ワークフローの定義

Version 2023.1
2024-01-02

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

ワークフローの定義
InterSystems IRIS Data Platform　Version 2023.1　2024-01-02

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

1

プロダクション内のワークフローの概要

この章では、InterSystems IRIS® 内のワークフロー機能の概要を説明します。

1.1 概要

ワークフロー管理システムによって、ユーザ間でのタスクの分配が自動化されます。事前に設定した方式に従ってタスクの分配を自動化することで、タスクの割り当てが効率化されると共に、タスク実行の責任がより明確になります。一般的な例としては、顧客から問題レポートを受け取り、適切な組織のメンバにそれらのレポートを転送して対応を求め、問題が解決されると、顧客に結果を報告するヘルプ・デスク・アプリケーションが挙げられます。

これ以降の項で説明するように、InterSystems IRIS ワークフロー・エンジンは、従来のスタンドアロンのワークフロー管理システムに比べ、きわめて高レベルの機能を備えています。

1.1.1 InterSystems IRIS への統合

InterSystems IRIS ワークフロー・エンジンでは、InterSystems IRIS アーキテクチャが最大限に活用されます。このため、Ensemble ワークフロー・エンジンは、企業のアプリケーション、テクノロジ、データ、および人間の参加者とシームレスに連携できます。このコラボレーションで最も重要な事項は、以下のとおりです。

- “ストレートスルー” ビジネス・プロセス (つまり、完全に自動化されたビジネス・プロセス) は、特に大量の注文の承認などの例外的なケースを扱うために人間の介入を組み込むことができます。
- InterSystems IRIS ワークフローでは企業アプリケーションを呼び出して、人手介入ワークフロー内のイベントを通知したり、人手介入ワークフローで必要とする追加情報を入手できます。
- 永続性:ワークフロー・エンジンは InterSystems IRIS 永続ストレージ機能を活用して、完了するまで数日または数週間かかる、長期間実行されるビジネス・プロセスをサポートします。

1.1.2 複合アプリケーションのサポート

InterSystems IRIS では、開発者が企業内のさまざまなシステムとテクノロジにまたがる複合アプリケーションを簡単に作成することができる、豊富な機能が備えられた開発環境を利用できます。ワークフロー・エンジンは InterSystems IRIS 内に完全に統合されているため、以下のことが実現します。

- 複合アプリケーションによって、地理的、技術的および部門の区分を越える複雑な情報交流のしくみを容易に組み込むことができます。
- ユーザベースのプロセス定義はビジネス・ロジックから切り離すことができ、開発者やアナリストは各セグメントを 1 つのまとまりの中に明確に定義できます。

- ワークフロー・システムは、機能および性能が増し、作成しやすく、メンテナンスが簡単になりました。

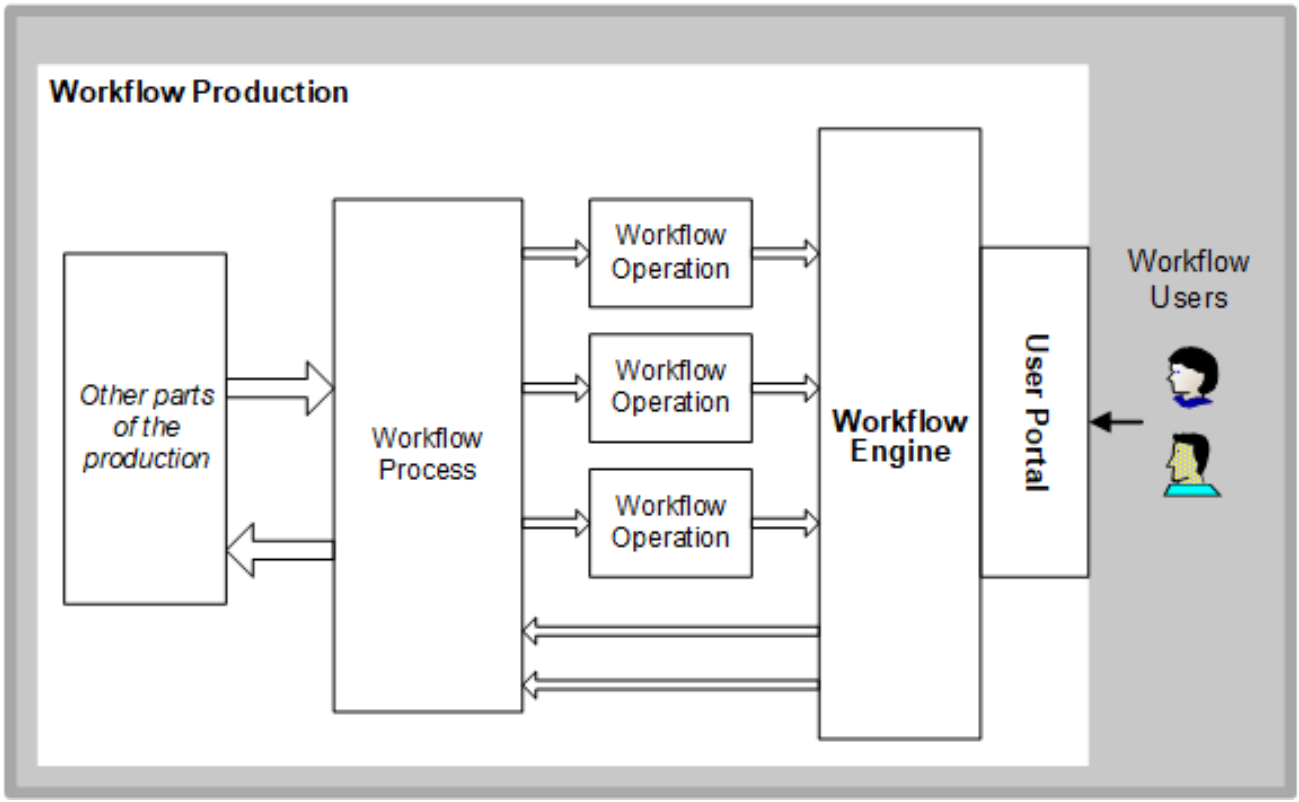
1.1.3 生産性機能

InterSystems IRIS ワークフロー・エンジンでは、以下のような生産性ツールを使用する自動統合が実現します。

- ビジネス・プロセス・デザイナ:ワークフロー・アナリストは、InterSystems IRIS のグラフィカル編集ツールであるビジネス・プロセス・デザイナを使用してワークフローを作成できます。ビジネス・プロセス・デザイナは、ビジネス・プロセス・ダイアグラムに基づいて本格的な実用コードを自動的に生成します。
- ビジネス・アクティビティ・モニタリング ： 開発者は企業向けダッシュボードおよびイベント・トリガを簡単に作成して、企業アナリストに現在のワークフローのステータスを提示できます。
- ビジュアル・トレース ： すべてのタスクはメッセージとして送信されるので、システム管理者は高機能なビジュアル・トレース・メッセージ・トレース・ツールを使用してタスクのステータスを表示できます。

1.2 プロダクション内のワークフロー・コンポーネント

以下の図は、プロダクション内のワークフロー・コンポーネントを示しています。



ワークフロー・プロセスは、タスク要求をワークフロー・オペレーションに送信する BPL プロセスです。このプロセスは、これらのワークフロー・オペレーション間で作業を調整する役割を果たします。このプロセスは、標準のビジネス・オペレーション（人的介入を必要としないもの）を呼び出すこともあります。ビジネス・プロセスを完了するには、両方のタイプの要素が必要な場合があります。

ワークフロー・オペレーションは、ワークフロー・ロールを表す特定用途のビジネス・オペレーションです（次の節で説明）。ワークフロー・オペレーションは、ワークフローの受信トレイを介して、関連付けられたロールに属するすべてのユーザにタスクを送信します。

ワークフロー・プロセスは、ビジネス・プロセスが他の種類の要求メッセージを他の種類のビジネス・オペレーションに送信するのとまったく同じ方法で、ワークフロー・オペレーションにタスク要求を送信します。

InterSystems IRIS の用語では、ワークフロー・タスクとは、進行中のビジネス・プロセスをサポートするためにオフラインで実行される処理の項目です。InterSystems IRIS は、タスク要求と呼ばれる特定用途のプロダクション・メッセージ・オブジェクトを使用してタスクを表します。タスク要求は、そのタスクの作業に関する情報を伝搬します。ワークフロー・プロセスは、タスク要求を送信して、タスク応答を受信します。他のすべてのメッセージと同様に、これらのメッセージは削除されるまでデータベースに保存されます（仮に削除される場合）。

1.3 ワークフロー・ロールとワークフロー・ユーザ

一般的な組織では、複数のユーザが特定の同一タスクを実行できることがあります。したがって、複数のユーザへのタスク分配を統制するために、InterSystems IRIS ではワークフロー・ロールが使用されます。ワークフロー・ロールは、特定の種類のタスクを実行できるユーザのリストです。

理論的には、1 つの企業では、ワークフロー・ロールをいくつでも定義できます。実際には、ロールは社内で、経理、財務、窓口、マネージャ、スーパーバイザ、役員などの部門や職位にマップします。

任意の数のワークフロー・ユーザをワークフロー・ロールのメンバにできます。ロールへのユーザの割り当ては、実際的な視点および組織的な視点から考えて、適切である必要があります。あるユーザにタスクの実行を許可する場合、そのユーザは対応するワークフロー・ロールのメンバである必要があります。

ワークフロー・ロールは、プロダクションごとではなく、相互運用対応ネームスペースごとに定義されます。つまり、同じワークフロー・ロール定義を、同じ相互運用対応ネームスペース内のすべてのプロダクションで使用できます。ワークフロー・ユーザが InterSystems IRIS ユーザ・アカウントに対応する場合は（通常はそうですが）、これと同じことがワークフロー・ユーザにも当てはまります。

1.4 ワークフローのユーザ・インタフェース

InterSystems IRIS では現在、ワークフローをサポートするための 2 つのユーザ・インタフェースが用意されています。これらのユーザ・インタフェースはそれぞれ異なるユーザ群を対象にしています。

1.4.1 実装担当者とスーパーバイザ

管理ポータルには、実装担当者やスーパーバイザがワークフローのロール、ユーザ、およびタスクを管理するために使用できるページが用意されています。これらのページにアクセスするには、[Interoperability] を選択して、[**管理**] をクリックし、[**ワークフロー**] をクリックします。

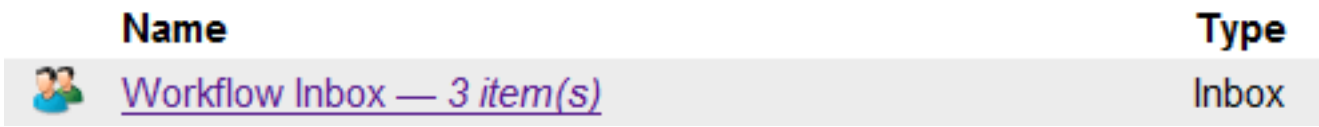
これらのページはエンド・ユーザ向けではありません。スーパーバイザはタスクを割り当てたり取り消したりできますが、他のアクション（タスクを完了済みとしてマークするなど）はここでは実行できません。

詳細は、"プロダクションの管理" の “ワークフロー・ロール、ワークフロー・ユーザ、およびワークフロー・タスクの管理” を参照してください。

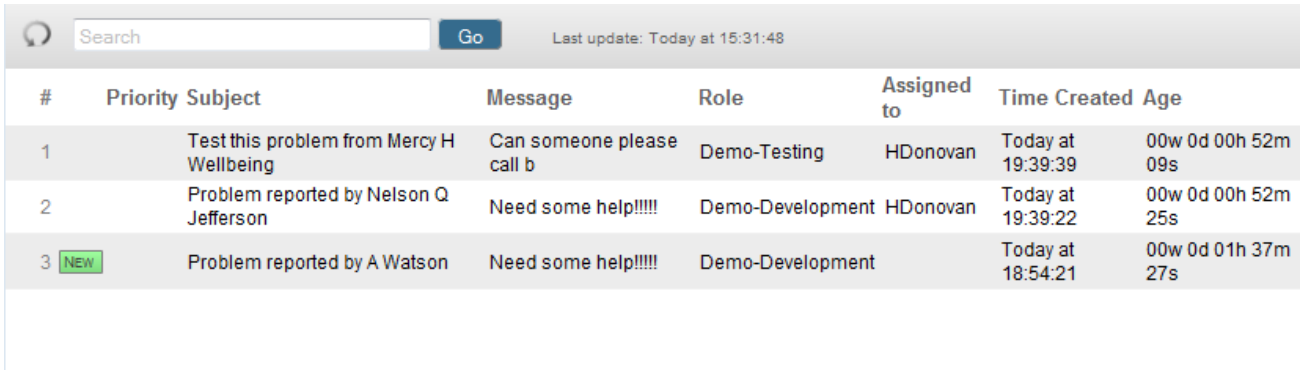
1.4.2 エンド・ユーザ

ユーザは InterSystems ユーザ・ポータル内で自身のワークフロー・タスクを管理し、このポータルには InterSystems IRIS ダッシュボード (および Analytics ダッシュボード) も表示されます。InterSystems ユーザ・ポータルには管理ポータルからアクセスできますが、多くの場合は、お使いのアプリケーションで InterSystems ユーザ・ポータルへの直接リンクが用意されています (ユーザ・ポータルにはタイトルはないため、すべてのユーザ向けの汎用として適しています)。

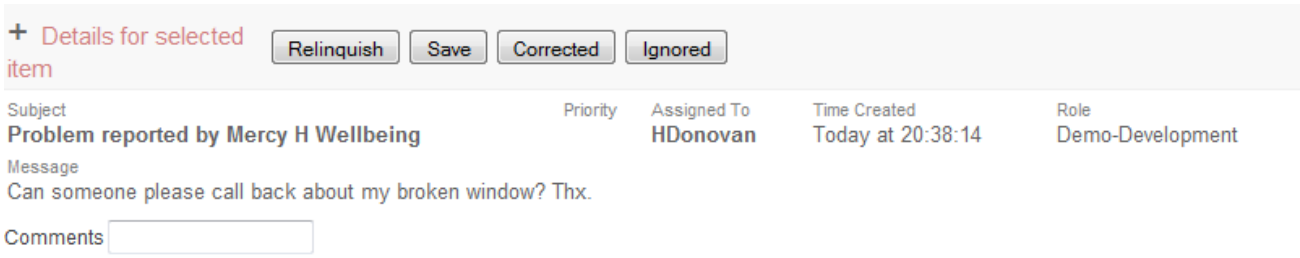
InterSystems IRIS ワークフロー・ユーザに対しては、ユーザ・ポータルのメイン領域にワークフローの受信トレイが表示されます。以下に例を示します。



ユーザが **[ワークフローの受信トレイ]** をクリックすると、ユーザ・ポータルには次のような内容が表示されます。



ユーザがタスクをクリックすると、対応するタスク・フォームが表示されます (このタスク・フォームを構成またはカスタマイズできます)。以下に例を示します。



詳細は、"ダッシュボードとユーザ・ポータルの使用法" の “ポータルの機能の使用法” を参照してください。

1.5 タスクのライフ・サイクル

プロダクション内のタスクのライフ・サイクルは以下のとおりです。

1. ワークフロー・プロセスは何らかの入力を受信してから、タスク要求 (メッセージ) を作成します。
2. ワークフロー・プロセスはこのタスク要求をワークフロー・オペレーションに送信します。
3. ワークフロー・オペレーションは、ワークフロー・エンジンへタスク要求を送信します。
4. ワークフロー・エンジンは、適切なタスク応答オブジェクトをインスタンス化します。このオブジェクトには、元のタスク要求からのフィールドが含まれており、同じユーザへのタスク分配を管理します。

5. タスク分配は、デフォルト方式またはカスタム分配方式のどちらかに従って行われます。詳細は、この章で後出する“タスク分配のオプション”を参照してください。
6. タスクが割り当てられて完了すると、ワークフロー・エンジンはビジネス・プロセスへタスク応答を返します。

1.6 タスク・フォームとカスタマイズ・オプション

InterSystems ユーザ・ポータル（前述の説明を参照）は、各ユーザに対して 1 つのワークフロー受信トレイを提供します。ユーザがタスクをクリックすると、そのタスクに固有のフォームが表示されます。このフォームは次のようにカスタマイズできます。

- このフォームはメタデータから生成されます。メタデータは、再利用可能なテンプレート・ファイル内に含まれているか、企業スタッフによって実行時に定義されます。
- このフォームは、企業の標準に適合するように簡単にスタイル設定できます。
- 企業がユーザに経験を積ませるために社内技術の活用を望む場合は、InterSystems IRIS のワークフロー・タスクを XML 文書、Web サービス、.NET、Java によって、またはリレーショナル構造として、さまざまな方法で社内の技術に対して使用できます。

1.7 タスク分配のオプション

ワークフロー・オペレーションがタスク要求を受け取ると、要求が InterSystems IRIS ワークフロー・エンジンに渡され、そのロール用に定義されたユーザの 1 人にタスクが分配されます。タスクを分配するために使用される方式は InterSystems IRIS のデフォルトで、以下のように機能します。

- 各ワークフロー・ユーザに対して 1 つのワークリストが存在し、このワークリストには、そのユーザに現在割り当てられているか関連付けられているすべてのタスクが一覧表示されます。特定のユーザに対するアクティブなタスクのみがワークリストに表示されます。タスクは、完了、破棄、取り消し、または別のユーザへの再割り当てのいずれかが行われた場合、リストから削除されます。
- ワークフロー・エンジンは、要求が送信されたワークフロー・ロールの各ユーザ用のワークリストで受け取るすべてのタスク要求を表示します。タスクが最初に表示される場合、ステータスは [未割り当て] です。このタスクは、これらのすべてのユーザと関連付けられていると表現されます。ステータスが [未割り当て] である間に、関連付けられた各ユーザは、自分のワークリストのタスクを確認します。
- ユーザはタスクの所有権をスーパーバイザによる割り当て、またはデフォルトによる割り当てから取得できますが、通常所有権を取得するにはタスクを受け入れる必要があります。ワークフロー・ロール内の任意のユーザは、ワークフローの受信トレイを使用して、FCFS（first-come, first-served ： 先着順）方式で自身のワークリスト上の任意のタスクを受け入れることができます。
- ユーザがタスクの所有権を取得すると、タスクはそのユーザに割り当てられたことになります。タスクのステータスは、[未割り当て] から [割り当て済み] に変わります。このタスクはそのタスクを受け入れたユーザのワークフロー受信トレイには残りますが、そのワークフロー・ロールの他のユーザのワークフロー受信トレイからは消えます。
- タスク要求には、特定のユーザを優先することを表すことができるプロパティがあります。そのユーザが非アクティブであるか、タスク要求が受信された時点で定義されていない場合、ワークフロー・エンジンはワークフロー・オペレーションにエラーを返し、タスクは実行されません。
- スーパーバイザは管理ポータルのワークフロー・ページからアクティブなタスクを再割り当てすることができます。この場合、タスクは以前割り当てられたユーザのワークリストから消えて、新しく割り当てられたユーザのワークリストに表示されます。

- 割り当てられたユーザは、タスクを完了せずに、そのタスクを放棄する場合があります。ユーザが未完了の状態のタスクを放棄すると、そのタスクは元の [未割り当て] のステータスに戻り、ワークフロー・エンジンはそのタスクを、それぞれの関連付けられたユーザ (ワークフロー・ロール内の各ユーザ) のワークフロー受信トレイに表示します。
- ワークフローの進行中にスーパーバイザがユーザ定義を削除するか、ユーザを非アクティブとしてマーキングすると、ユーザが突然自分のタスクをすべて放棄した場合と同じ結果になります。つまり、ユーザのワークリストにある各タスクは [未割り当て] ステータスになり、ワークフロー・エンジンはそのタスクが最初に送信された先であるワークフロー・ロールに属するすべてのユーザのワークフロー受信トレイに各タスクを表示します。
- スーパーバイザが、ワークリストの進行中にワークフロー・ロール定義を削除した場合、そのロールに対する未処理の要求は完了できますが、そのロールに対し、その後で要求を行うとエラーが発生します。ワークフロー・オペレーションの名前と (存在しない) ロール定義の名前の間に不一致が発生するためです。
- 各タスク要求は、1 つまたは複数のアクションを指定します。各アクションは、関連付けられたタスク・フォームで [承認] や [放棄] などのボタンとして表示される文字列です。割り当てられたユーザはこれらのボタンの 1 つをクリックして、タスクを完了させ、ビジネス・プロセスに応答を返信できます。
- 割り当てられたユーザがタスクに対するアクション・ボタンをクリックすると、ワークフロー・エンジンが、ユーザが選択したアクション名を含む、タスク応答オブジェクトのさまざまなプロパティを設定してワークフロー・オペレーションにこの応答を返し、ここから元の要求ビジネス・プロセスに応答が渡されます。ワークフロー・エンジンは、タスクを [完了] とマーキングして、すべてのワークリストからそのタスクを削除します。
- 完了後、すべてのタスクの完全なレコードはプロダクションのメッセージ・ウェアハウスに残り、管理ポータルから表示できます。ワークフローに直接関連するタスクの詳細は、Workflow Management Portal から表示できます。これは、メッセージが適切に完了せずに破棄されたか取り消された場合、またはエラーが発生した場合でも当てはまります。

タスクの分配に使用される方式は、タスク応答クラス **EnsLib.Workflow.TaskResponse** の OnNewTask() メソッドに埋め込まれます。必要に応じて、このクラスとそのメソッドのサブクラスを作成して、ロジックを変更できます。手順については、“ワークフローへのカスタム機能の組み込み”を参照してください。

1.7.1 カスタム・タスク分配

ワークフロー設計者は、ワークフロー定義内でさまざまなタスク分配方式を使用できます。InterSystems IRIS ワークフロー・エンジンには以下の方式が組み込まれています。

- FCFS (first come, first served)
- ジョブのタイトルによる割り当て
- 名前による割り当て
- ユーザ定義のランキングによる割り当て
- 現在のユーザの作業負荷による割り当て
- 他

また、カスタムのタスク分配方式を新規作成することもできます。

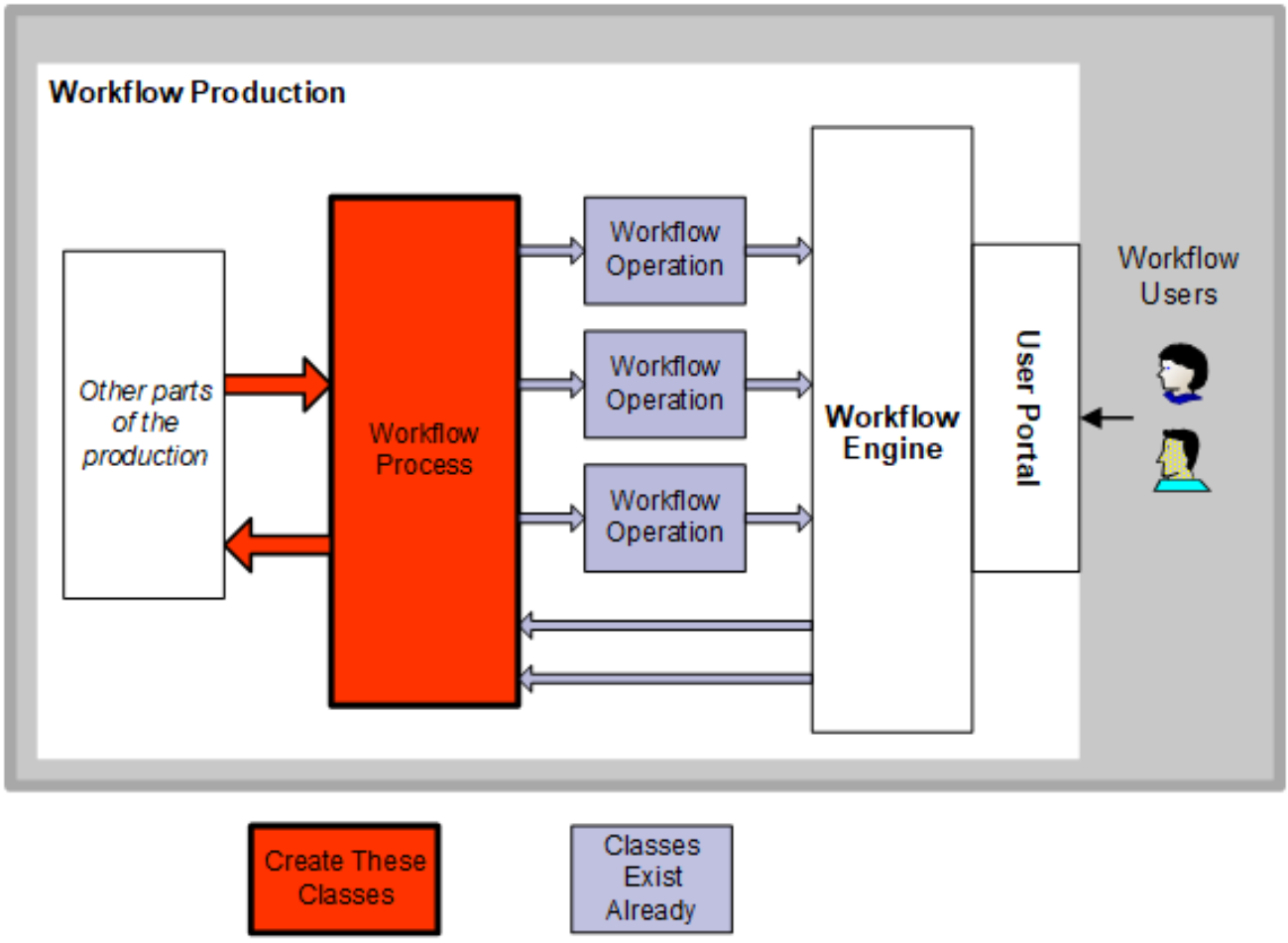
2
ワークフローの開発

この章では、ワークフローを開発してプロダクションに追加する方法を説明します。次の章では、開発時に実行できるオプションのカスタマイズについて詳述します。

付録“ワークフロー・サンプルの紹介”では、簡単なサンプルを紹介しています。

2.1 概要

以下の図は、プロダクション内のワークフロー要素を示しています。



これらのコンポーネントは以下のとおりです (左から右の順)。

- メッセージ要求クラスは、プロダクション内の別の場所にある発信元からワークフローの具体的な詳細情報を伝送します。これは、各自が定義できる標準のメッセージ要求クラスです。つまり、Ens.Request のサブクラスまたは別の永続クラスのサブクラス (前者の場合の方が多い) です。
- メッセージ応答クラスは、当該タスクが完了するか他の形で処理された後に、ワークフローの具体的な詳細情報を発信元に返送します。これは、各自が定義できる標準のメッセージ応答クラスです。つまり、Ens.Response のサブクラスまたは別の永続クラスのサブクラスです (前者の場合の方が多いです)。

  このドキュメントでは、このクラスや対応する要求クラスの定義に関する特別な詳細情報は提供していません。これらの実装は各自のニーズに全面的に依存しているからです。
- 通信の調整役として機能するワークフロー・プロセス。ワークフロー・プロセスを作成するには、以下の操作を行います。

  1. ビジネス・プロセス・デザイナを使用して、Ens.BusinessProcess.BPL を基本クラスとするビジネス・プロセスを作成します。このビジネス・プロセスには、受信メッセージを受信するためのロジック、および必要に応じて同じプロダクション内のビジネス・オペレーションを (非同期的に) 呼び出すためのロジックが含まれている必要があります。
  2. このビジネス・プロセスをプロダクションに通常どおりに追加します。

- 特別なメッセージ・クラスである EnsLib.Workflow.TaskRequest は、ビジネス・プロセスから要求を伝送します。

  同様に、特別なメッセージ・クラスである EnsLib.Workflow.TaskResponse は、ワークフロー・エンジンから応答を伝送します。

  サブクラスを作成して使用することもできますが、これらのクラスは柔軟な方法で情報を伝送できるように設計されているため、通常はその必要はありません。
- 各ワークフロー・オペレーションはタスク要求を受信して、ワークフロー・エンジンと通信し、ワークフロー・エンジンはタスクを特定のユーザに表示します。特に、各ワークフロー・オペレーションにはワークフロー・ロールが 1 つだけ関連付けられています。ワークフロー・オペレーションは、そのワークフロー・ロールに属するすべてのユーザにタスクを表示します (他のユーザにはこれらのタスクは表示されません)。

  これらのタスクのそれぞれを作成するには、EnsLib.Workflow.Operation を基本クラスとするビジネス・オペレーションを追加します。構成はほとんど必要ありません。

  または、EnsLib.Workflow.Operation のサブクラスを作成して、ワークフロー・エンジンと独自の方法で相互動作するためのそのコールバック・メソッドを実装することもできます。

2.2 ビジネス・プロセスの設計

最初のステップは、ビジネス・プロセス・クラスを設計して、いくつかの重要な決定を下すことです。

1. ワークフローへの呼び出し (人間のプロセス) が発生する箇所を特定します。
2. これらの各呼び出しはタスクです。各タスクに名前をつけます。
3. タスクが送信される各ワークフロー・ロールに名前をつけます。
4. 各タスクのタスク分配方式を決定します (通常はデフォルト方式)。
5. ビジネス・プロセスを呼び出すために必要な任意のメッセージ・タイプを作成します。

   このドキュメントでは、これらのメッセージの定義に関する特別な詳細情報は提供していません。これらの実装は各自のニーズに全面的に依存しているからです。

6. 次の節の説明に従ってビジネス・プロセス・クラスを作成します。

2.3 ワークフロー・プロセスの作成

ワークフロー・プロセスを作成するには、"BPL プロセスの開発" の説明に従って、ビジネス・プロセス・デザイナを使用します。

ビジネス・プロセスのタスク別に、以下を実行します。

- ワークフロー・オペレーションを非同期で呼び出すために <call> を指定します。
- <call> で、宛先ワークフロー・オペレーションを名前で指定します。各ワークフロー・オペレーションは 1 つのみのワークフロー・ロールと関連付けられており、そのワークフロー・ロールは構成可能な 1 つのユーザ・セットに対応しています。
- <call> で、適切なタスク要求オブジェクト (EnsLib.Workflow.TaskRequest またはサブクラス) を使用し、そのオブジェクトのプロパティを設定します。

  <call> では、タスク応答オブジェクト (EnsLib.Workflow.TaskResponse またはサブクラス) を検査し、このオブジェクトを使用して BPL の context 変数を設定することも必要になります (この変数は後の処理で使用されます)。
- <sync> を設定して、非同期 <call> の結果をキャッチします。

例については、付録 “ワークフロー・サンプルの紹介” を参照してください。

2.3.1 タスク要求の定義

プロセス内の各 <call> では、そのメッセージ・クラスとして EnsLib.Workflow.TaskRequest またはサブクラスを使用する必要があります。要求を定義するには、<call> に callrequest 変数のプロパティを設定する必要があります。

以下のテーブルは、EnsLib.Workflow.TaskRequest の使用可能なプロパティを一覧表示しています。これらのプロパティはすべてオプションですが、当然ながら、ビジネス・プロセスはワークフロー・エンジンが必要とするすべての情報を送信する必要があります。

プロパティ	目的
%Actions	オプション。このタスク用に定義されているアクションのカンマ区切りリスト。例えば、"Approve,Reject,NeedMoreInfo"。これらによって、タスクを確認するときに表示されるアクション・ボタンが決まります。選択したユーザ・アクションは、タスク応答の一部として返されます。
%Command	オプション。ワークフロー・エンジンに引き渡されるコマンド文字列。これを使用すると、データ駆動型のカスタム動作をタスクに追加できます。
%FormTemplate	オプション。このタスクの HTML フォーム・テンプレートを表示する InterSystems IRIS® Web ページの名前。
%FormFields	オプション。このタスクに関連付けられているフォームに表示されるフィールドのカンマ区切りリスト。
%FormValues	オプション。このタスクに関して表示されるフォーム内に表示する値の集まり。
%Message	オプション。このタスクの詳細なメッセージ本文。これは、ユーザがタスクを確認したときに表示されます。
%Priority	オプション。要求したタスクの優先順位 (1 が最高)。この値は、ユーザのワークリスト内にある項目の並べ替えに使用します。

プロパティ	目的
%Subject	オプション。このタスクの簡単な要約。これは、[ユーザ] ワークリストに表示されます。
%TaskHandler	オプション。このタスクの分配を管理するために使用する応答クラスの名前。この要求の応答タイプとしても使用されます。%TaskHandler で指定するクラスは、EnsLib.Workflow.TaskResponse のサブクラスである必要があります。
%Title	オプション。 指定した中で、このタスクの処理に適したタイトルの名前。このユーザがタスクに実際に割り当てられているかどうかは、このタスクに関して分配方式をどのように使用するかによって異なります。
%UserName	オプション。このタスクの処理に適したユーザの名前。このユーザがタスクに実際に割り当てられているかどうかは、このタスクに関して分配方式をどのように使用するかによって異なります。

2.3.2 タスク応答の使用

前述のとおり、指定されたタスクの <cal> では、タスク応答オブジェクト (EnsLib.Workflow.TaskResponse またはサブクラス) を調査し、このオブジェクトを使用して BPL の context 変数を設定することも必要になります (この変数は後の処理で使用されます)。EnsLib.Workflow.TaskResponse は、以下のプロパティを定義します。

プロパティ	目的
%Action	オプション。タスクが完了すると、タスク完了のためにユーザが選択したアクションの値が、このプロパティに含まれます。"%Actions" を参照してください。
%Actions	オプション。このタスク用に定義されているアクションのカンマ区切りリスト。例えば、"Approve,Reject,NeedMoreInfo"。これらによって、タスクを確認するときに表示されるアクション・ボタンが決まります。選択したユーザ・アクションは、タスク応答の %Action プロパティに含めて返されます。
%FormFields	オプション。このタスクに関連付けられているフォームに表示されるフィールドのカンマ区切りリスト。これは、最初のタスク要求で指定された値のコピーです。
%FormTemplate	オプション。このタスクのフォーム・テンプレートを表示する CSP ページの名前。これは、最初のタスク要求で指定された値のコピーです。
%FormValues	オプション。このタスクに関連付けられているフォームの値の集まり (存在する場合)。
%Message	オプション。 このタスクの詳細なメッセージ本文。これは、最初のタスク要求で指定された値のコピーです。
%Priority	オプション。 要求したタスクの優先順位 (1 が最高)。これは、最初のタスク要求で指定された値のコピーです。
%RoleName	オプション。このタスクを処理した役割の名前。これは、最初のタスク要求で指定された値のコピーです。
%Status	オプション。このタスクの外部ステータス。タスクの現在のステータスのクエリに使用されます。
%Subject	オプション。このタスクの簡単な要約。これは、最初のタスク要求で指定された値のコピーです。
%TaskStatus	オプション。このタスクの内部ステータス ([割り当て済み]、[未割り当て]、[取り消し]、[破棄]、[削除])。ワークフロー・エンジンでタスクの管理に使用されます。カスタム・コードでは、このプロパティの値を変更できません。
%UserName	オプション。このタスクを最後に処理したユーザの名前。この値は、タスク完了時に設定されます。
%UserRanking	オプション。 このタスクを最後に処理したユーザに関連付けられているランキング (該当するユーザに役割が割り当てられたランキングがある場合)。

プロパティ	目的
%UserTitle	オプション。このタスクを最後に処理したユーザに関連付けられているタイトル (該当するユーザに役割が割り当てられたタイトルがある場合)。

2.3.3 プロダクションへのワークフロー・プロセスの追加

ワークフロー・プロセスをプロダクションに追加するには、以下の操作を行います。

1. そのプロダクションを管理ポータルの [Interoperability]→[**構成する**]→[**プロダクション**] ページで表示します。
2. [**プロセス**] 列で、[追加] ボタン (プラス記号) を選択します。

   ダイアログ・ボックスが表示されます。
3. [**ビジネス・プロセス・クラス**] で、ビジネス・プロセス・エディタで作成したクラスを選択します。
4. [**プロセス名**] に、適切な名前を入力します。
5. [OK] をクリックします。

その他の設定の詳細は、"プロダクションの構成" を参照してください。

2.4 プロダクションへのワークフロー・オペレーションの追加

ワークフロー・オペレーションをプロダクションに追加するには、以下の操作を行います。

1. そのプロダクションを管理ポータルの [Interoperability]→[**構成する**]→[**プロダクション**] ページで表示します。
2. [**オペレーション**] 列で、[追加] ボタン (プラス記号) を選択します。

   ダイアログ・ボックスが表示されます。
3. [**ワークフロー**] タブを選択します。
4. [**クラス名**] リストから EnsLib.Workflow.Operation クラスを選択します。

   または、カスタム・サブクラスを選択します。
5. [**オペレーション名**] に、このビジネス・オペレーションの名前を入力します。この名前は、BPL ビジネス・プロセスの対応する <call> に含まれる target 属性と一致させる必要があります。
6. 必要に応じて、[**自動作成ロール**] を選択します。このオプションを有効にすると、[**オペレーション名**] で指定されたのと同じ名前のワークフロー・ロールが自動的に作成されます。

   希望に応じて、このオプションを後で有効にすることもできます。
7. [OK] を選択します。

その他の設定の詳細は、"プロダクションの構成" を参照してください。

2.5 次のステップ

- ワークフロー・ユーザを適切なワークフロー・ロールと関連付けます。"プロダクションの管理" の “ワークフロー・ロール、ワークフロー・ユーザ、およびワークフロー・タスクの管理” を参照してください。

- “ワークフローのテスト”の章の説明に従って、ワークフローをテストします。
- 必要に応じて、スーパーバイザがワークフローの進捗状況を監視できるようにするためのダッシュボードを追加します。InterSystems IRIS は、特定のワークフロー・メトリックを自動的に収集します。一般的な開発タスクと構成タスクは通常どおりです。

  1. 1 つ以上のビジネス・メトリック・クラスを作成します。″プロダクションの開発″ の“ビジネス・メトリックの定義”を参照してください。

     各クラスは、ワークフロー・ロールに関する統計情報を報告するメソッドを提供する EnsLib.Workflow.Engine クラスのメソッドを呼び出すことができます。

     クイック・リファレンスとして、これらのメソッドは付録“使用可能なワークフロー・メトリック”に列挙されています。
  2. プロダクションを構成するときに、各ビジネス・メトリックをビジネス・サービスとして追加します。
  3. ダッシュボードを定義します。″プロダクションの構成″ の“ダッシュボードの定義”を参照してください。

3

ワークフローへのカスタム機能の組み込み

この章では、ワークフローにカスタム機能を組み込む方法を説明します。

3.1 標準タスク・フォームの拡張

各タスクには、少なくとも以下の標準アイテム・セットを表示する 1 つのタスク・フォームが関連付けられています。

- そのタスクの TaskId、Owner、%Subject、および %Message という読み取り専用プロパティ
- [承認] ボタン (そのタスクが未割り当ての場合)
- [放棄] ボタンと [保存] ボタン (そのタスクが割り当て済みの場合)

他のボタンを追加するには (これらのボタンはそのタスクが割り当て済みの場合にのみ表示されます)、タスク要求の %Actions プロパティで、追加するボタン名のコンマ区切りリストを指定します。

他の編集可能な値を追加するには (これらの値はそのタスクが割り当て済みの場合にのみ表示されます)、タスク要求オブジェクトで次のいずれかのプロパティを設定します。

- %FormFields – 表示するタスク・フォームのフィールドのリストを提供します。

  タスク要求オブジェクトおよびタスク応答オブジェクトには、%FormFields プロパティがあります。タスク要求を呼び出すときに、%FormFields の値を、タスク・フォームに表示するフィールドのカンマ区切りリストに設定します。現時点では、これらはすべて単純な文字列値と見なされます (制御を強化する必要がある場合は、フォーム・テンプレートを使用)。

- %FormTemplate – 表示する HTML フォーム・テンプレートを定義するファイルを指定します。

  タスク要求オブジェクトおよびタスク応答オブジェクトには、%FormTemplate プロパティがあります。タスク要求を呼び出すときに、%FormTemplate の値を、表示するフォームを定義する CSP ファイルの名前に設定します。この CSP ページでは、フォームとコンテンツのみを定義します。HTML、HEAD、BODY の各セクションはありません。CSP ページでは、特定の JavaScript イベント・メソッドをオーバーロードできます。このフォームを処理するとき、変数 %task は現在のタスク応答オブジェクトです。

%FormFields または %FormTemplate を使用する場合は、各フィールドに表示されるデフォルト値を指定できます。そのためには、タスク要求オブジェクトの %FormValues プロパティを指定します。このプロパティは、フィールド名が添え字として使用された、文字列の配列です。このプロパティに値を追加するには、array インタフェースを使用します。

ワークフロー・エンジンでは、タスクを処理するときに、配列コレクション %FormValues を含むフォーム関連のすべてのプロパティが、タスク要求からタスク応答へコピーされます。フォーム内のフィールドの値が更新されるたびに、ワークフロー・エンジンは、タスク応答オブジェクトの %FormValues コレクションで値を更新します。

フォームには常に、%FormValues コレクションの最新の値が表示されます。これにより、特定タスク内でフォームを動的に処理できます。フォーム関連のプロパティの値を変更することにより、タスク応答コールバック・メソッドを、フォーム処理の実行方法に反映させることができます。タスク応答オブジェクトの %FormValues コレクションは、最終的なタスク応答が送り返されたときに、元の呼び出し側で使用できます。

3.2 カスタム・タスク・フォームの使用

ワークフロー・タスク用に、カスタム・フォームをユーザに対して表示できます。手順は以下のとおりです。

1. カスタマイズされたフォームのコンテンツを定義する HTML テンプレート・ファイルを作成します。
2. タスク要求オブジェクトの %FormTemplate プロパティを、この CSP ページの名前に設定します。例えば、タスク要求をワークフロー・オペレーションに送信する <call> 文には以下のような内容が含まれます。

```
<assign property='callrequest.%FormTemplate'
        value='"MyForm.csp"'
        action='set' />
```

3. タスク要求オブジェクトの %FormFields プロパティを、フィールド名のカンマ区切りリストに設定します。これは、フォーム用に定義されるフィールドのリストです。以下に例を示します。

```
<assign property='callrequest.%FormFields'
        value='"Details,CustomerName"'
        action='set' />
```

4. フォーム・フィールドに初期値を指定する場合は、タスク要求オブジェクトの %FormValues コレクション・プロパティの対応する要素 (配列キーがフィールドの名前) を、必要な値に設定します。以下に例を示します。

```
<assign property='callrequest.%FormValues'
        value='request.CustomerName'
        action='set'
        key='"CustomerName"' />
```

前述のとおり、ワークフロー・ユーザが自身が現在所有しているタスクをレビューするたびに、関連付けられたタスク・フォームが自動的に表示されます。デフォルトでは、このフォームは、タスク要求オブジェクトの %FormFields プロパティの値で定義されるフィールドを使用して、自動的に生成されます。呼び出し側のビジネス・プロセスが、これらの値を指定します。ただし、この生成されるフォームの代わりに、カスタム設計されたテンプレートを使用する方法もあります。

ワークフロー・フォーム・テンプレートでは、標準タスク・フォームに挿入される HTML のブロックが定義されます。この HTML ブロックに含めることのできるフォーム・フィールドの数に制限はありません。これらのフィールドは、ワークフロー・タスクでユーザがアクションを実行するたびに、ワークフロー・エンジンによって自動的に送信され、処理されます。テンプレート・ファイルに含まれる HTML は、完全な HTML ドキュメントではありません。フォームのカスタム部分を表示するために必要な HTML です。 具体的には、タスク・フォームによって生成される HTML は以下のようになります。

```
<html>
  <body>
    <form>
      —TEMPLATE CONTENTS INJECTED HERE—
    </form>
  </body>
</html>
```

以下のテンプレートでは、テキスト・ボックスと選択 (コンボボックス) の 2 つの HTML 入力コントロールを表示する HTML テーブルを定義します。コントロールの名前は、%FormFields プロパティで定義されているフィールドに対応します。この

例では、サーバ側の式も使用して、Details プロパティの初期値を取得します。%task 変数は、常に現在のタスク応答オブジェクトに設定されています。

```
<!– workflow template –>
<table>
  <tr>
    <td>Details:</td>
    <td>
      <input type="text"
             name="Details"
             value="#(%task.%FormValues.GetAt("Details"))#">
    </td>
  </tr>
  <tr>
    <td>Company:</td>
    <td>
      <select name="CustomerName">
        <option value="ABC Corp">ABC Corp</option>
        <option value="XYZ Corp">XYZ Corp</option>
      </select>
    </td>
  </tr>
</table>
```

HTML の他に、ワークフロー・テンプレート・ファイルには、以下の Javascript コールバック関数をオプションで含めることができます。

- onLoad() ー ワークフロー・フォームがブラウザにロードされたときに呼び出されます。
- onAction() ー フォームで表示されたアクション・ボタンのいずれかをユーザがクリックしたときに呼び出されます。

これらのコールバックをワークフロー・テンプレート・ファイルに追加するには、テンプレート・ファイルの <script> タグの中に関数定義を含めます。例えば、以下のようにします。

```
<script language="JavaScript">
        function onLoad(form)
        {
                // form is the workflow form object
                return true;
        }

        function onAction(form,action)
        {
                // form is the workflow form object
                // action is a string containing the user's action
                // returning false will cancel this action
                return true;
        }
        </script>
```

3.3 タスク分配方式のカスタマイズ

タスク分配方式は、タスク応答クラスで指定されます。カスタム・タスク分配方式を実装するには、以下の操作を行います。

- EnsLib.Workflow.TaskResponse のサブクラスを作成し、その OnNewTask() コールバック・メソッド (および場合によっては他のメソッド) をオーバーライドします。最初の項では、これのオプションについて詳述します。
- カスタム・タスク応答クラスを必ず呼び出してください。これについては、2 つ目の項で説明します。

3.3.1 カスタム・タスク応答クラスの作成

EnsLib.Workflow.TaskResponse のサブクラスを作成する際には、その OnNewTask() コールバック・メソッド (および場合によっては他のメソッド) をオーバーライドします。以下では、オーバーライドできるコールバック・メソッドを列挙しています。

- OnAction() ー ワークリスト・フォームからユーザがアクションを選択したときに呼び出されます。通常、これはタスク終了のマーキングとなります。
- OnAssign() ー ユーザがタスクに関連付けられている所有権を要求したときに呼び出されます。通常、このメソッドでは割り当てが実行されます。
- OnCancel() ー タスクが取り消されたとき、例えばタイム・アウトしたときなどに呼び出されます。
- OnFormSubmit() ー このタスクに関連付けられているタスク・フォームが送信されたときに呼び出されます。
- OnNewTask() ー ワークフロー・エンジンで新しいタスクを受信したときに呼び出されます。通常、このメソッドでは、タスクが現在の役割のメンバに関連付けられます。
- OnRelinquish() ー ユーザがタスクに関連付けられている所有権の放棄を要求したときに呼び出されます。通常、このメソッドでは、タスクの割り当てが解除され、役割に含まれる他のユーザにタスクが戻されます。
- OnRoleChange() ー このタスクに関連付けられているユーザまたはロールの定義が変更されたとき、例えばロールに含まれるユーザのリストが変更されたときなどに呼び出されます。

EnsLib.Workflow.TaskResponse コールバック・メソッドでは、ワークフロー・エンジン (EnsLib.Workflow.Engine) で定義されている API メソッドの数を呼び出すことにより、タスクの分配が制御されます。

以下では、上記のコールバックをオーバーライドする際に使用できる EnsLib.Workflow.Engine 内のクラス・メソッドを列挙しています。

1. AssignTask() ー 特定のユーザにタスクを割り当てます。
2. CompleteTask() ー タスクを [Completed] としてマーキングし、呼び出し側に応答を返します。
3. FindLeastBusyUser() ー “最もビジーでない” ユーザの名前を返します。これは、システム内で割り当てられているタスクが最も少ないユーザです。
4. SendTask() ー 特定のユーザにタスクを送信します。
5. SendTaskToAll() ー 現在のロールに含まれる全ユーザにタスクを送信します。
6. SendTaskToTitle() ー 現在のロールの中で、指定したタイトルを持つ 1 人または複数のユーザにタスクを送信します。
7. SendTaskToTop() ー 現在のロールに含まれる上位 n ユーザに、ロールの中での各ユーザのランキングに従ってタスクを送信します。
8. UnassignTask() ー タスクの割り当てを削除します。

これらのメソッドの詳細は、″Intersystems クラス・リファレンス″ を参照してください。

3.3.2 カスタム・タスク応答クラスの呼び出し

タスク要求メッセージには、使用する応答クラスを指定するクラス・パラメータ (RESPONSECLASSNAME) があります。このクラス・パラメータをオーバーライドするには、要求クラスの %TaskHandler プロパティを設定します。

すなわち、タスク要求で目的のタスク応答が使用されるようにするには、次の 2 つの方法があります。

- 必要に応じて、EnsLib.Workflow.TaskRequest のサブクラスを作成して、その RESPONSECLASSNAME パラメータをオーバーライドし、その値をカスタム・タスク応答クラスの名前に一致させます。以下に例を示します。

```
Class MyApp.MyWorkflowRequest Extends EnsLib.Workflow.TaskRequest
{

Parameter RESPONSECLASSNAME = "MyApp.MyWorkflowResponse";

}
```

次に、ワークフロー・プロセスの適切な部分でこのメッセージ・クラスを使用します。

- ワークフロー・プロセス内で、ワークフロー・オペレーションを呼び出す際に、要求インスタンスの %TaskHandler プロパティの値を目的のタスク応答クラス名に設定します。以下に例を示します。

```
set callrequest.%TaskHandler="MyApp.MyWorkflowResponse"
```

4

ワークフローのテスト

この章では、ワークフローのテストに関する項目について説明します。

4.1 テストのチェックリスト

ワークフローをテストする際は、少なくとも以下の項目をテストしてください。

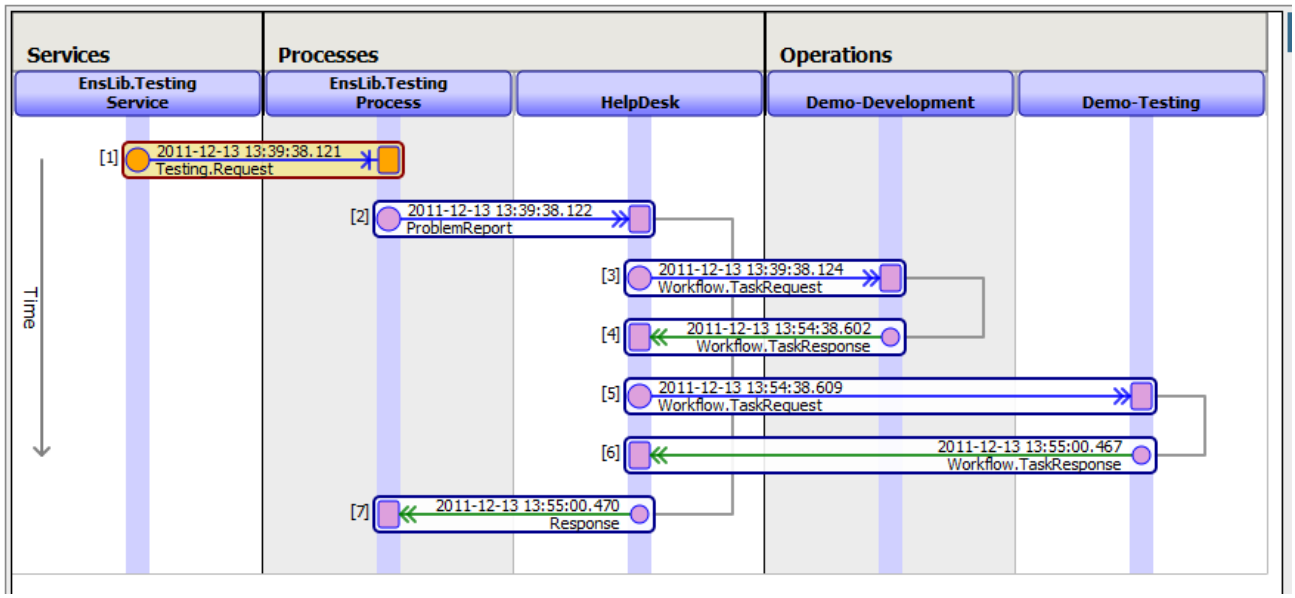
- タスク分配をテストします。
- タスクの取り消しをテストします。
- タスクのエスカレーションをテストします。
- 例外的なケースをテストします。
- タイムアウト動作をテストします。
- ダッシュボードを表示します (存在する場合)。

"プロダクションの管理" の "ワークフロー・ロール、ワークフロー・ユーザ、およびワークフロー・タスクの管理" の説明に従って、管理ポータル経由でアクティビティをモニタできます。次の節で説明するように、ビジュアル・トレースを使用することもできます。

4.2 ビジュアル・トレースでのワークフロー・アクティビティの表示

InterSystems IRIS® がメッセージおよびビジネス・オペレーションに対してサポートしているすべての統計機能、メンテナンス機能、および表示機能は、それぞれワークフロー・タスクおよびワークフロー・ロールにも適用されます。これらの機能には、メッセージ・ビューワ、メッセージのパージ、およびビジュアル・トレースが含まれます。詳細は、"プロダクションの管理" を参照してください。

特に留意すべきこととして、他のタイプのメッセージの場合と同様に、ビジュアル・トレースを使用してワークフロー・タスクの要求と応答をトレースできます。以下に例を示します。



この例で示しているセッションでは、タスク要求が Demo-Development ロールに送信されて、このロール内のユーザによって処理されました。次にワークフロー・プロセス (HelpDesk) が別のタスク要求を Demo-Testing ロールに送信しました。このロール内のユーザがこの要求を処理して、ワークフロー・プロセスは EnsLib.Testing.Process プロセスに応答を送信しました。

この例については、付録 “ワークフロー・サンプルの紹介” を参照してください。

A
ワークフロー・サンプルの紹介

この付録ではワークフロー・サンプルを紹介します。

A.1 サンプルの概要

Demo.Workflow サンプルには、Demo.Workflow.HelpDesk というビジネス・プロセスが含まれています。このビジネス・プロセスは、問題レポートに必要な基本情報 (送信者名と問題の説明) が含まれた受信要求を受け取ります。アクションを求めるこの要求を受け取ると、HelpDesk は、いずれかのアクションによって問題が解決されるまで続けられる一連のアクションを開始します。

HelpDesk は、最初に問題レポートをタスクとして開発グループに送信します。このグループのメンバが問題が解決したことをレポートすると、HelpDesk は問題レポートにメモを追加し、それをタスクとしてテスト・グループに送信します。このグループのメンバが修正が十分であるとレポートすると、HelpDesk はレポートした問題が解決したことを示すフラグを付けて、発信者に最終的な応答を返信します。

A.2 セットアップ・タスク

このサンプルをセットアップして使用するには、以下の操作を行います。

1. "ユーザ・アカウント" の説明に従って、ユーザをいくつか作成します。
2. "プロダクションの管理" の "ワークフロー・ロール、ワークフロー・ユーザ、およびワークフロー・タスクの管理" の説明に従って、これらのユーザのそれぞれをワークフロー・ユーザとして構成します。
3. このプロダクションを開始すると、InterSystems IRIS® によって Demo-Development と Demo-Testing という 2 つのロールが自動的に作成されます(これらのロールが作成されるのは、このプロダクション内の 2 つのワークフロー・オペレーションに対して **[自動作成ロール]** 設定が有効になっているためです)。
4. "プロダクションの管理" の "ワークフロー・ロール、ワークフロー・ユーザ、およびワークフロー・タスクの管理" の説明に従って、ワークフロー・ユーザをこれらのワークフロー・ロールに割り当てます。

   一部のユーザは両方のロールに割り当てることができます。
5. [テスト・サービス] ページを使用して、以下の操作を行っていくつかのメッセージを HelpDesk に送信します。
   a. **[Interoperability]→[テスト]→[ビジネス・ホスト]** を選択します。
   b. **[ターゲットタイプ]** で、**[ビジネス・プロセス]** をクリックします。

c. [ターゲット名] で、HelpDesk をクリックします。

d. [テスト] をクリックします。

e. [報告者] フィールドと [問題] フィールドに適切な値を入力します。

f. [テスト・サービスの呼び出し] をクリックします。

いくつかのメッセージを送信して、このシナリオを対象にします。

6. それぞれのユーザとして InterSystems ユーザ・ポータルにログインし、ワークフローの受信トレイを使用してタスクを管理します。

まず Demo-Development ロールのユーザとしてログインして、いくつかのタスクを完了済みとしてマークします。これにより、追加のテスト・タスクが生成されます。次に Demo-Testing ロールのユーザとしてログインして、これらのテスト・タスクを完了済みとしてマークします。

Tipヒント 必要に応じてネームスペースのデフォルトの Web アプリケーションを構成して、このアプリケーションがパスワード認証を受け付ける一方で、認証なしのアクセスを受け付けないようにしてください。詳細は、"アプリケーション" を参照してください。

A.3 サンプル・リクエスト・メッセージ

以下の要求メッセージ・クラスは、サンプルのビジネス・プロセス HelpDesk に入る要求メッセージを定義します。このメッセージは顧客から送られた技術的な問題を表します。要求メッセージ・クラスの定義は以下のようになります。

Class Definition

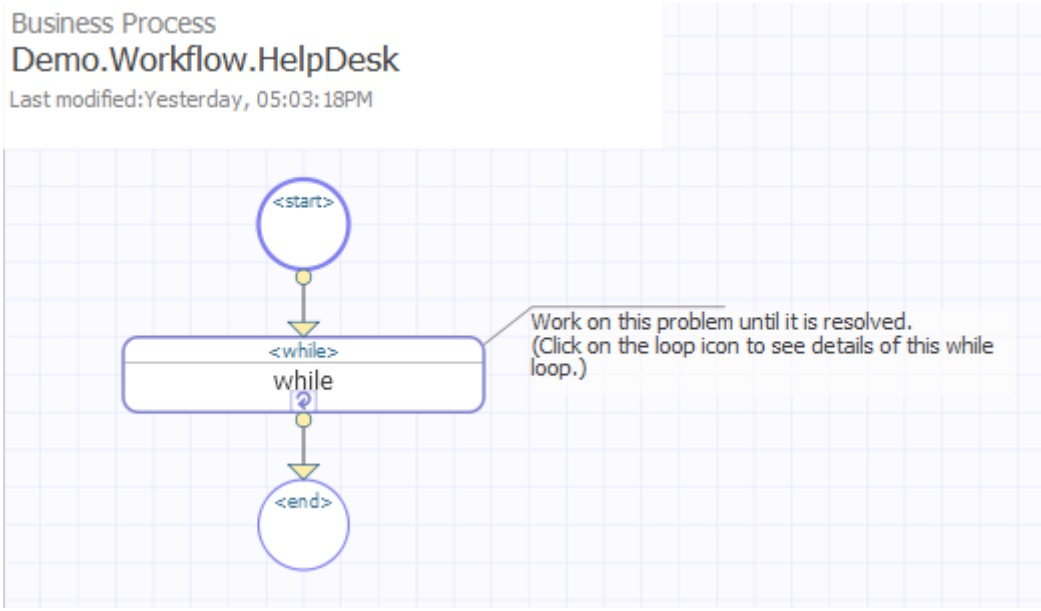
```
Class Demo.Workflow.ProblemReport
{
/// Name of customer reporting the problem.
Property ReportedBy As %String(MAXLEN = 60);

/// Description of the problem.
Property Problem As %String(MAXLEN = 200);
}
```

A.4 ビジネス・プロセス・クラスのサンプル

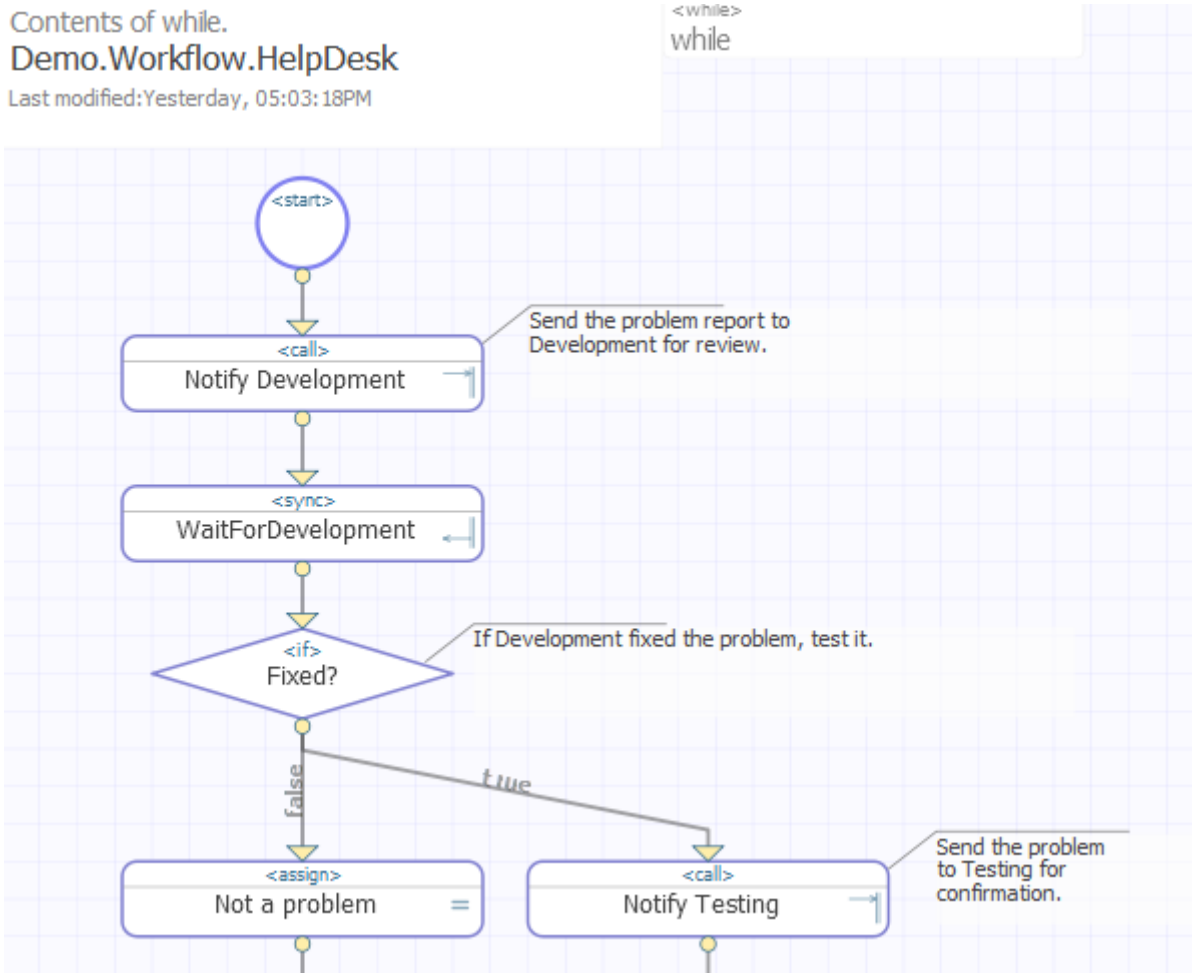
Demo.Workflow.ProblemReport 要求メッセージを受け取ると、BPL ビジネス・プロセスのサンプルの Demo.Workflow.HelpDesk は、いずれかのアクションによって問題が解決されるまで続けられる一連のアクションを開始します。

HelpDesk ビジネス・プロセスは while ループで構成されます。問題レポートはこのループ内で、まず、問題を修正するために開発(グループへ、次に修正をテストするためにテスト・グループへ送信されます。この一連のアクションが完了するとすぐ、while 条件 “Resolved” が満たされ、ビジネス・プロセスが終了し、最終的な応答が呼び出し側に返されます。

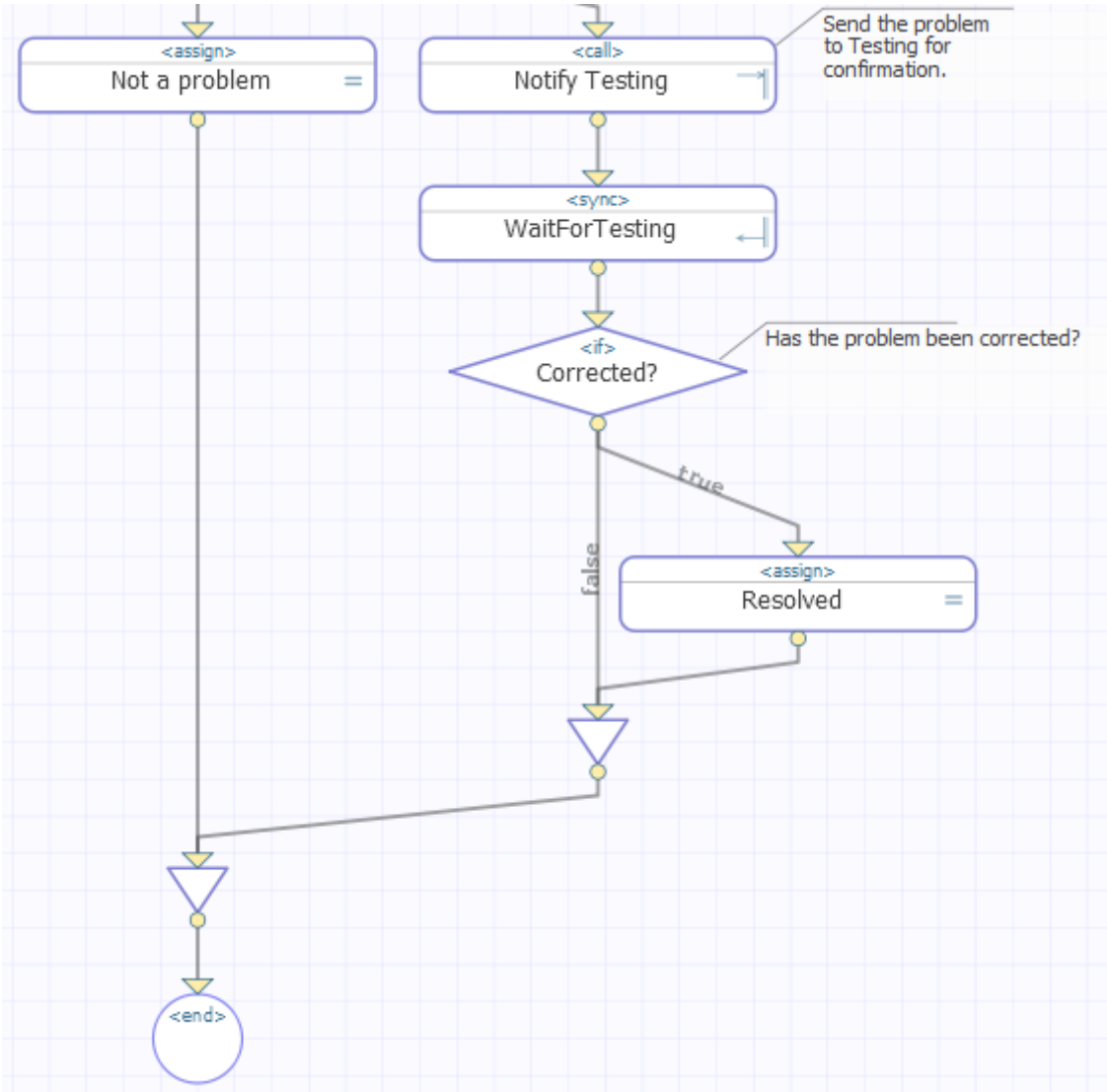
ビジネス・プロセス・デザイナで表示した場合、HelpDesk ビジネス・プロセスは次のように表示されます。



展開された while ループの上部は、以下のように表示されます。



下部は以下のとおりです。



While ループ内のコードは、ワークフローに対して 2 つの呼び出しを実行します。各呼び出しには、組み込みタスク要求クラスおよびタスク応答クラスが使用されます。2 つの <call> 要素の違いは、次のように BPL ソース・コード内で簡単に確認できます。

Class Definition

```
Class Demo.Workflow.HelpDesk Extends Ens.BusinessProcessBPL
{

XData BPL
{
<process request='Demo.Workflow.ProblemReport'
         response='Ens.Response' >
  <context>
    <property name='Resolved' type='%Boolean' initialexpression='0' />
    <property name='DevelopmentAction' type='%String' />
    <property name='TestingAction' type='%String' />
  </context>

  <sequence>
    <while name='while'
          condition='context.Resolved=0' >
      <annotation>
        Work on this problem until it is resolved.
        (Click on the loop icon to see details of this while loop.)
      </annotation>

      <call name='Notify Development'
            target='Demo-Development'
```

```
           async='1' >
        <annotation>
          Send the problem report to Development for review.
        </annotation>
        <request type='EnsLib.Workflow.TaskRequest' >
          <assign property='callrequest.%Actions'
                  value='"Corrected,Ignored"'
                  action='set' />
          <assign property='callrequest.%Subject'
                  value='"Problem reported by "_request.ReportedBy'
                  action='set' />
          <assign property='callrequest.%Message'
                  value='request.Problem'
                  action='set' />
          <assign property='callrequest.%FormFields'
                  value='"Comments"'
                  action='set' />
        </request>
        <response type='EnsLib.Workflow.TaskResponse' >
          <assign property='context.DevelopmentAction'
                  value='callresponse.%Action'
                  action='set' />
        </response>
      </call>

      <sync name='WaitForDevelopment' calls='Notify Development' type='all' />

      <if name='Fixed?' condition='context.DevelopmentAction="Corrected"' >
        <annotation>
          If Development fixed the problem, test it.
        </annotation>

        <true>
          <call name='Notify Testing'
                target='Demo-Testing'
                async='1' >
            <annotation>
              Send the problem to Testing for confirmation.
            </annotation>
            <request type='EnsLib.Workflow.TaskRequest' >
              <assign property='callrequest.%Actions'
                      value='"Corrected,Retest"'
                      action='set' />
              <assign property='callrequest.%Subject'
                      value='"Test this problem from "_request.ReportedBy'
                      action='set' />
              <assign property='callrequest.%Message'
                      value='request.Problem'
                      action='set' />
            </request>
            <response type='EnsLib.Workflow.TaskResponse' >
              <assign property='context.TestingAction'
                      value='callresponse.%Action'
                      action='set' />
            </response>
          </call>

          <sync name='WaitForTesting' calls='Notify Testing' type='all' />

          <if name='Corrected?' condition='context.TestingAction="Corrected"' >
            <annotation>Has the problem been corrected?</annotation>
            <true>
              <assign name='Resolved'
                      property='context.Resolved'
                      value='1'
                      action='set' />
            </true>
          </if>
        </true>

        <false>
          <assign name='Not a problem'
                  property='context.Resolved'
                  value='1'
                  action='set' />
        </false>

      </if>
    </while>
  </sequence>
</process>
}
}
```

A.5 コントロール・フローのサンプル

プロダクションの実行中、サンプルの BPL ビジネス・プロセス Demo.Workflow.HelpDesk は以下のように機能します。このコントロール・フローの手順を、前の節で示した BPL ソース・コードで対応している文と比較してみてください。

1. メッセージ・タイプ Demo.Workflow.ProblemReport によって、問題の報告者がビジネス・プロセスに伝えられ、問題を説明する簡単なテキスト文字列が表示されます。これら 2 つのプロパティの値は、このビジネス・プロセスを呼び出すビジネス・ホストによって提供されます。
2. Demo-Development ワークフロー・オペレーションへの <call> の準備に当たり、Demo.Workflow.HelpDesk ビジネス・プロセスは受信要求プロパティ ReportedBy および Problem を使用して、タスク要求フィールド %Subject および %Message にそれぞれ入力します。これ以外の 2 つの組み込みタスク・プロパティも機能を開始します。この call は、%Actions フィールドに `Corrected,Ignored` の値を割り当てることによって、実行可能なユーザ・アクションのリストを作成します。また、呼び出しはこのタスクを確認するユーザがデータを入力できるように、`Comments` と呼ばれるフォーム・フィールドも用意します。
3. ビジネス・プロセスは、Demo-Development への <call> を非同期で作成し、タスク応答をキャッチするための <sync> 要素を設定します。
4. ワークフロー・エンジンは、タスクを Demo-Development ワークフロー・ロールの各ワークフロー・ユーザと関連付けます。
5. ワークフロー・ユーザの 1 人がタスクを受け入れます。ワークフロー・エンジンはそのユーザにタスクを割り当てます。
6. 割り当てられたユーザは、[ ] フィールドを編集し、アクション [**修正されました**] または [**無視されました**] のいずれかをクリックします。
7. タスク応答が返されます。その %Action フィールドには、タスクを完了したユーザ・アクションの値が含まれます ([ ] または [ ])。ビジネス・プロセスは、DevelopmentAction と呼ばれる実行コンテキストのプロパティにこの値を保存します。
8. ビジネス・プロセスは、DevelopmentAction 値をテストするために、<if> 要素を使用します。結果は以下のとおりです。
   - DevelopmentAction が [ ] の場合、この値はヘルプ・デスクのビジネス・プロセスに、その <if> 要素の <true> 部分を実行する必要があることを知らせます。<true> 要素は、次の手順に示すように、Demo-Testing ワークフロー・ロールへの <call> を発行します。
   - DevelopmentAction が [ ] ではない場合、この値はヘルプ・デスクのビジネス・プロセスに、BPL ソース・コードの最後の方にある、その <if> 要素の <false> 部分を実行する必要があることを知らせます。ユーザによって DevelopmentAction が されると、コントロールはこの文に至ります。ビジネス・プロセスは、その呼び出し側に最終的な応答を返し、報告された問題は `Not a problem` であるとコメントして、ブーリアン応答値 Resolved を 1 (真) に設定します。
9. Demo-Testing ワークフロー・オペレーションへの <call> の準備に当たり、ビジネス・プロセスは元の受信要求プロパティ ReportedBy および Problem を使用して、タスク要求フィールド %Subject および %Message にそれぞれ入力します。ただし、前の <call> とは異なり、Demo-Testing への <call> には定義されたフォーム・フィールドはありません。また、実行可能なユーザ・アクションのリストが異なります。相違点は %Actions にあります。この <call> で %Actions フィールドに割り当てられる値は `Corrected,Retest` です。
10. ビジネス・プロセスは、Demo-Testing への <call> を非同期で作成し、タスク応答をキャッチするための <sync> 要素を設定します。
11. ワークフロー・エンジンは、タスクを Demo-Testing ワークフロー・ロールの各ワークフロー・ユーザと関連付けます。
12. ワークフロー・ユーザの 1 人がタスクを受け入れます。ワークフロー・エンジンはそのユーザにタスクを割り当てます。

13. 割り当てられたユーザは、[修正されました] または [再テスト] をクリックします。
14. タスク応答が返されます。その %Action フィールドには、タスクを完了したユーザ・アクションの値が含まれます ([ ] または [ ])。ビジネス・プロセスは、TestingAction と呼ばれる実行コンテキストのプロパティにこの値を保存します。
15. ビジネス・プロセスは、TestingAction 値をテストするために、<if> 要素を使用します。結果は以下のとおりです。
    - TestingAction が `Corrected` の場合、ビジネス・プロセスはその呼び出し側に最終的な応答を返し、報告された問題は `Resolved` であるとコメントして、ブーリアン応答値 Resolved を 1 (真) に設定します。
    - TestingAction が `Corrected` ではない場合、ブーリアン応答値 Resolved は初期値の 0 (偽) のままです。ビジネス・プロセスは再度 <while> ループの先頭に入ります。

A.6 ダッシュボードとメトリック

以下のビジネス・メトリック・クラス Demo.Workflow.WFMetric のサンプル・コードは、3 つの使用可能な統計メソッドを呼び出して、戻り値をメトリック・プロパティに割り当てています。次の付録では、使用可能なメソッドについて説明しています。

Class Definition

```
/// Sample business metric class for Workflow demo
Class Demo.Workflow.WFMetric Extends Ens.BusinessMetric
{

/// Active Tasks
Property ActiveTasks As Ens.DataType.Metric(AUTOHISTORY = 10, RANGELOWER = 0,
                                    RANGEUPPER = 50, UNITS = "Tasks")
                                    [ MultiDimensional ];

/// Active Load
Property Load As Ens.DataType.Metric(AUTOHISTORY = 10, RANGELOWER = 0,
                                    RANGEUPPER = 100, THRESHOLDUPPER = 90,
                                    UNITS = "%") [ MultiDimensional ];

/// Completed Tasks (since previous day)
Property CompletedTasks As Ens.DataType.Metric(AUTOHISTORY = 10, RANGELOWER = 0,
                                    RANGEUPPER = 100, UNITS = "Tasks")
                                    [ MultiDimensional ];

/// Calculate and update the set of metrics for this class
Method OnCalculateMetrics() As %Status
{
  // set the values of our metrics
  // %Instance is the current instance (RoleName in this case)
  Set tRole = ..%Instance

  Set ..ActiveTasks = ##class(EnsLib.Workflow.Engine).BamActiveTasks(tRole)
  Set ..Load = ##class(EnsLib.Workflow.Engine).BamActiveLoad(tRole)

  // Get task since start of previous day
  Set tStart = $ZDT($H-1,3)

  Set ..CompletedTasks =
      ##class(EnsLib.Workflow.Engine).BamCompletedTasks(tRole,tStart)

  Quit $$$OK
}

/// Set of instances for this metric class
/// There is one instance for every defined role.
Query MetricInstances() As %SQLQuery
{
  SELECT Name FROM EnsLib_Workflow.RoleDefinition
}

}
```

ここで使用されている ObjectScript 関数 $ZDT ($ZDATETIME) と $H ($HOROLOG) については、″ObjectScript リファレンス″ の “ObjectScript 関数” の章を参照してください。

B

使用可能なワークフロー・メトリック

EnsLib.Workflow.Engine クラスは、ワークフロー・ロールに関する統計を報告するメソッドを提供します。

ビジネス・メトリックを定義する際にこれらのメソッドを使用することで、ユーザがワークフローの進捗状況を追跡および分析できるようになります（詳細な手順は、"プロダクションの開発" の "ビジネス・メトリックの定義" を参照してください。また、サンプルを紹介している "ワークフロー・サンプルの紹介" も参照してください）。

そのメソッドは以下のとおりです。

BamActiveTasks()

```
ClassMethod BamActiveTasks(pRole As %String) As %Integer
```

指定のワークフロー・ロールのアクティブなタスクの数を返します。入力引数の pRole は、ワークフロー・ロールの構成名です。このメソッドは、このロールに割り当てられたタスクのリストを取得して、これらのタスクのうちまだ完了していないタスクを特定することで、アクティブなタスクの数を算出します。

BamActiveLoad()

```
ClassMethod BamActiveLoad(pRole As %String) As %Integer
```

指定のワークフロー・ロールのアクティブな負荷を示す値を返します。入力引数の pRole は、ワークフロー・ロールの構成名です。このメソッドは、このワークフロー・ロールのアクティブなタスクの現在の数を取得して、その数とワークフロー・ロール定義内の Capacity プロパティの値を比較し、その比較結果をこのワークフロー・ロールの総合処理能力のパーセンテージとして提示することにより、アクティブな負荷を算出します。数式は以下のとおりです。

```
(ActiveTasks/Capacity)*100
```

BamCompletedTasks()

```
ClassMethod BamCompletedTasks(pRole As %String,
                              pStart As %TimeStamp = "",
                              pEnd As %TimeStamp = "") As %Integer
```

指定のワークフロー・ロールの完了したタスクの数を返します。それぞれ以下の意味を持ちます。

- pRole は、ワークフロー・ロールの構成名です。
- pStart と pEnd は、%TimeStamp 形式：yyyy-mm-dd で表されます。

このメソッドは、このロールに割り当てられているタスクのリストを取得し、これらのタスクのうちステータスが [完了] になっているタスクを特定することで、完了したタスクの数を算出します。BamCompletedTasks() は、pStart と pEnd で指定されている期間内に開始したタスクのみを考慮対象とします。pEnd 時間が経過しているタスクは完了しているものとなります。

BamTasksWithStatus()

```
ClassMethod BamTasksWithStatus(pRole As %String,
                               pStatus As %String,
                               pStart As %TimeStamp = "",
                               pEnd As %TimeStamp = "") As %Integer
```

指定のワークフロー・ロールに特定のステータスのあるタスクの数を返します。それぞれ以下の意味を持ちます。

- pRole は、ワークフロー・ロールの構成名です。
- pStatus は、ロールのステータスを表す以下のいずれかの文字列です。
  - [ ]
  - [ ]
  - [ ]
  - [ ]
  - [ ]
- pStart と pEnd は、%TimeStamp 形式：yyyy-mm-dd で表されます。

このメソッドは、そのロールに割り当てられているタスクのリストを取得し、これらのタスクのうち指定されたステータスに現在なっているタスクの数を特定します。BamTasksWithStatus() は、pStart と pEnd で指定されている期間内に開始したタスクのみを考慮対象とします。pEnd 時間が経過したタスクは、その理由にかかわらず、完了しているものとなります。

目的のステータスが `Completed` である場合は、BamCompletedTasks() を使用するほうが簡単です。

BamAvgTaskTime()

```
ClassMethod BamAvgTaskTime(pRole As %String,
                           pStart As %TimeStamp = "",
                           pEnd As %TimeStamp = "") As %Integer
```

指定のワークフロー・ロールで完了したタスクの平均存続時間 (秒単位) を返します。それぞれ以下の意味を持ちます。

- pRole は、ワークフロー・ロールの構成名です。
- pStart と pEnd は、%TimeStamp 形式：yyyy-mm-dd で表されます。

このメソッドは、pStart と pEnd で指定されている期間内に開始したタスクのみを考慮対象とします。pEnd 時間が経過しているタスクは完了しているものとなります。

BamMinTaskTime()

```
ClassMethod BamMinTaskTime(pRole As %String,
                           pStart As %TimeStamp = "",
                           pEnd As %TimeStamp = "") As %Integer
```

指定のワークフロー・ロールで完了したタスクの最短存続時間 (秒単位) を返します。それぞれ以下の意味を持ちます。

- pRole は、ワークフロー・ロールの構成名です。
- pStart と pEnd は、%TimeStamp 形式：yyyy-mm-dd で表されます。

このメソッドは、pStart と pEnd で指定されている期間内に開始したタスクのみを考慮対象とします。pEnd 時間が経過しているタスクは完了しているものとなります。

BamMaxTaskTime()

```
ClassMethod BamMaxTaskTime(pRole As %String,
                           pStart As %TimeStamp = "",
                           pEnd As %TimeStamp = "") As %Integer
```

指定のワークフロー・ロールで完了したタスクの最長存続時間（秒単位）を返します。それぞれ以下の意味を持ちます。

- pRole は、ワークフロー・ロールの構成名です。
- pStart と pEnd は、%TimeStamp 形式：yyyy-mm-dd で表されます。

このメソッドは、pStart と pEnd で指定されている期間内に開始したタスクのみを考慮対象とします。pEnd 時間が経過しているタスクは完了しているものとなります。