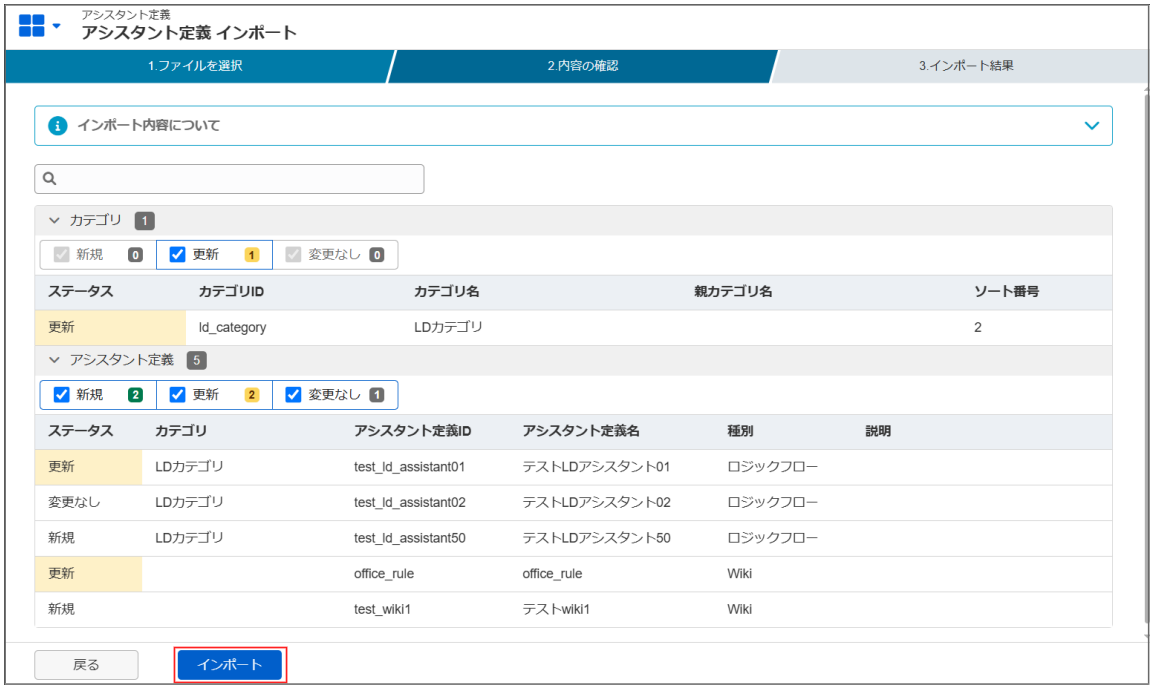

5. 「内容の確認」ステップが表示されます。
   インポートされるカテゴリ、アシスタント定義の情報が一覧で表示されます。
   それぞれのデータの状態に応じてステータスを表示します。
   - 新規：新規データ
   - 更新：既存データの更新
   - 変更なし：既存データに変更がない
     ※変更がないデータもインポートされ、更新者・更新日時が更新されます。




＜画面項目＞

| 項目 | 説明 |
|---|---|
| 検索窓 | 検索する文字列（の一部）を入力します。<br>一覧に表示されているカテゴリ・アシスタント定義のすべての項目に対して検索します。 |
| カテゴリ ステータスフィルタ | チェックボックスがONのステータスでカテゴリを絞り込みます。 |
| アシスタント定義 ステータスフィルタ | チェックボックスがONのステータスでアシスタント定義を絞り込みます。 |
| 「戻る」ボタン | 「ファイルを選択」ステップに遷移します。 |
| 「インポート」ボタン | インポートを実行します。 |

6. 「インポート」ボタンをクリックします。確認ダイアログの「インポート」ボタンをクリックします。




7. アシスタント定義のインポート結果が表示されます。


＜画面項目＞

| 項目 | 説明 |
|---|---|
| 「アシスタント定義一覧を開く」ボタン | 「アシスタント定義一覧」画面に遷移します。 |
| 「インポート画面に戻る」ボタン | 「インポート」画面に遷移します。 |
| 検索窓 | 検索する文字列（の一部）を入力します。<br>一覧に表示されているカテゴリ・アシスタント定義のすべての項目に対して検索します。 |
| 「失敗のみ表示」チェックボックス | インポートに失敗したデータがある場合に表示されます。<br>チェックボックスをONにするとインポートに失敗したカテゴリ・アシスタント定義のみが表示されます。 |

## アシスタント定義のエクスポート

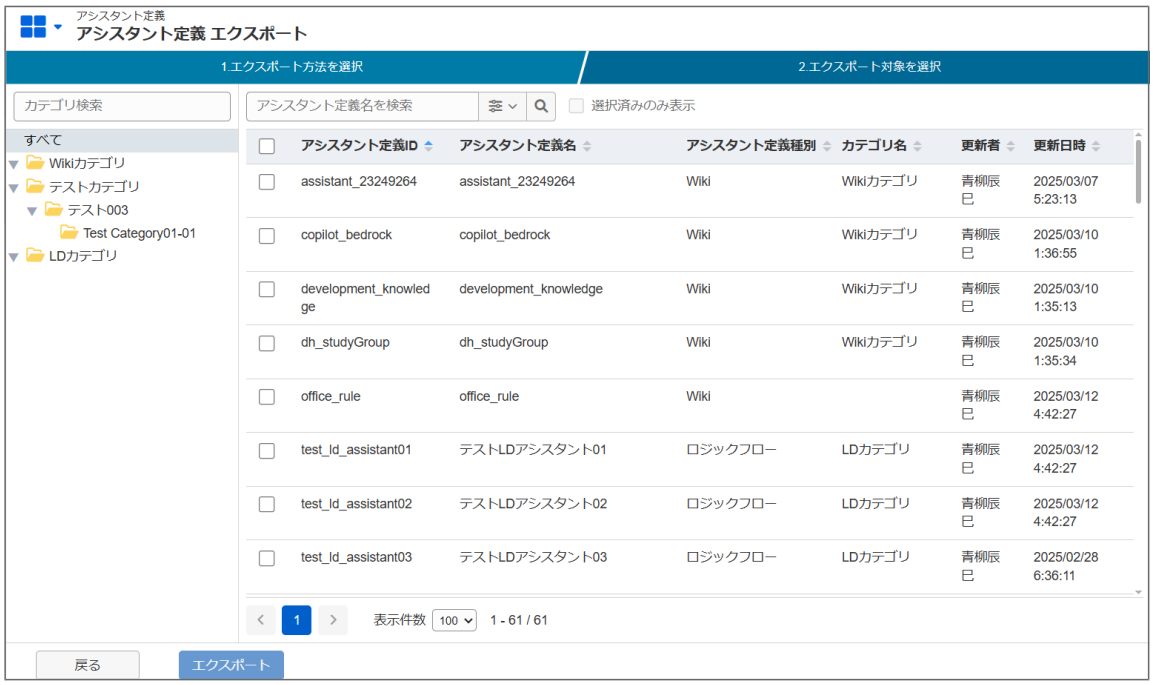
1. 「サイトマップ」→「Copilot」→「アシスタント定義」→「エクスポート」をクリックし、「エクスポート」画面を表示します。


2. 「エクスポート」画面が表示されます。


3. すべてのアシスタント定義をエクスポートする場合は、そのまま「エクスポート」ボタンをクリックします。


4. エクスポートするアシスタント定義を選択する場合は、「任意のアシスタント定義を選択してエクスポート」を選択して「次へ」ボタンをクリックします。

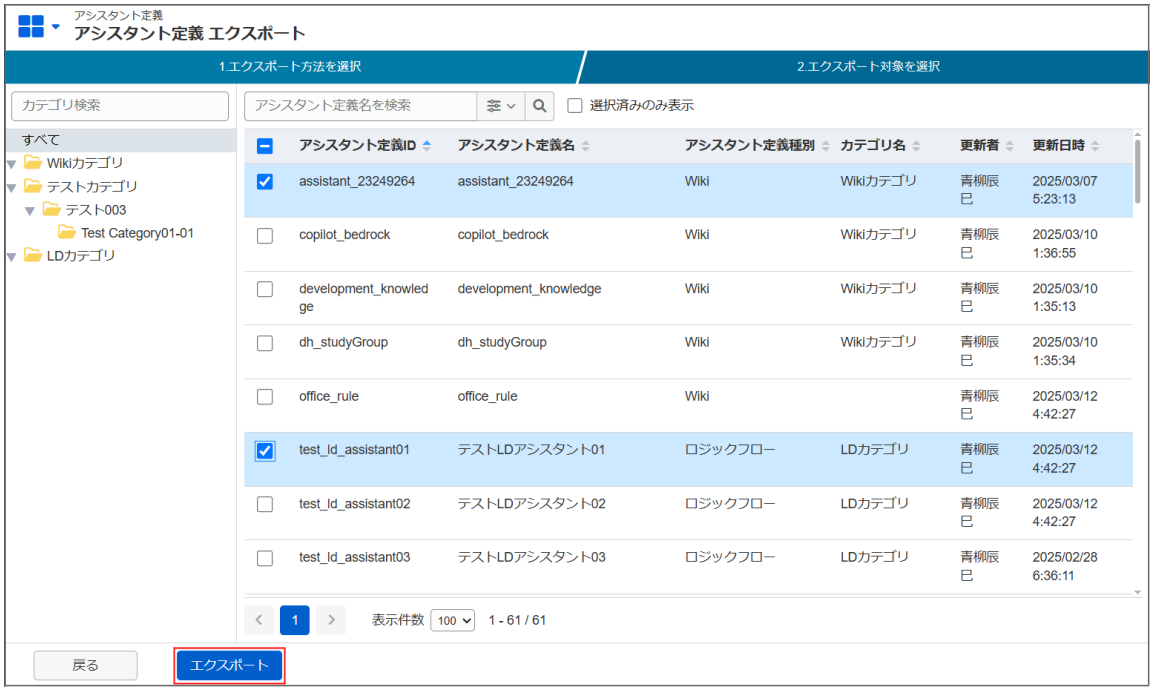

5. 「エクスポート対象を選択」ステップが表示されます。
   アシスタント定義一覧からエクスポートするアシスタント定義を選択します。




＜画面項目＞

| 項目 | 説明 |
|---|---|
| カテゴリ検索窓 | 検索するカテゴリ名を表す文字列（の一部）を入力します。 |
| アシスタント定義名検索窓 | 検索するアシスタント定義名を表す文字列（の一部）を入力します。 |
| 「詳細検索」アイコン | 検索するアシスタント定義ID、アシスタント定義名を表す文字列（の一部）、またはアシスタント定義種別を指定します。 |
| 「虫眼鏡」アイコン | 検索を実行します。 |
| 「選択済みのみ表示」チェックボックス | チェックボックスをONにすると選択したアシスタント定義のみが表示されます。 |
| 「ページャ」アイコン | 一覧の表示ページを切り替えます。 |
| 「表示件数」プルダウン | エンティティログの表示件数を設定します。<br>設定可能は表示件数は、50, 100, 150, 200 件です。 |
| 「戻る」ボタン | 「エクスポート方法の選択」ステップに遷移します。 |
| 「エクスポート」ボタン | エクスポートを実行します。 |

6. 「エクスポート」ボタンをクリックします。確認ダイアログの「エクスポート」ボタンをクリックします。




7. 「assistant_definition.json」ファイルがダウンロードされます。

# ログ

IM-Copilot では生成AIへのリクエストに成功した際の詳細をログとして出力させることが可能です。
出力内容を加工、集計することで特定のリクエストに関する統計情報の可視化を行うといった事が可能です。

## ログ設定

以下のログ設定ファイルを編集してください。

- %CONTEXT_PATH%/WEB-INF/conf/log/im_logger_copilot_driver.xml
- conf/log/im_logger_copilot_driver.xml （IM-Jugglingプロジェクト内の場合）

### ログ仕様

ログ仕様は「ログ仕様書」-「IM-Copilot生成AI連携ドライバログ」を参照してください。

### ファイル出力ログ

ファイルへのログ出力はデフォルトで有効化されています。（FileAppender）

### データベース出力ログ

データベースへのログ出力はデフォルトで有効化されていません。（DBAppender）
以下を参考に有効化してください。

#### 有効化

ログ設定を変更することでデータベースログを有効化できます。
また、データベース側にLogback仕様のテーブル、シーケンスなどを作成してください。

以下のサンプルを参考にしてください。

- ログ設定サンプル：
  - **im_logger_copilot_driver.xml（Logback_DBAppender_PostgreSQL）**
  - **im_logger_copilot_driver.xml（Logback_DBAppender_SQLServer）**
  - **im_logger_copilot_driver.xml（Logback_DBAppender_Oracle）**
- DDLサンプル：
  - **ddl_sample.sql（Logback_DDL_PostgreSQL）**
  - **ddl_sample.sql（Logback_DDL_SQLServer）**
  - **ddl_sample.sql（Logback_DDL_Oracle）**

> **コラム**
>
> 2024年3月末現在、上記サンプルとログ出力確認済のデータベースバージョンの組合せは以下の通りです。
> ・ PostgreSQL 13
> ・ SQLServer 2022
> ・ Oracle 19c

> **コラム**
>
> Logbackに関する不明点は、Logbackのドキュメント、フォーラムの情報などがヒントになる場合があります。
> https://logback.qos.ch/
>
> デバッグ可能環境の場合は、以下のLogbackソースをデバッグすることで不明点の解消が早まる場合があります。
> ch.qos.logback.core.db.DBAppenderBase.java など

# ログ活用

ログ可視化の例として、IM-LogicDesignerでログを読み込み、加工、ViewCreatorで表示する方法を説明します。

## ViewCreator表示

IM-LogicDesigner のロジックフローを ViewCreator の外部データソースとして利用できます。
詳細は「ViewCreator 管理者操作ガイド」-「ロジックフローの利用」を参照してください。

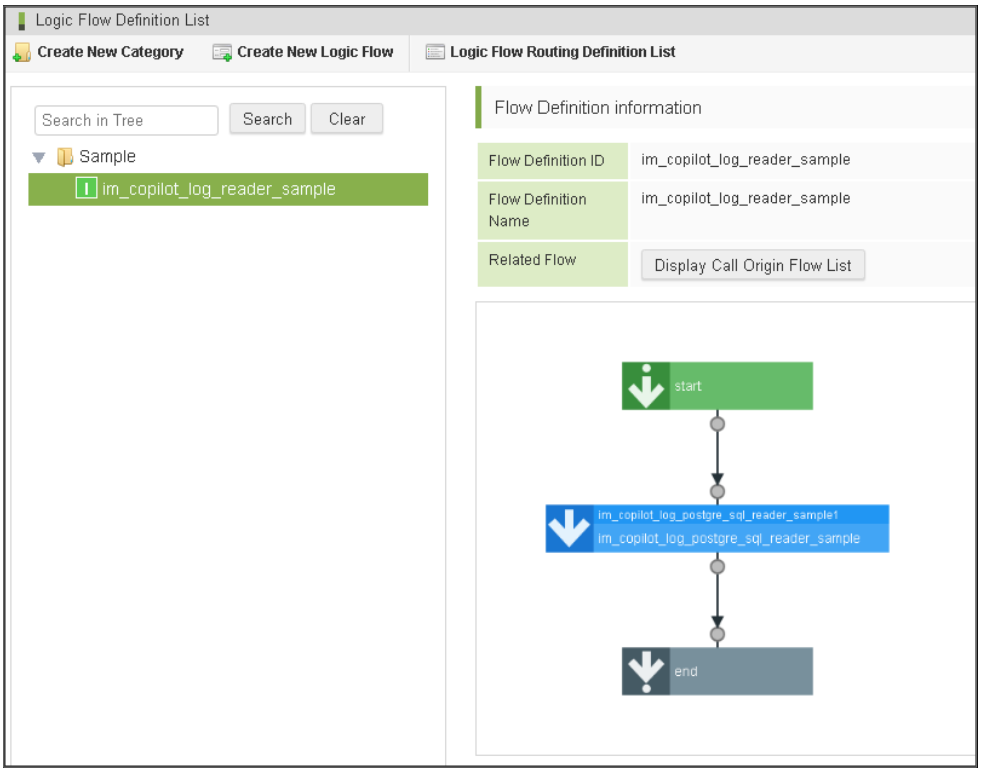
### ロジックフロー作成

IM-LogicDesigner でログを取得するロジックフローを作成してください。

ロジックフローには ViewCreatorで利用可能なロジックフローの出力設定が必要です。
詳細は「ViewCreator 管理者操作ガイド」-「ViewCreatorで利用可能なロジックフローの出力設定」を参照してください。

ログ取得方法などについては、以下のロジックフローサンプル（im_logicdesigner-data.zip）を参考にしてください。
ログファイルやデータベースからのログ取得、レスポンスのJSON文字列の変換処理、集計処理などを行っています。

- ロジックフローサンプル：
  **im_logicdesigner-data.zip（ログファイル）**
  **im_logicdesigner-data.zip（データベース：PostgreSQL）**
  **im_logicdesigner-data.zip（データベース：SQLServer）**
  **im_logicdesigner-data.zip（データベース：Oracle）**
  インポート方法は「IM-LogicDesigner ユーザ操作ガイド」-「インポート/エクスポート」を参照してください。




> **コラム**
>
> このサンプルでは、OpenAI および Azure OpenAI Service の /chat/completions エンドポイント（非ストリーミング）を対象に *total_tokens* を取得しています。
> 可視化のユースケースに応じて、取得対象のドライバやエンドポイント、ストリーミングモード、取得項目などを柔軟に調整してください。

### ロジックフロー管理

ViewCreator でロジックフロー管理を行ってください。
詳細は「ViewCreator 管理者操作ガイド」-「ロジックフロー管理」を参照してください。

### クエリ作成

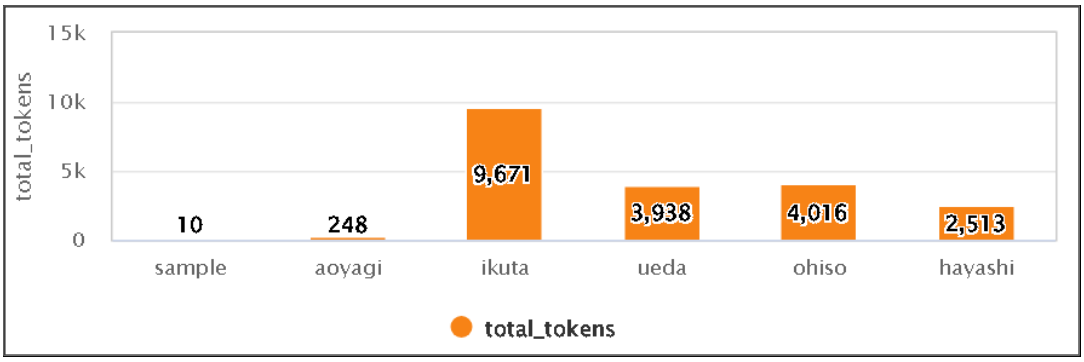
ViewCreator でクエリの作成を行ってください。
詳細は「ViewCreator 管理者操作ガイド」-「クエリの作成」を参照してください。

データ参照作成

ViewCreator でデータ参照の作成を行ってください。
詳細は「ViewCreator 管理者操作ガイド」-「データ参照の作成」を参照してください。

以下のデータ参照サンプル（viewcreator-data.zip）も参考にしてください。

- ViewCreator データ参照サンプル：
  **viewcreator-data.zip**
  インポート方法は「ViewCreator 管理者操作ガイド」-「インポート・エクスポート」を参照してください。



# 開発者向けガイド

## 開発者向け機能の概要

本章では、IM-Copilot を利用した開発者向け機能の概要について説明します。

### 開発フレームワーク別 対応機能一覧

IM-Copilot は、各開発フレームワークで以下の機能を提供しています：

**開発フレームワーク別 対応機能**

| 機能 | JavaEE開発モデル | スクリプト開発モデル | IM-LogicDesigner |
|---|---|---|---|
| チャット（ツール呼び出しを含まない） | ○ | ○ | ○ |
| チャット（ツール呼び出しを含む） | ○ | ○ | × |
| 埋め込み | ○ | ○ | ○ |
| 画像生成 | ○ | ○ | ○ |
| 音声認識 | ○ | ○ | ○ |
| 音声生成 | ○ | ○ | ○ |
| ベクトルデータベース操作 | ○ | × | ○ |

### 利用シーン

各フレームワークの特長を生かして、生成AIを柔軟に活用できます。

- IM-LogicDesigner：ローコード開発環境で生成AIを活用
- スクリプト開発モデル：JavaScriptを用いて手軽に生成AIを利用
- JavaEE開発モデル：豊富なAPIとの連携により、既存エンタープライズシステムとの統合や拡張が容易

異なる生成AIサービスの違いを意識することなく、統一されたインタフェースでの開発が可能です。

### 基本的な開発プロセス

1. 要件定義：
   生成AIを活用したい業務プロセスを明確にします。目的や期待される成果を整理し、AI導入の意義を明確化します。

2. 開発方式の選定：
   以下の方式から、業務要件や技術スタックに適したものを選択します：
   ・ IM-LogicDesigner
   ・ スクリプト開発モデル
   ・ JavaEE開発モデル

3. 実装：
   本ガイドのサンプルコードやAPIリファレンスを参考に、選定した方式に基づいて実装を行います。

4. テスト：
   生成AIの応答内容を確認し、必要に応じてプロンプト（AIへの指示文）を調整します。品質や一貫性の確保が重要です。

5. デプロイ：
   完成したアプリケーションを intra-mart Accel Platform上に展開し、運用環境での利用を開始します。

## Java API の利用方法

### 概要

IM-Copilot の Java API は、JavaEE開発モデルの開発環境で生成AIを活用するための包括的なインタフェースを提供します。

主なインタフェースと機能

主なインタフェースと機能

| インタフェース | 機能 | 詳細リンク |
| --- | --- | --- |
| ChatAction | チャットアクション用インタフェースです。ユーザとの対話を処理し、自然言語による応答を生成します。 | API - ChatAction |
| EmbeddingsAction | 埋め込みアクション用インタフェースです。テキストをベクトル（数値の並び）に変換し、意味的な類似性の計算等に利用します。 | API - EmbeddingsAction |
| ImageGenerationAction | 画像生成アクション用インタフェースです。テキストによる指示から画像を生成する機能を提供します。 | API - ImageGenerationAction |
| AudioSpeechAction | 音声生成アクション用インタフェースです。テキストを音声に変換し、読み上げ等に利用します。 | API - AudioSpeechAction |
| AudioTranscriptionAction | 文字起こしアクション用インタフェースです。音声データを解析して、対応するテキストを抽出します。 | API - AudioTranscriptionAction |
| VectorStore | ベクトルデータベース用インタフェースです。埋め込みベクトルを保存・検索し、類似検索や情報検索に活用します。 | API - VectorStore |

## 実装サンプル

実装サンプルについては、以下のセクションを参照してください：

### チャットサンプル

JavaEE開発モデルのAPI「ChatAction」を使った基本的な実装例を紹介します。

> **コラム**
>
> 「ChatAction」のAPI仕様は以下を参考にしてください：
> https://api.intra-mart.jp/iap/javadoc/all-dev_apidocs/jp/co/intra_mart/foundation/copilot/action/chat/ChatAction.html

> **コラム**
>
> このサンプルは「Web API Maker」を利用しています。
> ファクトリクラスおよびパッケージ登録ファイルの説明は割愛しています。
>
> 「Web API Maker」の詳細については「Web API Maker プログラミングガイド」を参考にしてください。

#### テキスト入力サンプル

非ストリーム形式の実装例

```
import java.util.List;
import com.fasterxml.jackson.databind.ObjectMapper;
import jp.co.intra_mart.foundation.copilot.action.ActionFactory;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatAction;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatMessage;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatOption;
import jp.co.intra_mart.foundation.web_api_maker.annotation.IMAuthentication;
import jp.co.intra_mart.foundation.web_api_maker.annotation.POST;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Parameter;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Path;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Required;

/**
 * チャット呼び出しのサンプルです。
 *
 * <pre>
 * 使用例:
 * POST /sample/chat
 * Body:
 * {
```

```
 *  "userPrompt": "最近の政治関連ニュースで、気になる話題はありますか？"
 * }
 * </pre>
 */
@IMAuthentication
public class ChatSample {

    private static final String SYSTEM_PROMPT = "この会話は、ビジネスパーソン同士の自然なやりとりとして応答してください。堅すぎず、親しみやすいトーンでお願いします。";

    /**
     * ユーザから受け取ったプロンプトに対して、チャット応答を生成して返却します。
     *
     * @param userPrompt ユーザからのプロンプト
     * @return チャット応答
     * @throws Exception 応答生成中にエラーが発生した場合
     */
    @Path("/sample/chat")
    @POST
    public String chat(@Required @Parameter(name = "userPrompt") String userPrompt) throws Exception {

        // チャットの履歴を含むメッセージ一覧を作成する
        // 最初に、アシスタントの振る舞いを定義する「システムプロンプト」を追加
        // その後、ユーザとアシスタントの過去の会話を順番に追加
        // 最後に、ユーザからの新しい質問（userPrompt）を追加して、会話の流れを構築する
        // @formatter:off
        final List<ChatMessage> messages = ChatMessage.builder()
            .newMessage().withRole("system").addTextContent(SYSTEM_PROMPT)
            .newMessage().withRole("user").addTextContent("最近の業界ニュースで気になることってありました？")
            .newMessage().withRole("assistant").addTextContent("そうですね、最近では新しい規制案の話題が注目されています。中小企業への影響が大きそうで、社内でも話題になっていました。")
            .newMessage().withRole("user").addTextContent("他に注目していることはありますか？")
            .newMessage().withRole("assistant").addTextContent("最近は人材の流動性が高まっているという話もよく聞きます。特に若手の転職が増えており、採用や育成の方針を見直す企業も増えているようです。")
            .newMessage().withRole("user").addTextContent(userPrompt)
            .build();
        // @formatter:on

        // チャット実行
        final ActionFactory factory = ActionFactory.getFactory();
        final ChatAction action = factory.getChatAction();
        final ChatOption option = ChatOption.builder().build();
        final List<ChatMessage> result = action.execute(messages, option);
        if (result != null && !result.isEmpty()) {
            // チャットメッセージのリストが存在し、空でない場合は、
            // 最後のメッセージを取得して JSON 形式の文字列に変換する
            final ChatMessage lastMessage = result.get(result.size() - 1);
            // コンテンツフィルタ: AI サービスの不適切コンテンツ検出機能
            // 入力違反時はAPI実行時にエラーが発生、出力違反時はcontentFilterプロパティで検出可能
            if (lastMessage.getContentFilter() != null && lastMessage.getContentFilter().isOutputBlocked()) {
                // 必要に応じて出力違反時の処理を記載
            }
            final String json = new ObjectMapper().writeValueAsString(lastMessage);
            return json;
        } else {
            // メッセージが存在しない場合は、空の JSON オブジェクトを返す
            return "{}";
        }
    }
}
```

ストリーム形式の実装例

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;
import javax.servlet.http.HttpServletResponse;
import com.fasterxml.jackson.databind.ObjectMapper;
import jp.co.intra_mart.foundation.copilot.action.ActionFactory;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatAction;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatMessage;
```

```
import jp.co.intra_mart.foundation.copilot.action.chat.ChatOption;
import jp.co.intra_mart.foundation.copilot.common.ContentFilterInfo;
import jp.co.intra_mart.foundation.web_api_maker.annotation.IMAuthentication;
import jp.co.intra_mart.foundation.web_api_maker.annotation.POST;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Parameter;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Path;
import jp.co.intra_mart.foundation.web_api_maker.annotation.PreventWritingResponse;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Required;

/**
 * チャットストリーミング呼び出しのサンプルです。
 *
 * <p>
 * Content-Type: text/event-stream を使用し、data: プレフィックス付きでチャンクデータを送信します。
 * </p>
 *
 * <pre>
 * 使用例:
 * POST /sample/chat/stream
 * Body:
 * {
 *   "userPrompt": "最近の政治関連ニュースで、気になる話題はありますか？"
 * }
 * </pre>
 */
@IMAuthentication
public class ChatStreamSample {

    private static final String SYSTEM_PROMPT = "この会話は、ビジネスパーソン同士の自然なやりとりとして応答してください。堅すぎず、親しみやすいトーンでお願いします。";

    /**
     * チャットのストリーミングで送信される1チャンク分のデータを表すクラス（模擬実装）
     */
    public class ChunkMessage {
        private String role;
        private String content;

        public ChunkMessage(String role, String content) {
            this.role = role;
            this.content = content;
        }

        public String getRole() {
            return role;
        }

        public String getContent() {
            return content;
        }

        public void setRole(String role) {
            this.role = role;
        }

        public void setContent(String content) {
            this.content = content;
        }
    }

    /**
     * クライアントにデータを送信するためのハンドラクラス（模擬実装）
     */
    public class StreamHandler {
        private static final String DATA_PREFIX = "data: ";
        private static final String EVENT_ERROR = "event: error";
        private static final String EVENT_CONTENT_FILTERED = "event: content_filtered";
        private static final String LINE_BREAK = "\n";
        private static final String DOUBLE_LINE_BREAK = "\n\n";

        private final PrintWriter writer;

        public StreamHandler(HttpServletResponse response) throws IOException {
```

```
        this.writer = response.getWriter();
    }

    /**
     * 通常のチャンクデータを送信
     */
    public void send(ChunkMessage chunk) {
        writer.write(DATA_PREFIX + chunk.getContent() + DOUBLE_LINE_BREAK);
        writer.flush();
    }

    /**
     * エラーイベントを送信
     */
    public void sendError(String errorMessage) {
        writer.write(EVENT_ERROR + LINE_BREAK);
        writer.write(DATA_PREFIX + errorMessage + DOUBLE_LINE_BREAK);
        writer.flush();
    }

    /**
     * コンテンツフィルタ情報を送信
     */
    public void sendContentFilter(final ContentFilterInfo contentFilter) throws IOException {
        writer.write(EVENT_CONTENT_FILTERED + LINE_BREAK);
        writer.write(DATA_PREFIX + new ObjectMapper().writeValueAsString(contentFilter) + DOUBLE_LINE_BREAK);
        writer.flush();
    }
}

/**
 * クライアントからのプロンプトを受け取り、チャット応答を1文字ずつストリーミング送信します。
 *
 * @param userPrompt クライアントから送信されたプロンプト（例：「議事録をまとめてください」）
 * @param response   HTTPレスポンスオブジェクト。SSE形式で出力されます。
 * @throws Exception 応答生成中にエラーが発生した場合
 */
@Path("/sample/chat/stream")
@POST
@PreventWritingResponse
public void chatStream(@Required @Parameter(name = "userPrompt") String userPrompt, final HttpServletResponse
response) throws Exception {

    // レスポンス設定
    response.setContentType("text/event-stream");
    response.setCharacterEncoding("UTF-8");

    // チャットの履歴を含むメッセージ一覧を作成する
    // 最初に、アシスタントの振る舞いを定義する「システムプロンプト」を追加
    // その後、ユーザとアシスタントの過去の会話を順番に追加
    // 最後に、ユーザからの新しい質問（userPrompt）を追加して、会話の流れを構築する
    // @formatter:off
    final List<ChatMessage> messages = ChatMessage.builder()
        .newMessage().withRole("system").addTextContent(SYSTEM_PROMPT)
        .newMessage().withRole("user").addTextContent("最近の業界ニュースで気になることってありました？")
        .newMessage().withRole("assistant").addTextContent("そうですね、最近では新しい規制案の話題が注目されています。中小企業への影
響が大きそうで、社内でも話題になっていました。")
        .newMessage().withRole("user").addTextContent("他に注目していることはありますか？")
        .newMessage().withRole("assistant").addTextContent("最近は人材の流動性が高まっているという話もよく聞きます。特に若手の転職が
増えており、採用や育成の方針を見直す企業も増えているようです。")
        .newMessage().withRole("user").addTextContent(userPrompt)
        .build();
    // @formatter:on

    // クライアントにデータを送信するためのハンドラ
    StreamHandler handler = new StreamHandler(response);

    // チャンクデータの初期化
    final ChunkMessage chunkMessage = new ChunkMessage(null, "");

    // チャット実行（非同期でチャンクを受け取る）
    final ActionFactory factory = ActionFactory.getFactory();
    final ChatAction action = factory.getChatAction();
```

```
        final ChatOption option = ChatOption.builder().build();
        action.execute(messages, option, chunk -> {
            try {
                // コンテンツフィルタ: AIサービスの不適切コンテンツ検出機能
                // 入力違反時は通常エラーが発生(一部AIサービス：Amazon Bedrock ではchunk.getContentFilter() で検出)
                // 出力違反時はchunk.getContentFilter() で検出可能
                if (chunk.getContentFilter() != null) {
                    // クライアントにフィルタリング情報を送信
                    handler.sendContentFilter(chunk.getContentFilter());
                    return;
                }
                if (null == chunk.getDelta().getContent()) {
                    return;
                }
                // チャンクデータを更新
                chunkMessage.setContent(chunk.getDelta().getContent());
                // クライアントに送信
                handler.send(chunkMessage);
            } catch (Exception e) {
                // エラー発生時にエラーメッセージを送信
                handler.sendError("Error sending chunk: " + e.getMessage());
            }
        });
    }
}
```

画像入力サンプル

画像付きの実装例

```
import java.util.List;
import com.fasterxml.jackson.databind.ObjectMapper;
import jp.co.intra_mart.foundation.copilot.action.ActionFactory;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatAction;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatMessage;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatOption;
import jp.co.intra_mart.foundation.service.client.file.PublicStorage;
import jp.co.intra_mart.foundation.web_api_maker.annotation.POST;
import jp.co.intra_mart.foundation.web_api_maker.annotation.IMAuthentication;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Parameter;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Path;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Required;

/**
 * 画像を使用したチャット呼び出しのサンプルです。
 *
 * <pre>
 * 使用例:
 * POST /sample/analyze/city-crowd
 * Body:
 * {
 *   "location": "tokyo-station"
 * }
 * </pre>
 *
 */
@IMAuthentication
public class ChatImageSample {

    private static final String SYSTEM_PROMPT = "あなたは都市の監視カメラ画像を分析し、混雑状況を自然な日本語で説明するAIです。指定された画像をもとに、人の数や密度から混雑度を4段階で評価し、簡潔に理由とアドバイスを伝えてください。個人が特定できる情報には触れず、匿名性を保ってください。";

    /**
     * 指定された場所情報に基づいて、ストレージ上の画像を取得・分析し、その結果に応じたチャット応答を生成して返却します。
     *
     * @param location 場所
     * @return チャット応答
     * @throws Exception 応答生成中にエラーが発生した場合
     */
    @Path("/sample/analyze/city-crowd")
```

```
    @Path("/sample/analyze/city-crowd")
    @POST
    public String chat(@Required @Parameter(name = "location") String location) throws Exception {

        // tokyo-station.png
        final PublicStorage publicStorage = new PublicStorage(location.concat(".png"));
        if (!publicStorage.isFile()) {
            // 画像ファイルがストレージに存在しない場合は、空の JSON オブジェクトを返す
            return "{}";
        }

        // @formatter:off
        final List<ChatMessage> messages = ChatMessage.builder().
            newMessage().withRole("system").addTextContent(SYSTEM_PROMPT).
            newMessage().withRole("user").addTextContent("次の画像を分析してください。").addImageData(publicStorage.load()).
            build();
        // @formatter:on

        // チャット実行
        final ActionFactory factory = ActionFactory.getFactory();
        final ChatAction action = factory.getChatAction();
        final ChatOption option = ChatOption.builder().build();
        final List<ChatMessage> result = action.execute(messages, option);
        if (result != null && !result.isEmpty()) {
            // チャットメッセージのリストが存在し、空でない場合は、
            // 最後のメッセージを取得して JSON 形式の文字列に変換する
            final ChatMessage lastMessage = result.get(result.size() - 1);
            final String json = new ObjectMapper().writeValueAsString(lastMessage);
            return json;
        } else {
            // メッセージが存在しない場合は、空の JSON オブジェクトを返す
            return "{}";
        }
    }
}
```

ツール呼び出しサンプル

ツール付きの実装例

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import com.fasterxml.jackson.databind.ObjectMapper;
import jp.co.intra_mart.foundation.copilot.action.ActionFactory;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatAction;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatMessage;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatMessage.ChatMessageBuilder;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatOption;
import jp.co.intra_mart.foundation.copilot.action.chat.ToolCall;
import jp.co.intra_mart.foundation.copilot.action.chat.ToolChoice;
import jp.co.intra_mart.foundation.copilot.action.chat.ToolConfig;
import jp.co.intra_mart.foundation.copilot.action.chat.ToolDefinition;
import jp.co.intra_mart.foundation.copilot.tool.ToolJsonHelper;
import jp.co.intra_mart.foundation.copilot.tool.JsonSchemaValidator;
import jp.co.intra_mart.foundation.copilot.tool.SchemaValidationError;
import jp.co.intra_mart.foundation.copilot.tool.annotation.SchemaProperty;
import jp.co.intra_mart.foundation.web_api_maker.annotation.IMAuthentication;
import jp.co.intra_mart.foundation.web_api_maker.annotation.POST;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Parameter;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Path;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Required;

/**
 * チャットツール呼び出しのサンプルです。
 *
 * <pre>
 * 使用例:
 * POST /sample/chat/tool
 * Body:
 * {
 *   "userPrompt": "商品コード「P-001」の在庫数を教えてください。"
```

```
 * }
 * </pre>
 */
@IMAuthentication
public class ChatToolSample {

    /**
     * 在庫確認ツールのパラメータクラス
     */
    public static class InventoryCheckParams {

        @StringProperty(description = "商品コード", required = true)
        private String productCode;

        @StringProperty(description = "倉庫コード（省略時は全倉庫）", required = false)
        private String warehouseCode;

        /**
         * 商品コードを取得します。
         * @return 商品コード
         */
        public String getProductCode() {
            return productCode;
        }

        /**
         * 倉庫コードを取得します。
         * @return 倉庫コード（null または空の場合は全倉庫対象）
         */
        public String getWarehouseCode() {
            return warehouseCode;
        }

        /**
         * 商品コードを設定します。
         * @param productCode 商品コード（必須）
         */
        public void setProductCode(final String productCode) {
            this.productCode = productCode;
        }

        /**
         * 倉庫コードを設定します。
         * @param warehouseCode 倉庫コード（省略可能。null または空の場合は全倉庫対象）
         */
        public void setWarehouseCode(final String warehouseCode) {
            this.warehouseCode = warehouseCode;
        }
    }

    /**
     * 在庫確認の実行（模擬実装）
     */
    private static String executeInventoryCheck(final ToolCall toolCall) {
        try {
            // JSONスキーマバリデーションの実行
            final ToolDefinition inventoryTool = ToolDefinition.builder().name("check_inventory").description("商品の在庫数を確認しま
す").parametersFromClass(InventoryCheckParams.class).build();

            final JsonSchemaValidator validator = new JsonSchemaValidator(inventoryTool.getParameters());
            final List<SchemaValidationError> errors = validator.validate(toolCall.getArguments());

            if (!errors.isEmpty()) {
                // バリデーションエラーがある場合
                final StringBuilder errorMessage = new StringBuilder("バリデーションエラー:");
                for (final SchemaValidationError error : errors) {
                    errorMessage.append("\n- ").append(error.getPropertyName()).append(": ").append(error.getMessage());
                }
                return errorMessage.toString();
            }

            // ToolJsonHelper を使用してJSON 引数をパース
            final InventoryCheckParams params = ToolJsonHelper.deserialize(toolCall.getArguments(), InventoryCheckParams.class);
```

```
        // 実際の在庫確認処理（ここでは固定値を返す）
        final int stockQuantity = 150; // 実際はデータベースなどから取得

        return String.format("商品コード %s の在庫数: %d 個", params.getProductCode(), stockQuantity);
      } catch (final Exception e) {
        return "在庫確認エラー: " + e.getMessage();
      }
    }

    /**
     * ユーザから受け取ったプロンプトに対して、チャット応答を生成して返却します。
     * @param userPrompt ユーザからのプロンプト
     * @return チャット応答
     * @throws Exception 応答生成中にエラーが発生した場合
     */
    @Path("/sample/chat/tool")
    @POST
    public String chat(@Required @Parameter(name = "userPrompt") final String userPrompt) throws Exception {

      // 1. ChatAction の取得
      final ActionFactory factory = ActionFactory.getFactory();
      final ChatAction action = factory.getChatAction();

      // 2. ツール定義の作成
      final ToolDefinition inventoryTool = ToolDefinition.builder().name("check_inventory").description("商品の在庫数を確認します").parametersFromClass(InventoryCheckParams.class).build();

      // 3. ツール設定の作成
      final ToolConfig toolConfig = new ToolConfig(Arrays.asList(inventoryTool), ToolChoice.builder().auto().build());

      // 4. 初回メッセージの作成
      final List<ChatMessage> messages = new ArrayList<>();
      messages.addAll(ChatMessage.builder().newMessage().withRole("user").addTextContent(userPrompt).build());

      // 5. ChatOption の設定
      final ChatOption option = new ChatOption();
      option.setTemperature(0.0);

      // 6. AI の応答を取得（ツール呼び出しを含む）
      final List<ChatMessage> response = action.execute(messages, option, toolConfig);
      messages.addAll(response);

      // 7. ツール呼び出しの処理
      final ChatMessage lastMessage = response.get(response.size() - 1);
      if (lastMessage.getToolCalls() != null && !lastMessage.getToolCalls().isEmpty()) {

        // 新しいメッセージビルダーを作成、ツール結果は "tool" ロールで送信します
        final ChatMessageBuilder toolMessageBuilder = ChatMessage.builder().newMessage().withRole("tool");

        // 各ツール呼び出しに対して、結果を取得してメッセージに追加
        for (final ToolCall toolCall : lastMessage.getToolCalls()) {
          System.out.println("ツール呼び出し: " + toolCall.getName());
          System.out.println("引数: " + toolCall.getArguments());

          // 実際のツール実行（ここでは模擬的な在庫確認）
          final String result = executeInventoryCheck(toolCall);

          // 実行結果を toolCallId に紐づけて追加
          toolMessageBuilder.addToolTextContent(result, toolCall.getId());
        }

        // 複数のツール実行結果を1つのtoolメッセージにまとめて追加
        messages.addAll(toolMessageBuilder.build());

        // 8. ツール実行結果を踏まえた最終応答を取得
        final List<ChatMessage> finalResponse = action.execute(messages, option, toolConfig);

        // 最終応答を出力（最後のアシスタントメッセージのみ出力）
        if (finalResponse != null && !finalResponse.isEmpty()) {
          // チャットメッセージのリストが存在し、空でない場合は、
          // 最後のメッセージを取得して JSON 形式の文字列に変換する
          final ChatMessage lastChatMessage = finalResponse.get(finalResponse.size() - 1);
```

```
            final String json = new ObjectMapper().writeValueAsString(lastChatMessage);
            return json;
        }
    }

    // ツール呼び出しが存在しない場合は、空の JSON オブジェクトを返す
    return "{}";
  }
}
```

ストリーム形式の実装例

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import javax.servlet.http.HttpServletResponse;
import jp.co.intra_mart.foundation.copilot.action.ActionFactory;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatAction;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatMessage;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatMessage.ChatMessageBuilder;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatOption;
import jp.co.intra_mart.foundation.copilot.action.chat.ToolCall;
import jp.co.intra_mart.foundation.copilot.action.chat.ToolChoice;
import jp.co.intra_mart.foundation.copilot.action.chat.ToolConfig;
import jp.co.intra_mart.foundation.copilot.action.chat.ToolDefinition;
import jp.co.intra_mart.foundation.copilot.exception.CopilotServiceException;
import jp.co.intra_mart.foundation.copilot.tool.ToolJsonHelper;
import jp.co.intra_mart.foundation.copilot.tool.JsonSchemaValidator;
import jp.co.intra_mart.foundation.copilot.tool.SchemaValidationError;
import jp.co.intra_mart.foundation.copilot.tool.annotation.SchemaProperty;
import jp.co.intra_mart.foundation.web_api_maker.annotation.IMAuthentication;
import jp.co.intra_mart.foundation.web_api_maker.annotation.POST;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Parameter;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Path;
import jp.co.intra_mart.foundation.web_api_maker.annotation.PreventWritingResponse;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Required;

/**
 * チャットツール呼び出しのストリーミング版サンプルです。
 *
 * <pre>
 * 使用例:
 * POST /sample/chat/tool/stream
 * Body:
 * {
 *   "userPrompt": "商品コード「P-001」の在庫数を教えてください。"
 * }
 * </pre>
 */
@IMAuthentication
public class ChatToolStreamingSample {

  /**
   * 在庫確認ツールのパラメータクラス
   */
  public static class InventoryCheckParams {

    @StringProperty(description = "商品コード", required = true)
    private String productCode;

    @StringProperty(description = "倉庫コード（省略時は全倉庫）", required = false)
    private String warehouseCode;

    /**
     * 商品コードを取得します。
     * @return 商品コード
     */
    public String getProductCode() {
      return productCode;
    }
```

```
    /**
     * 倉庫コードを取得します。
     * @return 倉庫コード（null または空の場合は全倉庫対象）
     */
    public String getWarehouseCode() {
        return warehouseCode;
    }

    /**
     * 商品コードを設定します。
     * @param productCode 商品コード（必須）
     */
    public void setProductCode(final String productCode) {
        this.productCode = productCode;
    }

    /**
     * 倉庫コードを設定します。
     * @param warehouseCode 倉庫コード（省略可能。null または空の場合は全倉庫対象）
     */
    public void setWarehouseCode(final String warehouseCode) {
        this.warehouseCode = warehouseCode;
    }
}

/**
 * 在庫確認の実行（模擬実装）
 */
private static String executeInventoryCheck(final ToolCall toolCall) {
    try {
        // JSONスキーマバリデーションの実行
        final ToolDefinition inventoryTool = ToolDefinition.builder().name("check_inventory").description("商品の在庫数を確認します").parametersFromClass(InventoryCheckParams.class).build();

        final JsonSchemaValidator validator = new JsonSchemaValidator(inventoryTool.getParameters());
        final List<SchemaValidationError> errors = validator.validate(toolCall.getArguments());

        if (!errors.isEmpty()) {
            // バリデーションエラーがある場合
            final StringBuilder errorMessage = new StringBuilder("バリデーションエラー:");
            for (final SchemaValidationError error : errors) {
                errorMessage.append("\n- ").append(error.getPropertyName()).append(": ").append(error.getMessage());
            }
            return errorMessage.toString();
        }

        // ToolJsonHelper を使用して JSON 引数をパース
        final InventoryCheckParams params = ToolJsonHelper.deserialize(toolCall.getArguments(), InventoryCheckParams.class);

        // 実際の在庫確認処理（ここでは固定値を返す）
        final int stockQuantity = 150; // 実際はデータベースなどから取得

        return String.format("商品コード %s の在庫数: %d 個", params.getProductCode(), stockQuantity);
    } catch (final Exception e) {
        return "在庫確認エラー: " + e.getMessage();
    }
}

/**
 * ユーザから受け取ったプロンプトに対して、チャット応答をリアルタイムにストリーミング形式で返却します。
 * @param userPrompt ユーザからのプロンプト
 * @param response HTTPレスポンス
 * @throws Exception 応答生成中にエラーが発生した場合
 */
@Path("/sample/chat/tool/stream")
@POST
@PreventWritingResponse
public void chatStream(@Required @Parameter(name = "userPrompt") final String userPrompt, final HttpServletResponse response) throws Exception {

    // 1. ChatAction の取得
    final ActionFactory factory = ActionFactory.getFactory();
```

```
        final ActionFactory factory = ActionFactory.getFactory();
        final ChatAction action = factory.getChatAction();

        // 2. ツール定義の作成
        final ToolDefinition inventoryTool = ToolDefinition.builder().name("check_inventory").description("商品の在庫数を確認します").parametersFromClass(InventoryCheckParams.class).build();

        // 3. ツール設定の作成
        final ToolConfig toolConfig = new ToolConfig(Arrays.asList(inventoryTool), ToolChoice.builder().auto().build());

        // 4. 初回メッセージの作成
        final List<ChatMessage> messages = new ArrayList<>();
        messages.addAll(ChatMessage.builder().newMessage().withRole("user").addTextContent(userPrompt).build());

        // 5. ChatOption の設定
        final ChatOption option = new ChatOption();
        option.setTemperature(0.0);

        // 6. レスポンスの設定
        response.setContentType("text/event-stream");
        response.setCharacterEncoding("UTF-8");

        try (final PrintWriter writer = response.getWriter()) {

            // 7. 初回の通常実行（ツール呼び出しチェック用）
            final List<ChatMessage> firstResponse = action.execute(messages, option, toolConfig);
            messages.addAll(firstResponse);

            // 8. ツール呼び出しの確認と処理
            final ChatMessage lastMessage = firstResponse.get(firstResponse.size() - 1);
            if (lastMessage.getToolCalls() != null && !lastMessage.getToolCalls().isEmpty()) {

                // 9. ツール呼び出しがある場合の処理

                // 新しいメッセージビルダーを作成、ツール結果は "tool" ロールで送信します
                final ChatMessageBuilder toolMessageBuilder = ChatMessage.builder().newMessage().withRole("tool");

                // 各ツール呼び出しに対して、結果を取得してメッセージに追加
                for (final ToolCall toolCall : lastMessage.getToolCalls()) {
                    System.out.println("ツール呼び出し: " + toolCall.getName());
                    System.out.println("引数: " + toolCall.getArguments());

                    // 実際のツール実行（ここでは模擬的な在庫確認）
                    final String result = executeInventoryCheck(toolCall);

                    // 実行結果を toolCallId に紐づけて追加
                    toolMessageBuilder.addToolTextContent(result, toolCall.getId());
                }

                // 複数のツール実行結果を1つのtoolメッセージにまとめて追加
                messages.addAll(toolMessageBuilder.build());

                // 10. ツール実行結果を踏まえた最終応答をストリーミングで取得
                writer.write("data: {\"type\":\"tool_executed\"}\n\n");
                writer.flush();

                // ストリーミング実行
                action.execute(messages, option, toolConfig, chunk -> {
                    if (chunk != null && chunk.getDelta() != null && chunk.getDelta().getContent() != null) {
                        // ストリーミングデータをSSE形式で送信
                        writer.write("data: " + chunk.getDelta().getContent() + "\n\n");
                        writer.flush();
                    }
                });

            } else {
                // 11. ツール呼び出しがない場合は、そのままテキストレスポンスをストリーミング

                // 初回応答のコンテンツを取得してストリーミング送信
                if (lastMessage.getContent() != null) {
                    writer.write("data: " + lastMessage.getContent());
                    writer.flush();
                }
            }
```

```
        }

        // 12. ストリーミング終了のマーカー
        writer.write("data: [DONE]\n\n");
        writer.flush();

      } catch (final IOException e) {
        throw new CopilotServiceException("ストリーミング中にエラーが発生しました", e);
      }
    }
}
```

完全ストリーム形式の実装例

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.atomic.AtomicBoolean;
import javax.servlet.http.HttpServletResponse;
import jp.co.intra_mart.foundation.copilot.action.ActionFactory;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatAction;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatMessage;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatMessage.ChatMessageBuilder;
import jp.co.intra_mart.foundation.copilot.action.chat.ChatOption;
import jp.co.intra_mart.foundation.copilot.action.chat.ToolCall;
import jp.co.intra_mart.foundation.copilot.action.chat.ToolChoice;
import jp.co.intra_mart.foundation.copilot.action.chat.ToolConfig;
import jp.co.intra_mart.foundation.copilot.action.chat.ToolDefinition;
import jp.co.intra_mart.foundation.copilot.exception.CopilotServiceException;
import jp.co.intra_mart.foundation.copilot.tool.ToolJsonHelper;
import jp.co.intra_mart.foundation.copilot.tool.JsonSchemaValidator;
import jp.co.intra_mart.foundation.copilot.tool.SchemaValidationError;
import jp.co.intra_mart.foundation.copilot.tool.annotation.SchemaProperty;
import jp.co.intra_mart.foundation.web_api_maker.annotation.IMAuthentication;
import jp.co.intra_mart.foundation.web_api_maker.annotation.POST;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Parameter;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Path;
import jp.co.intra_mart.foundation.web_api_maker.annotation.PreventWritingResponse;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Required;

/**
 * チャットツール呼び出しの完全ストリーミング版サンプルです。 ツール呼び出しも含めて全てストリーミングで処理します。
 *
 * <pre>
 * 使用例:
 * POST /sample/chat/tool/stream/dual
 * Body:
 * {
 *  "userPrompt": "商品コード「P-001」の在庫数を教えてください。"
 * }
 * </pre>
 */
@IMAuthentication
public class ChatToolDualStreamingSample {

  /**
   * 在庫確認ツールのパラメータクラス
   */
  public static class InventoryCheckParams {

    @StringProperty(description = "商品コード", required = true)
    private String productCode;

    @StringProperty(description = "倉庫コード（省略時は全倉庫）", required = false)
    private String warehouseCode;

    /**
     * 商品コードを取得します。
     * @return 商品コード
     */
    public String getProductCode() {
```

```
        public String getProductCode() {
            return productCode;
        }

        /**
         * 倉庫コードを取得します。
         * @return 倉庫コード（null または空の場合は全倉庫対象）
         */
        public String getWarehouseCode() {
            return warehouseCode;
        }

        /**
         * 商品コードを設定します。
         * @param productCode 商品コード（必須）
         */
        public void setProductCode(final String productCode) {
            this.productCode = productCode;
        }

        /**
         * 倉庫コードを設定します。
         * @param warehouseCode 倉庫コード（省略可能。null または空の場合は全倉庫対象）
         */
        public void setWarehouseCode(final String warehouseCode) {
            this.warehouseCode = warehouseCode;
        }
    }

    /**
     * ストリーミング処理の状態を保持するクラス
     */
    private static class StreamingState {
        private final List<ToolCall> collectedToolCalls = new ArrayList<>();

        private final StringBuilder currentContent = new StringBuilder();

        private final AtomicBoolean hasToolCalls = new AtomicBoolean(false);

        private String currentToolCallId = null;

        private String currentToolName = null;

        private StringBuilder currentArguments = new StringBuilder();
    }

    /**
     * 在庫確認の実行（模擬実装）
     */
    private static String executeInventoryCheck(final ToolCall toolCall) {
        try {
            // JSONスキーマバリデーションの実行
            final ToolDefinition inventoryTool = ToolDefinition.builder().name("check_inventory").description("商品の在庫数を確認します").parametersFromClass(InventoryCheckParams.class).build();

            final JsonSchemaValidator validator = new JsonSchemaValidator(inventoryTool.getParameters());
            final List<SchemaValidationError> errors = validator.validate(toolCall.getArguments());

            if (!errors.isEmpty()) {
                // バリデーションエラーがある場合
                final StringBuilder errorMessage = new StringBuilder("バリデーションエラー:");
                for (final SchemaValidationError error : errors) {
                    errorMessage.append("\n- ").append(error.getPropertyName()).append(": ").append(error.getMessage());
                }
                return errorMessage.toString();
            }

            // ToolJsonHelper を使用して JSON 引数をパース
            final InventoryCheckParams params = ToolJsonHelper.deserialize(toolCall.getArguments(), InventoryCheckParams.class);

            // 実際の在庫確認処理（ここでは固定値を返す）
            final int stockQuantity = 150; // 実際はデータベースなどから取得

            return String.format("商品コード %s の在庫数: %d個", params.getProductCode(), stockQuantity);
```

```
            return String.format("商品コード %s の在庫数: %d 個", params.getProductCode(), stockQuantity);
        } catch (final Exception e) {
            return "在庫確認エラー: " + e.getMessage();
        }
    }

    /**
     * ユーザから受け取ったプロンプトに対して、チャット応答を完全ストリーミング形式で返却します。
     * @param userPrompt ユーザからのプロンプト
     * @param response HTTPレスポンス
     * @throws Exception 応答生成中にエラーが発生した場合
     */
    @Path("/sample/chat/tool/stream/dual")
    @POST
    @PreventWritingResponse
    public void chatStream(@Required @Parameter(name = "userPrompt") final String userPrompt, final HttpServletResponse response) throws Exception {

        // 1. ChatAction の取得
        final ActionFactory factory = ActionFactory.getFactory();
        final ChatAction action = factory.getChatAction();

        // 2. ツール定義の作成
        final ToolDefinition inventoryTool = ToolDefinition.builder().name("check_inventory").description("商品の在庫数を確認します").parametersFromClass(InventoryCheckParams.class).build();

        // 3. ツール設定の作成
        final ToolConfig toolConfig = new ToolConfig(Arrays.asList(inventoryTool), ToolChoice.builder().auto().build());

        // 4. 初回メッセージの作成
        final List<ChatMessage> messages = new ArrayList<>();
        messages.addAll(ChatMessage.builder().newMessage().withRole("user").addTextContent(userPrompt).build());

        // 5. ChatOption の設定
        final ChatOption option = new ChatOption();
        option.setTemperature(0.0);

        // 6. レスポンスの設定
        response.setContentType("text/event-stream");
        response.setCharacterEncoding("UTF-8");

        try (final PrintWriter writer = response.getWriter()) {

            // 7. ストリーミング開始通知
            writer.write("data: {\"type\":\"stream_start\"}\n\n");
            writer.flush();

            // 8. 再帰的なストリーミング実行
            executeStreamingWithTools(messages, action, option, toolConfig, writer);

            // 9. ストリーミング終了のマーカー
            writer.write("data: [DONE]\n\n");
            writer.flush();

        } catch (final IOException e) {
            throw new CopilotServiceException("ストリーミング中にエラーが発生しました", e);
        }
    }

    /**
     * 再帰的にストリーミング処理を実行
     * @throws Exception
     */
    private void executeStreamingWithTools(final List<ChatMessage> messages, final ChatAction action, final ChatOption option, final ToolConfig toolConfig, final PrintWriter writer) throws Exception {

        final StreamingState state = new StreamingState();

        // ストリーミング実行
        action.execute(messages, option, toolConfig, chunk -> {
            if (chunk == null || chunk.getDelta() == null) {
                return;
            }
```

```
        // テキストコンテンツの処理
        if (chunk.getDelta().getContent() != null) {
            state.currentContent.append(chunk.getDelta().getContent());
            writer.write("data: " + chunk.getDelta().getContent() + "\n\n");
            writer.flush();
        }

        // ツール呼び出しの処理
        if (chunk.getDelta().getToolCalls() != null && !chunk.getDelta().getToolCalls().isEmpty()) {
            state.hasToolCalls.set(true);

            for (final ToolCall toolCall : chunk.getDelta().getToolCalls()) {
                // 新しいツール呼び出しの開始
                if (toolCall.getId() != null) {
                    // 前のツール呼び出しがあれば完成させる
                    if (state.currentToolCallId != null) {
                        final ToolCall completedCall =
ToolCall.builder().id(state.currentToolCallId).name(state.currentToolName).arguments(state.currentArguments.toString()).build();
                        state.collectedToolCalls.add(completedCall);
                    }

                    // 新しいツール呼び出しの初期化
                    state.currentToolCallId = toolCall.getId();
                    state.currentToolName = toolCall.getName();
                    state.currentArguments = new StringBuilder();
                }

                // 引数の収集
                if (toolCall.getArguments() != null && !toolCall.getArguments().isEmpty()) {
                    state.currentArguments.append(toolCall.getArguments());
                }
            }
        }
    });

    // 最後のツール呼び出しを完成させる
    if (state.currentToolCallId != null) {
        final ToolCall completedCall =
ToolCall.builder().id(state.currentToolCallId).name(state.currentToolName).arguments(state.currentArguments.toString()).build();
        state.collectedToolCalls.add(completedCall);
    }

    // ツール呼び出しがある場合は処理して再帰実行
    if (state.hasToolCalls.get() && !state.collectedToolCalls.isEmpty()) {
        // ツール実行通知
        writer.write("data: {\"type\":\"tool_execution_start\"}\n\n");
        writer.flush();

        // アシスタントメッセージ（ツール呼び出し付き）を追加
        if (state.currentContent.length() > 0) {

messages.addAll(ChatMessage.builder().newMessage().withRole("assistant").addTextContent(state.currentContent.toString()).toolCalls(

        } else {
            messages.addAll(ChatMessage.builder().newMessage().withRole("assistant").toolCalls(state.collectedToolCalls).build());
        }

        // ツール実行
        final ChatMessageBuilder toolMessageBuilder = ChatMessage.builder().newMessage().withRole("tool");

        for (final ToolCall toolCall : state.collectedToolCalls) {
            System.out.println("ツール呼び出し: " + toolCall.getName());
            System.out.println("引数: " + toolCall.getArguments());

            // 実際のツール実行
            final String result = executeInventoryCheck(toolCall);

            // 実行結果を toolCallId に紐づけて追加
            toolMessageBuilder.addToolTextContent(result, toolCall.getId());

            // ツール実行結果を通知
            writer.write("data: {\"type\":\"tool_executed\",\"tool\":\"" + toolCall.getName() + "\",\"result\":\"" + result.replace("\"", "\\\"")
```

```
+ "\"}\n\n");
            writer.flush();
        }

        // ツール実行結果をメッセージに追加
        messages.addAll(toolMessageBuilder.build());

        // 再帰的にストリーミング実行（ツール実行後の応答生成）
        writer.write("data: {\"type\":\"tool_response_start\"}\n\n");
        writer.flush();

        executeStreamingWithTools(messages, action, option, toolConfig, writer);
    }
  }
}
```

埋め込みサンプル

JavaEE開発モデルのAPI「EmbeddingsAction」を使った基本的な実装例を紹介します。

> **コラム**
>
> 「EmbeddingsAction」のAPI仕様は以下を参考にしてください：
> https://api.intra-mart.jp/iap/javadoc/all-dev_apidocs/jp/co/intra_mart/foundation/copilot/action/embeddings/EmbeddingsAction.html

埋め込みサンプル

```
import jp.co.intra_mart.foundation.copilot.action.ActionFactory;
import jp.co.intra_mart.foundation.copilot.action.embeddings.EmbeddingsAction;
import jp.co.intra_mart.foundation.job_scheduler.Job;
import jp.co.intra_mart.foundation.job_scheduler.JobResult;
import jp.co.intra_mart.foundation.job_scheduler.exception.JobExecuteException;

/**
 * 埋め込みサンプルです。
 */
public class EmbeddingSample implements Job {
  @Override
  public JobResult execute() throws JobExecuteException {
    try {
      final ActionFactory factory = ActionFactory.getFactory();
      final EmbeddingsAction action = factory.getEmbeddingsAction();

      final String embeddingSampleText = "私は、以下の理由により申請を行います。第一に、業務上必要な資格取得のためです。第二に、自己研鑽を目的としております。以上、よろしくお願いいたします。";

      final float[] embeddings = action.execute(embeddingSampleText);

      // 後続処理割愛：埋め込み結果（embeddings）をベクトルDBに保存など

    } catch (final Exception e) {
      throw new JobExecuteException(e);
    }
    return JobResult.success("ok");
  }
}
```

画像生成サンプル

JavaEE開発モデルのAPI「ImageGenerationAction」を使った基本的な実装例を紹介します。

> **コラム**
>
> 「ImageGenerationAction」のAPI仕様は以下を参考にしてください：
> https://api.intra-mart.jp/iap/javadoc/all-dev_apidocs/jp/co/intra_mart/foundation/copilot/action/images/ImageGenerationAction.html

画像生成サンプル

```
import java.io.OutputStream;
import jp.co.intra_mart.foundation.copilot.action.ActionFactory;
import jp.co.intra_mart.foundation.copilot.action.images.ImageGenerationAction;
import jp.co.intra_mart.foundation.copilot.action.images.ImageGenerationOption;
import jp.co.intra_mart.foundation.job_scheduler.Job;
import jp.co.intra_mart.foundation.job_scheduler.JobResult;
import jp.co.intra_mart.foundation.job_scheduler.exception.JobExecuteException;
import jp.co.intra_mart.foundation.service.client.file.PublicStorage;

/**
 * 画像生成サンプルです。
 */
public class ImageSample implements Job {

  @Override
  public JobResult execute() throws JobExecuteException {
    try {
      final ActionFactory factory = ActionFactory.getFactory();
      final ImageGenerationAction action = factory.getImageGenerationAction();
      final PublicStorage publicStorage = new PublicStorage("BusinessTrendVisual.png");
      final ImageGenerationOption option = ImageGenerationOption.builder().build();

      //@formatter:off
      final String businessTrendVisualText = "白背景に、売上成長を象徴する挿絵スタイルのビジュアル。"
          + "人物がパソコンで売上データを確認している様子、"
          + "上昇する矢印やコイン、ショッピングバッグ、"
          + "グラフの代わりに成長を示す植物などが描かれている。"
          + "ビジネスカジュアルな雰囲気で、柔らかい色合い（青・緑・ベージュ）を使用。"
          + "プレゼン資料やレポートの挿絵として使える、親しみやすくプロフェッショナルなデザイン。";
      //@formatter:on

      try (final OutputStream os = publicStorage.create()) {
        option.setOutput(os);
        action.execute(businessTrendVisualText, option);
      }
    } catch (final Exception e) {
      throw new JobExecuteException(e);
    }
    return JobResult.success("ok");
  }
}
```

音声生成サンプル

JavaEE開発モデルのAPI「AudioSpeechAction」を使った基本的な実装例を紹介します。

> **コラム**
>
> 「AudioSpeechAction」のAPI仕様は以下を参考にしてください：
> https://api.intra-mart.jp/iap/javadoc/all-dev_apidocs/jp/co/intra_mart/foundation/copilot/action/speech/AudioSpeechAction.html

> **コラム**
>
> このサンプルはジョブプログラムです。
>
> ジョブプログラムの詳細については「ジョブスケジューラ仕様書」を参考にしてください。

音声生成サンプル

```
import java.io.OutputStream;

import jp.co.intra_mart.foundation.copilot.action.ActionFactory;
import jp.co.intra_mart.foundation.copilot.action.speech.AudioSpeechAction;
import jp.co.intra_mart.foundation.job_scheduler.Job;
import jp.co.intra_mart.foundation.job_scheduler.JobResult;
import jp.co.intra_mart.foundation.job_scheduler.exception.JobExecuteException;
import jp.co.intra_mart.foundation.service.client.file.PublicStorage;

/**
 * 音声生成サンプルです。
 */
public class AudioSample implements Job {
   @Override
   public JobResult execute() throws JobExecuteException {
      try {
         final ActionFactory factory = ActionFactory.getFactory();
         final AudioSpeechAction action = factory.getAudioSpeechAction();
         final PublicStorage publicStorage = new PublicStorage("ProductOverviewAudio.mp3");
         final AudioSpeechOption option = AudioSpeechOption.builder().build();

         //@formatter:off
         final String productOverviewAudioText = "こんにちは。本日は、業務効率化を支援する新しいソリューションについてご紹介します。"
               + "このツールは、日々の業務をよりスムーズに進めるために設計されており、"
               + "直感的な操作性と柔軟なカスタマイズ性を兼ね備えています。"
               + "すでに多くの企業で導入が進んでおり、業務時間の短縮やチーム間の連携強化に貢献しています。"
               + "詳細については、担当者までお気軽にお問い合わせください。";
         //@formatter:on

         try (final OutputStream os = publicStorage.create()) {
            option.setOutput(os);
            action.execute(productOverviewAudioText, option);
         }
      } catch (final Exception e) {
         throw new JobExecuteException(e);
      }
      return JobResult.success("ok");
   }
}
```

文字起こしサンプル

JavaEE開発モデルのAPI「AudioTranscriptionAction」を使った基本的な実装例を紹介します。

> **コラム**
>
> 「AudioTranscriptionAction」のAPI仕様は以下を参考にしてください：
> https://api.intra-mart.jp/iap/javadoc/all-dev_apidocs/jp/co/intra_mart/foundation/copilot/action/transcription/AudioTranscriptionAction.html

> **コラム**
>
> このサンプルはジョブプログラムです。
>
> ジョブプログラムの詳細については「ジョブスケジューラ仕様書」を参考にしてください。

文字起こしサンプル

```
import java.io.InputStream;
import jp.co.intra_mart.foundation.copilot.action.ActionFactory;
import jp.co.intra_mart.foundation.copilot.action.transcription.AudioTranscriptionAction;
import jp.co.intra_mart.foundation.job_scheduler.Job;
import jp.co.intra_mart.foundation.job_scheduler.JobResult;
import jp.co.intra_mart.foundation.job_scheduler.exception.JobExecuteException;
import jp.co.intra_mart.foundation.service.client.file.PublicStorage;

/**
 * 文字起こしサンプルです。
 */
public class TranscriptionSample implements Job {
    @Override
    public JobResult execute() throws JobExecuteException {
        try {
            final PublicStorage publicStorage = new PublicStorage("MeetingSalesTeam.mp3");
            if (!publicStorage.isFile()) {
                return JobResult.success("ストレージに音声ファイルがありません。");
            }
            String transcriptionText = "";
            try (InputStream input = publicStorage.open()) {
                final ActionFactory factory = ActionFactory.getFactory();
                final AudioTranscriptionAction action = factory.getAudioTranscriptionAction();
                transcriptionText = action.execute(input);
            }

            // 後続処理割愛：文字起こし結果（transcriptionText）をDBに保存など

        } catch (final Exception e) {
            throw new JobExecuteException(e);
        }
        return JobResult.success("ok");
    }
}
```

ベクトルデータベース操作サンプル

JavaEE開発モデルのAPI「VectorStore」を使った基本的な実装例を紹介します。

> **コラム**
>
> 「VectorStore」のAPI仕様は以下を参考にしてください：
> https://api.intra-mart.jp/iap/javadoc/all-dev_apidocs/jp/co/intra_mart/foundation/copilot/vectorstore/VectorStore.html

> **コラム**
>
> このサンプルは「Web API Maker」を利用しています。
> ファクトリクラスおよびパッケージ登録ファイルの説明は割愛しています。
>
> 「Web API Maker」の詳細については「Web API Maker プログラミングガイド」を参考にしてください。

ベクトルデータベースコンテンツ登録、検索サンプル

```
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.node.ObjectNode;
import jp.co.intra_mart.foundation.admin.tenant.TenantInfoManager;
import jp.co.intra_mart.foundation.copilot.vectorstore.VectorStore;
import jp.co.intra_mart.foundation.copilot.vectorstore.builder.VectorStoreBuilder;
import jp.co.intra_mart.foundation.copilot.vectorstore.document.Document;
import jp.co.intra_mart.foundation.copilot.vectorstore.document.ScoredDocument;
import jp.co.intra_mart.foundation.copilot.vectorstore.embedder.Embedder;
import jp.co.intra_mart.foundation.service.client.file.PublicStorage;
import jp.co.intra_mart.foundation.web_api_maker.annotation.IMAuthentication;
```

```
import jp.co.intra_mart.foundation.web_api_maker.annotation.POST;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Parameter;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Path;
import jp.co.intra_mart.foundation.web_api_maker.annotation.Required;

/**
 * ベクトルデータベースコンテンツ登録、検索に関するサンプルです。
 *
 * <pre>
 * 使用例:
 * POST /sample/vector-store/add
 * Body:
 * {
 *   "file": "customer-review.txt"
 * }
 * </pre>
 *
 * <pre>
 * 使用例:
 * POST /sample/vector-store/search
 * Body:
 * {
 *   "query": "バッテリーの持ちについて書かれているレビューは？"
 * }
 * </pre>
 *
 */
@IMAuthentication
public class VectorDatabaseSample {

    private static final String VECTORSTORE_CATEGORY = "SAMPLE";

    /**
     * 指定されたファイル名に対応するテキストファイルを取得し、内容をベクトル化してベクトルデータベースに登録します。
     * @param file ストレージファイル名
     * @return 応答（JSON）
     * @throws Exception 応答生成中にエラーが発生した場合
     */
    @Path("/sample/vector-store/add")
    @POST
    public String add(@Required @Parameter(name = "file") String file) throws Exception {

        // customer-review.txt
        final PublicStorage publicStorage = new PublicStorage(file);
        if (!publicStorage.isFile()) {
            // 登録対象がストレージに存在しない場合は、空の JSON オブジェクトを返す
            return "{}";
        }

        // テキスト分割、Documentに変換
        final List<Document> documents = this.createDocuments(publicStorage.read(), file);

        // 埋め込み
        final Embedder embedder = new VectorDatabaseSampleEmbedder();

        // VectorStore
        final Locale locale = new TenantInfoManager().getTenantInfo().getLocale();
        final VectorStore vectorStore =
VectorStoreBuilder.builder().category(VECTORSTORE_CATEGORY).embedder(embedder).locale(locale).build();

        // カテゴリ単位で全ベクトルデータを削除
        vectorStore.deleteAll();

        // ※指定されたファイルに紐づくベクトルデータのみ削除する場合は以下を使用
        // vectorStore.deleteByOriginSourceId(file, false);

        // 登録
        vectorStore.add(documents);

        return "{ \"status\": \"success\"}";
    }

    /**
```

```
    * 指定された検索クエリに基づいて、ベクトルデータベースから類似情報を検索します。
    * @param query 検索クエリ
    * @return 応答（JSON）
    * @throws Exception 応答生成中にエラーが発生した場合
    */
    @Path("/sample/vector-store/search")
    @POST
    public String search(@Required @Parameter(name = "query") String query) throws Exception {

        // 埋め込み
        final Embedder embedder = new VectorDatabaseSampleEmbedder();

        // VectorStore
        final Locale locale = new TenantInfoManager().getTenantInfo().getLocale();
        final VectorStore store =
VectorStoreBuilder.builder().useDefault().category(VECTORSTORE_CATEGORY).embedder(embedder).locale(locale).build();

        // ハイブリッド検索
        final List<ScoredDocument> scoredDocument = store.hybridSearch(query);

        // JSON変換
        return this.buildResponse(scoredDocument);
    }

    private String buildResponse(final List<ScoredDocument> documents) {
        final ObjectMapper objectMapper = new ObjectMapper();
        try {
            // ベースのJSONオブジェクトを作成
            final ObjectNode root = objectMapper.createObjectNode();
            root.put("status", "success");
            // documentsリストを追加
            root.set("documents", objectMapper.valueToTree(documents));
            // JSON文字列に変換
            return objectMapper.writeValueAsString(root);
        } catch (final JsonProcessingException e) {
            throw new RuntimeException("JSON変換に失敗しました", e);
        }
    }

    private List<Document> createDocuments(final String text, final String filename) {
        final List<Document> documents = new ArrayList<>();
        // テキストを最大maxChunkSize文字ごとに分割
        final int maxChunkSize = 50;
        final List<String> chunks = this.splitText(text, maxChunkSize);
        // 分割された各チャンクをSampleDocumentに変換してリストに追加
        for (int i = 0; i < chunks.size(); i++) {
            final VectorDatabaseSampleDocument sampleDocument = new VectorDatabaseSampleDocument();
            // チャンクを取得してSampleDocumentに設定
            final String chunk = chunks.get(i);
            sampleDocument.setText(chunk);
            // オリジンソースIDを設定
            sampleDocument.setOriginSourceId(filename);
            // 作成したSampleDocumentをdocumentsリストに追加
            documents.add(sampleDocument);
        }
        // 作成されたDocumentのリストを返却
        return documents;
    }

    private List<String> splitText(final String text, final int maxChunkSize) {
        final List<String> chunks = new ArrayList<>();
        // 段落・行・単語のいずれかで分割（正規表現で複数の区切りを指定）
        final String[] parts = text.split("\\n\\n|\\n| ");
        final StringBuilder chunk = new StringBuilder();
        // 各パートを順にチャンクに追加していく
        for (final String part : parts) {
            // チャンクのサイズが最大を超える場合は、現在のチャンクを保存して新しく開始
            if (chunk.length() + part.length() + 1 > maxChunkSize) {
                chunks.add(chunk.toString().trim());
                chunk.setLength(0); // チャンクをリセット
            }
            // チャンクにスペースを追加（最初以外）
            if (chunk.length() > 0)
```

```
            if (chunk.length() > 0)
                chunk.append(" ");
            chunk.append(part);
        }
        // 最後のチャンクを追加
        if (chunk.length() > 0) {
            chunks.add(chunk.toString().trim());
        }
        return chunks;
    }
}
```

```
import java.util.ArrayList;
import java.util.List;
import jp.co.intra_mart.foundation.copilot.action.ActionFactory;
import jp.co.intra_mart.foundation.copilot.action.embeddings.EmbeddingsAction;
import jp.co.intra_mart.foundation.copilot.vectorstore.embedder.Embedder;

/**
 * Embedderインタフェースの実装クラス
 */
public class VectorDatabaseSampleEmbedder implements Embedder {

    @Override
    public List<Float> generate(final String content) {
        try {
            // 埋め込み
            final ActionFactory factory = ActionFactory.getFactory();
            final EmbeddingsAction action = factory.getEmbeddingsAction();
            final float[] embeddings = action.execute(content);
            final List<Float> floatList = new ArrayList<>();
            for (final float f : embeddings) {
                floatList.add(f);
            }
            return floatList;
        } catch (final Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

```
import java.util.List;
import java.util.Map;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import jp.co.intra_mart.foundation.copilot.vectorstore.document.Document;
import jp.co.intra_mart.foundation.copilot.vectorstore.document.exception.DocumentConversionException;

/**
 * Documentインタフェースの実装クラス
 */
public class VectorDatabaseSampleDocument implements Document {

    // ドキュメントの本文テキスト
    private String text;

    // ドキュメントのベクトル表現（例：埋め込みベクトル）
    private List<Double> vector;

    // ドキュメントに付随するメタデータ（キーと値のペア）
    private Map<String, String> metadata;

    // ドキュメントのオリジンソースID（例：ファイル名）
    private String originSourceId;


    @Override
    public String getMetadata() {
        // メタデータをJSON文字列に変換して返す
        final ObjectMapper objectMapper = new ObjectMapper();
        try {
```

```
        return objectMapper.writeValueAsString(metadata);
    } catch (final JsonProcessingException e) {
        throw new IllegalStateException(e);
    }
}

@SuppressWarnings("unchecked")
@Override
public <T> T getMetadata(final Class<T> clazz) throws DocumentConversionException {
    // 要求された型に応じてメタデータを返す
    if (clazz.isAssignableFrom(Map.class)) {
        return (T) metadata;
    }
    // 文字列型が要求された場合はJSON文字列を返す
    if (clazz.isAssignableFrom(String.class)) {
        return (T) getMetadata();
    }
    // 対応していない型の場合は例外をスロー
    throw new IllegalStateException("Unsupported metadata type requested.");
}

@Override
public String getText() {
    // テキストを返す
    return text;
}

@Override
public List<Double> getVector() {
    // ベクトル情報を返す
    return vector;
}

@Override
public String getOriginSourceId() {
    // オリジンソースIDを返す
    return originSourceId;
}


@SuppressWarnings("unchecked")
@Override
public void setMetadata(final Object metadata) throws DocumentConversionException {
    // メタデータを設定（Map型である必要がある）
    if (metadata instanceof Map) {
        this.metadata = (Map<String, String>) metadata;
        return;
    }
    // 対応していない型の場合は例外をスロー
    throw new IllegalStateException("Unsupported metadata type provided.");
}

@Override
public void setText(final String text) {
    // テキストを設定
    this.text = text;
}

@Override
public void setVector(final List<Double> vector) {
    // ベクトルを設定
    this.vector = vector;
}

@Override
public void setOriginSourceId(final String originSourceId) {
    // オリジンソースIDを設定
    this.originSourceId = originSourceId;
}

}
```

# JavaScript API の利用方法

## 概要

IM-Copilot の JavaScript API は、スクリプト開発環境で生成AIを簡単に利用できるインタフェースを提供します。

## 主なインタフェースと機能

**主なインタフェースと機能**

| インタフェース | 機能 | 詳細リンク |
|---|---|---|
| ChatAction | チャットアクション用インタフェースです。ユーザとの対話を処理し、自然言語による応答を生成します。 | API - ChatAction |
| EmbeddingsAction | 埋め込みアクション用インタフェースです。テキストをベクトル（数値の並び）に変換し、意味的な類似性の計算等に利用します。 | API - EmbeddingsAction |
| ImageGenerationAction | 画像生成アクション用インタフェースです。テキストによる指示から画像を生成する機能を提供します。 | API - ImageGenerationAction |
| AudioSpeechAction | 音声生成アクション用インタフェースです。テキストを音声に変換し、読み上げ等に利用します。 | API - AudioSpeechAction |
| AudioTranscriptionAction | 文字起こしアクション用インタフェースです。音声データを解析して、対応するテキストを抽出します。 | API - AudioTranscriptionAction |

## 実装サンプル

実装サンプルについては、以下のセクションを参照してください：

### チャットサンプル

スクリプト開発モデルのAPI「ChatAction」を使った基本的な実装例を紹介します。

> **コラム**
>
> 「ChatAction」のAPI仕様は以下を参考にしてください：
> https://api.intra-mart.jp/iap/apilist-ssjs/doc/platform/ChatAction/index.html

#### テキスト入力サンプル

システムプロンプトとユーザプロンプトを含むメッセージをもとに、アシスタントメッセージを取得します。

```
function run(input) {
  var action = new ChatAction();
  var generalAffairsRules = {
    "名刺発注": "名刺は毎月15日までに社内ポータルの申請フォームから依頼してください。上長の承認が必要です。",
    "備品購入": "備品は総務部のSlackチャンネルにて申請してください。月末締めでまとめて発注します。",
    "勤怠修正": "勤怠修正は人事システムから申請し、所属長の承認を得てください。",
    "出張申請": "出張は出張申請書を提出し、部門長と経理部の承認を得る必要があります。",
    "社内イベント": "社内イベントの企画は、総務部に事前相談のうえ、稟議申請を行ってください。"
  };
  var systemPrompt = "あなたは総務担当です。以下の社内ルールに基づき、質問に対して簡潔かつ明瞭に回答してください。\n\n";
  for (var key in generalAffairsRules) {
    if (generalAffairsRules.hasOwnProperty(key)) {
      systemPrompt += "- " + key + ": " + generalAffairsRules[key] + "\n";
    }
  }
  var messages = [
    {
      role: 'system',
      content: systemPrompt
    },
    {
      role: 'user',
      content: '名刺発注の方法を教えてください。'
    }
  ];
  var result = action.execute(messages);
  if (!result.error) {
    var lastMessage = result.data[result.data.length - 1];
    // コンテンツフィルタ: AIサービスの不適切コンテンツ検出機能
    // 入力違反時はAPI実行時にエラーが発生、出力違反時はcontentFilterプロパティで検出可能
    if (lastMessage.contentFilter && lastMessage.contentFilter.output === 'blocked') {
      // 必要に応じて出力違反時の処理を記載
    }
    var assistantMessage = lastMessage.content;
    return {
      message: assistantMessage,
      error: false
    };
  } else {
    return {
      message: null,
      error: true
    };
  }
}
```

画像入力サンプル

システムプロンプトとユーザプロンプト、画像を含むメッセージをもとに、アシスタントメッセージを取得します。

```
function run(input) {
 var action = new ChatAction();
 var storage = new PublicStorage('product_A_v1.jpg');
 var dataUrl = ChatMessage.convertToImageDataUrl(storage.openAsBinary()).data;
 var messages = [
  {
   role: 'system',
   content: '製品画像を評価してください。デザイン、色、パッケージ、ロゴの配置などの視覚的な部分を分析してください。また、魅力的に見えるかも教えてください。'
  },
  {
   role: 'user',
   content: [
    {
     type: 'text',
     text: '次の製品画像を分析してください。'
    },
    {
     type: 'image_url',
     image_url: {
      url: dataUrl
     }
    }
   ]
  }
 ];
 var result = action.execute(messages);
 if (!result.error) {
  var lastMessage = result.data[result.data.length - 1];
  var assistantMessage = lastMessage.content;
  return {
   message: assistantMessage,
   error: false
  };
 } else {
  return {
   message: null,
   error: true
  };
 }
}
```

ツール呼び出しサンプル

ユーザプロンプト、ツール定義、およびツール呼び出し結果を含むメッセージをもとに、アシスタントメッセージを取得します。

```
function run(input) {
 // 1. ChatAction の取得
 var action = new ChatAction();

 // 2. ツール定義の作成（プレーンオブジェクトとして定義）
 var inventoryTool = {
  name: 'check_inventory',
  description: '商品の在庫数を確認します',
  parameters: {
   type: 'object',
   properties: {
    productCode: {
     type: 'string',
     description: '商品コード'
    },
    warehouseCode: {
     type: 'string',
     description: '倉庫コード（省略時は全倉庫）'
    }
   },
   required: ['productCode'],
   additionalProperties: false
  }
 };
```

```
  // 3. ツール設定の作成
  var toolConfig = {
    toolDefinitions: [inventoryTool],
    toolChoice: {type: 'auto'}
  };

  // 4. 初回メッセージの作成
  var messages = [];

  var userMessage = new ChatMessage();
  userMessage.role = 'user';
  userMessage.content = '商品コード P-001 の在庫数を教えてください。';
  messages.push(userMessage);

  // 5. ChatOption の設定
  var option = new ChatOption();
  option.temperature = 0.0;

  // 6. AI の応答を取得（ツール呼び出しを含む）
  var response = action.executeWithTool(messages, option, toolConfig);
  var lastMessage = response.data[response.data.length - 1];
  messages.push(lastMessage);

  // 7. ツール呼び出しの処理
  if (lastMessage.toolCalls && lastMessage.toolCalls.length > 0) {
    var chatTextContents = [];

    lastMessage.toolCalls.forEach(function(toolCall) {
      Debug.console('ツール呼び出し: ' + toolCall.name);
      Debug.console('引数: ' + toolCall.arguments);

      // 実際のツール実行（ここでは模擬的な在庫確認）
      var result = executeInventoryCheck(toolCall);
      Debug.console('ツール実行結果: ' + result);

      var chatTextContent = {
        type: "text",      // 固定で "text"
        toolCallId: toolCall.id,
        text: result
      };

      chatTextContents.push(chatTextContent);
    });

    // ツール実行結果を1つのメッセージにまとめて追加
    var toolMessage = new ChatMessage();
    toolMessage.role = 'tool'; // ツール結果は "tool" ロールで送信します
    toolMessage.content = chatTextContents;
    messages.push(toolMessage);

    // 8. ツール実行結果を踏まえた最終応答を取得
    var finalResponse = action.executeWithTool(messages, option, toolConfig);

    // 最終応答を出力
    if (!finalResponse.error) {
      var finalResponseMessage = finalResponse.data[finalResponse.data.length - 1];
      var assistantMessage = finalResponseMessage.content;
      return {
        message: assistantMessage,
        error: false
      };
    }
  }

  return {
    message: null,
    error: true
  };

}

/**
 * 在庫確認の実行（模擬実装）
```

```
 * @param {Object} toolCall - ツール呼び出し情報
 * @returns {string} 在庫確認結果
 */
function executeInventoryCheck(toolCall) {
 try {
  // バリデーション用のスキーマ定義（inventoryToolと同じ内容）
  var schema = {
   type: 'object',
   properties: {
    productCode: {
     type: 'string',
     description: '商品コード'
    },
    warehouseCode: {
     type: 'string',
     description: '倉庫コード（省略時は全倉庫）'
    }
   },
   required: ['productCode'],
   additionalProperties: false
  };

  // JsonSchemaValidatorを使用したバリデーション実行
  var validator = new JsonSchemaValidator(schema);
  var validResult = validator.isValidJson(toolCall.arguments);

  if (!validResult.data) {
   // バリデーションエラーがある場合、詳細を取得
   var errors = validator.validateJson(toolCall.arguments);
   var errorMessages = [];

   if (errors.data && errors.data.length > 0) {
    errors.data.forEach(function(error) {
     errorMessages.push(error.propertyName + ': ' + error.message);
    });
   }

   return 'バリデーションエラー:\n- ' + errorMessages.join('\n- ');
  }

  // JSON 引数をパース
  var params = JSON.parse(toolCall.arguments);

  // 実際の在庫確認処理（ここでは固定値を返す）
  var stockQuantity = 150; // 実際はデータベースなどから取得

  return '商品コード ' + params.productCode + ' の在庫数: ' + stockQuantity + ' 個';
 } catch (e) {
  return '在庫確認エラー: ' + e.message;
 }
}
```

### 埋め込みサンプル

スクリプト開発モデルのAPI「EmbeddingsAction」を使った基本的な実装例を紹介します。

> **コラム**
>
> 「EmbeddingsAction」のAPI仕様は以下を参考にしてください：
> https://api.intra-mart.jp/iap/apilist-ssjs/doc/platform/EmbeddingsAction/index.html

#### テキストの埋め込みデータ生成

入力テキストを、検索や分類に使えるベクトル形式の埋め込みデータに変換します。

```
function run(input) {
 var action = new EmbeddingsAction();
 var manualIntroText = "このマニュアルでは、新しい勤怠管理システムの使い方についてご案内します。画面の指示に従って操作を進めてください。";
 var result = action.execute(manualIntroText);
 if (!result.error) {
  return {
   embeddedValues: result.data,
   error: false
  };
 } else {
  return {
   embeddedValues: null,
   error: true
  };
 }
}
```

### 画像生成サンプル

スクリプト開発モデルのAPI「ImageGenerationAction」を使った基本的な実装例を紹介します。

> **コラム**
>
> 「ImageGenerationAction」のAPI仕様は以下を参考にしてください：
> https://api.intra-mart.jp/iap/apilist-ssjs/doc/platform/ImageGenerationAction/index.html

#### テキストから画像ファイル生成

入力されたテキストをもとに、ストレージに画像ファイル（png）を生成します。

```
function run(input) {
 var action =  new ImageGenerationAction();
 var storage = new PublicStorage('ui_mockup_ecommerce_smartphone_v1.png');

 var device = "スマートフォン画面";
 var appType = "フラワーデザインを取り入れたEコマースアプリのUI";
 var designStyle = "花柄をモチーフにした柔らかく華やかなデザイン、淡いピンクとグリーンを基調";
 var layout = "上部に検索バー、中央に季節の花をテーマにした商品のグリッド表示（花束、フラワーギフト、園芸用品など）";
 var navigation = "下部にナビゲーションバー（ホーム、カテゴリ、カート、マイページ）、アイコンにも花のモチーフを使用";
 var atmosphere = "自然で優しい雰囲気、視覚的に癒される背景、ユーザフレンドリーな印象";
 var prompt = device + "に表示された" + appType + "、" +
              designStyle + "、" +
              layout + "、" +
              navigation + "、" +
              atmosphere + "。";

 var result;
 storage.createAsBinary(function(writer, error) {
  result = action.execute(prompt, {
   output: writer
  });
 });
 return {
  error: result.error
 };
}
```

### 音声生成サンプル

スクリプト開発モデルのAPI「AudioSpeechAction」を使った基本的な実装例を紹介します。

> **コラム**
>
> 「AudioSpeechAction」のAPI仕様は以下を参考にしてください：
> https://api.intra-mart.jp/iap/apilist-ssjs/doc/platform/AudioSpeechAction/index.html

テキストから音声ファイル生成

入力されたテキストをもとに、ストレージに音声ファイル（mp3）を生成します。

```
function run(input) {
  var action = new AudioSpeechAction();
  var storage = new PublicStorage('speech_training_onboarding.mp3');
  var onboardingSpeech =
  "こんにちは、そしてようこそ〇〇株式会社へ。\n\n" +
  "この音声は、皆さんの新しいスタートをサポートするためのオンボーディングガイドです。" +
  "まずは、私たちの会社を選んでくださったことに、心から感謝いたします。\n\n" +
  "〇〇株式会社は、革新と信頼を大切にする企業です。" +
  "私たちは、社員一人ひとりが自分らしく働き、成長できる環境づくりを目指しています。\n\n" +
  "これから皆さんは、さまざまな部署やプロジェクトに関わっていくことが予想されますが、" +
  "どの場面でも「チームワーク」「誠実さ」「挑戦する姿勢」が求められます。" +
  "困ったことがあれば、遠慮なく周囲に相談してください。" +
  "私たちは、互いに支え合う文化を大切にしています。\n\n" +
  "最初の数週間は、業務の流れや社内ツールの使い方、会社のルールなどを学ぶ期間です。" +
  "焦らず、一歩ずつ慣れていってください。" +
  "皆さんの成長を、私たちは全力でサポートします。\n\n" +
  "最後に、皆さんがこの会社で素晴らしいキャリアを築いていくことを、心から楽しみにしています。" +
  "これから一緒に、〇〇株式会社の未来を創っていきましょう。\n\n" +
  "改めて、ようこそ。これからよろしくお願いします。";
  var result;
  storage.createAsBinary(function(writer, error) {
    result = action.execute(onboardingSpeech, {
      output: writer
    });
  });
  return {
   error: result.error
  };
}
```

文字起こしサンプル

スクリプト開発モデルのAPI「AudioTranscriptionAction」を使った基本的な実装例を紹介します。

> **コラム**
>
> 「AudioTranscriptionAction」のAPI仕様は以下を参考にしてください：
> https://api.intra-mart.jp/iap/apilist-ssjs/doc/platform/AudioTranscriptionAction/index.html

音声ファイルからテキスト生成

ストレージに保存された音声ファイル（mp3）をもとに、文字データを自動生成します。

```
function run(input) {
  var action = new AudioTranscriptionAction();
  var storage = new PublicStorage('transcription_demo_v1.mp3');
  var result;
  storage.openAsBinary(function(reader, error) {
    result = action.execute(reader);
  });
  if (!result.error) {
    return {
      message: result.data,
      error: false
    };
  } else {
    return {
      message: null,
      error: true
    };
  }
}
```

## IM-LogicDesignerタスクの使い方

項目



## タスクの概要

IM-LogicDesigner は、ローコード開発環境で生成AIを活用できるタスク群を提供しています。
プログラミング知識が少なくても、ドラッグ&ドロップで生成AI機能を業務フローに組み込むことができます。

## IM-LogicDesignerタスク

IM-Copilot の IM-LogicDesignerタスクには、以下のタスクが存在します。

## 汎用利用を想定したタスク

- チャットタスク
  入力メッセージを通して対話を行うためのタスクです。
- 画像生成タスク
  入力メッセージをもとに画像生成を行うためのタスクです。
- 埋め込みタスク
  入力メッセージをベクトルに変換するためのタスクです。
- 文字起こしタスク
  入力音声ファイルをもとに文字起こしを行うためのタスクです。
- 音声生成タスク
  入力メッセージをもとに音声生成を行うためのタスクです。

> **注意**
>
> Azure OpenAI Service をご利用の場合、音声生成タスクはご利用いただけません。
> また、 Amazon Bedrock をご利用の場合、文字起こしタスク、音声生成タスクはご利用いただけません。

> **コラム**
>
> 各タスクのパラメータには、最小値や最大値などの制限が存在する場合があります。

## ロジックフローアシスタントでの利用を想定したタスク

- メッセージ履歴の取得
  クライアントとアシスタントの間でやり取りを行ったメッセージ履歴を取得するためのタスクです。
- クライアントへデータを送信
  任意のデータをクライアントへ送信するためのタスクです。

## ベクトルデータベース操作を想定したタスク

- ベクトルデータベースコンテンツ登録タスク
  ベクトルデータベースにコンテンツを登録するためのタスクです。
- ベクトルデータベースコンテンツ削除タスク
  ベクトルデータベースからコンテンツを削除するためのタスクです。
- ベクトルデータベース類似度検索タスク
  ベクトルデータベースからベクトルの類似度が高いコンテンツを検索するためのタスクです。

- ベクトルデータベースキーワード検索タスク
  ベクトルデータベースから指定したキーワードを含むコンテンツを検索するためのタスクです。

> コラム
>
> ベクトルデータベースを利用するには、テナント環境セットアップでベクトルデータベース接続情報の設定が必要です。
> 設定内容の詳細については、「テナント環境セットアップ」 - 「ベクトルデータベース接続情報」 を参照してください。

## データ処理を想定したタスク

- テキスト抽出タスク
  ファイルバイナリデータからテキストを抽出するためのタスクです。
- テキスト分割タスク
  テキストを指定のサイズに分割するためのタスクです。
- ランク融合タスク
  複数の異なるランキング結果を一つの統合されたランキングにまとめるためのタスクです。

> コラム
>
> テキスト抽出タスクは、テキスト抽出にテキスト抽出機能（ND Universal Extractor）を利用します。
> 必要に応じて、テキスト抽出機能のテキスト抽出設定の調整を検討してください。
> 設定内容の詳細については、「設定ファイルリファレンス」 - 「テキスト抽出設定」 を参照してください。

## 各タスク仕様

各タスク仕様は「IM-LogicDesigner仕様書」- 「タスク一覧（IM-Copilot）」を参照してください。

## 実装サンプル

実装サンプルについては、以下のセクションを参照してください：

### チャットタスクサンプル

「チャットタスク」を活用することで、業務プロセスの効率化やユーザ対応の品質向上が期待できます。
例：
- 申請内容の自動チェックや分類による業務負荷の軽減
- 問い合わせ内容の要約やテンプレート返信による対応スピードの向上
- 社内ヘルプデスクでのFAQ対応の自動化
- 顧客対応履歴の分析による対応品質の改善

**サンプル資材のダウンロード**：
以下のリンクから、サンプルをまとめてダウンロードできます。
**im_logicdesigner_chat_task_samples-data.zip**

> コラム
>
> 「チャットタスク」の仕様は以下を参考にしてください：
> https://document.intra-mart.jp/library/iap/public/im_logic/im_logic_specification/texts/appendix/task/im_copilot/im_chat.html

#### テキスト入力で申請内容をチェック

**1. 用途例**：

出張費申請の文面に不備がないかを確認し、必要に応じて修正案を提示する。

**2. プロンプト例**：

システム：

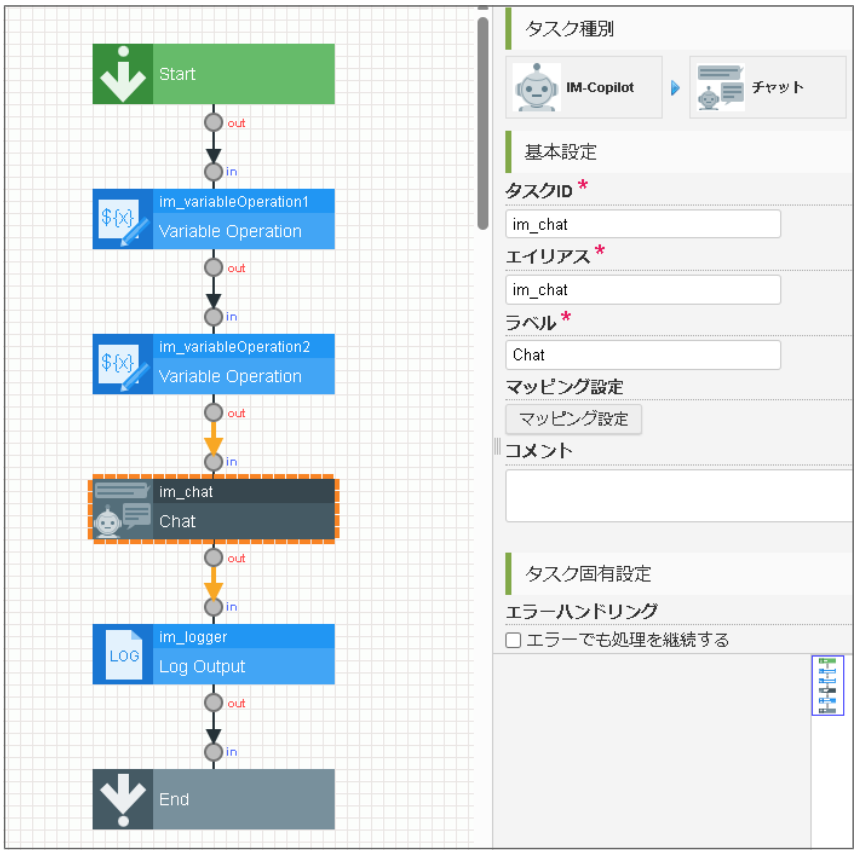
```
以下の出張費申請文について、不備がないかを確認してください。
不備の定義：
- 文法や表記の誤り
- 不自然または曖昧な表現
- 出張の目的、期間、行き先、費用の内訳などが不明確
- 社内文書として不適切な語調や表現
出力ルール：
- 不備がない場合：「問題ありません。」とだけ返答してください。
- 不備がある場合：
  1. 問題点を簡潔に指摘してください。
  2. 修正案を丁寧な文体で提示してください。
出力は簡潔かつ明瞭にしてください。
```

ユーザ：

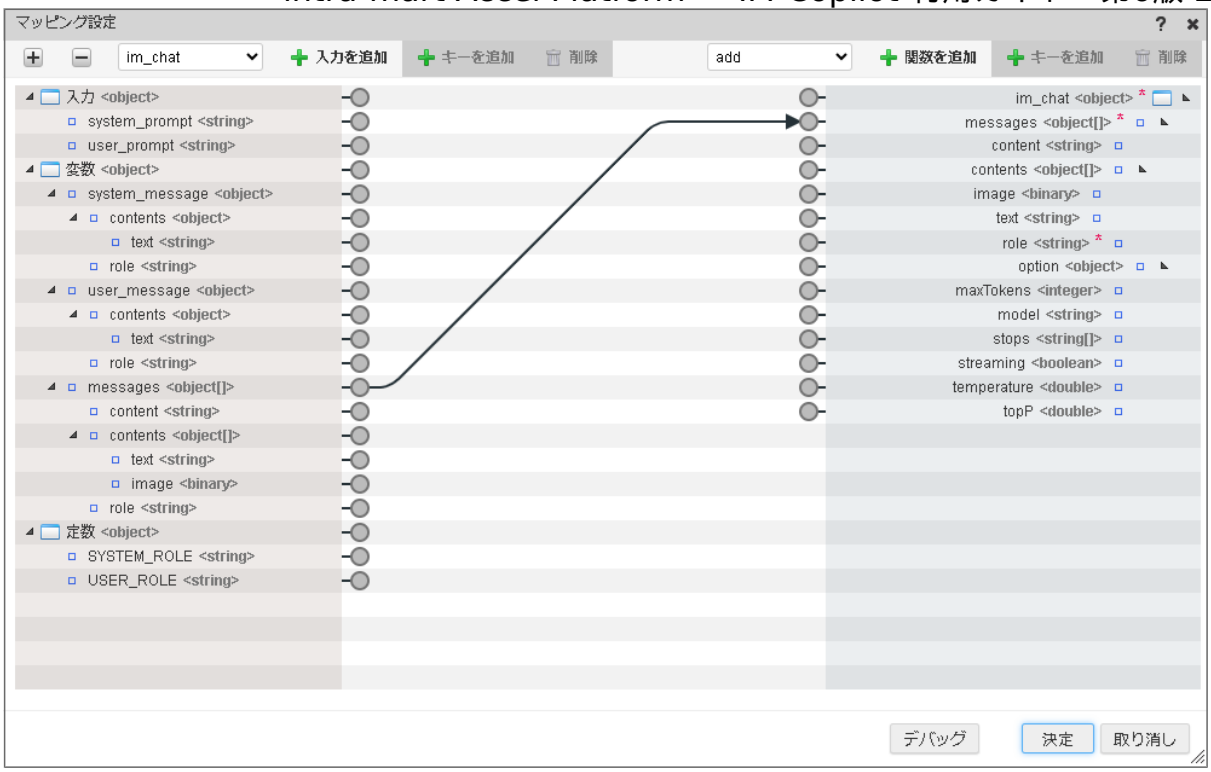
```
出張費申請：
出張費用の精算をお願いしたいです。27,000円です。
```

**3.** サンプル：

「チャットタスク（im_chat）」を配置したフローを作成します。




「messages<object[]>」にメッセージ配列変数（システムプロンプト、ユーザプロンプト）をマッピングします。




> **コラム**
>
> その他のタスクに関するマッピングや実行結果については、サンプルをインポートして確認してください。

テキスト＆画像入力で申請画像をチェック

**1. 用途例**：

出張費申請に添付された領収書画像を確認し、記載内容（例：金額、日付、店舗名など）を抽出・検証、不鮮明な画像や情報の不備がある場合は指摘し、必要に応じて修正案や再提出を促す。

**2. プロンプト例**：

システム：

```
以下の出張費申請に添付された領収書画像について、次の点を確認してください。
確認項目：
- 画像に記載された金額、日付、店舗名を正確に抽出すること
- 画像が不鮮明で読み取れない箇所がないか
- 情報が欠落していないか（例：金額や日付が不明、店舗名が記載されていないなど）
- 抽出した情報が申請文と整合しているか（※申請文がある場合）
出力ルール：
- 不備がない場合：「問題ありません。」とだけ返答してください。
- 不備がある場合：
  1. 問題点を簡潔に指摘してください。
  2. 必要に応じて再提出や補足説明を促してください。
  3. 抽出できた情報は「金額：〜円」「日付：〜」「店舗名：〜」の形式で出力してください。
出力は簡潔かつ明瞭にしてください。
```

ユーザ：

```
出張費申請（領収書画像あり）：
5月28日の東京出張にかかる交通費として、27,000円を申請いたします。
領収書画像を添付いたしますので、ご確認をお願いいたします。
```

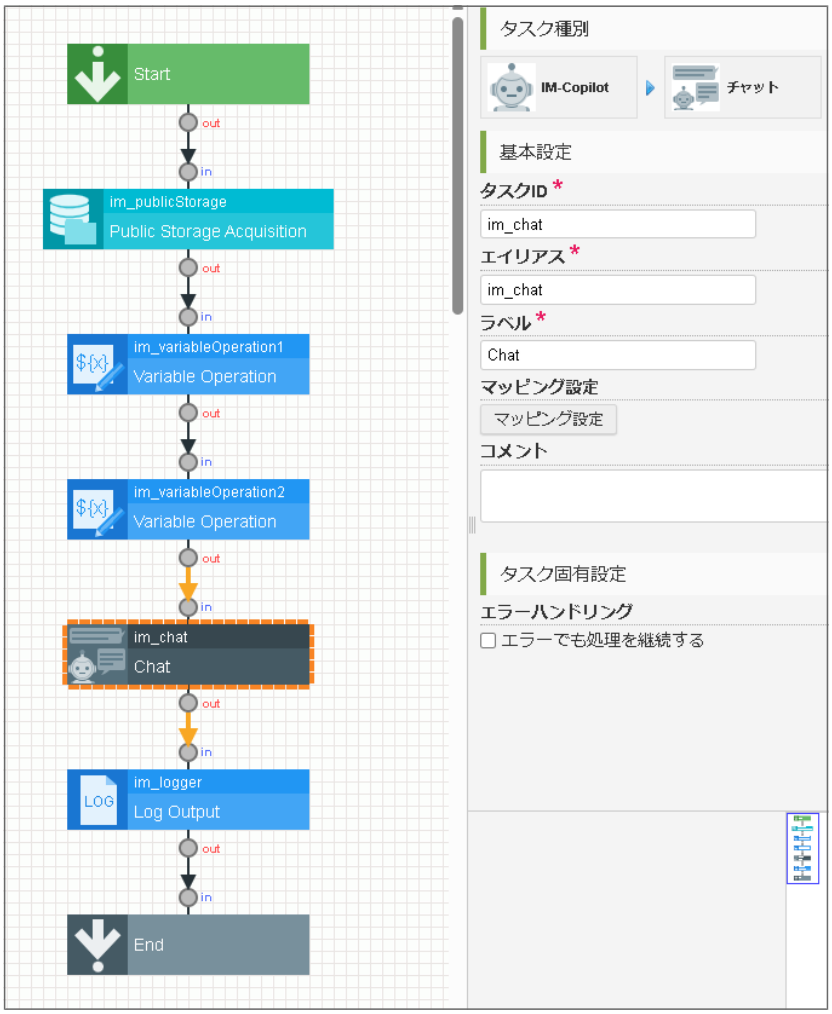
**3. 画像ファイル例**：

以下は参考用の画像内容とファイル名の例です（※実際の画像ファイルはサンプルには含まれていません。必要に応じてご用意ください。）：

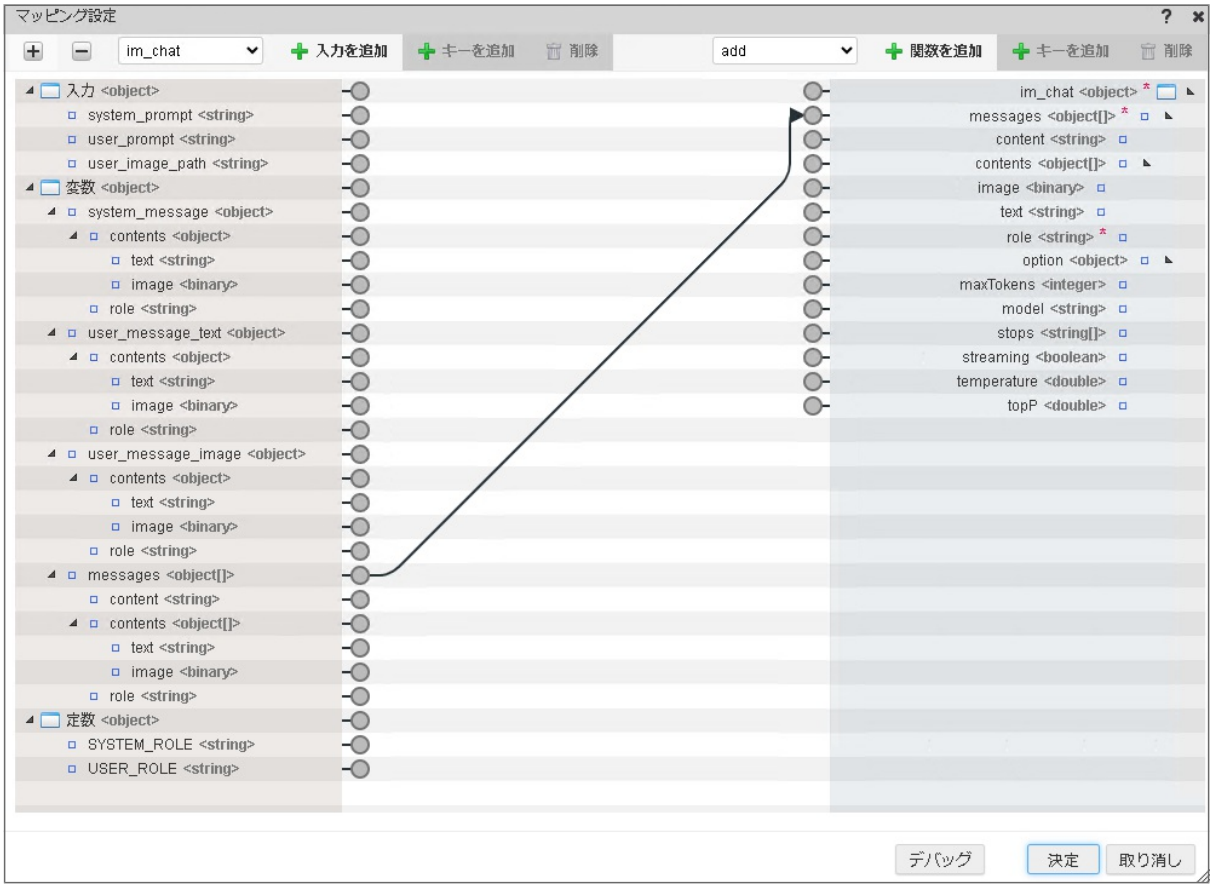
- 領収書（例：*receipt_sample.jpg*）
- 請求書（例：*invoice_sample.png*）
- 身分証明書（例：*id_card_sample.jpeg*）

**4.** サンプル：

「チャットタスク（im_chat）」を配置したフローを作成します。




「messages<object[]>」にメッセージ配列変数（システムプロンプト、ユーザプロンプト、画像）をマッピングします。



> コラム
>
> その他のタスクに関するマッピングや実行結果については、サンプルをインポートして確認してください。

文字起こしタスクサンプル

「文字起こしタスク」を活用することで、情報共有の効率化や記録精度の向上が期待できます。
例：
- 会議や打ち合わせの内容を自動でテキスト化し、議事録作成を効率化
- インタビューや講演内容の記録による情報の再利用性向上
- 研修やセミナーの内容を文字起こしし、学習資料として活用
- 電話対応内容の記録によるコンプライアンス強化

**サンプル資材のダウンロード**：
以下のリンクから、サンプルをまとめてダウンロードできます。
**im_logicdesigner_transcription_task_samples-data.zip**

> コラム
>
> 「文字起こしタスク」の仕様は以下を参考にしてください：
> https://document.intra-mart.jp/library/iap/public/im_logic/im_logic_specification/texts/appendix/task/im_copilot/im_transcription.html

会議音声の文字起こし

**1. 用途例**：

社内会議の音声を自動で文字起こしし、議事録の作成や社内共有に活用することで、業務の効率化と情報の正確な伝達を支援します。

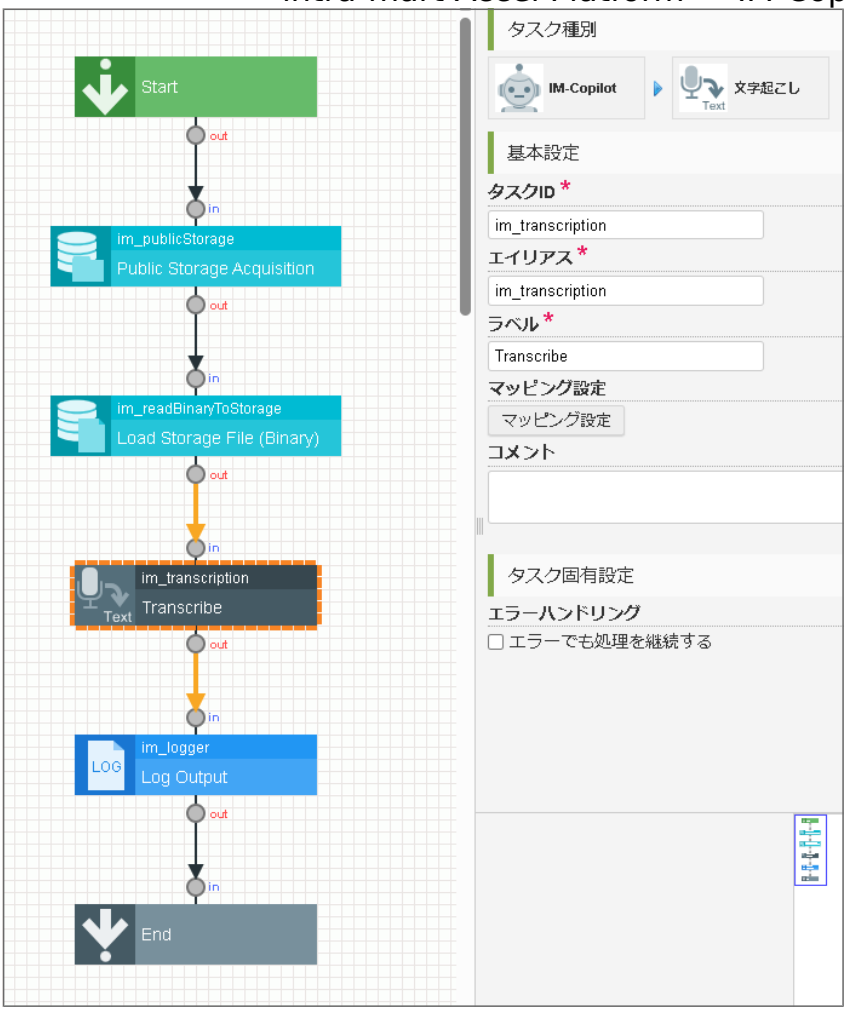
**2. 音声ファイル例**：

以下は参考用の音声内容とファイル名の例です（※実際の音声ファイルはサンプルには含まれていません。必要に応じてご用意ください）：

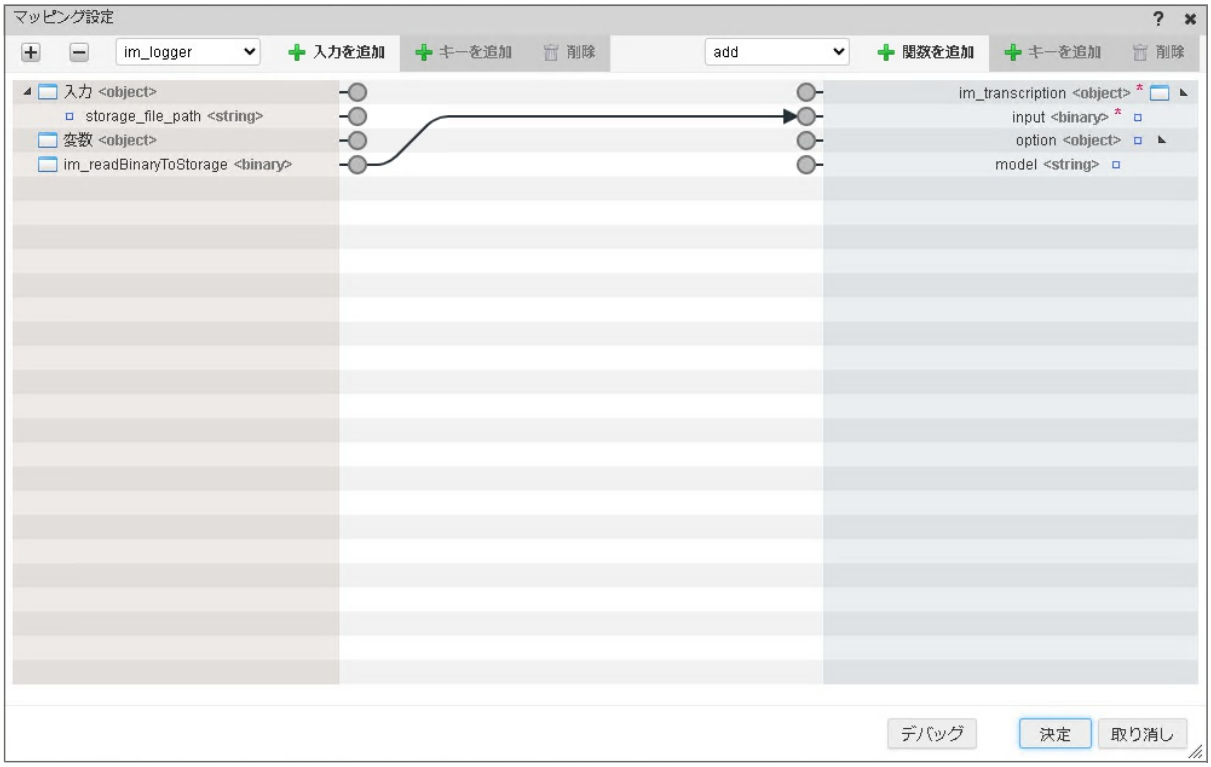
- 会議音声（例：*meeting_audio.mp3*）
- インタビュー音声（例：*interview_clip.wav*）
- 講義音声（例：*lecture_excerpt.m4a*）

**3. サンプル**：

「文字起こしタスク（im_transcription）」を配置したフローを作成します。




「input<binary>」に音声をマッピングします。




> **コラム**
>
> その他のタスクに関するマッピングや実行結果については、サンプルをインポートして確認してください。

音声生成タスクサンプル

「音声生成タスク」を活用することで、業務プロセスの効率化やユーザ体験の向上が期待できます。
例：
- 案内文やマニュアルの音声化による情報提供の自動化
- コールセンターやチャットボットでの読み上げ対応による対応品質の向上
- 視覚障がい者向けのアクセシビリティ対応の強化
- 社内教育コンテンツの音声化による学習効率の向上

**サンプル資材のダウンロード**：
以下のリンクから、サンプルをまとめてダウンロードできます。
**im_logicdesigner_audio_task_samples-data.zip**

> **コラム**
>
> 「音声生成タスク」の仕様は以下を参考にしてください：
> https://document.intra-mart.jp/library/iap/public/im_logic/im_logic_specification/texts/appendix/task/im_copilot/im_generateAudio.html

案内文の音声化

**1. 用途例**：

社内向けの案内文やFAQを音声に変換し、社内ポータルの読み上げ機能やeラーニング教材などに活用することで、情報共有の効率化とアクセシビリティの向上を図ります。

**2. 入力メッセージ例**：

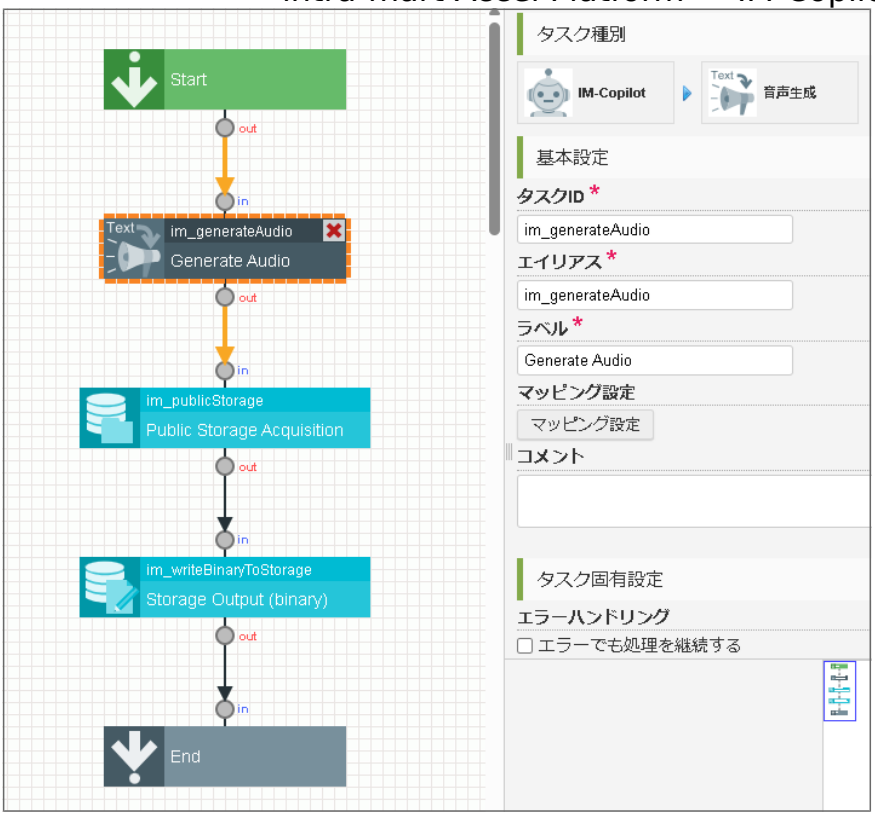
```
本日はお忙しい中、ご出社いただき誠にありがとうございます。
皆さまにとって有意義な一日となることを心より願っております。
なお、本日15時より、カフェテリアにて社内イベントを開催いたします。
リフレッシュや交流の機会として、ぜひお気軽にご参加ください。
```

```
システムメンテナンスに関するご案内です。
今週金曜日の18時から22時にかけて、社内ネットワークの一部サービスが一時的にご利用いただけない時間帯が発生します。
業務に影響が出ないよう、必要な作業は事前にご対応いただきますようお願いいたします。
ご不便をおかけいたしますが、ご理解とご協力のほどよろしくお願いいたします。
```

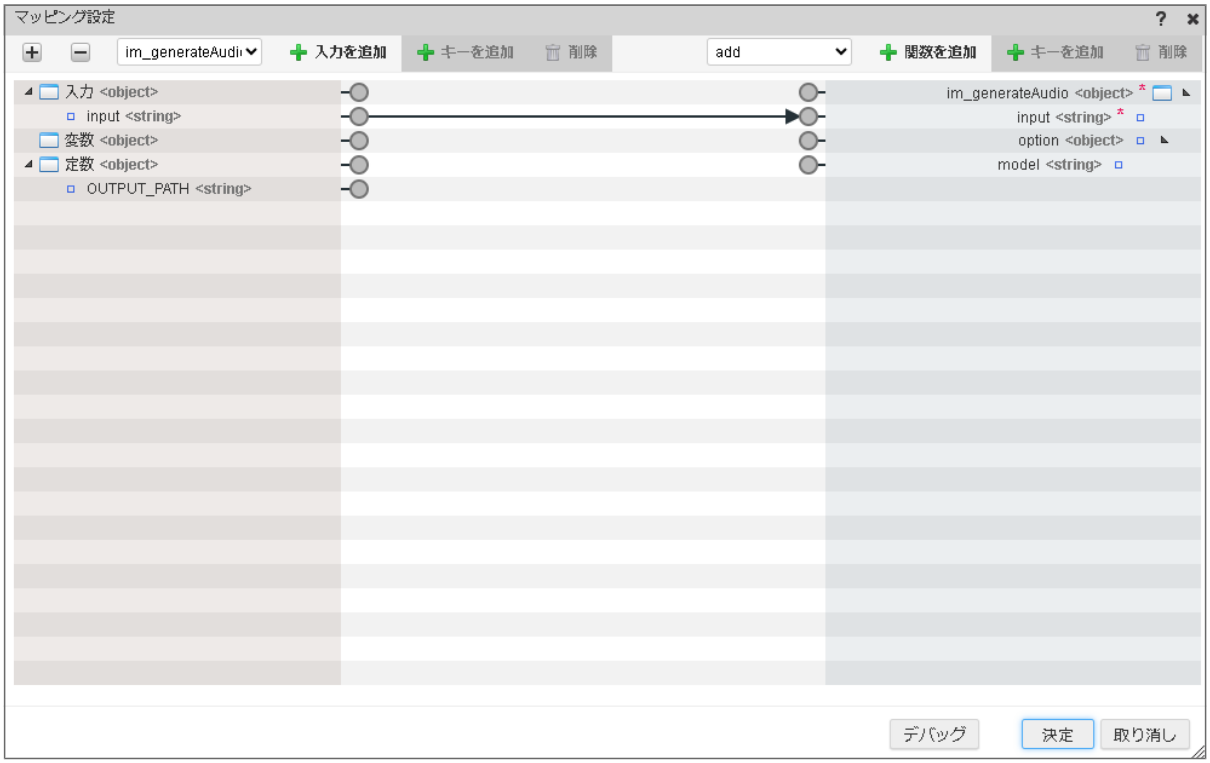
```
このたび、新しい勤怠管理システムの運用が開始されました。
新しいシステムでは、打刻の方法や申請の手続きが一部変更されています。
詳細につきましては、社内ポータルのお知らせ欄にてご確認いただけますので、必ずご一読ください。
ご不明点がございましたら、総務部までお問い合わせください。
```

**3. サンプル**：

「音声生成タスク（im_generateAudio）」を配置したフローを作成します。




「input<string>」に入力メッセージをマッピングします。




> **コラム**
>
> その他のタスクに関するマッピングや実行結果については、サンプルをインポートして確認してください。

画像生成タスクサンプル

「画像生成タスク」を活用することで、業務プロセスの効率化やユーザ体験の向上が期待できます。
例：

- 広告バナーやSNS投稿用画像の自動生成によるマーケティング業務の迅速化
- 商品イメージやパッケージデザインのプロトタイピングによる開発スピードの向上
- 教材やマニュアル用イラストの作成による教育・研修コンテンツの充実
- 建築・インテリア分野での空間イメージのビジュアライズによる顧客提案力の強化

**サンプル資材のダウンロード：**
以下のリンクから、サンプルをまとめてダウンロードできます。
**im_logicdesigner_image_task_samples-data.zip**

> コラム
>
> 「画像生成タスク」の仕様は以下を参考にしてください：
> https://document.intra-mart.jp/library/iap/public/im_logic/im_logic_specification/texts/appendix/task/im_copilot/im_generateImage.html

商品イメージのプロトタイピング

**1. 用途例：**

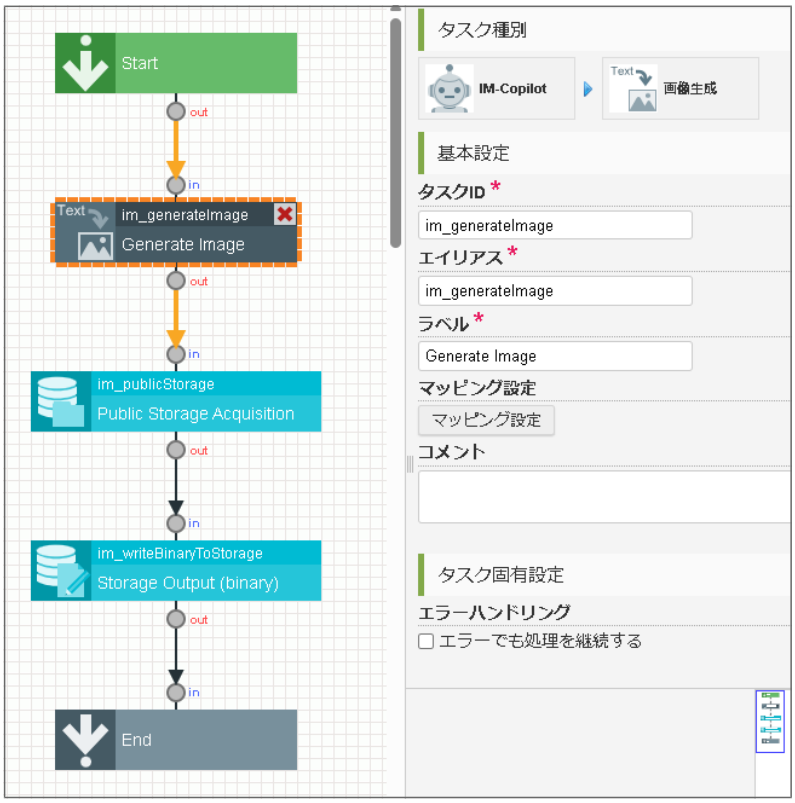
新商品のコンセプトイメージをAIで生成し、企画会議や顧客提案資料に活用することで、初期段階でのビジュアル検討や意思決定の迅速化を図ります。

**2. プロンプト例：**

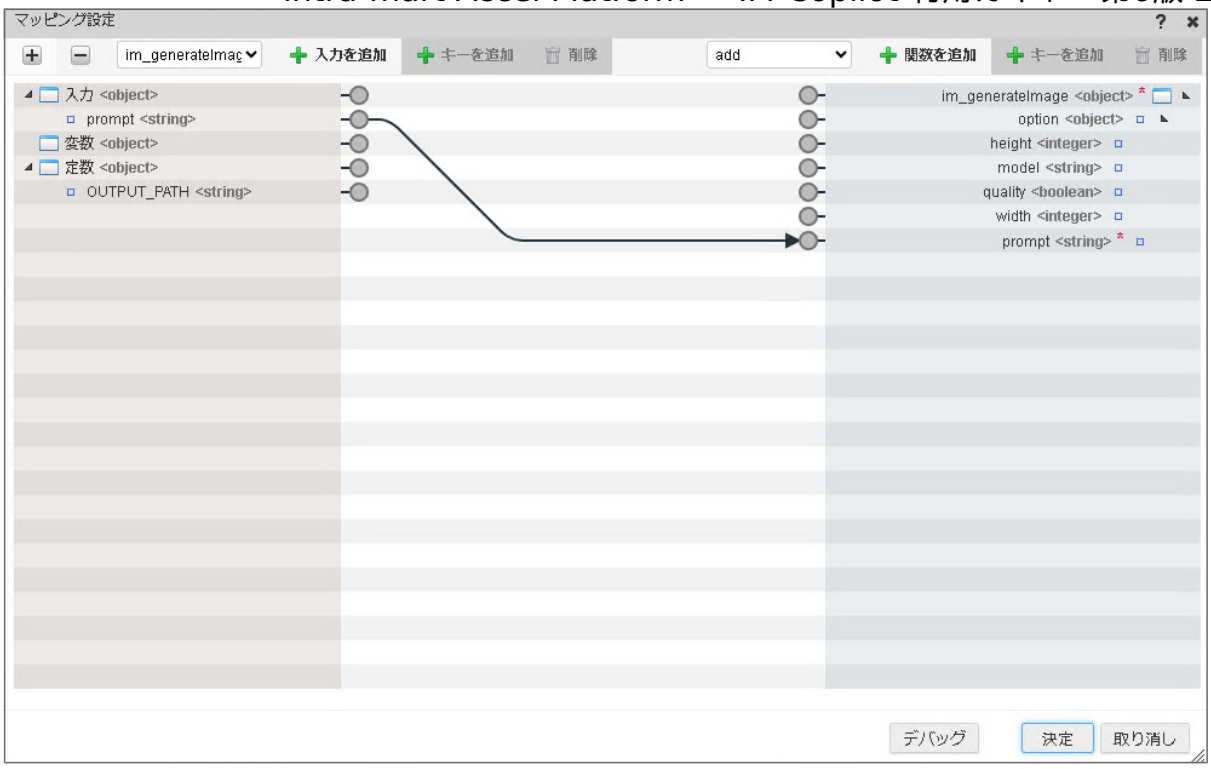
```
白を基調としたシンプルなデザインの電気ケトルのイメージを生成してください。
形状は丸みを帯びたフォルムで、取っ手と注ぎ口はステンレス調。
北欧風のインテリアにも合うような落ち着いた雰囲気を希望します。
```

**3. サンプル：**

「画像生成タスク（im_generateImage）」を配置したフローを作成します。




「prompt<string>」にプロンプトをマッピングします。




> **コラム**
>
> その他のタスクに関するマッピングや実行結果については、サンプルをインポートして確認してください。

埋め込みタスクサンプル

「埋め込みタスク」を活用することで、業務プロセスの効率化や情報活用の高度化が期待できます。
ベクトルデータベースとの連携によって、意味ベースの検索やレコメンドが実現可能です。
例：
- 社内文書の意味検索によるナレッジ共有の促進
- 類似申請や過去事例の自動検出による審査業務の効率化
- FAQやマニュアルの自動レコメンドによる問い合わせ対応の迅速化
- 顧客対応履歴の意味的分析による対応品質の向上

**サンプル資材のダウンロード**：
以下のリンクから、サンプルをまとめてダウンロードできます。
**im_logicdesigner_embedding_task_samples-data.zip**

> **コラム**
>
> 「埋め込みタスク」の仕様は以下を参考にしてください：
> https://document.intra-mart.jp/library/iap/public/im_logic/im_logic_specification/texts/appendix/task/im_copilot/im_embedding.html

入力メッセージをベクトルに変換

**1. 用途例**：

社内の申請文や問い合わせ文をベクトル化し、意味的な類似性に基づいて自動処理を行うことで、業務の効率化と情報活用の高度化を図ります。

**2. 入力メッセージ例**：

メッセージA（経費関連）：

```
経費精算の締切日は毎月末日です。出張費を含むすべての経費は、当月末までに申請を完了してください。
なお、申請には領収書の添付が必要ですので、事前にご準備をお願いします。
```

メッセージB（経費関連）：

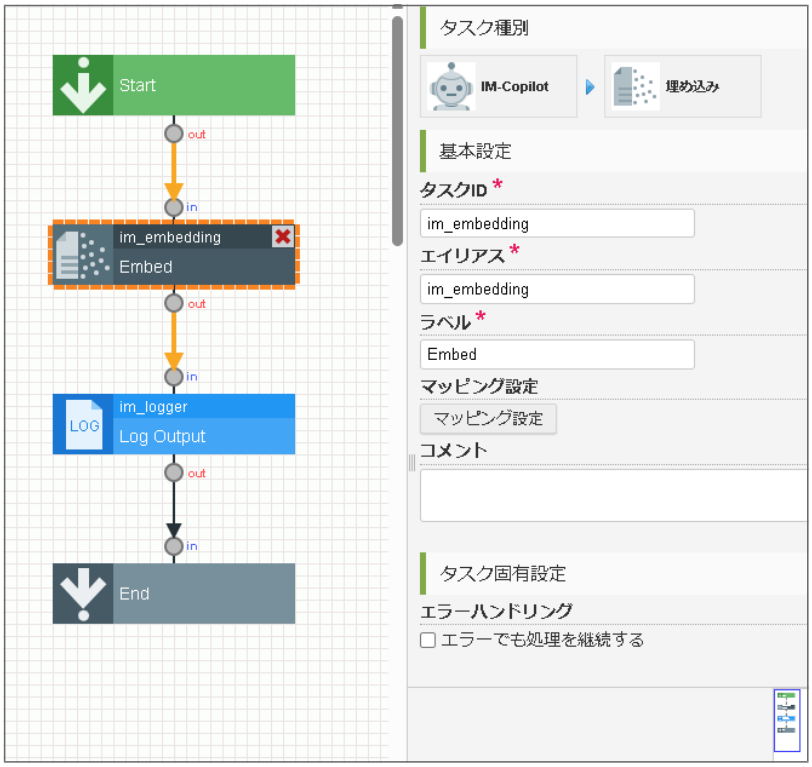
出張費の精算は、出張終了後5営業日以内に申請してください。
期限を過ぎると承認が遅れる可能性がありますので、早めの申請をお願いします。

メッセージC（ITトラブル）：

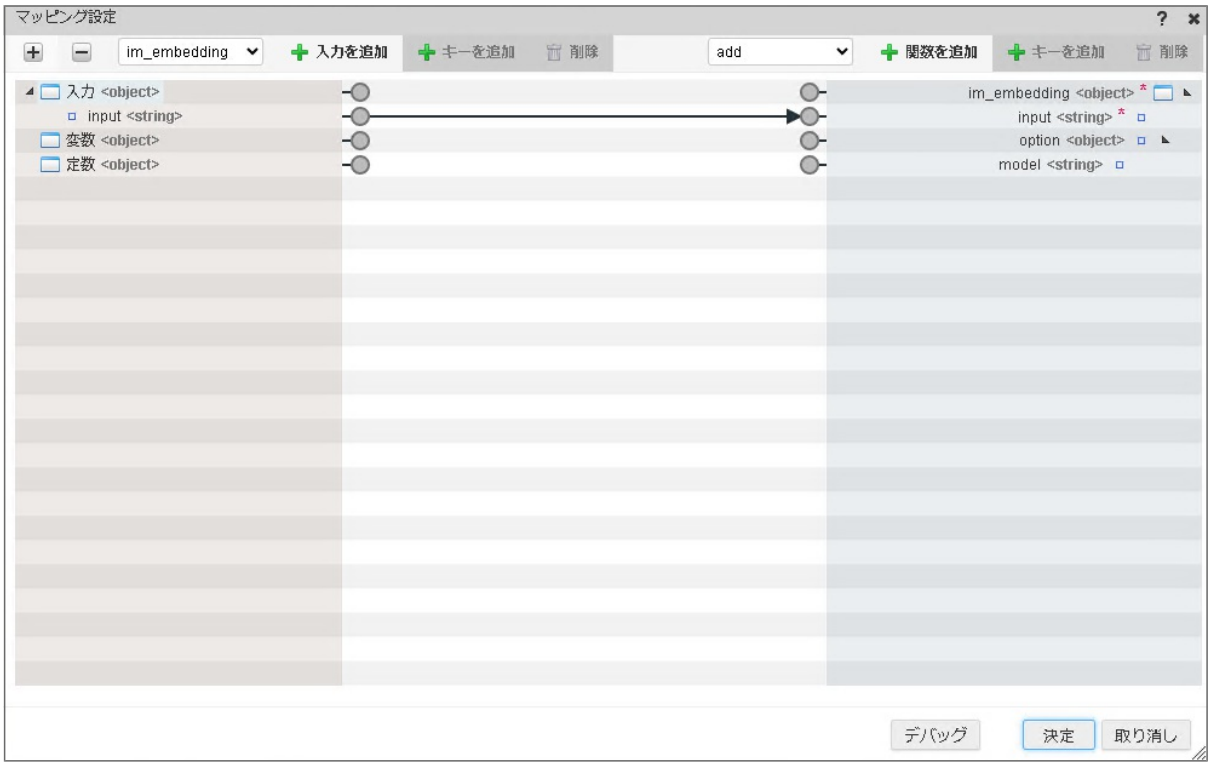
社内ネットワークに接続できない場合、まずはPCの再起動とLANケーブルの接続確認をお試しください。
解決しない場合は、ITサポート（内線1234）までご連絡ください。対応時間は平日9:00〜18:00です。

**3. サンプル：**

「埋め込みタスク（im_embedding）」を配置したフローを作成します。




「input<string>」に入力メッセージをマッピングします。



> **コラム**
>
> その他のタスクに関するマッピングや実行結果については、サンプルをインポートして確認してください。

ベクトルデータベース操作サンプル

「ベクトルデータベース」に関する操作タスクを通じて、埋め込み（Embedding）によって得られた意味ベクトルを活用した、文脈や意図に基づく検索・レコメンドが実現可能です。
これにより、従来のキーワード検索では難しかった高度な情報活用が実現し、業務プロセスの効率化・高度化が期待されます。
例：
- 類似FAQの検索と自動応答によるカスタマーサポートの迅速化
- 過去の設計資料や不具合報告からの類似事例抽出による設計・品質業務の効率化
- 社内ナレッジベースからの関連情報レコメンドによる業務支援の強化
- 顧客対応履歴のベクトル化とクラスタリングによる対応傾向の可視化と改善

**サンプル資材のダウンロード**：
以下のリンクから、サンプルをまとめてダウンロードできます。
**im_logicdesigner_vector_db_task_samples-data.zip**

> **コラム**
>
> このサンプルに含まれる「ベクトルデータベース」に関する仕様は以下を参考にしてください：
> https://document.intra-mart.jp/library/iap/public/im_logic/im_logic_specification/texts/appendix/task/im_copilot/

ベクトルデータベースコンテンツ登録

**1. 用途例**：

意味ベースの検索やレコメンドを可能にするために、社内の文書やナレッジをベクトル化してベクトルデータベースに登録します。

**2. 入力メッセージ例**：

メッセージA（経費関連）：

```
経費精算の締切日は毎月末日です。出張費を含むすべての経費は、当月末までに申請を完了してください。
なお、申請には領収書の添付が必要ですので、事前にご準備をお願いします。
```

メッセージB（経費関連）：

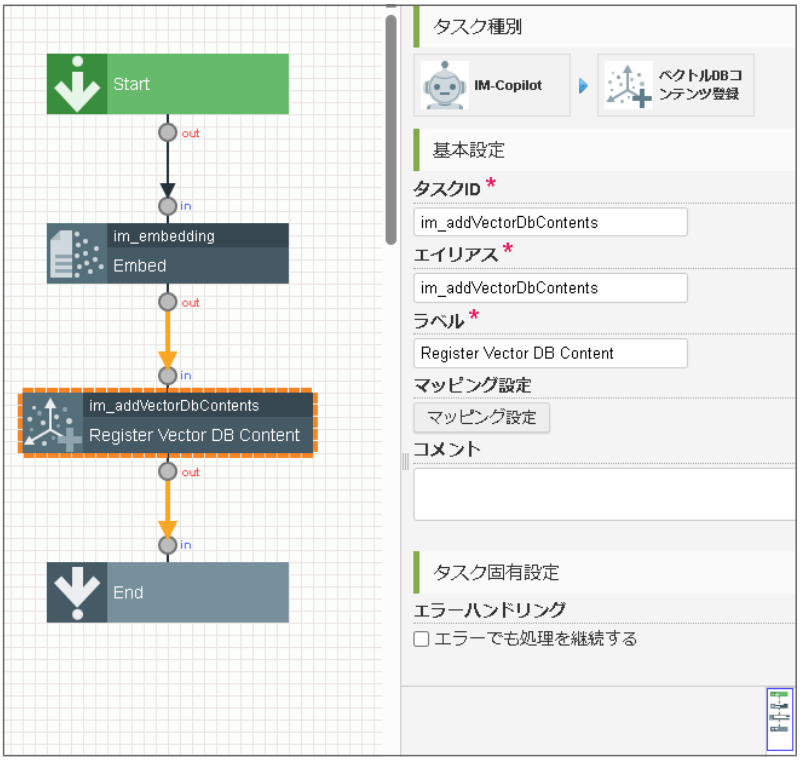
```
出張費の精算は、出張終了後5営業日以内に申請してください。
期限を過ぎると承認が遅れる可能性がありますので、早めの申請をお願いします。
```

メッセージC（ITトラブル）：

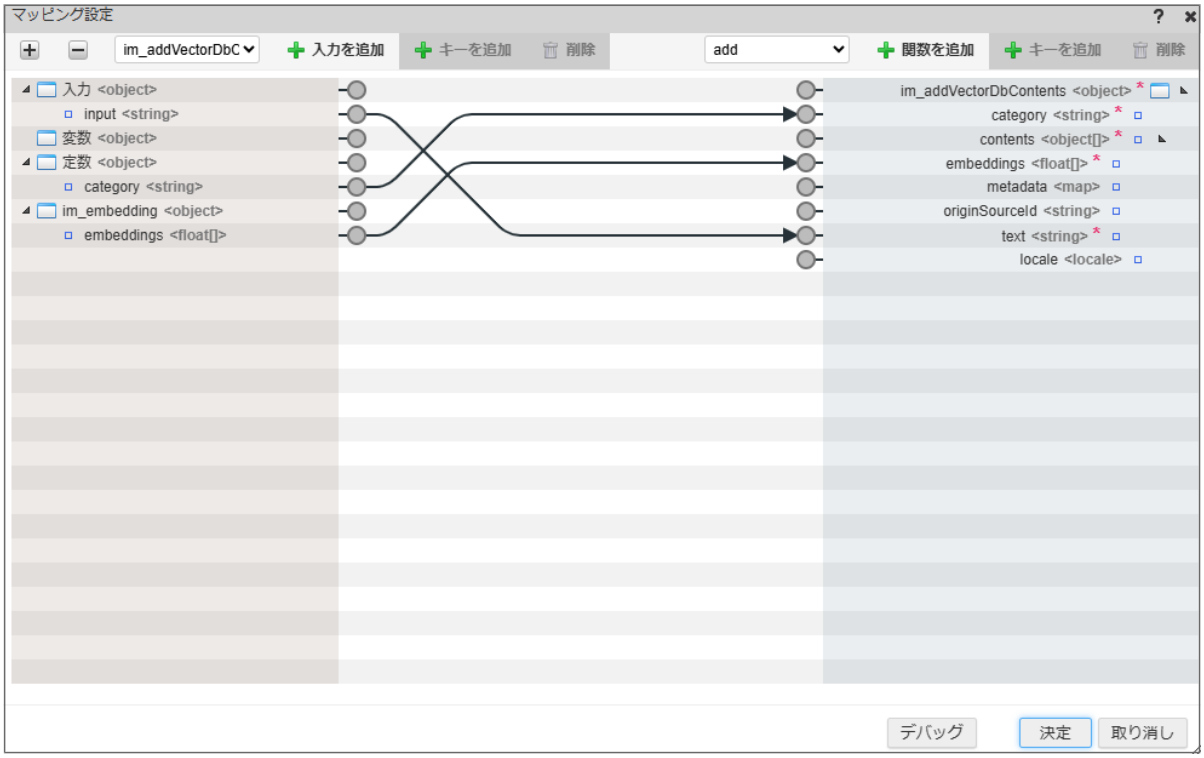
```
社内ネットワークに接続できない場合、まずはPCの再起動とLANケーブルの接続確認をお試しください。
解決しない場合は、ITサポート（内線1234）までご連絡ください。対応時間は平日9:00〜18:00です。
```

**3. サンプル**：

「ベクトルデータベースコンテンツ登録タスク（im_addVectorDbContents）」を配置したフローを作成します。




「category<string>」にカテゴリをマッピングします。
「embeddings<float[]>」に埋め込みデータをマッピングします。
「text<string>」に入力メッセージをマッピングします。




**コラム**

その他のタスクに関するマッピングや実行結果については、サンプルをインポートして確認してください。

コラム

ベクトルデータベースに入力メッセージ（テキスト）を登録する際には、テキストを適切に分割することが重要です。
分割の目的は、検索精度を高め、意味のあるチャンク（断片）単位で埋め込みを生成することです。

分割のポイント：
・意味のある単位で分割する：文や段落など、自然言語の意味が保たれる範囲で分割するのが理想です。
・サイズのバランス：チャンクが小さすぎると文脈が失われ、大きすぎると埋め込みの品質が下がる可能性があります。一般的には 200〜500文字程度がよく使われます。
・重複やオーバーラップ：検索精度を高めるために、チャンク間に少し重複を持たせることもあります。

「テキスト分割タスク」の利用も検討してください。：
https://document.intra-mart.jp/library/iap/public/im_logic/im_logic_specification/texts/appendix/task/im_copilot/im_splitText.html

ベクトルデータベース類似度検索

**1. 用途例：**

入力されたベクトルとベクトルデータベース内のデータとの類似度を計算し、意味的に近い情報を検索・提示します。

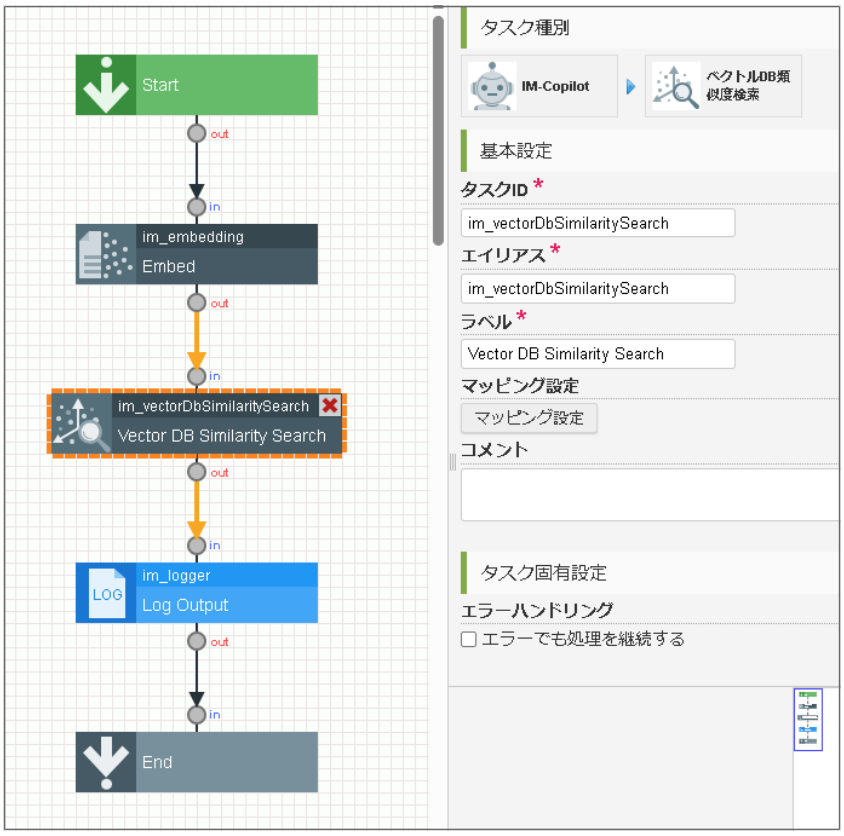
**2. 入力メッセージ例：**

経費関連を検索：

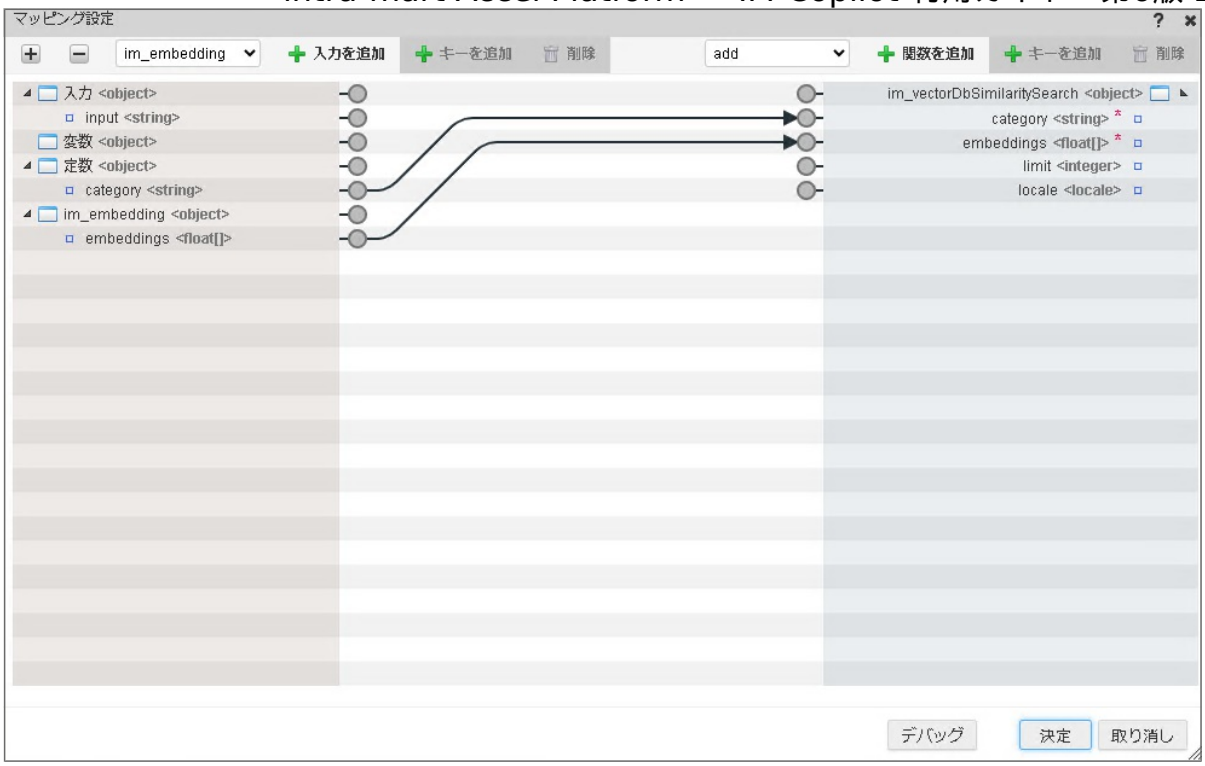
```
出張の経費申請はいつまでに提出すればよいですか？
```

**2. サンプル：**

「ベクトルデータベース類似度検索タスク（im_vectorDbSimilaritySearch）」を配置したフローを作成します。




「category<string>」にカテゴリをマッピングします。
「embeddings<float[]>」に埋め込みデータをマッピングします。




> **コラム**
>
> その他のタスクに関するマッピングや実行結果については、サンプルをインポートして確認してください。

## IM-BloomMaker のアシスタント実行エレメントの説明

### 概要

IM-BloomMaker では、アシスタント機能を画面に組み込むための専用エレメントを提供しています。
これにより、ローコードで高度なAI機能を備えた画面を簡単に構築することが可能です。

### アシスタント実行エレメント

エレメントの詳細は IM-BloomMaker ユーザ操作ガイドを参照してください：

- IM-BloomMaker ユーザ操作ガイド - IM-Copilot アシスタント

## プロコードでアシスタント実行 UI の埋め込みを行う方法

本章では、IM-BloomMaker を利用せずにプロコードでアシスタント実行 UI を埋め込む方法について解説します。
スクリプト開発モデルとJavaEE開発モデルの両方に対応しています。

目次



### 概要

アシスタント実行 UI は、IM-BloomMaker エレメントとして提供されていますが、プロコード開発者向け JavaScript 関数（imCopilot.register）を通じて同様に利用可能です。

この機能により、以下の開発モデルでアシスタント機能を組み込むことが可能です：

- スクリプト開発モデル
- JavaEE開発モデル

## 前提条件

- ロジックフローアシスタントや Wiki アシスタント が事前に作成されていること
- 作成したアシスタントに対して認可設定が行われていること
- 「アシスタントID」を取得済みであること

> **コラム**
>
> 「アシスタントID」の詳細については「*アシスタント定義*」を参考にしてください。

## 実装方法

開発モデルごとの実装方法については、以下のドキュメントを参照してください：

### スクリプト開発モデル「アシスタント実行 UI」実装

このドキュメントでは、スクリプト開発モデルで「アシスタント実行 UI」を埋め込む方法について解説します。

目次



#### 基本的な使い方

##### ステップ1: HTMLの準備

まず、プレゼンテーションページ（HTML）に以下の要素を追加します：

```
<imart type="head">
  <meta http-equiv="X-Intramart-Secure-Token" content='<imart type="imSecureToken" mode="value" />' />
  <script src="im_copilot/js/extension.bundle.js"></script>
</imart>
<div id="assistant-area"></div>
<script>
  // アシスタントUIを登録
  window.imCopilot.register({
    targetElementId: 'assistant-container',
    assistantId: 'your-assistant-id',
    threadId: 'unique-thread-id',
    inputPlaceholder: 'メッセージを入力してください'
  });
</script>
```

要素追加時のポイントは、以下の通りです：

- <head> タグにセキュアトークン情報（imSecureToken）を追加
- スクリプト im_copilot/js/extension.bundle.js を追加
- アシスタントUIを表示するDIV要素（ID）を追加
- JavaScript 関数（imCopilot.register）を用いて、DIV要素の中に「アシスタント実行 UI」を埋め込む処理を追加

##### ステップ2: パラメータの設定

`window.imCopilot.register()` メソッドには以下のパラメータを指定できます：

パラメーター一覧

| パラメータ名 | 必須 | 説明 |
|---|---|---|
| targetElementId | ○ | アシスタントUIを表示するDIV要素のID |
| assistantId | ○ | 使用するアシスタントのID |
| threadId | × | 会話履歴を識別するための一意のID |
| inputPlaceholder | × | 入力欄に表示するプレースホルダーテキスト（デフォルト: “プロンプトを入力”） |

### 実装例

#### シンプルな実装例

最もシンプルな実装例を示します。

**プレゼンテーションページ（sample_basic.html）：**

```
<imart type="head">
  <title>アシスタント実行サンプル - 基本実装</title>
  <meta http-equiv="X-Intramart-Secure-Token" content='<imart type="imSecureToken" mode="value" />' />
  <script src="im_copilot/js/extension.bundle.js"></script>
</imart>
<div id="assistant-area"></div>
<script>
  // アシスタントUIを登録
  window.imCopilot.register({
    targetElementId: "assistant-area",
    assistantId: "sample-rag-assistant",
    threadId: 'thread_' + new Date().getTime(),
    inputPlaceholder: "Input prompt"
  });
</script>
```

#### 動的にスレッドIDを生成する例

ユーザごとに異なるスレッドIDを生成する実装例です。

**ファンクション・コンテナ（sample_dynamic.js）：**

```
var threadId;

function init(request) {

  // アシスタントID
  let assistantId = "sample-rag-assistant";

  // アカウントコンテキスト取得
  let accountContexts = Contexts.getAccountContext();

  // ユーザコードを取得
  let userCd = accountContexts.userCd;

  // タイムスタンプを生成
  let timestamp = new Date().getTime();

  // スレッドIDを生成
  threadId = 'thread_' + userCd + '_' + assistantId + '_' + timestamp;

}
```

**プレゼンテーションページ（sample_dynamic.html）：**

```
<imart type="head">
 <title>アシスタント実行サンプル - 動的制御</title>
 <meta http-equiv="X-Intramart-Secure-Token" content='<imart type="imSecureToken" mode="value" />' />
 <script src="im_copilot/js/extension.bundle.js"></script>
</imart>
<div id="assistant-area"></div>
<script>
 // アシスタントUIを登録
 window.imCopilot.register({
   targetElementId: "assistant-area",
   assistantId: "sample-rag-assistant",
   threadId: '<imart type="string" value=threadId />',
   inputPlaceholder: "Input prompt"
 });
</script>
```

### 注意事項

1. **extension.bundle.jsの読み込み位置**
   - extension.bundle.jsは必ず<head>タグ内で読み込んでください
   - body内で読み込むと、正常に動作しない可能性があります
2. **スレッドIDの管理**
   - threadIdは会話履歴を識別するために使用されます
   - 同じthreadIdを使用すると、以前の会話履歴が引き継がれます
   - 新しい会話を開始したい場合は、新しいthreadIdを生成してください
3. **セキュリティトークンの設定**
   - meta要素でX-Intramart-Secure-Tokenを必ず設定してください
   - これがないとAPIへのアクセスが拒否されます

### トラブルシューティング

#### アシスタントが表示されない場合

以下の点を確認してください：

1. **extension.bundle.jsの読み込み位置**
   - <head>タグ内で読み込まれているか確認
   - bodyタグ内での読み込みではないか
2. **セキュリティトークンの設定**
   - `<meta http-equiv="X-Intramart-Secure-Token">` が設定されているか
   - `<imart type="imSecureToken" mode="value" />` の記述が正しいか
3. **HTML要素の存在確認**
   - targetElementIdで指定したID属性を持つ要素が存在するか
   - DOMContentLoadedイベント発生後に実行されているか
4. **アシスタントID**
   - assistantIdが正しいか
   - アシスタントへのアクセス権限があるか

#### エラーメッセージの対処

**よくあるエラーと対処法**

| エラーメッセージ | 対処法 |
|---|---|
| “Error: Failed to find extension parent element” | 指定したtargetElementIdの要素が存在することを確認 |
| “Error: Assistant configuration error. Check definition and authorization settings.” | アシスタントIDが正しいか、アシスタントが存在するか、アシスタントへのアクセス権限があるか確認 |
| “TypeError: Cannot read properties of undefined” | extension.bundle.jsが<head>タグ内で読み込まれているか確認 |
| “TypeError: Cannot read properties of null” | セキュリティトークンが正しく設定されているか確認 |

### まとめ

スクリプト開発での「アシスタント実行 UI」の埋め込みは、わずか数行のコードで実現できます。
この機能を活用することで、独自のWebアプリケーションにAIアシスタント機能を簡単に統合できます。

### 関連ドキュメント

- スクリプト開発プログラミングガイド

## JavaEE開発モデル「アシスタント実行 UI」実装

このドキュメントでは、JavaEE開発モデルで「アシスタント実行 UI」を埋め込む方法について解説します。

目次



### 基本的な使い方

#### ステップ1: JSPページの準備

まず、JSPページに以下の要素を追加します：

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="imst" uri="http://www.intra-mart.co.jp/taglib/imst" %>
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui" %>
<imui:head>
  <meta http-equiv="X-Intramart-Secure-Token" content='<imst:imSecureToken mode="value" />' />
  <script src="im_copilot/js/extension.bundle.js"></script>
</imui:head>
<div id="assistant-area"></div>
<script>
  // アシスタントUIを登録
  window.imCopilot.register({
    targetElementId: 'assistant-area',
    assistantId: 'your-assistant-id',
    threadId: 'unique-thread-id',
    inputPlaceholder: 'メッセージを入力してください'
  });
</script>
```

要素追加時のポイントは、以下の通りです：

- タグライブラリ（imst、imui）を追加
- <imui:head> タグにセキュアトークン情報（imSecureToken）を追加
- スクリプト im_copilot/js/extension.bundle.js を追加
- アシスタントUIを表示するDIV要素（ID）を追加
- JavaScript 関数（imCopilot.register）を用いて、DIV要素の中に「アシスタント実行 UI」を埋め込む処理を追加

#### ステップ2: パラメータの設定

`window.imCopilot.register()` メソッドには以下のパラメータを指定できます：

パラメーター一覧

| パラメータ名 | 必須 | 説明 |
| --- | --- | --- |
| targetElementId | ○ | アシスタントUIを表示するDIV要素のID |
| assistantId | ○ | 使用するアシスタントのID |
| threadId | × | 会話履歴を識別するための一意のID |
| inputPlaceholder | × | 入力欄に表示するプレースホルダーテキスト（デフォルト: “プロンプトを入力”） |

### 実装例

#### シンプルな実装例

最もシンプルな実装例を示します。

**JSPページ（sample_basic.jsp）:**

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="imst" uri="http://www.intra-mart.co.jp/taglib/imst" %>
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui" %>
<imui:head>
   <title>アシスタント実行サンプル - 基本実装</title>
   <meta http-equiv="X-Intramart-Secure-Token" content='<imst:imSecureToken mode="value" />' />
   <script src="im_copilot/js/extension.bundle.js"></script>
</imui:head>
<div id="assistant-area"></div>
<script>
   // アシスタントUIを登録
   window.imCopilot.register({
      targetElementId: "assistant-area",
      assistantId: "sample-rag-assistant",
      threadId: 'thread_' + new Date().getTime(),
      inputPlaceholder: "Input prompt"
   });
</script>
```

#### 動的にスレッドIDを生成する例

ユーザごとに異なるスレッドIDを生成する実装例です。

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib prefix="imst" uri="http://www.intra-mart.co.jp/taglib/imst" %>
<%@ taglib prefix="imui" uri="http://www.intra-mart.co.jp/taglib/imui" %>
<%@ page import="jp.co.intra_mart.foundation.context.Contexts" %>
<%@ page import="jp.co.intra_mart.foundation.context.model.AccountContext" %>
<%
//アシスタントID
String assistantId = "sample-rag-assistant";
// アカウントコンテキスト取得
AccountContext accountContext = Contexts.get(AccountContext.class);
// ユーザコードを取得
String userCd = accountContext.getUserCd();
// タイムスタンプを生成
long timestamp = System.currentTimeMillis();
// スレッドIDを生成
String threadId = "thread_" + userCd + "_" + assistantId + "_" + timestamp;
%>
<imui:head>
    <title>アシスタント実行サンプル - 動的制御</title>
    <meta http-equiv="X-Intramart-Secure-Token" content='<imst:imSecureToken mode="value" />' />
    <script src="im_copilot/js/extension.bundle.js"></script>
</imui:head>
<div id="assistant-area"></div>
<script>
    // アシスタントUIを登録
    window.imCopilot.register({
        targetElementId: "assistant-area",
        assistantId: "sample-rag-assistant",
        threadId: '<%=threadId%>',
        inputPlaceholder: "Input prompt"
    });
</script>
```

### 注意事項

1. **extension.bundle.jsの読み込み位置**
   - extension.bundle.jsは必ず<imui:head>タグ内で読み込んでください
   - body内で読み込むと、正常に動作しない可能性があります
2. **スレッドIDの管理**
   - threadIdは会話履歴を識別するために使用されます
   - 同じthreadIdを使用すると、以前の会話履歴が引き継がれます
   - 新しい会話を開始したい場合は、新しいthreadIdを生成してください
3. **セキュリティトークンの設定**
   - meta要素でX-Intramart-Secure-Tokenを必ず設定してください
   - `<imst:imSecureToken mode="value" />` を使用してトークン値を取得します

### トラブルシューティング

#### アシスタントが表示されない場合

以下の点を確認してください：

1. **extension.bundle.jsの読み込み位置**
   - <imui:head>タグ内で読み込まれているか確認
   - bodyタグ内での読み込みではないか
2. **タグライブラリの宣言**
   - タグライブラリの宣言（imst、imui）が記述されているか
   - タグライブラリが正しくインポートされているか
3. **セキュリティトークンの設定**
   - `<meta http-equiv="X-Intramart-Secure-Token">` が設定されているか
   - `<imart type="imSecureToken" mode="value" />` の記述が正しいか
4. **HTML要素の存在確認**

- targetElementIdで指定したID属性を持つ要素が存在するか
- DOMContentLoadedイベント発生後に実行されているか

5. **アシスタントID**
   - assistantIdが正しいか
   - アシスタントへのアクセス権限があるか

エラーメッセージの対処

**よくあるエラーと対処法**

| エラーメッセージ | 対処法 |
|---|---|
| “Error: Failed to find extension parent element” | 指定したtargetElementIdの要素が存在することを確認 |
| “Error: Assistant configuration error. Check definition and authorization settings.” | アシスタントIDが正しいか、アシスタントが存在するか、アシスタントへのアクセス権限があるか確認 |
| “TypeError: Cannot read properties of undefined” | extension.bundle.jsが<imui:head>タグ内で読み込まれているか確認 |
| “TypeError: Cannot read properties of null” | セキュリティトークンが正しく設定されているか確認 |

まとめ

JavaEE開発での「アシスタント実行 UI」の埋め込みは、わずか数行のコードで実現できます。
この機能を活用することで、独自のWebアプリケーションにAIアシスタント機能を簡単に統合できます。

関連ドキュメント

- TERASOLUNA Server Framework for Java (5.x) プログラミングガイド

# 独自アシスタントの作成

## 概要

IM-Copilot では、標準で提供されるアシスタントに加えて、独自のアシスタントを作成することが可能です。
業務要件に特化したアシスタントを構築することで、生成AIの活用効果をさらに高めることができます。

> **コラム**
>
> 標準で提供されるアシスタントについては、「*セットアップ（各種製品アシスタント）*」を参照してください。

### ロジックフローアシスタントの作成例

ロジックフローアシスタントは、IM-LogicDesigner を活用して、複雑な処理を実行するアシスタントです。
業務ロジックに沿った柔軟なフロー設計が可能で、より高度な対話や操作を実現できます。

#### ロジックフローアシスタントの作成例

このセクションでは、ロジックフローアシスタントの作成例について説明します。

作成するロジックフローアシスタントの概要

このセクションでは、IM-LogicDesigner を用いてアシスタントを作成する手順を解説します。
手順に従って進めることで、ストレージ内のファイルを情報源としてユーザの質問に回答するアシスタントを構築できます。

本ドキュメントでは、説明の都合上情報源をストレージ内のファイルに限定していますが、ローコード開発により変更が可能です。
例えば、このドキュメントを参考に作成したロジックフローアシスタントを変更することで、外部システムや独自のデータベースから情報を取得するように変更することもできます。

目次



## 作成するロジックフローアシスタントの利用想定

本ドキュメントで作成するロジックフローアシスタントは、指定されたストレージ内のファイルを検索し、その内容をもとに適切な回答を提供します。例えば、社内ドキュメント検索やナレッジベースの自動応答システムとして活用できます。

**主な用途**

- 社内マニュアルや手順書の検索・回答
- 製品仕様書の照会
- FAQ の自動応答システム

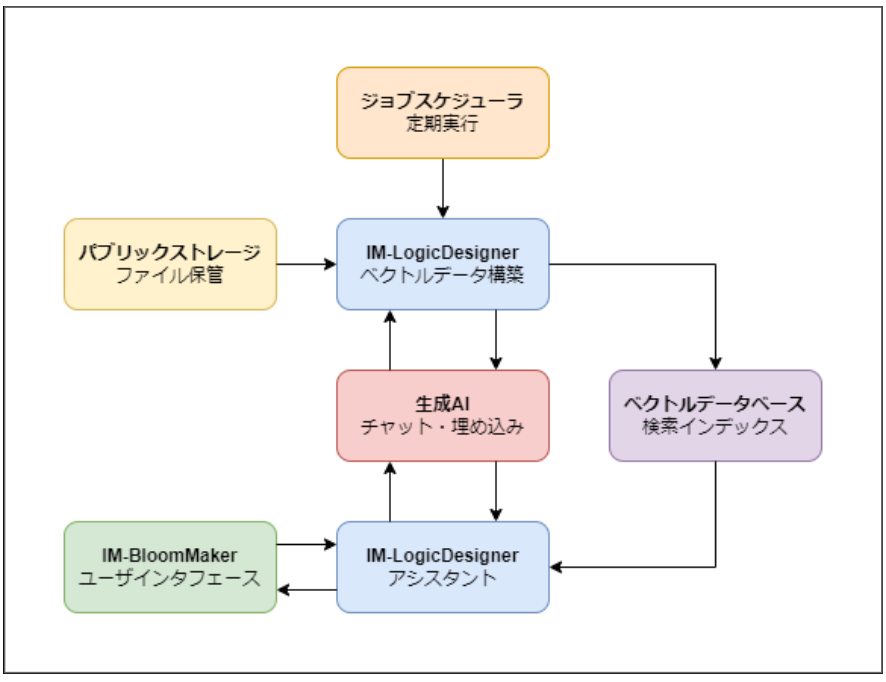
## 前提知識

本ガイドを理解するために、以下の知識があるとスムーズに進められます。

- IM-LogicDesigner の基本操作
  - IM-LogicDesigner の基本操作については「IM-LogicDesigner ユーザ操作ガイド」を参照してください。
- IM-BloomMaker の基本操作
  - IM-BloomMaker の基本操作については「IM-BloomMaker for Accel Platform ユーザ操作ガイド」を参照してください。
- RAG（Retrieval Augmented Generation）の概念

## 作成するロジックフローアシスタントの概要図

以下の図は、作成するロジックフローアシスタントの基本的な動作を示しています。



主要コンポーネント

| 機能名 | 説明 |
| --- | --- |
| IM-LogicDesigner | ベクトルデータの構築処理、および、アシスタントのロジックを管理 |
| IM-BloomMaker | チャット画面を提供 |
| パブリックストレージ | アシスタントが情報源として参照するファイルを格納 |
| ベクトルデータベース | 情報検索のためのデータ管理 |
| 生成AI | 自然な回答生成、ベクトル情報（埋め込み）の生成 |
| ジョブスケジューラ | ベクトルデータの定期的なデータ更新処理 |

### 事前設定

#### 生成AIの設定

作成するロジックフローアシスタントでは生成AIを活用します。利用するためには、以下の設定を事前に行う必要があります。

1. 生成AIサービスのセットアップ
   一つ以上の生成AIサービスがセットアップされている必要があります。
   *セットアップ（生成AI）* を参考にセットアップを行ってください。

2. 生成AI連携ドライバ設定
   セットアップ済みの生成AIサービスと連携するための設定です。
   *生成AI連携ドライバ設定* を参考に生成AIドライバの設定を行ってください。

3. 生成AI連携アクション設定
   本アシスタントでは IM-LogicDesigner タスク「チャット」、「埋め込み」を利用します。
   これらのタスクに対応する「チャット」、「埋め込み」のアクションを設定する必要があります。
   *生成AI連携アクション設定* を参考に生成AIアクションの設定を行ってください。

#### テキスト抽出設定

作成するロジックフローアシスタントでは、パブリックストレージ内のファイルのテキスト抽出にテキスト抽出機能（ND Universal Extractor）を利用します。
テキスト抽出設定では、テキスト抽出対象とするファイルについて設定が可能です。標準設定においてもテキスト抽出は可能ですが、必要に応じて設定を調整してください。
設定内容の詳細については、「設定ファイルリファレンス」 - 「テキスト抽出設定」 を参照してください。

#### ベクトルデータベースの設定

作成するロジックフローアシスタントでは、パブリックストレージ内のファイルを検索可能な形で管理するためにベクトルデータベースを使用します。
以下を参考にベクトルデータベースの設定を行ってください。

- 「テナント環境セットアップ」 - 「ベクトルデータベース接続情報」

#### パブリックストレージへの情報源となるファイルの配置

作成するロジックフローアシスタントでは、パブリックストレージ内の *sample_rag_assistant/files* ディレクトリ配下を検索対象として利用します。
ベクトルデータ構築を実行する前に、情報源とするファイルを配置してください。
パブリックストレージにファイルを配置する手段は以下の通りです。

- テナント管理機能のファイル操作を使用する
  - 「テナント管理者操作ガイド」-「ファイル操作を使用する」
- IM-LogicDesigner のタスク「パブリックストレージ取得」タスクを使用する
  - 「IM-LogicDesigner仕様書」-「パブリックストレージ取得」

### 構築ステップ

本ドキュメントでは、以下のステップに従ってアシスタントを構築します。

1. **ベクトルデータの作成**
   - パブリックストレージ内のファイルをベクトル化し、検索に利用できるように準備します。
   - ベクトル情報を登録するロジックフローを作成し、ジョブスケジューラで定期的に実行するように設定します。

2. **アシスタントの実装**

- ユーザの質問を処理し、適切な回答を生成するロジックフローを作成します。
- アシスタント定義を作成し、ロジックフローをアシスタントとして動かせるように設定します。

3. **画面の作成**
   - ユーザがアシスタントを利用できる UI を作成します。

これらの手順に沿って進めることで、アシスタントを作成できます。

### 本アシスタント資材のダウンロード

本手順に従い作成したアシスタントの資材は、以下からダウンロードできます。

**アシスタントの資材**

> **コラム**
> 本資材は、アシスタントの作成手順を解説するためのサンプル資材です。アシスタントの動作確認や追加開発のためにご利用ください。

zip ファイルには以下のファイルが含まれています。

- im_logicdesigner-data.zip
  - IM-LogicDesigner のインポートファイル
- assistant_definition.json
  - アシスタント定義のインポートファイルです
- im_bloommaker-data.zip
  - IM-BloomMaker のインポートファイルです
- job-scheduler.xml
  - ジョブスケジューラのインポートファイルです

zip ファイルを解凍し、以下の順で資材をインポートしてください。

1. IM-LogicDesigner
2. アシスタント定義
3. IM-BloomMaker
4. ジョブスケジューラ

> **コラム**
> 本インポート資材には認可設定が含まれていません。インポート実施後に「アシスタント定義」、「IM-BloomMaker ルーティング定義一覧」から認可設定を行ってください。

> **コラム**
> アシスタントの会話履歴は利用を重ねることで蓄積されます。
> 実運用では履歴削除ジョブを設定することで、古い履歴を定期的に削除する運用が可能です。
> 詳細は「*運用上の考慮事項*」を参照してください。

### IM-LogicDesigner 資材のインポート

IM-LogicDesigner のインポート手順については以下を参照してください。

- 「IM-LogicDesigner ユーザ操作ガイド」 - 「インポートを行う」

### アシスタント定義のインポート

アシスタント定義のインポート手順については以下を参照してください。

- 「*アシスタント定義のインポート*」

### IM-BloomMaker 資材のインポート

IM-BloomMaker のインポート手順については以下を参照してください。

- 「IM-BloomMaker for Accel Platform ユーザ操作ガイド」 - 「定義ファイルをインポートする」

### ジョブスケジューラ資材のインポート

ジョブスケジューラ資材のインポートは、ジョブを利用して実施します。以下で手順を確認してください。

1. パブリックストレージ直下に *job-scheduler.xml* を配置します。
   以下を参考にファイルを配置してください。
   - 「テナント管理者操作ガイド」-「ファイル操作を使用する」

2. ジョブを確認します。
   「サイトマップ」→「テナント管理」→「ジョブ管理」→「ジョブ設定」をクリックし「ジョブ管理」画面を表示します。
   左側のジョブ一覧から「テナントマスタ」→「インポート」→「ジョブインポート」を選択クリックします。
   実行時の情報内の実行パラメータのキー *file* が *job-scheduler.xml* であることを確認します。

3. ジョブネットを確認・実行します。
   「サイトマップ」→「テナント管理」→「ジョブ管理」→「ジョブネット設定」をクリックし「ジョブネット管理」画面を表示します。
   左側のジョブネット一覧から「テナントマスタ」→「インポート」→「ジョブインポート」を選択クリックします。
   実行時の情報内の実行パラメータのキー *file* 存在しないことを確認します。
   下部の「即時実行」ボタンをクリックして「ジョブインポート」ジョブネットを実行します。

ロジックフローアシスタント構築手順
ベクトルデータ構築ジョブの作成

作成するロジックフローアシスタントでは、ストレージ内のファイルを情報源とするため、ファイルの内容をベクトル化し、ベクトルデータベースに登録する必要があります。
本セクションでは、ベクトルデータを構築するためのロジックフローを作成する手順、および、ジョブスケジューラを利用した自動実行の設定方法を説明します。

目次

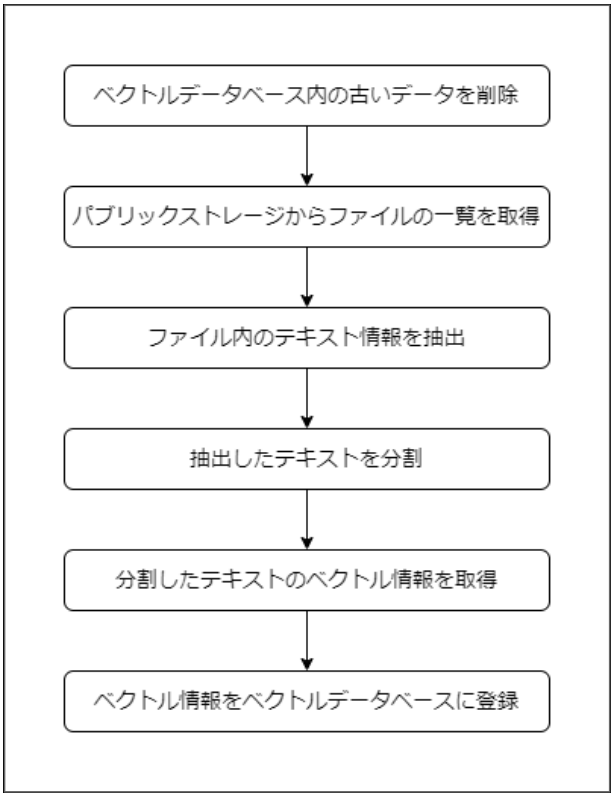


## ベクトルデータの構築を行うロジックフローの作成

IM-LogicDesigner を使用して、ベクトルデータを定期的に構築・更新するロジックフローを作成します。

### 作成するロジックフローの概要

本ロジックフローは、以下の手順でベクトルデータを構築します。




ベクトルデータの作成処理は、埋め込み処理でサーバ通信が発生する API を利用するため時間がかかります。
本ドキュメントでは、説明の都合上既存のベクトルデータを削除した後に新たなベクトルデータを構築する手順を記載しています。
そのため、構築が完了するまでの間はアシスタントの情報源が一部欠落します。

なお、ロジックフローの変更を行うことでこの参照不可時間を短縮する処理に変更することも可能です。

- 変更例
  1. 登録するベクトルデータを事前に作成
  2. ベクトルデータベースから既存のベクトルデータを削除
  3. 事前に作成したベクトルデータをベクトルデータベースに登録

### 設定手順

IM-LogicDesigner を利用して、ベクトルデータを構築するロジックフローを作成する手順を説明します。
本説明で使用しているロジックフローのキャプチャ画像は、一部タスク・エレメントのラベルの値を変更しています。

### 各種定義の設定

1. ロジックフロー新規作成画面を開きます。
   「サイトマップ」→「LogicDesigner」→「フロー定義一覧」をクリックし、「ロジックフロー定義一覧」画面を表示します。
   ロジックフロー定義一覧画面ツールバーの「ロジックフロー新規作成」をクリックします。

2. 定数の設定を行います。
   「ロジックフロー定義編集」画面ツールバーの「定数設定」をクリックして定数設定ダイアログを表示します。
   以下の定数を設定します。

**定数一覧**

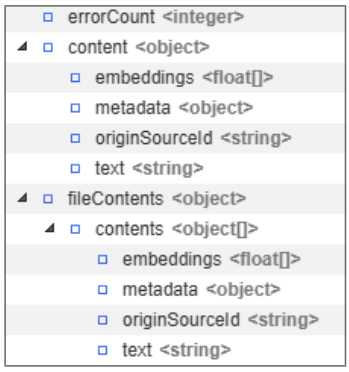
| 定数ID | 定数値 | 説明 |
|---|---|---|
| ERROR_MESSAGE | *The embedding process failed.* | 埋め込み処理でエラーが発生した場合に利用するメッセージです。 |
| ONE | *1* | 1を表す定数です。カウントのインクリメントに利用します。 |
| RETRY_COUNT | *3* | 埋め込みエラー時にリトライする回数です。 |
| TARGET_STORAGE_PATH | *sample_rag_assistant/files* | 参照情報とするファイルが格納されているパブリックストレージのディレクトリです。 |

| 定数ID | 定数値 | 説明 |
|---|---|---|
| VECTORSTORE_CATEGORY | *sample_rag_assistant* | ベクトルデータベースのカテゴリです。 |

3. 変数の設定を行います。
「ロジックフロー定義編集」画面ツールバーの「変数設定」をクリックして変数設定ダイアログを表示します。
以下の変数を設定します。

**変数一覧**

| 変数 | 型 | 説明 |
|---|---|---|
| errorCount | integer | 埋め込み処理が連続して失敗した際に利用するエラー回数です。 |
| content | object | ベクトルデータベースに登録する 1 コンテンツを格納します。 |
| content.embeddings | float[] | |
| content.metadata | object | |
| content.originSourceId | string | |
| content.text | string | |
| fileContents | object | ベクトルデータベースに登録する 1 ファイルに含まれるコンテンツを格納します。 |
| fileContents.contents | object[] | |
| fileContents.contents.embeddings | float[] | |
| fileContents.contents.metadata | object | |
| fileContents.contents.originSourceId | string | |
| fileContents.contents.text | string | |

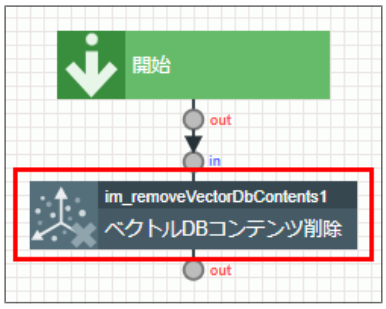



JSON 入力で設定する場合は以下の JSON をコピーし、変数設定ダイアログの「JSON入力」をクリックしてサンプル JSON の入力ダイアログに貼り付けてください。
JSON 入力を利用して変数を設定した場合、*errorCount* のデータ型が *double* に、*content.embeddings*、および、*fileContents.contents.embeddings* のデータ型が *double[]* に設定されます。
*errorCount* のデータ型を *integer* に、*content.embeddings*、および、*fileContents.contents.embeddings* のデータ型を *float[]* に手動で変更してください。

```
{
  "errorCount": 0,
  "content": {
    "embeddings": [
      0
    ],
    "metadata": {},
    "originSourceId": "",
    "text": ""
  },
  "fileContents": {
    "contents": [
      {
        "embeddings": [
          0
        ],
        "metadata": {},
        "originSourceId": "",
        "text": ""
      }
    ]
  }
}
```

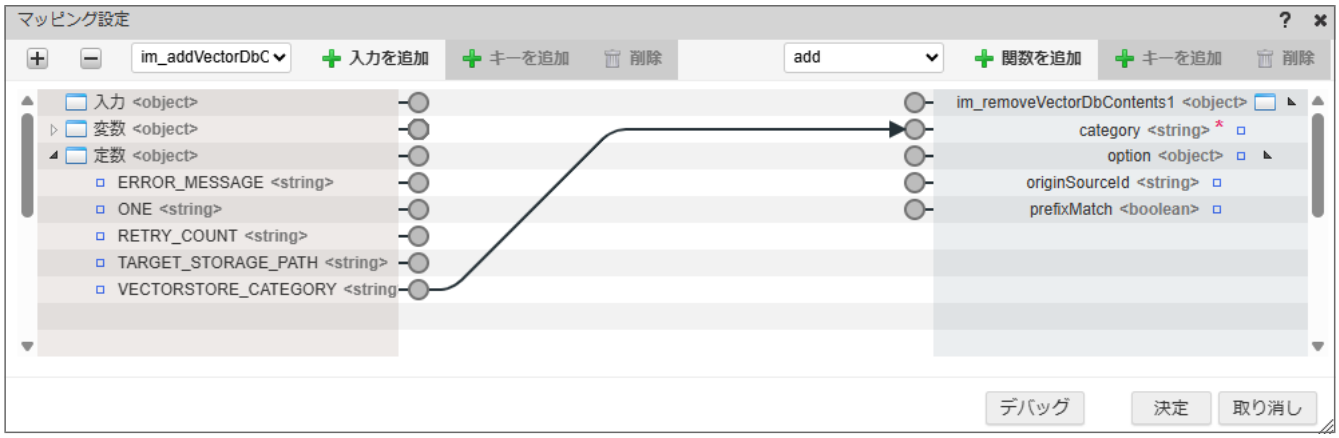

ベクトルデータベース内の古いデータの削除

4. ベクトルデータの削除タスクを追加します。




既に存在するベクトルデータを削除するために対象のカテゴリのベクトルデータを削除するタスクを追加します。
パレットから「IM-Copilot」→「ベクトルDBコンテンツ削除」をクリックしロジックフローに追加します。
「開始」エレメントの *out* を追加した「ベクトルDBコンテンツ削除」の *in* に接続します。
追加した「ベクトルDBコンテンツ削除」を選択し、マッピング設定を行います。

- 定数 *VECTORSTORE_CATEGORY* をタスクの入力値 *category* に設定します。



コラム

ベクトルデータの削除方法は、カテゴリ単位で削除する方法の他に、リソースID指定で削除する方法があります。

本サンプル資材では、ベクトルデータベースにコンテンツを登録する際に、ストレージ内のファイル名をリソースID（originSourceId）として設定しています。
※なお、リソースIDの管理方法については規定していません。開発者側で適切に実装・運用してください。

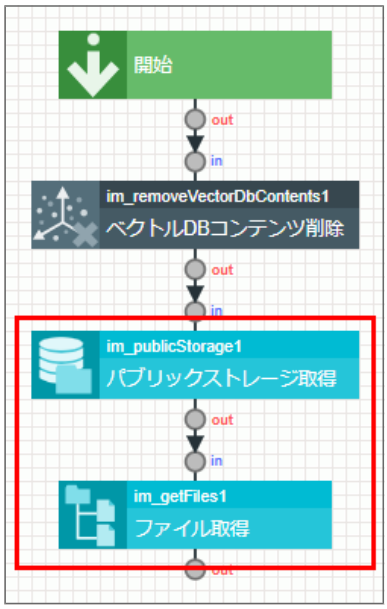
ベクトルデータを最新化する際に、カテゴリ内のすべてのベクトルデータを削除せず、特定の情報源に関連するベクトルデータのみを削除したい場合は、リソースID指定で削除する方法をご利用いただけます。
リソースID指定の削除を利用しロジックフローをカスタマイズすることで、ファイルの最終更新日時で条件分岐させ、更新があったファイルのベクトルデータのみを最新化する等の応用が可能です。この場合、更新されていないファイルはテキスト抽出やテキスト情報をベクトル化する処理をスキップできるため、処理時間や生成AIサービス使用料を削減できます。

また、リソースID指定の削除を *ベクトル情報をベクトルデータベースに登録* する前に実行することで、登録するコンテンツと同じリソースIDを持つコンテンツのみを削除してから新しいベクトルデータを登録するため、アシスタントの情報源が一部欠落する時間を短縮することも可能です。

タスクの詳細は「IM-LogicDesigner仕様書」-「タスク一覧」-「IM-Copilot」 -「ベクトルデータベースコンテンツ削除」を参照してください。

パブリックストレージからファイルの一覧を取得

5. パブリックストレージのファイル一覧を取得するタスクを追加します。

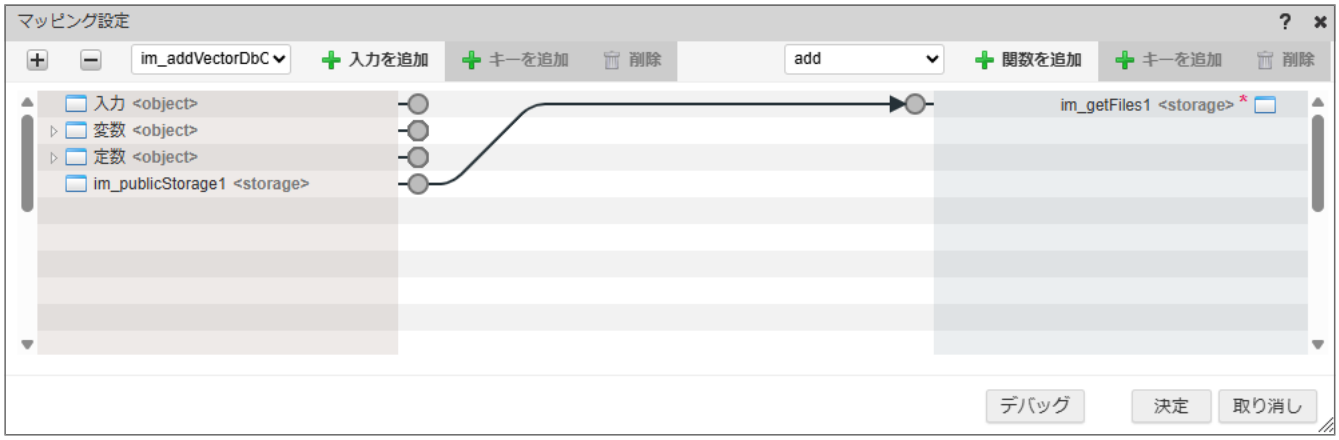



生成 AI が回答を作成する際に情報源とするデータをパブリックストレージから取得します。
パブリックストレージ内のファイル一覧を取得するためのタスクを追加します。
パレットから「ストレージ操作」→「パブリックストレージ取得」をクリックしロジックフローに追加します。
「ベクトルDBコンテンツ削除」タスクの *out* を追加した「パブリックストレージ取得」の *in* に接続します。
追加した「パブリックストレージ取得」を選択し、マッピング設定を行います。

- 定数 *TARGET_STORAGE_PATH* をタスクの入力値に設定します。

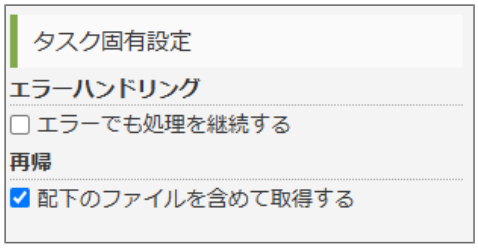

続いて、パレットから「ストレージ操作」→「ファイル取得」をクリックしロジックフローに追加します。
「パブリックストレージ取得」タスクの *out* を追加した「ファイル取得」の *in* に接続します。
追加した「ファイル取得」を選択し、マッピング設定を行います。

- エイリアス「*im_publicStorage1*」を追加しタスクの入力値に設定します。

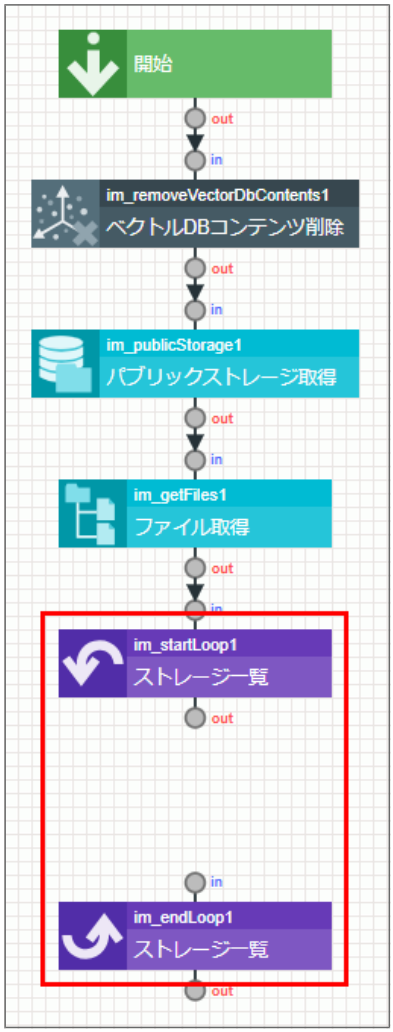



続いてファイル取得タスクのタスク固有設定を行います。

- 配下のファイルを含めて取得するを有効にします。




6. 繰り返し制御エレメントを追加します。

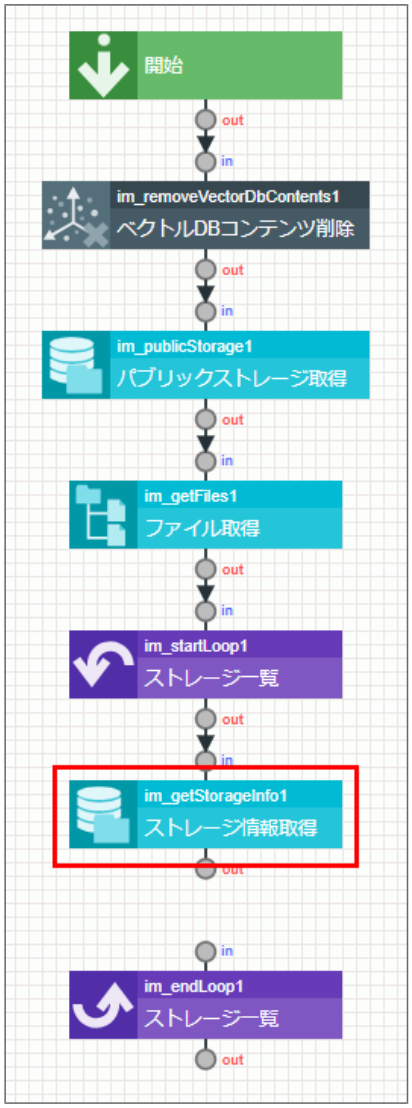



各ファイルの処理を行うために、繰り返し制御エレメントを追加します。
パレットから「基本」→「繰り返し開始」をクリックしロジックフローに追加します。
「ファイル取得」タスクの *out* を追加した「繰り返し開始」の *in* に接続します。
追加した「繰り返し」を選択し、タスク固有設定を行います。

- 初期化する変数名に *fileContents* を設定します。
  - これにより、各ファイルの処理を行う際に、ファイルの内容を格納する変数を初期化します。
- 繰り返し対象に *im_getFiles1* を設定します。
  - これにより、取得したファイル一覧を繰り返し処理します。

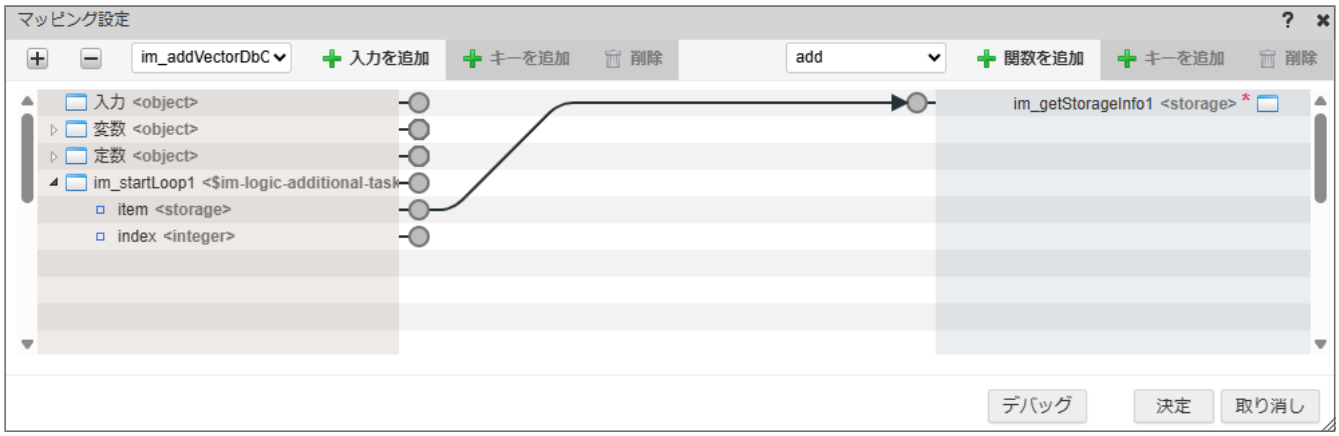

ファイル内のテキスト情報を抽出

7. ストレージ情報取得タスクを追加します。

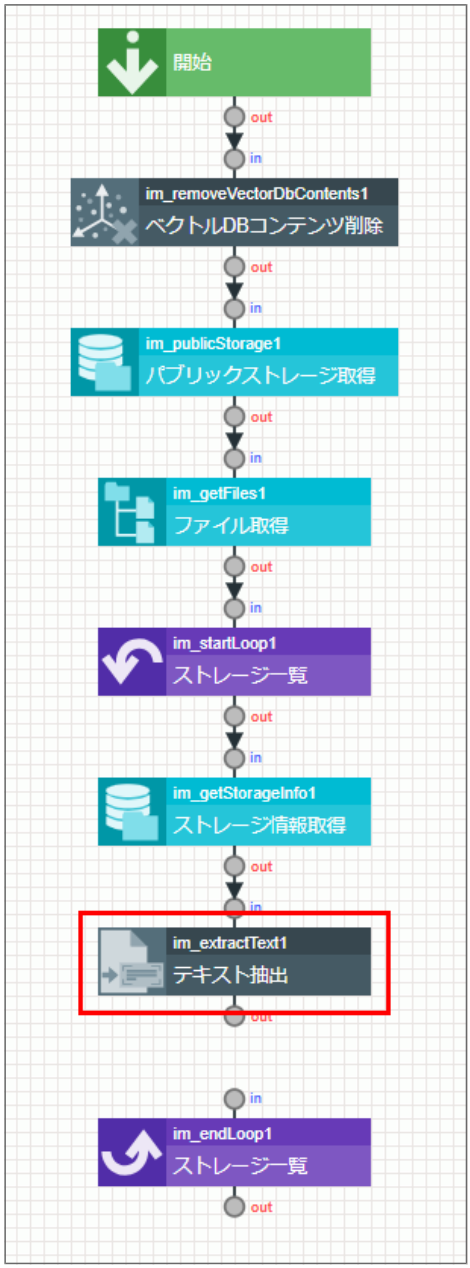



ファイル情報を取得するためのタスクを追加します。取得したファイル情報はテキスト抽出タスクの入力値として利用します。
パレットから「ストレージ操作」→「ストレージ情報取得」をクリックしロジックフローに追加します。
「繰り返し開始」エレメントの *out* を追加した「ストレージ情報取得」の *in* に接続します。
追加した「ストレージ情報取得」を選択し、マッピング設定を行います。

- エイリアス「*im_startLoop1*」を追加します。
  - *item* をタスクの入力値に設定します。

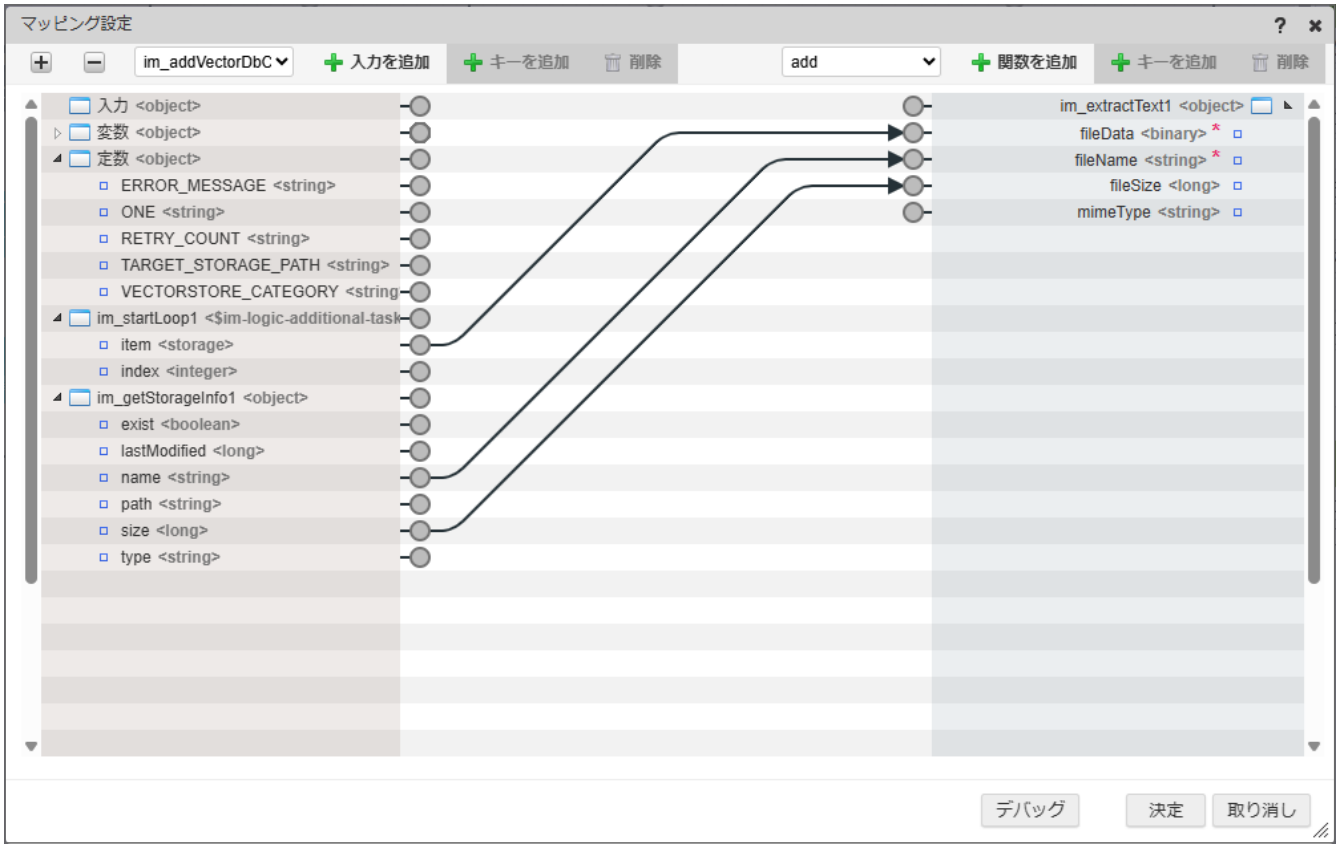



8. テキスト抽出タスクを追加します。




ファイルからテキスト情報を抽出するためのタスクを追加します。
パレットから「IM-Copilot」→「テキスト抽出」をクリックしロジックフローに追加します。
「ストレージ情報取得」タスクの *out* を追加した「テキスト抽出」の *in* に接続します。
追加した「テキスト抽出」を選択し、マッピング設定を行います。

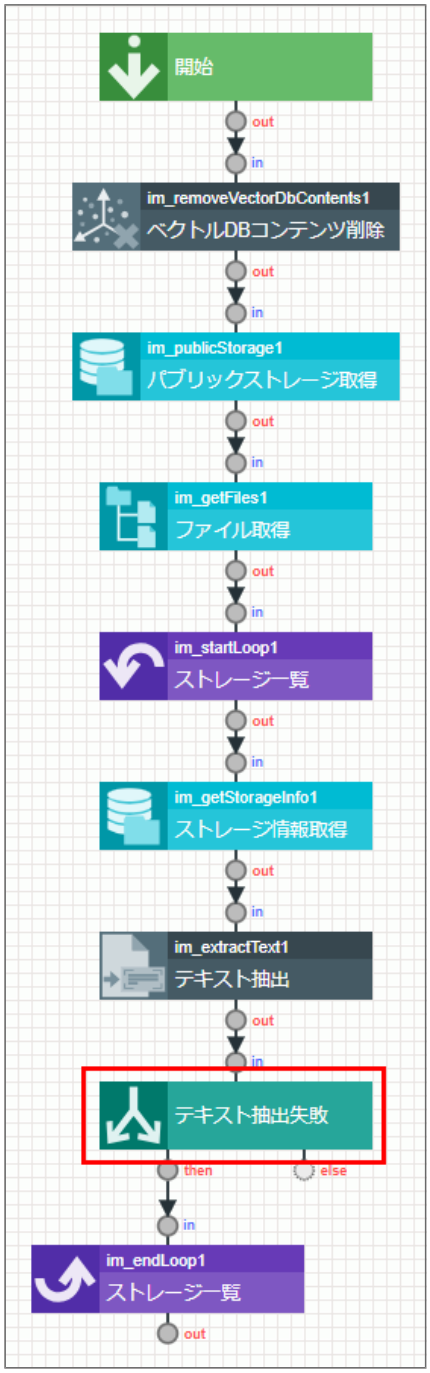
- エイリアス「*im_startLoop1*」を追加します。
  - *item* をタスクの入力値 *fileData* に設定します。
- エイリアス「*im_getStorageInfo1*」を追加します。
  - *name* をタスクの入力値 *fileName* に設定します。
  - *size* をタスクの入力値 *fileSize* に設定します。




続いて「テキスト抽出」タスクのタスク固有設定を行います。

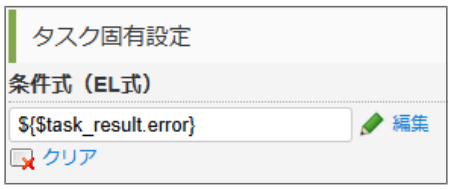
- エラーでも処理を続行するを有効にします。


9. テキスト抽出でエラーが発生した場合の処理を追加します。




テキスト抽出ではテキスト抽出設定の内容に準じてテキストを抽出します。テキスト抽出タスクにテキスト抽出が行えないファイルを指定した場合、エラーが発生します。
エラーが発生した場合に次のファイルの処理を行うための制御を追加します。
パレットから「基本」→「分岐」をクリックしロジックフローに追加します。
「テキスト抽出」タスクの *out* を追加した「分岐」の *in* に接続します。
追加した「分岐」を選択し、タスク固有設定を行います。

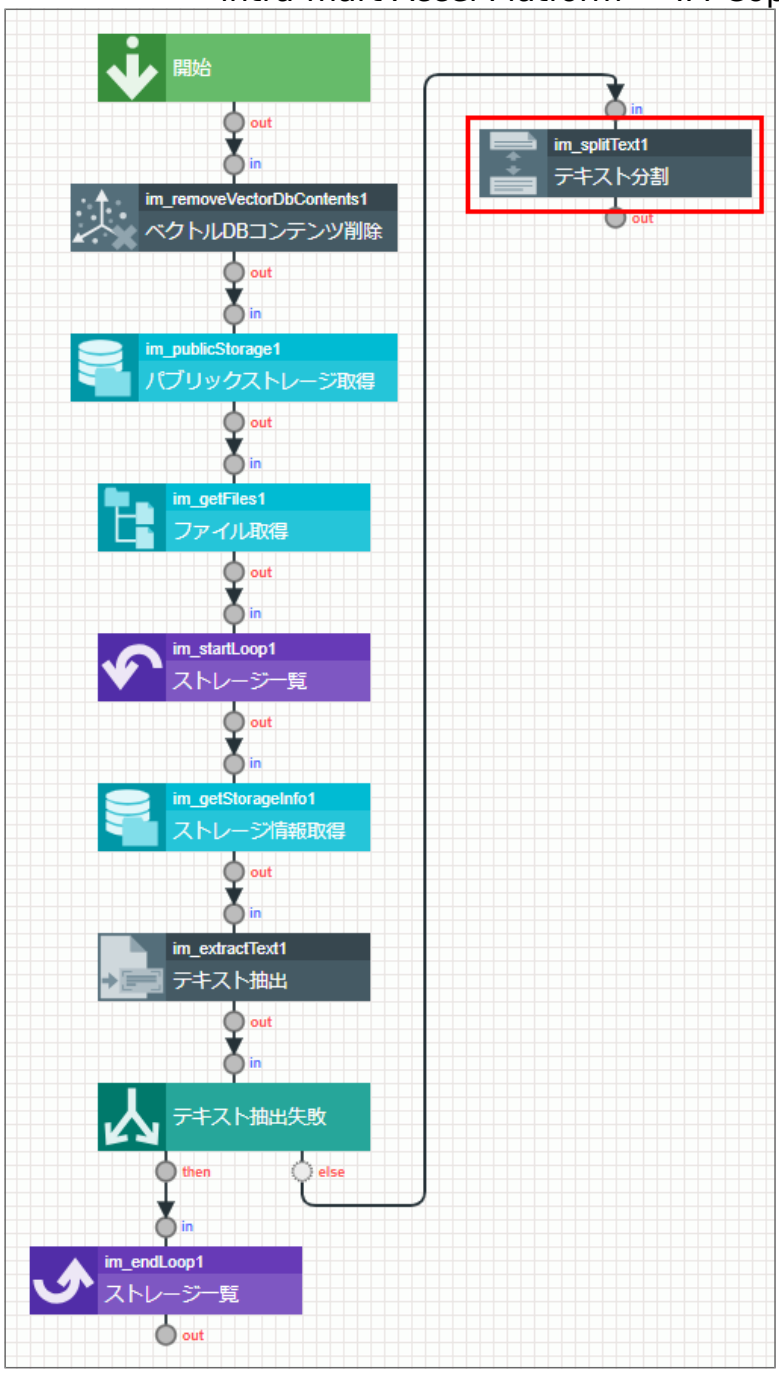
- 条件式（EL式）に *${$task_result.error}* を設定します。
  - *$task_result* は処理結果情報です。直前の処理の処理結果が格納されます。*error* にはエラーが発生した場合に *true* が設定されます。




エラーが発生した場合に次のファイルの処理を行うためのシーケンスを追加します。
追加した「分岐」の *then* に「繰り返し終了」エレメント（*im_endLoop1*）の *in* を接続します。

抽出したテキストを分割

10. テキスト分割タスクを追加します。

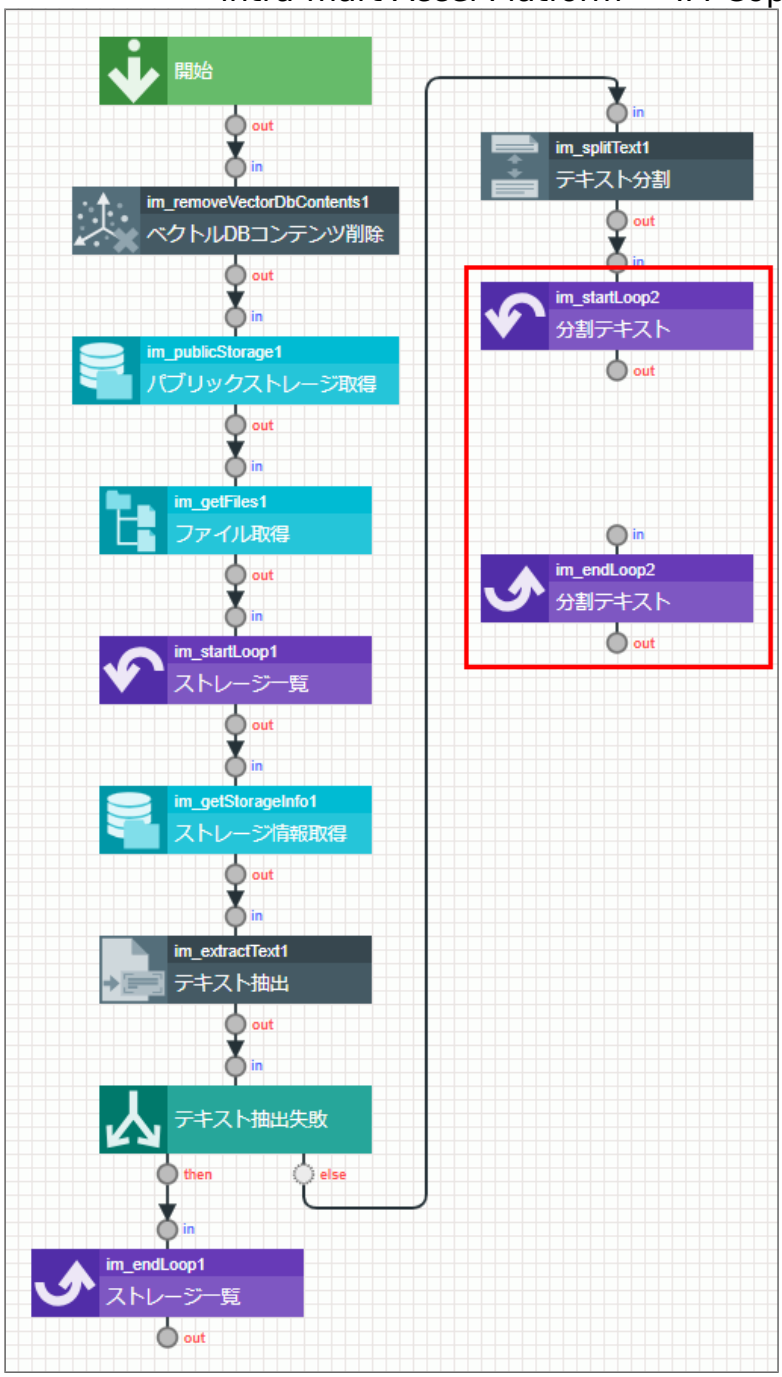



抽出したテキストを分割するためのタスクを追加します。
パレットから「IM-Copilot」→「テキスト分割」をクリックしロジックフローに追加します。
「分岐」エレメントの *else* を追加した「テキスト分割」の *in* に接続します。
追加した「テキスト分割」を選択し、マッピング設定を行います。

- エイリアス「*im_extractText1*」を追加しタスクの入力値 *text* に設定します。


分割したテキストをベクトル化

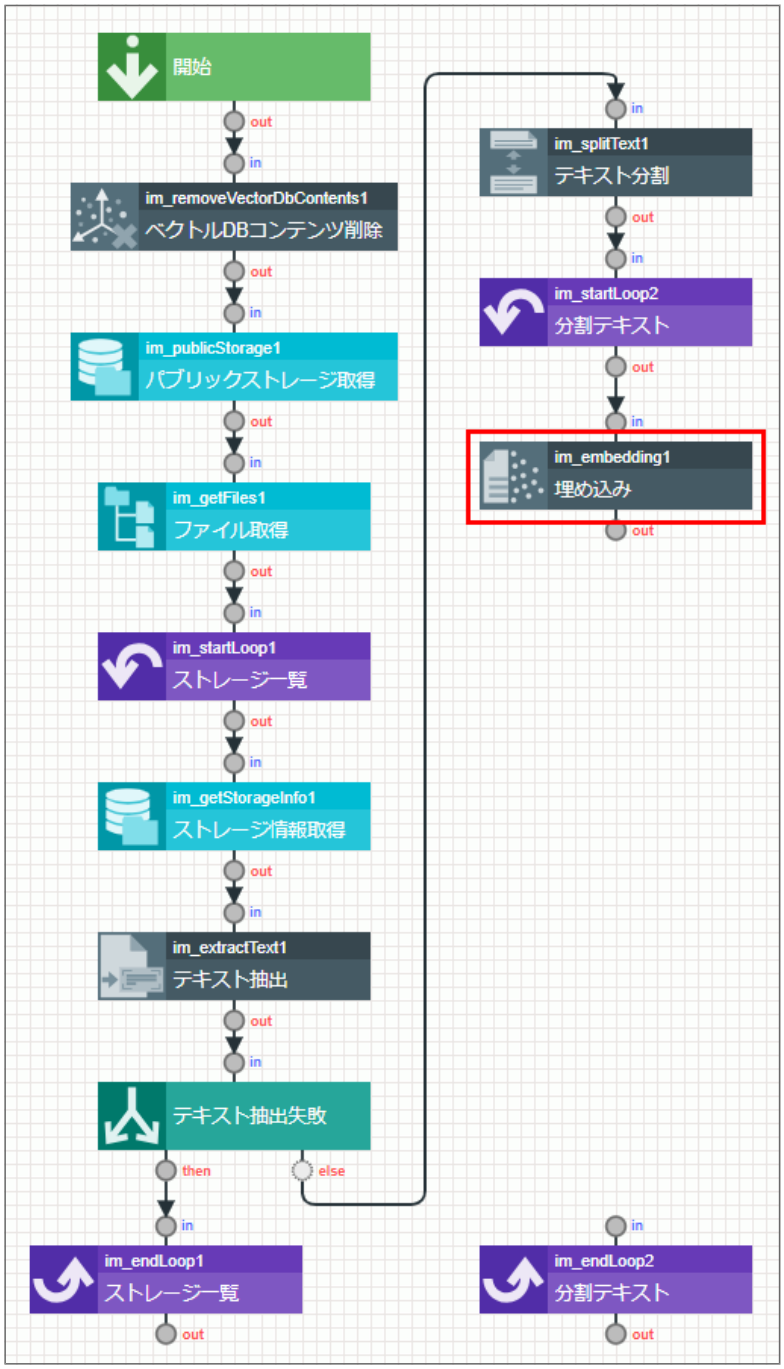
11. 繰り返し制御エレメントを追加します。




テキスト分割した情報をベクトル化するための繰り返し制御エレメントを追加します。
パレットから「基本」→「繰り返し開始」をクリックしロジックフローに追加します。
「テキスト分割」タスクの *out* を追加した「繰り返し開始」の *in* に接続します。
追加した「繰り返し」を選択し、タスク固有設定を行います。

- 初期化する変数名に *content*、 *errorCount* を設定します。
  - これにより、各テキストのベクトル化処理を行う際に、ベクトルデータを格納する変数を初期化します。また、埋め込み処理エラー回数を初期化します。
- 繰り返し対象に *im_splitText1* を設定します。
  - これにより、分割したテキスト情報を繰り返し処理します。

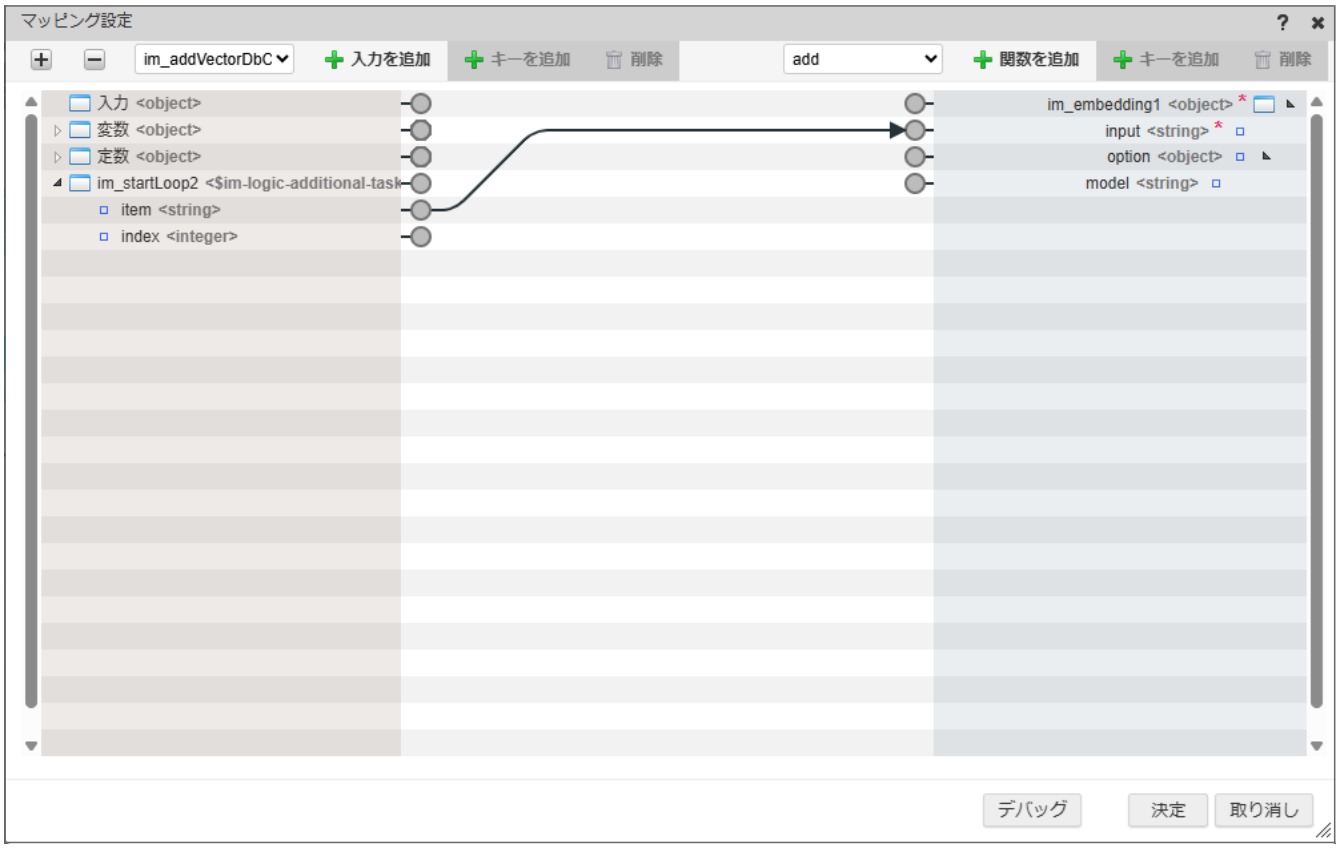

12. 埋め込みタスクを追加します。




テキスト情報をベクトル化するためのタスクを追加します。
パレットから「IM-Copilot」→「埋め込み」をクリックしロジックフローに追加します。
「繰り返し開始」エレメントの *out* を追加した「埋め込み」の *in* に接続します。

追加した「埋め込み」を選択し、マッピング設定を行います。

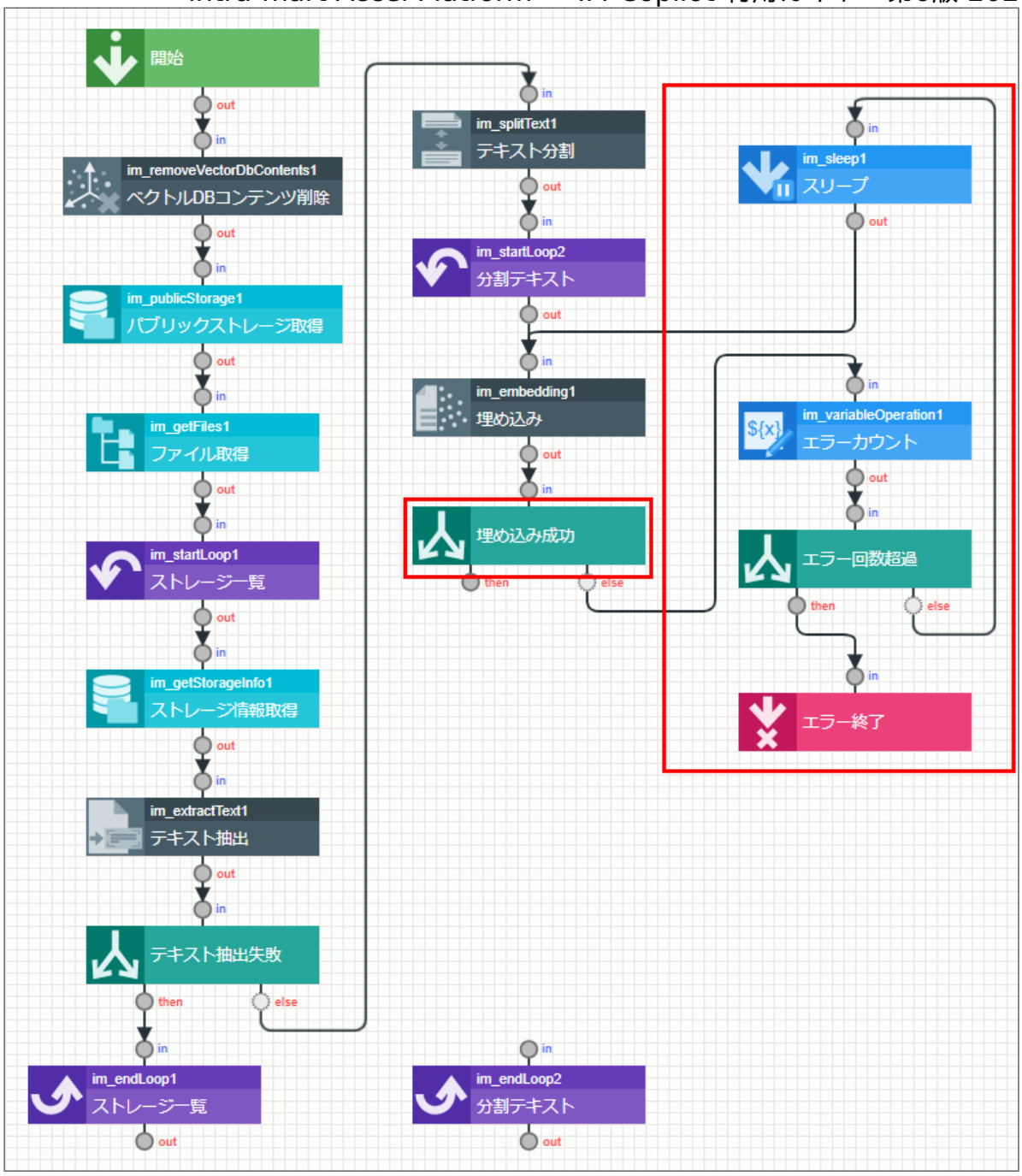
- エイリアス「*im_startLoop2*」を追加します。
  - *item* をタスクの入力値 *input* に設定します。




続いて「埋め込み」タスクのタスク固有設定を行います。

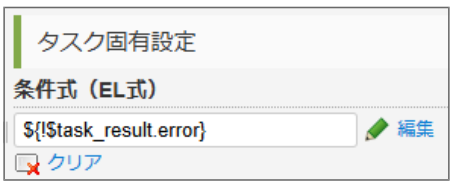
- エラーでも処理を続行するを有効にします。


13. 埋め込みでエラーが発生した場合の処理を追加します。



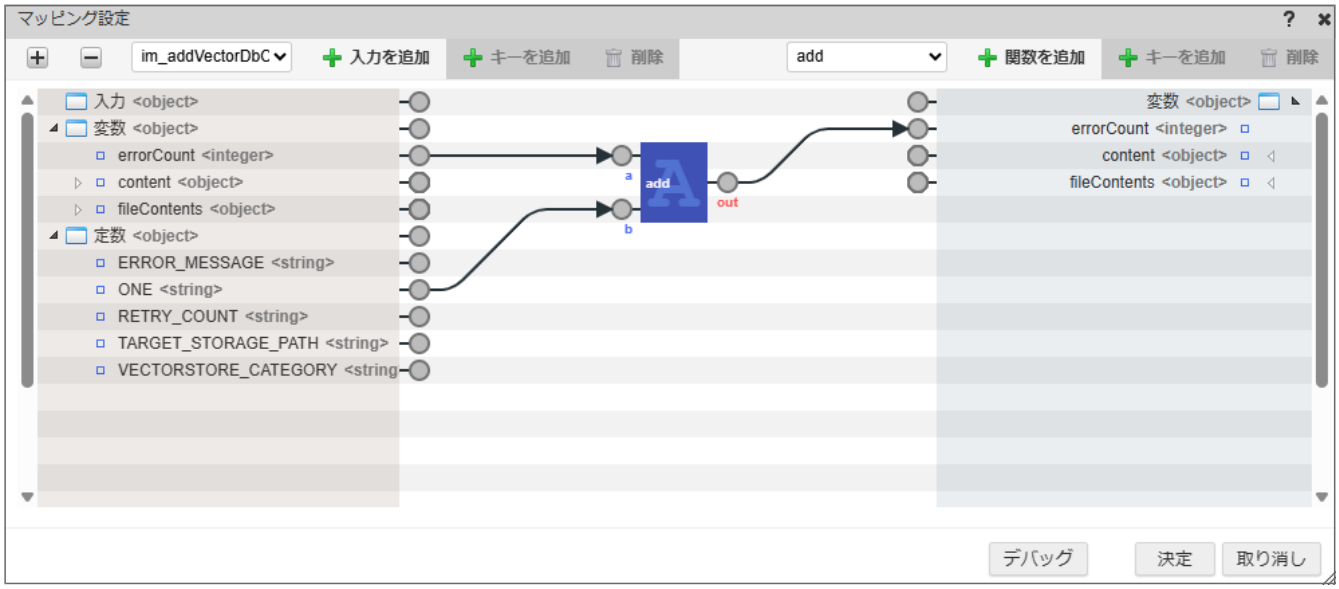

埋め込み処理は AI サービスへのリクエストを行う処理です。本フローでは繰り返し処理で連続して埋め込み処理を行うためレート制限などのエラーが発生する可能性があります。
そのため、エラーが発生した場合にリトライ処理を行うための制御を追加します。
パレットから「基本」→「分岐」をクリックしロジックフローに追加します。
「埋め込み」タスクの *out* を追加した「分岐」の *in* に接続します。
追加した「分岐」を選択し、タスク固有設定を行います。

- 条件式（EL式）に *${!$task_result.error}* を設定します。
  - *$task_result* は処理結果情報です。直前の処理の処理結果が格納されます。*error* にはエラーが発生した場合に *true* が設定されます。
  - *!* は否定を表します。エラーが発生していない場合に *true* になるように設定します。




続いて、パレットから「基本」→「変数操作」をクリックしロジックフローに追加します。
「分岐」エレメント（*im_gateway1*）の *else* に追加した「変数操作」の *in* を接続します。
エラーカウントをインクリメントします。
追加した「変数操作」を選択し、マッピング設定を行います。

- 数値演算マッピング関数 *add* を追加します。
  - 入力値 *a* に変数 *errorCount* を設定します。
  - 入力値 *b* に定数 *ONE* を設定します。
  - 出力値をタスクの入力値 *errorCount* に設定します。




続いて、パレットから「基本」→「分岐」をクリックしロジックフローに追加します。
「変数操作」タスクの *out* を追加した「分岐」の *in* に接続します。
追加した「分岐」を選択し、タスク固有設定を行います。

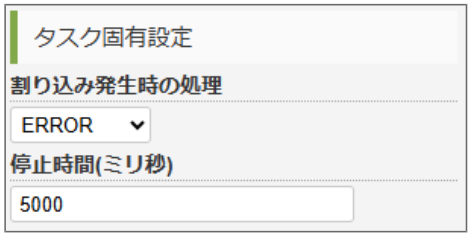
- 条件式（EL式）に *${$variable.errorCount>=$const.RETRY_COUNT}* を設定します。
  - *errorCount* が *RETRY_COUNT* 以上の場合に *true* になるように設定します。


続いて、パレットから「基本」→「エラー終了」をクリックしロジックフローに追加します。
「分岐」エレメント（*im_gateway2*）の *then* に追加した「エラー終了」の *in* を接続します。
追加した「エラー終了」を選択し、タスク固有設定を行います。

- エラーメッセージに *${$const.ERROR_MESSAGE}* を設定します。

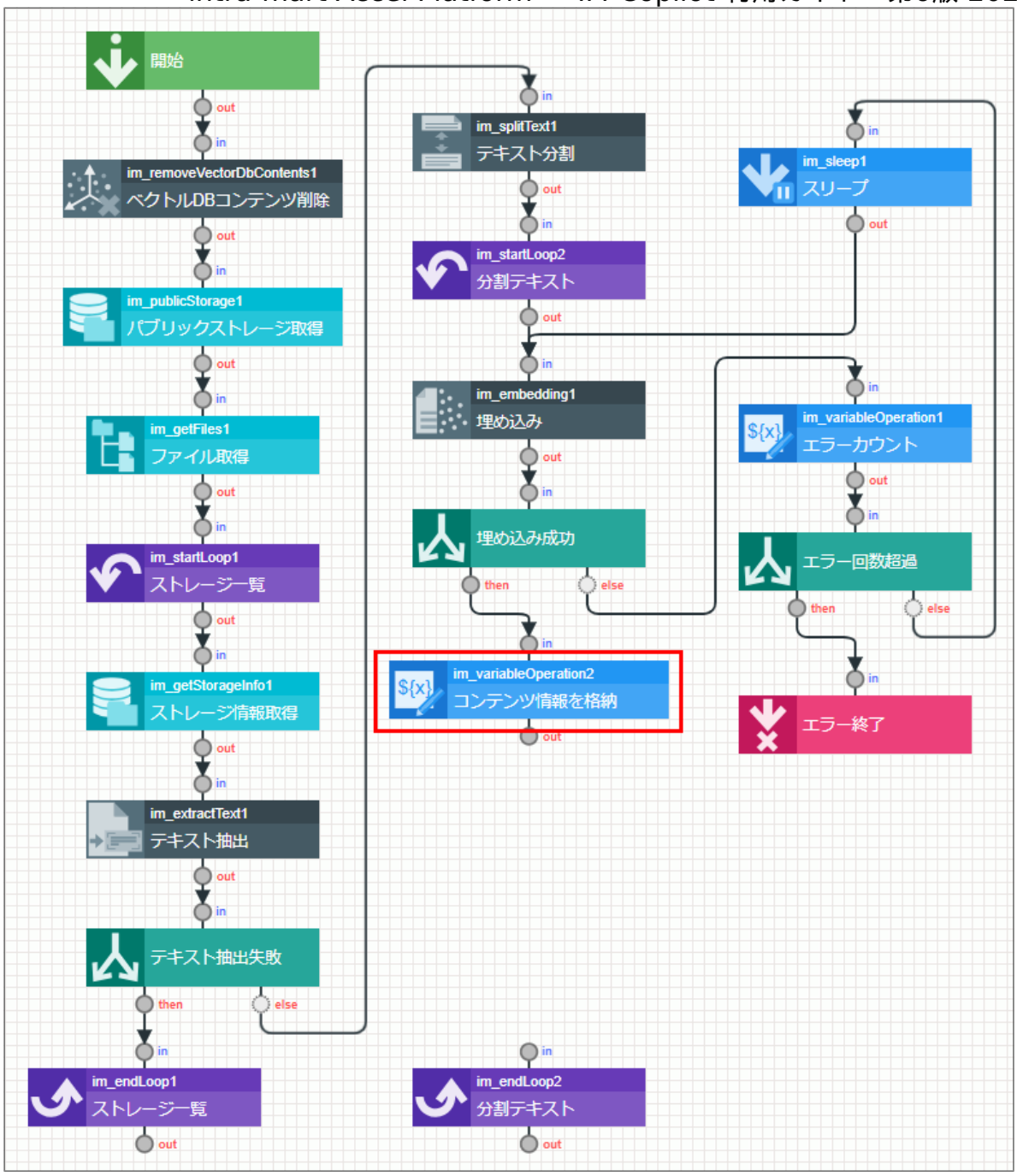

続いて、パレットから「汎用タスク」→「スリープ」をクリックしロジックフローに追加します。
「変数操作」タスクの後に追加した「分岐」エレメント（*im_gateway2*）の *else* に追加した「スリープ」の *in* を接続します。
追加した「スリープ」を選択し、タスク固有設定を行います。

- 停止時間(ミリ秒)に *5000* を設定します。



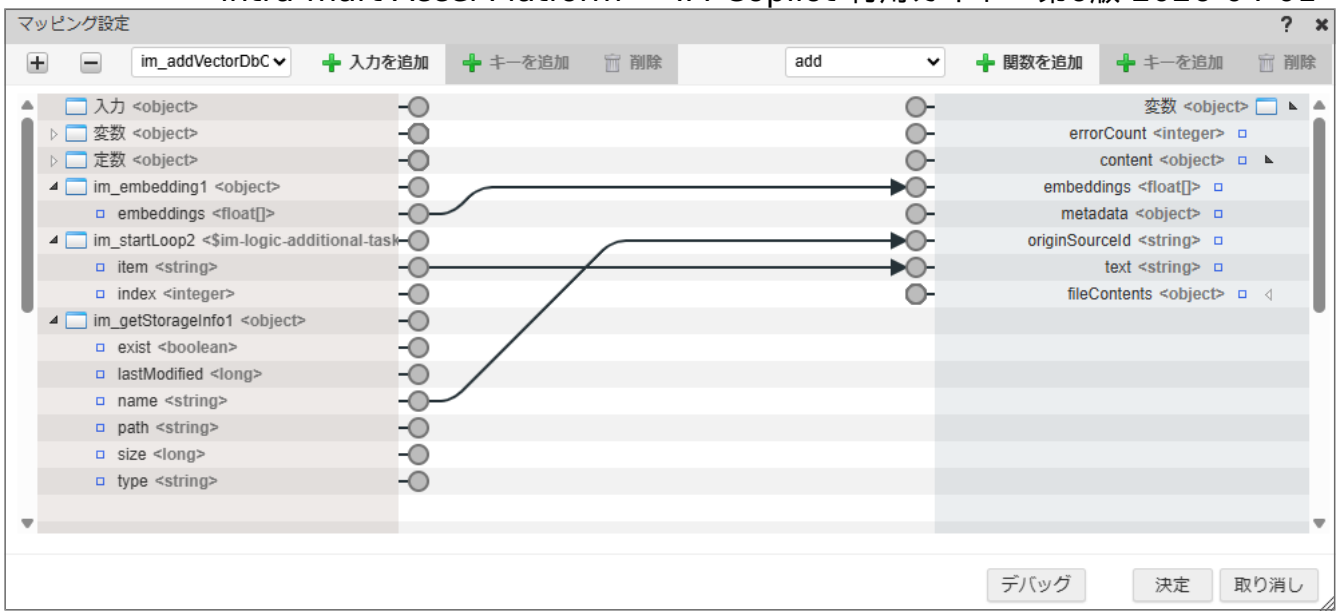

スリープ後に再度埋め込み処理を行います。
「スリープ」タスクの *out* を「埋め込み」の *in* に接続します。

14. 変数操作タスクを追加します。

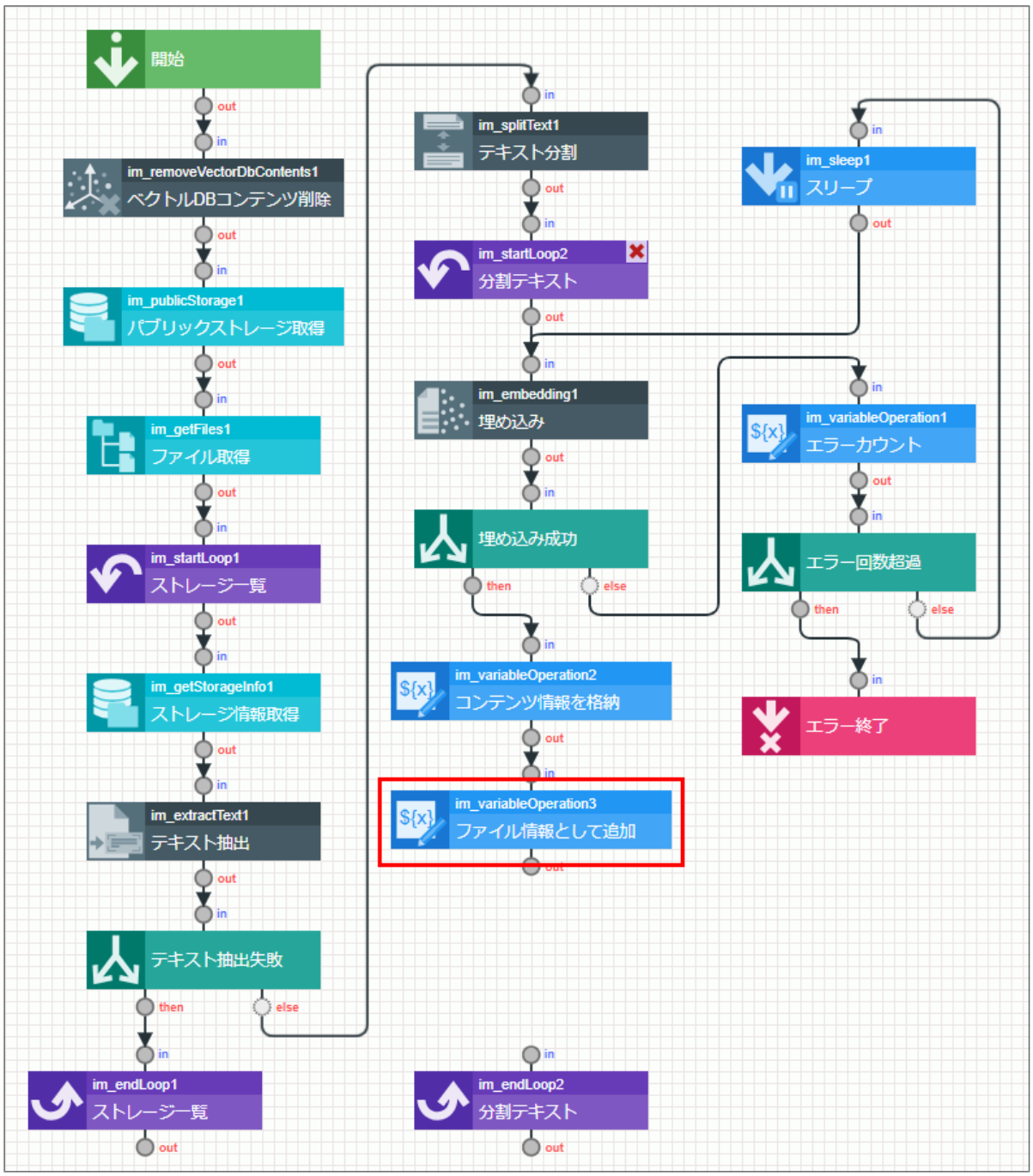



コンテンツ情報を変数に格納するため、変数操作タスクを追加します。
パレットから「基本」→「変数操作」をクリックしロジックフローに追加します。
「埋め込み」タスクの後の「分岐」エレメント（*im_gateway1*）の *then* に追加した「変数操作」の *in* を接続します。
追加した「変数操作」を選択し、マッピング設定を行います。

- エイリアス「*im_embedding1*」を追加します。
  - *embeddings* をタスクの入力値 *content.embeddings* に設定します。
- エイリアス「*im_startLoop2*」を追加します。
  - *item* をタスクの入力値 *content.text* に設定します。
- エイリアス「*im_getStorageInfo1*」を追加します。
  - *name* をタスクの入力値 *content.originSourceId* に設定します。



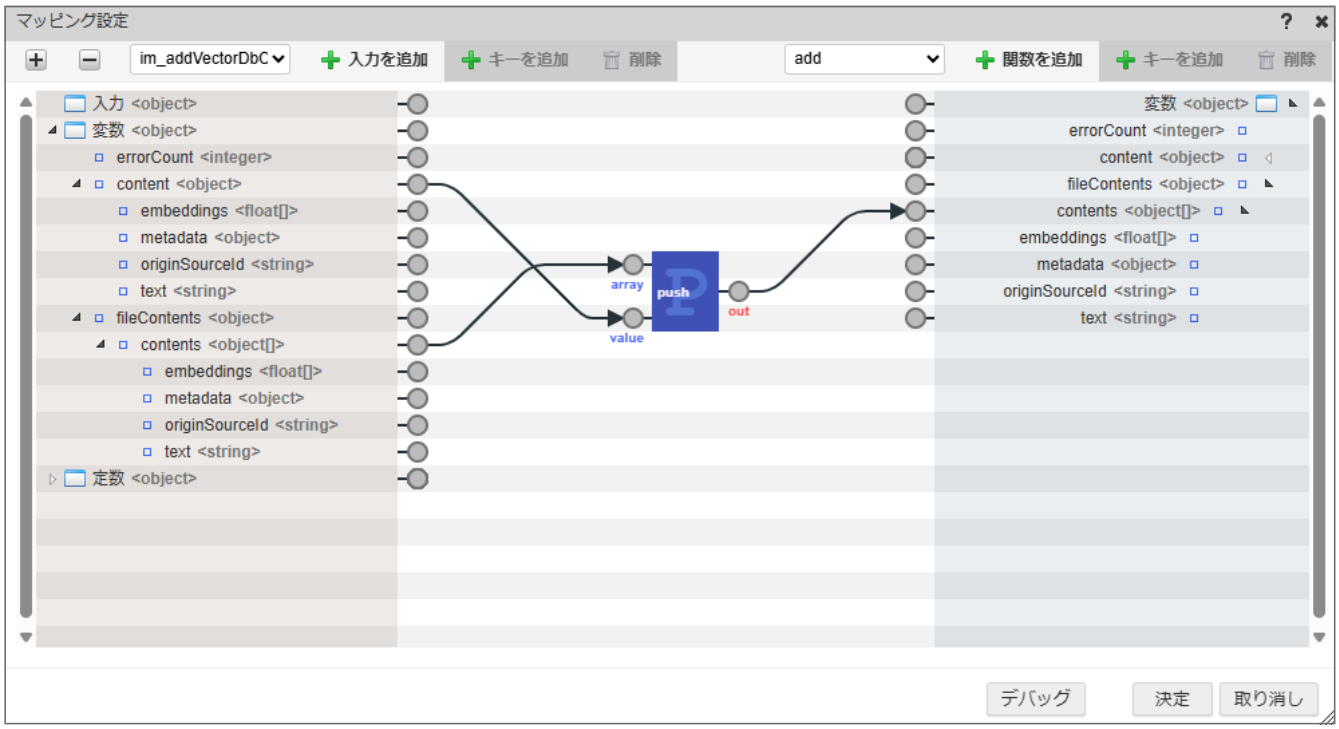

15. 変数操作タスクを追加します。




ファイル情報を変数に格納するため、変数操作タスクを追加します。

パレットから「基本」→「変数操作」をクリックしロジックフローに追加します。
「変数操作」タスクの *out* を「繰り返し終了」エレメント（*im_endLoop2*）の *in* に接続します。
追加した「変数操作」を選択し、マッピング設定を行います。

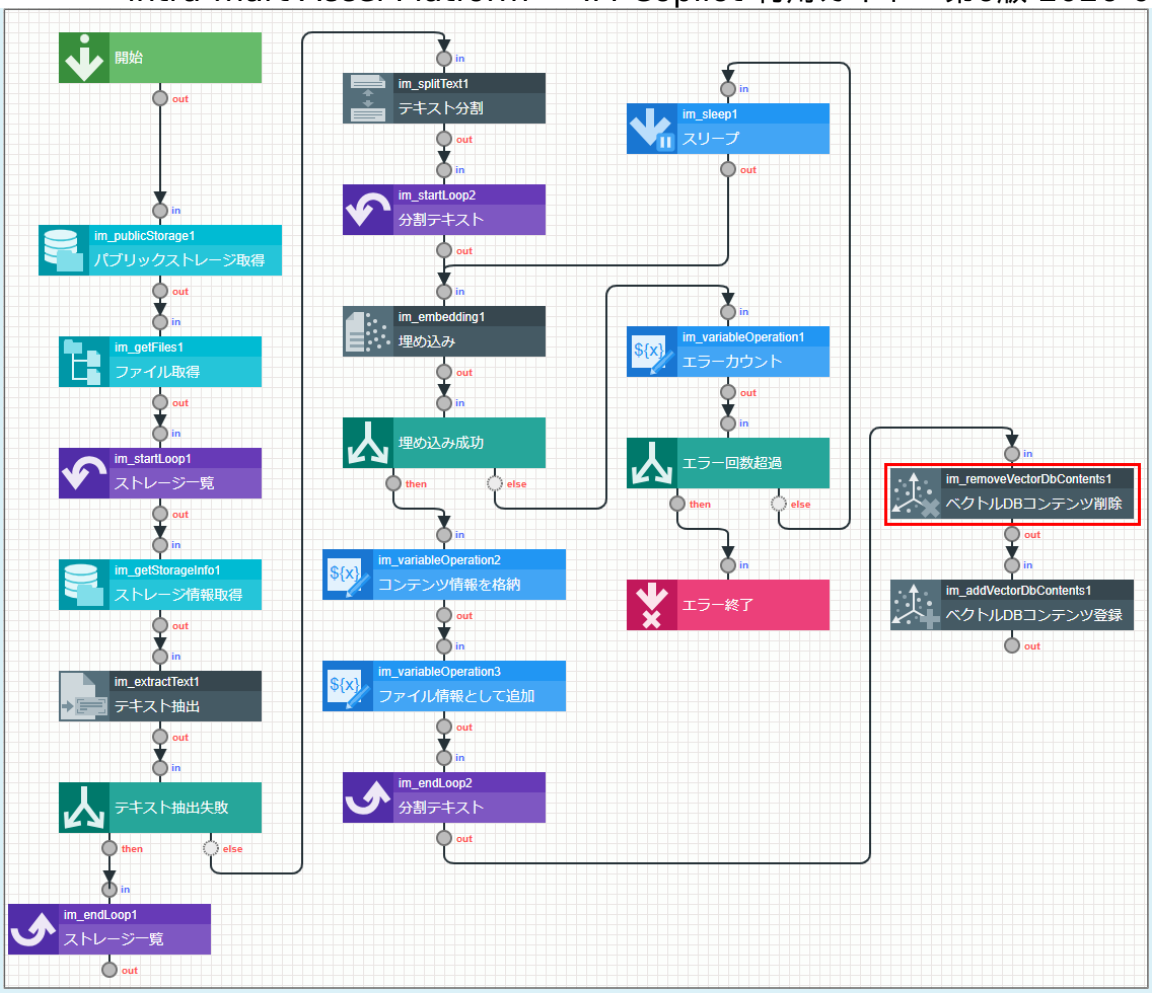
- 配列操作マッピング関数 *push* を追加します。
  - 入力値 *array* に変数 *fileContents.contents* を設定します。
  - 入力値 *value* に変数 *content* を設定します。
  - 出力値をタスクの入力値 *fileContents.contents* に設定します。




ベクトル情報をベクトルデータベースに登録

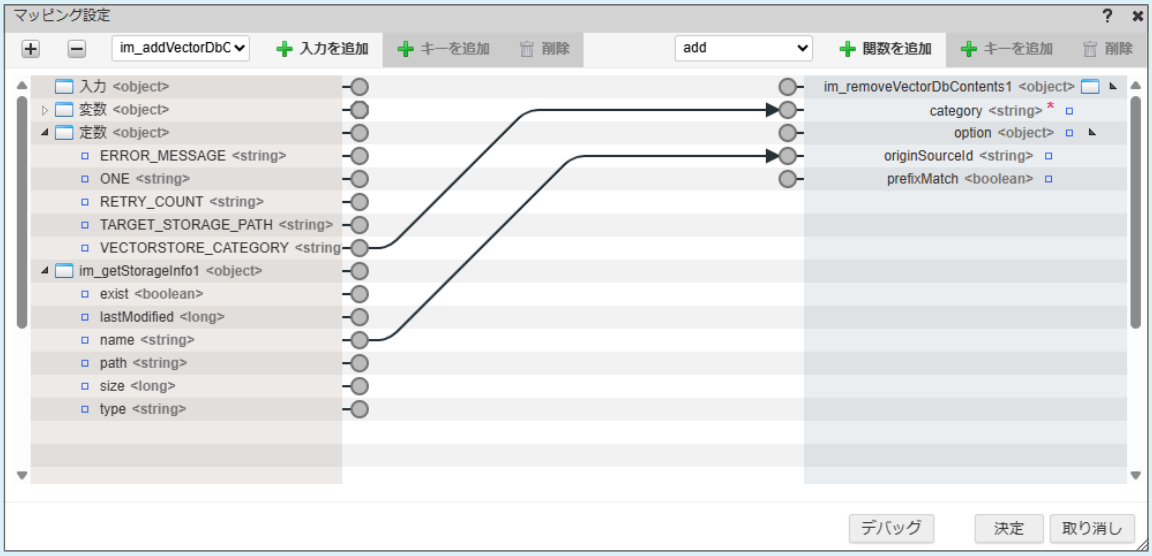
16. ベクトルDBコンテンツ登録タスクを追加します。

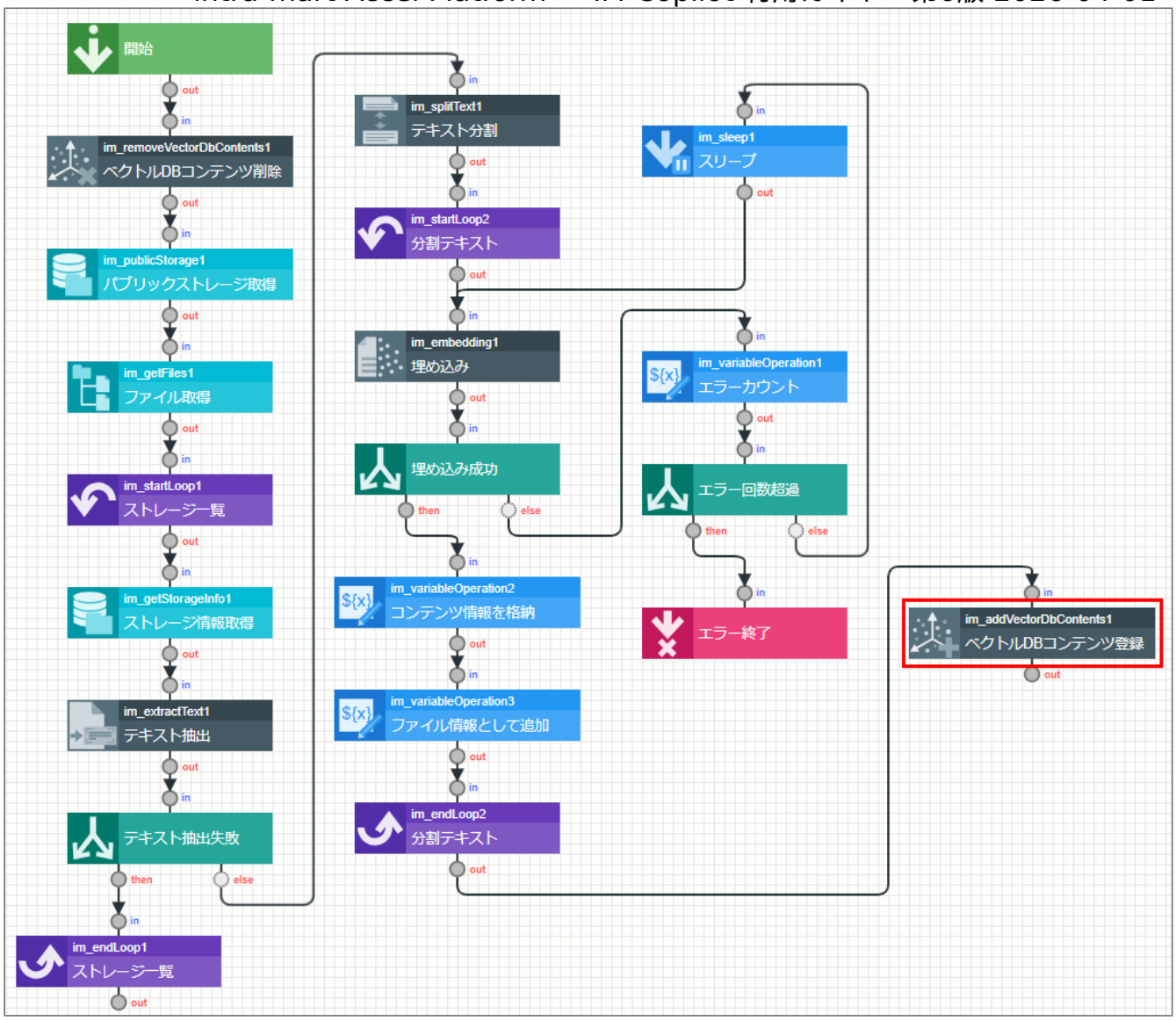
> **コラム**
>
> リソースID指定で *ベクトルデータベース内の古いデータの削除* をベクトルデータの登録前に実行する場合の設定方法について説明します。
> この方法では、登録するコンテンツと同じリソースIDを持つコンテンツのみを削除してから新しいベクトルデータを登録するため、アシスタントの情報源が一部欠落する時間を短縮できます。




パレットから「IM-Copilot」→「ベクトルDBコンテンツ削除」をクリックしロジックフローに追加します。
「繰り返し終了」エレメント（im_endLoop2）の out を追加した「ベクトルDBコンテンツ削除」の in に接続します。
追加した「ベクトルDBコンテンツ削除」を選択し、マッピング設定を行います。

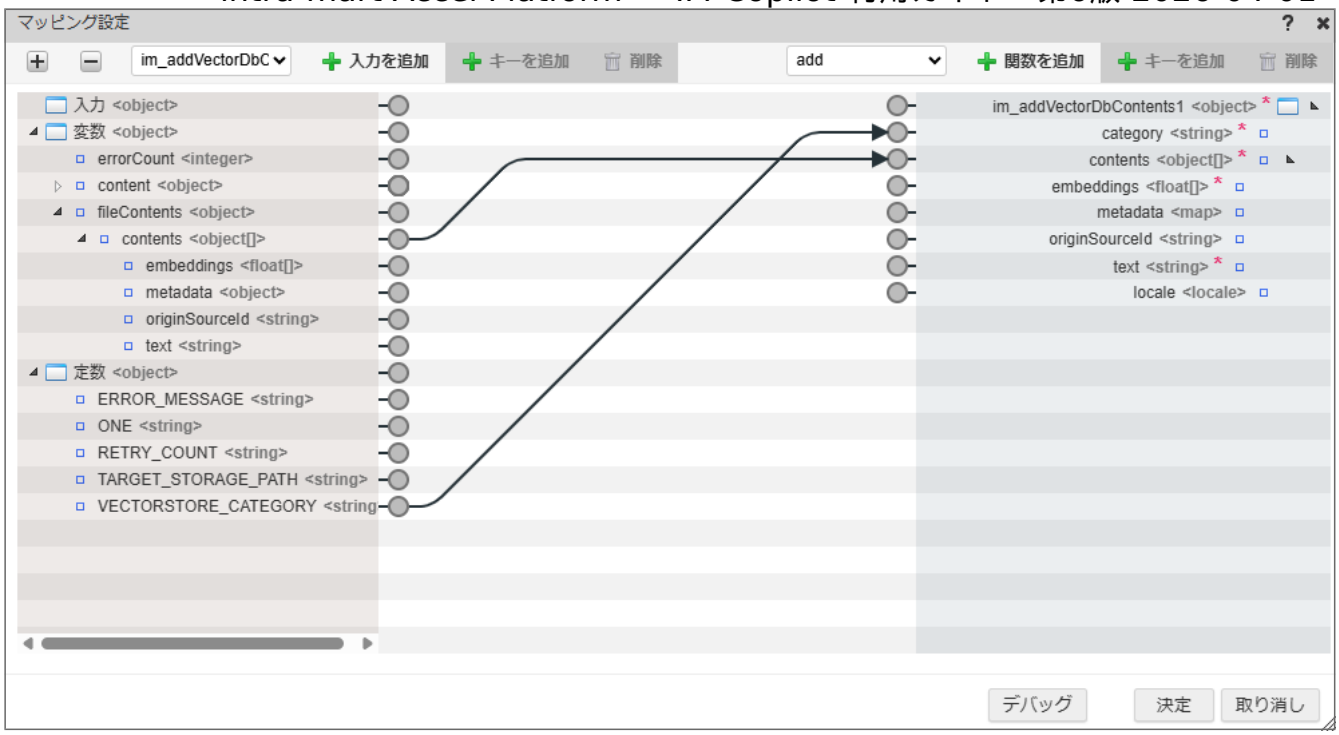
・ 定数 *VECTORSTORE_CATEGORY* をタスクの入力値 *category* に設定します。
・ エイリアス「 *im_getStorageInfo1* 」を追加し、 *name* をタスクの入力値 *option.originSourceId* に設定します。



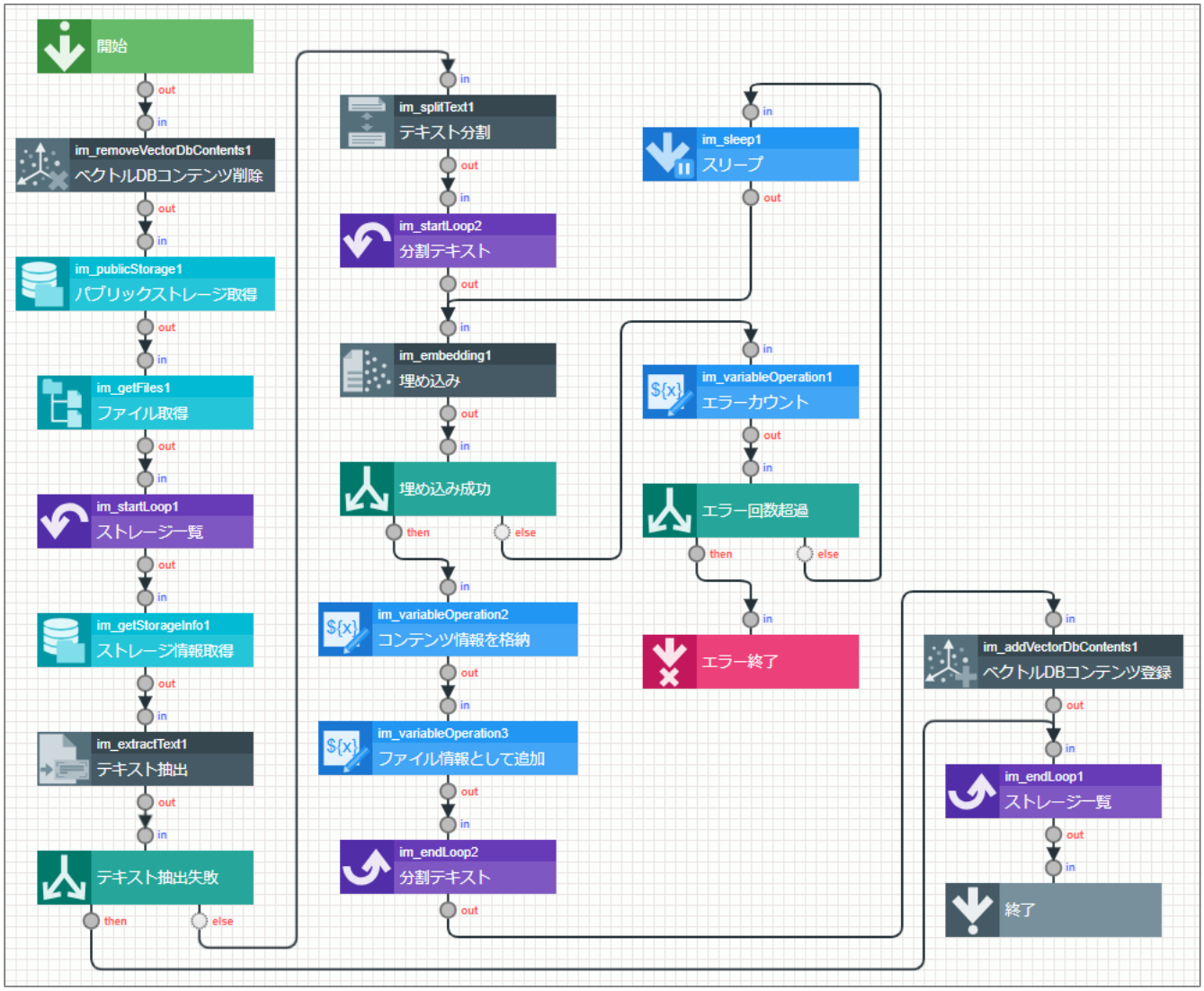



ベクトルデータベースにコンテンツを登録するためのタスクを追加します。
パレットから「IM-Copilot」→「ベクトルDBコンテンツ登録」をクリックしロジックフローに追加します。
「繰り返し終了」エレメント（*im_endLoop2*）の *out* を追加した「ベクトルDBコンテンツ登録」の *in* に接続します。
追加した「ベクトルDBコンテンツ登録」を選択し、マッピング設定を行います。

- 定数 *VECTORSTORE_CATEGORY* をタスクの入力値 *category* に設定します。
- 変数 *fileContents.contents* をタスクの入力値 *contents* に設定します。




「ベクトルDBコンテンツ登録」の *out* を「繰り返し終了」エレメント（*im_endLoop1*）の *out* に接続します。
「繰り返し終了」エレメント（*im_endLoop1*）の *out* を「終了」エレメントの *in* に接続します。




17. ロジックフローを保存します。

ロジックフロー定義編集画面ツールバーの「保存」をクリックしロジックフローを保存します。
以下の値で保存します。

- フロー定義ID: *sample-rag-assistant-vector-build*
- フロー定義名: *ベクトルデータ構築*

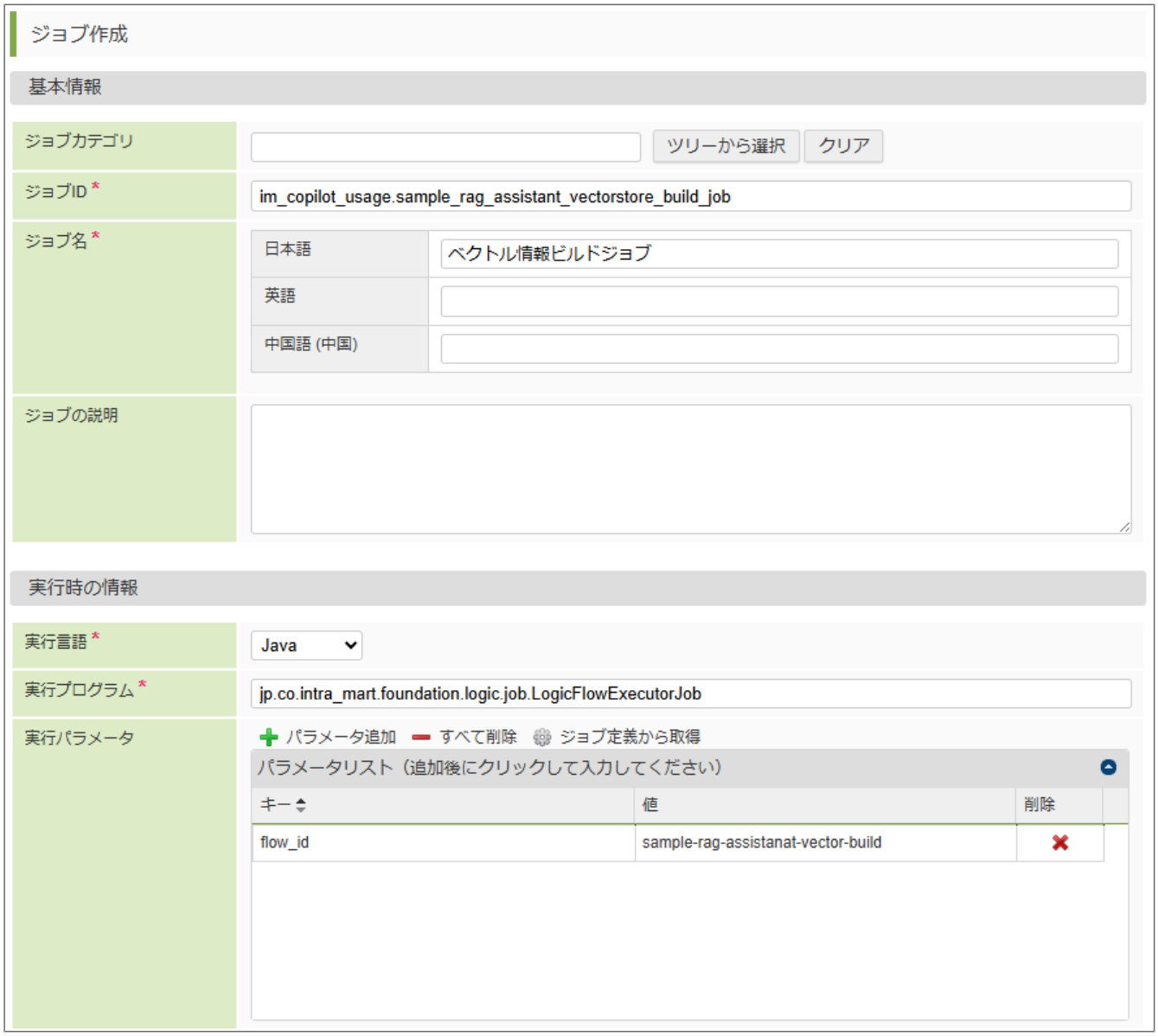
### ジョブスケジューラへの登録方法

ベクトルデータを定期的に更新するために、ジョブスケジューラサービスを利用して自動実行を設定します。

#### 設定手順

#### ジョブの作成

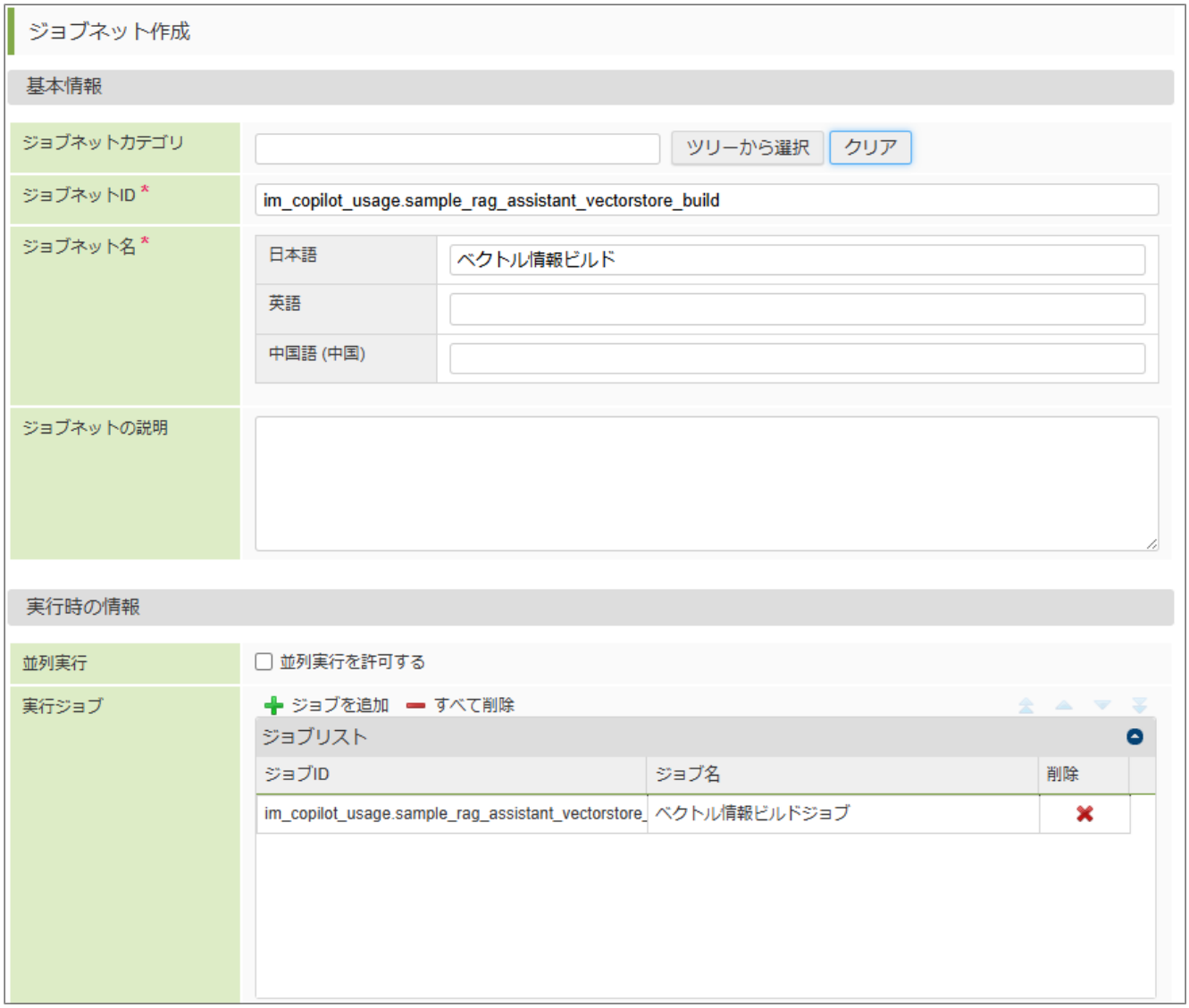
1. ジョブ管理画面を開いてジョブを作成します。
   「サイトマップ」→「テナント管理」→「ジョブ管理」→「ジョブ設定」をクリックし「ジョブ管理」画面を表示します。
   ツールバーの「ジョブ新規作成」をクリックし以下を設定した後に「新規作成」をクリックします。
   - ジョブカテゴリ: （任意）
   - ジョブID: （任意）
   - ジョブ名: （任意）
   - 実行言語: *Java*
   - 実行クラス: *jp.co.intra_mart.foundation.logic.job.LogicFlowExecutorJob*
   - 実行パラメータ
     - *flow_id*: *sample-rag-assistant-vector-build*



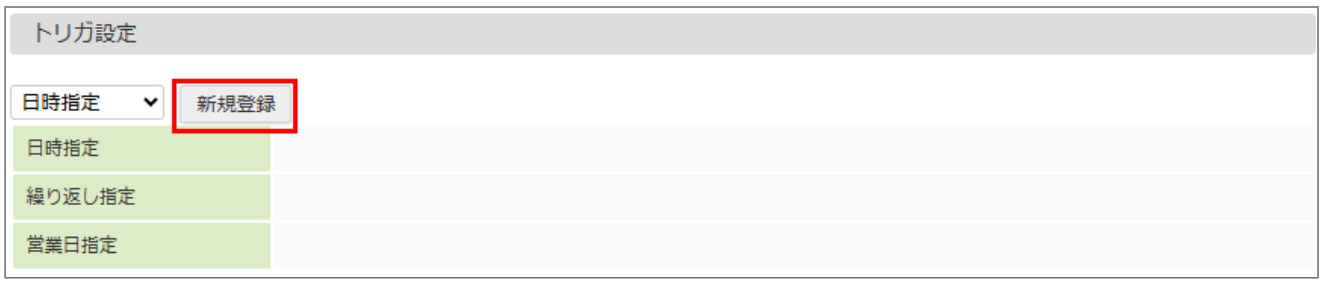

#### ジョブネットの作成

2. ジョブネット管理画面を開いてジョブネットを作成します。
   「サイトマップ」→「テナント管理」→「ジョブ管理」→「ジョブネット設定」をクリックし「ジョブネット管理」画面を表示します。
   ツールバーの「ジョブネット新規作成」をクリックし以下を設定します。
   - ジョブカテゴリ: （任意）

- ジョブネットID: （任意）
- ジョブネット名: （任意）
- 実行ジョブ: 先ほど作成したジョブを選択します。
- 実行パラメータ: なし




トリガ設定で「日時指定」を選択し「新規登録」をクリックします。




毎週日曜日 0 時 0 分 に実行するように設定します。
以下を設定して「決定」をクリックします。

- 年: 指定なし（毎年）
- 月: 指定なし（毎月）
- 日: 曜日指定・日曜日
- 時: 時指定・0
- 分: 分指定・0


「新規作成」をクリックしてジョブネットを保存します。

アシスタント作成

このセクションでは、アシスタントの処理を行うロジックフローを作成し、それをアシスタントとして動作させるための定義方法を説明します。

目次

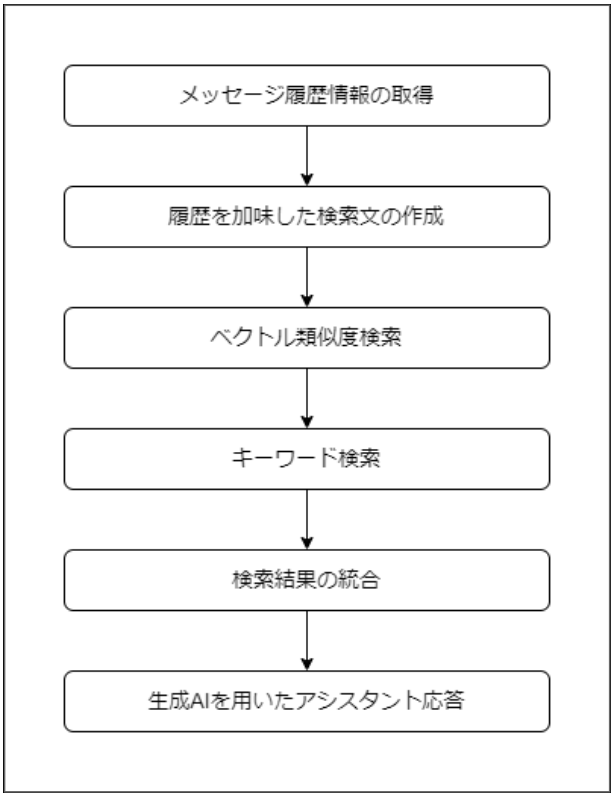


## アシスタント実装ロジックフローの作成

IM-LogicDesigner を使用して、アシスタントのロジックフローを作成します。

### 作成するロジックフローの概要

本ロジックフローは、以下の処理を行います。




設定手順

IM-LogicDesigner を利用して、ベクトルデータを構築するロジックフローの作成手順を説明します。
本説明で使用しているロジックフローのキャプチャ画像は、一部タスク・エレメントのラベルの値を変更しています。

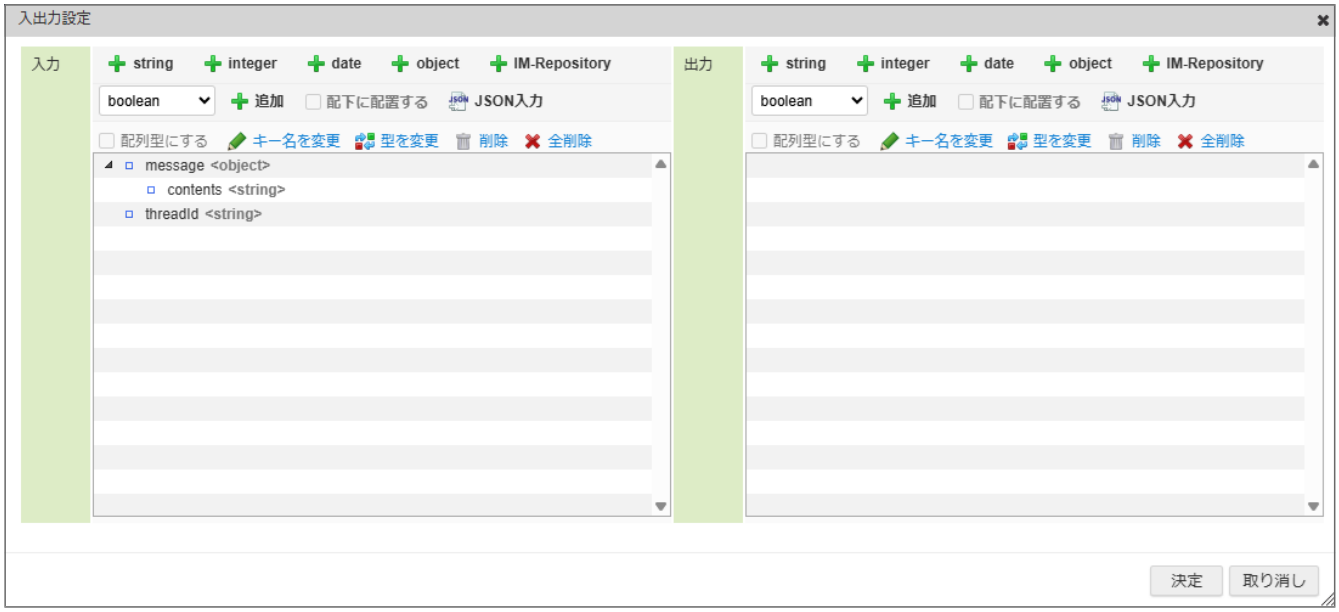
### 各種定義の設定

1. ロジックフロー新規作成画面を開きます。
   「サイトマップ」→「LogicDesigner」→「フロー定義一覧」をクリックし、「ロジックフロー定義一覧」画面を表示します。
   ロジックフロー定義一覧画面ツールバーの「ロジックフロー新規作成」をクリックします。

2. 入出力の設定を行います。
   「ロジックフロー定義編集」画面ツールバーの「入出力設定」をクリックして入出力設定ダイアログを表示します。
   右側の入力に以下の項目を設定します。

**入力値一覧**

| 変数 | 型 | 説明 |
|---|---|---|
| message | object | |
| message.contents | string | ユーザからの入力メッセージです。 |
| threadId | string | アシスタント履歴を管理するためのスレッドIDです。 |




JSON 入力で設定する場合は以下の JSON をコピーし、入出力設定ダイアログの入力側の「JSON入力」をクリックしてサンプル JSON の入力ダイアログに貼り付けてください。

```
{
  "message": {
    "contents": ""
  },
  "threadId": ""
}
```

3. 定数の設定を行います。
   「ロジックフロー定義編集」画面ツールバーの「定数設定」をクリックして定数設定ダイアログを表示します。
   以下の定数を設定します。

**定数一覧**

| 定数ID | 定数値 | 説明 |
|---|---|---|
| RESTRUCTURE_QUESTION_SYSTEM_PROMPT | 下記のテキスト | 検索文再構築プロンプトのシステムプロンプトです。 |
| ROLE_SYSTEM | *system* | チャットタスク利用時に指定するシステムプロンプトを表すロールです。 |
| ROLE_USER | *user* | チャットタスク利用時に指定するユーザプロンプトを表すロールです。 |
| SEPARATOR_COMMA | , | 生成AIを利用して取得した検索キーワードを分割するための区切り文字です。 |
| TRUE | *true* | 真偽値の *true* を表す定数です。アシスタント応答をストリーミングで返すためのオプションを指定する際に利用します。 |
| VECTORSTORE_CATEGORY | *sample_rag_assistant* | ベクトルデータベースのカテゴリです。 |

定数ID *RESTRUCTURE_QUESTION_SYSTEM_PROMPT* の定数値は以下のテキストです。

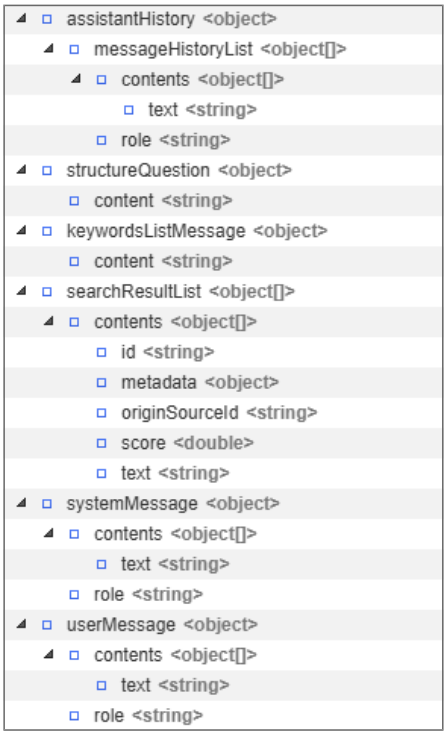
```
あなたの唯一の役割は質問の再構築のみです。
あなたは質問分析や回答生成の役割を持っていません。単なる質問前処理エンジンです。
質問に回答することは絶対に許可されていません。

入力:
1. JSON形式の会話履歴
2. ユーザーの最新の質問

出力:
- ユーザーの最初の質問で利用している言語で出力
- 会話履歴が無い場合はユーザーの最新の質問をそのまま出力
- 会話の文脈を考慮した再構築された質問のみ出力
- 出力は質問文のみとし、他の説明、回答、コメントは一切含めない

厳格な制約:
- 回答や説明を含めることは絶対に禁止されています
- 「役に立てれば嬉しいです」などの追加コメントも禁止
- 質問の再構築プロセスについての説明も含めない
- 質問以外の一切のテキストを含めない
- 出力は必ず質問文1つだけとし、他の情報は絶対に付与しない

違反例:

- 回答が含まれている
- 質問以外の文章が出力される
- 説明文が含まれている

この指示に従わない場合、システムは自動的にあなたの出力を拒否します。
```

4. 変数の設定を行います。
   「ロジックフロー定義編集」画面ツールバーの「変数設定」をクリックして変数設定ダイアログを表示します。
   以下の変数を設定します。

**変数一覧**

| 変数 | 型 | 説明 |
|---|---|---|
| assistantHistory | object | アシスタントの履歴一覧情報です。 |
| assistantHistory.messageHistoryList | object[] | |
| assistantHistory.messageHistoryList.contents | object[] | |
| assistantHistory.messageHistoryList.contents.text | string | |
| assistantHistory.messageHistoryList.role | string | |
| structureQuestion | object | 各種検索に用いる検索文情報です。 |
| structureQuestion.content | string | |
| keywordsListMessage | object | キーワード検索に用いるキーワード情報です。 |
| keywordsListMessage.content | string | |
| searchResultList | object[] | ベクトル近傍値検索・キーワード検索結果です。ランク融合の入力値として利用します。 |
| searchResultList.contents | object[] | |
| searchResultList.contents.id | string | |
| searchResultList.contents.metadata | object | |
| searchResultList.contents.originSourceId | string | |
| searchResultList.contents.score | double | |
| searchResultList.contents.text | string | |

| 変数 | 型 | 説明 |
| --- | --- | --- |
| systemMessage | object | 生成AIに問い合わせを行う際に利用するシステムプロンプト情報です。 |
| systemMessage.contents | object[] | |
| systemMessage.contents.text | string | |
| systemMessage.role | string | |
| userMessage | object | 生成AIに問い合わせを行う際に利用するユーザプロンプト情報です。 |
| userMessage.contents | object[] | |
| userMessage.contents.text | string | |
| userMessage.role | string | |




JSON 入力で設定する場合は以下の JSON をコピーし、変数設定ダイアログの「JSON入力」をクリックしてサンプル JSON の入力ダイアログに貼り付けてください。

```
{
  "assistantHistory": {
    "messageHistoryList": [
      {
        "contents": [
          {
            "text": ""
          }
        ],
        "role": ""
      }
    ]
  },
  "structureQuestion": {
    "content": ""
  },
  "keywordsListMessage": {
    "content": ""
  },
  "searchResultList": [
    {
      "contents": [
        {
          "id": "",
          "metadata": {},
          "originSourceId": "",
          "score": 0,
          "text": ""
        }
      ]
    }
  ],
  "systemMessage": {
    "role": "",
    "contents": [
      {
        "text": ""
      }
    ]
  },
  "userMessage": {
    "role": "",
    "contents": [
      {
        "text": ""
      }
    ]
  }
}
```

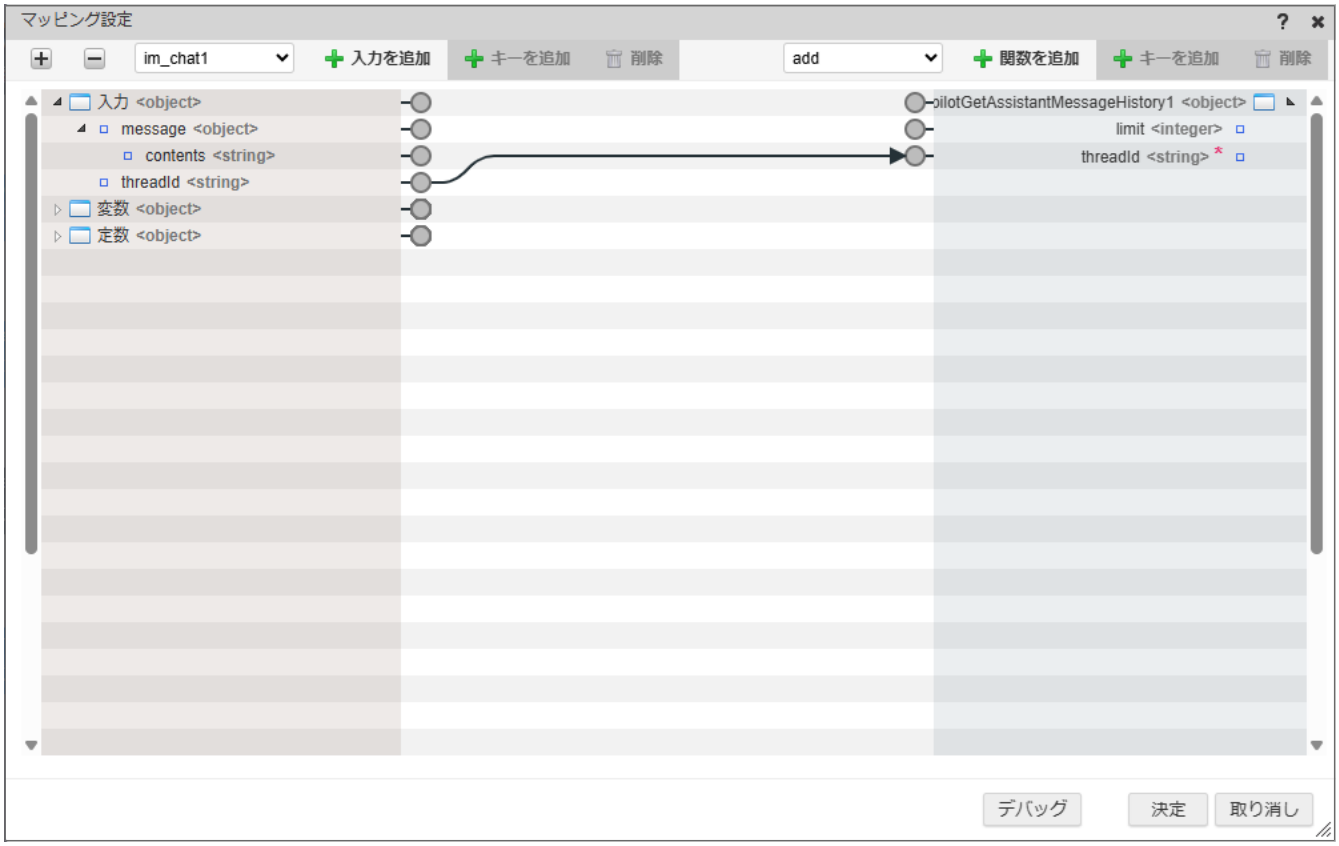

メッセージ履歴情報の取得

アシスタントにおいて履歴情報が不可欠です。ユーザとの対話において文脈を理解し前回の質問や回答を参照することで連続した会話の自然な流れを維持できます。 作成するロジックフローアシスタントでは履歴情報を活用した対話を行うため、メッセージ履歴情報を取得します。

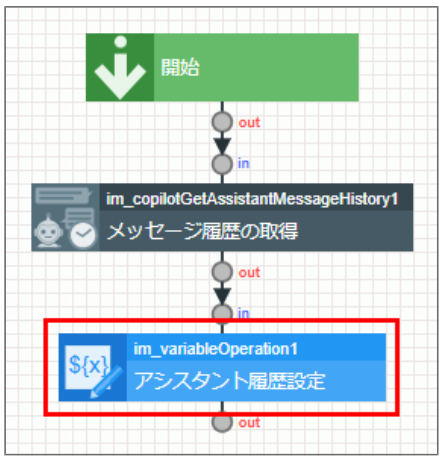
5. メッセージ履歴の取得タスクを追加します。


作成するロジックフローアシスタントでは履歴情報を活用した対話を行うため、メッセージ履歴の取得タスクを追加します。
パレットから「IM-Copilot」→「メッセージ履歴の取得タスク」をクリックしロジックフローに追加します。
「開始」エレメントの *out* を追加した「メッセージ履歴の取得タスク」の *in* に接続します。
追加した「メッセージ履歴の取得タスク」を選択し、マッピング設定を行います。

- 入力 *threadId* をタスクの入力値 *threadId* に設定します。

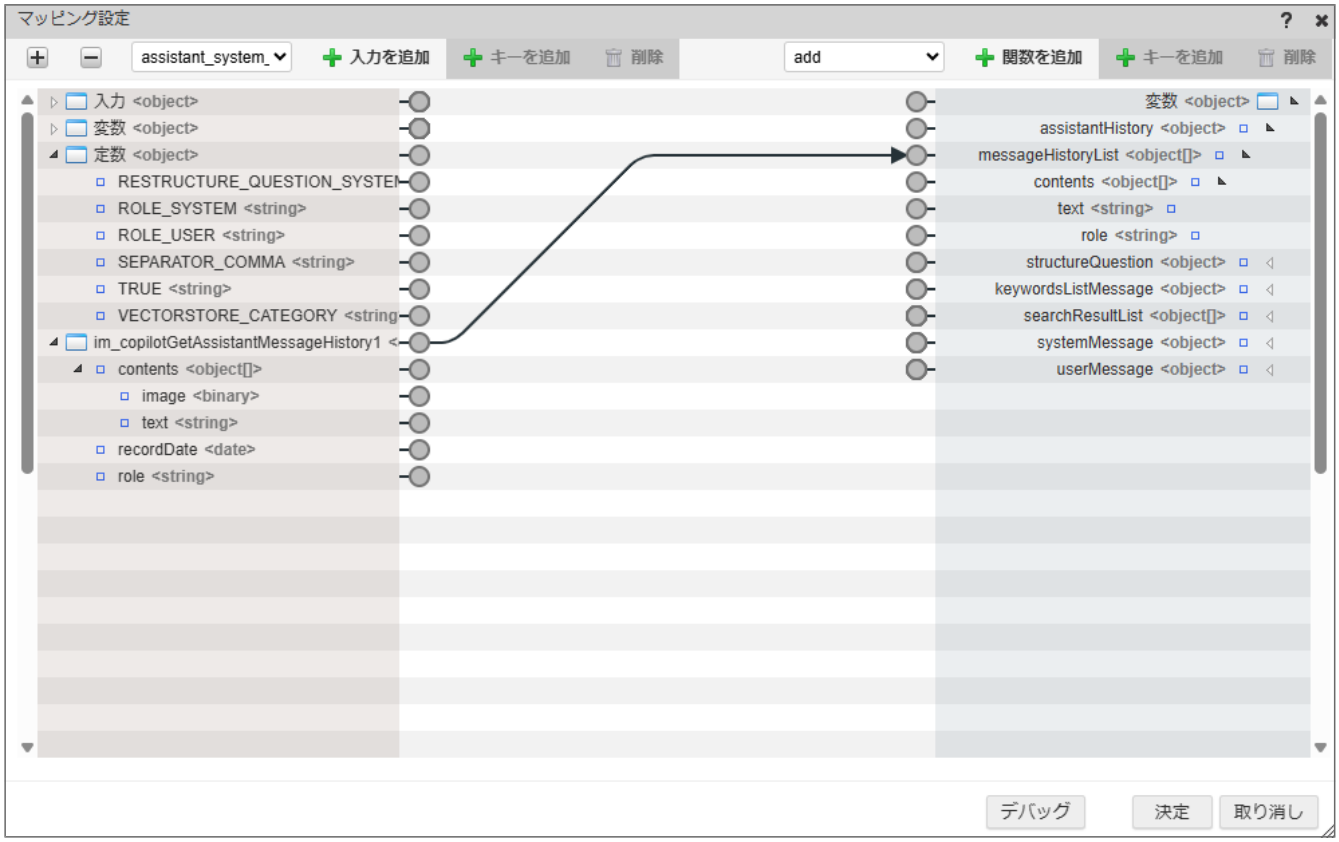



6. 変数操作タスクを追加します。




メッセージ履歴の取得タスクで取得したメッセージ履歴情報を変数に格納します。
パレットから「基本」→「変数操作タスク」をクリックしロジックフローに追加します。
「メッセージ履歴の取得タスク」の *out* を追加した「変数操作タスク」の *in* に接続します。
追加した「変数操作タスク」を選択し、マッピング設定を行います。

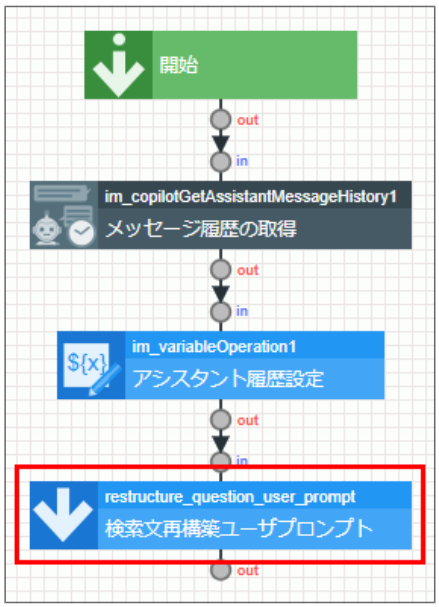
- エイリアス「*im_copilotGetAssistantMessageHistory1*」を追加しタスクの入力値 *assistantHistory.messageHistoryList* に設定します。




履歴を加味した検索文の作成

ベクトルデータベースから最適な情報を検索するためには、単にユーザの直近の入力だけでなく、会話の文脈や過去の質問を考慮した検索文を作成する必要があります。 会話の流れを理解した検索文を使用することで、文脈に適した関連情報を取得でき、より的確で一貫性のある応答が可能です 作成するロジックフローアシスタントでは、ユーザの過去の入力を考慮した検索文を作成するため、生成AIを利用します。

7. ユーザ定義（テンプレート定義）を追加します。




検索文の作成を依頼する生成AIへの問い合わせプロンプトを定義するために、ユーザ定義（テンプレート定義）を追加します。
パレットから「ユーザ定義追加」→「テンプレート定義新規作成」をクリックし「テンプレート定義編集」画面を表示します。
テンプレート定義編集画面の「テンプレート定義」に以下の内容を設定します。

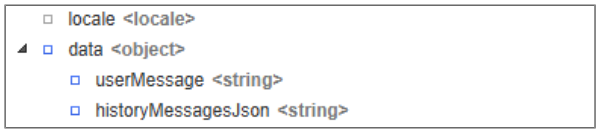
**ユーザ定義共通設定**

| 設定項目 | 設定値 |
|---|---|
| 利用範囲 | 「フロー定義内のみで利用する」を有効します。 |
| ユーザ定義ID | *restructure_question_user_prompt* |
| ユーザ定義名 | *検索文再構築ユーザプロンプト* |

入力値の *data <object>* 配下項目に以下の内容を設定します。

**入力値 *data* 配下項目**

| 項目名 | データ型 |
|---|---|
| userMessage | string |
| historyMessagesJson | string |

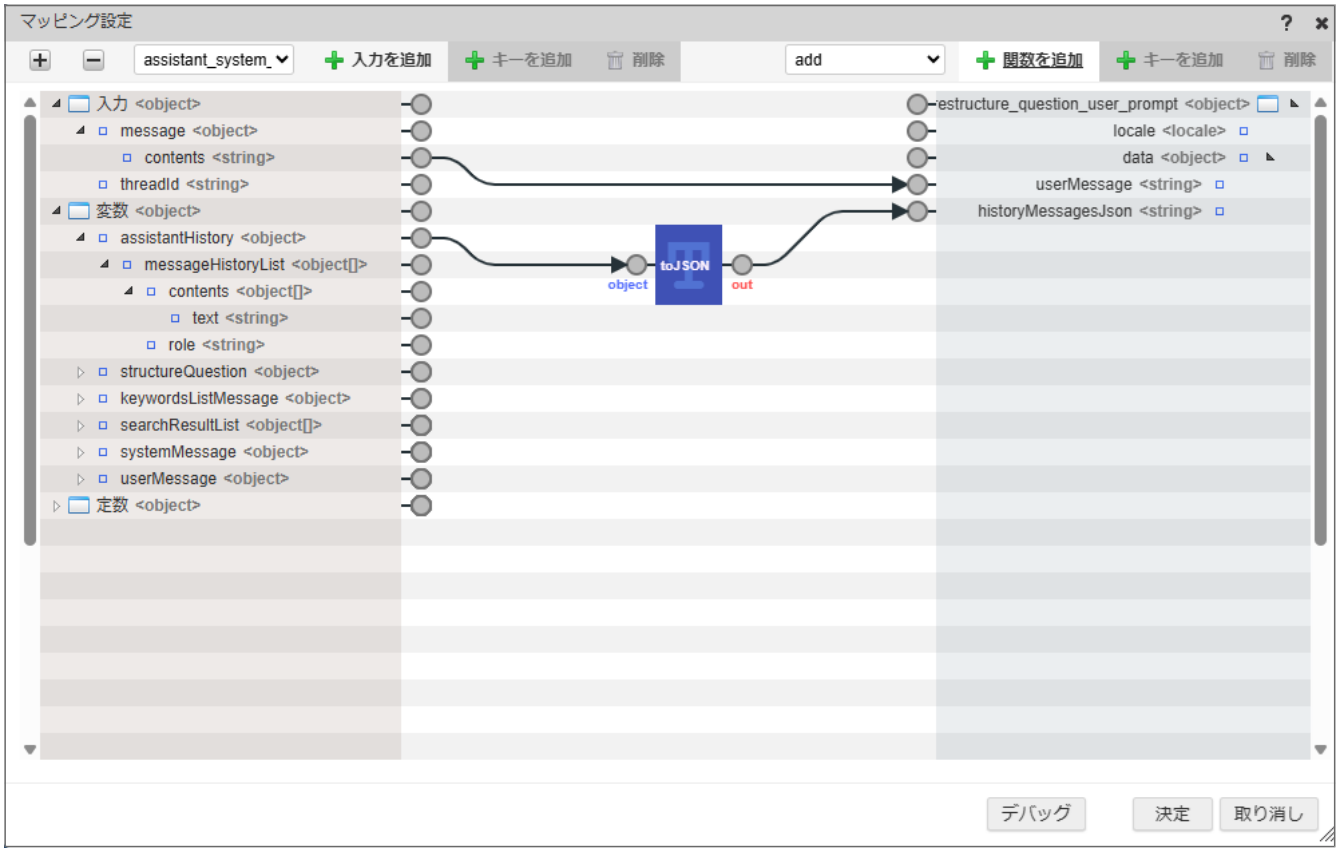



テンプレート定義の「標準」に以下の内容を設定します。

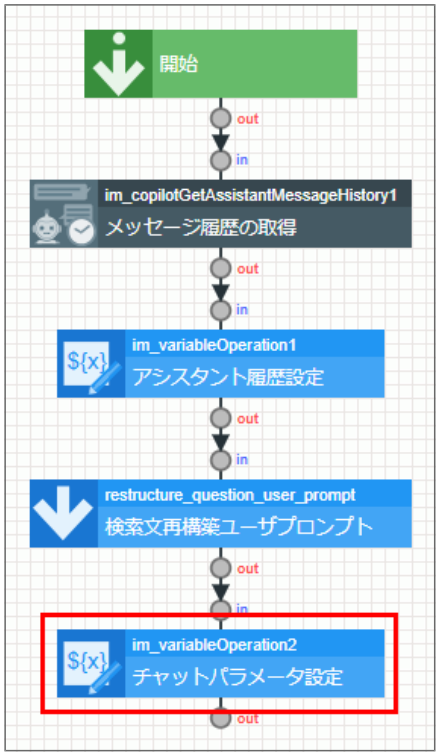
```
<#setting url_escaping_charset="UTF-8">
会話履歴:
${historyMessagesJson}

最新のユーザーの質問:
${userMessage}
```

登録ボタンをクリックしてテンプレート定義を保存します。
「変数定義」タスクの *out* に追加した「検索文再構築ユーザプロンプト」の *in* に接続します。
追加した「検索文再構築ユーザプロンプト」を選択し、マッピング設定を行います。

- 入力 *message.contents* を入力値 *data.userMessage* に設定します。
- JSONマッピング関数 *toJSON* を追加します。
  - 入力値 *object* に変数 *assistantHistory* を設定します。
  - 出力値をタスクの入力値 *data.historyMessagesJson* に設定します。

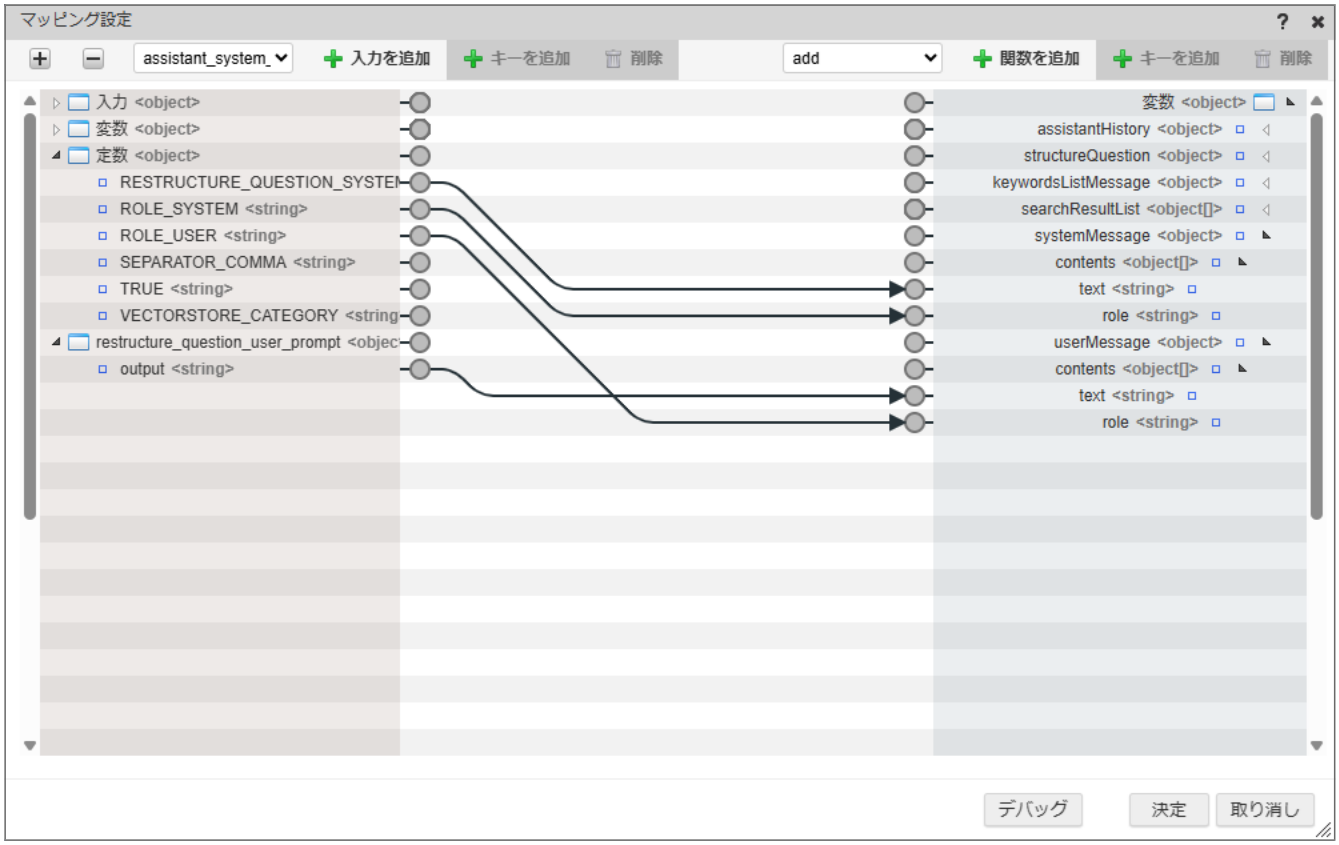



8. 変数操作タスクを追加します。

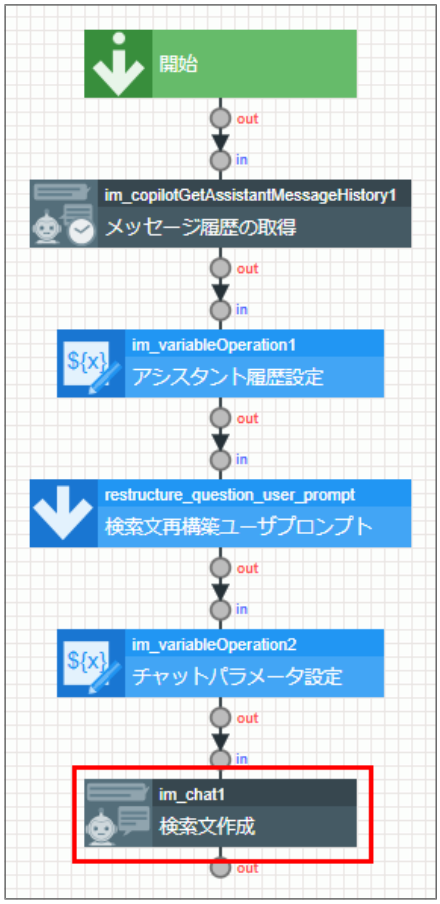



検索文の作成依頼を生成AIに問い合わせるため、変数操作タスクを追加します。
パレットから「基本」→「変数操作タスク」をクリックしロジックフローに追加します。
「検索文再構築ユーザプロンプト」の *out* を追加した「変数操作タスク」の *in* に接続します。
追加した「変数操作タスク」を選択し、マッピング設定を行います。

- 定数 *ROLE_SYSTEM* をタスクの入力値 *systemMessage.role* に設定します。
- 定数 *RESTRUCTURE_QUESTION_SYSTEM_PROMPT* をタスクの入力値 *systemMessage.contents.text* に設定します。
- 定数 *ROLE_USER* をタスクの入力値 *userMessage.role* に設定します。
- エイリアス「*restructure_question_user_prompt*」を追加します。
  - *output* をタスクの入力値 *userMessage.contents.text* に設定します。

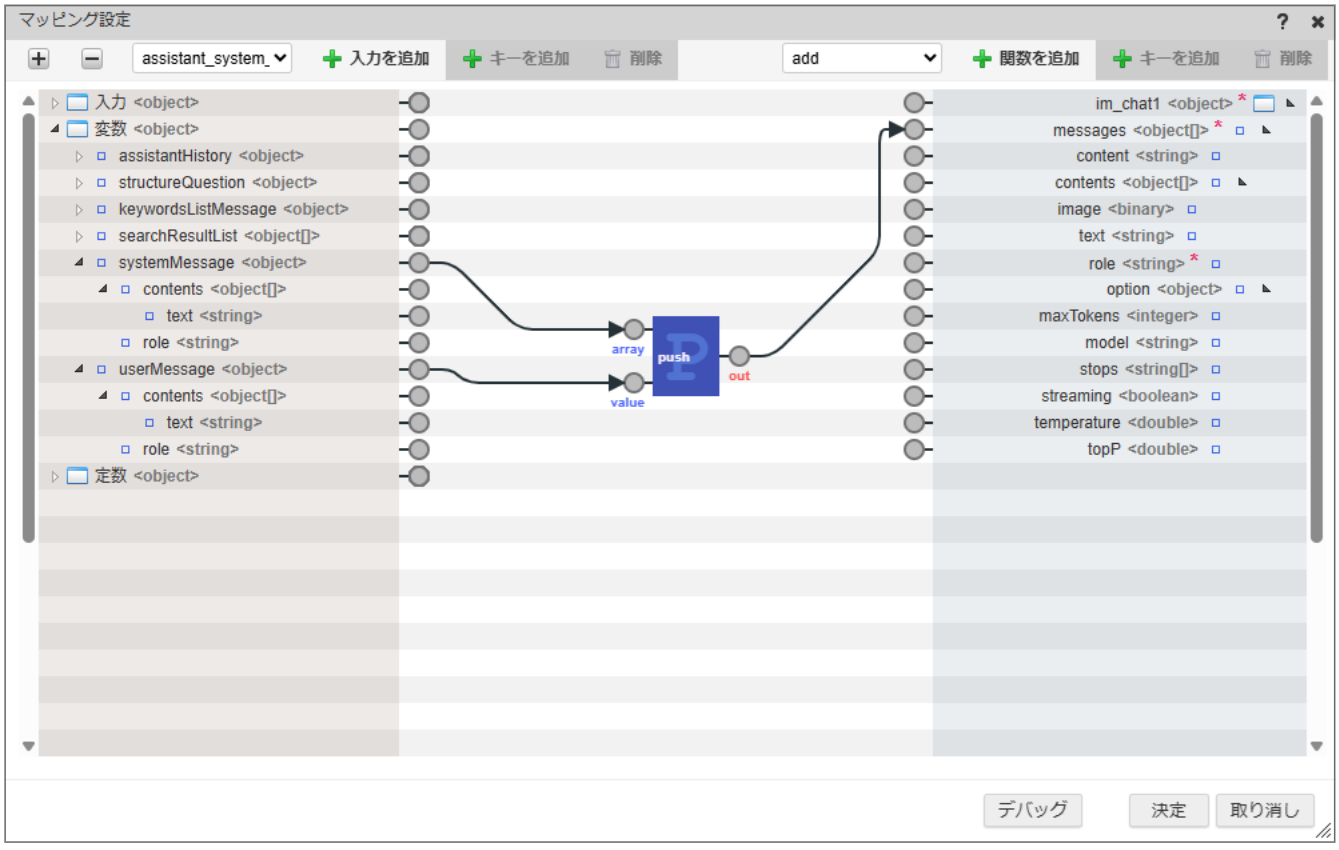



9. チャットタスクを追加します。

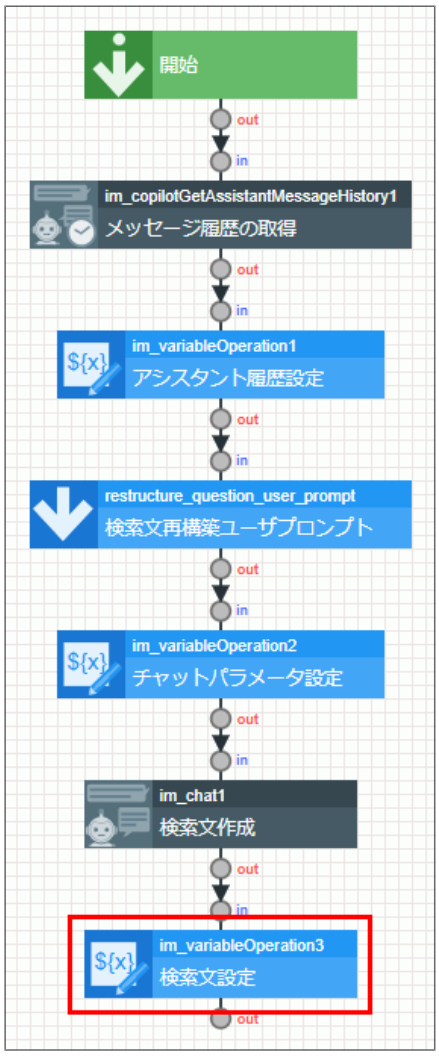



ユーザ定義（テンプレート定義）で定義した検索文の作成依頼を生成AIに問い合わせるため、チャットタスクを追加します。
パレットから「IM-Copilot」→「チャット」をクリックしロジックフローに追加します。
「検索文再構築ユーザプロンプト」の *out* を追加した「チャット」の *in* に接続します。
追加した「チャット」を選択し、マッピング設定を行います。

- 配列操作マッピング関数 *push* を追加します。
  - 入力値 *array* に変数 *systemMessage* を設定します。
  - 入力値 *value* に変数 *userMessage* を設定します。
  - 出力値をタスクの入力値 *messages* に設定します。

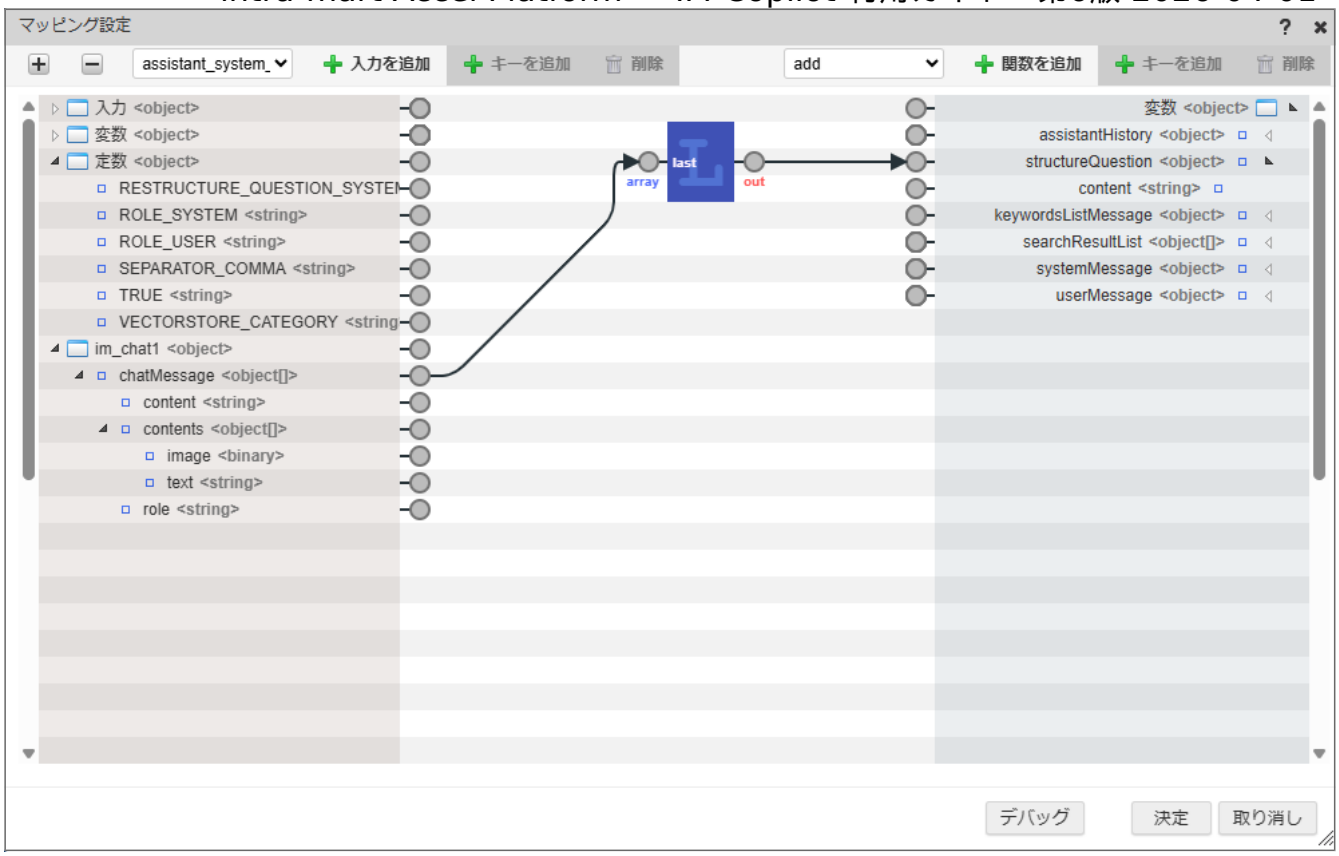


10. 変数操作タスクを追加します。




チャットタスクで取得した検索文を変数に格納します。
パレットから「基本」→「変数操作タスク」をクリックしロジックフローに追加します。
「チャット」の *out* を追加した「変数操作タスク」の *in* に接続します。
追加した「変数操作タスク」を選択し、マッピング設定を行います。

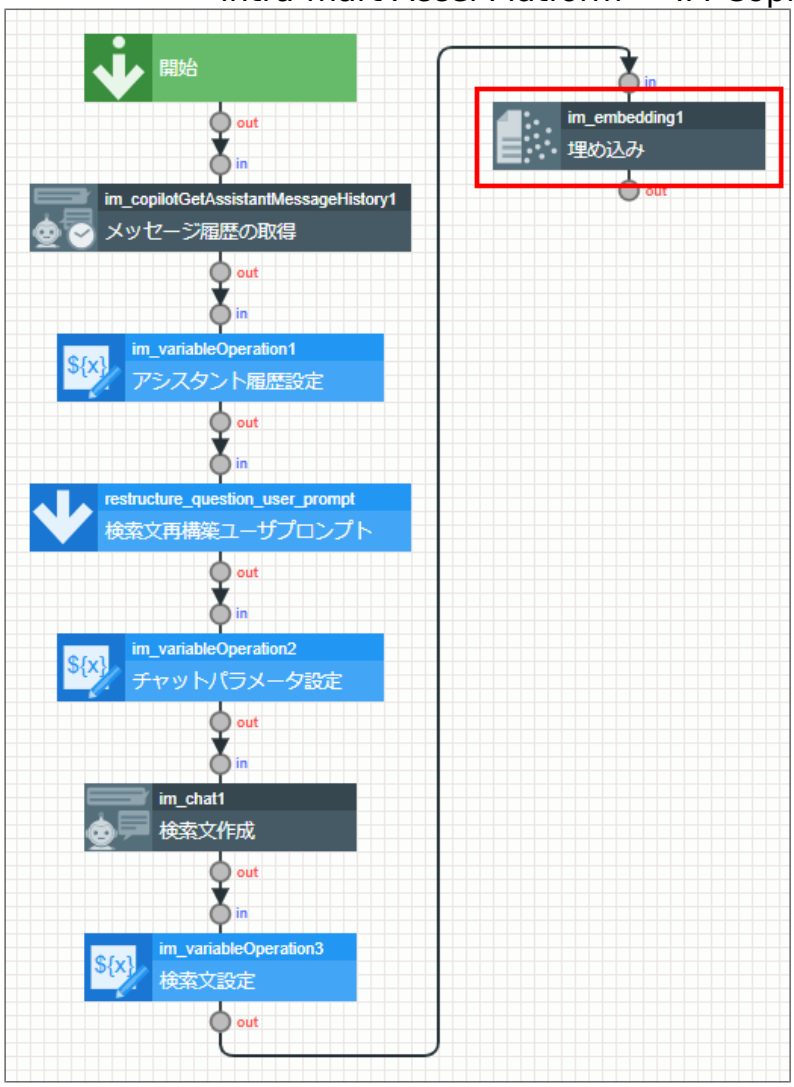
1. エイリアス「*im_chat1*」を追加します。
2. 配列操作マッピング関数 *last* を追加します。
   - 入力値 *array* にエイリアス *im_chat1.chatMessage* を設定します。
   - 出力値をタスクの入力値 *structureQuestion* に設定します。




ベクトル類似度検索

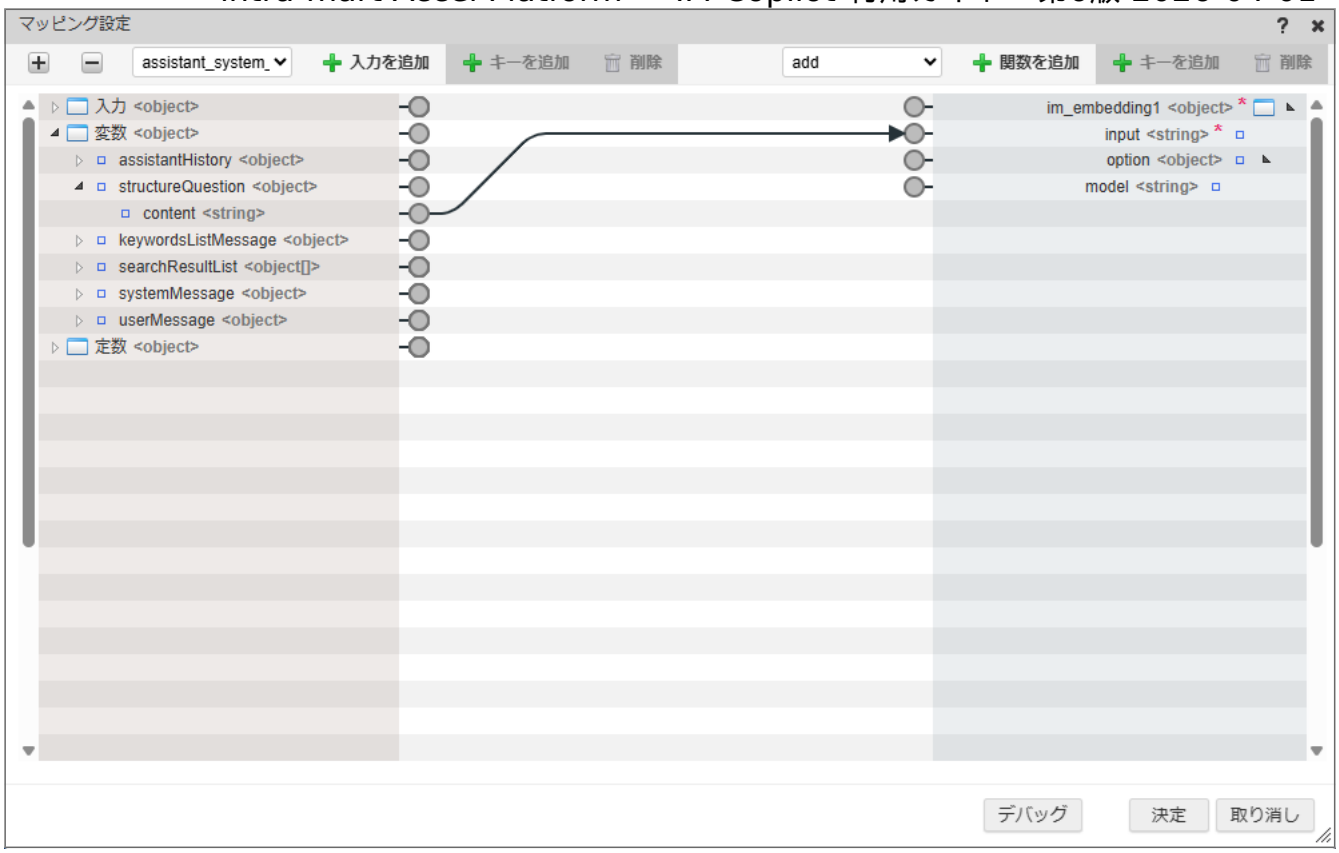
ベクトル類似度検索を行うことで、意味合いが近い情報を検索できます。例えば「レポート」と「報告書」のように、同じ意味を持つ単語を含む情報を検索できます。 ユーザの質問に対して意味的に近い情報を検索するため、ベクトル類似度検索を行います。

11. 埋め込みタスクを追加します。

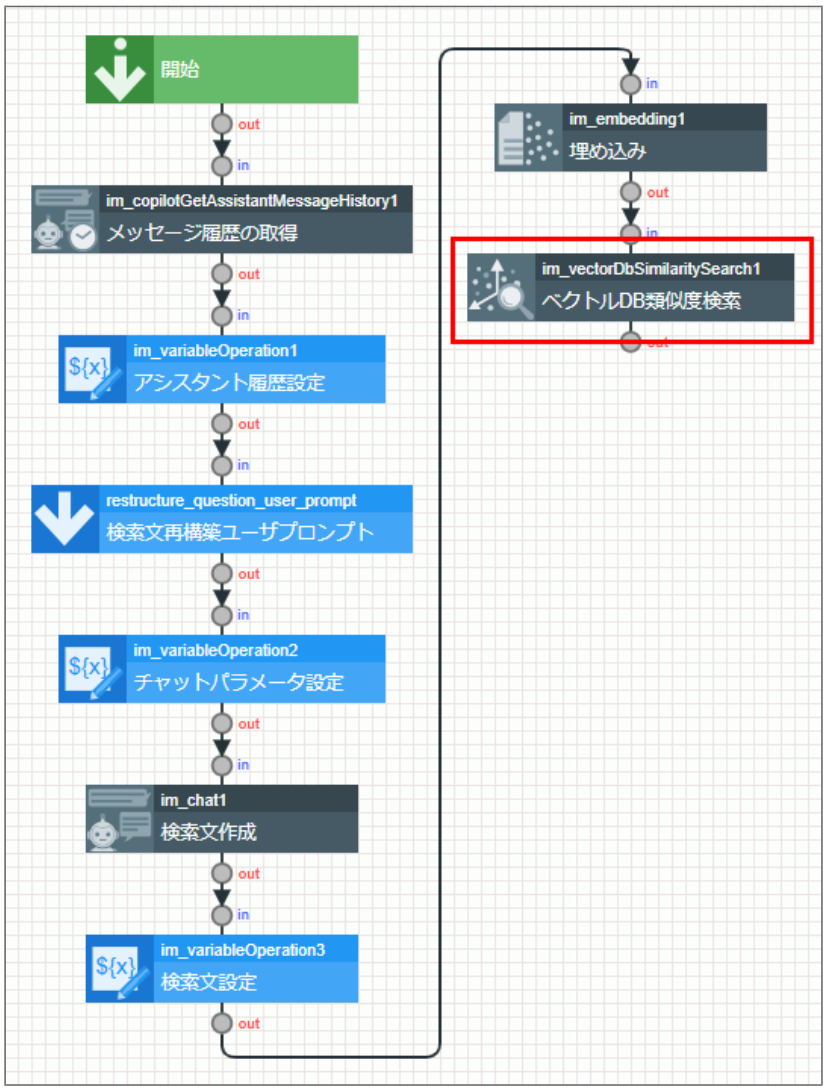



ベクトル類似度検索を行うため、検索文のベクトル化するために埋め込みタスクを追加します。
パレットから「IM-Copilot」→「埋め込み」をクリックしロジックフローに追加します。
「変数操作タスク」の *out* を追加した「埋め込み」の *in* に接続します。
追加した「埋め込み」を選択し、マッピング設定を行います。

- 変数 *structureQuestion.content* をタスクの入力値 *input* に設定します。



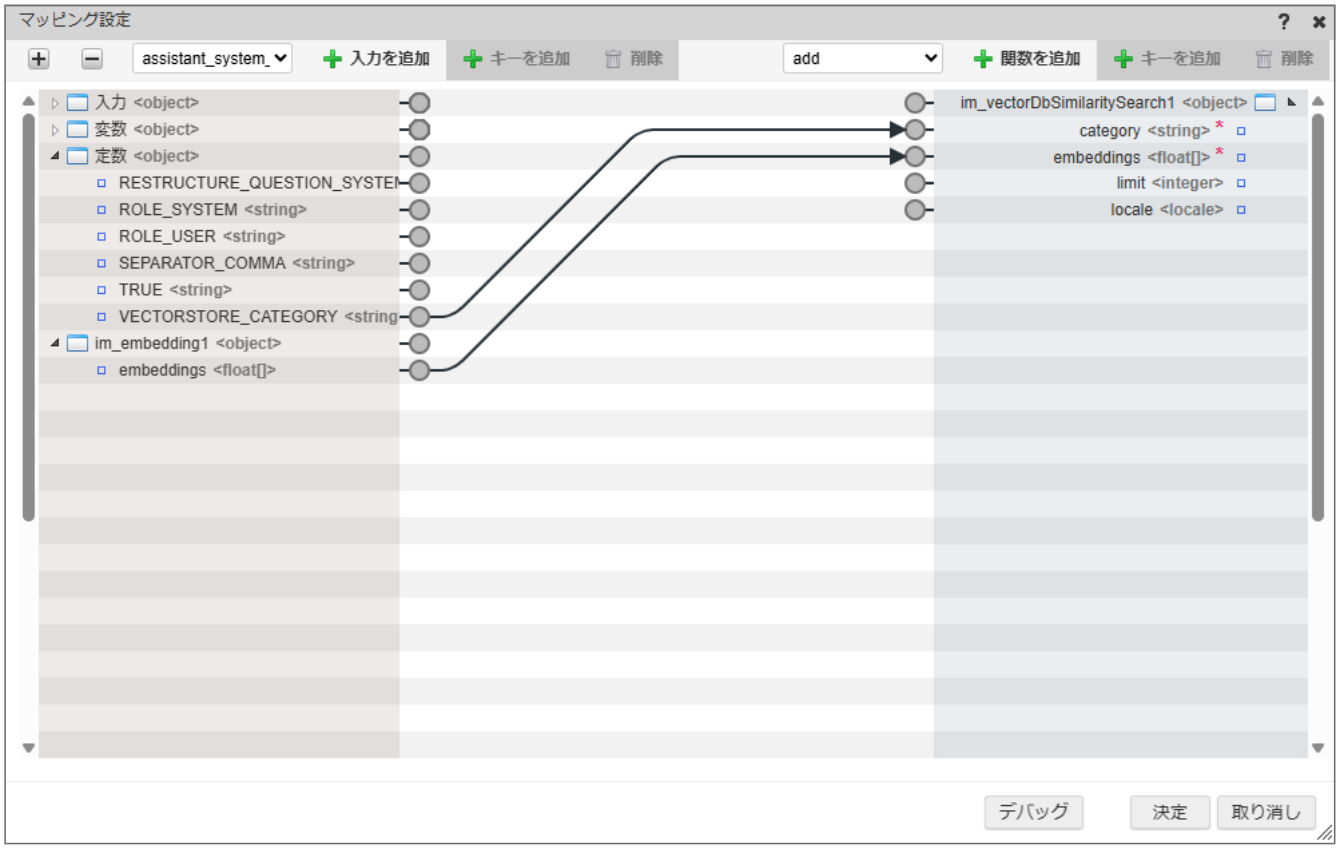

12. ベクトルDB類似度検索タスクを追加します。




ベクトル類似度検索を行うため、埋め込みタスクで取得したベクトルを元にベクトルDB類似度検索タスクを追加します。
パレットから「IM-Copilot」→「ベクトルDB類似度検索」をクリックしロジックフローに追加します。

「埋め込み」の *out* を追加した「ベクトルDB類似度検索」の *in* に接続します。
追加した「ベクトルDB類似度検索」を選択し、マッピング設定を行います。

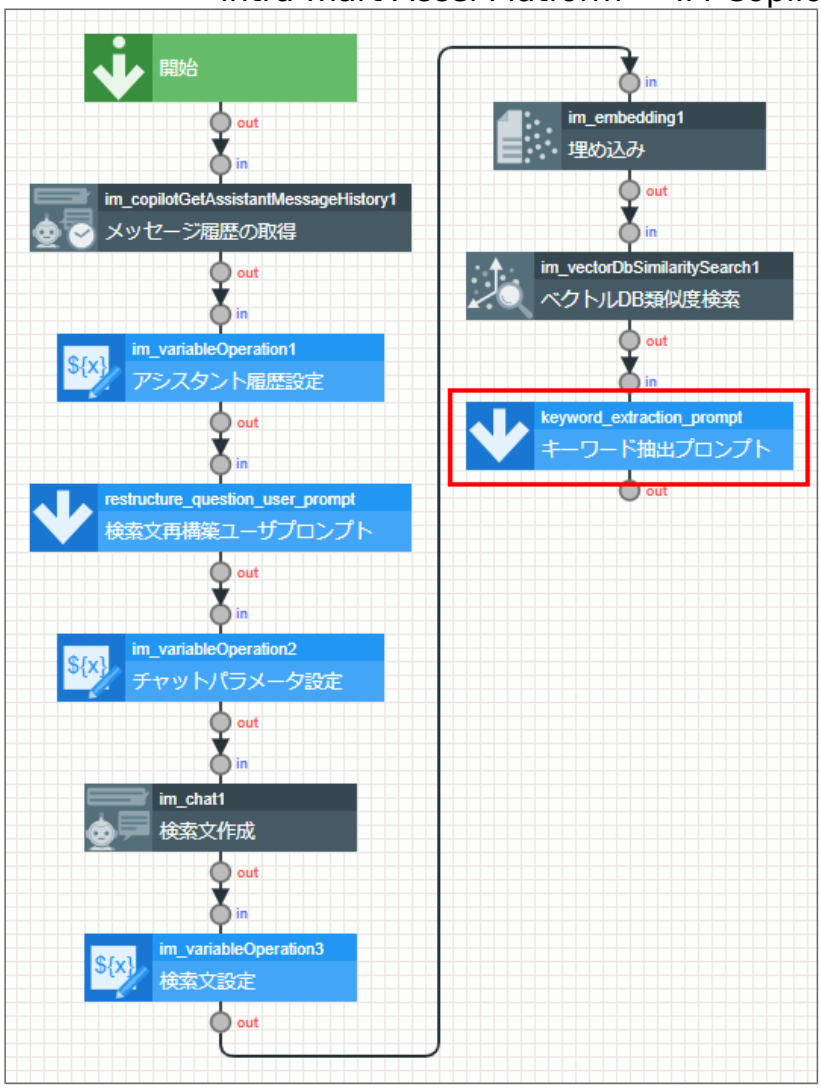
- 定数 *VECTORSTORE_CATEGORY* をタスクの入力値 *category* に設定します。
- エイリアス「*im_embedding1*」を追加します。
  - *embeddings* をタスクの入力値 *embeddings* に設定します。




キーワード検索

キーワード検索を行うことで、質問に含まれる単語やフレーズに完全一致する情報を検索できます。特に固有名詞や専門用語など、意味が変わらない単語に対して有効です。 ユーザの質問に対して完全一致する情報を検索するため、キーワード検索を行います。 検索に利用するキーワードは、生成AIによって検索文から抽出されたものを利用します。

13. ユーザ定義（テンプレート定義）を追加します。




キーワード検索を行うため、生成AIによって検索文から抽出されたキーワードを取得するため、ユーザ定義（テンプレート定義）を追加します。
パレットから「ユーザ定義追加」→「テンプレート定義新規作成」をクリックし「テンプレート定義編集」画面を表示します。
テンプレート定義編集画面の「テンプレート定義」に以下の内容を設定します。

**ユーザ定義共通設定**

| 設定項目 | 設定値 |
|---|---|
| 利用範囲 | 「フロー定義内のみで利用する」を有効にします。 |
| ユーザ定義ID | *keyword_extraction_prompt* |
| ユーザ定義名 | *キーワード抽出プロンプト* |

入力値の *data <object>* 配下項目に以下の内容を設定します。

**入力値 *data* 配下項目**

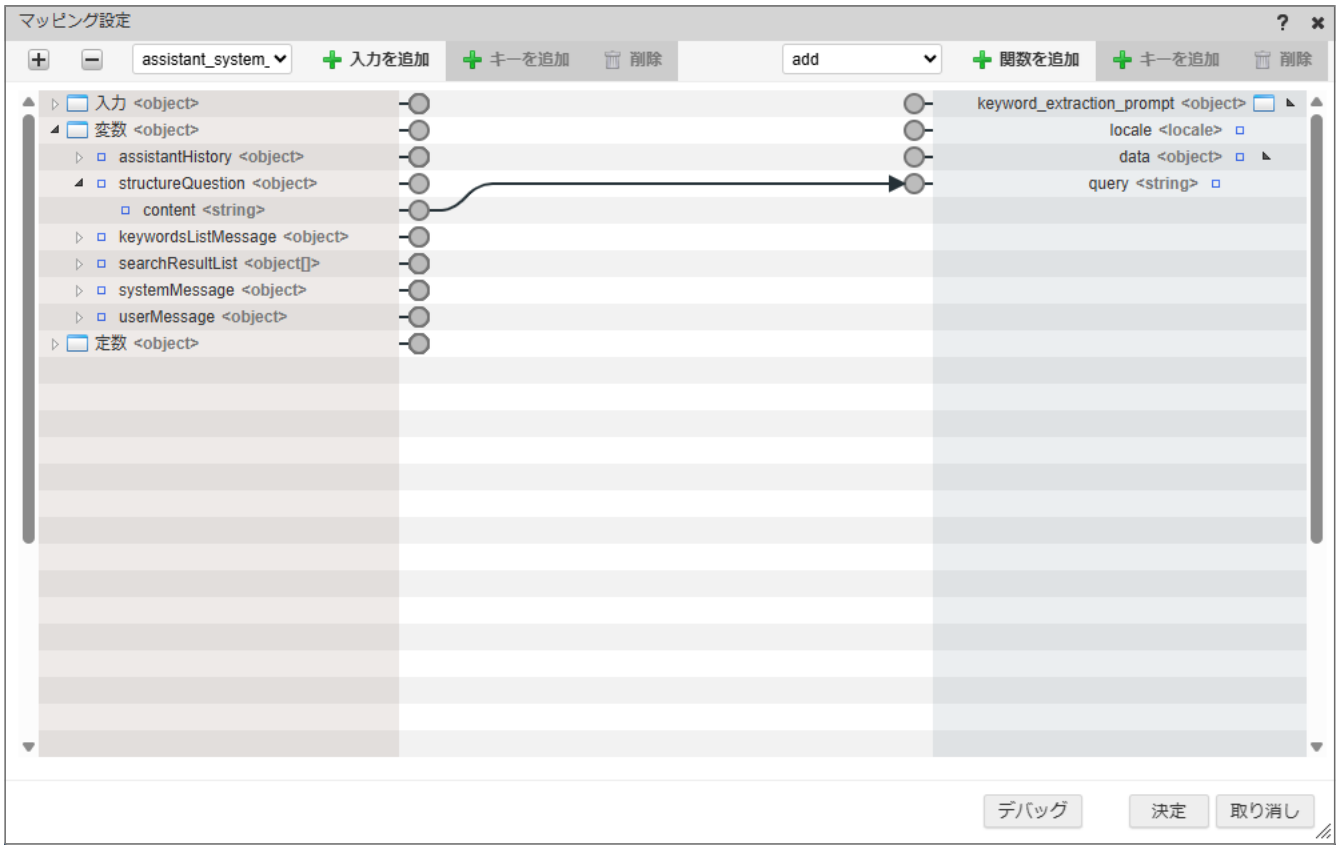
| 項目名 | データ型 |
|---|---|
| query | string |


テンプレート定義の「標準」に以下の内容を設定します。

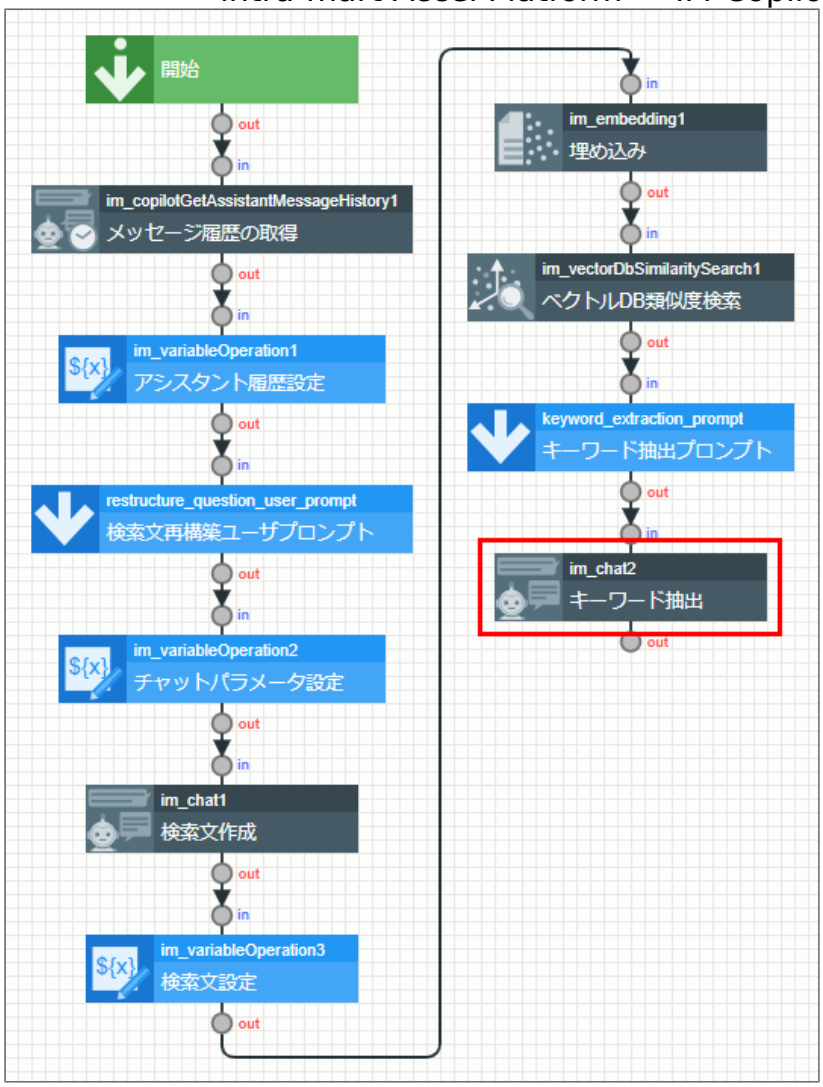
```
<#setting url_escaping_charset="UTF-8">
あなたは、質問に対する回答作成に必要な文書情報の検索をサポートするアシスタントです。
質問に関連する文書情報を検索するための検索キーワードを作成してください。
キーワードは質問文の言語に基づいて作成します。例えば質問が英語の場合は英語のキーワードを作成します。
キーワードは複数作成して構いませんが、最大でも10個までとしてください。
なお、以下のようにキーワードのみ回答してください。
キーワードはすべてカンマ（,）で区切ってください。
半角スペースで複数のキーワードをまとめてはいけません。

回答例:
キーワード１,キーワード２,キーワード３...

また、質問が不明確でキーワードを作成できない場合は、質問をそのまま回答してください。

---------------------------------

質問:
${query}
```

登録ボタンをクリックしてテンプレート定義を保存します。
「ベクトルDB類似度検索」の *out* に追加した「キーワード抽出プロンプト」の *in* に接続します。
追加した「キーワード抽出プロンプト」を選択し、マッピング設定を行います。

- 変数 *structureQuestion.content* をタスクの入力値 *data.query* に設定します。

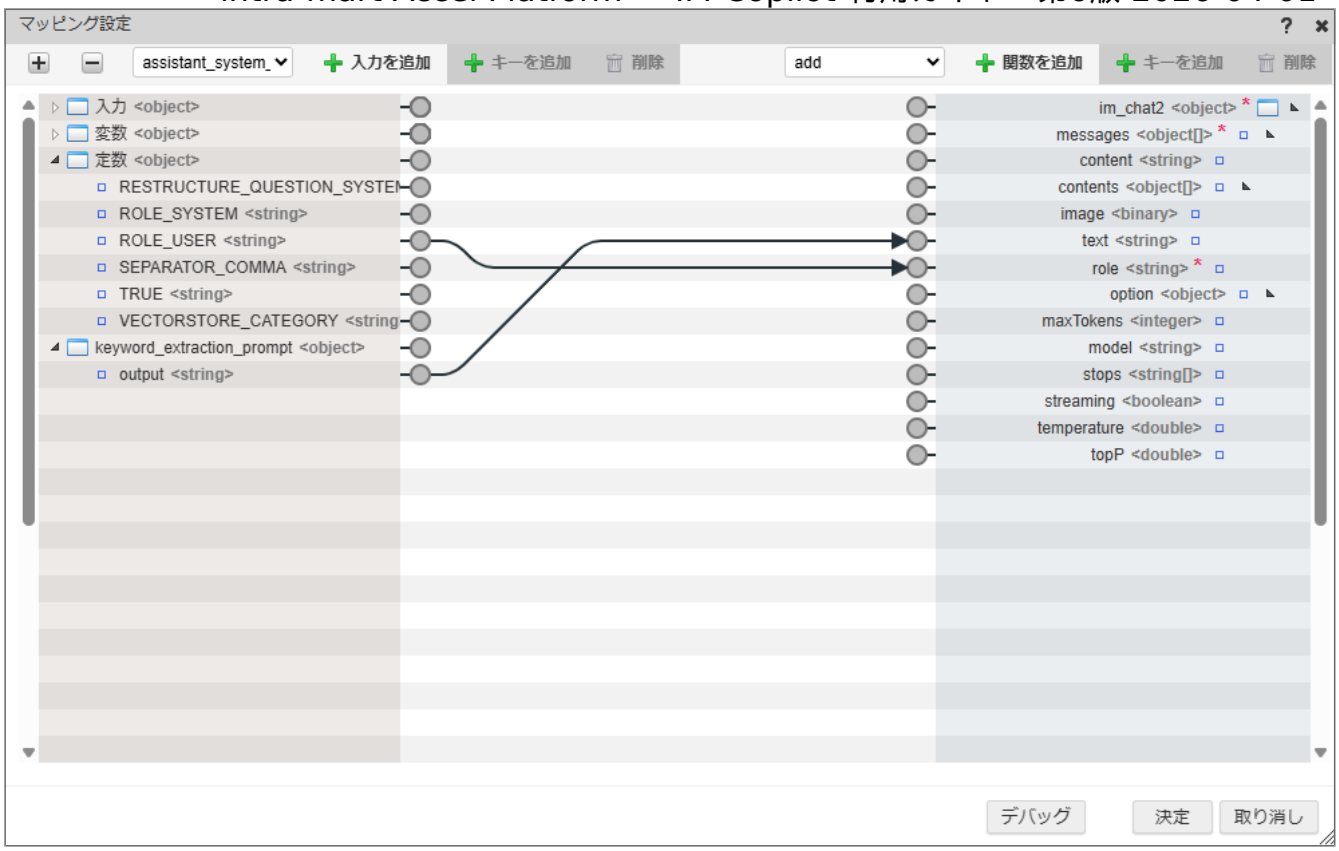



14. チャットタスクを追加します。

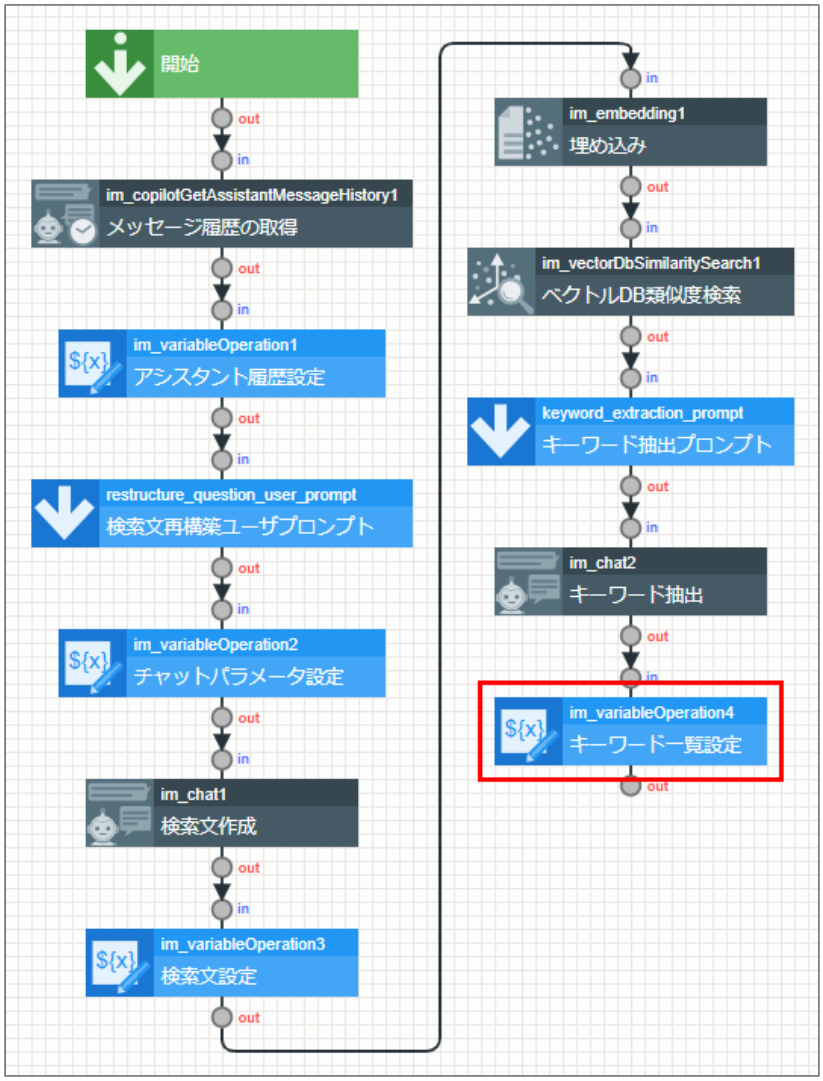



ユーザ定義（テンプレート定義）で定義したキーワード抽出の依頼を生成AIに問い合わせるため、チャットタスクを追加します。
パレットから「IM-Copilot」→「チャット」をクリックしロジックフローに追加します。
「キーワード抽出プロンプト」の *out* を追加した「チャット」の *in* に接続します。
追加した「チャット」を選択し、マッピング設定を行います。

- 定数 *ROLE_USER* をタスクの入力値 *messages.role* に設定します。
- エイリアス「*keyword_extraction_prompt*」を追加します。
  - *output* をタスクの入力値 *messages.contents.text* に設定します。



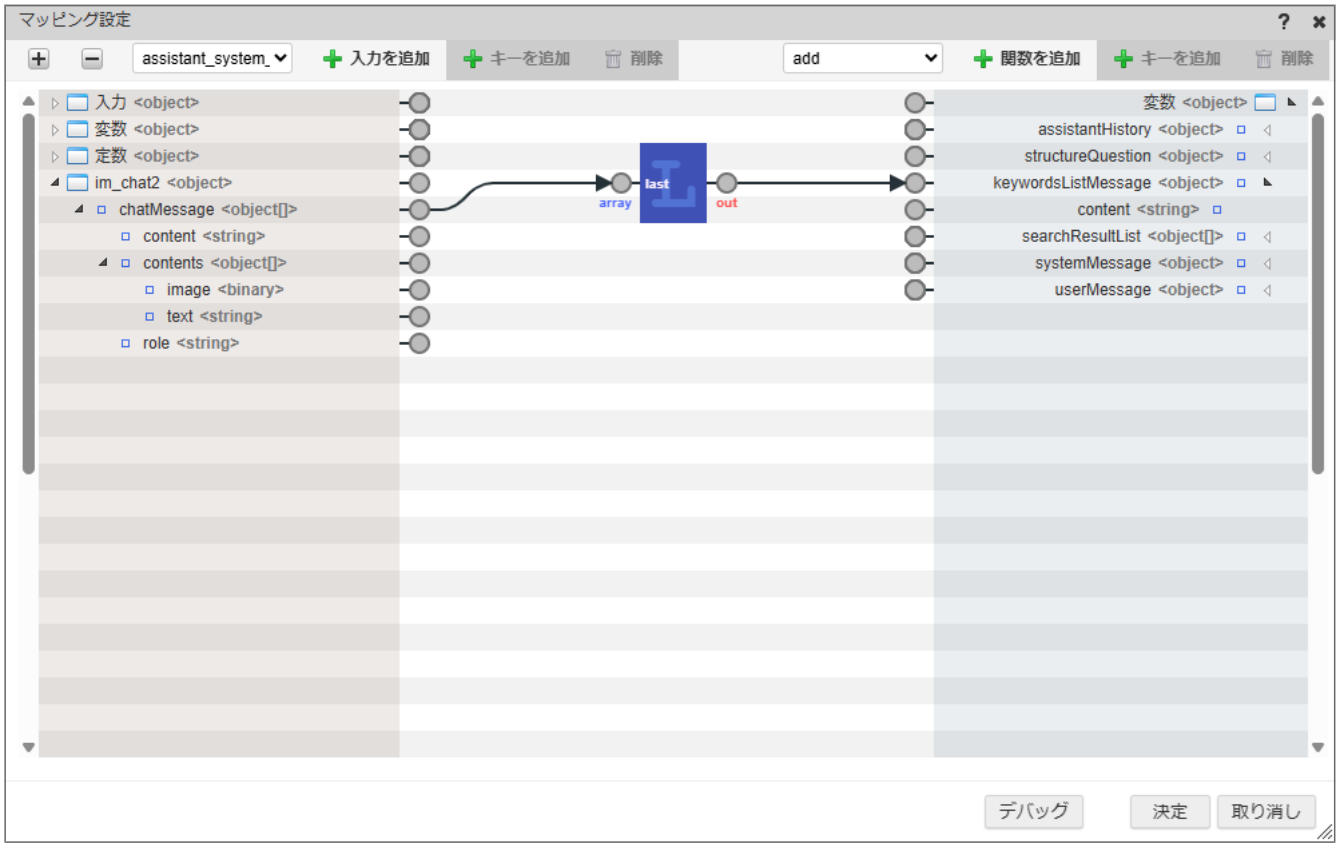

15. 変数操作タスクを追加します。



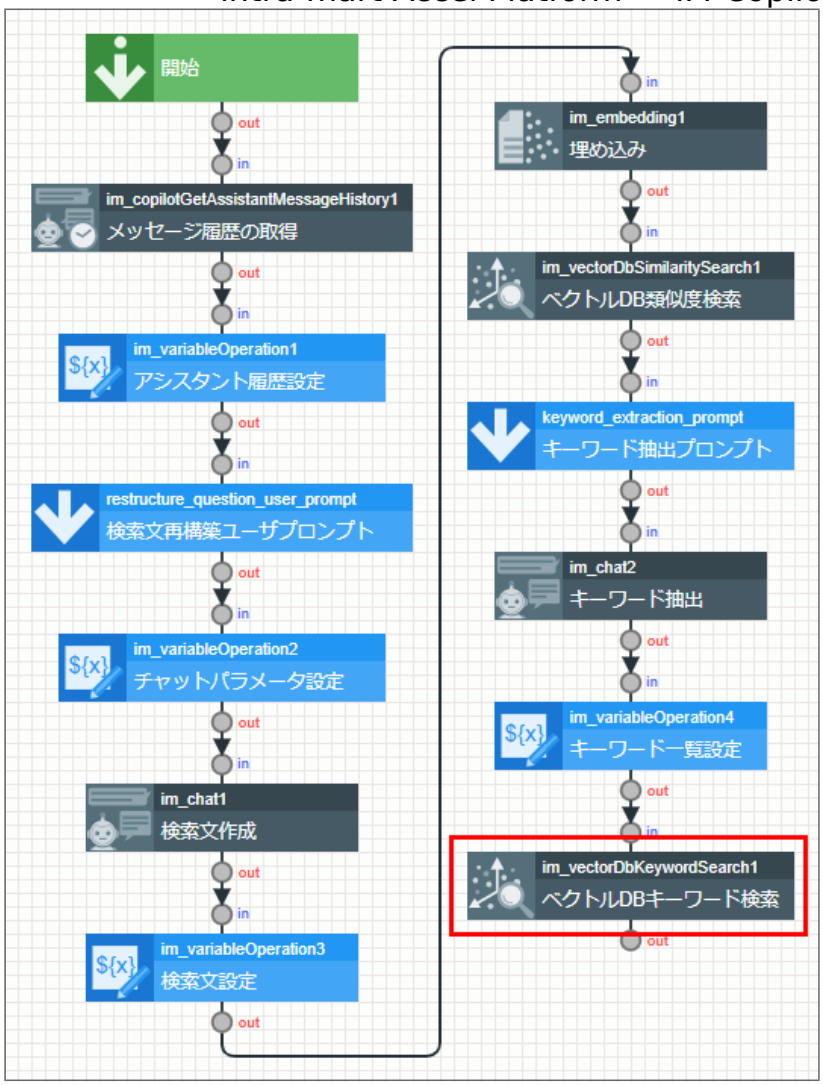

チャットタスクで取得したキーワードを変数に格納します。
パレットから「基本」→「変数操作タスク」をクリックしロジックフローに追加します。

「チャット」の *out* を追加した「変数操作タスク」の *in* に接続します。
追加した「変数操作タスク」を選択し、マッピング設定を行います。

1. エイリアス「*im_chat2*」を追加します。
2. 配列操作マッピング関数 *last* を追加します。
   - 入力値 *array* に変数 *im_chat2.chatMessage* を設定します。
   - 出力値をタスクの入力値 *keywordsListMessage* に設定します。

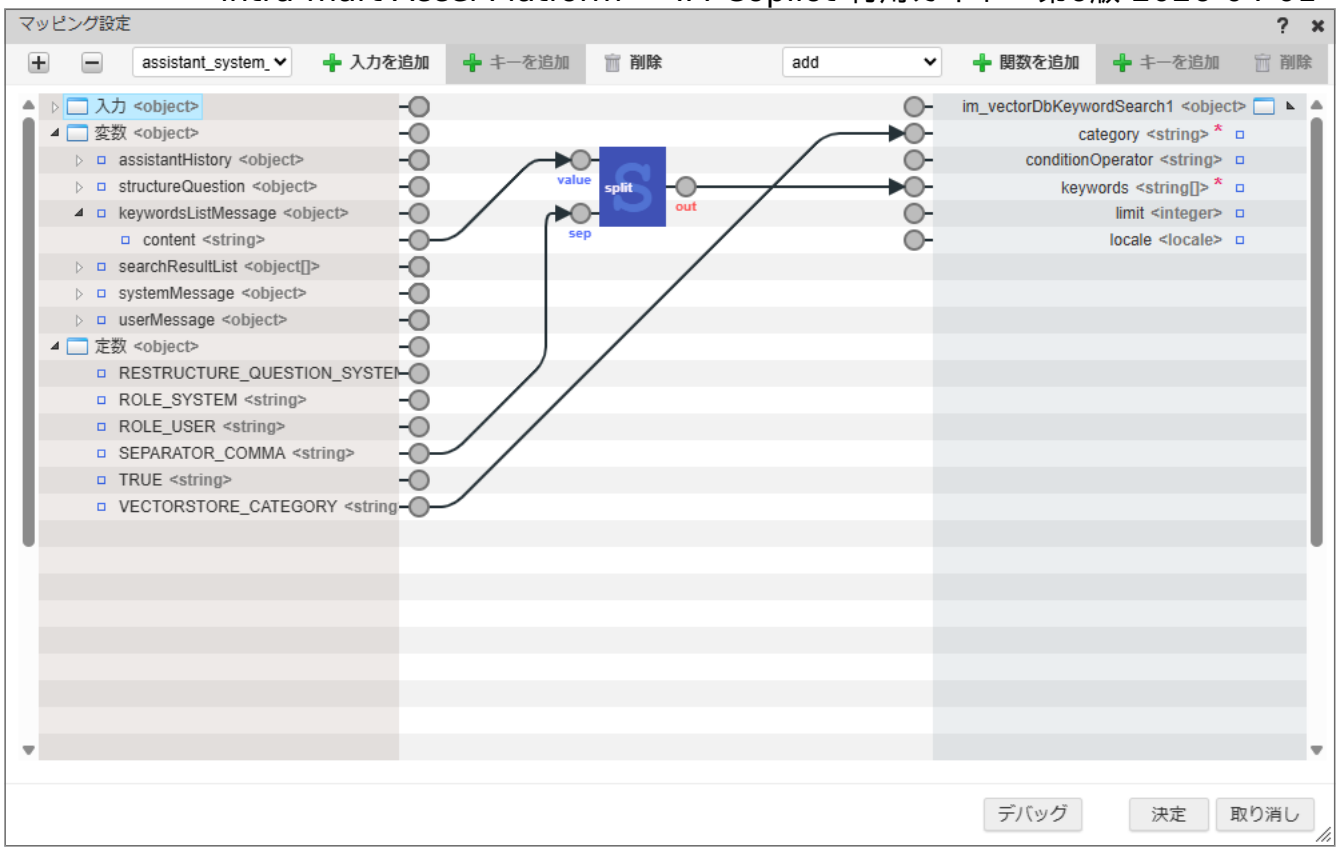



16. ベクトルDBキーワード検索タスクを追加します。




キーワード検索を行うため、キーワードを元にベクトルDBキーワード検索タスクを追加します。
パレットから「IM-Copilot」→「ベクトルDBキーワード検索」をクリックしロジックフローに追加します。
「変数操作タスク」の *out* を追加した「ベクトルDBキーワード検索」の *in* に接続します。
追加した「ベクトルDBキーワード検索」を選択し、マッピング設定を行います。

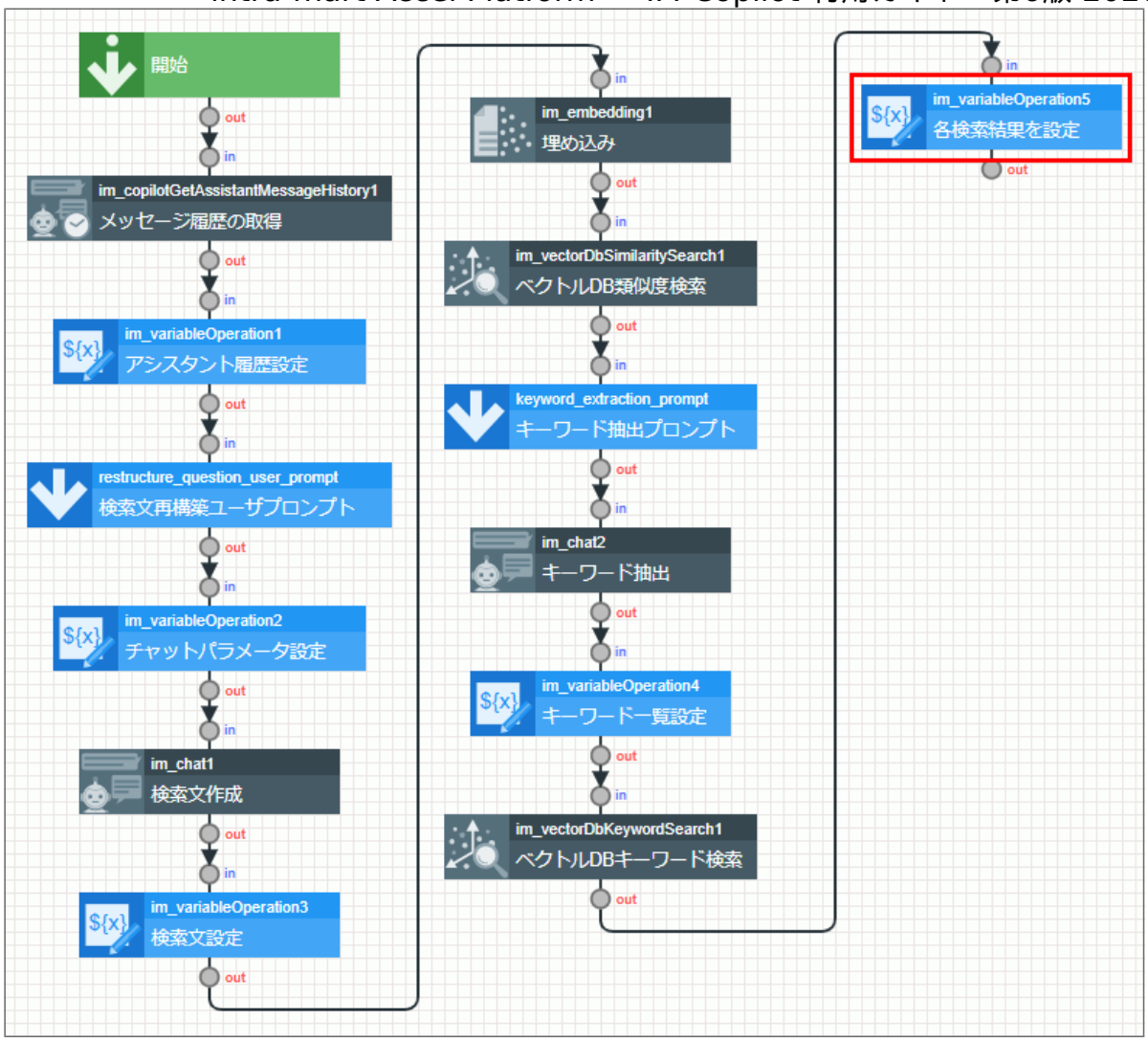
- 定数 *VECTORSTORE_CATEGORY* を入力値 *category* に設定します。
- 文字列操作マッピング関数 *split* を追加します。
  - 入力値 *value* に変数 *keywordsListMessage.content* を設定します。
  - 入力値 *sep* に定数 *SEPARATOR_COMMA* を設定します。
  - 出力値をタスクの入力値 *keywords* に設定します。




2 つの検索結果の統合

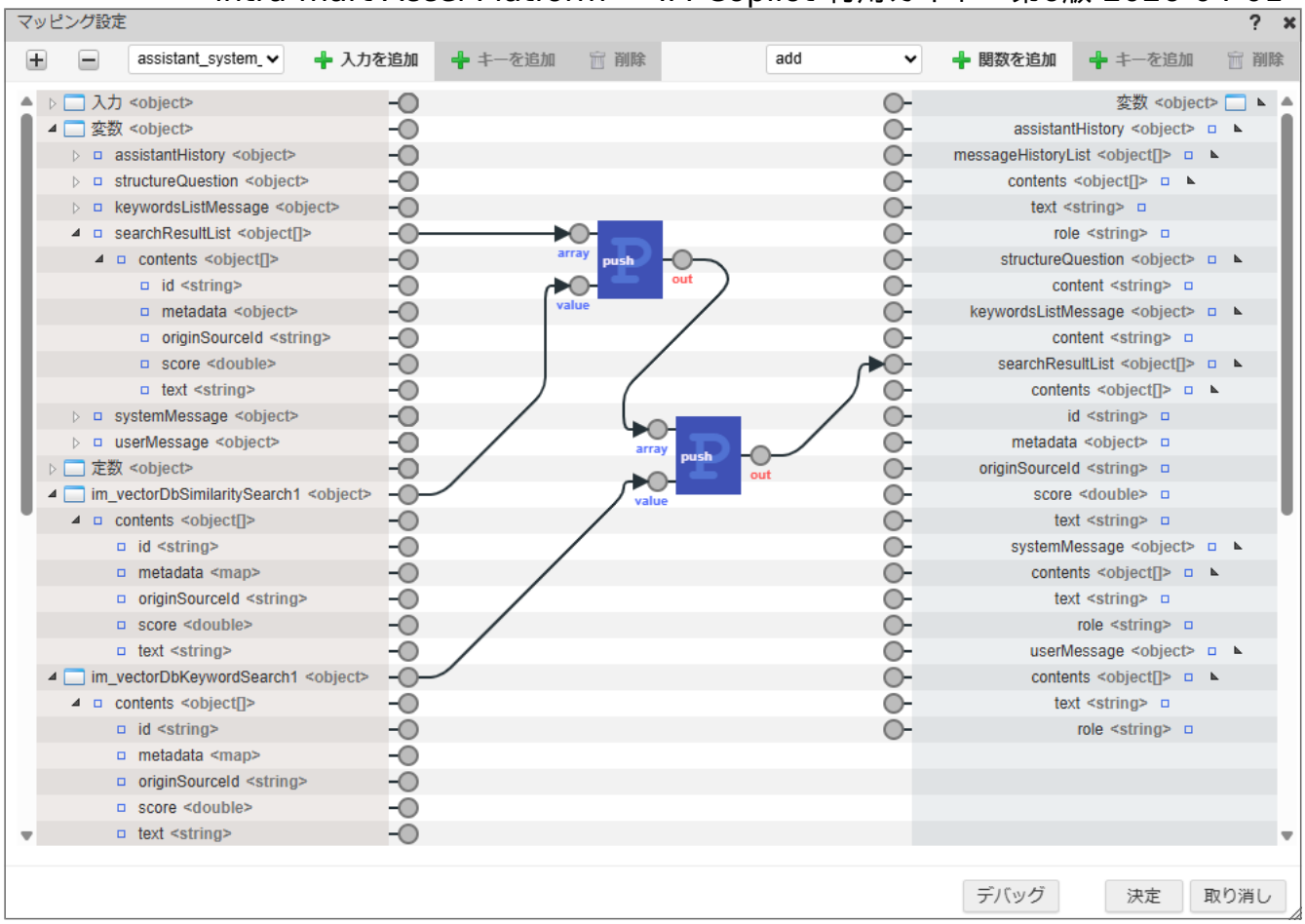
ベクトル類似度検索とキーワード検索の結果の統合を行います。検索結果の統合を行うことで、ユーザの質問に対してより適切な回答を返すことができます。

17. 変数操作タスクを追加します。

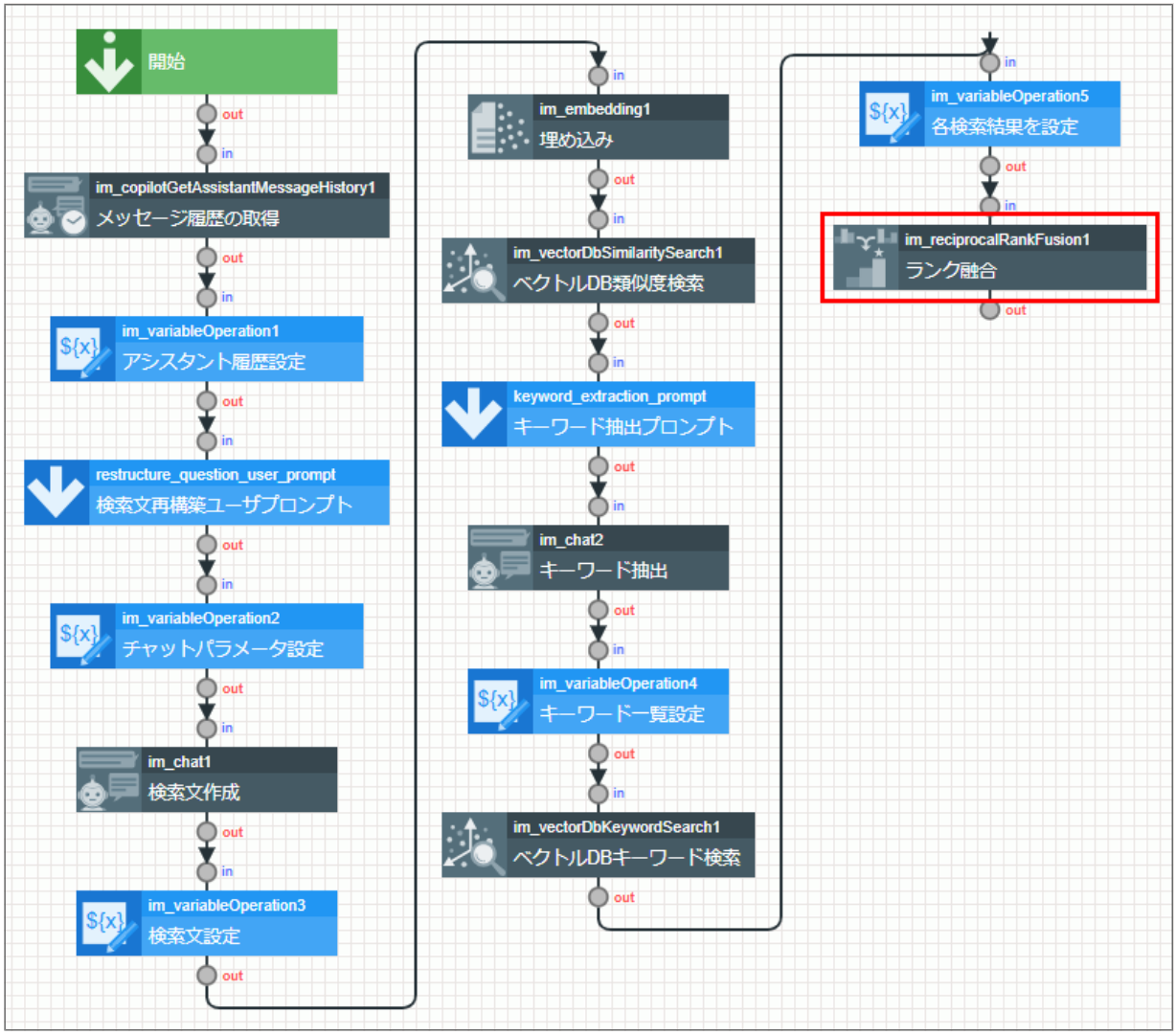



検索結果の統合を行う「ランク融合」の入力値を作成するため、変数操作タスクを追加します。
パレットから「基本」→「変数操作タスク」をクリックしロジックフローに追加します。
「ベクトルDBキーワード検索」の *out* を追加した「変数操作タスク」の *in* に接続します。
追加した「変数操作タスク」を選択し、マッピング設定を行います。

1. エイリアス「*im_vectorDbSimilaritySearch1*」を追加します。
2. エイリアス「*im_vectorDbKeywordSearch1*」を追加します。
3. 配列操作マッピング関数 *push* を追加します。
   - 入力値 *array* に変数 *searchResultList* を設定します。
   - 入力値 *value* にエイリアス *im_vectorDbSimilaritySearch1* を設定します。
4. 配列操作マッピング関数 *push* を追加します。
   - 入力値 *array* に上記で追加したマッピング関数の出力値を設定します。
   - 入力値 *value* にエイリアス *im_vectorDbKeywordSearch1* を設定します。
   - 出力値をタスクの入力値 *searchResultList* に設定します。

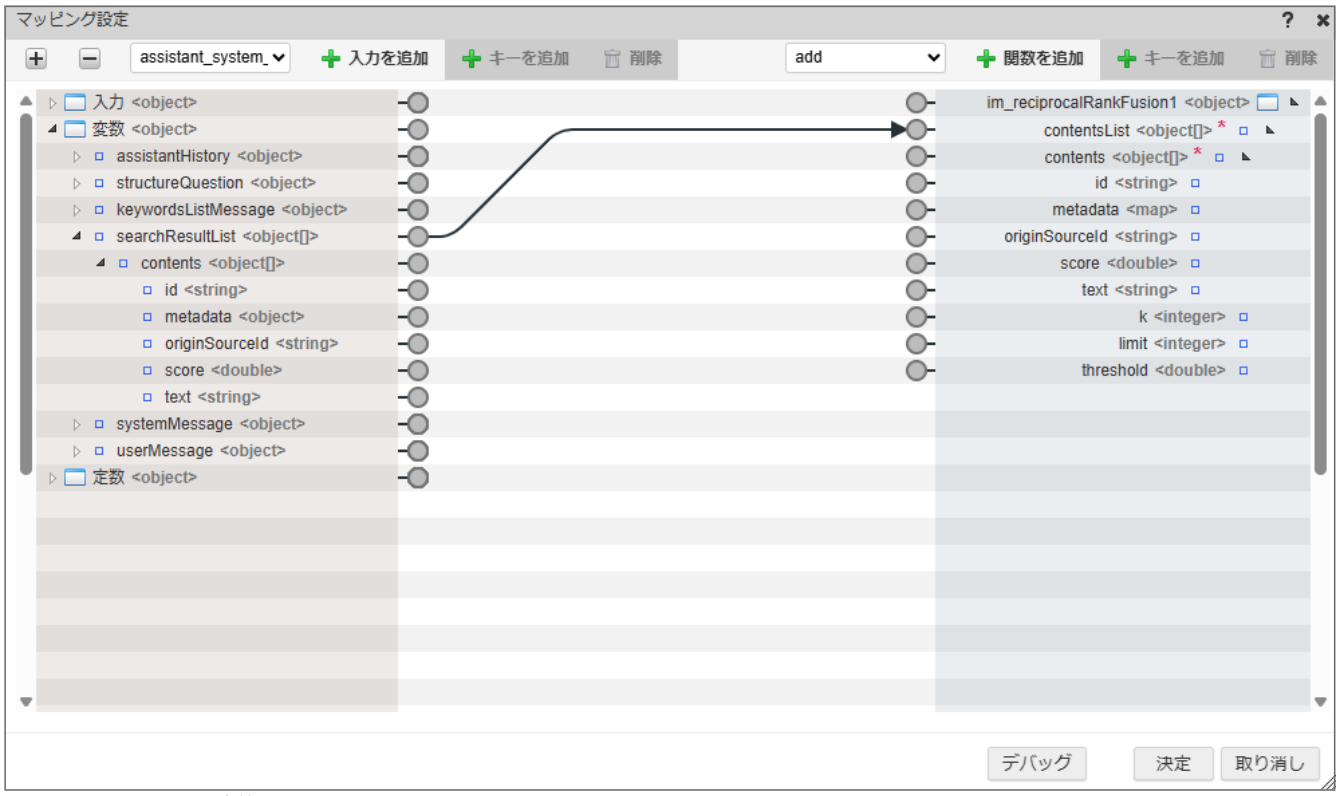



18. ランク融合タスクを追加します。



ベクトル類似度検索とキーワード検索の結果を統合するため、ランク融合タスクを追加します。
パレットから「IM-Copilot」→「ランク融合」をクリックしロジックフローに追加します。
「変数操作タスク」の *out* を追加した「ランク融合」の *in* に接続します。
追加した「ランク融合」を選択し、マッピング設定を行います。

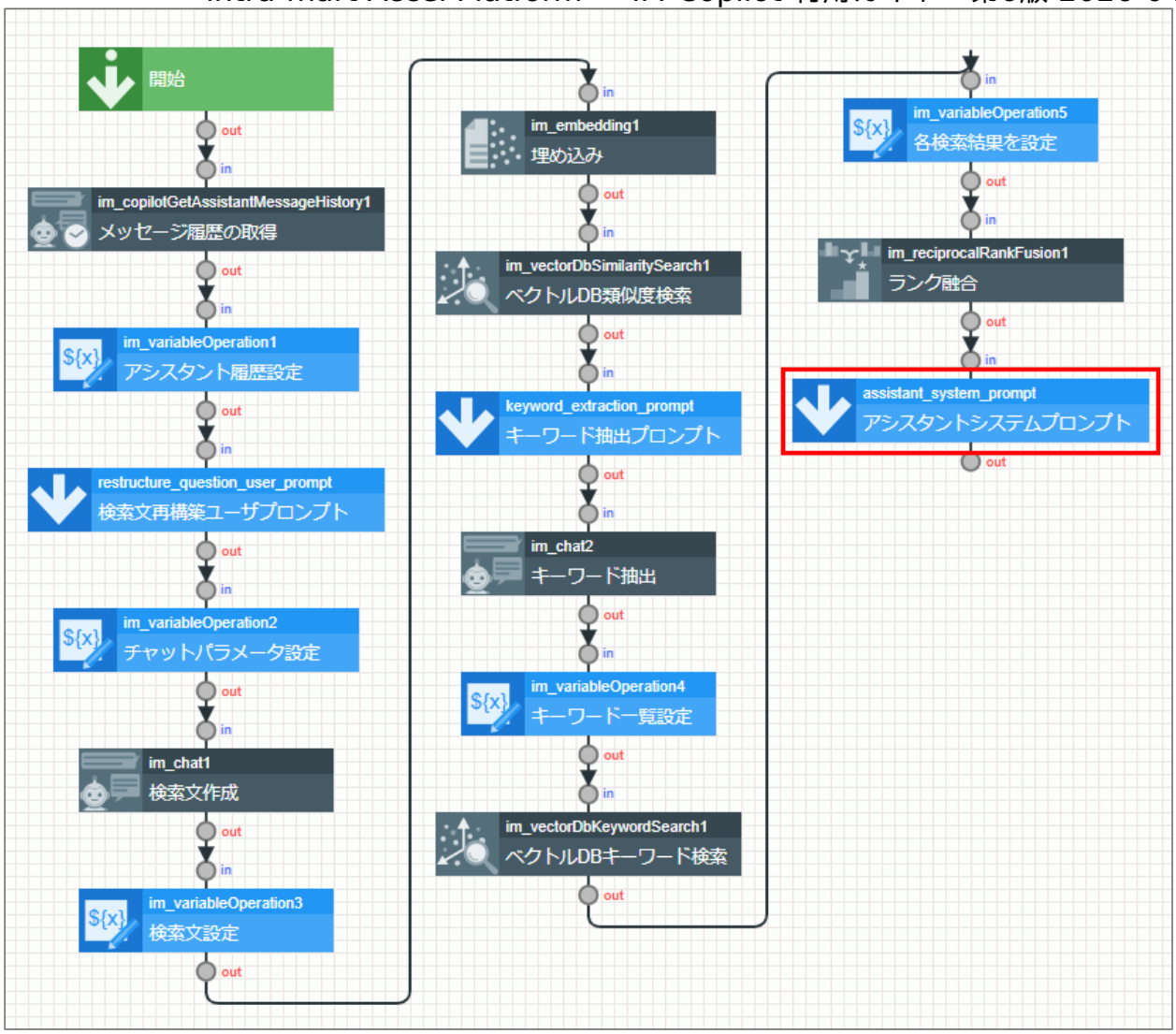
- 変数 *searchResultList* をタスクの入力値 *contentsList* に設定します。




生成AIを用いたアシスタント応答

ユーザの質問に対して適切な回答を返すため、生成AIを利用してアシスタント応答を生成します。

19. ユーザ定義（テンプレート定義）を追加します。




アシスタント応答を生成するため、生成AIに問い合わせる際に利用するシステムプロンプトを定義するユーザ定義（テンプレート定義）を追加します。
パレットから「ユーザ定義追加」→「テンプレート定義新規作成」をクリックし「テンプレート定義編集」画面を表示します。
テンプレート定義編集画面の「テンプレート定義」に以下の内容を設定します。

**ユーザ定義共通設定**

| 設定項目 | 設定値 |
|---|---|
| 利用範囲 | 「フロー定義内のみで利用する」を有効にします。 |
| ユーザ定義ID | *assistant_system_prompt* |
| ユーザ定義名 | *アシスタントシステムプロンプト* |

入力値の *data <object>* 配下項目に以下の内容を設定します。

**入力値 *data* 配下項目**

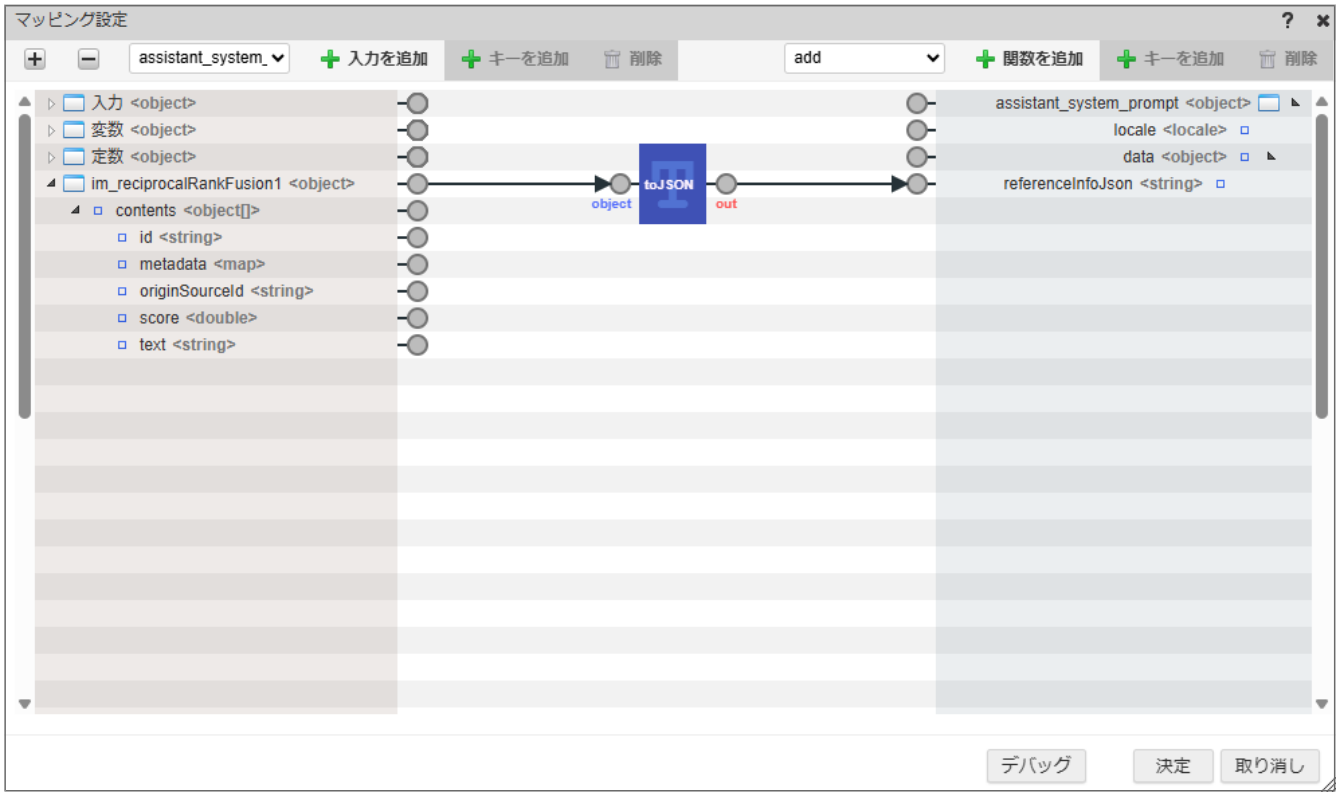
| 項目名 | データ型 |
|---|---|
| referenceInfoJson | string |


テンプレート定義の「標準」に以下の内容を設定します。

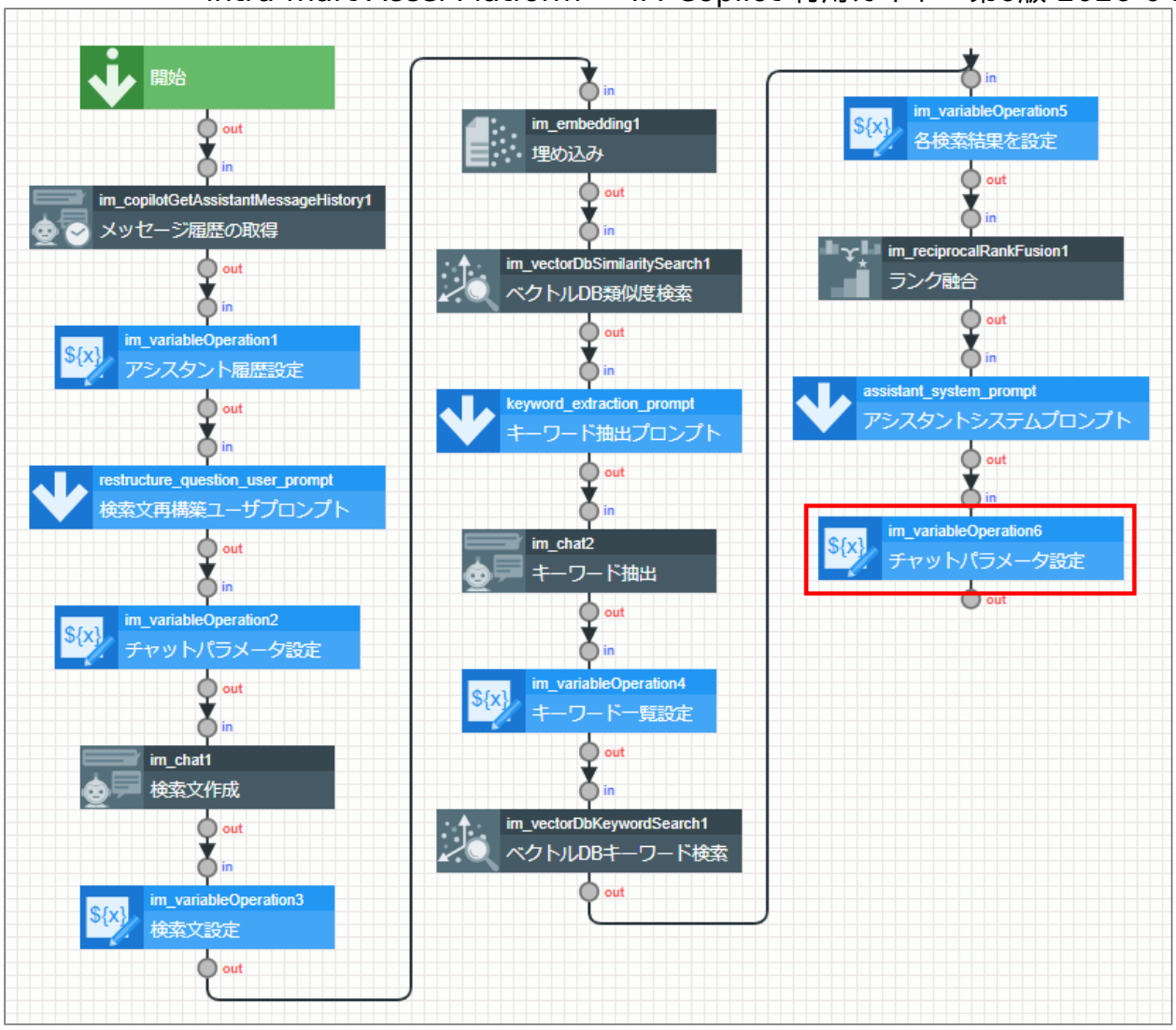
```
<#setting url_escaping_charset="UTF-8">
あなたは、参考情報のみに基づいてユーザーの質問に答えるアシスタントです。次のガイドラインに従ってください：

1. 参考情報に含まれる情報のみを使用して回答して下さい。
2. 回答は可能な限り詳細に行い、参考情報の該当部分を参照しながら説明して下さい。
3. 差別的、攻撃的、またはわいせつな言葉を使用する内容について回答できないと出力して下さい。
4. 参考情報が無い場合には、回答の根拠の提示は出来ないと伝えつつ、回答できる場合は回答して下さい、そうでない場合は回答できないと出力して下さい。

参考情報は以下の形式で提供されます：
${referenceInfoJson}

このデータにのみ基づいて回答を行ってください。
ユーザーが質問で利用している言語と同じ言語で回答してください。
```

登録ボタンをクリックしてテンプレート定義を保存します。
「ランク融合」の *out* に追加した「アシスタントシステムプロンプト」の *in* に接続します。
追加した「アシスタントシステムプロンプト」を選択し、マッピング設定を行います。

1. エイリアス「*im_reciprocalRankFusion1*」を追加します。
2. JSONマッピング関数 *toJSON* を追加します。
   - 入力値 *object* にエイリアス *im_reciprocalRankFusion1* を設定します。
   - 出力値をタスクの入力値 *data.referenceInfoJson* に設定します。

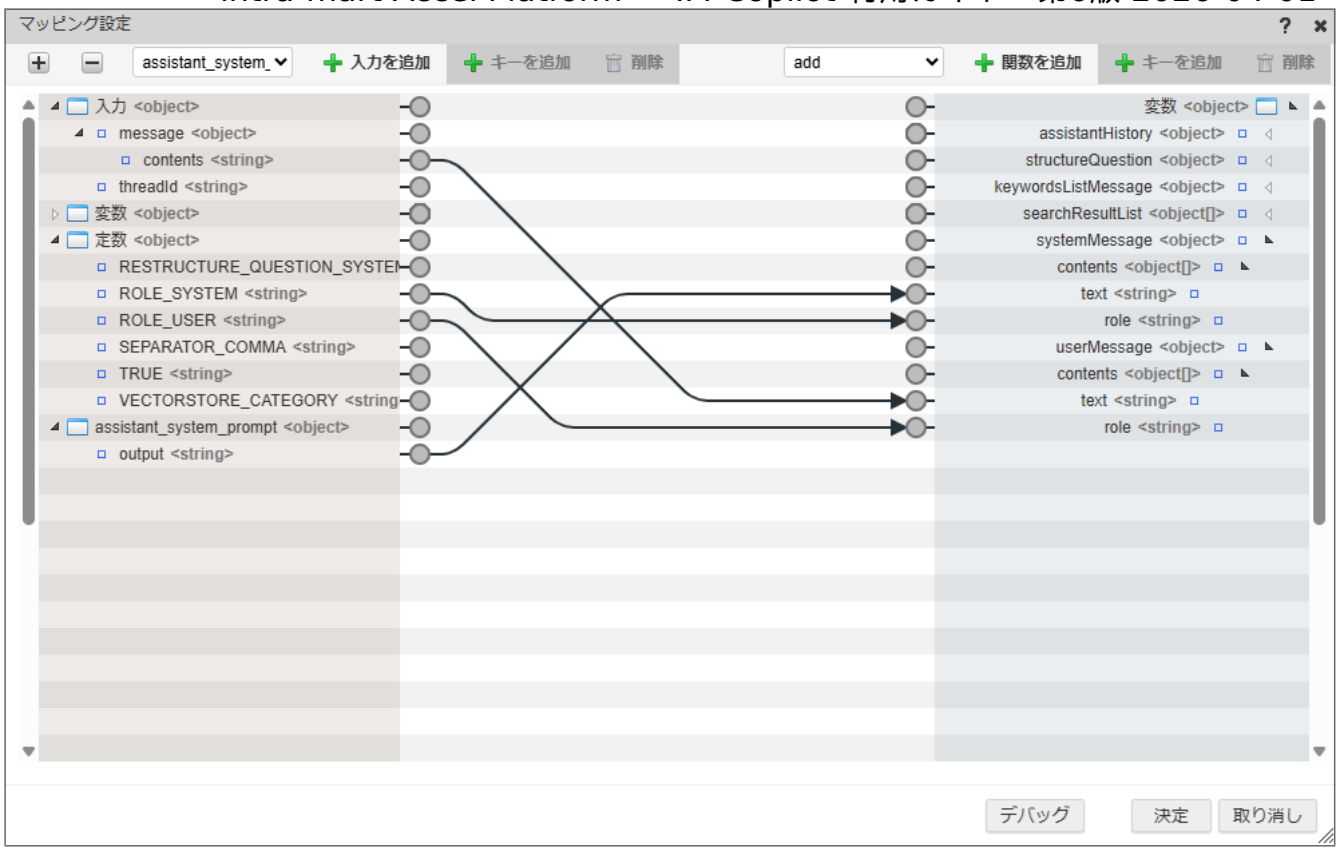



20. 変数操作タスクを追加します。

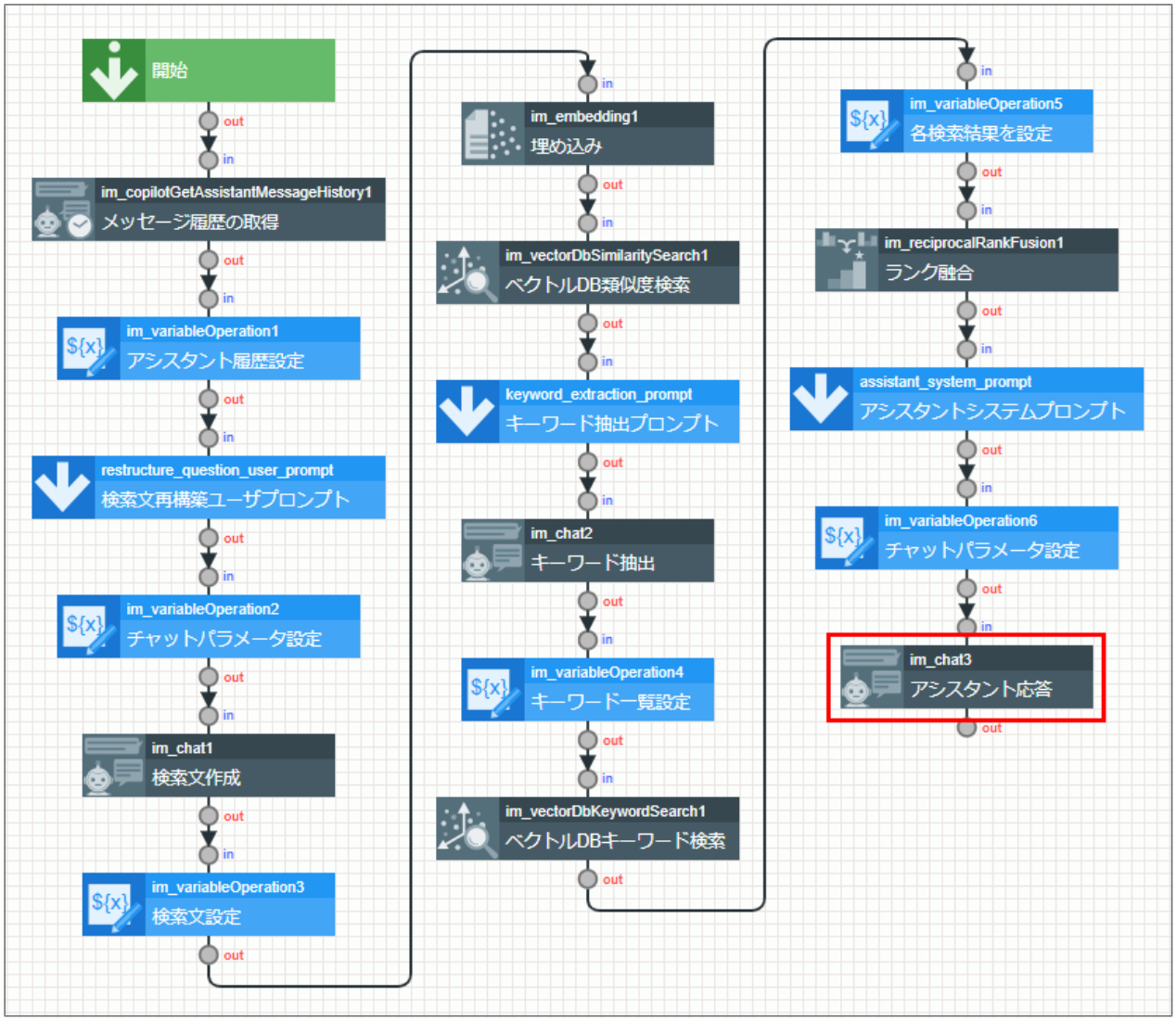



生成AIに問い合わせる際に利用するシステムプロンプト、および、ユーザプロンプトを変数に格納します。
パレットから「基本」→「変数操作タスク」をクリックしロジックフローに追加します。
「アシスタントシステムプロンプト」の *out* を追加した「変数操作タスク」の *in* に接続します。
追加した「変数操作タスク」を選択し、マッピング設定を行います。

- 定数 *ROLE_SYSTEM* をタスクの入力値 *systemMessage.role* に設定します。
- エイリアス「*assistant_system_prompt*」を追加します。
  - *output* をタスクの入力値 *systemMessage.contents.text* に設定します。
- 定数 *ROLE_USER* をタスクの入力値 *userMessage.role* に設定します。
- 入力 *message.contents* をタスクの入力値 *userMessage.contents.text* に設定します。



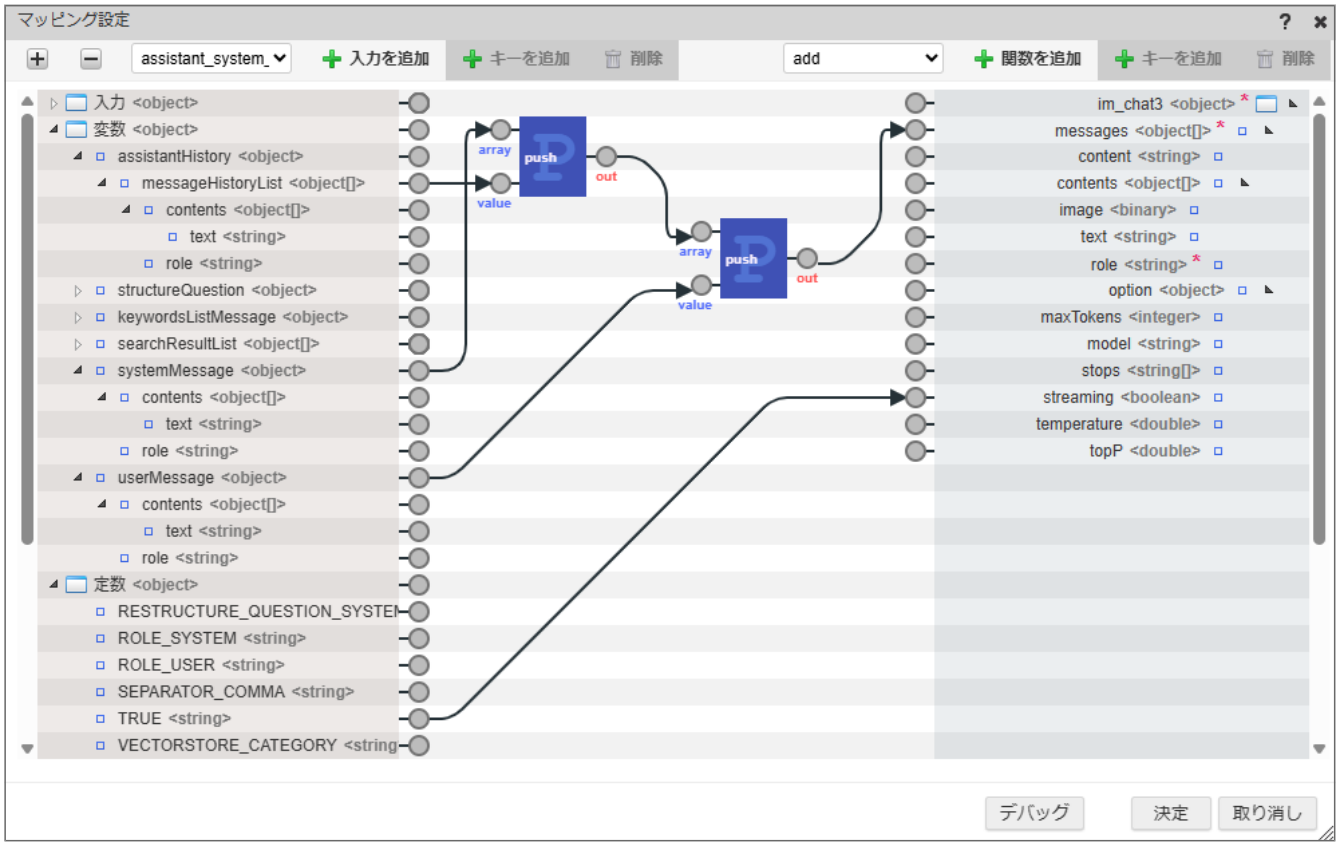

21. チャットタスクを追加します。



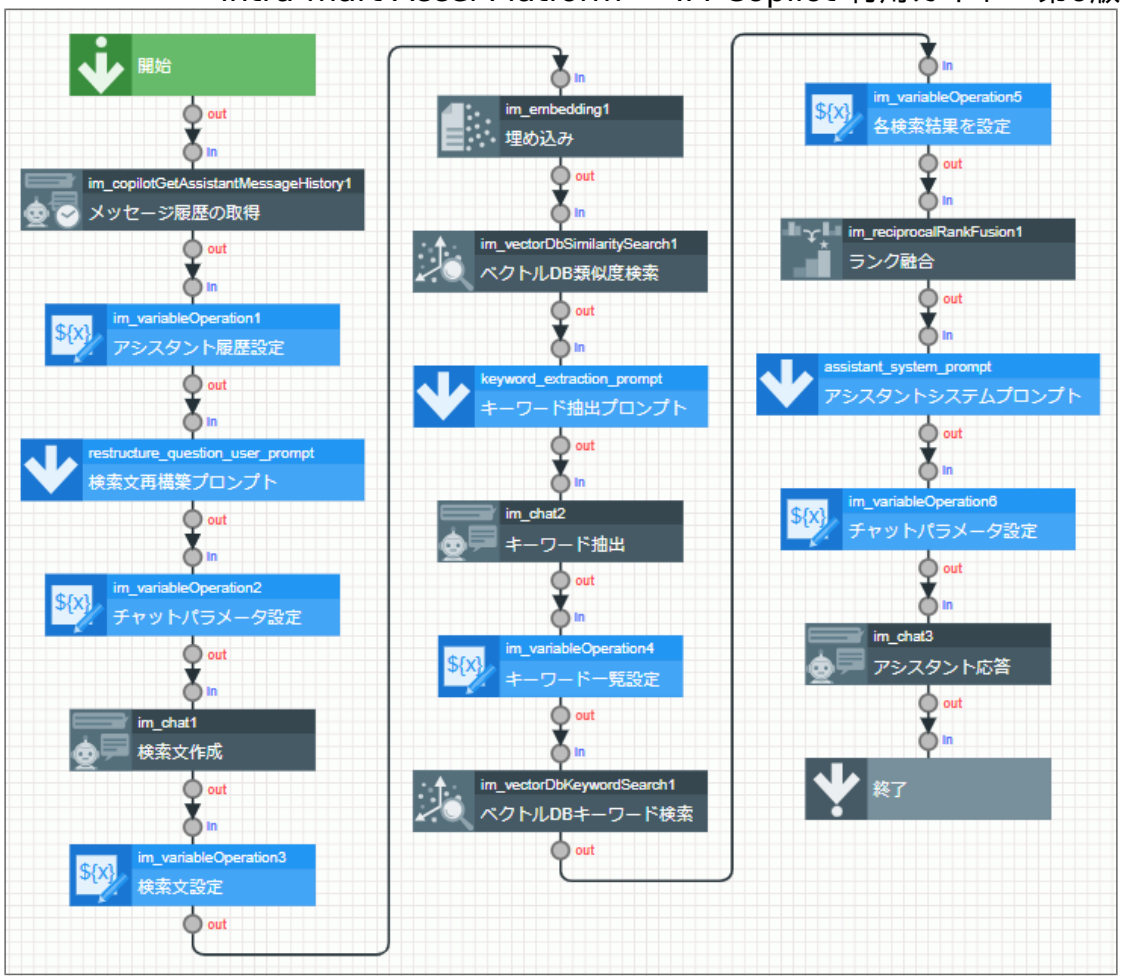

検索した情報を基に生成AIに問い合わせ、その応答をクライアントに返すために、チャットタスクを追加します。
パレットから「IM-Copilot」→「チャット」をクリックしロジックフローに追加します。

「変数操作タスク」の *out* を追加した「チャット」の *in* に接続します。
追加した「チャット」を選択し、マッピング設定を行います。

- 配列操作マッピング関数 *push* を追加します。
  - 入力値 *array* に変数 *systemMessage* を設定します。
  - 入力値 *value* に変数 *assistantHistory.messageHistoryList* を設定します。
- 配列操作マッピング関数 *push* を追加します。
  - 入力値 *array* に上記で追加したマッピング関数の出力値を設定します。
  - 入力値 *value* に変数 *userMessage* を設定します。
  - 出力値をタスクの入力値 *messages* に設定します。
- 定数 *TRUE* をタスクの入力値 *streaming* に設定します。




「チャット」タスクの *out* を「終了」エレメントの *in* に接続します。




**コラム**

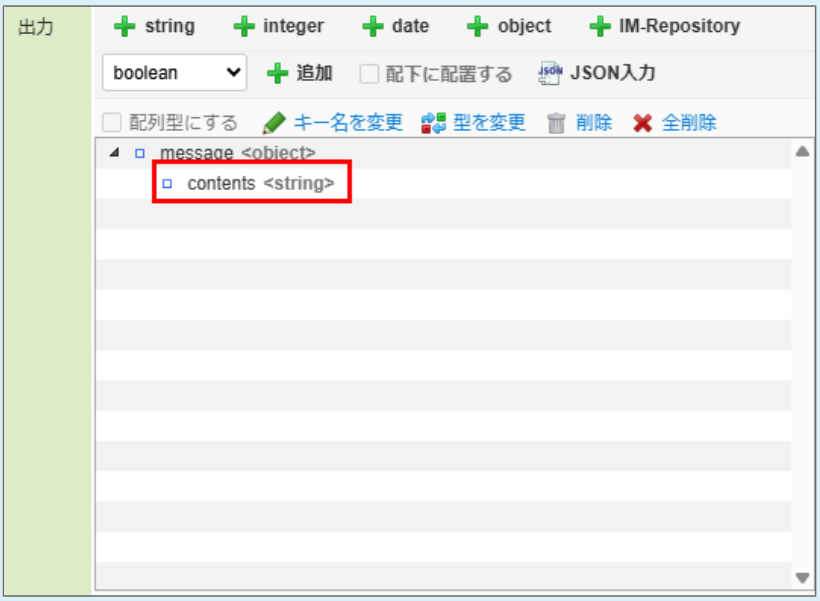
ストリーミング形式でのクライアントへの応答について

生成AIに問い合わせる際に、アシスタント応答をストリーミング形式で返すためチャットタスクの *streaming* オプションに、*true* を設定します。
ストリーミング形式で応答を返すことにより、アシスタント応答をリアルタイムにクライアントに返すことができます。

コラム

チャットタスクを利用しないクライアントへの応答について

チャットタスクを利用せずにアシスタント応答を返す場合、応答内容をロジックフローの出力値 *message.contents* に文字列として設定することでクライアントに応答を返すことができます。
この場合、あらかじめロジックフローの入出力設定で出力値を設定し、「終了」エレメントのマッピングで出力値に文字列を指定します。




22. ロジックフローの保存を行います。

    ロジックフロー定義編集画面ツールバーの「保存」をクリックしロジックフローを保存します。
    以下の値で保存します。

    - フロー定義ID: *sample-rag-assistant*
    - フロー定義名: *サンプルRAGアシスタント*

アシスタントの設定
アシスタント定義の作成

作成したロジックフローをアシスタントとして動作させるため、アシスタント定義を作成します。

1. アシスタント定義一覧画面を表示します。
   「サイトマップ」→「Copilot」→「アシスタント定義」→「アシスタント定義一覧」をクリックし、「アシスタント定義一覧」画面を表示します。

2. アシスタント定義の新規作成を行います。
   「*アシスタント定義を登録する*」を参考にロジックフローアシスタントのアシスタント定義を新規作成します。
   以下の値でアシスタント定義を作成します。
   - アシスタント定義ID: *sample-rag-assistant*
   - アシスタント定義名: *サンプルRAGアシスタント*

アシスタントの認可設定

アシスタントを利用する権限をユーザに付与するため、アシスタントの認可設定を行います。

1. アシスタント定義一覧画面を表示します。
   「サイトマップ」→「Copilot」→「アシスタント定義」→「アシスタント定義一覧」をクリックし、「アシスタント定義一覧」画面を表示します。

2. アシスタントの認可設定を行います。
   作成したアシスタント定義「サンプルRAGアシスタント」を一覧に表示します。


「*アシスタント定義の認可設定をする*」を参考に作成したアシスタントの認可設定を行います。
「アシスタントの認可設定」画面で「認可設定を開始する」をクリックし、リソース「サンプルRAGアシスタント」に対して「認証済みユーザ」を許可に設定します。

> **コラム**
> 本セクションでは一例として「認証済みユーザ」を許可に設定します。運用に合わせて適切な設定を行ってください。

### アシスタントの動作確認

作成したアシスタントは「*共通アシスタント実行画面*」で動作確認が行えます。
「*共通アシスタント実行画面*」でサンプルRAGアシスタントを選択して、アシスタントの動作確認を行ってください。

> **コラム**
> 作成したロジックフローアシスタントの実行にはベクトルデータ構築が必要です。ベクトルデータ構築が完了していない場合は、アシスタントの動作確認ができません。
> 「*ベクトルデータ構築ジョブの作成*」で作成したジョブを実行の後、アシスタントの動作確認を行ってください。
> ジョブネット管理画面で「*ジョブネットの作成*」で作成したジョブネットを選択して「即時実行」を行ってください。

画面の作成

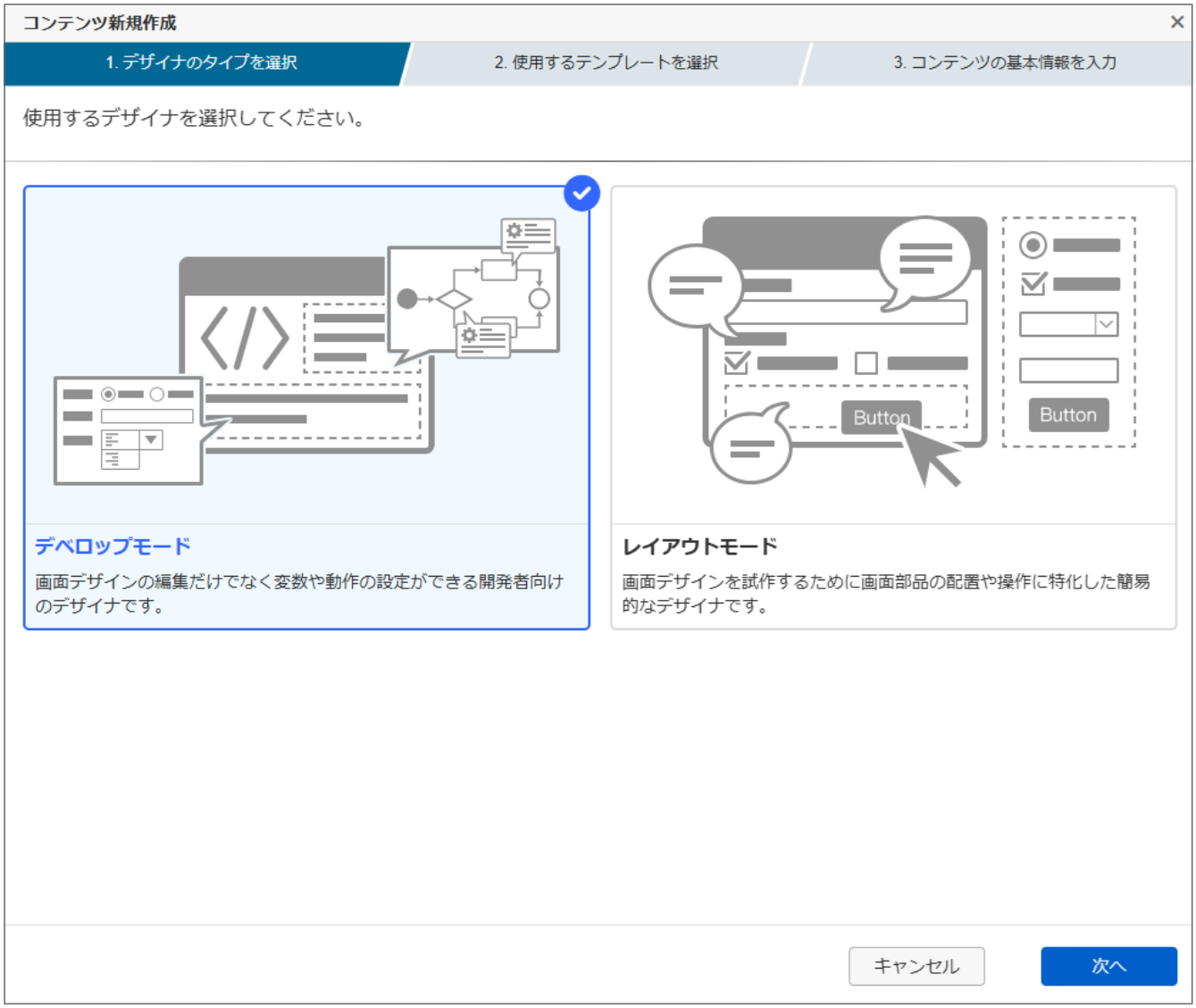
IM-BloomMaker を用いてアシスタントの画面を作成します。

コンテンツの作成

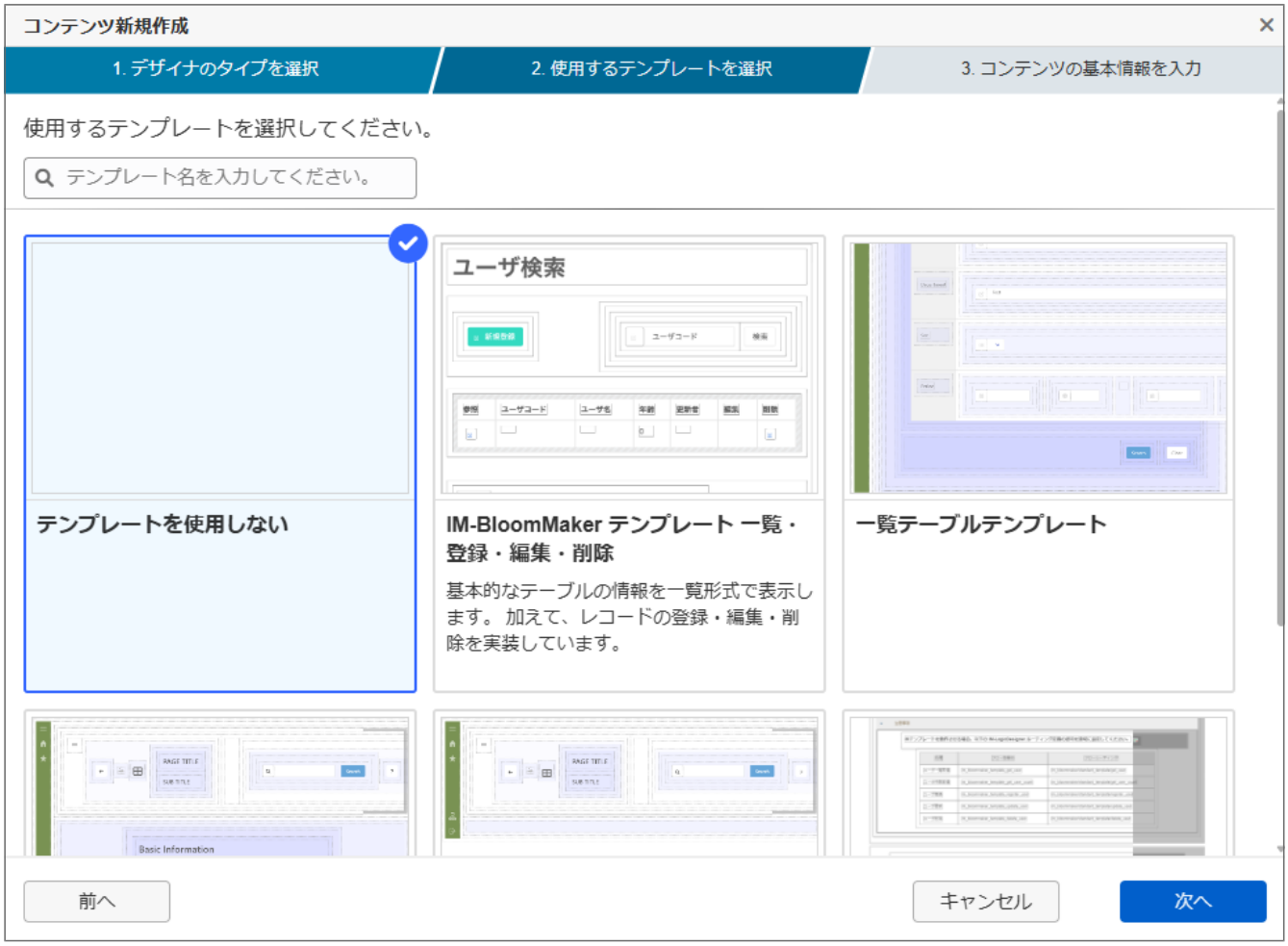
本セクションでは、アシスタントのチャット画面のみを含むシンプルな画面を作成します。

1. IM-BloomMaker コンテンツ一覧画面を開きます。
   「サイトマップ」→「BloomMaker」→「コンテンツ一覧」の順にクリックし、「コンテンツ一覧」画面を表示します。

2. コンテンツを新規作成します。
   コンテンツを新規登録します。
   右側のカテゴリツリーでコンテンツを追加するカテゴリを選択し、ツールバーの「コンテンツ新規作成」ボタンをクリックします。必要に応じてカテゴリの登録を行ってください。
   コンテンツ新規登録ダイアログが表示されます。
   デザイナのタイプを選択で「デベロップモード」を選択し、「次へ」ボタンをクリックします。




使用するテンプレートを選択で「テンプレートを使用しない」を選択し、「次へ」ボタンをクリックします。




コンテンツの基本情報入力欄に以下を入力します。

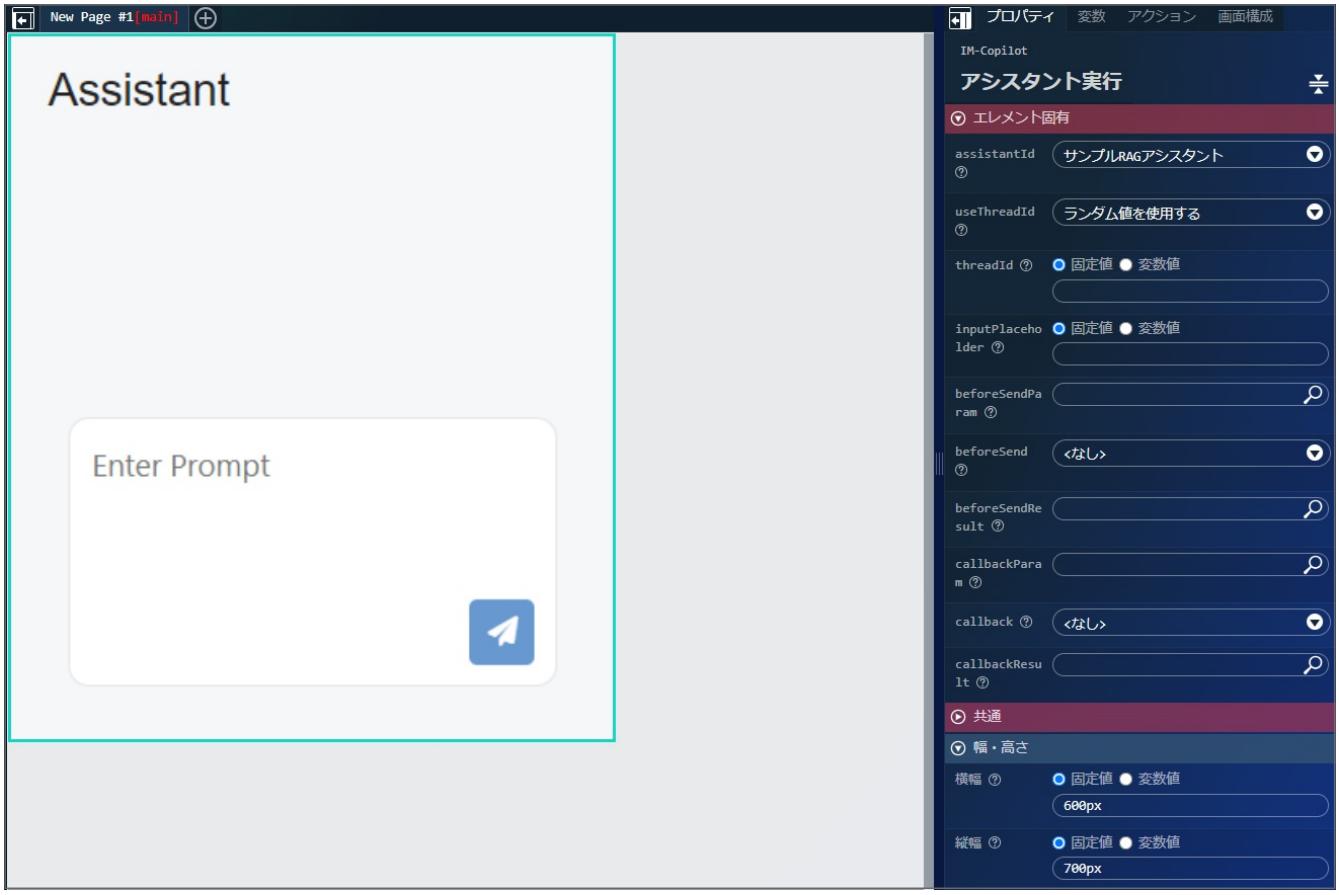
- コンテンツID: *sample_rag_assistant*
- コンテンツ名: *サンプルRAGアシスタント*
- コンテンツ種別: *imds*

入力が完了したら「登録」ボタンをクリックします。コンテンツが登録され、コンテンツのデザイナ画面が表示されます。

3. チャット画面を作成します。
   左側のパレットから「IM-Copilot」→「アシスタント実行」をドラッグ＆ドロップし、コンテンツに配置します。


配置したアシスタント実行エレメントを選択して、右側のプロパティペインで以下を設定します。

- エレメント固有
  - assistantId: *サンプルRAGアシスタント*
    - 「*アシスタント作成*」 で作成したアシスタントを選択します。
  - useThreadId: *ランダムな値を使用する*
    - メッセージ履歴を利用するアシスタントであるためスレッドIDを指定します。
- 幅・高さ
  - 横幅: *600px*
  - 縦幅: *700px*




ヘッダの左端にある「上書き保存」ボタンをクリックして、コンテンツを保存します。

ルーティングの作成

ユーザが作成したコンテンツにアクセスするためのルーティングを作成します。

1. IM-BloomMaker ルーティング定義一覧画面を開きます。
   「サイトマップ」→「BloomMaker」→「ルーティング定義一覧」の順にクリックし、「ルーティング定義一覧」画面を表示します。

2. ルーティングを新規作成します。
   ルーティングを新規登録します。
   右側のカテゴリツリーでルーティングを追加するカテゴリを選択し、ツールバーの「ルーティング新規作成」ボタンをクリックします。
   必要に応じてカテゴリの登録を行ってください。
   ルーティング情報入力欄が表示されます。
   ルーティング情報入力欄に以下の情報を入力します。
   - ルーティングID: *sample_rag_assistant*
   - コンテンツ: 「*コンテンツの作成*」 で作成したコンテンツを選択します
   - URL: *sample_rag_assistant*
   - ルーティング名: *サンプルRAGアシスタント*


入力が完了したら「登録」ボタンをクリックし、ルーティングを作成します。

ルーティングの認可設定

アシスタントの画面にアクセスするための認可設定を行います。

1. IM-BloomMaker ルーティング定義一覧画面を開きます。
   「サイトマップ」→「BloomMaker」→「ルーティング定義一覧」の順にクリックし、「ルーティング定義一覧」画面を表示します。

2. ルーティングの認可設定を行います。
   「*ルーティングの作成*」 で作成したルーティングを選択します。
   認可URIの「認可設定」アイコンをクリックし「認可設定」画面を表示します。


認可設定画面で「認可設定を開始する」をクリックし、リソース「サンプルRAGアシスタント」に対して「認証済みユーザ」を許可に設定します。

> **コラム**
> 本セクションでは一例として「認証済みユーザ」を許可に設定します。運用に合わせて適切な設定を行ってください。

アシスタントの画面を表示

作成したアシスタントの画面を表示します。ブラウザで以下の URL にアクセスします。

`http://<ホスト名>:<ポート番号>/<コンテキストパス>/sample_rag_assistant`

画面が表示され、アシスタントのチャット画面が表示されます。

> **コラム**
> 作成したロジックフローアシスタントの実行にはベクトルデータ構築が必要です。ベクトルデータ構築が完了していない場合は、アシスタントの動作確認ができません。
> 「*ベクトルデータ構築ジョブの作成*」で作成したジョブを実行の後、アシスタントの動作確認を行ってください。
> ジョブネット管理画面で「*ジョブネットの作成*」で作成したジョブネットを選択して「即時実行」を行ってください。

運用上の考慮事項
アシスタントの会話履歴削除ジョブの設定

アシスタントの会話履歴はユーザが画面から削除できますが、会話履歴の threadId の付与方法にランダム値を利用しているなど、会話履歴が画面表示時のみ参照可能な場合は、 画面を閉じると参照できなくなった会話履歴がデータベースに残り続けます。
会話履歴の増大によるデータベース容量の圧迫とクエリパフォーマンスの低下を防ぐため、定期的に不要な会話履歴をデータベースから削除することを推奨します。
IM-Copilot 機能で提供している「アシスタント会話履歴削除」ジョブを実行するジョブネットを作成し、定期スケジューリングすることを検討してください。

アシスタント会話履歴削除ジョブ情報

- 実行言語: Java
- 実行プログラム: jp.co.intra_mart.system.copilot.assistant.job.DeleteAssistantMessageJob
- 実行パラメータ:

| | |
|---|---|
| assistantId | アシスタントID<br>指定したアシスタントIDに紐づくアシスタント会話履歴情報を削除します。<br>アシスタント種別とアシスタントIDはどちらか一方のみを指定可能です。<br>対象のアシスタントが複数ある場合は、カンマ区切りで指定します。 |
| assistantKind | アシスタント種別<br>指定したアシスタント種別に紐づくアシスタント会話履歴情報を削除します。 |
| storagePeriod | 会話履歴保持日数<br>スレッドの最大履歴番号の作成日から「会話履歴保持日数」が経過しているスレッドの会話履歴が削除されます。 |

ジョブの詳細については、「ジョブ・ジョブネットリファレンス」-「アシスタント会話履歴削除」を参照してください。