PeopleSoft

EnterpriseOne 8.9
開発ツール
PeopleBook

2003 年 9 月

PeopleSoft EnterpriseOne 8.9
開発ツールPeopleBook
SKU AC89EOD0309

Copyright 2003 PeopleSoft, Inc. All rights reserved.

本書に含まれるすべての内容は、PeopleSoft, Inc.（以下、「ピープルソフト」）が財産権を有する機密情報です。すべての内容は著作権法により保護されており、該当するピープルソフトとの機密保持契約の対象となります。 本書のいかなる部分も、ピープルソフトの書面による事前の許可なく複製、コピー、転載することを禁じます。これには電子媒体、画像、複写物、その他あらゆる記録手段を含みます。

本書の内容は予告なく変更される場合があります。ピープルソフトは本書の内容の正確性について責任を負いません。 本書で見つかった誤りは書面にてピープルソフトまでお知らせください。

本書に記載されているソフトウェアは著作権によって保護されており、このソフトウェアの使用許諾契約書に基づいてのみ使用が許諾されます。この使用許諾契約書には、開示情報を含むソフトウェアと本書の使用条件が記載されていますのでよくお読みください。

PeopleSoft、PeopleTools、PS/nVision、PeopleCode、PeopleBooks、PeopleTalk、Vantiveはピープルソフトの登録商標です。Pure Internet Architecture、Intelligent Context Manager、The Real-Time Enterpriseはピープルソフトの商標です。 その他すべての会社名および製品名は、それぞれの所有者の商標である場合があります。 ここに含まれている内容は予告なく変更されることがあります。

オープンソースの開示

この製品には、Apache Software Foundation (http://www.apache.org/) が開発したソフトウェアが含まれています。 Copyright (c) 1999-2000 The Apache Software Foundation. All rights reserved. このソフトウェアは「現状のまま」提供されるものとし、特定の目的に対する商品性および適格性の黙示保証を含む、いかなる明示または黙示の保証も行いません。 Apache Software Foundationおよびその供給業者は、損害の発生原因を問わず、責任の根拠が契約、厳格責任、不法行為（過失および故意を含む）のいずれであっても、また損害の可能性が事前に知らされていたとしても、このソフトウェアの使用によって生じたいかなる直接的損害、間接的損害、付随的損害、特別損害、懲罰的損害、結果的損害に関しても一切責任を負いません。これらの損害には、商品またはサービスの代用調達、使用機会の喪失、データまたは利益の損失、事業の中断が含まれますがこれらに限らないものとします。

ピープルソフトは、いかなるオープンソースまたはシェアウェアのソフトウェアおよび文書の使用または頒布に関しても一切責任を負わず、これらのソフトウェアや文書の使用によって生じたいかなる損害についても保証しません。

目次

J.D. Edwards 開発ツール

開発ツールとは、統合アプリケーション開発のツール・セットです。これらのツールを使用すると、フォームやレポートなどの完全な対話型アプリケーションやバッチ・アプリケーションを開発することができます。また、開発プロセスを簡易化し、アプリケーションの作成に必要なプログラミング量を軽減します。

開発ツールでは、実証済みの J.D. Edwards メソドロジーと Microsoft Windows の使いやすいインターフェイスを使用してクライアント/サーバー環境用のアプリケーションを作成できます。

略語について

ここでは、J.D. Edwards ソフトウェアで使用されている略語について説明します。

APPL	アプリケーション
BDA	ビジネス・ビュー設計ツール
BSFN	ビジネス関数
BSVW	ビジネス・ビュー
CRP	カンファレンス・ルーム・パイロット
DD	データ辞書
DLL	ダイナミック・リンク・ライブラリ
DS または DSTR	データ構造体
ER	イベント・ルール
FDA	フォーム設計ツール
NER	イベント・ルール・ビジネス関数
OCM	オブジェクト構成マネージャ
OL	オブジェクト・ライブラリアン
OMW	オブジェクト管理ワークベンチ
QBE	例示照会
RDA	レポート設計ツール

SAR	ソフトウェア・アクション・リクエスト
Specs	スペック
TAM	テーブル・アクセス管理
TBLE	テーブル
TC	テーブル変換
TDA	テーブル設計ツール
TER	テーブル・イベント・ルール
UBE	ユニバーサル・バッチ・エンジン
UTB	ユニバーサル・テーブル・ブラウザ
WF	ワークフロー

オブジェクトとアプリケーションの概要

J.D. Edwards 開発ツールを使用してオブジェクトを作成し、オブジェクトを使用してアプリケーションを構築します。次の 2 つのセクションでは、オブジェクトとアプリケーションについて詳しく説明します。

オブジェクトについて

通常、オブジェクトとは、データだけでなく、そのデータの操作に使用する構造や機能を含む自己完結型のエンティティです。J.D. Edwards ソフトウェアでのオブジェクトとは、そのツールによって作成されたソフトウェアのスペックに基づく、再利用可能なエンティティを指しています。

スペックとは J.D. Edwards オブジェクトの完全な記述です。各オブジェクトには独自のスペックがあり、サーバー側とワークステーション側の両方に保管されます。スペックによっては 1 つのスペックが異なるタイプのオブジェクトを記述します。たとえば、データ構造体スペックには、ビジネス関数データ構造体、処理オプション構造体、メディア・オブジェクト構造体があります。

J.D. Edwards のアーキテクチャはオブジェクト・ベースです。つまり、個々のソフトウェア・オブジェクトはすべてのアプリケーションの基礎であり、開発者はそのオブジェクトを複数のアプリケーションで再利用することができます。このようなオブジェクトの利用法（より小さなコンポーネントに分割されるアプリケーション）によって、J.D. Edwards は真の分散処理を提供することができます。また、開発者は、J.D. Edwards 開発ツールを使用することで、これらのオブジェクトを作成できます。

J.D. Edwards のオブジェクトには次のものがあります。

- バッチ・アプリケーション
- ビジネス関数（部品化されたルーチン）
- ビジネス・ビュー
- データ辞書項目

- データ構造体
- イベント・ルール
- 対話型アプリケーション
- メディア・オブジェクト
- テーブル

アプリケーションについて

アプリケーションとは、特定のタスクを実行するオブジェクトの集合です。開発ツールを使用して、次の標準的な関連アプリケーション・グループを構築します。

- 設計、エンジニアリング、建設
- 流通
- エネルギーおよび化学産業向けシステム
- 会計
- 人事
- 製造
- テクニカル

上記のアプリケーションは、いずれも開発ツールを使用して生成されているので、共通のユーザー・インターフェイスで利用できます。また、これらのアプリケーションには、対話型アプリケーションとバッチ・アプリケーションの両方があります。たとえば、次のようなアプリケーションが存在します。

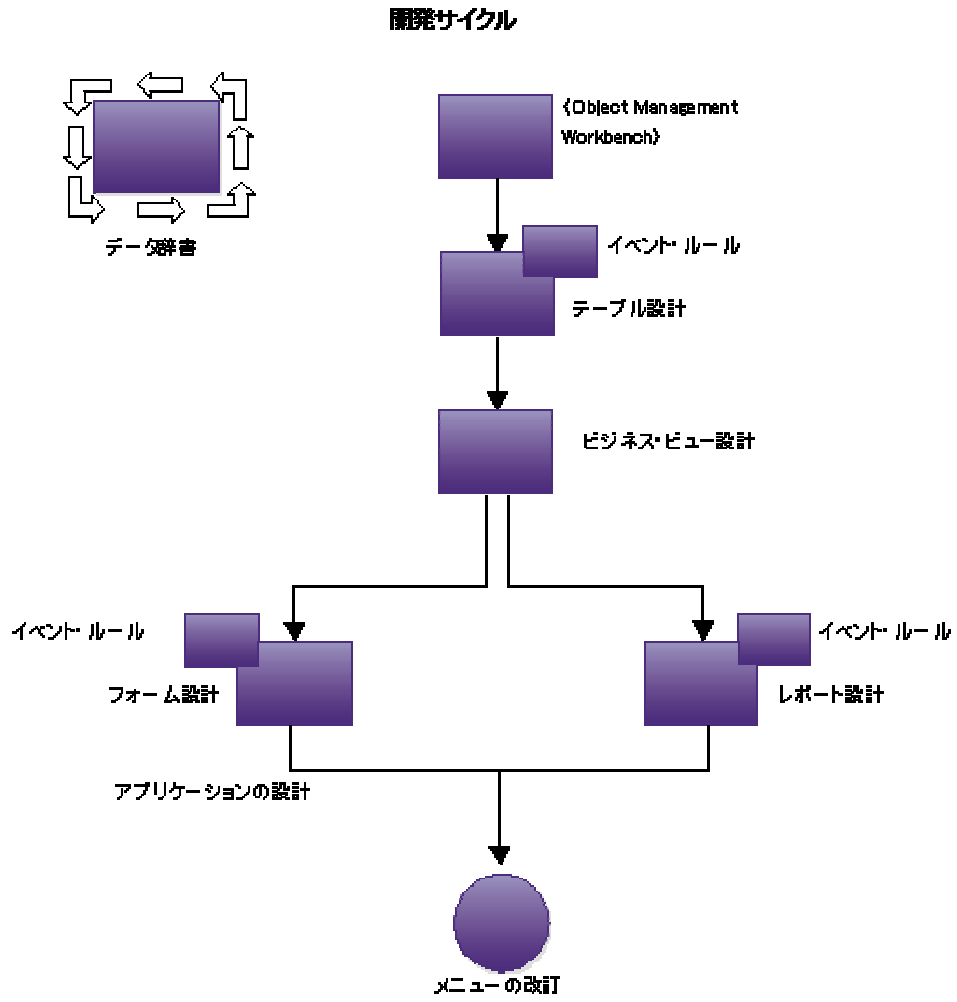
- 住所録の改訂
- 受注オーダー入力
- 総勘定元帳への転記
- 残高試算表報告書

アプリケーション構築の概要

このツールを使用すると、アプリケーションを構築することができます。アプリケーションの作成にすべてのツールを使用する必要はありませんが、この作業は必ず〈Object Management Workbench（オブジェクト管理ワークベンチ）〉から始めます。たとえば、個々のデータ項目の追加や変更を必要としない場合があります。その場合、〈Object Management Workbench〉から〈Table Design（テーブル設計）〉に進むことができます。アプリケーションに含めるすべてのデータ項目が、既存の 1 つ以上のデータベース・テーブルに含まれている場合は、テーブルの設計手順をとばして「ビジネス・ビューの設計」に進むことができます。

開発サイクルの理解

次の図は、各種ツール間の関係と共にソフトウェアの開発サイクルを示しています。



J.D. Edwards でのオブジェクト保管の概要

J.D. Edwards では、オブジェクトを次の 2 か所に保管します。

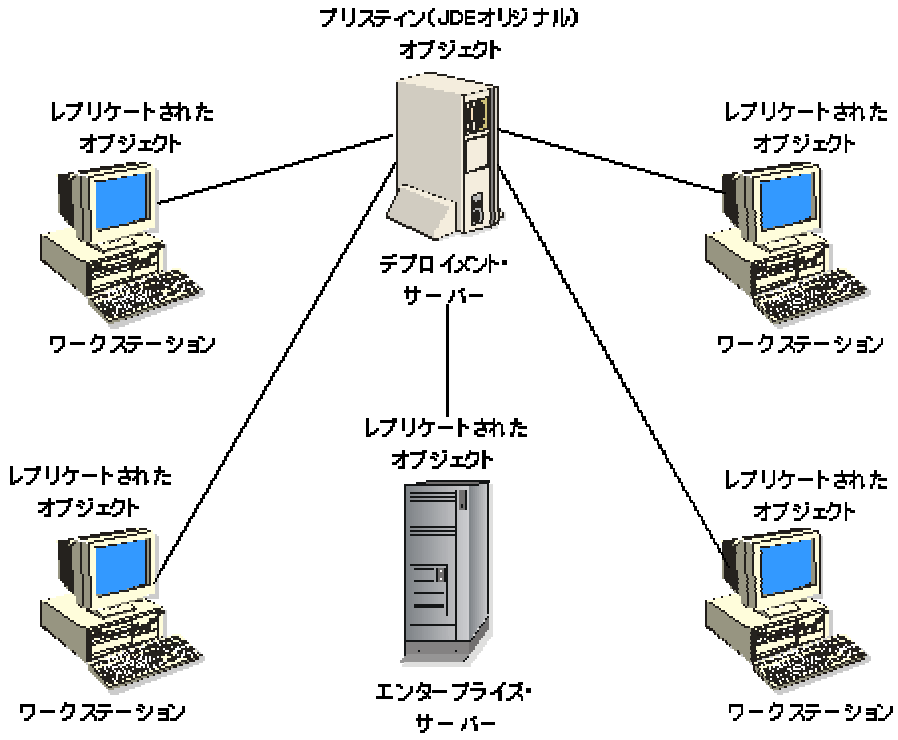
- セントラル・ストレージ・サーバー。セントラル・オブジェクトを保管します。

  セントラル・オブジェクトはセントラル・ロケーションに保管され、このセントラル・ロケーションから配布できます。スペックなどのオブジェクトは、リレーショナル・データベースに保管されます。DLL やソース・コードなどの他のオブジェクトは、ファイル・サーバーに保管されます。

- J.D. Edwards を実行し、レプリケートされたオブジェクトを保管する任意のマシン（ワークステーションやサーバーなど）

セントラル・オブジェクトの複製（レプリケート）セットは、J.D. Edwards を実行する個々のクライアント側とサーバー側に配置しておく必要があります。パス・コードは、オブジェクトが置かれているディレクトリを示します。

J.D. Edwardsによるオブジェクトの保管



サーバーとクライアント間でオブジェクトを移動するには、個別のオブジェクト・タイプに適した設計ツールを使用し、「チェックイン」および「チェックアウト」を使用します。オブジェクトを作成すると、最初はクライアント側に保管されます。オブジェクトをサーバー側にチェックインしなければ、その作成者しか利用することができません。サーバーにチェックインすると、オブジェクトは、他のユーザーがチェックアウトできるようになります。

オブジェクトをチェックアウトすると、すべてのオブジェクト・スペック・レコード（J.D. Edwards オブジェクトを定義するデータの集合）が、サーバーからワークステーション側に複製（レプリケート）されます。

J.D. Edwards 開発ツールセットの概要

J.D. Edwards ツール・セットは、対話型アプリケーションやバッチ・アプリケーションの作成に使用する複数のツールから構成されています。ここでは、各種ツールについて詳しく説明します。

Object Management Workbench（オブジェクト管理ワークベンチ）

〈Object Management Workbench〉は、すべてのオブジェクトを管理します。開発者は、〈Object Management Workbench〉を使用して、オブジェクトをセントラル・ロケーションの開発環境からチェックアウトし、自分のデスクトップにコピーします。これで、J.D. Edwards ツールを使用してオブジェクトを変更できます。変更した後は、オブジェクトをチェックインして、他のユーザーがオブジェクトにアクセスできるようにします。開発者は、〈Object Management Workbench〉を経由してのみ J.D. Edwards ツールにアクセスできます。

データ辞書

辞書に用語の定義が含まれているのと同様に、J.D. Edwards のデータ辞書はデータ項目の定義と属性を含むセントラル・リポジトリです。属性により、データ項目に関する次のような処理方法が決まります。

- レポートやフォームの表示方法（小数点以下表示桁数やデフォルト値を表示するかどうかなど）
- アプリケーションでのデータ入力時の検証
- カラム（列）とロー（行）への記述の割当方法
- フィールド・センシティブ・ヘルプのテキストの表示方法
- テーブルへの保管方法

データ辞書はアクティブであり、データ辞書に入力した変更はアプリケーションに自動的に反映されるので、変更を表示するためにソフトウェアをリコンパイルする必要がありません。

テーブル設計

リレーショナル・データベース・テーブルは、アプリケーションが使用するデータを保管します。テーブルは、1 アプリケーションに対し 1 つ以上作成できます。テーブルを作成するには、テーブル用のデータ項目を選択し、データの抽出や更新用のインデックスとして使用するためのキー・フィールドを割り当てます。

〈Table Design Aid（テーブル設計ツール）〉は、.H ファイルを作成します。このファイルをソースにチェックインするには、OMW を使用します。

ビジネス・ビュー設計

「ビジネス・ビュー」とは、アプリケーションとデータをリンクする論理的な媒体です。ビジネス・ビューでは、1 つ以上のテーブルからアプリケーションに使用するデータ項目を定義します（複数テーブルの結合など）。テーブルを作成した後、ビジネス・ビューに含めるデータ項目を 1 つ以上のテーブルから選択します。ビジネス・ビューを使用して、通常はアプリケーションで必要なカラムだけを選択します。これにより、ネットワーク上で移動する必要のあるデータ量を縮小して、パフォーマンスを向上させることができます。たとえば、従業員のすべてのデータを含むテーブルから、従業員の氏名と住所だけを含むビジネス・ビューを作成できます。

フォーム設計

対話型アプリケーションでは、ユーザーが(ビジネス・ビューに基づいて)データベースにアクセスできます。通常はフォームを使用して、データの追加、修正、表示に使用する対話型アプリケーションにアクセスします。

アプリケーションを作成するには、そのアプリケーションに必要なフォームのタイプを決め、各フォームにビジネス・ビューを関連付けます。フォームを設計するには、グリッド、編集フィールド、プッシュ・ボタン、ラジオ・ボタンなどのコントロールを追加します。

標準フォーム・タイプには、必要なプロセスの大半がすでに組み込まれているものがあります。たとえば、次のようなプロセスが組み込まれています。

Find/Browse (検索/表示)	〈Work with Sales Orders(受注オーダーの処理)〉など、照会フォームに使用します。
Fix/Inspect (修正/検査)	単一レコードの追加または変更に使用します。〈Sales Order Header(受注オーダー見出し)〉は、修正/検査フォームの一例です。
Header Detail (見出し詳細)	複数のレコード(特に正規化テーブル上のレコード)を同時に変更するために使用します〈Sales Order Detail(受注オーダー詳細)〉は、見出し詳細フォームの一例です。
Headerless Detail (見出しなし詳細)	正規化されていないテーブルの複数のレコードを変更するために使用します。〈Voucher Entry(伝票入力)〉は、見出しなし詳細フォームの一例です。
Search and Select (検索/選択)	データをビジュアル・アシストのフィールドに自動的に取り出すために使用します。
Parent/Child (親/子)	親子関係を持つ複数のレコードをエクスプローラのようなツリー形式で表示するときに使用します。〈Bill of Material(部品表)〉は、親/子フォームの一例です。
メッセージ	メッセージや要求アクションを表示するために使用します。〈Delete Confirmation(削除の確認)〉は、メッセージ・フォームの一例です。

レポート設計

〈Report Design(レポート設計)〉は、すべてのレポートとバッチ処理を作成するときに使います。〈Report Design〉では、フォームではなく各セクションを設計します。レポートを作成するには、レポートに表示するデータを指定します。そして、ビジネス・ロジックを規定するイベント・ルールを関連付け、フォーマットを制御する処理オプション、ページ区切り、合計機能、レポートによるデータの処理方法を指定します。レポートとバッチ処理の例を次に示します。

- 〈売上更新〉
- 〈総勘定元帳への転記〉
- 〈コスト・センター別残高試算表〉
- 〈請求書〉
- 〈テーブル変換〉
- 〈財務レポート作成〉

Business Function Design（ビジネス関数の設計）

「ビジネス関数」とは、イベント・ルールからコールできる再利用可能な「カプセル化ルーチン」です。ビジネス関数のコードは、業界の標準的な第 3 世代言語で記述することができます。〈Business Function Design（ビジネス関数の設計）〉は、フィールドでの特殊編集など、アプリケーションで必要な特殊タスクのバックグラウンド処理を実現するビジネス関数を作成するために使用します。

イベント・ルール

「イベント・ルール」とは、論理ステートメントです。イベント・ルールを作成して各イベントに関連付けます。そのイベントとは、フォームを開く、あるフィールドからカーソルを移動する、次の行に進む、レポート上で改頁するなど、アプリケーションで生じるアクティビティを指します。ユーザーやシステムがイベントを発生させると、イベント・ルールが応答します。イベントは、特定のフィールド、フォーム全体、グリッド、またはレポートのセクションなどのコントロールに関連付けられます。

イベント・ルールを使用すると、多くのプログラミング言語のように難しい構文を使用しなくても、複雑なビジネス・ロジックを作成することができます。たとえば、次のようなビジネス・ロジックを作成することができます。

- 算術演算を実行する。
- テーブル I/O（フェッチ、挿入、削除）を実行する。
- あるフォーム上のフィールドから他のフォーム上のフィールドにデータを渡す。
- 2 つのフォームまたはアプリケーションを接続する。
- システム関数を使用してコントロールを表示非表示にする。
- IF/WHILE 条件と ELSE 条件を評価する。
- フィールドの式に値を代入する。
- 指示どおりに変数（ワーク・フィールド）を作成する。
- 対話型アプリケーションの終了後にバッチ処理を実行する。
- ビジネス関数またはシステム関数（発行済み API）を関連付ける。
- ワークフロー処理を開始する。

イベント・ルールには、次の 2 つのタイプがあります。

- ビジネス関数イベント・ルール
- 埋込みイベント・ルール

イベント・ルール・ビジネス関数

ビジネス関数イベント・ルールは、C プログラミング言語ではなくイベント・ルールを通じて作成され、カプセル化された、再利用可能なビジネス・ロジックです。ビジネス関数イベント・ルールは、オブジェクトとして保管され、コンパイルされます。

埋込みイベント・ルール

埋込みイベント・ルールは、特定のテーブル、対話型アプリケーション、またはバッチ・アプリケーションに固有のルールで、再利用できません。たとえば、フォームからフォームへの呼出しを作成したり、処理オプションの値に基づいてフィールドを非表示にしたり、ビジネス関数を呼び出したりするものがあります。埋込みイベント・ルールは、アプリケーションのイベント・ルール（対話型またはバッチ）、あるいはテーブル・イベント・ルールに含めることができます。各種イベント・ルールの説明については、次の表を参照してください。

アプリケーション・イベント・ルール（対話型またはバッチ）	特定のアプリケーションに固有のビジネス・ロジックを追加することができます。対話型アプリケーションでは〈Form Design（フォーム設計）〉を介してイベント・ルールを関連付けますが、バッチ・イベント・ルールには〈Report Design（レポート設計）〉を使用します。
テーブル・イベント・ルール	J.D. Edwards データベース・トリガーを作成したり、テーブルに関連付けるルールを作成し、テーブル設計イベント・ルールを使用して実行することができます。テーブルに関連付けられているロジックは、アプリケーションがそのデータベースに対するアクションを起動するたびに実行されます。たとえば、参照整合性を維持するために、親が削除されたときに子をすべて削除するマスター・テーブルにルールを関連付けることもできます。親/子のロジックはテーブル・レベルにあるため、テーブルの削除を開始するアプリケーションに組み込む必要はありません。

処理オプション

処理オプションは対話型/バッチ・アプリケーションによるデータの処理方法を制御します。1つの対話型/バッチ・アプリケーションは複数のバージョンをもつことができます。通常、各バージョンは相互に異なっています。たとえば、処理オプションを使用して次のような内容を指示することができます。

- アプリケーションで表示するフォームの順序を指定
- デフォルト値の設定
- 〈Sales Order Entry（受注オーダー入力）〉の通貨やキットの扱いなど、特殊処理の有効化/無効化

基礎

「基礎」では、アプリケーションを開発する際に知っておく必要がある基本概念とツールについて説明します。また、アプリケーションを実際に設計する前に必要な予備手順についても説明します。

データ辞書

辞書に用語の定義が含まれているのと同様に、J.D. Edwards のデータ辞書はデータ項目の定義と属性を含むセントラル・リポジトリです。データ項目とは情報の単位です。データ項目の定義では、データ項目の使い方や項目のタイプ、長さを定義します。またデータ項目の属性は、データ項目の次のような属性を指定します。

- レポートやフォームの表示方法（小数点以下表示桁数やデフォルト値を表示するかどうかなど）
- アプリケーションでのデータ入力時の検証
- カラム（列）とロー（行）への記述の割当方法
- フィールド・センシティブ・ヘルプのテキストの表示方法
- テーブルへの保管方法

データ辞書は動的（ダイナミック）です。すなわち、データ項目に対して行った変更はいずれも、そのデータ項目を含むすべてのアプリケーションで直ちに有効になります。アプリケーションでは、実行時にデータ辞書にアクセスし、フィールド記述、カラム見出し、小数点以下表示桁数、編集規則など、データ項目の属性に対する変更をすぐに取り出します。

データ辞書を使用して、データ項目の属性を作成、表示、および更新します。類似の属性を持つデータ項目をコピーし、特定のニーズに合わせて修正できます。このようにすると、新規に作成するよりも早く簡単に作成できます。ただしこの場合には、コピーとコピー元とを区別するためにエイリアス名を修正してください。

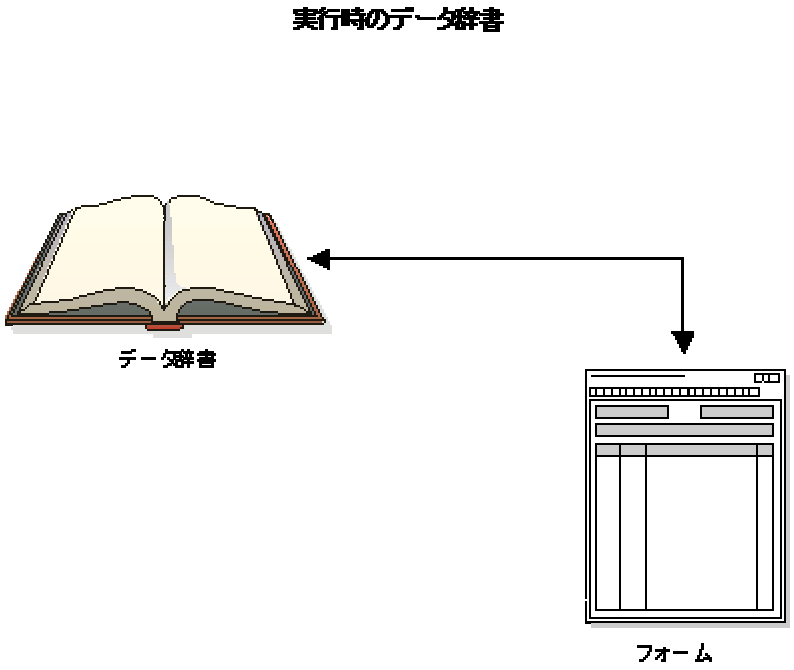
データ項目を変更すると、変更内容が実行時に J.D. Edwards ツール全体に直ちに反映されます。データ項目のタイプや属性を変更すると、それがデータの保管方法に反映され、レコード間の整合性が損なわれる可能性があります。

注意:

データ辞書は、データ項目を削除するときにそれがアプリケーションで使用されるかどうかを検証しません。アプリケーションが使用しているデータ項目を削除すると、アプリケーションで障害が発生します。

ランタイム時のデータ辞書の使用方法

次の図は、システムによる特定のデータ項目情報の処理方法を示しています。



実行時に、J.D. Edwards アプリケーション(〈Accounts Payable(買掛管理)〉や〈Sales Order Management(受注管理)〉など)は、データ辞書にアクセスし、割り当てられた属性を次のいずれかのフィールドに反映させます。

- Display Decimals(表示小数点以下桁数)
- File Decimals(ファイルへの小数点以下桁数)
- Alpha Description(名称記述)
- Data Type(データ・タイプ)
- Size(サイズ)
- Glossary(用語解説)
- Allow Blank Entry*(ブランク入力可否*)
- Upper Case Only *(大文字のみ*)
- All Triggers *(全トリガー*)
- Row and Column Headings *(ローおよびカラム見出し*)

アプリケーションでは、各フィールド情報をデータ辞書から取り出します。アスタリスク(*)が付いているフィールドは、〈Form Design(フォーム設計)〉と〈Report Design〉で一時変更することができます。この場合アプリケーションは、一時変更情報が存在すればそれらを使用して処理します。

項目を識別するための名称について

次の図では、データ項目を識別するための名称コンポーネントを示しています。

データ項目の命名規則コンポーネント

データ項目名		エイリアス		名称記述
Company	=	CO	=	Company
Cost Center	=	MCU	=	Business Unit

データ項目の作成後は名称記述のみを変更でき、データ項目名とエイリアス名は変更できません。データ項目が追加されると、データ辞書は既存のデータ項目名フィールドとエイリアス・フィールドをチェックし、固有であることを確認します。

データ辞書とデータ辞書項目の保管

中心となるデータ辞書情報は、次の 2 か所で保管されます。

- セントラル・オブジェクトのためのデータ辞書は、リレーショナル・データベースに保管されます。複製されるデータ辞書の変更についても、すべてこのデータベースで実行する必要があります。
- 複製のデータ辞書は、各クライアント・マシン上のスペック・テーブルの一部として、一連のデータ辞書テーブルが保管されます。

データ辞書項目は、エンタープライズ（ロジック）サーバーのリレーショナル・データベース・テーブルに常駐します。ワークステーションは、使用しているアプリケーションに必要なデータ項目のみをパブリッシャ・データ辞書（リレーショナル・データベース・テーブル）から取り出します。この複製は、OneWorld のインストール後にアプリケーションを初めて使用するときに行われます。このデータ辞書情報は、ワークステーションのパーマネント・キャッシュで同じローカル・パス・コード/スペック・ディレクトリの下にある次のグローバル・テーブルに保管されます。

- glbltbl.xdb（データの参照）
- glbltbl.ddb（データ項目）

データ項目を変更し、変更内容をワークステーションに直ちにレプリケートする場合は、〈Data Replication（データ・レプリケーション）〉アプリケーション(P98DREP)を使用する必要があります。データ・レプリケーションを設定してからデータ項目を変更した場合、サブスクライバとして設定されているマシンが次回 J.D. Edwards にサインオンすると、そのデータ項目のパーマネント・キャッシュが削除されます。その後、この変更されたデータ項目を使用するアプリケーションに次回アクセスすると、J.D. Edwards はパーマネント・キャッシュに情報が存在しないことを確認し、パブリッシャ・データ辞書（リレーショナル・データベース・テーブル）から情報を取り出します。

J.D. Edwards ERP ソフトウェアを WorldSoftware との共存環境で使用する場合は、2 つのデータ辞書を保守管理する必要があります。WorldSoftware では、$はビジネス・パートナー用として確保され、エイリアス名の先頭で使用されるため、1 つの辞書を共有することはできません。$はコンパイルされず、J.D. Edwards と共有できません。また World と J.D. Edwards では、ファイル・フォーマットもいくつか異なります。

参照

- データ辞書項目の管理については、『システム・アドミニストレーション』ガイドの「データ辞書の管理」

用語解説項目

用語解説項目とは、テーブルでは表現されないテキスト形式の項目です。用語解説項目は、一般に情報メッセージとして使用されます。

エラー・メッセージ

J.D. Edwards で使用されるエラー・メッセージも、データ項目として保管されています。通常のデータ項目に必要とされるフィールドはすべてを使用する必要はないので、エラー・メッセージを表示するためにもデータ辞書用語解説項目アプリケーションを使用してください。

デフォルト・トリガーとは

デフォルト・トリガーとは、アプリケーション実行時にデータ辞書レベルで処理される編集ルーチン、または表示ルーチンのことです。デフォルト・トリガーは再利用可能オブジェクトであるため、データ項目を使用する各アプリケーションに自動的に関連付けられます。ビジネス・ロジックは 1 度作成するだけで複数のアプリケーションに使用できるため、デフォルト・トリガーにより、時間を節約してコードを有効に利用できます。また、すべてのアプリケーション間でデータの正確なやりとりと一貫性が保持されます。

トリガーは、次の操作を行うために使用します。

- フィールドのデフォルト値を設定します。
- データ項目をその有効値をもつユーザー定義コード(UDC)テーブルにリンクします。
- ユーザーがカーソルをフィールドに合わせたときに、ビジュアル・アシスト検索プログラムを起動します。ビジュアル・アシストには、次のタイプがあります。
  - カレンダー
  - 電卓
  - 他のファイルの検索と選択
  - UDC
- フィールド用データの、編集およびフォーマット時に埋め込まれる規則とプロシージャを設定します。
- 開発者がデータに番号を割り当てるときに使用する自動採番方法を指定します。

データ辞書の使用方法

〈Object Management Workbench〉または〈Data Dictionary Application（データ辞書アプリケーション）〉プログラム(P92001)を使用して、新規のデータ辞書項目を作成したり、既存のデータ辞書項目を表示できます。データ辞書項目を作成した後、〈Data Dictionary Application〉プログラムを使用して、特殊用語や言語変換を定義します。

► データ辞書を使用するには

〈Data Dictionary（データ辞書）〉メニュー(GH951)から〈Work With Data Dictionary Items（データ辞書項目の処理）〉を選択します。

〈Work With Data Items（データ項目の処理）〉フォーム(W92001B)で、次のうち必要なフィールドに値を入力して[Find（検索）]をクリックします。

- 検索記述
- データ項目
- 記述
- エイリアス
- 用語解説グループ
- システムコード
- システム・コードレポート

フィールド記述

フィールド	記述
記述名	データ項目の記述。テキストを大文字および小文字で入力します。システムでは、このフィールドを使って同じようなデータ項目を検索します。名称記述を入力するために、次の規則に従ってください。 日付 すべての日付フィールドを日付で開始 金額 すべての金額フィールドを金額で開始 数量 すべての単位、数量、および容量フィールドを単位で開始 名前 すべての 30 バイトのフィールドを名前で開始 プロンプト Y/N フィールドをプロンプトで開始 住所 No. すべての住所番号（従業員、得意先、所有者）を住所番号で開始

フィールド	記述
データ 項目	情報の単位を参照したり、定義する識別子。32 英文字フィールドで、ブランクや・ % & , . + などの特殊文字は使用できません。 データ項目は変更できません。 ビジネス関数、データ構造体およびイベントルールなどで使用される、C 言語コードのデータ名（例、ADDRESSNUMBER）にもなります。 また、エイリアス名や名称記述でもデータ項目を識別することができます。
記述名	データ項目の記述。テキストを大文字および小文字で入力します。システムでは、このフィールドを使って同じようなデータ項目を検索します。名称記述を入力するために、次の規則に従ってください。 日付 すべての日付フィールドを日付で開始 金額 すべての金額フィールドを金額で開始 数量 すべての単位、数量、および容量フィールドを単位で開始 名前 すべての 30 バイトのフィールドを名前で開始 プロンプト Y/N フィールドをプロンプトで開始 住所 No. すべての住所番号（従業員、得意先、所有者）を住所番号で開始
エイリアス	情報単位を識別および定義するコード。8 文字のアルファベット順によるコード。ブランクおよび次の特殊文字（%、&、+など）は使用できません。システム・コード 55 から 59 を使用する新しい項目を作成します。エイリアスは変更できません。
用語解説グループ	OneWorld では、データ項目を識別するコード。ユーザー定義コード・テーブル(H98/DI)を検証します。用語解説グループ D または S の項目はデータベース・テーブルに含まれます。別の用語解説グループの項目（エラー・メッセージなど）はテーブルに追加されません。 World では、印刷するデータ辞書用語を選択するのに使用するデータ・タイプを指定するユーザー定義コード(98/GG)。 エラー・メッセージのデータ項目名は、自動的に割り当てられます。エラー・メッセージを独自に割り当てる場合は、5000 以上の 4 桁の番号で指定してください。 データベース・フィールド以外のデータ項目名（レポートなどで使用されますがファイルにはない用語解説グループ U）は#、$、または@で開始してください。ヘルプ・テキスト（用語解説グループ H）、データ辞書の<照会/改訂>プログラムのフィールドは、その次の項目名を特定するのに使用します。IBM メッセージ・ファイル（用語解説グループ J）に独自のメッセージを作成する場合は、データ項目名を 3 桁の文字で開始します。（例：CLT0001）
システム・コード	J.D. Edwards のシステムコードを示すユーザー定義コード(98/SY) 。

データ項目の定義

データベースのデータ項目を追加、変更、またはコピーするときには、スペックを含めてデータ項目を定義します。

データ項目の作成

データ辞書項目は J.D. Edwards オブジェクトのファンデーションです。データ辞書項目を作成することで、たとえばアプリケーションのフォーム上のフィールド、テーブルのカラム、ビジネス・ビューのフィールド、データ構造体のメンバー、UBE のフィールドとして使用できます。

▶ データ項目を作成するには

〈Object Management Workbench〉を開きます。

1. 〈Object Management Workbench（オブジェクト管理ワークベンチ）〉で、[Add（追加）]をクリックします。
2. 〈Add J.D. Edwards Object to the Project（J.D. Edwards オブジェクトのプロジェクトへの追加）〉で、フォームの[Control Table Objects（制御テーブル・オブジェクト）]セクションで[Data Item（データ項目）]オプションをクリックし、[OK]をクリックします。

   標準のデータ辞書項目を作成するか、用語解説データ項目を作成するかを確認するメッセージが表示されます。用語解説データ項目は、主にバッチ・エラー・メッセージ用として使用されます。

3. 標準のデータ辞書項目を作成するには、[No]をクリックします。
4. 〈Data Items Specification（データ項目スペック）〉で、適切な手順に従ってデータ項目名を指定します。

データ項目名の登録

基本データ項目を作成したら、それをアプリケーションに追加する前に、固有の名称と属性を新しいデータ辞書項目へ割り当てる必要があります。

▶ データ項目名を登録するには

1. 〈Data Item Specifications〉の次のフィールドに値を入力します。
   - データ項目
   - エイリアス
   - 用語解説グループ
2. 一般情報を定義する手順を実行します。

フィールド記述

フィールド	記述
データ 項目	情報の単位を参照したり、定義する識別子。32 英文字フィールドで、ブランクや・ % & , . + などの特殊文字は使用できません。 データ項目は変更できません。 ビジネス関数、データ構造体およびイベントルールなどで使用される、C 言語コードのデータ名（例、ADDRESSNUMBER）にもなります。 また、エイリアス名や名称記述でもデータ項目を識別することができます。
エイリアス	情報単位を識別および定義するコード。8 文字のアルファベット順によるコード。ブランクおよび次の特殊文字（%、&、+など）は使用できません。システム・コード 55 から 59 を使用する新しい項目を作成します。エイリアスは変更できません。
用語解説グループ	OneWorld では、データ項目を識別するコード。ユーザー定義コード・テーブル(H98/DI)を検証します。用語解説グループ D または S の項目はデータベース・テーブルに含まれます。別の用語解説グループの項目（エラー・メッセージなど）はテーブルに追加されません。 World では、印刷するデータ辞書用語を選択するのに使用するデータ・タイプを指定するユーザー定義コード(98/GG)。 エラー・メッセージのデータ項目名は、自動的に割り当てられます。エラー・メッセージを独自に割り当てる場合は、5000 以上の 4 桁の番号で指定してください。 データベース・フィールド以外のデータ項目名（レポートなどで使用されますがファイルにはない用語解説グループ U）は#、$、または@で開始してください。ヘルプ・テキスト（用語解説グループ H）、データ辞書の<照会/改訂>プログラムのフィールドは、その次の項目名を特定するのに使用します。IBM メッセージ・ファイル（用語解説グループ J）に独自のメッセージを作成する場合は、データ項目名を 3 桁の文字で開始します。（例：CLT0001）

データ辞書命名規則

データベースの整合性を確保し、データ項目が他のデータ項目によって上書きされないように、データ辞書命名規則に従ってください。

データ項目名

データ項目名は、データ項目を識別し、定義する 32 文字の英字フィールドです。このフィールドでは、翻訳を考慮して英文テキストの 30%程度の長さのスペースを設けます。

データ項目名は、ビジネス関数、データ構造体およびイベント・ルールで使用される C 言語コードのデータ名（たとえば、AddressNumber など）を形成します。データ項目は、エイリアスや名称記述によっても識別できます。

J.D. Edwards データ辞書項目を作成するときには、データ項目名の最初の文字に Y と Z を使用しないでください。Y と Z は、開発ビジネス・パートナーのパートナー用として確保されています（Y または Z から始まる J.D. Edwards データ項目は、J.D. Edwards ERP ソフトウェアの B73.2 リリース以前に作成されているため存在することがありますが J.D. Edwards 項目として存在し続けます）。

J.D. Edwards ソフトウェアのデータ項目名には、ブランクと次の文字は使用できません。

%

&

+

注:

データ項目を追加した後は、その名称を変更できません。

外部データ辞書項目のデータ項目名

外部データ辞書項目を作成するときには、データ項目名の最初の文字に Y か Z を使用して、J.D. Edwards データ辞書項目と区別してください。

データ項目名には最大 32 文字の英数字を使用でき、次のフォーマットを使用します。

Ysssddddddddddddddddddddddddddddd

ここではこれらの変数について説明します。

Y または Z – 外部データ辞書項目であることを示す、J.D. Edwards 指定の外部システム・コードの第 1 桁。

sss – システム・コード番号。この番号は、新規モジュールの企業レベルの開発では 55xx～59xx、J.D. Edwards カスタム開発では 60xx～69xx、またはこれら以外の開発パートナー・システム管理者が割り当てる番号のいずれかになります。

ddddddddddddddddddddddddddddd は、データ項目の固有の名称です。

データ項目エイリアス

データ項目エイリアスは 8 文字の英字コードです。データ辞書項目が J.D. Edwards アプリケーション専用である場合には、そのエイリアスは 5～8 文字になります。RPG プログラムでのテーブルで使用されるデータ項目を追加するときは、そのエイリアスを 4 文字以内とします。データ項目は、その名称や名称記述でも識別できます。

データ項目エイリアスは、検索、データベース・ルーチン(ビジネス関数で使われるアプリケーション・プログラム・インターフェイス)、およびテーブル作成時に〈Table Design(テーブル設計)〉で使用されます。エイリアスには、テーブルごとのプレフィックスが付けられ、それぞれのテーブルに固有のものとなります。たとえば、ABMCU は、MCU が住所録マスター(F0101)に含まれていることを示します。

エイリアスを割り当てるときには、名前の先頭に TIP または TERM という文字列を使用しないでください。TIP から始まるエイリアスは、J.D. Edwards のヒント情報用として確保され、TERM から始まるエイリアスは、J.D. Edwards ガイドに含まれる用語解説用として確保されています。

J.D. Edwards ソフトウェアのデータ項目名には、ブランクと次の文字は使用できません。

%

&

+

注:

データ項目を追加した後は、その名称を変更できません。

外部データ辞書項目のエイリアス

外部データ辞書項目は、J.D. Edwards の外部の開発者により、J.D. Edwards ソフトウェアで使用するために作成されるものです。外部データ項目の場合、データ辞書エイリアスは英数字で 8 文字以内です。外部データ項目には、次のフォーマットを使用します。

Ysssdddd

ここではこれらの変数について説明します。

Y または Z – 外部データ辞書項目であることを示す、J.D. Edwards 指定の外部システム・コードの第 1 桁。

sss – システム・コード番号。この番号は、新規モジュールの企業レベルの開発では 55xx～59xx、J.D. Edwards カスタム開発では 60xx～69xx、またはこれら以外の開発パートナー・システム管理者が割り当てる番号のいずれかになります。

dddd は、データ項目の固有の名称です。

一般情報の定義

データ項目の名前を設定した後は、〈Data Item Specifications〉で追加の一般情報を指定する必要があります。

► 一般情報を定義するには

1. 〈Data Item Specifications〉で、[Item Specifications]タブをクリックします。
2. 次の必須フィールドに値を入力します。
   - 記述
   - データ・タイプ
   - システム・コード
   - コントロール・タイプ
   - 記述

この記述は、別の言語に合わせて更新しない限り、基本言語専用です。

- カラム・タイトル

  この記述は、別の言語に合わせて更新しない限り、基本言語専用です。

- サイズ
- ファイル表示小数点
- 表示小数点以下桁数

3. 必要に応じて、次のフィールドにも値を入力します。
   - クラス
   - 発生項目数

     [Item Occurrences]は、配列で使用します。これにより、項目を別の項目の子として作成することができます。データ辞書は、親子の間で属性が一致するかどうかを確認します。親項目を変更すると、変更内容が子項目に反映されます。データ項目名では、親データ項目名と番号を使用し、たとえば、親項目が ABC の場合、子項目は ABC1、ABC2 となります。

4. 以下のいずれかのオプションをクリックします。
   - 大文字のみ
   - ロー・セキュリティ
   - ブランク使用可能
   - 自動インクルード
   - 合計しない

5. デフォルト・トリガーを関連付ける手順を実行します。

フィールド記述

記述	用語解説
記述	データ項目の記述。テキストを大文字および小文字で入力します。システムでは、このフィールドを使って同じようなデータ項目を検索します。名称記述を入力するために、次の規則に従ってください。 日付 すべての日付フィールドを日付で開始 金額 すべての金額フィールドを金額で開始 数量 すべての単位、数量、および容量フィールドを単位で開始 名前 すべての 30 バイトのフィールドを名前で開始 プロンプト Y/N フィールドをプロンプトで開始 住所 No. すべての住所番号（従業員、得意先、所有者）を住所番号で開始
システム・コード	「次の番号」検索用にシステム番号を指定します。ユーザー定義コード（システム・コード 98、コード・タイプ SY）を参照してください。

データ・タイプ	データの種類（数値、英字、日付など）。既存のアプリケーションで使用されている場合は、データ項目タイプを変更しないことをお勧めします。変更すると、テーブル、およびこのデータ項目を使用しているすべてのビジネス関数を検討する必要があります。データ・タイプは次のとおりです。 文字 単一文字。常にサイズは 1 です。 日付 日付。 整数 整数。 文字 (BLOB) EBCDIC（IBM メインフレームで使用される 8 ビット文字コード)から ASCII（7 ビット文字コード）に変換されます。 バイナリ (BLOB) 翻訳不可のデータ。マシン・コードに表示されます。Win.help の下に実行可能ファイルとして存在します。 バイナリ 2 つの選択を表すフラグ。 1 と 0 でオンまたはオフ、真または偽を表します。 文字列 常に同じ大きさまたは長さのデータ 変数 可変サイズの項目 識別子 コントロールのプログラム・ロジック内のユーザーIDID を用いて、C プログラムが書かれ、ポインタを戻すサード・パーティ・ソフトウェアを参照します。JDE API により、ID を参照するポインタが保存されます。C プログラムに渡されたパラメータがその ID です。 数字 長い整数
サイズ	データ項目のフィールド・サイズ。 注：すべての数値を 5 バイトの小数点なしで入力してください。データ項目タイプは P(パック済み)です。
ファイル表示小数点	保管されるデータ項目の小数点の右の位置番号。
表示小数点以下桁数	システムが表示する通貨、金額、数量フィールドにおける小数点以下桁数を指定します。たとえば、米ドルには 2 つ、カメルーンのフランには 3 つ、また、日本円にはありません。

コントロール・タイプ

データ項目に関連するグラフィカル・ユーザー・コントロールのタイプを定義します。たとえば、データ項目はプッシュ・ボタン、チェックボックス、ユーザー定義コードなどの形式で表示されます。

コントロール・タイプは通常、フォーム・デザインが特定のデータ項目に対するフォームに適切なコントロールを自動追加する時に使用します。たとえばデータ項目を通常、チェックボックスとして使用する場合、データ辞書のコントロールタイプはチェックボックスとなります。クィックフォームを使用する場合、デフォルトはフォームの汎用編集ではなくチェックボックスのコントロールに設定されます。

この設定はフォームデザインで一時変更できます。ただし、最も頻繁に使用する形式を予測して設定する必要があります。

記述

OneWorld の場合、ロー記述はフォームおよびレポートのフィールド記述に使用されます。

World では、テキストとレポートのタイトルが作成されます。照会機能のカラム記述と同じような方法で使用されます。入力できる文字数は 35 文字までです。必要に応じて略語を使用してください。主な略語は次のとおりです。

U/M
計量単位

YTD
年累計

MTD
月累計

PYE
前年度末

QTY
数量

G/L
総勘定元帳

A/P
買掛金

DEPR
減価償却

カラム・タイトル

レポートまたはフォームのカラム見出しに使う記述の 1 行目。この記述は、データ項目サイズを越えないようにしてください。カラム見出しは 1 行のみの場合は、このカラムに入力してください。1 行目だけでは不十分な場合は、カラム・タイトルの 2 行目を使用してください。

クラス

データ項目クラス。クラスは、データ項目の基本的な属性および特性を定義します。参照用のみです。

発生項目数

データ辞書でデータ項目を設定する際に、配列要素数を指定することができます。
こうすることで、1 配列要素につき 1 追加データ項目を作成できます。

配列データ項目名の長さは、配列要素数によって制限されます。

3 バイト	1 から 9 要素
2 バイト	10 から 99 要素
1 バイト	100 から 999 要素

大文字のみ

このフィールドの値が"Y"の場合、入力コントロールには大文字のみ入力できます。

ブランク使用可能

このオプションをオンにすると、次の条件でデータベースにブランクの値が書き込まれます。

1 フィールドがユーザー定義コードテーブルに対して編集されると、テーブルにブランクの値が有効であるかどうかにかかわらず、ブランクの値が使用できます。
2 フィールドの入力が必須の場合、ブランクの値は有効となります。

ロー・セキュリティ

ロー・セキュリティの設定にフィールドが使用できることを示すフラグのフィールド(将来使用)。

現在は、もし先頭に 2 バイトのテーブル・プレフィックスがついても 6 バイトを超えないように 4 バイトに制限されています。

データ辞書の中では、すべてのデータ項目がこの 4 バイトのデータ名によって参照されます。データベース・テーブルで使用されているように、2 文字のプレフィックスが各テーブル・スペック(DDS)の固有なデータ名を作成するために追加されます。エラー・メッセージを追加する場合は、フィールドはブランクにしてください。自動採番を使ってエラー・メッセージが割り当てられます。名前が追加に表示されました。エラー・メッセージの番号には 5000 以上の数字を割り当ててください。特殊文字は#、@、$を除いてデータ項目の一部としては使用できません。

$xxx、@xxx、および xxx を定義した箇所を使って、保護されたデータ名を作成できます。

システムでは、情報単位を識別および定義するコード。8 文字のアルファベット順によるコードです。ブランクおよび特殊文字(% & , . +)は使用できません。

エイリアスは変更できません。

自動インクルード

このチェックボックスでは、この項目を含むテーブルに対するすべてのデータベース フェッチにこのカラムを自動的に含むかどうかを指定します。このオプションは、特定のデータベース・トリガー処理に重要な項目またはセキュリティ違反に対してのみ設定してください。

合計しない	特定の開発プラットフォームだけに適用されるデータ項目を識別します。 有効な値は次のとおりです。 1 – システムのみ 2 – AS/400 のみ ブランク – 両方のプラットフォームに有効なデータ項目 --- フォーム固有 --- データ項目が数値タイプの場合、このフラグがオンになります。データ項目に"NOT TO TOTAL"のマークが付いていると、このフラグがオンになります。UBE のレポートでこの項目を使用すると、"SUPPRESS AT TOTAL"という項目プロパティがマークされます。

デフォルト・トリガーの関連付け

アプリケーションの実行時に、データ項目に関連付けられている表示ルーチンおよび編集ルーチンを実行するには、トリガーを使用します。

▶ デフォルト・トリガーを関連付けるには

1. 〈Data Item Specifications〉で、次の適切なタブをクリックして、適用可能なトリガーをデータ項目に関連付けます。
   - Default Value(デフォルト値)
   - Visual Assist(ビジュアル・アシスト)
   - Edit Rule(編集ルール)
   - Display Rule(表示ルール)
   - 自動採番
2. 各タブで、関連付けるデフォルト値のタイプを表す適切なオプションをクリックします。

   オプションをクリックすると、適切なデフォルト値を指定できるようにフィールドが追加表示されます。

上記のトリガーは、〈Form Design〉、〈Edit Control Properties(コントロール・プロパティの編集)〉でいずれも一時変更できますが、最も使用される可能性の高い指定を行う必要があります。

フィールド記述

フィールド	記述
ディフォルト値	フィールドがブランクの場合に、デフォルト値をデータ項目に割り当てます。 数値として入力する文字はデータ項目と同じ長さで、ゼロで開始します。 ユーザー定義コードのトリガーの場合、添付されたユーザー定義コードに対して入力された値が有効かどうかを確認してください。

フィールド	記述
ビジュアル・アシスト	ビジュアル・アシストには 3 つのタイプがあります。 電卓 – フィールドに電卓が割り当てられています。金額を入力する必要のあるすべての数値データ項目で使用します。 カレンダー – フィールドにカレンダーが割り当てられています。日付を入力する必要のあるデータ項目で使用します。 検索フォーム – フィールドに検索フォームを割り当てます。フィールドの有功値をテーブルから検索するすべてのフィールドで使用します。
編集ルール	編集ルールと手順をデータ項目に割り当てるために使用します。 編集ルールは、値を特定の範囲に指定するなど、データをフィールドに入力するときの編集方法を指定します。 ビジネス関数は、編集ルールを使用できない特殊なロジックを添付する場合に使用します。たとえば、得意先番号を入力するたびに住所録テーブルを自動的に検証する場合は、住所録を編集するビジネス関数を添付します。
表示ルール	表示方法を決定するデータ項目にルールと手順を割り当てます。 表示ルールは、データが表示されるときに適用されるフォーマット方法を説明するキーワードです。 これら 5 つの表示ルールのいずれを使用しても、表示されない特殊な処理が必要な場合にビジネス関数を使用します。たとえば、勘定科目コードのフォーマットを整える場合は、勘定科目コードの表示と呼ばれるビジネス関数を添付します。
自動採番	自動採番ロジックをこのデータ項目に割り当てるのに使います。 このルールでは、次の番号および自動採番テーブル(F0002)のデータが使用する配列インデックス番号の検索を行うのにどのシステム・コードを使うかを決定します。

デフォルト値トリガーの関連付け

デフォルト値がほとんどの状況で使用されるフィールドには、デフォルト値トリガーを使用します。処理オプション、ビジネス関数、フォーム・インターコネクト、データ構造体によって、このフィールドにデフォルト値以外の値が渡された場合、またはユーザーが異なる値を入力した場合、このデフォルト値は使用されません。デフォルト値トリガーは、次のイベントが発生したときに、指定のデフォルト値をフィールドに自動的に挿入します。

- Control is exited(コントロールが終了した)
- Dialog is initialized (ダイアログの初期設定時)

► デフォルト値トリガーを関連付けるには

1. 〈Data Item Specifications〉で、[Default Value（デフォルト値）]ボタンをクリックします。
2. 次のオプションをクリックします。
   - デフォルト値
3. デフォルト値を入力します。

フィールド記述

記述	用語解説
デフォルト値	フィールドにデフォルト値があるかどうか示します。次のいずれかのオプションを選んでください。 o デフォルト値なし – デフォルト値が割り当てられていません。 o デフォルト値 – フィールドにデフォルト値が割り当てられています。フィールドに割り当てられるデフォルト値を指定してください。

ビジュアル・アシスト・トリガーの関連付け

次の 3 タイプのビジュアル・アシスト・トリガーを使用することができます。

- 検索フォーム
- 電卓
- カレンダー

検索フォーム・トリガーを使用すると、フィールドを検索フォームにリンクして有効値を検索することができます。検索フォーム・トリガーを関連付けるには、検索フォームが必要です。

► ビジュアル・アシスト・トリガーを関連付けるには

1. 〈Data Item Specifications〉で、[Visual Assist(ビジュアル・アシスト)]タブをクリックします。
2. [Visual Assist]で次のオプションのうち 1 つをクリックします。
   - 電卓
   - カレンダー
   - 検索フォーム
   - JDE UTime
3. [Search Form(フォームの検索)]をクリックした場合は、[Browse(ブラウズ)]ボタンをクリックし、有効な値を表示するフォームを選択します。

   フォームの検索・アシスト・トリガーを関連付けた場合は、実行時に[Search Form]ボタンをクリックすると、指定したフォームが表示されます。検索条件を入力すると、有効な値が戻されます。これらの〈Search & Select(検索/選択)〉フォームは、ユーザー定義コード・テーブルではなくファイルに基づいています。

フィールド記述

フィールド	記述
ビジュアル・アシスト	フィールドまたはタイプでビジュアルアシストがあるかどうかを指示します。次のオプションから選んでください。 ビジュアルアシストなし – ビジュアルアシストを割り当てません。 電卓 – フィールドに電卓を割り当てます。金額を入力する必要のあるすべての数値データ項目で使用します。 カレンダー – フィールドにカレンダーを割り当てます。日付を入力する必要のあるデータ項目で使用します。 検索フォーム – フィールドに検索フォームを割り当てます。フィールドに有効な値がファイルで見つかった場合、すべてのフィールドで使用します。

編集ルール・トリガーの関連付け

編集ルール・トリガーを使用し、ビジネス関数または規則に基づいてフィールドの値を検証します。たとえば、次のような操作を実行する規則を定義できます。

- フィールドを検証して特定の値と比較
- フィールド値が指定した値の範囲にあるかどうかを確認
- フィールドを特定の UDC 検索および選択用フォームに接続
- Y および N 値をチェック

► ビジネス関数の編集ルール・トリガーを関連付けるには

1. 〈Data Item Specifications〉で、[Edit Rule]タブをクリックします。
2. [Edit Rule]で次のオプションをクリックします。
   - ビジネス関数
   - ルール
3. [Business Function(ビジネス関数)]オプションをクリックした場合は、[Browse]ボタンをクリックして、使用可能なビジネス関数を選択します。
4. 〈Business Function Search(ビジネス関数の検索)〉で、ビジネス関数を検索します。
5. 関数の目的を確認するには、その関数を選んで[Row(ロー)]メニューから[Attachments(添付)]を選択します。

   ビジネス関数の使用上の注意を含む〈Media Objects(メディア・オブジェクト)〉フォームが表示されます。情報を確認したらフォームを閉じます。

   注:
   ビジネス関数が存在しない場合は、作成する必要があります。

6. 〈Business Function Search〉で、[Select(選択)]をクリックしてビジネス関数を選択します。
7. [Rule(ルール)]オプションをクリックした場合は、[Search(検索)]ボタンをクリックして、使用可能なルールを選択します。

これらのルールには、次のようなユーザ定義コードを含めることができます。

EQ	Equal（等しい）
GE	以上
GT	より大きい
HNDL	テーブル・ハンドル
LE	以下
LT	Less Than（より小さい）
NE	Not Equal（等しくない）
NRANGE	範囲外
RANGE	範囲
UDC	ユーザー定義コード
VALUE	リスト
ZLNGTH	割り当て済の長さ（VARLEN フィールド）

表示ルール・トリガーの関連付け

表示ルール・トリガーは、データのフォーマットに使用します。表示ルール・トリガーは、ビジネス関数またはユーザー定義コードに基づいて関連付けしてください。次の 2 タイプの表示ルール・トリガーを使用することができます。

- ビジネス関数
- ルール

▶ 表示ルール・トリガーを関連付けるには

1. 〈Data Item Specifications〉で、[Display Rule（表示ルール）]タブをクリックします。
2. [Display Rule]で次のオプションをクリックします。
   - ルール
3. 表示されるフィールドの[Search]ボタンをクリックして、使用可能なユーザー定義コードから選択します。

次の値のうち 1 つを選択します。

***RAB** 値が右詰めされ、先行ブランクが埋め込まれます。これは、ビジネス・ユニットを定義するデータ項目に適用されます。

***RABN** 値が右詰めされ、先行ブランクが埋め込まれます。これは、ビジネス・ユニットを定義しないデータ項目に適用されます。

***RAZ** 値が右詰めされ、先行ゼロが埋め込まれます。たとえば、[Company(会社)]は 00001 として表示されます。

CODE 指定された編集コードを使用して数値フィールドがフォーマットされます。有効なコードのリストは、ユーザー定義コード(98/EC)を参照してください。コードは、パラメータ・フィールドに入力する必要があります。

MASK データが表示されるときに、指定した文字をデータに埋め込まれます。たとえば、社会保障番号(SSN_)を埋込みダッシュと共に表示するには、マスク・パラメータを次のように指定します。

bbb-bb-bbbb(b は、社会保障番号を構成する数字)

マスクは、文字または文字列型のデータ項目でのみ使用できます。

フィールド記述

フィールド	記述
フォーマット設定	フィールドに表示ルールがあるかどうか示します。次のオプションから選んでください。 O なし – 表示ルールはありません。 O ビジネス関数 – フィールドにビジネス関数を割り当てます。このオプションを選ぶときに使用可能なビジネス関数プロシージャをフィールドに指定します。ビジネス関数を選択するには、ビジュアル・アシストを使用すると便利です。 O ルール – フィールドに表示ルールを割り当てます。このアクションを選ぶときに使用可能となるフィールドの表示ルールを指定してください。ルールを表示して選択するには、ビジュアル・アシストを使用すると便利です。

ユーザー定義コード・トリガーの関連付け

UDC テーブルを使用して特定のフィールドのデータを検証するには、ユーザー定義コード・トリガーを関連付けます。フィールドにこの種のトリガーを関連付けると、指定した UDC テーブルに存在する値のみがそのフィールドに対して有効になります。

▶ ユーザー定義コード・トリガーを関連付けるには

1. 〈Data Item Specifications〉で、[Edit Rule]タブをクリックします。
2. [Edit Rule]で、次のオプションをクリックして、オンにします。
   - ルール
3. [Visual Assist]をクリックして、グリッドで UDC を選択します。
4. Tab キーを押して[Rule]フィールドから移動し、次の各フィールドに値を入力します。
   - システム・コード
   - レコード・タイプ
5. 〈Data Item Specifications〉で、[OK]をクリックします。

フィールド記述

フィールド	記述
システム・コード	J.D. Edwards のシステムコードを示すユーザー定義コード(98/SY) 。
ユーザー定義コード	ユーザー定義コードを含むテーブルを示すコード。このテーブルは UDC(ユーザー定義コード)タイプともいいます。

自動採番トリガーの関連付け

自動採番機能は、新規の GL 勘定科目番号、伝票番号、住所番号などの項目の自動採番処理を制御します。この機能では、使用する採番システム・コードを指定し、数値の自動増分により、位取りエラーや入力エラーを防ぐことができます。

自動採番機能は、ユーザーが番号を入力しなかった場合に、デフォルト値を数値データ項目に自動的に入力するときに使用します。「次の番号」は、システム・コードとインデックスの組合せによる配列の中から割り当てられます。したがって、次の番号はそれぞれの自動採番バケットから割り当てられることになります。

自動採番

自動採番テーブルは F0002 で、次のロジックが設定されています。

- システムごとに 1 レコード、および 10 の要素からなる配列

  自動採番テーブル(F0002)のキーはシステム・コードです。このテーブルには、個々の自動採番要素に対応する 10 のカラムが存在します。これらの各要素は、該当するシステム・コードのアプリケーションの特定のハード・コードに対して使用されます。

  たとえば、〈Next Numbers(自動採番)〉プログラムでシステム・コード 09 を指定すると、6 つのローにデータが入力され、4 つのローがブランクになります。コーディングされ、入力されたこれらのローはそれぞれがハード・コードとして使用されます。最初のローが新規勘定科目 ID を定義します。新規勘定科目を作成する J.D. Edwards のアプリケーションで、自動採番テーブルのシステム 09、ロー1 から勘定科目番号が取り出されます。ロー2 には、仕訳が入力されています。仕訳文書を作成する MBF(マスター・ビジネス関数)では、自動採番テーブルのシステム 09、ロー2 から文書番号が取り出されます。

〈Next Numbers〉プログラムでシステム 04 を指定した場合は、システム 04 で使用するハード・コードが存在する、まったく別のローのセットが使用されます。

- モジュラス 11 のチェックは任意

  チェック・ディジット・オプションを使用すると、システムによって割り当てられる個々の自動採番の末尾に番号を追加するかどうかを指定できます。

  たとえば、チェック・ディジットの使用時に自動採番が 2 の場合は、7 などのチェック・ディジットが追加され、最後の 2 つの番号が 27 になります。チェック・ディジットでは、番号を無作為に増分することで、転置された番号の割り当てを防ぐことができます。次の項で説明するアルゴリズムでは、チェック・ディジット機能が起動している間、72 という自動採番は割り当てられません。

IBM モジュラス 11 セルフチェック・アルゴリズム

基礎番号の各位置には、それぞれ重み係数が割り当てられています。モジュラス 11 は、(チェック・ディジットを含まない)右端の桁から位置を数えます。

モジュール 11 の重み係数は、1～31 までの各位置に対して、2、3、4、5、6、7、2、3、4、5、6、7、…2、3、4、5、6、7、2 となっています。

自動採番データは、一度設定したら変更しないでください。データを変更すると、次のような問題が生じます。

- システムのパフォーマンスに影響します。
- 自動採番は重複する番号を割り当てないため、最大番号に達すると最小番号に戻ります。
- プログラムを修正しなければ、各システム・レコードでの位置(Index)を変更したり、新しい項目を追加することはできません。

自動採番機能はデータ辞書と連動します。データ辞書に存在するデータ項目は、自動採番システムをポイントします。〈Next Numbers〉プログラム(P0002)は、〈General Systems(汎用システム)〉メニュー(G00)から使用できます。

▶ 自動採番トリガーを関連付けるには

1. 〈Data Item Specifications〉で、[Next Number(自動採番)]タブをクリックします。
2. [Next Number]で次のオプションをクリックします。
   - 自動採番
3. 次のフィールドに値を入力します。
   - システム・コード
   - インデックス

フィールド記述

記述	用語解説
自動採番	フィールドに自動採番の一時変更があるかどうか指定します。次のいずれかのオプションを選んでください。 O なし – 自動採番が割り当てられていません。 O 一時変更 – データ項目に自動採番のロジックを割り当てます。データが自動採番テーブルで使用する自動採番システム・コードと自動採番インデックスを指定してください。
システム・コード	「次の番号」検索用にシステム番号を指定します。ユーザー定義コード（システム・コード 98、コード・タイプ SY）を参照してください。
インデックス	自動採番プログラムによって検索される配列要素番号。たとえば、買掛伝票の 「次の番号」はシステム 04 の配列要素 02 です。

スマート・フィールド・トリガーの関連付け

スマート・フィールドとは、ビジネス関数の添付されたデータ辞書項目のことです。関連付けられたビジネス関数にはネームド・マッピングが含まれています。これは、データ項目の選択プロセスを特定の機能によって簡素化します。エンドユーザーは、どのビジネス関数を使用し、どのパラメータを受け渡すかを知る必要はなく、適切な情報を持つデータ項目を選択するだけで済みます。スマート・フィールドは、〈Report Design〉のすべてのセクション・タイプで使用できます。たとえば、スマート・フィールドを使用すると、表セクションのカラム見出しやテーブル・セクションのオブジェクト値を取得することができます。スマート・フィールドは、常に用語解説グループ K になります。

▶ スマート・フィールド・トリガーを関連付けるには

1. 〈Data Item Specifications〉で、[Item Specifications]タブをクリックします。
2. 次の必須フィールドに値を入力します。
   - 記述
   - システム・コード
   - データ・タイプ
   - コントロール・タイプ
   - 記述
   - カラム・タイトル
   - サイズ
   - ファイル表示小数点
   - 表示小数点以下桁数

3. 必要に応じて、次のフィールドにも値を入力します。
   - クラス
   - 発生項目数
4. 次のオプションのうち 1 つをクリックします。
   - 大文字のみ
   - ロー・セキュリティ
   - ブランク使用可能
   - 自動インクルード
   - 合計しない
5. [Form(フォーム)]メニューから[Smart Field(スマート・フィールド)]を選択します。
6. 〈Smart Field〉で、次のフィールドに値を入力します。
   - ビジネス関数

     スマート・フィールド用として作成されているビジネス関数を入力します。
   - Event Name(イベント名)

     スマート・フィールドのトリガーにするイベントを指定します。
   - Named mapping(ネームド・マッピング)

     ビジネス関数データ構造体に関連するネームド・マッピングを入力します。

     〈Smart Field〉では、[Browse]ボタンをクリックして、ビジネス関数を選択できます。また、[Named Mapping(ネームド・マッピング)]のビジュアル・アシストをクリックして値を選択することもできます。

用語解説の更新

各データ項目の説明は、用語解説で記述できます。このテキストは、アプリケーションの実行時にエンドユーザーに表示されます。したがって、アプリケーションでのフィールドの用途、使用法を説明するようにします。この用語解説は、アプリケーションの実行時にフィールド・レベル・ヘルプで参照できます。また、次の操作も実行できます。

- 新規データ項目の用語解説を追加する。
- フォーム固有またはシステム固有の用語解説を作成する。
- 各種言語の用語解説を作成する。
- J.D. Edwards の用語解説をニーズに合わせてカスタマイズする。

▶ 用語解説を更新するには

1. 〈Data Item Specifications〉で、[Item Glossary(項目用語解説)]タブをクリックします。
2. 〈Media Object(メディア・オブジェクト)〉ウィンドウに用語解説情報を入力します。

フィールド記述

フォームに対する用語解説を入力すると、その記述はそのフォームが表示されるときにのみヘルプ情報として表示されます。

各言語による用語解説の追加

どのデータ辞書項目についても、各種言語の用語解説テキストを追加できます。たとえば、基準となる英語の項目の用語解説を作成し、さらにフランス語やスペイン語、ドイツ語などの用語解説を追加することができます。言語別の用語解説は、データ辞書項目を作成した後に追加してください。

▶ 各言語による項目の用語解説を追加するには

1. 〈Work With Data Items〉で、変更する検索を選択します。
2. [Row(ロー)]メニューから[Glossary Overrides(用語解説一時変更)]を選択します。
3. 〈Work With Data Item Glossary Overrides(データ項目用語解説一時変更の処理)〉で、[Add]をクリックします。
4. [Data Item Glossary Header(データ項目用語解説見出し)]で、次の 2 つのフィールドに入力して、[OK]をクリックします。
   - Language(言語)
   - Form

     用語解説を特定のフォームに限定する場合は、フォーム名を入力します。

     フォーム名を入力しないと、用語解説は、該当する項目を使用するすべてのフォームに適用されます。
5. [OK]をクリックします。
6. 〈Work with Data Item Glossary Overrides〉で、追加したばかりのローを選択します。
7. [Row]メニューから[Glossary]を選択します。
8. 〈Data Item Glossary(データ項目用語解説)〉で、表示する用語解説を入力します。

項目の用語解説は、既存の言語一時変更によっても変更できます。

アプリケーション一時変更と代替言語別用語の定義

データ辞書項目を作成するときは、ロー記述、カラム記述、および用語解説を割り当てます。これらの記述では、柔軟に専門用語を使用できないことがあるので、各項目に特殊用語や言語の代替記述を割り当てることができます。代替記述を使用すると、ユーザーが使用しているオブジェクトのシステム(製品)コードに従って、同じデータ辞書項目をユーザーごとに異なるロー記述、カラム記述、および用語解説を使って表示できます。

たとえば、ビジネス・ユニット・フィールドの MCU はシステム全体で広く使用されます。そのロー記述は「Business Unit（ビジネス・ユニット）」で、これは財務アプリケーションで使用される用語ですが、このデータ項目は、流通アプリケーションでは「Branch/Plant（事業所）」、倉庫管理アプリケーションでは、「Warehouse（倉庫）」と表示されます。

定義した代替用語に加え、ユーザーは、それぞれ独自の言語一時変更をアプリケーション・レベルで実現できます。 一時変更では次の順序でチェックして解決します。

1. ユーザーがアプリケーション（〈Form Director Aid（フォーム・ディレクタ・ツール）〉や〈Report Design〉など）で言語一時変更を適用すると、言語一時変更で指定されている用語が存在する場合、その用語が使用されます。
2. ユーザーがアプリケーションで言語一時変更を指定しなかった場合、システム・コードがメニュー項目に添付されているかどうかが実行時に確認されます。メニュー項目にシステム・コードが添付されている場合、システム・コードで指定された代替用語が存在する場合、その用語が表示されます。
3. メニュー項目の代替用語が指定されていない場合は、システム・コードがアプリケーションに添付されているかどうかが実行時に確認されます。アプリケーションにシステム・コードが添付されている場合、システム・コードで指定されている代替用語が存在する場合、その用語が表示されます。
4. アプリケーションの代替用語が指定されていない場合は、データ辞書テキストが表示されます。

以上 4 つのいずれのケースでも、最初にユーザーの使用言語で代替用語の有無をチェックしてから言語以外がチェックされます。言語と言語一時変更は、常に言語以外の一時変更に優先するためです。たとえば、英語を基本言語にしている環境で、すべてのフォームにフランス語訳があり、ユーザーがフランス語一時変更を選択すると、それらを表示できる場合を考えてみます。ここでは、フォームには英語の代替用語が存在するデータ辞書項目が含まれていますが、フランス語版のデータ辞書項目には代替用語が存在しないとします。この場合、フォームが英語で表示されたときは主要用語または代替用語が適宜表示されますが、フランス語で表示されたときには代替用語が呼び出されても主要用語しか表示されません。これは、言語一時変更が代替用語の表示に優先するためです。

アプリケーション一時変更と代替言語別用語は、データ辞書項目を作成した後に追加する必要があります。

参照

- データ項目の代替言語別用語解説項目を作成する方法については、『開発ツール』ガイドの「用語解説の更新」

▶ アプリケーション一時変更を定義するには

1. 〈Work With Data Dictionary Items〉で、アプリケーション一時変更を定義する項目を確認します。
2. [Row]メニューから[Descript. Overrides（記述一時変更）]を選択します。
3. [Add]をクリックします。

   〈Data Item Descriptions〉フォームが表示されます。

4. 〈Data Item Descriptions〉で、以下のフィールドに入力して、[OK]をクリックします。

   - アプリケーション一時変更

     現在のアプリケーション一時変更に関連付けるシステム・コードを入力します。システム・コードは、システムの特定のカテゴリとアプリケーションに個別に関連付けられます。

   - 記述
   - カラム・タイトル

5. 以上の手順を、データ項目に関連付けるアプリケーション一時変更ごとに繰り返します。

► データ項目を言語に合わせて更新するには

1. 〈Work Wth Data Dictionary Items〉で、更新する項目を確認します。
2. [Row]メニューから[Descript. Overrides]を選択します。
3. [Add]をクリックします。

   〈Data Item Descriptions〉フォームが表示されます。

4. 次のフィールドに値を入力して[OK]をクリックします。

   - 言語
   - 記述
   - カラム・タイトル

5. 以上の手順を、データ項目に関連付ける言語ごとに繰り返します。

データ項目を使用するすべてのアプリケーション（対話型またはバッチ・アプリケーション）でローとカラムのテキストを変更できます。

► ローとカラムのテキストをすべてのアプリケーションで変更するには

1. 〈Work With Data Dictionary Items〉で、更新する項目を確認します。
2. [Row]メニューから[Descript. Overrides]を選択します。
3. 〈Work With Data Item Descriptions（データ項目記述の処理）〉で、言語レコードを選択して修正します。

名称記述を変更するには、[Glossary Overrides]をクリックしてレコードを作成します。その後で〈Glossary（用語解説）〉フォームを使って記述を変更します。

ローとカラムの記述の変更は、データ・レプリケーションによっては複製されません。ローとカラムの変更をワークステーションに配布するには、影響を受けるアプリケーションを含む、新規のフル・パッケージ、部分パッケージ、または更新パッケージを配信する必要があります。新規パッケージまたは更新パッケージにより、ワークステーションのキャッシュに保存されている既存のローとカラムは削除されます。

テーブル設計

リレーショナル・データベース・テーブルは、アプリケーションが使用するデータを保管します。テーブルでは、一連のデータをカラムとローに保管します。各カラムがデータ項目で、各ローがレコードです。テーブルは、1 アプリケーションに対し 1 つ以上作成できます。テーブルを作成するには、テーブルに含めるデータ項目（データ項目は、データ辞書に存在するものとします）を選択し、データの取出し/更新用のインデックスとしてキー・フィールドを割り当てます。J.D. Edwards ソフトウェアでテーブルの存在が認識されるように、テーブルを定義する必要があります。

以下の操作を実行するときには、常に〈Table Design〉を使ってテーブルを生成する必要があります。

- 新規テーブルの作成
- データ項目の追加または削除
- インデックスの追加または変更

テーブルのレコードは、インデックスで識別します。テーブルの固有のレコードは、プライマリ・インデックスで識別します。インデックスは、テーブルの 1 つまたは複数のキー、あるいはデータ項目から構成されます。インデックスにより、DBMS（データベース管理システム）は、レコードの並べ替えと検索を迅速に実行できます。

テーブルの追加

新規のテーブルを追加する前に、アプリケーションで必要なデータ項目が既存のテーブルに含まれているかどうかを判断する必要があります。既存のテーブルが存在しない場合は、新しく作成してください。

新規テーブルを追加するときには、次の監査証跡カラムを含める必要があります。

- User ID（ユーザーID）(USER)
- Program ID（プログラム ID）(PID)
- Machine Key（マシン・キー）(MKEY)
- Date Updated（更新日付）(UPMJ)
- Time of Day（時刻）(UPMT)

▶ テーブルを追加するには

1. 〈Object Management Workbench〉で、[Add]をクリックします。
2. 〈Add J.D. Edwards Object to the Project〉で、[Table（テーブル）]オプションから[OK]を選択します。
3. 〈Add Object〉で、以下のフィールドに入力し、[OK]をクリックします。
   - オブジェクト名
   - 記述
   - システム・コード
   - 製品システム・コード

- カラム・プレフィックス
- オブジェクト使用

4. 〈Table Design〉で、[Summary（集計）]タブをクリックし、以下のフィールドのデータを改訂して、テーブル・プロパティを変更します。
   - 記述
   - システム・コード
   - 製品システム・コード
   - 使用するオブジェクト
5. テーブルに添付を追加するために、[Attachments（添付）]タブをクリックして添付を追加します。

テーブルを追加するときには、次のガイドラインに従ってください。

テーブルのオブジェクト・ライブラリアン名は 8 文字以内で、Fwwxx とします。

F = データ・テーブル

ww（第 2 桁と第 3 桁） = システム・コード。次の例があります。

00 –ファンデーション環境

01 – 住所録

03 – 売掛管理

xx（第 4 桁と第 5 桁） = グループ・タイプ。次の例があります。

01 – マスター

02 – 残高

1X – トランザクション

WorldSoftware でのテーブル命名規則に関する注:
AS/400 のテーブルを使用している場合は、次の命名規則が適用されます。

LA～LZ – 論理ファイル

JA～JZ – 結合テーブル

テーブル記述はテーブルのトピックであり、住所録マスター(F0101)や品目マスター(F4101)など、それが表すファイル名と同じ名前にします。

テーブルの内容を 60 文字以内で記述できます。

カラムのプレフィックスは、テーブルのカラムを識別するための 2 文字のコードです。必ず最初の文字を数字にし、2 番目の文字を英数字にしてください。

テーブルの命名規則

テーブルのオブジェクト・ライブラリアン名は 8 文字以内で、Fxxxxyyy とします。

各変数の意味を次に示します。

F = データ・テーブル

xx（第 2 桁と第 3 桁） = システム・コード。次の例があります。

00 –ファンデーション環境

01 – 住所録

03 – 売掛管理

xx（第 4 桁と第 5 桁） = グループ・タイプ。次の例があります。

01 – マスター

02 – 残高

1X – トランザクション

yyy（第 6 桁から第 8 桁まで） = オブジェクト・バージョン。たとえば、同じような機能を実行しても特定の処理でわずかに異なるプログラムなどがあります。

LA～LZ – 論理ファイル

JA～JZ – 結合テーブル

テーブルを追加するときには、次のガイドラインに従ってください。

テーブルの内容を 60 文字以内で記述します。

テーブル記述がテーブルのトピックになっていることを確認します。テーブルが AS/400 で作成されたものである場合は、住所録マスター(F0101)や品目マスター(F4101)など、それが示すファイル名と同じ名前にします。

カラムのプレフィックスは、テーブルのカラムを識別するための 2 文字のコードです。

必ず最初の文字を英字にし、2 番目の文字を英数文字にしてください。「$」、「#」、「@」などの特殊文字は使用できません。

データ項目名はカラム・プレフィックスに後続します。たとえば、住所録マスター(F0101)の「Address Number（住所番号）」は ABAN8 になります。プレフィックスは、〈Tool Design Aid（ツール設計ツール）〉によって固有にされるので、J.D. Edwards ソフトウェアで固有にする必要はありません。この場合、ABAN8 は、F0101_ABAN8 として参照されます。

インデックス

インデックスにフィールドが 1 つしか存在しない場合は、フィールドを「Address Number」などのインデックス名としてリストします。

共存環境の場合、AS/400 上では ERP インデックスをより論理的なものにする必要があります。〈Table Design〉の［Generate Table（テーブル生成）］コマンドを実行すると、一致するファイルが AS/400 に存在するかどうかが自動的に確認されます。一致する AS/400 ファイルが存在しない場合、AS/400 で論理ファイルが作成されますが、一致する AS/400 ファイルが存在する場合は、作成されません。

インデックスに 2 つのフィールドが存在する場合は、「Address Number, Line Number ID」のようにフィールドを続けて示します。

インデックスに 3 つ以上のフィールドが含まれ、最初の 2 つのフィールドが別のインデックスの最初の 2 つのフィールドと一致する場合は、「Address Number, Line Number, A)」のように、最初の 2 つのフィールドの後に英字(A)を続けます。または、「Item Number, Branch, +」のように、最初の 2 つのフィールドの後にプラス記号(+)を続けます。

各インデックス・フィールドの間、および最後のインデックス・フィールドとプラス(+)記号の間は、カンマとスペースで区切ります。インデックスには 10 個以上のフィールドを含めないでください。

インデックスに複数のフィールドが存在する場合は、インデックス名の長さを 19 文字以内にします。20 文字以上にすると、「Re-definition is not identical…（再定義が一致しません…）」という警告メッセージが表示されます。これはフェッチに影響し、ビジネス関数で間違ったインデックス ID が使用されます。

フィールド記述

フィールド	記述
オブジェクト名	システム・オブジェクトを識別する名前。J.D. Edwards ERP アーキテクチャはオブジェクト指向です。ソフトウェアの個々のオブジェクトはすべてのアプリケーションのビルディング・ブロックとなっており、複数のアプリケーションでオブジェクトを再使用できます。各オブジェクトは、オブジェクト・ライブラリアンで管理されます。 オブジェクトの例は次のとおりです。 o バッチ・アプリケーション（レポートなど） o 対話型アプリケーション o ビジネス・ビュー o ビジネス関数 o ビジネス関数データ構造体 o イベント・ルール o メディア・オブジェクト・データ構造体
メンバ記述	ソフトウェア・バージョン・リポジトリ・ファイル内のレコード記述。メンバ記述は基本メンバの記述と一致させてください。

フィールド	記述
システム・コード・レポート	レポートおよびアプリケーション一時変更のシステム番号を示すユーザー定義コード(98/SY)。
システム・コード	J.D. Edwards のシステムコードを示すユーザー定義コード(98/SY) 。
ファイル接頭辞	特定のシステムに関連づけられたプレフィックス。J.D. EDWARDS のシステム全体を通じてこのフィールド名称を固有のものにするため、データ項目名の前にプレフィックスがつけられます。
機能使用	オブジェクトの使用を指定します。たとえば、オブジェクトはプログラム、マスターファイル、または取引仕訳を作成するのに使用されます。 UDC 98/FU を参照してください。
ソース言語	ビジネス関数が書き込まれるプログラミング言語を識別するソース言語
処理タイプ	処理タイプは、レポート、変換またはバッチ処理などの操作別にオブジェクトをグループ化します。98/E4 UDC テーブルで編集されます。

テーブル設計の処理

〈Table Design〉では、1 つのウィンドウに次のビューが表示されます。

- Table Columns(テーブル・カラム) : テーブルを構成するデータ項目が表示されます。
- Data Dictionary Browser(データ辞書ブラウザ) : データ項目を検索して選択し、[Column(カラム)]ビューに移動できます。
- Index(インデックス) : 固有のデータ項目を定義して、テーブルを迅速にソートし、更新できるようにします。
- Properties(プロパティ) : [Column]ビューで選択したデータ項目の属性が表示されます。これは表示専用ビューで、主にインデックスを定義するときに使用します。

データ項目やインデックスを変更または削除したら、テーブルを再構成する必要があります。それらを変更すると、そのテーブルを参照するビジネス・ビューとフォームに影響する可能性があるためです。

新しく修正したテーブルを生成するには、[Generate(生成)]を使用します。テーブルの既存データは上書きされます。

注意:

テーブルを削除したり、テーブルからカラムを削除すると、そのテーブルや削除されたテーブル・カラムを参照するビジネス・ビューはいずれも無効になり、アプリケーションを生成するときにエラー・メッセージが表示されます。

〈Table Design〉を使用してテーブルを削除した場合、削除されるのはテーブルのスペックのみです。物理テーブルは削除されません。

テーブルのデータ項目の選択

テーブルのカラムとは、アプリケーションで使用されるデータ辞書項目のデータが保管されるところです。データ項目は、データ辞書に存在しなければテーブルで使用することができません。テーブルには、複数のシステム・コードからのデータ項目を含めることができます。

▶ データ項目を選択するには

1. テーブルの追加手順を実行するか、または〈Object Management Workbench〉でテーブルを選択し、中央カラムの[Design(設計)]ボタンをクリックします。
2. 〈Table Design〉で、[Design Tools(設計ツール)]タブを選択して[Start Table Design Aid(テーブル設計ツールの開始)]をクリックします。
3. [Data Dictionary Browser]ビューの QBE フィールドを使用して、テーブルに追加するデータ辞書項目を検索します。
4. データ項目をテーブルに追加するには、そのデータ項目を[Data Dictionary Browser]ビューから[Table Columns]ビューにドラッグします。
5. テーブルからカラムを削除するには、それを選択して[Edit(編集)]メニューから[Delete(削除)]を選択します。

   テーブルのデータ項目をすべて選択した後に、インデックスを定義する必要があります。

インデックスの定義

特定のレコードを手早く検索してソートできるように、インデックスを使用します。テーブルのインデックスは、カード・ファイルのタブのようなものです。各インデックスは 1 つ以上のキーで構成され、各キーはそれぞれがデータ項目です。インデックスを使用すると最も簡単にデータにアクセスでき、順序に従ってデータを読み取る必要はありません。

テーブルには複数のインデックスを定義できますが、すべてのテーブルには主なキーとなるインデックスを 1 つ定義する必要があります。これは、テーブルの各レコードを個別に識別するためのものです。データベースは、最も詳細な情報を返すインデックスを使用する必要がありますが、プライマリ・インデックスを使用するとは限りません。また、ビジネス・ビューを構築するには、プライマリ・インデックスを使用します。

参照

- インデックスに関連するパフォーマンスの問題については、『開発ツール』ガイドの「パフォーマンス」
- ビジネス・ビューでテーブルのプライマリ・インデックスがどのように使用されるかについては、『開発ツール』ガイドの「ビジネス・ビュー設計」

► インデックスを定義するには

1. 〈Table Design〉で、〈Indices（インデックス）〉フォームをクリックして起動します。[Indices]メニューが表示されます。
2. [Indices]メニューから[Add New（追加）]を選択します。

   カラム・ビューからインデックス・ビューにドラッグ・アンド・ドロップする方法もあります。

   インデックス記述は名称なしフィールドであり、プライマリ・インデックスを示す文字 P と共に、鍵マークが付いています。

3. インデックス・タイトルをダブルクリックします。インデックス名を入力して Enter キーを押します。
4. [Table Columns]ビューで、カラム・ビューから 1 つ以上のカラムを選択し、インデックス上にドラッグします。

   固有なインデックスは、単一の鍵マークが付いています。マウスの右ボタンをクリックし、[Index]メニューから[Unique（固有）]を選択すると、キーの固有/非固有ステータスを切り替えることができます。〈Unique Primary Index（固有プライマリ・インデックス）〉は、非固有ステータスに変更できません。

5. [Ascending（昇順）]項目を右クリックして選択または選択解除し、インデックス・カラムのソート順序として昇順または降順を指定します。

   上向き矢印は、インデックス・カラムが昇順にソートされることを示します。

J.D. Edwards 設計規約

インデックス名を指定するときには、次のガイドラインに従ってください。

- インデックスにフィールドが 1 つしか存在しない場合は、フィールドを「Address Number」などのインデックス名で示します。
- 共存環境の場合は J.D. Edwards のインデックスと AS/400 の論理ファイルは、常に一致させてください。〈Table Design〉で[Generate Table（テーブル生成）]コマンドを実行すると、一致する AS/400 ファイルが存在するかどうかが自動的に確認されます。一致する AS/400 ファイルが存在しない場合は AS/400 で論理ファイルを作成し、存在する場合は論理ファイルを作成しません。
- インデックスに 2 つのフィールドが含まれている場合は、「Address Number, Line Number ID」のようにフィールドを続けて示します。
- インデックスに 3 つ以上のフィールドが含まれ、最初の 2 つのフィールドが別のインデックスの最初の 2 つのフィールドと一致する場合は、「Address Number, Line Number, A」のように、最初の 2 つのフィールドの後に英字(A)を続けます。または、「Item Number, Branch, +」のように、最初の 2 つのフィールドの後にプラス(+)記号を続けます。
- 各インデックス・フィールドの間、および最後のインデックス・フィールドとプラス(+)記号の間は、カンマとスペースで区切ります。
- インデックスには 10 個以上のフィールドを含めないでください。
- インデックスに複数のフィールドが存在する場合は、インデックス名の長さを 19 文字以内にします。20 文字以上にすると、「Re-definition is not identical…（再定義が一致しません…という警告メッセージが表示されます。これはフェッチに影響し、ビジネス関数で間違ったインデックス ID が使用されます。

テーブルのプレビュー

テーブルの印刷イメージをプレビューできます。

▶ テーブルをプレビューするには

〈Table Design〉で、〈Columns（カラム）〉フォームをクリックして起動します。次に、[File（ファイル）]メニューから[Print Preview（印刷プレビュー）]を選択します。

テーブルの印刷プレビューが表示されます。

テーブルの処理

〈OMW（オブジェクト管理ワークベンチ）〉を使用して、テーブルを一元管理します。

テーブルの生成

データ項目を選択してテーブルのインデックスを設定すると、特定のデータ・ソースに対するテーブルを構成することができます。インデックスが未設定の場合は、生成プロセスにエラーが発生します。

物理テーブルを作成するには、テーブルを生成する必要があります。テーブルを生成するまでは、テーブルを追加することも更新することもできません。テーブルを生成すると、ビジネス関数とテーブル・イベント・ルールのコンパイルで使用する.H ファイルも作成されます。

OMW は、〈Object Configuration Manager（オブジェクト構成マネージャ）〉アプリケーション(P986110)にアクセスしてテーブルを構成します。既存のデータ・ソースのどこにでも生成できます。データ・ソースを指定しないと、デフォルト・マップのテーブルが自動的に構成されます。パス・コードを変更すると、テーブルはそのロケーションに生成できます。この場合は、[Remove Table（テーブル削除）]を選択した場合と同様に DROP ステートメントが実行され、その後でテーブルが再作成されます。現在のテーブルを再生成すると、それに含まれているデータは失われます。

データ項目の削除や追加などを行ってテーブルを変更した場合は、テーブルを再生成する必要があります。データを紛失しないようにするには、データをエクスポートし、テーブルを生成してデータをコピーし直す必要があります。

▶ テーブルを生成するには

1. 〈Table Design〉で、[Table Operations（テーブル操作）]タブを選択して[Generate Table（テーブル生成）]をクリックします。
2. 〈Generate Table（テーブルの生成）〉で、次のフィールドに入力して[OK]をクリックします。
   - データ・ソース
   - パスワード

   生成処理が正常終了したかどうかを示すメッセージが表示されます。

インデックスの生成

追加のインデックスを作成したり既存のインデックスを変更する場合は、それらを再生成する必要があります。これにより、.H ファイルは変更されますが、テーブル全体を再生成する場合のように既存のデータは失われません。

▶ インデックスを生成するには

1. 〈Table Design〉で、[Table Operations]タブをクリックして[Generate Indexes（インデックス生成）]をクリックします。
2. [Generate Indexes]で、次のフィールドに入力して[OK]をクリックします。
   - データ・ソース
   - パスワード

ヘッダー・ファイルの生成

テーブルにはヘッダー・ファイルが存在しないものもあります。テーブル全体を生成しなくても、ヘッダー・ファイルは生成することができます。

▶ ヘッダー・ファイルを生成するには

〈Table Design〉で、[Design Tools]タブを選択して[Generate Header File（ヘッダー・ファイル生成）]をクリックします。

ヘッダー・ファイルが生成されます。

テーブルのコピー

テーブルは、データ・ソース間でコピーできます。この操作では、テーブル・スペックはコピーされません。テーブルをデータ・ソース間でコピーするには、テーブル変換を使用することもできます。

参照

- 〈Table Conversion（テーブル変換）〉ツールを使用してテーブルをコピーする方法については、『テーブル・コンバージョン』ガイドの「テーブル入力とデータのコピー」

▶ テーブルをコピーするには

1. 〈Table Design〉で、[Table Operations]タブを選択して[Copy Table（テーブルのコピー）]をクリックします。
2. 次のフィールドに値を入力して[OK]をクリックします。
   - データ・ソース
   - データ・ソース
   - オブジェクト所有者 ID
   - パスワード

フィールド記述

フィールド	記述
データ ソース	ONEWORLD は、基本データソースまたは基本データソース内のデータ項目が見つからない場合、このデータソースを使用します。
オブジェクト 所有者	データベース・テーブルの接頭辞または所有者

データベースからのテーブルの削除

テーブルをシステムから完全に削除するには、OMW 機能の[Remove Table from Database(データベースからのテーブル削除)]を使用する必要があります。〈Table Design〉を使用してテーブルを削除した場合は、テーブルのスペックしか削除されません。〈Table Design〉はテーブルを物理的に削除できません。

▶ テーブルをデータベースから削除するには

1. 〈Table Design〉で、[Table Operations]タブを選択して[Remove Table from Database]をクリックします。
2. 〈Remove Table(テーブルの削除)〉で、次のフィールドに入力して[OK]をクリックします。
   - データ・ソース
   - パスワード

テーブルのデータの表示

異なるデータベースのテーブルのデータを表示する場合は、〈Universal Table Browser(ユニバーサル・テーブル・ブラウザ)〉を利用することができます。このツールでは、テーブルの構造を判別するだけでなく、テーブルのデータの存否を確認できます。〈Universal Table Browser〉では、アクセスするデータベースから独立した状態で、JDEBASE API を使用してデータベースからデータを取り出します。

注:

ユーザーによる〈〈Universal Table Browser〉へのアクセスを禁止する必要がある場合は、J.D. Edwards セキュリティを直接使用することはできません。これは、〈Universal Table Browser〉が J.D. Edwards ツールで生成されたアプリケーションではなく、Windows の実行可能アプリケーションであるためです。代わりに、フォーム・セキュリティを〈Table and Data Source Selection(テーブル/データ・ソースの選択)〉フォーム(W98TAMC)に設定することができます。〈Universal Table Browser〉はこのフォームなしでは動作できないため、〈Universal Table Browser〉にセキュリティが設定されたことになります。〈Security Workbench(セキュリティ・ワークベンチ)〉によって設定したカラムとローのセキュリティは、〈Universal Table Browser〉にすべて適用されます。

► テーブルのデータを表示するには

〈Cross Application Development Tools（クロス・アプリケーション開発ツール）〉メニュー(GH902)から〈Universal Table Browser〉を選択します。

1. 〈Universal Table Browser〉で、[File]メニューから[Open Table（テーブルを開く）]を選択します。
2. 次の必須フィールドに値を入力します。
   - Table（テーブル）
   - Data Source Name（データ・ソース名）
3. 次のオプションをクリックします。
   - Format Data（フォーマット・データ）
4. [OK]をクリックします。

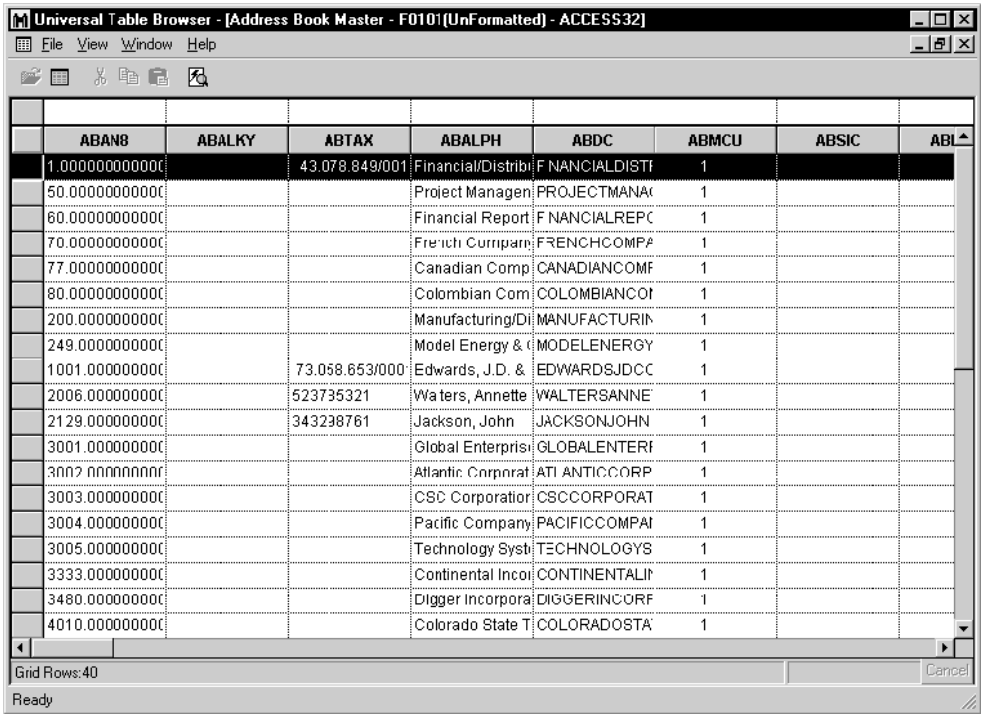
フィールド記述

記述	用語解説
テーブル	システム・オブジェクトを識別する名前。J.D. Edwards ERP アーキテクチャはオブジェクト指向です。ソフトウェアの個々のオブジェクトはすべてのアプリケーションのビルディング・ブロックとなっており、複数のアプリケーションでオブジェクトを再使用できます。各オブジェクトは、オブジェクト・ライブラリアンで管理されます。 オブジェクトの例は次のとおりです。 o バッチ・アプリケーション（レポートなど） o 対話型アプリケーション o ビジネス・ビュー o ビジネス関数 o ビジネス関数データ構造体 o イベント・ルール o メディア・オブジェクト・データ構造体 --- フォーム固有 --- ONEWORLD テーブル名を指定します。F0101 は住所録マスターです。 ビジュアル・アシストを使うと、フォームの検索や選択、およびテーブルの検索が可能です。

データ・ソース	データ・ソースの名前。 --- フォーム固有 --- テーブルが中に存在する有効なデータソース。このデフォルト値は、ユーザーがサインオンしている環境内の OCM 設定から取得します。データ検索フォームを使用可能にして、ONEWORLD データソースを選択するには、ビジュアル・アシストを使用します。
データのフォーマット	"YES"または"NO"入力フィールドは、プロンプト画面で"YES"または"NO"と簡単に応答するための共通の一文字入力フィールドです。 --- フォーム固有 --- ユニバーサルテーブルブラウザでデータの位置をフォーマットする（デフォルト）か、ローデータを表示するか指定します。 フォーマットあり。ユニバーサルテーブルブラウザで ONEWORLD データ辞書項目のスペックに従ってデータを表示します。たとえば、データ項目の「PROC」のサイズが 15 の数値フィールドで、4 つの表示小数点以下桁数があるとします。56.2185 の場合、ユニバーサルテーブルブラウザでは、56.2185 のように（データ辞書編集を使って）フォーマットされた値が表示されます。この値が 562185 としてデータベースに保存されている場合でも同じです。 フォーマットなし。ユニバーサルテーブルブラウザでデータベースおよびデータを取り込むロケーションのデータ項目タイプ（数値など）のスペックに従ってデータを表示します。たとえば、テーブルデータ項目の「PROC」はデータベースに保管される数値フィールドです。データベースによっては、このフィールドのデフォルトサイズが 32 で精度が 15 の数値データタイプの場合もあります。ONEWORLD は、データベースに表示小数点以下桁数を保管しないため、56.2185 という値は、データベースに 562185.000000000000000 として保管され表示されます。

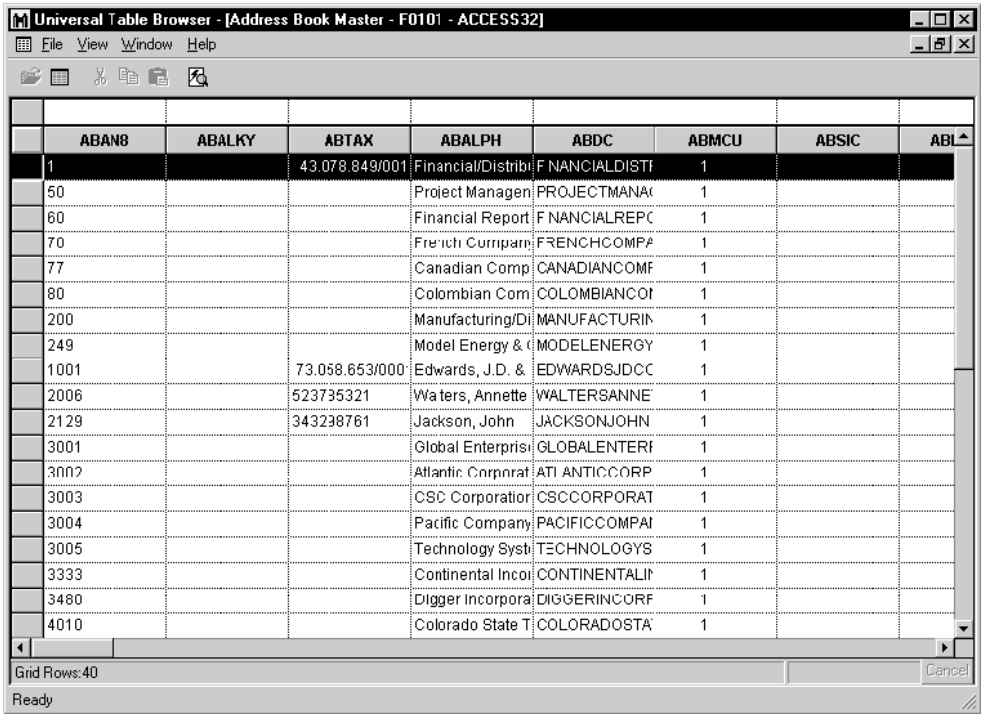
例：Universal Table Browser（未フォーマット・データ）

この例は、[Format Data（フォーマット・データ）]オプションをオフにして表示したときのデータベース・テーブルです。住所録マスター(F0101)の ABAN8 カラムの表示状態に注目してください。この情報は未フォーマットで、データベースに保管されたままの状態で表示されています。



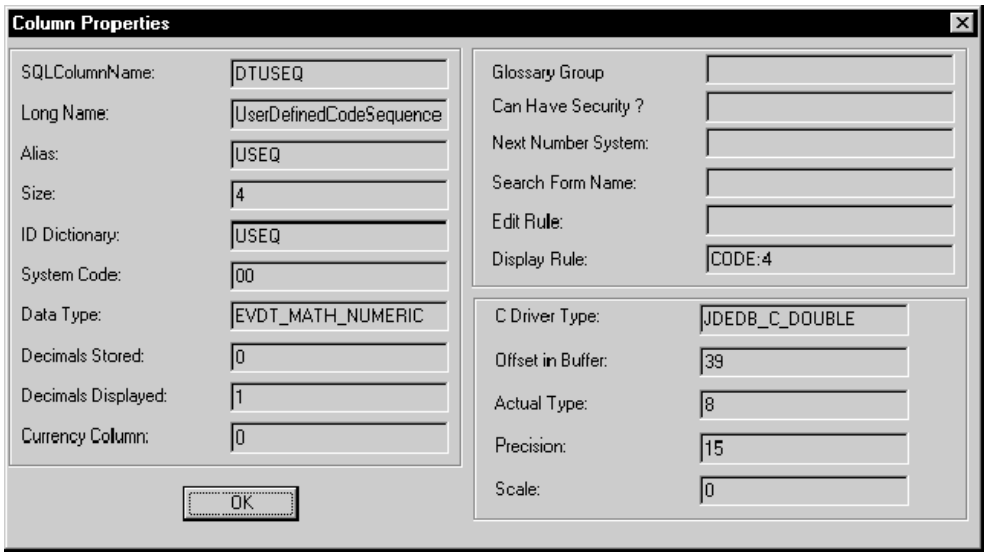
例：Universal Table Browser（フォーマット済みデータ）

この例は、[Format Data]オプションをオンにして表示したときのデータベース・テーブルです。住所録マスター(F0101)の ABAN8 カラムの情報が、データ辞書スペックに従ってフォーマットされています。



例:カラム・プロパティ

この例は、J.D. Edwards データ辞書項目の USEQ のカラム・プロパティを示しています。この J.D. Edwards 項目の SQL データベース名は DTUSEQ です。



ビジネス・ビュー設計

ビジネス・ビューとは、1 つ以上のテーブルからデータ項目が選択される論理的なビューです。テーブルの作成後に〈Business View Design(ビジネス・ビュー設計)〉を使用して、アプリケーションに必要なデータ項目のみを選択します。定義したビジネス・ビューを使用して、サポートされる J.D. Edwards ソフトウェア・データベースからのデータ取出しに必要となる適切な SQL ステートメントが生成されます。このビジネス・ビューを定義すると、対話型アプリケーションでデータを更新するフォームを作成したり、データを表示するレポートを設計できます。アプリケーションに必要なデータ項目のみを選択するため、ネットワークでのデータ移動が減少します。

ビジネス・ビューは、アプリケーションの構築やレポートの生成に必要となるだけでなく、次の特性を備えています。

- 1 つ以上のテーブルの一部またはすべてのデータ項目を含めることができる。
- J.D. Edwards アプリケーションを 1 つ以上のテーブルにリンクする。
- アプリケーションで使用される複数のテーブルからデータ項目を(結合テーブルまたは合併テーブルとして)定義する。

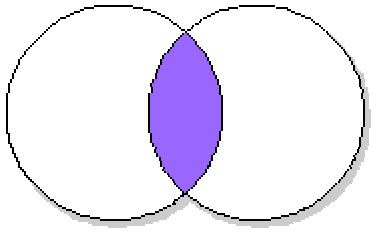
テーブルの結合

テーブルの結合によって、結合されるテーブルの各ローのデータが組み合わされます。結合カラムは、ローのキー・カラムに同じ値がある場合など、結合条件を満たすカラムです。プライマリ・テーブルは結合元のテーブル（通常は〈Table Design〉の左にあるテーブル）で、セカンダリ・テーブルは結合先のテーブル（通常は〈Table Design〉の右にあるテーブル）です。結合には次のようなタイプがあります。

- 単純結合。内部結合とも呼ばれます。この種の結合の場合は、プライマリ・テーブルとセカンダリ・テーブルの両方と一致するローのみが含まれます。

次の図は、テーブルの単純結合を示しています。

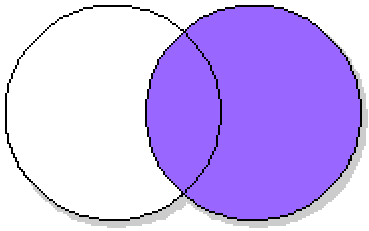
単純結合



- 右外部結合。この結合の場合は、プライマリ・テーブルとセカンダリ・テーブルの両方に共通するロー、およびセカンダリ・テーブルにのみ存在するローが含まれます。

次の図に、テーブルの右外部結合を示します。

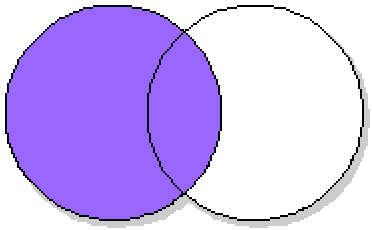
右外部結合



- 左外部結合。この種の結合の場合は、プライマリ・テーブルとセカンダリ・テーブルの両方に共通するロー、およびプライマリ・テーブルにのみ存在するローが含まれます。

次の図は、テーブルの左外部結合を示しています。

左外部結合



左外部結合と SQL 92 の左外部結合に関する注:

Left Outer Join（左外部結合）は、右側と一致しないレコードが常に含まれる以外は標準 SQL 92 Left Outer Join と同じです。右側カラムのクエリーは無視されます。

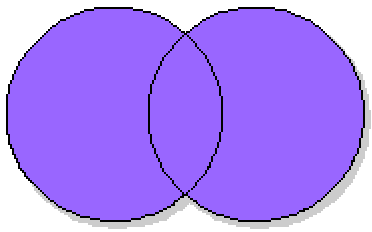
右側カラムの値がヌルとなっているレコードは、右側カラムの WHERE クローズにかかわらず常にセクションに含まれます。

テーブルの合併

テーブルの合併では、テーブル全体を結合します。最初にプライマリ・テーブルのローがチェックされた後、セカンダリ・テーブルで対応するカラムをもつローがチェックされます。2 つのテーブルのローで同じデータが存在する場合は、一方のレコードのみが取り出されます。

合併では、プライマル・テーブルからのロー、およびセカンダリ・テーブルから対応するカラムが取り出されます。

合併



Select Distinct（抽出選択）

［Select Distinct（抽出選択）］機能を使用すると、ビジネス・ビューの照会時にローの重複を解消することができます。

インデックス

ビジネス・ビューを構築するには、テーブルのプライマリ・インデックスを使用します。ビジネス・ビューによって、テーブルからアプリケーションに情報が送られます。プライマリ・キー以外の追加情報をアプリケーションに送る必要がある場合は、ビジネス・ビュー・インデックスの変更が必要になることがあります。

ビジネス・ビューの追加

ビジネス・ビューの設計を始める前には、アプリケーションの用途とそのアプリケーションが必要とするデータ項目について考えてください。そして、それらのデータ項目が J.D. Edwards のどのテーブルに含まれているかを確認します。新規のビジネス・ビューを追加してもパフォーマンスには影響しません。ただし必要以上に多数のカラムを含む既存のビューを使用すると、パフォーマンスが低下する場合があります。

ビジネス・ビューには通常、フォームやグリッドよりも多くのフィールドが存在します。ビジネス・ビューに含まれているフォームやグリッドに含まれていないフィールドが必要な場合は、ビジネス・ビューを変更せずに、フォームやグリッドにそのフィールドを追加できます。

ビジネス・ビューは、フォーム・タイプごとに異なるものを使用します。グリッド付きのフォームでは、パフォーマンスに影響しかねない業務固有の問題を考慮して、グリッドのフィールド数をビジネス・ビューよりも少なくします。一般に、検索/選択フォームは、項目数を必要最小限にしてコンパクトにします。検索のフィルタ処理に必要な基本フィールド、および（記述のように）必要な出力フィールドのみを組み込む必要があります。

検索/表示フォームと親/子フォームは、項目数が若干増えて中位のサイズになります。フィルタ処理に必要なフィールドと必須の出力フィールドのみを組み込んでください。

入力可能なフォームには、テーブルからのすべての項目があるため大きくなる傾向があります。これらには、監査情報を含む、レコードの追加や更新に必要なすべてのフィールドを設定してください。

参照

- ビジネス・ビューのパフォーマンスについては、『開発ツール』ガイドの「パフォーマンス」

► ビジネス・ビューを追加するには

1. 〈Object Management Workbench〉で、[Add]をクリックします。
2. 〈Add J.D. Edwards Object to the Project〉で、[Business View(ビジネス・ビュー)]オプションを選択して[OK]をクリックします。
3. 〈Add Object〉で、次のフィールドに入力して[OK]をクリックします。
   - オブジェクト名
   - 記述
   - システム・コード
   - 製品システム･コード
   - オブジェクト使用

J.D. Edwards 命名規約

ビジネス・ビューに名前を付けるときには、次のガイドラインに従ってください。

ビジネス・ビューの〈Object Management Workbench〉名は 10 文字以内で次のように指定します。

VzzzzzzzzA とします。

各文字は次の意味を表します。

V = ビジネス・ビュー

zzzzzzzz = プライマリ・テーブルを表す文字

A = ビューを指定する文字

たとえば、V0101A は住所録マスター(F0101)の最初のビューを表し、V0101B と V0101C はそれぞれ同じテーブルの 2 番目と 3 番目のビューを表します。

外部開発に関する考慮事項

外部開発とは、特定のクライアントのためにカスタム・アプリケーションを作成する開発パートナーやコンサルタントなどの開発者によるアプリケーションの作成を意味します。J.D. Edward オブジェクトと非 J.D. Edwards オブジェクト間の衝突を防ぐために、外部ビジネス・ビューの名前を設定するときには、注意する必要があります。外部アプリケーション用の新規ビジネス・ビューを作成する場合は、次の形式のビジネス・ビュー名を使用してください。

Vssss9999

各文字は次の意味を表します。

V = ビジネス・ビュー

ssss = 企業用のシステム・コード

9999 = 企業で固有の自動採番または文字パターン

ビジネス・ビューを表す記述を 60 文字以内で指定します。記述では、〈Item Master Browse（品目マスターのブラウズ）〉や〈Item Master Revisions（品目マスターの改訂）〉など、アプリケーション記述にフォームのタイプがわかるようにします。

プライマリ固有キー・フィールドはビジネス・ビューでも使用します。このフィールドは再編成しないでください。

注：

全カラムを含むビジネス・ビューの作成は、テーブルごとに少なくとも 1 つとします。このビジネス・ビューは、ファイルが基づくすべてのレポートのレベル 01 セクションで使用します。またビジネス・ビューは、見出し/詳細フォームを除いて各フォーム・タイプごとに 1 つのみ使用できます。見出し/詳細フォームの場合は、フォームの見出し部分に 1 つと詳細部分に 1 つ、合計 2 つのビジネス・ビューを使用することができます。

結合ビュー

結合ビューの名前は、結合される 2 つのテーブルの名前をスラッシュで区切って設定します。プライマリ・テーブルを最初に置きます。

たとえば、F4101 と F4102 の結合ビューで F4101 がプライマリ・テーブルの場合は、命名規則に従って「F4101/F4102」とします。

フィールド記述

記述	用語解説
オブジェクト名	システム・オブジェクトを識別する名前。J.D. Edwards ERP アーキテクチャはオブジェクト指向です。ソフトウェアの個々のオブジェクトはすべてのアプリケーションのビルディング・ブロックとなっており、複数のアプリケーションでオブジェクトを再使用できます。各オブジェクトは、オブジェクト・ライブラリアンで管理されます。 オブジェクトの例は次のとおりです。 o バッチ・アプリケーション（レポートなど） o 対話型アプリケーション o ビジネス・ビュー o ビジネス関数 o ビジネス関数データ構造体 o イベント・ルール o メディア・オブジェクト・データ構造体
記述	ソフトウェア・バージョン・リポジトリ・ファイル内のレコード記述。 メンバ記述は基本メンバの記述と一致させてください。

システム・コード	システム・コードを示すユーザー定義コード(98/SY)
製品システム・コード	レポートおよびアプリケーション一時変更のシステム番号を示すユーザー定義コード(98/SY)。
オブジェクト使用	オブジェクトの使用を指定します。たとえば、オブジェクトはプログラム、マスターファイル、または取引仕訳を作成するのに使用されます。 UDC 98/FU を参照してください。

ビジネス・ビュー設計の処理

〈Business View Design〉ツールは、次のビューを表示します。

- Table Joins(テーブル結合):ビジネス・ビューを作成するテーブルを定義します。
- Available Tables(使用可能テーブル):テーブルを検索して[Table Joins]ビューに移動します。
- Selected Columns(選択済みカラム):ビジネス・ビューに含まれているテーブルのデータ項目を表示します。
- プロパティ

新規のビジネス・ビューを作成したり、ビジネス・ビューにデータ項目を追加するときは、ビジネス・ビューを生成する必要があります。また、ビジネス・ビューを処理する際には次の点に注意してください。

- ビジネス・ビューからデータ項目を削除するときに、そのデータ項目がアプリケーションで使用されている場合は、関連アプリケーションの実行時にエラーが発生します。このエラーが発生した場合は、アプリケーションを開き、そのデータ項目をアプリケーションから削除するか、または現在のビジネス・ビューでカラムを再選択してコントロールを修正してください。
- ビジネス・ビューからテーブル全体を削除すると、そのビジネス・ビューを使用するアプリケーションは実行できません。この場合は、ビジネス・ビューを参照するすべての項目を修正してください。
- ビジネス・ビューを削除すると、それを使用するすべてのフォームで障害が発生します。この場合は、フォームを新しいビジネス・ビューに関連付けてすべてのコントロールを接続してください。

ビジネス・ビューを作成するためのテーブルの選択

ビジネス・ビューを作成するには、1 つ以上のテーブルを選択する必要があります。アプリケーションに必要なカラムのみを取り出して更新するために、1 つの大きいテーブルのビジネス・ビューを作成することもできますが、極端に大きいテーブルを取り出して更新する際のパフォーマンスが低下します。可能な場合は、多数のデータ項目を含む大型テーブルを 1 つ作成するのではなく、小型のテーブルを 2 つ作成して結合することを検討してください。

► ビジネス・ビューを作成するためのテーブルを選択するには

1. 〈Object Management Workbench〉で、新しいビジネス・ビューを作成するか、既存のビジネス・ビューを選択して、中央カラムの[Design(設計)]ボタンをクリックします。
2. [Design Tools(設計ツール)]タブを選択して、[Start the Business View Design Aid(ビジネス・ビュー設計ツールの開始)]をクリックします。
3. 〈Business View Design〉の[Available Tables]ペインで、次のフィールドに値を入力して[Find]をクリックします。
   - Description(記述)
   - Object Name(オブジェクト名)
   - Product Code(システム・コード)
4. 1 つ以上のテーブルを選択し、[Table Joins]ペインにドラッグします。このペインは、複数のテーブルを結合するかどうかにかかわらず、[Table Join]と呼ばれます。

   [Table Joins]ペインには、選択したテーブルと各テーブルを構成するカラムが表示されます。テーブルのインデックス・キーとなっているカラムの横には、鍵のマークが表示されます。プライマリ・テーブルは、ビジネス・ビューへのキーを提供します。ここから検索が始まります。

   注:
   アプリケーションの最大パフォーマンスを確保するために、テーブルの結合数は次のように制限されています。

   - すべての結合が単純結合の場合は 5 つのテーブル
   - 外部結合や合併が含まれている場合は 3 つのテーブル

5. 必要なテーブルのタイトル・バーをダブルクリックしてプライマリ・テーブルを指定します。

   この手順は任意です。ビジネス・ビューに複数のテーブルが含まれている場合は、最初に追加したテーブルが自動的にプライマリ・テーブルとして認識されます。ウィンドウの左上隅には、プライマリ・テーブルであることを示す王冠マークが表示されます。ビジネス・ビューにテーブルが 1 つしか含まれていなければ、デフォルトでそのテーブルがプライマリ・テーブルとなります。

   注:
   ビジネス・ビューからテーブルを削除するには、それを選択して[Table]メニューから[Delete]を選択するか、またはテーブルを右クリックしてポップアップ・メニューから[Delete]を選択します。

ビジネス・ビューに対するデータ項目の選択

ビジネス・ビューに使用する 1 つ以上のテーブルを選択し、プライマリ・テーブルを指定し終わったら、そのビジネス・ビューに含めるデータ項目を選択する必要があります。プライマリ・テーブルのすべてのデータ項目、およびプライマリ・テーブルと結合されたテーブルは、そのビジネス・ビューで使用することができます。

対話型アプリケーションまたはバッチ・ジョブに使用するデータ項目を選択します。アプリケーションの作成時には、ビジネス・ビューのすべての項目を使用しなくてもかまいません。

データ項目の選択に関する重要な注:

- 2 つのテーブルが結合されている場合は、両方のプライマリ・インデックスが自動的に選択されます。選択したテーブルのインデックス・キーは、ビジネス・ビューで使用されていることを示すために自動的にハイライトされます。プライマリ・テーブルのインデックス・キーを除き、ビジネス・ビューで使用しないインデックス・キーは削除できます。プライマリ・テーブルに対するインデックス・キーは、ビジネス・ビューから削除することができません。
- 結合を生成する場合は、両方のテーブルの項目が自動的に選択されます。同じデータ項目が複数のテーブルに表示される場合は、1 つのテーブルからのみデータ項目を選択する必要があります。
- 異種のデータ項目は結合できません。
- 同じデータ項目を 2 つのテーブルから選択しないでください。そのようにすると、データ項目がビジネス・ビューに 2 回表示されることになります。

► ビジネス・ビューに含めるデータ項目を選択するには

1. 〈Business View Design〉の[Table Joins]ペインで、ビジネス・ビューに含めるデータ項目をダブルクリックします。

   選択したデータ項目が〈Table Joins〉ペインでハイライトされます。各項目を選択するたびに、その項目が〈Selected Columns〉に追加されます。

   注意:
   ビジネス・ビューで選択するカラム数は、256 個以内にしてください。257 以上のカラムを含むビジネス・ビューを使用するアプリケーションは、実行時に障害が発生します。

2. ビジネス・ビューからデータ項目を削除するには、その項目を[Table Joins]ウィンドウでダブルクリックします。この操作によって、ビジネス・ビューから削除します。

Select Distinct の使用

ビジネス・ビューがプライマリ・テーブルのプライマリ・キーを含む場合（デフォルトのインプリメンテーション）、ビジネス・ビューの照会のローはすべて固有になります。プライマリ・キーの値は、プライマリ・テーブルのローごとに異なります。ビジネス・ビューがプライマリ・テーブルのプライマリ・キーを含んでいないと、ビジネス・ビューの照会時に重複ローが生じることがあります。ビジネス・ビューの設計時に[Select Distinct]機能を使用すると、ビジネス・ビューの照会で重複ローを解消することができます。

たとえば、〈Journal Entry（仕訳入力）〉では、伝票での行 No.で区別されます。この場合、伝票のすべての行 No.ではなく、行 No. 1 の最初の伝票番号だけが必要になります。[Select Distinct]機能は、伝票のすべての行 No.ではなく、最初の伝票番号のみを取り出します。

以下のカラムは通貨のサポートやセキュリティ機能のために使用されますが、これらが 1 つ以上存在するプライマリ・テーブルを使用するビジネス・ビューでは、[Select Distinct]機能により重複する値が出力される場合があります。

CO	会社
CRCD	通貨コード – FROM 通貨
CRDC	通貨コード – TO 通貨
CRCX	通貨コード – 指定
CRCA	通貨コード – A/B 金額
LT	元帳タイプ
AID	略式 ID
MCU	ビジネスユニット
KCOO	発注会社（会社コード）
EMCU	ビジネス・ユニット見出し
MMCU	事業所
AN8	住所番号

▶ [Select Distinct]を使用するには

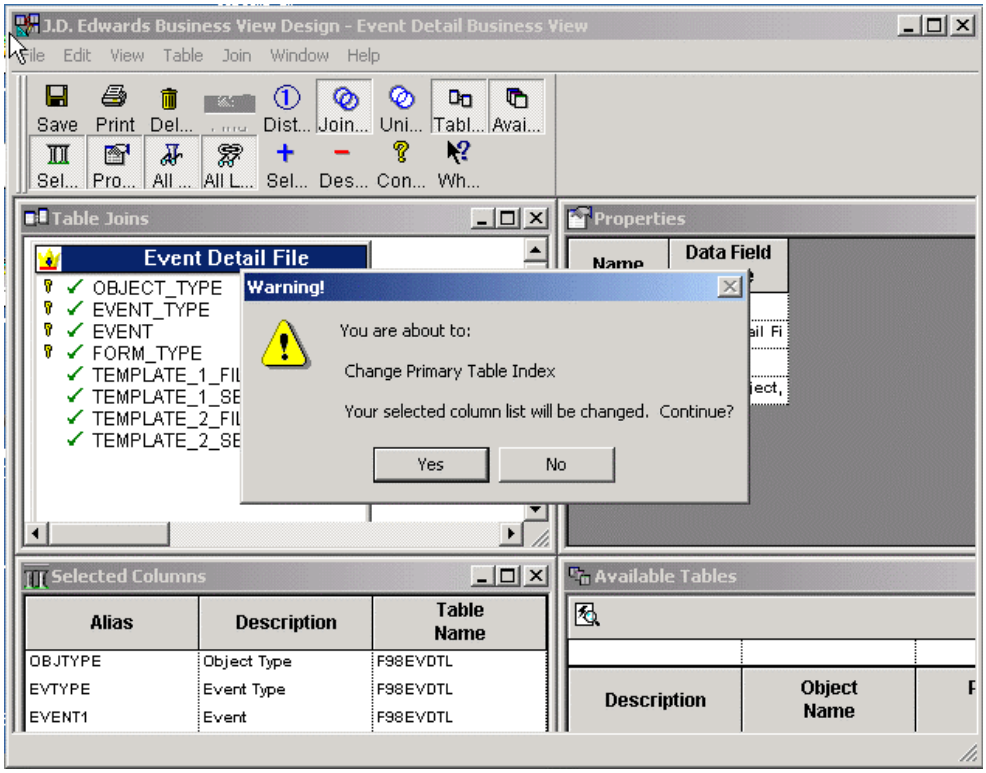
1. 〈Business View Design Aid(ビジネス・ビュー設計ツール)〉で、ビューのプライマリ・テーブルを選択します。
2. [Table]メニューから[Distinct Mode(抽出モード)]を選択します。
3. [Table]メニューから[Change Index(インデックスの変更)]を選択し、プライマリ・テーブルのインデックスを非固有インデックスに変更します。

処理の例:[Select Distinct]機能

ここでは[Select Distinct]機能の使用例を紹介します。この例に使用するのはイベント詳細ビジネス・ビュー(V98EVDTL)で、デフォルトでプライマリ・テーブルであるイベント詳細ファイル・テーブル(V98EVDTL)のプライマリ・インデックスが使用されます。J.D. Edwards テーブルのプライマリ・インデックスは、固有です。そのため、ビジネス・ビューの照会を生成するときに、重複する値は返されません。また、〈Business View Design Aid〉では、ビジネス・ビューを処理する際にプライマリ・テーブルの他のあらゆるインデックスを使用できます。[Select Distinct]機能の動作を確認する次の手順を実行します。

4. 〈Business View Design Aid〉で、プライマリ・テーブルを選択します。
5. [Table]メニューから[Change Index]を選択して、プライマリ・テーブルのインデックスを変更します。

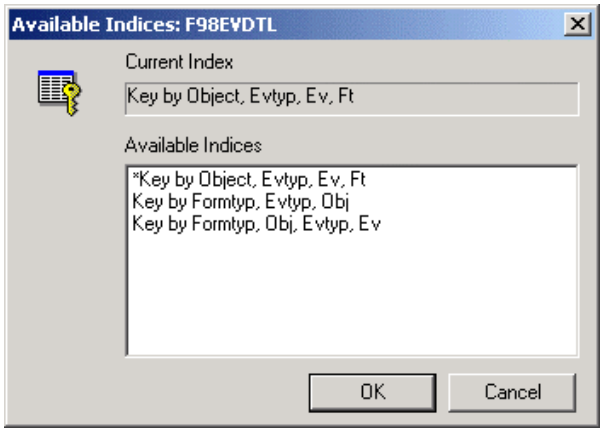
   [Change Index]はプライマリ・テーブルにのみ使用できます。



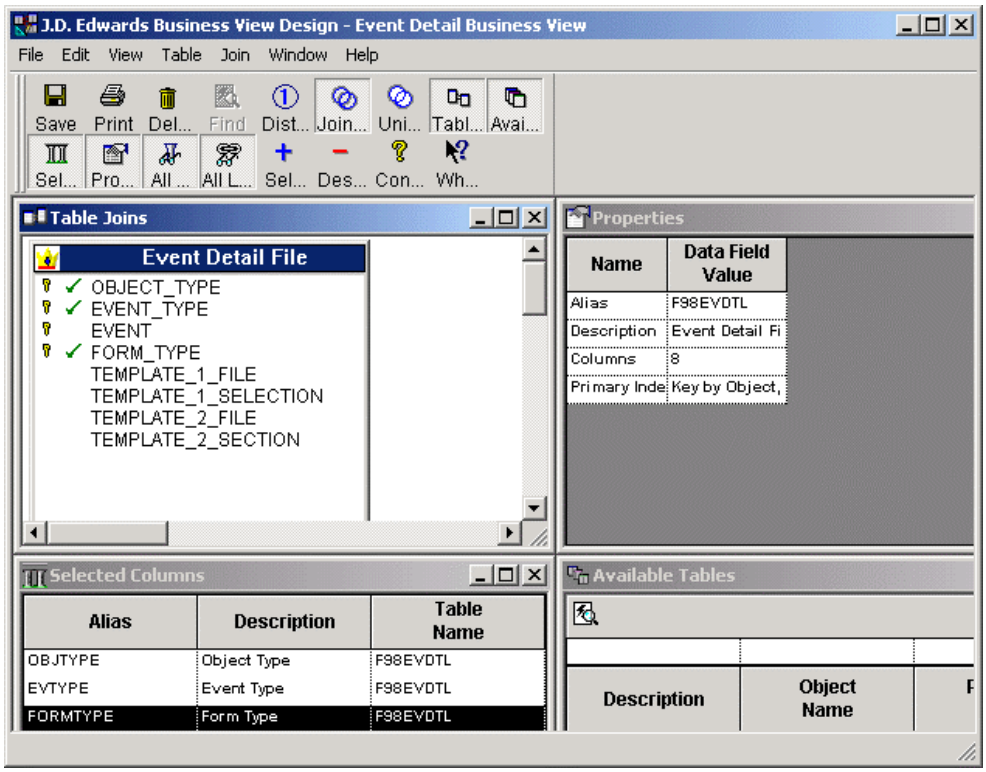
「選択されたカラム・リストが変更されます」という意味の警告メッセージが表示されます。

6. [Yes(はい)]を選択して、次の操作に進みます。

〈Available Indices(使用可能インデックス)〉フォームが表示されます。このフォームの最初の編集フィールドには、ビジネス・ビューで使われているテーブルの現在のインデックスが表示されます。デフォルトはプライマリ・インデックスです。



7. この例では、〈Available Indices〉から[Key by Formtyp, Evtype, Obj]を選択して[OK]をクリックします。



[Table Joins]リストと[Selected Columns]リストが変更されて、新しいインデックスが反映されます。

8. 変更を保存して、〈Business View Design Aid〉を終了します。

   ここで、[Select Distinct]をオフにし、変更されたビジネス・ビューのインデックス「Key by Formtyp, Evtyp, Obj」を使用して、V98EVDTL ビジネス・ビューを使用するアプリケーションを実行すると、次の SQL ステートメントが生成されます。

   ```
   SELECT EDOBJTYPE, EDEVTYPE, EDFORMTYPE FROM PVC. F98EVDTL
   ```

   この例を使用すると、テーブル F98EVDTL から 281 のロー・データを取り出すことができます。

9. V98EVDTL ビジネス・ビューを再び開きます。
10. [File]メニューから[Select Distinct]を選択します。
11. [Change Index]を選択し、〈Available Indices〉から「Key by Formtyp, Evtyp, Obj」のインデックスを再び選択して[OK]をクリックします。
12. ビジネス・ビューを保存して、〈Business View Design Aid〉を終了します。

    ソフトウェアの終了と再起動が必要な場合があります。J.D. Edwards ソフトウェアはビジネス・ビューをキャッシュ・メモリに保管するため、ビジネス・ビューを変更しても、そのビジネス・ビューはキャッシュ・メモリからクリアされるまで動作し続けるからです。

    [Select Distinct]をオンにし、変更されたビジネス・ビューのインデックス「Key by Formtyp, Evtyp, Obj」と、処理したばかりの V98EVDTL ビジネス・ビューを使用して同じアプリケーションを生成し、再実行します。次の SQL ステートメントが生成されます。

    ```
    SELECT DISTINCT EDOBJTYPE, EDEVTYPE, EDFORMTYPE FROM PVC.
    F98EVDTL
    ```

この例を使用すると、イベント詳細ファイル・テーブル(F98EVDTL)から 53 のロー・データのみを取り出すことができます。

結合テーブルの作成

単一アプリケーションで複数テーブルにアクセスできるように、結合テーブルを作成します。

結合は、一般に検索/表示フォームやレポートなど、データの入力をしないフォームで使用します。データベースを更新したり追加したりするフォームでは、レコード間の関係が厳密でなければならないため、通常は使用しません。2 つのテーブル間の関係が単純なときには入力可能フォームでも結合を使用します。結合では、フェッチを個別に使用するよりもすばやくデータを取り出せます。

ビジネス・ビューで複数のテーブルを使用する場合は、各テーブルのカラムどうしの結合を設定して、リンクする必要があります。このリンクは、あるテーブルのローと別のテーブルのローの対応関係を示します。

データ項目の属性は、データ辞書で定義されています。結合時にそのデータ項目を使用するかどうかを決める際、そのカラムの属性を表示するには〈Properties(プロパティ)〉フォームを使用してください。〈Selected Columns〉フォームでデータ項目をハイライトすると、そのデータ項目のプロパティが〈Properties〉フォームに表示されます。

各テーブルを検討し、アプリケーションやレポートに必要な各テーブルのデータと他のテーブルのデータとの関係を設定します。カラムを追加したりインデックスを作成したり、新規テーブルの作成が必要になることもあります。新規のインデックスを作成する場合は、前もって必要事項を慎重に検討してください。

▶ 結合テーブルを作成するには

1. 〈Business View Design〉で、プライマリ・テーブルのカラムと関連テーブルのカラムをクリックし、その 2 つのカラムをリンクする線を描きます。

   結合テーブルは同種のカラム間でしか作成できません。カラム名は異なってもかまいませんが、[Data Type(データ・タイプ)]と[Decimals(小数点以下桁数)]の属性は同じにしてください。データ項目が結合候補かどうかを判断するには、[Table Join]ペインでデータ項目をクリックし、データ項目ごとにポップアップ・ヒントに表示されるデータ項目属性を確認します。

2. 結合を削除するには、結合を選択して、[Join(結合)]メニューから[Delete]を選択するか、または結合を右クリックして、ポップアップ・メニューから[Delete]を選択します。
3. [Join]メニューから[Type(タイプ)]を選択し、次の結合タイプのうちいずれかを選択します。
   - Simple(単純)
   - Left Outer(左優先結合)
   - Right Outer(右優先結合)

   デフォルトの結合タイプは[Simple]です。

4. [Join]メニューから[Operators(演算子)]を選択し、次の演算子のうちいずれかを選択します。
   - Equal(等しい)(=)
   - Less than(より小さい)(<)
   - Greater than(より大きい)(>)
   - Less or equal(以下)(< =)
   - Greater or equal(以上)(> =)

   デフォルトの演算子は[Equal (=)]です。

5. 〈Selected Columns〉ビューをクリックして、カラムの順序を設定します。

合併テーブルの作成

合併は、同じ構造のテーブルからローを取り込むために使用します。合併は、どちらかのテーブルに存在するローを取り込みます。

▶ 合併テーブルを作成するには

1. 〈Business View Design〉で、[Table]メニューから[Union Mode(合併モード)]を選択します。

   または、ツールバーの該当するアイコンをクリックします。

   ビューが[Table Joins]を示します。

2. 合併テーブルの作成に使用するテーブルを選択します。

データ構造体

データ構造体は、あらゆるプログラミング言語や環境の主要要素です。データ構造体は、アプリケーションとテーブルやフォームの間でデータを渡すパラメータのリストです。J.D. Edwards は、次の場合にデータ構造体を使用します。

- システムが生成したデータ構造体
- ユーザーが生成したデータ構造体

システム生成のデータ構造体

システム生成のデータ構造体には、次の 2 種類があります。

- フォーム
- レポート

フォーム・データ構造体

ビジネス・ビューが関連付けられている各フォームには、デフォルトのデータ構造体があります。データ構造体は、フォーム・インターコネクトの際に他のフォームとの間でパラメータをやりとりします。このデータ構造体を保守管理するには、〈Form Design(フォーム設計)〉の[Form/Data Structure(フォーム/データ構造体)]メニュー・オプションを使用します。

レポート・データ構造体

ビジネス・ビューが関連付けられているバッチ・アプリケーションでは、データ構造体との間でパラメータをやりとりすることができます。このデータ構造体を作成して保守管理するには、〈Report Design(レポート設計)〉の[Report/Data Structure(レポート/データ構造体)]メニュー・オプションを使用します。フォーム・データ構造体とは違って、この種のデータ構造体にはデータ項目が自動設定されません。

ユーザー生成のデータ構造体

ユーザーは次の 3 種類のデータ構造体を作成することができます。

- メディア・オブジェクト・データ構造体
- 処理オプション・データ構造体
- ビジネス関数データ構造体

メディア・オブジェクト・データ構造体

メディア・オブジェクトに対するアプリケーションを使用可能にするには、データ構造体を作成し、アプリケーション・テーブルからメディア・オブジェクト・テーブルに引数を渡す必要があります。メディア・オブジェクト用のデータ構造体を処理するには、GT オブジェクト・タイプを作成するか、または既存のオブジェクト・タイプを選択して〈Object Management Workbench〉で修正します。

処理オプション・データ構造体

入力プロパティ・シートを作成するには、処理オプションを使用します。処理オプションをアプリケーションに渡すには、パラメータ・リストを使用します。処理オプションのデータ構造体テンプレートを作成したり、〈Object Management Workbench〉で既存のテンプレートを変更することができます。

ビジネス関数データ構造体

ソース言語として C 言語を使用するか、イベント・ルール・ビジネス関数を使用するかにかかわらず、どのビジネス関数の場合でも、アプリケーション間でパラメータをやりとりするには、データ構造体を定義する必要があります。その場合、DSTR オブジェクト・タイプを作成するか、または既存のオブジェクト・タイプを選択して〈Object Management Workbench〉で処理できます。また、テキスト置換メッセージのデータ構造体を作成したり、データ構造体やデータ構造体のデータ項目には用途の説明などの注記を付け加えることもできます。

参照

- テキスト置換メッセージについては『開発ツール』ガイドの「メッセージ処理」

インターコネクション用データ構造体の処理

インターコネクト用データ構造体は、〈Object Management Workbench〉ではなく〈Form Design〉ツールまたは〈Report Design〉ツールにより管理されます。アプリケーションやレポートは、このデータ構造体によりインターコネクトするフォームやセクション間で値を受け渡すことができます。

対話型アプリケーションのデフォルトのデータ構造体には、フォームで選択されたビジネス・ビューからのキーだけが入ります。デフォルトのデータ構造体に含まれていないデータ項目の値を受け渡す場合は、データ構造体にデータ項目を追加する必要があります。

レポートのデフォルトのデータ構造体は、ビジネス・ビューが関連付けられているレポート単位で作成されます。デフォルトのデータ構造体は空の構造体です。データ項目をデータ構造体に追加した場合、データ構造体とレポート自体は〈Report Design〉によって管理されます。

フォームのインターコネクト

デフォルトのデータ構造体は、フォームに対して選択されているビジネス・ビューのキーを使って作成されます。フォーム・データ構造体には〈Form Design〉からアクセスすることができます。[Form（フォーム）]メニューから[Data Structure（データ構造体）]を選択し、〈Form Data Structures（フォーム・データ構造体）〉フォームを表示します。

既存のデータ構造体に含まれる項目は、[Structure Members（構造体メンバー）]リストに表示されます。このフォームの右側には、データ構造体の修正に使用する 2 つのタブが表示されます。[Available Objects（使用可能オブジェクト）]タブには、関連付けられているビジネス・ビューのデータ項目が表示されます。[Available Objects]リストに表示されていないデータ構造体に項目を追加する場合は、その項目を[Dictionary Items（辞書項目）]リストから選択することができます。[Dictionary Items]タブでは、データ辞書からの項目にアクセスします。

▶ フォームデータ構造体を修正するには

1. 〈Form Design Aid(フォーム設計ツール)〉で、修正するデータ構造体を持つフォームに焦点を合わせて、[Form]メニューから[Data Structure]を選択します。

   〈Form Data Structure〉フォームが表示されます。

2. 関連付けられているビジネス・ビューやグリッド・カラムからオブジェクトを追加するには、[Available Objects]タブをクリックし、フォーム左側の[Structure Members]にオブジェクトをドラッグします。
3. 特定のデータ辞書項目を追加するには、以下の手順を実行します。
   a. [Dictionary Items]タブをクリックします。
   b. QBE ローで、次のいずれかのフィールドに値を入力して[Find]をクリックします。
   - Alias(エイリアス)
   - Data Item(データ項目)
   - Description(記述)
   - Product Code(システム・コード)
   - Data Type(データ・タイプ)
   - Item Size(項目サイズ)

   c. グリッドでデータ項目をクリックします。
   d. 次に、フォームの左側の[Structure Members]に適切なデータ辞書項目をドラッグします。
4. データ構造体からオブジェクトを削除するには、[Structure Members]で項目を選択して[Delete(削除)]をクリックします。
5. [OK]をクリックします。

レポート・データ構造体の変更

ビジネス・ビューが関連付けられているレポート・セクションでは、空のデフォルト・データ構造体が作成されます。このデータ構造体は、レポート・インターコネクトの際に他のレポートから値を受け取ります。必要な項目が既存のデータ構造体に含まれていなければ、項目をデータ辞書から選択して[Structure Members]リストに組み込むことができます。レポート・データ構造体を変更するには、〈Report Design〉を使用し、フォーム・データ構造体を変更する場合と同じ手順を実行します。

例:インターコネクト用データ構造体の変更

この例では、インターコネクト用データ構造体を変更する場合について説明します。

2 つの修正/検査フォームを作成した場合は、どちらにも同じビジネス・ビューが使用されます。第 2 フォームは、同じレコードからの追加カラムなど、より詳細な内容を表示します。この第 2 フォームでレコードを再度取り出さない場合は、第 2 フォームのデータ構造体を変更する必要があります。

第 2 フォームのデータ構造体には、ビジネス・ビューのキーに加えて、すべての情報（フォーム上のフィールド）とデータ項目を含める必要があります。この場合、そのフォームの［Form Options（フォームのオプション）］を［No fetch on form business view（フォーム・ビジネス・ビューでフォーム・データのフェッチなし）］に設定します。またこれは、第 2 フォームではレコードを新しい情報で更新せず、その情報を第 1 フォームに戻し、第 1 フォームでデータベースを更新する場合にも当てはまります。この場合は、［Update on form business view（フォーム・ビジネス・ビューの更新）］オプションをオフにする必要があります。

構成によっては、第 1 フォームでレコード全体を一度に取り出し、それを第 2 フォームに受け渡す場合も、第 2 フォームにレコードを再度取り出させる場合もあります。どちらの場合も、同じレコードからの異なるカラムを持つ、別々の 2 つのビジネス・ビューを使用します。第 2 フォームがあまりアクセスされないフォームである場合は、第 1 フォームのビジネス・ビューのフィールドを取り出しません。第 1 フォームでは、そのフォームに必要なカラムだけを含むビジネス・ビューを使用します。両方のフォームに必要なすべてのカラムを取り出す場合は、ネットワークをそれぞれ経由する 2 つの異なるビジネス・ビューによって生成される 2 つの別個の Select ステートメントを使用せずに、データベースから 1 つの Select ステートメントのみを実行し、すべてのデータをネットワーク上で一度に受け渡します。ワークステーションのハードウェアやメモリに制限がなければ、両方のフォームでそのレコードのすべてのカラムを取り出し、その情報をワークステーションのメモリに常駐させておいても問題はありません。

typedef の表示

データ構造体の typedef を表示することができます。typedef はクリップボードに保管されます。それをワード・プロセッサなどの別のアプリケーションに貼り付けて、保存したり表示したりすることができます。

データ構造体の作成

ビジネス関数を作成するには、データ構造体オブジェクトが必要です。データ構造体は、アプリケーションとビジネス関数の間でデータを受け渡すために使用されます。J.D. Edwards 開発ツールを使用すると、カプセル化されたデータ構造体オブジェクトを作成できます。〈Object Management Workbench〉で、オブジェクトとして個別に維持される各種のデータ構造体を作成することができます。

ビジネス関数データ構造体の作成

ビジネス関数データ構造体は、C ビジネス関数やイベント・ルール・ビジネス関数がそれらの間でデータを受け渡す際に使用されます。

▶ ビジネス関数データ構造体を作成するには

1. 〈Object Management Workbench〉で、［Add］をクリックします。
2. 〈Add J.D. Edwards Object to the Project〉で、［Data Structure］オプションを選択して［OK］をクリックします。

3. 〈Add Object〉で、次のフィールドに値を入力します。

   - オブジェクト名

     ビジネス関数データ構造体に適用される J.D. Edwards 命名規則は、「DxxxxyyyyA」です。

     D = データ構造体

     xxxx = システム・コード

     yyyy = 自動採番

     A = 関数が複数のデータ構造体を使用する場合は、A、B、C などの文字を使用して各構造体を区別します。

   - 記述
   - システム・コード
   - 製品システム・コード
   - オブジェクト使用

4. 次のオプションを選択して[OK]をクリックします。

   - Regular Data Structure(標準データ構造体)

参照

- 命名規則については、『開発スタンダード:アプリケーション設計』ガイドの「J.D. Edwards 命名規則の理解」

メディア・オブジェクト・データ構造体の作成

メディア・オブジェクトを処理するには、特別なデータ構造体オブジェクトが必要になります。メディア・オブジェクト・テーブル(F00165)には、特定のメディア・オブジェクト添付にアクセスする方法を決定する際に使用される情報が入っています。この情報は、[Object Name(オブジェクト名)]フィールドと[Media Object Key(メディア・オブジェクト・キー)]フィールド(GDTXKY)で保管されます。たとえば、〈Sales Order(受注オーダー)〉アプリケーションでは、GDTXKY は、入力された各受注オーダーのキーを保管します。F00165 テーブルには、次の情報が入っています。

- オブジェクト名(たとえば、GT4201A)
- GDTXKY(キー)(たとえば、受注オーダー番号|オーダー・タイプ|会社)
- GDTXVC(メディア・オブジェクト添付)

メディア・オブジェクト・キーには、アプリケーションの特定のレコードが保管される場所への実際のキーが含まれます。このキーは、通常、グリッドやフォームによって使用されるテーブルへのキーと同じです。メディア・オブジェクト・キーには、メディア・オブジェクト添付ごとに固有の値を設定し、システムが適切なアタッチメントを取り出すようにしておく必要があります。

► メディア・オブジェクト・データ構造体を作成するには

1. 〈Object Management Workbench〉で、[Add]をクリックします。
2. 〈Add J.D. Edwards Object to the Project〉で、[Media Object Data Structure(メディア・オブジェクト・データ構造体)]オプションを選択して[OK]をクリックします。

   〈Add Object(オブジェクトの追加)〉フォームが表示されます。

3. 〈Add Object〉で、以下のフィールドに入力して[OK]をクリックします。

   - オブジェクト名

     メディア・オブジェクト・データ構造体の名前に適用される J.D. Edwards の標準フォーマットは「GtxxxxyyA」です。

     GT = メディア・オブジェクト

     xxxx = テーブル名(文字の F を除きます)

     yy = 自動採番

     A = テーブルに複数のメディア・オブジェクトが存在する場合に、各オブジェクトを区別する文字。

   - 記述

     メディア・オブジェクトの件名に関する記述を 60 文字以内で入力します。

   - システム・コード
   - 製品システム・コード
   - オブジェクト使用

4. 〈Data Structure Design〉で、[Design Tools]タブを選択して[Data Structure Design]をクリックします。
5. 特定のデータ辞書項目を追加するには、以下の手順を実行します。

   a. [Dictionary Items]タブをクリックします。

   b. QBE ローで、次のいずれかのフィールドに値を入力して[Find]をクリックします。

      Alias(エイリアス)

      Data Item(データ項目)

      Description(記述)

      Product Code(システム・コード)

      Data Type(データ・タイプ)

      Item Size(項目サイズ)

   c. グリッドでデータ項目をクリックします。

d. フォームの左側の[Structure Members]に適切なデータ辞書項目をドラッグします。

6. 構造体メンバ名を変更するには、グリッドでローをダブルクリックして新しい名前を入力します。

データ構造体の命名規則に関する注:
ハンガリアン表記規則を使用してください。

7. [OK]をクリックします。

処理オプション・データ構造体の作成

入力プロパティ・シートを作成するには、処理オプションを使用します。

► 処理オプション・データ構造体を作成するには

1. 〈Object Management Workbench〉で、[Add]をクリックします。
2. 〈Add J.D. Edwards Object to the Project〉で、[Data Structure]オプションを選択して[OK]をクリックします。
3. 〈Add Object〉で、次のフィールドに値を入力します。
   - オブジェクト名

     処理オプション・データ構造体に適用される J.D. Edwards 命名規則は「Txxxxyyyy」です。

     T = 処理オプション・データ構造体

     xxxxyyyy = アプリケーションまたはレポートのプログラム番号

   - 記述
   - システム・コード
   - 製品システム・コード
   - オブジェクト使用
4. 次のオプションを選択し、[OK]をクリックします。
   - Processing Option Template(処理オプション・テンプレート)

データ構造体の定義

データ構造体を作成した後は、データ構造体に含まれるデータ・オブジェクトを定義する必要があります。既存のデータ構造体は、新しいデータ・オブジェクトを追加するか、またはオブジェクトを削除して修正できます。

参照

- 処理オプションおよびそれらのデータ構造体とテンプレートの処理については、『開発ツール』ガイドの「処理オプション」

データ構造体の設計

OMW でデータ構造体を作成したら、以下の操作を行ってデータ構造体の機能を指定します。

▶ データ構造体を設計するには

1. 〈Object Management Workbench〉で、処理するデータ構造体をチェックアウトします。
2. データ構造体を選択して、中央カラムの[Design]ボタンをクリックします。
3. 〈Data Structure Design〉で、[Design Tools]タブを選択して[Data Structure Design]をクリックします。
   a. 特定のデータ辞書項目を追加するには、以下の手順を実行します。
      i. [Dictionary Items]タブをクリックします。
      ii. QBE ローで、次のいずれかのフィールドに値を入力して[Find]をクリックします。

          Alias(エイリアス)

          Data Item(データ項目)

          Description(記述)

          Product Code(システム・コード)

          Data Type(データ・タイプ)

          Item Size(項目サイズ)

   c. グリッドでデータ項目をクリックします。

   d. フォームの左側の[Structure Members]にデータ辞書項目をドラッグします。

4. 構造体メンバ名を変更するには、[Structure Members]領域のグリッドでローをダブルクリックして新しい名前を入力します。

   データ構造体の命名規則に関する注:
   ハンガリアン表記規則を使用してください。

   使用するデータ辞書項目がわからない場合は、[Data Dictionary(データ辞書)]をクリックしてデータ辞書項目を検索し、選択したデータ項目の詳細を確認してください。

   データ構造体のデータ項目に関する詳細情報を表示するには、データ項目を選択して[Data Dictionary Detail(データ辞書詳細)]をクリックします。

5. [Structure Members]領域で、[Required Member/Req(必須メンバ/必須)]オプションをブランクにするか、または次のいずれかを選択して、必須状況を指定することができます。
   - X – このフィールドがオプションのフィールドであることを示します。
   - チェックマーク – このフィールドが必須フィールドであることを示します。

6. データ・フローの方向を選択します。[Input/Output/IO（入出力/IO）]オプションの矢印をクリックし、それを適切な方向とタイプに変更します。
7. データ構造体からオブジェクトを削除するには、[Structure Members]で項目を選択し、[Delete]をクリックします。
8. データ構造体の添付を定義するには、[Data Structure Attachments]をクリックします。
9. データ構造体項目の添付を定義するには、[Data Structure Item Attachments]をクリックします。
10. 相互参照ユーティリティを起動するには、[Cross Reference（相互参照）]をクリックします。
11. 作業が完了したら[OK]をクリックします。

typedef の作成

データ構造体を設計した後、typedef を作成します。typedef は、ビジネス関数で使用されるデータ構造体の定義を表す C コードです。作成した typedef はクリップボードに保管されるので、それを.h ファイルの適切なセクションに貼り付けることができます。この方法では.h ファイルにコードを入力する必要がないため、時間の節約になります。

▶ typedef を作成するには

1. 〈Object Librarian Data Structure Design〉で、[Design Tools]タブをクリックします。
2. [Create a type definition（typedef の作成）]ボタンをクリックします。

   typedef はクリップボードに保管されます。.h ファイルの適切なセクションに貼り付けることができます。

相互参照機能

相互参照機能を使用すると、特定の種類のオブジェクトが使用される箇所と使用方法を確認できます。また、オブジェクトとそのコンポーネントとの関係を表示することもできます。たとえば、次の処理を実行できます。

- ビジネス関数が使用されている各インスタンスを識別する。
- アプリケーションのフォーム・リストを表示する。
- ビジネス・ビューやフォーム・インターコネクトのすべてのフィールドを表示する。
- 特定のフィールド、イベント・ルール、またはコントロールが使用されているすべてのアプリケーションを相互参照する。

相互参照関係は、オブジェクトまたはコンポーネントの変更時にリビルドできます。

オブジェクトの検索

オブジェクトは、検索タイプとオブジェクト名で検索できます。

▶ オブジェクトを検索するには

〈Cross Application Development Tools〉メニュー(GH902)から〈Cross Reference Facility(相互参照機能)〉を選択します。

次のタブのいずれかをクリックします。

- Data Items(データ項目)
- Interactive Applications(対話型アプリケーション)
- Batch Applications(バッチ・アプリケーション)(相互参照でのバッチ・アプリケーション・オブジェクト・タイプは UBE)
- Business Functions(ビジネス関数)
- Business Views(ビジネス・ビュー)
- Data Structures(データ構造体)
- Tables(テーブル)
- Forms(フォーム)

データ項目の検索

特定のデータ項目が使用されている場所を確認できます。以下を検索できます。

- データ項目を使用するフォーム
- データ項目を使用している UBE
- データ項目を変数として使用している UBE イベント・ルール
- データ項目を変数として使用しているアプリケーション
- データ項目を変数として使用しているイベント・ルール・ビジネス関数
- スマート・フィールドのデータ項目によって呼び出されている関数
- データ項目によって呼び出されている編集ルール関数
- データ項目によって呼び出されている表示ルール関数
- データ項目によって使用されている検索フォーム
- データ項目を使用している処理オプション
- データ項目を使用している汎用テキスト・データ構造体
- データ項目を使用しているビジネス関数データ構造体
- データ項目を使用しているすべてのデータ構造体
- データ項目を使用しているテーブル
- データ項目を使用しているインデックス
- データ項目を使用しているビジネス・ビュー
- データ項目を使用しているテーブル・イベント・ルール

対話型アプリケーションの検索

対話型アプリケーションに関するさまざまな情報を確認できます。以下を検索できます。

- あるアプリケーションを呼び出しているアプリケーション
- ビジネス関数を呼び出しているアプリケーション
- アプリケーションによって使用されているデータ構造体
- アプリケーションによって使用されているテーブル
- アプリケーション用のフォーム
- アプリケーション用のデータ項目
- アプリケーションで変数として使用されているデータ項目
- アプリケーション用の処理オプション
- アプリケーション用のビジネス・ビュー

バッチ・アプリケーションの検索

通常は UBE と呼ばれるバッチ・アプリケーションに関する情報とそれがどのように使用されているかを確認できます。以下を検索できます。

- ビジネス関数を呼び出している UBE
- UBE によって使用されているテーブル
- UBE によって使用されているデータ構造体
- ある UBE によって呼び出されている UBE
- ある UBE を呼び出している UBE
- UBE 用の処理オプション
- UBE 用のビジネス・ビュー
- UBE によって使用されているデータ項目
- UBE で変数として使用されているデータ項目

ビジネス関数の検索

ビジネス関数に関するさまざまな情報とそれらがどのように使用されているかを確認できます。以下を検索できます。

- ビジネス関数が使用されている場所
- イベント・ルール・ビジネス関数によって使用されているデータ構造体
- アプリケーションによって呼び出されているビジネス関数
- UBE によって呼び出されているビジネス関数
- ビジネス関数によって使用されているテーブル
- あるビジネス関数によって呼び出されているビジネス関数
- あるビジネス関数を呼び出しているビジネス関数

- イベント・ルール・ビジネス関数で変数として使用されているデータ項目
- イベント・ルール・ビジネス関数によって使用されているテーブル
- イベント・ルール・ビジネス関数によって呼び出されているビジネス関数
- カラム見出し関数を呼び出しているスマート・フィールドのデータ項目
- 関数が使用されている場所
- 値ビジネス関数を呼び出しているスマート・フィールドのデータ項目
- 編集ルール関数を呼び出しているデータ項目
- 表示ルール関数を呼び出しているデータ項目
- ソース・ファイル名とヘッダー・ファイル名に使用されている文字列
- ビジネス関数を使用しているテーブル・イベント・ルール

ビジネス・ビューの検索

ビジネス・ビューに関する情報とそれらがどのように使用されているかを確認できます。以下を検索できます。

- ビジネス・ビューを使用しているアプリケーション
- ビジネス・ビューを使用している UBE
- ビジネス・ビューを使用しているフォーム
- ビジネス・ビュー用のデータ項目
- ビジネス・ビュー用のテーブル

データ構造体の検索

データ構造体に関するさまざまな情報とそれらがどのように使用されているかを確認できます。以下を検索できます。

- データ構造体を使用しているアプリケーション
- データ構造体を使用しているビジネス関数
- データ構造体を使用している UBE
- 処理オプション用のデータ項目
- 汎用テキスト・データ構造体用のデータ項目
- ビジネス関数データ構造体用のデータ項目
- すべてのデータ構造体用のデータ項目
- データ構造体を使用しているテーブル・イベント・ルール

テーブルの検索

テーブルに関するさまざまな情報とそれらがどのように使用されているかを確認できます。以下を検索できます。

- テーブルを使用しているビジネス関数
- テーブルを使用しているアプリケーション
- テーブルを使用している UBE
- テーブルを使用しているイベント・ルール・ビジネス関数
- テーブルを使用しているフォーム
- テーブル用のデータ項目
- テーブル用のインデックス
- テーブルを使用しているビジネス・ビュー
- テーブル・イベント・ルールによって使用されているデータ項目
- テーブル・イベント・ルールによって使用されているビジネス関数
- テーブル・イベント・ルールによって使用されているデータ構造体
- テーブル・イベント・ルールによって使用されているテーブル

フォームの検索

フォームに関するさまざまな情報とそれらがどのように使用されているかを確認できます。以下を検索できます。

- アプリケーションによって呼び出されているフォーム
- フォーム用のテーブル
- フォーム用のデータ項目
- フォーム用のビジネス・ビュー
- 検索フォームを使用しているデータ項目

イベント・ルールの検索

イベント・ルールを検索すると、使用可能な既存のイベントを確認できます。

► イベント・ルールを検索するには

1. 〈Cross Reference Facility〉検索フォームで、[Form]メニューから[Event Rule(イベント・ルール)]を選択します。
2. 〈ER Search(イベント・ルール検索)〉で、次のフィールドに値を入力して[Find]をクリックします。
   - オブジェクト名
   - フォーム名
   - ﾃﾞｰﾀﾌｨｰﾙﾄﾞ名

フィールド記述

記述	用語解説
オブジェクト名	システム・オブジェクトを識別する名前。J.D. Edwards ERP アーキテクチャはオブジェクト指向です。ソフトウェアの個々のオブジェクトはすべてのアプリケーションのビルディング・ブロックとなっており、複数のアプリケーションでオブジェクトを再使用できます。各オブジェクトは、オブジェクト・ライブラリアンで管理されます。 オブジェクトの例は次のとおりです。 o バッチ・アプリケーション(レポートなど) o 対話型アプリケーション o ビジネス・ビュー o ビジネス関数 o ビジネス関数データ構造体 o イベント・ルール o メディア・オブジェクト・データ構造体
フォーム名	画面、レポート、またはデータベースファイルのレコード形式に付けられた名前。
ﾃﾞｰﾀﾌｨｰﾙﾄﾞ名	データ基準ファイル、ビデオ画面、またはレポートのいずれかで使用しているデータフィールド名。このフィールド名は、データ記述スペックで参照されている 6 文字のレポート作成プログラム名です。

フィールド関係の表示

〈Field Relationships（フィールド関係）〉フォームは、次の相互参照検索タイプにのみ有効です。

DA アプリケーションによって使用されるデータ項目

FA アプリケーション用のフォーム

FI データ項目を使用するフォーム

SA アプリケーション用のデータ構造体

上記のインスタンスでは、〈Field Relationships〉フォームに次のようなフィールドのコントロール・タイプが表示されます。

- ビジネス・ビュー・カラムを表す BC
- フォーム・インターコネクト・コントロールを表す FI
- グリッド・コントロールを表す GC
- フォーム・コントロールを表す FC

▶ フィールド関係を表示するには

選択した〈Cross Reference〉検索フォームで、[Form]メニューから[Field Relationships（フィールド関係）]を選択します。

1. 〈Field Relationships〉で、必要に応じて次のフィールドに値を入力します。
   - オブジェクト
   - フォーム名
   - フィールド
   - コントロール

     [Control（コントロール）]フィールドには[Search]ボタンがありません。次のコントロール・タイプから指定できます。

     - BC – Business View Field（ビジネス・ビュー・フィールド）
     - PO – Processing Option（処理オプション）
     - FI – Form Interconnect（フォーム・インターコネクト）
     - VAR – Variable（変数）
     - FC – Form Control（フォーム・コントロール）
     - FCW – Form Control Work field（フォーム・コントロール・ワーク・フィールド）
     - GC – Grid Column（グリッド・カラム）
     - GCW –（グリッド・カラム・ワーク・フィールド）

2. [Find]をクリックします。

   特定のアプリケーションまたはフォームのすべてのコントロールが、グリッドに表示されます。

相互参照情報のリビルド(再作成)

オブジェクトを修正すると、相互参照情報がシステムのオブジェクトの情報と一致しなくなることがあります。相互参照ファイルは、オブジェクトの変更時に自動的にリビルドされないため、照会時にリビルドする必要があります。また、相互参照の作成スケジュールを定期的に設定して、相互参照情報が定期的に更新されるようにすることもできます。

〈Cross Reference〉ユーティリティの各相互参照フォームには、グリッドの右端のカラムにレコードの作成日が表示されます。情報が古くなっている場合は、任意の相互参照フォームから[Rebuild(再作成)]オプションを使用します。

相互参照の作成では、ローカル・スペックではなくリレーショナル・データベース・テーブルを使用します。

▶ 相互参照情報をリビルドするには

1. 〈Cross Reference〉で、[Form]メニューから[Rebuild Relationships(リビルド関係)]を選択します。
2. リビルドするオブジェクトを選択します。

   注:
   リビルド処理には数分かかることがあります。

イベント・ルール

「イベント・ルール」では、イベント・ルールとロジックの使用方法について説明します。また、BrowsER の使用方法についても説明します。

イベント・ルール設計

〈Event Rules Design（イベント・ルール設計）〉を使用して、アプリケーションのビジネス・ロジックを作成します。たとえば、次の処理を実行するイベント・ルールを作成できます。

- 算術演算を実行する。
- フォームのフィールドから他のフォームのフィールドにデータを渡す。
- データが挿入されるグリッド・ロー数をカウントする。
- 2 つのフォームをインターコネクト（相互接続）する。
- システム関数を使用してコントロールを表示/非表示にする。
- IF/WHILE 条件と ELSE 条件を評価する。
- フィールドに値や式を代入する。
- 変数またはプログラマ定義のフィールドを実行時に作成する。
- 対話型アプリケーションの終了時にバッチ処理を実行する。
- テーブルの入出力を処理し、データを検証し、レコードを抽出する。

はじめる前に

- ❑ 1 つ以上のフォームをもつアプリケーションを作成します。
- ❑ データベース項目と辞書項目の違いを理解します。
- ❑ コントロール、イベント、イベント・ルールの関係を理解します。
- ❑ アプリケーションで使用する各フォームの用途を確認します。
- ❑ 次の 4 点を考えておきます。
  - どんなロジックが必要か？
  - どのコントロールに対してロジックを作成するか？
  - どのイベントに対してそのロジックが発生するか？
  - どのランタイム構造体が影響を受けるか？

コントロールについて

コントロールとは、フォームに表示される再利用可能なオブジェクトのことです。たとえば、プッシュ・ボタン、編集フィールド、グリッドなどがあります。フォーム自体もコントロールと見なされます。

コントロールは、単純なものでも複雑なものでもかまいません。単純なコントロールは、ロジックを関連付けることのできる少数のイベントをもっています。複雑なコントロールは、ロジックを関連付けることのできる多数のイベントをもっています。

イベントについて

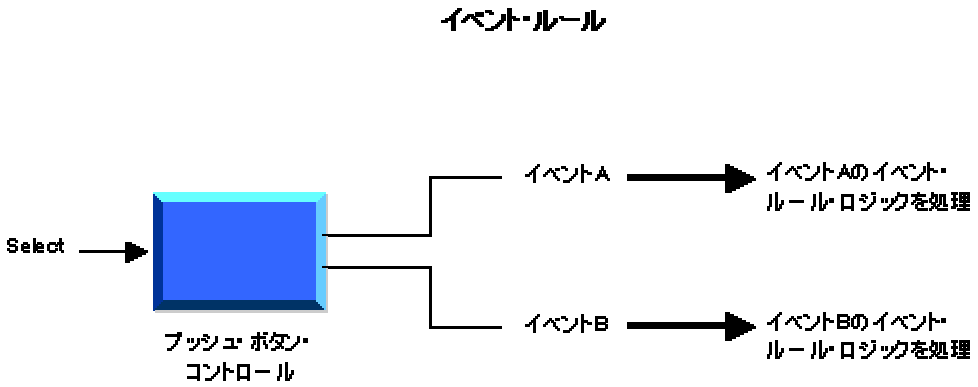
イベントとは、フォームに情報を入力したり、Tab キーを使用してフィールドから出るなど、フォーム上で発生するアクティビティのことです。イベントは、ユーザーやアプリケーションによって起動できます。1 つのコントロールで複数のイベントが実行される場合があります。また、Last Grid Record Read など、一定のアクションが発生したときに実行されるイベントもあります。

フォーム処理について

フォーム処理とは、各フォームに関連付けられたビジネス・ロジックを指します。デフォルトでは、各種 J.D. Edwards フォームはさまざまなイベントを自動的に処理します。追加のロジックを指定するには、〈Event Rules Design〉を使用します。フォーム処理は、フォームの初期化やフィールド値の変更など、特定のイベントの発生に応じて異なります。

イベント・ルールについて

イベント・ルールとは、作成してイベントに添付できる論理ステートメントです。J.D. Edwards ソフトウェアでは、ビジネス関数イベント・ルールと埋込みイベント・ルールという 2 種類のイベント・ルールを使用します。これらのルールは、実行時にイベントが発生すると実行されます。この関係を下に示します。



1 つのイベントに複数のイベント・ルールを関連付けることができます。イベント・ルールには、次のようなタイプがあります。

- IF/ELSE/END IF などの条件文
- WHILE ループ文
- ステートメントへの代入
- カプセル化関数の呼出し
- フォームまたはレポートのインターコネクション
- システム関数の呼出し
- テーブル入出力操作

イベント・ルール・ビジネス関数

イベント・ルール・ビジネス関数は、C プログラミングではなく、〈Event Rules Design〉を使用して作成する再利用可能なカプセル化ビジネス・ロジックです。この種のビジネス関数は、オブジェクトとして保管されコンパイルされます。イベント・ルール・ビジネス関数は、NER と呼ぶこともあります。

埋込みイベント・ルール

埋込みイベント・ルールは、特定のテーブル、対話型アプリケーション、またはバッチ・アプリケーションに固有のルールで、再利用できません。たとえば、フォームからフォームを呼び出したり、処理オプションの値に基づいてフィールドを非表示にしたり、ビジネス関数を呼び出したりするものがあります。埋込みイベント・ルールは、アプリケーションのイベント・ルール（対話型またはバッチ）、またはテーブル・イベント・ルールに含めることができます。これらは、コンパイルされるものとされないものがあります。

アプリケーション・イベント・ルール

特定のアプリケーションに固有のビジネス・ロジックを追加することができます。対話型アプリケーションでは〈Form Design（フォーム設計）〉を介してイベント・ルールを関連付けますが、バッチ・イベント・ルールには〈Report Design（レポート設計）〉を使用します。

テーブル・イベント・ルール

データベース・トリガーを作成したり、テーブル設計イベント・ルールを使用してテーブルに関連付けるルールを作成することができます。テーブルに関連付けられているロジックは、アプリケーションがそのデータベース・イベントを起動するたびに実行されます。たとえば、参照整合性を維持するために、親が削除されたときに子をすべて削除するマスター・テーブルにルールを関連付けることもできます。そのテーブルから情報を削除するアプリケーションにはそのロジックがテーブルに存在するので、親/子ロジックを埋め込んでおく必要がありません。

参照

- 個々のイベントについては、オンライン・ヘルプの「Published API」

イベント・ルール・ボタンについて

〈Event Rules Design〉フォームには、各種のビジネス・ロジックを生成できるように、次の大きなボタンが表示されます。

ビジネス関数 既存のビジネス関数を関連付けます。

システム関数 既存の J.D. Edwards システム関数を関連付けます。

IF/While IF/WHILE 条件文を作成します。

割当て 代入または複雑な式を作成します。

レポート・インターコネクト バッチ・アプリケーションやレポートへの接続を設定します。

フォーム・インターコネクト フォーム・インターコネクトを設定します。

オフの場合 IF から ENDIF までの範囲でのみ有効な ELSE 節を挿入します。

変数 アプリケーション固有の目的に合ったデータ辞書の特性が割り当てられていても、データ辞書には常駐しないプログラマ定義のフィールドを作成します。

テーブル I/O データベースへのアクセスをイベント・ルールでサポートできるようにします。テーブル入出力、データの検査、レコードの取出しを実行します。

フォーム・タイプに基づくイベント

各フォーム・タイプには、通常、次の表に示すようにそれぞれのフォーム・タイプで使用する特有のイベント・ルールがあります。

フォーム・タイプ	イベント・ルール
検索/表示	フォーム・レベルとグリッド・レベル
修正/検査	フォーム・レベルのみ
見出し詳細と見出しなし詳細	フォーム・レベルとグリッド・レベル

ランタイム処理

ランタイム処理とは、各種のイベント（フォームの初期化、ボタンのクリック、Tab キーによるフィールド間の移動など）とそれぞれに関連付けられているイベント・ルール・ロジックを評価することを意味します。イベント・ルールのロジックはイベントに関連付けられ、イベントはコントロールに関連付けられます。

〈Form Design〉には、各種のフォーム・タイプが用意されており、それぞれに事前定義済みのフィールドと固有の機能が含まれています。たとえば、検索/表示フォームには、適切な機能が関連付けられた[Find]メニュー・オプションやツールバー・ボタンが自動的に組み込まれます。フィルタ用フィールドや QBE フィールドに検索テキストを入力して、ツールバーの[Find]ボタンをクリックすると、ランタイム・エンジンが、レコードを取り出すロジックを処理します。

不要なイベント・ルールの生成を避けるために、さまざまなフィールド・タイプと、各フォーム・タイプを特徴付ける関連機能について理解してください。また、既存のイベント・ルールをカスタマイズすることもできます。

ランタイム・データ構造体

ランタイム・データ構造体は、データを読取り処理してデータベースに書き込むときに、データをメモリに保持する構造体、またはメモリ・ブロックです。実行時に各フォームで何が発生するか、所定のイベント・ポイントでランタイム構造体に何が格納されるかを知っておく必要があります。

ランタイム・システムは、ランタイム・データ構造体を動的に作成します。たとえば、フォームに非表示のコントロールが存在する場合、それらのコントロールは、フォームに表示されなくてもメモリが割り当てられています。フォームの処理オプション値を受け渡すと、処理オプション値を格納するメモリが割り当てられます。

使用可能オブジェクトとランタイム・データ構造体

使用可能オブジェクトは、データ・ソースの特性を示す 2 文字の英字コードで表され、オブジェクト・データが実行時に対話型アプリケーションでどのように使用されるかを決定します。使用可能オブジェクトは、フォームに対し、ランタイム・データ構造体を形成します。

ランタイム処理では、データがメモリの内部データ構造体に格納されます。データ構造体によって、データを同じフォーム上の別のフィールドやフォーム間で受け渡すことができます。データ構造体のフィールドには、実行時にデータを一時的に格納するものがあります。データは不要になった時点で削除され、別のレコードを処理できるようになります。

次の表に、使用可能なオブジェクトを示します。

BC ビジネス・ビューのカラム。フォーム・ビューのビジネス・ビュー・カラムも、グリッド・ビューのビジネス・ビュー・カラムも、このリストに表示されます。これらのカラムには、フェッチの実行時にデータベースから取得された値が入力されます。これらの値は、追加/更新時にデータベースに書き込まれます。

GC グリッドのカラム。値が参照するローは、GC にアクセスするイベントによって異なります。フェッチ・サイクルの最中で、ローが取り出されます。これは通常、選択されたローです。グリッド・カラム・オブジェクトは、値ではなくグリッドの特定の物理カラムを指す場合もあります。Set Grid Line Font（グリッド・ライン・フォントのセット）は、その一例です。

GB グリッド・バッファ。このバッファは 1 ローのデータ・スペースで、データベースから読み取られ、グリッドに書き込まれる行に依存しません。グリッド・バッファにより、グリッドの現在の状態に影響を与えずに、挿入または更新する行のカラム・データを操作できます。グリッド・バッファは、使用可能オブジェクトの GB（グリッド・バッファ・カラム）を介してアクセスします。GB は、イベント・ルールの使用可能オブジェクトのリストで、GC（グリッド・カラム）の後に表示されます。各グリッドには、各 GB カラムのインスタンスが 1 つのみ含まれます。グリッド・バッファは、特定の行に関連付けられません。

FC フォーム上のコントロール。コントロールがデータベース項目の場合、このフィールドは BC オブジェクトに対応します。また、コントロールがフィルタでない場合、FC オブジェクトは BC オブジェクトと同じ値を表し、これらの一方を変更すると両方が変更されます。

FI フォーム・インターコネクトを通じて渡される値。このオブジェクトは、フォームに送られる値を読み取ったり、送り返される値を設定するためにアクセスします。これらのオブジェクトは、フォーム・データ構造体の要素に対応します。

PO 処理オプションから渡される値。これらの値は起動時にアプリケーションに渡され、そのアプリケーションのすべてのフォームからアクセスできます。処理オプションは、ユーザーが入力することもできれば、アプリケーションの特定のバージョンで設定することもできます。

QC グリッドの QBE からのカラム。これらのオブジェクトは、グリッド上の任意の QBE の値を表します。これらにはワイルドカードを使用できますが、比較演算時は使用できません。また、これらのオブジェクトへの代入式にもワイルドカードを使用することはできますが、比較演算子を使用することはできません。比較を設定するには、システム関数を使用できます。QC オブジェクトは、更新グリッドでのデータ選択に影響しません。

HC ハイパー・コントロール項目。これらのオブジェクトは、フォームのハイパー・コントロールのハイパー・コントロール項目を表します。これらを使用すると、フォーム・メニューとツールバーの項目を有効/無効にしたり、イベント・ルールによって起動することができます。

VA イベント・ルール変数。これらのオブジェクトは、開発者がイベント・ルールに設定した変数を表します。システムによって処理されることはありません。

SV システム変数。これらのオブジェクトは、イベント・ルールにアクセスできる環境変数を表します。

SL システム値。これらのオブジェクトは、イベント・ルールにアクセスできる固定システム値を表します。

TP タブ・ページのオブジェクト

TK テーブル・イベント・ルールを含むテーブルのカラム

CO　エラーのリターン・コードなどの定数

TV　テキスト変数

RC　レポート定数(UBE)

RV　レポート変数(UBE)

IC　入力カラム(テーブル変換)

OC　出力カラム(テーブル変換)

FC がデータベース項目に関連付けられている場合、BC と FC は同じ内部構造体を共有します。フィルタ用フィールドは例外です。

イベント・ルールの操作

フィルタ・フォーム・コントロールを追加せずに、検索を特定の状況に合わせて自動調整するには、イベント・ルールを使用して QBE カラムに値を割り当てることができます。フィルタと異なり、イベント・ルールを使用すると、設計時ではなく実行時に比較演算子を変更できます。

使用可能オブジェクトの処理

使用可能なオブジェクトの値がイベント・ルールの処理中に変更されると、次の処理が行われます。

- ER 行が処理された直後、新しい値がビジネス・ビューのデータ、グリッド・データ、フォーム・コントロール・データ、フォーム・インターコネクト・データ、または処理オプション・データにコピーされます(内部のランタイム構造体)。
- フォーム・コントロール・データとグリッド・データが適切なフォーム・コントロールまたはグリッド・セル(外部画面)にコピーされます。

データベース項目に関連付けられているフォーム・コントロール・データは、ランタイム・データ構造体をビジネス・ビュー・データと共有します。ただし、フィルタ・フィールドは例外です。この共有状況は、次のことを意味します。

- FC データと BC データは常に一致します。
- FC ランタイム・データが変更されると、BC の参照がいずれも同じ値を反映します。
- BC 値が変更されると、FC ランタイム値も変更されます。この FC の変更は、FC ランタイム値が画面に移動するまでフォームに反映されません。
- Control is Exited の処理の際にフォーム・コントロールに入力されている値は、BC と FC によって共有されるランタイム・データ構造体に取り込まれます。

コントロール終了時（Control is Exited）の処理

Control is Exited イベントの処理は次のとおりです。

- コントロールの値が、内部のランタイム構造体に保存されます。
- Control is Exited イベントが処理されます。
- 前回のコントロール終了時以降に値が変更されている場合、次の処理が行われます。
    - Control is Exited and Changed – Inline（コントロールが終了し、変更された – インライン）イベントが処理されます。
    - 非同期処理が開始されます（次の 3 つのステップはスレッド上で実行されます）。
    - Control is Exited and Changed – Async（コントロールが終了し、変更された – 非同期）イベントが処理されます。
    - データ辞書が検証されます。
    - フォーム・コントロール・データがフォーマットされ、コントロールにコピーされます。

修正/検査フォームとトランザクション・フォームでは、このロジックは［OK］ボタンの各コントロールに対しても実行されます。

グリッド処理

グリッドでの選択や順序設定を操作するために、システム関数を使用することができます。これらの関数により、ランタイム・エンジンがグリッドに自動入力するときに使用するデータベース API 構造体と選択および順序設定用の構造体に直接アクセスできます。このために次のシステム関数を使用できます。

Set Selection（選択のセット）	選択構造体に 1 つの要素を作成します。
Set Lower Limit（下限のセット）	選択構造体に 1 つの要素を作成します。このシステム関数は、Set Selection と同じです。
Clear Selection（選択のクリア）	システム関数で定義された選択情報をすべて解除します。
Set Selection Append Flag	システム関数定義情報を QBE およびフィルタ用フィールド情報に追加するか、または前者を後者で置き換えるかを指定します。
Set Sequencing（順序のセット）	ソート構造体に 1 つの要素を作成します。
c	システム関数で定義された順序設定情報をすべて解除します。

見出しなし詳細フォームに「等しい」フィルタが少なくとも 1 つ存在するか、または 1 つの非フィルタ・データベース・フォーム・コントロールが存在する場合、フォームのグリッド・レコードは、それらが変更された場合にのみ更新されます。

カスタム・グリッドのページ単位処理では、Get Custom Grid Row イベントと Continue Custom Data Fetch などのシステム関数を使用することもできます。この処理により、一度に 1 ページ分のデータのみを取り出して、レコード数を制限することができます。これは、特に膨大な数のレコードを取り出すことがあるグリッドに対して有効です。

グリッド・カラムには、先行入力機能があります。フィールドに文字を入力すると、それに一致する文字列が履歴リストで検索されます。一致する文字列が存在する場合は、ハイライトされたテキストでフィールドに表示されます。この機能は、特にデータ入力作業で役立ち、入力操作の手間を軽減できます。

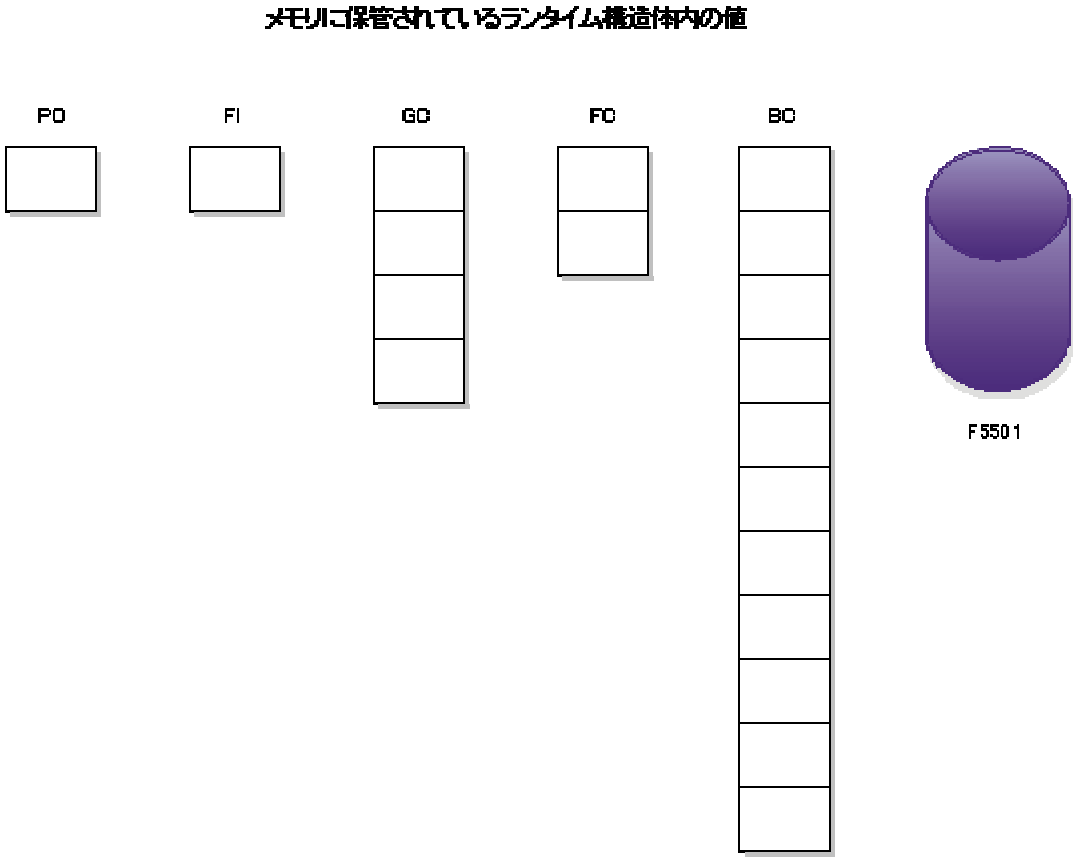
フォーム・フロー

フォーム・タイプには、それぞれに固有のプロパティと固有のイベント・フローがあります。処理は、イベント・ルール・ロジックを追加する場合も追加しない場合も行われます。ただしエンジンは、Dialog Is Initialized など特定のイベントで一時停止するため、ロジックを追加したり値を操作することができます。イベント・ルール・ロジックとユーザーの操作により、イベントがどのように処理されるかが決まります。

フォーム・タイプによってその順序と数は異なりますが、それぞれのフォームで発生するイベントがあります。たとえば Dialog Is Initialized を使用して、全タイプのフォームにロジックを追加できます。また Post Dialog is Initialized を使用して、検索/表示フォームではなく修正/検査フォームのロジックを追加することもできます。更新可能なグリッドが存在するフォームでは、Dialog is Initialized イベントと Post Dialog is Initialized イベントは、グリッドが存在しないフォームで実行する場合と多少異なります。更新可能なグリッドが存在するフォームでは、これら 2 つのイベント間でデータベースを検索して読み込みます。

〈Find/Browse〉フォームの典型的なイベント・フロー

次の例では、ランタイム構造体の値がメモリにどのように格納されるかを、それらのフォーム上での表示と対比させて説明します。この例では、メニューから直接呼び出した〈Find/Browse〉フォームを使用します。



ランタイム・エンジンは、一定の順序でイベントを処理します。ここでは、検索/表示フォームの典型的なイベントと、それらが処理される順序について説明します。この処理フローは、特定のユーザー操作と使用するイベント・ルール・ロジックによって異なる場合があります。

Pre Dialog Is Initialized

Dialog is Initialized イベントが処理されフォームが表示される前に、次のステップが実行されます。

- 次のようにランタイム構造体を初期化する（またはメモリがクリアされる）。

  BC = 0

  FC = 0

  GC = 0

  FI = 呼出し側フォームから渡された値（存在する場合）

  PO = 処理オプションから渡された値

- フォーム・コントロールを初期化する。
- エラー処理を初期化する。
- Static テキストを初期化する。
- ヘルプを初期化する。
- ツールバーを作成する。
- フォーム・インターコネクト・データを対応する BC カラムとフィルタ用フィールド（存在する場合）にロードする。
- スレッド処理を初期化する。
- フィルタ・コントロールを使用して、SQL SELECT ステートメントの WHERE 句を作成する。

フォームへのフォーム・インターコネクトが存在する場合は、FI 構造体に関する情報がコンピュータのメモリに格納される。この FI 値は、BC ランタイム構造体にコピーされる。

Dialog Is Initialized

エンジンが一時停止して、イベントが処理されます。Dialog Is Initialized イベントが関連付けられているすべてのイベント・ルール・ロジックが処理されます。エンジンが一時停止したとき、ランタイム構造体には次の値が格納されています。

BC = 渡されたすべての FI 値

FC = 渡されたすべての FI 値

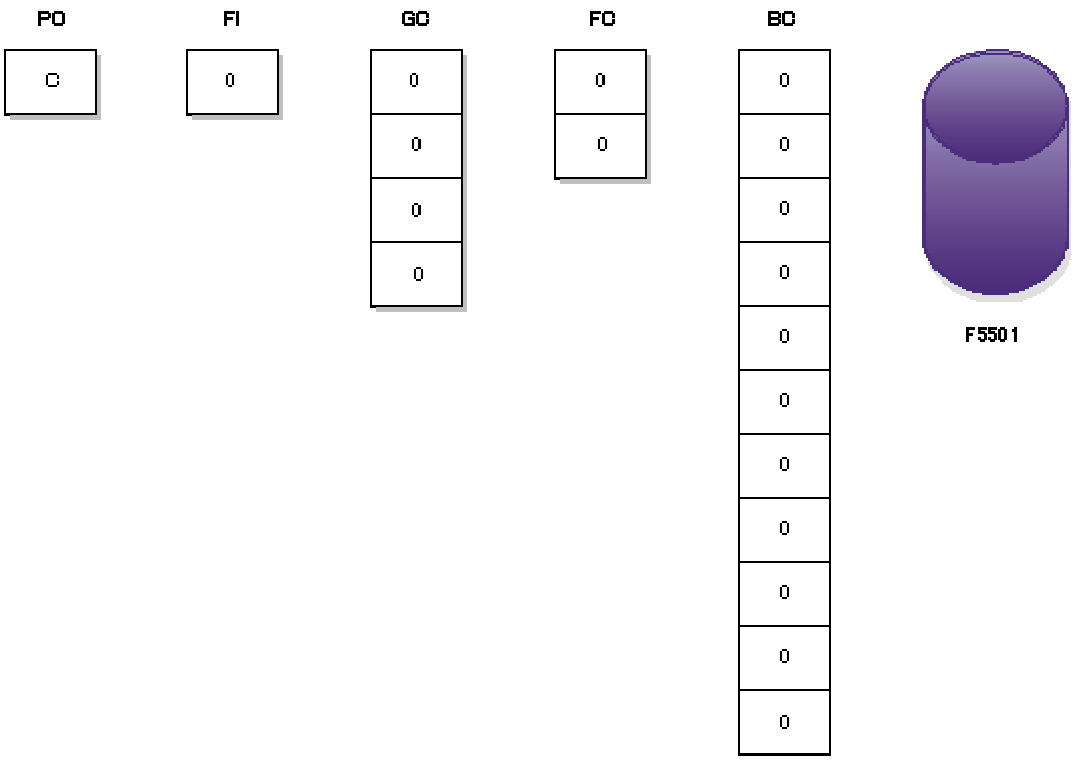
GC = 0

FI = 呼出し側フォームから渡された値（存在する場合）

PO = 処理オプションから渡された値

次の図に、Dialog Is Initialized を実行する直前にランタイム構造体に格納されている情報を示します。

Dialog Is Initializedが実行される前のランタイム構造体の内容



Dialog is Initialized イベントの一般的な用途は、次のとおりです。

- コントロールをフォームで非表示/表示する。
- コントロールを入力不可にする。

Dialog Is Initialized が処理されると、フォームが表示されます。

Post Dialog Is Initialized

エンジンが再び一時停止して、イベントが処理されます。Post Dialog Is Initialized の実行時に処理されるロジックを追加できます。エンジンが一時停止したとき、ランタイム構造体には次の値が格納されています。

BC = 0（または受け渡し済みの値）

FC = 0（または受け渡し済みの値）

GC = 0（または受け渡し済みの値）

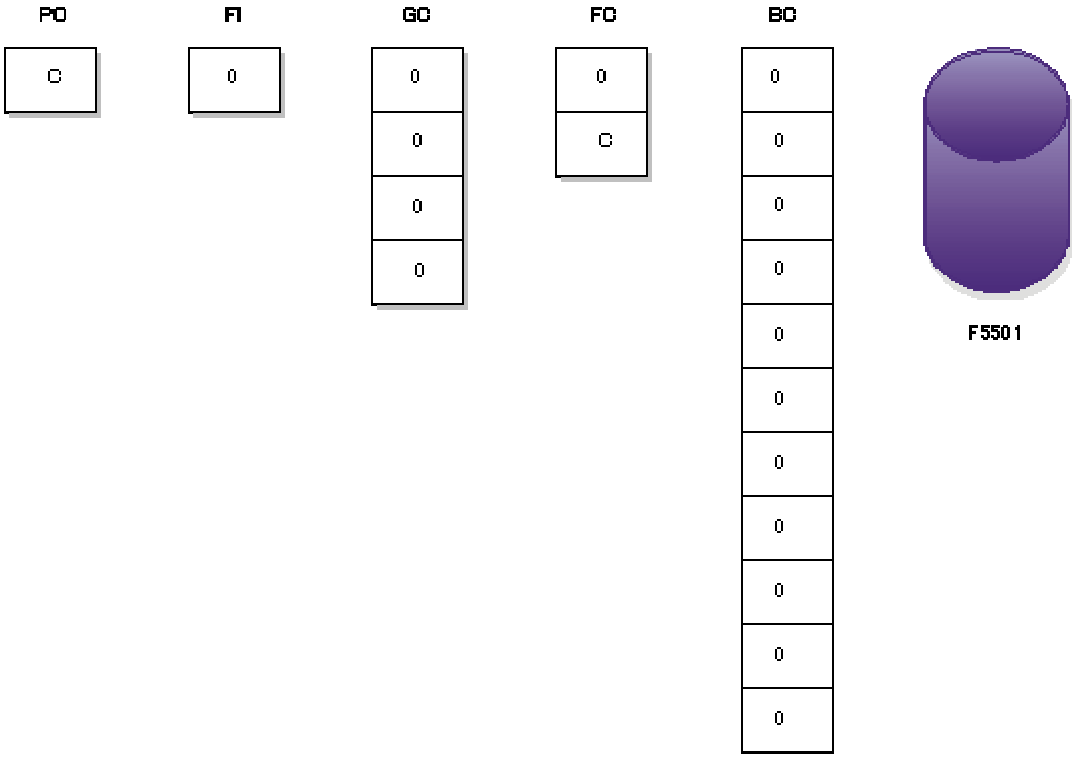
FI = 呼出し側フォームから渡された値（存在する場合）

PO = 処理オプションから渡された値

この時点ではまだレコードは取り出されていません。ロジックを追加したりランタイム構造体の値を操作できるように、エンジンは Fetch 処理の直前に一時停止します。この時点で、フィルタ用フィールドからの割当て処理を実行することもできます。

次の図に、Form Record is Fetched を実行する直前にランタイム構造体に格納されている情報を示します。

Form Record is Fetchedが実行される前のランタイム構造体の内容



Post Dialog is Initialized イベントの一般的な用途は、次のとおりです。

- SQL SELECT ステートメントの WHERE 句に使用するフィルタ用フィールドをロードする。
- 処理オプション値をフィルタ用フィールドにロードする。
- システム日付のフェッチなど、フォームに対する一回限りのロジックを実行する。

Post Dialog is Initialized イベントは、レコードが存在しない場合にも実行されます。

この時点で[Find]ボタンをクリックし、グリッドを自動入力する必要があります。J.D. Edwards 標準では、自動検索機能は無効化されています。その後、SQL SELECT が作成されます。

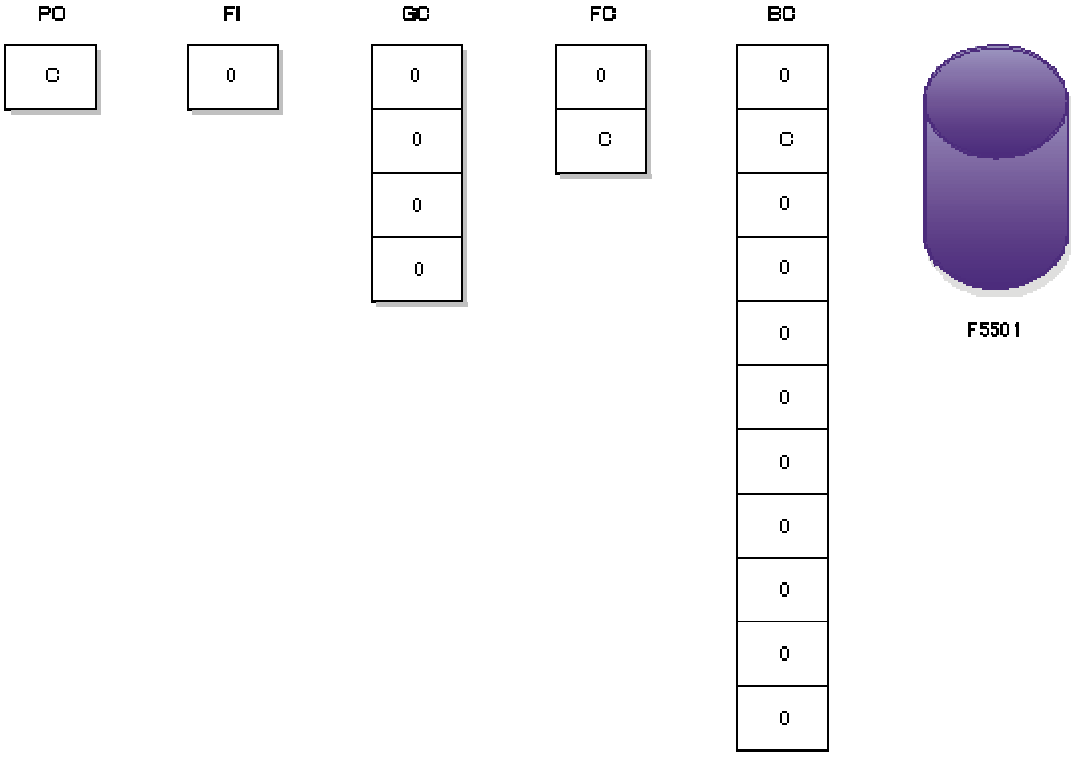
SQL SELECT

ユーザーが[Find]ボタンをクリックすると、WHEREくå を持つ SELECT ステートメントが作成されます。SQL SELECT ステートメントは、ビジネス・ビューのすべてのカラムを対象とします。

WHEREくå には、QBE 行かフィルタ用フィールドの値が含まれます。その後、WHERE 句は QBE とフィルタ条件に一致するすべてのレコードを取得するために使用されます。

次の図に、SQL SELECT が作成されて FETCH が発生する直前にランタイム構造体に格納されている情報を示します。

SQL Selectでビルドされる前のランタイム構造体の内容



WHILE ループでの動作

レコードは、WHILE ループで 1 つずつ取り出されます。WHILE ループでは次の処理が発生します。

- フォーム上でデータベース・レコードを表示できる間、およびグリッド・ローに空きがある間、検索する。
- ページ単位処理を実行する。
- SQL FETCH で 1 つのレコードを取得する。

ページ単位処理

ページ単位処理は、1 グリッドに表示できる数のレコードのみが取り出されることを意味します。たとえばグリッドにローが 3 つしか表示されない場合は、3 つのレコードが順番に取り出されます。次の 3 つのレコードを取り出すには、ユーザーがグリッド・スクロール・バーの下矢印をクリックする必要があります。グリッドはフォームの設計時にサイズを設定できます。

ページ単位処理が発生すると、グリッド表示がメモリにキャッシュされます。このメモリ・キャッシュは、メモリのランタイム構造体とは異なります。グリッドはキャッシュされるため、ユーザーはすべてのレコードをグリッドにロードした後、既に取り出されたレコードを上矢印を使用して表示できます。キャッシュによって、新しい FETCH は行われません。たとえば、レコード 45～47 がロードされた時点でユーザーが下矢印をクリックすると、レコード 48～50 がロードされます。次に上矢印をクリックしてレコード 45 をハイライトすると、レコード 45 はグリッド・キャッシュから取り出されるため、新しい FETCH で取り出す必要はありません。

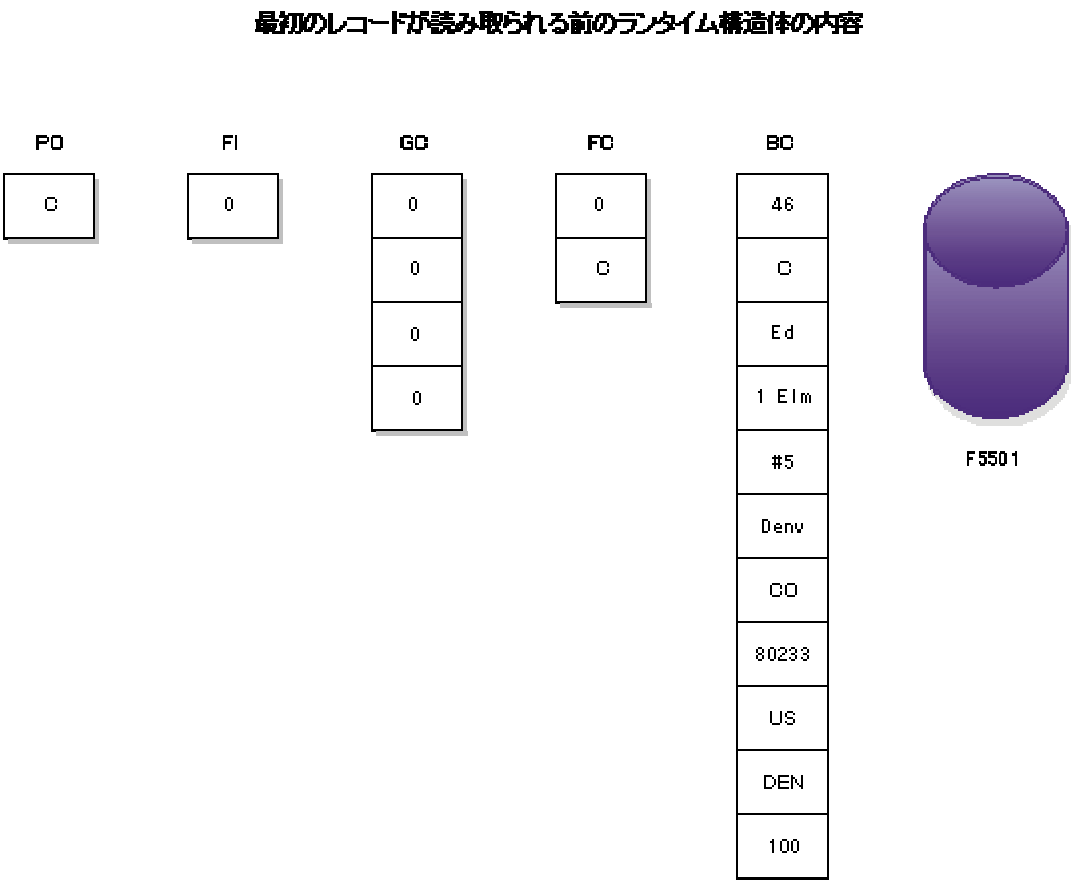
ページ単位処理が有効なときは、ユーザーが、グリッド・スクロール・バーの下矢印をクリックして最後のレコードを取り出されない限り、Last Grid Rec Has Been Fetche イベントに関連付けられたイベント・ルール・ロジックは実行されません。

ページ単位処理は通常、パフォーマンスを向上します。この機能は無効にできますが、J.D. Edwards の標準規則に従って、業務上の正当な理由のない限り、またはフォーム・タイプが〈Headerless Detail（見出しなし詳細）〉でない場合は無効にしないでください。

BC に代入されるデータベース値

WHILE ループで最初のレコードが取り出された後、データベース値が BC ランタイム構造体にコピーされます。テーブルでマークされた各カラムの値が BC ランタイム構造体要素に格納されます。

次の図に、WHILE ループの最初のレコードが読み込まれた時点でランタイム構造体に格納されている情報を示します。



Grid Rec Is Fetched

エンジンが再び一時停止して、イベントが処理されます。Grid Rec Is Fetched の実行時に処理されるロジックを追加できます。エンジンが一時停止したとき、ランタイム構造体には次の値が格納されています。

BC = データベースから取得された値（読み取られた最初のレコードの値）

FC = データベースから取得された値（データベース・フィールドの場合）

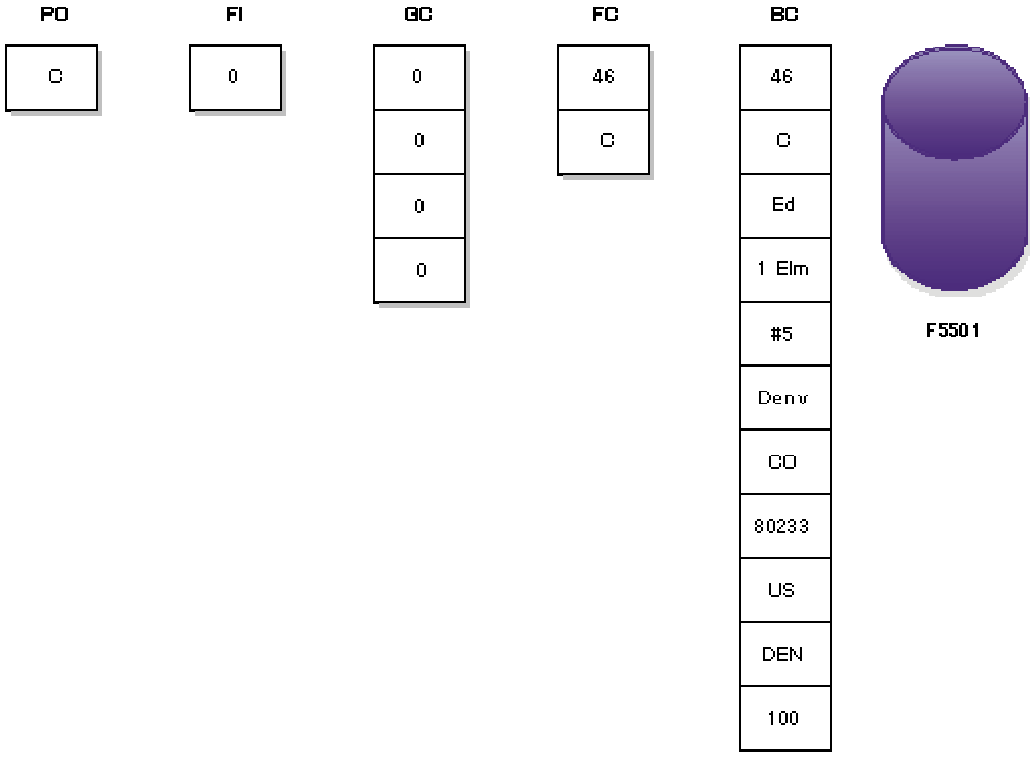
GC = 0

FI = 呼出し側フォームから渡された値（存在する場合）

PO = 処理オプションから渡された値

次の図に、Grid Rec Is Fetched を実行する直前にランタイム構造体に格納されている情報を示します。

Grid Rec is Fetched が実行される前のランタイム構造体の内容

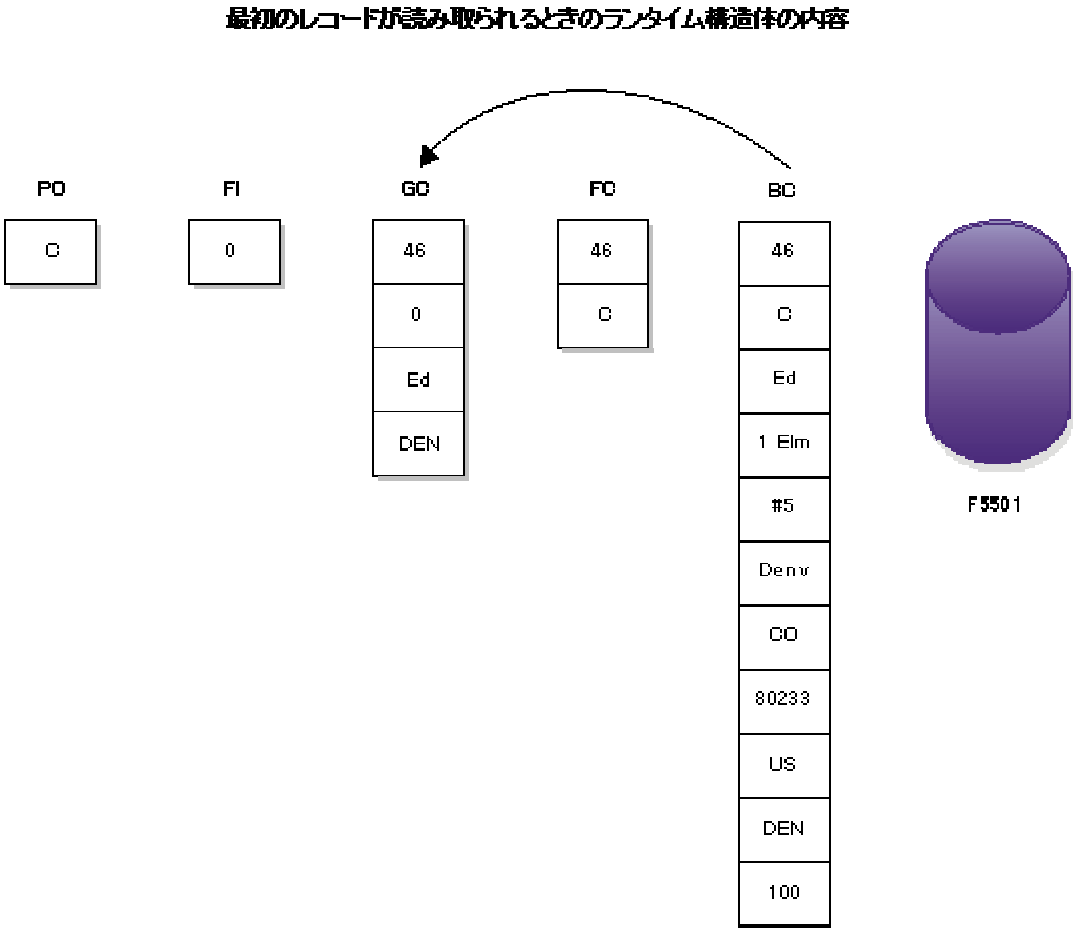


Grid Rec Is Fetched イベントの一般的な用途は、次のとおりです。

- グリッドのワーク・フィールドの値を計算する。
- ローのグリッドへの書込みを抑制する。

Grid Rec Is Fetched イベントの後（WHILE ループで最初のレコードが取り出されたとき）、GC ランタイム構造体に BC 値がコピーされます。

次の図に、WHILE ループで最初のレコードが読み込まれる時点でランタイム構造体に格納されている情報を示します。



Write Grid Line Before

エンジンが再び一時停止して、イベントが処理されます。Write Grid Line Before の実行時に処理されるロジックを追加できます。エンジンが一時停止したとき、ランタイム構造体には次の値が格納されています。

BC = データベースから取得された値（読み取られたばかりのレコードの値）

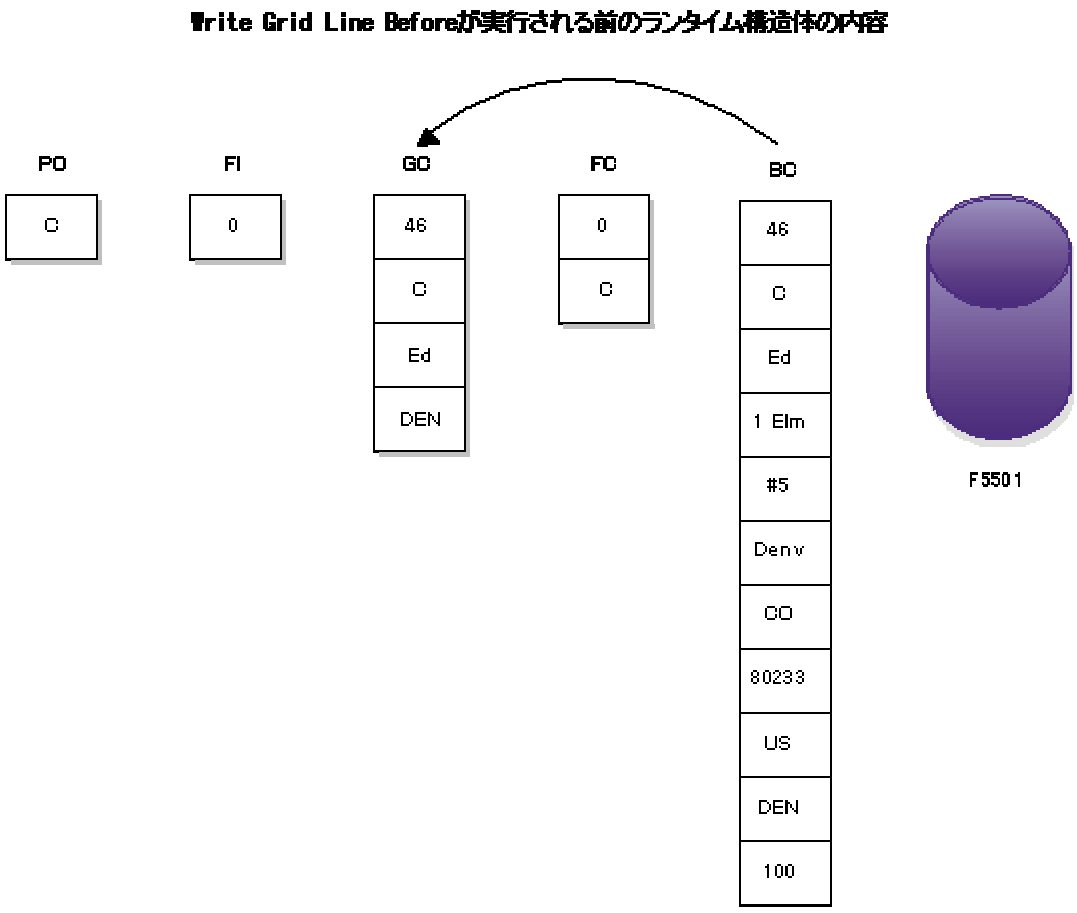
FC = データベースから取得された値（データベース・フィールドの場合）

GC = データベースから取得された値（直前に読み取られた値）

FI = 呼出し側フォームから渡された値（存在する場合）

PO = 処理オプションから渡された値

次の図に、Write Grid Line Before を実行する直前にランタイム構造体に格納されている情報を示します。

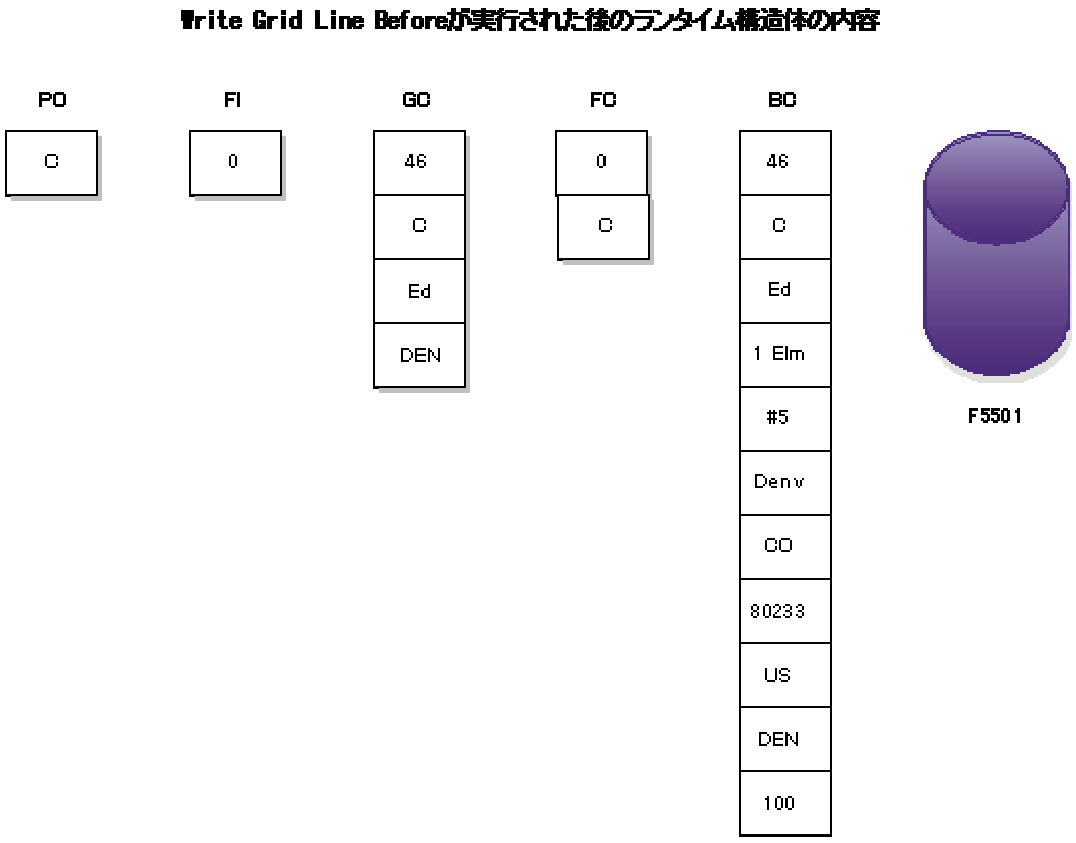


Write Grid Line Before イベントの一般的な用途は、次のとおりです。

- グリッド・ローの書込みを抑制する。
- ローがフォームに表示される前にロジックを追加する。
- グリッド・カラムの形式を変更する。
- 計量単位など、グリッドの値を変換する。
- グリッド・ローの追加情報（記述など）をビジネス・ビューにないテーブルから取り出す。

Write Grid Line Before イベントが処理されると、最初のレコードのデータベース値を含む GC 要素がフォームのグリッド・セルにコピーされます。

次の図に、この時点でランタイム構造体に格納されている情報を示します。



Write Grid Line After

エンジンが再び一時停止して、イベントが処理されます。Write Grid Line After の実行時に処理されるロジックを追加できます。エンジンが一時停止したとき、ランタイム構造体には次の値が格納されています。

BC = データベースから取得された値（読み取られた最初のレコードの値）

FC = データベースから取得された値（データベース・フィールドの場合）

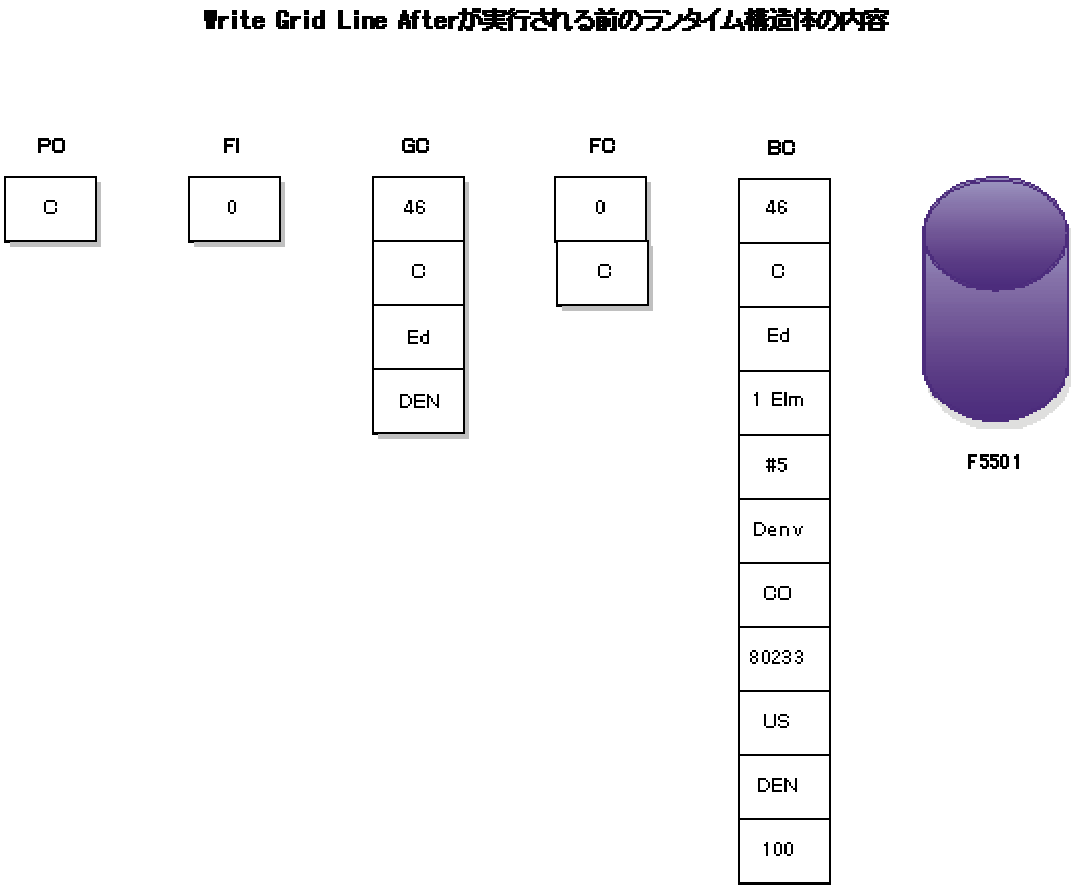
GC = データベースから取得された値（読み取られた最初のレコードの値）

FI = 呼出し側フォームから渡された値（存在する場合）

PO = 処理オプションから渡された値

グリッド・セルに現在のレコードが表示されます。

次の図に、Write Grid Line After を実行する直前に、ランタイム構造体に格納されている情報を示します。

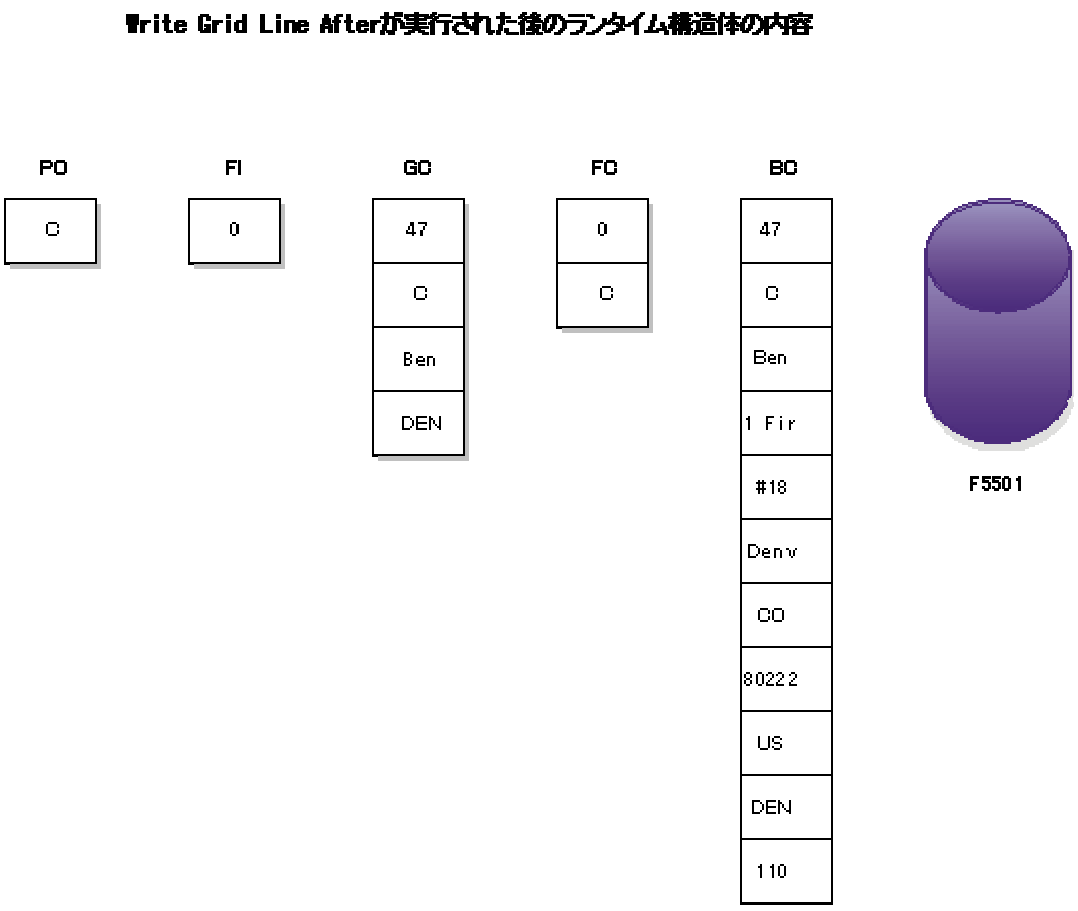


Write Grid Line After イベントは通常、ローがフォームに表示された後のロジックを追加するために使用します。

データベースのレコードは 1 つずつ読み込まれ、同じ処理ステップが実行されます。次のレコードが読み取られると、以下の処理ステップが実行されます。

- 条件に一致する次のレコードの SQL FETCH を実行する。
- データベースから BC 値を割り当てる。
- Grid Rec is Fetched を処理する。
- BC 値を GC に割り当てる。
- Write Grid Line Before を処理する。
- フォーム上のグリッド・ローに値を表示する。
- Write Grid Line After を処理する。

次の図に、2 番目のレコードで Write Grid Line After イベントが処理された後、ランタイム構造体に格納されている情報を示します。



グリッド・レコードが検索され、ユーザーが矢印キーをクリックして選択条件に一致するすべてのレコードを取り出すと、SQL FETCH は最終的に失敗します。グリッドがいっぱいでユーザーが下矢印キーをクリックしない場合は、一致するレコードがいくつか読み取られないことがあるため、FETCH が失敗することがあります。FETCH が失敗すると、Last Grid Rec Has Been Read が処理されます。

Last Grid Rec Has Been Read

エンジンが再び一時停止して、イベントが処理されます。Grid Rec Has Been Read の実行時に処理されるロジックを追加できます。エンジンが一時停止したとき、ランタイム構造体には次の値が格納されています。

BC = データベースから取得された値（読み取られた最後のレコードの値）

FC = データベースから取得された値（データベース・フィールドの場合）

GC = データベースから取得された値（読み取られた最後のレコードの値）

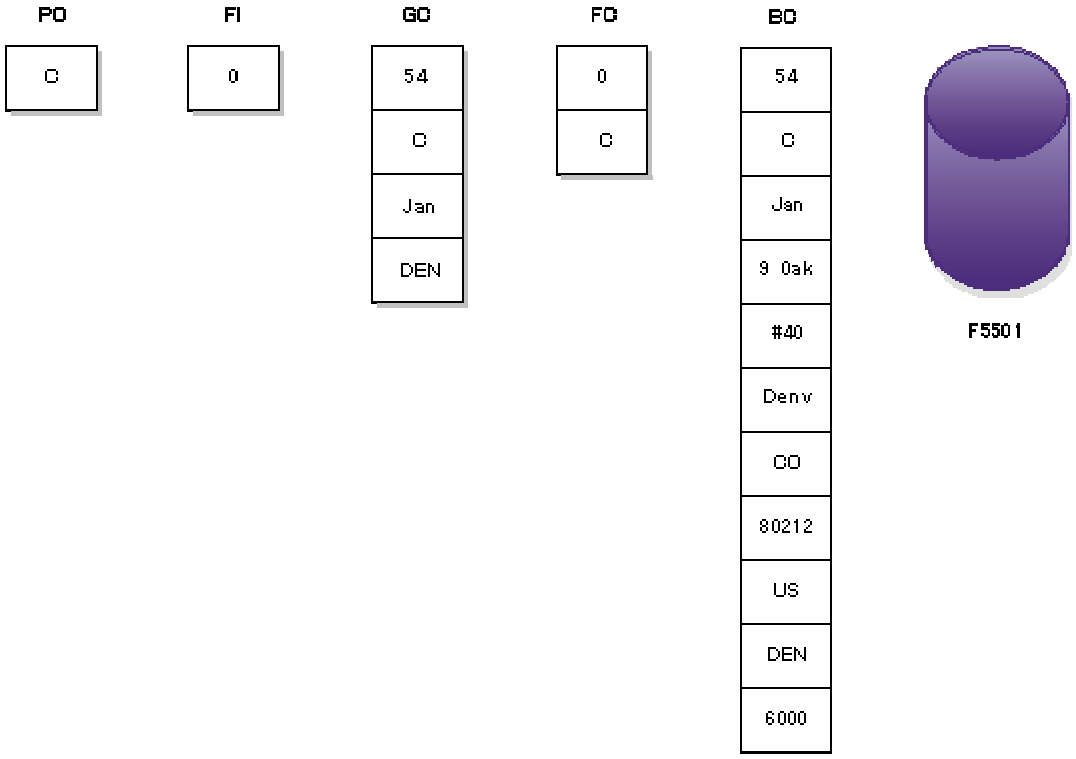
FI = 呼出し側フォームから渡された値（存在する場合）

PO = 処理オプションから渡された値

GC 値がフォームに表示されます。

次の図に、Last Grid Rec Has Been Read を実行する直前にランタイム構造体に格納されている情報を示します。

Last Grid Rec Has Been Readが実行される前のランタイム構造体の内容



Last Grid Rec Has Been Read イベントの一般的な用途は、次のとおりです。

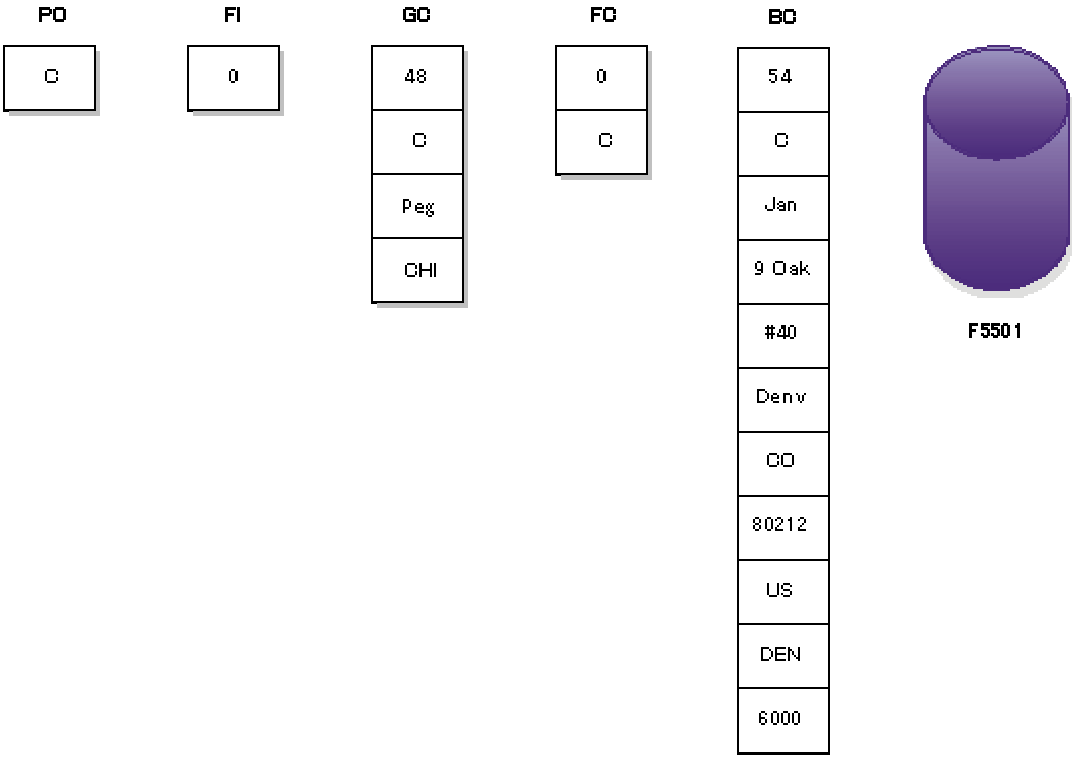
- 合計行をグリッドに書き込む。
- グリッド値に基づく合計を表示する。

選択ボタンの処理

ユーザーがグリッド・ローを選択して[Select(選択)]ボタンをクリックすると、フォーム値が GC 構造体にコピーし直されます。ユーザーがローをクリックしたので BC 構造体はそのまま変わらず、データベースは更新されません。そのため、インターコネクトの際に受け渡される値として GC を選択する必要があります。

次の図に、ユーザーがグリッド・ローを選択したときに、ランタイム構造体に格納されている情報を示します。BC 構造体と GC 構造体に格納されている値が異なることに注意してください。

ユーザーがグリッド・ローを選択したときのランタイム構造体の内容



Button Clicked

エンジンが一時停止して、イベントが処理されます。Button Clicked の実行時に処理されるロジックを追加できます。エンジンが一時停止したとき、ランタイム構造体には次の値が格納されています。

BC = データベースから取得された値（読み取られた最後のレコードの値）

FC = データベースから取得された値（データベース・フィールドの場合）

GC = 選択したグリッド・ローから取得された情報

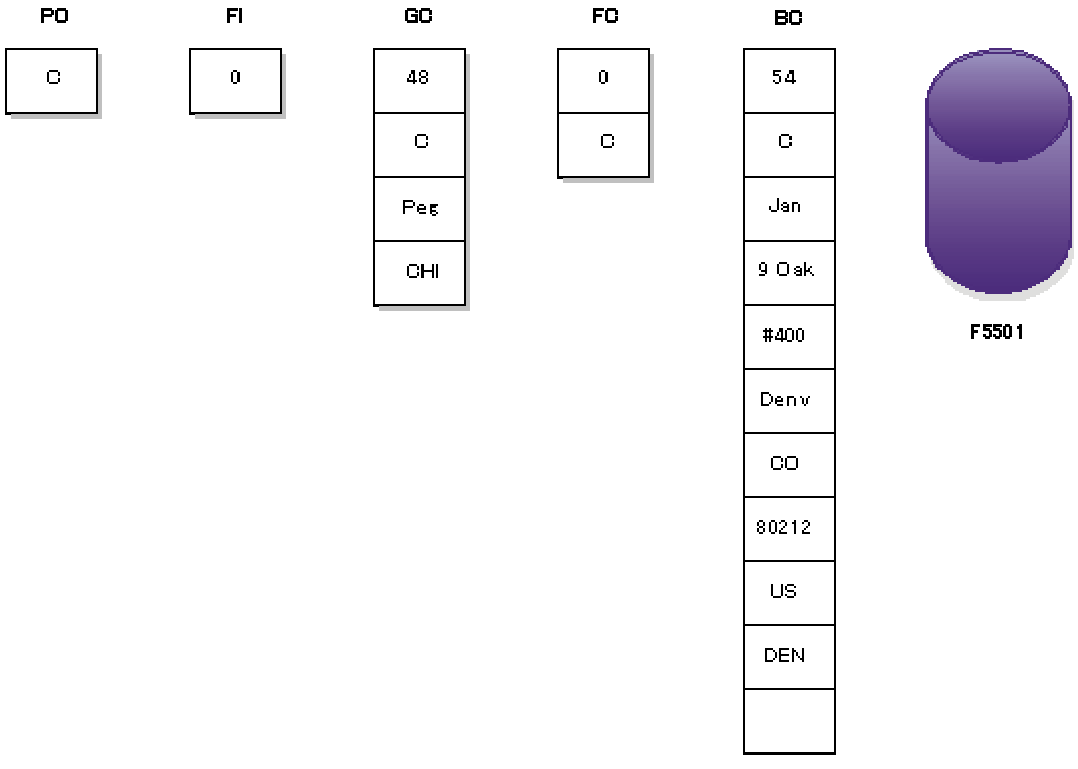
FI = 呼出し側フォームから渡された値（存在する場合）

PO = 処理オプションから渡された値

フォームでは、グリッド・セルに現在のレコードが表示されます。

次の図に、Button Clicked を実行する直前に、ランタイム構造体に格納されている情報を示します。

Button Clickedが実行される前のランタイム構造体の内容



Select Button Clicked イベントの一般的な用途は、次のとおりです。

- 別のフォームに接続する。
- フォームから GC 値を取り出し、受取側のフォームの FI 構造体に渡す。
- 複数のローが選択された場合に、Repeat Business Rules for Grid を使用してイベント・ルールを反復する(複数選択機能を有効化するには、フォーム・プロパティ[Multiple Select(複数選択)]も有効化する必要があります)。

注:

複数のグリッド・ローの選択を可能にする場合は、[Repeat Business Rules for Grid(グリッドのビジネス・ルールを反復)]をアクティブ化します。

追加ボタンの処理

ユーザーがグリッド・ローを選択した後、GC ランタイム構造体には、フォームのグリッド・レコードに表示されている値が割り当てられます。ユーザーは通常、追加操作の前にローを選択しませんが、ローがハイライトされている場合は、フォーム値が GC ランタイム構造体にコピーされます。ユーザーがレコードを選択しない場合、GC ランタイム構造体には、最後に読み取られたグリッド・レコードの値(存在する場合)が格納されます。データベースの更新は、ユーザーがローをクリックしただけでは行われません。

エンジンが一時停止して、Button Clicked イベントが処理されます。Button Clicked の実行時に処理されるロジックを追加できます。エンジンが一時停止したとき、ランタイム構造体には次の値が格納されています。

BC = データベースから取得された値(読み取られた最後のレコードの値)

FC = データベースから取得された値(データベース・フィールドの場合)

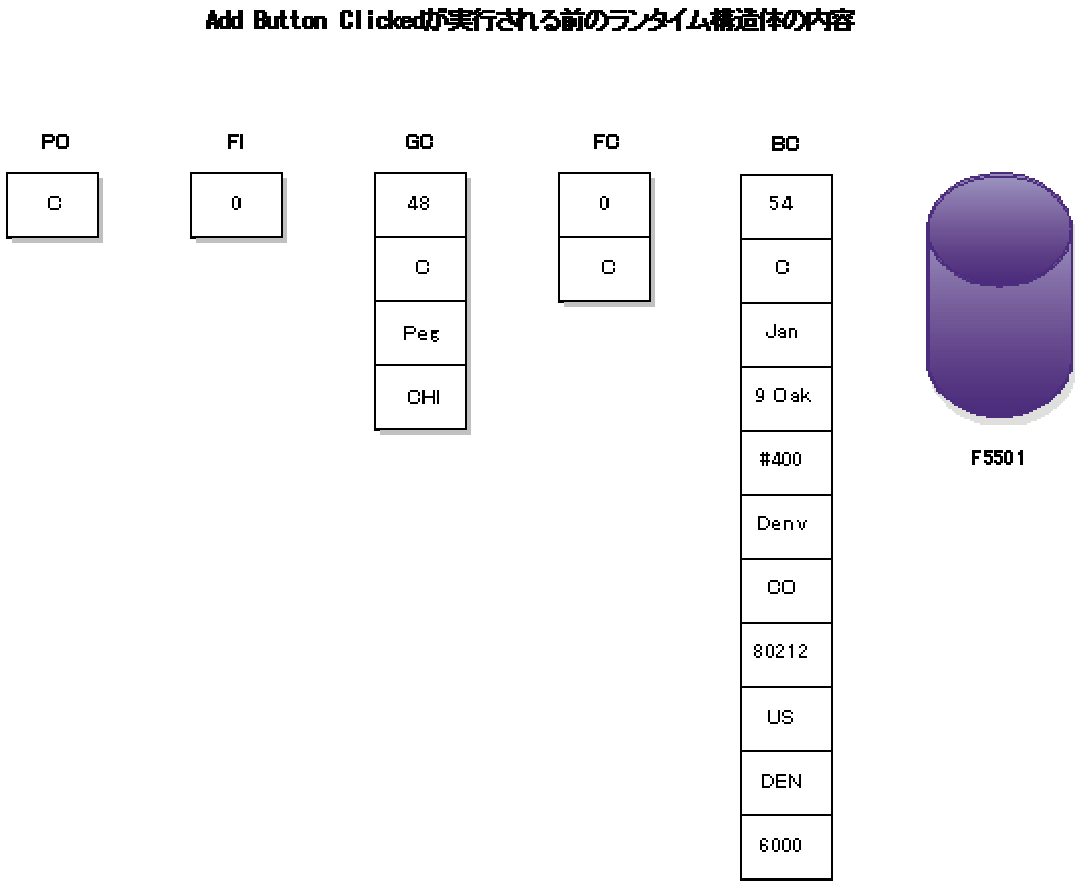
GC = 選択したグリッド・ローから取得された情報

FI = 呼出し側フォームから渡された値(存在する場合)

PO = 処理オプションから渡された値

これは追加操作なので、GC の内容はここでは関係ありません。BC と GC には同じ値が存在しません。

次の図に、Add Button Clicked を実行する直前にランタイム構造体に格納されている情報を示します。



Add Button Clicked イベントは通常、実際に追加操作が実行される修正/検査フォームや見出しなし詳細フォームなど、別のフォームに相互接続するために使用します。

新しいレコードを追加するため、一般に、ここの受け取り側のフォームに FI 構造体の GC 値を受け渡しません。

削除ボタンの処理

ユーザーがグリッド・ローを選択すると、フォームのグリッド・レコードに表示されている値が GC ランタイム構造体に割り当てられます。BC は依然として同じなので、データベースは更新されません。

エンジンが一時停止して、Button Clicked イベントが処理されます。Button Clicked の実行時に処理されるロジックを追加できます。エンジンが一時停止したとき、ランタイム構造体には次の値が格納されています。

BC = データベースから取得された値（読み取られた最後のレコードの値）

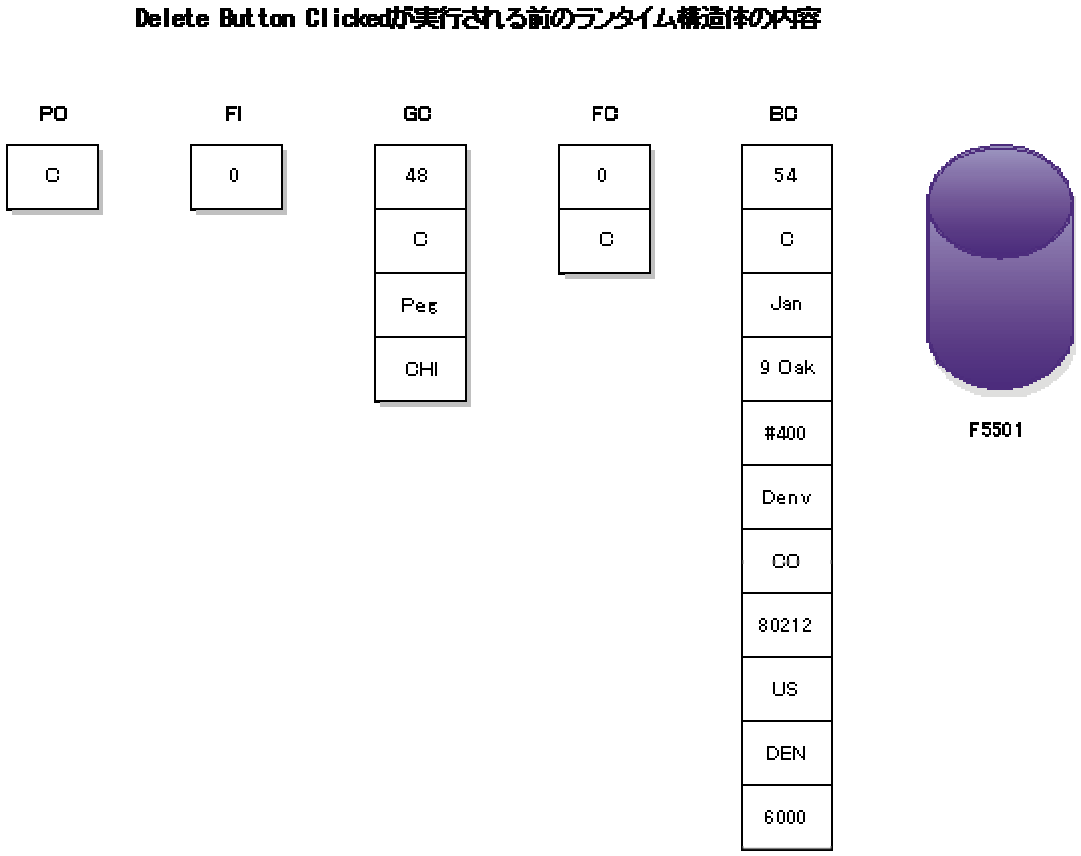
FC = データベースから取得された値（データベース・フィールドの場合）

GC = 選択したグリッド・ローから取得された情報

FI = 呼出し側フォームから渡された値（存在する場合）

PO = 処理オプションから渡された値

次の図に、Button Clicked を実行する直前にランタイム構造体に格納されている情報を示します。

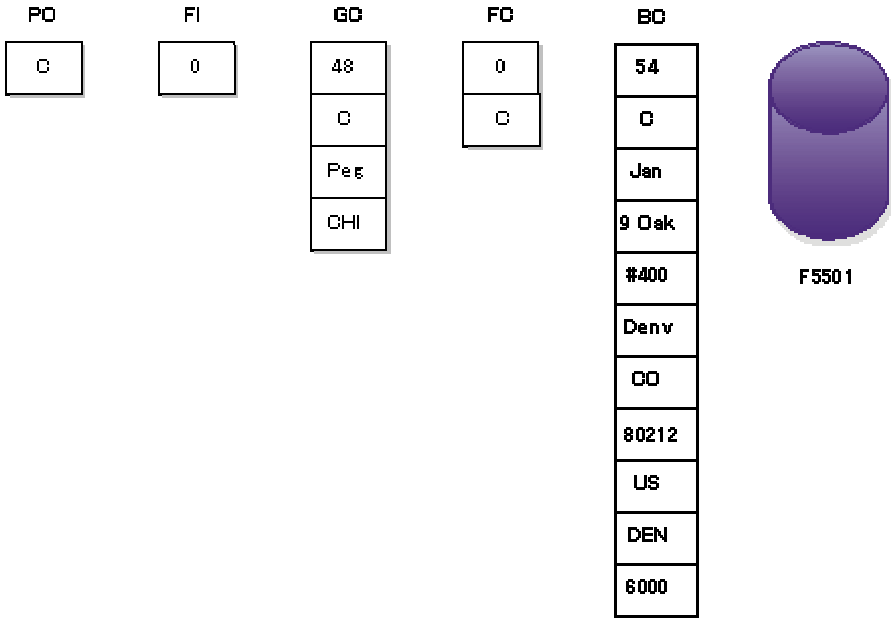


次に、Delete Grid Rec Verify Before イベントが発生します。

Delete Grid Rec Verify Before

エンジンが一時停止して、Delete Grid Rec Verify Before イベントが処理されます。Delete Grid Rec Verify After の実行時に処理されるロジックを追加できます。このイベントに関連付けられているロジックは、ユーザーが[Delete]をクリックしてから〈Verify(確認)〉確認フォームが表示されるまでの間に処理されます。

Delete Grid Rec Verify Beforeが実行されるときのランタイムデータ構造体の内容



次に、Delete Grid Rec Verify After イベントが発生します。

Delete Grid Rec Verify After

エンジンが一時停止して、Delete Grid Rec Verify After イベントが処理されます。Delete Grid Rec Verify After の実行時に処理されるロジックを追加できます。

削除の有効性を確認するための編集を行うこともできます。たとえば、他のテーブルに従属レコードが含まれている場合は、それらが存在する限り該当するレコードを削除できません。

このイベントに関連付けられているロジックは、ユーザーが〈Verify〉確認フォームで[OK]をクリックした後に処理されます。ユーザーがこのフォームで[Cancel(取消し)]をクリックした場合、このイベントに関連付けられているロジックは実行されません。

次に、Delete Grid Rec From DB Before イベントが発生します。

Delete Grid Rec From DB Before

エンジンが再び一時停止して、Delete Grid Rec From DB Before イベントが処理されます。Delete Grid Rec From DB Before の実行時に処理されるロジックを追加できます。エンジンが一時停止したとき、ランタイム構造体 FC はブランクです。

Delete Grid Rec From DB Before イベントに関連付けられているロジックは、ユーザーが[OK]をクリックし〈Verify〉ウィンドウが処理されてから、レコードが実際に削除されるまでの間に処理されます。

Suppress Delete システム関数を Delete Grid Rec From DB Before イベントと共に使用すると、自動ツール削除をオフにして、自分で削除を行うことができます。たとえばビジネス関数を使用して、削除を実行できます。

Delete Grid Rec From DB Before イベントが処理された後、SQL DELETE ステートメントが作成されます。エンジンが再び一時停止し、処理するロジックを追加できます。FC はブランクです。この時点で SQL DELETE が実行され、現在のレコードがデータベースから削除されます。複数のレコードを選択したときは、1 回の削除呼出しですべてのレコードを削除できます。

Delete Grid Rec From DB After

エンジンが再び一時停止して、Delete Grid Rec From DB After イベントが処理されます。Delete Grid Rec From DB After の実行時に処理されるロジックを追加できます。このロジックは、レコードがデータベースから実際に削除された後に実行されます。このイベントは、ビジネス関数を呼び出して、ビジネス・ビューに存在しない関連テーブルから情報を削除する場合に使用できます。

Delete Grid Rec From DB After イベントに関連付けられているロジックは、ユーザーがレコードを削除するために[OK]をクリックし、〈Verify〉確認フォームが処理されてレコードが削除された後に処理されます。

All Grid Recs Deleted From DB

エンジンが再び一時停止して、All Grid Recs Deleted from DB イベントが処理されます。All Grid Recs Deleted from DB の実行時に処理されるロジックを追加できます。この時点で、FC はブランクです。

ロジックは、レコードがデータベースから実際に削除された後に追加することもできます。All Grid Recs Deleted from DB イベントに関連付けられているロジックは、複数のグリッド行とそれらに対応するデータベース・レコードが削除された後に処理されます。このイベントに関連付けられているルールはフォームに表示されず、グリッド行とグリッド・レコードを操作できません。

イベント・ルール設計の処理

〈Event Rules Design〉を使用して、フォーム上のコントロールのイベント・ルール・ロジックを作成します。たとえば、検索/表示フォームで選択したレコードのデータを修正/検査フォームに渡し、そのレコードを改訂する場合を考えます。このタスクを達成するには、Button Clicked イベント用の[Select]ツールバー・オプションに関連付ける「フォーム・インターコネクト」イベント・ルールを作成します。

イベント・ルールを作成する前に、イベント・ルールで処理するコントロール（フォーム、ボタン、フィールド、グリッドなど）とイベントを検討します。次の質問に答えて、どのイベント・ロジックを実行するかを決定します。

- ユーザーがフォームを初期化するかどうか。
- ユーザーがボタンをクリックするかどうか。

- ユーザーがフィールドから出るかどうか。
- ユーザーがローを変更したりローを出るかどうか。

参照

- ❑ イベントに対する HTML ポスト（リフレッシュ）のオン/オフについては、『開発ツール』ガイドの「HTML クライアントについて」

イベント・ルールの作成と保存

フォームにコントロールを配置すると、アプリケーションの実行時に処理を発生させるためにはイベント・ルール・ロジックを作成する必要があります。フォームもコントロールの一種なので、フォームが初期化されるときに自動処理させるロジックを作成することもできます。

イベント・ルールを作成するには、〈Event Rules Design〉に表示されるツールバーのボタンの 1 つをクリックします。クリックしたボタンによって異なる作業領域が表示され、イベント・ルールを 1 行ずつ作成したり、操作することができます。〈Event Rules Design〉のボタンを使用すると、次の操作を行うことができます。

- ビジネス関数の関連付け
- システム関数の関連付け
- IF/WHILE ステートメントの作成
- 値または式の代入
- フォーム・インターコネクトの作成
- レポート・インターコネクトの作成
- IF ステートメントへの ELSE 句の挿入
- 変数の作成
- テーブル入出力の実行

► イベント・ルールを作成、保存するには

1. 〈Form Design〉で、フォーム上でコントロールを選択します。
2. [Edit]メニューから[Event Rules（イベント・ルール）]を選択します。
3. [Events（イベント）]リスト・ボックスから、Control Is Entered などのイベントを選択します。
4. 次のいずれかの[Event Rules]ボタンをクリックして、コントロールに適用するロジックを追加します。
    - Assignment/Expression（割当て/数式）
    - If/While
    - ビジネス関数
    - システム関数
    - Method Invocation（メソッドの起動）
    - Variables

- オフの場合
- Table I/O
- Report Interconnect (UBE)(レポート・インターコネクト)
- Form Interconnect

5. 次のボタンをクリックしてコメントを追加するか、イベント・ルールの特定の行を有効または無効にするか、削除します。
   - Comment(コメント)
   - Enable/Disable(有効/無効)
   - Delete(削除)

   [Enable/Disable]ボタンは、イベント・ルールの個別の行に作用します。イベント全体を使用不可にするには、フォーム最上部の[Disable Event(イベントを使用不可にする)]オプションをクリックします。

   イベント・ルールのロジックをテストする場合は、そのイベント・ルールの特定の行を使用不可にすることができます。ある行が不要であることを確認できたら、その行をイベント・ルールから削除することができます。

   連続する複数の行を有効、無効または削除するには、有効、無効または削除するすべてのイベント・ルールを Shift キーを押しながら選択します。適切な行を選択したら、[Enable/Disable]または[Delete]ボタンをクリックします。

6. [Save(保存)]をクリックしてイベント・ルールを保存します。
7. [OK]をクリックして〈Form Design〉に戻ります。

イベント・ルールを保存すると、イベント・ルールはアプリケーションと一緒に保存されます。アプリケーションを削除したり、アプリケーションを保存する前に取り消したりすると、イベント・ルールも削除されます。

フィールド記述

フィールド	記述
イベントの無効化	コントロールのイベント・ルールを使用不可にします(削除はしません)。アプリケーションのデバッグに便利です。問題が識別されるまで、特定のコントロールのルールを 1 つ以上使用不可にできます。 使用不可のイベント・ルールは、イベント・リストで赤くマークされます。
グリッドへのビジネス・ルールの繰返し – ER 設計	各グリッド選択にイベント・ルールを適用します。このチェックボックスはグリッド制御に対してのみ使用可能です。 注:全グリッドに複数選択があるとは限りません。開発者が、グリッド・プロパティ・フォームで複数の選択オプションを指定する必要があります。
保存	選択されたコントロールについて、全イベントに対するすべてのルールを保存します。 注:OK ボタンも終了前に保存を実行します。
コメント–イベント・ルール設計	イベント・ルールにコメントを挿入します。コメントはロジックに影響しません。ビジネス・ルールを説明するには、コメントを使用してください。

フィールド	記述
有効化/無効化 – イベント・ルール設計	イベント・ルールの単一行を使用可能にするか、使用不可にします。使用不可イベント・ルール行には、赤い感嘆符(!)が表示されます。
削除	オブジェクト、あるいはレコードを無効にします。 イベントルール設計では、削除ボタンは一度に 1 つの選択された行を除去します。IF/WHILE ステートメントを削除すると、関連 ELSE および終了句も削除されるが、それらのステートメント内のルールは削除されません。

イベント・ルールの検索

イベント・ルールのロジックが極端に大きくなった場合は、[Find]機能を使用してイベント・ルールで特定の文字列を検索することができます。

► イベント・ルールでテキスト文字列を検索するには

1. 〈Event Rules Design〉から、[Edit]メニューから[Find]を選択します。
2. 適切なオプションをクリックして検索方法を指定します。
3. 検索するテキストの文字列を入力して、[Find Next(次を検索)]ボタンをクリックします。

イベント・ルールの切取り、コピー、貼付け

イベント・ルールを切取りまたはコピーして、同じイベント、フォーム、アプリケーション、または異なるイベント、フォーム、アプリケーションにペーストすることができます。

また、イベント・ルールの行をワード処理文書に貼り付け(ペースト)することもできます。この機能は、プロジェクトを文書化するときに役立ちます。

次のボタンを使用することができます。

- 切取り
- コピー
- 貼付け
- 貼付けオプション

イベント・ルールを切り取ると、選択した行がソースから削除され、クリップボードに保管されます。

► イベント・ルールを切り取るには

1. 〈Event Rules Design〉で、イベント・ルールから切り取る行を選択します。
2. [Cut(カット)]ボタンをクリックします。

▶ イベント・ルールをコピーするには

1. 〈Event Rules Design〉で、イベント・ルールからコピーする行を選択します。
2. [Copy(コピー)]ボタンをクリックします。

イベント・ルールをコピーすると、選択したイベント・ルールの行がソースからコピーされ、クリップボードに保管されます。

▶ イベント・ルールをペーストするには

イベント・ルールをペーストすると、ペーストされたとおりにソースからのオブジェクトが解決されます。オブジェクトの一部が解決されると、それに最も近い一致するオブジェクトがコピー先イベント・ルールからペーストされます。コメント行は、部分的に解決されたイベント・ルール行の上とステータス・バーに表示され、そこに一部解決されたオブジェクトがあることがわかります。

1. 〈Event Rules Design〉で、イベント・ルールの各行を挿入したい位置の真上の行を選択します。
2. [Paste(ペースト)]ボタンをクリックします。

一部のオブジェクトは、コピー先イベント・ルールで解釈されません。これらのイベント・ルール行は無効にされ、コメント化されます。たとえば、ENDIF ステートメントに関連する IF ステートメントが欠落している場合、その ENDIF ステートメントはコメントとして扱われます。

条件文では、貼り付け操作によって、適切な論理構造を維持するために必要な要素が追加されます。たとえば、IF ステートメントを貼り付けるときに ENDIF ステートメントが存在しない場合は、貼り付け操作によって、一致する ENDIF ステートメントが追加され、ロジックが完全なものになります。

ペースト・オプションを設定すると、ペーストされた ER ブロックの前後にコメントを表示することができます。

▶ ペースト・オプションを設定するには

1. 〈Event Rules Design〉で、[Edit]メニューから[Paste Options(貼り付けオプション)]を選択します。
2. 〈Cut Copy Paste Options(カット/コピー/ペースト・オプション)〉で、[Paste Options]ボックスをクリックします。
3. コメントを表示させる行数を入力します。

情報を貼り付けると、貼付けの開始位置と終了位置がコメントで示されます。

イベント・ルールへのコメント行の追加

イベント・ルールコードを文書化するためにコメント行を追加することができます。

▶ イベント・ルールにコメント行を追加するには

1. 〈Event Rules Design〉で、コメントを追加したい位置にカーソルを合わせます。
2. [Comment]ボタンをクリックします。
3. コメントを入力して[OK]をクリックします。

   コメントは Visual C++の注釈として自動的に形式が設定されるので、挿入したテキストがコメントであることを示す記号(//)を入力する必要はありません。

イベント・ルール・ロジックの印刷

イベント・ルールのコードを印刷することができます。これは、特にイベント・ルールが多くのコード行から構成されているときに役立ちます。次のイベント・ルール・コードを印刷できます。

- アプリケーション全体
- フォーム
- コントロール
- 単一イベント

▶ イベント・ルール・コードを印刷するには

1. 〈Form Design Aid〉で、イベント・ルール・コードを印刷するコントロールをクリックします。
2. [Edit]メニューから[Event Rules]を選択します。
3. 〈Event Rules Design〉で、[Events]リスト・ボックスからイベントを選択します。

   イベント・ルールが作業領域に表示されます。

4. [File(ファイル)]メニューから[Print(印刷)]を選択します。
5. 〈Event Rules Printing(イベント・ルールの印刷)〉で、次のいずれかの出力先をクリックします。
   - Printer(プリンタ)
   - File(ファイル)
6. 次のいずれかの印刷範囲を選択します。
   - Application(アプリケーション)
   - Form(フォーム)
   - Control
   - Event(イベント)

7. 次のいずれかの改頁オプションを選択します。
    - Application
    - Form
    - Control
    - Event
8. 次の 1 つ以上の印刷 ER オプションを選択します。
    - Expand Arguments（引数の展開）
    - Date/Time Stamp（日時スタンプ）
    - Show Line Numbers（行番号の出力）
    - Show Comments（コメントの出力）
9. [OK]をクリックします。

イベント・ルールの検証

イベント・ルールを検証すると、既に存在しない変数やビジネス関数、不適切なビジネス・ビュー・カラムなどのエラーを検索できます。イベント・ルールは、作業中にもアプリケーションを保存するときにも検証できます。

イベント・ルールは、[Save]をクリックしたり[File]メニューから[Exit and save before exiting（保存して終了）]を選択すると、自動的に検証されます。終了時の検証でエラーが検出された場合、〈Form Design Aid〉は終了することもできれば、そのまま開いておくこともできます。

また、メニュー・オプション[Validate Event Rules（イベント・ルールの検証）]を選択することもできます。このオプションを選択すると、直ちにイベント・ルールが検証されます。〈Validate Event Rules〉は、2 つのレベルでエラーをチェックします。最初のレベルは、イベント・ルール・ビジネス関数やテーブル・イベント・ルールです。このレベルでは、イベント・ルール・コードがイベント・ルールを記述した言語から C 言語コードに変換され、コンパイルされます。2 番目のレベルはアプリケーションです。このレベルでは、データ・タイプの不一致やデータ構造体のエラー、変数やコントロールの紛失といったエラーがイベント・ルールでチェックされます。〈Generator（ジェネレータ）〉は、ロジックのエラーや構文をチェックしません。

エラーが検出されるとエラー・ログが作成され、b7¥prod¥log¥p1234.log というようなファイルに保管されます（prod の個所には、実際の環境名が入ります）。エラーが存在しない場合、ログは作成されません。

▶ イベント・ルールを検証するには

〈Form Design Aid〉で、[File]メニューから[Validate Event Rules]を選択します。

イベント・ルールの検証を開始すると、〈Validate Event Rules（イベント・ルールの検証）〉フォームにメッセージが表示されます。検証が正常に終了すると、「Validation Successful（検証完了）」というメッセージが表示されます。

IF ステートメントと WHILE ステートメントの処理

IF ステートメントと WHILE ステートメントは、イベント・ルールに対する条件命令です。イベント・ルールがアクティブ化されると、この 2 つのステートメントによって条件が評価され、ロジックのフローが実行されます。

IF ステートメント

IF ステートメントでは条件付きのイベント・ルール・ロジックが実行されます。つまり、条件が真のときにのみロジックが実行されます。イベント・ルールを処理するときに、IF ステートメントが 1 回だけ評価されます。

通常、IF ステートメントは次の形式になっています。

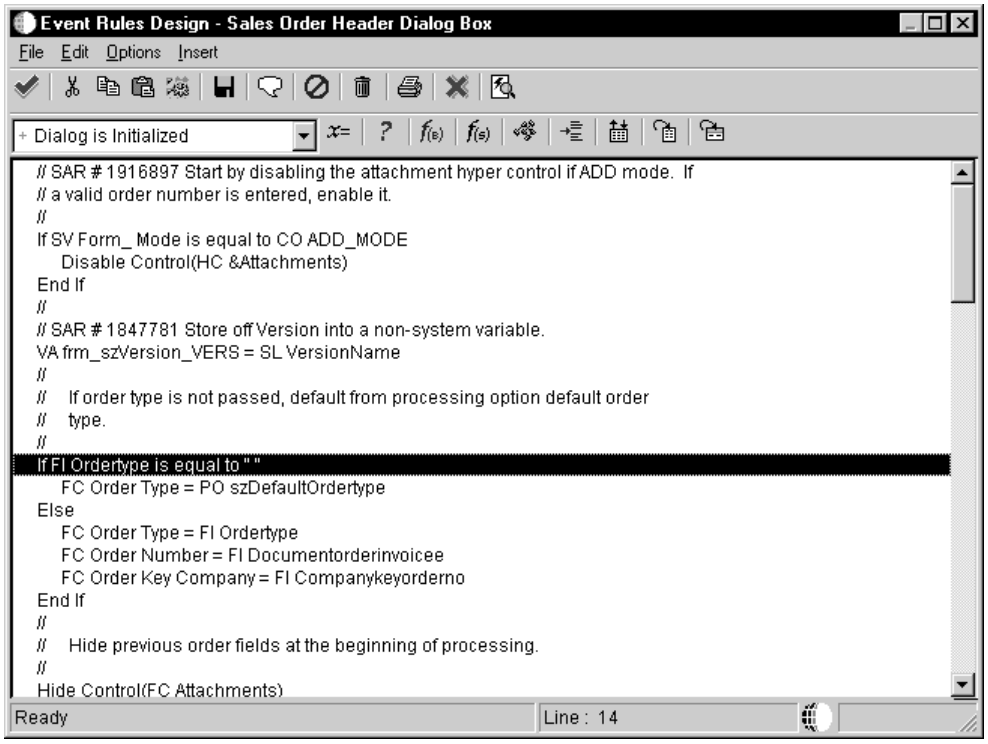
```
IF condition X is true
    Do Y
ELSE
    Do Z
ENDIF
```

IF ステートメントは次のように評価されます。

> 条件 X が真であれば、Y を実行し、ELSE ステートメントは評価しません。条件 X が真でなければ、ELSE 句に進み、Z を実行します。

ELSE 句の使用は任意です。ELSE 句が不要な場合は、次のステートメントに進んで評価します。

次の例は、〈Sales Order Entry（受注オーダーの入力）〉からの IF ステートメントを示しています。



WHILE ステートメント

WHILE ステートメントでは、条件付きのイベント・ルール・ロジックが繰り返されます。つまり、ロジックは条件が真である限り繰り返し実行されます。WHILE ステートメントによって、ループ（反復）テストの実行が制御されます。

通常、WHILE ステートメントは次の形式になっています。

```
WHILE condition X is true
    Do Y
ENDWHILE
```

WHILE ステートメントは、次のように評価されます。

> 条件 X を評価します。真であれば、Y を実行します。条件 X の評価を繰り返し、偽になったらループを終了します。

If/While ステートメントの作成

[If/While]ボタンを使用して、イベント発生時に実行する条件ロジックを作成します。IF ステートメントを作成すると、ELSE 節が自動的に挿入されます。ただし[Delete]ボタンを使用して ELSE 節を削除し、[Insert Else(Else の挿入)]ボタンを使用して再挿入できます。IF 節や WHILE 節を削除すると、関連する ELSE 節と ENDIF 節または ENDWHILE 節も削除されますが、これらの中のルールは削除されません。

各ステートメントの順序は、行単位のドラッグ&ドロップ操作によって変更できます。イベント・ルールの順序を変更すると、構文に問題が生じることがあります。[Save]ボタンまたは[OK]ボタンをクリックすると、構文が検証されます。構文エラーが検出された場合は、イベント・ルールを無効にして操作を継続するか、またはイベント・ルールを編集してエラーを解消できます。

範囲とリスト値を使用するときに有効な論理演算子は、「Is Equal To(等しい)」と「Is Not Equal To(等しくない)」のみです。条件設定ツールは、比較演算子が「等しい」または「等しくない」ことを確認して、範囲リテラルとリスト・リテラルの誤用を防ぎます。[Range(範囲)]または[List(リスト)]を選択したときに、比較演算子の「Is Equal To」または「Is Not Equal To」が比較演算子フィールドに設定されていない場合、セルはクリアされます。

ステートメントを変更するには、変更するセルまたは値を選択するためにセルをクリックします。これでセルの値を変更できます。[OK]をクリックすると構文がチェックされます。行の順序を変更するには、ロー見出しをクリックしてロー全体を選択します。次に、ツールバーの上/下矢印キーを使用して、その行を移動します。ただし、IF/WHILE 行は移動できません。

ステートメントを削除するには、ステートメントの行を選択して[Delete]をクリックします。ステートメントの一番上の節(IF または WHILE)は IF/WHILE 設計では削除できません。これは、〈Event Rules Design〉ウィンドウからのみ削除できます。

▶ If/While ステートメントを作成するには

1. 〈Event Rules Design〉で、[Event Rules Design]ペインからイベントを選択して[If/While]ボタンをクリックします。

   〈Criteria Design(条件設計)〉グリッドの各セルは、条件となる要素を表しています。セルをクリックするか、または Tab キーを使用して移動してセルを選択すると、有効なオプションのリストが表示されます。オプションを選択するには、オプションをダブルクリックするか、選択して Enter キーまたは Tab キーを押します。オプションを選択するには、マウスを使用するか、そのオプションの名前を入力します。

   〈Criteria Design〉には先行入力機能があり、項目の最初の数文字を入力すると、それらの文字から始まる有効な項目リストを自動的に表示できます。

2. 次のいずれかの演算子を選択します。
   - IF
   - WHILE
3. 使用可能なデータ項目リストから左オペランドを選択します。

   マウスの右ボタンをクリックして、有効なデータ項目を名前やオブジェクト・タイプでソートできます。タイプが 1 つしか存在しない場合、ソート・オプションは使用できません。

使用可能なデータ項目は、次のオブジェクト・タイプに従って分類できます。

BC　このフォームのビジネス・ビューのカラム

GC　このフォームのグリッドのカラム

FI　このフォームにフォーム・インターコネクトを通じて渡される値

FC　プッシュボタンなど、このフォーム上のコントロール

PO　アプリケーションの処理オプションからの値

HC　ハイパー・コントロール

SV　システム変数

SL　システム値

VA　変数

4. 次の論理演算子リストから比較演算子を選択します。
   - 同等(=)
   - 同等(=)ではない
   - より小さい(<)
   - 以下(<=)
   - より大きい(>)
   - 以上(>=)
5. オブジェクト・リストから右オペランドを選択します。
6. リテラルを割り当てるには、<Literal>を選択します。

   [Literal(リテラル)]ダイアログの内容は、データ項目によって異なります。

7. [Save]をクリックして条件ステートメントを保存し、〈Event Rules Design〉ウィンドウに戻ります。

   条件が不完全な場合(たとえば、右オペランドが欠落している場合など)は、条件を修正するか、削除してください。

複合 IF ステートメントを作成するには、[And]オプションまたは[Or]オプションを選択してロジックを継続することができます。

イベント・ルール・ロジックでのリテラル値の定義

イベント・ルールでは、リテラル値を定義することができます。

▶ リテラル値を定義するには

1. 〈Criteria Design〉の右オペランド・リストで、<Literal>をダブルクリックします。
2. 次のタブのいずれかをクリックします。
   - Single value（単一値）
   - Range of values（値の範囲）
   - List of values（値のリスト）
3. 選択したタブの適切なフィールドに値を入力します。
4. [OK]をクリックします。

フィールド記述

フィールド	記述
値 – リテラル	イベント・ルール機能設計で特定の英数値をリテラルとして定義します。
範囲 – リテラル	イベント・ルール機能設計内で、リテラルに対する英数値の範囲を定義します。
リスト – リテラル	リテラル値で使用するリストに値の追加や削除を行います。

イベント・ルール変数の処理

イベント・ルール変数はデータ辞書項目に関連付けられていますが、データ辞書には常駐していません。作成したイベント・ルール変数は、特定の対話型アプリケーション、バッチ・アプリケーション、またはイベント・ルール・ビジネス関数でのみ使用できます。

イベント・ルール変数はその使用方法を作成時に定義するので、実行時に自動的に初期化されます。

隠しフィールドの代わりにイベント・ルール変数を使用してください。実行時、イベント・ルール変数は隠しフィールドよりもシステム・リソースを使用しません。

イベント・ルール変数は、そのスコープによって用途が決まります。対話型アプリケーションとバッチ・アプリケーションでは、異なるスコープ・オプションを使用できます。たとえば次の処理を実行できます。

- レポートのどこからでもレポート変数を参照する。
- イベント変数を、それが作成されたイベントでのみ参照する。

対話型イベント・ルール変数

対話型イベント・ルールの変数は、次の 3 つのレベルで使用することができます。

- フォーム・レベル
- グリッド・レベル
- イベント・レベル

フォーム・レベル

フォーム・レベルの変数は、フォームのすべてのコントロールに対するどのイベントにも使用することができます。この種の変数はフォームと共に初期化され、フォームが閉じられるまで値を保持します。

グリッド・レベル

グリッド・レベル変数はフォーム変数のサブ・タイプで、グリッドを持つすべてのフォームで使用できます。そのフォームのすべてのイベントから使用でき、現在のローに適用されます。グリッドに追加された新規のローはいずれも同じ 1 組の変数を持ちます。これらの変数はグリッドに新しいローを追加するたびに初期化し直されます。これらの変数は、グリッドの一時作業フィールドとして使用できます。可能であれば、システム・パフォーマンスを改善するために、隠しワーク・フィールドの代わりに変数を使用してください。隠しワーク・フィールドの代わりに変数を使用すると、変数はフォーマットされないため、J.D. Edwards フォームの初期化時のパフォーマンスが向上し、時間を節約できます。グリッド変数はイベント・ルール行のすべてのタイプで使用できますが、グリッド・ローに関連付けられているイベントでのみ使用してください。

イベント・レベル

イベント・レベルの変数は、それが追加されたフォーム・コントロールのイベントでのみ使用することができます。この種の変数は、フォーム・コントロールに対するイベントが処理されるたびに初期化し直されます。

バッチ・アプリケーション・イベント・ルール変数

バッチ・イベント・ルール変数は、次のレベルで使用できます。

- レポート
- セクション
- イベント

イベント・ルール変数の作成

作成されたイベント・ルール変数は、それを追加した〈Event Rules Design〉の使用可能オブジェクト・リストに表示されます。イベント・ルール変数は、リストの使用可能オブジェクトを使用する場合と同様にイベント・ルールで使用します。イベント・レベル変数を作成してもイベント・ルールで使用しない場合、〈Form Design Aid〉は自動的に削除します。

イベント・ルール変数を追加した後で修正することはできません。代わりに、削除して新規作成する必要があります。

各変数には、指定したスコープに基づいて、次のいずれかのプレフィックスが自動的に割り当てられます。

- frm_（フォーム）
- evt_（イベント）
- grd_（グリッド）
- rpt_（レポート）
- sec_（セクション）

► イベント・ルール変数を作成するには

1. 〈Event Rules Design〉で、[Variables（変数）]ボタンをクリックします。

   〈Variables（変数）〉フォームには、処理するイベント・ルールのタイプ（対話型アプリケーション、バッチ・アプリケーションまたはビジネス関数イベント・ルール）によって異なるスコープ・オプションが表示されます。

2. [Add]ボタンの下にある[variable naming（変数名）]フィールドに値を入力します。

   注：
   イベント・ルール変数には、C 変数と同様の名前を設定し、xxx_yyzzzzzz_AAAA という形式にします。

   各文字は次の意味を表します。

   xxx = スコープに応じて、J.D. Edwards ソフトウェアが自動的に割り当てるプレフィックス。次のような例があります。

   frm_（フォーム・スコープ）

   evt_（イベント・スコープ）

   y = 次の C 変数のハンガリー表記。

   c –文字

   h –ハンドル要求

   mn –数字

   sz –文字列

   jd –ユリウス日

   id –ポインタ

   zzzzzz = 割り当てる変数名。語頭を大文字にします。

   AAAA = データ辞書エイリアス（すべて大文字）

   イベント・ルール変数にはスペースを使用しないでください。

たとえば、Branch/Plant（事業所）イベント・ルール変数は、evt_szBranchPlant_MCU という名前になります。

3. 作成した変数の用途に応じて、次のいずれかの[Scope（スコープ）]オプションをクリックします。
   - Form
   - Event
4. [Form Scope（フォーム・スコープ）]を選択してグリッド変数を使用する場合は、[Grid（グリッド）]オプションをオンにします。
5. データ辞書ビジュアル・アシストをクリックして、データ辞書項目を参照します。
6. 変数リストを表示するには、[Show All（すべて表示）]ボタンをクリックします。
7. 変数を関連付けるデータ項目を選択して、[Add（追加）]をクリックします。

   選択したスコープのタイプに基づいて、変数にプレフィックスが自動的に割り当てられます。
8. [OK]をクリックして、対話型、バッチ、またはビジネス関数イベント・ルールの設計モードに戻ります。

   新規の変数を使用できます。

フィールド記述

フィールド	記述
スコープ-変数	変数をどのように使用するか決定します。次のような例があります。 o そのセクション内のセクション変数 o レポートのレポート変数 o イベント変数が作成されたイベント内のみのイベント変数 対話型またはバッチ・アプリケーションには異なるスコープのオプションが使用可能です。

自動行採番に使用するイベント・ルールの作成

行番号の自動採番に使用するイベント・ルールを作成することができます。

▶ 自動行採番に使用するイベント・ルールを作成するには

1. 行番号の値を保持する変数を作成します。データ辞書項目の LNID を使用します。

   VA frm_LineNumberCounter_LNID
2. この変数を Post Dialog is Initialized イベント時点で初期化します。

   VA frm_LineNumberCounter_LNID = 0

3. Grid Record is Fetched イベントで、データベースから行が取り出されるたびに行番号を設定します。

   If BCLineNumber > VA frm_LineNUmber_LNID

      VA frm_LineNumberCounter_LNID = BC LineNumber

   End If

4. Add Last Entry Row to Grid イベントで、LineNumber を増分し、使用可能な次の行に新しい値を割り当てます。

   VA frm_LineNumberCounter_LNID = VA frm_LineNumberCounter_LNID + 1

   GC LineNumber = VA frm_LineNumberCounter_LNID

イベントへのシステム関数の関連付け

[System Function(システム関数)]ボタンを使用すると、既定の J.D. Edwards システム関数をイベントに関連付けることができます。たとえば、次の処理を実行できるシステム関数をイベントに関連付けることができます。

- コントロールの表示/非表示を切り替える。
- 日付を挿入する。
- メディア・オブジェクトを表示する。

► システム関数を関連付けるには

1. 〈Event Rules Design〉で、イベントを選択します。
2. [f(S)]ボタンをクリックします。
3. [System Functions]ボックスでカテゴリを選択します。
4. 関連付けるシステム関数を選択します。
5. [Available Objects list(使用可能なオブジェクト・リスト)]で、システム関数を渡すためのオブジェクトを選択します。
6. [OK]をクリックし、システム関数をイベント・ルールに追加します。

常用システム関数

ここでは、使用可能なシステム関数とその用途について説明します。

コントロール・システム関数

次の表に、各種のコントロール・システム関数を示します。

Clear Edit Control Error（編集コントロール・エラーのクリア）	特定のコントロールに設定されているエラーをクリアします。ランタイム・エンジンは、特定のイベントで自動的にエラーを設定してクリアしますが、他のイベントのエラーはクリアしないため、このシステム関数は重要ですたとえば、Dialog is Initialized イベントを使用した場合、このイベントは、エラーをリセットし、コントロールを再編集するために再実行できません。このような場合、Clear Edit Control Error を使用すると、他のイベントでエラーをクリアすることができます。
Disable Control（コントロールの無効化）	コントロールを動的に有効/無効に切り替えるために使用します。この関数を使用しない場合は、コントロールのプロパティを使用します。このシステム関数は、取引での為替レートなど、ユーザーによる変更を禁止する一定のフィールドを無効にするために使用できます。
Enable Control（コントロールの有効化）	コントロールを有効にするために使用します。たとえば、変更する権限を管理者のみに割り当てる場合に使用できます。
Hide Control（コントロールの非表示）	コントロールの表示/非表示を動的に切り替えるために使用します。ユーザーがフィールドの情報を特別に要求するまで、そのフィールドを非表示にできます。たとえば、住所録でレコードを追加するときに、既に追加されているレコードを表示できます。
Set Edit Control Error	特定のコントロールにエラーを設定し、フィールドを赤くするために使用します。
Set Status Bar Text（ステータス・バー・テキストのセット）	ステータス・バーにテキストを表示するために使用します。このテキストでエラーの原因を示し、エラーの訂正後にステータス・バーをクリアできます。また、処理がバックグラウンドで行われていたり、完了するまで一定の時間がかかるときにも使用できます。このような場合は、「Please Wait ...（お待ちください）」というメッセージを表示できます。
Was Value Entered（値の入力有無）	エラーが発生して情報が変更されたときなどに、データが変更されたかどうかを確認するために使用します。このシステム関数のオプションには、変更されたコントロールを判別する[All Controls（すべてをコントロール）]値があります。トランザクション・タイプのプログラムで有効です。

注:

次のコントロール・システム関数はできるだけ使用しないことをお薦めします。

- Set Edit Control Color
- Set Edit Control Font

Set Edit Control Color は必須フィールドに色を追加します。このシステム関数を使用する場合は、色彩を見分けにくいユーザーも存在することに注意してください。Set Edit Control Font を使用すると、フィールド（負数を使用するフィールドなど）が明確になりますが、標準以外のフォントを使用すると、わかりづらくなったり、フォームの外観を損ねる可能性があります。

参照

- ランタイム・エンジンによるエラーの設定とクリアについては、『開発ツール』ガイドの「メッセージ処理」

メッセージ・システム関数

Mail Send Message（メッセージの送信） 作成したメッセージを識別するメッセージ ID（シリアル番号）を返します。メッセージ ID は、他のメッセージ処理関数に対するキー・パラメータで、アプリケーションはこの ID を使用して他のシステム関数を実行します。顧客の与信限度額超過に対する承認を求める管理者にメッセージを生成する場合にも使用できます。

Forward Message（メッセージの転送） メッセージを転送するために使用します。メッセージを転送するときに元のメッセージをそのまま残しておくことができます。

Mail Delete Message（メッセージの削除） メール・メッセージを削除するために使用します。

Update Message（メッセージの更新） メッセージの書き換えに使用します。その後で Forward Message 関数を使用して新しいメッセージを転送できます。

汎用システム関数

Copy Currency Information （通貨情報のコピー）	通貨情報を変更したときに、その情報を使用するすべてのアプリケーションに変更が確実にコピーされるようにします。小数点以下桁数が頻繁に変更される地域でソフトウェアを開発する場合に有効です。また、会社で基準通貨を変更するときにも使用できます。
Press Button （ボタン押下）	イベント・ルールをイベント間で再利用し、それらのイベント・ルールを使用するすべての個所でコードを記述し、維持するという手間を省くために使用します。イベント・ルールを再利用するには、フォームでプッシュ・ボタンを作成し、再利用可能なすべてのイベント・ルールを button clicked イベントに配置します。次に、このシステム関数を使用してボタンを押し、この ER を実行したい他のすべてのイベントから ER を実行します。
Set Control Focus （コントロール・フォーカスのセット）	特定のスポットにフォーカスを設定するために使用します。たとえば、ユーザーがフォームを開いたときにグリッドの詳細部にフォーカスを設定することもできます。また、特にフォーカスが追加と変更で異なる場合に、デフォルトのカーソル位置を一時変更するために使用することもできます。
Stop Processing （処理の中止）	エラーが発生した場合などに処理を停止するために使用します。
Suppress Add （追加の抑制）	入出力用のフォームの標準フローを制御するために使用します。たとえば、マスター・ビジネス関数を使用するトランザクション入力プログラムでは、追加を含む、すべての入出力をマスター・ビジネス関数で実行します。
Suppress Delete （削除の抑制）	入出力用のフォームの標準フローを制御するために使用します。たとえば、売掛管理や受注オーダーなどの J.D. Edwards トランザクション処理では、参照上の整合性を確保するためにマスター・ビジネス関数を使用しています。この関数を使用して、既存の請求書に表示される顧客の住所録レコードを削除禁止にすることもできます。
Suppress Find （検索の抑制）	検索を抑制するために使用します。たとえば、フィルタ・ボタンと検索ボタンを使用できる、更新可能なグリッド・フォームでは、ユーザーが現在のレコードを更新し終えていない場合は、検索ボタンを使用できなくすることがあります。また、照会を実行するフォームでは、インデックスに一致する情報をユーザーが入力したかどうかを確認できます。
Suppress Update （更新の抑制）	更新を抑制するために使用します。入出力用フォームの標準フローを制御することができます。
Was Form Record Fetched （フォームレコードのフェッチ確認）	レコードが実際に取り出されたことを確認するために使用します。

グリッド・システム関数

次の表に、イベントに対するグリッド・システム関数を示します。

Clear Grid Buffer（グリッド・バッファのクリア）	バッファをクリアするために使用します。グリッド・バッファはグリッド・ローを処理するための一時的な格納域であるため、このシステム関数を使用して、バッファのすべてのカラムをクリアします。たとえば、計算で一時的な値を保持するためにグリッド・バッファを使用した場合は、計算の終了後に Clear Grid Buffer を使用してバッファをクリアすることができます。
Clear Grid Cell Error（グリッド・セル・エラーのクリア）	Clear Grid Buffer と使い方は同じです。
Clear QBE Column（QBE カラムのクリア）	Clear Grid Buffer と似ています。たとえば、購買の〈Find/Browse〉フォームの現在日などの値が QBE カラムに入力された場合に使用できます。ユーザーが検索を実行した後、別の日付を使用する必要がある場合は、QBE カラムをクリアできます。
Copy Grid Row to Grid Buffer（グリッド・ローのグリッド・バッファへのコピー）	Clear Grid Buffer を発行した後で、少数のグリッド・カラムのみを処理しなければならない場合に使用します。このシステム関数を使用すると、個々のカラムをコピーせずに、すべての GC フィールドを GB に移動できます。
Delete Grid Row（グリッド・ローの削除）	ランタイム・エンジンではなく開発者がグリッド・ローを操作している場合に使用できます。たとえば、マスター・ビジネス関数から Delete Doc ルーチンを使用しているときに使用します。
Disable Grid（グリッドの無効化）	グリッド全体を入力禁止にするために使用します。これは、個々のローを追加する前にヘッダーに詳細情報を入力したり、ヘッダーの情報をグリッドのデフォルト情報として使用する必要があるときに使用します。
Enable Grid（グリッドの有効化）	グリッドを入力可能にするために使用します。
Get Grid Row（グリッド・ローの取得）	たとえば、WHILE ループを通じてグリッドを再処理する必要がある場合など、グリッド・ローを個別に読み取る場合に使用します。
Get Max Grid Rows（グリッド・ローの最大数の取得）	個々のグリッド・ローを WHILE ループを通じてすべて再処理する必要がある場合に、処理するローの数を確認するために使用します。これは、WHILE ループの直前で使用します。
Get Selected Grid Row Count（選択済グリッド・ロー数の取得）	選択したローのロー番号を取得するために使用します。通常、その後の処理のためにローを変数として保存する必要があるときにのみ使用します。
Hide Grid Column（グリッド・カラムの非表示）	動的な処理のために使用します。

Hide Grid Row（グリッド・ローの非表示）	ロー全体をフォームで非表示にするために使用します。たとえば、グリッド付きのフォームを作成し、ローを要約する必要がある場合は、いくつかのローを合計して、総数を算出し、ローの総数を表示します。このシステム関数を使用する場合は、詳細情報のチェックボックスを設定し、グリッドを再取得する代わりにローを表示することができます。
Insert Grid Buffer Row（グリッド・バッファ・ローの挿入）	バッファ・スペースを操作するために使用します。このシステム関数を使用すると、GC が GB になります。
Insert Grid Row（グリッド・ローの挿入）	ローをグリッドに挿入するために使用します。通常、これらのローは、合計ローなどのカスタム（非データベース）ローです。
Set Grid Cell Error	Set Edit Control Error と使い方は同じです。
Set Grid Color（グリッド・カラーのセット）	Set Edit Control Color と使い方は同じです。
Set Grid Font（グリッド・フォントのセット）	Set Edit Control Fonts と使い方は同じです。
Set Grid Row Bitmap（グリッド・ロー・ビットマップのセット）	グリッドの外部に表示されるビットマップを操作するために使用します。これは、指定したロー見出しにビットマップを設定します。
Set QBE Column Compare Style（QBE カラム比較スタイルのセット）	QBE 行に比較演算子を割り当てるときに使用します。たとえば、日付カラムの QBE フィールドを「> January 1, 2005」となるように設定する必要がある場合は、QC WorkDate = "010105"と Set QBE Column Compare Style(FC Grid, GC WorkDate, <Greater Than>)を使用します。
Show Grid Column（グリッド・カラムの表示）	Show Edit Control と使い方は同じです。
Suppress Grid Line（グリッド行の抑制）	ローがグリッドの一部にならないようにする場合に使用します。たとえば、このシステム関数を Write Grid Line Before イベントで使用すると、行がグリッドに書き込まれなくなります。
Update Grid Buffer Row（グリッド・バッファ・ローの更新）	グリッド・バッファを割り当てた後は、このシステム関数を使用して GC=GB にします。

テレフォニー・システム関数

テレフォニー（電話機能）システム関数を使用すると、電話通信機能をアプリケーションに統合することができます。テレフォニー・システム関数は、Microsoft Windows Telephony Application Programming Interface（TAPI）に要求を伝達できます。たとえばこの機能は、顧客サポートを提供するカスタマー・サービス・センターで使用できます。問合わせが寄せられたときにアプリケーションを使用して発信者を識別し、その発信者に関する情報を自動的に取り出すことができます。たとえば、TAPI を使用すると次のことが行えます。

- 直接回線、PBX または音声応答装置（VRU）から着信コールに応答する。

  着信コールに応答するには、電話の呼出音が鳴ってからユーザーが受話器を取り上げるまで、および受話器を取り上げた直後の時点にイベント・ルールを追加します。

- コールを転送する。

  コールを転送するには、ユーザーがコールの転送を要求したかどうかを確認してから、コールの転送要求の直前、およびコールの転送完了通知をアプリケーションが受け取った直後にイベント・ルールを追加します。

- コールを保留状態にする。

  電話を保留状態にするには、コールが保留状態にされたという通知をアプリケーションが受け取った直後、および保留状態にあったコールにユーザーが対応した直後にイベント・ルールを追加します。ユーザーは、保留されているすべてのコールに関する情報を確認できなければなりません。

- コールを発信する。

  コールを発信するには、発信先の電話番号を J.D. Edwards テレフォニー・システム関数に受け渡す必要があります。テレフォニー・システム関数は、その後で TAPI を使用して、コールの発信に必要なステップを遂行します。コールを発信する直前、コールが応答された後、およびコールが終了した直後にイベント・ルールを追加します。

- コールを終了する。

  コールはいくつかの方法で終了できます。たとえば、サービス・スタッフまたは発信者が電話を切ったとき、スタッフが電話を転送したとき、または接続不良によりコールが切断されたときなどに終了できます。コールを適切に終了するには、コールの終了メッセージをアプリケーションが受け取る時点にイベント・ルールを追加します。

参照

- 各種システム関数については、オンラインの『Tools API Reference（ツール API リファレンス）』の「System Functions（システム関数）」

ビジネス関数のイベントへの関連付け

既存のビジネス関数をイベントに関連付けることができます。ビジネス関数には、次のコードが含まれます。

- 手動で生成する C コード（ソース言語 C）
- 〈Business Function Event Rule Design（ビジネス関数イベント・ルールの設計）〉を使用した場合に J.D. Edwards ソフトウェアが生成するコード（ソース言語 NER）

ビジネス関数の一般的な用途は次のとおりです。

- 参照整合性の確保（マスター・レコードが削除されたときの 2 次レコードの削除など）とルーチンの編集
- ランタイム・エンジンの負荷を過大にする可能性がある大規模で複雑な計算

► ビジネス関数を関連付けるには

1. 〈Event Rules Design〉で、イベントを選択します。
2. [f(B)]ボタンをクリックします。

   ビジネス関数に関する説明は、[Row(ロー)]メニューから[Attachments(添付)]を選択することで表示できます。

3. ビジネス関数を選択して[Select]をクリックします。
4. 使用可能なオブジェクトのリストから、ビジネス関数に渡すオブジェクトを選択します。
5. ビジネス関数にリテラルを割り当てるには、[Available Objects(使用可能なオブジェクト)]リスト・ボックスで[<Literal>]を選択します。
6. 値を 1 つ入力して[OK]をクリックします。

   範囲とリストは、ビジネス関数と併用できる有効なリテラルではありません。

7. [Value(値)]と[Data Items(データ項目)]間のデータ・フローの方向を指定して[OK]をクリックします。

   方向矢印をクリックすると、次の 4 つのオプションが切り替わります。

   - ! データは、ソースからターゲットに送られる。
   - z データは、ターゲットからソースに渡される。
   - $ データは、ソースとターゲットとの間を受け渡しされる。
   - データ・フローなし

   ビジネス関数のデータ構造体で、項目の方向がハードコーディングされている場合(たとえば、パラメータが入力パラメータ、出力パラメータまたは双方向パラメータとして既定されている場合など)は、その既定の方向がここで表示されます。項目の方向が重要でない場合は、ユーザーによって省かれたり、設定されることがあります。赤色表示される必須項目には、必ず入力します。ステータス・バーには、ターゲットへのフローの状態が表示されます。

8. このビジネス関数をトランザクション処理に組み入れるには、[Transaction]オプションで[Include(組込み)]をオンにします。

   このオプションは、修正/検査、見出し詳細、見出しなし詳細の各フォームから呼び出している場合のみ表示されます。

9. 非同期処理を有効にするには、[Asynchronously(非同期)]オプションをオンにします。

   このオプションは、Post Button Clicked イベントの発生時に、〈Fix/Inspect〉フォーム、〈Header Detail〉フォームまたは〈Headerless Detail〉フォームで[Cancel]または[OK]ボタンを押すと表示されます。

10. コメントを追加するには、次のボタンのいずれかをクリックします。
    - Business Function Notes(ビジネス関数の注記)
    - Structure Notes(構造体の注記)
    - Parameter Notes(パラメータの注記)
11. [OK]をクリックします。

J.D. Edwards 設計規約

ビジネス関数を組み込むときには、必ず方向を示す矢印を使用します。パラメータを使用しない場合は、↩記号を使用します。この記号は、次の機能も実行します。

- ビジネス・ビューが変更されたりワーク・フィールドが削除された場合など、パラメータが忘却されたり追加する必要があることを識別します。
- コードを読む者にとっては文書として役立ちます。
- 不要な場合にメモリを解放します。

数値は、世界各国で受け入れられるので、ビジネス関数から送り返されるイベント・ルール・フラグには、すべて数値を使用します。たとえば TRUE/FALSE の値を代入するには、T または Y と F または N を使用するかわりに、TRUE の場合は 1 を、FALSE の場合は 0 を使用します。

参照

- ビジネス関数については『開発ツール』ガイドの「ビジネス関数」

フォーム・インターコネクトの作成

フォーム・インターコネクトを使用すると、ソース・フォームからターゲット・フォームにデータを自動的に受け渡すことができます。

はじめる前に

- 受け取り側フォームのデータ構造体を定義し、値を受け渡すすべてのフィールドを含めます。

モーダル・フォーム・インターコネクションの作成

ほとんどのフォーム・インターコネクションはモーダルです。これは、フォームによって他のフォームへのフォーム・インターコネクトが実行されると、第 2 のフォームを終了しなければ第 1 のフォームに戻って操作できないことを意味します。

▶ モーダル・フォーム・インターコネクトを作成するには

1. 〈Event Rules Design〉で、イベントを選択します。
2. [Interconnect(インターコネクト)]ボタンをクリックします。
3. 〈Work with Applications〉で、イベント・ルールの接続先となるアプリケーションを選択します。

   〈Work with Forms(フォームの処理)〉に、選択したアプリケーションで使用可能なフォームが表示されます。
4. 接続するフォーム(ターゲット)を選択します。
5. 接続するフォームのバージョンを選択します。

   [Data Item]カラムに、ターゲット・フォームのデータ構造体に格納されているデータ項目が表示されます。ビジネス・ビューのプライマリ・テーブルのプライマリ・インデックス・キーがデータ構造体として自動的に設定されます。
6. [Available Objects]カラムで、受け渡すオブジェクトを選択します。

7. [>]ボタンを使用して、選択したオブジェクトを[Data Structure-Value]カラムに移動します。

   [Value]と[Data Items]間のデータ・フローの方向を指定します。

   データをフォーム間で受け渡さない場合は、[Direction(方向)]のすべての値を NULL に設定し、[OK]をクリックしてフォーム・インターコネクトを保存して終了します。

   方向矢印をクリックすると、次の 5 つのオプションが切り替わります。

   - データは、ソースからターゲットに送られる。
   - データは、ターゲットからソースに送られる。
   - データは、ソースからターゲットに送られ、ターゲットを終了すると、ソースに戻る。
   - ターゲットを終了すると、データはソースに戻る。
   - データ・フローなし

8. [Include in Transaction(トランザクションへの組込み)]オプションをオンにして、このインターコネクトをトランザクション処理に組み込みます。

   このオプションは、修正/検査、見出し詳細、見出しなし詳細の各フォームから呼び出している場合のみ表示されます。

9. コメントを追加するには、次のボタンのいずれかをクリックします。

   - Structure Notes
   - Parameter Notes

10. データ構造体を定義した後は、[OK]をクリックします。

    〈Event Rules Design〉に、フォーム・インターコネクトと次のステートメントが表示されます。

    Call (Application <name> Form <name>)

J.D. Edwards 設計規約

どのフォーム・タイプにも、次の規約が適用されます。

- データ構造体の最上部にあるプライマリ固有キー・フィールドのデフォルト位置はそのままにします。
- データ項目名と記述を変更し、フォーム間で渡される項目を記述します。

モードレス・フォーム・インターコネクションの作成

検索/表示フォームと 1 つ以上の修正/検査フォーム、または検索/表示フォームとトランザクション処理用フォーム(見出しフォームと見出しなし詳細フォーム)を結ぶモードレス・フォーム・インターコネクトを作成することができます。

モードレス処理では、一方のフォームを閉じては他方のフォームに戻るという逐次操作を行う代わりに、複数のフォームを同時に操作することができます。トランザクション・フォームを更新すると、データベースを再照会しなくても、更新内容が検索/表示フォームに表示されます。最初に検索/表示フォームにアクセスした後は、別の検索/表示フォームを含むあらゆるフォーム・タイプにアクセスできます。

モードレス・フォーム・インターコネクトは、呼び出されるたびに値をフォーム・データ構造体に渡します。呼出し側フォームで[Cancel]をクリックすると、呼び出されたフォームが破棄され、値が受け渡されません。呼び出されたフォームで[OK]をクリックすると、フォーム・データ構造体を介して呼出し側フォームに値が送り返されます。呼び出されたフォームはそのまま表示されます。

Dialog is Initialized イベントは、最初に呼び出されたときにしか発生しません。フォームが表示されるたびにイベント・ルールを実行する場合は、それらを Post Dialog is Initialized イベントに関連付ける必要があります。

更新モードのフォームがデータベースからレコードを取り出せないと、フォームのモードは追加モードに変更されます。[Close Form On Add(追加時にフォームを閉じる)]オプションがオンに設定されている場合、フォームは[OK]をクリックすると終了します。

検索/表示フォームを閉じると、すべてのモードレス・インターコネクト・レコードが閉じてからフォーム自体が閉じます。

モードレス・フォーム・インターコネクトに後続するイベント・ルールは、呼び出されたフォームに戻るまで待たずに直ちに実行されます。

モードレス処理のアプリケーションはメニューから実行します。〈Object Librarian(オブジェクト・ライブラリアン)〉からアプリケーションを実行すると、モードレス処理は機能しません。

▶ モードレス・フォーム・インターコネクトを構築するには

1. 使用する検索/表示フォーム(ソース)の〈Event Rules Design〉で、イベントを選択します。
2. [Form Interconnection(フォーム・インターコネクト)]ボタンをクリックします。
3. 〈Work with Applications〉で、イベント・ルールの接続先となるアプリケーションを選択します。

   〈Work with Forms〉に、選択したアプリケーションで使用可能なフォームが表示されます。
4. 接続先となる修正/検査フォーム(ターゲット)を選択します。

   〈Form Interconnect - Values(フォーム・インターコネクト - 渡す値)〉ウィンドウに、受入側フォームのデータ構造体が表示されます。
5. 接続先の修正/検査フォームの適切なバージョンを選択します。

   [Data Item]カラムに、ターゲット・フォームのデータ構造体に格納されているデータ項目が表示されます。ビジネス・ビューのプライマリ・テーブルのプライマリ固有インデックスのキーがデータ構造体として自動的に設定されます。
6. [Modeless(モードレス)]オプションをクリックします。
7. [Available Objects]カラムで、受け渡すオブジェクトを選択します。[>]ボタンを使用して、選択したオブジェクトを[Data Structure-Value(データ構造体-値)]カラムに移動します。
8. [Value]と[Data Items]間のデータ・フローの方向を指定します。

   データをフォーム間で受け渡さない場合は、[Direction]のすべての値を"↵"に設定し、[OK]をクリックして、フォーム・インターコネクトを保存して終了します。

   方向矢印をクリックすると、次の 4 つのオプションが切り替わります。

   - データは、ソースからターゲットに送られる。
   - データは、ターゲットからソースに送られる。

- データは、ソースからターゲットに送られ、ターゲットを終了すると、ソースに戻る。
- データ・フローなし

9. コメントを追加するには、次のボタンのいずれかをクリックします。
   - Structure Notes
   - Parameter Notes
10. データ構造体を定義した後は、[OK]をクリックします。

    〈Event Rules Design〉に、フォーム・インターコネクトと次のステートメントが表示されます。

    Call (Application <name> Form <name>)

はじめる前に

- ❑ 受取り側フォームのデータ構造体を設定し、値を受け渡すすべてのフィールドを含めます。

レポート・インターコネクトの作成

レポートを自動的に実行するには、レポート・インターコネクトを使用します。現行のイベント・ルールは、非同期処理が有効かどうかに基づき、処理を継続したり、レポートが完成するまで待機させることができます。同期レポート・インターコネクトを使用すると、第 2 プロセスを起動する第 1 プロセスは、第 2 プロセスが完了するまで待機してから動作を再開します。非同期処理を使用すると、第 1 プロセスが第 2 プロセスを起動しますが、そのまま動作を継続します。これら 2 つのプロセスは別々に実行されます。

▶ レポート・インターコネクトを作成するには

1. 〈Event Rules Design〉で、イベントを選択します。
2. [Report Interconnection（レポート・インターコネクト）]ボタンをクリックします。
3. 〈Work with Applications〉で、接続先のレポートを選択します。

   〈Work with Versions（バージョンの処理）〉で、選択したレポートのバージョンが表示されます。
4. 接続先のレポートのバージョンを選択します。
5. [Available Objects]カラムで、受け渡すオブジェクトを選択します。[>]ボタンを使用して、選択したオブジェクトを[Data Structure-Value]カラムに移動します。
6. [Value]と[Data Items]間のデータ・フローの方向を指定します。

   データをレポート間で受け渡さない場合は、[Direction]のすべての値を[↔]に設定し、[OK]をクリックして、レポート・インターコネクトを保存して終了します。

   方向矢印をクリックすると、次の 4 つのオプションが切り替わります。
   - データは、ソースからターゲットに送られる。
   - データは、ターゲットからソースに送られる。
   - ターゲットを終了すると、データはソースに戻る。
   - データ・フローなし

7. レポートを独立プロセスとして実行する場合は、次のオプションをクリックします。

   - Asynchronously

   レポートにデータを受け渡す場合は、このオプションをオンにしてください。

8. トランザクション処理にレポート・インターコネクトを組み込む場合は、次のオプションをクリックします。

   - Include in Transaction

9. コメントを追加するには、次のボタンのいずれかをクリックします。

   - Structure Notes
   - Parameter Notes

10. データ構造体を定義した後は、[OK]をクリックします。

    〈Event Rules Design〉に、レポート・インターコネクトと次のステートメントが表示されます。

    - Call (UBE <name> Version <name>)

割当ての作成

フィールドを固定値または演算式として定義するには、割当てを使用します。たとえば、ユーザーがフィールドから移動したときにデフォルト値を挿入する割当てを作成できます。また、ビジネス関数を作成して値を計算するのではなく、割当てを使用して値を計算することもできます。

割当てを作成するときは、次の操作を実行できます。

- 固定値を割り当てる。
- 演算式を作成して割り当てる。

式を作成するときは、単位やデータ・タイプが完全に一致するデータ項目のみを計算します。たとえば、異なる通貨または異なる 10 進法値を表す 10 進数は計算しないでください。これらを計算すると、データの整合性が損なわれることがあります。

▶ 値を割り当てるには

1. 〈Event Rules Design〉で、イベントを選択します。
2. ツールバーで[Assignment/Expression(割当て/数式)]をクリックします。
3. 〈Assignment(割当て)〉で、割当て値を受け取る割当て先オブジェクトを選択します。
4. 次のいずれかの方法で割当て元/オブジェクトのリテラル値を決めます。
   - 右側のカラムから[From Object(開始オブジェクト)]を選択し、次の論理ステートメントを作成する。[左側のカラム] = [右側のカラム]
   - テキスト入力ボックスにリテラル式(数値、テキストなど)を入力してリテラル・ステートメントを割り当てる。[左側のカラム] = [リテラル]
   - [f(X)]ボタンをクリックし、〈Expression Manager(計算式マネージャ)〉を使用して、複雑な計算式や数学関数を作成する。
5. [OK]をクリックします。

   〈Event Rules Design〉フォームに割当て式がテキスト形式で表示されます。

▶ 割当て式を作成するには

1. 〈Assignment(割当て)〉フォームで、[f(X)]ボタンをクリックします。
2. 〈Expression Manager〉の編集フィールドで式を作成します。

   式を入力するか、オブジェクト・リスト、電卓、および関数リストを使用します。次のボタンを使用して式を編集することができます。

Clear(クリア)	〈Assignment〉画面からすべてのデータがクリアされます。
Bksp(バックスペース)	直前の式または文字が削除されます。
Undo(やり直し)	直前の編集アクション(リストからオブジェクトや関数を挿入する操作を除く)が取り消されます。

3. 式を編集した後、[OK]をクリックします。

   〈Assignment〉画面では、スペルや構文のエラーがチェックされ、エラーがあればハイライトされます。エラーが検出されない場合、式は〈Event Rules Design〉に戻ります。

Expression Manager(計算式マネージャ)

〈Expression Manager〉には、操作に使用できる複数のセクションがあります。ここでは各セクションについて説明します。

割当ての表示

〈Expression Manager〉ウィンドウの最初のセクションには、割当て式の現行の状況が表示されます。[To Object(割当先オブジェクト)]には、式が割り当てられるオブジェクトが表示されます。[From Expression(開始の式)]にはブランクか、現在編集中の式が表示されます。

式編集フィールド

編集フィールドで式を作成して編集します。テキストは、キーボードから文字を入力するか、編集キーパッドを使用するか、または以下の式構成要素にポインタを合わせ、マウスをクリックして入力することができます。

- Objects(オブジェクト)
- Numbers and mathematical symbols(数値と演算記号)
- Advanced functions(上級関数)

編集フィールドには、あらゆる式を自由に入力できます。構文エラーが発生した場合は、エラーがハイライトされ、ステータス・バーに表示されます。

Available Information(使用可能な情報)

[Available Information(使用可能な情報)]リストには、式の作成/編集で使用できるすべてのオブジェクトが表示されます。このリスト・ボックスでオブジェクトを選択すると、そのデータ・タイプがステータス・バーに表示されます。

編集キーパッド

編集キーパッドを使用すると、キーボードを使用する代わりに式にデータを入力することができます。[Undo(元に戻す)]ボタンは、直前に行った編集操作の取消しが可能なときにのみ有効になります。このボタンでは、[Available Information]リストや[Advanced Functions(上級関数)]リストから挿入したオブジェクトを取り消すことはできません。

Advanced Functions(上級関数)

[Advanced Functions]リストには、式の作成/編集に使用できる関数が表示されます。関数の情報はステータス・バーに表示されます。最初のボックスには、関数に関する簡単な説明が表示され、2番目のボックスには関数の構文が表示されます。[Advanced Functions]は、次の関数から構成されます。

- Date functions(日付関数)
- General functions(汎用関数)
- Trigonometry functions(三角関数)
- Text functions(テキスト関数)

日付関数

関数	説 明
add_days(date, days)	指定された日数を日付に追加します。
add_months(date, months)	指定された月数を日付に追加します。
days_between(date1, date2)	date1 から date2 までの日数を返します。
months_between (date1, date2)	date1 から date2 までの月数を返します。
date_day(date)	指定された日付の日を返します。
date_month(date)	指定された日付の月を返します(1=1 月、2=2 月、...)
date_today( )	現在日を返します。
date_year(date)	指定された日付の年を返します(形式:19xx)
last_day(date)	指定された日付の月の最終日を返します(曜日ではない)。
next_day(date, day_of_week)	指定された日付の後の曜日で生じる次の日付を返します。

汎用関数

関数	説 明
abs	指定値の絶対値を返します。
cell	指定値を次の整数に切り上げます。
exp	e を底とする指定値の指数関数を計算します。
floor	指定値を次の整数に切り下げます。
mod	numeric1/numeric2 の余りを返します。
pow10	10 を指定値だけ累乗します。
pow	numeric1 の numeric2 乗を計算します。
round	numeric1 の numeric2 乗を丸めます。
sign	指定値の符号を返します。符号が負の場合は-1、それ以外の場合は 0 が返されます。
sqrt	指定値の正の平方根を計算します。

三角関数

関数	説 明
acos	指定値の逆余弦を計算します。
asin	指定値の逆正弦を計算します。
atan2	numeric1/numeric2 の逆正接を計算します。
cos	指定値の余弦を計算します。
cosh	指定値の双曲余弦を計算します。
log	指定値の自然対数を計算します。
log10	指定値の常用対数を計算します。
sin	指定値の正弦を計算します。
sinh	指定値の双曲正弦を計算します。
tan	指定値の正接を計算します。
tanh	指定値の双曲正接を計算します。

テキスト関数

関数	説 明	例
concat	string2 と string1 を連結します。	
indent	文字列を指定された文字数だけ字下げします。	indent("abcde",'5',5)="+++++abcde"
length	文字列の長さを返します。	length("axcvbnm") = 7
lower	指定された文字列を小文字に変換します。	lower("AbCdEfG") = "abcdefg"
lpad	文字列が指定の長さになるまで、文字列の前(左側)に文字を挿入します。	lpad("qwerty",'.',12) = "..... qwerty"
ltrim	先行出現の文字を文字列から削除します。	ltrim("aaaaaabcdef" , 'a') = "bcdef"
rpad	文字列が指定の長さになるまで、文字列の後ろ(右側)に文字を挿入します。	rpad("qwerty" , '.', 10) = "qwerty...."
rtrim	後続出現の文字を文字列から削除します。	rtrim("abcdef ",' ') = "abcdef"
substring	指定された位置から始まる文字列の部分を指定された文字数だけ抽出します。 **注:** 文字列の開始位置はゼロ位置と見なされます。そのため、10 番目の文字から始まる文字列を抽出する場合は、開始位置として 9 を指定してください。	substring ("SUNMONTUEWEDTHUFRIS AT", 9,3) = "WED"
upper	指定された文字列を大文字に変換します。	upper("qweRTY") = "QWERTY"

Table I/O(テーブル I/O)

〈Event Rules Design〉の[Table I/O(テーブル入出力)]ボタンを使用すると、ファイル入出力(I/O)を実行する命令を作成できます。そのため、C 言語を使用してビジネス関数を手作業でコーディングする必要がありません。〈Table I/O(テーブル入出力)〉では、イベント・ルールを通じてテーブルにアクセスすることができます。テーブル I/O を使用すると、次の操作を実行できます。

- データの検証
- レコードの取出し
- ファイル間のレコードの更新または削除
- レコードの追加

たとえば、〈Table I/O〉を使用すると、アプリケーションが使用しないテーブルの情報を表示することができます。

〈Log Viewer(ログ・ビューア)〉を使用すると、jdedebug.log のテーブル I/O SQL ステートメントを表示できます。そのためには、jde.ini ファイルで、デバッグを「File」に設定しておく必要があります。

使用可能な操作

〈Table I/O〉は、JDB API セットに書き込んで入出力を実行する下位レベルのビジネス関数に相当するイベント・ルールです。〈Table I/O〉を使用すると、次の操作を実行することができます。

FetchSingle 基本操作での[Select]と[Fetch(取出し)]を組み合わせます。[Select]にはインデックス付きカラムが使用され、[Fetch]にはインデックスなしカラムが使用されます。この操作では入出力用のテーブルを開きますが、それを閉じません。ハンドルを使用していないテーブルは、それを使用するフォームを閉じると、いずれも自動的に閉じます。

Insert 新規のローを挿入します。

Update 既存のローを更新します。(現在、ハンドルを使用する、または使用しないテーブルに)マッピングされているカラムのみが更新されます。テーブルおよびテーブルへのハンドルによって部分キー更新を実行できます。すべてのキーを指定しないと、いくつかのレコードが更新されることがあります。

Delete テーブルまたはビジネス・ビューのローを 1 つ以上削除します。

Open テーブルまたはビジネス・ビューを開きます。

Close テーブルまたはビジネス・ビューを閉じます。

Select 後続の FetchNext 操作のために 1 つ以上のローを選択します。

SelectAll 後続の FetchNext オペレーションのためにすべてのローを選択します。

FetchNext 指定したローを取り出します。FetchNext 操作を繰り返し実行するか、または FetchNext 操作のループによって、複数のレコードを取り出せます。

有効なマッピング演算子

特定のテーブル入出力オペレーションをマッピングするときには、次の演算子を使用することができます。

テーブル入出力操作	マッピング演算子
FetchSingle	インデックス・フィールド:=SelectAll
	非インデックス・フィールド:ターゲットのコピー
Insert	全フィールド:ソースのコピー
Update	インデックス・フィールド:= =
	非インデックス・フィールド:ソースのコピー

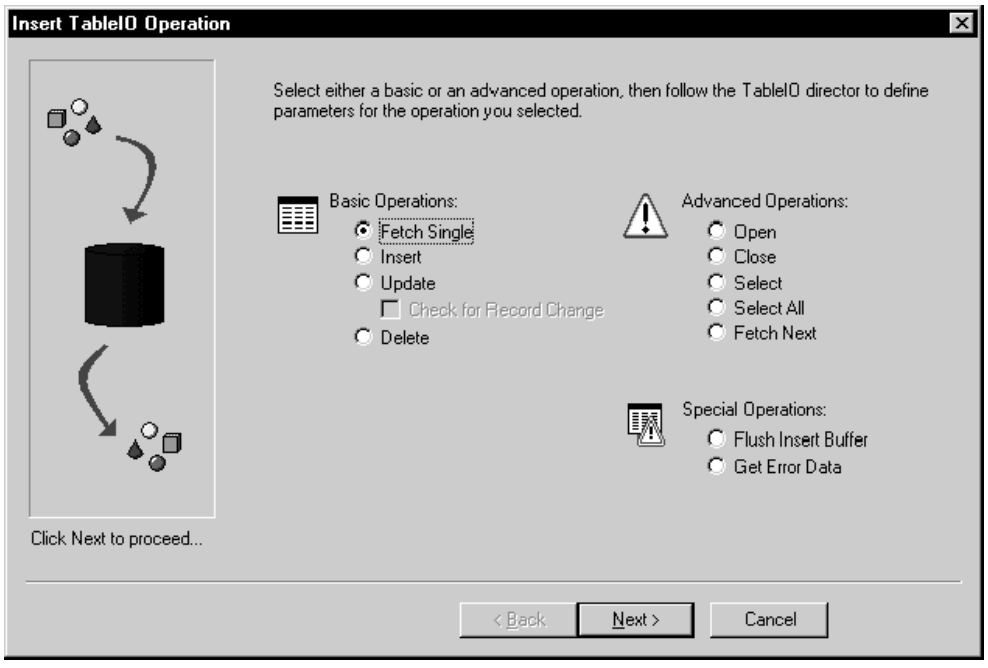
Delete	全フィールド: =
Open	該当なし
Close	該当なし
Select	全フィールド:SelectAll
SelectAll	該当なし

テーブル I/O(入出力制御)イベント・ルールの作成

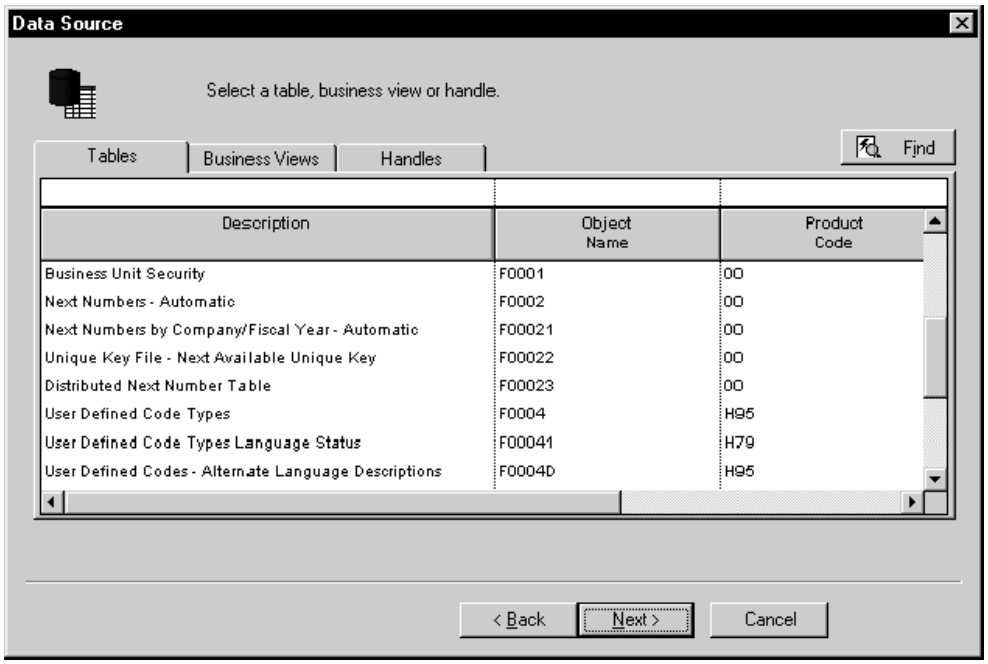
テーブル I/O イベント・ルールを使用すると、データベースへのアクセスをイベント・ルールでサポートすることができます。テーブル入出力、データ検証、およびレコードの取出しを実行するには、イベント・ルール・サポートが必要です。

▶ テーブル入出力イベント・ルールを作成するには

1. 〈Event Rules Design〉で、[Table I/O]ボタンをクリックします。



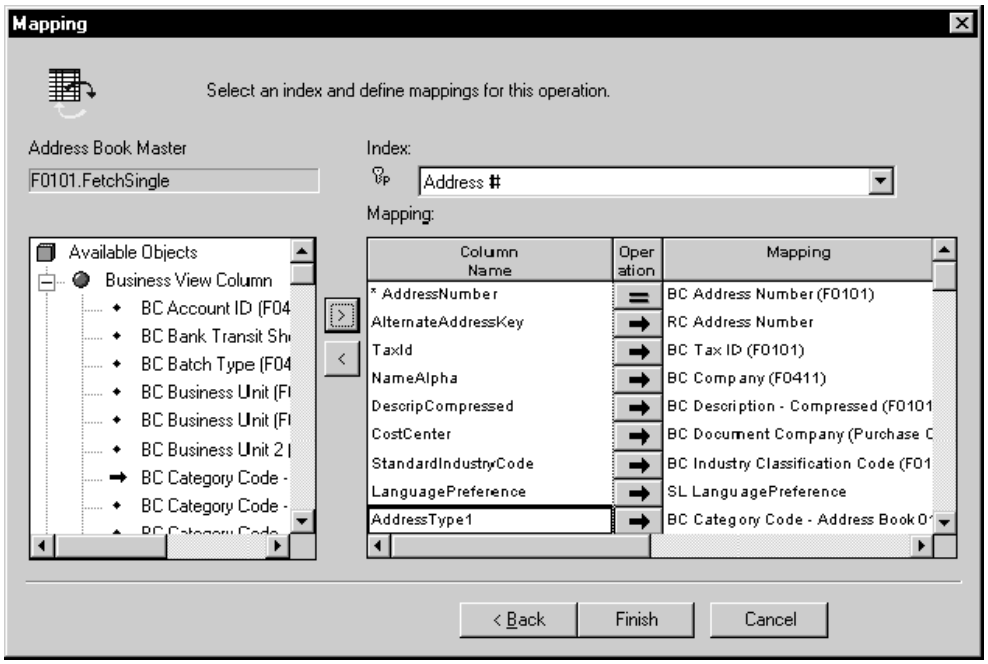
2. 〈Insert TableIO Operation(テーブル IO 操作の挿入)〉で、操作を選択して[Next(次へ)]をクリックします。



3. 〈Data Source(データ・ソース)〉で、入出力を実行するテーブル、ビジネス・ビュー、またはハンドルを選択して、[Next]をクリックします。

   〈Mapping(マッピング)〉フォームには、選択したテーブル・カラムにマップ可能なオブジェクトが表示されます。使用可能オブジェクトは、SELECT ステートメントで WHERE 節を作成するために使用されます。FETCH ステートメントでは、変数カラムの使用可能オブジェクトに値をマップします。たとえば、テーブルからレコードを 1 つ FETCH する場合は、各カラムをマップして SELECT ステートメントの WHERE 節を作成します。

   キー・カラムの横には、アスタリスク(*)が表示されます。



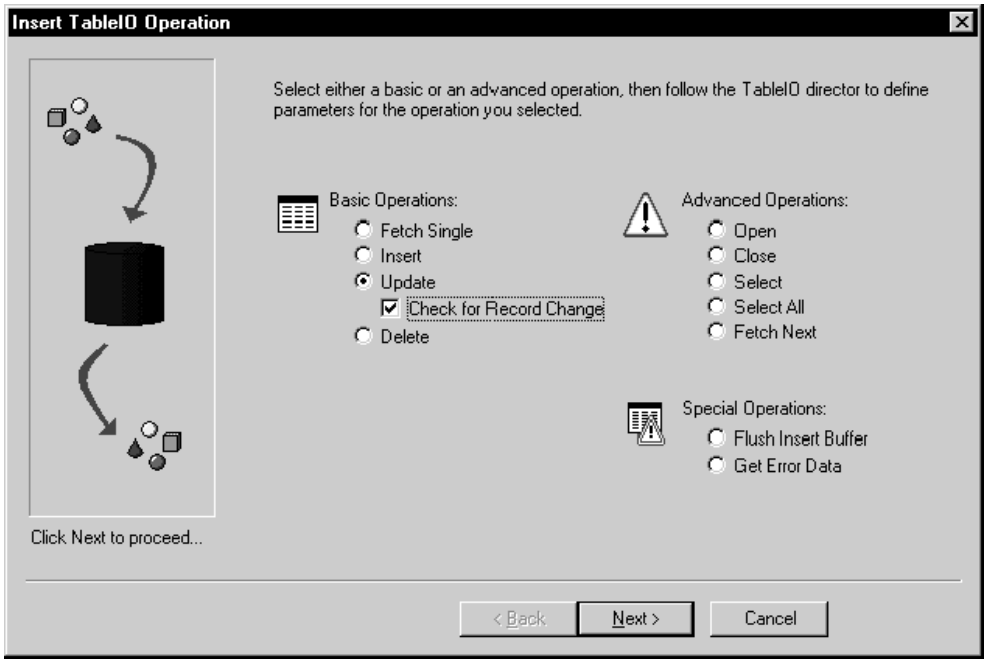
4. 使用するカラムを選択し、そのカラムにマップする使用可能オブジェクトをダブルクリックします。
5. 目的の演算子が表示されるまで[Operator(演算子)]ボタンをクリックします。

   次の選択用演算子を使用できます。デフォルト値は[Equal(等しい)]です。

   - Equal(等しい)
   - Not Equal(等しくない)
   - Less Than(より小さい)
   - Less Than or Equal To(以下)
   - Greater Than(より大きい)
   - Greater Than or Equal(以上)
   - Like(近似)
6. [Finish(終了)]をクリックして操作を保存し、〈Event Rules Design〉に戻ります。

レコード変更について

[Update(更新)]オプションを選択すると、[Check for Record Change(レコード変更のチェック)]も有効にすることができます。[Check for Record Change]オプションは、レコードが取り出されてから更新されるまでの間に変更されたかどうかを示します。



バッファ利用インサートについて

バッファ利用インサートを使用すると、1 つのデータベース・テーブルに多数(数百または数千)のレコードを挿入し、挿入エラーが発生した場合ユーザーからの即時フィードバックが必要でないときにパフォーマンスを向上することができます。バッファ利用インサートは、テーブル変換、テーブル I/O、バッチ処理およびビジネス関数で使用できます。これらは、対話型アプリケーションでは使用できません。バッファ利用インサートは、Oracle(V8 以上)、DB2/400 および SQL Server でのみ使用できます。JDEbase データベース・ミドルウェアは、データベース管理システムに一度に 1 バッファロードずつレコードを引き渡します。バッファリングを要求すると、データベース・レコードが個別に挿入され、バッファがいっぱいになると、自動的にフラッシュされます。

これは、JDEbase データベース・ミドルウェアがバッファをデータベース管理システムに引き渡すことを表します。バッファは明示的にフラッシュすることもできます。たとえば、トランザクションにコミットしたりテーブルやビューを閉じると、バッファは自動的にフラッシュします。ビジネス関数やテーブル変換エンジン、テーブル I/O でも、バッファのフラッシュを明示的に要求できます。

バッファ利用インサート・エラー・メッセージの概要

挿入に失敗しても即時フィードバックはないので、バッファ利用インサートを使用するかどうかは慎重に判断する必要があります。挿入に失敗すると、JDE ログにエラーが記録されます。そのため、バッファ利用インサートは、主にバッファ・アプリケーションで使用されます。

〈Table Conversion(テーブル変換)〉アプリケーションを使用している場合を除き、より詳細な情報をミドルウェアに要求して、詳細エラー・メッセージを取得する必要があります。〈Table Conversion〉では、テーブル変換エンジンがこのタスクを自動的に実行します。トレーシング機能をオンに設定すると、詳細エラー・メッセージを受信することができます。それ以外では、インサートが失敗したというエラーメッセージが表示されます。重複エラーが記録されないように、出力テーブルをクリアする必要があります。

▶ テーブル入出力にバッファ利用インサートを使用するには

1. 〈Event Rules Design〉で、[Table I/O]ボタンをクリックします。
2. 〈Insert TableIO Operation〉の[Advanced Operations(上級操作)]で、[Open(開く)]オプションを選択します。
3. [Next]をクリックします。
4. 〈Data Source(データ・ソース)〉で、使用するテーブルを選択し、[Advanced Options(上級オプション)]をクリックします。
5. 〈Advanced Options〉で、[Buffered Inserts(バッファ利用インサート)]を選択して[OK]をクリックします。

バッファ利用インサートへの特別な操作の使用

バッファ利用インサートを設定した後は、[Special Operations(特別な操作)]を使用して、バッファをフラッシュしたりエラー・メッセージを表示することができます。

▶ バッファ利用インサートに特別な操作を使用するには

1. 〈Event Rules(イベント・ルール)〉で、次の操作を実行するときに、それぞれのオプションをクリックします。

   - [Flush Insert Buffer(インサート・バッファのフラッシュ)]

     データの整合性を維持するには、インサート以外の操作を実行する前にインサートをフラッシュする必要があります。これを実行できない場合は、直近のインサートの結果が他の操作に反映されず、それらの操作が正しく行われないことがあります。

     [Flush Insert Buffer]オプションを特定のテーブルに対して使用するときは、テーブルを閉じる前にバッファをフラッシュして、エラー情報に依然としてアクセスできるようにする必要があります。インサートごとに[Get Error Data(エラーの取出し)]オプションを使用すると、この情報に確実にアクセスできます。

   - [Get Error Data(エラーの取出し)]

     [Get Error Data]オプションは、正しく挿入されなかったレコードでのエラーを取り出します。バッファをフラッシュしたり、別のインサートを開始する時点によっては、特定のインサートのエラー情報を上書きすることがあります。エラー情報が重要な場合は、次のインサートを開始する前にエラー情報を取り出してください。

特別なエラー処理を実行する必要がある場合は、テーブル入出力のインサートと[Flush Insert Buffer]オプションを実行するたびにエラー処理を設定する必要があります。エラー情報は、必ず次のテーブル操作を開始する前に取り出してください。

レポート用 SV_File_IO_Status の CO_ERROR_DETAILS_AVAILABLE に関する注:
このエラーは、〈Table I/O〉でインサートに失敗した場合の[Insert(インサート)]または[Flush Insert Buffer]操作中に設定します。リターン・コードに基づいて、このインサートの失敗理由に関する情報があるかどうかを判断することができます。インサートからのリターン・コードが CO_ERROR_DETAILS_AVAILABLE の場合は、Get Error Data テーブル I/O 操作を呼び出すことができます。Get Error Data 操作は、要求されたカラムすべてのインサートに使用された値を返します。次の例では、CO_ERROR_DETAILS_AVAILABLE の使用方法を示します。

```
F0101.Insert  //Attempt an insert
      SL AgingDaysAP1 -> TK Address Number
      RC Page - -> TK Tax ID
      SL TargetEnvironment -> TK Description - Compressed
If SV File_IO_Status is equal to CO ERROR_DETAILS_AVAILABLE
   F0101.Get Error Data  //Failed with errors so get errors
                         //Map values used in insert to the
                         //specified fields.
        SL AgingDaysAP1 <- TK Address Number
        RC TESTT <- TK Tax ID
        SL ReportName <- TK Description - Compressed
End If
```

ハンドルについて

J.D. Edwards ソフトウェアでは、テーブル入出力ターム・ハンドルは、一種のファイル・ポインタにあたります。このファイル・ポインタは、J.D. Edwards アプリケーションや UBE を、データベース・マネージャと交信するミドルウェアに接続します。ハンドルはデータベース・テーブルをポイントします。これらは、ミドルウェアのアドレスへの参照です。標準のファイル・ポインタと違って、ハンドルはその使用方法をある程度制御できます。ハンドルを使用すると次の操作を実行できます。これらは、ハンドルを使用しないテーブル入出力操作によっては実行できません。

- 1 つのテーブルやビジネス・ビューの複数のインスタンスを同時に開くことができます。
- サインオンしている環境以外の環境でテーブルやビジネス・ビューを開くことができます。この機能は、J.D. Edwards ソフトウェアのアップグレードを受け取ったり、別のシステムから J.D. Edwards ソフトウェアにデータを変換する必要があるときに特に役立ちます。
- フォームやイベント・ルール・ビジネス関数にハンドルを受け渡すと、テーブルやビジネス・ビューを何度も開けなくて済むようになります。

注:

ハンドルは、トランザクション処理で使用できません。

フォームやイベント・ルール・ビジネス関数にハンドルを受け渡す場合は、そのフォームやイベント・ルール・ビジネス関数のデータ構造体に、ハンドル・データ項目であるメンバーが含まれていなければなりません。フォーム・インターコネクトやビジネス関数の呼び出しでは、イベント・ルールからハンドル・データ構造体にハンドル値を割り当てます。このハンドルは、ハンドルを受け取るフォームやイベント・ルール・ビジネス関数で他のハンドルと同じように使用できます。

ハンドルは、暗黙の開閉ステートメントを使用できる他のテーブル入出力操作と違って、明示的に開閉する必要があります。ハンドルを他の操作で使用する場合は、事前に開いておく必要があります。[Open]以外の操作はいずれもテーブルやビジネス・ビューの場合と同じようにハンドルに対して機能します。ハンドルの使用を終了したら明示的に閉じます。ハンドルは、テーブルやビジネス・ビューの代わりにハンドルを選択した場合を除き、テーブルやビジネス・ビューと同じように閉じます。

ハンドルの使用

ハンドルを使用するには、次の操作を実行してください。

- データ辞書でハンドルを定義する。
- イベント・ルールでハンドル変数を作成する。
- ハンドルを明示的に開く。

ハンドル・データ項目を作成した後は、ハンドル変数を作成する必要があります。ハンドル変数は、他の変数と同じように作成します。ハンドル変数の作成に必要であれば、どのスコープでも使用できます。

ハンドル変数を作成したら、ハンドルを明示的に開きます。その後、必要なテーブル入出力を実行したら、ハンドルを明示的に閉じてください。

▶ ハンドルを使用するには

クラス・タイプとして Handle、データ・タイプとして 7 を使用します。ハンドルのエイリアスには、そのテーブルと同じ名前を設定してください。

1. 〈Data Item Specifications〉で、ハンドル・データ項目を作成するには、[Edit Rule]タブをクリックして、ハンドルのテーブルまたはビジネス・ビュー名を入力し、このデータ辞書項目を保存します。

   データ辞書項目名には、最大 8 文字まで使用でき、次の形式にします。

   HFxxxxxx
   各文字列は次の意味を表します。

   HF = テーブル入出力データ項目を示します。

   xxxxxx = テーブル名で使用されているシステム・コードとグループ・タイプを示します。

   たとえば、テーブル F4211 のテーブル入出力データ項目名は HF4211 になります。

ハンドル・データ項目はデータ構造体でも作成できます。

2. 〈Event Rules Design〉で、[Table I/O]ボタンをクリックします。
3. [Advanced Operations(上級操作)]から[Open]をクリックします。
4. [Next]をクリックします。
5. 〈Data Source〉で、[Handles]タブをクリックします。
6. 開くハンドルを選択して、[Next]をクリックします。
7. テーブルを開く環境名を含む変数を選択します。

   テーブルをログイン環境で開く場合は、システム値の SL LoginEnvironment を選択します。〈Table I/O〉を〈Table Conversion〉で使用する場合は、〈Table Conversion〉のソース環境とターゲット環境のシステム値も存在します。

8. [Finish]をクリックします。
9. 〈Data Source〉で、[Handles]タブをクリックします。
10. 閉じるハンドルを選択して、[Finish]をクリックします。

参照

- データ項目については、『開発ツール』ガイドの「データ辞書」

テーブル・イベント・ルール

テーブルに対するアクション発生時に、自動的に実行されるデータベース・トリガー(またはプログラム)を組み込むには、テーブル・イベント・ルールを使用します。テーブルに対するアクションはイベントといいます。J.D. Edwards データベース・トリガーを作成するときは、最初に、データベース・トリガーを起動するイベントを決める必要があります。次に、〈Event Rules Design〉を使用して、データベース・トリガーを作成します。

テーブル・イベント・ルールでは、テーブル・レベルでの埋め込みロジックを提供します。テーブル・イベント・ルールには、それぞれに固有のロケーション、イベント、システム関数があります。テーブル・イベント・ルールを使用すると、テーブルに対する変更やイベントが、呼出し側アプリケーションにもユーザーにも通知されません。また、テーブル・イベント・ルールでは、フォーム・インターコネクトもレポート・インターコネクトも使用できません。

テーブル・イベント・ルールは、データの整合性を確保するために使用できます。たとえば、住所録のレコードを削除するときは、電話番号やカテゴリ・コードなど、関連するレコードをすべて削除することがあります。また通貨に対しても使用できます。Currency Conversion is On イベント・ルールは、テーブル・イベント・ルールの通貨情報を処理します。

次のイベントには、テーブルごとにイベント・ルールを関連付けることができます。

- After Record is Deleted(レコードが削除された後)
- After Record is Fetched(レコードが取り込まれた後)
- After Record is Inserted(レコードが挿入された後)
- After Record is Updated(レコードが更新された後)
- Before Record is Deleted(レコードが削除される前)

- Before Record is Fetched(レコードが取り込まれる前)
- Before Record is Inserted(レコードが挿入される前)
- Before Record is Updated(レコードが更新される前)
- Currency Conversion is On(通貨処理がオン)

参照

- 通貨換算については『開発ツール』ガイドの「通貨」
- 典型的な用途や処理順序など API については、J.D. Edwards Knowledge Garden の『ERP 8.0/9.0 Tools APIs Reference』

テーブル・イベント・ルールの作成

テーブル・イベント・ルールを作成するには、次の処理を実行する必要があります。

- 〈Event Rules Design〉でデータベース・トリガーを作成する。
- J.D. Edwards データベース・トリガーを C 言語コードとして生成する。

▶ テーブル・イベント・ルールを作成するには

1. 〈Object Management Workbench〉で、イベント・ルールを関連付けるテーブルをチェックアウトして[Design]をクリックします。
2. 〈Table Design(テーブル設計)〉で、[Design Tools(設計ツール)]タブを選択して[Start Table Trigger Design Aid(テーブル・トリガー設計ツールの起動)]をクリックします。

   〈Event Rules Design〉ツールが起動されます。このツールから、テーブルのどのイベントにもイベント・ルールを関連付ける(組込む)ことができます。
3. [Events]リスト・ボックスからイベントを選択します。
4. 次のイベント・ルール用ボタンを 1 つクリックします。

ビジネス関数 既存のビジネス関数を関連付けます。

システム関数 既存の J.D. Edwards システム関数を関連付けます。

If/While IF/WHILE 条件文を作成します。

Assign 割当てまたは複雑な式を作成します。

オフの場合 ELSE 節を挿入します。ELSE 節は、IF から ENDIF までの範囲でのみ有効です。

Variables アプリケーション固有の目的に合ったデータ辞書特性が割り当てられていても、データ辞書には常駐しないプログラマ定義のフィールドを作成します。

Table I/O データベースへのアクセスをイベント・ルールでサポートできるようにします。テーブル入出力、データの検査、レコードの取出しを実行します。

データ構造体を作成してテーブル・イベント・ルール(TER)関数に関連付ける必要はありません。テーブル自体が、テーブル・イベント・ルール関数に渡されるデータ構造体です。

5. 〈Event Rules Design〉で、[Save]をクリックしてイベント・ルール・スペックを保存し、[Close（閉じる）]をクリックして〈Table Design〉に戻ります。
6. 〈Table Design〉で新しいテーブルを作成する場合は、[Table Operations（テーブル操作）]タブを選択して[Generate Table（テーブルの生成）]をクリックします。

   注意:
   この手順を実行するとデータがすべてクリアされてしまうため、既存のテーブルに対して実行しないでください。この手順は、新規のテーブルに対してのみ実行してください。

7. 〈Generate Table（テーブルの生成）〉で、次のフィールドに入力して[OK]をクリックします。
   - Data Source
   - Password
8. 〈Table Design〉で、[Design Tools]タブを選択して[Build Table Triggers（テーブル・トリガーの構築）]をクリックします。

   [Build Triggers（ビルド・トリガー）]オプションによって、次のステップが実行されます。

   - イベント・ルールが C 言語ソース・コードに変換されます。これにより、ファイル OBNM.c と OBNM.hxx（OBNM =オブジェクト名）が作成されます。ソース・ファイルには、テーブル・イベント・ルールのイベントごとに関数が 1 つずつ保管されます。
   - 生成されたコードをコンパイルするための makefile が作成されます。
   - make file が実行され、新規の関数がコンパイルされて、JDBTRIG.DLL に追加されます。これは、テーブル・イベント・ルール関数を含む統合 DLL です。
9. 作成ログを表示するには、[Generate Header File（ヘッダー・ファイルの生成）]をクリックし、生成されたファイルを開きます。

以上で、テーブル・イベント・ルールの作成は完了です。新規に作成/変更されたテーブル・イベント関数は、対応するイベントがテーブルに対して発生すると、常にデータベース API から呼び出すことができます。

動的一時変更の作成

アプリケーションには、一般フォーム・コントロールやグリッド・カラムのコントロール・プロパティ（ローやグリッドの記述、ビジュアル・アシスト、編集ルールなど）が、ユーザーが特定の情報を入力したり、特定のデータがロードされるまでわからないものがあります。このような場合は、実行時にコントロール・プロパティを動的に一時変更する必要があります。たとえば、アプリケーションのフィールドにスタティック・テキストの代わりに終了日を表示させたり、ユーザーがビジュアル・アシストのボタンをクリックしたときにビジュアル・アシスト検索フォームを変更しなければならないことがあります。

データ辞書プロパティを実行時に一時変更するには、システム関数を使用することができます。システム関数の Set Data Dictionary Item を使用すると、次の操作を実行できます。

- データ辞書項目であるフォーム・コントロールとグリッド・カラムを一時変更する。
- すべてのデータ辞書プロパティが変更されるように、データ辞書項目を全面的に変更する。
- 以前の項目とタイプが異なるデータ項目を新規に作成する。

  文字列変数や固定文字列のデータ項目名を使用することができます。

Set Data Dictionary Overrides システム関数の使用対象は次のとおりです。

- すべてのフォーム・コントロールとグリッド・カラム
- 特定のデータ辞書プロパティの変更
- 変更されていないデータ辞書項目

次のシステム関数は、実行時にテキストを一時変更するために使用できます。

- Set Control Text
- Set Grid Column Heading
- Set Form Title

  これらのシステム関数は通常、テキスト変数と併用します。〈Parent/Child(親/子)〉フォームでは、グリッド・カラム用のシステム関数を使用することもできます。グリッド・カラム一時変更はすべてのグリッド・ローに適用され、既存の機能は変わりません。

ビジュアル・アシストを一時変更するには、次のイベントを使用することができます。

- Visual Assist Button Clicked
- Post Visual Assist Button Clicked

システム関数の Suppress Default Visual Assist を使用しても、ビジュアル・アシストを一時変更し、次のビジュアル・アシスト・フォームに変更を適用することができます。デフォルトのビジュアル・アシスト・フォームを抑制すると、フォーム・インターコネクトを使って、検索/選択フォームの代わりに別のフォームを呼び出すことができます。

非同期プロセスの処理

スレッドとは、メインの実行パスとは別個の実行パスです。スレッドを作成する操作は、メイン・プログラムがそのまま実行し続けることを除けば、関数の呼出しと同じです。これは、2 つの実行ポイントがコードを通じて進行することを意味します。異なるスレッドが同じメモリ領域を共有し、オペレーティング・システムがそれらにプロセッサ時間を割り当てます。

スレッドを使用すると、選択した J.D. Edwards イベントやビジネス関数をバックグラウンドで処理する間に、ユーザーは引続きアプリケーションを操作することができます。

スレッド処理には、次のようなパフォーマンス上の利点があります。

- グリッドおよび編集コントロールの処理の向上
- 一定のイベントがバックグラウンドで処理されるので、システムが終了するのをユーザーが待たずに済む。
- イベント処理の柔軟性の向上
- マルチタスク処理の実現
- データ入力などの対話型タスクの継続
- カーソル移動の高速化

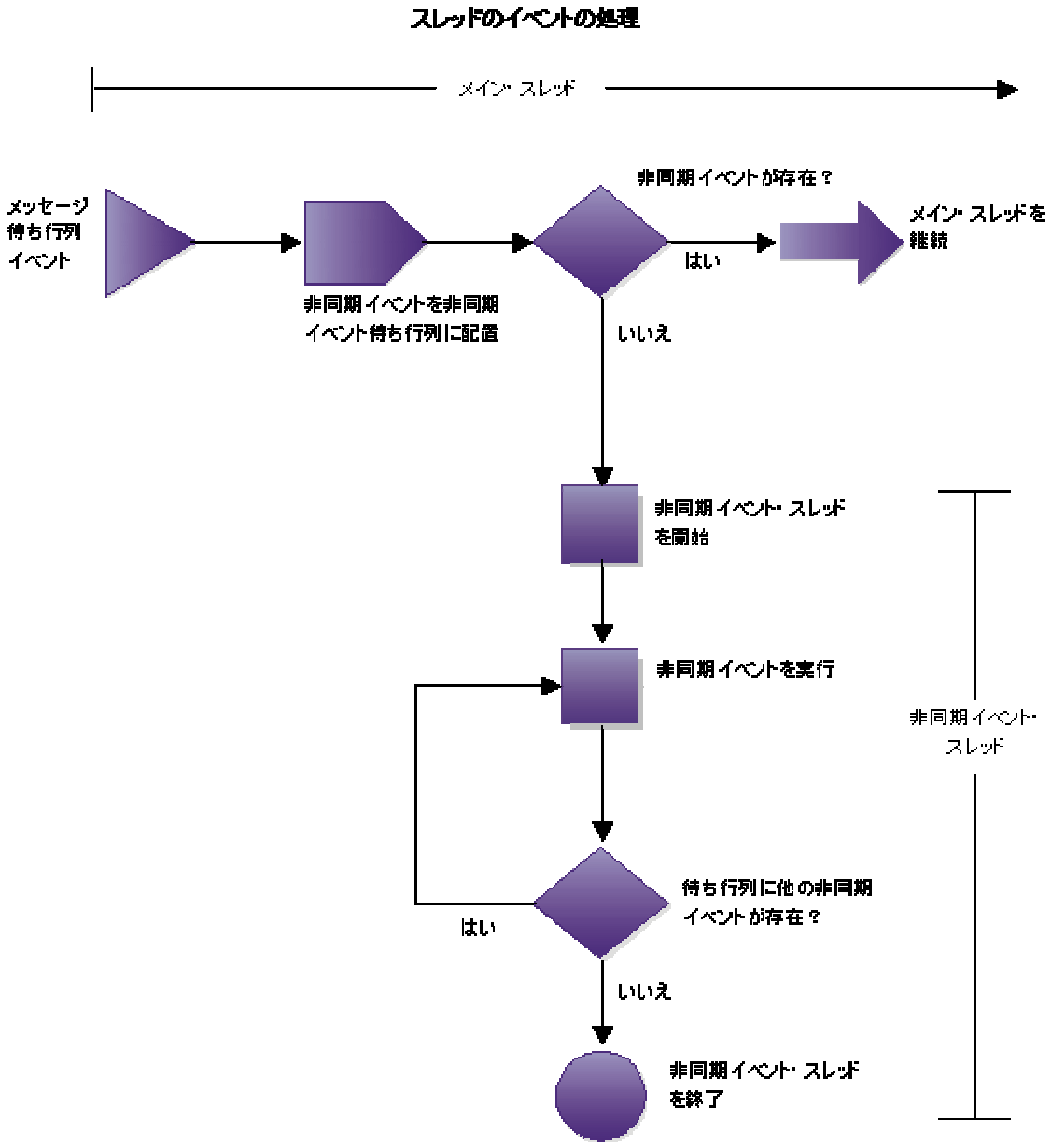
スレッドを使用するとアイドル時間を利用してバックグラウンド処理を実行できるため、パフォーマンスを大幅に改善できますが、単一スレッド・アプリケーションには存在しなかった問題が発生する可能性も生じます。

たとえば、メインの実行パスで処理されるイベントと、スレッドで処理されるイベントは、同時に実行することができます。2 つのイベントが同じデータ部分への同時アクセスを試みると、異常な結果が生じる可能性があります。

非同期処理はフォーム・インターコネクトをサポートしません。非同期処理はバックグラウンド処理で使用し、ユーザーは非同期スレッドを直接操作しないようにしてください。たとえば、Row is Exited and Changed – Asynch などの非同期処理を使用する場合は、そのワーカー・スレッドでフォームを開かないでください。そうしないと、バックグラウンドで処理すべきプロセスをユーザーに操作させることになります。

非同期イベント

次の図は、スレッドのイベントがどのように処理されるかを示しています。



一度に処理される非同期イベントは 1 つだけです。固有の非同期イベントの複数インスタンスが処理待ちになることがあります。メイン・スレッドは、非同期イベントが待ち行列に存在する場合も存在しない場合も継続します。上の図では、非同期イベントの有無により、2 次スレッドを開始するかどうかが決まります。このイベントはメイン・スレッドを変更しません。

[OK]または[Cancel]を処理する前に、すべての非同期イベント処理を完了する必要があります。

次のイベントをスレッド上で実行できます。

- Control is Exited and Changed – Asynch
- Row is Exited and Changed – Asynch
- Column is Exited and Changed – Asynch

Control is Exited and Changed – Asynch

あるコントロールからカーソルが移動すると、次の 3 つのイベントが順番どおりに処理されます。

- Control is Exited
- Control is Exited and Changed – Inline
- Control is Exited and Changed – Asynch

最初の 2 つのイベントが処理されなければ、ユーザーは操作を続行できません。Control is Exited and Changed – Asynch イベントはスレッドで実行されます。イベントの実行中も引き続きデータを入力することができます。

Row is Exited and Changed – Asynch

あるグリッド・ローからカーソルが移動すると、次の 3 つのイベントが順番どおりに処理されます。

- Row is Exited
- Row is Exited and Changed – Inline
- Row is Exited and Changed – Asynch

Row is Exited イベントは、どのグリッド上でも使用することができます。他の 2 つのイベントは、更新グリッド（見出し詳細フォーム、見出しなし詳細フォーム）でのみ使用できます。

最初の 2 つのイベントが処理されなければ、ユーザーは操作を続行できません。Row is Exited and Changed – Asynch イベントはスレッドで実行されます。イベントの実行中も引き続きデータを入力することができます。このイベントが特定のローについて処理中であることを示す砂時計が、ロー見出しに表示されます。

Column is Exited and Changed – Asynch

あるグリッド・カラムからカーソルが移動すると、次の 3 つのイベントが順番どおりに処理されます。

- Column is Exited
- Column is Exited and Changed – Inline
- Column is Exited and Changed – Asynch

これらのイベントは、更新グリッドに対してのみ有効です（見出し詳細フォームと見出しなし詳細フォーム）。

最初の 2 つのイベントが処理されなければ、ユーザーは操作を続行できません。Column is Exited and Changed – Async イベントはスレッドで実行されます。イベントの処理中も引き続きデータを入力することができます。

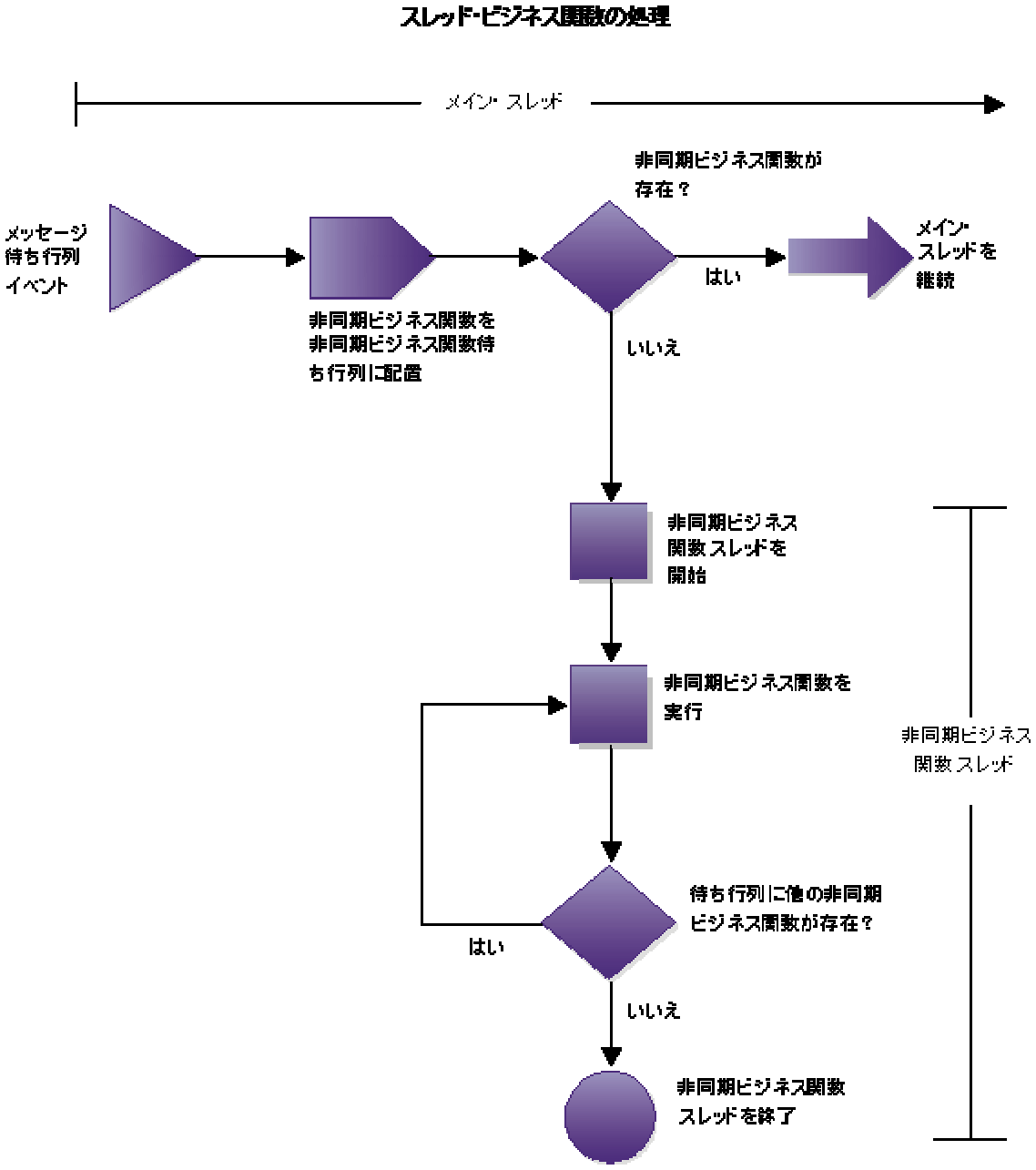
非同期ビジネス関数

[OK]および[Cancel]の処理時に動作するビジネス関数は、多くの場合、パフォーマンスに悪影響を及ぼします。このようなビジネス関数は、できるだけスレッドで実行してください。〈Event Rules Design〉は、ビジネス関数パラメータを選択するフォームに非同期オプションを設定します。このオプションは、OK Post Button Clicked、Cancel Post Button Clicked、および Close Post Button Clicked 以外のすべてのイベントに対して無効です。このオプションをオンに設定すると、ビジネス関数がスレッドで処理されます。このオプションをオンにすると、パラメータをパスできるか、または一切パスできません。フォームが End Dialog イベントに到達しているために開かない場合があるので、ビジネス関数からはパラメータを受け渡しできません。

非同期ビジネス関数でエラーを設定しようとしても無視されます。非同期ビジネス関数はフォームと無関係なので、エラーを設定できません。ただし、ランタイム・エンジンは、jdeCallObject からのリターン・コードを使って、非同期ビジネス関数が失敗したかどうかを確認できます。エラーが発生した場合には、メッセージ・ボックスが表示されます。メッセージ・ボックスにより、エラーが発生したときに処理を中止できます。エラーの詳細は、ログで確認できます。

非同期ビジネス関数は、フォームが閉じるときにさまざまなデータベース呼出しを行うフォームで役立ちます。たとえば、JdeCache 呼出しやワークテーブルを使用してレコードを一時的に保管し、[OK]ボタンがクリックされた時点で一時レコードを実際のファイルに書き込むフォームは、スレッドを使用するビジネス関数に適しています。

次の図は、スレッドを使用するビジネス関数の処理を示しています。



非同期ビジネス関数は、一度に 1 つしか処理されません。メイン・スレッドは、非同期イベントが待ち行列に存在するかどうかに関わらず継続します。上の図では、非同期イベントの有無により、2 次スレッドを開始するかどうかが決まります。このイベントは、メイン・スレッドに影響を与えません。

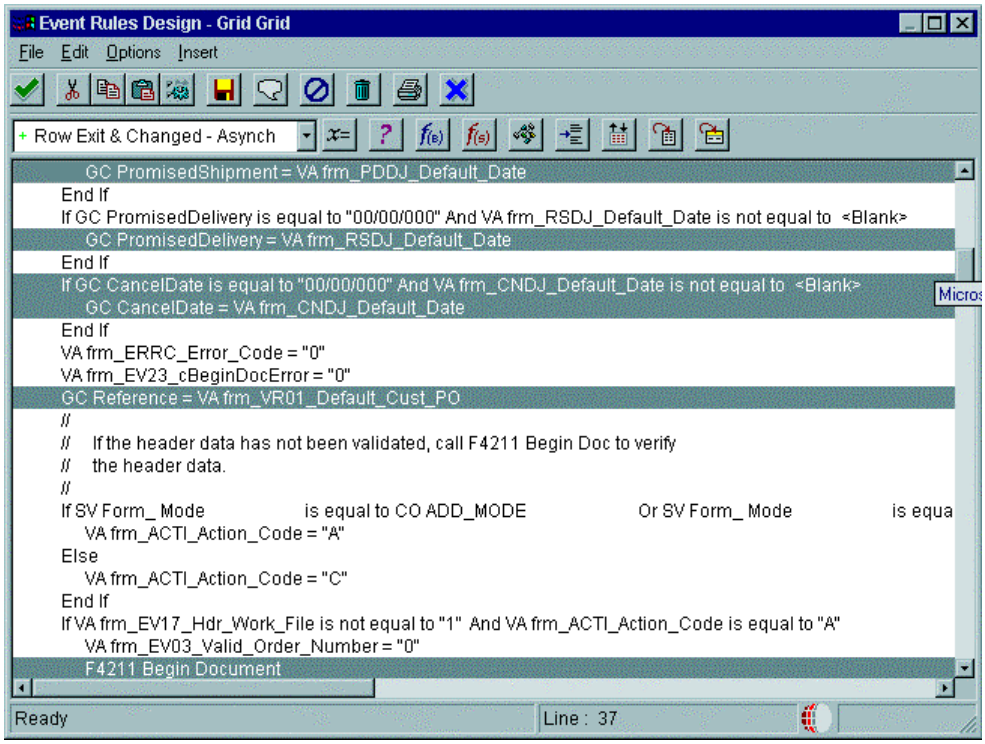
J.D. Edwards エクスプローラを閉じる前に、非同期ビジネス関数の処理を完了する必要があります。非同期ビジネス関数が依然として動作している場合は、メッセージが表示され、J.D. Edwards エクスプローラは終了しません。

例：非同期イベントの処理

以下は、Row is exited and Changed – Asynch イベントに関連付けられている関数の一例です。

〈Sales Order Detail（受注明細）フォーム(P4210)では、グリッドの Row is exited and Changed – Asynch イベントに次のようなロジックが関連付けられています。

ハイライトされている最初の 5 行では、データベース更新用の一定のテーブル・カラムにデフォルト情報が入力されます。ハイライトされている最後の行では、F4211Begin Document（マスター・ビジネス関数）が呼び出されます。



BrowsER

BrowsER は、イベント・ルールを表示し、対話型/バッチ・アプリケーションのレイアウトを設計するために使用することができます。BrowsER を使用すると、設計ツールを使用して別の作業を行わなくても 1 つ以上のイベント・ルールを有効、無効にすることができます。この機能は、個々のイベント・ルールをデバッグするために使用します。

BrowsER では、対話型アプリケーションの各フォームや、バッチ・アプリケーションの各セクションの構造を表示します。フォームやセクションは、各セクションのイベントおよびイベント・ルールと共に階層構造で表示されます。

BrowsER の処理

BrowsER を使用すると、イベント・ルールを有効/無効にして、アプリケーションに対する操作の影響を確認できます。

▶ BrowsER を処理するには

1. 〈Object Management Workbench〉で、プロジェクトを選択して、[Design]をクリックします。
2. 〈Interactive Application Design（対話型アプリケーション設計）〉で、[Design Tools]タブを選択して[Browse Event Rules（イベント・ルールの表示）]をクリックします。
3. 〈Browsing（表示）〉フォームで、プラス（+）ボタン、またはマイナス（-）ボタンをクリックすると、対話型フォームまたはバッチ・レポートのセクションのイベントの階層表示を展開または圧縮表示できます。

   各イベント・ルールは、それぞれが関連付けられているイベントの下と、イベント・ルール・ロジックが含まれているコントロールの横に表示されます。イベント・ルールがコントロールの横に表示されない場合は、イベント・ルール・ロジックがそのコントロールに存在しません。

4. デバッグを行うためにイベント・ルールを無効にするには、そのイベント・ルール上でダブルクリックします。
5. 無効にしたイベント・ルールをダブルクリックすると、有効になります。

   〈BrowsER（ブラウズ ER）〉フォームからはイベント・ルールを印刷または修正することはできません。

BrowsER オプションの処理

次のいずれかの BrowsER オプションを選択すると、コードの表示や検索を容易に行うことができます。

- Expand Tree（ツリーの展開）
- Expand Node
- Show Object IDs
- Hide Object with no ER（イベント・ルールなしのオブジェクトを非表示）
- Filter ER Records
- Search

▶ BrowsER オプションを処理するには

〈Browsing（参照）〉フォームで、マウスの右ボタンを押したまま、表示されるメニューからオプションを選択します。

Expand Node

[Expand Node(ノードの展開)]を使用すると、ハイライトされているノードを展開し、そのノードの各コード行を表示することができます。

Show Object IDs

[Show Object IDs(オブジェクト ID の表示)]は、C 言語コードで使用します。このオプションは、各オブジェクト固有の ID を表示し、コード生成プロセスでエラーを生成するオブジェクトの識別に役立ちます。

Filter ER Records

[Filter ER Records(ER レコードのフィルタ)]を使用すると、次のような特定のコード・タイプのイベント・ルール・レコードを検索できます。

- Assignments(割当て)
- Business functions(ビジネス関数)
- Criterion(基準)
- Comments(コメント)
- Form Interconnects(フォーム・インターコネクト)
- Option(オプション)
- System functions(システム関数)

Search

〈Search(検索)〉を選択すると、特定のコード行を検索することができます。この機能は、特定の文字列や変数を検索する際に使用できます。

Visual ER Compare の使用

〈Visual ER Compare〉は、ローカル・ワークステーションの ER を、定義済みのパス・コードのセントラル・オブジェクト・データ・ソースの ER、TAM スペックまたは ESU バックアップと比較できるプログラムです。たとえば、アプリケーションの ER を変更し、その変更をサーバー・アプリケーションの ER と比較する場合は、Visual ER Compare を使用できます。

Visual ER Compare は、行単位での画面上での比較を実現します。ユーティリティのターゲット ER (ローカル・バージョン)は、行をソース ER から直接移動して変更できます。行を削除したり、無効にすることも可能です。画面上で比較できる他に、変更を詳しく示すレポートを印刷することもできます。

Visual ER Compare の起動

以下のタスクは、組み込み可能な ER を付帯するオブジェクト(アプリケーション、UBE、テーブル、ビジネス関数)にのみ適用してください。

► Visual ER Compare を起動するには

〈クロス・アプリケーション開発ツール〉メニュー(GH902)から〈オブジェクト管理ワークベンチ〉プログラム(P98220)を選択します。

1. 〈Object Management Workbench〉で、オブジェクトをチェックアウトします。
2. チェックアウトしたオブジェクトを選択し、中央カラムの[Design(設計)]ボタンをクリックします。
3. 〈Design(設計)〉フォームで、[Design Tools]タブをクリックします。
4. [Visual ER Merge(ビジュアル ER マージ)]をクリックします。
5. [Select the Location of Source Specifications(ソース・スペックの保管場所を選択)]で、次のいずれかのオプションをクリックします。
   - Central Objects Path Code(セントラル・オブジェクト・パス・コード)
   - Remote Specifications Location(リモート・スペックの位置)
   - Software Update Backup(ソフトウェア・アップデートのバックアップ)

Visual ER Compare インターフェイスについて

Visual ER Compare を起動すると、〈Visual ER Compare(ビジュアル ER の比較)〉フォームが表示されます。このフォームでは、ER のツリー構造のメニューが左側に表示されます。残りの領域には、ソース ER(中央パネル)とターゲット ER(右パネル)に表示されます。ターゲット ER はローカル ER です。

[ER]メニューからは、ソースとターゲットの両方に存在していても異なるルールと一方にしか存在しないルールが色違いのフォントで表示されます。1 つまたは複数の子が変更された場合は、親ノードに変更が表示されます。ER 領域では、異なる行がハイライトされます。行が一方の側に追加または削除されると、もう一方の側にブランク行が表示されます。感嘆符は行が無効であることを示します。内容が変更されたり、追加または削除されたルールは、色違いのフォントで表示されます。表示色は、[View(表示)]メニューから[User Options(ユーザー・オプション)]を選んで[Set Colors(色の設定)]を選択して変更できます。

Visual ER Compare はアルゴリズムを使って行を比較し、変更が行われたかどうかを確認します。ターゲット行の一致の割合がソース行と異なる場合、その行は不一致とマークされます。比較精度は、[View]メニューから[User Options] – [Comparison Factors(比較要素)]を選択して変更できます。無効にされた ER 行を比較に取り込むには、[Disable Partial Matching for Disabled ER(無効にされた ER の無効な部分一致)]をクリックします。行の変更を示すのに必要な不一致率を変更するには、[Partial Match Ratio(部分一致率)]フィールドに数値を入力します。デフォルト値の 0.50 は、ターゲット行とソース行の不一致率が 50%以上でないと、ターゲット行が変更された行としてマークされないことを意味します。

別のソースを選択するには、[File]メニューから[Open Source(ソースを開く)]を選択します。

Visual ER Compare の処理

変更された特定の ER コンポーネントを識別し、表示するには、[ER]メニューを使用します。親ノードに変更のマークが設定されている場合は、親ノードを展開して、どの子が変更されているかを確認します。[ER]メニューからイベントをダブルクリックすると、それに関連するコードが表示されます。一度に複数のイベントを表示することができます。異なる ER イベントを同時に表示するには、[Window(ウィンドウ)]メニューの[Tile(タイル)]オプションを使用します。

また、変更から変更に移動することもできます。その場合は、ソース領域とターゲット領域のどちらかを右クリックし、前進するには[Next ER Difference(次の異なる ER)]、戻るには[Previous ER Difference(前の異なる ER)]を選択します。

ターゲット ER は、Visual ER Compare によって変更できます。変更を印刷することもできます。ソースからターゲットに変更を一括コピーするには、AutoMerge 機能を使用します。

ターゲット ER の変更

ターゲット ER を変更するには、以下の操作を実行します。

選択した行をソースからターゲットにコピーするには	コピーする行を選んで、ソース領域を右クリックして[Copy Right(右コピー)]を選択します。
選択した行をターゲットから削除するには	削除する行を選んで、ターゲット領域を右クリックして[Delete]を選択します。
選択したターゲットをターゲットで有効化/無効化するには	有効/無効にする行を選んで、ターゲット領域を右クリックして[Enable/Disable ER(ERの有効/無効)]を選択します。

注:

連続した複数の行を選択するには、Shift キーを使用し、非連続の複数の行を選択するには、Control キーを使用します。

変更を行ったら、右クリックして[Save ER(ER の保存)]を選択します。この操作により、変更がバッファに保存されます。新しいソースを開いたり Visual ER Compare を終了するときは、変更を再度保存するかを確認するメッセージが表示されます。このときに変更の保存を選択すると、使用しているワークステーションでオブジェクトが更新され、選択しなければ変更が失われます。

Visual ER Compare レポートの印刷

ソース ER とターゲット ER を比較するレポートを印刷できます。特定のイベントやオブジェクトのすべての ER の比較を印刷できます。

イベントのレポートを印刷するには、[ER]メニューからイベントをダブルクリックし、どちらかの領域を右クリックして、[Print ER(ER の印刷)]を選択します。

オブジェクト全体のレポートを印刷するには、[File]メニューから[Print ER]を選択します。

AutoMerge の使用

ターゲット ER をソース ER に合わせて変更するときには、AutoMerge を使用します。AutoMerge を使用すると、特定のイベントを変更したりオブジェクトのすべての ER を更新することができます。

注意:

オブジェクト全体で AutoMerge を実行する場合は、事前にソースとターゲットを比較して、検出されたすべての変更が実際に必要かどうかを確認してください。

AutoMerge をイベントで使用するには、[ER]メニューからイベントをダブルクリックし、どちらかの領域を右クリックして、[AutoMerge(自動マージ)]を選択します。

AutoMerge をオブジェクト全体で使用するには、[View]メニューから[Advanced Operations] – [Auto Merge]を選択します。

ビジネス関数

「ビジネス関数」では、C 言語ビジネス関数とイベント・ルール・ビジネス関数の両方について説明します。また、マスター・ビジネス関数、〈Business Function Builder（ビジネス関数ビルダー）〉、およびビジネス関数ドキュメンテーションに関する情報も説明します。

ビジネス関数の概要

ビジネス関数を使用すると、関連するビジネス・ロジックをグループ化して J.D. Edwards アプリケーションの機能を強化することができます。Journal Entry Transactions、Calculating Depreciation、および Sales Order Transactions は、ビジネス関数の例です。

ビジネス関数は、次の方法を使って作成できます。

- イベント・ルール・スクリプト言語

  イベント・ルール・スクリプト言語を使用して作成したビジネス関数は、イベント・ルール・ビジネス関数と呼ばれます。ビジネス関数には、できればイベント・ルール・ビジネス関数を使用するようにしてください。ただし、C 言語ビジネス関数のほうがニーズにより適合する場合もあります。

- C プログラミング・コード

  J.D. Edwards ソフトウェアは、C 言語で作成したビジネス関数を生成しません。C 言語ビジネス関数は、主にキャッシュのために使用します。また、次のような用途もあります。

  - バッチ・レベルのエラー・メッセージ処理
  - WHERE 節の OR プロパティの変更
  - 大型の関数
  - 複雑な Select ステートメント

    C 言語ビジネス関数は、大型の関数に適しています。大型の関数を使用する場合は、それらを個々に独立した小型の関数に分割し、より大きな関数からそれらを呼び出すことができます。

ビジネス関数を作成したら、J.D. Edwards アプリケーションに関連付けて、アプリケーションの機能と柔軟性と制御性を向上することができます。

ビジネス関数のコンポーネント

ビジネス関数の作成プロセスでは、いくつかのコンポーネントが作成されます。〈Object Management Workbench〉(OMW)は、コンポーネントを作成するツールのエントリ・ポイントです。次のコンポーネントが作成されます。

コンポーネント	作成場所
ビジネス関数スペック	〈Object Management Workbench〉 〈Business Function Source Librarian(ビジネス関数ソース・ライブラリアン)〉および〈Business Function Design(ビジネス関数の設計)〉
データ構造体スペック	〈Object Management Workbench〉 〈Business Function Parameter Design(ビジネス関数パラメータの設計)〉
.C ファイル	〈Business Function Design〉で生成 〈IDE〉で修正
.H ファイル	〈Business Function Design〉で生成 〈IDE〉で修正

DLL はいくつかのカテゴリに分類されます。これにより、ツール、会計、製造、流通などの主要な機能グループをより適切に区分することができます。ビジネス関数は、そのほとんどがシステム・コードに基づいて統合 DLL に組み込まれます。たとえばシステム・コードが 01 の会計ビジネス関数の場合は、CFIN.DLL に属します。

ビジネス関数を追加したり修正するときには、次のガイドラインに従ってください。

- J.D. Edwards ビジネス関数を追加しない場合は、カスタム親 DLL を作成します。UDC テーブルの H92/PL で定義されているシステム・コードに基づいて、親 DLL をビジネス関数に割り当てます。ビジネス関数が作成される DLL にシステム・コードが割り当てられていない場合は、CCUSTOM が使用されます。DLL はビジネス関数の作成後に変更できます。
- ビジネス関数コードの記述に際して、他のビジネス関数の呼出しでは、jdeCallObject プロトコルを使用してください。jdeCallObject を使用せずに異なる DLL のビジネス関数の呼出しを試みると、リンカ・エラーが発生する場合があります。リンカ・エラーが発生すると、関数呼出しは機能しなくなります。

次の表に、〈Business Function Builder〉がビルドを管理している DLL の例を示します。

DLL 名	機能グループ
CAEC	アーキテクチャ
CALLBSFN	統合 BSFN ライブラリ
CBUSPART	ビジネス・パートナー
CCONVERT	変換ビジネス関数

CCORE	コアのビジネス関数
CCRIN	業種間アプリケーション
CDBASE	ツール– データベース
CDDICT	ツール– データ辞書
CDESIGN	設計ビジネス関数
CDIST	流通
CFIN	会計
CHRM	人事管理
CINSTALL	ツール – インストール
CINV	在庫
CLOC	ローカライズ
CLOG	ロジスティクス関数
CMFG	製造
CMFG1	製造 – 修正ビジネス関数
CMFGBASE	製造の基本関数
COBJLIB	ツール– オブジェクト・ライブラリアン
COBLIB	Busbuild 関数
COPBASE	流通/ロジスティクスの基本関数
CRES	リソース・スケジュール
CRUNTIME	ツール – ランタイム
CSALES	受注オーダー
CTOOL	ツール – 設計ツール
CTRAN	輸送
CTRANS	ツール – 翻訳

CWARE	倉庫
CWRKFLOW	ツール – ワークフロー
JDBTRG1	テーブル・トリガー・ライブラリ 1
JDBTRG2	テーブル・トリガー・ライブラリ 2
JDBTRG3	テーブル・トリガー・ライブラリ 3
JDBTRG4	テーブル・トリガー・ライブラリ 4
JDBTRIG	データベース・トリガーの親 DLL

テーブル・トリガーと標準ビジネス関数に関する注:
標準ビジネス関数では、テーブル・トリガーを使用しないでください。

分散ビジネス関数の機能

〈Object Management Workbench〉では、イベント・ルール・ビジネス関数またはビジネス関数を構成する次の 3 つの主要コンポーネントを管理します。

- Object Name（オブジェクト名）

  実際のソース・ファイルです。

- Function Name（関数名）

  ビジネス関数名またはイベント・ルール名です。

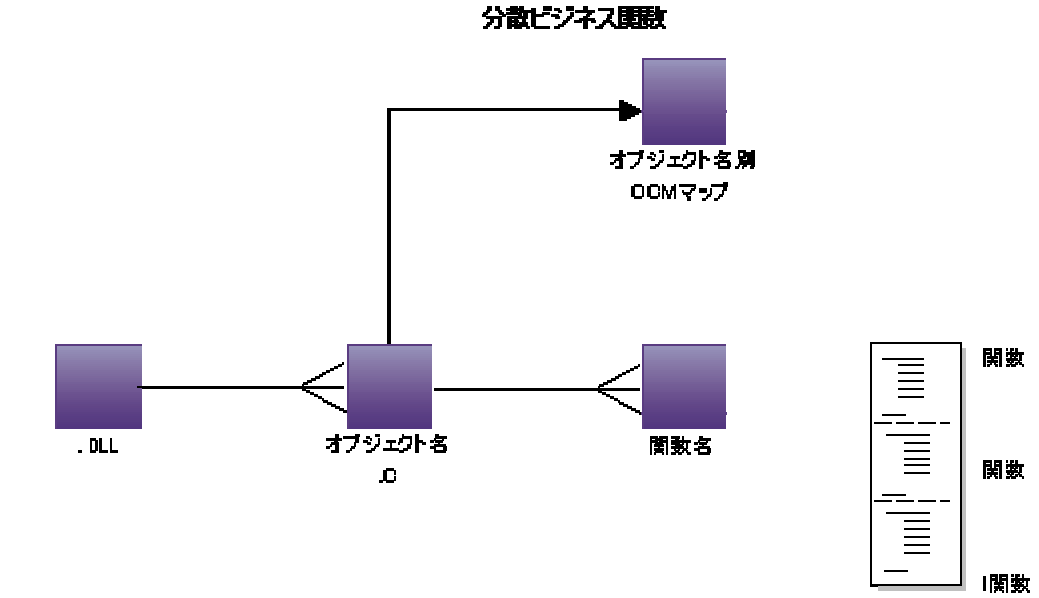
  注:
  各関数には、データ構造体が必要です。

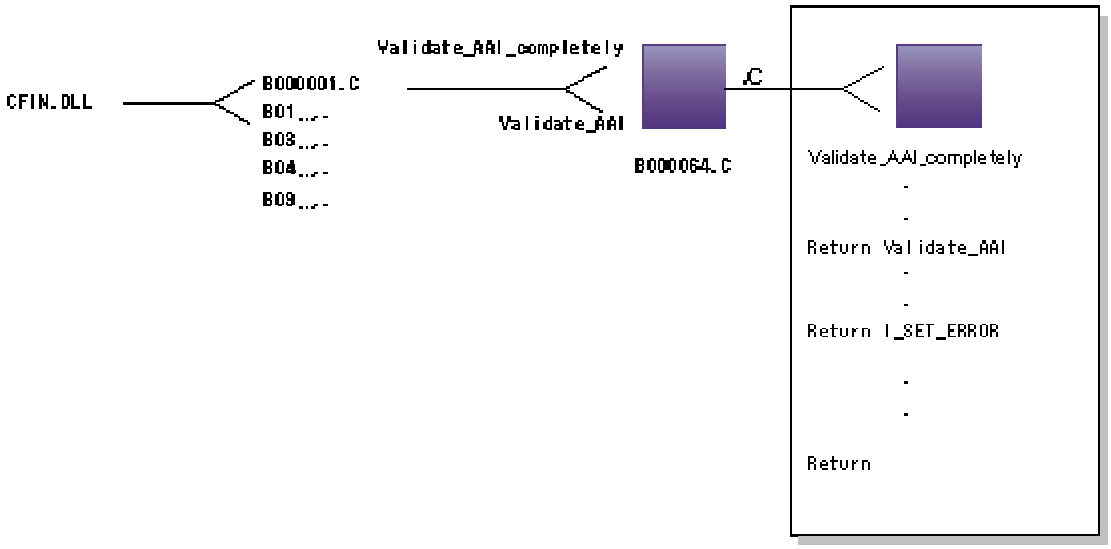
- DLL Name（DLL 名）

  ダイナミック・リンク・ライブラリの略称です。

ビジネス関数が呼び出されると、〈Object Configuration Manager〉(OCM)はビジネス関数をどこで実行するかを識別します。ビジネス関数がサーバーにマッピングされた後は、そのビジネス関数からの呼出しで、ワークステーションにマッピングし直すことができなくなります。

次の図は、分散ビジネス関数の処理を示しています。





参照

- 〈Object Configuration Manager〉については、『CNC インプリメンテーション』ガイドの「オブジェクト構成マネージャ(OCM)」

イベント・ルール・ビジネス関数の作成

イベント・ルール・ビジネス関数(BSFN ER)は、ソース言語が C 言語ではなく、イベント・ルールのビジネス関数オブジェクトです。これは、イベント・ルール・スクリプト言語を使用して作成します。このスクリプト言語はプラットフォーム独立型で、J.D. Edwards ソフトウェアのオブジェクトとしてデータベースに保管されます。イベント・ルール・ビジネス関数は、NER と呼ぶこともあります。

はじめる前に

- データ構造体を作成します。データ構造体の作成については『開発ツール』ガイドの「データ構造体」

▶ イベント・ルール・ビジネス関数を設計するには

1. 〈Object Management Workbench〉で、ビジネス関数を作成します。

   [Function Location(関数位置)]では、通常は[Client/Server Function(クライアント/サーバー関数)]を選択します。[Client Only(クライアントのみ)]と[Server Only(サーバーのみ)]の関数位置は、必要な場合に限って使用してください。

2. 〈Business Function Design〉で、[Design Tools]タブを選択して[Start Business Function Design Aid(設計ツールの開始)]をクリックします。
3. [Parent DLL(親 DLL)]フィールドに値を入力します。
4. [Business Function Location(C/S)(ビジネス関数のロケーション)]フィールドで、このビジネス関数を常駐させるロケーションを指定します。
5. [Row(ロー)]メニューから[Parameters(パラメータ)]を選択します。
6. 〈Business Function Parameters(ビジネス関数パラメータ)〉で、[Form(フォーム)]メニューから[Select(選択)]を選択します。
7. [Find]をクリックし、ビジネス関数を関連付けるデータ構造体を検索します。

   既存のデータ構造体がニーズに合わない場合は、新しく作成してください。

8. 新規のデータ構造体を作成するには、[Add]をクリックしてデータ構造体の作成手順を実行します。

ビジネス関数に既にデータ構造体が関連付けられている場合は、次のフォームが表示されます。他のデータ構造体を選択したり、使用するデータ構造体を編集することもできます。

▶ イベント・ルール・ビジネス関数を作成または編集するには

1. 〈Business Function Design〉で、[Form]メニューから[Edit(編集)]を選択し、関数を構成するイベント・ルールを作成または編集します。

   新規の BSFN イベント・ルールを作成する場合は、ソース言語として NER を選択してください。

2. [File]メニューから[Exit]を選択します。
3. 〈Business Function Event Rules Design(イベント・ルール・ビジネス関数の設計)〉で、イベント・ルール・ビジネス関数を作成します。

4. [Save(保存)]ボタンか[OK]ボタンをクリックして、イベント・ルールのソース・コードを保存し、〈Business Function Design〉に戻ります。
5. 〈Business Function Design〉で、[OK]をクリックして関数を保存し、〈Business Function Source Librarian〉に戻ります。
6. [Design Tools]タブで、[Build Business Function(ビジネス関数のビルド)]を選択して、.c ファイル、.h ファイル、make file を作成し、ビジネス関数をビルドします。

   ソース言語が NER であるため、〈Business Function Source Librarian〉は、イベント・ルール・ビジネス関数用の objname.c ファイルと objname.h ファイルを作成します。イベント・ルール・ビジネス関数は、ビジネス関数に関連する親 DLL に組み入れられます。

7. 新規に作成したイベント・ルール・ビジネス関数をイベントに関連付けるには、〈Event Rules Design(イベント・ルールの設計)〉の[f(B)]ボタンをクリックします。

   使用可能なビジネス関数オブジェクトを参照するときは、ソース言語が NER であることを確認してください。

例:イベント・ルール・ビジネス関数

イベント・ルール・ビジネス関数はモジュール方式です。つまり、複数のプログラムのさまざまな部分で再利用することができます。このようなモジュール方式により書き直しの手間が減り、コードの再利用が可能になります。

コードのすべての部分をビジネス関数モジュールに組み込む必要はありません。たとえば、コードが独特で特定のプログラムにしか適合せず、他のプログラムで再利用されない場合は、ビジネス関数に組み込む代わりに、1 ヵ所にとどめておきます。そして、ロジック全体を隠しコントロール(Button Clicked イベント)で関連付け、システム関数を使用してロジックを必要に応じて処理できます。

イベント・ルール・ビジネス関数の一例は、N3201030 です。このビジネス関数は、構成作業オーダーの汎用テキストと作業オーダー詳細レコード(F4802)を作成します。F3296 の受注オーダーの構成に基づき、受け渡された作業オーダーでの品目の構成セグメントとそれよりも下位のすべてのセグメントが汎用テキストに含められます。

この関数は、〈Event Rules Design〉で次のように表示されます。

```
//
// Convert the related sales order number into a math numeric.  If that
fails,
// exit the function.
//
String, Convert String To Numeric
If VA evt_cErrorCode is equal to "1"
  //
  // Validate that the work order item is a configured item.
  //
  F4102 Get Item Manufacturing Information
  If VA evt_cStockingType is not equal to "C"And BF
cSuppressErrorMessages is not equal to "1"
  BF szErrorMessageID = "3743"
  Else
  BF szErrorMessageID = ""
  //
  // Delete all existing "A" records from F4802 for this work order.
  //
  VA evt_cWODetailRecordType = "A"
  F4802.Delete
  F4802.Close
  //
  // Get the segment delimiter from configurator constants.
  //
  F3293 Get Configurator Constant Row
  If VA evt_cSegmentDelimiter is less than or equal to  <Blank>
  VA evt_cSegmentDelimiter = "/"
End If
//
F3296.Open
F3296.Select
If SV File_IO_Status                  is equal to CO SUCCESS

F3296.FetchNext
//
// Retrieve the F3296 record of the work order item, and determine its
key
// sequence by parsing ATSQ looking for the last occurrence of '1'.  The
substring
// of ATSQ to this point becomes the key for finding the lower level
configured
// strings.
//
If VA evt_mnCurrentSOLine is equal to BF mnRelatedSalesOrderLineNumber
// Get the corresponding record from F32943.  Process the results of
that fetch
// through B3200600 to add the parent work order configuration to the
work order
// generic text.
```

```
F32943.FetchSingle
If SV File_IO_Status                  is equal to CO SUCCESS

  VA evt_szConfiguredString = concat([VA evt_ConfiguredStringSegment01],
  [VA evt_ConfiguredStringSegment02])
  Confg String Format Segments Cache
End If
//
// Find the last level in ATSQ that is not "00".  Note that the first three
// characters represent the SO Line Number to the left of the decimal.
Example:
// SO line 13.001 will have ATSQ characters "013".  Each configured item can
have
// 99 lower-level P-Rule items and a total of ten levels.  Therefore every
pair
// thereafter is tested.
//
VA evt_mnSequencePosition = "1"
While VA evt_mnSequencePosition is less than "23"
And VA evt_szCharacterPair is not equal to "00"
VA evt_mnSequencePosition = [VA evt_mnSequencePosition] + 2
VA evt_szCharacterPair = substr([VA evt_szTempATSQ],[VA
evt_mnSequencePosition],2)
End While
VA evt_szParentATSQ = substr([VA evt_szTempATSQ],0,[VA
evt_mnSequencePosition])
//
// For each record in F3296 for the related sales order, find those with the
same
// key substring of ATSQ.  Retrieve the associated record from F32943 if
// available and pass the configured string to N3200600 for addition to the
work
// order generic text.
//
F3296.FetchNext
While SV File_IO_Status                    is equal to CO SUCCESS

VA evt_szChildATSQ = substr([VA evt_szTempATSQ],0,[VA
evt_mnSequencePosition])
If VA evt_szChildATSQ is equal to VA evt_szParentATSQ
F32943.FetchSingle
If SV File_IO_Status                  is equal to CO SUCCESS

VA evt_szConfiguredString = concat([VA evt_ConfiguredStringSegment01],
[VA evt_ConfiguredStringSegment02])
Confg String Format Segments Cache
End If
End If
F3296.FetchNext
End While
F32943.Close
//
// Unload segments cache into the work order generic text. B3200600 Mode 6
Confg String Format Segments Cache
//
End If
End If
F3296.Close
//
End If
Else
// The related sales order number is invalid. Return an error.
If BF cSuppressErrorMessages is not equal to "1"
Set NER Error("0002", BF szRelatedSalesOrderNumber)
End If
```

```
End If
End If
```

C 言語ビジネス関数の概要

J.D. Edwards ソフトウェアには、イベント・ルール・ビジネス関数(NER)および C 言語ビジネス関数という 2 種類のビジネス関数が含まれています。C 言語ビジネス関数は C プログラム言語で記述されており、NER で使用できない関数を実行するために使用されます。C 言語ビジネス関数には、ヘッダー・ファイル(.h)とソース・ファイル(.c)の両方があります。

ヘッダーファイル・セクション

次の表に、ビジネス関数ヘッダー・ファイルの主要なセクションを示します。

セクション	内 容	説明とガイドライン
ヘッダー・ファイル・コメント	• ヘッダー・ファイル名 • 記述 • 履歴 • プログラマ名 • ソフトウェア・アクション・リクエスト(SAR)番号 • 著作権情報	〈Business Function Source Librarian〉の入力プロセスで作成されるコメント。 プログラマ名と SAR 番号は、プログラマが手作業で更新します。
テーブル・ヘッダー組込項目	ビジネス関数が直接アクセスするテーブルに関連付けられているヘッダー・ファイルのインクルード・ステートメント	テーブルのヘッダー・ファイルには、テーブルのフィールドとテーブル自体の ID の定義が含まれます。
外部ビジネス関数ヘッダー組込項目	ビジネス関数が直接アクセスする外部定義ビジネス関数に関連付けられているヘッダーのインクルード・ステートメント	既定のデータ構造体を使用するための jdeCallObject による外部関数呼出し
グローバル定義	ビジネス関数によって使用されるグローバル定数	記号名を大文字で入力し、単語と単語を下線で区切ります。グローバル定義の使用は控えてください。
構造体定義	内部処理のためのデータ構造体定義	構造体の名前を定義します。名前の競合を避けるために、ソース・ファイル名を構造体の名前のプレフィックスとして使用します。
DS テンプレート・タイプ定義	• 〈Business Function Design〉によって生成されたデータ構造体のタイプ定義 • 〈Business Function Design〉によって生成されたデータ構造体の記号定数	この構造体は〈Object Management Workbench〉を通じて修正してください。
ソース・プリプロセッサ	• 未定義の DEBFRTN(既に定義されている場合) • JDEBFRTN の定義法をチェックします。 • JDEBFRTN を定義します。	ビジネス関数の宣言とプロトタイプが、環境とこのヘッダーを含むソース・ファイルに対して適切に定義されるようにします。

ビジネス関数 プロト タイプ	ソース・ファイルに含まれているすべての ビジネス関数のプロトタイプ	ソース・ファイルのビジネス関数、それらに受け渡すパラメータ、および戻り値を定義します。
内部関数プロトタイプ	ソース・ファイルのビジネス関数のサポートに必要なすべての内部関数のプロトタイプ	ソース・ファイルのビジネス関数に関連する内部関数、各内部関数に受け渡すパラメータ、および戻り値のタイプを定義します。

ビジネス関数ヘッダー・ファイルの一例

〈Business Function Design〉によって次のヘッダー・ファイルが作成されているとします。このヘッダー・ファイルは、ビジネス関数ヘッダー・ファイルに必要な要素のみで構成されています。

```
/******************************************************************************
 *    Header File:  B98SA001.h
 *
 *    Description:  Check for In Add Mode Header File
 *
 *       History:
 *         Date        Programmer  SAR# - Description
 *         ----------  ----------  ------------------------------------------
 *   Author 03/07/1995   Hotchkiss  Unknown  - Created
 *
 *
 * Copyright (c) 1994  J.D. Edwards & Company
 *
 * This unpublished material is proprietary to J.D. Edwards & Company.
 * All rights reserved.  The methods and techniques described herein are
 * considered trade secrets and/or confidential.  Reproduction or
 * distribution, in whole or in part, is forbidden except by express
 * written permission of J.D. Edwards & Company.
 ******************************************************************************/


#ifndef _B98SA001_H
#define _B98SA001_H
/******************************************************************************
 * Table Header Inclusions
 ******************************************************************************/


/******************************************************************************
```

```
 * External Business Function Header Inclusions
 ****************************************************************************/


/****************************************************************************
 * Global Definitions
 ****************************************************************************/


/****************************************************************************
 * Structure Definitions
 ****************************************************************************/


/****************************************************************************
 * DS Template Type Definitions
 ****************************************************************************/
/*****************************************
 * TYPEDEF for Data Structure
 *    Template Name: Check for In Add Mode
 *    Template ID:   104438
 *    Generated:     Tue Mar 07 09:33:47 1995
 *
 * DO NOT EDIT THE FOLLOWING TYPEDEF
 *    To make modifications, use the OneWorld Data Structure
 *    Tool to Generate a revised version, and paste from
 *    the clipboard.
 *
 *****************************************/


#ifndef DATASTRUCTURE_104438
#define DATASTRUCTURE_104438


typedef struct tagDS104438
{
  char              cEverestEventPoint01;               /* OneWorld Event Point 01 */
} DS104438, FAR *LPDS104438;


#define IDERRcEverestEventPoint01_1              1L
```

```
#endif

/*****************************************************************************
 * Source Preprocessor Definitions
 *****************************************************************************/
#if defined (JDEBFRTN)
   #undef JDEBFRTN
#endif

#if defined (WIN32)
   #if defined (b98sa001_c)
      #define JDEBFRTN(r) _declspec(dllexport) r
   #else
      #define JDEBFRTN(r) _declspec(dllimport) r
   #endif
#else
   #define JDEBFRTN(r) r
#endif

/*****************************************************************************
 * Business Function Prototypes
 *****************************************************************************/
JDEBFRTN(ID) JDEBFWINAPI CheckForInAddMode
  (LPBHVRCOM lpBhvrCom, LPVOID lpVoid, LPDS104438 lpDS);

/*****************************************************************************
 * Internal Function Prototypes
 *****************************************************************************/

#endif    /* _B98SA001_H */
```

次の表に、このヘッダー・ファイルの各行の内容を示します。

	ヘッダー・ファイル行	入力元	説 明
1.	ヘッダー・ファイル	〈Object Management Workbench〉	ビジネス関数ヘッダー・ファイルの名前を確認します。
2.	記述	〈Object Management Workbench〉	説明を確認します。
3.	履歴	IDE（統合開発環境）	プログラマ名と適切な SAR 番号を使って修正ログを手作業で更新します。
4.	#ifndef	Business Function Design	記号定数を定義して、同じ内容を繰り返し入力しなくても済むようにします。
5.	テーブル・ヘッダー組込項目	Business Function Design	ビジネス関数がテーブルにアクセスすると、関連テーブルが入力され、〈Business Function Design〉がそのテーブル・ヘッダー・ファイルのインクルード・ステートメントを生成します。
6.	外部ビジネス関数ヘッダー組込項目	Business Function Design	このアプリケーションには、外部ビジネス関数がありません。
7.	グローバル定義	IDE（統合開発環境）	ビジネス関数の定数と定義。J.D. Edwards では、このブロックの使用を推奨しません。グローバル変数は推奨できません。グローバル定義は.h ファイルでなく、.c ファイルで行います。
8.	構造体の定義	IDE（統合開発環境）	ビジネス関数、内部関数、データベース API の 3 者間で情報を受け渡すために使用するデータ構造体です。
9.	データ構造体の TYPEDEF	Business Function Design	データ構造体のタイプ定義です。アプリケーションやレポートとビジネス関数との間でデータを受け渡すために使用されます。プログラミングの際には、これをクリップボードにコピーして、ヘッダー・ファイルに貼り付けます。タイプ定義の要素は次のとおりです。 コメント・ブロック – データ構造体に関する説明です。 プリプロセッサの指示文 – データ・タイプが 1 回のみ定義されるようにします。 Typedef – 新しいデータ・タイプを定義します。 #define – 関連データ構造体要素がエラーになった場合に処理で使用される ID を含みます。 #endif – データ構造体のタイプ定義とその関連情報の定義を終了します。

	ヘッダー・ファイル行	入力元	説 明
10.	ソース・プリプロセッサの定義	Business Function Design	ビジネス関数ヘッダー・ファイルには、いずれもこのセクションがあり、このヘッダーが組込み先に基づいて、ビジネス関数のプロトタイプを宣言できるようにします。
11.	ビジネス関数プロト タイプ	Business Function Design	ビジネス関数のプロトタイプのために使用されます。
12.	JDEBFRTN(ID) JDEBFWINAPI CheckForInAddMode	Business Function Design	ビジネス関数のスタンダード。 ビジネス関数は、それらのすべてが同じリターン・タイプとパラメータ・データ・タイプを共有します。関数名とデータ構造体番号のみが、ビジネス関数によって異なります。 パラメータは次のとおりです。 LPBHVRCOM：ビジネス関数とのデータの受け渡しに使用するデータ構造体へのポインタ。値は、環境ハンドルを含みます。 LPVOID：void のデータ構造体へのポインタ。現在はエラー処理のために使われていますが、今後はセキュリティのために使用されます。 LPDS#####：ビジネス関数と呼出し元のアプリケーションやレポートとの間で受け渡される情報を格納するデータ構造体へのポインタ。この番号は、〈Object Librarian〉によって生成されます。 パラメータ名（lpBhvrCom、lpVoid、lpDS）は、すべてのビジネス関数で共通です。 JDEBFRTN(ID)JDEBFWINAPI：ビジネス関数の宣言では、常にこのリターン・タイプが使用されます。これにより、ビジネス関数は正しくエクスポート、インポートされます。
13.	内部関数のプロトタイプ	Business Function Design	このソース・ファイルでビジネス関数をサポートするために必要な内部関数のプロトタイプです。

ビジネス関数ソース・ファイルの構成について

〈Object Management Workbench〉は、ビジネス関数ソース・ファイルのテンプレートを作成します。ビジネス関数ソース・ファイルは、次の表のように複数の主要セクションで構成されています。

セクション	内 容	説 明
ソース・ファイルのコメント・ブロック	• ソース・ファイル名 • 説 明 • 履歴 • プログラマ名 • 日付 • SAR 番号 • 記述 • 著作権情報	〈Business Function Source Librarian〉で指定された情報から作成されます。 プログラマ名と SAR 番号は、プログラムが手動で更新します。
注記コメントブロック	ビジネス関数ソースに関する追加の関連注記	使用されている複雑なアルゴリズム、ソースのビジネス関数の相互関係などについて説明します。
ビジネス関数コメント・ ブロック	• ビジネス関数名 • 記述 • パラメータの記述リスト	
ビジネス関数 ソース・コード	ビジネス関数のソース・コード	
内部関数コメント・ブロック	• 関数名 • コメント • 戻り値 • パラメータ	これらのブロックをコピーし、指定されたセクションに値を設定して、内部関数について説明します。コメント・ブロックは、内部関数ソース・コードの前に記述します。
内部関数ソース・コード	コメント・ブロックで説明されている内部関数のソース・コード	このコードは、ビジネス関数の開発者が必要に応じて入力します。内部関数コメント・ブロックの後に記述します。

例:ビジネス関数ソース・ファイル

〈Business Function Design〉で、次のように Check for In Add Mode というソース・ファイルが作成されているとします。このソース・ファイルは、ビジネス関数ソース・ファイルに必要な最小限の要素から構成されています。メイン処理セクションのソース・コードは手作業で入力され、関数ごとに異なります。その他の要素は、〈Business Function Design〉によってすべて生成されます。

```
#include <jde.h>
#define b98sa001_c

/*****************************************************************************
 *    Source File:  B98SA001.c
 *
 *    Description:  Check for In Add Mode Source File
 *
 *       History:
 *         Date        Programmer  SAR# - Description
 *         ----------  ----------  ------------------------------------------
 *   Author 03/07/1995  Hotchkiss    Unknown - Created
 *
 * Copyright (c) 1994  J.D. Edwards & Company
 *
 * This unpublished material is proprietary to J.D. Edwards & Company.
 * All rights reserved.  The methods and techniques described herein are
 * considered trade secrets and/or confidential.  Reproduction or
 * distribution, in whole or in part, is forbidden except by express
 * written permission of J.D. Edwards & Company.
 *****************************************************************************/
/*****************************************************************************
 * Notes:
 *
 *****************************************************************************/

#include <B98SA001.h>

/*****************************************************************************
 *  Business Function:  CheckForInAddMode
 *
```

```
*       Description:  Check for In Add Mode
*
*        Parameters:
*          LPBHVRCOM       lpBhvrCom    Business Function Communications
*          LPVOID          lpVoid       Void Parameter - DO NOT USE!
*          LPDS104438      lpDS         Parameter Data Structure Pointer
*
*****************************************************************************/

JDEBFRTN(ID) JDEBFWINAPI CheckForInAddMode (LPBHVRCOM lpBhvrCom, LPVOID lpVoid,
LPDS104438 lpDS)
{
   /*************************************************************************
    *  Variable declarations
    *************************************************************************/

   /*************************************************************************
    * Declare structures
    *************************************************************************/

   /*************************************************************************
    * Declare pointers
    *************************************************************************/

   /*************************************************************************
    * Check for NULL pointers
    *************************************************************************/
   if ((lpBhvrCom == NULL) ||
       (lpVoid    == NULL) ||
       (lpDS      == NULL))
   {
      jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4363", 0);
      return ER_ERROR;
   }

   /*************************************************************************
    * Set pointers
```

```
    ***************************************************************************/

   /***************************************************************************
    * Main Processing
    ***************************************************************************/

   /***************************************************************************
    * Function Clean Up
    ***************************************************************************/

   return (ER_SUCCESS);
}

/* Internal function comment block */
/*****************************************************************************
 *   Function:  Ixxxxxxx_a   // Replace "xxxxxxx" with a source file number
                             // and "a" with the function name
 *
 *      Notes:
 *
 *    Returns:
 *
 * Parameters:
 *****************************************************************************/
```

次の表に、このソース・ファイルに表示される各行を示します。

	ソース・ファイル行	入力元	説明とガイドライン
1.	#include <jde.h>	Business Function Design	すべての基本 J.D. Edwards 定義を組み入れます。
2.	#define b98sa001_c	Business Function Design	関連ヘッダー・ファイルの定義をこのソース・ファイルに対して正しく作成します。
3.	ソース・ファイル	〈Object Management Workbench〉	ファイル・コメント・セクションの情報を確認します。プログラマ名、SAR 番号、説明を入力してください。
4.	#include <B98SA001.h>	〈Object Management Workbench〉	このアプリケーションで使用するヘッダー・ファイルを取り込みます。
5.	ビジネス関数	Business Function Design	ビジネス関数コメント・ブロックの名前と説明を確認します。
6.	JDEBFRTN(ID) JDEBFWINAPI CheckForInAddMode (LPBHVRCOM lpBhvrCom, LPVOID lpVoid, LPDS104438 lpDS)	Business Function Design	ビジネス関数宣言のヘッダーを含みます。
7.	変数の宣言	IDE	ビジネス関数に対してローカルの変数を宣言します。
8.	構造体の宣言	IDE	ビジネス関数、内部関数、データベースの 3 者間でデータを受け渡すためのローカル・データ構造体を宣言します。
9.	ポインタの宣言	IDE	ポインタを宣言します。
10.	NULL ポインタのチェック	Business Function Design	ビジネス関数のスタンダード アプリケーションとビジネス関数間で受け渡される構造体がすべて有効であることを確認します。

	ソース・ファイル行	入力元	説明とガイドライン
11.	jdeErrorSet (lpBhvrCom, lpVoid, (ID) 0, "4363", 0); return ER_ERROR;	Business Function Design	交信用データ構造体のいずれかが無効な場合に、呼出し元のアプリケーションに返す標準エラーを設定します。
12.	ポインタの設定	IDE	ポインタを宣言し、適切な値を割り当てます。
13.	主要な処理機能	IDE	ビジネス関数の主要な機能を設定します。
14.	関数のクリーンアップ	IDE	動的に割り当てられたメモリをすべて解放します。
15.	内部関数コメント・ブロック	IDE	ビジネス関数をサポートするために必要な内部関数を定義します。これらも共通の C プログラミング標準に従います。上述した例と同じような形式で、それぞれの内部関数に対してもコメント・ブロックが必要となります。

J.D. Edwards ソフトウェアで数値を表現するときは、常に MATH_NUMERIC データ・タイプを使用します。フォームやバッチ処理の数値フィールドの値はいずれも MATH_NUMERIC データ構造体へのポインタの形でビジネス関数に送られます。MATH_NUMERIC は、データ辞書データ・タイプとして使用されます。

アプリケーション・プログラミング・インターフェイス(API)の使用

アプリケーション・プログラミング・インターフェイス(API)は、あらかじめ定義されているタスクを実行するルーチンです。J.D. Edwards の API は、他社製アプリケーションと J.D. Edwards ソフトウェア間の処理を容易にします。これらの API は、J.D. Edwards データ・タイプを処理したり、共通の機能を提供したり、データベースにアクセスするために使用できる関数です。API には、共通ライブラリ・ルーチンや J.D. Edwards データベース(JDEBASE) API などの複数のカテゴリがあります。

J.D. Edwards API を使用したプログラムは次の理由により柔軟性があります。

- 機能の更新時にコード修正が不要である。
- J.D. Edwards データ構造体の変更時にソースの修正が極小か、皆無である。
- 共通機能が API を通じて提供され、エラーがほとんど発生しない。

API のコードを変更したときは、一般にビジネス関数をリコンパイルして再リンクするだけで十分です。

共通ライブラリ API の使用

共通ライブラリ API は、外貨を有効にするかどうかの決定や日付表示フォーマットの処理、リンク・リスト情報の取出し、数値や日付情報の取出しといった J.D. Edwards 機能に固有の API です。これらの API を使用してデータを設定するには、API を呼び出し、呼出し後にデータを修正します。API では、MATH_NUMERIC、JDEDATE、LINKLIST などがよく使用されますが、その他の共通ライブラリ API も使用できます。

J.D. Edwards には、ビジネス関数を作成するときに使用する 2 つのメイン・データ・タイプがあります。これらは以下のとおりです。

- MATH_NUMERIC
- JDEDATE

これらのデータ・タイプは変更される可能性があります。そのため、これらのデータ・タイプの変数を操作するには、J.D. Edwards の共通ライブラリ API を使用する必要があります。

MATH_NUMERIC データ・タイプ

J.D. Edwards ソフトウェアでは、数値は常に MATH_NUMERIC データ・タイプで表されます。フォームやバッチ処理の数値フィールドの値はいずれも MATH_NUMERIC データ構造体へのポインタの形でビジネス関数に送られます。MATH_NUMERIC は、データ辞書データ・タイプとして使用されます。

このデータ・タイプは次のように定義されます。

```
struct tagMATH_NUMERIC


{
   char  String [MAXLEN_MATH_NUMERIC + 1];
   char  Sign;
   char  EditCode;
   short nDecimalPosition;
   short nLength;
   WORD  wFlags;
   char  szCurrency [4];
   short nCurrencyDecimals;
   short nPrecision;
};


typedef struct tagMATH_NUMERIC MATH_NUMERIC, FAR *LPMATH_NUMERIC;
```

次の表に、MATH_NUMERIC の各要素を示します。

MATH_NUMERIC 要素	説　明
String	区切り文字なしの文字列
Sign	マイナス符号は数値が負であることを示し、それ以外では、値は 0x00 になります。
EditCode	数値の表示形式を設定するデータ辞書の編集コード
nDecimalPosition	小数点以下の桁数
nLength	String の桁数
wFlags	処理フラグ
szCurrency	通貨コード
nCurrencyDecimals	通貨の小数部の桁数
nPrecision	データ辞書サイズ

JDEDATE データ・タイプ

J.D. Edwards ソフトウェアでは、日付は常に JDEDATE データ・タイプで表されます。フォームやバッチ処理の日付フィールドの値は、いずれも JDEDATE データ構造体へのポインタによりビジネス関数に渡されます。JDEDATE はデータ辞書データ・タイプとして使用されます。

このデータ・タイプは次のように定義されます。

```
struct tagJDEDATE
{
   short nYear;;
   short nMonth;;
   short nDay;
};


typedef struct tagJDEDATE JDEDATE, FAR *LPJDEDATE;
```

次の表に、JDEDATE データ・タイプの各要素を示します。

JDEDATE 要素	説　明
nYear	年（4 桁）
nMonth	月
nDay	日

データベース API の使用

J.D. Edwards ソフトウェアでは複数のデータベースがサポートされます。アプリケーションは複数のデータベースのデータにアクセスすることができます。API には次の利点があります。

- 複数のデータベース管理システムへの標準インターフェイスを提供する。
- SQL に関するごくわずかな知識で、複雑なデータベース操作を実行できる。
- データベース・システムで追加、修正、削除、照会の各操作を実行できる。
- データベースとの間でデータをやりとりするためにメモリ領域を管理する。
- ランタイムで SQL ステートメントを作成して実行する。
- 埋込み型の SQL よりも柔軟である。
- クライアント/サーバー環境でのアプリケーションの開発方法を改善する。
- 関数呼出しからの標準リターン・コードを提供する。

標準と移植性について

リレーショナル・データベースの開発は、次の標準の影響を受けます。

- ANSI (American National Standards Institute)標準
- X/OPEN (European body)標準
- ISO (International Standards Institute)の SQL 標準

業界標準により、異なるリレーショナル・データベース・システムを同じように処理できるのが理想的です。主要なベンダ各社は業界標準をサポートしていますが、SQL 言語の機能を強化する拡張機能も提供しています。さらにベンダ各社は、自社製品のアップグレードと新バージョンを定期的にリリースしています。

このような機能拡張とアップグレードは移植性に影響します。ソフトウェア開発が産業に及ぼす影響により、アプリケーションには、データベース・ベンダの違いによって左右されない、データベースへの標準インターフェイスが必要です。ベンダ各社が新リリースを提供する場合、既存のアプリケーションへの影響はごくわずかに抑える必要があります。このような移植性に関するさまざまな問題を解決するために、多くの組織がオープン・データベース・コネクティビティ(ODBC)と呼ばれる標準データベース・インターフェイスを採用しています。

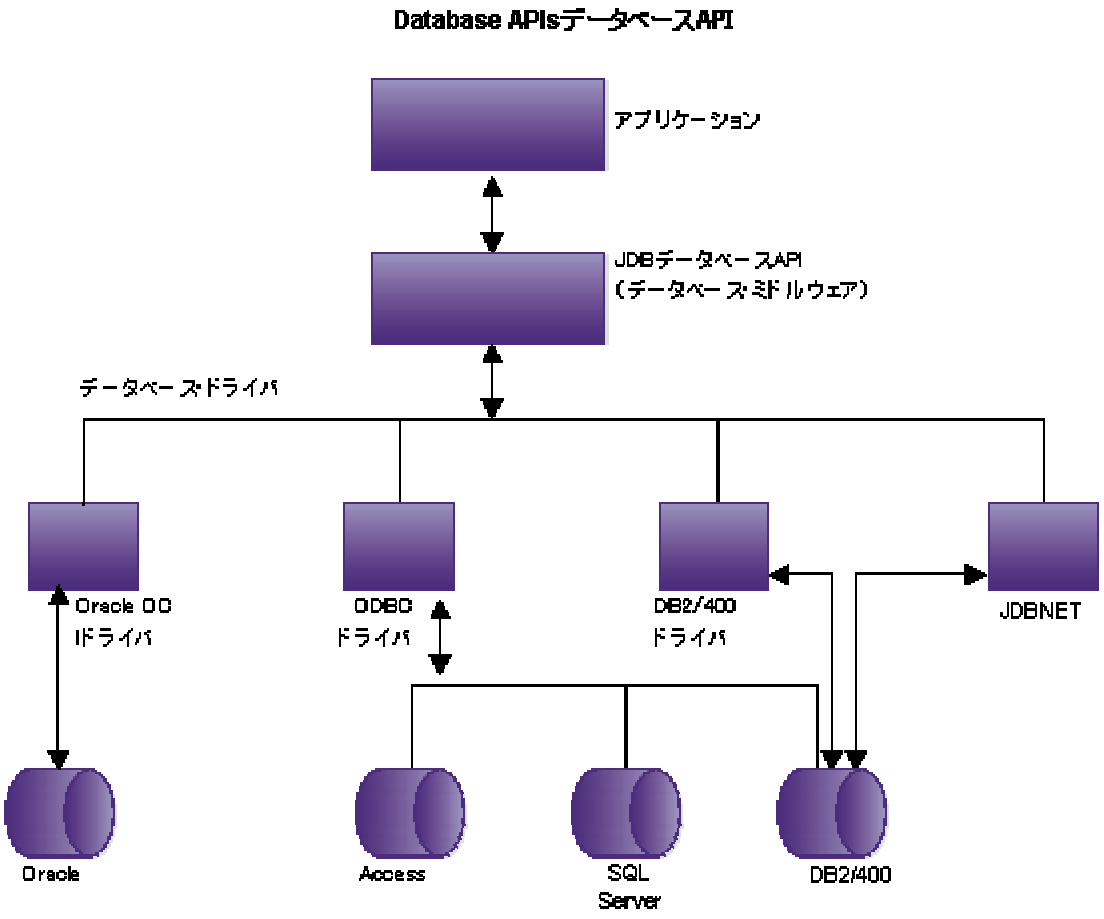
J.D. Edwards オープン・データベース・コネクティビティ(ODBC)

J.D. Edwards ODBC は、複数のリレーショナル・データベース管理システムに単一アクセス機能を提供します。ODBC により、一組の関数を使用してさまざまなタイプのデータベースにインターフェイスすることができます。開発者は、各アプリケーションが使用するデータベースのタイプを知らなくても、アプリケーションを開発しコンパイルできます。データベース・ドライバは、J.D. Edwards ODBC インターフェイスと特定のデータベース・システムと交信を可能にするものをインストールします。

ドライバは、API 要求を処理し、データベースと交信し、API に結果を返すアプリケーションです。データベースへの入出力(I/O)バッファも処理します。これにより、プログラマは、一般的なデータ・ソースと交信するアプリケーションを作成することができます。データベース・ドライバは、API 要求の処理と適切なデータ・ソースとの交信を担当します。アプリケーションは、他のデータベースを処理するためにコンパイルする必要はありません。アプリケーションが他のデータベースで同じ操作を実行しなければならない場合は、新しいドライバをロードします。

ドライバ・マネージャは、J.D. Edwards データベース関数呼出しに対するすべてのアプリケーション要求を処理します。ドライバ・マネージャは、要求を処理したりドライバに受け渡したりします。

次の図は、J.D. Edwards ソフトウェアによるデータベース API の使用方法を示しています。



J.D. Edwards アプリケーションは、異種データベースからのデータにアクセスします。その際に、JDB API を使用してアプリケーションと複数のデータベース間のインターフェイスを提供します。アプリケーションとビジネス関数は、JDB API を使用してプラットフォーム固有の SQL ステートメントを動的に生成します。JDB は、レプリケーションや相互データ・ソース結合などの付加機能もサポートします。

標準 JDEBASE API カテゴリ

次の表に、JDEBASE API のカテゴリを示します。制御および要求レベルの API を使用して、ビジネス関数を開発し、テストすることができます。

カテゴリ	説 明
制御レベル	データベース接続の初期化および終了の機能を提供します。
要求レベル	データベース・トランザクションを実行するための機能を提供します。要求レベルの機能は、次のタスクを実行します。 • データベースのテーブルとビジネス・ビューに接続/接続解除します。 • 選択、挿入、更新、削除のデータ処理操作を実行します。 • Fetch(取込み)コマンドによりデータを取り出します。
カラム・レベル	カラムとテーブルの情報を処理し、修正します。
グローバル・テーブル/ カラム・スペック	カラム・スペックを作成し、処理する機能を提供します。

データベースへの接続

要求を実行するには、ドライバ・マネージャとドライバで、開発環境、各アプリケーション接続および SQL ステートメントに関する情報を管理する必要があります。この情報をアプリケーションに返すポインタをハンドルといいます。API は、これらのハンドルを各関数呼出しに組み入れなければなりません。開発環境によって使用されるハンドルは次のとおりです。

ハンドル	用途
HENV	環境ハンドルには、現在のデータベース接続および有効なコネクション・ハンドルに関するデータが含まれています。データベースに接続している各アプリケーションには環境ハンドルを持たせる必要があります。このハンドルはデータ・ソースに接続するために必須です。
HUSER	ユーザー・ハンドルには、特定の接続に関するデータが含まれています。各ユーザー・ハンドルは、関連する環境ハンドルを持っています。コネクション・ハンドルはデータ・ソースに接続するために必須です。トランザクション処理を使用する場合は、huser を初期化するとトランザクションが開始されます。
HREQUEST	要求ハンドルには、データ・ソースへの特定要求に関するデータが含まれています。アプリケーションに要求ハンドルを指定してから SQL ステートメントを実行する必要があります。各要求ハンドルはユーザー・ハンドルと関連付けられています。

データベース交信ステップについて

いくつかの API を使用して、以下のデータベース交信ステップを実行できます。

- データベースとの交信を初期化する。
- アクセスする特定のデータへの接続を確立する。
- データベース上でステートメントを実行する。
- データベースとの接続を解除する。
- データベースとの交信を終了する。

次の表に、一部の API レベルと、それに関連付けられている交信ハンドルおよび API 名を示します。

API レベル	交信ハンドル	API 名
制御レベル（アプリケーションまたはテスト・ドライバ）	環境ハンドル	JDB_InitEnv
制御レベル（アプリケーションまたはテスト・ドライバ）	ユーザーハンドル（作成済）	JDB_InitUser
要求レベル（ビジネス関数）	ユーザー・ハンドル（取出し済）	JDB_InitBhvr
要求レベル（ビジネス関数）	要求ハンドル	JDB_OpenTable
	要求ハンドル	JDB_FetchKeyed()
	要求ハンドル	JDB_CloseTable
	ユーザーハンドル	JDB_FreeBhvr
制御レベル（アプリケーションまたはテスト・ドライバ）	ユーザーハンドル	JDB_FreeUser
制御レベル（アプリケーションまたはテスト・ドライバ）	環境ハンドル	JDB_FreeEnv

外部ビジネス関数からの API の呼出し

外部ビジネス関数から API を呼び出すことができます。その場合はまず最初に、使用する.dll の関数呼出し規則を設定する必要があります。これは、cdcl と stdcall のいずれかになります。コードは、呼出し規則に応じてわずかに変更されることがあります。この情報は、.dll のドキュメンテーションに記載されています。DLL の呼出し規則がわからない場合は、dumpbin コマンドを実行して確認できます。MSDOS プロンプトから次のコマンドを実行してください。

```
dumpbin /EXPORTS ExternalDll.DLL.
```

Dumpbin によって DLL に関する情報が表示されます。この出力に、「_」から始まり@マークといくつかの数字が後続する関数名が含まれている場合、DLL は呼出し規則として stdcall を使用し、それ以外の場合は cdecl を使用します。

stdcall 呼出し規則

以下に示すサンプル・コードは Windows プログラムの標準コードです。J.D. Edwards ソフトウェア固有のコードではありません。

```
# ifdef JDENV_PC

HINSTANCE hLibrary = LoadLibrary(_TEXT("YOUR_LIBRARY.DLL")); // substitute the name of the external dll

if(hLibrary)

{

// create a typedef for the function pointer based on the parameters and return type of the function to be called. This information can be obtained

// from the header file of the external dll. The name of the function to be called in the following code is "StartInstallEngine". We create a typedef for

// a function pointer named PFNSTARTINSTALLENGINE. Its return type is BOOL. Its parameters are HUSER, LPCTSTR, LPCTSTR, LPTSTR & LPTSTR.

// Substitute these with parameter and return types for your particular API.

typedef BOOL (*PFNSTARTINSTALLENGINE) (HUSER, LPCTSTR, LPCTSTR, LPTSTR, LPTSTR);

// Now create a variable for your function pointer of the type you just created. Then make call to GetProcAddress function with the first

// parameter as the handle to the library you just loaded. The second parameter should be the name of the function you want to call prepended

// with an "_", and appended with an "@" followed by the total number of bytes for your parameters. In this example, the total number of bytes in the

// parameters for "StartInstallEngine is 20 ( 4 bytes for each parameter ). The "GetProcAddress" API will return a pointer to the function that you need to

// call.

PFNSTARTINSTALLENGINE lpfnStartInstallEngine = (PFNSTARTINSTALLENGINE) GetProcAddress(hLibrary, "_StartInstallEngine@20");

if ( lpfnStartInstallEngine )

{

// Now call the API by passing in the requisite parameters.

lpfnStartInstallEngine(hUser, szObjectName, szVersionName, pszObjectText, szObjectType);

}

#endif
```

cded 呼出し規則

cded 呼出し規則を使用するプロセスは std 呼出し規則の場合と同様です。主な違いは GetProcAddress の 2 番目のパラメータです。この呼出しの前にあるコメントに注意してください。

```
# ifdef JDENV_PC

HINSTANCE hLibrary = LoadLibrary(_TEXT("YOUR_LIBRARY.DLL")); // substitute the name of the external dll

if(hLibrary)

{

// create a typedef for the function pointer based on the parameters and return type of the function to be called. This information can be obtained

// from the header file of the external dll. The name of the function to be called in the following code is "StartInstallEngine". We create a typedef for

// a function pointer named PFNSTARTINSTALLENGINE. Its return type is BOOL. Its parameters are HUSER, LPCTSTR, LPCTSTR, LPTSTR & LPTSTR.

// Substitute these with parameter and return types for your particular API.

typedef BOOL (*PFNSTARTINSTALLENGINE) (HUSER, LPCTSTR, LPCTSTR, LPTSTR, LPTSTR);

// Now create a variable for your function pointer of the type you just created. Then make call to GetProcAddress function with the first

// parameter as the handle to the library you just loaded. The second parameter should be the name of the function you want to call. In this

// case it will be "StartInstallEngine" only. The "GetProcAddress" API will return a pointer to the function that you need to call.

PFNSTARTINSTALLENGINE lpfnStartInstallEngine = (PFNSTARTINSTALLENGINE) GetProcAddress(hLibrary, "StartInstallEngine");

if ( lpfnStartInstallEngine )

{

// Now call the API by passing in the requisite parameters.

lpfnStartInstallEngine(hUser, szObjectName, szVersionName, pszObjectText, szObjectType);

}

#endif
```

注:

これらの呼出しは Windows クライアント・マシン上でのみ機能します。LoadLibrary と GetProcAddress は Windows の API です。ビジネス関数を Windows 以外のサーバーでコンパイルするとエラーが発生します。

J.D. Edwards ソフトウェアからの Visual Basic プログラムの呼出し

J.D. Edwards ビジネス関数から Visual Basic プログラムを呼び出し、Visual Basic プログラムから J.D. Edwards ビジネス関数にパラメータを渡すには、次のプロセスに従います。

- プログラムの関数名をエクスポートし、J.D. Edwards ビジネス関数にパラメータを返す Visual Basic の.dll に Visual Basic プログラムを組み入れます。
- 次に、win32 関数の LoadLibrary を使用して Visual Basic の.dll をロードするビジネス関数を作成します。
- 作成するビジネス関数で、win32 関数の GetProcAddress を呼び出し、Visual Basic 関数を取得して呼び出します。

ビジネス関数の処理

ビジネス関数はいずれも既定の構造と形式に従う必要があります。コードの各行は、J.D. Edwards ビジネス関数プログラミング標準に従って記述します。これらの標準に従うと、アプリケーション開発に J.D. Edwards の証明済みのソフトウェア・エンジニアリング・アプローチを活用することができます。次の手順に従ってビジネス関数を作成します。

- 〈Object Management Workbench〉を使用して、ビジネス関数データ構造体を作成する。
- 〈Object Management Workbench〉を使用して、ビジネス関数のソースファイルとヘッダー・ファイルを作成する。
- データ構造体のタイプ定義を作成して、ヘッダー・ファイルに追加する。

参照

- ビジネス関数のデータ構造体の作成については、『開発ツール』ガイドの「ビジネス関数データ構造体の作成」

Business Function Design へのアクセス

ここでは〈Business Function Design〉にアクセスする方法を説明します。

▶ 〈Business Function Design〉にアクセスするには

〈Object Management Workbench〉で、処理するビジネス関数をチェックアウトします。

1. ビジネス関数がハイライトされていることを確認し、中央カラムの[Design(設計)]ボタンをクリックします。
2. 〈Business Function Design〉で[Design Tools]タブを開いて、[Start Business Function Design Aid]をクリック選択します。

オブジェクト・コードの表示

オブジェクト・コードは、ビジネス関数をカテゴリに分類するために使用します。その後、これらのコードを検索用のフィルタ条件として使用できます。

▶ オブジェクト・コードを表示するには

1. 〈Business Function Design〉で、[Row]メニューから[Codes(コード)]を選択します。
2. 次のフィールドを確認します。
   - ソース・モジュール
   - 関数名
   - 記述
   - ビジネス関数カテゴリ
   - 機能用途

関連するテーブルと関数の追加

ビジネス関数には、テーブルと関数を関連付けることができます。ソース・ファイルとヘッダー・ファイルのコードを生成するするために、関連するテーブルと関数をビジネス関数オブジェクトに追加してください。

▶ 関連するテーブルを追加するには

1. 〈Business Function Design〉で、[Form]メニューから[Tables(テーブル)]を選択します。
2. 〈Related Tables(関連テーブル)〉で、関連するテーブルの名前を入力します。
3. [OK]をクリックします。

▶ 関連する関数を追加するには

1. 〈Business Function Design〉で、[Form]メニューから[Function(関数)]を選択します。
2. 〈Related Business Functions(関連ビジネス関数)〉で、関連するビジネス関数ソース・オブジェクトの名前を入力します。
3. [OK]をクリックします。

関連付けられたレコードが、オブジェクト・ライブラリアン – オブジェクト・リレーションシップ・テーブル(F9863)に保管されます。

ビジネス関数のデバッグ

イベント・ルール・ビジネス関数のソース・コードは C で生成されるので、C 言語と NER のビジネス関数のデバッグでは共に同じ手順に従ってください。

参照

- ビジネス関数のデバッグについては、『開発ツール』ガイドの「デバッグ」

ビジネス関数の親 DLL またはロケーションの変更

ビジネス関数では親 DLL の変更が必要になる場合があります。

▶ ビジネス関数の親 DLL を変更するには

1. 〈Business Function Design〉で、[Parent DLL]に新しい DLL 名を入力するか、またはビジネス関数のロケーションを変更して[OK]をクリックします。
2. 〈Object Management Workbench – [Business Function Design](オブジェクト管理ワークベンチ – [ビジネス関数の設計])〉で、[OK]をクリックします。
3. 〈Object Management Workbench〉で、ビジネス関数をチェックインします。

   このビジネス関数をできるだけ早くすべてのパス・コードに転送してください。

以上の変更は、次のパッケージで使用可能になります。

カスタム DLL の作成と指定

ビジネス関数の DLL は統合されています。そのため、カスタム・ビジネス関数はそれぞれ、作成するカスタム DLL に組み込む必要があります。これにより、カスタム・ビジネス関数は、J.D. Edwards ビジネス関数から独立したものになります。構築プログラムは、Object Librarian ヘッダー・ファイル(F9860)を参照して、カスタム DLL の有無を確認します。

カスタム DLL の作成

ビジネス関数をカスタム DLL に組み込むには、そのカスタム DLL があらかじめ Object Librarian マスター・テーブル(F9860)に存在している必要があります。したがって、カスタム DLL を作成する必要があります。

▶ カスタム DLL を作成するには

1. 〈Object Management Workbench〉で、[Add]をクリックします。
2. 〈Add J.D. Edwards Object to the Project〉で、[Business Function Library(ビジネス関数ライブラリ)]オプションを選択し、[OK]をクリックします。
3. [Add Object(オブジェクトの追加)]で、次のフィールドに入力し、[OK]をクリックします。
   - オブジェクト名
   - 記述
   - システム・コード
   - 製品システム・コード
   - オブジェクト使用

カスタム・ビジネス関数に対するカスタム DLL の指定

カスタム・ビジネス関数を作成したときには、1 つのカスタム DLL を指定してください。指定しないと、カスタム・ビジネス関数はデフォルトの J.D. Edwards CALLBSFN DLL に自動的に組み込まれます。

▶ カスタム・ビジネス関数用のカスタム DLL を指定するには

カスタム DLL を作成した後は、そのカスタム DLL を指定する必要があります。

1. カスタム DLL の名前を[Parent DLL]フィールドに入力して、[OK]をクリックします。

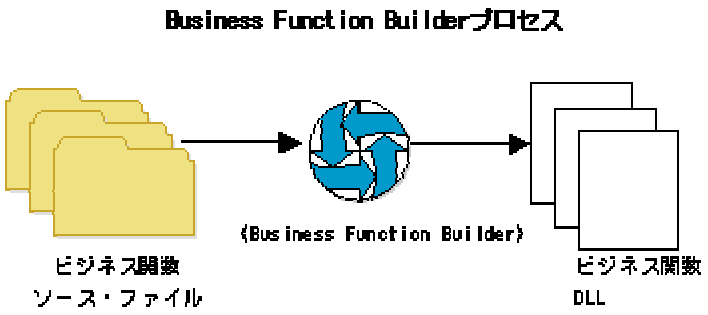
   注:
   必要な場合は、ビジネス関数のロケーションを変更することもできます。

2. ビジネス関数のビルドを実行します。

Business Function Builder の処理

ビジネス関数コードをダイナミック・リンク・ライブラリ(DLL)に組み入れるには、〈Business Function Builder〉を使用します。これにより、C 言語ビジネス関数、イベント・ルール・ビジネス関数およびテーブル・イベント・ルールをビルドすることができます。〈Business Function Builder〉を実行してビジネス関数をビルドすると、コンパイルとリンクの 2 つのプロセスが実行されます。コンパイルにより、ビジネス関数オブジェクトが作成され、リンクによりそのオブジェクトが DLL の一部になります。

次の図に、ビジネス関数のビルド・プロセスを示します。



〈Business Function Builder〉フォームについて

〈Business Function Builder〉フォームは、次の 3 つの主要コンポーネントで構成されています。

- Functions List(関数リスト)
- Available Business Functions(使用可能なビジネス関数)
- Build Output(ビルド出力)

これらのコンポーネントに加えて、フォーム下部にステータス・ウィンドウも表示されます。このステータス・ウィンドウには、処理中のビルドの状況が表示されます。ステータス・バーの[Stop(停止)]ボタンをクリックすると、処理を中断することができます。

ツールバーのコンボ・ボックスは[Optimize(最適化)]モードに設定してください。デバッグする場合は、[Debug(デバッグ)]モードに設定する必要があります。

Functions List

[Functions List]には、カスタム DLL を含む、J.D. Edwards の統合 DLL が表示されます。ここには、構築時にリンクまたはコンパイルすることができる関数がリストアップされます。このリストにドラッグした関数はいずれも[Functions]リストの作成時に構築されます。

Available Business Functions

[Available Business Functions]には、ビルドできる関数が表示されます。各々のビジネス関数 DLL (たとえば、CCUSTOM や COBLIB など)には、構築可能なビジネス関数の固有のセットがあります。

Build Output

〈Build Output〉には、処理結果が表示されます。ビジネス関数のコンパイルとリンク時には、〈Business Function Builder〉が〈Build Output〉を更新して、処理の進捗状況を表示します。この情報は、ビルド中に発生した問題の診断が必要な場合に役立ちます。処理が完了すると、〈Business Function Builder〉が、処理が正常終了したかどうかを示します。フォームの内容を表示するには、スクロール・バーを使用します。またフォームに表示されているテキストの切取り、貼付け、印刷も可能です。

単一のビジネス関数の構築

〈Business Function Builder〉は通常、個別にビジネス関数をビルドするのに使用します。ビジネス関数のソース・コードを変更するたびに、ビジネス関数をビルドしてテストする必要があります。

▶ 1 つのビジネス関数をビルドするには

1. 〈Object Management Workbench〉で、ビルドするビジネス関数を選択し、中央カラムの[Design]ボタンをクリックします。
2. 〈Business Function Design〉で、[Design Tools]タブを選択して[Busbuild Standalone]をクリックします。

   〈BusBuild〉フォームが表示され、〈Business Function Builder〉がビジネス関数のビルドを開始します。

〈Build Output〉に処理結果が表示されます。処理が完了すると、〈Build Output〉の下端に「***Build Finished***」というメッセージが表示されます。この行の下に、処理が正常終了したかどうかを示すテキストが表示されます。処理が正常終了した場合はビジネス関数をテストできます。正常終了しなかった場合は、問題を解決して再び処理する必要があります。

処理を中止するには、フォームの右下隅の[Stop]ボタンをクリックします。DLL が元の状態に戻ります。

ビジネス関数オブジェクトのリンク

〈Object Management Workbench〉でコンピュータに 1 つ以上の J.D. Edwards アプリケーションをインストールすると、ビジネス関数オブジェクトもインストールされ、〈Business Function Builder〉が呼び出されて Link All 操作が実行されます。この操作により、ビジネス関数オブジェクトがローカルのビジネス関数 DLL と統合されます。リンク操作では、DLL を置き換える代わりにローカルのカスタム修正を保持します。この処理は、カスタム修正がマシンに存在しない場合は不要です。

注:

[Link All]は個々の DLL をリンクするだけで、ビジネス関数のコンパイルは行いません。

構築オプションの設定

[Build]メニューのオプションを使用すると、統合ビジネス関数の構築方法を制御することができます。次のオプションを使用できます。

Build makefile を生成し、選択されたビジネス関数をコンパイルして、現在の統合 DLL に関数をリンクします。古いコンポーネントのみをリビルドします。

Compile makefile を生成し、選択されたビジネス関数をコンパイルします。アプリケーションは、現在の統合 DLL に関数をリンクしません。

ANSI Check 選択したビジネス関数の ANSI 互換性をチェックします。

Link 各統合 DLL の makefile を生成します。その後で統合 DLL を構築します。アプリケーションは、選択したビジネス関数をコンパイルしません。

Link All 各統合 DLL の makefile を生成します。その後で統合 DLL を作成し、呼び出されるすべてのビジネス関数をその DLL リンクします。アプリケーションは、選択したビジネス関数をコンパイルしません。

Rebuild Libraries 統合 DLL と.obj ファイルからの Static ライブラリを再構築します。

Build All 各 DLL のすべてのオブジェクトをコンパイルしてリンクします。

Stop Build 構築処理を中断します。既存の統合 DLL には、一切手が加えられません。

Suppress Output 〈Build Output〉に表示されるテキストを制限します。

Browse Info ビジネス関数をコンパイルするときにブラウズ情報を生成します。このオプションを OFF に設定すると処理が高速化します。

Precompiled Header ビジネス関数をコンパイルするときにプリコンパイル済みのヘッダーを生成します。複数のビジネス関数をコンパイルするときに、プリコンパイル済みのヘッダーを使用すると、一般にコンパイル処理が高速化します。

Debug Info コンパイル時にデバッグ情報を生成します。Visual C++では、デバッグ情報を使って構築された関数をデバッグすることができます。このオプションを OFF に設定すると、処理が高速化します。

Full Bind 各々の J.D. Edwards 統合 DLL に対するすべての外部ランタイム参照を解決します。

ユーティリティ・プログラムの使用

[Tool(ツール)]メニューには、ビルド処理を支援する複数のユーティリティ・プログラムが登録されています。これらのプログラムを次に示します。

Synchronize JDEBLC 〈Synchronize JDEBLC(JDEBLC の同期化)〉プログラムを実行して、J.D. Edwards ビジネス関数を新規 DLL グループに再編します。このプログラムは、ローカル JDEBLC 親スペック・テーブルの DLL フィールドと、オブジェクト・ライブラリアン・マスター(F9860)の親 DLL を同期化します。このプログラムは注意して使用してください。このプログラムは通常、新しいパッケージ・ビルドからビジネス関数 DLL を手動でドラッグした後に、ビジネス関数ロード・ライブラリで問題が発生した場合のみ使用します。

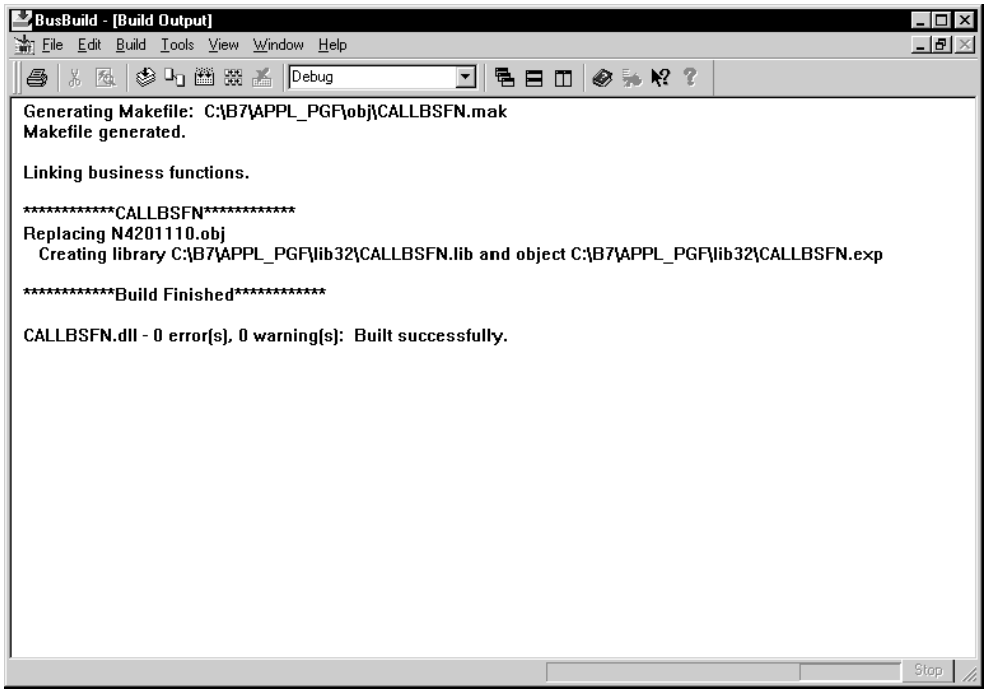
Dumpbin このプログラムを実行して、特定のビジネス関数が正しくビルドされているかどうかを確認します。選択された統合 DLL に組み入れられたすべてのビジネス関数が表示されます。

PDB (Program DeBug file) Scan リンクしようとするオブジェクト・ファイルのいずれかが PDB 情報によって正しくコンパイルされないと、重大な CVPACK エラーが発生します。この問題を解決するために〈PDB Scan〉を使用すると、PDB 情報を使用してビルドされたオブジェクト・フィールドを特定することができます。〈PDB Scan〉が報告したビジネス関数をリコンパイルしてください。

〈Build Output〉の読み取り

〈Build Output〉は、処理状況に関する重要な情報を表示する一連のセクションから構成されています。この情報を使用すると処理が正常終了したかどうかを確認でき、処理中にエラーが発生した場合はトラブルシューティングを実行できます。

次の図に、処理中に生成されるメッセージの一例を示します。



Makefile セクション

makefile セクションには、〈Business Function Builder〉が特定の処理で makefile を生成した場所が表示されます。〈Business Function Builder〉は処理する DLL ごとに makefile を 1 つ生成します。処理している DLL ごとに「Generating Makefile（makefile の生成中）」というステートメントが常に表示されます。makefile ステートメントが表示されない場合はエラーが発生しています。エラーを解決するには次の操作を実行してください。

- ローカル・オブジェクト・ディレクトリが存在するか確認します。
- ローカル・オブジェクト・ディレクトリの権限と makefile に問題がないか確認します。

Begin DLL セクション

Begin DLL セクションは、〈Business Function Builder〉が特定の DLL を処理していることを示します。たとえば、このセクションの上のセクションが「*****CDIST*****」から始まる場合があります。Begin DLL セクションは、処理している DLL ごとに表示されます。

Compile セクション

DLL をビルドする前に〈Business Function Builder〉は DLL のビジネス関数をコンパイルします。〈Business Function Builder〉がコンパイルしようとする各ビジネス関数の順次編成リストが表示されます。コンパイル処理の過程では、次の警告やエラーが発生することがあります。

- コンパイラ警告

  コンパイラ警告が発生すると、〈Business Function Builder〉は、「warning CXXXX」(XXXX は数値)というメッセージと警告に関する簡単な説明文を表示します。警告に関する情報を参照するには、Visual C++のオンライン・ヘルプで、CXXXX の値を検索します。警告が発生しても、通常、ビジネス関数は、正常にコンパイルできます。ただし、〈Global Build(グローバル・ビルド)〉フォームで[Warnings As Errors(エラーとしての警告)]オプションをオンにすると、警告が発生した場合にビジネス関数の構築を中止できます。

- コンパイラ・エラー

  コンパイラ・エラーが発生すると、〈Business Function Builder〉は、「error CXXXX」(XXXX は数値)というメッセージとエラーに関する簡単な説明文を表示します。エラーに関する詳細情報を参照するには、Visual C++のオンライン・ヘルプで CXXXX の値を検索します。エラーが発生した場合はビジネス関数を正常にコンパイルできないので、エラーを解決する必要があります。

Link セクション

〈Business Function Builder〉は、DLL のビジネス関数のコンパイルを終了するとそれらをリンクします。このリンク処理では、DLL の.lib ファイルと.dll ファイルを作成します。リンク処理の過程では、次の警告やエラーが発生することがあります。

- リンカ警告

  リンカ警告が発生すると、〈Business Function Builder〉は、「warning LNKXXXX」(XXXX は数値)というメッセージと警告に関する簡単な説明文を表示します。警告に関する情報を確認するには、Visual C++のヘルプで LNKXXXX の値を検索します。警告が発生しても通常、ビジネス関数は正常にリンクできます。〈Global Build〉フォームで[Warnings As Errors]オプションをオンにすると、警告が発生した場合に DLL の構築を中止できます。

- リンカ・エラー

  リンカ・エラーが発生すると、〈Business Function Builder〉は「error LNKXXXX」(XXXX は数値)というメッセージとエラーに関する簡単な説明文を表示します。エラーに関する詳細情報を参照するには、Visual C++のヘルプで LNKXXXX の値を検索します。エラーが発生してもそれが軽度のエラーであれば、〈Business Function Builder〉は DLL を作成します。ただし、DLL の構築時にエラーが発生したことを知らせるメッセージを表示します。重大なエラーが発生した場合、〈Business Function Builder〉は DLL を構築しません。

Rebase セクション

Rebase セクションには、再調整に関する情報が表示されます。〈Rebase〉は、DLL のパフォーマンスを微調整してそれらのロード時間を短縮します。このために、DLL のロード・アドレスを変更し、システム・ローダがイメージを再配置しなくても済むようにします。システムは DLL 全体を自動的に読み取り、またフィックス、デバッグ情報、チェックサム情報、タイム・スタンプ値を更新します。

Summary セクション

Summary セクションには、構築に関する最も重要な情報が表示されます。このセクションは、処理が無事終了したかどうかを示します。このセクションは、「*****Build Finished*****」から始まります。ビルドを試みた DLL ごとの要約レポートも表示されます。このレポートには、次の項目が表示されます。

- 警告数
- エラー数
- DLL の構築が無事に終了したかどうか

すべてのビジネス関数のビルド

[Build All(すべてをビルド)]を使用すると、すべてのビジネス関数をビルドすることができます。[Build All]は、グローバル・リンクと同じ操作を実行するだけではなく、各 DLL のすべてのオブジェクトもリコンパイルします。[Build All]は通常、システム管理者が実行します。[Build All]では、処理時間が長くなることがあります。[Build All]を実行するには、J.D. Edwards の[Explorer(エクスプローラ)]メニューから〈BusBuild〉にアクセスする必要があります。

▶ すべてのビジネス関数をビルドするには

1. 〈Object Management Workbench〉で、ビルドするビジネス関数を選択し、中央カラムの[Design]ボタンをクリックします。
2. 〈Business Function Design〉で、[Design Tools]タブを選択して[Busbuild Standalone]をクリックします。

   〈BusBuild〉フォームが表示されます。
3. 〈BusBuild〉で、[Build]メニューから[Build All]を選択します。
4. 〈Build Mode(ビルド・モード゛)〉の次のオプションを 1 つ選択します。

Debug デバッグ情報を含むビルド。処理を実行した後、構築されたビジネス関数を Visual C デバッガでデバッグすることができます。

Optimize デバッグ情報を含まないビルド。一般に Visual C デバッガを使用してデバッグできません。

Performance Build 最適化ビルドと同じですが、ビジネス関数のパフォーマンスの測定に役立つ情報を含みます。このオプションは、J.D. Edwards の開発者のみ使用してください。

5. [Source Directory(ソース・ディレクトリ)]フィールドに入力します。

   このフィールドでは、ビジネス関数ソースが常駐する場所を指定します。ビジネス関数ソースには、.c、.h、イベント・ルール・ビジネス関数、テーブル・イベント・ルールがすべて含まれます。フル・パッケージには通常、すべてのビジネス関数ソースが含まれます。

   ディレクトリを参照するには、このフィールドの右側のボタンをクリックします。

次のロケーションを 1 つ選択して、ビジネス関数ソースの保管場所を指定します。

Local（ローカル） すべてのビジネス関数ソースがローカル・マシン上に置かれます。

Path Code（パス・コード） すべてのビジネス関数ソースが、選択したパスコードによって指定されているパスに置かれます。

Package（パッケージ） すべてのビジネス関数ソースが、選択したパッケージによって指定されているパスに置かれます。パッケージが正しく構築された場合は、通常、必要なビジネス関数ソースをすべて含みます。ロケーションには[Package]を使用することをお勧めします。

Pick Directory ファイル・サーバー上の別のディレクトリにビジネス関数ソースが保管されます。ディレクトリを指定してください。

たとえば、ファイル・サーバー上のロケーション¥¥SERVER¥SHARE1¥PROD_A に PROD_A というパッケージが物理的に存在し、次のサブディレクトリが含まれているとします。

Source すべてのビジネス関数の.c が入っています。

Include すべてのビジネス関数の.h が入っています。

Spec すべてのスペック・ファイルが入っています。

サーバー上での PROD_A の構成がこのような場合は、ボタンをクリックして[Package]を選んでグリッドから[PROD_A]を選択します。〈Business Function Builder〉の実行時には、このディレクトリからビジネス関数ソースを使用します。

6. [Foundation Directory（ファンデーション・ディレクトリ）]フィールドに入力します。

このフィールドでは、このビルドで使用されるファンデーションを指定します。選択したファンデーションが、これらのビジネス関数が動作するファンデーションになります。

このフィールドの右側のボタンをクリックして、ロケーションを参照します。

次のロケーションの 1 つからそのファンデーションを指定することができます。

Local 推奨ファンデーションは、ローカル J.D. Edwards ファンデーションです。

Foundation 登録済みの J.D. Edwards ファンデーションの一覧がファンデーション・テーブルに表示されます。このテーブルからファンデーションを選択します。

Pick Directory J.D. Edwards ファンデーションは、ファイル・サーバー上のディレクトリに保管されます。ディレクトリを指定してください。このロケーションを使用することをお勧めします。

たとえば、新規に作成したファンデーションがサーバー上のロケーション ¥¥SERVER¥SHARE1¥System¥New¥Optimize にあるとします。このディレクトリには、次のサブディレクトリをもちます。

Lib32 すべてのファンデーションの.lib が入っています。

Include すべてのファンデーションの.h が入っています。

Includev すべてのベンダの.h が入っています。

7. [Output Destination Directory(出力先ディレクトリ)]フィールドに入力します。

   このフィールドを使用して、ビルドの出力先を指定します。ビルド出力のファイル・タイプには、.DLL、.LIB、.OBJ、.LOG があります。

   このフィールドの右側のボタンをクリックして、ロケーションを参照します。

   次のロケーションを 1 つ選択して、出力先のディレクトリを指定することができます。

Local ローカル・マシンにすべてのビジネス関数出力が保管されます。

Path Code 選択したパス・コードによって指定されているパスにすべてのビジネス関数出力が保管されます。このパス・コードには、チェックイン済みの最新のビジネス関数に対する変更が保管されます。

Package 選択したパッケージにすべてのビジネス関数出力が保管されます。パッケージは、ビジネス関数ソースの安定的なスナップショットです。出力先ディレクトリのロケーションには[Package]を指定することをお勧めします。

Pick Directory ファイル・サーバー上のディレクトリにビジネス関数出力が保管されます。ディレクトリを指定してください。

たとえば、ファイル・サーバー¥¥SERVER¥SHARE1¥PROD_A 上にパッケージ PROD_A があるとします。ビルドが完了したら、〈Business Function Builder〉で、ビルド出力を PROD_A の次のサブディレクトリに置くように指定する必要があります。

Bin32 built .dll ファイルを格納します。

Lib32 build .lib ファイルを格納します。

Obj built .obj ファイルを格納します。

Work built .log ファイルを格納します。

サーバー上での PROD_A の構成が上記のような場合は、ボタンをクリックして[Package]を選んでグリッドから[PROD_A]を選択します。〈Business Function Builder〉は、処理の際に PROD_A のこれらのサブディレクトリにビルド出力ファイルを配置します。

8. 次のいずれかのオプションをクリックします。

Treat Warnings As Errors(警告はエラーとして対処)	このオプションをオンにすると、〈Business Function Builder〉は警告が発生した場合にビジネス関数をビルドしません。
Clear Output DestinationBefore Build(出力先をビルド前にクリア)	このオプションをオンにすると、〈Business Function Builder〉はビジネス関数をビルドする前に、bin32、lib32、obj の各出力ディレクトリの内容を削除します。
Select Which DLLs to Build(ビルドする DLL の選択)	このオプションをオンにしないと、〈Business Function Builder〉はすべての DLL をビルドします。このオプションをオンにすると、[Select]ボタンをクリックして、ビルドするビジネス関数 DLL を選択できます。このオプションは、1 つ以上の DLL を構築する場合に使用します。すべての DLL のサブセットのみをビルドする場合は、[Clear Output Destination Before Build (ビルド前の出力先のクリア)]オプションがオフになっていることを確認します。
Stop Level(ストップ・レベル)	ビルドを中止するエラー・レベルを選択できます。エラーが発生しても無視して、ビルドを継続できます。DLL にエラーがある場合は、ビルド処理を中止するように指定できます。最初のコンパイル・エラーで中止できます。
Generate "Missing Source" Report (ソース無レポートの生成)	このオプションをオンにすると、〈Business Function Builder〉は出力先の作業ディレクトリでレポートを生成します。このレポートは、NoSource.txt という名前で、.c ファイルを持たずオブジェクト・ライブラリアン・マスター(F9860)にレコードが存在するビジネス関数ソース・ファイル名を表示します。このレポートの情報を解決するには、ビジネス関数で該当する名前の.c ファイルを作成するか、または F9860 テーブルからソース・ファイルを削除します。このオプションはオンにすることをお勧めします。
Generate ER Source (ER ソースの生成)	このオプションをオンにすると、〈Business Function Builder〉は、ビジネス関数をビルドする前に、イベント・ルール・ビジネス関数とテーブル・イベント・ルールのソースを生成します。
Verify Check-in (チェックインの検証)	指定のパス・コードにチェックインされたオブジェクトのみがビルドされます。Notchkdn.txt というログ・ファイルが、Nosource.txt と同じディレクトリに書き出されます。パス・コードにチェックインされていないオブジェクトが、このログと Buildlog.txt にリストされます。

[From RDB(RDB から)]オプションをオンにすると、任意のパス・コードからビルドすることができます。このオプションをオフにすると、〈Business Function Builder〉は、イベント・ルールのソースがソース・ディレクトリのスペック・ファイルから生成できると想定します。

〈Package Build(パッケージ・ビルド)〉によって開始されたビルドのトラブルシューティングを行う場合は、上記の設定にあらかじめ適正な値を設定しておいてください。そうすれば、[Build]をクリックするだけで、問題のある DLL を再構築することができます。

注:

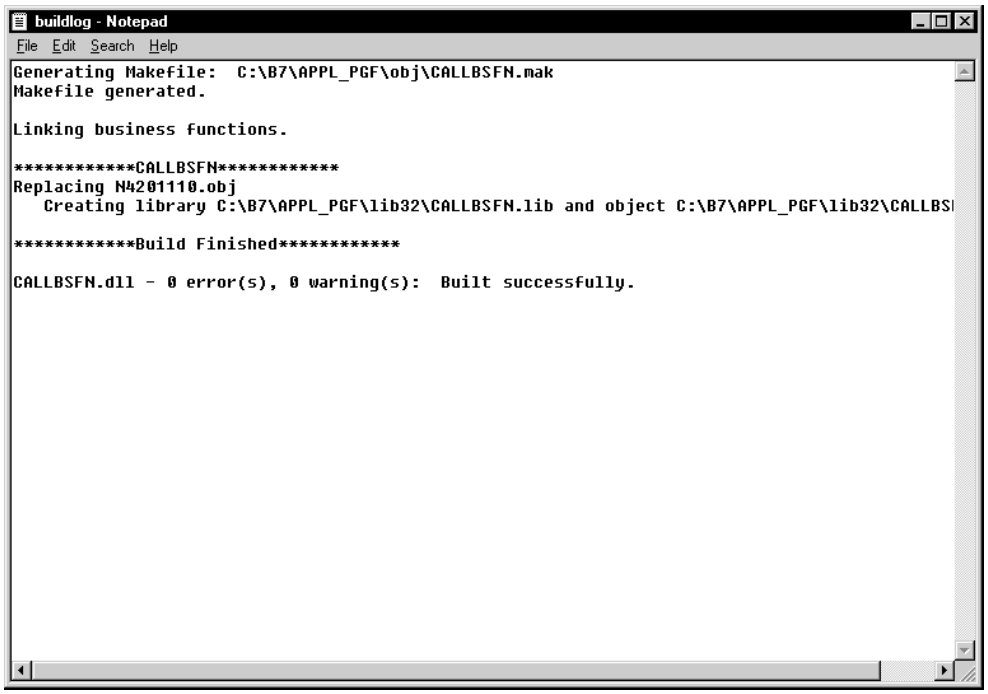
このビルドは、パッケージ・ビルドで[Build BSFN(ビジネス関数のビルド)]オプションをオンに設定して実行することもできます。

エラー出力の検討

オブジェクトの構築時には、作業ディレクトリが自動的に作成されます。このディレクトリは、指定した出力先ディレクトリ（C:¥b7¥appl_pgf¥work¥buildlog.txt など）に配置されます。

このディレクトリには、エラーと情報ログが入っています。Buildlog は、〈Business Function Builder〉の〈Build Output〉フォームと同じ情報を格納します。

次の例に、サンプル・ログを示します。



エラーの解決

〈Business Function Builder〉ツールを使用すると、エラーの解決に役立ちます。ビジネス関数のビルド時に未解決の外部エラーに関するメッセージが表示された場合でも、統合 DLL はビルドされ、J.D. Edwards ソフトウェアは正常に動作します。ただし、エラーが解決されていないビジネス関数を実行できません。

特定のビジネス関数が統合 DLL に存在するかどうかを確認するには、dumpbin ツールを使用します。ビジネス関数が存在する場合は、その名前が dumpbin 出力で表示されます（名前の後にカッコで囲まれたゼロ以外の数値が付けられます）。

重大な CVPACK エラーを解決するには、PDB スキャンを使用します。〈Business Function Builder〉が、PDB（プログラム・デバッグ・ファイル）情報によって構築されたオブジェクト・ファイルをリンクしようとすると、CVPACK エラーが発生します。PDB スキャンは、問題のオブジェクト・ファイルを検出します。このようなオブジェクト・ファイルは、〈Business Function Builder〉を使ってマシン上でリコンパイルする必要があります。

ビジネス関数は、Visual C++を使用してコンパイルすると正しく動作しません。〈Business Function Builder〉の外部で構築されたビジネス関数を識別するには、PDB スキャンを使用することができます。これらの関数は、〈Business Function Builder〉を使って再構築すると正しく動作します。

DLL のいずれかが同期しない場合は、[Build]オプションを使用して再構築する必要があります。このオプションは makefile を生成し、DLL のすべてのビジネス関数を再リンクします。

〈Business Function Builder〉の[Tools]メニューに登録されている[Synchronize JDEBLC(JDEBLC の同期化)]オプションは、配置や構築に問題のあるビジネス関数を解決します。このオプションは、サーバーDLL を参照し、ローカル・ワークステーションのスペックとサーバーのスペックが一致するかどうか確認します。一致しない場合は、〈Business Function Builder〉がサーバー上の適切な DLL でビジネス関数を再構築して再リンクします。

〈Build Log(ビルド・ログ)〉には、次のセクションがあります。

Build Header ソース・パス、ファンデーション・パス、出力先パスを含む、特定のビルドの構成が表示されます。

Build Messages コンパイル動作とリンク動作が表示されます。コンパイル時には、コンパイルされたビジネス関数ごとに 1 行ずつ出力されます。コンパイル・エラーはいずれも「error cxxxx」と表示されます。リンク処理の過程では、テキスト「Creating library ....」が出力されます。このテキストの後に、リンカ警告またはエラーが続く場合があります。

Build Summary 最後のセクションには、各 DLL のビルドの要約が表示されます。この要約は、「x error(s), x warnings (y)」という形式で表示され、ビルドの状況を示します。警告とエラーがなければ、ビルドは完了です。エラーが表示された場合は、ログで作業エラーを検索し、エラーの原因を究明します。ビルドのエラーは、構文エラーやファイルの欠如といったエラーが一般的です。

トランザクション用マスター・ビジネス関数

トランザクション用マスター・ビジネス関数は、レコードが相互に依存するトランザクション・テーブルに必要なすべてのデフォルト値と編集機能を提供する共通の関数セットです。この種の関数は、データベースへの挿入、更新または削除されるトランザクションの整合性を維持するロジックを含んでいます。イベント・フローはロジックを制御します。処理されるレコードを保管するには、Cache API を使用します。次の場合は、トランザクション用マスター・ビジネス関数の使用を検討してください。

- トランザクション・ファイル・レコードを J.D. Edwards 以外のソースから受け入れる。
- 複数のアプリケーションが同じトランザクション・ファイルを更新する。

次のトランザクション・テーブルは、トランザクション用マスター・ビジネス関数の候補の一例です。

- 取引明細テーブル(F0911)が、アプリケーション・スイートと外部ソースでの更新を受け入れる。
- 従業員トランザクション詳細ファイル・テーブル(F06116)が、バッチ・ソース、対話型ソース、および外部ソースからの更新を受け入れる。

マスター・ビジネス関数の命名規則

マスター・ビジネス関数のソースには、(他のビジネス関数と同様に)Bxxxxxxxx という名前を設定します。ただし、オブジェクトがファンクショナル・サーバーとして AS/400 に既に存在する場合は、命名規約の XTXXXXZ1 を使用します。このソースは、各モジュールの外部ビジネス関数に適用されます。

各モジュールのデータ構造体は、DxxxxxxxxA,B,C という名前にします(たとえば、B0400047 - D0400047A [Begin Document(ドキュメントの開始)]、D0400047B [Edit Line(行の編集)]、D0400047C [Edit Document(ドキュメントの編集)]、D0400047D [End Document(ドキュメントの終了)]など)。

個別ビジネス関数の名前の形式は file-name module-name です。Document は Doc に短縮します(たとえば、F0411BeginDoc など)。

関数用途コード 192 は、ビジネス関数をマスター・ビジネス関数として指定します。

キャッシュはトランザクション・レコードをトラッキングします。キャッシュの命名規則は FxxUIyyy です。

xx - システムコード

yyy - 固有の番号

マスター・ビジネス関数の処理オプションの作成

MBF(マスター・ビジネス関数)は、さまざまなアプリケーションから呼び出すことができます。MBF の処理オプションをアプリケーションごとに複製するのではなく、通常は、独立した処理オプション・テンプレートを作成してください。

▶ マスター・ビジネス関数の処理オプションを作成するには

1. 処理オプション・テンプレートを作成します。

   処理オプション・テンプレートの名前は、Txxxxxxxx という形式にします(MBF 名の B または XT を T に置き換えます)。

2. MBF 処理オプションを関連付けるためのダミー・アプリケーションを作成します。

   これで、異なるバージョンの MBF 処理オプションをバージョン・リストを通じて設定できます。このダミー・アプリケーションには、Pxxxxxxxx という名前を設定します(MBF 名の B または XT を P に置き換えます)。このアプリケーションを生成するには、フィールドが選択されていないフォームが 1 つだけ必要です。

異なるバージョンのマスター・ビジネス関数処理オプションを設定するには、対話型バージョンを使用できます。これで、各種の呼出しプログラムにより、バージョン名がバージョン・パラメータの BeginDoc に渡されます。

処理オプションは、BeginDoc では、ビジネス関数の AllocatePOVersionData を呼び出して、バージョン名によりその処理オプション値を取り出すことができます。他のモジュールで必要な処理オプションは、ヘッダー・キャッシュに書き出されるので、AllocatePOVersionData を何度も呼び出さずに後でアクセスさせることができます。

トランザクション用マスター・ビジネス関数モジュールの命名

関数名には、標準命名規則の Fxxxxx を使用します。テーブルに複数のマスター・ビジネス関数が存在する場合は、次のようにプログラム名を関数名に含めます。

FxxxxxProgram Name Module

次に例を示します。

- F3112SuperBackFlushBegDoc
- F3112WorkOrderRoutingsBegDoc

トランザクション用マスター・ビジネス関数の要素

マスター・ビジネス関数には通常、次の基本要素を使用します。

Begin Document	ヘッダー情報の入力が完了したときに呼び出されます。 • 初期ヘッダーを作成します（作成されていない場合）。 • デフォルト、編集、処理の各オプションを含めることもできます。
Edit Line	ロー情報の入力が完了したときに呼び出されます。 • 詳細情報用のキャッシュを作成します（作成されていない場合）。
Edit Document	トランザクションをコミットするときに呼び出されます。 • 文書に対する残りの編集を処理し、すべてのレコードにコミットできることを確認します。
End Document	トランザクションをコミットする必要があるときに呼び出されます。 • ヘッダーおよび明細キャッシュのすべてのレコードを処理し、入出力を実行してキャッシュを削除します。
Clear Cache	すべてのキャッシュ・レコードを削除するときに呼び出されます。 • ヘッダーおよび明細キャッシュ・レコードを削除します。

Begin Document

関数名

FxxxxxBeginDocument

機　能

Begin Document 要素は次のように機能します。

- データ辞書のデフォルトや UDC 編集機能などのデフォルト情報を、ヘッダーに挿入して編集します。
- 必要に応じてデータベースから情報を取り出し、選択した文書処理の可否を確認します。
- すべてのレコードに共通の情報を検査して処理します。
- エラーが存在しない場合、レコードをヘッダー・キャッシュに書き出します。

- すべての詳細レコードに共通の情報が格納されます。これにより、同様の検査とテーブル入出力を実行する際にすべての詳細レコードを使用する必要がなくなるため、パフォーマンスが向上します。
- ヘッダー・フィールドの情報が変更され、Begin Document が既に呼び出されているときにヘッダー・キャッシュを更新して、新しい情報を反映します。

特別に必要なロジックまたは処理

この関数は、最初の呼出しでジョブ番号を割り当てます。ジョブ番号を取り出すために、この関数は、システム・コード 00 とインデックス番号 04 を使用して X0010GetNextNumber を呼び出します。Begin Document が再び呼び出されると、既に割り当てられているジョブ番号を受け渡します。そのため、別のジョブ番号を割り当てる必要がありません。

フックアップ・ヒント

- この関数は、Edit Line を呼び出す前に少なくとも 1 度呼び出します。
- この関数を呼び出したときのヘッダー・フィールドの検査でエラーが発生した場合は、この関数を再度呼び出して、Edit Line を呼び出す前にエラーがクリアされたことを確認します。
- この関数を異なるイベントから繰り返し呼び出す場合は、この関数をアプリケーションの隠しボタンに含めて、重複コードを削減し、整合性を確保してください。このボタンは、ユーザーが明細レコードを追加/削除してヘッダー情報の追加を終了するために、グリッドのフォーカスから呼び出されることがあります。また、ユーザーがグリッドを使用しない［Copy（コピー）］操作の場合、このボタンは［OK］ボタンで呼び出すこともできます。
- 非同期イベントからボタンを呼び出すと非同期フローが中断され、ボタンが同期モード（インライン）で処理されます。

共通のパラメータ

名前	エイリアス	I/O	説　明
Job Number	JOBS	I/O	その関数が既に呼び出されている場合には、BeginDocument で作成されたジョブ番号を受け渡します。呼び出されていない場合は、ゼロを渡し、その関数にジョブ番号を割り当てます。
Document Action	ACTN	I	A または 1 = 追加 C または 2 = 変更 D = 削除 これは、個々の明細行ではなく、そのドキュメント全体に対する操作です。たとえば、［Edit Line］では、数行の明細行を修正/追加/削除することがありますが、そのような場合は、Begin Document の Document Action を「変更」にします。

Process Edits	EV01	I	オプション 0 = 編集なし 0 以外 = フル編集 **注:** 通常、GUI インターフェイスでは部分編集を使用し、バッチ・インターフェイスではフル編集を使用します。このパラメータがブランクの場合、デフォルト・オプションはフル編集です。
ErrorConditions	EV02	O	ブランク = エラーなし 1 = 警告 2 = エラー
Version	VERS	I	このフィールドは、この MBF がバージョンを使用する場合には必ず指定します。
Header Field One	****	I/O	ドキュメント全体に共通のすべてのヘッダー・フィールドを受け渡します。Begin Document は、それらのフィールドをすべて処理して検査し、さらにデータ辞書の編集、UDC の編集、デフォルト値などもチェックします。また、Begin Document は、テーブルへのフェッチを実行し、これらのヘッダー・フィールドに一致するレコードが、「削除」と「変更」の場合には存在し、「追加」の場合には存在しないことを確認します。
Header Field Two	****	I/O	
. . .	****	I/O	
Header Field XX	****	I/O	
Work Field / Processing Flag One	****	I	プログラムが必要とするワーク・フィールドをすべてリストします。これらは、処理や検査日などのフラグでもかまいません。これらのフィールドは、使用されることもされないこともあります。たとえば、通貨制御をヘッダー・キャッシュに保存して、すべての詳細レコードが通貨を使用するように、または使用しないようにすることがあります。
Work Field / Processing Flag One	****	I	
. . .		I	
Work Field / Processing Flag One		I	

アプリケーション固有のパラメータ

- ヘッダー・レベルの情報処理に必要なフィールドをリストします。
- 編集の実行に必要なワーク・フィールドをリストします。
- ヘッダー・レベルの情報処理に必要なすべての処理オプションをリストします。

Edit Line

関数名

FxxxxxEditLine

機　能

Edit Line 要素は次のように機能します。

- すべてのユーザー入力を検証し、計算を実行し、デフォルトの情報を取り出します。Edit Line は通常、取り出されるレコードごとに呼び出され、ファイルでその単一レコードに対する編集を実行します。
- デフォルト値のヘッダー・キャッシュ・レコードを読み取ります。
- ADD では、住所録情報などのデフォルト情報をブランク・カラムに入力します。デフォルト値は、次の要素から得ることができます。
  - 行の別のカラム
  - 行に渡されたカラム上で実行された処理
  - 処理オプション
  - Begin Document モジュールで指定されたヘッダー・レコードからの保存値
  - データ辞書のデフォルト値
- 適正な情報に合わせてカラムを編集します。これには、カラム間の相互依存編集が含まれます。また、UDC とデータ辞書の編集を実行します。
- エラーが発生しなかった場合にレコードを明細キャッシュに書き出します。レコードが既にワーク・ファイルに存在する場合はワーク・ファイルの行が取り出され、変更内容に従って更新されます。レコードがダイレクト・モードでグリッドから削除され、そのレコードがデータベースに存在しない場合は、明細キャッシュからも削除されます。レコードがデータベースに存在する場合は、レコードのアクション・コードが削除に変更され、レコードは、End Doc でファイルが処理されるまで明細キャッシュに保管されます。

特別に必要なロジックまたは処理

次のような特殊処理が行われます。

- 処理される文書タイプによって、デフォルト値の異なる編集と挿入が行われます。一例としては、仕訳入力 MBF による伝票と請求書の処理があります。税額計算は伝票の場合のみ呼び出されます。
- 必要なイベント処理に応じて、実行する編集機能をプロセス編集フラグが指定します。たとえば、対話型プログラムでは、Grid Record is Fetch イベントが動作したときは Partial Edits を実行して、説明やデフォルト値などを取り出すことができます。Row is Exited and Changed イベントが動作したときは Full Edits を実行し、ユーザーの入力をすべて検証できます。

一般用途とフックアップ

対話型	Grid Record is Fetched Row is Exited and Changed（Asynch）
バッチ	グループ・セクション、カラム・セクション、表セクションの Do Section

共通のパラメータ

名前	エイリアス	I/O	説　明
Job Number	JOBS	I	キーとして使用されるか、またはキャッシュや作業ファイルに固有の名前を作成するために使用されます。Begin Document から取り出されます。
Line Number	LNID	I/O	トランザクション行を識別する固有の番号。明細キャッシュの行番号としても使用できます。
Line Action	ACTN	I	A または 1 = 追加 C または 2 = 変更 D または 3 = 削除
Process Edits（任意指定）	EV01	I	0 = 編集なし 1 = フル編集 2 = 部分編集 **注:** GUI インターフェイスでは通常、部分編集を使用し、バッチ・インターフェイスではフル編集を使用します。このパラメータがブランクの場合、デフォルトではフル編集です。
Error Conditions	ERRC	O	0 = エラーなし 1 = 警告 2 = エラー
Update Or Write to Work File	EV02	I	1 = レコードがワークテーブルに書き出されるか、更新されるか、またはその両方が実行されます。
Record Written to Work File	EV03	I/O	1 = レコードが作業ファイルに書き出されます。これは、作業ファイルへの I/O 呼出しを減らします。 ブランク = レコードはワークテーブルに書き出されません。
Detail Field One	****	I/O	編集されるすべての明細フィールドを受け渡します。一般にこれらはグリッド・レコード・フィールドになります。Edit Line は、検証、データ辞書の編集、UDC の編集を実現し、デフォルトを提供します。
Detail Field Two	****	I/O	
Detail Field XX	****	I/O	

Work Field / Processing Flag One	****	I	プログラムが必要とするワーク・フィールドをすべてリストします。これらのフィールドは、処理や検査日などのフラグでもかまいません。
Work Field / Processing Flag One	****	I	
Work Field / Processing Flag One	****	I	

Edit Document

機　能

Edit Document 要素は次のように機能します。

- 複数行の編集が必要な場合にキャッシュ・レコードを読み取ります。
- ヘッダー情報が必要な場合にヘッダー・キャッシュ・レコードを読み取ります。
- ドキュメントの複数行に関連する相互依存編集を実行します。たとえば Edit Document は、割合の合計が 100%になるようにすべてのレコードを処理し、最後のレコードに情報が存在しないことを検証します。

特別に必要なロジックまたは処理

処理するドキュメントのタイプによって、異なるロジックが実行されます。たとえば、伝票と請求書は仕訳入力編集オブジェクトを通じて処理されますが、残高調整はこれらのドキュメントのタイプによって異なります。

フックアップ・ヒント

- この関数は、Edit Line を呼び出してから End Document を呼び出すまでの間に少なくとも 1 回呼び出します。
- 検証時にエラーが発生した場合はこの関数を再度呼び出し、End Document を呼び出す前にエラーが解消されたことを確認します。
- この関数は、OK button clicked イベントで呼び出してエラーが発生した場合に、ユーザーがアプリケーションを終了する前にエラーが訂正されるようにします。

共通のパラメータ

名前	エイリアス	I/O	説　明
Job Number	JOBS	I	Begin Document から取り出されます。
ErrorConditions	EV01	O	ブランク = エラーなし 1 = 警告 2 = エラー

アプリケーション固有のパラメータ

Begin Document や Edit Line ですべてのレコードが既に追加され、ドキュメント全体を処理するために必要な情報がいずれもキャッシュに保持されているので、この関数では少数のパラメータしか必要になりません。

End Document

関数名

FxxxxxEndDocument

機　能

End Document 要素は次のように機能します。

- 自動採番した番号をドキュメントに割り当てます。伝票の場合、伝票を調整しデータベースにコミットできるようになる前ではなく、仕訳入力編集オブジェクトを呼び出す前に実行してください。このモジュールを追加/削除/更新する前のイベントに配置すると、伝票はこのイベントを実行する前にすべての編集の受け渡しを完了します。
- キャッシュ・レコードを読み取ります。
- ADD の際には、新しいローをテーブルに書き込みます。
- CHG の際には、既存のローを取り出して更新します。
- DEL の際には、テーブルからローを削除します。
- 情報を追加し、関連するテーブルを更新します。たとえば、以下の項目を追加して更新します。
    - 伝票に関連する手作業小切手
    - 〈Address Book（住所録）〉での年累計伝票発行金額カラム
    - 〈Address Book〉用の住所、電話番号、人名録
    - バッチ・ヘッダー
- すべての更新処理が正常終了した後に、その伝票とワーク・フィールドのキャッシュをクリアします。
- 処理オプションで指定されている場合は、データベースに書き出すため、文書を要約します。別の方法でワークテーブルを読み取り、コントロールのブレーク時点でレコードを記述します。
- 通貨換算を実行します。

特別に必要なロジックまたは処理

特別なロジックや処理は不要です。

フックアップ・ヒント

この関数は通常、[OK]ボタンの Post Button Clicked イベントで呼び出され、〈Asynch（非同期）〉でフックアップされます。C 言語コードでは、データベースへの挿入や更新が正常終了した後、Clear Cache を呼び出してキャッシュをクリアします。

共通のパラメータ

名前	エイリアス	I/O	説　明
Job Number	JOBS	I	Begin Document から取り出されます。
Computer ID	CTID	I	アプリケーションで〈GetAuditInfo〉(B9800100)から取り出されます（任意指定）。
Error Conditions	EV01	O	ブランク = エラーなし 1 = 警告 2 = エラー
Program ID	PID	I	通常、ハードコード化します。

アプリケーション固有のパラメータ

- [Time and Date Stamp（日時スタンプ）]フィールドなど、更新や書込みの処理に必要なフィールドをリストします。
- 更新や書込みを実行する際に必要なワーク・フィールドをリストします。
- 更新や書込みの処理に必要なすべての処理オプションをリストします。

Clear Cache

関数名

FxxxxxClearCache

機　能

Clear Cache 要素は、ヘッダーおよび明細キャッシュからレコードを削除します。

特別に必要なロジックまたは処理

固有のキャッシュ名（Dxxxxxxxx[ジョブ番号]）をキャッシュの命名規則として選択した場合は、キャッシュ API の jdeCacheTerminateAll を使用して、キャッシュを削除します。

共通のパラメータ

名前	エイリアス	I/O	説　明
Job Number	JOBS	I	クリアするトランザクションのジョブ番号。このジョブ番号は、BeginDoc から返されます。
Clear Header	EV01	I	ヘッダー・キャッシュをクリアするかどうかを指定します。 1 = キャッシュをクリアする。
Clear Detail	EV02	I	明細キャッシュをクリアするかどうかを指定します。 1 = キャッシュをクリアする。
Line Number From（任意指定）	LNID	I	明細キャッシュのレコードのクリアを開始する位置を指定します。ブランクの場合は、最初のレコードからクリアし始めます。
Line Number Thru（任意指定）	NLIN	I	明細キャッシュのレコードのクリアを停止する位置を指定します。ブランクの場合は、キャッシュの最後まで削除します。

Cancel Document

関数名

FxxxxxxCancelDoc

機　能

任意指定の Cancel Document 要素は、主にファイルのクローズやキャッシュのクリアなどを実行するために[Cancel]ボタンで使用します。Cancel Document は、基本的な関数のクリーンアップを実現するアプリケーション固有の関数です。

特別に必要なロジックまたは処理

これはアプリケーション固有の関数です。

共通のパラメータ

名前	エイリアス	I/O	説　明
Job Number	JOBS	I	クリアするトランザクションのジョブ番号。この番号は、BeginDoc から返されます。

キャッシュ構造体

キャッシュ構造体には、トランザクションのすべての行が保管されます。トランザクション行は、編集が完了した後にキャッシュに書き出されます。その後で EndDoc モジュールがキャッシュを読み込んで、データベースを更新します。キャッシュ構造体のレイアウトは次のとおりです。

ヘッダー

フィールド名	フィールド	キー	タイプ	サイズ
Job Number	JOBS	X	Num	
Document Action	ACTN		Char	1
処理オプション				
Currency Flag	CYCR		Char	1
Business View Fields				
Work Fields				

Job Number BeginDoc モジュールがジョブを開始するときにシステムから割り当てられる固有番号。これにより、キャッシュを使用しているワークステーションのジョブごとにキャッシュのトランザクションが区別されます。ジョブ番号には、自動採番の 00/4 を使用します。固有のキャッシュ名（Dxxxxxxxxx [ジョブ番号]）を使用する場合は、キャッシュごとに 1 つのトランザクションしか処理しないので、必ずしもジョブ番号フィールドをキーとしてキャッシュに保管する必要はありません。そのため、任意のフィールドをキャッシュへのキーとして使用できます。

Document Action 文書に対する操作。有効な値は次のとおりです。

A または 1 = 追加

C または 2 = 変更

D = 削除

処理オプション 処理オプションの値は AllocatePOVersionData によって読み取られ、MBF の他のモジュールで必要になります。

Currency Flag 通貨をオンにするかどうか、および通貨換算のために使用する方法（N、Y または Z）を指定するシステム値

Business View Fields トランザクションを処理し、データベースに書き出すために必要なフィールド。ヘッダー・キャッシュに保存されないレコード形式のフィールドは、いずれも API を使用してレコードをデータベースに追加するときに初期化されます。

Work Fields ビジネス・ビューの一部ではなくても、トランザクションの編集と更新に必要なフィールド。

たとえば、[Last Line Number（最終行番号）]は、明細キャッシュに書き出される最終行番号です。これはヘッダー・レベルで保管され、MBF によって取り出されて増分されます。増分された行番号はヘッダー・キャッシュに渡され、次のトランザクションのために保管されます。

明 細

フィールド名	フィールド	キー	タイプ	サイズ
Job Number	JOBS	X	Char	8
Line Number	（アプリケーション固有）	X	Num	
Line Action	ACTN		Char	1
Business View Fields				
Work Fields				

Job Number BeginDoc モジュールがジョブを開始するときに割り当てられる固有番号。これにより、キャッシュを使用しているクライアントのジョブごとに、キャッシュのトランザクションが区別されます。固有のキャッシュ名（Dxxxxxxxxx[ジョブ番号]）を使用する場合は、キャッシュごとに 1 つのトランザクションしか処理しないので、必ずしもジョブ番号フィールドをキーとしてキャッシュに保管する必要はありません。そのため、キャッシュへのキーとして行番号のみ使用できます。

Line Number 明細キャッシュで行を識別する行番号として使用される固有番号。また、この行番号は、トランザクションをデータベースに書き出すときに、トランザクションに割り当てることもできます。トランザクション行は、エラーが存在しない場合のみ明細キャッシュに書き出されます。

Line Action トランザクション行に対する操作。有効な値は次のとおりです。

A または 1 = 追加

C または 2 = 変更

D = 削除

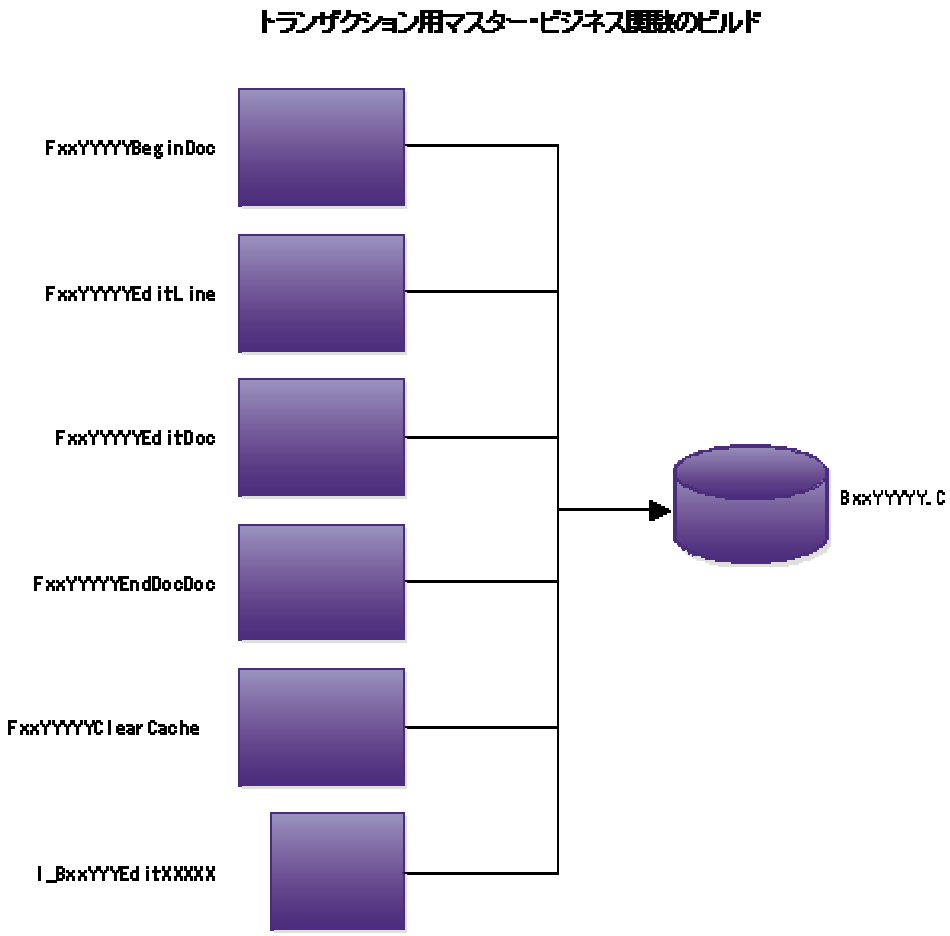
Business View Fields データベースに書き込まれるトランザクションの処理に必要なフィールド。明細キャッシュに保存されないレコード形式のフィールドは、いずれも API を使用してレコードをデータベースに追加するときに初期化されます。

Work Fields ビジネス・ビューの一部ではなくても、トランザクション行の編集と更新に必要なフィールド

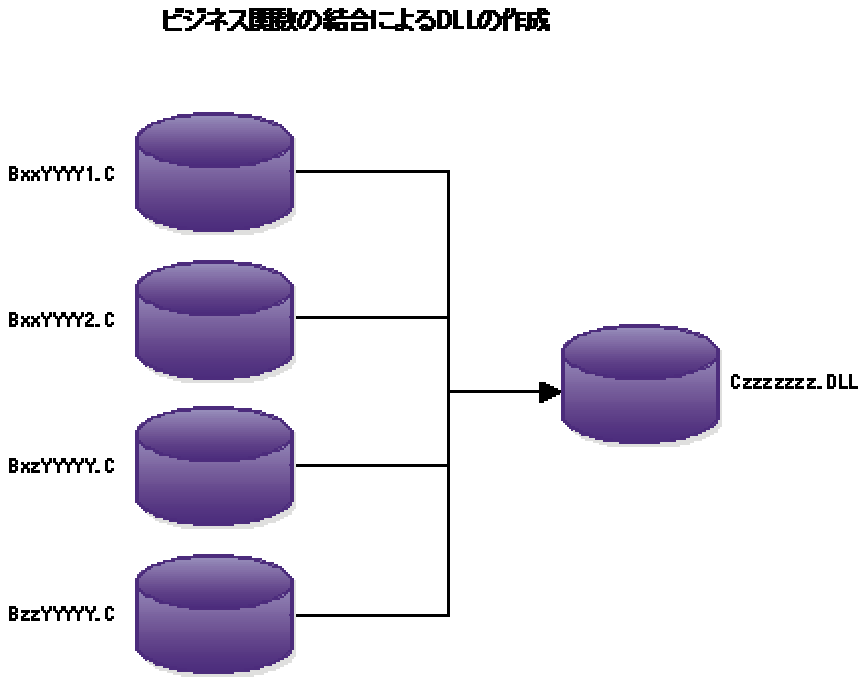
トランザクション用マスター・ビジネス関数の構築

次の図に、トランザクション用マスター・ビジネス関数がどのように構築されるかを示します。

最初に、既に述べた要素を使用して、個別ビジネス関数を作成します。



次に、これらのビジネス関数を組み合わせて DLL を作成します。



トランザクション用マスター・ビジネス関数の実装

トランザクション用マスター・ビジネス関数を実装するには、単一レコード処理か、または文書処理を使用することができます。

単一レコード処理

対話型プログラムのフロー・サンプル

1. Post Dialog is Initialized（任意指定）
   - Begin Document 関数を使用します。
2. Set Focus on Grid
3. Row is Exited and Changed または Row is Exited and Changed ASYNC
   - Edit Line 関数を使用します。
4. Delete Grid Record Verify – After
   - Edit Line 関数を使用して、1 つのレコードを削除します。
   - Edit Document 関数を使用して、1 群のレコードを削除します。
5. OK button pressed イベント
   - Begin Doc を呼び出します。
   - Edit Document 関数を使用します。

6. Post OK Button Pressed
    - End Document 関数を使用します。

マスター・ビジネス関数は通常、指定されたテーブルに対するテーブル入出力をすべて実行します。そのため、ツールでは次の項目を必ず無効にしてください。

- Add Grid Record to DB – Before
    - Suppress Add
- Update Grid Record to DB – Before
    - Suppress Update
- Delete Grid Record to DB – Before
    - Suppress Delete

バッチ・プログラムのフロー・サンプル

1. レポート・ヘッダーのセクションを実行します。
    - Begin Document 関数を使用します。
2. グループ・セクションのセクションを実行します。
    - Edit Line 関数を使用します。
3. 条件セクションのセクションを実行します（任意指定）。
    - Edit Document 関数を使用します。
4. レポート・フッターのセクションを実行します。
    - End Document 関数を使用します。

伝票処理

プログラムのフロー・サンプル

1. Dialog is Initialized
    - Open Batch Edit Object モジュールを使用します。
2. Grid is entered – ヘッダーへの入力を終了
    - Begin Document Edit Object モジュールを使用します。
3. Row is exited
    - Edit Line Edit Object モジュールを使用します。
4. OK button is pressed
    - Edit Document Edit Object モジュールを使用します。
5. Before Add/Delete from Database イベント
    - Suppress Add/Delete
    - End Document Edit Object モジュールを使用します。
6. Cancel button is pressed
    - Close Batch Edit Object モジュールを使用します。

マスター・ファイル用マスター・ビジネス関数

J.D. Edwards では、呼出しプログラムが既定のトランザクションを処理できる MBF（マスター・ビジネス関数）を提供しています。MBF は必要なロジックをカプセル化し、データの整合性を高め、呼出しプログラムをデータベース構造から分離します。

MBF を使用する理由は次のとおりです。

- 再利用可能な、アプリケーション固有のコードを作成する。
- 重複コードを削減する。
- フックアップの一貫性を確保する。
- インターオペラビリティ（相互運用）モデルをサポートする。
- OCM を通じて、処理を分散できるようにする。
- イベント主導アーキテクチャを設計する。

MBF は通常、仕訳や購買オーダーなど複数行のビジネス・トランザクションに使用されます。ただし、複雑度、重要性、または外部取引先からのメンテナンス要件のために、MBF サポートも必要とするマスター・ファイルが存在します。マスター・ファイルのメンテナンスに関する要求事項は、マスター・テーブルと複数業種向けトランザクションに関するものと異なります。

一般にマスター・テーブル MBF は、複数業種向けトランザクション MBF よりもシンプルです。トランザクション MBF はプログラム固有ですが、マスター・テーブル MBF はテーブルに繰り返しアクセスします。

インタオペラビリティを確保するため、テーブル入出力の代わりにマスター・ファイル MBF を使用することができます。これにより、テーブル・イベント・ルールではなくビジネス関数を使用して、関連テーブルの更新を実行できます。複数のレコードが使用される代わりに、すべての編集と処理は 1 回の呼出しで実行されます。

マスター・テーブル MBF は、その基本フォームにおいて次の特徴を備えています。

単一呼出し　一般に MBF を 1 回呼び出すだけで、マスター・ファイル・レコードを編集、追加、更新、削除できます。編集専用オプションも使用できます。

単一データ構造体　要求を作成し、必要なすべての値を提供するために必要なフィールドが 1 つのデータ構造体に収められます。データ・フィールドは、関連マスター・テーブルのカラムに直接対応する必要があります。

キャッシュなし　マスター・ファイル・レコードはそれぞれが独立しているため、キャッシュは不要です。個々の呼出しによって提供される情報とデータベースの現在の条件により、要求された機能を実行する上で MBF が必要なすべての情報が提供されます。

標準エラー処理　他の MBF と同様に、マスター・テーブル MBF は、話型環境とバッチ環境の両方で実行できることが必要です。そのため、呼出しプログラムはエラーの配信方式を判別する必要があります。

照会機能　J.D. Edwards のデータベースから外部システムを完全に独立させられるようにするために、照会オプションが用意されています。これにより外部システムは、マスター・テーブルのキーに関する記述情報にアクセスする際に、その情報を保守管理する際に使用するインターフェイスと同じインターフェイスを使用することができます。

アプリケーションに対する効果　J.D. Edwards アプリケーションに対しては、マスター・テーブル MBF の導入の影響を最小にする必要があります。マスター・テーブル MBF を導入する前に標準をよく検討し、遵守してください。

マスター・テーブルを扱うアプリケーションは、システムを使用して検索/表示フォームでのすべての入出力を処理します。これにより、J.D. Edwards ソフトウェアの全検索機能を使用することができます。

マスター・テーブルを扱うアプリケーションはいずれも、すべての修正/検査フォームが相互に独立したものになるように設計する必要があります。各フォームは、システムを使用してレコードを取り出し、マスター・テーブル MBF を使用して、すべての編集と更新を行います。このような独立設計には、次のような 2 つの大きな利点があります。

- 複数フォームでの従属フィールドに関係する編集を簡素化する様式でアプリケーションを構成します。
- すべてのマスター・テーブルを扱うアプリケーションとそれらの全フォームで、一貫性のある「モードレス処理」を実現できます。

ただし、このようなシンプルなモデルでないケースもあります。次のような場合があります。

- 非常に大型のファイル形式

  マスター・テーブルのカラム数と呼出しのデータ構造体に含まれる必須コントロール・フィールド数の合計がデータ構造体に関する技術的な制約を超える場合は、MBF を分割することができます。そのため、基本データを処理してすべての追加と削除を行う 1 つの MBF と、基本データが設定されている場合に呼出しプログラムで追加データを更新するための 1 つ以上の MBF に分割することをお勧めします。この場合は、一般に技術的な制約とは無関係に MBF を分割するのが妥当です。たとえば、得意先マスター・テーブルがデータ構造体の制限を超えた場合は、次の 2 つの MBF を使用してファイルを処理します。

  - F0301ProcessMasterData
  - F0301ProcessBillingData

  この例では、F0301ProcessMasterData 関数が基本データを処理し、F0301ProcessBillingData がそれ以外のデータを更新するために使われます。

- 従属的な詳細ファイル

  基本マスター・テーブルには、1 対多の関係が可能にする正規化されている追加情報(テーブル)が存在することがあります。仮に J.D. Edwards でマスター・テーブル MBF をデータベースの設計方式に基づいて厳密に設計したとすると、3 つの呼出しになります。詳細な関係の少なくとも 1 つの発生をマスター・ファイル MBF に含めることは有効です。これにより、ユーザーは、シンプルなインターフェイスを使用して適切なマスター・テーブル情報を設定し、平易なニーズに対応することができます。〈Address Book〉の住所と電話番号はこの好例です。基本 ID 情報、住所、および電話番号を使用して簡易住所録 API を呼び出すことで、住所録レコードを作成できることが望まれています。

データ構造体

単一レコード・マスター・ビジネス関数の標準パラメータ

名前	エイリアス	I/O	必須/任意指定	説　明
Action Code	ACTN	I	必須	A = 追加 I = 照会 C = 変更 D = 削除 S = 例外と同じ(レコードは、ユーザーが変更したもの以外、変わりません)
Update Master File	EV01	I	任意指定	0 = 更新なし、編集のみ(デフォルト) 1 = 更新を実行
Process Edits	EV02	I	任意指定	1 = すべて編集(デフォルト) 2 = 部分編集(データ辞書なし)
Suppress Error Messages	SUPPS	I	任意指定	1 = エラー・メッセージを表示しない 0 = 標準エラー処理(デフォルト)
Error Message ID	DTAI	O	任意指定	エラー・コードを返す。
Version	VERS	I	将来使用	XJDE0001 にデフォルト

アプリケーション固有の制御パラメータ(例:住所録)

名前	エイリアス	I/O	必須/任意指定	説　明
Address Book Number	AN8	I/O	任意指定	[Action Code(アクション・コード)]が A(追加)の場合、AN8は任意指定です。それ以外では必須です。
Same as except	AN8	I	任意指定	[Action Code]が S の場合は必須。レコードは、ユーザーが変更したもの以外、変わりません。

アプリケーション・パラメータ(例:住所録)

名前	エイリアス	I/O	必須/任意指定
Alpha Name	ALPH	I/O	必須
Long Address Number	ALKY	I/O	任意指定
Search Type	AT1	I	必須
Mailing Name	MLMN	I	必須
Address Line 1	ADD1	I	任意指定
City	CTY1	I	任意指定
State	ADDS	I	任意指定
Postal Code	ADDZ	I	任意指定

命名規則

- ファンクショナル・サーバー・ソースの名前は、イベント・ルール・ビジネス関数では Nxxxxxxx、C 言語ビジネス関数では Bxxxxxxx とします。これは、他のビジネス関数の命名規則と同じです。このソースには、モジュールごとに外部ビジネス関数があります。各モジュールのデータ構造体の名前は、DxxxxxxA、B、C というようにします（たとえば、N03000052 – D03000052A F0301ProcessMasterData、D03000052B F0301ProcessBillingData）。
- 個別ビジネス関数は、ファイル名 ProcessMasterData という名前にします（たとえば、F0301ProcessMasterData）。データ構造体の制限により、追加のマスター・テーブル関数が必要な場合は、関数名をファイル名 Processxxxxxxxxx とします。x の個所は、処理されるデータ・タイプの記述名にします（たとえば、F0301ProcessBillingData）。
- 〈Object Librarian〉の関数用途コード 192 は、ビジネス関数をファンクショナル・サーバーとして指定するために使用します。

パフォーマンスに対する影響

大型のテーブルでは、その処理方法にかかわらずパフォーマンスの問題が生じることがあります。パフォーマンスを改善するには、次の 2 つのオプションがあります。

- データを論理グループに分けて、データ構造体を小型化して、導入しやすくすることができます。ただしこの場合、ユーザーはレコードを追加したり、テーブルのレコード全体を更新するために複数回呼出しを行わなければなりません。
- 300 フィールドを実現するデータ構造体を使用する。これも導入に手間がかかり、ユーザーはすべてのフィールドを適用しないこともあります。

さまざまなインターフェイスを通じて、後で補足データを追加できます。ほとんどのプロセスでは、データの一部をすぐに追加する必要がありますが、関連データを後で追加できます。たとえば、ユーザーは得意先マスター・レコードを定義しても、得意先請求指示の定義は後日まで待つことがあります。そのため、一方の MBF で基本データを処理し、他方の MBF で追加データを処理するように、MBF を分割する最初のオプションを選択することをお勧めします。

ビジネス関数ドキュメンテーション

ビジネス関数ドキュメンテーションは、個々のビジネス関数の役割と使い方を説明します。ビジネス関数のドキュメンテーションには、次のような情報を記述します。

- 用途
- パラメータ（使用するデータ構造体）
- 個々のパラメータの説明。必要な入出力を示し、戻り値について説明します。
- 関連テーブル（アクセスするテーブル）
- 関連するビジネス関数（その関数から呼び出されるビジネス関数）
- 特殊取扱指示

ビジネス関数のドキュメント化には、ビジネス関数の設計とデータ構造体の設計を使用します。

ビジネス関数ドキュメンテーションの作成

ビジネス関数ドキュメンテーションは、次のレベルで作成することができます。

ビジネス関数の注記 使用する特定のビジネス関数に関するドキュメンテーション

データ構造体の注記 ビジネス関数のデータ構造体に関する注記

パラメータの注記 データ構造体の実際のパラメータに関する注記

▶ ビジネス関数ドキュメンテーションを作成するには

1. 〈Object Management Workbench〉で、ドキュメントを作成するビジネス関数を選択して中央カラムの[Design]ボタンをクリックします。
2. 〈Business Functions Design〉で、[Attachments(添付)]タブをクリックします。
3. [Attachments]領域にドキュメンテーションを入力します。

ビジネス関数ドキュメンテーションのテンプレート

ビジネス関数ドキュメンテーション・テンプレートには、次のセクションがあります。

Purpose	このセクションには、関数の機能概要を入力します。
Setup Notes and Prerequisites	このセクションには、事前に実行すべき関数、初期化する必要のある特別な値、関数を実行するための推奨イベントなど、関数を使用する上での特記事項を入力します。また、関数を使用した後にメモリを個別にクリアする必要の有無も示します。
Special Logic	このセクションには、ビジネス関数ロジックに関する追加事項を入力します。通常は、概要よりも詳しい説明が必要な複雑な関数のみに使用します。
Technical Specification	このセクションには、英語で記述された関数の技術スペックを入力します。これは既存のワープロ文書から直接コピーしてもかまいません。

データ構造体ドキュメンテーションの作成

データ構造体ドキュメンテーションを作成するには、以下の操作を行います。

▶ データ構造体ドキュメンテーションを作成するには

1. 〈Object Management Workbench〉で、ドキュメントを作成するデータ構造体をチェックアウトします。
2. データ構造体がハイライトされていることを確認して、中央カラムの[Design(設計)]ボタンをクリックします。
3. 〈Data Structure Design〉で、[Attachments]タブをクリックします。

4. [Attachments]領域にドキュメンテーションを入力します。

   その他の方法として、フォームの左側のアイコン・バーを右クリックして[Templates]を選択します。テンプレートを選択して必要な情報を入力します。

データ構造体ドキュメンテーションのテンプレート

データ構造体ドキュメンテーションのテンプレートには、次のセクションがあります。

Special Input Expected	適合しない場合は削除します。
Special Output Returned	適合しない場合は削除します。
Significant Data Values	適合しない場合は削除します。

パラメータ・ドキュメンテーションの作成

パラメータ・ドキュメンテーションの作成手順はデータ構造体ドキュメンテーションの場合と同様ですが、1 つ例外があります。注記を入力する構造体でデータ項目を選択した後に、[Data Structure Item Attachments]バインダ・クリップをクリックします。パラメータ・ドキュメンテーションには、特別なテンプレートはありません。

データ構造体項目に関しては、データ項目でやりとりされる、モード・パラメータなどの情報を明確にするための特別な注記を入力できます。注記では、関数を指定した時にその関数が受け入れる有効な値と、その使用方法を指定する必要があります。たとえば有効な値が Add は 1、Delete は 2 の場合があります。その情報をパラメータ・ドキュメンテーションで指定します。

ビジネス関数ドキュメンテーションの生成

ビジネス関数ドキュメンテーションを生成すると、ビジネス関数に関する情報のオンライン・リストが作成されます。このリストは、〈Business Function Documentation Viewer(ビジネス関数ドキュメンテーション・ビューワ)〉プログラム(P98ABSFN)を使用して表示できます。すべてのビジネス関数でビジネス関数ドキュメンテーションを生成するのは時間がかかるので、このタスクは通常、システム管理者が実行します。新規のビジネス関数ドキュメンテーションを作成した場合は、該当するビジネス関数専用のビジネス関数ドキュメンテーションを再生成する必要があります。

▶ ビジネス関数ドキュメンテーションを生成するには

〈Cross Application Development Tools〉メニュー(GH902)から〈Generate BSFN Documentation(ビジネス関数ドキュメンテーションの生成)〉を選択します。

1. 〈Work With Batch Versions(バッチ・バージョンの処理)〉で、バージョン XJDE0001 を選択します。
2. 〈Version Prompting(バージョン・プロンプト)〉で、ビジネス関数ドキュメンテーションを一括生成しない場合は、次のオプションを選択します。
   - Data Selection(データ選択)
3. [Submit(投入)]をクリックします。
4. 〈Data Selection(データ選択)〉で、データ選択基準を作成して[OK]をクリックします。

   ドキュメンテーションを生成する関数だけを選択します。

注:
選択した基準によっては、処理オプションの設定も必要になることがあります。

5. 〈Report Output Destination(レポート出力先)〉で、レポートをローカルに実行する場合は、次のいずれかの出力先を選択します。
   - On Screen(画面)
   - To Printer(プリンタ)

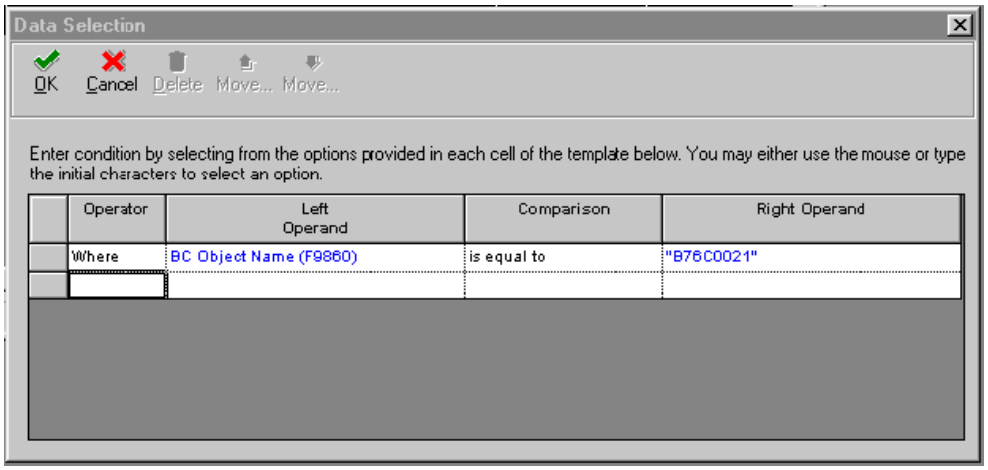
ドキュメンテーションを生成したビジネス関数ごとに HTML(ハイパーテキスト・マークアップ・ランゲージ)リンクが作成されます。また、インデックス HTML ファイルも作成されます。これらの HTML ファイルは、出力待ち行列ディレクトリに配置されます。出力は次のような形式になります。

Function Name
Function Description from
Parent DLL:
Location:
Language:
Purpose
Special Handling
Data Structure
Parameter Name dataitem data type req/ opt

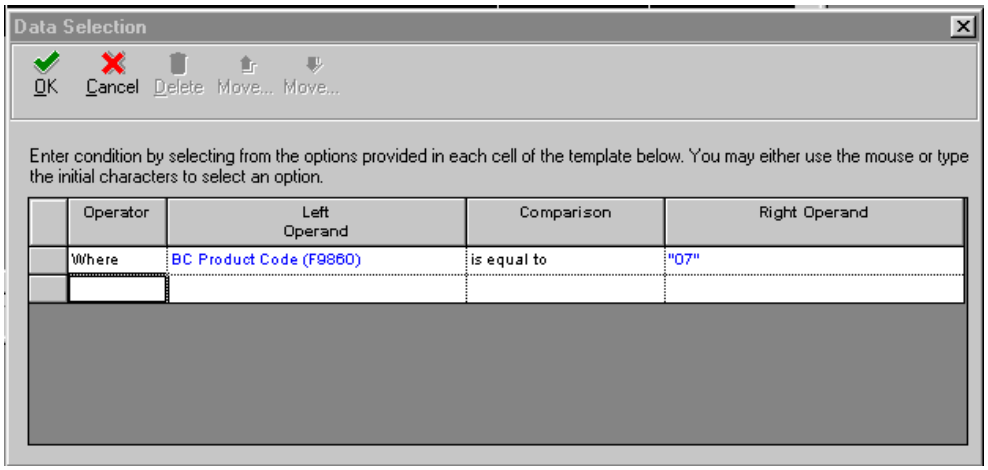
データ選択のヒント

データ選択を使うと、ドキュメンテーションを生成するビジネス関数を選択できます。Generate BSFN Documentation (R98ABSFN)は、データ選択基準に従って、ビジネス関数ドキュメンテーションをフィルタ処理します。すべてのビジネス関数のドキュメンテーションを生成するときは、レポートの実行に時間がかかることがあります。データ選択を使用すると、1 つのビジネス関数、すべてのビジネス関数、またはそれらの任意の組み合わせでドキュメンテーションを生成できます。

たとえば 1 つのビジネス関数のドキュメンテーションを生成する場合は、データ項目の BC Object Name (F9860)を使用できます。



給与計算など特定の製品コードのビジネス関数すべてのドキュメンテーションを生成する場合は、データ項目[BC Product Code(F9860)(BC 製品コード(F9860))]を使います。



〈Data Selection〉フォームの右側のオペランドを使用して値の範囲やリストを選択し、データ選択の精度を向上することもできます。

ビジネス関数に関連する値を使用して、ドキュメンテーションの生成対象となるビジネス関数を絞り込むことが可能です。たとえば J.D. Edwards ソフトウェアの以前のリリースでドキュメンテーションを既に作成していて、そのソフトウェアのアップグレードまたはアップデートの後に新規または修正されたビジネス関数ドキュメンテーションのみが必要な場合は、BC Date - Updated (F9860)を使用することができます。その他に次のような例があります。

- マスター・ビジネス関数ドキュメンテーションを選択するには、BC Function Type(F9860)を使用します。
- クライアント実行ビジネス関数のドキュメンテーションを生成するには、BC Location Business Function (F9860)を使用します。
- NER だけのドキュメンテーションを生成するには、BC Object Type(F9860)を使用します。

〈Business Function Search〉からのドキュメンテーションの表示

イベント・ルールでビジネス関数に接続すると、〈Business Function Search〉フォームが表示されます。その後で、呼び出す関数を選択することができます。[Row]メニューから[Data Structure Notes（データ構造体注記）]または[Attachments]を選択すると、ビジネス関数のドキュメンテーションが表示されます。

〈Business Function – Values to Pass（ビジネス関数 – 送り値）〉からのドキュメンテーションの表示

〈Business Function – Values to Pass〉で、次のいずれかのボタンをクリックすると、1 つのビジネス関数のドキュメンテーションを表示できます。

BSFN Notes ビジネス関数の注記を表示します。

Structure Notes データ構造体全体の注記を表示します。

Parameter Notes 特定のパラメータの注記を表示します。

参照

- 〈Business Function – Values to Pass〉フォームへのアクセスについては、『開発ツール』ガイドの「イベント・ルール・ビジネス関数」

〈Business Function Documentation Viewer〉からのドキュメンテーションの表示

〈Business Function Documentation Viewer〉を使うと、すべてのビジネス関数または選択したビジネス関数のドキュメンテーションを表示できます。レポートの生成後、〈Business Function Documentation Viewer〉プログラム(P98ABSFN)にアクセスし、情報を表示します。ビジネス関数ドキュメンテーションは、この方法で表示することをお勧めします。

〈Business Function Documentation（ビジネス関数ドキュメンテーション）〉フォームには、生成済みの HTML インデックスが表示されます。インデックス全体を表示するか、またはインデックスに含まれる適切な文字をクリックして、特定の関数を選択することができます。ビジネス関数をダブルクリックすると、その関数に固有のドキュメンテーションが表示されます。

メディア・オブジェクトは、メディア・オブジェクト待ち行列に基づいて、ビジネス関数の HTML インデックスをロードします。メディア・オブジェクトの待ち行列テーブルで、Business Function Doc という待ち行列が定義されています。

この待ち行列は、ビジネス関数 HTML ファイルが配置されているディレクトリをポイントする必要があります。通常、システム管理者は全ビジネス関数のドキュメンテーションを生成します。生成プロセスではドキュメンテーション・ファイルがローカル・ディレクトリに置かれるので、管理者はそのファイルをデプロイメント・サーバーの中央ディレクトリにコピーする必要があります。ファイルは、メディア・オブジェクト・ビジネス関数注記のメディア・オブジェクト待ち行列にコピーしてください。ソフトウェアのスタンドアロン・バージョンを使用している場合、このパスは通常は jde.ini ファイルの Network Queue Settings セクションからの出力ディレクトリです。このエントリが jde.ini ファイルにない場合は、J.D. Edwards ソフトウェア・ディレクトリの print queue ディレクトリにあります。

注:

メディア・オブジェクト待ち行列の定義については、『システム・アドミニストレーション』ガイドの「ERP9.0 のテキスト項目」を参照してください。

▶ 〈Business Function Documentation Viewer〉からドキュメンテーションを表示するには

〈Cross Application Development Tools〉メニュー(GH902)から〈Business Function Documentation Viewer〉レポート(R9000C)を選択します。

生成したビジネス関数ドキュメンテーションの HTML インデックスを表示します。

ビジネス関数処理フェイルオーバーについて

ビジネス関数が正しく処理できなかった場合、J.D. Edwards ソフトウェアは、トランザクションの回復と再処理を試みることがあります。システムでは、プロセス障害とシステム障害という 2 つの重要な障害状況が認識されます。

プロセス障害は、jdenet_k プロセスが異常終了したときに発生します。プロセス障害では、J.D. Edwards サーバー処理が新しい jdenet_k プロセスを起動して処理を継続します。

システム障害は、J.D. Edwards サーバー処理がすべて失敗したり、マシン自体がダウンしたり、クライアントがネットワーク障害によってサーバーにアクセスできないときに発生します。システム障害では、ビジネス関数処理が補助サーバーとローカル・クライアントのいずれかに再ルーティングしなければなりません。この障害からの回復は、次のように試みられます。

サーバーの呼出しに失敗すると、システムはサーバーへの再接続を試みます。

- 再接続に成功してキャッシュが存在しない場合、システムはサーバーでビジネス関数を再実行します。キャッシュが存在する場合は、ユーザーをアプリケーションから強制退去させます。
- 再接続に失敗してキャッシュが存在しない場合、システムは補助サーバーか、ローカル・クライアントに切り替えます。キャッシュが存在する場合は、ユーザーをアプリケーションから強制退去させます。

1 つのモジュールが切り替わると、後続のモジュールがすべて新しい位置に切り替わります。

キャッシュ

キャッシュは、頻繁にアクセスされるリモート・オブジェクトの内容のローカル・コピーを保管する処理です。キャッシュを使用してパフォーマンスを改善することができます。J.D. Edwards ソフトウェアは、情報のキャッシュ処理に次の 2 つのプロセスを使用します。

- システムは、固定情報に関連付けられているテーブルなど、一定のテーブルを起動時にデータベースから読み込んだときに自動的にキャッシュします。これらのテーブルをユーザーのワークステーションやサーバーにキャッシュして、データへのアクセスと取出しを高速化します。
- 個々のアプリケーションがキャッシュを使用できるようにします。JDECACHE API は、サーバーやワークステーションのメモリを一時的なストレージとして使用できるようにしています。

JDECACHE について

JDECACHE は、アプリケーションがメモリに保管する必要のある、あらゆるタイプのインデックス付きデータを、アプリケーションが動作しているプラットフォームとは無関係に保持できる JDEKRNL のコンポーネントです。これは、テーブル全体をデータベースから読み取り、メモリに保管できることを意味します。アプリケーションが動作しているコンピュータの制限を除き、アプリケーションが使用できるデータのタイプ、データのサイズ、およびデータ・キャッシュの数には制限がありません。固定長のレコードも可変長のレコードもいずれもサポートされます。サポートされているプラットフォームで JDECACHE を使用するには、API 呼び出しの簡単なセットに関する知識があるだけで十分です。

JDECACHE によって取り扱われるデータは RAM に存在します。そのため、JDECACHE を実際に使用する必要があるかどうかを確認してください。JDECACHE を使用する場合は、レコードとインデックスを念入りに設計にします。 J.D. Edwards ソフトウェアとその他のアプリケーションもこのメモリを必要とするので、JDECACHE に保管するレコード数を最小限にします。

JDECACHE は、複数カーソル、複数インデックス、パーシャル・キーのそれぞれの処理をサポートします。JDECACHE は、データ処理のためのキャッシュでの配置が自由に行えます。これにより、キャッシュでの検索が減少し、パフォーマンスが向上します。

JDECACHE 使用の機会

アプリケーションでの JDECACHE API の使用例を紹介します。

アプリケーションで、ユーザーがグリッドに入力したレコードを Button Clicked イベントでの[OK]処理がアクティブ化されるまで保管する必要があるときは、ワークテーブルを使用します。[OK]処理では、すべてのレコードをデータベースに対して同時に更新する必要があります。これはトランザクション処理とよく似ています。たとえば、購買オーダー明細入力グリッドで 30 行の情報を入力してからトランザクションを取り消した場合、ワークテーブルのレコードはすべて削除され、データベースには書き込まれません。トランザクション処理では、ユーザーが明細ローを終了するたびに各フィールドが編集され、そのレコードがワークテーブルに書き出されています。

ワークテーブルを使用せずにこのような状況を実施した場合は、ユーザーがローを終了するたびに、データベース・テーブルに対する取消不可能な更新が行われます。ワークテーブルを使用すると、テーブルの更新を制限し、テーブルの更新が[OK]ボタンの処理でのみ発生し、トランザクション境界に含まれるようにすることができます。ワークテーブルは、種々の処理のためにグリッドに対するデータ境界を定義します。これは、複数のアプリケーションやプロセス(ビジネス関数など)がワークテーブルのデータにアクセスして更新や計算を実行しなければならないときに有効です。

この例では、ワークテーブルは適切に機能しますが、キャッシュを使用することで、よりパフォーマンスが向上すると思われます。JDECACHE を使用すると、ユーザーが 1 つの購買オーダーに入力したレコードをメモリに保管できます。保管するレコード数は、各レコードのキャッシュ・バッファのサイズ、ローカル・メモリのサイズ、使用するビジネス関数の実行場所(たとえばサーバーまたはワークステーション)などによって異なります。通常は、1000 以上のレコードを保管する必要はありません。たとえば、住所録テーブル全体をメモリにキャッシュしないでください。

パフォーマンスに関する考慮事項

JDECACHE のパフォーマンスを改善するには、次のガイドラインに従ってください。

- キャッシュするレコード数を最小限に抑えます。
- 使用するカラム(セグメント)を少なくすると、検索、挿入、削除処理の速度が上がります。場合によっては、キャッシュにさらに追加するかどうかを判断する前に、各カラムを比較しなければならないことがあります。
- キャッシュに保管するレコード数を少なくすると、すべての操作がスピードアップします。

JDECACHE API セット

JDECACHE を操作するには、公開の API を使用します。JDECACHE API をどのように編成すれば、それらを有効に導入できるかを理解する必要があります。

JDECACHE 管理 API

JDECACHE 管理 API を使用すると、次のようなキャッシュ管理ができます。

- キャッシュの設定
- キャッシュのクリア
- キャッシュの終了

キャッシュの統計情報を取り出すには、jdeCacheGetNumRecords と jdeCacheGetNumCursors の 2 つの API を使用します。これらには、HCACHE ハンドルのみが渡されます。これら以外のすべての JDECACHE 管理 API には、常に次のハンドルを受け渡してください。

- HUSER
- HCACHE

これら 2 つは、キャッシュの識別と管理に不可欠なハンドルです。

JDECACHE 管理 API のセットは、次の API から構成されています。

- jdeCacheInit
- jdeCacheInitMultipleIndex
- jdeCacheInitUser
- jdeCacheInitMultipleIndexUser
- jdeCacheGetNumRecords
- jdeCacheGetNumCursors
- jdeCacheClear
- jdeCacheTerminate
- jdeCacheTerminateAll

jdeCacheInit/jdeCacheInitMultipleIndex API は、キャッシュをユーザー別に初期化します。そのため、ある 1 人のユーザーが J.D. Edwards ソフトウェアにログインして、同じアプリケーションの 2 つのセッションを同時に実行した場合、これら 2 つのアプリケーション・セッションは同じキャッシュを共有します。その結果、1 番目のアプリケーションがキャッシュからレコードを削除すると、2 番目のアプリケーションは、そのレコードにアクセスできなくなります。逆に、2 人のユーザーが J.D. Edwards ソフトウェアにログインして、同じアプリケーションを同時に実行した場合、これら 2 つのアプリケーション・セッションは異なるキャッシュを使用します。そのため、1 番目のアプリケーションがそのキャッシュからレコードを削除しても、2 番目のアプリケーションは、それに固有のキャッシュに格納されているレコードにアクセスできます。

jdeCacheInitUser/jdeCacheInitMultipleIndexUser API は、キャッシュをアプリケーション別に初期化します。そのため、ユーザーが J.D. Edwards ソフトウェアにログインして、同じアプリケーションの 2 つのセッションを同時に実行した場合、これら 2 つのアプリケーション・セッションは別々のキャッシュを使用します。その結果、1 番目のアプリケーションがそのキャッシュからレコードを削除しても、2 番目のアプリケーションは、それに固有のキャッシュに格納されているレコードにアクセスできます。

JDECACHE 操作 API

JDECACHE 操作 API は、キャッシュのデータの取出しと操作のために使用することができます。各 API は、現在操作されているレコードへのポインタとして機能するカーソルを実現します。このカーソルは、キャッシュのナビゲーションに不可欠です。JDECACHE 操作 API には、次のタイプのハンドルを受け渡してください。

HCACHE	処理されるキャッシュを特定します。
HJDECURSOR	処理されるキャッシュの位置を特定します。

JDECACHE 操作 API のセットは、次の API から構成されています。

- jdeCacheOpenCursor
- jdeCacheResetCursor
- jdeCacheAdd
- jdeCacheFetch
- jdeCacheFetchPosition

- jdeCacheUpdate
- jdeCacheDelete
- jdeCacheDeleteAll
- jdeCacheCloseCursor
- jdeCacheFetchPositionByRef
- jdeCacheSetIndex
- jdeCacheGetIndex

JDECACHE の処理

JDB 環境は、JDECACHE 環境を作成、管理、解消します。JDECACHE 環境で使用するキャッシュは、それぞれが JDB ユーザーに関連付けられます。そのため、JDECACHE API を呼び出す前に、JDB_InitBhvr API を呼び出す必要があります。

JDECACHE は、キャッシュを初期化してから使用できます。キャッシュを初期化する前にインデックスを定義する必要があります。このインデックスは、レコードのどのフィールドを使用してキャッシュ・レコードが固有に識別されるかをキャッシュに対して指定するものです。インデックスが参照するデータのグループごとに別個のキャッシュを作成します。

JDECACHE API の呼び出し

JDECHACHE API は、一定の順序で呼び出す必要があります。JDECACHE 関連 API は、次の順序に従って呼び出す必要があります。ただし 6 番目は、実際の JDECACHE API を任意の順序で呼び出すことができます。

1. JDB_InitBhvr
2. インデックスを作成します。
3. jdeCacheInit または jdeCacheInitMultipleIndex
4. jdeCacheAdd
5. jdeCacheOpenCursor
6. JDECACHE オペレーション

   以下の 8 つの JDECACHE オペレーションは任意の順序で実行できます。

   - jdeCacheFetch
   - jdeCacheOpenCursor（第 2 カーソル）
   - jdeCacheFetchPosition
   - jdeCacheUpdate
   - jdeCacheDelete
   - jdeCacheDeleteAll
   - jdeCacheResetCursor
   - jdeCacheCloseCursor（第 2 カーソルを開いた場合）

7. jdeCacheCloseCursor
8. jdeCacheTerminate
9. JDB_FreeBhvr

インデックスの設定

JDECACHE にデータを保管したり取り出したりするには、少なくとも 1 つのカラムから構成されるインデックスを 1 つだけ設定する必要があります。インデックスは、インデックス構造体で最大 25 カラム（セグメントと呼ばれます）に制限されます。提供されるデータ・タイプを使用して、インデックスの様相をキャッシュ・マネージャに伝えます。インデックスのカラム（セグメント）数と各カラムのオフセットとサイズをデータ構造体で指定します。パフォーマンスを最大にするには、セグメント数を最小限にします。

インデックス情報をもたせるデータ構造体は、次のように定義します。

```
#define JDECM_MAX_UM_SEGMENTS 25

struct _JDECMKeySegment

{

    short int nOffset;          /* Offset from beginning of structure in bytes */

    short int nSize;            /* Size of data item in bytes */

    int idDataType;             /* EVDT_MATH_NUMERIC or EVDT_STRING*/

}   JDECMKEYSEGMENT;

struct _JDECMKeyStruct

{

    short int nNumSegments;

    JDECMKEYSEGMENT CacheKey[JDECM_MAX_NUM_SEGMENTS];

}   JDECMINDEXSTRUCT;
```

JDECACHE でインデックスを作成するときは、次の規則に従います。

- 1 つのインデックスを設定する場合は、インデックス構造体を 1 つの要素を保持する配列として宣言します。複数のインデックスを設定する場合は、インデックス構造体を複数の要素を保持する配列として宣言します。無数のインデックスを作成できます。
- インデックス構造体には memset()を使用します。複数のインデックスで memset()を使用するときには、インデックス構造体のサイズとインデックスの総数を掛け合わせます。
- CacheKey 配列のカラム数に一致するセグメント数を要素として割り当てます。
- offsetof()を使用して、カラムを格納する構造体でのカラムのオフセットを指定します。

次に、住所録テーブル(F0101)を保持するキャッシュの一例を示します。このキャッシュのデータは、ABAT1、ABAC01、ABAC02、ABDC の各カラムから構成されるインデックスによって参照されます。

```
/*Declare the index array of one element*/
JDECMINDEXSTRUCT              Index[1];
/*Initialize the structure*/
memset(&Index,0x00,sizeof(JDECMINDEXSTRUCT));
Index.nNumSegments = 4;
Index.CacheKey[0].nOffset = offsetof(F0101, abat1);
Index.CacheKey[0].nSize = sizeof(f0101.abat1);
Index.CacheKey[0].idDataType         =EVDT_STRING;
Index.CacheKey[1].nOffset = offsetof(F0101, abac01);
Index.CacheKey[1].nSize = sizeof(f0101.abac01);
Index.CacheKey[1]. idDataType        =EVDT_STRING;
Index.CacheKey[2].nOffset = offsetof(F0101, abac02);
Index.CacheKey[2].nSize = sizeof(f0101.abac02);
Index.CacheKey[2]. idDataType        =EVDT_STRING;
Index.CacheKey[3].nOffset = offsetof(F0101, abdc);
Index.CacheKey[3].nSize = sizeof(f0101.abdc);
Index.CacheKey[3]. idDataType        =EVDT_STRING;
```

フラグの idDataType は、特殊キーのデータ・タイプを示します。次の例では、データ・タイプが MATH_NUMERIC の ABAN8 を使用しています。

```
Index.nNumSegments = 1;
Index.CacheKey[0].nOffset = offsetof(F0101, aban8);
Index.CacheKey[0].nSize = sizeof(f0101. aban8);
Index.CacheKey[0]. idDataType =EVDT_MATH_NUMERIC;
```

複数のインデックスが設定されたキャッシュの例を以下に示します。

```
/*Declare the index array of 2 elements*/
JDECMSTRUCT  Index[2];
int  numIndexes=2;
/*Initialize the structure*/
memset(&Index,0x00,sizeof(JDECMSTRUCT)* numIndexes);
Index[0].nkeyID = 1;
Index[0].nNumSegments = 1;
Index[0].CacheKey[0].nOffset=offsetof(F0101,abat1);
Index[0].CacheKey[0].nSize=sizeof(f0110.abat1);
```

```
Index[0].CacheKey[0].idDataType=EVDT_STRING;

Index[1].nkeyID = 2;

Index[1].nNumSegments = 2;

Index[1].CacheKey[0].nOffset=offsetof(F0101,abac02);

Index[1].CacheKey[0].nSize=sizeof(f0110. abac02);

Index[1].CacheKey[0].idDataType=EVDT_STRING;

Index[1].nNumSegments = 1;

Index[1].CacheKey[1].nOffset=offsetof(F0101,abat1);

Index[1].CacheKey[1].nSize=sizeof(f0110.abat1);

Index[1].CacheKey[1].idDataType=EVDT_STRING;
```

キャッシュの初期化

インデックスを設定したら、jdeCacheInit または jdeCacheInitMultipleIndex を呼び出して、キャッシュを初期化（作成）します。

固有のキャッシュ名を受け渡し、JDECACHE がそのキャッシュを識別できるようにします。この API にインデックスを受け渡し、キャッシュに保持させるデータの参照方法を JDECACHE に知らせます。各キャッシュはユーザーに関連付けられなければならないので、JDB_InitUser の呼び出しから得られたユーザー・ハンドルも受け渡す必要があります。この API は、JDECACHE が作成したキャッシュに HCACHE ハンドルを返します。このハンドルは、キャッシュを識別するためにその後のすべての JDECACHE に表示されます。

インデックスのキーは、そのキャッシュのすべての jdeCacheInit と jdeCacheInitMultipleIndex 呼び出しで、キャッシュが終了するまで常に同じでなければなりません。インデックスのキーは、そのインデックスを使用するたびにインデックスとしての数、順序、タイプが一致する必要があります。

キャッシュの初期化が無事終了すると、JDECACHE API を使用して JDECACHE 操作を実行することができます。jdeCacheInit から取得されたキャッシュ・ハンドルは、すべての JDECACHE 操作で受け渡さなければなりません。

JDECACHE は、データの格納時にキャッシュにアクセスする内部インデックス定義構造体を作成します。インデックスが定義され、それが jdeCacheInit に受け渡されるときの様子を次の例で示します。

例：インデックス定義構造体

このシナリオでは、キャッシュが保持する各レコードが次の構造体から構成されるようにします。

```
int nInt1

char cLetter1

char cLetter2

char cLetter3

char szArray(5)
```

キャッシュの各レコードが、以下に格納される値によって固有のインデックスが設定されるようにします。

- nInt1
- cLetter1
- cLetter3

この情報を jdeCacheInit に受け渡すと、JDECACHE は、内部使用のために次のインデックス定義構造体を作成します。インデックス定義構造体は STRUCT 文字用です。

例：インデックス定義構造体

インデックス・キー番号 インデックス・キー #1	インデックス・キー・オフセット 0	インデックス・キー・オフセット 整数
インデックス・キー#2	4	文字
インデックス・キー#3	6	文字

キャッシュにアクセスするためのインデックスの使用

インデックスを使用してキャッシュにアクセスするときは、API に送られるインデックスのキーと、そのキャッシュに対する jdeCacheInit 呼び出しで使用されるインデックスのキーが、数、順序、オフセット位置およびタイプにおいて一致しなければなりません。そのため、jdeCacheInit に受け渡されたインデックスで使用されていたフィールドの位置がオフセット 99 である場合、このフィールドは、JDECACHE アクセス API に受け渡されたインデックス構造体においてもオフセット 99 である必要があります。

インデックス構造体を必要とする API を呼び出すときは、必ず jdeCacheInit の呼び出しで使用した場合と同じインデックス構造体を使用してください。

インデックス・オフセットを jdeCacheInit で指定しなければならない理由と、キャッシュからレコードを呼び出すときにそれらを使用する方法を、次の例で紹介します。この例では、受け渡されたキーと JDECACHE 内部インデックス定義構造体を併用して、キャッシュ・レコードにアクセスする方法を示します。

例：JDECACHE 内部インデックス定義構造体

この例では、次のインデックス・キーの値に一致するレコードを検索するとします。

- 1
- c
- i

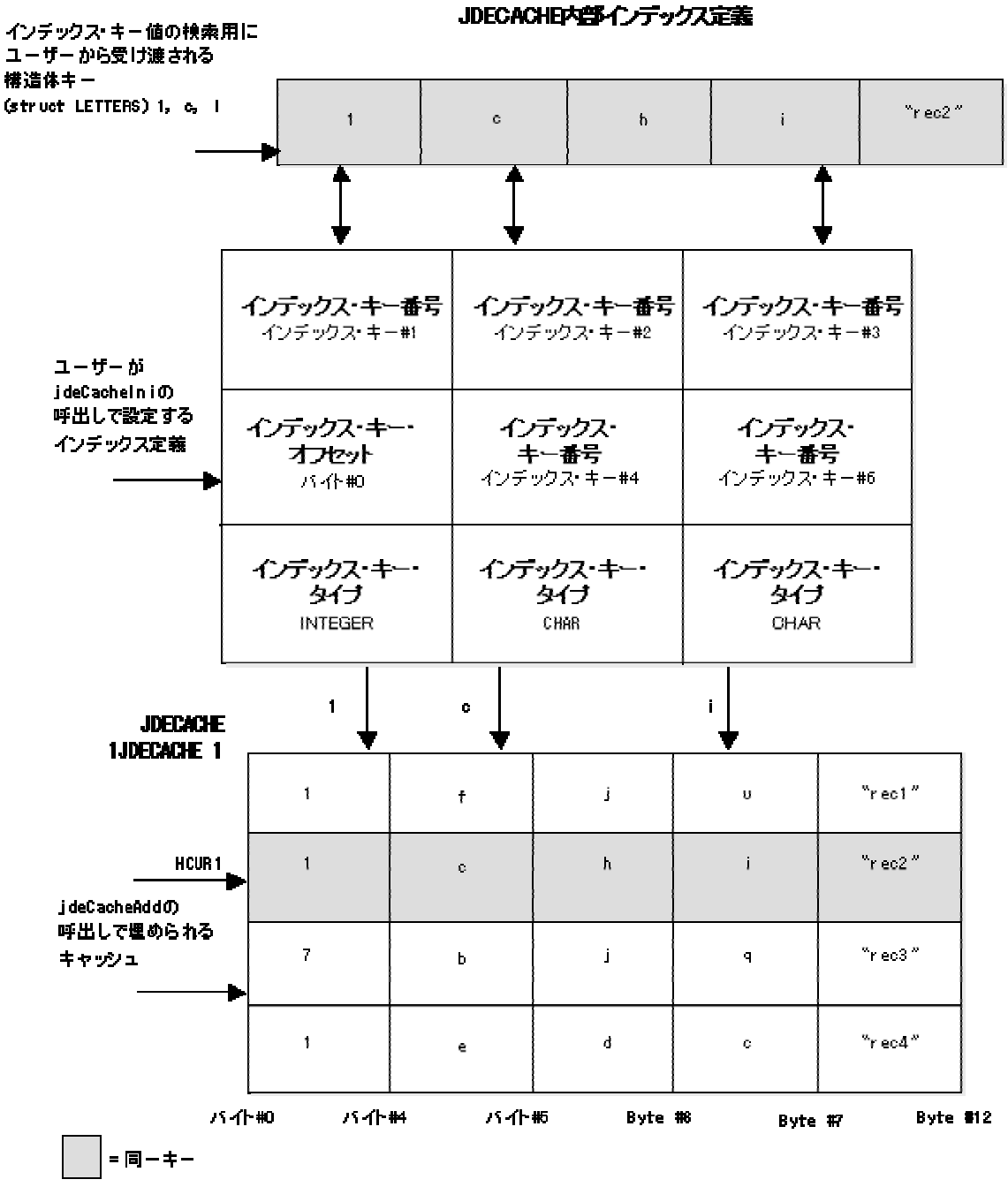
JDECACHE は、ユーザーによって受け渡された構造体の、jdeCacheInit の呼び出しで定義されているバイト・オフセット位置にある値にアクセスします。

JDECACHE は、受け渡された構造体から取り出した値の 1、c、i と、個々のキャッシュ・レコードの対応するバイト・オフセット位置にある対応値を比較します。これらのキャッシュレ・コードは、jdeCacheAdd によってキャッシュに挿入された構造体として保持されており、最初に受け渡された構造体と同じです。受け渡されたキーに一致する構造体は、HCUR1 がポイントする 2 番目の構造体です。

キャッシュにアクセスするには、これよりも小さいキーだけを保管する構造体を作成しないでください。JDECACHE は、一般的なインデクシング・システムとは違って、キャッシュ・レコードのインデックスを実際のキャッシュ・レコードと別個に保管しません。これは、JDECACHE がメモリ常駐データを処理し、メモリをできるだけ保護するように設計されているためです。そのため JDECACHE は、インデクシングだけのために余分な構造体を保持してメモリを浪費することはしません。代わりに JDECACHE レコードには、インデックスの保持とデータの保持という二重の目的があります。このことは、キーを使用して JDECACHE からレコードを取り出すときは、レコードをキャッシュに保持するために使用される構造体と同じタイプの構造体にキーを保管する必要があることを意味します。

インデックスで定義されていたオフセットが jdeCacheInit に受け渡されたキャッシュ以外のキャッシュにアクセスするには、キー構造体を使用しないでください。キャッシュにアクセスするときにキーを格納する構造体と、キャッシュ・レコードを保管するために使用する構造体は、同じにする必要があります。

次の図に、インデックス定義で使用されていたものとは別の構造体を使用してキャッシュに挿入したときの状況を示します。



jdeCacheTerminate の呼び出しを挟まずに同じキャッシュ名と同じユーザー・ハンドルを使って jdeCacheInit を再度呼び出すと、最初の jdeCacheInit によって初期化されたキャッシュはそのまま維持されます。jdeCacheInit を同じキャッシュ名を使って呼び出すときには、毎回同じインデックスを使って呼び出してください。同じキャッシュで別のインデックスを使って jdeCacheInit を呼び出した場合、JDECACHE API は機能しません。

検索用のキーは、常にキャッシュ・レコードを保管する場合と同じ構造体タイプを使用する必要があります。

jdeCacheInit/jdeCacheTerminate 規則の適用

同じキャッシュをビジネス関数やフォーム全体で使用する場合を除き、jdeCacheInit または jdeCacheInitMultipleIndex では、対応する jdeCacheTerminate が必要です。この場合、終了していない jdeCacheInit や jdeCacheInitMultipleIndex 呼出しは、jdeCacheTerminateAll によって終了する必要があります。

jdeCacheTerminate 呼出しは、対応する直前の jdeCacheInit を終了します。これはネストされているビジネス関数で同じキャッシュを使用できることを意味します。それぞれの関数で、キャッシュ名を受け渡す jdeCacheInit を実行します。その関数を終了する前に jdeCacheTerminate を呼び出します。これは、キャッシュを削除しません。代わりに、キャッシュと受け渡された HCACHE ハンドルとの関連付けを解消します。キャッシュは、jdeCacheTerminate 呼出しと jdeCacheInit 呼出しの数が一致したときにのみ、メモリから完全に解消されます。これと対照的に、jdeCacheTerminateAll を 1 回呼び出すと、jdeCacheInit や jdeCacheInitMultipleIndex 呼出し、または jdeCacheTerminate 呼出し数とは無関係にメモリからキャッシュが解消されます。

複数のビジネス関数/フォームでの同じキャッシュの使用

2 つの以上のビジネス関数やフォームで同じキャッシュを使用しなければならない場合は、最初のビジネス関数/フォームで jdeCacheInit を呼び出し、それにデータを付加します。このビジネス関数/フォームを終了した後に jdeCacheTerminate を呼び出さないでください。JdeCacheTerminate は、メモリからキャッシュを削除するからです。代わりに、後続のビジネス関数/フォームで、初回の jdeCacheInit 呼出しで使用したものと同じインデックスとキャッシュ名を使って jdeCacheInit をもう一度呼び出します。最初のキャッシュは終了していないので、JDECACHE は名前が一致するキャッシュを検索し、それをユーザーに割り当てます。キャッシュにはすでにレコードが保管されているので、キャッシュをリフレッシュする必要はありません。このキャッシュでそのまま通常のキャッシュ操作を継続できます。

複数のビジネス関数/フォームでキャッシュを繰り返し初期化する場合は、jdeCacheTerminateAll を使用して、初期化されたキャッシュのすべてのインスタンスを終了します。解消するキャッシュは、この API に受け渡す HCACHE に対応するキャッシュ名を使って指定します。この API は、jdeCacheInit が呼び出された回数に応じて jdeCacheTerminate を呼び出す必要がないときに使用します。複数のビジネス関数/フォームで同じキャッシュを初期化したときに別のフォームやビジネス関数に移動すると、ローカル変数の HCACHE は失われます。同じキャッシュを複数のビジネス関数/フォームで共有するために、別のフォームやビジネス関数で同じキャッシュを使用する場合は、フォームやビジネス関数を終了するときに jdeCacheTerminate を呼び出さないでください。

JDECACHE カーソル

JDECACHE カーソル（JDECACHE カーソル・マネージャ）は、JDECACHE のコンポーネントであり、レコードの取出しと更新に使用する JDECACHE カーソルを実装します。JDECACHE カーソルはユーザーのキャッシュのレコードへのポインタです。キャッシュ・フェッチ API の呼出し時には、カーソルが指しているレコードに続くレコードがキャッシュから取り出されます。

JDECACHE カーソルのオープン

JDECACHE データの操作はカーソルに依存します。JDECACHE データ処理 API を使用する場合は、カーソルを開いておく必要があります。HJDECURSOR タイプのカーソル・ハンドルを取得するには、カーソルを開く必要があります。このハンドルは、次に（jdeCacheAdd API 以外の）すべての JDECACHE データ処理 API に受け渡されます。カーソルを開くには jdeCacheOpenCursor API を呼び出します。この API の呼出しによって、（jdeCacheAdd 以外の）すべてのデータ処理 API の呼出しも可能になります。カーソルを開かないと、これらの API は機能しません。この API を呼び出すと、カーソルが JDECACHE データセットを開き、その中で動作します。この API はデータセットを開きますが、データを取り出せません。このことは、カーソルを開く前に jdeCacheInit を呼び出してキャッシュを初期化し、jdeCacheAdd を呼び出してデータを配置しなければならないことを意味します。

複数のカーソルをキャッシュに取得するには、jdeCacheOpenCursor を呼び出し、異なる HJDECURSOR ハンドルを受け渡します。複数カーソル環境では、カーソルはいずれも相互に独立したものになります。

カーソルの処理を終了したら、カーソルを停止して閉じる必要があります。この操作を行うには、jdeCacheCloseCursor API を呼び出し、jdeCacheOpenCursor に受け渡した HJDECURSOR ハンドルに一致する HJDECURSOR を受け渡します。カーソルが閉じたら、jdeCacheOpenCursor を呼び出して開かない限り、再び使用することはできません。

参照

- 複数のカーソルについては、『開発ツール』ガイドの「JDECACHE の複数カーソル・サポート」

HJDECURSOR

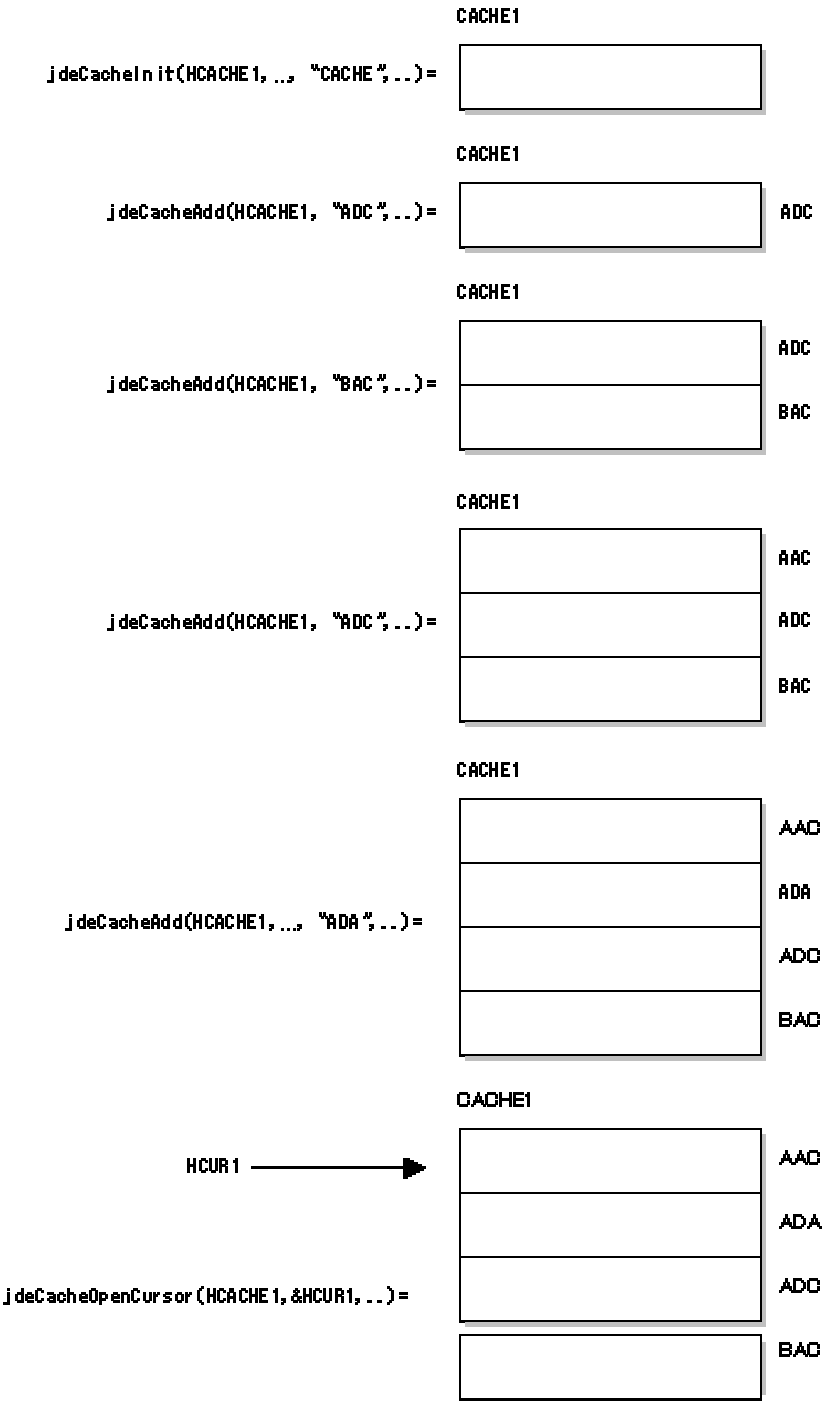
HJDECURSOR は、カーソル・ハンドルのデータ・タイプです。これは、jdeCacheAdd 以外のすべての JDECACHE データ処理 API に受け渡す必要があります。

JDECACHE データセット

JDECACHE データセットは、カーソルの現在位置にあるレコードから、それ以降の最後のレコードまでをすべて含みます。そのため、カーソルがデータセットの中央にある場合、カーソルの現在位置よりも前にあるキャッシュのレコードはいずれもデータセットの一部と見なされません。JDECACHE データセットは、所定のインデックス・キーの昇順に並べられているキャッシュ・レコードから構成されます。このことは、JDEACACHE でのレコードの配置順序がそのまま JDECACHE カーソルがレコードを取り出す順序になるとは限らないことを意味します。JDEACACHE カーソルは、インデックス・キーの昇順でレコードを順次取り出します。レコードを順に取り出すときに、カーソルは前進して、データセットのサイズを縮小します。カーソルがデータセットの最後レコードを越えて進むと、エラーが返されます。

次に、JDECACHE キャッシュと JDECACHE データセットの作成例を示します。

例: JDECACHEのキャッシュとデータベースの作成



カーソル・アドバンシング API

カーソル・アドバンシング JDECACHE フェッチ API は、カーソルの基本概念を実現します。カーソル・アドバンシング API セットは、JDECACHE からレコードを取り出す前に JDECACHE データセットでカーソルを次のレコードに進める API から構成されています。カーソル・アドバンシング・フェッチ API の例を次に示します。

- jdeCacheFetch
- jdeCacheFetchPosition

jdeCacheFetch を呼び出すと、最初に JDECACHE データセットの次のレコードにカーソルを合わせてから、そのレコードを取り出します。JDECACHE カーソルでは、呼出しによってデータセットの特定のレコードにカーソルを合わせることもできます。そのためには、所定のキーに一致するレコードにカーソルを進めて、それを取り出す jdeCacheFetchPosition API を呼び出します。

一定の位置から始まるレコードを順に取り出す必要がある場合は、カーソル・アドバンシング・フェッチ API の組み合わせを使用することができます。jdeCacheFetchPosition を呼び出し、取り出しを開始するレコードのキーを渡します。これにより、データセットの目的の位置までカーソルが進み、そのレコードを取り出すことができます。その後 jdeCacheFetch を呼び出すと、データセットの最後に到達するか、またはプログラムが何らかの理由で停止するまで、常に現在のカーソル位置からレコードが取り出されます。

非カーソル・アドバンシング API

非カーソル・アドバンシング JDECACHE カーソル API は、レコードを取り出す前にカーソルを進めません。代わりに、カーソルは取出し済みレコードの位置に残ります。非カーソル・アドバンシング・フェッチ API の例を次に示します。

- jdeCacheUpdate
- jdeCacheDelete

レコードの更新

キーがわかっている特定のレコードを更新する場合は、jdeCacheFetchPosition を呼び出して既知のキーを渡し、そのキーに一致するレコードの位置にカーソルを合わせます。カーソルが目的の位置をポイントしたら jdeCacheUpdate を呼び出し、jdeCacheFetchPosition の呼出しで使用した場合と同じ HJDECURSOR を渡します。

インデックス・キーを変更するとキャッシュがレコードをソートし直し、カーソルが更新された位置をポイントします。ただし、jdeCacheFetch を呼び出すと、更新されたセットの次のレコードが呼び出されます。そのため、正しいレコードが取り出されないことがあります。これは、変更されたインデックス・キーによってレコードの順序が変更されるためです。

一連のレコードを更新するには、必要に応じて jdeCacheFetchPosition を呼び出し、その先頭にカーソルを戻します。次に jdeCacheUpdate を呼び出し、jdeCacheFetchPosition の呼出しで使用した場合と同じ HJDECURSOR を受け渡します。この呼び出しは、カーソルがポイントしているレコードのみを更新します。残りのレコードを更新するには、一連のレコードの最後に到達するまで jdeCacheFetch を繰り返し呼び出し、jdeCacheFetchPosition の呼出しで使用した場合と同じ HJDECURSOR を受け渡します。順次更新は、インデックス・キーの値を変更した場合は正常に行われませんが、インデックス・キー以外の値を更新する場合は正常に行われます。

レコードの削除

キーがわかっている特定のレコードを削除する場合は、最初に jdeCacheFetchPosition を呼び出して、キーに一致するレコードの位置にカーソルを合せます。次に、jdeCacheDelete をキャッシュからレコードを削除します。jdeCacheDelete には jdeCacheFetchPosition を呼び出したときに使用した場合と同じ HJDECURSOR を受け渡します。レコードを削除したら jdeCacheFetch を使用して、削除されたレコードの次のレコードを取り出します。このプロセスは、jdeCacheDelete を呼び出したときにのみ有効です。

また、jdeCacheDeleteAll を呼び出し、削除するレコードのフル・キーを受け渡して特定のレコードを削除することもできます。この場合、jdeCacheDeleteAll の後で jdeCacheFetch は機能しませんが、これは jdeCacheFetchPosition か jdeCacheResetCursor によって解決できます。

レコードの一連のセットを削除するには、最初に jdeCacheFetchPosition を呼び出して、セットの最初のレコードにカーソルを合わせるか、または jdeCacheDeleteAll を呼び出して、セットの最初のレコードを削除します。次に、jdeCacheDelete を続けて呼び出します。この場合、jdeCacheDeleteAll の後で jdeCacheFetch は機能しませんが、これは jdeCacheFetchPosition か jdeCacheResetCursor によって解決できます。

パーシャル・キーに一致するレコードを削除する場合は、jdeCacheDeleteAll を呼び出してパーシャル・キーを受け渡します。パーシャル・キーに一致するすべてのレコードが削除されます。この API を呼び出した後、jdeCacheFetch は機能しません。

jdeCacheFetchPosition API

jdeCacheFetchPosition API は、データセットの特定のレコードを検索します。そのため、特定のキーを必要とします。この API は、フル・キーとパーシャル・キーによる検索を実行します。

注:

キーの数として 0 を受け渡すと、フル・キー検索が実行されます。

参照

- パーシャル・キーについては、『開発ツール』ガイドの「JDECACHE のパーシャル・キー」

jdeCacheFetchPositionByRef API

jdeCacheFetchPositionByRef API は、データセットのアドレスを返します。この API は、キャッシュの 1 つのレコードを検索し、データの参照(ポインタ)を返します。jdeCacheFetchPositionByRef は、キャッシュに保管されている 1 つの大きなデータ・ブロックを取り出します。キャッシュが空か、または複数のレコードを保管している場合、この API は失敗します。

カーソルのリセット

JDECACHE カーソルは複数カーソルをサポートし、またデータセットでのカーソル振幅も無制限にサポートします。これは、カーソルがデータセットの先頭から最後まで繰り返し何度も往復できることを意味します。カーソルは前方にのみ移動します。カーソルをリセットする（カーソルをデータセットの先頭に戻す）には、jdeCacheResetCursor API を呼び出す必要があります。これにより、最新の JDECACHE データセットが得られます。

また、現在のデータセットの外部にある特定の位置にカーソルをリセットするには、jdeCacheFetchPosition API を呼び出します。

参照

- カーソルのリセットについては、『開発ツール』ガイドの「JDECACHE データセット」

カーソルのクローズ

カーソルが不要になったときは、jdeCacheCloseCursor の呼出しを使用してカーソルを閉じます。これにより、データセットとカーソルが共に閉じられます。その後、閉じた HJDECURSOR を受け渡して JDECACHE API を呼び出しても、jdeCacheOpenCursor を呼び出していなければいずれもエラーとなります。

JDECACHE カーソルは長時間開いていてもオーバーヘッドは生じませんが、それに必要なメモリを解放するには不要になった時点ですぐに閉じてください。

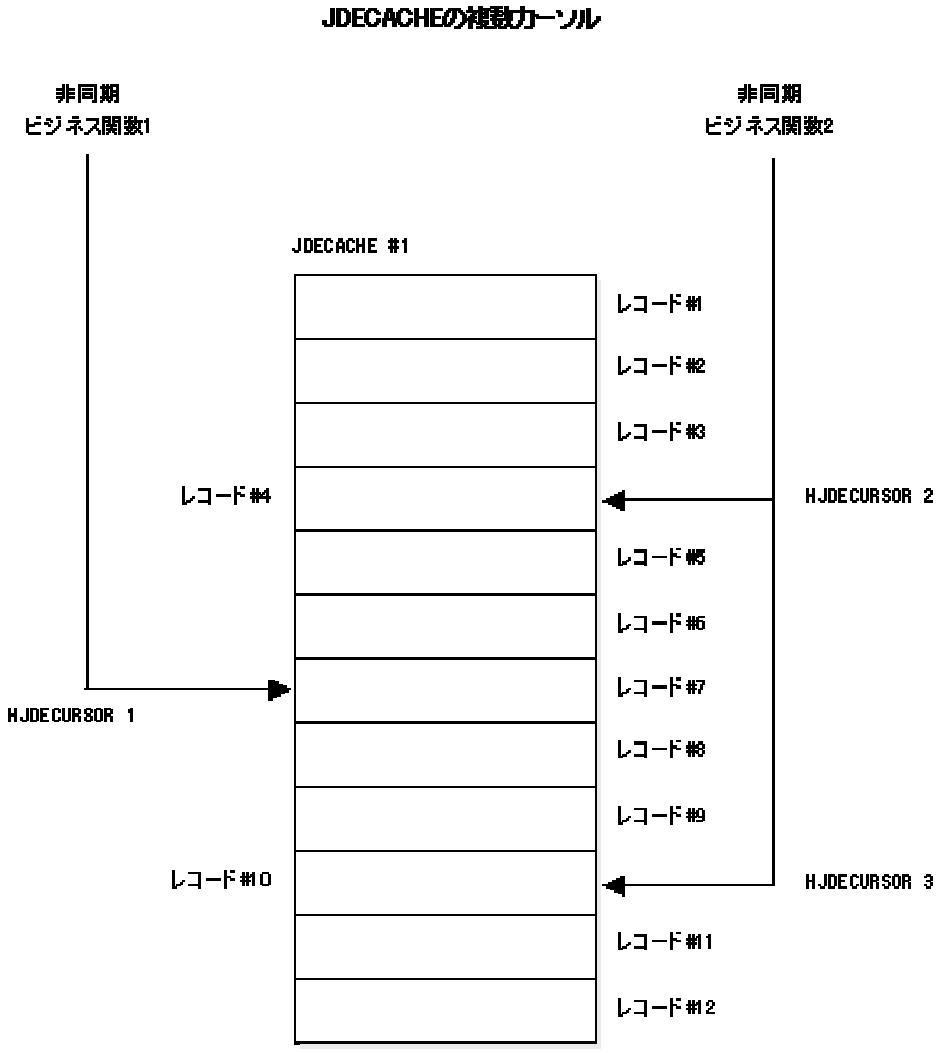
JDECACHE の複数カーソル・サポート

JDECACHE は複数のカーソルをサポートします。これは、キャッシュごとにカーソルが最大 100 まで用意でき、同時にそのキャッシュのデータセットにアクセスできることを意味します。

JDECACHE の複数カーソルは、2 つ以上の非同期処理ビジネス関数が 1 つのキャッシュを使用できるように設計されています。これは、非同期処理ビジネス関数がカーソルを開き、キャッシュでの相対位置が相互に独立しているキャッシュにアクセスできることを意味します。1 つのビジネス関数でカーソルを移動しても、開いている別のカーソルに一切影響しません。

J.D. Edwards ソフトウェア・アプリケーション・グループには、複数カーソルの使用を制限するものがあります。たとえば、複数カーソルは必要な場合に限り使用してください。また、両方のカーソルがレコードを取り出している場合を除き、2 つのカーソルを使用して同じレコードに同時にポイントすることはできません。

次の図に、JDECACHE での複数カーソルを示します。



JDECACHE のパーシャル・キー

JDECACHE のパーシャル・キーは、既定のインデックスと同じ順序で配列され、既定のインデックスの最初のキーから始まる JDECACHE キーのサブセットです。たとえば、既定のインデックスが N 個のキーから構成されている場合、パーシャル・キーは、キー1、2、3、4...N-1 の、その特定の順序で配列されたサブセットになります。この順序は重要です。パーシャル・キーのコンポーネントは、インデックスのキー・コンポーネントと同じ順序で表示されなければなりません（インデックスは、jdeCacheInit に受け渡されます）。

たとえば、インデックスが A、B、C、D、E という順序でフィールドを保管する構造体として定義されているとします。

この場合、このインデックスから同期できるパーシャル・キーは、A、AB、ABC、ABCD という順序になります。

これは、既定のインデックスが A、B、C、D、E の場合に同期することができるパーシャル・キーの唯一のセットです。

JDECACHE パーシャル・キーの使用法

JDECACHE パーシャル・キーは JDECACHE カーソルを実現します。JDECACHE パーシャル・キーを実装するときは、JDECACHE カーソルが、既定のインデックスすなわち完全なインデックスに従ってキャッシュに配列されたレコードから構成される JDECACHE データセットで動作することを考慮してください。jdeCacheFetchPosition API を呼び出してパーシャル・キーを受け渡すと、JDECACHE カーソルが起動し、パーシャル・キーに一致する JDECACHE データセットの最初のレコードをポイントします。jdeCacheFetchPosition API が呼び出された場合、その後 jdeCacheFetch を呼び出すと、すでに取り出されたレコードからデータセットの最後のレコードまでの、データセットのすべてのレコードが取り出されます。カーソルは、パーシャル・キーに一致する最後のレコードで停止せず、レコードがパーシャル・キーに一致しない場合でも、次の jdeCacheFetch の呼出しによって次のレコードを取り出していきます。パーシャル・キーが jdeCacheFetchPosition に送られると、パーシャル・キーは JDECACHE にフェッチの開始位置を指示するだけです。JDECACHE データセットのレコードは常に順序付けられているので、フェッチによりパーシャル・キーに一致するすべてのレコードが必ず取得されます。

JDECACHE は、jdeCacheFetchPosition の 4 番目のパラメータで、この API に送られるキーのキー・フィールド数が示されるので、パーシャル・キーが受け渡されることを認識します。キー・フィールドの数が、jdeCacheInit の呼び出し時に示されたキーの数より少ない場合、そのキーはパーシャル・キーになります。キーの数が N の場合、JDECACHE はパーシャル・キー機能を実現するために、最初の N 個のキー・フィールドを使用して比較を行います。jdeCacheFetchPosition の呼出しで、jdeCacheInit の呼出しで指定された数よりも大きいキーの数が渡されると、エラーが返されます。

パーシャル・キーを削除するには、jdeCacheDeleteAll を呼び出します。この呼出しにより、パーシャル・キーに一致するすべてのレコードが削除されます。使用するパーシャル・キーを JDECACHE に指示するには、キー・フィールドの数をこの API に受け渡します。

構造体キー・フィールドの実数と、jdeCacheFetchPosition または jdeCacheDeleteAll に送るキーの数を示す数値が一致することを確認してください。

JDECACHE のサンプル・プログラム

ここでは、住所録テーブル(F0101)をキャッシュに読み込むプログラムのサンプルを紹介します。キャッシュには、住所番号である ABAN8 を使ってインデックスを設定します。このキャッシュとの関係で、個々の JDECACHE API を呼び出し、キャッシュされたデータを処理します。このプログラムは、必要に応じてカット&ペーストできますが、結果を印刷しません。入出力の呼出しは適宜配置できます。

```
/*************************************************************************/
/*                                                                       */
/*                Source          :       TESTCACHE.C            */
/*                                                                       */
/*                Programmer      :       Chikoore, T                    */
/*                                                                       */
/*                Date            :       12/09/96                       */
/*                                                                       */
/*                Description     :       To illustrate the use  */
/*                                        of the jdeCache APIs   */
/*                                        which will be used to  */
/*                                        simulate the 400's     */
/*                                        work files                     */
/*                                                                       */
/*************************************************************************/
#include <windows.h>
#include <windowsx.h>
#include <jde.h>
#include <F0101.H>
JDE_MEMORY_POOL GPoolCommon;
int
WinMain(
    HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPSTR lpCmdLine,
    int nCmdShow)
{
    /*Environment variables*/
    HENV            hEnv                    =       NULL;

HUSER               hUser                   =       NULL;
```

```
            /*JDB - Related variables*/
JDEDB_RESULT        JDBcode                         =       JDEDB_FAILED;
HREQUEST            hAccessRequest                  =       NULL;
/*JDECACHE - Related variables*/
char                cNotUsed                        =       '¥0';
JDECM_RESULT        jdeCachecode                    =       JDECM_FAILED;
HCACHE              hF0101Cache                     =       NULL;
HJDECURSOR                  hF0101CacheCursor1              =       NULL;
HJDECURSOR                  hF0101CacheCursor2              =       NULL;
JDECMINDEXSTRUCT Index[1];
/*Table - Related variables*/
ID                  idTable                         =       ID_F0101;
F0101               dsInsertStruct,dsRetrieveStruct,f0101;
/*Just variables*/
KEY1_F0101          dsF0101CacheKey;
short int           nNumColsInIndex         =       1;
short int           nCursor1FetchCounter            =       0;
short int           nCursor2FetchCounter            =       0;
/*************************************************************************/
/*                                                                       */
/*                          Section 1                                    */
/*                                                                       */
/*  Initialize the JDB and USER environments as usual                    */
/*  The cache environment is implicitly initialized by the               */
/*  JDB_InitEnv API, you do not have to worry about it                   */
/*                                                                       */
/*************************************************************************/

/*Initialize the JDE memory pool management utility*/
GPoolCommon = jdeMemoryManagementInit();
/*Initalize the Environment*/
JDBcode = JDB_InitEnv(&hEnv);
if(JDBcode != JDEDB_PASSED)
{
    jdeMemoryManagementTerminate();
    return 0;
}   /*END IF*/
```

```
/*initialize the User*/
JDBcode = JDB_InitUser(hEnv,&hUser,"P0101",JDEDB_COMMIT_AUTO);
if(JDBcode != JDEDB_PASSED)
{
    JDB_FreeEnv(hEnv);
    jdeMemoryManagementTerminate();
    return 0;
}   /*END IF*/
/*********************************************************************/
/*                                                                   */
/*                          Section 2                                */
/*                                                                   */
/*  Set up the index key that will be used to access the cache.      */
/*  One and only one index is allowed per cache.                     */
/*                                                                   */
/*  CODE:   Here the address book number (aban8) is being */
/*          set up as the key.                                       */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
/*Initialize the structure*/
memset(&Index,0x00,sizeof(JDECMINDEXSTRUCT));
/*Set up the cache index*/
Index->nKeyID = 1;
Index->nNumSegments = 1;
Index->CacheKey[0].nOffset = offsetof(F0101, aban8);
Index->CacheKey[0].nSize = sizeof(f0101.aban8);
Index->CacheKey[0].idDataType = EVDT_MATH_NUMERIC;
/*********************************************************************/
/*                                                                   */
/*                          Section 3                                */
/*                                                                   */
/*  Initialize the cache. The address of the cache handle    */
/*  for that particular cache is passed as well as the               */
/*  cache name and index. The cache handle will be use to    */
/*  identify this particular cache when calling any of the   */
/*  cache APIs                                                       */
```

```
/*                                                                    */
/*   CODE:    The cache named F0101 is being initialized             */
/*                                                                    */
/*                                                                    */
/********************************************************************/
/*Initialize the user cache*/
jdeCachecode = jdeCacheInit(hUser, &hF0101Cache, "F0101",Index);
if(jdeCachecode != JDECM_PASSED)
{
      JDB_FreeUser(hUser);
      JDB_FreeEnv(hEnv);
      jdeMemoryManagementTerminate();
      return 0;
}     /*END IF*/
/************************************************************************/
/*                                                                    */
/*                              Section 4                             */
/*                                                                    */
/*   CODE:    We are going to open the address book table            */
/*            (F0101)and place the contents of the whole table       */
/*            into the F0101cache which we have just initalized      */
/*            in Section 3.                                          */
/*                                                                    */
/************************************************************************/
/*Open the address book table*/
JDBcode = JDB_OpenTable(hUser, idTable, 0, NULL, 0 ,NULL,&hAccessRequest);
if(JDBcode !=  JDEDB_PASSED)
{
      jdeCacheTerminate(hUser, hF0101Cache);
      JDB_FreeUser(hUser);
      JDB_FreeEnv(hEnv);
      jdeMemoryManagementTerminate();
      return 0;
}     /*END IF*/
/*Select all rows of the address book*/
JDBcode = JDB_SelectAll(hAccessRequest);
if (JDBcode != JDEDB_PASSED)
```

```
{
      jdeCacheTerminate(hUser, hF0101Cache);
      JDB_CloseTable(hAccessRequest);
      JDB_FreeUser(hUser);
      JDB_FreeEnv(hEnv);
      jdeMemoryManagementTerminate();
      return 0;
}     /*END IF*/
/*Initialize the structure*/
memset(&dsInsertStruct,0x00,sizeof(F0101));
/*Fetch all rows of the address book*/
while (JDB_Fetch(hAccessRequest, &dsInsertStruct, FALSE) == JDEDB_PASSED)
{
      /*Add the row to the cache*/
      jdeCachecode = jdeCacheAdd(hF0101Cache, (void *) &dsInsertStruct,
      (long int)
      sizeof(F0101));
      if(jdeCachecode == JDECM_FAILED)
      {
                  break;
      }           /*END IF*/
      /*Re-Initialize the structure*/
      memset(&dsInsertStruct,0x00,sizeof(F0101));
}     /*END WHILE*/
/*************************************************************************/
/*                                                                       */
/*                              Section 5                                */
/*                                                                       */
/*   CODE:    We are going to open the cursors. Please                  */
/*            note that cursors can only be opened after                */
/*            the cache has been filled, since the cursor has  */
/*            to point to the first record in the cache                 */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
/*Initialize the first cursor*/
jdeCachecode = jdeCacheOpenCursor(hF0101Cache, &hF0101CacheCursor1);
```

```
if(jdeCachecode != JDECM_PASSED)
{
      jdeCacheTerminate(hUser, hF0101Cache);
      JDB_CloseTable(hAccessRequest);
      JDB_FreeUser(hUser);
      JDB_FreeEnv(hEnv);
      jdeMemoryManagementTerminate();
      return 0;
}     /*END IF*/
/*Initialize the second cursor*/
jdeCachecode = jdeCacheOpenCursor(hF0101Cache, &hF0101CacheCursor2);
if(jdeCachecode != JDECM_PASSED)
{
      jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor1);
      jdeCacheTerminate(hUser, hF0101Cache);
      JDB_CloseTable(hAccessRequest);
      JDB_FreeUser(hUser);
      JDB_FreeEnv(hEnv);
      jdeMemoryManagementTerminate();
      return 0;
}     /*END IF*/
/*************************************************************************/
/*                                                                       */
/*                              Section 6                                */
/*                                                                       */
/*   CODE:    We are going to find the record with key 1001   */
/*            in the F0101 cache. If its not there, we add it.   */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
/*Initialize the key structure. It is important that this is done*/
memset(&dsF0101CacheKey,0x00,sizeof(KEY1_F0101));
/*Set up the key*/
ParseNumericString(&dsF0101CacheKey.aban8,"1001");
/*Initialize the structure to retrieve into*/
memset(&dsRetrieveStruct,0x00,sizeof(F0101));
/*Find the cache record keyed 1001 using cursor1*/
```

```
jdeCachecode = jdeCacheFetchPosition(hF0101Cache,
hF0101CacheCursor1,&dsF0101CacheKey,nNumColsInIndex,&dsRetrieveStruct,
sizeof(F0101));

if(jdeCachecode != JDECM_PASSED)

{

      /*Initialize the structure*/

      memset(&dsInsertStruct,0x00,sizeof(F0101));

      /*Fill out the structure to insert here*/

      ParseNumericString(&dsInsertStruct.aban8,"1001");

      strcpy(dsInsertStruct.abtax, "000011171");

      strcpy(dsInsertStruct.abalph, "John Doe");

      strcpy(dsInsertStruct.abdc, "DOEJOHN");

      strcpy(dsInsertStruct.abmcu, "1001");

      /*Add the record to the cache*/

      jdeCachecode = jdeCacheAdd(hF0101Cache, (void *) &dsInsertStruct,
      (long int)sizeof(F0101));

      if(jdeCachecode != JDECM_PASSED)

      {

                  jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor1);

                  jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor2);

                  jdeCacheTerminate(hUser, hF0101Cache);

                  JDB_CloseTable(hAccessRequest);

                  JDB_FreeUser(hUser);

                  JDB_FreeEnv(hEnv);

                  jdeMemoryManagementTerminate();

                  return 0;

      }           /*END IF*/

}     /*END IF*/

/*************************************************************************/

/*                                    Section 7                          */

/*                                                                       */

/*    CODE:    We are going to update the record with key 1002          */

/*                in the F0101 cache.                                    */

/*                                                                       */

/*                                                                       */

/*************************************************************************/

/*Set up the key*/

ParseNumericString(&dsF0101CacheKey.aban8,"1002");
```

```
/*Find the cache record keyed 1002 using cursor1 and keep the cursor

pointing to that record because we want to update it*/

jdeCachecode = jdeCacheFetchPosition(hF0101Cache, hF0101CacheCursor1,
&dsF0101CacheKey, nNumColsInIndex,&dsRetrieveStruct, sizeof(F0101));

if(jdeCachecode != JDECM_PASSED)

{

      jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor1);

      jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor2);

      jdeCacheTerminate(hUser, hF0101Cache);

      JDB_CloseTable(hAccessRequest);

      JDB_FreeUser(hUser);

      JDB_FreeEnv(hEnv);

      jdeMemoryManagementTerminate();

      return 0;

}

/*Update the structure we have just retrieved here*/

strcpy(dsRetrieveStruct.abalph, "Put Alpha Name Here");

/*Update the record with the record just retrieved. The cursor is already
pointing to this record so we do not need a key*/

jdeCachecode =jdeCacheUpdate(hF0101Cache, hF0101CacheCursor1,
&dsRetrieveStruct,sizeof(KEY1_F0101));

if(jdeCachecode != JDECM_PASSED)

{

      jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor1);

      jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor2);

      jdeCacheTerminate(hUser, hF0101Cache);

      JDB_CloseTable(hAccessRequest);

      JDB_FreeUser(hUser);

      JDB_FreeEnv(hEnv);

      jdeMemoryManagementTerminate();

      return 0;

}     /*END IF*/

/************************************************************************/

/*                                                                      */

/*                              Section 8                               */

/*                                                                      */

/*   CODE:    We are going to delete the record with key 1001          */

/*            in the F0101 cache.                                       */
```

```
/*                                                                    */
/*                                                                    */
/*                                                                    */
/**********************************************************************/
/*Set up the key*/
ParseNumericString(&dsF0101CacheKey.aban8,"1001");
/*Find the cache record keyed 1002 using cursor1 and keep the cursor
pointing to that record because we want to update it*/
jdeCachecode = jdeCacheFetchPosition(hF0101Cache, hF0101CacheCursor1,
&dsF0101CacheKey,nNumColsInIndex,&dsRetrieveStruct,sizeof(F0101));
if(jdeCachecode != JDECM_PASSED)
{
      jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor1);
      jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor2);
      jdeCacheTerminate(hUser, hF0101Cache);
      JDB_CloseTable(hAccessRequest);
      JDB_FreeUser(hUser);
      JDB_FreeEnv(hEnv);
      jdeMemoryManagementTerminate();
      return 0;
}
/*Delete the record keyed 1001 using cursor 1.*/
jdeCachecode = jdeCacheDelete(hF0101Cache, hF0101CacheCursor1);
if(jdeCachecode != JDECM_PASSED)
{
      jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor1);
      jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor2);
      jdeCacheTerminate(hUser, hF0101Cache);
      JDB_CloseTable(hAccessRequest);
      JDB_FreeUser(hUser);
      JDB_FreeEnv(hEnv);
      jdeMemoryManagementTerminate();
      return 0;
}     /*END IF*/
/**********************************************************************/
/*                                                                    */
/*                          Section 9                                 */
```

```
/*                                                                    */
/*   CODE:    We are going to look for the record with key           */
/*            1001 in the F0101 cache. We have just deleted it       */
/*            so we should not find it.                              */
/*                                                                    */
/*                                                                    */
/*                                                                    */
/*                                                                    */
/**********************************************************************/
/*Initialize the structure to retrieve into*/
memset(&dsRetrieveStruct,0x00,sizeof(F0101));
/*Find the cache record keyed 1001*/
jdeCachecode = jdeCacheFetchPosition(hF0101Cache, hF0101CacheCursor1,
&dsF0101CacheKey, nNumColsInIndex,&dsRetrieveStruct, sizeof(F0101));
if(jdeCachecode == JDECM_PASSED)
{
      jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor1);
      jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor2);
      jdeCacheTerminate(hUser, hF0101Cache);
      JDB_CloseTable(hAccessRequest);
      JDB_FreeUser(hUser);
      JDB_FreeEnv(hEnv);
      jdeMemoryManagementTerminate();
      return 0;
}     /*END IF*/
/**********************************************************************/
/*                                                                    */
/*                              Section 10                            */
/*                                                                    */
/*   CODE:    Illustration of multiple cursors at                    */
/*            work. We will use the two declared                     */
/*             cursors to alternatingly retrieve data.          */
/*            Cursor 2 should alway retrieve what                    */
/*            Cursor 1 has already retrieved.                        */
/*                                                                    */
/*                                                                    */
/**********************************************************************/
```

```
/*First we need to reset both cursors to make
sure that we are starting from the beginning of the dataset.*/
/*Reset cursor 1*/
jdeCachecode = jdeCacheResetCursor(hF0101Cache, hF0101CacheCursor1);
if(jdeCachecode != JDECM_PASSED)
{
      /*reset failed*/
      jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor1);
      jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor2);
      jdeCacheTerminate(hUser, hF0101Cache);
      JDB_CloseTable(hAccessRequest);
      JDB_FreeUser(hUser);
      JDB_FreeEnv(hEnv);
      jdeMemoryManagementTerminate();
      return 0;
}
/*Reset cursor 2*/
jdeCachecode = jdeCacheResetCursor(hF0101Cache, hF0101CacheCursor2);
if(jdeCachecode != JDECM_PASSED)
{
      /*reset failed*/
      jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor1);
      jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor2);
      jdeCacheTerminate(hUser, hF0101Cache);
      JDB_CloseTable(hAccessRequest);
      JDB_FreeUser(hUser);
      JDB_FreeEnv(hEnv);
      jdeMemoryManagementTerminate();
      return 0;
}
/*Clear the structure we are to retrieve records into*/
memset(&dsRetrieveStruct,0x00,sizeof(F0101));
/*Initialize the counter associated with cursor1*/
nCursor1FetchCounter=0;
/*Fetch the next record using cursor 1*/
while(jdeCacheFetch(hF0101Cache,hF0101CacheCursor1,&dsRetrieveStruct,NULL)
!= JDECM_FAILED)
```

```
{
    /*Increment the counter associated with cursor1*/
    nCursor1FetchCounter++;
    /*Check if cursor 1 has reached the max number of fetches*/
    if(nCursor1FetchCounter == 2)
    {
        /*Initialize the counter associated with cursor1*/
        nCursor2FetchCounter = 0;
        /*Fetch the next record using cursor 2*/
        while(jdeCacheFetch(hF0101Cache,hF0101CacheCursor2,
        &dsRetrieveStruct,NULL) != JDECM_FAILED)
    }       /*END IF*/
}   /*END WHILE*/
/************************************************************************/
/*                                                                      */
/*                          Section 11                                  */
/*                                                                      */
/*  CODE:   END!!!                                                      */
/*          Do the normal clean up work.                                */
/*                                                                      */
/*                                                                      */
/************************************************************************/
/*First close all open cursor before terminating the cache*/
/*Close open cursor 1*/
jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor1);
/*Close open cursor 2*/
jdeCacheCloseCursor(hF0101Cache,hF0101CacheCursor2);
jdeCachecode = jdeCacheTerminate(hUser,hF0101Cache);
if(jdeCachecode != JDECM_PASSED)
{
    /*This is just a check for failure otherwise its the same as below*/
    JDB_CloseTable(hAccessRequest);
    JDB_FreeUser(hUser);
    JDB_FreeEnv(hEnv);
    jdeMemoryManagementTerminate();
    return 0;
}   /*END IF*/
```

```
/*Close the table*/
JDBcode = JDB_CloseTable(hAccessRequest);
/*Free the user*/
JDBcode = JDB_FreeUser(hUser);
/*Free the environment, the cache is implicitly killed here*/
JDBcode = JDB_FreeEnv(hEnv);
/*Terminate the memory management utility*/
jdeMemoryManagementTerminate();

    return(0);

}
```

JDECACHE の標準

JDECACHE を使用する場合は、次の標準に従ってください。

キャッシュ・ビジネス関数ソース名

キャッシュ・ビジネス関数名は、ビジネス関数の標準命名規則に従ってください。

キャッシュ・ビジネス関数ソース記述

キャッシュ・ビジネス関数のソース記述には、次の標準が適用されます。

- キャッシュ・ビジネス関数記述は、ビジネス関数記述標準に従ってください。
- 最初の語は、名詞の Cache にします。
- 2 番目の語は、動詞の Process にします。
- 個別キャッシュ関数では、Process の後の語句でキャッシュを表します。共通キャッシュ関数では、Process の後の語句で、個別キャッシュ関数が属するグループを表します。

キャッシュ・ビジネス関数記述

キャッシュ・ビジネス関数記述には、次の標準が適用されます。

- ソース・ファイルが個別関数を含む場合、その関数名とソース名を一致させます。
- ソース・ファイルが一群のキャッシュ関数を含む場合、個々の関数名はキャッシュ・ビジネス関数ソース記述標準と同じ標準に従ってください。

キャッシュ・プログラミング標準

キャッシュのプログラミングには、次のように各種標準が適用されます。

- 一般標準
- キャッシュの終了とクリア
- キャッシュ名
- キャッシュ・データ構造体の定義
- データ構造体標準データ項目
- キャッシュ・アクション・コード標準
- グループ・キャッシュ・ビジネス関数ヘッダー・ファイル
- 個別キャッシュ・ビジネス関数ヘッダー・ファイル

一般標準

標準キャッシュ関数には、個別キャッシュ関数およびグループ・キャッシュ関数の 2 種類があります。個別キャッシュ関数は、1 つのキャッシュでファイル・サーバーとして機能する単一のビジネス関数です。グループ・キャッシュ関数は、複数の定義済みキャッシュでファイル・サーバーとして機能する単一のビジネス関数です。キャッシュのプログラミングには、次の一般標準が適用されます。

- キャッシュ API は、標準化されたキャッシュ関数でのみ使用します。標準キャッシュ関数を使用すると、コードの作成と保守が容易に行えます。標準関数はキャッシュの使用を簡素化します。
- キャッシュ関数が標準のキャッシュ・アクションを含むことを確認します。また、キャッシュ固有の、キャッシュに対する追加アクションを含めることもできます。
- キャッシュの初期化、キャッシュのロード、キャッシュからのレコードの取出し、キャッシュからのレコードの削除、レコードの修正、キャッシュの終了の各操作を実行するには、キャッシュ関数を使用します。

キャッシュの終了とクリア

キャッシュは、引き続き使用可能にしておく必要があるかどうかに応じて、終了するかクリアします。この判断は、アプリケーションの設計によって異なります。

必要に応じて、キャッシュを終了する前にキャッシュ・データ構造体のメモリを解放します。キャッシュ・データ構造体は、キャッシュ・インデックス構造体とは異なります。キャッシュを終了するとキャッシュ・インデックス構造体のメモリが解放されますが、キャッシュ・データ構造体には影響しません。

キャッシュ名

キャッシュ名には、次の標準が適用されます。

- キャッシュ名は最大 50 文字とします。
- キャッシュ名は、キャッシュの使い方に応じて、データ構造体名、またはデータ構造体名にジョブ番号を付加したものにします。
- キャッシュを終了する場合は、キャッシュ名にジョブ番号を追加します。
- キャッシュがマスター・ビジネス関数で使用されているときには、キャッシュはアプリケーションによって終了されません。ジョブ番号がキーとしてキャッシュに保管されます。キャッシュを終了する代わりに、ジョブ番号を含むすべてのレコードがトランザクションの終了時にクリアされます。キャッシュは非同期処理により終了されません。この場合、キャッシュ名はデータ構造体名のみを含みます。

キャッシュ・データ構造体の定義

キャッシュ・データ構造体は、次の要素を含みます。

- グループ・キャッシュ関数

  グループ・キャッシュ関数のデータ構造体は、個別キャッシュのレイアウトとインデックスでもあります。このデータ構造体の場合は、typedef を実行してビジネス関数ヘッダー・ファイルに貼り付けます。

- 個別キャッシュ関数

  個別キャッシュ関数のキャッシュ・レイアウトとキャッシュ・インデックスは、ビジネス関数ヘッダー・ファイルで定義する必要があります。

データ構造体標準データ項目

次のデータ項目は、すべてのキャッシュ・ビジネス関数に対する標準のデータ項目です。これらのデータ項目は、データ構造体の最初のデータ項目にします。

- NACTN – キャッシュ・アクション・コード
- NKEYS – キーの数
- CURSOR – キャッシュ・カーソル
- DTAI – エラー・メッセージ ID
- JOBS – ジョブ番号（任意指定）

キャッシュに保管されている情報を受け渡す追加のデータ項目は、標準データ項目の後に設定します。キャッシュ・インデックスの順序に従って、最初にキー・フィールドをリストアップすることをお勧めします。

標準キャッシュ・アクション・コード

事前定義済みの次の標準キャッシュ・アクション・コードを使用します。

CACHE_GET

- カーソルを開きます。
- jdeCacheFetch API を使って、キャッシュから単一のレコードを取り出します。

  フェッチ（取出し）は、フル・キーとパーシャル・キーのどちらによっても実行できます。使用されるキーの数は、ビジネス関数データ構造体の標準データ項目を介して関数に受け渡されます。

- キャッシュ・カーソルはフェッチを実行する際に不要であり、キャッシュ・カーソルはレコードが取り出された後に返されません。
- 成功または失敗を返します。

CACHE_ADD

- jdeCacheAdd API を使ってキャッシュにレコードを挿入します。
- すべてのキーを定義して、すべてのキー・フィールドの値を受け渡す必要があります。
- キャッシュではレコードの重複は許可されません。
- キャッシュ・カーソルは、キャッシュにレコードを追加する際に不要であり、キャッシュ・カーソルは、レコードが追加された後に返されません。
- 成功または失敗を返します。

CACHE_UPDATE

- カーソルを開きます。
- オプションで、jdeCacheFetchPosition API を使って、更新用のレコードをキャッシュから取り出します。
- jdeCacheUpdate API を使って、取り出されたレコードを更新します。
- すべてのキーを定義して、すべてのキー・フィールドの値を受け渡す必要があります。
- キャッシュ・カーソルは、キャッシュのレコードを更新する際に不要であり、返されません。
- 成功または失敗を返します。

CACHE_ADD_UPDATE

- 既存のレコードを削除して新規レコードに置換します。レコードが存在しない場合は、追加します。
- jdeCacheFetch API を使用して、レコードを取り出します。レコードが見つかると、jdeCacheDelete API を使用してレコードを削除します。jdeCacheAdd API を使用して、ビジネス関数データ構造体を介して値を受け渡し、新しいレコードを追加します。
- すべてのキーを定義して、すべてのキー・フィールドの値を受け渡す必要があります。
- キャッシュ・カーソルは不要であり、返されません。
- 新しいレコードが追加されたら、成功または失敗を返します。

CACHE_DELETE

- 受け渡されたキーの数に従って、1 つのレコード、1 組のレコードまたはすべてのレコードをキャッシュから削除します。
- キーの数がインデックスにおけるキーの総数に等しい場合、jdeCacheDelete API を使って 1 つのレコードをキャッシュから削除します。
- パーシャル・キーを定義して一連のレコードをキャッシュから削除するために使うことができます。jdeCacheDelete API を使ってレコードを削除します。
- キーの数がゼロに設定されている場合、キャッシュからすべてのレコードを削除します。jdeCacheDeleteAll API を使ってレコードを削除します。
- キャッシュ・カーソルは、キャッシュのレコードを削除する際に不要であり、返されません。
- 成功または失敗を返します。

CACHE_GET_NEXT

- カーソルを開き、キャッシュ・カーソルが空の場合は jdeCacheFetchPosition を使用して最初のレコードを取り出します。
- カーソルに値が含まれている場合は、次のレコードを取り出します。
- フル・キーか、パーシャル・キーを受け渡します。キーの数は一定にします。
- ファイルの最後を確認するために、前のレコードと次のレコードで定義されているキーの数を比較します。
- キーの突合せに失敗するとカーソルを閉じ、データ・ポインタを解放します。

CACHE_TERMINATE

jdeCacheTerminate API を使ってキャッシュを終了します。キャッシュのユーザー数がゼロのときにのみキャッシュを終了します。

CACHE_TERMINATE_ALL

- jdeCacheTerminateAll API を使ってキャッシュを終了します。
- データ・ポインタを解放します。
- キャッシュにアクセスしているユーザーの数に関係なくキャッシュを終了します。

CACHE_CLOSE_CURSOR

jdeCacheCloseCusor API を使ってカーソルを閉じます。

追加機能

J.D. Edwards には、アプリケーションの機能拡張に使用できる追加の開発ツールと機能が用意されています。次の追加機能があります。

- 処理オプション
- トランザクション処理
- レコードのロック
- 通貨
- ワンポイント・ヒント

処理オプション

処理オプションを使用して、対話型アプリケーションまたはバッチ・アプリケーションにおけるデータの処理方法を制御することができます。処理オプションは、アプリケーションやレポートを表示したり、それらが動作する仕方を変更するために使用できます。同じアプリケーションの異なるバージョンにそれぞれ独自の処理オプション値を組み込んで、新しいアプリケーションを作成せずにアプリケーションの動作を変更することができます。また、処理オプションを使用して次のような内容を指示することができます。

- ユーザーがシステムを移動するためのパスを制御する。
- デフォルト値を設定する。
- さまざまな会社やユーザーに合わせてアプリケーションをカスタマイズする。
- フォームと報告書の書式を制御する。
- レポートの改頁や合計計算を制御する。
- 関連アプリケーションのデフォルト・バージョンまたはバッチ処理を指定する。

処理オプションは、アプリケーションの実行時に自動表示されるように定義できます。

また、処理オプションのバージョン作成が必要な場合もあります。処理オプションのバージョンの作成手順は、対話型バージョンの作成手順と同じです。

処理オプション・テンプレート

処理オプション・テンプレートには、1 つ以上の処理オプションが含まれます。各処理オプションはテンプレートでそれぞれのローに表示され、次の 3 つの変数で定義されます。

タブ・タイトル 処理オプションを分類するタイトル。このテキストは、処理オプションのタブ上に表示されます。

コメント 処理オプションと共にフォームに表示される任意のテキスト。各コメントはページに、処理オプションの代わりに表示されます。コメントを追加すると、処理オプション用のスペースが減ります。

データ項目 データ辞書の項目。どの処理オプションも、データ辞書に登録されているデータ項目でなければなりません。ビジネスユニットの範囲や日付など、デフォルト値を割り当てるデータ項目を指定します。

実行時には、処理オプション・テンプレートのうち、ページと呼ばれる領域に一連のタブが表示されます。各タブは、処理オプションのカテゴリを表します。タブをクリックするとページが変わり、そのカテゴリの処理オプション・セットが表示されます。

注意:

処理オプション・テキストの変更と、処理オプション・テンプレートの変更との間で矛盾が生じる可能性があります。テンプレートの変更は、別のパッケージが構築されるまで有効になりませんが、テキストの変更はすぐに有効になります。

以下に、処理オプションを作成して実行する方法を示します。

1. 「テンプレート」と呼ばれるパラメータ・リストを作成し、処理オプションを作成する。
2. このテンプレートをアプリケーションに関連付け、イベント・ルールを作成して、アプリケーションがこれらの値を使用するようにする。
3. アプリケーションのバージョンを作成する。
4. 実行時の処理オプションの処理方法を指定する。

実行時には、アプリケーションをどのように設定しているかに応じて、次のイベントのうち 1 つが発生します。

- 処理オプションが表示され、ユーザーが処理オプション値を選択できるようにする。
- ユーザーがバージョンを選択できるバージョン・リストが表示される。
- 事前定義済みのオプション・セットを使用してアプリケーションが実行される。

〈processing option design aid（処理オプション設計ツール）〉プログラム(PODA)の処理中は、すべての処理オプション・テンプレート情報は、チェックインするまで POTEXT TAM に保管されています。チェックインした処理オプション・テンプレートは、POTEXT TAM から処理オプション・テキスト・テーブル(F98306)に移動します。処理オプションのデータ値は、バージョン・リスト・テーブル(F983051)に保管されます。バッチ・バージョンの場合、バージョン・リスト・テーブルには、一時変更（レポート一時変更、データ順序、データ選択またはロケーション一時変更）のスペックを指す識別子が付いています。

アプリケーションの各バージョンは、処理オプション値のリストに関連付けることができます。対話型アプリケーションやバッチ・アプリケーションの起動時に指定される処理オプションは、アプリケーションが実行時に使用する処理オプションになります。これにより、処理オプションは、バッチ・アプリケーションを投入してから実際にそれを実行するまでの間、ユーザーが変更できなくなります。処理オプションは、アプリケーションを特定の目的で実行する場合にも使用できます。このような処理オプションは、F983051 テーブルに永続的に保管されず、この特定の実行でのみ使用されます。

処理オプション・データ構造体の作成(テンプレート)

実行時にアプリケーションに渡されるデータ項目値を含む処理オプション・データ構造体(テンプレート)を作成することができます。テンプレートに対して行った変更は、テンプレートをチェックインするまで、すべてワークステーションに留まります。そのため、既存のテンプレートの現在のユーザーが、変更の影響をその場で受けずに済みます。変更をチェックインすると、次の JITI(ジャスト・イン・タイム・インストール)によって、変更内容が他のユーザーにレプリケートされます。

はじめる前に

- 処理オプション・データ構造体を作成します。『開発ツール』ガイドの「処理オプション・データ構造体の作成」を参照してください。

► 処理オプション・データ構造体(テンプレート)を定義するには

1. 〈Object Management Workbench(オブジェクト管理ワークベンチ)〉で、定義する処理オプション・データ構造体をチェックアウトします。
2. データ構造体がハイライトされていることを確認して、中央カラムの[Design(設計)]ボタンをクリックします。
3. 〈Processing Option Design(処理オプション設計)〉で、[Design Tools(設計ツール)]タブを選択して[Start the Processing Option Design Aid(処理オプション設計ツールの起動)]をクリックします。

   〈Processing Options Design〉ツールが起動します。このフォームの左側の領域に、処理オプションの外観が表示されます。

4. 〈Data Dictionary Browser〉で、処理オプションに必要なデータ項目を検索します。
5. 次のいずれかの方法を使用して、処理オプションに追加するデータ項目を選択します。
   - 〈Data Dictionary Browser〉で項目をダブルクリックする。タブの下、フォームの左側に項目が表示されます。
   - [Data Dictionary Browser]から構造体メンバの適切な位置にデータ項目をドラッグします。
6. 編集する項目をクリックします。
   - コントロールを囲むハッチングを使用して配置し直します。
   - テキストを選択して削除または上書きします。

   〈Processing Options Design〉は、タブの幅に合わせてデータ項目のサイズと位置を自動的に調整します。

7. テキスト・ボタン(A)を選択して、コメントを追加します。

8. フォームの左側の領域でオブジェクトを選択し、[View(表示)]メニューから[Properties(プロパティ)]を選択します。

   データ項目を選択した場合は、そのプロパティを表示し、必要に応じて項目名を変更できます。項目名は固有にする必要があります。

   [Help Override Data Item(ヘルプ一時変更データ項目)]タブをクリックすると、ヘルプの取得先の代替 szDict を追加できます。

9. 処理オプションを右クリックし、メニューから[Properties]を選択します。
10. 〈JDE.DataItem Properties(JDE.DataItem のプロパティ)〉で、[Help Override Data Item]タブをクリックし、次のフィールドに値を入力します。
    - Data Item Help Override Name(データ項目ヘルプ一時変更名)

    注:
    ヘルプの一時変更データ項目名を指定する場合は、『開発スタンダード:アプリケーション設計』ガイドに定義されている命名ガイドラインに従う必要があります。

11. [OK]をクリックします。
12. タブ・プロパティを表示するには、タブを選択し、[View]メニューから[Properties]を選択します。

    タブを右クリックし、ショートカット・メニューから[Current Tab Properties(現在のタブ・プロパティ)]を選択しても同じことができます。

    タブ項目を選択した場合は、タブのショート・ネームとロング・ネームを入力できます。

    タブのヘルプ・ファイル名を追加するには、[Help File Name(ヘルプ・ファイル名)]フィールドを使用します。

13. 新しいタブを追加するには、[File(ファイル)]メニューから[New Tab(新規タブ)]を選択します。

    既存のタブを右クリックし、ショートカット・メニューから[New Tab]を選択しても同じことができます。

参照

- テンプレート・オブジェクトの作成手順については、「処理オプション・データ構造体の作成」

J.D. Edwards 処理オプション命名規則

特別なビジネス・ケースを除き、J.D. Edwards 命名規則にできるだけ従ってください。命名規則に従うことで、プログラミングの一貫性が確保されます。

処理オプション・データ構造体

データ構造体の〈Object Librarian〉名は、最大 10 文字または 9 文字で（先頭に T を付けるかどうかによって異なります）、「Txxxxxyyyy」という形式で表記できます。各文字は次の意味を表します。

T = 処理オプション・データ構造体

xxxxxyyyy = アプリケーションやレポートのプログラム番号

たとえば、アプリケーション P0101 の処理オプション・データ構造体名は T0101 になります。

タブ・タイトル

処理オプションのタブ・タイトルを定義するときには、次のガイドラインに従ってください。

- タブ・タイトルには略語を使用しません。
- 将来使用の処理オプションは、現在使用できないことを示すために、"FUTURE"という語を入力します。タブ全体が使用できない場合は、タブの拡張記述の後に"FUTURE"と入力します。1 つの処理オプションが使用できない場合は、データ項目記述の後に"FUTURE"と入力します。
- 各タブがそれぞれ 1 つだけ存在し、複数のタブに分割されないことを確認します。たとえば、[プロセス 1]、[プロセス 2]の代わりに[プロセス]を使用します。
- P4310 などのアプリケーション名は、使用するバージョンの参照時にテキストに含めます。[Version（バージョン）]タブは、「Enter the version to be used for each program. If left blank, ZJDE0001 will be used.（各プログラムで使用するバージョンを入力してください。ブランクにすると、ZJDE0001 が使用されます。）」というコメント・ブロックから始めてください。
- アプリケーション固有のタブはなるべく使用を差し控え、他に適切なカテゴリが存在しないときに限って使用します。アプリケーション固有タブの名前は翻訳を考慮して、英字で 10 文字以内とします。
- 8 つの標準タブ・タイトルのうちのいずれかを使用します。ここに、拡張記述とそれぞれの処理オプションが加わります。
    - Display（表示）：特定のフィールドを表示するかどうか、または入力時にどのフォーマットでフォームを表示するかを指定します。
    - Defaults（デフォルト）：特定のフィールドにデフォルト値を割り当てます。
    - Edits：特定のフィールドに対してデータ検証を実行するかどうかを指定します。
    - Process（処理）：アプリケーションのプロセス・フローを制御します。

アプリケーション固有のタブ:

- Currency:通貨固有のオプションです。
- Categories:デフォルトのカテゴリ・コードを割り当てます。
- Print:レポートの出力を制御します。
- Versions:このアプリケーションから呼び出されるアプリケーションの実行バージョンを指定します。

コメント

- 処理オプションのコメントを入力するときには、次のガイドラインに従ってください。

注:
複数の処理オプションをグループ化するときは、処理オプションやコメントに番号を付けることができます。状況に合わせて選択してください。

- タブの全オプションに番号を付けてください。タブごとに 1 から始まる続き番号を使用します。
- 処理オプションを表すには、「Cutomer Master」などの名詞を使用します。必要なアクションは、その処理オプションの用語解説で定義されます。
- 処理オプションが必須オプションの場合は、処理オプションの末尾に「Required(必須)」という語を付けます。
- 複数の処理オプションが同じトピックに関係するときは、コメント・ブロックを使用します。コメント・ブロックは、処理オプションの論理グループのタイトルです。

データ項目

処理オプションのデータ項目を選択するときには、次のガイドラインに従ってください。

- 必要に応じて、データ項目の名前を説明調に変更します。
- データ項目要素の名前を変更するときには、イベント・ルール変数に対する命名規則に従ってフィールド要素の名前を設定し、szCategoryCode3_CT03 というようにエイリアスも含めてください。
- データ辞書の用語解説が適用できるときは、関連データ項目を使用します。用語解説は、処理オプションから表示できます。EV01 などの汎用ワーク・フィールドは使用しないでください。

処理オプションでの使用言語に関する考慮事項

処理オプション・テンプレートは、言語機能を組み込むために変更できます。

► テキスト翻訳用のテンプレートを変更するには

〈System Administration Tools（システム・アドミニストレーション・ツール）〉メニュー(GH9011)から[Processing Options Text Translation（処理オプション・テキストの翻訳）]を選択します。

1. 〈Work With PO Text Translations（PO テキスト翻訳の処理）〉で、次のフィールドに値を入力して[Find]をクリックします。
   - Template Name（テンプレート名）
   - To Language（翻訳先言語）

   〈Work with PO Text Translations（PO テキスト翻訳の処理）〉に、指定した処理オプション・テンプレートと言語に適合する処理オプション・テキストが表示されます。
2. テキスト・タイプを変更するローを選択して[Select]をクリックします。

   テキスト・タイプは、タブ、項目、コメントのいずれかです。
3. 〈PO Text Translation（PO テキスト翻訳）〉で、新規のテキストを入力します。

► 翻訳テキスト付きテンプレートを追加するには

言語に対応するアプリケーションの新しい処理オプション・テンプレートを追加するときは、以下のタスクを実行します。

1. アプリケーションを作成します。
2. 基本言語での処理オプション・テンプレートを作成します。
3. 言語テキストを追加します。

処理オプション・テンプレートの関連付け

実行時に処理オプションを使用可能にするには、処理オプション・テンプレートをアプリケーションに関連付ける必要があります。処理オプションは、デフォルト値の設定やフォーマットの制御、レポート区切りの制御、合計の制御、レポートにおけるデータ処理方法の制御などに使用することができます。処理オプション・テンプレートには、次の特徴があります。

- 独立したオブジェクトとして存在する。
- 複数のアプリケーションに関連付けることができる。

処理オプション・テンプレートを関連付けるときに、1 つ以上の処理オプションが一定のイベントで処理するように設計されている場合は、その処理オプションを有効にするイベント・ルール・ロジックを関連付ける必要があります。

► 処理オプション・テンプレートを関連付けるには

1. 〈Application Design Aid(アプリケーション設計ツール)〉で、[Form]メニューから[Design]を選択します。
2. [Application(アプリケーション)]から[Select Processing Options(処理オプションの選択)]を選択します。
3. 〈Select Processing Option Template(処理オプションテン・プレートの選択)〉で、使用する処理オプション・テンプレートを選択し、[OK]をクリックします。

注意:
アプリケーションのバージョンおよび関連付けられるイベント・ルールは、特定の処理オプション・テンプレートによりアプリケーションに渡されるデータ・ブロックによって異なります。そのテンプレートをアプリケーションから切り離したり、アプリケーションに異なるテンプレートを接続すると、アプリケーションが正常に動作しないことがあります。

処理オプション・テンプレートを変更するには、最初に既存のアプリケーション・バージョンをすべて削除する必要があります。次に、イベント・ルール用のすべてのアプリケーション・オブジェクトを検証し、テンプレートを変更したり、削除した後でも使用できることを確認してください。

トランザクション処理

トランザクションとは、一般的なタスクを完遂し、データの一貫性を維持するためにデータベースに対して実行される(1 つ以上の SQL ステートメントから構成された)作業の論理単位です。トランザクション・ステートメントは密接に関連する、相互依存のアクションを実行します。それぞれのステートメントはタスクの一部しか実行しませんが、すべてがタスクを完了させるために必要です。

トランザクション処理では、データベースで関連するデータが同時に追加または削除されるため、アプリケーションでデータの一貫性が保たれます。この処理では、コミット・コマンドが発行されるまでは、データはデータベースに書き込まれません。コミット・コマンドが発行されると、データはデータベースに永続的なデータとして書き込まれます。

たとえば、2 つのデータベース・テーブルを更新するデータベース操作をもつトランザクションの場合は、すべての更新が両方のテーブルに対して実行されるか、まったく更新されないかのどちらかになります。これにより、ビジネス・データの整合性が保証され、データの一貫性が確保されます。

トランザクション処理中には、一貫したデータベース・ビューが表示されます。他のユーザーによる変更がに表示されることはありません。

トランザクション処理では、トランザクションが次の状態にあることが確実にされます。

- アトミック – トランザクション全体に対するデータベースの変更が完了するか、または何も変更されません。
- 一貫性 – データベース変更は、データベースが整合性のとれている状態から、別の整合性のとれた状態に移行します。

- 独立性 – 同時に動作しているアプリケーションからのトランザクションは、相互に割り込みません。トランザクションによる更新は、トランザクションがコミットされるまで、同時に実行されている他のトランザクションに認識されません。
- 耐久性 – データベース操作全体がデータベースに永続的に書き込まれます。

コミットとロールバック

トランザクションの範囲は、トランザクションの開始と終了によって定義されます。トランザクションの終了は、トランザクションがコミットされるか、またはロールバックされると発生します。トランザクションのコミットもロールバックも発生しない場合、トランザクションはシステムの終了時にロールバックされます。

トランザクション処理では、コミットを使ってデータベース操作を制御します。コミットは、データベースに対するコマンドです。トランザクションは、自動でも手動でもコミットできます。自動コミットでは、データベースの変更が永続的な変更としてデータベースに書き込まれます（コミットされます）。手動コミットでは、データベースの変更は、コミットまたはロールバックが発生したときに限り、永続的な変更としてデータベースに書き込まれます。

コミット

コミットは、ステートメントによって実行されたオペレーションの結果を永続的に保管するための、データベースに対する明示的なコマンドです。このイベントは、トランザクションを正常に終了します。

2 フェーズ・コミット（手動コミット・モード）

2 フェーズ・コミットは、分散トランザクションを調整します。これは少なくとも 1 つの更新ステートメントが、同じトランザクションの 2 つの別個のデータ・ソースに対して実行されたときにのみ生じます。

ロールバック

ロールバックは、ステートメントによって実行されたオペレーションの結果を取り消すための、データベースに対する明示的なコマンドです。このイベントは、トランザクションが正常に終了しなかったことを示します。

トランザクション境界内の挿入、更新または削除で何らかの問題が発生すると、そのトランザクションのすべてのレコード操作がロールバックします。トランザクションが無事終了した場合は、コミットが実行され、レコードが他のプロセスで使用可能になります。

ネットワーク障害などが原因で重大な問題が発生した場合は、DBMS が自動ロールバックを実行します。同様に、ユーザーがフォーム上の[Cancel]をクリックすると、ロールバック・コマンドがシステム関数を通じて発行されます。

トランザクション処理について

J.D. Edwards ソフトウェア・トランザクションとは、任意の数のデータベース上で実行される（1 つ以上の SQL ステートメントから構成される）作業の論理的な単位です。単一ステートメント・トランザクションは 1 つのステートメントから構成され、複数ステートメント・トランザクションは複数のステートメントから構成されます。

複数のデータベース操作をグループ化するトランザクションを ERP アプリケーションの中で作成できます。これによって特定のコマンドを実行し、トランザクションで要求された更新を行うまで、データベース操作をバッファに蓄積するようデータベース管理システムに対して要求することができます。トランザクションの一部として組み込んでいないデータベース操作では、データベースがすぐに更新されます。

アプリケーションでトランザクション処理機能がオンに設定されている場合、更新されたレコードは、更新がコミットされるまで表示できません。そのトランザクションのプロセスだけが、トランザクションが完了するまでそのトランザクションのレコードにアクセスできます。

〈Application Design（アプリケーション設計）〉ツールでは、アプリケーションでトランザクション処理機能を有効にし、トランザクションを構成するデータベース操作を定義できます。トランザクションやアプリケーションの中には、有効にしないものもあります。それらは、データベースの構成に従って、適宜有効にしてください。

DB2 に常駐するテーブルに対するデータベース操作でトランザクション処理機能をオンに設定した場合は、これらのテーブルをジャーナリングする必要があります。ジャーナリングすると、追加の処理が必要になるため、パフォーマンスが低下する可能性があります。これに関して問題が発生した場合は、DB2 管理者に連絡してください。

トランザクション処理の一般的なメッセージやエラーは、jde.log または jdedebug.log に書き込まれます。

データの相互依存

データの相互依存性とは、トランザクションを完了させるためのデータ要素を指します。たとえば、伝票には、買掛金元帳テーブル(F0411)と取引明細テーブル(F0911)の両方にレコードがあります。これら 2 つのテーブル間にはデータの相互依存関係が存在しないため、データが一方のテーブルにしかないときは、トランザクションは完了しません。

トランザクション境界

データの相互依存は、トランザクション境界によって限定されます。トランザクション境界は、1 つのトランザクションを構成するすべてのデータ要素を包含しています。また、単一フォームのデータ要素だけを対象にすることもできます。トランザクションに他のフォームからのデータを含める場合は、トランザクション境界を拡張し、そのフォームのデータも対象にする必要があります。

トランザクション処理のシナリオ

トランザクションの一般的なフローは以下のとおりです。

1. アプリケーションが起動し、ランタイム・エンジンを呼び出す。
2. ランタイム・エンジンがトランザクションを初期化する。
3. ランタイム・エンジンがビューを開く。
4. ランタイム・エンジンがデータベース操作を実行する。
5. ランタイム・エンジンがデータベース操作をコミットする。

接続された 2 つのフォームを同じトランザクション境界に含むには、親フォームのトランザクション処理をアクティブ化し、第 2 フォームへのインターコネクトで[Include in parent(親に組込み)]を指定します。第 2 フォームのトランザクション処理をアクティブ化する必要はありません。インターコネクト・フォームでの設定は、呼び出されたフォームでの設定に優先するからです。

次の表は、2 つのフォームの関係と、各シナリオに存在する境界を示しています。トランザクション境界は、フォーム・インターコネクトとビジネス関数インターコネクトを通じて定義されます。次の例では、フォーム 1 の[OK]ボタンをクリックするとフォーム 2 が起動されます。トランザクション境界は TP のオン/オフを指定することで変更できます。次の表は、トランザクション境界の定義の仕方によって生じる状況を示しています。

シナリオ		TP オン	TP オフ	フォーム、ビジネス関数インターコネクト、テーブル I/O	コメント
A	フォーム 1		X		すべてのフォームで自動コミットが使用されます。
	フォーム 2		X		
B	フォーム 1		X	X	どちらのフォームでも手動コミットが使用されないため、〈Form Interconnect Properties(フォーム・インターコネクト・プロパティ)〉の[Include in Parent]フラグは無視されます。
	フォーム 2		X		すべてのフォームで自動コミットが使用されます。
C	フォーム 1	X			フォーム 1(親)では手動コミット・モードが使用され、フォーム 2(子)では自動コミット・モードが使用されます。
	フォーム 2		X		[Include in Parent]フラグがオフになっているため、トランザクション境界はフォーム 2(子)を含むように拡張されません。
D	フォーム 1	X		X	フォーム 2(子)のトランザクション処理フラグがオフの場合でも、[Include in Parent]フラグはオンになります。
	フォーム 2		X		トランザクション境界が、フォーム 2(子)を含むように拡張されます。

E	フォーム 1		X		[Include in Parent]フラグがオフになっているため、フォーム 1(親)とフォーム 2(子)は独立体として動作します。
	フォーム 2	X			フォーム 1 は自動コミット・モードで動作し、フォーム 2 は手動コミット・モードで動作します。
F	フォーム 1		X	X	標準動作です。フォーム 1(親)のトランザクション処理がオフになっているため、フォーム 2(子)の[Include in Parent]フラグがオンになっていても、トランザクション境界は子まで拡張されません。
	フォーム 2	X			フォーム 2(子)は手動コミット・モードで動作し、インターコネクトは無視されます。
G	フォーム 1	X			どちらのトランザクション処理もオンになります。
	フォーム 2	X			[Include in Parent]フラグがオフのため、各フォームがトランザクション境界となり、個別にコミットが発行されます。
H	フォーム 1	X		X	どちらのトランザクション処理もオンになります。ただし、[Include in Parent]フラグがオンのため、フォーム 2 のトランザクション処理は無視されます。
	フォーム 2	X			トランザクション境界には両方のフォームが含まれます。フォーム 2 はフォーム 1 の子です。

トランザクション処理とビジネス関数

アプリケーションやバッチ処理によって、主要なトランザクション境界が設定されます。あるビジネス関数が別のビジネス関数を呼び出す場合、呼出し先の関数でのデータベース操作も親アプリケーションの境界にグループ化されます。

マスター・ビジネス関数は固有の境界を定義しません。複数のマスター・ビジネス関数に 1 つの論理トランザクションを作成させることが必要になる場合があります。そのため、呼出し元のアプリケーションが境界を定義する必要があります。

アプリケーションがいくつかのビジネス関数を呼び出し、それらのビジネス関数をトランザクション境界に組み入れる必要がある場合は、トランザクション処理機能を有効にします。問題が発生した場合にビジネス関数に対するデータベース操作をロールバックする必要があるときは、ビジネス関数インターコネクトで[Include in Transaction]を指定します。

注:

トランザクションでビジネス関数を使用するときは、デッドロックが生じないよう注意してください。テーブル処理のロジックを 2 つの関数の間で分割した場合、トランザクションに一方の関数だけを組み入れ、もう一方の関数を組み入れないと、デッドロックが発生することがあります。情報用のレコードを選択し、さらに他のテーブルのデータを更新したり、挿入したりするビジネス関数を使用する場合は、ビジネス関数を分割することができます。

リモート・ビジネス関数でのトランザクション処理

トランザクション対応アプリケーションで、サーバーのビジネス関数がレコードを変更し、トランザクションの外部クライアントのビジネス関数がそのレコードへアクセスしようとすると、クライアント関数は、サーバーのビジネス関数がデータをコミットするまでロックされます。データがコミットされるまで、クライアント・アプリケーションは、サーバー側のビジネス関数によって行われたデータベースの変更にアクセスできません。サーバーのビジネス関数がコミットに失敗したり、ユーザーがサーバーのビジネス関数でトランザクションをキャンセルした場合、ビジネス関数のトランザクションはロールバックします。サーバーとクライアントはいずれもトランザクションにコミットしなければなりません。そうでないとトランザクションは、サーバーとクライアントでロールバックします。

トランザクション処理用システム関数

J.D. Edwards には、いくつかのトランザクション処理用システム関数が用意されています。状況によっては、追加のトランザクション処理機能のためのシステム関数を使用しなければならないこともあります。

たとえば、FormA および FormB という 2 つのフォームがあり、FormA のトランザクション処理が有効化されているシナリオを考えます。FormA は、Post OK Button is Clicked イベントの[Include in Parent]オプションがオンの状態で FormB を呼び出します。FormB は FormA のトランザクション境界を継承するため、ユーザーが FormB でのイベントを取り消すと次の処理が行われます。

- FormB の入力は書き込まれません。
- 制御が FormA に戻されます。
- FormA の入力が書き込まれてコミットされます。

このシナリオでは、システム関数 Rollback Transaction を使用すると、FormA の入力のコミットを防止できます。

バッチ・プロセスでトランザクション境界を定義するときには、次のシステム関数を使用できます。

- Begin Transaction – トランザクションの開始位置を定義する。
- Commit Transaction – トランザクションの終了位置を定義する。
- Rollback Transaction – トランザクションをロールバックする。

注:

特定のシステム関数については、Knowledge Garden の「Online APIs」を参照してください。

トランザクション処理のための操作

トランザクション処理は次のフォーム・タイプで使用できます。

- 修正/検査
- 見出し詳細
- 見出しなし詳細

トランザクション処理は、次のイベントが正常に処理されている間のみ有効です。

- OK Button Clicked（OK ボタンをクリック）
- OK Post Button Clicked（OK ボタンをクリック後）
- Add Record to DB – Before
- Add Record to DB – After
- Update Record to DB – Before
- Update Record to DB – After
- Add Grid Rec to DB – Before（グリッド・レコードを DB に追加 – 前）
- Add Grid Rec to DB – After（グリッド・レコードを DB に追加 – 後）
- All Grid Recs Added to DB
- Update Grid Rec to DB – Before（グリッド・レコードで DB を更新 – 前）
- Update Grid Rec to DB – After（グリッド・レコードで DB を更新 – 後）
- All Grid Recs Updated to DB
- Delete Grid Rec from DB – Before（DB からグリッド・レコードを削除 – 前）
- Delete Grid Rec from DB – After（DB からグリッド・レコードを削除 – 後）
- All Grid Recs Deleted from DB

これらのイベントの外で発生する処理は、トランザクション境界に含まれません。

フォームのトランザクション処理の定義

〈Form Design（フォーム設計）〉の〈Form Properties（フォーム・プロパティ）〉で［Transaction（トランザクション）］オプションを指定して、フォームに対するトランザクション処理を定義する必要があります。これは、フォームで処理されるすべてのデータがデータベースに同時にコミットされることを意味します。

トランザクションにフォームが 1 つ含まれている場合は、そのフォーム自体がトランザクション境界であるため、この定義のみが必要です。ただし、トランザクションに他のフォームからのデータが含まれている場合は、フォーム・インターコネクトを通じて境界を該当するフォームまで拡張します。

注：

ビジネス関数またはテーブル I/O を使用してトランザクション境界を拡張することもできます。

参照

- トランザクション境界については、『開発ツール』ガイドの「トランザクション境界の拡張」

▶ フォームのトランザクション処理を定義するには

1. 〈Form Design〉で、[Form Properties]にアクセスするフォームをダブルクリックします。
2. 次の[Style(スタイル)]オプションをクリックします。

レポートのトランザクション処理は、セクション・レベルで行われます。

フィールド記述

フィールド	記述
トランザクション	コミット・コマンドが発行されるまで、フィールド・データが待ち行列に保管されるようにします。これで、すべてのデータをいつでもテーブルに移動します。トランザクション境界が他のフォームを含む場合、ビジネス関数の親チェックボックスに組み込む相互接続をパスして形成する値 – パスする値をチェックします。

トランザクション境界の拡張

トランザクション境界を他のフォームまで拡張するには、該当するフォーム間で親/子の関係を設定します。境界を拡張するには、〈Event Rules Design(イベント・ルール設計)〉で、フォーム・インターコネクトを通じて[Transaction Processing(トランザクション処理)]フラグをオンにします。

はじめる前に

- 〈Form Design〉で、トランザクション境界の各フォームのフォーム・プロパティを定義して、トランザクション境界にトランザクション処理を組み込みます。『開発ツール』ガイドの「フォームのトランザクション処理の定義」を参照してください。

フォーム間でトランザクション境界を拡張する

親フォームで手動コミットを使用する場合は、接続先のフォームでも手動コミットを使用する必要があります。

▶ フォーム間でトランザクション境界を拡張するには

1. 〈Form Design Aid〉を開きます。
2. 対象となる親フォームを選択します。
3. [Form]メニューから[Menu/Toolbar Exits(メニュー/ツールバー・エグジット)]を選択します。
4. [OK]ローを選択して[Event Rules]ボタンをクリックします。
5. 〈Event Rules Design〉で、Button Clicked イベントを選択して[Form Interconnect(フォーム・インターコネクト)]ボタンをクリックします。
6. 〈Work with Applications(アプリケーションの処理)〉で、使用するアプリケーションを選択します。
7. 〈Work with Forms(フォームの処理)〉で、トランザクション境界に組み込むフォームを選択します。
8. 〈Work with Versions(バージョンの処理)〉で、使用するアプリケーションのバージョンを選択します。

9. 〈Form Interconnect(フォーム・インターコネクト)〉で、次の[Transaction Processing]オプションを選択して[OK]をクリックします。

    - Include in Transaction

ビジネス関数を使用したトランザクション境界の拡張

トランザクション境界には、ビジネス関数を組み込むことができます。親フォームが自動コミットを使用する場合は、トランザクション境界を拡張するビジネス関数も自動コミットを使用します。[Include in Transaction]がオンに設定されていないビジネス関数は、すべて自動コミットを使用します。非同期的に処理するビジネス関数もトランザクションに組み込むことができます。

▶ ビジネス関数を使用してトランザクション境界を拡張するには

1. 〈Form Design Aid〉を開きます。
2. 対象となる親フォームを選択します。
3. [Form]メニューから[Menu/Toolbar Exits]を選択します。
4. [OK]ローを選択して[Event Rules]ボタンをクリックします。
5. 〈Event Rules Design〉で Button Clicked イベントを選択し、[Business Functions(ビジネス関数)]ボタンをクリックします。
6. 〈Business Function Search(ビジネス関数の検索)〉から、トランザクション境界に組み込むビジネス関数を選択します。
7. 〈Business Function(ビジネス関数)〉で、次の[Transaction Processing]オプションを選択します。

    - Include in Transaction

[Asynchronous(非同期)]と[Include in Transaction]の両方がオンに設定されたビジネス関数が、トランザクション境界に組み込まれます。

テーブル I/O を使用したトランザクション境界の拡張

テーブル I/O では、トランザクション処理をオープン・テーブル操作でのみ使用できます。テーブルを開くときに、そのテーブルに対する操作が手動コミット(トランザクションの一部)か自動コミットになるかが決まります。[Include in Transaction]がオンに設定されていないオープン・テーブル操作は、自動コミットを使用します。

▶ テーブル I/O を使用してトランザクション境界を拡張するには

1. 〈Form Design Aid〉を開きます。
2. 対象となる親フォームを選択します。
3. [Form]メニューから[Menu/Toolbar Exits]を選択します。
4. [OK]ローを選択して[Event Rules]ボタンをクリックします。
5. 〈Event Rules Design〉で、Button Clicked イベントを選択し、[Table I/O(テーブル I/O)]ボタンをクリックします。
6. 〈Insert TableIO Operation(テーブル IO 操作の挿入)〉で、[Advanced Operations(上級操作)]の[Open(開く)]オプションを選択して[Next(次へ)]をクリックします。
7. 〈Data Source(データ・ソース)〉で、[Advanced Options(上級オプション)]をクリックします。

8. 〈Advanced Options〉で、[Include In Transaction]オプションを選択して[OK]をクリックします。
9. 〈Data Source〉で、[Finish(終了)]をクリックします。

   イベント・ルールに[Open]オペレーションが表示されます。

レポートに対するトランザクション処理の定義

J.D. Edwards ソフトウェアには、対話型トランザクション処理機能の他に、レポートとバッチ処理用のトランザクション処理機能も用意されています。トランザクション処理機能をバッチ処理で使用可能にするには、レポート・プロパティの[Advanced(上級)]タブをクリックして[Transaction Processing]を選択します。次に、Transaction Processing システム関数を使用して、トランザクション境界の開始と終了を定義します。トランザクション境界を拡張して、ビジネス関数またはテーブル I/O を組み込むこともできます。

注:

これらのシステム関数については、Knowledge Garden の「Online APIs」を参照してください。

トランザクション処理と Lock Manager のための jde.ini の設定

トランザクション処理を有効にするには、エンタープライズ・サーバーとワークステーションの jde.ini ファイルを修正する必要があります。トランザクション処理は、それぞれの J.D. Edwards ソフトウェア・ワークステーションでワークステーション jde.ini ファイルの設定を変更して有効にします。デプロイメント・サーバーでは、こうした変更をパッケージ・デプロイメントを通じてワークステーションに配信される常駐 jde.ini ファイルに対して行い、変更された jde.ini ファイルを含むパッケージを配布する必要があります。

並行リリース・サポートについて

リリース B73.3 以前では、トランザクション処理は、主に jde.ini の[TP MONITOR ENVIRONMENT (TP モニタ環境)]セクションの設定で制御されていました。このセクションの設定は、サーバーで 9 つ、ワークステーションで 7 つあります。

リリース B73.3 では、[TP MONITOR ENVIRONMENT]セクションは jde.ini ファイルから取り除かれ、[LOCK MANAGER(ロック・マネージャ)]セクションに置き換えられています。この新しいセクションには、サーバーとクライアントの両方の jde.ini ファイルで、それぞれ 3 つの設定があります。以前の[TP MONITOR ENVIRONMENT]セクションにあった設定は、廃止されるか、または内部デフォルト値が割り当てられたことによって除外されました。

[TP MONITOR ENVIRONMENT]セクションの設定がリリース B73.3 で除外された理由を以下に示します。

設 定	除外理由
Status	廃止。リリース B732.2 の時点で、TP モニタは、常に ON に設定されます。
LogPath	jde.ini ファイルからベース・ディレクトリが割り当てられます。
LogStatements	廃止。ステートメントは常に記録されます。
LogBufferSize	内部のデフォルト・バッファ・サイズ(1 MB)が使用されます。
DisplayServerErrorMsg	廃止。クライアントは、常にサーバーのエラー・メッセージを表示します。
ServerRetryInterval	内部のデフォルト間隔が使用されます。
RegistryCleanupInterval	内部のデフォルト間隔が使用されます。
RegistryRecordLifeSpan	内部のデフォルト・スパンが使用されます。
ServerTimeout	この値は JDENET 設定から提供されます。

移行期間中は、jde.ini ファイルの両方のセクションが同時にサポートされます。この移行期間には、次の 3 つのシナリオが考えられます。

- [LOCK MANAGER]セクションが存在しない。このシナリオでは、J.D. Edwards は、[TP MONITOR ENVIRONMENT]セクションの設定をチェックします。
- [LOCK MANAGER]と[TP MONITOR ENVIRONMENT]が共に存在する。このシナリオでは、J.D. Edwards は、[LOCK MANAGER]セクションの設定を使用します。
- 両方のセクションが共に存在しない。このシナリオでは、トランザクション処理を開始することができず、障害が発生します。

トランザクション処理ログについて

コミット・コーディネータは 2 つの段階で動作します。

第 1 段階

第 1 段階では、Log Manager に各データ・ソースのログを永続バックアップ・ストレージ用のハード・ディスクにフラッシュさせます。これらのログには、遂行されたすべてのデータベース操作が記録されています。

この第 1 段階では、他のデータ・ソースがコミットした後にデータ・ソースのいずれかがコミットに失敗しても、ログの内容を参照することで、データベースはいずれも整合性のとれた状態に確実に復帰できます。各データ・ソースですべてのログが正常にフラッシュされると、第 2 段階が開始されます。

第 2 段階

第 2 段階では、コーディネータは、各データ・ソースをそれぞれのトランザクションにコミットさせます。データ・ソースのいずれかがコミットに失敗すると、第 1 段階で生成されたログからコミット・ログ・レポートが生成されます（このレポートは、jde.ini の LOGPATH で指定されたディレクトリに書き込まれます。このディレクトリには、トランザクションの一部であったすべての SQL ステートメントのデータ・ソース別リストが保管されます）。このコミット・ログには、受け渡されたデータ・ソースと受け渡されなかったデータ・ソースに関する詳細も記録されます。

ログは、データベース管理者がデータ・ソースを手動で同期し、それらすべてを整合性のとれた状態にするときに役立ちます。コミット・ログ・レポートは、少なくとも 1 つのデータ・ソースがコミットに失敗した場合のみ生成されます。データ・ソースがすべて正常にコミットすると、コミット・ログ・レポートは生成されず、第 1 段階で生成されたログは、Log Manager によってすべて削除されます。

リリース B73.3 以降、ログ・ファイルは、jde.ini の［INSTALL（インストール）］セクションで指定されたディレクトリに置かれます。

トランザクション処理と Lock Manager のための jde.ini の設定

先に述べたように、移行期間中は、［TP MONITOR ENVIRONMENT］と［LOCK MANAGER］の両セクションがサポートされます。

B73.3 以前のリリースを使用する場合は、［TP MONITOR ENVIRONMENT］セクションの設定を入力します。B73.3 以上のリリースを使用する場合は、［LOCK MANAGER］セクションの設定を入力します。

注：

Lock Manager には、以下の設定が適用されます。

- Server
- RequestedService
- AvailableService

残りの設定は、トランザクション処理に関連します。

［TP MONITOR ENVIRONMENT］セクションの設定（B73.3 以前）

ここでは、サーバーとワークステーションの両方で jde.ini ファイルの［TP MONITOR ENVIRONMENT］セクションの設定を入力する方法を説明します。これらの設定は、J.D. Edwards ERP の B73.3 以前のリリースを使用する場合のみ有効です。

▶ サーバーの[TP MONITOR ENVIRONMENT]設定を入力するには

1. エンタープライズ・サーバーで、jde.ini ファイルを見つけます。
2. メモ帳などの ASCII エディタを使用して jde.ini ファイルを検討し、以下の設定が正しいことを確認します。

```
[TP MONITOR ENVIRONMENT]
Status=status value
LogPath=log path
LogStatements=log on/off value
LogBufferSize=log buffer value
RequestedService=service value
Server=server name
ServerTimeout=timeout value
AvailableService=service value
RegistryCleanupInterval=cleanup value
RegistryRecordLifeSpan=life span value
RegistryRecordLifeSpan=lifespan value
LogServices=service value
```

次の表に、jde.ini ファイルの変数を示します。

設 定	値
Status	トランザクション処理のオン/オフを指定します。通常、すべてのアプリケーションでトランザクション処理を使用しない場合や、テストなどでトランザクション処理を一時的に無効にする場合を除き、トランザクション処理モニタはオンに設定してください。有効値は ON と OFF です。B73.2 の時点で、トランザクション処理はオフに設定できなくなりました。そのため、この値はバージョン B73.2 以上では無視されます。
LogPath	この設定は、トランザクション・ログを配置するディレクトリを指定します。このパスは、jde.log と jdedebug.log の保管場所に一致しなければなりません。 たとえば UNIX マシンでは、このパスは次のようになります。 /u10/owdevel/tc283984/b73.2
LogStatements	トランザクション・モニタがトランザクションで実行された処理をすべて記録するかどうかを指定します。有効値は ON と OFF です。
LogBufferSize	操作ログを、それらがディスクにコピーされるまでメモリに保管しておくために確保されているメモリ容量（バイト数）を示します。この値は J.D. Edwards ソフトウェアの内部デフォルト値であり、変更できません。

RequestedService

この設定は、クライアントがサーバーに要求するサービスを指定します。有効な値は次のとおりです。

- TS：タイム・スタンプ・サービスを要求する。
- NONE：サービスを要求しない。

Server

Transaction Management Server(TMS)を管理するサーバーを指定します。たとえば、サーバー名を inteInta などとします。

ServerTimeout

すべてのネットワーク操作のタイムアウトを秒単位で示します。この値は、ネットワーク・トラフィックに基づいて調整できます。これは、ワークステーションの jde.ini ファイルで必須の値です。また、バッチ・ジョブをサーバーで実行する場合は、サーバーの jde.ini ファイルでも必要になります。この値は J.D. Edwards ソフトウェアの内部デフォルト値であり、変更できません。

AvailableService

この Transaction Management Server が提供するサービスを示します。Transaction Manager がワークステーションで初期化されると、この値を TMS に照会します。これを TM-TMS ハンドシェイキングといいます。この値がワークステーション jde.ini ファイルの値に一致する場合は、システムが動作している適切な時点で該当するサービスが呼び出されます。有効な値は次のとおりです。

- TS：Record Change Detector（タイムスタンプ・サービス）
- LM：ロック管理サービス
- ALL：TS と LM の両方のサービス
- NONE：サービスを提供しない。

RegistryCleanup Interval

有効期限が切れたレコードを TMS レコード・レジストリからすべて削除するまでの期間。この間隔は分単位で指定されます。この値は J.D. Edwards ソフトウェアの内部デフォルト値であり、変更できません。

RegistryRecordLife Span

レコードが TMS レコード・レジストリで存続できる最長期間。この期間を過ぎると、レコードは失効し、削除されます。このライフ・スパンは分単位で指定されます。この値は J.D. Edwards ソフトウェアの内部デフォルト値であり、変更できません。

s

この設定は、jde.log ファイルの補足的な設定で、TMS の Trace Log をオンにします。有効な値は次のとおりです。

- 1: 1：TMS のトレーシングをオンにします。
- 0: 0：TMS のトレーシングをオフにします。

この設定のデフォルト値は 0 です。TMS のトレーシングは、他のすべてのデバッグ法を使用し尽くした後にのみオンに設定してください。

参照

- ログについては、『開発ツール』ガイドの「トランザクション処理ログについて」

► ワークステーションの[TP MONITOR ENVIRONMENT]設定を入力するには

1. トランザクション処理をサーバーで有効にする手順を実行します。

   注:
   トランザクション処理は、ワークステーションで有効にする前にサーバーで有効にしてください。サーバーの jde.ini ファイルを設定する前に、ワークステーションの jde.ini ファイルを設定すると、まだサーバーが提供していないサービスを要求してしまうことがあります。その場合、エラーが生成されます。

2. パッケージ・インストールの一部としてワークステーションに送信される jde.ini ファイルを検索します。このファイルは、J.D. Edwards デプロイメント・サーバーの次のリリース共有パスにあります。

¥¥B*xxx*¥CLIENT¥MISC¥jde.ini

   xxx は、インストールされているリリース・レベル（B732 など）です。

3. メモ帳などの ASCII エディタを使用して jde.ini ファイルを表示し、以下の設定が正しいことを確認します。

[TP MONITOR ENVIRONMENT]
Status=status value
LogPath=log path
LogStatements=log on/off value
LogBufferSize=log buffer value
RequestedService=*service value*
Server=server name
ServerTimeout=timeout value

次の表に、jde.ini ファイルの変数を示します。

設 定	値
Status	トランザクション処理のオン/オフを指定します。通常、トランザクション処理モニタは、すべてのアプリケーションでトランザクション処理を使用しない場合や、テストなどでトランザクション処理を一時的に無効にする場合を除き、オンに設定してください。有効値は ON と OFF です。B73.2 の時点で、トランザクション処理はオフに設定できなくなりました。そのため、この値はバージョン B73.2 以上では無視されます。
LogPath	トランザクション・ログを配置するディレクトリを指定します。このパスは、jde.log と jdedebug.log の保管場所に一致しなければなりません。 たとえば UNIX マシンでは、このパスは次のようになります。 /u10/owdevel/tc283984/b73.2
LogStatements	トランザクションで実行された全操作のログをトランザクション・モニタに維持させるかどうかを指定します。有効な値は ON と OFF です。
LogBufferSize	操作ログを、それらがディスクにコピーされるまでメモリに保管しておくために確保されているメモリ容量（バイト数）を示します。この値は J.D. Edwards ソフトウェアの内部デフォルト値であり、変更できません。
RequestedService	クライアントがサーバーに要求するサービスを指定します。有効な値は次のとおりです。 • TS：タイム・スタンプ・サービスを要求する。 • NONE：サービスを要求しない。
Server	Transaction Management Server を管理するサーバーを指定します。たとえば、サーバー名を intelnta などとします。
ServerTimeout	すべてのネットワーク操作のタイムアウトを秒単位で示します。この値は、ネットワーク・トラフィックに基づいて調整できます。これは、ワークステーションの jde.ini ファイルで必須の値です。また、バッチ・ジョブをサーバーで実行する場合は、サーバーの jde.ini ファイルでも必要になります。この値は J.D. Edwards ソフトウェアの内部デフォルト値であり、変更できません。

このセクションの最後の 3 行は、レコード変更検出に関係し、ワークステーションでレコード変更データベース・ロック機能を実行する場合に設定する必要があります。

注：

パッケージを配布する代わりに、jde.ini ファイルをすべてのワークステーションに手動でコピーすることもできます。

参照

- ログについては、『開発ツール』ガイドの「トランザクション処理ログについて」

[LOCK MANAGER]セクションの設定(B73.3 以上)

ここでは、サーバーとワークステーションの両方の jde.ini ファイルで[LOCK MANAGER]セクションの設定を入力する方法を説明します。これらの設定は[TP MONITOR ENVIRONMENT]セクションの値を入力している場合でも使用されます。

▶ サーバーの[LOCK MANAGER]設定を入力するには

1. エンタープライズ・サーバーで、jde.ini ファイルを見つけます。
2. メモ帳などの ASCII エディタを使用して jde.ini ファイルを表示し、以下の設定が正しいことを確認します。

```
[LOCK MANAGER]
Server=server name
AvailableService=available server service
RequestedService=client service request
```

次の表に、各変数を示します。

設 定	値
Server	レコードを処理する際に使用するロック・マネージャ・サーバーの名前を指定します。たとえば、サーバー名を intelnta などとします。 バッチ・アプリケーションをワークステーションで実行する場合など、クライアントをサーバーとして使用する場合は、ワークステーション jde.ini ファイルの[LOCK MANAGER]セクションにある同じ設定に一致させる必要があります。
AvailableService	サーバーが提供するサービスを示します。有効な値は次のとおりです。 • TS:タイム・スタンプ・サービスを提供する。 • NONE:サービスを提供しない。 サーバーにのみ適用されます。
RequestedService	クライアントがサーバーに要求するサービスのタイプを示します。有効な値は次のとおりです。 • TS:タイム・スタンプ・サービスを要求する。 • NONE:サービスを要求しない。

注意:

トランザクション処理は、ワークステーションで有効にする前にサーバーで有効にしてください。サーバーの jde.ini ファイルを設定する前にワークステーションの jde.ini ファイルを設定すると、まだサーバーが提供していないサービスを要求してしまうことがあります。その場合、エラーが生成されます。

▶ ワークステーションの[LOCK MANAGER]設定を入力するには

1. ワークステーションの jde.ini ファイルを見つけます。
2. メモ帳などの ASCII エディタを使用して jde.ini ファイルを表示し、以下の設定が正しいことを確認します。

```
[LOCK MANAGER]

Server=server name

RequestedService=client service request
```

次の表に、各変数を示します。

設 定	値
Server	レコードを処理する際に使用するロック・マネージャ・サーバーの名前を指定します。たとえば、サーバー名を intelnta などとします。 バッチ・アプリケーションをワークステーションで実行する場合など、クライアントをサーバーとして使用する場合は、ワークステーション jde.ini ファイルの[LOCK MANAGER]セクションにある同じ設定に一致させる必要があります。
RequestedService	クライアントがサーバーに要求するサービスのタイプを示します。有効な値は次のとおりです。 • TS:タイム・スタンプ・サービスを要求する。 • NONE:サービスを要求しない。

レコード・ロック機能

J.D. Edwards ソフトウェアは、データ・ロック機能を導入していません。J.D. Edwards は、ベンダのデータベース管理システムが提供するロック機能に依存しています。これにより、機能の重複が抑えられ、パフォーマンスが向上します。

ベンダのデータベースは、必要に応じて自動的にロックしないことがあります。このような場合は、明示的な指示によってデータ・ロック機能をコントロールすることができます。たとえば、レコード・ロック機能を使用して、自動採番機能の整合性を確保することができます。

レコード・ロック機能について

次のいずれかの方法で J.D. Edwards ソフトウェアのデータをロックすることができます。

- オプティミスティック・ロック機能

  オプティミスティック・ロック機能(「レコード変更検出」とも呼ばれます)を使用すると、ユーザーがレコードを照会してからそのレコードを更新するまでの間にレコードが変更された場合、レコードの更新を防止できます。

- ペシミスティック・ロック機能

  ペシミスティック・ロック機能を使用すると、同一レコードの同時更新を防止できます。レコードは、更新される前にロックされます。

オプティミスティック・ロック機能

ワークステーションの jde.ini ファイルでは、「レコード変更検出」をオンに設定できます。このタイプのデータベース・ロック機能は、ユーザーが照会している最中に変更されたレコードを更新できなくします。レコードが変更された場合は、レコードを再度選択して変更しなければなりません。この機能は、ビジネス関数、テーブル I/O、およびイベント・ルール・ビジネス関数で使用できます。

たとえば、2 人のユーザーが〈Address Book〉アプリケーションで作業を行っているとします。この場合、「レコード変更検出」がどのように行われるかを次の表に示します。

時 刻	操作
10:00	ユーザーA が、住所録レコード 1001 を選択して点検する。
10:05	ユーザーB が、住所録レコード 1001 を選択して点検する。 現在、2 人のユーザーが住所録レコード 1001 を開いている。
10:10	ユーザーB が、住所録レコード 1001 のフィールドを更新して[OK]をクリックする。 住所録レコード 1001 が更新され、ユーザーB が入力した情報を反映する。
10:15	ユーザーA が、住所録レコード 1001 のフィールドを更新して[OK]をクリックする。 住所録レコード 1001 は更新されず、ユーザーA がレコードを表示している間に、そのレコードが変更されたことを伝えるメッセージが表示される。ユーザーA がレコードを変更するには、レコードを選択し直して、更新を実行する必要がある。

レコード変更の発生が検出されると、取り出した後にレコードが変更されたことを知らせるメッセージが表示されます。

ペシミスティック・ロック機能

ペシミスティック・ロック機能は、単に「レコード・ロック機能」とも呼ばれます。レコード・ロック機能を使用すると、複数のユーザーまたはアプリケーションが同じレコードを同時に更新するのを防ぐことができます。たとえば、ユーザーが自動採番を使用するトランザクションを入力するとします。この場合[OK]をクリックすると自動採番機能が適切な自動採番レコードを選択し、この番号がトランザクション・ファイルにまだ存在しないことを確認した上で、番号を増分して自動採番レコードを更新します。最初のプロセスがレコードの更新を完了する前に、別のプロセスが同じ自動採番レコードへのアクセスを試みると、自動採番機能は、レコードがロック解除されるまで待機してから 2 番目のプロセスを実行します。

J.D. Edwards ソフトウェアのレコード・ロック機能は、発行済みの JDEBase API を呼び出して実現されます。レコード・ロック機能を使用するときは、更新が完了するまでレコードがロックされるので、レコードの選択と更新に要する時間を考慮する必要があります。トランザクション処理機能は、特別なロック API を使用します。ロックされたレコードは、トランザクションの一部として含まれていることもあれば、含まれていないこともあります。レコード・ロック API は、トランザクションとその境界に依存しません。これらは、コミット・モードが手動/自動の場合も常にロックします。

トランザクション境界内で、レコード・ロック API（JDB_FetchForUpdate や JDB_UpdateCurrent など）によって更新されるレコードは、更新のために選択されてからコミットやロールバックが発生するまでロックされます。レコード・ロック API を使わずに更新されるトランザクション境界内のレコードは、更新されてからコミットやロールバックが発生するまでロックされます。ビジネス関数を使ってトランザクション処理を定義し、起動する場合も同じです。

トランザクション境界内でのペシミスティック・レコード・ロック機能の使用

レコード・ロック機能は、トランザクション処理機能との関連で使用しなければならないことがあります。たとえば、読取り操作から更新までの間レコードをロックする場合は、レコード・ロック機能を使用する必要があります。

ビジネス関数とペシミスティック・レコード・ロック機能

ビジネス関数でテーブルを更新する場合は、レコード・ロック機能をビジネス関数で使用しなければならないことがあります。更新されるテーブルは、別のユーザーやジョブによって使用される可能性があるためです。この場合、レコードをロックする時間をできるだけ短くする必要があります。更新のための選択やフェッチを、できるだけ更新の直前に行うようにしてください。

通貨

国際的に事業を展開している企業では、会計処理のニーズが拡大し、処理の複雑さが増大します。これは、異なる通貨で事業を行い、レポート作成と会計処理のさまざまな要件に従わなければならないためです。国際的な企業に必要な基本事項は次のとおりです。

- 外貨を国内通貨に換算する。
- レポート作成と比較のために、複数の通貨を 1 つの通貨に変換する。
- 操業する国で定められた規制を遵守する。
- 為替レートの変動に応じて通貨を再評価する。

通貨処理機能

J.D. Edwards ソフトウェアの通貨処理機能には、次の機能が組み込まれています。

- 通貨の取出しは、データベース・トリガーとテーブル・イベント・ルールを通じて実行される。
- 通貨取出しロジックは、ビジネス関数で処理される。
- システム API により、キャッシュ処理済みのテーブルにアクセスすることができる。

利　点

J.D. Edwards ソフトウェアでは、開発者が通貨の取出しを制御することができます。いわゆる通貨管理システムではなく、通貨自体を処理することによって柔軟性が高まり保守しやすくなっています。J.D. Edwards の通貨処理機能には次のような利点があります。

- 通貨テーブルを追加してもシステム・モジュールを変更する必要がない。新しいビジネス関数を追加するだけでよい。
- ビジネス・ロジックは、ビジネス・ロジックに関する知識があることを前提にするシステム・モジュールではなく、ビジネス関数に組み込まれる。

- テーブル・イベント・ルールの概念によって、通貨取出しのロジックをテーブル・オブジェクトのレベルで関連付けることができる。
- テーブル・イベント・ルールは、アプリケーション・イベントではなくテーブル・イベントによって起動することができる。
- 通貨のビジネス関数が関連付けられているテーブルを使用するアプリケーションは、いずれも同じロジックが適用されるため、個々のアプリケーションを変更する必要がない。
- ランタイム・エンジンには、ハードコード化されたロジックが埋め込まれていない。

通貨の処理

識別した金額がデータベースに書き込まれるか、またはデータベースから取り出されるとき、あるいは処理中の計算に使用されるときには、正しい小数点位置がきわめて重要になります。通貨は、Math_Numeric 属性の通貨フィールドにより小数点位置を調整されるため、通貨処理機能が必要です。通貨処理機能には、金額の換算と為替レートの変動による通貨の再評価が含まれます。

通貨処理を実現するには、次のステップを実行する必要があります。

- 通貨を設定します。
- 通貨情報を取り出すロジックを含むビジネス関数を作成します。通貨ビジネス関数は、通貨トリガーと呼ばれます。
- TER(テーブル・イベント・ルール)で、通貨トリガーを Currency Conversion イベントに関連付けます。
- 〈Event Rules Design〉を使用して、TER 関数を設計します。その後、イベント・ルールは、〈OCM(オブジェクト構成マネージャ)〉アプリケーションによって C 言語に変換され、コンパイルされて、連結 DLL になります。
- 必要に応じてアプリケーションを修正します。

JDB API は、Currency Conversion イベントの起動時に適切な TER 関数を呼び出します。

ビルド・トリガー(Build Trigger)オプションについて

[Build Triggers(ビルド・トリガー)]オプションによって、次のステップが実行されます。

- イベント・ルールが C 言語ソース・コードに変換されます。

  これにより、ファイル OBNM.c と OBNM.hxx が作成されます(OBNM はオブジェクト名です)。ソース・ファイルには、TER イベントごとに関数が 1 つずつ含まれます。

  たとえば買掛金元帳テーブル(F0411)を処理している場合、[Build Triggers]オプションによって F0411.c という名前の C ソース・メンバが作成されます。この C 言語コードを参照して、すべてのパラメータが正しく設定されているかどうかを確認できます。イベント・ルールから C 言語への変換時にエラーが発生した場合は、エラー・ログが生成されます。エラー・テーブル名は eF0411.log となります。

- 新しい関数がコンパイルされ、JDBTRIG.DLL に追加されます。この DLL は、TER 関数を含む連結 DLL です。

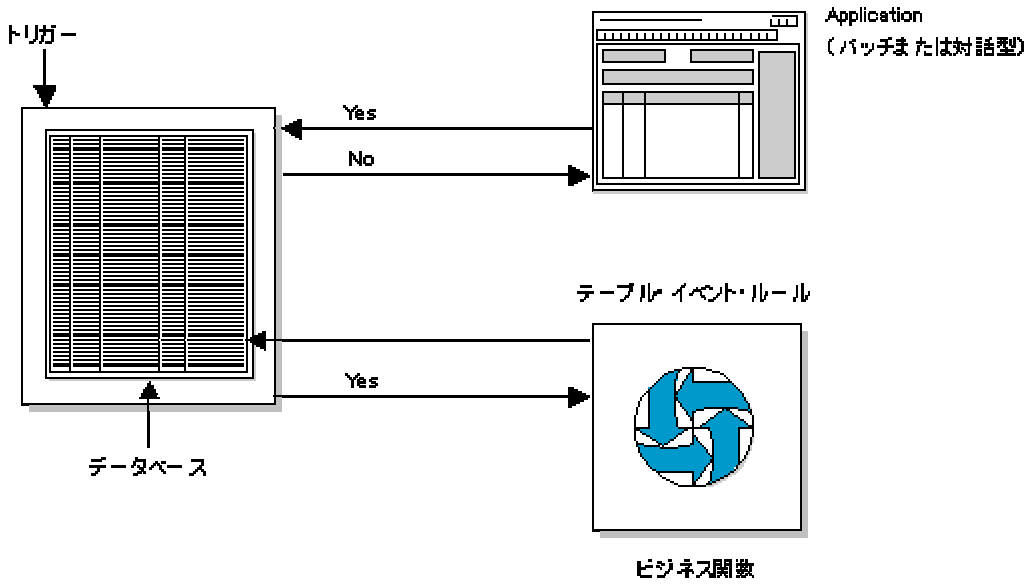
テーブル・イベント・ルールによる通貨処理の処理方法について

通貨処理が有効な場合は、Currency Conversion イベントが実行されます。

通貨のテーブル・トリガーは、レコードが取り出されてから、データベースに追加される前に実行されます。

次の図に、通貨換算プロセスを示します。

通貨換算



FETCH 時:	ADD/UPDATE 時:
1. アプリケーションがデータを要求する。	1. アプリケーションがデータを送信する。
2. 通貨処理はオンになっているか?	2. 通貨処理はオンになっているか?
3. Yes の場合は通貨トリガーを実行する。	3. Yes の場合は通貨トリガーを実行する。
4. 通貨トリガーが次の TER を呼び出す。TER が次の処理を実行する。	4. 通貨トリガーが次の TER を呼び出す。TER が次の処理を実行する。
• ビジネス関数を実行する。	• ビジネス関数を実行する。
• ビジネス・ロジックを実行する。	• ビジネス・ロジックを実行する。
• それに従ってデータを整理する。	• それに従ってデータを整理する。
5. データをデータベースに返してから、アプリケーションに返す。	5. データベースが更新される。

Math_Numeric 属性の通貨フィールド値をビジネス関数に渡すときには、それぞれのデータ構造体に通貨値が挿入される必要があります。通貨値を含む Math_Numeric ワーク・フィールドにも、正しい通貨情報が必要です。

システム関数の Copy Currency Info を使用すると、イベント・ルールでコントロール（ワーク・フィールドなど）に通貨情報をコピーすることができます。通貨トリガーは、アプリケーションのイベント・ルールから、または他のビジネス関数から呼び出せます。

通貨換算の設定

ビジネスに複数の通貨を使用する場合は、どの通貨換算方法を使用するかを指定する必要があります。

▶ 通貨換算を設定するには

〈Multi-Currency Setup（多通貨処理システム・セットアップ）〉メニュー(G1141)から〈Set Multi-Currency Option（多通貨オプションの設定）〉を選択します。

1. 〈System Setup（システム・セットアップ）〉で、[General Accounting Constants（一般会計固定情報）]をクリックします。
2. 〈General Accounting Constants〉で、次のフィールドに値を入力します。
   - 多通貨換算(Y/N/Z)

通貨換算フラグは、会社 00000 の CRYR フィールドの会社固定情報テーブル(F0010)に保管されます。

[Multi-Currency Conversion]オプションを N に設定すると、通貨換算は JDB とランタイム・エンジンでは行われません。

通貨依存のコントロールの表示

アプリケーションの設計時には、チェックボックスやラジオ・ボタンなどの通貨依存のコントロールを実行時に表示するか非表示にするかを指定できます。

▶ 通貨依存のコントロールを表示するには

1. 〈Form Design〉で、フォームに表示するコントロールをダブルクリックします。
2. [Control Options（コントロール・オプション）]をクリックします。
3. 通貨フィールドを表示する場合は、[No Display if Currency is Off（通貨がオフの場合、表示しない）]オプションがオフに設定されていることを確認します。

   このオプションがオンの場合、通貨依存のコントロールは表示されません。このオプションをオフにすると、通貨フィールドが表示されます。

通貨換算の変更を適用するために、現在の J.D. Edwards ソフトウェア・セッションを終了し、新規セッションを開始する必要があります。

通貨換算トリガーの作成

使用するテーブルに通貨フィールドが含まれている場合は、各カラムの小数点以下表示桁数を指定する必要があります。変換元のフィールドまたは変換先のフィールドが通貨フィールドで、通貨トリガーを作成していない場合は、値が計算に使用される際に問題が発生する可能性があります。通貨換算トリガーを作成しないと、システムではフィールドの小数点位置を判断できません。

▶ 通貨換算トリガーを作成するには

1. 〈Object Management Workbench(オブジェクト管理ワークベンチ)〉から、イベント・ルールを関連付けたいテーブルをチェックアウトします。
2. テーブルがハイライトされていることを確認して、中央カラムの[Design(設計)]ボタンをクリックします。
3. 〈Table Design〉で、[Design Tools]タブをクリックして[Start Table Trigger Design Aid]をクリックします。
4. 〈Event Rules Design〉で、Currency Conversion イベントを選択し、使用する通貨トリガーを関連付けます。
5. [Business Functions(ビジネス関数)]ボタンをクリックします。

   〈Business Function Search(ビジネス関数の検索)〉フォームが表示されます。

   QBE 行を使用すると、選択したビジネス関数を検索することができます。また[Category CUR(カテゴリ CUR)]や[System Code 11(システム・コード 11)]を使用すると、既存の通貨ビジネス関数を検索できます。ビジネス関数の用途、パラメータ、およびプログラミング要件を示す注記を読むには、[Attachments(添付)]ボタンをクリックします。
6. 処理するビジネス関数を選択して、[Select(選択)]をクリックします。
7. 〈Business Functions〉で、テーブル・カラムをビジネス関数構造体に追加して[OK]をクリックします。

   使用可能オブジェクトは、テーブル・カラムだけです。
8. 〈Event Rules Design〉で、[Save(保存)]を選択して[OK]をクリックします。
9. 〈Table Design〉で、[Table Operations(テーブル操作)]タブを選択して[Generate Table(テーブル生成)]をクリックします。
10. テーブルのデータ・ソースを選択して[OK]をクリックします。
11. 〈Table Design〉で、[Design Tools]タブを選択して[Build Table Triggers(テーブル・トリガーの構築)]をクリックします。

テーブル・イベント・ルールが作成されます。これで、新規作成または修正されたテーブル・イベント・ルール関数は、それぞれに対応するイベントがテーブルで発生するたびにデータベース API から呼び出されます。

ワンポイント・ヒント

J.D. Edwards では、多くの J.D. Edwards アプリケーションにワンポイント・ヒントを組み込んでいます。ワンポイント・ヒントは、ユーザーがアプリケーションを起動したり、フォームにアクセスするたびに表示される短い情報テキストです。これらのヒントを変更したり、独自のヒントを作成することができます。ワンポイント・ヒントは逐次表示されるので、ユーザーはそれらをいくつも参照することができます。ユーザーがヒントのフォームを閉じると、その閉じられたヒントの位置が記録され、ユーザーが再びオブジェクトを起動すると、次のヒントが表示されます。

ワンポイント・ヒントの処理

J.D. Edwards ソフトウェアのワンポイント・ヒントは、データ辞書項目の用語解説テキストです。1 つのヒントで 1 つのデータ辞書項目を作成できます。データ辞書用語解説は翻訳が可能なので、ワンポイント・ヒントはさまざまな言語で表示できます。

ヒントは、アプリケーション、フォーム、アプリケーション・バージョンのそれぞれに関連付けることができます。ヒントは、指定した順序で表示され、ユーザーのオプションを一時変更して、ヒント集に適用するワンポイント・ヒント機能をオフにできます。

ヒントとオブジェクトを関連付けた後は、ヒントの順序を変更したり、新しいヒントを追加したり、既存のヒントをヒント集から削除することができます。

はじめる前に

- ワンポイント・ヒントの 1 つ 1 つに対してデータ辞書項目を作成します。データ辞書項目の作成については、『開発ツール』ガイドの「データ項目の作成」を参照してください。

► ワンポイント・ヒント・ユーティリティを処理するには

〈System Administration Tools〉メニュー(GH9011)で、〈Tip of the Day（ワンポイント・ヒント）〉を選択します。

1. 〈Work With Tips of the Day（ワンポイント・ヒントの処理）〉で、[Find]をクリックして、ワンポイント・ヒントを追加するオブジェクトを表示します。

   検索範囲を絞るには、[Application Name（アプリケーション名）]フィールドにアプリケーションの名前を入力します。

   フォームの左側に、ツリー形式のファイル構造が表示されます。オブジェクトにはそれぞれ固有のフォルダがあります。ワンポイント・ヒントは、アプリケーション、フォーム、アプリケーション・バージョンそれぞれに関連付けることができます。フォルダ名には、アプリケーション、フォーム、アプリケーション・バージョンがこの順序で記されています。たとえば、RDA¥¥は、RDA アプリケーションだけを示します。P0911¥V0911I¥ZJDE0001 は、アプリケーション P0911 の ZJDE0001 バージョンのフォーム V0911I を示します。

2. オブジェクトに関連する特定のワンポイント・ヒントを表示するには、オブジェクトのフォルダを拡大表示します。

   このオブジェクトにワンポイント・ヒントとして関連付けられているデータ辞書項目がファイル構造に表示されます。

3. ワンポイント・ヒントのテキストを表示するには、ワンポイント・ヒントをクリックします。

   データ辞書項目に関連付けられている用語解説テキストがフォームの右側に表示されます。

4. ワンポイント・ヒントの順序を変更する、またはワンポイント・ヒントがすでに関連付けられているオブジェクトからワンポイント・ヒントを追加または削除するには、オブジェクトの下にあるいずれかのヒントをダブルクリックします。

   〈Tips of the Day Revisions（ワンポイント・ヒントの改訂）〉フォームが表示されます。

► ワンポイント・ヒントをオブジェクトに追加するには

ワンポイント・ヒントがまだ関連付けられていないアプリケーション、フォームまたはアプリケーション・バージョンに、ワンポイント・ヒントを追加することができます。既存のワンポイント・ヒント集にヒントを追加するには、〈Work With Tips of the Day〉フォームに表示されている既存のいずれかのヒントをダブルクリックして、〈Tips of the Day Revisions〉フォームにアクセスしてください。

1. 〈Work With Tips of the Day〉で、[Add]をクリックします。
2. 〈Tips of the Day Revisions〉で、以下のフィールドに入力します。
   - J.D. Edwards ERP ツール
   - すべてのユーザーにヒントを表示
3. グリッドで、アプリケーションに関連付ける個々のワンポイント・ヒントのデータ辞書項目を追加します。各ローで、以下のカラムに入力します。
   - ワン・ポイント・ヒント順序
   - データ項目

   注：
   ワンポイント・ヒント集に用語解説テキストが存在しない場合は、そのヒント集にデータ辞書項目を追加できません。

4. [OK]をクリックします。

フィールド記述

記述	用語解説
J.D. Edwards ERP ツール	ワンポイント・ヒントのセットを添付する ONEWORLD アプリケーション
フォーム名	フォームを識別するための ID
バージョン ID	アプリケーションやレポートの実行方法の指定に使用するユーザー定義のスペックです。バージョンを使用することで、ユーザー定義の処理オプション値やデータ選択、順序オプションなどをグループ化して保存します。対話型バージョンは（通常、タスク・レベルで）アプリケーションと関連付けられています。バッチバージョンはバッチ・プログラムまたはレポートと関連付けられています。バッチ・プログラムを実行する場合はバージョンを選択する必要があります。
記述	ツール・セットを添付するアプリケーションの記述 。
すべてのユーザーにヒントを表示	ユーザーがワンポイント・ヒントを非表示にできないようにします。このオプションは、現在のワンポイント・ヒントのセットにのみ適用されます。
ワン・ポイント・ヒント順序	ワンポイント・ヒントの表示順序。
データ項目	情報単位を識別および定義するコード。8 文字のアルファベット順によるコード。ブランクおよび次の特殊文字（%、&、+など）は使用できません。システム・コード 55 から 59 を使用する新しい項目を作成します。エイリアスは変更できません。

メッセージ処理

J.D. Edwards のメッセージ処理機能を使用すると、実効性に優れたユーザー・フレンドリーな様式で、適切な情報をエンドユーザーに配信することができます。メッセージ処理を使用するようにアプリケーションを設計するときには、ユーザーがタスクを実行する上でどのような情報が必要であるかを判断する必要があります。メッセージはリアルタイムで配信することができ、配信されたメッセージはステータス・バーに表示されます。ユーザーへの情報提供に使用する方法は、次のように状況に依存します。たとえば、次の処理を実行できます。

- レコードの入力時にエラーが発生した場合は、対話型エラー・メッセージを使用する。
- 情報を伝達して応答する必要がある場合は、〈Workflow Center（ワークフロー・センター）〉に送信される情報メッセージを使用する。
- 情報が緊急で、即刻注意する必要がある場合は、警告メッセージを使用する。
- レポートの処理中など、バッチ処理中にエラーが検出された場合は、バッチ・エラー・メッセージを使用する。

システム生成メッセージは、次の 3 つの要素で構成されています。

- メッセージ本体

  必要としているのは単純なメッセージか、またはテキスト置換メッセージか。すべてのテキスト置換部が使用可能かどうか。

- メッセージに適用されるロジック

  メッセージの送信に関して、イベント・ルール・ロジックなどの特定の基準が満たされているかどうか。

- メッセージ・タイプ

  たとえば、メッセージはユーザーによる処理を必要とするかどうか。メッセージの送信時に必須パラメータがすべて使用可能かどうか。

メッセージ・タイプ

主に次の 2 つのタイプのメッセージが用意されています。

- エラー/警告メッセージ
- 情報メッセージ

  情報メッセージは、ユーザーが現在のフォームから別のフォームに接続してエラーを訂正したり、情報を検討して適切な処置を下すことを可能にするアクション・メッセージにもなります。アクション・メッセージ機能を使用する場合は、フォームの接続用パラメータを、実行時に確実に使用できるようにしておく必要があります。

これらのメッセージ・タイプでは、簡易メッセージやテキスト置換メッセージを使用することができます。テキスト置換メッセージでは、可変的なテキスト置換を使用できます。置換値は、実行時にメッセージの変数部分に挿入されます。これにより、メッセージのインスタンスごとに固有のカスタマイズされたメッセージがユーザーに配信されます。

テキスト置換メッセージには、次の 2 つのタイプがあります。

- エラー・メッセージ（用語解説グループ E）
- ワークフロー・メッセージ（用語解説グループ Y）

どちらのタイプも同じように作成されます。

エラー・メッセージ

エラー・メッセージは、情報が正しく入力されていないこと（存在しない 3 桁の UDC コードを入力した場合など）、または技術的な問題があるために情報を取り出せないことを示すメッセージです。一般的に、エラー・メッセージはユーザーに処置方法を伝えます。

エラー・メッセージはデータ辞書に保管されています。主要なデータ辞書設計ツールは〈Data Dictionary Application〉プログラム(P7992002)です。

ワークフロー・メッセージ

内部/外部メッセージの送受信の他に、アクション・メッセージを受信できます。このメッセージは、システム・ワークフロー・プロセスにより自動的に送信されるタイプのメッセージです。

アクション・メッセージ

ワークフロー処理ではアクション・メッセージが生成されます。これは相手にアクションを起こさせるメッセージで、顧客レコード変更の承認または拒否の回答を求めるものなどがこれにあたります。稲妻の絵のアイコンが付いたメッセージはアクション・メッセージです。

アクション・メッセージを開くとショートカット・アイコンがあり、これをクリックするとアプリケーションへ直接リンクします。情報そのものではなくショートカットなので、メッセージの送信後に変更があった場合にも、クリックするたびにその時点で最新の情報をデータベースから検索することができます。

アクション・メッセージを特定の待ち行列に送信するように、ワークフロー・プロセスを設定できます。

レベル・メッセージ

レベル・メッセージは、エラー・メッセージをレポートの適切なレベル区切りに分類するために使用します。これらは、メッセージの容器のようなものです。レベル・メッセージは、「Here are your batch errors（バッチ・エラー）」や「Here are the document errors（ドキュメント・エラー）」などの条件として考えることができます。レベル・メッセージは、エラー・メッセージのあらゆる論理グループを区分するために使用します。先頭に「LM」の付いているメッセージがレベル・メッセージです。また、用語解説グループ Y に属しているレベル・メッセージは、ワークフロー・メッセージであることを示します。

情報メッセージ

レベル・メッセージ(LM)でなく、用語解説グループのメッセージ(Y)は、情報メッセージと見なされます。これらのメッセージは通常、ユーザーに直接関係のある情報を提供し、ユーザーのアクションを求めます。

エラー処理について

エラー処理を使用して、オペレーティング・システムによるプログラムのシャットダウンを防止します。

イベント・ドリブン・モデル

対話型メッセージは、特定のイベントが発生した場合に表示されます。これは、一定のイベントに基づいてエラーがクリア/設定/表示されることを意味します。あるイベントにエラーが発生したときエラーをクリアするには、そのイベントを再び動作させます。

たとえば、ユーザーがフィールドに不適切な値を入力したとします。この場合、Control is Exited イベントが動作したときにエラーが設定されます（このイベントにロジックを割り当てているか、またはハイ・レベル・トリガーによってフィールドが検証された場合）。ユーザーは、ビジュアルな指示（エラーとなったフィールドは一般的に赤で表示されます）を通じてそのエラーを認識します。加えて、エラーに関する情報がステータス・バーに表示されます。F8 ファンクション・キーを押すと、エラーの発生原因とその解決方法を参照することができます。または、複数のエラーが発生し、それを 1 つのフォームに順番に表示する必要がある場合は、F7 ファンクション・キーを押します。

エラーを解決するには、有効な値を入力して、そのフィールドから出ます。これにより、同じ Control is Exited イベントが再び動作します。このイベントはエラーをクリアし、不適切な値や無効な値が再び入力されたら再度エラーを設定します。入力された値が有効であれば、エラーは設定されません。

エラーはいつ設定されるか？

ユーザーがフォームのフィールドかグリッドのセルにデータを入力し、Tab キーを押してフィールドから移動すると、入力された値をランタイム・エンジンが照合します。検証処理が失敗した場合（入力された値が適切な UDC テーブルで見つからない場合や、無効な日付値が入力された場合など）は、適切なエラー・コードを含むエラー・メッセージが発行されフィールドがハイライトされるように、エラーを設定してください。エラー・メッセージとメッセージ件数は、ステータス・バーに表示されます。

強調表示すべきフィールドをアプリケーションはどのように認識するのか？

エラーが表示される時点で、ハンドルは問題のフィールドに置かれます。エラーがグリッド内部の場合には、そのグリッド・セルのローとカラムの座標が識別されます。システムは、アプリケーションのフォーム・ハンドルと共にこの情報を Error_Ctrl_Key 構造体に格納します。エラーが表示されると、コントロールの色が変わり、エラー・リンク・リストからエラー情報が表示されます。システム関数の Set Control Error か Set Grid Cell Error を使用すると、強調表示するフィールドまたはセルを指定できます。

イベント ID は、エラーの設定でどのように使用されるか？

J.D. Edwards ソフトウェア・アプリケーションは、イベント・ドリブン方式です。個々のキーストロークやマウス・クリックはイベントと見なされます。ユーザーがコントロールに値を入力し、タブでフィールドから出ると、Leave Control イベントか、Leave Column イベントがそのフィールドを検証します。エラーはいずれも Error_Event_Key 構造体のイベント ID を使用して設定されます。このイベント ID は、イベントが再処理されるときに、このイベントで設定されたエラーをクリアするために使用されます。たとえば、ユーザーが無効な住所録番号を入力し、タブでフィールドから移動すると、エラーが表示されます。このエラーは、Control Handle、Event ID および Grid Cell Coordinate（存在する場合）によって設定されます。このエラーをクリアするには、ユーザーが有効な住所録番号を入力した上で、フィールドから移動しなければなりません。このコントロールに関して他にエラーが見つからなければ、エラーがクリアされます。

エラー設定

エラー設定には、自動と手動があります。

自動エラー設定

ランタイム・エンジンは、次の場合にフィールドを検証して自動的にエラーを設定します。

- 項目がデータ辞書項目であり、それに対する編集ルール・トリガーがデータ辞書で定義されている場合。この場合、このデータ項目のフィールドはすべての J.D. Edwards ソフトウェア・アプリケーションで検証されます。データ辞書トリガーは、〈Form Design〉ツールの一時変更を使用して割り当てたり再定義することもできます。ただし一時変更されている場合は、その一時変更が定義されたアプリケーションにのみ有効になります。
- Control is Exited イベント、Column is Exited イベントまたは OK Button Processing イベントが発生したときに検証が実行されます。検証により、無効なデータや無効な日付などをチェックすることができます。

手動エラー設定

システム関数やビジネス関数 API を使用すると、他のエラー・メッセージを設定することができます。

システム関数

エラー・メッセージを設定するために、システム関数を使用することができます。この目的には次のシステム関数を使用できます。

- Set Edit Control Error
- Set Grid Cell Error

Set Edit Control Error を使用するのは、イベント・ルールを通じて［Form Control（フォーム・コントロール）］フィールドにエラーを設定する場合です。システム関数の引数として、次のパラメータを受け渡します。

- Control（たとえば、エラーとなったフィールド）
- Error Code（この場合のリテラルは、〈Data Dictionary Application〉プログラム（P7992002）または変数を通じて作成されるエラー・メッセージを意味します）

Set Grid Cell Error を使用するのは、イベント・ルールを通じて[Grid Control(グリッド・コントロール)]フィールドにエラーを設定する場合です。システム関数の引数として、次のパラメータを受け渡します。

- Grid(値を受け取りません)
- Row Number
- Column Number
- Error Code

システム関数は、テキスト置換メッセージであるデータ項目を使用できません。

ビジネス関数 API

エラー・メッセージを手作業で設定するには、ビジネス関数を使用します。テキストが置換されないエラー・メッセージを設定するには、次のビジネス関数 API のどちらかを使用します。

- jdeErrorSet (lpBhvrCom, lpErrInfo, idItem, lpzError)
- jdeSetGBRError (lpBhvrCom, lpVoid, (ID)0, "IError#I")

テキストが置換されないエラー・メッセージの一例を次に示します。

```
{
    jdeSetGBRError(lpBhvrCom,lpVoid,IDERRcRetainedEarningsLedger_3,"4524");
    idReturn = ER_ERROR;
}
```

リターン・コード完了メッセージは、次の操作に一切影響しません。これは情報伝達のために使います。

テキスト置換エラー・メッセージを設定するには、次のビジネス関数 API のどちらかを使用します。

- jdeErrorSet (lpBhvrCom, lpvoid, idItem, lpzError, lpDs)は、置換文を使ったビジネス関数を通じてエラーを設定します。これは、常に呼び出されます。
- jdeSetGBRErrorSubText

JdeSetGBRError と jdeSetGBRErrorSubText は同じように機能しますが、jdeSetGBRErrorSubText は、置換用の情報を保管するデータ構造体のアドレスを受け渡す追加のパラメータをもっています。

テキスト置換エラー・メッセージの設定例を次に示します。

```
/*****************************************************************************
* Declare structures
*****************************************************************************/
DSDE0018    dsDE0018;.
.
.
/*****************************************************************************
* Main Processing
```

```
*****************************************************************************/
      strncpy(dsDE0018.szAccountID, (const char *)(lpDS->szAccountID),
                  sizeof(dsDE0018.szAccountID));
.
.
.
if (idReturn != ER_SUCCESS)
     {
                  jdeSetGBRErrorSubText(lpBhvrCom, lpVoid,
                  IDERRszAccountID_1, "044E", &dsDE0018);
                  JDB_CloseTable(hRequestF0901);
                  JDB_FreeBhvr(hUser);
                  return ER_ERROR;
     }
```

この例での DSDE0018 は、エラー・メッセージ用のデータ構造体で、jdeapper.h で宣言されています。jdeapper.h では、J.D. Edwards によって使用されるテキスト置換エラー・メッセージ用のすべてのデータ構造体が宣言されています。J.D. Edwards 以外の置換エラー・メッセージは、それぞれ固有のグローバル宣言ヘッダー・ファイルか、構造体を使用する関数のヘッダー・ファイルで定義する必要があります。この例の IDERRszAccountID_1 のコード行には、置換する値が指示されています。

ビジネス関数は、呼び出されたときに問題のフィールドが無効な場合にエラーを設定できる検証ルーチン(jdeddValidation)を呼び出すことができます。

テキスト置換エラー・メッセージを C 言語ビジネス関数で使用するには、カスタム・ビジネス関数の共通のグローバル・ヘッダー・ファイルでデータ構造体のインスタンスを作成します。エラー・データ構造体が存在する場合は、jdeapperr.h に存在します。

エラーのリセット

システムは、[OK]ボタンか、[Find]ボタンでエラーをリセットできます。この処理は、手動エラーと自動エラーの両方で発生します。

OK ボタンでのエラーのリセット

システムは、見出し詳細フォームか見出しなし詳細フォームで[OK]ボタンがクリックされると、エラーをリセットします。次のアクションが発生します。

1. Button Clicked イベントで Error_Event_Key 構造体を設定し、コントロール・ハンドルが[OK]ボタンになります。
2. OK Button Clicked でイベント・エラーをクリアします。
3. 次のイベントによってエラーをクリアします。
   - Button Clicked Processing Done
   - Add Record to DB – Before

- Add Record to DB – After
- Update Record to DB – Before
- Update Record to DB – After
- All Grid Recs Added to DB
- All Grid Recs Updated to DB
- Delete Record from DB – Before
- Delete Record from DB – After
- All Grid Recs Deleted from DB
- Delete Grid Rec Verify – Before
- Delete Grid Rec Verify – After
- Row is Exited

4. エラーが依然として発生する場合は、処理を中止します。
5. Button Clicked のイベント・ルールを実行します。
6. ValidateAllData – これには、フォーム・コントロールが含まれます。
7. エラーが発生した場合は、処理を中止します。
8. グリッドから取り除かれたローをすべて削除します。
9. Add/Update to Database Before/After のイベント・ルールを実行します。
10. グリッドを変更します（未処理のグリッド・ローを検証します）。
11. エラーが発生した場合は、処理を中止します。
12. After Button Clicked のイベント・ルールを実行します。
13. エラーが発生した場合は、処理を中止します。
14. エラーが発生しなかった場合は、画面をクリアして終了します。

注：

[OK]ボタンの Button Clicked イベントでは、グリッド・コントロール・フィールドのエラーの検証と設定を行わないでください。Button Clicked イベントは、グリッド・ローを処理しません。

Find ボタンでのエラーのリセット

システムは[Find]ボタンでエラーをリセットします。その際、次の手順を実行してください。

1. Button Clicked イベントで Error_Event_Key 構造体を設定します。
2. OK/Select Button Clicked でイベント・エラーをクリアします。
3. 次のイベントによってエラーをクリアします。
   - Button Clicked
   - Last grid record has been read.

- Form Record is Fetched
- Grid Record is Fetched
- Confirm Delete – Before
- Confirm Delete – After

4. Button Click のイベント・ルールを実行します。

C 言語ビジネス関数による複数レベルのエラー・メッセージ処理

ビジネス関数が別のビジネス関数を呼び出し、その別のビジネス関数からエラーが発行されたときには、マッピング・キー配列を設定する必要があります。そうしないと、エラーが誤って設定されます。

jdeCallObject(ビジネス関数の中から別のビジネス関数を呼び出すための標準 API)を別のビジネス関数を呼び出すために使用したときには、6 番目のパラメータがエラー・マッピング用になります。ビジネス関数はそれぞれ固有のヘッダーと定義を持っています。そのため、第 2 レベルのビジネス関数を呼び出す理由を認識しておく必要があります。たとえば、第 2 レベルのビジネス関数を呼び出して会社番号を検証するとします。その場合、最初のビジネス関数のヘッダーで会社番号 ID を定義し、第 2 ビジネス関数でその会社番号 ID を確認できるようにしておく必要があります。

▶ 複数レベルのエラー・メッセージ処理を作成するには

1. 呼び出されるビジネス関数のヘッダー・ファイルを開き、マッピング・フィールドの最大数を指定します。

呼出し関数のヘッダー・ファイル	呼出される関数のヘッダー・ファイル
#define IDERRszComputerid_1 1L	#define IDERRcHeader_1 1L
#define IDERRszCompany_2 2L	#define IDERRcEvent_2 2L
#define IDERRszDate_3 3L	#define IDERRDetail_3 3L
#define IDERRszValue_4 4L	#define IDERRCompany_4 4L
#define IDEERszPoint_5 5L	#define IDERRDateOpen_5 5L

2. 呼出し側ビジネス関数の C 言語ファイルでエラー・マップ・セクションを作成します。

   アクティブなエラー・メッセージの次の例では、1 つのフィールド(IDERRCompany_4)をマップする必要があります。

```
/**************************************************
* Business Function: B1234
*
*
* Parameters:
*              LPBHVRCOM
```

```
*               LPVOID
*               LPDS1234
*************************************************/
/*************************************************
* Error Mapping Section
*
* Map to function "ValidateCompanyNumber"
* #define N1234 1
*************************************************/
```

数字の「1234」は、呼び出されるビジネス関数のデータ構造体番号です。数字の「1」は、マッピングされるフィールドの番号です。

各#define ステートメントの前に、この番号が使用されるビジネス関数名を示すコメントを設定します。

変数セクションで、エラーを返す必要がある各関数の cm_xxx マップ配列を作成します。

次に例を示します。

```
/*************************************************
* Variable declarations
*************************************************/
CALLMAP                 cm_1234[N1234]={{IDERRCompany_2,IDERRCompany_4}};
```

この配列は、IDERRCompany_2 から IDERRCompany_4 にマッピングされます。

jdeCallObject の関数呼出しは次のとおりです。

```
idReturn = jdeCallObjdect("ValidateCompanyNumber", ValidateCompanyNumber, lpBhvrCom,lpVoid, &ds1234,
cm_1234, N1234, (char *)NULL, (char *)NULL, (int)0);
```

ビジネス関数は、[ID]フィールドにエラーを設定します。

エラー・メッセージの処理

J.D. Edwards ソフトウェアは、所定のイベントに基づいて自動的にメッセージを表示します。たとえば、無効な値がフィールドに入力されると、常にエラー・メッセージが表示されます。

J.D. Edwards ソフトウェアはデータ辞書の用語解説を使用してメッセージを表示します。このメッセージはデータ項目の用語解説部に含まれています。データ辞書の既存のフレームワークを使用すると、メッセージ内容を別に作成せずに済みます。

既存のエラー・メッセージの検索

メッセージは、新規に作成する代わりにできるだけ既存のものを使用してください。既存のメッセージを検索するには、〈Work with Glossary Items（用語解説項目の処理）〉を使用します。目的のエラー・メッセージに類似した用語解説グループのタイプと記述を使用することで、検索範囲を狭めることができます。

▶ 既存のエラー・メッセージを検索するには

〈Data Dictionary Design（データ辞書設計）〉メニュー（GH951）で、〈Work With Data Dictionary Items〉を選択します。

1. 〈Work With Data Items〉で、[Form]メニューから[Glossary Data Item（用語解説データ項目）]を選択します。
2. 〈Work with Glossary Items〉で、見出し領域のいずれかのフィールドに値を入力して検索対象を特定のエラー・メッセージに限定し、[Find]をクリックします。
3. 使用するメッセージを選択して[Select]をクリックします。
4. [Item Glossary]タブをクリックし、エラー・メッセージのテキストを確認します。

簡易エラー・メッセージの作成

簡易エラー・メッセージ（Static メッセージ）は、たとえば、「Enter a valid date（有効な日付を入力してください）」というようなリテラル・テキストです。簡易エラー・メッセージでは、テキスト置換を行いません。

対話型メッセージ・データ項目の追加

新規のデータ項目を作成するか、または既存の項目を選択して、メッセージを表示できます。エラー・メッセージに機能を付与する前に、イベント・ルール・ロジックを使用して、それをフォーム・コントロールに接続する必要があります。

► 対話型メッセージ・データ項目を追加するには

1. 〈Work with Glossary Items〉で、[Add]ボタンをクリックします。
2. 〈Glossary Items〉で、[Item Specifications]タブをクリックして次のフィールドに入力します。
   - 用語解説グループ
   - システム・コード
   - 記述
   - エラー・レベル
3. [OK]をクリックします。

テキスト置換エラー・メッセージの作成

テキスト置換エラー・メッセージは、可変テキストをエラー・メッセージの用語解説テキストに直接置き換えることができるエラー・メッセージです。これにより、同じ基本メッセージを特定の変数値と共に表示することができます。

たとえば、ユーザーがフィールドに無効な検索タイプを入力したとします。この場合、エラー・メッセージの「Search type xx is not contained in the 00 ST table（検索タイプ xx は、00 ST テーブルにありません）」というエラー・メッセージが表示されます。メッセージの変数 xx は、ユーザーが入力した検索タイプで置換されます。

テキスト置換エラー・メッセージをバッチ処理で使用する場合は、データ辞書データ構造体に値を渡すためのビジネス関数を作成する必要があります。（ユーザー・エラーを設定する）イベント・ルール・エンジンは、テキスト置換エラー・メッセージに値を渡さないからです。

対話型メッセージ・データ項目の追加

新規のデータ項目を作成するか、または既存の項目を選択して、メッセージを表示できます。エラー・メッセージに機能を付与する前に、イベント・ルール・ロジックを使用してフォーム・コントロールに接続する必要があります。

この例では、エラーID 018A に対して、データ辞書でエラー・メッセージを作成します。

► 対話型メッセージ・データ項目を追加するには

1. 〈Work With Data Items〉で、[Form]メニューから[Glossary Data Item]を選択します。
2. 〈Work with Glossary Items〉で、[Add]ボタンをクリックします。
3. 〈Glossary Items〉で、[Item Specifications]タブをクリックして次のフィールドに入力します。
   - 用語解説グループ
   - システム・コード

- 記述
- エラー・レベル

記述を入力するときは、各サブセット変数の先頭に「&」を付けます。この例では、短い記述を使用し、「&2 not found in User Defined Code &3 &4」としています。

次に、実行時に表示される用語解説テキストを定義します。

ランタイム・メッセージの用語解説テキストの定義

対話型メッセージ・データ項目を追加した後、その項目の用語解説を追加する必要があります。

▶ ランタイム・メッセージの用語解説テキストを定義するには

1. 〈Work with Glossary Items〉で、[Row]メニューから[Glossary]を選択します。
2. メッセージのテキストを入力します。

   「&x」を使用して、テキストに変数を割り当てます。これらの変数は、データ構造体のメンバに対応します。

   データ構造体の順序が&x 構造体のインデックスとなります。構造体に 4 つのメンバがある場合は、最初のメンバに&1、2 番目のメンバに&2 を割り当てます。次に例を示します。

   Description = &1

   UDC Value = &2

   System = &3

   Type = &4

3. テキストが完成したら、[OK]をクリックします。

注:
用語解説テキストの最後にブランクなしでピリオドを配置し、繰り返しの問題が生じないようにします。

データ構造体テンプレートのメッセージへの関連付け

テキスト置換エラー・メッセージは各々がデータ構造体に対応します。データ構造体が存在しなければ、それを作成する必要があります。メッセージのメンバは、データ構造体のメンバに一致させてください。

参照

- データ構造体の追加については、『開発ツール』ガイドの「データ構造体」

► データ構造体テンプレートをメッセージに関連付けるには

1. 〈Glossary Item〉で、[Data Structure Template(データ構造体テンプレート)]タブをクリックします。
2. 既存の構造体を検索するには、[Text Substitution(テキスト置換)]をクリックして[Browse]ボタンをクリックします。
3. 〈Individual Object Search & Select(個別オブジェクトの検索/選択)〉で、データ構造体を選択して[OK]をクリックします。

   対話型メッセージに合った構造体が存在しない場合は、作成する必要があります。

対話型メッセージ・データ項目の関連付け

対話型エラー・メッセージを実行時に表示させるには、システム関数を関連付ける必要があります。

► メッセージ・データ項目を関連付けるには

1. 〈Event Rules Design(イベント・ルール設計)〉で、システム関数ボタンをクリックします。
2. 使用するシステム関数を選択します。
3. 使用可能なオブジェクトを使って、[Parameters(パラメータ)]に制御値を設定します。
4. [Error Code Velue in Parameters(パラメータのエラー・コード値)]で、<Literal>を選択します。

   リテラル値で表示するエラー・メッセージを含んでいるデータ辞書項目の番号を使用します。

Send Message システム関数の処理

Send Message API は、次のテーブルに関連付けられています。

- F01131 – PPAT メッセージコントロール・テーブル
- F01131M – JDEM マルチ・レベル・メッセージ・テーブル
- F01133 – PPAT メッセージ詳細テーブル
- F00165 – メディア・オブジェクト・テーブル

PPAT メッセージ制御テーブルと PPAT メッセージ詳細テーブル(F01133)は、メッセージ配信用の〈Message Center(メッセージ・センター)〉アプリケーションによって使用されます。これらのテーブルには、メッセージの受信者、メッセージの現在状況、メッセージの送信者、メッセージの送信日、メッセージが属するメールボックス等のメール配信に関係するすべての情報が格納されています。またメディア・オブジェクト・テーブル(F00165)には、メッセージの件名と本文が格納されます。

システム関数 Send Message は、次の 3 つの要素から構成されています。

- アドレス指定と送信先
- メッセージ(該当する場合は置換テキストも含む)
- アクション・メッセージ(メッセージから別のプログラムを呼び出す)

システム関数のパラメータとそれぞれの内容を次の表に示します。

パラメータ	説 明	標準値	必須/任意
Address Number	メッセージ受信者の住所録番号	住所録番号 – データ辞書項目 AN8	必須
Mailbox	メッセージ配信先のメールボックス。たとえば、「Personal in Basket」	ユーザー定義コード(02/MB)に登録されているメールボックスに関連する 2 文字のフィールド	必須
*Subject	メッセージの件名	メッセージの簡潔な記述。メッセージ・テンプレートを使用する場合はブランクを使用します。	任意 メッセージ・テンプレートを使用する場合は[None]を選択します。
*Text	メッセージの本文	送信されるメッセージ。メッセージ・テンプレートを使用する場合はブランクを使用します。	任意 メッセージ・テンプレートを使用する場合は[None(なし)]を選択します。
Active	送信するメッセージと他のフォームの間の接続の有無を示す値	この項目を使用すると、フォーム・インターコネクト・プロセスでプロンプトが表示されます。	任意
DDMessage	データ辞書からのメッセージ・テンプレート	用語解説をメッセージで使用できる有効なデータ辞書項目	任意
MessageKey	送信メッセージのキー値となるリターン・パラメータ	データ辞書項目(たとえば、SERK)	任意

[Subject(件名)]フィールドと[Text(テキスト)]フィールドを[DDMessage(DD メッセージ)]フィールドと共に使用すると、DDMessage(メッセージ・テンプレート)は、[Subject]フィールドと[Text]フィールドに連結されます。

▶ Send Message システム関数を使用するには

1. 〈Object Management Workbench(オブジェクト管理ワークベンチ)〉で、処理するテーブルをチェックアウトします。
2. テーブルがハイライトされていることを確認して、中央カラムの[Design(設計)]ボタンをクリックします。
3. 〈Table Design(テーブル設計)〉で、[Design Tools(設計ツール)]タブをクリックし、[Start Table Trigger Design Aid(テーブル・トリガー設計ツールの起動)]をクリックします。
4. [System Functions]をクリックし、[Function Selection(関数の選択)]タブを開きます。
5. [Function Selection]領域で、[Messaging(メッセージ処理)]を選んで[Send Message(メッセージの送信)]を選択します。

6. [Parameters]で、メッセージの受信者を[Recipient(受信者)]フィールドで指定します。

   このフィールドでは通常、住所録番号を使用しますが、電子メール・アドレスを使用することもできます。

7. メールボックスを指定します。

   メールボックスでは、送信されるメッセージのタイプを指定します。メールボックスは、ユーザー定義コード・テーブル(02/MB)に保管されています。ユーザー定義コード・テーブル(02/MB)からのメールボックスは、[Mailbox(メールボックス)]フィールドに焦点を合わせると表示されます。

8. 「Active」という名前のデータ項目をクリックし、[Define Active Message(アクティブ・メッセージの定義)]を選択して、フォーム・インターコネクト・プロセスを開始します。

   メッセージのアクティブ・メッセージ部は、イベント・ルールによるフォーム・インターコネクトと同じように機能します。

アクティブ・メッセージの定義

次の手順に従ってアクティブ・メッセージを定義します。

► アクティブ・メッセージを定義するには

1. [Define Active Message]をダブルクリックします。

   〈Work with Applications(アプリケーションの処理)〉が表示されます。

2. 処理するアプリケーションを選択します。
3. 処理するフォームを選択します。
4. 必要に応じて、処理するバージョンを選択します。

   処理オプションがフォームに関連付けられている場合は、〈Processing Options(処理オプション)〉フォームが表示されます。

   〈Form Interconnections(フォーム・インターコネクト)〉フォームが表示されます。ここで、受け渡される値をすべて割り当てることができます。

5. 受け渡す値を指定して、[OK]をクリックします。
6. 〈System Functions(システム関数)〉で、[Define Message(メッセージの定義)]を選択します。

   プロンプトに従って、メッセージを選択します。

7. [Dictionary Item(辞書項目)]フィールドに適切なエイリアスを入力し、[Find]をクリックします。

   置換個所には、&1、&2 などの番号が設定されます。パラメータ番号は置換番号に一致します。置換変数は、メッセージの[Description(記述)]と[Glossary]用語解説の両方に表示されます。

8. 適切なオブジェクトを置換パラメータにマッピングすると、メッセージが完成します。

バッチ・エラー・メッセージ

バッチ・エラー・メッセージ・システムは、バッチ・プログラムの処理中にエラーを確認できる一貫したインターフェイスを提供します。バッチ・プログラムでジョブの正否に関する全メッセージの処理が完了すると、ユーザーにメッセージが送信されます。メッセージの有用性を高めるために、ツリー構造（または親/子構造）を使用して関連メッセージがグループ化されます。さらに柔軟性と機能性を持たせるために、テキスト置換を使用してメッセージをアクティブにすることができます。これにより、ユーザーはメッセージのどこかをクリックして関連フォームを開くことができます。

バッチ・エラー・メッセージ処理について

バッチ・ジョブの完了後に表示されるエラー・メッセージは、〈Work Center〉に表示されます。これらのバッチ・エラー・メッセージを作成する際には、J.D. Edwards ソフトウェア・ユーザーに必要と思われるメッセージを判断する必要があります。たとえば、仕訳レポートの実行時に生成される多数のメッセージを作成するとします。レポートの貸借が一致している場合は、レポートの正常終了を示すメッセージを作成できます。また、レポートの貸借が一致しない場合は、各種のエラーを記述する複数レベルのメッセージを作成できます。この場合、最初のレベルではレポートがエラーで終了したことを示し、他のレベルではエラーの詳細を記述します。

レベル区切りメッセージ

レベル区切りメッセージは、バッチ処理によって作成される他のエラー・メッセージのコンテナとして機能するメッセージです。レベル区切りメッセージには、アプリケーションへのショートカットを提供してユーザーによるアクションを要求するアクション・メッセージを使用することができます。また、アクションを要求するのではなく、ユーザーに対して何らかの情報確認を要求するメッセージでもかまいません。

レベル区切りメッセージの役割を理解するには、バッチ・ジョブでエラーがどのように発生するかを調べる必要があります。バッチ処理全体で、エラーはほとんどがレベル区切りで発生する編集によって生成されます。一般的に、さまざまなレベル区切りで発生するエラーをグループ化します。これらのエラーをグループ化するメカニズムがレベル区切りメッセージです。このことは、〈Work Center〉アプリケーション・プログラム・インターフェイス(API)を使用してエラーを管理するレポーティング・プログラムでは、レベル区切りのフェーズごとに 1 つのレベル区切りメッセージを作成する必要があることを意味しています。

レベル区切りメッセージの使用法について

レベル区切りメッセージとアクション・メッセージは、1 つ以上のエラーをグループ化（またはパッケージ化）するために使用されます。

単独のメッセージは、最終エラー・メッセージ以外すべてレベル区切りメッセージです。Work Center API は、「job Completed（ジョブ完了）」レベル区切りメッセージを作成し管理しますが、それ以外のメッセージはいずれも開発者が作成します。レベル区切りメッセージはエラー・メッセージではありません。これらは、バッチ、文書、行などのエラーの発生場所に関する情報をユーザーに通知するために設定するメッセージをパッケージ化またはカテゴリ化するものです。これらのメッセージは、関数の jdeSetGBRError または jdeSetGBRErrorSubText を使って設定します。

たとえば、「Job R89004 ZJDE0001.c」と「Batch 3230 G in Error」はレベル区切りメッセージです。「Job R89004 ZJDE0001.c」はシステムによって自動的に生成されるレベル 1 メッセージで、「Batch 3230 G in Error」はレベル 2 メッセージです。「AA 20050606 3031」と「Intercompany Out of Balance（会社間勘定の貸借不一致）」は実際のエラー・メッセージです。

レベル区切りメッセージは 2 つの要素から構成されます。最初の要素はメッセージのテキストで、第 2 の要素はそれがアクション・メッセージかどうかを示します。

レベル区切りメッセージはいずれもテキスト要素を含んでいますが、それらすべてがアクション・メッセージとして有効であるとは限りません。レベル区切りメッセージのアクション・メッセージ部を有効にするかどうかの決定は、レポートの設計者に委ねられています。

関連メッセージをグループ化するには、設計時に適切なレベル区切りを使用します。Work Center API は、これらのレベル区切りで呼び出す必要があります。レベル区切りは通常、1 つのレコードの処理が終了し、新しいレコードが読み取られようとするときに、随時設定されるものです。この概念は、ネストされた複数のレコード区切りがレポートに存在する場合も同じです。

次の例では、貸借不一致の仕訳のアップロードが完了したときにメッセージがどのように表示されるかを示します。

第 1 レベルのメッセージ

第 1 レベルのメッセージは、ユーザーが各自の［Personal in-Baskets（受信トレイ）］を開いたときに表示されます。メッセージの横のプラス記号は、下位にメッセージ・レベルが存在することを示します。第 1 レベルのメッセージではバッチ・ジョブ名を示し、バッチ・ジョブがエラーになったことを伝え、ユーザーにエラーの詳細を検証するように指示することができます。

第 2 レベルのメッセージ

第 2 レベルのメッセージは、ユーザーが第 1 レベルのメッセージの横のプラス記号をダブルクリックすると表示されます。この例では、第 2 レベルのメッセージで、検証が必要なバッチ番号をユーザーに通知しています。

第 3 レベルのメッセージ

第 3 レベルのメッセージは、ユーザーが第 2 レベルのメッセージの横のプラス記号をダブルクリックすると表示されます。この例では、第 3 レベルのメッセージで、貸借不一致に起因してバッチ・ジョブがエラーになったことを通知し、問題の解決策を提供しています。

参照

- セクションのレベル区切りについては『エンタープライズ・レポート・ライティング』ガイドの「レベル区切りヘッダーおよびフッター・セクションの処理」

テキスト置換エラー・メッセージ

エラー・メッセージでは、ユーザーにできるだけ有用な情報を伝える必要があります。これは、実行時に日付や金額などの変数テキストをメッセージに埋め込むテキスト置換機能を通じて実現できます。テキスト置換メッセージはデータ辞書を使用して設定し、これによりアプリケーション一時変更や専門用語の統一性を保てます。たとえば、メッセージの「Voucher Batch 2453 contains errors（伝票バッチ 2453 にエラーがあります）」で、値の 2453 はメッセージへのパラメータとして編集されているとします。この場合、この値は実行時に置き換えられ、メッセージのその他の情報は、データ辞書エラー用語解説にそのまま保管されます。このメッセージは、データ辞書での略式記述です。メッセージを開くと、テキスト置換されたデータ辞書用語解説が表示されます。

参照

- 対話型メッセージおよびバッチ・メッセージのデータ項目作成、ランタイム・メッセージの用語解説テキストの定義、およびメッセージのデータ構造体の新規作成については、『開発ツール』ガイドの「テキスト置換エラー・メッセージの作成」

アクション・メッセージ

ユーザーはエラーを検討して解決策を決めた後、通常は改訂プログラムにアクセスしてエラーを解決する必要があります。ユーザーが対応する改訂プログラムを〈Work Center〉から直接呼び出してエラーを解決できるように、アクション・メッセージと呼ばれる特定のレベル区切りメッセージを設定することができます。アクション・メッセージは J.D. Edwards ソフトウェア・アプリケーションを呼び出して、必要な変数を受け渡します。たとえば、ユーザーはエラー・メッセージのショートカットをクリックするだけで、エラーのレコードをもつ〈Voucher Revision（伝票の改訂）〉フォームを自動的にロードできます。適切なアプリケーション、およびアプリケーションに受け渡す必要のある有効値を決めてください。アクション・メッセージはグリッドでハイライトされ、非アクション・メッセージと区別されます。

Work Center API

Work Center API を呼び出すと、この API は親/子順序を想定します。言い換えると、Work Center API を呼び出すと、この API は、ランタイム・エラー・メッセージ・スタックに存在するエラーは、いずれもその API 呼出しのインスタンスに送られたレベルに属するものとして想定するということです。これは、その時点でエラー・スペースに存在するエラーは、ビジネス関数を通じて設定されたものも、J.D. Edwards ソフトウェアのイベント・ルールを通じて設定されたものも、いずれも Work Center API に受け渡されたレベルの子として、パッケージ化またはグループ化されることを意味します。これらのエラー・メッセージはその後エラー・スペースからクリアされ、レコードの新しいセットに基づいてメッセージの新しいグループを作成できます。

Work Center API を呼び出すタイミングはとても重要です。レポート・プログラムは通常、詳細レコードのセットに通じるヘッダー・レベルのレコードの編集から開始します。詳細レコードは、読み取られ、処理される最初のレコードになります。したがって、Work Center API の呼出しは、降順のレベル区切り順序でレベル区切り番号を送ることになります。

たとえば API のレベル区切り呼出しの順序は、実際には 4、4、4、3、4、4、3、2、4、4、3、2、1 などとなります。

この順序は、呼出し構造が 4 つのレベルを下から開始したことを示しています。レベル 4 の最初の呼出しにより、Work Center API は、その時点で発生したメッセージを検出し、レベル 4 メッセージを親として使って子メッセージを作成することができます。ただし、エラーが発生しなければメッセージは作成されません。この例では、レベル 4 が 3 回呼び出された後に API がレベル 3 で呼び出されたことを示しています。レベル 3 の呼出しが行われると、Work Center API はレベル 4 のメッセージが書き出されたかどうかを検証します。言い換えると、レベル 4 の呼出しが行われたときにエラーが発生していなければ、Work Center API はレベル 3 のメッセージを作成しないということです。レベル 4 の呼出し時に少なくとも 1 つのエラーが発生していると、レベル 3 とレベル 2 のメッセージが作成されます。

Work Center API は、すべてのレベルで呼び出す必要があります。Work Center エラー・メッセージは親/子構造に基づいて作成されるため、レベルの呼出しがスキップされると、API は子メッセージと作成済みの子レベルをグループ化できません。

たとえば、レベル呼出し構造 6、6、5、4、3、4、4、3、2、1 は有効です。6、6、4、3、4、4、3、2、1 という呼出し順序は、レベル 6 が呼び出されたときに対応するレベル 5 の呼出しが存在しないため無効です。

Work Center API は、レポート・ジョブが完了に近づいたときにレベル 1 によって呼び出されなければなりません。そのため、レベル 1 がすべてのエラーおよびレベル区切りメッセージの親になり、ジョブ完了メッセージを表示します。Work Center API へのレベル 1 呼出しは必須です。この API へのレベル 1 呼出しは、親のない〈Work Center〉レコードの作成を防止するだけでなく、〈Work Center〉システムによって使用された割当記憶域をすべてクリアします。API へのレベル 1 呼出しはレポートで 1 回のみ、一般的にレポートの基本セクションの End Section イベントで発生する必要があります。

レベル区切りメッセージの作成

レベル区切りメッセージは、エラー・メッセージのコンテナとして機能します。アクション・メッセージかどうかに関係なく、レベル区切りメッセージを作成することができます。

レベル区切りメッセージは、データ辞書項目、エラー・データ構造体、ビジネス関数エラー構造体およびビジネス関数を作成して作ります。

参照

- アクティブ・メッセージと非アクティブ・メッセージについては、『開発ツール』ガイドの「アクション・メッセージ」

レベル区切りメッセージのデータ辞書項目の作成

データ辞書項目は、〈Work Center〉に表示されるレベル区切りメッセージのテキスト部です。データ辞書項目を作成する前に、作成済みのレベル区切りメッセージを検討し、必要なレベル区切りメッセージが存在しているかどうかを判断する必要があります。

▶ レベル区切りメッセージのデータ辞書項目を作成するには

1. 既存のレベル区切りメッセージの一覧を検索するには、〈Work With Data Dictionary Items〉を使用します。

   作成済みのレベル区切りメッセージごとに、データ辞書項目とそれを参照する少なくとも 1 つのビジネス関数も作成されています。既存のレベル区切りメッセージを確認するには、〈Object Management Workbench〉(OMW)の QBE ローを使用して、BLM の 3 文字から始まるすべてのビジネス関数(オブジェクト・タイプ BSFN)を検索します。

   BLM 命名規則は、特に J.D. Edwards 作成のレベル区切りメッセージ用ビジネス関数に関連します。レベル区切りメッセージ用の独自ビジネス関数の名前には、BLMxxyy の形式を使用することをお勧めします。xx は使用するシステム・コード(50～55)で、yy は固有番号です。この固有番号は、他の項目にも適用されます。文字数は最大 8 文字です。

   注:
   [Description(記述)]フィールドを使用すると、既存のメッセージがレポートで使用できるかどうかを判断する上で役立ちます。

   レポートの設計に適合すると思われるレベル区切りメッセージ・ビジネス関数がいくつかある場合は、それらを書き留め、データ辞書項目自体でテキストを確認します。その際、データ辞書を使用して、LM の後に続く番号が、対応する BLM 関数名の後に続く番号と一致するデータ項目を照会します。

   LM 命名規則は、特に J.D. Edwards 作成のテキスト置換を含むレベル区切りメッセージに関連します。これらは用語解説グループ Y に分類されます。テキスト置換を含む独自レベル区切りメッセージの名前には、Lxxyy の形式を使用することをお勧めします。xx は使用するシステム・コード(50～55)で、yy は固有番号です。文字数は最大 8 文字です。

   たとえば、記述を表示するレベル区切りメッセージがビジネス関数 Set Level A/R Receipt Pre-Post error (BLM0025)の場合は、データ項目 Receipt &1 Customer &2 In Error (LM0025)を照会します。

2. テキスト記述が基準に適合した場合は、レベル区切りメッセージが使用する置換変数が有効かどうかを確認します。

   LM メッセージのテキストに置換変数(&1 など)が含まれている場合は、レベル区切りメッセージが使用するデータ構造体のデータ項目を確認する必要があります。レポートでは、構造体のデータ項目を受け渡します。

   データ項目がデータ構造体に存在するかどうかを確認するには、OMW を使用して、以前に使用したものと同じ番号が後続するデータ構造体 DELM を照会します。

   DELM 命名規則は、特に J.D. Edwards 作成のレベル区切りメッセージ・データ構造体に関連します。独自のレベル区切りメッセージに名前を設定するときには、DELMxxyy 形式を使用してください。xx は使用するシステム・コード、yy は固有番号です。この固有番号は、以前に使用した番号と同じにする必要があります。文字数は最大 8 文字です。

   たとえば、DELM0025 がレベル区切りメッセージ LM0025 のテキスト置換用に作成されたデータ構造体であるとします。この構造体をチェックアウトして、個々のフィールドを表示する必要があります。

データ辞書項目のデータ構造体の作成

作成されたレベル区切りメッセージは、それぞれに対応するデータ構造体が必要です。データ構造体は、メッセージが表示されるときに置き換えられる適正なデータ項目と一緒に定義します。たとえば[Address Book Number(住所録番号)]の特定のエラーを表すレベル区切りメッセージの場合は、対応するデータ構造体に〈Address Book〉システムの固有番号の項目を示すデータ項目 AN8 があります。この制限は、新しい多くのデータ構造体が作成されることを意味しています。

データ構造体を作成するには、OMW を使用します。名前には、Lxx データ辞書項目の作成時に使用したものと同じ固有番号を使用してください。この番号は DELxx に付け加えられます。たとえばレベル区切りメッセージ L55025 では、データ構造体の名前として DEL55025 を使用します。

このデータ構造体では typedef を作成することも必要です。作成するビジネス関数にこの typedef を組み入れます。typedef はデータ構造体を C 言語コードに変換して、ビジネス関数で使用できるようにします。

参照

- データ構造体については、『開発ツール』ガイドの「データ構造体」

レベル区切りメッセージ・ビジネス関数データ構造体の作成

レベル区切りメッセージを配信するためのビジネス関数を作成します。作成するビジネス関数には、テキスト置換用の変数とメッセージをアクティブにする際に必要な変数の他に、いくつかの標準パラメータが必要です。これは、これらすべての変数を受け渡すための第 2 データ構造体を作成しなければならないことを意味します。

DLM 命名規則は、特に J.D. Edwards 作成のビジネス関数データ構造体に関連します。独自のレベル区切りメッセージに名前を設定するときには、次の基準に従ってください。DLxxyy という形式にします。xx は使用しているシステム・コード(55～59)、yy は固有番号です。この固有番号は、以前に使用した番号と同じにします。文字数は最大 8 文字です。

たとえば J.D. Edwards 作成のレベル区切りメッセージ LM0025 には、そのビジネス関数データ構造体の名前として DLM0025 を使用しています。それぞれのレベル区切りメッセージでビジネス関数を使用する必要があります。このデータ構造体と、前記の手順でデータ辞書項目のために作成したデータ構造体とは別のものです。両者には、ビジネス関数のデータ構造体はデータ変数をレベル区切り関数に移動するために使用されるという違いがあります。データ辞書項目用のデータ構造体は、特定のデータ辞書項目の用語解説にマッピングされるデータの保管に使用されます。

ビジネス関数のデータ構造体を作成するときには、次の項目を組み入れてください。

- レベル区切りテキスト置換メッセージ用のすべてのデータ項目
- メッセージをアクティブにするのに必要なすべてのデータ項目(所定の J.D. Edwards アプリケーションでフォームのデータ構造体をロードするために必要な変数)。フォーム・インターコネクト用のデータ項目は、ハンガリアン表記のプレフィックスの後に文字 FI_が表示されるように、名前のみ変更する必要があります(例:jdFI_GlDate、mnFI_Openamount)。メッセージをアクティブにしない場合は、これらの項目を組み入れる必要はありません。

- データ項目 ev01 を追加し、その変数名を OneWorldEventPoint01 から cIncludeInterconnect に変更します。このパラメータは、メッセージがアクティブであるかどうかを確認するためのフラグとして使用されます。このパラメータは、レベル区切りメッセージの本来の意図が J.D. Edwards アプリケーションを呼び出すことでない場合でも、すべてのレベル区切りメッセージに共通のパラメータにしてください。これにより、レベル区切りメッセージを異なるアプリケーションで使用できるようになりますが、アプリケーションを起動することはできません。アプリケーションを起動するには、データ構造体の値に 1 を設定する必要があります。
- データ項目 genlng を追加し、その変数名を GenericLong から idGenlong に変更します。このパラメータは、すべての〈Work Center〉メッセージ処理を制御するために使用されます。これは、システムのワーク・フィールド以外での使用は意図されていません。

レベル区切りビジネス関数の作成

DLxx データ構造体を作成した後に、レベル区切りエラーを処理し、アクティブ・メッセージのすべてのマッピングを実行するビジネス関数を作成します。

▶ レベル区切りビジネス関数を作成するには

1. 新しいビジネス関数のオブジェクトを〈Object Management（オブジェクト管理）〉で作成します。

   ビジネス関数の名前は、BLxx の後に固有番号を付けます。たとえばレベル区切りメッセージのデータ辞書項目名を Receipt &1 Customer &2 In Error (L55025)に設定した場合は、ビジネス関数名を Set Level A/R Receipt Pre-Post error (BL55025)に設定します。J.D. Edwards の命名規則は BLMxx です。

2. 〈Object Management Workbench（オブジェクト管理ワークベンチ）〉の〈Business Function Design（ビジネス関数の設計）〉フォームに、実際の関数の名前を入力します。

   関数名を付けるときは、名前の先頭を SetLevel_xx にします。xx はシステム・コードです。関数には、レベル区切りメッセージの用途を示す他の記述語を追加します。たとえば、ビジネス関数名を BLM0025 に設定します。

3. 〈Business Function Design〉で、ローを強調表示し、[Row（ロー）]メニューから[Parameters（パラメータ）]を選択して、ビジネス関数データ構造体をビジネス関数に追加します。

   このデータ構造体は、既に作成した DLxx データ構造体にしてください。

4. .h ファイルを生成するには、[Form Create（フォーム作成）]を選択します。
5. ビジネス関数の typedef をコピーし、そのヘッダ・ファイルに貼り付けます。
6. テキスト置換データ構造体 DExx の typedef を作成します。

   typedef を作成するには、使用されない関数行を作成する必要があります。

7. typedef を作成したら、ビジネス関数ヘッダー・ファイルの[Structure Definitions（構造定義）]セクションにデータ構造体テンプレートを貼り付けます。
8. 既存の BLMxxxx.C ビジネス関数のソース・コードをコピーして、レベル区切りメッセージのソース・ファイルを修正します。
9. 新規のレベル区切りビジネス関数にソースをコピーした後、コピーされたソースを検討し、変更を行います。

サンプル・ソース・コードのハイライト

次のシェル上のソース・コードのサンプルでは、使用する必要があるコード部分とビジネス関数用の独自コードを入力する個所を示しています。

ビジネス関数のデータ構造体から dsTextData データ構造体に、フィールドを手作業でマッピングする必要があります。dsTextData データ構造体は、レベル区切りメッセージでのテキスト置換用のデータ構造体です。また、ビジネス関数のデータ構造体から dsFormData データ構造体にもフィールドを手作業でマッピングする必要があります。dsFormData データ構造体は、アクティブ・メッセージで使用するデータ構造体です。

変数の宣言

「Variable declarations」のセクションでは、次の行を設定してレベル区切りメッセージの変数を宣言します。

```
char szForm[11];     /* Name of form in application */
char szDDitem[11];   /* Data dictionary name of the level message */
char szDLLName[11];  /* Name of the application DLL */
char szDsTmp1[11];   /* Name of the text substitution data structure */
```

構造体の宣言

「Declare structures」のセクションでは、タイプのレベル区切りメッセージの専用コードを入力します。次の行は、既存のビジネス関数からの引用例です。

```
DSDELM0002 dsTextData;     /* Instance of text substitution structure */
FORMDSW0411Z1D dsFormData; /* Instance of form interconnect structure */
```

ポインタの設定

「Set pointers」のセクションでは、次の行を設定してレベル区切りメッセージの動作を確保します。

```
if (lpDS->idGenLong == (ID) 0)
{
      jdeSetGBRError (lpBhvrCom, lpVoid, (ID) 0, "4363");
      if (hUser)
      {
      JDB_FreeBhvr (hUser);
      }
      return ER_ERROR;
}
else
      lpDSwork = (LPDS_B0100011A) jdeRetrieveDataPtr (hUser, lpDS->idGenLong);
```

主要な処理機能

「Main Processing」のセクションでは次の行を設定します。斜体の項目を独自の項目に置き換えます。

```
strncopy      ((char*) szDsTmp1, (const char*) (”DELM002”), sizeof (szDsTmp1) -1);
strncopy      (szDDitem, (const char*) ( “LM0002”), sizeof (szDDitem));
memset                ((void*) (&dsTextData), (int) (’¥0’ ), sizeof (dsTextData));
```

lpDS データ構造体から dsTextData への値の割当て

「Assign values from lpDS data structure to dsTextData here」のセクションでは、レベル区切りメッセージの専用コードを入力します。値を割り当てるときには、ビジネス関数データ構造体項目をデータ辞書データ構造体項目にマップします。次の行は、既存のビジネス関数からの引用例です。

```
strncpy (dsTextData.szEdiuserid,(const char *)(lpDs-> szEdiuserid),
      sizeof(dsTextData.szEdiuserid));
strncpy (dsTextData.szEdibatchnumber,(const char *)(lpDS-> szEdibatchnumber),
      sizeof(dsTextData.szEdibatchnumber));
strncpy (dsTextData.szEditransactnumber,
      (const char *)(lpDS->szEditransactnumber),
      sizeof(dsTextData.szEditransactnumber));
```

この例では、dsTextData.szEditransactnumber はデータ辞書データ構造体項目で、pDS->szEditransactnumber はビジネス関数データ構造体項目です。文字列では Strncopy API を使用し、数値では Mathcopy API を使用します。日付に memcpy を使用すると、文字が直接割り当てられます。

lpDS データ構造体から dsTextData への値の割当て

「Assign values from lpDS data structure to dsTextData here」のセクションでは、次の行を設定してレベル区切りメッセージの動作を確保します。

```
JDBRS_GetDSTMPLSpecs (hUser, (char*) szDsTmpl, &lpDSwork->lpBlob->lpTSDSMPL);
if (lpDSwork->lpBlob->lpTSDSMPL != (LPDSTMPL)NULL)
{
      lpDSwork->lpBlob->lpTSTEXT=(LPSTR)AllocBuildStrFromDstmplName((LPDSTMPL)
      lpDSwork->lpBlob->lpTSDSMPL,(char*)szDsTmpl,
      (LPVOID)&dsTextData);
      strncpy (lpDSwork->lpBlob->szDDItem, (const char *)(szDDitem),
      sizeof(lpDSwork->lpBlob->szDDItem));
}
if(lpDS->cIncludeInterconnect == ’1’)
```

フォーム・インターコネクト処理

「Form interconnect processing」のセクションでは、レベル区切りメッセージの専用コードを入力します。次の行は、既存のビジネス関数からの引用例です。「Form interconnect processing」のセクションでは、次の行を設定してレベル区切りメッセージの動作を確保します。

```
strncpy     (szDLLName, (const char*)("P0411Z1"), sizeof (szDLLName));
memset      ((void *)(&dsFormData), (int)('¥0'), sizeof(dsFormData));
memset      ((void *)(szForm), (int)('¥0'), suzeif(szForm))
strncpy     ((char *) szForm, (const char *)("W0411Z1D"), sizeof (szForm) -1);
```

LpDS データ構造体から dsFormData への値の割当て

「Assign values from LpDS data structure to dsFormData」のセクションでは、レベル区切りメッセージの専用コードを入力します。次の行は、既存のビジネス関数からの引用例です。この例は、ビジネス関数のデータ構造体からフォームのデータ構造体に情報をどのように受け渡すかを示しています。

```
strncpy (dsFormData.EDUS,(const char *)(lpDS->szEdiuserid),
      sizeof(dsFormData.EDUS));
strncpy (dsFormData.EDBT,(const char *)(lpDS-> szEdibatchnumber),
      sizeof(dsFormData.EDBT));
strncpy (dsFormData.EDTN, (const char *)(lpDS->szEditransactnumber),
      sizeof(dsFormData.EDTN));
ParseNumericString(&dsFormData.EDLN, "1.0");
dsFormData.EV01 = '1';
```

SVRDTL テーブルからのフォーム・データ構造体 ID の取得

「Get the form data structure id from the SVRDTL table」のセクションでは、次の行を設定してレベル区切りメッセージの動作を確保します。

```
lptam = jdeTAMInit (FILENAME_SVRDTL);
 strncpy ((char *) lpDSwork->lpBlob->szForm, (const char *)(szForm),size of
      (lpDswork->lpBlob->szForm) -1);
if (lptam != (LPTAM) NULL)
{
      lpASVRdtl = TAMAllocFetchByKey (lptam, INDEX4_ASVRDTL, szForm, 1);
      if (lpASVRdtl != (void *) NULL)
      {
                  JDBRS_GetDSTMPLSpecs(hUser, (char*)lpASVRdtl->szFITemplateName,
      &lpDswork->lpBlob->lpFINDSMPL);
                  if (lpDSwork->lpBlob->lpFIDSMPL != (LPDSTMPL)Null)
                  {
```

```
    lpDSwork->lpBlob->lpFITEXT=(LPSTR) AllocBuildStrFromDstmplName
                        ((LPSTMPL)
                        lpDSWork->lpBlob->lpFIDSMPL,
    (char*)lpASVRdtl->szFITEmplateName
                                (LPVOID) &dsFormData);
            strngcpy (lpDSwork->lpBlob->szDLLName, (const char *)(szDLLName),
                sizeof(lpDSwork->lpBlob->szDLLName));
            }
    TAMFree(lpASVRdtl);
  }
 TAMTerminate(lptam);
 }
}
```

関数のクリーンアップ

「Function Clean Up」のセクションでは、次の行を設定してレベル区切りメッセージの動作を確保します。

```
  if(hUser)
  {
      JDB_FreeBhvr(hUser);
  }
  return (ER_SUCCESS);
}
```

Work Center 初期化 API の呼出し

ビジネス関数を作成したら、コンパイルして〈Object Management Workbench（オブジェクト管理ワークベンチ）〉にチェックインします。その後で Work Center API を関連付けます。API の関連付けに際しては、最初に〈Work Center〉でイベント・ルールを呼び出します。これは通常、親セクションの init section で呼び出します。

処理 Work Center API の呼出し

〈Work Center〉システムを初期化したら、レポートでさまざまなレベル区切り時点を指定し、これらのタイミングごとに〈Work Center〉システムを呼び出してエラーをグループ化する必要があります。

► 処理 Work Center API を呼び出すには

1. 〈Report Design〉から、レポート・バッチ・ジョブに適切なレベル区切りをすべて設定します。

   所定のイベントですべてのエラーを論理的にグループ化するイベントを分析する必要があります。一般にこれは、1 群のレコードですべての編集を完了したときのイベントか、または個々のレコードの編集がすべて行われた直後となります。

2. レベル区切りを設定するごとに次の手順を実行します。

   a. 適切なレベル区切りで、レベル区切り用メッセージ・ビジネス関数を呼び出します。レベル区切りメッセージ・ビジネス関数は、特定のレベル区切りで発生するエラー・グループ化のタイプに関連する必要があります。たとえば SetLevel_SFVoucher は、伝票レベル区切りに関連するエラーをグループ化します。レポートでは、このビジネス関数は Do Section で呼び出されるのが普通です。インターコネクトがブランクの場合は、アクション・メッセージを呼び出しません。

   注:
   この API を送るレベルは 1 にはなりません。これは、処理を終了したときに 1 回だけ送られます。

   b. レベル区切りメッセージを呼び出した直後に〈Work Center〉エラー・メッセージ関数を呼び出します。この関数は ProcessErrorsToPPAT という名前です。B0100011.c は JDEM へのバッチ・エラーを処理して反復呼出しを行います。

   c. 送信するパラメータを識別するには、API に関する Windows ヘルプ・ファイルを参照してください。

次のテーブルを使って、どのパラメータを送るかを指定します。ここで、データ辞書レベル区切りメッセージのレベルを指定します。メッセージをレベル 2 から開始します。レベル 1 は、システムによって自動的に生成されます。

パラメータ	説 明	許容値	標準値
mnLeveloftotaling	Work Center API に処理させるメッセージ・レベルです。	有効数値:1～20。 数値の「1」は、UBE がほぼ終了するときにのみ使用します。「1」は、「Job Completed（ジョブ完了）」メッセージを書き出します。	数値:1～20
idDataBaseWorkField	Work Center API によって使用されるワーク・フィールド。これは、[GENLNG]ワーク・フィールドが受け渡される場所です。	UBE で定義されているワーク・フィールドにします。GENLNG データ項目を使用してください。	GENLNG ワーク・フィールド

パラメータ	説　明	許容値	標準値
cErrorPreProcessFlag	このパラメータにより、UBE は一定のイベントで Work Center API を呼び出して、「エラー・スペース」をフラッシュし、未送信のレベル区切りメッセージ別にエラーをグループ化することができます。	ブランクまたは文字「P」を使用します。 ブランクの場合、エラー・スペースのエラーは、最初のパラメータで送信されるレベル・メッセージによってグループ化されます。	ブランク値またはデフォルト
szUserid	cAllowUserIDToChange オプションが初期化関数で設定されているとき、このパラメータは新しいユーザーID を受け入れます。	ブランクまたは有効なユーザーID を使用します。 ブランクの場合、最後に使用されたユーザーID にすべてのメッセージが書き出されます。	ブランク値またはデフォルト

cAllowUserIdToChange パラメータ

初期化 API の cAllowUserIdToChange パラメータは、ProcessErrorsToPPAT API の szUserid パラメータと連携して動作します。このパラメータにより、バッチ・エラーが発生したときに、元のレコードを作成したユーザーにはエラーが送信され、（夜間オペレータなどの）ジョブの投入者には送信されないように UBE を設定することができます。たとえば、単一のバッチ・ジョブが、50 名のユーザーによって作成された 1,000 のトランザクションを含んでいる場合、エラーのあるトランザクションを作成したユーザーだけにエラー・メッセージが送信されます。夜間オペレータもメッセージを受信しますが、それは「Job completed normally（ジョブが正常終了しました）」や「Job completed normally with errors（ジョブは正常終了しましたが、エラーが存在します）」などのメッセージです。トランザクションに問題がなかったユーザーには、エラー・メッセージは送られません。

この機能を設定するには、バッチ・エラー処理システムを初期化するときに cAllowUserIDToChange パラメータに"1"を入力する必要があります。レベル 2 のレベル区切りメッセージを処理してから ProcessErrorstoPPAT API を呼び出したときでも、szUserid パラメータを使ってメッセージの受信者を指定することができます。メッセージを受信するユーザーは、トランザクション・レコードを検討することで特定できます。

Work Center プロセスの終了

すべてのメッセージが〈Work Center〉に送信されたら、レポート・ジョブを終了する前に〈Work Center〉プロセスを終了する必要があります。バッチ・プログラムの終了間際に、〈Work Center〉エラー・メッセージ関数の〈ProcessErrorsToPPAT〉を最後に 1 回呼び出して、レベル 1 に送ります。レベル 1 は、レベルの合計が 1 で、それが完了していることを示します。これにより、ジョブ完了メッセージが作成され、Work Center API によって作成されたワークスペースがすべて解放されます。

Work Center API を使用してエラーを処理するレポート設計は、いずれも処理の最後にレベル 1 を使用して、この API を呼び出す必要があります。この呼出しを実行するには、重大なエラーを監視し、早期に終了するジョブを報告する必要もあります。

レポートの処理を終了すると、〈Work Center〉メッセージが作成されます。これは、その後 J.D. Edwards の〈Work Center〉アプリケーションを使用して読み取ることができます。バッチ・エラーは処理されて JDEM システムに送られます。作成されたメッセージは、そのメッセージの送信先として別のユーザーを指定した場合を除き、レポートを実行したユーザーに送信されます。

エラーが発生しなかった場合、Work Center API はジョブが正常終了したことを示すメッセージを〈Work Center〉に送信します。

▶ Work Center 初期化 API を呼び出すには

1. 〈Report Design（レポート設計）〉で、グローバルなワーク・フィールドを作成します。ワーク・フィールドを作成するときには、データ辞書項目 GENLNG を使用します。
2. 〈Work Center〉初期化関数を呼び出します。レポートがこのビジネス関数を検索します。この関数は InitializePPATapi で、実際のソース・ファイル名は B0100025.C です。F01131 は EditJDEM エラー・メッセージです。

   一般にこの関数は、Initialize Section API を使用するレポートの基本セクションで呼び出されます。

   どのパラメータを渡すかを識別するには次の表を使用してください。

パラメータ	説　明	許容値	標準値
szUserid	ユーザーID 一時変更。受け渡されないと、UBE を実行したユーザーにすべてのメッセージが送信されます。このフィールドに値を入力すると、次の〈Address Number（住所番号）〉一時変更パラメータに置き換わります。	ブランクまたは有効なユーザーID を使用します。 ブランクの場合は、すべてのメッセージが UBE 投入者に送信されます。	ブランク値
mnAddressnumber	ユーザーID 一時変更。このフィールドは、ユーザーの住所はわかっていても、ユーザーID がわからない場合に使用します。	ブランクまたは有効なユーザーの住所番号。 ブランクの場合は、UBE 投入者にすべてのメッセージが送信されます。	ブランク値
cDoNotLogWarnings	この機能をオンに設定すると、警告メッセージが〈Work Center〉に一切送信されなくなります。	ブランクまたは数値の「1」を使用します。 ブランクの場合、警告とエラーがすべて処理されます。	ブランク値
cAllowUserIDToChange	この機能をオンに設定すると、メッセージを受信するユーザーを随時変更できます。一定の制限があります。	ブランクまたは数値の「1」を使用します。 ブランクの場合、すべてのメッセージが 1 人のユーザーに送信されます。	ブランク値
szMailboxdesignator	メールボックス（または待ち行列）一時変更。このフィールドをオンに設定すると、メッセージは一時変更されたメールボックスに送信されます（〈Work Center〉用）。	ブランクまたは有効なメールボックス指定子。 ブランクの場合、すべてのメッセージがデフォルト待ち行列に書き出されます。	ブランク値

idPPATworkField	Work Center API によって使用されるワーク・フィールド。これは、[GENLNG]ワーク・フィールドが受け渡される場所です。	UBE で定義されているワーク・フィールドにします。GENLNG データ項目を使用してください。	GENLNG ワーク・フィールド
cSendErrorsToReport	この機能をオンに設定すると、(「Job Completed」以外の)メッセージは〈Work Center〉に送信されません。その場合、メッセージは、別の Work Center 関数を通じて読み取ることができます。	ブランクまたは数値の「1」を使用します。	ブランク値

3. この API で、F01131 Edit JDEM Error Message 関数を選択します。
4. 送信するパラメータを識別するには、API に関する Windows ヘルプ・ファイルを参照してください。

デバッグ

デバッグとは、任意の実行箇所でプログラムの処理状態を確認する方法です。デバッグ機能を使用すると、問題を解決したり、プログラムの実行をテストして確認することができます。

デバッガでは、プログラムの実行を停止して、特定の位置でのプログラムの状態を調べることができます。つまり、特定したコード行を実行した時点での入力パラメータ、出力パラメータ、および変数の値をみることができます。実行フローやデータ整合性などの問題を調べるためにプログラムの実行を途中で停止し、コードを 1 行ずつ実行して調べることもできます。

J.D. Edwards ソフトウェアでは、次の 2 つのデバッグ・ツールを使用することができます。

- J.D. Edwards ソフトウェアの〈Event Rules Debugger〉
- Microsoft Visual C++ Debugger

〈Event Rules Debugger〉を使用して、イベント・ルールのみでなく次のデバッグを実行します。

- 対話型アプリケーション
- レポート
- テーブル・コンバージョン

Visual C++ Debugger は、C 言語ビジネス関数のデバッグに使用します。

デバッグ・プロセスの概要

デバッグ・プロセスは、問題発生箇所の特定とそれらの問題の解決という 2 つの作業から構成されます。この作業の目的は、各プログラムを一定の範囲で区切り、その中でプログラムがどのように動作するかを正確に検証することです。

Event Rules Debugger を使用してデバッグを実行している最中にプログラムを変更する場合は、次の作業を行ってください。

1. 〈Event Rules Debugger〉を停止します。
2. デバッグ情報をリビルドします。
3. ブレークポイントをリセットします。
4. アプリケーションを実行します。

次の表に、デバッガで使用可能な一部の共通機能を示します。

Feature(機能)	使用可能なツール	説　明
Go	Visual C++と〈Event Rules Debugger〉	ブレークポイントに達した後にプログラムを再起動するコマンド。 Visual C++では、アプリケーションを起動すると、ブレークポイントに到達するまで動作します。 〈Event Rules Debugger〉では、アプリケーションを最初に J.D. Edwards Explorer から起動する必要があります。
Breakpoints	Visual C++と〈Event Rules Debugger〉	特定の行に達したときにデバッガに停止するように指示するコマンド。ブレークポイントは、デバッグを開始するコード行に設定することができます。
Step	Visual C++と〈Event Rules Debugger〉	現在のコード行を実行するコマンド。Step 機能により、プログラムを行単位で実行できます。この機能を使用すると、コード行ごとに実行結果を確認することができます。
Step Into	Visual C++と〈Event Rules Debugger〉	現在のコード行に関数呼出しが含まれている場合に使用するコマンド。デバッガが関数の中に入り、行単位でデバッグすることができます。関数が終了すると、デバッガは呼出し側ルーチンの関数呼出しの次のコード行に戻ります。J.D. Edwards ソフトウェアでは、Step Into コマンドを使用すると、呼出し側 J.D. Edwards アプリケーションの中から呼び出される第 2 のアプリケーションをデバッグすることができます。
Skip	Visual C++と〈Event Rules Debugger〉	1 コード行の実行をスキップするコマンド。実行されなかったコード行は赤く変わります。
Step Over		1 コード行を実行するコマンド。そのコード行が関数呼出し、レポート・インターコネクトまたはフォーム・インターコネクトの場合には、その呼出しの中までは入りません。
Stop	Visual C++と〈Event Rules Debugger〉	デバッグ・プロセスを停止するコマンド。 Event Rules Debugger を停止すると、アプリケーションは、デバッガが起動されなかったかのように動作し続けます。 Visual C++ではデバッグ・プロセスが停止し、実行中のアプリケーションも終了します。
Watch	Visual C++と〈Event Rules Debugger〉	プログラムの実行中に変数の値を表示するコマンド。また式を検査することもでき、変数が変化したときに特定の式がどのように変化するかを確認できます。

インタープリティブ・コードとコンパイル済コード

J.D. Edwards ソフトウェアは、インタープリティブ・コードとコンパイル済コードの両方を使用します。

インタープリティブ・コードとは、実行時にコンパイルされるコードをいいます。プログラム命令から機械語命令への翻訳が実行時に行われます。インタープリティブ・コードはアプリケーションの中に存在します。〈Form Design〉と〈Report Design〉の中で使用されるイベント・ルール・スクリプト・コードはインタープリティブ・コードです。インタープリティブ・コードを使用すると、何か変更するたびに再コンパイルせずにアプリケーションをカスタマイズできます。

コンパイル済コードは、単独で呼び出すことができるオブジェクト・ファイルにコンパイルされて保管されます。プログラム命令から機械語命令への翻訳は、コンパイル時に行われています。たとえば、テーブル・イベント・ルール、イベント・ルール・ビジネス関数、ビジネス関数は事前にコンパイルされています。これらはアプリケーションの外に常駐し、ほとんど変化しません。Visual C++を使用すると、コンパイル済コードはデバッグすることができます。イベント・ルール・スクリプト言語は、C、Java または現在使用している言語に翻訳できます。ただし、ロジックは一度解読される必要があります。コンパイル済コードには、一般にインタープリティブ・コードほど柔軟性がありません。

Event Rules Debugger の処理

〈Event Rules Debugger〉は、J.D. Edwards の対話型/バッチ・アプリケーションのデバッグ作業に欠かせないデバッグ・ツール(ブレークポイント、ステップ・コマンド、変数の点検など)を提供します。Event Rules Debugger は、イベント・ルール・ビジネス関数とテーブル・イベント・ルールのデバッグに使用することができます。〈Event Rules Debugger〉がアプリケーションのデバッグ情報を生成するときには、そのアプリケーションのイベント・ルール・ビジネス関数とテーブル・イベント・ルールに関する情報を取り入れます。

〈Event Rules Debugger〉を設定して使用するには、次の 2 つのステップが必要です。

- Event Rules Debugger の実行とブレークポイントの設定
- アプリケーション、レポートまたはテーブル変換の実行とデバッグ

デバッグ情報テーブルを保存しても、デバッガを作業時にアクティブにしたくない場合は、デバッグ情報を非アクティブ化することができます。デバッグ作業を継続する際は、このテーブルを随時アクティブにすることができます。

Event Rules Debugger について

〈Event Rules Debugger〉は、次の 5 つのメイン・コントロールから構成される独立型ツール・プログラムです。

- Object Browse Window
- Event Rules Window
- Breakpoint Manager
- Variable Selection and Display Window
- Search combo box

〈Event Rules〉ウィンドウ以外のコントロールは、いずれもメイン・アプリケーションのどの側にもドッキングすることができます。ウィンドウにマウスを合わせて右クリックすると、ウィンドウをドッキングしたり非表示にすることができます。デバッガを閉じると、ドッキング設定は次回の実行時まで保存されます。

Object Browse Window

〈Object Browse（オブジェクト参照）〉ウィンドウには、デバック情報が作成されたアプリケーションとデバッグが可能なアプリケーションの一覧が表示されます。ツリーをたどって特定のイベントにアクセスし、そのイベントの〈Event Rules〉ウィンドウを開くことができます。

Event Rules Window

〈Event Rules〉ウィンドウには、1 つのイベントのイベント・ルールが表示されます。イベントの名前とパスは、各〈Event Rule〉ウィンドウのタイトル・バーに表示されます。タイトルは、マウスの右ボタンをクリックして、そのロング・バージョンとショート・バージョンの間で切り替えることができます。〈Event Rules〉ウィンドウには、現在実行中のイベント・ルールの行が表示されます。

〈Event Rule〉ウィンドウの左側には、イベント・ルールの行の状態を示すアイコンが表示されます。次の 3 つの状態があります。

- Breakpoint（ブレークポイント）
- Disabled（無効）
- Current line of execution（現在実行中の行）

〈Event Rules〉ウィンドウを使用して、ブレークポイントを設定および解除できます。次の方法を使用できます。

- イベント・ルールの行をダブルクリックします。
- 行を選択し、[Debug]メニューから[Breakpoints（ブレークポイント）]を選択します。
- 行を右クリックし、ポップアップ・メニューから[Breakpoints]を選択します。
- 行を選択して、[F9]キーを押します。

Variable Selection and Display Window

〈Variable Selection and Display（変数の選択と表示）〉ウィンドウは、左右の 2 つのビューから構成されます。左側のビューには、変数タイプを親ノード、そのタイプの現在の変数を子ノードとしてリストするツリー構造が表示されます。表示される変数は、現在の〈Event Rule〉ウィンドウに関連付けられています。右側のコントロールは〈Variable Display（変数の表示）〉ウィンドウです。このウィンドウには、ユーザーが選択した変数とそれらの現在の値が表示されます。〈Variable Display〉ウィンドウには、〈Variables Tree（変数のツリー）〉で必要な変数をダブルクリックすると、その変数を追加することができます。

変数の値は、アプリケーションをデバッグしている最中に変更することができます。変数の値を変更するには、〈Variable Display〉ウィンドウで変数をダブルクリックして別の値を入力します。新しい値は、〈Variable Display〉ウィンドウに表示されます。不適切な値を入力する（数値を文字値に変更したりする）と、新しい値は設定されず値は変更されません。

変数には、次の 3 つの特別な値が表示されることがあります。

blank この変数値のは、ブランクのみが入っています。これは、文字列および文字タイプのみに適用されます。

null この変数の値は、ASCII の「¥0」に設定されています。

unknown この変数の値は、ランタイム・エンジンから取得できていないことを示しています。これは、アプリケーションが動作していなかったり、変数が有効範囲から外れているときに起こります。

注:

変数の点検と修正は、イベント・ルール・ビジネス関数とテーブル・イベント・ルールのデバッグでは実行できません。

Breakpoint Manager

ブレークポイントは、プログラムの実行がいつ、どこで中断するかをデバッガに伝えます。プログラムがブレークポイントで停止したときには、〈Variable Selection and Display〉ウィンドウを使用してランタイム構造体の状態を調べ、イベント・ルールの中をたどり、式を検査することができます。

〈Breakpoint Manager〉は、アプリケーションで設定されているブレークポイントとその位置をトラッキングします。新しいブレークポイントを設定すると、〈Breakpoint Manager〉でエントリが作成されます。このエントリには、アプリケーション名、フォーム名、イベント名およびイベント・ルール行が入っています。

〈Breakpoint Manager〉を右クリックすると、次の操作を実行できます。

- ブレークポイントの削除
- ブレークポイントの一括削除
- ブレークポイント位置への移動

また〈Breakpoint Manager〉でエントリをダブルクリックすると、そのブレークポイントが設定されている〈Event Rule〉ウィンドウを開くこともできます。

Search Combo Box

イベント・ルール・テキストを検索するには、ツールバーの[Search combo box(コンボ・ボックスの検索)]を使用します。検索する場合は、[Search]コンボ・ボックスにテキストを入力して[Enter]キーまたは[F3]キーを押します。イベント・ルール・テキストで検索テキストが見つかると、そのテキストがハイライトされます。[Enter]キーまたは[F3]キーを再度押すと、テキストの次の候補が検索され、ハイライトされます。

この検索では、特殊コードを使用できます。特殊コードによる検索では、テキストに対応する次のようなコードを使用します。たとえば、「^If」は、「If」で始まるすべての行を検索し、「If$」は、「If」で終了するすべての行を検索します。

次の表に示す特殊コードを使用すると、より高度な検索を実行できます。

^	カレット(^)は、行頭を意味します。たとえば、正規表現の「^A」は、行頭の「A」だけを検索します。
^	左角カッコの直後のカレット(^)は、角カッコ内の文字を除いた文字列を検索対象とします。たとえば、「[^0-9]」は、数値でない文字を検索します。
$	ドル記号($)は、行末を意味します。たとえば、「abc$」は、行末の「abc」だけを検索します。
|	代替記号(|)を使用すると、左右のどちらかの表現を検索対象にできます。たとえば、「a|b」は、「a」または「b」を検索します。
.	ドット(.)は、任意の 1 文字を検索します。
*	アスタリスク(*)は、その直前にある文字を検索します。
+	プラス(+)は、アスタリスクによく似ていますが、この場合は、その直前にある文字の 1 回以上の繰り返しを検索します。
?	疑問符(?)は、その直前の文字の 0 回または 1 回の繰り返しを検索します。
()	小カッコは、パターンの評価順序に影響し、一致する文字列を別の表現に置き換える場合に使用できるタグ付き記号として機能します。
[]	文字セットを囲む角カッコ([])は、そのカッコ内の任意の文字を検索対象とします。

アプリケーションのデバッグ

Event Rules Debugger では、対話型/バッチ・アプリケーションをデバッグできます。

▶ アプリケーションをデバッグするには

〈Application Development Tools(アプリケーション開発ツール)〉メニュー(GH902)から〈Debug Application(アプリケーションのデバッグ)〉を選択します。

1. デバッグするオブジェクトを選択します。
2. 表示するフォーム(対話型アプリケーションの場合)またはセクション(バッチ・アプリケーションの場合)とイベントを選択します。
3. ブレークポイントを設定するイベント・ルール行を選択します。
4. [Debug]メニューから[Breakpoint]を選択します。

   その行に、ブレークポイントを示す赤い丸が表示されます。

   ブレークポイントを削除するには、[Debug]メニューから[Breakpoint]を選択します。
   [Debug]メニューの各オプションは、オン、オフに切り替わります。
5. デバッガを終了せずにアイコン化(最小化)します。

6. 〈J.D. Edwards Menus（J.D. Edwards メニュー）〉または〈Object Librarian〉から、デバッグするアプリケーションを実行します。

   アプリケーションはブレークポイントに達すると停止し、〈Event Rules Debugger（イベント・ルール・デバッガ）〉が表示されます。

   実行停止しているときは、変数のビューを使用してランタイム構造体の値を検査し、修正できます。

7. [Debug]メニューから、次のオプションを 1 つ選択します。
   - Go
   - Stop
   - Step Over
   - Step Into

変数の検査または修正

アプリケーションのデバッグ時に、対話型アプリケーションまたはバッチ・アプリケーションが異常終了する行に達したら、異常終了しないように該当する変数を修正することができます。修正結果を保存し、アプリケーションを実行し直して、さらにデバッグする必要があるかどうかを調べます。

▶ 変数を修正するには

1. 〈Event Rules Debugger〉の〈Event Rules Variables（イベント・ルール変数）〉ウィンドウで変数をダブルクリックします。
2. 次のフィールドの情報を改訂します。
   - New Value（新規の値）

ジャスト・イン・タイム・デバッグ

〈Event Rules Debugger〉は、「ジャスト・イン・タイム・デバッグ」も使用します。ジャスト・イン・タイム・デバッグは、デバッグ情報を構築し、〈Event Rules Debugger〉を実行した後に、アプリケーションを実行します。

ランタイム・エンジンがイベント・ルール・オブジェクトやビジネス関数の呼出しで問題に遭遇すると、メッセージ・ボックスが表示され、デバッガの起動が可能になります。デバッガは起動すると最前面に設定され、処理できなかったイベント・ルール行の左に黄色い矢印が表示されます。

ブレークポイントの設定

ブレークポイントは、Dialog is Initialized などの開始イベントに少なくとも 1 つは設定してください。また、デバッガを停止させるイベントは、いずれも 1 行以上のイベント・ルール・コードがなくてはなりません。

Microsoft Visual C++によるビジネス関数のデバッグ

C 言語で記述されたビジネス関数をデバッグするには、Microsoft Visual C++を使用できます。B7331 以降の J.D. Edwards ERP ソフトウェアを使用している場合は、Microsoft Visual C++のバージョン 6.0 を使用してください。対話型アプリケーションやバッチ・アプリケーションに関連付けられているビジネス関数をデバッグできます。

対話型アプリケーションに関連付けられたビジネス関数のデバッグ

対話型アプリケーションやバッチ・アプリケーションに関連付けられているビジネス関数をデバッグできます。

▶ 対話型アプリケーションに関連付けられたビジネス関数をデバッグするには

1. J.D. Edwards 5 アプリケーションを終了します。

   Visual C++でデバッグするには、アプリケーションを終了する必要があります。

2. Visual C++を起動し、すべてのワークスペースが閉じていることを確認します。
3. [File]メニューから[Open]を選択します。
4. 〈List Files of Type〉で、実行ファイル(.exe)を選択します。
5. パス¥b7¥System¥bin32 の OEXPLORE.EXE を選択し、[OK]をクリックします。

   プロジェクト・ワークスペースが作成されます。

6. OEXPLORE.EXE プロジェクト・ヘッダーを選択します。
7. [Project(プロジェクト)]メニューから[Settings(設定)]を選択します。
8. [Debug]タブをクリックします。
9. 〈Category(カテゴリ・リスト)〉で、[Additional DLLs(追加する DLL)]を選択します。
10. [Browse]ボタンをクリックして、パス¥b7¥bin32 の CALLBSFN.dll または他の DLL を選択します。

    このパスは、パス・コードによって異なることがあります。

11. [OK]をクリックします。
12. デバッグするソースの.h ファイルと.c ファイルを選択し、[File]メニューから[Open]を選択します。
13. コードにブレークポイントを設定するには、[Edit]メニューから[Breakpoints]を選択します。

    次のメッセージが表示された場合は、[OK]をクリックします。

    cannot open *.pdb

    ブレークポイントが次の有効な行に移動したことを通知するメッセージが表示された場合は、ソース・コードとオブジェクトの不一致が存在し、ビジネス関数を再構築しなければならないことがあります。

14. [Build(ビルド)]メニューから[Start Debug(デバッグの開始)] - [Go(実行)]を選択します。

    J.D. Edwards ソフトウェアのサインオン・ウィンドウが表示されます。

15. 通常どおりアプリケーションにサインオンします。
16. アプリケーションを実行します。

    [Debug]でアプリケーションがビジネス関数に到達すると、デバッガが開くか、Visual C で C 言語コードが表示され、コードを 1 行ずつ検証できます。

ビジネス関数と C 言語ビジネス関数のイベント・ルールをデバッグする場合は、J.D. Edwards デバッガと Visual C++デバッガを併用できます。 J.D. Edwards ソフトウェアにログインするまで、上記の手順に従います。 J.D. Edwards にログインしたら、J.D. Edwards デバッガの手順に従います。C 言語コードがアクセスされると、プログラムの実行が停止します。その後は Visual C++を使用して、デバッグ作業を継続することができます。これは、問題を特定しようとしているときに、その問題が C 言語ビジネス関数にあるのか、そのビジネス関数を呼び出すアプリケーションにあるのかがわからない場合に有効です。

バッチ・アプリケーションに関連付けられたビジネス関数のデバッグ

バッチ・アプリケーションに関連付けられたビジネス関数はデバッグすることができます。

▶ バッチ・アプリケーションに関連付けられたビジネス関数をデバッグするには

1. J.D. Edwards アプリケーションを終了します。
2. Visual C++を起動し、すべてのワークスペースが閉じていることを確認します。
3. [File]メニューから[Open]を選択します。
4. [Files of Type(ファイル・タイプ)]フィールドで、実行可能ファイル(.exe)を指定します。
5. LaunchUBE.exe を検索して選択します。
6. 〈LaunchUBE(UBE の起動)〉で、使用するレポートとバージョンを参照します。
7. [Server Name(サーバー名)]フィールドで、レポートの実行場所を指定します。

   Local を選択するか、サーバーを指定します。

8. [Debug Level(デバッグ・レベル)]を設定します。
9. [Print Options(印刷オプション)]で、次のいずれかを選択します。
   - On Screen
   - To Printer
10. [Submit(投入)]をクリックして、レポートを投入します。

Visual C++デバッガの処理

イベント・ルールのスクリプト言語または C 言語コードで記述されたビジネス関数をデバッグするには、Microsoft 社の Visual C++デバッガを使用します。J.D. Edwards システム全体を Visual C++デバッガから実行することができます（Visual C++デバッガから oexplore.exe ファイルを起動することができます）。これにより、プログラマは CASE ツールによって生成されたアプリケーション・コードから、イベント・ルールで呼び出されるビジネス関数へと「寄り道」することができます。

プログラムを実行し、必要に応じて対話形式で停止したり再開するには、対話型デバッガを使用します。デバッグ時には、特定の変数とパラメータの値をチェックして、プログラムが正常に動作しているかどうかを確認することができます。また、ウォーク・スルーの実施や一行ずつコードを実行して、どのコードが実際に処理されるかを調べることもできます。

デバッグ・コマンドは[Debug]メニューに表示されます。ツールバーをカスタマイズしてデバッグ・ボタンを設定すると、メニューの代わりにそれを使用することができます。

Visual C++デバッガの便利な機能

Visual C++では、[Debug]メニューからさまざまな機能を利用できます。Visual C++デバッガを使用すると、問題をすばやく解決できます。

Go コマンド

[Debug]メニューから[Go]を選択すると、プログラムを実行することができます。プログラムは、ブレークポイントが設定されていなければ最後まで実行されます。

Step コマンド

Step コマンドは[Debug]メニューから選択できます。このコマンドを選択すると、現在のコード行が実行されます。そのコード行が実行されると、次に実行されるコード行に黄色い矢印カーソルが表示されます。

Step Into コマンド

Step Into コマンドには、[Debug]メニューからアクセスすることができます。このコマンドを使用するのは、現在のコード行に関数呼出しが含まれている場合です。デバッガはその関数まで処理を進めて、1 行ずつデバッグすることができます。関数が終了すると、デバッガは呼出し側ルーチンの関数呼出しの次のコード行に戻ります。分岐先関数のソース・コードが PC 上に存在しなければ、Step コマンドを使用した場合と同様に、そのコード行がスキップされます。

C 言語の標準関数に分岐すると、意図しない関数を実行することがあります。その場合は、それらの関数をスキップしてください。

ブレークポイントの設定

特定のコード行に達するまでプログラムを実行するには、ブレークポイントを使用します。ブレークポイントを設定すると、Go コマンドではそのコード行に到達するまでプログラムが実行されます。

ブレークポイントを設定するには、コード行のどこかにカーソルを合わせます。[Debug]メニューから[Breakpoints]を選択すると、ブレークポイントが設定されるコード行の左に赤い八角形が表示されます。プログラムを実行すると、そのブレークポイントまでのコード行がすべて実行されます。ブレークポイントの後に実行を再開するには、前述の[Breakpoint]、[Step]、[Step Into]、および[Go]コマンドを使用することができます。

[Watch]の使用

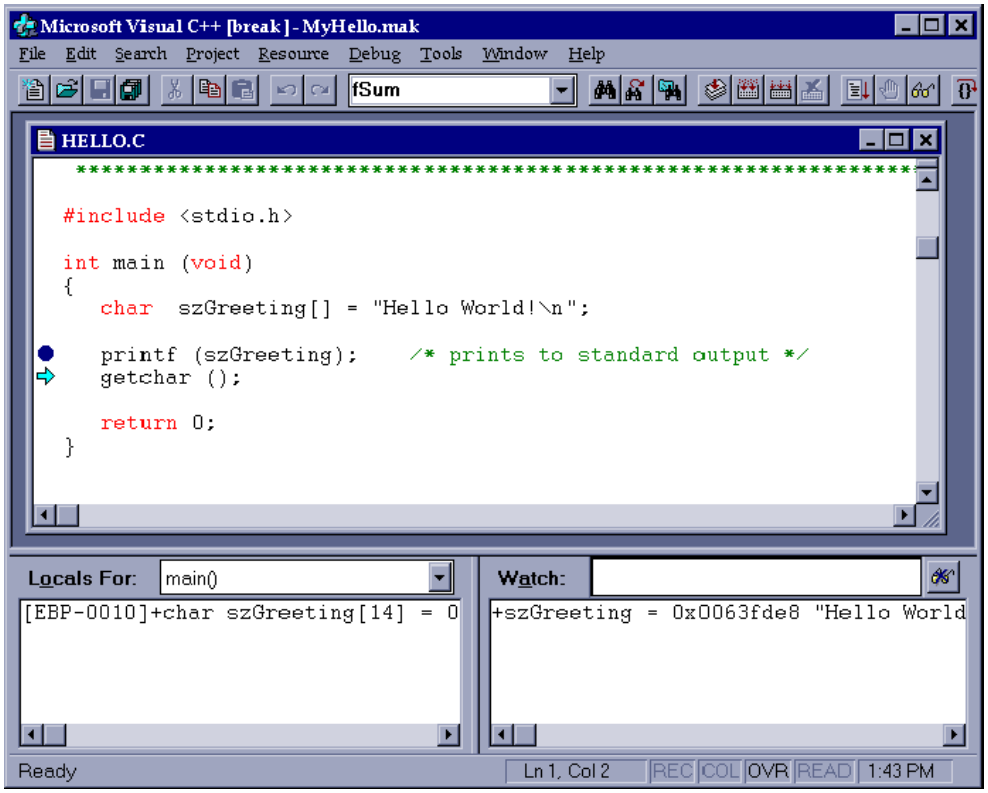
[Watch(ウォッチ)]を使用すると、変数にどのような値が設定されるかを検査することができます。[Watch]を使用するには、調べたい項目をクリックして〈Watch〉ウィンドウまでドラッグします。

〈Locals〉ウィンドウ

ある関数に対するすべてのローカル変数とパラメータは、データ型および値と共に〈Locals(ローカル)〉ウィンドウに表示されます。〈Locals〉ウィンドウのすべての項目値は、デバッグ時に必要に応じて変更することができます。これは、デバッグで無限ループに陥った場合に便利な機能です。

例:Visual C++デバッガによるデバッグ

次の例では、ブレークポイント、Watch、Locals、および Step コマンドが使われています。



- ブレークポイントは、printf を呼び出すコード行に設定されています。このコードの左側には、赤い八角形が表示されています。printf の呼出しは、Step コマンドで実行されています。
- 現在実行されている行の左側には、黄色い矢印が表示されています。この行には、getchar が呼び出されています。このコード行はまだ実行されていません。
- 〈Locals〉ウィンドウと〈Watch〉ウィンドウに szGreeting が表示されています。この 2 つのウィンドウをそれぞれスクロールし、行全体を確認できます。〈Watch〉ウィンドウに szGreeting の値の初期値を表示することができます。
- 〈Locals〉ウィンドウと〈Watch〉ウィンドウで変数の横にプラス記号がある場合は、そのデータを展開できることを示します。これは、複雑なデータ構造体の値を検査する場合に便利です。

環境のカスタマイズ

Visual C++デバッグ環境を修正することができます。エディタ、色、フォント、デバッガ、デフォルト・ディレクトリ、ワークスペースおよびヘルプをカスタマイズできます。

▶ 環境をカスタマイズするには

1. 〈Event Rules Debugger〉で、[Tools]メニューから[Options]を選択します。
2. 〈Options(オプション)〉で、必要に応じて環境をカスタマイズします。

デバッグ作業の方針

デバッグ作業を短時間で進めるには、いくつかの判断基準があります。問題の性格を見極めることから始めてください。

そのプログラムは異常終了するか？

プログラムが異常終了する場合は、未処理の例外が原因と考えられます。未処理の例外とは、メモリが適正に処理されていないことです。問題が同じ場所で発生する場合は、容易にトラッキングできます。コード上で問題と思われる箇所にブレークポイントを設定し、問題が確認されるまでプログラムを実行してください。

問題が C 言語コードにある場合は、コードをトレースすれば見つけることができます。

問題が J.D. Edwards で生成されるコードにある場合は、問題を見つけるのが難しくなります。デバッガは、役に立つエラー・メッセージを表示します。最も起こり得る問題は、欠落しているオブジェクトをどう処理するかです。呼び出されたビジネス関数が存在しなければ、ツールは関数が欠落していることを示すエラー・メッセージを表示します。たとえば、「Business function load failed – CALLBSFN.DLL(ビジネス関数のロードに失敗しました – CALLBSFN.DLL)」というメッセージが表示された場合は、ビジネス関数をリビルドするか、あるいはそれをチェックアウトし、〈BusBuild〉を使用してビジネス関数をビルドすることで、この問題を解決することができます。

ビジネス関数はいずれもより大きな DLL に組み込まれます。これらの中で最も汎用性の高いのが CALLBSFN.DLL です。アプリケーション固有の DLL は、一般に CALLBSFN にありません。たとえば、J.D. Edwards 会計ビジネス関数は CFIN.DLL を使用します。オブジェクト(ビジネス関数)は、ワークステーションに再度チェックアウトして、〈BusBuild〉によって CALLBSFN.DLL(または特定の DLL)に再度組み込まなければならないことがあります。

その他のオブジェクトが欠落している場合は、より突発的な形で終了します。メディア・オブジェクト(汎用テキスト)のオブジェクトは、いずれも適切に転送されなければなりません。存在しないアプリケーションへのロー・エグジットがアプリケーションに存在する場合は、プログラムにおける異常終了の直接的な原因になります。

他の種類のオブジェクトが欠落している場合にも、プログラムが異常終了する率が高くなり、あまり役に立ちません。アプリケーションのすべての部分を見直して、各オブジェクトが過不足なく存在し、正しく構築されているかどうかを確認する必要があります。一般的なエラーは、メディア・オブジェクトを忘れることが原因です。プログラムにまったく入れない場合は、オブジェクトが欠落している可能性が大です。

プログラムが同じ箇所で終了するかどうかを確認してください。メモリが使用後にリストアされないと、最終的にプログラムでメモリ不足が生じることがあります。そのような場合には、ワークステーションを再起動してメモリをリストアする必要があります。

アプリケーションに予期しないエラーが発生するか？

イベント・ルールからビジネス関数が呼び出されると、その処理はユーザーにはわかりません。この場合、次のような長所と短所があります。

- ユーザーは関数の機能を理解する必要がなく、機能しているかどうかだけを考えればよい(長所)
- いずれもエラーの原因になりかねない 51 個以上のパラメータを持つ関数を呼び出していることが珍しくない(短所)

このような問題には、次の 2 つの解決策があります。

- 関数のスペックを検証し、適切にフックしているか確認する。
- コードを 1 行ずつ調べて、問題個所を突き止める。

プログラムの出力に不適切なものがあるか？

プログラムからの不適切な出力は、コードのロジックに問題があることを示します。コードをトレースして問題を見つけてください。

問題の原因は他にあるのか？

少し時間をかけて、問題の原因がどこにあるかを考えてください。

たとえば、「NULL ポインタ・エラー」を例にとります。コードのどこかにメモリ・リークがありますが、どこにあるのでしょうか。リークの場所を特定するには、グリッド・ロー間で切り替えて、Row is Exited イベントを強制的に実行できます。これで問題が発生した場合は、それがグリッド処理中に発生したことになるので、リークの検索範囲が限定されます。

関数が正常に呼び出されるか？

呼び出される関数の先頭にブレークポイントを設定すると、その関数が呼び出され、入力が予想どおりであるかどうかを確認することができます。そのためには、ブレークポイントをビジネス関数の先頭または Grid Row is Fetched などのイベントの先頭に設定し、次の作業を実行します。

- すべてのコード・パスを 1 行ずつ実行する。

  デバッガを使用してコードをテストします。J.D. Edwards デバッガでも Visual C++デバッガでも、プログラマは処理中に値を変更することができます。この機能を使用して、コードのすべての行をトレースします。また、これによって、コードから発生することだけでなく、コードに発生し得るすべてのことをテストできます。このテスト方式では、エラーが正しく処理されているかどうかも明らかになります。たとえば、If 分岐をたどって不適切なデータを入力し、予期したエラーが発生するかどうかを調べられます。

- 問題の兆候ではなく原因を見つける。
- オブジェクトの転送/再インストールによって問題を調べる。

デバッグ・ログ

jde.ini ファイルには、デバッグ関連の複数のセクションがあります。最初にチェックが必要なステートメントは、[JDE_CG]セクションにあります。Target = Release 行を、次のように Target = Debug に修正してください。ERP のアプリケーション・インストールでは、jde.ini ファイルは Target = Release に設定された状態で納品されています。ビジネス関数をデバッグするには、jde.ini ファイルの設定を Target = Debug に変更し、デバッグするビジネス関数をリビルドする必要があります。

```
[JDE_CG]

STDLIBDIR=c:¥MSDEV¥LIB

TPLNAME=EXEFORM2

ERRNAME=CGERR

TARGET=Debug

INCLUDES=c:¥MSDEV¥INCLUDE;$(SYSTEM)¥INCLUDE;$(SYSTEM)¥INCLUDEV;
$(SYSTEM)¥CG;$(APP)¥INCLUDE;

LIBS=c:¥MSDEV¥LIB;$(SYSTEM)¥LIB32;$(SYSTEM)¥LIBV32;$(APP)¥LIB32;

MAKEDIR=c:¥MSDEV¥BIN

USER=DEMO
```

jde.ini ファイルの[DEBUG]セクション行を次のように Output = NONE から Output = FILE に変更すると、SQL ステートメントとイベントのログをファイルに出力できます。これは、問題が JDEDB API に関連する特定の事柄に限定されるので、役立つデバッグ・ツールになります。

```
[DEBUG]

TAMMULTIUSERON=0

Output=FILE

ServerLog=0

LEVEL=BSFN,EVENTS
```

```
DebugFile=c:¥jdedebug.log
JobFile=c:¥jde.log
Frequency=10000
RepTrace=0
```

メモリ頻度の設定によって、特定の 1000 番目の位置でメモリー・ヒープを検証することができます。ブレークポイントを設定し、そのコードを検査することができます。

参照

- トラブルシューティング用のその他のログについては、『サーバー&ワークステーション・アドミニストレーション』ガイドの「エンタープライズ・サーバー・プロセスのログ・ファイルとデバッグ・ログ・ファイルの表示」

SQL Log Tracing

〈SQL Log Tracing（SQL ログのトレーシング）〉を使用すると、生成されてデータベースに出力される正確な SQL ステートメントを確認するのに役立ちます。

► SQL Log tracing をオンにするには

1. ワークステーションの［コントロール パネル］で、［Administrative Tools（管理ツール）］を選んで［Data Sources (ODBC)（データソース(ODBC)）］を選択します。
2. 32 ビット ODBC ドライバを選択し、［Tracing（トレース）］タブをクリックします。
3. システムをトレースするタイミングを指定します。
4. ［Log file Path（ログファイル・パス）］でログの出力パスを指定します。

デバッグ・トレーシング

デバッグ・トレーシングは、データベース/ランタイム・イベント、ビジネス関数およびシステム関数をトレースするために使用します。デバッグ・トレーシングをオンにすると、トレーシングの結果をログ・ファイルに保管することができます。このデバッグ・ログは、SQL ステートメントがツールによってどのように作成されたかを調べるときにも有効です。

► デバッグ・トレーシングをオンに設定するには

1. jde.ini ファイルの［DEBUG］セクションで、「Output=NONE」を次のいずれかの値に変更します。

Output=FILE データベース・トレーシングとランタイム・トレーシングの両方を出力します。

Output=EXCFILE ランタイム・トレーシングのみを出力します。

2. デバッグの個別のニーズに合わせて、Level=の値を変更します。

   Level に設定できる値は、Level=行の後のコメント行に記述されています。値はどれを組み合わせてもかまいません。値はカンマで区切ります。Level の値は次のとおりです。

EVENTS イベントがいつ開始され、いつ終了したかを記録します。

BSFN ビジネス関数の入力時刻と値が戻される時刻が記録されます。

SF_? システム関数の実行時刻が記録されます。疑問符の箇所には、コントロールやメッセージ処理など特定のタイプのシステム関数を指定することができます。

システム関数トレーシング

jdedebug.log を検討すると、システム関数が設計時と同じカテゴリにグループ化されていることがわかります。ログには必要に応じて、システム関数が動作するプログラム要素の情報が含まれています。たとえば、グリッド・システム関数では多くの場合ロー番号が出力されます。また、コントロール・システム関数では、一般にコントロールのエイリアスとコントロール ID が出力されます。

次の例に、jde.ini オプションを設定した jdedebug.log の〈Journal Entry〉の実行出力を示します。

```
Output=EXCFILE

Level=EVENTS,BSFN,SF_GRID,SF_CONTROL
```

RT: >>>Beginning ER: Dialog is Initialized App: P0911 Form: W0911I
RT: <<<Finished ER: Dialog is Initialized App: P0911 Form: W0911I
RT: >>>Beginning ER: Post Dialog is Initialized App: P0911 Form: W0911I
RT: <<<Finished ER: Post Dialog is Initialized App: P0911 Form: W0911I
RT: >>>Beginning ER: Add Button Clicked App: P0911 Form: W0911I
RT: >>>Beginning ER: Dialog is Initialized App: P0911 Form: W0911A
RT: SYSFN: Hide Control < > 0
RT: SYSFN: Disable Control <ICU> 5258
RT: SYSFN: Hide Grid Column COL: 5
RT: SYSFN: Hide Control <ATDOW> 5392
RT: SYSFN: Hide Control <REMA> 5405
RT: SYSFN: Hide Control < > 5295
RT: SYSFN: Hide Control < > 5385
RT: SYSFN: Hide Control <DOC> 5297
RT: SYSFN: Hide Control <KCO> 5299
RT: SYSFN: Hide Grid Column COL: 7
RT: SYSFN: Hide Grid Column COL: 8
RT: SYSFN: Hide Grid Column COL: 9
RT: SYSFN: Hide Grid Column COL: 11
RT: BSFN: Calling : BatchOpenOnInitialization App: P0911 Form: W0911A
RT: BSFN: Returned 0: BatchOpenOnInitialization App: P0911 Form: W0911A
RT: BSFN: Calling : GetAuditInfo App: P0911 Form: W0911A
RT: BSFN: Returned 0: GetAuditInfo App: P0911 Form: W0911A
RT: <<<Finished ER: Dialog is Initialized App: P0911 Form: W0911A
RT: >>>Beginning ER: Clear Screen Before Add App: P0911 Form: W0911A
RT: SYSFN: Enable Control <PCTOW> 5390
RT: SYSFN: Hide Control <ATDOW> 5392
RT: SYSFN: Hide Grid Column COL: 5
RT: SYSFN: Hide Control <REMA> 5405
RT: SYSFN: Enable Control <CRCD> 5273
RT: SYSFN: Enable Control <LT> 5292
RT: SYSFN: Enable Control <LT> 5351
RT: SYSFN: Show Grid Column COL: 6
RT: SYSFN: Hide Grid Column COL: 12
RT: SYSFN: Show Control <LT> 5292
RT: SYSFN: Show Control <CRDC> 5271
RT: SYSFN: Hide Control <LT> 5351
RT: SYSFN: Hide Control <CCD0> 5358
RT: BSFN: Calling : GetLocalComputerId App: P0911 Form: W0911A
RT: BSFN: Returned 0: GetLocalComputerId App: P0911 Form: W0911A
RT: <<<Finished ER: Clear Screen Before Add App: P0911 Form: W0911A
RT: >>>Beginning ER: Post Dialog is Initialized App: P0911 Form: W0911A
RT: <<<Finished ER: Post Dialog is Initialized App: P0911 Form: W0911A
RT: >>>Beginning ER: Add Last Entry Row to Grid App: P0911 Form: W0911A
RT: <<<Finished ER: Add Last Entry Row to Grid App: P0911 Form: W0911A

Web アプリケーションの開発

J.D. Edwards 開発ツールでは、クライアントごとにフォームの設計を少しずつ変更する必要がある場合でも、Windows クライアントと Web クライアントの両方で使用するアプリケーションを容易に開発し、保守管理することができます。Windows、Java、または HTML アプリケーションを作成できます。フォームを設計するとき、〈Form Design Aid(フォーム設計ツール)〉によって、フォームを 3 つのモードのいずれかで表示することができます。設計チームは、どのクライアント・タイプでどのモードを使用するかを決める必要があります。たとえば Windows で生成するフォームでは Mode 1、Java で生成するフォームでは Mode 2、HTML で生成するフォームでは Mode 3 を使用できます。

フォームに配置するコントロールのほとんどは、これらのどのモードにも適合しますが、コントロールはいずれもこの 3 つのモードのそれぞれに合わせて、該当するフォーム上で表示/非表示(有効/無効)にすることができます。このようにして、(モードごとに 1 つとなる)アプリケーションの 3 つのバージョンを生成する元になるフォームを 1 セット開発することができます。J.D. Edwards Web アプリケーションをカスタマイズしたり、アプリケーションを新規に開発する場合は、〈Web Generation Facility (Web 生成機能)〉を使用して、それらから Java アプリケーションか HTML アプリケーションを生成する必要があります。

アプリケーションは通常モード 1、デフォルト・モードで作成します。このアプリケーションは、Windows に対して自動的に有効になります。アプリケーションを Java や HTML アプリケーションに変更する場合は、モード 2 またはモード 3 を使用できます。

Web ベースのアプリケーション(Java または HTML)はワークステーションで開発や変更が行えますが、アプリケーションをテストできる Web 対応のワークステーションが必要になります。J.D. Edwards ERP クライアントは、すべて Web 対応です。クライアントでは、以下のコンポーネントが必要になります。

- Internet Explorer 4.01 以上
- J.D. Edwards〈Web Generation Facility〉(ERP クライアントと共にインストール)
- J.D. Edwards Web サーバーへのネットワーク・アクセス

J.D. Edwards Web アプリケーションをカスタマイズする場合や新規アプリケーションを作成する場合は、それを再生成して指定の Java または HTML アプリケーションを作成する必要があります。

Windows アプリケーションには、Web で使用しやすいように変更を加えることができます。たとえば、Web アプリケーションでは、使用するフィールドが少なくなります。

注:

スタイル・シートでフォント・サイズを変更すると、コントロールの移動やコントロール・テキストの折返し、非表示など、フォームの外観を変更できます。

Java アプリケーションをテストするには、次のタスクを実行する必要があります。

1. 〈Form Design〉で、アプリケーションを保存します。
2. アプリケーションを Web サーバーに対して生成します。
3. アプリケーションをメニューに関連付けて、実行モードを指定します。
4. アプリケーションを実行します。

Web アプリケーションを作成するには、標準のアプリケーションを作成する場合と同じように J.D. Edwards 開発ツールを使用します。

J.D. Edwards 開発ツールを使用して独自のアプリケーションを作成し、そのアプリケーションを Web アプリケーションとして使用できるように設計する場合は、最適なパフォーマンスを確保するために次の事項を考慮してください。

5. 各データ要求とビジネス関数要求は、Web サーバーではなく、データ・サーバーやエンタープライズ・サーバーで実行されます。そのため、Web クライアントのパフォーマンスは、ネットワーク・トラフィックによる制限と共にネットワーク接続やモデム接続の速度によって制限されます。できれば、データ要求とビジネス関数要求を WAN やインターネット接続にとって最も効率的な形で制限するか、バンドルしてください。また、複数のテーブル入出力呼出しを直列にリンクしている（たとえば、各々の呼出しが直前の呼出しに依存している）場合は、イベント・ルール・ビジネス関数を使ってすべての呼出しを結合し、それらをエンタープライズ・サーバー上でコンパイルし、実行するようにしてください。

6. できれば、Web 対応アプリケーションはイベント・ルールを使用してロジックを実行するように設定してください。これにより、C 言語のビジネス関数やイベント・ルール・ビジネス関数を使用する場合よりもパフォーマンスを向上させることができます。なお、ビジネス関数を呼び出すと、ロジックは常に遠隔 J.D. Edwards エンタープライズ・サーバーで実行されます。ただし、前述したようにイベント・ルールが直列にリンクされた複数のテーブル入出力呼出しやビジネス関数呼出しを含んでいる場合には、これらのイベント・ルールは、イベント・ルール・ビジネス関数としてエンタープライズ・サーバーでより高速に動作します。

7. Web 対応アプリケーションは、適切な量（大きさ）のデータを要求するように設計してください。アプリケーションが各グリッド・ラインの前後でビジネス関数要求を実行すると、パフォーマンスは低下します。また、ビジネス関数をデータを要求する手段として使用することは避け、代わりにビジネス・ビューやテーブル入出力を使用するようにしてください。

8. QBE で検索条件を使用することの重要性をユーザーに教育してください。これは、一般的なインターネット接続で、データの転送がモデムの速度によって制約されることを考慮した場合に重要になります。たとえば、ユーザーは常に検索範囲を限定するようにする必要があります。データの取出しは、一度に 10 個のレコードしか取得できないように設定されているためです。Web 用のカスタム・アプリケーションを開発する場合は、QBE フィールドを必須項目として定義することを検討します。

〈Batch Versions – Web Version（バッチ・バージョン – Web バージョン）〉アプリケーション（P98305W）は、Web クライアント用として特別に作成されたバージョン・リスト・アプリケーションです。このアプリケーションを使用すると、アプリケーションの Web バージョンを参照することができます。

HTML クライアントについて

HTML クライアントは、以下の重要なイベントで自動的にポスト(リフレッシュ)します。

- Form startup(フォームの起動)
- Set focus on grid(フォーカスをグリッドに設定)(ER が存在するときのみ)
- Check box(チェックボックス)(ER が存在するときのみ)
- Radio button(ラジオ・ボタン)(ER が存在するときのみ)
- Bitmap clicked(ビットマップがクリックされた)
- URL clicked(URL がクリックされた)
- Tab is selected(タブが選択された)
- Node expanded in tree control(ツリー・コントロールでノードが拡大表示された)
- Drag-and-drop action occurs in tree control(ドラッグ&ドロップ操作がツリー・コントロールで発生した)

上記の重要なイベントの他に、クライアントは、グリッドの表示/非表示または有効/無効関数を含むイベントが発生したときにもページをリフレッシュします。これらのイベントの一覧を以下に示します。

編集コントロール用	Control is exited(コントロールが終了した)
	Control is exited and changed-inline(コントロールが終了し、変更された - インライン)
	Control is exited and changed-asynchronous(コントロールが終了し、変更された - 非同期)
グリッド・コントロール用	Column is exited(カラムが終了した)
	Column is exited and changed-inline(カラムが終了し、変更された - インライン)
	Column is exited and changed-asynchronous(カラムが終了し、変更された - 非同期)
親/子フォーム・グリッド用	Tree node level is changed(ツリー・ノードのレベルが変更された)
ツリー・コントロール用	Tree node is selected(ツリー・ノードが選択された)

イベントに適用されるこれらのポスト・ルールは、HTML クライアントによって使用されるデフォルトのポスト・モデルを形成します。HTML ポスト・モデルは、Web でよく使用される大量データ入力アプリケーションや照会アプリケーションで有用性に関する問題を生じることがあります。この問題は、アプリケーションがイベント発生時にグリッドを表示/非表示または有効/無効にして、HTML ページを強制的にリフレッシュするときに発生します。一般的に、このリフレッシュによって画面にちらつきが生じます。この継続時間は、ネットワーク・トラフィックやハードウェア、システム構成などによって異なります。そのため、HTML ポスト・モデルを一時変更した方がいい場合もあります。

フォームのプロパティ設定の[HTML Auto Refresh(HTML 自動リフレッシュ)]オプションを使用すると、イベントによるフォームの自動ポストをオフに設定することができます(デフォルトではオン)。また、1 つのイベントでポストをオンに設定するには、イベント・ルールで[HTMLPOST(HTML ポスト)]オプションを使用します(デフォルトでは、この属性をオフに設定します)。ポストを有効にして、無効を一時変更します。たとえば、フォームのポスティングをオフに設定しても、1 つのイベントの ER でオンに設定した場合は、その ER で[HTMLPOST]オプションをオンに設定したイベントを除く、フォームでのすべてのイベントでポストが抑制されます。

Windows クライアントと Java クライアントは、フォームの属性もイベントの属性も無視します。

はじめる前に

- J.D. Edwards HTML Web サーバーをインストールします。Knowledge Garden の Update Center(アップデート・センター)上で『J.D. Edwards HTML Web Server Installation(J.D. Edwards HTML Web サーバー・インストール・ガイド)』を参照してください。

参照

- HTML フォームの設計については、『Form Design Aid Guide(フォーム設計ツール)』ガイドの「Designing Forms Using Multiple Modes(複数モードを使用したフォームの設計)」
- 各種モードの使用と、各種モードでのフォーム上のコントロールのカスタマイズについては、『Form Design Aid Guide(フォーム設計ツール)』ガイドの「Form Controls(フォーム・コントロール」

パフォーマンス

このセクションには、アプリケーションのパフォーマンスの向上に役立つヒントとツールが記載されています。

トリガー

データ辞書トリガーは、フォーマット設定と妥当性検査の両方で使用することができます。トリガーを使用してデータ値のフォーマットを設定したり編集したりすると、時間がかかります。J.D. Edwards ソフトウェアで生じるフォーマット設定は、ほとんどが日付フィールドや計算フィールドに関連します。これらのフィールドは、編集やフォーマット設定で非常に大きなオーバーヘッドを必要とします。すべてのトリガーができる限り効率的にコーディングされているかを確認してください。

一時変更

データ辞書一時変更を使用すると、フォーム・コントロールやグリッド・カラムやレポート・フィールドの設計時でのデータ辞書特性を一時変更することができます。これは、アプリケーションの起動時にオーバーヘッドが増加する原因になります。一時変更を検証し、適用するために特別な処理が必要になるからです。適用される一時変更は、ランタイム処理時のパフォーマンスにプラスの影響を与えることもあれば、マイナスの影響を及ぼすこともあります。たとえば、一時変更機能を使用して一定の検査機能を無効にすると、実行時のオーバーヘッドが軽減されるので、パフォーマンスが向上します。逆に、編集/フォーマット設定トリガーや自動採番などに一時変更を行うと、実行時のオーバーヘッドが増加し、パフォーマンスが低下します。

妥当性検査

妥当性検査は、コントロールやカラムやフィールドに関連付けられたデータ辞書項目に基づきます。一時変更は、妥当性検査にも影響する場合があります。妥当性検査の負荷は、トリガーを持つデータ項目で最大になります。ユーザー定義コードの妥当性検査でも入出力でデータ値を検査する必要があります。通常の妥当性検査の負荷が最大になるのは、処理に特別なロジックを必要とする J.D. Edwards ソフトウェア固有のデータ・タイプの場合です。

テーブル設計のパフォーマンス

SELECT ステートメントを使用するときには注意する必要があります。テーブルでは、既存のインデックスを使用するようにしてください。新規のインデックスを作成するよりは、キーを追加使用する方が適しています。インデックスを追加すると、オーバーヘッドが増えます。

パーシャル・キーは、順次フェッチ（取出し）か、またはさらにわずかなレコードを参照する場合にのみ使用します。

テーブルを開いてからフェッチ・キーを開くと、ポインタが壊れるので、代わりに単一フェッチ]を実行してください。他のほぼどのデータベース API もポインタを破壊します。

テーブルを初めて開くときには、パフォーマンスに大きく影響します。

フェッチ・マッチング・キーは≧キーを使用するので、必要以上のものを選択する可能性があります。要求に見合った JDB API を使用してください。

インデックスに関する考慮事項

適切なインデックスを追加すると、データベースからデータを選択したり、取り出したりするときのパフォーマンスがほぼ確実に向上します。ただし、インデックスを追加するたびに、データベースにレコードを追加、更新、削除するときのメンテナンスのオーバーヘッドが増大します。テーブルに対して新規のデータベース・インデックスを追加するかどうかを判断する際には、この 2 つの要素を考慮する必要があります。

J.D. Edwards ソフトウェアは、データベースに効率的にアクセスするように設計されています。データベース・テーブルとインデックスの最適な設計を定めるときには、正規化などを含め、一般的なデータベース設計時の考慮事項を加味してください。

結合フィールドは、2 つのテーブルのキーにしてください。

ロー設定サイズに対する制限事項

ローサイズの下限の分母は、SQLSERVER データベースで指定されているスペックとなります。

SQL Server 6.5 では、1 つのデータベースに 20 億ものテーブルを設定でき、1 つのテーブルに 250 のカラムを設定できます。1 ローあたりの最大バイト数は 1962 バイトです。varchar または varbinary カラムの総定義幅が 1962 バイトを超えるテーブルを作成した場合、テーブルは作成されますが、警告メッセージも表示されます。

このようなローに 1962 バイトを超えるデータを挿入したり、ローを更新した結果、ローの合計サイズが 1962 バイトを超えてしまうと、エラー・メッセージが表示され、ステートメントが失敗します。

World Software との共存に関する考慮事項

J.D. Edwards ERP ソフトウェアのテーブルが AS/400 のテーブルと一致することを確認してください。構造の異なるデータがあると問題が発生します。

共存環境で独自の変更を予定している場合、WorldSoftware RPG プログラムは、WorldSoftware によって読み取られるテーブルの構造を読み取ることができる必要があります。これは、WorldSoftware がアクセスするテーブルは、AS/400 上で異種データベース（Access、Oracle、SQL Server）を使用している J.D. Edwards ERP ソフトウェアではなく、WorldSoftware データベース構造で最初に作成する必要があることを意味します。最初に WorldSoftware 環境を設定した上で、J.D. Edwards ERP を設定してください。

AS/400 上と J.D. Edwards ERP ソフトウェア上で、日時データの変換が同一に設定されていることを確認してください。

各種データベースのインデックス制限

ここでは、次のデータベースのインデックス制限について説明します。

- Access32
- SQLSERVER
- OS/400 用 DB2
- Oracle

Access32

Access 32 には、次のインデックス制限が適用されます。

- テーブル 1 つあたりの最大インデックス数は 32 です。
- インデックスの最大フィールド数は 10 です。
- レコードの最大フィールド数は 255 です。

SQLSERVER

SQLSERVER には、次のインデックス制限が適用されます。

- テーブル 1 つあたりの最大クラスタ・インデックス数は 1 つです。
- テーブル 1 つあたりの最大非クラスタ・インデックス数は 249 です。
- 複合インデックスの最大カラム数は 16 です。

OS/400 用 DB2

OS/400 用 DB2 には、次のインデックス制限が適用されます。

- インデックス名の最大識別子数は 10 です。
- インデックスの最大サイズは 1 テラバイトです。
- インデックス・キーの最大長は 2,000 です。
- インデックス・キー1 つあたりの最大カラム数は 120 です。
- テーブル 1 つあたりの最大インデックス数は約 4,000 です。

Oracle

Oracle には、次のインデックス制限が適用されます。

- インデックスの最大長は、NULL 値を格納できるキーの数を差し引いて 254 バイトです。
- インデックス 1 つあたりの最大カラム数は 16 か、または最大キー長の 900 です。

例:テーブル・インデックス

F32944 テーブルでは、インデックスは次のように定義されます。

```
{
    MATH_NUMERIC ktkit;              /* 0 to 48 */
    char     ktmc01[251];            /* 49 to 299 */
    char     ktmc02[[252];           /* 300 to 550 */
    char     ktmc03[251];            /* 551 to 801 */
    char     ktmc04[251];            /* 802 to 1052 */
    MATH_NUMERIC ktseqn;             /* 1053 TO 1101 */
} KEY1_F32944, FAR *LPKEY1_F32944;
#define ID_F32944_KIT_CONFIGURED_STRING_ID_B 2L
```

この場合のキーの長さは 1101(>900)なので、これは許容されません。

キー・カラムの違反

種々のカラムでインデックスを定義するときには、一方がプライマリ固有インデックスでありうる 2 つのインデックスを同じカラムで定義しないでください。

例:キー・カラムの違反

この例では、与信および回収期間パターン・テーブル(F03B08)のインデックスが次のように定義されています。

```
#define ID_F03B08_COMPANY_FISCAL_YEAR 1L /* PRIMARY &
UNIQUE */
typedef struct
{
   char rdco[6];          /* 0 to 5 (ASC)*/
   MATH_NUMERIC rdctry;      /* 6 to 54 (DESC) */
   MATH_NUMERIC rdfy;         /* 55 to 103 (DESC) */
   MATH_NUMERIC rdpn;         /* 104 to 152 (DESC) */
} KEY1_F03B08, FAR *LPKEY1_F03B08;
#define ID_F03B08_COMPANY_FISCAL_YEAR_(ASCEND) 2L /*
NONUNIQUE */
typedef struct
{
   char rdco[6];          /* 153 to 158 (ASC) */
   MATH_NUMERIC rdctry;      /* 159 to 207 (ASC) */
   MATH_NUMERIC rdfy;         /* 208 to 256 (ASC) */
   MATH_NUMERIC rdpn;         /* 257 to 305 (ASC) */
} KEY2_F03B08, FAR *LPKEY2_F03B08;
```

これは、同じカラムに 2 つのインデックスが定義されているためキー・カラム違反であり、許されません。

スペック・ファイルの破損

スペック・ファイルが破損した場合は、ソースであるデータベースでテーブルを再作成してください。

テーブル入出力オブジェクト

テーブル入出力オブジェクトには、次のガイドラインが適用されます。

- テーブル入出力のヘッダー・サイズは 206 バイトです。
- テーブル入出力マッピング項目のサイズは 181 バイトです。
- サイズが 32K に適合するテーブル入出力マッピング項目の最大数は、(32768-206)/181 から約 180 項目です。

リテラル値のスペースを確保するために、130 要素以下のマッピング・サイズを使用してください。この 130 要素は、テーブル入出力ステートメントのマッピング数に関係します。テーブルには、180 個を超えるカラムが存在することがありますが、180 未満のカラムを問題なくマッピングできます。マッピング数が 180 を超えるテーブル入出力を使用しなければならない場合は、特定の条件に合わせて複数のテーブル入出力ステートメントを作成することをお勧めします。

ビジネス・ビューのパフォーマンス

ここでは、単一テーブルまたは複数テーブルのビジネス・ビューの使用、ビジネス・ビュー、合併または結合ビジネス・ビューに含まれるフィールド数の制限がパフォーマンスに及ぼす影響について説明します。

単一テーブルのビジネス・ビューと複数テーブルのビジネス・ビューのどちらを使用すべきか？

アプリケーションは通常、業務を遂行する上で複数のデータベース・テーブルからのデータにアクセスします。これには次の 2 つの方法があります。

- 複数テーブルのビジネス・ビューを使用して、すべての関連データ・フィールドにアクセスする。
- 単一テーブルのビジネス・ビューを使用して、プライマリ・テーブルからデータ・フィールドにアクセスし、ビジネス関数かテーブル入出力を使用して、セカンダリ・テーブルからデータ・フィールドにアクセスする。

どちらの方法がよいかは、一般には、実行するデータベース入出力操作の数を調べることで適否を判断することができます。データ・ソース間結合を使用する結合ビジネス・ビューは、パフォーマンスの低下をもたらします。

セカンダリ・テーブルがマスター・テーブルの場合には、一般に単一テーブルのビジネス・ビューを使用する方が適切です。これは、データベースのキャッシュを活用するからです。たとえば、プライマリ・テーブルに会社番号があり、それに関連する会社名がセカンダリ・テーブルである会社マスター・ファイルに格納されているとします。実際には、プライマリ・テーブルのいくつかのレコードで同じ会社マスター・レコードを取り出すことがありますが、そのような場合には、一般にビジネス関数やテーブル入出力を使用して会社名を明示的に取り出す方をお勧めします。

J.D. Edwards ソフトウェアは、特にユーザー定義コードのフェッチに最適化されています。ユーザー定義コード・テーブルは、複数テーブルのビジネス・ビューに含めないでください。

ビジネス・ビューでフィールド数を制限すべきか？

不要なフィールドを含んでいる既存のビジネス・ビューを使用するか、または目的のフィールドだけを含む新規のビジネス・ビューを作成するかの選択が必要になる場合があります。余分なフィールドがあるとサーバーとワークステーションの両方で負荷が増える原因になりますが、パフォーマンスには通常さほど影響しません。更新されるテーブルを含んでいる場合、J.D. Edwards ソフトウェアは、テーブルのすべてのフィールドを、それがビジネス・ビューに含まれていない場合でも自動的に処理することができます。

実行時のパフォーマンスを向上するためには、データ構造体のフィールド数を最小にする方が、ビジネス・ビューのフィールド数を最小にするよりも有効です。特定のニーズを満たし、パフォーマンスを低下させないビジネス・ビューを新規に作成することができます。

合　併

合併は、SQL SELECT ステートメントの中で最も負荷のかかるデータベース操作です。できるだけ合併を使用しないようにお勧めします。合併が適しているのは、アプリケーションと仕訳転記などのプロセスがテーブルの同じ部分を使用するものの、そのタイミングが異なっている場合です。

参照

- システムで使用される各種テーブルについては、『一般会計』ガイドの「転記処理」

結合ビジネス・ビューとテーブル入出力

結合ビジネス・ビューを使用するのは、複数の主要テーブルを読み取って更新する場合です。テーブル入出力を使用するのは、テーブルの読取りまたは更新が単に他の処理による意図しない結果として発生する場合です。

グリッド・カラムの数をアプリケーションの基本的な用途に必要な最小限に抑えてアプリケーションを作成することを検討してください。関連ビジネス・ビューにカラムを追加すれば、アプリケーションの基本機能を手軽に補完することができます。したがって、ビジネス・ビューにカラムを追加する方がグリッドにカラムを追加するよりも適切です。

データ構造体のパフォーマンス

ここでは、データ構造体とデータ構造体オブジェクトのフィールド数の制限がパフォーマンスに及ぼす影響について説明します。

データ構造体のフィールド数は制限すべきか？

データ構造体のフィールド数を最小にしてみる必要があるかどうかについては、特にコンフィギュラブル・ネットワーク・コンピューティング(CNC)機能によりサーバーとワークステーション間でデータ構造体を受け渡さなければならないとき、パフォーマンスの測定値が目安になります。

データ構造体オブジェクト

データ構造体オブジェクトには、次のガイドラインが適用されます。

- データ構造体のヘッダー・サイズは 237 バイトです。
- データ構造体の項目サイズは 72 バイトです。
- 32K にフィットするデータ構造体項目の最大数は、(32768-237)/72 から 450 要素です。

リテラル値のためのスペースを確保するために、構造体のサイズは 350 要素以下に制限してください。100 要素を超えるデータ構造体の作成については再検討する必要があります。

データ選択およびデータ順序設定

次の 2 つのシステム関数を Initialize Section イベントで使用して、そのセクションでのデータ選択を条件付きで変更します。

- Set User Selection
- Set Selection Append Flag

Initialize Section イベントは、一般にレポートの第 1 レベル 1 セクションで使用し、レポートのデータ構造体を介して受け渡される値に基づいてデータを条件付きで選択します。

たとえば〈Bill of Material Inquiry（部品表の照会）〉は、〈Bill of Material〉レポートを呼び出して、[Parent Item（親品目）]、[Parent Branch（親事業所）]、[Type of Bill（部品表タイプ）]、[Batch Quantity（バッチ数量）]の 4 つのパラメータを受け渡します。〈Bill of Material〉レポートの第 1 レベル 1 セクションの Initialize Section イベントで、データ構造体のレポート値がチェックされます。これらがブランクでなければ、これらの選択条件を追加するために、システム関数の Set User Selection が使用されます。

フォーム設計

すべてのフォーム・タイプ間でパフォーマンスを向上させるために、次のガイドラインに従ってください。

- グリッドのカラム数をアプリケーションによる必要最小限の数に制限します。
- ビジネス・ビューのカラム数をアプリケーションによる必要最小限の数に留めます。
- フォーム・コントロールは、表示/非表示にかかわらず、アプリケーションによる必要最小限の数に制限します。
- 非表示のフォーム・コントロールの代わりに、イベント・ルール変数をワーク・フィールドとして使用します。
- 編集やデフォルト値などの不要なフォーム・コントロールとグリッド・コントロールでは、表示/非表示にかかわらず、データ辞書機能を無効にします。
- 各グリッド・ローで実行される入出力の量は、アプリケーションで必要な最小限の量に制限します。たとえば、関連記述などはできる限り省きます。
- 不要なイベント・ルールの処理をスキップできるときには、常にシステム関数の Stop Processing を使用します。
- 設計では、キャッシュとリンク・リストとワークテーブルなど、その時点で使用できる最も効率的な一時データの保管法を考慮してください。

検索/表示

次の標準に従うと、検索/表示フォームのパフォーマンスが向上します。

- QBE 割当ては使用しないでください。
- グリッドのソート順序は、J.D. Edwards ERP で定義されているインデックスと AS/400 で定義されている論理ファイルの両方と、部分的または全面的に一致させます。論理ファイルとインデックスには、少なくともグリッド・ソートのすべてのフィールドを格納する必要があり、またグリッド・ソートで選択されたフィールドは、論理ファイル/フィールド・インデックスと同じ順序にします。インデックスや論理ファイルには、グリッド・ソートに含まれない追加フィールドが存在する場合があります。たとえば部分的な一致で、グリッド・ソートが KIT, MMCU である場合、論理ファイルとインデックスには、KIT, MMCU, TBM, BQTY が含まれていることがあります。

見出し詳細と見出しなし詳細

次の標準に従うと、見出し詳細フォームと見出しなし詳細フォームのパフォーマンスが向上します。

- 最大のパフォーマンスを得るために、グリッド・ソートとビジネス・ビューのテーブルのインデックスを一致させます。AS/400 テーブルと J.D. Edwards ERP テーブルの両方にインデックスを存在させます。
- グリッドのソート順序は、J.D. Edwards ERP で定義されているインデックスと AS/400 で定義されている論理ファイルの両方と、部分的または全面的に一致させます。論理ファイルとインデックスには、少なくともグリッド・ソートのすべてのフィールドを格納する必要があり、またグリッド・ソートで選択されたフィールドは、論理ファイル/フィールド・インデックスと同じ順序にします。インデックスや論理ファイルには、グリッド・ソートに含まれない追加フィールドが存在する場合があります。たとえば部分的な一致で、グリッド・ソートが KIT, MMCU である場合、論理ファイルとインデックスには、KIT, MMCU, TBM, BQTY が含まれていることがあります。

バッチ・アプリケーションのパフォーマンス

ここでは、レベル区切りの設定がパフォーマンスに及ぼす影響について説明します。また、パフォーマンスの原因についても説明します。

パフォーマンス

バッチ処理のサーバー側パフォーマンスが極端に低下し、その原因を判断できない場合は、jde.ini ファイルで RequestedService が適切に設定されているかどうかを確認するように、システム管理者に依頼してください。

イベント・ルールのパフォーマンス

ロジックを作成するときには、ローが変更されていなければスキップしてください。すべてのローで変更をチェックしないでください。

マスター・ビジネス関数の end docs を Post Button Clicked イベントに設定します。

非同期処理では、Post Button Clicked イベントを[OK]と[Cancel]にのみ設定してください。これにより、ビジネス関数が動作し続けているときにフォームが終了しなくなります。ビジネス関数が動作し続けているときにフォームが終了すると、ビジネス関数は値を返したり、エラーを処理することができません。

反復処理にはシステム関数を使用してください。

システム関数は、たとえば、グリッド・ラインの書き出しなどで、アクセスできない情報を取得するために使用することもできます。

FetchKeyed は、Clear Selection、Select Keyed および Fetch から構成されます。FetchKeyed は、キーが変化しない場合は、使用しないでください。

キーが変化しない場合は、パフォーマンスを向上するために、Select を使用してから Fetch Next をループで使用してください。ただし、レコードを逐次読み取る必要がない場合に限ります。

フィールドは、Dialog is Initialized イベントで表示/非表示にします。ロジックは Post Dialog is Initialized イベントに置きます。

IF ステートメントは常にコーディングしてください。

暗示ではなく明示的な比較を使用してください。この場合、コードは増えますが動作は高速になります。

ループのコード行が実際に必要かどうか確認してください。毎回実行する必要がない行は、ループに配置しないでください。

コードでのリターン回数を減らし、exit は 1 つだけにしてください。

一時的な値は、イベント・ルール変数を使用して保管します。一時的な値を保管するために、隠しコントロールやグリッド・カラムを使用しないでください。

ワークテーブルの代用としての jdeCache

J.D. Edwards ソフトウェアの旧バージョンでは、開発者は作業中の配列を含めるために一時的なローカル・テーブルを使用していました。この方法は、現在は使用されません。〈jdeCache〉ルーチンは、これと同じ機能を提供しており、より優れたパフォーマンスを実現できます。

複数のビジネス関数が同じキャッシュにアクセスするときは、関連する関数を同じソース・メンバーに配置してください。たとえば、EditGridLine 関数はキャッシュにレコードを追加し、それに対応する EndDocument 関数は、キャッシュされたレコードを取り出してデータベースに挿入します。これらのビジネス関数が異なるソース・メンバに配置されていると、CNC 構成の変化が原因でアプリケーションが失敗することがあります。

ビジネス関数での JDB API 呼出しまたはイベント・ルール・コードでのテーブル入出力コマンドの選択

データベースを管理するためにビジネス関数で JDB API 呼出しを使用すべきか、またはイベント・ルール・コードでテーブル入出力コマンドを使用すべきかは、コンテキストに従って選択してください。どちらを選択しても、パフォーマンスには影響しません。たとえば、イベント・ルール・コードがデータベースにアクセスしなければならない場合は、ビジネス関数を記述して呼び出す代わりにイベント・ルールでのテーブル入出力を使用してください。

JDBFetchNext の使用と複数の JDBFetch ステートメントの使用

データベースから複数の関連データベース・レコードを読み取る必要がある場合は、パフォーマンス上の観点から、一度に 1 つずつレコードを選択し、JDBFetchNext を使用して関連の各レコードをループすることをお勧めします。代わりに JDBFetch を繰り返し呼び出すと、通常はパフォーマンスが明らかに低下していきます。

ビジネス関数のパフォーマンス

ビジネス関数では、データ辞書編集を使用することができます。

コードに妥当性検査を設定し、その妥当性検査がマスター・ビジネス関数の一部でもある場合、ツールは妥当性検査が完了していることを認識し、妥当性検査を再度実行する代わりに復帰して、それが既に完了していることを伝えます。

グリッドでカスタム選択を実行すると、パフォーマンスが大幅に低下します。ユーザーによるグリッドの順序変更を制限できます。ビジネスでグリッドの順序変更が必要になる場合は、ソート順序と一致するテーブル・インデックスが存在することを確認してください。

一部をメモリに保管するには、ビジネス関数キャッシュを使用してください。

リンクされたリストを使用せず、代わりにキャッシュを使用してください。

構造体を作成し、項目を追加する代わりにポインタを受け渡してください。

変数が必要な場合にのみ使用されることを確認してください。

Static 変数を使用しないでください。代わりに単一のレコード・キャッシュを使用してください。Static 変数を使用することがあるとすれば、それは何らかのデータの一部を繰り返し取得し、それらを集計する場合が考えられます。

jdeAlloc は実際にスペースを割り当てるので、一般的には使用しないでください。ただし、複数呼出しの保管領域を確保する場合は、jdeAlloc を使用してもかまいません。

たとえば、サーバーとワークステーションのコンポーネントに分割される関数を使用し、親/子情報が存在するとします。クライアントは、親/子関係に似た情報を要求します。サーバーは、その情報の所在を認識する必要があります。この場合、jdeAlloc を使用して情報を逐次保管することができます。

jdeCallObject – 関数を形成すると、ソースに 1 つの.c ファイルが置かれます。そこに複数の関数が存在する場合には、主な関数をサーバーで実行すると、すべての関数がサーバー上で動作します。ただし、関数は、ワークステーションで実行するものとサーバーで実行するものに分けられている場合があります。複数の関数を使用する場合は、すべて全面的に依存しており、同じ場所で実行することを確認してください。jdeCallObject は DLL とプラットフォームを横断します。

メモリの割当て

割当て済みのメモリが不要になった時点で解放されないと、メモリ・リークが発生します。このため、アプリケーションの実行中に継続的にパフォーマンスが低下していく可能性があります。jdeCache はメモリを割り当てることを認識しておく必要があります。メモリ・リークを防ぐために、マスター・ビジネス関数の BeginDoc で作成されたキャッシュは、対応する EndDoc 関数で解消してください。

予期しないメモリ・リークが頻繁に発生する原因の 1 つは、エラーや他の条件を処理するときに割当済みメモリを解放できないことであると考えられます。割当済みのメモリを解放できないと、ビジネス関数で代替実行パスが使用されることになります。

ビジネス関数でメモリ・リークが発生していないかどうかは jdedebug.log を調べて確認します。メモリ・リークの検出には、他社製のツールも使用できます。

テーブルの開閉を一致させる

同じビジネス関数のそれぞれの jdbOpen および対応する jdbClose を、すべての実行パスで常に一致させてください。ビジネス関数でテーブルの jdbOpen と jdbClose が 1 対 1 になっていないと、パフォーマンスが大幅に低下します。

エラー・メッセージ処理のパフォーマンス

エラー・メッセージは、データ辞書でエラーまたは警告として分類することができます。エラーは赤色の停止符号として、警告は黄色の警告記号として表示されます。妥当性検査は、データ辞書を使って自動的に行われます。イベント・ルールにより検証されるエラー・メッセージを作成することができます。その場合は、次のことを確認してください。

- イベント・ルールを使用して、これらのエラー・メッセージで手作業でチェックします。
- エラーが発生したフィールドとエラー・メッセージ番号を指定します。
- 既に表示されているエラーが再表示されないように、イベント・ルールには IF ステートメントを組み込みます。

注:

エラー・メッセージがデータ辞書に存在してもイベント・ルールに存在しない場合は、エラーがそのつど表示されます。

トランザクション処理のパフォーマンス

トランザクション処理を使用するアプリケーションを設計したときには、できるだけ狭い範囲を維持し、トランザクションを開始してからコミットメントやロール・バックを開始するまでの時間を最短にできるようにしてください。トランザクションの一部として組み込まれているレコードを更新すると、そのレコードは、トランザクションがコミットされるまでロックされます。トランザクションのどの部分が失敗しても、トランザクション全体がロールバックされます。

J.D. Edwards 修正ルール

高機能かつ柔軟性に優れている J.D. Edwards 開発ツールを使用すると、ビジネス・ソリューションやアプリケーションの特定部分を、カスタム修正を行わずにカスタマイズできます。J.D. Edwards では、これを「modless modifications（修正なき修正）」と呼んでいます。この修正は、開発者の支援を受けずに容易に実行できます。次の内容について、モードレス修正を実行することができます。

- ユーザー一時変更
- ユーザー定義コード
- メニューの改訂
- すべてのテキスト
- 処理オプション値
- データ辞書属性
- ワークフロー・プロセス

モードレス修正の柔軟性によって作業効率が向上すると共に、次のような機能を含む利点が得られます。

- Microsoft Excel スプレッドシートなどの別のアプリケーションに、グリッド・レコードをエクスポートする。
- 異なるカラムに基づいてグリッドのソート順序を変更できる。
- グリッドのフォントおよび色を変更できる。
- 処理オプションを使って主要な機能を制御する。

J.D. Edwards ではできるだけ柔軟で耐久性に優れたツールを目指していますが、実際に修正が必要になった場合も考慮しています。ユーザーがソフトウェア修正を行った場合でも、次のリリース・レベルへのシームレス、かつ予測可能なアップグレードを行うモードレス修正を可能にするために、ルールを設けています。

スムーズにアップグレードするには、独自の修正を加える前に、アップグレードのための準備が必要です。適切な修正プランを立てれば、最小限の作業でアップグレードすることができます。このため、判断事項は最小限で済み、アップグレード所要時間の短縮によりコストが削減されます。

J.D. Edwards ソフトウェアでは、サーバーにチェックインされたカスタム修正がすべてトラッキングできます。この機能を使用すると、マージする前に〈Object Librarian Modifications（オブジェクト・ライブラリアン修正）〉レポート(R9840D)を実行できるので、修正されたオブジェクト一覧を確認することができます。

J.D. Edwards ソフトウェアは、コントロール・テーブル（メニュー、ユーザー定義コード、バージョン、データ辞書など）と、住所録マスター(F0101)や受注見出しテーブル(F4201)などのトランザクション・テーブルで構成されています。トランザクション・ファイルにはビジネス・データが保管されており、コントロール・テーブルは修正可能なデータが入っている状態で出荷されます。

これらのテーブルは、どちらもアップグレード時に自動マージ処理に送られます。この場合、コントロール・テーブルはアップグレードされた J.D. Edwards データとマージされ、トランザクション・ファイルは社内の既存データを維持しつつ、新しいスペックのレコードとして変換されます。オブジェクト・スペック(ビジネス・ビュー、テーブル、データ構造体、処理オプション、イベント・ルール、およびアプリケーションなど)をマージする場合は、次の項に記載されている規則に基づき、それぞれマージまたは上書きされることになります。

アップグレード時に保持される修正と置換される修正

カスタムのソフトウェア修正を行う必要がでてきた場合、次の J.D. Edwards ソフトウェア一般修正規則に従うと、スムーズで予測可能なアップグレード処理が保証されます。ここでは、アップグレード処理によって保持される修正と置換される修正について説明します。

- 「保持」とは、アップグレード時に、J.D. Edwards ソフトウェアによってこれらの修正がアップグレード用に出荷された新しい J.D. Edwards アプリケーションと自動的にマージされることを意味します。スペックが J.D. Edwards のスペックと矛盾する場合、アップグレード処理では独自スペックが使用されます。両者に明らかな矛盾がなければ、アップグレード処理では 2 つのスペックがマージされます。
- 「置換」は、この種の修正がアップグレード時にマージされず、置換されることを意味します。アップグレード処理の完了後に、あらためて修正を行う必要があります。

修正したオブジェクトがアップグレード処理で識別される前に、〈Object Librarian Modifications Report〉レポート(R9840D)を実行することができます。

修正用一般規則

J.D. Edwards ソフトウェア・オブジェクトには、次の一般修正規則が適用されます。

- 新しいオブジェクトを追加するときには、システム・コード 55～59 を使用します。J.D. Edwards ソフトウェアは、異なるアプリケーションとグループの分類を可能にする予約済みのシステム・コードを使用します。システム・コード 55～59 を使用してカスタム修正を行うと、J.D. Edwards ソフトウェアは修正内容を J.D. Edwards アプリケーションでオーバーレイしません。
- ZJDE または XJDE で始まるカスタム・バージョン名や新規バージョン名を作成しないでください。これらは、テンプレートやバージョンをコピーしたり新しく作成するために J.D. Edwards から送付される標準バージョン・テンプレート用に予約されているプレフィックスです。これらのプレフィックスを修正に使用すると、名前の競合が発生した場合にカスタム・バージョンは保持されません。
- アップグレードの際は、最後に修正されたセントラル・オブジェクト・セットからパッケージをビルドし、開発用サーバー、セントラル・オブジェクトおよびオブジェクト・ライブラリアンの各データ・ソースのバックアップを実行して、比較やトラブルシューティングのためにこれらのスペックにアクセスできるようにしてください。

対話型アプリケーションに関する規則

既存の J.D. Edwards 対話型アプリケーションのコントロール、グリッド・カラム、およびハイパー項目は削除せずに、非表示または無効にしてください。J.D. Edwards ソフトウェアでは、これらの項目が計算に使用されていたり、変数として使用されていることがあるので、削除すると主要な機能が使用できなくなることがあるためです。

アップグレード時に置換される修正

既存の J.D. Edwards ソフトウェア・アプリケーションのカスタム・フォームは、アップグレード中に置換されます。カスタム・フォームは、既存の J.D. Edwards アプリケーションに配置する代わりに、システム・コード 55～59 を使用するカスタム・アプリケーションを作成し、そのカスタム・アプリケーションに配置することをお勧めします。そして、カスタム・アプリケーションのカスタム・フォームを呼び出すフォーム・エグジットとロー・エグジットを既存のアプリケーションに追加します。外部アプリケーションをロー・エグジットから呼び出しても、アプリケーションのフォームから呼び出しても、システムで違いは認識されず、パフォーマンスは変わりません。

アップグレード中には、カスタム・フォームに加えて次の対話型アプリケーション要素が置換されます。

オブジェクト	コメント
新規アプリケーション	アプリケーションを新規に作成する方法と、〈Application Design Aid〉の[Copy]機能を使用して既存のアプリケーションをコピーする方法があります。後者の方法では、イベント・ルールを含む、アプリケーションのすべてのスペックをコピーすることができます。 **注意:** コピー機能を使って既存のアプリケーションをコピーして修正した場合、JD.Edwards がオリジナルのアプリケーションに対して行っていた変更があっても、それらはアップグレードの際にはコピーされた新しいアプリケーションに適用されません。
既存のフォームに追加された新規ハイパー項目	
既存のフォームに追加された新規コントロール	
既存のフォームに追加された新規グリッド・カラム	
スタイル変更	スタイル変更にはフォントと色が含まれます。J.D. Edwards の新規コントロールには、標準ベース定義が設定されています。スタイルを改訂した場合は、アプリケーションに追加する新規コントロールも同様に改訂する必要があります。
コード・ジェネレーター時変更	
データ辞書ー時変更	
位置とサイズの変更	ソフトウェアの新リリースに含まれている新しいコントロールとカスタム・コントロールの配置位置が重なった場合、これらのコントロールは相互に重なり合って表示されます。これは、イベント・ルールやアプリケーションの機能に影響しません。アップグレード後にアプリケーション設計ツールを使用すれば、これらのコントロールの位置を変更できます。
タブまたはカラムの順序変更	アップグレード処理時には、J.D. Edwards の新規コントロールがカスタム・タブの最後に追加されます。アップグレード後にタブ順序を確認できます。

アップグレード時に保持される修正

アップグレードの際に、次の対話型アプリケーション要素が保持されます。

オブジェクト	コメント
新規アプリケーション	アプリケーションを作成するには、最初から作成する方法と、〈Application Design Aid〉の「Copy」機能を使用して、既存のアプリケーションをコピーする方法があります。後者の方法では、イベント・ルールを含む、アプリケーションのすべてのスペックをコピーすることができます。 コピー機能を使って既存のアプリケーションをコピーして修正した場合、J.D. Edwards がオリジナルのアプリケーションに対して行っていた変更があっても、それらはアップグレードの際にはコピーされた新しいアプリケーションに適用されません。
既存のフォームに追加された新規ハイパー項目	
既存のフォームに追加された新規コントロール	
既存のフォームに追加された新規グリッド・カラム	
スタイル変更	スタイル変更にはフォントと色が含まれます。J.D. Edwards の新規コントロールには、標準ベース定義が設定されています。スタイルを改訂した場合は、アプリケーションに追加する新規コントロールも同様に改訂する必要があります。
コード・ジェネレーター時変更	
データ辞書一時変更	
位置とサイズの変更	ソフトウェアの新リリースに含まれている新しいコントロールとカスタム・コントロールの配置位置が重なった場合、これらのコントロールは相互に重なり合って表示されます。これは、イベント・ルールやアプリケーションの機能に影響しません。アップグレード後、〈Application Design Aid〉を使用すれば、これらのコントロールの位置を変更できます。
タブまたはカラムの順序変更	アップグレード処理時には、J.D. Edwards の新規コントロールがカスタム・タブの最後に追加されます。アップグレード後にタブの配列を見直すことができます。

レポート

〈Report Design Aid(レポート設計ツール)〉のスペックには、次の規則が適用されます。

既存の J.D. Edwards ソフトウェア・レポートのオブジェクトは削除するのではなく、非表示にしてください。これらの項目が計算に使用されていたり、変数として使用されていることがあるため、削除すると主要な機能が使用できなくなることがあるためです。

アップグレード時に保持される修正

アップグレードの際に、次のレポート要素が保持されます。

オブジェクト	コメント
新規レポート	レポートを作成するには、最初から作成する方法と、〈Report Design Aid〉の「Copy」機能を使用して、既存のレポートをコピーする方法があります。後者の方法では、イベント・ルールを含む、レポートのすべてのスペックをコピーすることができます。 コピー機能を使って既存のレポートをコピーして修正した場合、J.D. Edwards がオリジナルのレポートに対して行っていた変更があっても、それらはアップグレードの際にはコピーされた新しいレポートに適用されません。
既存のレポートに追加された新しい定数	
既存のレポートに追加された新規変数（英字）	
既存のレポートに追加された新規変数（数値）	
既存のレポートに追加された新規日付変数	
既存のレポートに追加された新規ランタイム変数	
既存のレポートに追加された新規データベース変数	
既存のレポートに追加された新規データ辞書変数	
スタイル変更	スタイル変更にはフォントと色が含まれます。J.D. Edwards の新規コントロールには、標準ベース定義が設定されています。スタイルを改訂した場合は、レポートに追加する新規コントロールも同様に改訂する必要があります。
オブジェクトのロケーション変更とサイズ変更	ソフトウェアの新リリースに含まれている、コントロールなどの新しいオブジェクトとカスタム・オブジェクトの配置位置が重なった場合、これらのオブジェクトは相互に重なり合って表示されます。これは、イベント・ルールやレポートの機能に影響しません。アップグレード後、〈Report Design Aid〉を使用すれば、オブジェクトの位置を変更できます。
データ辞書一時変更	

アップグレード時に置換される修正

アップグレード時には、既存の J.D. Edwards ソフトウェア・レポートのカスタム・セクションが置換されます。カスタム・セクションは、既存のレポートに追加する代わりに、〈Report Interconnect（レポート・インターコネクト）〉を使って、システム・コード 55～59 を使用する新しいカスタム・レポートに接続してください。レポート・インターコネクトを通じて呼び出されるレポートの場合も、システム・パフォーマンスは変わりません。

アプリケーション・テキストの変更

アップグレード時に保持される修正

アップグレードの際に、次のアプリケーション・テキスト要素が保持されます。

- 〈Application Design Aid〉で行った一時変更
- 〈Report Design Aid〉で行った一時変更
- 〈Interactive Vocabulary Override（対話型用語一時変更）〉で行った一時変更
- 〈Batch Vocabulary Override（バッチ用語一時変更）〉で行った一時変更

テーブル・スペック

アップグレード時には、あるリリース・レベルのテーブル・スペックが次のレベルにマージされます。

アップグレード時に保持される修正と置換される修正

次の表に、アップグレード時に保持されるテーブル・スペック要素と、置換される要素を示します。

オブジェクト	保持	置換	コメント
新規テーブル	X		
J.D. Edwards テーブルへのカスタム・インデックス	X		
既存の J.D. Edwards テーブルに追加/削除されたカラム		X	この修正にはフィールド長、フィールド・タイプおよび小数点以下桁数の変更も含まれます。 既存の J.D. Edwards テーブルに新しいカラムを追加する代わりに、システム・コード 55～59 を使用するタグ・ファイルを使用してください。

注：

カスタム・タグ・ファイルでは、J.D. Edwards データ辞書のデータ項目の変更を確認してください。データ項目サイズなど、ある種のデータ項目属性はリリース間で変更されることがあり、データの整合性やデータベースへの格納方法に影響を及ぼす可能性があります。このため、〈Table Conversion〉ツールを使用して、タグ・ファイル・データを新しいリリース・レベルに変換しなければならないことがあります。基本 J.D. Edwards ファイルでは、アップグレード・プロセスが J.D. Edwards ソフトウェア・データベースを新しいリリース・レベルにアップグレードして、データ辞書の変更を実行します。アップグレード・プロセスでは、カスタム・タグ・ファイル上のカスタム・インデックスが保持されます。

コントロール・テーブル

コントロール・テーブルには、ユーザー定義コード(UDC)、メニューおよびデータ辞書項目が存在します。アップグレード時には、〈Change Table(テーブル変更)〉プロセスによって、コントロール・テーブルがあるリリース・レベルから次のレベルにマージされます。このプロセスでは、データ・マージを実行するための基礎として J.D. Edwards テーブルではなくカスタムのコントロール・テーブルを使用します。

アップグレード時に保持される修正と置換される修正

次の表に、アップグレード時に保持されるコントロール・テーブル要素と、置換される要素を示します。

オブジェクト	保持	置換	コメント
データ辞書カスタム変更	X		ロー、カラム、用語解説テキストへの変更が含まれます。アップグレード・プロセスでは、基礎としてデータ辞書が使用されます。値が J.D. Edwards データ項目と矛盾する場合は、J.D. Edwards 値が一時変更されます。新規のデータ項目は、システム・コード 55～59 を使用して作成してください。
ユーザー定義コード	X		アップグレード・プロセスでは、ハード・コーディングされた J.D. Edwards の新しい値をすべてマージします(J.D. Edwards 所有の値は、システム 90 以上と H90 以上です)。このプロセスでは、カスタム値と競合する J.D. Edwards のハード・コーディングされた値もすべて報告します。
メニュー	X		カスタム・メニューが J.D. Edwards 基本メニューと競合する場合は、カスタム変更により J.D. Edwards 基本メニューが一時変更されます。
既存の J.D. Edwards コントロール・テーブルに追加/削除されたカラム		X	

ビジネス・ビュー

既存のビジネス・ビューからはカラムを削除しないでください。アプリケーションが使用するビジネス・ビューを修正すると、アプリケーション実行時に予測不可能な結果をもたらす場合があります。カラムを非表示するか削除する必要があれば、〈Application Design Aid〉または〈Report Design Aid〉を使用して、これらのツール上で操作してください。ビジネス・ビューから少数のカラムを削除しても、パフォーマンスはほとんど変化しません。

アップグレード時に保持される修正と置換される修正

次の表に、アップグレード時に保持されるビジネス・ビュー要素と、置換される要素を示します。

オブジェクト	保持	置換
新規カスタム・ビジネス・ビュー	X	
既存の J.D. Edwards ビジネス・ビューに追加された新規のカラム、結合またはインデックス	X	
J.D. Edwards ビジネス・ビューから削除されたカラム		X

データ構造体

次の表に、アップグレード時に保持されるデータ構造体要素と、置換される要素を示します。

オブジェクト	保持	置換
カスタム・フォーム・データ構造体	X	
カスタム処理オプション・データ構造体	X	
カスタム・レポート・データ構造体	X	
カスタム・ビジネス関数データ構造体	X	
カスタム汎用テキスト・データ構造体	X	
既存の J.D. Edwards フォーム・データ構造体への修正		X
既存の J.D. Edwards 処理オプション・データ構造体への修正		X
既存の J.D. Edwards レポート・データ構造体への修正		X
既存の J.D. Edwards ビジネス関数データ構造体への修正		X
既存の J.D. Edwards 汎用テキスト・データ構造体への修正		X

J.D. Edwards データ構造体に対するカスタム修正を次のリリース・レベルに送るには、〈Object Management Workbench Modifications Report〉レポート(R9840D)を実行し、修正されたデータ構造体の一覧を表示し、このレポートを手引きにして、データ構造体を手作業で変更し直します。

イベント・ルール

アップグレード・プロセスを開始する前に、次の作業を完了しておくことをお勧めします。

- 修正済みアプリケーションを識別するために、〈Object Management Workbench Modifications Report〉レポート(F9840D)を実行します。
- 修正済みアプリケーションのイベント・ルールを印刷し、カスタム・イベント・ルールを復元する際にイベントのロジックを検討できるようにします。

次の表に、アップグレード時に保持されるイベント・ルール要素、および置換される要素を示します。

オブジェクト	保持	置換	コメント
カスタム・アプリケーション、レポートおよびテーブルのカスタム・イベント・ルール	X		
カスタム・ビジネス関数のカスタム・イベント・ルール	X		
新規カスタム・コントロールのカスタム・イベント・ルール	X		

J.D. Edwards イベント・ルールを同じイベントに設定していない J.D. Edwards のアプリケーション、レポートおよびテーブルのイベント	X		
J.D. Edwards イベント・ルールを同じイベントに設定していない J.D. Edwards ビジネス関数のイベント	X		
既存のイベント・ルールを同じイベントに設定している J.D. Edwards のアプリケーション、レポートおよびテーブルのイベント		X	アップグレード・プロセスでは、カスタム・イベント・ルールを無効にし、それらをイベント・ルールの最後に追加します。アップグレード・プロセスで無効にされたイベント・ルールを通知する〈Object Librarian Modifications Report〉レポート(R9840D)が生成されます。
イベント・ルールを同じイベントに設定している J.D. Edwards ビジネス関数のイベント		X	アップグレード・プロセスでは、カスタム・イベント・ルールを無効にし、それらをイベント・ルールの最後に追加します。アップグレード・プロセスで無効にされたイベント・ルールを通知する〈Object Librarian Modifications Report〉レポート(R9840D)が生成されます。

カスタム・イベント・ルールを J.D. Edwards オブジェクトに復元するには、そのイベント・ルールをハイライトし、イベントの正しい位置までドラッグして有効化します。

バージョン

アップグレード・プロセスの前に、XJDE または ZJDE で始まる新規カスタム・バージョン名がないことを確認してください。

アップグレード時に保持される修正と置換される修正

次の表に、アップグレード時に保持されるバージョン要素、および置換される要素を示します。

オブジェクト	保持	置換
非 JDE バージョン	X	
バージョン・スペック	X	
処理オプション・データ	X	
ZJDE および XJDE バージョンのすべてのスペック		X
XJDE バージョンのすべての処理オプション・データ		X

この表に示す情報の他に処理オプション・データがコピーされますが、非 JDE バージョンの処理オプション・データは JDE 処理オプション・テンプレートを使用して変換されません。実行時に警告メッセージが表示され、一部のデータが消失することがあります。また、J.D. Edwards テンプレートにおけるカスタム・バージョンのイベント・ルールの修正は、J.D. Edwards 親テンプレートとは調整されません。

ビジネス関数

新規のカスタム・ビジネス関数では、常に新しい（カスタム）親 DLL を作成して、カスタム修正を保管してください。

カスタム変更を次のリリース・レベルに送るには、〈Object Librarian Modifications〉レポート(R9840D)を実行し、修正されたビジネス関数の一覧を表示し、このレポートを手引きにして、ビジネス関数を手動で変更し直します。

予測不能な結果を避けるために、ビジネス関数で他のビジネス関数を呼び出す場合は、常に標準API(jdeCallObject)を使用してください。

J.D. Edwards の既存の標準ビジネス関数のソースに対する修正を判別するには、Microsot WinDiff など、サードパーティのソース比較ツールを使用します。ビジネス関数で API に対する修正を判別する方法については、最新 API に関する J.D. Edwards ソフトウェア・オンライン・ヘルプを参照してください。

アップグレード時に保持される修正と置換される修正

次の表に、アップグレード時に保持されるビジネス関数要素と、置換される要素を示します。

オブジェクト	保持	置換	コメント
新規のカスタム・ビジネス関数オブジェクト	X		
既存の J.D. Edwards ビジネス関数オブジェクトに加えられた修正		X	NER ビジネス関数を修正できます。

フォームとコントロールの処理

このセクションでは、次のフォーム・タイプの処理について説明します。

- 検索/表示
- 親/子
- 修正/検査
- 見出し詳細
- 見出しなし詳細
- 検索/選択
- メッセージ

また、ここでは編集およびグリッド・コントロールの処理についても説明します。

検索/表示フォームのプロセス・フロー

検索/表示フォームは、ビジネス・ビューをクエリーしたり、ビジネス・ビューから操作対象レコードを選択するために使用されます。

デフォルトのフラグ

デフォルトでは、検索/表示フォームのフォーム・オプション・フラグはチェックされません。このフォーム・タイプに影響することがある唯一のフォーム・オプション・フラグは[No Fetch on Grid Business View(グリッド・ビジネス・ビューでのフェッチなし)]フラグです。これ以外のどのフォーム・オプション・フラグを選択しても、フォームの処理には影響しません。

[Entry Point(エントリ・ポイント)]プロパティがデフォルトでアクティブ化されるフォームは、検索/表示フォームと親/子表示フォームのみです。検索/表示フォームは通常、アプリケーションへのエントリ・ポイントになります。アプリケーション開発者は[Entry Point]プロパティを非アクティブ化することができます。

ダイアログの初期化

1. スレッド処理を初期化します。
2. エラー処理プロセスを初期化します。
3. ビジネス・ビュー・カラムを初期化します。
4. フォーム・コントロールを初期化します。
5. グリッド・フィールドを初期化します。
6. Static テキストを初期化します。
7. ヘルプを初期化します。
8. イベント・ルール構造体を初期化します。

9. ツールバーを作成します。
10. フォーム・インターコネクト・データを、対応するビジネス・ビュー・カラムとフィルタ用フィールドに必要に応じてロードします。
11. 次のイベント・ルールを実行します。: Dialog is Initialized
12. 次のイベント・ルールを実行します。: Post Dialog is Initialized
13. 詳細データの選択と順序設定を開始します(グリッド・オプション[Automatically Find On Entry]がアクティブ化されている場合)。

ヘッダー・データの取出し

検索/表示フォームには、ヘッダー・レコードはありません。

詳細データの選択と順序設定

ここで内部構造体が作成され、ユーザーによって指定されたデータ選択条件とデータ順序設定条件の式が形成されます。これはその後、データベースの実際の選択と順序設定を行うためにデータベース・エンジンに送られます。選択用のデータは、フィルタ用フィールドと QBE カラムの値に基づいて使用されます。データは、次のステップで取り出されるまで保管されます。

フォーム・オプション・フラグの[No Fetch On Grid Business View]がオフの場合は、次の 2 つの処理が実行されます。

- 選択と順序設定を実行します。
- データの取出しを開始します。

フォーム・オプション・フラグの[No Fetch On Grid Business View]がオンの場合、データの取出しは行われません。

データの取出し

要求が JDEKRNL に送られ、データベースからデータが実際にフェッチされます。この場合、システムはレコードを 1 つずつ読み取り、レコードごとに次の処理を実行します。

1. データベースからのレコードのフェッチを試みます。
2. 成功すると、次の処理が実行されます。
   - ビジネス・ビュー・データ構造体にデータをコピーします。
   - 次のイベント・ルールを実行します。: Grid Record is Fetched
3. アプリケーションの開発者が、このグリッド・レコードの書出しを抑制していない場合は、次の処理が実行されます。
   - ビジネス・ビュー・データをグリッド・データ構造体にコピーします。
   - 次のイベント・ルールを実行します。: Write Grid Line – Before
   - ローをグリッドに追加します。この時点で、グリッド・コントロールにローが書かれます。
   - 次のイベント・ルールを実行します。: Write Grid Line – After
   - 次のレコードを読み取るためにグリッド・データ構造体をクリアします。

- グリッド・ラインの抑制フラグを解除します。

以上のステップは、データベースから読み取られたレコードごとに発生します。すべてのレコードが読み取られた後は、次の処理が実行されます。

4. 次のイベント・ルールを実行します。: Last Grid Record Has Been Read

フォームのクローズ

1. 次のイベント・ルールを実行します。: End Dialog
2. フォーム・インターコネクト・データを、対応するビジネス・ビュー・カラムから必要に応じてロードします。
3. エラー処理を終了します。
4. スレッド処理を終了します。
5. ヘルプを終了します。
6. ビジネス・ビュー・カラム、フォーム・コントロール、グリッド・カラム、およびイベント・ルールのそれぞれの構造体を含む、フォームの構造体をすべて解放します。
7. ウィンドウを消去します。

メニュー/ツールバー項目

ここでは、次のメニュー項目とツールバー項目について説明します。

- Select
- Close
- Delete
- Find
- Copy
- Add
- ユーザー定義項目

Select

[Select(選択)]は、検索/表示フォームに自動的に設定される標準項目です。検索/表示フォームの[Select]には、デフォルトの処理は存在しません。[Select]はユーザー定義項目として動作します。

Close

[Close]は、検索/表示フォームに自動的に表示される標準項目です。この項目はフォームを閉じます。

1. 次のイベント・ルールを実行します。: Button Clicked
2. 次のイベント・ルールを実行します。: Post Button Clicked
3. フォームのクローズを開始します。

Find

[Find(検索)]は、検索/表示フォームに自動的に表示される標準項目です。この項目をユーザーがクリックすると、ランタイム・エンジンに信号が送られ、ランタイム・エンジンがデータベースを呼び出し、フィルタ用フィールドの情報に基づいてグリッドを再ロードします。

1. 次のイベント・ルールを実行します。:Button Clicked
2. フィルタ用フィールドにエラーがない場合は、次の処理が実行されます。
   - データの選択と順序設定を開始します。
   - 次のイベント・ルールを実行します。:Post Button Clicked

Delete

[Delete(削除)]は、検索/表示フォームに追加できる標準項目です。これは、グリッドのレコードをデータベースから削除します。

1. 次のイベント・ルールを実行します。:Button Clicked
2. 削除可能な選択済みグリッド・ローごとに次の処理が実行されます。
   - グリッド・ロー・データをビジネス・ビュー構造体にコピーします。
   - [Suppress Delete]フラグを解除します。
   - 次のイベント・ルールを実行します。:Delete Grid Rec Verify - Before.
3. [Suppress Delete]フラグが設定されていない場合は、次の処理が実行されます。
   - [Delete Confirmation]ダイアログを表示します。[No]または[Cancel]がクリックされると、この処理の残りの部分をスキップします。
   - 次のイベント・ルールを実行します。:Delete Grid Rec Verify - After
   - 次のイベント・ルールを実行します。:Delete Grid Rec from DB - Before
   - ビジネス・ビューのレコードをデータベースから削除します。
   - グリッド・コントロールからグリッド・ローを削除します。
   - 次のイベント・ルールを実行します。:Delete Grid Rec from DB - After
   - 次のイベント・ルールを実行します。:All Grid Recs Deleted
   - 次のイベント・ルールを実行します。:Post Button Clicked

Copy

[Copy(コピー)]は、検索/表示フォームに追加できる標準項目です。コピー機能により、ユーザーはオブジェクトを複製し、名前を変更できます。検索/表示フォームの[Copy]機能には、デフォルトの処理は存在しません。これはユーザー定義項目として動作します。フォーム・インターコネクトがButton Clicked イベントに設定されていると、呼び出されたフォームはコピー・モードで開きます。

Add

[Add(追加)]は、検索/表示フォームに追加できる標準項目です。検索/表示フォームの[Add]には、デフォルトの処理は存在しません。これはユーザー定義項目として動作します。フォーム・インターコネクトが Button Clicked イベントに設定されている場合、呼び出されたフォームは追加モードで開きます。

ユーザー定義項目

ユーザー定義項目は、検索/表示フォームに追加できる非標準項目で、標準項目では対応できない特別な処理を実行します。

1. 次のイベント・ルールを実行します。:Button Clicked
2. 次のイベント・ルールを実行します。:Post Button Clicked

親/子表示フォームのプロセス・フロー

親/子表示フォームは、ビジネス・ビューを照会し、データを階層形式で表現するために使用します。そのビジネス・ビューからレコードを選択して操作することもできます。ここでは、親/子表示フォーム・タイプのプロセス・フローについて説明します。

デフォルトのフラグ

親/子表示フォームには、デフォルトでオンに設定されるフォーム・オプション・フラグはありません。このフォーム・タイプに影響することがある唯一のフォーム・オプション・フラグは[No Fetch on Grid Business View]フラグです。これ以外のどのフォーム・オプション・フラグをオンにしても、フォームの処理には影響しません。

[Entry Point]プロパティがデフォルトでアクティブ化されるフォームは、親/子表示フォームと検索/表示フォームのみです。親/子表示フォームは一般にアプリケーションへのエントリ・ポイントになります。エントリ・ポイントのプロパティは、アプリケーションの開発者によってオフにすることもできます。

ダイアログの初期化

1. スレッド処理を初期化します。
2. エラー処理プロセスを初期化します。
3. ビジネス・ビュー・カラムを初期化します。
4. フォーム・コントロールを初期化します。
5. グリッド/ツリー・フィールドを初期化します。
6. Static テキストを初期化します。
7. ヘルプを初期化します。
8. イベント・ルール構造体を初期化します。
9. ツールバーを作成します。

10. フォーム・インターコネクト・データを、対応するビジネス・ビュー・カラムとフィルタ用フィールドに必要に応じてロードします。
11. 次のイベント・ルールを実行します。: Dialog is Initialized
12. 次のイベント・ルールを実行します。: Post Dialog is Initialized
13. グリッド・オプション[Automatically Find On Entry(入力時に自動検索)]がオンに設定されている場合は、詳細データの選択と順序設定を開始します。

ヘッダー・データの取出し

親/子表示フォームには、ヘッダー・レコードはありません。

詳細データの選択と順序設定

ここで内部構造体が作成され、ユーザーによって指定されたデータ選択条件とデータ順序設定条件の式が形成されます。この構造体は、その後、データベースの実際の選択と順序設定を行うためにデータベース・エンジンに送られます。データは、次のステップで取り出されるまで保管されます。

選択用のデータは、フィルタ用フィールドと QBE カラムから取り出されます。

フォーム・オプション・フラグの[No Fetch On Grid Business View]がオフの場合は、次の処理が実行されます。

- 選択と順序設定を実行します。
- データの取出しを開始します。

データの取出し

要求が JDEKRNL に送られ、データベースからデータを実際にフェッチします。この場合、レコードを 1 つずつ読み取り、レコードごとに必要な処理を実行します。親/子表示フォームでは、データの取出しは 2 とおりの方式で実行されます。ツリーの第 1 レベルのノードを取得する際に実行されるフェッチは、検索/表示フォームで実行されるフェッチと同じです。

ツリーの第 1 レベル・ノードにおけるデータの取出し

1. データベースからのレコードのフェッチを試みます。
2. レコードがフェッチされた場合は、次の処理を実行します。
   - ビジネス・ビュー・データ構造体にデータをコピーします。
   - 次のイベント・ルールを実行します。: Grid Record is Fetched
3. アプリケーションの開発者が、このレコードの書出しを抑制していない場合は、次の処理が実行されます。
   - ビジネス・ビュー・データをグリッド・データ構造体にコピーします。
   - 次のイベント・ルールを実行します。: Write Grid Line – Before
   - グリッドとそれに対応するツリー上のカラムにローを追加します。この時点で、ローとそれに対応するツリー・ノードがコントロールに書かれます。

- 次のイベント・ルールを実行します。: Write Grid Line – After
- 次のレコードを読み取るためにデータ構造体をクリアします。

以上のステップは、データベースから読み取られたレコードごとに発生します。すべてのレコードが読み取られた後は、次の処理が実行されます。

4. 親/子表示フォームのグリッドが常に非表示にされている場合は、ヘッダー・ノードを拡大し、ヘッダーの下の最初の子ノードに選択を移動します。これにより、イベント Tree – Node Level Changed および Tree – Node Selection Changed が開始されます。
5. 次のイベント・ルールを実行します。: Last Grid Record Has Been Read

親/子表示フォームでは、特定のツリー・ノードが展開されるたびにデータの取出しが実行されます。ただし、このフェッチは、展開されるノードごとに 1 度だけ実行されます。特定のノードを縮小してから再度展開すると、ツリーとグリッドに内部構造体から再びデータが送られます。ノード展開イベントで実行する処理は、以下で指定します。

ツリー・ノードの展開時におけるデータの取出し

1. 次のイベント・ルールを実行します。: Tree – Node is expanding
2. 内部フラグをチェックして、フェッチが抑制されているかどうかを調べます。システム関数 Suppress Fetch on Node Expand によってフェッチが抑制されている場合は、ここで処理が停止します。
3. フォームの設計時に設定されたキーの置換(子キーを親キーにコピーする)を実行します。
4. データベースからのレコードのフェッチを試みます。
5. レコードがフェッチされた場合は、次の処理を実行します。
   - ビジネス・ビュー・データ構造体にデータをコピーします。
   - 次のイベント・ルールを実行します。: Grid Record is Fetched

   アプリケーションの開発者が、このレコードの書出しを抑制していない場合は、次の処理が実行されます。

   - ビジネス・ビュー・データをグリッド・データ構造体にコピーします。
   - 次のイベント・ルールを実行します。: Write Grid Line – Before
   - グリッドとそれに対応するツリー上のカラムにローを追加します。この時点で、ローとそれに対応するツリー・ノードがコントロールに書かれます。
   - 次のイベント・ルールを実行します。: Write Grid Line – After
   - 次のレコードを読み取るためにデータ構造体をクリアします。
   - 次のイベント・ルールを実行します。: Last Grid Record Has Been Read
   - 展開ノードの下の最初の子ノードにツリーの選択を移します。イベントが開始されます。: Tree – Node Level Changed および Tree – Node Selection Changed

フォームのクローズ

1. 次のイベント・ルールを実行します。:End Dialog
2. フォーム・インターコネクト・データを、対応するビジネス・ビュー・カラムから必要に応じてロードします。
3. エラー処理を終了します。
4. スレッド処理を終了します。
5. ヘルプを終了します。
6. ビジネス・ビュー・カラム、フォーム・コントロール、グリッド・カラム、およびイベント・ルールのそれぞれの構造体を含む、フォームの構造体をすべて解放します。
7. ウィンドウを消去します。

メニュー/ツールバー項目

ここでは、親/子フォームのメニュー/ツールバー項目のプロセス・フローについて説明します。

Select

[Select]は、親/子表示フォームに自動的に設定される標準項目です。親/子表示フォームの[Select]には、デフォルトの処理は存在しません。[Select]はユーザー定義項目として動作します。

Close

[Close]は、親/子表示フォームに自動的に表示される標準項目です。この項目はフォームを閉じます。

1. 次のイベント・ルールを実行します。:Button Clicked
2. 次のイベント・ルールを実行します。:Post Button Clicked
3. フォームのクローズを開始します。

Delete

[Delete]は、親/子表示フォームに追加可能な標準項目です。親/子表示フォームは、検索/表示フォームと同様に次の操作を実行します。

1. 次のイベント・ルールを実行します。:Button Clicked
2. 選択されたツリー・ノードで、次の処理を実行します。
   - グリッド・ロー・データをビジネス・ビュー構造体にコピーします。
   - [Suppress Delete]フラグを解除します。
   - 次のイベント・ルールを実行します。:Delete Grid Rec Verify – Before

3. [Suppress Delete]フラグが設定されていない場合は、次の処理が実行されます。
    - [Delete Confirmation]ダイアログを表示します。ユーザーが[No]または[Cancel]をクリックすると、この処理は中断されます。
    - 次のイベント・ルールを実行します。:Delete Grid Rec Verify – After
    - 次のイベント・ルールを実行します。:Delete Grid Rec from DB – Before
4. [Suppress Delete]フラグが設定されていない場合は、次の処理が実行されます。
    - ビジネス・ビューのレコードをデータベースから削除します。
    - ノードの子レコードはデータベースから削除されません。これらのレコードをデータベースから削除するかどうかは、開発者に委ねられています。
    - グリッド・コントロールからグリッド・ローを削除します。
    - 次のイベント・ルールを実行します。:Delete Grid Rec from DB – After
5. 次のイベント・ルールを実行します。:All Grid Recs Deleted
6. 次のイベント・ルールを実行します。:Post Button Clicked

Find

[Find]は、親/子表示フォームに自動的に表示される標準項目です。ユーザーが[Find]をクリックすると、ランタイム・エンジンに信号が送られ、ランタイム・エンジンがデータベースを呼び出し、フィルタ用フィールドと QBE カラムの情報に基づいてグリッドを再ロードします。

1. 次のイベント・ルールを実行します。:Button Clicked
2. フィルタ用フィールドにエラーがない場合は、次の処理が実行されます。
    - データの選択と順序設定を開始します。
    - 次のイベント・ルールを実行します。:Post Button Clicked

Copy

[Copy]は、親/子表示フォームに追加可能な標準項目です。親/子表示フォームの[Copy]には、デフォルトの処理は存在しません。フォームは、ユーザー定義項目として動作します。フォーム・インターコネクションが Button Clicked イベントに設定されていると、呼び出されたフォームはコピー・モードで開きます。

Add

[Add]は、親/子表示フォームに追加可能な標準項目です。親/子表示フォームの[Add]には、デフォルトの処理は存在しません。フォームは、ユーザー定義項目として動作します。フォーム・インターコネクトが Button Clicked イベントに設定されている場合、呼び出されたフォームは追加モードで開きます。

ユーザー定義項目

ユーザー定義項目は、親/子表示フォームに追加できる非標準項目で、標準項目では対応できない特別な処理を実行します。

3. 次のイベント・ルールを実行します。:Button Clicked
4. 次のイベント・ルールを実行します。:Post Button Clicked

修正/検査フォームのプロセス・フロー

修正/検査フォームは、単一のデータベース・レコードを更新、挿入するために使用します。このフォームは、〈Single Record Maintenance(単一レコード・メンテナンス)〉フォームとも呼ばれています。このフォーム・タイプを使用すると、ユーザーに対して情報を求めるプロンプトや静的情報を表示することもできます。たとえば、サインオン画面には、サインオン情報の入力を求めるプロンプトが表示されます。ここでは、修正/検査フォームのプロセス・フローについて説明します。

デフォルトのフラグ

デフォルトで設定されるフォーム・オプション・フラグはありません。

ダイアログの初期化

1. スレッド処理を初期化します。
2. エラー処理プロセスを初期化します。
3. ビジネス・ビュー・カラムを初期化します。
4. フォーム・コントロールを初期化します。
5. Static テキストを初期化します。
6. ヘルプを初期化します。
7. イベント・ルール構造体を初期化します。
8. ツールバーを作成します。
9. フォーム・インターコネクト・データを、対応するビジネス・ビュー・カラムにロードします。

   注:
   コントロールとビジネス・ビューのカラムは内部メモリを共有するため、ビジネス・ビューにコピーすると、実際には内部フォーム・コントロールのメモリ位置にコピーされます。

10. 次のイベント・ルールを実行します。:Dialog is Initialized
11. [No Fetch Form Business View(フォーム・ビジネス・ビューのフェッチなし)]がオンの場合は、次の処理を実行します。
    - 次のイベント・ルールを実行します。:Post Dialog is Initialized

12. フォームが追加モードの場合は、次の処理を実行します。
    - ダイアログのクリアを開始します。
13. フォームが更新モードで[No Fetch Form Business View]がオンの場合は、次の処理を実行します。
    - モードを追加モードに変更します。
    - 内部のフォーム・コントロール値を画面に表示します。
    - ダイアログのクリアを開始します。
14. フォームが更新モードで[No Fetch Form Business View]がオフの場合は、次の処理を実行します。
    - データの取出しを開始します。

データの取出し

要求が JDEKRNL に送られ、データベースからデータを実際にフェッチします。フェッチは、フォーム・データ構造体を通じてフォームに送られる情報に基づいて行われます。

1. データベースからのレコードのフェッチを試みます。
2. レコードがフェッチされた場合は、次の処理を実行します。
    - ビジネス・ビュー・データ構造体にデータをコピーします。

注:
ビジネス・ビューにコピーすると、実際には内部のフォーム・コントロールのメモリ位置にコピーされます。

---

3. レコードが見つかり、フォームがコピー・モードの場合は、次の処理を実行します。
    - 追加モードに切り替えます。
    - 内部のフォーム・コントロール値を画面に表示します。
    - ダイアログのクリアを開始します。
4. レコードが見つかり、フォームがコピー・モードでない場合は、次の処理を実行します。
    - 次のイベント・ルールを実行します。:Post Dialog is Initialized
5. レコードがフェッチされなかった場合は、次の処理を実行します。
    - 追加モードに切り替えます。
    - 内部のフォーム・コントロール値を画面に表示します。
    - ダイアログのクリアを開始します。
6. 最終的なデータベース・モードに基づいてデフォルトのフォーカスを設定します。

ダイアログのクリア

1. フォームがコピー・モードで呼び出された場合は、次の処理を実行します。
   - [Do Not Clear After Add(追加後にクリアしない)]フラグがオフであるキー(プライマリ・インデックス)コントロールをクリアします。
2. フォームがコピー・モードで呼び出されなかった場合は、次の処理を実行します。
   - [Do Not Clear After Add]フラグがオフであるすべてのフォーム・コントロールをクリアします。
3. 次のイベント・ルールを実行します。:Clear Screen Before Add
4. 次のイベント・ルールを実行します。:Post Dialog is Initialized

フォームのクローズ

1. 次のイベント・ルールを実行します。:End Dialog
2. フォーム・インターコネクト・データを、対応するビジネス・ビュー・カラムから必要に応じてロードします。
3. エラー処理を終了します。
4. スレッド処理を終了します。
5. ヘルプを終了します。
6. ビジネス・ビュー・カラム、フォーム・コントロール、イベント・ルールのそれぞれの構造体を含む、フォームのすべての構造体を解放します。
7. ウィンドウを消去します。

メニュー/ツールバー項目

ここでは、修正/検査フォームのメニュー/ツールバー項目のプロセス・フローについて説明します。

OK

[OK]は、修正/検査フォームに自動的に設定される標準項目です。この項目はフォームの情報を検査し、JDEKRNL の関数呼出しを通じてデータベースに更新または追加します。

1. フォームにエラーまたは警告が表示される場合は、[OK]の処理を停止します。
2. 次のイベント・ルールを実行します。:Button Clicked
3. フォームのコントロールごとに、次の処理を実行します。
   - 現在のコントロールがフォーム・コントロールで、検査をパスしていない場合は、次の処理を実行します。

     次のイベント・ルールを実行します。:Control is Exited

     次のイベント・ルールを実行します。:Control is Exited and Changed – Inline

     次のイベント・ルールを実行します。:Control is Exited and Changed – Async

データ辞書を検査します。

4. フォームにエラーが表示される場合は、[OK]の処理を停止します。
5. フォームが追加モードの場合は、次の処理を実行します。
   - 次のイベント・ルールを実行します。:Add Record to Database – Before
6. フォームが更新モードの場合は、次の処理を実行します。
   - 次のイベント・ルールを実行します。:Update Record to Database – Before
7. 次のイベント・ルールを実行します。:Post Button Clicked
8. フォームが追加モードの場合は、次の処理を実行します。
   - フォームがコピー・モードで呼び出された場合、または[End Form On Add]フラグがオンになっている場合は、次の処理を実行します。

     フォームのクローズを開始します。
   - オフの場合

     ダイアログのクリアを開始します。
9. フォームが更新モードの場合は、次の処理を実行します。
   - データベースの更新または追加中にエラーが発生しなかった場合は、次の処理を実行します。

     フォームのクローズを開始します。

Cancel

[Cancel]は、修正/検査フォームに自動的に設定される標準項目です。

1. 次のイベント・ルールを実行します。:Button Clicked
2. 次のイベント・ルールを実行します。:Post Button Clicked
3. フォームのクローズを開始します。

ユーザー定義項目

ユーザー定義項目は、修正/検査フォームに追加できる非標準項目で、標準項目では対応できない特別な処理を実行します。

1. 次のイベント・ルールを実行します。:Button Clicked
2. 次のイベント・ルールを実行します。:Post Button Clicked

見出し詳細フォームのプロセス・フロー

見出し詳細フォームは、ヘッダーの情報とグリッドの情報が関係しているときに使用します。ヘッダー部は 1 つのビジネス・ビューを使用し、フォームの詳細部は別のビジネス・ビューを使用します。このフォーム・タイプと見出しなし詳細フォーム・タイプは、トランザクション・フォームと呼ばれます。ここでは、見出し詳細フォーム・タイプのプロセス・フローについて説明します。

デフォルトのフラグ

このフォーム・タイプでは、フォーム・オプション・フラグは自動的に設定されません。フォーム・オプション・フラグはいずれも、アプリケーションの開発者が任意に設定することができます。

ダイアログの初期化

1. スレッド処理を初期化します。
2. エラー処理プロセスを初期化します。
3. ビジネス・ビュー・カラムを初期化します。
4. フォーム・コントロールを初期化します。
5. グリッド・フィールドを初期化します。
6. Static テキストを初期化します。
7. ヘルプを初期化します。
8. イベント・ルール構造体を初期化します。
9. ツールバーを作成します。
10. フォーム・インターコネクト・データを、対応するビジネス・ビュー・カラムとフィルタ用フィールドに必要に応じてロードします。
11. 次のイベント・ルールを実行します。:Dialog is Initialized
12. フォーム・オプション・フラグの[No Fetch On Form Business View]がオンの場合は、次の処理を実行します。
    - フォームが追加モードでなく[Copy]ボタンによって呼び出されていない場合は、次の処理を実行します。

      次のイベント・ルールを実行します。:Post Dialog is Initialized
    - フォーム・コントロール値を画面にコピーします。
13. フォームが更新モードの場合は([Copy]ボタンによって呼び出されたフォームを含む)、次の処理を実行します。
    - フォーム・オプション・フラグの[No Fetch On Form Business View]がオンの場合は、次の処理を実行します。

      詳細データの選択と順序設定を開始します。

- フォーム・オプション・フラグの[No Fetch On Form Business View]がオフの場合は、次の処理を実行します。

    ヘッダー・データの取出しを開始します。

14. フォームが追加モードの場合は、次の処理を実行します。
    - ダイアログのクリアを開始します。

ヘッダー・データの取出し

ヘッダーのビジネス・ビュー・カラム、またはフィルタ用フィールドに基づいて、ヘッダーのビジネス・ビューに対するキー構造体が作成されます。次にデータベースが呼び出され、指定されたレコードの取出しが試みられます。

1. データベースのフェッチに成功すると、次の処理を実行します。
    - フェッチされたデータベース・レコードを、ヘッダーのビジネス・ビュー・カラムにコピーします。
    - フォームが[Copy]ボタンによって開かれていない場合は、次の処理を実行します。

        次のイベント・ルールを実行します。:Post Dialog is Initialized(フォーム・コントロールとビジネス・ビュー・カラムにメモリが割り当てられている場合、フォーム・コントロールは、このイベントの発生時に、文字列や文字属性フィールドの場合にブランクを意味する値をデータベースから受け取ることがあるので注意してください)
    - ビジネス・ビュー・カラムをフォーム・コントロールにコピーします。
    - グリッド・オプションの[Automatically Find on Entry]がオンの場合は、次の処理を実行します。

        詳細データの選択と順序設定を開始します。
    - グリッド・オプションの[Automatically Find on Entry]がオフの場合は、次の処理を実行します。

        入力ローのグリッドへの追加を開始します。
2. データベースのフェッチに失敗した場合は、次の処理を実行します。
    - ダイアログのクリアを開始します。

詳細データの選択と順序設定

ここで内部構造体が作成され、ユーザーによって指定された詳細部のデータ選択条件とデータ順序設定条件の式が形成されます。この構造体はその後、データベースの実際の選択と順序設定を行うためにデータベース・エンジンに送られます。データは次のステップで取り出されます。

選択用のデータは、フィルタ用フィールドか、フィルタ用フィールドが存在しない場合は、キーであるビジネス・ビュー・カラムから取り出されます。

1. フォーム・オプション・フラグの[No Fetch On Grid Business View]がオフの場合は、次の処理を実行します。
    - 選択と順序設定を実行します。
    - データの取出しを開始します。

2. フォーム・オプション・フラグの[No Fetch On Grid Business View]がオンの場合は、次の処理を実行します。
    - 入力ローのグリッドへの追加を開始します。

データの取出し

要求が JDEKRNL に送られ、データベースからデータを実際にフェッチします。この場合、システムはレコードを 1 つずつ読み取り、レコードごとに次の処理を実行します。

1. データベースからの詳細レコードのフェッチを試みます。
2. レコードがフェッチされた場合は、次の処理を実行します。
    - 詳細ビジネス・ビュー・データ構造体にデータをコピーします。
    - 次のイベント・ルールを実行します。: Grid Record is Fetched
    - アプリケーションの開発者が、このグリッド・レコードの書出しを抑制していない場合は、次の処理が実行されます。

        ビジネス・ビュー・データをグリッド・データ構造体にコピーします。

        次のイベント・ルールを実行します。: Write Everest Grid Line – Before

        ローをグリッドに追加します。この時点で、グリッド・コントロールにローが書かれます。

        次のイベント・ルールを実行します。: Write Everest Grid Line – After
    - 次のレコードを読み取るためにグリッド・データ構造体をクリアします。
    - グリッド・ラインの抑制フラグを解除します。

以上のステップは、データベースから読み取られたレコードごとに発生します。すべてのレコードが読み取られた後、ランタイム・エンジンは次の操作を実行します。

3. フォームが[Copy]ボタンによって開かれた場合は、次の処理を実行します。
    - 追加モードに切り替えます。
    - 詳細データ取出し処理が完了すると、ダイアログのクリアを開始します。
4. レコードがフェッチされなかった場合は、次の処理を実行します。
    - 追加モードに切り替えます。
5. 次のイベント・ルールを実行します。: Last Grid Record Has Been Read

    （このイベントは、レコードが実際にフェッチされたかどうかに関係なく実行されます。）
6. 入力ローのグリッドへの追加を開始します。

ダイアログのクリア

1. フォームがコピー・モードで呼び出された場合は、次の処理を実行します。
   - [Do Not Clear After Add]フラグがオフであるキー・コントロールをクリアします。
2. フォームがコピー・モードで呼び出されなかった場合は、次の処理を実行します。
   - [Do Not Clear After Add]フラグがオフであるすべてのフォーム・コントロールをクリアします。
3. 次のイベント・ルールを実行します。:Clear Screen Before Add
4. 既存のローを削除します。
5. 次のイベント・ルールを実行します。:Post Dialog is Initialized
6. 入力ローのグリッドへの追加を開始します。

入力ローのグリッドへの追加

1. グリッド・データ構造体をクリアします。
2. 次のイベント・ルールを実行します。:Add Last Entry Row to Grid
3. ローをグリッド・コントロールに追加します。

フォームのクローズ

1. 次のイベント・ルールを実行します。:End Dialog
2. フォーム・インターコネクト・データを、対応するビジネス・ビュー・カラムから必要に応じてロードします。
3. エラー処理を終了します。
4. スレッド処理を終了します。
5. ヘルプを終了します。
6. ビジネス・ビュー・カラム、フォーム・コントロール、グリッド・カラム、およびイベント・ルールのそれぞれの構造体を含む、フォームの構造体をすべて解放します。
7. ウィンドウを消去します。

メニュー/ツールバー項目

ここでは、見出し詳細フォームのメニュー/ツールバー項目のプロセス・フローについて説明します。

OK

[OK]は、検索/表示フォームに自動的に設定される標準項目です。この項目は、フォームの情報を検査し、JDEKRNL の関数呼出しを通じてデータベースに更新または追加します。

1. フォームにエラーまたは警告が表示される場合は、[OK]の処理を停止します。
2. 次のイベント・ルールを実行します。:Button Clicked

3. フォームのコントロールごとに、次の処理を実行します。
   - 現在のコントロールがフォーム・コントロールで、検査をパスしていない場合は、次の処理を実行します。

     次のイベント・ルールを実行します。: Control is Exited

     次のイベント・ルールを実行します。: Control is Exited and Changed – Inline

     次のイベント・ルールを実行します。: Control is Exited and Changed – Asynch

     データ辞書を検査します。
   - 現在のコントロールがグリッド・コントロールの場合は、次の処理を実行します。

     グリッド・ローごとに、次の処理を実行します。

     現在のグリッド・ローで Leave Row 処理を実行する必要があり、ローが更新可能な場合は、次の処理を実行します。

     次のイベント・ルールを実行します。: 現在のローに対する Row is Exited and Changed – Inline

     次のイベント・ルールを実行します。: 現在のローに対する Row is Exited and Changed – Asynch
4. フォームにエラーが表示される場合は、[OK]の処理を停止します。
5. 削除スタックにあるグリッド・ローをすべてデータベースから削除します。削除スタックの各グリッド・ローで、次の処理を実行します。
   - 削除スタックからグリッド・データ構造体に情報をコピーします。
   - グリッド・データ構造体をビジネス・ビュー・データ構造体にコピーします。
   - 次のイベント・ルールを実行します。: Delete Grid Record from DB – Before
   - データベースの削除が抑制されていない場合は、次の処理を実行します。

     ビジネス・ビューのレコードをデータベースから削除します。

     グリッド・コントロールからグリッド・ローを削除します。

     次のイベント・ルールを実行します。: Delete Grid Record from DB – After
6. 次のイベント・ルールを実行します。: All Grid Recs Deleted from DB
7. フォーム・オプション・フラグの[No Update On Form Business View(フォーム・ビジネス・ビューの更新なし)]がオフの場合は、次の処理を実行します。
   - フォームが追加モードの場合は、次の処理を実行します。

     次のイベント・ルールを実行します。: Add Record to Database – Before

     データベースの追加が抑制されていない場合は、次の処理を実行します。

     データベースにヘッダーのビジネス・ビュー・レコードを追加します。
   - 次のイベント・ルールを実行します。: Add Record to Database – After
   - フォームが更新モードの場合は、次の処理を実行します。

     次のイベント・ルールを実行します。: Update Record to Database – Before

     データベースの更新が抑制されていない場合は、次の処理を実行します。

     データベースに対してレコードを更新します。

次のイベント・ルールを実行します。: Update Record to Database – After

8. フォーム・オプション・フラグの[No Update On Grid Business View(グリッド・ビジネス・ビューの更新なし)]がオンの場合は、次の処理を実行します。
   - 変更または追加された各グリッド・ローで、次の処理を実行します。

     ビジネス・ビュー・データ構造体をクリアします。

     このローの元のキー値をビジネス・ビュー・データ構造体にリセットします。

     グリッド・データ構造体をビジネス・ビュー・データ構造体にコピーします。

     すべての非フィルタ・データベース・フォーム・コントロールをビジネス・ビュー・データ構造体にコピーします。

     すべての「イコール」フィルタをビジネス・ビュー・データ構造体にコピーします。

     フォームが追加モードの場合は、次の処理を実行します。

     次のイベント・ルールを実行します。: Add Grid Rec to DB – Before

     ビジネス・ビュー・データ構造体のレコードをデータベースに追加します。

     次のイベント・ルールを実行します。: Add Grid Rec to DB – After

     フォームが更新モードの場合は、次の処理を実行します。

     次のイベント・ルールを実行します。: Update Grid Rec to DB – Before

     ビジネス・ビュー・データ構造体のレコードをデータベースに更新します。

     次のイベント・ルールを実行します。: Update Grid Rec to DB – After
   - フォームが追加モードの場合は、次の処理を実行します。

     次のイベント・ルールを実行します。: All Grid Recs Added to DB
   - フォームが更新モードの場合は、次の処理を実行します。

     次のイベント・ルールを実行します。: All Grid Recs Updated to DB
9. 次のイベント・ルールを実行します。: Post Button Clicked
10. フォームが追加モードの場合は、次の処理を実行します。
    - フォームがコピー・モードで呼び出されたか、または[End Form On Add]フラグがオンの場合は、次の処理を実行します。

      フォームのクローズを開始します。

      オフの場合

      ダイアログのクリアを開始します。
11. フォームが更新モードで、データベースに対する更新または追加中にエラーが発生しなかった場合は、次の処理を実行します。
    - フォームのクローズを開始します。

Cancel

[Cancel]は、検索/表示フォームに自動的に設定される標準項目です。

1. 次のイベント・ルールを実行します。:Button Clicked
2. 次のイベント・ルールを実行します。:Post Button Clicked
3. フォームのクローズを開始します。

Delete

[Delete]は、見出し詳細フォームに追加可能な標準項目です。削除は、選択したグリッド・レコードに適用されます。この時点では、データベースから実際に削除されません。そのため、ユーザーが[Cancel]をクリックしてもレコードはデータベースから削除されません。[Delete]が押されると削除の確認が行われ、データベースからの実際の削除は[OK]ボタンが押されたときに行われます。

4. 次のイベント・ルールを実行します。:Button Clicked
5. 選択された削除可能な各グリッド・ローで、次の処理を実行します。
   - [Suppress Delete]フラグを解除します。
   - 次のイベント・ルールを実行します。:Delete Grid Rec Verify – Before
   - [Suppress Delete]フラグが設定されていない場合は、次の処理が実行されます。

     [Delete Confirmation]ダイアログを表示します。

     ユーザーが[No(いいえ)]または[Cancel(取消し)]をクリックすると、この処理はスキップされます。

     次のイベント・ルールを実行します。:Delete Grid Rec Verify – After

     [Suppress Delete]フラグが設定されていない場合は、次の処理が実行されます。

     レコードがデータベースから読み取られた場合は、次の処理を実行します。

     このレコードを、削除スタック(ユーザーが[OK]をクリックした場合に削除されるレコード)に追加します。

     グリッド・コントロールからグリッド・ローを削除します。

Find

[Find]は、見出し詳細フォームに追加可能な標準項目です。この項目をユーザーがクリックすると、ランタイム・エンジンに信号が送られ、ランタイム・エンジンがデータベースを呼び出し、フィルタ用フィールドの選択条件に基づいてグリッドを再ロードします。

1. 次のイベント・ルールを実行します。:Button Clicked
2. フィルタ用フィールドにエラーがない場合は、次の処理が実行されます。
   - 更新モードに切り替えます。
   - 詳細データの選択と順序設定を開始します。
   - 次のイベント・ルールを実行します。:Post Button Clicked

ユーザー定義項目

ユーザー定義項目は、見出し詳細フォームに追加できる非標準項目で、標準項目では対処できない特別な処理を実行します。

1. 次のイベント・ルールを実行します。:Button Clicked
2. 次のイベント・ルールを実行します。:Post Button Clicked

見出しなし詳細フォームのプロセス・フロー

見出しなし詳細フォームは、ヘッダーの情報と詳細領域の情報が関係しているときに使用します。ヘッダー部は 1 つのビジネス・ビューを使用し、フォームの詳細部は別のビジネス・ビューを使用します。このフォーム・タイプと見出し詳細フォーム・タイプは、トランザクション・フォームと呼ばれます。ここでは、見出しなし詳細フォーム・タイプのプロセス・フローについて説明します。

デフォルトのフラグ

見出しなし詳細フォームの場合は、次のフォーム・オプション・フラグが自動的にオンに設定され、オフにすることはできません。

- No Update On Form Business View
- No Fetch On Form Business View

その他のフォーム・オプション・フラグは、アプリケーションの開発者が任意に設定することができます。

ダイアログの初期化

1. スレッド処理を初期化します。
2. エラー処理プロセスを初期化します。
3. ビジネス・ビュー・カラムを初期化します。
4. フォーム・コントロールを初期化します。
5. グリッド・フィールドを初期化します。
6. Static テキストを初期化します。
7. ヘルプを初期化します。
8. イベント・ルール構造体を初期化します。
9. ツールバーを作成します。
10. フォーム・インターコネクト・データを、対応するビジネス・ビュー・カラムとフィルタ用フィールドに適宜ロードします。
11. 次のイベント・ルールを実行します。:Dialog is Initialized

12. フォームが追加モードでない場合は、次の処理を実行します。

    - フォームがコピー・モードでない場合

      次のイベント・ルールを実行します。: Post Dialog is Initialized（更新モードでは、このイベントは Dialog is Initialized の直後に実行されるため、FC 値は NULL またはゼロのままであることに注意してください）

    - グリッド・オプションの［Automatically Find on Entry］がオンの場合は、次の処理を実行します。

      データの選択と順序設定を開始します。

    - グリッド・オプションの［Automatically Find on Entry］がオフの場合は、次の処理を実行します。

      入力ローのグリッドへの追加を開始します。

    - フォームが追加モードの場合は、次の処理を実行します。

      ダイアログのクリアを開始します。

データ選択およびデータ順序設定

ここで内部構造体が作成され、ユーザーによって指定されたデータ選択条件とデータ順序設定条件の式が形成されます。これは、その後、データベースの実際の選択と順序設定を行うためにデータベース・エンジンに送られます。データは、次のステップで取り出されるまで保管されます。

選択用のデータは、フィルタ用フィールドか、フィルタ用フィールドが存在しない場合は、キーであるビジネス・ビュー・カラムから取り出されます。

1. フォーム・オプション・フラグの［No Fetch On Grid Business View］がオフの場合は、次の処理を実行します。
   - 選択と順序設定を実行します。
   - データの取出しを開始します。
2. フォーム・オプション・フラグの［No Fetch On Grid Business View］がオンの場合は、次の処理を実行します。
   - 入力ローのグリッドへの追加を開始します。

データの取出し

要求が JDEKRNL に送られ、データベースからデータを実際にフェッチします。この場合、レコードを 1 つずつ読み取り、レコードごとに次の処理を実行します。

1. データベースからのレコードのフェッチを試みます。
2. レコードがフェッチされた場合
   - ビジネス・ビュー・データ構造体にデータをコピーします。
   - 次のイベント・ルールを実行します。: Grid Record is Fetched
   - アプリケーションの開発者が、このグリッド・レコードの書出しを抑制していない場合は、次の処理が実行されます。

     ビジネス・ビュー・データをグリッド・データ構造体にコピーします。

次のイベント・ルールを実行します。:Write Grid Line – Before

ローをグリッドに追加します。この時点で、グリッド・コントロールにローが書かれます。

次のイベント・ルールを実行します。:Write Grid Line – After

- 次のレコードを読み取るためにグリッド・データ構造体をクリアします。
- グリッド・ラインの抑制フラグを解除します。

以上のステップは、データベースから読み取られたレコードごとに発生します。すべてのレコードが読み取られた後、ランタイム・エンジンは次の操作を実行します。

3. フォームがコピー・モードの場合は、次の処理を実行します。
   - 追加モードに切り替えます。
   - 処理データの取出し処理が完了すると、ダイアログのクリアを開始します。
4. レコードがフェッチされなかった場合は、次の処理を実行します。
   - 追加モードに切り替えます。
5. 次のイベント・ルールを実行します。:Last Grid Record Has Been Read

   注:
   このイベントは、レコードが実際にフェッチされたかどうかに関係なく実行されます。

6. 入力ローのグリッドへの追加を開始します。

ダイアログのクリア

1. フォームがコピー・モードで呼び出された場合は、次の処理を実行します。
   - [Do Not Clear After Add]フラグがオフであるキー・コントロールをクリアします。
2. フォームがコピー・モードで呼び出されなかった場合は、次の処理を実行します。
   - [Do Not Clear After Add]フラグがオフであるフォーム・コントロールをすべてクリアします。
3. 次のイベント・ルールを実行します。:Clear Screen Before Add
   - 既存のグリッド・ローを削除します。
4. 次のイベント・ルールを実行します。:Post Dialog is Initialized
   - 入力ローのグリッドへの追加を開始します。

入力ローのグリッドへの追加

1. グリッド・データ構造体をクリアします。
2. 次のイベント・ルールを実行します。:Add Last Entry Row to Grid
3. ローをグリッド・コントロールに追加します。

フォームのクローズ

1. 次のイベント・ルールを実行します。: End Dialog
2. フォーム・インターコネクト・データを、対応するビジネス・ビュー・カラムから適宜ロードします。
3. エラー処理を終了します。
4. スレッド処理を終了します。
5. ヘルプを終了します。
6. ビジネス・ビュー・カラム、フォーム・コントロール、イベント・ルールのそれぞれの構造体を含む、フォームのすべての構造体を解放します。
7. ウィンドウを消去します。

メニュー/ツールバー項目

ここでは、見出しなし詳細フォームのメニュー/ツールバー項目のプロセス・フローについて説明します。

OK

[OK]は、見出しなし詳細フォームに自動的に設定される標準項目です。この項目は、フォームの情報を検査し、JDEKRNL の関数呼出しを通じてデータベースに更新または追加します。

1. フォームにエラーまたは警告が表示される場合は、[OK]の処理を停止します。
2. 次のイベント・ルールを実行します。: Button Clicked
3. フォームの各コントロールで、次の処理を実行します。
   - 現在のコントロールがフォーム・コントロールで、検査をパスしていない場合は、次の処理を実行します。

     次のイベント・ルールを実行します。: Control is Exited

     次のイベント・ルールを実行します。: Control is Exited and Changed – Inline

     次のイベント・ルールを実行します。: Control is Exited and Changed – Asynch

     データ辞書を検査します。
   - 現在のコントロールがグリッド・コントロールの場合、各々のグリッド・ローで以下の操作を実行します。

     次のイベント・ルールを実行します。: 現在のローに対する Row is Exited and Changed – Inline

     次のイベント・ルールを実行します。: 現在のローに対する Row is Exited and Changed – Asynch
4. フォームにエラーがある場合は、[OK]の処理を停止します。

5. 削除スタックにあるグリッド・ローをすべてデータベースから削除します。詳しくは、Delete を参照してください。削除スタックの各グリッド・ローで、
   - グリッド・ロー・データをビジネス・ビュー構造体にコピーします。
   - グリッド・データ構造体をビジネス・ビュー・データ構造体にコピーします。
   - 次のイベント・ルールを実行します。: Delete Grid Record from DB – Before
   - データベースの削除が抑制されていない場合は、次の処理を実行します。

     ビジネス・ビューのレコードをデータベースから削除します。

     グリッド・コントロールからグリッド・ローを削除します。

     次のイベント・ルールを実行します。: Delete Grid Record from DB – After
6. 次のイベント・ルールを実行します。: All Grid Recs Deleted from DB
7. フォーム・オプション・フラグの[No Update On Grid Business View]がオフの場合は、次の処理を実行します。
   - 変更または追加された各グリッド・ローで、次の処理を実行します。

     ビジネス・ビュー・データ構造体をクリアします。

     このローの元のキー値をビジネス・ビュー・データ構造体にリセットします。

     グリッド・データ構造体をビジネス・ビュー・データ構造体にコピーします。

     すべての非フィルタ・データベース・フォーム・コントロールをビジネス・ビュー・データ構造体にコピーします。

     すべての「イコール」フィルタをビジネス・ビュー・データ構造体にコピーします。

     フォームが追加モードの場合は、次の処理を実行します。

       次のイベント・ルールを実行します。: Add Grid Rec to DB – Before

       ビジネス・ビュー・データ構造体のレコードをデータベースに追加します。

       次のイベント・ルールを実行します。: Add Grid Rec to DB – After

     フォームが更新モードの場合は、次の処理を実行します。

       次のイベント・ルールを実行します。: Update Grid Rec to DB – Before

       ビジネス・ビュー・データ構造体のレコードをデータベースに更新します。

       次のイベント・ルールを実行します。: Update Grid Rec to DB – After
   - フォームが追加モードの場合は、次の処理を実行します。

     次のイベント・ルールを実行します。: All Grid Recs Added to DB
   - フォームが更新モードの場合は、次の処理を実行します。

     次のイベント・ルールを実行します。: All Grid Recs Updated to DB
8. 次のイベント・ルールを実行します。: Post Button Clicked
9. フォームが追加モードの場合は、次の処理を実行します。
   - フォームがコピー・モードで呼び出された場合、または[End Form On Add]がオンになっている場合は、次の処理を実行します。

     フォームのクローズを開始します。

- オフの場合

    ダイアログのクリアを開始します。

10. フォームが更新モードの場合は、次の処理を実行します。
    - データベースの更新または追加中にエラーが発生しなかった場合は、次の処理を実行します。

        フォームのクローズを開始します。

Cancel

[Cancel]は、見出しなし詳細フォームに自動的に設定される標準項目です。

1. 次のイベント・ルールを実行します。:Button Clicked
2. 次のイベント・ルールを実行します。:Post Button Clicked
3. フォームのクローズを開始します。

Delete

[Delete]は、見出しなし詳細フォームに追加可能な標準項目です。この時点では、データベースから実際に削除されません。そのため、ユーザーが[Cancel]をクリックしても、レコードはデータベースから削除されません。[Delete]が押されると削除の確認が行われ、データベースからの実際の削除は、[OK]ボタンが押されたときに行われます。

1. 次のイベント・ルールを実行します。:Button Clicked
2. 選択された削除可能な各グリッド・ローで、次の処理を実行します。
    - [Suppress Delete]フラグを解除します。
    - 次のイベント・ルールを実行します。:Delete Grid Rec Verify - Before
    - [Suppress Delete]フラグが設定されていない場合は、次の処理が実行されます。

        [Delete Confirmation]ダイアログを表示します。

        ユーザーが[No]または[Cancel]をクリックすると、この処理は中断されます。

        次のイベント・ルールを実行します。:Delete Grid Rec Verify - After

        [Suppress Delete]フラグが設定されておらず、データベースからレコードが読み込まれた場合は、次の処理を実行します。

            このレコードを、削除スタック(ユーザーが[OK]をクリックした場合に削除されるレコード)に追加します。

    - グリッド・コントロールからグリッド・ローを削除します。

Find

[Find]は、見出しなし詳細フォームに追加可能な標準項目です。この項目をユーザーがクリックすると、ランタイム・エンジンに信号が送られ、ランタイム・エンジンがデータベースを呼び出し、フィルタ用フィールドの選択条件に基づいてグリッドを再ロードします。

1. 次のイベント・ルールを実行します。:Button Clicked
2. フィルタ用フィールドにエラーがない場合は、次の処理が実行されます。
   - 更新モードに切り替えます。
   - データの選択と順序設定を開始します。
   - 次のイベント・ルールを実行します。:Post Button Clicked

ユーザー定義項目

ユーザー定義項目は、見出しなし詳細フォームに追加できる非標準項目で、標準項目では対処できない特別な処理を実行します。

3. 次のイベント・ルールを実行します。:Button Clicked
4. 次のイベント・ルールを実行します。:Post Button Clicked

検索/選択フォームのプロセス・フロー

検索/選択フォームは、指定済みのファイルのレコードから既定の 1 つのフィールドを選択するために使用します。

重要

検索/選択フォームは、辞書エイリアスに基づいて、呼出し側フォームに値を 1 つのみ返します。データ辞書エイリアスが値に一致しない場合は、データ構造体から最初の値が返されます。

ここでは、検索/選択フォーム・タイプのプロセス・フローについて説明します。検索/選択フォームは、そのフォームのデータ構造体要素とデータ・タイプが同じデータ項目にのみ、ビジュアル・アシストとして関連付けてください。

デフォルトのフラグ

このフォーム・タイプでは、フォーム・オプション・フラグはいずれも自動的に設定されませんが、アプリケーションの開発者は多くの場合、グリッド・オプション・フラグの[Automatically Find on Entry]をオンに設定します。このフォーム・タイプに影響する唯一のフォーム・オプション・フラグは[No Fetch on Grid Business View]フラグです。他のフォーム・オプション・フラグを変更しても、フォームの処理には影響しません。

ダイアログの初期化

1. スレッド処理を初期化します。
2. エラー処理プロセスを初期化します。
3. ビジネス・ビュー・カラムを初期化します。
4. フォーム・コントロールを初期化します。
5. グリッド・フィールドを初期化します。
6. Static テキストを初期化します。
7. ヘルプを初期化します。
8. イベント・ルール構造体を初期化します。
9. ツールバーを作成します。
10. フォーム・インターコネクト・データを、対応するビジネス・ビュー・カラムとフィルタ用フィールドに適宜ロードします。
11. 次のイベント・ルールを実行します。: Dialog is Initialized
12. 次のイベント・ルールを実行します。: Post Dialog is Initialized
13. グリッド・オプションの[Automatically Find on Entry]がオンの場合は、次の処理を実行します。
    - 詳細データの選択と順序設定を開始します。

ヘッダー・データの取出し

検索/選択フォームには、ヘッダー・レコードは存在しません。

詳細データの選択と順序設定

ここで内部構造体が作成され、ユーザーによって指定されたデータ選択条件とデータ順序設定条件の式が形成されます。この構造体は、その後、データベースの実際の選択と順序設定を行うためにデータベース・エンジンに送られます。データは、次のステップで取り出されるまで保管されます。

選択用のデータは、フィルタ用フィールドから決定されます。

フォーム・オプション・フラグの[No Fetch On Grid Business View]がオフの場合は、次の処理を実行します。

- 選択と順序設定を実行します。
- データの取出しを開始します。

データの取出し

要求が JDEKRNL に送られ、データベースからデータを実際にフェッチします。この場合、レコードを 1 つずつ読み取り、レコードごとに次の処理を実行します。

1. データベースからのレコードのフェッチを試みます。
2. レコードがフェッチされた場合は、次の処理を実行します。
   - ビジネス・ビュー・データ構造体にデータをコピーします。
   - 次のイベント・ルールを実行します。: Grid Record is Fetched
   - アプリケーションの開発者が、このグリッド・レコードの書出しを抑制していない場合は、次の処理が実行されます。
     - ビジネス・ビュー・データをグリッド・データ構造体にコピーします。
     - 次のイベント・ルールを実行します。: Write Grid Line – Before
     - ローをグリッドに追加します。この時点で、グリッド・コントロールにローが書かれます。
     - 次のイベント・ルールを実行します。: Write Grid Line – After
   - 次のレコードを読み取るためにグリッド・データ構造体をクリアします。
   - グリッド・ラインの抑制フラグを解除します。

以上のステップは、データベースから読み取られたレコードごとに発生します。次の操作が 1 回だけ行われます。

- 次のイベント・ルールを実行します。: Last Grid Record Has Been Read

ダイアログのクリア

ダイアログのクリアは、このフォーム・タイプには適用されません。

フォームのクローズ

1. 次のイベント・ルールを実行します。: End Dialog
2. フォーム・インターコネクト・データを、対応するビジネス・ビュー・カラムから適宜ロードします。
3. エラー処理を終了します。
4. ヘルプを終了します。
5. ビジネス・ビュー・カラム、フォーム・コントロール、グリッド・カラム、およびイベント・ルールのそれぞれの構造体を含む、フォームの構造体をすべて解放します。
6. ウィンドウを消去します。

メニュー/ツールバー項目

ここでは、検索/選択フォームのメニュー/ツールバー項目のプロセス・フローについて説明します。

Select

[Select]は、検索/選択フォームに自動的に設定される標準項目です。この項目は、フォーム・インターコネクションに値を返してフォームを閉じます。

1. 選択されたグリッド・ローをビジネス・ビュー・カラムにコピーします。
2. 次のイベント・ルールを実行します。: Button Clicked
3. 次のイベント・ルールを実行します。: Post Button Clicked
4. フォームのクローズを開始します。

Close

[Close]は、検索/選択フォームに自動的に設定される標準項目です。

1. 次のイベント・ルールを実行します。: Button Clicked
2. 次のイベント・ルールを実行します。: Post Button Clicked
3. フォームのクローズを開始します。

Find

[Find]は、検索/選択フォームに追加可能な標準項目です。この項目をユーザーがクリックすると、ランタイム・エンジンに信号が送られ、ランタイム・エンジンがデータベースを呼び出し、フィルタ用フィールドの選択条件に基づいてグリッドを再ロードします。

1. 次のイベント・ルールを実行します。: Button Clicked
2. データの選択と順序設定を開始します。
3. 次のイベント・ルールを実行します。: Post Button Clicked

ユーザー定義項目

ユーザー定義項目は、検索/選択フォームに追加できる非標準項目で、標準ボタンでは対処できない特別な処理を実行します。

1. 次のイベント・ルールを実行します。: Button Clicked
2. 次のイベント・ルールを実行します。: Post Button Clicked

メッセージ・フォームのプロセス・フロー

メッセージ・フォーム・タイプは、ユーザーへの情報伝達や、確認用のセカンダリ・ウィンドウとして表示されるフォームです。このフォームの動作は、Windows メッセージ・ボックスの動作と同じです。このフォームには、ツールバーもステータス・バーもなく、スタティック・テキストとボタンのみを設定できます。これは、ツールバーの代わりに標準ボタンをフォームに表示できる唯一のフォーム・タイプです。

ダイアログの初期化

1. フォーム・コントロールを初期化します。
2. Static テキストを初期化します。
3. ヘルプを初期化します。
4. イベント・ルール構造体を初期化します。
5. 次のイベント・ルールを実行します。: Dialog is Initialized

フォームのクローズ

1. ヘルプを終了します。
2. コントロールとイベント・ルールのすべての構造体を解放します。
3. ウィンドウを消去します。

ボタン

このフォームには、デフォルトで[OK]ボタンが設定されます。他のオプションには、[Cancel]、[Yes]、[No]があります。これらのボタンはいずれもダイアログを閉じ、これ以外のデフォルト処理はありません。

J.D. Edwards 標準

メッセージ・フォームには、Static テキストとボタンだけを設定できます。イベント・ルールは、Dialog is Initialized とボタン・イベントのみに制限されています。このイベント・ルールには、フォーム・インターコネクト呼出しを含めることができません。

編集コントロールのプロセス・フロー

エディット・コントロールは、フォーム上のテキスト・フィールドです。すべてのフォーム・タイプは、メッセージ・フォームを除き、編集コントロールをもつことができます。編集コントロールには、2 つのタイプがあります。1 つは、一般的にはデータベース・フィールドです。これは、ビジネス・ビューの項目に関連付けられ、それを通じて特定のデータ辞書項目に接続します。データベース・フィールドは、データベース・レコードのフィールドを表します。もう 1 つのタイプは、通常はデータ辞書フィールドと呼ばれ、このコントロールも特定のデータ辞書項目に接続します。

データベース・フィールドは、さらにフィルタ用フィールドと非フィルタ用フィールドに分かれています。データベース・フィールドは、デフォルトでは非フィルタ用フィールドです。フィルタ用フィールドは、データベース・フェッチの選択条件を変更するために使用します。フィルタには、「等しい」、「等しくない」、「より小さい」、「以下」、「より大きい」、または「以上」の比較演算子を使用できます。フィルタは、選択されていないときにワイルドカード(*)を表示するように指定できます。

編集コントロールの値の保存は、数値、文字列、文字など、関連するデータ辞書項目のタイプに基づいています。編集コントロールは、関連するデータ辞書項目のプロパティの影響を受けます。たとえば、編集コントロールが、長さが 30 文字の文字列タイプのデータ辞書項目に関連している場合は、30 文字を超える文字をフィールドに入力できません。

プロパティとオプション

編集コントロールでは、次の表に示すプロパティとオプションを使用できます。これらのプロパティはすべてのフォーム・タイプで使用でき、データベース・フィールドとデータ辞書フィールドの両方で使用できます。ここでは、処理に影響するプロパティに限定して説明します。インターフェイスにのみ関連するプロパティについては、コントロールの処理に関係する範囲で説明します。

Disable	このプロパティが設定されているコントロールはグレー表示になり、ユーザーによって変更できません。無効なフィールドの値は、イベント・ルールを使用して変更できます。コントロールが無効なときは、関連するテキスト(左側の Static テキスト)も無効になります。このプロパティは、コントロールのデフォルトではありません。
Visible	このプロパティが設定されているコントロールは表示できますが、設定されていないコントロールは表示できません。非表示のコントロールの値は、イベント・ルールを使用して変更できます。
Do not clear after add	このオプションは、フォームが追加モードで、そのフォームをクリアするときにのみ適用されます。このオプションが設定されているコントロールは、それぞれの値を維持します。このオプションは、コントロールのデフォルトではありません。
Required entry field	このオプションを選択すると、レコードを保存(OK 処理)する前に、値をコントロールに入力することが必須になります。このオプションは、Null もブランクも受け入れません。このオプションは、コントロールのデフォルトではありません。
Default cursor on add/update mode	このオプションは、追加/更新モード時にカーソルが配置される位置を指定します。このオプションを割当て可能なコントロールは、フォーム上の 1 つのみです。複数のコントロールに割り当てると、システムはカーソル位置を判別できません。このオプションは、コントロールのデフォルトではありません。
No display if currency is Off	このオプションは、[Multicurrency Conversion(多通貨換算)]がオフのときにコントロールを非表示にします。このオプションは、コントロールのデフォルトではありません。

コントロールへフォーカスが当たった場合

4. 次のイベント・ルールを実行します。: Control is Entered

Windows クライアントでは、ユーザーが[Tab]キーを押すなどして、フォーカスをコントロール間で移動すると、フォーカスを失うコントロールが"Control is Exited"メッセージを受け取る前に、フォーカスを受け取るコントロールが「Control is Entered」メッセージを実際に受け取ります。これは、Windows メッセージの順序によるもので、変更はできません。

コントロールから出たとき

1. フォーカスが別のウィンドウに移ると、処理を停止します。

   このウィンドウに再びフォーカスが当たると、フォーカスは元のコントロールに戻ります。

2. 画面から内部ストレージにテキストをコピーします。

   このステップで、文字列は適切なタイプに変換されます。

3. このコントロールが入力必須フィールドとしてマークされていて、値が入力されていない場合は、フィールドをエラー状態に設定します。
4. 次のイベント・ルールを実行します。: Control is Exited.
5. このコントロールが初めて終了された場合、または前回終了されてこのコントロールの値が変更された場合は、次の処理を実行します。
   - Begin Control is Exited and Changed (Inline)

コントロールから出て、値が変更された場合(インライン部)

1. 次のイベント・ルールを実行します。: Control is Exited and Changed Inline
2. フォームが閉じていない場合は、次の処理を実行します。
   - スレッド上で Control is Exited and Changed (Asynchronous)の実行を試みます。
3. フォームが閉じていたり非同期の実行に失敗した場合は、次の処理を実行します。
   - インライン関数として Control is Excited and Changed (Asynchronous)を開始します。

コントロールから出て、値が変更された場合(非同期部)

1. 次のイベント・ルールを実行します。: Control is Exited and Changed Asynch
2. イベント・ルールの処理時にエラーが設定されなかった場合は、次の処理を実行します。
   - このフィールドがフィルタ用フィールドでない場合、または比較タイプが「イコール」のフィルタ用フィールドの場合(検査は、比較タイプが「イコール」のフィルタ用フィールドでのみ行われます)は、データ辞書の妥当性検査が実行されます。
   - このコントロールに関連する記述フィールドがある場合は、それを自動入力します(エラーが発生した場合は、関連記述をクリアします)。
   - 妥当性検査でエラーが発生しなかった場合は、コントロールに値をコピーします。

J.D. Edwards 標準

フィルタ用フィールドには、いずれも「Display Wildcard（表示ワイルドカード）」を設定しています。インデックス・フィールドのみをフィルタ用フィールドとして指定します（パフォーマンス上の理由）。必要な場合は、イベント・ルールを、インライン・イベントではなく Control is Exited and Changed Asynch に置く必要があります（パフォーマンス上の理由）。

グリッド・コントロールのプロセス・フロー

グリッド・コントロールは、フォーム上のスプレッドシートのようなものです。グリッド・コントロールは、次のフォーム・タイプに使用することができます。

- 検索/表示
- 検索/選択
- 見出し詳細
- 見出しなし詳細

グリッド・コントロールにはカラムがあります。カラムには次の 2 つのタイプがあり、設計時にどちらかを指定します。

- データベース・カラム
- データ辞書カラム

データベース・カラムは、ビジネス・ビューの項目に関連付けられ、それを通じて辞書項目に接続します。データベース・カラムは、データベース・レコードのフィールドを表します。

データ辞書カラムと呼ばれるタイプは 1 つだけですが、この 2 つのタイプはいずれも特定のデータ辞書項目に接続しています。ただしデータベース・カラムは、ビジネス・ビュー・フィールドにも接続している点で異なります。

グリッドには、表示グリッドと更新グリッドがあります。表示グリッドは表示専用で、セルを個別に選択できません。検索/表示フォームと検索/選択フォームには、表示グリッドがあります。

更新グリッドを使用すると、レコードを追加/更新できます。更新グリッドのセルは、個別に選択できます。見出し詳細フォーム/見出しなし詳細フォームには、更新グリッドがあります。

グリッド・コントロールには、QBE（例示照会プログラム）行もあります。QBE カラムは、グリッド・カラムに 1 対 1 で対応します。データベース・フェッチの選択条件を変更するには、QBE を使用します。QBE カラムには、データベース・カラムでのみ入力できます。それは、選択に作用することが QBE の目的であり、ビジネス・ビューにはデータベース・カラムしか存在しないからです。QBE カラムでは、次のいずれかの比較タイプを入力できます。

- Equal（等しい）
- Not equal（等しくない）
- Less than（より小さい）

- Less than or equal to（以下）
- Greater than（）より大きい
- Greater than or equal to（以上）

特別に指定しない限り、比較タイプは同等になります。QBE で比較タイプを入力するか、またはシステム関数を使用して指定できます。ワイルドカード（*または%）を使用すると、文字列フィールドのあいまい検索を行えます。

グリッド・ローに含まれている値は、論理単位として機能します。グリッド・ローは、レコードを受け入れる前に検査する必要がありますが、個々のカラムの検査は不要です。編集コントロールの検査とグリッド・ローの検査は対応しますが、編集コントロールの検査とグリッド・カラムの検査は対応しません。Column is Exited イベントは、ユーザーがセルから出るという操作をした場合のみ実行されます。セルのデータ辞書検査は、セルを変更した後に出たとき、またはローを追加した後にローを初めて出たときに実行されます。プログラムに基づいてセルを変更した場合、Row is Exited イベントは、レコードを受け入れる前に実行されますが、カラムのイベントと妥当性検査は、フォーカスがカラムに物理的に設定されない限り実行されません。

他社のスプレッドシートには、グリッド・セル値がタブ区切り文字列として（1 ローごとに 1 つ）保管されます。この値は、セルごとにまたはローごとに取り出すことができます。グリッド・カラムの内部ストレージは、対話型エンジンにもあります。グリッド・カラム値の実際のストレージは、関連する辞書項目のタイプ（数値、文字列、文字など）に基づき、値の画面表示から区別されます。1 つのローのデータのみが所定の時間に使用できます。各イベントは、特定のローのコンテキストで実行されます。

グリッド・カラムは、関連する辞書項目のプロパティによって影響されます。たとえば、グリッド・カラムが長さ 30 文字の文字列タイプの辞書項目に関連付けられている場合は、31 文字以上をセルに入力できません。

プロパティとオプション

グリッド・コントロールでは、次の表に示すプロパティとオプションを使用できます。注：が記されている箇所を除き、これらのプロパティはすべてのグリッドで使用できます。ここでは、処理に影響するプロパティに限定して説明します。インターフェイスにのみ関連するプロパティについては、グリッドの処理に関連する範囲で説明します。

Disable	このプロパティが設定されているグリッドはグレー表示になり、ユーザーによって変更できません。アプリケーションの開発者は、イベント・ルール(ER)を通じて、無効にされたフィールドの値を変更できます。このプロパティは、グリッドのデフォルトではありません。
Visible	このプロパティが設定されているグリッドは表示できますが、設定されていないグリッドは表示できません。アプリケーションの開発者は、ER を通じて、非表示のグリッドの値を変更できます。
Hide Query By Example	このプロパティが設定されているグリッドでは、QBE 行がありません。QBE は、ユーザーに対しても、ER に対しても無効になります。このプロパティは、表示グリッドのデフォルトではありませんが、更新グリッドのデフォルトです。
Update Mode	このプロパティは更新グリッドにのみ適用されますが、実行時にグリッドに影響しません。
Multiple Select	このプロパティを設定すると、複数のグリッド・ローを同時に選択できます。これは、ER の実行と削除に影響します。このオプションは、グリッドのデフォルトではありません。

Automatically Find On Entry	このオプションは、フォームを開いたときにグリッド・レコードをフェッチするかどうかを指定します。このオプションが設定されていないグリッドは、フォームを開いたときにグリッド・ローが表示されません。このプロパティは、グリッドのデフォルトではありません。
Auto Find On Changes	このオプションは、レコードを変更した子フォームを閉じた後にグリッド・レコードをフェッチするかどうかを指定します。このオプションは、モードレス・フォーム・インターコネクトが存在しないフォームでのみ使用してください。このプロパティは、グリッドのデフォルトではありません。
No Adds on Update Grid	このプロパティは更新にのみ適用されます。グリッドに入力ローを表示するかどうかを指定します。入力ローが表示されないと、レコードを追加できません。このプロパティをオンに設定すると、既存のレコードのみを変更できます（レコードの更新のみ可能です）。このオプションは、グリッドのデフォルトではありません。
Disable Page-At-A-Time Processing	このオプションを設定すると、[Find]をクリックしたときに使用可能なすべてのグリッド・レコードがフェッチされます。このオプションを設定しないと、グリッド・レコードの最初のページのみがフェッチされ、その後、ユーザーがページを下にスクロールすると、追加のレコードが表示されます。追加のレコードを要求するたびに、1 ページ分のデータが返されます。これにより、大型のファイルではパフォーマンスが大幅に向上しますが、このオプションをオンにする際は十分考慮してください。このオプションは、グリッドのデフォルトではありません。
Clear Grid After Add	現在、このフィールドは実行時に参照されません。このフラグの有無は、グリッドに影響しません。
Refresh Grid After Update	現在、このフィールドは実行時に参照されません。このフラグの有無は、グリッドに影響しません。
Process All Rows In Grid	このオプションを設定すると、すべてのグリッドの各ローで少なくとも一度は Row is Exited、Row is Exited and Changed（非同期部）、および Row is Exited Validation という 3 つの Row is Exited イベントが実行されてから、データベース・レコードが更新または追加されます。

グリッドにフォーカスが当たるとき

次のイベント・ルールを実行します。: Set Focus on Grid

グリッドからフォーカスを外すとき

次のイベント・ルールを実行します。:Kill Focus on Grid

ローへの入力を開始するとき

1. ステータス・バーを更新して、現在のロー番号を表示します。
2. ローが入力ロー（更新グリッドの最後のロー）でない場合は、次の処理を実行します。
   - 次のイベント・ルールを実行します。:Row is entered

ローから出たとき

1. フォームがフォーカスを失っていない場合
   - 次のイベント・ルールを実行します。: Row is exited
   - グリッドが更新グリッドで、ローが前回終了されてから変更された場合は、次の処理を実行します。

     次のイベント・ルールを実行します。: Row is Exited and changed – Inline

     グリッドから離れていない場合（1 つのローを終了し、別のローに入る場合）は、次の処理を実行します。

     スレッド上で Row is Exited and Changed (Asynchronous portion)の実行を試みます。
   - グリッドから離れたり非同期の実行が失敗した場合は、次の処理を実行します。

     Row is Excited and Changed (Asynchronous portion)をインライン関数として開始します。

ローの内容を変更してローから出たとき（非同期部）

1. グリッド・ローのヘッダーにビットマップを設定し、このローが処理中であることを示します。
2. 次のイベント・ルールを実行します。: Row is Exited and Changed – Asynch
3. このローを追加してから初めて終了する場合は、次の処理を実行します。
   - Row is Exited Validation を開始します。

ローから出たときの妥当性検査

このローのうちデータ辞書の検査をまだ実行していない各グリッド・セルで、次の処理を実行します。

1. このデータベース項目がフォームのヘッダー部でもある場合、グリッド・カラムはその値から自動入力されるので、この検査をスキップして次のグリッド・セルに進みます。

   注:
   フィルタ用フィールドが「イコール」フィルタの場合にのみ、グリッド・カラムに自動入力されます。

2. セルに格納されているテキストを保管できる場合（数値にアルファベットが含まれておらず、範囲外のデータ・パラメータが存在しない場合など）は、次の処理を実行します。
   - このセルでデータ辞書の検査を開始します。

     このセルの関連記述カラムが存在する場合は、データ辞書の検査から返された情報をそこに自動入力します。

セルの内容を変更して、そのセルから出たとき

更新グリッドで、セルの内容を(ユーザーのキー入力またはビジュアル・アシストによって)変更し、そのセルが終了された場合は、次の処理を実行します。

1. Row is Exited Validation イベントと Column is Exited イベントを呼び出す前に、このセルに設定されているエラーをすべてクリアします。
2. セルに格納されているテキストを保管できる場合(数値にアルファベットが含まれておらず、範囲外のデータ・パラメータが存在しない場合)は、次の処理を実行します。
   - データ辞書の検査を実行します。
   - このセルの関連記述カラムが存在する場合は、データ辞書の検査から返された情報をそこに自動入力します。

グリッド・ローをダブルクリックするとき

表示グリッドで、グリッド・ローをダブルクリックすると、[Select]ボタンが押されたことになります。

キーを押すとき

更新グリッドで、入力ロー(グリッドの最後のロー)のキーを押すと、新しいローがグリッドに追加されます。

J.D. Edwards スタンダード

Row is Exited and Changed - Asynch イベントは、ローの内容を検査することと同じです。このイベントは、Edit Line マスター・ビジネス関数でよく使用されます。パフォーマンスへの影響を考慮して、ERは、1 つのインライン・イベントの代わりに Row is Exited and Changed - Asynch に設定してください。

日付参照スキャン

日付スキャン・プログラムを実行して、イベント・ルールまたはビジネス関数の日付参照を検索することができます。J.D. Edwards には、2 つの日付参照スキャン・プログラムが用意されています。一方のプログラムはビジネス関数をスキャンし、他方はイベント・ルールをスキャンします。ビジネス関数で日付参照をスキャンする場合は、システム日付の参照もスキャンされます。

ビジネス関数日付参照スキャン

ビジネス日付参照スキャンを実行すると、ビジネス関数の日付参照をチェックすることができます。このスキャンでは、システム日付関数の参照も報告します。スキャンの実行後、このプロセスで作成されたレポートを検討できます。次の J.D. Edwards ソフトウェア日付参照スキャンを使用することができます。

- ビジネス関数日付参照スキャン
- イベント・ルール日付参照スキャン

▶ 〈Business Function Date Reference Scan(ビジネス関数日付参照スキャン)〉レポート(R9000085)を実行するには

1. 〈Object Management Workbench〉で、次のオブジェクトをチェックアウトして、必須スペックをワークステーションに転送します。
   - B9000085 – 〈Business Function Date Reference Scan〉レポート
   - D9000085 – 日付参照スキャンのデータ構造体
   - R9000085 – 〈Business Function Date Reference Scan〉レポート
2. B9000085 を選択して、[Design(設計)]ボタンをクリックします。
3. [Design Tools]タブを選択して[Build Business Function]をクリックします。
4. 実行するバージョンを選んで[Select(選択)]をクリックします。
5. 〈Data Selection and Sequencing(データの選択と順序設定)〉で、使用するオプションを選択して[Submit(投入)]をクリックします。
6. デフォルトの印刷ディレクトリ(通常、B9¥PrintQueue ディレクトリ)で PDF レポートとテキスト・ログ・ファイルを確認します。

〈Business Function Date Reference Scan〉レポート(R9000085)を実行すると、2 つのレポート・バージョンが生成されます。一方のバージョンは、スキャン・プログラムが検出した日付参照の数を示します。他方のバージョンは、日付参照の詳細情報を示します。

イベント・ルール日付参照スキャン

イベント・ルールをスキャンして、日付参照のイベント・ルールを確認することができます。スキャンの実行後、このプロセスで生成されたレポートを検討できます。

イベント・ルールのスキャン・プロセスは、ビジネス関数のスキャン・プロセスと同じです。次のオブジェクトが使用されます。

- B9000085 – 〈Business Function Date Reference Scan〉レポート
- D9000085 – 日付参照スキャンのデータ構造体
- R9000085 – 〈Business Function Date Reference Scan〉レポート

〈Business Function Date Reference Scan〉レポート(R9000085)を実行すると、2 つのレポートが生成されます。一方のレポートは、スキャン・プログラムが検出した日付参照の数を示します。他方のレポートは、日付参照の詳細情報を示します。