IBM MQSeries Workflow

プログラミングの手引き

バージョン 3.2.2

SH88-7352-04
(英文原典：SH12-6291-05)

IBM MQSeries Workflow

プログラミングの手引き

バージョン 3.2.2

SH88-7352-04
(英文原典：SH12-6291-05)

ご注意!

本書、および本書がサポートする製品をご使用になる前に、997ページの『付録C. 特記事項』にある一般的な情報を必ずお読みください

本書は、IBM MQSeries Workflow (製品番号 5697-FM3) のバージョン 3、リリース 2、モディフィケーション 2、および新版において特に断りのない限り、それ以降のすべてのリリースおよびモディフィケーション・レベルに適用されます。

本書は SH88-7352-03 に対する新版です。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

原 典:	SH12-6291-05 IBM MQSeries Workflow Programming Guide Version 3.2.2
発 行:	日本アイ・ビー・エム株式会社
担 当:	ナショナル・ランゲージ・サポート

第1刷 2000.8

この文書では、平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財) 日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

本書について

本書は、IBM MQSeries Workflow のクライアント・アプリケーション・プログラミング・インターフェース (以後 MQSeries Workflow API と呼びます) と、XML メッセージ・インターフェース (XMI) の使用法について説明します。本書は、第 1 部で API の基礎を成す概念について説明し、残りは API のリファレンス・マニュアルになっています。本書ではまた、MQSeries Workflow の事前定義データ構造、および MQSeries Workflow の制御のもとで実行されるアプリケーションをデバッグする方法についても説明します。

本書の対象読者

本書の対象読者は、MQSeries Workflow API を使用するプログラムを設計および実行するプログラマー、および IBM MQSeries Workflow を使ったワークフロー・モデルの設計に携わるプログラマーです。本書では、読者は熟練したプログラマーで、モデル定義プロセスの概念を理解していることが前提になっています。プログラマーは、各自が使用しているオペレーティング・システムに通じていなければなりません。

追加情報の入手方法

MQSeries Workflow のホーム・ページ、
http://www.software.ibm.com/ts/mqseries/workflow を参照してください。

関連資料のリストについては、1009ページの『MQSeries Workflow 資料』を参照してください。

本書の編成

第 1 部では、さまざまな MQSeries Workflow API について概説します。バージョン 3 の API に共通なすべての概念について説明し、サポートされている API を紹介します。

- 1ページの『第1部 プログラミングの概念』では、すべての MQSeries Workflow API の基礎となる概念について説明しています。この中では、動作ごとに関数 / メソッドをグループに分けて、基本メソッドとアクセス機能メソッドについて全般的な説明をします。

- 149ページの『第2部 C および C++ API』では、C および C++ の API に固有な概念を説明し、アプリケーション・プログラムがコンパイルおよびリンクされる方法について示します。
- 167ページの『第3部 ActiveX コントロール』では、ActiveX コントロールについて概説します。
- 177ページの『第4部 JAVA API』では、Java API について概説します。
- 211ページの『第5部 XML メッセージ・インターフェース』では、MQSeries Workflow 実行サーバーからのアクションを要求するための XML の使用に関するいくつかの概念について説明します。さらに、XML を使って実行サーバーからプログラムを呼び出す方法についても説明します。

247ページの『第6部 MQSeries Workflow API の使用』では、MQSeries Workflow 実行機能によってサポートされている機能について概説します。MQSeries Workflow API によってサポートされているすべての関数 / メソッドは、オブジェクトごとに要約されています。

本書の次の部分の内容はリファレンス情報となっています。

- 353ページの『第7部 プログラミング・インターフェース』では、アプリケーションがプロセス・インスタンスおよびコンテナー・データを処理したり MQSeries Workflow 実行サービスにおいてログオンおよびログオフしたりするために、ワークリストおよび作業項目を処理するのに使える MQSeries Workflow API について説明しています。すべてのアクション、アクティビティー・インプリメンテーション、およびプログラム実行管理関数 / メソッドはオブジェクトごとに説明されています。基本メソッドおよびアクセス機能メソッドについては、85ページの『第11章 関数 / メソッドの種類』を参照してください。
- 733ページの『第8部 ActiveX コントロールを使った作業』では、ActiveX コントロールでサポートされているメソッドとイベントについて説明されています。

817ページの『第9部 例およびシナリオ』には、API を使用する方法を示す例が提供されています。

877ページの『第10部 ロータス ノーツ API の使用』では、MQSeries Workflow とロータス ノーツの統合のために提供されている API について説明しています。

981ページの『付録A. 構文図の読み方』では、構文図の各部分について説明しています。

983ページの『付録B. MQSeries FlowMark バージョン 2 互換モード』では、FlowMark バージョン 2 プログラムを実行する方法、また MQSeries Workflow バージョン 3 とバージョン 2 の相違点について説明しています。さらに、バージョン 2 の C++ プログラムを変更してバージョン 3 用プログラムにするための方法についても述べています。

巻末には、本書で使用している用語を定義した用語集、参照文献、および索引があります。

改訂の要約

IBM MQSeries Workflow バージョン 3.2.2 でこのマニュアルに加えられた変更は次のとおりです。

- 作業項目を照会したりワークリストを作成したりするための実行サービス関数 / メソッドにおいて、新しいフィルター基準 CREATION_TIME を指定できるようになりました。
- 実行サービスの新しい関数 / メソッド SetPersonAbsent() が公開されました。
- コンテナーの新しい関数 / メソッド SetStringCcsid() が公開されました。
- アクティビティー・インスタンスの新しい関数 / メソッド InContainer()、OutContainer()、PersistentObject()、および Terminate() が公開されました。
- プロセス・インスタンスの新しい関数 / メソッド OutContainer() が公開されました。
- 作業項目および通知の新しい関数 / メソッド PersistentOidOfProcessInstance() が公開されました。
- 作業項目およびアクティビティー・インスタンス通知の新しい関数 / メソッド PersistentOidOfActivityInstance() および ActivityInstance() が公開されました。
- アクティビティー・インプリメンテーションにおいて、関連するアクティビティー・インスタンス・オブジェクト ID が検索できるようになりました。すなわち、プログラム実行エージェントの新しい関数 / メソッド PersistentOidOfActivityInstance() および RemotePersistentOidOfActivityInstance() が公開されました。
- アクティビティー・インスタンスの最大優先度、したがって作業項目またはアクティビティー・インスタンス通知の最大優先度もともに 999 にすることができます。

IBM MQSeries Workflow バージョン 3.2.1 でこのマニュアルに加えられた変更は次のとおりです。

- プロセス・インスタンスの作成・開始のための XML メッセージ・インターフェースのサポートが追加されました。さらに、ユーザー定義のプログラム実行サーバーにおいて MQSeries Workflow 実行サーバーによりアクティビティー・インプリメンテーションを開始することができます。

- プロセス・テンプレート InContainer() 関数 / メソッドは InitialInContainer() に名前を変更できます。 InContainer() は使用すべきではありません。
- プロセス・テンプレートの新しい関数 / メソッド ProgramTemplate() が公開されました。
- プログラム・テンプレートを記述するための新しいクラスと新しい関数が追加されました。プログラム・テンプレートは、同期および非同期のどちらでも関数 / メソッド Execute() をサポートします。
- プログラム・データにおいて、新しいアクセス関数 / メソッド ExecutionMode()、ExecutionUser()、ProgramTrusted() が公開されました。
- 実行サービスにおいて、システム・グループだけを指定するための新しい割り振り方法が用意されました。これは、IBM MQSeries のクラスター機能の活用のためです。
- プロセス・インスタンスを照会したりプロセス・インスタンス・リストを作成したりするための実行サービス関数 / メソッドにおいて、新しいフィルター属性 START_TIME を指定できるようになりました。

IBM MQSeries Workflow バージョン 3.2 でこのマニュアルに加えられた変更は次のとおりです。

- HP-UX と Sun Solaris がサポートされるようになりました。
- JAVA サポートが追加されました。
- ActiveX がプロセス・インスタンス・モニターをサポートします。
- 構成識別子の指定がサポートされます。
- 実行サービスの新しいアクション関数 / メソッド Refresh() が公開されました。
- プロセス・テンプレートにおいて、新しいアクション関数 / メソッド ExecuteProcessInstance() を同期および非同期が公開されました。このことは、非同期通信プロトコルが追加されたことを意味します。
- プロセス・インスタンスの新しいアクション関数 / メソッド Restart() が公開されました。
- 作業項目とアクティビティー・インスタンス通知の新しいアクション関数 / メソッド StartTool() が公開されました。作業項目の新しい関数 / メソッド CancelCheckOut() が公開されました。
- 作業項目 Restart()、ForceFinish()、および ForceRestart() 関数 / メソッドから制限が取り除かれました。*プロセス* によってインプリメントされた作業項目が再始動または終了できるようになりました。

IBM MQSeries Workflow バージョン 3.1.2 でこのマニュアルに加えられた変更は次のとおりです。

- アイテム・オブジェクトの新しいアクション、関数 / メソッド Delete() が公開されました。
- アイテムの変更が現在のクライアントにプッシュされるようになりました。この機能は作業項目、アクティビティー・インスタンス通知、およびプロセス・インスタンス通知に適用され、C 言語 API と C++ API でサポートされます。
- プロセス・インスタンス・モニター・サポートが C 言語 API と C++ API に追加されました。

IBM MQSeries Workflow バージョン 3.1.1 でこのマニュアルに加えられた変更は次のとおりです。

- OS/2(R) のサポートが追加されました。
- 担当者オブジェクトの新しいアクション関数 / メソッド Refresh()、SetAbsence()、および SetSubstitute() が公開されました。
- プロセス・テンプレートの新しいアクション関数 / メソッド Delete() が公開されました。
- プロセス・テンプレートとプロセス・インスタンスの IsTerminatedOnError() アクセス関数 / メソッドは削除されました。
- 新しいオブジェクト FmcjError が追加されました。これは、作業項目が InError 状態である理由を記述するものです。作業項目とアクティビティー・インスタンス通知によってエラーの理由が戻されます。
- FmcjWorkitem::Checkout() は新しいプログラム定義、外部サービスの定義を戻します。新しいオブジェクト FmcjExternalOptions が追加され、外部サービスのプロパティーの照会が可能になりました。
- 作業項目の新しいアクション関数 / メソッド Terminate() が公開されました。
- ActiveX サポートが追加されました。
- OS/2 におけるバージョン 2 REXX サポートが追加されました。
- バージョン 2 ロータス ノーツのサポートが追加されました。
- バージョン 2 C++ Logon() に、バージョン 3 セッション・モードの使用を考慮するオプションが設けられました。詳しくは、985ページの『MQSeries FlowMark バージョン 2 との相違点』を参照してください。

第1部 プログラミングの概念

第 1 部では、MQSeries Workflow のプログラミングの概念について総体的に紹介します。

第1章 プログラミングの概念について

この章では、IBM MQSeries Workflow (以後 MQSeries Workflow) で使用するアプリケーション・プログラムの設計との関連において、ワークフローのモデル定義の概念について説明します。

MQSeries Workflow は、プロセスをモデル定義し、モデル定義したワークフロー・モデルのアクティビティーにアプリケーションを割り当てる手段となります。これによって、ワークフロー・マネージャーは、アクティビティーの制御やデータの流れを自動化することができます。

作業は、アクティビティー・インスタンスを実行する担当者に渡すことができます。アクティビティー・インスタンスを実行しなければならないアプリケーション・プログラムは、ユーザーがアクティビティー・インスタンスを開始した時点で開始されるよう設計することができます。

プロセスのモデル定義におけるプログラマーの役割

ワークフロー・モデルが定義されると、プログラム・アクティビティーのサポートに必要なアプリケーションとデータ構造が識別されます。プログラマーは、新しいアプリケーションを作成するか、既存のアプリケーションを統合するか、既存のアプリケーションを改良することにより、それらのプログラム・アクティビティーをサポートできます。

既存のアプリケーションをワークフロー・モデルを使用して改良するために、プログラマーは、その企業で使用されているそれらのアプリケーションが機能的に分解可能であるかどうかを判別しなければなりません。制御と流れの論理をアプリケーションから切り離し、開始および終了条件をワークフロー・モデルに移し、プログラムをワークフロー・マネージャーによって適切な時点で呼び出される複数のモジュールに分割します。分割されたモジュールは、ワークフロー・モデルで定義されるプログラム・アクティビティーを実行するために割り当てられるアプリケーションになります。

ほとんどのアプリケーションには多くの異なる関数が含まれており、またそれらのアプリケーションの多くは、プロセスのさまざまな段階でさまざまなアクティビティーをサポートします。プログラムの 1 つの関数によって生成された出力は、同じプログラムの別の関数への入力として使用することができます。

したがって、同じアプリケーションを使用して、1 つのワークフロー・モデルに含まれる多くの異なるプログラム・アクティビティーをサポートすることができます。

貴社においても、 ERP (エンタープライズ・リソース・マネージメント) またはワープロ・アプリケーションやスプレッドシート・アプリケーションなどのパッケージ・アプリケーションを使用しているかもしれません。

このようなプログラムは、分解できない場合もあります。しかしプログラマーは、コンテナーの内容を照会したり、アクティビティー・インスタンスの開始時に入力コンテナーからプログラムにデータを送ったり、アクティビティーの完了時に出力コンテナーにデータを送ったりするシェル・プロシージャーを作成することができます。

割り当てられたプログラムからの戻りコードは、後に終了条件や分岐条件の評価に使用することができます。

第2章 プログラミング・インターフェース

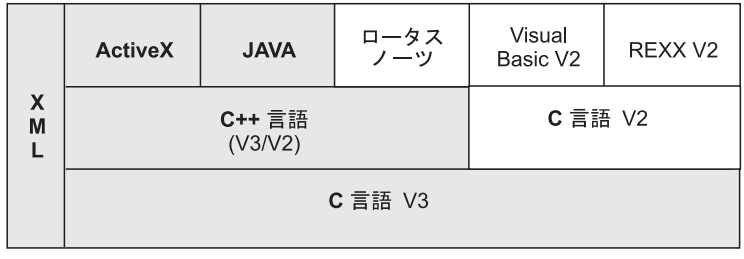
MQSeries Workflow には、アプリケーション・プログラム・インターフェースおよび XML メッセージ・インターフェースのサポート、および一連の事前定義データ構造メンバーが用意されており、ワークフロー・モデルで使用するアプリケーションを開発するプログラマーをサポートします。また、いくつかのプログラミング・サンプルも用意されています。

プログラミング言語ベースのプログラミング・モデルにおいて、クライアント・アプリケーションは関数 / メソッドを呼び出して要求を実行します。メッセージ・ベースのプログラミング・モデルの場合、要求とその要求を実行するのに必要な情報は、クライアント・アプリケーションと特定のサーバーとの間のメッセージ・キューイング・システムを介して交換されるメッセージに入れられます。

MQSeries Workflow 事前定義データ構造体メンバーは、現行プロセス、アクティビティー、またはブロックについての情報を提供します。また、それらのメンバーは、プロセス・インスタンスまたはアクティビティー・インスタンスの操作特性に関連付けられます。

本書では、次のような MQSeries Workflow プログラミング援助機能について説明します。

- MQSeries Workflow C 言語 API
- MQSeries Workflow C++ 言語 API
- MQSeries Workflow ActiveX コントロールおよび OLE オブジェクト
- MQSeries Workflow Java API
- MQSeries Workflow XML メッセージ・インターフェース
- MQSeries Workflow ロータス ノーツ API



MQSeries Workflow の実行機能サービスを要求するための基本となるインターフェースは、C 言語 API と XML メッセージ・インターフェースです。 C 呼び出しをサポートする言語であればどの言語からであっても C 言語の関数を呼び出すことができます。詳細については、155ページの『第14章 コンパイルとリンク』を参照してください。 C 言語 API の上に、C++ 言語 API が提供されます。 C++ API はインライン・メソッドの小規模な層であり、ソース・コードで提供されているので、一般によく使用される C++ コンパイラーであればコンパイル可能です。 ActiveX API と Java API は、C++ 層の上にインプリメントされています。ロータス ノーツ API についても同じですが、処理は FlowMark バージョン 2 互換モードで実行されます。 MQSeries Workflow では、ドキュメント記述言語として XML 1.0 標準規格を使用します。バージョン 3 API だけでなく、FlowMark バージョン 2 C 言語、C++、VisualBasic、および REXX 用の各 API もサポートされています。

MQSeries Workflow API としては、次のような関数 / メソッドが用意されています。

- プロセス・モデルの実行、つまり、プロセス・インスタンスおよびコンテナー・データの処理、およびワークリストと作業項目の処理
- 実行の進行状況のモニター
- プロセス管理者関数の発行
- MQSeries Workflow サーバーが送信した情報の受信
- アクティビティー・インプリメンテーションに関連したコンテナー・データの処理。ロータス ノーツ API はチェックアウト / チェックイン機構を使用します。

第3章 プログラム言語 API の前提条件

MQSeries Workflow のアプリケーション開発では、適切な環境が設定されていることが前提となります。つまり、

- アプリケーションを開発しようとするマシン上に MQSeries Workflow 開発キットがインストールされていること。
- サポートされている言語のいずれかのコンパイラーがインストールされていて構成されていること。

詳細については、149ページの『第2部 C および C++ API』、167ページの『第3部 ActiveX コントロール』、および 177ページの『第4部 JAVA API』を参照してください。

第4章 MQSeries Workflow アプリケーションの構築

概要

MQSeries Workflow アプリケーション・プログラミング・インターフェース (API) を使って実行できる作業は、基本的に次の 2 とおりあります。

- MQSeries Workflow の GUI (グラフィカル・ユーザー・インターフェース) やコマンド行インターフェースを使う代わりに、独自のクライアント・アプリケーションを作成できます。たとえば、次のことができます。
  - ユーザーに対して提供される MQSeries Workflow の機能を制御する。
  - ユーザーが慣れている方法で MQSeries Workflow 機能を提示する。
  - ユーザー介入なしに、選択した MQSeries Workflow タスクを実行する。
- ワークフロー・プロセス・モデルにアクティビティーまたはサポート・ツールをインプリメントするプログラムを作成できます。

通常、これら 2 種類のプログラムには、 『MQSeries Workflow クライアント・アプリケーション』および『MQSeries Workflow アクティビティー・インプリメンテーションまたはサポート・ツール』で説明されているいくつかの特定のパーツが含まれています。各言語に対応する章を参照してください。

MQSeries Workflow API の基礎となる概念は、MQSeries Workflow API を使用するすべてのプログラムで共通です。ここではその概念を要約します。この後の章のほうで詳しく説明します。

プログラム言語 API の概念

作業項目およびプロセス・インスタンスなどのすべての永続オブジェクトには、サーバーから照会された時点での状態を表す一時オブジェクトによってアクセスします。 C 言語 API では、いわゆる*ハンドル* がそのような一時オブジェクトを指すポインターを表します。

オブジェクトでのアクションを要求するには、適切な MQSeries Workflow サーバーとのセッションを確立しておく必要があります。そうすれば、アクション自体を同期的に実行することができます。一部のアクションは非同期でも実行できます。

サーバーからクライアントへは、許可されているオブジェクトだけが戻されます。

C 言語 API の各関数と C++、ActiveX、または Java 言語の各 API の各メソッド (以後関数 / メソッド) を、別々にオブジェクトでのアクションに使ったり、オブジェクトの各プロパティーへのアクセスに使ったりできます。このアプローチでは、コンパイラーで関数 / メソッドのパラメーターを検査することができ、MQSeries Workflow でサポートされるオブジェクト - アクション・パラダイムをうまく表現することができます。

C および C++ 言語では、詳細なエラー情報はいわゆる*結果オブジェクト* によって提供されます。このオブジェクトは、アクション関数 / メソッドによって設定される戻りコードに加えて使用できます。結果オブジェクトの詳細については、163ページの『第16章 結果オブジェクト』を参照してください。

オブジェクトはアプリケーション・プログラマーによって管理されますが、オブジェクト・メモリーは MQSeries Workflow API によって所有されます。アプリケーション・プログラマーは、割り振りまたは照会、および割り振り解除のメカニズムを使って一時オブジェクトの存続期限を決めます。 MQSeries Workflow API は一時オブジェクトの内部構造を隠します。

XML メッセージ・インターフェースの概念

どの永続オブジェクトにも固有名でアクセスしますが、そのことは、固有性を保つためにそのオブジェクトの実際の名前に、そのオプションの識別子の印刷可能バージョンを埋め込むことが必要になる可能性があるということになります。

プログラム言語 API の場合とは違って、アクションを要求するためにセッションを確立する必要はありません。しかし、アクションそのものの許可は必要です。

すべてのアクションは非同期的に実行されます。相関データがメッセージの一部に含まれているので、アプリケーションは MQSeries Workflow に送った要求と実行サーバーからの応答を互いに関係付けることができます。

エラー処理

アクション、アクティビティー・インプリメンテーション、プログラム実行管理関数 / メソッド、またはメッセージは、いずれも戻り値としていわゆる*戻りコード* を戻すことによって、呼び出しが正常に完了したかどうかを示します。Java 言語では、メソッドが正常に実行されない場合に適切な FmcException を戻します。 XML メッセージ・インターフェースは、戻りコードを応答メッセージに入れて戻します。戻りコードは、事前定義コード・セットのうちの 1 つ

です (『戻りコードのリスト』を参照)。各呼び出しの説明と一緒に、それぞれの関数 / メソッドの正確な戻りコードまたは例外のリストを示してあります。

プログラムは、今後発生する可能性のあるすべての戻りコードまたは例外を処理できるように設計してください。つまり、戻りコードが FMC_OK 以外のものであるかどうかを確認するだけではなく、戻りコードを明示的に検査する場合には、予期しないエラーに常に注意しなければなりません。たとえば、*if* ステートメントをコーディングする場合は、 *else* ステートメントもコーディングしなければなりません。また、*switch* ステートメントをコーディングする場合は、 *default* ケースもコーディングしなければなりません。

戻りコード以外に、呼び出しの結果をさらに詳しく説明するための、いわゆる *結果オブジェクト* にも、C および C++ 言語でアクセスできます。 163ページの『第16章 結果オブジェクト』を参照してください。

基本およびアクセス関数 / メソッドは、なんらかの値を戻したり、照会された値を戻り値として戻したりしません。それらは、一時オブジェクトを照会するものであって、省略時値を戻すことができるため、通常はエラーが発生することはありません。しかし、アプリケーション開発時に、誤ったハンドルを指定したり文字値を入れるには小さすぎるバッファーを指定したりすると、エラーになることがあります。トレースの検査以外でそのようなエラー状態を探索するには、*結果オブジェクト* を照会します。

戻りコードのリスト

以下のリストに、MQSeries Workflow API が発行する戻りコードまたは例外の数値を示します。可能な限り整数値ではなく記号名を使用してください。

表 1. 戻りコードのリスト

数値	記号値
0	FMC_OK
1	FMC_ERROR
10	FMC_ERROR_USERID_UNKNOWN
11	FMC_ERROR_ALREADY_LOGGED_ON
12	FMC_ERROR_PASSWORD
13	FMC_ERROR_COMMUNICATION
14	FMC_ERROR_TIMEOUT
15	FMC_ERROR_INVALID_CODE_PAGE
16	FMC_ERROR_INVALID_CHAR
100	FMC_ERROR_INTERNAL
101	FMC_ERROR_SERVER
102	FMC_ERROR_UNKNOWN

表 1. 戻りコードのリスト (続き)

数値	記号値
103	FMC_ERROR_MESSAGE_FORMAT
104	FMC_ERROR_MESSAGE_DATA
105	FMC_ERROR_RESOURCE
106	FMC_ERROR_NOT_LOGGED_ON
107	FMC_ERROR_NEW_OWNER_NOT_FOUND
108	FMC_ERROR_NO_OLD_OWNER
109	FMC_ERROR_OLD_OWNER_ABSENT
110	FMC_ERROR_NEW_OWNER_ABSENT
111	FMC_ERROR_ALREADY_STARTED
112	FMC_ERROR_MEMBER_NOT_FOUND
113	FMC_ERROR_MEMBER_NOT_SET
114	FMC_ERROR_WRONG_TYPE
115	FMC_ERROR_MEMBER_CANNOT_BE_SET
116	FMC_ERROR_MEMBER_INVALID
117	FMC_ERROR_FORMAT
118	FMC_ERROR_DOES_NOT_EXIST
119	FMC_ERROR_NOT_AUTHORIZED
120	FMC_ERROR_WRONG_STATE
121	FMC_ERROR_NOT_UNIQUE
122	FMC_ERROR_EMPTY
123	FMC_ERROR_NO_MANUAL_EXIT
124	FMC_ERROR_PROFILE
125	FMC_ERROR_INVALID_FILTER
126	FMC_ERROR_PROGRAM_EXECUTION
127	FMC_ERROR_PROTOCOL
128	FMC_ERROR_TOOL_FUNCTION
129	FMC_ERROR_INVALID_TOOL
130	FMC_ERROR_INVALID_HANDLE
131	FMC_ERROR_NOT_EMPTY
132	FMC_ERROR_INVALID_USER
133	FMC_ERROR_OWNER_ALREADY_ASSIGNED
134	FMC_ERROR_INVALID_NAME
135	FMC_ERROR_INVALID_PROGRAMID
136	FMC_ERROR_SIZE_EXCEEDED
137	FMC_ERROR_INVALID_TEMPLATE_NAME
138	FMC_ERROR_INFINITE_RECURSION
406	FMC_ERROR_WRONG_ACT_IMPL_KIND
500	FMC_ERROR_NON_LOCAL_USER
501	FMC_ERROR_WRONG_KIND
502	FMC_ERROR_INVALID_ACTIVITY
503	FMC_ERROR_CHECKOUT_NOT_POSSIBLE
504	FMC_ERROR_BACK_LEVEL_VERSION
505	FMC_ERROR_NEWER_VERSION

表 1. 戻りコードのリスト (続き)

数値	記号値
506	FMC_ERROR_INVALID_CORRELATION_ID
507	FMC_ERROR_NOT_ALLOWED
508	FMC_ERROR_BACK_LEVEL_OBJECT
509	FMC_ERROR_INVALID_CONTAINER
510	FMC_ERROR_UNEXPECTED_CONTAINER
511	FMC_ERROR_NO_PROGRAM_FOR_PLATFORM
800	FMC_ERROR_BUFFER
801	FMC_ERROR_INVALID_SESSION
802	FMC_ERROR_INVALID_TIME
804	FMC_ERROR_NO_MORE_DATA
805	FMC_ERROR_INVALID_OID
807	FMC_ERROR_INVALID_THRESHOLD
808	FMC_ERROR_INVALID_SORT
810	FMC_ERROR_INVALID_DESCRIPTION
811	FMC_ERROR_INVALID_INVOCATION_TYPE
812	FMC_ERROR_OWNER_NOT_FOUND
813	FMC_ERROR_INVALID_LIST_TYPE
814	FMC_ERROR_INVALID_RESULT_HANDLE
815	FMC_ERROR_MESSAGE_CATALOG
816	FMC_ERROR_INVALID_SPECIFICATION
817	FMC_ERROR_QRY_RESULT_TOO_LARGE
818	FMC_ERROR_NO_VERSION_2_FILTER
819	FMC_ERROR_INVALID_USER_CONTEXT
820	FMC_ERROR_MESSAGE_STRING
821	FMC_ERROR_MESSAGE_SIZE_EXCEEDED
900	FMC_ERROR_NO_SYS_ADMIN
901	FMC_ERROR_INVALID_SESSION_MODE
902	FMC_ERROR_PROGRAM_UNDEFINED
904	FMC_ERROR_PEA_NOT_LOCAL
905	FMC_ERROR_INVALID_ABSENCE_SPEC
1000	FMC_ERROR_NOT_SUPPORTED
1012	FMC_ERROR_PROGRAM_NOT_DEFINED
1014	FMC_ERROR_PEA_NOT_REACHABLE
1015	FMC_ERROR_INVALID_PEA_FROM_CTNR
1016	FMC_ERROR_INVALID_PEA_FROM_MODEL
1017	FMC_ERROR_INVALID_SYSTEM_FROM_CTNR
1018	FMC_ERROR_INVALID_SYSTEM_FROM_MODEL
1019	FMC_ERROR_SUB_PROC_TERMINATED_BY_ERROR
1020	FMC_ERROR_NO_PEA_FOUND_FOR_AUTO_START
1021	FMC_ERROR_NO_CTNR_ACCESS
1022	FMC_ERROR_INVALID_CONFIGURATION_ID
1023	FMC_ERROR_MIGRATION_OF_RUNNING_PROGRAM
1024	FMC_ERROR_MIGRATION_OF_CHECKEDOUT_SUSPENDED

表 1. 戻りコードのリスト (続き)

数値	記号値
1025	FMC_ERROR_MIGRATION_NO_SUBPROCESS
1100	FMC_ERROR_XML_DOCUMENT_INVALID
1101	FMC_ERROR_NO_MQSWF_DOCUMENT
1102	FMC_ERROR_XML_MESSAGE_NOT_SUPPORTED
1103	FMC_ERROR_XML_WRONG_DATA_STRUCTURE
1104	FMC_ERROR_XML_DATA_MEMBER_NOT_FOUND
1105	FMC_ERROR_XML_DATA_MEMBER_WRONG_TYPE
1106	FMC_ERROR_XML_BACKOUT_COUNT_EXCEEDED
1107	FMC_ERROR_XML_DOCUMENT_FORMAT
1108	FMC_ERROR_XML_PARAMETER_INCORRECT
1109	FMC_ERROR_XML_PARAMETER_SIGNATURE_INCORRECT
1110	FMC_ERROR_XML_INVALID_ELEMENT
2000	FMC_ERROR_INVALID_QUEUE_SCOPE
32013	FMC_ERROR_USER_SUPPORT_MISMATCH
32014	FMC_ERROR_SUPPORT_MODE_MISMATCH
32015	FMC_ERROR_IMPLEMENTATION_SUPPORT_MISMATCH
32202	FMC_ERROR_USER_NOT_AUTHORIZED
32203	FMC_ERROR_LOCAL_USER_REQUIRED
32204	FMC_ERROR_EXIT_ERROR

ActiveX GUI コントロールの例外のリスト

以下のリストに、MQSeries Workflow ActiveX GUI コントロールが発行する例外の数値を示します。可能な限り整数値ではなく記号名を使用してください。

表 2. ActiveX の例外のリスト

数値	記号値
1500	FMC_METHOD_EXCEPTION
1501	FMC_WRONG_INDEX
1502	FMC_MEMORY_EXCEPTION
1503	FMC_ERROR_PARAMETER
1504	FMC_OLE_EXCEPTION
1505	FMC_OLE_DISPATCH_EXCEPTION
1506	FMC_USER_EXCEPTION
1507	FMC_OBJECT_NOT_VALID
1508	FMC_OBJECT_STILL_VALID
1509	FMC_GUI_ALREADY_CONNECTED
1510	FMC_GUI_NOT_CONNECTED
1511	FMC_WRONG_CONTAINER_TYPE
1512	FMC_UNKNOWN_ITEM
1513	FMC_SET_CONTAINER_VALUE
1514	FMC_RECURSION_ERROR

デバッグに関する考慮事項

前提条件

プログラム言語インターフェースを使用する MQSeries Workflow アプリケーションをデバッグするには、適切な環境が設定されていることが前提となります。つまり、

- MQSeries Workflow DLL にアクセスできること。これは、MQSeries Workflow の標準インストールであれば自動的に保証されます。
- 個々のデバッグ要件を反映するテスト・データベースが作成されていること (『テスト・データベースの作成』を参照)。
- テストを実行できるよう、サーバー・マシンで MQSeries Workflow サーバーが実行されていること。
- 必要なサーバーに接続できること。これは、MQSeries Workflow 提供の構成検査プログラム *fmczchk* によって調べることができます (*IBM MQSeries Workflow: インストールの手引き* を参照)。
- アクティビティー・インプリメンテーションをデバッグしたい場合、作業項目を割り当てられるユーザー用に MQSeries Workflow プログラム実行エージェントを開始しておかなければならない。

  注: プロセス・モデルのアクティビティーをインプリメントするプログラムは、MQSeries Workflow ワークフロー・マネージャーで使用できるよう登録する必要があります。デバッグしたいプログラムは、選択したオペレーティング・システム用に登録されていて、しかも登録名とパス情報を使って検出できることを確認してください。

テスト・データベースの作成

テスト・データベースを作成するには、そのようなデータベースを作成する以外に次のことも必要です。

- トポロジー・データの追加 (データベースのブートストラップの方法については、*IBM MQSeries Workflow: インストールの手引き* を参照)。
- テスト・データの追加 (*IBM MQSeries Workflow: 定義機能の開始*、『実行機能搬出および搬入ユーティリティーの使用』の章を参照)。

クライアント・アプリケーションのデバッグ

クライアント・アプリケーションをテストするには、任意のデバッガーの制御のもとでそのアプリケーションを開始します。マルチスレッドのアプリケーションの場合、ご自分の責任のもとにスレッドを正しく同期化してください。

注: MQSeries Workflow アクションに関する詳しい情報を入手するために MQSeries Workflow のトレース機能を使ったり、問題判別のために構成検査プログラムを使うことも検討することができます。

アクティビティー・インプリメンテーションまたはサポート・ツールのデバッグ

アクティビティー・インプリメンテーションおよびサポート・ツールは、MQSeries Workflow プログラム実行エージェントの制御のもとで実行されるため、デバッグを可能にするには特別な注意が必要です。

Workflow バージョン 2 と同様、独自に作成した FDL を変更して、デバッガーをプログラム・インプリメンテーションとして登録するオプションがあります。これは Java 言語で使用できるオプションです。

C、C++、および ActiveX の場合、MQSeries Workflow は未変更の FDL の使用をサポートします。 MQSeries Workflow には、デバッグを可能にするための 2 つの環境変数があります。 *FMC_PEA_DEBUGGER_NAME* は、デバッガーの名前を指定するのに使います。デバッガーのファイル名のフルパスを指定することができますが、PATH ステートメントによってデバッガーにアクセスできるようにすることもできます。そして *FMC_PEA_DEBUG_ACT_IMPL* を ″YES″ に設定すると、プログラム実行エージェントは、アクティビティー・インプリメンテーションではなく指定したデバッガーを開始します。たとえば、

FMC_PEA_DEBUGGER_NAME = IDEBUG.EXE

FMC_PEA_DEBUG_ACT_IMPL = YES

プログラム実行エージェントは、適切な環境において別個のオペレーティング・システム・プロセスでデバッガーを開始します。デバッガー・プロセスのプロセス環境はプログラム実行エージェントによって設定され、アクティビティー・インプリメンテーションに継承されるので、ご使用のアクティビティー・インプリメンテーションはプログラム実行エージェントによっても引き続き認識され、API 要求を出す許可が付与されています。

実行可能プログラムのデバッグ時にプログラム実行エージェントは、呼び出したデバッガーにアクティビティー・インプリメンテーションの名前とそのパラメーターを渡します。

ダイナミック・リンク・ライブラリーのデバッグ時にプログラム実行エージェントは、呼び出したデバッガーに、DLL をロードするプログラムの名前とアクティビティー・インプリメンテーションのパラメーターを渡します。

注: DLL は、**隔離モード**で実行するよう登録されている必要があります。

隔離モードで DLL をロードする MQSeries Workflow プログラムは *FMCXDLL.EXE* です。これによって、*FmcDebugDllV2* と *FmcDebugDllV3* の 2 つの関数が提供されます。 *FmcDebugDllV3* は MQSeries Workflow バージョン 3 の DLL を、*FmcDebugDllV2* は FlowMark バージョン 2 の互換 DLL をデバッグするのに使います。

これらは、バージョン 2 またはバージョン 3 の DLL のエントリー・ポイントを呼び出します。これらの関数にブレークポイントを設定すると、デバッガーは DLL のエントリー・ポイントが呼び出される前に停止するので、アクティビティー・インプリメンテーションをステップ実行することができます。

注: Microsoft デバッガー msdev.exe は、これらのエントリー・ポイントを処理できません。したがって、DLL をデバッグする必要がある場合には、以下のステートメントをコードに追加してください。

```
#if defined(_MSC_VER) && defined(_DEBUG)
DebugBreak();
#endif
```

プログラム実行エージェントが msdev.exe を開始すると、msdev ウィンドウで FMCXDLL.EXE が実行されます。コード内の DebugBreak() ステートメントによって、デバッガーが DLL のデバッグを開始できるようになります。 DebugBreak() は msdev.exe の制御下でのみ機能し、そうでない場合は処理できない例外を作成することに注意してください。

隔離 DLL をデバッグする場合は、次のことに注意してください。

1. 非隔離 DLL にのみ特有の問題が生じることがありますが、隔離 DLL のデバッグ中にそれに気付くことはありません。たとえば、再入可能 DLL の複数のインスタンスを並行して実行する場合を考えてみましょう。非隔離モードでは、DLL は 1 回だけロードされ、複数のスレッドでプログラム実行エージェントのコンテキストにおいて実行されます。隔離モードでは、DLL は複数の FMCXDLL プロセスのコンテキストで実行されます。 DLL のデータ・セグメントはプロセスごとに固有のものですが、1 つのプロセス内のそれぞれのスレッドによって共用されるため、デバッガーなしではその作用がわからないかもしれません。
2. DLL または DLL の中のエントリー・ポイントが見つからない場合、デバッガー・ウィンドウは表示されず、アクティビティー・インプリメンテーションの状態は *InError* になります。そのような問題を判別するには、MQSeries Workflow のトレース機能を使います。

3. 作成する DLL では、サポートされている (標準の) 呼び出し規則とシグニチャーを使うようにしてください。 MQSeries Workflow では、*FMC_APIENTRY* 呼び出し規則を定義しています (fmcjcglo.h を参照)。ご使用の DLL が FlowMark バージョン 2 互換の DLL として登録されているなら、それにはパラメーターとして実行セッション識別子 (バージョン 3 では呼び出し先プログラム識別子)、および追加パラメーターを指すポインターの 2 つが渡されます。 MQSeries Workflow バージョン 3 の DLL には、追加パラメーター引き数だけが渡されます。バージョン 3 のプログラム実行エージェントは、独自にプログラム識別子を判別できるからです。

第5章 クライアント / サーバー通信とデータ・アクセス・モデル

MQSeries Workflow サーバーからアクションを要求する場合やアクションの結果を監視する場合には、次のようにします。

- 同期プロトコルを使用して、アクションを要求し、アクションを呼び出すのに使用したオブジェクトの変更個所を表示します。
- 同期プロトコルを使用して、作成または変更したデータをプルします。
- 作成または変更したオブジェクトのうち、サーバーがプッシュしたものについての非送信請求情報を受信します。
- 非同期プロトコルを使用して、アクションを要求し、その後の結果を表示します。現在のところ非同期モードでは、プロセス・インスタンスの実行のみサポートされています。

たとえば、プロセス・インスタンス・オブジェクトを開始するよう要求する場合は、次のようにします。

- 即時の結果として、プロセス・インスタンスの状態が更新されます。
- 新しく作成したオブジェクトを表示 (プル) するために作業項目を照会することができます。
- 送信 (プッシュ) されてくる新しい作業項目を自動的に受信することができます。

同期クライアント / サーバー通信

同期プロトコルを使用することは、MQSeries Workflow サーバーに要求を発行して、応答があるまで待機することを意味します。関数 / メソッドが関係するすべてのアクションはこの方式で実行されます。応答があるか、実行サービス・オブジェクトで設定したタイムアウトを超過するまで、アプリケーション (スレッド) はブロックされます。

注: XML メッセージ・インターフェースでは、同期通信はサポートされていません。

非同期クライアント / サーバー通信

非同期プロトコルを使用することは、MQSeries Workflow サーバーに要求を発行しますが、応答があるまで待機しないことを意味します。ExecuteProcessInstanceAsync() 関数 / メソッドはこの方式で実行されます。アプリケーション (スレッド) はブロックされず、後で応答を受け取ります。

非同期でアクションを発行した場合は、MQSeries Workflow がその要求を受け入れたかどうかを示す確認応答を受け取ることになります。特定の応答を受け取るために使用できる相関識別 (ID) も受け取ることがあります。受け取った応答を相関させるためにユーザー・コンテキストを指定することができます。

たとえば、非同期でプロセス・インスタンスが実行されるように要求した場合は、次のようになります。

- 要求直後の結果としては、要求が受け入れられたかどうかが通知されます。
- 相関 ID を保持するためのバッファーを指定した場合、その特定の応答を待機するために Receive() 呼び出しで使用できる ID が与えられます。
- ユーザー・コンテキストを指定した場合、そのコンテキストが応答の一部として戻されます。ユーザー・コンテキストはユーザー固有の相関関係に使用できます。

注: 非同期での通信は、C++ と C 言語でサポートされています。メッセージ・ベースの要求は、すべて非同期で実行されます。

プッシュ・データのアクセス・モデル

MQSeries Workflow サーバーがプッシュした非送信請求情報を受信することは、新しいオブジェクトまたは変更を加えたオブジェクトについての通知が自動的になされるように通信を設定することを意味します。

注: プッシュ・データ・アクセス・モデルは、Java と XML メッセージ・インターフェースではサポートされていません。

MQSeries Workflow サーバーがプッシュした情報を取得する方法は、以下のとおりです。

1. サーバーはデータの送信要求を受け入れなければなりません。つまり、
   - 考慮しているプロセス・インスタンスの設定では、*REFRESH_POLICY PUSH* を指定していなければなりません。この設定は、システム・グループを通じてドメイン・レベルからシステムおよびプロセス・テンプレートへ継承されます。各指定は、それより下位のレベルで上書きすることができます。

- ユーザーは、情報を受信できるようにするために、*Present* セッション・モードか *PresentHere* セッション・モードでログオンしなければなりません。

2. アプリケーションはプッシュされたデータを受信するために関数 / メソッドを使用しなければなりません。

これらの前提条件を満たしていれば、MQSeries Workflow 実行サーバーは変更を作業項目にプッシュしたり、通知をアイテムの所有者にプッシュしたりします。

1. アイテムの作成時の状態。
2. アイテムの削除時の状態。
3. アイテムの 1 次プロパティーが変更されているかどうか。 1 次プロパティーの定義については、96ページの『アクセス関数 / メソッド』を参照してください。

しかし、アクションの結果として一時オブジェクトがすでに関連データによって更新されているため、アクションの呼び出し側に上記の情報は送信されません。

使用不可になっている作業項目の変更はプッシュされません。そのような作業項目については、削除だけがプッシュされます。

例:

プロセス・インスタンスが中断され、かつそのリフレッシュ・ポリシーがプッシュである場合、MQSeries Workflow 実行サーバーは、現在 present でログオンしているアイテムのうち使用不可になっていないアイテムのすべての所有者に情報を送信します。

プロセス・インスタンスの記述が変更され、かつそのリフレッシュ・ポリシーがプッシュである場合、MQSeries Workflow 実行サーバーは、現在 present でログオンしているプロセス・インスタンス通知のすべての所有者に情報を送信します。

作業項目の所有者によって作業項目がユーザー N に転送され、かつそれに関連したプロセス・インスタンスのリフレッシュ・ポリシーがプッシュである場合、ユーザー N が present でログオンしていれば、 MQSeries Workflow 実行サーバーは情報をユーザー N に送信します。アクションの要求者としての作業項目の所有者は、情報をそれ以上取得することができません。

注:

1. フィルター基準およびソート順はアプリケーション側で処理します。影響を受けるワークリストについての指示は、クライアントにプッシュされません。
2. ActiveX API には、ワークリストに関してプッシュ 処理オプションがあります。このオプションは、プッシュした情報をそのリストに含めるかどうかを制御します。このオプションをセットした場合、フィルターに掛かるかどうかに関係なく、すべてのアイテム情報がリストに含められます。この情報はリストの先頭に含められるため、ソートの影響も受けません。

情報の受信

C および C++ 言語において実行サービスオブジェクトは、MQSeries Workflow 実行サーバーがプッシュした情報 (実行データ) を好きなときに受信するための手段です。 Receive() 呼び出しは、なんらかの情報が受信されるか指定されたタイムアウト値に到達するまで、呼び出しアプリケーションをブロックします。それで、アプリケーション全体がブロックされてしまうのを防ぐために、多くの場合、アプリケーションはデータを受信するためのスレッドをいくつかに分けて開始します。

タイムアウト値が -1 なら、それは無限に待機するよう指定することになります。このタイムアウト値を指定した場合、データの受信を停止してからでないとアプリケーションを終了できないことに注意してください。データ受信の終了を通知できるようにしたい場合は、TerminateReceive() 関数 / メソッドを使用することにより、アプリケーションの受信部分に終了指示を送信できます。

注:

1. Logoff() 呼び出しによって実行サーバーとのセッションは終了しますが、Receive() 呼び出しは Logoff() 呼び出しの後でも有効です。しかし、ログオフが実行されると、実行サーバーは情報のプッシュを停止してしまいます。受け取り側アプリケーション・スレッドに TerminateReceive() を送信しなかった場合は、他の方法でそのスレッドを終了しなければなりません。TerminateReceive() を呼び出すことができるのは、セッションが存在している場合だけです。
2. 情報が受信されないでクライアント入力キューにとどまっていると、そのような "無活動" メッセージを取り除くため、MQSeries(R) の有効期限メカニズムが働きます。クライアント・メッセージの満了時刻は、MQSeries Workflow 用に構成することができます。

データの受信時には、相関 ID を指定することによって、どの情報を読み取るべきかを明示することができます。相関 ID が指定されていないか、またはそれが FMCJ_NO_CORRELID を指していない場合は、送達されるデータすべてが受信されます。相関識別 (ID) は受信が成功した結果として設定されることに注意してください。

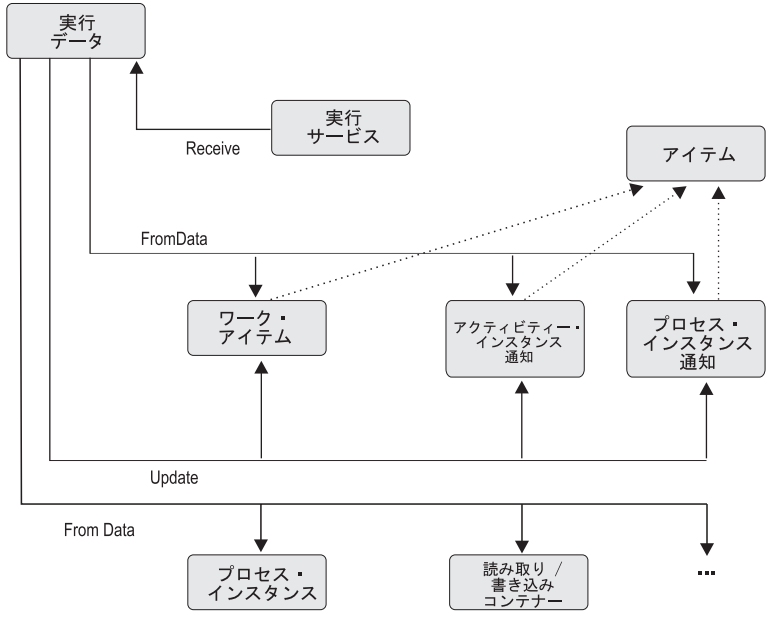


図 1. MQSeries Workflow サーバーが送信するデータの処理. 凡例: ⟶ 継承 (C++); ⟶ アクセスの提供

一度実行データが受信されたなら、その型を判別したり対応するアクションを呼び出したりすることができるようになります。たとえば、作業項目の作成が指示されているなら、実行データから作業項目への変換を要求することができます。作業項目の変更を指示する際には、作業項目の永続オブジェクト ID を要求して、該当する作業項目を更新することもできます。

ExecuteProcessInstanceAsync() 要求への応答を受け取ると、作成および実行されたプロセス・インスタンスを分析することが可能になります。たとえば、応答の状態を使用してプロセス・インスタンスが正常に実行されたかどうかを判別することができます。そして、その出力コンテナーを読み取ることができます。エラーが発生した場合に、エラー記述を調べることもできます。

注: ActiveX では、データが変更されたことをアプリケーションに通知するためにイベント・メカニズムを使用します。 803ページの『イベント』を参照してください

第6章 MQSeries Workflow セッション

MQSeries Workflow サーバーと通信するには、ユーザーとサーバーとの間にセッションが設定されていなければなりません。サーバーは、明示的に識別されるか (システム・グループまたはシステム・グループのシステム)、またはユーザーのプロファイルから取られます。ユーザー・プロファイル中に情報が見つからない場合、構成プロファイルが読み取られます。

注: XML メッセージ・インターフェースを使用するのに認証は必要ありません。つまり、セッションを確立する必要はありません。

セッションは、ログオンすることで確立されます。それ以降、サーバーにサービスを要求できるようになります。それに応じて、ログオンするユーザーとサーバーとの間のセッションを表すサービス・オブジェクトがセットアップされます。

ログオンするには、選択されたシステム上で管理サーバーが起動されていて稼働していなければなりません。管理サーバーがセッションを管理したりユーザーの認証を検査したりするからです。さらにこのサーバーは、エラー・ログへの重大エラーの書き込みも実行します。

取り出された (または作成された) すべてのオブジェクトは、照会された (作成された) セッションに属します。それらのオブジェクトにはセッション識別子が付けられ、それ以後そのオブジェクトに対するアクションは、ログオン・ユーザーの許可によりその同じセッションで実行されます。

スレッドは MQSeries Workflow では明示的にはサポートされていませんが (オブジェクトはスレッド・セーフではありません)、MQSeries Workflow でスレッドを使えないというわけではありません。セッションは複数のスレッドにまたがることもあります。ただし、オブジェクトの同期には注意しなければなりません。また、Java 以外のすべての言語では、API リソースが正しく管理されるよう、各スレッドで Connect() および Disconnect() の関数 / メソッドを使わなければなりません。

単一または複数のアプリケーション・プログラムで複数のサービス・オブジェクトを割り振ることができ、別々のユーザーまたは同一のユーザーと並行してログオンすることができます。それぞれのセッションは、サービス・オブジェクトによって分離状態を保ちます。したがって、サービス・オブジェクトは、

それぞれが 1 つのセッションを表します。同じサービス・オブジェクトを介して 2 度目にログオンを要求した場合、それが別のユーザーからのものであれば拒否されます。同じユーザーからのものであれば受諾されますが、ログオンは繰り返されません。ログオン要求はすでに正常に完了しているからです。

セッションは、*省略時解釈* モードまたは*在席* モードで実行できます。在席 (present) モードの場合、自動的に開始されるアクティビティー・インスタンスのスケジュールは自分の都合に合わせて立てたり、MQSeries Workflow サーバーがプッシュした情報を受信したりすることができます。在席セッションは、各ユーザーにつき 1 つだけ確立できます。

サービス・オブジェクトには、タイムアウト値を設定することができます。これは、サーバーからの応答をアプリケーションが待つ時間です。つまり、アプリケーションは最大限その時間中はブロックされます。タイムアウトはミリ秒数で指定します。 -1 の値は無限のタイムアウト値を表します。タイムアウト値はいつでも変更できます。

注: MQSeries Workflow は IBM MQSeries の通信機構を使用します。アプリケーションで独自のシグナル・ハンドラーをセットアップする場合は、*MQSeries アプリケーション・プログラミングの手引き* の中の特に *UNIX 信号処理* の章を参照して、MQSeries での制限についての情報を入手してください。

第7章 データの照会

MQSeries Workflow サーバーからデータを照会するには、基本的に次の 3 通りの方法があります。

- サービス・オブジェクトによる照会。これは、許可されているすべてのオブジェクトを戻します。クライアントに戻されるオブジェクトの数は、フィルターとしきい値によって制限することができます。 (ActiveX ではサポートされていません。)
- 永続リスト定義を使用する照会。これは、リスト定義によって決まるすべてのオブジェクトを戻します。
- 個々の要求。たとえば、ユーザー設定の要求や特定のオブジェクトのリフレッシュ要求。

注: XML メッセージ・インターフェースでは、データの照会はサポートされていません。

永続リスト

永続リストは、同一タイプのオブジェクトの集合を表します。さらに、このリストによってアクセスできるすべてのオブジェクトは特性が同じです。リストは、公開して使用することができます (すべてのユーザーが見ることができる)。またはプライベートに使用することもできます (所有者があってその所有者しか見ることができない)。

このリストに含まれているオブジェクトの特性は、いわゆる*フィルター基準* によって指定されます。指定されるフィルター基準と、照会を発行するユーザーの許可とによって、リストの内容が決まります。したがって、その内容自体は永続的に保管されるのではなく、照会要求が出された時点で決まります。つまり、パブリック・リストの結果は、照会を適用するユーザーによって異なります。

照会の結果としてサーバーからクライアントに送られるオブジェクトの数は、*しきい値* を指定することで制限することができます。しきい値は、*ソート基準* が適用された後で使われます。

リストは、プロセス・テンプレート・リスト、プロセス・インスタンス・リスト、またはワークリストのいずれかです。

フィルター、ソート基準、およびしきい値の使用

フィルターは、基準を指定するための文字列で、フィルター構文図に定められている規則に従ったものでなければなりません。正確な構文については、該当する関数 / メソッドを参照してください。基準の例を以下にいくつか示します。

```
"NAME = 'MyProcessInstance'"
"NAME LIKE 'My*Ins?ance'"
"LAST_MODIFICATION_TIME > '1998-2-19 11:38:0'"
"STATE IN (READY,RUNNING)"
```

ソート基準は、基準を指定するための文字列で、ソート基準構文図に定められている規則に従ったものでなければなりません。正確な構文については、該当する関数 / メソッドを参照してください。基準の例を以下にいくつか示します。

```
"NAME ASC"
"NAME ASC, LAST_MODIFICATION_TIME DESC"
```

オブジェクトはサーバーでソートされることに注意してください。つまり、サーバーのコード・ページでソート順序が決まります。

しきい値は、クライアントに戻されるオブジェクトの最大数を指定します。しきい値はオブジェクトのソート後に適用されます。

コレクションの処理

オブジェクト・セットの照会結果は、いわゆるオブジェクトのベクトル (C または C++ 言語) またはオブジェクトの配列 (ActiveX および Java 言語) です。

ベクトルは、呼び出し元から提供され、MQSeries Workflow API によってデータが入れられます。ベクトル要素 (オブジェクト) の所有権はそのベクトルに帰着します。それは、ベクトルが削除されると自動的に削除されます。

戻されたすべてのオブジェクトが、提供されるベクトルに付加されます。現在のオブジェクトだけを読み取りたい場合、まずベクトルをクリアした後で照会メソッドを呼び出さなければなりません。つまり、C++ API ではベクトルのすべての要素を消去する必要があります。これは、C 言語 API でベクトル・ハ

ンドルを 0 に設定する必要があることを意味します。[1] 0 に初期化しない場合、新しく照会するオブジェクトが追加できるよう、ベクトル・ハンドルは適切な種類のベクトル・オブジェクトを参照している**必要があります**。つまり、0 以外のハンドルは、既存とみなされているベクトルにアクセスするときに C 言語で使用します。

C 言語の場合、照会結果はオブジェクト・セットに初期化されたベクトル・ハンドルになります。 0 に設定されたハンドルが渡された場合は、新しいオブジェクトにより拡張された既存のベクトルです。オブジェクトにアクセスするための特別なベクトル・アクセス関数が用意されています (下記参照)。ベクトル要素が読み取られると、それは単独のオブジェクトとなるため、使わなくなったものは削除しなければなりません。そのオブジェクトに対するすべての操作は、そのオブジェクトだけを参照するものであり、オブジェクトのコピー元であるベクトル要素には何の影響もありません。たとえば、Refresh() はそのオブジェクトだけを変更し、ベクトル内のオリジナル・コピーは変更しません。したがって、その後でベクトル全体にわたって反復すると、どの要素も未変更のままです。

C++ 言語の場合、照会結果は vector<class T> のインスタンスになります。オブジェクトへは、該当するベクトル・メソッドによってアクセスします。 STL の資料を参照してください。ベクトル要素を読む場合には、そのオブジェクトへの (const または非 const の) 参照が戻されます。つまり、そのオブジェクトを変更すると、実際にベクトル要素を変更することになります。その後でベクトル全体にわたって反復すると、要素が変更されています。

配列は MQSeries Workflow API により提供され、データが入れられます。配列要素 (オブジェクト) の所有権はその配列に帰着します。

C 言語のベクトル

ここで、さまざまなベクトル・アクセス関数について説明します。これらのすべての関数は、別々のオブジェクトに関するものではありますが、よく似ており要件も類似しているからです。これらの関数はすべて API によってローカルに処理されます。つまり、サーバーと通信することはありません。実行するのに、サーバーへの接続や特定の許可は必要ありません。

1. 新しいベクトル・ハンドルを宣言するか、既存のベクトル・オブジェクトを再利用の前に割り振り解除してください。

戻りコード

C 言語の関数または結果オブジェクトは、次に示すコードを戻します。括弧内の数字はそれぞれの整数値です。

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスを受け取ることになっているにもかかわらず、0 が渡されました。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_NO_MORE_DATA(804)
ベクトルに要素が入っていないか、またはそれ以上要素が入っていません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

ベクトル・アクセス関数を使うと、以下に示す操作が可能です。 **'Xxx' は何らかの有効範囲を表します。** たとえば、FmcjProcessInstanceVectorFirstElement() は FmcjXxxVectorFirstElement() によって表されるという具合です。

FmcjXxxVectorDeallocate

指定された一時ベクトル・オブジェクト用に予約されている記憶域をアプリケーションが割り振り解除するのに使われます。含まれているすべての要素も割り振り解除されます。

C 言語のハンドルは 0 に設定されるので、もう使えなくなります。

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjXxxVectorDeallocate(FmcjXxxVectorHandle * handle)
```

パラメーター

handle 入出力。割り振り解除するベクトルのハンドルのアドレス。

FmcjXxxVectorFirstElement

ベクトルの第 1 要素を戻します。要素は単独のオブジェクトとなるので、使われなくなったら削除しなければなりません。ベクトルの次の要素に移ります。

ベクトルが空の場合、またはエラーが発生した場合は 0 が戻されます。

C 言語のシグニチャー

```
FmcjXxxHandle FMC_APIENTRY FmcjXxxVectorFirstElement(
              FmcjXxxVectorHandle hdlVector )
```

パラメーター

hdlVector　　入力。照会するベクトルのハンドル。

戻り値のデータ型

FmcjXxxHandle

ベクトルの第 1 要素のハンドルまたは 0。

FmcjXxxVectorNextElement

現行のベクトル位置にあるベクトル要素を戻します。初期ベクトル位置は第 1 要素です。要素は単独のオブジェクトとなるので、使われなくなったら削除しなければなりません。ベクトルの次の要素に移ります。

ベクトルが空であるか、ベクトル内にもう要素がないか、またはエラーが発生した場合は、0 が戻されます。

C 言語のシグニチャー

```
FmcjXxxHandle FMC_APIENTRY FmcjXxxVectorNextElement(
              FmcjXxxVectorHandle hdlVector )
```

パラメーター

hdlVector　　入力。照会するベクトルのハンドル。

戻り値のデータ型

FmcjXxxHandle

現在位置のベクトル要素のハンドルまたは 0。

FmcjXxxVectorSize

ベクトル内の要素の数を戻します。

C 言語のシグニチャー

```
unsigned long FMC_APIENTRY FmcjXxxVectorSize(
              FmcjXxxVectorHandle hdlVector )
```

パラメーター

hdlVector 入力。照会するベクトルのハンドル。

戻り値のデータ型

unsigned long

ベクトル内の要素の数。

例

以下に、ベクトルを読み取る方法についての C 言語の例を示します。最初に第 1 要素の呼び出しを使っていますが、最初から次の要素 (NextElement) の呼び出しを使うことも可能です。

First/NextElement() 呼び出しの使用

```
#include <stdio.h>
#include <fmcjcrun.h>
int main()
{
   APIRET                           rc;
   FmcjExecutionServiceHandle       service      = 0;
   FmcjProcessInstanceVectorHandle hdlVector    = 0;
   FmcjProcessInstanceHandle        hdlInstance  = 0;
   unsigned long                    i            = 0;
   unsigned long                    numElements  = 0;
   char                             tInfo[FMC_PROCESS_INSTANCE_NAME_LENGTH]="";


  FmcjGlobalConnect();
  FmcjExecutionServiceAllocate(&service);
  rc = FmcjExecutionServiceLogon( service,
                                  "ADMIN", "PASSWORD",
                                  Fmc_SM_Default, Fmc_SA_Reset
                                 );
  if ( rc != FMC_OK )
    return rc;
  printf("Logged on¥n");
```

```
  rc= FmcjExecutionServiceQueryProcessInstances(
                               service,
                               FmcjNoFilter,
                               FmcjNoSortCriteria,
                               FmcjNoThreshold,
                               &hdlVector  );
  if ( rc != FMC_OK )
     return rc;
  printf("Queried process instances¥n");


  hdlInstance= FmcjProcessInstanceVectorFirstElement(hdlVector);
  numElements= FmcjProcessInstanceVectorSize(hdlVector);
  printf("Instances in the vector:¥n");
  for( i=0; i< numElements; i++ )
  {
  printf("- name: %s¥n",
           FmcjProcessInstanceName(hdlInstance,tInfo,
                                   FMC_PROCESS_INSTANCE_NAME_LENGTH));
  FmcjProcessInstanceDeallocate(&hdlInstance);
  hdlInstance= FmcjProcessInstanceVectorNextElement(hdlVector) ;
  }
  FmcjProcessInstanceVectorDeallocate(&hdlVecor);


  FmcjExecutionServiceLogoff(service);
  printf("Logged off¥n");
  FmcjExecutionServiceDeallocate(&service);
   FmcjGlobalDisconnect();
   return FMC_OK;
}
```

NextElement() 呼び出しのみの使用

```
#include <stdio.h>
#include <fmcjcrun.h>
int main()
{
  APIRET                          rc;
  FmcjExecutionServiceHandle      service     = 0;
  FmcjProcessInstanceVectorHandle hdlVector   = 0;
  FmcjProcessInstanceHandle       hdlInstance = 0;
  char                            tInfo[FMC_PROCESS_INSTANCE_NAME_LENGTH]="";
```

```
  FmcjGlobalConnect();
  FmcjExecutionServiceAllocate(&service);
  rc = FmcjExecutionServiceLogon( service,
                                  "ADMIN", "PASSWORD",
                                  Fmc_SM_Default, Fmc_SA_Reset
                                );
  if ( rc != FMC_OK )
    return rc;
  printf("Logged on¥n");


  rc= FmcjExecutionServiceQueryProcessInstances(
                               service,
                               FmcjNoFilter,
                               FmcjNoSortCriteria,
                               FmcjNoThreshold,
                               &hdlVector  );
  if ( rc != FMC_OK )
    return rc;
  printf("Queried process instances¥n");


  printf("Instances in the vector:¥n");
  while (0 != (hdlInstance=FmcjProcessInstanceVectorNextElement(hdlVector)))
  {
    printf("- name: %s¥n",
           FmcjProcessInstanceName(hdlInstance,tInfo,
                                   FMC_PROCESS_INSTANCE_NAME_LENGTH));
    FmcjProcessInstanceDeallocate(&hdlInstance);
  }
   FmcjProcessInstanceVectorDeallocate(&hdlVector);


  FmcjExecutionServiceLogoff(service);
  printf("Logged off¥n");
  FmcjExecutionServiceDeallocate(&service);
  FmcjGlobalDisconnect();
  return FMC_OK;
}
```

ActiveX の配列

ActiveX の場合、オブジェクト・セットの照会結果は配列に保管されます。配列は、それぞれの ActiveX コントロールによって提供されます。配列の割り振りや削除はできません。

新しい照会を実行するたびに、配列に保管されている既存のオブジェクトがすべて削除され、新しいオブジェクトが追加されます。

すべての配列は、そこに含まれるオブジェクトの数およびオブジェクト自体を照会するための同じメソッドを提供します。

すべての配列インデックスは 0 (ゼロ) から始まります。つまり、有効なインデックス番号の範囲は 0 から GetSize()-1 までです。オブジェクトのインデックス番号は、照会を実行するたびに、ソート基準や戻されるオブジェクトの数に基づいて変わることがあるので、保管しないでください。

例外

次の例外が発生することがあります。

FMC_WRONG_INDEX(1501)
インデックスが配列の範囲を超えています。

Add

ContainerArray または ContainerElementArray に新しいオブジェクトを追加します。

シグニチャー

```
long Add()
```

ExecutionServiceArray に新しい実行サービスを追加します。

シグニチャー

```
long Add           ( BSTR system, BSTR systemGroup)
long AddDefault    ()
long AddSystemGroup( BSTR systemGroup )
```

指定した文字列を StringArray に追加します。

シグニチャー

```
long Add ( BSTR string )
```

パラメーター

string 入力。 StringArray に追加する文字列。

system 入力。実行サーバーが実行されているシステム。

systemGroup 入力。システムの属するシステム・グループ。

戻り値のデータ型

long 追加するオブジェクトの配列内でのインデックス。

GetAt

指定したインデックスにあるオブジェクトを戻します。

シグニチャー

```
Object GetAt ( long index )
```

パラメーター

index 入力。取り出すオブジェクトのインデックス。

戻り値のデータ型

Object 配列内の要素の型のオブジェクト。

GetSize

配列内の要素の数を戻します。

ActiveX のシグニチャー

```
long GetSize()
```

戻り値のデータ型

long 配列のカーディナリティー。

RemoveAll

StringArray からすべてのオブジェクトを除去します。

シグニチャー

```
void RemoveAll ( )
```

RemoveAt

指定したインデックス位置にあるオブジェクトを除去します。
ExecutionServiceArray、ContainerArray、ContainerElementArray、および StringArray で呼び出すことができます。

シグニチャー

```
void RemoveAt ( long index )
```

パラメーター

index インデックス。削除するオブジェクトのインデックス。

SetAt

指定したインデックス位置にある StringArray 要素の値を設定します。

シグニチャー

```
void SetAt( long index, BSTR string )
```

パラメーター

index 入力。設定する配列値のインデックス。

string 入力。設定する値。

イベント

New*Object*
新しい実行サービスが ExecutionServiceArray に追加されたこと、または新しいリスト・オブジェクトが ProcessInstanceListArray、ProcessTemplateListArray、または Worklist 配列に追加されたことを示します。

シグニチャー

```
void New Object( long index )
```

パラメーター

index 入力。配列内の新しい要素のインデックス。

*Object*Remove

ExecutionServiceArray から実行サービスが削除されたこと、またはリスト・オブジェクトが ProcessInstanceListArray、ProcessTemplateListArray、または Worklist 配列から削除されたことを示します。

シグニチャー

```
void Objectremove( long index )
```

パラメーター

index 入力。配列内の新しい要素のインデックス。

Java の配列

Java の場合、オブジェクト・セットの照会結果は配列に保管されます。配列は、対応する型の変数として宣言します。たとえば、

```
ProcessInstance[] processInstances;
```

新しい照会を実行するたびに、配列に保管されている既存のオブジェクトがすべて削除され、新しいオブジェクトが追加されます。

配列に含まれるオブジェクトの数は length 変数を使って調べることができます。たとえば、

```
processInstances.length
```

すべての配列インデックスは 0 (ゼロ) から始まります。つまり、有効なインデックス番号の範囲は 0 から length-1 までです。オブジェクトにアクセスするには、そのインデックス番号を指定します (たとえば、processInstances[0])。オブジェクトのインデックス番号は、照会を実行するたびに、ソート基準や戻されるオブジェクトの数に基づいて変わることがあるので、保管しないでください。

第8章 コンテナーの処理

コンテナーは、*実行機能* のプロセス・テンプレート、プロセス・インスタンス、作業項目、アクティビティー・インプリメンテーション、またはサポート・ツールの入力データまたは出力データを表します。各コンテナーは、*データ構造体* によって定義されます。コンテナーはそのデータ構造体の型であると宣言されます。

データ構造 / コンテナーの型

データ構造体は、その名前によって一意に識別され、順序の指定されたデータ・メンバー のリストを内容とします。実行機能では、クライアントとサーバーとの間でやりとりされる 32KB のストリームになることが可能です。

データ構造体と、入力コンテナーまたは出力コンテナーとしてのその使用方法は、モデル定義において定義されます。 MQSeries Workflow には DEFAULT_DATA_STRUCTURE という名前の特別なデータ構造体があり、これにはインストール時にはユーザー定義のデータ・メンバーは含まれていません。 DEFAULT_DATA_STRUCTURE は削除できませんが、モデル定義において拡張することはできます。

データ・メンバー / コンテナー要素

データ構造体のデータ・メンバーには名前があり、*データ型* が指定されています。データ型は、基本型、STRING、LONG、BINARY、または FLOAT、あるいは別のデータ構造体です。データ・メンバーのデータ型としてデータ構造体を使うことにより (ネスト)、データ・メンバーを再帰的に定義できます。

データ・メンバーによって 1 つの 1 次元配列を表すことができます。データ・メンバーが配列を表す場合、その配列内の要素数は括弧 () で囲んで指定します。

1 つのデータ構造体に対して、512 個までのユーザー定義のデータ・メンバーが可能です。データ・メンバーの配列を表すデータ・メンバーの場合は、その中に要素として含むデータ・メンバーの数になります。

データ・メンバーは、コンテナー内でその完全修飾名を使って指定されます。データ・メンバーの完全修飾名[2] はドット表記の名前です。ネストされたデータ・メンバーの階層は左から右へと表記され、その名前はドットで区切ります。

データ・メンバーで実際にデータ・メンバーの配列を指定する場合、特定のデータ・メンバーのインデックス番号は大括弧 ([n]) または小括弧 ((n)) で囲んで指定します。

データ構造体がコンテナーの型を表す場合、そのデータ・メンバー (階層の第 1 レベル) はコンテナー*要素* とも呼ばれます。それらは、コンテナーの*構造*メンバーを定義します。コンテナー要素 (階層の n 番目のレベル) のデータ型がデータ構造体の場合 (ネスト)、そのコンテナー要素にもコンテナー要素または構造メンバーが含まれます。

基本データ型のコンテナー要素は、コンテナーのリーフ *(葉)* とも呼ばれます。これらは、値を入れることのできるメンバーです。つまり、その値を問い合わせたり新しい値に設定したりできます。

たとえば、データ構造体 PERSON が入力コンテナーまたは出力コンテナーを記述し、その PERSON は次のように定義されているとします。

```
Name          STRING
Addr          ADDRESS
  Street      STRING
  POBOX       LONG(2)
```

PERSON には、Name および Addr という名前の 2 つの構造データ・メンバーが含まれています。 Name は基本データ型 STRING であり、Addr のデータ型は ADDRESS です。つまり、データ構造体 ADDRESS は、データ構造体 PERSON 内にネストされています。

この PERSON で記述されている入力コンテナーまたは出力コンテナーには、2 つのコンテナー要素 (Name および Addr という名前の構造メンバー) があります。 Addr はその構造がここで定義されています。コンテナー要素 Addr のコンテナー要素つまり構造メンバーは Street と POBOX です。

コンテナーのリーフ、つまり値を持つことの可能なコンテナー要素と、コンテナー内におけるその完全修飾名は、次のとおりです。

2. XML における完全修飾名はネスト階層によって表されます。

```
Name
Addr.Street
Addr.POBOX[0]
Addr.POBOX[1]
```

POBOX の配列のサイズは 2 であるため、有効なインデックス番号は 0 と 1 であることに注意してください。すべての配列インデックスは 0 から始まります。

また、完全修飾名の前にデータ構造体 PERSON の名前を付けないことにも注意してください。そのデータ構造体は、コンテナーの型を示すものです。

具体的な例については、817ページの『第9部 例およびシナリオ』を参照してください。

XML メッセージ・インターフェース

XML メッセージ・インターフェースの場合、データ・メンバー (コンテナー要素) は次のように表されます。

- データ・メンバー名は XML 要素名で表されます。
- ネストされたデータ構造はその構造にしたがって XML 子要素に分解されます。つまり、完全修飾名にはドット表記がありません。
- 配列はその要素を列挙して表記されます。
- データ・メンバー型は XML 要素内容には含まれません。

たとえば、

```
<Name>
  <Addr>
    <Street></Street>
    <POBOX></POBOX>
    <POBOX></POBOX>
  </Addr>
</Name>
```

詳しくは、216ページの『コンテナー・データ』を参照してください。

事前定義データ・メンバー

すべてのコンテナーは、MQSeries Workflow で事前定義されているデータ・メンバーを自動的に指定します。それらには、アクティビティーまたはプロセスの操作特性に関連付けられた値を入れることができます。事前定義データ・メンバーは、定義担当者によって定義される必要のない、自動的に使用可能になるデータ・メンバーです。これには、コンテナー API によってアクセスできます。その名前は、予約文字の "_" で始まります。

事前定義データ・メンバーの値は、
- アクティビティー終了基準の評価に使用できます。
- アクティビティー・インプリメンテーションまたはサポート・ツールからアクセスできます。
- それ以降のアクティビティーの操作特性を変更するために動的に設定できます。

事前定義データ・メンバーは、定義担当者に柔軟性を提供します。プロセスまたはアクティビティーの操作特性に関する決定は、実行機能において下されます。また、アクティビティー・インプリメンテーションおよびサポート・ツールにとっては、API 関数 / メソッドを使用して操作特性を取得するための手段になります。

事前定義データ・メンバーの集合として、次のものがあります。
- 固定データ・メンバー
- プロセス情報データ・メンバー
- アクティビティー情報データ・メンバー

固定データ・メンバーは、現行アクティビティー・インスタンスについての情報を提供します。これは、API 関数 / メソッドを使っては設定できません。例外は _RC データ・メンバーですが、これはそれ以外の方法ではプログラムが戻りコードを指定できない場合に限って設定します。

プロセス情報およびアクティビティー情報のデータ・メンバーは、プロセスまたはアクティビティーの操作特性に関連付けられます。これらのメンバーは、ユーザー定義のデータ・メンバーと同じような働きをします。つまり、他のユーザー定義データ・メンバーの値と同じように、プロセス・インスタンスまたはアクティビティー・インスタンスの特定の操作特性の値にアクセスしたり変更したりできます。

以下に、それぞれの事前定義データ・メンバーの完全修飾名と簡単な説明を示します。

事前定義データ・メンバーの配列はありません。

固定データ・メンバー

固定データ・メンバー _ACTIVITY、_PROCESS、および _PROCESS_MODEL は、API 関数 / メソッドを使って設定することはできません。その値は、API 関数 / メソッドを使って読み取ることはできます。固定データ・メンバー _RC は出力コンテナーでは使用可能ですが、使用しているコンパイラーがプログラム出口コードをサポートしていない場合にのみ設定してください。

_ACTIVITY

このデータ・メンバーには、対象となるアクティビティー・インスタンスの完全修飾名が入れられます。このデータ・メンバーの値は、アクティビティー・インスタンスまたはそれに関連付けられている作業項目が開始されると自動的に設定されます。

データ型: STRING

_PROCESS

このデータ・メンバーには、関連するプロセス・インスタンスの名前が入れられます。このデータ・メンバーの値は、アクティビティー・インスタンスまたはそれに関連付けられている作業項目が開始されると自動的に設定されます。

データ型: STRING

_PROCESS_MODEL

このデータ・メンバーには、関連するプロセス・モデルの名前が入れられます。このデータ・メンバーの値は、アクティビティー・インスタンスまたはそれに関連付けられている作業項目が開始されると自動的に設定されます。

データ型: STRING

_RC このデータ・メンバーには、アクティビティー・インプリメンテーションの戻りコードが入れられます。多くの場合、これは終了条件および分岐条件を評価するために使用されます。これを入力コンテナーから読み取ることはできず、プログラムの終了時に自動的にアクティビティー・インプリメンテーションの終了コードに設定 (上書き) されます。

終了コードをサポートしないコンパイラーをご使用の場合は、コンテナー API を使ってその値を設定できます。

データ型: LONG

プロセス情報のデータ・メンバー

プロセス情報のデータ・メンバーは、プロセス・インスタンスのプロパティーを動的に指定するのに使われます。一般に、プロセス定義担当者は、プロセス・インスタンス・プロパティーの設定値をどこから取得するかを選択することができます。

- 最上位レベルのプロセス・インスタンスから値を継承することができます。
- 入力コンテナーのプロセス情報データ・メンバーから値を獲得することができます。そしてその値を省略時値として設定するか、またはプロセス・インスタンスの開始時に入力コンテナーの中で指定します。

DATA_FROM_INPUT_CONTAINER 標識によって指定された場合、プロセス情報データ・メンバーの値はプロセス・インスタンスの開始時に MQSeries Workflow で読み取られます。プロセス情報データ・メンバーの値を設定しないなら、省略時値が使われます (後述の説明を参照)。

_PROCESS_INFO.Role

プロセス・インスタンスのアクティビティー・インスタンスに担当者が割り当てる役割は達成しなければなりません。

設定されるすべての役割は、アクティビティー・インスタンスに設定されている役割に対する追加基準になります。指定されたすべての役割のメンバーである人だけが適格となります。

役割が設定されていない場合に、アクティビティー・インスタンスに役割を指定しないなら、役割基準は適用されません。

データ型: STRING

_PROCESS_INFO.Organization

プロセス・インスタンスの作業項目を受け取るために、担当者が属していなければならない組織。この設定は、アクティビティー・インスタンスに組織が指定されていない場合にのみ使用されます。

組織が設定されておらず、アクティビティー・インスタンスにも組織が指定されていない場合の省略時値は、プロセス・インスタンスを開始する担当者の組織です。

データ型: STRING

_PROCESS_INFO.ProcessAdministrator

次の場合に通知を受ける担当者のユーザー ID。

- プロセス・インスタンスの有効期限が切れた場合。
- アクティビティー・インスタンスを実行するための基準を満たす担当者がいない場合。
- 通知を受けるのに有効な担当者が指定されていない場合。
- アクティビティー・インスタンスの期限切れを通知された担当者がアクションの許容時間を超過した場合、つまり、2 度目の通知が送られた場合。

これを設定しない場合の省略時のプロセス管理者は、プロセス・インスタンスを開始した担当者になります。

データ型: STRING

_PROCESS_INFO.Duration

プロセス・インスタンスを処理するのに費やせる期間を指定します。この値は秒数で指定します。

これを設定しない場合の省略時値は "Endless" (無制限) です。

データ型: LONG

アクティビティー情報のデータ・メンバー

アクティビティー情報のデータ・メンバーは、アクティビティー・インスタンスのプロパティーを動的に指定するのに使われます。一般に、プロセス定義担当者は、アクティビティー・インスタンス・プロパティーの設定値をどこから取得するかを選択することができます。

- 入力コンテナーのアクティビティー情報データ・メンバーから値を獲得することができます。そしてその値を省略時値として設定するか、または、アクティビティー・インスタンスあるいは対応する作業項目の開始時に入力コンテナーの中で指定します。

これを指定する場合、アクティビティー情報データ・メンバーの値は、アクティビティー・インスタンスのスケジューリング時に MQSeries Workflow によって読まれます。値を設定しない場合、省略時値が使われます (後述の説明を参照)。

次に示す標識は、アクティビティー情報データ・メンバーの読み取りを指定するものです。

- DONE_BY STAFF DEFINED_IN INPUT_CONTAINER
- NOTIFICATION DEFINED_IN INPUT_CONTAINER
- PRIORITY DEFINED_IN INPUT_CONTAINER

_ACTIVITY_INFO.Priority

アクティビティー・インスタンスの優先順位として割り当てられる数値。 MQSeries Workflow ではこの値をどんな意味にも解釈しません。あくまでクライアントで使用するためのものです。 0〜9 の任意の整数値を指定できます。指定した値が無効であるか、またはデータ・メンバーが設定されていない場合は、省略時値の 0 が使われます。

データ型: LONG

_ACTIVITY_INFO.MembersOfRoles

アクティビティー・インスタンス用の作業項目を受け取るために担当者が果たさなければならない 1 つまたは複数の役割。複数の役割をセミコロン (;) で区切って指定できます。

該当するデータ・メンバーに設定した 1 つまたは複数の役割は、プロセス・インスタンスに設定されている役割に対する追加基準になります。指定されたすべての役割のメンバーである人だけが適格となります。

これを設定しない場合、プロセス・インスタンスに指定された役割が使用されます。プロセス・インスタンスに対して役割が設定されていない場合に、アクティビティー・インスタンスに役割を指定しないなら、役割基準は適用されません。

注: _ACTIVITY_INFO.People データ・メンバーを使用して特定の担当者が設定されている場合、この指定は無視されます。

データ型: STRING

_ACTIVITY_INFO.CoordinatorOfRole

アクティビティー・インスタンス用の作業項目を受け取るために担当者がコーディネートしなければならない 1 つまたは複数の役割。コーディネートする複数の役割をセミコロン (;) で区切って指定できます。

作業項目を受け取るためには、プロセス・インスタンスとアクティビティー・インスタンスに指定されているすべての役割のメンバーであることに加えて、指定されたすべての役割のコーディネーターとして適格な担当者が割り当てられなければなりません。

これを設定しない場合、プロセス・インスタンスとアクティビティー・インスタンスで指定された役割だけが使用されます。メンバーとなる役割もコーディネーターとなる役割も指定されていないなら、役割基準は使用されません。

注: _ACTIVITY_INFO.People データ・メンバーを使用して特定の担当者が設定されている場合、この基準は無視されます。

データ型: STRING

_ACTIVITY_INFO.Organization

アクティビティー・インスタンスの作業項目を受け取るために、担当者が属していなければならない組織。

このデータ・メンバーを使用して組織が設定されている場合、プロセス・インスタンスに設定された組織は無視されます。

これを設定しない場合、プロセス・インスタンスに指定された組織が使用されます。組織が設定されておらず、プロセス・インスタンス・プロパティーにも組織が指定されていない場合の省略時値は、プロセス・インスタンスを開始する担当者の組織です。

注: _ACTIVITY_INFO.People データ・メンバーを使用して特定の担当者が設定されている場合、この基準は無視されます。

データ型: STRING

_ACTIVITY_INFO.OrganizationType

このデータ・メンバーは、アクティビティー・インスタンスの作業項目を下位組織の担当者に割り当てる必要があるかどうかを指示するのに使われます。

指定された組織とその全下位組織の担当者をすべて適格にするには、データ・メンバーの値を 0 に設定します。

指定された組織のメンバーと、下位組織の第 1 レベルの管理者のみを適格な担当者として限定するには、このデータ・メンバーを 0 以外の値に設定します。

これを設定しない場合の省略時値は 0 です。
_ACTIVITY_INFO.Organization データ・メンバーに組織が設定されていない場合、ここに設定された値は無視されます。

注: _ACTIVITY_INFO.People データ・メンバーを使用して特定の担当者が設定されている場合、この基準は無視されます。

データ型: long

_ACTIVITY_INFO.LowerLevel

担当者がアクティビティー・インスタンスの作業項目を受け取らねばならない下限レベル。設定できる値は 0〜9 です。省略時値は 0 です。

ここに指定されたレベルが上限レベルに指定された値よりも大きい場合、またはレベルが設定されていない場合、省略時値 0 が使用されます。

注: _ACTIVITY_INFO.People データ・メンバーを使用して特定の担当者が設定されている場合、この基準は無視されます。

データ型: LONG

_ACTIVITY_INFO.UpperLevel

担当者がアクティビティー・インスタンスの作業項目を受け取る上限レベル。設定できる値は 0〜9 です。省略時値は 9 です。

ここに指定されたレベルが下限レベルに指定された値よりも小さい場合、またはレベルが設定されていない場合、省略時値 9 が使用されます。

注: _ACTIVITY_INFO.People データ・メンバーを使用して特定の担当者が設定されている場合、この基準は無視されます。

データ型: LONG

_ACTIVITY_INFO.People

このデータ・メンバーは、アクティビティー・インスタンスの作業項目を受け取る担当者を個別に識別するのに使います。複数のエントリーをセミコロン (;) で区切って指定できます。

このデータ・メンバーを使用して担当者が識別される場合、_ACTIVITY_INFO.MembersOfRoles、_ACTIVITY_INFO.CoordinatorOfRole、_ACTIVITY_INFO.Organization、_ACTIVITY_INFO.OrganizationType、_ACTIVITY_INFO.LowerLevel、および _ACTIVITY_INFO.UpperLevel のデータ・メンバーに設定された値は無視されます。

値が設定されていない場合は、上記のデータ・メンバーに設定された値が使用されます。その値が設定されていない場合は、プロセス・インスタンスのスタッフ定義に設定された値が使用されます。

プロセス・インスタンスに値が設定されていない場合、プロセス開始者の組織およびそのすべての下位組織内の担当者が、アクティビティー・インスタンスの作業項目を受け取ります。

データ型: STRING

_ACTIVITY_INFO.PersonToNotify

アクティビティー・インスタンスの完了期限として指定された期限が切れてもアクティビティー・インスタンスが完了していない場合に通知する担当者を識別するために使用されます。

データ・メンバーで指定されるユーザー ID が無効である場合、またはデータ・メンバーが設定されていない場合、プロセス管理者に通知されます。

データ型: STRING

_ACTIVITY_INFO.Duration

アクティビティーを完了するための最大許容秒数を指定するために使用されます。

指定された期限が切れてもアクティビティーが完了しない場合、定義されている担当者に通知されます。

データ・メンバーで指定される値が無効である場合、またはデータ・メンバーが設定されていない場合、通知はされません。

データ型: LONG

_ACTIVITY_INFO.Duration2

アクティビティー・インスタンス通知に対する処置を実行するための最大許容秒数を指定するために使用されます。

指定された秒数が満了しても通知に対する処置がなされない場合は、プロセス管理者に通知されます。

データ・メンバーで指定される値が無効である場合、またはデータ・メンバーが設定されていない場合、通知はされません。

データ型: LONG

不明なコンテナーの構造の判別

不明なコンテナーの構造またはそのリーフ (あるいはその両方) を判別するための関数 / メソッドとして、さまざまなものが用意されています。それらの関数がコンテナーに適用されると、コンテナー要素の集合が戻されます。コンテナー要素の集合が利用可能になったなら、同様の関数 / メソッドを再帰的に適用して、ネストされた構造を順次下位レベルに下がっていくことができます。

注:

1. ActiveX のシグニチャーはオブジェクト定義言語 (ODL) で提供されています。たとえば、*BSTR* 型は、VisualBasic の型が実際には String であるストリングに使用されます。
2. XML メッセージ・インターフェースの場合、常にコンテナーの全体がメッセージ内に記述されます。したがって、アプリケーションは、メッセージ内のコンテナーを分析すればそのコンテナーの構造を判別できます。

リーフの判別

以下の関数 / メソッドは、コンテナー内のリーフの数を調べたりリーフ自体を取り出したりするためのものです。すべてのリーフを要求すると、ユーザー定義のリーフまたはそのリーフ数が提供されるだけでなく、MQSeries Workflow の事前定義データ・メンバーも提供されます。

ActiveX のシグニチャー

```
long LeafCount()
void Leaves( ContainerElementArray * leaves )
long AllLeafCount()
void AllLeaves( ContainerElementArray * leaves )
```

C 言語のシグニチャー

```
unsigned long FmcjContainerLeafCount( FmcjContainerHandle handle )
FmcjContainerElementVectorHandle
FmcjContainerLeaves( FmcjContainerHandle handle )
unsigned long FmcjContainerAllLeafCount( FmcjContainerHandle handle )
FmcjContainerElementVectorHandle
FmcjContainerAllLeaves( FmcjContainerHandle handle )
```

C++ 言語のシグニチャー

```
unsigned long LeafCount()
void Leaves( vector<FmcjContainerElement> const & leaves ) const
unsigned long AllLeafCount()
void AllLeaves( vector<FmcjContainerElement> const & leaves ) const
```

Java のシグニチャー

```
public abstract int leafCount() throws FmcException
public abstract ContainerElement[] leaves() throws FmcException
public abstract int allLeafCount() throws FmcException
public abstract ContainerElement[] allLeaves() throws FmcException
```

パラメーター

handle 入力。照会するコンテナーのハンドル。

leaves 入出力。データが入れられるコンテナー要素のベクトルまたは配列。

戻り値のデータ型

ContainerElement[]/FmcjContainerElementVectorHandle
リーフであるコンテナー要素。

long/unsigned long/int
ユーザー定義のリーフまたはすべてのリーフ (ユーザー定義のリーフおよび事前定義のリーフ) の数。

構造メンバーの判別

以下の関数 / メソッドは、コンテナー内の構造メンバーの数を調べたり構造メンバー自体を取り出したりするためのものです。

ActiveX のシグニチャー

```
long MemberCount()
void StructMembers( ContainerElementArray * members )
```

C 言語のシグニチャー

```
unsigned long FmcjContainerMemberCount( FmcjContainerHandle handle )
FmcjContainerElementVectorHandle
FmcjContainerStructMembers( FmcjContainerHandle handle )
```

C++ 言語のシグニチャー

```
unsigned long MemberCount()
void StructMembers( vector<FmcjContainerElement> const & members ) const
```

Java のシグニチャー

```
public abstract int memberCount() throws FmcException
public abstract ContainerElement[] structMembers() throws FmcException
```

パラメーター

handle 入力。照会するコンテナーのハンドル。

members 入出力。データが入れられるコンテナー要素のベクトルまたは配列。

戻り値のデータ型

ContainerElement[]/FmcjContainerElementVectorHandle
コンテナーの一部であるコンテナー要素。

long/unsigned long/int
コンテナー内の構造メンバーの数。

型の判別

次の関数 / メソッドは、コンテナーの型、つまり関連するデータ構造の名前を提供します。

ActiveX のシグニチャー

```
BSTR Type()
```

C 言語のシグニチャー

```
char * FmcjContainerType( FmcjContainerHandle handle,
                          char *              containerTypeBuffer,
                          unsigned long       bufferLength )
```

C++ 言語のシグニチャー

```
string Type()
```

Java のシグニチャー

```
public abstract String type() throws FmcException
```

パラメーター

bufferLength 入力。コンテナーの型を入れるバッファーの長さ。FMC_CONTAINER_TYPE_LENGTH バイト以上でなければなりません。

containerTypeBuffer
入出力。コンテナーの型を入れるバッファー。

handle 入力。照会するコンテナーのハンドル。

戻り値のデータ型

BSTR/char*/string/String
コンテナーの型。

コンテナー要素の分析

コンテナー要素へのアクセスが確立されたなら、そのプロパティーと名前の照会、およびそれがリーフ / 配列かまたは構造かの確認が行われます。コンテナーに対する前述の関数 / メソッドを繰り返し適用することにより、ネストされた構造を順次下位レベルに下がっていくことができます。

コンテナー要素の名前または型の判別

次の関数 / メソッドを使用すると、コンテナー要素の名前またはその型を調べることができます。

ActiveX のシグニチャー

```
BSTR Name()
BSTR FullName()
BSTR Type()
```

C 言語のシグニチャー

```
char* FmcjContainerElementName    (FmcjContainerElementHandle handle,
                                   char *                     buffer,
                                   unsigned long              bufferLength )
char* FmcjContainerElementFullName(FmcjContainerElementHandle handle,
                                   char *                     buffer,
                                   unsigned long              bufferLength )
char* FmcjContainerElementType    (FmcjContainerElementHandle handle,
                                   char *                     buffer,
                                   unsigned long              bufferLength )
```

C++ 言語のシグニチャー

```
string Name() const
string FullName() const
string Type() const
```

Java のシグニチャー

```
public abstract String name() throws FmcException
public abstract String fullName() throws FmcException
public abstract String type() throws FmcException
```

パラメーター

bufferLength 入力。データを入れるバッファーの長さ。

buffer 入出力。コンテナー要素の名前または型を入れるバッファー。

handle 入力。照会するコンテナー要素のハンドル。

戻り値のデータ型
BSTR/char*/string/String
コンテナーの名前または型。

コンテナー要素の構造プロパティーの判別

次の関数 / メソッドを実行することにより、対象となるコンテナー要素がリーフか、それ自体が構造か、それとも配列として表されているかを判別できます。

ActiveX のシグニチャー

```
boolean IsArray()
boolean IsLeaf()
boolean IsStruct()
```

C 言語のシグニチャー

```
bool FmcjContainerElementIsArray ( FmcjContainerElementHandle handle )
bool FmcjContainerElementIsLeaf  ( FmcjContainerElementHandle handle )
bool FmcjContainerElementIsStruct( FmcjContainerElementHandle handle )
```

C++ 言語のシグニチャー

```
bool IsArray () const
bool IsLeaf  () const
bool IsStruct() const
```

Java のシグニチャー

```
public abstract boolean isArray () throws FmcException
public abstract boolean isLeaf  () throws FmcException
public abstract boolean isStruct() throws FmcException
```

パラメーター
handle 入力。照会するコンテナー要素のハンドル。

戻り値のデータ型

boolean/bool コンテナー要素が配列、リーフ、または構造のどれかを表す標識。

コンテナー要素のリーフの判別

以下の関数 / メソッドは、コンテナー要素のリーフの数を調べたりリーフ自体を取り出したりするためのものです。

注: これらの関数 / メソッドがリーフそれ自体で呼び出されると、明らかにコンテナー要素がリーフであるため LeafCount() は 1 を戻しますが、Leaves() を照会してもそれ以上リーフは戻されません。

ActiveX のシグニチャー

```
long LeafCount()
void Leaves( ContainerElementArray * leaves )
```

C 言語のシグニチャー

```
unsigned long
FmcjContainerElementLeafCount( FmcjContainerElementHandle handle )
FmcjContainerElementVectorHandle
FmcjContainerElementLeaves( FmcjContainerElementHandle handle )
```

C++ 言語のシグニチャー

```
unsigned long LeafCount()
void Leaves( vector<FmcjContainerElement> const & leaves ) const
```

Java のシグニチャー

```
public abstract int leafCount() throws FmcException
public abstract ContainerElement[] leaves() throws FmcException
```

パラメーター

handle 入力。照会するコンテナーのハンドル。

leaves 入出力。データが入れられるコンテナー要素のベクトルまたは配列。

戻り値のデータ型

ContainerElement[]/FmcjContainerElementVectorHandle
リーフであるコンテナー要素。

long/unsigned long/int
ユーザー定義のリーフの数。

コンテナー要素の構造メンバーの判別

以下の関数 / メソッドは、コンテナー要素の構造メンバーの数を調べたり構造メンバー自体を取り出したりするためのものです。

ActiveX のシグニチャー

```
long MemberCount()
void StructMembers( ContainerElementArray * members )
```

C 言語のシグニチャー

```
unsigned long
FmcjContainerElementMemberCount( FmcjContainerElementHandle handle )
FmcjContainerElementVectorHandle
FmcjContainerElementStructMembers( FmcjContainerElementHandle handle )
```

C++ 言語のシグニチャー

```
unsigned long MemberCount()
void StructMembers( vector<FmcjContainerElement> const & members ) const
```

Java のシグニチャー

```
public abstract int memberCount() throws FmcException
public abstract ContainerElement[] structMembers() throws FmcException
```

パラメーター

handle 入力。照会するコンテナー要素のハンドル。

members 入出力。データが入れられるコンテナー要素のベクトルまたは配列。

戻り値のデータ型

ContainerElement[]/FmcjContainerElementVectorHandle
構造メンバーであるコンテナー要素。

long/unsigned long/int
構造メンバーの数。

配列の要素の判別

以下の関数 / メソッドは、配列内の要素の数を調べたり要素自体を取り出したりするためのものです。

ActiveX のシグニチャー

```
long Cardinality()
void ArrayElements( ContainerElementArray * elements )
```

C 言語のシグニチャー

```
unsigned long
FmcjContainerElementCardinality( FmcjContainerElementHandle handle )
FmcjContainerElementVectorHandle
FmcjContainerElementArrayElements( FmcjContainerElementHandle handle )
```

C++ 言語のシグニチャー

```
unsigned long Cardinality() const
void ArrayMembers( vector<FmcjContainerElement> const & elements ) const
```

Java のシグニチャー

```
public abstract int cardinality() throws FmcException
public abstract ContainerElement[] arrayElements() throws FmcException
```

パラメーター

handle 入力。照会するコンテナー要素のハンドル。

elements 入出力。データが入れられるコンテナー要素のベクトルまたは配列。

戻り値のデータ型

ContainerElement[]/FmcjContainerElementVectorHandle
照会する配列コンテナー要素の一部であるコンテナー要素。

long/unsigned long
コンテナー要素によって記述される配列のカーディナリティー。

認識されているコンテナー要素へのアクセス

コンテナー要素のドット (dotted) 名が分かったなら、その名前を使うことによって、コンテナーまたはコンテナー要素構造全体を繰り返し検索しなくてもそのコンテナー要素に直接アクセスすることができます。

注:

1. 修飾名は文字で始まっていなければならず、大括弧または小括弧で始めることはできません。すなわち、配列であるコンテナー要素の要素にアクセスする場合は、 ArrayElements() 関数 / メソッドを呼び出す必要があります。
2. ActiveX のシグニチャーはオブジェクト定義言語 (ODL) で提供されています。たとえば、*BSTR* 型は、VisualBasic の型が実際には String であるストリングに使用されます。

ActiveX のシグニチャー

```
long GetElement( BSTR              qualifiedName,
                 ContainerElement * element      )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjContainerGetElement(
                    FmcjContainerHandle   handle,
                    char const *          qualifiedName,
                    FmcjContainerElementHandle  * element      )
APIRET FMC_APIENTRY FmcjContainerElementGetElement(
                    FmcjContainerElementHandle handle,
                    char const *          qualifiedName,
                    FmcjContainerElementHandle  * element      )
```

C++ 言語のシグニチャー

```
APIRET GetElement( string const &         qualifiedName,
                   FmcjContainerElement & element      ) const
```

Java のシグニチャー

```
public abstract
ContainerElement getElement( String qualifiedName ) throws FmcException
```

パラメーター

element 出力。コンテナー要素。

handle 入力。照会するコンテナーまたはコンテナー要素のハンドル。

qualifiedName
入力。コンテナー要素の完全修飾名。

戻り値のデータ型

long/APIRET この関数 / メソッドを呼び出した結果の戻りコード。戻りコードの説明を参照してください。

コンテナーの値へのアクセス

次の関数 / メソッドを実行すると、コンテナー・リーフの値が戻されます。利用可能な情報がない場合は、FMC_ERROR_MEMBER_NOT_SET が戻されます。

リーフが値の配列の場合には、インデックスを指定しなければなりません。インデックスを指定することにより、完全修飾名はインデックスとその括弧はなしで指定します。

注: ActiveX のシグニチャーはオブジェクト定義言語 (ODL) で提供されています。たとえば、*BSTR* 型は、VisualBasic の型が実際には String であるストリングに使用されます。

ActiveX のシグニチャー

```
long GetValueDbl( BSTR      qualifiedName,
                  double *  value,
                  boolean   isArray,
                  long      index )
long GetValueLng( BSTR      qualifiedName,
                  long *    value,
                  boolean   isArray,
                  long      index )
long GetValueStr( BSTR      qualifiedName,
                  BSTR *    value,
                  boolean   isArray,
                  long      index )
```

C 言語のシグニチャー

```
unsigned long
       FMC_APIENTRY FmcjContainerArrayBinaryLength(
                    FmcjContainerHandle    handle,
                    char const *           qualified name,
                    unsigned long          index )
APIRET FMC_APIENTRY FmcjContainerArrayBinaryValue(
                    FmcjContainerHandle    handle,
                    char const *           qualifiedName,
              unsigned long                index,
                    FmcjBinary *           value,
                    unsigned long          bufferLength )
unsigned long
       FMC_APIENTRY FmcjContainerBinaryLength(
                    FmcjContainerHandle    handle,
                    char const *           qualified name )
APIRET FMC_APIENTRY FmcjContainerBinaryValue(
                    FmcjContainerHandle    handle,
                    char const *           qualifiedName,
                    FmcjBinary *           value,
                    unsigned long          bufferLength )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjContainerArrayFloatValue(
                    FmcjContainerHandle    handle,
                    char const *           qualifiedName,
                    unsigned long          index,
                    double *               value        )
APIRET FMC_APIENTRY FmcjContainerFloatValue(
                    FmcjContainerHandle    handle,
                    char const *           qualifiedName,
                    double *               value        )
                    unsigned long          bufferLength )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjContainerArrayLongValue(
                    FmcjContainerHandle    handle,
                    char const *           qualifiedName,
                    unsigned long          index,
                    long *                 value        )
APIRET FMC_APIENTRY FmcjContainerLongValue(
                    FmcjContainerHandle    handle,
                    long *                 value )
```

C 言語のシグニチャー

```
unsigned long
       FMC_APIENTRY FmcjContainerArrayStringLength(
                    FmcjContainerHandle    handle,
                    char const *           qualified name,
                    unsigned long          index )
APIRET FMC_APIENTRY FmcjContainerArrayStringValue(
                    FmcjContainerHandle    handle,
                    char const *           qualifiedName,
                    unsigned long          index,
                    char *                 value,
                    unsigned long          bufferLength )
unsigned long
        FMC_APIENTRY FmcjContainerArrayStringLength(
                     FmcjContainerHandle   handle,
                     char const *          qualified name )
APIRET FMC_APIENTRY FmcjContainerStringValue(
                     FmcjContainerHandle   handle,
                     char const *          qualifiedName,
                     char *                value,
                     unsigned long         bufferLength )
```

C++ 言語のシグニチャー

```
unsigned long BinaryLength( unsigned long index )
APIRET Value( string const &    qualifiedName,
              unsigned long     index,
              FmcjBinary *      value,
              unsigned long     bufferLength ) const
unsigned long BinaryLength()
```

C++ 言語のシグニチャー

```
APIRET Value( string const &    qualifiedName,
              unsigned long     index,
              long &            value ) const
APIRET Value( string const a    qualifiedName,
              long &            value ) const
```

C++ 言語のシグニチャー

```
APIRET Value( string const &    qualifiedName,
              unsigned long     index,
              double &          value ) const
APIRET Value( string const a    qualifiedName,
              double &          value ) const
```

C++ 言語のシグニチャー

```
APIRET Value( string const &    qualifiedName,
              unsigned long     index,
              string &          value ) const
APIRET Value( string const a    qualifiedName,
              string &          value ) const
```

Java のシグニチャー

```
public abstract
byte[] getBuffer2(  String qualifiedName,
                    int    index        ) throws FmcException
public abstract
byte[] getBuffer(   String qualifiedName ) throws FmcException
```

Java のシグニチャー

```
public abstract
double getDouble2(  String qualifiedName,
                    int    index        ) throws FmcException
public abstract
double getDouble(   String qualifiedName ) throws FmcException
```

Java のシグニチャー

```
public abstract
int getLong2(       String qualifiedName,
                    int    index        ) throws FmcException
public abstract
int getLong(        String qualifiedName ) throws FmcException
```

Java のシグニチャー

```
public abstract
String getString2(  String qualifiedName,
                    int    index        ) throws FmcException
public abstract
String getString(   String qualifiedName ) throws FmcException
```

パラメーター

bufferLength 入力。値を渡すために利用可能なバッファーの長さ。実際の長さ以上でなければなりません。実際の長さを調べるには、該当する Length() 関数 / メソッドを使用してください。

handle 入力。照会するコンテナーのハンドル。

index 入力。リーフが値の配列である場合、照会する配列要素のインデックス。

isArray 入力。 *True* に設定すると、配列が照会され、インデックスが使用されます。

qualifiedName
入力。コンテナー内のリーフの完全修飾名。

value 出力。リーフの値。

戻り値のデータ型

byte[]/double/int/String
リーフの値。

unsigned long
値を読み取るために必要とされるバッファー長の最小値。

long/APIRET この関数 / メソッドを呼び出した結果の戻りコード。戻りコードの説明を参照してください。

コンテナー要素の値へのアクセス

次の関数 / メソッドを実行すると、コンテナー要素リーフの値が戻されます。リーフが値の配列の場合には、インデックスを指定しなければなりません。利用可能な情報がない場合は、FMC_ERROR_MEMBER_NOT_SET が戻されます。コンテナー・リーフの照会とは異なり、コンテナー要素それ自体が照会されるリーフであるため、リーフの名前を指定する必要はありません。

注: ActiveX のシグニチャーはオブジェクト定義言語 (ODL) で提供されています。たとえば、*BSTR* 型は、VisualBasic の型が実際には String であるストリングに使用されます。

ActiveX のシグニチャー

```
long GetValueDbl( double *  value,
                  long      index )
long GetValueLng( long *    value,
                  long      index )
long GetValueStr( BSTR *    value,
                  long      index )
```

C 言語のシグニチャー

```
unsigned long
       FMC_APIENTRY FmcjContainerElementArrayBinaryLength(
                    FmcjContainerElementHandle handle,
                    unsigned long              index )
APIRET FMC_APIENTRY FmcjContainerElementArrayBinaryValue(
                    unsigned long              index,
                    FmcjBinary *               value,
                    unsigned long              bufferLength )
unsigned long
       FMC_APIENTRY FmcjContainerElementBinaryLength(
                    FmcjContainerElementHandle handle )
APIRET FMC_APIENTRY FmcjContainerElementBinaryValue(
                    FmcjContainerElementHandle handle,
                    FmcjBinary *               value,
                    unsigned long              bufferLength )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjContainerElementArrayFloatValue(
                    FmcjContainerElementHandle handle,
                    unsigned long              index,
                    double *                   value   )
APIRET FMC_APIENTRY FmcjContainerElementFloatValue(
                    FmcjContainerElementHandle handle,
                    double *                   value  )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjContainerElementArrayLongValue(
                    FmcjContainerElementHandle handle,
                    unsigned long              index,
                    long *                     value  )
APIRET FMC_APIENTRY FmcjContainerElementLongValue(
                    FmcjContainerElementHandle handle,
                    long *                     value )
```

C 言語のシグニチャー

```
unsigned long
       FMC_APIENTRY FmcjContainerElementArrayStringLength(
                    FmcjContainerElementHandle handle,
                    unsigned long              index )
APIRET FMC_APIENTRY FmcjContainerElementArrayStringValue(
                    FmcjContainerElementHandle handle,
                    unsigned long              index,
                    char *                     value,
                    unsigned long              bufferLength )
unsigned long
        FMC_APIENTRY FmcjContainerElementArrayStringLength(
                     FmcjContainerElementHandle handle    )
APIRET FMC_APIENTRY FmcjContainerElementStringValue(
                     FmcjContainerElementHandle handle,
                     char *                     value,
                     unsigned long              bufferLength )
```

C++ 言語のシグニチャー

```
unsigned long BinaryLength( unsigned long index )
APIRET Value( unsigned long     index,
              FmcjBinary *      value,
              unsigned long     bufferLength ) const
unsigned long BinaryLength()
APIRET Value( FmcjBinary *      value,
              unsigned long     bufferLength ) const
```

C++ 言語のシグニチャー

```
APIRET Value( unsigned long     index,
              long &            value ) const
APIRET Value( long &            value ) const
APIRET Value( unsigned long     index,
              double &          value ) const
APIRET Value( double &          value ) const
APIRET Value( unsigned long     index,
              string &          value ) const
APIRET Value( string &          value ) const
```

Java のシグニチャー

```
public abstract
byte[] getBuffer2( int index ) throws FmcException
public abstract
byte[] getBuffer() throws FmcException
public abstract
double getDouble2( int index ) throws FmcException
public abstract
double getDouble() throws FmcException
public abstract
int getLong2( int index ) throws FmcException
public abstract
int getLong() throws FmcException
public abstract
String getString2( int index ) throws FmcException
public abstract
String getString() throws FmcException
```

パラメーター

bufferLength　入力。値を渡すために利用可能なバッファーの長さ。実際の長

さ以上でなければなりません。実際の長さを調べるには、該当する Length() 関数 / メソッドを使用してください。

handle 入力。照会するコンテナー要素のハンドル。

index 入力。リーフが値の配列である場合、照会する配列要素のインデックス。 ActiveX の場合は、配列ではないコンテナー要素に関してインデックスは無視されます。

value 出力。リーフの値。

戻り値のデータ型

byte[]/double/int/String
リーフの値。

unsigned long
値を読み取るために必要とされるバッファー長の最小値。

long/APIRET この関数 / メソッドを呼び出した結果の戻りコード。戻りコードの説明を参照してください。

コンテナーの値の設定

次の関数 / メソッドは、コンテナー・リーフの値を設定するのに使います。

リーフが値の配列の場合には、インデックスを指定しなければなりません。インデックスを指定することにより、完全修飾名はインデックスとその括弧はなしで指定します。

注: ActiveX のシグニチャーはオブジェクト定義言語 (ODL) で提供されています。たとえば、*BSTR* 型は、VisualBasic の型が実際には String であるストリングに使用されます。

ActiveX のシグニチャー

```
long SetValueDbl( BSTR      qualifiedName,
                  double    value,
                  boolean   isArray,
                  long      index )
long SetValueLng( BSTR      qualifiedName,
                  long      value,
                  boolean   isArray,
                  long      index )
long SetValueStr( BSTR      qualifiedName,
                  BSTR      value,
                  boolean   isArray,
                  long      index )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjContainerSetArrayBinaryValue(
                    FmcjContainerHandle     handle,
                    char const *            qualifiedName,
                    unsigned long           index,
                    FmcjBinary const *      value,
                    unsigned long           dataLength )
APIRET FMC_APIENTRY FmcjContainerSetBinaryValue(
                    FmcjContainerHandle     handle,
                    char const *            qualifiedName,
                    FmcjBinary const *      value,
                    unsigned long           dataLength )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjContainerSetArrayFloatValue(
                    FmcjContainerHandle    handle,
                    char const *           qualifiedName,
                    unsigned long          index,
                    double                 value        )
APIRET FMC_APIENTRY FmcjContainerSetFloatValue(
                    FmcjContainerHandle    handle,
                    char const *           qualifiedName,
                    double                 value        )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjContainerSetArrayLongValue(
                    FmcjContainerHandle    handle,
                    char const *           qualifiedName,
                    unsigned long          index,
                    long                   value        )
APIRET FMC_APIENTRY FmcjContainerSetLongValue(
                    FmcjContainerHandle    handle,
                    long                   value )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjContainerSetArrayStringValue(
                    FmcjContainerHandle    handle,
                    char const *           qualifiedName,
                    unsigned long          index,
                    char const *           value    )
APIRET FMC_APIENTRY FmcjContainerSetStringValue(
                     FmcjContainerHandle   handle,
                     char const *          qualifiedName,
                     char const *          value )
```

C++ 言語のシグニチャー

```
APIRET SetValue( string const &      qualifiedName,
                 unsigned long       index,
                 FmcjBinary const *  value,
                 unsigned long       dataLength ) const
APIRET SetValue( string const &      qualifiedName,
                 FmcjBinary const *  value,
                 unsigned long       dataLength ) const
```

C++ 言語のシグニチャー

```
APIRET SetValue( string const &    qualifiedName,
                 unsigned long     index,
                 long              value ) const
APIRET SetValue( string const a    qualifiedName,
                 long              value ) const
```

C++ 言語のシグニチャー

```
APIRET SetValue( string const &    qualifiedName,
                 unsigned long     index,
                 double            value ) const
APIRET SetValue( string const a    qualifiedName,
                 double            value ) const
```

C++ 言語のシグニチャー

```
APIRET SetValue( string const &    qualifiedName,
                 unsigned long     index,
                 string const &    value        ) const
APIRET SetValue( string const &    qualifiedName,
                 string const &    value        ) const
```

Java のシグニチャー

```
public abstract
void setBuffer2(  String   qualifiedName,
                  int      index,
                  byte     value[]      ) throws FmcException
public abstract
void setBuffer(   String   qualifiedName,
                  byte     value[]      ) throws FmcException
```

Java のシグニチャー

```
public abstract
void setDouble2(  String   qualifiedName,
                  int      index,
                  double   value        ) throws FmcException
public abstract
void setDouble(   String   qualifiedName,
                  double   value        ) throws FmcException
```

Java のシグニチャー

```
public abstract
void setLong2(    String   qualifiedName,
                  int      index,
                  long     value        ) throws FmcException
public abstract
void setLong(     String   qualifiedName,
                  long     value        ) throws FmcException
```

Java のシグニチャー

```
public abstract
void setString2(  String   qualifiedName,
                  int      index,
                  String   value       ) throws FmcException
public abstract
void setString(   String   qualifiedName,
                  String   value       ) throws FmcException
```

パラメーター

dataLength 入力。 2 進値の長さ。

handle 入力。設定するコンテナーのハンドル。

index 入力。リーフが値の配列である場合、設定する配列要素のインデックス。

isArray 入力。 *True* に設定すると、配列が設定され、インデックスが使用されます。

qualifiedName
入力。コンテナー内のリーフの完全修飾名。

value 入力。リーフの値。 BINARY 型のリーフの値は、一連の 2 桁の 16 進数として指定しなければならないことに注意してください。たとえば、ストリング ’abc<cr><lf>’’ は ’6162630d0a’ として表されます (<cr> は ASCII の復帰文字、<lf> は ASCII の改行文字をそれぞれ表します)。

戻り値のデータ型

long/APIRET この関数 / メソッドを呼び出した結果の戻りコード。戻りコードの説明を参照してください。

戻りコード/FmcException

以下の戻りコードが戻される、または結果オブジェクトによって記述される可能性があります。あるいは以下の例外が送出されることがあります。括弧内の数字はその整数値を表します。

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR_BUFFER(800)
提供されたバッファーが小さ過ぎます。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスを受け取ることになっているにもかかわらず、0 が渡されました。

FMC_ERROR_EMPTY(122)
オブジェクトはまだサーバーから読み取られていません。

FMC_ERROR_FORMAT(117)
この修飾名は構文規則に従っていません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_MEMBER_CANNOT_BE_SET(115)
指定されたメンバーは MQSeries Workflow 事前定義の固定データ・メンバーです。情報としてのみ利用可能です。

FMC_ERROR_MEMBER_NOT_FOUND(112)
指定されたメンバーは、コンテナーまたはコンテナー要素の一部ではありません。

FMC_ERROR_MEMBER_NOT_SET(113)
指定されたメンバーには値がありません。

第9章 プロセス・インスタンスのモニター

MQSeries Workflow では、指定したプロセス・インスタンスのモニターを取得することができます。プロセス・インスタンス・モニターでは、一般に以下のことが可能です。

- プロセス・インスタンス実行の進行状況の監視。
- 実行の状態の判別。つまり、どのアクティビティー・インスタンスが現在進行中か、どのアクティビティー・インスタンスが実行を待機しているか、どのアクティビティー・インスタンスが InError 状態か、どのアクティビティー・インスタンスが特定のアクションを待機しているかなどを判別することです。この操作により、最大作業時間を超過したことを示す通知が出されていないかどうかを判別することが可能になります。
- 実行履歴の表示。つまり、プロセス・インスタンスが取ったパスとその理由。この操作により、実行のボトルネックとなっている個所や、最も時間を要している部分を判別することができます。

注: XML メッセージ・インターフェースにおいては、プロセス・インスタンスのモニターはサポートされていません。

プロセス・インスタンス・モニターの取得

一度プロセス・インスタンス[3]にアクセスすると、**プロセス・インスタンス・モニター[4]** を取得できます。一時プロセス・インスタンス・モニター・オブジェクトは、それが記述するプロセス・インスタンスに直接含まれているアクティビティー・インスタンスについての全情報と、それらのアクティビティー・インスタンスに接続されている制御コネクター・インスタンスについての全情報を表します。

3. またはアクティビティー・インスタンスまたは (ワーク) アイテム

4. ActiveX はプロセス・インスタンス・モニターとブロック・インスタンス・モニターを区別しません。両方ともインスタンス・モニター と呼ばれます。

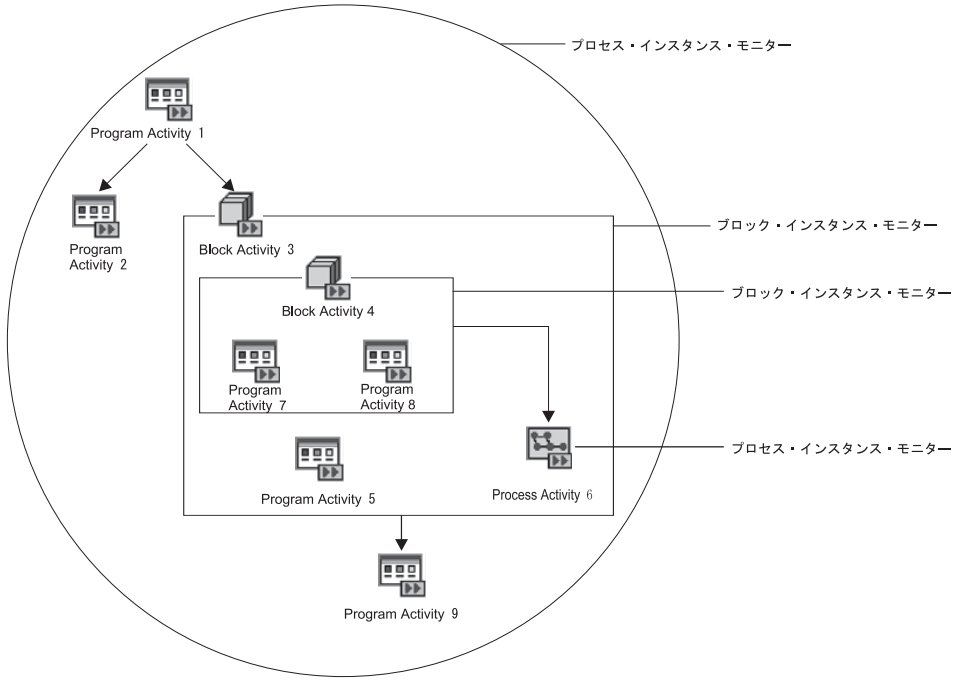


図 2. プロセス・インスタンス・モニターとブロック・インスタンス・モニター

たとえば、図のプロセス・インスタンス・モニターは *Program Activity 1*、*Program Activity 2*、および *Program Activity 9* という 3 つのプログラム・アクティビティーと、 *Block Activity 3* というブロック型のアクティビティー 1 つを記述しています。これらのアクティビティーの間には、3 つの制御コネクターがあります。

ここで記述されているアクティビティー・インスタンスと制御コネクター・インスタンスについての要求をプロセス・インスタンス・モニターに出し、それらのプロパティー (たとえばアクティビティーの状態とそのグラフィカルなレイアウトや、制御コネクター・インスタンス評価およびアクティビティーの結果) を判別して、描こうとしているポイントに接続したり、描こうとしているポイントを屈曲させたりすることができます。

ブロック 型のアクティビティーが出現したなら、その**ブロック・インスタンス・モニター**を取得することができます。 プロセス・インスタンス・モニターと同じように、ブロック・インスタンス・モニター・オブジェクトは、それが記述するブロック・アクティビティー・インスタンスの中に直接含まれているアクティビティー・インスタンスについての全情報と、それらのアクティビティー・インスタンスに接続されている制御コネクター・インスタンスについて

の全情報を表します。たとえば、*Block Activity 3* のブロック・インスタンス・モニターは *Block Activity 4*、*Program Activity 5*、および *Process Activity 6* を記述しています。 *Block Activity 4* と *Process Activity 6* の間には制御コネクターがあります。

プロセス 型のアクティビティーが出現したなら、それを包含しているモニター・オブジェクトを使用するか、そのアクティビティーのインプリメント・(サブ) プロセス・インスタンスを取り出した後、関連したプロセス・インスタンス・モニターを取得するという手順で、そのアクティビティーについてのプロセス・インスタンス・モニターを再び取得することができます。取得したプロセス・インスタンス・モニターは、他のプロセス・インスタンス・モニターとは完全に異なるモニターです。

プロセス・インスタンス・モニターを取得すると、*deep オプション* を使用できるようになります。このオプションを使用すれば、1 つの同じステップの中で、ブロック型のアクティビティーについてのすべての モニターを MQSeries Workflow 実行サーバーから戻すことができます。これ以降はどのブロック・インスタンス・モニターも、この取り出し時におけるプロセス・インスタンスの状態を示すようになります。つまり、API 呼び出しによってブロック・インスタンス・モニターを取得すると、API はこのモニターをキャッシュから探し出し、それを呼び出し側に戻します。 deep オプションを使用しない場合は、ブロック・インスタンス・モニターが使用できない可能性があります。その場合、API は要求されたモニターを実行サーバーから自動的に取り出します。それは、すでに取り出していた状態よりも新しい状態を表します。

注: 現在のところ、deep オプションは無視されます。

モニターの所有権

他の一時オブジェクトと同じように、プロセス・インスタンス・モニターの所有者は API の呼び出し側です。プロセス・インスタンス・モニターが不要になったら、オブジェクトを削除 / 割り振り解除する必要があります。

それでも、ブロック・インスタンス・モニターはプロセス・インスタンス・モニターの一部とみなされます。ブロック・インスタンス・モニターはプロセス・インスタンス・モニターの一部として、API によってキャッシュに入れられます。 C 言語の中で割り振り解除することはできません。 C++ 言語で削除すると C++ 表現だけが削除され、API キャッシュ内のブロック・インスタンス・モニター自体は残ります。ブロック・インスタンス・モニターを所有するプロセス・インスタンス・モニターが削除 / 割り振り解除されると、そのブロック・インスタンス・モニターも**自動的に削除されます**。つまり、ブロック・

インスタンス・モニター・オブジェクトまたはハンドルは、それが属するプロセス・インスタンス・モニターが存在するのでない限り使用できません。プロセス・インスタンス・モニターが存在していないのにブロック・インスタンス・モニター・オブジェクトまたはハンドルを使用すると、予期しない結果が戻されます。存在しないオブジェクトまたはハンドルを使用することは、MQSeries Workflow の*契約によるプログラミング* という概念に違反するため、プログラムでトラップが発生することもあります。

第10章 許可に関する考慮事項

一般に、許可は明示的または暗黙的に担当者に付与されます。「暗黙的に」という意味は、特定の MQSeries Workflow アクションの実行の結果として権限が与えられるということです。そのようなアクションの実行そのものによって、特定の権限を要求することができます。

システム管理者 の役割を果たす担当者には、特殊権限が付与されます。システム管理者には、(ワーク) アイテムに対する特権を除く、すべての特権が付与されます。あらゆるアクションを発行できるのは (ワーク) アイテムの所有者だけです。とはいえ、システム管理者も (ワーク) アイテムを自分のところに転送することはできます。システム管理者の役割は、常時 1 人の担当者に割り当てなければなりません。

プロセス・インスタンスを開始すると、そのプロセス管理者 が判別されます。プロセス管理者と判別された担当者は、そのプロセス・インスタンスでのプロセス管理権を受け取ります。

プロセス・インスタンスのプロセス管理者になるべき担当者は、プロセス・モデルの定義時に指定されます。プロセス管理者の識別は次のような方法でできます。

- PROCESS_ADMINISTRATOR キーワードのユーザー識別の指定。この場合、プロセス・モデルの定義時にプロセス管理者はすでに認識されています。
- PROCESS_ADMINISTRATOR TAKEN_FROM の指定によるプロセス入力コンテナー内のメンバーの指定。
- DATA FROM INPUT_CONTAINER の指定。この後で、入力コンテナー内のプロセス情報メンバー _PROCESS_INFO.ProcessAdministrator フィールドからプロセス管理者が取られます (詳細は、43ページの『プロセス情報のデータ・メンバー』を参照)。

次に示す表は、それぞれの許可と、権限が付与されている場合に呼び出すことのできる MQSeries Workflow 機能を示しています。 E/I (明示的 / 暗黙的) の欄は、許可がどのように担当者に付与されるかを示しています。

注: プログラム言語 API の場合、ユーザーが MQSeries Workflow に対して認証された (ログオンした) なら、それ以上の特別な権限がなくても、閲覧を

許可されたすべてのオブジェクトを検索することができます。それは、そのユーザーが作成したすべてのオブジェクトと、特別に保護されていないかまたはパブリック使用のすべてのオブジェクトです。

表 3. 担当者の許可

名前	E/I	許可される機能
許可定義の許可	E	許可情報の作成、更新、および削除。 パスワードの検索と更新。 該当する FDL 許可キーワードは AUTHORIZATION です。
操作管理の許可	E	すべての操作管理機能を実行できます。該当する FDL 許可キーワードは OPERATION です。
スタッフ定義の許可	E	スタッフ情報の作成、検索、更新、および削除。したがって、これには許可定義の許可も含まれます。 パブリックとプライベートのプロセス・インスタンス・リスト、プロセス・テンプレート・リスト、、およびワークリストの作成、検索、更新、および削除。 該当する FDL 許可キーワードは STAFF です。
トポロジー定義の許可	E	トポロジー情報の作成、検索、更新、および削除。該当する FDL 許可キーワードは TOPOLOGY です。
プロセス・モデル定義許可	E	プロセス・モデルおよびプロセス・テンプレートの作成、取り出し、更新、および削除。該当する FDL 許可キーワードは PROCESS_MODELING です。

表 3. 担当者の許可 (続き)

名前	E/I	許可される機能
プロセスの許可	E	プロセス・インスタンスがどのカテゴリーにも属していない場合、以下のプロセス・インスタンス機能を実行できます。プロセス・インスタンスが特定のカテゴリーに属している場合は、すべてのカテゴリーまたは以下の特定のカテゴリーに対する権限がなければなりません。 • 作成 • 開始 • 作成および開始 • プロセス・インスタンス名の設定 • 照会 • リフレッシュ プロセス・テンプレートがどのカテゴリーにも属していない場合、以下のプロセス・テンプレート機能を実行できます。プロセス・テンプレートが特定のカテゴリーに属している場合は、すべてのカテゴリーまたは以下の特定のカテゴリーに対する権限がなければなりません。 • 照会 • リフレッシュ 該当する FDL 許可キーワードは PROCESS_CATEGORY です。

表 3. 担当者の許可 (続き)

名前	E/I	許可される機能
プロセス管理の許可	E	プロセス・インスタンスがどのカテゴリーにも属していない場合に、処理権限を持ち、以下の追加のプロセス・インスタンス機能を実行できます。プロセス・インスタンスが特定のカテゴリーに属している場合は、すべてのカテゴリーまたは以下の特定のカテゴリーに対する管理権がなければなりません。 • 削除 • 再始動 • 再開 • 中断 • 中止 プロセス・インスタンスがどのカテゴリーにも属していない場合、すべてのプロセス・インスタンスについて割り当てられた作業項目に対して以下の作業項目機能を実行できます。プロセス・インスタンスが特定のカテゴリーに属している場合は、すべてのカテゴリーまたは以下の特定のカテゴリーに対する権限がなければなりません。 • 強制終了 • 強制再始動 該当する FDL 許可キーワードは PROCESS_CATEGORY AS ADMINISTRATOR です。
プロセス管理者	I	該当するプロセス・インスタンスに対するプロセス管理者権限が付与されます。
プロセス作成者	I	次に示すプロセス・インスタンス機能を実行できます。 • プロセス・インスタンス名の設定 • 開始していない場合の削除 • 照会 • リフレッシュ • 開始

表 3. 担当者の許可 (続き)

名前	E/I	許可される機能
作業項目権限	E	すべての担当者または選択された担当者に対する許可がある場合は、すべての担当者の (ワーク) アイテムに対して次に示す機能を実行できます。 • 照会 • リフレッシュ • 転送 該当する FDL 許可キーワードは WORKITEMS_OF です。
作業項目所有者	I	以下のもの以外の割り当てられた (ワーク) アイテムに対して、すべての機能を実行できます。 • 強制終了 • 強制再始動

第11章 関数 / メソッドの種類

MQSeries Workflow の関数 / メソッドは、実行する要求の種類と動作の特徴に応じて、いくつかのカテゴリーに分類できます。

基本	一時オブジェクトを管理します
アクセス	一時オブジェクトのプロパティーを読み取ります
アクション	永続オブジェクトを読み取ったり操作したりします
アクティビティー・インプリメンテーション	アクティビティー・インプリメンテーションまたはサポート・ツールのコンテナーを処理します
プログラム実行管理	プログラム実行エージェントを処理します

基本およびアクセス関数 / メソッドについて、以下にさらに詳しく一般的な説明を示します。これらのすべての関数 / メソッドは、別々のオブジェクトに関するものではありますが、よく似ており要件も類似しているからです。これらの関数はすべて API によってローカルに処理されます。つまり、サーバーと通信することはありません。他のカテゴリーの関数 / メソッドについては、353ページの『第7部 プログラミング・インターフェース』で別個に説明します。それらは、クライアント / サーバー通信またはプログラム実行エージェントとの通信を必要とする関数 / メソッドです。

基本関数 / メソッド

基本関数 / メソッドは、基本的に一時オブジェクトを割り振ったり構成したりするために、また割り振り解除したり破棄したりするために用意されています。これらは、サービス・オブジェクトなどのサポート・オブジェクトを作成するためのものです。また、そのようなオブジェクトの破棄以外に、MQSeries Workflow API によって暗黙で割り振られる永続オブジェクトの一時表現の破棄にも使えます。 161ページの『第15章 メモリー管理』も参照してください。

基本関数 / メソッドは、さまざまな API で、必要なものだけが提供されています。たとえば、Java では IsComplete()、IsEmpty()、およびエージェント・コンストラクターのみがサポートされます。

一時オブジェクトの特性のおかげで、実行するのにサーバーとの接続や特定の許可は必要ありません。

戻りコード

C 言語の関数および MQSeries Workflow の結果オブジェクトは、次に示すコードを戻します。括弧内の数字はそれぞれの整数値です。

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスを受け取ることになっているにもかかわらず、0 が渡されました。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_NAME(134)
指定された名前は無効です。それは 0 ポインターであるか、または構文規則に従っていません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。また、検出されたすべての例外について MQSeries Workflow トレースを検査してください。

基本関数 / メソッドを使うと、下記の基本的な操作を実行できます。 **Xxx** はなんらかのクラスまたは有効範囲を示します。たとえば、FmcjProcessInstanceEqual() は FmcjXxxEqual() によって表されます。

割り振り

以下の関数 / メソッドは、アプリケーションでそれぞれのオブジェクトをセットアップするのに使います。これは、文字列ベクトルなどのオブジェクトのサポートのために必要です。また、ActiveX で、永続オブジェクトによって初期化されるオブジェクトのために必要です。永続オブジェクトを表す一時オブジェクトは、永続オブジェクトの作成時または MQSeries Workflow サーバーからの照会時に、MQSeries Workflow API によって暗黙的に割り振られます。

C++ API において、コンストラクターは**すべての**クラス用の public コンストラクターとされ、それによってそれぞれのインスタンスをコレクションに入れることができます。これらをアプリケーションから呼び出すと、該当するクラスの空のオブジェクトが作成されます。ただし、それらのオブジェクトはまだ永続オブジェクトを表していません。

作成されたオブジェクトはすべて一時オブジェクトです。

ActiveX のシグニチャー

```
ActivityInstance *              NewActivityInstance()
ProcessTemplate *               NewProcessTemplate()
ProcessInstance *               NewProcessInstance()
Workitem *                      NewWorkitem()
ActivityInstanceNotification *  NewActivityInstanceNotification()
ProcessInstanceNotification *   NewProcessInstanceNotification()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjExecutionServiceAllocate( FmcjExecutionServiceHandle * handle )
APIRET FMC_APIENTRY FmcjExecutionServiceAllocateForGroup(
                              char const *                 systemGroup,
                              FmcjExecutionServiceHandle * handle )
APIRET FMC_APIENTRY FmcjExecutionServiceAllocateForSystem(
                              char const *                 system,
                              char const *                 systemGroup,
                              FmcjExecutionServiceHandle * handle )
APIRET FMC_APIENTRY
FmcjStringVectorAllocate( FmcjStringVectorHandle * hdlVector )
```

C++ 言語のシグニチャー

```
FmcjXxx()
FmcjDateTime( bool initWithCurrentDateTime= false )
FmcjDateTime( unsigned short year,   unsigned short month,
              unsigned short day,    unsigned short hour,
              unsigned short minute, unsigned short second )
FmcjExecutionService( string const & systemGroup )
FmcjExecutionService( string const & system,
                      string const & systemGroup )
```

Java のシグニチャー

```
Agent()
```

パラメーター

handle
入出力。オブジェクト構成時に設定するオブジェクトのハンドルのアドレス。渡されるハンドルが、依然として有効なオブジェクトを指していることがないようにしてください。そのオブジェクトは、新しいオブジェクトのハンドルの設定の前に自動的に割り振り解除されるわけではありません。

initWithCurrentTime
入力。日付 / 時間を現在の日時で初期化するかどうかを示す標識。

system
入力。実行サーバーが実行されている特定のシステム。

systemGroup
入力。実行サーバーの属するシステム・グループ。システム・グループを指定するだけで、MQSeries のクラスター機能を利用することができます。

year/month/day
入力。日付 / 時間の日付の部分。

hour/minute/second
入力。日付 / 時間の時間の部分。

戻り値のデータ型

APIRET
割り振りによって設定された戻りコード。

Object*
新たに構成されたオブジェクト。

代入

C++ API アプリケーションにおいて代入演算子を使うと、指定されたオブジェクトの内容をターゲット・オブジェクトに代入した後、そのターゲット・オブジェクトが戻されます。代入は、まずターゲット・オブジェクトを削除してから、指定されたオブジェクトの内容を代入することによってなされます。

C++ 言語のシグニチャー

```
FmcjXxx & operator=( FmcjXxx const & anObject )
```

パラメーター

anObject 入力。内容を代入する代入元オブジェクト。

比較 / 同等性

以下の関数 / メソッドによって、アプリケーションは、2 つの一時オブジェクトを比較して、それらが同じ永続オブジェクトや API オブジェクトであるかどうかを調べることができます。

通常、比較はオブジェクト識別子に基づいて実行されます。 2 つの一時オブジェクトが同じ永続オブジェクトを表す場合は、真が戻されます。比較される一時オブジェクトの内容は、それ以上検査されません。つまり、2 つの一時オブジェクトが永続オブジェクトの同じ状態のものかどうかは検査されません。

例外:

- サービス・オブジェクトは、それらが同じセッションを表す場合に等しくなります。
- エラー・オブジェクトは、同じエラーを報告する場合、つまり 2 つのエラー・オブジェクトに同じ戻りコードと同じパラメーターが含まれる場合に等しくなります。
- 2 つのプログラム・データ・オブジェクトが同じ作業項目に属する場合、それらは等しくなります。
- 制御コネクター・インスタンス・オブジェクトは、同じソース・アクティビティー・インスタンスとターゲット・アクティビティー・インスタンスが含まれている場合に等しくなります。
- ポイント・オブジェクトと記号レイアウト・オブジェクトは、そのプロパティーが等しい場合に等しくなります。

C 言語において、渡されたハンドルの 1 つが無効の場合、結果オブジェクトの戻りコードは *invalid handle (無効なハンドル)* に設定されます。どちらも無効の場合は真が戻され、そうでない場合は偽が戻されます。

ActiveX のシグニチャー

```
boolean IsEqual( IDispatch * anObject )
```

C 言語のシグニチャー

```
bool FMC_APIENTRY FmcjXxxEqual(  FmcjXxxHandle  handle1,
                                 FmcjXxxHandle  handle2 )
```

C++ 言語のシグニチャー

```
bool operator==( FmcjXxx const & anObject ) const
```

パラメーター

anObject 入力。該当するオブジェクトと比較するオブジェクト。
handle1 入力。比較する最初のオブジェクト。
handle2 入力。比較するもう一方のオブジェクト。

コピー

以下の関数 / メソッドによって、アプリケーションは特定の一時オブジェクトのコピーを作成できます。そのコピーは別個のオブジェクトとなり、独自の状態を持つことになります。

実行サービスは例外です。コピーで作成された実行サービスは、元のオブジェクトによって確立されたのと同じセッションを指します。特にこれは、いずれかのオブジェクトからのログオフを要求するとその (共通の) セッションがクローズされることを意味します。

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjXxxCopy( FmcjXxxHandle   handle,
                                 FmcjXxxHandle * newHandle )
```

C++ 言語のシグニチャー

```
FmcjXxx( FmcjXxx const & anObject )
```

パラメーター

anObject 入力。コピー元オブジェクト。
handle 入力。コピー元オブジェクトのハンドル。
newHandle 入出力。オブジェクト構成時に設定するハンドルのアドレス。渡されるハンドルが、依然として有効なオブジェクトを指していることがないようにしてください。そのオブジェクトは、新しいオブジェクトのハンドルの設定の前に自動的に割り振り解除されるわけではありません。

割り振り解除

以下の関数 / メソッドは、アプリケーションで、指定した一時オブジェクトを削除するのに使います。一時オブジェクトを削除しても、表示された永続オブジェクト (存在する場合) には影響はありません。

C 言語のハンドルは 0 に設定されるので、もう使えなくなります。 C++ のデストラクターは、FmcjXxx のインスタンスが削除されると自動的に呼び出されます。 ActiveX では、オブジェクトを Nothing に設定すると、その使用回数が 0 になって削除が可能になります。

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjXxxDeallocate( FmcjXxxHandle * handle )
```

C++ 言語のシグニチャー

```
virtual FmcjXxx()
```

パラメーター

handle

入出力。割り振り解除するオブジェクトのハンドルのアドレス。

IsComplete()

MQSeries Workflow サーバーからオブジェクトが完全に読み取られていれば (つまり 1 次と 2 次の両方のプロパティーが使用可能であれは)、真を戻します (96ページの『アクセス関数 / メソッド』も参照)。

ActiveX のシグニチャー

```
boolean IsComplete()
```

C 言語のシグニチャー

```
bool FMC_APIENTRY FmcjXxxIsComplete( FmcjXxxHandle handle )
```

C++ 言語のシグニチャー

```
bool IsComplete()
```

Java のシグニチャー

```
public abstract boolean IsComplete() throws FmcException
```

パラメーター

handle

入力。照会するオブジェクトのハンドル。

戻り値のデータ型

bool/boolean

オブジェクトがサーバーから完全に読み取られている場合は真、そうでない場合は偽。

IsEmpty()

一時オブジェクトにまだ実際のデータ値が入っていないかどうかを戻します。一時オブジェクトは作成されたばかりで、その内容はまだ省略時値です。まだ永続オブジェクトを反映していません。

ActiveX のシグニチャー

```
boolean IsEmpty()
```

C++ 言語のシグニチャー

```
bool IsEmpty()
```

Java のシグニチャー

```
public abstract boolean IsEmpty() throws FmcException
```

戻り値のデータ型

bool/boolean

オブジェクトがサーバーからまだ読み取られていない場合は真、そうでない場合は偽。

Kind()

照会されたオブジェクトの種類を戻します。

ActiveX のシグニチャー

```
Enum Kind()
```

C 言語のシグニチャー

```
enum FmcjXxxEnum FMC_APIENTRY FmcjXxxKind( FmcjXxxHandle handle )
```

C++ 言語のシグニチャー

```
FmcjXxx::Enum Kind() const
```

Java のシグニチャー

```
public abstract Enum kind() throws FmcException
```

パラメーター

handle 入力。照会するオブジェクトのハンドル。

戻り値のデータ型

FmcjXxxEnum/Enum

オブジェクトの種類 (列挙の個々の要素)。 102ページの『列挙値へのアクセス』も参照してください。

C 言語の例: 基本関数の使用

```
#include <stdio.h>
#include <fmcjcrun.h>
int main()
{
  APIRET                       rc;
  FmcjExecutionServiceHandle   service   = 0;
  FmcjWorkitemVectorHandle     wList     = 0;
  FmcjWorkitemHandle           workitem1 = 0;
  FmcjWorkitemHandle           workitem2 = 0;
  FmcjWorkitemHandle           workitem3 = 0;


  FmcjGlobalConnect();
  /* logon */
  FmcjExecutionServiceAllocate(&service);
  rc = FmcjExecutionServiceLogon( service,
                                  "USERID", "password",
                                   Fmc_SM_Default, Fmc_SA_Reset
                                 );


  /*  Query Workitems */
  rc= FmcjExecutionServiceQueryWorkitems( service,
                                          FmcjNoFilter,
                                          FmcjNoSortCriteria,
                                          FmcjNoThreshold,
                                          &wList );
  printf( "¥nQuery workitems returns rc : %u¥n", rc );
  fflush(stdout);


  if ( rc == FMC_OK && FmcjWorkitemVectorSize(wList) >= 2 )
  {                                    /* access first element         */
    workitem1= FmcjWorkitemVectorFirstElement(wList);
    if ( FmcjWorkitemIsComplete(workitem1) )
      printf( "Surprise - more than primary data available¥n" );
    else
      printf( "Primary data of first workitem available¥n" );
    fflush(stdout);


                                     /* access next element          */
    workitem2= FmcjWorkitemVectorNextElement(wList) ;
    if ( FmcjWorkitemEqual(workitem1,workitem2) )
      printf( "Surprise - workitems are equal¥n" );
    else
      printf( "Workitems represent different objects¥" );
    fflush(stdout);
```

```
                                    /* copy workitem                */
    FmcjWorkitemCopy(workitem1,&workitem3);
    if ( FmcjWorkitemEqual(workitem1,workitem3) )
      printf( "Workitems represent same persistent object¥n" );
    else
      printf( "Surprise - workitems are not equal¥n" );
    fflush(stdout);


                                     /* cleanup                      */
    FmcjWorkitemDeallocate(&workitem1);
    FmcjWorkitemDeallocate(&workitem2);
    FmcjWorkitemDeallocate(&workitem3);
  }
   FmcjWorkitemVectorDeallocate( &wList );


  /* logoff */
  FmcjExecutionServiceLogoff(service);
  FmcjExecutionServiceDeallocate(&service);
  FmcjGlobalDisconnect();
  return FMC_OK;
}
```

C++ の例: 基本メソッドの使用

```
#include <iomanip.h>
#include <bool.h>
#include <vector.h>
#include <fmcjstr.hxx>
#include <fmcjprun.hxx>
int main()
{
  FmcjGlobal::Connect();
   // logon
  FmcjExecutionService service;
  APIRET rc = service.Logon("USERID", "password");


  FmcjWorkitem workitem1;
  if ( workitem1.IsEmpty() )
    cout << "Transient workitem object has been created" << endl;
  else
    cout << "Surprise - workitem contains actual data" << endl;
```

```
  // Query Workitems
  vector<FmcjWorkitem>        wList;
  rc= service.QueryWorkitems( FmcjNoFilter,
                              FmcjNoSortCriteria,
                              FmcjNoThreshold,
                              wList );
  cout << "Query workitems returns rc : " << rc << endl ;


  if ( rc == FMC_OK && wList.size() >= 2 )
  {
    workitem1= wList[0];              // assign first element
    if ( workitem1.IsComplete() )
      cout << "Surprise - more than primary data available" << endl;
    else
      cout << "Primary data of first workitem available" << endl;


    FmcjWorkitem workitem2= wList[1];  // access next element
    if ( workitem1 == workitem2 )
      cout << "Surprise - workitems are equal" << endl;
    else
      cout << "Workitems represent different objects" << endl;


                                       // copy workitem
    FmcjWorkitem workitem3(workitem1);
   if ( workitem1 == workitem3 )
      cout << "Workitems represent same persistent object" << endl;
    else
      cout << "Surprise - workitems are not equal" << endl;
   }                                    // destructors called automatically


  // logoff
  rc = service.Logoff();
  FmcjGlobal::Disconnect();
  return FMC_OK;
}                                      // destructors called automatically
```

アクセス関数 / メソッド

アクセス関数 / メソッドは、一時オブジェクトのプロパティーを読み取ったり変更したりするためのものです。一時オブジェクトが永続オブジェクトを表す場合、永続オブジェクトが取り出されて一時オブジェクトを設定した時点、または永続オブジェクトが作成または更新された時点の、その永続オブジェクトの状態が戻り値に反映されます。その取り出しは、該当する関数 / メソッドの作成、照会、またはリフレッシュを使って MQSeries Workflow サーバーから

なされたものです。作成または更新は、MQSeries Workflow サーバーが新しい情報を送信 (プッシュ) するときにクライアントで実行されます。

一時オブジェクトが空 または不完全 である場合、またはアクセスされたプロパティーがオプション であって設定されていない場合のために、省略時値が用意されています。

省略時値は、文字値のプロパティーでは空文字列または空バッファー、整数値プロパティーでは 0、ブール値プロパティーでは偽、時刻値プロパティーではすべてのメンバーを 0 に設定したタイム・スタンプ、列挙値プロパティーでは "NotSet"、そして複数値プロパティーでは空ベクトルです。

C++、ActiveX、または Java で作成されたばかりの一時オブジェクトは空 です。これは、まだどの永続オブジェクトも反映していないからです。 *IsEmpty()* メソッドを使用すると、一時オブジェクトの内容がまだ省略時値かどうかだけを判別できます。空のオブジェクトでは、アクション関数 / メソッドは実行できないことに注意してください。

省略時解釈では、MQSeries Workflow API は永続オブジェクトに関する 2 つのビューを提供します。これらは、永続オブジェクトを、いわゆる *1* 次 プロパティーと、いわゆる *2* 次 プロパティーに分けます。アクセスの点では、1 次プロパティーの方が 『より重要』 であるとみなされます。これは、オブジェクトが照会されるとただちに戻されます。 2 次プロパティーと、1 次プロパティーのリフレッシュは、明示的な Refresh() 要求においてのみ戻され、オブジェクトごとにリフレッシュされます。 *IsComplete()* 関数 / メソッドを使用することによって、1 次および 2 次オブジェクト値がサーバーから読み取られたかどうかを判別することができます。

注: 2 次プロパティーへのアクセス時に、ActiveX API は不完全なオブジェクトを自動的にリフレッシュします。

1 次と 2 次のほかに、永続オブジェクトのプロパティーはオプションにすることができます。つまり、値がある場合とない場合があるということです。省略時値が戻された場合、*IsNull()* 関数 / メソッドを使うことによって、その値が明示的に設定されたものかどうか、またその値は実際には値が設定されていないことを示しているのかどうかを判別できます。たとえば、Threshold() が 0 を戻した場合には、しきい値が 0 に設定されていてオブジェクトが戻されない場合と、しきい値に値を設定できず可能なすべてのオブジェクトが戻される場合とが考えられます。 Java ではヌルのオブジェクトを戻すことができるため、IsNull() メソッドは必要ではありません。

ヌルであるということは、完全に読み取ったということに対する否定です。オブジェクトが未完了である限り、IsNull() は 2 次のオプション・プロパティーに関して真を戻します。実際の値が何か、またそれが設定されているかどうかが何も分からないためです。たとえば、ドキュメンテーション (documentation) は、オブジェクトの 2 次のオプション・プロパティーです。オブジェクトが照会されると、1 次プロパティーだけがサーバーから取り出されます。Documentation() 関数 / メソッドは、空文字列またはバッファーを戻します。ドキュメンテーションが設定されているかどうかを判別するには、DocumentationIsNull() 関数 / メソッドを使います。しかし、IsComplete() が偽を戻す限り、実際のドキュメンテーションの設定がどうであっても、結果は『真』になります。 2 次データが取り出されるまでは、ドキュメンテーションは設定されていないとみなされます。

永続オブジェクトの状態や、現行のログオンまたはログオフの状態がどうであっても、一時オブジェクトが存在する限りデータ値にアクセスすることができます。一般に、一時オブジェクトの存続期間はプログラマー自身で決めなければなりません。

一時オブジェクトの特性のおかげで、サーバーへの接続や特定の許可がなくても、オブジェクト・プロパティーにアクセスしたり、一時オブジェクトのオブジェクト・プロパティーを更新したりできます。

戻りコード

アクセス関数 / メソッドは、要求された値をその戻り値として提供します。アクセス関数 / メソッドの実行中にエラーが発生した場合、省略時値が戻されます。 C++ または C 言語の場合、発生したどのエラーに関しても MQSeries Workflow 結果オブジェクトを照会することができます。 Java は FmcException を生成します。コードとして可能性があるものは、次のとおりです。括弧内の数字はそれぞれの整数値です。

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスを受け取ることになっているにもかかわらず、0 が渡されました。

FMC_ERROR_BUFFER(800)
提供されたバッファーが最大値を入れるには小さすぎます。必要な長さについては、fmcmxcon.h を参照してください。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまり省略時値が戻されます。

FMC_ERROR_DOES_NOT_EXIST(118)
オブジェクトは存在しません。たとえば、メッセージ・カタログ内にメッセージが見つかりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。また、検出されたすべての例外について MQSeries Workflow トレースを検査してください。

FMC_ERROR_INVALID_CONFIGURATION_ID(1022)
指定された構成は無効です。それは 0 か、または構文規則に従っていません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_RESULT_HANDLE(814)
提供された結果オブジェクトのハンドルは無効です。それは 0 であるか、または結果オブジェクトを指していません。

FMC_ERROR_INVALID_TIME(802)
渡された時間は無効です。

FMC_ERROR_MESSAGE_CATALOG(815)
メッセージ・カタログが見つかりません。

FMC_ERROR_PROFILE(124)
プロファイルが見つからない、またはオープンできません。

FMC_ERROR_PROGRAM_EXECUTION(126)
関数 / メソッドはアクティビティー・インプリメンテーション内 (たとえば、SetConfiguration()) から呼び出すことはできません。

FMC_ERROR_WRONG_STATE(120)
オブジェクトが誤った状態になっているため、関数 / メソッドを実行できません。たとえば、ログオン後に構成を変更することはできません。

アクセス関数 / メソッドを使うと、以下に示す操作を実行できます。 **Xxx はなんらかのクラスまたは有効範囲を表し、″Property″ はプロパティーを照会することを示します**。たとえば、FmcjItemDescription() は FmcjXxxProperty() によって表されます。

bool 型の値へのアクセス

ブール 型のプロパティーの値を戻します。情報が入手できない場合、省略時解釈として偽 が戻されます。

ActiveX のシグニチャー

```
boolean Property()
```

C 言語のシグニチャー

```
bool FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
```

C++ 言語のシグニチャー

```
bool Property() const
```

Java のシグニチャー

```
public abstract boolean property() throws FmcException
```

パラメーター

handle 入力。照会するオブジェクトのハンドル。

戻り値のデータ型

bool/boolean プロパティー値。

宣言の例

ActiveX boolean ManualStartMode();

C 言語 bool FMC_APIENTRY FmcjWorkitemManualStartMode(FmcjWorkitemHandle handle);

C++ bool ManualStartMode() const;

Java public abstract boolean manualStartMode() throws FmcException;

日付 / 時刻型の値へのアクセス

日付 / 時刻プロパティーの値を戻します。情報が入手できない場合、タイム・スタンプとして 0 が戻されます。

ActiveX のシグニチャー

```
void Property( DateAndTime * time )
```

C 言語のシグニチャー

```
FmcjCDateTime FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
```

C++ 言語のシグニチャー

```
FmcjDateTime Property() const
```

Java のシグニチャー

```
public abstract Calendar property() throws FmcException
```

パラメーター

handle 入力。照会するオブジェクトのハンドル。

time 入出力。設定する日付 / 時刻オブジェクト。

戻り値のデータ型

FmcjCDateTime/ FmcjDateTime/Calendar
プロパティー値。

宣言の例

ActiveX void EndTime(DateAndTime * time);

C 言語 FmcjCDateTime FMC_APIENTRY FmcjWorkitemEndTime(FmcjWorkitemHandle handle);

C++ FmcjDateTime EndTime() const;

Java public abstract Calendar endTime() throws FmcException;

列挙値へのアクセス

プロパティーの列挙値を戻します。実際の値を判別するには、対応する整数値ではなく、可能な限り記号名を使ってください。整数値はどんな場合にも同じであるとは限りません。

情報を入手できない場合、"NotSet" またはそれと類似の標識が戻されます。

ActiveX のシグニチャー

```
Enum Property()
```

C 言語のシグニチャー

```
enum FmcjXxxEnum FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
```

C++ 言語のシグニチャー

```
FmcjXxx::Enum Property() const
```

Java のシグニチャー

```
public abstract Enum property() throws FmcException
```

パラメーター

handle 入力。照会するオブジェクトのハンドル。

戻り値のデータ型

FmcjXxxEnum/Enum

プロパティー値。列挙の個々の要素。

宣言の例

ActiveX AssignReason ReceivedAs();

C 言語 FmcjItemAssignReason FMC_APIENTRY FmcjWorkitemReceivedAs(FmcjWorkitemHandle handle);

C++ FmcjItem::AssignReason ReceivedAs() const;

Java public abstract AssignReason receivedAs() throws FmcException;

以下の列挙型と定数が定義されています。型は、ActiveX、C 言語、C++、Java の順に示されています。括弧内の数値は対応する整数値です。可能な限り記号名だけを使用してください。

割り当て理由

この列挙型は、ActiveX では AssignReason、 C 言語 では FmcjItemAssignReason、 C++ では FmcjItem::AssignReason Java では com.ibm.workflow.api.ItemPackage.AssignReason という名前になっています。値は次のいずれか。

NotSet(0) 割り当て理由がわからないことを示します。

- **ActiveX** AssignReason_NotSpecified
- **C 言語** Fmc_IR_NotSet
- **C++** FmcjItem::NotSpecified
- **Java** AssignReason.NOT_SPECIFIED

Normal(1) ユーザーにアイテムを受け取る許可が与えられているため、作業項目または通知がユーザーに割り当てられたことを示します。

- **ActiveX** AssignReason_Normal
- **C 言語** Fmc_IR_Normal
- **C++** FmcjItem::Normal
- **Java** AssignReason.NORMAL

Substitute(2) ユーザーはアイテムを受け取るべき担当者の代行者であるため、作業項目または通知が割り当てられたことを示します。

- **ActiveX** AssignReason_Substitute
- **C 言語** Fmc_IR_Substitute
- **C++** FmcjItem::Substitute
- **Java** AssignReason.Substitute

ProcessAdministrator(3)
ユーザーはプロセス管理者であるため、作業項目または通知が割り当てられたことを示します。

- **ActiveX** AssignReason_ProcessAdministrator
- **C 言語** Fmc_IR_ProcessAdministrator

C++ FmcjItem::ProcessAdministrator

Java AssignReason.PROCESS_ADMINISTRATOR

SystemAdministrator(4)
ユーザーはシステム管理者であるため、作業項目または通知が割り当てられたことを示します。

ActiveX AssignReason_SystemAdministrator

C 言語 Fmc_IR_SystemAdministrator

C++ FmcjItem::SystemAdministrator

Java AssignReason.SYSTEM_ADMINISTRATOR

ByTransfer(5) 作業項目または通知がユーザーに転送されたことを示します。

ActiveX AssignReason_ByTransfer

C 言語 Fmc_IR_ByTransfer

C++ FmcjItem::ByTransfer

Java AssignReason.BY_TRANSFER

監査設定値
この列挙型は、ActiveX では AuditSetting、 C 言語 では FmcjProcessTemplateAuditSetting、 C++ では FmcjProcessTemplate::AuditSetting、 および Java では com.ibm.workflow.api.ProcessTemplatePackage.AuditSetting という名前になっています。値は次のいずれか。

NotSet(0) 監査設定値がわからないことを示します。

ActiveX Audit_NotSet

C 言語 Fmc_TA_NotSet

C++ FmcjProcessTemplate::NotSet

Java AuditSetting.NOT_SET

NoAudit(1) 監査が実行されないことを示します。

ActiveX Audit_NoAudit

C 言語 Fmc_TA_NoAudit

C++ FmcjProcessTemplate::NoAudit

Java AuditSetting.NO_AUDIT

Condensed(2)

圧縮監査が実行されることを示します。

ActiveX	Audit_Condensed
C 言語	Fmc_TA_Condensed
C++	FmcjProcessTemplate::Condensed
Java	AuditSetting.CONDENSED

Full(3) 全監査が実行されることを示します。

ActiveX	Audit_Full
C 言語	Fmc_TA_Full
C++	FmcjProcessTemplate::Full
Java	AuditSetting.FULL

アクティビティー・インスタンスのエスカレーション

この列挙型は、 ActiveX では AIEscalation、 C 言語 では FmcjActivityInstanceEscalation、 C++ では FmcjActivityInstance::Escalation、および Java では com.ibm.workflow.api.ActivityInstancePackage.Escalation という名前になっています。値は次のいずれか。

NotSet(0) アクティビティー・インスタンス上に通知があるかどうかわからないことを示します。

ActiveX	AIEscalation_NotSpecified
C 言語	Fmc_AE_NotSet
C++	FmcjActivityInstance::NotSpecified
Java	Escalation.NOT_SPECIFIED

NoNotification(1)

現在のところ、アクティビティー・インスタンスに通知が発生していないことを示します。

ActiveX	AIEscalation_NoNotification
C 言語	Fmc_AE_NoNotification
C++	FmcjActivityInstance::NoNotification
Java	Escalation.NO_NOTIFICATION

FirstNotification

最初の通知が発生したことを示します。

ActiveX(4) AIEscalation_FirstNotification

C 言語(4) Fmc_AE_FirstNotification

C++(4) FmcjActivityInstance::FirstNotification

Java(2) Escalation.FIRST_NOTIFICATION

SecondNotification

2 番目の通知が発生したことを示します。

ActiveX(5) AIEscalation_SecondNotification

C 言語(5) Fmc_AE_SecondNotification

C++(5) FmcjActivityInstance::SecondNotification

Java(3) Escalation.SECOND_NOTIFICATION

アクティビティー・インスタンスの状態

この列挙型は、 ActiveX では ActivityInstanceState、 C 言語 では FmcjActivityInstanceStateValue、 C++ では FmcjActivityInstance::state、および Java では com.ibm.workflow.api.ActivityInstancePackage.ExecutionState という名前になっています。値は次のいずれか。

NotSet(0) アクティビティー・インスタンスの状態がわからないことを示します。

ActiveX AIState_Undefined

C 言語 Fmc_AS_NotSet

C++ FmcjActivityInstance::undefined

Java ExecutionState.UNDEFINED

Ready(1) アクティビティー・インスタンスが作動可能状態であることを示します。

ActiveX AIState_Ready

C 言語 Fmc_AS_Ready

C++ FmcjActivityInstance::ready

Java ExecutionState.READY

Running(2) アクティビティー・インスタンスが実行状態であることを示します。

ActiveX AIState_Running

C 言語 Fmc_AS_Running

	C++	FmcjActivityInstance::running
	Java	ExecutionState.RUNNING
Finished	アクティビティー・インスタンスが終了済み状態であることを示します。	
	ActiveX(4)	AIState_Finished
	C 言語(4)	Fmc_AS_Finished
	C++(4)	FmcjActivityInstance::finished
	Java(3)	ExecutionState.FINISHED
Terminated	アクティビティー・インスタンスが中止状態であることを示します。	
	ActiveX(8)	AIState_Terminated
	C 言語(8)	Fmc_AS_Terminated
	C++(8)	FmcjActivityInstance::terminated
	Java(4)	ExecutionState.TERMINATED
Suspended	アクティビティー・インスタンスが一時中断状態であることを示します。	
	ActiveX(16)	AIState_Suspended
	C 言語(16)	Fmc_AS_Suspended
	C++(16)	FmcjActivityInstance::suspended
	Java(5)	ExecutionState.SUSPENDED
Inactive	アクティビティー・インスタンスがまだ非アクティブであることを示します。	
	ActiveX(32)	AIState_Inactive
	C 言語(32)	Fmc_AS_Inactive
	C++(32)	FmcjActivityInstance::inactive
	Java(6)	ExecutionState.INACTIVE
CheckedOut	アクティビティー・インスタンスがすでにチェックアウトされたことを示します。	
	ActiveX(64)	AIState_CheckedOut
	C 言語(64)	Fmc_AS_CheckedOut
	C++(64)	FmcjActivityInstance::checkedOut

Java(7) ExecutionState.CHECKED_OUT

InError アクティビティー・インスタンスが正常に実行されなかったことを示します。

ActiveX(128) AIState_InError

C 言語(128) Fmc_AS_InError

C++(128) FmcjActivityInstance::inError

Java(8) ExecutionState.IN_ERROR

Executed アクティビティー・インスタンスがすでに実行されたことを示します。

ActiveX(256) AIState_Executed

C 言語(256) Fmc_AS_Executed

C++(256) FmcjActivityInstance::executed

Java(9) ExecutionState.EXECUTED

Planning アクティビティー・インスタンスが計画中状態であることを示します。

ActiveX(512) AIState_Planning

C 言語(512) Fmc_AS_Planning

C++(512) FmcjActivityInstance::planning

Java(10) ExecutionState.PLANNING

ForceFinished
アクティビティー・インスタンスが強制終了済み状態であることを示します。

ActiveX(1024) AIState_ForceFinished

C 言語(1024) Fmc_AS_ForceFinished

C++(1024) FmcjActivityInstance::forceFinished

Java(11) ExecutionState.FORCE_FINISHED

Skipped アクティビティー・インスタンスが実行されずにスキップされたことを示します。

ActiveX(2048) AIState_Skipped

C 言語(2048) Fmc_AS_Skipped

C++(2048) FmcjActivityInstance::skipped

Java(12) ExecutionState.SKIPPED

Deleted アクティビティー・インスタンスが削除されたことを示します。

ActiveX(4096) AIState_Deleted

C 言語(4096) Fmc_AS_Deleted

C++(4096) FmcjActivityInstance::deleted

Java(13) ExecutionState.DELETED

Terminating アクティビティー・インスタンスが中止処理中状態であることを示します。

ActiveX(8192) AIState_Teminating

C 言語(8192) Fmc_AS_Terminating

C++(8192) FmcjActivityInstance::terminating

Java(14) ExecutionState.TERMINATING

Suspending アクティビティー・インスタンスが一時中断処理中状態であることを示します。

ActiveX(16384) AIState_Suspending

C 言語(16384) Fmc_AS_Suspending

C++(16384) FmcjActivityInstance::suspendting

Java(15) ExecutionState.SUSPENDING

アクティビティー・インスタンス・タイプ

この列挙型は、 ActiveX では ActivityInstanceType、 C 言語 では FmcjActivityInstanceType、 C++ では FmcjActivityInstance::Type、および Java では com.ibm.workflow.api.ActivityInstancePackage.Type という名前になっています。値は次のいずれか。

NotSet(0) アクティビティー・インスタンスの種類がわからないことを示します。

ActiveX AIType_NotSet

C 言語 Fmc_AT_NotSet

C++ FmcjActivityInstance::NotSet

Java Type.NOT_SET

Process(1) アクティビティー・インスタンスがプロセスによってインプリメントされていることを示します。

ActiveX	AIType_Process
C 言語	Fmc_AT_Process
C++	FmcjActivityInstance::Process
Java	Type.PROCESS

Program(2) アクティビティー・インスタンスがプログラムによってインプリメントされていることを示します。

ActiveX	AIType_Program
C 言語	Fmc_AT_Program
C++	FmcjActivityInstance::Program
Java	Type.PROGRAM

Block アクティビティー・インスタンスがブロックによってインプリメントされていることを示します。

ActiveX(16)	AIType_Block
C 言語(16)	Fmc_AT_Block
C++(16)	FmcjActivityInstance::Block
Java(3)	Type.BLOCK

コネクターの状態

この列挙型は、 ActiveX では ConnectorState、 C 言語 では FmcjControlConnectorInstanceStateValue、 C++ では FmcjControlConnectorInstance::state、および Java では com.ibm.workflow.api.ControlConnectorInstancePackage.EvaluationState という名前になっています。値は次のいずれか。

False(0) 制御コネクターの評価結果が False であることを示します。

ActiveX	ConnectorState_False
C 言語	Fmc_CS_False
C++	FmcjControlConnectorInstance::False
Java	EvaluationState.IS_FALSE

True(1) 制御コネクターの評価結果が True であることを示します。

ActiveX	ConnectorState_True

C 言語 Fmc_CS_True

C++ FmcjControlConnectorInstance::True

Java EvaluationState.IS_TRUE

NotEvaluated(2)
制御コネクターがまだ評価されていないことを示します。

ActiveX ConnectorState_NotEvaluated

C 言語 Fmc_CS_NotEvaluated

C++ FmcjControlConnectorInstance::NotEvaluated

Java EvaluationState.NOT_EVALUATED

NotSet(3) 制御コネクターの評価結果がわからないことを示します。

ActiveX ConnectorState_NotSet

C 言語 Fmc_CS_NotSet

C++ FmcjControlConnectorInstance::NotSet

Java EvaluationState.NOT_SET

コネクターの種類

この列挙型は、 ActiveX では ConnectorType、 C 言語 では FmcjControlConnectorInstanceType、 C++ では FmcjControlConnectorInstance::Type、および Java では com.ibm.workflow.api.ControlConnectorInstancePackage.Type という名前になっています。値は次のいずれか。

NotSet(0) 制御コネクター・インスタンスの種類がわからないことを示します。

ActiveX ConnectorType_Undefined

C 言語 Fmc_CT_NotSet

C++ FmcjControlConnectorInstance::Undefined

Java Type.UNDEFINED

Condition(1) 制御コネクター・インスタンスが分岐条件の可能なコネクターであることを示します。

ActiveX ConnectorType_Condition

C 言語 Fmc_CT_Condition

C++ FmcjControlConnectorInstance::Condition

Java Type.CONDITION

Otherwise(2) 制御コネクター・インスタンスが “otherwise” コネクターであることを示します。

ActiveX ConnectorType_Otherwise

C 言語 Fmc_CT_Otherwise

C++ FmcjControlConnectorInstance::Otherwise

Java Type.OTHERWISE

実行データの種類

この列挙型は、 C 言語 では FmcjExecutionDataKindEnum、 C++ では FmcjExecutionData::KindEnum という名前になっています。値は次のいずれか。

NotSet(0) 実行データの種類がわからないことを示します。

ActiveX 該当しない

C 言語 Fmc_DART_NotSet

C++ FmcjExecutionData::NotSet

Java サポートなし

Terminate(2) 実行データの受信を終了できることを示します。

ActiveX 該当しない

C 言語 Fmc_DART_Terminate

C++ FmcjExecutionData::Terminate

Java サポートなし

ItemDeleted(1000)

実行データが作業項目または通知の削除を記述していることを示します。

ActiveX 該当しない

C 言語 Fmc_DART_ItemDeleted

C++ FmcjExecutionData::ItemDeleted

Java サポートなし

Workitem(1002)

実行データが作業項目の作成または更新を記述していることを示します。

ActiveX 該当しない

C 言語 Fmc_DART_Workitem

C++ FmcjExecutionData::Workitem

Java サポートなし

ActivityInstanceNotification(1003)

実行データがアクティビティー・インスタンス通知の作成または更新を記述していることを示します。

ActiveX 該当しない

C 言語 Fmc_DART_ActivityInstanceNotification

C++ FmcjExecutionData::ActivityInstanceNotification

Java サポートなし

ProcessInstanceNotification(1004)

実行データがプロセス・インスタンス通知の作成または更新を記述していることを示します。

ActiveX 該当しない

C 言語 Fmc_DART_ProcessInstanceNotification

C++ FmcjExecutionData::ProcessInstanceNotification

Java サポートなし

ExecuteInstanceResponse(1100)

プロセス・インスタンスの作成と実行を求める非同期要求に対する応答を、実行データが記述していることを示します。

ActiveX 該当しない

C 言語 Fmc_DART_ExecuteInstanceResponse

C++ FmcjExecutionData::ExecuteInstanceResponse

Java サポートなし

ExecuteProgramResponse(1101)

プログラムの実行を求める非同期要求に対する応答を、実行データが記述していることを示します。

ActiveX 該当しない

C 言語 Fmc_DART_ExecuteProgramResponse

C++ FmcjExecutionData::ExecuteProgramResponse

Java サポートなし

実行モード

この列挙型は、 ActiveX では ExeMode、 C 言語 では FmcjProgramTemplateExeMode、および C++ では FmcjProgramTemplate::ExeMode という名前になっています。

NotSet(0) 実行モードがわからないことを示します。

ActiveX	ExeMode_NotSet
C 言語	Fmc_GM_NotSet
C++	FmcjProgramTemplate::NotSet

Normal(1) プログラムはグローバル・トランザクションの一員ではないことを示します。

ActiveX	ExeMode_Normal
C 言語	Fmc_GM_Normal
C++	FmcjProgramTemplate::Normal

Safe(2) プログラムはグローバル・トランザクションの一員であることを示します。

ActiveX	ExeMode_Safe
C 言語	Fmc_GM_Safe
C++	FmcjProgramTemplate::Safe

実行ユーザー

この列挙型は、 ActiveX では ExeUser、 C 言語 では FmcjProgramTemplateExeUser、 C++ では FmcjProgramTemplate::ExeUser という名前になっています。値は次のいずれか。

NotSet(0) 実行ユーザーについてわからないことを示します。

ActiveX	ExeUser_NotSet
C 言語	Fmc_GU_NotSet
C++	FmcjProgramTemplate::NotSpecified

Agent(1) プログラムはプログラム実行エージェントの識別子の下で実行されることを示します。

ActiveX	ExeUser_Agent
C 言語	Fmc_GU_Agent
C++	FmcjProgramTemplate::Agent

Starter(2) プログラムは、プログラムの開始者のユーザー ID の下で実行されることを示します。

ActiveX	ExeUser_Starter
C 言語	Fmc_GU_Starter
C++	FmcjProgramTemplate::Starter

EXE オプション・スタイル

この列挙型は、 ActiveX では ExeOptionsStyle、 C 言語 では FmcjExeOptionsStyle、 C++ では FmcjExeOptions::Style、および Java では com.ibm.workflow.api.ProgramDataPackage.Style という名前になっています。値は次のいずれか。

NotSet(0) EXE のスタイルがわからないことを示します。

ActiveX	EOStyle_NotSet
C 言語	Fmc_EO_NotSet
C++	FmcjExeOptions::NotSet
Java	Style.NOT_SET

Visible(1) EXE が表示された状態で開始することを示します。

ActiveX	EOStyle_Visible
C 言語	Fmc_EO_Visible
C++	FmcjExeOptions::Visible
Java	Style.VISIBLE

Invisible(2) EXE が非表示の状態で開始することを示します。

ActiveX	EOStyle_Invisible
C 言語	Fmc_EO_Invisible
C++	FmcjExeOptions::Invisible
Java	Style.INVISIBLE

Minimized(3) EXE が最小化状態で開始することを示します。

ActiveX	EOStyle_Minimized
C 言語	Fmc_EO_Minimized
C++	FmcjExeOptions::Minimized
Java	Style.MINIMIZED

Maximized(4) EXE が最大化状態で開始することを示します。

ActiveX	EOStyle_Maximized
C 言語	Fmc_EO_Maximized
C++	FmcjExeOptions::Maximized
Java	Style.MAXIMIZED

外部サービス・オプション時間枠

この列挙型は、 ActiveX では ExternalOptionsTimePeriod、 C 言語 では FmcjExternalOptionsTimePeriod、 C++ では FmcjExternalOptions::TimePeriod、 Java では com.ibm.workflow.api.ProgramDataPackage.TimePeriod という名前になっています。値は次のいずれか。

NotSet(0) 外部サービス・タイムアウトがわからないことを示します。

ActiveX	TimePeriod_NotSet
C 言語	Fmc_EX_NotSet
C++	FmcjExternalOptions::NotSet
Java	TimePeriod.NOT_SET

TimeInterval(1)
プログラム実行エージェントが、指定された時間間隔だけ開始済み外部サービスの応答を待つことを示します。

ActiveX	TimePeriod_TimeInterval
C 言語	Fmc_EX_TimeInterval
C++	FmcjExternalOptions::TimeInterval
Java	TimePeriod.TIME_INTERVAL

Forever(2) プログラム実行エージェントが、開始済みの外部サービスからの応答を永久に待つことを示します。

ActiveX	TimePeriod_Forever
C 言語	Fmc_EX_Forever
C++	FmcjExternalOptions::Forever
Java	TimePeriod.FOREVER

Never(3) プログラム実行エージェント が、開始済み外部サービスの応答を待たないことを示します。

ActiveX	TimePeriod_Never

C 言語 Fmc_EX_Never

C++ FmcjExternalOptions::Never

Java TimePeriod.NEVER

インプリメンテーション・データの基礎

この列挙型は、 ActiveX では ImplementationDataBasis、 C 言語 では FmcjImplementationDataBasis、 ActiveX では ImplementationDataBasis、および Java では com.ibm.workflow.api.ProgramDataPackage.Basis という名前になっています。値は次のいずれか。

NotSet(0) インプリメント・プログラムのオペレーティング・システム・プラットフォームがわからないことを示します。

ActiveX Basis_NotSpecified

C 言語 Fmc_DP_NotSet

C++ FmcjImplementationData::NotSpecified

Java Basis.NOT_SET

OS2(1) プログラムが OS/2 プログラムであることを示します。

ActiveX Basis_OS2

C 言語 Fmc_DP_OS2

C++ FmcjImplementationData::OS2

Java Basis.OS2

AIX(2) プログラムが AIX プログラムであることを示します。

ActiveX Basis_AIX

C 言語 Fmc_DP_AIX

C++ FmcjImplementationData::AIX

Java Basis.AIX

HPUX(3) プログラムが HP-UX プログラムであることを示します。

ActiveX Basis_HPUX

C 言語 Fmc_DP_HPUX

C++ FmcjImplementationData::HPUX

Java Basis.HPUX

Windows95(4)

プログラムが Windows 95 プログラムであることを示します。

ActiveX Basis_Windows95

C 言語 Fmc_DP_Windows95

C++ FmcjImplementationData::Windows95

Java Basis.WINDOWS_95

WindowsNT(5)
プログラムが Windows NT または Windows 2000 プログラムであることを示します。

ActiveX Basis_WindowsNT

C 言語 Fmc_DP_WindowsNT

C++ FmcjImplementationData::WindowsNT

Java Basis.WINDOWS_NT

OS/390(6) プログラムが OS/390 プログラムであることを示します。

ActiveX Basis_OS390

C 言語 Fmc_DP_OS390

C++ FmcjImplementationData::OS390

Java Basis.OS390

Solaris(7) プログラムが Solaris プログラムであることを示します。

ActiveX Basis_Solaris

C 言語 Fmc_DP_Solaris

C++ FmcjImplementationData::Solaris

Java Basis.SOLARIS

インプリメンテーション・データ・タイプ
この列挙型は、 ActiveX では ImplementationDataProgramType、 C 言語 では FmcjImplementationDataType、 C++ では FmcjImplementationData::Type、および Java では com.ibm.workflow.api.ProgramDataPackage.Type という名前になっています。値は次のいずれか。

NotSet(0) インプリメンテーションについてわからないことを示します。

ActiveX IOProgramType_NotSet

C 言語 Fmc_DT_NotSet

C++ FmcjImplementationData::NotSet

Java ImplementationData.NOT_SET

EXE(1) プログラムが実行可能ファイルのプログラムであることを示します。

ActiveX	IOProgramType_EXE
C 言語	Fmc_DT_EXE
C++	FmcjImplementationData::EXE
Java	ImplementationData.EXE

DLL(2) プログラムがダイナミック・リンク・ライブラリーによってインプリメントされていることを示します。

ActiveX	IOProgramType_DLL
C 言語	Fmc_DT_DLL
C++	FmcjImplementationData::DLL
Java	ImplementationData.DLL

External プログラムがなんらかの外部サービスであることを示します。

ActiveX(4)	IOProgramType_External
C 言語(4)	Fmc_DT_External
C++(4)	FmcjImplementationData::External
Java(3)	ImplementationData.EXTERNAL

アイテムの状態

この列挙型は、 ActiveX では State、 C 言語 では FmcjItemStateValue、 C++ では FmcjItem::state、および Java では com.ibm.workflow.api.ItemPackage.ExecutionState という名前になっています。値は次のいずれか。

NotSet(0) アイテムの状態がわからないことを示します。

ActiveX	ItemState_Undefined
C 言語	Fmc_IS_NotSet
C++	FmcjItem::undefined
Java	ExecutionState.UNDEFINED

Ready(1) アイテムが作動可能状態であることを示します。

ActiveX	ItemState_Ready
C 言語	Fmc_IS_Ready
C++	FmcjItem::ready

	Java	ExecutionState.READY
Running(2)	アイテムが実行状態であることを示します。	
	ActiveX	ItemState_Running
	C 言語	Fmc_IS_Running
	C++	FmcjItem::running
	Java	ExecutionState.RUNNING
Finished	アイテムが終了済み状態であることを示します。	
	ActiveX(4)	ItemState_Finished
	C 言語(4)	Fmc_IS_Finished
	C++(4)	FmcjItem::finished
	Java(3)	ExecutionState.FINISHED
Terminated	アイテムが中止状態であることを示します。	
	ActiveX(8)	ItemState_Terminated
	C 言語(8)	Fmc_IS_Terminated
	C++(8)	FmcjItem::terminated
	Java(4)	ExecutionState.TERMINATED
Suspended	アイテムが一時中断状態であることを示します。	
	ActiveX(16)	ItemState_Suspended
	C 言語(16)	Fmc_IS_Suspended
	C++(16)	FmcjItem::suspended
	Java(5)	ExecutionState.SUSPENDED
Disabled	アイテムが使用不可になっていることを示します。	
	ActiveX(32)	ItemState_Disabled
	C 言語(32)	Fmc_IS_Disabled
	C++(32)	FmcjItem::disabled
	Java(6)	ExecutionState.DISABLED
CheckedOut	アイテムがチェックアウト済みであることを示します。	
	ActiveX(64)	ItemState_CheckedOut
	C 言語(64)	Fmc_IS_CheckedOut

C++(64) FmcjItem::checkedOut

Java(7) ExecutionState.CHECKED_OUT

InError アイテムが InError 状態であることを示します。

ActiveX(128) ItemState_InError

C 言語(128) Fmc_IS_InError

C++(128) FmcjItem::inError

Java(8) ExecutionState.IN_ERROR

Executed アイテムがすでに実行されたことを示します。

ActiveX(256) ItemState_Executed

C 言語(256) Fmc_IS_Executed

C++(256) FmcjItem::Executed

Java(9) ExecutionState.EXECUTED

Planning アイテムが計画中状態であることを示します。

ActiveX(512) ItemState_Planning

C 言語(512) Fmc_IS_Planning

C++(512) FmcjItem::Planning

Java(10) ExecutionState.PLANNING

ForceFinished
アイテムが強制終了済みであることを示します。

ActiveX(1024) ItemState_ForceFinished

C 言語(1024) Fmc_IS_ForceFinished

C++(1024) FmcjItem::ForceFinished

Java(11) ExecutionState.FORCE_FINISHED

Deleted アイテムが削除済みであることを示します。

ActiveX(4096) ItemState_Deleted

C 言語(4096) Fmc_IS_Deleted

C++(4096) FmcjItem::Deleted

Java(12) ExecutionState.DELETED

Terminating アイテムが中止処理中状態であることを示します。

ActiveX(8192) ItemState_Terminating

C 言語(8192) Fmc_IS_Terminating

C++(8192) FmcjItem::Terminating

Java(13) ExecutionState.TERMINATING

Suspending アイテムが一時中断処理中状態であることを示します。

ActiveX(16384)
ItemState_Suspending

C 言語(16384)
Fmc_IS_Suspending

C++(16384) FmcjItem::Suspending

Java(14) ExecutionState.SUSPENDING

アイテムの種類

この列挙型は、 ActiveX では Kind、 C 言語 では FmcjItemType、 C++ では FmcjItem::ItemType、 Java では com.ibm.workflow.api.ItemPackage.ItemType という名前になっています。値は次のいずれか。

NotSet(0) アイテムの種類がわからないことを示します。

ActiveX Kind_Unknown

C 言語 Fmc_IT_NotSet

C++ FmcjItem::unknown

Java ItemType.UNKNOWN

Workitem(1) アイテムが作業項目であることを示します。

ActiveX Kind_Workitem

C 言語 Fmc_IT_Workitem

C++ FmcjItem::Workitem

Java ItemType.WORK_ITEM

ProcessInstanceNotification
アイテムがプロセス・インスタンス通知であることを示します。

ActiveX(3) Kind_ProcessInstanceNotification

C 言語(3) Fmc_IT_ProcessInstanceNotification

C++(3) FmcjItem::ProcessInstanceNotification

Java(2) ItemType.PROCESS_INSTANCE_NOTIFICATION

FirstActivityInstanceNotification
アイテムが第 1 のアクティビティー・インスタンス通知であることを示します。

ActiveX(4) Kind_FirstActivityInstanceNotification

C 言語(4) Fmc_IT_FirstActivityInstanceNotification

C++(4) FmcjItem::FirstActivityInstanceNotification

Java(3) ItemType.FIRST_ACTIVITY_INSTANCE_NOTIFICATION

SecondActivityInstanceNotification
アイテムが第 2 のアクティビティー・インスタンス通知であることを示します。

ActiveX(5) Kind_SecondActivityInstanceNotification

C 言語(5) Fmc_IT_SecondActivityInstanceNotification

C++(45) FmcjItem::SecondActivityInstanceNotification

Java(4) ItemType.SECOND_ACTIVITY_INSTANCE_NOTIFICATION

永続リストの種類
この列挙型は、 ActiveX では TypeOfList、 C 言語 では FmcjPersistentListTypeOfList、 C++ では FmcjPersistentList::TypeOfList、および Java では com.ibm.workflow.api.PersistentListPackage.TypeOfList という名前になっています。値は次のいずれか。

NotSet(0) リストの種類がわからないことを示します。

ActiveX TypeOfList_NotSet

C 言語 Fmc_LT_NotSet

C++ FmcjPersistentList::NotSet

Java TypeOfList.NOT_SET

Public(1) リスト定義がパブリック (public) になっていることを示します。

ActiveX TypeOfList_Public

C 言語 Fmc_LT_Public

C++ FmcjPersistentList::Public

Java TypeOfList.PUBLIC

Private リスト定義がプライベート (private) になっていることを示します。

ActiveX(3) TypeOfList_Private

C 言語(3) Fmc_LT_Private

C++(3) FmcjPersistentList::Private

Java(2) TypeOfList.PRIVATE

プロセス・インスタンスのエスカレーション
この列挙型は、 ActiveX では PIEscalation、 C 言語 では FmcjProcessInstanceEscalation、 C++ では FmcjProcessInstance::Escalation、および Java では com.ibm.workflow.api.ProcessInstancePackage.Escalation という名前になっています。値は次のいずれか。

NotSet(0) プロセス・インスタンス上に通知があるかどうかわからないことを示します。

ActiveX PIEscalation_NotSet

C 言語 Fmc_PE_NotSet

C++ FmcjProcessInstance::NotSet

Java Escalation.NOT_SET

NoNotification(1)
現在のところ、プロセス・インスタンスに通知が発生していないことを示します。

ActiveX PIEscalation_NoNotification

C 言語 Fmc_PE_NoNotification

C++ FmcjProcessInstance::NoNotification

Java Escalation.NO_NOTIFICATION

ProcessInstanceNotification
プロセス・インスタンス通知が発生したことを示します。

ActiveX(3) PIEscalation_ProcessNotification

C 言語(3) Fmc_PE_ProcessNotification

C++(3) FmcjProcessInstance::ProcessNotification

Java(2) Escalation.PROCESS_NOTIFICATION

プロセス・インスタンスの状態

この列挙型は、 ActiveX では ProcInstanceState、 C 言語 では FmcjProcessInstanceStateValue、 C++ では FmcjProcessInstance::state、および Java では com.ibm.workflow.api.ProcessInstancePackage.ExecutionState という名前になっています。値は次のいずれか。

NotSet(0) プロセス・インスタンスの状態がわからないことを示します。

ActiveX	State_Undefined
C 言語	Fmc_PS_NotSet
C++	FmcjProcessInstance::undefined
Java	ExecutionState.UNDEFINED

Ready(1) プロセス・インスタンスが作動可能状態であることを示します。

ActiveX	State_Ready
C 言語	Fmc_PS_Ready
C++	FmcjProcessInstance::ready
Java	ExecutionState.READY

Running(2) プロセス・インスタンスが実行状態であることを示します。

ActiveX	State_Running
C 言語	Fmc_PS_Running
C++	FmcjProcessInstance::running
Java	ExecutionState.RUNNING

Finished プロセス・インスタンスが終了済み状態であることを示します。

ActiveX(4)	State_Finished
C 言語(4)	Fmc_PS_Finished
C++(4)	FmcjProcessInstance::finished
Java(3)	ExecutionState.FINISHED

Terminated プロセス・インスタンスが中止状態であることを示します。

ActiveX(8)	State_Terminated
C 言語(8)	Fmc_PS_Terminated
C++(8)	FmcjProcessInstance::terminated

	Java(4)	ExecutionState.TERMINATED
Suspended	プロセス・インスタンスが一時中断状態であることを示します。	
	ActiveX(16)	State_Suspended
	C 言語(16)	Fmc_PS_Suspended
	C++(16)	FmcjProcessInstance::suspended
	Java(5)	ExecutionState.SUSPENDED
Terminating	プロセス・インスタンスが中止処理中状態であることを示します。	
	ActiveX(32)	State_Terminating
	C 言語(32)	Fmc_PS_Terminating
	C++(32)	FmcjProcessInstance::terminating
	Java(6)	ExecutionState.TERMINATING
Suspending	プロセス・インスタンスが一時中断処理中状態であることを示します。	
	ActiveX(64)	State_Suspending
	C 言語(64)	Fmc_PS_Suspending
	C++(64)	FmcjProcessInstance::suspending
	Java(7)	ExecutionState.SUSPENDING
Deleted	プロセス・インスタンスが削除済み状態であることを示します。	
	ActiveX(128)	State_Deleted
	C 言語(128)	Fmc_PS_Deleted
	C++(128)	FmcjProcessInstance::deleted
	Java(8)	ExecutionState.DELETED

作業項目のプログラム検索

この列挙型は、 ActiveX では WorkitemProgramRetrieval、 C 言語 では FmcjWorkitemProgramRetrieval、 C++ では FmcjWorkitem::ProgramRetrieval、および Java では com.ibm.workflow.api.WorkItemPackage.ProgramRetrieval という名前になっています。値は次のいずれか。

NotSet(0) どのプログラム定義を取り出すかについて何も設定されていないことを示します。

ActiveX	WIProgramRetrieval_NotSet
C 言語	Fmc_WS_NotSet
C++	FmcjWorkitem::NotSet
Java	ProgramRetrieval.NOT_SET

CommonDataOnly(1)
プログラム定義の共通部分を取り出すことを示します。

ActiveX	WIProgramRetrieval_CommonDataOnly
C 言語	Fmc_WS_CommonDataOnly
C++	FmcjWorkitem::CommonDataOnly
Java	ProgramRetrieval.COMMON_DATA_ONLY

SpecifiedDefinitions(2)
指定されたプログラム定義を取り出すことを示します。

ActiveX	WIProgramRetrieval_SpecifiedDefinitions
C 言語	Fmc_WS_SpecifiedDefinitions
C++	FmcjWorkitem::SpecifiedDefinitions
Java	ProgramRetrieval.SPECIFIED_DEFINITIONS

AllDefinitions すべてのプログラム定義を取り出すことを示します。

ActiveX(4)	WIProgramRetrieval_AllDefinitions
C 言語(4)	Fmc_WS_AllDefinitions
C++(4)	FmcjWorkitem::AllDefinitions
Java(3)	ProgramRetrieval.ALL_DEFINITIONS

整数型の値へのアクセス

long 型、*unsigned long* 型、または *int* 型のプロパティーの値を戻します。情報を入手できない場合、0 が戻されます。

ActiveX のシグニチャー

```
long Property()
```

C 言語のシグニチャー

```
long FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
unsigned long FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
```

C++ 言語のシグニチャー

```
long Property() const
unsigned long Property() const
```

Java のシグニチャー

```
public abstract int property() throws FmcException
```

パラメーター

handle 入力。照会するオブジェクトのハンドル。

戻り値のデータ型

long/unsigned long/int
プロパティー値。

宣言の例

ActiveX long Priority();

C 言語 unsigned long FMC_APIENTRY FmcjWorkitemPriority(FmcjWorkitemHandle handle);

C++ unsigned long Priority() const;

Java public abstract int priority() throws FmcException;

ストリング型の値へのアクセス

ストリング 型のプロパティーの値を戻します。情報が入手できない場合、空ストリングまたは空バッファーが戻されます。

注: ActiveX のシグニチャーはオブジェクト定義言語 (ODL) で提供されています。たとえば、*BSTR* 型は、VisualBasic の型が実際には String であるストリングに使用されます。

ActiveX のシグニチャー

```
BSTR Property()
```

C 言語のシグニチャー

```
char * FMC_APIENTRY FmcjXxxProperty(  FmcjXxxHandle  handle,
                                      char *         buffer,
                                      unsigned long  bufferLength )
```

C++ 言語のシグニチャー

```
string Property() const
```

Java のシグニチャー

```
public abstract String property() throws FmcException
```

パラメーター

handle 入力。照会するオブジェクトのハンドル。

buffer 入出力。プロパティー値を入れるバッファーを指すポインター。

bufferLength 入力。バッファーの長さ。最大限の値を入れるのに十分な大きさでなければなりません (最低限必要な長さについてはファイル fmcmxcon.h を参照)。 1 つのバッファーを使って、すべての文字値を取り出すことができます。

戻り値のデータ型

BSTR/char*/string/String

プロパティー値。

宣言の例

ActiveX BSTR Description();

C 言語 char* FMC_APIENTRY FmcjWorkitemDescription(FmcjWorkitemHandle handle);

C++ string Description() const;

Java public abstract String Description() throws FmcException;

複数値プロパティーへのアクセス

値の集合を提供することにより、複数値プロパティーの値を戻します。この集合は、C++ および C 言語ではベクトルとして、また ActiveX と Java では配列として表されます。 C++ では、データを入れる集合オブジェクトを、呼び出し元が提供しなければなりません。 1 つの値を読み取るには、該当するアクセス関数 / メソッドを使います (29ページの『C 言語のベクトル』を参照)。

情報を入手できない場合、未変更のベクトルまたは空の配列が戻されます。

既存の配列要素はすべて上書きされます。ただし、C++ のベクトル要素は、提供されたベクトルに付加されます。実際の値だけを読みたい場合、ベクトルのすべての要素を消去する必要があります。

ActiveX のシグニチャー

```
void Property( ValueTypeArray * value )
```

C 言語のシグニチャー

```
FmcjValueTypeVectorHandle
FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
```

C++ 言語のシグニチャー

```
void Property( vector<ValueType> & value ) const
```

Java のシグニチャー

```
public abstract ValueType[] property() throws FmcException
```

パラメーター

handle
入力。照会するオブジェクトのハンドル。

value 入出力。プロパティー値を入れるベクトルまたは配列。

戻り値のデータ型

FmcjValueTypeVectorHandle/ValueType[]
プロパティー値のベクトルまたは配列。

宣言の例

ActiveX	void Staff(StringArray * staff);
C 言語	FmcjStringVectorHandle FMC_APIENTRY FmcjWorkitemStaff(FmcjWorkitemHandle handle);
C++	void Staff(vector<string> & staff) const;
Java	public abstract String[] staff() throws FmcException;

オブジェクト値のプロパティーへのアクセス

それ自体がオブジェクトで記述されたプロパティーの値を戻します。

ActiveX のシグニチャー

```
Object  Property()
```

C 言語のシグニチャー

```
FmcjObjectHandle
FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
```

C++ 言語のシグニチャー

```
FmcjObject Property() const
```

Java のシグニチャー

```
public abstract Object property() throws FmcException
public abstract ExecutionService
locate( String systemGroup, String system) throws FmcException
public abstract
ExecutionAgent getExecutionAgent() throws FmcException
```

パラメーター

handle
入力。照会するオブジェクトのハンドル。

system
入力。実行サーバーが実行されているシステム。

systemGroup
入力。実行サーバーの実行されているシステム・グループ。

戻り値のデータ型

ExecutionAgent
アクティビティー・インプリメンテーションのコンテキストを提供するプログラム実行エージェント。

ExecutionService
実行サーバーへのインターフェースを提供する実行サービス。

Object/Handle/FmcjObject
プロパティー値。

宣言の例

ActiveX	fmcError ErrorReason();
C 言語	FmcjErrorHandle FMC_APIENTRY FmcjWorkitemErrorReason(FmcjWorkitemHandle handle);
C++	FmcjError ErrorReason() const;
Java	public abstract FmcError errorReason() throws FmcException;

ポインター値のプロパティーへのアクセス

なんらかのオブジェクトを指すポインターであるプロパティーの値を戻します。

ActiveX のシグニチャー

```
Object * Property()
```

C 言語のシグニチャー

```
FmcjObjectHandle
FMC_APIENTRY FmcjXxxProperty( FmcjXxxHandle handle )
```

C++ 言語のシグニチャー

```
FmcjObject * Property() const
```

Java のシグニチャー

```
public abstract Object property() throws FmcException
```

パラメーター

handle

入力。照会するオブジェクトのハンドル。

戻り値のデータ型

Object*/Handle/FmcjObject*

オブジェクトへのポインターまたはハンドル、またはオブジェクト自体。

宣言の例

ActiveX Container * InContainer();

C 言語 FmcjReadOnlyContainerHandle FMC_APIENTRY FmcjProgramDataInContainer(FmcjProgramDataHandle handle);

C++ FmcjReadOnlyContainer* InContainer() const;

Java public abstract ReadOnlyContainer inContainer() throws FmcException;

オプション・プロパティーが設定されているかどうかの判別

この関数 / メソッドは、オプション・プロパティーが設定されているかどうかを示します。

プロパティーが 2 次プロパティーであって、かつ照会されるオブジェクトがまだ完全には読み取られていない場合、そのプロパティーが設定されているかどうかはわからないため、省略時値として真が戻されます。

注: Java ではヌル・オブジェクトを戻すことができるので、IsNull() メソッドは公開されていません。

ActiveX のシグニチャー

```
boolean PropertyIsNull()
```

C 言語のシグニチャー

```
bool FMC_APIENTRY FmcjXxxPropertyIsNull( FmcjXxxHandle handle )
```

C++ 言語のシグニチャー

```
bool PropertyIsNull() const
```

パラメーター

handle 入力。照会するオブジェクトのハンドル。

戻り値のデータ型

bool/boolean プロパティーが設定されていない場合は真、それ以外の場合は偽。

宣言の例

ActiveX boolean DescriptionIsNull();

C 言語 bool FMC_APIENTRY FmcjWorkitemDescriptionIsNull(FmcjWorkitemHandle handle);

C++ bool DescriptionIsNull() const;

整数型の値の設定

この関数 / メソッドは、指定されたプロパティーを、指定された値に設定します。

ActiveX のシグニチャー

```
void SetProperty( long newValue )
```

C 言語のシグニチャー

```
void FMC_APIENTRY FmcjXxxSetProperty( FmcjXxxHandle  handle,
                                      long           newValue );
```

C++ 言語のシグニチャー

```
void SetProperty( long newValue );
```

Java のシグニチャー

```
public abstract void setProperty( int newValue ) throws FmcException
```

パラメーター

handle 入力。照会するオブジェクトのハンドル。

newValue 入力。プロパティーの新しい値。

宣言の例

ActiveX void SetTimeout(long newValue);

C 言語 void FMC_APIENTRY FmcjExecutionServiceSetTimeout(FmcjExecutionServiceHandle handle, long newValue);

C++ void SetTimeout(long newValue) const;

Java public abstract void SetTimeout(int newValue) throws FmcException;

例として、FmcjService::SetTimeout 関数 / メソッドがあります。これは、この FmcjService オブジェクトによって、クライアントから MQSeries Workflow サーバーに対して出される要求のタイムアウト値を設定します。言い替えると、クライアントが応答を待つ時間を設定します。

新しいタイムアウト値を設定すると、それはクライアントとサーバーとの間の通信を必要とするすべての関数 / メソッドで使われます。これは、任意の頻度に設定 (変更) できます。指定の単位はミリ秒です。負の値は -1 と解釈され、その場合、タイムアウト値は無限になります。

省略時タイムアウト値は、ユーザー・プロファイルである APITimeOut 値から取られます。それが見つからない場合、構成プロファイルから取られます。それも見つからない場合の省略時値は 180000 マイクロ秒です。

注: クライアント・サーバー呼び出しを発行して FMC_ERROR_TIMEOUT が戻された場合でも、MQSeries Workflow サーバーは正常に要求を処理している場合があります。しかし、その間に通信がタイムアウトであると報告されたため、サーバーは FMC_OK を戻すことができなかった可能性があります。要求が処理されなかった場合は、タイムアウトの設定値をもっと大きくしてからもう一度呼び出してみてください。

オブジェクト値のプロパティーの設定

この関数 / メソッドは、指定されたプロパティーを、指定されたオブジェクトに設定します。

Java のシグニチャー

```
public abstract void addProperty( Object value )
public abstract
void setContext( String args[], Properties properties )
public abstract
void setContext( Applet applet, Properties properties )
```

パラメーター

applet 入力。エージェントのインスタンスを生成したアプレット。通信プロトコルとして IIOP が使用されている場合、この情報は必須です。

args 入力。エージェント bean のインスタンスを生成したアプリケーションに渡されるコマンド行引き数。

properties 入力。インスタンスが生成された時点でアプリケーションまたはアプレットに渡された環境プロパティー。

value 入力。プロパティーの値。

宣言の例

Java

```
public abstract void addPropertyChangeListener(
PropertyChangeListener value );
```

オブジェクトの更新

この関数 / メソッドは、MQSeries Workflow サーバーから送信されてきた情報によって、指定されたオブジェクトを更新します。指定されたオブジェクトのために、更新情報が用意されていなければなりません。

サーバーは、作業項目 (使用不可でない限り)、アクティビティー・インスタンス通知、およびプロセス・インスタンス通知の更新情報をプッシュします。関連するプロセス・インスタンスのプロセス設定には、そのプロセス・インスタンス自体の REFRESH_POLICY PUSH を指定するか、省略時のプロセスを指定してください。ログオンは在席 (present) セッション・モードで実行している必要があります。

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjXxxUpdate( FmcjXxxHandle           handle,
                                   FmcjExecutionDataHandle data );
```

C++ 言語のシグニチャー

```
APIRET Update( FmcjExecutionData const & data );
```

パラメーター

handle 入力。更新するオブジェクトのハンドル。

data 入力。更新に使用するデータ。

戻りコード

C 言語の関数および MQSeries Workflow の結果オブジェクトは、次に示すコードを戻します。括弧内の数字はそれぞれの整数値です。

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)

パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスを受け取ることになっているにもかかわらず、0 が渡されました。

FMC_ERROR_EMPTY(122)

オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)

提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_OID(805)

実行データは、指定されたオブジェクトを更新できるデータではありません。この実行データは、指定されたオブジェクトに属していません。

FMC_ERROR_WRONG_KIND(501)

実行データは、指定されたオブジェクトを更新できるデータではありません。これは更新データではありません。

FMC_ERROR_INTERNAL(100)

MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

C 言語の例: 値へのアクセス

```
#include <stdio.h>
#include <fmcjcrun.h>
int main()
{
  APIRET                       rc;
  FmcjExecutionServiceHandle   service   = 0;
  FmcjWorkitemHandle           workitem  = 0;
  FmcjStringVectorHandle       sList     = 0;
  char                         category[FMC_CATEGORY_NAME_LENGTH+1];
  char                         generalBuffer[200];
  unsigned long                priority  = 0;
  int                          enumValue = 0;
  FmcjCDateTime                startTime;
  unsigned long                i         = 0;


 FmcjGlobalConnect();
```

```
/* logon */
FmcjExecutionServiceAllocate(&service);
rc = FmcjExecutionServiceLogon( service,
                                "USERID", "password",
                                Fmc_SM_Default, Fmc_SA_Reset
                              );


                                     /* set the timeout for requests */
FmcjExecutionServiceSetTimeout( service, 60000 );


/* assumption: workitem has been queried from the server             */
                                     /* access a value of type bool  */
if ( FmcjWorkitemCategoryIsNull( workitem ) )
  printf( "Category is not set¥n" );
else                                 /* access a value of type char  */
{                                    /* use a buffer which fits      */
  FmcjWorkitemCategory( workitem, category, FMC_CATEGORY_NAME_LENGTH+1 );
  printf( "Category   : %s¥n", category );
}


                                     /* access a date/time value     */
startTime= FmcjWorkitemStartTime( workitem );
printf( "Start time : %s¥n",
  FmcjDateTimeAsString(&startTime, generalBuffer, 200) );


                                     /* access a value of type long  */
priority = FmcjWorkitemPriority( workitem );
printf( "Priority   : %u¥n", priority );


                                    /* access an enumerated value   */
enumValue= FmcjWorkitemReceivedAs( workitem );
if ( enumValue == Fmc_IR_Normal )
  printf( "Received as: %s¥n","qualified user" );
...
                                     /* access a multi-valued field  */
 sList= FmcjWorkitemSupportTools( workitem );
 printf( "Support tools: " );
 for( i=0; i< FmcjStringVectorSize(sList); i++ )
 {                                   /* use a large buffer           */
   printf("%s ", FmcjStringVectorNextElement(sList, generalBuffer, 200) );
 }
```

```
  /* logoff */
  FmcjExecutionServiceLogoff(service);
  FmcjExecutionServiceDeallocate(&service);
  FmcjGlobalDisconnect();
  return FMC_OK;
}
```

C++ example: accessing values

```
#include <iomanip.h>
#include <bool.h>
#include <vector.h>
#include <fmcjstr.hxx>
#include <fmcjprun.hxx>
int main()
{
  FmcjGlobal::Connect();
   // logon
  FmcjExecutionService service;   APIRET rc = service.Logon("USERID", "password");
                                        // set the timeout for requests
  service.SetTimeout( 60000 );



  // assumption: workitem has been queried from the server
                                          // access a value of type bool
  if ( workitem.CategoryIsNull() )
    cout << "Category is not set" << endl;
  else                                    // access a value of type char
  {                                       // use a buffer which fits
    cout << "Category   : " << workitem.Category()<< endl;
  }



                                        // access a value of type date/time
  cout << "Start time : " << workitem.StartTime()<< endl;



                                       // access a value of type long
  cout << "Priority   : " << workitem.Priority()<< endl;



                                        // access an enumerated value
  FmcjItem::AssignReason reason= workitem.ReceivedAs();
  cout << "Received as: " <<
          ((reason == FmcjItem::Normal) ? "normal user" : "...")
       << endl;
```

```
  vector<string> tools; int j;          // access a multi-valued field
  workitem.SupportTools( tools );
  cout << "Support tools: " ;
  while ( j < tools.size() )
    cout << tools[j++] << " ";
   // logoff
  rc = service.Logoff();
   FmcjGlobal::Disconnect();
  return FMC_OK;
}                                       // destructors called automatically
```

アクション関数 / メソッド

アクション関数 / メソッドは、クライアント・サーバー呼び出しであり、これには MQSeries Workflow サーバーとの通信が関係しています。したがって、ログオンする必要があります。

アクション関数 / メソッドは、それぞれ永続オブジェクトを表しているサービス・オブジェクトおよび一時オブジェクトに対して発行できます。これらのオブジェクトは、ユーザー・セッションのコンテキストを記録して、MQSeries Workflow サーバーに至る通信パスを確立します。結果として、アクション呼び出しを発行するために空のオブジェクトは使用できないことになります。

アクション関数 / メソッドとは、サーバーの応答を待つ同期要求、または別の時点でサーバーが応答するのを待つ非同期要求、または MQSeries Workflow から情報を受信している関数 / メソッドです。

すべてのアクション関数 / メソッドについては、353ページの『第7部 プログラミング・インターフェース』で別個に説明します。例は、817ページの『第9部 例およびシナリオ』に示されています。

アクティビティー・インプリメンテーション関数 / メソッド

アクティビティーまたはサポート・ツールは、MQSeries Workflow API を使用するプログラムによってインプリメントできます。その場合、アクティビティー・インプリメンテーション関数 / メソッドにより、プログラム実行エージェントが実行中のプログラムを認識する際に使用しているプログラム識別のような情報へアクセスしたり、またはアクティビティー・インスタンスの対応する作業項目の入出力コンテナーやサポート・ツールの入力コンテナーへアクセスすることができます。また、アクティビティーをインプリメントするプログラ

ムでこれを使うことによって、更新後の出力コンテナーを MQSeries Workflow へ戻して、それらの値を基にしてナビゲーションを続けられるようにすることができます。

通常、アクティビティーまたはサポート・ツールをインプリメントしているプログラムは、特定の MQSeries Workflow 実行サーバーからの要求に応じて MQSeries Workflow プログラム実行エージェントの制御のもとで実行されます。 MQSeries Workflow 実行サーバーは、作業項目またはサポート・ツールの開始要求を受信すると、開始すべきインプリメント・プログラムを判別してから、該当する要求を、入出力コンテナーと一緒に (必要な場合) ログオン・ユーザーの MQSeries Workflow プログラム実行エージェントに送ります。コンテナーはプログラム実行エージェントに送られるので、入出力コンテナーは、インプリメント・プログラムによって MQSeries Workflow プログラム実行エージェントから要求され、そこに戻されます。アクティビティー・インプリメンテーションまたはサポート・ツールの内部からコンテナーを処理するために、実行サービス・オブジェクトを作成して MQSeries Workflow 実行サーバーへログオンする必要はありません。

しかし、コンテナーだけにアクセスするのではなく、たとえば作業項目が属するプロセス・インスタンスについての情報も照会したい場合には、プログラムの開始を要求した MQSeries Workflow 実行サーバーへログオンする必要があります。実行サービスの Passthrough() 関数 / メソッドを使用すれば、アクティビティー・インプリメンテーション・プログラムから、実行サーバーとのセッションを開始できます。このようにすれば、作業項目の環境を使用することができます。つまり、その他のユーザー ID、パスワード、システム・グループ、およびシステム情報は必要ありません。

MQSeries Workflow プログラム実行エージェントでは、同時に複数のプログラムを実行することができます。コンテナーが要求されると、それによって呼び出しプログラムが判別され、サーバーから送られてきたコンテナーが提供されてそのプログラムで使えるようになります。

アクティビティー・インプリメンテーションが単独で作業全体を処理するのではなく、(別個のオペレーティング・システム・プロセスとして実行される) サブプログラムの開始によって作業を分散する場合、それらのサブプログラムがコンテナーを要求すると、プログラム実行エージェントは呼び出しプログラムを見分けることができません。そのような場合のために、プログラム実行エージェントを呼び出すプログラムは、実際のアクティビティー・インプリメンテーションのプログラム識別子を提供しなければなりません。つまり、*遠隔* コンテナー呼び出しまたはパススルー呼び出しを使う必要があります。そのためには、アクティビティー・インプリメンテーションがプログラム識別子を取り出

して、開始されるプログラムにそれを渡す必要があります。プログラム実行エージェントがプログラム識別子を提供するのは、*信頼できる* プログラムに対してだけであることに注意してください。

アクセスに関する一般情報

すでに述べたように、アクティビティー・インプリメンテーションまたはサポート・ツールはプログラム実行エージェントに認識される際に使用されるプログラム識別 (Java ではクラス名 FmcjPEA または ExecutionAgent) を検索することができます。さらに、プログラム識別を認識しているアクティビティー・インプリメンテーションまたはプログラムは、起動されたアクティビティー・インプリメンテーションを使用することになっているユーザー、または関連するアクティビティー・インスタンスの永続オブジェクト識別を照会することができます。

注: ActiveX のシグニチャーはオブジェクト定義言語 (ODL) で提供されています。たとえば、*BSTR* 型は、VisualBasic の型が実際には String であるストリングに使用されます。

ActiveX のシグニチャー

```
BSTR ProgramID()
BSTR PersistentOidOfActivityInstance()
BSTR RemotePersistentOidOfActivityInstance( BSTR programID )
BSTR RemoteUserID( BSTR programID )
BSTR UserID()
```

C 言語のシグニチャー

```
char * FMC_APIENTRY FmcjPEAProgramID(
                            char *         buffer,
                            unsigned long  bufferLength )
char * FMC_APIENTRY FmcjPEAPersistentOidOfActivityInstance(
                            char *         buffer,
                            unsigned long  bufferLength )
char * FMC_APIENTRY FmcjPEARemotePersistentOidOfActivityInstance(
                            char const *   programID,
                            char *         buffer,
                            unsigned long  bufferLength )
char * FMC_APIENTRY FmcjPEARemoteUserID(
                            char const *   programID,
                            char *         buffer,
                            unsigned long  bufferLength )
char * FMC_APIENTRY FmcjPEAUserID(
                            char *         buffer,
                            unsigned long  bufferLength )
```

C++ 言語のシグニチャー

```
static string ProgramID()
static string PersistentOidOfActivityInstance()
static string RemotePersistentOidOfActivityInstance(
                                string const & programID )
static string RemoteUserID(   string const & programID )
static string UserID()
```

Java のシグニチャー

```
public abstract
String programID() throws FmcException
public abstract
String persistentOidOfActivityInstance() throws FmcException
public abstract
String remotePersistentOidOfActivityInstance( String programID )
throws FmcException
public abstract
String userID() throws FmcException
public abstract
String remoteUserID( String programID ) throws FmcException
```

パラメーター

buffer 入出力。検索する値を入れるバッファーを指すポインター。

bufferLength 入力。バッファーの長さ。最大限の値を入れるのに十分な大きさでなければなりません (最低限必要な長さについてはファイル fmcmxcon.h を参照)。 1 つのバッファーを使って、すべての文字値を取り出すことができます。

programID 入力。アクティビティー・インプリメンテーションがプログラム実行エージェントに認識されるのに必要なプログラム ID。

戻り値のデータ型

BSTR/char*/string/String
その値。

戻りコード

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、バッファーのアドレスを受け取ることになっているにもかかわらず、0 が渡されました。

FMC_ERROR_BUFFER(800)
提供されたバッファーが最大値を入れるには小さすぎます。必要な長さについては、fmcmxcon.h を参照してください。

FMC_ERROR_COMMUNICATION(13)
プログラム実行エージェントに接続できません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。また、検出されたすべての例外について MQSeries Workflow トレースを検査してください。

FMC_ERROR_INVALID_PROGRAMID(135)
指定したプログラム ID は無効または不明です。

FMC_ERROR_PROGRAM_EXECUTION(126)
関数 / メソッドはアクティビティー・インプリメンテーションまたはサポート・ツール内で呼び出されていません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。たとえば、アクティビティー・インプリメンテーションは承認されていないので、そのプログラム ID を検索できません。

FMC_ERROR_TOOL_FUNCTION(128)
関数 / メソッドはサポート・ツール内ではサポートされていません。

ダイナミック・リンク・ライブラリー

アクティビティー・インプリメンテーションまたはサポート・ツールのプログラムとしては、実行プログラム以外に、ダイナミック・リンク・ライブラリー (DLL) または共用ライブラリーも可能です。 DLL である場合、隔離モードまたは非隔離モードで実行できます。隔離モードの場合、その DLL は、プログラム実行エージェント・プロセスとは異なるオペレーティング・システム・プロセスで実行されます。非隔離モードの場合、その DLL は、プログラム実行エージェントの独自のオペレーティング・システム・プロセスで実行されます。

DLL のシグニチャーは次のようになります。 C++ では、*extern "C"* 構文を使用します。

C 言語のシグニチャー

```
int FMC_APIENTRY entryPoint( char const * arguments )
```

パラメーター

arguments
入力。プログラムに渡されるパラメーター。

FlowMark バージョン 2 互換モードの場合、DLL シグニチャーは次のようになります。

C 言語のシグニチャー

```
int FMC_APIENTRY entryPoint( char const * programID,
                             char const * arguments )
```

パラメーター

programID
入力。プログラム実行エージェントがプログラムを識別するプログラム ID (以前はセッション ID と呼ばれていたもの)。

arguments
入力。プログラムに渡されるパラメーター。

また、DLL では DLL 初期化ルーチンや DLL 終了ルーチンも指定できます。プログラム実行エージェントが DLL をロードするとただちに、DLL は (使用可能であれば) DLL 初期化エントリー・ポイントを呼び出します。そしてプロ

グラム実行エージェントが DLL をアンロードする直前に、DLL は (使用可能であれば) DLL 終了エントリー・ポイントを呼び出します。

C 言語シグニチャー

```
void FmcDllInit()
```

C 言語シグニチャー

```
void FmcDllTerm()
```

たとえば、ロードされた状態の非分離 DLL の場合を考えます。初期化によって獲得された資源は終了ルーチンによって解放されるまでの間、DLL の存続時間を通じて保持されます。一度限り獲得するオブジェクトの例として、リソース・マネージャーやオープン・ファイル・ハンドルとのセッションがあります。

アクティビティー・インプリメンテーションの例については、857ページの『第77章 アクティビティー・インプリメンテーション』または″API Programming Examples″ サポート・パックを参照してください。

プログラム実行管理関数 / メソッド

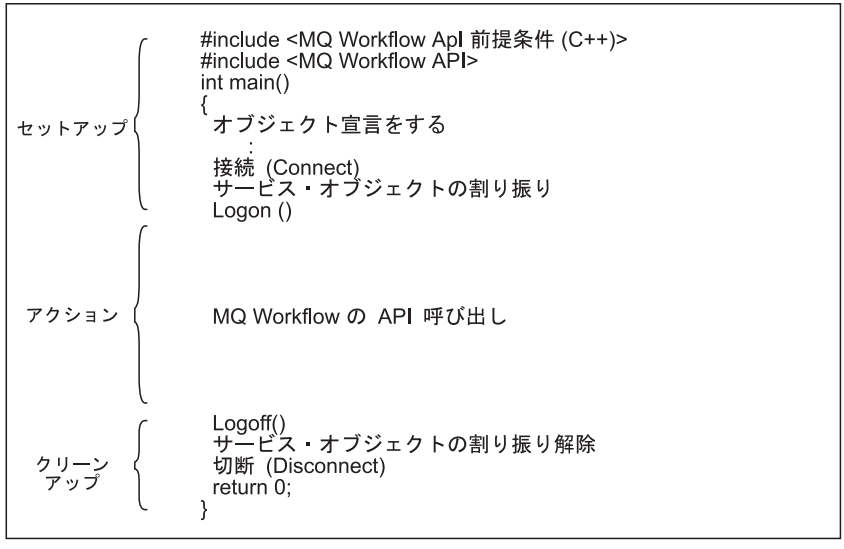
プログラム実行管理関数 / メソッドは、MQSeries Workflow プログラム実行エージェントの管理に使われます。それによって、ユーザーに関連したプログラム実行エージェントを開始または停止 (シャットダウン) できます。

第2部 C および C++ API

第 2 部では、MQSeries Workflow の C 言語および C++ API に固有な概念について概説します。

第12章 MQSeries Workflow クライアント・アプリケーション

多くの場合、MQSeries Workflow C または C++ クライアント・アプリケーションには、以下に示す部分が含まれています (これほど明確に分かれているとは限りません)。



多くの場合、プログラムをセットアップするには、使用するプログラム変数またはオブジェクトを宣言し、アクションに必要な MQSeries Workflow API ヘッダー・ファイルを組み込みます。 C++ API を使用する場合は、*ブール (bool)*、*文字列 (string)*、およびベクトル *(vector)* の定義が必要です。それぞれのファイルを、MQSeries Workflow API ヘッダーの前に組み込みます。

次に、Connect() 関数 / メソッドを呼び出すことによって MQSeries Workflow API を初期化して、API が保有するリソースが正しく割り振られるようにする必要があります。 Connect() および Disconnect() を各スレッドの先頭と末尾で呼び出さなければなりません。

次に、接続先のサーバーを表すサービス・オブジェクトを割り振る必要があります。サービス・オブジェクトの割り振りが完了したら、ログオンすることができます。ログオンによって、ログオンするユーザーと、サービス・オブジェ

クトで表されるサーバーとの間のセッションが確立されます。クライアント / サーバー通信を必要とするそれ以降の呼び出しは、すべてこのセッションを通して実行されます。

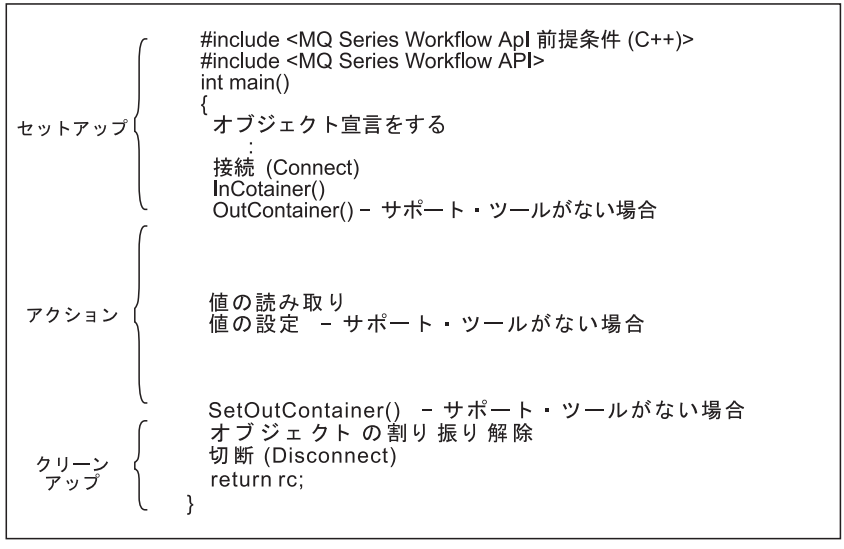
正常にログオンしたなら、アクションまたはプログラム実行管理関数 / メソッドを発行することによって、自分が許可されている MQSeries Workflow オブジェクトを照会したり管理することができます。

プログラムの終わりではログオフし、サーバーとのセッションをクローズし、そしてプログラムで確保していたすべてのリソース (特にサービス・オブジェクト) を割り振り解除します。

最後のステップとして MQSeries Workflow API を切断し、API が保持していたリソースが正しく割り振り解除されるようにします。

第13章 MQSeries Workflow アクティビティー・インプリメンテーションまたはサポート・ツール

多くの場合、MQSeries Workflow C または C++ アクティビティー・インプリメンテーションまたはサポート・ツールのインプリメンテーションには以下に示す部分が含まれています。



多くの場合、プログラムをセットアップするには、使用するプログラム変数またはオブジェクトを宣言し、アクションに必要な MQSeries Workflow API ヘッダー・ファイルを組み込みます。 C++ API を使用する場合は、*ブール (bool)*、*文字列 (string)*、およびベクトル *(vector)* の定義が必要です。それぞれのファイルを、MQSeries Workflow API ヘッダーの前に組み込みます。

次に、Connect() 関数 / メソッドを呼び出すことによって MQSeries Workflow API を初期化して、API が保有するリソースが正しく割り振られるようにする必要があります。 Connect() および Disconnect() を各スレッドの先頭と末尾で呼び出さなければなりません。

これでアクティビティー・インプリメンテーションは、このプログラムを開始した MQSeries Workflow プログラム実行エージェントからアクティビティー

の入出力コンテナーを取り出すことができます。サポート・ツールが検索できるのは、アクティビティーの入力コンテナーだけです。

コンテナーにアクセスできる場合、プログラミング論理に従って値を読み取ったり設定したりできます。

プログラムの終わりにアクティビティー・インプリメンテーションは、最終出力コンテナーを MQSeries Workflow プログラム実行エージェントに戻します。プログラムで保持されていたすべてのリソースは割り振り解除されます。プログラムの戻り値が、プログラムの総体的な結果をプログラム実行エージェントに知らせます。

プログラムの戻りコードに加えて出力コンテナーが、アクティビティー・インプリメンテーションの実行を要求した MQSeries Workflow サーバーに送り返されます。戻りコード (_RC) を出口ルーチンまたは変換条件において使用するなら、MQSeries Workflow ナビゲーションに役立ちます。[5]

最後のステップとして MQSeries Workflow API を切断し、API が保持していたリソースが正しく割り振り解除されるようにします。

アクティビティー・インプリメンテーションもまた、クライアント・アプリケーションと同じ動作をして (173ページの『第18章 MQSeries Workflow クライアント・アプリケーション』を参照)、MQSeries Workflow サーバーのサービスを要求することができます。ただし、通常は、それは実行の起動元のサーバーです。次に、Logon() 関数 / メソッドの代わりに Passthrough() 関数 / メソッドを使い、作業項目の開始要求からサーバーに認識されているユーザー識別子と権限によって、プログラム実行の原因となったサーバーにログオンします。

5. アプリケーションの終了コードをサポートしないコンパイラーでは、出力コンテナーの _RC データ・メンバーを設定することができます。

第14章 コンパイルとリンク

MQSeries Workflow で使用するために開発するすべての C++ および C 言語プログラムは、MQSeries Workflow から提供されるヘッダー・ファイルを組み込んでいなければならず、対応するライブラリー・ファイルとリンクしなければなりません。これらのファイルは、MQSeries Workflow 開発キットのインストールを選んだ場合にシステムにインストールされます。省略時解釈では、次のようにインストールされます。

- AIX(R) の場合、/usr/lpp/fmc/api ディレクトリーにヘッダー・ファイル。/usr/lpp/fmc/lib ディレクトリーに共用ライブラリー・ファイル。
- HP-UX の場合、/opt/fmc/api ディレクトリーにヘッダー・ファイル。/opt/fmc/lib ディレクトリーに共用ライブラリー・ファイル。共用ライブラリーは /usr/lib にリンクされます。
- OS/2 の場合、選択したドライブの ¥fmcos2¥api ディレクトリーに。
- Solaris の場合、/opt/fmc/api ディレクトリーにヘッダー・ファイル。/opt/fmc/lib ディレクトリーに共用ライブラリー・ファイル。共用ライブラリーは /usr/lib にリンクされます。
- Windows NT または Windows 200の場合、選択したドライブの ¥fmcwinnt¥api ディレクトリーに。
- Windows 95 の場合、選択したドライブの ¥fmcwin95¥api ディレクトリーに。
- Windows 98 の場合、選択したドライブの ¥fmcwin98¥api ディレクトリーに。

MQSeries Workflow C++ API を使用する場合は、ブール、文字列、およびベクトルの定義を用意する必要があります。これらの定義をサポートするコンパイラーの場合、ご使用のコンパイラーの定義を使ってください。該当するファイルを MQSeries Workflow C++ API ヘッダーの前に組み込みます。これらの定義のいずれをもサポートしないコンパイラーの場合、 MQSeries Workflowは API の stl サブディレクトリーに特定のファイルが組み込まれるようにします。それらのファイルは、*bool.h* (ブール定義用、最初に組み込むことが必要)、*fmcjstr.hxx* (文字列定義用)、および *vector.h* (ベクトル定義用) です。サポートされるコンパイラーに組み込む必要のある定義については、158ページの『C++ の前提条件ヘッダー・ファイル』を参照してください。

bool.h と vector.h は MQSeries Workflow に付属するものであり、Hewlett-Packard 社が著作権を持っている Standard Template Library の一部です。このライブラリーの解説書は、STLDOC.PS という名前のファイルとして MQSeries Workflow CD-ROM で提供されています。これは、API の stl サブディレクトリーにインストールされます。

注: Windows 環境において、MQSeries Workflow は ANSI コード・ページのすべての入力を解釈します。つまり、MQSeries Workflow が印刷可能文字をテストする際に違いが生じる場合があります。たとえばアプリケーションが isprint() のような関数を使って印刷可能文字をテストする場合などです。

組み込むヘッダー・ファイルはご使用の MQSeries Workflow 機能に応じて決まり、またリンクするライブラリー・ファイルはご使用のコンパイラーによって決まります。使用する機能に応じて、次のヘッダー・ファイルを組み込む必要があります。

機能	C-API ヘッダー	C++ ヘッダー
実行機能クライアント	fmcjcrun.h	fmcjprun.hxx
実行機能アクティビティー・インプリメンテーション:		
- コンテナー・アクセスのみ	fmcjccon.h	fmcjpcon.hxx
- コンテナーおよびサーバーのアクセス	fmcjcrun.h	fmcjprun.hxx
実行機能サポート・ツール		
- コンテナー・アクセスのみ	fmcjccon.h	fmcjpcon.hxx
- コンテナーおよびサーバーのアクセス	fmcjcrun.h	fmcjprun.hxx

MQSeries Workflow のダイナミック・リンク・ライブラリーは、それぞれ次のように分割されています。

fmcjdcom 共通機能が含まれており、常にリンクしなければなりません。
fmcjdcbr テンプレートと永続リストが含まれています。
fmcjdcon コンテナー機能が含まれています。
fmcjdrun 実行機能用の機能だけが含まれています (つまりプロセス・インスタンス、作業項目、通知、およびインスタンス・モニターを処理します)。

したがって、リンクするライブラリー (.lib ファイル) は次のようになります。

機能	fmcjdcom	fmcjdcbr	fmcjdcon	fmcjdrun
実行機能クライアント	x	x	x	x

機能	fmcjdcom	fmcjdcbr	fmcjdcon	fmcjdrun
実行機能アクティビティー・インプリメンテーション:				
- コンテナー・アクセスのみ	x		x	
- コンテナーおよびサーバーのアクセス	x	x	x	x
実行機能サポート・ツール				
- コンテナー・アクセスのみ	x		x	
- コンテナーおよびサーバーのアクセス	x	x	x	x

C++ および C 言語 MQSeries Workflow API にアクセスするアプリケーションのコンパイルおよびリンクには、一般に使用されているコンパイラーであればどれでも使用できます。ご使用になるコンパイルおよびリンクのオプションは、必ず FMC_APIENTRY マクロの中で定義されている呼び出し規則で MQSeries Workflow API が呼び出されるものでなければなりません (ファイル fmcjcglo.h を参照)。 FMC_APIENTRY は、C の標準呼び出し規則に合わせて定義されており、MQSeries Workflow から提供されるヘッダー・ファイルを使うときは自動的に適用されなければなりません。マルチスレッド・ライブラリーを使う必要があります。

C 呼び出しをサポートする言語であればどの言語からであっても C 言語の関数を呼び出すことができます。 C++ API には、一般的に使用されているどの C++ コンパイラーからでもアクセスできます。 C++ API はソース・コードとして渡されるからです (インライン・メソッド)。

サポートされるコンパイラー

しかし、保守サービスに関してサポートされるのは、以下に示すコンパイラーおよび環境だけです。

- AIX の場合:
  - IBM C Set++(R) バージョン 3.1.4
  - IBM VisualAge C++ Professional for AIX, Versions 4.0 and 5.0
  - IBM C for AIX, Version 5.0
- HP-UX の場合、HP aC++ Compiler S700 バージョン A.01.15.01
- OS/2 の場合、IBM VisualAge(R) for C++ バージョン 3.0 および 4.0
- Solaris の場合:
  - Sun WorkShop Compiler, Versions 4.2 and 5.0
  - C++ の場合のみ、Kuck&Associates Inc. KAI C++ Version 3.3
- Windows プラットフォームの場合:

– IBM VisualAge for C++, Versions 3.5 and 4.0
– Microsoft Visual C++, Versions 5.0 and 6.0

C++ の前提条件ヘッダー・ファイル

次の表は、C++ API に関して、サポートされるコンパイラーがブール、文字列、ベクトルを提供するかどうか (コンパイラー・タイプまたはコンパイラー付属の include)、それとも MQSeries Workflow 提供の定義を使用する必要があるかを示します。

プラットフォーム	ブール	ベクトル / 文字列
AIX C Set++ 3.1.4	MQSeries Workflow	MQSeries Workflow
AIX IBM VA 4.0/5.0	コンパイラー・タイプ	コンパイラーの include
HP-UX	コンパイラー・タイプ	コンパイラーの include
OS/2 IBM VA 3.0	MQSeries Workflow	MQSeries Workflow
OS/2 IBM VA 4.0	コンパイラー・タイプ	コンパイラーの include
Solaris	コンパイラー・タイプ	コンパイラーの include
Windows IBM VA 3.5	MQSeries Workflow	MQSeries Workflow
Windows IBM VA 4.0	コンパイラー・タイプ	コンパイラーの include
Windows MSVC	コンパイラー・タイプ	コンパイラーの include

コンパイル・ステートメントの例

コンパイル・ステートメントの例を示します。

- AIX の場合:

```
xlC_r -o <object file> -I/usr/lpp/fmc/api -L/usr/lpp/fmc/lib
                              -l<MQ Workflow libs> <source file>
```

- HP-UX の場合:

```
aCC -D_THREAD_SAFE -DRWSTD_MULTI_THREAD -D_REENTRANT
    -o <object file> -I /opt/fmc/api -l<MQSeries Workflow libs> <source file>
```

- OS/2 および IBM VisualAge for C++ 3.0 (最小必須スタック・サイズは 64000):

```
icc /GM+ /Su4 /B"/STACK:64000" <optional parameters> <source file>
```

- OS/2 および IBM VisualAge for C++ 4.0 の場合 (マルチスレッド・ライブラリーが使用されていること、および 4 バイトの enum サイズが指定されていることを確認)。
- Solaris と C 言語の場合:

```
cc -o <object file> -I /opt/fmc/api -l<MQSeries Workflow libs> <source file>
```

- Solaris および Sun Workshop C++ Compiler の場合:

```
CC -mt -o <object file> -I /opt/fmc/api -l<MQ Workflow libs> <source file>
```

- Solaris および KAI C++ Compiler の場合:

```
KCC --thread_safe
    -o <object file> -I /opt/fmc/api -l<MQSeries Workflow libs> <source file>
```

- Windows プラットフォームおよび Microsoft Visual C++ 5.0 または 6.0 の場合:

```
cl -MD <optional parameters> <source file>
```

- Windows プラットフォームと IBM VisualAge for C++ 3.5 の場合:

```
icc /GM+ /Su4 <optional parameters> <source file>
```

- Windows プラットフォームおよび IBM VisualAge for C++ 4.0 の場合 (マルチスレッド・ライブラリーが使用されていること、および 4 バイトの enum サイズが指定されていることを確認):

第15章 メモリー管理

ワークフロー・プロセス・モデル、そのインスタンス、およびその結果の作業項目は、すべて永続的に MQSeries Workflow データベースに保管されるオブジェクトです。つまり、これらはアプリケーション・プログラムとは独立して存在するということです。

永続オブジェクトがアプリケーション・プログラムから照会されると、永続オブジェクトは照会時における永続オブジェクトの状態を持つ*一時オブジェクト*として表されます。複数の照会が発行されると、同一の永続オブジェクト (場合によってはそのオブジェクトのさまざまな状態) を表す複数の一時オブジェクトが存在することになります。

一時オブジェクトとそのメモリーの存続期間は、プログラマーが*完全に管理* します。それらがもう必要なくなる時期、つまりオブジェクトを割り振り解除したり (C 言語) 破棄したり (C++) すべき時期は、プログラマー自身が一番よく知っているからです。しかし、アプリケーション・プログラムが終了すると、一時オブジェクトはもう使えなくなります。

特定の一時オブジェクトは、プログラマーが*明示的に割り振り*ます。そのようなオブジェクトはサポート・オブジェクトであり、永続オブジェクトを反映しません。その例として、代行者として一連の担当者を指定する場合の FmcjStringVector (C-API) や実行サーバーからサービスを要求できるようにするための FmcjExecutionService オブジェクトがあります。

永続オブジェクトを反映する一時オブジェクトは、永続オブジェクトを (たとえば照会によって) 作成または検索するときに*暗黙的に割り振られ*ます。

一時オブジェクトの存続期間は完全にプログラマー自身が管理しますが、その実際の内部オブジェクト構造は、MQSeries Workflow API によってカプセル化されます。オブジェクトに対する要求を発行できるよう、MQSeries Workflow API にはハンドルが用意されています (C 言語)。 C++ API では、そのハンドルがクラスの唯一のデータ・メンバーです。したがって、プログラマーは内部的な変更にかかわりません。しかもそれによって MQSeries Workflow は、サーバーから渡されたオブジェクトの集まりを遅延読み取りすることができるので、パフォーマンスが向上します。

MQSeries Workflow API は、*契約によるプログラミング* の概念に従っています。つまり、0 (ヌル) でないハンドルが渡されると、それはオブジェクトへのアクセスに使える有効なハンドルであるという前提になっています。このような前提は、特に照会の場合に注意する必要があります。 0 以外のベクトル・ハンドルはオブジェクトの既存のベクトルを指すと想定され、新たにオブジェクトを追加するのに使われます。言い換えると、**新しいハンドルはいずれも 0 に初期化しなければなりません。**

すべてのリソース・メモリーは最終的にアプリケーション・プロセスそのものによって所有されることになるので、そのプロセス内のさまざまなスレッドからすべてのオブジェクトにアクセスすることができます。 MQSeries Workflow でスレッドを使用できないということはありません。これは、再入可能なものとしてコーディングされます。ただし、MQSeries Workflow はスレッドを明示的にサポートしてはいません。さまざまなスレッドから同じ一時オブジェクトにアクセスしたい場合は、そのオブジェクトでのアクセスを同期化する必要があります。オブジェクトはスレッド・セーフでは**ありません。**

第16章 結果オブジェクト

一般に、結果オブジェクトは、最後に実行した MQSeries Workflow API 要求 (該当するスレッド内) の結果を示します。特にこれは、エラー状態をさらに詳細に分析するのに使うことができ、次のような情報が入れられます。

- 戻りコード。
- 結果の発生元。つまり、結果が書き込まれる原因となったファイルと、エラーまたは要求完了が発生した行および関数。
- 関係するオブジェクトを記述したパラメーター (5 個まで)。

この結果は、形式化メッセージ・テキストとして取り出すことができ、そのテキストにはすべてのパラメーターが付け加えられています。そのメッセージ・テキストの作成時には現行のロケールが考慮に入れられ、選択されている言語でメッセージが示されるようになっています。

MQSeries Workflow は、オブジェクトの同期の管理に関してはスレッドを明示的にサポート**しません**が (プログラマーの責任)、MQSeries Workflow でスレッドの使用が禁止されているわけでありません。スレッドごとに結果オブジェクトが用意されるのはそのためです。

関数 / メソッドの呼び出しのすべての結果は、要求が実行されるスレッドに関連付けられる結果オブジェクトに書き込まれます。
FmcjResult::ObjectOfCurrentThread() メソッドあるいは
FmcjResultObjectOfCurrentThread() 関数を使って、スレッドごとに 1 回結果オブジェクトにアクセスすれば十分です。結果オブジェクトは、要求ごとに自動的に更新されます。

結果オブジェクトは、スレッドで最初の MQSeries Workflow API 呼び出しが出された時点で自動的に MQSeries Workflow によって割り振られます。これにはいつでもアクセスでき、また任意の頻度でアクセスできます。

たとえば、C 言語の場合、次のようにして結果オブジェクトにアクセスして使うことができます。

```
#include <stdio.h>
#include <fmcjcrun.h>
int main()
{
  FmcjResultHandle        result      = 0;
  FmcjStringVectorHandle  parms       = 0;
  char                    buffer[2000]= "";
  result= FmcjResultObjectOfCurrentThread();
  printf( "Accessed result object of current thread¥n" );
  printf( "Return code: %i¥n", FmcjResultRc(result) );
  printf( "Text       : %s¥n", FmcjResultMessageText(result,buffer,2000) );
  printf( "Origin     : %s¥n", FmcjResultOrigin(result,buffer,2000) );
  parms= FmcjResultParameters(result);
  while ( 0 != FmcjStringVectorNextResultParmElement( parms, buffer, 2000 ) )
    printf( "Parameter  : %s¥n", buffer );
  return 0;
}
```

注: 文字列ベクトルに対して NextResultParmElement() 関数が使われているのは、パラメーターの読み取り中に結果オブジェクトが変更されないようにするためです。

たとえば、C++ 言語の場合、次のようにして結果オブジェクトにアクセスして使うことができます。

```
#include <iomanip.h>
#include <bool.h>
#include <vector.h>
#include <fmcjstr.hxx>
#include <fmcjprun.hxx>
int main()
{
  vector<string> parms;
  FmcjResult *pResult = FmcjResult::ObjectOfCurrentThread();
  cout << "Accessed result object of current thread" << endl;
  cout << "Return code: " << pResult->Rc()           << endl;
  cout << "Text       : " << pResult->MessageText() ;
  cout << "Origin     : " << pResult->Origin()       << endl;
  pResult->Parameters(parms);
  cout << "Parameter  : ";
           for (int i=0; i<parms.size(); i++)
           {
             cout << parms[i] << " ";
           }
  cout << endl;
  delete pResult;                      // cleanup object from heap
  return 0;
}
```

注: 結果オブジェクトの C++ の一時表現は、他のすべてのオブジェクトと同じように破棄されます。結果オブジェクトを取り出すたびに、別々の表現が作成されます。

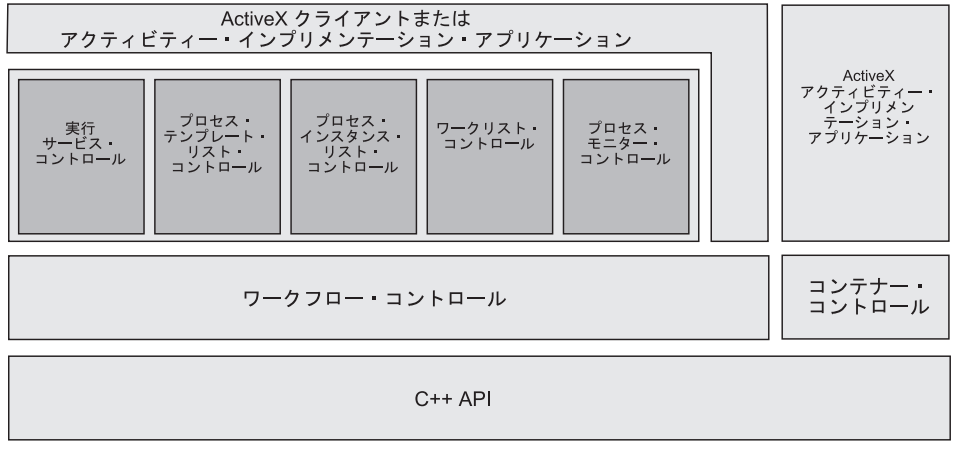
第3部 ActiveX コントロール

第 3 部では、MQSeries Workflow ActiveX コントロールについて概説しています。

第17章 構成要素の概説

MQSeries Workflow には、クライアント・アプリケーション・プログラムまたはアクティビティー・インプリメンテーションやサポート・ツールの作成に使用できる ActiveX コントロールがいくつか付属しています。提供されるコントロールには以下のものがあります。

- IBM MQSeries Workflow コントロール 3.1
- IBM MQSeries ExecutionService コントロール 3.1
- IBM MQSeries ProcessTemplateList コントロール 3.1
- IBM MQSeries ProcessInstanceList コントロール 3.1
- IBM MQSeries Worklist コントロール 3.1
- IBM MQSeries ProcessMonitor コントロール 3.1
- IBM MQSeries Container コントロール 3.1



ActiveX API は C++ API の上にインプリメントされていて、ActiveX コントロールの実行サーバーに対するアクセス層になります。 Workflow Control と Container コントロールは、C++ API 層に対する OLE インターフェースです。 Workflow コントロールの上には、Container コントロールを除くすべてのコントロールがあります。 Container コントロール以外のすべてのコントロールには、実行機能 GUI に加えて設計時 GUI も含まれています。 Container コントロールは、コンテナーにアクセスしようとしているアクティビティー・インプリメンテーションやサポート・ツールで使用できます。 MQSeries Workflow 標準実行機能クライアント自体が、提供されている ActiveX コント

ロールを使ってインプリメントされていることに注意してください (817ページの『第9部 例およびシナリオ』も参照)。

機能の概説

Workflow コントロールは、Visual Basic ユーザー・アプリケーション内で以下のように動作します。

- Visual Basic ユーザー・アプリケーションには、多くの場合、1 つの (非ビジュアル) Workflow コントロールが含まれています。
- Workflow コントロールには 1 つの ExecutionServiceArray が含まれています。
- ExecutionServiceArray には複数の ExecutionService が含まれることがあり、各 ExecutionService は 1 つの MQSeries Workflow 実行サーバーに接続しています。
- 各 ExecutionService には、各リスト・タイプ (ProcessTemplateList、ProcessInstanceList、および Worklist) ごとに 1 つずつ配列が含まれています。
- それらの配列には複数のオブジェクトが含まれることがあります。つまり、WorklistArray には複数のワークリストを含めることが可能であり、そのおのおのを Worklist コントロールに接続できます。
- 1 つまたは複数の (ビジュアル) ExecutionService コントロールを Workflow コントロールに接続し、使用可能な実行サービスに関する特定の情報を表示することができます。
- 1 つまたは複数の (ビジュアル) コントロールを Workflow コントロールに接続し、特定のリストのオブジェクトを表示することができます。

Workflow コントロールの概説

Workflow コントロールには固有のオブジェクトがいくつか含まれています。それらのオブジェクトはコントロールにより直接保守されます。ExecutionServiceArray オブジェクトは、任意の数の ExecutionService オブジェクトを作成し保守できます。各 ExecutionService オブジェクトは、1 つの MQSeries Workflow 実行サーバーへの参照を処理します。

各 **ExecutionService** には、**ProcessTemplateListArray**、**ProcessInstanceListArray**、および **WorklistArray** が 1 つずつ含まれています。各配列ごとに、配列要素の追加、取り出し、削除を実行するメソッドや、配列中の項目数を決定するためのメソッドがあります。

さらに、Workflow コントロールは 1 つの **StringArray** オブジェクトを保守します。この保守されているオブジェクトは、たとえば特定の作業項目に関してサポート・ツールを照会する場合 (**Workitem::SupportTools**) や、担当者 ID (PersonID) のリストを照会する場合 (**Workitem::Staff**) に使用されます。

また、AssignReason、Kind、または State などの列挙型もいくつか含まれています。たとえば、State にはあるアイテムの現在の状態に対応するエントリーが含まれます (Ready、Running、Disabled、Suspended など)。

ExecutionService の処理方法

ExecutionService オブジェクトを処理するには、WorkFlow コントロール OCX へのアクセス権がプログラムになければなりません。 Visual Basic のプログラミング環境の場合、これを実現するには、使用可能なフォームの 1 つに特定の OCX を埋め込みます。 ExecutionServiceArray には、Workflow コントロールによってアクセスできます。 **Add** メソッドか **AddDefault** メソッドを使用することによって、新しく ExecutionService を作成できます。新しく作成した ExecutionService オブジェクトに対するアクセスは、**GetAt** メソッドによって得られます。

新しい ExecutionService オブジェクトに対するアクセス権があれば、このオブジェクトに含まれているメソッドをすべて発行できます。このプロセスには GUI は関係していません。

リストの処理方法

リスト・コントロールを処理するには、ExecutionService オブジェクトが作成済みでなければなりません。たとえば、見ることが許可されているすべてのワークリストにアクセスするためには、ExecutionService が管理している WorklistArray にデータを入れなければなりません。そのためには、**QueryWorklists()** メソッドを呼び出します。その後、**WorklistArray()** メソッドを使用することによって、Worklist オブジェクトを含むオブジェクトへのアクセス権を取得しなければなりません。個々の worklist オブジェクトへのアクセス権を取得するには、WorklistArray の **GetAt()** メソッドを使うこともできます。そのほかのリストも、すべて同じように扱われます。詳しくは、34ページの『ActiveX の配列』を参照してください。このプロセスには GUI は関係していません。

ProcessTemplateList コントロールの概説

ProcessTemplateList コントロールは、特定のリストを介して表示できる ProcessTemplate オブジェクトを保守します。配列にデータを入れるには、ProcessTemplateList クラスの **QueryProcessTemplates()** メソッドを使用します。 **GetSize()** を使用すると、リスト内のアイテム数を調べることができます。また特定の ProcessTemplate オブジェクトを処理するには **GetAt()** メソッドを使用します。

ProcessInstanceList コントロールの概説

ProcessInstanceList コントロールは、特定のリストを介して表示できる ProcessInstance オブジェクトを保守します。配列にデータを入れるには、ProcessInstanceList クラスの **QueryProcessInstances()** メソッドを使用します。 **GetSize()** を使用すると、リスト内のアイテム数を調べることができます。また特定の ProcessInstance オブジェクトを処理するには **GetAt()** メソッドを使用します。

Worklist コントロールの概説

Worklist コントロールは、特定のリスト (作業項目、アクティビティー・インスタンス通知、またはプロセス・インスタンス通知) を介して表示できるオブジェクトを保守します。保守するオブジェクトは、ActivityInstanceNotifArray、ProcessInstanceNotifArray、および WorkitemArray です。

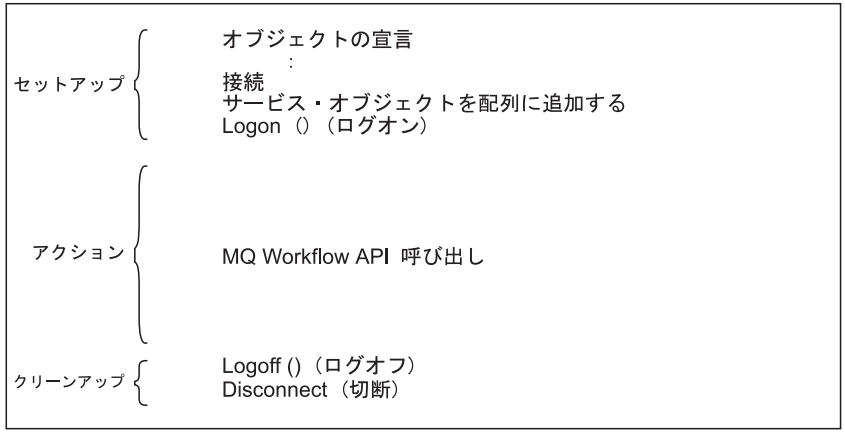
このうち、たとえば WorkitemArray オブジェクトは、任意の数の作業項目オブジェクトを作成して保守できます。配列要素の追加、取り出し、削除を実行するメソッドや、配列中の項目数を決定するためのメソッドがあります。

Monitor コントロールの概説

Monitor コントロールは、プロセス・インスタンスまたはアクティビティー・インスタンスに対するモニターを表します。モニターにアクセスするには、**ObtainMonitor()** メソッドを使用します。

第18章 MQSeries Workflow クライアント・アプリケーション

多くの場合、MQSeries Workflow ActiveX クライアント・アプリケーションには、以下に示す部分が含まれています (これほど明確に分かれているとは限りません)。



プログラムをセットアップするには、MQSeries Workflow コントロールが VisualBasic フォームの上になければなりません。多くの場合、使用するプログラム変数またはオブジェクトを宣言します。

次に、Connect() メソッドを呼び出すことによって MQSeries Workflow API を初期化して、API が保有するリソースが正しく割り振られるようにする必要があります。 Connect() および Disconnect() を各スレッドの先頭と末尾で呼び出さなければなりません。

次に、接続先のサーバーを表すサービス・オブジェクトを割り振る必要があります。そのためには、その目的のために提供されている実行サービス配列にオブジェクトを追加します。サービス・オブジェクトの割り振りが完了したら、ログオンすることができます。ログオンによって、ログオンするユーザーと、サービス・オブジェクトで表されるサーバーとの間のセッションが確立されます。クライアント / サーバー通信を必要とするそれ以降の呼び出しは、すべてこのセッションを通して実行されます。

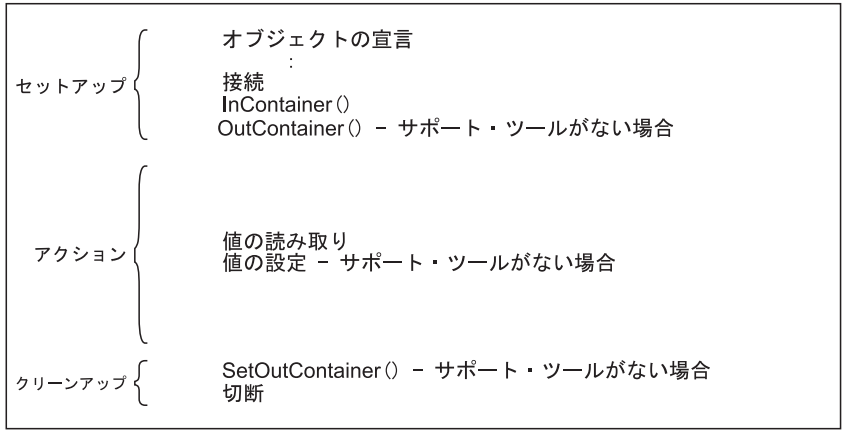
正常にログオンしたなら、アクションまたはプログラム実行管理メソッドを発行することによって、自分が許可されている MQSeries Workflow オブジェクトを照会したり管理することができます。

プログラムの終わりではログオフし、サーバーとのセッションをクローズします。

最後のステップとして MQSeries Workflow API を切断し、API が保持していたリソースが正しく割り振り解除されるようにします。

第19章 MQSeries Workflow アクティビティー・インプリメンテーションまたはサポート・ツール

多くの場合、MQSeries Workflow ActiveX アクティビティー・インプリメンテーションまたはサポート・ツールのインプリメンテーションには以下に示す部分が含まれています。



プログラムをセットアップするには、Container コントロールが VisualBasic フォームの上になければなりません。多くの場合、使用するプログラム変数またはオブジェクトを宣言します。

次に、Connect() メソッドを呼び出すことによって MQSeries Workflow API を初期化して、API が保有するリソースが正しく割り振られるようにする必要があります。 Connect() および Disconnect() を各スレッドの先頭と末尾で呼び出さなければなりません。

これでアクティビティー・インプリメンテーションは、このプログラムを開始した MQSeries Workflow プログラム実行エージェントからアクティビティーの入出力コンテナーを取り出すことができます。サポート・ツールが検索できるのは、アクティビティーの入力コンテナーだけです。

コンテナーにアクセスできる場合、プログラミング論理に従って値を読み取ったり設定したりできます。

プログラムの終わりにアクティビティー・インプリメンテーションは、最終出力コンテナーを MQSeries Workflow プログラム実行エージェントに戻します。出力コンテナーの _RC 値が、プログラムの総体的な結果を実行サーバーに知らせます

出力コンテナーが、アクティビティー・インプリメンテーションの実行を要求した MQSeries Workflow サーバーに送り返されます。戻りコード (_RC) を出口ルーチンまたは変換条件において使用するなら、MQSeries Workflow ナビゲーションに役立ちます。

最後のステップとして MQSeries Workflow API を切断し、API が保持していたリソースが正しく割り振り解除されるようにします。

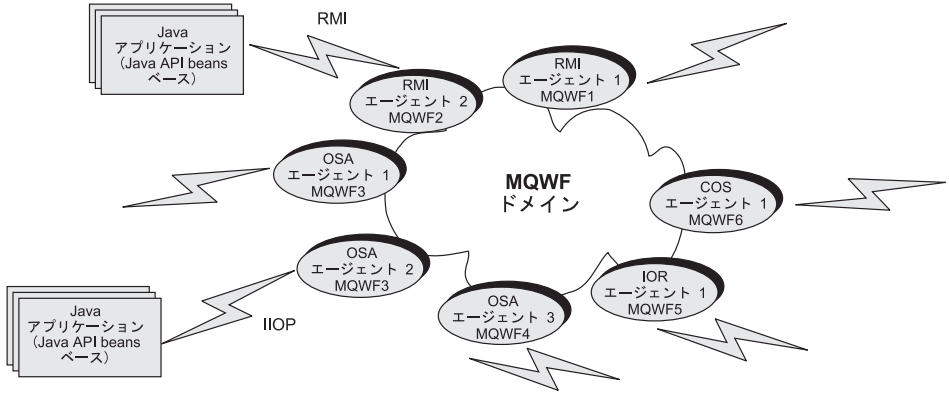
アクティビティー・インプリメンテーションもまた、クライアント・アプリケーションと同じ動作をして (151ページの『第12章 MQSeries Workflow クライアント・アプリケーション』を参照)、MQSeries Workflow サーバーのサービスを要求することができます。ただし、通常は、それは実行の起動元のサーバーです。次に、Logon() メソッドの代わりに Passthrough() メソッドを使い、作業項目の開始要求からサーバーに認識されているユーザー識別子と権限によって、プログラム実行の原因となったサーバーにログオンします。

第4部 JAVA API

MQSeries Workflow Java API は、次のもので構成されています。

- MQSeries Workflow ドメインを Java 環境に接続するエージェント。
- Java アプリケーションに MQSeries Workflow API 機能を提供する API Beans のセット。

第20章 Java CORBA エージェント



シン・クライアント をサポートするために、Java エージェント方式が取られています。 Java CORBA エージェントは、MQSeries Workflow ドメインのプロキシーとしての役割を果たします。

Java CORBA エージェントは Java でインプリメントされており、MQSeries Workflow C++ API を、 Java Native Interface (JNI) 経由で Java 環境からアクセス可能な形式に変換します。片方では Java CORBA エージェントがネイティブの製品 API のラップとなり、もう一方では Java 形式の API をネットワークに提供します。

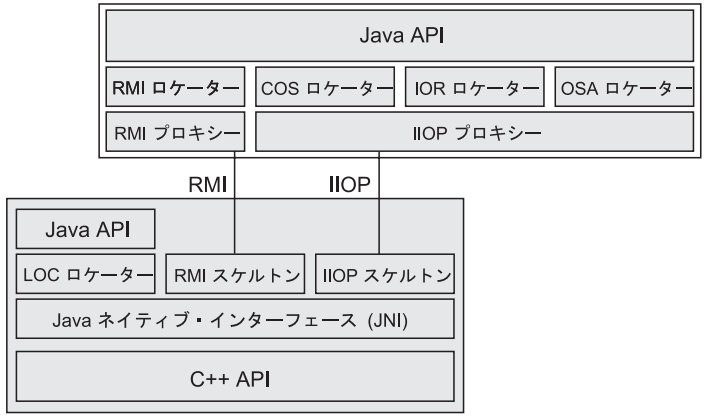
第21章 通信層

Java CORBA エージェントは MQSeries Workflow マシンで実行されるのに対し、Java クライアントはネットワーク上のさまざまな場所で実行されます。MQSeries Workflow は、クライアントが Java CORBA エージェントにアクセスできるよう CORBA、RMI、およびローカル環境をサポートしています。

- CORBA は、分散コンピューティング用のオブジェクト管理グループ (OMG) 標準です。 ORB を使えば、既存のオブジェクトを簡単にネットワーク上で公表することができます。現在 JDK/JRE 1.1.x 用にサポートされている ORB は、 Inprise's VisiBroker for Java Version 3.3 または Version 3.4 です。 JDK/JRE 1.2.x の場合は、 Inprise's VisiBroker for Java Version 3.4 を使用しなければなりません。
- Java RMI (Remote Method Invocation、遠隔メソッド呼び出し) は完全に Java ベースであり、追加のソフトウェアは一切必要ありません。 RMI はほとんどの Java 環境に組み込まれています。

  注: Java RMI エージェントは、プロトタイピング以外には使用しないでください。このエージェントは現在のところ実動用には適していません。
- ローカル・バインディングは、Java CORBA エージェントを組み込む特殊な方法です。このメカニズムは通信層をバイパスし、手続き呼び出しを使用します。 クライアント・アプリケーションがローカル・バインディングを使用する場合、アプリケーションは MQSeries クライアントになることのトレードオフ (取引) を検討する必要があります。その場合、ローカル・バインディングはエージェント側のアプリケーション (サーブレットや Java ベースの非 GUI アクティビティー・インプリメンテーション) に最適なものになります。

第22章 ロケーター方式



クライアントがエージェントを検出するための方法がいくつかあります。 MQSeries Workflow でサポートされている各種の方法を次にリストします。

- OSA 命名: 使用が非常に簡単な VisiBroker 固有の命名機能 (スマート・エージェント)。 1 つの OSAgent をサブネットワーク上で実行するだけで、ネットワーク上のすべてのオブジェクトとそれらの名前の記録を保持します。スマート・エージェントが UDP 経由で情報を同期化する際、必要な情報はクライアント・プログラムが探しているオブジェクトの名前だけです。
- IOR 命名: CORBA アプリケーション用の、ベンダーから独立した命名サービスは、InterOrbReferences 経由で存在しています。特定オブジェクトの文字列化 ID (IOR) は、Web サーバー上のファイルで公表されます。公表された URL を使ってクライアントからこのファイルにアクセスすることにより、オブジェクトへの実際の参照を取得することができます。
- COS 命名: CORBA Naming Service はネイティブの CORBA ディレクトリー・サービスです。 COS を使って、オブジェクトの ID は CORBA システムに公表されます。
- RMI レジストリー: RMI レジストリーには最も多くの Java 開発キットが添付されています。 オブジェクト・インプリメンテーションが登録されている場所で独立型プログラムとして実行するか、またはアプリケーションに組み込むことができます。アプリケーションへの組み込みには、命名機能を提供するのに別個のプログラムが不要になるという利点があります。 RMI レジ

ストリーを使ってオブジェクトを探し出すには、RMI レジストリーを実行しているホストが明らかでなければなりません。

注: Java RMI エージェントは、プロトタイピング以外には使用しないでください。このエージェントは現在のところ実動用には適していません。

- LOC 命名: この方法は、Java API を、同じ物理マシンにある MQSeries Workflow C++ API に接続する場合に使用できます。この方法が効果的なのは、API を提供しているが、固有のクライアントを提供していないプラットフォーム (たとえば、AIX) でクライアントを作成する必要がある場合です。この方法は、追加の通信オーバーヘッドを生じさせずに servlet テクノロジーを使って Web サーバーから MQSeries Workflow API にアクセスするときにも効果的です。ローカル・バインディングは手続き呼び出しを使用するからです。

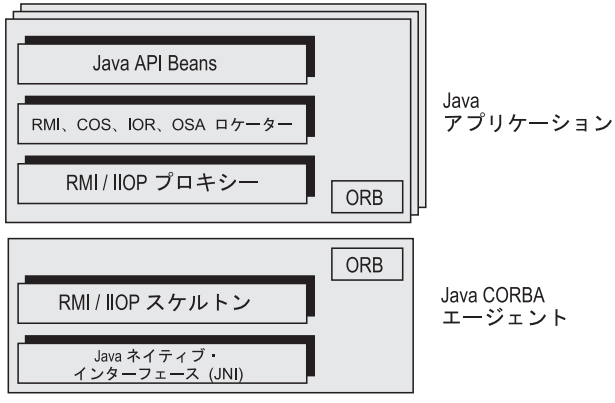
第23章 Java API Beans

Java にはクライアント側の通信層と API Beans 層の両方がインプリメントされています。このため、Java 仮想計算機を提供するマシンであればどのマシンでも、MQSeries Workflow Java API を使って開発されたアプリケーションを実行することができます。

API Beans は、他の MQSeries Workflow API と同等の機能性を提供します。エージェントを導入するため、名前、コンテキスト、およびロケーター・ポリシーを指定しなければなりません。

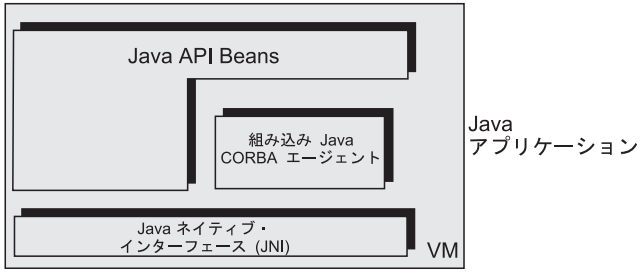
Java API を使ったシナリオを下にいくつか示します。

イントラネットでの Java



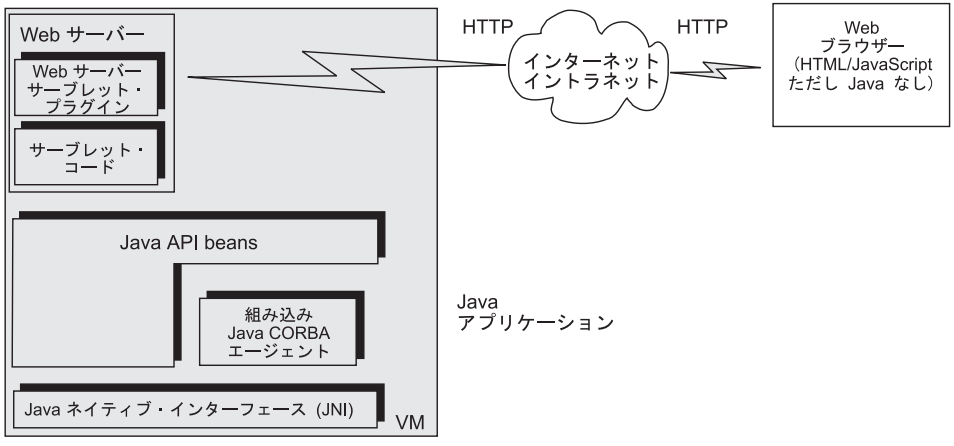
この場合、非 LOC ロケーター・ポリシーを使用し、外部エージェントを指定します。 API Beans、および非 RMI プロトコルでは VisiBroker ORB (COS,IOR,OSA) をインストールしておく必要があります。

プログラミング言語としての Java



この場合、Java CORBA エージェントの LOC_LOCATOR ポリシーを使用します。 Java API Beans をインストールしておく必要があります。

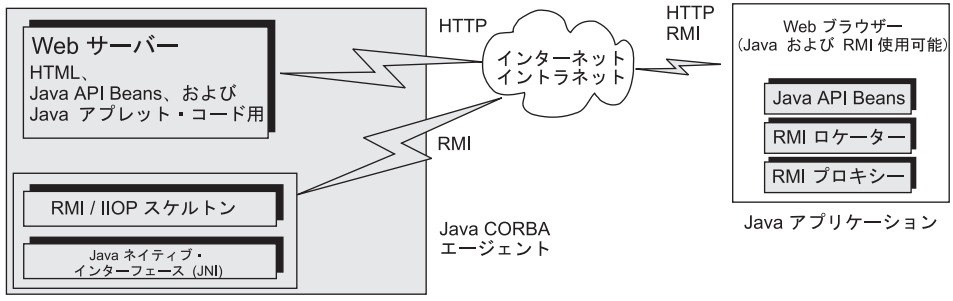
インターネットでの Java (サーブレット)



この場合、LOC_LOCATOR ポリシーを使用する必要があります。 API Beans をインストールしておく必要があります。

注: 要求をディスパッチするアプリケーションは、 MQSeries Workflow Java API が MQSeries Workflow セッションの状態を保守していることを認識していなければなりません。たとえば、Web Server Farm と IBM Websphere e-Network-Dispatcher のように要求をファームの任意の Web Server にディスパッチするディスパッチャー機能とがある場合、ディスパッチャーは Web Server 類縁にアクセスする Web Browser を保守するよう構成しなければなりません。

インターネットでの Java (アプレット RMI)



この場合、RMI_LOCATOR ポリシーを使用し、RMI エージェントを指定する必要があります。 RMI エージェントおよび Java API Beans をインストールしておく必要があります。アプレットはエージェント・オブジェクトのコンテキストで指定する必要があります。

制約事項

クライアント・アプリケーション内で複数のエージェント bean (com.ibm.workflow.api.Agent) をインスタンス化することができますが、特定のロケーター・ポリシーと接点を持つエージェントはすべてのエージェント bean に対して同一でなければなりません。つまり、Agent.setName() に渡される値は、使用するクライアント・アプリケーションによって作成されるすべてのエージェント bean に対して同一でなければなりません。

第24章 Java CORBA エージェントのスレッド化に関する考慮事項

MQSeries Workflow C および C++ プログラミング言語には、関数 / メソッド FmcjGlobalConnect()/FmcjGlobal::Connect() および FmcjGlobalDisconnect()/FmcjGlobal::Disconnect() という、アプリケーションがスレッドを開始または終了するときに呼び出されるメソッドがあります。これらの呼び出しを使用してスレッドをフレーミングすると、 MQSeries Workflow 実行サーバーとの通信が API によって適正に処理されるようになります。ただし、MQSeries Workflow Java API にはこのような呼び出しはありません。

Java CORBA エージェントには MQSeries Workflow Java API を呼び出すスレッドに対する制御権はありません。それは分散環境では、CORBA ORB、RMI “エンジン” または “Webserver の Servlet Engine” などスレッドを処理するスレッド・プールの所有者がいるからです。通常、特定の作業スレッドが終了したときに、通知を受けるメカニズムはありません。これらのどの所有者もクライアントと作業スレッドとの間の類似性を保守してはいないので、 MQSeries Workflow Java API に Connect() および Disconnect() スレッドを提供しても、スレッド内から通信を制御するためには役立ちません。

MQSeries Workflow Java API は MQSeries をメッセージのトランスポート手段として使用しており、結局のところ作業スレッドを制御できないため、MQSeries リスナーがチャネルを使い尽くしてしまう、つまりエラー ″AMQ9513 Maximum number of channels reached (チャネル数が最大に達しました)″ が生じる可能性があります。

デフォルトで設定されている MQSeries Client API (MQIC) の代わりに、MQSeries Queue Manager API (MQI) を使用することでチャネルの不足を回避できます。

MQSeries Workflow インストール時には MQIC API を使用するように設定されますが、それは MQI API を使用するにはキュー・マネージャーがクライアント・マシンにインストールされていなければならないためです。しかし MQI API は（おそらくスレッドが終了してしまったため）接続に一定時間の通信がないと通知を行い、その接続を終結処理します。

ローカル・バインディング (LOC_LOCATOR) を使用する Java CORBA エージェントまたは Java API Beans を MQSeries Workflow サーバーと同じマシン上で実行している場合は、 MQIC API ではなく MQI API を使用するよう変更できます。

- AIX の場合:
  - ソフト・リンク /usr/lib/libfmccfmlc.sl があれば除去します。(たとえば、**rm /usr/lib/libfmccfmlc.sl**)
  - 新しいソフト・リンク **ln -s /usr/lpp/fmc/lib/mqserver/libfmccfmlc.a /usr/lib/libfmccfmlc.a** を作成します
- OS/2 の場合、CONFIG.SYS の LIBPATH 項目を <MQSeries Workflow インストール・ディレクトリー>¥bin¥mqclientから <MQSeries Workflow インストール・ディレクトリー>¥bin¥mqserver に変更します
- SUN Solaris の場合:
  - ソフト・リンク /usr/lib/libfmccfmlc.so があれば除去します。(**rm/usr/lib/libfmccfmlc.so**)
  - 新しいソフト・リンク **ln -s /usr/lpp/fmc/lib/mqserver/libfmccfmlc.so /usr/lib/libfmccfmlc.so** を作成します
- Windows NT または Windows 2000 の場合、システム PATH 設定を <MQSeries Workflowインストール・ディレクトリー>¥bin¥MQClient から <MQSeries Workflowインストール・ディレクトリー>¥bin¥MQServer に変更します。

注: Java CORBA エージェントを再起動して、変更した設定を有効にします。

ローカル・バインディングを使用する Java CORBA エージェントまたは Java API Beans が MQSeries Workflow サーバーと同じマシン上で実行している場合は、特別な MQSeries Workflow セットアップを行ってキュー・マネージャーを構成しなければなりません。詳細については、MQSeries Workflow サービス・チーム (flowmark@de.ibm.com) に ″How to set up an MQ Client Concentrator″ と題する資料を請求してください。

OSA、IOR、および COS ロケーター・ポリシー

OSA、IOR、または COS ロケーター・ポリシーを使用している場合、スレッド・プールのスレッドの数を指定できます。

いずれかの CORBA ロケーター・ポリシー (Visibroker Smart Agent (OSA)、相互協調可能オブジェクト参照 (Interoperable Object Reference、IOR)、または COS 命名 (COS)) を経由して Java CORBA エージェントにアクセスしている

場合、 Visibroker ORB はスレッド・プールをデフォルトのスレッド化ポリシーとして使用します。チャネルが不足することのないように、スレッドの上限数 **OAthreadMax** およびスレッドの下限数 **OAthreadMin** を同じ値に設定し、ワークロードに応じて作業スレッドが動的に作成されたり終了されたりしないようにしなければなりません。これはすべてのオペレーティング・システムについて当てはまります。

このような設定にすると、必要とされるシステム・リソースの量が一定して大きくなることに注意してください。ワークロードを処理するのに最適なスレッドの数を決めることもまた難しいものです。それで、エージェントのあるマシン (前述参照) 上にキュー・マネージャーを置くことをお勧めします。

オープンするスレッドの数を指定するには、特定の構成の Java CORBA エージェント始動スクリプト (バッチ・ファイル) を編集し、以下のステートメントを ″com.ibm.workflow.agent.Main″ を含む行の末尾に追加します。

```
-OAthreadMin xxx -OAthreadMax xxx     xxx はスレッドの数を表します
```

たとえば、Windows NT 上の構成 TTT の場合:

```
jre -classpath .... com.ibm.workflow.agent.Main -yTTT
                を次のように変更します
jre -classpath .... com.ibm.workflow.agent.Main -yTTT -OAthreadMin xxx
                                                      -OAthreadMax xxx
```

Visibroker ORB コマンド行パラメーターの詳細については、 Visibroker for Java に添付されている資料を参照してください。

RMI ロケーター・ポリシー

テスト環境やプロトタイピング環境などで、多数の並行ユーザーを処理することのないアプリケーションには、 RMI ロケーター・ポリシーの使用が適しています。 RMI 仕様では特定のスレッド・ディスパッチ・ポリシーが定義されておらず、各 RMI インプリメンテーションが異なる実行時特性を示すことができるようになっています。

たとえば、標準のインプリメンテーションにはいくつかの欠陥があります。

- クライアント要求を処理するサーバー・スレッド (エージェント・スレッド) の数の構成可能な上限がありません。そのため、サーバー (エージェント) が大量の要求をかかえて、リソースが不足してしまうこともありえます。RMI は、カスタム・ソケットに対して提供しているフックを、カスタム・スレッド化ポリシーのインプリメントには提供していないので、この問題の解決方法を開発することは非常に難しいことです。

- スレッドを再使用することができないため、リソースがむだになります。

Microsoft Internet Explorer バージョン 4 / バージョン 5 および RMI

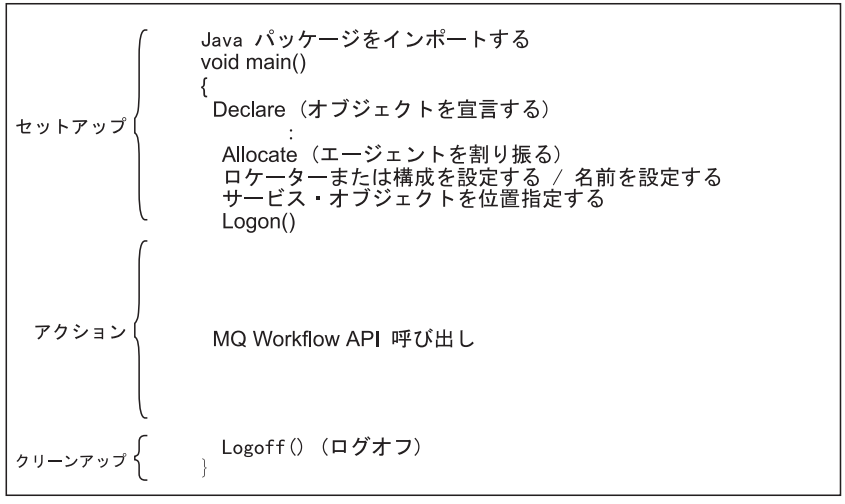
Microsoft は公式には RMI をサポートしていませんが、 rmi.zip という名前のファイルが Microsoft の ftp サイトにあります (ftp://ftp.microsoft.com/developr/MSDN/UnSup-ed/rmi.zip)。 'UnSup-ed' とは「サポートされていない」という意味であることに注意してください。

Windows NT の場合はこのファイルを ¥Winnt¥Java¥Trustlib ディレクトリーに展開し、 Windows 95 または Windows 98 の場合はこのファイルを ¥Windows¥Java¥Trustlib に展開しなければなりません。レジストリーの TrustedClasspath に rmi.zip を次のように追加します。

```
REGEDIT4 [HKEY_LOCAL_MACHINE¥SOFTWARE¥Microsoft¥Java VM]
        "TrustedClasspath"="c:¥winnt¥java¥trustlib¥rmi.zip;..."
```

第25章 MQSeries Workflow クライアント・アプリケーション

多くの場合、MQSeries Workflow Java クライアント・アプリケーションには、以下に示す部分が含まれています (これほど明確に分かれているとは限りません)。



多くの場合、プログラムをセットアップするには、使用するプログラム変数またはオブジェクトを宣言し、アクションに必要な MQSeries Workflow Java API パッケージをインポートします。

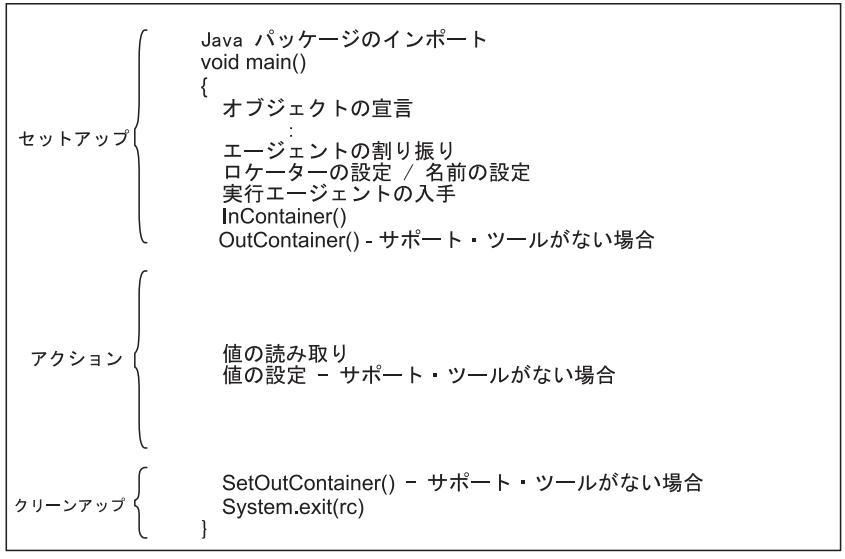
次に、接続先のサーバーを表すサービス・オブジェクトにアクセスする必要があります。そのためには、適切なエージェントによってオブジェクトを検索します。サービス・オブジェクトの割り振りが完了したら、ログオンすることができます。ログオンによって、ログオンするユーザーと、サービス・オブジェクトで表されるサーバーとの間のセッションが確立されます。クライアント / サーバー通信を必要とするそれ以降の呼び出しは、すべてこのセッションを通して実行されます。

正常にログオンしたなら、アクションまたはプログラム実行管理メソッドを発行することによって、自分が許可されている MQSeries Workflow オブジェクトを照会したり管理することができます。

プログラムの終わりではログオフし、サーバーとのセッションをクローズします。

第26章 MQSeries Workflow アクティビティー・インプリメンテーションまたはサポート・ツール

多くの場合、MQSeries Workflow Java アクティビティー・インプリメンテーションまたはサポート・ツールのインプリメンテーションには以下に示す部分が含まれています。



多くの場合、プログラムをセットアップするには、使用するプログラム変数またはオブジェクトを宣言し、アクションに必要な MQSeries Workflow Java API パッケージをインポートします。

ここで、実行エージェント・オブジェクトを探す必要があります。そのためには、適切なエージェントを割り振って要求します。

これでアクティビティー・インプリメンテーションは、このプログラムを開始した MQSeries Workflow プログラム実行エージェントからアクティビティーの入出力コンテナーを取り出すことができます。サポート・ツールが検索できるのは、アクティビティーの入力コンテナーだけです。

コンテナーにアクセスできる場合、プログラミング論理に従って値を読み取ったり設定したりできます。

プログラムの終わりにアクティビティー・インプリメンテーションは、最終出力コンテナーを MQSeries Workflow プログラム実行エージェントに戻します。プログラムの戻り値が、プログラムの総体的な結果をプログラム実行エージェントに知らせます。

出力コンテナーが、アクティビティー・インプリメンテーションの実行を要求した MQSeries Workflow サーバーに送り返されます。戻りコード (_RC) を出口ルーチンまたは変換条件において使用するなら、MQSeries Workflow ナビゲーションに役立ちます。

アクティビティー・インプリメンテーションもまた、クライアント・アプリケーションと同じ動作をして (151ページの『第12章 MQSeries Workflow クライアント・アプリケーション』を参照)、MQSeries Workflow サーバーのサービスを要求することができます。ただし、通常は、それは実行の起動元のサーバーです。次に、Logon() メソッドの代わりに Passthrough() メソッドを使い、作業項目の開始要求からサーバーに認識されているユーザー識別子と権限によって、プログラム実行の原因となったサーバーにログオンします。

注: アクティビティー・インプリメンテーションは現在 LOC_LOCATOR ポリシーだけをサポートしています。

Java でのアクティビティー・インプリメンテーションの例については、

`<MQSeries Workflow インストール・ディレクトリー>¥smp¥java¥actimpl¥ActImpl.java`

を参照してください。

第27章 コンパイル

MQSeries Workflow Java API Beans で使用するために開発されるすべてのプログラムに、MQSeries Workflow 提供のパッケージを組み込む必要があります。これらのファイルは、MQSeries Workflow 開発キットのインストールを選んだ場合にシステムにインストールされます。省略時解釈では、インストール・ディレクトリーの ¥bin¥java3220 サブディレクトリーにインストールされます。

MQSeries Workflow Java API にアクセスするアプリケーションをコンパイルまた実行するために JDK 1.1.x (x は 6 以上) または JDK 1.2.y (y は 2 以上) を使用することができます。コンパイル・ステートメントの例を示します。

```
javac -O <java file>.java
```

-O は、最適化構築を表すオプション・パラメーターです。 CLASSPATH は fmcojapi.jar を参照していなければなりません。

使用するロケーター・ポリシーによっては、Java CORBA Agent へのアクセスを取得することができます。

LOC、RMI

CLASSPATH 環境変数は、fmcojagt.jar を参照していなければなりません。たとえば、

```
CLASSPATH=.;d:¥fmcwinnt¥bin¥java3220¥fmcojagt.jar;
```

CLASSPATH 環境変数は自分で設定しなければならないことに注意してください。最大限の柔軟性を持たせられるように、MQSeries Workflow インストールでは設定されません。 MQSeries Workflow インストールでは Java CORBA エージェントの複数バージョンがサポートされており、適切なファイルが <MQSeries Workflow インストール・ディレクトリー>¥bin¥javaVRMC (VRMC は V がバージョン、R がリリース、M がモディフィケーション、C が CSD レベルを表す) に保管されています。

OSA、IOR

CLASSPATH 環境変数は、fmcojagt.jar および Inprise VisiBroker Java 3.3 または 3.4 を参照していなければなりません。 JDK 1.2.y の場合は、VisiBroker Java 3.4 を使用しなければなりません。たとえば、

```
CLASSPATH=.;d:¥fmcwinnt¥bin¥java3220¥fmcojagt.jar;
           d:¥inprise¥vbroker¥lib¥vbjapp.jar;
           d:¥inprise¥vbroker¥lib¥vbjorb.jar;
           d:¥inprise¥vbroker¥lib¥vbjtools.jar;
```

COS CLASSPATH 環境変数は、fmcojagt.jar および Inprise VisiBroker Java バージョン 3.3 または3.4 を参照していなければなりません。 JDK 1.2.y の場合は、VisiBroker Java バージョン 3.4 を使用しなければなりません。さらに、CORBA ネーミング・サービス vbjcosnm.jar を指定する必要があります。たとえば、

```
CLASSPATH=.;d:¥fmcwinnt¥bin¥java3220¥fmcojagt.jar;
           d:¥inprise¥vbroker¥lib¥vbjapp.jar;
           d:¥inprise¥vbroker¥lib¥vbjorb.jar;
           d:¥inprise¥vbroker¥lib¥vbjtools.jar;
           d:¥inprise¥vbroker¥lib¥vbjcosnm.jar
```

COS ネーミング・サービスが開始していなければなりません。たとえば、

```
java -DJDKrenameBug -DORBservices=CosNaming -DSVCnameroot=
      <Root Name Context> com.visigenic.vbroker.services.CosNaming.ExtFactory
      <Root Name Context>
```

OSA、COS、または IOR ロケーター・ポリシーと JDK 1.2.x をともに使用する

Java 開発キット (JDK)/Java 実行時環境 (JRE) 1.2.x には、標準装備のオブジェクト要求ブローカー (Object Request Broker, ORB) が添付しています。しかし、MQSeries Workflow がサポートしているのは *Inprise Visibroker Java 3.4* のみです。したがって、標準装備の ORB ではなく Visibroker ORB を使用するように JDK/JRE を構成しなければなりません。この点については、MQSeries Workflow Java CORBA エージェントの *MQSeries Workflow インストールの手引き* に詳細に説明されています。

MQSeries Workflow Java API を使用する Java クライアント・アプリケーションもすべて影響を受けます。次のいずれかを用いて、それらのアプリケーションを起動しなければなりません。

```
oldjava -classpath <CLASSPATH> <class>
```

ここで <CLASSPATH> には、JDK/JRE 1.2.x jar ファイルより前に Visibroker Java .jar ファイルを含めなければなりません。 fmcojapi.jar も指定しておく必要があります。

または:

```
java  -Xbootclasspath:<BOOTCLASSPATH> <class>
```

ここで <BOOTCLASSPATH> には、JDK/JRE 1.2.x jar ファイルより前に Visibroker Java .jar ファイルを含めなければなりません。 fmcojapi.jar は通常の CLASSPATH 環境変数に指定する必要があります。

第28章 IBM VisualAge for Java 内から MQSeries Workflow Java API を使用する方法

1. Inprise VisiBroker Java ORB を使用することにしている場合、まずその ORB を作成する必要があります。
   - ワークベンチで、ORB のプロジェクト (たとえば、“VisiBroker ORB”) を追加します。
   - ファイル vbjorb.jar および vbjcosnm.jar を新たに作成したプロジェクトにインポートします。パッケージ ’com.visigenic.vbroker.CORBA’ に関連した問題はすべて無視することができます。
2. MQSeries Workflow JAVA API のプロジェクト (たとえば、“MQWF Java API”) を作成して、ファイル fmcojagt.jar をそのプロジェクトにインポートします。ローカルまたは RMI バインディングを使用しているために先に VisiBroker ORB をインポートしなかった場合、無視できるはずの com.ibm.workflow.corba パッケージでいくつかの問題が起こります。

   MQSeries Workflow Java API を VisualAge for Java にインポートすることは必要ですので注意してください。 “Window” - “Options...” - “Resources” - “Workspace class path:” フィールドに fmcojagt.jar を入力するだけでは不十分です。
3. MQSeries Workflow Java API を使用する実際のプロジェクトを作成します (たとえば、’MyProject’)。通常どおりプログラムを作成します (たとえば、共通 ’main’ メソッドを使用するクラス HelloWorld)。
4. 作成したプログラムを実行する前に、 HelloWorld クラスの ’Properties (プロパティー)’ を選択します。 “Classpath (クラス・パス)” ページを開いて、“Project path (プロジェクト・パス)” を選択します。 “Edit (編集)” を選択して、 “VisiBroker ORB” および “MQWF Java API” パッケージをプロジェクト・クラス・パスに組み込むように選択します。これでアプリケーションを実行する用意ができました。

第29章 トラブルシューティング

大まかな指針: MQSeries Workflow Java API を使用して問題がある場合は、必ず JIT (Just in Time) コンパイラーをオフにしてください。詳細な指針については、JDK/JRE アプリケーション・サーバー資料を調べてください。

第30章 オブジェクト管理

ワークフロー・プロセス・モデル、そのインスタンス、およびその結果の作業項目は、すべて永続的に MQSeries Workflow データベースに保管されるオブジェクトです。つまり、これらはアプリケーション・プログラムとは独立して存在するということです。

永続オブジェクトがアプリケーション・プログラムから照会されると、永続オブジェクトは照会時における永続オブジェクトの状態を持つ*一時オブジェクト*として表されます。複数の照会が発行されると、同一の永続オブジェクト (場合によってはそのオブジェクトのさまざまな状態) を表す複数の一時オブジェクトが存在することになります。

一時オブジェクトの存続期間は、プログラマーが*完全に管理* します。それらがもう必要なくなる時期、つまりオブジェクトが参照されなくなる時期は、プログラマー自身が一番よく知っているからです。しかし、アプリケーション・プログラムが終了すると、一時オブジェクトはもう使えなくなります。

特定の一時オブジェクトは、プログラマーが*明示的に割り振り*ます。そのようなオブジェクトはサポート・オブジェクトであり、永続オブジェクトを反映しません。その例として、Agent や実行サーバーからサービスを要求できるようにするための ExecutionService オブジェクトがあります。

永続オブジェクトを反映する一時オブジェクトは、永続オブジェクトを (たとえば照会によって) 作成または検索するときに*暗黙的に割り振られ*ます。

一時オブジェクトの存続期間は完全にプログラマー自身が管理しますが、その実際の内部オブジェクト構造は、MQSeries Workflow API によってカプセル化されます。

すべてのリソース・メモリーは最終的にアプリケーション・プロセスそのものによって所有されることになるので、そのプロセス内のさまざまなスレッドからすべてのオブジェクトにアクセスすることができます。 MQSeries Workflow でスレッドを使用できないということはありません。これは、再入可能なものとしてコーディングされます。ただし、MQSeries Workflow はスレッドを明示的にサポートしてはいません。さまざまなスレッドから同じ一時オブジェクトにアクセスしたい場合は、そのオブジェクトでのアクセスを同期化する必要があります。オブジェクトはスレッド・セーフでは**ありません**。

Java API Beans 使用時のガーベッジ・コレクション

ガーベッジ・コレクションは、通常、Java プログラマーの介入なしにバックグラウンドで実行されます。このことは、オブジェクトが RMI 伝送プロトコル経由で通信する分散 Java 環境にも当てはまります。しかし、それ以外のプロトコル (CORBA の IIOP など) では、参照されていないオブジェクトをエージェント側で除去するための手続きをとる必要があります。 CORBA を使用する場合、Java 仮想計算機により暗黙に実行されるメモリー管理は、クライアントおよびエージェント上でのオブジェクト除去と同期していません。そのため、参照されていないクライアント・オブジェクトの、エージェント側にあるコピーは、自動的に除去のためにマークされません。オブジェクト要求ブローカー (ORB: Object Request Broker) は、登録済みのオブジェクトに対して参照を保持しているか保持していないかを判別することができません (ORB によっては判別できるものもありますが、そのために所有 CORBA 拡張子を使っています)。接続メソッドを使って ORB により登録された、クライアント・オブジェクトのエージェント側のコピーは、明示的に切断する必要があります。

MQSeries Workflow Java API Beans を使用する場合、CORBA オブジェクト要求ブローカー (ORB) によってデータ転送がなされたときにハウスキーピング (後片付け) を行う、組み込みのガーベッジ・コレクション・メカニズム、つまり MQSeries Workflow Java API Beans 収集機能が用意されています。

MQSeries Workflow Java API Beans エージェントを開始する前に、収集機能を制御するパラメーターのセットを設定しなければなりません。次のような制御パラメーターがあります。

- 収集サイクル時間値 (ミリ秒単位) は、クライアント側収集機能およびサーバー側収集機能の両方に対して有効です。省略時値は 300000 ミリ秒です。
- 収集機能のしきい値は、参照されなくなったオブジェクトの累積数をカウントするときの最大数を指定します。サイクル時間よりしきい値の方が優先されます。省略時値は 1000 です。
- 収集率は、クライアント収集機能とサーバー収集機能のサイクル時間の関係を定義します。この収集率は、クライアント側の収集機能のサイクルを計算するときに、サーバー側の収集機能サイクルの乗数として使用します。省略時値は 90 です。つまり、サーバーの収集機能サイクル時間の 90% を表します。これにより、クライアント側の収集機能アクションが、必ずサーバー側の収集機能アクションに先行するようになります。

これらのパラメーターは、構成時に初期化されます。

第31章 Java API Beans および Java CORBA エージェントへのサービスの適用

LOC ロケーター・ポリシー

ローカル・バインディングを使用するとは、Java CORBA エージェントをアプリケーションのプロセスの中で実行することを意味します。通信インターフェースではなくコール・レベル・インターフェースが使用されます。サービスの更新の場合は、新規の Java CORBA エージェント・コードが次のディレクトリーにインストールされます。

```
<Installation Directory>¥bin¥java322x
```

ここで、x はサービス・パックの番号です。ディレクトリー名 javaVRMS は、プログラム・コードのバージョン、リリース、モディフィケーション、およびサービス・パック・レベルを表します。 CLASSPATH が必ずコードの最新レベルへのパスを使用するようにしてください。

注: サーブレット (たとえば、MQSeries Workflow HTML クライアント) とともに LOC_LOCATOR ポリシーを使用する場合は、必ずサーブレット・エンジンのサーブレット登録を更新することが必要になりえます。

COS、IOR、OSA および RMI ロケーター・ポリシー

Java API Beans および Java CORBA エージェントは RMI および CORBA テクノロジーに基づいています。これらのテクノロジーを使用することにより、分散アプリケーションの開発が可能になります。したがって、サーバー上のコードとクライアント上のコードとが一致することが必要です。これらのコードが一致しないと、クライアントはサーバーのオブジェクトとともに作業することが**できません**。 FmcException が ″Could not locate Agent for Domain ..″ (ドメイン .. のエージェントが見つかりませんでした) というメッセージとともに返されます。

MQSeries Workflow バージョン 3.2.1 以降では、新規コード・バージョンへのマイグレーションが簡単になりました。

Windows NT の場合のシナリオ

使用するマシンに Java CORBA エージェント バージョン 3.2.1 がインストールされている場合、 Java コード (つまり Java CORBA エージェント (fmcojagt.jar)) は <インストール・ディレクトリー>¥bin¥java3210 にインストールされています。 RMI、OSA、IOR、または COS ロケーター・ポリシーのいずれかを活用する多様な構成があるとすれば、エージェント構成それぞれに対してエージェントを開始するバッチ・ファイルが 1 つ存在することになります。

注: エージェントに適用される命名規則があります。つまり、同一のロケーター・ポリシーを活用するエージェントが複数ある場合は、各エージェントが固有の名前を持っていなければなりません。その例外が OSA ロケーター・ポリシーで、この場合は同じ名前のエージェントが複数あっても構いません。こうすることにより、これらのエージェントの間で自動的にワークロードが平衡化されます。 (この機能の詳細については、Java 資料の Inprise Visibroker をお調べください。)

新規バージョン (MQSeries Workflow のサービス・パック (CSD) など) が使用可能になると、新規バージョンの Java コンポーネント・コードは適切なディレクトリー、たとえば <インストール・ディレクトリー>¥bin¥java321x (x はサービス・パックの番号) にインストールされます。ディレクトリー名 javaVRMS は、プログラム・コードのバージョン、リリース、モディフィケーション、およびサービス・パック・レベルを表します。

新規レベルのコードをアクティブにするには、新規の Java CORBA エージェント構成を作成することが必要です。構成中に、使用するコードのレベルを指定しなければなりません。最新のレベルを選択してください。それから、使用するクライアントを新規バージョンにマイグレーションできます。つまり、次々にクライアントに対して新規バージョンを適用することになります。最後のクライアントのマイグレーションが完了したら、古いバージョンの Java CORBA エージェントを停止して、その構成を削除することができます。

クライアント上で Java API Beans にサービスを適用する場合は、CLASSPATH を手動で設定しなければなりません。新規 Java API Beans コードを先にインストールしておいて、その後でも CLASSPATH が使用できるようにするため、意図的に CLASSPATH が自動更新**されない**ようになっています。

注:

1. 新規レベルのコードで Java CORBA エージェントを作成する場合は、**固有の**名前を使用しなければなりません。したがって、新規レベルのコードを使

用するクライアントもその新しい名を使用することが必要です。このことは特に OSA_LOCATOR ポリシーを使用するクライアントに適用されます。それは Visibroker Smart Agent テクノロジーが setName() 要求のあて先となる ORB を選択するからです。レベルの異なるコードで実行される同じ名前の Java CORBA エージェントがある場合、バインドされているエージェントが古いか新しいかに応じて、一方の要求は処理が成功しても、他方の要求が失敗することがあります。

2. サービス・パックは累積情報、つまり最新のサービス・パックにはそれ以前のすべてのサービス・パックの修正が含まれたものです。したがって、すべてのサービス・パックを最初から順番にインストールすることは必要ありません。

第5部 XML メッセージ・インターフェース

以降の章では、MQSeries Workflow の XML メッセージ・ベースのインターフェースについて説明します。 ここでは、メッセージの形式と XML の使用法について説明します。

- MQSeries Workflow への要求の送信

  MQSeries Workflow サーバーでアクションを開始するには、メッセージを MQSeries Workflow XML 入力キューに送ります。

  このようにするなら、MQSeries Workflow XML メッセージ形式をサポートするどんなアプリケーションからでも、MQSeries Workflow にアクションを要求できます。

- アクティビティー・インプリメンテーションの起動

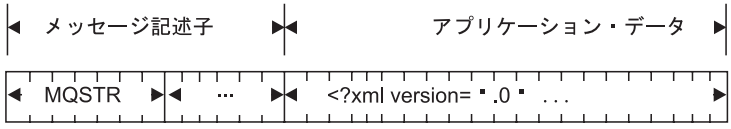
  MQSeries Workflow が適切なメッセージをユーザー定義の MQSeries キューに送信すると、アクティビティー・インプリメンテーションが起動されます。

  こうして、MQSeries キュー上で listen する任意のアプリケーションを開始できます。キューは、XML メッセージを処理できる任意の MQSeries アプリケーションに入力することができます。そのようなアプリケーションとしては、部署内アプリケーションや MQSeries Integrator V2 などの標準プログラムを使用できます。

第32章 MQSeries Workflow XML メッセージ

MQSeries Workflow は MQSeries を使ってメッセージを交換します。
MQSeries メッセージは、次のように 2 つの部分で構成されています。

1. MQSeries メッセージ記述子 (MQMD)。これには、メッセージを説明する構造化データが入れられます。
2. アプリケーション・データ。これには、MQSeries Workflow XML メッセージそのものが入れられます。



MQMD、アプリケーション・データ、および MQSeries ネットワーク内でメッセージを送受信する方法の詳細については、 *IBM MQSeries アプリケーション・プログラミングの手引き* の『MQSeries メッセージ』の章を参照してください。

関係のある MQSeries メッセージ記述子 (MQMD) フィールド

MQSeries Workflow では、MQSeries メッセージ記述子のうち次に示すフィールドが使われます。

- UserIdentifier

  メッセージを送信したユーザー。 MQSeries Workflow に送信された要求メッセージの場合、この情報は、要求の実行の代行の対象となる MQSeries Workflow ユーザーとして使われます。さらに、このユーザー ID を使って許可検査も実行されます。 MQSeries Workflow から送信された起動メッセージの場合、このフィールドには、アクティビティー・インプリメンテーションが始動される対象のユーザーが入れられます。

- Format

  MQSeries Workflow XML メッセージがこのメッセージに入っていることを示す固定文字列。この値は MQSeries 定数 FMTMQ_STRING (MQSTR) によって定義されます。互換性を保つため、フォーマット FMCXML もサポートされています。ただし、使用は勧められていません。

- ReplyToQ/ReplyToQMgr

応答の送信先のキューおよびキュー・マネージャーを指定します。

- Persistence

  メッセージが永続または一時のどちらであるかを指定します。 MQSeries Workflow に送信される要求の場合、MQSeries Workflow 応答の永続性 (persistence) は要求と同じです。 MQSeries Workflow から送信される XML 要求は永続要求であり、呼び出されたアクティビティー・インプリメンテーションから送信される応答も永続でなければなりません。

- Expiry

  一時メッセージの存続時間を 0.1 秒単位で設定することができます。永続メッセージの場合は無期限 (MQEI_UNLIMITED) に設定する必要があります。MQSeries Workflow に送信される要求の場合、MQSeries Workflow 応答の有効期限は、要求中の有効期限値から、実行に要する時間を引いたものに設定します。 MQSeries Workflow から送信される XML 要求には有効期限がありません。

- CorrelID

  応答メッセージを要求メッセージに関連付けるのに使われるデータ。MQSeries Workflow に送信される要求の場合、MQSeries Workflow 応答の相関 ID は要求と同じです。 MQSeries Workflow から送信される XML 要求には相関 ID が含まれており、アプリケーションから送信される応答ではその相関 ID を戻さなければなりません。

- BackoutCount

  トランザクションが失敗してロールバックされたためにメッセージが入力キューに戻された回数。すなわち、このバックアウト・カウントはメッセージの処理が失敗した回数を表します。

詳細については、*IBM MQSeries アプリケーション・プログラミングの手引き* の『MQSeries メッセージ』の章を参照してください。

アプリケーション・データ

MQSeries Workflow では、メッセージ記述に XML 1.0 標準が使われます。XML のリファレンス情報については、http://www.w3.org/TR/REC-xml をご覧ください。

一般に、MQSeries Workflow XML メッセージには次のような情報が入れられます。

- MQSeries Workflow XML メッセージ・ヘッダー。これは、たとえばユーザー・コンテキストなど、すべてのメッセージに共通な情報です。

- メッセージを含む要求または応答 (たとえば、″ProcessTemplateExecute″ 要求) を指定する MQSeries Workflow メッセージ名。
- 要求を実行したり、応答 (たとえば入力コンテナー) を分析したりするのに必要なパラメーター。

MQSeries Workflow XML メッセージの処理時に、 MQSeries Workflow はメッセージの形式が正しいかを検査します。231ページの『第35章 エラー処理』を参照してください。

MQSeries Workflow XML メッセージ・ヘッダー

MQSeries Workflow XML メッセージ・ヘッダーの内容は、下記のような情報です。

- 応答を送信する必要があるかどうか。

  メッセージ・ベースのインターフェースの場合、要求 / 応答のシナリオは同期的に発生する場合と非同期で発生する場合の両方の可能性があります。そういうわけで、個々の要求に対する応答の作成はオプションになっています。しかし、応答は特に必要ない場合でもエラーを報告するための例外応答は必須とすることができます。そのために以下のオプションが用意されています。

  – なし (″No″)
  – エラーのみ (″IfError″)
  – すべて (″Yes″)

  要求された応答は、要求メッセージの MQMD に指定されている応答キューに送られます。 *ResponseRequired* が設定されていないと、要求に対して MQSeries Workflow が想定するデフォルト値は ″Yes″ であり、応答に対するデフォルト値は ″No″ となります。

- ユーザー・コンテキスト。

  このフィールドには、254 バイト以下のコンテキスト・データを指定できます。このデータは、要求と応答を関係付けるのに使うことができます。MQSeries Workflow への要求に指定されたユーザー・コンテキスト・データは、対応する応答に入れて戻されます。

  したがって、メッセージを送信する構成要素内に状態情報を保存しておかなくて済みます。たとえば、MQSeries Integrator V2 などを仲介してメッセージを経路指定する場合は、同じ仲介を通して応答を返送するのが望ましいでしょう。そうすれば、その次にメッセージは元の要求側に送り返されます。

ユーザー・コンテキスト・データには最初の要求者 (または経路全体) を保持するための情報を入れることができ、状態情報を保守する中間処理も不要になります。

コンテナー・データ

コンテナーに関する一般的な説明については、39ページの『第8章 コンテナーの処理』を参照してください。次の例には、基本型の 3 つのコンテナー要素、すなわち基本型の LONG の *Amount*、基本型 STRING の *Currency* および *Risk*、およびタイプ CustomerData のネストされたコンテナー要素 *Customer* を含むタイプ *CreditData* のコンテナーを示します。 *CustomerData* には、基本型 STRING のコンテナー要素 *Name* および基本型 LONG の要素 2 つを持つ配列 *Account* が含まれます。

```
<CreditData>
  <Customer>
    <Name>User1</Name>
    <Account>4711</Account>
    <Account>1100</Account>
  </Customer>
<Amount>100000</Amount>
<Currency>CurrencyX</Currency>
<Risk>high</Risk>
</CreditData>
```

メッセージ・ベース・インターフェースのコンテナーでは、次のような規則が適用されます。

- コンテナー (実際には定義済みデータ・メンバーを含まないコンテナーのユーザー定義部分) は、型によって、つまり関連するデータ構造の名前によって識別されます。すなわち、コンテナーを記述する XML 要素の名前はコンテナーの型を指定するデータ構造の名前であり、上記の例では CreditData です。
- コンテナーの要素はそれぞれの名前で指定します。その型は、XML メッセージの一部ではありません。
- コンテナー要素は基本型にすることもできますし、型に応じて別のデータ構造を示すこともできます。基本コンテナー要素は PCDATA 要素にマップされ、基本型ではないコンテナー要素はその構造にしたがって XML 副要素に分解されます。
- コンテナーまたは基本型ではないコンテナー要素を表す XML 要素の構造は、関連するデータ構造の構造を反映したものです。したがって、データ・メンバーには接頭部が付加されることはなく、名前がドット表記されることはありません。

XML はコンテキスト・フリーを特徴としますが、データ・メンバーとデータ構造を同じ名前にすることはできないことに注意してください。また、同じ名前の 2 つのデータ・メンバーは、それらが異なるデータ構造のものであっても、同じ型でなければなりません。 MQSeries Workflow 定義機能の検査でそのような不具合のいずれかが検出されると、警告が出されます。

- XML タグにはブランクが含まれていてはなりません。 XML タグはデータ構造の後に続き、データ・メンバー名にはブランクが含まれていてはなりません。ブランクを含む名前が XML メッセージ内で参照されている場合、ブランクを除いた名前が使用されます。たとえば、″Default Data Structure″ は <DefaultDataStructure> として参照されます。

  注: ブランクのあるなしでのみ区別されている名前がモデル内にあると、あいまいさが生じてしまいます。データ・モデル作成者はこのようなことが起きないようにしなければなりません。

- データ構造名とデータ・メンバー名には、 '<'、'>'、および '/' などの予約 XML 文字を含めることはできません。これらが含まれた名前が XML メッセージ内で使用されると、生成される XML の形式が正しくないため、構文解析ができません。
- XML メッセージ内でのコンテナー要素 (データ構造メンバー) の指定はオプショナルです。データ・メンバーが XML メッセージ内で指定されていない場合、 MQSeries Workflow はその値を *Null* (設定なし) に設定します。

データ型コード化規則:

- 基本型 STRING、LONG、および FLOAT のコンテナー要素の値は直接コード化できます。
- ブール型パラメーターには、″false″ および ″true″ (大文字小文字を区別しない) を使用します。値 0 と 1 もサポートされています。
- バイナリー・データは、印刷可能バージョンでコード化しなければなりません。 MQSeries Workflow は base64 コード化方式を使用して、XML メッセージのバイナリー・データを表します。このコード化の場合、長さ情報が保たれることに注意してください。 base64 コード化規則については、http://www.cis.ohio-state.edu/htbin/rfc/rfc2045.html を参照してください。
- 先行ブランク、末尾ブランク、改行文字、およびタブ文字は LONG、FLOAT、およびブール型の値からは除去されます。これらの文字は XML メッセージの書式設定をしやすくするために用いることは許可されています。ただし、データ型 STRING または BINARY の値からはそれらが除去されるため、指定された値は変更されずに残ることになります。

配列の表記:

- 配列はその要素を列挙して指定します。
- 配列は複数の要素列挙して指定するものなので、要素を一つしか持たない配列は非配列要素と区別できないためサポートされません。
- <null/> 要素は、配列が*Null* (設定なし) であることを指定するために用いることができます。たとえば、型 *Error* のコンテナーがあるとします。 Error には ID と 3 つの理由コード (理由コードの配列) が含まれています。 2 番目の理由コードが設定されていないとすると、次のように指定できます。

```
<Error>
    <ID>111</ID>
    <ReasonCode>12</ReasonCode>
    <ReasonCode><null/></ReasonCode>
    <ReasonCode>5050</ReasonCode>
</Error>
```

定義済みデータ・メンバー:

- 定義済みデータ・メンバーはユーザー定義データ・メンバーと同じ方法で指定できます。定義済みデータ・メンバーは XML コンテナー要素 (たとえば、<ProcInstInputData>) の直後に定義します。また、それ自体のデータ構造に基づいて分解されなければなりません。たとえば、

```
<_ACTIVITY_INFO>
   <Priority>1</Priority>
</_ACTIVITY_INFO>
```

プロセス・インスタンスの実行例

以下の例は、プロセス・インスタンスの実行を要求する XML メッセージを示しています。

```
<?xml version="1.0" standalone="yes"?>
<WfMessage>
   <WfMessageHeader>
      <ResponseRequired>Yes</ResponseRequired>
      <UserContext>This data is sent back in the response</UserContext>
   </WfMessageHeader>
   <ProcessTemplateExecute>
      <ProcTemplName>OnlineCreditRequest</ProcTemplName>
      <ProcInstName>Credit_Request#658321</ProcInstName>
      <KeepName>true</KeepName>
      <ProcInstInputData>
          <_ACTIVITY_INFO>
            <Priority>1</Priority>
          </_ACTIVITY_INFO>
          <CreditData>
            <Customer>
              <Name>User1</Name>
              <Account>4711</Account>
              <Account>1100</Account>
            </Customer>
            <Amount>100000</Amount>
```

```
            <Currency>CurrencyX</Currency>
            <Risk>high<Risk>
          </CreditData>
    </ProcInstInputData>
  </ProcessTemplateExecute>
</WfMessage
```

コード・ページのサポート

XML では、Unicode のほかに ISO 定義の文字セットでメッセージを指定できます。 MQSeries Workflow に送られる時点で XML メッセージは、必要に応じてコード化キーワードに指定されている形式から、MQSeries Workflow コード・ページに変換されます。 MQSeries Workflow から送信される XML メッセージ (応答およびアクティビティー・インプリメンテーション呼び出し要求) は常に UTF-8 でコード化されます。

MQSeries Workflow XML パーサーによってサポートされるコードページのリストについては、 readme.1st の “XML code page support” の章を参照してください。

第33章 要求を MQSeries Workflow に送信する

MQSeries Workflow メッセージ・ベースのインターフェースは、 MQSeries Workflow からサービスを要求するのに使用できます。これを下の図に示します。

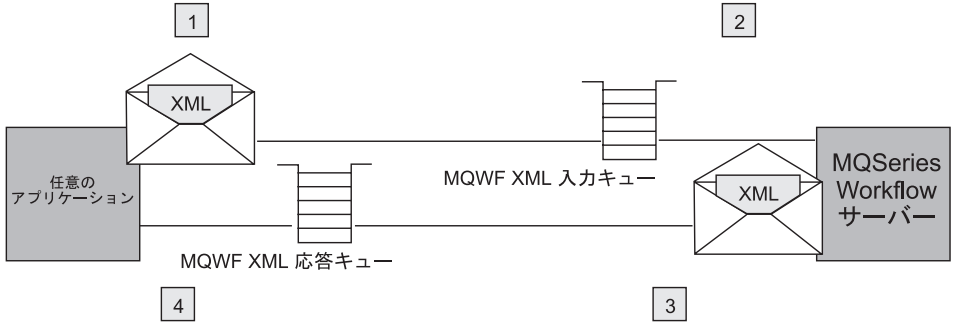


図 3. MQSeries Workflow への要求の送信

1. アプリケーションは、MQSeries Workflow XML メッセージを作成して、それを MQSeries Workflow XML 入力キューに入れます。
2. MQSeries Workflow は、XML 入力キューから XML メッセージを読んで、要求を処理します。
3. MQSeries Workflowは、応答 MQSeries Workflow XML メッセージを作成し、それを応答キューに入れます。応答キュー情報は、着信 XML メッセージの MQSeries Message Descriptor (MQMD) の一部であることに注意してください。

   MQMD、アプリケーション・データ、および MQSeries ネットワーク内でメッセージを送受信する方法の詳細については、*IBM MQSeries アプリケーション・プログラミングの手引き* の『MQSeries メッセージ』の章を参照してください。
4. アプリケーションは、着信メッセージを読んで、応答を処理します。

サポートされる機能

XML メッセージ・インターフェースでは、下記の要求がサポートされます。

- 617ページの『CreateAndStartInstance()』
- 632ページの『ExecuteProcessInstance()』

XML 入力キュー

XML 入力キュー <prefix>.<SystemGroupName>.<SystemName>.EXE.XML と <prefix>.<SystemGroupName>.EXE.XML は、MQSeries Workflow が listen する対象の MQSeries キューです。

このキューへの入力としては、XML メッセージだけが受け入れられます。XML メッセージは MQSeries Workflow XML メッセージ形式に従っていなければなりません。従っていないなら、GeneralError XML メッセージが応答キューに入れられます。

詳細については、213ページの『関係のある MQSeries メッセージ記述子 (MQMD) フィールド』および 214ページの『アプリケーション・データ』の章を参照してください。エラー処理の詳細については、 231ページの『第35章 エラー処理』を参照してください。

認証と許可

MQSeries Workflow のメッセージ・ベース・インターフェースは、認証を MQSeries に依存しています。 MQSeries Workflow では、特に追加の認証は実行しません。 MQSeries のセキュリティーのセットアップについては、*IBM MQSeries システム管理の手引き* の『MQSeries オブジェクトの保護』の章を参照してください。

着信メッセージの MQMD の UserIdentifier フィールドの値は、 要求が実行される対象の MQSeries Workflow ユーザーとして使用されます。そのようなユーザーの許可検査も通常どおりに実行されます。

注: MQSeries UserIdentifier 制約は MQSeries Workflow システム用に定義されたものとは異なります。許可は MQSeries Workflow で検査されるので、XML メッセージの MQMD 内の UserIdentifier は、有効な MQSeries Workflow ユーザーでなければなりません。これは、アプリケーション・プログラマーと MQSeries Workflow 管理者が確認しなければなりません。

プロセス・インスタンスの作成および開始の例

以下の例は、プロセス・インスタンスの作成および開始を要求する XML メッセージを示しています。処理入力データはデータ構造 *Application* によって記述されていることを前提としています。

```
STRUCTURE 'Application'
  'InsuredID'      : LONG;
  'Type'           : LONG;
  'SpecialRisk'    : Long;
  'Accident'       : Long;
  'Ammount'        : FLOAT;
  'StartDate'      : STRING;
  'Term'           : Long;
  'Payment'        : Long;
  'Doctor'         : STRING;
  'Weight'         : Long;
  'Height'         : STRING;
  'Smoker'         : Long;
  'Illness'        : STRING;
  'Hospitalization': STRING;
  'Risks'          : STRING;
END 'Application'
```

XML メッセージは次のようになります。

```
<?xml version="1.0" standalone="yes"?>
<WfMessage>
   <WfMessageHeader>
      <ResponseRequired>Yes</ResponseRequired>
      <UserContext>This data is sent back in the response</UserContext>
   </WfMessageHeader>
   <ProcessTemplateCreateAndStartInstance>
      <ProcTemplName>Medical_Opinion</ProcTemplName>
      <ProcInstName>Medical_Opinion#448</ProcInstName>
      <KeepName>false</KeepName>
      <ProcInstInputData>
          <Application>
            <InsuredID>A</InsuredID>
            <Type>4711</Type>
            <StartDate>12.01.2000</StartDate>
            <Doctor>DoctorX</Doctor>
            <Weight>200</Weight>
            <Smoker>1<Smoker>
          </Application>
    </ProcInstInputData>
  </ProcessTemplateCreateAndStartInstance>
</WfMessage>
```

第34章 アクティビティー・インプリメンテーションの起動

多くの場合、アクティビティー・インプリメンテーションは、MQSeries Workflow がプログラム実行エージェントまたはプログラム実行サーバーへ内部呼び出し要求メッセージを送信することによって開始されます。次に、アクティビティーをインプリメントするようにモデル定義されたプログラムが呼び出されます。またメッセージ・ベース・インターフェースを使えば、MQSeries Workflow がユーザー定義の MQSeries キューにその呼び出し要求メッセージを XML 形式で送ることもできます。

MQSeries Workflow の観点からは、そのようなキューを listen する MQSeries アプリケーションは、プログラム実行サーバーと同じように機能しなければなりません。アクティビティー・インプリメンテーションを呼び出すために必要な情報すべてが渡されます。 MQSeries Workflow から要求された場合、MQSeries アプリケーションは適切な応答を戻さなければなりません。

したがって、このようなアプリケーションはユーザー定義のプログラム実行サーバー (UPES) と呼ばれます。ユーザー定義のプログラム実行サーバーとしては、作成する任意のアプリケーションまたは MQSeries Integrator V2 などのプログラムで、MQSeries Workflow XML メッセージ形式を処理できるものが可能です。

UPES、およびこの UPES によって実行されるプログラム活動は、 MQSeries Workflow 定義機能にモデル化されています。このアクティビティーは活動プロパティー・シートを使用してモデル化されています。

アクティビティー・インプリメンテーション用の 2 つの呼び出しモードは、次のようにモデル定義できます。

- 同期呼び出し (通常のケース)。この場合、MQSeries Workflow は、UPES から結果データの入った完了メッセージが戻されるまで待ってから、アクティビティー・インスタンスが完了したとみなします。
- 非同期呼び出し。この場合、完了メッセージは不要であり、呼び出しメッセージが送られた直後にアクティビティー・インスタンスは完了したとみなされます。 MQSeries Workflow からは結果データは返されずに、プロセス・ナビゲーションが継続します。

下の図は、アクティビティー・インプリメンテーションの同期呼び出しを示しています。

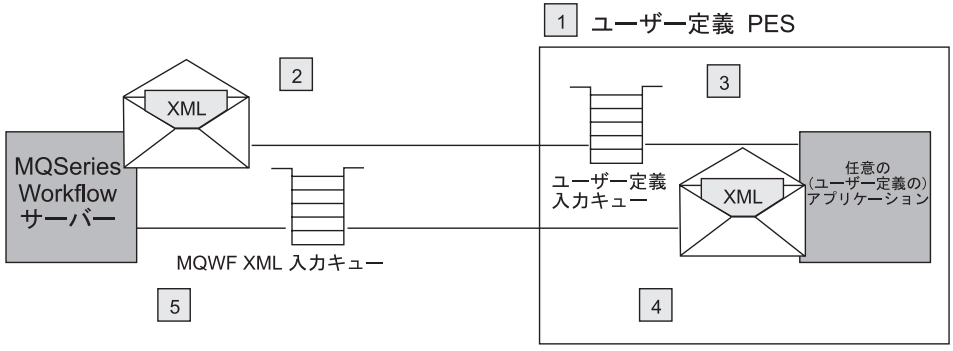


図 4. アクティビティー・インプリメンテーションの始動

1. MQSeries Workflow 定義機能を使って UPES を定義しておかなければなりません。
2. アクティビティー・インプリメンテーションを開始すると、MQSeries Workflow はプログラム呼び出しメッセージを UPES に送ります。
3. UPES キューを listen しているアプリケーションは、XML メッセージを読み取ってから、該当するアクションを実行します。実行される可能性のあるアクションは次のとおりです。
   - たとえば、別の会社への EDI メッセージの送信のように、メッセージを別の形式に変換してから別の受信者に経路指定する。
   - キューから要求を*取り出し (get)*、1 つ以上の DBMS またはその他のリソース・マネージャーを更新し、そして応答をキューに*入れる (put)* というトランザクションを 1 作業単位の中で実行する。
   - たとえば、まだ MQSeries Workflow でサポートされていないプラットフォーム上のプログラムの呼び出しなど、指定されたアクティビティー・インプリメンテーションを呼び出す。
4. アクティビティー・インプリメンテーションが完了するとアプリケーションは、必要なら MQSeries Workflow の XML 応答メッセージを作成して応答キューに入れます。応答キュー情報は、着信 XML 呼び出しメッセージの MQMD の一部であることに注意してください。
5. MQSeries Workflow は、応答メッセージを読んで処理し、適宜そのアクティビティー状態を変更します。

ユーザー定義のプログラム実行サーバー (UPES)

UPES は、MQSeries Workflow 定義機能においてモデル定義されることによって、MQSeries Workflow システム用に定義および構成されます。基本的な属性は、名前とそれが表すキューです。詳しくは、MQSeries Workflow 定義機能のオンライン・ヘルプを参照してください。

UPES キューを listen するアプリケーションは、MQSeries Workflow によっては管理されません。アプリケーションの管理はシステム管理者が担当します。MQSeries Workflow の観点からすると、呼び出しメッセージが正常に UPES キューに入れられた時点で、アクティビティー・インプリメンテーションの呼び出しは正常に完了します。

MQSeries キューの作成および管理方法に関する詳細については、 *IBM MQSeries システム管理の手引き* を参照してください。

アクティビティー・インスタンスの特性に応じて、アクティビティー・インプリメンテーションを起動して MQSeries Workflow プロセス・インスタンスに対して非同期で実行するだけでよいこともあれば、その完了に合わせてプロセス・インスタンス・ナビゲーションを同期化しなければならないこともあります。後者の場合、実行結果を知らせるために完了メッセージを MQSeries Workflow に送る必要があります。前者の場合、呼び出し要求の送信が正常に終わったら、ただちにそのアクティビティー・インスタンスは完了したとみなされます。インプリメンテーションを同期または非同期のどちらで始動するかの情報は、次のように、定義機能内でモデル定義されます。

- 同期

  アクティビティー・インプリメンテーションが始動され、アクティビティー・インスタンスは*実行中 (Running)* 状態になります。アクティビティー・インプリメンテーションが終了して MQSeries Workflow が完了メッセージを受信すると、アクティビティー・インスタンスは、たとえば*終了済み (Finished)* のような該当する状態に設定されます。

  要求と応答の関連付けは、アクティビティー・インプリメンテーションの相関 ID (XML 要素 *ActImplCorrelID*) を使ってなされます。この ID は MQSeries Workflow からの呼び出し要求において渡されるもので、応答の中で返送されなければなりません。

  MQSeries Workflow ActivityImplInvoke メッセージの *ResponseRequired* 要素は、呼び出しが同期であることおよび MQSeries Workflow が応答を待機することを指定する 'Yes' に設定されます。

- 非同期

アクティビティー・インプリメンテーションが始動され、アクティビティー・インスタンスは、たとえば*終了済み (Finished)* のような該当する状態に設定されます。アクティビティー・インプリメンテーションの完了に関する通知は送られないことになっています。完了メッセージが受信されても、無視されます。

MQSeries Workflow ActivityImplInvoke メッセージの *ResponseRequired* 要素は、呼び出しが非同期であることおよび MQSeries Workflow が応答を待機しないことを指定する 'No' に設定されます。

完了メッセージ

アクティビティー・インプリメンテーションを非同期で実行するよう指定した場合、完了メッセージはないものとみなされます。その場合、アクティビティー・インプリメンテーションの始動の発信メッセージが正常に *put* されると、呼び出しは完了したとみなされます。

アクティビティー・インプリメンテーションを同期的に実行するよう指定した場合、MQSeries Workflow 実行サーバー側で完了メッセージが予期されています。このメッセージは、戻りコードと出力コンテナーを渡してアクティビティー・インプリメンテーションの正常完了を報告するか、またはエラー・コードと理由コードを渡して失敗を報告します。エラー・コードと理由コードは、MQSeries Workflow で認識されているものでなければなりません。有効なコードのリストについては、*fmcmretc.h* を参照してください。

許可

MQSeries Workflow から送信された呼び出しメッセージの場合、MQMD の UserIdentifier フィールドはアクティビティー・インスタンス が開始される対象のユーザーのユーザー ID に設定されます。また、呼び出しメッセージの <Starter> 要素自体もそのユーザー ID に設定されます。 UPES アプリケーションは、この情報を使って独自の許可方式を実現できます。

注: MQMD の CorrelID もそのユーザー ID に設定されます。この情報は特定のユーザーについてのみ XML メッセージを listen するために UPES によって使用されることがあります。

同期呼び出しの例

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This document is generated by a MQSeries Workflow Version 3.2.2 server -->
<WfMessage>
  <WfMessageHeader>
```

```
  <ResponseRequired>Yes</ResponseRequired>
</WfMessageHeader>
<ActivityImplInvoke>
  <ActImplCorrelID>FFABCEDF0123456789FF</ActImplCorrelID>
  <Starter>user1</Starter>
  <ProgramID>
    <ProcTemplID>84848484FEFEFEFE</ProcTemplID>
    <ProgramName>PerformOrder</ProgramName>
  </ProgramID>
  <ImplementationData>
     <ImplementationPlatform>AIX</ImplementationPlatform>
     <ProgramParameters>custNo=1234</ProgramParameters>
     <ExeOptions>
        <PathAndFileName>/usr/local/bin/perforder</PathAndFileName>
        <WorkingDirectoryName>/usr/local/data</WorkingDirectoryName>
        <InheritEnvironment>true</InheritEnvironment>
        <StartInForeGround>true</StartInForeGround>
        <AutomaticClose>true</AutomaticClose>
        <WindowStyleVisible>true</Visible>
        <RunInXTerm>true</RunInXTerm>
     </ExeOptions>
  </ImplementationData>
  <ImplementationData>
      <ImplementationPlatform>OS390</ImplementationPlatform>
      <ExternalOptions>
        <ServiceName>CICS42</ServiceName>
        <ServiceType>CICS</ServiceType>
        <InvocationType>EXCI</InvocationType>
        <ExecutableName>ORDR</ExecutableName>
        <ExecutableType>REG1</ExecutableType>
        <IsLocalUser>true</IsLocalUser>
        <IsSecurityRoutineCall>true</IsSecurityRoutineCall>
        <CodePage>850</CodePage>
        <TimeoutPeriod>TimeInterval</TimeoutPeriod>
        <TimeoutInterval>60</TimeoutInterval>
        <IsMappingRoutineCall>false</IsMappingRoutineCall>
      </ExternalOptions>
  </ImplementationData>
  <ProgramInputData>
      <_ACTIVITY>AssessRisk_SubProcess</_ACTIVITY>
      <_PROCESS>CreditRequest#123</_PROCESS>
      <_PROCESS_MODEL>CreditRequest</_PROCESS_MODEL>
      <CreditData>
        <Customer>
          <Name>User1</Name>
        </Customer>
        <Amount>1000</Amount>
        <Currency>CurrencyX</Currency>
      </CreditData>
  </ProgramInputData>
  <ProgramOutputDataDefaults>
      <_ACTIVITY>AssessRisk_SubProcess</_ACTIVITY>
      <_PROCESS>CreditRequest#123</_PROCESS>
      <_PROCESS_MODEL>CreditRequest</_PROCESS_MODEL>
      <CreditData>
```

```
          <Risk>high</Risk>
        </CreditData>
    </ProgramOutputDataDefaults>
  </ActivityImplInvoke>
<WfMessage>
```

UPES 応答の例

```
<WfMessage>
  <WfMessageHeader>
    <ResponseRequired>No</ResponseRequired>
  </WfMessageHeader>
  <ActivityImplInvokeResponse>
    <ActImplCorrelID>FFABCEDF0123456789FF</ActImplCorrelID>
    <ProgramRC>0</ProgramRC>
    <ProgramOutputData>
       <CreditData>
          <Customer>
            <Name>User1</Name>
          </Customer>
          <Amount>1000</Amount>
          <Currency>CurrencyX</Currency>
          <Risk>low</Risk>
       </CreditData>
    </ProgramOutputData>
  </ActivityImplInvokeResponse>
</WfMessage>
```

第35章 エラー処理

この章では、着信メッセージの処理中に起こりうるエラーを MQSeries Workflow が処理する方法について説明します。

MQSeries Workflow XML メッセージのライフ・サイクル

MQSeries Workflow が受け取る各メッセージは、一連の決まった処理手順を経ます。これらの処理手順のいずれかで、エラーが生じることもありえます。

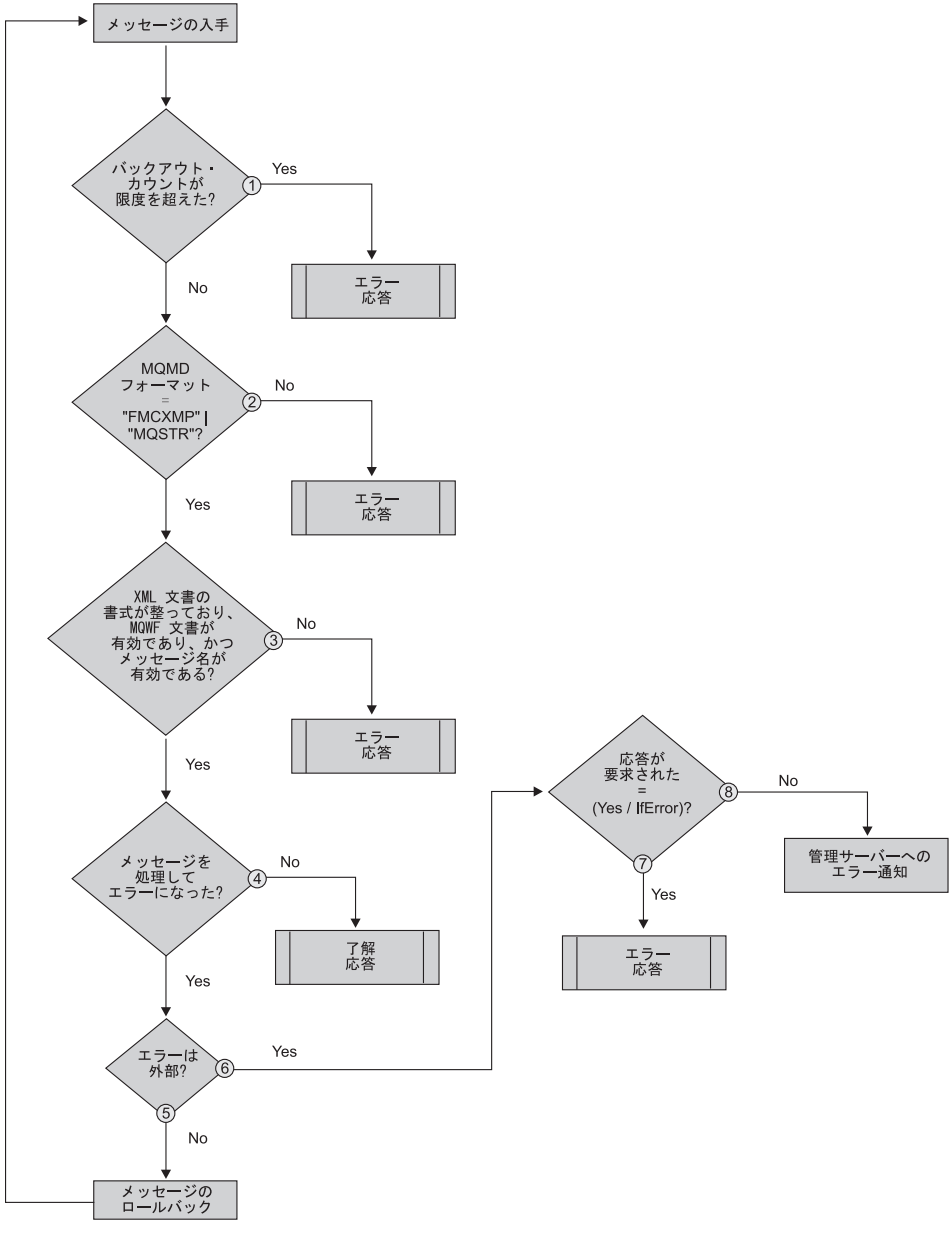
一般に、MQSeries Workflow 内での XML メッセージのライフ・サイクルは次のように記述できます。

1. MQSeries Workflow XML 入力キューからの XML メッセージの取得
2. メッセージの構文解析
3. メッセージ名の判別、たとえば ProcessTemplateExecute
4. アプリケーション・データ・パラメーターの妥当性検査
   - 必須パラメーターが提供されているか
   - パラメーター値が構文的に正しいか
   - 意味的従属性が満たされているか
   - コンテナー値がコンテナー・スキーマに適合しているか
   - コンテナー値が構文的に正しいか
5. メッセージの処理
6. XML 応答メッセージの応答キューへの書き込み
7. トランザクションの確定

エラーが起きると、エラーを記述する応答が作成され、入力メッセージの MQMD フィールドの Reply2q および ReplyToQMgr フィールドで表される応答キューに書き込まれます。応答キュー・アドレスは、この章の残りの部分では REPLY2Q と省略して表記します。

一般的なエラー処理

次のフローチャートには、MQSeries Workflow XML メッセージのライフ・サイクルおよび起こりうるエラーとそれに対する処理方法がより具体的に記述されています。



1. XML 入力キューからメッセージを受け取ると、そのバックアウト・カウントすなわちそれまでの処理試行回数が検査されます。バックアウト・カウントが限度を超えていると、 GeneralError が戻されます。詳しくは、234ページの『応答の送信』を参照してください。

   注: トランザクションがロールバックされると、バックアウト・カウントが増えます。 5 を参照してください。

2. MQMD Format フィールドが正しい値となっているか、すなわちこのフィールドが MQSeries 定数 FMTMQ_STRING (MQSTR) に設定されているかが検査されます。値が正しくないと、GeneralError 応答が戻されます。

   注: 互換性を保つため、フォーマット FMCXML もサポートされています。ただし、使用は勧められていません。

3. XML メッセージの書式が整っているか検査されます。さらに、メッセージ名には XML インターフェースでサポートされている関数を指定しなければなりません。この XML メッセージの書式が整っていないか、またはメッセージ名がサポートされていない場合には、GeneralError 応答が戻されます。

4. XML メッセージが処理されます。処理が正常に行われると、トランザクションは確定し、処理が正常に行われたことを示す応答が戻されます。エラーが起きた場合、処理が継続するかどうかはエラーの種類によります。

5. エラーが内部の理由 (たとえばデータベース・ロック) による場合、このようなエラーは通常一時的に検出されるため、 MQSeries Workflow はメッセージの処理をやり直そうとします。

   処理し直すために、メッセージがロールバックされます。メッセージがバックアウトされて、バックアウト・カウントが増えます。

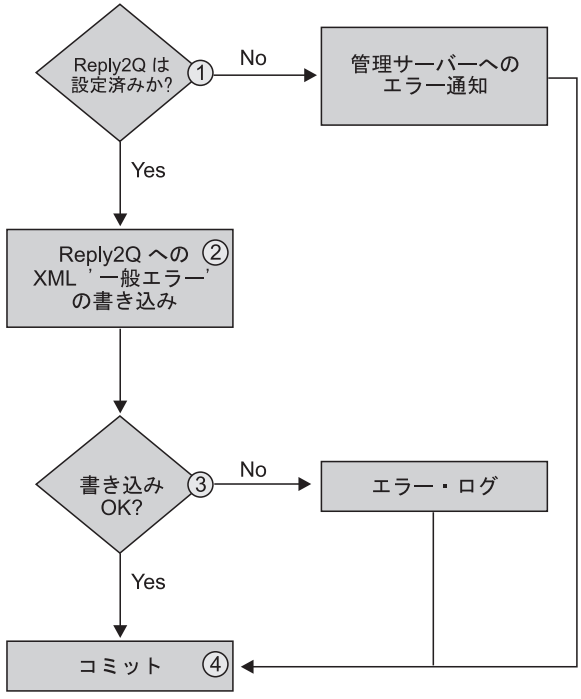
   注: エラーの重大度に応じて、メッセージをロールバックすることで MQSeries Workflow サーバーがシャットダウンすることもあります。

6. エラーの原因が外部の理由 (たとえばパラメーターの構文に誤りがある) である場合、要求された関数への応答があるなら、そこにエラーが記述されます。

7. 着信メッセージの *ResponseRequired* が 'Yes' または 'IfError' に設定されている場合、エラーを記述する応答が戻されます。

8. そうでなければ、エラーが検出されたことを示すエラー通知が MQSeries Workflow 管理サーバーに送信されます。

応答の送信



XML メッセージの送信者に応答が戻された場合:

1. REPLY2Q が設定されているかどうかが検査されます。設定されていないと、エラーが検出されたことを示すエラー通知が MQSeries Workflow 管理サーバーに送信されます。

   管理サーバーはエラーをデータベースに記録します。このエラーは管理クライアントを使用して照会できるようになります。詳細については、*IBM MQSeries Workflow 管理の手引き*『エラー・ログ』の章を参照してください。
2. REPLY2Q が設定されている場合、応答は指定された応答キューに書き込まれます。
3. 'Put (書き込み)' が失敗すると、エラー・ログ項目が書き込まれます。
4. 'Put (書き込み)' が成功するか失敗するかいずれにしても、トランザクションおよびそれに伴うメッセージ読み取りはコミットされ、次のメッセージが処理されます。

エラー処理の詳細

この章では、いくつかのエラーについて詳細に説明します。

MQMD の不適切なメッセージ・フォーマット

まず始めにエラーが起きるとすれば、 XML 着信メッセージが MQMD の無効な Format フィールドを指定している場合です。有効なフォーマットは MQFMT_STRING (MQSTR) です。互換性を保つため、フォーマット FMCXML もサポートされています。ただし、使用は勧められていません。

Format フィールドが間違っていると、 GeneralError XML 応答メッセージが REPLY2Q に送信されます。 GeneralError メッセージの仕様については、 238 ページの『GeneralError メッセージ』を参照してください。

応答キューへの応答の書き込みが失敗すると、エラーが検出されたことを示すエラー通知が MQSeries Workflow 管理サーバーに送信されます。元の XML メッセージがコミットされ、入力キューからは除去されます。

戻されるエラーは次のようなものです。

```
:msgID.       FMC_ERROR_XML_DOCUMENT_FORMAT
:msgNum.      1107
:severity.    Error
:msgText.     "The MQMD format field value '%1$s' of the XML document is incorrect.¥¥n"
$1: The value of the MQMD format field
```

次の XML メッセージが戻されます。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This document is generated by a MQSeries Workflow Version 3.2.2 server -->
<WfMessage>
  <WfMessageHeader>
    <ResponseRequired>No</ResponseRequired>
  </WfMessageHeader>
  <GeneralError>
     <Exception>
        <RC>1107</RC>
        <Parameters>
           <Parameter>ABC </Parameter>
        </Parameters>
        <MessageText>
            FMC01107E The MQMD format field value 'ABC ' of the XML document
            is incorrect.¥n
        </MessageText>
        <Origin>p:¥v322¥src¥fmcmmsg.cxx(113)</Origin>
     </Exception>
  </GeneralError>
</WfMessage>
```

間違ったメッセージ名または書式が整っていない XML 文書

次にエラーが検出されるのは XML メッセージの構文解析中です。 XML メッセージは書式が整っているか[6]を検査し、メッセージ名は調査されます。

この XML メッセージの書式が整っていないか、またはメッセージ名が不明である場合には、GeneralError 応答が REPLY2Q 送信されます。応答キューへの応答の書き込みが失敗すると、エラー通知が MQSeries Workflow 管理サーバーに送信されます、すなわち ADMINPUTQ に書き込まれます。元の XML メッセージがコミットされ、入力キューからは除去されます。

この段階では、*ResponseRequired* タグの判別が常に可能というわけではないので、このタグの設定はどれも無視されます。

戻される可能性のあるエラーは次のとおりです。

```
:msgID.       FMC_ERROR_XML_DOCUMENT_INVALID
:msgNum.      1100
:severity.    Error
"Incorrect XML document. The message that is returned by the XML parser is %1$s¥¥n"
$1: The error message that occurred during parsing of the XML message
:msgID.       FMC_ERROR_NO_MQSWF_DOCUMENT
:msgNum.      1101
:severity.    Error
"The XML document is not a valid MQSeries Workflow XML document.¥¥n"
:msgID.       FMC_ERROR_XML_MESSAGE_NOT_SUPPORTED
:msgNum.      1102
:severity.    Error
"MQSeries Workflow message '%1$s' is not XML enabled.¥¥n"
$1: The XML MQSeries Workflow message name, for example, ProcessTemplateDelete
```

メッセージ処理エラー

着信 XML メッセージを処理している間、アプリケーション・データの妥当性が検査されます。検査されるのは以下の点です。

- XML メッセージ・ヘッダーの *ResponseRequired* タグが正しく設定されているか
- 'UserContext' が長さの制約規則 (<= 254 バイト) に従っているか
- すべてのパラメーターが正しい:
  - 必須パラメーターが提供されているか
  - パラメーター値が構文的に正しいか
  - 意味的従属性が満たされているか

---

6. 書式の整った XML 文書は XML 標準によって定義され、一定レベルの構文上の正確さを保証します。その他に有効な MQSeries Workflow 文書では、ルート要素名が 'WfMessage' であり、オプションのネスト要素 'WfMessageHeader' を伴い、その後にメッセージ名が続くという形を取る必要があります。

– コンテナー値がコンテナー・スキーマに適合しているか
– コンテナー値が構文的に正しいか

エラーが生じた場合、要求された関数の応答メッセージがエラー (たとえば ProcessTemplateCreateAndStartInstanceResponse) を記述するのに使用されます。GeneralError 応答は送信されません。

エラーが起きたときに、*ResponseRequired* の設定によって応答の処理が異なります。

Yes 必ず応答が送信されます。

IfError エラー応答のみ送信されます。

No 応答は送信されません。

注: 着信 XML 応答のエラー (たとえば ActivityImplInvokeResponse) は着信要求と同様に扱われます。唯一の違いは、エラーのときに特別のエラー応答の変わりに GeneralError が送信されます。

XML 処理エラー応答の例:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This document is generated by a MQSeries Workflow Version 3.2.2 server -->
<WfMessage>
   <WfMessageHeader>
     <ResponseRequired>No</ResponseRequired>
   </WfMessageheader>
   <ProcessTemplateCreateAndStartInstanceResponse>
    <Exception>
      <RC>1105</RC>
      <Parameters>
         <Parameter>InsuredID</Parameter>
         <Parameter>Application</Parameter>
      </Parameters>
      <MessageText>FMC01105E Data member 'InsuredID' value of data structure
      'Application' has the wrong type.¥n</MessageText>
      <Origin>d:¥v32_67¥src¥fmcmctnm.cxx(340)</Origin>
    </Exception>
   </ProcessTemplateCreateAndStartInstanceResponse>
<WfMessage>
```

応答が戻されるときのエラー

XML 応答メッセージが MQSeries キューに書き込まれるときに、エラーが起きる場合があります。たとえば、指定されたキューが定義されていない場合や指定されたキューがいっぱいになっている場合です。

注: ActivityImplInvoke メッセージの書き込み先の UPES キューが MQSeries Workflow トポロジー・データの一部ではない場合に、同様のエラーが起こりえます。

応答キューへの応答の書き込みが失敗すると、エラーと応答の最初の 500 バイトがエラーログに記録されます。元の XML メッセージがコミットされ、入力キューからは除去されます。書き込み エラーはすべて永続的なもの、つまり後続のすべての書き込みも失敗するものと想定されます。したがって、着信メッセージの処理は再試行されません。

エラー・ログ項目の例:

```
             MQSeries Workflow 3.2 Error Report
Report creation = 09.05.00 17:29:11
Error location  = File=e:¥v322¥src¥fmccdxmm.cxx, Line=565,
Function=FmcXMLMQDevice::Put(FmcDeviceCtxRef&,FmcDevDataRef&,FmcDeviceControler&)
Error data      = FmcMQPUTException, MQ queue manager name=FMCQM, MQ queue name=DDDD,
                  MQ completion code=2, MQ reason code=2085,
                  Application data (frist 500 bytes)=
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This document is generated by a MQSeries Workflow Version 3.2.2 server -->
<WfMessage>
   <WfMessageHeader>
      <ResponseRequired>No</ResponseRequired>
   </WfMessageHeader>
   <UserContext>This data is sent back in response</UserContext>
   <ProcessTemplateCreateAndStartInstanceResponse>
    <Exception>
      <RC>1108</RC>
      <Parameters>
         <Parameter>ResponseRequired</Parameter>
```

バックアウト・カウントが限度を超えた場合

通常着信 XML メッセージはエラー処理が完了した後にコミットされます。

データベース・ロックのような内部エラーの場合では、オープン・トランザクションはロールバックされ、バックアウト・カウントが増えます。バックアウト・カウントが実行時データベースに設定されている限度に達すると、GeneralError 応答メッセージが REPLY2Q に書き込まれ、着信メッセージがコミットされます。

戻されるエラーは次のようになります。

```
:msgID.       FMC_ERROR_XML_BACKOUT_COUNT_EXCEEDED
:msgNum.      1106
:severity.    Error
:msgText.     "The backout count of the XML document is exceeded.
               The XML document cannot be processed.¥¥n"
```

GeneralError メッセージ

GeneralError 応答メッセージは、エラーが起きたことおよび人手による介入が必要かもしれないことを受信者に知らせるために送信されます。応答は必要ありません。

次は GeneralError XML メッセージを記述する DTD の一部抜粋です。

```
<!ELEMENT WfMessage
          WfMessageHeader?,
          GeneralError >
<!ELEMENT WfMessageHeader    (ResponseRequired?, UserContext?)   >
<!ELEMENT UserContext        (#PCDATA)                           >
<!ELEMENT ResponseRequired   (#PCDATA)                           >
          <!-- Expected values: {No,IfError,Yes}               -->
<!ELEMENT GeneralError       (Exception)                         >
<!ELEMENT Exception          (Rc?,Parameters?,MessageText,Origin)>
<!ELEMENT Parameters         (Parameter*)                        >
<!ELEMENT Parameter          (#PCDATA)                           >
<!ELEMENT Origin             (#PCDATA)                           >
```

たとえば、

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- This document is generated by a MQSeries Workflow Version 3.2.2 server -->
<WfMessage>
  <WfMessageHeader>
    <ResponseRequired>No</ResponseRequired>
  </WfMessageHeader>
  <GeneralError>
     <Exception>
        <RC>1107</RC>
        <Parameters>
           <Parameter>ABC </Parameter>
        </Parameters>
        <MessageText>
           FMC01107E The MQMD format field value 'ABC ' of the XML document is incorrect.¥n
        </MessageText>
        <Origin>p:¥v322¥src¥fmcmmsg.cxx(113)</Origin>
     </Exception>
  </GeneralError>
</WfMessage>
```

第36章 MQSeries Workflow XML メッセージの形式

MQSeries Workflow メッセージの形式を記述するのに使用される XML 構文は、下記のとおりです。下記の形式のコンテナーの内容はあくまで 1 つの提案にすぎません。形式は個々のセットアップによって異なるからです。したがって、ご使用のデータ構造に適した指定を追加しない限り、XML メッセージを妥当性検査するためにこの DTD 記述を使うことはできません。

自分のコンテナーを指定する必要はないことに注意してください。しかし、将来他のどんな XML アプリケーションでも使用したり妥当性検査したりできるようにするためには、それを指定しておくのがよいでしょう。

```
<!-- FmcXMLIF.dtd == DTD for MQSeries Workflow messages -->
<!-- Message ======================================== -->
<!ELEMENT WfMessage
     ( WfMessageHeader?,
       ( ProcessTemplateCreateAndStartInstance
       | ProcessTemplateCreateAndStartInstanceResponse
       | ProcessTemplateExecute
       | ProcessTemplateExecuteResponse
       | ActivityImplInvoke
       | ActivityImplInvokeResponse
       | GeneralError ) ) >
<!-- ================================================
       Workflow Message Header
     ================================================ -->
<!ELEMENT WfMessageHeader           (ResponseRequired?,UserContext?)>
<!-- Opaque -->
<!ELEMENT UserContext               (#PCDATA)> <!-- Length<=254 bytes -->
<!-- Enumerated type -->
<!ELEMENT ResponseRequired          (#PCDATA)>
                                  <!-- Expected values: {No,IfError,Yes} -->

<!-- ================================================
       Specific Messages
     ================================================ -->
<!-- ProcessTemplateCreateAndStart ================== -->
<!ELEMENT ProcessTemplateCreateAndStartInstance
      ( ProcTemplName,
        ProcInstName?,
        KeepName?,
        ProcInstInputData? ) >
<!ELEMENT ProcessTemplateCreateAndStartInstanceResponse
      ( ProcessInstance
      | Exception ) >

<!-- ProcessTemplateExecute ========================= -->
<!ELEMENT ProcessTemplateExecute
      ( ProcTemplName,
```

```
            ProcInstName?,
            KeepName?,
            ProcInstInputData? ) >
<!ELEMENT ProcessTemplateExecuteResponse
      ( ( ProcessInstance,
        ProcInstOutputData? )
      | Exception ) >

<!-- ActivityImplInvoke ============================== -->
<!ELEMENT ActivityImplInvoke
      ( ActImplCorrelID,
        Starter,
        ProgramID,
        (ImplementationData)*,
        ProgramInputData,
        ProgramOutputDataDefaults ) >
<!ELEMENT ActivityImplInvokeResponse
      ( ActImplCorrelID,
        (( ProgramRC,
           ProgramOutputData )
      | Exception )) >

<!-- GeneralError ==================================== -->
<!ELEMENT GeneralError  (Exception) >

<!-- ================================================
           Data Structures
     ================================================ -->
<!ENTITY %STRING  "(#PCDATA)">
<!ENTITY %LONG    "(#PCDATA)">
<!ELEMENT null EMPTY>
<!ELEMENT _PROCESS_INFO
          ( Role?, Organization?, ProcessAdministrator?, Duration? )>
<!ELEMENT _ACTIVITY_INFO
          ( PRIORITY?, MembersOfRoles?, CoordinatorOfRole?,
            Organization?, OrganizationType?,
            LowerLevel?, UpperLevel?,
            People?, PersonToNotify?,
            Duration?, Duration2? )>
<!ELEMENT _ACTIVITY            %STRING;>
<!ELEMENT _PROCESS             %STRING;>
<!ELEMENT _PROCESS_MODEL       %STRING;>
<!ELEMENT _RC                  %LONG;>
<!ELEMENT ROLE                 %STRING;>
<!ELEMENT Organization         %STRING;>
<!ELEMENT ProcessAdministrator %STRING;>
<!ELEMENT Priority             %STRING;>
<!ELEMENT MembersOfRoles       %STRING;>
<!ELEMENT CoordinatorOfRole    %STRING;>
<!ELEMENT OrganizationType     %LONG;>
<!ELEMENT LowerLevel           %LONG;>
<!ELEMENT UpperLevel           %LONG;>
<!ELEMENT People               %STRING;>
<!ELEMENT PersonToNotify       %STRING;>
<!ELEMENT Duration             %LONG;>
<!ELEMENT Duration2            %LONG;>
```

```
<!ENTITY %_CONTAINER_INFO
         "_RC?, _PROCESS?, _PROCESS_MODEL?, _ACTIVITY?,
          _PROCESS_INFO?, _ACTIVITY_INFO?" >

<!-- ====================================================
      Sample Data Structure CreditData
===================================================== -->
<!ELEMENT CreditData
          ( Customer, Amount?, Currency?, Risk? ) >
<!ELEMENT Risk     %LONG;>
<!ELEMENT Currency %STRING;>
<!ELEMENT Amount   %LONG;>
<!ELEMENT Customer %STRING;>

<!-- ====================================================
     Sample Entity Container
     any used data structure must be included, for example,
     <!ENTITY %CONTAINER "(%_CONTAINER_INFO;,(CreditData|abcd|smart))">
===================================================== -->
<!ENTITY %CONTAINER "(%_CONTAINER_INFO;,CreditData)">
<!ELEMENT ProcInstInputData          %CONTAINER;>
<!ELEMENT ProcInstOutputData         %CONTAINER;>
<!ELEMENT ProgramInputData           %CONTAINER;>
<!ELEMENT ProgramOutputData          %CONTAINER;>
<!ELEMENT ProgramOutputDataDefaults  %CONTAINER;>

<!-- Process Instance ================================ -->
<!ELEMENT ProcessInstance
      ( ProcInstID,
        ProcInstName,
        ProcInstParentName?,
        ProcInstTopLevelName,
        ProcInstDescription?,
        ProcInstState,
        LastStateChangeTime,
        LastModificationTime,
        ProcTemplID,
        ProcTemplName,
        Icon,
        Category? ) >

<!-- Program ID ====================================== -->
<!ELEMENT ProgramID
      ( ProcTemplID,
        ProgramName ) >

<!-- Implementation Data ============================= -->
<!ELEMENT ImplementationData
      ( ImplementationPlatform ,
        ProgramParameters,
        ( ExeOptions
        | DllOptions
        | ExternalOptions ) ) >

<!ELEMENT ExeOptions
      ( PathAndFileName,
        WorkingDirectoryName?,
        Environment?,
```

```
        InheritEnvironment,
        StartInForeGround?,
        AutomaticClose?,
        WindowStyle?,
        RunInXTerm? ) >

<!ELEMENT DllOptions
      ( PathAndFileName,
        EntryPointName,
        ExecuteFenced?,
        KeepLoaded? )


<!ELEMENT ExternalOptions
      ( ServiceName,
        ServiceType,
        InvocationType,
        ExecutableName,
        ExecutableType,
        IsLocalUser,
        IsSecurityRoutineCall,
        CodePage?,
        TimeoutPeriod,
        TimeoutInterval?,
        IsMappingRoutineCall,
        MappingType?,
        ForwardMappingFormat?,
        ForwardMappingParameters?,
        BackwardMappingFormat?,
        BackwardMappingParameters? ) >

<!-- Exception ======================================== -->
<!ELEMENT Exception
          (Rc?, Parameters?, MessageText, Origin?) >
<!ELEMENT Parameters
          (Parameter*) >

<!-- Data Elements ================================== -->
<!-- Booleans -->
<!ELEMENT AutomaticClose           (#PCDATA) > <!--  Expected values: {true, false} -->
<!ELEMENT DllV2Compatible          (#PCDATA) > <!--  Expected values: {true, false} -->
<!ELEMENT ExecuteFenced            (#PCDATA) > <!--  Expected values: {true, false} -->
<!ELEMENT InheritEnvironment       (#PCDATA) > <!--  Expected values: {true, false} -->
<!ELEMENT IsLocalUser              (#PCDATA) > <!--  Expected values: {true, false} -->
<!ELEMENT IsMappingRoutineCall     (#PCDATA) > <!--  Expected values: {true, false} -->
<!ELEMENT IsSecurityRoutineCall    (#PCDATA) > <!--  Expected values: {true, false} -->
<!ELEMENT KeepLoaded               (#PCDATA) > <!--  Expected values: {true, false} -->
<!ELEMENT KeepName                 (#PCDATA) > <!--  Expected values: {true, false} -->
<!ELEMENT RunInXTerm               (#PCDATA) > <!--  Expected values: {true, false} -->
<!ELEMENT StartInForeGround        (#PCDATA) > <!--  Expected values: {true, false} -->

<!-- Strings -->
<!ELEMENT BackwardMappingFormat     (#PCDATA) >
<!ELEMENT BackwardMappingParameters (#PCDATA) >
<!ELEMENT Category                  (#PCDATA) >
<!ELEMENT EntryPointName            (#PCDATA) >
<!ELEMENT Environment               (#PCDATA) >
<!ELEMENT ExecutableName            (#PCDATA) >
<!ELEMENT ExecutableType            (#PCDATA) >
<!ELEMENT ForwardMappingFormat      (#PCDATA) >
```

```
<!ELEMENT ForwardMappingParameters  (#PCDATA) >
<!ELEMENT Icon                      (#PCDATA) >
<!ELEMENT InvocationType            (#PCDATA) >
<!ELEMENT MappingType               (#PCDATA) >
<!ELEMENT MessageText               (#PCDATA) >
<!ELEMENT Origin                    (#PCDATA) >
<!ELEMENT Parameter                 (#PCDATA) >
<!ELEMENT PathAndFileName           (#PCDATA) >
<!ELEMENT ProcInstDescription       (#PCDATA) >
<!ELEMENT ProcInstName              (#PCDATA) >
<!ELEMENT ProcInstParentName        (#PCDATA) >
<!ELEMENT ProcInstTopLevelName      (#PCDATA) >
<!ELEMENT ProcTemplName             (#PCDATA) >
<!ELEMENT ProgramName               (#PCDATA) >
<!ELEMENT ProgramParameters         (#PCDATA) >
<!ELEMENT ServiceName               (#PCDATA) >
<!ELEMENT ServiceType               (#PCDATA) >
<!ELEMENT Starter                   (#PCDATA) >
<!ELEMENT WorkingDirectoryName      (#PCDATA) >

<!-- Opaque -->
<!ELEMENT ActImplCorrelID           (#PCDATA) > <!-- Length =  80 bytes -->
<!ELEMENT ProcInstID                (#PCDATA) > <!-- Length <= 64 bytes -->
<!ELEMENT ProcTemplID               (#PCDATA) > <!-- Length <= 64 bytes -->

<!-- Numbers -->
<!ELEMENT CodePage                  (#PCDATA) >
<!ELEMENT ProgramRC                 (#PCDATA) >
<!ELEMENT Rc                        (#PCDATA) >
<!ELEMENT TimeoutInterval           (#PCDATA) >

<!-- Timestamps YYYY-MM-DD-hh.mm.ss.000000 (000000 milliseconds) -->
<!ELEMENT LastModificationTime      (#PCDATA) >
<!ELEMENT LastStateChangeTime       (#PCDATA) >

<!-- Enumerated types -->
<!ELEMENT ImplementationPlatform    (#PCDATA) > <!--  Expected values:
                                                  { OS2,        AIX,
                                                    HPUX,      Windows95,
                                                    WindowsNT, OS390,
                                                    Solaris } -->
<!ELEMENT ProcInstState             (#PCDATA) > <!-- Expected values:
                                                  { Ready,      Running,
                                                    Finished,   Terminated,
                                                    Suspended,  Terminating,
                                                    Suspending,Deleted } -->
<!ELEMENT WindowStyle               (#PCDATA) > <!-- Expected values:
                                                  { Visible,    Invisible,
                                                    Minimized,  Maximized } -->
<!ELEMENT TimeoutPeriod             (#PCDATA) > <!-- Expected values:
                                                  { TimeInterval
                                                    Forever     Never } -->
```

第6部 MQSeries Workflow API の使用

第37章 MQSeries Workflow 実行機能 API の使用

実行機能 API の概要

通常、MQSeries Workflow アプリケーション・プログラムを作成する場合には、次のような検討すべきさまざまなことがあります。

- 以下のことのためのクライアント・アプリケーションを作成することができます。
  - プロセス・インスタンスの管理
  - ワークリストまたは作業項目 (あるいはその両方) の処理
  - プロセス・インスタンスまたは作業項目の管理
  - 実行の進行状況のモニター
- ワークフロー・プロセスにアクティビティーまたはサポート・ツールをインプリメントするプログラムを作成できます。

多くの場合、これらのプログラムでは、MQSeries Workflow API のサブセットしか使いません。たとえば、アクティビティー・インプリメンテーションは通常そのコンテナーにしかアクセスしません。つまり、いわゆる “コンテナー API” しか使いません。 MQSeries Workflow API、つまりヘッダー・フィールドとライブラリー構造、ActiveX コントロール、または import パッケージは、この事実を考慮に入れています。

実行機能サービスを要求するためには、クライアント・アプリケーションと MQSeries Workflow 実行サーバーとの間に通信を確立する必要があります。

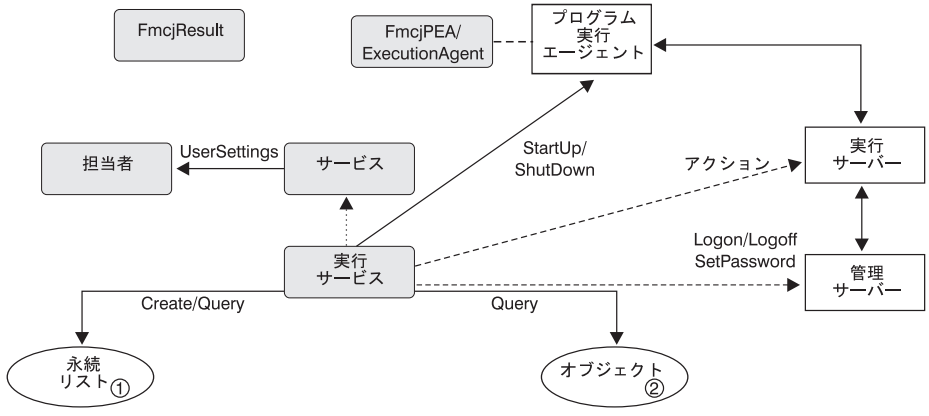


図 5. クライアント / サーバー通信のセットアップ. 凡例: ⟶ 継承 (C++); ⟶ アクセスの提供; -- --⟶ メッセージの送信

最初のステップとして、FmcjExecutionService または ExecutionService オブジェクトを取得する (構成 / 割り振り / 配置) 必要があります。FmcjExecutionService または ExecutionService オブジェクトは、ユーザーと MQSeries Workflow 実行サーバーとの間のセッションを表します。このオブジェクトは基本的に、指定された MQSeries Workflow 実行サーバーへの通信パスをセットアップしたり、ユーザー・セッションを確立 (Logon() と Passthrough()) したり終了 (Logoff()) したりするための基本的な関数 / メソッドを提供します。実行サーバーだけでなく管理サーバーにもログオンするためには、管理サーバーを起動および実行して必要な認証を完了する必要があります。ただし、そのことは明らかにはわかりません。

実行サーバーとのセッションが確立されたなら、次のことが可能になります。

- 許可されているオブジェクト - プロセス・テンプレート、プロセス・インスタンス、アイテム (作業項目、アクティビティー・インスタンス通知、プロセス・インスタンス通知)、またはそれらのオブジェクトを含むリスト - の照会。
- 永続リスト、すなわち MQSeries Workflow データベース内に含まれているオブジェクトについての永続ビューの作成。
- ログオン・ユーザーについての情報の照会、またはそのログオン・ユーザーのパスワードの変更。
- ログオン・ユーザーに関連したプログラム実行エージェントの始動とシャットダウン。この操作が必要になるのは、MQSeries Workflow 独自の手段によって作業項目が実行される場合です。

C および C++ 言語の場合、関数 / メソッドを呼び出すと常に結果オブジェクトが更新されます。間違った要求についての詳細は、ここから取り出せます。詳しくは、10ページの『エラー処理』を参照してください。

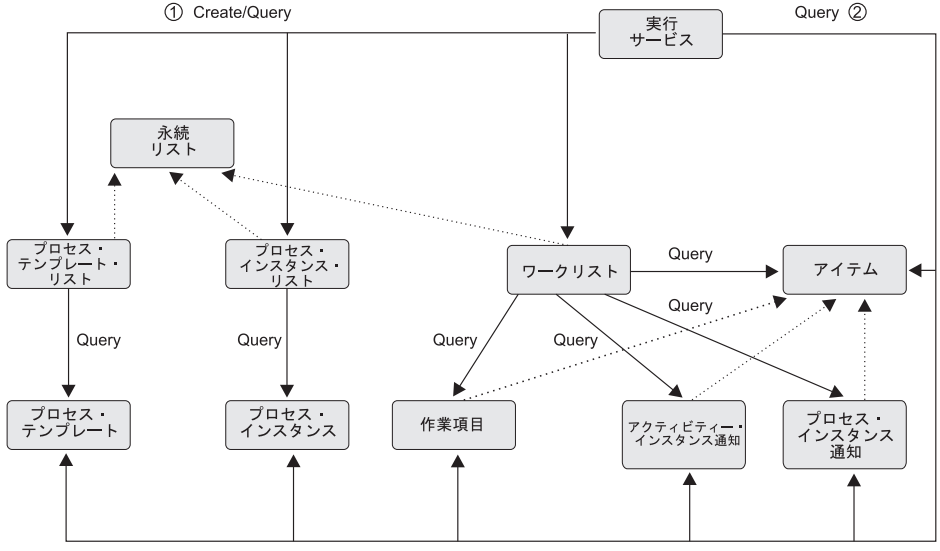


図 6. オブジェクトの照会. 凡例: ⟶ 継承 (C++); ⟶ アクセスの提供

実行サーバーとのセッションが確立されている場合には、永続リスト (プロセス・テンプレート・リスト、プロセス・インスタンス・リスト、ワークリスト) を作成または照会したり、許可されている他のオブジェクトを照会したりすることができます。実行機能では、現在有効なバージョンのプロセス・テンプレートだけを取り出せることに注意してください。将来または過去のバージョンを表示することはできません。

永続リストは、ユーザーに許可されているオブジェクトのセットを表します。永続リストは、それらのオブジェクトについてのビューです。 このリストによってアクセスできるすべてのオブジェクトは特性が同じです。その特性はフィルターによって指定されます。たとえば、指定したフィルターに応じて、ワークリストには作業項目のセットだけが入れられます。このリストからアクティビティー・インスタンス通知やプロセス・インスタンス通知にアクセスすることはできません。ワークリストの内容 (作業項目) は照会することができ、その属性にアクセスすることができます。実行サーバーから作業項目が読み取られるとすぐに、作業項目の開始などの次のアクションを呼び出すことができます。

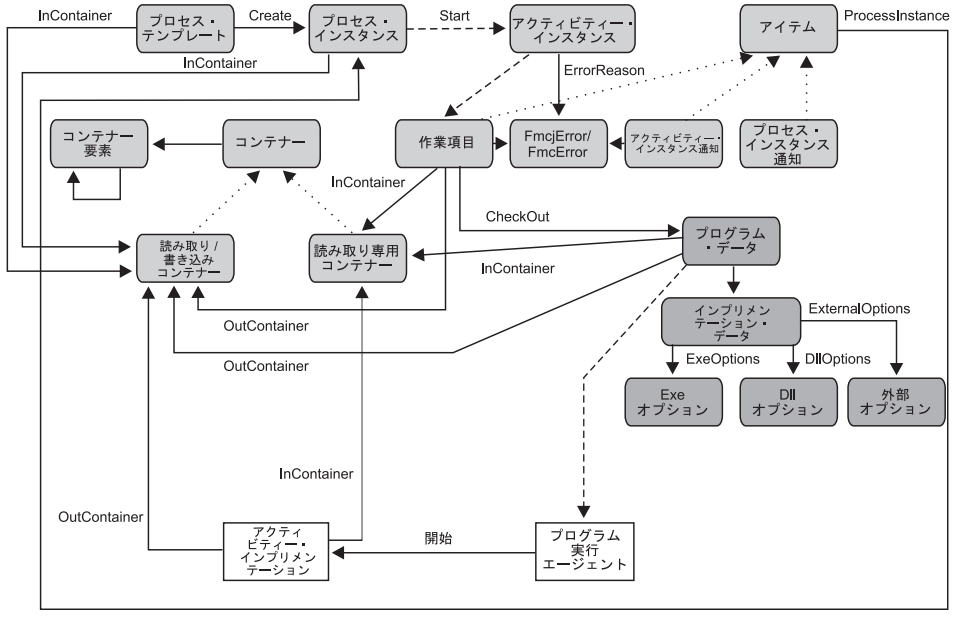


図 7. プロセス・インスタンスと (ワーク) アイテムの処理. 凡例: ⟶ 継承 (C++); ⟶ アクセスの提供; ⟶ データを渡す、または処理結果

有効なバージョンのプロセス・テンプレートを取り出せば、プロセス・インスタンスを作成および開始することができます。プロセス・インスタンスを開始すると、入力データが必要になることがあります。値の読み取りと書き込みには、コンテナー関数 / メソッドを使うことができます。詳しくは、39ページの『第8章 コンテナーの処理』を参照してください。

プロセス・インスタンス・トリガーを開始すると、アクティビティー・インスタンスのスケジューリングがなされます。その結果、作業項目のセットと、場合によってはアクティビティー・インスタンス通知やプロセス・インスタンス通知のセットがもし時間内に処理されていないなら、それらが作成されることになります。その後は、MQSeries Workflow 独自の手段またはユーザー独自の手段を使用することによって、プログラムがインプリメントした作業項目を実行することができます。

ユーザー独自の手段によって実行する場合には、作業項目がチェックアウトされます。チェックアウトすることによって、インプリメント・オプションと入力コンテナー・データの基礎プログラム、プログラム・データ、および記述を実行するのに必要なすべての情報を得ることができます。

MQSeries Workflow 独自の手段によって実行する場合には、そのプログラム・データが自動的にプログラム実行エージェントに送信され、適切なアクティビティー・インプリメンテーションを開始します。次いで、アクティビティー・インプリメンテーションはプログラム実行エージェントに適切な要求を出して、入力コンテナーと出力コンテナーにアクセスします。クライアント・アプリケーション・プログラムまたはアクティビティー・インプリメンテーション・プログラムのどちらから呼び出しても、同じコンテナー・アクセス関数 / メソッドが適用されます。

作業項目とそれに関連したアクティビティー・インスタンスが正常に実行されなかった場合、FmcjError または FmcError オブジェクトから InError 状態の原因を分析するのに役立つ情報が出されます。

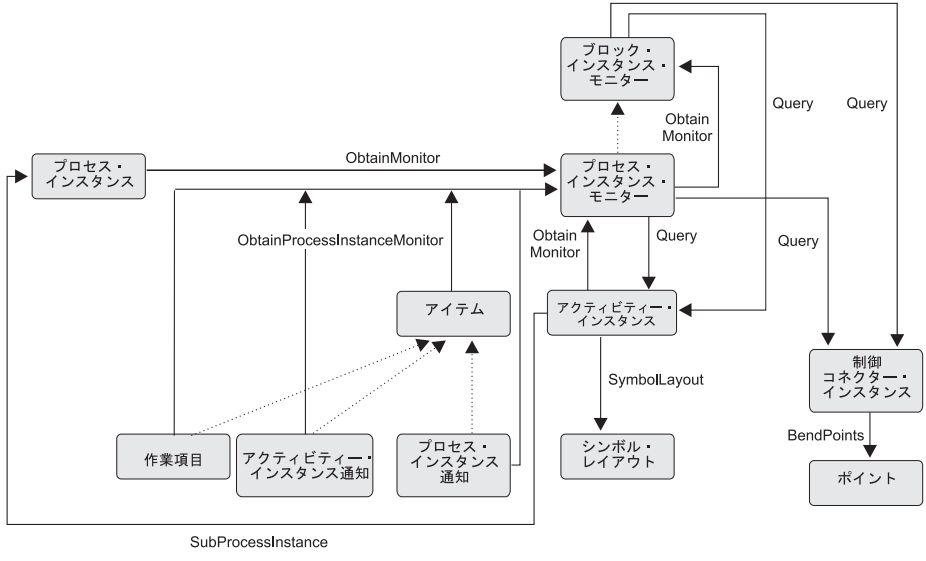


図 8. プロセス・インスタンスのモニター. 凡例: ⟶ 継承 (C++); ⟶ アクセスの提供

プロセス・インスタンスまたはアイテム (作業項目、アクティビティー・インスタンス通知、またはプロセス・インスタンス通知) を取り出せば、それに関連したプロセス・インスタンス・モニターを取得することができます。プロセス・インスタンス・モニターを使用することによって、アクティビティー・インスタンスや制御コネクター・インスタンスの状態を分析することができます。プロセス・インスタンスが経由したパスを調べることもできます。この情報をグラフィカルに表示する場合は、アクティビティー・インスタンス、記号レイアウト、および制御コネクター・インスタンスの位置や屈曲点が役立ちます。

一度プロセス・インスタンス・モニターを取得しておけば、あとはブロック型のアクティビティーのブロック・インスタンス・モニターや、プロセス型のアクティビティーのプロセス・インスタンス・モニター、つまりサブプロセス・インスタンスを取得することによって、プロセス・モデルを繰り返すことができます。詳しくは、75ページの『第9章 プロセス・インスタンスのモニター』を参照してください。

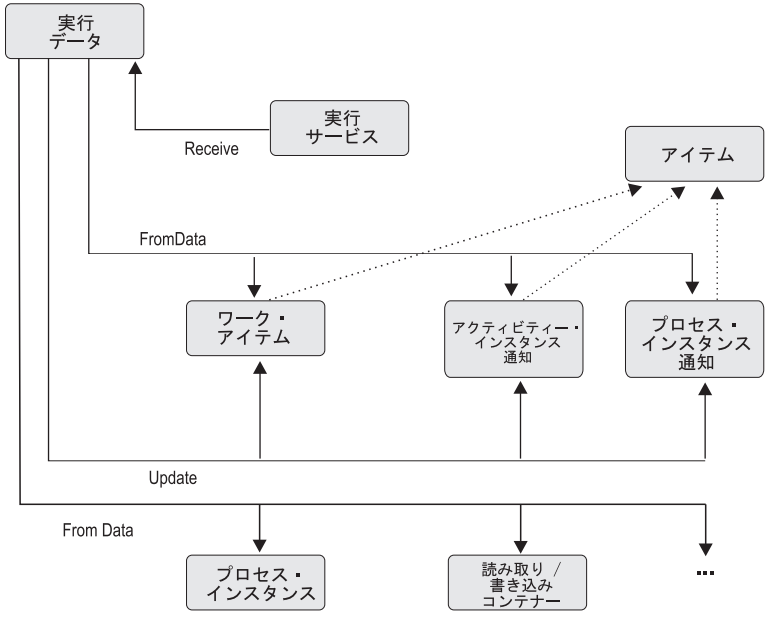


図 9. MQSeries Workflow サーバーが送信するデータの処理. 凡例: ⟶ 継承 (C++); ⟶ アクセスの提供

プロセス設定が *プッシュ・リフレッシュ・ポリシー (push refresh policy)* を指定している場合、 MQSeries Workflow 実行サーバーは変更を作業項目にプッシュするか、通知を現在のクライアントにプッシュします。この場合、またはアプリケーションが非同期要求を出した場合、クライアント・アプリケーションはサーバーから送られたデータや応答を受信するための方法をセットアップする必要があります。受信後は、送信されてきた情報に応じて、該当するオブジェクトを更新、作成、または削除することができます。詳しくは、19ページの『第5章 クライアント / サーバー通信とデータ・アクセス・モデル』を参照してください。

API クラス / オブジェクト

以下に、クラスのアルファベット順リストと、C 言語の関数接頭部のリストを示します。このリストに続く関数 / メソッドはいずれも、それぞれのオブジェ

クトで有効な呼び出しです。ここに示す名前は有効な ActiveX または Java クラスです。それを有効な C++ クラスにするには、接頭部 Fmcj を付加してください。有効な C 言語関数呼び出しにするには、接頭部 Fmcj を付け、実際の関数名を拡張子として追加します。たとえば、ActiveX クラスが Workitem であれば、C++ クラスの名前は FmcjWorkitem になり、C 言語のすべての関数は FmcjWorkitem で始まります。 FmcjWorkitemStart は、有効な C 言語関数の 1 つです。

クラス / オブジェクト	説明
ActivityInstance	ワークフロー・プロセス・テンプレート・アクティビティーのインスタンス。
ActivityInstanceArray	アクティビティー・インスタンスの ActiveX 照会結果。
ActivityInstanceNotifArray	アクティビティー・インスタンス通知の ActiveX 照会結果。
ActivityInstanceNotification	アクティビティー・インスタンスに関連した通知。
ActivityInstanceNotificationVector	アクティビティー・インスタンス通知の C 言語照会結果。
ActivityInstanceVector	アクティビティー・インスタンスの C 言語照会結果。
エージェント	MQSeries Workflow ドメインにアクセスするための Java API 内のエージェント。
BlockInstanceMonitor	ブロック型アクティビティー・インスタンスのモニター。ActiveX の場合は InstanceMonitor を参照。
Container	作業項目またはプロセス・インスタンスのデータ・コンテナー。
ContainerArray	コンテナーを保持するための ActiveX の手段。
ContainerElement	データ・コンテナーの要素。
ContainerElementArray	コンテナー要素の ActiveX 照会結果。
ContainerElementVector	コンテナー要素の C 言語照会結果。
ControlConnectorArray	制御コネクター・インスタンスの ActiveX 照会結果。
ControlConnectorInstance	2 つのアクティビティー・インスタンスの間を接続する制御コネクターのインスタンス。
ControlConnectorInstanceVector	制御コネクター・インスタンスの C 言語照会結果。

クラス / オブジェクト	説明
DateAndTime	日付時刻値の ActiveX 表現。 FmcjCDateTime は、これに相当する C 言語の構造体。 C++ クラスの名前は FmcjDateTime です。 Java は Calendar オブジェクトを使用します。
DllOptions	ダイナミック・リンク・ライブラリーのプログラム・インプリメンテーション定義。
ExecutionData	MQSeries Workflow 実行サーバーにより、または非同期要求への応答により出される情報。
ExecutionAgent	MQSeries Workflow プログラム実行エージェントの Java 表現。 C++ クラスの名前は FmcjPea です。
ExecutionService	サービスを要求するための、ユーザーと MQSeries Workflow 実行サーバーとの間のセッションの表現。
ExecutionServiceArray	実行サービスを保持するための ActiveX の手段。
ExeOptions	実行可能プログラムのためのプログラム・インプリメンテーション定義。
ExternalOptions	外部サービスのためのプログラム・インプリメンテーション定義。
FmcError	Java での InError 状態の原因の説明。 C++ クラスの名前は FmcjError、ActiveX クラスの名前は fmcError です。
FmcException	例外の Java 表現。
Global	C および C++ でグローバル API 関数 / メソッドである関数 / メソッドを分類する手段。
ImplementationData	プログラム・インプリメンテーション定義。
InstanceMonitor	ActiveX におけるプロセス・インスタンスまたはアクティビティー・インスタンスに対するモニター。
Item	ユーザーに関連したアイテム。作業項目または通知。ActiveX では使用不可。
ItemVector	アイテムの C 言語照会結果。
Message	既知のメッセージ ID の NLS 形式化メッセージを要求する手段 (C 言語および C++ のみ)。
PersistentList	永続的に保管されるリスト定義。ActiveX では使用不可。

クラス / オブジェクト	説明
Person	MQSeries Workflow 実行サーバーへログオンしたユーザーについてのユーザー固有の設定値。
Point	制御コネクター・インスタンスの屈曲点の記述。
PointArray	制御コネクター・インスタンス屈曲点の ActiveX 照会結果。
PointVector	屈曲点の C 言語照会結果。
ProcessInstance	ワークフロー・プロセス・テンプレートのインスタンス。
ProcessInstanceList	プロセス・インスタンスを分類するためのリスト。
ProcessInstanceListArray	プロセス・インスタンス・リストの ActiveX 照会結果。
ProcessInstanceListVector	プロセス・インスタンス・リストの C 言語照会結果。
ProcessInstanceMonitor	プロセス・インスタンスのモニター。ActiveX の場合は InstanceMonitor を参照。
ProcessInstanceNotifArray	プロセス・インスタンス通知の ActiveX 照会結果。
ProcessInstanceNotification	プロセス・インスタンスに関連した通知
ProcessInstanceNotificationVector	プロセス・インスタンス通知の C 言語照会結果。
ProcessInstanceVector	プロセス・インスタンスの C 言語照会結果。
ProcessTemplate	アクティビティーとコンテナー、およびその制御フローとデータ・フローで構成されるワークフロー・プロセス・テンプレート。
ProcessTemplateList	プロセス・テンプレートを分類するためのリスト
ProcessTemplateListArray	プロセス・テンプレート・リストの ActiveX 照会結果。
ProcessTemplateListVector	プロセス・テンプレート・リストの C 言語照会結果。
ProcessTemplateVector	プロセス・テンプレートの C 言語照会結果。
ProgramData	アクティビティー・インプリメンテーションのプログラム定義。

クラス / オブジェクト	説明
ProgramTemplate	プロセス・テンプレートに含まれているプログラム定義。
ReadOnlyContainer	読み取り専用のデータ・コンテナー。
ReadWriteContainer	読み取りと書き込みが可能なデータ・コンテナー。
Result	要求の詳細結果。C 言語および C++ 専用。
Service	MQSeries Workflow サービスに共通のいくつかの局面を提供。ActiveX では使用不可。
StringArray	結果として文字列リストが作成される ActiveX 照会結果、または文字列リストを提供するための ActiveX での手段。
StringVector	結果として文字列リストが作成される C 言語照会結果、または文字列リストを提供するための C 言語での手段。
SymbolLayout	アクティビティー・インスタンスのグラフィカル・レイアウトの記述。
Workitem	作業対象のユーザー割り当てアクティビティー・インスタンス。
WorkitemArray	作業項目の ActiveX 照会結果。
WorkitemVector	作業項目の C 言語照会結果。
Worklist	作業項目または通知を分類するためのリスト
WorklistArray	ワークリストの ActiveX 照会結果。
WorklistVector	ワークリストの C 言語照会結果。

オブジェクトごとの関数 / メソッド

アクティビティー・インスタンス

プロセス・テンプレート・アクティビティーのインスタンスのアクティビティー・インスタンス表現。これはプロセス・インスタンスの一部です。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	アクティビティー・インスタンス・オブジェクトを構成します。	86

基本メソッド	説明	ページ
Copy()	コピーによって、アクティビティー・インスタンス・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	アクティビティー・インスタンス・オブジェクトの記憶域を割り振り解除します。	91
destructor()	アクティビティー・インスタンス・オブジェクトを破棄します。	91
Equal()	2 つのアクティビティー・インスタンスを比較します。	89
IsComplete()	完全なアクティビティー・インスタンス情報を入手できるかどうかを示します。	91
IsEmpty()	使用できるアクティビティー・インスタンス情報がないかどうかを示します。	92
Kind()	アクティビティー・インスタンスの種類 (プログラム、プロセス、ブロックなど) を示します。	93, 109
operator=()	アクティビティー・インスタンスをこれに代入します。	88
operator==()	2 つのアクティビティー・インスタンスを比較します。	89

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。

注: 「**設定**」欄の値は、該当属性が 1 次属性 (P) であってアクティビティー・インスタンスが照会されるとただちに設定されるのか、それともこの属性が 2 次属性 (S) であって個々のアクティビティー・インスタンスのリフレッシュの後にしか設定されないのかを示しています。インスタンス・モニター (プロセス型またはブロック型) から戻されるアクティビティー・インスタンスには、1 次値と 2 次値の両方が含まれることに注意してください。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	設定 / 型	説明	ページ
ActivationTime()	P/D	アクティビティー・インスタンスの活動化時刻を戻します。	100

アクセス・メソッド	設定 / 型	説明	ページ
ActivationTimeIsNull()	P/B	活動化時刻がセットされているかどうかを示します。	134
Category()	P/C	アクティビティー・インスタンスのプロセス・カテゴリーを戻します。	128
CategoryIsNull()	P/B	カテゴリーが設定されているかどうかを示します。	134
Description()	P/C	アクティビティー・インスタンスの記述を戻します。	128
DescriptionIsNull()	P/B	記述が設定されているかどうかを示します。	134
Documentation()	S/C	アクティビティー・インスタンスの文書を戻します。	128
DocumentationIsNull()	S/B	文書が設定されているかどうかを示します。	134
EndTime()	S/D	アクティビティー・インスタンスの終了時刻を戻します。	100
EndTimeIsNull()	S/B	終了時刻がセットされているかどうかを示します。	134
ErrorReason()	S/O	アクティビティー・インスタンスが InError 状態である理由を説明したエラー・オブジェクトを戻します。	131
ErrorReasonIsNull()	S/B	エラーの理由が設定されているかどうかを示します。	134
ExitCondition()	S/C	アクティビティー・インスタンスの終了条件を戻します。	128
FirstNotificationTime()	S/D	アクティビティー・インスタンスの最初の通知がなされる予定の時刻、またはなされた時刻を戻します。	100
FirstNotificationTimeIsNull()	S/B	最初の通知時刻が設定されているかどうかを示します。	134
FirstNotifiedPersons()	S/M	アクティビティー・インスタンスの最初の通知を受け取った担当者を戻します。	130

アクセス・メソッド	設定 / 型	説明	ページ
FullName()	P/C	アクティビティー・インスタンスの完全修飾名 (ドット表記) を戻します。	128
Icon()	P/C	アクティビティー・インスタンスに関連付けられているアイコンを戻します。	128
Implementation()	P/C	アクティビティー・インスタンスのインプリメント・プログラムの名前を戻します。	128
ImplementationIsNull()	P/B	インプリメンテーションが設定されているかどうかを示します。 .	134
InContainerName()	S/C	アクティビティー・インスタンスの入力コンテナーの名前を戻します。	128
LastModificationTime()	P/D	アクティビティー・インスタンスの 1 次属性が最後に変更された時刻を戻します。	100
LastStateChangeTime()	P/D	アクティビティー・インスタンスの状態が最後に変更された時刻を戻します。	100
ManualExitMode()	S/B	アクティビティー・インスタンスの終了モードが手動かどうかを示します。	99
ManualStartMode()	S/B	アクティビティー・インスタンスの開始モードが手動かどうかを示します。	99
Name()	P/C	アクティビティー・インスタンスの名前を戻します。	128
OutContainerName()	S/C	アクティビティー・インスタンスの出力コンテナーの名前を戻します。	128
PersistentOid()	P/C	アクティビティー・インスタンスのオブジェクト識別子の表現を戻します。	128
Priority()	P/I	アクティビティー・インスタンスの優先順位を戻します。	127

アクセス・メソッド	設定 / 型	説明	ページ
PriorityIsNull()	P/B	優先順位が設定されているかどうかを示します。	134
ProcessAdmin()	S/C	アクティビティー・インスタンスのプロセス管理者を戻します。	128
ProcessAdminIsNull()	S/B	プロセス管理者が設定されているかどうかを示します。	134
ProcessInstanceName()	P/C	アクティビティー・インスタンスが属するプロセス・インスタンスの名前を戻します。	128
ProcessInstanceState()	P/E	アクティビティー・インスタンスが属するプロセス・インスタンスの状態を戻します。	102, 125
ProcessInstanceSystemGroupName()	S/C	アイテムが属するプロセス・インスタンスのシステム・グループの名前を戻します。	128
ProcessInstanceSystemName()	S/C	アクティビティー・インスタンスが属するプロセス・インスタンスのシステムの名前を戻します。	128
SecondNotificationTime()	S/D	アクティビティー・インスタンスの第 2 の通知がなされる予定の時刻、またはなされた時刻を戻します。	100
SecondNotificationTimeIsNull()	S/B	第 2 の通知時刻が設定されているかどうかを示します。	134
SecondNotifiedPersons()	S/M	アクティビティー・インスタンスの第 2 の通知を受け取った担当者を戻します。	130
Staff()	S/M	アクティビティー・インスタンスの作業項目を割り当てられているすべての担当者を戻します。	130
StartCondition()	S/C	アクティビティー・インスタンスの開始条件を戻します。	128
Starter()	P/C	アクティビティー・インスタンスの開始者を戻します。	128

アクセス・メソッド	設定 / 型	説明	ページ
StarterIsNull()	P/B	開始者が設定されているかどうかを示します。	134
StartTime()	P/D	アクティビティー・インスタンスの開始時刻を戻します。	100
StartTimeIsNull()	P/B	開始時刻が設定されているかどうかを示します。	134
State	P/E	アクティビティー・インスタンスの状態を戻します。	102, 106
StateOfNotification()	S/E	アクティビティー・インスタンスの通知状態を戻します。	102, 105
SupportTools()	P/M	アクティビティー・インスタンスに関連したサポート・ツールを戻します。	130
SymbolLayout()	S/O	アクティビティー・インスタンス記号レイアウトを戻します。	131

アクション関数 / メソッドについては、141ページの『アクション関数 / メソッド』を参照してください。

アクション・メソッド	説明	ページ
InContainer()	アクティビティー・インスタンスの入力コンテナーを取り出します。	355
ObtainInstanceMonitor()	ActiveX 内でアクティビティー・インスタンスが属するプロセス・インスタンスのプロセス・インスタンス・モニターを取り出します。	524
ObtainProcessInstanceMonitor()	アクティビティー・インスタンスが属するプロセス・インスタンスのプロセス・インスタンス・モニターを取り出します。	357
OutContainer()	アクティビティー・インスタンスの出力コンテナーを取り出します。	361
PersistentObject()	指定されたアクティビティー・インスタンスを取り出します。	363

アクション・メソッド	説明	ページ
SubProcessInstance()	プロセス型のアクティビティー・インスタンスをインプリメントするプロセス・インスタンスを取り出します。	366
Terminate()	指定した アクティビティー・インスタンスを終了させます。	369

アクティビティー・インスタンス配列

アクティビティー・インスタンス配列は、ActiveX におけるアクティビティー・インスタンスの照会結果を表します。

配列アクセス・メソッドについては、34ページの『ActiveX の配列』を参照してください。

アクセス・メソッド	説明	ページ
GetAt()	指定された位置の要素を戻します。	36
GetSize()	アクティビティー・インスタンス配列内の要素の数を戻します。	36

アクティビティー・インスタンス通知

アクティビティー・インスタンス通知はアクティビティー・インスタンスの通知を表します。 **FmcjItem のすべての関数 / メソッドは、アクティビティー・インスタンス通知に対しても使用できます。**

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	アクティビティー・インスタンス通知オブジェクトを構成します。	86
Copy()	コピーによって、アクティビティー・インスタンス通知オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	アクティビティー・インスタンス通知オブジェクトの記憶域を割り振り解除します。	91
destructor()	アクティビティー・インスタンス通知オブジェクトを破棄します。	91

基本メソッド	説明	ページ
Kind()	C++ 言語において、オブジェクトがアクティビティー・インスタンス通知であることを示します。	93, 122
operator=()	アクティビティー・インスタンス通知をこれに代入します。	88
operator==()	2 つのアクティビティー・インスタンス通知を比較します。	89

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。

注: 「**設定**」欄の値は、該当属性が 1 次属性 (P) であってアクティビティー・インスタンス通知が照会されるとただちに設定されるのか、それともこの属性が 2 次属性 (S) であって個々のアクティビティー・インスタンス通知のリフレッシュの後にしか設定されないのかを示しています。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	設定 / 型	説明	ページ
ActivityKind()	P/E	関連したアクティビティー・インスタンスの種類 (プログラムまたはプロセスなど) を戻します。	93, 109
ErrorReason()	S/O	関連するアクティビティー・インスタンスが InError 状態である理由を説明したエラー・オブジェクトを戻します。	131
ErrorReasonIsNull()	S/B	エラーの理由が設定されているかどうかを示します。	134
ExitCondition()	S/C	関連するアクティビティー・インスタンスの終了条件を戻します。	128

アクセス・メソッド	設定 / 型	説明	ページ
Expired()	P/B	関連するアクティビティー・インスタンスが開始済みでもう有効期限が切れているかどうかを戻します。	99
FirstNotificationTime()	S/D	アクティビティー・インスタンスの最初の通知時刻 (つまり該当の通知が作成された時刻) を戻します。	100
Implementation()	P/C	関連するアクティビティー・インスタンスのインプリメント・プログラムまたはプロセスの名前を戻します。	128
ImplementationIsNull()	P/B	インプリメンテーションが設定されているかどうかを示します。	134
ManualExitMode()	S/B	関連するアクティビティー・インスタンスの終了モードが手動かどうかを示します。	99
ManualStartMode()	S/B	関連するアクティビティー・インスタンスの開始モードが手動かどうかを示します。	99
PersistentOidOfActivityInstance()	P/C	関連するアクティビティー・インスタンスのオブジェクト ID を戻します。	128
Priority()	P/I	関連するアクティビティー・インスタンスの優先順位を戻します。	127
SecondNotificationTime()	S/D	関連するアクティビティー・インスタンスの 2 番目の通知時刻を戻します。	100
SecondNotificationTimeIsNull()	S/B	第 2 の通知時刻が設定されているかどうかを示します。	134
Staff()	S/M	関連するアクティビティー・インスタンスに対して作業項目を所有しているすべての担当者を戻します。	130

アクセス・メソッド	設定 / 型	説明	ページ
StartCondition()	S/C	関連するアクティビティー・インスタンスの開始条件を戻します。	128
StartOverdue()	P/B	関連するアクティビティー・インスタンスの開始が期限切れかどうかを示します。	99
State	P/E	関連するアクティビティー・インスタンスの状態を戻します。	102, 106
StateOfNotification()	S/E	関連するアクティビティー・インスタンスの通知状態を戻します。	102, 105
SupportTools()	P/M	アクティビティー・インスタンスに関連したサポート・ツールを戻します。	130
SupportToolsIsNull()	P/B	サポート・ツールが設定されているかどうかを示します。	134

アクション関数 / メソッドについては、141ページの『アクション関数 / メソッド』を参照してください。

アクション・メソッド	説明	ページ
ActivityInstance()	指定されたアクティビティー・インスタンスを取り出します。	373
PersistentObject()	指定されたアクティビティー・インスタンス通知を取り出します。	376
StartTool()	指定されたサポート・ツールを開始します。	379
ObtainInstanceMonitor()	ActiveX において関連するプロセス・インスタンスのインスタンス・モニターを戻します。	524

アクティビティー・インスタンス通知配列

アクティビティー・インスタンス通知配列は、ActiveX におけるアクティビティー・インスタンス通知の照会結果を表します。

配列アクセス・メソッドについては、34ページの『ActiveX の配列』を参照してください。

アクセス・メソッド	説明	ページ
GetAt()	指定された位置の要素を戻します。	36
GetSize()	配列内の要素の数を戻します。	36

アクティビティー・インスタンス通知ベクトル

アクティビティー・インスタンス通知ベクトルは、C 言語でのアクティビティー・インスタンス通知の照会結果を表します。

ベクトル関数については、29ページの『C 言語のベクトル』を参照してください。

ベクトル・メソッド	説明
Deallocate()	アクティビティー・インスタンス通知ベクトル・オブジェクトを割り振り解除します。
FirstElement()	アクティビティー・インスタンス通知ベクトルの第 1 要素を戻します。
NextElement()	アクティビティー・インスタンス通知ベクトルの次の要素を戻します。
Size()	アクティビティー・インスタンス通知ベクトル内の要素の数を戻します。

アクティビティー・インスタンス・ベクトル

アクティビティー・インスタンス・ベクトルは、C 言語でのアクティビティー・インスタンスの照会結果を表します。

ベクトル・アクセス関数については、29ページの『C 言語のベクトル』を参照してください。

アクセス・メソッド	説明
Deallocate()	アクティビティー・インスタンス・ベクトル・オブジェクトを割り振り解除します。
FirstElement()	アクティビティー・インスタンス・ベクトルの第 1 要素を戻します。
NextElement()	アクティビティー・インスタンス・ベクトルの次の要素を戻します。
Size()	アクティビティー・インスタンス・ベクトル内の要素の数を戻します。

エージェント

エージェント・オブジェクトは、Java における MQSeries Workflow インスタンスを表します。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	エージェント・オブジェクトを作成します。最初の時点で、エージェントにはコンテキストもロケーター・ポリシーも名前もありません。	86

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
addPropertyChangeListener()	O	プロパティーの変更を通知するリスナー・セットに、特定のリスナーを追加します。	136
addVetoableChangeListener()	O	拒否可能プロパティーの変更を通知するリスナー・セットに、特定のリスナーを追加します。	136
getConfigurationID()	C	プロファイル・アクセスに使用する構成を戻します。	128
getExecutionAgent()	O	LOC_LOCATOR ポリシーが使用されている場合、呼び出し元のアクティビティー・インプリメンテーションにプログラム実行エージェントを戻します。それ以外の場合はヌルを戻します。	131

アクセス・メソッド	タイプ	説明	ページ
getLocator()	I	ロケーター・ポリシーを戻します (COS_LOCATOR、IOR_LOCATOR、LOC_LOCATOR、OSA_LOCATOR、または RMI_LOCATOR のいずれか)。	127
getName()	C	エージェントの名前を戻します。エージェントがバインドされていないなら、空の文字列が戻されます。	128
isBound()	B	エージェント bean が Java CORBA エージェントにバインドされているかどうかを示します。	99
locate()	O	指定されたシステム・グループおよびシステムの中で実行サービスを検索します。	131
removePropertyChangeListener()	O	指定されたリスナーをリスナーのセットから除去します。	136
removeVetoableChangeListener()	O	指定されたリスナーをリスナーのセットから除去します。	136

アクセス・メソッド	タイプ	説明	ページ
setConfigurationID()	C	エージェントがプロファイル・アクセスに使用する構成 ID を設定します。自動的に LOC_LOCATOR のロケーター・ポリシーが設定されます。このようなわけで、ポリシーがすでに setLocator() を使用して設定されている場合、java.beans.PropertyVetoException がスローされます。 1 つのアプリケーション・プロセスには 1 つの構成だけしか使用できません。 LOC_LOCATOR 以外の構成 ID はコマンド行パラメーターとしてエージェント・プログラムに渡されます。	128
setContext()	O	エージェントのコンテキストを設定します。この呼び出しは setName() 呼び出しより先に行われなければなりません。アプレットは、*agent.setContext(this,null);* を発行してコンテキストを設定する必要があります。	131

アクセス・メソッド	タイプ	説明	ページ
setLocator()	I	ロケーター・ポリシーを設定します (COS_LOCATOR、IOR_LOCATOR、LOC_LOCATOR、OSA_LOCATOR、または RMI_LOCATOR のいずれか)。 LOC_LOCATOR が設定されているなら、プロファイル・アクセス用の省略時構成 ID が自動的に使用されます。このようなわけで、ポリシーがすでに setConfigurationID() を使用して設定されている場合、java.beans.PropertyVetoException がスローされます。LOC_LOCATOR 以外の構成 ID はコマンド行パラメーターとしてエージェント・プログラムに渡されます。 この呼び出しは setName() 呼び出しより先に行われなければなりません。 **注:** Java RMI エージェントは、プロトタイピング以外には使用しないでください。このエージェントは現在のところ実動用には適していません。	135
setName()	C	この呼び出しの結果として接触するエージェントの名前を設定します。	128
toString()	C	エージェントの名前を戻します。	128
versionInfo()	C	エージェントのバージョンを戻します。つまり、エージェントがバインドされている場合に、有用な情報だけを戻します。	128

ブロック・インスタンス・モニター

ブロック・インスタンス・モニター・オブジェクトは、ブロック 型のアクティビティー・インスタンスのモニターを表します。**ブロック・インスタンス・モニターのすべての関数 / メソッドは、プロセス・インスタンス・モニターに対しても使用できます。**

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
destructor()	ブロック・インスタンス・モニター・オブジェクト、つまり C++ インターフェースの一時表現を破棄します。ただし、内部ブロック・インスタンス・モニター・オブジェクトについては、プロセス・インスタンス・モニターに属しているために割り振り解除されません。このオブジェクトが割り振り解除されるのは、プロセス・インスタンス・モニターが破棄 / 割り振り解除される時点です。	91

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。ブロック・インスタンス・モニターはプロセス・インスタンス・モニターの一部であるため、すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
ActivityInstances()	M	ブロック・インスタンス・モニターによって表現されていて、ブロック 型のアクティビティー・インスタンスの一部となっているアクティビティー・インスタンスを戻します。アクティビティー・インスタンスには、1 次値と 2 次値の両方が含まれます。	130

アクセス・メソッド	タイプ	説明	ページ
ControlConnectorInstances()	M	ブロック・インスタンス・モニターによって表現されていて、ブロック 型の制御コネクター・インスタンスの一部となっているアクティビティー・インスタンスを戻します。	130

アクション関数 / メソッドについては、141ページの『アクション関数 / メソッド』を参照してください。

アクション・メソッド	説明	ページ
ObtainBlockInstanceMonitor()	ブロック 型のアクティビティー・インスタンスのブロック・インスタンス・モニターを戻します。アクティビティー・インスタンスはブロック・インスタンス・モニターが表現するアクティビティー・インスタンスのセットの一部です。	383
ObtainProcessInstanceMonitor()	プロセス 型のアクティビティー・インスタンスのプロセス・インスタンス・モニターを戻します。アクティビティー・インスタンスはブロック・インスタンス・モニターが表現するアクティビティー・インスタンスのセットの一部です。	386
Refresh()	MQSeries Workflow 実行サーバーからブロック・インスタンス・モニターをリフレッシュします。	389

コンテナー

コンテナーは、プロセス・インスタンスまたは作業項目の入力または出力データ・コンテナーを表します。**コンテナーのすべての関数 / メソッドは、読み取り専用および読み取り / 書き込みコンテナーに使用できます。**

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
IsEmpty()	使用できるコンテナー情報がないかどうかを示します。	92

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
AllLeafCount()	I	MQSeries Workflow の事前定義メンバーを含め、コンテナーのリーフ要素の数を戻します。	49
AllLeaves()	M	MQSeries Workflow の事前定義メンバーを含め、コンテナーのすべてのリーフ要素を戻します。	49
ArrayBinaryLength()	I	指定されたコンテナー・リーフ要素の値の長さを戻します (C 言語)。リーフは配列の一部であり、BINARY 型です。	60
ArrayBinaryValue()	C	指定されたコンテナー・リーフ要素の値を戻します (C 言語)。リーフは配列の一部であり、BINARY 型です。	60
ArrayFloatValue()	F	指定されたコンテナー・リーフ要素の値を戻します (C 言語)。リーフは配列の一部であり、FLOAT 型です。	61
ArrayLongValue()	I	指定されたコンテナー・リーフ要素の値を戻します (C 言語)。リーフは配列の一部であり、LONG 型です。	61
ArrayStringLength()	I	指定されたコンテナー・リーフ要素の値の長さを戻します (C 言語)。リーフは配列の一部であり、STRING 型です。	62
ArrayStringValue()	C	指定されたコンテナー・リーフ要素の値を戻します (C 言語)。リーフは配列の一部であり、STRING 型です。	62
BinaryLength()	I	指定されたコンテナー・リーフ要素の値の長さを戻します。リーフは BINARY 型です。 バイナリーは ActiveX ではサポートされていません。	60,62

アクセス・メソッド	タイプ	説明	ページ
BinaryValue()	C	指定されたコンテナー・リーフ要素の値を戻します (C 言語)。リーフは BINARY 型です。バイナリーは ActiveX でサポートされていません。	60
FloatValue()	F	指定されたコンテナー・リーフ要素の値を戻します (C 言語)。リーフは FLOAT 型です。	61
getBuffer()	C	指定されたコンテナー・リーフ要素の値を戻します (Java)。リーフは BINARY 型です。	63
getBuffer2()	C	指定されたコンテナー・リーフ要素の値を戻します (Java)。リーフは配列の一部であり、BINARY 型です。	63
getDouble()	F	指定されたコンテナー・リーフ要素の値を戻します (Java)。リーフは FLOAT 型です。	63
getDouble2()	F	指定されたコンテナー・リーフ要素の値を戻します (Java)。リーフは配列の一部であり、FLOAT 型です。	63
GetElement()	O	コンテナー要素にアクセスするためのもの。	58
getLong()	I	指定されたコンテナー・リーフ要素の値を戻します (Java)。リーフは LONG 型です。	64
getLong2()	C	指定されたコンテナー・リーフ要素の値を戻します (Java)。リーフは配列の一部であり、LONG 型です。	64
getString()	C	指定されたコンテナー・リーフ要素の値を戻します (Java)。リーフは STRING 型です。	64
getString2()	C	指定されたコンテナー・リーフ要素の値を戻します (Java)。リーフは配列の一部であり、STRING 型です。	64
GetValueDbl()	F	指定されたコンテナー・リーフ要素の値を戻します (ActiveX)。リーフは FLOAT 型です。	60

アクセス・メソッド	タイプ	説明	ページ
GetValueLng()	I	指定されたコンテナー・リーフ要素の値を戻します (ActiveX)。リーフは LONG 型です。	60
GetValueStr()	C	指定されたコンテナー・リーフ要素の値を戻します (ActiveX)。リーフは STRING 型です。	60
LeafCount()	I	コンテナーのユーザー定義リーフ要素の数を戻します。	49
Leaves()	M	コンテナーのすべてのユーザー定義リーフ要素を戻します。	49
LongValue()	I	指定されたコンテナー・リーフ要素の値を戻します (C 言語)。リーフは LONG 型です。	61
MemberCount()	I	コンテナー内の構造メンバーの数を戻します。	50
SetStringCcsid()	C	コンテナーでストリングを読み書きするために使用する CCSID を設定します。	128
StringLength()	I	指定されたコンテナー・リーフ要素の値の長さを戻します (C 言語)。リーフは STRING 型です。	62
StringValue()	C	指定されたコンテナー・リーフ要素の値を戻します (C 言語)。リーフは STRING 型です。	62
StructMembers()	M	コンテナーの構造メンバーを戻します。	50
Type()	C	コンテナーの型、つまりデータ構造体の名前を戻します。	51
Value()	C/I/F/N	指定されたコンテナー・リーフ要素の値を戻します (C++ 言語)。	62

アクティビティー・インプリメンテーション関数 / メソッドについては、141 ページの『アクティビティー・インプリメンテーション関数 / メソッド』を参照してください。

アクティビティー・インプリメンテーション・メソッド	説明	ページ
InContainer()	アクティビティー・インプリメンテーション内から入力コンテナーにアクセスします。 Java の場合は ExecutionAgent を参照。	393
OutContainer()	アクティビティー・インプリメンテーション内から出力コンテナーにアクセスします。 Java の場合は ExecutionAgent を参照。	396
RemoteInContainer()	アクティビティー・インプリメンテーションによって開始されたプログラム内から入力コンテナーにアクセスします。 Java の場合は ExecutionAgent を参照。	398
RemoteOutContainer()	アクティビティー・インプリメンテーションによって開始されたプログラム内から出力コンテナーにアクセスします。 Java の場合は ExecutionAgent を参照。	401
SetOutContainer()	アクティビティー・インプリメンテーション内から出力コンテナーを設定します。 Java の場合は ExecutionAgent を参照。	403
SetRemoteOutContainer()	アクティビティー・インプリメンテーションによって開始されたプログラム内から出力コンテナーを設定します。 Java の場合は ExecutionAgent を参照。	406

コンテナー配列

コンテナー配列は、ActiveX におけるコンテナーの配列を表します。

配列アクセス・メソッドについては、34ページの『ActiveX の配列』を参照してください。

アクセス・メソッド	説明	ページ
Add()	配列に要素を追加します。	35
GetAt()	指定された位置の要素を戻します。	36
GetSize()	配列内の要素の数を戻します。	36

アクセス・メソッド	説明	ページ
RemoveAt()	指定された位置の要素を削除します。	37

コンテナー要素

コンテナー要素は、コンテナーの任意の要素を表します。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	コンテナー要素オブジェクトを構成します。	86
Copy()	コピーによって、コンテナー要素オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	コンテナー要素オブジェクトの記憶域を割り振り解除します。	91
destructor()	コンテナー要素オブジェクトを破棄します。	91
Equal()	2 つのコンテナー要素を比較します。	89
operator=()	コンテナー要素を別のものに代入します。	88
operator==()	2 つのコンテナー要素を比較します。	89
IsEmpty()	使用できるコンテナー要素情報がないかどうかを示します。	92

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。コンテナー要素はコンテナーの一部を記述するものであるため、すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
ArrayBinaryLength()	I	指定されたコンテナー要素リーフ要素の値の長さを戻します (C 言語)。リーフは配列の一部であり、BINARY 型です。	65

アクセス・メソッド	タイプ	説明	ページ
ArrayBinaryValue()	C	指定されたコンテナー要素リーフ要素の値を戻します (C 言語)。リーフは配列の一部であり、BINARY 型です。	65
ArrayElements()	M	コンテナー要素の配列要素を戻します。	57
ArrayFloatValue()	F	指定されたコンテナー要素リーフ要素の値を戻します (C 言語)。リーフは配列の一部であり、FLOAT 型です。	66
ArrayLongValue()	I	指定されたコンテナー要素リーフ要素の値を戻します (C 言語)。リーフは配列の一部であり、LONG 型です。	66
ArrayStringLength()	I	指定されたコンテナー要素リーフ要素の値の長さを戻します (C 言語)。リーフは配列の一部であり、STRING 型です。	66
ArrayStringValue()	C	指定されたコンテナー要素リーフ要素の値を戻します (C 言語)。リーフは配列の一部であり、STRING 型です。	66
BinaryLength()	I	指定されたコンテナー要素リーフ要素の値の長さを戻します。リーフは BINARY 型です。バイナリーは ActiveX でサポートされていません。	65,67
BinaryValue()	C	指定されたコンテナー要素リーフ要素の値を戻します (C 言語)。リーフは BINARY 型です。バイナリーは ActiveX でサポートされていません。	65
Cardinality()	I	コンテナー要素の配列要素の数を戻します。	57
FloatValue()	F	指定されたコンテナー要素リーフ要素の値を戻します (C 言語)。リーフは FLOAT 型です。	66
FullName()	C	コンテナー要素の完全修飾ドット表記名を戻します。	53
getBuffer()	C	指定されたコンテナー要素リーフ要素の値を戻します (Java)。リーフは BINARY 型です。	67

アクセス・メソッド	タイプ	説明	ページ
getBuffer2()	C	指定されたコンテナー要素リーフ要素の値を戻します (Java)。リーフは配列の一部であり、BINARY 型です。	67
getDouble()	F	指定されたコンテナー要素リーフ要素の値を戻します (Java)。リーフは FLOAT 型です。	67
getDouble2()	F	指定されたコンテナー要素リーフ要素の値を戻します (Java)。リーフは配列の一部であり、FLOAT 型です。	67
GetElement()	O	コンテナー要素の要素にアクセスするためのもの。	58
getLong()	I	指定されたコンテナー要素リーフ要素の値を戻します (Java)。リーフは LONG 型です。	67
getLong2()	I	指定されたコンテナー要素リーフ要素の値を戻します (Java)。リーフは配列の一部であり、LONG 型です。	67
getString()	C	指定されたコンテナー要素リーフ要素の値を戻します (Java)。リーフは STRING 型です。	67
getString2()	C	指定されたコンテナー要素リーフ要素の値を戻します (Java)。リーフは配列の一部であり、STRING 型です。	67
GetValueDbl()	F	指定されたコンテナー・リーフ要素の値を戻します (ActiveX)。リーフは FLOAT 型です。	65
GetValueLng()	I	指定されたコンテナー・リーフ要素の値を戻します (ActiveX)。リーフは LONG 型です。	65
GetValueStr()	C	指定されたコンテナー・リーフ要素の値を戻します (ActiveX)。リーフは STRING 型です。	65
IsArray()	B	コンテナー要素が配列かどうかを示します。	54
IsLeaf()	B	コンテナー要素がリーフかどうかを示します。	54
IsStruct()	B	コンテナー要素が構造自体かどうかを示します。	54

アクセス・メソッド	タイプ	説明	ページ
LeafCount()	I	コンテナー要素のリーフ要素の数を戻します。	55
Leaves()	M	コンテナー要素のすべてのリーフ要素を戻します。	55
LongValue()	I	指定されたコンテナー要素リーフ要素の値を戻します (C 言語)。リーフは LONG 型です。	66
MemberCount()	I	コンテナー要素内の構造メンバーの数を戻します。	56
Name()	C	コンテナー要素の名前を戻します。	53
StringLength()	I	指定されたコンテナー要素リーフ要素の値の長さを戻します (C 言語)。リーフは STRING 型です。	66
StringValue()	C	指定されたコンテナー要素リーフ要素の値を戻します (C 言語)。リーフは STRING 型です。	66
StructMembers()	M	コンテナー要素の構造メンバーを戻します。	56
Type()	C	コンテナー要素の型、つまりデータ構造体の名前を戻します。	53
Value()	C/I/F/N	指定されたコンテナー要素リーフ要素の値を戻します (C++ 言語)。	67

コンテナー要素配列

コンテナー要素配列は、ActiveX におけるコンテナー要素の照会結果を表します。

配列アクセス・メソッドについては、34ページの『ActiveX の配列』を参照してください。

アクセス・メソッド	説明	ページ
Add()	配列に要素を追加します。	35
GetAt()	指定された位置の要素を戻します。	36
GetSize()	配列内の要素の数を戻します。	36
RemoveAt()	指定された位置の要素を削除します。	37

コンテナー要素ベクトル

コンテナー要素ベクトルは、C 言語におけるコンテナー要素の照会結果を表します。

ベクトル関数については、29ページの『C 言語のベクトル』を参照してください。

ベクトル・メソッド	説明
Deallocate()	コンテナー要素ベクトル・オブジェクトを割り振り解除します。
FirstElement()	コンテナー要素ベクトルの第 1 要素を戻します。
NextElement()	コンテナー要素ベクトルの次の要素を戻します。
Size()	コンテナー要素ベクトル内の要素の数を戻します。

制御コネクター配列

制御コネクター配列は、ActiveX における制御コネクター・インスタンスの照会結果を表します。

配列アクセス・メソッドについては、34ページの『ActiveX の配列』を参照してください。

アクセス・メソッド	説明	ページ
GetAt()	指定された位置の要素を戻します。	36
GetSize()	配列内の要素の数を戻します。	36

制御コネクター・インスタンス

制御コネクター・インスタンス・オブジェクトは、2 つのアクティビティー・インスタンスとその状態との間を接続する制御コネクターを表します。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	制御コネクター・インスタンス・オブジェクトを構成します。	86
Copy()	コピーによって、制御コネクター・インスタンス・オブジェクトの記憶域を割り振って初期化します。	90

基本メソッド	説明	ページ
Deallocate()	制御コネクター・インスタンス・オブジェクトの記憶域を割り振り解除します。	91
destructor()	制御コネクター・インスタンス・オブジェクトを破棄します。	91
Equal()	2 つの制御コネクター・インスタンス・オブジェクトを、ソース・アクティビティー・インスタンスとターゲット・アクティビティー・インスタンスという関係に基づいて比較します。	89
IsEmpty()	使用できる制御コネクター・インスタンス情報がないかどうかを示します。	92
Kind()	制御コネクター・インスタンスの種類を示します (分岐条件または "他方向" コネクター)。	93, 111
operator=()	制御コネクター・インスタンス・オブジェクトをこれに代入します。	88
operator==()	2 つの制御コネクター・インスタンス・オブジェクトを、ソース・アクティビティー・インスタンスとターゲット・アクティビティー・インスタンスという関係に基づいて比較します。	89

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
BendPoints()	M	制御コネクター・インスタンスの屈曲点を戻します。	130
Name()	C	制御コネクター・インスタンスに関連付けられている名前を戻します。	128
NameIsNull()	B	名前が設定されているかどうかを示します。	134

アクセス・メソッド	タイプ	説明	ページ
PersistentOidOfSourceActivity()	C	この制御コネクター・インスタンスのソースとなっているアクティビティー・インスタンスのオブジェクト ID を戻します。	128
PersistentOidOfTargetActivity()	C	この制御コネクター・インスタンスのターゲットとなっているアクティビティー・インスタンスのオブジェクト ID を戻します。	128
State()	E	制御コネクター・インスタンスの状態、それが評価されているのか、そして評価されているのであればその結果を戻します。	102, 110
TransitionCondition()	C	制御コネクター・インスタンスの分岐条件を戻します。	128
TransitionConditionIsNull()	B	分岐条件が設定されているかどうかを示します。	134

制御コネクター・インスタンス・ベクトル

制御コネクター・インスタンス・ベクトルは、C 言語における制御コネクター・インスタンスの照会結果を表します。

ベクトル・アクセス関数については、29ページの『C 言語のベクトル』を参照してください。

アクセス・メソッド	説明
Deallocate()	制御コネクター・インスタンス・ベクトル・オブジェクトを割り振り解除します。
FirstElement()	制御コネクター・インスタンス・ベクトルの第 1 要素を戻します。
NextElement()	制御コネクター・インスタンス・ベクトルの次の要素を戻します。
Size()	制御コネクター・インスタンス・ベクトル内の要素の数を戻します。

DateAndTime/ FmcjDateTime/ FmcjCDateTime

DateAndTime オブジェクトは、ActiveX 言語において日付時刻の値を表します。 FmcjDateTime オブジェクトは、C++ 言語において日付時刻の値を表します。 FmcjCDateTime 日時は、C 言語において日付時刻の値を表します。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。次に示すメソッドは、C++ 言語でしか使えません。

基本メソッド	説明	ページ
constructor()	日付 / 時刻オブジェクトを構成します。	86
destructor()	日付 / 時刻オブジェクトを破棄します。	91
operator=()	日付 / 時刻を別のものに代入します。	88
operator==()	2 つの日付 / 時刻オブジェクトを比較します。	89
operator string()	日付 / 時刻オブジェクトの文字列表現を戻します。	128
IsEmpty()	使用できる日付 / 時刻情報がないかどうかを示します。	92

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。日付 / 時刻オブジェクトはクライアントのみのサポート・オブジェクトを表すものなので、1 次属性と 2 次属性の区別は適用されません。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

次に示すメソッドは、C++ 言語および ActiveX でしか使えません。

アクセス・メソッド	タイプ	説明	ページ
Day()	I	日付 / 時刻オブジェクトの日付を戻します。	127
Hour()	I	日付 / 時刻オブジェクトの時間値を戻します。	127
Minute()	I	日付 / 時刻オブジェクトの分を戻します。	127
Month()	I	日付 / 時刻オブジェクトの月を戻します。	127
Second()	I	日付 / 時刻オブジェクトの秒を戻します。	127

アクセス・メソッド	タイプ	説明	ページ
Year()	I	日付 / 時刻オブジェクトの年を戻します。	127

次に示すメソッドは、C 言語でしか使えません。

アクセス関数	タイプ	説明	ページ
FmcjDateTimeAsString	C	日付 / 時刻構造体の文字列表現を戻します。	128
FmcjDateTimeCurrentTime	D	現在の日付 / 時刻を戻します。	100
FmcjDateTimeIsValid	B	渡される日付 / 時刻が有効な日付 / 時刻であるかどうかを示します。これは、1970-1-1 12:00 (yyyy-mm-dd hh:mm) 以上でなければなりません。	99

Dll オプション

DllOptions オブジェクトは、ダイナミック・リンク・ライブラリーのプログラム・インプリメンテーション定義を表します。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	DLL オプション・オブジェクトを構成します。	86
Copy()	コピーによって、DLL オプション・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	DLL オプション・オブジェクトの記憶域を割り振り解除します。	91
destructor()	DLL オプション・オブジェクトを破棄します。	91
IsEmpty()	入手できる DLL オプション情報がないかどうかを示します。	92
operator=()	DLL オプション・オブジェクトをこれに代入します。	88

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジ

ェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
EntryPointName()	C	DLL のエントリー・ポイントの名前を戻します。	128
ExecuteFenced()	B	DLL を別のアドレス空間で実行する必要があるかどうかを示します。	99
ExecuteFencedIsNull()	B	隔離実行が設定されているかどうかを示します。	134
KeepLoaded()	B	DLL をロードしたままにする必要があるかどうかを示します。	99
KeepLoadedIsNull()	B	ロード保持が設定されているかどうかを示します。	134
PathAndFileName()	C	DLL のパスとファイル名を戻します。	128

ExecutionAgent/FmcjPEA

PEA または ExecutionAgent オブジェクトは、MQSeries Workflow プログラム実行エージェントを表します。

アクティビティー・インプリメンテーション関数 / メソッドについては、141 ページの『アクティビティー・インプリメンテーション関数 / メソッド』を参照してください。

アクティビティー・インプリメンテーション・メソッド	説明	ページ
InContainer()	アクティビティー・インプリメンテーション内から入力コンテナーにアクセスします (Java)。 Java 以外では Container を参照。	393
OutContainer()	アクティビティー・インプリメンテーション内から出力コンテナーにアクセスします (Java)。 Java 以外では Container を参照。	396
PersistentOidOfActivityInstance()	関連するアクティビティー・インスタンスのオブジェクト ID を戻します。	143

アクティビティー・インプリメンテーション・メソッド	説明	ページ
ProgramID()	呼び出されたアクティビティー・インプリメンテーションまたはサポート・ツールがプログラム実行エージェントに認識されるのに必要なプログラム識別子を戻します。	143
RemoteInContainer()	アクティビティー・インプリメンテーションによって開始されたプログラム内から入力コンテナーにアクセスします (Java)。 Java 以外では Container を参照。	398
RemoteOutContainer()	アクティビティー・インプリメンテーションによって開始されたプログラム内から出力コンテナーにアクセスします (Java)。 Java 以外では Container を参照。	401
RemotePersistentOid OfActivityInstance()	このプログラムを開始したアクティビティー・インプリメンテーションが最初に開始された時点で対象となったオブジェクト ID を戻します。	143
RemoteUserID()	このプログラムを開始したアクティビティー・インプリメンテーションが最初に開始された時点で対象となったユーザー識別子を戻します。	143
UserID()	アクティビティー・インプリメンテーションの開始の対象となったユーザー識別子を戻します。	143
SetOutContainer()	アクティビティー・インプリメンテーション内から出力コンテナーを設定します (Java)。 Java 以外では Container を参照。	403
SetRemoteOutContainer()	アクティビティー・インプリメンテーションによって開始されたプログラム内から出力コンテナーを設定します (Java)。 Java 以外では Container を参照。	406

実行データ

実行データ・オブジェクトは、MQSeries Workflow 実行サーバーから送信されたデータを表します。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	実行データ・オブジェクトを構成します。	86
Copy()	コピーによって、実行データ・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	実行データ・オブジェクトの記憶域を割り振り解除します。	91
destructor()	実行データ・オブジェクトを破棄します。	91
IsEmpty()	使用できる実行データ情報がないかどうかを示します。	92
Kind()	データの種類 (作業項目の作成、削除などを記述しているかどうか、など) を戻します。	93, 112
operator=()	実行データ・オブジェクトをこれに代入します。	88

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
ActivityInstanceNotificationFromData()	P	実行データからアクティビティー・インスタンス通知を作成します。	132
ErrorFromData()	P	実行データからエラー記述オブジェクトを作成します。	132
IsError()	B	実行データがエラー状態を記述しているかどうかを示します。	99

アクセス・メソッド	タイプ	説明	ページ
PersistentOid()	C	実行データが記述するオブジェクトのオブジェクト ID の表現を戻します。	128
ProcessInstanceFromData()	P	実行データからプロセス・インスタンスを作成します。	132
ProcessInstanceNotificationFromData()	P	実行データからプロセス・インスタンス通知を作成します。	132
ReadOnlyContainerFromData()	P	実行データからコンテナー・オブジェクトを作成します。	132
WorkitemFromData()	P	実行データから作業項目を作成します。	132
UserContext()	C	ユーザー・コンテキストを戻します。	128
UserContextIsNull()	B	ユーザー・コンテキストが指定されているかどうかを示します。	134

実行サービス

実行サービス・オブジェクトは、実行サーバーとのユーザー・セッションを表します。 **FmcjService によって提供されるすべての関数 / メソッドも使用できます。**

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
Allocate()	実行サービス・オブジェクトの記憶域を割り振ります。接続先の実行サーバーは、MQSeries Workflow ユーザーのプロファイルまたは現在設定されている構成の構成プロファイルから取られます。	86

基本メソッド	説明	ページ
AllocateForSystem()	指定された実行サービス・オブジェクトの記憶域を割り振ります。接続先の実行サーバーは、現在設定されている構成の中で指定されているパラメーターから取られます。	86
AllocateForGroup()	指定された実行サービス・オブジェクトの記憶域を割り振ります。接続先の実行サーバーは、現在設定されている構成の中で指定されているシステム・グループ内の任意のシステムです。	86
constructor()	実行サービス・オブジェクトを構成します。	86
Copy()	コピーによって、実行サービス・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	実行サービス・オブジェクトの記憶域を割り振り解除します。	91
destructor()	実行サービス・オブジェクトを破棄します。	91
Equal()	2 つの実行サービス・オブジェクトを比較して、それらが同じセッションを表すものかどうかを調べます。	89
operator=()	実行サービス・オブジェクトをこれに代入します。	88
operator==()	2 つの実行サービス・オブジェクトを比較して、それらが同じセッションを表すものかどうかを調べます。	89

アクション関数 / メソッドについては、141ページの『アクション関数 / メソッド』を参照してください。

アクション・メソッド	説明	ページ
CreateProcessInstanceList()	実行サーバー上で新しいプロセス・インスタンス・リストを作成します。	412
CreateProcessTemplateList()	実行サーバー上で新しいプロセス・テンプレート・リストを作成します。	420

アクション・メソッド	説明	ページ
CreateWorklist()	実行サーバー上で新しいワークリストを作成します。	427
Logoff()	接続されている実行サーバーからログオフします。	436
Logon()	実行サーバーにログオンします。	439
Logon2()	(Java) 実行サーバーにログオンし、追加パラメーターを提供します。	439
persistentActivityInstance()	Java API で渡されたオブジェクト識別子で指定されるアクティビティー・インスタンスを取り出します。	363
persistentActivityInstanceNotification()	Java API で渡されたオブジェクト識別子で指定されるアクティビティー・インスタンス通知を取り出します。	376
persistentProcessInstance()	Java API で渡されたオブジェクト識別子で指定されるプロセス・インスタンスを取り出します。	580
persistentProcessInstanceNotification()	Java API で渡されたオブジェクト識別子で指定されるプロセス・インスタンス通知を取り出します。	613
persistentProcessTemplate()	Java API で渡されたオブジェクト識別子で指定されるプロセス・テンプレートを取り出します。	645
persistentWorkItem()	Java API で渡されたオブジェクト識別子で指定される作業項目を取り出します。	703
QueryActivityInstanceNotifications()	ログオンしたユーザーがアクセスできるアクティビティー・インスタンス通知を取り出します。	452
QueryItems()	ログオンしたユーザーがアクセスできる作業項目または通知を取り出します。	460
QueryProcessInstanceLists()	ログオンしたユーザーがアクセスできるプロセス・インスタンス・リストを取り出します。	468
QueryProcessInstanceNotifications()	ログオンしたユーザーがアクセスできるプロセス・インスタンス通知を取り出します。	470

アクション・メソッド	説明	ページ
QueryProcessInstances()	ログオンしたユーザーがアクセスできるプロセス・インスタンスを取り出します。	478
QueryProcessTemplateLists()	ログオンしたユーザーがアクセスできるプロセス・テンプレート・リストを取り出します。	484
QueryProcessTemplates()	ログオンしたユーザーがアクセスできるプロセス・テンプレートを取り出します。	486
QueryWorkitems()	ログオンしたユーザーがアクセスできる作業項目を取り出します。	492
QueryWorklists()	ログオンしたユーザーがアクセスできるワークリストを取り出します。	500
Receive()	MQSeries Workflow 実行サーバーが送信した実行データを受信します。	502
SetPersonAbsent()	指定したユーザーの不在または在席を指定します。	509
TerminateReceive()	情報をクライアント入力キューに入れて、MQSeries Workflow 実行サーバーが送信した実行データの受信を終了できることを示します。	512

アクティビティー・インプリメンテーション関数 / メソッドについては、141 ページの『アクティビティー・インプリメンテーション関数 / メソッド』を参照してください。

アクティビティー・インプリメンテーション・メソッド	説明	ページ
Passthrough()	アクティビティー・インプリメンテーションと実行サーバーとの間にセッションを確立します。	445
RemotePassthrough()	アクティビティー・インプリメンテーションによって開始されたプログラムと実行サーバーとの間にセッションを確立します。	506

プログラム実行管理関数 / メソッドについては、147ページの『プログラム実行管理関数 / メソッド』を参照してください。

管理メソッド	説明	ページ
PEAShutDown()	ユーザーに関連したプログラム実行エージェントをシャットダウンするよう要求します。	447
PEAStartUp()	ユーザーに関連したプログラム実行エージェントを開始します。	450

実行サービス配列

実行サービス配列は、実行サービスを保持するための ActiveX の手段です。

配列アクセス・メソッドについては、34ページの『ActiveX の配列』を参照してください。

アクセス・メソッド	説明	ページ
Add()	配列に要素を追加します。 MQSeries Workflow ユーザー・プロファイル、または現在設定されている構成の構成プロファイルから、指定されたシステムおよびシステム・グループが検索されます。	35
AddDefault()	配列に実行サービスを追加します。 MQSeries Workflow ユーザー・プロファイル、または現在設定されている構成から、システムおよびシステム・グループが検索されます。	35
AddSystemGroup()	配列に実行サービスを追加します。指定されたシステム・グループ内の任意のシステムを取り出すことができます。それは、MQSeries Workflow ユーザーのプロファイルまたは現在設定されている構成の構成プロファイルから取られます。	35
GetAt()	指定された位置の要素を戻します。	36
GetSize()	配列内の要素の数を戻します。	36
RemoveAt()	指定された位置の要素を削除します。	37

イベント	説明	ページ
ExecutionServiceRemove()	実行サービスを配列から除去します。	38
NewExecutionService()	配列に新しい実行サービスを追加します。	37

EXE オプション

ExeOptions オブジェクトは、実行可能プログラムのプログラム・インプリメンテーション定義を表します。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	EXE オプション・オブジェクトを構成します。	86
Copy()	コピーによって、EXE オプション・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	EXE オプション・オブジェクトの記憶域を割り振り解除します。	91
destructor()	EXE オプション・オブジェクトを破棄します。	91
operator=()	EXE オプション・オブジェクトをこれに代入します。	88
IsEmpty()	入手できる EXE オプション情報がないかどうかを示します。	92

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
AutomaticClose()	B	EXE 終了時に、EXE を開始したウィンドウをクローズするかどうかを示します。	99
AutomaticCloseIsNull()	B	自動クローズが設定されているかどうかを示します。	134
Environment()	C	EXE の環境設定を示します。	128
EnvironmentIsNull()	B	環境が設定されているかどうかを示します。	134
InheritEnvironment()	B	環境設定を、オペレーティング・システムの環境設定と組み合わせる必要があるかどうかを示します。	99

アクセス・メソッド	タイプ	説明	ページ
PathAndFileName()	C	EXE のパスとファイル名を戻します。	128
RunInXTerm()	B	EXE を別の xterm で開始する必要があるかどうかを示します。	99
RunInXTermIsNull()	B	xterm での実行が設定されているかどうかを示します。	134
StartInForeGround()	B	EXE を前景で開始するかどうかを示します。	99
StartInForeGroundIsNull()	B	前景での開始が設定されているかどうかを示します。	134
WindowStyle()	E	初期ウィンドウ・スタイルを示します。	102, 115
WindowStyleIsNull()	B	ウィンドウ・スタイルが設定されているかどうかを示します。	134
WorkingDirectoryName()	C	EXE の作業ディレクトリーを示します。	128
WorkingDirectoryNameIsNull()	B	作業ディレクトリーが設定されているかどうかを示します。	134

外部サービス・オプション

ExternalOptions オブジェクトは、外部サービスのプログラム・インプリメンテーション定義を表します。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	外部オプション・オブジェクトを構成します。	86
Copy()	コピーによって、外部オプション・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	外部オプション・オブジェクトの記憶域を割り振り解除します。	91
destructor()	外部オプション・オブジェクトを破棄します。	91
operator=()	外部オプション・オブジェクトをこれに代入します。	88
IsEmpty()	入手できる外部オプション情報がないかどうかを示します。	92

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
BackwardMappingFormat()	C	実行可能ファイルが使用する構造体から MQSeries Workflow コンテナーへのマッピングの形式を指定します。	128
BackwardMappingFormatIsNull()	B	逆方向マッピング形式が設定されているかどうかを示します。	134
BackwardMappingParameters()	M	逆方向マッピング・パラメーターがあれば、それを戻します。	130
BackwardMappingParametersIsNull()	B	逆方向マッピング・パラメーターが設定されているかどうかを示します。	134
CodePage()	I	サービスのコード・ページを指定します。	127
CodePageIsNull()	B	コード・ページが設定されているかどうかを示します。	134
ExecutableName()	C	起動タイプおよびサービスによって実行可能ファイルが呼び出されるように指定します。	128
ExecutableType()	C	実行可能ファイルのタイプを識別します。	128
ForwardMappingFormat()	C	MQSeries Workflow コンテナーから、実行可能ファイルが使用する構造体へのマッピングの形式を指定します。	128

アクセス・メソッド	タイプ	説明	ページ
ForwardMappingFormatIsNull()	B	順方向マッピング形式が設定されているかどうかを示します。	134
ForwardMappingParameters()	M	順方向マッピング・パラメーターがあれば、それを戻します。	130
ForwardMappingParametersIsNull()	B	順方向マッピング・パラメーターが設定されているかどうかを示します。	134
InvocationType()	C	サービスにおいて実行可能ファイルを呼び出すための呼び出しメカニズムを指定します。	128
IsLocalUser()	B	MQSeries Workflow ユーザー ID を使わないでローカル・ユーザーが解決されるかどうかを戻します。	99
IsMappingRoutineCall()	B	順方向および逆方向のマッピング・ルーチンが呼び出されるかどうかを指定します。	99
IsSecurityRoutineCall()	B	セキュリティー・ルーチンが呼び出されるかどうかを指定します。	99
MappingType()	C	発生することになっているマッピングのタイプを識別します。	128
MappingTypeIsNull()	B	マッピング・タイプが設定されているかどうかを示します。	134
ServiceName()	C	呼び出されるサービスを識別します。	128
ServiceType()	C	呼び出されるサービスのタイプ (CICS(R) または IMS(TM)) を識別します。	128

アクセス・メソッド	タイプ	説明	ページ
TimeoutPeriod()	E	プログラム実行エージェントが、開始したサービスからの応答を待機する時間 (無限、一定の期間、または待機しない) を指定します。	102, 116
TimeoutInterval()	I	タイムアウト間隔を秒数で指定します。	127
TimeoutIntervalIsNull()	B	タイムアウトの時間間隔が設定されているかどうかを示します。	134

FmcError

FmcError または FmcjError オブジェクトは、作業項目またはアクティビティー・インスタンスが InError 状態である理由の説明を表示します。また、非同期応答として戻されるエラーについて説明します。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	エラー・オブジェクトを構成します。	86
Copy()	コピーによって、エラー・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	エラー・オブジェクトの記憶域を割り振り解除します。	91
destructor()	エラー・オブジェクトを破棄します。	91
Equal()	2 つのエラー・オブジェクトを、戻りコードとパラメーターに基づいて比較します。	89
IsEmpty()	使用できるエラー情報がないかどうかを示します。	92
operator=()	エラー・オブジェクトをこれに代入します。	88
operator==()	2 つのエラー・オブジェクトを、戻りコードとパラメーターに基づいて比較します。	89

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
MessageText()	C	NLS 形式化メッセージとしてエラーを戻します。	128
Parameters()	M	エラーのパラメーターを戻します。そのパラメーターはメッセージ・テキストに組み込まれます。	130
Rc()	I	エラー・オブジェクトに保管されている戻りコードを戻します。	127

FmcException

FmcException オブジェクトは、Java によって送出された例外の記述を表します。

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
MessageText()	C	NLS 形式化メッセージとして例外を戻します。	128
nestedException()	-	通信層から出される例外を戻します。 **注:** ネストされた例外は、org.omg.CORBA.SystemException または java.rmi.RemoteException のどちらか (使用する通信プロトコルによって異なる) にダウン・キャストすれば、検査することができます。しかし、そうするとクライアント・コードはプロトコル従属になってしまいます (両方のケースを処理する場合を除く)。ローカル・バインディングを使用する場合、ネスト例外は必ず空です。	128
origin()	C	例外が発生したモジュールを戻します。	128

アクセス・メソッド	タイプ	説明	ページ
Parameters()	M	エラーのパラメーターを戻します。そのパラメーターはすでにメッセージ・テキストに組み込まれています。	130
Rc()	I	エラー・オブジェクトに保管されている戻りコードを戻します。	127

Global

API グローバル・オブジェクトは、グローバルMQSeries Workflow API 関数 / メソッドを分類するのに使用されます。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明
Connect()	現在のスレッドで API を初期化します。Java ではサポートされていません。
Disconnect()	現在のスレッドで API を消去します。Java ではサポートされていません。

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
ConfigurationID()	C	プロファイル・アクセスに使用する構成 ID を戻します。	128
SetConfigurationID()	C	プロファイル・アクセスに使用する構成 ID を設定します。最初にプロファイルを使用する前にのみ設定できます。通常は Logon() です。 1 つのアプリケーション・プロセスには 1 つの構成だけしか使用できません。	128

インプリメンテーション・データ

インプリメンテーション・データ・オブジェクトは、プログラム・インプリメンテーション定義を表します。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	インプリメンテーション・データ・オブジェクトを構成します。	86
Copy()	コピーによって、インプリメンテーション・データ・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	インプリメンテーション・データ・オブジェクトの記憶域を割り振り解除します。	91
destructor()	インプリメンテーション・データ・オブジェクトを破棄します。	91
operator=()	インプリメンテーション・データ・オブジェクトをこれに代入します。	88
IsEmpty()	使用できるインプリメンテーション・データ情報がないかどうかを示します。	92
Kind()	インプリメンテーション・データの実際の種類 (DLL か EXE か) を示します。	93, 118

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
CommandLineParameters()	C	呼び出されるプログラムに渡すコマンド行パラメーターを戻します。	128
CommandLineParametersIsNull()	B	コマンド行パラメーターが設定されているかどうかを示します。	134

アクセス・メソッド	タイプ	説明	ページ
DllOptions()	P	インプリメンテーションが DLL の場合に、DLL の記述を戻します。	132
ExeOptions()	P	インプリメンテーションが EXE の場合に、EXE の記述を戻します。	132
ExternalOptions()	P	インプリメンテーションが外部サービスの場合に、外部オプションの記述を戻します。	132
options()	P	EXE、DLL、または Java の外部サービスの記述を戻します。	132
Platform()	E	このインプリメンテーション・データが記述しているオペレーティング・システム・プラットフォームを戻します。	102, 117

インプリメンテーション・データ・ベクトル

インプリメンテーション・ベクトルは、C 言語でのインプリメンテーションの照会結果を表します。

ベクトル・アクセス関数については、29ページの『C 言語のベクトル』を参照してください。

アクセス・メソッド	説明
Deallocate()	インプリメンテーション・ベクトル・オブジェクトを割り振り解除します。
FirstElement()	インプリメンテーション・データ・ベクトルの第 1 要素を戻します。
NextElement()	インプリメンテーション・データ・ベクトルの次の要素を戻します。
Size()	インプリメンテーション・データ・ベクトルの要素の数を戻します。

インスタンス・モニター

インスタンス・モニター・オブジェクトは、ActiveX API におけるモニターを表します。プロセス・インスタンスのモニター、ブロック 型のアクティビティー・インスタンスのモニター、またはプロセス 型のアクティビティー・インスタンスのモニターのいずれかです。

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
ActivityInstances()	M	インスタンス・モニターによって表されているアクティビティー・インスタンスを戻します。アクティビティー・インスタンスには、1 次値と 2 次値の両方が含まれます。	130
ControlConnectorInstances()	M	インスタンス・モニターによって表されている制御コネクター・インスタンスを戻します。	130

アクション関数 / メソッドについては、141ページの『アクション関数 / メソッド』を参照してください。

アクション・メソッド	説明	ページ
ObtainInstanceMonitor()	ブロック またはプロセス 型のアクティビティー・インスタンスのインスタンス・モニターを戻します。アクティビティー・インスタンスはインスタンス・モニターが表現するアクティビティー・インスタンスのセットの一部です。	383
Refresh()	MQSeries Workflow 実行サーバーからインスタンス・モニターをリフレッシュします。	389

Item

アイテムは、作業項目、アクティビティー・インスタンス通知、またはプロセス・インスタンス通知を表します。したがって、**アイテムのすべての関数 / メソッドは、作業項目、アクティビティー・インスタンス通知、およびプロセス・インスタンス通知にも使用されます。**

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	アイテム・オブジェクトを構成します。	86
Copy()	コピーによって、アイテム・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	アイテム・オブジェクトの記憶域を割り振り解除します。	91
destructor()	アイテム・オブジェクトを破棄します。	91
Equal()	2 つのアイテムを比較します。	89
IsComplete()	完全なアイテム情報を入手できるかどうかを示します。	91
IsEmpty()	使用できるアイテム情報がないかどうかを示します。	92
Kind()	アイテムの実際の種類 (作業項目か、それともなんらかの通知か) を示します。	93, 122
operator=()	アイテム・オブジェクトをこれに代入します。	88
operator==()	2 つのアイテムを比較します。	89

アクセスおよび mutator 関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。

注: 「**設定**」欄の値は、該当属性が 1 次属性 (P) であってアイテムが照会されるとただちに設定されるのか、それともこの属性が 2 次属性 (S) であって個々のアイテムのリフレッシュの後にしか設定されないのかを示しています。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	設定 / 型	説明	ページ
Category()	P/C	アイテムのプロセス・カテゴリーを戻します。	128
CategoryIsNull()	P/B	カテゴリーが設定されているかどうかを示します。	134
CreationTime()	P/D	アイテムの作成時刻を戻します。	100
Description()	P/C	アイテムの記述を戻します。	128
DescriptionIsNull()	P/B	記述が設定されているかどうかを示します。	134
Documentation()	S/C	アイテムの文書を戻します。	128
DocumentationIsNull()	S/B	文書が設定されているかどうかを示します。	134
EndTime()	S/D	アイテムの終了時刻を戻します。	100
EndTimeIsNull()	S/B	終了時刻がセットされているかどうかを示します。	134
Icon()	P/C	アイテムに関連付けられているアイコンを戻します。	128
InContainerName()	S/C	アイテムの入力コンテナーの名前を戻します。	128
LastModificationTime()	P/D	アイテムの 1 次属性が最後に変更された時刻を戻します。	100
Name()	P/C	アイテムの名前を戻します。C 言語の場合、作業項目やアクティビティー・インスタンス通知は 33 バイト以上のバッファーを、プロセス・インスタンス通知は 64 バイト以上のバッファーを必要とします。	128
OutContainerName()	S/C	アイテムの出力コンテナーの名前を戻します。	128
Owner()	P/C	アイテム所有者のユーザー ID を戻します。	128
PersistentOid()	P/C	アイテムのオブジェクト識別子の表現を戻します。	128

アクセス・メソッド	設定 / 型	説明	ページ
PersistentOidOfProcessInstance()	P/C	関連するプロセス・インスタンスのオブジェクト ID を戻します。	128
ProcessAdmin()	S/C	アイテムのプロセス管理者のユーザー ID を戻します。	128
ProcessInstanceName()	P/C	アイテムが属するプロセス・インスタンスの名前を戻します。	128
ProcessInstanceState()	P/E	アイテムが属するプロセス・インスタンスの状態を戻します。	102, 125
ProcessInstanceSystemGroupName()	S/C	アイテムが属するプロセス・インスタンスのシステム・グループの名前を戻します。	128
ProcessInstanceSystemName()	S/C	アイテムが属するプロセス・インスタンスのシステムの名前を戻します。	128
ReceivedAs()	P/E	アイテムが受信された理由を戻します。	102, 103
ReceivedTime()	P/D	現在の所有者によってアイテムが受信された時刻を戻します。	100
StartTime()	P/D	アイテムの開始時刻を戻します。	100
StartTimeIsNull()	P/B	開始時刻が設定されているかどうかを示します。	134

mutator メソッド	説明	ページ
Update()	MQSeries Workflow 実行サーバーが送信した実行データに合わせてアイテムを更新します。アイテムのオブジェクト ID と実行データが記述するオブジェクトのオブジェクト ID とは同じでなければなりません。	137

アクション関数 / メソッドについては、141ページの『アクション関数 / メソッド』を参照してください。

アクション・メソッド	説明	ページ
Delete()	アイテムを削除します。	521
ObtainProcessInstanceMonitor()	アイテムが属するプロセス・インスタンスのプロセス・インスタンス・モニターを取り出します。	524
ProcessInstance()	アイテムが属するプロセス・インスタンスを取り出します。	527
Refresh()	アイテムの完全情報を取り出します。	530
SetDescription()	アイテムの記述を設定します。	533
SetName()	アイテムの名前を設定します。	536
Transfer()	指定されたユーザーにアイテムを転送します。	539

アイテム・ベクトル

アイテム・ベクトルは、C 言語でのアイテムの照会結果を表します。

ベクトル・アクセス関数については、29ページの『C 言語のベクトル』を参照してください。

アクセス・メソッド	説明
Deallocate()	アイテム・ベクトル・オブジェクトを割り振り解除します。
FirstElement()	アイテム・ベクトルの第 1 要素を戻します。
NextElement()	アイテム・ベクトルの次の要素を戻します。
Size()	アイテム・ベクトル内の要素の数を戻します。

メッセージ

メッセージ・オブジェクトは、MQSeries Workflow 提供のメッセージ・カタログにアクセスするのに使います。

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
MessageText()	C	メッセージ ID に基づいて、NLS 形式化メッセージを戻します。渡されるパラメーターも組み込まれます。	128

永続リスト

永続リストは、永続リスト定義を表します。**永続リストのすべての関数 / メソッドは、プロセス・インスタンス・リスト、プロセス・テンプレート・リスト、ワークリストに対しても使用できます。**

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
IsEmpty()	使用できる永続リスト情報がないかどうかを示します。	92

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
Description()	C	永続リストの記述を戻します。	128
DescriptionIsNull()	B	記述が設定されているかどうかを示します。	134
Filter()	C	永続リストのフィルターを戻します。	128
FilterIsNull()	B	フィルターが設定されているかどうかを示します。	134
Name()	C	永続リストの名前を戻します。	128
OwnerOfList()	C	永続リストの所有者のユーザー ID を戻します。	128
OwnerOfListIsNull()	B	所有者が設定されているかどうかを示します。パブリック・リストに所有者はありません。	134
SortCriteria()	C	永続リストのソート基準を戻します。	128

アクセス・メソッド	タイプ	説明	ページ
SortCriteriaIsNull()	B	ソート基準が設定されているかどうかを示します。	134
Threshold()	I	永続リストのしきい値を戻します。	127
ThresholdIsNull()	B	しきい値が設定されているかどうかを示します。	134
Type()	E	永続リストのタイプがパブリック (public) かプライベート (private) のどちらのリストかを示します。	102, 123

アクション関数 / メソッドについては、141ページの『アクション関数 / メソッド』を参照してください。

アクション・メソッド	説明	ページ
Delete()	永続リストを削除します。	544
Refresh()	永続リストをリフレッシュします。	546
SetDescription()	永続リストの記述を設定します。	549
SetFilter()	永続リストのフィルターを設定します。	552
SetSortCriteria()	永続リストのソート基準を設定します。	555
SetThreshold()	永続リストのしきい値を設定します。	558

担当者

担当者オブジェクトは、ログオン・ユーザーの設定値を表します。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	担当者オブジェクトを構成します。	86
Copy()	コピーによって、担当者オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	担当者オブジェクトの記憶域を割り振り解除します。	91
destructor()	担当者オブジェクトを破棄します。	91
Equal()	2 つの担当者を比較します。	89
operator=()	担当者をこれに代入します。	88
operator==()	2 つの担当者を比較します。	89

基本メソッド	説明	ページ
IsComplete()	完全な担当者情報を入手できるかどうかを示します。	91
IsEmpty()	使用できる担当者情報がないかどうかを示します。	92

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。

注: 「**設定**」欄の値は、該当属性が 1 次属性 (P) であって担当者が照会されるとただちに設定されるのか、それともこの属性が 2 次属性 (S) であって個々の担当者のリフレッシュの後にしか設定されないのかを示しています。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	設定 / 型	説明	ページ
CategoriesAuthorizedFor()	P/M	担当者が基本的な特権または管理者特権を許可されているカテゴリーを戻します。担当者がすべてのカテゴリーに関して管理者権限を持っている場合、ここではカテゴリーは戻されません。担当者がすべてのカテゴリーに関して基本権限を持っている場合、管理権限で許可されているカテゴリーが戻されます。	130

アクセス・メソッド	設定 / 型	説明	ページ
CategoriesAuthorizedForAsAdmin()	P/M	担当者が管理者特権を許可されているカテゴリーを戻します。担当者がすべてのカテゴリーに関して管理者権限を持っている場合、ここではカテゴリーは戻されません。	130
Description()	P/C	担当者の記述を戻します。	128
DescriptionIsNull()	P/B	記述が設定されているかどうかを示します。	134
FirstName()	P/C	担当者のファーストネームを戻します。	128
FirstNameIsNull()	P/B	ファーストネームが設定されているかどうかを示します。	134
IsAbsent()	P/B	担当者が不在かどうかを示します。	99
IsAdminForCategory()	P/B	指定したカテゴリーの管理者権限が担当者に付与されているかどうかを示します。カテゴリーが存在しないなら、偽を戻します。担当者がすべてのカテゴリーに関して管理者権限を持っている場合、カテゴリーが存在するかどうかに関係なく、真を戻します。	99
IsAdministrator()	S/B	担当者が管理者かどうかを示します。	99

アクセス・メソッド	設定 / 型	説明	ページ
IsAuthorizedForAllCategories()	P/B	担当者がすべてのカテゴリーに関して基本的な特権または管理者特権 (あるいは両方) を付与されているかどうかを示します。	99
IsAuthorizedForAllCategoriesAsAdmin()	P/B	担当者がすべてのカテゴリーに関して管理者特権を付与されているとされているかどうかを示します。	99
IsAuthorizedForAllPersons()	P/B	担当者が、すべての担当者のアイテムを見ることを許可されているかどうかを示します。	99
IsAuthorizedForAuthorizationDefinition()	P/B	担当者が、権限を定義することを許可されているかどうかを示します。	99
IsAuthorizedForOperationAdministration()	P/B	担当者が、運用管理の権限を付与されているかどうかを示します。	99
IsAuthorizedForProcessDefinition()	P/B	担当者が、プロセス・モデルを定義することを許可されているかどうかを示します。	99
IsAuthorizedForStaffDefinition()	P/B	担当者が、担当者を定義することを許可されているかどうかを示します。	99
IsAuthorizedForTopologyDefinition()	P/B	担当者が、トポロジー・データを定義することを許可されているかどうかを示します。	99
IsManager()	S/B	担当者がマネージャーかどうかを示します。	99

アクセス・メソッド	設定 / 型	説明	ページ
IsResetAbsence()	P/B	担当者のログオン時に不在フラグをリセットする必要があるかどうかを示します。	99
LastName()	P/C	担当者のラストネームを戻します。	128
LastNameIsNull()	P/B	ラストネームが設定されているかどうかを示します。	134
Level()	P/I	担当者のレベルを戻します。	127
Manager()	S/C	担当者のマネージャーのユーザー識別子を戻します。	128
ManagerIsNull()	S/B	担当者のマネージャーが設定されているかどうかを示します。	134
MiddleName()	P/C	担当者のミドルネームを戻します。	128
MiddleNameIsNull()	P/B	ミドルネームが設定されているかどうかを示します。	134
NamesOfManagedOrganizations()	S/M	担当者が管理する組織の名前を戻します。	130
NamesOfRoles()	P/M	担当者が属する役割の名前を戻します。	130
NamesOfRolesToCoordinate()	S/M	担当者がコーディネーターとなる役割の名前を戻します。	130
OrganizationName()	P/C	担当者が属する組織の名前を戻します。	128
OrganizationNameIsNull()	P/B	組織名が設定されているかどうかを示します。	134
PersonID()	P/C	担当者の担当者 ID を戻します。	128

アクセス・メソッド	設定 / 型	説明	ページ
PersonIDIsNull()	P/B	担当者 ID が設定されているかどうかを示します。	134
PersonsAuthorizedFor()	P/M	この担当者が明示的にまたは置換によって代行する担当者を戻します。担当者が他の**すべての**担当者について権限を付与されている場合、ここで担当者は戻されません。	130
PersonsAuthorizedForMe()	S/M	この担当者について許可されている担当者を戻します。	130
PersonsToStandInFor()	S/M	この担当者が代行する担当者を戻します。	130
Phone()	P/C	担当者の電話番号を戻します。	128
PhoneIsNull()	P/B	電話が設定されているかどうかを示します。	134
SecondPhone()	P/C	担当者の代替電話番号を戻します。	128
SecondPhoneIsNull()	P/B	代替電話が設定されているかどうかを示します。	134
Substitute()	P/C	担当者の代行者を戻します。	128
SubstituteIsNull()	P/B	代行者が設定されているかどうかを示します。	134
SystemName()	P/C	担当者のホーム・システムを戻します。	128
UserID()	P/C	担当者のユーザー識別子を戻します。	128

アクション関数 / メソッドについては、141ページの『アクション関数 / メソッド』を参照してください。

アクション・メソッド	説明	ページ
Refresh()	サーバーから、完全な担当者情報を検索します。	561
SetAbsence()	ログオン・ユーザーの不在フラグを、指定した値に設定します。	563
SetSubstitute()	ログオン・ユーザーの代行者を、指定した値に設定します。	565

ポイント

ポイント・オブジェクトは、制御コネクターの屈曲点を表します。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	ポイント・オブジェクトを構成します。	86
Copy()	コピーによって、ポイント・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	ポイント・オブジェクトの記憶域を割り振り解除します。	91
destructor()	ポイント・オブジェクトを破棄します。	91
Equal()	2 つのエラー・オブジェクトを、その内容に基づいて比較します。	89
IsEmpty()	使用できるポイント情報がないかどうかを示します。	92
operator=()	ポイント・オブジェクトをこれに代入します。	88
operator==()	2 つのエラー・オブジェクトを、その内容に基づいて比較します。	89

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
XPosition()	I	ポイントの X 座標を戻します。	127
YPosition()	I	ポイントの Y 座標を戻します。	127

ポイント配列

ポイント配列は、ActiveX におけるポイントの照会結果を表します。

配列アクセス・メソッドについては、34ページの『ActiveX の配列』を参照してください。

アクセス・メソッド	説明	ページ
GetAt()	指定された位置の要素を戻します。	36
GetSize()	配列内の要素の数を戻します。	36

ポイント・ベクトル

ポイント・ベクトルは、C 言語におけるポイントの照会結果を表します。

ベクトル・アクセス関数については、29ページの『C 言語のベクトル』を参照してください。

アクセス・メソッド	説明
Deallocate()	ポイント・ベクトル・オブジェクトを割り振り解除します。
FirstElement()	ポイント・ベクトルの第 1 要素を戻します。
NextElement()	ポイント・ベクトルの次の要素を戻します。
Size()	ポイント・ベクトル内の要素の数を戻します。

プロセス・インスタンス

プロセス・インスタンス・オブジェクトは、ワークフロー・プロセス・テンプレートのインスタンスを表します。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	プロセス・インスタンス・オブジェクトを構成します。	86
Copy()	コピーによって、プロセス・インスタンス・オブジェクトの記憶域を割り振って初期化します。	90

基本メソッド	説明	ページ
Deallocate()	プロセス・インスタンス・オブジェクトの記憶域を割り振り解除します。	91
destructor()	プロセス・インスタンス・オブジェクトを破棄します。	91
Equal()	2 つのプロセス・インスタンスを比較します。	89
IsComplete()	完全なプロセス・インスタンス情報を入手できるかどうかを示します。	91
IsEmpty()	使用できるプロセス・インスタンス情報がないかどうかを示します。	92
operator=()	プロセス・インスタンスをこれに代入します。	88
operator==()	2 つのプロセス・インスタンスを比較します。	89

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。

注: 「**設定**」欄の値は、該当属性が 1 次属性 (P) であってプロセス・インスタンスが照会されるとただちに設定されるのか、それともこの属性が 2 次属性 (S) であって個々のプロセス・インスタンスのリフレッシュの後にしか設定されないのかを示しています。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	設定 / 型	説明	ページ
AuditMode()	S/E	プロセス・インスタンスの監査モードを戻します。	102, 104
Category()	P/C	プロセス・インスタンスのカテゴリーを戻します。	128
CategoryIsNull()	P/B	カテゴリーが設定されているかどうかを示します。	134
CreationTime()	S/D	プロセス・インスタンスの作成時刻を戻します。	100
Creator()	S/C	プロセス・インスタンスの作成者を戻します。	128

アクセス・メソッド	設定 / 型	説明	ページ
Description()	P/C	プロセス・インスタンスの記述を戻します。	128
DescriptionIsNull()	P/B	記述が設定されているかどうかを示します。	134
Documentation()	S/C	プロセス・インスタンスの文書を戻します。	128
DocumentationIsNull()	S/B	文書が設定されているかどうかを示します。	134
EndTime()	S/D	プロセス・インスタンスの終了時刻を戻します。	100
EndTimeIsNull()	S/B	終了時刻がセットされているかどうかを示します。	134
Icon()	P/C	プロセス・インスタンスに関連付けられているアイコンを戻します。	128
InContainerName()	S/C	プロセス・インスタンスの入力コンテナーの名前を戻します。	128

アクセス・メソッド	設定 / 型	説明	ページ
InContainerNeeded()	P/B	プロセス・インスタンスを開始するのに入力コンテナーが必要かどうかを示します。入力コンテナーが必要なのは次のような場合です。 • 他の特定のコンテナーに対するマッピングがある。 • スタッフの割り当てデータがそこから取られる。 • 通知関連のデータがそこから取られる。 • transitionまたは終了の条件がコンテナー要素を参照している。 • 記述がコンテナー要素を参照している。 • *プロセス始動時のデータに関するプロンプト* がプロセス・モデル用に設定されている。	99
LastModificationTime()	P/D	プロセス・インスタンスの 1 次属性が最後に変更された時刻を戻します。	100
LastStateChangeTime()	P/D	プロセス・インスタンスの状態が最後に変更された時刻を戻します。	100
Name()	P/C	プロセス・インスタンスの名前を戻します。	128
NotificationTime()	S/D	プロセス・インスタンスの通知時刻を戻します。	100
NotificationTimeIsNull()	S/B	通知時刻が設定されているかどうかを示します。	134
NotifiedPerson()	S/C	通知を受け取った担当者を戻します。	128
NotifiedPersonIsNull()	S/B	通知を受ける担当者が設定されているかどうかを示します。	134

アクセス・メソッド	設定 / 型	説明	ページ
OrganizationName()	S/C	プロセス・インスタンスの組織の名前を戻します。	128
OrganizationNameIsNull()	S/B	組織名が設定されているかどうかを示します。	134
OutContainerName()	S/C	プロセス・インスタンスの出力コンテナーの名前を戻します。	128
ParentName()	P/C	このプロセス・インスタンスの親プロセス・インスタンスの名前を戻します。	128
ParentNameIsNull()	P/B	親名が設定されているかどうかを示します。	134
PersistentOid()	P/C	プロセス・インスタンスのオブジェクト識別子の表現を戻します。	128
ProcessAdmin()	S/C	プロセス・インスタンスのプロセス管理者のユーザー ID を戻します。	128
ProcessAdminIsNull()	S/B	プロセス管理者が設定されているかどうかを示します。	134
ProcessTemplateName()	P/C	プロセス・インスタンスの派生元のプロセス・テンプレートの名前を戻します。	128
RoleName()	S/C	プロセス・インスタンスの役割の名前を戻します。	128
RoleNameIsNull()	S/B	役割が設定されているかどうかを示します。	134
Starter()	S/C	プロセス・インスタンスの開始者を戻します。	128
StarterIsNull()	S/B	開始者が設定されているかどうかを示します。	134
StartTime()	S/D	プロセス・インスタンスの開始時刻を戻します。	100
StartTimeIsNull()	S/B	開始時刻が設定されているかどうかを示します。	134

アクセス・メソッド	設定 / 型	説明	ページ
State()	P/E	プロセス・インスタンスの状態を戻します。	102, 125
StateOfNotification()	S/E	プロセス・インスタンスの通知状態を戻します。	102, 124
SuspensionExpirationTime()	P/D	プロセス・インスタンスの中断満了時刻を戻します。	100
SuspensionExpirationTimeIsNull()	P/B	中断満了時刻が設定されているかどうかを示します。	134
SuspensionTime()	P/D	プロセス・インスタンスが一時中断された時刻を戻します。	100
SuspensionTimeIsNull()	P/B	中断時間が設定されているかどうかを示します。	134
SystemGroupName()	P/C	プロセス・インスタンスが実行されているシステム・グループの名前を戻します。	128
SystemName()	P/C	プロセス・インスタンスが実行されているシステムの名前を戻します。	128
TopLevelName()	P/C	このプロセス・インスタンスの最上位レベルのプロセス・インスタンスの名前を戻します。	128

アクション関数 / メソッドについては、141ページの『アクション関数 / メソッド』を参照してください。

アクション・メソッド	説明	ページ
Delete()	プロセス・インスタンスを削除します。	570
InContainer()	プロセス・インスタンスの入力コンテナーを取り出します。	572
ObtainMonitor()	プロセス・インスタンスのプロセス・インスタンス・モニターを取り出します。	575
OutContainer()	プロセス・インスタンスの出力コンテナーを取り出します。	578

アクション・メソッド	説明	ページ
PersistentObject()	渡されたオブジェクト識別子で指定されるプロセス・インスタンスを取り出します。	580
Refresh()	プロセス・インスタンスの完全情報を取り出します。	583
Restart()	プロセス・インスタンスを再始動します。	586
Resume()	一時中断状態のプロセス・インスタンスの実行を再開します。	588
SetDescription()	プロセス・インスタンスの記述を設定します。	591
SetName()	プロセス・インスタンスの名前を設定します。	593
Start()	プロセス・インスタンスを開始します。	596
Start2()	Java でプロセス・インスタンスを開始し、入力コンテナーを提供します。	596
Suspend()	プロセス・インスタンスを一時中断します。	599
Suspend2()	指定されたカレンダー日付まで、Java においてプロセス・インスタンスを一時中断します。	599
SuspendUntil()	指定した時刻までプロセス・インスタンスを一時中断します。	599
Terminate()	プロセス・インスタンスを中止します。	602

プロセス・インスタンス・リスト

プロセス・インスタンス・リストは、プロセス・インスタンスのグループを表します。**永続リストのすべての関数 / メソッドは、プロセス・インスタンス・リストに対しても使用できます。**

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	プロセス・インスタンス・リスト・オブジェクトを構成します。	86

基本メソッド	説明	ページ
Copy()	コピーによって、プロセス・インスタンス・リスト・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	プロセス・インスタンス・リスト・オブジェクトの記憶域を割り振り解除します。	91
destructor()	プロセス・インスタンス・リスト・オブジェクトを破棄します。	91
Equal()	2 つのプロセス・インスタンス・リストを比較します。	89
operator=()	プロセス・インスタンス・リストをこれに代入します。	88
operator==()	2 つのプロセス・インスタンス・リストを比較します。	89

アクション関数 / メソッドについては、141ページの『アクション関数 / メソッド』を参照してください。

アクション・メソッド	説明	ページ
QueryProcessInstances()	プロセス・インスタンス・リストによって含められるプロセス・インスタンスを検索します。	608

プロセス・インスタンス・リスト配列

プロセス・インスタンス・リスト配列は、ActiveX におけるプロセス・インスタンス・リストの照会結果を表します。

配列アクセス・メソッドについては、34ページの『ActiveX の配列』を参照してください。

アクセス・メソッド	説明	ページ
GetAt()	指定された位置の要素を戻します。	36
GetSize()	配列内の要素の数を戻します。	36

イベント	説明	ページ
NewProcessInstanceList()	配列に新しいプロセス・インスタンス・リストを追加します。	37
ProcessInstanceListRemove()	プロセス・インスタンス・リストを配列から除去します。	38

プロセス・インスタンス・リスト・ベクトル

プロセス・インスタンス・リスト・ベクトルは、C 言語におけるプロセス・インスタンス・リストの照会結果を表します。

ベクトル・アクセス関数については、29ページの『C 言語のベクトル』を参照してください。

アクセス・メソッド	説明
Deallocate()	プロセス・インスタンス・リスト・ベクトル・オブジェクトを割り振り解除します。
FirstElement()	プロセス・インスタンス・リスト・ベクトルの第 1 要素を戻します。
NextElement()	プロセス・インスタンス・リスト・ベクトルの次の要素を戻します。
Size()	プロセス・インスタンス・リスト・ベクトル内の要素の数を戻します。

プロセス・インスタンス・モニター

プロセス・インスタンス・モニター・オブジェクトは、プロセス・インスタンスのモニターを表します。 **FmcjBlockInstanceMonitor のすべての関数 / メソッドは、プロセス・インスタンス・モニターに対しても使用できます。**

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
Deallocate()	プロセス・インスタンス・モニター・オブジェクトの記憶域を割り振り解除します。含まれているすべてのブロック・インスタンス・モニターも割り振り解除されます。	91
destructor()	プロセス・インスタンス・モニター・オブジェクトを破棄します。含まれているすべてのブロック・インスタンス・モニターも破棄されます。	91

プロセス・インスタンス通知

プロセス・インスタンス通知は、プロセス・インスタンスについて出される通知を表します。 **FmcjItem のすべての関数 / メソッドは、プロセス・インスタンス通知に対しても使用できます。**

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	プロセス・インスタンス通知オブジェクトを構成します。	86
Copy()	コピーによって、プロセス・インスタンス通知オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	プロセス・インスタンス通知オブジェクトの記憶域を割り振り解除します。	91
destructor()	プロセス・インスタンス通知オブジェクトを破棄します。	91
Kind()	C++ 言語において、オブジェクトがプロセス・インスタンス通知であることを示します。	93, 122
operator=()	プロセス・インスタンス通知をこれに代入します。	88
operator==()	2 つのプロセス・インスタンス通知を比較します。	89

アクション関数 / メソッドについては、141ページの『アクション関数 / メソッド』を参照してください。

アクション・メソッド	説明	ページ
ObtainInstanceMonitor()	ActiveX において関連するプロセス・インスタンスのインスタンス・モニターを戻します。	524
PersistentObject()	指定されたプロセス・インスタンス通知を取り出します。	613

プロセス・インスタンス通知配列

プロセス・インスタンス通知配列は、ActiveX におけるプロセス・インスタンス通知の照会結果を表します。

配列アクセス・メソッドについては、34ページの『ActiveX の配列』を参照してください。

アクセス・メソッド	説明	ページ
GetAt()	指定された位置の要素を戻します。	36
GetSize()	配列内の要素の数を戻します。	36

プロセス・インスタンス通知ベクトル

プロセス・インスタンス通知ベクトルは、C 言語におけるプロセス・インスタンス通知の照会結果を表します。

ベクトル・アクセス関数については、29ページの『C 言語のベクトル』を参照してください。

アクセス・メソッド	説明
Deallocate()	プロセス・インスタンス通知ベクトル・オブジェクトを割り振り解除します。
FirstElement()	プロセス・インスタンス通知ベクトルの第 1 要素を戻します。
NextElement()	プロセス・インスタンス通知ベクトルの次の要素を戻します。
Size()	プロセス・インスタンス通知ベクトル内の要素の数を戻します。

プロセス・インスタンス・ベクトル

プロセス・インスタンス・ベクトルは、C 言語におけるプロセス・インスタンスの照会結果を表します。

ベクトル・アクセス関数については、29ページの『C 言語のベクトル』を参照してください。

アクセス・メソッド	説明
Deallocate()	プロセス・インスタンス・オブジェクトの記憶域を割り振り解除します。
FirstElement()	プロセス・インスタンス・ベクトルの第 1 要素を戻します。
NextElement()	プロセス・インスタンス・ベクトルの次の要素を戻します。
Size()	プロセス・インスタンス・ベクトル内の要素の数を戻します。

プロセス・テンプレート

プロセス・テンプレート・オブジェクトは、定義機能ワークフロー・プロセス・モデルに相当する実行機能を表します。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	プロセス・テンプレート・オブジェクトを構成します。	86
Copy()	コピーによって、プロセス・テンプレート・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	プロセス・テンプレート・オブジェクトの記憶域を割り振り解除します。	91
destructor()	プロセス・テンプレート・オブジェクトを破棄します。	91
Equal()	2 つのプロセス・テンプレートを比較します。	89
IsComplete()	完全なプロセス・テンプレート情報を入手できるかどうかを示します。	91
IsEmpty()	使用できるプロセス・テンプレート情報がないかどうかを示します。	92
operator=()	プロセス・テンプレートをこれに代入します。	88
operator==()	2 つのプロセス・テンプレートを比較します。	89

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。

注: 「**設定**」欄の値は、該当属性が 1 次属性 (P) であってプロセス・テンプレートが照会されるとただちに設定されるのか、それともこの属性が 2 次属性 (S) であって個々のプロセス・テンプレートのリフレッシュの後にしか設定されないのかを示しています。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	設定 / 型	説明	ページ
AuditMode()	S/E	プロセス・テンプレートの監査モードを戻します。	102, 104
Category()	P/C	プロセス・テンプレートのカテゴリーを戻します。	128

アクセス・メソッド	設定 / 型	説明	ページ
CategoryIsNull()	P/B	カテゴリーが設定されているかどうかを示します。	134
CreationTime()	P/D	プロセス・テンプレートの作成時刻を戻します。	100
Description()	P/C	プロセス・テンプレートの記述を戻します。	128
DescriptionIsNull()	P/B	記述が設定されているかどうかを示します。	134
Documentation()	S/C	プロセス・テンプレートの文書を戻します。	128
DocumentationIsNull()	S/B	文書が設定されているかどうかを示します。	134
Icon()	P/C	プロセス・テンプレートに関連付けられているアイコンを戻します。	128
InContainerName()	S/C	プロセス・テンプレートの入力コンテナーの名前を戻します。	128
InContainerNeeded()	P/B	プロセス・テンプレートのインスタンスを開始するのに入力コンテナーが必要かどうかを示します。入力コンテナーが必要なのは次のような場合です。 • 他の特定のコンテナーに対するマッピングがある。 • スタッフの割り当てデータがそこから取られる。 • 通知関連のデータがそこから取られる。 • トランザクションまたは終了の条件がコンテナー要素を参照している。 • 記述がコンテナー要素を参照している。 • *プロセス始動時のデータに関するプロンプト* がプロセス・モデル用に設定されている。	99
LastModificationTime()	P/D	プロセス・テンプレートの 1 次属性が最後に変更された時刻を戻します。	100

アクセス・メソッド	設定 / 型	説明	ページ
Name()	P/C	プロセス・テンプレートの名前を戻します。	128
OrganizationName()	S/C	プロセス・テンプレートの組織の名前を戻します。	128
OrganizationNameIsNull()	S/B	組織名が設定されているかどうかを示します。	134
OutContainerName()	S/C	プロセス・テンプレートの出力コンテナーの名前を戻します。	128
PersistentOid()	P/C	プロセス・テンプレートのオブジェクト識別子の表現を戻します。	128
ProcessAdmin()	S/C	プロセス・テンプレートのインスタンスのプロセス管理者のユーザー ID を戻します。	128
ProcessAdminIsNull()	S/B	プロセス管理者が設定されているかどうかを示します。	134
RoleName()	S/C	プロセス・テンプレートの役割の名前を戻します。	128
RoleNameIsNull()	S/B	役割が設定されているかどうかを示します。	134
ValidFromTime()	P/D	プロセス・テンプレートが有効になる時刻を戻します。	100

アクション関数 / メソッドについては、141ページの『アクション関数 / メソッド』を参照してください。

アクション・メソッド	説明	ページ
CreateAndStartInstance()	プロセス・テンプレートのインスタンスを作成して開始します。	617
CreateAndStartInstance2()	Java でプロセス・テンプレートを作成し、入力コンテナーを提供します。	617
CreateInstance()	プロセス・テンプレートのインスタンスを作成します。	625
Delete()	指定されたプロセス・テンプレートを削除します。	629
Delete2()	指定されたプロセス・テンプレート・バージョンを Java で削除します。	629

アクション・メソッド	説明	ページ
ExecuteProcessInstance()	指定されたプロセス・テンプレートからインスタンスを作成また実行します。	632
ExecuteProcessInstanceAsync()	指定されたプロセス・テンプレートからインスタンスを作成また実行します (回答を待たない)。	632
InitialInContainer()	プロセス・テンプレートの初期定義入力コンテナーを取り出します。	642
PersistentObject()	渡されたオブジェクト識別子で指定されるプロセス・テンプレートを取り出します。	645
ProgramTemplate()	渡された名前で指定されているプログラム・テンプレートを取り出します。	647
Refresh()	プロセス・テンプレートの完全情報を取り出します。	651

プロセス・テンプレート・リスト

プロセス・テンプレート・リストは、プロセス・テンプレートのグループを表します。**永続リストのすべての関数 / メソッドは、プロセス・テンプレート・リストに対しても使用できます。**

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	プロセス・テンプレート・リスト・オブジェクトを構成します。	86
Copy()	コピーによって、プロセス・テンプレート・リスト・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	プロセス・テンプレート・リスト・オブジェクトの記憶域を割り振り解除します。	91
Equal()	2 つのプロセス・テンプレート・リストを比較します。	89
destructor()	プロセス・テンプレート・リスト・オブジェクトを破棄します。	91
operator=()	プロセス・テンプレート・リストをこれに代入します。	88
operator==()	2 つのプロセス・テンプレート・リストを比較します。	89

アクション関数 / メソッドについては、141ページの『アクション関数 / メソッド』を参照してください。

アクション・メソッド	説明	ページ
QueryProcessTemplates()	プロセス・テンプレート・リストによって含められるプロセス・テンプレートを検索します。	656

プロセス・テンプレート・リスト配列

プロセス・テンプレート・リスト配列は、ActiveX におけるプロセス・テンプレート・リストの照会結果を表します。

配列アクセス・メソッドについては、34ページの『ActiveX の配列』を参照してください。

アクセス・メソッド	説明	ページ
GetAt()	指定された位置の要素を戻します。	36
GetSize()	配列内の要素の数を戻します。	36

イベント	説明	ページ
NewProcessTemplateList()	配列に新しいプロセス・テンプレート・リストを追加します。	37
ProcessTemplateListRemove()	プロセス・テンプレート・リストを配列から除去します。	38

プロセス・テンプレート・リスト・ベクトル

プロセス・テンプレート・リスト・ベクトルは、C 言語におけるプロセス・テンプレート・リストの照会結果を表します。

ベクトル・アクセス関数については、29ページの『C 言語のベクトル』を参照してください。

アクセス・メソッド	説明
Deallocate()	プロセス・テンプレート・リスト・ベクトル・オブジェクトを割り振り解除します。
FirstElement()	プロセス・テンプレート・リスト・ベクトルの第 1 要素を戻します。

アクセス・メソッド	説明
NextElement()	プロセス・テンプレート・リスト・ベクトルの次の要素を戻します。
Size()	プロセス・テンプレート・リスト・ベクトル内の要素の数を戻します。

プロセス・テンプレート・ベクトル

プロセス・テンプレート・ベクトルは、C 言語におけるプロセス・テンプレートの照会結果を表します。

ベクトル・アクセス関数については、29ページの『C 言語のベクトル』を参照してください。

アクセス・メソッド	説明
Deallocate()	プロセス・テンプレート・ベクトル・オブジェクトの記憶域を割り振り解除します。
FirstElement()	プロセス・テンプレート・ベクトルの第 1 要素を戻します。
NextElement()	プロセス・テンプレート・ベクトルの次の要素を戻します。
Size()	プロセス・テンプレート・ベクトル内の要素の数を戻します。

プログラム・データ

プログラム・データ・オブジェクトは、プログラム・インプリメンテーション定義を表します。 C++ においてこれは、プログラム・テンプレートから private として継承されます。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	プログラム・データ・オブジェクトを構成します。	86
Copy()	コピーによって、プログラム・データ・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	プログラム・データ・オブジェクトの記憶域を割り振り解除します。	91
destructor()	プログラム・データ・オブジェクトを破棄します。	91

基本メソッド	説明	ページ
Equal()	2 つのプログラム・データ・オブジェクトを比較して、それらが同じ作業項目に属するものかどうかを調べます。	89
IsEmpty()	使用できるプログラム・データ情報がないかどうかを示します。	92
operator=()	プログラム・データ・オブジェクトをこれに代入します。	88
operator==()	2 つのプログラム・データ・オブジェクトを比較して、それらが同じ作業項目に属するものかどうかを調べます。	89

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
Description()	C	インプリメント・プログラムの記述を戻します。	128
DescriptionIsNull()	B	記述が設定されているかどうかを示します。	134
ExecutionMode()	E	プログラムがグローバル・トランザクションに参加できるかどうかを示します。	102, 114
ExecutionUser()	E	プログラムの実行を代行する対象となるユーザーを戻します。	102, 114
Icon()	C	インプリメント・プログラムに関連付けられているアイコンを戻します。	128
Implementations()	M	プログラムのインプリメンテーション定義を戻します。	130
InContainer()	P	プログラムの入力コンテナーを戻します。	132
IsUnattended()	B	プログラムを無人で実行できるかどうかを示します。	99
OutContainer()	P	プログラムの出力コンテナーを戻します。	132

アクセス・メソッド	タイプ	説明	ページ
ProgramTrusted()	B	プログラムが信頼できるもの (トラステッド) かどうかを示します。トラステッド (信頼される) プログラムだけがそのプログラム ID を受信できます。	99

プログラム・テンプレート

プログラム・テンプレート・オブジェクトは、プログラム・インプリメンテーション定義を表します。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	プログラム・テンプレート・オブジェクトを構成します。	86
Copy()	コピーによって、プログラム・テンプレート・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	プログラム・テンプレート・オブジェクトの記憶域を割り振り解除します。	91
destructor()	プログラム・テンプレート・オブジェクトを破棄します。	91
Equal()	2 つのプログラム・テンプレート・オブジェクトを、その名前とそれぞれが属するプロセス・テンプレートに基づいて比較します。	89
IsEmpty()	使用できるプログラム・テンプレート情報がないかどうかを示します。	92
operator=()	プログラム・テンプレート・オブジェクトをこれに代入します。	88
operator==()	2 つのプログラム・テンプレート・オブジェクトを、その名前とそれぞれが属するプロセス・テンプレートに基づいて比較します。	89

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般

形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
Description()	C	インプリメント・プログラムの記述を戻します。	128
DescriptionIsNull()	B	記述が設定されているかどうかを示します。	134
ExecutionMode()	E	プログラムがグローバル・トランザクションに参加できるかどうかを示します。	102, 114
ExecutionUser()	E	プログラムの実行を代行する対象となるユーザーを戻します。	128, 114
Icon()	C	インプリメント・プログラムに関連付けられているアイコンを戻します。	128
Implementations()	M	プログラムのインプリメンテーション定義を戻します。	130
InitialInContainer()	P	プログラムの初期定義入力コンテナーを戻します。	132
InContainerAccess()	B	入力コンテナーがプログラムからアクセスされるかどうかを示します。	99
IsUnattended()	B	プログラムを無人で実行できるかどうかを示します。	99
InitialOutContainer()	P	プログラムの初期定義出力コンテナーを戻します。	132
OutContainerAccess()	B	出力コンテナーがプログラムからアクセスされるかどうかを示します。	99
ProgramTrusted()	B	プログラムが信頼できるもの (トラステッド) かどうかを示します。トラステッド (信頼される) プログラムだけがそのプログラム ID を受信できます。	99
StructuresFromActivity()	B	渡された任意のコンテナーをプログラムで処理できるかどうかを示します。	99
ValidFromTime()	P/D	プロセス・テンプレートが有効になり、それによってプログラム・テンプレートが有効になる時刻を戻します。	100

アクション関数 / メソッドについては、141ページの『アクション関数 / メソッド』を参照してください。

アクション・メソッド	説明	ページ
Execute()	システムのプログラム実行サーバーによるプログラムの実行を要求します。	659
Execute2()	システムのプログラム実行サーバーによるプログラムの実行を要求します。	659
ExecuteAsync()	システムのプログラム実行サーバーによるプログラムの実行を要求します。応答を待ちません。	659
ExecuteAsync2()	システムのプログラム実行サーバーによるプログラムの実行を要求します。応答を待ちません。	659
ExecuteAsyncWithOptions()	システムのプログラム実行サーバーによるプログラムの実行を要求します。応答を待ちません。プログラムの優先順位を指定できます。	659
ExecuteWithOptions()	システムのプログラム実行サーバーによるプログラムの実行を要求します。プログラムの優先順位を指定できます。	659

ReadOnly コンテナー

読み取り専用コンテナーは、作業項目の入力データ・コンテナーを表します。**コンテナーのすべての関数 / メソッドは、読み取り専用コンテナーに使用できます。**

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	読み取り専用コンテナー・オブジェクトを構成します。	86
Copy()	コピーによって、読み取り専用コンテナー・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	読み取り専用コンテナー・オブジェクトの記憶域を割り振り解除します。	91
Equal()	2 つの読み取り専用コンテナーを比較します。	89
destructor()	読み取り専用コンテナー・オブジェクトを破棄します。	91
operator=()	読み取り専用コンテナーをこれに代入します。	88
operator==()	2 つの読み取り専用コンテナーを比較します。	89

ReadWrite コンテナー

読み取り / 書き込みコンテナーは、プロセス・インスタンスの入力コンテナーまたは作業項目の出力コンテナーを表します。**コンテナーのすべての関数 / メソッドは、読み取り / 書き込みコンテナーに使用できます。**

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	読み取り / 書き込みコンテナー・オブジェクトを構成します。	86
Copy()	コピーによって、読み取り / 書き込みコンテナー・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	読み取り / 書き込みコンテナー・オブジェクトの記憶域を割り振り解除します。	91
Equal()	2 つの読み取り / 書き込みコンテナーを比較します。	89
destructor()	読み取り / 書き込みコンテナー・オブジェクトを破棄します。	91
operator=()	読み取り / 書き込みコンテナーを別のものに代入します。	88
operator==()	2 つの読み取り / 書き込みコンテナーを比較します。	89

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。

型列の値は、設定されるプロパティーの型を表しています。バイナリー型 (N)、文字列型 (C)、浮動小数点型 (F)、または整数型 (I) のいずれかです。関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
SetArrayBinaryValue()	N	指定されたコンテナー・リーフ要素の値を設定します (C 言語)。リーフ要素は配列の一部であり、BINARY 型です。	69
SetArrayFloatValue()	F	指定されたコンテナー・リーフ要素の値を設定します (C 言語)。リーフ要素は配列の一部であり、FLOAT 型です。	69

アクセス・メソッド	タイプ	説明	ページ
SetArrayLongValue()	I	指定されたコンテナー・リーフ要素の値を設定します (C 言語)。リーフ要素は配列の一部であり、LONG 型です。	69
SetArrayStringValue()	C	指定されたコンテナー・リーフ要素の値を設定します (C 言語)。リーフ要素は配列の一部であり、STRING 型です。	70
SetBinaryValue()	N	指定されたコンテナー・リーフ要素の値を設定します (C 言語)。リーフ要素は BINARY 型です。	69
SetBuffer()	N	指定されたコンテナー・リーフ要素の値を設定します (Java)。リーフ要素は BINARY 型です。	71
SetBuffer2()	N	指定されたコンテナー・リーフ要素の値を設定します (Java)。リーフ要素は配列の一部であり、BINARY 型です。	71
SetDouble()	F	指定されたコンテナー・リーフ要素の値を設定します (Java)。リーフ要素は FLOAT 型です。	71
SetDouble2()	F	指定されたコンテナー・リーフ要素の値を設定します (Java)。リーフ要素は配列の一部であり、FLOAT 型です。	71
SetFloatValue()	F	指定されたコンテナー・リーフ要素の値を設定します (C 言語)。リーフ要素は FLOAT 型です。	69
SetLong()	I	指定されたコンテナー・リーフ要素の値を設定します (Java)。リーフ要素は LONG 型です。	71
SetLong2()	I	指定されたコンテナー・リーフ要素の値を設定します (Java)。リーフ要素は配列の一部であり、LONG 型です。	71
SetLongValue()	I	指定されたコンテナー・リーフ要素の値を設定します (C 言語)。リーフ要素は LONG 型です。	69
SetString()	N	指定されたコンテナー・リーフ要素の値を設定します (Java)。リーフ要素は STRING 型です。	72

アクセス・メソッド	タイプ	説明	ページ
SetString2()	N	指定されたコンテナー・リーフ要素の値を設定します (Java)。リーフ要素は配列の一部であり、STRING 型です。	72
SetStringValue()	C	指定されたコンテナー・リーフ要素の値を設定します (C 言語)。リーフ要素は STRING 型です。	70
SetValue()	N/F/C/I	指定されたコンテナー・リーフ要素の値を設定します (C++ 言語)。	70
SetValueDbl()	F	指定されたコンテナー・リーフ要素の値を設定します (ActiveX)。リーフ要素は FLOAT 型です。	68
SetValueLng()	F	指定されたコンテナー・リーフ要素の値を設定します (ActiveX)。リーフ要素は LONG 型です。	68
SetValueStr()	F	指定されたコンテナー・リーフ要素の値を設定します (ActiveX)。リーフ要素は STRING 型です。	68

結果オブジェクト

結果オブジェクトは、C++ および C 言語での関数 / メソッド呼び出しの結果を表します。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
destructor	結果オブジェクトの C++ 表現を破棄します。	91

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。結果オブジェクトはクライアントのみのサポート・オブジェクトを表すものなので、1 次属性と 2 次属性の区別は適用されません。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
MessageText()	C	NLS 形式化メッセージとして結果を戻します。	128
ObjectOfCurrentThread()	P	この関数 / メソッドの呼び出し元のスレッドに関連した結果オブジェクトを戻します。	
Origin()	C	結果の起源、つまりファイル、行、関数を戻します。	128
Parameters()	M	結果のパラメーターを戻します。そのパラメーターはすでにメッセージ・テキストに組み込まれています。	130
Rc()	I	結果オブジェクトに保管されている戻りコードを戻します。	127

Service

サービス・オブジェクトは、MQSeries Workflow サービス・オブジェクトに共通のいくつかの局面を表します。**サービスのすべての関数 / メソッドは、実行サービスにも適用されます。**

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。サービス・オブジェクトはクライアントのみのサポート・オブジェクトを表すものなので、1 次属性と 2 次属性の区別は適用されません。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
IsLoggedOn()	B	ユーザーがこのサービス・オブジェクトを介してログオンしているかどうかを示します。	99
SetTimeout()	I	サーバーが応答するまでクライアントが待つ時間を設定します。時間はミリ秒で指定します。	135
SystemGroupName()	C	サーバーの属するシステム・グループの名前を戻します。	128

アクセス・メソッド	タイプ	説明	ページ
SystemName()	C	サーバーの属するシステムの名前を戻します。	128
Timeout()	I	サーバーが応答するまでクライアントが待つ時間を戻します。	127
UserID()	C	ログオン・ユーザーのユーザー識別子を戻します。	128

アクション関数 / メソッドについては、141ページの『アクション関数 / メソッド』を参照してください。

アクション・メソッド	説明	ページ
Refresh()	サーバーの情報 (特にログオン状況) をリフレッシュします。	665
SetPassword()	ログオン・ユーザーのパスワードを設定します。	667
UserSettings()	ログオン・ユーザーのユーザー設定値を取り出します。	670

文字列配列

文字列配列は、ActiveX における文字列のリストを表します。

配列アクセス・メソッドについては、34ページの『ActiveX の配列』を参照してください。

アクセス・メソッド	説明	ページ
Add()	配列に要素を追加します。	35
GetAt()	指定された位置の要素を戻します。	36
GetSize()	配列内の要素の数を戻します。	36
RemoveAll()	すべての要素を削除します。	36
RemoveAt()	指定された位置の要素を削除します。	37
SetAt()	要素を指定した位置に設定します。	37

文字列ベクトル

C 言語において、文字列ベクトルは一連の文字列情報を表すのに使われます。たとえば、ログオン・ユーザーに許可されているカテゴリーを示す文字列ベクトルが戻されます。あるいは、代行する担当者を指定するには文字列ベクトルを使用しなければなりません。

ベクトル・アクセス関数については、29ページの『C 言語のベクトル』を参照してください。

アクセス・メソッド	説明
AddElement()	文字列ベクトルに文字列を追加します。
Allocate()	文字列ベクトルの記憶域を割り振ります。
Deallocate()	文字列ベクトルの記憶域を割り振り解除します。
FirstElement()	文字列ベクトルの第 1 要素を戻します。
FirstResultParmElement()	結果オブジェクトのパラメーターを表す文字列ベクトルの第 1 要素を戻します。この関数を呼び出しても結果オブジェクトが変更されることはないので、整合性のある読み取りができます。
NextElement()	文字列ベクトルの次の要素を戻します。
NextResultParmElement()	結果オブジェクトのパラメーターを表す文字列ベクトルの次の要素を戻します。この関数を呼び出しても結果オブジェクトが変更されることはないので、整合性のある読み取りができます。
RemoveElement()	文字列ベクトルから文字列を削除します。
ResultParmDeallocate()	結果オブジェクトのパラメーターを表す文字列ベクトルの記憶域を割り振り解除します。この関数を呼び出しても結果オブジェクトが変更されることはないので、整合性のある読み取りができます。
ResultParmSize()	結果オブジェクトのパラメーターを表す文字列ベクトル中の要素の数を戻します。この関数を呼び出しても結果オブジェクトが変更されることはないので、整合性のある読み取りができます。
Size()	文字列ベクトル内の要素の数を戻します。

記号レイアウト

記号レイアウト・オブジェクトは、名前付きアイコンのグラフィカル情報を表します。

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	記号レイアウト・オブジェクトを構成します。	86
Copy()	コピーによって、記号レイアウト・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	記号レイアウト・オブジェクトの記憶域を割り振り解除します。	91
destructor()	記号レイアウト・オブジェクトを破棄します。	91
Equal()	2 つの記号レイアウト・オブジェクトを、その内容に基づいて比較します。	89
IsEmpty()	使用できる記号レイアウト情報がないかどうかを示します。	92
operator=()	記号レイアウト・オブジェクトをこれに代入します。	88
operator==()	2 つの記号レイアウト・オブジェクトを、その内容に基づいて比較します。	89

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
XPosition()	I	名前付きアイコンの X 座標を戻します。	127
XPositionOfName()	I	アイコンに関連した名前の X 座標を戻します。	127
YPosition()	I	名前付きアイコンの Y 座標を戻します。	127
YPositionOfName()	I	アイコンに関連した名前の Y 座標を戻します。	127

作業項目

作業項目は、処理のためにユーザーに割り当てられるアクティビティー・インスタンスを表します。**アイテム (Item) のすべての関数 / メソッドは、作業項目に対しても使用できます。**

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	作業項目・オブジェクトを構成します。	86
Copy()	コピーによって、作業項目・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	作業項目・オブジェクトの記憶域を割り振り解除します。	91
destructor()	作業項目・オブジェクトを破棄します。	91
Kind()	C++ 言語において、オブジェクトが作業項目であることを示します。	93, 122
operator=()	作業項目をこれに代入します。	88
operator==()	2 つの作業項目を比較します。	89

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。

注: 「**設定**」欄の値は、該当属性が 1 次属性 (P) であって作業項目が照会されるとただちに設定されるのか、それともこの属性が 2 次属性 (S) であって個々の作業項目のリフレッシュの後にしか設定されないのかを示しています。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	設定 / 型	説明	ページ
ActivityKind()	P/E	関連したアクティビティー・インスタンスの種類 (プログラムまたはプロセスなど) を戻します。	93, 109
ErrorReason()	S/O	関連するアクティビティー・インスタンスが InError 状態である理由を説明したエラー・オブジェクトを戻します。	131
ErrorReasonIsNull()	S/B	エラーの理由が設定されているかどうかを示します。	134
ExitCondition()	S/C	作業項目の終了条件を戻します。	128
FirstNotificationTime()	S/D	作業項目の最初の通知がなされる予定の時刻、またはなされた時刻を戻します。	100
FirstNotificationTimeIsNull()	S/B	最初の通知時刻が設定されているかどうかを示します。	134
Implementation()	P/C	関連するアクティビティー・インスタンスのインプリメント・プログラムの名前を戻します。	128
ImplementationIsNull()	P/B	インプリメンテーションが設定されているかどうかを示します。	134
ManualExitMode()	S/B	作業項目の終了モードが手動かどうかを示します。	99
ManualStartMode()	S/B	作業項目の開始モードが手動かどうかを示します。	99
PersistentOidOfActivityInstance()	P/C	関連するアクティビティー・インスタンスのオブジェクト ID を戻します。	128
Priority()	P/I	作業項目の優先順位を戻します。	127
SecondNotificationTime()	S/D	作業項目の第 2 の通知がなされる予定の時刻、またはなされた時刻を戻します。	100

アクセス・メソッド	設定 / 型	説明	ページ
SecondNotificationTimeIsNull()	S/B	第 2 の通知時刻が設定されているかどうかを示します。	134
Staff()	S/M	関連するアクティビティー・インスタンスに対して作業項目を所有しているすべての担当者を戻します。	130
StartCondition()	S/C	作業項目の開始条件を戻します。	128
State	P/E	作業項目の状態を戻します。	102, 119
StateOfNotification()	S/E	作業項目の通知状態を戻します。	102, 105
SupportTools()	P/M	作業項目に関連したサポート・ツールを戻します。	130
SupportToolsIsNull()	P/B	サポート・ツールが設定されているかどうかを示します。	134

アクション関数 / メソッドについては、141ページの『アクション関数 / メソッド』を参照してください。

アクション・メソッド	説明	ページ
ActivityInstance()	指定されたアクティビティー・インスタンスを取り出します。	676
CancelCheckOut()	作業項目のチェックアウトを取り消します。	679
CheckIn()	作業項目をチェックインします。	681
CheckOut()	作業項目をチェックアウトします。	684
Finish()	手動終了の作業項目を終了します。	690
ForceFinish()	作業項目を強制終了します。	693
ForceRestart()	作業項目を強制再始動します。	696
InContainer()	作業項目の入力コンテナーを取り出します。	698
ObtainInstanceMonitor()	ActiveX において関連するプロセス・インスタンスのインスタンス・モニターを戻します。	524
OutContainer()	作業項目の出力コンテナーを取り出します。	701
PersistentObject()	指定された作業項目を取り出します。	703
Restart()	作業項目を再始動します。	706
Start()	作業項目を開始します。	708

アクション・メソッド	説明	ページ
StartTool()	指定されたサポート・ツールを開始します。	710
Terminate()	作業項目を中止します。	713

作業項目配列

作業項目配列は、ActiveX における作業項目の照会結果を表します。

配列アクセス・メソッドについては、34ページの『ActiveX の配列』を参照してください。

アクセス・メソッド	説明	ページ
GetAt()	指定された位置の要素を戻します。	36
GetSize()	配列内の要素の数を戻します。	36

作業項目・ベクトル

作業項目ベクトルは、C 言語における作業項目の照会結果を表します。

ベクトル・アクセス関数については、29ページの『C 言語のベクトル』を参照してください。

アクセス・メソッド	説明
Deallocate()	作業項目ベクトル・オブジェクトの記憶域を割り振り解除します。
FirstElement()	作業項目ベクトルの第 1 要素を戻します。
NextElement()	作業項目ベクトルの次の要素を戻します。
Size()	作業項目ベクトル内の要素の数を戻します。

ワークリスト

ワークリストは、アイテムのグループを表します。**永続リストのすべての関数 / メソッドは、ワークリストに対しても使用できます。**

基本関数 / メソッドについては、85ページの『基本関数 / メソッド』を参照してください。

基本メソッド	説明	ページ
constructor()	ワークリスト・オブジェクトを構成します。	86

基本メソッド	説明	ページ
Copy()	コピーによって、ワークリスト・オブジェクトの記憶域を割り振って初期化します。	90
Deallocate()	ワークリスト・オブジェクトの記憶域を割り振り解除します。	91
destructor()	ワークリスト・オブジェクトを破棄します。	91
Equal()	2 つのワークリストを比較します。	89
operator=()	ワークリストを別のものに代入します。	88
operator==()	2 つのワークリストを比較します。	89

アクセス関数 / メソッドについては、96ページの『アクセス関数 / メソッド』を参照してください。すべてのプロパティーは 1 次プロパティーです。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	タイプ	説明	ページ
BeepOption()	B	ワークリストの内容が変更された場合にビープ音を発生するかどうかを示します。	99

アクション関数 / メソッドについては、141ページの『アクション関数 / メソッド』を参照してください。

アクション・メソッド	説明	ページ
QueryActivityInstanceNotifications()	ワークリストによって含められるアクティビティー・インスタンス通知を取り出します。	718
QueryItems()	ワークリストによって含められるすべてのアイテムを取り出します。	721
QueryProcessInstanceNotifications()	ワークリストによって含められるプロセス・インスタンス通知を取り出します。	724
QueryWorkitems()	ワークリストによって含められる作業項目を取り出します。	728

ワークリスト配列

ワークリスト配列は、ActiveX におけるワークリストの照会結果を表します。

配列アクセス・メソッドについては、34ページの『ActiveX の配列』を参照してください。

アクセス・メソッド	説明	ページ
GetAt()	指定された位置の要素を戻します。	36
GetSize()	配列内の要素の数を戻します。	36

イベント	説明	ページ
NewWorklist()	配列に新しいワークリストを追加します。	37
WorklistRemove()	ワークリストを配列から除去します。	38

ワークリスト・ベクトル

ワークリスト・ベクトルは、C 言語におけるワークリストの照会結果を表します。

ベクトル・アクセス関数については、29ページの『C 言語のベクトル』を参照してください。

型列の値は、戻されるプロパティーの型を表しています。ブール型 (B)、文字列型 (C)、日付 / 時刻型 (D)、列挙型 (E)、整数型 (I)、または複数値型 (M)、オブジェクト・ポインター (P)、またはオブジェクト本体 (O) のいずれかです。一般形式の関数 / メソッド宣言は、示されているページにあります。

アクセス・メソッド	説明
Deallocate()	ワークリスト・ベクトル・オブジェクトを割り振り解除します。
FirstElement()	ワークリスト・ベクトルの第 1 要素を戻します。
NextElement()	ワークリスト・ベクトルの次の要素を戻します。
Size()	ワークリスト・ベクトル内の要素の数を戻します。

第7部 プログラミング・インターフェース

以下の章では、アクションまたはアクティビティー・インプリメンテーション関数 / メソッド用の MQSeries Workflow アプリケーション・プログラミング・インターフェースについてアルファベット順に説明しています。

各項には、API 関数 / メソッドの機能についての説明があり、その後に次のようなサブセクションが続いています。

使用上の注意 その呼び出しの特性についての一般情報です。

許可 API 呼び出しを実行するのに必要な権限を示します。

必要な接続 セッションを確立する先の MQSeries Workflow サーバーを示します。

API インターフェース宣言
API 関数 / メソッド宣言用に組み込むファイルの名前とパッケージの名前を個別に示します。

ActiveX のシグニチャー
API 呼び出しの ActiveX 構文を示します。

注: ActiveX のシグニチャーはオブジェクト定義言語 (ODL) で提供されています。たとえば、*BSTR* 型は、VisualBasic の型が実際には String であるストリングに使用されます。

C 言語のシグニチャー
API 呼び出しの C 言語構文を示します。

C++ 言語のシグニチャー
API 呼び出しの C++ 言語構文を示します。

Java のシグニチャー
API 呼び出しの Java 構文を示します。

パラメーター それぞれのパラメーターについて説明し、パラメーターが入力または出力のどちらのパラメーターかの標識を示します。

戻り値のデータ型
呼び出しから戻される値について説明します。

戻りコード / FmcException
その呼び出しから発生するすべての戻りコードのリストを示します。

例 呼び出しの例を示します。

第38章 アクティビティー・インスタンスのアクション

FmcjActivityInstance または ActivityInstance オブジェクトは、プロセス・インスタンスのアクティビティーのインスタンスを表します。アクティビティー・インスタンスは、オブジェクトの識別子、またはプロセス・インスタンス内での完全修飾名によって一意に識別されます。アクティビティー・インスタンスの完全修飾名はドット表記の名前です。ネストされたブロック 型のアクティビティーの階層は左から右へと表記され、その名前はドットで区切ります。

以下の部分では、アクティビティー・インスタンスに適用できるアクションについて説明します。関数 / メソッドの完全なリストについては、258ページの『アクティビティー・インスタンス』を参照してください。

InContainer()

この関数 / メソッドは、アクティビティー・インスタンスに関連する入力コンテナーを、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセスの許可
- プロセス管理の許可
- プロセス・インスタンス作成者として
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ActivityInstance

ActiveX のシグニチャー

```
long InContainer( Container * input )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjActivityInstanceInContainer( FmcjActivityInstanceHandle    hdlInstance
                                 FmcjReadOnlyContainerHandle * input     )
```

C++ 言語のシグニチャー

```
APIRET InContainer( FmcjReadOnlyContainer & input ) const
```

Java のシグニチャー

```
public abstract
ReadOnlyContainer inContainer() throws FmcException
```

パラメーター

hdlInstance 入力。処理するアクティビティー・インスタンスのハンドル。

input 入出力。入力コンテナー。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

ReadOnlyContainer
入力コンテナー。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
アクティビティー・インスタンスはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

ObtainProcessInstanceMonitor()/ ObtainInstanceMonitor

この関数 / メソッドは、アクティビティー・インスタンスが属するプロセス・インスタンスのプロセス・インスタンス・モニターを MQSeries Workflow 実行サーバーから取り出します。

deep オプションを指定すると、ブロック型のアクティビティー・インスタンスすべてが解決され、それらのブロック・インスタンス・モニターもサーバーから取り出されます。

注: 現時点では、deep はサポートされていません。

C++ の場合、初期化するプロセス・インスタンス・モニター・オブジェクトが空でなければ、そのオブジェクトがまず破棄されてから、新しいオブジェクトが割り当てられます。 C の場合は、オブジェクトの所有権についてはアプリケーションの側が完全に制御することになります。つまり、プロセス・インスタンス・モニターのハンドルがすでに何らかのオブジェクトを指しているかどうかのチェックは実行されません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- プロセスの許可
- プロセス管理の許可
- プロセス管理者として
- プロセス作成者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstance

ActiveX のシグニチャー

```
InstanceMonitor*
ObtainInstanceMonitor( boolean deep, long * returnCode )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjActivityInstanceObtainProcessInstanceMonitor(
       FmcjActivityInstanceHandle             hdlInstance,
       bool                                   deep,
       FmcjProcessInstanceMonitorHandle *     monitor)
```

C++ 言語のシグニチャー

```
APIRET ObtainProcessInstanceMonitor(
       FmcjProcessInstanceMonitor &           monitor,
       bool                                   deep= false ) const
```

Java のシグニチャー

```
public abstract
ProcessInstanceMonitor obtainProcessInstanceMonitor( boolean deep )
throws FmcException
```

パラメーター

deep 入力。ブロック型のアクティビティー・インスタンスを解決するかどうか (つまりそのモニターも取り出すかどうか) を指定する標識。ただし、現時点で deep は無視されます。

hdlInstance 入力。プロセス・インスタンス・モニターを取り出すアクティビティー・インスタンス。

monitor 入出力。設定するプロセス・インスタンス・モニター・オブジェクトまたはプロセス・インスタンス・モニター・オブジェクトのハンドルのアドレス。

returnCode 入出力。このメソッドの呼び出し結果 (下の戻りコードを参照)。

戻り値のデータ型

APIRET このメソッドの呼び出し結果 (下の戻りコードを参照)。

InstanceMonitor*/ProcessInstanceMonitor*/ ProcessInstanceMonitor
プロセス・インスタンス・モニターまたはプロセス・インスタンス・モニターへのポインター。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
アクティビティー・インスタンスはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

OutContainer()

この関数 / メソッドは、アクティビティー・インスタンスに関連する出力コンテナーを、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- プロセスの許可
- プロセス管理の許可
- プロセス・インスタンス作成者として
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstance

ActiveX のシグニチャー

```
long OutContainer( Container * output )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjActivityInstanceOutContainer( FmcjActivityInstanceHandle    hdlInstance,
                                  FmcjReadOnlyContainerHandle * output     )
```

C++ 言語のシグニチャー

```
APIRET OutContainer( FmcjReadOnlyContainer & output ) const
```

Java のシグニチャー

```
public abstract
ReadOnlyContainer outContainer() throws FmcException
```

パラメーター

hdlInstance 入力。処理するアクティビティー・インスタンスのハンドル。

output 入出力。出力コンテナー。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

ReadOnlyContainer
出力コンテナー。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
アクティビティー・インスタンスはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

PersistentObject()

この関数 / メソッドは、MQSeries Workflow 実行サーバーから渡されたオブジェクト識別子で識別されるアクティビティー・インスタンスを取り出します (アクション呼び出し)。

アクティビティー・インスタンスの検索元の MQSeries Workflow 実行サーバーは、実行サービス・オブジェクトによって識別されます。その場合、一時オブジェクトは、アクティビティー・インスタンスのすべての情報 (1 次および 2 次) を使って作成また更新されます。

C++ の場合、初期化するアクティビティー・インスタンス・オブジェクトが空でなければ、そのオブジェクトがまず破棄されてから、新しいオブジェクトが割り当てられます。 C の場合は、オブジェクトの所有権についてはアプリケーションの側が完全に制御することになります。つまり、アクティビティー・インスタンスのハンドルがすでに何らかのオブジェクトを指しているかどうかのチェックは実行されません。Java の場合、アクティビティー・インスタンスは新たに作成されます。実行サービスがファクトリーとして機能します。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセスの許可
- プロセス管理の許可
- プロセス・インスタンス作成者として
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long PersistentObject( ExecutionService * service, BSTR oid )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjActivityInstancePersistentObject(
       FmcjExecutionServiceHandle               service,
       char const *                  oid,
       FmcjActivityInstanceHandle * hdlInstance )
```

C++ 言語のシグニチャー

```
APIRET PersistentObject( FmcjExecutionService const & service,
                         string const &               oid    )
```

Java のシグニチャー

```
public abstract
  ActivityInstance ExecutionService.persistentActivityInstance( String oid )
throws FmcException
```

パラメーター

hdlInstance 入出力。設定するアクティビティー・インスタンスオブジェクトのハンドルのアドレス

oid 入力。検索するアクティビティー・インスタンスのオブジェクト識別子。

service 入力。実行サーバーとのセッションを表すサービス・オブジェクト。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

ActivityInstance
取り出されたアクティビティー・インスタンス。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
アクティビティー・インスタンスはもう存在していません。

FMC_ERROR_INVALID_OID(805)
指定した oid は無効です。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

SubProcessInstance()

この関数 / メソッドは、アクティビティー・インスタンスをインプリメントしているプロセス・インスタンスを MQSeries Workflow 実行サーバーから取り出します (アクション呼び出し)。

プロセス・インスタンスに関するすべての 1 次情報および 2 次情報が取り出されます。

C++ の場合、初期化するプロセス・インスタンス・オブジェクトが空でなければ、そのオブジェクトがまず破棄されてから、新しいオブジェクトが割り当てられます。 C の場合は、オブジェクトの所有権についてはアプリケーションの側が完全に制御することになります。つまり、プロセス・インスタンスのハンドルがすでに何らかのオブジェクトを指しているかどうかのチェックは実行されません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセスの許可
- プロセス管理の許可
- プロセス作成者として
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ActivityInstance

ActiveX のシグニチャー

```
ProcessInstance* SubProcessInstance( long * returnCode )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjActivityInstanceSubProcessInstance(
       FmcjActivityInstanceHandle   hdlInstance,
       FmcjProcessInstanceHandle *  instance )
```

C++ 言語のシグニチャー

```
APIRET SubProcessInstance( FmcjProcessInstance & instance ) const
```

Java のシグニチャー

```
public abstract
ProcessInstance subProcessInstance() throws FmcException
```

パラメーター

hdlInstance 入力。照会するアクティビティー・インスタンス・オブジェクトのハンドル。

instance 入出力。検索 (初期化) するサブプロセス・インスタンス・オブジェクト。

returnCode 入出力。このメソッドの呼び出し結果 (下の戻りコードを参照)。

戻り値のデータ型

APIRET

このメソッドの呼び出し結果 (下の戻りコードを参照)。

ProcessInstance*/ ProcessInstance

サブプロセス・インスタンスまたはサブプロセス・インスタンスへのポインター。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)

パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)

オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)

提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)

アクティビティー・インスタンスはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)

関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)

ログオンしていません。

FMC_ERROR_COMMUNICATION(13)

指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)

MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)

内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)

戻されるメッセージが許可された最大サイズを超えました。シ

ステム、システム・グループ、またはドメインの
MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Terminate()

この関数 / メソッドは、プログラムまたはプロセスによってインプリメントされたアクティビティー・インスタンスを中止します (アクション呼び出し)。

アクティビティー・インスタンスがプログラムによってインプリメントされている場合、その状態は *CheckedOut (チェックアウト済み)*、または *Running (実行中)* のいずれかでなければならず、プロセス・インスタンスは *Running (実行中)*、*Suspending (一時中断処理中)*、*Suspended (一時中断)*、または *Terminating (中止処理中)* のいずれかの状態でなければなりません。アクティビティー・インスタンスがプロセスによってインプリメントされている場合、その状態は *Running (実行中)*、*Suspending (一時中断処理中)*、または *Suspended (一時中断済み)* のいずれかでなければならず、プロセス・インスタンスは *Running (実行中)*、*Suspending (一時中断処理中)*、 *Suspended (一時中断済み)*、または *Terminating (中止処理中)* のいずれかの状態でなければなりません。

プロセスによってインプリメントされるアクティビティー・インスタンスは、制御の自立性に関して自立走行式でないそのサブプロセスと共に中止させられます。

その後、アクティビティー・インスタンスは *Terminating (中止処理中)* または *Terminated (中止済み)* 状態になります。

Terminated (中止) 状態に達すると終了条件が偽とみなされ、出力コンテナー、特に戻りコード (_RC) は設定されず、ナビゲーションが終了します。ナビゲーションはプロセス管理権をもつユーザーによって明示的に継続させることが可能です。つまり、ForceFinish() または ForceRestart() 修復アクションを呼び出すことができます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセス管理の許可

- 関連する作業項目の開始者として
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ActivityInstance

ActiveX のシグニチャー

```
long Terminate()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjActivityInstanceTerminate( FmcjActivityInstanceHandle hdlInstance )
```

C++ 言語のシグニチャー

```
APIRET Terminate()
```

Java のシグニチャー

```
public abstract
void   terminate() throws FmcException
```

パラメーター

hdlInstance 入力。中止するアクティビティー・インスタンスのハンドル。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
アクティビティー・インスタンスはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_STATE(120)
アクティビティー・インスタンスまたはプロセス・インスタンスの状態が間違っています。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

第39章 アクティビティー・インスタンス通知のアクション

FmcjActivityInstanceNotification または ActivityInstanceNotification オブジェクトは、あるユーザーに割り当てられたアクティビティー・インスタンス上での通知を表します。

ユーザーに割り当てられているその他のアイテムは、プロセス・インスタンス通知および作業項目です。 FmcjItem または Itemは、すべてのアイテムの共通プロパティーを表します。

C++ 言語において FmcjActivityInstanceNotification は FmcjItem クラスのサブクラスであり、すべてのプロパティーとメソッドを継承します。 Java 言語において ActivityInstanceNotification は Item クラスのサブクラスであり、すべてのプロパティーとメソッドを継承します。同じように C 言語では、関数の共通のインプリメンテーションを FmcjItem から取得します。つまり、それらの関数には共通の接頭部 FmcjItem が付いており、定義においては先頭に接頭部 FmcjActivityInstanceNotification が付けられます。 ActiveX では継承がサポートされていないため、すべての関数は ActivityInstanceNotification で明示的に定義されます。しかし、それらのメソッドは Item アクションとして記述される点に注意してください。

アクティビティー・インスタンス通知は、オブジェクトの識別子によって一意に識別されます。

以下の部分では、アクティビティー・インスタンス通知に適用できるアクションについて説明します。関数 / メソッドの完全なリストについては、264ページの『アクティビティー・インスタンス通知』を参照してください。

ActivityInstance()

関数 / メソッドは、アクティビティー・インスタンス通知が関連するアクティビティー・インスタンスを MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

アクティビティー・インスタンスに関するすべての 1 次情報および 2 次情報が取り出されます。

C++ の場合、初期化するアクティビティー・インスタンス・オブジェクトが空でなければ、そのオブジェクトがまず破棄されてから、新しいオブジェクトが

割り当てられます。C の場合は、オブジェクトの所有権についてはアプリケーションの側が完全に制御することになります。つまり、アクティビティー・インスタンスのハンドルがすでに何らかのオブジェクトを指しているかどうかのチェックは実行されません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセスの許可
- プロセス管理の許可
- プロセス作成者として
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ActivityInstanceNotification

ActiveX のシグニチャー

```
ActivityInstance* ActivityInstance( long * returnCode )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjActivityInstanceNotificationActivityInstance(
       FmcjActivityInstanceNotificationHandle hdlItem,
       FmcjActivityInstanceHandle *     instance )
```

C++ 言語のシグニチャー

```
APIRET ActivityInstance( FmcjActivityInstance & instance ) const
```

Java のシグニチャー

```
public abstract
ActivityInstance activityInstance() throws FmcException
```

パラメーター

hdlItem 入力。照会するアクティビティー・インスタンス通知・オブジェクトのハンドル。

instance 入出力。検索 (初期化) するアクティビティー・インスタンス・オブジェクト。

returnCode 入出力。このメソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻り値のデータ型

APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ActivityInstance*/ ActivityInstance

アクティビティー・インスタンスへのポインター、またはアクティビティー・インスタンス通知が関連するアクティビティー・インスタンス。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)

パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)

オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)

提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
アクティビティー・インスタンス通知はもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

PersistentObject()

この関数 / メソッドは、MQSeries Workflow 実行サーバーから渡されたオブジェクト識別子で識別されるアクティビティー・インスタンス通知を取り出します (アクション呼び出し)。

アクティビティー・インスタンス通知の検索元の MQSeries Workflow 実行サーバーは、実行サービス・オブジェクトによって識別されます。その場合、一時オブジェクトは、アクティビティー・インスタンス通知のすべての情報 (1 次および 2 次) を使って作成また更新されます。

C++ の場合、初期化するアクティビティー・インスタンス通知オブジェクトが空でなければ、そのオブジェクトがまず破棄されてから、新しいオブジェクトが割り当てられます。 C の場合は、オブジェクトの所有権についてはアプリケーションの側が完全に制御することになります。つまり、アクティビティー・インスタンス通知のハンドルがすでに何らかのオブジェクトを指している

かどうかのチェックは実行されません。 Java の場合、アクティビティー・インスタンス通知は新たに作成されます。実行サービスがファクトリーとして機能します。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- アイテムの所有者であること
- 作業項目許可
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long PersistentObject( ExecutionService * service, BSTR oid )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjActivityInstanceNotificationPersistentObject(
       FmcjExecutionServiceHandle                    service,
       char const *                                  oid,
       FmcjActivityInstanceNotificationHandle *      hdlItem )
```

C++ 言語のシグニチャー

```
APIRET PersistentObject( FmcjExecutionService const & service,
                         string const &               oid )
```

Java のシグニチャー

```
public abstract
  ActivityInstanceNotification
  ExecutionService.persistentActivityInstanceNotification( String oid )
throws FmcException
```

パラメーター

hdlItem 入出力。設定するアクティビティー・インスタンス通知オブジェクトのハンドルのアドレス。

oid 入力。検索するアクティビティー・インスタンス通知のオブジェクト識別子。

service 入力。実行サーバーとのセッションを表すサービス・オブジェクト。

戻り値のデータ型

ActivityInstanceNotification
取り出されたアクティビティー・インスタンス通知。

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
アクティビティー・インスタンス通知はもう存在していません。

FMC_ERROR_INVALID_OID(805)
指定した oid は無効です。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

StartTool()

この関数 / メソッドは、指定したサポート・ツールを開始します (アクション呼び出し)。

サポート・ツールは、通知の元となるアクティビティー・インスタンスに関連したツールの 1 つでなければなりません。ログオン・ユーザーに関連したプログラム実行エージェント上で開始されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

アクティビティー・インスタンス通知の所有者であること

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ActivityInstanceNotification

ActiveX のシグニチャー

```
long StartTool( BSTR toolName )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjActivityInstanceNotificationStartTool(
                    FmcjActivityInstanceNotificationHandle hdlItem,
                    char const *                           toolName )
```

C++ 言語のシグニチャー

```
APIRET StartTool( string const & toolName ) const
```

Java のシグニチャー

```
public abstract
void   startTool( String toolName ) throws FmcException
```

パラメーター

hdlItem 入力。処理するアクティビティー・インスタンス通知のハンドル。

toolName 入力。開始するサポート・ツール。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
作業項目はもう存在していません。

FMC_ERROR_INVALID_TOOL(129)
ツール名が 1 つも提供されていないか、指定したツールがアクティビティー・インスタンス通知に対して定義されていません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

第40章 ブロック・インスタンス・モニターのアクション

FmcjBlockInstanceMonitor または BlockInstanceMonitor オブジェクトは、ブロック 型のアクティビティー・インスタンスのモニターを表します。

注: ブロック・インスタンス・モニターの所有権は、それを包含しているプロセス・インスタンス・モニターのものです。プロセス・インスタンス・モニターが削除されると、ブロック・インスタンス・モニターも自動的に削除されます。このアクションの後は、ブロック・インスタンス・モニター・ハンドルまたはオブジェクトを使用することは無効になります。

FmcjBlockInstanceMonitor または BlockInstanceMonitor オブジェクトは、モニターに共通のいくつかの局面を表します。 C++ 言語の場合、FmcjBlockInstanceMonitor は FmcjProcessInstanceMonitor クラスのスーパークラスで、すべての共通のプロパティーとメソッドを提供します。 Java 言語の場合、BlockInstanceMonitor は ProcessInstanceMonitor クラスのスーパークラスで、すべての共通のプロパティーとメソッドを提供します。同じように C 言語では、関数の共通のインプリメンテーションを FmcjBlockInstanceMonitor から取得します。つまり、それらの関数には共通の接頭部 FmcjBlockInstanceMonitor が付いており、定義においては先頭に接頭部 FmcjProcessInstanceMonitor が付けられます。 ActiveX では継承がサポートされていません。そのため、すべてのメソッドは InstanceMonitor クラスで定義されます (515ページの『第43章 インスタンス・モニターのアクション』を参照)。

以下の部分では、ブロック・インスタンス・モニターに適用できるアクションについて説明します。関数 / メソッドの完全なリストについては、273ページの『ブロック・インスタンス・モニター』を参照してください。

ObtainBlockInstanceMonitor()

この関数 / メソッドは、指定したアクティビティー・インスタンスのブロック・インスタンス・モニターを MQSeries Workflow 実行サーバーから取り出します (アクション呼び出し)。ブロック・インスタンス・モニターがすでに取り出されている場合、そのモニターが呼び出し側に戻されます。

指定するアクティビティー・インスタンスはブロック 型で、かつこのブロック・インスタンス・モニターの一部でなければなりません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- プロセスの許可
- プロセス管理の許可
- プロセス管理者として
- プロセス作成者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX 該当しない (515ページの『第43章 インスタンス・モニターのアクション』を参照)。

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.BlockInstanceMonitor

C 言語のシグニチャー

```
FmcjBlockInstanceMonitorHandle
    FMC_APIENTRY FmcjBlockInstanceMonitorObtainBlockInstanceMonitor(
       FmcjBlockInstanceMonitorHandle hdlMonitor,
       FmcjActivityInstanceHandle      activity )
```

C++ 言語のシグニチャー

```
FmcjBlockInstanceMonitor *
ObtainBlockInstanceMonitor( FmcjActivityInstance const & activity ) const
APIRET
ObtainBlockInstanceMonitor( FmcjActivityInstance const & activity,
                            FmcjBlockInstanceMonitor &   monitor ) const
```

Java のシグニチャー

```
public abstract
    BlockInstanceMonitor obtainBlockInstanceMonitor(
       ActivityInstance activity ) throws FmcException
```

パラメーター

activity 入力。ブロック・インスタンス・モニターを取り出すブロック型アクティビティー・インスタンス。

hdlMonitor 入力。ブロック型のアクティビティー・インスタンスが含まれているブロック・インスタンス・モニター。

monitor 入出力。取り出されたブロック・インスタンス・モニター。

戻り値のデータ型

APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

FmcjBlockInstanceMonitor*/ Handle/ BlockInstanceMonitor
ブロック・インスタンス・モニター、あるいはブロック・インスタンス・モニターへのポインターまたはハンドル。

APIRET または MQSeries Workflow の**結果オブジェクト**は、次に示すコードを戻します。または以下の FmcExceptions が生じることがあります。

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
指定されたアクティビティー・インスタンスはブロック・インスタンス・モニターによって記述されていません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_KIND(501)
指定されたアクティビティー・インスタンスはブロック型ではありません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

ObtainProcessInstanceMonitor()

この関数 / メソッドは、指定したアクティビティー・インスタンスのプロセス・インスタンス・モニターを MQSeries Workflow 実行サーバーから取り出します (アクション呼び出し)。プロセス・インスタンス・モニターがすでに取り出されている場合、そのモニターが呼び出し側に戻されます。

指定するアクティビティー・インスタンスはプロセス 型で、かつこのブロック・インスタンス・モニターの一部でなければなりません。

deep オプションを指定すると、ブロック型のアクティビティー・インスタンスが解決され、それらのブロック・インスタンス・モニターもサーバーから取り出されます。

注: 現時点では、deep はサポートされていません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- プロセスの許可
- プロセス管理の許可
- プロセス管理者として
- プロセス作成者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX 該当しない (515ページの『第43章 インスタンス・モニターのアクション』を参照)。

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.BlockInstanceMonitor

C 言語のシグニチャー

```
FmcjProcessInstanceMonitorHandle
    FMC_APIENTRY FmcjBlockInstanceMonitorObtainProcessInstanceMonitor(
       FmcjBlockInstanceMonitorHandle hdlMonitor,
       FmcjActivityInstanceHandle     activity,
       bool                           deep      )
```

C++ 言語のシグニチャー

```
FmcjProcessInstanceMonitor *
    ObtainProcessInstanceMonitor(
       FmcjActivityInstance const &     activity,
       bool                             deep= false   ) const
APIRET ObtainProcessInstanceMonitor(
       FmcjActivityInstance const &     activity,
       FmcjProcessInstanceMonitor &     monitor,
       bool                             deep= false   ) const
```

Java のシグニチャー

```
public abstract
  ProcessInstanceMonitor
  obtainProcessBlockInstanceMonitor( ActivityInstance activity,
                                     boolean          deep    )
throws FmcException
```

パラメーター

activity 入力。プロセス・インスタンス・モニターを取り出すプロセス型アクティビティー・インスタンス。

deep 入力。ブロック型のアクティビティー・インスタンスを解決するかどうか (つまりそのモニターも取り出すかどうか) を指定する標識。ただし、現時点で deep は無視されます。

hdlMonitor 入力。プロセス型のアクティビティー・インスタンスが含まれているブロック・インスタンス・モニター。

monitor 出力。取り出されたプロセス・インスタンス・モニター。

戻り値のデータ型

APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

FmcjProcessInstanceMonitor*/ Handle/ ProcessInstanceMonitor
プロセス・インスタンス・モニター、あるいはプロセス・インスタンス・モニターへのポインターまたはハンドル。

APIRET または MQSeries Workflow の**結果オブジェクト**は、次に示すコードを戻します。または以下の FmcExceptions が生じることがあります。

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
指定されたアクティビティー・インスタンスはブロック・インスタンス・モニターによって記述されていません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_KIND(501)
指定されたアクティビティー・インスタンスはプロセス型ではありません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Refresh()

この関数 / メソッドは、MQSeries Workflow 実行サーバーからブロック・インスタンス・モニターをリフレッシュします (アクション呼び出し)。

ブロック・インスタンス・モニターに関するすべての情報が検索されます。

deep オプションを指定すると、ブロック型のアクティビティー・インスタンスが解決され、それらのブロック・インスタンス・モニターもサーバーからリフレッシュされます。

注: 現時点では、deep はサポートされていません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- プロセスの許可
- プロセス管理の許可
- プロセス管理者として
- プロセス作成者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX 該当しない (515ページの『第43章 インスタンス・モニターのアクション』を参照)。

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.BlockInstanceMonitor

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjBlockInstanceMonitorRefresh(
       FmcjBlockInstanceMonitorHandle hdlMonitor,
       bool                           deep      )
```

C++ 言語のシグニチャー

```
APIRET Refresh( bool deep= false )
```

Java のシグニチャー

```
public abstract
void   refresh( boolean deep ) throws FmcException
```

パラメーター

deep 入力。ブロック型のアクティビティー・インスタンスを解決するかどうか (つまりそのモニターも取り出すかどうか) を指定する標識。ただし、現時点で deep は無視されます。

hdlMonitor 入力。リフレッシュするブロック・インスタンス・モニターのハンドル。

戻り値のデータ型

APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1) パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122) オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130) 提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118) プロセス・インスタンスはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119) 関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106) ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14) タイムアウトになりました。

第41章 コンテナー・アクティビティー・インプリメンテーションの関数 / メソッド

FmcjContainer または Container オブジェクトは、プロセス・テンプレート、プロセス・インスタンス、作業項目、アクティビティー・インプリメンテーション、またはサポート・ツールのデータ・コンテナーを表します。コンテナーは、読み取り専用の入力コンテナーか、読み取り / 書き込みの入力または出力コンテナーのいずれかです。

コンテナーで定義されている関数 / メソッドでは、基本型のデータ・メンバーの値にアクセスしたり (コンテナー・リーフ)、コンテナーの部分構造体であるコンテナー要素を入手したりできます。

FmcjContainer または Container オブジェクトは、読み取り / 書き込みコンテナーに共通のいくつかの局面を表します。 C++ 言語の場合、FmcjContainer は FmcjReadOnlyContainer クラスと FmcjReadWriteContainer クラスのスーパークラスで、すべての共通のプロパティーとメソッドを提供します。 Java 言語の場合、Container は ReadOnlyContainer クラスと ReadWriteContainer クラスのスーパークラスで、すべての共通のプロパティーとメソッドを提供します。同じように C 言語では、関数の共通のインプリメンテーションを FmcjContainer から取得します。つまり、共通関数には接頭部 FmcjContainer が付いており、定義においては先頭に接頭部 FmcjReadOnlyContainer および FmcjReadWriteContainer が付けられます。 ActiveX では継承がサポートされていません。 Container クラスではすべてのメソッドが利用可能です。

以下の部分では、アクティビティー・インプリメンテーションまたはサポート・ツールとプログラム実行エージェントの間の通信用に使用されるアクティビティー・インプリメンテーション関数について説明します。コンテナーの関数 / メソッドの完全なリストについては、274ページの『コンテナー』を参照してください。

InContainer()

この関数 / メソッドは、MQSeries Workflow プログラム実行エージェントから入力コンテナーを取り出します (アクティビティー・インプリメンテーション呼び出し)。

これは、アクティビティー・インプリメンテーションまたはサポート・ツール内から使用できます。

使用上の注意

- 141ページの『アクティビティー・インプリメンテーション関数 / メソッド』にある一般的な情報を参照してください。

許可

アクティビティー・インプリメンテーションまたはサポート・ツールであること

必要な接続

なし。ただし MQSeries Workflow プログラム実行エージェントはアクティブでなければなりません。

API インターフェース宣言

ActiveX	IBM MQSeries Workflow 3.1
C 言語	fmcjccon.h または fmcjcrun.h
C++	fmcjpcon.hxx または fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionAgent

ActiveX のシグニチャー

```
long InContainer()
```

C 言語のシグニチャー

```
APIRET  FMC_APIENTRY FmcjContainerInContainer(
        FmcjReadOnlyContainerHandle * input )
```

C++ 言語のシグニチャー

```
static APIRET InContainer( FmcjReadOnlyContainer & input )
```

Java のシグニチャー

```
public abstract
       ReadOnlyContainer ExecutionAgent.inContainer()
throws FmcException
```

パラメーター

input 入出力。設定するアクティビティー・インプリメンテーションまたはサポート・ツールの入力コンテナー・ハンドルのアドレス (C)、またはその入力コンテナーのアドレス (C++)。

戻り値のデータ型

long/ APIRET

この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ReadOnlyContainer

アクティビティー・インプリメンテーションまたはサポート・ツールの入力コンテナー。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)

パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)

オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)

提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_NO_CTNR_ACCESS(1021)

プログラムに入力コンテナーがありません。

FMC_ERROR_NOT_AUTHORIZED(119)

関数 / メソッドを使う許可がありません。

FMC_ERROR_PROGRAM_EXECUTION(126)

関数 / メソッドがアクティビティー・インプリメンテーションまたはサポート・ツール内から呼び出されていないか、プログラム実行エージェントがアクティブではありません。

FMC_ERROR_COMMUNICATION(13)
指定されたプログラム実行エージェントに到達できません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

例

- C 言語の例については、857ページの『実行可能なアクティビティー・インプリメンテーションのプログラミング (C 言語)』を参照してください。
- C++ の例については、858ページの『実行可能なアクティビティー・インプリメンテーションのプログラミング (C++)』を参照してください。

OutContainer()

この関数 / メソッドは、MQSeries Workflow プログラム実行エージェントから出力コンテナーを取り出します (アクティビティー・インプリメンテーション呼び出し)。

これは、アクティビティー・インプリメンテーション内から使用できます。

使用上の注意

- 141ページの『アクティビティー・インプリメンテーション関数 / メソッド』にある一般的な情報を参照してください。

許可

アクティビティー・インプリメンテーションであること

必要な接続

なし。ただし MQSeries Workflow プログラム実行エージェントはアクティブでなければなりません。

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjccon.h または fmcjcrun.h
C++	fmcjpcon.hxx または fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionAgent

ActiveX のシグニチャー

```
long OutContainer()
```

C 言語のシグニチャー

```
APIRET  FMC_APIENTRY FmcjContainerOutContainer(
        FmcjReadWriteContainerHandle * output )
```

C++ 言語のシグニチャー

```
static APIRET OutContainer( FmcjReadWriteContainer & output )
```

Java のシグニチャー

```
public abstract
       ReadWriteContainer ExecutionAgent.outContainer()
throws FmcException
```

パラメーター

output 入出力。設定されるアクティビティー・インプリメンテーションの出力コンテナー・ハンドルのアドレス (C)、またはその出力コンテナーのアドレス (C++)。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ReadWriteContainer
アクティビティー・インプリメンテーションの出力コンテナー。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_NO_CTNR_ACCESS(1021)
プログラムに出力コンテナーがありません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_PROGRAM_EXECUTION(126)
関数 / メソッドがアクティビティー・インプリメンテーション内から呼び出されていないか、プログラム実行エージェントがアクティブではありません。

FMC_ERROR_TOOL_FUNCTION(128)
サポート・ツールは出力コンテナーにアクセスできません。

FMC_ERROR_COMMUNICATION(13)
指定されたプログラム実行エージェントに到達できません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

例

- C 言語の例については、857ページの『実行可能なアクティビティー・インプリメンテーションのプログラミング (C 言語)』を参照してください。
- C++ の例については、858ページの『実行可能なアクティビティー・インプリメンテーションのプログラミング (C++)』を参照してください。

RemoteInContainer()

この関数 / メソッドは、MQSeries Workflow プログラム実行エージェントから入力コンテナーを取り出します (アクティビティー・インプリメンテーション呼び出し)。

これは、アクティビティー・インプリメンテーションまたはサポート・ツールによって開始されたプログラム内から使用できます。

使用上の注意

- 141ページの『アクティビティー・インプリメンテーション関数 / メソッド』にある一般的な情報を参照してください。

許可

有効なプログラム識別

必要な接続

なし。ただし MQSeries Workflow プログラム実行エージェントはアクティブでなければなりません。

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjccon.h または fmcjcrun.h
C++	fmcjpcon.hxx または fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionAgent

ActiveX のシグニチャー

```
long RemoteInContainer( BSTR programID )
```

C 言語のシグニチャー

```
APIRET  FMC_APIENTRY FmcjContainerRemoteInContainer(
        char const *                  programID,
        FmcjReadOnlyContainerHandle * input )
```

C++ 言語のシグニチャー

```
static APIRET RemoteInContainer(
       string const &               programID,
       FmcjReadOnlyContainer &      input )
```

Java のシグニチャー

```
public abstract
   ReadOnlyContainer ExecutionAgent.remoteInContainer( String programID )
throws FmcException
```

パラメーター

input 入出力。設定されるアクティビティー・インプリメンテーションまたはサポート・ツールの入力コンテナー・ハンドルのアドレス (C)、またはその入力コンテナーのアドレス (C++)。

programID 入力。アクティビティー・インプリメンテーションまたはサポート・ツールがプログラム実行エージェントに認識されるのに必要なプログラム識別子。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ReadOnlyContainer
アクティビティー・インプリメンテーションまたはサポート・ツールの入力コンテナー。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_PROGRAMID(135)
プログラム識別が無効です。

FMC_ERROR_NO_CTNR_ACCESS(1021)
プログラムに入力コンテナーがありません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_PROGRAM_EXECUTION(126)
関数 / メソッドがアクティビティー・インプリメンテーション内から呼び出されていないか、プログラム実行エージェントがアクティブではありません。

FMC_ERROR_COMMUNICATION(13)
指定されたプログラム実行エージェントに到達できません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

RemoteOutContainer()

この関数 / メソッドは、MQSeries Workflow プログラム実行エージェントから出力コンテナーを取り出します (アクティビティー・インプリメンテーション呼び出し)。

これは、アクティビティー・インプリメンテーションによって開始されたプログラム内から使用できます。

使用上の注意

- 141ページの『アクティビティー・インプリメンテーション関数 / メソッド』にある一般的な情報を参照してください。

許可

有効なプログラム識別

必要な接続

なし。ただし MQSeries Workflow プログラム実行エージェントはアクティブでなければなりません。

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjccon.h または fmcjcrun.h
C++	fmcjpcon.hxx または fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionAgent

ActiveX のシグニチャー

```
long RemoteOutContainer( BSTR programID )
```

C 言語のシグニチャー

```
APIRET  FMC_APIENTRY FmcjContainerRemoteOutContainer(
        char const *                  programID,
        FmcjReadWriteContainerHandle * output )
```

C++ 言語のシグニチャー

```
static APIRET RemoteOutContainer(
       string const &                 programID,
       FmcjReadWriteContainer &       output )
```

Java のシグニチャー

```
public abstract
   ReadWriteContainer ExecutionAgent.remoteOutContainer( String programID )
throws FmcException
```

パラメーター

output 入出力。設定されるアクティビティー・インプリメンテーションの出力コンテナー・ハンドルのアドレス (C)、またはその入力コンテナーのアドレス (C++)。

programID 入力。アクティビティー・インプリメンテーションがプログラム実行エージェントに認識されるのに必要なプログラム識別子。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ReadWriteContainer
アクティビティー・インプリメンテーションの出力コンテナー。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_PROGRAMID(135)
プログラム識別が無効です。

FMC_ERROR_NO_CTNR_ACCESS(1021)
プログラムに出力コンテナーがありません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_PROGRAM_EXECUTION(126)
関数 / メソッドがアクティビティー・インプリメンテーション内から呼び出されていないか、プログラム実行エージェントがアクティブではありません。

FMC_ERROR_TOOL_FUNCTION(128)
サポート・ツールは出力コンテナーにアクセスできません。

FMC_ERROR_COMMUNICATION(13)
指定されたプログラム実行エージェントに到達できません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

SetOutContainer()

この関数 / メソッドは、出力コンテナーを MQSeries Workflow プログラム実行エージェントに戻します (アクティビティー・インプリメンテーション呼び出し)。

これは、アクティビティー・インプリメンテーション内から任意の頻度で使用できます。しかし、出力コンテナーはアクティビティー・インプリメンテーションが終了するまで MQSeries Workflow 実行サーバーに戻されないことに注意してください。プログラム実行エージェントによって一時的に保持されます。

使用上の注意

- 141ページの『アクティビティー・インプリメンテーション関数 / メソッド』にある一般的な情報を参照してください。

許可

アクティビティー・インプリメンテーションであること

必要な接続

なし。ただし MQSeries Workflow プログラム実行エージェントはアクティブでなければなりません。

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjccon.h または fmcjcrun.h
C++	fmcjpcon.hxx または fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionAgent

ActiveX のシグニチャー

```
long SetOutContainer()
```

C 言語のシグニチャー

```
APIRET  FMC_APIENTRY FmcjContainerSetOutContainer(
        FmcjReadWriteContainerHandle const output )
```

C++ 言語のシグニチャー

```
static APIRET SetOutContainer( FmcjReadWriteContainer const & output )
```

Java のシグニチャー

```
public abstract
  void ExecutionAgent.setOutContainer( ReadWriteContainer output )
throws FmcException
```

パラメーター

output 入力。渡すアクティビティー・インプリメンテーションの出力コンテナー・ハンドル (C)、またはその出力コンテナー (C++)。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_CONTAINER(509)
渡されたコンテナーがアクティビティー・インプリメンテーションに有効な出力コンテナーではありません。スキーマまたはバージョンが間違っています。

FMC_ERROR_NO_CTNR_ACCESS(1021)
プログラムに出力コンテナーがありません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_PROGRAM_EXECUTION(126)
関数 / メソッドがアクティビティー・インプリメンテーション内から呼び出されていないか、プログラム実行エージェントがアクティブではありません。

FMC_ERROR_TOOL_FUNCTION(128)
サポート・ツールは出力コンテナーを設定できません。

FMC_ERROR_COMMUNICATION(13)
指定されたプログラム実行エージェントに到達できません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

例

- C 言語の例については、857ページの『実行可能なアクティビティー・インプリメンテーションのプログラミング (C 言語)』を参照してください。
- C++ の例については、858ページの『実行可能なアクティビティー・インプリメンテーションのプログラミング (C++)』を参照してください。

SetRemoteOutContainer()

この関数 / メソッドは、出力コンテナーを MQSeries Workflow プログラム実行エージェントに戻します (アクティビティー・インプリメンテーション呼び出し)。

これは、アクティビティー・インプリメンテーションによって開始されたプログラム内から任意の頻度で使用できます。しかし、出力コンテナーはアクティビティー・インプリメンテーションが終了するまで MQSeries Workflow 実行サーバーに戻されないことに注意してください。プログラム実行エージェントによって一時的に保持されます。

使用上の注意

- 141ページの『アクティビティー・インプリメンテーション関数 / メソッド』にある一般的な情報を参照してください。

許可

有効なプログラム識別

必要な接続

なし。ただし MQSeries Workflow プログラム実行エージェントはアクティブでなければなりません。

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjccon.h または fmcjcrun.h

C++ fmcjpcon.hxx または fmcjprun.hxx

JAVA com.ibm.workflow.api.ExecutionAgent

ActiveX のシグニチャー

```
long SetRemoteOutContainer( BSTR programID )
```

C 言語のシグニチャー

```
APIRET  FMC_APIENTRY FmcjContainerSetRemoteOutContainer(
        char const *                       programID,
        FmcjReadWriteContainerHandle const output )
```

C++ 言語のシグニチャー

```
static APIRET SetRemoteOutContainer(
       string const &                  programID,
       FmcjReadWriteContainer const &  output )
```

Java のシグニチャー

```
public abstract
void   ExecutionAgent.setRemoteOutContainer( String             programID,
                                             ReadWriteContainer output   )
throws FmcException
```

パラメーター

output 入力。渡すアクティビティー・インプリメンテーションの出力コンテナー・ハンドル (C)、またはその出力コンテナー (C++)。

programID 入力。アクティビティー・インプリメンテーションがプログラム実行エージェントに認識されるのに必要なプログラム識別子。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_CONTAINER(509)
渡されたコンテナーがアクティビティー・インプリメンテーションに有効な出力コンテナーではありません。スキーマまたはバージョンが間違っています。

FMC_ERROR_INVALID_PROGRAMID(135)
プログラム識別が無効です。

FMC_ERROR_NO_CTNR_ACCESS(1021)
プログラムに出力コンテナーがありません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_PROGRAM_EXECUTION(126)
関数 / メソッドがアクティビティー・インプリメンテーション内から呼び出されていないか、プログラム実行エージェントがアクティブではありません。

FMC_ERROR_TOOL_FUNCTION(128)
サポート・ツールは出力コンテナーを設定できません。

FMC_ERROR_COMMUNICATION(13)
指定されたプログラム実行エージェントに到達できません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

第42章 実行サービスのアクション

FmcjExecutionService または ExecutionService オブジェクトは、実行機能サービスを要求するための、ユーザーと MQSeries Workflow 実行サーバーの間のセッションを表します。

実行サービス・オブジェクトは基本的に、指定された MQSeries Workflow 実行サーバーへの通信パスをセットアップしたり、ユーザー・セッションを確立 (ログオン) したり終了 (ログオフ) したりするための基本的な関数 / メソッドを提供します。

FmcjExecutionService または ExecutionService の構成または割り振りの際、実行サーバーが属する MQSeries Workflow システムやシステム・グループを指定することができます。省略時値は、ログオン時に、現行のユーザー・プロファイルまたは構成プロファイルから (この順序で) 取られます。また、プロファイルを検索する場所の構成を指定できます。

実行サーバーとのセッションが確立されている場合には、許可されているオブジェクト (たとえばプロセス・テンプレート、プロセス・インスタンス、または作業項目) を照会することができます。その後、照会したオブジェクトの属性を読んで、さらにアクションを要求することができます。たとえば、プロセス・テンプレートが照会された後、プロセス・インスタンスの作成を要求することができます。

実行サービス・オブジェクトが破棄または割り振り解除される際に、まだアクティブなセッションを表していれば、ログオフが自動的に呼び出されます (このセッションを参照しているオブジェクトが他にない場合)。しかし、実行サービス・オブジェクトを割り振り解除する前には、ログオンとログオフの呼び出しを対にして行うようにしてください。

FmcjService または Service は、サービスの共通のプロパティーを表します。

C++ 言語において FmcjExecutionService は FmcjService クラスのサブクラスであり、すべてのプロパティーとメソッドを継承します。 Java 言語において ExecutionService は Service クラスのサブクラスであり、すべてのプロパティーとメソッドを継承します。同じように C 言語では、関数の共通のインプリメンテーションを FmcjService から取得します。つまり、それらの関数には共通の接頭部 FmcjService が付いており、定義においては先頭に接頭部

FmcjExecutionService が付けられます。 ActiveX では継承がサポートされていないため、すべての関数は ExecutionService で明示的に定義されます。しかし、それらのメソッドは Service アクションとして記述される点に注意してください。

以下の部分では、実行サービスに適用できるアクションについて説明します。関数 / メソッドの完全なリストについては、291ページの『実行サービス』を参照してください。

CreateProcessInstanceList()

この関数 / メソッドは、MQSeries Workflow 実行サーバーにプロセス・インスタンス・リストを作成し、プロセス・インスタンスをユーザーの好みに応じて、またはユーザー・グループの必要に応じてグループ化できるようにします (アクション呼び出し)。

プロセス・インスタンス・リストは、以下のものによって識別されます。
- 名前 (タイプごとに固有)。
- タイプ (そのリストがパブリック用かプライベート用かを示す標識)。
- 所有者 (タイプがプライベートである場合のリストの所有者)。

リストがパブリックである場合、所有者の指定は無視されます。リストがプライベートで所有者が指定されていない場合、ログオン・ユーザー用にリストが作成されます。

プロセス・インスタンス・リストをパブリックとして、または別のユーザー (ログオン・ユーザー以外) 用のプライベートとして作成する場合、ログオン・ユーザーにはスタッフ定義権限が必要です。

プロセス・インスタンス・リストは、同じプロパティーのプロセス・インスタンスのセットをグループ化します。それらのプロパティーは、検索フィルターを使って定義されます。リストに含めるプロセス・インスタンスの数は、クライアントに戻されるプロセス・インスタンスの最大数を指定するしきい値を使って制限できます。そのしきい値は、指定したソート基準に従ってプロセス・インスタンス・リストがソートされた後で適用されます。プロセス・インスタンスはサーバーでソートされることに注意してください。つまり、サーバーのコード・ページでソート順序が決まります。

プロセス・インスタンス・リストの名前を指定する場合には、以下の規則が適用されます。
- 最大 32 文字 (バイト) を指定できます。

- 現行のロケールに応じて、以下のものを除くあらゆる印刷可能文字を使用できます。

  * ? " ; : .
- ブランクも使用できますが、先行ブランクを使わない、末尾ブランクを使わない、複数のブランクを連続して使わない、という制限があります。

記述を指定する場合には、以下の規則が適用されます。

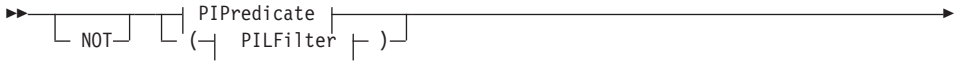
- 最大 254 文字 (バイト) を指定できます。
- 現行のロケールに応じて、復帰文字と改行文字を含むあらゆる印刷可能文字を使用できます。

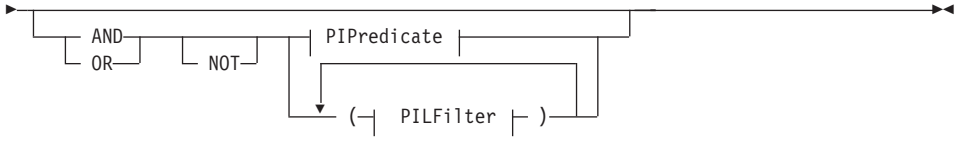
プロセス・インスタンス・リストのフィルターは、プロセス・インスタンスのフィルターからなる文字列として指定します (981ページの『付録A. 構文図の読み方』を参照)。

注:

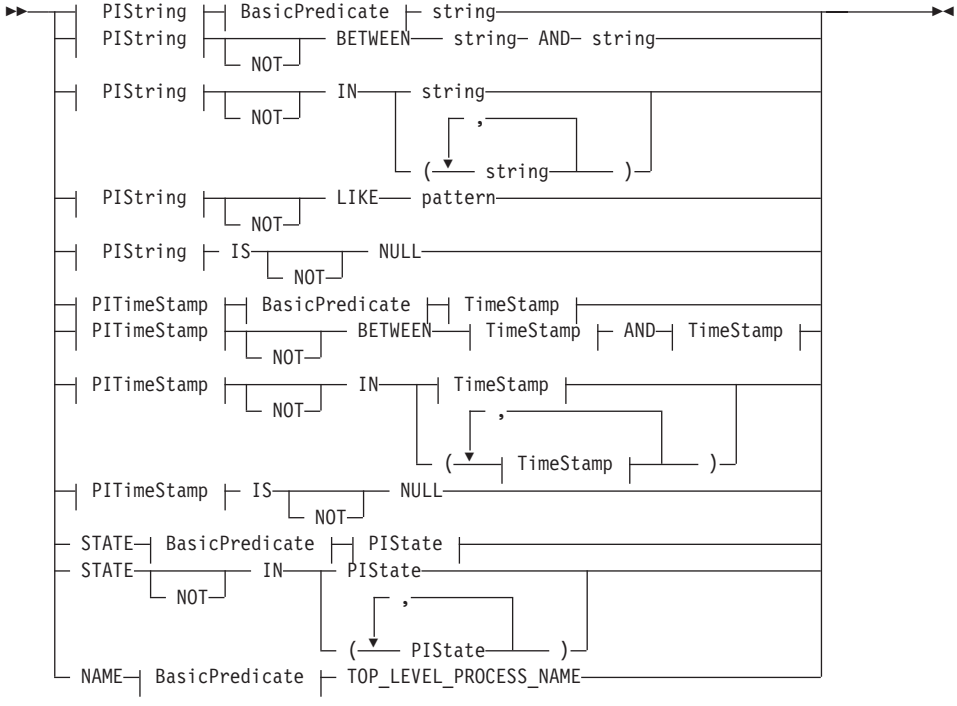
1. *string* (文字列) 定数は、単一引用符 (') で囲む必要があります。
2. ストリング定数内の単一引用符は二重にする ('') 必要があります。
3. *pattern* (パターン) は文字列定数であり、その中のアスタリスクおよび疑問符には特別な意味があります。
   - 疑問符 (?) は任意の単一文字を表します。
   - アスタリスク (*) は、0 文字以上の文字列を表します。
   - エスケープ文字は円記号 (¥) であり、パターン自体に実際の疑問符やアスタリスクが含まれる場合にはこれを使う必要があります。実際の円記号には二重にする必要があります (¥¥)。
4. *TimeStamp* は現地時間に基づいた 24 時間表示のストリング定数です。
5. *TimeStamp* 内のオプションを指定しない場合は、0 (ゼロ) に設定されます。
6. LIKE オペランドのパターンに混合データが含まれている場合は、そのパターンの中にパーセント記号 (%) または下線 (_) を指定することができません。これらのいずれかの文字を使用すると、SQL エラーが戻されます。

PILFilter

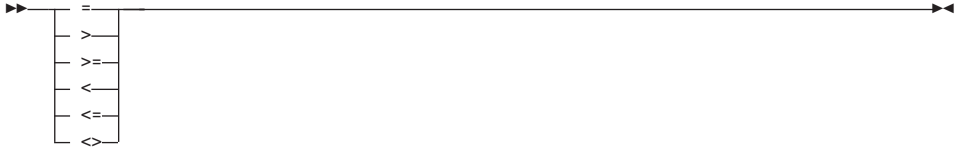




PIPredicate



BasicPredicate

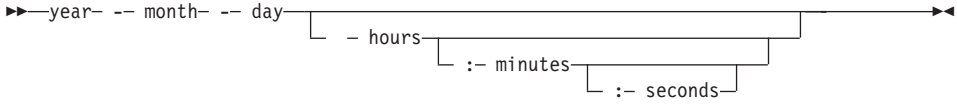


PIState

PIString

PITimeStamp

TimeStamp

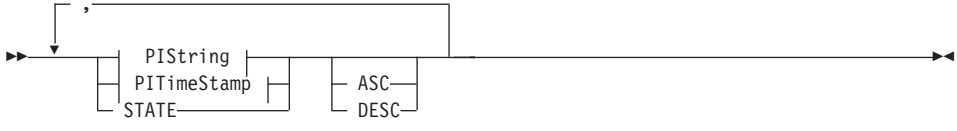


プロセス・インスタンス・リストのソート基準は文字列として指定されます。

注: 省略時のソート順は昇順 (ASC) です。

状態は、PIState の図で示されている順序でソートされます。

PILOrderBy



使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし、またはスタッフ定義あるいはシステム管理者

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long CreateProcessInstanceList(
       BSTR                              name,
       long                              type,
       BSTR                              owner,
       boolean                           ownerIsNull,
       BSTR                              description,
       boolean                           descriptionIsNull,
       BSTR                              filter,
       boolean                           filterIsNull,
       BSTR                              sortCriteria,
       boolean                           sortCriteriaIsNull,
       long                              threshold,
       boolean                           thresholdIsNull )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceCreateProcessInstanceList(
       FmcjExecutionServiceHandle         service,
       char const *                       name,
       enum FmcjPersistentListTypeOfList  type,
       char const *                       owner,
       char const *                       description,
       char const *                       filter,
       char const *                       sortCriteria,
       unsigned long *                    threshold,
       FmcjProcessInstanceListHandle *    newList )
```

C++ 言語のシグニチャー

```
APIRET CreateProcessInstanceList(
       string const &                 name,
       FmcjPersistentList::TypeOfList  type,
       string const *                 owner,
       string const *                 description,
       string const *                 filter,
       string const *                 sortCriteria,
       unsigned long const *          threshold,
       FmcjProcessInstanceList &      newList ) const
```

Java のシグニチャー

```
public abstract
ProcessInstanceList createProcessInstanceList(
       String                         name,
       TypeOfList                     type,
       String                         owner,
       String                         description,
       String                         filter,
       String                         sortCriteria,
       Integer                        threshold   ) throws FmcException
```

パラメーター

description 入力。プロセス・インスタンス・リストに関するユーザー定義の記述。

descriptionIsNull
入力。記述がリストに設定されているかどうかを示します。

filter 入力。プロセス・インスタンス・リストに含められるプロセス・インスタンスを特徴付けるフィルター基準。

filterIsNull 入力。フィルターがリストに設定されているかどうかを示します。

name 入力。プロセス・インスタンス・リスト用にユーザーが定義する名前。

newList 入出力。新しく作成されるプロセス・インスタンス・リスト。

owner 入力。プライベート・タイプの場合のリストの所有者。パブリック用リストの場合は無視。

ownerIsNull 入力。リスト所有者が設定されているかどうかを示します。パブリック・リストには所有者は必要ありません。

service 入力。実行サーバーとのセッションを表すサービス・オブジェクトに対するハンドル。

sortCriteria 入力。プロセス・インスタンス・リスト内のプロセス・インスタンスに適用されるソート基準。

sortCriteriaIsNull
入力。ソート基準がリストに設定されているかどうかを示します。

threshold 入力。クライアントに渡されるプロセス・インスタンス・リスト内のプロセス・インスタンスの最大数を定義するしきい値。

thresholdIsNull
入力。リストのためのしきい値が設定されているかどうかを示します。

type 入力。プライベート用リストを作成するか、パブリック用リストを作成するかを示します。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ProcessInstanceList
新しく作成されるプロセス・インスタンス・リスト。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_DESCRIPTION(810)
指定した記述は無効です。

FMC_ERROR_INVALID_FILTER(125)
指定したフィルターは無効です。

FMC_ERROR_INVALID_LIST_TYPE(813)
指定したリスト・タイプは無効です。

FMC_ERROR_INVALID_NAME(134)
指定したプロセス・インスタンス・リスト名は構文規則に従っていません。

FMC_ERROR_INVALID_USER(132)
リストの所有者に指定したユーザー ID は構文規則に従っていません。

FMC_ERROR_INVALID_SORT(808)
指定したソート基準は無効です。

FMC_ERROR_INVALID_THRESHOLD(807)
指定したしきい値は無効です (可能な最大値を超えています)。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_NOT_AUTHORIZED(119)
許可がありません。

FMC_ERROR_OWNER_NOT_FOUND(812)
プロセス・インスタンス・リストの所有者になる人が見つかりません。

FMC_ERROR_NOT_UNIQUE(121)
プロセス・インスタンス・リストの名前が、指定されたタイプ内で固有ではありません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例
- ActiveX の例については、823ページの『プロセス・インスタンス・リストの作成 (ActiveX)』を参照してください。
- C 言語の例については、824ページの『プロセス・インスタンス・リストの作成 (C 言語)』を参照してください。
- C++ の例については、825ページの『プロセス・インスタンス・リストの作成 (C++)』を参照してください。
- Java の例については、826ページの『プロセス・インスタンス・リストの作成 (Java)』を参照してください。

CreateProcessTemplateList()

この関数 / メソッドは、MQSeries Workflow 実行サーバーにプロセス・テンプレート・リストを作成し、プロセス・テンプレートをユーザーの好みに応じて、またはユーザー・グループの必要に応じてグループ化できるようにします (アクション呼び出し)。

プロセス・テンプレート・リストは、以下のものによって識別されます。
- 名前 (タイプごとに固有)。
- タイプ (そのリストがパブリック用かプライベート用かを示す標識)。
- 所有者 (タイプがプライベートである場合のリストの所有者)。

リストがパブリックである場合、所有者の指定は無視されます。リストがプライベートで所有者が指定されていない場合、ログオン・ユーザー用にリストが作成されます。

プロセス・テンプレート・リストをパブリックとして、または別のユーザー (ログオン・ユーザー以外) 用のプライベートとして作成する場合、ログオン・ユーザーにはスタッフ定義権限が必要です。

プロセス・テンプレート・リストは、同じプロパティーのプロセス・テンプレートのセットをグループ化します。それらのプロパティーは、フィルターを使って定義されます。リストに含めるプロセス・テンプレートの数は、クライアントに戻されるプロセス・テンプレートの最大数を指定するしきい値を使って制限できます。そのしきい値は、指定したソート基準に従ってプロセス・テンプレート・リストがソートされた後で適用されます。プロセス・テンプレートはサーバーでソートされることに注意してください。つまり、サーバーのコード・ページでソート順序が決まります。

プロセス・テンプレート・リストの名前を指定する場合には、以下の規則が適用されます。

- 最大 32 文字 (バイト) を指定できます。
- 現行のロケールに応じて、以下のものを除くあらゆる印刷可能文字を使用できます。

  * ? " ; : .
- ブランクも使用できますが、先行ブランクを使わない、末尾ブランクを使わない、複数のブランクを連続して使わない、という制限があります。

記述を指定する場合には、以下の規則が適用されます。

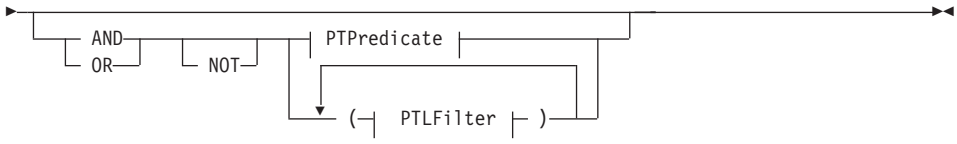
- 最大 254 文字 (バイト) を指定できます。
- 現行のロケールに応じて、復帰文字と改行文字を含むあらゆる印刷可能文字を使用できます。

プロセス・テンプレート・リストのフィルターは、プロセス・テンプレートのフィルターからなる文字列として指定します (981ページの『付録A. 構文図の読み方』を参照)。

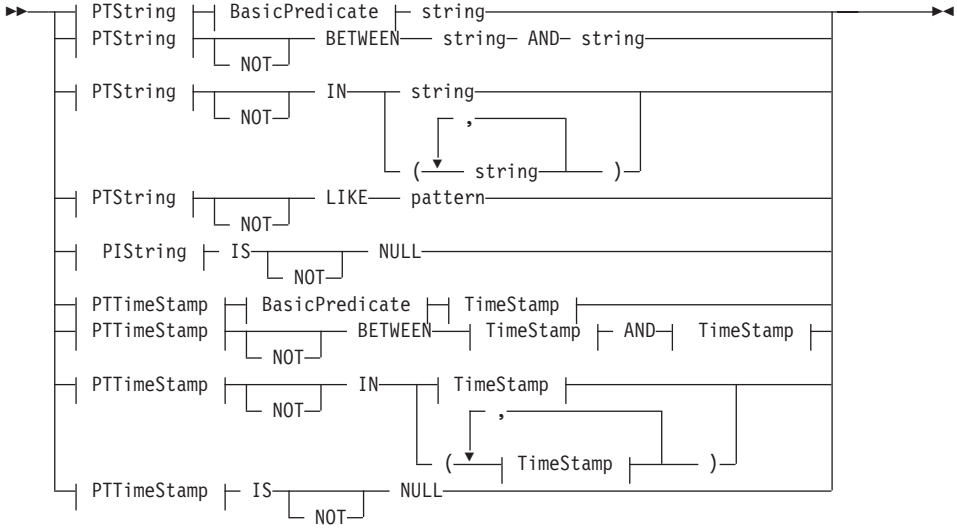
注:

1. *string* (文字列) 定数は、単一引用符 (') で囲む必要があります。
2. ストリング定数内の単一引用符は二重にする ('') 必要があります。
3. *pattern* (パターン) は文字列定数であり、その中のアスタリスクおよび疑問符には特別な意味があります。
   - 疑問符 (?) は任意の単一文字を表します。
   - アスタリスク (*) は、0 文字以上の文字列を表します。
   - エスケープ文字は円記号 (¥) であり、パターン自体に実際の疑問符やアスタリスクが含まれる場合にはこれを使う必要があります。実際の円記号には二重にする必要があります (¥¥)。
4. *TimeStamp* は現地時間に基づいた 24 時間表示のストリング定数です。
5. *TimeStamp* 内のオプションを指定しない場合は、0 (ゼロ) に設定されます。
6. LIKE オペランドのパターンに混合データが含まれている場合は、そのパターンの中にパーセント記号 (%) または下線 (_) を指定することができません。これらのいずれかの文字を使用すると、SQL エラーが戻されます。

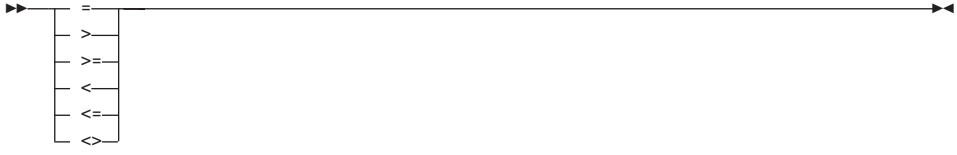
PTLFilter



PTPredicate



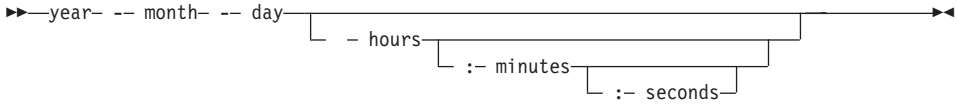
BasicPredicate



PTString

PTTimeStamp

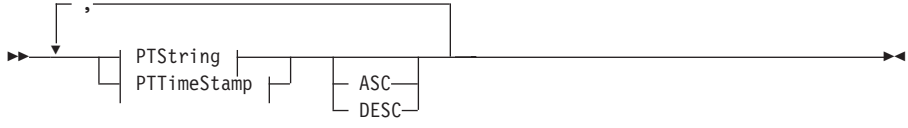
TimeStamp



プロセス・テンプレート・リストのソート基準は文字列として指定されます。

注: 省略時のソート順は昇順 (ASC) です。

PTLOrderBy



使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし、またはスタッフ定義あるいはシステム管理者

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long CreateProcessTemplateList(
       BSTR                                name,
       long                                type,
       BSTR                                owner,
       boolean                             ownerIsNull,
       BSTR                                description,
       boolean                             descriptionIsNull,
       BSTR                                filter,
       boolean                             filterIsNull,
       BSTR                                sortCriteria,
       boolean                             sortCriteriaIsNull,
       long                                threshold,
       boolean                             thresholdIsNull )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceCreateProcessTemplateList(
       FmcjExecutionServiceHandle          service,
       char const *                        name,
       enum FmcjPersistentListTypeOfList   type,
       char const *                        owner,
       char const *                        description,
       char const *                        filter,
       char const *                        sortCriteria,
       unsigned long *                     threshold,
       FmcjProcessTemplateListHandle *     newList )
```

C++ 言語のシグニチャー

```
APIRET CreateProcessTemplateList(
       string const &                  name,
       FmcjPersistentList::TypeOfList  type,
       string const *                  owner,
       string const *                  description,
       string const *                  filter,
       string const *                  sortCriteria,
       unsigned long const *           threshold,
       FmcjProcessTemplateList &       newList ) const
```

Java のシグニチャー

```
public abstract
ProcessTemplateList createProcessTemplateList(
       String                          name,
       TypeOfList                      type,
       String                          owner,
       String                          description,
       String                          filter,
       String                          sortCriteria,
       Integer                         threshold   ) throws FmcException
```

パラメーター

description 入力。プロセス・テンプレート・リストに関するユーザー定義の記述。

descriptionIsNull
入力。記述がリストに設定されているかどうかを示します。

filter 入力。プロセス・テンプレート・リスト内のプロセス・テンプレートを特徴付けるフィルター基準。

filterIsNull 入力。フィルターがリストに設定されているかどうかを示します。

name 入力。プロセス・テンプレート・リスト用にユーザーが定義する名前。

newList 入出力。新しく作成されるプロセス・テンプレート・リスト。

owner 入力。プライベート・タイプの場合のリストの所有者。パブリック用リストの場合は無視。

ownerIsNull 入力。リスト所有者が設定されているかどうかを示します。パブリック・リストには所有者は必要ありません。

service 入力。実行サーバーとのセッションを表すサービス・オブジェクトに対するハンドル。

sortCriteria 入力。プロセス・テンプレート・リスト内のプロセス・テンプレートに適用されるソート基準。

sortCriteriaIsNull
入力。ソート基準がリストに設定されているかどうかを示します。

threshold 入力。プロセス・テンプレート・リスト内のプロセス・テンプレートの最大数を定義するしきい値。

thresholdIsNull
入力。リストのためのしきい値が設定されているかどうかを示します。

type 入力。プライベート用リストを作成するか、パブリック用リストを作成するかを示します。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ProcessTemplateList
新しく作成されるプロセス・テンプレート・リスト。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_DESCRIPTION(810)
指定した記述は無効です。

FMC_ERROR_INVALID_FILTER(125)
指定したフィルターは無効です。

FMC_ERROR_INVALID_LIST_TYPE(813)
指定したリスト・タイプは無効です。

FMC_ERROR_INVALID_NAME(134)
指定したプロセス・テンプレート・リスト名は構文規則に従っていません。

FMC_ERROR_INVALID_USER(132)
リストの所有者に指定したユーザー ID は構文規則に従っていません。

FMC_ERROR_INVALID_SORT(808)
指定したソート基準は無効です。

FMC_ERROR_INVALID_THRESHOLD(807)
指定したしきい値は無効です (可能な最大値を超えています)。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_NOT_AUTHORIZED(119)
許可がありません。

FMC_ERROR_OWNER_NOT_FOUND(812)
プロセス・テンプレート・リストの所有者になる人が見つかりません。

FMC_ERROR_NOT_UNIQUE(121)
プロセス・テンプレート・リストの名前が、指定されたタイプ内で固有ではありません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- ActiveX の例については、823ページの『プロセス・インスタンス・リストの作成 (ActiveX)』を参照してください。
- C 言語の例については、824ページの『プロセス・インスタンス・リストの作成 (C 言語)』を参照してください。
- C++ の例については、825ページの『プロセス・インスタンス・リストの作成 (C++)』を参照してください。
- Java の例については、826ページの『プロセス・インスタンス・リストの作成 (Java)』を参照してください。

CreateWorklist()

この関数 / メソッドは、MQSeries Workflow 実行サーバーにワークリストを作成し、作業項目または通知をユーザーの好みに応じて、またはユーザー・グループの必要に応じてグループ化できるようにします (アクション呼び出し)。

ワークリストは、以下のものによって識別されます。

- 名前 (タイプごとに固有)。
- タイプ (そのリストがパブリック用かプライベート用かを示す標識)。
- 所有者 (タイプがプライベートである場合のリストの所有者)。

リストがパブリックである場合、所有者の指定は無視されます。リストがプライベートで所有者が指定されていない場合、ログオン・ユーザー用にリストが作成されます。

ワークリストをパブリックとして、または別のユーザー (ログオン・ユーザー以外) 用のプライベートとして作成する場合、ログオン・ユーザーにはスタッフ定義権限が必要です。

ワークリストは、同じプロパティーの作業項目または通知のセットをグループ化します。それらのプロパティーは、フィルターを使って定義されます。ワークリストに含めるアイテムの数は、クライアントに戻されるアイテムの最大数を指定するしきい値を使って制限できます。そのしきい値は、指定したソート基準に従ってワークリストがソートされた後で適用されます。アイテムはサーバーでソートされることに注意してください。つまり、サーバーのコード・ページでソート順序が決まります。

ワークリストの名前を指定する場合には、以下の規則が適用されます。

- 最大 32 文字 (バイト) を指定できます。
- 現行のロケールに応じて、以下のものを除くあらゆる印刷可能文字を使用できます。

  `* ? " ; : .`
- ブランクも使用できますが、先行ブランクを使わない、末尾ブランクを使わない、複数のブランクを連続して使わない、という制限があります。

記述を指定する場合には、以下の規則が適用されます。

- 最大 254 文字 (バイト) を指定できます。
- 現行のロケールに応じて、復帰文字と改行文字を含むあらゆる印刷可能文字を使用できます。

ワークリストのフィルターは、ワークリスト中のアイテムに対するフィルター文字列として指定します (981ページの『付録A. 構文図の読み方』を参照)。

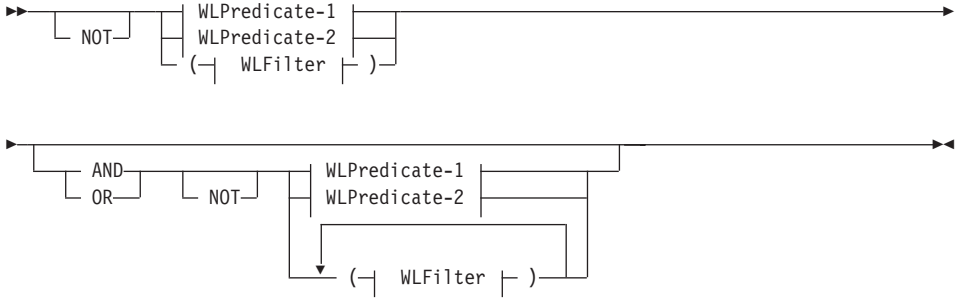
注:

1. *string* (文字列) 定数は、単一引用符 (') で囲む必要があります。
2. ストリング定数内の単一引用符は二重にする (’’) 必要があります。
3. *pattern* (パターン) は文字列定数であり、その中のアスタリスクおよび疑問符には特別な意味があります。
   - 疑問符 (?) は任意の単一文字を表します。
   - アスタリスク (*) は、0 文字以上の文字列を表します。

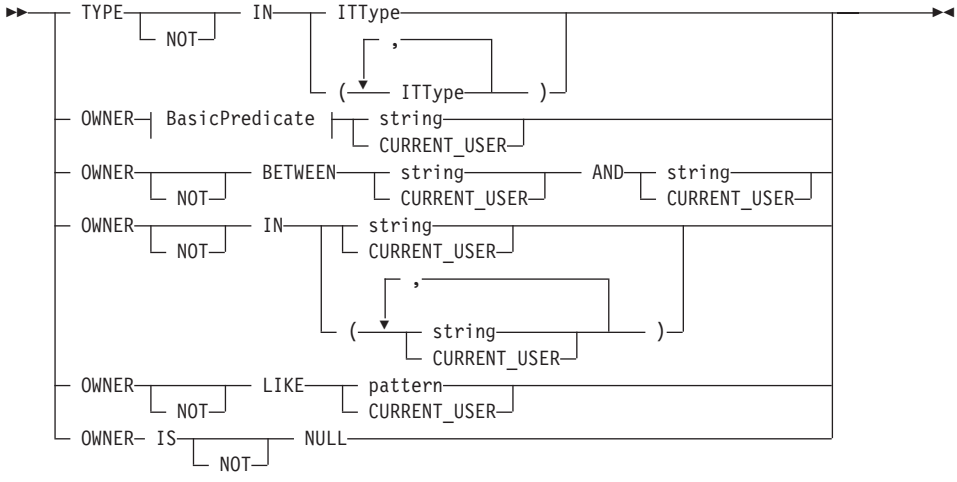
- エスケープ文字は円記号 (¥) であり、パターン自体に実際の疑問符やアスタリスクが含まれる場合にはこれを使う必要があります。実際の円記号には二重にする必要があります (¥¥)。

4. *TimeStamp* は現地時間に基づいた 24 時間表示のストリング定数です。
5. *TimeStamp* 内のオプションを指定しない場合は、0 (ゼロ) に設定されます。
6. LIKE オペランドのパターンに混合データが含まれている場合は、そのパターンの中にパーセント記号 (%) または下線 (_) を指定することができません。これらのいずれかの文字を使用すると、SQL エラーが戻されます。

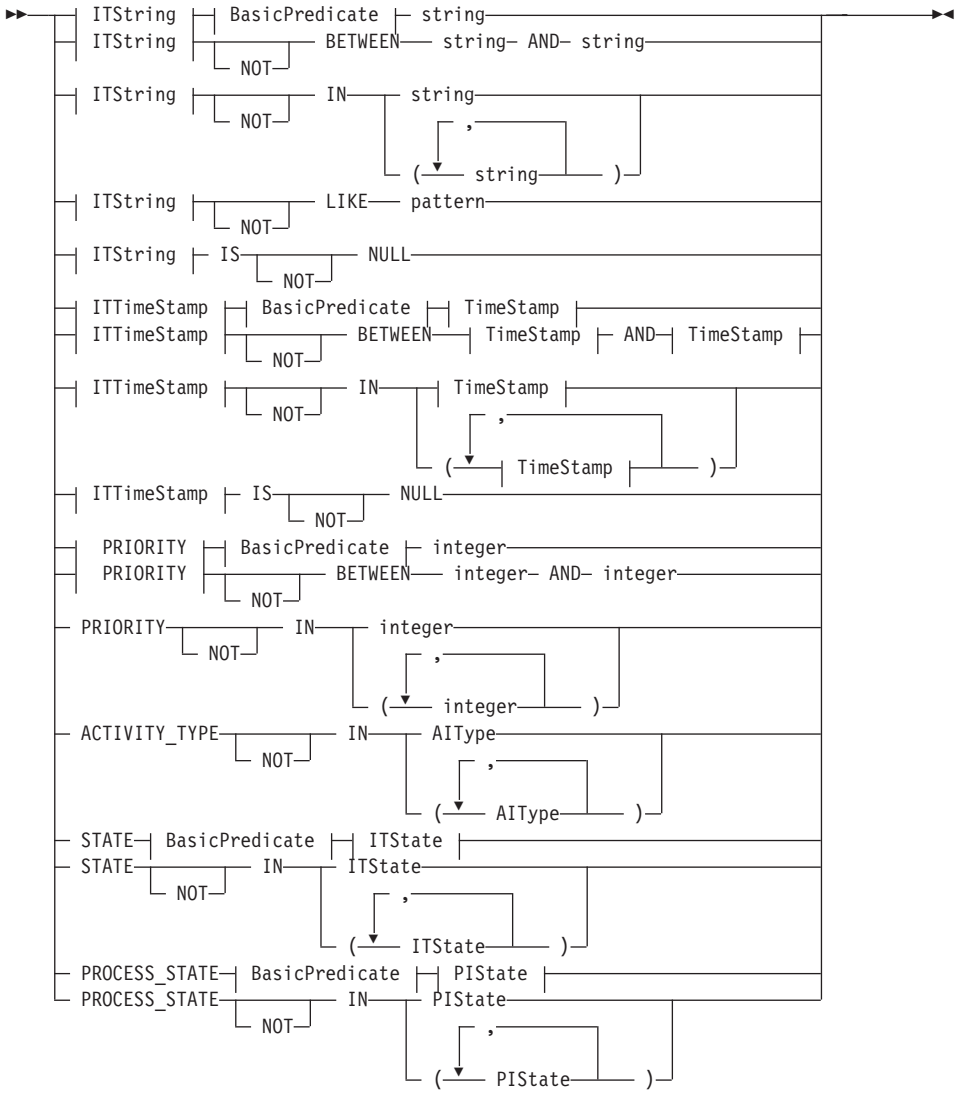
WLFilter



WLPredicate-1



WLPredicate-2



AIType

BasicPredicate

ITState

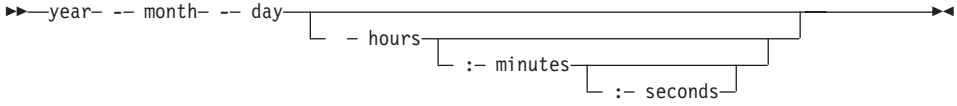
ITString

ITTimeStamp

ITType

PIState

TimeStamp



ワークリストのソート基準は文字列として指定されます。

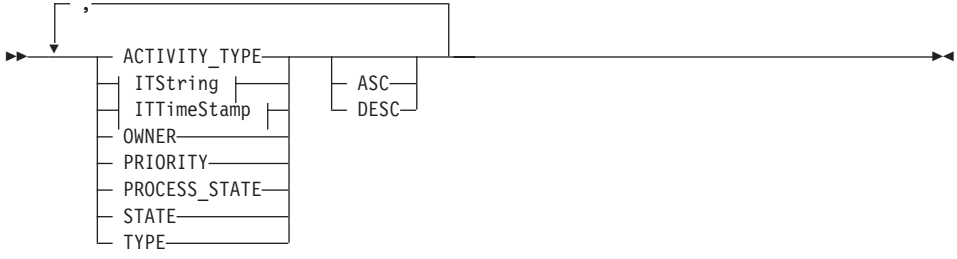
注: 省略時のソート順は昇順 (ASC) です。

活動タイプは、AIType の図で示されている順序でソートされます。

アイテム・タイプは、ITType の図で示されている順序でソートされます。

状態は、ITState または PIState の図で示されている順序でソートされます。

WLOrderBy



使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし、またはスタッフ定義あるいはシステム管理者

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long CreateWorklist(
       BSTR                              name,
       long                              type,
       BSTR                              owner,
       boolean                           ownerIsNull,
       BSTR                              description,
       boolean                           descriptionIsNull,
       BSTR                              filter,
       boolean                           filterIsNull,
       BSTR                              sortCriteria,
       boolean                           sortCriteriaIsNull,
       long                              threshold,
       boolean                           thresholdIsNull )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceCreateWorklist(
       FmcjExecutionServiceHandle        service,
       char const *                      name,
       enum FmcjPersistentListTypeOfList type,
       char const *                      owner,
       char const *                      description,
       char const *                      filter,
       char const *                      sortCriteria,
       unsigned long *                   threshold,
       FmcjWorklistHandle *              newList )
```

C++ 言語のシグニチャー

```
APIRET CreateWorklist(
       string const &                   name,
       FmcjPersistentList::TypeOfList   type,
       string const *                   owner,
       string const *                   description,
       string const *                   filter,
       string const *                   sortCriteria,
       unsigned long const *            threshold,
       FmcjWorklist &                   newList ) const
```

Java のシグニチャー

```
public abstract
WorkList createWorkList(
       String                           name,
       TypeOfList                       type,
       String                           owner,
       String                           description,
       String                           filter,
       String                           sortCriteria,
       Integer                          threshold   ) throws FmcException
```

パラメーター

description 入力。ワークリストに関するユーザー定義の記述。

descriptionIsNull
入力。記述がリストに設定されているかどうかを示します。

filter 入力。ワークリスト内のアイテムを特徴付けるフィルター基準。

filterIsNull 入力。フィルターがリストに設定されているかどうかを示します。

name 入力。ワークリスト用にユーザーが定義する名前。

newList 入出力。新しく作成されるワークリスト。

owner 入力。プライベート・タイプの場合のリストの所有者。パブリック用リストの場合は無視。

ownerIsNull 入力。リスト所有者が設定されているかどうかを示します。パブリック・リストには所有者は必要ありません。

service 入力。実行サーバーとのセッションを表すサービス・オブジェクトに対するハンドル。

sortCriteria 入力。ワークリスト内のアイテムに適用されるソート基準。

sortCriteriaIsNull
入力。ソート基準がリストに設定されているかどうかを示します。

threshold 入力。ワークリスト内のアイテムの最大数を定義するしきい値。

thresholdIsNull
入力。リストのためのしきい値が設定されているかどうかを示します。

type 入力。プライベート用リストを作成するか、パブリック用リストを作成するかを示します。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

WorkList 新しく作成されるワークリスト。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_DESCRIPTION(810)
指定した記述は無効です。

FMC_ERROR_INVALID_FILTER(125)
指定したフィルターは無効です。

FMC_ERROR_INVALID_LIST_TYPE(813)
指定したリスト・タイプは無効です。

FMC_ERROR_INVALID_NAME(134)
指定したワークリスト名は構文規則に従っていません。

FMC_ERROR_INVALID_USER(132)
リストの所有者に指定したユーザー ID は構文規則に従っていません。

FMC_ERROR_INVALID_SORT(808)
指定したソート基準は無効です。

FMC_ERROR_INVALID_THRESHOLD(807)
指定したしきい値は無効です (可能な最大値を超えています)。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_NOT_AUTHORIZED(119)
許可がありません。

FMC_ERROR_OWNER_NOT_FOUND(812)
ワークリストの所有者になる人が見つかりません。

FMC_ERROR_NOT_UNIQUE(121)
ワークリストの名前が、指定されたタイプ内で固有ではありません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- ActiveX の例については、823ページの『プロセス・インスタンス・リストの作成 (ActiveX)』を参照してください。
- C 言語の例については、824ページの『プロセス・インスタンス・リストの作成 (C 言語)』を参照してください。
- C++ の例については、825ページの『プロセス・インスタンス・リストの作成 (C++)』を参照してください。
- Java の例については、826ページの『プロセス・インスタンス・リストの作成 (Java)』を参照してください。

Logoff()

この関数 / メソッドを使用することによりアプリケーションは、MQSeries Workflow 実行サーバーとのユーザー・セッションのうち指定したものを終了することができます (アクション呼び出し)。

正常にログオフが実行された場合は、それ以降、この実行サービス・オブジェクトを使ってクライアント / サーバー呼び出しをしても受け入れられません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long Logoff()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceLogoff(
       FmcjExecutionServiceHandle service )
```

C++ 言語のシグニチャー

```
APIRET Logoff()
```

Java のシグニチャー

```
public abstract
void   logoff() throws FmcException
```

パラメーター

service 入力。実行サーバーとのセッションを表すサービス・オブジェクトに対するハンドル。

戻り値のデータ型

long/ APIRET

この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)

パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)

提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_NOT_LOGGED_ON(106)

ログオンしていません。

FMC_ERROR_COMMUNICATION(13)

指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)

MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)

内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)

戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

例については、817ページの『第9部 例およびシナリオ』を参照してください。

Logon()

この関数 / メソッドを使用することによりアプリケーションは、MQSeries Workflow 実行サーバーとのユーザー・セッションを確立することができます (アクション呼び出し)。

Logon() が正常に完了しない限り、MQSeries Workflow API のその他のアクションやプログラム実行管理関数 / メソッドを使用することはできません。

ログオンするためのユーザー ID は、登録済みの MQSeries Workflow ユーザーでなければなりません。

実行サーバーで*統一ログオン* がサポートされている場合は、空のパスワードとユーザー ID を指定してもかまいません。その場合、ログオンするためのユーザー ID がオペレーティング・システムから検索されます。つまりクライアントでログオンはすでに実行されていなければなりません。クライアントは信用されているので、実行サーバーはパスワード検査を実行しません。

ログオンが正常に実行されると、実行サービス・オブジェクトは単一のユーザー・セッションを表すことになります。それ以降、別のユーザー ID を使用してログオンを要求すると、それは拒否されます。しかし、同じユーザーであっても、別の実行サービス・オブジェクト (各セッションごとに 1 つ) を使用することによって、必要な数のセッションを確立することができます。

ログオン時に、操作のモードを指定することができます。*在席 (present)* セッション・モードで操作している場合、実行サーバーはユーザーが (たとえばコンテナー・データなどを求める) アクティビティー・インプリメンテーションからの要求に応答できることを想定することが可能です。これによって、自動的に開始するアクティビティー・インスタンスのスケジュールを自分の都合に合わせて立てることができます (ただし、あわせてプログラム実行エージェントを開始する必要があります)。

さらに、*在席* モードは、実行サーバーが出した (push した) 非送信請求メッセージをそのセッションで処理できる旨を、MQSeries Workflow に対して示します。追加の前提条件については、20ページの『プッシュ・データのアクセス・モデル』を参照してください。

在席セッションは、各ユーザーにつき 1 つだけ確立できます。他の在席セッションを強制的にログオフして、その代わりに新しく在席セッションを確立するには、*present here* オプションを使用することができます。また、在席セッション・モードを使用するには、プログラム実行エージェントをシャットダウンする必要があることにも注意してください。

省略時 (default) セッション・モードで操作している場合、実行サーバーは応答があることを想定しません。アクティビティー・インスタンスは自動的には開始せず、メッセージはプッシュ (push) されません。*省略時* セッション・モードでは、1 ユーザーに対して複数のセッションが可能です。

セッション・モードを指定するために、以下の列挙型を使用できます。

ActiveX SessionMode

C 言語 FmcjServiceSessionMode

C++ FmcjService::SessionMode

JAVA com.ibm.workflow.api.ServicePackage.SessionMode

列挙型定数には以下の値を使用できます。可能な限り整数値ではなく記号名を使うようにしてください。

Default 省略時 (非在席) セッション・モードで操作することを示します。

- **ActiveX** SessionMode_Default
- **C 言語** Fmc_SM_Default
- **C++** FmcjService::Default respectively FmcjExecutionService::Default
- **JAVA** SessionMode.DEFAULT

Present 在席セッション・モードで操作することを示します。

- **ActiveX** SessionMode_Present
- **C 言語** Fmc_SM_Present
- **C++** FmcjService::Present respectively FmcjExecutionService::Present

JAVA SessionMode.PRESENT

PresentHere 在席セッション・モードで操作することを示します。すでに在席セッション・モードのセッションが存在していれば、そのセッションはログオフされます。

ActiveX SessionMode_PresentHere

C 言語 Fmc_SM_PresentHere

C++ FmcjService::PresentHere respectively
FmcjExecutionService::PresentHere

JAVA SessionMode.PRESENT_HERE

ログオン時には、不在モードに設定している場合に戻っているかどうかを指定することもできます。不在でない場合には、作業の割り当てに参加します。そうでなければ、作業項目は 1 つも割り当てられません。

不在を設定するために、以下の列挙型を使用できます。

ActiveX AbsenceIndicator

C 言語 FmcjServiceAbsenceIndicator

C++ FmcjService::AbsenceIndicator

JAVA com.ibm.workflow.api.ServicePackage.AbsenceIndicator

列挙型定数には以下の値を使用できます。可能な限り整数値ではなく記号名を使うようにしてください。

NotSet 値が指定されていないことを示します。この場合、個人レコードの定義が適用されます。不在はリセットされるか、またはここで指定した定義に合わないものになります。

ActiveX AbsenceIndicator_NotSet

C 言語 Fmc_SA_NotSet

C++ FmcjService::NotSet respectively
FmcjExecutionService::NotSet

JAVA AbsenceIndicator.NOT_SET

Reset 不在設定をリセットすることを示します。つまり席に戻っているということです。

ActiveX AbsenceIndicator_Reset

C 言語 Fmc_SA_Reset

	C++	FmcjService::Reset respectively FmcjExecutionService::Reset
	JAVA	AbsenceIndicator.RESET
Leave	これは不在設定を現状のままにしておくことを示します。	
	ActiveX	AbsenceIndicator_Leave
	C 言語	Fmc_SA_Leave
	C++	FmcjService::Leave respectively FmcjExecutionService::Leave
	JAVA	AbsenceIndicator.LEAVE

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

登録済み MQSeries Workflow ユーザーであること

必要な接続

なし

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long Logon           ( BSTR userID,     BSTR password )
long LogonWithOptions( BSTR             userID,
                       BSTR             password,
                       SessionMode      sessionMode,
                       AbsenceIndicator absenceIndicator )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceLogon (
       FmcjExecutionServiceHandle  service,
       char const *                      userID,
       char const *                      password,
       enum FmcjServiceSessionMode       sessionMode,
       enum FmcjServiceAbsenceIndicator  absenceIndicator )
```

C++ 言語のシグニチャー

```
APIRET Logon( string const & userID, string const & password )
APIRET Logon(
       string const &                    userID,
       string const &                    password,
       SessionMode                       sessionMode = Present,
       AbsenceIndicator                  absenceIndicator = NotSet )
```

Java のシグニチャー

```
public abstract
void   logon ( String userID,   String password )
public abstract
void   logon2( String           userID,
               String           password,
               SessionMode      sessionMode,
               AbsenceIndicator absenceIndicator ) throws FmcException
```

パラメーター

absenceIndicator
入力。不在セットを操作する方法を示す標識。

password
入力。ユーザーのパスワード。一元化ログオンでは入力する必要はありません。

service
入力。実行サーバーとの間で確立するセッションを表すサービス・オブジェクトに対するハンドル。

sessionMode
入力。確立するセッションのモード。

userID
入力。ログオンの対象となるユーザーのユーザー ID。一元化ログオンでは入力する必要はありません。

戻り値のデータ型

long/ APIRET

この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)

パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)

提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_ALREADY_LOGGED_ON(11)

ユーザーがすでに在席モードでログオンしているか、または実行サービス・オブジェクトはすでに別のユーザー・セッションを表しています。

FMC_ERROR_BACK_LEVEL_VERSION(504)

クライアントのバージョンが古いため、このサーバーではサポートされません。

FMC_ERROR_INVALID_ABSENCE_SPEC(905)

無効な不在設定が指定されています。

FMC_ERROR_INVALID_SESSION_MODE(901)

無効なセッション・モードが指定されています。

FMC_ERROR_NEWER_VERSION(505)

クライアントのバージョンがサーバーのバージョンよりも新しいため、サポートされません。

FMC_ERROR_PASSWORD(12)

パスワードが間違っています。

FMC_ERROR_PROFILE(124)

必要なユーザーまたはワークステーション・プロファイルの項目が見つかりません。

FMC_ERROR_USERID_UNKNOWN(10)

MQSeries Workflow にユーザー ID が登録されていません。

FMC_ERROR_COMMUNICATION(13)

指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)

MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

例については、817ページの『第9部 例およびシナリオ』を参照してください。

Passthrough()

この関数 / メソッドは、このプログラム内から MQSeries Workflow 実行サーバーとのユーザー・セッションを確立するために、アクティビティー・インプリメンテーションによって使われます (アクティビティー・インプリメンテーション呼び出し)。

正常に実行されると、同じ実行サーバーとのセッションは、このプログラムによってインプリメントされた作業項目が開始した場所からセットアップされます。セッションがセットアップされるユーザーと、作業項目が開始されたユーザーは同じです。

使用上の注意

- 141ページの『アクティビティー・インプリメンテーション関数 / メソッド』にある一般的な情報を参照してください。

許可

MQSeries Workflow によって開始されたアクティビティー・インプリメンテーション

必要な接続

なし。ただし MQSeries Workflow プログラム実行エージェントはアクティブでなければなりません。

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long Passthrough()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServicePassthrough(
       FmcjExecutionServiceHandle service )
```

C++ 言語のシグニチャー

```
APIRET Passthrough()
```

Java のシグニチャー

```
public abstract
void passthrough()  throws FmcException
```

パラメーター

service 入力。実行サーバーとの間で確立するセッションを表すサービス・オブジェクトに対するハンドル。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_PROGRAM_EXECUTION(126)
Passthrough がアクティビティー・インプリメンテーション内から呼び出されていないか、プログラム実行エージェントがアクティブではありません。

FMC_ERROR_TOOL_FUNCTION(128)
サポート・ツールからまたはプログラム実行サーバーが開始したプログラムから Passthrough を呼び出すことはできません。

FMC_ERROR_USERID_UNKNOWN(10)
作業項目を開始したユーザーが存在していません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- C 言語の例については、857ページの『実行可能なアクティビティー・インプリメンテーションのプログラミング (C 言語)』を参照してください。
- C++ の例については、858ページの『実行可能なアクティビティー・インプリメンテーションのプログラミング (C++)』を参照してください。

PEAShutDown()

この関数 / メソッドは、ログオンしているユーザーに関連したプログラム実行エージェントをシャットダウンするのに使います (プログラム実行管理関数 / メソッド呼び出し)。

これによって、プログラム実行エージェントはアクティビティー・インプリメンテーションがまだ実行中であるかどうかにかかわらずシャットダウンします。したがって、実行中のすべてのアクティビティー・インプリメンテーションの結果が正しく実行サーバーに渡されるのを待つ必要があります。

使用上の注意

- 147ページの『プログラム実行管理関数 / メソッド』にある一般的な情報を参照してください。

許可

ログオンが必要です。

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long PEAShutDown()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServicePEAShutDown(
       FmcjExecutionServiceHandle service )
```

C++ 言語のシグニチャー

```
APIRET PEAShutDown()
```

Java のシグニチャー

```
public abstract
void programExecutionAgentShutDown() throws FmcException
```

パラメーター

service 入力。ユーザーおよびシャットダウンするプログラム実行エージェントを識別する実行サービスオブジェクトのハンドル。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
ログオンしているユーザーのプログラム実行エージェントが実行されていません。

FMC_ERROR_NOT_AUTHORIZED(119)
アクティビティー・インプリメンテーションまたはサポート・ツールからこの呼び出しを発行する許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

PEAStartUp()

この関数 / メソッドは、ログオンしているユーザーに関連したプログラム実行エージェントを開始するのに使用します (プログラム実行管理呼び出し)。

この呼び出し側アプリケーションが実行されているのと同じノードでプログラム実行エージェントが開始されます。 1 人のユーザーにつき 1 つのプログラム実行エージェントがサポートされます。このセッションまたは他のセッションからのすべてのユーザーの作業は、このプログラム実行エージェントに送信されます。

ユーザーのセッションが終了しても、プログラム実行エージェントが自動的にシャットダウンされることは**ありません**。プログラム実行エージェントはアクティビティー・インプリメンテーションが完了するまで待機可能でなければなりません。

プログラム実行エージェントがすでに別のノードで実行していることが分かれば、シャットダウンして再試行することができます。実行中のすべてのアクティビティー・インプリメンテーションを待つ必要があります。

使用上の注意

- 147ページの『プログラム実行管理関数 / メソッド』にある一般的な情報を参照してください。

許可

有効なユーザー・セッション

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long PEAStartUp()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServicePEAStartUp(
       FmcjExecutionServiceHandle service )
```

C++ 言語のシグニチャー

```
APIRET PEAStartUp()
```

ActiveX のシグニチャー

```
public abstract
void programExecutionAgentStartUp() throws FmcException
```

パラメーター

service 入力。実行サーバーとのセッションを表すサービス・オブジェクトに対するハンドル。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_ALREADY_STARTED(111)
ログオンしているユーザーのプログラム実行エージェントはすでに実行されています。

FMC_ERROR_NOT_AUTHORIZED(119)
アクティビティー・インプリメンテーションまたはサポート・ツールからこの呼び出しを発行する許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

QueryActivityInstanceNotifications()

この関数 / メソッドは、ユーザーがアクセスできるアクティビティー・インスタンス通知を、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

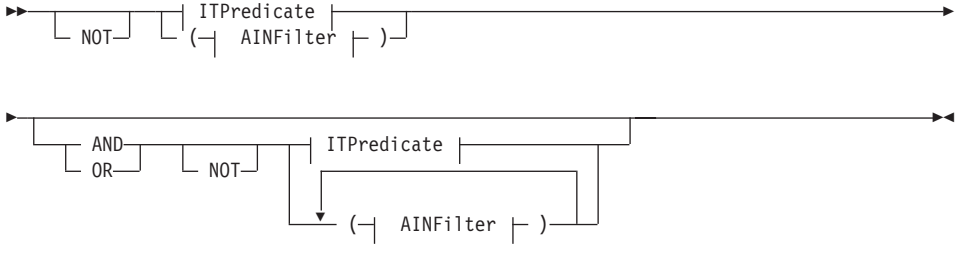
C および C++ では、検索したアクティビティー・インスタンス通知はすべて、提供されるベクトルに追加されます。現在のアクティビティー・インスタンス通知だけを読み取りたい場合、まずベクトルをクリアした後でこの関数 / メソッドを呼び出さなければなりません。つまり、C 言語ではベクトル・ハンドルを 0 に設定し、C++ API ではベクトルのすべての要素を消去する必要があります。

検索するアクティビティー・インスタンス通知は、フィルターによって特徴付けることができます。アクティビティー・インスタンス通知のフィルターは、文字列として指定されます。

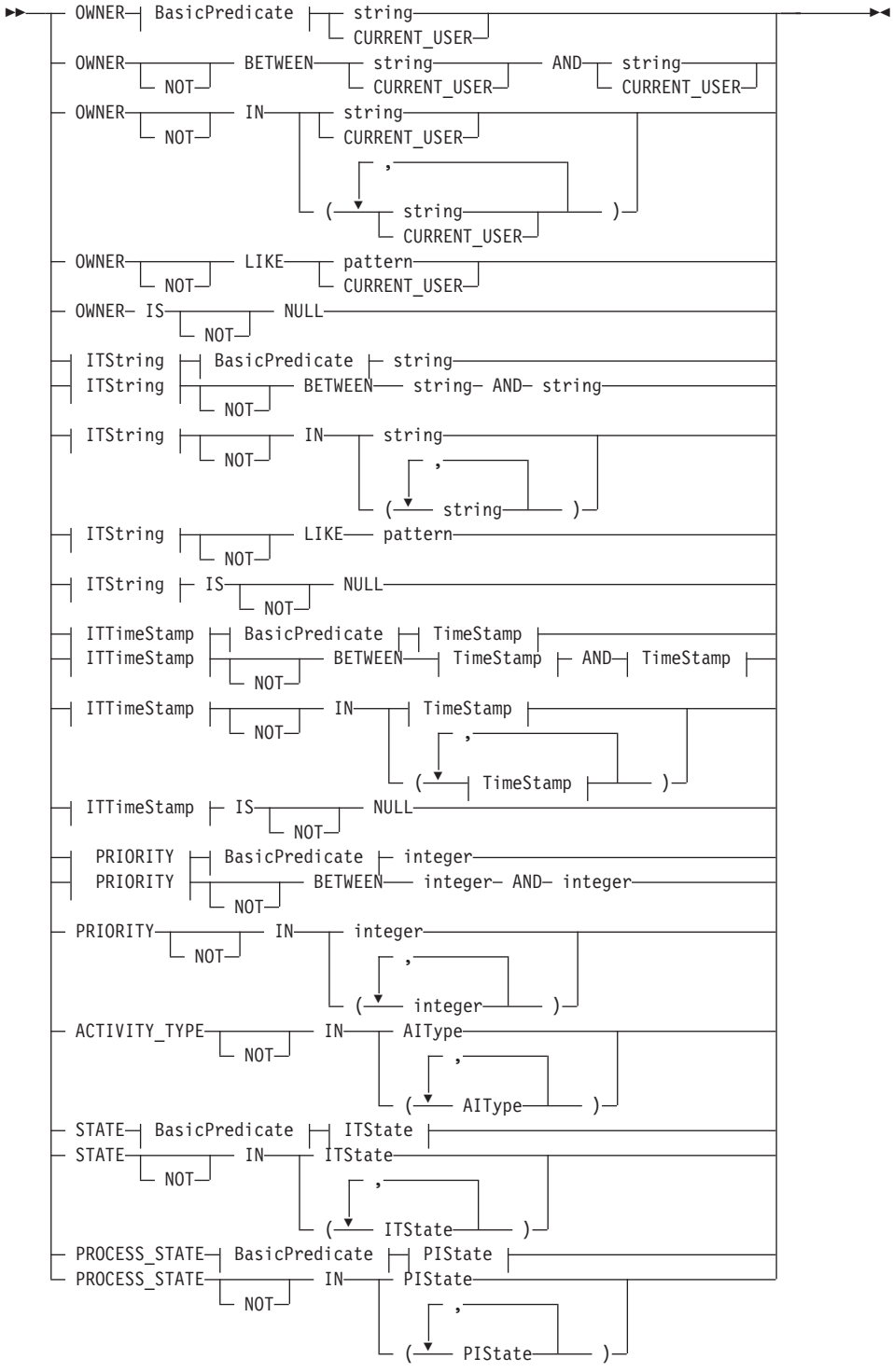
注:

1. *string* (文字列) 定数は、単一引用符 (') で囲む必要があります。
2. ストリング定数内の単一引用符は二重にする ('') 必要があります。
3. *pattern* (パターン) は文字列定数であり、その中のアスタリスクおよび疑問符には特別な意味があります。
   - 疑問符 (?) は任意の単一文字を表します。
   - アスタリスク (*) は、0 文字以上の文字列を表します。
   - エスケープ文字は円記号 (¥) であり、パターン自体に実際の疑問符やアスタリスクが含まれる場合にはこれを使う必要があります。実際の円記号には二重にする必要があります (¥¥)。
4. *TimeStamp* は現地時間に基づいた 24 時間表示のストリング定数です。
5. *TimeStamp* 内のオプションを指定しない場合は、0 (ゼロ) に設定されます。
6. LIKE オペランドのパターンに混合データが含まれている場合は、そのパターンの中にパーセント記号 (%) または下線 (_) を指定することができません。これらのいずれかの文字を使用すると、SQL エラーが戻されます。

AINFilter

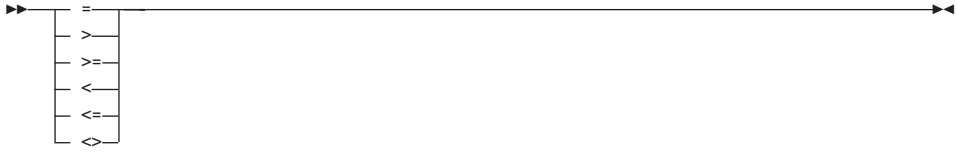


ITPredicate


OWNER
BasicPredicate
string
CURRENT_USER
OWNER
NOT
BETWEEN
string
CURRENT_USER
AND
string
CURRENT_USER
OWNER
NOT
IN
string
CURRENT_USER
,
(
string
CURRENT_USER
)
OWNER
NOT
LIKE
pattern
CURRENT_USER
OWNER
IS
NOT
NULL
ITString
BasicPredicate
string
ITString
NOT
BETWEEN
string
AND
string
ITString
NOT
IN
string
,
(
string
)
ITString
NOT
LIKE
pattern
ITString
IS
NOT
NULL
ITTimeStamp
BasicPredicate
TimeStamp
ITTimeStamp
NOT
BETWEEN
TimeStamp
AND
TimeStamp
ITTimeStamp
NOT
IN
TimeStamp
,
(
TimeStamp
)
ITTimeStamp
IS
NOT
NULL
PRIORITY
BasicPredicate
integer
PRIORITY
NOT
BETWEEN
integer
AND
integer
PRIORITY
NOT
IN
integer
,
(
integer
)
ACTIVITY_TYPE
NOT
IN
AIType
,
(
AIType
)
STATE
BasicPredicate
ITState
STATE
NOT
IN
ITState
,
(
ITState
)
PROCESS_STATE
BasicPredicate
PIState
PROCESS_STATE
NOT
IN
PIState
,
(
PIState
)

AIType

BasicPredicate



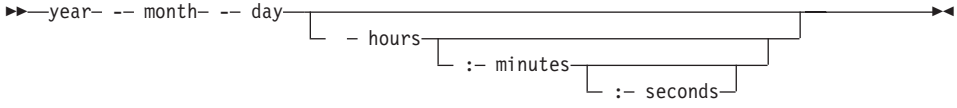
ITState

ITString

ITTimeStamp

PIState

TimeStamp



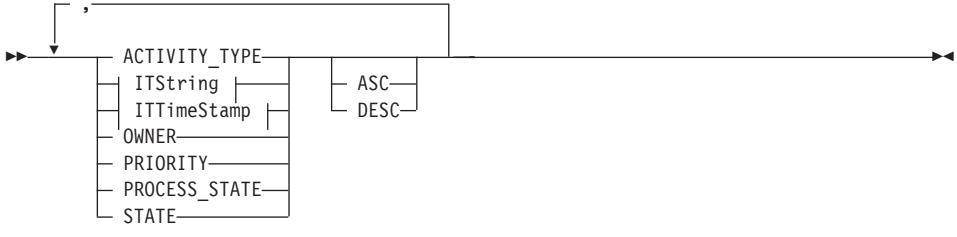
アクティビティー・インスタンス通知は、ソートすることができます。アクティビティー・インスタンス通知のソート基準は文字列として指定されます。

注: 省略時のソート順は昇順 (ASC) です。

活動タイプは、AIType の図で示されている順序でソートされます。

状態は、ITState または PIState の図で示されている順序でソートされます。

AINOrderBy



取り出すアクティビティー・インスタンス通知の数は、クライアントに戻されるアクティビティー・インスタンス通知の最大数を指定するしきい値を使って制限できます。そのしきい値は、指定したソート基準に従ってアクティビティー・インスタンス通知がソートされた後で適用されます。アクティビティー・インスタンス通知はサーバーでソートされることに注意してください。つまり、サーバーのコード・ページでソート順序が決まります。

各アクティビティー・インスタンス通知ごとに取り出される 1 次情報は以下のとおりです。

- ActivityType
- Category
- CreationTime
- Description
- Icon
- Implementation
- Kind
- LastModificationTime
- Name
- Owner
- Priority
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State
- SupportTools

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	サポートなし
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryActivityInstanceNotifications(
       FmcjExecutionServiceHandle                    service,
       char const *                                  filter,
       char const *                                  sortCriteria,
       unsigned long const *                         threshold,
       FmcjActivityInstanceNotificationVectorHandle * notifications )
```

C++ 言語のシグニチャー

```
APIRET QueryActivityInstanceNotifications(
       string const *                              filter,
       string const *                              sortCriteria,
       unsigned long const *                       threshold,
       vector<FmcjActivityInstanceNotification> & notifications ) const
```

Java のシグニチャー

```
public abstract
ActivityInstanceNotification[] queryActivityInstanceNotifications(
       String                                      filter,
       String                                      sortCriteria,
       Integer                                     threshold   )
throws FmcException
```

パラメーター

filter 入力。検索するアクティビティー・インスタンス通知を特徴付けるフィルター基準。

notifications 入出力。アクティビティー・インスタンス通知の適格ベクトル。

service 入力。実行サーバーとのセッションを表すサービス・オブジェクトに対するハンドル。

sortCriteria 入力。見つかったアクティビティー・インスタンス通知に適用されるソート基準。

threshold 入力。クライアントに戻されるアクティビティー・インスタンス通知の最大数を定義するしきい値。

戻り値のデータ型

APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ActivityInstanceNotification[]
適格アクティビティー・インスタンス通知。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_FILTER(125)
指定したフィルターは無効です。

FMC_ERROR_INVALID_SORT(808)
指定したソート基準は無効です。

FMC_ERROR_INVALID_THRESHOLD(807)
指定したしきい値は無効です。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
戻されるアクティビティー・インスタンス通知の数は照会結果に許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_QUERY_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- C 言語の例については、843ページの『プロセス・インスタンスの照会 (C 言語)』を参照してください。
- C++ の例については、844ページの『プロセス・インスタンスの照会 (C++)』を参照してください。
- Java の例については、845ページの『プロセス・インスタンスの照会 (Java)』を参照してください。

QueryItems()

この関数 / メソッドは、ユーザーがアクセスできる作業項目または通知を、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

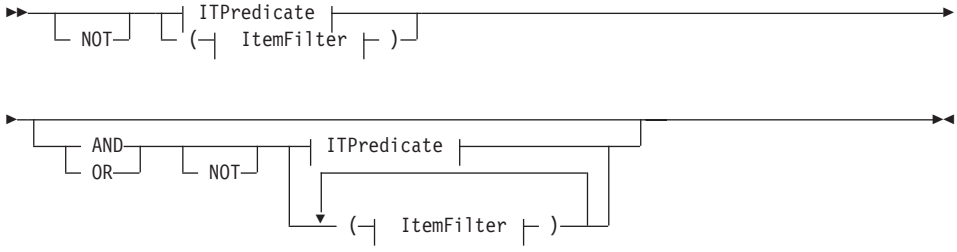
C および C++ では、検索したアイテムはすべて、提供されるベクトルに追加されます。現在のアイテムだけを読み取りたい場合、まずベクトルをクリアした後でこの関数 / メソッドを呼び出さなければなりません。つまり、C 言語ではハンドルを 0 に設定し、C++ API ではベクトルのすべての要素を消去する必要があります。

検索するアイテムは、フィルターによって特徴付けることができます。アイテム・フィルターは、文字列として指定されます。

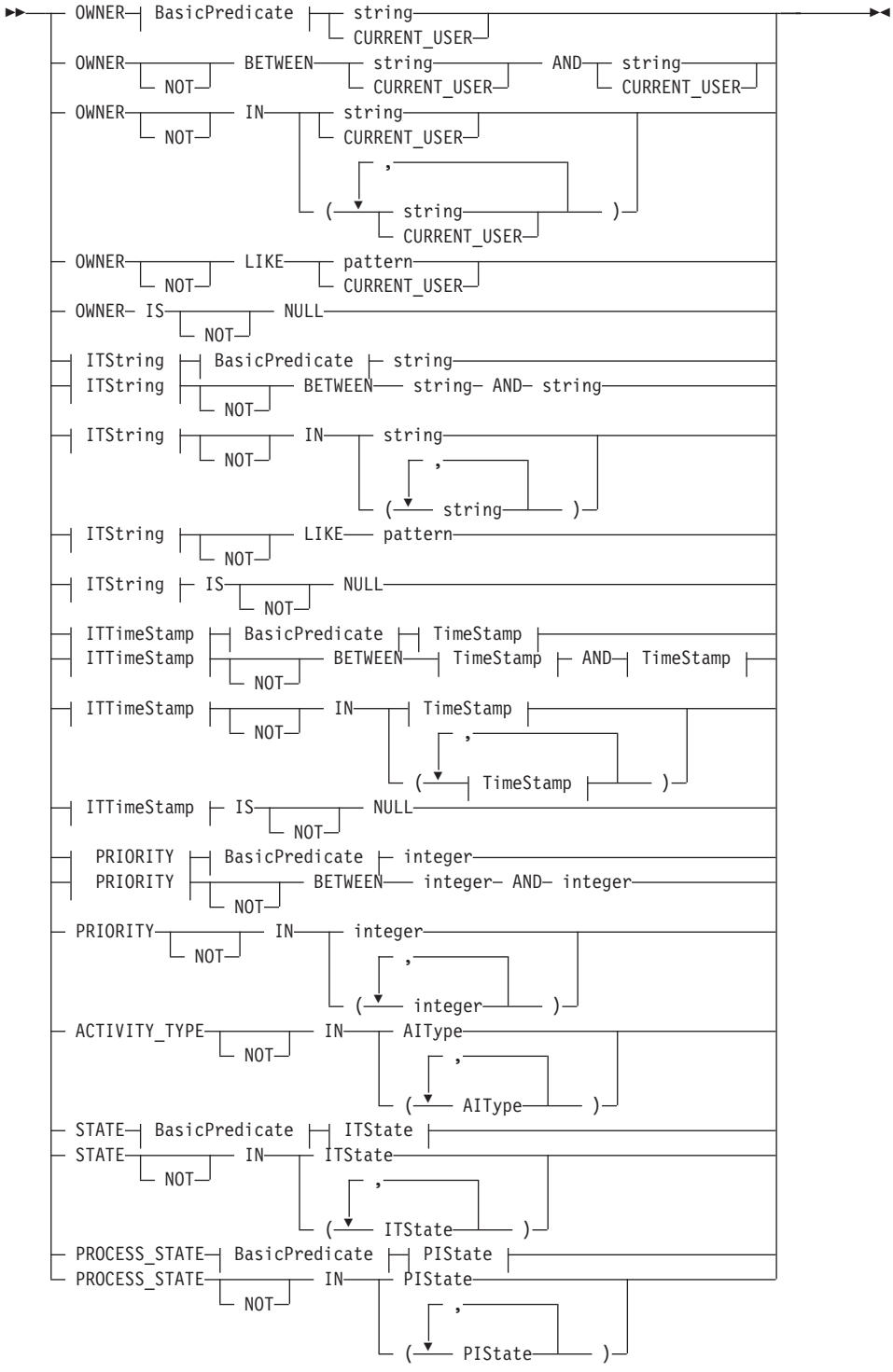
注:

1. *string* (文字列) 定数は、単一引用符 (') で囲む必要があります。
2. ストリング定数内の単一引用符は二重にする ('') 必要があります。
3. *pattern* (パターン) は文字列定数であり、その中のアスタリスクおよび疑問符には特別な意味があります。
   - 疑問符 (?) は任意の単一文字を表します。
   - アスタリスク (*) は、0 文字以上の文字列を表します。
   - エスケープ文字は円記号 (¥) であり、パターン自体に実際の疑問符やアスタリスクが含まれる場合にはこれを使う必要があります。実際の円記号には二重にする必要があります (¥¥)。
4. *TimeStamp* は現地時間に基づいた 24 時間表示のストリング定数です。
5. *TimeStamp* 内のオプションを指定しない場合は、0 (ゼロ) に設定されます。
6. LIKE オペランドのパターンに混合データが含まれている場合は、そのパターンの中にパーセント記号 (%) または下線 (_) を指定することができません。これらのいずれかの文字を使用すると、SQL エラーが戻されます。

ItemFilter

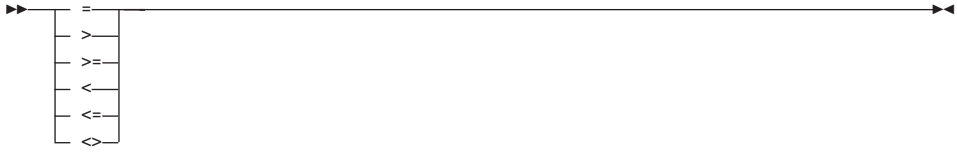


ITPredicate

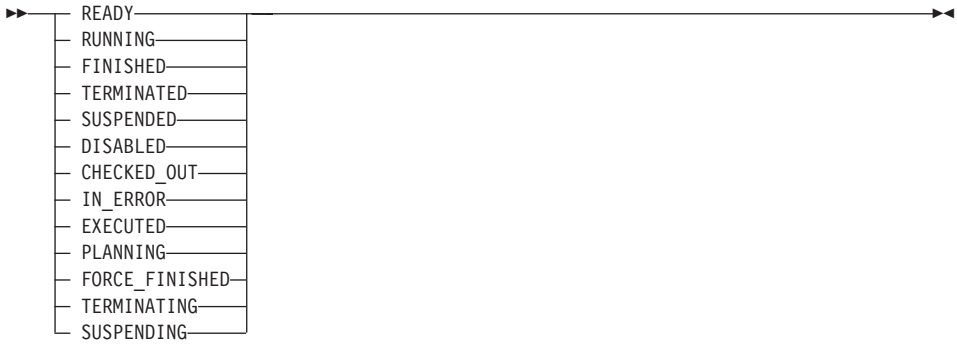

OWNER
BasicPredicate
string
CURRENT_USER
OWNER
NOT
BETWEEN
string
CURRENT_USER
AND
string
CURRENT_USER
OWNER
NOT
IN
string
CURRENT_USER
,
(
string
CURRENT_USER
)
OWNER
NOT
LIKE
pattern
CURRENT_USER
OWNER
IS
NOT
NULL
ITString
BasicPredicate
string
ITString
NOT
BETWEEN
string
AND
string
ITString
NOT
IN
string
,
(
string
)
ITString
NOT
LIKE
pattern
ITString
IS
NOT
NULL
ITTimeStamp
BasicPredicate
TimeStamp
ITTimeStamp
NOT
BETWEEN
TimeStamp
AND
TimeStamp
ITTimeStamp
NOT
IN
TimeStamp
,
(
TimeStamp
)
ITTimeStamp
IS
NOT
NULL
PRIORITY
BasicPredicate
integer
PRIORITY
NOT
BETWEEN
integer
AND
integer
PRIORITY
NOT
IN
integer
,
(
integer
)
ACTIVITY_TYPE
NOT
IN
AIType
,
(
AIType
)
STATE
BasicPredicate
ITState
STATE
NOT
IN
ITState
,
(
ITState
)
PROCESS_STATE
BasicPredicate
PIState
PROCESS_STATE
NOT
IN
PIState
,
(
PIState
)

AIType

BasicPredicate



ITState

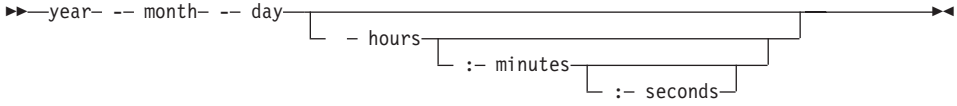


ITString

ITTimeStamp

PIState

TimeStamp



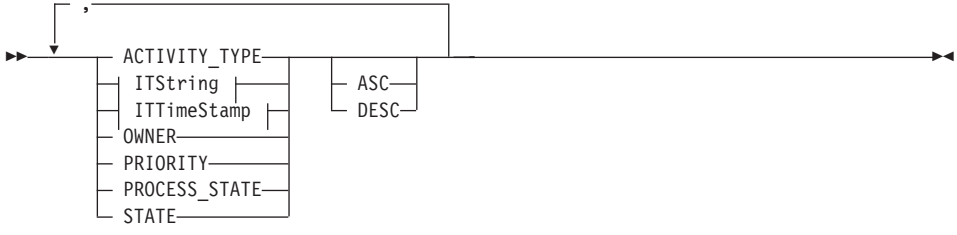
アイテムはソートすることができます。アイテムのソート基準は文字列として指定されます。

注: 省略時のソート順は昇順 (ASC) です。

活動タイプは、AIType の図で示されている順序でソートされます。

状態は、ITState または PIState の図で示されている順序でソートされます。

ItemOrderBy



取り出すアイテムの数は、クライアントに戻されるアイテムの最大数を指定するしきい値を使って品目そのしきい値は、指定したソート基準に従ってアイテムがソートされた後で適用されます。アイテムはサーバーでソートされることに注意してください。つまり、サーバーのコード・ページでソート順序が決まります。

各アイテムごとに取り出される 1 次情報は以下のとおりです。

- ActivityType
- Category
- CreationTime
- Description
- Icon
- Implementation
- Kind
- LastModificationTime
- Name
- Owner
- Priority
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State
- SupportTools

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	サポートなし
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryItems(
       FmcjExecutionServiceHandle  service,
       char const *                filter,
       char const *                sortCriteria,
       unsigned long const *       threshold,
       FmcjItemHandle *            items )
```

C++ 言語のシグニチャー

```
APIRET QueryItems(
       string const *                filter,
       string const *                sortCriteria,
       unsigned long const *         threshold,
       vector<FmcjItem> &          items ) const
```

Java のシグニチャー

```
public abstract
Item[] queryItems(
       String                         filter,
       String                         sortCriteria,
       Integer                        threshold   ) throws FmcException
```

パラメーター

filter 入力。検索するアイテムを特徴付けるフィルター基準。

items 入出力。アイテムの適格ベクトル。

service 入力。実行サーバーとのセッションを表すサービス・オブジェクトに対するハンドル。

sortCriteria 入力。見つかったアイテムに適用されるソート基準。

threshold 入力。クライアントに戻されるアイテムの最大数を定義するしきい値。

戻り値のデータ型

APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

Item[] 適格アイテム。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_FILTER(125)
指定したフィルターは無効です。

FMC_ERROR_INVALID_SORT(808)
指定したソート基準は無効です。

FMC_ERROR_INVALID_THRESHOLD(807)
指定したしきい値は無効です。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
戻されるアイテムの数は照会結果に許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_QUERY_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- C 言語の例については、843ページの『プロセス・インスタンスの照会 (C 言語)』を参照してください。
- C++ の例については、844ページの『プロセス・インスタンスの照会 (C++)』を参照してください。
- Java の例については、845ページの『プロセス・インスタンスの照会 (Java)』を参照してください。

QueryProcessInstanceLists()

この関数 / メソッドは、ユーザーがアクセスできるプロセス・インスタンス・リストを、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

C および C++ では、検索したプロセス・インスタンス・リストはすべて、提供されるベクトルに追加されます。現在のプロセス・インスタンス・リストだけを読み取りたい場合、まずベクトルをクリアした後でこの関数 / メソッドを呼び出さなければなりません。つまり、C 言語ではベクトル・ハンドルを 0 に設定し、C++ API ではベクトルのすべての要素を消去する必要があります。ActiveX の場合、ExecutionService 上のプロセス・インスタンス・リスト配列が更新されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long QueryProcessInstanceLists()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryProcessInstanceLists(
       FmcjExecutionServiceHandle            service,
       FmcjProcessInstanceListVectorHandle * lists )
```

C++ 言語のシグニチャー

```
APIRET QueryProcessInstanceLists(
       vector<FmcjProcessInstanceList> & lists ) const
```

Java のシグニチャー

```
public abstract
ProcessInstanceList[] queryProcessInstanceLists() throws FmcException
```

パラメーター

lists 入出力。プロセス・インスタンス・リストのベクトル。

service 入力。実行サーバーとのセッションを表すサービス・オブジェクトに対するハンドル。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ProcessInstanceList[]
適格プロセス・インスタンス・リスト。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
戻されるプロセス・インスタンス・リストの数は照会結果に許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_QUERY_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- ActiveX の例については、832ページの『ワークリストの照会 (ActiveX)』を参照してください。
- C 言語の例については、833ページの『ワークリストの照会 (C 言語)』を参照してください。
- C++ の例については、835ページの『ワークリストの照会 (C++)』を参照してください。
- Java の例については、837ページの『ワークリストの照会 (Java)』を参照してください。

QueryProcessInstanceNotifications()

この関数 / メソッドは、ユーザーがアクセスできるプロセス・インスタンス通知を、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

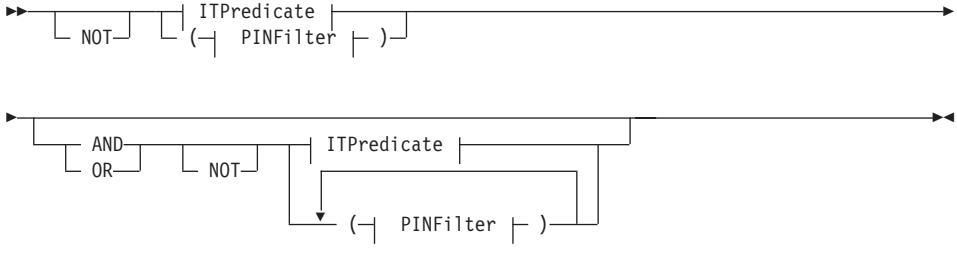
C および C++ では、検索したプロセス・インスタンス通知はすべて、提供されるベクトルに追加されます。現在のプロセス・インスタンス通知だけを読み取りたい場合、まずベクトルをクリアした後でこの関数 / メソッドを呼び出さなければなりません。つまり、C 言語ではベクトル・ハンドルを 0 に設定し、C++ API ではベクトルのすべての要素を消去する必要があります。

検索するプロセス・インスタンス通知は、フィルターによって特徴付けることができます。プロセス・インスタンス通知フィルターは文字列として指定されます。

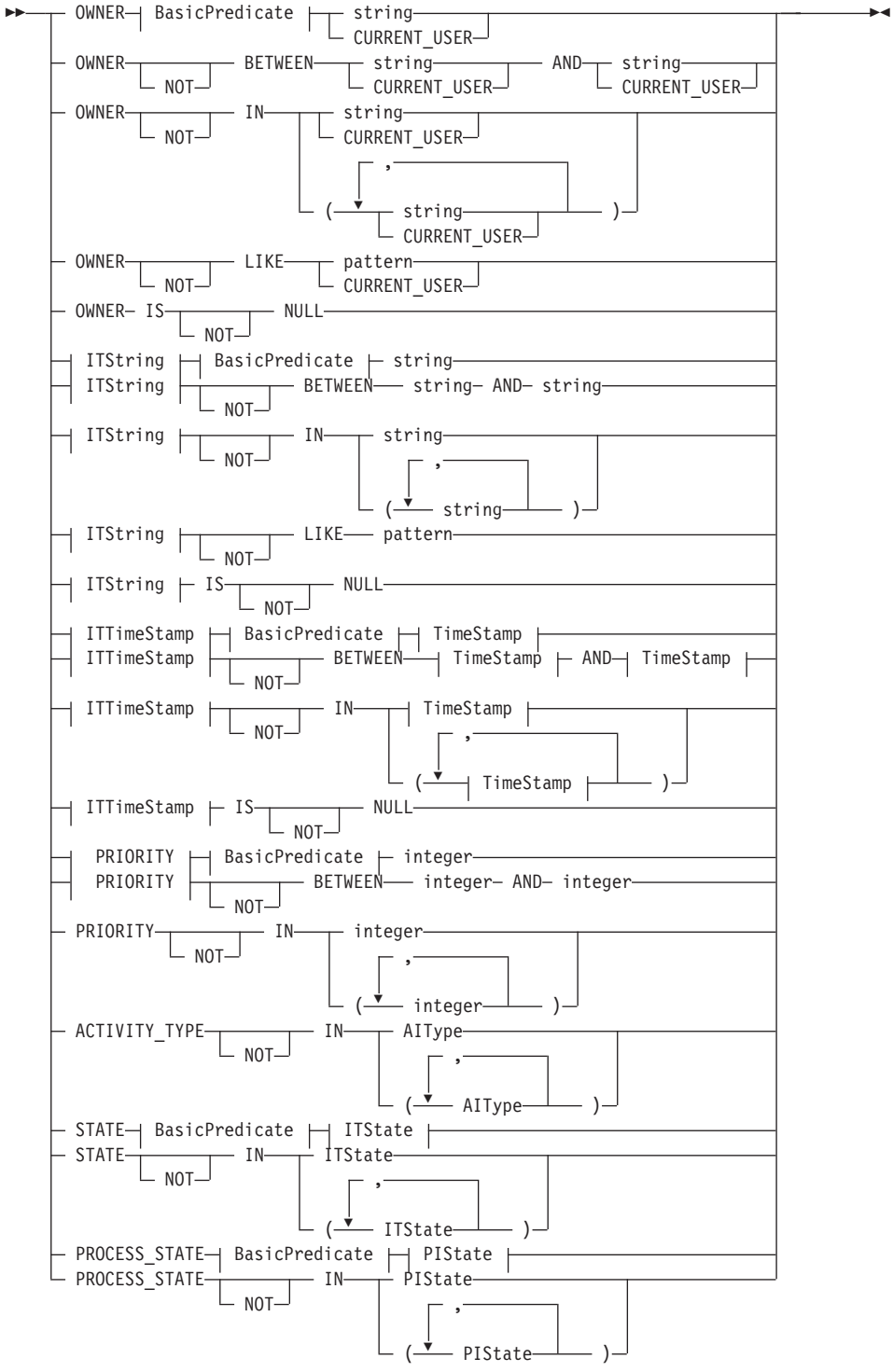
注:

1. *string* (文字列) 定数は、単一引用符 (') で囲む必要があります。
2. ストリング定数内の単一引用符は二重にする ('') 必要があります。
3. *pattern* (パターン) は文字列定数であり、その中のアスタリスクおよび疑問符には特別な意味があります。
   - 疑問符 (?) は任意の単一文字を表します。
   - アスタリスク (*) は、0 文字以上の文字列を表します。
   - エスケープ文字は円記号 (¥) であり、パターン自体に実際の疑問符やアスタリスクが含まれる場合にはこれを使う必要があります。実際の円記号には二重にする必要があります (¥¥)。
4. *TimeStamp* は現地時間に基づいた 24 時間表示のストリング定数です。
5. *TimeStamp* 内のオプションを指定しない場合は、0 (ゼロ) に設定されます。
6. LIKE オペランドのパターンに混合データが含まれている場合は、そのパターンの中にパーセント記号 (%) または下線 (_) を指定することができません。これらのいずれかの文字を使用すると、SQL エラーが戻されます。

PINFilter

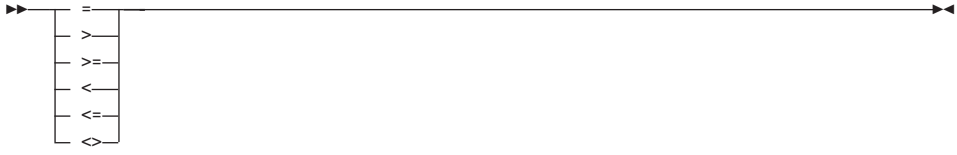


ITPredicate

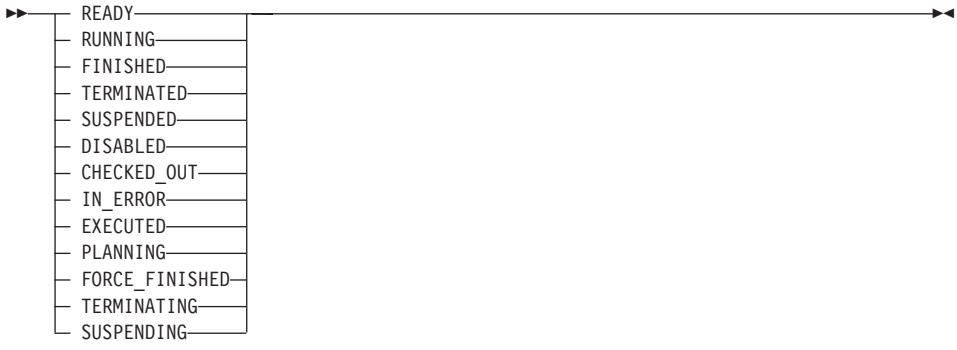

OWNER
BasicPredicate
string
CURRENT_USER
OWNER
NOT
BETWEEN
string
CURRENT_USER
AND
string
CURRENT_USER
OWNER
NOT
IN
string
CURRENT_USER
,
(
string
CURRENT_USER
)
OWNER
NOT
LIKE
pattern
CURRENT_USER
OWNER
IS
NOT
NULL
ITString
BasicPredicate
string
ITString
NOT
BETWEEN
string
AND
string
ITString
NOT
IN
string
,
(
string
)
ITString
NOT
LIKE
pattern
ITString
IS
NOT
NULL
ITTimeStamp
BasicPredicate
TimeStamp
ITTimeStamp
NOT
BETWEEN
TimeStamp
AND
TimeStamp
ITTimeStamp
NOT
IN
TimeStamp
,
(
TimeStamp
)
ITTimeStamp
IS
NOT
NULL
PRIORITY
BasicPredicate
integer
PRIORITY
NOT
BETWEEN
integer
AND
integer
PRIORITY
NOT
IN
integer
,
(
integer
)
ACTIVITY_TYPE
NOT
IN
AIType
,
(
AIType
)
STATE
BasicPredicate
ITState
STATE
NOT
IN
ITState
,
(
ITState
)
PROCESS_STATE
BasicPredicate
PIState
PROCESS_STATE
NOT
IN
PIState
,
(
PIState
)

AIType

BasicPredicate



ITState

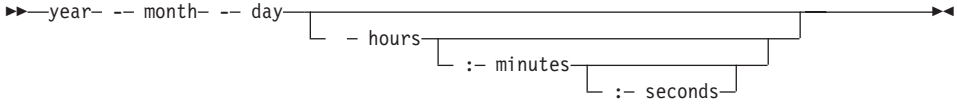


ITString

ITTimeStamp

PIState

TimeStamp



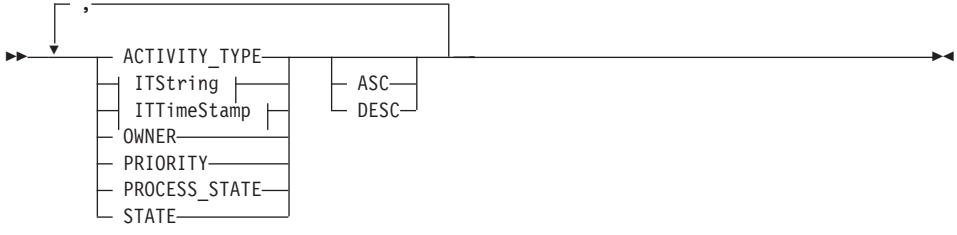
プロセス・インスタンス通知は、ソートすることができます。プロセス・インスタンス通知のソート基準は文字列として指定されます。

注: 省略時のソート順は昇順 (ASC) です。

活動タイプは、AIType の図で示されている順序でソートされます。

状態は、ITState または PIState の図で示されている順序でソートされます。

PINOrderBy



取り出すプロセス・インスタンス通知の数は、クライアントに戻されるアクティビティー・インスタンス通知の最大数を指定するしきい値を使って制限できます。そのしきい値は、指定したソート基準に従ってアクティビティー・インスタンス通知がソートされた後で適用されます。プロセス・インスタンス通知はサーバーでソートされることに注意してください。つまり、サーバーのコード・ページでソート順序が決まります。

各プロセス・インスタンス通知ごとに取り出される 1 次情報は以下のとおりです。

- Category
- CreationTime
- Description
- Icon
- Kind
- LastModificationTime
- Name
- Owner
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	サポートなし
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryProcessInstanceNotifications(
       FmcjExecutionServiceHandle                    service,
       char const *                                  filter,
       char const *                                  sortCriteria,
       unsigned long const *                         threshold,
       FmcjProcessInstanceNotificationVectorHandle * notifications )
```

C++ 言語のシグニチャー

```
APIRET QueryProcessInstanceNotifications(
       string const *                              filter,
       string const *                              sortCriteria,
       unsigned long const *                       threshold,
       vector<FmcjProcessInstanceNotification> & notifications ) const
```

Java のシグニチャー

```
public abstract
ProcessInstanceNotification[] queryProcessInstanceNotifications(
       String                     filter,
       String                     sortCriteria,
       Integer                    threshold   ) throws FmcException
```

パラメーター

filter 入力。検索するプロセス・インスタンス通知を特徴付けるフィルター基準。

items 入出力。プロセス・インスタンス通知の適格ベクトル。

service 入力。実行サーバーとのセッションを表すサービス・オブジェクトに対するハンドル。

sortCriteria 入力。見つかったプロセス・インスタンス通知に適用されるソート基準。

threshold 入力。クライアントに戻されるプロセス・インスタンス通知の最大数を定義するしきい値。

戻り値のデータ型

APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ProcessInstanceNotification[]
適格プロセス・インスタンス通知。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_FILTER(125)
指定されたフィルターはプロセス・インスタンス通知に適用できません。

FMC_ERROR_INVALID_SORT(808)
指定されたソート基準はプロセス・インスタンス通知に適用できません。

FMC_ERROR_INVALID_THRESHOLD(807)
指定したしきい値は無効です。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
戻されるプロセス・インスタンス通知の数は照会結果に許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_QUERY_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- C 言語の例については、843ページの『プロセス・インスタンスの照会 (C 言語)』を参照してください。
- C++ の例については、844ページの『プロセス・インスタンスの照会 (C++)』を参照してください。
- Java の例については、845ページの『プロセス・インスタンスの照会 (Java)』を参照してください。

QueryProcessInstances()

この関数 / メソッドは、ユーザーがアクセスできるプロセス・インスタンスを、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

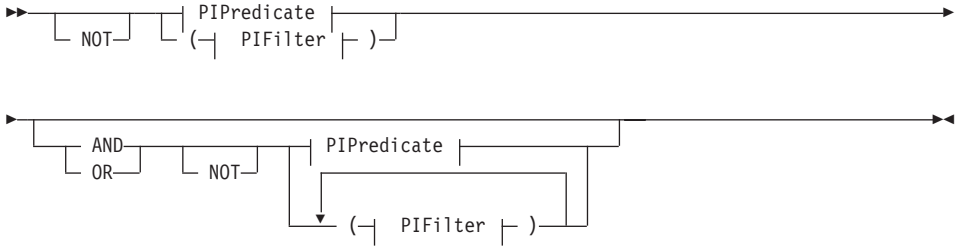
C および C++ では、検索したプロセス・インスタンスはすべて、提供されるベクトルに追加されます。現在のプロセス・インスタンスだけを読み取りたい場合、まずベクトルをクリアした後でこの関数 / メソッドを呼び出さなければなりません。つまり、C 言語ではベクトル・ハンドルを 0 に設定し、C++ API ではベクトルのすべての要素を消去する必要があります。

プロセス・インスタンスのフィルターは、フィルター述語を含むフィルター文字列として指定します。

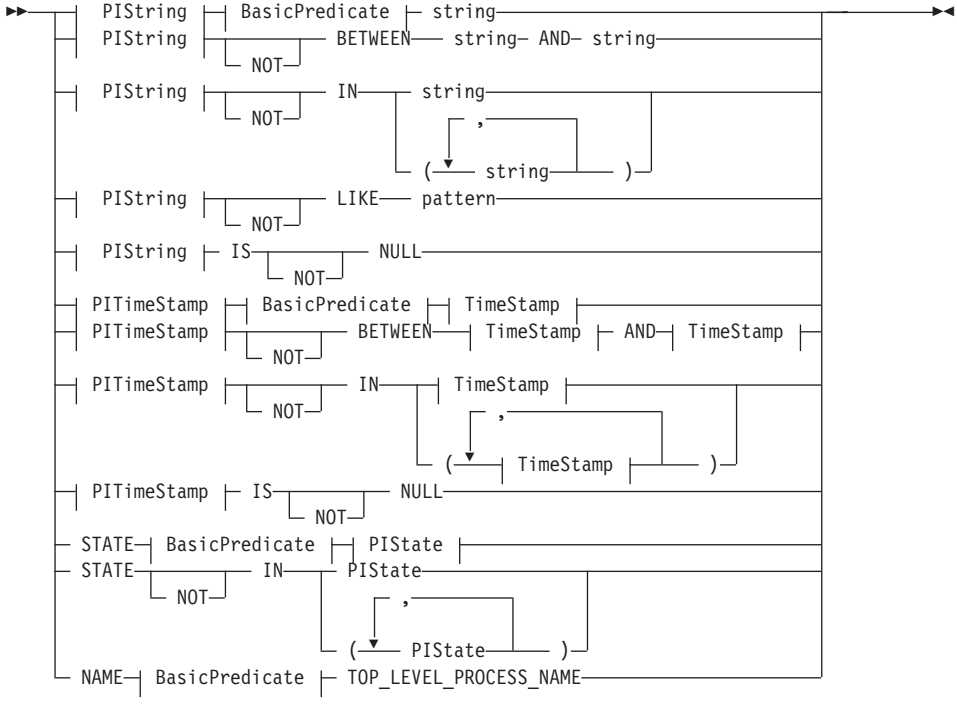
注:

1. *string* (文字列) 定数は、単一引用符 (') で囲む必要があります。
2. ストリング定数内の単一引用符は二重にする ('') 必要があります。
3. *pattern* (パターン) は文字列定数であり、その中のアスタリスクおよび疑問符には特別な意味があります。
   - 疑問符 (?) は任意の単一文字を表します。
   - アスタリスク (*) は、0 文字以上の文字列を表します。
   - エスケープ文字は円記号 (¥) であり、パターン自体に実際の疑問符やアスタリスクが含まれる場合にはこれを使う必要があります。実際の円記号には二重にする必要があります (¥¥)。
4. *TimeStamp* は現地時間に基づいた 24 時間表示のストリング定数です。
5. *TimeStamp* 内のオプションを指定しない場合は、0 (ゼロ) に設定されます。
6. LIKE オペランドのパターンに混合データが含まれている場合は、そのパターンの中にパーセント記号 (%) または下線 (_) を指定することができません。これらのいずれかの文字を使用すると、SQL エラーが戻されます。

PIFilter



PIPredicate



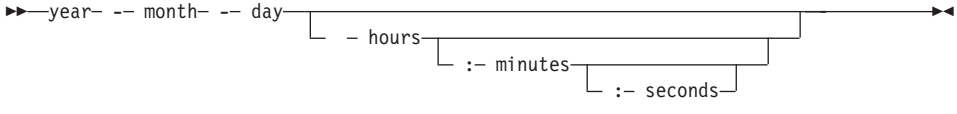
BasicPredicate

PIState

PIString

PITimeStamp

TimeStamp

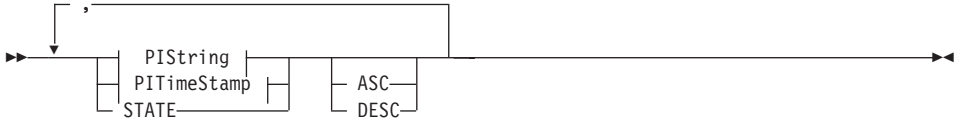


プロセス・インスタンスは、ソートすることができます。プロセス・インスタンスのソート基準は文字列として指定されます。

注: 省略時のソート順は昇順 (ASC) です。

状態は、PIState の図で示されている順序でソートされます。

PIOrderBy



取り出すプロセス・インスタンスの数は、クライアントに戻されるプロセス・インスタンスの最大数を指定するしきい値を使って制限できます。そのしきい値は、指定したソート基準に従ってプロセス・インスタンスがソートされた後で適用されます。プロセス・インスタンスはサーバーでソートされることに注意してください。つまり、サーバーのコード・ページでソート順序が決まります。

各プロセス・インスタンスごとに取り出される 1 次情報は以下のとおりです。

- Category
- Description
- Icon
- InContainerNeeded
- LastModificationTime
- LastStateChangeTime
- Name
- ParentName
- ProcessTemplateName
- State
- SuspensionTime
- SystemName
- SystemGroupName
- TopLevelName

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	サポートなし
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryProcessInstances(
       FmcjExecutionServiceHandle        service,
       char const *                      filter,
       char const *                      sortCriteria,
       unsigned long const *             threshold,
       FmcjProcessInstanceVectorHandle * instances )
```

C++ 言語のシグニチャー

```
APIRET QueryProcessInstances(
       string const *                  filter,
       string const *                  sortCriteria,
       unsigned long const *           threshold,
       vector<FmcjProcessInstance> &   instances ) const
```

Java のシグニチャー

```
public abstract
ProcessInstance[]  queryProcessInstances(
       String                          filter,
       String                          sortCriteria,
       Integer                         threshold   ) throws FmcException
```

パラメーター

filter 入力。検索するプロセス・インスタンスを特徴付けるフィルター基準。

instances 入出力。プロセス・インスタンスの適格ベクトル。

service 入力。実行サーバーとのセッションを表すサービス・オブジェクトに対するハンドル。

sortCriteria 入力。見つかったプロセス・インスタンスに適用されるソート基準。

threshold 入力。クライアントに戻されるプロセス・インスタンスの最大数を定義するしきい値。

戻り値のデータ型

APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ProcessInstance[]
適格プロセス・インスタンス。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_FILTER(125)
指定されたフィルターはプロセス・インスタンスに適用できません。

FMC_ERROR_INVALID_SORT(808)
指定されたソート基準はプロセス・インスタンスに適用できません。

FMC_ERROR_INVALID_THRESHOLD(807)
指定したしきい値は無効です。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
戻されるプロセス・インスタンスの数は照会結果に許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_QUERY_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- C 言語の例については、843ページの『プロセス・インスタンスの照会 (C 言語)』を参照してください。
- C++ の例については、844ページの『プロセス・インスタンスの照会 (C++)』を参照してください。
- Java の例については、845ページの『プロセス・インスタンスの照会 (Java)』を参照してください。

QueryProcessTemplateLists()

この関数 / メソッドは、ユーザーがアクセスできる現在のプロセス・テンプレート・リストを、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

C および C++ では、検索したプロセス・テンプレート・リストはすべて、提供されるベクトルに追加されます。現在のプロセス・テンプレート・リストだけを読み取りたい場合、まずベクトルをクリアした後でこの関数 / メソッドを呼び出さなければなりません。つまり、C 言語ではベクトル・ハンドルを 0 に設定し、C++ API ではベクトルのすべての要素を消去する必要があります。ActiveX の場合、ExecutionService 上のプロセス・テンプレート・リスト配列が更新されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long QueryProcessTemplateLists()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryProcessTemplateLists(
       FmcjExecutionServiceHandle            service,
       FmcjProcessTemplateListVectorHandle * lists )
```

C++ 言語のシグニチャー

```
APIRET QueryProcessTemplateLists(
       vector<FmcjProcessTemplateList> &  lists ) const
```

Java のシグニチャー

```
public abstract
ProcessTemplateList[] queryProcessTemplateLists() throws FmcException
```

パラメーター

lists 入出力。プロセス・テンプレート・リストのベクトル。

service 入力。実行サーバーとのセッションを表すサービス・オブジェクトに対するハンドル。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ProcessTemplateList[]
適格プロセス・テンプレート・リスト。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
戻されるプロセス・テンプレート・リストの数は照会結果に許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_QUERY_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- ActiveX の例については、832ページの『ワークリストの照会 (ActiveX)』を参照してください。
- C 言語の例については、833ページの『ワークリストの照会 (C 言語)』を参照してください。
- C++ の例については、835ページの『ワークリストの照会 (C++)』を参照してください。
- Java の例については、837ページの『ワークリストの照会 (Java)』を参照してください。

QueryProcessTemplates()

この関数 / メソッドは、プロセス・テンプレートを MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

C および C++ では、検索したプロセス・テンプレートはすべて、提供されるベクトルに追加されます。現在のプロセス・テンプレートだけを読み取りたい場合、まずベクトルをクリアした後で、この関数 / メソッドを呼び出さなけれ

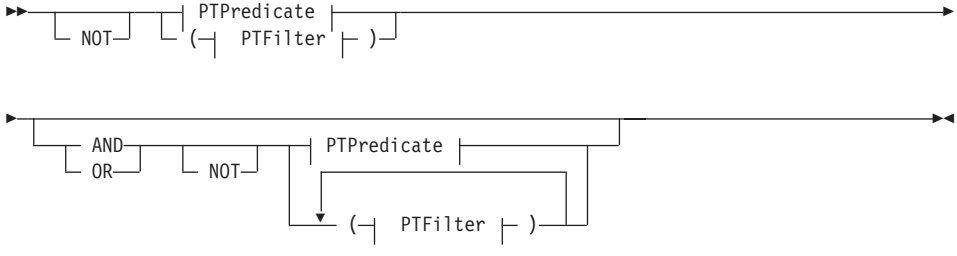
ばなりません。つまり、C 言語ではベクトル・ハンドルを 0 に設定し、C++ API ではベクトルのすべての要素を消去する必要があります。

プロセス・テンプレートのフィルターは、フィルター述語を含むフィルター文字列として指定します。

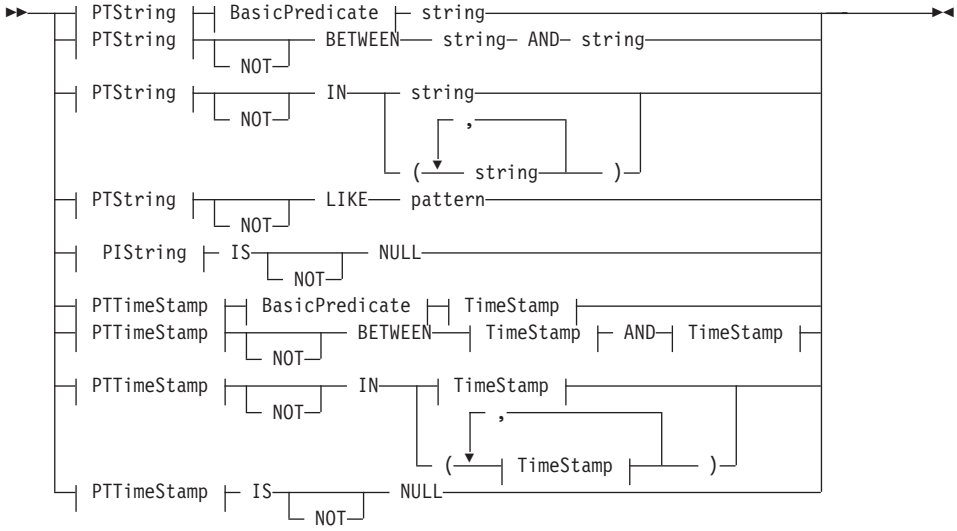
注:

1. *string* (文字列) 定数は、単一引用符 (') で囲む必要があります。
2. ストリング定数内の単一引用符は二重にする ('') 必要があります。
3. *pattern* (パターン) は文字列定数であり、その中のアスタリスクおよび疑問符には特別な意味があります。
   - 疑問符 (?) は任意の単一文字を表します。
   - アスタリスク (*) は、0 文字以上の文字列を表します。
   - エスケープ文字は円記号 (¥) であり、パターン自体に実際の疑問符やアスタリスクが含まれる場合にはこれを使う必要があります。実際の円記号には二重にする必要があります (¥¥)。
4. *TimeStamp* は現地時間に基づいた 24 時間表示のストリング定数です。
5. *TimeStamp* 内のオプションを指定しない場合は、0 (ゼロ) に設定されます。
6. LIKE オペランドのパターンに混合データが含まれている場合は、そのパターンの中にパーセント記号 (%) または下線 (_) を指定することができません。これらのいずれかの文字を使用すると、SQL エラーが戻されます。

PTFilter



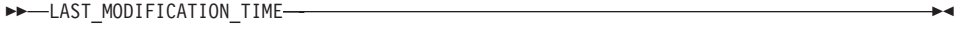
PTPredicate



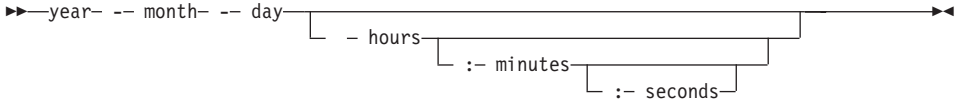
BasicPredicate

PTString

PTTimeStamp



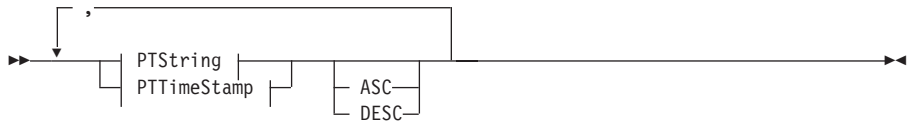
TimeStamp



プロセス・テンプレートは、ソートすることができます。プロセス・テンプレートのソート基準は文字列として指定されます。

注: 省略時のソート順は昇順 (ASC) です。

PTOrderBy



取り出すプロセス・テンプレートの数は、クライアントに戻されるプロセス・テンプレートの最大数を指定するしきい値を使って制限できます。そのしきい値は、指定したソート基準に従ってプロセス・テンプレートがソートされた後で適用されます。プロセス・テンプレートはサーバーでソートされることに注意してください。つまり、サーバーのコード・ページでソート順序が決まります。

各プロセス・テンプレートごとに取り出される 1 次情報は以下のとおりです。

- Category
- CreationTime
- Description
- Icon
- InContainerNeeded
- LastModificationTime
- Name

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX サポートなし

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ExecutionService

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryProcessTemplates(
       FmcjExecutionServiceHandle        service,
       char const *                      filter,
       char const *                      sortCriteria,
       unsigned long const *             threshold,
       FmcjProcessTemplateVectorHandle * templates )
```

C++ 言語のシグニチャー

```
APIRET QueryProcessTemplates(
       string const *                 filter,
       string const *                 sortCriteria,
       unsigned long const *          threshold,
       vector<FmcjProcessTemplate> &  templates ) const
```

Java のシグニチャー

```
public abstract
ProcessTemplates[] queryProcessTemplates(
       String                         filter,
       String                         sortCriteria,
       Integer                        threshold   ) throws FmcException
```

パラメーター

filter 入力。検索するプロセス・テンプレートを特徴付けるフィルター基準。

service 入力。実行サーバーとのセッションを表すサービス・オブジェクトに対するハンドル。

sortCriteria 入力。見つかったプロセス・テンプレートに適用されるソート基準。

templates 入出力。プロセス・テンプレートの適格ベクトル。

threshold 入力。クライアントに戻されるプロセス・テンプレートの最大数を定義するしきい値。

戻り値のデータ型

APIRET

この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ProcessTemplate[]

適格プロセス・テンプレート。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)

パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)

提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_FILTER(125)

指定されたフィルターはプロセス・テンプレートに適用できません。

FMC_ERROR_INVALID_SORT(808)

指定されたソート基準はプロセス・テンプレートに適用できません。

FMC_ERROR_INVALID_THRESHOLD(807)

指定したしきい値は無効です。

FMC_ERROR_NOT_LOGGED_ON(106)

ログオンしていません。

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)

戻されるプロセス・テンプレートの数は照会結果に許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_QUERY_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_COMMUNICATION(13)

指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)

MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)

内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- C 言語の例については、843ページの『プロセス・インスタンスの照会 (C 言語)』を参照してください。
- C++ の例については、844ページの『プロセス・インスタンスの照会 (C++)』を参照してください。
- Java の例については、845ページの『プロセス・インスタンスの照会 (Java)』を参照してください。

QueryWorkitems()

この関数 / メソッドは、ユーザーがアクセスできる作業項目を、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

C および C++ では、検索した作業項目はすべて、提供されるベクトルに追加されます。現在の作業項目だけを読み取りたい場合、まずベクトルをクリアした後でこの関数 / メソッドを呼び出さなければなりません。つまり、C 言語ではベクトル・ハンドルを 0 に設定し、C++ API ではベクトルのすべての要素を消去する必要があります。

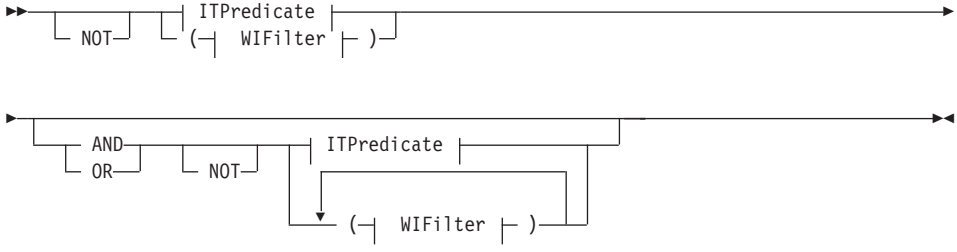
検索する作業項目は、フィルターによって特徴付けることができます。作業項目のフィルターは、文字列として指定されます。

注:

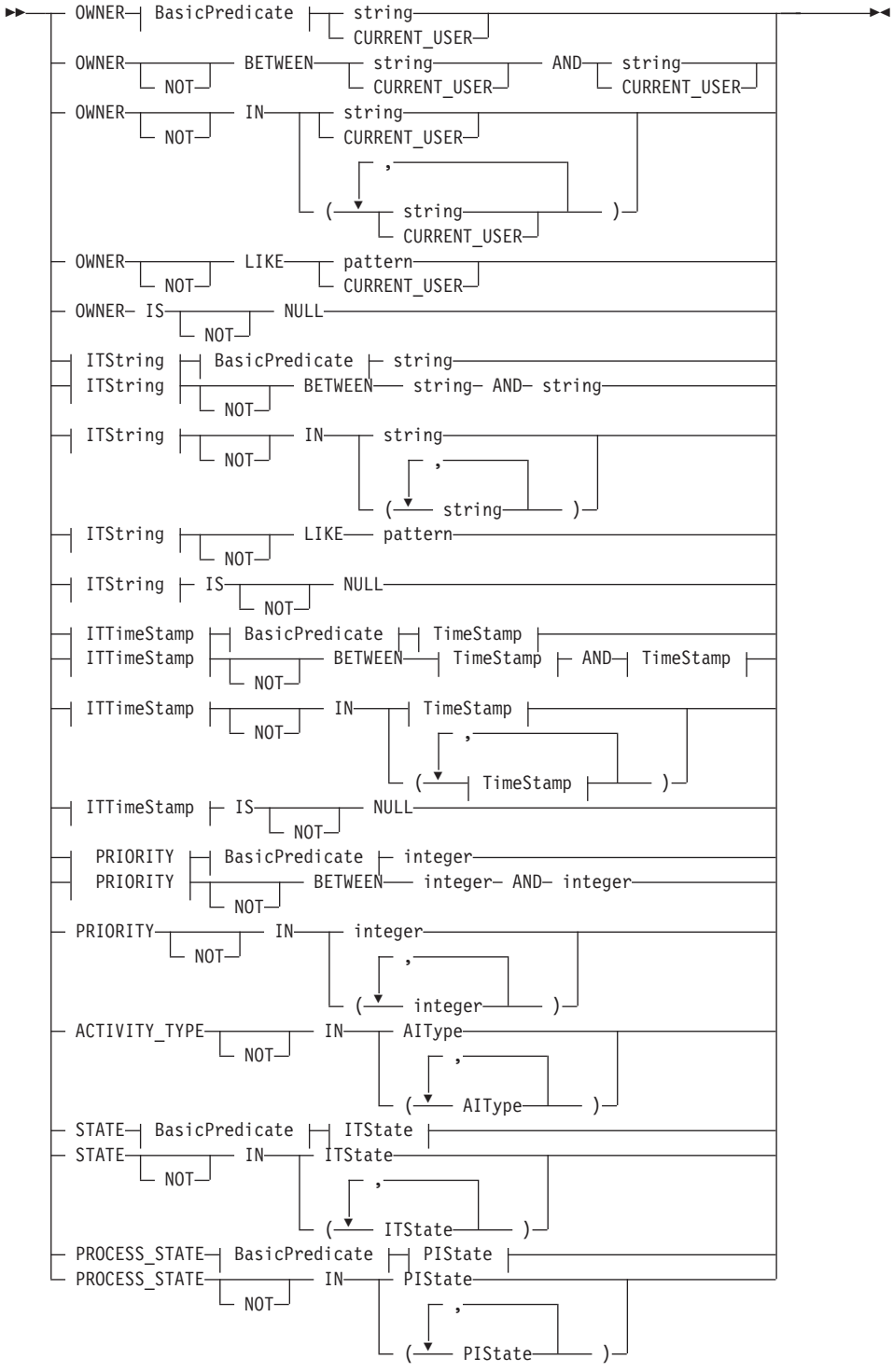
1. *string* (文字列) 定数は、単一引用符 (') で囲む必要があります。
2. ストリング定数内の単一引用符は二重にする ('') 必要があります。
3. *pattern* (パターン) は文字列定数であり、その中のアスタリスクおよび疑問符には特別な意味があります。
   - 疑問符 (?) は任意の単一文字を表します。
   - アスタリスク (*) は、0 文字以上の文字列を表します。
   - エスケープ文字は円記号 (¥) であり、パターン自体に実際の疑問符やアスタリスクが含まれる場合にはこれを使う必要があります。実際の円記号には二重にする必要があります (¥¥)。
4. *TimeStamp* は現地時間に基づいた 24 時間表示のストリング定数です。
5. *TimeStamp* 内のオプションを指定しない場合は、0 (ゼロ) に設定されます。

6. LIKE オペランドのパターンに混合データが含まれている場合は、そのパターンの中にパーセント記号 (%) または下線 (_) を指定することができません。これらのいずれかの文字を使用すると、SQL エラーが戻されます。

WIFilter

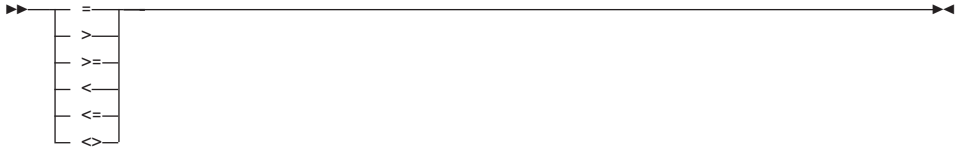


ITPredicate

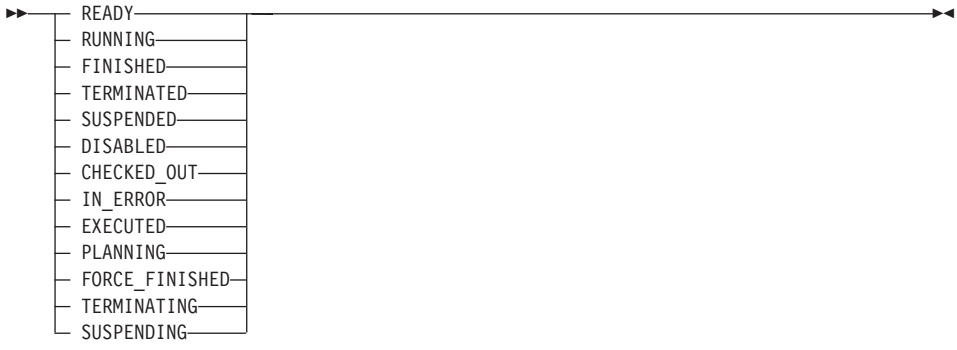

OWNER BasicPredicate string CURRENT_USER
OWNER NOT BETWEEN string CURRENT_USER AND string CURRENT_USER
OWNER NOT IN string CURRENT_USER , (string CURRENT_USER)
OWNER NOT LIKE pattern CURRENT_USER
OWNER IS NOT NULL
ITString BasicPredicate string
ITString NOT BETWEEN string AND string
ITString NOT IN string , (string)
ITString NOT LIKE pattern
ITString IS NOT NULL
ITTimeStamp BasicPredicate TimeStamp
ITTimeStamp NOT BETWEEN TimeStamp AND TimeStamp
ITTimeStamp NOT IN TimeStamp , (TimeStamp)
ITTimeStamp IS NOT NULL
PRIORITY BasicPredicate integer
PRIORITY NOT BETWEEN integer AND integer
PRIORITY NOT IN integer , (integer)
ACTIVITY_TYPE NOT IN AIType , (AIType)
STATE BasicPredicate ITState
STATE NOT IN ITState , (ITState)
PROCESS_STATE BasicPredicate PIState
PROCESS_STATE NOT IN PIState , (PIState)

AIType

BasicPredicate



ITState

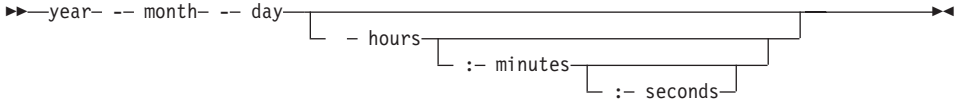


ITString

ITTimeStamp

PIState

TimeStamp



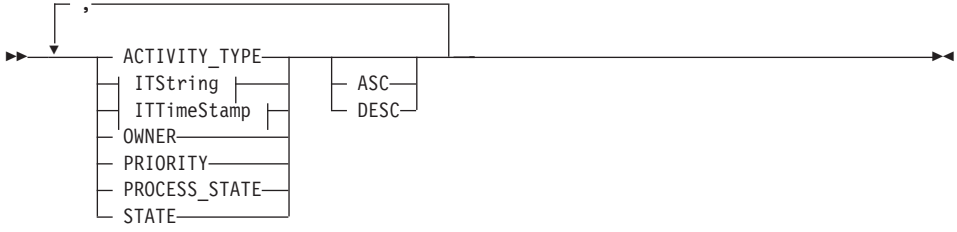
作業項目は、ソートすることができます。作業項目のソート基準は文字列として指定されます。

注: 省略時のソート順は昇順 (ASC) です。

活動タイプは、AIType の図で示されている順序でソートされます。

状態は、ITState または PIState の図で示されている順序でソートされます。

WIOrderBy



取り出す作業項目の数は、クライアントに戻される作業項目の最大数を指定するしきい値を使って制限できます。そのしきい値は、指定したソート基準に従ってアイテムがソートされた後で適用されます。アイテムはサーバーでソートされることに注意してください。つまり、サーバーのコード・ページでソート順序が決まります。

各作業項目ごとに取り出される 1 次情報は以下のとおりです。

- ActivityType
- Category
- CreationTime
- Description
- Icon
- Implementation
- Kind
- LastModificationTime
- Name
- Owner
- Priority
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State
- SupportTools

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	サポートなし
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryWorkitems(
       FmcjExecutionServiceHandle  service,
       char const *                filter,
       char const *                sortCriteria,
       unsigned long const *       threshold,
       FmcjWorkitemHandle *        workitems )
```

C++ 言語のシグニチャー

```
APIRET QueryWorkitems(
       string const *                filter,
       string const *                sortCriteria,
       unsigned long const *         threshold,
       vector<FmcjWorkitem> &        workitems ) const
```

Java のシグニチャー

```
public abstract
WorkItem[] queryWorkItems(
       String                         filter,
       String                         sortCriteria,
       Integer                        threshold   ) throws FmcException
```

パラメーター

filter 入力。検索する作業項目を特徴付けるフィルター基準。

service 入力。実行サーバーとのセッションを表すサービス・オブジェクトに対するハンドル。

sortCriteria 入力。見つかった作業項目に適用されるソート基準。

threshold 入力。クライアントに戻される作業項目の最大数を定義するしきい値。

workitems 入出力。作業項目の適格ベクトル。

戻り値のデータ型

APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

WorkItem[] 適格作業項目。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_FILTER(125)
指定されたフィルターは作業項目に適用できません。

FMC_ERROR_INVALID_SORT(808)
指定されたソート基準は作業項目に適用できません。

FMC_ERROR_INVALID_THRESHOLD(807)
指定したしきい値は無効です。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
戻される作業項目の数は照会結果に許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_QUERY_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- C 言語の例については、843ページの『プロセス・インスタンスの照会 (C 言語)』を参照してください。
- C++ の例については、844ページの『プロセス・インスタンスの照会 (C++)』を参照してください。
- Java の例については、845ページの『プロセス・インスタンスの照会 (Java)』を参照してください。

照会元プロセス・インスタンスに類似した作業項目を照会します。

QueryWorklists()

この関数 / メソッドは、ユーザーがアクセスできるワークリストを、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

C および C++ では、検索したワークリストはすべて、提供されるベクトルに追加されます。現在のワークリストだけを読み取りたい場合、まずベクトルをクリアした後でこの関数 / メソッドを呼び出さなければなりません。つまり、C 言語ではベクトル・ハンドルを 0 に設定し、C++ API ではベクトルのすべての要素を消去する必要があります。 ActiveX の場合、ExecutionService 配列上のワークリスト配列が更新されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long QueryWorklists()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceQueryWorklists(
       FmcjExecutionServiceHandle  service,
       FmcjWorklistVectorHandle *  lists )
```

C++ 言語のシグニチャー

```
APIRET QueryWorklists( vector<FmcjWorklist> & lists ) const
```

Java のシグニチャー

```
public abstract
WorkList[] queryWorkLists()  throws FmcException
```

パラメーター

lists 入出力。ワークリストのベクトル。

service 入力。実行サーバーとのセッションを表すサービス・オブジェクトに対するハンドル。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

WorkList[] 適格ワークリスト。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_QRY_RESULT_TOO_LARGE(817)
戻されるワークリストの数は照会結果に許可された最大サイズ

を超えました。システム、システム・グループ、またはドメインの MAXIMUM_QUERY_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- ActiveX の例については、832ページの『ワークリストの照会 (ActiveX)』を参照してください。
- C 言語の例については、833ページの『ワークリストの照会 (C 言語)』を参照してください。
- C++ の例については、835ページの『ワークリストの照会 (C++)』を参照してください。
- Java の例については、837ページの『ワークリストの照会 (Java)』を参照してください。

Receive()

この関数 / メソッドは、MQSeries Workflow 実行サーバーによって出されたデータの受け取り、または非同期要求の応答の受け取りを開始します。

相関 ID を使うと、特定の応答を受け取ることができます。送られてきたデータを受け取るには、相関 ID は 0 (ヌル) ポインターであるか、または FMCJ_NO_CORRELID を指定する、つまりゼロ (0x00) しか含まれていないバッファーを指す必要があります。 0 ポインターが渡されない限り、戻される時に相関 ID が設定されることに注意してください。つまり、それぞれの要求を出すごとに、これをリセットする必要があります。

タイムアウト値では、アプリケーションがデータの到着を待つ最大時間を指定します。指定の時間内にデータが到着しない場合には、タイムアウト・エラーが出ます。タイムアウト値が -1 なら、それは無限に待機するよう指定することになります。

データが正しく受信されると、実行データには、送信されたデータが含まれているので、そのデータを使ってオブジェクトを更新したり新しいオブジェクトを作成したりできます。実行データオブジェクトがサポートしている関数 / メソッドについては、290ページの『実行データ』を参照してください。

受け取る実行データの内容は、以下の列挙型を使用して指定できます。

ActiveX サポートなし

C 言語 FmcjExecutionDataKindEnum

C++ FmcjExecutionData::KindEnum

JAVA サポートなし

列挙型定数には以下の値を使用できます。可能な限り整数値ではなく記号名を使うようにしてください。

NotSet(0) 実行データの内容がわからないことを示します。

C 言語 Fmc_DART_NotSet

C++ FmcjExecutionData::NotSet

Terminate(2) データの受信を終了できることを示します。

C 言語 Fmc_DART_Terminate

C++ FmcjExecutionData::Terminate

ItemDeleted(1000)

作業項目、アクティビティー・インスタンス通知、またはプロセス・インスタンス通知がすでに削除されていることを示します。

C 言語 Fmc_DART_ItemDeleted

C++ FmcjExecutionData::ItemDeleted

Workitem(1002)

作業項目がすでに作成または更新されていることを示します。

C 言語 Fmc_DART_Workitem

C++ FmcjExecutionData::Workitem

ActivityInstanceNotification(1003)

アクティビティー・インスタンス通知がすでに作成または更新されていることを示します。

C 言語 Fmc_DART_ActivityInstanceNotification

C++ FmcjExecutionData::ActivityInstanceNotification

ProcessInstanceNotification(1004)

プロセス・インスタンス通知がすでに作成または更新されていることを示します。

C 言語 Fmc_DART_ProcessInstanceNotification

C++ FmcjExecutionData::ProcessInstanceNotification

ExecuteInstanceResponse(1100)

実行データが ExecuteProcessInstance() 要求の応答を含んでいることを示します。

C 言語 Fmc_DART_ExecuteInstanceResponse

C++ FmcjExecutionData::ExecuteInstanceResponse

ExecuteProgramResponse(1101)

実行データが ExecuteProgram() 要求の応答を含んでいることを示します。

C 言語 Fmc_DART_ExecuteProgramResponse

C++ FmcjExecutionData::ExecuteProgramResponse

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー (在席セッション・モード)

API インターフェース宣言

ActiveX 該当しない

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA サポートなし

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceReceive(
       FmcjExecutionServiceHandle   service,
       FmcjCorrelID *               correlID,
       FmcjExecutionDataHandle *    data,
       signed long                  timeout )
```

C++ 言語のシグニチャー

```
APIRET Receive( FmcjCorrelID *      correlID,
                FmcjExecutionData & data,
                signed long         timeout  ) const
```

パラメーター

correlID 入出力。このデータを直前の要求と相関させるための相関 ID。データを受信できるようにするには、ヌル (0) ポインターであるか、Fmcj_No_CorrelID を指す必要があります。

data 出力。 MQSeries Workflow 実行サーバーから送られるデータ。

service 入力。実行サーバーとの現在のセッションを表すサービス・オブジェクトに対するハンドル。

timeout 入力。データの到着を待つ最大時間 (ミリ秒)。

戻り値のデータ型

APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_MESSAGE_DATA(104)
クライアントが不明なメッセージを受信しました。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

RemotePassthrough()

この関数 / メソッドは、このプログラム内から MQSeries Workflow 実行サーバーとのユーザー・セッションを確立するために、アプリケーション・プログラムによって使われます (アクティビティー・インプリメンテーション呼び出し)。

MQSeries Workflow プログラム実行エージェントによって開始されるアクティビティー・インプリメンテーションは、それ以上の識別なしでプログラム実行エージェントからサービスを要求することができます。それは、プログラム実行エージェントによって認識されています。

そのアクティビティー・インプリメンテーションが作業を他のいくつかのプログラムに分配して、それらのプログラムを別々のオペレーティング・システム・プロセスとして開始した場合、それらのプロセスはプログラム実行エージェントによって認識されず、サービスを要求することはできません。しかし、アクティビティー・インプリメンテーションは、プログラム ID 識別をプログラム実行エージェントに対して要求し、その ID を開始済みのプログラムに渡すことができます。つまり、開始済みのプログラムは、実際のアクティビティー・インプリメンテーションが活動している限り、プログラム実行エージェントと交信する許可を得ることになります。

その後、開始されたプログラムは、このプログラム識別を指定することによって、自力でプログラム実行エージェントからのサービスを要求できます。

正常に実行されると、同じ実行サーバーとのセッションは、元の作業項目が開始した場所からセットアップされます。セッションがセットアップされるユーザーと、元の作業項目が開始されたユーザーは同じです。

使用上の注意

- 141ページの『アクティビティー・インプリメンテーション関数 / メソッド』にある一般的な情報を参照してください。

許可

有効なプログラム識別

必要な接続

なし。ただし MQSeries Workflow プログラム実行エージェントはアクティブでなければなりません。

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long RemotePassthrough( BSTR programID )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceRemotePassthrough(
       FmcjExecutionServiceHandle  service )
       char const *                programID )
```

C++ 言語のシグニチャー

```
APIRET RemotePassthrough( string const & programID )
```

Java のシグニチャー

```
public abstract
void remotePassthrough( String programID ) throws FmcException
```

パラメーター

programID 入力。実際に開始されるアクティビティー・インプリメンテーションがプログラム実行エージェントに認識されるのに必要なプログラム識別子。

service 入力。実行サーバーとの間で確立するセッションを表すサービス・オブジェクトに対するハンドル。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_PROGRAMID(135)
プログラム識別が無効です。

FMC_ERROR_PROGRAM_EXECUTION(126)
Passthrough はアクティビティー・インプリメンテーションが開始したプログラムから呼び出されていないか、プログラム実行エージェントがアクティブではありません。

FMC_ERROR_TOOL_FUNCTION(128)
サポート・ツールが開始したプログラムから、またはプログラム実行サーバーが開始したプログラムから Passthrough を呼び出すことはできません。

FMC_ERROR_USERID_UNKNOWN(10)
作業項目を開始したユーザーが存在していません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

SetPersonAbsent()

この関数 / メソッドは、指定したユーザーの不在指示を、指定した値に設定します (アクション呼び出し)。

不在の担当者はスタッフ決定に参加しません。つまり、どの作業項目も割り当てられません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- 同じユーザーとして、つまり自分の不在の変更を要求する
- スタッフ権限
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long SetPersonAbsent( BSTR userID, boolean newValue )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceSetPersonAbsent(
       FmcjExecutionServiceHandle  service,
       char const *                userID,
       bool                        newValue )
```

C++ 言語のシグニチャー

```
APIRET SetPersonAbsent( string const * userID   = 0,
                        bool           newValue = true )
```

Java のシグニチャー

```
public abstract
void   setPersonAbsent() throws FmcException
public abstract
void   setPersonAbsent2( String userID, boolean newValue )
throws FmcException
```

パラメーター

service 入力。担当者の不在を設定するサービス・オブジェクトのハンドル。

newValue 入力。担当者が不在の場合は真、それ以外の場合は偽。

userID 入力。不在を設定する担当者のユーザー ID。 C++ でユーザー ID が指定されていない場合、ログオン・ユーザーの不在が設定されます。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_USERID_UNKNOWN(10)
指定されたユーザー ID が不明です。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。シ

ステム、システム・グループ、またはドメインの
MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

TerminateReceive()

この関数 / メソッドは、クライアントの入力キューに、MQSeries Workflow 実行サーバーからのデータ受け取りを終了してもよいという情報を入れます。

このようにして、受け取り側のアプリケーションは、データの受け取りを終了してもよいことを通知されます。その後、どんなアクションが実行されるかは、アプリケーションによります。

correlID パラメーターが FMCJ_NO_CORRELID に初期化されたバッファーを指している場合、つまりゼロ (0x00) しか含まれていないバッファーを指している場合、このデータを明示的に受信するために使用できる相関 ID が戻されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

なし

API インターフェース宣言

ActiveX サポートなし

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA サポートなし

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjExecutionServiceTerminateReceive(
       FmcjExecutionServiceHandle  service,
       FmcjCorrelID *              correlID )
```

C++ 言語のシグニチャー

```
APIRET TerminateReceive( FmcjCorrelID * correlID = 0 )
```

パラメーター

correlID 入出力。この要求の相関のための相関 ID。

service 入力。サービス・オブジェクトのハンドル。

戻り値のデータ型

APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_CORRELATION_ID(506)
渡された相関 ID が FMCJ_NO_CORRELID ではありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

第43章 インスタンス・モニターのアクション

InstanceMonitor オブジェクトは、ActiveX API のプロセス・インスタンス、ブロック 型のアクティビティー・インスタンス、または プロセス 型のアクティビティー・インスタンスに対するモニターを表します。

以下の部分では、インスタンス・モニターに適用されるアクションについて説明します。関数 / メソッドの完全なリストについては、305ページの『インスタンス・モニター』を参照してください。

ObtainInstanceMonitor()

この関数 / メソッドは、指定したアクティビティー・インスタンスのインスタンス・モニターを MQSeries Workflow 実行サーバーから取り出します (アクション呼び出し)。インスタンス・モニターがすでに取り出されている場合、そのモニターが呼び出し側に戻されます。

指定されるアクティビティー・インスタンスは、ブロック 型またはプロセス型でなければならず、このインスタンス・モニターの一部でなければなりません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- プロセスの許可
- プロセス管理の許可
- プロセス管理者として
- プロセス作成者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow 3.1

C 言語 該当しない (383ページの『第40章 ブロック・インスタンス・モニターのアクション』を参照)

C++ 該当しない (383ページの『第40章 ブロック・インスタンス・モニターのアクション』を参照)

JAVA 該当しない (383ページの『第40章 ブロック・インスタンス・モニターのアクション』を参照)

ActiveX のシグニチャー

```
InstanceMonitor *
ObtainInstanceMonitor( long *             returnCode,
                       ActivityInstance * activity,
                       boolean            deep    )
```

パラメーター

activity 入力。インスタンス・モニターを取り出すアクティビティー・インスタンス。

deep 入力。ブロック 型のアクティビティー・インスタンスのモニターも検索するかどうかを示す標識。ただし、現時点では deep はサポートされていません。

returnCode 入出力。この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

戻り値のデータ型

InstanceMonitor*

取り出されたインスタンス・モニター。

戻りコード

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)

パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)

オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
指定されたアクティビティー・インスタンスはインスタンス・モニターによって記述されていないか、またはもう存在しません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Refresh()

この関数 / メソッドは、MQSeries Workflow 実行サーバーからインスタンス・モニターをリフレッシュします (アクション呼び出し)。

インスタンス・モニターに関するすべての情報を検索します。

deep オプションを指定すると、*ブロック* 型のアクティビティー・インスタンスが解決され、それらのインスタンス・モニターもサーバーからリフレッシュされます。

注: 現時点では、deep はサポートされていません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- プロセスの許可
- プロセス管理の許可
- プロセス管理者として
- プロセス作成者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow 3.1

C 言語 該当しない (383ページの『第40章 ブロック・インスタンス・モニターのアクション』を参照)

C++ 該当しない (383ページの『第40章 ブロック・インスタンス・モニターのアクション』を参照)

JAVA 該当しない (383ページの『第40章 ブロック・インスタンス・モニターのアクション』を参照)

C++ 言語のシグニチャー

```
long Refresh( boolean deep )
```

パラメーター

deep 入力。ブロック型のアクティビティー・インスタンスを解決するかどうか (つまりそのモニターも取り出すかどうか) を指定する標識。ただし、現時点で deep は無視されます。

戻り値のデータ型

long この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

戻りコード

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・インスタンスはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

第44章 アイテムのアクション

FmcjItem または Item オブジェクトは、作業項目、アクティビティー・インスタンス通知、またはプロセス・インスタンス通知を表します。

FmcjItem または Item オブジェクトは、作業項目と通知とで共通ないくつかの性質を表します。 C++ 言語の場合、FmcjItem は FmcjWorkitem クラス、FmcjActivityInstanceNotification クラス、および FmcjProcessInstanceNotification クラスのスーパークラスであり、すべての共通のプロパティーとメソッドを提供します。 Java 言語の場合、Item は WorkItem クラス、ActivityInstanceNotification クラス、および ProcessInstanceNotification クラスのスーパークラスであり、すべての共通のプロパティーとメソッドを提供します。同じように C 言語では、関数の共通のインプリメンテーションを FmcjItem から取得します。つまり、それらの関数には共通の接頭部 FmcjItem が付いており、定義においては先頭に接頭部 FmcjWorkitem、FmcjActivityInstanceNotification、および FmcjProcessInstanceNotification が付けられます。 ActiveX では継承がサポートされていないため、すべてのメソッドは該当するクラス上で明示的に定義されます。しかし、それらのメソッドは Item アクションとして記述される点に注意してください。

アイテムは、オブジェクトの識別子によって一意に識別されます。

以下の部分では、アイテムに適用できるアクションについて説明します。関数 / メソッドの完全なリストについては、306ページの『Item』を参照してください。

Delete()

この関数 / メソッドは、指定したアイテムを MQSeries Workflow 実行サーバーで削除します (アクション呼び出し)。

通知はいつでも削除できます。作業項目は、*Ready*、*Finished*、*ForceFinished*、または *Disabled* のいずれかの状態でなければなりません。作業項目が *Ready (作動可能)* 状態であり、アクティビティー・インスタンスに関連した唯一の作業を表している場合、関連しているプロセス・インスタンスが *Terminating (中止処理中)* または *Terminated (中止済み)* でなければ、削除は拒否されます。

アイテムの一時的な表示には影響がありません。 C および C++ では、一時オブジェクトが必要なくなった時点で破棄または割り振り解除する必要があります。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

アイテムの所有者であること

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Item

ActiveX のシグニチャー

```
long ActivityInstanceNotification.Delete()
long ProcessInstanceNotification.Delete()
long Workitem.Delete()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjItemDelete( FmcjItemHandle hdlItem )
#define FmcjActivityInstanceNotificationDelete FmcjItemDelete
#define FmcjProcessInstanceNotificationDelete  FmcjItemDelete
#define FmcjWorkitemDelete                     FmcjItemDelete
```

C++ 言語のシグニチャー

```
APIRET Delete()
```

Java のシグニチャー

```
public abstract
void   delete() throws FmcException
```

パラメーター

hdlItem 入力。削除するアイテムのハンドル。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
アイテムはもう存在していません。

FMC_ERROR_NOT_ALLOWED(507)
アイテムはアクティビティー・インスタンスに関連した作業を表しています。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_STATE(120)
アイテムが間違った状態です。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

ObtainProcessInstanceMonitor()/ ObtainInstanceMonitor

この関数 / メソッドは、アイテムが属するプロセス・インスタンスのプロセス・インスタンス・モニターを MQSeries Workflow 実行サーバーから取り出します。

deep オプションを指定すると、ブロック型のアクティビティー・インスタンスが解決され、それらのブロック・インスタンス・モニターもサーバーから取り出されます。

注: 現時点では、deep はサポートされていません。

C++ の場合、初期化するプロセス・インスタンス・モニター・オブジェクトが空でなければ、そのオブジェクトがまず破棄されてから、新しいオブジェクトが割り当てられます。 C の場合は、オブジェクトの所有権についてはアプリケーションの側が完全に制御することになります。つまり、プロセス・インスタンス・モニターのハンドルがすでに何らかのオブジェクトを指しているかどうかのチェックは実行されません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- プロセスの許可
- プロセス管理の許可

- プロセス管理者として
- プロセス作成者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.Item

ActiveX のシグニチャー

```
InstanceMonitor*
ActivityInstanceNotification.ObtainInstanceMonitor(
                             long *    returnCode,
                             boolean   deep      )
InstanceMonitor*
ProcessInstanceNotification.ObtainInstanceMonitor(
                             long *    returnCode,
                             boolean   deep      )
InstanceMonitor*
Workitem.ObtainInstanceMonitor(
                             long *    returnCode,
                             boolean   deep      )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjItemObtainProcessInstanceMonitor(
       FmcjItemHandle                       hdlItem,
       bool                                 deep,
       FmcjProcessInstanceMonitorHandle *   monitor)
#define FmcjActivityInstanceNotificationObtainProcessInstanceMonitor
        FmcjItemObtainProcessInstanceMonitor
#define FmcjProcessInstanceNotificationObtainProcessInstanceMonitor
        FmcjItemObtainProcessInstanceMonitor
#define FmcjWorkitem
        FmcjItemObtainProcessInstanceMonitor
```

C++ 言語のシグニチャー

```
APIRET ObtainProcessInstanceMonitor(
       FmcjProcessInstanceMonitor &    monitor,
       bool                            deep= false ) const
```

Java のシグニチャー

```
public abstract
ProcessInstanceMonitor obtainProcessInstanceMonitor(
                       boolean deep ) throws FmcException
```

パラメーター

deep 入力。ブロック型のアクティビティー・インスタンスを解決するかどうか (つまりそのモニターも取り出すかどうか) を指定する標識。ただし、現時点で deep は無視されます。

hdlItem 入力。プロセス・インスタンス・モニターを取り出すアイテム。

monitor 入出力。設定するプロセス・インスタンス・モニターオブジェクトまたはプロセス・インスタンス・モニターオブジェクトのハンドルのアドレス。

returnCode 入出力。このメソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻り値のデータ型

APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ProcessInstanceMonitor*/ ProcessInstanceMonitor
プロセス・インスタンス・モニターへのポインター、またはアイテムの属するプロセス・インスタンス・モニター。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
アイテムはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

ProcessInstance()

この関数 / メソッドは、アイテムが属するプロセス・インスタンスを MQSeries Workflow 実行サーバーから取り出します。

プロセス・インスタンスに関するすべての 1 次情報および 2 次情報が取り出されます。

C++ の場合、初期化するプロセス・インスタンス・オブジェクトが空でなければ、そのオブジェクトがまず破棄されてから、新しいオブジェクトが割り当てられます。 C の場合は、オブジェクトの所有権についてはアプリケーション

の側が完全に制御することになります。つまり、プロセス・インスタンスのハンドルがすでに何らかのオブジェクトを指しているかどうかのチェックは実行されません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセスの許可
- プロセス管理の許可
- プロセス作成者として
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Item

ActiveX のシグニチャー

```
ProcessInstance*
ActivityInstanceNotification.ProcessInstance( long * returnCode)
ProcessInstancer*
ProcessInstanceNotification.ProcessInstance( long * returnCode)
ProcessInstance*
Workitem.ProcessInstance( long * returnCode )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjItemProcessInstance(
                              FmcjItemHandle  hdlItem,
       FmcjProcessInstanceHandle *  instance )
#define FmcjActivityInstanceNotificationProcessInstance
        FmcjItemProcessInstance
#define FmcjProcessInstanceNotificationProcessInstance
        FmcjItemProcessInstance
#define FmcjWorkitemProcessInstance
        FmcjItemProcessInstance
```

C++ 言語のシグニチャー

```
APIRET ProcessInstance( FmcjProcessInstance & instance ) const
```

Java のシグニチャー

```
public abstract
ProcessInstance processInstance() throws FmcException
```

パラメーター

hdlItem 入力。照会するアイテム・オブジェクトのハンドル。

instance 入出力。検索 (初期化) するプロセス・インスタンス・オブジェクト。

returnCode 入出力。このメソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻り値のデータ型

APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ProcessInstance*/ ProcessInstance

プロセス・インスタンスへのポインター、またはアイテムの属するプロセス・インスタンス。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
アイテムはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Refresh()

この関数 / メソッドは、MQSeries Workflow 実行サーバーからアイテムをリフレッシュします (アクション呼び出し)。

アイテムに関するすべての 1 次情報および 2 次情報が取り出されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- アイテムの所有者であること
- 作業項目許可
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Item

ActiveX のシグニチャー

```
long ActivityInstanceNotification.Refresh()
long ProcessInstanceNotification.Refresh()
long Workitem.Refresh()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjItemRefresh( FmcjItemHandle hdlItem )
#define FmcjActivityInstanceNotificationRefresh FmcjItemRefresh
#define FmcjProcessInstanceNotificationRefresh  FmcjItemRefresh
#define FmcjWorkitemRefresh                     FmcjItemRefresh
```

C++ 言語のシグニチャー

```
APIRET Refresh()
```

Java のシグニチャー

```
public abstract
void   refresh() throws FmcException
```

パラメーター

hdlItem 入力。リフレッシュするアイテム・オブジェクトのハンドル。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
アイテムはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

SetDescription()

この関数 / メソッドは、アイテムの記述を、指定した値に設定します (アクション呼び出し)。

記述が指定されていない場合、記述の説明は関連したアクティビティー・インスタンスまたはプロセス・インスタンスの記述にリセットされます。

アイテム記述を指定する場合には、以下の規則が適用されます。

- 最大 254 文字 (バイト) を指定できます。
- 現行のロケールに応じて、復帰文字と改行文字を含むあらゆる印刷可能文字を使用できます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

アイテムの所有者であること

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Item

ActiveX のシグニチャー

```
long ActivityInstanceNotification.SetDescription(
                                  BSTR       description,
                                  boolean    isNull     )
long ProcessInstanceNotification.SetDescription(
                                  BSTR       description,
                                  boolean    isNull     )
long Workitem.SetDescription(     BSTR      description,
                                  boolean    isNull     )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjItemSetDescription(
                            FmcjItemHandle  hdlItem,
                            char const *    description )
#define FmcjActivityInstanceNotificationSetDescription
        FmcjItemSetDescription
#define FmcjProcessInstanceNotificationSetDescription
        FmcjItemSetDescription
#define FmcjWorkitemSetDescription
        FmcjItemSetDescription
```

C++ 言語のシグニチャー

```
APIRET SetDescription( string const * description )
```

Java のシグニチャー

```
public abstract
void   setDescription( String description ) throws FmcException
```

パラメーター

description 入力。設定する記述、または記述へのポインター。 NULL ポインター、またはヌル・オブジェクト (Java) も可。

hdlItem 入力。記述を設定するアイテム・オブジェクトのハンドル。

isNull 入力。 *True* に設定されていれば、アイテムの記述をすべてリセットすることを示します。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
アイテムはもう存在していません。

FMC_ERROR_INVALID_DESCRIPTION(810)
この記述は構文規則に従っていません。

FMC_ERROR_NOT_AUTHORIZED(119)
許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

SetName()

この関数 / メソッドは、アイテムの名前を設定します (アクション呼び出し)。

名前が設定されていなければ、アイテムの名前は省略時値 (つまりアクティビティー・インスタンス名またはプロセス・インスタンス名) にリセットされます。

作業項目またはアクティビティー・インスタンス通知の名前を指定する場合には、以下の規則が適用されます。

- 最大 32 文字 (バイト) を指定できます。
- 現行のロケールに応じて、以下のものを除くあらゆる印刷可能文字を使用できます。

  ! " ' ( ) * + , - . / : ; < = > [¥] ^
- ブランクも使用できますが、先行ブランクを使わない、末尾ブランクを使わない、複数のブランクを連続して使わない、という制限があります。
- 数字から始めることはできません。
- 使用できないキーワード。
  AND、OR、NOT、IS、NULL、MOD、LOWER、UPPER、VALUE、SUBSTR、_BLOCK は使用できません。

プロセス・インスタンス通知の名前を指定する場合には、以下の規則が適用されます

- 最大 63 文字 (バイト) を指定できます。
- 現行のロケールに応じて、以下のものを除くあらゆる印刷可能文字を使用できます。

  * ? " ; : . $
- ブランクも使用できますが、先行ブランクを使わない、末尾ブランクを使わない、複数のブランクを連続して使わない、という制限があります。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

アイテムの所有者であること

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.Item

ActiveX のシグニチャー

```
long ActivityInstanceNotification.SetName( BSTR name )
long ProcessInstanceNotification.SetName( BSTR name )
long Workitem.SetName( BSTR name )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjItemSetName( FmcjItemHandle  hdlItem,
                                     char const *    name )
#define FmcjActivityInstanceNotificationSetName FmcjItemSetName
#define FmcjProcessInstanceNotificationSetName  FmcjItemSetName
#define FmcjWorkitemSetName                     FmcjItemSetName
```

C++ 言語のシグニチャー

```
APIRET SetName( string const * name )
```

Java のシグニチャー

```
public abstract
void   setName( String name ) throws FmcException
```

パラメーター

hdlItem 入力。処理するアイテムのハンドル。

name 入力。アイテムの名前。 NULL (0) ポインターまたはヌル・オブジェクト (Java) も可。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
アイテムはもう存在していません。

FMC_ERROR_INVALID_NAME(134)
この名前は構文規則に従っていません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Transfer()

この関数 / メソッドは、指定されたユーザーにアイテムを転送します (アクション呼び出し)。

通知は常に転送可能です。作業項目の状態は *Ready (作動可能)* 、*InError (エラー)* 、*Executed (実行済み)* 、*Suspending (一時中断処理中)* 、*Suspended (一時中断済み)* 、または *Terminated (中止済み)* であり、関連するプロセス・インスタンスの状態は *Running (実行中)* 、*Suspending (一時中断処理中)* 、または *Suspended (一時中断済み)* のいずれかでなければなりません。 InError (エラー) または Terminated (中止済み) の状態の作業項目だけがプロセス管理者に転送できます。

アイテムを転送するユーザーは、アイテムの所有者であるか、またはアイテムの所有者の作業項目許可を持っていなければならず、なおかつ新しい所有者に対する作業項目許可を持っている必要があります。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- 転送元または転送先のユーザーに対する Workitem 権限
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Item

ActiveX のシグニチャー

```
long ActivityInstanceNotification.Transfer( BSTR userID )
long ProcessInstanceNotification.Transfer( BSTR userID )
long Workitem.Transfer( BSTR userID )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjItemTransfer( FmcjItemHandle hdlItem,
                                      char const *   userID )
#define FmcjActivityInstanceNotificationTransfer FmcjItemTransfer
#define FmcjProcessInstanceNotificationTransfer  FmcjItemTransfer
#define FmcjWorkitemTransfer                     FmcjItemTransfer
```

C++ 言語のシグニチャー

```
APIRET Transfer( string const & userID )
```

Java のシグニチャー

```
public abstract
void   transfer( String userID ) throws FmcException
```

パラメーター

hdlItem 入力。転送するアイテム・オブジェクトのハンドル。

userID 入力。アイテムの転送先となるユーザーのユーザー ID。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
アイテムはもう存在していません。

FMC_ERROR_NEW_OWNER_ABSENT(110)
アイテム転送先のユーザーが不在で、アイテムは転送されませんでした。

FMC_ERROR_NEW_OWNER_NOT_FOUND(107)
アイテム転送先のユーザーが不明です。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_OWNER_ALREADY_ASSIGNED(133)
アイテム転送先のユーザーには、すでにそのアイテムがあります。

FMC_ERROR_WRONG_STATE(120)
アイテムまたはプロセス・インスタンスが間違った状態です。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

第45章 永続リストのアクション

FmcjPersistentList または PersistentList オブジェクトは、ユーザーが許可されているのと同じタイプからなるオブジェクトのセットを表します。さらに、このリストによってアクセスできるすべてのオブジェクトは特性が同じです。その特性はフィルターによって指定されます。また、ソート基準を適用することができ、その後、サーバーからクライアントへと転送されるオブジェクトの数を制限するしきい値を適用できます。

名前から分かるように、リストの定義は永続的に保管されます。しかし、リストに含まれるオブジェクトは、照会時に動的に集められたものです。

永続リストとしては、プロセス・テンプレート・リスト、プロセス・インスタンス・リスト、またはワークリストが可能です。

FmcjPersistentList または PersistentList オブジェクトは、リストの共通の側面を表します。 C++ 言語の場合、FmcjPersistentList は FmcjProcessInstanceList クラス、FmcjProcessTemplateList クラス、および FmcjWorklist クラスのスーパークラスであり、すべての共通のプロパティーとメソッドを提供します。 Java 言語の場合、PersistentList は ProcessInstanceList クラス、ProcessTemplateList クラス、および Worklist クラスのスーパークラスであり、すべての共通のプロパティーとメソッドを提供します。同じように C 言語では、関数の共通のインプリメンテーションを FmcjPersistentList から取得します。つまり、共通関数には接頭部 FmcjPersistentList が付いており、定義においては先頭に接頭部 FmcjProcessInstanceList、FmcjProcessTemplateList、および FmcjWorklist が付けられます。 ActiveX では継承がサポートされていないため、すべてのメソッドは該当するクラス上で明示的に定義されます。しかし、それらのメソッドは PersistentList アクションとして記述される点に注意してください。

永続リストは、その名前、タイプ、所有者によって一意に識別されます。これは、一般的なアクセスを目的として定義することができます。その場合にはパブリック *(public)* タイプとなります。あるいは、特定の一部のユーザーが使用するようにも定義できます。その場合はプライベート *(private)* タイプとなります。

以下の部分では、永続リストに適用できるアクションについて説明します。関数 / メソッドの完全なリストについては、310ページの『永続リスト』を参照してください。

Delete()

この関数 / メソッドは、指定した永続リストを MQSeries Workflow 実行サーバーから削除します (アクション呼び出し)。

永続リストの一時的な表示には影響がありません。 C および C++ では、一時オブジェクトが必要なくなった時点で破棄または割り振り解除する必要があります。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- リストの所有者であること
- スタッフ定義
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.PersistentList

ActiveX のシグニチャー

```
long ProcessInstanceList.Delete()
long ProcessTemplateList.Delete()
long Worklist.Delete()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjPersistentListDelete( FmcjPersistentListHandle hdlList )
#define FmcjProcessInstanceListDelete FmcjPersistentListDelete
#define FmcjProcessTemplateListDelete FmcjPersistentListDelete
#define FmcjWorklistDelete            FmcjPersistentListDelete
```

C++ 言語のシグニチャー

```
APIRET Delete()
```

Java のシグニチャー

```
public abstract
void   delete() throws FmcException
```

パラメーター

hdlList 入力。削除する永続リストのハンドル。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_DOES_NOT_EXIST(118)
永続リストはもう存在していません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Refresh

この関数 / メソッドは、MQSeries Workflow 実行サーバーから永続リストをリフレッシュします (アクション呼び出し)。

永続リストに関するすべての情報 (たとえば、記述、フィルター、ソート基準) が検索されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- リストの所有者であること
- スタッフ定義
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.PersistentList

ActiveX のシグニチャー

```
long ProcessInstanceList.Refresh()
long ProcessTemplateList.Refresh()
long Worklist.Refresh()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjPersistentListRefresh( FmcjPersistentListHandle hdlList )
#define FmcjProcessInstanceListRefresh FmcjPersistentListRefresh
#define FmcjProcessTemplateListRefresh FmcjPersistentListRefresh
#define FmcjWorklistRefresh            FmcjPersistentListRefresh
```

C++ 言語のシグニチャー

```
APIRET Refresh()
```

Java のシグニチャー

```
public abstract
void   refresh() throws FmcException
```

パラメーター

hdlList 入力。リフレッシュする永続リストのハンドル。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
永続リストはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

SetDescription()

この関数 / メソッドは、永続リストの記述を、指定した値に設定します (アクション呼び出し)。

記述が指定されていないなら、この永続リストの記述が削除されます。

永続リストの記述を指定する場合には、以下の規則が適用されます。

- 最大 254 文字 (バイト) を指定できます。
- 現行のロケールに応じて、復帰文字と改行文字を含むあらゆる印刷可能文字を使用できます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- リストの所有者であること
- スタッフ定義
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.PersistentList

ActiveX のシグニチャー

```
long ProcessInstanceList.SetDescription(
                         BSTR         description,
                         boolean      isNull     )
long ProcessTemplateList.SetDescription(
                         BSTR         description,
                         boolean      isNull     )
long Worklist.SetDescription(
                         BSTR         description,
                         boolean      isNull     )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjPersistentListSetDescription( FmcjPersistentListHandle hdlList,
                                  char const *             description )
#define FmcjProcessInstanceListSetDescription
        FmcjPersistentListSetDescription
#define FmcjProcessTemplateListSetDescription
        FmcjPersistentListSetDescription
#define FmcjWorklistSetDescription
        FmcjPersistentListSetDescription
```

C++ 言語のシグニチャー

```
APIRET SetDescription( string const * description )
```

Java のシグニチャー

```
public abstract
void   setDescription( String description ) throws FmcException
```

パラメーター

description 入力。設定する記述、または記述へのポインター。 NULL ポインター、またはヌル・オブジェクト (Java) も可。

hdlList 入力。記述を設定する永続リスト・オブジェクトのハンドル。

isNull 入力。 *True* に設定されていれば、記述をすべて削除することを示します。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
永続リストはもう存在していません。

FMC_ERROR_INVALID_DESCRIPTION(810)
この記述は構文規則に従っていません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

SetFilter()

この関数 / メソッドは、永続リストのフィルターを、指定した値に設定します (アクション呼び出し)。

フィルターが指定されていないなら、この永続リストのフィルターが削除されます。つまり、許可されているすべてのオブジェクトがこのリストによって選択されます。

有効なフィルター構文の説明については、該当するリスト作成の部分を参照してください。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- リストの所有者であること
- スタッフ定義
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.PersistentList

ActiveX のシグニチャー

```
long ProcessInstanceList.SetFilter(
                          BSTR         filter,
                          boolean      isNull     )
long ProcessTemplateList.SetFilter(
                          BSTR         filter,
                          boolean      isNull     )
long Worklist.SetFilter( BSTR         filter,
                          boolean      isNull     )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjPersistentListSetFilter( FmcjPersistentListHandle  hdlList,
                             char const *              filter )
#define FmcjProcessInstanceListSetFilter FmcjPersistentListSetFilter
#define FmcjProcessTemplateListSetFilter FmcjPersistentListSetFilter
#define FmcjWorklistSetFilter            FmcjPersistentListSetFilter
```

C++ 言語のシグニチャー

```
APIRET SetFilter( string const * filter )
```

Java のシグニチャー

```
public abstract
void   setFilter( String filter ) throws FmcException
```

パラメーター

filter 入力。設定するフィルター、またはフィルターへのポインター。 NULL ポインター、またはヌル・オブジェクト (Java) も可。

hdlList 入力。フィルターを設定する永続リスト・オブジェクトのハンドル。

isNull 入力。 *True* に設定されていれば、フィルターをすべて削除することを示します。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
永続リストはもう存在していません。

FMC_ERROR_INVALID_FILTER(125)
指定したフィルターは無効です。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

SetSortCriteria()

この関数 / メソッドは、永続リストのソート基準を、指定した値に設定します (アクション呼び出し)。

ソート基準を指定しない場合、永続リストの現在のソート基準が削除されます。つまり、このリストによって選択されるオブジェクトはソートされなくなります。

有効なソート基準の構文の説明については、該当するリスト作成の部分を参照してください。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- リストの所有者であること
- スタッフ定義
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.PersistentList

ActiveX のシグニチャー

```
long ProcessInstanceList.SetSortCriteria(
                         BSTR         sortCriteria,
                         boolean      isNull     )
long ProcessTemplateList.SetSortCriteria(
                         BSTR         sortCriteria,
                         boolean      isNull     )
long Worklist.SetSortCriteria(
                         BSTR         sortCriteria,
                         boolean      isNull     )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjPersistentListSetSortCriteria( FmcjPersistentListHandle hdlList,
                                   char const *             sortCriteria )
#define FmcjProcessInstanceListSetSortCriteria
        FmcjPersistentListSetSortCriteria
#define FmcjProcessTemplateListSetSortCriteria
        FmcjPersistentListSetSortCriteria
#define FmcjWorklistSetSortCriteria
        FmcjPersistentListSetSortCriteria
```

C++ 言語のシグニチャー

```
APIRET SetSortCriteria( string const * sortCriteria )
```

Java のシグニチャー

```
public abstract
void   setSortCriteria( String sortCriteria ) throws FmcException
```

パラメーター

hdlList 入力。ソート基準を設定する永続リスト・オブジェクトのハンドル。

sortCriteria 入力。設定するソート基準、またはソート基準へのポインター。 NULL ポインター、またはヌル・オブジェクト (Java) も可。

isNull 入力。 *True* に設定されていれば、ソート基準をすべて削除することを示します。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
永続リストはもう存在していません。

FMC_ERROR_INVALID_SORT(808)
指定したソート基準は無効です。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

SetThreshold()

この関数 / メソッドは、永続リストのしきい値を、指定した値に設定します (アクション呼び出し)。

しきい値が指定されていないなら、この永続リストのしきい値が削除されます。つまり、リストに含まれているすべてのオブジェクトが、照会時に提供されることになります。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- リストの所有者であること
- スタッフ定義
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.PersistentList

ActiveX のシグニチャー

```
long ProcessInstanceList.SetThreshold(
                         long         threshold,
                         boolean      isNull     )
long ProcessTemplateList.SetThreshold(
                         long         threshold,
                         boolean      isNull     )
long Worklist.SetThreshold(
                         long         threshold,
                         boolean      isNull     )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjPersistentListSetThreshold( FmcjPersistentListHandle hdlList,
                                unsigned long const *    threshold )
#define FmcjProcessInstanceListSetThreshold  FmcjPersistentListSetThreshold
#define FmcjProcessITemplateListSetThreshold FmcjPersistentListSetThreshold
#define FmcjWorklistSetThreshold             FmcjPersistentListSetThreshold
```

C++ 言語のシグニチャー

```
APIRET SetThreshold( unsigned long const * threshold )
```

Java のシグニチャー

```
public abstract
void   setThreshold( Integer threshold ) throws FmcException
```

パラメーター

hdlList 入力。しきい値を設定する永続リスト・オブジェクトのハンドル。

threshold 入力。設定するしきい値、またはしきい値へのポインター。NULL ポインター、またはヌル・オブジェクト (Java) も可。

isNull 入力。 *True* に設定されていれば、しきい値をすべて削除することを示します。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
永続リストはもう存在していません。

FMC_ERROR_INVALID_THRESHOLD(807)
指定したしきい値は無効です。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

第46章 担当者のアクション

FmcjPerson または Person (担当者) オブジェクトは、MQSeries Workflow のユーザーを表します。担当者は、そのユーザー識別子によって一意的に識別されます。

以下の部分では、担当者に適用できるアクションについて説明します。関数 / メソッドの完全なリストについては、311ページの『担当者』を参照してください。

Refresh()

この関数 / メソッドは、MQSeries Workflow 実行サーバーから担当者をリフレッシュします (アクション呼び出し)。

担当者に関するすべての 1 次情報および 2 次情報が取り出されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Person

ActiveX のシグニチャー

```
long Refresh()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjPersonRefresh( FmcjPersonHandle hdlPerson )
```

C++ 言語のシグニチャー

```
APIRET Refresh()
```

Java のシグニチャー

```
public abstract
void   refresh() throws FmcException
```

パラメーター

hdlPerson 入力。リフレッシュする担当者のハンドル。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

SetAbsence()

この関数 / メソッドは、ログオン・ユーザーの不在指示を、指定した値に設定します (アクション呼び出し)。

不在の担当者はスタッフ決定に参加しません。つまり、どの作業項目も割り当てられません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.Person

ActiveX のシグニチャー

```
long SetAbsence( boolean newValue )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjPersonSetAbsence(
       FmcjPersonHandle  hdlPerson,
       bool              newValue )
```

C++ 言語のシグニチャー

```
APIRET SetAbsence( bool newValue )
```

Java のシグニチャー

```
public abstract
void   setAbsence( boolean newValue ) throws FmcException
```

パラメーター

hdlPerson 入力。不在を設定する担当者オブジェクトのハンドル。

newValue 入力。担当者が不在の場合は真、それ以外の場合は偽。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

SetSubstitute()

この関数 / メソッドは、ログオンしているユーザーの代行者を設定します (アクション呼び出し)。

代行者は、ログオン・ユーザー以外の登録済み MQSeries Workflow ユーザー ID でなければなりません。代行者を指定しない場合、ログオン・ユーザーの代行者が削除されます。

注: 代行者を変更すると、ログオン・ユーザーの (ワーク・) アイテムにアクセスする権限のある担当者が変更されてしまいます。更新された定義を読み取るには、担当者オブジェクトをリフレッシュする必要があります。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.Person

ActiveX のシグニチャー

```
long SetSubstitute( BSTR substitute, boolean isNull )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjPersonSetSubstitute(
                    FmcjPersonHandle  hdlPerson,
                    char const *      substitute )
```

C++ 言語のシグニチャー

```
APIRET SetSubstitute( string const * substitute )
```

Java のシグニチャー

```
public abstract
void   setSubstitute( String substitute ) throws FmcException
```

パラメーター

hdlPerson 入力。代行者を設定する担当者オブジェクトのハンドル。

isNull 入力。 *True* に設定した場合、代行ユーザーの指定はすべて削除されます。

substitute 入力。設定する代行者、または代行者へのポインター。 NULL ポインター、またはヌル・オブジェクト (Java) も可。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_INVALID_USER(132)
指定されたユーザー ID は構文規則に対応していません。あるいは、同じユーザーが同時にログオン・ユーザーと代行者の両方になることはできません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_USERID_UNKNOWN(10)
指定されたユーザー ID は、登録済みの MQSeries Workflow ユーザー ID ではありません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

第47章 プロセス・インスタンスのアクション

FmcjProcessInstance または ProcessInstance オブジェクトは、プロセス・テンプレートのインスタンスを表します。プロセス・インスタンスは、そのオブジェクト識別子または名前によって一意的に識別されます。プロセス・インスタンス作成時の保持オプションに従って、固有のプロセス・インスタンス名がユーザーが提供されているか、または MQSeries Workflow によって生成されています。

以下の図は、可能性のあるプロセス・インスタンスの状態と、適切な許可が与えられている場合および関数 / メソッド記述に示されているさらに限定的な要件が満たされた場合に、そのような状態のもとで可能なアクションをまとめています。

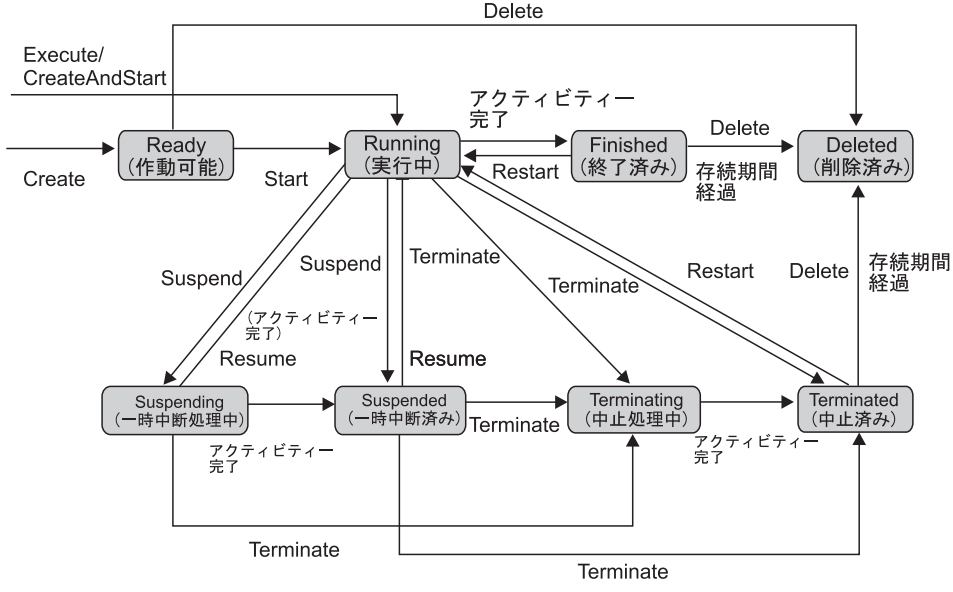


図 10. プロセス・インスタンスの状態

以下の部分では、プロセス・インスタンスに適用できるアクションについて説明します。関数 / メソッドの完全なリストについては、318ページの『プロセス・インスタンス』を参照してください。

Delete()

この関数 / メソッドは、指定したプロセス・インスタンスを MQSeries Workflow 実行サーバーから削除します (アクション呼び出し)。

プロセス・インスタンスは最上位プロセスでなければならず、その状態は *Ready (作動状態)*、*Finished (終了済み)*、または *Terminated (中止済み)* でなければなりません。プロセス・インスタンスがまだ開始していない間は、作成者がこれを削除できます。

プロセス・インスタンスの一時的な表示には影響がありません。 C および C++ では、一時オブジェクトが必要なくなった時点で破棄または割り振り解除する必要があります。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- プロセス管理の許可
- プロセス・インスタンス作成者として
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX のシグニチャー

```
long Delete()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjProcessInstanceDelete( FmcjProcessInstanceHandle hdlInstance )
```

C++ 言語のシグニチャー

```
APIRET Delete()
```

Java のシグニチャー

```
public abstract
void   delete() throws FmcException
```

パラメーター

hdlInstance 入力。削除するプロセス・インスタンスのハンドル。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・インスタンスはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_STATE(120)
プロセス・インスタンスが間違った状態です。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

InContainer()

この関数 / メソッドは、プロセス・インスタンスに関連する入力コンテナーを、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

C++ の場合、初期化するコンテナー・オブジェクトが空でなければ、そのオブジェクトがまず破棄されてから、新しいオブジェクトが割り当てられます。 C の場合は、オブジェクトの所有権についてはアプリケーションの側が完全に制御することになります。つまり、コンテナー・ハンドルがすでに何らかのオブジェクトを指しているかどうかのチェックは実行されません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- プロセスの許可

- プロセス管理の許可
- プロセス・インスタンス作成者として
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ProcessInstance

ActiveX のシグニチャー

```
long InContainer( Container * input )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjProcessInstanceInContainer(
       FmcjProcessInstanceHandle      hdlInstance,
       FmcjReadWriteContainerHandle * input )
```

C++ 言語のシグニチャー

```
APIRET InContainer( FmcjReadWriteContainer & input )
```

Java のシグニチャー

```
public abstract
ReadWriteContainer inContainer() throws FmcException
```

パラメーター

hdlInstance 入力。入力コンテナーを検索するプロセス・インスタンス・オブジェクトのハンドル。

input 入出力。設定するプロセス・インスタンスの入力コンテナー・ハンドルのアドレス (C)、またはその入力コンテナーのアドレス (C++)。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

ReadWriteContainer
プロセス・インスタンスの入力コンテナー。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・インスタンスはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

ObtainMonitor()

この関数 / メソッドは、プロセス・インスタンスのモニターを MQSeries Workflow 実行サーバーから取り出します (アクション呼び出し)。

deep オプションを指定すると、ブロック型のアクティビティー・インスタンスが解決され、それらのブロック・インスタンス・モニターもサーバーから取り出されます。

注: 現時点では、deep はサポートされていません。

C++ の場合、初期化するプロセス・インスタンス・モニター・オブジェクトが空でなければ、そのオブジェクトがまず破棄されてから、新しいオブジェクトが割り当てられます。 C の場合は、オブジェクトの所有権についてはアプリケーションの側が完全に制御することになります。つまり、プロセス・インスタンス・モニターのハンドルがすでに何らかのオブジェクトを指しているかどうかのチェックは実行されません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセスの許可
- プロセス管理の許可
- プロセス・インスタンス作成者として
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ProcessInstance

ActiveX のシグニチャー

```
InstanceMonitor* ObtainMonitor( long *    returnCode,
                                boolean       deep )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjProcessInstanceObtainMonitor(
       FmcjProcessInstanceHandle          hdlInstance,
       bool                               deep,
       FmcjProcessInstanceMonitorHandle * monitor )
```

C++ 言語のシグニチャー

```
APIRET ObtainMonitor( FmcjProcessInstanceMonitor & monitor,
                      bool                         deep= false )
```

Java のシグニチャー

```
public abstract
ProcessInstanceMonitor obtainMonitor( boolean deep ) throws FmcException
```

パラメーター

deep 入力。ブロック型のアクティビティー・インスタンスを解決するかどうか (つまりそのモニターも取り出すかどうか) を指定する標識。ただし、現時点で deep は無視されます。

hdlInstance 入力。モニターを検索するプロセス・インスタンス・オブジェクトのハンドル。

monitor 入出力。設定するプロセス・インスタンスのモニター・ハンドルのアドレス (C)、またはそのモニターのアドレス (C++)。

returnCode 入出力。メソッド呼び出しの結果へのポインター。下記の戻りコードを参照してください。

戻り値のデータ型

APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

InstanceMonitor*/ProcessInstanceMonitor
プロセス・インスタンス・モニターまたはプロセス・インスタンス・モニターへのポインター。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・インスタンスはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

OutContainer()

この関数 / メソッドは、プロセス・インスタンスに関連する出力コンテナーを、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

C++ の場合、初期化するコンテナー・オブジェクトが空でなければ、そのオブジェクトがまず破棄されてから、新しいオブジェクトが割り当てられます。 C の場合は、オブジェクトの所有権についてはアプリケーションの側が完全に制御することになります。つまり、コンテナー・ハンドルがすでに何らかのオブジェクトを指しているかどうかのチェックは実行されません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセスの許可
- プロセス管理の許可
- プロセス・インスタンス作成者として
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX のシグニチャー

```
long OutContainer( Container * output )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjProcessInstanceOutContainer(
       FmcjProcessInstanceHandle     hdlInstance,
       FmcjReadOnlyContainerHandle * output )
```

C++ 言語のシグニチャー

```
APIRET OutContainer( FmcjReadOnlyContainer & output ) const
```

Java のシグニチャー

```
public abstract
ReadOnlyContainer outContainer() throws FmcException
```

パラメーター

hdlInstance 入力。出力コンテナーを検索するプロセス・インスタンス・オブジェクトのハンドル。

output 入出力。設定するプロセス・インスタンスの出力コンテナー・ハンドルのアドレス (C)、またはその出力コンテナーのアドレス (C++)。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

ReadOnlyContainer
プロセス・インスタンスの出力コンテナー。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・インスタンスはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

PersistentObject()

この関数 / メソッドは、MQSeries Workflow 実行サーバーから渡されたオブジェクト識別子で識別されるプロセス・インスタンスを取り出します (アクション呼び出し)。

プロセス・インスタンスの検索元の MQSeries Workflow 実行サーバーは、実行サービス・オブジェクトによって識別されます。その場合、一時オブジェクトは、プロセス・インスタンスのすべての情報 (1 次および 2 次) を使って作成また更新されます。

C++ の場合、初期化するプロセス・インスタンス・オブジェクトが空でなければ、そのオブジェクトがまず破棄されてから、新しいオブジェクトが割り当てられます。 C の場合は、オブジェクトの所有権についてはアプリケーションの側が完全に制御することになります。つまり、プロセス・インスタンスのハンドルがすでに何らかのオブジェクトを指しているかどうかのチェックは実行されません。 Java の場合、プロセス・インスタンスは新たに作成されます。実行サービスがファクトリーとして機能します。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセスの許可
- プロセス管理の許可
- プロセス・インスタンス作成者として
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long PersistentObject( ExecutionService * service, BSTR oid )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjProcessInstancePersistentObject(
       FmcjExecutionServiceHandle  service,
       char const *                oid,
       FmcjProcessInstanceHandle * hdlInstance )
```

C++ 言語のシグニチャー

```
APIRET PersistentObject( FmcjExecutionService const & service,
                         string const &               oid )
```

Java のシグニチャー

```
public abstract
  ProcessInstance ExecutionService.persistentObject( String oid )
throws FmcException
```

パラメーター

hdlInstance 入出力。設定するプロセス・インスタンスオブジェクトのハンドルのアドレス。

oid 入力。検索するプロセス・インスタンスのオブジェクト識別子。

service 入力。実行サーバーとのセッションを表すサービス・オブジェクト。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

ProcessInstance
取り出されたプロセス・インスタンス。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・インスタンスはもう存在していません。

FMC_ERROR_INVALID_OID(805)
指定した oid は無効です。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Refresh()

この関数 / メソッドは、MQSeries Workflow 実行サーバーからプロセス・インスタンスをリフレッシュします (アクション呼び出し)。

プロセス・インスタンスに関するすべての 1 次情報および 2 次情報が取り出されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセスの許可
- プロセス管理の許可
- プロセス・インスタンス作成者として
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX のシグニチャー

```
long Refresh()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjProcessInstanceRefresh( FmcjProcessInstanceHandle hdlInstance )
```

C++ 言語のシグニチャー

```
APIRET Refresh()
```

Java のシグニチャー

```
public abstract
void   refresh() throws FmcException
```

パラメーター

hdlInstance 入力。リフレッシュするプロセス・インスタンスオブジェクトのハンドル。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・インスタンスはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Restart()

この関数 / メソッドは、MQSeries Workflow 実行サーバー上のプロセス・インスタンスを再始動します (アクション呼び出し)。

再始動できるのは、*Finished (終了済み)* または *Terminated (中止済み)* の最上位プロセス・インスタンスだけです。プロセス管理者は変更されません。プロセス開始者はこの関数 / メソッドの要求発行者に設定されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセス管理の許可
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX のシグニチャー

```
long Restart()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjProcessInstanceRestart( FmcjProcessInstanceHandle hdlInstance )
```

C++ 言語のシグニチャー

```
APIRET Restart()
```

Java のシグニチャー

```
public abstract
void   restart() throws FmcException
```

パラメーター

hdlInstance 入力。再始動するプロセス・インスタンスオブジェクトのハンドル。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

戻りコード

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・インスタンスはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_KIND(501)
このプロセス・インスタンスは最上位のプロセス・インスタンスではありません。

FMC_ERROR_WRONG_STATE(120)
プロセス・インスタンスが間違った状態です。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Resume()

この関数 / メソッドは、一時中断済み、または一時中断処理中のプロセス・インスタンスの処理を再開します (アクション呼び出し)。

deep オプションが真の場合、制御の自立性に関して自立走行式でないすべてのサブプロセスも再開されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- プロセス管理の許可

- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ProcessInstance

ActiveX のシグニチャー

```
long Resume( boolean deep )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjProcessInstanceResume( FmcjProcessInstanceHandle hdlInstance,
                           bool                      deep )
```

C++ 言語のシグニチャー

```
APIRET Resume( bool deep )
```

Java のシグニチャー

```
public abstract
void   resume( boolean deep ) throws FmcException
```

パラメーター

deep 入力。 deep が真の場合、自立走行式でないすべてのサブプロセスも再開されます。

hdlInstance　入力。再開するプロセス・インスタンスのハンドル。

戻り値のデータ型

long/ APIRET　この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

戻りコード / FmcException

FMC_OK(0)　関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・インスタンスはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_STATE(120)
プロセス・インスタンスが間違った状態です。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

SetDescription()

この関数 / メソッドは、プロセス・インスタンスの記述を、指定した値に設定します (アクション呼び出し)。

記述が指定されていないなら、このプロセス・インスタンスの記述が削除されます。

プロセス・インスタンスの記述を指定する場合には、以下の規則が適用されます。

- 最大 254 文字 (バイト) を指定できます。
- 現行のロケールに応じて、復帰文字と改行文字を含むあらゆる印刷可能文字を使用できます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセスの許可
- プロセス管理の許可
- プロセス・インスタンス作成者として
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX のシグニチャー

```
long SetDescription( BSTR description, boolean isNull )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjProcessInstanceSetDescription(
       FmcjProcessInstanceHandle  hdlInstance,
       char const *               description )
```

C++ 言語のシグニチャー

```
APIRET SetDescription( string const * description )
```

Java のシグニチャー

```
public abstract
void   setDescription( String  description ) throws FmcException
```

パラメーター

description 入力。設定する記述、または記述へのポインター。 NULL ポインター、またはヌル・オブジェクト (Java) も可。

hdlInstance 入力。記述を設定するプロセス・インスタンス・オブジェクトのハンドル。

isNull 入力。 *True* に設定した場合、記述の指定がすべて削除されます。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・インスタンスはもう存在していません。

FMC_ERROR_INVALID_DESCRIPTION(810)
この記述は構文規則に従っていません。

FMC_ERROR_NOT_AUTHORIZED(119)
許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

SetName()

この関数 / メソッドは、プロセス・インスタンスの名前を、指定した値に設定します (アクション呼び出し)。

プロセス・インスタンスは *Ready* 状態のままでなければなりません。

プロセス・インスタンスの名前を指定する場合には、以下の規則が適用されます。

- 最大 63 文字 (バイト) を指定できます。
- 現行のロケールに応じて、以下のものを除くあらゆる印刷可能文字を使用できます。

  ```
  * ? " ; : . $
  ```

- ブランクも使用できますが、先行ブランクを使わない、末尾ブランクを使わない、複数のブランクを連続して使わない、という制限があります。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- プロセスの許可
- プロセス管理の許可
- プロセス・インスタンス作成者として
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX のシグニチャー

```
long SetName( BSTR name )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjProcessInstanceSetName( FmcjProcessInstanceHandle hdlInstance,
                            char const *              name )
```

C++ 言語のシグニチャー

```
APIRET SetName( string const & name )
```

Java のシグニチャー

```
public abstract
void   setName( String name ) throws FmcException
```

パラメーター

hdlInstance 入力。名前を設定するプロセス・インスタンス・オブジェクトのハンドル。

name 入力。設定する名前。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・インスタンスはもう存在していません。

FMC_ERROR_INVALID_NAME(134)
この名前は構文規則に従っていません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_NOT_UNIQUE(121)
プロセス・インスタンスの名前が固有ではありません。

FMC_ERROR_WRONG_STATE(120)
プロセス・インスタンスが間違った状態です。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Start()

この関数 / メソッドは、作動可能状態のプロセス・インスタンスを開始します (アクション呼び出し)

正常に実行された場合、開始者はこのアクションのリクエスターとして設定され、プロセス管理者が決定されます。

開始するプロセス・インスタンスに初期値を渡す場合、入力コンテナーを指定することができます (FmcjProcessInstance::InContainer() も参照)。入力コンテナーの指定なしに、入力を必要とするプロセス・インスタンスが開始された場合、入力コンテナーの値は設定されません。このため、たとえばアクティビテ

ィー・インプリメンテーション内から入力コンテナー値を照会すると、FMC_ERROR_MEMBER_NOT_SET が戻されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセスの許可
- プロセス管理の許可
- プロセス・インスタンス作成者として
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX のシグニチャー

```
long Start()
long StartWithContainer( Container * input )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjProcessInstanceStart( FmcjProcessInstanceHandle    hdlInstance,
                          FmcjReadWriteContainerHandle input )
```

C++ 言語のシグニチャー

```
APIRET Start()
APIRET Start( FmcjReadWriteContainer const & input )
```

Java のシグニチャー

```
public abstract
void   start() throws FmcException
public abstract
void   start2( ReadWriteContainer input ) throws FmcException
```

パラメーター

hdlInstance 入力。開始するプロセス・インスタンスオブジェクトのハンドル。

input 入力。プロセス・インスタンスの入力コンテナー。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・インスタンスはもう存在していません。

FMC_ERROR_INVALID_CONTAINER(509)
渡されたコンテナーがプロセス・インスタンスに有効ではありません。スキーマまたはバージョンが間違っています。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_STATE(120)
プロセス・インスタンスが間違った状態です。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Suspend()

この関数 / メソッドは、プロセス・インスタンスを一時中断 (一時的に停止) します (アクション呼び出し)。

このプロセス・インスタンスは *Running* 状態である必要があります。 deep オプションが真の場合、制御の自立性に関して自立走行式でないすべてのサブプロセスも一時中断されます。自立走行式のサブプロセスは考慮されません。

プロセス・インスタンスは、実行中のプログラムのアクティビティー・インプリメンテーションまたは中断中の非自立走行式サブプロセスが存在している限り、*Suspending* 状態のままです。アクティビティー・インプリメンテーションが実行を完了し、かつ非自立走行式のサブプロセスが *Suspended (一時中断済み)* 状態に達すると、プロセス・インスタンスは *Suspended* 状態になります。

オプションとして、プロセス・インスタンスをいつまで中断するかの日付を指定できます。その後、自動的に再開され、もし deep オプションが指定されていれば自立走行式でないサブプロセスも同時に再開されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセス管理の許可
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX のシグニチャー

```
long Suspend( boolean deep )
long SuspendUntilDateTime( DateAndTime * time,
                           boolean       deep )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjProcessInstanceSuspend(
       FmcjProcessInstanceHandle  hdlInstance,
       bool                       deep )
APIRET FMC_APIENTRY FmcjProcessInstanceSuspendUntil(
       FmcjProcessInstanceHandle  hdlInstance,
       FmcjCDateTime const *      time,
       bool                       deep )
```

C++ 言語のシグニチャー

```
APIRET Suspend( bool deep )
APIRET Suspend( FmcjDateTime const & time, bool deep )
```

Java のシグニチャー

```
public abstract
void   suspend( boolean deep ) throws FmcException
public abstract
void   suspend2( Calendar time, boolean deep ) throws FmcException
```

パラメーター

deep 入力。自立走行式でないサブプロセスが一緒に一時中断されるかどうかを表す標識。

hdlInstance 入力。一時中断するプロセス・インスタンスオブジェクトのハンドル。

time 入力。プロセス・インスタンスをいつまで一時中断するかを表す日付 / 時刻 (C)、またはその日付 / 時刻へのポインター (C++)。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・インスタンスはもう存在していません。
FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。
FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。
FMC_ERROR_WRONG_STATE(120)
プロセス・インスタンスが間違った状態です。
FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。
FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。
FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。
FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Terminate()

この関数 / メソッドは、プロセス・インスタンスおよびそのすべての非自立走行式サブプロセスを中止します (アクション呼び出し)。

プロセス・インスタンスは、*Running*、*Suspended (一時中断済み)*、または *Suspending* 状態でなければなりません。

実行中のアクティビティー・インプリメンテーション、もしくは中止処理中の非自立走行式サブプロセスがあれば、プロセス・インスタンスは Terminating (中止処理中) の状態になります。アクティビティー・インプリメンテーションが実行を完了した場合、または非自立走行式のサブプロセスが中止した場合、プロセス・インスタンスは *Terminated* 状態になります。プロセス・インスタンスが *Terminated (中止済み)* 状態に達すると、『完了したアイテムの削除』オプションの設定値に応じて削除されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセス管理の許可
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstance

ActiveX のシグニチャー

```
long Terminate()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjProcessInstanceTerminate( FmcjProcessInstanceHandle hdlInstance )
```

C++ 言語のシグニチャー

```
APIRET Terminate()
```

Java のシグニチャー

```
public abstract
void   terminate() throws FmcException
```

パラメーター

hdlInstance 入力。中止するプロセス・インスタンスオブジェクトのハンドル。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・インスタンスはもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_STATE(120)
プロセス・インスタンスが間違った状態です。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)

MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)

内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)

戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)

タイムアウトになりました。

第48章 プロセス・インスタンス・リストのアクション

プロセス・インスタンス・リストはプロセス・インスタンスの集まりを表します。このリストによってアクセスできるすべてのプロセス・インスタンスは特性が同じです。その特性はフィルターによって指定されます。また、ソート基準を適用することができ、その後、実行サーバーからクライアントへと転送されるプロセス・インスタンスの数を制限するしきい値を適用できます。

プロセス・インスタンス・リストの定義は永続的に保管されます。

プロセス・インスタンス・リストは、その名前、タイプ、所有者によって一意に識別されます。これは、一般的なアクセスを目的として定義することができます。その場合にはパブリック *(public)* タイプとなります。あるいは、特定の一部のユーザーが使用するようにも定義できます。その場合はプライベート *(private)* タイプとなります。

定義可能なその他のリストには、プロセス・テンプレート・リストとワークリストがあります。 FmcjPersistentList または PersistentList は、すべてのリストの共通プロパティーを表します。

C++ 言語において FmcjProcessInstanceList は FmcjPersistentList クラスのサブクラスであり、すべてのプロパティーとメソッドを継承します。 Java 言語において ProcessInstanceList は PersistentList クラスのサブクラスであり、すべてのプロパティーとメソッドを継承します。同じように C 言語では、関数の共通のインプリメンテーションを FmcjPersistentList から取得します。つまり、共通関数には接頭部 FmcjPersistentList が付いており、定義においては先頭に接頭部 FmcjProcessInstanceList が付けられます。 ActiveX では継承がサポートされていないため、すべてのメソッドは ProcessInstanceList 上で明示的に定義されます。しかし、それらのメソッドは PersistentList アクションとして記述される点に注意してください。

以下の部分では、プロセス・インスタンス・リストに適用できるアクションについて説明します。関数 / メソッドの完全なリストについては、324ページの『プロセス・インスタンス・リスト』を参照してください。

QueryProcessInstances()

この関数 / メソッドは、指定されたプロセス・インスタンス・リストによって特徴付けられるすべてのプロセス・インスタンスの 1 次情報を、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

当てはまるプロセス・インスタンスの集まりの中で、ユーザーに許可が与えられているものだけが検索されます。プロセス・インスタンスが以下のいずれかの条件を満たしている場合、ユーザーはそのプロセス・インスタンスの許可を与えられます。

- どのカテゴリーにも属していない。
- あるカテゴリーに属しており、ユーザーにはそのカテゴリーに関するグローバル・プロセス許可、グローバル・プロセス管理許可、選択プロセス許可、選択プロセス管理許可のいずれかを与えられている。

各プロセス・インスタンスごとに取り出される 1 次情報は以下のとおりです。

- Category
- Description
- Icon
- InContainerNeeded
- LastModificationTime
- LastStateChangeTime
- Name
- ParentName
- ProcessTemplateName
- StartTime
- State
- SuspensionExpirationTime
- SuspensionTime
- SystemName
- SystemGroupName
- TopLevelName

C および C++ では、検索したプロセス・インスタンスはすべて、提供されるプロセス・インスタンス・ベクトルに追加されます。現在プロセス・インスタンス・リストに含まれているプロセス・インスタンスだけを読みたい場合は、まずベクトルをクリアした後でこの関数 / メソッドを呼び出さなければなりません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessInstanceList

ActiveX のシグニチャー

```
long QueryProcessInstances()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjProcessInstanceListQueryProcessInstances(
       FmcjProcessInstanceListHandle      hdlList,
       FmcjProcessInstanceVectorHandle *  instances )
```

C++ 言語のシグニチャー

```
APIRET QueryProcessInstances(
       vector<FmcjProcessInstance> & instances ) const
```

Java のシグニチャー

```
public abstract
ProcessInstance[] queryProcessInstances() throws FmcException
```

パラメーター

hdlList 入力。照会するプロセス・インスタンス・リストのハンドル。

instances 入出力。適格プロセス・インスタンスのベクトル。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

ProcessInstance[]
適格プロセス・インスタンス。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・インスタンス・リストはもう存在していません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- C 言語の例については、833ページの『ワークリストの照会 (C 言語)』を参照してください。
- C++ の例については、835ページの『ワークリストの照会 (C++)』を参照してください。

第49章 プロセス・インスタンス通知のアクション

FmcjProcessInstanceNotification または ProcessInstanceNotification オブジェクトは、あるユーザーに割り当てられたプロセス・インスタンス上での通知を表します

ユーザーに割り当てられているその他のアイテムは、アクティビティー・インスタンス通知および作業項目です。 FmcjItem または Item は、すべてのアイテムの共通プロパティーを表します。

C++ 言語において、FmcjProcessInstanceNotification は FmcjItem クラスのサブクラスであり、すべてのプロパティーとメソッドを継承します。 Java 言語において、ProcessInstanceNotification は Item クラスのサブクラスであり、すべてのプロパティーとメソッドを継承します。同じように C 言語では、関数の共通のインプリメンテーションを FmcjItem から取得します。つまり、共通関数には接頭部 FmcjItem が付いており、定義においては先頭に接頭部 FmcjProcessInstanceNotification が付けられます。 ActiveX では継承がサポートされていないため、すべての関数は ProcessInstanceNotification で明示的に定義されます。しかし、それらのメソッドは Item アクションとして記述される点に注意してください。

プロセス・インスタンス通知は、そのオブジェクト識別子によって一意的に識別されます。

以下の部分では、プロセス・インスタンス通知に適用できるアクションについて説明します。関数 / メソッドの完全なリストについては、326ページの『プロセス・インスタンス通知』を参照してください。

PersistentObject()

この関数 / メソッドは、MQSeries Workflow 実行サーバーから渡されたオブジェクト識別子で識別されるプロセス・インスタンス通知を取り出します (アクション呼び出し)。

プロセス・インスタンス通知の検索元の MQSeries Workflow 実行サーバーは、実行サービス・オブジェクトによって識別されます。その場合、一時オブジェクトは、プロセス・インスタンス通知のすべての情報 (1 次および 2 次) を使って作成また更新されます。

C++ の場合、初期化するプロセス・インスタンス通知・オブジェクトが空でなければ、そのオブジェクトがまず破棄されてから、新しいオブジェクトが割り当てられます。 C の場合は、オブジェクトの所有権についてはアプリケーションの側が完全に制御することになります。つまり、プロセス・インスタンス通知のハンドルがすでに何らかのオブジェクトを指しているかどうかのチェックは実行されません。 Java の場合、プロセス・インスタンス通知は新たに作成されます。実行サービスがファクトリーとして機能します。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- アイテムの所有者であること
- 作業項目許可
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long PersistentObject( ExecutionService * service, BSTR oid )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjProcessInstanceNotificationPersistentObject(
       FmcjExecutionServiceHandle               service,
       char const *                             oid,
       FmcjProcessInstanceNotificationHandle *  hdlItem )
```

C++ 言語のシグニチャー

```
APIRET PersistentObject( FmcjExecutionService const & service,
                         string const &               oid )
```

Java のシグニチャー

```
public abstract
  ProcessInstanceNotification
  ExecutionService.processInstanceNotification( String oid )
throws FmcException
```

パラメーター

hdlItem 入出力。設定するプロセス・インスタンス通知オブジェクトのハンドルのアドレス。

oid 入力。検索するプロセス・インスタンス通知のオブジェクト識別子。

service 入力。実行サーバーとのセッションを表すサービス・オブジェクト。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

ProcessInstanceNotification
取り出されたプロセス・インスタンス通知。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・インスタンス通知はもう存在していません。

FMC_ERROR_INVALID_OID(805)
指定した oid は無効です。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

第50章 プロセス・テンプレートのアクション

FmcjProcessTemplate または ProcessTemplate オブジェクトは、プロセス・モデルの凍結した状態です。このオブジェクトは変換操作によってプロセス・モデルから作成されます。プロセス・モデルが参照するすべてのプログラム定義とデータ構造がプロセス・テンプレートにコピーされます (初期バインド)。サブプロセスは後でバインドされます。サブプロセスの定義は実行中に検索されます。

プロセス・テンプレートは、そのオブジェクト識別子または名前と発行日 (valid-from date) によって一意的に識別されます。この*発効日* は、いつからこのプロセス・テンプレートをプロセス・インスタンスの作成に使えるようになるかということです。

プロセス・テンプレートが実行サーバーから照会されると、その時点で有効なプロセス・テンプレートだけが戻されます。

以下の部分では、プロセス・テンプレートに適用できるアクションについて説明します。関数 / メソッドの完全なリストについては、328ページの『プロセス・テンプレート』を参照してください。

CreateAndStartInstance()

この関数 / メソッドは、指定されたプロセス・テンプレートからプロセス・インスタンスを作成して、そのプロセス・インスタンスを開始します (アクション呼び出し)。

keepName オプションに応じて、プロセス・インスタンス名を指定する必要があります。プロセス・インスタンス名を*現状のまま* にする場合、空文字列を指定することはできません。

プロセス・インスタンスの名前を指定する場合には、以下の規則が適用されます。

- 最大 63 文字 (バイト) を指定できます。
- 現行のロケールに応じて、以下のものを除くあらゆる印刷可能文字を使用できます。

  `* ? " ; : . $`

- ブランクも使用できますが、先行ブランクを使わない、末尾ブランクを使わない、複数のブランクを連続して使わない、という制限があります。

MQSeries Workflow によって固有名が生成され得る場合、次のことが当てはまります。

- プロセス・インスタンス名を指定しない場合、または空の名前を入力した場合は、省略時の名前 *ProcessTemplateName$Oid* でインスタンスが作成されます (*Oid* は、プロセス・インスタンス・オブジェクト識別子の印刷可能バージョン)。プロセス・インスタンス名は 63 バイト以下でなければならないため、プロセス・テンプレート名は短縮されることがあります。
- プロセス・インスタンス名が指定された場合、それが固有である限りその名前は保持されます。指定されたプロセス・インスタンス名が別のインスタンスですでに使用されている場合、名前 *name$Oid* でインスタンスが作成されます (*Oid* は、プロセス・インスタンス・オブジェクト識別子の印刷可能バージョン)。プロセス・インスタンス名は 63 バイト以下でなければならないため、名前は短縮されることがあります。

渡された名前パラメーターの値は変更されません。 FmcjProcessInstance::Name() は、作成されたプロセス・インスタンスの実際の名前を戻します。新しく作成されたプロセス・インスタンスには、1 次属性値だけが入っています。

作成および開始するプロセス・インスタンスに初期値を渡す場合、入力コンテナーを指定することができます。 FmcjProcessInstance::InContainer() も参照してください。入力コンテナーの指定なしに、入力を必要とするプロセス・インスタンスが開始された場合、入力コンテナーの値は設定されません。たとえばアクティビティー・インプリメンテーション内から入力コンテナー値を照会すると、FMC_ERROR_MEMBER_NOT_SET が戻されます。

予約済みパラメーターには、NULL (0) ポインターまたは空ストリングを渡します。

createAndStartInstance によって入力コンテナーを渡すこともできます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- プロセスの許可

- プロセス管理の許可
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ProcessTemplate

ActiveX のシグニチャー

```
ProcessInstance* CreateAndStartInstance(
       BSTR                          name,
       boolean                       nameIsNull,
       BSTR                          reserved1,
       BSTR                          reserved2,
       boolean                       keepName,
       long *                        returnCode  )
ProcessInstance* CreateAndStartInstanceWithCnr(
       BSTR                          name,
       boolean                       nameIsNull,
       BSTR                          reserved1,
       BSTR                          reserved2,
       Container *                   input,
       boolean                       keepName,
       long *                        returnCode  )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjProcessTemplateCreateAndStartInstance(
       FmcjProcessTemplateHandle     hdlTemplate,
       char const *                  name,
       char const *                  reserved1,
       char const *                  reserved2,
       FmcjReadWriteContainerHandle  input,
       bool                          keepName,
       FmcjProcessInstanceHandle *   newInstance )
```

C++ 言語のシグニチャー

```
APIRET CreateAndStartInstance(
       string const *                 name,
       string const *                 reserved1,
       string const *                 reserved2,
       FmcjProcessInstance &          newInstance,
       bool                           keepName = false ) const;
APIRET CreateAndStartInstance(
       string const *                 name,
       string const *                 reserved1,
       string const *                 reserved2,
       FmcjReadWriteContainer const & input,
       FmcjProcessInstance &          newInstance,
       bool                           keepName = false ) const;
```

Java のシグニチャー

```
public abstract
ProcessInstance createAndStartInstance(
                String                 name,
                String                 reserved1,
                String                 reserved2,
                boolean                keepName ) throws FmcException
public abstract
ProcessInstance createAndStartInstance2(
                String                 name,
                String                 reserved1,
                String                 reserved2,
                ReadWriteContainer     input,
                boolean                keepName ) throws FmcException
```

XML メッセージ

```
<!-- ProcessTemplateCreateAndStart ================== -->
<!ELEMENT ProcessTemplateCreateAndStartInstance
          ( ProcTemplName,
            ProcInstName?,
            KeepName?,
            ProcInstInputData? ) >
<!ELEMENT ProcTemplName             (#PCDATA) >
<!ELEMENT ProgInstName              (#PCDATA) >
<!ELEMENT KeepName                  (#PCDATA) >
               <!--  Expected values: {true, false} -->
<!ELEMENT ProcInstInputData (%CONTAINER;) >
```

XML メッセージ (続き)

```
<!ELEMENT ProcessTemplateCreateAndStartInstanceResponse
          ( ProcessInstance
          | Exception ) >
<!ELEMENT ProcessInstance
          ( ProcInstID,
            ProcInstName,
            ProcInstParentName?,
            ProcInstTopLevelName,
            ProcInstDescription?,
            ProcInstState,
            LastStateChangeTime,
            LastModificationTime,
            ProcTemplID,
            ProcTemplName,
            Icon,
            Category? ) >
```

XML メッセージ (続き)

```
<!ELEMENT ProcInstID                (#PCDATA) >
<!ELEMENT ProcInstDescription       (#PCDATA) >
<!ELEMENT ProcInstName              (#PCDATA) >
<!ELEMENT ProcInstParentName        (#PCDATA) >
<!ELEMENT ProcInstTopLevelName      (#PCDATA) >
<!ELEMENT ProcInstState             (#PCDATA)  >
               <!-- Expected values: {Ready,Running,Finished,Terminated,
                                       Suspended, Terminating,
                                       Suspending,Deleted} -->
<!ELEMENT LastModificationTime      (#PCDATA) >
<!ELEMENT LastStateChangeTime       (#PCDATA) >
<!ELEMENT ProcTemplID               (#PCDATA) >
<!ELEMENT ProcTemplName             (#PCDATA) >
<!ELEMENT Icon                      (#PCDATA) >
<!ELEMENT Category                  (#PCDATA) >
<!ELEMENT Exception
          (Rc?, Parameters?, MessageText, Origin?) >
          <!-- Message text is optional, as it will be ignored
               in messages being sent *to* the Wf server. -->
<!ELEMENT Parameters
          (Parameter*) >
<!ELEMENT Parameter                 (#PCDATA) >
<!ELEMENT Rc                        (#PCDATA) >
<!ELEMENT MessageText               (#PCDATA) >
<!ELEMENT Origin                    (#PCDATA) >
```

XML メッセージ (続き)

```
<!--==========================================
     Sample Entity Container
     Any used data structure must be included here, for example,
     <!ENTITY %CONTAINER "(%_CONTAINER_INFO;,(CreditData|abcd|smart))">
   ==========================================-->
<!ENTITY %CONTAINER "(%_CONTAINER_INFO;,CreditData)">
```

パラメーター

hdlTemplate 入力。使用するプロセス・テンプレートオブジェクトのハンドル。

input 入力。プロセス・インスタンスの入力コンテナー。

keepName 入力。指定した名前だけをプロセス・インスタンスに使用できる場合には真。固有名が生成され得る場合には偽。

name 入力。作成して開始するプロセス・インスタンスの名前。

nameIsNull 入力。作成または開始するプロセス・インスタンスに名前を指定するかどうかを示します。

newInstance 入出力。新しく作成し開始するプロセス・インスタンス。

returnCode 入出力。このメソッドの呼び出しの結果。下記を参照してください。

reserved1/reserved2

入力。 0 (NULL) ポインターまたは空ストリングを渡します。

戻り値のデータ型

APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ProcessInstance*/ ProcessInstance

新たに作成および開始されたプロセス・インスタンスへのポインター、または新たに作成および開始されたプロセス・インスタンス。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)

パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・テンプレートが存在していないか、または無効になっています。

FMC_ERROR_INVALID_CONTAINER(509)
渡されたコンテナーが開始したプロセス・インスタンスに有効ではありません。スキーマまたはバージョンが間違っています。

FMC_ERROR_INVALID_NAME(134)
指定したプロセス・インスタンス名は構文規則に従っていません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_NOT_UNIQUE(121)
プロセス・インスタンスの名前が固有ではありません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

FMC_ERROR_XML_BACKOUT_COUNT_EXCEEDED(1106)
許可されている最大バックアウト・カウントを超えています。

FMC_ERROR_XML_DOCUMENT_FORMAT(1107)
MQMD フォーマット・フィールドの値が正しくありません。

FMC_ERROR_XML_DOCUMENT_INVALID(1100)
この文書は有効な XML 文書ではありません。

FMC_ERROR_XML_INVALID_ELEMENT(1110)
XML メッセージに無効な要素があります。

FMC_ERROR_XML_NO_MQSWF_DOCUMENT(1101)
この文書は有効な MQSeries Workflow XML 文書ではありません。

FMC_ERROR_XML_PARAMETER_INCORRECT(1108)
XML メッセージに無効なパラメーターがあります。

FMC_ERROR_XML_PARAMETER_SIGNATURE_CORRECT(1109)
XML メッセージに無効なパラメーターの組み合わせがあります。

FMC_ERROR_XML_WRONG_DATA_STRUCTURE(1103)
コンテナーの型が正しくありません。

FMC_ERROR_XML_DATA_MEMBER_NOT_FOUND(1104)
指定されたデータ・メンバーは、コンテナーの一部ではありません。

FMC_ERROR_XML_DATA_MEMBER_WRONG_TYPE(1105)
渡されたデータ・メンバー値の型が正しくありません。

XML の例 - MQSeries Workflow に送信される MQSeries Workflow XML 要求:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<WfMessage>
 <WfMessageHeader>
   <ResponseRequired>Yes</ResponseRequired>
   <UserContext>This data is sent back in the response</UserContext>
 </WfMessageHeader>
 <ProcessTemplateCreateAndStartInstance>
   <ProcTemplName>OnlineCreditRequest</ProcTemplName>
   <ProgInstName>Credit Request #658321<ProgInstName>
   <KeepName>true</KeepName>
   <ProcInstInputData>
      <CreditData>
        <Customer>
         <Name>User1</Name>
        </Customer>
        <Amount>1000</Amount>
        <Currency>CurrencyX</Currency>
```

```
      </CreditData>
   </ProcInstInputData>
 </ProcessTemplateCreateAndStartInstance>
</WfMessage>
```

XML example - MQSeries Workflow response sent from MQSeries Workflow to the reply queue:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<WfMessage>
 <WfMessageHeader>
   <ResponseRequired>No</ResponseRequired>
   <UserContext>This data is sent back in the response</UserContext>
 </WfMessageHeader>
<ProcessTemplateCreateAndStartInstanceResponse>
    <ProcessInstance>
      <ProcInstID>42424242EFEFEFEF</ProcInstID>
      <ProcInstName>Credit Request#658321</ProcInstName>
      <ProcInstTopLevelName>Credit Request#658321</ProcInstTopLevelName>
      <ProcInstDescription>Sample description</ProcInstDescription>
      <ProcInstState>Finished</ProcInstState>
      <LastStateChangeTime>1999-05-18 14:35:00</LastStateChgTime>
      <LastModificationTime>1999-05-19 23:40:00</LastModTime>
      <ProcTemplID>84848484FEFEFEFE</ProcTemplID>
      <ProcTemplName>OnlineCreditRequest</ProcTemplName>
      <Icon>fmcpcred</Icon>
      <Category>Finance</Category>
    </ProcessInstance>
 </ProcessTemplateCreateAndStartInstanceResponse>
</WfMessage>
```

CreateInstance()

この関数 / メソッドは、指定されたプロセス・テンプレートからプロセス・インスタンスを作成します (アクション呼び出し)。

keepName オプションに応じて、プロセス・インスタンス名を指定する必要があります。プロセス・インスタンス名を*現状のまま* にする場合、空文字列を指定することはできません。

プロセス・インスタンスの名前を指定する場合には、以下の規則が適用されます。

- 最大 63 文字 (バイト) を指定できます。
- 現行のロケールに応じて、以下のものを除くあらゆる印刷可能文字を使用できます。

  `* ? " ; : . $`
- ブランクも使用できますが、先行ブランクを使わない、末尾ブランクを使わない、複数のブランクを連続して使わない、という制限があります。

MQSeries Workflow によって固有名が生成され得る場合、次のことが当てはまります。

- プロセス・インスタンス名を指定しない場合、または空の名前を入力した場合は、省略時の名前 *ProcessTemplateName$Oid* でインスタンスが作成されます (*Oid* は、プロセス・インスタンス・オブジェクト識別子の印刷可能バージョン)。プロセス・インスタンス名は 63 バイト以下でなければならないため、プロセス・テンプレート名は短縮されることがあります。
- プロセス・インスタンス名が指定された場合、それが固有である限りその名前は保持されます。指定されたプロセス・インスタンス名が別のインスタンスですでに使用されている場合、名前 *name$Oid* でインスタンスが作成されます (*Oid* は、プロセス・インスタンス・オブジェクト識別子の印刷可能バージョン)。プロセス・インスタンス名は 63 バイト以下でなければならないため、名前は短縮されることがあります。

渡された名前パラメーターの値は変更されません。 FmcjProcessInstance::Name() は、作成されたプロセス・インスタンスの実際の名前を戻します。新しく作成されたプロセス・インスタンスには、1 次属性値だけが入っています。

予約済みパラメーターには、NULL (0) ポインターまたは空ストリングを渡します。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- プロセスの許可
- プロセス管理の許可
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx

JAVA com.ibm.workflow.api.ProcessTemplate

ActiveX のシグニチャー

```
ProcessInstance*  CreateInstance(
       BSTR                          name,
       boolean                       nameIsNull,
       BSTR                          reserved1,
       BSTR                          reserved2,
       boolean                       keepName,
       long *                        returnCode  )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjProcessTemplateCreateInstance(
       FmcjProcessTemplateHandle   hdlTemplate,
       char const *                name,
       char const *                reserved1,
       char const *                reserved2,
       bool                        keepName,
       FmcjProcessInstanceHandle * newInstance )
```

C++ 言語のシグニチャー

```
APIRET CreateInstance(
       string const *        name,
       string const *        reserved1,
       string const *        reserved2,
       FmcjProcessInstance & newInstance,
       bool                  keepName = false ) const
```

Java のシグニチャー

```
public abstract
ProcessInstance createInstance(
               String        name,
               String        reserved1,
               String        reserved2,
               boolean       keepName ) throws FmcException
```

パラメーター

hdlTemplate 入力。使用するプロセス・テンプレートオブジェクトのハンドル。

keepName 入力。指定した名前だけをプロセス・インスタンスに使用できる場合には真。固有名が生成され得る場合には偽。

name 入力。作成するプロセス・インスタンスの名前。

nameIsNull 入力。作成するプロセス・インスタンスに名前を指定するかどうかを示します。

newInstance 入出力。新しく作成されるプロセス・インスタンス。

reserved1/reserved2
入力。 0 (NULL) ポインターまたは空ストリングを渡します。

returnCode 入出力。このメソッドの呼び出しの結果。下記を参照してください。

戻り値のデータ型

APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ProcessInstance*/ ProcessInstance
新たに作成されたプロセス・インスタンスへのポインター、または新たに作成されたプロセス・インスタンス。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・テンプレートが存在していないか、または無効になっています。

FMC_ERROR_INVALID_NAME(134)
指定したプロセス・インスタンス名は構文規則に従っていません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_NOT_UNIQUE(121)
プロセス・インスタンスの名前が固有ではありません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Delete()

この関数 / メソッドは、指定したプロセス・テンプレートを実行サーバーから削除します (アクション呼び出し)

プロセス・テンプレートはバージョン管理されているため、現在有効なプロセス・テンプレートを削除するか、旧バージョンのプロセス・テンプレートを削除するか、それとも将来のバージョンのプロセス・テンプレートを削除するかを指定することができます。それらのオプションをすべて指定すると、すべてのバージョンのプロセス・テンプレートが削除されます。削除は、どんな場合も現在の既存のプロセス・テンプレートだけに適用されます。

プロセス・テンプレートの一時的な表示には影響がありません。 C および C++ では、一時オブジェクトが必要なくなった時点で破棄または割り振り解除する必要があります。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセス・モデル定義許可
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplate

ActiveX のシグニチャー

```
long Delete( boolean   pastVersions,
             boolean   currentVersion,
             boolean   futureVersions )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjProcessTemplateDelete( FmcjProcessTemplateHandle hdlTemplate,
                           bool                      pastVersions,
                           bool                      currentVersion,
                           bool                      futureVersions )
```

C++ 言語のシグニチャー

```
APIRET Delete( bool pastVersions  = true,
               bool currentVersion= true,
               bool futureVersions= true )
```

Java のシグニチャー

```
public abstract
void   delete() throws FmcException
public abstract
void   delete2( boolean pastVersions,
                boolean currentVersion,
                boolean futureVersions ) throws FmcException
```

パラメーター

currentVersion
入力。現行バージョンのプロセス・テンプレートを削除するかどうかの指定。

futureVersions
入力。将来のバージョンのプロセス・テンプレートを削除するかどうかの指定。

hdlTemplate 入力。削除するプロセス・テンプレートのハンドル。

pastVersions 入力。過去のバージョンのプロセス・テンプレートを削除するかどうかの指定。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・テンプレートまたはその指定のバージョンはもはや存在しません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

ExecuteProcessInstance()

この関数 / メソッドは、指定されたプロセス・テンプレートからプロセス・インスタンスを作成して、そのプロセス・インスタンスを実行します (アクション呼び出し)。

アクティビティー・インプリメンテーションを実行するには、プログラム実行エージェントが開始済みでなければならないことに注意してください。

この関数 / メソッドは同期で、また非同期で呼び出すことが可能です。非同期で呼び出す場合、プロセス・インスタンスの実行時間は、アプリケーション待ち時間内で完了する短いものでなければなりません。非同期で呼び出す場合には、ユーザー・コンテキストを指定して、後で受け取る応答に相関させることができます。さらに、相関 ID を受け取って、それを特定の応答の待機に使用することができます。相関 ID を保持するバッファーを指定する場合、それは初めに FMCJ_NO_CORRELID を指していなければなりません、つまり含まれているのがゼロ (0x00) だけでなければなりません。

keepName オプションに応じて、プロセス・インスタンス名を指定する必要があります。プロセス・インスタンス名を *現状のまま* にする場合、空文字列を指定することはできません。

プロセス・インスタンスの名前を指定する場合には、以下の規則が適用されます。

- 最大 63 文字 (バイト) を指定できます。
- 現行のロケールに応じて、以下のものを除くあらゆる印刷可能文字を使用できます。

  ```
  * ? " ; : . $
  ```

- ブランクも使用できますが、先行ブランクを使わない、末尾ブランクを使わない、複数のブランクを連続して使わない、という制限があります。

MQSeries Workflow によって固有名が生成され得る場合、次のことが当てはまります。

- プロセス・インスタンス名を指定しない場合、または空の名前を入力した場合は、省略時の名前 *ProcessTemplateName$Oid* でインスタンスが作成されます (*Oid* は、プロセス・インスタンス・オブジェクト識別子の印刷可能バージョン)。プロセス・インスタンス名は 63 バイト以下でなければならないため、プロセス・テンプレート名は短縮されることがあります。
- プロセス・インスタンス名が指定された場合、それが固有である限りその名前は保持されます。指定されたプロセス・インスタンス名が別のインスタンスですでに使用されている場合、名前 *name$Oid* でインスタンスが作成されます (*Oid* は、プロセス・インスタンス・オブジェクト識別子の印刷可能バージョン)。プロセス・インスタンス名は 63 バイト以下でなければならないため、名前は短縮されることがあります。

渡された名前パラメーターの値は変更されません。 FmcjProcessInstance::Name() は、作成されたプロセス・インスタンスの実際の名前を戻します。新しく作成されたプロセス・インスタンスには、1 次と 2 次のすべての属性が含まれています。

作成および開始するプロセス・インスタンスに初期値を渡す場合、入力コンテナーを指定することができます。 FmcjProcessTemplate::InContainer() も参照してください。入力コンテナーの指定なしに、入力を必要とするプロセス・インスタンスが開始された場合、入力コンテナーの値は設定されません。たとえばアクティビティー・インプリメンテーション内から入力コンテナー値を照会すると、FMC_ERROR_MEMBER_NOT_SET が戻されます。

予約済みパラメーターには、NULL (0) ポインターまたは空ストリングを渡します。

完了時には、実行されたプロセス・インスタンスとその出力コンテナーが戻されます。プロセス・インスタンスには 1 次属性の値だけが入れられます。プロ

セス・インスタンスが中止された場合、コンテナーは戻されません。つまり、0 ポインターまたは空のコンテナーが戻されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセスの許可
- プロセス管理の許可
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	サポートなし

ActiveX のシグニチャー

```
ProcessInstance* ExecuteProcessInstance(
       Container *                  output,
       BSTR                         name,
       boolean                      nameIsNull,
       BSTR                         reserved1,
       BSTR                         reserved2,
       boolean                      keepName,
       long *                       returnCode  )
ProcessInstance* ExecuteProcessInstanceWithCnr(
       Container *                  input,
       Container *                  output,
       BSTR                         name,
       boolean                      nameIsNull,
       BSTR                         reserved1,
       BSTR                         reserved2,
       boolean                      keepName,
       long *                       returnCode  )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjProcessTemplateExecuteProcessInstance(
       FmcjProcessTemplateHandle     hdlTemplate,
       char const *                  name,
       char const *                  reserved1,
       char const *                  reserved2,
       FmcjReadWriteContainerHandle  input,
       bool                          keepName,
       FmcjProcessInstanceHandle *   newInstance,
       FmcjReadOnlyContainerHandle * output     )
APIRET FMC_APIENTRY FmcjProcessTemplateExecuteProcessInstanceAsync(
       FmcjProcessTemplateHandle     hdlTemplate,
       char const *                  name,
       char const *                  reserved1,
       char const *                  reserved2,
       FmcjReadWriteContainerHandle  input,
       bool                          keepName,
       FmcjCorrelID *                correlID,
       char const *                  userContext )
```

C++ 言語のシグニチャー

```
APIRET ExecuteProcessInstance(
       FmcjProcessInstance &           newInstance,
       FmcjReadOnlyContainer &         output,
       string const *                  name = 0,
       string const *                  reserved1 = 0,
       string const *                  reserved2 = 0,
       bool                            keepName = false ) const
APIRET ExecuteProcessInstance(
       FmcjReadWriteContainer const &  input,
       FmcjProcessInstance &           newInstance,
       FmcjReadOnlyContainer &         output,
       string const *                  name = 0,
       string const *                  reserved1 = 0,
       string const *                  reserved2 = 0,
       bool                            keepName = false ) const
APIRET ExecuteProcessInstanceAsync(
       string const *                  name = 0,
       string const *                  reserved1 = 0,
       string const *                  reserved2 = 0,
       bool                            keepName = false,
       FmcjCorrelID  *                 correlID = 0,
       string const *                  userContext = 0 )
APIRET ExecuteProcessInstanceAsync(
       FmcjReadWriteContainer const &  input,
        string const *                 name = 0,
        string const *                 reserved1 = 0,
        string const *                 reserved2 = 0,
        bool                           keepName = false,
        FmcjCorrelID  *                correlID = 0,
        string const *                 userContext = 0 )
```

XML メッセージ

```
<!-- ProcessTemplateExecute ========================== -->
<!ELEMENT ProcessTemplateExecute
          ( ProcTemplName,
            ProcInstName?,
            KeepName?,
            ProcInstInputData? ) >
<!ELEMENT ProcTemplName             (#PCDATA) >
<!ELEMENT ProgramName               (#PCDATA) >
<!ELEMENT KeepName                  (#PCDATA) >
                            <!--  Expected values: {true, false} -->
<!ELEMENT ProcInstInputData (%CONTAINER;) >
<!ELEMENT ProcessTemplateExecuteResponse
          ( ( ProcessInstance,
              ProcInstOutputData? )
          | Exception ) >
<!ELEMENT ProcessInstance
          ( ProcInstID,
            ProcInstName,
            ProcInstParentName?,
            ProcInstTopLevelName,
            ProcInstDescription?,
            ProcInstState,
            LastStateChangeTime,
            LastModificationTime,
            ProcTemplID,
            ProcTemplName,
            Icon,
            Category? ) >
```

XML メッセージ (続き)

```
<!ELEMENT ProcInstID                (#PCDATA) >
<!ELEMENT ProcInstDescription       (#PCDATA) >
<!ELEMENT ProcInstName              (#PCDATA) >
<!ELEMENT ProcInstParentName        (#PCDATA) >
<!ELEMENT ProcInstTopLevelName      (#PCDATA) >
<!ELEMENT ProcInstState             (#PCDATA)  >
     <!-- Expected values: { Ready,      Running,
                             Finished,   Terminated,
                             Suspended,  Terminating,
                             Suspending, Deleted    } -->
<!ELEMENT LastModificationTime      (#PCDATA) >
<!ELEMENT LastStateChangeTime       (#PCDATA) >
<!ELEMENT ProcTemplID               (#PCDATA) >
<!ELEMENT ProcTemplName             (#PCDATA) >
<!ELEMENT Icon                      (#PCDATA) >
<!ELEMENT Category                  (#PCDATA) >
<!ELEMENT ProcInstOutputData        (%CONTAINER;) >
<!ELEMENT Exception
          (Rc?, Parameters?, MessageText, Origin?) >
          <!-- Message text is optional, as it will be ignored
               in messages being sent *to* the Wf server. -->
<!ELEMENT Parameters
          (Parameter*) >
<!ELEMENT Parameter                 (#PCDATA) >
<!ELEMENT Rc                        (#PCDATA) >
<!ELEMENT MessageText               (#PCDATA) >
<!ELEMENT Origin                    (#PCDATA) >
```

XML メッセージ (続き)

```
<!--=========================================
     Sample Entity Container
     Any used data structure must be included here, for example,
     <!ENTITY %CONTAINER "(%_CONTAINER_INFO;,(CreditData|abcd|smart))">
   ==========================================-->
<!ENTITY %CONTAINER "(%_CONTAINER_INFO;,CreditData)">
```

パラメーター

correlID 入出力。指定した場合、この要求を後からの応答に相関させるための相関 ID が含まれます。

hdlTemplate 入力。使用するプロセス・テンプレートオブジェクトのハンドル。

input 入力。プロセス・インスタンスの入力コンテナー。

keepName 入力。指定した名前だけをプロセス・インスタンスに使用できる場合には真。固有名が生成され得る場合には偽。

name 入力。実行するプロセス・インスタンスの名前。

nameIsNull 入力。実行するプロセス・インスタンスに名前を指定するかどうかを示します。

newInstance 入出力。実行されたプロセス・インスタンス。

output 出力。プロセス・インスタンスの出力コンテナー。

returnCode 入出力。このメソッドの呼び出しの結果。下記を参照してください。

reserved1/reserved2
入力。 0 (NULL) ポインターまたは空ストリングを渡します。

userContext 入力。相関に使用できるユーザー定義のコンテキスト。

戻り値のデータ型

APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ProcessInstance*
新たに作成および実行されたプロセス・インスタンスへのポインター。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・テンプレートが存在していないか、または無効になっています。

FMC_ERROR_INVALID_CONTAINER(509)
指定したコンテナーがプロセス・インスタンス実行に有効ではありません。スキーマまたはバージョンが間違っています。

FMC_ERROR_INVALID_CORRELATION_ID(506)
指定された相関 ID が FMCJ_NO_CORRELID を指していません。

FMC_ERROR_INVALID_NAME(134)
指定したプロセス・インスタンス名は構文規則に従っていません。

FMC_ERROR_INVALID_USER_CONTEXT(819)
指定したユーザー・コンテキストが 254 文字を超えています。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_NOT_UNIQUE(121)
プロセス・インスタンスの名前が固有ではありません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

FMC_ERROR_XML_BACKOUT_COUNT_EXCEEDED(1106)
許可されている最大バックアウト・カウントを超えています。

FMC_ERROR_XML_DOCUMENT_FORMAT(1107)
MQMD フォーマット・フィールドの値が正しくありません。

FMC_ERROR_XML_DOCUMENT_INVALID(1100)
この文書は有効な XML 文書ではありません。

FMC_ERROR_XML_INVALID_ELEMENT(1110)
XML メッセージに無効な要素があります。

FMC_ERROR_XML_NO_MQSWF_DOCUMENT(1101)
この文書は有効な MQSeries Workflow XML 文書ではありません。

FMC_ERROR_XML_PARAMETER_INCORRECT(1108)
XML メッセージに無効なパラメーターがあります。

FMC_ERROR_XML_PARAMETER_SIGNATURE_CORRECT(1109)
XML メッセージに無効なパラメーターの組み合わせがあります。

FMC_ERROR_XML_WRONG_DATA_STRUCTURE(1103)
コンテナーの型が正しくありません。

FMC_ERROR_XML_DATA_MEMBER_NOT_FOUND(1104)
指定されたデータ・メンバーは、コンテナーの一部ではありません。

FMC_ERROR_XML_DATA_MEMBER_WRONG_TYPE(1105)
渡されたデータ・メンバー値の型が正しくありません。

XML の例 - MQSeries Workflow に送信される MQSeries Workflow XML 要求:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<WfMessage>
 <WfMessageHeader>
   <ResponseRequired>Yes</ResponseRequired>
   <UserContext>This data is sent back in the response</UserContext>
 </WfMessageHeader>
<ProcessTemplateExecute>
    <ProcTemplName>OnlineCreditRequest</ProcTemplName>
      <ProcInstName>Credit Request #658321</ProcInstName>
    <KeepName>true</KeepName>
    <ProcInstInputData>
      </CreditData>
        <Customer>
         <Name>User1</Name>
        </Customer>
        <Amount>1000</Amount>
        <Currency>CurrencyX</Currency>
      </CreditData>
    </ProcInstInputData>
  </ProcessTemplateExecute>
</WfMessage>
```

XML の例 - MQSeries Workflow から応答キューに送信される MQSeries Workflow XML 応答:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<WfMessage>
 <WfMessageHeader>
   <ResponseRequired>No</ResponseRequired>
   <UserContext>This data is sent back in the response</UserContext>
```

```
 </WfMessageHeader>
<ProcessTemplateExecuteResponse>
    <ProcessInstance>
      <ProcInstID>42424242EFEFEFEF</ProcInstID>
      <ProcInstName>Credit Request #658321</ProcInstName>
      <ProcInstTopLevelName>Credit Request #658321</ProcInstTopLevelName>
      <ProcInstDescription>Sample description</ProcInstDescription>
      <ProcInstState>Finished</ProcInstState>
      <LastStateChangeTime>1999-05-18 14:35:00</LastStateChgTime>
      <LastModificationTime>1999-05-19 23:40:00</LastModTime>
      <ProcTemplID>84848484FEFEFEFE</ProcTemplID>
    <ProcTemplName>OnlineCreditRequest</ProcTemplName>
      <Icon>fmcpcred</Icon>
      <Category>Finance</Category>
    </ProcessInstance>
    <ProcInstOutputData>
      <CreditData>
        <Customer>
         <Name>User1</Name>
        </Customer>
        <Amount>1000</Amount>
        <Currency>CurrencyX</Currency>
      </CreditData>
    </ProcInstOutputData>
  </ProcessTemplateExecuteResponse>
</WfMessageHeader>
```

InitialInContainer()

この関数 / メソッドは、プロセス・テンプレートに関連する入力コンテナーを、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセスの許可
- プロセス管理の許可
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ProcessTemplate

ActiveX のシグニチャー

```
long InitialInContainer( Container * input )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjProcessTemplateInitialInContainer(
       FmcjProcessTemplateHandle      hdlTemplate,
       FmcjReadWriteContainerHandle * input )
```

C++ 言語のシグニチャー

```
APIRET InitialInContainer( FmcjReadWriteContainer & input )
```

Java のシグニチャー

```
public abstract
ReadWriteContainer initialInContainer() throws FmcException
```

パラメーター

hdlTemplate 入力。入力コンテナーを検索するプロセス・テンプレート・オブジェクトのハンドル。

input 入出力。設定するプロセス・テンプレートの入力コンテナー・ハンドルのアドレス (C)、またはその入力コンテナーのアドレス (C++)。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

ReadWriteContainer
プロセス・テンプレートの入力コンテナー。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・テンプレートが存在していないか、または無効になっています。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

PersistentObject()

この関数 / メソッドは、MQSeries Workflow 実行サーバーから渡されたオブジェクト識別子で識別されるプロセス・テンプレートを取り出します (アクション呼び出し)。

プロセス・テンプレートの検索元の MQSeries Workflow 実行サーバーは、実行サービス・オブジェクトによって識別されます。その場合、一時オブジェクトは、プロセス・テンプレートのすべての情報 (1 次および 2 次) を使って作成また更新されます。

C++ の場合、初期化するプロセス・テンプレート・オブジェクトが空でなければ、そのオブジェクトがまず破棄されてから、新しいオブジェクトが割り当てられます。 C の場合は、オブジェクトの所有権についてはアプリケーションの側が完全に制御することになります。つまり、プロセス・テンプレートのハンドルがすでに何らかのオブジェクトを指しているかどうかのチェックは実行されません。 Java の場合、プロセス・テンプレートは新たに作成されます。実行サービスがファクトリーとして機能します。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセスの許可
- プロセス管理の許可
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long PersistentObject( ExecutionService * service, BSTR oid )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjProcessTemplatePersistentObject(
       FmcjExecutionServiceHandle  service,
       char const *                oid,
       FmcjProcessTemplateHandle * hdlTemplate )
```

C++ 言語のシグニチャー

```
APIRET PersistentObject( FmcjExecutionService const & service,
                         string const &               oid )
```

Java のシグニチャー

```
public abstract
  ProcessTemplate ExecutionService.processTemplate( String oid )
throws FmcException
```

パラメーター

hdlTemplate 入出力。設定するプロセス・テンプレートオブジェクトのハンドルのアドレス

oid 入力。検索するプロセス・テンプレートのオブジェクト識別子。

service 入力。実行サーバーとのセッションを表すサービス・オブジェクト。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

ProcessTemplate
取り出されたプロセス・テンプレート。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・テンプレートが存在していないか、または無効になっています。

FMC_ERROR_INVALID_OID(805)
指定した oid は無効です。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

ProgramTemplate()

この関数 / メソッドは、MQSeries Workflow 実行サーバーから渡されたプログラム・テンプレート識別子で識別されるプログラム・テンプレートを取り出します (アクション呼び出し)。

プログラム・テンプレートは、それに関連する入力コンテナーと出力コンテナーに関するデータ、および指定したすべてのプラットフォームや他のさまざまなプロパティーのインプリメンテーション・データで構成されます。定義機能中にそのプログラムについて「*structures from activity*」が指定された場合は、入力コンテナーと出力コンテナーに関する情報が得られません。実行時には任意のコンテナーを渡すことができます。

プログラム・テンプレートにコンテナーが提供されると、それらは初期コンテナーとなります。この場合、データ・メンバーの省略時値は設定されません。また、事前定義データ・メンバーも設定されません。

システムから継承された可能性のある値が戻されるため、この関数 / メソッド呼び出しの結果は要求が実行されたシステムによって異なります。

プログラム・テンプレートのバージョンは、対応するプロセス・テンプレートのコンテキスト内に示されたものです。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセスの許可
- プロセス管理の許可
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplate

ActiveX のシグニチャー

```
ProgramTemplate * ProgramTemplate( BSTR programName, long * returnCode )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjProcessTemplateProgramTemplate(
       FmcjProcessTemplateHandle   hdlTemplate,
       char const *                programName,
       FmcjProgramTemplateHandle * program )
```

C++ 言語のシグニチャー

```
APIRET ProgramTemplate( string const &         programName,
                        FmcjProgramTemplate &  program    ) const
```

Java のシグニチャー

```
public abstract
ProgramTemplate  programTemplate( String programName )
throws FmcException
```

パラメーター

hdlTemplate 入力。プログラム・テンプレートを検索するプロセス・テンプレートのハンドル。

program 入出力。検索するプログラム・テンプレート。

programName
入力。検索するプログラム・テンプレートの名前。

returnCode 入出力。このメソッドの呼び出しの結果。下記を参照してください。

戻り値のデータ型

APIRET/long* この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

ProgramTemplate/ProgramTemplate*
検索するプログラム・テンプレート / 検索するプログラム・テンプレートへのポインター。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_BACK_LEVEL_OBJECT
要求は、MQSeries Workflow 3.2.1 のインストール後に変換されたプロセス・テンプレートでのみ実行できます。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・テンプレートが存在していないか、または無効になっています。または、プロセス・テンプレート内にプログラム・テンプレートがありません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Refresh()

この関数 / メソッドは、MQSeries Workflow 実行サーバー (アクション呼び出し)。

プロセス・テンプレートに関するすべての 1 次情報および 2 次情報が取り出されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- プロセスの許可
- プロセス管理の許可
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProcessTemplate

ActiveX のシグニチャー

```
long Refresh()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjProcessTemplateRefresh( FmcjProcessTemplateHandle hdlTemplate )
```

C++ 言語のシグニチャー

```
APIRET Refresh()
```

Java のシグニチャー

```
public abstract
void   refresh() throws FmcException
```

パラメーター

hdlTemplate 入力。リフレッシュするプロセス・テンプレートオブジェクトのハンドル。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・テンプレートが存在していないか、または無効になっています。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

第51章 プロセス・テンプレート・リストのアクション

プロセス・テンプレート・リストはプロセス・テンプレートの集まりを表します。このリストによってアクセスできるすべてのプロセス・テンプレートは特性が同じです。その特性はフィルターによって指定されます。また、ソート基準を適用することができ、その後、実行サーバーからクライアントへと転送されるプロセス・テンプレートの数を制限するしきい値を適用できます。

プロセス・テンプレート・リストの定義は永続的に保管されます。

プロセス・テンプレート・リストは、その名前、タイプ、所有者によって一意に識別されます。これは、一般的なアクセスを目的として定義することができます。その場合にはパブリック *(public)* タイプとなります。あるいは、特定の一部のユーザーが使用するようにも定義できます。その場合はプライベート *(private)* タイプとなります。

定義可能なその他のリストには、プロセス・インスタンス・リストとワークリストがあります。 FmcjPersistentList または PersistentList は、すべてのリストの共通プロパティーを表します。

C++ 言語において、FmcjProcessTemplateList は FmcjPersistentList クラスのサブクラスであり、すべてのプロパティーとメソッドを継承します。 Java 言語において ProcessTemplateList は PersistentList クラスのサブクラスであり、すべてのプロパティーとメソッドを継承します。同じように C 言語では、関数の共通のインプリメンテーションを FmcjPersistentList から取得します。つまり、共通関数には接頭部 FmcjPersistentList が付いており、定義においては先頭に接頭部 FmcjProcessTemplateList が付けられます。 ActiveX では継承がサポートされていないため、すべての関数は ProcessTemplateList で明示的に定義されます。しかし、それらのメソッドは PersistentList アクションとして記述される点に注意してください。

以下の部分では、プロセス・テンプレート・リストに適用できるアクションについて説明します。関数 / メソッドの完全なリストについては、332ページの『プロセス・テンプレート・リスト』を参照してください。

QueryProcessTemplates()

この関数 / メソッドは、指定されたプロセス・テンプレート・リストによって特徴付けられるすべてのプロセス・テンプレートの 1 次情報を、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

当てはまるプロセス・テンプレートの集まりの中で、ユーザーに許可が与えられているものだけが検索されます。プロセス・テンプレートが以下のいずれかの条件を満たしている場合、ユーザーはそのプロセス・テンプレートの許可を与えられます。

- どのカテゴリーにも属していない。
- あるカテゴリーに属しており、ユーザーにはそのカテゴリーに関するグローバル・プロセス許可、グローバル・プロセス管理許可、選択プロセス許可、選択プロセス管理許可のいずれかを与えられている。

各プロセス・テンプレートごとに取り出される 1 次情報は以下のとおりです。

- Category
- CreationTime
- Description
- Icon
- InContainerNeeded
- LastModificationTime
- Name
- ValidFromTime

C および C++ では、検索したプロセス・テンプレートはすべて、提供されるプロセス・テンプレート・ベクトルに追加されます。現在プロセス・テンプレート・リストに含まれているプロセス・テンプレートだけを読みたい場合は、まずベクトルをクリアした後でこの関数 / メソッドを呼び出さなければなりません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ProcessTemplateList

ActiveX のシグニチャー

```
long QueryProcessTemplates()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjProcessTemplateListQueryProcessTemplates(
       FmcjProcessTemplateListHandle     hdlList,
       FmcjProcessTemplateVectorHandle * templates )
```

C++ 言語のシグニチャー

```
APIRET QueryProcessTemplates(
       vector<FmcjProcessTemplate> & templates ) const;
```

Java のシグニチャー

```
public abstract
ProcessTemplate[] queryProcessTemplates() throws FmcException
```

パラメーター

hdlList 入力。照会するプロセス・テンプレート・リストのハンドル。

templates 入出力。適格プロセス・テンプレートのベクトル。

戻り値のデータ型

long/ APIRET この関数 / メソッドの呼び出し結果 (下の戻りコードを参照)。

ProcessTemplate[]
適格プロセス・テンプレート。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
プロセス・テンプレート・リストはもう存在していません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- C 言語の例については、833ページの『ワークリストの照会 (C 言語)』を参照してください。
- C++ の例については、835ページの『ワークリストの照会 (C++)』を参照してください。

第52章 プログラム・テンプレートのアクション

FmcjProgramTemplate オブジェクトまたは ProgramTemplate オブジェクトは、プロセス・テンプレート内のプログラムの定義を表します。

プログラム・テンプレートは、その名前、およびそれが含まれているプロセス・テンプレートによって一意に識別されます。つまり、それを含んでいるプロセス・テンプレートを通してバージョンが決定されます。

以下の部分では、プログラム・テンプレートに適用できるアクションについて説明します。関数 / メソッドの完全なリストについては、336ページの『プログラム・テンプレート』を参照してください。

Execute()

この関数 / メソッドは、指定したプログラム・テンプレートを、ユーザーがログオンしているシステムのプログラム実行サーバー (PES) 上で実行することを要求します。

この関数 / メソッドは同期で、また非同期で呼び出すことが可能です。非同期で呼び出す場合、プログラムの実行時間は、アプリケーション待ち時間内で完了する短いものでなければなりません。非同期で呼び出す場合には、ユーザー・コンテキストを指定して、後で受け取る応答に相関させることができます。さらに、相関 ID を受け取って、それを特定の応答の待機に使用することができます。相関 ID を保持するバッファーを指定する場合、それは初めに FMCJ_NO_CORRELID を指していなければなりません、つまり含まれているのがゼロ (0x00) だけでなければなりません。

プログラム・テンプレートの *input container access* の定義によっては、実行のために入力コンテナーを指定する必要があります。 *output container access* の定義によっては、プログラム実行によって戻される値を保持するために、出力コンテナーを指定することができます。プログラムが出力コンテナーにアクセスしたもののコンテナーがない場合は、プログラム用に定義された出力コンテナーが使用されます。 *structures from activity* が定義されている場合は、どんなコンテナーでも処理できることをプログラムが宣言しているため、どんなタイプのコンテナーを渡すこともできます。 *structures from activity* が定義されていない場合、渡されるすべてのコンテナーは、プログラム設定で定義されているタイプに適合するものでなければなりません。

FmcjProcessTemplate::ProgramTemplate() によって戻される初期コンテナーには省略時値がありません。プログラムに初期値を渡す場合、この関数 / メソッドを呼び出す前に、入力コンテナーまたは出力コンテナーで初期値を設定することができます。

出力コンテナーがあるなら、完了時にその出力コンテナーが戻されます。出力コンテナーの _RC データ・メンバーは、プログラム戻りコードを表します。

優先順位の指定は OS/390 ワークロード管理に影響します。優先順位の値は 0 ～ 999 の間でなければなりません。

注:

1. Passthrough() は、PES 上で実行されるプログラムから呼び出すことはできません。
2. 出力コンテナーは入出力パラメーターです。これは Java の場合、出力コンテナーが入力パラメーターとして渡されること、また実行メソッドの戻り値として戻されることを意味します。入力パラメーターは変更されません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

ログオンしていること

必要な接続

MQSeries Workflow プログラム実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.ProgramTemplate

ActiveX のシグニチャー

```
long Execute()
long ExecuteWithOptions( Container *  input,
                         Container *  output
                         long         priority )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjProgramTemplateExecute(
       FmcjProcessTemplateHandle     hdlTemplate,
       FmcjReadWriteContainerHandle  input,
       FmcjReadWriteContainerHandle  output )
APIRET FMC_APIENTRY FmcjProgramTemplateExecuteWithOptions(
       FmcjProcessTemplateHandle     hdlTemplate,
       unsigned long                 priority,
       FmcjReadWriteContainerHandle  input,
       FmcjReadWriteContainerHandle  output )
APIRET FMC_APIENTRY FmcjProgramTemplateExecuteAsync(
       FmcjProcessTemplateHandle     hdlTemplate,
       FmcjReadWriteContainerHandle  input,
       FmcjReadWriteContainerHandle  output,
       FmcjCorrelID *                correlID,
       char const *                  userContext )
APIRET FMC_APIENTRY FmcjProgramTemplateExecuteWithOptionsAsync(
       FmcjProcessTemplateHandle     hdlTemplate,
       unsigned long                 priority,
       FmcjReadWriteContainerHandle  input,
       FmcjReadWriteContainerHandle  output,
       FmcjCorrelID *                correlID,
       char const *                  userContext )
```

C++ 言語のシグニチャー

```
APIRET Execute(      FmcjReadWriteContainer const * input    = 0,
                     FmcjReadWriteContainer *       output   = 0,
                     unsigned long                  priority = 0 ) const
APIRET ExecuteAsync( FmcjReadWriteContainer const * input       = 0,
                     FmcjReadWriteContainer const * output      = 0,
                     FmcjCorrelID  *                correlID    = 0,
                     string const *                 userContext = 0,
                     unsigned long                  priority    = 0) const
```

Java のシグニチャー

```
public abstract
ReadWriteContainer execute() throws FmcException
public abstract
ReadWriteContainer execute2( ReadWriteContainer input,
                             ReadWriteContainer output,
                             long                priority )
throws FmcException
```

パラメーター

correlID 入出力。指定した場合、この要求を後からの応答に相関させるための相関 ID が含まれます。

hdlTemplate 入力。実行するプログラム・テンプレート・オブジェクトのハンドル。

input 入力。プログラムの入力コンテナー。

output 入出力。プログラムの出力コンテナー。

priority 入力。実行するプログラムの優先順位。

userContext 入力。相関に使用できるユーザー定義のコンテキスト。

戻り値のデータ型

long/APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ReadWriteContainer
プログラムの出力コンテナー。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_EXIT_ERROR(32204)
プログラム実行サーバー出口がエラーを報告しました。

FMC_ERROR_IMPLEMENTATION_SUPPORT_MISMATCH(32015)
PES が実行されているオペレーティング・システム・プラットフォーム用のプログラム定義が見つかりません。

FMC_ERROR_INVALID_CONTAINER(509)
コンテナーのタイプまたはバージョンが正しくありません。または必要なコンテナーが渡されていません。

FMC_ERROR_INVALID_CORRELATION_ID(506)
指定された相関 ID が FMCJ_NO_CORRELID を指していません。

FMC_ERROR_INVALID_SPECIFICATION(816)
指定する優先順位は 0<=priority<=999 の範囲内でなければなりません。

FMC_ERROR_INVALID_USER_CONTEXT(819)
指定したユーザー・コンテキストが 254 文字を超えています。

FMC_ERROR_LOCAL_USER_REQUIRED(32203)
プログラムはローカル・ユーザーによって実行されなければなりません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_SUPPORT_MODE_MISMATCH(32014)
プログラムの実行モードと PES の実行モードが一致しません。

FMC_ERROR_UNEXPECTED_CONTAINER(510)
コンテナーが渡されましたが、プログラムではコンテナーを予期していません。

FMC_ERROR_USER_NOT_AUTHORIZED(32202)
ユーザーにはプログラムを実行する権限がありません。

FMC_ERROR_USER_SUPPORT_MISMATCH(32013)
プログラムの実行ユーザーと PES の実行ユーザーが一致しません。

FMC_ERROR_COMMUNICATION(13)
指定されたプログラム実行サーバーに到達できません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

FMC_ERROR_XML_BACKOUT_COUNT_EXCEEDED(1106)
許可されている最大バックアウト・カウントを超えています。

FMC_ERROR_XML_DOCUMENT_FORMAT(1107)
MQMD フォーマット・フィールドの値が正しくありません。

FMC_ERROR_XML_DOCUMENT_INVALID(1100)
この文書は有効な XML 文書ではありません。

FMC_ERROR_XML_INVALID_ELEMENT(1110)
XML メッセージに無効な要素があります。

FMC_ERROR_XML_NO_MQSWF_DOCUMENT(1101)
この文書は有効な MQSeries Workflow XML 文書ではありません。

FMC_ERROR_XML_PARAMETER_INCORRECT(1108)
XML メッセージに無効なパラメーターがあります。

FMC_ERROR_XML_PARAMETER_SIGNATURE_CORRECT(1109)
XML メッセージに無効なパラメーターの組み合わせがあります。

FMC_ERROR_XML_WRONG_DATA_STRUCTURE(1103)
コンテナーの型が正しくありません。

FMC_ERROR_XML_DATA_MEMBER_NOT_FOUND(1104)
指定されたデータ・メンバーは、コンテナーの一部ではありません。

FMC_ERROR_XML_DATA_MEMBER_WRONG_TYPE(1105)
渡されたデータ・メンバー値の型が正しくありません。

第53章 サービスのアクション

FmcjService または Service オブジェクトは、MQSeries Workflow サービス・オブジェクトに共通の性質を表します。

C++ 言語の場合、FmcjService は FmcjExecutionService クラスのスーパークラスであり、すべての共通のプロパティーとメソッドを提供します。 Java 言語の場合、Service は ExecutionService クラスのスーパークラスであり、すべての共通のプロパティーとメソッドを提供します。同じように C 言語では、関数の共通のインプリメンテーションを FmcjService から取得します。つまり、共通関数には接頭部 FmcjService が付いており、定義においては先頭に接頭部 FmcjExecutionService が付けられます。 ActiveX では継承がサポートされていないため、すべてのメソッドは ExecutionService 上で明示的に定義されます。しかし、それらのメソッドは Service アクションとして記述される点に注意してください。

以下の部分では、サービスに適用できるアクションについて説明します。関数 / メソッドの完全なリストについては、342ページの『Service』を参照してください。

Refresh()

この関数 / メソッドは、ログオン状況をサーバーからリフレッシュします (アクション呼び出し)。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

ログオンが必要です。

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.Service

ActiveX のシグニチャー

```
long ExecutionService.Refresh()
```

C 言語のシグニチャー

```
APIRET FMC_FMC_APIENTRY
       FmcjServiceRefresh( FmcjServiceHandle  service )
#define FmcjExecutionServiceRefresh FmcjServiceRefresh
```

C++ 言語のシグニチャー

```
APIRET Refresh()
```

Java のシグニチャー

```
public abstract
void refresh() throws FmcException
```

パラメーター

service 入力。 MQSeries Workflow サーバーとのセッションを表すサービス・オブジェクトのハンドル。

戻り値のデータ型

APIRET/long この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

SetPassword()

この関数 / メソッドは、ユーザーのパスワードを変更するためのものです (アクション呼び出し)。

注: パスワードには大文字小文字の区別があります。

パスワードを指定する場合には、以下の規則が適用されます。
- 最大 32 文字 (バイト) を指定できます。
- 現行のロケールに応じて、あらゆる印刷可能文字を使用できます。
- DBCS 文字は使用しないでください。

注: マルチプラットフォーム環境で作業する場合、またはコード・ページを切り替える予定の場合は、可能な限り英数字とブランクだけを使用するようにしてください。特殊文字がすべてのコード・ページで使用できるとは限らないからです。

使用上の注意
- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

ログオンが必要です。

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.Service

ActiveX のシグニチャー

```
long ExecutionService.SetPassword( BSTR newPassword )
```

C 言語のシグニチャー

```
APIRET FMC_FMC_APIENTRY
       FmcjServiceSetPassword( FmcjServiceHandle service,
                               char const *      newPassword )
#define FmcjExecutionServiceSetPassword FmcjServiceSetPassword
```

C++ 言語のシグニチャー

```
APIRET SetPassword( string const & newPassword ) const
```

Java のシグニチャー

```
public abstract
void   setPassword( String newPassword ) throws FmcException
```

パラメーター

newPassword
入力。使用する新しいパスワード。

service 入力。 MQSeries Workflow サーバーとのセッションを表すサービス・オブジェクトのハンドル。

戻り値のデータ型

long/ APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_USERID_UNKNOWN(10)
このユーザーは存在していません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_PASSWORD(12)
パスワードが MQSeries Workflow の構文規則に従っていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

UserSettings()

この関数 / メソッドは、ログオンしているユーザーのすべての設定値を戻します (アクション呼び出し)。

このサービス・オブジェクトを通してログオンしているユーザーがいない場合、設定に応じて空のオブジェクトまたはヌル・ポインターが戻されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

ログオンが必要です。

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.Service

ActiveX のシグニチャー

```
IDispatch* ExecutionService.UserSettings( long * returnCode )
```

C 言語のシグニチャー

```
APIRET FMC_FMC_APIENTRY
       FmcjServiceUserSettings( FmcjServiceHandle  service,
                                FmcjPersonHandle * user )
#define FmcjExecutionServiceUserSettings FmcjServiceUserSettings
```

C++ 言語のシグニチャー

```
APIRET UserSettings( FmcjPerson & user ) const
```

Java のシグニチャー

```
public abstract
Person userSettings() throws FmcException
```

パラメーター

returnCode 入出力。このメソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

service 入力。 MQSeries Workflow サーバーとのセッションを表すサービス・オブジェクトのハンドル。

user 入出力。ログオン・ユーザーの設定値を含む担当者オブジェクト (C++)、またはログオン・ユーザーの設定値を指す担当者ハンドルのアドレス (C)。

戻り値のデータ型

APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

IDispatch*/ Person
担当者設定値へのポインター、またはログオン・ユーザーの担当者設定値。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

第54章 作業項目のアクション

FmcjWorkitem または Workitem オブジェクトは、処理のためにユーザーに割り当てられるアクティビティー・インスタンスを表します。

ユーザーに割り当てられているその他のアイテムは、プロセス・インスタンス通知およびアクティビティー・インスタンス通知です。 FmcjItem または Item は、すべてのアイテムの共通プロパティーを表します。

C++ 言語において、FmcjWorkitem は FmcjItem クラスのサブクラスであり、すべてのプロパティーとメソッドを継承します。 Java 言語において、WorkItem は Item クラスのサブクラスであり、すべてのプロパティーとメソッドを継承します。同じように C 言語では、関数の共通のインプリメンテーションを FmcjItem から取得します。つまり、共通関数には接頭部 FmcjItem が付いており、定義においては先頭に接頭部 FmcjWorkitem が付けられます。ActiveX では継承がサポートされていないため、すべての関数は Workitem で明示的に定義されます。しかし、共通メソッドは Item アクションとして記述される点に注意してください。

作業項目は、そのオブジェクト識別子によって一意的に識別されます。

以下の図は、適切な許可が与えられている場合および関数 / メソッド記述に示されているさらに限定的な要件が満たされた場合に、起こりうる作業項目の状態とそのような状態のもとで可能なアクションをまとめています。アクションと状態は、作業項目の属するプロセス・インスタンスの状態によって決まることに注意してください。

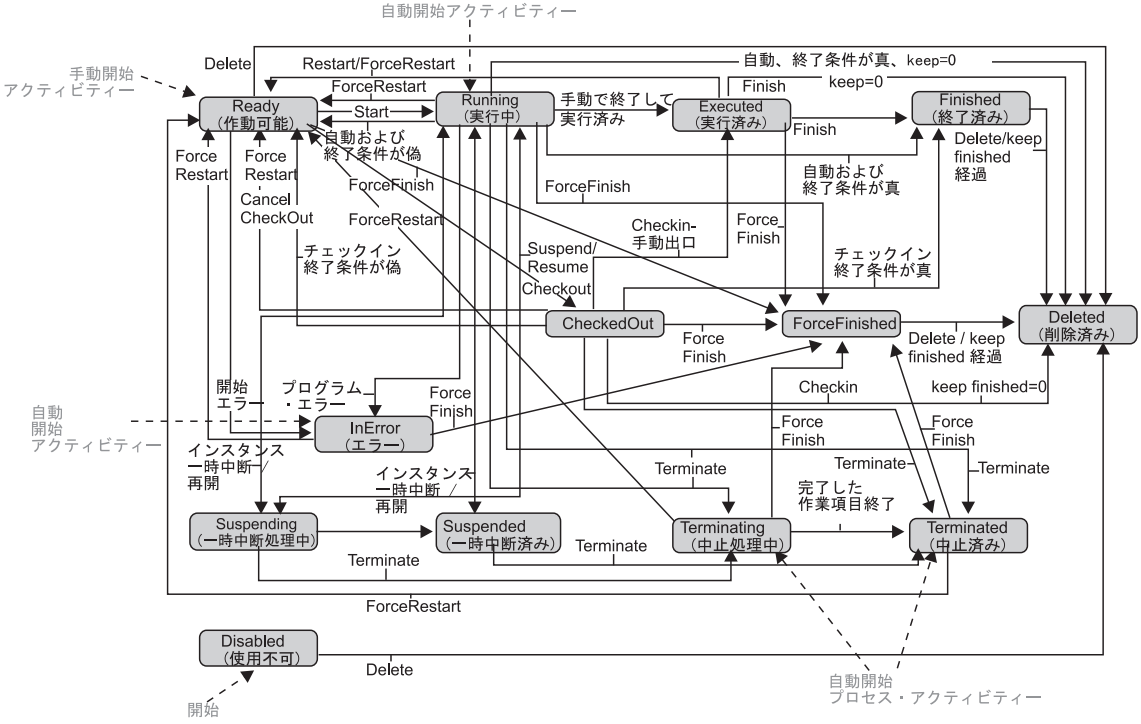


図 11. 作業項目の状態 - プロセス・インスタンスの状態は Running (実行中)

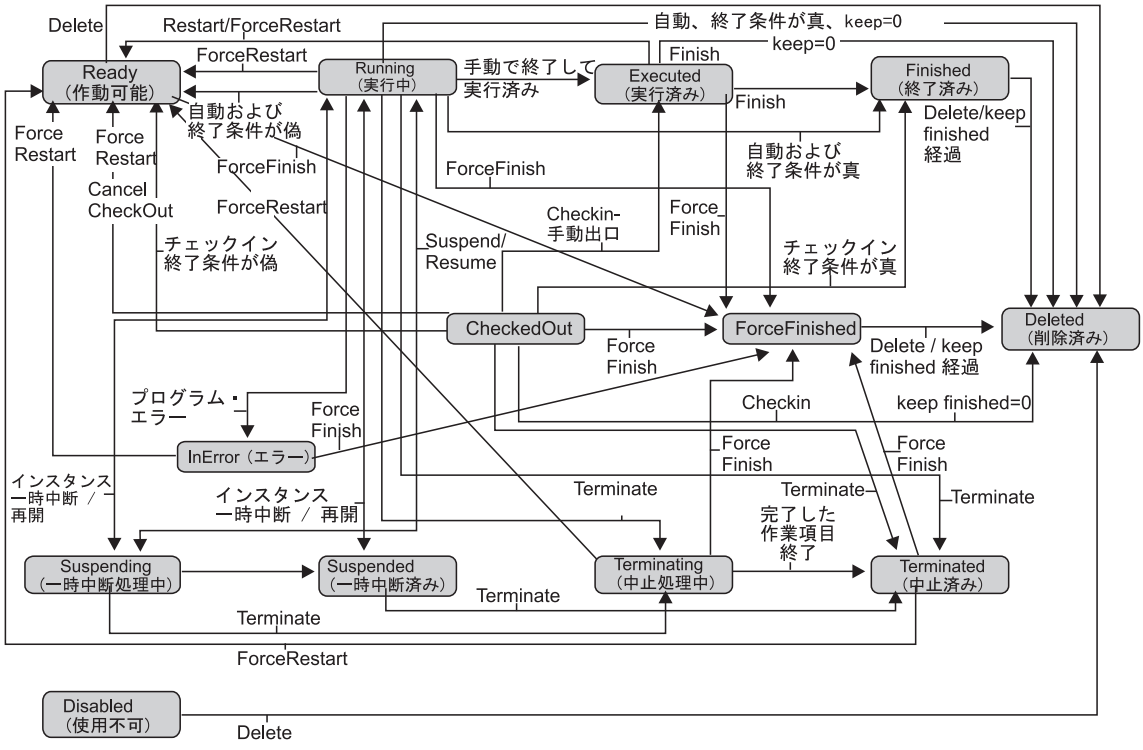


図 12. 作業項目の状態 - プロセス・インスタンスの状態は Suspending (中断処理中) または Suspended (中断)

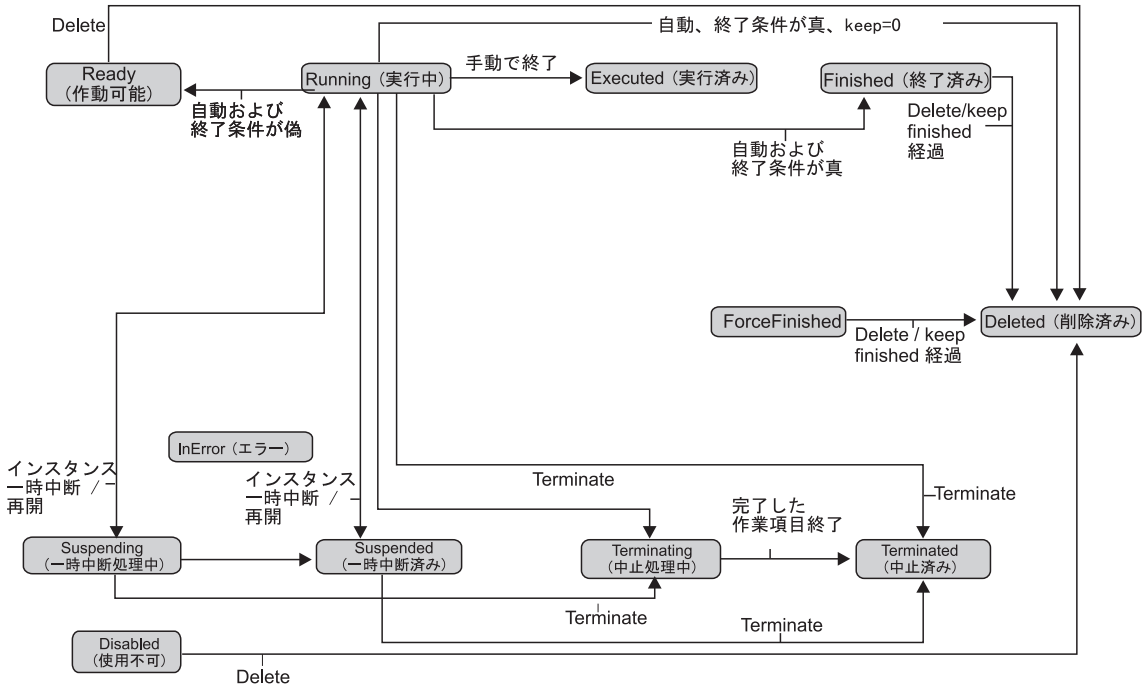


図 13. 作業項目の状態 - プロセス・インスタンスの状態は Terminating (中止処理中) または Terminated (中止済み)

以下の部分では、作業項目に適用できるアクションについて説明します。関数 / メソッドの完全なリストについては、346ページの『作業項目』を参照してください。

ActivityInstance()

関数 / メソッドは、作業項目が関連するアクティビティー・インスタンスを MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

アクティビティー・インスタンスに関するすべての 1 次情報および 2 次情報が取り出されます。

C++ の場合、初期化するアクティビティー・インスタンス・オブジェクトが空でなければ、そのオブジェクトがまず破棄されてから、新しいオブジェクトが割り当てられます。 C の場合は、オブジェクトの所有権についてはアプリケーションの側が完全に制御することになります。つまり、アクティビティー・インスタンスのハンドルがすでに何らかのオブジェクトを指しているかどうかのチェックは実行されません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- プロセスの許可
- プロセス管理の許可
- プロセス作成者として
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

ActiveX のシグニチャー

```
ActivityInstance* ActivityInstance( long * returnCode )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjWorkitemActivityInstance(
       FmcjWorkitemHandle              hdlWorkitem,
       FmcjActivityInstanceHandle *    instance )
```

C++ 言語のシグニチャー

```
APIRET ActivityInstance( FmcjActivityInstance & instance ) const
```

Java のシグニチャー

```
public abstract
ActivityInstance activityInstance() throws FmcException
```

パラメーター

hdlWorkitem 入力。照会する作業項目・オブジェクトのハンドル。

instance 入出力。検索 (初期化) するアクティビティー・インスタンス・オブジェクト。

returnCode 入出力。このメソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

戻り値のデータ型

APIRET この関数 / メソッドを呼び出した結果の戻りコード。下記の戻りコードの説明を参照してください。

ActivityInstance*/ ActivityInstance

アクティビティー・インスタンスへのポインター、または作業項目が関連するアクティビティー・インスタンス。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)

パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)

オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)

提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)

作業項目はもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)

関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)

ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

CancelCheckOut()

この関数 / メソッドは、作業項目のチェックアウトを取り消します (アクション呼び出し)。

作業項目はすでにチェックアウトされていなければならず、これにより *Ready* 状態になります。関連付けられているプロセス・インスタンスは、 *Running (実行中)*、*Suspending (一時中断処理中)*、*Suspended (一時中断済み)*、または *Terminating (中止処理中)* のいずれかの状態でなければなりません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

作業項目の所有者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.WorkItem

ActiveX のシグニチャー

```
long CancelCheckOut()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjWorkitemCancelCheckOut( FmcjWorkitemHandle hdlWorkitem )
```

C++ 言語のシグニチャー

```
APIRET CancelCheckOut()
```

Java のシグニチャー

```
public abstract
void  cancelCheckOut() throws FmcException
```

パラメーター

hdlWorkitem 入力。処理する作業項目のハンドル。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
作業項目はもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_STATE(120)
作業項目またはプロセス・インスタンスの状態は、必要とされている状態ではありません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

CheckIn()

この関数 / メソッドは、ユーザー処理のために以前にチェックアウトした作業項目からチェックインするためのものです (アクション呼び出し)。

作業項目をチェックインすると、ユーザー処理が終了したこと、そして MQSeries Workflow の制御下でのワークフロー処理を継続できることを MQSeries Workflow へ通知することになります。ユーザー処理の戻りコード、

および出力コンテナーの値 (オプション) が MQSeries Workflow へ戻されます。多くの場合、終了条件の中でそれらコンテナーの値と戻りコードを使用することによって、処理が成功したかどうかに応じてナビゲーションを継続し、また分岐条件の中で使用して継続方法を指示することができます。

出力コンテナーを指定する場合、そのコンテナーは作業項目に有効なコンテナーでなければなりません、つまりそのコンテナーには正しいスキーマおよびバージョン定義が含まれていなければなりません。言い換えると、そのコンテナーは*CheckOut()* 要求で検索された (更新済みの) 出力コンテナー、または作業項目用に検索された出力コンテナー、といったコンテナーでなければなりません。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

作業項目の所有者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

ActiveX のシグニチャー

```
long CheckIn( Container * output, long returnCode )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjWorkitemCheckIn( FmcjWorkitemHandle           hdlWorkitem,
                     FmcjReadWriteContainerHandle output,
                     long                         returnCode )
```

C++ 言語のシグニチャー

```
APIRET CheckIn( FmcjReadWriteContainer const * output,
                long                           returnCode )
```

Java のシグニチャー

```
public abstract
void   checkIn( ReadWriteContainer output,
                int                returnCode ) throws FmcException
```

パラメーター

hdlWorkitem 入力。処理する作業項目のハンドル。

output 入力。出力コンテナーを指すハンドルまたはポインター。ヌル・ポインターも可。

returnCode 入力。ユーザー処理の戻りコード。

戻り値のデータ型

long/ APIRET

この関数 / メソッドを呼び出した結果の戻りコード。下記を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)

パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
作業項目はもう存在していません。

FMC_ERROR_INVALID_CONTAINER(509)
指定された出力コンテナーは無効です。スキーマまたはバージョンが間違っています。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_STATE(120)
作業項目はチェックアウトされていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

CheckOut()

この関数 / メソッドは、Ready 状態の作業項目をユーザー処理用にチェックアウトします (アクション呼び出し)。

作業項目は、プログラムによってインプリメントされる必要があります。

この場合、チェックアウトは MQSeries Workflow の継承プログラム呼び出しメカニズムによっては処理がなされないという意味です。 MQSeries Workflow は、ユーザー固有の方法で処理がなされることを想定して、作業項目の状況を *CheckedOut* に変更します。

関連付けられているプロセス・インスタンスは、*Running (実行中)* 状態でなければなりません。

呼び出し元は、特定のオペレーティング・システムごとのプログラム定義を要求できます。要求するプログラム・データを指定するには、下記の列挙型を使用することができます。

ActiveX WorkitemProgramRetrieval

C 言語 FmcjWorkitemProgramRetrieval

C++ FmcjWorkitem::ProgramRetrieval

JAVA com.ibm.workflow.api.WorkItemPackage.ProgramRetrieval

列挙型定数には以下の値を使用できます。可能な限り整数値ではなく記号名を使うようにしてください。

NotSet 値が設定されていないことを示します。

- **ActiveX** WIProgramRetrieval_NotSet
- **C 言語** Fmc_WS_NotSet
- **C++** FmcjWorkitem::NotSet
- **JAVA** ProgramRetrieval.NOT_SET

CommonDataOnly

すべてのプラットフォームに共通のデータ (記述、アイコン、不在標識、および入出力コンテナー) だけを戻します。プラットフォーム指定は無視されます。

- **ActiveX** WIProgramRetrieval_CommonDataOnly
- **C 言語** Fmc_WS_CommonDataOnly
- **C++** FmcjWorkitem::CommonDataOnly
- **JAVA** ProgramRetrieval.COMMON_DATA_ONLY

SpecifiedDefinitions

指定されたプラットフォームのプログラム定義を戻します。プラットフォームを指定する必要があります。

- **ActiveX** WIProgramRetrieval_SpecifiedDefinitions

C 言語 Fmc_WS_SpecifiedDefinitions

C++ FmcjWorkitem::SpecifiedDefinitions

JAVA ProgramRetrieval.SPECIFIED_DEFINITIONS

AllDefinitions 使用可能なすべてのプログラム定義を戻します。プラットフォーム指定は無視されます。

ActiveX WIProgramRetrieval_AllDefinitions

C 言語 Fmc_WS_AllDefinitions

C++ FmcjWorkitem::AllDefinitions

JAVA ProgramRetrieval.ALL_DEFINITIONS

プログラム定義を検索するプラットフォームを指定するために、以下の列挙型を使用できます。

ActiveX ImplementationDataBasis

C 言語 FmcjImplementationDataBasis

C++ FmcjImplementationData::Basis

JAVA com.ibm.workflow.api.ProgramDataPackage.Basis

列挙型定数には以下の値を使用できます。可能な限り整数値ではなく記号名を使うようにしてください。

NotSet 値が設定されていないことを示します。

ActiveX Basis_NotSpecified

C 言語 Fmc_DP_NotSet

C++ FmcjImplementationData::NotSpecified

JAVA Basis.NOT_SPECIFIED

OS2 OS/2 プラットフォーム用のプログラム定義を要求することを示します。

ActiveX Basis_OS2

C 言語 Fmc_DP_OS2

C++ FmcjImplementationData::OS2

JAVA Basis.OS2

AIX AIX プラットフォーム用のプログラム定義を要求することを示します。

ActiveX Basis_AIX

C 言語 Fmc_DP_AIX

C++ FmcjImplementationData::AIX

JAVA Basis.AIX

HPUX HP-UX プラットフォーム用のプログラム定義を要求することを示します。

ActiveX Basis_HPUX

C 言語 Fmc_DP_HPUX

C++ FmcjImplementationData::HPUX

JAVA Basis.HPUX

Windows95 Windows 95 プラットフォーム用のプログラム定義を要求することを示します。

ActiveX Basis_Windows95

C 言語 Fmc_DP_Windows95

C++ FmcjImplementationData::Windows95

JAVA Basis.WINDOWS_95

WindowsNT Windows NT プラットフォーム用のプログラム定義を要求することを示します。

ActiveX Basis_WindowsNT

C 言語 Fmc_DP_WindowsNT

C++ FmcjImplementationData::WindowsNT

JAVA Basis.WINDOWS_NT

OS390 OS/390(R) プラットフォーム用のプログラム定義を要求することを示します。

ActiveX Basis_OS390

C 言語 Fmc_DP_OS390

C++ FmcjImplementationData::OS390

JAVA Basis.WINDOWS_OS390

Solaris Solaris プラットフォーム用のプログラム定義を要求することを示します。

ActiveX Basis_Solaris

C 言語 Fmc_DP_Solaris

C++ FmcjImplementationData::Solaris

JAVA Basis.Solaris

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

作業項目の所有者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.WorkItem

ActiveX のシグニチャー

```
ProgramData CheckOut( WorkitemProgramRetrieval requestedData,
                      ImplementationDataBasis  platform,
                      long *                   returnCode   )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjWorkitemCheckOut( FmcjWorkitemHandle                hdlWorkitem,
                      enum FmcjWorkitemProgramRetrieval requestedData,
                      enum FmcjImplementationDataBasis  platform,
                      FmcjProgramDataHandle *           programData  )
```

C++ 言語のシグニチャー

```
APIRET CheckOut( ProgramRetrieval                  requestedData,
                 FmcjImplementationData::Basis     platform,
                 FmcjProgramData &                 programData  )
```

Java のシグニチャー

```
public abstract
ReadOnlyContainer checkOut() throws FmcException
public abstract
ProgramData       checkOut2(
                  ProgramRetrieval  requestedData,
                  Basis             platform     ) throws FmcException
```

パラメーター

hdlWorkitem 入力。処理する作業項目のハンドル。

platform 入力。プログラム定義が戻されるプラットフォーム。

programData 入出力。設定するプログラム定義に対するハンドルのアドレス (C)。またはそのプログラム定義オブジェクト (C++)。

requestedData
入力。戻されるプログラム定義を示す標識。

returnCode 入出力。このメソッドの呼び出しの戻りコード。下記を参照してください。

戻り値のデータ型

APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

ProgramData プログラム定義。

ReadOnlyContainer
作業項目の入力コンテナー。このコンテナーはプログラム定義の一部です。バージョン 2 との互換性のために戻されます。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_CHECKOUT_NOT_POSSIBLE(503)
作業項目をチェックアウトできません。

FMC_ERROR_DOES_NOT_EXIST(118)
作業項目はもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_STATE(120)
作業項目またはプロセス・インスタンスの状態が間違っています。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Finish()

この関数 / メソッドは、手動終了作業項目の実行を終了します (アクション呼び出し)。

作業項目は、*Executed* 状態でなければなりません。すなわち、少なくとも 1 回は実行する必要があります。その後、作業項目は *Finished* 状態になります。それは、「完了したアイテムの削除」オプションに応じて削除されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

作業項目の所有者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.WorkItem

ActiveX のシグニチャー

```
long Finish()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjWorkitemFinish( FmcjWorkitemHandle hdlWorkitem )
```

C++ 言語のシグニチャー

```
APIRET Finish()
```

Java のシグニチャー

```
public abstract
void   finish() throws FmcException
```

パラメーター

hdlWorkitem 入力。処理する作業項目のハンドル。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
作業項目はもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_STATE(120)
作業項目の状態が間違っています。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

ForceFinish()

この関数 / メソッドは、MQSeries Workflow がこの事象を認識しなかった場合に、完了したと認識されている作業項目の実行を終了します (アクション呼び出し)。

実行サーバーがアクティビティー・インプリメンテーションの完了メッセージを受信する前に打ち切られた場合に、そのようになることがあります。

プログラムによってインプリメントされる作業項目は、*Ready (作動可能)*、*Running (実行中)*、*Executed (実行)*、 *CheckedOut (チェックアウト済み)*、*InError (エラー)*、*Terminating (中止処理中)*、 *Terminated (中止)* のいずれかの状態でなければなりません。プロセスによってインプリメントされる作業項目は、*Ready (作動可能)*、*Executed (実行済み)*、 *InError (エラー)*、または *Terminated (中止済み)* のいずれかの状態でなければなりません。関連したプロセス・インスタンスは、*Running (実行中)*、*Suspending (一時中断処理中)*、*Suspended (一時中断済み)*、または *Terminating (中止処理中)* のいずれかの状態でなければなりません。

その後、作業項目は *ForceFinished* 状態になります。終了条件は真であるとみなされ、ナビゲーションが進められます。

作業項目は、「完了したアイテムの削除」オプションに応じて削除されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

作業項目所有者として、なおかつ以下のいずれか
- プロセス管理の許可

- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.WorkItem

ActiveX のシグニチャー

```
long ForceFinish()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjWorkitemForceFinish( FmcjWorkitemHandle hdlWorkitem )
```

C++ 言語のシグニチャー

```
APIRET ForceFinish()
```

Java のシグニチャー

```
public abstract
void   forceFinish() throws FmcException
```

パラメーター

hdlWorkitem 入力。処理する作業項目のハンドル。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
作業項目はもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_STATE(120)
作業項目の状態が間違っています。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

ForceRestart()

この関数 / メソッドは、MQSeries Workflow が作業項目を再始動できるようにします (アクション呼び出し)。

プログラムによってインプリメントされる作業項目は、*Running (実行中)*、*Executed (実行済み)*、 *CheckedOut (チェックアウト済み)*、*InError (エラー)*、*Terminating (中止処理中)*、 *Terminated (中止済み)* のいずれかの状態でなければなりません。プロセスによってインプリメントされる作業項目は、*Executed (実行済み)*、*InError (エラー)*、または *Terminated (中止済み)* のいずれかの状態でなければなりません。関連したプロセス・インスタンスは、*Running (実行中)*、*Suspending (一時中断処理中)*、または *Suspended (一時中断済み)* のいずれかの状態でなければなりません。

その後、それは *Ready* 状態にリセットされます。自動アクティビティー・インスタンスは、手動で開始することが必要になったことに注意してください。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

作業項目所有者として、なおかつ以下のいずれか
- プロセス管理の許可
- プロセス管理者として
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

ActiveX のシグニチャー

```
long ForceRestart()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjWorkitemForceRestart(
       FmcjWorkitemHandle hdlWorkitem )
```

C++ 言語のシグニチャー

```
APIRET ForceRestart()
```

Java のシグニチャー

```
public abstract
void   forceRestart() throws FmcException
```

パラメーター

hdlWorkitem 入力。処理する作業項目のハンドル。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
作業項目はもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_STATE(120)
作業項目またはプロセス・インスタンスの状態が間違っています。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

InContainer()

この関数 / メソッドは、作業項目に関連する入力コンテナーを、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。

- 作業項目の所有者として

- 作業項目許可
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.WorkItem

ActiveX のシグニチャー

```
long InContainer( Container * input )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjWorkitemInContainer( FmcjWorkitemHandle            hdlWorkitem,
                         FmcjReadOnlyContainerHandle * input )
```

C++ 言語のシグニチャー

```
APIRET InContainer( FmcjReadOnlyContainer & input ) const
```

Java のシグニチャー

```
public abstract
ReadOnlyContainer inContainer() throws FmcException
```

パラメーター

hdlWorkitem 入力。処理する作業項目のハンドル。

input 入出力。入力コンテナー。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

ReadOnlyContainer
入力コンテナー。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
作業項目はもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

OutContainer()

この関数 / メソッドは、作業項目に関連する出力コンテナーを、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- 作業項目の所有者として
- 作業項目許可
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

ActiveX のシグニチャー

```
long OutContainer( Container * output )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjWorkitemOutContainer( FmcjWorkitemHandle             hdlWorkitem,
                          FmcjReadWriteContainerHandle * output )
```

C++ 言語のシグニチャー

```
APIRET OutContainer( FmcjReadWriteContainer & output ) const
```

Java のシグニチャー

```
public abstract
ReadWriteContainer outContainer() throws FmcException
```

パラメーター

hdlWorkitem 入力。処理する作業項目のハンドル。

output 入出力。出力コンテナー。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

ReadWriteContainer
出力コンテナー。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
作業項目はもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

PersistentObject()

この関数 / メソッドは、MQSeries Workflow 実行サーバーから渡されたオブジェクト識別子で識別される作業項目を取り出します (アクション呼び出し)。

作業項目の検索元の MQSeries Workflow 実行サーバーは、実行サービス・オブジェクトによって識別されます。その場合、一時オブジェクトは、作業項目のすべての情報 (1 次および 2 次) を使って作成また更新されます。

C++ の場合、初期化する作業項目・オブジェクトが空でなければ、そのオブジェクトがまず破棄されてから、新しいオブジェクトが割り当てられます。 C の場合は、オブジェクトの所有権についてはアプリケーションの側が完全に制御することになります。つまり、作業項目のハンドルがすでに何らかのオブジェクトを指しているかどうかのチェックは実行されません。 Java の場合、作業項目は新たに作成されます。実行サービスがファクトリーとして機能します。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

以下のいずれか 1 つです。
- 作業項目の所有者として

- 作業項目許可
- システム管理者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.ExecutionService

ActiveX のシグニチャー

```
long PersistentObject( ExecutionService * service, BSTR oid )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjWorkitemPersistentObject( FmcjExecutionServiceHandle service,
                              char const *               oid,
                              FmcjWorkitemHandle *       hdlWorkitem )
```

C++ 言語のシグニチャー

```
APIRET PersistentObject( FmcjExecutionService const & service,
                         string const &               oid )
```

Java のシグニチャー

```
public abstract
  WorkItem ExecutionService.persistentWorkItem( String oid )
throws FmcException
```

パラメーター

hdlWorkitem 入出力。設定する作業項目オブジェクトのハンドルのアドレス
oid 入力。検索する作業項目のオブジェクト識別子。
service 入力。実行サーバーとのセッションを表すサービス・オブジェクト。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。
WorkItem 取り出された作業項目。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。
FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。
FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。
FMC_ERROR_DOES_NOT_EXIST(118)
作業項目はもう存在していません。
FMC_ERROR_INVALID_OID(805)
指定した oid は無効です。
FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。
FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。
FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。
FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。
FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。
FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Restart()

この関数 / メソッドは、MQSeries Workflow に対して作業項目の再始動を依頼します (アクション呼び出し)。

作業項目は、*Executed* 状態でなければなりません。その後、それは *Ready (作動可能)* 状態にリセットされます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

作業項目の所有者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

ActiveX のシグニチャー

```
long Restart()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjWorkitemRestart( FmcjWorkitemHandle hdlWorkitem )
```

C++ 言語のシグニチャー

```
APIRET Restart()
```

Java のシグニチャー

```
public abstract
void   restart() throws FmcException
```

パラメーター

hdlWorkitem 入力。処理する作業項目のハンドル。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
作業項目はもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_STATE(120)
作業項目の状態が間違っています。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Start()

この関数 / メソッドは、作動可能状態の作業項目を開始します (アクション呼び出し)。

関連付けられているプロセス・インスタンスは、*Running (実行中)* 状態でなければなりません。

関連したアクティビティー・インスタンスがプログラムによってインプリメントされている場合、ログオン・ユーザーに関連したプログラム実行エージェント上でプログラムが開始されます。

作業項目は *Running (実行中)* 状態になります。アクティビティー・インプリメンテーションまたは関連したプロセス・アクティビティーを開始できない場合、作業項目は *InError (エラー)* 状態になります。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

作業項目の所有者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.WorkItem

ActiveX のシグニチャー

```
long Start()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjWorkitemStart( FmcjWorkitemHandle hdlWorkitem )
```

C++ 言語のシグニチャー

```
APIRET Start()
```

Java のシグニチャー

```
public abstract
void   start() throws FmcException
```

パラメーター

hdlWorkitem 入力。処理する作業項目のハンドル。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
作業項目はもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_STATE(120)
作業項目またはプロセス・インスタンスの状態が間違っています。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

StartTool()

この関数 / メソッドは、指定したサポート・ツールを開始します (アクション呼び出し)。

サポート・ツールは、作業項目の派生元のアクティビティー・インスタンスに関連したツールの 1 つでなければなりません。ログオン・ユーザーに関連したプログラム実行エージェント上で開始されます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

作業項目の所有者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkItem

ActiveX のシグニチャー

```
long StartTool( BSTR toolName )
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjWorkitemStartTool( FmcjWorkitemHandle hdlWorkitem,
                       char const *       toolName   )
```

C++ 言語のシグニチャー

```
APIRET StartTool( string const & toolName ) const
```

Java のシグニチャー

```
public abstract
void   startTool( String toolName ) throws FmcException
```

パラメーター

hdlWorkitem 入力。処理する作業項目のハンドル。

toolName 入力。開始するサポート・ツール。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
作業項目はもう存在していません。

FMC_ERROR_INVALID_TOOL(129)
ツール名が 1 つも提供されていないか、指定したツールが作業項目に対して定義されていません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

Terminate()

この関数 / メソッドは、プログラムまたはプロセスによってインプリメントされた作業項目を中止します (アクション呼び出し)。

作業項目がプログラムによってインプリメントされている場合、その状態は *CheckedOut (チェックアウト済み)*、または *Running (実行中)* のいずれかでなければならず、プロセス・インスタンスは *Running (実行中)*、*Suspending (一時中断処理中)*、または *Suspended (一時中断)* のいずれかの状態でなければなりません。作業項目がプロセスによってインプリメントされている場合、その状態は *Running (実行中)*、*Suspending (一時中断処理中)*、または *Suspended (一時中断済み)* のいずれかでなければならず、プロセス・インスタンスは *Running (実行中)*、*Suspending (一時中断処理中)*、 *Suspended (一時中断済み)*、または *Terminating (中止処理中)* のいずれかの状態でなければなりません。

プロセスによってインプリメントされる作業項目は、制御の自立性に関して自立走行式でないそのサブプロセスと共に中止させられます。

その後、作業項目は *Terminated (中止処理中)* または *Terminating (中止済み)* 状態になります。

Terminated (中止済み) 状態に達すると終了条件が偽とみなされ、出力コンテナー、特に戻りコード (_RC) は設定されず、ナビゲーションが終了します。ナビゲーションはプロセス管理権をもつユーザーによって明示的に継続させることが可能です。つまり、ForceFinish() または ForceRestart() 修復アクションを呼び出すことができます。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

作業項目の所有者として

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.WorkItem

ActiveX のシグニチャー

```
long Terminate()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjWorkitemTerminate( FmcjWorkitemHandle hdlWorkitem )
```

C++ 言語のシグニチャー

```
APIRET Terminate()
```

Java のシグニチャー

```
public abstract
void   terminate() throws FmcException
```

パラメーター

hdlWorkitem　入力。中止する作業項目のハンドル。

戻り値のデータ型

long/ APIRET　このメソッドの呼び出しの戻りコード。下記を参照してください。

戻りコード / FmcException

FMC_OK(0)　関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
作業項目はもう存在していません。

FMC_ERROR_NOT_AUTHORIZED(119)
関数 / メソッドを使う許可がありません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_WRONG_STATE(120)
作業項目またはプロセス・インスタンスの状態が間違っています。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

第55章 ワークリストのアクション

FmcjWorklist あるいは Worklist オブジェクトは、一連のアイテム、つまり一連の作業項目または通知を表します。このリストによってアクセスできるすべてのアイテムは特性が同じです。その特性はフィルターによって指定されます。また、ソート基準を適用することができ、その後、実行サーバーからクライアントへと転送されるアイテムの数を制限するしきい値を適用できます。

ワークリストの定義は永続的に保管されます。しかし、ワークリストに含まれるアイテムは、照会時に動的に集められたものです。

ワークリストは、その名前、タイプ、所有者によって一意に識別されます。これは、一般的なアクセスを目的として定義することができます。その場合にはパブリック *(public)* タイプとなります。あるいは、特定の一部のユーザーが使用するようにも定義できます。その場合はプライベート *(private)* タイプとなります。

定義可能なその他のリストには、プロセス・テンプレート・リストとプロセス・インスタンス・リストがあります。 FmcjPersistentList または PersistentList は、すべてのリストの共通プロパティーを表します。

C++ 言語において、FmcjWorklist は FmcjPersistentList クラスのサブクラスであり、すべてのプロパティーとメソッドを継承します。 Java 言語において WorkList は PersistentList クラスのサブクラスであり、すべてのプロパティーとメソッドを継承します。同じように C 言語では、関数の共通のインプリメンテーションを FmcjPersistentList から取得します。つまり、共通関数には接頭部 FmcjPersistentList が付いており、定義においては先頭に接頭部 FmcjWorklist が付けられます。 ActiveX では継承がサポートされていないため、すべての関数は Worklist で明示的に定義されます。しかし、共通メソッドは PersistentList アクションとして記述される点に注意してください。

以下の部分では、ワークリストに適用できるアクションについて説明します。関数 / メソッドの完全なリストについては、349ページの『ワークリスト』を参照してください。

QueryActivityInstanceNotifications()

この関数 / メソッドは、指定されたワークリストによって特徴付けられるすべてのアクティビティー・インスタンス通知の 1 次情報を、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

当てはまるアクティビティー・インスタンス通知の集まりの中で、ユーザーに許可が与えられているものだけが検索されます。ユーザーがアクティビティー・インスタンス通知に対する許可を付与されるのは、下記の場合です。

- そのアクティビティー・インスタンス通知の所有者である場合
- 作業項目権限を付与されている場合
- システム管理者である場合

各アクティビティー・インスタンス通知ごとに取り出される 1 次情報は以下のとおりです。

- ActivityType
- Category
- CreationTime
- Description
- Icon
- Implementation
- Kind
- LastModificationTime
- Name
- Owner
- Priority
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State
- SupportTools

C および C++ では、検索したアクティビティー・インスタンス通知はすべて、提供されるアクティビティー・インスタンス通知・ベクトルに追加されます。現在ワークリストに含まれているアクティビティー・インスタンス通知だけを読みたい場合は、まずベクトルをクリアした後でこの関数 / メソッドを呼び出さなければなりません。つまり、C 言語ではハンドルを 0 に設定し、C++ API ではベクトルのすべての要素を消去する必要があります。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.WorkList

ActiveX のシグニチャー

```
long QueryActivityInstanceNotifs()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjWorklistQueryActivityInstanceNotifications(
       FmcjWorklistHandle                              hdlList,
       FmcjActivityInstanceNotificationVectorHandle * notifications )
```

C++ 言語のシグニチャー

```
APIRET QueryActivityInstanceNotifications(
       vector<FmcjActivityInstanceNotification> & notifications ) const
```

Java のシグニチャー

```
public abstract
ActivityInstanceNotification[] queryActivityInstanceNotifications()
throws FmcException
```

パラメーター

hdlList 入力。照会するワークリストのハンドル。

notifications 入出力。適格アクティビティー・インスタンス通知のベクトル。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

ActivityInstanceNotification[]
適格アクティビティー・インスタンス通知。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
ワークリストはもう存在していません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- ActiveX の例については、849ページの『ワークリストから作業項目を照会する (ActiveX)』を参照してください。
- C 言語の例については、850ページの『ワークリストから作業項目を照会する (C 言語)』を参照してください。
- C++ の例については、851ページの『ワークリストから作業項目を照会する (C++)』を参照してください。
- Java の例については、853ページの『ワークリストから作業項目を照会する (Java)』を参照してください。

QueryItems()

この関数 / メソッドは、指定されたワークリストによって特徴付けられるすべてのアイテムの 1 次情報を、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

当てはまるアイテムの集まりの中で、ユーザーに許可が与えられているものだけが検索されます。ユーザーがアイテムに対する許可を付与されるのは、下記の場合です。

- そのアイテムの所有者である場合
- 作業項目権限を付与されている場合
- システム管理者である場合

各アイテムごとに取り出される 1 次情報は以下のとおりです。

- Category
- CreationTime
- Description
- Icon
- Kind
- LastModificationTime
- Name

- Owner
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State

アイテムが実際の作業項目またはアクティビティー・インスタンス通知である場合、以下の 1 次情報が追加で検索されます。
- ActivityType
- Implementation
- Priority
- SupportTools

C および C++ の場合、検索したすべてのアイテムは、提供されるアイテム・ベクトルに追加されます。現在ワークリストに含まれているアイテムだけを読みたい場合は、まずベクトルをクリアした後でこの関数 / メソッドを呼び出さなければなりません。つまり、C 言語ではハンドルを 0 に設定し、C++ API ではベクトルのすべての要素を消去する必要があります。

使用上の注意
- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	該当しない
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkList

C 言語のシグニチャー

```
APIRET FMC_APIENTRY
FmcjWorklistQueryItems( FmcjWorklistHandle     hdlList,
                        FmcjItemVectorHandle * items )
```

C++ 言語のシグニチャー

```
APIRET QueryItems( vector<FmcjItem> & items ) const
```

Java のシグニチャー

```
public abstract Item[] queryItems() throws FmcException
```

パラメーター

hdlList 入力。照会するワークリストのハンドル。

items 入出力。適格アイテムのベクトル。

戻り値のデータ型

APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

Item[] 適格アイテム。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
ワークリストはもう存在していません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- C 言語の例については、850ページの『ワークリストから作業項目を照会する (C 言語)』を参照してください。
- C++ の例については、851ページの『ワークリストから作業項目を照会する (C++)』を参照してください。
- Java の例については、853ページの『ワークリストから作業項目を照会する (Java)』を参照してください。

QueryProcessInstanceNotifications()

この関数 / メソッドは、指定されたワークリストによって特徴付けられるすべてのプロセス・インスタンス通知の 1 次情報を、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

当てはまるプロセス・インスタンス通知の集まりの中で、ユーザーに許可が与えられているものだけが検索されます。ユーザーはプロセス・インスタンス通知の許可を与えられるのは、次の場合です。

- そのプロセス・インスタンス通知の所有者である場合
- 作業項目権限を付与されている場合
- システム管理者である場合

各プロセス・インスタンス通知ごとに取り出される 1 次情報は以下のとおりです。

- Category
- CreationTime
- Description
- Icon
- Kind
- LastModificationTime
- Name
- Owner
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State

C および C++ では、検索したプロセス・インスタンス通知はすべて、提供されるプロセス・インスタンス通知・ベクトルに追加されます。現在ワークリストに含まれているプロセス・インスタンス通知だけを読みたい場合は、まずベクトルをクリアした後でこの関数 / メソッドを呼び出さなければなりません。つまり、C 言語ではハンドルを 0 に設定し、C++ API ではベクトルのすべての要素を消去する必要があります。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX	IBM MQSeries Workflow コントロール 3.1
C 言語	fmcjcrun.h
C++	fmcjprun.hxx
JAVA	com.ibm.workflow.api.WorkList

ActiveX のシグニチャー

```
long QueryProcessInstanceNotifs()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjWorklistQueryProcessInstanceNotifications(
       FmcjWorklistHandle                              hdlList,
       FmcjProcessInstanceNotificationVectorHandle * notifications )
```

C++ 言語のシグニチャー

```
APIRET QueryProcessInstanceNotifications(
       vector<FmcjProcessInstanceNotification> & notifications ) const
```

Java のシグニチャー

```
public abstract
ProcessInstanceNotification[] queryProcessInstanceNotifications()
throws FmcException
```

パラメーター

hdlList 入力。照会するワークリストのハンドル。

notifications 入出力。適格プロセス・インスタンス通知のベクトル。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

ProcessInstanceNotification[]
適格プロセス・インスタンス通知。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
ワークリストはもう存在していません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- ActiveX の例については、849ページの『ワークリストから作業項目を照会する (ActiveX)』を参照してください。
- C 言語の例については、850ページの『ワークリストから作業項目を照会する (C 言語)』を参照してください。
- C++ の例については、851ページの『ワークリストから作業項目を照会する (C++)』を参照してください。
- Java の例については、853ページの『ワークリストから作業項目を照会する (Java)』を参照してください。

QueryWorkitems()

この関数 / メソッドは、指定されたワークリストによって特徴付けられるすべての作業項目の 1 次情報を、MQSeries Workflow 実行サーバーから検索します (アクション呼び出し)。

当てはまる作業項目の集まりの中で、ユーザーに許可が与えられているものだけが検索されます。ユーザーは作業項目の許可を与えられるのは、次の場合です。

- その作業項目の所有者である場合
- 作業項目権限を付与されている場合
- システム管理者である場合

各作業項目ごとに取り出される 1 次情報は以下のとおりです。

- ActivityType
- Category
- CreationTime
- Description
- Icon
- Implementation
- Kind
- LastModificationTime
- Name
- Owner
- Priority
- ProcessInstanceName
- ReceivedAs
- ReceivedTime
- StartTime
- State
- SupportTools

C および C++ では、検索した作業項目はすべて、提供される作業項目・ベクトルに追加されます。現在ワークリストに含まれている作業項目だけを読みたい場合は、まずベクトルをクリアした後でこの関数 / メソッドを呼び出さなければなりません。つまり、C 言語ではハンドルを 0 に設定し、C++ API ではベクトルのすべての要素を消去する必要があります。

使用上の注意

- 141ページの『アクション関数 / メソッド』にある一般的な情報を参照してください。

許可

なし

必要な接続

MQSeries Workflow 実行サーバー

API インターフェース宣言

ActiveX IBM MQSeries Workflow コントロール 3.1

C 言語 fmcjcrun.h

C++ fmcjprun.hxx

JAVA com.ibm.workflow.api.WorkList

ActiveX のシグニチャー

```
long QueryWorkitems()
```

C 言語のシグニチャー

```
APIRET FMC_APIENTRY FmcjWorklistQueryWorkitems(
       FmcjWorklistHandle         hdlList,
       FmcjWorkitemVectorHandle * workitems )
```

C++ 言語のシグニチャー

```
APIRET QueryWorkitems( vector<FmcjWorkitem> & workitems ) const
```

Java のシグニチャー

```
public abstract
WorkItem[] queryWorkItems() throws FmcException
```

パラメーター

hdlList 入力。照会するワークリストのハンドル。

workitems 入出力。適格作業項目のベクトル。

戻り値のデータ型

long/ APIRET このメソッドの呼び出しの戻りコード。下記を参照してください。

WorkItem[] 適格作業項目。

戻りコード / FmcException

FMC_OK(0) 関数 / メソッドは正常に完了しました。

FMC_ERROR(1)
パラメーターが未定義の位置を参照しています。たとえば、ハンドルのアドレスが 0 になっています。

FMC_ERROR_EMPTY(122)
オブジェクトはまだデータベースから読み取られていません。つまりこのオブジェクトはまだ永続オブジェクトを表していません。

FMC_ERROR_INVALID_HANDLE(130)
提供されたハンドルは無効です。それは 0 であるか、または要求した型のオブジェクトを指していません。

FMC_ERROR_DOES_NOT_EXIST(118)
ワークリストはもう存在していません。

FMC_ERROR_NOT_LOGGED_ON(106)
ログオンしていません。

FMC_ERROR_COMMUNICATION(13)
指定されたサーバーに到達できません。接続先サーバーがプロファイルの中で定義されていなければなりません。

FMC_ERROR_INTERNAL(100)
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_FORMAT(103)
内部メッセージの形式エラーです。 IBM 担当員に連絡してください。

FMC_ERROR_MESSAGE_SIZE_EXCEEDED(821)
戻されるメッセージが許可された最大サイズを超えました。システム、システム・グループ、またはドメインの MAXIMUM_MESSAGE_SIZE 定義を参照してください。

FMC_ERROR_TIMEOUT(14)
タイムアウトになりました。

例

- ActiveX の例については、849ページの『ワークリストから作業項目を照会する (ActiveX)』を参照してください。

- C 言語の例については、850ページの『ワークリストから作業項目を照会する (C 言語)』を参照してください。
- C++ の例については、851ページの『ワークリストから作業項目を照会する (C++)』を参照してください。
- Java の例については、853ページの『ワークリストから作業項目を照会する (Java)』を参照してください。

第8部 ActiveX コントロールを使った作業

以下の章では、ActiveX コントロールについて説明します。

注: ActiveX のシグニチャーはオブジェクト定義言語 (ODL) で提供されています。たとえば、*BSTR* 型は、VisualBasic の型が実際には String であるストリングに使用されます。

第56章 ExecutionService コントロール

ExecutionServiceArray 中の ExecutionService にアクセスするには、ExecutionService コントロールを Workflow コントロールに接続する必要があります。この接続は、ExecutionService コントロール・クラスの ConnectGUI() メソッドによって確立します。接続が確立された時点で、ユーザーは ExecutionService コントロール・ウィンドウのツリー・ビュー内に表示される ExecutionService への完全なアクセスが得られます。

第57章 リスト・コントロール

たとえば、WorklistArray 中のワークリストにアクセスするには、各ワークリストを Worklist コントロールに接続する必要があります。この接続は、Worklist コントロール・クラスの ConnectGUI() メソッドによって確立します。接続が確立された時点で、ユーザーは Worklist コントロール中の Worklist オブジェクトに対するアクセスが得られます。この同じメカニズムを使用することによって、GUI 内で使用される他のどのタイプのリストにも接続できます。

第58章 モニター・コントロール

Monitor コントロール OCX は、Process Monitor GUI をインプリメントする ActiveX コンポーネントです。モニターにアクセスするには、Monitor コントロールを Workflow コントロールに接続する必要があります。この接続は、Monitor コントロール・クラスの ConnectGUI() メソッドによって確立します。接続が確立された時点で、インスタンス・モニター・オブジェクトに対する完全なアクセスが得られます。

Monitor コントロールを使用すれば、1 つのプロセス・モデルのグラフを表示できます。変更されたアクティビティー・インスタンス状態を表示するものとしては、Refresh() メソッドが用意されています。この OCX 1 つで、異なる複数のプロセス・モデルのグラフを表示することは考慮されていません。異なるプロセス・モデルを表示するには、複数の別個の Monitor OCX インスタンスを使うことが必要です。

第59章 ActiveX コントロール・メソッドの典型的なシナリオ

たとえば特定のワークリストの名前を入手する、といった典型的なシナリオがどうなるかを知りたいと思われるかもしれません。前提条件確立のための一連のステップは、以下のようになります。

1. ExecutionService コントロールのアクセス・メソッド **ExecutionServiceArray** によって、ローカル・コントロール・オブジェクト ExecutionServiceArray にアクセスできます。
2. その ExecutionServiceArray のアクション・メソッド **Add** によって ExecutionService のインデックスが提供され、アクセス・メソッド **GetAt** によって一時 ExecutionService オブジェクトが提供されます。
3. ExecutionService のアクション・メソッド **QueryWorklists** によって、一時 Worklist オブジェクトが提供されます。
4. ExecutionService のアクセス・メソッド **WorklistArray** によって、ローカル・コントロール・オブジェクト WorklistArray へのアクセスが得られます。
5. WorklistArray のアクセス・メソッド **GetAt** によって、その WorklistArray から特定の Worklist オブジェクトにアクセスできます。
6. Worklist のアクセス・メソッド **Name** によって、特定の Worklist の名前を取得できます。

第60章 MQWorkflowCtrl

メソッド

MQWorkflow コントロールは以下に示すメソッドをサポートします。

ConfigurationID

プロファイル・アクセスに使用する構成 ID を戻します。

シグニチャー

```
BSTR ConfigurationID()
```

戻り値のデータ型

BSTR 構成 ID。

Connect

このメソッドは、特定のスレッドに関する MQSeries Workflow API 処理を初期化します。

シグニチャー

```
long Connect()
```

戻り値のデータ型

long 処理が正常に完了すると、FMC_OK が戻されます。

ContainerArray

ContainerArray オブジェクトへのアクセスを提供します。

シグニチャー

```
ContainerArray * ContainerArray()
```

戻り値のデータ型
ContainerArray*

ContainerArray オブジェクトへのポインター。

CurrentDateAndTime

現在の日時で初期化された新しい DateAndTime オブジェクトを作成します。

シグニチャー

```
DateAndTime * CurrentDateAndTime()
```

戻り値のデータ型
DateAndTime*

DateAndTime オブジェクトへのポインター。

DateAndTime

新しい (初期化していない) DateAndTime オブジェクトを作成します。

シグニチャー

```
DateAndTime * DateAndTime()
```

戻り値のデータ型
DateAndTime*

DateAndTime オブジェクトへのポインター。

Disconnect

このメソッドは、特定のスレッドに関する MQSeries Workflow API 処理の初期化を解除します。

シグニチャー

```
long Disconnect()
```

戻り値のデータ型
long 処理が正常に完了すると、FMC_OK が戻されます。

ExecutionServiceArray

ExecutionServiceArray オブジェクトへのアクセスを提供します。

シグニチャー

```
ExecutionServiceArray * ExecutionServiceArray()
```

戻り値のデータ型

ExecutionServiceArray*

ExecutionServiceArray オブジェクトへのポインター。

NewActivityInstanceNotification

空のアクティビティー・インスタンス通知・オブジェクトを作成します。

シグニチャー

```
ActivityInstanceNotification * NewActivityInstanceNotification()
```

戻り値のデータ型

ActivityInstanceNotification*

作成されたアクティビティー・インスタンス通知へのポインター。

NewProcessInstance

空のプロセス・インスタンス・オブジェクトを作成します。

シグニチャー

```
ProcessInstance * NewProcessInstance()
```

戻り値のデータ型

ProcessInstance*

作成されたプロセス・インスタンスへのポインター。

NewProcessInstanceNotification

空のプロセス・インスタンス通知・オブジェクトを作成します。

シグニチャー

```
ProcessInstanceNotification * NewProcessInstanceNotification()
```

戻り値のデータ型

ProcessInstanceNotification*

作成されたプロセス・インスタンス通知へのポインター。

NewProcessTemplate

空のプロセス・テンプレート・オブジェクトを作成します。

シグニチャー

```
ProcessTemplate * NewProcessTemplate()
```

戻り値のデータ型

ProcessTemplate*

作成されたプロセス・テンプレートへのポインター。

NewWorkitem

空の作業項目・オブジェクトを作成します。

シグニチャー

```
Workitem * NewWorkitem()
```

戻り値のデータ型

Workitem* 作成された作業項目へのポインター。

ProgramID

プログラム実行エージェントに認識されているアクティビティー・インプリメンテーションのプログラム ID を戻します。

シグニチャー

```
BSTR ProgramID()
```

戻り値のデータ型

BSTR プログラム ID。

RemoteUserID

元のアクティビティー・インプリメンテーションがプログラム実行エージェントによって開始された時の対象ユーザー ID を戻します。

シグニチャー

```
BSTR RemoteUserID()
```

戻り値のデータ型

BSTR ユーザー ID。

SetConfigurationID

このメソッドは、プロファイル・アクセスに使用する構成 ID を設定します。

シグニチャー

```
long SetConfigurationID( BSTR configID )
```

パラメーター

configID 入力。設定する構成 ID。すでに定義されている構成でなければなりません。

戻り値のデータ型

long 処理が正常に完了すると、FMC_OK が戻されます。

StringArray

StringArray オブジェクトへのアクセスを提供します。

シグニチャー

```
StringArray * StringArray()
```

戻り値のデータ型

StringArray* StringArray オブジェクトへのポインター。

UserID

アクティビティー・インプリメンテーションがプログラム実行エージェントによって開始された時の対象ユーザー ID を戻します。

シグニチャー

```
BSTR UserID()
```

戻り値のデータ型

BSTR ユーザー ID。

第61章 ContainerCtrl

プロパティー

Container コントロールには **Visible** というプロパティーがあり、これは変更可能です。

メソッド

Container コントロールは以下に示すメソッドをサポートします。

Container

このメソッドは、MQSeries Workflow Container オブジェクトへのアクセスを提供します。

シグニチャー

```
Container * Container()
```

戻り値のデータ型

Container* Container オブジェクトへのポインター。

ProgramID

プログラム実行エージェントに認識されているアクティビティー・インプリメンテーションのプログラム ID を戻します。

シグニチャー

```
BSTR ProgramID()
```

戻り値のデータ型

BSTR プログラム ID。

RemoteUserID

元のアクティビティー・インプリメンテーションがプログラム実行エージェントによって開始された時の対象ユーザー ID を戻します。

シグニチャー

```
BSTR RemoteUserID()
```

戻り値のデータ型

BSTR ユーザー ID。

UserID

アクティビティー・インプリメンテーションがプログラム実行エージェントによって開始された時の対象ユーザー ID を戻します。

シグニチャー

```
BSTR UserID()
```

戻り値のデータ型

BSTR ユーザー ID。

イベント

Container コントロールは、以下に示すイベントを生成します。

Error

Visual Basic コードが実行されていない時点で発生したエラーの結果としてのみ発生します。

シグニチャー

```
void Error( short    number,
            BSTR *   description,
            SCODE    scode,
            BSTR     source,
            BSTR     helpFile,
            long     helpContext,
            boolean *cancel     )
```

パラメーター

number 出力。エラー番号 (整数)。

description 出力。エラーの説明。

scode 出力。 scode エラー (長整数)。

source 出力。エラーの発生源。

helpFile 出力。ヘルプ・ファイルの名前。

helpContext 出力。ヘルプ・コンテキスト識別子。

cancel 入力。 *True* に設定すると、メッセージは次の層に送られません。

第62章 すべての GUI コントロールでサポートされるメソッド

以下に示すメソッドは、ExecutionService コントロール、ProcessTemplateList コントロール、ProcessInstanceList コントロール、Worklist コントロール、および Monitor コントロールによってサポートされます。

AboutBox

コントロールの情報ボックスを開きます。

シグニチャー

```
void AboutBox()
```

ReadUserSettings

レジストリーからユーザー設定値を復元します。

レポート・ビューに表示される列の値が復元されます。

シグニチャー

```
boolean ReadUserSettings( BSTR name )
```

パラメーター

name 入力。レジストリー・キーの名前を示す文字列。その値が復元されます。

戻り値のデータ型

boolean 処理が正常に完了すると、*True* が戻されます。

RemoveGUI

ExecutionService コントロールと MQWorkflow コントロールの間の接続を除去します。

シグニチャー

```
long RemoveGUI()
```

戻り値のデータ型

long 処理が正常に完了すると、FMC_OK が戻されます。

SetHelpFile

実行機能モードで使用するヘルプ・ファイルのパスとファイル名を定義します。

シグニチャー

```
void SetHelpFile( BSTR name )
```

パラメーター

name 入力。ヘルプ・ファイルの完全修飾パス名の文字列。

ShowContextMenu

コンテキスト・メニューを使用可能または使用不可にします。

シグニチャー

```
void ShowContextMenu( boolean toggle )
```

パラメーター

toggle 入力。 *True* を指定した場合、マウスの右ボタンをクリックすると、コンテキスト・メニューが表示されます。 *False* を指定すると、コンテキスト・メニューは使用不可になります。

WriteUserSettings

レジストリー内にユーザー設定値を保管します。

レポート・ビューに表示するよう選択した列と、現行の列サイズが保管されます。

シグニチャー

```
boolean WriteUserSettings( BSTR name )
```

パラメーター

name 入力。現行値を保管するレジストリー・キーの名前を示す文字列。

戻り値のデータ型

boolean 処理が正常に完了すると、*True* が戻されます。

第63章 すべてのリスト・コントロールでサポートされるメソッド

以下に示すメソッドは、リスト・コントロール、ProcessTemplateList コントロール、ProcessInstanceList コントロール、および Worklist コントロールによってサポートされます。

ConnectGUI

指定したプロセス・テンプレート・リスト、指定したプロセス・インスタンス・リスト、または指定したワークリストを GUI に接続します。

シグニチャー

```
long ConnectGUI( IDispatch * list, IDispatch * es, IDispatch * wfc )
```

パラメーター

es 入力。実行サービス・オブジェクトへのポインター。

list 入力。リスト・オブジェクトへのポインター。

wfc 入力。 MQWorkflowCtrl オブジェクトへのポインター。

戻り値のデータ型

long 処理が正常に完了すると、FMC_OK が戻されます。

ContextMenuDelete

コンテキスト・メニュー項目「Delete (削除)」を呼び出します。

シグニチャー

```
long ContextMenuDelete()
```

戻り値のデータ型

long *FmcjProcessTemplate::Delete()*、*FmcjProcessInstanceListDelete()*、*FmcjWorklist::Delete()* の戻りコード。

ContextMenuListProperties

コンテキスト・メニュー項目「Properties (プロパティー)」を呼び出します。

このメソッドは、プロセス・テンプレート・リスト、プロセス・インスタンス・リスト、またはワークリストのプロパティーに関する情報を示すダイアログを表示します。

シグニチャー

```
void ContextMenuListProperties()
```

ContextMenuListSettings

コンテキスト・メニュー項目「Processlist settings (プロセス・リストの設定値)」を呼び出します。

このメソッドは、プロセス・テンプレート・リスト、プロセス・インスタンス・リスト、またはワークリストの名前、タイプ、フィルター、およびソート基準に関する情報を示すダイアログを表示します。

シグニチャー

```
void ContextMenuListSettings ()
```

ContextMenuListRefresh

コンテキスト・メニュー項目「Refresh ProcessTemplateList (ProcessTemplateList のリフレッシュ)」、「Refresh ProcessInstanceList (ProcessInstanceList のリフレッシュ)」、または「Refresh Worklist (Worklist のリフレッシュ)」を呼び出します。

シグニチャー

```
long ContextMenuListRefresh()
```

戻り値のデータ型

long 処理が正常に完了すると、FMC_OK が戻されます。

ContextMenuProperties

コンテキスト・メニュー項目「Show Properties (プロパティーの表示)」を呼び出します。

このメソッドは、ProcessTemplateList、ProcessInstanceList、または Worklist コントロールから選択したオブジェクトのプロパティーを表示します。

シグニチャー

```
void ContextMenuProperties()
```

ContextMenuViewIcon

ProcessTemplateList、ProcessInstanceList、または Worklist コントロールのグリッドをアイコン・モードで表示します。

シグニチャー

```
void ContextMenuViewIcon()
```

ContextMenuViewList

ProcessTemplateList、ProcessInstanceList、または Worklist コントロールのグリッドをリスト・モードで表示します。

シグニチャー

```
void ContextMenuViewList()
```

ContextMenuViewReport

ProcessTemplateList、ProcessInstanceList、または Worklist コントロールのグリッドをレポート・モードで表示します。

シグニチャー

```
void ContextMenuViewReport()
```

ContextMenuViewSmallIcon

ProcessTemplateList、ProcessInstanceList、または Worklist コントロールのグリッドを小アイコン・モードで表示します。

シグニチャー

```
void ContextMenuViewSmallIcon()
```

FindFirst

プロセス・テンプレート・リスト、プロセス・インスタンス・リスト、またはワークリスト配列の最初のオブジェクトを戻します。

プロセス・テンプレートまたはプロセス・インスタンス・リストのシグニチャー

```
IDispatch * FindFirst( long * index )
```

ワークリストのシグニチャー

```
IDispatch * FindFirst( long * objecttype, long * index )
```

パラメーター

index 出力。戻されるインデックスを保管するための long フィールドへのポインター。選択されたオブジェクトが見つからない場合は、1 が戻されます。

objecttype 出力。実際の作業項目、アクティビティー・インスタンス通知、またはプロセス・インスタンス通知が戻されたかどうかの指示。

戻り値のデータ型

IDispatch* オブジェクト・ポインター。選択オブジェクトが見つからない場合は NULL (0) が戻されます。

FindNext

プロセス・テンプレート・リスト、プロセス・インスタンス・リスト、またはワークリスト配列の次のオブジェクトを戻します。あらかじめ FindFirst() を呼び出しておく必要があります。

プロセス・テンプレートまたはプロセス・インスタンス・リストのシグニチャー

```
IDispatch * FindNext( long * index  )
```

ワークリストのシグニチャー

```
IDispatch * FindNext( long * objecttype, long * index )
```

パラメーター

index 出力。戻されるインデックスを保管するための long フィールドへのポインター。選択されたオブジェクトが見つからない場合は、1 が戻されます。

objecttype 出力。実際の作業項目、アクティビティー・インスタンス通知、またはプロセス・インスタンス通知が戻されたかどうかの指示。

戻り値のデータ型

IDispatch* オブジェクト・ポインター。選択オブジェクトが見つからない場合は NULL (0) が戻されます。

戻り値

IDispatch* ProcessTemplate オブジェクトのポインター。選択されているオブジェクトが見つからないなら、0 (ヌル) が戻されます。

パラメーター

index 出力。戻されるインデックスを保管するための long フィールドへのポインター。選択されたオブジェクトが見つからない場合は、1 が戻されます。

GetItemAt

プロセス・テンプレート・リスト、プロセス・インスタンス・リスト、またはワークリストの指定されたインデックス位置にあるオブジェクトを戻します。

シグニチャー

```
IDispatch * GetItemAt( long index )
```

パラメーター

index 入力。対象となるオブジェクトの配列インデックス。

戻り値のデータ型

IDispatch* オブジェクト・ポインター。

GetItemCount

プロセス・テンプレート・リスト配列、プロセス・インスタンス・リスト配列、またはワークリスト配列の中のオブジェクトの数を戻します。

シグニチャー

```
long GetItemCount()
```

戻り値のデータ型

long 配列内の要素の数を戻します。

第64章 すべての GUI コントロールによって生成されるイベント

以下に示すイベントは、ExecutionService コントロール、ProcessTemplateList コントロール、ProcessInstanceList コントロール、Worklist コントロール、および Monitor コントロールによって生成されます。

Click

このコントロールでユーザーがマウス・ボタンを押すと発生します。

シグニチャー

```
void Click()
```

DblClick

このコントロールでユーザーがマウス・ボタンを押して放した後、もう一度押して放すと発生します。

シグニチャー

```
void DblClick()
```

KeyPress

ANSI キーを押して放すと発生します。

シグニチャー

```
void KeyPress( short  keyAscii )
```

パラメーター

keyAscii 出力。標準的な ANSI 数値キー・コードを戻す整数。

第65章 すべての非モニター GUI コントロールによって生成されるイベント

以下に示すイベントは、ExecutionService コントロール、ProcessTemplateList コントロール、ProcessInstanceList コントロール、および Worklist コントロールによって生成されます。

Error

Visual Basic コードが実行されていない時点で発生したエラーの結果としてのみ発生します。

シグニチャー

```
void Error( short    number,
            BSTR *   description,
            SCODE    scode,
            BSTR     source,
            BSTR     helpFile,
            long     helpContext,
            boolean  cancel     )
```

パラメーター

number 出力。エラー番号 (整数)。

description 出力。エラーの説明。

scode 出力。 scode エラー (長整数)。

source 出力。エラーの発生源。

helpFile 出力。ヘルプ・ファイルの名前。

helpContext 出力。ヘルプ・コンテキスト識別子。

cancel 入力。 *True* に設定すると、メッセージは次の層に送られません。

KeyDown

オブジェクトのフォーカスがこのコントロールにある場合にキーを押すと発生します。

シグニチャー

```
void KeyDown( short * keyCode, short shift )
```

パラメーター

keyCode 出力。キーのコード。 vbKeyF1 (F1 キー) や vbKeyHome (HOME キー) など。

shift 出力。イベント発生時の SHIFT キー、CTRL キー、および ALT キーの状態に対応する整数。

KeyUp

オブジェクトのフォーカスがこのコントロールにある場合にキーを放すと発生します。

シグニチャー

```
void KeyUp( short * keyCode, short shift )
```

パラメーター

keyCode 出力。キーのコード。 vbKeyF1 (F1 キー) や vbKeyHome (HOME キー) など。

shift 出力。イベント発生時の SHIFT キー、CTRL キー、および ALT キーの状態に対応する整数。

MouseDown

ProcessInstanceList コントロール・ウィンドウでユーザーがマウス・ボタンを押すと発生します。

シグニチャー

```
void MouseDown( short            button ,
                short            shift ,
                OLE_XPOS_PIXELS  x ,
                OLE_YPOS_PIXELS  y     )
```

パラメーター

button 出力。押されたボタンを識別する整数を戻します。

shift 出力。イベント発生時の SHIFT キー、CTRL キー、および ALT キーの状態に対応する整数。

x 出力。マウス・ポインターの現在位置 (横座標) を指定する整数を戻します。

y 出力。マウス・ポインターの現在位置 (縦座標) を指定する整数を戻します。

MouseMove

ユーザーがマウスを移動すると発生します。

シグニチャー

```
void MouseMove( short           button ,
                short           shift ,
                OLE_XPOS_PIXELS x ,
                OLE_YPOS_PIXELS y     )
```

パラメーター

button 出力。押されたボタンを識別する整数を戻します。

shift 出力。イベント発生時の SHIFT キー、CTRL キー、および ALT キーの状態に対応する整数。

x 出力。マウス・ポインターの現在位置 (横座標) を指定する整数を戻します。

y 出力。マウス・ポインターの現在位置 (縦座標) を指定する整数を戻します。

MouseUp

ProcessInstanceList コントロール・ウィンドウでユーザーがマウス・ボタンを放すと発生します。

シグニチャー

```
void MouseUp(   short           button ,
                short           shift ,
                OLE_XPOS_PIXELS x ,
                OLE_YPOS_PIXELS y     )
```

パラメーター

button	出力。押されたボタンを識別する整数を戻します。
shift	出力。イベント発生時の SHIFT キー、CTRL キー、および ALT キーの状態に対応する整数。
x	出力。マウス・ポインターの現在位置 (横座標) を指定する整数を戻します。
y	出力。マウス・ポインターの現在位置 (縦座標) を指定する整数を戻します。

第66章 すべてのリスト・コントロールによって生成されるイベント

以下に示すイベントは、ProcessTemplateList コントロール、ProcessInstanceList コントロール、および Worklist コントロールによって生成されます。

ViewChanged

リストのグリッド・ビューが変更された時点で発生します。

シグニチャー

```
void ViewChanged()
```

第67章 ExecutionServiceCtrl

プロパティー

デザイン・モードで ExecutionService コントロールのプロパティーを表示するには、そのコントロールのグリッド域にマウス・ポインターを移動して、マウスの右ボタンをクリックし、**「Properties (プロパティー)」**を選択します。

プロパティーのダイアログには、**「Fonts (フォント)」**と**「Pictures (ピクチャー)」**という 2 つのタブがあります。

タブは、すべて設計モードでも実行モードでも使用できます。

「Fonts (フォント)」タブには、標準的なフォント・ダイアログが表示されます。そこでは、ExecutionService コントロールに表示するテキストのフォント、フォント・サイズ、スタイル、および文字飾りを操作できます。

「Pictures (ピクチャー)」タブには、ExecutionService コントロールの表示画面のさまざまな要素にアイコンを関連付ける機能が用意されています。

特定のアイテムに現在関連付けられているアイコンを表示するには、**「Property Name (プロパティー名)」**ドロップダウン・リストからそのアイテムを選択します。関連付けられているアイコンが**「Preview (プレビュー)」**域に表示されます。

現在選択されているアイテムのアイコンを変更するには、**「Browse (参照)」**ボタンをクリックします。標準的な参照ダイアログが表示されます。指定したアイテムに現在関連付けられているアイコンは、ブラウザーによって選択されたアイコンによって置き換えられます。**「Clear (クリア)」**ボタンをクリックすると、指定したアイテムに現在関連付けられているアイコンが除去され、アイテムはアイコンのない状態になります。

プロパティー **Appearance**、**BorderStyle**、**Font**、**IconMQWorkflow**、**IconOneProcessInstanceList**、**IconOneProcessTemplateList**、**IconOneWorklist**、**IconProcessInstanceLists**、**IconProcessTemplateLists**、**IconSystem**、**IconWorklists** は変更可能です。

メソッド

ExecutionService コントロールは、すべてのコントロールによってサポートされるメソッドのほかに、以下のメソッドをサポートします。 753ページの『第62章 すべての GUI コントロールでサポートされるメソッド』を参照。

ConnectGUI

ExecutionServiceCtrl オブジェクトと MQWorkflowCtrl オブジェクトを接続します。

シグニチャー

```
long ConnectGUI( IDispatch * wfc )
```

パラメーター

wfc 入力。 MQWorkflowCtrl オブジェクトへのポインター。

戻り値のデータ型

long 処理が正常に完了すると、FMC_OK が戻されます。

ContextMenuDeleteProcInstList

コンテキスト・メニュー項目「Delete Process Instance List (プロセス・インスタンス・リストの削除)」を呼び出します。

シグニチャー

```
long ContextMenuDeleteProcInstList( long index1, long index2 )
```

パラメーター

index1 入力。 ExecutionServiceArray 内のインデックス。

index2 入力。 ProcessInstanceListArray 内のインデックス。

戻り値のデータ型

long *FmcjProcessInstanceList::Delete()* の戻りコード。

ContextMenuDeleteProcTempList

コンテキスト・メニュー項目「Delete Process Template List (プロセス・テンプレート・リストの削除)」を呼び出します。

シグニチャー

```
long ContextMenuDeleteProcTempList( long index1, long index2 )
```

パラメーター

index1 入力。 ExecutionServiceArray 内のインデックス。
index2 入力。 ProcessTemplateListArray 内のインデックス。

戻り値のデータ型

long *FmcjProcessTemplateList::Delete()* の戻りコード。

ContextMenuDeleteWorklist

コンテキスト・メニュー項目「Delete Worklist (ワークリストの削除)」を呼び出します。

シグニチャー

```
long ContextMenuDeleteWorklist( long index1, long index2 )
```

パラメーター

index1 入力。 ExecutionServiceArray 内のインデックス。
index2 入力。 WorklistListArray 内のインデックス。

戻り値のデータ型

long *FmcjWorklistList::Delete()* の戻りコード。

ContextMenuLogoff

コンテキスト・メニュー項目「Logoff (ログオフ)」を呼び出します。

シグニチャー

```
long ContextMenuLogoff( long index )
```

パラメーター

index 入力。 ExecutionServiceArray 内のインデックス。

戻り値のデータ型

long *FmcjExecutionService::Logoff()* の戻りコード。

ContextMenuLogon

コンテキスト・メニュー項目「Logon (ログオン)」を呼び出します。

シグニチャー

```
long ContextMenuLogon( long  index,
                       BSTR  userID,
                       BSTR  password )
```

パラメーター

index 入力。 ExecutionServiceArray 内のインデックス。

userID 入力。ログオンするのに使うユーザー ID。

password 入力。パスワード。

戻り値のデータ型

long *FmcjExecutionService::Logon()* の戻りコード。

ContextMenuLogonDialog

コンテキスト・メニュー項目「Logon (ログオン)」を呼び出します。このメソッドを呼び出すと、別個のログオン・ダイアログが表示されます。そこでユーザーは、ユーザー ID やパスワードなどの詳細なログオン・パラメーターを指定することになります。

シグニチャー

```
long ContextMenuLogonDialog()
```

戻り値のデータ型

long *FmcjExecutionService::Logon()* の戻りコード。

ContextMenuNewProcInstList

コンテキスト・メニュー項目「Create New Process Instance List (プロセス・インスタンス・リストの新規作成)」を呼び出します。

ダイアログが表示され、そこでユーザーは詳細なリスト作成パラメーターを指定することになります。

シグニチャー

```
long ContextMenuNewProcInstList( long index )
```

パラメーター

index 入力。 ExecutionServiceArray 内のインデックス。

戻り値のデータ型

long *FmcjExecutionService::CreateProcessInstanceList()* の戻りコード。

ContextMenuNewProcTempList

コンテキスト・メニュー項目「Create New Process Template List (プロセス・テンプレート・リストの新規作成)」を呼び出します。

ダイアログが表示され、そこでユーザーは詳細なリスト作成パラメーターを指定することになります。

シグニチャー

```
long ContextMenuNewProcTempList( long index  )
```

パラメーター

index 入力。 ExecutionServiceArray 内のインデックス。

戻り値のデータ型

long *FmcjExecutionService::CreateProcessTemplateList()* の戻りコード。

ContextMenuNewWorklist

コンテキスト・メニュー項目「Create New Work List (ワークリストの新規作成)」を呼び出します。

ダイアログが表示され、そこでユーザーは詳細なリスト作成パラメーターを指定することになります。

シグニチャー

```
long ContextMenuNewWorklist( long index )
```

パラメーター

index 入力。 ExecutionServiceArray 内のインデックス。

戻り値のデータ型

long Return code of *FmcjExecutionService::CreateWorklist()* の戻りコード。

ContextMenuProperties

コンテキスト・メニュー項目「Properties (プロパティー)」を呼び出します。

シグニチャー

```
void ContextMenuProperties()
```

ContextMenuRefresh

コンテキスト・メニュー項目「Refresh (リフレッシュ)」を呼び出します。

シグニチャー

```
void ContextMenuRefresh()
```

ContextMenuRefreshProcInstLists

コンテキスト・メニュー項目「Refresh Process Instance Lists (プロセス・インスタンス・リストのリフレッシュ)」を呼び出します。

シグニチャー

```
void ContextMenuRefreshProcInstLists( long index )
```

パラメーター

index 入力。 ExecutionServiceArray 内のインデックス。

ContextMenuRefreshProcInstances

コンテキスト・メニュー項目「Refresh Process Instances (プロセス・インスタンスのリフレッシュ)」を呼び出します。

シグニチャー

```
void ContextMenuRefreshProcInstances( long index )
```

パラメーター

index 入力。 ExecutionServiceArray 内のインデックス。

ContextMenuRefreshProcTempLists

コンテキスト・メニュー項目「Refresh Process Template Lists (プロセス・テンプレート・リストのリフレッシュ)」を呼び出します。

シグニチャー

```
void ContextMenuRefreshProcTempLists( long index )
```

パラメーター

index 入力。 ExecutionServiceArray 内のインデックス。

ContextMenuRefreshProcTemplates

コンテキスト・メニュー項目「Refresh Process Templates (プロセス・テンプレートのリフレッシュ)」を呼び出します。

シグニチャー

```
void ContextMenuRefreshProcTemplates( long index )
```

パラメーター

index 入力。 ExecutionServiceArray 内のインデックス。

ContextMenuRefreshWorkitems

コンテキスト・メニュー項目「Refresh Work Items (作業項目のリフレッシュ)」を呼び出します。

シグニチャー

```
void ContextMenuRefreshWorkitems( long index )
```

パラメーター

index 入力。 ExecutionServiceArray 内のインデックス。

ContextMenuRefreshWorklists

コンテキスト・メニュー項目「Refresh Worklists (ワークリストのリフレッシュ)」を呼び出します。

シグニチャー

```
void ContextMenuRefreshWorklists( long index )
```

パラメーター

index 入力。 ExecutionServiceArray 内のインデックス。

ContextMenuUserInformation

コンテキスト・メニュー項目「User Information (ユーザー情報)」を呼び出します。

シグニチャー

```
void ContextMenuUserInformation( long index )
```

パラメーター

index 入力。 ExecutionServiceArray 内のインデックス。

イベント

ExecutionService コントロールは、すべての非モニター・コントロールによって生成されるイベントのほかに、以下のイベントを生成します。 763ページの『第64章 すべての GUI コントロールによって生成されるイベント』、および765ページの『第65章 すべての非モニター GUI コントロールによって生成されるイベント』を参照。

ItemCollapsed

アイテムが縮小されると発生します。

シグニチャー

```
void ItemCollapsed( BSTR name, long type, long index )
```

パラメーター

name 出力。アイテム名。

type 出力。アイテムのタイプ。

index 出力。 ExecutionServiceArray 内の対応する ExecutionService のインデックス。

ItemCollapsing

アイテムの縮小中に発生します。

シグニチャー

```
void ItemCollapsing ( BSTR      name,
                      long      type,
                      long      index,
                      boolean * cancel )
```

パラメーター

name 出力。アイテム名。

type 出力。アイテムのタイプ。

index 出力。 ExecutionServiceArray 内の対応する ExecutionService のインデックス。

cancel 入出力。ブール変数を指すポインター。 *True* が戻されると、そのアイテムは縮小されていません。

ItemExpanded

アイテムが拡張されると発生します。

シグニチャー

```
void ItemExpanded( BSTR name, long type, long index )
```

パラメーター

name 出力。アイテム名。
type 出力。アイテムのタイプ。
index 出力。 ExecutionServiceArray 内の対応する ExecutionService のインデックス。

ItemExpanding

アイテムの拡張中に発生します。

シグニチャー

```
void ItemExpanding( BSTR      name,
                    long      type,
                    long      index,
                    boolean * cancel )
```

パラメーター

name 出力。アイテム名。
type 出力。アイテムのタイプ。
index 出力。 ExecutionServiceArray 内の対応する ExecutionService のインデックス。
cancel 入出力。ブール変数を指すポインター。 *True* が戻されると、そのアイテムは縮小されていません。

SelChanged

選択が変更された場合に発生します。

シグニチャー

```
void SelChanged( BSTR name, long type, long index )
```

パラメーター

name	出力。新たに選択されたアイテムのアイテム名。
type	出力。新たに選択されたアイテムのタイプ。
index	出力。 ExecutionServiceArray 内の対応する ExecutionService のインデックス。

SelChanging

アイテムの変更中に発生します。

シグニチャー

```
void SelChanging( BSTR name, long type, long index, boolean * cancel )
```

パラメーター

name	出力。新たに選択されたアイテムのアイテム名。
type	出力。新たに選択されたアイテムのタイプ。
index	出力。 ExecutionServiceArray 内の対応する ExecutionService のインデックス。
cancel	入力。ブール変数を指すポインター。 *True* に設定されているなら、アイテムは変更されていません。

第68章 ProcessTemplateListCtrl

GUI 処理を実行するには、プロセス・テンプレート・リストを ProcessTemplateList コントロールに接続する必要があります。この接続は、ProcessTemplateListCtrl クラスの **ConnectGUI()** メソッドによって確立します。この接続が確立されているなら、ユーザーは ProcessTemplateList コントロール中の ProcessTemplateList オブジェクトに対する完全なアクセスが得られます。

プロパティー

デザイン・モードで ProcessTemplateList コントロールのプロパティーを表示して調整するには、そのコントロールのグリッド域内にマウス・ポインターを移動して、マウスの右ボタンをクリックし、**「Properties (プロパティー)」**を選択します。

プロパティーのダイアログには、**「Fonts (フォント)」**、**「Colors (色)」**、**「Column Selection (列の選択)」**、および**「Column Attribute (列の属性)」**という 4 つのタブがあります。

タブは、すべて設計モードでも実行モードでも使用できます。

このダイアログから、さまざまな方法でコントロールの表示内容を構成できます。**「View (ビュー)」**には、**「Report (レポート)」**、**「List (リスト)」**、**「Small Icon (小アイコン)」**、および**「Icon (アイコン)」**という 4 つのオプションがあります。

「Fonts (フォント)」タブには、標準的なフォント・ダイアログが表示されます。 このダイアログでは、コントロールのウィンドウに表示するテキストのフォント、フォント・サイズ、スタイル、および文字飾りを操作できます。

「Colors (色)」タブには、コントロールのウィンドウのカラー・スキームを操作する機能が用意されています。

コントロールの前景と背景の色を設定したり更新したりできます。**「ForeColor (前景)」**または**「BackColor (背景)」**を選択し、16 色のボタンのうち 1 つをクリックしてください。その後、**「Apply (適用)」**ボタンをクリックしてください。

「Column Selection (列の選択)」タブを使用すると、列や情報をコントロールの表示内容に追加したり表示内容から除去したりできます。列を追加するには、右側のペインにあるアイテムを強調表示させてから、**「Add (追加)」**ボタンをクリックします。列を削除するには、左側のペインにあるアイテムを強調表示させてから、**「Delete (削除)」**ボタンをクリックします。

「Column Attribute (列の属性)」タブには、コントロールによってそのアイテムが表示される際の特定の局面を操作するメカニズムが用意されています。それらは、次のとおりです。

1. **「Icon view (アイコン・ビュー)」**チェック・ボックスは、アイコンの表示スタイルが有効な場合に、含められる情報の列 (名前が左側のリスト・ボックスに表示されている) を指定するのに使います。ある時点で選択されている列は、右側のアイコンの下に表示されます。
2. **「Column width (列の幅)」**は、列のアイテムに使用されるピクセル数を指定するのに使います。これは**レポート**形式の表示内容に限り有効です。列の幅を調整するには、左側のリスト・ボックスのアイテムを強調表示してから、**「Column width (列の幅)」**入力ボックスに数値を入力して、**「Apply (適用)」**ボタンをクリックします。
3. **「Alignment (調整)」**は、列のテキストを左または右にそろえる場合に使います。これは**レポート**形式の表示内容に限り有効です。

実行機能モードで、マウス・ポインターを特定のアイテム上に置いてマウスの右ボタンをクリックすると、そのアイテムのコンテキスト・メニューが表示されます。**「Properties (プロパティー)」**オプションを選択すると、タブ形式のダイアログが表示されます。

プロパティーのダイアログには、**「General (一般)」**、**「Data and Staff (データおよびスタッフ)」**、**「History (履歴)」**、および**「Documentation (ドキュメンテーション)」**という 4 つのタブがあります。**「General (一般)」**タブには、プロセス・テンプレート・リストに属する情報の以下の部分が表示されます。

「Data and Staff (データおよびスタッフ)」タブには、リスト・アイテム・オブジェクトの管理に関する情報が表示されます。それらは、次のとおりです。

- プロセス・インスタンスの入出力コンテナー名。
- プロセス・インスタンスのプロセス管理者。
- プロセス・インスタンスを開始したユーザー。
- プロセス・インスタンスに関してユーザーに割り当てられているアクティビティーの役割と組織の基準。

「History (履歴)」タブには、テンプレート情報が表示されます。それらは、次のとおりです。

- TimeStamps:

  プロセス・テンプレートが作成された日付と時刻、開始された日付と時刻、および発行日の日付と時刻。

「Documentation (文書)」タブには、リスト・アイテム・オブジェクトの注釈として追加されたものが表示されます。

注: クライアントにおいて、または ProcessTemplateList コントロールを使用して作成したアプリケーションのうち、メニュー・バーを使用するものにおいては、メニュー・バーの**「Process (プロセス)」**オプションによって、プロセス・アイテムのプロパティー・ダイアログ (および他のすべてのコンテキスト・メニュー項目) にアクセスすることもできます。

ProcessTemplateList の設定値

プロセス・テンプレート・リストの設定値を表示して調整するには、そのコントロールのウィンドウ域の未使用部分にマウス・ポインターを移動して、マウスの右ボタンをクリックし、**「ProcessTemplateList settings (ProcessTemplateList の設定値)」**を選択します。

プロパティー **Appearance**、**Arrange**、**BackColor**、**BorderStyle**、**CtrlColumnOrder**、**CtrlColumnWidth**、**Font**、**ForeColor**、**View** は変更可能です。

メソッド

ProcessTemplateList コントロールは、すべてのリスト・コントロールによってサポートされるメソッドのほかに、以下のメソッドをサポートします。 753ページの『第62章 すべての GUI コントロールでサポートされるメソッド』、および 757ページの『第63章 すべてのリスト・コントロールでサポートされるメソッド』を参照。

ContextMenuCreateInstance

コンテキスト・メニュー項目「Create Instance (インスタンスの作成)」を呼び出します。

シグニチャー

```
long ContextMenuCreateInstance()
```

戻り値のデータ型

long Return code of *FmcjProcessTemplate::CreateInstance()* の戻りコード。

RefreshProcessTemplateList

ProcessTemplateList コントロールをリフレッシュします。

シグニチャー

```
long RefreshProcessTemplateList()
```

戻り値のデータ型

long 処理が正常に完了すると、FMC_OK が戻されます。

イベント

763ページの『第64章 すべての GUI コントロールによって生成されるイベント』、765ページの『第65章 すべての非モニター GUI コントロールによって生成されるイベント』、および 769ページの『第66章 すべてのリスト・コントロールによって生成されるイベント』を参照。

第69章 ProcessInstanceListCtrl

プロセス・インスタンス・リストにアクセスするには、ProcessInstanceList オブジェクトを ProcessInstanceList コントロールに接続する必要があります。この接続は、ProcessInstanceListCtrl クラスから関数 ConnectGUI() によって確立されます。この接続が確立されているなら、ユーザーは ProcessInstanceList コントロール中の ProcessInstanceList オブジェクトに対する完全なアクセスが得られます。

プロパティー

デザイン・モードで ProcessInstanceList コントロールのプロパティーを表示して調整するには、そのコントロールのグリッド域内にマウス・ポインターを移動して、マウスの右ボタンをクリックし、**「Properties (プロパティー)」**を選択します。

プロパティーのダイアログには、**「Fonts (フォント)」**、**「Colors (色)」**、**「Column Selection (列の選択)」**、および**「Column Attribute (列の属性)」**という 4 つのタブがあります。

タブは、すべて設計モードでも実行モードでも使用できます。

このダイアログから、さまざまな方法でコントロールの表示内容を構成できます。**「View (ビュー)」**には、**「Report (レポート)」**、**「List (リスト)」**、**「Small Icon (小アイコン)」**、および**「Icon (アイコン)」**という 4 つのオプションがあります。

「Fonts (フォント)」タブには、標準的なフォント・ダイアログが表示されます。 そこでは、コントロールのウィンドウに表示するテキストのフォント、フォント・サイズ、スタイル、および文字飾りを操作できます。

「Colors (色)」タブには、コントロールのウィンドウのカラー・スキームを操作する機能が用意されています。

コントロールの前景と背景の色を設定したり更新したりできます。
「ForeColor (前景)」または**「BackColor (背景)」**を選択し、16 色のボタンのうち 1 つをクリックしてから、**「Apply (適用)」**ボタンをクリックしてください。

「Column Selection (列の選択)」タブを使用すると、列や情報をコントロールの表示内容に追加したり表示内容から除去したりできます。列を追加するには、右側のペインにあるアイテムを強調表示させてから、**「Add (追加)」**ボタンをクリックします。列を削除するには、左側のペインにあるアイテムを強調表示させてから、**「Delete (削除)」**ボタンをクリックします。

「Column Attribute (列の属性)」タブには、コントロールによってそのアイテムが表示される際の特定の局面を操作するメカニズムが用意されています。それらは、次のとおりです。

1. **「Icon view (アイコン・ビュー)」**チェック・ボックスは、アイコンの表示スタイルが有効な場合に、含められる情報の列 (名前が左側のリスト・ボックスに表示されている) を指定するのに使います。ある時点で選択されている列は、右側のアイコンの下に表示されます。
2. **「Column width (列の幅)」**は、列のアイテムに使用されるピクセル数を指定するのに使います。これは**レポート**形式の表示内容に限り有効です。列の幅を調整するには、左側のリスト・ボックスのアイテムを強調表示してから、**「Column width (列の属性)」**入力ボックスに数値を入力して、**「Apply (適用)」**ボタンをクリックします。
3. **「Alignment (調整)」**は、列のテキストを左または右にそろえる場合に使います。これは**レポート**形式の表示内容に限り有効です。

実行機能モードで、マウス・ポインターを特定のアイテム上に置いてマウスの右ボタンをクリックすると、そのアイテムのコンテキスト・メニューが表示されます。**「Properties (プロパティー)」**オプションを選択すると、タブ形式のダイアログが表示されます。

プロパティーのダイアログには、**「General (一般)」**、**「Data and Staff (データおよびスタッフ)」**、**「History (履歴)」**、および**「Documentation (ドキュメンテーション)」**という 4 つのタブがあります。**「General (一般)」**タブには、ProcessInstance に関する情報の以下の部分が表示されます。それらは、次のとおりです。

- 名前
- 記述
- プロセス・カテゴリー
- 状況
- そのプロセス・インスタンスの上位プロセス
- そのプロセス・インスタンスの最上位プロセス
- そのプロセス・インスタンスが監査されているかどうか

- エラーが発生した場合にそのプロセス・インスタンスを終了するかどうか
- そのプロセス・インスタンスの開始者に、そのプロセス・インスタンスのデータを入力するよう求めるプロンプトが出されるかどうか

「Data and Staff (データおよびスタッフ)」タブには、リスト・アイテム・オブジェクトの管理に関する情報が表示されます。それらは、次のとおりです。

- プロセス・インスタンスの入出力コンテナー名。
- プロセス・インスタンスのプロセス管理者。
- プロセス・インスタンスを開始したユーザー。
- プロセス・インスタンスに関してアクティビティーを割り当てられているユーザーの役割と組織の基準。

「History (履歴)」タブには、3 種類のアクティビティー情報が表示されます。それらは、次のとおりです。

- TimeStamps (タイムスタンプ)

  プロセス・インスタンスが作成された日付と時刻、開始された日付と時刻、および最後に変更された日付と時刻。
- Notification (通知)

  アクティビティーに関する最初の通知が送信される (または送信された) 日付と時刻。
- Finished (終了)

  アクティビティーが完了した日付と時刻。

「Documentation (文書)」タブには、リスト・アイテム・オブジェクトの注釈として追加されたものが表示されます。

注: 実行機能クライアントにおいて、または ProcessInstanceList コントロールを使用して作成したアプリケーションのうち、メニュー・バーを使用するものにおいては、メニュー・バーの**「Process (プロセス)」**オプションによって、プロセス・アイテムのプロパティー・ダイアログ (および他のすべてのコンテキスト・メニュー項目) にアクセスすることもできます。

ProcessInstanceList の設定値

プロセス・インスタンス・リストの設定値を表示して調整するには、そのコントロールのウィンドウ域の未使用部分にマウス・ポインターを移動して、マウスの右ボタンをクリックし、**「ProcessInstanceList settings (ProcessInstanceList の設定値)」**を選択します。

ProcessInstanceList 設定値のダイアログには、**「General (一般)」**、**「Filter (フィルター)」**、および**「Sort (ソート)」**という 3 つのタブが表示されます。

「General (一般)」ページには、プロセス・テンプレート・リストの名前とタイプが表示されます。このページで、プロセス・テンプレート・リストのしきい値や記述に修正を加えることもできます。その他の 2 ページは、フィルターとソートの基準を表示したり修正を加えたりするためのページです。

プロパティー **Appearance**、**Arrange**、**BackColor**、**BorderStyle**、**CtrlColumnOrder**、**CtrlColumnWidth**、**Font**、**ForeColor**、**View** は変更可能です。

メソッド

ProcessInstanceList コントロールは、すべてのリスト・コントロールによってサポートされるメソッドのほかに、以下のメソッドをサポートします。 753ページの『第62章 すべての GUI コントロールでサポートされるメソッド』、および 757ページの『第63章 すべてのリスト・コントロールでサポートされるメソッド』を参照。

ContextMenuRestart

コンテキスト・メニュー項目「Restart (再始動)」を呼び出します。

このメソッドは、コントロール内で選択されているプロセス・インスタンスのリストを再始動します。詳細については、586ページの『Restart()』を参照してください。

シグニチャー

```
long ContextMenuRestart()
```

戻り値のデータ型

long *FmcjProcessInstance::Restart()* の戻りコード。

ContextMenuResume

コンテキスト・メニュー項目「Resume (再開)」を呼び出します。

このメソッドは、コントロール内で選択されているプロセス・インスタンスのリストを再開します。詳細については、588ページの『Resume()』を参照してください。

シグニチャー

```
long ContextMenuResume()
```

戻り値のデータ型

long *FmcjProcessInstance::Resume()* の戻りコード。

ContextMenuResumeDeep

コンテキスト・メニュー項目「Resume deep (全プロセスの再開)」を呼び出します。

このメソッドは、コントロール内で選択されているプロセス・インスタンスのリストを再開します。詳細については、588ページの『Resume()』を参照してください。

シグニチャー

```
long ContextMenuResumeDeep()
```

戻り値のデータ型

long *FmcjProcessInstance::Resume()* の戻りコード。

ContextMenuStart

コンテキスト・メニュー項目「Start (開始)」を呼び出します。

このメソッドは、コントロール内で選択されているプロセス・インスタンスのリストを開始します。詳細については、596ページの『Start()』を参照してください。

シグニチャー

```
long ContextMenuStart()
```

戻り値のデータ型

long *FmcjProcessInstance::Start()* の戻りコード。

ContextMenuSuspend

コンテキスト・メニュー項目「Suspend (一時中断)」を呼び出します。

このメソッドは、コントロール内で選択されているプロセス・インスタンスのリストを一時中断します。詳細については、599ページの『Suspend()』を参照してください。

シグニチャー

```
long ContextMenuSuspend()
```

戻り値のデータ型

long *FmcjProcessInstance::Suspend()* の戻りコード。

ContextMenuSuspendDeep

コンテキスト・メニュー項目「Suspend deep (全プロセスの一時中断)」を呼び出します。

このメソッドは、コントロール内で選択されているプロセス・インスタンスのリストを一時中断します。詳細については、599ページの『Suspend()』を参照してください。

シグニチャー

```
long ContextMenuSuspendDeep()
```

戻り値のデータ型

long *FmcjProcessInstance::Suspend()* の戻りコード。

ContextMenuTerminate

コンテキスト・メニュー項目「Terminate (中止)」を呼び出します。

このメソッドは、コントロール内で選択されているプロセス・インスタンスのリストを中止します。詳細については、602ページの『Terminate()』を参照してください。

シグニチャー

```
long ContextMenuTerminate ()
```

戻り値のデータ型

long *FmcjProcessInstance::Terminate()* の戻りコード。

RefreshProcessInstanceList

ProcessInstanceList コントロールをリフレッシュします。

シグニチャー

```
long RefreshProcessInstanceList()
```

戻り値のデータ型

long 処理が正常に完了すると、FMC_OK が戻されます。

イベント

763ページの『第64章 すべての GUI コントロールによって生成されるイベント』、765ページの『第65章 すべての非モニター GUI コントロールによって生成されるイベント』、および 769ページの『第66章 すべてのリスト・コントロールによって生成されるイベント』を参照。

第70章 WorklistCtrl

以下の説明は WorklistCtrl コントロールに関連したメソッドに関するものです。ワークリストは、ユーザーの作業項目の集合で構成されています。Worklist オブジェクトはワークリストを反映したもので、このワークリストは IBM MQSeries Workflow 実行機能サーバーのデータベースに保管されます。

プロパティー

デザイン・モードで Worklist コントロールのプロパティーを表示して調整するには、そのコントロールのグリッド域内にマウス・ポインターを移動して、マウスの右ボタンをクリックし、「Properties (プロパティー)」を選択します。

プロパティーのダイアログには、**「Fonts (フォント)」**、**「Colors (色)」**、**「Column Selection (列の選択)」**、および**「Column Attribute (列の属性)」**という 4 つのタブがあります。

タブは、すべて設計モードでも実行モードでも使用できます。

「Fonts (フォント)」タブには、標準的なフォント・ダイアログが表示されます。 そこでは、Worklist コントロールのウィンドウに表示するテキストのフォント、フォント・サイズ、スタイル、および文字飾りを操作できます。

「Colors (色)」タブには、コントロールのウィンドウのカラー・スキームを操作する機能が用意されています。

コントロールの前景と背景の色を設定したり更新したりできます。**「ForeColor (前景)」**または**「BackColor (背景)」**を選択し、16 色のボタンのうち 1 つをクリックしてから、**「Apply (適用)」**ボタンをクリックしてください。

「Column Selection (列の選択)」タブを使用すると、列や情報をコントロールの表示内容に追加したり表示内容から除去したりできます。列を追加するには、右側のペインにあるアイテムを強調表示させてから、**「Add (追加)」**ボタンをクリックします。列を削除するには、左側のペインにあるアイテムを強調表示させてから、**「Delete (削除)」**ボタンをクリックします。

「Icon view and Width (アイコン・ビューと幅)」タブには、コントロールによってそのアイテムが表示される際の特定の局面を操作するメカニズムが用意されています。それらは、次のとおりです。

1. **「Icon view (アイコン・ビュー)」**チェック・ボックスは、アイコンの表示スタイルが有効な場合に、含められる情報の列 (名前が左側のリスト・ボックスに表示されている) を指定するのに使います。ある時点で選択されている列は、右側のアイコンの下に表示されます。それらは**「Description (記述)」** と**「Activity Type (アクティビティーのタイプ)」**です。
2. **「Column width (列の幅)」**は、列のアイテムに使用されるピクセル数を指定するのに使います。これは**レポート**形式の表示内容に限り有効です。列の幅を調整するには、左側のリスト・ボックスのアイテムを強調表示してから、**「Column width (列の幅)」**入力ボックスに数値を入力して、**「Apply (適用)」**ボタンをクリックします。
3. **「Alignment (調整)」**は、列のテキストを左または右にそろえる場合に使います。これは**レポート**形式の表示内容に限り有効です。

実行機能モード

マウス・ポインターを特定のアイテム上に置いてマウスの右ボタンをクリックすると、そのアイテムのコンテキスト・メニューが表示されます。
「Properties(プロパティー)」オプションを選択すると、タブ形式のダイアログが表示されます。

プロパティーのダイアログには、**「General (一般)」**、**「Staff (スタッフ)」**、**「Start & exit (開始と終了)」**、**「History (履歴)」**、および**「Documentation (ドキュメンテーション)」**という 5 つのタブがあります。
「General (一般)」タブには、ワークリスト・アイテム・オブジェクトに関する情報の以下の部分が表示されます。それらは、次のとおりです。

- Activity name (アクティビティーの名前)
- Activity status (アクティビティーの状況)
- Activity type (アクティビティーのタイプ)
- Program (プログラム)

  アクティビティーに割り当てられているプログラムまたはプロセスの名前。
- Received as (受信)

  ワークリスト上にアクティビティーがある理由を定義するユーザーのカテゴリー。

  – そのアクティビティーが割り当てられているユーザー。

– 定義されているユーザーが不在と宣言されているためにアクティビティーを受信した代理人。
– 定義されているユーザーが不在と宣言されており、そのユーザーに代理人が指定されていないためにアクティビティーを受信したプロセス管理者。

注: 定義機能においてプロセスに対するプロセス管理者が定義されていない場合は、そのプロセス・インスタンスを開始した担当者がそのプロセス・インスタンスのプロセス管理者として割り当てられます。

– プロセス管理者が削除されているためにアクティビティーを受信したシステム管理者。

「Staff (スタッフ)」タブには、ワークリスト・アイテム・オブジェクトの管理に関する情報が表示されます。それらは、次のとおりです。

- On the worklist of (ワークリスト使用者)

  そのアクティビティーが割り当てられているユーザーのユーザー ID、あるいはそのアクティビティーがすでに開始されているなら、そのアクティビティーを開始したユーザーのユーザー ID。
- プロセス管理者

  アクティビティーが属するプロセス・インスタンスのプロセス管理者。
- Priority (優先順位)

  定義機能でそのアクティビティーに割り当てられている優先順位。
- Started on Server (開始サーバー)

  アクティビティーが開始されたサーバーの名前。

「Start & exit (開始と終了)」タブには、作業項目の開始条件と終了条件に関連する情報が表示されます。それらは、次のとおりです。

- Start (開始)

  アクティビティーの開始モード (手動または自動)。
- Start condition (開始条件):

  アクティビティーが開始するために満たされなければならない条件 (自動開始)、またはワークリストに追加されるために満たされていなければならない条件 (手動開始)。
- Exit (終了)

  アクティビティーの終了モード (手動または自動)。
- Exit condition (終了条件)

  アクティビティーが完了するために満たされなければならない条件。

「History (履歴)」タブには、以下のアクティビティー情報が表示されます。

- Received (受信)
  アクティビティーがワークリストに到着した日付と時刻。
- Notification (通知)
  そのアクティビティーに関する最初の通知が出される日付と時刻。
- Finished (終了)
  アクティビティーが完了した日付と時刻。

「Documentation (文書)」タブには、リストの注釈として追加されたものが表示されます。

注: クライアントにおいて、または Worklist コントロールを使用して作成したアプリケーションのうち、メニュー・バーも使用するものにおいては、メニュー・バーの**「Activity (アクティビティー)」**オプションによって、作業項目のプロパティー・ダイアログ (および他のすべてのコンテキスト・メニュー項目) にアクセスすることもできます。

Worklist の設定値

Worklist コントロールの設定値を表示して調整するには、そのコントロールのウィンドウ域の未使用部分にマウス・ポインターを移動して、マウスの右ボタンをクリックし、**「Worklist settings (Worklist の設定値)」**を選択します。

「General (一般)」タブには、ワークリストとその外観に関する情報のさまざまな部分が表示されます。最初の 3 列の順序 (左から右へ) とソート順 (昇順または降順) を設定できます。列の順序は、レポート形式が有効な場合にのみ適用されます。

設定値を調整したなら、**「OK (了解)」**ボタンを選択します。選択基準を満たすアイテムがワークリスト上に表示されます。フィルター基準が保管され、次回のログオンまで保たれます。

プロパティー **Appearance**、**Arrange**、**BackColor**、**BorderStyle**、**CtrlColumnOrder**、**CtrlColumnWidth**、**Font**、**ForeColor**、**View** は変更可能です。

メソッド

Worklist コントロールは、すべてのリスト・コントロールによってサポートされるメソッドのほかに、以下のメソッドをサポートします。 753ページの『第62章 すべての GUI コントロールでサポートされるメソッド』、および 757ページの『第63章 すべてのリスト・コントロールでサポートされるメソッド』を参照。

ContextMenuFinish

コンテキスト・メニュー項目「Finish (終了)」を呼び出します。

このメソッドは、コントロール内で選択されている作業項目のリストを終了します。詳細については、690ページの『Finish()』を参照してください。

シグニチャー

```
long ContextMenuFinish()
```

戻り値のデータ型

long *FmcjWorkitem::Finish()* の戻りコード。

ContextMenuForceFinish

コンテキスト・メニュー項目「Force finish (強制終了)」を呼び出します。

このメソッドは、コントロール内で選択されている作業項目のリストを終了します。詳細については、693ページの『ForceFinish()』を参照してください。

シグニチャー

```
long ContextMenuForceFinish()
```

戻り値のデータ型

long *FmcjWorkitem::ForceFinish()* の戻りコード。

ContextMenuForceRestart

コンテキスト・メニュー項目「Force restart (強制再始動)」を呼び出します。

このメソッドは、コントロール内で選択されている作業項目のリストを再始動します。詳細については、696ページの『ForceRestart()』を参照してください。

シグニチャー

```
long ContextMenuForceRestart()
```

戻り値のデータ型

long *FmcjWorkitem::ForceFinish()* の戻りコード。

ContextMenuRestart

コンテキスト・メニュー項目「Restart (再始動)」を呼び出します。

このメソッドは、コントロール内で選択されている作業項目のリストを再始動します。詳細については、706ページの『Restart()』を参照してください。

シグニチャー

```
long ContextMenuRestart()
```

戻り値のデータ型

long *FmcjWorkitem::Restart()* の戻りコード。

ContextMenuSelectAll

コンテキスト・メニュー項目「Select all (全選択)」を呼び出します。

このメソッドは、List (リスト) コントロール内のすべてのオブジェクトを選択します。

シグニチャー

```
long ContextMenuSelectAll()
```

戻り値のデータ型

long 処理が正常に完了すると、FMC_OK が戻されます。

ContextMenuStart

コンテキスト・メニュー項目「Start (開始)」を呼び出します。

このメソッドは、コントロール内で選択されている作業項目のリストを開始します。詳細については、708ページの『Start()』を参照してください。

シグニチャー

```
long ContextMenuStart()
```

戻り値のデータ型

long *FmcjWorkitem::Start()* の戻りコード。

ContextMenuStartTool

コンテキスト・メニュー項目「Start tool (開始ツール)」を呼び出します。

このメソッドは、コントロール内で選択されているサポート・ツールのリストを開始します。詳細については、710ページの『StartTool()』を参照してください。

シグニチャー

```
long ContextMenuStartTool()
```

戻り値のデータ型

long *FmcjWorkitem::StartTool()* の戻りコード。

ContextMenuTransfer

コンテキスト・メニュー項目「Transfer (転送)」を呼び出します。

このメソッドは、コントロール内で選択されている作業項目のリストを別のユーザーに転送します。「*fromUserID (転送元ユーザー ID)*」と「*toUserID (転送先ユーザー ID)*」を指定する、それぞれ別個のダイアログが表示されます。FDL のユーザー ID のリストがドロップダウン・リストに表示されます。すべてのユーザーに対する権限がある場合、このドロップダウン・リストには何も表示されません。この場合、*toUserID* を明示的に指定しなければなりません。List (リスト) コントロール内で選択されているすべてのアイテムが転送されます。

詳細については、539ページの『Transfer()』を参照してください。

シグニチャー

```
long ContextMenuTransfer()
```

戻り値のデータ型

long *FmcjWorkitem::Transfer()* の戻りコード。

PushOption

このメソッドは、このワークリストの PUSH オプションの現在の設定値を戻します。

True が戻された場合は、ワークリストの PUSH 処理がオンになっています。この場合、PUSH 処理の対象アイテム (作業項目と通知) によって、ワークリスト内のアイテムが動的に更新されます。ユーザーは、新しいアイテムをワークリストに追加したり、すでにワークリストの中に入っているアイテムの変更内容 (状態、タイム・スタンプ) を確認したりするために、ワークリストのリフレッシュ処理を呼び出す必要はありません。

MQSeries Workflow 実行機能クライアントは、このメソッドを呼び出すことによって特定のワークリストの「Settings (設定)」ダイアログにある現在の設定を表示します。

シグニチャー

```
boolean PushOption()
```

戻り値のデータ型

boolean ワークリストの PUSH がオンになっているなら *True* が戻され、そうでない場合は *False* が戻されます。

RefreshWorklist

Worklist コントロールを最新表示します。

シグニチャー

```
long RefreshWorklist()
```

戻り値のデータ型

long 処理が正常に完了すると、FMC_OK が戻されます。

SetPushOption

このメソッドは、現在のワークリストの PUSH 処理のオン / オフを切り替えるために使います。

このメソッドは、ユーザーが特定のワークリストの「Settings (設定)」ダイアログにある PUSH チェック・ボックスの値を変更した場合に、MQSeries Workflow 実行機能クライアントによって呼び出されます。

シグニチャー

```
void SetPushOption( boolean value )
```

パラメーター

value 入力。 PUSH 処理がオンになっているかどうかを示す標識。

イベント

Worklist コントロールは、すべての非モニター・コントロールによって生成されるイベントのほかに、以下のイベントを生成します。 763ページの『第64章 すべての GUI コントロールによって生成されるイベント』、765ページの『第65章 すべての非モニター GUI コントロールによって生成されるイベント』、および 769ページの『第66章 すべてのリスト・コントロールによって生成されるイベント』を参照。

ActivityInstanceNotificationChanged

この OLE イベントは、現在のワークリストの既存のアクティビティー・インスタンス通知が変更された場合に発生します。

index パラメーターには、現在のワークリストの ActivityInstanceNotifArray 内での変更後のアクティビティー・インスタンス通知のインデックスが示されます。

MQSeries Workflow 実行機能クライアントが使う WorklistCtrl は、このイベントを使って GUI に表示されている値をリフレッシュします。

シグニチャー

```
ActivityInstanceNotificationChanged( long index )
```

パラメーター

index 入力。現在のワークリストにあるアクティビティー・インスタンス通知のインデックス。

ProcessInstanceNotificationChanged

この OLE イベントは、現在のワークリストの既存のプロセス・インスタンス通知が変更された場合に発生します。

index パラメーターには、現在のワークリストの ProcessInstanceNotifArray 内での変更後のプロセス・インスタンス通知のインデックスが示されます。

MQSeries Workflow 実行機能クライアントが使う WorklistCtrl は、このイベントを使って GUI に表示されている値をリフレッシュします。

シグニチャー

```
ProcessInstanceNotificationChanged( long index )
```

パラメーター

index 入力。現在のワークリストにあるプロセス・インスタンス通知のインデックス。

WorkitemChanged

この OLE イベントは、現在のワークリストの既存の作業項目が変更された場合に発生します。

index パラメーターには、現在のワークリストの WorkitemArray 内での変更後の作業項目のインデックスが示されます。

MQSeries Workflow 実行機能クライアントが使う WorklistCtrl は、このイベントを使って GUI に表示されている値をリフレッシュします。

シグニチャー

```
WorkitemChanged( long index )
```

パラメーター

index 入力。現在のワークリストにある作業項目のインデックス。

Starting

作業項目が開始された時点で発生します。

シグニチャー

```
void Starting( BSTR name )
```

パラメーター

name 出力。開始された作業項目の名前を示す文字列。

第71章 MonitorCtrl

プロパティー

デザイン・モードで Monitor コントロールのプロパティーを表示して調整するには、そのコントロールのグリッド域内にマウス・ポインターを移動して、マウスの右ボタンをクリックし、「Properties (プロパティー)」を選択します。

プロパティーのダイアログには、**「General (一般)」**と**「Colors (色)」**という 2 つのタブがあります。

タブは、すべて設計モードでも実行モードでも使用できます。

「General (一般)」タブには「Zoom factor (ズーム因数)」が表示され、現行セッションに対して変更することが可能です。有効な値は 10 ～ 200 です。ズーム因数の省略時値は 100 になっています。

「Colors (色)」タブには、コントロールのウィンドウのカラー・スキームを操作する機能が用意されています。

現在の色設定がそこに表示されます。選択されているアイテムで使われている色、および背景用の色 (PaperColor) を現行セッションに対して変更できます。

メソッド

Monitor コントロールは、すべてのコントロールによってサポートされるメソッドのほかに、以下のメソッドをサポートします。 753ページの『第62章 すべての GUI コントロールでサポートされるメソッド』を参照。

ActivityProperties()

このメソッドは、現在選択しているアクティビティー・インスタンスのプロパティー・ページを表示します。

シグニチャー

```
void ActivityProperties()
```

ConnectGUI

MonitorCtrl オブジェクトと MQWorkflowCtrl オブジェクトを接続します。

シグニチャー

```
long ConnectGUI( IDispatch * wfc )
```

パラメーター

wfc 入力。 MQWorkflowCtrl オブジェクトへのポインター。

戻り値のデータ型

long 処理が正常に完了すると、FMC_OK が戻されます。

ControlConnectorProperties

このメソッドは、現在選択している ControlConnector のプロパティー・ページを表示します。

シグニチャー

```
void ControlConnectorProperties()
```

OpenMonitor

このメソッドは、モニター・パラメーターの指定どおりにプロセス・モデルのグラフを表示します。グラフのレイアウトは、基礎になっている C++ API 層が提供するレイアウト・コーディネートによって制御されます。この要求は 2 回以上出すと無視されます。

シグニチャー

```
void OpenMonitor( IDISPATCH * monitor )
```

パラメーター

monitor 入力。インスタンス・モニター・オブジェクトへのポインター。

Refresh

このメソッドは、現在のプロセス・モデルのグラフをリフレッシュします。アクティビティー・インスタンスの状態をリフレッシュする場合は、このメソッドを使う必要があります。自動リフレッシュはサポートしていません。

シグニチャー

```
long Refresh()
```

戻り値のデータ型

long *FmcjBlockInstanceMonitor::Refresh()* の戻りコード。

イベント

Monitor コントロールは、すべてのコントロールによって生成されるイベントのほかに、以下のイベントを生成します。 763ページの『第64章 すべての GUI コントロールによって生成されるイベント』を参照。

AfterRefreshing

この OLE イベントは、リフレッシュが終了した時点で生成されます。これは *BeforeRefreshing* イベントの後に発生します。

シグニチャー

```
void AfterRefreshing()
```

BeforeRefreshing

この OLE イベントは、ユーザーが *Refresh* メニュー項目をクリックすると発生します。これは *DoRefresh* イベントの後に発生します。

シグニチャー

```
void BeforeRefreshing()
```

BlockActivityClick

この OLE イベントは、ブロック 型のアクティビティー・インスタンスの位置で、ユーザーがマウスの右ボタンをクリックすると発生します。

シグニチャー

```
void BlockActivityClick( IDISPATCH *       activity,
                         OLE_XPOS_PIXELS   x,
                         OLE_XPOS_PIXELS   y,
                         long              button,
                         boolean *         enableDefault )
```

パラメーター

activity 入力。アクティビティー・インスタンスへのポインター。

button 入力。どちらのボタンが押されたかを示します。 0 は左ボタン、2 は右ボタンを表します。

enableDefault 入力。 Monitor コントロールがその省略時アクションを実行してコンテキスト・メニューを表示するかどうかの標識。

x 入力。クリックが発生した時点のマウスの X 座標。

y 入力。クリックが発生した時点のマウスの Y 座標。

BlockActivityDoubleClick

この OLE イベントは、ブロック 型のアクティビティー・インスタンスの位置で、ユーザーがマウスの左ボタンをダブルクリックすると発生します。

シグニチャー

```
void BlockActivityDoubleClick( IDISPATCH *       activity,
                               OLE_XPOS_PIXELS   x,
                               OLE_XPOS_PIXELS   y,
                               long              button,
                               boolean *         enableDefault )
```

パラメーター

activity 入力。アクティビティー・インスタンスへのポインター。

button 入力。どちらのボタンが押されたかを示します。 0 は左ボタン、2 は右ボタンを表します。

enableDefault 入力。 Monitor コントロールがその省略時アクションを実行して新しいモニター・ウィンドウを開くかどうかの標識。

x 入力。クリックが発生した時点のマウスの X 座標。

y 入力。クリックが発生した時点のマウスの Y 座標。

ControlConnectorClick

この OLE イベントは、制御コネクター・インスタンスの位置で、ユーザーがマウスの右ボタンをクリックすると発生します。

シグニチャー

```
void ControlConnectorClick( IDISPATCH *       connector,
                            OLE_XPOS_PIXELS   x,
                            OLE_XPOS_PIXELS   y,
                            long              button,
                            boolean *         enableDefault )
```

パラメーター

connector 入力。制御コネクター・インスタンスへのポインター。

button 入力。どちらのボタンが押されたかを示します。 0 は左ボタン、2 は右ボタンを表します。

enableDefault 入力。 Monitor コントロールがその省略時アクションを実行してコンテキスト・メニューを表示するかどうかの標識。

x 入力。クリックが発生した時点のマウスの X 座標。

y 入力。クリックが発生した時点のマウスの Y 座標。

ControlConnectorDoubleClick

この OLE イベントは、制御コネクター・インスタンスの位置で、ユーザーがマウスの左ボタンをダブルクリックすると発生します。

シグニチャー

```
void ControlConnectorDoubleClick( IDISPATCH *       connector,
                                  OLE_XPOS_PIXELS   x,
                                  OLE_XPOS_PIXELS   y,
                                  long              button,
                                  boolean *         enableDefault )
```

パラメーター

connector 入力。制御コネクター・インスタンスへのポインター。

button 入力。どちらのボタンが押されたかを示します。 0 は左ボタン、2 は右ボタンを表します。

enableDefault 入力。 Monitor コントロールがその省略時アクションを実行して制御コネクター・インスタンスのプロパティー・ページを表示するかどうかの標識。
x 入力。クリックが発生した時点のマウスの X 座標。
y 入力。クリックが発生した時点のマウスの Y 座標。

DoActivityEnter

この OLE イベントは、ユーザーがアクティビティー・インスタンスを選択して Enter キーを押すと発生します。

シグニチャー

```
void DoActivityEnter( IDISPATCH * activity,
                      boolean *   enableDefault )
```

パラメーター

activity 入力。アクティビティー・インスタンスへのポインター。
enableDefault 入力。 Monitor コントロールが、マウスのダブルクリックと同じ省略時アクションを実行するかどうかの標識。これはアクティビティー・インスタンスのタイプに依存しています。

DoControlConnectorEnter

この OLE イベントは、ユーザーが制御コネクター・インスタンスを選択して Enter キーを押すと発生します。

シグニチャー

```
void DoControlConnectorEnter( IDISPATCH * connector,
                              boolean *   enableDefault )
```

パラメーター

connector 入力。制御コネクター・インスタンスへのポインター。
enableDefault 入力。 Monitor コントロールがその省略時アクションを実行するかどうかの標識。現在のところ省略時アクションはありません。

DoRefresh

この OLE イベントは、ユーザーがコンテキスト・メニューから「*Refresh (リフレッシュ)*」 を選択すると発生します。

シグニチャー

```
void DoRefresh( boolean * enableDefault )
```

パラメーター

enableDefault 入力。 Monitor コントロールがその省略時アクションを実行して、Monitor ウィンドウの中に表示されているすべてのアクティビティー・インスタンスと制御コネクター・インスタンスの完全なリフレッシュを実行するかどうかの標識。

DoShowContextMenu

このイベントは、マウスの右ボタンをクリックしてコンテキスト・メニューを表示させた場合に発生します。ユーザーは enableDefault パラメーターを *False* に設定することにより、ContextMenu の表示を取り消すことができます。

シグニチャー

```
void DoShowContextMenu( boolean * enableDefault )
```

パラメーター

enableDefault 入力。 Monitor コントロールがその省略時アクションを実行してコンテキスト・メニューを表示するかどうかの標識。

Error

モニターのオープン時にエラーが出ると発生します。

シグニチャー

```
void Error( short     returnCode,
            BSTR *    messageText,
            boolean * cancelDisplay )
```

パラメーター

returnCode 入力。戻りコード。
messageText 入力。エラーについて説明した形式化メッセージ・テキスト。
cancelDisplay
入力。 *True* に設定されているなら、表示が取り消されます。

MonitorOpen

この OLE イベントは、ユーザーがインプレース (またはコンテキスト) メニューのモニター項目、「*Open Activity (アクティビティーのオープン)*」をクリックすると発生します。

シグニチャー

```
void MonitorOpen( IDISPATCH * activity )
```

パラメーター
activity 入力。アクティビティー・インスタンスへのポインター。

ProcessActivityClick

この OLE イベントは、*プロセス* 型のアクティビティー・インスタンスの位置で、ユーザーがマウスの右ボタンをクリックすると発生します。

シグニチャー

```
void ProcessActivityClick( IDISPATCH *       activity,
                           OLE_XPOS_PIXELS   x,
                           OLE_XPOS_PIXELS   y,
                           long              button,
                           boolean *         enableDefault )
```

パラメーター
activity 入力。アクティビティー・インスタンスへのポインター。
button 入力。どちらのボタンが押されたかを示します。 0 は左ボタン、2 は右ボタンを表します。
enableDefault 入力。 Monitor コントロールがその省略時アクションを実行してコンテキスト・メニューを表示するかどうかの標識。
x 入力。クリックが発生した時点のマウスの X 座標。
y 入力。クリックが発生した時点のマウスの Y 座標。

ProcessActivityDoubleClick

この OLE イベントは、プロセス 型のアクティビティー・インスタンスの位置で、ユーザーがマウスの左ボタンをダブルクリックすると発生します。

シグニチャー

```
void ProcessActivityDoubleClick( IDISPATCH *       activity,
                                 OLE_XPOS_PIXELS   x,
                                 OLE_XPOS_PIXELS   y,
                                 long              button,
                                 boolean *         enableDefault )
```

パラメーター

activity 入力。アクティビティー・インスタンスへのポインター。

button 入力。どちらのボタンが押されたかを示します。 0 は左ボタン、2 は右ボタンを表します。

enableDefault 入力。 Monitor コントロールがその省略時アクションを実行するかどうかの標識。現在のところ省略時アクションはありません。

x 入力。クリックが発生した時点のマウスの X 座標。

y 入力。クリックが発生した時点のマウスの Y 座標。

ProgramActivityClick

この OLE イベントは、プログラム 型のアクティビティー・インスタンスの位置で、ユーザーがマウスの右ボタンをクリックすると発生します。アクティビティー・インスタンスのプロパティー・ページが表示されます。

シグニチャー

```
void ProgramActivityClick( IDISPATCH *       activity,
                           OLE_XPOS_PIXELS   x,
                           OLE_XPOS_PIXELS   y,
                           long              button,
                           boolean *         enableDefault )
```

パラメーター

activity 入力。アクティビティー・インスタンスへのポインター。

button 入力。どちらのボタンが押されたかを示します。 0 は左ボタン、2 は右ボタンを表します。

enableDefault 入力。 Monitor コントロールがその省略時アクションを実行してコンテキスト・メニューを表示するかどうかの標識。
x 入力。クリックが発生した時点のマウスの X 座標。
y 入力。クリックが発生した時点のマウスの Y 座標。

ProgramActivityDoubleClick

この OLE イベントは、プログラム 型のアクティビティー・インスタンスの位置で、ユーザーがマウスの左ボタンをダブルクリックすると発生します。アクティビティー・インスタンスのプロパティー・ページが表示されます。

シグニチャー

```
void ProcessActivityDoubleClick( IDISPATCH *       activity,
                                 OLE_XPOS_PIXELS   x,
                                 OLE_XPOS_PIXELS   y,
                                 long              button,
                                 boolean *         enableDefault )
```

パラメーター

activity 入力。アクティビティー・インスタンスへのポインター。
button 入力。どちらのボタンが押されたかを示します。 0 は左ボタン、2 は右ボタンを表します。
enableDefault 入力。 Monitor コントロールがその省略時アクションを実行して、選択されているアクティビティー・インスタンスのプロパティー・ページを表示するかどうかの標識。
x 入力。クリックが発生した時点のマウスの X 座標。
y 入力。クリックが発生した時点のマウスの Y 座標。

第9部 例およびシナリオ

第72章 シナリオ

以下に示すシナリオは、MQSeries Workflow に付属しているものです。シナリオとは、製品の機能の一部を実際に示すためのものです。それらは実行したり、調べてみたりすることができます。シナリオは以下の方法で実行します。

1. FDL を搬入します。
2. MQSeries Workflow システムを起動します。
3. シナリオを実行する。サンプルのインストールを選択した場合、シナリオは省略時にはインストール先ディレクトリーの ¥bin サブディレクトリーにインストールされています。

アップデートがあるかどうかは、該当するディレクトリー内の readme ファイルを参照してください。

- クレジット要求のサンプル

  Windows NT と ActiveX の場合には ¥fmcwinnt¥scenario¥credit に、Windows 95 と ActiveX の場合には ¥fmcwin95¥scenario¥credit に、また Windows 98 と ActiveX の場合には ¥fmcwin98¥scenario¥credit にあります。

  – FDL: fmccred.fdl
  – アクティビティー・インプリメンテーション: fmcn6bna.vbp、fmcn6bni.vbp、fmcn6bnp.vbp、fmcn6bnr.vbp

- 生命保険要求のサンプル

  Windows NT と C 言語の場合は ¥fmcwinnt¥scenario¥life に、Windows 95 と C 言語の場合は ¥fmcwin95¥scenario¥life に、 Windows 98 と C 言語の場合は ¥fmcwin98¥scenario¥life にあります。

  – FDL: fmclife.fdl

第73章 例

以下に示すサンプルは MQSeries Workflow に付属しています。これらの例は一部の API の使用法を示すためのものです。これらはコンパイルしてリンクした後、実行することができます。一部の例には、実行可能なバージョンも示されています。サンプルのインストールを選択した場合、シナリオはインストール先ディレクトリーの ¥bin サブディレクトリーにインストールされています。

注: 最新の例については、MQSeries Workflow サポート・パックの場合、

```
//http://www-4.ibm.com/software/ts/mqseries/txppacs
```

にアクセスしてください。一例として次のものがあります。

- MQSeries Workflow API Programming Examples (MQSeries Workflow API プログラミング例)
- MQSeries Workflow HTML Client (MQSeries Workflow HTML クライアント)

アップデートがあるかどうかは、該当するディレクトリー内の readme ファイルを参照してください。

- アクティビティー・インプリメンテーションのコンテナー処理
  - fmctjcim.c (サポートしているすべてのプラットフォームと C 言語の場合。インストール・ディレクトリーの ¥smp¥c¥actimpl サブディレクトリーにあります。)
  - fmctjpim.cxx (サポートしているすべてのプラットフォームと C++ 言語の場合。インストール・ディレクトリーの ¥smp¥c++¥actimpl サブディレクトリーにあります。)
  - fmcnshow.vbp (Windows NT と ActiveX の場合は ¥fmcwinnt¥smp¥vb¥actimpl に、Windows 95 と ActiveX の場合は ¥fmcwin95¥smp¥vb¥actimpl に、 Windows 98 と ActiveX の場合は ¥fmcwin98¥smp¥vb¥actimpl にあります。)

    注: これらのプログラムは、不明のコンテナーを分析することができます。fmcnshow プログラムは特に、新しいプロセス・モデルをテストするための初期のアクティビティー・インプリメンテーションとして使えます。

- 実行機能クライアント

  fmcn6rtc.vbp (Windows NT と ActiveX の場合は ¥fmcwinnt¥smp¥vb¥rtc に、Windows 95 と ActiveX の場合は ¥fmcwin95¥smp¥vb¥rtc に、 Windows 98 と ActiveX の場合は ¥fmcwin98¥smp¥vb¥rtc にあります。)

- Hello world

  サポートしているすべてのプラットフォーム (OS/2 を除く) と Java 言語の場合:

  – HelloApplication.java (インストール・ディレクトリーの ¥smp¥java¥HelloApplication サブディレクトリーにあります。)
  – HelloApplet.java および HelloApplet.html (インストール・ディレクトリーの ¥smp¥java¥HelloApplet サブディレクトリーにあります。)
  – HelloApplet1.java および HelloApplet1.html (インストール・ディレクトリーの ¥smp¥java¥HelloApplet1 サブディレクトリーにあります。)
  – HelloServlet.java および HelloServlet.html (インストール・ディレクトリーの ¥smp¥java¥HelloServlet サブディレクトリーにあります。)

これ以降の章では、いくつかの例をさらに紹介します。それらの例は、説明されている概念を具体的に示すためだけのものです。

第74章 永続リストの作成方法

以下の例では、永続リスト (つまりオブジェクト・セットの永続ビュー) の作成方法を示します。これらの例では、プロセス・インスタンスのビューを定義します。ほかにも、プロセス・テンプレート・リストやワークリストを定義できます。

プロセス・インスタンス・リストの作成 (ActiveX)

```
Dim eService As ExecutionService
Dim Err      As String

MQWorkflowCtrl1.Connect
Index = MQWorkflowCtrl1.ExecutionServiceArray.Add("SYSTEM", "SYS_GRP")

If Index < 0 Then
  Err = "Error adding execution service to service array"
  MsgBox Err, vbCritical, "Error"
  MQWorkflowCtrl1.Disconnect
  Return
End If
Set eService = MQWorkflowCtrl1.ExecutionServiceArray.GetAt(Index)
Rc = eService.Logon("USERID", "password")
If Rc <> 0 Then
  Err = "Logon failed, rc = " + Str(Rc)
  MsgBox Err, vbCritical, "Error"
  MQWorkflowCtrl1.Disconnect
  Return
End If

Rc = eService.CreateProcessInstanceList(
              "PIL1",
              TypeOfList.TypeOfList_Private, "ADMIN", False,
              "", True,
              "", True,
              "", True,
              0, True )
If Rc <> 0 Then
  Err = "CreateProcessInstanceList failed, rc = " + Str(Rc)
  MsgBox Err, vbCritical, "Error"
End If

eService.Logoff
MQWorkflowCtrl1.Disconnect
```

プロセス・インスタンス・リストの作成 (C 言語)

```
#include <stdio.h>
#include <fmcjcrun.h>                  /* MQ Workflow Runtime API */
int main()
{
  APIRET                             rc           = FMC_OK;
  FmcjExecutionServiceHandle         service      = 0;
  FmcjProcessInstanceListHandle      instanceList = 0;
  unsigned long                      threshold    = 10;
  int                                enumValue    = 0;
  char name[50]  = "MyTenInstances";
  char desc[50]   = "This list contains no more than 10 instances";
   FmcjGlobalConnect();
   /* logon */
  rc= FmcjExecutionServiceAllocate( &service );
  if (rc != FMC_OK)
  {
    printf("Service object could not be allocated - rc: %u%¥n",rc);
    return -1;
  }


  rc= FmcjExecutionServiceLogon( service,
                                 "USERID", "password",
                                 Fmc_SM_Default, Fmc_SA_NotSet
                               );
  if (rc != FMC_OK)
  {
    printf("Logon failed - rc: %u%¥n",rc);
    FmcjExecutionServiceDeallocate( &service );
    return -1;
  }


 /* create a process instance list */
 rc = FmcjExecutionServiceCreateProcessInstanceList(
                           service,
                           name,
                           Fmc_LT_Private,
                           "USERID",
                           desc,
                           FmcjNoFilter,
                           FmcjNoSortCriteria,
                           &threshold,
                           &instanceList );
```

```
  if ( rc != FMC_OK)
    printf( "CreateProcessInstanceList returns: %u%¥n",rc );
  else
    printf( "CreateProcessInstanceList okay¥n" );
  FmcjExecutionServiceLogoff( service );
  FmcjExecutionServiceDeallocate( &service );
  FmcjGlobalDisconnect();
  return 0;
}
```

プロセス・インスタンス・リストの作成 (C++)

```
#include <iomanip.h>
#include <bool.h>                       // bool
#include <fmcjstr.hxx>                  // string
#include <vector.h>                     // vector
#include <fmcjprun.hxx>                 // MQ Workflow Runtime API
int main()
{
  FmcjGlobal::Connect();
  // logon
  FmcjExecutionService service;
  APIRET rc = service.Logon("USERID", "password");
  if ( rc != FMC_OK )
  {
    cout << "Logon failed, - rc: " << rc << endl;
    return -1;
  }
  // create a process instance list
  FmcjProcessInstanceList instanceList;
  string           name  ("MyTenInstances");
  string           desc  ("List contains no more than 10 instances");
  string           onwer ("USERID");
  unsigned long    threshold= 10;
```

```
  rc = service.CreateProcessInstanceList(
                      name,
                      FmcjPersistentList::Private,
                      &owner,
                      &desc,
                      FmcjNoFilter,
                      FmcjNoSortCriteria,
                      &threshold,
                      instanceList );
  if ( rc != FMC_OK)
     cout << "CreateProcessInstanceList returns: " << rc << endl;
  else
     cout << "CreateProcessInstanceList okay" << endl;
  service.Logoff();
  FmcjGlobal::Disconnect();
  return 0;
}
```

プロセス・インスタンス・リストの作成 (Java)

```
import com.ibm.workflow.api.*;
import com.ibm.workflow.api.ServicePackage.*;
import com.ibm.workflow.api.PersistentListPackage.*;
public class CreateProcInstList
{
  public static void main(String[] args)
  {
    // Check the arguments. The first argument is the name of the MQSeries
    // Workflow agent the client will connect to. The second argument defines
    // the locator policy the client will use when trying to contact the agent.
    // The third/fourth argument define the userid/password, which, if not
    // specified, default to USERID and password
    if ((args.length < 2 ) || (args.length > 4 ))
    {
      System.out.println("Usage:");
      System.out.println("java CreateProcessInstanceList
                                  <agent> <LOC|RMI|OSA|IOR|COS>
                                  [userid] [password]");
      System.exit(0);
    }
```

```
try
    {
      // An agent bean representing a MQSeries Workflow domain
      String userid = "USERID";
      String passwd = "password";
      Agent agent = new Agent();
      // Parse the command line and set the locator to be used to
      // communicate with the agent.
      if      (args[1].equalsIgnoreCase("LOC"))
      {
        agent.setLocator(Agent.LOC_LOCATOR);
      }
      else if (args[1].equalsIgnoreCase("RMI"))
      {
        agent.setLocator(Agent.RMI_LOCATOR);
      }
      else if (args[1].equalsIgnoreCase("OSA"))
      {
        agent.setLocator(Agent.OSA_LOCATOR);
      }
      else if (args[1].equalsIgnoreCase("IOR"))
      {
        agent.setLocator(Agent.IOR_LOCATOR);
      }
      else if (args[1].equalsIgnoreCase("COS"))
      {
        agent.setLocator(Agent.COS_LOCATOR);
      }
      else
      {
        System.out.println("Invalid locator policy: " + args[1]);
        System.exit(0);
      }



      if (args.length >=3 ) userid = args[2].toUpperCase();
      if (args.length >=4 ) passwd = args[3];
      // Set the name of the Agent to be contacted. Setting the name
      // automatically instructs the agent bean to contact the Agent using
      // the current locator policy. For this reason the 'setLocator' must be
      // called before 'setName' is invoked. If the agent bean cannot contact
      // the Agent, it will raise a java.beans.PropertyVetoException instead
      // of returning from the 'setName' call.
      agent.setName(args[0]);
      // Locate the default execution service in the system group named
      // 'SYS_GRP' and the system named 'FMCSYS'. This call intentionally
      // always returns successful (to prevent intrusion attempts which guess
      // at service names until they find a valid one). Of course, only using
      // a valid systemgroup and/or system name will return an ExecutionService
      // which can be used to log on.
      ExecutionService service = agent.locate("", "");
```

```
  // Log on to the execution service. If the UserID and/or the password is
  // invalid, a FmcException will be thrown.
  service.logon(userid, passwd);
  System.out.println("Logon successful");
  String ListName      ="MyTenInstances";
  String ListDesc      = "List contains no more than 10 instances";
  String ListFilter    = "";
  String ListSort      = "";
  int    ListThreshold = 10;


  try
  {
    service.createProcessInstanceList( ListName, TypeOfList.PRIVATE,
                                       userid  , ListDesc, ListFilter,
                                       ListSort, ListThreshold);
    System.out.println("Private ProcessInstanceList created successfully");
  }
  catch(FmcException e)
  {
     if ( e.rc == FmcException.FMC_ERROR_NOT_UNIQUE )
     {
       System.out.println("ProcessInstanceList: '" + ListName +
                          "' already exists");
     }
  }


 finally
 {
    // Logoff from the execution service. This (like any other remote call)
    // may raise an FmcException indicating a communication failure.
    service.logoff();
    System.out.println("Logoff successful");
  }
}
```

```
catch(FmcException e)
    {
      // Catch and report details about the FmcException
      System.out.println("FmcException occured");
      System.out.println("  RC         : " + e.rc);
      System.out.println("  Origin     : " + e.origin);
      System.out.println("  MessageText: " + e.messageText);
      System.out.println("  Exception  : " + e.getMessage());
      System.out.println("  Parameters : ");
      for ( int i = 0; i < e.parameters.length ; i++)
      {
        System.out.println("    " + e.parameters[i] );
      }
      System.out.println("  StackTrace : ");
      e.printStackTrace();
    }


catch(Exception e)
    {
      // Catch and report any exception that occurred.
      e.printStackTrace();
    }
    System.exit(0);
  }
}
```

第75章 永続リストを照会する方法

以下の例では、MQSeries Workflow 実行サーバーから永続リストを検索する方法と、リストの特性を照会する方法を示します。ここでは、例としてワークリストを取り上げます。ほかにも、プロセス・テンプレート・リストやプロセス・インスタンス・リストを照会できます。

ワークリストの照会 (ActiveX)

```
Dim eService As ExecutionService
Dim wl       As Worklist
Dim Err      As String
Dim Msg      As String
Dim s        As Integer
Dim i        As Integer
MQWorkflowCtrl1.Connect
Index = MQWorkflowCtrl1.ExecutionServiceArray.Add("SYSTEM", "SYS_GRP")

If Index < 0 Then
  Err = "Error adding execution service to service array"
  MsgBox Err, vbCritical, "Error"
  MQWorkflowCtrl1.Disconnect
  Return
End If
Set eService = MQWorkflowCtrl1.ExecutionServiceArray.GetAt(Index)
Rc = eService.Logon("USERID", "password")
If Rc <> 0 Then
  Err = "Logon failed, rc = " + Str(Rc)
  MsgBox Err, vbCritical, "Error"
  MQWorkflowCtrl1.Disconnect
  Return
End If
Rc = eService.QueryWorklists
If Rc <> 0 Then
  Err = "QueryWorklists failed, rc = " + Str(Rc)
  MsgBox Err, vbCritical, "Error"
Else
  s = eService.WorklistArray.GetSize
  For i = 0 To s - 1
    Set wl = eService.WorklistArray.GetAt(i)

    Msg = "Worklist: Name = " + wl.Name
    MsgBox Msg, vbInformation, "Worklist"

  Next i

End If
eService.Logoff
MQWorkflowCtrl1.Disconnect
```

ワークリストの照会 (C 言語)

```
#include <stdio.h>
#include <memory.h>
#include <fmcjcrun.h>                  /* MQ Workflow Runtime API */
int main()
{
  APIRET                        rc           = FMC_OK;
  FmcjExecutionServiceHandle    service      = 0;
  FmcjWorklistHandle            worklist     = 0;
  FmcjWorklistVectorHandle      lists        = 0;
  unsigned long                 numWList     = 0;
  unsigned long                 i            = 0;
  unsigned long                 enumValue    = 0;
  char                          tInfo[4096+1]= "";


  FmcjGlobalConnect();
  /* logon */
  rc= FmcjExecutionServiceAllocate( &service );
  if (rc != FMC_OK)
  {
    printf("Service object could not be allocated - rc: %u%¥n",rc);
    return -1;
  }
  rc= FmcjExecutionServiceLogon( service,
                                 "USERID", "password",
                                 Fmc_SM_Default, Fmc_SA_NotSet
                               );
  if (rc != FMC_OK)
  {
    printf("Logon failed - rc: %u%¥n",rc);
    FmcjExecutionServiceDeallocate( &service );
    return -1;
  }


  /* query worklists  */
  rc = FmcjExecutionServiceQueryWorklists( service, &lists );
  if ( rc != FMC_OK)
     printf( "QueryWorklists() returns: %u%¥n",rc );
  else
     printf( "QueryWorklists() returns okay¥n" );
```

```
if (rc == FMC_OK)
{
  numWList= FmcjWorklistVectorSize(lists);
  printf ("Number of worklists returned : %u¥n", numWList);
  for( i=1; i<= numWList; i++ )
  {
    worklist= FmcjWorklistVectorNextElement(lists);
    FmcjWorklistName( worklist, tInfo, 4097 );
    printf("- Name                          : %s¥n",tInfo);


    enumValue= FmcjWorklistType(worklist);
    if ( enumValue == Fmc_LT_Private )
      printf("- Type                          : %s¥n","private");
    if ( enumValue == Fmc_LT_Public )
      printf("- Type                          : %s¥n","public");
    FmcjWorklistOwnerOfList( worklist, tInfo, 4097 );
    printf("- OwnerOfList                   : %s¥n",tInfo);
    printf("- OwnerOfList is null ?         : %u¥n",
              FmcjWorklistOwnerOfListIsNull(worklist) );


    FmcjWorklistDescription( worklist, tInfo, 4097 );
    printf("- Description                   : %s¥n",tInfo);
    printf("- Description is null ?         : %u¥n",
              FmcjWorklistDescriptionIsNull(worklist) );


    FmcjWorklistFilter( worklist, tInfo, 4097 );
    printf("- Filter                        : %s¥n",tInfo);
    printf("- Filter is null ?              : %u¥n",
              FmcjWorklistFilterIsNull(worklist) );


    FmcjWorklistSortCriteria( worklist, tInfo, 4097 );
    printf("- SortCriteria                  : %s¥n",tInfo);
    printf("- SortCriteria is null ?        : %u¥n",
              FmcjWorklistSortCriteriaIsNull(worklist) );


    printf("- Threshold                     : %u¥n",
              FmcjWorklistThreshold(worklist) );
    printf("- Threshold is null ?           : %u¥n",
              FmcjWorklistThresholdIsNull(worklist) );
                                /* deallocate just read object  */
    FmcjWorklistDeallocate(&worklist);
  }
  FmcjWorklistVectorDeallocate(&lists);
}
```

```
  FmcjExecutionServiceLogoff( service );
  FmcjExecutionServiceDeallocate( &service );
  FmcjGlobalDisconnect();
  return 0;
}
```

ワークリストの照会 (C++)

```
#include <iomanip.h>
#include <bool.h>                       // bool
#include <fmcjstr.hxx>                  // string
#include <vector.h>                     // vector
#include <fmcjprun.hxx>                 // MQ Workflow Runtime API
int main()
{
  FmcjGlobal::Connect();
  // logon
  FmcjExecutionService service;
  APIRET rc = service.Logon("USERID", "password");
  if ( rc != FMC_OK )
  {
    cout << "Logon failed, - rc: " << rc << endl;
    return -1;
  }


  // query worklists
  vector<FmcjWorklist> lists;
  FmcjWorklist         worklist;
  rc = service.QueryWorklists( lists );
  if ( rc != FMC_OK)
     cout << "QueryWorklists() returns: " << rc << endl;
  else
     cout << "QueryWorklists returns okay" << endl;


  if (rc == FMC_OK)
  {
    unsigned int numWList= lists.size();
    cout << "Number of worklists returned : " << numWList << endl;


    for( unsigned long i=0; i< numWList; i++ )
    {
      worklist= lists[i];
      cout << "Name              : " << worklist.Name()              << endl;
```

```
        cout << "Type              : " <<
                ((worklist.Type() == FmcjPersistentList::Private) ? "private" :
                 (worklist.Type() == FmcjPersistentList::Public)  ? "public"  :
                 "not set" )           << endl;


       cout << "Owner             : " << worklist.OwnerOfList()       << endl;
        cout << "Owner null ?      : " << worklist.OwnerOfListIsNull() << endl;


        cout << "Description       : " << worklist.Description()       << endl;
        cout << "Description null ?: " << worklist.DescriptionIsNull() << endl;


       cout << "Filter            : " << worklist.Filter()            << endl;
        cout << "Filter null ?     : " << worklist.FilterIsNull()      << endl;
        cout << "SortCriteria      : " << worklist.SortCriteria()      << endl;
        cout << "SortCriteria null?: " << worklist.SortCriteriaIsNull()<< endl;


        cout << "Threshold         : " << worklist.Threshold()         << endl;
        cout << "Threshold null ?  : " << worklist.ThresholdIsNull()   << endl;
        cout << endl;     }     cout << endl;   }


  rc = service.Logoff();
  FmcjGlobal::Disconnect();
  return 0;
}
```

ワークリストの照会 (Java)

```
import com.ibm.workflow.api.*;
import com.ibm.workflow.api.ServicePackage.*;
import com.ibm.workflow.api.PersistentListPackage.*;
public class QueryWorkLists
{
  public static void main(String[] args)
  {
    // Check the arguments. The first argument is the name of the MQSeries
    // Workflow agent the client will connect to. The second argument defines
    // the locator policy the client will use when trying to contact the agent.
    // The third/fourth argument define the userid/password, which, if not
    // specified, default to USERID and password
    //
    if ((args.length < 2 ) || (args.length > 4 ))
    {
      System.out.println("Usage:");
      System.out.println("java QueryWorkLists [userid] [password]");
      System.exit(0);
    }
```

```
try
{
  // An agent bean representing a MQSeries Workflow domain
  String userid = "USERID";
  String passwd = "password";
  Agent agent = new Agent();
  // Parse the command line and set the locator to be used to
  // communicate with the agent.
  if      (args[1].equalsIgnoreCase("LOC"))
  {
    agent.setLocator(Agent.LOC_LOCATOR);
  }
  else if (args[1].equalsIgnoreCase("RMI"))
  {
    agent.setLocator(Agent.RMI_LOCATOR);
  }
  else if (args[1].equalsIgnoreCase("OSA"))
  {
    agent.setLocator(Agent.OSA_LOCATOR);
  }
  else if (args[1].equalsIgnoreCase("IOR"))
  {
    agent.setLocator(Agent.IOR_LOCATOR);
  }
  else if (args[1].equalsIgnoreCase("COS"))
  {
    agent.setLocator(Agent.COS_LOCATOR);
  }
  else
  {
    System.out.println("Invalid locator policy: " + args[1]);
    System.exit(0);
  }


  if (args.length >=3 ) userid = args[2].toUpperCase();
  if (args.length >=4 ) passwd = args[3];
  // Set the name of the Agent to be contacted. Setting the name
  // automatically instructs the agent bean to contact the Agent using
  // the current locator policy. For this reason the 'setLocator' must be
  // called before 'setName' is invoked. If the agent bean cannot contact
  // the Agent, it will raise a java.beans.PropertyVetoException instead
  // of returning from the 'setName' call.
  agent.setName(args[0]);
```

```
// Locate the default execution service in the system group named
     // 'SYS_GRP' and the system named 'FMCSYS'. This call intentionally
     // always returns successful (to prevent intrusion attempts which guess
     // at service names until they find a valid one). Of course, only using
     // a valid systemgroup and/or system name will return an ExecutionService
     // which can be used to log on.
     ExecutionService service = agent.locate("", "");
     // Log on to the execution service. If the UserID and/or the password is
     // invalid, a FmcException will be thrown.
     // do a forced logon
     service.logon2(userid, passwd, SessionMode.PRESENT_HERE,
                    AbsenceIndicator.LEAVE );
     System.out.println("Logon successful");



     // Query the set of worklists the logged on user can access.
     WorkList[] worklists = service.queryWorkLists();
     if (worklists.length == 0)
     {
       System.out.println("  No worklist found");
     }
     else
     {
       System.out.println("  Number of worklists returned: " + worklists.length
);



       // Iterate over the worklists, printing out their names.
       for (int ndx = 0; ndx < worklists.length; ndx++)
       {
         System.out.println("     Name               :" + worklists[ndx].name());
         if (worklists[ndx].type() == TypeOfList.PUBLIC )
         {
           System.out.println("     Type               :Public " );
         }
         else if (worklists[ndx].type() == TypeOfList.PRIVATE )
         {
           System.out.println("     Type               :Private" );
         }
         else
         {
           System.out.println("     Type               :NotSet " );
         }



         System.out.println("    Owner              :" + worklists[ndx].ownerOfList());
         System.out.println("    Description        :"+ worklists[ndx].description());
         System.out.println("    Filter             :"+ worklists[ndx].filter());
         System.out.println("    SortCriteria       :"+ worklists[ndx].sortCriteria());
         System.out.println("    Threshold          :"+ worklists[ndx].threshold());
         System.out.println("    ");
       }
     }/* End if*/
```

```
      // Logoff from the execution service. This (like any other remote call)
      // may raise an FmcException indicating a communication failure.
      service.logoff();
      System.out.println("Logoff successful");
    }


catch(FmcException e)
    {
      // Catch and report details about the FmcException
      System.out.println("FmcException occured");
      System.out.println("  RC         : " + e.rc);
      System.out.println("  Origin     : " + e.origin);
      System.out.println("  MessageText: " + e.messageText);
      System.out.println("  Exception  : " + e.getMessage());
      System.out.println("  Parameters : ");
      for ( int i = 0; i < e.parameters.length ; i++)
      {
        System.out.println("    " + e.parameters[i] );
      }
      System.out.println("  StackTrace : ");
      e.printStackTrace();
    }


 catch(Exception e)
    {
      // Catch and report any exception that occurred.
      e.printStackTrace();
    }
    System.exit(0);
  }
}
```

第76章 オブジェクトのセットを照会する方法

以下の例では、許可のあるオブジェクトを照会する方法を示します。ここでは、随時照会の例を示すためにプロセス・インスタンスの照会を使います。さらに、事前定義リストであるワークリストの内容を照会する方法を示すために、作業項目を使います。

注: ActiveX は、事前定義リストからのオブジェクトの照会のみをサポートします。

プロセス・インスタンス・リストからのプロセス・インスタンスの照会 (ActiveX)

```
Dim eService As ExecutionService
Dim pil      As ProcessInstanceList
Dim Err      As String
Dim Msg      As String
Dim s        As Integer
Dim i        As IntegerDim
MQWorkflowCtrl1.Connect
Index = MQWorkflowCtrl1.ExecutionServiceArray.Add("SYSTEM", "SYS_GRP")

If Index < 0 Then
  Err = "Error adding execution service to service array"
  MsgBox Err, vbCritical, "Error"
  MQWorkflowCtrl1.Disconnect
  Return
End If
Set eService = MQWorkflowCtrl1.ExecutionServiceArray.GetAt(Index)
Rc = eService.Logon("USERID", "password")
If Rc <> 0 Then
  Err = "Logon failed, rc = " + Str(Rc)
  MsgBox Err, vbCritical, "Error"
  MQWorkflowCtrl1.Disconnect
  Return
End If

Rc = eService.QueryProcessInstanceLists
If Rc <> 0 Then
  Err = "QueryProcessInstanceLists failed, rc = " + Str(Rc)
  MsgBox Err, vbCritical, "Error"
Else
  s = eService.ProcessInstanceListArray.GetSize

  If s > 0 Then
    Set pil = eService.ProcessInstanceListArray.GetAt(0)
    Rc = pil.QueryProcessInstances
    If Rc <> 0 Then
      Err = "QueryProcessInstances failed, rc = " + Str(Rc)
      MsgBox Err, vbCritical, "Error"
    Else
      Msg = "Number of instances returned: " + Str(pil.GetSize)
      MsgBox Msg, vbInformation, "ProcessInstances"
    End If
  Else
    Err = "No ProcessInstanceList available"
    MsgBox Err, vbCritical, "Error"
  End If
End If

eService.Logoff
MQWorkflowCtrl1.Disconnect
```

プロセス・インスタンスの照会 (C 言語)

```
#include <stdio.h>
#include <memory.h>
#include <fmcjcrun.h>                  /* MQ Workflow Runtime API */
int main()
{
  APIRET                           rc           = FMC_OK;
  FmcjExecutionServiceHandle       service      = 0;
  FmcjProcessInstanceHandle        instance     = 0;
  FmcjProcessInstanceVectorHandle iList         = 0;
  unsigned long                    numIList     = 0;
  unsigned long                    i            = 0;
  char                             tInfo[4096+1]= "";
  FmcjGlobalConnect();
  /* logon */
  rc= FmcjExecutionServiceAllocate( &service );
  if (rc != FMC_OK)
  {
    printf("Service object could not be allocated - rc: %u%¥n",rc);
    return -1;
  }
  rc= FmcjExecutionServiceLogon( service,
                                 "USERID", "password",
                                 Fmc_SM_Default, Fmc_SA_NotSet
                               );
  if (rc != FMC_OK)
  {
    printf("Logon failed - rc: %u%¥n",rc);
    FmcjExecutionServiceDeallocate( &service );
    return -1;
  }
  /* query process instances  */
  rc= FmcjExecutionServiceQueryProcessInstances(
                          service,
                          FmcjNoFilter, FmcjNoSortCriteria, FmcjNoThreshold,
                          &iList               );
  if ( rc != FMC_OK)
     printf( "QueryProcessInstances() returns: %u%¥n",rc );
  else
     printf( "QueryProcessInstances() returns okay¥n" );
```

```
  if (rc == FMC_OK)
  {
    numIList= FmcjProcessInstanceVectorSize(iList);
    printf ("Number of instances returned : %u¥n", numIList);
    for( i=1; i<= numIList; i++ )
    {
      instance= FmcjProcessInstanceVectorNextElement(iList);
      FmcjProcessInstanceName( instance, tInfo, 4097 );
      printf("- Name                        : %s¥n",tInfo);
      FmcjProcessInstanceDeallocate(&instance);
    }
    FmcjProcessInstanceVectorDeallocate(&iList);
  }
  FmcjExecutionServiceLogoff( service );
  FmcjExecutionServiceDeallocate( &service );
  FmcjGlobalDisconnect();
  return 0;
}
```

プロセス・インスタンスの照会 (C++)

```
#include <iomanip.h>
#include <bool.h>                       // bool
#include <fmcjstr.hxx>                  // string
#include <vector.h>                     // vector
#include <fmcjprun.hxx>                 // MQ Workflow Runtime API
int main()
{
  FmcjGlobal::Connect();
  // logon
  FmcjExecutionService service;
  APIRET rc = service.Logon("USERID", "password");
  if ( rc != FMC_OK )
  {
    cout << "Logon failed, - rc: " << rc << endl;
    return -1;
  }



  // query process instances
  vector<FmcjProcessInstance> instances;
  rc = service.QueryProcessInstances(
                   FmcjNoFilter, FmcjNoSortCriteria, FmcjNoThreshold,
                   instances );
  if ( rc != FMC_OK)
     cout << "QueryProcessInstances returns: " << rc << endl;
  else
     cout << "QueryProcessInstances okay" << endl;
```

```
  if ( rc == FMC_OK )
  {
    cout << "Number of instances returned: " << instances.size() << endl;
    for ( int i=0; i < instances.size(); i++ )
      cout << "- Name: " << instances[i].Name() << endl;
  }
  service.Logoff();
  FmcjGlobal::Disconnect();
  return 0;
}
```

プロセス・インスタンスの照会 (Java)

```
import com.ibm.workflow.api.*;
import com.ibm.workflow.api.ServicePackage.*;
public class QueryProcInst
{
  public static void main(String[] args)
  {
    // Check the arguments. The first argument is the name of the MQSeries
    // Workflow agent the client will connect to. The second argument defines
    // the locator policy the client will use when trying to contact the agent.
    // The third/fourth argument define the userid/password, which, if not
    // specified, default to USERID and password

    if ((args.length < 2 ) || (args.length > 4 ))
    {
      System.out.println("Usage:");
      System.out.println(" java QueryProcessInstances [userid] [password]");
      System.exit(0);
    }
```

```
try
    {
      // An agent bean representing a MQSeries Workflow domain
      String userid = "USERID";
      String passwd = "password";
      Agent agent = new Agent();
      // Parse the command line and set the locator to be used to
      // communicate with the agent.
      if      (args[1].equalsIgnoreCase("LOC"))
      {
        agent.setLocator(Agent.LOC_LOCATOR);
      }
      else if (args[1].equalsIgnoreCase("RMI"))
      {
        agent.setLocator(Agent.RMI_LOCATOR);
      }
      else if (args[1].equalsIgnoreCase("OSA"))
      {
        agent.setLocator(Agent.OSA_LOCATOR);
      }
      else if (args[1].equalsIgnoreCase("IOR"))
      {
        agent.setLocator(Agent.IOR_LOCATOR);
      }
      else if (args[1].equalsIgnoreCase("COS"))
      {
        agent.setLocator(Agent.COS_LOCATOR);
      }
      else
      {
        System.out.println("Invalid locator policy: " + args[1]);
        System.exit(0);
      }


    if (args.length >=3 ) userid = args[2].toUpperCase();
      if (args.length >=4 ) passwd = args[3];
      // Set the name of the Agent to be contacted. Setting the name
      // automatically instructs the agent bean to contact the Agent using
      // the current locator policy. For this reason the 'setLocator' must be
      // called before 'setName' is invoked. If the agent bean cannot contact
      // the Agent, it will raise a java.beans.PropertyVetoException instead
      // of returning from the 'setName' call.
      agent.setName(args[0]);
      // Locate the default execution service in the system group named
      // 'SYS_GRP' and the system named 'FMCSYS'. This call intentionally
      // always returns successful (to prevent intrusion attempts which guess
      // at service names until they find a valid one). Of course, only using
      // a valid systemgroup and/or system name will return an ExecutionService
      // which can be used to log on.
      ExecutionService service = agent.locate("", "");
```

```
      // Log on to the execution service. If the UserID and/or the password is
      // invalid, a FmcException will be thrown.
      // do a forced logon
      service.logon2(userid, passwd, SessionMode.PRESENT_HERE,
                     AbsenceIndicator.LEAVE );
      System.out.println("Logon successful");


      // Query a set of processinstances (30 at maximum), sort them by name
      ProcessInstance[] procInstances =
        service.queryProcessInstances("","NAME DESC", 30);
      if (procInstances.length == 0)
      {
        System.out.println("  No process instances found");
      }
      else
      {
        System.out.println("Number of instances returned: " + procInstances.length);


        // Iterate over the process instances, printing out their names.
        for (int ndx = 0; ndx < procInstances.length; ndx++)
        {
          System.out.println("   - Name: " + procInstances[ndx].name());
        }
      }


   // Logoff from the execution service. This (like any other remote call)
      // may raise an FmcException indicating a communication failure.
      service.logoff();
      System.out.println("Logoff successful");
    }


catch(FmcException e)
    {
      // Catch and report details about the FmcException
      System.out.println("FmcException occured");
      System.out.println("  RC         : " + e.rc);
      System.out.println("  Origin     : " + e.origin);
      System.out.println("  MessageText: " + e.messageText);
      System.out.println("  Exception  : " + e.getMessage());
      System.out.println("  Parameters : ");
      for ( int i = 0; i < e.parameters.length ; i++)
      {
        System.out.println("    " + e.parameters[i] );
      }
      System.out.println("  StackTrace : ");
      e.printStackTrace();
    }
```

```
    catch(Exception e)
    {
      // Catch and report any exception that occurred.
      e.printStackTrace();
    }
    System.exit(0);
  }
}
```

ワークリストから作業項目を照会する (ActiveX)

```
Dim eService As ExecutionService
Dim wl       As Worklist
Dim Err      As String
Dim Msg      As String
Dim s        As Integer
Dim i        As Integer
MQWorkflowCtrl1.Connect
Index = MQWorkflowCtrl1.ExecutionServiceArray.Add("SYSTEM", "SYS_GRP")

If Index < 0 Then
  Err = "Error adding execution service to service array"
  MsgBox Err, vbCritical, "Error"
  MQWorkflowCtrl1.Disconnect
  Return
End If
Set eService = MQWorkflowCtrl1.ExecutionServiceArray.GetAt(Index)
Rc = eService.Logon("USERID", "password")
If Rc <> 0 Then
  Err = "Logon failed, rc = " + Str(Rc)
  MsgBox Err, vbCritical, "Error"
  MQWorkflowCtrl1.Disconnect
  Return
End If

Rc = eService.QueryWorklists
If Rc <> 0 Then
  Err = "QueryWorklists failed, rc = " + Str(Rc)
  MsgBox Err, vbCritical, "Error"
Else
  s = eService.WorkliistArray.GetSize

  If s > 0 Then
    Set wl = eService.WorklistArray.GetAt(0)
    Rc = wl.QueryWorkitems
    If Rc <> 0 Then
      Err = "QueryWorkitems failed, rc = " + Str(Rc)
      MsgBox Err, vbCritical, "Error"
    Else
      Msg = "Number of workitems returned: " + Str(wl.GetSize)
      MsgBox Msg, vbInformation, "Workitems"
    End If
  Else
    Err = "No Worklist available"
    MsgBox Err, vbCritical, "Error"
  End If
End If

eService.Logoff
MQWorkflowCtrl1.Disconnect
```

ワークリストから作業項目を照会する (C 言語)

```
#include <stdio.h>
#include <string.h>
#include <fmcjcrun.h>                   /* MQ Workflow Runtime API */
int main (int argc, char ** argv)
{
  APIRET                        rc            = FMC_OK;
  FmcjExecutionServiceHandle    service       = 0;
  FmcjWorklistVectorHandle      wLists        = 0;
  FmcjWorklistHandle            worklist      = 0;
  FmcjWorkitemVectorHandle      wVector       = 0;
  FmcjWorkitemHandle            workitem      = 0;
  unsigned long                 numWList      = 0;
  char                          tInfo[4096+1] = "";
  FmcjGlobalConnect();
  /*  Logon */
  rc= FmcjExecutionServiceAllocate( &service );
  if (rc != FMC_OK)
  {
    printf("Service object could not be allocated: %u%¥n",rc);
    return -1;
  }
  rc= FmcjExecutionServiceLogon( service,
                                 "USERID", "password",
                                 Fmc_SM_Default, Fmc_SA_NotSet );
  if ( rc != FMC_OK )
  {
    printf("Logon failed - rc    : %u%¥n",rc);
    rc= FmcjExecutionServiceDeallocate( &service );
    return -1;
  }
  /* query worklists  */
  rc = FmcjExecutionServiceQueryWorklists( service, &wLists );
  if ( rc != FMC_OK)
     printf( "QueryWorklists() returns: %u%¥n",rc );
  else
     printf( "QueryWorklists() returns okay¥n" );
```

```
  if (rc == FMC_OK)
  {
    numWList= FmcjWorklistVectorSize(wLists);
    printf ("Number of worklists returned : %u¥n", numWList);
    if ( numWList == 0 )
    {
      printf("No worklist found ¥n");
      FmcjWorklistVectorDeallocate(&wLists);
      rc= FmcjExecutionServiceDeallocate( &service );
      return -1;
    }
    worklist= FmcjWorklistVectorFirstElement(wLists);
    FmcjWorklistName( worklist, tInfo, 4097 );
    printf("Name                          : %s¥n",tInfo);
    /* query workitems  */
    rc= FmcjWorklistQueryWorkitems( worklist, &wVector );
    printf("¥nQuery workitems of list returns rc: %u¥n",rc);
    if (rc == FMC_OK)
    {
      while ( 0 != (workitem= FmcjWorkitemVectorNextElement(wVector)) )
      {
        FmcjWorkitemName( workitem, tInfo, 4097 );
        printf("- Name                        : %s¥n",tInfo);
        FmcjWorkitemDeallocate(&workitem);
      }
    }
    FmcjWorklistDeallocate(&worklist);
    FmcjWorklistVectorDeallocate(&wLists);
  }
  /*  Logoff */
  rc= FmcjExecutionServiceLogoff(service);
  rc= FmcjExecutionServiceDeallocate( &service );
  FmcjGlobalDisconnect();
  return 0;
}
```

ワークリストから作業項目を照会する (C++)

```
#include <iomanip.h>
#include <bool.h>                       // bool
#include <fmcjstr.hxx>                  // string
#include <vector.h>                     // vector
#include <fmcjprun.hxx>                 // MQ Workflow Runtime API
int main()
{
  FmcjGlobal::Connect();
```

```
  // logon
  FmcjExecutionService service;
  APIRET rc = service.Logon("USERID", "password");
  if ( rc != FMC_OK )
  {
    cout << "Logon failed, - rc: " << rc << endl;
    return -1;
  }


  // query worklists
  vector<FmcjWorklist> lists;
  FmcjWorklist         worklist;
  rc = service.QueryWorklists( lists );
  if ( rc != FMC_OK)
     cout << "QueryWorklists() returns: " << rc << endl;
  else
     cout << "QueryWorklists returns okay" << endl;
  if (rc == FMC_OK)
  {
    unsigned int numWList= lists.size();
    cout << "Number of worklists returned : " << numWList << endl;
    if ( numWList == 0 )
    {
      cout << "No worklist found" << endl;
      return -1;
    }


    worklist= lists[0];
    cout << "Name              : " << worklist.Name()              << endl;
    vector<FmcjWorkitem> wVector;
    FmcjWorkitem         workitem;
    rc= worklist.QueryWorkitems( wVector );
    cout << "Query workitems of list returns: " << rc << endl;
    cout << "Number of workitems              " << wVector.size() << endl;


    if (rc == FMC_OK)
    {
      for ( int i= 0; i < wVector.size(); i++ )
      {
        workitem= wVector[i];
        cout << "Name              : " << workitem.Name() << endl;
      }
    }
  }
  rc = service.Logoff();
  FmcjGlobal::Disconnect();
  return 0;
}
```

ワークリストから作業項目を照会する (Java)

```
import com.ibm.workflow.api.*;
import com.ibm.workflow.api.ServicePackage.*;
public class QueryWorkItems
{
  public static void main(String[] args)
  {
    // Check the arguments. The first argument is the name of the MQSeries
    // Workflow agent the client will connect to. The second argument defines
    // the locator policy the client will use when trying to contact the agent.
    // The third/fourth argument define the userid/password, which, if not
    // specified, default to USERID and password

    if ((args.length < 2 ) || (args.length > 4 ))
    {
      System.out.println("Usage:");
      System.out.println(" java QueryWorkitems [userid] [password]");
      System.exit(0);
    }
```

```
try
{
  // An agent bean representing a MQSeries Workflow domain
  String userid = "USERID";
  String passwd = "password";
  Agent agent = new Agent();
  // Parse the command line and set the locator to be used to
  // communicate with the agent.
  if      (args[1].equalsIgnoreCase("LOC"))
  {
    agent.setLocator(Agent.LOC_LOCATOR);
  }
  else if (args[1].equalsIgnoreCase("RMI"))
  {
    agent.setLocator(Agent.RMI_LOCATOR);
  }
  else if (args[1].equalsIgnoreCase("OSA"))
  {
    agent.setLocator(Agent.OSA_LOCATOR);
  }
  else if (args[1].equalsIgnoreCase("IOR"))
  {
    agent.setLocator(Agent.IOR_LOCATOR);
  }
  else if (args[1].equalsIgnoreCase("COS"))
  {
    agent.setLocator(Agent.COS_LOCATOR);
  }
  else
  {
    System.out.println("Invalid locator policy: " + args[1]);
    System.exit(0);
  }
```

```
 if (args.length >=3 ) userid = args[2].toUpperCase();
 if (args.length >=4 ) passwd = args[3];
 // Set the name of the Agent to be contacted. Setting the name
 // automatically instructs the agent bean to contact the Agent using
 // the current locator policy. For this reason the 'setLocator' must be
 // called before 'setName' is invoked. If the agent bean cannot contact
 // the Agent, it will raise a java.beans.PropertyVetoException instead
 // of returning from the 'setName' call.
 agent.setName(args[0]);
 // Locate the default execution service in the system group named
 // 'SYS_GRP' and the system named 'FMCSYS'. This call intentionally
 // always returns successful (to prevent intrusion attempts which guess
 // at service names until they find a valid one). Of course, only using
 // a valid systemgroup and/or system name will return an ExecutionService
 // which can be used to log on.
 ExecutionService service = agent.locate("", "");
 // Log on to the execution service. If the UserID and/or the password is
 // invalid, a FmcException will be thrown.
 // do a forced logon
 service.logon2(userid, passwd, SessionMode.PRESENT_HERE,
                AbsenceIndicator.LEAVE );
 System.out.println("Logon successful");


 // Query the set of worklists the logged on user can access.
 WorkList[] worklists = service.queryWorkLists();
 if (worklists.length == 0)
 {
   System.out.println("  No worklist found");
 }
 else
 {
   System.out.println("  Number of worklists returned: " + worklists.length);
   WorkList worklist = worklists[0];
   System.out.println("  Name: "+worklist.name());



    // Query the set of workitems in the first worklist.
    WorkItem[] workitems = worklist.queryWorkItems();
    System.out.println("  Number of workitems: " + workitems.length);
    // Iterate over the workitems, printing out their names.
    for (int ndx = 0; ndx < workitems.length; ndx++)
    {
      System.out.println("    " + workitems[ndx].name());
    }
 }/* End if*/



 // Logoff from the execution service. This (like any other remote call)
 // may raise an FmcException indicating a communication failure.
 service.logoff();
 System.out.println("Logoff successful");
}
```

```
 catch(FmcException e)
    {
      // Catch and report details about the FmcException
      System.out.println("FmcException occured");
      System.out.println("  RC         : " + e.rc);
      System.out.println("  Origin     : " + e.origin);
      System.out.println("  MessageText: " + e.messageText);
      System.out.println("  Exception  : " + e.getMessage());
      System.out.println("  Parameters : ");
      for ( int i = 0; i < e.parameters.length ; i++)
      {
        System.out.println("    " + e.parameters[i] );
      }
      System.out.println("  StackTrace : ");
      e.printStackTrace();
    }


   catch(Exception e)
    {
      // Catch and report any exception that occurred.
      e.printStackTrace();
    }
    System.exit(0);
  }
}
```

第77章 アクティビティー・インプリメンテーション

以下の例では、アクティビティー・インプリメンテーションの内部からコンテナーを照会したり設定したりする方法を示します。詳細については、製品に添付されているサンプルを参照してください。

実行可能なアクティビティー・インプリメンテーションのプログラミング (C 言語)

```
#include <stdio.h>
#include <fmcjccon.h>                  /* MQ Workflow Container API */
int main()
{
  FILE                            *  file1             = 0;
  APIRET                             rc                = FMC_OK;
  FmcjReadOnlyContainerHandle        input             = 0;
  FmcjReadWriteContainerHandle       output            = 0;
  char                               stringBuffer[4097]="";
/*- keep results in a file ------------------------------------------------*/
  file1 = fopen ("sample.out", "a");
  if ( file1 == 0 )
    return -1;
  fprintf(file1,"¥n----- C-API Activity Implementation called -----¥n");
  fflush(file1);


  FmcjGlobalConnect();
/*-- retrieve the input container from the PEA who started the program --*/
  rc = FmcjContainerInContainer( &input );
  fprintf(file1, "Get Input Container - rc:  %u¥n", rc);
  if (rc != FMC_OK)
  {
    fclose(file1);
    return 1;
  }
  fprintf(file1, "Input Container Name: %s¥n",
          FmcjReadOnlyContainerType(input, stringBuffer, 4097));
```

```
/*-- retrieve the output container from the PEA who started the program -*/
  rc = FmcjContainerOutContainer( &output );
  fprintf(file1, "Get Output Container - rc:  %u¥n", rc);
  if (rc != FMC_OK)
  {
    fclose(file1);
    return 1;
  }
  fprintf(file1, "Output Container Name: %s¥n",
          FmcjReadWriteContainerType(output, stringBuffer, 4097));


/*----- Modify output values ------------------------------------------*/
  rc= FmcjReadWriteContainerSetLongValue(output, "aFieldInTheOutput",42);
  fprintf(file1, "¥nSetting long value returns rc: %u¥n", rc);
  ...
/*-- return the output container to the PEA who started the program -----*/
  rc = FmcjContainerSetOutContainer( output );
  fprintf(file1, "¥nSet Output Container - rc: %u¥n",rc);
  fflush(file1);
  FmcjGlobalDisconnect();
  fclose(file1);
  return 0;                             // _RC passed to MQSeries Workflow
}
```

実行可能なアクティビティー・インプリメンテーションのプログラミング (C++)

```
#include <fstream.h>
#include <bool.h>                       // bool
#include <fmcjstr.hxx>                  // string
#include <vector.h>                     // vector
#include <fmcjpcon.hxx>                 // MQ Workflow Container API
int main()
{
/*- keep results in a file ---------------------------------------------*/
  ofstream file1("sample.out");
  if ( file1 == 0 )
    return -1;
  file1 << "¥n----- C++-API Activity Implementation called -----¥n" << endl;
```

```
  FmcjGlobal::Connect();
/*-- retrieve the input container from the PEA who started the program --*/
  FmcjReadOnlyContainer input;
  APIRET rc = FmcjContainer::InContainer( input );
  file1 << "Get Input Container - rc: " << rc << endl;
  if (rc != FMC_OK)
  {
    file1.close();
    return 1;
  }
  file1 << "Input Container Name: " << input.Type() << endl;


/*-- retrieve the output container from the PEA who started the program -*/
  FmcjReadWriteContainer  output;
  rc = FmcjContainer::OutContainer( output );
  file1 << "Get Output Container - rc: " << rc << endl;
  if (rc != FMC_OK)
  {
    file1.close();
    return 1;
  }
  file1 << "Output Container Name: " << output.Type() << endl;
/*----- Modify output values ------------------------------------------*/
  rc= output.SetValue("aFieldInTheOutput",42L);
  file1 << "Setting long value returns rc: " << rc << endl;
  ...


/*-- return the output container to the PEA who started the program -----*/
  rc = FmcjContainer::SetOutContainer( output );
  file1 << "Set Output Container - rc: " << rc << endl;
  FmcjGlobal::Disconnect();
  file1.close();
  return 0;                             // _RC passed to MQSeries Workflow
}
```

```
import com.ibm.workflow.api.*;
// Various needed classes
import java.io.*;
import java.util.*;
public class ActivityImplementation {
  public static PrintWriter out;
  public static void main(String args[])
  {
    try
    { out = new PrintWriter(
                new BufferedWriter(new FileWriter("ActivityImplementation.log")));
    }
    catch (IOException e) {}
    // Maximum nesting depth is 10
    ContainerElement[][] members = new ContainerElement[10][];
    // Maximum element number is 200
    ContainerElement[][] leaves = new ContainerElement[200][];
    String membername;
    String membertype;
    String strvalbuf;
    byte[] binvalbuf = { 0,1,2,3,4,5,6,7,8,9,10 };
    int lvalbuf      = 0;
    double fvalbuf   = 0.0;
    String tInfo;
    int j = 0, k = 0, l = 0;
    int index = 0;
     Calendar ltime = Calendar.getInstance();
  int param = -1;
  int pimreturn = 0;
  String setstr ="";
  int setlong = 0;
  double setdbl = 0;
  ltime.setTime(new Date());
  BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
  boolean again = false;
  String choice;
```

```
/*-------------------------------------------------------------------*/
/* Check command line parameters                                     */
/*-------------------------------------------------------------------*/
  if (args.length!=1)
  {
    System.out.println("¥nUsage: java ActivityImplementation <-r{0|1|2|3|4|5}>¥n"
                       + "       -rx: x specifies the return code of the program¥n¥n"
                       + "       Program output is written to ActivityImplementation.log¥n");
    System.exit(-1);
  }
  try { // general try clause for whole example
  Agent agent = new Agent();
  agent.setLocator(Agent.LOC_LOCATOR);
  agent.setName("LOCAL");
  ExecutionService service = agent.locate("","");
  System.out.println();
  if (args[0].equals("-r0")) param=0;
  if (args[0].equals("-r1")) param=1;
  if (args[0].equals("-r2")) param=2;
  if (args[0].equals("-r3")) param=3;
  if (args[0].equals("-r4")) param=4;
  if (args[0].equals("-r5")) param=5;
  switch (param)
  {
   case -1:
     System.out.println("¥nUsage: java ActivityImplementation <-r{0|1|2|3|4|5}>¥n"
                       + "       -rx: x specifies the return code of the program¥n¥n"
                       + "       Program output is written to ActivityImplementation.log¥n");
     System.exit(-1);
     break;
   case 0: pimreturn = 0; break;
   case 1: pimreturn = 1; break;
   case 2: pimreturn = 2; break;
   case 3: pimreturn = 3; break;
   case 4: pimreturn = 4; break;
   case 5: pimreturn = 5; break;
  }
  out.write("¥n¥n" + "-----------------------------------------------------¥n"
        + "API Tutorial - Activity implementation example program.¥n"
        + "Called at : " + ltime.getTime().toString() + "¥n"
        + "Desired RC: " + pimreturn + "¥n¥n");
```

```
/*********************************************************************/
/* First we cope with the activity's INPUT container:                */
/*  - get the container                                              */
/*  - get the leaves of the container                                */
/*  - display the member values                                      */
/*********************************************************************/
/*----- Get Input Container ------------------------------------------*/
  ReadOnlyContainer inctnr;
  ExecutionAgent eAgent = agent.getExecutionAgent();
  inctnr = eAgent.inContainer();
  out.write("¥nReceived input container '" + inctnr.type() + "'¥n");
  System.out.println("¥nReceived input container '" + inctnr.type() + "'¥n");
/*----- Get Leave Count ----------------------------------------------*/
  int ulLeafCount = inctnr.allLeafCount();
  out.write("Input container AllLeafCount() =  " + ulLeafCount + "¥n");
/*----- Get the Leaves -----------------------------------------------*/
  leaves[0]=inctnr.allLeaves();
  if (ulLeafCount != leaves[0].length)
  {
    out.write("LeafCount - vector size mismatch: "
          + "ulLeafCount = " + ulLeafCount
          + " size = "       + leaves[0].length + "¥n");
  }



/*----- Show the data members ----------------------------------------*/
  out.write("Input container leaves:¥n");
  ContainerElement element = leaves[0][0];
  for (j=0; j < ulLeafCount; j++)
  {
    membername = element.fullName();
    membertype = element.type();
    if (membertype.equals("STRING"))
    {
      try
      {
        strvalbuf = element.getString();
        out.write("STRING '" + membername + "'= " + strvalbuf  + "¥n");
      }
      catch (FmcException e)
      {
       switch (e.rc)
       {
         case FmcException.FMC_ERROR_MEMBER_NOT_SET:
           out.write("STRING '" + membername  + "'= <not set>" + "¥n");
           break;
         default:
           out.write("Failed to access " + membername + " with RC " + e.rc + "¥n");
           break;
       }
      }
    } /* end if STRING */
```

```
if (membertype.equals("LONG"))
{
  try
  {
    lvalbuf = element.getLong();
    out.write("LONG '" + membername + "'= " + lvalbuf  + "¥n");
  }
  catch (FmcException e)
  {
   switch (e.rc)
   {
     case FmcException.FMC_ERROR_MEMBER_NOT_SET:
       out.write("LONG '" + membername  + "'= <not set>" + "¥n");
       break;
     default:
       out.write("Failed to access " + membername + " with RC " + e.rc + "¥n");
       break;
   }
  }
} /* end if LONG */


if (membertype.equals("FLOAT"))
{
  try
  {
    fvalbuf = element.getDouble();
    out.write("FLOAT '" + membername + "'= " + fvalbuf  + "¥n");
  }
  catch (FmcException e)
  {
   switch (e.rc)
   {
     case FmcException.FMC_ERROR_MEMBER_NOT_SET:
       out.write("FLOAT '" + membername  + "'= <not set>" + "¥n");
       break;
     default:
       out.write("Failed to access " + membername + " with RC " + e.rc + "¥n");
       break;
   }
  }
} /* end if FLOAT */
```

```
  if (membertype.equals("BINARY"))
  {
    try
    {
      binvalbuf = element.getBuffer();
      out.write("BINARY '" + membername + "'= <not shown>¥n");
    }
    catch (FmcException e)
    {
     switch (e.rc)
     {
       case FmcException.FMC_ERROR_MEMBER_NOT_SET:
         out.write("BINARY '" + membername  + "'= <not set>" + "¥n");
         break;
       default:
         out.write("Failed to access " + membername + " with RC " + e.rc + "¥n");
         break;
     }
    }
   } /* end if BINARY */
  element = leaves[0][j];
} /* end for */
```

```
/*********************************************************************/
/* Now we process the activity's OUTPUT container, but only if the      */
/* program was not started with the -s switch                           */
/*  - get the container                                                 */
/*  - navigate through the ContainerElement levels to the leaves        */
/*  - modify the member values                                          */
/*  - send the container to the server                                  */
/*********************************************************************/
/*----- Get Output Container ------------------------------------------*/
  ReadWriteContainer outctnr = eAgent.outContainer();
  out.write("Received output container '" + outctnr.type() + "'¥n");
  System.out.println("Received output container '" + outctnr.type() + "'");
/*----- Navigate through the ContainerElement structures ---------------*/
  int level             = 0;  // Current nesting level
  int start             = 0;  // Start "stack pointer" of vectors
  int current           = 0;  // End "stack pointer" of vectors
  boolean AllLeavesReached = false;
  // Get the number of 1st level container elements
  int ulMemberCount = outctnr.memberCount();
  if (ulMemberCount == 0)       // No data structure at all, error?
  {
    AllLeavesReached = true;
  }
  else                          // Get the vector of 1st level container elements
  {
    members[level] = outctnr.structMembers();
  }
  while (AllLeavesReached == false)
  {
    out.write("Number of members: " + ulMemberCount + "¥n");
    // Check consistency
    if (members[level].length != ulMemberCount)
    {
      out.write("MemberCount - Vector Size mismatch: "
            + "ulMemberCount = " + ulMemberCount
            + " Size = "         + members[level].length + "¥n");
    }
    element = members[level][0];
    out.write("IsStruct-IsLeaf-IsArray-Cardinality-Membername¥n");
```

```
for (j=0; j < ulMemberCount; j++)
    {
      out.write("       " + element.isStruct());
      if (element.isStruct())
      {
        // Put the next nesting level of this data structure "on the stack"
        leaves[current] = element.structMembers();
        current++;
      }
      out.write("      " + element.isLeaf() + "       " + element.isArray()
            + "           " + element.cardinality() + " "
            + element.fullName() + "¥n");
      element = members[level][j];
    } /* end for */
    if (start >= current)        // Nothing "on the stack"
    {
      AllLeavesReached = true;
    }
    else
    {
      level++;
      // Get the next vector from the stack and increase stack pointer
      members[level] = leaves[start];
      start++;
      ulMemberCount = members[level].length;
    }
  } /* end while */
```

```
/*----- Modify the data members ----------------------------------------*/
  out.write("¥n¥nSetting the data members...¥n");
  System.out.println("¥nSetting the data members...¥n");
  leaves[0] = outctnr.leaves();
  ulLeafCount = leaves[0].length;
  try
  {
    for (j=0; j < ulLeafCount; j++)
    {
      element = leaves[0][j];
      membername = element.fullName();
      membertype = element.type();
      if (membertype.equals("STRING"))
      {
        if ( membername.endsWith("]") || membername.endsWith(")") )
        // Array element, take the array function
        {
              // Remember the index and remove the suffix from the name
          if (membername.endsWith("]"))
          {
            try
            {
              index=
              Integer.parseInt(membername.substring(membername.indexOf("[")+1,
                               membername.indexOf("]")-1));
            }
            catch (NumberFormatException e) {}
          }


          else
          {
            try
            {
              index=
              Integer.parseInt(membername.substring(membername.indexOf("(")+1,
                               membername.indexOf(")")-1));
            }
            catch (NumberFormatException e) {}
          }
          System.out.print("Enter a value for STRING array member '"
               + membername + "[" + index + "]'? [y/n] ");
          setstr = in.readLine();
```

```
    if (setstr.equalsIgnoreCase("y"))
    {
      System.out.print("Value: ");
      setstr = in.readLine();
      outctnr.setString2(membername,index,setstr);
      strvalbuf = outctnr.getString2(membername,index);
      out.write("Setting ArrayStringValue '" + membername + "[" + index +
                "]'=" + strvalbuf + "¥n");
      System.out.println("Setting ArrayStringValue '" + membername + "[" +
                         index + "]' = " + strvalbuf + "¥n");
    }
  }


  else
  {
    System.out.print("Enter a value for STRING member '"
         + membername + "'? [y/n] ");
    setstr = in.readLine();
    if (setstr.equalsIgnoreCase("y"))
    {
      System.out.print("Value: ");
      setstr = in.readLine();
      outctnr.setString(membername,setstr);
      strvalbuf = outctnr.getString(membername);
      out.write("Setting StringValue '" +
                  membername + "'= " + strvalbuf + "¥n");
      System.out.println("Setting StringValue '" + membername +
                         "'= " + strvalbuf + "¥n");
    }
  }
} /* end if STRING */
```

```
if (membertype.equals("LONG"))
      {
        if ( membername.endsWith("]") || membername.endsWith(")") )
        // Array element, take the array function
        {
          // Remember the index and remove the suffix from the name
          if (membername.endsWith("]"))
          {
            try
            {
              index=
              Integer.parseInt(membername.substring(membername.indexOf("[")+1,
                               membername.indexOf("]")-1));
            }
            catch (NumberFormatException e) {}
          }
          else
          {
            try
            {
              index=
              Integer.parseInt(membername.substring(membername.indexOf("(")+1,
                               membername.indexOf(")")-1));
            }
            catch (NumberFormatException e) {}
          }
          System.out.print("Enter a value for LONG array member '"
               + membername + "[" + index + "]'? [y/n] ");
          setstr = in.readLine();


           if (setstr.equalsIgnoreCase("y"))
          {
            do
            {
              again=false;
              System.out.print("¥nValue: ");
              choice = in.readLine();
              try { setlong = Integer.parseInt(choice); }
              catch (NumberFormatException e) { again=true; }
               } while (again);
            outctnr.setLong2(membername,index,setlong);
            lvalbuf = outctnr.getLong2(membername,index);
            out.write("Setting ArrayLongValue '" + membername + "[" + index +
                      "]' = " + lvalbuf + "¥n");
            System.out.println("Setting ArrayLongValue '" + membername + "[" +
                               index + "]' = " + lvalbuf + "¥n");
          }
        }
```

```
 else
  {
    System.out.print("Enter a value for LONG member '"
         + membername + "'? [y/n] ");
    setstr = in.readLine();
    if (setstr.equalsIgnoreCase("y"))
    {
      do
      {
        again=false;
        System.out.print("¥nValue: ");
        choice = in.readLine();
        try { setlong = Integer.parseInt(choice); }
        catch (NumberFormatException e) { again=true; }
         } while (again);
      outctnr.setLong(membername,setlong);
      lvalbuf = outctnr.getLong(membername);
      out.write("Setting LongValue '" +membername+ "' = " +lvalbuf+ "¥n");
      System.out.println("Setting LongValue '" +membername+"' = " +
                         lvalbuf + "¥n");
    }
  }
} /* end if LONG */
```

```
if (membertype.equals("FLOAT"))
     {
       if ( membername.endsWith("]") || membername.endsWith(")") )
       // Array element, take the array function
       {
              // Remember the index and remove the suffix from the name
         if (membername.endsWith("]"))
         {
           try
           {
             index=
             Integer.parseInt(membername.substring(membername.indexOf("[")+1,
                              membername.indexOf("]")-1));
           }
           catch (NumberFormatException e) {}
         }
         else
         {
           try
           {
             index=
             Integer.parseInt(membername.substring(membername.indexOf("(")+1,
                              membername.indexOf(")")-1));
           }
           catch (NumberFormatException e) {}
         }
         System.out.print("Enter a value for FLOAT array member '"
              + membername + "[" + index + "]'? [y/n] ");
         setstr = in.readLine();


         if (setstr.equalsIgnoreCase("y"))
         {
           do
           {
             again=false;
             System.out.print("¥nValue: ");
             choice = in.readLine();
             try { setdbl = (Double.valueOf(choice)).doubleValue(); }
             catch (NumberFormatException e) { again=true; }
              } while (again);
           outctnr.setDouble2(membername,index,setdbl);
           fvalbuf = outctnr.getDouble2(membername,index);
           out.write("Setting ArrayFloatValue '" + membername + "[" + index +
                     "]' = " + fvalbuf + "¥n");
           System.out.println("Setting ArrayFloatValue '" + membername + "[" +
                              index + "]' = " + fvalbuf + "¥n");
         }
       }
```

```
else
   {
     System.out.print("Enter a value for FLOAT member '"
          + membername + "'? [y/n] ");
     setstr = in.readLine();
     if (setstr.equalsIgnoreCase("y"))
     {
       do
       {
         again=false;
         System.out.print("¥nValue: ");
         choice = in.readLine();
         try { setdbl = (Double.valueOf(choice)).doubleValue(); }
         catch (NumberFormatException e) { again=true; }
          } while (again);
       outctnr.setDouble(membername,setdbl);
       fvalbuf = outctnr.getDouble(membername);
       out.write("Setting FloatValue '" +membername+ "' = " +fvalbuf+ "¥n");
       System.out.println("Setting FloatValue '" + membername + "' = " +
                          fvalbuf + "¥n");
     }
   }
 } /* end if FLOAT */
```

```
if (membertype.equals("BINARY"))
      {
        if ( membername.endsWith("]") || membername.endsWith(")") )
        // Array element, take the array function
        {
               // Remember the index and remove the suffix from the name
          if (membername.endsWith("]"))
          {
            try
            {
              index=
              Integer.parseInt(membername.substring(membername.indexOf("[")+1,
                               membername.indexOf("]")-1));
            }
            catch (NumberFormatException e) {}
          }
          else
          {
            try
            {
              index=
              Integer.parseInt(membername.substring(membername.indexOf("(")+1,
                               membername.indexOf(")")-1));
            }
            catch (NumberFormatException e) {}
          }
          outctnr.setBuffer2(membername,index,binvalbuf);
          binvalbuf = outctnr.getBuffer2(membername,index);
          out.write("Setting ArrayBinaryValue '" + membername + "[" + index +
                    "]' = <not shown>¥n");
          System.out.println("Setting ArrayBinaryValue '" + membername + "[" +
                             index + "]' = <not shown>¥n");
        }



       else
       {
         outctnr.setBuffer(membername,binvalbuf);
         binvalbuf = outctnr.getBuffer(membername);
         out.write("Setting BinaryValue '" + membername + "' = <not shown>¥n");
         System.out.println("Setting BinaryValue '" + membername + "' = <not shown>¥n");
       }
     } /* end if BINARY */
   } /* end for */
  }/* End try*/
  catch (FmcException e)
  {
  }/* End catch*/


/*--------------------------------------------------------------------*/
/* Send output container to PEA                                       */
/*--------------------------------------------------------------------*/
  System.out.println("¥nSetting output container.");
  out.write("¥nSetting output container.¥n");
  eAgent.setOutContainer(outctnr);
```

```
/*-------------------------------------------------------------------*/
/* Logoff and deinit the API environment.                            */
/*-------------------------------------------------------------------*/
  System.out.println("¥nLogging off.");
  out.write("¥nLogging off.¥n");
  service.logoff();
  out.flush();
  out.close();
  System.exit(pimreturn);
  } // end of general try clause


  catch(FmcException e)
  {
    int c;
    // Catch and report details about the FmcException
    System.out.println("FmcException occured");
    System.out.println("  RC         : " + e.rc);
    System.out.println("  Origin     : " + e.origin);
    System.out.println("  MessageText: " + e.messageText);
    System.out.println("  Exception  : " + e.getMessage());
    System.out.println("  Parameters : ");
    for ( c = 0; c < e.parameters.length ; c++)
    {
      System.out.println("    " + e.parameters[c] );
    }
    System.out.println("  StackTrace : ");
    e.printStackTrace();
    out.write("FmcException occured");
    out.write("¥n  RC         : " + e.rc);
    out.write("¥n  Origin     : " + e.origin);
    out.write("¥n  MessageText: " + e.messageText);
    out.write("¥n  Exception  : " + e.getMessage());
    out.write("¥n  Parameters : ");
    for ( c = 0; c < e.parameters.length ; c++)
    {
      out.write("    " + e.parameters[c] );
    }
    out.write("¥n  StackTrace : ");
    e.printStackTrace(out);
    out.flush();
    out.close();
  }
```

```
  catch(Exception e)
  {
    // Catch and report any exception that occurred.
    e.printStackTrace();
    e.printStackTrace(out);
    out.flush();
    out.close();
  }
} // end of void main
} // end of ActivityImplementation
```

第10部 ロータス ノーツ API の使用

この第 10 部では、MQSeries Workflow とロータス ノーツを統合するために用意されている API の概要を示します。

ロータス ノーツ API は C 言語 API です。これは、LotusScript コード内で使うためのものです。 C コードから直接呼び出すこともできます。しかしそれはお勧めしません。ロータス ノーツ API のほとんどの関数には、ロータス ノーツ内でしか使えないパラメーターが必要だからです。

ロータス ノーツ API は、OS/2、Windows NT、および Windows 95/98 プラットフォームでご使用いただけます。

第78章 要件

MQSeries Workflow C++ API をインストールしていなければなりません。

ヘッダー・ファイルとライブラリー・ファイル

ロータス ノーツ API を使って開発されるすべての LotusScript 関数には、ロータス ノーツ API 用の外部 LotusScript ファイルを組み込む必要があります。 MQSeries Workflow のインストール手順において API ファイルのインストールを選択していた場合、それらのファイルはすでにシステムにインストールされています。それらのファイルは、EXMP4API.LSS と EXMP4ARC.LSS であり、省略時には、選択したドライブ上の ¥fmc¥os2¥lnc、fmc¥winnt¥lnc、または ¥fmc¥win95¥lnc サブディレクトリーにインストールされます。

ロータス ノーツ API を使って開発されるすべての C プログラムには、ロータス ノーツ API ヘッダー・ファイルを組み込む必要があります。インストール手順において API ファイルのインストールを選択していた場合、それらのファイルはすでにシステムにインストールされています。それらのファイルは、FMC4API.H、FMC4GLO.H、および FMC4ARC.H であり、省略時には、選択したドライブ上の ¥fmc¥os2¥api、fmc¥winnt¥api、または ¥fmc¥win95¥api サブディレクトリーにインストールされます。アプリケーションには FMC4API.H を組み込まなければなりません。必要な他のヘッダー・ファイルは、すべて FMC4API.H によって暗黙のうちに組み込まれます。

MQSeries Workflow の最初のインストール時にこれらのファイルがインストールされていなかった場合は、*IBM MQSeries Workflow: インストールの手引き* で、そのインストール方法を確認してください。

DLL ファイルと共用ライブラリー・ファイル

MQSeries Workflow DLL ファイルが入れられているディレクトリーが環境設定において指定されていることを確認してください。それは、CONFIG.SYS ファイルの LIBPATH ステートメントです。これは、MQSeries Workflow の標準インストールによって自動的に実行されます。

コンパイル

ロータス ノーツ API 用にサポートされているコンパイラーは次のとおりです。

- OS/2: IBM VisualAge for C++ 3.0
- Windows NT または Windows 95: Microsoft Visual C++ 5.0

これらが必要なのは、ご自分の C プログラムの中でロータス ノーツ API を使う予定の場合のみです。

第79章 コーディング例

ロータス ノーツのデータベースをインストールすると、ファイル FMC4R*xxx*.NTF および FMC4S*xxx*.NTF が提供されます (*xxx* は、たとえば米国英語の場合の ENU のような言語識別子)。 FMC4R*xxx*.NTF は MQSeries Workflow ロータス ノーツのクライアントのデータベース設計テンプレートであり、FMC4S*xxx*.NTF はサンプル・データベース・テンプレートです。この 2 つのテンプレートでは、MQSeries Workflow API 呼び出しおよび戻りコードの定義が含まれている外部 LotusScript ファイル EXMP4API.LSS および EXMP4ARC.LSS を使います。

用意されたテンプレートで何か処理する前に、ロータス ノーツのデータベース・テンプレートをロータス ノーツのデータ・ディレクトリー (通常は NOTES¥DATA) にコピーし、外部 LotusScript ファイルをロータス ノーツ・ディレクトリー (通常は NOTES) にコピーしてください。

データベース設計テンプレート FMC4R*xxx*.NTF は、ロータス ノーツ API を使用する MQSeries Workflow 実行機能クライアントの主な機能を提供します。データベース・テンプレート FMC4S*xxx*.NTF は、主機能の他に、MQSeries Workflow アクティビティーをロータス ノーツにインプリメントする方法と、他の機能を標準機能に追加する方法についてのいくつかのサンプルも提供します。すべてのテンプレートは、ソース・コードで配布されています。それをコーディング例として使ったり、テンプレートを直接変更したりすることができます。 MQSeries Workflow に用意されているファイルを変更する場合は、配慮すべき点がいくつかあります。以下に、それらについて述べます。

サンプル FDL

FMC4S*xxx*.NTF サンプルを使って作業する前に、FMC4SMP.FDL を MQSeries Workflow データベースに搬入し、プロセス NotesCreditRequest を変換してください。

データベース設計の概要

FMC4R*xxx*.NTF および FMC4S*xxx*.NTF は設計テンプレートです。FMC4S*xxx*.NTF は FMC4R*xxx*.NTF から設計を継承します。このテンプレートをカスタマイズしたい場合、次の点に気を付けてください。

- 最低限、ロータス ノーツの開発の経験が必要です。
- テンプレートを変更する場合、データベース・テンプレート名 (IBM_MQSeries_Workflow_XXX_3.1 および IBM_MQSeries_Workflow_XXX_3.1_Sample) も変更してください。
- 変更されていないテンプレートだけがこの製品の一部であり、直接サポートされます。テンプレートを変更する場合は、この変更によって生じるエラーを修正するのはユーザーの責任です。

MQSeries Workflow オブジェクトの設定を表示するためのフォーム

サンプル・テンプレートでは、さまざまな MQSeries Workflow オブジェクトの設定情報を表示するのにいくつかのフォームを使います (多くの場合、サブフォームを使用)。 API 呼び出しによって、MQSeries Workflow オブジェクトの表示用にノーツ文書を作成する場合、その文書中の**「フォーム (Form)」**項目には、それぞれの値が入れられます (955ページの『第91章 ロータス ノーツのクライアントで使用されるフィールド』を参照)。**「ユーザー設定 (User settings)」**、**「プロセス・テンプレート (Process template)」**、**「プロセス・インスタンス (Process instance)」**、**「作業項目 (Work item)」**、**「作業項目通知 (Work-item notification)」**、および**「プロセス・インスタンス通知 (Process-instance notification)」**のフォームには、それぞれのフォームの別名が含まれています。

これらの各フォームには、ビューから設定アクションを使用することになるフォームで文書を表示するための特定の論理が含まれています。文書に保管されている実際のフォーム名は変更されません。

さらに、**「ユーザー設定 (User settings)」**と**「作業項目 (Work item)」**のフォームには、フォームの保管アクション時にそれぞれの更新 API 機能を呼び出すための論理も含まれています。

これらのフォームの見栄えを変更することはできますが、処理ロジックを変更する場合は十分に注意してください。

ダイアログ内で使用するフォーム

特定のアクションから呼び出されるダイアログでは、次のようないくつかのフォームが使われます。

- **「ユーザー設定 - パスワード変更 (User settings - Change password)」**
- **「作業項目 - サポート・ツール開始 (Work item - Start support tool)」**
- **「作業項目 - 転送 (Work item - Transfer)」**

これらのフォームの見栄えを変更することはできますが、処理ロジックを変更する場合は十分に注意してください。

標準オブジェクトの作成に使用するフォーム

元の文書が削除されている場合、**「アプリケーション設定 (Application settings)」**フォームを使って、アプリケーション設定文書を作成することができます。このフォームには、複数のアプリケーション設定文書を作成できないようにするための論理が含まれています。

「ユーザー設定 (User settings)」フォームを使うと、追加のユーザー設定文書を作成することができます。ユーザー設定文書には、MQSeries Workflow システムに接続するのに必要な情報を指定できます。このフォームには、ユーザー ID と MQSeries Workflow システム・グループが同じアプリケーション設定文書を複数作成できないようにするための論理が含まれています。また、ユーザーのワークステーションで MQSeries Workflow プロファイルに定義されている MQSeries Workflow システム・グループだけを表示する論理も含まれています。

「配布 MQSeries Workflow テンプレート (Distribute MQSeries Workflow template)」フォームは、現在のデータベース・テンプレートから新たにデータベースを作成するのに必要な情報を提供するのに使われます。このフォームには、それまでにこのフォームで作成されたすべての文書を削除する論理が入っているので、必要のなくなった文書はすべて削除されます。

これらのフォームの見栄えを変更することはできますが、処理ロジックを変更する場合は十分に注意してください。

MQSeries Workflow プロセス・インスタンスを開始するのに使用するフォーム

「クレジット要求 (Credit Request)」フォームは、開始データを使って MQSeries Workflow プロセスを開始するのに使用します。まず、なんらかのデータを入力するようユーザーに対するプロンプトが表示されます。このフォームには、アクション・ボタン**「クレジット要求の開始 (Start Credit Request)」**が含まれています。このボタンを使って、次のようなアクションを開始できます。

1. 正しいテンプレート文書の検索
2. ロータス ノーツ API を使ったインスタンスの作成
3. インスタンス文書へのアクセス
4. インスタンス文書が見つかった場合に、ロータス ノーツ API によるフォームからのデータの追加とインスタンスの開始

このフォームは、入力データを使って MQSeries Workflow プロセスを開始する方法の例でしかありません。提供されるコードは、すべての起こり得る状況を処理するためのものではなく、最低限必要な論理を示すためのものです。

実稼働環境では、そのようなフォームに機能を追加する必要があるかもしれません。たとえば、文書をスキャンするためのその他のアクションを追加してから、その文書を中央リポジトリーに保管することができます。その情報を指すキーを MQSeries Workflow プロセスに渡せば、その後のアクティビティーのインプリメンテーション時に入力コンテナーを介してそのキーを使うことができます。

MQSeries Workflow アクティビティーのインプリメントに使用するフォーム

ExmWorkitemStart および **ExmWorkitemCheckout** エージェントを使った場合、次のような命名規則を使用して、ロータス ノーツ内でアクティビティーをインプリメントするかどうかが決定されます。すなわち、アクティビティーのインプリメンテーションが *NOTESFORMxyz* の場合、アクティビティーは *xyz* という名前のフォームでインプリメントされているとみなされます。

フォーム **AssessCreditRisk**、**RequestCreditApproval**、**AcceptCredit**、および **RejectCredit** がその例です。これらは、FMC4SMP.FDL の中で定義されている *NotesCreditRequest* プロセスにおいて使われます。

これらのフォームには、入力コンテナーの値と、出力コンテナーを設定するフィールドの値を示すフィールドが含まれています。さらに、次のものも含まれています。

- **「処理済み (Done)」**。これは、ユーザー入力を保管し、作業項目のチェックイン用の API 関数を呼び出します。
- **「保管 - 後で再処理 (Save - Rework later)」**。これはユーザー入力を保管し、文書をクローズします。
- **「取り消し (Undo)」**。これは、作業項目を強制的に再始動するための API 関数を呼び出します。この場合、作業項目のチェックアウトが取り消されます。

これらのフォームは、ロータス ノーツにおける MQSeries Workflow アクティビティーのインプリメンテーションの例です。提供されるコードは、すべての起こり得る状況を処理するためのものではなく、最低限必要な論理を示すためのものです。

実稼働環境では、そのようなフォームにたくさんの機能を追加する必要があるかもしれません。たとえば、スキャン済みの文書など、プロセスに関連した文書を表示するための特定のアクションを追加したり、印刷またはメール発送用の文書を使用できる機能を強化したり、顧客を電話で自動的に呼び出すアクションを追加したりすることができます。

ロータス ノーツ API で使用するビュー

「**MQSeries Workflow API -** *viewname*」という名前のすべてのビューは、ロータス ノーツ API で使用されるものであり、変更しないでください。

エンド・ユーザーが使用するビューとフォルダー

エンド・ユーザーが見るビューとフォルダーには、設定値、プロセス・テンプレート、プロセス・インスタンス、および事前定義列を含む作業項目が示されます。この列は、サイズを変更したり、再ソート用に使用したりできます。ビューの選択は、`_ExmDocType` および `_ExmState` に基づきます。ビューとフォルダーのアクションによって、そのビュー用に選択した文書の種類に適したエージェントが呼び出されます。

エンド・ユーザーに対して示されるビューとフォルダーは、自由に変更できます。フィールドを追加したり、フィールドを省略したり、フィールド順やソートおよびカテゴリー化の基準を変更したりできます。**「作業項目 / アクティビティー別 (Work items/By Activity)」**には、カテゴリー化されたビューの例が示されます。

注: ビューでは、そのビュー内のさまざまな MQSeries Workflow オブジェクトに関連したユーザー ID やデータベースは示されません。ユーザーが複数

の MQSeries Workflow システムにログオンする場合や、いつも同じ MQSeries Workflow システムにログオンするとは限らない場合、この列をビューに追加するのがよいかもしれません。

また、エンド・ユーザーに使わせないようにする必要のあるアクションを除去したり、それらのアクションをエンド・ユーザーのニーズに適したアクションに置き換えたりすることもできます。

エージェント

エージェントは、自分のビューまたはフォームに簡単にアクションを追加する手段です。また、たとえば、プロセス・インスタンスを作成してから、特定の省略時データを使ってそれを開始するような、複数アクションを結合するためのエージェントを作成することもできます。

サンプル・データベース・テンプレートで提供されるエージェントを使うと、エージェントごとに 1 つのロータス ノーツ API 関数を呼び出すことができます。そのエージェントは、現在選択されている (カーソル選択) 文書を処理するよう設計されています。複数選択はサポートされていません。ただし **resync** エージェントは、特定のタイプの文書をすべて処理することができます。用意されているエージェントには、基本的なエラー処理機能が含まれています。

必要に応じてエージェントを変更することはできますが、エージェントは、さまざまなビューおよびフォルダーのアクションからも使われることに注意してください。

ナビゲーター

サンプルに付属の **MQSeries Workflow** ナビゲーターは、さまざまなビューおよびフォルダーをグラフィックに表示します。ホットスポットを 1 つずつクリックするごとに、違うビューやフォルダーが呼び出されます。

必要に合わせてナビゲーターを変更することができます。たとえば、さらにホットスポットを追加したり、既存のホットスポットの背後にあるアクションを変更したりできます。

外観

提供されるサンプル・データベースでは、ロータス ノーツとロータス ノーツ API のいくつかの組み合わせ方法が示されます。まったく異なるさまざまなシナリオが可能です。

以下にいくつかの案を示します。

- **MQSeries Workflow ワークリストをエンド・ユーザーには表示しない**

  基本的なエンド・ユーザー・インターフェースとして、グラフィカル・ナビゲーターを使います。エンド・ユーザーが処理しようとしているアクティビティーの種類ごとに、それぞれ異なるグラフィカル表現を作成します。エンド・ユーザーがホットスポットをクリックしたら、LotusScript を使って、エンド・ユーザーのワークリストから該当するアクティビティーを検索します。もし見つかった場合、それをチェックアウトし、フォーム内に表示します。何も見つからなかった場合、ワークリストをリフレッシュしてもう一度検索します。フォーム内に**「次 (Next)」**アクションを用意します。これは、現在の作業項目をチェックインし、次のものを検索するものです。

- **サーバー・ベースのエージェントを使用する自動化機能を追加する**

  ノーツ・サーバーで実行される時間起動型エージェントが、データベース内のユーザー ID およびパスワード情報にアクセスできるようにする場合、そのエージェントを使うことによって、特定の時刻に MQSeries Workflow に自動的にログオンし、ユーザーのワークリストをリフレッシュしてから、見つかった作業項目を、特別なアルゴリズムに基づいて処理することができます (たとえば、通知のために電子メールを送り、1 人しか割り当てられていない作業項目をすべて自動的にチェックアウトし、特定のフラグを設定した作業項目を対応する文書内にチェックインします)。

- **MQSeries Workflow データベースとアプリケーション・データベースを分離する**

  独自のフォームとエージェントを含む独自のロータス ノーツ・データベースを作成して、それらによって LotusScript を使って MQSeries Workflow データベースにアクセスして、データを検索することができます。その利点は、たとえば、コードが分離されて、MQSeries Workflow がエンド・ユーザーから隠される点にあります。

第80章 制約事項

MQSeries Workflow ロータス ノーツのクライアントには、下記の制約事項が適用されます。

- ロータス ノーツ API によって作成されるノーツ文書内のすべてのフィールド (標準フィールド、入力コンテナー・フィールド、出力コンテナー・フィールド) の合計長は、ロータス ノーツの要約バッファーの最大サイズまでに制限されます。
- 他のユーザーのワークリストの処理はサポートされません。
- ロータス ノーツ API は、ワークリストのリフレッシュではプル・モデルしかサポートしません。つまり、ユーザー (またはプログラム) が他のユーザーによって加えられた変更を見るには、ワークリストを明示的にリフレッシュしなければなりません。これと同じことが、プログラム実行クライアントによって開始された実行中のプログラムの状態変更の場合にも言えます。それは、そのプログラムがロータス ノーツのクライアントから開始されたものであっても同じです。
- 特定の名前は予約されています。
  - 文書内の `_EXM` で始まるすべてのフィールド名は、ロータス ノーツ API 用に予約されています。
  - 文書内の `_Impl` で始まるすべてのフィールド名は、MQSeries Workflow ロータス ノーツのクライアントのデータベース・テンプレート内の特定のインプリメンテーション用に予約されています。

  フィールドについては、955ページの『第91章 ロータス ノーツのクライアントで使用されるフィールド』を参照してください。

第81章 データ型と関数

MQSeries Workflow のデータ型は、次に示すロータス ノーツのデータ型と相互に対応しています。

MQSeries Workflow のデータ型	ロータス ノーツのデータ型
STRING	TEXT
ARRAY of STRING	TEXTLIST
FLOAT	NUMBER
LONG	NUMBER

すべての関数呼び出しが正常に完了したなら、戻りコードは常に 0 です。

API 関数は次のようなグループに分かれています。

一般ノーツ・アクション
一般ノーツ・アクションが使われるのは、MQSeries Workflow でのログオンまたはログオフ、MQSeries Workflow パスワードの変更、ユーザーがログオンしているかどうかの検査、MQSeries Workflow システム・グループのリスト表示、および MQSeries Workflow データベース内のユーザー設定の更新の場合です。

プロセス・テンプレート・アクション
プロセス・テンプレート・アクションは、プロセス・インスタンスの作成に使われます。

プロセス・インスタンス・アクション
プロセス・インスタンス・アクションは、プロセス・インスタンスの削除、再始動、再開、開始、中断、または中止に使われます。

プロセス・インスタンス通知アクション
プロセス・インスタンス通知アクションは、プロセス・インスタンス通知を削除するのに使われます。

作業項目アクション
作業項目アクションが使われるのは、作業項目のチェックインまたはチェックアウト、作業項目の削除、モニター、再始動、開始、中止、転送、サポート・ツールの入手、作業項目用のサポート・ツールの開始、または MQSeries Workflow データベース内の作業項目の属性の更新においてです。

作業項目通知アクション
作業項目通知アクションは、作業項目通知を削除するのに使われます。

複製アクション

複製アクションが使われるのは、セッションごとの MQSeries Workflow ユーザー設定、プロセス・インスタンス、プロセス・インスタンス通知、プロセス・テンプレート、および作業項目の複製、また、作業項目ごとの作業項目入力コンテナーの複製においてです。

第82章 コンテナー・データのマッピング

ここでは、MQSeries Workflow 入力コンテナーと、ロータス ノーツの文書の値との相互のマッピングについて説明します。

ワークフロー・モデルによっては、関数 ExmnStartInstance および ExmnCheckInWorkitem によって MQSeries Workflow コンテナーにデータを入れていなければなりません。さらに、ワークフロー・モデルによっては、関数 ExmnCheckOutWorkitem がロータス ノーツにおいて処理される作業項目用のコンテナー・データを検索します。

開始しようとするプロセス・インスタンスやチェックインしようとする作業項目に入力コンテナーが必要な場合、 MQSeries Workflow コンテナー内のデータ・メンバーと同じ名前のロータス ノーツ項目の値を使用することによって、コンテナーのリーフにデータが入れられます。ロータス ノーツ項目が存在しない場合、そのコンテナー内のどのデータ・メンバーも設定されません。

完全修飾名の例を、図14 に示します。

```
FlowMark data structure          Fully qualified item name
-----------------------          -------------------------
Adress[2]
    Name
       Firstname                 Adress[0].Name.Firstname
       Lastname                  Adress[0].Name.Lastname
    City                         Adress[0].City
    Age                          Adress[0].Age
                                 Adress[1].Name.Firstname
                                 Adress[1].Name.Lastname
                                 Adress[1].City
                                 Adress[1].Age
```

図 14. 完全修飾名の例

第83章 一般的なヒント

1. パラメーターがポインターとして指定されていても、ロータス ノーツ API 関数では、そのポインターが有効なアドレスを指しているかどうかを検査しません。したがって、必ず有効なポインターを使うようにしてください。無効なポインターは、トラップや未定義の動作の原因になることがあります。
2. 関数呼び出しでロータス ノーツのサーバーやロータス ノーツのデータベースを指定する必要があるたびに、その関数は、ロータス ノーツのデータベースに対して読み取りや書き込み (つまり更新) を実行します。ロータス ノーツのデータベースがサーバー上になくてローカルにある場合、ロータス ノーツのサーバー・パラメーターにはヌル値または空文字列を指定します。ロータス ノーツのデータベースが指定されると、関数はロータス ノーツのデータベースを更新します。そのような関数を呼び出す場合は、パフォーマンスに配慮してください。必要でない限り、ロータス ノーツのデータベースを即時に更新することがないようにしてください。

   ロータス ノーツのデータベースが指定されると、関数は次のことを実行します。

   a. MQSeries Workflow 要求を実行する。
   b. 指定された MQSeries Workflow オブジェクトをロータス ノーツのデータベースに複製する。

   この場合、戻りコード EXMN_API_ERROR_REPLICATION_INTERNAL は、MQSeries Workflow 要求は正常に完了したが複製は正常に完了しなかったことを意味します。この場合にデータを複製する必要があるなら、対応する ExmnReplicate 関数を別個に呼び出します。
3. エラー EXMN_API_ERROR_TIMEOUT が発生したなら、関数をやり直してください。エラーがしばしば発生する場合は、MQSeries Workflow のユーザーまたはマシン・プロファイル内の環境変数 APITimeOut を設定または変更してください。

第84章 一般ノーツ・アクション

一般ノーツ・アクションは次のとおりです。

ExmnChangePassword
MQSeries Workflow パスワードの変更

ExmnIsLoggedOn
ユーザーがログオンしているかどうかの検査

ExmnListDatabases
MQSeries Workflow システム・グループのリスト

ExmnLogoff
MQSeries Workflow からのログオフ

ExmnLogon
MQSeries Workflow へのログオン

ExmnUpdateUserSettings
MQSeries Workflow 内のユーザー設定値の更新

MQSeries Workflow パスワードの変更

ExmnChangePassword 関数は、MQSeries Workflow ユーザーのパスワードを変更するのに使います。

```
NAPIRET ExmnChangePassword(char * pszFMUserID,
                           char * pszFMDatabase,
                           char * pszFMOldPassword,
                           char * pszFMNewPassword)
```

図 15. MQSeries Workflow パスワードの変更

パラメーター	I/O	説明
pszFMUserID	I	パスワードを変更する MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow データベースの名前。
pszFMOldPassword	I	現在の MQSeries Workflow ユーザー・パスワード。
pszFMNewPassword	I	新しい MQSeries Workflow ユーザー・パスワード。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_OLD_PASSWORD
現行のパスワードとして指定されたパスワードは正しいものではありません。

EXMN_API_ERROR_PASSWORD
パスワードは MQSeries Workflow のパスワード構文規則に従っていません。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow 内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

ユーザーがログオンしているかどうかの検査

ExmnIsLoggedOn 関数は、ExmnLogon 呼び出しがすでに出されたかどうかを調べるのに使います。

```
NAPIRET ExmnIsLoggedOn(char * pszFMUserID,
                       char * pszFMDatabase)
```

図 16. ログオンの検査

パラメーター	I/O	説明
pszFMUserID	I	ログオン検査の対象の MQSeries Workflow ユーザー ID。

パラメーター	I/O	説明
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。

戻りコード:

EXMN_API_OK
指定されたユーザーはログオンしています。

EXM_API_ERROR_NOT_LOGGED_ON
指定されたユーザーについて ExmnLogon 関数呼び出しは発行されていません。

MQSeries Workflow システム・グループのリスト

ExmnListDatabases 関数は、MQSeries Workflow 構成プロファイルに指定されているすべての MQSeries Workflow システム・グループ・アイテムのリストを調べるのに使います。 MQSeries Workflow にログオンできるようにするには、MQSeries Workflow ユーザーまたはマシン・プロファイルの中で MQSeries Workflow システムを定義する必要があります。

```
NAPIRET ExmnListDatabases(long   lFMDBBufferSize,
                          long * plFMDBNeededBufferSize,
                          char * pszFMDatabaseList)
```

図 17. MQSeries Workflow データベースのリスト

パラメーター	I/O	説明
lFMDBBufferSize	I	*pszFMDatabaseList* に割り振られたメモリーのサイズ。
plFMDBNeededBufferSize	O	このパラメーターは、*pszFMDatabaseList* の実際のバッファー・サイズを戻します。
pszFMDatabaseList	O	構成されているすべての MQSeries Workflow システム・グループのリストを戻します。このリストはヌル文字で終了する文字列で戻されます。その中のシステム・グループ名は 1 つずつドット (·) で区切られています。たとえば、プロファイル内にシステム・グループ EXMDB、EXMDB1、および EXMDB2 を指定している場合、ExmnListDatabases 関数は文字列 `EXMDB.EXMDB1.EXMDB2` を戻します。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR_PROFILE
MQSeries Workflow ユーザーまたはマシン・プロファイル内にグループ・エントリーが見つかりませんでした。

EXMN_API_ERROR_BUFFER_TOO_SMALL
pszFMDatabaseList 用に指定されたバッファーが小さすぎます。

plFMDBNeededBufferSize で戻されたサイズ以上のバッファーを再割り振りしてから、この関数をもう一度実行してください。

MQSeries Workflow からのログオフ

ExmnLogon 関数は、ExmnLogon で確立した MQSeries Workflow セッションを中止するのに使います。

```
NAPIRET ExmnLogoff(char * pszFMUserID,
                   char * pszFMDatabase)
```

図 18. MQSeries Workflow からのログオフ

パラメーター	I/O	説明
pszFMUserID	I	ログオフする MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

MQSeries Workflow へのログオン

ExmnLogon 関数は、MQSeries Workflow にログオンするのに使います。またこの関数は、指定されたロータス ノーツ・データベース内のユーザー設定値の更新も実行します。

```
NAPIRET ExmnLogon(char * pszFMUserID,
                  char * pszFMPassword,
                  char * pszFMDatabase,
                  char * pszNotesServer,
                  char * pszFMUSNotesDB)
```

図 19. MQSeries Workflow へのログオン

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMPassword	I	MQSeries Workflow ユーザー・パスワード。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszNotesServer	I	*pszFMUSNotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMUSNotesDB	I	MQSeries Workflow ユーザー設定値が保管されているロータス ノーツ・データベースの名前。この関数でデータを更新しない場合は、*pszFMUSNotesDB* にヌル・ポインターまたは空文字列を指定します。ロータス ノーツの有効なデータベース名を指定した場合、この関数は、MQSeries Workflow に指定された値を使ってそのデータベースを更新します。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR_USERID_UNKNOWN
未知のユーザー ID。

EXMN_API_ERROR_PASSWORD
パスワードが間違っています。

EXMN_API_ERROR_ALREADY_LOGGED_ON
ユーザーは、指定されたシステム・グループにすでにログオンしています。

EXMN_API_ERROR_PROFILE
システム・グループのプロファイル・データが見つかりません。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_REPLICATION_INTERNAL
MQSeries Workflow に正常にログオンした後、ロータス ノーツのデータベース内のユーザー設定値の複製に失敗しました。

データベースを更新するために、ExmnReplicateFMUserSettings を呼び出してください。

MQSeries Workflow 内のユーザー設定値の更新

ExmnUpdateUserSettings 関数は、ロータス ノーツのデータベースに指定されているフィールドの値を使って MQSeries Workflow ユーザー設定値を更新するのに使います。

```
NAPIRET ExmnUpdateUserSettings(char * pszFMUserId,
                               char * pszFMDatabase,
                               char * pszNotesServer,
                               char * pszFMUSNotesDB)
```

図 20. MQSeries Workflow ユーザー設定値の更新

パラメーター	I/O	説明
pszFMUserId	I	MQSeries Workflow ユーザー ID。

パラメーター	I/O	説明
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszNotesServer	I	データベース *pszFMDatabase* の含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMUSNotesDB	I	MQSeries Workflow ユーザー設定値が保管されているロータス ノーツ・データベースの名前。この関数でデータを更新しない場合は、*pszFMUSNotesDB* にヌル・ポインターまたは空文字列を指定します。ロータス ノーツの有効なデータベースを指定した場合、この関数は、ロータス ノーツのこのデータベースで指定されている値を使って MQSeries Workflow データベースを更新します。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

第85章 プロセス・テンプレート・アクション

プロセス・テンプレート・アクションは次のとおりです。

ExmnCreateInstance
プロセス・インスタンスの作成

プロセス・インスタンスの作成

ExmnCreateInstance 関数は、MQSeries Workflow において新しいプロセス・インスタンスを作成するのに使います。ロータス ノーツのデータベースが指定されている場合、この関数は、複製関数を使うことによって、新しいプロセス・インスタンスでロータス ノーツのデータベースを更新します。

```
NAPIRET ExmnCreateInstance(char * pszFMUserID,
                           char * pszFMDatabase,
                           char * pszTemplateOID,
                           char * pszProcessInstanceName,
                           long   lInstanceOIDBufLen,
                           long * plInstanceOIDLen,
                           char * pszInstanceOID,
                           char * pszNotesServer,
                           char * pszFMPINotesDB,
                           long   lInstanceNIDBufLen,
                           char * pszInstanceNID)
```

図 21. プロセス・インスタンスの作成

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszTemplateOID	I	インスタンスを作成する元となるプロセス・テンプレート・オブジェクトの識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
pszProcessInstanceName	I	作成するプロセス・インスタンスの名前。インスタンス名を入力しない場合は省略時名 *ProcessTemplateNameXXX* (*XXX* はオブジェクト ID) が使われます。
lInstanceOIDBufLen	I	*pszInstanceOID* に割り振るバッファーのサイズ。可能なら 200 バイトにしてください。

パラメーター	I/O	説明
plInstanceOIDLen	O	このパラメーターには、プロセス・インスタンス識別子の実際の長さが入れられます。ExmnCreateInstance 関数から EXMN_API_ERROR_BUFFER_TOO_SMALL が戻された場合は、*plInstanceOIDLen* で戻される長さを使って、*pszInstanceOID* のバッファーを再割り振りしてください。
pszInstanceOID	O	このパラメーターには、新たに作成された MQSeries Workflow プロセス・インスタンスの識別子が入れられます。このバッファーのサイズは、*lInstanceOIDBufLen* に指定したものでなければなりません。
pszNotesServer	I	*pszFMPINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMPINotesDB	I	MQSeries Workflow プロセス・インスタンスのためのロータス ノーツのデータベースの名前。この関数でロータス ノーツのデータを更新しない場合、*pszFMPINotesDB* でヌル・ポインターまたは空文字列を指定します。
lInstanceNIDBufLen	I	*pszInstanceNID* 用に割り振られるバッファーのサイズ (現行リリースのロータス ノーツの場合、8 以上)。
pszInstanceNID	O	新しいプロセス・インスタンス文書の NoteID。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR
解析されたインスタンス・オブジェクトが未定義の位置を参照しています。

EXMN_API_ERROR_BUFFER_TOO_SMALL
pszInstanceOID に割り振られたバッファーが小さすぎます。インスタンスは作成されてロータス ノーツのデータベースに複製されましたが、新しいインスタンスの識別子を戻すことができませんでした。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_NOT_UNIQUE
プロセス・インスタンスの名前が固有ではありません。

EXMN_API_ERROR_WRONG_STATE
この状態では不可能です。

EXMN_API_ERROR_DOES_NOT_EXIST
プロセス・テンプレートが存在していません。

EXMN_API_ERROR_NOT_AUTHORIZED
このメソッドを使う許可がありません。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_EMPTY
プロセス・テンプレートはまだ空のままです。

EXMN_API_ERROR_REPLICATION_INTERNAL
インスタンスの作成は正常に完了しましたが、ロータス ノーツのデータベース内のプロセス・インスタンスの複製に失敗しました。ロータス ノーツのデータベースを更新するため、ExmnReplicateInstances を呼び出してください。

第86章 プロセス・インスタンス・アクション

プロセス・インスタンス・アクションは次のとおりです。

ExmnDeleteInstance
プロセス・インスタンスの削除

ExmnRestartInstance
プロセス・インスタンスの再始動

ExmnResumeInstance
プロセス・インスタンスの再開

ExmnStartInstance
プロセス・インスタンスの開始

ExmnSuspendInstance
プロセス・インスタンスの中断

ExmnTerminateInstance
プロセス・インスタンスの中止

プロセス・インスタンスの削除

ExmnDeleteInstance 関数は、終了または中止しているプロセス・インスタンスを MQSeries Workflow から削除するのに使います。

注: プロセス・インスタンスを削除する許可が MQSeries Workflow によって検査されます。

```
NAPIRET ExmnDeleteInstance(char * pszFMUserID,
                           char * pszFMDatabase,
                           char * pszInstanceOID,
                           char * pszNotesServer,
                           char * pszFMPINotesDB,
                           char * pszInstanceNID)
```

図 22. プロセス・インスタンスの削除

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。

パラメーター	I/O	説明
pszInstanceOID	I	削除するプロセス・インスタンス・オブジェクトの識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
pszNotesServer	I	*pszFMPINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMPINotesDB	I	ロータス ノーツのデータベースを更新する場合、MQSeries Workflow プロセス・インスタンスのためのロータス ノーツのデータベースの名前。このロータス ノーツ・データベースを更新しない場合には、ヌル・ポインターか空文字列を指定してください。
pszInstanceNID	I	削除するプロセス・インスタンス・オブジェクトの識別子 (ロータス ノーツのデータベース内のもの)。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_DOES_NOT_EXIST
作業項目が存在していません。

EXMN_API_ERROR_NOT_AUTHORIZED
このメソッドを使う許可がありません。

EXMN_API_ERROR_WRONG_STATE
この状態では不可能です。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_EMPTY
作業項目オブジェクトはまだ空です。

EXMN_API_ERROR_REPLICATION_INTERNAL
MQSeries Workflow 内のプロセス・インスタンスは削除されましたが、ロータス ノーツのデータベース内の指定されたプロセス・インスタンスの削除に失敗しました。ロータス ノーツのデータベースを更新するため、関数 ExmnReplicateInstances を呼び出してください。

プロセス・インスタンスの再始動

ExmnRestartInstance 関数は、MQSeries Workflow プロセスを再始動するのに使います。再始動できるのは、最上位のプロセスだけです。

```
NAPIRET ExmnRestartInstance(char * pszFMUserID,
                            char * pszFMDatabase,
                            char * pszInstanceOID,
                            char * pszNotesServer,
                            char * pszFMPINotesDB,
                            char * pszInstanceNID)
```

図 23. プロセス・インスタンスの再始動

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszInstanceOID	I	再始動するプロセス・インスタンス・オブジェクトの識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
pszNotesServer	I	*pszFMPINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。

パラメーター	I/O	説明
pszFMPINotesDB	I	MQSeries Workflow プロセス・インスタンスのためのロータス ノーツのデータベースの名前。このロータス ノーツ・データベースを更新しない場合には、ヌル・ポインターか空文字列を指定してください。
pszInstanceNID	I	再始動するプロセス・インスタンス・オブジェクトの識別子 (ロータス ノーツのデータベース内のもの)。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_DOES_NOT_EXIST
作業項目が存在していません。

EXMN_API_ERROR_NOT_AUTHORIZED
このメソッドを使う許可がありません。

EXMN_API_ERROR_WRONG_STATE
この状態では不可能です。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_EMPTY
プロセス・テンプレートはまだ空のままです。

EXMN_API_ERROR_REPLICATION_INTERNAL
メソッドは正常に完了しましたが、ロータス ノーツのデータベース内の指定されたプロセス・インスタンスの複製に失敗しました。ロータス ノーツのデータベースを更新するため、関数 ExmnReplicateInstances を使用してください。

プロセス・インスタンスの再開

ExmnResumeInstance 関数は、中断状態の MQSeries Workflow プロセス・インスタンスの処理を再開するのに使います。

```
NAPIRET ExmnResumeInstance(char * pszFMUserID,
                           char * pszFMDatabase,
                           char * pszInstanceOID,
                           short  sInDeep,
                           char * pszNotesServer,
                           char * pszFMPINotesDB,
                           char * pszInstanceNID)
```

図 24. プロセス・インスタンスの再開

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszInstanceOID	I	再開するプロセス・インスタンスの識別子。**注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
sInDeep	I	*sInDeep* を真 (true) に設定すると、すべてのサブプロセスの処理も再開されます。
pszNotesServer	I	*pszFMPINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMPINotesDB	I	MQSeries Workflow プロセス・インスタンスのためのロータス ノーツのデータベースの名前。このロータス ノーツ・データベースを更新しない場合には、ヌル・ポインターか空文字列を指定してください。
pszInstanceNID	I	再開するプロセス・インスタンス・オブジェクトの識別子 (ロータス ノーツのデータベース内のもの)。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_DOES_NOT_EXIST
作業項目が存在していません。

EXMN_API_ERROR_NOT_AUTHORIZED
このメソッドを使う許可がありません。

EXMN_API_ERROR_WRONG_STATE
この状態では不可能です。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_EMPTY
作業項目オブジェクトはまだ空です。

EXMN_API_ERROR_REPLICATION_INTERNAL
メソッドは正常に完了しましたが、ロータス ノーツのデータベース内の指定されたプロセス・インスタンスの複製に失敗しました。ロータス ノーツのデータベースを更新するため、関数 ExmnReplicateInstances を使用してください。

プロセス・インスタンスの開始

ExmnStartInstance 関数は、MQSeries Workflow プロセス・インスタンスを開始するのに使います。現在ログオンしているユーザーがプロセスの開始者になります。プロセスに入力コンテナーが必要な場合、ExmnStartInstance 関数は、*pszInstanceOID* で指定されているロータス ノーツ文書で提供される情報をその入力コンテナーに入れます。文書フィールドがどのように MQSeries Workflow 入力コンテナーにマッピングされるかについては、893ページの『第82章 コンテナー・データのマッピング』を参照してください。

またこの関数は、ロータス ノーツのデータベース内のプロセス状況の更新も開始します。

```
NAPIRET ExmnStartInstance(char * pszFMUserID,
                          char * pszFMDatabase,
                          char * pszInstanceOID,
                          char * pszNotesServer,
                          char * pszFMPINotesDB,
                          char * pszInstanceNID)
```

図 25. プロセス・インスタンスの開始

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszInstanceOID	I	開始するプロセス・インスタンス・オブジェクトの識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
pszNotesServer	I	*pszFMPINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMPINotesDB	I	MQSeries Workflow プロセス・インスタンスのためのロータス ノーツのデータベースの名前。入力コンテナーが必要な場合、このデータベースの名前を指定する必要があります。このロータス ノーツ・データベースを更新しない場合には、ヌル・ポインターか空文字列を指定してください。

パラメーター	I/O	説明
pszInstanceNID	I	開始するプロセス・インスタンス・オブジェクトの識別子 (ロータス ノーツのデータベース内のもの)。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR
解析されたコンテナー・オブジェクトが未定義の位置を参照しているか、またはコンテナーが空です。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_DOES_NOT_EXIST
作業項目が存在していません。

EXMN_API_ERROR_NOT_AUTHORIZED
このメソッドを使う許可がありません。

EXMN_API_ERROR_MEMBER_NOT_SET
プロセスには、指定されていないロータス ノーツ・データベース内の入力フィールドが必要です。

EXMN_API_ERROR_WRONG_STATE
この状態では不可能です。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_EMPTY
プロセス・テンプレートはまだ空のままです。

EXMN_API_ERROR_REPLICATION_INTERNAL
メソッドは正常に完了しましたが、ロータス ノーツのデータベース内の指定されたプロセス・インスタンスの複製に失敗しました。

プロセス・インスタンスの中断

ExmnSuspendInstance 関数は、MQSeries Workflow プロセス・インスタンスを中断する (つまり一時的に停止する) のに使います。

```
NAPIRET ExmnSuspendInstance(char * pszFMUserID,
                            char * pszFMDatabase,
                            char * pszInstanceOID,
                            short  sInDeep,
                            char * pszNotesServer,
                            char * pszFMPINotesDB,
                            char * pszInstanceNID)
```

図 26. プロセス・インスタンスの中断

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszInstanceOID	I	中断するプロセス・インスタンスの識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
sInDeep	I	*sInDeep* を真 (true) に設定すると、すべてのサブプロセスの処理も中断されます。
pszNotesServer	I	*pszFMPINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMPINotesDB	I	MQSeries Workflow プロセス・インスタンスのためのロータス ノーツのデータベースの名前。このロータス ノーツ・データベースを更新しない場合には、ヌル・ポインターか空文字列を指定してください。

パラメーター	I/O	説明
pszInstanceNID	I	中断するプロセス・インスタンス・オブジェクトの識別子 (ロータス ノーツのデータベース内のもの)。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_DOES_NOT_EXIST
作業項目が存在していません。

EXMN_API_ERROR_NOT_AUTHORIZED
このメソッドを使う許可がありません。

EXMN_API_ERROR_WRONG_STATE
この状態では不可能です。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_EMPTY
作業項目オブジェクトはまだ空です。

EXMN_API_ERROR_REPLICATION_INTERNAL
メソッドは正常に完了しましたが、ロータス ノーツのデータベース内の指定されたプロセス・インスタンスの複製に失敗しました。

プロセス・インスタンスの中止

ExmnTerminateInstance 関数は、MQSeries Workflow プロセス・インスタンスを中止するのに使います。これを実行すると、MQSeries Workflow 実行機能サーバーでプロセス・インスタンスに中止のマークが付けられます。

```
NAPIRET ExmnTerminateInstance(char * pszFMUserID,
                              char * pszFMDatabase,
                              char * pszInstanceOID,
                              char * pszNotesServer,
                              char * pszFMPINotesDB,
                              char * pszInstanceNID)
```

図 27. プロセス・インスタンスの中止

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszInstanceOID	I	中止するプロセス・インスタンス・オブジェクトの識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
pszNotesServer	I	*pszFMPINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMPINotesDB	I	MQSeries Workflow プロセス・インスタンスのためのロータス ノーツのデータベースの名前。このロータス ノーツ・データベースを更新しない場合には、ヌル・ポインターか空文字列を指定してください。
pszInstanceNID	I	中止するプロセス・インスタンス・オブジェクトの識別子 (ロータス ノーツのデータベース内のもの)。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_DOES_NOT_EXIST
作業項目が存在していません。

EXMN_API_ERROR_NOT_AUTHORIZED
このメソッドを使う許可がありません。

EXMN_API_ERROR_WRONG_STATE
この状態では不可能です。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_EMPTY
作業項目オブジェクトはまだ空です。

EXMN_API_ERROR_REPLICATION_INTERNAL
メソッドは正常に完了しましたが、ロータス ノーツのデータベース内の指定されたプロセス・インスタンスの複製に失敗しました。

第87章 プロセス・インスタンス通知アクション

プロセス・インスタンス通知アクションは次のとおりです。

ExmnDeletePINotification
プロセス・インスタンス通知の削除

プロセス・インスタンス通知の削除

ExmnDeletePINotification 関数は、終了したプロセス・インスタンス通知を MQSeries Workflow から削除するのに使います。

```
NAPIRET ExmnDeletePINotification(char * pszFMUserID,
                                 char * pszFMDatabase,
                                 char * pszPINotificationOID,
                                 char * pszNotesServer,
                                 char * pszFMWINotesDB,
                                 char * pszPINotificationNID)
```

図 28. プロセス・インスタンス通知の削除

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszPINotificationOID	I	中止するプロセス・インスタンス通知オブジェクトの識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
pszNotesServer	I	*pszFMWINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMWINotesDB	I	MQSeries Workflow 作業項目のためのロータス ノーツのデータベースの名前。このロータス ノーツ・データベースを更新しない場合には、ヌル・ポインターか空文字列を指定してください。

パラメーター	I/O	説明
pszPINotificationNID	I	削除するプロセス・インスタンス通知オブジェクトの識別子 (ロータス ノーツのデータベース内のもの)。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_DOES_NOT_EXIST
作業項目が存在していません。

EXMN_API_ERROR_NOT_AUTHORIZED
このメソッドを使う許可がありません。

EXMN_API_ERROR_WRONG_STATE
この状態では不可能です。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_EMPTY
作業項目オブジェクトはまだ空です。

EXMN_API_ERROR_REPLICATION_INTERNAL
メソッドは正常に完了しましたが、ロータス ノーツのデータベース内の指定されたプロセス・インスタンスの複製に失敗しました。

第88章 作業項目アクション

作業項目アクションは次のとおりです。

ExmnCheckInWorkitem
作業項目のチェックイン

ExmnCheckOutWorkitem
作業項目のチェックアウト

ExmnDeleteWorkitem
作業項目の削除

ExmnGetSupportTool
サポート・ツールの入手

ExmnManualExitWorkitem
作業項目の手動終了

ExmnRestartWorkitem
作業項目の再始動

ExmnStartSupportTool
作業項目のサポート・ツールの開始

ExmnStartWorkitem
作業項目の開始

ExmnTerminateWorkitem
作業項目の中止

ExmnTransferWorkitem
作業項目の転送

ExmnUpdateWorkitem
作業項目の属性の更新

作業項目のチェックイン

ExmnCheckInWorkitem 関数は、ユーザー処理のためにチェックアウトされた MQSeries Workflow 作業項目をチェックインするのに使います。コンテナー・データが必要な場合、この関数は、ロータス ノーツ文書中の対応する項目によってコンテナーにデータを入れます。文書フィールドがどのように MQSeries

Workflow コンテナーにマッピングされるかについては、893ページの『第82章 コンテナー・データのマッピング』を参照してください。

```
NAPIRET ExmnCheckInWorkitem(char * pszFMUserID,
                            char * pszFMDatabase,
                            char * pszWorkItemOID,
                            long   sReturnCode,
                            char * pszNotesServer,
                            char * pszFMWINotesDB,
                            char * pszWorkitemNID)
```

図 29. 作業項目のチェックイン

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszWorkItemOID	I	チェックインする作業項目オブジェクトの識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
sReturnCode	I	ロータス ノーツのアプリケーションで設定される戻りコード。
pszNotesServer	I	*pszFMWINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMWINotesDB	I	MQSeries Workflow 作業項目のためのロータス ノーツのデータベースの名前。このロータス ノーツ・データベースを更新しない場合には、ヌル・ポインターか空文字列を指定してください。
pszWorkitemNID	I	チェックインする作業項目オブジェクトの識別子 (ロータス ノーツのデータベース内のもの)。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR
pszWorkItemOID は、未定義の位置を参照しているか、値が間違っています。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_DOES_NOT_EXIST
作業項目が存在していません。

EXMN_API_ERROR_NOT_AUTHORIZED
このメソッドを使う許可がありません。

EXMN_API_ERROR_WRONG_STATE
この状態ではチェックインできません。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_EMPTY
作業項目オブジェクトはまだ空です。

EXMN_API_ERROR_REPLICATION_INTERNAL
メソッドは正常に完了しましたが、ロータス ノーツのデータベース内の指定された作業項目の削除に失敗しました。

作業項目のチェックアウト

ExmnCheckOutWorkitem 関数は、ユーザー処理用の MQSeries Workflow 作業項目をチェックアウトするのに使います。つまり、MQSeries Workflow の継承ツール呼び出しメカニズムによっては処理がなされない、ということです。MQSeries Workflow は、処理はユーザー固有の方法で実行されるものとみなします。

ユーザー処理が完了したら、それ以後のワークフロー処理のために作業項目をもう一度チェックインしなければなりません。チェックアウト・作業項目に関連しているコンテナー・データがある場合、ExmnCheckOutWorkitem 関数は、完全修飾名によって対応するロータス ノーツ項目を作成します。コンテナー・データについては、893ページの『第82章 コンテナー・データのマッピング』を参照してください。

```
NAPIRET ExmnCheckOutWorkitem(char * pszFMUserID,
                             char * pszFMDatabase,
                             char * pszWorkItemOID,
                             char * pszNotesServer,
                             char * pszFMWINotesDB,
                             char * pszWorkitemNID)
```

図 30. 作業項目のチェックアウト

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszWorkItemOID	I	チェックアウトする作業項目オブジェクトの識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
pszNotesServer	I	*pszFMWINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMWINotesDB	I	MQSeries Workflow 作業項目のためのロータス ノーツのデータベースの名前。このロータス ノーツ・データベースを更新しない場合には、ヌル・ポインターか空文字列を指定してください。
pszWorkitemNID	I	チェックアウトする作業項目オブジェクトの識別子 (ロータス ノーツのデータベース内のもの)。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR
pszWorkItemOID は、未定義の位置を参照しているか、値が間違っています。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_DOES_NOT_EXIST
作業項目が存在していません。

EXMN_API_ERROR_NOT_AUTHORIZED
このメソッドを使う許可がありません。

EXMN_API_ERROR_WRONG_STATE
この状態ではチェックアウトできません。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_EMPTY
作業項目オブジェクトはまだ空です。

EXMN_API_ERROR_REPLICATION_INTERNAL
メソッドは正常に完了しましたが、ロータス ノーツのデータベース内の指定された作業項目の削除に失敗しました。

作業項目の削除

ExmnDeleteWorkitem 関数は、終了した作業項目を MQSeries Workflow から削除するのに使います。ロータス ノーツのデータベースからも作業項目を削除するため、ロータス ノーツのデータベースを指定します。

```
NAPIRET ExmnDeleteWorkitem(char * pszFMUserID,
                           char * pszFMDatabase,
                           char * pszWorkItemOID,
                           char * pszNotesServer,
                           char * pszFMWINotesDB,
                           char * pszWorkitemNID)
```

図 31. 作業項目の削除

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszWorkItemOID	I	削除する作業項目オブジェクトの識別子。**注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
pszNotesServer	I	*pszFMWINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMWINotesDB	I	MQSeries Workflow 作業項目のためのロータス ノーツのデータベースの名前。このロータス ノーツ・データベースを更新しない場合には、ヌル・ポインターか空文字列を指定してください。
pszWorkitemNID	I	削除する作業項目オブジェクトの識別子 (ロータス ノーツのデータベース内のもの)。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR
pszWorkItemOID は、未定義の位置を参照しているか、値が間違っています。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_DOES_NOT_EXIST
作業項目が存在していません。

EXMN_API_ERROR_NOT_AUTHORIZED
このメソッドを使う許可がありません。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_EMPTY
作業項目オブジェクトはまだ空です。

EXMN_API_ERROR_REPLICATION_INTERNAL
メソッドは正常に完了しましたが、ロータス ノーツのデータベース内の指定された作業項目の削除に失敗しました。

作業項目のためのサポート・ツールの入手

ExmnGetSupportTools 関数は、作業項目用に定義されているサポート・ツールのリストを入手するのに使います。

```
NAPIRET ExmnGetSupportTools(char * pszFMUserID,
                            char * pszFMDatabase,
                            char * pszWorkItemOID,
                            long   lSuppToolsBufLen,
                            long * plSuppToolsLen,
                            char * pszSupportToolList)
```

図 32. 作業項目のためのサポート・ツールの入手

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。

パラメーター	I/O	説明
pszWorkItemOID	I	サポート・ツールのリストを要求する作業項目オブジェクトの識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
lSuppToolsBufLen	O	*pszSupportToolList* に指定されるサポート・ツール・リスト用に割り振られるバッファーの長さ。
plSuppToolsLen	O	このパラメーターには、*pszSupportToolList* で戻される文字列の実際のサイズが入れられます。
pszSupportToolList	O	作業項目のために定義されているサポート・ツールのリストを戻します。このリストは文字列で戻されます。その中のツール名は 1 つずつピリオド (.) で区切られています。たとえば、リストに項目 `Editor` および `Calculator` が入っている場合、文字列は `Editor.Calculator` となります。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR
pszWorkItemOID は、未定義の位置を参照しているか、値が間違っています。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_BUFFER_TOO_SMALL
pszSupportToolList に割り振られたバッファーが小さすぎます。*plSuppToolsLen* で戻されたサイズを使ってバッファーを割り振ってから、この関数をもう一度呼び出してください。

作業項目の手動終了

ExmnManualExitWorkitem 関数は、特定のタイプのプログラム・アクティビティー (手動作業のチェックリストなど) を正常に完了するのに使います。

```
NAPIRET ExmnManualExitWorkitem(char * pszFMUserID,
                               char * pszFMDatabase,
                               char * pszWorkItemOID,
                               char * pszNotesServer,
                               char * pszFMWINotesDB,
                               char * pszWorkitemNID)
```

図 33. 作業項目の手動終了

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszWorkItemOID	I	手動終了する作業項目オブジェクトの識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
pszNotesServer	I	*pFMWINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMWINotesDB	I	MQSeries Workflow 作業項目のためのロータス ノーツのデータベースの名前。このロータス ノーツ・データベースを更新しない場合には、ヌル・ポインターか空文字列を指定してください。
pszWorkitemNID	I	完了する作業項目オブジェクトの識別子 (ロータス ノーツのデータベース内のもの)。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR
pszWorkItemOID は、未定義の位置を参照しているか、値が間違っています。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_DOES_NOT_EXIST
作業項目が存在していません。

EXMN_API_ERROR_NOT_AUTHORIZED
このメソッドを使う許可がありません。

EXMN_API_ERROR_WRONG_STATE
この状態では不可能です。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_EMPTY
作業項目オブジェクトはまだ空です。

EXMN_API_ERROR_REPLICATION_INTERNAL
メソッドは正常に完了しましたが、ロータス ノーツのデータベース内の指定された作業項目の複製に失敗しました。

作業項目の再始動

ExmnRestartWorkitem 関数は、作業項目を Ready 状態に戻して ExmnStartWorkitem 関数を再び呼び出せるようにするために使います。

```
NAPIRET ExmnRestartWorkitem(char * pszFMUserID,
                            char * pszFMDatabase,
                            char * pszWorkItemOID,
                            char * pszNotesServer,
                            char * pszFMWINotesDB,
                            char * pszWorkitemNID)
```

図 34. 作業項目の再始動

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。

パラメーター	I/O	説明
pszWorkItemOID	I	再始動する作業項目の識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
pszNotesServer	I	*pszFMWINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMWINotesDB	I	MQSeries Workflow 作業項目のためのロータス ノーツのデータベースの名前。このロータス ノーツ・データベースを更新しない場合には、ヌル・ポインターか空文字列を指定してください。
pszWorkitemNID	I	再始動する作業項目オブジェクトの識別子 (ロータス ノーツのデータベース内のもの)。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR
pszWorkItemOID は、未定義の位置を参照しているか、値が間違っています。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_DOES_NOT_EXIST
作業項目が存在していません。

EXMN_API_ERROR_NOT_AUTHORIZED
このメソッドを使う許可がありません。

EXMN_API_ERROR_WRONG_STATE
この状態では不可能です。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_EMPTY
作業項目オブジェクトはまだ空です。

EXMN_API_ERROR_REPLICATION_INTERNAL
メソッドは正常に完了しましたが、ロータス ノーツのデータベース内の指定された作業項目の複製に失敗しました。

サポート・ツールの開始

ExmnStartSupportTool 関数は、作業項目に関連するサポート・ツールを開始するのに使います。

```
NAPIRET ExmnStartSupportTool(char * pszFMUserID,
                             char * pszFMDatabase,
                             char * pszWorkItemOID,
                             char * pszSupportTool)
```

図 35. サポート・ツールの開始

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszWorkItemOID	I	サポート・ツールを開始する作業項目の識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
pszSupportTool	I	開始するサポート・ツールの名前。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR
pszWorkItemOID は、未定義の位置を参照しているか、値が間違っています。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

作業項目の開始

ExmnStartWorkitem 関数は、作業項目の表すアクティビティーに割り当てられたインプリメンテーション・ツールまたはプロセスを開始します。そのツールは、MQSeries Workflow のツール呼び出しメカニズムにより開始されます。作業項目の新しい状態をロータス ノーツのデータベースに複製するため、ロータス ノーツのデータベースを指定します。

```
NAPIRET ExmnStartWorkitem(char * pszFMUserID,
                          char * pszFMDatabase,
                          char * pszWorkItemOID,
                          char * pszNotesServer,
                          char * pszFMWINotesDB,
                          char * pszWorkitemNID)
```

図 36. 作業項目の開始

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszWorkItemOID	I	開始する作業項目の識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
pszNotesServer	I	*pszFMWINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMWINotesDB	I	MQSeries Workflow 作業項目のためのロータス ノーツのデータベースの名前。このロータス ノーツ・データベースを更新しない場合には、ヌル・ポインターか空文字列を指定してください。
pszWorkitemNID	I	開始する作業項目オブジェクトの識別子 (ロータス ノーツのデータベース内のもの)。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR
pszWorkItemOID は、未定義の位置を参照しているか、値が間違っています。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_DOES_NOT_EXIST
作業項目が存在していません。

EXMN_API_ERROR_NOT_AUTHORIZED
このメソッドを使う許可がありません。

EXMN_API_ERROR_WRONG_STATE
この状態では不可能です。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_EMPTY
作業項目オブジェクトはまだ空です。

EXMN_API_ERROR_REPLICATION_INTERNAL
メソッドは正常に完了しましたが、ロータス ノーツのデータベース内の指定された作業項目の複製に失敗しました。

作業項目の中止

ExmnTerminateWorkitem 関数は、MQSeries Workflow 作業項目を中止するのに使います。

```
NAPIRET ExmnTerminateWorkitem(char * pszFMUserID,
                              char * pszFMDatabase,
                              char * pszWorkItemOID,
                              char * pszNotesServer,
                              char * pszFMWINotesDB,
                              char * pszWorkitemNID)
```

図 37. 作業項目の中止

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszWorkItemOID	I	中止する作業項目の識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
pszNotesServer	I	*pszFMWINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMWINotesDB	I	MQSeries Workflow 作業項目のためのロータス ノーツのデータベースの名前。このロータス ノーツ・データベースを更新しない場合には、ヌル・ポインターか空文字列を指定してください。
pszWorkitemNID	I	中止する作業項目オブジェクトの識別子 (ロータス ノーツのデータベース内のもの)。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR
pszWorkItemOID は、未定義の位置を参照しているか、値が間違っています。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_DOES_NOT_EXIST
作業項目が存在していません。

EXMN_API_ERROR_NOT_AUTHORIZED
このメソッドを使う許可がありません。

EXMN_API_ERROR_WRONG_STATE
この状態では不可能です。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_EMPTY
作業項目オブジェクトはまだ空です。

EXMN_API_ERROR_REPLICATION_INTERNAL
メソッドは正常に完了しましたが、ロータス ノーツのデータベース内の指定された作業項目の複製に失敗しました。

作業項目の転送

ExmnTransferWorkitem 関数は、作業項目を他の MQSeries Workflow ユーザーに転送するのに使います。

```
NAPIRET ExmnTransferWorkitem(char * pszFMUserID,
                             char * pszFMDatabase,
                             char * pszWorkItemOID,
                             char * pszUserID,
                             char * pszNotesServer,
                             char * pszFMWINotesDB,
                             char * pszWorkitemNID)
```

図 38. 作業項目の転送

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszWorkItemOID	I	転送する作業項目の識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
pszUserID	I	作業項目の転送先となるユーザーのユーザー ID。
pszNotesServer	I	*pszFMWINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMWINotesDB	I	MQSeries Workflow 作業項目のためのロータス ノーツのデータベースの名前。このロータス ノーツ・データベースを更新しない場合には、ヌル・ポインターか空文字列を指定してください。
pszWorkitemNID	I	転送する作業項目オブジェクトの識別子 (ロータス ノーツのデータベース内のもの)。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR
pszWorkItemOID は、未定義の位置を参照しているか、値が間違っています。

EXMN_API_ERROR_USERID_UNKNOWN
未知のユーザー ID。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_DOES_NOT_EXIST
作業項目が存在していません。

EXMN_API_ERROR_NOT_AUTHORIZED
このメソッドを使う許可がありません。

EXMN_API_ERROR_WRONG_STATE
この状態では不可能です。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_EMPTY
作業項目オブジェクトはまだ空です。

EXMN_API_ERROR_REPLICATION_INTERNAL
メソッドは正常に完了しましたが、ロータス ノーツのデータベース内の指定された作業項目の複製に失敗しました。

作業項目の更新

ExmnUpdateWorkitem 関数は、ロータス ノーツのデータベースに指定されているフィールドの値を使って MQSeries Workflow 作業項目を更新するのに使います。

```
NAPIRET ExmnUpdateWorkitem(char * pszFMUserId,
                           char * pszFMDatabase,
                           char * pszWorkitemOId,
                           char * pszNotesServer,
                           char * pszFMWINotesDB,
                           char * pszWorkitemNID)
```

図 39. MQSeries Workflow での作業項目の更新

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszWorkItemOID	I	更新する作業項目の識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。

パラメーター	I/O	説明
pszNotesServer	I	*pszFMWINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMWINotesDB	I	MQSeries Workflow 作業項目が保管されているロータス ノーツ・データベースの名前。
pszWorkitemNID	I	更新する作業項目オブジェクトの識別子 (ロータス ノーツのデータベース内のもの)。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR
pszWorkItemOID は、未定義の位置を参照しているか、値が間違っています。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

第89章 作業項目通知アクション

作業項目通知アクション:

ExmnDeleteWINotification
作業項目の通知の削除

作業項目の通知の削除

ExmnDeleteWINotification 関数は、終了した作業項目通知を MQSeries Workflow から削除するのに使います。

```
NAPIRET ExmnDeleteWINotification(char * pszFMUserID,
                                 char * pszFMDatabase,
                                 char * pszWINotificationOID,
                                 char * pszNotesServer,
                                 char * pszFMWINotesDB,
                                 char * pszWINotificationNID)
```

図 40. 作業項目の通知の削除

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszWINotificationOID	I	削除する作業項目通知オブジェクトの識別子。 **注:** この値は、ロータス ノーツ文書の _ExmDocObjectId 項目から入手できます。
pszNotesServer	I	*pszFMWINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMWINotesDB	I	MQSeries Workflow 作業項目のためのロータス ノーツのデータベースの名前。このロータス ノーツ・データベースを更新しない場合には、ヌル・ポインターか空文字列を指定してください。

パラメーター	I/O	説明
pszWINotificationNID	I	削除する作業項目通知オブジェクトの識別子 (ロータス ノーツのデータベース内のもの)。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

EXMN_API_ERROR_DOES_NOT_EXIST
作業項目が存在していません。

EXMN_API_ERROR_NOT_AUTHORIZED
このメソッドを使う許可がありません。

EXMN_API_ERROR_WRONG_STATE
この状態では不可能です。

EXMN_API_ERROR_TIMEOUT
タイムアウトになりました。

EXMN_API_ERROR_SERVER
サーバーが使用可能でないか、または未知です。

EXMN_API_ERROR_INTERNAL
MQSeries Workflow の内部エラーが発生しました。 IBM 担当員に連絡してください。

EXMN_API_ERROR_RESOURCE
内部リソース問題。

EXMN_API_ERROR_MESSAGE_FORMAT
内部メッセージの形式エラーです。

EXMN_API_ERROR_EMPTY
作業項目オブジェクトはまだ空です。

第90章 複製アクション

複製関数は、指定された MQSeries Workflow オブジェクトを MQSeries Workflow データベースから読み取り、ロータス ノーツのデータベースを同期化します。戻りコード EXMN_API_OK が発行された場合は、データベースが同期化されていることを意味します。その場合、必ずしもロータス ノーツのデータベースが更新されたということではありません。

複製アクションは次のとおりです。

ExmnReplicateFMUserSettings
セッションごとの MQSeries Workflow ユーザー設定値の複製

ExmnReplicateInstances
セッションごとのプロセス・インスタンスの複製

ExmnReplicatePINotification
セッションごとのプロセス・インスタンス通知の複製

ExmnReplicateTemplates
セッションごとのプロセス・テンプレートの複製

ExmnReplicateWINotification
セッションごとの作業項目通知の複製

ExmnReplicateWorkitems
セッションごとの作業項目の複製

セッションごとの MQSeries Workflow ユーザー設定値の複製

ExmnReplicateFMUserSettings 関数は、ユーザー設定値をロータス ノーツのデータベースに複製するのに使います。

```
ExmnReplicateFMUserSettings(char * pszFMUserID,
                            char * pszFMDatabase,
                            char * pszNotesServer,
                            char * pszFMUSNotesDB)
```

図 41. セッションごとの FMUserSettings の複製

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。

パラメーター	I/O	説明
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
pszNotesServer	I	*pszFMUSNotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMUSNotesDB	I	MQSeries Workflow ユーザー設定値のためのロータス ノーツのデータベースの名前。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

セッションごとのプロセス・インスタンスの複製

ExmnReplicateInstances 関数は、指定したフィルター基準を満たすすべての MQSeries Workflow プロセス・インスタンス・オブジェクトを、ロータス ノーツのデータベースに複製する場合に使います。

```
NAPIRET ExmnReplicateInstances(char *       pszFMUserID,
                               char *       pszFMDatabase,
                               ExmnReplFlag fReplicateFlag,
                               char *       pszInstanceOIDList,
                               char *       pszLastReplicationDate,
                               char *       pszNotesServer,
                               char *       pszFMPINotesDB,
                               char *       pszInstanceNIDList)
```

図 42. セッションごとのプロセス・インスタンスの複製

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。

パラメーター	I/O	説明
fReplicateFlag	I	フィルター・フラグをセットします。以下のいずれかです。 **ReplAll** 指定されたロータス ノーツ・データベースに、すべてのプロセス・インスタンスを複製します。*pszInstanceOIDList* と *pszLastReplicationDate* は無視されます。 **ReplObjectList** *pszInstanceOIDList* と *pszInstanceNIDList* の中にオブジェクト ID が示されているプロセス・インスタンスだけを複製します。 **ReplLastDate** 予約済み。
pszInstanceOIDList	I	*fReplicateFlag* が ReplObjectList に設定されている場合にリフレッシュされるオブジェクトの MQSeries Workflow ID のリスト (ピリオドで区切る)。
pszLastReplicationDate	I	ヌルまたは空文字列を指定します。
pszNotesServer	I	*pszFMPINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMPINotesDB	I	MQSeries Workflow プロセス・インスタンスのためのロータス ノーツのデータベースの名前。
pszInstanceNIDList	I	*fReplicateFlag* が ReplObjectList に設定されている場合にリフレッシュされるオブジェクトのロータス ノーツ ID のリスト。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR
pszWorkItemOID は、未定義の位置を参照しているか、値が間違っています。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

セッションごとのプロセス・インスタンス通知の複製

ExmnReplicatePINotification 関数は、指定したフィルター基準を満たすすべての MQSeries Workflow プロセス・インスタンス通知オブジェクトを、ロータス ノーツのデータベースに複製する場合に使います。

```
NAPIRET ExmnReplicatePINotification(char *       pszFMUserID,
                                    char *       pszFMDatabase,
                                    ExmnReplFlag fReplicateFlag,
                                    char *       pszPINotificationOIDList,
                                    char *       pszLastReplicationDate,
                                    char *       pszNotesServer,
                                    char *       pszFMWINotesDB,
                                    char *       pszPINotificationNIDList)
```

図 43. セッションごとのプロセス・インスタンス通知の複製

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。

パラメーター	I/O	説明
fReplicateFlag	I	フィルター・フラグをセットします。以下のいずれかです。 **ReplAll** 指定されたロータス ノーツ・データベースに、すべてのプロセス・インスタンス通知を複製します。*pszPINotificationOIDList* と *pszLastReplicationDate* は無視されます。 **ReplObjectList** *pszPINotificationOIDList* と *pszPINotificationNIDList* の中にオブジェクト ID が示されているプロセス・インスタンス通知だけを複製します。 **ReplLastDate** 予約済み。
pszPINotificationOIDList	I	*fReplicateFlag* が ReplObjectList に設定されている場合にリフレッシュされるオブジェクトの ID のリスト (ピリオドで区切る)。
pszLastReplicationDate	I	ヌルまたは空文字列を指定します。
pszNotesServer	I	*pszFMWINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMWINotesDB	I	MQSeries Workflow プロセス・インスタンス通知のためのロータス ノーツのデータベースの名前。
pszInstanceNIDList	I	*fReplicateFlag* が ReplObjectList に設定されている場合にリフレッシュされるオブジェクトのロータス ノーツ ID のリスト。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

セッションごとのプロセス・テンプレートの複製

ExmnReplicateTemplates 関数は、指定したフィルター基準を満たすすべてのプロセス・テンプレートを、ロータス ノーツのデータベースに複製する場合に使います。

```
NAPIRET ExmnReplicateTemplates(char *       pszFMUserID,
                               char *       pszFMDatabase,
                               ExmnReplFlag fReplicateFlag,
                               char *       pszTemplateOIDList,
                               char *       pszLastReplicationDate,
                               char *       pszNotesServer,
                               char *       pszFMPTNotesDB,
                               char *       pszTemplateNIDList)
```

図 44. セッションごとのプロセス・テンプレートの複製

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
fReplicateFlag	I	フィルター・フラグをセットします。以下のいずれかです。 **ReplAll** 指定されたロータス ノーツ・データベースに、すべてのプロセス・テンプレートを複製します。*pszTemplateOIDList* と *pszLastReplicationDate* は無視されます。 **ReplObjectList** *pszTemplateOIDList* と *pszTemplateNIDList* の中にオブジェクト ID が示されているプロセス・テンプレートだけを複製します。 **ReplLastDate** 予約済み。
pszTemplateOIDList	I	*fReplicateFlag* が ReplObjectList に設定されている場合にリフレッシュされるオブジェクトの ID のリスト (ピリオドで区切る)。
pszLastReplicationDate	I	ヌルまたは空文字列を指定します。

パラメーター	I/O	説明
pszNotesServer	I	*pszFMPTNotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMPTNotesDB	I	MQSeries Workflow プロセス・テンプレートのためのロータス ノーツのデータベースの名前。
pszTemplateNIDList	I	*fReplicateFlag* が ReplObjectList に設定されている場合にリフレッシュされるオブジェクトのロータス ノーツ ID のリスト。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

セッションごとの作業項目通知の複製

ExmnReplicateWINotification 関数は、指定したフィルター基準を満たすすべての作業項目通知オブジェクトを、ロータス ノーツのデータベースに複製する場合に使います。

```
NAPIRET ExmnReplicateWINotification(char *       pszFMUserID,
                                    char *       pszFMDatabase,
                                    ExmnReplFlag fReplicateFlag,
                                    char *       pszWINotificationOIDList,
                                    char *       pszLastReplicationDate,
                                    char *       pszNotesServer,
                                    char *       pszFMWINotesDB,
                                    char *       pszWINotificationNIDList)
```

図 45. セッションごとの作業項目通知の複製

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。

パラメーター	I/O	説明
fReplicateFlag	I	フィルター・フラグをセットします。以下のいずれかです。 **ReplAll** 指定されたロータス ノーツ・データベースに、すべての作業項目通知を複製します。*pszWINotificationOIDList* と *pszLastReplicationDate* は無視されます。 **ReplObjectList** *pszWINotificationOIDList* と *pszWINotificationNIDList* の中にオブジェクト ID が示されている作業項目通知だけを複製します。 **ReplLastDate** 予約済み。
pszWINotificationOIDList	I	*fReplicateFlag* が ReplObjectList に設定されている場合にリフレッシュされるオブジェクトのリスト (ピリオドで区切る)。
pszLastReplicationDate	I	ヌルまたは空文字列を指定します。
pszNotesServer	I	*pszFMWINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMWINotesDB	I	MQSeries Workflow 作業項目のためのロータス ノーツのデータベースの名前。
pszWINotificationNIDList	I	*fReplicateFlag* が ReplObjectList に設定されている場合にリフレッシュされるオブジェクトのロータス ノーツ ID のリスト。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

セッションごとの作業項目の複製

ExmnReplicateWorkitems 関数は、指定したフィルター基準を満たすすべての作業項目オブジェクトを、ロータス ノーツのデータベースに複製する場合に使います。

```
NAPIRET ExmnReplicateWorkitems(char *       pszFMUserID,
                               char *       pszFMDatabase,
                               ExmnReplFlag fReplicateFlag,
                               char *       pszWorkitemOIDList,
                               char *       pszLastReplicationDate,
                               char *       pszNotesServer,
                               char *       pszFMWINotesDB,
                               char *       pszWorkitemNIDList)
```

図 46. セッションごとの作業項目の複製

パラメーター	I/O	説明
pszFMUserID	I	MQSeries Workflow ユーザー ID。
pszFMDatabase	I	MQSeries Workflow システム・グループの名前。
fReplicateFlag	I	フィルター・フラグをセットします。以下のいずれかです。 **ReplAll** 指定されたロータス ノーツ・データベースに、すべての作業項目を複製します。*pszWorkitemOIDList* と *pszLastReplicationDate* は無視されます。 **ReplObjectList** *pszWorkitemOIDList* と *pszWorkitemNIDList* の中にオブジェクト ID が示されている作業項目だけを複製します。 **ReplLastDate** 予約済み。
pszWorkitemOIDList	I	*fReplicateFlag* が ReplObjectList に設定されている場合にリフレッシュされるオブジェクトのリスト (ピリオドで区切る)。
pszLastReplicationDate	I	ヌルまたは空文字列を指定します。

パラメーター	I/O	説明
pszNotesServer	I	*pszFMWINotesDB* で指定されているデータベースが含まれているロータス ノーツのサーバーの名前。データベースがローカルである場合、ヌル・ポインターまたは空文字列を指定します。
pszFMWINotesDB	I	MQSeries Workflow 作業項目のためのロータス ノーツのデータベースの名前。
pszWorkitemNIDList	I	*fReplicateFlag* が ReplObjectList に設定されている場合にリフレッシュされるオブジェクトのロータス ノーツ ID のリスト。

戻りコード:

EXMN_API_OK
メソッドは正常に完了しました。

EXMN_API_ERROR_NOT_LOGGED_ON
指定したユーザーのセッションが確立されていません。

第91章 ロータス ノーツのクライアントで使用されるフィールド

下記の表において、MQSeries Workflow のロータス ノーツクライアントに提供されるフォームで使用されるフィールドについて説明します。これらの表の**設定**欄は、その項目が照会時に直ちに設定される (I) か、それともリフレッシュ後に初めて設定される (R) かを示しています。

アプリケーション設定値

表 4. アプリケーション設定値

フィールド名	設定	データ型	フィールドの説明と内容
ロータス ノーツ API のフィールド			
FORM		Text	“ExmAs” に設定。
_ExmDocType	I	Text	“ExmAs” に設定。
インプリメンテーション固有のフィールド			
_ImplSuppressNotLoggedOn		Text	次のいずれか。 **値** **説明** **0** No **1** Yes
_ImplOriginalForm		Text	

ユーザー設定値

表 5. ユーザー設定値

フィールド名	設定	データ型	フィールドの説明と内容
ロータス ノーツ API のフィールド			
FORM		Text	“ExmUs” に設定。
_ExmDocType	I	Text	“ExmUs” に設定。

表 5. ユーザー設定値 (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmUserId	I	Text	ログオン・ユーザーの MQSeries Workflow ユーザー ID。 ユーザー ID の長さは 1〜32 バイト。
_ExmDatabase	I	Text	MQSeries Workflow システム・グループの名前。 名前の長さは 1〜32 バイト。
_ExmPersonId	I	Text	ログオン・ユーザーの担当者 ID。 担当者 ID の長さは 0〜32 バイト。
_ExmFirstName	I	Text	ログオン・ユーザーのファーストネーム。 名前の長さは 0〜32 バイト。
_ExmMiddleName	I	Text	ログオン・ユーザーのミドルネーム。 名前の長さは 0〜32 バイト。
_ExmLastName	I	Text	ログオン・ユーザーのラストネーム。 名前の長さは 0〜32 バイト。
_ExmPhone1	I	Text	ログオン・ユーザーの電話番号。 0〜32 バイト。
_ExmPhone2	I	Text	ログオン・ユーザーの代替電話番号。 0〜32 バイト。
_ExmLevel	I	Text	ログオン・ユーザーのレベル。 0〜9 の数値。

表 5. ユーザー設定値 (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmAbsent	I	Text	ログオン・ユーザーが不在と宣言されているかどうか。更新 API 関数により MQSeries Workflow で変更可。値は次のいずれか。 **値** **説明** **0** 不在でない **1** 不在
_ExmDeleteFinished	I	Text	終了した項目を MQSeries Workflow データベースから削除するかどうか。更新 API 関数により MQSeries Workflow で変更可。値は次のいずれか。 **値** **説明** **0** No **1** Yes
_ExmManager	I	Text	ログオン・ユーザーのマネージャー。 0〜32 バイト。
_ExmOrganization	I	Text	ログオン・ユーザーの属する組織。 0〜32 バイト。
_ExmOrgManager	I	Text	ログオン・ユーザーに役割マネージャーが指定されているかどうか。 値は次のいずれか。 **値** **説明** **0** No **1** Yes
_ExmRolesAssigned	I	Text	ログオン・ユーザーの役割。 複数値フィールド。 0〜32 バイト。

表 5. ユーザー設定値 (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmRolesCoordinating	I	Text	ログオン・ユーザーが調整する役割。 複数値フィールド。 0〜32 バイト。
_ExmAuthUsers	I	Text	ログオン・ユーザーが作業項目にアクセスできる担当者のリスト。 複数値フィールド。 0〜32 バイト。
_ExmAuthAllUsers	I	Text	ログオン・ユーザーに他のユーザーのアクティビティーへのアクセス権が付与されているかどうか。 値は次のいずれか。 **値** **説明** **0** No **1** Yes
_ExmAuthAccessPermitted	I	Text	ログオン・ユーザーの作業項目にアクセスできる担当者のリスト。 複数値フィールド。 0〜32 バイト。
_ExmAuthStaff	I	Text	ログオン・ユーザーにスタッフを定義する許可があるかどうか。 値は次のいずれか。 **値** **説明** **0** No **1** Yes

表 5. ユーザー設定値 (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmAuthOther	I	Text	ログオン・ユーザーにプロセスのモデル定義が許可されているかどうか。 値は次のいずれか。 **値** **説明** **0** No **1** Yes
_ExmAuthCategories	I	Text	ログオン・ユーザーに基本許可が付与されているカテゴリー。 複数値フィールド。 0〜32 バイト。
_ExmAuthCategoriesAdmin	I	Text	ログオン・ユーザーに管理者許可が付与されているカテゴリー。 複数値フィールド。 0〜32 バイト。
_ExmAuthAllCategoriesAdmin	I	Text	ログオン・ユーザーにすべてのカテゴリーのプロセス・インスタンスを管理することが許可されているかどうか。 値は次のいずれか。 **値** **説明** **0** No **1** Yes
_ExmOwnSubstitute	I	Text	ログオン・ユーザーの代行者となるユーザーについての情報。 0〜32 バイト。
_ExmUsersSubstituted	I	Text	ログオン・ユーザーが代行者となっている担当者。 複数値フィールドで、更新 API 関数により MQSeries Workflow で変更可。 0〜32 バイト。

表 5. ユーザー設定値 (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmDescription	I	Text	ログオン・ユーザーについての説明。 0〜1 024 バイト。
インプリメンテーション固有のフィールド			
_ImplConnectionState		Text	値は次のいずれか。 **値　説明** **'0'**　切断 **'1'**　接続
_ImplPassword		Text	このフィールドは暗号化されています。 0〜32 バイト。
_ImplPasswordVerify		Text	このフィールドは暗号化されています。 0〜32 バイト。
_ImplOriginalForm		Text	
_ImplDatabaseList		Text	複数値フィールド。
_ImplAbsentDisp		Keyword	
_ImplDeleteFinishedDisp		Keyword	

プロセス・インスタンス

表 6. プロセス・テンプレート

フィールド名	設定	データ型	フィールドの説明と内容
ロータス ノーツ API のフィールド			
FORM		Text	“ExmPi” に設定。
_ExmDocType	I	Text	“ExmPi” に設定。
_ExmDocObjectId	I	Text	1〜n バイト。
_ExmDatabase	I	Text	MQSeries Workflow システム・グループの名前。 名前の長さは 1〜8 バイト。

表 6. プロセス・テンプレート (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmUserId	I	Text	担当者の MQSeries Workflow ユーザー ID。 ユーザー ID の長さは 1〜32 バイト。
_ExmName	I	Text	プロセス・インスタンスの名前。 1〜63 バイト。
_ExmDescription	I	Text	プロセス・インスタンスについての説明。 0〜254 バイト。
_ExmCategory	I	Text	プロセス・インスタンスのカテゴリー。 0〜32 バイト。
_ExmState	I	Text	プロセス・インスタンスの状況。 値は次のいずれか。 **値　説明** **'0'** Undefined **'1'** Ready **'2'** Running **'4'** Finished **'8'** Terminated **'16'** Suspended
_ExmParentName	I	Text	この通知に関連したプロセス・インスタンスの上位プロセスの名前。 0〜63 バイト。
_ExmTopLevelName	I	Text	この通知に関連したプロセス・インスタンスの最上位プロセスの名前。 1〜63 バイト。

表 6. プロセス・テンプレート (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmAudit	R	Text	このプロセス・インスタンスについて監査ログを作成するかどうか。 値は次のいずれか。 **値** **説明** **'0'** No **'1'** Yes
_ExmTerminateOnError	R	Text	終了条件または分岐条件の評価時にエラーが発生した場合に、プロセス・インスタンスを中止するかどうか。 値は次のいずれか。 **値** **説明** **'0'** No **'1'** Yes
_ExmInContainerNeeded	I	Text	このプロセス・インスタンスに入力データが必要かどうか。 値は次のいずれか。 **値** **説明** **'0'** No **'1'** Yes
_ExmInContainerName	R	Text	プロセス・インスタンスの入力コンテナー・データ構造体の名前。 1〜32 バイト。
_ExmOutContainerName	R	Text	プロセス・インスタンスの出力コンテナー・データ構造体の名前。 1〜32 バイト。
_ExmProcessAdmin	R	Text	このプロセス・インスタンスのプロセス管理者のユーザー ID。 0〜32 バイト。

表 6. プロセス・テンプレート (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmStarter	R	Text	このプロセス・インスタンスを開始したユーザーのユーザー ID。 0〜32 バイト。
_ExmRole	R	Text	スタッフ割り当てに使用できる役割。 0〜32 バイト。
_ExmOrganization	R	Text	スタッフ割り当てに使用できる組織。 0〜32 バイト。
_ExmDocumentation	R	Text	プロセス・インスタンスのドキュメンテーション。 0〜4 096 バイト。
_ExmStartTime	R	Time	プロセス・インスタンスが開始された日付と時刻。
_ExmNotificationTime	R	Time	プロセス・インスタンスの通知が発生した日付と時刻。
_ExmEndTime	I	Time	プロセス・インスタンスが完了した日付と時刻。
内部フィールド			
_ExmReplicationDate	I	Text	
インプリメンテーション固有のフィールド			
_ImplOriginalForm		Text	

プロセス・インスタンス通知

表 7. プロセス・インスタンス通知

フィールド名	設定	データ型	フィールドの説明と内容
ロータス ノーツ API のフィールド			
FORM		Text	“ExmWiPin” に設定。
_ExmDocType	I	Text	“ExmWiPin” に設定。

表 7. プロセス・インスタンス通知 (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmDocObjectId	I	Text	1〜n バイト。
_ExmDatabase	I	Text	MQSeries Workflow システム・グループの名前。 名前の長さは 1〜8 バイト。
_ExmUserId	I	Text	担当者の MQSeries Workflow ユーザー ID。 ユーザー ID の長さは 1〜32 バイト。
_ExmExpired	I	Text	プロセス・インスタンスの最大期間が満了しているかどうか。 値は次のいずれか。 **値　説明** **'0'**　No **'1'**　Yes
_ExmName	I	Text	プロセス・インスタンスの名前。 1〜63 バイト。
_ExmDescription	I	Text	プロセス・インスタンスについての説明。 0〜254 バイト。
_ExmCategory	I	Text	プロセス・インスタンス通知の属するプロセスのカテゴリー。 0〜32 バイト。

表 7. プロセス・インスタンス通知 (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmState	I	Text	プロセス・インスタンスの状況。 値は次のいずれか。 **値** **説明** **'0'** Undefined **'1'** Ready **'2'** Running **'4'** Finished **'8'** Terminated **'16'** Suspended
_ExmParentName	I	Text	この通知に関連したプロセス・インスタンスの上位プロセスの名前。 0～63 バイト。
_ExmTopLevelName	I	Text	この通知に関連したプロセス・インスタンスの最上位プロセスの名前。 1～63 バイト。
_ExmAudit	I	Text	この通知に関連したプロセス・インスタンスについて監査ログを作成するかどうか。 値は次のいずれか。 **値** **説明** **'0'** No **'1'** Yes

表 7. プロセス・インスタンス通知 (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmTerminateOnError	I	Text	終了条件または分岐条件の評価時にエラーが発生した場合に、この通知に関連したプロセス・インスタンスを中止するかどうか。 値は次のいずれか。 **値** **説明** **'0'** No **'1'** Yes
_ExmInContainerNeeded	I	Text	この通知に関連したプロセス・インスタンスに入力データが必要かどうか。 値は次のいずれか。 **値** **説明** **'0'** No **'1'** Yes
_ExmInContainerName	I	Text	プロセス・インスタンスの入力コンテナー・データ構造体の名前。 1〜32 バイト。
_ExmOutContainerName	I	Text	プロセス・インスタンスの出力コンテナー・データ構造体の名前。 1〜32 バイト。
_ExmProcessAdmin	I	Text	プロセス・インスタンスのプロセス管理者のユーザー ID。 1〜32 バイト。
_ExmStarter	I	Text	この通知に関連したプロセス・インスタンスを開始したユーザーのユーザー ID。 1〜32 バイト。

表 7. プロセス・インスタンス通知 (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmRole	I	Text	この通知に関連したプロセス・インスタンスのスタッフ割り当てに使用できる役割。 0〜32 バイト。
_ExmOrganization	I	Text	この通知に関連したプロセス・インスタンスのスタッフ割り当てに使用できる組織。 0〜32 バイト。
_ExmDocumentation	I	Text	プロセス・インスタンスのドキュメンテーション。 0〜4 096 バイト。
_ExmStartTime	I	Time	プロセス・インスタンスが開始された日付と時刻。
_ExmNotificationTime	I	Time	この通知に関連したプロセス・インスタンスについて通知の発生した日付と時刻。
_ExmEndTime	I	Time	項目が完了した日付と時刻。
_ExmProcessInstanceName	I	Text	この通知に関連したプロセス・インスタンスの名前。 1〜54 バイト。
内部フィールド			
_ExmReplicationDate	I	Text	
インプリメンテーション固有のフィールド			
_ImplOriginalForm		Text	

プロセス・テンプレート

表 8. プロセス・テンプレート

フィールド名	設定	データ型	フィールドの説明と内容
ロータス ノーツ API のフィールド			
FORM		Text	“ExmPt” に設定。

表 8. プロセス・テンプレート (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmDocType	I	Text	“ExmPt” に設定。
_ExmDocObjectId	I	Text	1〜n バイト。
_ExmDatabase	I	Text	MQSeries Workflow システム・グループの名前。 名前の長さは 1〜8 バイト。
_ExmUserId	I	Text	担当者の MQSeries Workflow ユーザー ID。 ユーザー ID の長さは 1〜32 バイト。
_ExmName	I	Text	プロセス・テンプレートの名前。 1〜32 バイト。
_ExmDescription	I	Text	プロセス・テンプレートについての説明。 0〜1 024 バイト。
_ExmCategory	I	Text	プロセス・テンプレートのカテゴリー。 0〜32 バイト。
_ExmAudit	R	Text	このプロセス・テンプレートから作成されるプロセス・インスタンスについて監査ログを作成するかどうか。 値は次のいずれか。 **値** **説明** **'0'** No **'1'** Yes

表 8. プロセス・テンプレート (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmTerminateOnError	R	Text	終了条件または分岐条件の評価時にエラーが発生した場合に、このテンプレートから作成されるプロセス・インスタンスを中止するかどうか。 値は次のいずれか。 **値** **説明** **'0'** No **'1'** Yes
_ExmInContainerNeeded	I	Text	このテンプレートから作成されるプロセス・インスタンスに入力データが必要かどうか。 値は次のいずれか。 **値** **説明** **'0'** No **'1'** Yes
_ExmInContainerName	R	Text	プロセスの入力コンテナー・データ構造体の名前。 1〜32 バイト。
_ExmOutContainerName	R	Text	プロセスの出力コンテナー・データ構造体の名前。 1〜32 バイト。
_ExmProcessAdmin	R	Text	このテンプレートから作成されるプロセス・インスタンスのプロセス管理者のユーザー ID。 0〜32 バイト。
_ExmRole	R	Text	スタッフ割り当てに使用できる役割。 0〜32 バイト。
_ExmOrganization	R	Text	スタッフ割り当てに使用できる組織。 0〜32 バイト。

表 8. プロセス・テンプレート (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmDocumentation	R	Text	プロセス・テンプレートのドキュメンテーション。 0〜4 096 バイト。
内部フィールド			
_ExmReplicationDate	I	Text	
インプリメンテーション固有のフィールド			
_ImplOriginalForm		Text	

作業項目

表 9. 作業項目

フィールド名	設定	データ型	フィールドの説明と内容
ロータス ノーツ API のフィールド			
FORM		Text	“ExmWi” に設定。
_ExmDocType	I	Text	“ExmWi” に設定。
_ExmDocObjectId	I	Text	1〜n バイト。
_ExmDatabase	I	Text	MQSeries Workflow システム・グループの名前。 名前の長さは 1〜8 バイト。
_ExmUserId	I	Text	担当者の MQSeries Workflow ユーザー ID。 ユーザー ID の長さは 1〜32 バイト。
_ExmName	I	Text	作業項目の名前。 1〜32 バイト。
_ExmDescription	I	Text	作業項目についての説明。更新 API 関数により MQSeries Workflow で変更可。0〜1 024 バイト。

表 9. 作業項目 (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmState	I	Text	作業項目の状況。 値は次のいずれか。 **値** **説明** **'0'** Undefined **'1'** Ready **'2'** Running **'4'** Finished **'8'** Terminated **'16'** Suspended **'32'** Disabled
_ExmEscalated	I	Text	作業項目がエスカレートされるかどうか (つまり作業項目に関する通知があるかどうか)。 値は次のいずれか。 **値** **説明** **'0'** No **'1'** Yes
_ExmImplementation	I	Text	作業項目をインプリメントするツールまたはプロセスの名前。 1〜32 バイト。
_ExmReceivedAs	I	Text	この作業項目がワークリスト内に入れられた理由。 値は次のいずれか。 **値** **説明** **'0'** Undefined **'1'** Normal **'2'** Substitute **'3'** ProcessAdministrator **'4'** SystemAdministrator **'5'** ByTransfer

表 9. 作業項目 (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmProcessInstanceName	I	Text	作業項目の属するプロセス・インスタンスの名前。 1〜54 バイト。
_ExmProcessCategory	I	Text	作業項目の属するプロセスのカテゴリー。 0〜32 バイト。
_ExmManualStartMode	R	Text	項目の開始モードが手動かどうか。 値は次のいずれか。 **値** **説明** **'0'** No **'1'** Yes
_ExmStartCondition	R	Text	作業項目に関連した開始条件。 値は次のいずれか。 **値** **説明** **'ANY'** 入力コネクターの 1 つが真でなければならない。 **'ALL'** 入力コネクターのすべてが真でなければならない。
_ExmManualExitMode	R	Text	作業項目の終了モードが手動かどうか。 値は次のいずれか。 **値** **説明** **'0'** No **'1'** Yes
_ExmExitCondition	R	Text	作業項目の終了条件。 4 KB 以下のデータ。

表 9. 作業項目 (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmStaff	R	Text	作業項目によって表されているアクティビティーに関連したすべての担当者のユーザー ID。 複数値フィールド。 0〜32 バイト。
_ExmProcessAdmin	R	Text	作業項目の属するプロセスのプロセス管理者のユーザー ID。 1〜32 バイト。
_ExmPriority	I	Text	作業項目の優先順位。 0〜9 の数値。
_ExmDocumentation	R	Text	作業項目のドキュメンテーション。 0〜4 096 バイト。
_ExmStartTime	I	Time	アクティビティーが開始された (つまり作業項目が Ready 状態になった) 時刻と日付。
_ExmNotificationTime	R	Time	作業項目の通知が発生した日付と時刻。
_ExmEndTime	R	Time	作業項目が完了した日付と時刻。
内部フィールド			
_ExmReplicationDate	I	Text	
インプリメンテーション固有のフィールド			
_ImplOriginalForm		Text	
_ImplNewOwner		Text	
_ImplTool		Text	
_ImplToolsList		Text	複数値フィールド。
_ImplRc		Text	

作業項目通知

表 10. 作業項目通知

フィールド名	設定	データ型	フィールドの説明と内容
ロータス ノーツ API のフィールド			
FORM		Text	“ExmWiWin” に設定。
_ExmDocType	I	Text	“ExmWiWin” に設定。
_ExmDocObjectId	I	Text	1〜n バイト。
_ExmDatabase	I	Text	MQSeries Workflow システム・グループの名前。 名前の長さは 1〜8 バイト。
_ExmUserId	I	Text	担当者の MQSeries Workflow ユーザー ID。 ユーザー ID の長さは 1〜32 バイト。
_ExmExpired	I	Text	作業項目の最大期間が満了しているかどうか。 値は次のいずれか。 **値** **説明** **'0'** No **'1'** Yes
_ExmOverdue	I	Text	作業項目の開始が遅れているかどうか。 値は次のいずれか。 **値** **説明** **'0'** No **'1'** Yes

表 10. 作業項目通知 (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmEscalation	I	Text	エスカレートが第 1 のエスカレートか第 2 のエスカレートか。 値は次のいずれか。 **値** **説明** **'1'** 第 1 のエスカレート **'2'** 第 2 のエスカレート
_ExmName	I	Text	作業項目の名前。 1〜32 バイト。
_ExmDescription	I	Text	作業項目についての説明。 0〜1 024 バイト。
_ExmState	I	Text	作業項目の状況。 値は次のいずれか。 **値** **説明** **'0'** Undefined **'1'** Ready **'2'** Running **'4'** Finished **'8'** Terminated **'16'** Suspended **'32'** Disabled
_ExmEscalated	I	Text	作業項目がエスカレートされるかどうか。 値は次のいずれか。 **値** **説明** **'0'** No **'1'** Yes
_ExmImplementation	I	Text	作業項目をインプリメントするツールまたはプロセスの名前。 1〜32 バイト。

表 10. 作業項目通知 (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmReceivedAs	I	Text	この作業項目通知がワークリスト内に入れられた理由。 値は次のいずれか。 **値** **説明** **'0'** Undefined **'1'** Normal **'2'** Substitute **'3'** ProcessAdministrator **'4'** SystemAdministrator **'5'** ByTransfer
_ExmProcessInstanceName	I	Text	作業項目通知の属するプロセス・インスタンスの名前。 1〜54 バイト。
_ExmProcessCategory	I	Text	作業項目通知の属するプロセスのカテゴリー。 0〜32 バイト。
_ExmManualStartMode	R	Text	項目の開始モードが手動かどうか。 値は次のいずれか。 **値** **説明** **'0'** No **'1'** Yes
_ExmStartCondition	R	Text	作業項目に関連した開始条件。 値は次のいずれか。 **値** **説明** **'ANY'** 入力コネクターの 1 つが真でなければならない。 **'ALL'** 入力コネクターのすべてが真でなければならない。

表 10. 作業項目通知 (続き)

フィールド名	設定	データ型	フィールドの説明と内容
_ExmManualExitMode	R	Text	作業項目の終了モードが手動かどうか。 値は次のいずれか。 **値** **説明** **'0'** No **'1'** Yes
_ExmExitCondition	R	Text	作業項目の終了条件。 4 KB 以下のデータ。
_ExmStaff	R	Text	作業項目によって表されているアクティビティーに関連したすべての担当者のユーザー ID。 複数値フィールド。 0〜32 バイト。
_ExmPriority	I	Text	作業項目の優先順位。 0〜9 の数値。
_ExmDocumentation	R	Text	作業項目のドキュメンテーション。 0〜4 096 バイト。
_ExmStartTime	R	Time	アクティビティーが開始された (つまり作業項目が Ready 状態になった) 時刻と日付。
_ExmNotificationTime	R	Time	作業項目の第 2 の通知が発生した日付と時刻。
_ExmEndTime	R	Time	作業項目通知が完了した日付と時刻。
インプリメンテーション固有のフィールド			
_ImplOriginalForm		Text	

第11部 付録および後付け

付録A. 構文図の読み方

本書を通じて、構文は以下のように記述されています。すべてのスペースとその他の文字にはそれぞれ意味があります。

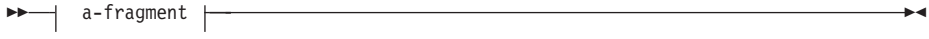
- 構文図は、左から右、上から下へ、行の主線に沿って読んでください。

  ▶▶— 記号は、ステートメントの始まりを示します。

  —▶ 記号は、ステートメントの構文が次の行に続いていることを示します。

  ▶— 記号は、ステートメントが前の行から続いていることを示します。

  —▶◀ 記号は、ステートメントの終わりを示します。
- 構文図は、いくつかのフラグメントに分割されていることがあります。 1 つのフラグメントは縦線によって区切られており、縦線と縦線の間にフラグメントの名前が示されます。フラグメント自体は、主構文図と同じ構文規則に従います。

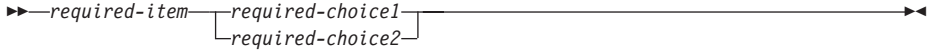


- 必須項目は水平線 (主線) と同じ高さに示されます。

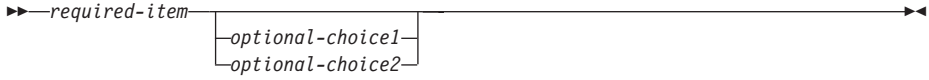
- オプション項目は、主線の下側 (または上側) に示されます。

- 1 つ以上の項目から選択できる場合は、上下に重ねて示されます。

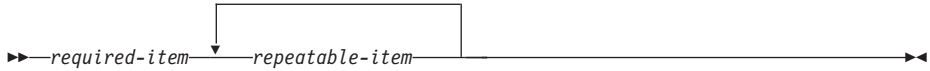
  項目の 1 つを選択することが必須の場合は、重なっている項目の 1 つが主線と同じ高さに示されます。



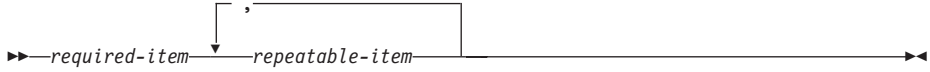
  項目の 1 つを選択することがオプションの場合は、重なっている項目全部が主線の下側に示されます。



- 主線の上側に左へ戻る矢印がある場合、それは繰り返すことができる項目を示しています。



反復矢印にコンマが含まれている場合は、反復項目と反復項目の間をコンマで区切る必要があります。



- キーワードは英大文字で示されます (例: NAME)。示されているとおりに入力する必要があります。変数は小文字のイタリック文字で示されます。(例: *string*)。これはユーザーの入力する値です。

付録B. MQSeries FlowMark バージョン 2 互換モード

MQSeries Workflow API は、MQSeries FlowMark バージョン 2.3 API 互換モードをサポートしています。これにより、MQSeries FlowMark バージョン 2.3 プログラムを実行することができます。ただし、どのアプリケーションでも、バージョン 2 の関数 / メソッドの代わりに MQSeries Workflow バージョン 3 の関数 / メソッドを使用するようにしてください。

互換モードでは、以下の言語とコンパイラーがサポートされています。

- C 言語 API
  - AIX と IBM C Set++ バージョン 3.1.4
  - OS/2 と IBM VisualAge for C++ 3.0
  - Windows NT または Windows 95 と IBM VisualAge for C++ 3.5 または Microsoft Visual C++ 5.0
- C++ 言語 API
  - AIX と IBM C Set++ バージョン 3.1.4
  - HP-UX および HP aC++ Compiler S700 バージョン A.01.15.01
  - OS/2 と IBM VisualAge for C++ 3.0
  - Windows NT または Windows 95 と IBM VisualAge for C++ 3.5 または Microsoft Visual C++ 5.0
- REXX API
  - OS/2 とオペレーティング・システム提供のコンパイラー
- VisualBasic API
  - Windows NT または Windows 95 および Microsoft VisualBasic 5.0

バージョン 2.3 プログラムを実行するには、コンパイルとリンクのステップを繰り返せば十分です。

MQSeries Workflow バージョン 3 には、このための新しいヘッダー、ライブラリー、およびダイナミック・リンク・ライブラリーが含まれています。そのヘッダー・ファイルには MQSeries FlowMark バージョン 2.3 の名前が付けられており、ソース・コードを変更する必要がないようになっています。ライブラリーと動的リンク・ライブラリーには新しい名前が付けられています。ただし、それらのファイル (DLL) をそのバージョン 2 の名前に変更することもできます。そうしない場合には、リンク・ステップを新しい名前に対応したものにする必要があります。互換 API、および組み込みファイル、リンクするファイルについて、次の表にまとめます。

表 11. MQSeries FlowMark バージョン 2 互換 API (AIX)

言語	AIX		
	ヘッダー	lib	DLL
C	exmajapc.h	fmcjdapc.lib	libfmcjdapc.a
C++	exmpjapi.hxx	fmcjdcom.lib fmcjdcbr.lib fmcjdrun.lib fmcjdcon.lib	libfmcjdcom.a libfmcjdcbr.a libfmcjdrun.a libfmcjdcon.a

表 12. MQSeries FlowMark バージョン 2 互換 API (HP-UX)

言語	HP-UX		
	ヘッダー	lib	DLL
C++	exmpjapi.hxx	fmcjdcom.lib fmcjdcbr.lib fmcjdrun.lib fmcjdcon.lib	libfmcjdcom.sl libfmcjdcbr.sl libfmcjdrun.sl libfmcjdcon.sl

表 13. MQSeries FlowMark バージョン 2 互換 API (OS/2)

言語	OS/2		
	ヘッダー	lib	DLL
C	exmpjapc.h	fmcjdapc.lib	fmcjdapc.dll
C++	exmpjapi.hxx	fmcjdcom.lib fmcjdcbr.lib fmcjdrun.lib fmcjdcon.lib	fmcjdcom.dll fmcjdcbr.dll fmcjdrun.dll fmcjdcon.dll
REXX			fmcjdrex.dll

表 14. MQSeries FlowMark バージョン 2 互換 API (NT/95)

言語	Windows NT/95		
	ヘッダー	lib	DLL
C	exmwjapc.h	fmcjdapc.lib	fmcjdapc.dll
C++	exmpjapi.hxx	fmcjdcom.lib fmcjdcbr.lib fmcjdrun.lib fmcjdcon.lib	fmcjdcom.dll fmcjdcbr.dll fmcjdrun.dll fmcjdcon.dll
VisualBasic	exmbjapv.bas		fmcjdapv.dll

MQSeries Workflow バージョン 3 では機能や柔軟性の点で拡張されているため、API に関連していくつかの点が違っています。それらの違いには十分注意してください。バージョン 2 のプログラムの動作に影響があるかもしれません。これは、主として戻りコードや許可の定義に適用されます。

MQSeries FlowMark バージョン 2 との相違点

- バージョン 3 は、バージョン 2 よりも詳細な状態を示すため、それらの状態はそれぞれ最善の結果をもたらすものにマッピングされます。

  作業項目が開始されてもプログラム実行エージェントが実行されていないか、または開始すべきプログラムが見つからない場合、作業項目は *InError* 状態になります。 InError 状態は、バージョン 2 では *Running (実行中)* 状態となるので、バージョン 2 の Restart() または Terminate() を発行することができます。実際の InError 状態のために、バージョン 3 の ForceRestart() または ForceFinish() が呼び出されますが、それには呼び出し側にプロセス管理者権限が必要です。

  作業項目をチェックアウトすると、その作業項目の状態は *CheckedOut* になります。 CheckedOut 状態は、バージョン 2 では *Running (実行中)* 状態となるので、バージョン 2 の Restart() または Terminate() を発行することができます。実際の CheckedOut 状態のために、バージョン 3 の ForceRestart() または ForceFinish() が呼び出されますが、それには呼び出し側にプロセス管理者権限が必要です。

  手動終了作業項目によりそのアクティビティー・インプリメンテーションが実行されると、*Executed* 状態になります。これはバージョン 2 の *Ready* 状態にマッピングされます。この状態では、プログラムはすでに実行されており、バージョン 2 の ManualExit() で終了できます。

- バージョン 3 でサポートされる許可の概念は、バージョン 2 の概念より制限の厳しいものになっています。

  作業項目の ForceFinish() および ForceRestart() には、プロセス管理者権限が必要です。この権限は、バージョン 2 の C 言語 ChangeActivityState() 関数を使って終了や再始動を要求するときにも必要です。

  許可の変更はすぐにアクティブになります。これは、許可がサーバーだけによって検査されるためです。 UserSettings() メソッドが呼び出されると、それはユーザーの設定値を戻します。すなわち、呼び出されるたびにサーバーから現行のユーザー権限を取り出します。バージョン 2 では、ユーザー設定値はログオン時に 1 回だけ取り出されました。

- バージョン 3 の戻りコードは、それぞれ最善の結果をもたらすバージョン 2 の戻りコードにマッピングされます。

指定された時間内にクライアントが応答を受信しないと、ERROR_TIMEOUT が戻されます。その場合、サーバーが稼働していないことも考えられます。

- 有効範囲やデータベースおよびサーバーの指定では、大文字小文字が区別されます。英大文字には変換されません。有効範囲やデータベースは、バージョン 3 のシステム・グループ仕様に対応し、サーバーは、システム仕様に対応します。
- パスワードには大文字小文字の区別があります。
- Logon() では、不在設定の指定が許可されます。互換モードの場合のように不在の設定がない場合、不在の動作 (不在情報がリセットされるかどうか) は、担当員レコードから取られます。
- MQSeries Workflow サーバーで*統一ログオン* が可能な場合は、空のパスワードとユーザー ID を指定してもかまいません (439ページの『Logon()』も参照)。
- Logon() では、セッション・モードの指定が許可されます。互換モードの場合のようにセッション・モードの設定がない場合、セッション・モードはユーザー・プロファイルまたは構成プロファイルから取られます。そこにも見つからない場合は、セッション・モード *Present here* が使用されます。PRESENT HERE は、他の在席セッションのユーザー・ログオフを強制します。

  PRESENT セッション・モードを使用しているアプリケーションがアプリケーションのテスト段階で異常終了しても、サーバー上にはセッション・レコードが保持されています。そのような場合、セッションが期限切れになるまで新しいログオン試行を待たなければなりません。

  PRESENT HERE は、存在している保留中の在席セッションのログオフを強制します。 *already logged on* 戻りコードに依存している場合は、*present* モードをプロファイルで設定してください。 *Default* は複数の並行セッションを可能にします。

  構成ユーティリティー fmczutil を使用すると、プロファイル値を設定または消去できます。キー (key) は V2_SESSION_MODE、値 (value) は DEFAULT、 PRESENT、または PRESENTHERE です。
- IsDeleteFinishedItems() は常に偽を戻します。この設定はユーザーからプロセスに移ったためです。この設定を変更しようとすると FMC_OK が戻されますが、何も変更されません。
- PersonsAuthorizedFor() メソッドは、ログオン・ユーザーが全担当者に対して許可を受けている場合は、どの担当者も戻しません。
- SetPersonsToStandInFor() にはスタッフ許可が必要です。

- ログオン・ユーザーが全カテゴリーの管理者である場合は、そのカテゴリーが存在しない場合でも IsAdminForCategory(x) が真を戻します。
- プロセス・テンプレートは、MQSeries Workflow でバージョン管理されています。これは、プロセス・テンプレートが有効でなくなる、つまり*無効* になる可能性があることを意味しています。
- プロセス・インスタンスの作成時には、1 次値だけが戻されます。
- プロセス・インスタンスの名前は、生成方法が異なります。プロセス・テンプレート名に _n が埋め込まれることはなくなりましたが、代わりに $ 記号が入り、その後にオブジェクト ID が続く形式で固有の名前が設定されます。
- プロセス・インスタンスの入力コンテナー名および出力コンテナー名は 2 次属性です。つまり、それらを読み取る前にプロセス・インスタンスをリフレッシュしなければなりません。
- プロセス・テンプレートまたはプロセス・インスタンスに対する IsTerminatedOnError() メソッドの戻り値は常に false です。
- MQSeries Workflow では、サブプロセスの自立走行の概念を取り入れています。 deep オプションが指定された場合、制御の自立性に関して自立走行式でないサブプロセスだけが中断または再開されます。 Ready 状態のプロセス・インスタンスの Suspend() はサポートされていません。
- ワークリストを作成しても、アイテムの所有者が登録済みユーザーかどうかや、その作成者がそのユーザーのアイテムを見る権限を持っているかどうかは検査されません。とにかくワークリストは作成されます。アイテムの所有者はバージョン 3 ではフィルターの一部であるため、ワークリストの内容 (つまりアイテム) が照会されるときにだけ所有者が適用されます。不明なアイテム所有者を指定した場合や、この操作を行っている人が指定の所有者のアイテムを見る権限を持っていない場合は、アイテムはまったく表示されません。
- 作業項目の記述を変更しただけでも、作業項目の最終修正時刻が変更されます。
- 作業項目のチェックアウトは、対応する MQSeries Workflow 設定でチェックアウトが定義されている場合にのみ可能です。定義されていない場合、FMC_ERROR_CHECKOUT_NOT_POSSIBLE が戻されます。
- 作動可能状態の作業項目の削除は、それがアクティビティー・インスタンスに関連した最後の作業でない限り可能です。中止された、または中止処理中のプロセス・インスタンスでは、作動可能状態の作業項目を常に削除できます。

- バージョン 3 では、0〜999 の間で優先順位を設定できます (FlowMark バージョン 2 では 0〜9)。互換性 API により、フィルター基準として MAX_PRIORITY に 999 を指定し、 9 より大きい優先順位を持つオブジェクトが検索できるようになりました。
- 通知期間が設定されていないなら、通知は**全く出されません**。通知先の担当者が指定されていないか不明な場合、プロセス管理者が通知を受けます。
- 通知に対する Finish() 操作は、FMC_OK を戻すシステムでサポートされています。つまり、通知は必ず明示的に削除しなければならないということです。
- 入出力コンテナーは、アクティビティー・インプリメンテーションまたはサポート・ツールによってアクセスされる場合にのみ、プログラム実行エージェントに送信されます。この動作は、バージョン 3 で設定することができます。
- プログラム実行エージェントはプログラム識別 (バージョン 2 では、セッション ID) を、*承認* プログラムにのみ提供します。この特性は、バージョン 3 で設定することができます。
- Passthrough() はサポート・ツールから呼び出すことができません。
- クライアントとサーバーの間で渡されるコンテナーの最大サイズは 32KB です。
- 1 つのコンテナーにはバイナリー・データ型のリーフを含めることができます。
- ExmcChangeActivityState() は、Requested ready 状態または Running (実行中) 状態を最初に検査することなく、終了アクションを実行します。現時点では、アクティビティー名はプロセス・インスタンス内で固有でなければなりません。つまり、ブロック内の固有アクティビティー名のサポートはありません。また、プロセスが実行中でない場合、EXMPJ_WRONG_ACTIVITY_STATE も戻されます。
- バンドルはまだサポートされていません。

MQSeries FlowMark バージョン 2 の C 言語プログラム

既存のアプリケーション・プログラムの実行

MQSeries Workflow バージョン 3 は、MQSeries FlowMark バージョン 2 アプリケーション・プログラムを変更せずに実行できるように提供されています。追加された許可要件については、985ページの『MQSeries FlowMark バージョン 2 との相違点』を参照してください。バージョン 2 プログラムを実行するには、コンパイルとリンクを実行すれば十分です。

MQSeries Workflow バージョン 3 環境で既存の MQSeries FlowMark バージョン 2 C 言語アプリケーション・プログラムを実行したい場合は、以下のようにしてください。

- 以下のバージョン 3 用の新規ヘッダー・ファイルが使用されるように、MQSeries Workflow バージョン 3 パスを検索するようにします。
  - *exmwjapc.h* (Windows NT または Windows 95)
  - *exmajapc.h* (AIX)
- アプリケーション構築ステップを変更して、MQSeries FlowMark バージョン 2 ライブラリーの代わりに MQSeries Workflow バージョン 3 の API ライブラリー *fmcjdapc.lib* とリンクします。
- アプリケーションをコンパイルおよびリンクします。

MQSeries FlowMark バージョン 2 の Visual Basic プログラム

既存のアプリケーション・プログラムの実行

MQSeries Workflow バージョン 3 は、MQSeries FlowMark バージョン 2 アプリケーション・プログラムを変更せずに実行できるように提供されています。追加された許可要件については、985ページの『MQSeries FlowMark バージョン 2 との相違点』を参照してください。

MQSeries Workflow バージョン 3 環境で既存の MQSeries FlowMark バージョン 2 Visual Basic アプリケーション・プログラムを実行したい場合は、以下のことを確認してください。

- 新しいバージョン 3 提供の宣言 *exmbjapv.bas* が使用されるように、MQSeries Workflow バージョン 3 パスを検索するようにします。
- MQSeries Workflow バージョン 3 のダイナミック・リンク・ライブラリー fmcjdapv.dll を、PATH ステートメント中のディレクトリーに入れます。

MQSeries FlowMark バージョン 2 の REXX プログラム

既存のアプリケーション・プログラムの実行

MQSeries Workflow バージョン 3 は、MQSeries FlowMark バージョン 2 アプリケーション・プログラムを変更せずに実行できるように提供されています。追加された許可要件については、985ページの『MQSeries FlowMark バージョン 2 との相違点』を参照してください。

MQSeries Workflow バージョン 3 環境で既存の MQSeries FlowMark バージョン 2 C++ アプリケーション・プログラムを実行したい場合は、以下のようにしてください。

- fmcjdrex.dlo に含まれる MQSeries Workflow REXX 関数をロードします。
- MQSeries Workflow バージョン 3 のダイナミック・リンク・ライブラリー fmcjdapv.dll を、PATH ステートメント中のディレクトリーに入れます。

MQSeries FlowMark バージョン 2 の C++ プログラム

既存のアプリケーション・プログラムの実行

MQSeries Workflow バージョン 3 は、MQSeries FlowMark バージョン 2 アプリケーション・プログラムを変更せずに実行できるように提供されています。追加された許可要件については、985ページの『MQSeries FlowMark バージョン 2 との相違点』を参照してください。バージョン 2 プログラムを実行するには、再コンパイルと再リンクを実行すれば十分です。

MQSeries Workflow バージョン 3 環境で既存の MQSeries FlowMark バージョン 2 C++ アプリケーション・プログラムを実行したい場合は、以下のようにしてください。

- 新しいバージョン 3 提供の *exmpjapi.hxx* ヘッダー・ファイルが使用されるように、MQSeries Workflow バージョン 3 パスの検索を確認します。

  注: MQSeries FlowMark バージョン 2 の exmpjapi.hxx ヘッダー・ファイルは自己完結しているため、アプリケーション中に他の MQSeries FlowMark ヘッダー・ファイルを組み込む必要はありません。他のファイルが組み込まれている場合には、すべて削除してください。

- アプリケーション構築ステップを変更して、MQSeries Workflow バージョン 3 の API ライブラリー *fmcjdcom.lib* と、 *fmcjdcbr.lib*、*fmcjdrun.lib* または *fmcjdcon.lib* (あるいはこれらのいくつか) を、MQSeries FlowMark バージョン 2 ライブラリーの代わりにリンクします。 155ページの『第14章 コンパイルとリンク』を参照してください。
- アプリケーションをコンパイルおよびリンクします。

MQSeries Workflow バージョン 3 メソッドの使用

MQSeries Workflow バージョン 3 API 付属の新しいメソッドを使用するために、既存の MQSeries FlowMark バージョン 2 の C++ アプリケーション・プ

ログラムを拡張したい場合、ご使用のアプリケーションを最初に移行する必要があります。これは、MQSeries FlowMark 2 からのいくらかの拡張および変更があるためです。

以下のリストに従って移行します。

1. **実行すべき一般的なステップ**

   注: 一括変更ステップのいくつかは互いに関連があるため、以下に示されているステップの順序に従う必要があります。

   ヘッダー・ファイルの組み込み:

```
#include <bool.h>                       // true, false  (dependent inclusion)
#include <fmcjstr.hxx>                  // string       (dependent inclusion)
#include <vector.h>                     // vector       (dependent inclusion)
#include <fmcjprun.hxx>                 // C++ runtime client interface
   or
#include <fmcjpcon.hxx>                 // C++ container interface
```

   a. MQSeries FlowMark バージョン 2 ヘッダー・ファイル exmpjapi.hxx の代わりに、 MQSeries Workflow バージョン 3 C++ 実行機能 API ヘッダー・ファイル *fmcjprun.hxx* または *fmcjpcon.hxx* を組み込む。

      注: MQSeries FlowMark バージョン 2 の exmpjapi.hxx ヘッダー・ファイルは自己完結しているため、アプリケーション中に他の MQSeries FlowMark ヘッダー・ファイルを組み込む必要はありません。他のファイルが組み込まれている場合には、すべて削除してください。

   b. fmcjprun.hxx または fmcjpcon.hxx の前に、条件付きで bool.h を組み込む。

      ご使用のコンパイラーがブール定義をサポートしていない場合、MQSeries Workflow 付属のブールの定義を組み込みます。サポートされている場合、ご使用のコンパイラーのブール定義を使用してください。

      注: bool.h は、ストリング定義ファイルの前に組み込む必要があります。

   c. fmcjprun.hxx または fmcjpcon.hxx の前に、条件付きで fmcjstr.hxx を組み込む。

      ご使用のコンパイラーがストリング・クラスをサポートしていない場合、MQSeries Workflow 付属のストリング・クラスの定義を組み込みます。サポートされている場合、ご使用のコンパイラー・ストリング定義ファイルを組み込んでください。

   d. fmcjprun.hxx の前に、条件付きで vector.h を組み込む。

ご使用のコンパイラーがベクトルをサポートしていない場合、MQSeries Workflow 付属のベクトルの定義を組み込みます。サポートされている場合、ご使用のコンパイラー・ベクトル定義ファイルを組み込んでください。

エラー・コードの名前:

e. すべての EXM_API_ERROR の出現個所を FMC_ERROR に変更する。

このステップの後、エラー・コードが正しく指定されます。

クラスの名前:

f. すべての Exm の出現個所を Fmcj に変更する。 結果的に生じる FmcjServer 参照を FmcjExecutionService に変更する。結果的に生じる FmcjWorkitemNotification 参照を FmcjActivityInstanceNotification に変更する。結果的に生じる FmcjItemBase 参照を FmcjItem に変更する。結果的に生じる FmcjUser 参照を FmcjPerson に変更する。

このステップの後、クラス名が正しく指定されます。

メソッドの名前:

g. すべての GetXxx() メソッド名を Xxx() に変更する。ただし、その名前を変更しない GetElement() メソッドを**除く**。すべての ChangeXxx() メソッド名を SetXxx() に変更する。

すなわち、アクセス機能メソッドは一貫して、データ・メンバー名に応じて名前を付けられます。 mutator メソッドには接頭部 ″Set″ が付きます。

すべての EndCondition() メソッド呼び出しを ExitCondition() に変更する。
すべての ExecutionSessionID() メソッド呼び出しを ProgramID() に変更する。
すべての PersistentHandle() メソッド呼び出しを PersistentOid() に変更する。
すべての ReadPersistentObject() メソッド呼び出しを PersistentObject() に変更する。
すべての Organization() メソッド呼び出しを OrganizationName() に変更する。
すべての Role() メソッド呼び出しを RoleName() に変更する。
すべての RolesToCoordinate() メソッド呼び出しを NamesOfRolesToCoordinate() に変更する。
すべての QueryWorkitemNotifications() メソッド呼び出しを QueryActivityInstanceNotifications() に変更する。

これは、バージョン 3 C++ API および C 言語 API の終結処置および互換性のためです。 PersistentHandle() メソッド名は、C-API ハンドルとの混同を避けるために変更されています。

このステップの後、サポートされているすべてのメソッド名が正しく指定されます。

2. **必須の特定ステップ**

これらは、名前付きクラスおよびメソッドを使用する場合に実行する**必要のある**ステップです。

- **FmcjActivityInstanceNotification**

  IsEscalated() メソッドは、列挙型として自動調整の正確な状態を戻す StateOfNotification() に変更されました。

  IsFirstEscalatation() によって以前に照会された情報、および IsSecondEscalation() メソッドは、Kind() メソッドを使用することによってアクセスすることができます。 Kind() メソッドは、アイテムのタイプを列挙型として戻します。

  IsProcessType() および IsProgramType() メソッドは、関連したアクティビティー・インスタンスの種類を列挙型として戻す ActivityKind() に変更されています。

  NotificationTime() メソッドは FirstNotificationTime() および SecondNotificationTime() に変更されているため、両方の時を照会することができます。

  ManualExit() および ManualStart() アクセス機能メソッドは、オブジェクトが完了していないかぎり偽 (ブール省略時値) を戻します。

  Finish() メソッドはもう必要ないので、削除する必要があります。

- **FmcjContainer**

  ActivityInfo() および ProcessInfo() メソッドは削除する必要があります。そのデータ・メンバーにアクセスする前にこれらのメソッドを呼び出す必要はありません。コンテナーを照会するときにその完全修飾名を指定するだけで十分です。

  アクティビティー・インプリメンテーションは、その入出力コンテナーを処理しているときにそのプログラム識別をプログラム実行エージェントに渡す必要はありません。すなわち、InContainer() または OutContainer() 呼び出しからプログラム ID パラメーターを削除するか、または適切な RemoteInContainer() または RemoteOutContainer() メソッドを呼び出す必要があります。

- **FmcjExecutionService** とそれに対応する **FmcjService**

常に特定のシステムまたはホーム・システムのどちらかに接続されるので、コンストラクター FmcjExecutionService(systemGroup) - ExmServer(scope) - は、もうサポートされていません。

Name() メソッドは、SystemName() に置換する必要があります。

Scope() メソッドは、SystemGroupName() に置換する必要があります。

ユーザー ID は大文字小文字が区別されます。以前のように、ログオン時にすべてが大文字に変換されなくなりました。

CreateWorklist() メソッドでは、永続リストの追加パラメーター、すなわちワークリスト所有者、タイプ、説明、ソート基準、および限界値を指定する必要があります。フィルター属性はストリングになり、作業項目の所有者はフィルター基準の一部になりました。

QueryProcessTemplates() および QueryProcessInstances() メソッドにより、新しくソート基準と限界値を指定することができます。フィルター属性はストリングになりました。

UserSettings() メソッドは、APIRET 値を戻すアクション・メソッドになりました。 MQSeries FlowMark バージョン 2 では、Logon() 要求が正常に処理された結果としてユーザー情報が提供されました。 MQSeries Workflow バージョン 3 では、ユーザー設定を明示的に照会する必要があります。

Passthrough() は、プログラム識別を渡す必要はありません。すなわち、プログラム ID パラメーターを削除するか、または RemotePassthrough() メソッドを呼び出す必要があります。

- **FmcjFilter**

  柔軟性と拡張性を強化するために、FmcjFilter クラスが削除されました。FmcjFilter オブジェクトの代わりに、フィルター式が含まれるストリングを指定する必要があります。

- **FmcjItem**

  アイテム・タイプ列挙型 ″WorkitemNotification″ がより細分化され、FirstActivityInstanceNotification と SecondInstanceActivityNotification に分けられました。アイテム上でそれが通知かどうかを調べる検査は、適切な or ステートメントに置換され、それが最初の通知か 2 番目の通知かを検査します。

  メソッド ManualExit()、ManualStart()、ExitCondition()、StartCondition()、Priority()、および Staff() は、プロセス・インスタンス通知に適用できません。このように、それらは FmcjItem クラスから FmcjWorkitem および FmcjActivityInstanceNotification クラスへ移されました。すなわち、それぞれの種類のオブジェクト上でのみ、それらを呼び出すことができます。

- **FmcjPerson**

  終了したアイテムを削除するかどうかを指定する機能は、ログオン・ユーザーからプロセス・モデルに移されました。それで、IsDeleteFinishedItems() および SetDeleteFinishedItems() 呼び出しは削除する必要があります。

- **FmcjProcessInstance**

  IsAudited() メソッドは、列挙型として監査のタイプを戻す AuditMode() に変更されました。

- **ExmProcessInstanceNotification**

  プロセス・インスタンス通知は、プロセス・インスタンスの有効期限が切れたときに取り上げられるだけなので、Expired() メソッドは削除されました。

  Finish() メソッドはもう必要ないので、削除する必要があります。

- **FmcjProcessTemplate**

  IsAudited() メソッドは、列挙型として監査のタイプを戻す AuditMode() に変更されました。

  CreateInstance() および CreateAndStartInstance() メソッドには、追加の、まだ予約されているパラメーターがあり、少なくとも 0 ポインターを用意しておく必要があります。

- **FmcjWorkitem**

  IsEscalated() メソッドは、列挙型として自動調整の正確な状態を戻す StateOfNotification() に変更されました。

  IsProcessType() および IsProgramType() メソッドは、列挙型としてアクティビティーから継承され、作業項目の種類を戻す ActivityKind() に変更されました。

  NotificationTime() メソッドは FirstNotificationTime() および SecondNotificationTime() に変更されているため、両方の時を照会することができます。

  ManualExit() および ManualStart() アクセス機能メソッドは、オブジェクトが完了していないかぎり偽 (ブール省略時値) を戻します。

  ManualExit() アクション・メソッドは、結果的に生じる状態にもっと良く適合するように Finish() に変更されました。

  CheckIn() メソッドには、作業項目出力コンテナーはもう必要ありません。コンテナー自体の代わりに、ご使用の出力コンテナーを指すポインターを渡す必要があります。

CheckOut() メソッドは、MQSeries Workflow が認識しているアクティビティー・インプリメンテーションについてのすべての情報を戻すことがあります。 MQSeries FlowMark バージョン 2 の動作を実現するために、共通データだけを要求し、そこから入力コンテナーを抜き出す必要があります。

- **FmcjWorklist**

  フィルターは、フィルター・オブジェクトとしてではなくストリングとして戻されます。

  IsDefault() メソッドは削除されています。ユーザーか他のだれかが作成した場合、1 つのワークリストだけが存在します。

  作業項目の所有者がフィルター指定の一部になったため、複数の所有者が存在する可能性があります。

  QueryWorkitems() メソッドでは、もう随時フィルターの指定をすることができません。ワークリスト定義を作成するときは、持続フィルターを指定することができます。フィルターを掛ける必要がある場合、適切なフィルターを持つワークリストを作成するか、または FmcjExecutionServive::QueryWorkitems() メソッドを使用してください。これにより随時フィルターを指定することができます。

- **CppStartApi** および **CppFinishApi** は、もう必要がなく、削除する必要があります。

3. **オプションの特定ステップ**

   これらは、実行するかしないかを選択できるステップです。

   - FmcjItem::StartTime()。 このメソッドを使用した場合、アイテムの作成時刻が実際に戻されていました。この意味体系を保持したい場合、メソッド名を CreationTime() に変更してください。 StartTime は、開始時刻を戻します。
   - リフレッシュ。 リフレッシュ・メソッドを使用してアクション後の関係のあるオブジェクト値を更新した場合、これはいつも必要というわけではありません。アクション・メソッドを呼び出した結果として値が変更され、オブジェクトが自動的に更新されます。たとえば、FmcjWorkitem::Start() を呼び出すと作業項目の状態が更新されます。
   - FmcjPerson::CategoriesAuthorizedFor()。このメソッドは、基本的特権の許可を受けているカテゴリーだけを戻します。 MQSeries FlowMark バージョン 2 の動作を保持したい場合、CategoriesAuthorizedForAsAdmin() を追加する必要があります。

付録C. 特記事項

本書において、日本では発表されていない IBM 製品 (機械およびプログラム)、プログラミングまたはサービスについて言及または説明する場合があります。しかし、このことは、弊社がこのような IBM 製品、プログラミングまたはサービスを、日本で発表する意図があることを必ずしも示すものではありません。本書で IBM ライセンス・プログラムまたは他の IBM 製品に言及している部分があっても、このことは当該プログラムまたは製品のみが使用可能であることを意味するものではありません。 IBM 製品、プログラム、またはサービスに代えて、 IBM の有効な知的所有権またはその他の法的に保護された権利を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM によって明示的に指定されたものを除き、他社の製品と組み合わせた場合の操作の評価と検証はお客様の責任で行っていただきます。

IBM は、本書で解説されている主題について特許権 (特許出願を含む)、商標権、または著作権を所有している場合があります。本書の提供は、これらの特許権、商標権、および著作権について、本書で明示されている場合を除き、実施権、使用権等を許諾することを意味するものではありません。実施権、使用権等の許諾については、下記の宛先に、書面にてご照会ください。

〒106-0032 東京都港区六本木 3 丁目 2-31
AP 事業所
IBM World Trade Asia Corporation
Intellectual Property Law & Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

本書に対して、周期的に変更が行われ、これらの変更は、文書の次版に組み込まれます。 IBM は、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム（本プログラムを含む）との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Deutschland
Informationssysteme GmbH
Department 3982
Pascalstrasse 100
70569 Stuttgart
Germany

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、 IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。また、IBM 以外の製品に関するパフォーマンスの正確性、互換性、またはその他の要求は確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

これらの情報は、計画の目的でのみ提供するものです。この情報は、説明されている製品が一般的に提供可能になる前に変更される場合があります。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あ

るいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。これらの例は、すべての場合について完全にテストされたものではありません。 IBM はこれらのプログラムの信頼性、可用性、および機能について法律上の瑕疵担保責任を含むいかなる明示または暗示の保証責任も負いません。

サンプル・ソース・コードのすべての部分、またはすべての派生した創作物には、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年). このコードの部分は、IBM Corp. のサンプル・プログラムからとったものです。
© Copyright IBM Corp.1993, 1999. All rights reserved.

この情報をソフトコピーでご覧になっている場合には、写真およびカラーの図表は現れない場合があります。

商標

次のものは、IBM Corporation の米国およびその他の国における商標です。

- AIX
- CICS
- C Set++
- FlowMark
- IBM
- IMS
- MQSeries
- OS/2
- OS/390

- VisualAge

Lotus Notes は Lotus Development Corporation の登録商標であり、 Domino および Lotus Go Webserver は同社の商標です。

Microsoft、Windows、Windows NT、および Windows ロゴは Microsoft Corporation の米国およびその他の国における商標です。

UNIX は、The Open Group がライセンスしている米国およびその他の国における登録商標です。

ActionMedia、LANDesk、MMX、Pentium および ProShare は Intel Corporation の米国およびその他の国における商標です。

Java およびすべての Java 関連の商標およびロゴは Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標または登録商標です。

用語集

この用語集には、本書で使用している重要語および省略語の定義を記載しています。探している用語が見つからない場合には、索引か、*IBM Dictionary of Computing* (New York: McGraw-Hill, 1994) を参照してください。

A

管理サーバー (administration server). MQSeries Workflow のうち、MQSeries Workflow システム内で管理機能を実行する構成要素。その機能には、MQSeries Workflow システムの起動 / 停止、エラー管理の実行、システム・グループの管理機能への参加が含まれる。

アクティビティー (activity). プロセス・モデルを構成するステップの 1 つ。プログラム・アクティビティー、プロセス・アクティビティー、ブロック・アクティビティーのいずれか。

アクティビティー情報メンバー (activity information member). アクティビティーの操作特性に関する定義済みデータ構造メンバー。

API. アプリケーション・プログラミング・インターフェース (application programming interface)。

アプリケーション・プログラミング・インターフェース (application programming interface). プログラムで MQSeries Workflow のワークフロー・マネージャーのサービスを要求するのに使うインターフェースとして MQSeries Workflow のワークフロー・マネージャーに用意されているもの。サービスは、同期的に提供される。

監査記録 (audit trail). データベースの中のリレーショナル表の 1 つで、プロセス・インスタンスの実行中に発生する各事象ごとに 1 つの項目が含まれているもの。

許可 (authorization). MQSeries Workflow におけるユーザーの権限レベルを決定する、ユーザー・スタッフ定義の属性。システム管理者はすべての機能を実行できる。

B

屈曲点 (bend point). コネクターが開始したり終了したり方向を変更したりする点。

ブロック・アクティビティー (block activity). 一群のアクティビティーから成る複合アクティビティー。制御およびデータ・コネクターに接続可能。ブロック・アクティビティーは Do-Until ループを実現するために使われる。ブロック・アクティビティー内のすべてのアクティビティーは、ブロック・アクティビティーの終了条件が満たされるまで処理される。*複合アクティビティー (composite activity)* も参照。

定義機能 (Buildtime). MQSeries Workflow の構成要素の 1 つで、グラフィカル・ユーザー・インターフェースが含まれており、ワークフロー・モデルの作成 / 保守、リソースの管理、およびシステム・ネットワークの定義のために使用するもの。

C

カーディナリティー (cardinality). (1) 関係の属性で、メンバー条件の数量について述べているもの。 1 対 1、1 対多、多対 1、多対多の 4 種類

カーディナリティーがある。 (2) データベース表の中の行数、またはデータベース表の 1 列の中の互いに異なる値の数。

下位組織 (child organization). 企業の管理単位の階層に含まれる組織のうち、上位組織のある組織。 1 つの下位組織に対して、上位組織が 1 つ、そして下位組織が複数個可能。上位組織とは、階層内で 1 レベル上位の組織のこと。*上位組織 (parent organization)* と対比。

クリーンアップ・サーバー (cleanup server). MQSeries Workflow の構成要素で、論理的にしか削除されていない情報を MQSeries Workflow 実行機能データベースから物理的に削除するためのもの。

複合アクティビティー (composite activity). いくつかのアクティビティーから構成されるアクティビティー。複合アクティビティーは、ブロック・アクティビティーでありバンドル・アクティビティーである。

コンテナー API (container API). MQSeries Workflow の制御下で実行されているプログラムがアクティビティーの入力および出力コンテナーからデータを入手したり、アクティビティーの出力コンテナーにデータを格納したりするための MQSeries Workflow API。

制御コネクター (control connector). プロセスに含まれる 2 つのノードの間で制御の潜在的な流れを定義するもの。実際の制御の流れは、制御コネクターに関連付けられた分岐条件の真理値によって実行時に決まる。

コーディネーター (coordinator). 役割を調整する人として指定された担当者に対して自動的に割り当てられる定義済みの役割。

D

データ・コネクター (data connector). コンテナーとコンテナーの間のデータの流れを定義するもの。

データ・コンテナー (data container). アクティビティーまたはプロセスの入力データおよび出力データの記憶域。*入力コンテナー (input container)* および*出力コンテナー (output container)* を参照。

データ・マッピング (data mapping). データ・コネクターについて、関連するソース・コンテナーのフィールドと関連するターゲット・コンテナーのフィールドの対応を指定するもの。

データ構造 (data structure). データ構造メンバーの集合からなる名前付きエンティティー。入力および出力コンテナーはデータ構造への参照により定義され、参照先のデータ構造タイプのレイアウトを採用する。

データ構造メンバー (data structure member). データ構造を構成する変数の 1 つ。

省略時制御コネクター (default control connector). プロセス・ダイアグラムで標準の制御コネクターを視覚的に表現するもの。他の制御パスが有効でなければ、制御はこのコネクターを流れる。

ドメイン (domain). メタモデルが同じで、同じスタッフ情報とトポロジー情報を共用する MQSeries Workflow システム・グループの集合。ドメイン内の構成要素間の通信は、メッセージ・キューイングを使ってなされる。

動的スタッフ割り当て (dynamic staff assignment). 役割、組織、またはレベルなどの基準を指定することによりスタッフをアクティビティーに割り当てる手法。アクティビティーが作動可能になったときに、選択基準に合うユーザーが、作業対象のアクティビティーを受け取る。*レベル (level)*、*組織*

(organization)、*プロセス管理者 (process administrator)*、および*役割 (role)* も参照。

E

終了アクティビティー (end activity). 出ていく制御コネクターがないアクティビティー。

実行サーバー (execution server). MQSeries Workflow の構成要素で、実行時にプロセス・インスタンスの処理を実行するもの。

終了条件 (exit condition). アクティビティーが完了したかどうかを指定する論理式。

搬出 (export). MQSeries Workflow データベースから情報を検索し、それを MQSeries Workflow 定義言語 (FDL) または HTML 形式で使用できるようにする MQSeries Workflow のユーティリティー・プログラム。*搬入 (import)* と対比。

F

固定メンバー (fixed member). 現行アクティビティーに関する情報を提供する定義済みデータ構造メンバー。固定メンバーの値は、MQSeries Workflow のワークフロー・マネージャーによって設定される。

MQSeries Workflow 定義言語 (FDL) ((FDL) MQSeries Workflow Definition Language). MQSeries Workflow システム・グループ相互間で MQSeries Workflow 情報を交換するための言語。この言語は MQSeries Workflow の搬出 / 搬入機能で使用される。この言語にはスタッフ、プログラム、データ構造、およびトポロジーに関するワークフロー定義が含まれる。これにより、非 MQSeries Workflow 構成要素が MQSeries Workflow と対話することが可能になる。*搬出 (export)* および*搬入 (import)* も参照。

フォーク・アクティビティー (fork activity). 複数の制御コネクターの元になっているアクティビティー。

フォーム (form). ロータス ノーツにおいて、フォームによって、ロータス ノーツに情報を入力する方法、およびその情報の表示と印刷の方法が制御される。

式 (formula). ロータス ノーツにおいて、たとえばデータベースから文書を選択したり表示用の値を計算したりするのに使われる数式。

完全修飾名 (fully-qualified name). 完全に修飾された名前。つまり、その名前が指す構造メンバーより上のすべての名前を階層順に含み、さらにそのメンバー自体の名前を付けたもの。

I

搬入 (import). MQSeries Workflow 定義言語 (FDL) 形式で情報を受け取り、それを MQSeries Workflow データベースに入れる MQSeries Workflow ユーティリティー・プログラム。*搬出 (export)* と対比。

入力コンテナー (input container). アクティビティーまたはプロセスへの入力に使用されるデータのための記憶域。*ソース (source)* および*データ・マッピング (data mapping)* も参照。

L

レベル (level). MQSeries Workflow データベースの各担当者に割り当てられる 0〜9 の数値。定義機能でスタッフを定義する担当者は、ランクや経験などの意味をこれらの数値に割り当てることができる。レベルは、アクティビティーを担当者に動的に割り当てるために使用できる基準の 1 つ。

ローカル・ユーザー (local user). スタッフ解決において、親プロセスと同じシステム・グループにホーム・サーバーがあるユーザーのこと。

ローカル・サブプロセス (local subprocess). 親プロセスと同じ MQSeries Workflow システム・グループで処理されるサブプロセス。

論理式 (logical expression). 演算子とオペランドで構成され、評価結果が真、偽、または整数になる式。 (0 でない整数は偽に相当する。) *終了条件 (exit condition)* および*分岐条件 (transition condition)* も参照。

M

マネージャー (manager). 組織の長として定義された担当者に対して自動的に割り当てられる定義済みの役割。

メッセージ・キューイング (message queuing). ソフトウェア構成要素相互間の通信で非同期メッセージを使用する通信方法。

N

ナビゲーション (navigation). 1 つのプロセス内で、完了したアクティビティーから、それ以降のアクティビティーへ移ること。その経路は、制御コネクターとそれに関連する分岐条件によって、またアクティビティーの開始条件によって決まる。*制御コネクター (control connector)*、*終了条件 (exit condition)*、*分岐条件 (transition condition)*、および*開始条件 (start condition)* も参照。

ノード (node). (1) プロセス・ダイアグラム内のアクティビティーの総称。 (2) MQSeries Workflow システムに対してホストとなるオペレーティング・システムのイメージ。

通知 (notification). MQSeries Workflow の機能の 1 つで、プロセスまたはアクティビティーが指定した期間内に完了しない場合に、指定された担当者に通知するもの。

通知作業項目 (notification work item). アクティビティーまたはプロセスの通知を表す作業項目。

O

組織 (organization). 企業の管理単位。組織は、アクティビティーを担当者に動的に割り当てるために使用できる基準の 1 つ。*下位組織 (child organization)* および*上位組織 (parent organization)* を参照。

出力コンテナー (output container). 他のアクティビティーが使用するために、または条件を評価するために、アクティビティーまたはプロセスによって生成されるデータ用の記憶域。シンク *(sink)* も参照。

P

上位組織 (parent organization). 企業の管理単位の階層に含まれる組織のうち、1 つまたは複数の下位組織のある組織。下位組織は、階層内でその上位組織よりも 1 レベル下位の組織。*下位組織 (child organization)* と対比。

親プロセス (parent process). あるプロセスをサブプロセスとして開始したプロセス・アクティビティーが含まれるプロセス・インスタンス。

パターン・アクティビティー (pattern activity). バンドル・アクティビティー内のアクティビティーのうち、実行時に複数のインスタンス (パターン・アクティビティー・インスタンス) が作成されるその作成元となる単一の単純なアクティビティー。

担当者 (person (複数形 : people)). MQSeries Workflow データベースで定義されている企業内のスタッフのメンバー。

事前定義データ構造メンバー (predefined data structure member). MQSeries Workflow で事前に定義されていて、ユーザー・アプリケーションと MQSeries Workflow 実行機能の間の通信で使用されるデータ構造メンバー。

プロセス (process). プロセス・モデル (process model) およびプロセス・インスタンス (process instance) の同義語。実際の意味は文脈によって異なる。

プロセス・アクティビティー (process activity). プロセス・モデルの一部であるアクティビティー。プロセス・アクティビティーを実行すると、プロセス・モデルのインスタンスが作成また実行される。

プロセス管理者 (process administrator). 特定のプロセス・インスタンスの管理者。管理者にはプロセス・インスタンスに対してすべての操作を実行する権限がある。管理者は、スタッフ解決および通知のターゲットにもなる。

プロセス・カテゴリー (process category). プロセス・インスタンスに対していろいろな機能を実行できるユーザーの集合を制限するために、プロセス定義担当者がそのプロセス・モデルに対して指定できる属性。

プロセス定義 (process definition). *プロセス・モデル (process model)* の同義語。

プロセス・ダイアグラム (process diagram). プロセス・モデルのプロパティーを示すプロセスのグラフィック表示。

プロセス・インスタンス (process instance). MQSeries Workflow 実行機能で実行されるプロセスのインスタンス。

プロセス・インスタンス・リスト (process instance list). ユーザー定義の基準に従って選択またソートされるプロセス・インスタンスの集合。

プロセス・インスタンス・モニター (process instance monitor). MQSeries Workflow クライアント構成要素で、特定のプロセス・インスタンスの状態をグラフィック表示したもの。

プロセス管理 (process management). プロセス・インスタンスに関連した MQSeries Workflow 実行機能タスク。プロセス・インスタンスの作成、開始、一時中断、再開、中止、再始動、および削除で構成される。

プロセス・モデル (process model). プロセス・モデルの中に表示されるプロセスの集合。それらのプロセスは、プロセス・ダイアグラムの中にグラフィック形式で表される。プロセス・モデルには、プロセス・アクティビティーと関連付けられたスタッフ、プログラム、およびデータ構造の定義が含まれる。プロセス・モデルをプロセス・テンプレートに搬入および変換したなら、そのプロセス・テンプレートは繰り返し実行できる。*ワークフロー・モデル (workflow model)* および*プロセス定義 (process definition)* と同義。

プロセス・モニター API (process monitor API). アプリケーションでプロセス・インスタンス・モニターの機能を実現するためのアプリケーション・プログラミング・インターフェース。

プロセス関連データ (process-relevant data). プロセス・インスタンス内のアクティビティーの順序を制御するのに使われるデータ。

プロセス状況 (process status). プロセス・インスタンスの状況。

プロセス・テンプレート (process template). プロセス・インスタンスを作成する基礎として使うことのできるプロセス・モデルの固定フォーム。 MQSeries Workflow 実行機能において変換済みのフォーム。*プロセス・インスタンス (process instance)* も参照。

プロセス・テンプレート・リスト (process template list). ユーザー定義の基準に従って選択またソートされているプロセス・テンプレートの集合。

プログラム (program). プログラム・アクティビティーの実装またはサポート・ツールとなる、コンピューター・ベースのアプリケーション。プ

ログラム・アクティビティーは、MQSeries Workflow プログラム登録の中でプログラムに関連付けられている論理名を使用して、実行可能プログラムを参照する。*プログラム登録 (program registration)* も参照。

プログラム・アクティビティー (program activity). 登録済みプログラムにより実行されるアクティビティー。このアクティビティーを開始すると、プログラムが呼び出される。*プロセス・アクティビティー (process activity)* と対比。

プログラム実行エージェント (program execution agent). MQSeries Workflow の構成要素で、プログラム・アクティビティー (.EXE ファイルや .DLL ファイルなど) の実装を管理するもの。

プログラム登録 (program registration). プログラムを MQSeries Workflow に登録して、それが MQSeries Workflow によって実行される際にプログラムの管理のために必要な情報がそろっているようにすること。

R

役割 (role). スタッフ・メンバーに対して定義される責任。役割は、アクティビティーを担当者に動的に割り当てるために使用できる基準の 1 つ。

S

スケジューリング・サーバー (scheduling server). 一時中断している作業項目の再開や期限切れプロセスの検出などのアクションのスケジュールを時刻事象に基づいて立てるための MQSeries Workflow 構成要素。

サーバー (server). MQSeries Workflow システムを構成するサーバー (実行サーバー、管理サーバー、スケジューリング・サーバー、およびクリーンアップ・サーバー)。

シンク (sink). プロセスまたはブロック・アクティビティーの出力コンテナーを表すシンボル。

ソース (source). プロセスまたはブロック・アクティビティーの入力コンテナーを表すシンボル。

特定資源割り当て (specific resource assignment). ユーザー ID を指定することによって、資源をプロセスまたはアクティビティーに割り当てる手法。

標準クライアント (standard client). プロセス・インスタンスの作成と制御、ワークリストおよび作業項目の処理、およびログオン・ユーザーの個人データの操作を実行する MQSeries Workflow 構成要素。

開始アクティビティー (start activity). 入ってくる制御コネクターがないアクティビティー。

開始条件 (start condition). 入ってくるすべての制御コネクターが評価された後、それらの制御コネクターが接続されているアクティビティーを開始するかどうかを判別する条件。

サブプロセス (subprocess). プロセス・アクティビティーによって開始されるプロセス・インスタンス。

代行者 (substitute). アクティビティーの割り当て先の担当者が不在であると宣言された場合に、自動的にそのアクティビティーの転送先となる担当者。

サポート・ツール (support tool). エンド・ユーザーがアクティビティーを完了するために MQSeries Workflow MQSeries Workflow クライアントのワークリストから開始するプログラム。

記号参照 (symbolic reference). アクティビティーの記述テキストまたはプログラム登録のコマンド行パラメーターに含まれている特定のデータ項目、プロセス名、またはアクティビティー名への参照。記号参照は、データ項目の完全修飾名ま

たは _PROCESS か _ACTIVITY のいずれかを、パーセント記号 (%) で囲んだもので表される。

システム (system). MQSeries Workflow ドメインの中での MQSeries Workflow の最小単位。これは、一群の MQSeries Workflow サーバーで構成される。

システム・グループ (system group). 同じデータベースを共用する MQSeries Workflow システムの集合。

システム管理者 (system administrator). (1) すべての許可を付与された定義済みの役割。 1 つの MQSeries Workflow システムの中で 1 人の担当者に割り当てることができる。 (2) コンピューターのインストール時にコンピューター・システムの設計、制御、および使用の管理を担当する担当者。

T

最上位プロセス (top-level process). サブプロセスではないプロセス・インスタンスのうち、ユーザーのプロセス・インスタンス・リストまたはアプリケーション・プログラムから開始するプロセス・インスタンス。

分岐条件 (transition condition). 条件付き制御コネクターに関連付けられている論理式。これが指定されている場合、制御がそれに関連する制御コネクターを通るためには、これが真でなければならない。*制御コネクター (control connector)* も参照。

変換 (translate). プロセス・モデルを実行機能のプロセス・テンプレートに変換するアクション。

U

ユーザー ID (user ID). MQSeries Workflow のユーザーを固有に識別する英数字ストリング。

V

検証 (verify). プロセス・モデルが完全かどうかを調べるアクション。

W

ワークフロー (workflow). ある企業のビジネス・プロセスに従って実行される一連のアクティビティー。

Workflow Management Coalition (WfMC). ワークフロー管理システムのベンダーとユーザーからなる非営利組織。この組織の目的は、ワークフロー管理システムのためのワークフロー規格を奨励し、さまざまな実装の間の相互操作性を実現すること。

ワークフロー・モデル (workflow model). *プロセス・モデル (process model)* の同義語。

作業項目 (work item). プロセス・インスタンス内のアクティビティーのコンテキストで処理される作業の表示。

ユーザーの作業項目セット (work item set of a user). 1 ユーザーに割り当てられたすべての作業項目。

ワークリスト (worklist). ユーザーに割り当てられて、ワークフロー管理システムから取り出される作業項目のリスト。

ワークリスト・ビュー (worklist view). ワークリストの属性であるフィルター基準に従って、ユーザーの作業項目セットから選択される作業項目および通知のリスト。そのワークリストにソート基準が指定されていれば、ソートすることもできる。

参照文献

ここに示されている資料を注文する際には、IBM 担当員または IBM 事業所にご連絡ください。

MQSeries Workflow 資料

ここに示されている資料は、MQSeries Workflow のライブラリーに含まれています。

- *IBM MQSeries Workflow: List of Workstation Server Processor Groups* (GH12-6357) は、MQSeries Workflow のプロセッサー・グループのリストです。
- *IBM MQSeries Workflow: 概説およびアーキテクチャー* (GH88-7348) では、MQSeries Workflow の基本概念について説明しています。また MQSeries Workflow のアーキテクチャーやそれぞれの構成要素の関係について説明しています。
- *IBM MQSeries Workflow: 定義機能の開始* (SH88-7354) では、MQSeries Workflow の定義機能の使用方法について説明しています。
- *IBM MQSeries Workflow: 実行機能の開始* (SH88-7349) では、クライアントの実行方法について説明しています。
- *IBM MQSeries Workflow: プログラミングの手引き* (本書、SH88-7352) では、アプリケーション・プログラミング・インターフェース (API) について説明しています。
- *IBM MQSeries Workflow: インストールの手引き* (SH88-7350) では、MQSeries Workflow のインストールとカスタマイズについての情報および手順について説明しています。
- *IBM MQSeries Workflow: 管理の手引き* (SH88-7351) では、MQSeries Workflow システムの管理方法について説明しています。

関連資料

- *IBM MQSeries: アプリケーション・プログラミングの手引き* (SC88-7253)
- *IBM MQSeries: システム管理の手引き* (SC88-7776)
- *Frank Leymann, Dieter Roller, Production Workflow: Concepts and Techniques* (New Jersey: Prentice Hall PTR, 1999)
- *Frank Leymann, Dieter Roller, "Workflow-based Applications", IBM Systems Journal 36, no. 1 (1997): 102-123*。次のインターネット URL での参照も可能です: http://www.almaden.ibm.com/journal/sj/361/leymann.html
- *Workflow Handbook 1997,* (WfMC との共同出版、 Peter Lawrence 編集)。
- *Extensible Markup Language (XML) 1.0,* W3C Recommendation February 1998: http://www.w3.org/TR/REC-xml

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

［サ行］

［タ行］

［ナ行］

［ハ行］

［マ行］

［ヤ行］

［ラ行］

［ワ行］

［数字］

A

D

E

F

M

X

部品番号: CT8L4JA
プログラム番号: 5697-FM3

Printed in Japan

(1P) P/N: CT8L4JA

SH88-7352-04

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12

Spine information:

IBM MQSeries Workflow プログラミングの手引き

バージョン 3.2.2