Oracle Forms Developer and Oracle Reports Developer

アプリケーション作成ガイド

リリース 6*i*

2000 年 1 月

部品番号 : J00449-01

ORACLE®

Oracle Forms Developer and Oracle Reports Developer アプリケーション作成ガイド , リリース 6*i*

部品番号 : J00449-01

原本名：Oracle Forms Developer and Oracle Reports Developer: Guidelines for Building Applications, Release 6*i*

原本部品番号：A73073-02

原本協力者：Frederick Bethke, Marcie Caccamo, Ken Chu, Frank Rovitto

目次

2 視覚的効果の高いアプリケーションの設計

3 パフォーマンスの提案

6 オープン・アーキテクチャの利用

はじめに

このマニュアルのガイドラインは、Forms Developer および Reports Developer の強力な機能を十分に活用できるよう支援することを目的としています。これらのアプリケーションを使用した経験の有無にかかわらず、このマニュアルに記載した概念と提案によって、既存の Forms Developer または Reports Developer アプリケーションの Web 上での配布、効果的なグラフィカル・ユーザー・インタフェースの設計、単一アプリケーションを構成するさまざまなモジュールの追跡と管理などの作業が容易になります。

このマニュアルの内容は、製品に付属するオンライン・ヘルプと対応しています。作業を成し遂げる**方法**や製品内で使用可能なオプションを理解するためにいつでもオンライン・ヘルプを利用できますが、オプションを選択する**理由**を判断し、各決定の結果を理解する際にはこのマニュアルが役に立ちます。計画を展開するにはこのマニュアルを使用し、計画を実行するうえでの知識を得るためにはオンライン・ヘルプを使用してください。

ガイドラインは、ユーザー、営業担当員、販売コンサルタントおよび Oracle Applications グループの経験を結集したものです。ガイドラインは、独自の社内標準の開発の基礎として、または既存の標準を補強するために使用できます。

対象読者

このマニュアルは、Forms Developer または Reports Developer を使用するすべてのユーザーを対象としています。初心者および熟練したユーザーの両方のニーズを考慮しています。

構成

このマニュアルは、次の章から構成されます。

章	説明
第 1 章「作成したアプリケーションの管理」	Forms Developer および Reports Developer で利用できるツールを使用して、アプリケーションの開発を開始および管理する方法を説明します。項目は次のとおりです。 ■ プロジェクトの設定と管理 ■ 各種のネットワーキング・シナリオのもとでチーム開発を可能にする方法 ■ プロジェクトのソース制御 ■ プラットフォーム間のプロジェクトのエクスポート アプリケーション・ライフ・サイクル時の異なる環境へのプロジェクトのエクスポート
第 2 章「視覚的効果の高いアプリケーションの設計」	Form Builder、Report Builder および Graphics Builder を使用して、Forms Developer および Reports Developer アプリケーションを開発するための視覚的な面での考慮事項を提示します。
第 3 章「パフォーマンスの提案」	作成したアプリケーションのパフォーマンスを向上させるためのガイドラインを示します。
第 4 章「多言語アプリケーションの設計」	多言語アプリケーションの設計方法を説明します。
第 5 章「移植性のあるアプリケーションの設計」	Windows 95、Macintosh および UNIX に簡単に移植できるアプリケーションの開発方法を説明します。キャラクタ・モード端末用の開発についても説明します。
第 6 章「オープン・アーキテクチャの利用」	次の場合における、Forms Developer および Reports Developer の使用方法を説明します。 ■ OLE オブジェクトおよび ActiveX コントロールを組み込んだアプリケーションの作成 ■ 外部ファンクションによるアプリケーションのカスタマイズ ■ オープン API を使用したアプリケーションの作成と変更 ODBC 準拠データソースに対するアプリケーションの実行

表記規則

このマニュアルでは、次の規則を使用しています。

規則	意味
太字	強調を示します。ボタン名、ラベル、およびその他のユーザー・インタフェースも示します。
斜体テキスト	新しい項目を示します。
クーリエフォント	パス名およびファイル名を示します。
クーリエ大文字	次のものを示します。 ■ ファイル拡張子 (.PLL または .FMX) ■ 環境変数 ■ SQL コマンド ■ ビルトイン / パッケージ名 ■ 実行プログラム名

1

作成したアプリケーションの管理

アプリケーション開発の最も重要な局面の 1 つは、アプリケーションを構成するモジュールの管理です。大規模なアプリケーションは、文字どおり何千ものモジュール、何万行ものコードで構成されることがあります。さらに、プロジェクト全体に重要なコンパイルされてアプリケーションとしては組み込まれないモジュール（設計の仕様、テスト・スクリプトおよびドキュメントなど）についても、その進捗状況を追跡し、管理する必要があります。

この章では、Forms Developer および Reports Developer を使用してアプリケーション開発プロセスの管理を支援する方法を説明します。

項	説明
1.1 項「ソフトウェア開発のライフ・サイクル : 概要」	アプリケーション開発の主要なプロセスを簡単に解説し、そのプロセスごとに Project Builder について説明します。
1.2 項「設計および開発におけるプロジェクト・ドキュメントの管理」	アプリケーションの開発段階で、ドキュメントを管理する方法を説明します。
1.3 項「テスト・フェーズでのプロジェクト・ドキュメントの管理」	テスト段階で、プロジェクト・ドキュメントの適切な構成を QA グループが確実にテストできる方法を説明します。
1.4 項「リリース・フェーズでのプロジェクト・ドキュメントの管理」	作成したアプリケーションのインストール可能なバージョンを顧客に確実に配布する方法を説明します。
1.5 項「完成したアプリケーションの配布」	作成したアプリケーションを Oracle Installer によってインストール可能なアプリケーションに変換する方法を説明します。

1.1 ソフトウェア開発のライフ・サイクル : 概要

アプリケーション開発には、通常、次の 4 つのフェーズがあります。

- **設計 :** アプリケーションの初期の仕様を作成します。この仕様は、顧客からのフィードバック、プロジェクト管理または開発チーム・メンバーからの情報、改善に関する要求、必須のバグ修正、システム分析など、さまざまな情報資源に基づいています。
- **開発 :** 個々のモジュールを作成または変更し、場合によっては、幅広い種類の言語、ツールまたはプラットフォームを取り込みます。
- **テスト :** モジュールをテストします。通常、これには、単体テストとシステム・テストの 2 段階があります。単体テストでは、モジュールまたは機能レベルでテストを行います。たとえば、メニューやボタンなどのユーザー・インタフェース要素のテストです。システム・テストでは、コードの主要部分の統合についてテストします。たとえば、ユーザー・インタフェースのバックエンドです。
- **配布 :** モジュールはインストール可能な形式にパッケージされ、顧客に配布されます。

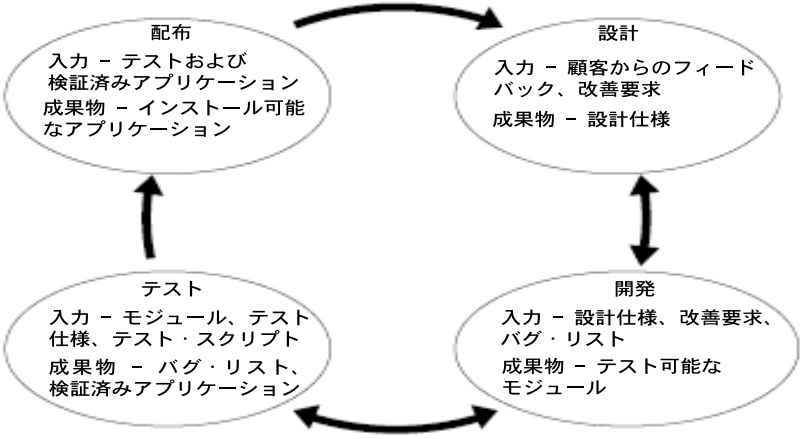


図 1-1 開発ライフ・サイクルのフェーズ : 入力および成果物

アプリケーションのサイズが増え、複雑になるにつれ、この 4 フェーズが繰り返し行われ、生成される情報（実際のコード、バグ・レポート、改善要求など）の量が増えます。それでもなお、最終成果物（顧客がインストールするアプリケーション）の、完全性を保証できるように、すべてのフェーズのすべての入力や成果物を追跡および更新できることが必要です。

この章では、作成したアプリケーションのコード・ベースを Forms Developer または Reports Developer を使用して管理し、バージョンの整合を維持する方法を説明します。

1.1.1 Project Builder による管理計画の遂行

開発プロジェクトは、管理作業を次の 2 つのカテゴリに分割できます。

- プロジェクト管理 : これには、必要な設備、予算およびアプリケーション開発の完了までの各個人の作業量の割当てが含まれます。
- ソフトウェア構成管理 : これには、開発者へのモジュールの割当て、モジュール間の依存関係の決定、開発中のコードの更新およびバージョンの管理が含まれます。

Forms Developer および Reports Developer のコンポーネントである Project Builder を使用すると、ソフトウェア構成の管理作業が容易になり、ユーザーおよびユーザーのチームは、アプリケーションの設計、コーディング、テスティングなどの本来の目的に集中することができます。

1.1.2 Project Builder について

ソフトウェア管理作業を簡素化するため、Project Builder には次の方法があります。

- モジュールをアプリケーションまたはアプリケーションのコンポーネントと関連付けます。
- ファイル形式に基づいてアクションを自動化します。
- モジュール間の依存関係を作成して、カスケードの変更方法を示します。つまり、他のモジュールが変更されたことから再コンパイルの必要があるモジュールを示します。
- デフォルトの接続文字列をモジュールに割り当てます。
- 最終的なインストール・セットに含めるモジュールを指定します。
- チーム・メンバー間でプロジェクトおよびサブプロジェクトを共有し、それらを異なる環境に移植します。
- Project Builder のユーザー・インタフェースから、その他のツールを起動します。

これらの機能の詳細は、1.1.3 項「Project Builder の利点の理解」で説明します。ただし、Project Builder の用語によくわからないものがある場合は、次に進む前に 1.1.2.1 項「Project Builder の用語について」を読んでください。この項では、Project Builder の機能説明に関するコンテキストを記述した基本用語の一部が定義されています。

1.1.2.1 Project Builder の用語について

Project Builder は、次に示す、プロジェクトおよびサブプロジェクトの概念に基づいています。

- プロジェクトとは、作成したアプリケーションの一部となるモジュールおよびファイルへのポインタの集合です。
- サブプロジェクトとは、他のプロジェクトに含まれるプロジェクトで、より詳細なレベルの編成単位が提供されます。サブプロジェクト内のファイル編成には、たいていサブ

ディレクトリ内のファイル編成がミラーリングしていますが、これは必須ではありません。

Project Builder を十分に理解するには、プロジェクトやサブプロジェクトの他に、次の用語も重要です。

- **タイプ :** タイプとは、すべての項目の基礎であり、これによって、Project Builder で使用できるアクションが制御されます。Project Builder のタイプは、主にデフォルトの拡張子から、関連付けられたファイル形式を認識します。たとえば、.TXT はテキスト・ファイルです。Project Builder では、Form Builder モジュール、テキスト・ファイル、C ソース・ファイルなどの、一般的に使用される多数のファイルのタイプが事前定義されています。また、タイプ・ウィザードを使用して他のアプリケーションのタイプを定義することもできます。
- **プロジェクト項目 :** プロジェクトを構成するコンポーネントを項目といいます。項目は、単に、プロジェクトの一部となるファイルの説明です。各項目は、項目のタイプの種類、ファイル・システムにおける位置、サイズ、最終変更日時などが、関連付けられたプロパティ・パレットに詳しく表示されます。項目のアクションおよびマクロ（次を参照）も定義されています。

  項目はファイルそのものではなく、ファイルの説明であることを覚えておいてください。プロジェクトから項目を削除すると、そのファイルがプロジェクトの一部ではなくなったことのみが Project Builder に通知されます。ファイル自体は削除されていません。
- **アクション :** アクションとは、所定のタイプのファイルに適用されるコマンド文字列です。たとえば、テキスト項目の編集アクションは、メモ帳またはワードパッドを起動するコマンド文字列です。
- **マクロ :** マクロとは、アクションの変更に使用できる変数です。マクロは、定数または簡単な式（また、これにもその他の定数または式、あるいはその両方が含まれることがある）の場合があります。たとえば、データベースへの接続で指定したすべての情報は、Project Builder によって、接続を要求する可能性があるすべてのコマンドに含まれる ORACONNECT マクロに挿入されます。その後、自動的にログオンできるようにマクロ内の情報がアクションに挿入されます。

  スクリプト自体を編集せずに、スクリプトまたはバッチ・ファイル内の環境変数を使用して、スクリプトのアクションを簡単に変更することがあるように、アクション自体を変更しなくてもマクロを使用してアクションをカスタマイズできます。たとえば、作成したアプリケーションのコンパイルをデバッグ・モードで行うか、または最適化モードで行うかを決めるためにマクロを定義する場合があります。アプリケーションの配布用バージョンの作成段階では、デバッグ・フラグを使った生成（Build）コマンドのタイプをすべて検索して変更するのではなく、単に 1 つのマクロの定義を変更してデバッグをオフにするのみで済みます。
- **グローバル・レジストリ :** グローバル・レジストリには、事前定義済みの Project Builder のタイプが含まれています。

- **ユーザー・レジストリ :** 各ユーザーには、新規タイプの定義、既存タイプの再定義、およびアクションまたはマクロの変更や作成を行うユーザー・レジストリがあります。
- **プロジェクト・レジストリ・ファイル :** プロジェクト・レジストリ・ファイルには、プロジェクト内に含まれたモジュールへのポインタ、デフォルトの接続文字列およびプロジェクトのホーム・ディレクトリへのポインタなど、プロジェクトの追跡に必要となる情報が含まれています。

Project Builder インタフェースには、プロジェクトを構成する項目の操作に使用する、次の 3 つのツールがあります。

- プロジェクト・ナビゲータには、作成したアプリケーション内のモジュールを表示できる、使い慣れた " ナビゲータ " または " エクスプローラ " スタイルのインタフェースがあります。さらに、Project Builder のフィルタ機能を使用して、参照するモジュールのみを表示することができます。プロジェクト・ナビゲータから直接、編集ツールを起動することもできます。
- プロパティ・パレットでは、選択済み項目のプロパティを調査および変更できます。
- ランチャとは、補助ツールバーで、開発ツールへアクセスするための別の手段です。ランチャへのボタンの追加や、希望するサードパーティ製のツールとの関連付けが可能です。

図 1-2 には、これら 3 つのツールがすべて示されています。

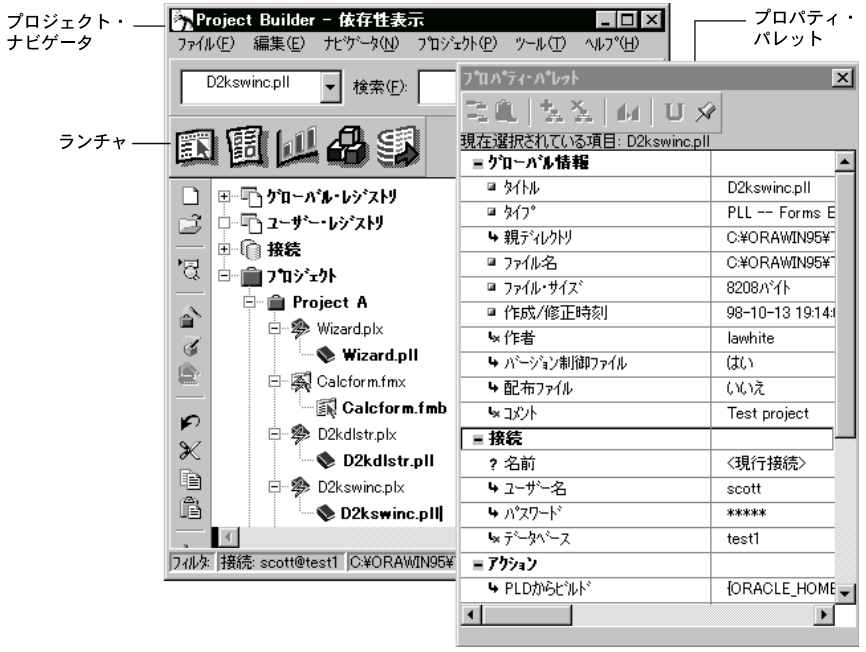


図 1-2 Project Builder ユーザー・インタフェース

1.1.2.2 Project Builder が既存の開発の役割分担に与える影響

アプリケーション開発の作業が円滑に進むように役割が必要です。プロジェクト・マネージャ、開発マネージャおよびチーム・リーダーなどは開発グループで共通の役割分担であり、定義は不要です。ただし Project Builder には、プロジェクト管理者という新しい役割分担があります。

プロジェクト管理者はプロジェクトを作成しプロジェクトを開発者が利用できるようにします。プロジェクト管理者はグローバル・レジストリをメンテナンスし、必要に応じてそれを変更しチームの開発者にその変更をエクスポートします。また、テスト環境などの異なる環境、または異種プラットフォーム間の開発の場合はその他のプラットフォームにプロジェクト情報をエクスポートすることもあります。

プロジェクトの管理時にプロジェクト管理者が行う作業は、次のチーム・メンバーの役割分担に影響することがあります。

- 開発者

- ソース制御管理者
- テスト担当者（QA）
- リリース担当者

もちろん、各チーム・メンバーの正確な役割分担は開発グループによって異なります。また、チームの 1 メンバーが複数の役割分担を果たす場合があります。たとえば、チーム・リーダーがプロジェクト管理者、または開発者がソース制御の担当者ということもあります。

1.1.3 Project Builder の利点の理解

Project Builder の基本用語を理解したら、Project Builder がもたらす利点について調べます（不明な用語がある場合は、1.1.2.1 項「Project Builder の用語について」を参照してください）。

1.1.3.1 アプリケーションへのモジュールの関連付け

モジュールを同じプロジェクトに追加するだけで、アプリケーション内のすべてのモジュールをそのアプリケーションに関連付けることができます。これによって、大規模なアプリケーションを 1 つのエンティティとして追跡したり、モジュール間の依存関係を判別することなどができます。

1.1.3.2 ファイル形式に基づいたアクションの自動化

Project Builder には、デフォルト・アクション（オープン、編集または印刷など）がすでに割り当てられた、拡張子のタイプ・リストがあります。モジュールを選択して、マウスの右ボタンをクリックすると、そのタイプに関連付けられたアクションがポップアップ・メニューに表示されます。デフォルトでは、タイプ定義に含まれているアクションは、プロジェクト内のそのタイプのモジュールすべてに適用されます。また、これらのアクションを修正したり、これらに追加することもできます。

アクションは、単なるコマンド文字列です。実際のコマンド文字列を使用してアクションを定義する利点（もちろん、簡単なだけでなく）の 1 つは、概念上アクションが数種類のタイプと関連付けられることです。たとえば、Word 文書の編集とテキスト文書の編集は、概念上非常に似ていますが、異なるツールが必要です。Project Builder では、2 種類のタイプの同じ編集（Edit) コマンドを関連付けられますが、各タイプのコマンド文字列が異なるため、混乱することはありません。このように、処理しているモジュールのタイプにかかわらず、1 つのコマンドによって該当するアクションが実行されます。

1.1.3.3 モジュール間の依存関係の作成

どのモジュールが他のどのモジュールに依存するかを知ることは、変更があった場合に、再コンパイルしなければならないモジュールを決定するのに必要です。また、変更の影響を管理するのにも重要です。たとえば、あるライブラリを変更するとどのフォームが古くなるか、などです。

Project Builder では、タイプ定義にモジュール・タイプの依存関係が含まれています。このようにして、プロジェクト内の既存のモジュール間の依存関係が認識されます。モジュールへの変更も追跡できるので、変更されたモジュール**および**それに依存したモジュールが自動的に再コンパイルされます。

実際には、Project Builder によりプロジェクト内にまだ存在しない依存関係が認識され、それらのマーカーが作成されます。これらのマーカーは、暗黙的項目と呼ばれています。たとえばプロジェクトに、Project Builder の「Form Builder ドキュメント」で定義した .FMB ファイルが含まれているとします。「Form Builder 実行モジュール」(.FMX ファイル）は存在しない、つまり、まだ生成されていない可能性があります。しかし、Project Builder では .FMB ファイルの存在によってこの .FMX ファイルの存在を暗黙的に認識できるため、それをマークするための暗黙的項目が作成されます。

暗黙的項目の存在を判別するために、Project Builder では、各定義済みタイプの**「配布可能タイプ」**プロパティの値と、**「< タイプ > からビルド」**アクションに必要な入力項目、つまりソースとが相互に関連付けられます。前述の例では、「Form Builder モジュール」タイプの**「配布可能タイプ」**プロパティは「Form Builder 実行モジュール」、つまり .FMX として定義されています。Form Builder 実行モジュール用に定義された**「< タイプ > からビルド」**アクションは、「FMB からビルド」になります。これは、.FMB ファイルが .FMX ファイル作成用の入力項目で、逆に .FMX ファイルは .FMB ソースのターゲットであることを意味しています。

暗黙的項目の連鎖は複数のファイルから構成されます。たとえば、ライブラリ・ファイルに C ソース・ファイルを追加するとします。この場合、Project Builder によってあるファイル形式から別のファイル形式（オブジェクト・ファイルなど）への**「< タイプ > からビルド」**アクションの完全パスを取得するために、その他のすべてのタイプのモジュールが追加されます。

Project Builder によって、コンパイル可能なモジュールとその結果作成された実行プログラムの間にのみ存在する依存関係が検出されますが、モジュールに依存する項目の下にあるプロジェクトへモジュールを追加すると、手動で依存関係を設定できます。たとえば、.FMB ファイルが PL/SQL ライブラリに依存している場合、.FMB の下にあるプロジェクトに .PLL を追加することで、その依存関係が Project Builder で認識されます。

1.1.3.4 モジュールへのデフォルトの接続文字列の割当て

Project Builder を使用すると、最も頻繁に使用される接続文字列をすべて定義し、自分のユーザー・レジストリ内の接続ノードの下にそれらの定義を格納できます。さらに、接続ノードから接続をドラッグしてモジュールにドロップすると、接続をモジュールに割り当てられます。そのモジュールを編集する必要がある場合、たとえばフォームであれば、プロジェクト・ナビゲータからそのフォームを選択し、ポップアップ・メニューから**「編集」**を選択できます。Project Builder では、Form Builder のオープンとデータベースへの接続が自動的に行われます。

1.1.3.5 最終的なインストール・セットに含めるモジュールの指定

Project Builder では、最終的なインストールパッケージに含めるモジュール（たとえば、.EXE ファイル、.DLL ファイル、.HLP ファイルなど）を容易に判断および追跡できます。配布用のファイルにマークを付けるには、**「配布ファイル」**プロパティを「はい」に設定します。インストール・パッケージの作成準備が整うと、配布ウィザード・アクションによって、**「配布ファイル」**プロパティを「はい」に設定したすべてのモジュールが 1 つの単位にパッケージされます。

注意 : タイプまたは個別のプロジェクト項目について、**「配布ファイル」**プロパティを設定できます。

1.1.3.6 プロジェクトおよびサブプロジェクト・レジストリ・ファイルの共有および移植

Project Builder によって、プロジェクトに関する情報を他のチーム・メンバーまたは他のプラットフォームにエクスポートできます。タイプ、アクション、マクロ、およびプロジェクト・レジストリ・ファイルについての情報は、カスタマイズしたすべての情報を含め、他の環境およびプラットフォームにインポートできるテキスト・ベースのエクスポート・ファイルに書き込まれます。これによって、異種プラットフォーム間の開発とテストが可能になります。

1.1.3.7 その他の製品コンポーネントおよびサード・パーティ製ツールへのアクセス

次の数種類の方法で、Project Builder ユーザー・インタフェースからその他のツールにアクセスできます。

- アクション : プロジェクト・ナビゲータ内のモジュールを選択し、マウスの右ボタンをクリックしてアクセスします。選択した項目に関連付けられたすべてのアクションをリストするポップアップ・メニューが表示されます。リストされたアクションによってコマンド文字列で指定したすべてのツールが起動されます。また、プロジェクト・ナビゲータ内のエントリをダブルクリックして、デフォルト・アクションを起動することもできます。
- **「生成」**、**「配布」**およびソース制御アクションは、関連付けられたツールを起動します。
- ランチャ・ツールバー : Form Builder、Report Builder、Graphics Builder など、多くのコンポーネントを起動します。また、ランチャ・ツールバーに作成したボタンを追加したり、希望するサードパーティ製のツールを関連付けることもできます。

1.1.3.8 ソース制御パッケージの使用方法

Forms Developer および Reports Developer では、次のソース制御パッケージへのインタフェースを提供しています。

- Intersolv 社の PVCS

- PureAtria 社の Clearcase
- StarBase 社の Versions(StarTeam のソース制御コンポーネント)

また、他のソース制御ツールを指し示すように、Project Builder で提供されているこのソース制御アクションを変更すれば、他のソース制御ツールも使用できます。

さまざまなソース制御パッケージが使用可能であり、Forms Developer および Reports Developer と併用できるため、プロジェクトのソース制御を行う様々な方法について、この章ですべて扱うことはできません。ただし、総合的なガイドラインは、該当箇所に記述されています。

1.2 設計および開発におけるプロジェクト・ドキュメントの管理

よいアプリケーションを作成するには、設計が重要であると繰り返し述べられてきました。設計フェーズにおける成果物には、設計書、仕様書、会議の議事録、ユーザー・インタフェース・プロトタイプ、顧客の調査結果（新規アプリケーションの場合）、ユーザー・テスト、および改善に関する要求のリスト（アプリケーションの改訂の場合）など、プロジェクトに追加され、追跡されるすべての文書が含まれます。

これは、開発作業に向いているプロジェクト管理者を設計プロセスの初期に決め、ただちにプロジェクトの作成を開始する必要があることを示します (プロジェクト管理者の役割分担に関する詳細は、1.1.2.2 項「Project Builder が既存の開発の役割分担に与える影響」を参照してください)。この項では、設計及び開発フェーズでプロジェクトを管理するために Project Builder を設定する場合のプロジェクト管理者および開発チームのメンバーの役割分担を説明します。特に、この項では、次のことを説明します。

- Project Builder のインストール
- プロジェクトの作成
- プロジェクトおよびプロジェクト・ドキュメントの処理
- 複数のプラットフォーム間でのプロジェクトおよびプロジェクト・ドキュメントの管理

注意 : Project Builder による簡単な作業の実行に関する手順は、Project Builder のオンライン・ヘルプに書かれているため、この章では説明しません。

1.2.1 Project Builder のインストール

Project Builder は、自動的に `ORACLE_HOME¥PJ10` にインストールされます。このディレクトリにある重要なファイルは、次のとおりです。

- グローバル・レジストリ・ファイル（`TYPESnn.UPD`）。ここで、nn は各国語を示します。
- デフォルトのユーザー・レジストリ・ファイル（`PJUSERnn.UPD`）。ここで、nn は各国語を示します。

Project Builder のインストール中に処理する最も重要な問題は、どのようにしてチーム・メンバーもこれらのさまざまなファイルを使用できるようにするかということです。1.2.1.1 項「プロジェクトおよびユーザー・レジストリのインストール」に、オプションを説明します。

1.2.1.1 プロジェクトおよびユーザー・レジストリのインストール

Project Builder は、セキュリティ用のプロトコルを共有するネイティブ・ファイルに依存します。したがって、プロジェクト・ファイルは不慮の変更に対処できないことがあります。これは、グローバル・レジストリおよびユーザー・レジストリの構成方法を決める際に考慮に入れる必要があります。表 1-1 に、使用可能なオプションをリストします。

表 1-1 レジストリのインストール・オプション

オプション	処理	状況	推奨事項
Project Builder とともに、グローバル・レジストリを共有ネットワーク・ドライブに、ユーザー・レジストリをローカル・マシンにインストール。	チームでネットワークを使用している場合、開発者はグローバル・レジストリのコピーの 1 つにアクセスできます。これによって、使用中のグローバル・レジストリは、すべて最新のものに更新されます。	すべてのチーム・メンバーがグローバル・レジストリに対する書込みアクセス権を持っている場合、誤って上書きされる可能性があります。	グローバル・レジストリに誤って上書きされるのを防ぐには、自分のみが書込みアクセス権を持つディレクトリにインストールします。
Project Builder とともに、自分のユーザー・レジストリのみでなく、各チーム・メンバーが使用できるグローバル・レジストリのコピーをインストール。	変更されたファイルのコピーをチーム・メンバーが使用できるようにするだけで、グローバル・レジストリの更新を波及できます（同じプラットフォームの場合）。	個人のグローバル・レジストリは、誤って書込みまたは削除が行われる危険があります。	Project Builder のエクスポート機能を使用して、提供しているコピーのかわりに変更されたレジストリ・ファイルを波及します。より厳しい処置をとれば、レジストリ・ファイルに対する不用意な態度を防止できることがあります。
Project Builder とともに、グローバル・レジストリおよびチーム・メンバー共用の 1 つのユーザー・レジストリをインストール。		タイプ、アクション、プロジェクトおよびプロジェクト・モジュールは、変更の矛盾が発生する危険があります。	このオプションは選択しないでください。しかし、これを選択する必要がある場合、開発チームのメンバーが編集できるのは、プロジェクトではなくモジュールのみにします。

1.2.2 プロジェクトの作成

次の項では、開発者チームへ配布する 1 つのプロジェクトの作成について説明します。ただし、これはグループにとって最適なオプションではない可能性があります。開発中のアプリケーションが非常に大きい場合、またはコンポーネントがチーム・メンバー間で分割されている場合は、複数に分割したより小さなプロジェクト、つまり各開発者あるいは開発グループが責任を負う各プロジェクトの内容を作成するように選択しても構いません。

1 つのプロジェクトを分配するように決めた場合、Project Builder のプロジェクトでは、指定位置に存在しないモジュールへのポインタが受け入れられることに注意してください（プロパティ・パレット内の情報を調べると、モジュールの存在を判別できます。存在しない場合は、**「作成 / 修正時刻」**および**「ファイル・サイズ（バイト）」**がブランクになっています）。これは、チーム・メンバー全員ですべてのモジュールを使用可能にしなくても、1 つの大きなプロジェクトを分配できることを意味しています。

プロジェクトの作成は、各開発チーム・メンバーと同様にプロジェクト管理者も参加する必要がある、継続した作業です。この項では、各役割分担での責任の特徴を説明します。

1.2.2.1 プロジェクトの作成 : プロジェクト管理者

プロジェクト管理者としてプロジェクト・レジストリ・ファイルを作成し、プロジェクトに何を含めるかを決定します。Project Builder に付属のプロジェクト・ウィザードを使用してプロジェクトを作成する場合、プロジェクト・レジストリ・ファイルを作成してさまざまなプロパティを手動で編集する場合でも、必ず次の作業を完了する前に事前計画を立ててください。

1. プロジェクトを作成するには、次のようにします。
   a. プロジェクトのディレクトリ構造を設定します。
   b. モジュールを追加します。
   c. デフォルト・アクション、マクロおよび接続文字列を設定します。
   d. 必要な依存関係を手動で設定します。
2. 共同のソース制御プロジェクトを設定するため、ソース制御管理者とともに次の処理を行います。
   a. 新規のタイプを定義し、既存のタイプを編集します。
   b. アクションおよびマクロをカスタマイズします。
   c. 再使用可能な接続を作成します。
3. チーム・メンバーがプロジェクトを使用できるようにします。

次のトピックでは、これらの各作業を完了するための推奨事項を説明します。

1.2.2.1.1 ステップ 1: プロジェクトの作成

プロジェクト・ウィザードには、プロジェクトの作成用に使いやすいインタフェースがあります。また、プロジェクト・ウィザードを使用せずに（ツールバーの「新規プロジェクト」ツールを使用して）新規プロジェクトを作成し、プロパティ・パレットでプロジェクトのプロパティを設定することもできます。

最も簡単な方法は、新規プロジェクトがグローバル・レジストリの情報をそのまま継承するプロジェクト・レジストリ・ファイルを持つデフォルトの設定の場合です。その他の場合はまずありません。Project Builder では次のトピックで説明しているように、作成したプロジェクトを追跡し記録する前により詳しい情報が必要です。

ステップ 1a: プロジェクトのディレクトリ構造の設定

プロジェクトのディレクトリ構造は、非常に重要です。たとえばプロジェクト・ディレクトリの子ではない、あるディレクトリに置かれたモジュールをプロジェクトが持つとします。そこで、検索のアクションを作成し、プロジェクト・モジュールを変更するとします。どのようにして“親なし”のモジュールを検索すればよいでしょうか。ハードコードされたパスを用いて代理のアクションを作成しますか。それでは移植性がありません。ルートから検索しますか。それも効率的ではありません。

推奨事項 :

- プロジェクト項目のデフォルト・ディレクトリ、またはその子にあたるディレクトリにモジュールを置きます（サブプロジェクトの追加時にはよい方法です）。
- できる限り、実際のディレクトリ構造をミラーリングするように、プロジェクトおよびサブプロジェクトを編成します。

モジュールをプロジェクトに追加する標準的な方法は、「プロジェクトにファイルを追加」ダイアログです。追加するモジュールがプロジェクト項目のデフォルト・ディレクトリにない場合、ダイアログには必ずフルパスが挿入されることに注意してください。その場合は、相対パス名が使用されます。

ステップ 1b: モジュールの追加

ディレクトリ構造の計画を立てたら、プロジェクトにモジュールを追加できます。

推奨事項 : 可能であれば、プロジェクトの編成に役立つサブプロジェクトを常に使用してください。ただし、すべてのフォームまたはすべてのレポートを単純にグループ化しないでください。モジュールをコンポーネントごとにグループ化します。たとえば、.FMB ファイル、.FMX ファイル、PL/SQL ライブラリ、メニュー、ビットマップ、アイコンなどを含む大きなフォーム内のすべてのモジュールのサブプロジェクトを作成する場合があります。この結果、Project Builder では検出されない一部の必要な依存関係をより簡単に作成できます。

ステップ 1c: デフォルト・アクション、マクロおよび接続文字列の設定

このステップでは、アクションおよびマクロをサイト別に編集する必要があります。たとえば、サイトで標準的なコンパイラおよびコンパイラ・オプションを使用するビルド・アクションの変更などです。まだ行っていない場合は、テスト・データまたは必要な表が含まれた、最も頻繁に使用されるデータベースの接続文字列を作成することもできます。

ステップ 1d: 手動による必要な依存関係の設定

Project Builder では、モジュール間の一部の依存関係（.FMX ファイルが、FMT ファイルによってビルドされる .FMB ファイルからビルドされること) が認識されていますが、**「配布可能タイプ」**および**「< タイプ > からビルド」**アクションのクロス・リファレンスによってわかる依存関係のみです。

その他にも、PL/SQL ライブラリ、メニュー、アイコンなどの依存関係が存在する可能性があります。図 1-3「手動で追加された依存関係」に示すとおり、依存モジュールのエントリのモジュールに依存したモジュールのエントリを作成すれば、Project Builder にこれらの依存関係を通知できます。

図 1-3 手動で追加された依存関係

この図は、WIZARD.PLL、NAVIGATE.PLL、および NAVWIZ.MMB への NAVWIZ.FMB の依存の関係を示しています。

1.2.2.1.2 ステップ 2: ソース制御管理者との作業

プロジェクトを作成すると、ソース制御パッケージを導入できます。通常、サード・パーティ製のソース制御パッケージでも、プロジェクトの概念をインプリメントできます。

推奨事項: ソース制御管理者とともに、Project Builder の開発プロジェクトをミラーリングするソース制御プロジェクトを設定してください。

複雑なアプリケーションをソース制御するためにプロジェクトを設定する場合は、隠しモジュールも必ず含めるようにしてください。たとえば、フォームをチェックインする場合、使用するメニュー、PL/SQL ライブラリ、ユーザー・イグジット、アイコンまたは特殊フォントを必ず組み込んでください。Windows で実行されるアプリケーションでは、ソース制御される OCX または ActiveX コントロールが使用されていることもあります。

1.2.2.1.3 ステップ 3: チーム・メンバーがプロジェクトの使用可能化

プロジェクトの作成およびソース制御の設定を行う準備作業が終了したら、すべてのプロジェクト情報をプロジェクト・エクスポート・ファイルにエクスポートし、チーム・メンバーに場所を通知します。これで、メンバーはプロジェクトをインポートできます。

チーム・メンバーに実際のプロジェクト・レジストリ・ファイルの場所を通知することもできますが、Project Builder では、使用しているオペレーティング・システムのセキュリティ機能によってプロジェクト・モジュールの削除または上書きが禁止されていることを覚えておいてください。単なる削除および上書きは可能です。プロジェクトの統合性を維持するには、Project Builder のプロジェクト更新プロセスに従って、変更されたレジストリ・ファイルを分配するのではなく、プロジェクトへの変更を常にインポートおよびエクスポートしてください。

チーム・メンバーにプロジェクト・エクスポート・ファイルの場所を通知する場合は、設定したディレクトリ構造も通知して開発用のマシンでその構造がミラーリングされるようにする必要があります。異なるプロジェクト位置へのマッピングには、`Q:¥myproj` や `R:¥` などのプロジェクト位置用に異なる値を指定したプロジェクト・レジストリ・ファイルのコピーがそれぞれ必要なので、プロジェクトの最も簡単な設定方法は、チーム・メンバー全員がそれぞれのマシン上の同じプロジェクト項目のデフォルト・ディレクトリにプロジェクト位置をマッピングすることです。

チーム・メンバーは、割り当てられたモジュールをチェックアウトできます。

1.2.2.2 プロジェクトの作成チーム・メンバー

プロジェクト管理者が、1.2.2.1 項「プロジェクトの作成 : プロジェクト管理者」に記述された作業を完了すると、プロジェクトのチーム・メンバーは作業をうまく調整できるようになります。プロジェクトのチーム・メンバーは、次の作業を行うことになります。

1. ディレクトリ構造の設定およびプロジェクトのインポート
2. ユーザー・レジストリのカスタマイズ
   a. 新規のタイプの定義、および既存のタイプの編集
   b. アクションおよびマクロのカスタマイズ
   c. 再使用可能な接続の作成
3. 割り当てられたモジュールのチェックアウト

1.2.2.2.1 ステップ 1: ディレクトリ構造の設定およびプロジェクトのインポート

プロジェクトが使用可能であることを管理者から通知されたら、プロジェクト情報をインポートして、割り当てられたモジュールで作業ディレクトリを設定できます。

推奨事項 : プロジェクト管理者がプロジェクト用にすでに作成したものをミラーリングするようにディレクトリ構造を設定する場合、ファイル管理は簡単になります。

1.2.2.2.2 ステップ 2: ユーザー・レジストリのカスタマイズ

プロジェクトの設定で最初に行う必要があることの 1 つは、ユーザー・レジストリをカスタマイズすることです。

ステップ 2a: 新規のタイプの定義、および既存のタイプの編集

グローバル・レジストリに表示されないタイプのプロジェクトにモジュールを追加する場合は、ユーザー・レジストリに新しいタイプを定義して、それにアクションやマクロなどを割り当てることができます。

加えて、グローバル・レジストリの特定のタイプのデフォルトのコマンドまたはマクロを書き替える必要があるかもしれません。これは、グローバル・レジストリ内のタイプをコピーして、ユーザー・レジストリにペーストして編集すると簡単です。これで、プロジェクト内のそのタイプのすべてのモジュールには、ユーザー・レジストリ内のタイプの変更が継承されます。

推奨事項: ユーザー・レジストリにコピーしてから編集する方法でグローバル・タイプを変更する場合は、プロジェクト管理者に通知してください。このような変更は、チーム全体に役立つ可能性があります。

ステップ 2b: アクションおよびマクロのカスタマイズ

ユーザー・レジストリに追加するタイプに関連付けられたアクションおよびマクロもカスタマイズできますが、Project Builder の階層内のその他の位置でもアクションとマクロを変更できることを必ず覚えておいてください。項目を編集する位置によって、変更の影響範囲が変わります。

次の表に、アクションまたはマクロが存在する可能性がある位置、そのアクションまたはマクロの有効範囲、およびそれを上書きできるものをすべてリストします。

アクションまたはマクロの割当て先	影響範囲	上書きできるもの
グローバル・レジストリ（Global Registry）	グローバル・レジストリ下のすべてのユーザー・レジストリおよびプロジェクトで、マクロまたはアクションが割り当てられたタイプの全項目	ユーザー・レジストリまたはプロジェクト、サブプロジェクト、エントリ内のアクションおよびマクロ
ユーザー・レジストリ	ユーザー・レジストリ下のすべてのプロジェクトで、マクロまたはアクションが割り当てられたタイプの全項目	プロジェクトまたはサブプロジェクト、エントリ内のアクションまたはマクロ
プロジェクト	プロジェクト内でマクロまたはアクションが割り当てられるタイプの全項目	サブプロジェクトまたは項目内のアクションまたはマクロ
サブプロジェクト	サブプロジェクト内でマクロまたはアクションが割り当てられるタイプの全項目	項目内のアクションまたはマクロ

アクションまたはマクロの割当て先	影響範囲	上書きできるもの
エントリ	エントリのみ	上書き不可

ステップ 2c: 再使用可能な接続の作成

テスト用に作成したデータを含む一連の表がある場合は、プロジェクト管理者から与えられたリストに作成した接続を追加できます。接続を作成したらプロジェクト・ナビゲータ内の接続のエントリを選択して、プロジェクト・ファイル・エントリにドラッグし、選択したモジュールのエントリにドロップすれば、モジュールに接続を割り当てることができます。これで、データベース接続が必要なツールをオープンするアクションを選択すると Project Builder がログオンします。

1.2.2.2.3 ステップ 3: 割り当てられたモジュールのチェックアウト

ディレクトリ構造の設定を完了し、プロジェクトをインポートすると割り当てたモジュールを作業領域に移入できます。メイン・メニューの**「ファイル」**→**「管理」**からアクセスできるソース制御コマンドの**「チェックイン」**、**「チェックアウト」**、および**「ソース制御オプション」**は、各タイプに定義されたアクションに関連付けられています。これは、必要であれば、コマンドの結果に影響を及ぼすように、アクションを変更できることを意味します。ただし、これはソース制御にはお薦めできません。

1.2.3 プロジェクトおよびプロジェクト・ドキュメントの処理

プロジェクトが開発フェーズに入るとプロジェクトの統合性を維持することがさらに重要になります。

推奨事項: 複数のチーム・メンバーに影響するプロジェクトへの変更（グローバル・レジストリの変更や新規モジュールが含まれたサブプロジェクトの追加など）は、プロジェクト管理者のみが行うようにしてください。

1.2.3.1 プロジェクトの処理 : プロジェクト管理者

アプリケーションの開発段階でプロジェクト管理者の役割はプロジェクトのメンテナンスおよびサポートです。さらに、開発での成果物の管理を担当するか、またはそれを開発マネージャとともに担当する場合もあります。その場合は、次のことを行う必要があります。

- 新規モジュールおよび依存関係の追加
- プロジェクト・レジストリ・ファイルに対する変更のエクスポート
- バージョン・ラベルの適用

1.2.3.1.1 新規モジュールおよび依存関係の追加

先にプロジェクトを作成してから新規モジュールをプロジェクトに追加し、依存関係を手動で追加しなければならない場合もあります。これを行うプロセスは、初めてプロジェクトを作成する場合と同様です。詳細は、1.2.2.1.1 項「ステップ 1: プロジェクトの作成」を参照してください。

1.2.3.1.2 プロジェクト・レジストリ・ファイルへの変更のエクスポート

新規モジュールを追加して必要な変更を行ったら、その変更をエクスポートして、チーム・メンバーがモジュールを使用できるようにします。これを行うプロセスは、初めてプロジェクトをエクスポートする場合と同様です。詳細は、1.2.2.1.1 項「ステップ 1: プロジェクトの作成」を参照してください。

1.2.3.1.3 バージョン・ラベルの適用

互いにさまざまな改訂バージョンを同期化しようとしても（たとえば、毎晩チェックインすることによって）、あるモジュールは完成しようとしているのに、別のモジュールはまだバグの吸収やヘッダーの変更が必要だったりすることはよくあることです。改訂バージョンの同期をとるのは、通常実用的ではありません。

もっとよい方法は、重要なマイルストーンに達したことを改訂バージョンのグループにマークするために、シンボリック・バージョン・ラベルを用いることでバージョンを同期化することです。たいていの主要なソース制御ツールでは、ソース制御プロジェクトにシンボリック・ラベルを付けることができます。

1.2.3.2 プロジェクト・ドキュメントの処理 : チーム・メンバー

プロジェクトが設定され、モジュールが割り当てられると、Project Builder を次のことに使用できます。

- モジュールの編集
- 手動によるモジュールおよび依存関係の追加
- プロジェクトの作成
- モジュールのチェックインおよびチェックアウト

1.2.3.2.1 モジュールの編集

推奨事項 : Project Builder を使用してモジュールを編集する最も効率的な方法は、必要なオプションを使用してツールが起動されるように編集対象のモジュールのタイプに関連付けられたアクションをカスタマイズすることです。さらに、必ず個別のモジュールまたはプロジェクトのいずれかに接続文字列を関連付けるようにしてください。そこで、ユーザー・レジストリ内のその位置から接続をドラッグし、モジュールまたはプロジェクト・エントリにドロップできます。このようにモジュールを準備しておけば、ポップアップ・メニュー項目を選択するか、またはプロジェクト・エントリをダブルクリックすることによって該当する

アプリケーションでモジュールがオープンします。必要なときに、すでにログオンされています。

また、ランチャを使用して開発ツールにアクセスすることもできます。ランチャは、Forms Developer および Reports Developer ツールのツールバー・ボタンにすでに設定されていますが、ボタンを作成してサード・パーティ製ツールの実行プログラムに割り当てることもできます。

注意: ランチャを介してツールを起動後モジュールをオープンする場合、そのツールは、関連付けられたすべての接続文字列が認識されていません。データベースに手動でログオンする必要があります。

1.2.3.2.2 手動によるモジュールおよび依存関係の追加

1.2.2.1.1 項「ステップ 1: プロジェクトの作成」を参照するか、またはプロジェクト管理者に連絡してください。

1.2.3.2.3 プロジェクトの作成

「ビルド」コマンド ―**「選択項目をビルド」**、**「変更分をビルド」**および**「すべてビルド」**― は、メイン・メニュー（**「プロジェクト」**の下）から使用できます。これらもまた、アクションに関連付けられています。この場合は、**「< タイプ > からビルド」**アクションです。

これは、異なるモジュール・タイプに 1 つのコマンドを選択でき、そのモジュールが**「< タイプ > からビルド」**アクションの定義に応じてコンパイルされることを意味します。つまり、その特定のタイプではなく、実際に作成したいターゲットを選択できるということです。

たとえば、.FMX ファイルのための**「< タイプ > からビルド」**アクションによって、Form Compiler が起動され、対応する .FMB から .FMX ファイルが作成されます。**「ビルド」**コマンドによってコンパイルされるのは .FMB ですが、どのように .FMB がコンパイルされるかを決めるのは作成された .FMX ファイルに関連付けられているアクションです。

対応するターゲットの**「< タイプ > からビルド」**アクションの定義を変更すると、**「ビルド」**コマンドの結果を変更できます。

「選択項目をビルド」を選択して、選択したモジュールをコンパイルするか、または**「すべてビルド」**を選択して強制的にすべてのモジュールをコンパイルします。Project Builder ではモジュールが古くなって再コンパイルが必要なことを検出できるため、古いモジュールが含まれるプロジェクトのエントリを選択後、**「変更分をビルド」**を選択すれば、その古いモジュールのみをコンパイルできます。

注意: **「ビルド」**コマンドは、ポップアップ・メニューからも使用できます。

1.2.3.2.4 モジュールのチェックインおよびチェックアウト

モジュールがソース制御のために変換を必要としている場合（たとえば、ソース制御がテキストにのみ動作し、モジュールがバイナリの場合）、チェックインの前に**「ファイルを RCS にチェックイン」**アクションを編集して、テキストへの変換を自動化できます。

また、**「ファイルを RCS からチェックアウト」**アクションは、バイナリに戻ったモジュールをテキスト・ベースでソース制御されたバージョンに変更するために、同じような方法で編集できます。

1.2.4 複数のプラットフォーム間でのプロジェクトおよびプロジェクト・ドキュメントの管理

今日では、多数のアプリケーションが複数のプラットフォーム上で実行され、さまざまなプラットフォーム上で開発が行われます。第 5 章「移植性のあるアプリケーションの設計」では、プロジェクトの基礎となるアプリケーションに移植性を持たせることに役立ちます。

プロジェクトに移植性を持たせるためにも、Project Builder では数種類の主要なプラットフォーム上での開発もサポートされています。そのためには、プラットフォームを反映したグローバル・レジストリを装備している必要があります。つまり、定義されたタイプがそのプラットフォームに存在し、アクションおよびマクロがプラットフォームの構文ルールに従って書かれている必要があります。これは、グローバル・レジストリと、拡張子別のすべてのユーザー・レジストリおよびプロジェクト・レジストリ・ファイルには移植性がないことを示します。

ただし、1.1.3.6 項「プロジェクトおよびサブプロジェクト・レジストリ・ファイルの共有および移植」にあるように、プロジェクトに関する情報をテキスト・ファイルにエクスポートし、そのファイルを別のプラットフォームへインポートできます。

1.2.4.1 複数のプラットフォーム間でのプロジェクトの管理 : プロジェクト管理者

複数のプラットフォームで開発中のプロジェクトを担当する管理者の場合は、次の作業を行います。

- プラットフォームのコードを含むソース制御プロジェクトを分岐させます。
- 別のプラットフォームにプロジェクトおよびプロジェクト情報をエクスポートします。

1.2.4.1.1 プラットフォームのコードを含むソース制御プロジェクトの分岐

ソース制御管理者とともに、チーム・メンバーが新規プラットフォームのコードを分離できるようにする分岐ソース制御プロジェクトを作成してください。

1.2.4.1.2 別のプラットフォームへのプロジェクトおよびプロジェクト情報のエクスポート

別のプラットフォームにプロジェクトを分配するためにエクスポート・ファイルを作成するのは、同じプラットフォームでチーム・メンバーに分配するエクスポート・ファイルを作成する場合と同じです。Project Builder で作成されるエクスポート・ファイルはテキスト・ファイルなので、別のプラットフォームに簡単に移行できます。

1.2.4.2 複数のプラットフォーム間でのプロジェクト・ドキュメントの管理 : チーム・メンバー

別のまたは二次的なプラットフォームで開発作業中のチーム・メンバーの役割分担は、実際には基盤となるプラットフォームで開発中のチーム・メンバーの役割分担とほとんど同じです。ただし、大きな違いが 1 つあります。別のプラットフォームであらかじめ作成されたプロジェクトを受け取ったら、次の作業を行います。

- プラットフォーム要件を満たすためのカスタマイズされたアクションとマクロの修正

1.2.4.2.1 プラットフォーム要件を満たすためのカスタマイズされたアクションとマクロの修正

サポートされているすべてのプラットフォーム用に、管理者が Project Builder では、事前定義済みのアクションおよびマクロの等価バージョンが、それらと同じ場所に提供されています。ただし、一部のアクションがカスタマイズされているか、または新規アクションが追加されている場合、新しいプラットフォーム上で動作するようにアクションを編集するか、または等価の新規アクションを作成する必要があります。

1.3 テスト・フェーズでのプロジェクト・ドキュメントの管理

テスト・フェーズは開発作業とは分かれた異なるもの、つまりテストは開発した後に行うものと考えられることがよくあります。しかし実際には、開発チームに有益な情報を与える、同時進行で行うプロセスです。

Project Builder をテスト・フェーズに統合するためのオプションには、少なくとも次の 3 つがあります。

- テスト担当者は、Project Builder をインストールしません。管理者が Project Builder の機能を使用して、テストするモジュールのコンパイルおよびソース制御を行ってから、テスト担当者に渡します。テスト担当者の手順は変更ありません。
- テスト担当者は、開発者が使用する 1 つ以上のプロジェクトをインポートします。
- 開発プロジェクトに基づいているが、テスト担当者がインポートするようにカスタマイズされているプロジェクトを作成します（たとえば、サポート・ドキュメント、仕様書またはソースを含まない）。

推奨事項 : 2 番目と 3 番目のオプションを組み合せるのが最適な方法です。作成したアプリケーションとプロジェクトを対応付ける方法も、テスト・フェーズでは役立ちます。テスト・スクリプトを自動的に実行するアクションを作成するか、またはスクリプト・タイプを追加してテストするモジュールに依存させることができます。

単体テストの段階では、プロジェクトが機能単位で編成されているか、または分けられたプロジェクトが機能単位ごとに作成されている場合、テスト担当者は開発者と同じプロジェクトを使用できます。プロジェクトはエクスポートすることもできるので、単体テストはさまざまな環境およびプラットフォームで実行できます。

システム・テストでは、特に、より小規模な複数のプロジェクトを連結する必要がある場合はテストするモジュールのみを含む、開発プロジェクトの縮小された新しいバージョンが必要な場合があります。

1.3.1 開発側の作業

このフェーズにおける開発グループの目標は、できる限り円滑にテストするモジュールをテスト・グループに渡すことです。

1.3.1.1 テスト・フェーズ : プロジェクト管理者

テスト用プロジェクトの作成およびエクスポートに関連する作業は、プロジェクトを作成して開発チームにエクスポートする場合に必要な作業と同じです。

- 配布可能モジュールに基づいてテスト・プロジェクトを作成します（オプション）。
- テスト・バージョンを作成します。
- 異なるテスト環境にプロジェクトをエクスポートします。

1.3.2 テスト側の作業

テスト・チームのメンバーは、通常、アプリケーションのモジュールへの変更について責任を負いませんが、入力（テストするモジュール）と成果物（完全にテストされたモジュールおよびテスト・フェーズでは解決できなかったバグのリスト）があります。

Project Builder は、開発チーム・メンバーの場合と同様に、テスト・チームでも入力および成果物を追跡し記録するのに役立ちます。テスト担当者は、スクリプトやログのプロジェクトへの追加、デバッグ・オプションを含めるようなアクションの変更、およびテスト情報を含むサブプロジェクトの追加などの処理が可能です。

1.3.2.1 テスト・フェーズ : テスト担当者

Project Builder を使用して、作成したアプリケーションをテストすることに決まったら、開発者が最初にプロジェクトを設定した際に行ったのとほぼ同じ準備作業を行う必要があります。その場合は、次のことを行う必要があります。

- テスト・プロジェクトをインポートして、テスト環境を設定します。
- テスト・スクリプトとテスト・データをプロジェクトに追加します。
- テストで役立つようにアクションおよびマクロを変更します。

1.3.2.1.1 テスト・プロジェクトのインポートおよびテスト環境の設定

テスト・プロジェクトのインポートとテスト環境の設定を行うプロセスは、開発でのプロジェクトのインポートとテスト環境の設定を行うプロセスと同じです。詳細は、1.2.2 項「プロジェクトの作成」を参照してください。

1.3.2.1.2 プロジェクトへのテスト・スクリプトおよびテスト・データの追加

テスト・スクリプトなどの項目をプロジェクトに追加することが必要な場合があります。また、テスト・データが含まれたデータベース・アカウントに接続文字列を追加することが必要な場合もあります。

作成したアプリケーション内のモジュールに対応付けられたアクションを自動化できるように、テスト・スクリプトの実行も自動化できます。

1.3.2.1.3 テストに役立てるためのアクションおよびマクロの変更

「デバッガ付きで実行」を指定したアクションがまだない場合は、既存のアクションを変更してデバッグ・フラグを挿入するか、または新規アクションを作成できます。

1.4 リリース・フェーズでのプロジェクト・ドキュメントの管理

作成したアプリケーションを徹底的にテストしリリースできる段階になったら、Project Builder で顧客にアプリケーションを配布するプロセスを簡素化できます。

1.4.1 開発側の作業

リリース・フェーズでは、インストールに必要なすべてのモジュールのテストと検証が完了したバージョンが開発側からリリース担当者に渡されます。Project Builder では、最終的なアプリケーションに含まれたすべてのモジュールにマークが付けられ、それらに特別なコマンドが対応付けられるため、プロジェクトのコンパイルやソース制御などの他のプロセスと同じ方法でこの受渡しを自動化できます。

1.4.1.1 リリース・フェーズ : プロジェクト管理者

作成したアプリケーションを徹底的にテストし、リリースできる段階になったら、残りの作業はプロジェクトのパッケージだけです。プロジェクトのパッケージ

1.4.1.1.1 プロジェクトのパッケージ

Project Builder は配布ウィザードを提供します。配布ウィザードは、アプリケーションをインストール可能なコンポーネントとしてパッケージするのみでなく、次のような機能も提供します。

- 完成したプロジェクトの作業用領域へのコピーまたは FTP。作業用領域からファイルを配布メディアにコピーしたりアーカイブしたり、あるいは内部的に使用可能にしたりできます。
- プロジェクトを Oracle Installer を通じて Windows 95 および NT にインストール可能にするのに必要なスクリプトの生成。Forms Developer または Reports Developer のランタイム環境をプロジェクトと同時にパッケージすることも可能なため、Oracle Installer

を 1 度起動するだけで、アプリケーションのパッケージ全体に加え、必要に応じてランタイム環境もインストールすることができます。

- ファイルを TAR または ZIP 形式にするようにカスタマイズした配布アクションの実行

配布ウィザードが実際にパッケージするモジュールは、各項目に関連付けられた**「配布ファイル」**プロパティの値によって判定されます（「はい」の場合はパッケージにモジュールを追加し、「いいえ」の場合はそのままにします）。

1.5 完成したアプリケーションの配布

作成したアプリケーションをパッケージしたら、顧客にアプリケーションを使用してもらうことができます。顧客は、アプリケーションのインストールだけでなく、アプリケーションが依存するランタイム環境をインストールするためにも Oracle Installer を使用する必要があります。顧客のインストール・プロセスを簡素化するため、Forms Developer および Reports Developer には Oracle File Packager が付属しています。これによって、Windows NT および Windows 95 上に Oracle Installer でアプリケーションをインストールできるようになります。この項のステップが終了したら、顧客は 1 つの機能を使って、アプリケーションと必須のランタイム環境という必要な要素をすべてインストールできます。

1.5.1 はじめる前に

作成されたアプリケーションのパッケージ方法を説明する前に、次に示す、インストールやパッケージ・プロセスに関連する用語および背景知識をよく理解しておく方がよいでしょう。

- 1.5.1.1 項「用語」
- 1.5.1.2 項「Oracle Installer ファイル」
- 1.5.1.3 項「TEMPLATES ディレクトリの内容」

1.5.1.1 用語

この表には、インストールおよびパッケージ・プロセスで重要な用語が定義されています。

用語	定義
作業用領域	ファイルおよびインストール・スクリプトが置かれる PC またはネットワーク上の領域で、ここから配布媒体にコピーされる。
配布媒体	ユーザーがアプリケーションをインストールする媒体セット（たとえば、テープ、CD またはフロッピーディスク）。

用語	定義
インストール可能なコンポーネント	Oracle Installer ファイル（MAP、VRF、INS および DEI）のセットが含まれた製品（たとえば、Forms Runtime、GUI Common Files など）。
製品ファイル（PRD ファイル）	任意の作業用領域でインストールできるコンポーネントがすべてリストされたファイル。
Oracle File Packager	Oracle Installer ファイルを使った Windows アプリケーションのインストールを可能にするのに必要な、製品ファイルおよび Oracle Installer ファイル（MAP、VRF、INS および DEI）を作成するウィザード。

1.5.1.2 Oracle Installer ファイル

Oracle Installer ファイルによって、ユーザーのマシンでアプリケーションをインストール（または削除）する方法および場所が制御されます。Oracle File Packager で Oracle Installer ファイルが作成されますが、その一部を手動で変更することが必要な場合があります。Installer ファイルのサンプルをご覧になる場合は、次の位置を参照してください。

```
¥TEMPLATES¥RELEASE¥YOURAPP

    ¥FORMSAPP

        FORMSAPP.MAP
        FORMSAPP.VRF
        FORMSAPP.INS
        FORMSAPP.DEI

    ¥DEV2KAPP

        DEV2KAPP.MAP
        DEV2KAPP.VRF
        DEV2KAPP.INS
        DEV2KAPP.DEI
```

これらのファイルはすべてテキスト・ファイルで、テキスト・エディタでの参照および編集が可能です。

1.5.1.2.1 PRD ファイル

PRD ファイルには、任意の作業用領域でインストールできるコンポーネントがすべてリストされています。また、これにはベースのファイル名と各コンポーネントの Oracle Installer ファイルの場所が指定されています。つまり、PRD には Oracle Installer の「使用可能な製品」ペインに表示されるファイルがすべてリストされます。その PRD の名前は、

WIN95.PRD および NT.PRD など、それが記述されるプラットフォームを示しています。作業用領域ごと、プラットフォームごとに 1 つの PRD ファイルがあります。

列名	説明
####	製品番号。これは、変更しないでください。
Product	作成したアプリケーションを識別するために使用する一意の名前。
Parent	“ root ”のままにします。
Filename	MAP、VRF、INS および DEI インストール・スクリプトのファイル名。
Version	作成したアプリケーションのバージョン番号。
Interface Label	Oracle Installer の「使用可能な製品」ウィンドウに表示されるアプリケーション名。
Location	インストール・スクリプト・ファイル（MAP、INS、VRF および DEI）と、作成したアプリケーションを構成するすべてのファイルが含まれたディレクトリへの相対パス。
Size	インストール可能なコンポーネントの合計サイズ。CHECKMAP ユーティリティで自動的に設定します。
Visible?	Oracle Installer の「使用可能な製品」ウィンドウでコンポーネントを参照可能または参照不可にします。
Selected?	Oracle Installer の「使用可能な製品」ウィンドウでコンポーネントを選択または選択解除します。
Open?	親または子のコンポーネントに使用。このフィールドを変更する必要はありません。
説明	作成したアプリケーションの説明。
Volume	「Filename」フィールドに表示されるものと一致する必要があります。CD または LAN のインストールでは使用しません。

1.5.1.2.2 MAP ファイル

MAP ファイルは、作成したアプリケーションを構成するすべてのファイルがリストされた表です。

列名	説明
Source	ユーザーのマシンにコピーするファイル。
Destination	ファイルがコピーされるディレクトリ。

列名	説明
Group	プログラム項目が含まれるプログラム・グループ。
項目	メニューに表示される項目またはアイコンの名前。
Command	項目またはアイコンの起動時に実行されるコマンド。書式は次のとおりです。 コマンド・ライン working_directory alternate_icon working_directory と alternate_icon はオプションです。ただし、コマンド・ラインのみが表示される場合は、セミコロンで終わる必要があります。

注意:「Group」、「Item」および「Command」は、「スタート」メニューに表示されるアプリケーションの場合のみ必要になります。

1.5.1.2.3 VRF ファイル

VRF ファイルでは、正しい依存関係がすべて識別およびインストールされていることが検証されます。たとえば、Forms Runtime に依存するアプリケーションを指定すると、アプリケーションのインストール・プロセスで、ユーザーのマシンに Forms Runtime がすでに存在するかどうかが自動的に検出されます。まだ存在しない場合は、Forms Runtime がインストールされます。

また、ユーザーは VRF ファイルから製品のインストール位置などの情報を入力するように要求されます。さらに、VRF ファイルにはユーザーの環境が設定され、Windows レジストリに環境変数などの情報が定義されます。

1.5.1.2.4 INS ファイル

INS ファイルによって、インストール可能なコンポーネントの構成および必須の環境変数の設定、Oracle Installer への製品登録を行うファイルがインストールされます。これは、MAP ファイルおよび VRF ファイルとともに機能します。

1.5.1.2.5 DEI ファイル

DEI ファイルによって、インストール可能なコンポーネントを構成するファイルが削除されます。また、削除が完了すると環境変数が削除され、コンポーネントの登録が取り消されます。これは、MAP ファイルとともに機能します。

1.5.1.3 TEMPLATES ディレクトリの内容

TEMPLATES ディレクトリには、作業用領域の設定およびカスタマイズに必要なものがすべて含まれています。Oracle 配布媒体の TEMPLATES ディレクトリには、次のものが含まれています。

- RELEASE サブディレクトリ : 作業用領域の作成の出発点として機能します。

- RELEASE¥INSTALLR¥INSTALL¥WIN95.PRD: 次に示す、Windows 95 上の Forms、Reports および Graphics Runtime 環境のインストール可能なコンポーネントがリストされた PRD ファイルです。
  - Required Support Files
  - System Support Files
  - GUI Common Files
  - Tools Utilities
  - Forms Runtime
  - Reports Runtime
  - Graphics Runtime
- RELEASE¥INSTALLR¥INSTALL¥NT.PRD: Windows NT 上の Forms、Reports および Graphics Runtime 環境のインストール可能なコンポーネントがリストされた PRD ファイルです（コンポーネント・リストの前の箇条書きを参照）。

1.5.2 アプリケーションのインストール可能化

この項には、顧客が 1 ステップまたは複数ステップのインストール・プロセスを作成する方法が含まれています。

- １ステップ・プロセス : 顧客は、1 つの PRD ファイルからアプリケーションおよび必要なランタイム環境をインストールします。この他に考えられるのは、顧客が Oracle Installer を一度起動してから、必要となるアプリケーションと必須のランタイム環境を加えて、すべてインストールする方法です。
- 複数ステップ・プロセス : 顧客は、多数の異なる作業用領域からアプリケーションをインストールします。この方法は、多数の Forms Developer または Reports Developer アプリケーションを分類する必要がある場合、または顧客がすでに共通の領域から必須のランタイム環境を使用できる場合に有効です。

どちらのプロセスを選択しても、Oracle Installer によってアプリケーションをインストールできるようにするには、次のいずれかを行います。

- Oracle 配布媒体から、自分専用の作業用領域の出発点として機能するように、TEMPLATES¥RELEASE ディレクトリを自分のマシンにコピーします。
- Oracle Installer でのアプリケーションのインストールを可能にするために必要な PRD、MAP、VRF、INS および DEI ファイルを、Oracle File Packager を使用して作成します。
- 自分の開発領域から作業用領域にファイルをコピーします。その場所から、ファイルを配布媒体にコピーできます。

この章では、この後、これらの作業を完了するための特定の方法を説明します。

1.5.2.1 作成したアプリケーションの Windows 上での利用

作成したアプリケーションを Windows 95 および NT 上にインストールできる場合は、Oracle File Packager を使用して、Oracle Installer ファイルを作成し、開発領域から作業用領域にファイルをコピーできます。次のステップでは、1 ステップおよび複数ステップのインストール方法を説明します。

ステップ 1: Oracle File Packager のインストール

1. （Oracle 配布媒体にある）TEMPLATES¥OISFP10 から SETUP.EXE をクリックして、Oracle Installer を起動します。SETUP.EXE によって、実行中のオペレーティング・システムが認識され、該当する Oracle Installer が起動されます。
2. インストール可能な製品のリストから、「Oracle File Packager」を選択します。
3. 表示される指示に従って、インストール・プロセスを完了します。

ステップ 2: 作業用領域の準備

1. TEMPLATES¥RELEASE を PC またはネットワーク・ドライブ上のドライブにコピーします。
2. 作成したアプリケーションが NT 環境用であっても、TEMPLATES¥RELEASE¥FORWIN95 下に作成したアプリケーションのサブディレクトリを作成します。

   複数のアプリケーションをインストールする場合は、それぞれのサブディレクトリを作成します。

ステップ 3: 作業用領域へのファイルの移動、および Oracle Installer ファイルの作成

ステップ 2 で設定した各作業用領域について、このステップを繰り返します。

1. 「スタート」メニューから「Oracle for Windows NT」または「Oracle for Windows 95」を選択後、「Oracle File Packager」を選択します。
2. オンライン・ヘルプを使用して、Oracle File Packager に提示されたステップに従います。

注意 :

- ステップ 3 で指定した内部文字列は、使用している Oracle Installer ファイル（MAP、INS、VRF および DEI）に含まれています。
- 「Staging Area Location」の入力を要求されたら、TEMPLATES¥RELEASE¥FORWIN95 下のサブディレクトリを指定します。

ステップ 4: PRD ファイルの NT.PRD および WIN95.PRD のマージ

このステップでは、1 ステップのインストール・プロセスを作成します。複数ステップのインストールの場合は、ステップ 5 に進んでください。

1. 作成したアプリケーションの PRD ファイルから作成された行をコピーして、RELEASE¥INSTALLR¥INSTALL¥WIN95.PRD または RELEASE¥INSTALLR¥INSTALL¥NT.PRD（あるいはその両方）にペーストします。

ステップ 5: Oracle Installer ファイルの変更

1. 「スタート」メニューのアイコンとしてアプリケーションを表示する場合、「Group」、「Item」および「Command」フィールドをアプリケーションの MAP ファイルに追加します。
2. アプリケーションに依存関係を設定する場合は、VRF ファイルに追加します。

   たとえば、アプリケーションが Forms Runtime に依存するように設定する場合、インストール・プロセスで、ユーザー・マシンに Forms Runtime がすでに存在するかどうかが自動的に検出されます。まだ存在しない場合は、Forms Runtime がインストールされます。
3. 各作業用領域で SETUP.EXE をクリックして、Oracle Installer を起動します。「使用可能な製品」ペインにリストされたファイルを調べます。このペインにファイルを表示したくない（たとえば、VRF ファイル内でファイルの依存関係がすでに設定され、明示的にインストール対象にする必要がない）場合、作業用領域の PRD ファイルを編集して、ファイルの「Visible?」を参照不可に変更します。

ステップ 6: インストール内容のテスト

予想されるエンド・ユーザー環境の典型である“ クリーンな ” マシン（まだ何もインストールされていないマシン）でインストール内容をテストします。開発者のマシンで行ったテストは信用しないでください。そのマシンには、MAP ファイルから不注意に削除されたアイコンやライブラリなどのファイル、または INS ファイルに含まれていないレジストリ設定がすでに存在している可能性があるためです。これは、インストールで問題を起こす、最も共通した原因の 1 つです。

1. アプリケーションをインストールし、必要なコンポーネントがインストールされたことを確認します。
2. アプリケーションを起動して、正しく実行されていることを確認します。
3. Oracle Installer を使用して、アプリケーションの削除をテストします。

ステップ 7: 作業用領域への配布媒体のコピー

作成したアプリケーションを CD、テープ、フロッピーディスク、または他の媒体にコピーできる、あるいは単に LAN や他のネットワーク・マシンにコピーできる段階になったら、必ず、作業用領域全体、つまり TEMPLATES¥RELEASE をそのまま含めるようにしてください。サブディレクトリのみを含めた場合は、必須のランタイム環境にアクセスできなくなります。

2

視覚的効果の高いアプリケーションの設計

この章では、グラフィカル・ユーザー・インタフェース（GUI）を開発する場合に役立つガイドラインを示します。

項	説明
2.1 項「プロセスの理解」	GUI ベースのアプリケーション開発プロセスを簡単に説明します（すでに GUI ベースのアプリケーションを開発した経験があれば、この項を読み飛ばして、2.1.3 項「ユーザー・インタフェースの計画」をお読みください）。
2.2 項「効果的なフォームの作成」	フォーム作成で考慮する必要のある視覚的効果について解説します。この項では、最初に基本的な Form Builder の専門用語を概説し、標準を実施する手段としてのオブジェクト・ライブラリの重要性を説明した後、各 GUI でフォーム・オブジェクトを使用するためのガイドラインを示します。
2.3 項「効果的なレポートの作成」	レポート・オブジェクトの配置および改ページの発生場所の制御方法を理解するために役立ちます。
2.4 項「効果的な図表の作成」	データを図形表示する場合の視覚的な考慮事項をいくつか示します。

2.1 プロセスの理解

効果的な GUI の開発プロセスを理解するよりも、そのアプリケーションを使用するユーザーについて理解することが重要です。実際、効果的なアプリケーションを開発するには、そのアプリケーションで実行される作業、作業の実行順序、実行環境、および期待される機能など、ユーザーについて理解する必要があります。

他の多くのアプリケーション開発と同じ方法で行おうとするならば、この方法は、重点を根本的に変更することが必要になるでしょう。アプリケーションの開発では、通常、内側から外側に作成していきます。データソースからコードへ、コードから GUI へと効果的な GUI を開発する場合、これとは逆のプロセスで行わなければなりません。つまり、最初にユー

ザーと話し合い、次にユーザー特有の業務をサポートするユーザー・インタフェースを設計して、最後にすべての作業を実行する基本となるコード・ベースを作成します。

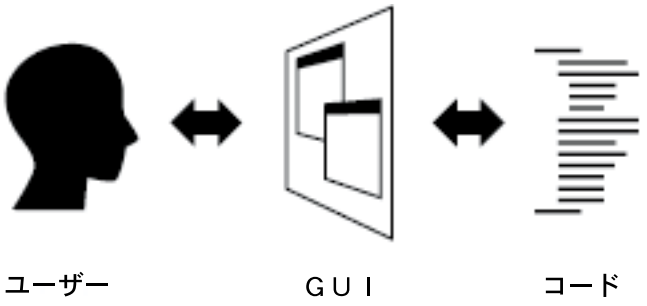


図 2-1 ユーザーを第一に考える

パッケージされた一連の標準やガイドラインを使用して開発しても、ユーザーのニーズを正確に理解して開発するかわりにはなりません。この章は、ユーザーの特定のグループ専用のインタフェースを独自に作成する場合や、ユーザーについての理解を深めるために役立ちます。

2.1.1 ステージとは

図 2-2 に示すように、GUI 開発のプロセスには 4 段階の主なステージがあります。

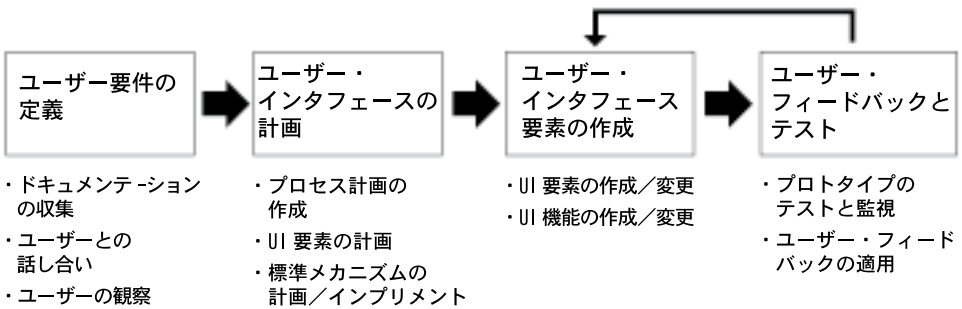


図 2-2 ユーザー・インタフェース開発のステージ

この項の残りの部分は、これらの各ステージを達成するためのガイドラインです。

- 2.1.2 項「ユーザー要件の定義」
- 2.1.3 項「ユーザー・インタフェースの計画」
- 2.1.4 項「ユーザー・インタフェース要素の作成」
- 2.1.5 項「ユーザー・フィードバックの収集」

注意 : この章では、GUI 開発のあらゆる問題についての対処方法を網羅しているわけではありません。特定のステージでの作業の進め方について詳細を知りたい場合は、図書館またはコンピュータ書籍を扱っている書店で入手できるでしょう。ユーザー要件の定義およびユーザー・フィードバックの収集の情報源としては、『Handbook of Usability Testing』（Jeffrey Rubin 著）が特に優れています。

2.1.2 ユーザー要件の定義

GUI 開発の最初のステージでは、ユーザーのニーズおよび現在のアプリケーションから作成可能であるものを判断します。このステージを飛ばして次の設計フェーズに移りたいと思われるかもしれませんが、リスクが大きいので注意が必要です。対象となるユーザーおよび実行したい作業について十分理解していないと、効果的な GUI の作成は事実上不可能です。

ユーザー要件を定義するには

- **ドキュメンテーションの収集** : 関連する方針、プロシージャ・マニュアルおよびシステムに関する既存のドキュメンテーション（電子化されたものまたはハードコピー）は、ユーザーとの話し合いで必要なバックグラウンドを明確な形で示す場合に役立ちます。
- **ジョブを処理するユーザーの観察** : ユーザーが実行する作業および実行する作業の順序のリストを作成します。
- **ユーザーとの話し合い** : GUI ベースのシステムで求められているものを把握します。話し合いを行う場合、次の点を考慮します。
  - ユーザーが行う作業のみでなく、作業方法も尋ねます。たとえば、1 人の事務員が同時に複数の注文を処理する必要があるか、または 1 つの注文を処理するだけでよいのかを尋ねます。
  - ユーザーが現行のシステム（コンピュータ化されていなくても）に満足しているか、不満があるかを把握します。
  - ユーザーに GUI の構想を尋ねます。その構想をできるだけ詳しく説明してもらいます。
  - ユーザーをよく理解するようにします。ユーザーは、通常、長時間にわたって同じジョブを行っているか、またはさまざまなジョブを次々に行っているかを把握します。アプリケーションを一定の時間に使用するか、またはたまにしか使用しないかも知る必要があります。あまり使用しないアプリケーションの場合、多くのボタン、テキストおよびガイドラインを用意して、作業時間の無駄を削減できます。日

常的に使用するアプリケーションの場合、ショートカットやアクセラレータ・キーを用意すれば、操作に慣れているユーザーは作業を速やかに完了できます。

- ユーザー側で不具合があるかどうか、または考慮する必要のある特定の状況を調べます。たとえば、ユーザーがアプリケーションを使用するときに立って作業をしているかどうかが挙げられます。ユーザーが立って作業している場合、過度のナビゲーションに費やす時間と精神的な余裕はありません。

- **さまざまなユーザーのサンプル採取**: 1 つのカスタマ・サイトでのユーザーからのフィードバックでは、特定の経験に偏ってしまいます。

2.1.3 ユーザー・インタフェースの計画

第 2 のステージでは、ユーザーのニーズを満たすユーザー・インタフェースをインプリメントする方法を計画して文書にします。次のものが含まれます。

- 遵守する一連の標準を開発し、必要に応じて自分のチームに標準として採用させます。2.1.3.1 項「標準の作成」を参照してください。
- プラットフォーム固有の要件および利用環境におけるその他の制限事項を考慮します。2.1.3.2 項「移植性の検討」を参照してください。
- 各画面を詳しくマップして、ユーザーのニーズを満たす効果的な表示をするために、どのタイプのインタフェース要素を使用するかを決めます。2.1.3.3 項「プロトタイプの作成」を参照してください。

2.1.3.1 標準の作成

一貫性のある一連の開発標準は、あらゆる開発の作業を成功させるために欠かせません。さまざまな GUI 要素のレイアウト、使用および動作に関する標準を開発し、実施することで、アプリケーションから独立している部分にも共通のルック & フィールを装備できます。

Forms Developer および Reports Developer では、複数のメカニズムが提供されており、一貫性のある一連の標準を開発できます。

表 2-1 標準のメカニズム

メカニズム	説明
オブジェクト・ライブラリ（Form Builder）	オブジェクト・ライブラリは、開発者が作成して開発チーム全体で使用できるようにした一連のオブジェクトおよび標準です。サブクラスで使用すれば、オブジェクト・ライブラリ内のあるオブジェクトに対して加えられた変更内容が、そのオブジェクトが使用されるすべてのアプリケーションに確実に波及するようにできます。**オブジェクト・ライブラリとは、使用している Form Builder アプリケーションの標準化に際して選択されたメカニズムのことです。** Form Builder には、独自のサイト要件に見合うようにカスタマイズできる 2 つのオブジェクト・ライブラリがあります。 ■ 標準オブジェクト・ライブラリ。Windows 95/NT 環境用に最適化された標準が含まれています。 ■ Oracle アプリケーション・オブジェクト・ライブラリ。Windows 95/NT、Solaris およびキャラクタ・モードなどの標準のプラットフォーム間アプリケーションが含まれています。 詳細は、Form Builder のオンライン・ヘルプのトピック「オブジェクト・ライブラリについて」および「サブクラス化について」を参照してください。
オブジェクト・グループ（Form Builder）	オブジェクト・グループは、オブジェクトのグループのコンテナです。関連オブジェクトを他のモジュールにコピーしたり、サブクラスにするためにパッケージ化したいときに、オブジェクト・グループを定義します。 たとえば、複数のタイプのオブジェクトを使用した面会スケジューラを作成するとします。オブジェクトは、日付と面会内容を表示するための、ウィンドウ、キャンバス、ブロックおよび項目と、スケジュールや他の機能のロジックを含むトリガーです。これらのオブジェクトをオブジェクト・グループにパッケージ化すると、単純な操作 1 つで、他のフォームにすべてのオブジェクトをコピーすることができます。 詳細は、Form Builder のオンライン・ヘルプのトピック「オブジェクト・グループ作成のためのガイドライン」を参照してください。

表 2-1 標準のメカニズム

メカニズム	説明
可視属性（Form Builder）	可視属性は、フォームに設定するフォント、カラーおよびパターンなどのプロパティで、ご使用のアプリケーションの GUI に表示されるメニュー・オブジェクトです。可視属性は、次のプロパティに組み込むことができます。 ■ フォント・プロパティ：フォント名、フォント・サイズ、フォント・スタイル、フォント間隔およびフォントの太さ ■ カラーおよびパターン・プロパティ：フォアグランド・カラー、バックグランド・カラー、塗りパターン、文字モードの論理属性および白黒 詳細は、Form Builder のオンライン・ヘルプのトピック「可視属性作成のためのガイドライン」を参照してください。
テンプレート（Form Builder、Report Builder）	Form Builder では、テンプレートを作成して、チームの他のメンバーに新しいフォームのデフォルトの出発点を提供できます。テンプレートには、通常、グラフィックス（会社のロゴなど）、プログラム単位、標準のウィンドウのレイアウト、ツールバー、メニューおよび他の共通オブジェクトなどの一般オブジェクトが組み込まれています。 Report Builder では、独自のテンプレートを作成してレポートの外観を制御できますが、さまざまな事前定義テンプレートを利用することもできます。レポート・ウィザードを使用して、レポートに組み込むオブジェクトを選択した後で、テンプレートを選択してこれらのオブジェクトを配置し、標準のフォーマット属性を適用します。 詳細は、Form Builder または Report Builder のオンライン・ヘルプの索引から「テンプレート」を検索してください。

2.1.3.2 移植性の検討

アプリケーションを複数の環境で利用する場合には、各プラットフォームでのさまざまな GUI 要素の作成方法、およびプラットフォーム相互の要素の制限事項を理解しておくことが重要です。たとえば、プラットフォーム間でのフォーマットの制限のために、Windows 用に作成した対話形式のボタンが小さくなり、Solaris で表示したときに読みにくくなります。プラットフォーム固有の制限事項とそれらに関連した作業のガイドラインの詳細は、第 5 章「移植性のあるアプリケーションの設計」を参照してください。ユーザー・インタフェースの大きな制限要素となる文字モードの検討事項についても、同じ章を参照してください。

2.1.3.3 プロトタイプの作成

プロトタイプは、ご使用のアプリケーションを使いやすくするための最も効率的な手段です。最も効果的なプロトタイプは、対話的な開発モデルに従って、ストーリーボードで始まり、十分に機能的なアプリケーションで終了します。プロセスは、次の手順のとおりです。

1. アプリケーションの実際の表示や動作方法についての明確な青写真ができるように、ストーリーボードを下書きします。ストーリーボードとは、画面を枠ごとに描画したもので、複数の画面間の切替えおよび外観が示されます。ユーザー要件を定義したときに指定した作業と各画面を関連付ける方法を記述するストーリーを組み込みます。

   次に、製品発注用のアプリケーションのストーリーボードから 3 つのパネルの例を示します。

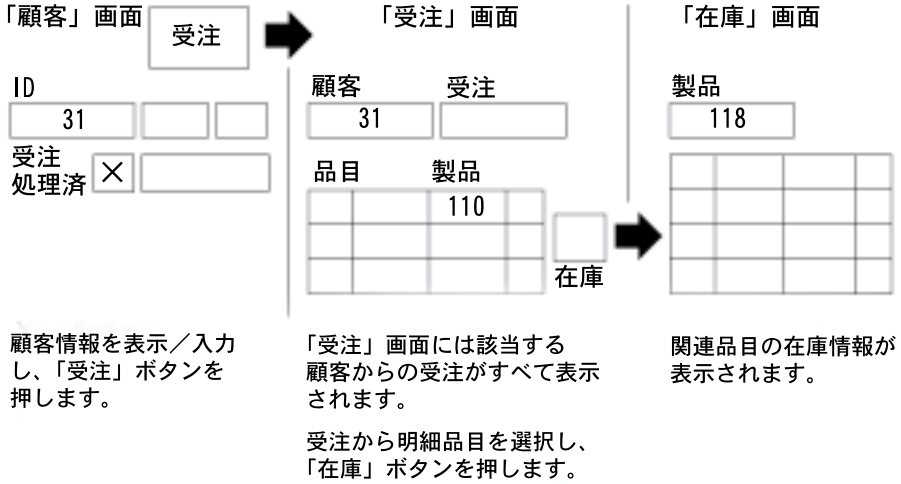


図 2-3 ストーリーボードの例

2. ストーリーボードをユーザーに示します。計画したアプリケーションでユーザーのニーズが対処され、**実行方法どおりに**作業がサポートされていることを検証します。

3. ストーリーボードをペーパー・プロトタイプに拡張します。ストーリーボードが上位レベルでの作業およびウィンドウ・フローの概略示すのに対して、ペーパー・プロトタイプでは、アプリケーション全体をかなり詳細に図解したものです。ペーパー・プロトタイプでは、通常、計画した各ウィンドウごとに 1 枚のペーパーがあり、ウィジェットを用いて仕上げ、作業の流れとナビゲーションを示す矢印などが含まれます。

4. ペーパー・プロトタイプをユーザーに示します。ボタンの配置およびサポートされるダイアログのレイアウトなどの詳細を絞り込むことができるように、構成に関する問題の多くは、ストーリーボードのフェーズで識別しなければなりません。ユーザーに対して

セッションを指示する場合の助言は、2.1.5 項「ユーザー・フィードバックの収集」を参照してください。

5. ユーザーからのフィードバックに基づき、Forms Developer または Reports Developer を使用して機能的なプロトタイプを作成します。次の項は、プロトタイプに対して適切なオブジェクトを選択するときに役立ちます。
   - 2.2 項「効果的なフォームの作成」
   - 2.3 項「効果的なレポートの作成」
   - 2.4 項「効果的な図表の作成」
6. 機能的なプロトタイプをユーザーに実地で試してもらいます。以前のセッションに関与していないユーザーを含むようにしてください。新規ユーザーがそのアプリケーションを簡単に理解できるかどうかを判断するためです。
7. ユーザー要件に示されたすべての目的が達成されるまで、ステップ 5 およびステップ 6 を繰り返します。

2.1.4 ユーザー・インタフェース要素の作成

ユーザー・インタフエースの作成を開始するには、事前に十分な時間をかけて概念モデルを開発しておかなければなりません（つまり、ユーザーおよびユーザーが実行する作業を十分理解し、それらの作業がサポートされるダイアログのフローをスムーズに設計しておく必要があります）。この章では、次の 3 項が、ユーザー・インタフェース要素を慎重に選択する場合に役立ちます。

- 2.2 項「効果的なフォームの作成」
- 2.3 項「効果的なレポートの作成」
- 2.4 項「効果的な図表の作成」

2.1.5 ユーザー・フィードバックの収集

ペーパー、あるいは Forms Developer または Reports Developer で実行されるプロトタイプをすでに開発してある場合は、最初のフェーズで話し合ったユーザーに試験的に実行してもらいます。ユーザー・フィードバックを効果的に収集するには、次のようにします。

- 作業ベースのアプローチを使用して、ユーザー・テストの方法を作成します。
- 広い視野で見るために、少なくとも 6 種類の基本的なタイプのユーザーを使用します。
- メモ、音声およびビデオによる監視を使用して、ユーザーのアクティビティを記録します。
- プロトタイプのパフォーマンスについてユーザーに質問します。
- 複数の設計者に結果を解釈してもらいます。

注意 : ユーザー・インタフェースが適切であれば、開発中のアプリケーションを実際に使用するユーザーのみがコメントを入れることができます。

ユーザーによるプロトタイプのテストが終了し、ユーザーからのフィードバックを収集したら、作成ステージに戻り、フィードバックに従ってユーザー・インタフェースを変更し、変更後にもう一度テストします。要件定義フェーズで定義した目的が達成されるまで、この作業のサイクルを繰り返します。

2.2 効果的なフォームの作成

この項では、Form Builder を使った効果的な GUI の作成方法を説明します。

注意 : この項は、欧州中心の観点から記述されています（欧州以外の顧客を対象に開発する場合は、ユーザーの文化的な背景に配慮する必要があります。実際には、ターゲットとなる客層の複数のメンバーに設計内容を見直してもらいます）。

2.2.1 フォームの理解

フォームの特定の検討事項を記述する前に、いくつかの基本的なフォームの概念を簡単に紹介します（Form Builder を使用した経験があれば、2.2.2 項「フォーム作成のガイドライン」に進んでください）。これらと他のレポートに関連したトピックの詳細は、Report Builder のオンライン・ヘルプまたは Forms Developer クイック・ツアーの「Report Builder」の項、あるいはその両方を参照してください。

2.2.1.1 モジュールとは

Form Builder を使用してアプリケーションを作成する場合、モジュールと呼ばれる個々のアプリケーション・コンポーネントを使用して作業します。Form Builder には、次の 4 種類のモジュールがあります。

モジュールのタイプ	説明
Form モジュール	オブジェクトおよびコード・ルーチンの収集。フォーム・モジュールで定義できるオブジェクトには、ウィンドウ、テキスト項目（フィールド）、チェックボックス、ボタン、警告、値リストおよびトリガーと呼ばれる PL/SQL コードのブロックがあります。
メニュー・モジュール	メニュー（メイン・メニュー・オブジェクトおよび多数のサブメニュー・オブジェクト）の収集およびメニュー項目のコマンド。
PL/SQL ライブラリ・モジュール	ユーザー命名プロシージャ、ファンクションおよびアプリケーション内の他のモジュールからコールできるパッケージの収集。

モジュールのタイプ	説明
オブジェクト・ライブラリ・モジュール	アプリケーション開発に使用できるオブジェクトの収集。詳細は、表 2-1「標準のメカニズム」を参照してください。

この章では、PL/SQL ライブラリ・モジュールの使用方法は記述していません。このトピックの詳細は、Form Builder のオンライン・ヘルプを参照してください。

2.2.1.2 フォーム、ブロック、項目、領域および枠とは

簡単に言うと、フォーム（またはフォーム・モジュール）は、データソースに格納された情報へのアクセスができるアプリケーションです。フォームを参照すると、チェックボックスおよびラジオ・グループなどのインタフェース「項目」が表示され、ユーザーはデータソースと対話形式で情報をやりとりすることができます。これらのインタフェース項目は、「ブロック」と呼ばれるコンテナに属しています。図 2-4 では、Customer ID、First name、Title などのフィールドはすべて同じブロックに属しています。

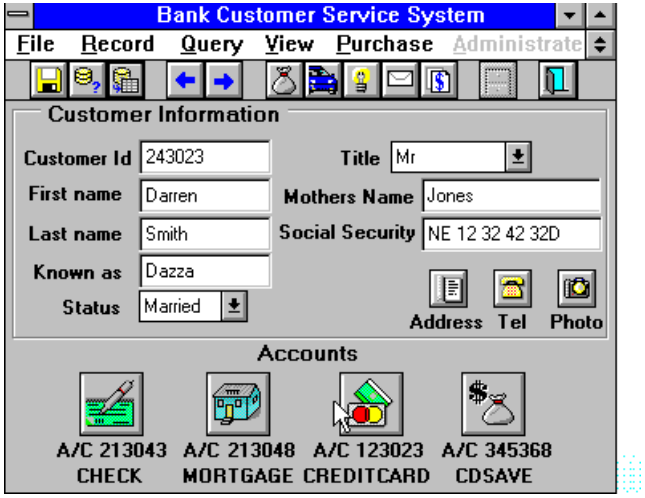


図 2-4 サンプル・フォーム

ブロックには、次の 2 種類があります。データソースとユーザー間のリンクとして使用される「データ・ブロック」と、データソースとは関係付けられていない「制御ブロック」の 2 種類があります。各データ・ブロックによって、ユーザーはデータソース内の 1 つの表にあるデータを表示したりアクセスできます。シングル・レコード・ブロックをブロックとして使用して、1 回にデータの 1 行を表示できますが、マルチ・レコード・ブロックをブロックとして使用すると、一度にデータの多数の行を表示できます。図 2-4 のすべてのフィールドは、シングル・レコード・ブロックです。

「領域」とは、ブロック内の他のブロックと論理グループのフィールドを区切る四角形または線のことです。図 2-4 では、「Customer Information」フィールドを Accounts のアイコンから区切る四角形が領域です。

「枠」とは、あるブロック内に特定の項目を配置する事前定義済みの方法です。たとえば、図 2-4 に示すブロックは、マージンおよびオフセット、項目とプロンプト間の距離などが設定された枠によって配置されています。

2.2.1.3 ウィンドウおよびキャンバスとは

「ウィンドウ」とは、Form Builder アプリケーションを構成するすべての表示可能なオブジェクトが含まれたコンテナです。シングル・フォームには、任意の数のウィンドウを組み込むことができます。通常、最も単純なフォームでも、それに対応付けられたウィンドウが複数あります。次のタイプのウィンドウが使用できます。

ウィンドウのタイプ	説明
コンテナ（MDI）	すべての他のウィンドウが入っています。たいていの場合、ツールバーとメイン・メニューが含まれています。（Windows の場合のみ）
モードレス	ツールバーおよびメニューのように、ユーザーが他の任意のウィンドウと対話できます。モードレス・ウィンドウは、ユーザーが多数の作業から自由に選択するときに、GUI で最も頻繁に使用されます。
モーダル	ユーザーは現在のウィンドウにしかアクセスできず、そこで変更を確定するか取り消すかする必要があります。ツールバーおよびメニューにはアクセスできません。ユーザーが作業を続行する前に特定の作業を完了しなければならないときに、モーダル・ウィンドウを使用します。

次は、一般的な Form Builder ウィンドウの例です。

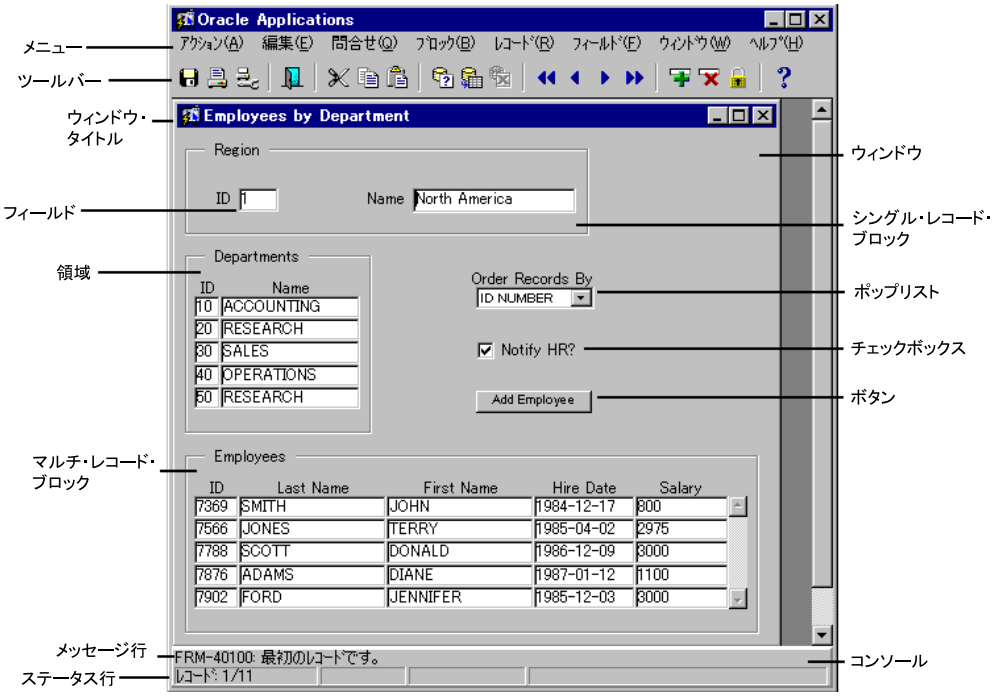


図 2-5 一般的な Form Builder ウィンドウ

ほとんどの Windows 95/NT Form Builder ウィンドウと同じように、このウィンドウには次のものが含まれます。

- ウィンドウのタイトル
- メニューバーおよびプルダウン・メニュー
- データに対応していないボタンや他のコントロール
- ブロック内のデータ項目
- メッセージ行およびステータス行が含まれたコンソール

「キャンバス」は、インタフェース項目が表示されるバックグラウンド・オブジェクトです。キャンバスには、次の 4 種類があります。

キャンバスのタイプ	説明
コンテント・キャンバス	表示されるウィンドウのペイン全体を示します（ウィンドウがスクロール可能な場合は、表示されるペインはより大きなものになります）。各ウィンドウには、少なくとも 1 つのコンテント・キャンバスがあります。
スタック・キャンバス	現行のウィンドウに割り当てられたコンテント・キャンバスの一番上（つまりスタックされている）に表示されます。スタック・キャンバスは、移入されていないフィールドなど、状況によっては不明確になるコンテント・キャンバス内の領域に対して使います。「ビューポート」を使用すれば、スタック・キャンバスの表示範囲を制御できます。
タブ・キヤンバス	1 つの動的なキャンバス上で、大量の関連情報をグループ化し、表示できる一連のタブです。
ツールバー・キャンバス	個々のウィンドウにツールバーを作成するために使用します。

各ウィンドウで、1 つ以上のキャンバスが表示される場合もあります。特定の条件を満たしていれば、状況によって、ウィンドウ内のキャンバスを表示できます。

2.2.2 フォーム作成のガイドライン

次の項では、Form Builder で効果的な GUI を作成するための特定の推奨事項を示します。

- 2.2.2.1 項「オブジェクト・ライブラリの使用」
- 2.2.2.2 項「基本的な設計原理の理解」
- 2.2.2.3 項「カラーの追加」
- 2.2.2.4 項「キャンバスの作成」
- 2.2.2.5 項「ウィンドウの作成」
- 2.2.2.6 項「領域の作成」
- 2.2.2.7 項「ブロックへの項目の追加」
- 2.2.2.8 項「メッセージの設計」
- 2.2.2.9 項「オンライン・ヘルプのインプリメント」
- 2.2.2.10 項「効果的なメニューの作成」

2.2.2.1 オブジェクト・ライブラリの使用

フォームの開発で使用できる最も重要な標準化の手段は、オブジェクト・ライブラリでしょう。オブジェクト・ライブラリは、開発者が作成した一連のオブジェクトおよび標準です。各オブジェクトまたは標準によって、枠、ウィンドウまたは領域の全体の外観とレイアウトが決まります。オブジェクト・ライブラリ内にこれらのオブジェクトを常駐させると、開発プロジェクトまたはサイトのすべての開発者がそのオブジェクトを利用できるようになり、別の場所で作業をしている開発者でも、同じアプリケーションを作成できるようになります(つまり、共通のルック & フィールを使って同じアプリケーション内の別個のモジュールを作成できます)。サブクラスで使用すれば、オブジェクト・ライブラリ内のあるオブジェクトに対して加えられた変更内容が、そのオブジェクトが使用されるすべてのアプリケーションに確実に波及するようにできます。

オブジェクト・ライブラリの使用方針としては、標準の各論理グループに別のオブジェクト・ライブラリを作成するのが効果的です。たとえば、会社の標準に対してオブジェクト・ライブラリを 1 つ作成して社内で利用できるようにする場合、プロジェクトの特定のニーズ専用にオブジェクト・ライブラリを作成します。

Form Builder では、専用のオブジェクト・ライブラリの作成を開始するために 2 種類のサンプルが用意されています。

- 標準オブジェクト・ライブラリ。多言語のサポートが要件に含まれていない場合に、Windows 95 専用で利用されるオブジェクトが入ります。
- Oracle アプリケーション・オブジェクト・ライブラリ。マルチ・プラットフォームでの利用にお薦めします。

オブジェクト・ライブラリを作成する前に、標準または Oracle アプリケーション・オブジェクト・ライブラリの内容を調べて、正しく動作するかどうか、または変更の必要があるかどうかをテストするとよいでしょう。

テストの対象	処置
データ・ブロックの標準オブジェクト・ライブラリ内の項目	1. データ・ブロック・ウィザードを使用してデータ・ブロックを作成します。 2. 制御ブロックまたはデータ・ブロックの項目をクリックしてから、マウスの右ボタンをクリックします。その項目に適用するスマート・クラスのリストが表示されるので、任意のスマート・クラスをクリックします。
制御ブロックの標準オブジェクト・ライブラリ内の項目	1. STNDRD20.OLB をオープンにします。 2. ブロック内に項目をドラッグ・アンド・ドロップします。
標準オブジェクト・ライブラリ内の可視属性のみ	STNDRD20.OLB テンプレート・フォームをオープンします。

テストの対象	処置
Oracle アプリケーション・オブジェクト・ライブラリ内のオブジェクト	APPSTDS.OLB テンプレート・フォームをオープンします。

標準オブジェクト・ライブラリを使用する場合、必ず VAGs タブの下にあるすべての属性をフォームの可視属性ノードにサブクラス化します。多数の標準がこれらの可視属性に基づいているため、可視属性を適用しないと正しく表示されません。可視属性をコピーしないでサブクラス化すると、常に確実に最新の定義にアクセスできます。

すべてあるいはほとんどのフォームで、特定の可視属性グループのセットを使用する予定であれば、標準オブジェクト・ライブラリからサブクラス化された可視属性がすでに含まれているテンプレート・フォームを作成します。レイアウト・ウィザードで入力を要求されたときに、このテンプレートに名前を付けることができます。

2.2.2.2 基本的な設計原理の理解

この項では、フォーム作成の一般的なガイドラインをいくつか示します。

- インチ、センチメートルまたはポイントの各測定単位を使った実際の座標システムを使用してください。単位を 1 つだけ選んで、すべてのモジュールに共通に使用します。

ポイント	すべての数字を使用してサイズを指定できるので、最も使いやすい単位といえます。テキストは常にポイントで指定するので、オブジェクトがポイント単位で指定されていれば、テキストに関係付けられたオブジェクトのサイズの指定はより簡単です。
インチおよびセンチメートル	ポイントと比べて高い精度で指定できます（ただし、テキストとオブジェクトのサイズの比較は簡単にはできません）。ターゲット環境の表示が SVGA 専用またはそれ以上の精度であれば、効果的です。
ピクセル	お薦めできません。これを使用するのは、すべてのユーザーが同じ画面と解像度を使用しており、今後もその状態が続くことが確定している場合のみです。
文字	文字モードのプラットフォームを実行する場合にのみ使用します。

- 必要に応じてダイアログを使用可能または使用禁止にして、ユーザーの作業を制御します。これを判断する場合、ユーザーの作業状況によって、操作を自由にさせるか、または制限するかを考慮し、ユーザーに関する知識に基づいて決定する必要があります。

  例 : アプリケーションに割込みを実行し、後で再開するといった単純な作業については、すべてのユーザーが自由に決められるようにする必要があります。ただし、会社が発行

する請求書をユーザーが整理し直すことができるようにすると、会社の標準が無視される可能性があるため、望ましくありません。

ユーザーが制御する範囲は、ユーザーの経験のレベルによっても違ってきます。操作に習熟したユーザーと未経験のユーザーの両方を対象にアプリケーションを開発する場合には、必要に応じて段階的に操作方法の指示が得られるように、マニュアルに即したウィザードを組み込むことを検討してください。

- 作業が終了したら、ウィンドウをクローズする、別のウィンドウをオープンする、情報メッセージを表示する、のうちいずれかが実行されるようにし、作業の終了がユーザーに明白にわかるようにします。
- 必要な大きさのウィンドウを作成します。
- 情報をグループ化するためにブランク領域を使用します。
- 標準オブジェクト・ライブラリ内の枠オブジェクトを使用して、フォーム・モジュール間で一貫性のあるレイアウト・スタイルが適用されるようにします。
- トップダウンの作業順序に基づいて、画面レイアウトの方向を決めます。左から右へ、次に上から下へと使用順序に従って、ブロック、領域および項目を配置します。
- シングル・レコード・ブロックでは、できるだけ項目を左詰めします（右詰めフィールドには、通貨および数値が入ります）。マルチ・レコード・ブロックでは、項目を横方向にスタックして、それらを一番上の行に配置します。

2.2.2.3 カラーの追加

- ユーザーの注意を喚起するために、使用するカラーの種類は控えます。
- カラーに意味を持たせ、一貫して使用します。例 : 必須またはオプションと、表示専用フィールドのカラーとの区別を付けてコーディングします。他の各フィールドのカラーを同じ方法でコーディングします。
- 場合によっては、ユーザーがカラーを変更できるようにします。
- 状態または他の情報をやり取りする手段としては、カラーのみに依存しないで、音声または他のハイライトなどの代替方法を常に用意しておいてください。たとえば、否定的なメッセージについてはすべて赤色で表示する場合、カラーが使用されていなくてもメッセージが明確にわかるように、カッコを付けます。また、ユーザーがカラーを識別できない場合を考慮して、メッセージ内の特定のカラーが参照されないようにします。
- オブジェクト・ライブラリまたは可視属性を使用して、カラーの使用方法を標準化します。
- カラーを選択するときは、次のことに注意してください。
  - 赤と青の組合せは、目によくありません。
  - 青の文字は画面上では後退して見えるので、背景から見分けにくくなります。

- 濃い青色の背景は、他の明るいカラーと同じように、長時間作業すると目によくありません。
- 色（特に赤と緑）の区別が付けられない人が多数います。
- カラーの持つ意味は、国によって違います。職業および状況などのニーズに注意し、ターゲット市場における文化的なカラーの区別に従ってコーディングしてください。たとえば、緑は、通常ほとんどの西欧諸国では肯定的なイメージが持たれていますが、化学関係の専門家の間では、緑色は「危険」を意味します。

カラー	意味
青	涼しい
黒	利益（経済的）
緑	進め、OK、危険（化学関係）
赤	暑い、止まれ、危険、損失（経済的）
黄	警告、注意

2.2.2.4 キャンバスの作成

次の表には、キャンバス作成のための推奨事項が示されています。

表 2-2 キャンバス作成のための推奨事項

キャンバスのタイプ	推奨事項
一般	■ 項目と領域の間には、十分な空白を入れます。 ■ 個々のキャンバスにオプション情報を入れます。 ■ できるだけ、ウィンドウのスクロールは避けてください。調査の結果、ユーザーが作業を完了するためにウィンドウをスクロールしているときに、急激に生産性が低下することが判明しました。 ■ 同時に表示する必要のあるキャンバスごとにウィンドウを作成する計画を立てます。 ■ レイアウトが計画どおりに Super VGA モードのモニターに表示されても、VGA のように解像度の異なるモードでは、レイアウトが画面からはみ出してしまう場合があります。対象となるすべてのユーザーのモニターでレイアウトをテストしてください。

表 2-2 キャンバス作成のための推奨事項

キャンバスのタイプ	推奨事項
コンテント・キャンバス	■ コンテント・キャンバスを「即時表示」に設定します。 ■ コンテント・キャンバスの表示サイズは、割り当てられたウィンドウの現行のサイズによって決まることに注意してください。 ■ キャンバス上のオブジェクトの斜体文字修飾の効果を最大にするために、キャンバスを白色以外のカラーにすることを検討してください。また、白色のバックグラウンドは明るすぎて疲れることがよくあります。 ■ ウィンドウごとに 1 つのコンテント・キャンバスを使用します。複数のコンテント・キャンバスを使用すると、ウィンドウ全体が切り替えられる理由をユーザーが理解していない場合に、混乱を生じます。やむをえず複数のコンテント・キャンバスを使用する場合には、すべてのコンテント・キャンバスを論理的に関係付けて、ユーザーが明示的にキャンバス間を移動できるようにする必要があります。複数のコンテント・キャンバスをインプリメントして成功するのは、ワード・プロセッサのアプリケーションでユーザーが同じ文書で印刷プレビュー、標準およびアウトラインなどの複数の表示を選択する場合です。
スタック・キャンバス	■ ボイラープレートを含むオブジェクトのグループを表示または非表示にするには、スタック・キャンバスを使用します。 ■ スタック・キャンバスのサイズは、必要な項目を入れるのに十分な大きさに設定してください。 ■ スタック・キャンバスをインプリメントする前に、スタック・キャンバスの動作方法を理解しておいてください。たとえば、スタック・キャンバスによってあいまいになったフィールドをナビゲートするために、ユーザーが「次のフィールド」または「次のレコードへ」を使用する場合、スタック・キャンバスがなくなったように見えます（つまり、自動的にコンテント・キャンバスの下に配置されます）。スタック・キャンバスを再表示させるには、ユーザーはスタック・キャンバス上の項目をナビゲートするか、またはスタック・キャンバスのナビゲートを止めて「Show Canvas」アクションを選択します。
タブ・キャンバス	■ タブの数は、4 ～ 6 に制限します。 ■ 1 つのオブジェクトに関する関連情報を編成するには、タブを使用します。たとえば、給与、諸手当、職種などの従業員情報は、タブ設定されたダイアログとして動作します。

2.2.2.5 ウィンドウの作成

次の表には、ウィンドウ作成のための推奨事項が示されています。

表 2-3 ウィンドウの推奨事項

属性	推奨事項
一般	■ ウィンドウの端で 3D の陰をつけた線を使用しないでください。 ■ 環境からカラー設定を継承します。 ■ ボタンおよびデータ整合のチェックボックス用の線を除いて、ウィンドウのブランクの一番上と一番下の線は残しておいてください。 ■ 領域の線およびブロックの境界線を除いて、左端と右端の文字セルの列はブランクのままにしておきます。 ■ モードレス（モーダルではない）ウィンドウを使用すると、スクロールが可能で、他のウィンドウとの間でのマウスのナビゲーションが可能です（標準オブジェクト・ライブラリ内の STD_DIALOG_WINDOW_MODELESS オブジェクトを使用します）。 ■ 他の場所でマウス・ナビゲーションができないようにするには、モーダル・ウィンドウを使用します。また、プロシージャの一部である依存作業用に使用します（標準オブジェクト・ライブラリ内の STD_DIALOG_WINDOW_MODAL オブジェクトを使用します）。
タイトル	■ 「ウィンドウ」メニュー内でアイコン化された名前とエントリが有効になるように、フォームの中の各ウィンドウに一意のタイトルを付けます。
位置	■ 最初にオープンされたときに、各ウィンドウの全体が確実に表示されるようにします。 ■ すべてのウィンドウを移動可能に設定します。 ■ フォームが終了したときのウィンドウの位置を保持します。
スクロールバー	■ デフォルトでスクロールが不要になるように、ウィンドウを設計します。スクロールは、ユーザーがウィンドウのサイズを変更した場合にのみ認められます。
ツールバー	■ ツールバーは、コンテナ・ウィンドウ（Windows）またはルート・ウィンドウ（他のすべてのプラットフォーム）にのみ配置します。 ■ マウスで上をなぞったときにのみ各ボタンの真下に表示される、ツールチップ・ヘルプ内のツールバー・ボタンにヒントを実装します (2.2.2.9.1 項「ツールチップのインプリメント」を参照)。

2.2.2.5.1 モードレス・ウィンドウのタイトルの選択

標準オブジェクト・ライブラリ内の STD_DIALOG_WINDOW_MODELESS オブジェクトでは、位置決め、クローズ、サイズ変更および配置に関するすべての問題が対処されますが、ウィンドウには独自のタイトルを選択しなければなりません。選択するには、次のようにします。

- ウィンドウで製品固有の作業を実行する場合は、「Transfer Items」、「Post Journals」および「AutoCreate Quotes」の場合と同じく、<*Verb*><*Noun*> という形式を使用します。
- ウィンドウ名は、ウィンドウがデータのシングル・インスタンスに関連している場合を除いて、複数作成します。つまり、「Item」ではなく「Items」を使います。
- < ウィンドウ名 > - < コンテキスト > のフォームで、子ウィンドウにコンテキストを付けます。コンテキストは、一番上のマスター・レコードにするか、または新規レコードの場合は [新規] にします。

  例 : 割当て（OR1）- [John Doe]

  製品発注書（ABC）- [新規]

2.2.2.6 領域の作成

次の表には、領域作成のための推奨事項が示されています。

表 2-4 領域の推奨事項

属性	推奨事項
一般	■ ユーザーにとって意義がある場合か、または画面を使いやすくする場合を除いて、グループ項目に領域を作成したり、ボイラープレートを追加しないでください。 ■ 枠の凹凸を使用して、領域ブロックを囲む線や四角形を作ります。 ■ ブロック全体を含む領域に対しては、枠を使用します。枠には、枠と項目の間の空白（マージン）、間隔および可視属性など、中に含まれる項目のレイアウトを制御するプロパティがあります。標準の枠を使用することで、作成しているウィンドウのレイアウトの一貫性を維持できます（レイアウト・ウィザードで枠を作成しても、オブジェクト・ライブラリ内に格納された枠を適用すればいつでも上書きできます）。

表 2-4 領域の推奨事項

属性	推奨事項
タイトル	■ 情報が書き込まれていなければ、領域にタイトルを追加できます。太字体を使用してください。 ■ 四角形または線の一番上にタイトルを位置付けます。タイトル・テキストの先行ブランクと後続ブランクはそのまま残しておきます。 ■ タイトルを表示するには、次のいずれかのウィジェットを使用します。 • ボイラープレート（静的な領域のタイトル用）。 • 枠タイトル（枠用）。 • 表示項目。外観はボイラープレートと類似しています（動的な領域のタイトル用）。 • ポップリスト（その他の領域用）。 • チェックボックス（領域全体が適用可能か、または適用不可かを示す場合）

2.2.2.7 ブロックへの項目の追加

次の表は、ある項目の他に別のフォームの項目を選択する場合に役立ちます。また、標準オブジェクト・ライブラリ内のオブジェクトまたは標準を変更する場合に、使用できるガイドラインを示します。項目は、アルファベット順に示します。

表 2-5 項目の推奨事項

項目	使用条件	推奨事項
ボイラープレート・テキスト	■ プロンプトでもタイトルでもないテキスト。	■ 大文字および小文字を使用します。 ■ イタリックと下線の多用を避けてください。 ■ フォント・スタイルは、一貫して使用してください。たとえば、強調するために太字を使用する場合、他の目的には使用しないようにします。 ■ フォント、サイズおよびカラーのバリエーションを使いすぎないようにしてください。

表 2-5 項目の推奨事項

項目	使用条件	推奨事項
ボタン（アイコンではない）	■ ダイアログの応答（モーダル・ウィンドウ）として、項目関連のアクションに使用します。	■ 標準オブジェクト・ライブラリ内の STD_BUTTON_*type* オブジェクトのいずれかを使用します。 ■ 1 つのウィンドウに 6 個まで使用します。できるだけ 1 つの行または 1 つの列に配置します。 ■ ボタンとボタンの間を 0.1 インチの空白をとって整列します。ボタンの論理グループの間隔を 0.5 インチに設定します。 ■ 一番右のボタンの右端とウィンドウの右端の間隔を 0.1 インチに設定します。 ■ たとえば、‘ Print Invoice ’のように、ラベル付けされた語句を大文字にします。 ■ ボタン・ラベルの終わりに省略符号（...）を使用するのは、ボタンによってモーダル・ウィンドウがオープンされるか、またはアクションが完了する前に、ユーザーが別のウィンドウ（モーダルまたはモードレス）でアクションに関する情報をさらに入力するよう求められた場合です。 ■ 「OK」ボタンと「取消し」ボタンを一緒に配置します。 ■ 最初に肯定と取消しを意味するボタンを押し、最後に独自のボタンを押します。 ■ ダイアログが拡張できることを示す場合には、山形（>>）を使用します。 ■ ラベル付けされたボタン（「OK」および「取消し」を除く）には、必ずアクセラレータ・キー（下線が引かれた文字）を用意します。 • ラベルの最初または 2 番目の語の最初の文字を使用します（「File」の場合は「F」、「Start Posting」では「P」）。それよりも強いリンクがある場合（「Exit」の「X」など）は、その文字を使用します。 • できるだけ、母音ではなく子音を使用します。 • ウィンドウ内のアクセス・キーは一意にします。その場合、メニューの最上位レベルで使用されるキーと競合しないようにしてください。
チェックボックス	■ チェックボックス上のラベルが「TRUE」（チェックあり）および「FALSE」（チェックなし）のいずれかの状態として想定できる場合にのみ使用します。それ以外の場合は、2 つの項目の付いたラジオ・ボタンを使用します。	■ 標準オブジェクト・ライブラリ内の STD_CHECKBOX オブジェクトを使用します。 ■ 肯定的な文のラベルを使用します。 悪い例 : 今後はこの警告が表示されません。 よい例 : 今後はこの警告が表示されます。

表 2-5 項目の推奨事項

項目	使用条件	推奨事項
表示項目	■ ユーザーが入力しない表示専用のフィールドに使用します。たとえば、財務系のアプリケーションの「合計」フィールドがこれに当たります。	■ 標準オブジェクト・ライブラリ内の STD_DISPLAY_ITEM オブジェクトを使用します。
アイコン	■ 頻繁に使用されるアクションまたは重要なアクションのみに使用します。 ■ 作業または実社会の擬似オブジェクトを表すわかりやすい絵を使用します。	■ ツールバーに頻繁に使用するボタンを配置します。 ■ 関連したツールをグループ化し、他のグループとの間に空白を設けて区分けします。 ■ 利用できないボタンを使用禁止にします。 ■ ユーザーがアイコンの意味を取り違えることが多いので、常にツールチップが利用できるように装備します。 ■ アイコンは文化によって意味が異なります。ときには、アイコンを使用するユーザーがわかるように、別のものに置き換えて訳す必要があることに注意してください。 ■ 「Run」を示すのに、走っている人の絵を使用するなど、視覚的な語呂合わせはしないでください。アイコンの意味がわかりにくくなる上に、他国の言語では意味が通じません。
リスト（ポップリストおよび T list も参照）	■ ユーザーにとって、値を入力するよりも選択した方が早い場合に使用します。 ■ 選択できるリスト形式でデータ・エントリおよびテキスト値の表示に使用します。 ■ 表示された値エントリが比較的短い（各エントリにつき 30 文字まで）場合に使用します。 ■ ユーザーが新規の値を入力する可能性がある場合には、コンボ・ボックス・スタイルを使用します。	■ エントリが 15 以下の場合は、ポップリストを使用します。エントリが 15 以上である場合、LOV（値リスト）を使用します。実際の資産が多数ある 30 以上のエントリには、T list を使用します。 ■ 関連したすべてのフィールドを同じ長さにします。 ■ 入力できないように設定されたテキスト項目には、キャンバスのバックグラウンドと同じカラーを使用します。
値リスト	■ ユーザーが 15 以上の行からなるリストから選択する、またはデータの複数の列を表示する必要がある場合に使用します。	■ 値が 1 つしかないときに、ユーザーに対して 1 行が自動的に選択されます。 ■ 選択後、カーソルが次のフィールドに自動的に移動するようにします。 ■ 値リストに 100 以上の行がある場合、選択する前に、ユーザーに対して有効な値のリストを減らすように求めます。

表 2-5 項目の推奨事項

項目	使用条件	推奨事項
ポップリスト	■ 適用できる値が 1 つしかなく、リストの選択肢が 15 以下である場合に使用します。	■ ポップリストをインプリメントする前に、頻繁に利用するユーザーがリストから選択するよりも、直接入力した方が速いかどうかを考えます。
ポップアップ・メニュー	■ メニュー・オプションを、アプリケーション全体ではなく項目に関連付ける場合に使用します。 ■ 頻繁に使用するコマンドへのアクセスを提供する場合に使用します。	■ 標準オブジェクト・ライブラリ内の STD_POPUP_MENU_ITEM オブジェクトを使用します。
プロンプト	■ フィールド、チェックボックス、リストなどのラベルとして使用します。	■ 記述する要素の一番上または左に位置付けます。 ■ 常に、シングル・レコード・ブロックのプロンプトはフィールドの左側に、マルチ・レコード・ブロックのプロンプトはフィールドの上側に位置付けます。 ■ ある範囲のフィールドを指定するには、「から」および「まで」を使い、「開始」および「終了」、または「低」および「高」は使用しないでください。 ■ パーセンテージを示す場合は、フィールドの後ろ側にパーセント記号（%）を付けます。プロンプトの中には、パーセント記号を入れないでください。
ラジオ・グループ	■ 相互に排他的な選択を示す場合に使用します。 ■ 表示される情報のタイプなど、「モード」を設定する場合に使用します。	■ 標準オブジェクト・ライブラリ内の STD_RADIO_GROUP オブジェクトを使用します。 ■ 横方向ではなく、縦方向に使用します。 ■ 関連するボタンをラジオ・グループに入れて、タイトルを付けます。 ■ 選択肢が二者択一（ON/OFF、YES/NO）の場合は、かわりにチェックボックスを使用します。 ■ 常にデフォルト値を表示します。
T List	■ 適用できる値が 1 つしかなく、リストの選択肢が 30 を超えない場合に使用します。	■ 常に、データの 5 行以上を表示します。 ■ 使用可能な実際の資産が多数あるフォームの中でのみ使用します。
テキスト項目	■ データ・エントリおよび文字値の表示用に使用します。 ■ 長い値、または実装されていない値（つまり、短い事前定義済みのリストにはない）に使用します。	■ 入力できないように設定されたテキスト項目には、キャンバスのバックグラウンドと同じカラーを使用します。 ■ ユーザーがフィールドに値を入力できる場合は、斜体を使用します。 ■ 標準オブジェクト・ライブラリ内の STD_TEXT_ITEM または STD_DATE_*type* オブジェクトのいずれかを使います。

2.2.2.8 メッセージの設計

メッセージは、ウィンドウのコンソール領域内または「警告」と呼ばれるポップアップ・ウィンドウ内に表示されます。メッセージの表示方法は、メッセージのタイプおよびユーザーから応答を求められるかどうかによって異なります。次に提案事項を説明します。

表 2-6 メッセージの表示

メッセージのタイプ	推奨事項	例
エラー	■ エラー・メッセージを表示するには、標準オブジェクト・ライブラリ内の「Alerts」タブの下にある STD_ALERT_STOP オブジェクトを使用します。 ■ エラーによって処理が中断する恐れがある場合に使用します。ダイアログ内に停止を示すアイコンを組み込みます。 ■ なるべく使用を控えてください。	「この命令を承認する権限がありません」

表 2-6 メッセージの表示

メッセージのタイプ	推奨事項	例
警告	■ 標準オブジェクト・ライブラリ内の「Alerts」タブの下にある STD_ALERT_CAUTION_1、STD_ALERT_CAUTION_2、または STD_ALERT_CAUTION_3 オブジェクトを使用します。メッセージ・テキストの説明を参照するには、ライブラリ内で該当するオブジェクトを 1 回クリックします。 ■ 処理を続行する前にユーザーの応答が必要な質問を提示するときに使用します。ダイアログで優先させる記号のアイコン（!）を組み込みます。 ■ 警告は短く簡潔にします。たとえば、「本当にこの命令を削除しますか ?」というメッセージではなく、「この命令を削除しますか ?」を使用します。 ■ 肯定的な表現で質問をしてください（「変更を保存しますか ?」を使用して、「本当に変更内容を保存したくありませんか ?」とはしないでください）。	「この請求書のすべての行をコピーしますか？」
情報	■ 標準オブジェクト・ライブラリ内の STD_ALERT_INFORMATION オブジェクトを使用します。 ■ 選択が必要な箇所で何も選択されてないときに、ユーザーの注意を喚起するメッセージを提示するために使用します。情報アイコン（円の中に「i」の文字が表示される）と「OK」ボタンを組み込みます。	「現時点で、製品数と出荷数が合いません。」 「応答待ちをしている項目があります。」
ヒント	■ コンソール内の Form Builder のメッセージ行に表示されます。 ■ かなり短いメッセージ、または応答する必要のないプロセス・インジケータを提示するために使用します。	「作業中 ...」 「最初のレコードで」 「処理命令 12 行 /37 行」

2.2.2.8.1 メッセージ・テキストの作成

できれば、エラー・メッセージも組み込みます。

- 行われた操作
- 原因（エラーになった理由）

- アクション（解決方法）

次に、メッセージ・テキストのよい例と悪い例を示します。

悪い例	よい例
日付が無効です。	「日付を DD-MON-YY として入力し直してください。」
終了日付よりも後の日付を開始日付に入力しないでください。	「開始日付は、終了日付より前の日付に設定してください。」
エラー : 1623、制約違反です。	「このフィールドに一意の値を入力し直してください。」
このメッセージは受信しないでください。	このメッセージは表示しないでください。
誤ってツールが紛失しています。	適切なメッセージに置き換えるか、またはこのメッセージ自体を削除します。

メッセージ・テキストを書くときには、これらのガイドラインを遵守してください。

推奨事項	例
能動態を使用します。	「これが行われる必要があります」という表現は避けて、「ただちにこれを行ってください」にしてください。
命令法を使用します。	「コミッション計画を入力できます」ではなく、「コミッション計画を入力してください」とします。
「することがあります」または「〜できるでしょう」のような類推表現ではなく、「〜できます」を使用します。	「印刷されたリリースは削除できないことがあります」ではなく、「印刷されたリリースは削除できません」とします。
できるだけ、実際のフィールド名に任せます。	フィールドのラベルが「販売組合員」である場合、「異なる販売員名を入力してください」というメッセージは使用しないでください。
コマンドおよびキーワードには、大文字を使用してください。	`ALTER CLUSTER` 文はサポートされていません。
ユーモアは使用しないでください。	しくじりました！
非難めいたメッセージを避けてください。ユーザー側に間違いがあったことを遠回しに言うような表現は使用しないようにします。問題の解決に関係がない場合には、ユーザーの間違いを指摘しないでください。	「値が選択されていません」ではなく、「値を選択してください」という表現を心がけてください。
指示が含まれたメッセージの場合は、「〜してください」などの丁寧語を使用します。	「値を選択しなさい」よりも、「値を選択してください」という表現を選びます。

推奨事項	例
ユーザーに対しては、「ユーザー」ではなく、「あなた」という呼び方をします。「わたし」、「彼」または「彼女」という呼称は使用しないでください。	「ユーザーは、モジュールをバックアップする必要があります」ではなく、「モジュールをバックアップしてください」という表現を心がけます。
エラーが発生したときに、状況判断ヘルプが利用できるように設定してください。	2.2.2.9.2 項「オンライン・ヘルプのインプリメント」を参照してください。

2.2.2.9 オンライン・ヘルプのインプリメント

この項では、2 種類のオンライン・ヘルプの使用方法を説明します。

ツールチップ　ポップアップ・ヒントは、またはミクロヘルプとも呼ばれています。ユーザーがマウスを動かして画面上の項目の上に移ったときに表示されます。

オンライン・ヘルプ　状況判断ヘルプと、ユーザーが関連項目にジャンプできるハイパーテキスト・リンクが含まれます。

2.2.2.9.1 ツールチップのインプリメント

項目には、「ツールチップ」と呼ばれるプロパティと「ツールチップ可視属性グループ」と呼ばれるものがあります。プロパティ・パレットの「ツールチップ」プロパティ・フィールドには、ポップアップで表示したいテキストを入力します。アプリケーション全体に一貫性を持たせるには、標準オブジェクト・ライブラリから STD_TOOLTIP 可視属性を適用します。可視属性を適用しない場合は、ツールチップにはプラットフォーム固有のデフォルトが使用されます。

2.2.2.9.2 オンライン・ヘルプのインプリメント

- ヘルプ・オーサリング・ツールを使用することを検討してください。
- 他のトピックにハイパーリンクされたスタンドアロン・トピックを作成します。
- テキストは短く簡潔にしてください。
- 各ダイアログに「ヘルプ」ボタン、メイン・メニューには「ヘルプ」オプション、ツールバーには「ヘルプ」ボタンをそれぞれ装備します。

2.2.2.10 効果的なメニューの作成

Form Builder では、各フォームにデフォルト・メニューが用意されています。デフォルト・メニューには、問合せ、挿入および削除を含むすべての基本的なデータベース操作のコマンドが含まれています。アプリケーション固有の要件がデフォルト・メニューと合致しない場合、カスタム・メニューを手軽に作成できます（詳細は、Form Builder のオンライン・ヘルプの「メニューの作成」を参照してください）。メニューを作成するときは、次のことに注意してください。

- コマンドが属している作業に従って、コマンドを編成します。
- Forms Developer ではスクロールできるメニューバーがサポートされていますが、ユーザーに見えるように、画面上に有効な項目をすべて表示するよう注意してください。
- 利用できない項目は使用禁止にします。
- サブメニューをできるだけ 2 つのレベルに制限します。
- ラベルを大文字にします。

2.3 効果的なレポートの作成

Report Builder を使用して効果的なレポートを作成する場合、最初のステップは、効果的なフォームまたは図表を設計する場合と同じです。この項の残りの部分に進む前に、まだお読みでなければ、2.1.2 項「ユーザー要件の定義」をお読みください。

次に、作成するレポートのためのユーザー要件を決定する場合に役立つ問題点をいくつか示します。

- ユーザーがレポートでどんなデータを参照したいですか、参照するデータの優先順位をどうしますか？
- ユーザーが詳細を参照するためにデータで「ドリルダウン」することを希望していますか？希望する場合には、該当するボタンをレポートに組み込みます。各ボタンには対応した PL/SQL コードのブロックがあり、2 次的なレポートを起動し、ビデオや音声などを出力できます。
- ユーザーが、レポートでデータをチャートの形で表示することを希望していますか？希望する場合、どんな種類のデータですか？Graphics Builder を使用してチャートを作成すれば、（2 次的な問合せを実行しなくても）Report Builder からチャートにデータを渡すことができます。
- ユーザーがフォーム・アプリケーションを使用してデータを変更する場合、後にデータを印刷することを希望していますか？希望する場合は、フォームからレポートをコールし、データをレポートに渡します。
- ユーザーがフォームへのレポートの埋込みを希望していますか？希望する場合は、フォントおよびカラーの標準がフォームと同じテンプレートを設計します。
- ユーザーがレポートを HTML、PDF またはハードコピーで表示することを希望していますか？（任意で）印刷する前に、ライブ・プレビューアでフォーマットを変更する必要がなるべくないようにしますか？希望する場合は、レポートの宛先パラメータ （DESTYPE）を指定するか、またはユーザーが同様の操作をできるようにする必要があります。
- 「ユーザー ID SCOTT に対して、売り上げランキングの上位 10 位までを表示」のように、ユーザーがレポートのパラメータを指定しますか？指定する場合は、ユーザー・パラメータを作成し、フォームまたは「ランタイム・パラメータ・フォーム」内でユーザーに値を指定させるようにする必要があります。

- ネットワーク・トラフィックおよびマシンのパフォーマンスに基づいて、レポートをクライアント / サーバーまたは 3 層アーキテクチャで実行しますか？
- レポートを波及させたい会社の標準がありますか？ある場合は、標準のテンプレートを定義する必要があります。
- Web レポートの場合、いくつかのレポートを静的にしますか、または Web サイトを動的に生成しますか？

2.3.1 レポートの理解

レポートの特定の検討事項を記述する前に、いくつかの基本的なレポートの概念を簡単に紹介します（Report Builder を使った事のある場合は、このセクションをとばしてください）。これらと他のレポートに関連したトピックの詳細は、Report Builder のオンライン・ヘルプまたは Reports Developer クイック・ツアーの「Report Builder」の項、あるいはその両方を参照してください。

レポートを作成するときは、次の 2 つのアプリケーション・コンポーネントを使用して作業を行います。

モジュールのタイプ	説明
レポート・モジュール	オブジェクトおよびコード・ルーチンの収集。レポート・モジュールに定義できるオブジェクトには、繰返し枠、枠、フィールド、ボイラープレート、アンカーおよびトリガーと呼ばれる PL/SQL コードのブロックがあります。
PL/SQL ライブラリ・モジュール	ユーザー命名プロシージャ、ファンクションおよびアプリケーション内の他のモジュールからコールできるパッケージの収集。

この項では、PL/SQL ライブラリの使用方法は扱っていません。このトピックの詳細は、Report Builder のオンライン・ヘルプを参照してください。

レポートによっては、レポート・ウィザード（レポートのタイプを選択し、データ・モデルおよびデータのレイアウトを定義する）や、レポート・エディタのライブ・プレビューア（レポートの詳細をチューニングする）を使用します。他のレポートでは、レポート・エディタの他のビューを使用します。

表示	使用目的
データ・モデル・ビュー	2 つ以上の問合せを使用してレポートを作成する。
レイアウト・モデル・ビュー	■ 複数のセクションを使用してレポートを作成する（たとえば、表およびマトリックスのスタイルを使った 1 つのレポート）。 ■ 新規のレイアウト・オブジェクトを追加する（たとえば、ボタンなど）。 ■ オブジェクトのサイズ設定または位置付けの方法を制御する。

表示	使用目的
パラメータ・フォーム・ビュー	ユーザーがレポートを実行する前にパラメータの値を指定できるダイアログを提示する。

この章では、視覚的効果の高いアプリケーションを作成する方法を説明するため、この項の残りの部分では、レイアウト・モデル・ビューで見られるテンプレート、オブジェクトおよび設定の使用方法を中心に説明します。

2.3.2 Report Builder でのテンプレートの使用

レポートの開発で使用できる最も重要な標準化の手段は、「テンプレート」でしょう。テンプレートは、レポート全体の外観を決めるボイラープレートおよびレイアウト設定の集合です。Report Builder にはいくつかのテンプレートが実装されていますが、独自のテンプレートを作成することもできます。会社またはグループのテンプレートを作成して、開発チーム全体でそれらを利用すれば、共通のルック & フィールを実現できます。テンプレートの作成方法の手順は、Report Builder のオンライン・ヘルプを参照してください。

2.3.3 レイアウト・オブジェクトの理解

レポート・エディタのレイアウト・ビューには、次のオブジェクトが含まれる場合があります。

オブジェクト	説明
枠	繰返し枠、フィールド、ボイラープレート、ボタンおよび子枠を制御するコンテナ。Form Builder の枠とは違って、Report Builder の枠には、子オブジェクトの位置を制御するフォーマットのプロパティがありません。
繰返し枠	次のものを制御するコンテナ。 ■ レポート・データを入れるフィールド ■ 繰返し枠に所有される他のオブジェクト
フィールド	レポートのデータ、日付、ページ番号などを表示するコンテナ。
ボイラープレート	オブジェクトを囲い込んでいるもの（レポート、枠または繰返し枠）、またはオブジェクトに連結されているものから要求されるたびに表示されるテキストまたはグラフィックス。
アンカー	レポート・レイアウト内の 2 つのオブジェクトを互いに関連付ける方法を判断するオブジェクト（つまり、親 / 子の関連および相対的な位置付けをされているオブジェクト）。
ボタン	ユーザーがクリックするたびにアクションが実行されるオブジェクト。

アンカーを除き、レイアウト・オブジェクトには、オブジェクトがアクティブにされるたびに起動する PL/SQL ブロックのようなフォーマット・トリガーを持つ場合があります。

2.3.4 Report Builder でのレイアウト・オブジェクトの制御

レポートを設計するときには、レポート全体のサイズとその中に含まれる個々のオブジェクトのサイズがそれぞれ異なるために、印刷されたレポートのページ区切りに影響が出る場合があるので注意してください。次の問合せに基づいて作成されたレポートを検討します。

```
select ename, sal from emp
where sal > 2000
```

このレポートのサイズは、複数の要素を基にして設定されています。

- 問合せの条件を満たすデータの量。少数のレコードから数百、数千の範囲に及ぶことがあります。
- レポートを実行するとき。たとえば、昇給後の週に動的に変更された可能性のあるレコード数の配分（たとえば、給与が 2000 ドルを超えている人の数が増える可能性があります）。
- レポートからデータまたはオブジェクトを除外するグループ・フィルタまたはフォーマット・トリガーを使用するかどうか。

同じオブジェクトのインスタンスでも、サイズが異なる場合があります。たとえば、COMMENTS というデータベース内に VARCHAR2 列があるとします。COMMENTS では、1 つのレコードに 2 つの文が含まれる場合があります。あるいは、10 個の文が含まれることもあるでしょう。COMMENTS 列に対応するレイアウト内のフィールドは、長さの異なる値を入れられるようにサイズ設定する必要があります。また、そのフィールドの周りにあるオブジェクトは、上書きされたり、レポート内に大きなギャップが生じないように、「プッシュ」または「プル」される必要があるでしょう。

Report Builder では、レポート・エディタのレイアウト・ビューのさまざまなメカニズムによって、オブジェクトのサイズ設定および位置付けの方法を制御できます。これらのメカニズムの詳細は、次の項で説明します。

- 2.3.4.1 項「アンカーの使用」
- 2.3.4.2 項「「印刷オブジェクト・オン」プロパティおよび「基本印刷オン」プロパティの使用」
- 2.3.4.3 項「水平拡張度および垂直拡張度の理解」
- 2.3.4.4 項「「前で改ページ」プロパティおよび「後で改ページ」の使用」
- 2.3.4.5 項「「ページ保護」プロパティの使用」
- 2.3.4.6 項「「アンカー・オブジェクトと連動」プロパティの使用」

2.3.4.1 アンカーの使用

アンカーは、レポート・レイアウト内のオブジェクトを相互に関連付ける方法です。2 つのオブジェクトをアンカーすると、一方のオブジェクトが親、他方のオブジェクトが子と見なされます。オブジェクト間で親子の関連を定義すると、アンカーによってレポート内のオブジェクトの階層が確立されます。Report Builder では、このようなオブジェクトの階層に基づいて、相互に関連したオブジェクトの印刷方法、2 つのオブジェクトを同じページに収めるかどうか、周囲のオブジェクトのサイズによってオブジェクトをプッシュまたはプルする方法が決定されます。

アンカーは、次の 2 通りの方法で作成できます。

- Report Builder で自動的に作成。これは、暗黙のアンカーと呼ばれます。ほとんどの場合、暗黙のアンカーのみです。
- レポート・エディタのレイアウト・ビューのアンカー・ツールを使用して、独自に作成。これは、明示的なアンカーと呼ばれます。明示的なアンカーは、何らかの理由で暗黙のアンカーを上書きする必要がある場合にのみ使用します。Report Builder のオンライン・ヘルプの「アンカーについて」を参照してください。

2.3.4.2 「印刷オブジェクト・オン」プロパティおよび「基本印刷オン」プロパティの使用

「印刷オブジェクト・オン」プロパティでは、レポートに表示されるオブジェクトの頻度が決定されます。「基本印刷オン」プロパティでは、「印刷オブジェクト・オン」プロパティを基盤とするオブジェクトを指定します。

たとえば、全ページの「印刷オブジェクト・オン」および「アンカー・オブジェクト」の「基本印刷オン」を指定すると、オブジェクトは、アンカーしているオブジェクト（親オブジェクト）が表示されるすべての論理ページに印刷されるようにトリガーされます。レポート・ウィザードによって作成されたオブジェクトには、それぞれ専用に設定されたプロパティがあります。ほとんどの場合、Report Builder で選択される値は、そのオブジェクトに最適の値です。これらのプロパティを独自に設定する必要があるのは、Report Builder で設定されたデフォルト値を上書きしたい場合のみです。

「印刷オブジェクト・オン」プロパティを適用すると、Report Builder では、オブジェクトの一部が印刷されている最初の論理ページがオブジェクトの第 1 ページとなるように配慮されます。同様に、最終ページは、オブジェクトの一部が印刷されている最後の論理ページになるように配慮されます。たとえば、「第 1 ページ」の「印刷オブジェクト・オン」と、「インクローズ・オブジェクト」の「基本印刷オン」を指定すると、その囲みオブジェクトが表示される最初の論理ページにオブジェクトが印刷されるようにトリガーされます。

注意 :

- オブジェクトが繰返し枠内にある場合、「基本印刷オン」によって繰返し枠の各インスタンスが参照されます。オブジェクトが繰返し枠の外側にあり、明示的に繰返し枠にアンカーされている場合は、「基本印刷オン」によって繰返し枠全体が参照されます。

- オブジェクトが論理ページに印刷されるようにトリガーされていても、論理ページにそのオブジェクトが印刷されるとは限りません。他の設定（たとえば、「前で改ページ」）や、ページで使用可能なスペースの量によって、Report Builder では、最初に印刷するようにトリガーされたページ以外のページにオブジェクトが印刷される場合があります。

詳細は、Report Builder のオンライン・ヘルプで、索引項目の「水平拡張度」および「垂直拡張度」を参照してください。

2.3.4.3 水平拡張度および垂直拡張度の理解

水平拡張度および垂直拡張度の各プロパティでは、複数のオブジェクトまたはデータを収めるオブジェクトの水平サイズと垂直サイズを実行時に変更する方法を指定します。

- 枠および繰返し枠の場合、中に含まれるオブジェクトによって枠または繰返し枠のサイズを変えるかどうかを拡張度によって定義します。
- テキストを含むオブジェクトの場合、中に含まれたテキストのサイズによってフィールドまたはボイラープレートのサイズを変えるかどうかを拡張度によって定義します。サイズが固定しているテキストは、サイズが定義されたオブジェクト内でラップされ、空間的な余裕がない場合は切り捨てられます。数値または日付データでは、データが定義されたサイズ内に収まらない場合は、アスタリスクとして表示されます。
- イメージ、描画およびチャート・オブジェクトの場合、Report Builder では、比例したスケール変更が行われます。イメージ、描画およびチャート・オブジェクトの拡張度オプションによってスケール変更係数が決まります。

レポート・ウィザードによって作成されたオブジェクトには、それぞれ専用に設定されたプロパティがあります。ほとんどの場合、Report Builder で選択される値は、そのオブジェクトに最適の値です。これらのプロパティを独自に設定する必要があるのは、Report Builder で設定されたデフォルト値を上書きしたい場合のみです。

拡張度の設定を変えると、出力が予期しない結果になることがあります。たとえば、水平拡張度を可変として設定したオブジェクトを縮小すると、右揃えされたすべてのオブジェクトは、可変オブジェクトに暗黙にアンカーされているため、左側に移動してしまいます。

詳細は、Report Builder のオンライン・ヘルプで、「水平拡張度」および「垂直拡張度」を参照してください。

2.3.4.4 「前で改ページ」プロパティおよび「後で改ページ」の使用

ワード・プロセッサで処理された文書とは違って、レポートとそれに含まれるオブジェクトは、実行時のサイズおよび位置が異なります。結果として、あるレポートの改ページは、予想しにくくなります。

- 最初に印刷するようにトリガーされたページの**後**のページにオブジェクトをフォーマットしたい場合には、「前で改ページ」プロパティを使用します。ただし、「前で改ページ」に設定されたオブジェクトの下のすべてのオブジェクトが次のページに移動するとは限らないので注意してください。下にあるオブジェクトのいずれかが「前で改ペー

ジ」に設定されておらず、ページ内に収まる場合は、「前で改ページ」に設定されているオブジェクトの上に印刷されることがあります。

- オブジェクトのすべての子を次のページに移動させたい場合には、「後で改ページ」プロパティを使用します。言い換えると、このオブジェクトに（暗黙または明示的に）アンカーされているあらゆる子オブジェクトは、「前で改ページ」に設定されているものと見なされます。ただし、「後で改ページ」に設定されたオブジェクトの下のすべてのオブジェクトが次のページに移動するとは限らないので注意してください。下にあるオブジェクトのいずれかが「後で改ページ」に設定されておらず、他のオブジェクトの子でもない場合は、「後で改ページ」に設定されているオブジェクトの上に印刷されることがあります。

詳細は、Report Builder のオンライン・ヘルプで、索引項目の「前で改ページ」および「後で改ページ」を参照してください。

2.3.4.5 「ページ保護」プロパティの使用

オブジェクト全体とその内容を同じ論理ページに収めるには、「ページ保護」プロパティを使用します。オブジェクトの内容がページに収まらない場合は、次の論理ページに移されます。ただし、「ページ保護」に設定されたオブジェクトの下のすべてのオブジェクトが次のページに移動するとは限らないので注意してください。下にあるオブジェクトのいずれかがページに収まる場合は、「ページ保護」に設定されているオブジェクトの上に印刷されることがあります。

詳細は、Report Builder のオンライン・ヘルプで、索引項目の「ページ保護」を参照してください。

2.3.4.6 「アンカー・オブジェクトと連動」プロパティの使用

あるオブジェクトと同じ論理ページにアンカーされているオブジェクトを連動させるには、「アンカー・オブジェクトと連動」プロパティを使用します。オブジェクトまたはそれがアンカーしているオブジェクト、あるいはその両方が同じ論理ページに収まらない場合は、次の論理ページに移されます。

繰返し枠に対して「アンカー・オブジェクトと連動」を設定する場合は、繰返し枠の最初のインスタンスがアンカーされているオブジェクトと同じ論理ページに収められなければなりません。そうでないと、「アンカー・オブジェクトと連動」の条件を満たせません。繰返し枠以外の任意のレイアウト・オブジェクトに対して「アンカー・オブジェクトと連動」を「はい」に設定すると、アンカーしているオブジェクトと同じページにオブジェクト全体をフォーマットできます。

2 つのオブジェクト間のアンカーは、明示的または暗黙のいずれかになります。したがって、「アンカー・オブジェクトと連動」を設定すると、2 つのオブジェクト間にアンカーを明示的に作成した場合と同じ効果が得られます。

2.4 効果的な図表の作成

Graphics Builder を使用すると、フォームおよびレポートの両方に組み込む図表を作成できます。図表は、それ自体で 1 つのアプリケーションとすることもできますが、フォームやレポートに組み込むこともできます。

次の場合に、図表を使用できます。

- 異なるカテゴリ間の関連を表示する場合（たとえば、テニス・シューズの販売数を婦人靴の販売数と比較する場合）
- 特定の値ではなく、傾向を示す場合
- ユーザーと、図形の領域および形状（マップおよびイメージのセクターなど）との対話を可能にする場合。Graphics Builder では、レイアウト・エディタで作成した形状をマウスを使用して加工できます。不規則で透明なボタンは、ユーザーが効果的に絵を選択できるようにするため、ダイアグラムまたはビットマップ・イメージの領域の上に配置できます。

グラフィックスを作成するときには、次のガイドラインに従ってください。

- 簡潔に作成します。図表に入れる線、バー、スライスなどが多すぎると、ユーザーが圧倒されて、作成したグラフやチャートが使用されなくなります。大量のデータがある場合は、最上位レベルでデータを要約し、ドリルダウンを使用して、より詳細な情報を表示するようにします。あるいは、複雑なグラフをもっと小さい個々のグラフに分けてから、ユーザーが表示したいグラフを選択してフォームを作成できるように検討します。
- 3-D 効果は、リソースを集約できるなど、情報をやり取りするのに便利な場合にのみ使用します。
- 作成したグラフィックスを利用環境内のすべての表示デバイスでテストし、解像度の最も低いモニターでも問題なく実行されることを確認します。
- 移行を示すにはカラーを、違いを示すには主要なカラーを使用します。カラーの使用の詳細は、2.2.2.3 項「カラーの追加」を参照してください。
- 複雑なグラフには、凡例を使用します。
- フォームからグラフィックス・モジュールにマウス・イベントを渡せることに注意してください。たとえば、フォームのチャート項目に「When-Mouse-Click」トリガーを作成し、このトリガーから OG.MouseDown プロシージャをコールして、マウス情報を図表に渡せます。その後、図表からフォームへ、ユーザーがクリックした図表のセクターの詳細を含む情報が戻されます。

2.4.1 望ましいグラフの選択

次の表は、最もよく使用される図表をインプリメントするためのガイドラインです。

表 2-7 図表の推奨事項

図表のタイプ	使用条件	推奨事項
棒グラフ	個別のオブジェクトと関連した値との間の関連を示す場合。	バーは 20 ～ 25 本に制限します。
円グラフ	部分と全体の関連を示す場合。通常、パーセンテージの値を示すために使用されます。	スライスは 10 個までに制限します。
折れ線グラフ	連続したデータの累積効果を示す場合。	線は 6 ～ 8 種類に制限します。
ダブル -Y 軸グラフ	広い範囲に分布する値の中でデータを比較する場合。	プロットは 4 以下に制限します。
ガント・チャート	期間データをスケジュールする場合。	バーは 40 ～ 50 本に制限します。
株価チャート	毎日の平均値、株価市場の値、高値、安値および現行値を追跡するデータを表示する場合。	行は 30 行以下に制限します。
複合グラフ	実際の値（バー）と予測値（折れ線）を比較する場合。	行は 30 行以下に制限します。
散布図	X 軸と Y 軸上の数値データの関連を示す場合。	行は、1 インチ当り 50 行未満に制限します。

3

パフォーマンスの提案

この章では、作成したアプリケーションのパフォーマンスを改善するための提案について説明します。この章は、次の項から構成されます。

3.1 概要

次の表は、利用可能なパフォーマンスの提案の概要と、この章のどこで詳細な説明がなされているかを示したものです。

詳細へ進む前に、3.2 項の紹介情報を読んでください。3.3 項のパフォーマンス測定に関する資料も参考になります。

提案は、内容に従って次の順序でグループ化されています。

1. すべての環境の Forms Developer および Reports Developer アプリケーションに適用される提案
2. すべての環境の、特定の Builder アプリケーション（Forms、Reports または Graphics）のための提案
3. クライアント / サーバー（2 層）環境における Forms Developer および Reports Developer アプリケーションのための提案

4. 3 層環境における Forms Developer および Reports Developer アプリケーションのための提案

パフォーマンスの提案	説明
すべてのアプリケーション :	
ソフトウェアのアップグレード	3-9 ページ
ハードウェアのアップグレード	3-10 ページ
配列処理の使用	3-10 ページ
冗長な問合せの削除	3-10 ページ
データ・モデルの改善	3-11 ページ
適切なオプティマイザの選択	3-12 ページ
問合せ SQL 内での計算の実行	3-12 ページ
明示的なカーソルの回避	3-13 ページ
グループ・フィルタの使用	3-13 ページ
Developer コンポーネント間での作業の共有	3-13 ページ
待ち時間を先に移動	3-13 ページ
Forms アプリケーションの場合 : （一般的な提案はすべて Forms にも当てはまります）	
配列処理のチューニング	3-14 ページ
ストアド・プロシージャに基づくデータ・ブロック	3-14 ページ
トランザクションの SQL 処理の最適化	3-16 ページ
トリガーの SQL 処理の最適化	3-16 ページ
フォーム間ナビゲーションの制御	3-17 ページ
レコード・グループのフェッチ・サイズの増加	3-17 ページ
LONG のかわりに LOB を使用	3-17 ページ
グローバル変数を削除	3-17 ページ
Microsoft Windows でのウィジェットの作成を削減	3-17 ページ
ロックの必要性を検証	3-18 ページ

パフォーマンスの提案	説明
Reports アプリケーションの場合 : （一般的な提案はすべて Reports にも当てはまります）	
レイアウトのオーバーヘッドの削減	3-18 ページ
フォーマット・トリガーを慎重に使用	3-19 ページ
表のリンクを考慮	3-20 ページ
ランタイム・パラメータの設定を制御	3-20 ページ
デバッグ・モードをオフにする	3-20 ページ
透明なオブジェクトの使用	3-21 ページ
非グラフィック・オブジェクトに固定サイズを使用	3-21 ページ
グラフィック・オブジェクトに可変サイズを使用	3-21 ページ
イメージの解像度の削減を使用	3-21 ページ
ワード・ラップの回避	3-21 ページ
フォーマット属性の簡素化	3-22 ページ
ブレーク・グループの使用を制限	3-22 ページ
Graphics Builder との作業の重複を回避	3-22 ページ
PL/SQL とユーザー・イグジットのいずれかを選択	3-23 ページ
DML に SRW.DO_SQL ではなく PL/SQL を使用	3-23 ページ
ローカル PL/SQL の使用を評価	3-24 ページ
SRW.SET_ATTR をコールする際に複数の属性を使用	3-24 ページ
ARRAYSIZE パラメータを調整	3-24 ページ
LONGCHUNK パラメータを調整	3-24 ページ
COPIES パラメータを調整	3-24 ページ
プレビュー時の事前フェッチを回避	3-25 ページ
適切なドキュメント・ストレージの選択	3-25 ページ
ファイル検索のパス変数を指定	3-26 ページ
複数層サーバーを使用	3-26 ページ

パフォーマンスの提案	説明
Graphics アプリケーションの場合 : （一般的な提案はすべて Graphics にも当てはまります）	
グラフィック・ファイルの事前ロード	3-26 ページ
必要な場合のみ図表を更新	3-27 ページ
図表の更新をループから移動	3-27 ページ
できるだけ共通要素を使用	3-27 ページ
DDL 文への DO_SQL プロシージャを制限	3-27 ページ
オブジェクトの参照にハンドルを使用	3-27 ページ
ショートカット・ビルトインを使用しないことを検討	3-27 ページ
クライアント / サーバー（2 層）環境のアプリケーションの場合 : （一般的な提案はすべてクライアント / サーバーにも当てはまります）	
最適なインストール構成を選択	3-28 ページ
最適なアプリケーションの場所を選択	3-28 ページ
3 層環境のアプリケーションの場合 : （ここまでの一般的な提案はすべて 3 層環境にも当てはまります）	
第 1 層と第 2 層のスケーラビリティの最大化	3-29 ページ
ランタイム・ユーザー・インタフェースへの変更を最小化	3-29 ページ
スタック・キャンバスを調整	3-30 ページ
上位レベルで妥当性チェックを実行	3-30 ページ
メニュー項目の使用可および使用不可を回避	3-30 ページ
画面サイズを小さくする	3-30 ページ
グラフィック URL へのパスを識別	3-30 ページ
マルチメディアの使用を制限	3-31 ページ
アプリケーション・サーバーから起動されるアニメーションの使用を回避	3-31 ページ

3.2 紹介 : パフォーマンス

パフォーマンスを改善するための設定をする前に、特定の目的とその達成に関わることについて明確な見解があれば役立ちます。

改善する領域、およびその領域のパフォーマンスを識別または測定する方法を正確に知る必要があります。

さらに、パフォーマンスを改善するには、今日のコンピューティング環境における多くの相互関係や依存性、関連コスト、およびある領域のパフォーマンスを改善する上で生じるトレードオフについて理解する必要があります。

3.2.1 いつのパフォーマンスか

Forms Developer および Reports Developer は、設計時および実行時に使用します。設計時は、プログラマがアプリケーションを作成するときであって、パフォーマンスの点では通常重要ではありません。ランタイムは、日々のビジネス環境で複数のエンド・ユーザーがアプリケーションを実行する時のことであって、常に最も重要になります。したがって、この章ではこれ以降ランタイム環境におけるパフォーマンスについて説明します。

3.2.2 何のパフォーマンスか

アプリケーションのパフォーマンスの見方はいろいろあります。アプリケーションのストレージ要件、コーディング効率、ネットワーク負荷およびサーバー利用などが領域の一部です。どの状況も異なっており、各サイトおよび部署には独自の優先順位があり、どのパフォーマンス領域が最も重要であるかについての独自の見解があります。さらに、これらの領域における「良い」パフォーマンスおよび「悪い」パフォーマンスは相対的なものです。絶対的な基準はありません。

通常、最も認識しやすい領域はエンド・ユーザーへの応答時間（つまり、アプリケーションを使用している人が選択や入力を行った後で待つ時間）に関するパフォーマンスです。この

場合も、絶対的な基準はありません。実際の応答時間がどうであれ、ユーザーはどのような物に慣れているか、何を期待しているか、に主に依存する見解を持っています。実際の数値は非現実的な値です。エンド・ユーザーが応答時間に不満である場合、この領域は確実に改善の対象となります。

3.2.3 相互関係

アプリケーションはそれだけでは実行できません。クライアント / サーバー環境において、アプリケーションは基本となるハードウェアとオペレーティング・システムの 2 つと、ハードウェアとソフトウェアのネットワーク通信に依存しています。3 層環境では、状況はより複雑になります。さらに、アプリケーションは 1 つ以上のデータベース・サーバーと対話し、実行時には他のソフトウェア・コンポーネントをコールすることもあります。

また、アプリケーションは多くの場合これらのハードウェアおよびソフトウェア・リソースを他のアプリケーションと共有しています。この共有のため、単体では効率的なアプリケーションが、他の非効率的なアプリケーションによって悪影響を受ける可能性があります。

したがって、アプリケーションのパフォーマンスは単なるそれ自体の設計や利用の結果ではなく、多くの異なるコンポーネントや要素の対話を組み合せた非常に複雑な結果です。

3.2.4 トレードオフ

パフォーマンスの改善の中には直接的な効果があるものがあります。たとえば、アプリケーションの無駄なコードを削除する場合などです。

ただし、他の改善はそれほど明快ではありません。たとえば、あるアプリケーションのネットワーク優先順位を必要性によって高くすると、他の優先順位が相対的に下がります。また別の例として、あるタイプのアプリケーションへのアクセス時間を改善するためにデータベースを再構築した場合、実際には他の重要なアプリケーションへのアクセス時間に影響を与えていることがあります。

1 つのアプリケーションのレベルでの典型的なトレードオフは、領域と速度です。いくつかのコンポーネントをロード対象からはずすことで、主なストレージ要件を減らすことはできますが、これは（ロード対象からはずれたコンポーネントを必要なときにロードしなければならないため）アプリケーションの応答時間に影響を与える可能性があります。一方、ロード操作を起動段階へ移動することで前述の応答時間は改善されますが、最初の起動時のオーバーヘッドの増加が犠牲になります。

特定の改善作業を決定する前に、密接な関係をより明確に理解し、優先順位に従って選択をすると役に立ちます。

3.3 パフォーマンスの測定

アプリケーションのパフォーマンスが適切かどうかの見分け方

応答時間の場合、エンド・ユーザーの意見が最も重要です。他の領域については、より具体的なデータ、場合によっては非常に多くのデータが必要です。

3.3.1 Forms Developer および Reports Developer 固有の測定方法

ビルトイン・パッケージの Ora_Prof は、Forms Developer および Reports Developer の両方に付属しています。このツールを使用すると、アプリケーションの PL/SQL を検証し、コードのある一部分を実行するのに必要な時間を調べることができます。

次の製品固有の測定ツールも利用できます。

3.3.1.1 Forms の測定

ランタイム・オプション STATISTICS=YES を設定することで、Forms アプリケーションの一般情報を得ることができます。

3.3.1.1.1 PECS

Form Builder の Performance Event Collection Services（PECS）を使用して、アプリケーションのランタイム動作についての詳細情報を収集することができます。

ランタイム・オプション PECS=ON を指定して、PECS データ収集をアクティブ化します。

PECS の最も簡単な用途は、アプリケーション全体の統計を集めることです。これには（ランタイム・オプションで PECS をアクティブ化するのみで）既存のアプリケーションへの変更は必要ありません。

PECS はアプリケーションの特定の領域に絞ることもできます。PECS は、コードに挿入して詳細に検証するセクション、イベントまたはクラスを識別できる、多数のビルトインを提供します。

データの収集が完了すると、PECS Assistant を使用して参照し分析することができます。Assistant はさまざまなタイプのレポートを生成し、経過時間、CPU 時間、イベントまたはオカレンス、PL/SQL コードの利用状況などが参照できます。アプリケーションのランタイム動作を分析することで、改善の可能性がある領域を見つけることができます。たとえば、コードの一部が他と比べ実行時間が非常に長い場合、より詳しい調査の対象となります。

3.3.1.2 Reports の測定

Report Builder には次の 2 つの測定ツールがあります。Reports プロファイル・オプションと Reports トレース・オプションです。

3.3.1.2.1 Reports プロファイル

Reports プロファイル・オプションを設定すると、レポートがどこにどれだけの処理時間を費やしたかを示すログ・ファイルを生成します。これはパフォーマンスのボトルネックの識別に役立ちます。

プロファイル・オプションを設定するには、PROFILE=<filename> を指定します。ここで、< ファイル名 > は必要なログ・ファイルの名前です。プロファイルはレポート・パラメータまたはコマンド・ライン引数のどちらにもなります。

サンプルのレポートからの典型的なプロファイル出力を次に示します。

合計経過時間 :	29.00 秒
Reports 時間 :	24.00 秒（全体の 82.75%）
Oracle 時間 :	5.00 秒（全体の 17.24%）
UPI:	1.00 秒
SQL:	4.00 秒

このプロファイルから、レポートの実行時間（合計経過時間）、フェッチしたデータをフォーマットするのに費やした時間（Reports 時間）、およびデータの取出しを待つのに費やした時間（Oracle 時間）を見ることができます。UPI 時間は、データベース接続および SQL の解析と実行に費やす時間のことです。SQL 時間は、データベース・サーバーがデータのフェッチに費やした時間と、SRW.DO_SQL() 文（アプリケーションに含まれる DML 文と DDL 文）の実行に費やした時間のことです。

この例でプロファイルは、ほとんどの時間が問合せやフェッチではなくデータのレイアウトに費やされていることを示しています。

3.3.1.2.2 Reports トレース

Reports トレース・オプションは、レポートの実行時にレポートが行う一連のステップを示すファイルを生成します。トレース・オプションは、ファイルにすべてのイベントのログを取るか、またはステップのサブセットのみをログに残すように設定することができます。トレース・ファイルは、パフォーマンスのチューニングだけでなく、何がいつ実行されたかを知る上でも役立つ多くの情報を提供します。

トレース・オプションは、メイン・メニュー（「ツール」→「トレース」を選択）、コマンド・ライン引数 TRACEFILE（トレース情報のファイル名）、TRACEMODE または TRACEOPTS（トレースが必要なイベント・タイプのリスト）のどちらからでも設定できます。

3.3.2 サーバーおよびネットワーク固有の測定

データベース・サーバーおよびネットワーク・システムは通常、これらの領域のパフォーマンス情報を得るために使用する測定および分析ツールを提供します。

たとえば、SQL をチューニングするのに非常に貴重な支援ツールは、Oracle データベース・サーバーが備える SQL トレース機能です。SQL トレースによって、データベースに送られる SQL だけでなく、文のデータの解析、実行およびフェッチに要する時間が参照可能になります。トレース・ファイルの生成が完了したら、TKPROF ユーティリティを使用して Explain Plan を生成します。これは Oracle Optimizer が使用する実行プランのマップです。Explain Plan は、たとえば表の完全スキャンを使用した場所を示し、（パフォーマンス・ヒットによっては）アプリケーションがインデックスを使用すれば利点があることを提案します。Explain Plan の詳細は、『Oracle SQL 言語リファレンス・マニュアル』を参照してください。

アプリケーションが使用しているサーバーやネットワーク・システムが提供する測定ツールや分析ツールで確認するだけでなく、これらの領域の管理者にも問い合せてください。既存の環境でアプリケーションのパフォーマンスを改善するための方法を直接支援、または提案できるかもしれません。

3.4 パフォーマンス改善の一般的なガイドライン

次のパフォーマンス改善ガイドラインは、Forms Developer および Reports Developer 全般（コンポーネント Builder すべて）と、実行環境（クライアント / サーバーおよび 3 層）の両方に適用できます。

一般的なガイドラインでは次の領域について説明します。

- ハードウェアとソフトウェアのアップグレード
- データ設計（データ・モデリング）
- コンポーネント間での作業の共有
- 待ち時間の移動
- デバッグ・モード

3.4.1 ハードウェアとソフトウェアのアップグレード

パフォーマンスを改善する最も簡単な方法は、ハードウェアまたはソフトウェア、あるいはその両方をアップグレードすることです。アップグレードに関わる作業はありますが、新しいコンポーネントによって得られるパフォーマンスの改善は通常その作業に値するものです。

3.4.1.1 ソフトウェアのアップグレード

3.4.1.1.1 Oracle ソフトウェアのアップグレード

Oracle ソフトウェアの各後継リリースでは、パフォーマンスが向上し、前のリリースの機能も拡張されます。改善はさまざまなカテゴリにわたって見られ、当然ながらリリースごとに変化します。ただし多くの場合、新しいリリースでは新機能だけでなく、何らかのパフォーマンスの向上として新しいチューニング支援や自動パフォーマンス改善などの機能も提供します。

たとえばリリース 1.6 では、リリース 1.5 を改良し、複数のアプリケーション・サーバーを効率的に使用することが可能になるロード・バランサーを提供しています。リリース 2 では、戻りレコード表機能を提供します。この機能では、ストアド・プロシージャへ 1 度変更を渡すことで、ネットワーク・トリップを防ぎながら複数の表へ順番に変更を配布することができます。また別の例として、リリース 6 は、データベース・サーバーへのインタフェースとなる OCI 言語をより効率的に使用する変更された内部コードを含み、顧客のアクションを必要としない改善を提供します。

最新のより効率的なリリースへのアップグレードを考慮 してください。

3.4.1.1.2 他のソフトウェア・コンポーネントのアップグレード

当然ながら Forms Developer および Reports Developer は、他のソフトウェア、特に Oracle データベース・サーバーおよび PL/SQL 言語コンポーネントと同時に実行されます。関連するコンポーネントのパフォーマンスの向上は、多くの場合アプリケーションのパフォーマンスの向上に直結します。これらの領域のチューニング機能は、アプリケーションのパフォーマンスを改善するより多くの機会を提供します。

関連ソフトウェアの最新のリリースを使用することで、常により良いパフォーマンスを得ることができます。たとえば Oracle8 データベース・サーバーは、Oracle7 と比べて随所でパフォーマンスが改善されています。たとえば、表と索引の分割、並列処理の向上、整合性の検証の譲歩などです。

したがって、データベース・サーバーや他の関連ソフトウェアの選択を制御したり、それに影響を与えたりすることができる範囲で、より最新の効率的なレベルにアップグレードすることを考慮してください。

3.4.1.2 ハードウェアのアップグレード

ハードウェア・システムの基本となる容量または速度、あるいはその両方を増加することは、パフォーマンスを改善する明白なアプローチです。このアプローチには、デスクトップおよびサーバー・マシンだけでなくその間のネットワーク接続も含まれます。

3.4.2 データ利用の提案

データベースへのアクセスは、典型的な Forms Developer および Reports Developer アプリケーションでの主要アクティビティです。データを効率的に読み書きできることが、パフォーマンス全体へ重要な影響を与えます。

3.4.2.1 配列処理の使用

Forms Developer および Reports Developer では、Oracle データベース・サーバーの配列処理機能を使用することができます。これは、一度に 1 つずつではなくバッチでデータベースからレコードをフェッチすることを可能にし、結果的にデータベースのコールを大幅に減らすことになります。配列処理のマイナス面は、実行プラットフォームで返されたレコードの配列を格納するためにより多くの領域が必要となる点です。

本番環境でネットワーク上の負荷が主なボトルネックとなる場合は、Developer 製品のランタイム `ARRAYSIZE` パラメータをランタイム環境と同じくらい大きい値に設定します。

3.4.2.2 冗長な問合せの削除

理想的には、アプリケーションには冗長な問合せ（不要なデータを返す問合せ）があってはいけません。パフォーマンスが明らかに低下するからです。ただし、アプリケーションが異なるユーザーに異なる書式を作成する必要がある場合、および異なる問合せ文を使用する必

要がある場合には、この状況が発生します。2 つの異なるアプリケーションを開発することで問題が解決するのは明らかですが、メンテナンスを容易にするためには 1 つのアプリケーションのみであることが望ましいと言えます。

たとえば、レポートで SRW.SET_MAXROW() プロシージャを使用する冗長な問合せを使用しないようにすることができます。Before Report トリガーの次のコードは、ユーザー・パラメータによって Query_Emp または Query_Dept のいずれかを使用不可にします。

```
IF :Parameter_1 = 'A' then
    SRW.SET_MAXROW('Query_Emp',0);
ELSE
    SRW.SET_MAXROW('Query_Dept',0);
END IF;
```

SRW.SET_MAXROW() を使用する場合は、いくつか気を付ける必要がある点があります。

- SRW.SET_MAXROW() を使用する意味がある場所は、Before Report トリガー内（問合せを解析した後）のみです。SRW.SET_MAXROW() がこの時点の後でコールされた場合、パッケージされている例外処理 SRW.MAXROW_UNSET が発生してしまいます。
- 問合せは解析されて結合されますが、レポートへはデータを返しません。

3.4.2.3 データ・モデルの改善

アプリケーションがデータベースに多くの時間を費やすことがわかっている場合、データの構造とその利用方法を見直すことに利点がある場合がよくあります。Forms Developer および Reports Developer は、設定に基づいたロジック向けに最適化されている非手続き型ツールであり、スキーマ設計が不適切である場合、きわめて悪い影響を及ぼす可能性があります。たとえば、何度も正規化したデータ・モデルは不要な結合や問合せを回避できますが、適切なインデックスがなければ表の完全スキャンを無駄に増やすことになります。

アプリケーションの固有の特徴から最も効率的なデータ・モデルが決定します。問合せ駆動型アプリケーションは表を非正規化することで利点があります。正規化した表は更新や挿入を頻繁に行うアプリケーションに適しています。

データベースおよびサーバーの効率的な設計と操作は明らかにクライアント・アプリケーションに利点をもたらします。ただし、データベースの作成と管理は大きな問題であり、通常はアプリケーション開発者が関わる領域ではないため、データベース・パフォーマンスについてはここでは紹介するのみとします。詳細は、『Oracle Server チューニング』マニュアルを参照してください。マニュアル、および SQL トレースや TKPROF ユーティリティなどのサーバー・ツールを使用して、データ・モデルの改善できる場所を調べることができます。

その領域が直接制御できる範囲外だとしても、データベース・サーバーの管理者に、特定のパフォーマンス問題を処理することが相互の利益になるかどうか問い合せてください。

3.4.2.4 SQL および PL/SQL の効率的な使用

Forms Developer および Reports Developer は、SQL を使用してデータベースと対話し、データを取り出します。アプリケーションをチューニングする場合、十分な SQL の作業知識があり、データベースが SQL 文を実行する方法を理解していると役に立ちます。

3.4.2.4.1 適切なオプティマイザの選択

アプリケーションの非効率的な SQL は、パフォーマンスに甚大な影響を与える可能性があります。これは特にアプリケーションに多くの問合せがある場合に当てはまります。

Oracle データベース・サーバーには 2 つの SQL オプティマイザがあります。コスト・ベースとルール・ベースのオプティマイザです。コスト・ベースのオプティマイザを使用すると、SQL の複雑なチューニングに関わることなく大量の自動最適化が行えます。さらに、詳細にチューニングするためのヒントが提供されています。ルール・ベース（帰納的）オプティマイザを使用して、SQL を詳細にチューニングして最適化のレベル以上まで達することも可能ですが、実際には新たな作業と SQL 処理の理解が必要となります。

ほとんどのアプリケーションの場合、コスト・ベースのオプティマイザで満足のいく最適化レベルが得られます。実際、チューニングをしていないコスト・ベースの最適化が、手動でチューニングしたルール・ベースの最適化より優れていることもよくあります。ただし、データの広がりおよびオプティマイザを制御するルールを理解している開発者で、さらに高いレベルの効率を目指す場合は、ルール・ベース方法を試すこともできます。

いずれにしても、どちらかのオプティマイザを選択することが重要です。次のいずれかを選択します。

- コスト・ベースのオプティマイザを（表で `ANALYZE` を実行するか、または `init.ora` パラメータを設定して）アクティブ化するか、あるいは
- ルール・ベースのオプティマイザが提供する提案とアクセス・パス選択に従ってすべての SQL を最適化します。

3.4.2.4.2 問合せ SQL 内での計算の実行

アプリケーション内で計算を実行する場合、一般的な目安としてできるだけ多くの計算を問合せ SQL 内で実行した方がパフォーマンスが向上します。SQL に計算が含まれている場合、データはデータベースによって計算されてから返されますが、アプリケーションが計算する場合は返されたデータをキャッシュしてから計算を行います。Oracle 7.1 以降では、問合せ選択リストにサーバー・ストアド・ユーザー定義 PL/SQL ファンクション・コールを含めることができます。これは計算されたデータがデータベースからの結果の一部として返されるため、それ以上計算する必要がなく、（たとえば、計算列内で）ローカル PL/SQL ファンクションを使用するよりも効率的です。

Oracle 8 では、メソッドのコールで、引数の受渡しを簡素化し高速にする SELF パラメータを使用することができます。

3.4.2.4.3 明示的なカーソルの回避

アプリケーションで明示的なカーソルを宣言し使用することで、データベースの問合せを完全に制御することができます。ただし、このようなカーソルが必要なことはほとんどありません（Forms Developer および Reports Developer は、必要なカーソルをすべて暗黙的に作成し、管理します）。明示的なカーソルはネットワーク通信量も増加させるため、ほとんどのアプリケーションでは使用を避けなければなりません。

3.4.2.5 グループ・フィルタの使用

グループ・フィルタは Reports および Graphics コンポーネントで利用することができます。

グループ・フィルターは主に、取り出したレコード数を最初または最後の n レコードに制限するために使用しますが、PL/SQL フィルタ条件を作成するオプションもあります。グループ・フィルタを使用する場合も、問合せはデータベースに渡され、すべてのデータがアプリケーションに返されますが、ここでフィルタを行います。このため、アプリケーションは最初の 5 レコードのみを表示しますが、返された結果には問合せによって返されたレコードすべてが含まれています。

このため、できるだけグループ・フィルタを問合せの WHERE 節に組み込むことがより効率的です。こうしてデータベースが返すデータを制限します。

3.4.2.6 コンポーネント間での作業の共有

Forms Developer および Reports Developer は、複数のコンポーネントを使用するアプリケーションを作成する機能を提供します。たとえば、データをフェッチして Forms で操作し、Forms から Reports に出力を作成するようコールすることができます。Report も Graphics をコールして出力を視覚的に表示することができます。

各コンポーネントはデータを再問合せすることができるため、Forms はデータを保持するためにレコード・グループを作成し、次にパラメータとして Reports へ渡し、Repots も同様に Graphics へ渡すとより効率的です（このテクニックは問合せの分割とも言います）。このテクニックを使用する場合、データの問合せは一度のみ行われます。

3.4.2.7 待ち時間を先に移動

あるコンポーネントが別のコンポーネントをコールし、コールされたコンポーネントがまだメモリーにない場合、ロードするのにいくらか時間がかかります。このような必要に応じての呼出しはメモリーをより効率的に使用できますが、エンド・ユーザーにとってはかなりの待ち時間が発生します。

（コール側コンポーネントが）コールされるコンポーネントを先にロードしておくことで待ち時間を減らすことができます。これによって起動時間は長くなりますが（また、メモリーの利用効率も悪くなりますが）、通常、起動時の待ち時間は処理中の待ち時間よりも気にならないものです。

（このテクニックは Forms が Reports をコールする場合の方が Reports が Graphics をコールする場合よりも有効です）。

3.4.3 Forms 固有の提案

この章のここまでの一般的な提案はすべて Forms アプリケーションにも適用されます。さらに、次の項目について考慮します。

3.4.3.1 配列処理のチューニング

配列処理に関する一般的な数値やトレードオフについてはすでに記述しました。Form では、問合せの設定はブロック・プロパティ「問合せ配列サイズ」で制御します。更新 / 挿入 / 削除については配列処理設定はブロック・プロパティ「DML 配列サイズ」で制御します。

3.4.3.2 ストアド・プロシージャに基づくデータ・ブロック

可能な限り、データ・ブロックはストアド・プロシージャに基づかせます。

ストアド・プロシージャは処理をサーバーに移動する最も直接的な方法です。正確に設計されていれば、ストアド・プロシージャは多くのネットワークのラウンド・トリップを取り除くこともできます。たとえば、問合せをストアド・プロシージャに基づくようにすることで、外部キーの参照および計算を Post-Query トリガー内ではなくサーバー上で行うことができます。このようなトリガーは通常、1 行あたり少なくとも 1 ラウンド・トリップを追加するので、配列フェッチの利点がなくなります。

同様に、ストアド・プロシージャを通して更新を行うことで、監査証跡または非正規化データを新しいネットワーク・ラウンド・トリップなしに記述することができます。DML を実行する前に必要となる妥当性チェックについても同様です。これによって、これまで Pre-Update、Pre-Insert、Pre-Delete トリガーで発生していたネットワーク・ラウンド・トリップを排除します。

2.0 より前のリリースを使用している場合は、On-Select や On-Fetch などのトランザクション・トリガーを記述することで、手動でストアド・プロシージャに基づくデータ・ブロックを作成することができます。リリース 2.0 以降を使用している場合は、ストアド・プロシージャを通してフェッチを行うことができます。

ストアド・プロシージャにはまた、問合せに関する 2 つのオプションもあります。1 つ目のオプションはデータ・ブロックの問合せを、「Ref カーソル」を返すストアド・プロシージャに基づいて作成します。もう 1 つのオプションは、「レコード表」を返すストアド・プロシージャです。

3.4.3.2.1 Ref カーソルに基づく問合せ

Ref カーソルは PL/SQL 構造体で、ストアド・プロシージャがカーソルをオープンし、クライアントに「ポインタ」またはカーソルへの参照を返すのを可能にします。これによってクライアントは、カーソル自体をオープンした時と同じようにカーソルからレコードをフェッチすることができます。Forms の場合は、Forms が表またはビューに直接基づくデータ・ブロックのカーソル自体をオープンした時と全く同じように、配列フェッチを使用して Ref カーソルを通してレコードをフェッチします。

Ref カーソルに基づくデータ・ブロックには、ビューに基づくデータ・ブロックと多くの類似点がありますが、Ref カーソルには 2 つの主な利点があります。1 つ目の利点は、ストアド・プロシージャがより優れたデータのカプセル化を提供する点です。表への直接問合せによるアクセスを拒否することで、意味のある方法（たとえば、特定の情報を作成するために特別な方法で結合するように設計された表）または効率的な方法（たとえば、インデックスが利用できるような方法での問合せ）でのみアプリケーションがデータを問い合わせることを保証できます。

2 つ目の利点は、ストアド・プロシージャをより柔軟にすることができる点です。プロシージャは、カーソルのオープンで複数の Select 文のどれを実行するかをランタイム時に決定することができます。この判定はユーザーの役割または権限によって決まります。たとえば、マネージャは Emp 表のすべての列を参照できますが、事務員には給与の列が空白で示されます。あるいは、異なるデータ・セット（たとえば過去のデータと現在のデータ）を 1 つのデータ・ブロックに表示できるようにパラメータによって判定します。PL/SQL を記述できる場合には、判定を好きなだけ複雑にすることができます。唯一の制限は、異なる Select 文のすべてが互換性のある列を返さなくてはならないことです。Select 文をランタイム時に動的に構成することはできません（データベースは動的 SQL を使用する Ref カーソルをまだサポートしていません）。

注意 : REF カーソルを使用すると Query-by-Example は使用できません。

3.4.3.2.2 レコード表に基づく問合せ

PL/SQL リリース 2.3 で紹介した、レコード表はデータベースの表に似ているメモリー内構造です。ストアド・プロシージャは、配列のように 1 行ずつ構築することができる、文字どおりあらゆるデータから成るメモリー内の表を作成します。Ref カーソルは SQL での作成方法がわかれば何でも返すことができますが、レコード表は PL/SQL での作成方法がわかれば何でも返すことができます。サーバー側での参照や計算の実行ができるだけでなく、返されたレコードのどのレコードを含めるか、または除くかについて複雑な判定を行うこともできます。

PL/SQL では比較的簡単にできて、SQL ではかなり困難なことの例としては、各部署内で給与が上位 5 番目までに入る従業員を返す場合などがあります（SQL でこれが困難なのは、上位 5 番目の給与と同じ給与の人が複数いる可能性があるからです。PL/SQL では、これは比較的簡単なループです）。

Forms の問合せにコールされた場合、プロシージャはサーバ側でレコード表を作成します。次に、ネットワーク・パケット・サイズで許されるできるだけ少ない物理ネットワーク・ラウンド・トリップを使用して、クライアントに結果全体を一度に返します。表の各レコードは Forms ブロックの行になります。これはサーバーのリソースを解放し、ネットワーク帯域幅を非常に効率的に使用しますが、クライアントのリソースを犠牲にし、おそらく不要なレコードのネットワーク通信量を浪費します。

マスター / ディテール問合せに使用する場合は、レコード表テクニックが問合せのすべての詳細レコードを返す点に注意してください。したがって、このテクニックは小さ目の問合せにのみ適しています。

要約すると、レコード表は結果を判定する上では優れた柔軟性がありますが、使用には注意が必要です。

注意 : REF カーソルを使用すると Query-by-Example は使用できません。

3.4.3.2.3 レコード表に基づく挿入 / 更新 / 削除

リリース 2.0 では、ブロックでの挿入、更新および削除を行うストアド・プロシージャに返されるレコード表も使用することができます。ストアド・プロシージャは、必要な表に変更を“広げる”ことができ、データ・モデルが高度に正規化されている場合は多くのネットワーク・ラウンド・トリップを潜在的に節約することもできます。その他の用途としては、監査証跡の記述があります。そのためには、Insert、Update、Delete の各プロシージャが必要です。

普通の表に基づくブロックの場合、Forms はレコードが挿入、更新または削除されたものなのかを判定するために各レコードの状態を自動的に保持します。コミットの際に、Forms は挿入されたレコードすべてに対するレコード表、更新されたレコードおよび削除されたレコードについてもそれぞれのレコード表を作成します。次にそれぞれのプロシージャをコールし、対応するレコード表を渡します。問合せの場合は、レコード表は“1 回”で渡されます。この場合は、いずれにしてもレコードはすべてサーバーに送ってコミットする必要があるため、一度に表全体を送ることに不都合はありません。

3.4.3.2.4 テクニックの組合せ

最後に、これらのテクニックをそれぞれ好きなように組み合せることができることについて説明します。たとえば、Ref カーソルを通して問合せを選択し、レコード表を通して DML を実行することで、両方の利点を活用することができます。

3.4.3.3 トランザクションの SQL 処理の最適化

デフォルトでは、Forms はフォームを暗示的に実行するか、またはデータのポスティングや問合せの一部として実行する各 SQL 文に別々のデータベース・カーソルを割り当てます。各カーソルの文は毎回ではなく Runform セッションで実行された最初の時だけ解析する必要があるため、この動作によって処理が向上します。

Forms では、すべての暗示的な SQL 文（問合せ SELECT 以外）に対して 1 つのカーソルを使用することで、メモリーを節約することができます。これを行うには、ランタイム・オプション `OptimizeTP` を No に設定します。ただし、通常、メモリーの節約はそれほど意味がなく、これを行った場合、Insert、Update、Delete および Update 文の Select 文は実行されるたびに解析する必要があるため、処理速度が低下します。

このため、`OptimizeTP=NO` 設定を使用することは避けることをお薦めします。

3.4.3.4 トリガーの SQL 処理の最適化

デフォルトでは、Forms はトリガーでフォームが明示的に実行する各 SQL 文ごとに別々のデータベース・カーソルを割り当てます。各カーソルの文は毎回ではなく Runform セッ

ションで実行された最初の時だけ解析する必要があるため、この動作によって処理が向上します。

Forms では、トリガーのすべての SQL 文に対して 1 つのカーソルを使用することでメモリーを節約することもできます（これを行うには、ランタイム・オプション OptimizeSQL を No に設定します）。ただし、通常、メモリーの節約はそれほど意味がなく、これを行った場合、SQL 文は実行されるたびに解析する必要があるため、処理速度が低下します。

このため、OptimizeSQL=NO 設定を使用することは避けることをお薦めします。

3.4.3.5 フォーム間ナビゲーションの制御

大きいアプリケーションは複数の小さいフォームに分割して、必要に応じてさまざまなフォーム間をナビゲートする方が効率的な場合があります。

頻繁に使用するフォームは最初に使用した後オープンしたままにしておくことで、使用するときに毎回オープンしてクローズするよりナビゲーション時間を減らすことができます。フォームのクローズおよび再オープンは相当のオーバーヘッドにつながり、パフォーマンスを低下させます。

フォームをオープンしたままにするためには、NEW_FORM ビルトインではなく OPEN_FORM ビルトインを使用してナビゲートします（NEW_FORM は新しいフォームをオープンする際、前に使用していたフォームをクローズします）。

3.4.3.6 レコード・グループのフェッチ・サイズの増加

フェッチ・サイズが大きいほど、レコード・グループを得るために必要なフェッチ回数が減ります。アプリケーションでレコード・グループを使用する場合（または、ランタイム時にレコード・グループを作成する場合）、「レコード・グループのフェッチ・サイズ」プロパティを使用してこのサイズを設定します。

3.4.3.7 LONG のかわりに LOB を使用

Oracle8 サーバーを使用している場合、LONG や LONG RAW ではなく LOB（大きいオブジェクト）データ型を使用する方がより効率的です。

3.4.3.8 グローバル変数を削除

各グローバル変数は 255 バイト使用します。アプリケーションで多くのグローバル変数を作成する場合、変数は不要になった時点で削除することを考慮してください（削除には、Erase ビルトインを使用します）。

3.4.3.9 Microsoft Windows でのウィジェットの作成を削減

次の提案を採用すると、Microsoft Windows 上で実行される非常に大きいフォームのリソース使用が改善される可能性があります（これらの提案は、標準設計の場合では使用を控え、リソース使用が問題になっている場合にのみ使用してください）。

- コマンドのいくつかは、フォーム上のボタンではなく「Special」メニュー上の項目として利用できるようにします。
- 水平スクロール領域は、列のサブセットのみを一度に表示するその他の領域に変更します。
- 1 つのウィンドウ内で複数のその他の領域を使用するか、ウィンドウ・フローを改善することでウィンドウ数を減らします。
- 特定の情報（主キーおよび説明フィールド）の複数行とそれほど重要でない情報の 1 行を、オーバーフロー領域またはコンビネーション・ブロックを使用して表示します。
- 「表示」プロパティをできるだけ使用します。
- 一般に、フォームで使用するナビゲート・ウィジェット（非表示項目）の数を制限します。

3.4.3.10 ロックの必要性を検証

ユーザーがデータを更新すると、Forms によってレコードがロックされ、データベースへのラウンド・トリップが実行されます。

1 人のユーザーのみがデータを更新している場合は、ロックは必要ありません。ロックを解除するには、Form の「隔離モード」プロパティを「逐次可能」に、ブロック・プロパティ「ロック・モード」を「延期」に設定します。

ただし、Forms のロックの抑止はデータの同時使用がないことが確実である場合にのみ行うようにしてください。

3.4.4 Reports 固有の提案

この章のここまでの一般的な提案はすべて Reports アプリケーションにも適用されます。さらに、次の項目について考慮します。

3.4.4.1 中心とする領域

データベースからデータを取り出した後、Reports はユーザーが作成したレイアウト・モデルに従って出力をフォーマットする必要があります。レイアウトの生成に要する時間は要素数に依存しますが、ほとんどの時間は他のオブジェクトによるオブジェクトの上書きを防ぐことと、フォーマット・トリガー内の計算やファンクションの実行などに使用されています。これらの 2 つの領域の効率を改善すれば、相当の利点があります。

3.4.4.2 レイアウトのオーバーヘッドの削減

デフォルトのレイアウトを作成するときに、Reports はレポートを実行した場合に上書きされないように保護するため、事実上各オブジェクトの周囲に枠を挿入します。実行時に、すべてのレイアウト・オブジェクト（枠、フィールド、ボイラープレートなど）は、上書きされる可能性がないかを検証されます。状況によっては（たとえば、ボイラープレート・テキスト列見出し）、オブジェクトが上書きされる危険性はないことが明らかであれば、すぐに

周囲の枠を削除することができます。これによって、Reports がフォーマットする必要があるオブジェクト数が減り、パフォーマンスが改善されます。

同様に、オブジェクトのサイズ（水平および垂直方向のいずれかまたは両方を拡大、縮小する変数）が未定義の場合、Reports はオブジェクトとその周囲をフォーマットする前にオブジェクトのサイズのインスタンスを決定する必要があるため、新たに別の処理が必要になります。できるだけこのサイズ指定を固定すると、オブジェクト間のサイズおよび位置関係が確定されるため、別の処理を排除できます。

3.4.4.3 フォーマット・トリガーを慎重に使用

一般的には、フォーマット・トリガーを使用するよりも、宣言的なフォーマット・コマンドを使用する方が好ましいといえます。ただし、フォーマット・トリガーは実行時に変更を行うのには便利です。具体的には、

- 実行時にオブジェクトを動的に使用不可および使用可に変更します。
- 実行時にオブジェクトの外観を動的に変更します。

フォーマット・トリガーを使用する際には必ず慎重に扱い、トリガーは出力媒体上の関連するオブジェクトのすべてのインスタンスだけでなく、実行時にオブジェクトをフォーマットするときにも発生することに注意してください。

上で説明したこれらの 2 つの目的は同じように見えますが、次の例を考えてみてください。ある表レポートは、縦に伸ばすことが可能な 1 つの繰返し枠を含み、ページ保護が設定されています。このレポートがフォーマットされるとき、最初のページの最下段にあと 1 行余白があります。Reports は次の繰返し枠のインスタンスのフォーマットを開始し、関連するフォーマット・トリガーを発生させます。繰返し枠の中のオブジェクトの 1 つが拡張されていることがわかると、この繰返し枠のインスタンスは次のページへ移動されるため、繰返し枠のフォーマット・トリガーが再度発生します。このため、繰返し枠は（2 ページ目の 1 行目に）一度しか表示されないにもかかわらず、フォーマット・トリガーは 2 度発生しています。このフォーマット・トリガーに INSERT 文が含まれていた場合、同じデータの行が 2 行挿入されてしまいます。

また、フォーマット・トリガーはトリガーの発生頻度を最少にするために、オブジェクト / 枠の階層で最高のレベルに位置づける必要があります。たとえば、枠内に 4 つのフィールドがある場合、フィールド・レベルのフォーマット・トリガーは 4 回発生しますが、枠レベルのフォーマット・トリガーが発生する必要があるのは一度のみです。

PL/SQL がフォーマット・トリガー内になければならない場合、できるだけ発生頻度の少ないオブジェクトのトリガーに含めます。たとえば、PL/SQL をフィールドではなく枠のフォーマット・トリガーに含めることで、レポートの実行をより高速にすることができます。フォーマット・トリガーの PL/SQL はそのオブジェクトのインスタンスごとに実行されます。オブジェクトの発生頻度が少ないほど、PL/SQL の実行回数が少なくなり、レポートの実行が高速になります。

特定のオブジェクトに対してフォーマット・トリガーが何回発生するかはわからないため、フォーマット・トリガーでは計算の実行や、DML の使用を行わないでください。

フィールドの表示属性を動的に変更する必要がある場合、すべての属性の変更を SRW.ATTR() を 1 回コールして設定するか、あるいは SRW.SET ビルトインを数回コールし、各コールで各属性を設定することができます。後者のテクニックを使用するとコードが読みやすくなりますが、ランタイム効率は特に多くの属性を設定しなければならない場合は、SRT.ATTR() を 1 回呼び出す方が効率的です。

3.4.4.4 表のリンクを考慮

ほとんどの操作と同じように、複数の表を含むデータ・モデルを作成する方法は何通りもあります。たとえば、部署と従業員の表の結合という標準的な場合を考えてみます。つまり、要件は企業の各部署の従業員すべてをリストすることです。Reports Developer では、プログラマは 1 つの問合せを作成することも、2 つの問合せを作成してその問合せ間にマスター / ディテール・リレーションを使用することもできます。

アプリケーション側では、データ・モデルの設計の際に、多数の簡単な（単一表の）問合せではなく、少数の大きい（複数表の）問合せを使用して、実際の問合せ数を最小にするのがより好ましいといえます。Reports Developer では、問合せが発行されるたびに解析、結合が行われ、カーソルが実行されます。このため、1 つの問合せが、複数ではなく 1 つのカーソルに必要なデータすべてを返すことができます。また、マスター / ディテール問合せでは、ディテール問合せは各マスター・レコードが取り出されるたびに再解析され、再結合され、再実行される点に注意してください。この例では、レポートで 2 つの問合せをマージして、マスター / ディテール効果を作成するためにブレーク・グループを使用するとより効率的です。

ただし、問合せが大きくなり、より複雑になるとメンテナンスが困難になる点に気を付けてください。各サイトはパフォーマンスとメンテナンス要件のバランスを取る方法を決める必要があります。

3.4.4.5 ランタイム・パラメータの設定を制御

レポートを効率的に実行するように設計した場合、特定のランタイム引数を設定することでレポートのパフォーマンス全体をさらに改善することができます。

ARAYSIZE および RUNDEBUG 設定については前に説明しました。

レポートのパラメータ・フォームまたはオンライン・プレビューが必要ない場合は、PARAMFORM および BATCH システム・パラメータを適切に設定することで、これらのファンクションを無視することができます。

3.4.4.6 デバッグ・モードをオフにする

アプリケーションを通常の本番環境で使用している場合は、デバッグ・モードで実行されていないことを確認してください。

アプリケーションをデバッグ・モードで実行すると、情報が収集され、余分な内部操作が実行されます。アプリケーションのデバッグが終了したら、これらの操作はもはや不要で、パフォーマンスを低下させるのみです。

Reports では、デバッグ・モードはランタイム・パラメータ RUNDEBUG によって制御されます。このパラメータを「No」に設定します。

3.4.4.7 透明なオブジェクトの使用

レイアウト・オブジェクト（たとえば、枠および繰返し枠）の枠と塗りパターンを透明にします。

透明なオブジェクトは、PostScript ファイルで作成する必要はありません。結果として、オブジェクトが透明の場合は処理が速くなります。

3.4.4.8 非グラフィック・オブジェクトに固定サイズを使用

非グラフィック・レイアウト・オブジェクト（たとえば、ボイラープレート・テキストやテキスト付きフィールド）を固定サイズにします。つまり、「固定」の「水平拡張度」および「垂直拡張度」です。特に、繰返し枠とその内容を固定するとパフォーマンスが改善されます。

サイズ可変の非グラフィック・オブジェクトは、Report Builder がフォーマットする前にサイズを決定する必要があるため、余分な処理が必要になります。サイズが固定の非グラフィック・オブジェクトは、サイズがすでにわかっているためこの処理が不要です。

3.4.4.9 グラフィック・オブジェクトに可変サイズを使用

グラフィック・レイアウト・オブジェクト（たとえば、イメージおよび Oracle Graphics オブジェクト）を可変サイズにします。つまり、「可変」な「水平拡張度」および「垂直拡張度」です。

固定サイズのグラフィック・オブジェクトは通常、オブジェクト内に内容がおさまるようにスケール変更する必要があります。オブジェクトの内容をスケール変更するには別の処理が必要です。オブジェクトが可変サイズの場合、内容に合わせて伸縮できるため、スケール変更は必要ありません。

3.4.4.10 イメージの解像度の削減を使用

サイズを減らすイメージ・オブジェクトの「イメージ解像度の削減」を指定します（このオプションは、「書式」メニューの描画オプションにあります）。

イメージのサイズを減らす場合は、大きいサイズのときと比べて表示に必要な情報が少なくなります。「イメージ解像度の削減」は不要な情報を削除し、イメージを格納するのに必要な領域を減らします。これは複数の色を使用する大きいイメージの場合に、特に有効です。

3.4.4.11 ワード・ラップの回避

1 行のテキストを含むフィールドを作成し、（たとえば、SUBSTR ファンクションを使用して）指定された幅の中に内容がおさまることを確認します。

テキスト付きフィールドが 1 行以上に広がる場合は、Report Builder はワード・ラップ・アルゴリズムを使用して、フィールドをフォーマットする必要があります。フォーマットする行が確実に 1 行のみであるようにすると、ワード・ラップ・アルゴリズムの追加処理を避けることができます。

3.4.4.12 フォーマット属性の簡素化

同一のフィールドまたはボイラープレート・テキスト内での異なるフォーマット属性（たとえば、フォント）の使用を最小限にします。

フィールドまたはボイラープレート・オブジェクトのテキストに異なるフォーマット属性を多く含む場合は、フォーマットに時間がかかります。

3.4.4.13 ブレーク・グループの使用を制限

ブレーク順序プロパティは、ブレーク・グループのできるだけ少ない列で設定するようにしてください（ブレーク順序はグループの列名の左にある小さな三角形で示されます）。各ブレーク・グループには、ブレーク順序を設定するために少なくとも 1 列は含まれていることが必要です。

ブレーク・グループのソートが必要な場合は、SQL の ORDER BY 節を使用します。これにより、ブレーク順序ですでにソートされた行を返すことになり、クライアント側で行うソートの量を減らすためパフォーマンスが改善されます。

ブレーク順序設定がある各列に、Reports は適切な問合せの ORDER BY 節に新しい列を追加します。ORDER BY の列が少なければ、データベース・サーバーがデータを返す前に行う必要がある作業も少なくなります。ブレーク・グループの作成は、問合せで定義された ORDER BY 節を冗長にする可能性があります。このような場合、データベースで余分な処理を必要とするため、冗長な ORDER BY 節は削除する必要があります。

ブレーク順序列はできるだけ小さく、またデータベース列も（合計列または計算列とは逆に）できるだけ小さくします。この両方の条件は、できるだけ効率的にデータをフォーマットする前に Reports が行うローカル・キャッシングに役立ちます。明らかに、これらの条件が簡単に合致することはありませんが、それでも考慮に値します。

3.4.4.14 Graphics Builder との作業の重複を回避

レポートで参照する Graphics Builder の図表がレポートと同じデータを一部またはすべて使用している場合、レポートから図表にデータを渡します。渡すデータは、Report Builder の図表のプロパティ・パレットで指定することができます。

レポートと図表が同じデータを使用する場合、データの受渡しによって必要なフェッチの回数を減らすことができます。レポートから図表へデータを渡さない場合は、データはレポートと図表によってそれぞれ 1 回ずつフェッチされるため、合わせて 2 回フェッチされます。

3.4.4.15 PL/SQL とユーザー・イグジットのいずれかを選択

状況に応じて、PL/SQL かユーザー・イグジットのいずれかがパフォーマンスを改善することがあります。次に、PL/SQL とユーザー・イグジットのいずれを使用するかを決定する際に考慮する項目を示します。

- Report Builder オブジェクトへの参照が多数必要な場合は、通常 PL/SQL の方が高速です。
- レポートを移植可能にする必要がある場合、またはアクションをグループまたはレポート・レベルで実行する場合は、PL/SQL を使用します。
- レポートを移植可能にする必要がなければ、DML を実行するために PL/SQL ではなくユーザー・イグジットを使用します。
- ユーザー・イグジットは大きいレポートの各レコードについてアクションを実行する場合や、複雑な計算を何度も行う必要がある場合に特に有効です。

PL/SQL を使用するとコードが手続き型になり、しかも移植可能なので強くお薦めします。PL/SQL は、レポート・レベルのオブジェクトを参照する場合にもパフォーマンスに利点があります。ユーザー・イグジットは、パフォーマンスは優れていても結合が必要であり、移植もできません。ユーザー・イグジットは、アクションを非常に大きいレポートで実行する必要がある場合、または大量の計算を行う場合に有効です。PL/SQL プロシージャは、アクションが各グループに対してのみの場合、あるいはレポート・レベルの場合に、より有効になります。また、PL/SQL ですべて行えるわけではありません。外部装置の制御のようなアクションは C プログラムで作成する必要があります。

注意 : パフォーマンスの改善が見られない場合や、ユーザー・イグジットを使用する理由が特にない場合は、簡単で移植可能な PL/SQL を使用してください。

3.4.4.16 DML に SRW.DO_SQL ではなく PL/SQL を使用

DML 文にパラメータを渡すのでなければ、DML には PL/SQL を使用します。

SRW.DO_SQL() は、SRW.DO_SQL() への各コールの際にコマンドの解析、結合および（通常の問合せと同じように）新規カーソルのオープンを必要とするため、使用はできるだけ控えてください。ただし問合せとは異なり、この操作はオブジェクトが SRW.DO_SQL() を発生させるたびに行われます。たとえば、PL/SQL ファンクションが SRW.DO_SQL() を呼び出し、ファンクションが常駐するグループによって 100 個のレコードが返された場合、解析 / 結合 / カーソルの作成の操作は 100 回発生します。このため SRW.DO_SQL() は、通常の SQL 内で行われない操作にのみ使用し、できるだけ実行回数が少ないところ（たとえば、レポートにつき 1 回のみ発生するトリガー）で使用することをお薦めします。

PL/SQL 内で DML 文を記述すると、同じ文を含む SRW.DO_SQL をコールするよりも高速になります。DML 文に SRW.DO_SQL を使用するのは、DML 文を作成するのに結合パラメータを連結することができるからです。たとえば、ランタイム・パラメータ・フォームで入力したパラメータで決定した名前の表を SRW.DO_SQL で作成することができます。

```
SRW.DO_SQL ('CREATE TABLE' || :tname || '(ACOUNT NUMBER
    NOT NULL PRIMARY KEY, COMP NUMBER (10,2))');
```

使用方法 : DML には Oracle 7.1 以降に付属の dbms_sql パッケージを使用することもできます。詳細は、Oracle データベース・サーバーのマニュアルを参照してください。

3.4.4.17 ローカル PL/SQL の使用を評価

PL/SQL コードはローカル（オブジェクト・ナビゲータのレポートのプログラム単位ノード）、またはサーバーの PL/SQL ライブラリに格納することもできます。

条件によって、ローカル PL/SQL は外部の PL/SQL にあるプロシージャやファンクションを参照するよりも高速に実行できる場合があります。ただし、ローカル PL/SQL が自分の環境で高速に実行できるとしても、ライブラリ・メソッド（たとえば、多くのアプリケーション間におけるコードの共有）の利点を損なうことを考慮してください。

3.4.4.18 SRW.SET_ATTR をコールする際に複数の属性を使用

1 回のコールで複数の属性を指定して、SRW.SET_ATTR setattr> 参照のコール回数を最小限にします。各属性ごとに別々にコールするのではなく、1 回の SRW.SET_ATTR のコールで複数の属性を指定することができます。

理由 : SRW.SET_ATTR の呼出し回数が少ないほど、PL/SQL を高速に実行できます。

3.4.4.19 ARRAYSIZE パラメータを調整

配列処理の値についてはすでに記述しました。

Report Builder の ARRAYSIZE 実行引数（たとえば ARRAYSIZE=10）については、できるだけ大きい値を入力します。配列サイズは行数ではなくキロバイトで指定します。ARRAYSIZE は、Report Builder がレポート実行時の問合せごとに使用可能なメモリーのキロバイト数を表します。Report Builder は、1 レコードずつではなく、複数のレコードをバッチでフェッチする Oracle の配列処理を使用します。結果として、バッチ処理でフェッチするデータの量を制御することができます。

3.4.4.20 LONGCHUNK パラメータを調整

Report Builder の LONGCHUNK 実行引数（たとえば、LONGCHUNK=10）については、できるだけ大きい値を入力します。使用するマシンの推奨値については、使用しているオペレーティング・システムの Oracle インストール情報を参照してください。LONGCHUNK は、Report Builder が LONG 値を取り出す増分のサイズを規定します。LONGCHUNK のサイズはキロバイトで指定します。

LONGCHUNK にできるだけ大きいサイズを指定することで、Report Builder が LONG 値を取り出す増分の回数を減らすことができます。

3.4.4.21 COPIES パラメータを調整

PostScript に出力する場合は、COPIES=1 と指定します。

PostScript レポートに対して 1 より大きい値が COPIES に設定されている場合、Report Builder は情報を整理するためにテンポラリ・ストレージにページを保存する必要があります。これによって Report Builder が使用するテンポラリ・ディスク領域の量が相当増え、ファイルへの書込みによってパフォーマンスを低下させる可能性があります。

3.4.4.22 プレビュー時の事前フェッチを回避

Report Builder には、合計ページ数、レポート・マージンまたはヘッダー・ページの総数を表示する機能があります。これは非常に便利な機能ですが、1 ページ目を表示する前にレポート全体を処理する必要があります。

プレビューアまたはライブ・プレビューアのレポートを設計する際に「事前フェッチ」操作を避けることで、レポートの 1 ページ目を高速に表示することができます。

次の項目の場合は、参照される時に依存するデータより先に事前にフェッチすることになります。

- 合計ページ / パネル数
- 総計
- 計算のブレーク列
- Null が指定された場合の値があるブレーク列

ページ総数フィールド・ソースを使用する場合、Report Builder はページ総数を調べるためにすべてのページをテンポラリ・ストレージに保存する必要があります。これによって Report Builder が使用するテンポラリ・ディスク領域の量が相当増え、ファイルへの書込みによってパフォーマンスを低下させる可能性があります。

クロス積グループも事前フェッチを発生させます。クロス積グループのデータをクロス表にするために、Report Builder はデータすべてを先にフェッチする必要があります。これらの項目は実際にパフォーマンスが低下することに注意してください。プレビューアまたはライブ・プレビューアは遅くなりますが、ファイルや別の宛先に書き込む際のパフォーマンスには影響しません。

注意 : 列は表示されていなくても事前フェッチを発生させます。たとえば、総計はレポート出力には表示されませんが、レポートには存在するため、Report Builder が計算する時に事前フェッチが発生します。

3.4.4.23 適切なドキュメント・ストレージの選択

パフォーマンスを向上させるためにドキュメントをファイルに格納します。セキュリティのためドキュメントはレポートおよびデータベースに格納します。データベースからレポートをオープンしたりデータベースにレポートを保存したりする場合、いくつかの Report Builder の表がメモリーに入れられます。したがって、表をキャッシュするのに十分なリソースがあることを確認する必要があります。

ドキュメントについては、ファイルへの書込みおよびファイルからの読込みは、データベースよりはるかに速くなります。

例外 : ファイルのアクセスに混雑しているネットワークまたは遅いマシンを使用する必要がある場合、ファイルを格納したことによるパフォーマンスの改善を感じられない可能性があります。

3.4.4.24 ファイル検索のパス変数を指定

パス変数を指定するとファイル検索およびテンポラリ・ファイルの作成 / アクセスが高速になります（Report Builder は 2 つの環境変数 REPORTSnn_PATH および REPORTSnn_TMP を提供します。これらはファイルを検索する場所とテンポラリ・ファイルを保存する場所を制御します。nn は Report Builder のリリース・レベルを表します）。REPORTSnn_PATH は、レポートによって参照されるファイルがあるパスを指定します。REPORTSnn_TMP は、テンポラリ・ファイルのための十分な空き領域があり、応答時間が速い装置（たとえば RAM ディスク）上にあるパスを指定します。

REPORTSnn_PATH は、Report Builder がファイル（たとえば、リンク・ファイル・ボイラープレート）を検索するデフォルトのパスです。レポートによって参照されるファイルがある場所を指定することで、Report Builder がファイルを取り出すために行う必要がある検索の量を減らすことができます（レポート定義でパスをハードコードするのではなく、REPORTSnn_PATH を指定することで、レポートの移植性も維持できます）。REPORTSnn_TMP は、Report Builder がテンポラリ・ファイルを作成するパスです。

サーバーを使用する場合は、server-name.ora ファイルの SOURCEDIR= パラメータを指定します。REPORTSnn パスを使用する前にこのディレクトリが検索されます。

3.4.4.25 複数層サーバーを使用

複数層 Reports Server は、デスクトップ機で実行するには実用的ではない大規模な生産レポートを効率的に扱うように設計された機能です。

この機能を使用すると、タスクにより適した強力なサーバー上で複数の大きいレポートを同時に実行することができます。サーバーは必要であれば複数の Reports エンジンを起動できるため、さらに効率を高めることができます。さらに、（レポートの生成が 1 回のみですむように）レポート出力をネットワーク上の複数の Reports ユーザーが利用可能なサーバーにキャッシュすることができます。

3.4.5 Graphics 固有の提案

この章のここまでの一般的な提案はすべて Graphics アプリケーションにも適用されます。さらに、次の項目について考慮します。

3.4.5.1 グラフィック・ファイルの事前ロード

グラフィックを使用するアプリケーションの起動時間は、OGD グラフィック・ファイルを事前ロードすればより高速になります。ランタイム時に必要なファイルが分からない場合は、ダミーの OGD を作成し事前ロードすることができます。

3.4.5.2 必要な場合のみ図表を更新

ほとんどの Graphics PL/SQL ビルトインへの引数の一つである更新フラグのダメージの理解および制御ダメージ・フラグをデフォルトにできる場合、Graphics の図表リストが変更されるたびに再描画が発生することを示す TRUE に設定します。このような再描画は必ずしも必要ではありません。

3.4.5.3 図表の更新をループから移動

（ボタン・プロシージャ、トリガーなどを含む）PL/SQL プログラム単位が図表を一度に更新すれば、パフォーマンスが改善されます。更新は、必要がなければループに含めないでください。

3.4.5.4 できるだけ共通要素を使用

Forms または Reports から Graphics アプリケーションをコールする場合、できるだけ多くの要素を共有するように設計します。たとえば、すでに Forms がフェッチしたデータをチャートにする場合、（図表がデータベースに再問合せするのではなく）同じデータを渡してレコード・グループに表示します。

すべてのデータが共有され、Graphics アプリケーションがデータベース・サーバーをコールする必要がなければ、Graphics アプリケーションがコールされるときに、LOGON パラメータを NO に設定します（LOGON が NO に設定されていない場合は、Graphics がサーバーに再接続するため、起動が遅くなります）。

同様に、フォームまたはレポートおよび図表では、同じカラー・パレットおよび同じフォントを使用します。さらに、できるだけ同じ座標系を使用します。

3.4.5.5 DDL 文への DO_SQL プロシージャを制限

DO_SQL プロシージャは DDL 文の実行に使用することができます。ただし、このプロシージャは DML 文の実行には使用しないでください。一般に、DML 文は DO_SQL プロシージャを使用するよりも、プログラム内で実行する方がより効率的です。

3.4.5.6 オブジェクトの参照にハンドルを使用

ビルトイン・サブプログラムを使用して Graphics オブジェクト上で操作する場合は、オブジェクトを識別する必要があります。PL/SQL で何度もオブジェクトを参照する場合、オブジェクトにハンドル（つまり、ポインタ）を割り当て、名前でオブジェクトを識別するのではなく、ハンドルでオブジェクトを識別するとより効率的です。ハンドルを使用することで内部検索時間を削減することができます。

3.4.5.7 ショートカット・ビルトインを使用しないことを検討

Graphics は、オブジェクトの作成のプロセス、属性の取得または設定を簡単にする、一連のビルトイン・サブプログラムを提供します。属性レコード・アプローチのかわりに、これら

のビルトインを使用することで、開発時間を短縮し、読みやすく理解しやすいプログラム単位にすることができます。

ただし、これらのビルトインの使用には実行時のパフォーマンスにマイナスの影響を与えます。ビルトイン問合せをコールするたびに、Graphics は新しい内部属性レコードを定義し移植する必要があります。また複数のルーチンの実行には、1 つのみのルーチンの実行よりも時間がかかります。さらに、これらのビルトイン問合せを使用する場合、アプリケーションはデフォルト設定に依存する必要があります。

おおまかなガイドラインとしては、3 つ以上の属性を設定する必要がある場合には、属性マスクを使用するか、または事前定義済みのデフォルトを使用して独自のショートカットのライブラリを作成する方が効率的です。

3.5 クライアント / サーバー構造の場合

従来のクライアント / サーバー構造では、アプリケーションはクライアント上で実行され、データベースとそのソフトウェアはサーバー側に常駐します。この章のここまでの一般的な提案はすべてクライアント / サーバーの設定にも適用されます。さらに、次のクライアント / サーバー固有の提案について考慮します。

3.5.1 最適なインストール構成を選択

Forms Developer および Reports Developer では、インストール時にソフトウェアの格納場所を選択することができます。各構成には長所と短所があります。状況に応じて最適な構成を選択してください。

- サーバーで製品を管理 : これはクライアント側のディスク領域を節約しますが、クライアントはネットワーク上でソフトウェアを実行する必要があるため、ソフトウェアの実行速度が低下します。
- クライアントで製品を管理 : すべての製品ファイルがクライアント・マシンに完全に複製されます。クライアントのディスク領域のほとんどを占め、インストールおよびメンテナンス・オーバーヘッドが加わりますが、最も高速に実行することができます。

3.5.2 最適なアプリケーションの場所を選択

アプリケーションを作成した後、クライアントまたはサーバーのどちらに保存するかを選択できます。サーバー側にアプリケーションを保存すると、アプリケーションへのアクセスの共有が可能になり、クライアント側のディスク領域の節約にもなります。一方、アプリケーションをローカルのクライアントに保存すると、高速にアクセスできます。

ディスク領域と共有に関する考慮事項に加え、サーバーへ保存すると、さらに優れたセキュリティが提供されます。

これらの考慮事項の中から状況に最も適した保存先を選択します。

3.6 3 層構造の場合

3 層構造では、第 1 層は実行時のユーザーのデスクトップです。これは Forms クライアント部分として知られる Forms ランタイム製品の一部をロードする Java アプレットを実行します。第 2 層はアプリケーション・サーバーです。Forms サーバー部分として知られる Forms ランタイム製品の残りの部分を実行します。第 3 層はデータベース・サーバーです。Forms クライアントと Forms サーバー間、および Forms サーバーとデータベース・サーバー間で通信が行われます

この章のここまでの一般的な提案はすべて 3 層構造にも適用されます。たとえば、アプリケーション・サーバー・コンポーネントとデータベース・サーバーとの対話は、2 層クライアント / サーバー環境のアプリケーション・サーバーとデータベースとの対話と本質的には同一です。このため、PL/SQL の使用の改善やデータベースの効率的な使用などの領域はここでも関連します。

3 層環境では、アプリケーション・サーバーとデータベース（第 2 層と第 3 層）間のみでなく、デスクトップ機のクライアントとアプリケーション・サーバー（第 1 層と第 2 層）間でも通信が行われます。このため、ネットワーク利用を減少させることがここで説明する重要な領域になります。

以下の提案は 3 層環境に固有の提案です。

3.6.1 第 1 層と第 2 層のスケーラビリティの最大化

第 1 層のデスクトップ機のクライアントとアプリケーション・サーバー機のサーバーとの対話は、エンド・ユーザー数が増加するにつれて、より重要なものになっていきます。次の提案は、アプリケーションのスケーラビリティを最大化するのに役立ちます（これらの提案は、どの Forms Developer アプリケーションおよび Reports Developer アプリケーションにも適用されます）。

3.6.1.1 ネットワーク帯域幅の増加

第 1 層と第 2 層の間のネットワーク接続は通常 3 層環境では頻繁に使用されるため、ネットワーク効率がパフォーマンスの重要な領域となります。ここの帯域幅を増やすことで、かなりの改善をもたらすことができます。

3.6.1.2 ランタイム・ユーザー・インタフェースへの変更を最小化

実行中にユーザー・インタフェースに変更するには、第 1 層と第 2 層間で対話が必要になります。このような変更は（エンド・ユーザーが感じる）パフォーマンスを低下させます。

次のタイプのランタイム・アクティビティを避けることで実行を高速にすることができます。

- 動的（ランタイム）可視属性への変更
- 現行レコードの可視属性の使用

- 項目のサイズおよび位置の変更
- ラベルおよびプロンプトの変更
- オブジェクトの使用可、使用不可、非表示

一般原則として、画面の頻繁なリフレッシュに関係するアクティビティは制限します。たとえば、短い間隔の可視タイマーまたはクロックの使用は避けてください（1 分より長い間隔のタイマーであれば、通常問題ありません）。ユーザー・インタフェースは経過クロック時間ではなく、ユーザーからの対話によってイベントを開始するように設計します。

3.6.1.3 スタック・キャンバスを調整

アプリケーションでスタック・キャンバスを使用する場合、可視プロパティを「いいえ」に設定し、エントリでレイズ・プロパティを「いいえ」に設定します。これは、実行時のインタフェースへの変更を最小化します。

3.6.1.4 上位レベルで妥当性チェックを実行

上位レベルで妥当性チェックを行うようにします。アプリケーションの設計およびスケーラビリティの決定がトレードオフに関わる場合があります。たとえば、フィールド・レベルの妥当性チェックは、ブロック・レベルの妥当性チェックよりもかなり多くのネットワーク通信量を発生しますが、ユーザーにとってはより対話的です。

3.6.1.5 メニュー項目の使用可および使用不可を回避

プログラム的にメニュー項目を使用可や使用不可にすると、Webforms のパフォーマンスが低下します。

3.6.1.6 画面サイズを小さくする

アプリケーションでグラフィックを使用する場合、表示するファイルのサイズを制限するとパフォーマンスの低下を防ぎます。表示サイズを小さくするには、たとえば次のようにします。

- 表示でのレイヤー数を制限します。
- オブジェクトをプログラムによって作成します。
- データ量の多い表示には、ストアド・プロシージャを利用します。

3.6.1.7 グラフィック URL へのパスを識別

アプリケーションでグラフィック（JPG ファイル）を使用する場合、環境変数 `FORMSnn_MAPPING` と `FORMSnn_PATH` を使用して URL の場所を識別します。

3.6.1.8 マルチメディアの使用を制限

マルチメディアはユーザー・インタフェースにとって重要な場合のみ使用します。マルチメディアを使用する場合は、メディア情報を含む URL をコールするボタン・トリガーを定義（または再定義）します。

3.6.1.9 アプリケーション・サーバーから起動されるアニメーションの使用を回避

ネットワーク上でアニメーションを実行すると非常にコストがかかります。このような要素が必要な場合は、クライアント側にあるグラフィック・ファイルのアニメーションの使用を考えてください。

3.6.1.10 ハイパーリンクを利用

カスタム・ハイパーリンクを利用して、ハイパーリンク・ドリルダウンを作成します。このテクニックを使用する場合、コードは実際に必要になるまでユーザーのマシンにはロードされません。

3.6.1.11 コードをライブラリに格納

できるだけ多くのコードをライブラリに入れて、オブジェクトおよびアプリケーション間で共有するコードを最大限に増やし、ロード中のファイル・サイズを最小に抑えます。

リリース 2.0 以降、ライブラリは複数のフォーム間で共有されています。つまり、各フォームごとにプログラム単位が再ロードされ、アンパックする必要がないということです。メモリーにはプログラム単位のコピーが 1 つしかないため、メモリーもそれほど使用しません。

3.6.1.12 JAR ファイルによる起動時のオーバーヘッドの削減

デスクトップ機でアプリケーションの実行を開始する時には、いくつかの Java クラス・ファイルが利用可能となっている必要があります。一般的なアプリケーションでは、これらのファイルは大量にあり、サーバーからこれらをダウンロードすると起動オーバーヘッドが増加します。

リリース 6.0 以降では、これらの Java クラス・ファイルのいくつかを JAR ファイルとしてパッケージしています。JAR ファイルは、アプリケーション起動が高速になるように、アプリケーション・サーバーではなくデスクトップ機に保存することができます。

アプリケーションに必要な残りのクラス・ファイルも、デスクトップ機の JAR ファイルに加えることができます。これは Oracle Java Developer Kit を使用して行うことができます。

3.6.1.13 事前ロードによる起動時のオーバーヘッドの削減

高速なユーザー・インタフェースが望ましい状況では、アプリケーションを事前ロードすると便利です。つまり、実際に集中的な利用が必要となる前に開始します。このようにして、初期ロード段階が完了しているため後続でのコールが高速になります。

3.6.1.14 Just-in-Time のコンパイルを使用

デスクトップからアプリケーションが起動されると、ユーザーはコンパイルしない状態でダウンロードして、デスクトップで実行を開始する時にコンパイルするように選択することができます。このオプションは呼出し時間全体を高速にします。

3.6.2 第 2 層と第 3 層のスケーラビリティの最大化

第 2 層と第 3 層の対話（アプリケーション・サーバーとデータベース・サーバー間の対話）のための提案は、この章で先に記述したクライアント / サーバー環境のための提案と同じです。たとえば、DML 配列サイズ・プロパティやストアド・プロシージャに基づくデータ・ブロックなどを使用することができます。データベース対話のための先の提案はすべてここでも適用されます。

3.6.3 第 2 層のハードウェア能力の増加

3 層システムの基本となるハードウェアすべての能力を増加すると、ほぼ確実にパフォーマンスに好影響を与えます。

最近のテスト結果では、第 2 層プロセッサの処理能力のアップグレードによって、最も顕著な改善が得られることを提案しています。ただし、各サイトおよび状況はそれぞれ異なり、これらの結果が一般的に適用できるわけではありません。

3.6.4 第 2 層のソフトウェア能力の増加

3 層構造では第 2 層コンポーネントの複数のバージョンを使用することができます。Forms Developer または Reports Developer サーバー・コンポーネントのコピーを実行する、複数の中間サーバー・マシンを利用できます。

処理を統合するには Oracle Application Server を使用します。クライアント（第 1 層）マシンからの要求は Oracle Application Server へ送られ、Oracle Application Server から第 2 層サーバーのいずれかへ渡されます。

複数の第 2 層サーバー操作とワーク・ロードの共有によって、第 2 層のパフォーマンスが改善されます。

4

多言語アプリケーションの設計

この章では、多言語アプリケーションの設計方法を説明します。

項	説明
4.1 項「各国語サポート（NLS）」	各国語サポートの概要とコンポーネント
4.2 項「開発時に各国語サポートを使用」	各国語サポートを使用するアプリケーションの開発方法

4.1 各国語サポート（NLS）

Oracle の各国語サポートでは、多言語アプリケーションを設計できます。多言語アプリケーションは、いくつかの異なる言語での利用が可能です。Oracle では、ほとんどのヨーロッパ、中近東およびアジアの言語をサポートしています。

各国語サポートでは、次のことができます。

- 国際的なキャラクタ・セットの使用（マルチバイト・キャラクタ・セットを含む）。
- 言語および地域の適切な表記規則に従ったデータの表示。
- 使用しているアプリケーションのユーザー・インタフェースで表示される文字列の抽出および翻訳。

4.1.1 言語環境変数

次のパラメータを言語環境変数として使用し、言語の設定を指定できます。

パラメータ	指定
NLS_CALENDAR	使用されるカレンダ・システム
NLS_CREDIT	正の通貨の値の表示に使用する文字列

パラメータ	指定
NLS_CURRENCY	各国通貨記号
NLS_DATE_FORMAT	日付に使用するデフォルト書式マスク
NLS_DATE_LANGUAGE	日付に使用するデフォルト言語
NLS_DEBIT	負の通貨の値の表示に使用する文字列
NLS_ISO_CURRENCY	ISO 通貨記号
NLS_LANG	Forms Developer および Reports Developer で使用する言語の設定
DEVELOPER_NLS_LANG	Builder の言語
USER_NLS_LANG	ランタイム・コンポーネントの言語
NLS_LIST_SEPARATOR	リストの項目の区分に使用する文字
NLS_MONETARY_CHARACTERS	通貨の値に対する小数点文字と桁グループ・セパレータ
NLS_NUMERIC_CHARACTERS	数値に対する小数点文字と桁グループ・セパレータ
NLS_SORT	文字データに使用するソート・タイプ

4.1.1.1 NLS_LANG

NLS_LANG 言語環境変数では、Forms Developer および Reports Developer が使用する言語を指定します。

NLS_LANG では次の項目を指定します。

- ユーザーに表示される、「処理中 ...」メッセージのようなメッセージ用の言語を指定。
- DATE 型、NUMBER 型に使用されるデフォルトの書式マスクを指定。
- ソート基準
- キャラクタ・セット

NLS_LANG の構文は次のとおりです。

```
NLS_LANG=language_territory.charset
```

language では、メッセージおよび月日を表示する言語とその表記規則を指定します。言語を指定しない場合、この値はアメリカ英語がデフォルトとなります。

territory では、地域と、デフォルト日付書式、数値に使用する小数点文字、通貨記号、および週と日の計算結果の表記規則を指定します。地域を指定しない場合、この値はアメリカがデフォルトとなります。

charset では、データを表示するキャラクタ・セットを指定します。これは使用する言語およびプラットフォームと一致するキャラクタ・セットでなくてはなりません。この引数では、メッセージの表示に使用するキャラクタ・セットも指定します。

たとえば、アプリケーションをフランス語で実行するとします。このアプリケーションが使用されるのはフランスです。データは WE8ISO8859P1 キャラクタ・セットを使用して表示します。NLS_LANG を次のように設定します。

```
NLS_LANG=French_France.WE8ISO8859P1
```

今度は、アプリケーションをフランス語で実行するが、そのアプリケーションをスイスで使用するとします。NLS_LANG を次のように設定します。

```
NLS_LANG=French_Switzerland.WE8ISO8859P1
```

注意 : NLS_LANG の言語パラメータおよび地域パラメータは、サーバー側とクライアント側で同じ値に設定することをお薦めします（サーバー側の char_set パラメータの値は、データベースが作成され変更できない場合に指定します）。SQL コマンドの ALTER SESSION を使用して、サーバー側の言語パラメータと地域パラメータの値を変更します。たとえば、次の文は言語パラメータをフランス語に、地域パラメータをフランスに変更します。

```
ALTER SESSION
    SET NLS_LANGUAGE = French NLS_TERRITORY = France
```

4.1.1.2 DEVELOPER_NLS_LANG および USER_NLS_LANG

リソースとメッセージ・ファイルの 2 つのセットを同時に使用する必要がある場合、次の 2 つの言語環境変数を使用できます。

- DEVELOPER_NLS_LANG では、Builder の言語を指定します。
- USER_NLS_LANG では、ランタイム・コンポーネントの言語を指定します。

DEVELOPER_NLS_LANG と USER_NLS_LANG の構文は NLS_LANG と同じです。

これらの変数は、次のような状況において NLS_LANG のかわりに使用されます。

- あなたが開発者で、Builder を英語で使用するが Builder で別の言語のアプリケーションを開発中である場合、この 2 つの変数を使用することで、Builder と Runtime とで異なる言語の設定ができます。
- 両方向キャラクタ・セットを使用する言語で実行させるアプリケーションを作成中である場合。
- あなたが開発者で現在 Builder のローカル言語バージョンがない言語で実行させるアプリケーションを作成中である場合。

これらの環境変数が特に設定されていない場合は、NLS_LANG のデフォルト値が使用されます。

4.1.2 キャラクタ・セット

言語環境変数のキャラクタ・セット・コンポーネントでは、ユーザーの環境でのデータ表示に使用するキャラクタ・セットを指定します。異なるキャラクタ・セットで異なるバイナリ値で表される文字がある場合でも、Net8 では、あるキャラクタ・セットを使用して作成されたデータが正しく処理され、異なるキャラクタ・セットを使用するシステムで表示できるようにします。

言語環境変数の詳細は、4.1.1 項「言語環境変数」を参照してください。

4.1.2.1 キャラクタ・セット設計の際の考慮点

多言語アプリケーションまたは複数のキャラクタ・セットで実行する単一言語アプリケーションを設計中の場合も、実行時に最も広範囲に使用されるキャラクタ・セットを判断し、そのキャラクタ・セットの言語環境変数セットで生成する必要があります。

あるキャラクタ・セットでアプリケーションを設計および生成し、別のキャラクタ・セットで実行すると、パフォーマンスが損なわれる可能性があります。さらに、ランタイム・キャラクタ・セットに生成キャラクタ・セットのすべての文字が含まれていないと、認識されていない文字の部分に疑問符が表示されます。

PDF では、マルチバイト・キャラクタ・セットはサポートしていません。

注意 : Form Builder の場合は、ランタイム環境で指定されたキャラクタ・セットと関係なく、生成のときに使用されるキャラクタ・セットが実行時にも使用されます。

4.1.2.2 Windows プラットフォーム上でのフォント・エイリアシング

特定のフォントでアプリケーションを作成しても、そのアプリケーションの実行時に、異なるフォントが使用されている場合があります。これは、非西欧言語環境で英語のフォントを使用する場合に発生しやすくなります。これは、Forms Developer および Reports Developer でフォントに対応付けられているキャラクタ・セットが、言語環境変数で指定されているキャラクタ・セットと一致しているかどうかがチェックされるためです。この 2 つが一致していない場合、Forms Developer または Reports Developer ではそのフォントを、対応付けられたキャラクタ・セットが言語環境変数で指定されているキャラクタ・セットに一致する別のフォントに自動的に置換します。この自動置換によって、データベースから戻されたデータはそのアプリケーションで正しく表示されます。

注意 : 英語フォントを使用してローカル文字を入力すると、Windows では別のフォントとの暗黙的対応付けを行います。

しかし、この置換を希望しない場合もあります。すべての希望するフォントを、フォント・エイリアス・ファイルにある WE8ISO8859P1 キャラクタ・セットにマッピングすれば、この置換は行われません。たとえば、作成したアプリケーションで Arial フォントを使用できない場合は、フォント・エイリアス・ファイルに次の行を追加します。

```
Arial.....=Arial...WE8ISO8859P1
```

言語環境変数の詳細は、4.1.1 項「言語環境変数」を参照してください。

4.1.3 言語と地域

キャラクタ・セットは各言語に必要な個々のキャラクタが利用できることを保証し、各国規則のサポートはデータ項目を正確にローカル表示します。

指定された言語によって次の特性に対するデフォルト規則が決定します。

- サーバー・メッセージの言語
- 月と曜日の名前とその省略形（SQL ファンクションの TO_CHAR および TO_DATE で指定）
- AM、PM、AD、BC に相当する記号
- ORDER BY を指定した時の文字列データのデフォルトのソート基準（GROUP BY は ORDER_BY が指定されないかぎりバイナリ・ソートを使用）
- 記述方向
- 肯定および否定の応答文字列

たとえば、言語がフランス語に設定されると、次のメッセージは、

```
ORA-00942:table or view does not exist
FRM-10043: Cannot open file.
```

次のように表示されます。

```
ORA-00942:table ou vue inexistante
FRM-10043: Ouverture de fichier impossible
```

指定された地域によって次のデフォルトの日付および数値書式特性の規則が決定します。

- 日付書式
- 小数点文字および桁グループ・セパレータ
- 各国通貨記号
- ISO 通貨記号
- 週開始日
- クレジットおよびデビット記号
- ISO 週フラグ
- リスト・セパレータ

たとえば、地域をフランスに設定すると数値は小数点文字としてコンマを使用する書式になります。

4.1.4 両方向サポート

両方向サポートでは、記述方向が右から左である中東および北アフリカの言語でのアプリケーションを設計できます。両方向サポートでは、次の制御ができます。

- レイアウト方向 項目の右側にラベルが付いている項目の表示とチェックボックスおよびラジオ・ボタンの正しい配置を含みます。
- 読込み順序 右から左または左から右のテキスト方向を含みます。
- 文字位置 左上から右上への原点の切替えを含みます。
- 初期キーボード状態 ユーザーがデータを入力するときに、ローカル文字およびローマ字のいずれを自動的にだすかを制御します（ユーザーはこの設定を上書きできます）。

両方向アプリケーションを設計する際に、言語環境変数を NLS_LANG でなく、DEVELOPER_NLS_LANG および USER_NLS_LANG を使用する場合があります。たとえば、Windows 環境でアラビア語のアプリケーションを開発しながらアメリカ英語のインタフェースを使用する場合は、次のように環境変数を設定します。

```
DEVELOPER_NLS_LANG=AMERICAN_AMERICA.AR8MSWIN1256
USER_NLS_LANG=ARABIC_territory.charset
```

言語環境変数の詳細は、4.1.1 項「言語環境変数」を参照してください。

4.1.4.1 Form Builder での両方向サポート

両方向アプリケーションでのオブジェクトの表示方法を指定するには次の４つのプロパティ使用します。方向、整列、読込み順序および初期キーボード状態

方向は、各オブジェクトに可能な限りの機能性を提供する保護プロパティです。テキスト項目および表示項目以外のすべてのオブジェクトでは、方向プロパティは唯一の両方向プロパティであり、その設定によって両方向機能の他の面を制御します。（ただし、リスト項目には初期キーボード状態プロパティも含まれています）

テキスト項目および表示項目には、方向プロパティはありませんが、そのかわりに、これらの項目について整列、読込み順序および初期キーボード状態プロパティを明確に設定できます。

キーボード状態を１つの言語に制限することができます。たとえば、キーボード状態をローカルに設定するとエンド・ユーザーがキーボードを他の言語に切り替えることはできません。

両方向プロパティがデフォルトに設定されると、これらのプロパティは言語環境変数で指定されている記述方向から値を継承します。ほとんどの場合、これによって希望する機能性が提供されます。継承したデフォルト値の上書きは、両方向プロパティを指定するのみでできます。

両方向プロパティの継承は次のとおりです。

フォーム	デフォルトの設定は言語環境変数の値から導出されます。
警告、ブロック、値リスト、ウィンドウ、キャンバスビューなどのすべてのオブジェクト	デフォルト設定は、フォームの値から導出されます。
テキスト項目、表示項目、チェックボックス、ボタン、ラジオ・グループ、リスト項目などのすべての項目	デフォルト設定は、キャンバスビューの値から導出されます。

両方向機能に関連したほとんどのプロパティは、プログラムに基づいたフェッチおよび設定ができます。詳細は、適切なビルトイン・サブプログラムの説明を参照してください。たとえば、ボタンの方向プロパティの値の取得の詳細は、Form Builder のオンラインヘルプ GET_ITEM_PROPERTY の説明を参照してください。

4.1.4.2 Report Builder での両方向サポート

両方向アプリケーションでのオブジェクトの表示方法を指定するには次の 3 つのプロパティを使用します。整列、方向（オブジェクトに対する）および方向（レポートに対する）。両方向プロパティは、次の階層構造にあるオブジェクトに追加されます。

モジュール

ボイラープレート

フィールド

外部ボイラープレート

ボタン

パラメータ・フォーム・ボイラープレート

このリストにないオブジェクトでは両方向サポート（たとえば、イメージ）は必要ないか、または、前述オブジェクトのプロパティのうちの 1 つのデフォルトになります。

4.1.5 Unicode

Unicode はグローバル・キャラクタ・セットで、多言語テキストを 1 つのアプリケーションで表示可能にします。これによって多国籍企業は、1 つの多言語アプリケーションを開発し世界中に配布することが可能になります。

世界市場では次のようなキャラクタ・セットが必要です。

- 製品を一度インプリメントするのみですべての言語が利用でき、どこにでもきわめて簡単にインプリメントすることができます。

- 既存の主要なスクリプトをすべて含みます。
- 多言語ユーザーおよび企業をサポートします。
- インターネットを介して世界規模でのデータ交換を可能にします。

4.1.5.1 Unicode サポート

Forms Developer および Reports Developer では、Unicode サポートを提供しています。Unicode を使用する場合、西欧言語、東欧言語、両方向中東言語などの 1 バイト言語と、中国語、日本語、韓国語（CJK）などのマルチバイトのアジア言語との両方の複数言語を同じアプリケーション内で表示することができます。

すべての言語を網羅する 1 つのキャラクタ・セットを使用することで、様々な言語の様々なキャラクタ・セットを備える必要がなくなります。

たとえば、日本語などのマルチバイト言語を表示するためには、NLS_LANG 環境変数を次のように設定する必要があります（Windows プラットフォームの場合）。

```
Japanese_Japan.JA16SJIS
```

ドイツ語などの 1 バイト言語を表示するためには、NLS_LANG 環境変数を次のように設定する必要があります（Windows プラットフォームの場合）。

```
German_Germany.WE8ISO8859P1
```

この方法の明らかな短所は、アプリケーションが一度に 1 つのキャラクタ・セットの文字しか表示できない点です。複合キャラクタ・セット・データは表示できません。

Unicode キャラクタ・セットでは、特定の言語キャラクタ・セットのかわりに、NLS_LANG のキャラクタ・セット部分を UTF8 に設定することができます。これによって、異なる言語およびキャラクタ・セットの文字も同時に表示できます。

たとえば、日本語とドイツ語を画面上に表示するには、NLS_LANG 変数を次のように設定します。

```
Japanese_Japan.UTF8
```

または

```
German_Germany.UTF8
```

Unicode 機能によって、アプリケーション開発者およびエンド・ユーザーはフォームの多言語テキストを表示することが可能になります。これには Unicode、多言語テキスト、GUI オブジェクトのテキスト（たとえば、ボタンのラベル）、キーボードからのテキスト入力、およびクリップボードのテキストを含む、データベースのテキストが含まれます。Forms Developer および Reports Developer は現在、Windows NT 4.0 および Windows 95（限定サポート）上で Unicode をサポートしています。

注意 : Web 用のアプリケーションを開発する場合は、Java によって Unicode サポートが提供されているため Unicode を使用できます。

4.1.5.2 フォント・サポート

Forms Developer および Reports Developer は、各言語のフォントおよび入力方法については Windows オペレーティング・システムに依存しています。特定の言語でテキストを入力および表示するには、その言語をサポートするバージョンの Windows を実行する必要があります。フォント・サポートは制限されていますが、Windows NT オペレーティング・システムのフォント・サポートには限定されていません。

Windows NT 4.0 では、True Type Big Fonts が提供されています。これらのフォントには、多言語のテキストの表示または印刷に必要なすべての文字が含まれています。多言語のテキストを入力、表示、または印刷したときに正しい文字が表れない場合は、おそらく Big Font が使用されていません。Microsoft 社が NT 4.0 以前で提供している Big Fonts は次のものです。Arial、Courier New、Lucida Console、Lucida Sans Unicode および Times New Roman です。サードパーティの Unicode フォントも使用できます。

4.1.5.3 Unicode サポートの使用不可

Unicode サポートを使用可能にするには、次のように NLS_LANG を設定します。

```
NLS_LANG=language_territory.UTF8
```

言語環境変数の詳細は、4.1.1 項「言語環境変数」を参照してください。

4.2 開発時に各国語サポートを使用

英語以外の言語で Form Builder、Report Builder または Graphics Builder を使用する場合は、言語環境変数で正しい言語と地域を指定するのみで済みます。メッセージ、メニューとメニュー項目、ダイアログ・ボックス、プロンプトとヒント、および警告が適切な言語で表示され、デフォルトの数値と日付、範囲、およびパラメータが適切な形式で表示されます。適切なメッセージ・ファイルが使用できない場合のデフォルトは、US メッセージ・ファイルです。

言語環境変数の詳細は、4.1.1 項「言語環境変数」を参照してください。

4.2.1 書式マスク

4.2.1.1 書式マスク設計の際の考慮点

多言語アプリケーションの日付および通貨フィールドの処理時は、テキストの変換および異なるデータ表示方法が可能になるように、すべての画面上の項目（ボイラープレート、テキスト項目、ボタンなどのインタフェース・オブジェクト、および値リスト）は、スペースを多めに取っておく必要があります。たとえば、日付を 9 文字の DD-MON-YY で表記するアメリカ英語のアプリケーションを開発してノルウェー語で実行する場合は、DD.MM.YYYY という 10 文字のノルウェーでの日付表記ができるようにフィールドのサイズを拡張する必要があります。

また、特別な書式マスクの作成に書式マスク文字を使用する必要があるかどうか、または地域コンポーネントの NLS_LANG で指定されているデフォルトの書式マスクが使用できるかどうかを考慮する必要があります。

無指定の場合のデータ型変換に対しては、PL/SQL では American_America デフォルト書式の DD-MON-YY の項目を想定しているので、PL/SQL で地域固有の型である項目を使用する場合は、正しい書式マスクを指定する必要があります。PL/SQL の地域固有の項目を変換するには、TO_DATE を使用します。

月の名前を含む文字列は、ハードコードしないでください。ハードコードされた月の名前が不可欠な場合は、COPY ビルトインは使用しないでください。COPY を使用すると、指定されている言語によっては、月の名前が誤りになる場合があります。

言語依存例（お薦めしていません）:

```
:emp.hiredate := '30-DEC-97';
copy ('30-DEC-97','emp.hiredate');
```

言語非依存例（お薦めします）:

```
:emp.hiredate := TO_DATE('30-12-1997','DD-MM-YYYY');
```

言語環境変数の詳細は、4.1.1 項「言語環境変数」を参照してください。

4.2.1.2 デフォルト書式マスク

言語環境変数では、月日、数字、日付および通貨などのデータの表示に使用するデフォルトの書式マスクのセットを指定します。具体的には、Forms Developer および Reports Developer では現行の言語環境変数で指定されている地域に対応付けられたデフォルト書式マスクが使用されます。

- Builder では Builder で項目、範囲またはパラメータのデフォルト値が表示されるとき。
- 実行時にはユーザーが、DATE 型または NUMBER 型のような地域固有のテキスト項目にデータを入力する場合。

たとえば、現在の地域がアメリカだとします。タイプ DATE の項目を作成し、デフォルト値 20-OCT-98 を入力します。その後で地域をノルウェーに変更すると、その項目のデフォルト値は 20.10.1998 に変更されます。

言語環境変数の詳細は、4.1.1 項「言語環境変数」を参照してください。

4.2.1.3 書式マスク・キャラクタ

次の書式マスク・キャラクタは、デフォルトの書式マスクの上書きを可能にします。

キャラクタ	戻り値
D	曜日を表す番号（1～7）
DY	曜日名（省略形）

キャラクタ	戻り値
DAY	曜日名（9 文字目まで空白で埋める）
WW	週を表す番号 : アルゴリズム `int((day-jan1)/7)` で計算
IW	ISO 週を表す番号
MON	月名（省略形）
MONTH	月名（9 文字目まで空白で埋める）
RM	ローマ数字月の文字
I、IY、IYY、IYYY	ISO 年の下 1 桁、下 2 桁、下 3 桁、ISO 年それぞれの番号
BC、AD、B.C、A.D.	BC または AD インジケータ（ピリオド有りまたは無し）
AM、PM、A.A、P.M.	AM または PM インジケータ（ピリオド有りまたは無し）

キャラクタ	戻り値
C	国際通貨記号
L	各国通貨記号
D	小数点
G	桁グループ・セパレータ（千単位）

4.2.2 文字データのソート

多言語アプリケーションを設計する際に、文字のバイナリ値ではなく特定の言語のアルファベット規則に従って、文字データをソートする場合があります。SQL ファンクションの NLSSORT を使用すると、これが可能です。

4.2.2.1 WHERE 節内の文字列の比較

WHERE 節にある文字列は、文字のバイナリ値で比較されます。ある文字がデータベースのキャラクタ・セット内でより上位のバイナリ値を持っていると、別の文字より大きいと見なされます。しかし、バイナリ値に基づいた文字列は特定の言語のアルファベット順とは一致しないので、このような比較は誤った結果になります。

たとえば、値 ABC、ABZ、BCD、および ÄBC を含む COL1 という列があるとします。データベースのキャラクタ・セットは ISO 8859/1 です。次の問合せを記述します。

```
SELECT COL1 FROM TAB1 WHERE COL1 > 'B'
```

Ä は B より数値が大きいため、問合せは BCD および ÄBC を返します。

次の問合せを記述するとします。

```
SELECT COL1 FROM TAB1 WHERE NLSSORT(COL1) > NLSSORT('B')
```

言語環境変数の言語コンポーネントがドイツ語に設定されている場合、ドイツ語のアルファベットでは Ä は B よりも前なので、問合せは BCD を返します。言語環境変数の言語コンポーネントがスウェーデン語に設定されている場合、スウェーデン語のアルファベットでは Ä は Z よりも後なので、問合せは BCD と ÄBC を返します。

4.2.2.2 ORDER BY 節の制御

言語環境変数の言語コンポーネントが正しく設定されると、ORDER BY 節で NLSSORT を使用する必要はありません。

次の問合せは正しい結果となります。

```
SELECT ENAME FROM EMP
ORDER BY ENAME
```

4.2.3 NLS パラメータ

4.2.3.1 ALTER SESSION の使用

SQL コマンド ALTER SESSION を使用して NLS デフォルトを上書きします。たとえば、パラメータ（言語や地域など）を作成し、ユーザーがその値を指定するとします。次にユーザーが指定した通りにセッションを変更することができます。

Form Builder では、ALTER SESSION で次の NLS パラメータのいずれかを指定することができます。ただし、Report Builder と Graphics Builder では NLS_SORT パラメータしか指定できません。

パラメータ	説明
NLS_LANGUAGE	サーバーがメッセージおよびエラーを返す時に使用する言語
NLS_TERRITORY	デフォルト日付および通貨マスクに使用する地域
NLS_DATE_FORMAT	日付に使用するデフォルト書式マスク
NLS_DATE_LANGUAGE	日付に使用するデフォルト言語
NLS_NUMERIC_CHARACTERS	小数点文字および桁グループ・セパレータ
NLS_ISO_CURRENCY	ISO 通貨記号
NLS_CURRENCY	各国通貨記号
NLS_SORT	文字列ソート基準
NLS_CALENDAR	現行のカレンダ・システム

たとえば、この文は小数点をカンマに、桁グループ・セパレータをピリオドに変更します。

```
ALTER SESSION
    SET NLS_NUMERIC_CHARACTERS = ',.'
```

これらの新規文字は、数値書式要素 D および G の使用時に戻されます。

```
SELECT TO_CHAR(SUM(sal), 'L999G999D99') Total FROM emp

TOTAL
---------------
    FF29.025,00
```

この文では、この ISO 通貨記号がアメリカ地域の ISO 通貨記号に変更されます。

```
ALTER SESSION
    SET NLS_ISO_CURRENCY = America
```

アメリカ向けに定義された ISO 通貨記号が使用されます。

```
SELECT TO_CHAR(SUM(sal), 'C999G999D99') Total FROM emp

TOTAL
---------------
    USD29.025,00
```

この文では、各国通貨記号が DM に変更されます。

```
ALTER SESSION
    SET NLS_CURRENCY = 'DM'
```

L 数値書式要素の使用時には、新しい各国通貨記号が戻されます。

```
SELECT TO_CHAR(SUM(sal), 'L999G999D99') Total FROM emp

TOTAL
---------------
    DM29.025,00
```

言語環境変数の詳細は、4.1.1 項「言語環境変数」を参照してください。

4.2.3.2 SQL ファンクションでの NLS パラメータの使用

SQL を使用する場合は、次の NLS パラメータを使用してデフォルト NLS 動作を上書きすることができます。

SQL ファンクション	NLS パラメータ
TO_DATE	NLS_DATE_LANGUAGE
	NLS_CALENDAR
TO_NUMBER	NLS_NUMERIC_CHARACTERS
	NLS_CURRENCY
	NLS_ISO_CURRENCY
TO_CHAR	NLS_DATE_LANGUAGE
	NLS_NUMERIC_CHARACTERS
	NLS_CURRENCY
	NLS_ISO_CURRENCY
	NLS_CALENDAR
NLS_UPPER	NLS_SORT
NLS_LOWER	NLS_SORT
NLS_INITCAP	NLS_SORT
NLSSORT	NLS_SORT

4.2.3.3 Form Builder NLS パラメータ

Form Builder ビルトイン・ファンクションを使用して、PL/SQL コード用に言語環境変数の現在の設定値を取得できます。

環境変数	DEVELOPER_NLS_LANG（NLS_LANG のデフォルト）	USER_NLS_LANG（NLS_LANG のデフォルト）
ビルトイン	GET_FORM_PROPERTY	GET_APPLICATION_PROPERTY
パラメータ	MODULE_NLS_LANG	USER_NLS_LANG

USER_NLS_LANG および DEVELOPER_NLS_LANG は NLS_LANG の値のデフォルトなので、これらの変数が特に指定されていない場合は、Form Builder NLS パラメータには NLS_LANG の値が保持されます。

両方の Form Builder NLS パラメータには 4 つのバリエーションがあり、これにより完全な環境変数とその特定の一部のいずれかを取り出すことができます。次の表は、USER_NLS_

LANG 環境変数を戻す GET_APPLICATION_PROPERTY ビルトインの 4 つのパラメータを示しています。

パラメータ	戻り値
USER_NLS_LANG	完全な USER_NLS_LANG 変数
USER_NLS_LANGUAGE	言語部分のみ
USER_NLS_TERRITORY	地域部分のみ
USER_NLS_CHARACTER_SET	キャラクタ・セット部分のみ

DEVELOPER_NLS_LANG 環境変数を取り出すには、MODULE_NLS_LANG パラメータを使用して GET_FORM_PROPERTY をコールします。

言語環境変数の詳細は、4.1.1 項「言語環境変数」を参照してください。

4.2.3.4 Report Builder レポート定義ファイル

キャラクタ・モードでレポートを使用する場合、レポートの物理ページ幅をプリンタ定義ファイル（.PRT ファイル）で定義されているページ幅より 1 文字少なく定義する必要があります。そうでないと、マルチバイト・キャラクタは行の最後の文字スペースから開始し、次の行にまたがって終わることになります。たとえば、物理ページ幅が 80 文字でプリンタ定義の幅が 80 文字の場合、マルチバイト・キャラクタは 80 文字目から開始する可能性があります。マルチバイト・キャラクタは分けることができないため、行はマルチバイト・キャラクタを完成させるために 81 文字目にオーバーフローします。これを避けるには、レポートの物理ページ幅を 79 に設定する必要があります。

次のアラビア語およびヘブライ語固有の NLS パラメータを、プリンタ定義ファイルで設定できます。

パラメータ	用途	値
nls locale	プリント・エンジンの場所を設定	hebrew または arabic
nls datastorageorder	論理データ・ストレージまたは可視データ・ストレージの設定	logical または visual
nls contextuallayout	アラビア語プリンタの事前文脈依存レイアウトを実行	no または yse
nls contextualshaping	アラビア語プリンタの事前文脈依存整形を実行	no または yse
nls pcharste	プリンタのキャラクタ・セットを指定	任意のキャラクタ・セット

5

移植性のあるアプリケーションの設計

Forms Developer および Reports Developer を使用すると、再コンパイルするだけで Windows、Motif、Web、キャラクタ・モードなど、複数のプラットフォーム上で 1 つのアプリケーションを利用できます。Forms Developer および Reports Developer では、標準の制御オブジェクト（ボタン、チェックボックス、ラジオ・ボタンなど）がターゲットのプラットフォームに適した書式に自動的に変換されます。開発前に慎重に計画することによって、環境が異なるユーザーを満足させる 1 つのアプリケーションを、期待どおりの自然なルック & フィールで作成できます。

この章は、移植性のあるアプリケーションを開発するときに直面する問題をあらかじめ想定するのに役立ちます。また、プラットフォーム間で簡単にアプリケーションを移植するための提案を行います。

項	説明
5.1 項「はじめる前に」	移植性のあるアプリケーションを開発する前に答える必要のある高度な質問をいくつか示します。
5.2 項「移植性のあるフォームの設計」	GUI およびオペレーティング・システムに関する移植性の問題について説明します。キャラクタ・モード環境に特有の検討事項の他に、異なるプラットフォームでの開発のアプローチについても説明します。
5.3 項「移植性のあるレポートの設計」	最大限の移植性があるレポートの開発について説明します。
5.4 項「移植性のある図表の設計」	最大限の移植性がある図表の開発について説明します。

アプリケーションを管理するために Project Builder を使用している場合は、1.2.4 項「複数のプラットフォーム間でのプロジェクトおよびプロジェクト・ドキュメントの管理」を参照してください。

5.1 はじめる前に

アプリケーションを移植するつもりがなくても、設計の開始前に、そのアプリケーションで処理する内容について検討することは重要です。最低限、次の質問に答えてください。

- どのプラットフォームをサポートしますか。2 つ以上のプラットフォームで実行する場合、フォント、カラー、レイアウト、画面のサイズおよび画面の解像度などを考えて、いくつかに名前を付ける必要があります。5.2.1 項「GUI の検討事項」は、これらの問題の解決に役立ちます。
- キャラクタ・モードのサポートは必要ですか。必要ならば、オプションはかなり制限されます。推奨事項については、5.2.4 項「キャラクタ・モードのフォームの設計」や 5.3.1 項「キャラクタ・モード環境のレポートの設計」を参照してください。
- どんな図表を表示しますか。図表は、同じ配布プラットフォーム上でさえもかなり変化します。モニターに依存する制限の説明は、5.2.1.2 項「モニターの検討事項」を参照してください。
- アプリケーションは、ユーザー・イグジットまたは外部ファンクションに依存しますか。その場合、各ターゲットのプラットフォームに対してそれらを書き直す必要があります。5.2.2.1 項「ユーザー・イグジットの挿入」で、いくつかの提案と処理方法を説明しています。

5.2 移植性のあるフォームの設計

複数のプラットフォームに対して新しいフォームを設計する場合でも、新しい環境に対して既存のフォームを準備する場合でも、次の項で説明するとおり、2 つの同じ重要な領域に関する問題に直面します。

- 5.2.1 項「GUI の検討事項」
- 5.2.2 項「オペレーティング・システムの検討事項」

複数のプラットフォームに対する開発経験がない場合、複数のプラットフォームに渡る開発方法の推奨事項について、5.2.3 項「複数のプラットフォームに渡るフォームの開発方針」もお読みになることをお薦めします。キャラクタ・モードで開発する場合、この環境に特有の検討事項に関しては、5.2.4 項「キャラクタ・モードのフォームの設計」を参照してください。

5.2.1 GUI の検討事項

移植性のあるアプリケーションを開発する場合、まず、すべてのプラットフォームで同じように見える GUI にするか、または、ユーザーがアプリケーションにプラットフォーム固有の自然なルック & フィールを継承することを期待するのかを判断する必要があります。ほとんどの場合、おそらく後者のアプローチを選択します。ただし、ユーザーが複数のプラットフォームでアプリケーションを使用するような場合、おそらくローカルな規則を無視して、すべてのプラットフォームで同じに見えることを要求します。これを判断する唯一の方法

は、ユーザーがどのように作業し、どの作業を実行しようとしているかに細心の注意を払いながら、ユーザーに尋ねることです（ユーザーのニーズを判断する方法についての提案は、2.1.2 項「ユーザー要件の定義」を参照してください）。

決定したら、次のステップは、サポートする各プラットフォームのオブジェクト・ライブラリを作成することです。オブジェクト・ライブラリは、開発者が作成した一連のオブジェクトおよび標準です。各オブジェクトまたは標準によって、枠、ウィンドウまたは領域の全体の外観とレイアウトが決まります。オブジェクト・ライブラリ内にこれらのオブジェクトを常駐させると、開発プロジェクトまたはサイトのすべての開発者がそのオブジェクトを利用できるようになり、別の場所で作業をしている開発者でも、同じアプリケーションを作成できるようになります（つまり、共通のルック & フィールを使って同じアプリケーション内の別個のモジュールを作成できます）。

オブジェクト・ライブラリの利点を完全に利用するには、各ターゲットのプラットフォームごとに 1 つのライブラリを作成することをお薦めします。ライブラリを支援するために、Form Builder では、キャラクタ・モードに加えて、すべての Forms Developer の GUI 配布環境（Windows 95、Motif）で正しく機能する一連のレイアウトおよび項目となる、Oracle アプリケーション・オブジェクト・ライブラリが提供されています。使用しているプラットフォームで、これらの項目とオブジェクトを 1 つずつテストしてください。ほとんどのオブジェクトは変更せずにライブラリに追加できますが、プラットフォーム固有の要件に合わせるために少し調整が必要なものもあります。

5.2.3 項「複数のプラットフォームに渡るフォームの開発方針」では、オブジェクト・ライブラリを総合的な開発方針に取り込む方法の詳細を説明しています。

5.2.1.1 座標システムの選択

GUI 端末では、実インチ、実センチメートルまたは実ポイント座標システムを使用します。これらのシステムを使用すると、近似文字セル・サイズに指定するかわりに、オブジェクトのサイズを希望どおりの正確な形にできます。

キャラクタ・モードで設計している場合は、文字座標システムを使用して格子指定をオンにします。これにより、オブジェクトのサイズが確実に文字セル・サイズの倍数単位になります。キャラクタ・モード・アプリケーション設計の詳細は、5.2.4 項「キャラクタ・モードのフォームの設計」を参照してください。

5.2.1.2 モニターの検討事項

同じプラットフォームであっても、モニターのサイズや解像度が異なれば、アプリケーションの有用性に大きく影響します。たとえば、Windows 95 が稼動しているラップトップで 6 ポイント・フォントを読むことはできませんが、17 インチ・モニターでは同じフォントを完全に読むことができます。アプリケーションを本当の意味で移植性のあるものにする唯一の方法は、実行環境の各モニターでアプリケーションを完全にテストすることです。

実行環境にサイズの異なるモニターがいくつかある場合、最も小さいサイズに合わせて設計します。ユーザーが使っているモニターとそれぞれを使っている人数を調べることに時間をとると、アプリケーションをより効率よく計画するのに役立ちます。たとえば、移動販売員が指示追跡と販売管理アプリケーションのためにラップトップを使用し、他の人が全員 17

インチの SVGA 端末を使用する場合、ウィンドウのサイズを 2 つのクリティカル・ラップトップ・アプリケーションに制限することによって、作業を簡単にできます。

表 5-1 プラットフォームの制限 : モニター

プラットフォーム	モニターの制限
Windows	サイズは、絶対的な大きさでなく、画面の解像度によって決まります。たとえば、測定システムが同じように見えても、1 インチ当り 96 ドット（96dpi）の 17 インチのモニターで開発されたウィジェットは、20 インチ 96dpi モニターで表示される同じウィジェットより小さく見えます。言い換えれば、インチは必ずしも Windows 上のインチではないということです。
Motif	Motif ユーザーの多くは、グレイスケール・モニターに制限されているので、これらのユーザー向けにカラーは使用できません。

5.2.1.3 カラーの使用

カラーの使用は、同時に正常に動作する 3 または 4 つの基本的なカラーに抑えます。通常、多くのプラットフォームで使用可能なカラーには、青、赤、マゼンタ、シアン、緑および黄色があります。

多数のカラーを使用すると、システムのカラーの上限を超える可能性があり、フォアグラウンド・カラーのみが元のままで、バックグラウンド・オブジェクトが異常なカラーに指定される原因になります。モノクロームやグレイスケール・モニターを含む、すべてのターゲット・システムでカラーの組合せを必ずテストし、それらが予想どおり表示されることを確認します。

表 5-2 プラットフォームの制限 : カラー

プラットフォーム	カラーの制限
Windows	ウィジェットには、システム・カラー・パレットで定義された 16 カラーのうちいずれか 1 つを指定できます。別なカラーを割り当てると、ウィジェットは 16 カラーのうちの最も近いカラーに指定されます。
Motif	Motif ユーザーの多くは、グレイスケール・モニターに制限されているので、重要な区別をするのにカラーを使用しないでください。

5.2.1.4 フォントの問題の解決

フォントは、GUI システムを使用してユーザーに親近感や安心感を与えるのに重要な役割を果たします。表 5-3 に、各 GUI のプラットフォームでお薦めするフォントを示します。

表 5-3 プラットフォーム推奨：フォント

プラットフォーム	フォント
Windows	MS Sans Serif
Motif	Helvetica

移植性のあるアプリケーションを開発する場合、太字イタリック体、および下線などのフォント・スタイルをどのように使用するかを早い段階で決定します（一般に、下線とイタリック体の両方を使用する必要ありません。太字は控えめに、強調の場合にのみ使用します）。異なる図表オブジェクトのタイプ・サイズも標準化する必要があります。たとえば、異なるプラットフォームのフォントを変換する必要がある場合、すべてのラベルを 10 ポイントにすると効果的です。

ユーザーの期待に添うように、移植したアプリケーションは、各プラットフォームで期待されるフォントに変換する必要があります。このため、次のいずれかの方法を使用して、プラットフォーム間でフォントを変換する必要があります。

- 各実行プラットフォームでフォント・エイリアスを定義します。
- ポート固有のクラスを定義します。

次の 2 つの項で、これらの手順の概要を簡単に述べます。

注意 : Motif では、指定したフォントの異なるサイズは、それぞれフォント・ファイルから明示的にインストールする必要がある個別のエンティティと見なされます。たとえば、10、12、および 28 ポイントの Arial フォントを含む Windows ベースのフォームを Motif に移植するとします。単に Arial が Motif にインストール済であることを検証するだけでなく、必要なポイント・サイズ、10、12 および 28 もそれぞれインストール済であることを確認する必要があります。Forms Developer では、ターゲットのプラットフォームに必要なフォントが見つからない場合、プラットフォーム固有の " 近似 " アルゴリズムを使用して他のフォントを代用します。

5.2.1.4.1 フォント・エイリアスの定義

Forms Developer では、各プラットフォームに対してフォント・エイリアス・ファイルが提供されています（`ORACLEHOME¥TOOLS¥COMMON60` ディレクトリの `UIFONT.ALI`）。ほとんどの場合、このファイルによって、複数のプラットフォームでフォントが一貫して表示されることが保証されます。ただし、アプリケーションでカスタム・フォントまたは非標準フォントを使用する場合、すべてのターゲットのプラットフォームでは認識されないものもあります。フォント・エイリアス・ファイルを変更すると、認識されないフォントの代用を定義できます。

ファイルにこの書式で各行を入力します。

```
source_font = destination_font
```

各フォントに対して、次の属性を指定できます。

```
<face>.<size>.<style>.<weight>.<width>.<character_set>
```

例 :

MS Windows から Motif へ移植する場合、すべての MS Sans Serif フォントを Helvetica に変更します。

```
"MS Sans Serif"=Helvetica
```

詳細とフォント・マッピングの例は、Form Builder のオンライン・ヘルプを参照してください。

5.2.1.4.2 クラスの使用

フォント・エイリアシングをさらに制御する必要がある場合、クラスを使用します。たとえば、MS Sans Serif から Helvetica へ正確に変換するのではなく、Motif で異なるフォントを持つポップリストおよびテキスト項目が必要だとします。これを行うには、次のようにします。

1. ポップリストおよびテキスト項目用に 2 つのクラスを作成します。
2. MS Windows では、両方のクラスに `Window.olb` 内のフォントとして MS Sans Serif を使用するように指定します（`Window.olb` の詳細は、5.2.3.1 項「シングル・ソースの作成」を参照してください）。
3. Motif.olb では、ポップリスト・クラスに Helvetica 9 ポイント・フォントを、テキスト項目クラスに Helvetica 11 ポイントを使用するように指定します。

このアプローチにより、アプリケーション内のオブジェクトの各クラスに使用するフォントをカスタマイズでき、より高度な柔軟性を提供できます。

5.2.1.5 アイコンの使用

アイコンはプラットフォーム固有です。アプリケーションでアイコン・ボタンを使用する場合、プラットフォームごとに個別のアイコン・ディレクトリを作成します。各プラットフォームのアイコンには同じ名前を使用し、アイコン・ディレクトリを指し示すようにそれぞれの環境変数を設定します。MS Windows と Motif では、この変数は `UI_ICON` です。

アプリケーションにアイコンを含める場合、次のことを覚えておいてください。

- 小さなモニター（ラップトップのような）に表示されたアイコンは、小さすぎて読めません。
- 特定のアイコンは、特定のプラットフォームで特別な意味を持ちます。

5.2.1.6 ボタンの使用

MS Windows では、ボタンの縁（デフォルトを示すための、ボタンの周りのはっきりした境界線）は Motif のものと比べると非常に小さくなっています。このため、Motif で実行すると、ボタンが縮んで見えます。Motif では、ORACLE_HOME/BIN にある Motif のリソース・ファイル、Tk2Motif を変更することで、これを避けることができます (Oracle では、Motif のリソース・ファイルを使用して UNIX ベースのアプリケーションの視覚的な外観を制御します)。

1. 表示タイプの Tk2Motif ファイルを検索します。
   - .gs（グレイ・スケール）
   - .bw（白黒）
   - .rgb（カラー）
2. Tk2Motif ファイルを編集し、Tk2Motif の「expandNonDefaultButtons」プロパティを True に設定します。

一般に、Motif のより大きなボタン・サイズに適応できるように、常に Windows ボタンに十分な空白を持たせるようにします。

移植性を最大限にするには、すべてのボタンをナビゲート不可にします。Windows と Motif では、ボタンをクリックすると、実際にユーザーがボタンにナビゲートすることを意味します。トリガーはボタン・ナビゲーションに依存することが多いので、プラットフォーム間でのこの違いにより、アプリケーションでの動作が大きく異なります。

注意 : Windows と Motif のボタンをナビゲート不可にすることは、プラットフォーム間での一貫性が標準のプラットフォームの動作を遵守するよりも重要な場合に、行う必要があるトレードオフのよい例です。

5.2.1.7 メニューの作成

表 5-4 に示すように、メニューの配置および動作はプラットフォームによって異なります。

表 5-4 プラットフォームの制限 : メニュー

プラットフォーム	メニューの制限
Windows	マルチプル・ドキュメント・インタフェース（MDI）とシングル・ドキュメント・インタフェース（SDI）をサポートします。MDI では、アプリケーションに属するすべてのウィンドウが 1 つのウィンドウに含まれ、アプリケーション全体で 1 つのメニューしかありません。SDI は、各ウィンドウにそれ自体のメニューがあるという点で、Motif と似ています。
Motif	各ウィンドウには連結されたメニューがあります。親ウィンドウのメニューを、子ウィンドウで繰り返しても繰り返さなくても構いません。

MDI をサポートしているバージョンの Windows を使用しており、すべてのプラットフォームでアプリケーションを同じ外観にする場合、Motif で、親ウィンドウのメニューを子ウィンドウで繰り返さないと指定します。これによって、MS Windows でまったく同じように見える親ウィンドウ・メニューを設計できます。

注意 : 画面がフォーム・モジュール・ウィンドウ間で切り替わるときのちらつきを防ぐには、すべてのメニュー・オプションを 1 つのメニュー・アプリケーションにまとめ、SET_MENU_ITEM_PROPERTY ビルトインを使用して動的にそれぞれのメニュー項目を使用可能または使用不可にします。

5.2.1.8 コンソールの作成

表 5-5 に示すように、メニューと同様、コンソールの配置および動作もプラットフォーム間で異なります。

表 5-5 プラットフォームの制限 : コンソール

プラットフォーム	コンソールの制限
Windows	MDI ウィンドウの一番下にのみ表示。
Motif	ユーザー指定のウィンドウにのみ表示。

複数のプラットフォーム間で一貫性を維持するには、MDI Windows アプリケーションの動作に匹敵するように、Motif アプリケーションの親ウィンドウにコンソールを配置します。

5.2.1.9 その他

- 複数のプラットフォームに対してフォームを作成すると、すべてのプロンプトが右揃えになります。他のプラットフォームに移植されるとテキストは拡張されることが多く、左揃えされたプロンプトが原因でフィールドがシフトされ、不揃いのマージンが作成されます。
- 複数のプラットフォーム間で完全に移植性のある複雑な機能を提供するには、アプリケーションで、1 つ以上の再利用可能なコンポーネントを使用します。
- 表 5-6 に、GUI の移植に関するその他の問題を示しています。

表 5-6 プラットフォームの制限 : 一般

プラットフォーム	全般の制限
Windows	既知の位置付けに関する問題は、VGA 画面で直角を形成する 2 本の線が、実際に SVGA でオーバーラップする原因となります。この問題を避けるには、凹凸を使用します。
Motif	(なし)

5.2.2 オペレーティング・システムの検討事項

アプリケーションが特定のオペレーティング・システムに固有の機能に依存する場合、完全に移植性のあるものにはなりません。次に、一般的なルールを示します。

- メッセージを記述するときに、ポート固有の用語を避けるようにします。たとえば、「ヘルプを参照するには [F1] を押してください」というメッセージは、移植性がありません。
- パス名をハードコードしないでください。パス名はプラットフォームごとに変わります。かわりに、実行時に環境変数を使用して、Form Builder でファイルを検索できるようにします。

  フォームから OPEN.BMP という名前のイメージ・ファイルを読み込む必要があるとします。Windows 専用のアプリケーションでは、READ_IMAGE_FILE のコールでパス名を簡単にコードできます。

  ```
  Read_Image_File('c:¥orawin95¥myapp¥open.bmp', 'BMP', 'block1.image3');
  ```

  ただし、アプリケーションを移植する場合、パス名はプラットフォームごとに異なるのでハードコードすると動作しなくなります。かわりに、外部変数を使用してパス名を表記できます。

  たとえば、Windows95 または WindowsNT では次のようにします。

  1. ORACLE キーの下に path_var というレジストリ・エントリを作成します。UNIX では、path_var という名前のシェル変数を作成します。
  2. TOOL_ENV パッケージの GETVAR プロシージャを使って、このプラットフォームに依存するメソッドを使っている値 path_var を取り出します。

     ```
     path_var varchar2(255);
     ...
     Tool_env.getvar('MYPATH', path_var);
     Read_Image_File(path_var||'open.bmp', 'BMP','block1.image3')'
     ```

     変数 path_var で表されたプラットフォーム固有のパス名が、イメージ・ファイルの名前、OPEN.BMP に追加されます。Window95 では、path_var はパス名 C:¥ORAWIN95¥MYAPP¥ になります。UNIX では、path_var は /oracle_home/myapp/ のようになります。
- HOST ビルトイン・プロシージャによってすべてコールされます。ホスト・コマンドにより、ポート固有のオペレーティング・システム・コマンドが実行されます。アプリケーションを移植しやすくするために、次のようにします。
  1. 各プラットフォームに個別のプロシージャ・ライブラリ（.PLL）を作成します。
  2. すべてのオペレーティング・システム・コマンドを、適切なプロシージャ・ライブラリに置きます。
  3. ジェネリック・プロシージャ・ライブラリを作成します。

4. 各スクリプトが同じ名前を持つことを確認しながら、各プラットフォームのスクリプト・ファイルを書き直します。
5. フォーム・モジュールで、すべてのコールがジェネリック・プロシージャ・ライブラリを参照することを確認します。
6. 指定されたプラットフォームでコンパイルする前に、そのプラットフォームの .PLL をジェネリック・プロシージャ・ライブラリにコピーします。
7. コンパイルします。

5.2.3 項「複数のプラットフォームに渡るフォームの開発方針」では、このプロシージャ・ライブラリの処理を、移植性のあるアプリケーションの推奨開発方針に合わせる方法を説明します。

- 状況判断ヘルプには移植性がありません。アプリケーションで、通常の状況判断ヘルプを使用する場合、Forms Developer の移植性のあるヘルプ・コンポーネント、つまりオンライン・ヘルプ・クラスでそれを置換してください。このコンポーネントを使うと、Windows 95のヘルプと同じようにアプリケーションで状況判断ヘルプを提供できます。

  そのコンポーネントは Form Builder と PL/SQL 本来の機能を使って作成されているので、フォームをサポートするすべてのプラットフォームに移植可能です。作成するヘルプ・テキストはデータベースに格納されるので、すべてのユーザーがアクセス可能で、更新内容はただちにすべてのユーザーが使用できます。

  アプリケーションでオンライン・ヘルプ・クラスを使用するには、次のようにします。

  1. データベース・オブジェクトをインストールします。
  2. アプリケーションのヘルプ・テキストを作成します。
  3. PL/SQL ライブラリを連結し、必要に応じてヘルプを呼び出すためにキーヘルプ・トリガーにコードを追加します。

- 表 5-7 に示したプラットフォーム固有のメソッドは、組み込まないようにします。

表 5-7 プラットフォーム固有のメソッド

プラットフォーム	メソッド
Windows	■ VBX コントロール ■ OLE コンテナ ■ ActiveX（OCX） ■ DLL （ORA_FFI)
Motif	ロード可能なライブラリのコール（ORA_FFI）

これらのオブジェクトは、サポートされていないプラットフォームではプレースホルダ（空白）のままです。これらのオブジェクトをアプリケーションに組み込む必要がある場合、プレースホルダを非表示にする方法の詳細は、5.2.3.4 項「オブジェクトを隠す」を参照してください。

5.2.2.1 ユーザー・イグジットの挿入

ユーザー・イグジットは自分で記述する 3GL プログラムで、コンパイル時にフォームにリンクされます。ユーザー・イグジットは常にポート固有です。

移植性のあるフォームから 3GL プログラムをコールする前に、プログラムが信頼する情報およびプロセスがすべてのプラットフォームで使用可能であることを確認します。たとえば、Windows レジストリからの情報に依存するプログラムでは、他のプラットフォームでこの情報にアクセスできないため、プログラムを再設計しなければならないか、またはプログラムを完全に破棄することを意味します。

ユーザー・インタフェース・イグジットを介した 3GL プログラムのアクセスではなく、ORA_FFI ビルトイン・パッケージ（Oracle ファンクション・インタフェース）の使用も考慮します。ユーザー・イグジット・インタフェースを使って外部ファンクションにアクセスする場合、ユーザー・イグジットを再リンクするか、または 3GL プログラムを変更するたびに各プラットフォームの DLL を置換しなければなりません。ORA_FFI を使用すると、PL/SQL 言語規則を使用する PL/SQL インタフェースを介して外部ファンクションをコールできるので、プログラムを変更した場合も再リンクする必要はありません。このため、フォームから 3GL プログラムにアクセスする方法として、ORA_FFI が選択されます。

5.2.3 複数のプラットフォームに渡るフォームの開発方針

この項では、移植性のあるフォームを開発するために使用できるテクニックをいくつか紹介します。

- 5.2.3.1 項「シングル・ソースの作成」では、各実行プラットフォームで最大限の機能を提供するシングル・ソースを作成するためのアーキテクチャを説明します。
- 5.2.3.2 項「可視属性のサブクラス化」では、オブジェクト・ライブラリに格納される可視属性を明示的にサブクラス化する重要性について説明します。
- 5.2.3.3 項「get_application_property ビルトインの使用」では、移植性のあるアプリケーションを開発するときの、この Form Builder ビルトインの使用方法を説明します。
- 5.2.3.4 項「オブジェクトを隠す」では、オブジェクトが特定のプラットフォームで有効でないときに表示されるプレースフォルダを削除するためのコードのサンプルが提供されています。

5.2.3.1 シングル・ソースの作成

すべての実行プラットフォームに対して、最低限の共通分母を目的としたシングル・ソースの作成を検討しているときは、この方針は、アプリケーションに備えることのできる美学を厳格に制限します。方針をより効率よくするには、各プラットフォームの本来のルック & フィールでアプリケーションを配布するシングル・ソースを作成します。図に示したアーキテクチャで、これを実現するための方法を説明します。

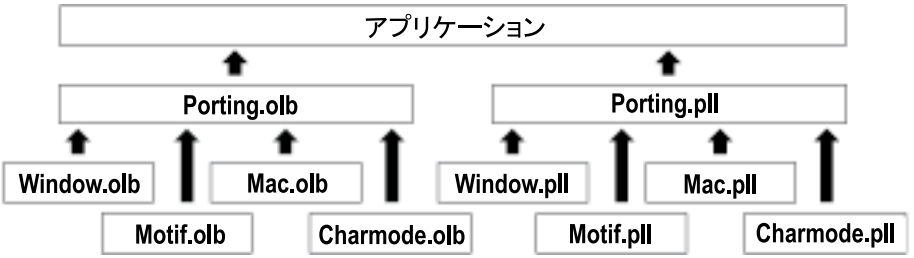


図 5-1 ポート固有のインプリメンテーション

このアーキテクチャでアプリケーションのモデルを作るには、次のようにします。

1. すべての標準とオブジェクトのオブジェクト・ライブラリ（Porting.olb）を作成します。オブジェクト・ライブラリの使用の詳細は、第 2 章の 2.2.2.1 項「オブジェクト・ライブラリの使用」を参照してください。
2. 各実行プラットフォームに個別のオブジェクト・ライブラリ（Window.olb、Motif.olb、Mac.olb、Charmode.olb）を作成します。
3. ポート固有のコードに対して共通ライブラリ（Porting.pll）を作成します。
4. 各プラットフォームのポート固有のコードに対して個別のライブラリ（Window.pll、Motif.pll、Mac.pll、Charmode.pll）を作成します。
5. 各プラットフォームの UI リポジトリ（.OLB）およびライブラリ（.PLL）で、その特定のプラットフォームに理想的な方法でアプリケーション・オブジェクトを処理するコードを開発します。各 UI リポジトリとライブラリで、指定されたオブジェクトに同じ名前を使用します。
6. リポジトリの標準とオブジェクトおよびライブラリのポート固有のコードを参照しながら、アプリケーションを記述します。
7. 特定のプラットフォームのアプリケーションをコンパイルする準備ができたら、UI リポジトリを Porting.olb と Porting.pll にコピーし、コンパイルします。

5.2.3.2 可視属性のサブクラス化

可視属性は、フォームやメニュー・オブジェクトに設定するフォント、カラーおよびパターンなどのプロパティです。フォーム・オブジェクトの可視属性を慎重に定義することで、ユーザーが各プラットフォームでその環境に固有の自然なルック & フィールを享受できることが保証されます。

項目やキャンバスなどの多くの Form Builder オブジェクトでは、その外観を定義するために可視属性が参照されます。可視属性は、それを参照するオブジェクトと同じモジュールで定義される必要があります。

可視属性は、通常オブジェクト・ライブラリに格納されます。各モジュールでこれらの可視属性のサブクラスを作成するようにします。フォームのオブジェクト・ライブラリからオブジェクトをサブクラス化すると、ライブラリ・オブジェクトに対する変更は自動的にフォーム・オブジェクトに適用されます。ただし、これは、ライブラリ・オブジェクトの可視属性への変更には適用されません。このため、可視属性をコピーするのではなく、サブクラス化することによって、可視属性の最新の定義を確実にモジュールに反映できます。

5.2.3.3 get_application_property ビルトインの使用

GET_APPLICATION_PROPERTY ビルトイン・ファンクションでは、アプリケーションの情報が戻されるので、1 つ以上のこれらの変数の設定に基づいて、実行時に動的に反応できます。

- DISPLAY_HEIGHT と DISPLAY_WIDTH 現行の図表の大きさを指定します。単位は、フォームの座標システムの設定方法により異なります。
- OPERATING_SYSTEM アプリケーションを現在実行中のプラットフォームの名前（MSWINDOWS、WIN32COMMON、SunOS、VMS、UNIX または HP-UX）を指定します。
- USER_INTERFACE アプリケーションを現在実行中のユーザー・インタフェース・テクノロジの名前（WEB、MOTIF、MSWINDOWS、MSWINDOWS32、PM、X、VARCHAR2MODE、BLOCKMODE または UNKNOWN）を指定します。

変数の値によって、実行プラットフォームで使用できないオブジェクトを動的に隠し、その空白に他のオブジェクトを再配置し、必要に応じて、利用可能なプラットフォームの標準に合うようにオブジェクトの属性を変更できます。詳細は、5.2.3.4 項「オブジェクトを隠す」を参照してください。

5.2.3.4 オブジェクトを隠す

OLE、VBX、および ActiveX オブジェクトをサポートしていないプラットフォームでプレースホルダが表示されるのを防ぐために、メニューまたはボタンからコールされる個別のウィンドウにこれらのオブジェクトを配置し、メニュー項目やボタンを動的に使用可能または使用不可にできます。または、このコードの断片部分を使用して、ポート固有のオブジェクトを隠したり表示でき、それらの場所に他のオブジェクトを再配置できます。

```
WHEN-NEW-FORM-INSTANCE trigger:
declare
ui  varchar2(15) ;
begin
ui := get_application_property (user_interface);
if ui = 'CHARMODE' or ui = 'MOTIF' then
   set_item_property ('VBXOBJECT1', displayed, property_false);
   set_item_property ('OLEOBJECT1', displayed, property_false);
   set_item_property ('TEXTITEM1', position, 43, 4);
end if;
end;
```

注意 : 項目プロンプトは、対応付けられた項目を隠すと自動的に隠されます。

5.2.4 キャラクタ・モードのフォームの設計

キャラクタ・モードおよびビットマップ環境の両方に対してアプリケーションを作成している場合、シングル・ソースにすることは必ずしも最善のアプローチではありません。最低限の共通分母であるキャラクタ・モードの開発によって、GUI ユーザーからビットマップ制御と対応付けられた使いやすさが奪われます。表 5-8 に示す「ビットマップ専用」は、使用してはならない機能です。

表 5-8 キャラクタ・モードとビットマップ環境

キャラクタ・モード	ビットマップ専用
■ ボックス	■ イメージ
■ 水平線	■ カラー
■ 垂直線	■ 描画
■ ASCII テキスト	■ 楕円
■ 太字テキスト	■ ドリルダウン・ボタン（レポート）
■ 下線	■ イタリック体のテキスト
	■ ビットマップ・パターン
	■ 対角線
	■ マルチメディア・サポート

キャラクタ・モード環境にはこれらの GUI ファンクションを使用不可にする方法がありますが、この作業は非常に時間がかかり、失敗する場合もあります。このため、最初からこれらの本質的にかなり異なるユーザーをサポートすることがわかっている場合、完全に独立した 2 つのアプリケーションを作成することが、全員、つまり開発者とそのユーザーにとって最善の方法です。

Form Builder をできる限りキャラクタ・モードのように**見せる**場合、キャラクタ・モードの開発はより簡単です。表 5-9 にお薦めするプロパティ設定を示しています。

表 5-9 キャラクタ・モードに似たプロパティの設定

プロパティ	推奨事項 / 注意事項
ボイラープレート・フォント	■ WindowsFixedSys、Regular、9 ポイント ■ Motif フォント＝ Fixed、サイズ＝ 12.0、ウェイト＝ Medium、スタイル＝ Plain

表 5-9 キャラクタ・モードに似たプロパティの設定

プロパティ	推奨事項 / 注意事項
座標情報	■ 座標システム : 実際単位[1] ■ 実際単位 : ポイント ■ 文字セル幅 : 6 ■ 文字セル高さ : 14
表示	■ 格子表示 : オン ■ 格子指定 : オン ■ キャンバス表示 : オフ
書式→レイアウト・オプション→定規	■ 単位 : 文字セル ■ 水平文字セル・サイズ : 6 ■ 垂直文字セル・サイズ : 14 ■ 格子間隔 : 1 ■ 格子当りのスナップ・ポイント数 : 2

[1] キャラクタ・モード環境からビットマップ環境へのフォームの移植性の向上フォームをキャラクタ・モードでのみ利用する場合は、文字座標システムを使用します。

キャラクタ・モードのアプリケーションを厳密に開発する場合、次のことを覚えておいてください。

表 5-10 キャラクタ・モード・アプリケーションの推奨事項

項目	推奨事項 / 注意事項
一般	■ すべてを等幅フォントにします。 ■ ウィジェットに現行のフォーカスがない場合でも、各ウィジェットのキーボード等価を作成します。 ■ スクロールは使用しにくい ため避けます。 ■ ユーザーにプレースホルダを見せたくない場合、OLE、VBX および ActiveX オブジェクトを隠します。 ■ すべての UI ウィジェットがキャラクタ・モード等価で表されるので、ウィジェットに全体を表示するのに十分な空白があることを確認します。 ■ ユーザーはマウスで LOV を移動できないので、`set_lov_property` ビルドインを使って動的に LOV を位置付けます。

表 5-10 キャラクタ・モード・アプリケーションの推奨事項

項目	推奨事項 / 注意事項
ナビゲーション	■ ユーザーにはマウスがないので、ユーザーはアプリケーション内からウィンドウまたはフォーム間でナビゲートできません。フォーム間でナビゲートするためのボタンまたはメニューを提供します。 ■ ウィンドウをマウスで再配置できないので、表示されたウィンドウによってそのウィンドウに必要なコンテキストが隠れないことを確認してください。ユーザーがウィンドウを使用した作業を終えたら、プログラムでウィンドウを使用不可にするか、またはウィンドウの「終了時に隠す」プロパティを「はい」にします。
レイアウト	■ 画面には、80 × 24 文字セルしかありません。第 1 行はメニューに使用されます。一番下の最後の 2 行はコンソール表示と、メッセージおよびステータス行です。残ったスペースを十分に利用するように慎重に画面を計画します。 ■ フォントは均等間隔なので、プロポーショナル・フォントより、平均して多くのスペースを消費します。ボイラープレートとテキストウィジェットがセル当り 1 文字で表せるように画面を設計します。
座標システム	■ 文字座標システムを使用して、格子指定をオンにします。これにより、オブジェクトのサイズが確実に文字セル・サイズの倍数単位になります。
メニュー	■ メニューは画面の第 1 行に表示されます。 ■ カット、コピーおよびペーストのような共通のメニュー項目は使用できません。 ■ 画面の第 1 行への複雑なナビゲーションを削減するために、共通して使用されるメニュー項目にホット・キーを定義します。
凹凸	■ 使用できません。
ボタン	■ ボタン・パレットは使用しないでください。かわりに、すべてのアクションをメニューから使用できるようにします。キャラクタ・モードでは、ボタンへナビゲートするときにコンテキストを保持できないので、ボタンはこのモードで正常に動作しません。
アイコン	■ 使用できません。GUI 環境のすべてのアイコン・ボタンがメニュー・オプションによっても表されることを確認します。 ■ アイコン・ボタンは、ボタンのラベルと一緒にしか表示されません。ラベルに意味があり、それらを表示するために十分なスペースがあることを確認します。
カラー	■ マイナス記号が前に付いた負数のカラーは使用できません。 ■ モノクローム表示では、カラーは黒または白に指定されます。バックグラウンドとフォアグラウンドの両方が黒に指定されるので、どちらにも暗い色は使用しないでください。

5.3 移植性のあるレポートの設計

複数のプラットフォームで実行するレポートを準備する場合、次のことを考慮します。

- **フォント :** すべてのターゲット GUI で、すべてのフォント・タイプ、スタイルおよびサイズが使用できるわけではありません。次のどちらかの方法で、処理できます。
  - ターゲット GUI に存在していることがわかっているフォントを使用するか、ターゲット GUI のデフォルトのフォントにうまくマップできるフォントを使用します。
  - フォント・マッピング・ファイル（UIFONT.ALI）を変更して、フォントが正しくマップされることを確認します。UIFONT.ALI ファイルの使用方法の詳細は、5.2.1.4.1 項「フォント・エイリアスの定義」を参照してください。

  注意 : 画面のフォントとプリンタ・フォントの基準は常に同じとは限らないので、レポートを印刷すると、画面に表示したものと同じように見えない場合があります。特に、テキスト・フィールドは印刷ページで拡張され、隣接するフィールドがシフトする原因となり、場合によっては予想していなかったページ・ブレークが新しく作成されることになります。これを避けるには、拡張専用のフィールドを使い、各フィールドが論理的に可能な最大フォントを収められるくらい十分な大きさであることを確認します。

- **カラー :** 可能ならば、ターゲット GUI に存在していることがわかっているカラーを使用します。わからなければ、ターゲット GUI のデフォルトのカラーにうまくマップできるカラーを使用します。青、マゼンタ、赤、シアン、緑および黄色は、通常多くのプラットフォームで使用可能です。移植可能なレポートにカラーを組み込む場合のいくつかの推奨事項は、5.2.1.3 項「カラーの使用」を参照してください。

- **DPI:** モニターで使用されているインチ当たりのドット数（DPI）は、レポートを実行する人が使用する DPI と異なる場合があります。DPI は、画面上での英数字をワードラップする方法に影響するのみです。プレビューア表示で表示される可能性があるレポートを設計する場合、それを実行する人と同じ DPI を使用するようにします。

- **ボタン :** レポートでボタンを表示する場合、プレビューアを介してレポートを表示するユーザーは、ボタンを押してマルチメディア・オブジェクト（サウンド、ビデオ、イメージ）を表示したり、他のレポートへのドリルダウンなど、PL/SQL コードを介してアクションを実行できます。移植可能なボタンの作成のガイドラインは、5.2.1.6 項「ボタンの使用」を参照してください。キャラクタ・モードでボタンを含むレポートを実行する場合は、ボタンは単純に無視されることに注意してください。それらによってプレースホルダは作成されません。

5.3.1 キャラクタ・モード環境のレポートの設計

キャラクタ・モードのレポートは、ユーザーがレポート出力を掲示板やスプレッドシート、ダンプ・ファイル、文字専用プリンタに送る必要がある環境で必要になることがよくあります。キャラクタ・モードの出力にも、多くの利点があります。

- **移植性 :** キャラクタ・モードのレポートは、厳密には ASCII または EBCDIC ファイルなので、どこにでも印刷またはエクスポートできます。

- **プリンタ資源の活用 :** キャラクタ・モードのレポートには、複雑なポストスクリプト出力と異なり、特別なフォーマットは必要ないため、古いプリンタを活用できます。
- **プリンタ・コードのサポート :** Reports Developer ではプリンタのエスケープ・コードがサポートされているので、実行時にユーザーが特別なフォント・サイズやハイライト表示など、プリンタ固有の機能を利用できます。プリンタ定義ファイルとプリンタ・コードの詳細は、Report Builder のオンライン・ヘルプを参照してください。
- **パフォーマンス :** キャラクタ・モードのレポートは、等価のビットマップのレポートより高速に実行できます。ビットマップのレポートは、通常フォーマットに時間がかかり、出力ファイル（Postscript、PCL5）が大きくなります。

5.3.1.1 設計の検討事項

ビットマップ環境用に作成されたレポートは、簡単にはキャラクタ・モードに変更できません。キャラクタ・モード環境でレポートを実行する必要があるとわかっている場合、キャラクタ・モードのレポートとしてレポートを作成するのが最善の方法です。ただし、ビットマップ・レポートをキャラクタ・モードに変換する必要がある場合は、Report Builder のオンライン・ヘルプの ASCII レポートの作成を参照してください。これには、キャラクタ・モード・レポートを作成するための段階的な手順も含まれています。

5.4 移植性のある図表の設計

スタンドアロンの図形（フォームやレポートなどのコンテナ・アプリケーションの一部でない図形）がある場合、移植はかなり簡単です。ただし、ほとんどの図形はフォームやレポートに埋め込まれているので、異なるプラットフォームに移植すると問題が生じます。複数の環境の図形を開発する場合、次のガイドラインに従います。

- 特に図形がフォームやレポートに埋め込まれるときに、テキスト・スケールが均一であることを確認するために、スケーラブル TRUETYPE フォントを使い、**チャート・ラベル以外**のすべてのテキスト・オブジェクトに対して、「スケーラブル・フォント」フラグを「True」に設定します。「スケーラブル・フォント」フラグは、チャートのテキスト・ラベルには使用できません。チャートを更新するとすぐに、フォントは元のサイズにリセットされます。このため、チャート・サイズの最大の範囲で読みやすいチャート・ラベルのフォントとサイズを選択します。8 ～ 10 ポイント・サイズできれいに表示される、小さく鮮明なフォントを選択するようにします。これより大きくすると、フォームやレポートの狭いチャート領域に埋め込んだときにチャートが読めなくなります。
- タイマーおよびドラッグ・アンド・ドロップのコードは、スタンドアロンの Graphics アプリケーションでのみサポートされています。フォームやレポートにこれらの機能を挿入しても無視されます。
- カラーの使用は、ニーモニック名（red、green、blue、yellow、magenta、cyan、black、white、gray、darkgray、darkyellow、darkcyan、darkmagenta、darkblue、darkgreen および darkred）と同様に、Designer（パレットの左上隅）で使用可能な、中心となる 16 色に制限します。

- PL/SQL でなくレイアウト・エディタを使ってカラーを設定します。レイアウト・エディタで選択したカラーは、自動的に使用可能な近似カラーに調整されます。コードでカラーを設定すると、カラーがシステムの現行の解像度で使用できない場合、エラーが発生します。
- アプリケーション・プロパティ `og_get_ap_platform` のコールと、ビルドイン・サブプログラム `og_append_directory` および `og_append_file` のコールによって、プラットフォーム依存のコードを分離します。詳細は、Graphics Builder のオンライン・ヘルプを参照してください。

6

オープン・アーキテクチャの利用

この章では、Forms Developer および Reports Developer で利用できる、オープンで拡張可能な環境開発を利用する上で役立つガイドラインを提供します。

項	説明
6.1 項「OLE オブジェクトと ActiveX コントロールの処理」	サポートするコンポーネント・テクノロジの内容を説明し、OLE オブジェクトおよび ActiveX コントロールをはじめとするアプリケーションを作成するための手順およびガイドラインを提供します。
6.2 項「外部ファンクションを使用したアプリケーションのカスタマイズ」	3GL 外部ファンクションを使用してアプリケーションのカスタマイズと、拡張を行う方法を説明します。
6.3 項「Form Builder アプリケーションの作成と修正を行うためのオープン API の使用」	オープン API の概要を説明し、さらに、オープン API を使用して Form Builder アプリケーションの作成と変更を行う方法を説明します。
6.4 項「ODBC データソースに対して実行するアプリケーションの設計」	ODBC サポートの内容を説明し、さらに、ODBC に対してアプリケーションを実行するための詳細な手順およびガイドラインを提供します。

6.1 OLE オブジェクトと ActiveX コントロールの処理

この項では OLE および ActiveX の内容を説明し、さらにこのテクノロジの利用方法を説明します。この項では、次の項目について説明します。

- 6.1.1 項「OLE」
  - 6.1.1.1 項「OLE を利用する場合」
  - 6.1.1.9 項「アプリケーションへの OLE オブジェクトの追加」
  - 6.1.1.10 項「OLE オブジェクトの操作」

注意 : OLE および ActiveX のサポートは、Windows プラットフォームのみに制限されています。

6.1.1 OLE

Object Linking and Embedding (OLE) は Microsoft の標準機能で、様々なソフトウェア・コンポーネントを 1 つのアプリケーションに統合して再使用できます。

たとえば、アプリケーションを Microsoft Excel 文書と統合すると、Forms Developer（または Reports Developer）および Microsoft Excel の機能を提供できるようになります。ユーザーは、Forms Developer または Reports Developer の機能を使用してデータベースのデータの表示や操作を行う一方で、Microsoft Excel で提供されるテキスト処理機能を使用して文書の書式を整えることができます。

OLE オブジェクトをアプリケーションに取り込めば、専用のコンポーネントの様々なグループをシームレスに統合して完全なアプリケーションを作成できます。最初から新規にアプリケーション全体を作成する必要はありません。アプリケーションを短時間に、低コストで配布できます。

6.1.1.1 OLE を利用する場合

OLE は次のような場合に使用します。

- 既存の OLE 準拠アプリケーションをアプリケーションで利用する場合。

  たとえば、アプリケーションの機能をワープロ文書、スプレッドシート文書、ノブ・コントロール、ビデオ・クリップ、サウンドで拡張できます。

- アプリケーション・ユーザーに使い慣れているインタフェースを提供する場合。

  Microsoft Windows では、ほとんどのユーザーが Microsoft Word や Microsoft Excel に慣れています。ワープロやスプレッドシートの機能をアプリケーションに作成するのではなく、Word や Excel の文書をアプリケーションに埋め込むことができます。

- アプリケーションを主に Windows プラットフォームで利用する場合。

6.1.1.2 OLE サーバーと OLE コンテナについて

OLE の基本概念は、クライアントとサーバーです。クライアントとは、別のアプリケーションのサービスを要求し利用するアプリケーションです。サーバーとは、このようなサービスを提供するアプリケーションです。

- **OLE サーバー・アプリケーション**

  OLE サーバー・アプリケーションでは、OLE コンテナに埋め込まれたり、またはリンクされた OLE オブジェクトが作成されます。サーバー・アプリケーションでは、OLE オブジェクトの作成、保存および操作が行われます。たとえば、サーバーでは、オブジェクトの一部がはげてしまった場合に、そのオブジェクトをどう再ペイントするかが決定されます。

  Graphics Builder と Microsoft Word は、OLE サーバーの例です。

- **OLE コンテナ・アプリケーション**

  OLE サーバー・アプリケーションとは異なり、OLE コンテナ・アプリケーションでは埋込みまたはリンク用の文書は作成されません。かわりに、OLE コンテナ・アプリケーションでは、OLE サーバー・アプリケーションによって作成されたオブジェクトを格納および表示するための場所が提供されます。

  Form Builder と Report Builder は、OLE コンテナ・アプリケーションの一例です。

6.1.1.3 埋込みオブジェクトとリンク・オブジェクトについて

OLE オブジェクトをアプリケーションにリンクまたは埋め込むことができます。

- **埋込みオブジェクト**　埋込みオブジェクトには、表示データと、アプリケーションに保存されているネイティブ・データまたはデータベースに LONG RAW 列として保存されているネイティブ・データが含まれています。
- **リンク・オブジェクト**　リンク・オブジェクトには、表示情報とネイティブ・データへの参照しか含まれていません。リンク・オブジェクトの内容は、アプリケーションに保存されたり、またはデータベースの LONG RAW 列として保存されるのではなく、別個のリンク・ファイルとして保存されます。

リンクと埋込みの間には機能的違いはありません。OLE コンテナは、リンク・オブジェクトの場合であっても、埋込みオブジェクトの場合であっても、同じコードを実行していずれも同様に処理します。唯一の違いは、OLE オブジェクトを埋め込むとアプリケーションのサイズが増えることです。この結果として考慮するべき点はパフォーマンス（特にファイル・サーバー）への影響です。アプリケーションのサイズが大きくなるほど、オープンやメモリーへのロードに時間がかかります。

6.1.1.4 レジストレーション・データベースについて

各クライアント・マシンには OLE レジストレーション・データベースが含まれています。レジストレーション・データベースには、OLE オブジェクトを分類する一連のクラスが保存

されています。レジストレーション・データベースに含まれている情報により、OLE コンテナにおける埋込みやリンクに使用可能なオブジェクト・クラスが決定されます。

OLE サーバー・アプリケーションでは、レジストレーション・データベースのメンバーとなる一連のクラスがエクスポートされます。各コンピュータには、レジストレーション・データベースが 1 つあります。OLE サーバー・アプリケーションのインストール時にレジストレーション・データベースが存在しない場合は、レジストレーション・データベースが 1 つ作成されます。

1 つの OLE サーバー・アプリケーションによって、レジストレーション・データベースに多くの OLE クラスを追加できます。レジストレーション・データベースにクラスを追加するプロセスは、OLE サーバー・アプリケーションのインストールの間に透過的に行われます。たとえば、Microsoft Excel をインストールする場合、複数のクラスがレジストレーション・データベースに追加されます。レジストレーション・データベースにインストールされるクラスには、Excel Application、Excel Application5、Excel Chart、Excel Sheet、ExcelMacrosheet、ExcelWorkSheet などがあります。

6.1.1.5 OLE アクティブ化スタイルについて

OLE オブジェクトをアクティブ化することにより、OLE サーバー・アプリケーションの機能にアクセスできます。OLE オブジェクトをアクティブ化する方法には、内部アクティブ化と外部アクティブ化の 2 つがあります。

- **内部アクティブ化 :** 内部アクティブ化により、ユーザーは、別のウィンドウに切り替えることなく、OLE オブジェクトをアプリケーションの内部で操作できます。

  内部アクティブ化時には、境界線が引かれ、アクティブ化されたオブジェクトはその中に表示されます。アクティブ化されたオブジェクトのツールバー、メニュー、およびその他のコントロールは一時的に標準のメニュー・オプションと入れ替わります。入れ替わったメニュー・オプションやツールバーを使用して、OLE サーバー・アプリケーションで使用可能な機能にアクセスできます。標準のメニュー・オプションやツールバーは、内部アクティブ化を解除すると再表示されます。内部アクティブ化を解除するには、境界線の外側の任意の場所をクリックします。

  注意 : 内部アクティブ化は埋込みオブジェクトに使用できますが、リンク・オブジェクトには利用できません。

- **外部アクティブ化 :** 外部アクティブ化により、ユーザーは OLE オブジェクトを別ウィンドウで操作できます。OLE オブジェクトがアクティブ化されるときに、オブジェクトの OLE サーバー・アプリケーションが起動され、OLE オブジェクトが別の OLE サーバー・アプリケーション・ウィンドウに表示されます。別のウィンドウには、OLE サーバー・アプリケーションのメニュー・オプションとツールバーがあります。外部アクティブ化を解除するには、OLE サーバー・アプリケーションを明示的に終了する必要があります。

  外部アクティブ化は、埋込みオブジェクトおよびリンク・オブジェクトに利用できます。

リンクされたソース・ファイルの内容が外部アクティブ化により変更された場合、リンクされたオブジェクトは手動で、または自動的に更新できます。手動で更新する場合には、リンクされたソース・ファイルの変更がオブジェクトに反映されるように明示的に指示する必要があります。自動更新は、リンクされたソース・ファイルの変更後にただちに行われます。

注意 : 内部アクティブ化も外部アクティブ化も共に、OLE コンテナの OLE アクティブ化プロパティの設定に基づいて行われます。OLE サーバー・アプリケーションにアクセスできる場合、埋込み OLE オブジェクトがアクティブ化されたときに内部アクティブ化と外部アクティブ化のどちらが行われるかは、OLE コンテナのアクティブ化プロパティの設定値により決まります。リンク・オブジェクトの場合は、外部アクティブ化によりアクティブ化されます。内部アクティブ化プロパティが「はい」に設定されていたとしても、内部アクティブ化はリンク・オブジェクトには適用されません。

6.1.1.6 OLE オートメーションについて

OLE オブジェクト内のデータと対話したり、OLE オブジェクト内のデータを操作したいと考えることがあります。その場合に、PL/SQL および OLE オートメーションを使います。

OLE オートメーションにより、サーバー・アプリケーションで、OLE コンテナ・アプリケーションから起動できる一連のコマンドやファンクションを提供できます。これらのコマンドやファンクションを使用すれば、OLE コンテナ環境から OLE オブジェクトを操作できます。

Forms Developer および Reports Developer では、OLE サーバー・アプリケーションにより提供されるコマンドやファンクションには、PL/SQL を使用してアクセスします。OLE コマンドやファンクションの作成、操作、およびアクセスを行うための PL/SQL アプリケーション・プログラミング・インタフェースは、ビルトインで提供されます。

注意 : OLE コンテナ・アプリケーションの OLE オブジェクトの操作に利用できるオプションの多くは、OLE サーバー・アプリケーションによって決まります。たとえば、OLE ポップアップ・メニューのオプション（別名 OLE 動詞）は、OLE サーバー・アプリケーションにより提供されます。オブジェクト・クラスなどの、レジストレーション・データベースに含まれている情報も、OLE サーバー・アプリケーションによって決まります。

6.1.1.7 OLE サポート

Forms Developer および Reports Developer は、OLE オートメーションに加え、OLE サーバー・サポートおよび OLE コンテナ・サポートも提供します。

コンポーネント	コンテナ	サーバー・アプリケーション	OLE2 オートメーション
Form Builder	あり	なし	あり
Graphics Builder	なし	あり	あり
Procedure Builder	なし	なし	なし

コンポーネント	コンテナ	サーバー・アプリケーション	OLE2 オートメーション
Project Builder	なし	なし	なし
Query Builder	なし	なし	なし
Report Builder	あり	なし	あり
Schema Builder	なし	なし	なし

6.1.1.7.1 OLE コンテナ・サポート

OLE コンテナ・アプリケーションとして、Form Builder および Report Builder では次のことをサポートしています。

- OLE コンテナへの OLE サーバー・オブジェクトの埋込みとリンク。
- OLE コンテナに埋め込まれた内容の内部アクティブ化（Form Builder のみ）。

  Form Builder では、内部アクティブ化により、OLE サーバー・アプリケーションからメニューやツールバーにアクセスして埋め込まれた OLE オブジェクトを編集できます。
- OLE オートメーション・サポートにより、PL/SQL から OLE オブジェクト、プロパティおよびメソッドへプログラム上のアクセスを行う。

  PL/SQL を使って、OLE サーバーにより提供されるコマンドやファンクションを呼び出すことができます。
- OLE オブジェクトをデータベース内の LONG RAW 列にシームレスに保存します。

  OLE オブジェクトをデータベースに保存したり、OLE オブジェクトをデータベースから問い合せたりできます。リンク・オブジェクトを保存した場合には、イメージとリンク情報のみがデータベースに残ります。リンク・オブジェクトの内容は、リンクされたソース・ファイルに残ります。埋込みオブジェクトを保存した場合は、埋込みオブジェクトのすべての内容がデータベースに残ります。

6.1.1.7.2 OLE サーバー・サポート

Graphics Builder は OLE サーバー・アプリケーションです。Graphics Builder の図表を Forms Developer アプリケーションや Reports Developer アプリケーションに埋め込んだり、リンクしたりできます。

推奨事項 : Graphics Builder 図表をアプリケーションに追加する場合は、OLE オブジェクトとしての埋込みやリンクは行わないでください。図表は、チャート・ウィザードを使用してアプリケーションに追加してください。

6.1.1.7.3 OLE コンテナ・プロパティ

OLE コンテナ・プロパティにより、OLE 表示属性、OLE コンテナとサーバーとの対話、コンテナの保存などが決まります。

注意 : コンテナ・プロパティの他に、OLE オブジェクト・プロパティも設定できます。各 OLE オブジェクトには複数のプロパティがあります。OLE オブジェクトのプロパティにアクセスするには、マウスの右ボタンをクリックしてポップアップ・メニューを表示します。

この項では、Forms Developer および Reports Developer でサポートしている OLE コンテナ・プロパティを示します。

コンポーネント	プロパティ	説明
Form Builder	■ OLE アクティブ・スタイル	OLE コンテナ項目をアクティブ化するイベントを指定します。ダブルクリック、フォーカスインまたは手動があります。
	■ OLE クラス	OLE コンテナに常駐できる OLE オブジェクトのクラスを決定します。
	■ OLE 同一ウィンドウ・アクティブ	埋め込み OLE オブジェクトの編集に OLE 内部アクティブ化を使うかどうかを指定します。
	■ OLE の詳細サポート	埋め込みオブジェクトの OLE サーバーで内部アクティブ化の際にインサイド・アウト・オブジェクト・サポートを使用可能にするかどうかを指定します。インサイド・アウト・アクティブ化により、複数の埋め込みオブジェクトに対して、OLE コンテナ内にアクティブな編集ウィンドウを持たせることができます。
	■ OLE ポップアップ・メニュー項目	マウス・カーソルを OLE オブジェクト上に置いてマウスの右ボタンを押したときに、どの OLE ポップアップ・メニューが表示され、使用可能になるかを決定します。OLE ポップアップ・メニュー・コマンドによって OLE オブジェクトを操作します。
	■ OLE サイズ変更スタイル	OLE コンテナにおける OLE オブジェクトの表示形式を決定します。
	■ OLE テナント形態	OLE コンテナにおける OLE オブジェクトの表示方法を決定します。内容、アイコンまたはサムネイルがあります。

コンポーネント	プロパティ	説明
	■ OLE テナント・タイプ	OLE コンテナのテナントとなる OLE オブジェクトのタイプを決定します。埋込み、リンク、任意、静的またはなしがあります。
	■ OLE ポップアップ・メニュー表示	マウスの右ボタンをクリックしたときに OLE オブジェクトと対話するためのコマンド・ポップアップ・メニューを表示するかどうかを決定します。
	■ OLE テナント・タイプ表示	OLE オブジェクト・タイプを定義する境界線が OLE コンテナを囲むかどうかを決定します。
Report Builder	新規作成	OLE オブジェクトをレポート・アプリケーションに埋め込むことを指定します。
	ファイルから作成	OLE オブジェクトをレポート・アプリケーションにリンクすることを指定します。
	アイコンとして表示	OLE オブジェクトがアイコンとして表示されるかどうかを指定します。デフォルトでは、OLE オブジェクトは空のコンテナとして表示されます。

6.1.1.7.4 OLE/ActiveX ビルトイン

この項では、各コンポーネントでサポートされている OLE および ActiveX ビルトインをリストします。

コンポーネント	ビルトイン	説明
Form Builder	■ ACTIVATE_SERVER	OLE コンテナと対応付けられている OLE サーバーをアクティブ化して、OLE サーバーが OLE オートメーション・イベントを OLE コンテナから受け取ることができるようにします。
	■ ADD_OLEARGS	OLE オブジェクトのメソッドに渡される引数の型および値を設定します。

コンポーネント	ビルトイン	説明
	■ CALL_OLE	指定された OLE オブジェクトのメソッドにコントロールを渡します。
	■ CALL_OLE_<return type>	指定された OLE オブジェクトのメソッドにコントロールを渡します。指定した型の戻り値を受け取ります。 CHAR、NUM、OBJ、RAW、VAR のそれぞれの引数型に対応する 5 つのバージョンのファンクションが用意されています（戻り時のデータ型によって分けられます）。
	■ CLOSE_SERVER	OLE コンテナに対応付けられている OLE サーバーを非アクティブ化します。OLE サーバーと OLE コンテナの間の接続を終了します。
	■ CREATE_OLEOBJ	最初のフォームにおいて OLE オブジェクトを作成し、オブジェクトの永続性を確立します。2 番目のフォームにおいて、以前にインスタンス化された OLE オブジェクトの永続性を変更します。
	■ CREATE_VAR	空の名前未定可変レコードを作成します。 このファンクションには 2 つのバージョンがあります。1 つはスカラー、もう 1 つは配列です。
	■ DESTROY_VARIANT	CREATE_VAR ファンクションにより作成された可変レコードを破棄します。
	■ EXEC_VERB	OLE サーバーで、動詞名または動詞索引により指定された動詞を実行します。OLE 動詞により、OLE オブジェクトで実行できるアクションを指定します。
	■ FIND_OLE_VERB	OLE 動詞索引を戻します。OLE 動詞により、OLE オブジェクトで実行できるアクションが指定されます。各 OLE 動詞には対応する OLE 動詞索引があります。

コンポーネント	ビルトイン	説明
	■ GET_INTERFACE_POINTER	ハンドルを OLE2 オートメーション・オブジェクトに戻します。
	■ GET_OLEARG_<type>	OLE 引数スタックから n 番目の引数を取得します。 CHAR、NUM、OBJ、RAW、VAR のそれぞれの引数型に対応する 5 つのバージョンのファンクションが用意されています（データ型の値によって分けられます）。
	■ GET_OLE_MEMBERID	OLE オブジェクトの名前付きメソッドまたはプロパティのメンバー ID を取得します。
	■ GET_VAR_BOUNDS	OLE 可変レコードの配列の範囲を取得します。
	■ GET_VAR_DIMS	OLE 可変レコードが配列であるかどうかを判別します。配列である場合は、その配列が何次配列であるかを取得します。
	■ GET_VAR_TYPE	OLE 可変レコードの型を取得します。
	■ GET_VERB_COUNT	OLE サーバーで認識される動詞の数を戻します。OLE 動詞により、OLE オブジェクトで実行できるアクションが指定されます。使用できる動詞の数は OLE サーバーによって決まります。
	■ GET_VERB_NAME	指定された動詞索引と対応付けられている動詞名を戻します。
	■ INITIALIZE_CONTAINER	サーバー互換ファイルからの OLE オブジェクトを OLE コンテナに挿入します。
	■ INIT_OLE_ARGS	定義されて OLE オブジェクトのメソッドに渡される引数の数を設定します。
	■ LAST_OLE_ERROR	最新の OLE エラー条件を識別する番号を戻します。

コンポーネント	ビルトイン	説明
	■ LAST_OLE_EXCEPTION	コールされたオブジェクトで発生した最新の OLE 例外を識別する番号を戻します。
	■ OLEVAR_EMPTY	型 VT_EMPTY の OLE 可変レコード
	■ PTR_TO_VAR	まず、指定されたアドレスを含む型 VT_PTR の OLE 可変レコードを作成します。次に、ファンクション VARPTR_TO_VAR によりその可変レコードと型を渡します。
	■ RELEASE_OBJ	OLE オブジェクトへの接続を停止します。
	■ SERVER_ACTIVE	指定されたコンテナと対応付けられているサーバーが実行中であるかどうかを示します。
	■ SET_OLE	OLE プロパティの値を変更します。 新規値のデータ型によって、NUMBER、VARCHAR、OLEVAR の 3 種類のバージョンのプロシージャがあります。
	■ SET_VAR	新しく作成された OLE 可変レコードをその初期値に設定します。または、既存の OLE 可変レコードを新規値にリセットします。 新規値のデータ型によって CHAR、NUMBER、OLEVAR、表の 4 種類のバージョンのプロシージャがあります。
	■ TABLE_FROM_BLOCK	表にブロックからデータを移入します。
	■ TO_VARIANT	OLE 可変レコードを作成し、それを値に割り当てます。 このファンクションには 4 つのバージョンがあります。
	■ VARPTR_TO_VAR	可変レコードのポインタを単純な可変レコードに変更します。
	■ VAR_TO_TABLE	OLE 配列可変レコードを読み込み、PL/SQL 表にデータを移入します。

コンポーネント	ビルトイン	説明
	■ VAR_TO_<type>	OLE 可変レコードを読み込み、その値を等価の PL/SQL 型に変換します。 CHAR、NUM、OBJ、RAW、TABLE、VARPTR のそれぞれの引数型に対応する 6 つのバージョンのファンクションが用意されています（データ型の値によって分けられます）。
	■ VAR_TO_VARPTR	既存の可変レコードを指し示す OLE 可変レコードを作成します。
Developer OLE2 パッケージ	■ ADD_ARG	指定された引数リストに引数を追加します。
	■ CREATE_ARGLIST	OLE サーバーに渡す引数リストを作成します。
	■ CREATE_OBJ	新しく作成された OLE オブジェクトにハンドルを戻します。これは通常、スペルチェッカーなどのユーザー・インタフェースのない OLE オブジェクトに使われます。
	■ DESTROY_ARGLIST	指定した引数リストを破棄します。
	■ GET_CHAR_PROPERTY	OLE オブジェクトの文字プロパティを戻します。
	■ GET_NUM_PROPERTY	OLE オブジェクトの数値プロパティを戻します。
	■ GET_OBJ_PROPERTY	OLE オブジェクトのオブジェクト・タイプ・プロパティを戻します。
	■ INVOKE	指定した OLE サーバー・プロシージャを実行します。
	■ INVOKE_CHAR	指定した OLE サーバー・ファンクションを実行します。このファンクションは、文字値を戻します。
	■ INVOKE_NUM	指定した OLE サーバー・ファンクションを実行します。このファンクションは、数値を戻します。

コンポーネント	ビルトイン	説明
	■ INVOKE_OBJ	指定した OLE サーバー・ファンクションを実行します。このファンクションは、オブジェクト・タイプ値を戻します。
	■ LAST_EXCEPTION	OLE エラーを戻します。
	■ SET_PROPERTY	OLE プロパティを指定した値に設定します。
	■ RELEASE_OBJ	指定した OLE オブジェクトのすべてリソースの割当てを解除します。

6.1.1.8 OLE ガイドライン

OLE オブジェクトの処理時は、次のガイドラインを考慮してください。

項目	推奨事項
OLE オブジェクトの埋込みまたはリンク	次の場合には、OLE オブジェクトをリンクします。 ■ ユーザーが OLE サーバー環境において OLE オブジェクトを処理する場合（ユーザーが外部アクティブ化を行いたいと考えている）。たとえば、作成したアプリケーションではなく、Microsoft Excel を使った方がスプレッドシートの編集を快適に行える場合に、OLE オブジェクトをリンクします。 ■ OLE オブジェクトを複数のアプリケーションで使用する場合。 ■ アプリケーションのサイズが気になる場合。 次の場合には、OLE オブジェクトを埋め込みます。 ■ ユーザーが、アプリケーションを使って OLE オブジェクトを処理できる場合。ユーザーが内部アクティブ化を行いたいと考えている場合。 ■ 複数の OLE ソース・ファイルを持つアプリケーションを維持するのではなく、1 つのアプリケーションを維持する場合。 ■ アプリケーションのサイズを気にしていない場合。
OLE アクティブ・スタイル	外部アクティブ化を使用します。リンクされたオブジェクトをアクティブ化するには、外部アクティブ化による場合のみです。
表示形式	パフォーマンスを最適化する場合には、OLE オブジェクトの「表示形式」プロパティを「アイコン」に設定します。

項目	推奨事項
OLE オブジェクトの作成は設計時または実行時のどちらで行いますか？	OLE オブジェクトは設計時に作成します。 OLE コンテナを Form Builder で作成する場合、OLE コンテナは Forms Builder により自動的に初期化されます。 対照的に、OLE オブジェクトを実行時に挿入する場合は、OLE オブジェクトを手動で初期化する必要があります。 **注意 :** OLE オブジェクトを Forms Runtime の稼動時に手動で挿入すると、OLE オブジェクトは、次のレコード問合せまで、OLE コンテナに表示されます。後続のレコード問合せがあると、OLE コンテナは Form Builder で定義された状態で、またはデータベースから OLE オブジェクトが移入された状態で表示されます。
移植性	OLE オブジェクトがサポートされるのは、Microsoft Windows のみです。移植性が問題になる場合は、OLE オブジェクトをアプリケーションに取り込まないでください。Forms Developer（または Reports Developer）で機能を開発するか、あるいは 3GL 外部ファンクションを開発することを検討してください。
Report Builder での OLE プロパティの設定	Report Builder の OLE コンテナ・プロパティは、「OLE オブジェクトの作成」ダイアログでしか利用できません。すなわち、Report Builder のプロパティ・パレットには、OLE コンテナ・プロパティはありません。Report Builder で作業を行っている場合は、OLE プロパティを「OLE オブジェクトの作成」ダイアログで設定します。

6.1.1.9 アプリケーションへの OLE オブジェクトの追加

OLE オブジェクトをアプリケーションに追加するための詳細な手順は、オンライン・ヘルプを参照してください。

6.1.1.10 OLE オブジェクトの操作

OLE サーバー・アプリケーションは、OLE オブジェクトのプログラム上の操作を可能にする一連のコマンドを提供します。

OLE オブジェクトの操作方法は次のとおりです。

- OLE プロパティの取得と設定
- 特別なコマンドを実行するための OLE メソッドのコール

  注意 : OLE メソッドをコールする前に、まず、OLE オブジェクトのメソッドとプロパティを Forms Developer または Reports Developer にインポートする必要があります。OLE メソッドおよびプロパティをインポートすると、ネイティブ環境において OLE オブジェクトと対話できるようになります。

アプリケーションから OLE メソッドにアクセスするには、STANDARD（Form Builder のみ）および OLE2 ビルトイン・パッケージから行います。

6.1.1.11 OLE の例

この項では、OLE のスタート・ガイドとしていくつかの例を紹介します。

6.1.1.11.1 例 1: バインド変数構文を使用した OLE プロパティの設定

フォーム・アプリケーションにおいて、:item('item_name').ocx.server_name.property バインド変数構文を使用して、プロパティ値を割り当てるか、または取り出すことができます。

例 :

```
:item('OLEitem').OCX.SpreadSheet.CellForeGroundColor:=
:item('OLEitem').OCX.SpreadSheet.CellForeGroundColor + 1;
```

OLEitem は項目名、SpreadSheet は OLE コントロール・サーバー名、CellForeGroundColor はプロパティ名です。

6.1.1.11.2 例 2: プロパティ・アセサーを使用した OLE プロパティの設定

フォーム・アプリケーションにおいて、プロパティ・アセサー・ファンクションおよびプロシージャを使用して、プロパティ値を取得、設定できます。

例 :

```
Variant OleVar;
Variant := EXCEL_WORKSHEET.ole_Range(:CTRL.interface,
  To_variant('A1'));
```

EXCEL_WORKSHEET は、OLE インポータから作成されたプログラム単位の名前です。OLE_RANGE は、プロパティ・アセサー名です。

6.1.1.11.3 例 3: Excel スプレッドシートのセルの変更

この例では、Excel スプレッドシートにおいてセル値を取得、設定します。

```
PACKAGE spreadsheet IS
procedure setcell(trow number, col number, val number);
function getcell(trow number, col number) return number;
END;

PACKAGE BODY spreadsheet IS
obj_hnd ole2.obj_type;/* store the object handle */
FUNCTION get_object_handle return ole2.obj_type IS
BEGIN
  /* If the server is not active, activate the server
     and get the object handle.
  */
```

```
  if not forms_ole.server_active ('spreadsheet') then
    forms_ole.activate_server('spreadsheet');
    obj_hnd := forms_ole.get_interface_pointer('spreadsheet');
  end if;
  return obj_hnd;
END;
 /*
Excel cells are accessed with the following syntax in Visual Basic:
ActiveSheet.Cells(row, column).Value
In PL/SQL, we need to first get the active sheet using the
forms_ole.get_interface_pointer built-in.  We can then use that to call the
Cells method with the row and column in an argument list to get a handle to
that specific cell.  Lastly, we access the value of that cell.
*/

PROCEDURE SETCELL (trow number, col number, val number) IS
  d ole2.obj_type;
  c ole2.obj_type;
  n number;
  lst ole2.list_type;
BEGIN
  /* Activate the server and get the object handle
     to the spreadsheet.
  */
  d := get_object_handle;
  /* Create an argument list and insert the specified
     row and column into the argument list.
  */
  lst := ole2.create_arglist;
  ole2.add_arg(lst,trow);
  ole2.add_arg(lst,col);
  /* Call the Cells method to get a handle to the
     specified cell.
  */
  c := ole2.invoke_obj(d,'Cells',lst);
  /* Set the value of that cell. */
  ole2.set_property(c,'Value',val);
  /* Destroy the argument list and the cell object
     handle.
  */
  ole2.destroy_arglist(lst);
  ole2.release_obj(c);
END;
```

```
 FUNCTION GETCELL(trow number, col number) return number IS
  c ole2.obj_type;
  d ole2.obj_type;
  n number;
  lst ole2.list_type;
BEGIN
  /* Activate the server and get the object handle
     to the spreadsheet.
  */
  d := get_object_handle;
  /* Create an argument list and insert the specified
     row and column into the argument list.
  */
  lst := ole2.create_arglist;
  ole2.add_arg(lst,trow);
  ole2.add_arg(lst,col);
  /* Call the Cells method to get the value in the
     specified cell.
  */
  c := ole2.invoke_obj (d,'Cells',lst);
  n := ole2.get_num_property (c, 'Value');
  /* Destroy the argument list. */
  ole2.destroy_arglist(lst);
  ole2.release_obj(c);
  return n;
END;
END;
```

セルにアクセスするには、次のコードを使用します。

```
spreadsheet.setcell(3, 5, 91.73);
:block1.item1 := spreadsheet.getcell(2, 4);
```

6.1.2 ActiveX コントロール

ActiveX コントロール（OLE または OCX コントロールと呼ばれていたもの）は、スタンドアロン・ソフトウェア・コンポーネントで、アプリケーションに埋め込むことにより軽快なユーザー・インタフェース・コントロールを提供します。

ActiveX コントロールと OLE オブジェクトとの違いは次のとおりです。

- ActiveX コントロールは別個のアプリケーションではなく、ActiveX コンテナにプラグインするサーバーです。ActiveX コントロールは自己完結型です。
- 各 ActiveX コントロールは、一連のプロパティ、メソッドおよびイベントを提供します。プロパティによって ActiveX コントロールの物理属性および論理属性が定義され、メソッドによって ActiveX コントロールが実行するアクションが定義されます。イベントは ActiveX コントロールの状態の変更を示します。

- ActiveX コントロールは、アプリケーションとともに配布およびインストールする必要があります。

6.1.2.1 ActiveX コントロールを利用する場合

ActiveX コントロールは通常、自己完結型機能性を追加することによりアプリケーションを拡張するために使用されます。

たとえば、タブ付プロパティ・パレット、スピン・コントロール、カレンダ・コントロール、ヘルプ・コントロールなどでアプリケーションを拡張できます。

独自の ActiveX コントロールまたは OLE サーバーを開発するには、かなりの努力が必要です。サードパーティ・ベンダーにより開発および配布されている ActiveX コントロールおよび OLE サーバーを使用することをお薦めします。

6.1.2.2 ActiveX コントロールの操作

各 ActiveX コントロールは、一連のプロパティ、メソッドおよびイベントを提供します。プロパティによって ActiveX コントロールの物理属性および論理属性が定義され、メソッドによって ActiveX コントロールが実行するアクションが定義されます。イベントは ActiveX コントロールの状態の変更を示します。

ActiveX コントロールの操作方法は次のとおりです。

- ActiveX コントロール・プロパティの設定と取得
- ActiveX コントロール・メソッドのコール

  注意 : ActiveX コントロール・メソッドを起動する前に、まず、そのメソッドおよびイベントを Forms Developer にインポートする必要があります。ActiveX メソッドおよびイベントをインポートすると、Forms Developer のネイティブ環境において ActiveX コントロールと対話できるようになります。

- ActiveX コントロール・イベントへの応答

ActiveX コントロールを操作するには、（Forms Developer の）`STANDARD` と `OLE2` ビルトイン・パッケージを使用します。

6.1.2.3 ActiveX イベントへの応答

ActiveX イベントへの応答は、ActiveX イベント・パッケージまたは On-Dispatch-Event トリガーに独自のコードを書き込んで行います。

各 ActiveX イベントは、イベント・パッケージに定義されている PL/SQL プロシージャに対応付けられています。コントロールがイベントを起動すると、プロシージャのコードは自動的に実行されます。

プロシージャ名は、対応するイベントを表す内部番号により決まります。イベントにより生成される制限付きプロシージャには、次に示すものに似たアプリケーション・プログラミング・インタフェースがあります。

```
PROCEDURE /*Click*/ event4294966696(interface OleObj);
```

注意 : ActiveX プロシージャは制限モードで実行されます。On-Dispatch-Event トリガー内でイベント・プロシージャをコールする際に、そのプロシージャが制限モードで実行されるのか、または非制限モードで実行されるのかを、FORMS4W.DISPATCH_EVENT コールを使用して明示的に定義できます。制限プロシージャの定義時に、OUT パラメータは監視されません。

6.1.2.4 ActiveX コントロールの配布

ActiveX コントロールを含むアプリケーションを利用するには、その ActiveX コントロールを利用する必要があります。

ActiveX コントロールを利用するには、次のことを行う必要があります。

- ActiveX コントロールをクライアント・マシンに登録します。

  ActiveX コントロールに付属しているインストール・プログラムを使用して ActiveX コントロールをインストールする場合、登録は自動的に行われます。

  手動による登録の場合は、regActiveX32.exe または regsvr32.exe を使用します。これらは共に Microsoft の開発ツールに付属し、また ActiveX コントロール・ベンダーからも入手できます。

- ActiveX DLL をクライアント・マシン（たとえば、C:¥WINDOWS¥SYSTEM）にコピーします。

  ほとんどの ActiveX コントロールでは、Microsoft Foundation Class ランタイム・ライブラリ（MFC40.DLL）などのサポート DLL を必要とします。DLL は、¥WINDOWS¥SYSTEM ディレクトリまたは検索パスに存在する必要があります。DLL が古かったり、またはない場合、ActiveX コントロールは正しく登録されません。

注意 : ActiveX コントロールは、サード・パーティの ActiveX コントロール・ベンダーにより配布されたものなのか、アプリケーション開発ツールにバンドルされたものなのかにかかわらず、ActiveX コントロールを配布する前に、追加料金の支払いまたは追加ライセンスの取得が必要になる場合があります。

6.1.2.5 ActiveX サポート

サポートとは、ActiveX コントロールを作成、操作し、ActiveX コントロールとの通信を行う機能を備えていることです。

コンポーネント	コンテナ
Form Builder	あり
Graphics Builder	なし
Procedure Builder	なし

コンポーネント	コンテナ
Project Builder	なし
Query Builder	なし
Report Builder	なし
Schema Builder	なし

6.1.2.5.1 ActiveX プロパティ

この項では、Forms Developer でサポートされている ActiveX プロパティをリストします。

コンポーネント	プロパティ	説明
Form Builder	OLE クラス	OLE コンテナに常駐できる OLE オブジェクトのクラスを決定します。
	コントロール・プロパティ	ActiveX コントロール・プロパティの設定を可能にします。 「ActiveX プロパティ」ダイアログへのアクセスは、プロパティ・パレットを使用するか、または ActiveX コントロールをクリックしてからマウスの右ボタンをクリックして行います。
	コントロールについて	ActiveX コントロールに関する情報を表示します。
	コントロール・ヘルプ	コントロール専用ヘルプ（利用できる場合）

6.1.2.5.2 ActiveX/OLE ビルトイン

各コンポーネントでサポートされている ActiveX および OLE ビルトインのリストは、6.1.1.7.4 項を参照してください。

6.1.2.6 ActiveX ガイドライン

この項では、ActiveX コントロールを処理するためのガイドラインを提供します。

項目	推奨事項
独自の ActiveX コントロールの作成	独自の ActiveX コントロールまたは OLE サーバーを開発するには、かなりの努力が必要です。サードパーティ・ベンダーにより開発および配布されている ActiveX コントロールおよび OLE サーバーを使用することをお薦めします。

項目	推奨事項
ActiveX コントロールの初期化	シングル・レコードは Forms Runtime の起動時にただちに初期化されるため、ブロック構造の ActiveX コントロールを、「単一レコード」プロパティを「はい」に設定した状態で使用します。 複数レコードの場合、各レコードは、そのレコードにナビゲートするまで初期化されません。 初期化が行われない場合、ActiveX コントロール項目は空であるため、ActiveX コントロールがまったく利用できないといった印象を与えます。
OLE 可変レコード・タイプの管理	■ Microsoft Excel などの OLE サーバーの中には可変レコード・タイプを使用するものがあります。可変レコード・タイプとの間で必要な変換を行うには、STANDARD ビルトイン・パッケージを使用します。 ■ 可変レコード・タイプの存続期間および有効範囲は、トリガーに限定されています（可変レコードのメモリー領域はトリガーの終了時に解放されます）。可変レコード・タイプの存続期間および有効範囲を拡張するには、To_Variant() の永続パラメータを TRUE に設定し、結果をグローバル変数に割り当てます。 **注意 :** グローバルな可変レコードは、Destroy_Variant() を使用して明示的に破棄する必要があります。同様に、Create_OleObj() を使用して作成された OLE オブジェクトは有効範囲においてグローバルです（永続パラメータのデフォルトは TRUE）。Release_Obj() を明示的に呼び出してグローバル・オブジェクトを開放する必要があります。
ActiveX ファイルの移動	ActiveX ファイルは“インストール”ディレクトリに保持します。ActiveX ファイルを別のディレクトリに移動しないでください。 インストール時に、ActiveX コントロールのインストール先ディレクトリは、Windows 95 および Windows NT の Windows レジストレーション・データベースに登録され、開発環境で ActiveX コントロールを参照できるようになります。 ActiveX コントロールを別のディレクトリに移動したり、ディレクトリ名を変更したりすると、レジストリの情報が無効になります。 ActiveX コントロールの移動やそのディレクトリ名の変更が必要な場合は、ほとんどの Microsoft 開発製品に付属している regsrv32.exe または regActiveX32.exe ユーティリティを使って、ActiveX コントロールを新規の場所に再登録します。

項目	推奨事項
移植性の問題	ActiveX がサポートされるのは、Windows プラットフォーム上のみです。ActiveX コントロールは、Web 上または UNIX 上では使用できません。移植性が問題になる場合は、ActiveX コントロールを使用しないでください。

ActiveX コールのデバッグ	オブジェクト・タイプがコンパイル時にチェックされない場合は、そのオブジェクトのクラスに定義されていないオブジェクトに対してファンクションをコールすることができます。ファンクションは、名前ではなく ID により結合されているため、意図していない別のファンクションがコールされて、異常なエラーが発生することがあります。
	
	正しいメソッドのコールを保証する 1 つの方法は、ハードコードされた定数を GET_OLE_MEMBERID のコールに置き換えて、生成されたファンクションを変更することです。例は次のとおりです。
	
	`Procedure Ole_Add(interface OleObj, TimeBegin VARCHAR2, TimeEnd VARCHAR2, Text VARCHAR2, BackColor OleVar := OleVar_Null) IS`
	
	`BEGIN`
	
	`Init_OleArgs (4);`
	
	`Add_OleArg (TimeBegin);`
	
	`Add_OleArg (TimeEnd);`
	
	`Add_Olearg (Text);`
	
	`Add_OleArg (BackColor);`
	
	`Call_Ole (interface, 2);`
	
	`END ;`
	
	`Call_ole()` を次のように置き換えます。`Call_Ole (interface, Get_Ole_MemberID(interface, 'Add'));`
	
	`GET_OLE_MEMBERID` コールの後に、`FORM_SUCCESS` があるかどうかをチェックできます。

項目	推奨事項
制限	■ ActiveX イベント・プロシージャは制限されています。一般的に、同一のイベントが複数のアイテムに適用するために GO_ITEM が必要である場合を除き、GO_ITEM を ActiveX プロシージャ・コード内でコールすることはできません。ただし、同一イベントが複数項目に適用され、GO_ITEM が必要な場合は除きます。こうした場合には、On-Dispatch トリガー（ブロックまたはフォーム・レベル）において、DISPATCH_EVENT(RESTRICTED_ALLOWED) をコールすることにより、GO_ITEM ビルトインを使用できます。**注意 :** イベント・プロシージャは On-Dispatch トリガー・コードに引き続いて自動的にコールされるため、明示的にコールする必要はありません。 ■ ActiveX コントロール向けの初期化イベントは Forms Runtime では起動しません。これらの初期化イベントは意図的に使用不可に設定されています。コントロールのネイティブ初期化イベントのかわりに、When-New-Item-Instance または When-New-Record-Instance を使用できます。

6.1.2.7 アプリケーションへの ActiveX コントロールの追加

ActiveX コントロールをアプリケーションに追加する方法の詳細は、オンライン・ヘルプを参照してください。

6.1.2.8 ActiveX の例

この項では、ActiveX コントロールのスタート・ガイドとしていくつかの例を紹介します。

6.1.2.8.1 例 1: ActiveX コントロール・プロパティの設定

Form Builder において、:item('item_name').ocx.server_name.property バインド変数構文を使用して、AactiveX のプロパティ値を割り当てるか、または取り出すことができます。

例 :

```
:item('ActXitem').OCX.Spindial.spindialctrl.1.Needleposition:=
:item('ActXitem').OCX.Spindial.spindialctrl.1.Needleposition + 1;
```

ActXitem は項目名、Spindial.spindialctrl.1 は ActiveX コントロール・サーバー名、Needleposition はプロパティ名です。

system.cursor_item が ActiveX コントロールである場合は、次のコードも使用できます。

```
:form.cursor_item.OCX.spindial.spindialctrl.1.Needlposition :=
:form.cursor_item.OCX.spindial.spindialctrl.1.Needlposition + 1;
```

6.1.2.8.2 例 2: ActiveX コントロール・プロパティの取得

Form Builder において、プロパティ・アセサー・ファンクションおよびプロシージャを使用して、ActiveX プロパティを取得、設定できます。

例 :

```
tblname varchar2;
tblname := table_pkg.TableName(:item('Oblk.Oitm').interface);
```

Table_pkg は、OLE インポータから作成されたプログラム単位の名前です。TableName は、プロパティ・アセサー名です。Oblk は、ブロック名、Oitm は項目名です。

6.1.2.8.3 例 3: ActiveX コントロール・メソッドのコール

この例では、SpreadTable パッケージ内の GetCellByColRow メソッドを使用して Spread Table ActiveX コントロールからセル値を取得します。

```
DECLARE
   Cur_Row number;
   Cur_Col number;
   The_OLE_Obj OleObj;
BEGIN
   Cur_Row:=SpreadTable.CurrentRow(:ITEM('BLK.ITM').interface);
   Cur_Col:=SpreadTable.CurrentCol(:ITEM('BLK.ITM').interface);
   The_OLE_Obj:=SpreadTable.GetCellByColRow(:ITEM('BLK.ITM').interface,
                                            Cur_Col, Cur_Row);
END;
```

6.2 外部ファンクションを使用したアプリケーションのカスタマイズ

外部ファンクションを使用すると、アプリケーションをカスタマイズして機能を強化できます。

この項では、次のことを説明します。

- 6.2.1 項「外部ファンクション」
- 6.2.2 項「外部ファンクションのインタフェース」
- 6.2.3 項「外部ファンクションのガイドライン」
- 6.2.4 項「外部ファンクションの作成」
- 6.2.5 項「外部ファンクションの例」

6.2.1 外部ファンクション

外部ファンクションは 3GL プログラミング言語で書かれたサブプログラムで、ユーザー固有の要件に合せてアプリケーションをカスタマイズできます。

外部ファンクションは、Oracle データベース、Forms Developer および Reports Developer の変数、項目、列およびパラメータとの対話に使用します。また、Windows DLL または API などの外部定義ファンクションをコールすることもできます。

6.2.1.1 外部ファンクションを利用する場合

外部ファンクションは次の作業を行うためによく使用されます。

- 複雑なデータ操作を行います。
- データを、オペレーティング・システムのテキスト・ファイルから Forms Developer および Reports Developer に渡します。
- LONG RAW データを操作します。
- サーバーで処理するために PL/SQL ブロック全体を渡す。
- アプリケーションのフォントおよびカラーの属性を設定します。
- メールをアプリケーションから直接送ります。
- Windows ヘルプをアプリケーションの一部として表示します。
- Microsoft Windows SDK にアクセスします。
- パイプなどの低レベルのシステム・サービスを利用します。
- プリンタまたはロボットなどのリアル・タイム・デバイスを制御します。

6.2.1.2 外部ファンクションのタイプ

3 種類の外部ファンクションを開発できます。

6.2.1.2.1 Oracle プリコンパイラ外部ファンクション Oracle プリコンパイラ外部ファンクションは最も一般的な外部ファンクションです。Oracle プリコンパイラを使用すると、Oracle データベース、Forms Developer または Reports Developer の変数、項目、列およびパラメータなどにアクセスする外部ファンクションを作成できます。

Oracle プリコンパイラ外部ファンクションには、Oracle プリコンパイラ・インタフェースが取り込まれます。このインタフェースにより、埋込み SQL コマンドを使用して、次のサポートされたホスト言語のいずれかでプログラムを作成できます。Ada、C、COBOL、FORTRAN、Pascal および PL/I。

Oracle プリコンパイラ外部ファンクションのソース・ファイルには、埋込み SQL 文を使用したホスト・プログラミング言語文および Oracle プリコンパイラ文が含まれています。Oracle プリコンパイラ外部ファンクションをプリコンパイルすると、埋込み SQL 文は等価

のホスト・プログラミング言語文に置換されます。プリコンパイルにより、ホスト言語コンパイラでコンパイルできるソース・ファイルができます。

6.2.1.2.2 Oracle コール・インタフェース（OCI）外部ファンクション OCI 外部ファンクションには、Oracle コール・インタフェースが取り込まれます。このインタフェースにより、Oracle データベースのコールを含むサブプログラムを作成できます。OCI のみが取り込まれた（Oracle プリコンパイラ・インタフェースは取り込まれない）外部ファンクションでは、Forms Developer または Reports Developer の変数、項目、列、パラメータにアクセスできません。

注意 : Oracle プリコンパイラ・インタフェースと OCI の両方を組み合せた外部ファンクションを開発することもできます。

6.2.1.2.3 Oracle 以外の外部ファンクション Oracle 以外の外部ファンクションには、Oracle プリコンパイラ・インタフェースも、OCI も取り込まれません。たとえば、Oracle 以外の外部ファンクションは、すべて C 言語のみで作成されることもあります。オラクル以外の外部ファンクションでは、Oracle データベースをはじめ、Forms Developer または Reports Developer の変数、項目、列、パラメータのいずれにもアクセスできません。

6.2.2 外部ファンクションのインタフェース

Forms Developer および Reports Developer では、プログラミング言語として PL/SQL を使用します。Windows DLL の C ファンクションなどの外部ファンクションをコールするには、PL/SQL に外部ファンクションとの通信インタフェースが必要です。

外部ファンクションとの通信には、Oracle 外部ファンクション・インタフェース（ORA_FFI）またはユーザー・イグジット・インタフェースという 2 つの固有のインタフェースが使用できます。

6.2.2.1 Oracle 外部ファンクション・インタフェース（ORA_FFI）

ORA_FFI は、Forms Developer および Reports Developer で PL/SQL プログラムから 3GL ルーチンをコールするための移植性のある汎用メカニズムです。

PL/SQL インタフェースから起動する外部ファンクションは、ダイナミック・ライブラリに含まれている必要があります。ダイナミック・ライブラリの例としては、Microsoft Windows のダイナミック・リンク・ライブラリや UNIX システムの共有ライブラリがあります。

6.2.2.2 外部ファンクションへのユーザー・イグジット・インタフェース

ユーザー・イグジット・インタフェースは、Forms Developer および Reports Developer で PL/SQL サブプログラムから 3GL ルーチンをコールするためのプラットフォーム固有のメカニズムです。

ユーザー・イグジット・インタフェースから起動する外部ファンクションは、ダイナミック・リンク・ライブラリ（.DLL）に含まれているか、または実行可能なアプリケーションにリンクされている必要があります。

6.2.2.3 ORA_FFI とユーザー・イグジットとの比較

この項では、ORA_FFI およびユーザー・イグジットを使用する場合の長所と短所を説明します。

外部ファンクション	長所	短所
ユーザー・イグジット	■ ユーザー・イグジットは実行プログラムにリンクされています。この“緊密な結び付き”により、現行のデータベース接続を利用できます。	■ ユーザー・イグジットを使用する上で最も重大な短所は、メンテナンス上の負担です。ユーザー・イグジットの変更や Forms Developer または Reports Developer のアップグレードを行ったら、その都度ユーザー・イグジットを再リンクする必要があります。 ■ ユーザー・イグジットは汎用ではありません。プラットフォーム固有です。

外部ファンクション	長所	短所
ORA_FFI	■ ORA_FFI は、純粋な PL/SQL 仕様部です。ORA_FFI 仕様部は、Forms Developer または Reports Developer コンポーネント内ではなく、ライブラリ（.PLL ファイル）内に存在します。Forms Developer または Reports Developer のアップグレードや外部ファンクションの変更を行っても、PLL ファイルを変更したり、再生成する必要はありません。 ■ ORA_FFI は汎用です。 ■ Forms Developer および Reports Developer には、すでに使用可能なライブラリ（Windows API ファンクション）へのアクセスを可能にする ORA_FFI パッケージ（D2KWUTIL.PLL）が複数用意されています。	■ OAR_FFI を使用しながら、Pro*C で独自の外部コード・モジュールを作成している場合は、現行のオープン・データベース接続を使用できません。2 番目の接続をオープンする必要があります。 ■ 構造体や配列などの複雑なデータ型を渡すことはできません。たとえば、EXEC TOOLS GET または EXEC TOOLS PUT を使用して、Forms Developer または Reports Developer インタフェースとデータをやりとりすることはできません。

6.2.3 外部ファンクションのガイドライン

この項では、外部ファンクションを処理するためのガイドラインを提供します。

項目	推奨事項
どの外部ファンクション・インタフェースを使用しますか？	Oracle 外部ファンクション・インタフェース（ORA_FFI）を使用します。ORA_FFI は、移植性のある汎用インタフェースで、メンテナンスはほとんど、またはまったく必要ありません。
外部ファンクションから画面 I/O を実行できますか？	外部ファンクションからホスト言語の画面 I/O を実行しないでください。この制限が存在するのは、画面 I/O を実行するためにホスト言語が使用するランタイム・ルーチンと、画面 I/O を実行するために Forms Developer および Reports Developer が使用するルーチンが競合するためです。ただし、外部ファンクションからホスト言語によるファイル I/O を実行することはできます。
ユーザー・イグジットを作成するためにどのホスト言語を使用しますか？	どのホスト言語を使用するかは好みの問題です。しかし、C 言語をお薦めします。 **注意 :** C 言語のランタイム・ファンクションの中には、.DLL ファイルで使用できないものがあります。詳細は、使用しているコンパイラのマニュアルを参照してください。

項目	推奨事項

ユーザー・イグジットをプリコンパイルするためにどのプリコンパイラを使用しますか？	Pro*C バージョン 2.2.4/8.0.4 を使用します。
	プリコンパイル時に、次の MSVC コンパイラ・フラグを必ず指定します。
	`Large, Segment Setup:SS != DS, DSloads on function entry`
	`Assume 'extern' and Uninitialized Data 'far' is checked Yes`
	`In Windows Prolog/Epilogue, Generate prolog/Epilogue for None`

Forms Developer または Reports Developer の以前のバージョンからのアップグレード時に、ユーザー・イグジットを再コンパイルする必要がありますか？	はい。特に、それぞれ別のユーザー・イグジット・セットを持つ実行プログラムが複数ある場合、メンテナンス上の負担が発生します。 ユーザー・イグジットの変更時、あるいは Forms Developer または Reports Developer のアップグレード時には、ユーザー・イグジットを Forms Developer アプリケーションまたは Reports Developer アプリケーションに再リンクする必要があります。
外部ファンクションを Web 上で利用できますか？	`ORA_FFI` およびユーザー・イグジットは Web 上で機能しません。Web 上で利用する場合、外部ファンクションは、ブラウザ側ではなく、サーバー側の DLL とインタフェースします。

外部ファンクションの詳細は、次の出版物を参照してください。

Oracle プリコンパイラ・インタフェース	Oracle プリコンパイラに応じたプリコンパイラ・プログラマーズ・ガイド
ホスト言語のサポート	使用しているオペレーティング・システム固有の Oracle インストレーション・ガイド
外部ファンクション処理時のオペレーティング・システム固有の要件	オンライン・ヘルプ
OCI	『Oracle8 コール・インタフェース・プログラマーズ・ガイド』
DLL の作成	オンライン・ヘルプと使用しているコンパイラのマニュアル
ORA_FFI	オンライン・ヘルプ
ユーザー・イグジット	オンライン・ヘルプ
PL/SQL	『PL/SQL ユーザーズ・ガイド』またはオンライン・ヘルプ

6.2.4 外部ファンクションの作成

この項では、外部ファンクションのインタフェースを作成するための詳細な手順を説明します。

- 外部ファンクションへの ORA_FFI インタフェースの作成
- 外部ファンクションへのユーザー・イグジット・インタフェースの作成

6.2.4.1 外部ファンクションへの ORA_FFI インタフェースの作成

次の例では、WinSample という名前の PL/SQL パッケージを作成します。WinSample パッケージには、ダイナミック・ライブラリ KRNL386.EXE にある外部ファンクション GetPrivateProfileString へのインタフェースが含まれています。

注意 : 外部ファンクションへの ORA_FFI インタフェースを作成する場合は、2 つの基本的な手順を実行します。第一に、サブプログラムを作成し、外部ファンクション（ディスパッチャ・ファンクション）に関連付けます。PL/SQL サブプログラムを外部ファンクションに対応付けることにより、対応付けられた PL/SQL サブプログラムをコールするたびに、外部ファンクションを起動することができます。外部ファンクションと PL/SQL サブプログラムとの対応付けが必要な理由は、Forms Developer および Reports Developer で PL/SQL 構成体が使用されるためです。第二に、引数をディスパッチャ・ファンクションに渡す PL/SQL ファンクションを作成します。ディスパッチャ・ファンクションは外部ファンクションを起動します。

1. パッケージ仕様部を作成します。

   パッケージ仕様部はライブラリを表している必要があります。また、起動しようとする PL/SQL ファンクションを定義する必要があります。

   例 :

```
PACKAGE WinSample IS
FUNCTION GetPrivateProfileString
(Section IN  VARCHAR2,
Entry IN  VARCHAR2,
DefaultStr IN  VARCHAR2,
ReturnBuf IN  OUT VARCHAR2,
BufLen IN  PLS_INTEGER,
Filename IN  VARCHAR2)
RETURN PLS_INTEGER;
END;
```

   この例では、WinSample.GetPrivateProfileString PL/SQL ファンクションを呼び出して、ダイナミック・ライブラリ KRNL386.EXE にある GetPrivateProfileString 外部ファンクションを起動します。

   注意 : C ファンクション GetPrivateProfileString のパラメータをチェックして、一致する PL/SQL パラメータ型と PL/SQL 戻り型を指定します。C データ型 int は、

PL/SQL パラメータ IN PLS_INTEGER および PL/SQL 戻り型 PLS_INTEGER と等価です。C データ型 char は、PL/SQL パラメータ IN VARCHAR2 と等価です。

2. ライブラリとファンクション・ハンドルを定義します。

   例 :

```
PACKAGE BODY WinSample IS
lh_KRNL386 ORA_FFI.LIBHANDLETYPE;
fh_GetPrivateProfileString ORA_FFI.FUNCHANDLETYPE;
```

   この手順において、ライブラリおよびファンクションのハンドル型を宣言します。後で、ORA_FFI.LOAD_LIBRARY および ORA_FFI.REGISTER_FUNCTION を使用してライブラリをロードして、ファンクションを登録します。これらのファンクションはそれぞれ、指定したライブラリおよびファンクションにハンドル（ポインタ）を戻します。ORA_FFI.LIBHANDLETYPE および ORA_FFI.FUNCHANDLETYPE は、これらのハンドルの PL/SQL データ型です。

3. ディスパッチャ・ファンクションを作成します。ディスパッチャ・ファンクションは外部ファンクションを起動します。

   例 :

```
FUNCTION i_GetPrivateProfileString
(funcHandle IN ORA_FFI.FUNCHANDLETYPE,
Section IN OUT VARCHAR2,
Entry IN OUT VARCHAR2,
DefaultStr IN OUT VARCHAR2,
ReturnBuf IN OUT VARCHAR2,
BufLen IN PLS_INTEGER,
Filename IN OUT VARCHAR2)
 RETURN PLS_INTEGER;
 PRAGMA INTERFACE(C,i_GetPrivateProfileString,11265);
```

   外部ファンクションをコールするディスパッチャ・ファンクションの最初の引数には、少なくとも 1 つのパラメータが必要です。また、最初のパラメータは、サブプログラムが起動する登録された外部ファンクションのハンドルである必要があります。

   ディスパッチャ・ファンクションを PL/SQL ファンクションからコールする場合は、手順 2（fh_GetPrivateProfileString）での定義に従ってファンクション・ハンドルを渡します。

   ディスパッチャ・ファンクションのコール時に、PRAGMA 文は、ダイナミック・ライブラリと通信するメモリー位置（前述のコードで指定した 11265）に制御を渡します。

4. ディスパッチャ・ファンクションを呼び出す PL/SQL ファンクションを作成します。この PL/SQL ファンクションは、パッケージ仕様部で定義したファンクション（手順 1）です。

例:

```
FUNCTION GetPrivateProfileString
(Section IN  VARCHAR2,
Entry IN  VARCHAR2,
DefaultStr IN  VARCHAR2,
ReturnBuf IN  OUT VARCHAR2,
BufLen IN  PLS_INTEGER,
Filename IN  VARCHAR2)
RETURN PLS_INTEGER IS
Section_l VARCHAR2(512)    := Section;
Entry_l VARCHAR2(512)    := Entry;
DefaultStr_l VARCHAR2(512) := DefaultStr;
ReturnBuf_l VARCHAR2(512)  := RPAD(SUBSTR(NVL
(ReturnBuf,' '),1,512),512,CHR(0));
BufLen_l PLS_INTEGER    := BufLen;
Filename_l VARCHAR2(512)   := Filename;
rc PLS_INTEGER;
BEGIN
     rc  := i_GetPrivateProfileString
(fh_GetPrivateProfileString,
Section_l,
Entry_l,
DefaultStr_l,
ReturnBuf_l,
BufLen_l,
Filename_l);
ReturnBuf := ReturnBuf_l;
RETURN (rc);
END;
```

これは、アプリケーションからコールする PL/SQL ファンクションです。このファンクションが引数をディスパッチャ・ファンクション i_GetPrivateProfileString に渡すと、i_GetPrivateProfileString が KRNL386.EXE にある C ファンクション GetPrivateProfileString を起動します。ディスパッチャ・ファンクションの最初の引数はファンクション・ハンドルでなければならないことを思い出してください。ここで、fh_GetPrivateProfileString は、手順 2 で宣言されているファンクション・ハンドルを渡すために使用されます。

5. パッケージ本体を作成します。

パッケージ本体では、外部ファンクションを初期化するために、次の 4 つの手順が実行される必要があります。

- ライブラリをロードします。
- ライブラリにあるファンクションを登録します。
- パラメータを登録します（ある場合）。

- 戻り値の型を登録します（ある場合）。

例：

```
BEGIN
/* Load the library .*/
 lh_KRNL386 := ORA_FFI.LOAD_LIBRARY
('location of the DLL here','KRNL386.EXE');

/* Register the foreign function. */
fh_GetPrivateProfileString := ORA_FFI.REGISTER_FUNCTION(lh_
KRNL386,'GetPrivateProfileString',ORA_FFI.PASCAL_STD);

/* Register the parameters. */
ORA_FFI.REGISTER_PARAMETER
(fh_GetPrivateProfileString,ORA_FFI.C_CHAR_PTR); ORA_FFI.REGISTER_PARAMETER
(fh_GetPrivateProfileString,ORA_FFI.C_CHAR_PTR); ORA_FFI.REGISTER_PARAMETER
(fh_GetPrivateProfileString,ORA_FFI.C_CHAR_PTR); ORA_FFI.REGISTER_PARAMETER
(fh_GetPrivateProfileString,ORA_FFI.C_CHAR_PTR);
   ORA_FFI.REGISTER_PARAMETER
(fh_GetPrivateProfileString,ORA_FFI.C_INT); ORA_FFI.REGISTER_PARAMETER
(fh_GetPrivateProfileString,ORA_FFI.C_CHAR_PTR);

/* Register the return type. */
ORA_FFI.REGISTER_RETURN(fh_GetPrivateProfileString,ORA_FFI.C_INT);
END WinSample;
```

手順 2 において、ライブラリおよびファンクションに対して 2 つのハンドルを宣言したことを思い出してください。この手順において、ORA_FFI.LOAD_LIBRARY および ORA_FFI.REGISTER_FUNCTION ファンクションを使用して、ハンドルに値を割り当てます。

ORA_FFI.LOAD_LIBRARY は、2 つの引数を取ります。ダイナミック・ライブラリの位置と名前です。ORA_FFI.REGISTER_FUNCTION は、3 つの引数を取り出します。ファンクションが含まれるライブラリのライブラリ・ハンドル、ファンクション名、コール標準です。コール標準は、C_STD（C コール標準の場合）または PASCAL_STD（Pascal コール標準の場合）のいずれかです。

ライブラリのロードおよびファンクションの登録後に、パラメータおよび戻り値の型（ある場合）を登録する必要があります。

ORA_FFI.REGISTER_PARAMETER と ORA_FFI.REGISTER_RETURN はそれぞれ 2 つの引数を取ります。ファンクション・ハンドルおよび引数型です。

6. Forms Developer または Reports Developer を使用してパッケージが含まれるライブラリ・ファイル (.PLL）を作成し、それをアプリケーションに連結します。
7. アプリケーションから外部ファンクションをコールします。

   例 :

```
x := Winsample.GetPrivateProfileString
('Oracle', 'ORACLE_HOME', '<Not Set>', 'Value', 100,'oracle.ini');
```

6.2.4.2 外部ファンクションへのユーザー・イグジット・インタフェースの作成

ユーザー・イグジットは汎用ではありません。プラットフォーム固有です。ユーザー・イグジットのインプリメントの詳細の中には、各オペレーティング・システムに固有のものがあります。次の例では、Windows 95 におけるユーザー・イグジットの作成方法について説明します。

Microsoft Windows では、ユーザー・イグジットから起動する外部ファンクションは、ダイナミック・リンク・ライブラリ（.DLL）に含まれています。DLL は、含まれているコードが起動されたときにのみメモリーにロードされるライブラリです。

6.2.4.2.1 例 : Windows 95 でのユーザー・イグジットの作成

次の例では、ID 列を EMP 表に追加する外部ファンクションを作成します。

この例では、次に示す複数のサンプル・ファイルを使用します。

- **UE_SAMP.MAK** は、IAPXTB コントロール構造体を含むプロジェクト・ファイルです。このプロジェクトを作成すると、UE_SAMP.DLL が生成されます。
- **IFXTB60.DLL** は、ユーザー・イグジット・インタフェースから起動する外部ファンクションを含むデフォルト・ファイルです。このファイルは、Form Builder に付属する

DLL ファイルで、当初はユーザ定義の外部ファンクションを含んでいません。このファイルは、インストール時に ORACLE_HOME¥BIN ディレクトリに配置されます。新規の外部ファンクションの作成時には、既存の IFXTB60.DLL ファイルを新規の IFXTB60.DLL と置き換えてください。

- **UE_XTB.C** は、IAPXTB コントロール構造体を作成するためのテンプレート・ソース・ファイルです。UE_XTB.C には、IAPXTB コントロール構造体のエントリの例が含まれています。このファイルを変更し、外部ファンクション・エントリを追加します。
- **UE.H** は、IAPXTB 構造体の定義に使用されるサンプル・ヘッダー・ファイルです。
- **IFXTB60.DEF** には、独自の DLL を作成するために必要な定義が含まれています。IFXTB60.DEF を使用して、外部ファンクションをエクスポートします。IFXTB60.DEF には複数のエクスポート文が含まれています。これらのエクスポート文は、ユーザー・イグジット・インタフェースにアクセスするために Form Builder で使用されますので、変更しないでください。
- **UEZ.OBJ** は、独自の .OBJ ファイルにリンクする .OBJ ファイルです。

ユーザー・イグジットのサンプル・ファイルは、ORACLE_HOME ディレクトリ（たとえば、C:¥ORAWIN95¥FORMS60¥USEREXIT）にあります。

1. 外部ファンクションを作成します。

   たとえば、テキスト・ファイル UEXIT.PC を作成してから、次を追加します。

```
/* UEXIT.PC file */
/* This foreign function adds an ID column to the EMP table. */
#ifndef UE
#include "ue.h"
#endif
#ifndef _WINDLL
#define SQLCA_STORAGE_CLASS  extern
#endif
EXEC SQL INCLUDE sqlca.h;
void  AddColumn() {
EXEC SQL alter table EMP add ID varchar(9);
}
```

2. Pro*C プリコンパイラで外部ファンクションをプリコンパイルます。

   たとえば、Pro*C を使用して UEXIT.PC ファイルをプリコンパイルします。UEXIT.PC をプリコンパイルすると、Pro*C では、UEXIT.C という名前の C ファイルが作成されます。

   注意 : プリコンパイル時に、次の MSVC コンパイラ・フラグを必ず指定します。

```
Large, Segment Setup:SS != DS, DSloads on function entry

Assume 'extern' and Uninitialized Data 'far' is checked Yes
```

```
In Windows Prolog/Epilogue, Generate prolog/Epilogue for None
```

3. ヘッダー・ファイルを作成します。

   ヘッダー・ファイルは、外部ファンクションを定義する必要があります。

   たとえば、次の行を追加して、サンプル・ヘッダー・ファイル UE.H を変更します。

```
extern   void   AddColumn();
```

4. IAPXTB コントロール構造体を作成します。

   たとえば、UE.H のインクルード文 UE.H (# include "ue.h")、ユーザー・イグジット名 (Add_ID_Column)、外部ファンクション名 (AddColumn)、および言語タイプ (XITCC) を追加して、サンプル・ファイル UE_XTB.C を変更します。

```
#ifndef UE
#include "ue.h"
#endif /* UE */
#include "ue_samp.h"
/* Define the user exit table */
exitr iapxtb[] =  {  /* Holds exit routine pointers */
     "Add_ID_Column", AddColumn, XITCC,
     (char *) 0, 0, 0         /* zero entry marks the end */
}; /* end iapxtb */
```

5. DLL ファイルを作成します。DLL ファイル作成の手順は、使用しているコンパイラによって異なります。詳細は、使用しているコンパイラのマニュアルを参照してください。

   例えば、コンパイラを使用して、次のものを含むプロジェクトを作成します。UE_SAMP.MAK、IFXTB60.DEF、UEZ.OBJ、UE_XTB.C および UEXIT.C

   DLL を作成する前に、次のファイルをリンクする必要があります。

```
LIBC.LIB
OLDNAMES
C:¥ORAWIN95¥FORMS60¥USEREXIT¥IFR60.LIB
C:¥ORAWIN95¥PRO20¥USEREXIT¥SQLLIB18.LIB
C:¥ORAWIN95¥PRO20¥USEREXIT¥SQXLIB18.LIB
```

   UE_SAMP.MAK プロジェクトを作成すると、UE_SAMP.DLL. という名前の DLL ファイルが作成されます。UE_SAMP.DLL エントリを、レジストリの FORMS60_USEREXITS パラメータで定義されている DLL リストに追加します。

   UE_SAMP.DLL を IFXTB60.DLL に改名し、C:¥ORAWIN95¥BIN ディレクトリにある IFXTB60.DLL をバックアップし、新規の IFXTB60.DLL に C:¥ORAWIN95¥BIN ディレクトリにコピーすることもできます。

6. 外部ファンクションをユーザー・イグジットから起動します。

たとえば、ユーザー・イグジットから外部ファンクションを呼び出す When-Button-Pressed トリガーを作成します。

次の文では、USER_EXIT ビルトインにあるユーザー・イグジット名 Add_ID_Column を指定して、AddColumn 外部ファンクションを起動する方法を示します。

```
/* Trigger:  When-Button-Pressed */
USER_EXIT('Add_ID_Column');
```

6.2.5 外部ファンクションの例

この項では、複数の例を示しながら外部ファンクションの使用方法を説明します。

6.2.5.1 Windows ヘルプをコールする ORA_FFI の使用

```
/* WinHelp ORA_FFI.                                                    */
/*                                                                     */
/*                                                                     */
/* Usage: WinHelp.WinHelp(helpfile VARCHAR2,                           */
/*                        command  VARCHAR2,                           */
/*                        data    {VARCHAR2/PLS_INTEGER See Below})    */
/*                                                                     */
/*        command can be one of the following:                         */
/*                                                                     */
/*            'HELP_INDEX'         Help Contents                       */
/*            'HELP_CONTENTS'            "                             */
/*            'HELP_CONTEXT'       Context Key  (See below)            */
/*            'HELP_KEY'           Key Search                          */
/*            'HELP_PARTIALKEY'    Partial Key Search                  */
/*            'HELP_QUIT'          Quit                                */
/*                                                                     */
/*        data contains a string for the key search or a numeric context */
/*        value if using topics.                                       */
/*                                                                     */
/* Winhelp.Winhelp('C:¥ORAWIN95¥TOOLS¥DOC60¥US¥IF60.HLP',                */
/*                      'HELP_PARTIALKEY',                             */
/*                      'ORA_FFI');                                    */
/*                                                                     */
/* The commented sections replace the line below if using HELP_CONTEXT keys */

PACKAGE WinHelp IS
  FUNCTION WinHelp(helpfile IN VARCHAR2,
                   command  IN VARCHAR2,
                   data     IN VARCHAR2)
                   RETURN PLS_INTEGER;
END;
```

```
PACKAGE BODY WinHelp IS
  lh_USER ora_ffi.libHandleType;
  fh_WinHelp ora_ffi.funcHandleType;

  FUNCTION i_WinHelp(funcHandle IN ora_ffi.funcHandleType,
                     hwnd       IN PLS_INTEGER,
                     helpfile   IN OUT VARCHAR2,
                     command    IN PLS_INTEGER,
                     data       IN OUT VARCHAR2)
                     RETURN PLS_INTEGER;

  PRAGMA INTERFACE(C,i_WinHelp,11265);

  FUNCTION WinHelp(helpfile IN VARCHAR2,
                   command  IN VARCHAR2,
                   data     IN VARCHAR2)
                   RETURN PLS_INTEGER
  IS
    hwnd_l     PLS_INTEGER;
    helpfile_l VARCHAR2(512) := helpfile;
    command_l  PLS_INTEGER;
    data_l     VARCHAR2(512) := data;
    rc         PLS_INTEGER;

    FUNCTION Help_Convert(command IN VARCHAR2)
                          RETURN PLS_INTEGER
    IS
    BEGIN
      /* The windows.h definitions for command */

      /* HELP_CONTEXT      0x0001 */
      /* HELP_QUIT         0x0002 */
      /* HELP_INDEX        0x0003 */
      /* HELP_CONTENTS     0x0003 */
      /* HELP_HELPONHELP   0x0004 */
      /* HELP_SETINDEX     0x0005 */
      /* HELP_SETCONTENTS  0x0005 */
      /* HELP_CONTEXTPOPUP 0x0008 */
      /* HELP_FORCEFILE    0x0009 */
      /* HELP_KEY          0x0101 */
      /* HELP_COMMAND      0x0102 */
      /* HELP_PARTIALKEY   0x0105 */
      /* HELP_MULTIKEY     0x0201 */
      /* HELP_SETWINPOS    0x0203 */

      if command = 'HELP_CONTEXT'    then return(1);   end if;
      if command = 'HELP_KEY'        then return(257); end if;
```

```
      if command = 'HELP_PARTIALKEY' then return(261); end if;
      if command = 'HELP_QUIT'       then return(2);   end if;
      /* If nothing else go to the contents page */
      return(3);
    END;

  BEGIN
    hwnd_l :=
    TO_PLS_INTEGER(Get_Item_Property(name_in('SYSTEM.CURSOR_ITEM'),WINDOW_HANDLE));

    command_l := Help_Convert(command);

    rc  := i_WinHelp(fh_WinHelp,
                     hwnd_l,
                     helpfile_l,
                     command_l,
                     data_l);

    RETURN (rc);
  END ;

BEGIN
  BEGIN
    lh_USER := ora_ffi.find_library('USER.EXE');
    EXCEPTION WHEN ora_ffi.FFI_ERROR THEN
    lh_USER := ora_ffi.load_library(NULL,'USER.EXE');
  END ;

  fh_WinHelp :=
ora_ffi.register_function(lh_USER,'WinHelp',ora_ffi.PASCAL_STD);

  ora_ffi.register_parameter(fh_WinHelp,ORA_FFI.C_INT);        /* HWND   */
  ora_ffi.register_parameter(fh_WinHelp,ORA_FFI.C_CHAR_PTR);   /* LPCSTR */
  ora_ffi.register_parameter(fh_WinHelp,ORA_FFI.C_INT);        /* UINT   */
  ora_ffi.register_parameter(fh_WinHelp,ORA_FFI.C_CHAR_PTR);   /* DWORD  */

  ora_ffi.register_return(fh_WinHelp,ORA_FFI.C_INT);           /* BOOL   */

END WinHelp;
```

6.2.5.2 Windows でファイル・オープン・ダイアログを開くための ORA_FFI の使用

```
PACKAGE OraDlg IS
FUNCTION OraMultiFileDlg
(Title IN  VARCHAR2,
```

```
Filter IN  VARCHAR2,
Dir IN  VARCHAR2,
FileString IN OUT VARCHAR2)
RETURN PLS_INTEGER;
FUNCTION OraSingleFileDlg
(Title IN  VARCHAR2,
Filter IN  VARCHAR2,
Dir IN  VARCHAR2,
FileString IN OUT VARCHAR2)
RETURN PLS_INTEGER;
END OraDlg;
PACKAGE BODY OraDlg IS
     lh_ORADLG ora_ffi.libHandleType;
fh_OraMultiFileDlg ora_ffi.funcHandleType;
fh_OraSingleFileDlg ora_ffi.funcHandleType;
FUNCTION i_OraMultiFileDlg
(funcHandle IN ora_ffi.funcHandleType,
Title IN OUT VARCHAR2,
Filter IN OUT VARCHAR2,
Dir IN OUT VARCHAR2,
FileString IN OUT VARCHAR2)
RETURN PLS_INTEGER;
PRAGMA INTERFACE(C,i_OraMultiFileDlg,11265);
FUNCTION OraMultiFileDlg
(Title IN  VARCHAR2,
Filter IN  VARCHAR2,
Dir IN  VARCHAR2,
FileString IN OUT VARCHAR2)
RETURN PLS_INTEGER IS
Title_l VARCHAR2(128)  :=  RPAD(SUBSTR(NVL(Title,'Open'),1,128),128,CHR(0));
Filter_l VARCHAR2(128)  :=  RPAD(SUBSTR(NVL
(Filter,'All Files (*.*)|*.*|'),1,128),128,CHR(0));
Dir_l VARCHAR2(256)  :=  RPAD(SUBSTR(NVL(Dir,' '),1,256),256,CHR(0));
FileString_l VARCHAR2(2000)  :=  RPAD(SUBSTR(NVL(FileString,'
'),1,2000),2000,CHR(0));
rc PLS_INTEGER;
BEGIN
     rc  := i_OraMultiFileDlg(fh_OraMultiFileDlg,
Title_l,
Filter_l,
Dir_l,
FileString_l);
     FileString := FileString_l;
RETURN (rc);
END ;
FUNCTION i_OraSingleFileDlg
(funcHandle IN ora_ffi.funcHandleType,
```

```
Title IN OUT VARCHAR2,
Filter IN OUT VARCHAR2,
Dir IN OUT VARCHAR2,
FileString IN OUT VARCHAR2)
RETURN PLS_INTEGER;
PRAGMA INTERFACE(C,i_OraSingleFileDlg,11265);
FUNCTION OraSingleFileDlg
(Title IN  VARCHAR2,
Filter IN  VARCHAR2,
Dir IN  VARCHAR2,
FileString IN OUT VARCHAR2)
RETURN PLS_INTEGER IS
Title_l VARCHAR2(128)  :=  RPAD(SUBSTR(NVL(Title,'Open'),1,128),128,CHR(0));
Filter_l VARCHAR2(128)  :=  RPAD(SUBSTR(NVL
(Filter,'All Files (*.*)|*.*|'),1,128),128,CHR(0));
Dir_l VARCHAR2(256)  :=  RPAD(SUBSTR(NVL(Dir,' '),1,256),256,CHR(0));
FileString_l VARCHAR2(2000)  :=  RPAD(SUBSTR(NVL(FileString,'
'),1,2000),2000,CHR(0));
rc PLS_INTEGER;
BEGIN
     rc  := i_OraSingleFileDlg(fh_OraSingleFileDlg,
Title_l,
Filter_l,
Dir_l,
FileString_l);
     FileString := FileString_l;
RETURN (rc);
END ;
BEGIN
     BEGIN
     lh_ORADLG := ora_ffi.find_library('ORADLG.DLL');
     EXCEPTION WHEN ora_ffi.FFI_ERROR THEN
     lh_ORADLG := ora_ffi.load_library(NULL,'ORADLG.DLL');
     END ;
     fh_OraMultiFileDlg := ora_ffi.register_function
(lh_ORADLG,'OraMultiFileDlg',ora_ffi.PASCAL_STD);
     ora_ffi.register_parameter(fh_OraMultiFileDlg,ORA_FFI.C_CHAR_PTR);
     ora_ffi.register_parameter(fh_OraMultiFileDlg,ORA_FFI.C_CHAR_PTR);
     ora_ffi.register_parameter(fh_OraMultiFileDlg,ORA_FFI.C_CHAR_PTR);
     ora_ffi.register_parameter(fh_OraMultiFileDlg,ORA_FFI.C_CHAR_PTR);
     ora_ffi.register_return(fh_OraMultiFileDlg,ORA_FFI.C_LONG);
     fh_OraSingleFileDlg := ora_ffi.register_function
(lh_ORADLG,'OraSingleFileDlg',ora_ffi.PASCAL_STD);
     ora_ffi.register_parameter(fh_OraSingleFileDlg,ORA_FFI.C_CHAR_PTR);
     ora_ffi.register_parameter(fh_OraSingleFileDlg,ORA_FFI.C_CHAR_PTR);
     ora_ffi.register_parameter(fh_OraSingleFileDlg,ORA_FFI.C_CHAR_PTR);
     ora_ffi.register_parameter(fh_OraSingleFileDlg,ORA_FFI.C_CHAR_PTR);
```

```
    ora_ffi.register_return(fh_OraSingleFileDlg,ORA_FFI.C_LONG);
END OraDlg;
```

6.2.5.3 STDIN/STDOUT 型インタフェースで UNIX（SUN）実行プログラムをコールするための ORA_FFI の使用

```
/* Copyright (c) 1997 by Oracle Corporation */
/*
   NAME
     ora_pipe_io_spec.sql - Specification for access to Unix Pipe mechanism
   DESCRIPTION
     Demonstration of how to use the ORA_FFI Package to provide access to the
     Unix Pipe C functions.
   PUBLIC FUNCTION(S)
     popen    - Open the Pipe command
     get_line - Get a line of Text from a Pipe
     put_line - Put a line of Text into a Pipe
     pclose   - Close the Pipe
     is_open  - Determine whether the Pipe descriptor is open.
   NOTES

     In Order to use these routines you could write  the following
     PL/SQL Code:

        -- Example of Calls to ora_pipe_io functions
        DECLARE
            stream ora_pipe_io.PIPE;
            buffer VARCHAR2(240);
        BEGIN
            stream := ora_pipe_io.popen('ls -l', ora_pipe_io.READ_MODE);

            loop
                exit when not ora_pipe_io.get_line(stream, buffer, 240);
                :directory.file := buffer;
                down;
            end loop;

            ora_pipe_io.pclose(stream);
        END;

   MODIFIED   (MM/DD/YY)
    smclark    08/05/94 -  Creation
*/

PACKAGE ora_pipe_io is

    /*
```

```
** Arguments to popen.
*/
READ_MODE constant VARCHAR2(1) := 'r';
WRITE_MODE constant VARCHAR2(1) := 'w';

/* ------------- TYPE PIPE ----------- */
/*
** Public Type PIPE - Handle to a Un*x pipe
**
** Do not modify the private members of this type
*/
TYPE PIPE is RECORD
    (file_handle ORA_FFI.POINTERTYPE,
     is_open boolean,
     read_write_mode VARCHAR2(1));


/* ------------ FUNCTION POPEN ----------- */
/*
** Function POPEN -- Open a Un*x pipe command
**
** Given a Unix command to execute and a Pipe read/write mode in which
** to execute the instruction this Function will execute the Command
** and return a handle, of type PIPE, to the resulting Input/Output
** stream.
**
** The command to be executed is limited to 1024 characters.
*/
FUNCTION popen(command in VARCHAR2,
               ctype in VARCHAR2)
RETURN PIPE;


/* ------------ PROCEDURE PCLOSE ----------- */
/*
** Procedure PCLOSE -- Close a pipe
**
** Close a previously opened pipe.
**
** Raises a VALUE_ERROR exception if incorrect arguments are passed.
*/
PROCEDURE pclose(stream in out PIPE);


/* ------------ FUNCTION GET_LINE ----------- */
/*
** Function GET_LINE
```

```
    ** -- Get a line of text into a buffer from the read mode pipe.
    **
    ** Get a line of text from a previously opened pipe.
    **
    ** Raises a VALUE_ERROR exception if incorrect arguments are passed.
    ** For example
    ** if you pass a pipe which has never been opened (using popen)
    */
    FUNCTION get_line(stream in out PIPE,
                      s in out VARCHAR2,
                      n in PLS_INTEGER)
    RETURN BOOLEAN;


    /* ------------ PROCEDURE PUT_LINE ----------- */
    /*
    ** Procedure PUT_LINE -- Put a line of text into a a write mode pipe.
    **
    ** Put a line of text into a previously opened pipe.
    **
    ** Raises a VALUE_ERROR exception if incorrect arguments are passed.
    ** For example
    ** if you pass a pipe which has never been opened (using popen)
    **
    ** The Internal buffer for the string to write is limited to 2048 bytes
    */
    PROCEDURE put_line(stream in out PIPE,
                       s in VARCHAR2);


    /* ------------ FUNCTION IS_OPEN ----------- */
    /*
    ** Function IS_OPEN -- Determines whether a pipe is open.
    **
    ** Returns TRUE if the pipe is open, FALSE if the pipe is closed.
    */
    FUNCTION is_open(stream in PIPE)
    RETURN BOOLEAN;
END;

/*    ora_pipe_io_body.sql - Body of Package for access to Unix Pipe mechanism
   DESCRIPTION
     Demonstration of how to use the ORA_FFI Package to provide access to the
     Unix Pipe C functions.
   PUBLIC FUNCTION(S)
     popen    - Open the Pipe command
     get_line - Get a line of Text from a Pipe
```

```
     put_line - Put a line of Text into a Pipe
     pclose   - Close the Pipe
     is_open  - Determine whether the Pipe descriptor is open.
   PRIVATE FUNCTION(S)
     icd_popen, icd_fgets, icd_fputs, icd_pclose
   NOTES
   MODIFIED   (MM/DD/YY)
    smclark    11/02/94 -  Modified for production release changes to ORA_FFI.
    smclark    08/05/94 -  Creation
*/

 PACKAGE BODY ora_pipe_io is
    lh_libc   ora_ffi.libHandleType;
    fh_popen  ora_ffi.funcHandleType;
    fh_pclose ora_ffi.funcHandleType;
    fh_fgets  ora_ffi.funcHandleType;
    fh_fputs  ora_ffi.funcHandleType;

    /* ------------ FUNCTION ICD_POPEN ----------- */
    /*
    ** Function ICD_POPEN -- Interface routine to C function popen
    **
    ** This function acts as the interface to the popen function in
    ** libc.
    */
    FUNCTION icd_popen(funcHandle in ora_ffi.funcHandleType,
                       command in out VARCHAR2,
                       ctype in out VARCHAR2)
    return ORA_FFI.POINTERTYPE;

    pragma interface(c, icd_popen, 11265);


    /* ------------ PROCEDURE ICD_PCLOSE ----------- */
    /*
    ** Function ICD_PCLOSE -- Interface routine to C function pclose
    **
    ** This function acts as the interface to the pclose function in
    ** libc.
    */
    PROCEDURE icd_pclose(funcHandle in ora_ffi.funcHandleType,
                         stream in out ORA_FFI.POINTERTYPE);

    pragma interface(c, icd_pclose, 11265);


    /* ------------ FUNCTION ICD_FGETS ----------- */
```

```
/*
** Function ICD_FGETS -- Interface routine to C function fgets
**
** This function acts as the interface to the fgets function in
** libc.
*/
FUNCTION icd_fgets(funcHandle in ora_ffi.funcHandleType,
                   s in out VARCHAR2, n in PLS_INTEGER,
                   stream in out ORA_FFI.POINTERTYPE)
RETURN ORA_FFI.POINTERTYPE;

pragma interface(c, icd_fgets, 11265);


/* ------------ FUNCTION ICD_FPUTS ----------- */
/*
** Function ICD_FPUTS -- Interface routine to C function fputs
**
** This function acts as the interface to the fputs function in
** libc.
*/
PROCEDURE icd_fputs(funcHandle in ora_ffi.funcHandleType,
                    s in out VARCHAR2,
                    stream in out ORA_FFI.POINTERTYPE);

pragma interface(c, icd_fputs, 11265);


/* ------------ FUNCTION POPEN ----------- */
/*
** Function POPEN -- Open a Un*x pipe command
*/
FUNCTION popen(command in VARCHAR2,
               ctype in VARCHAR2)
RETURN PIPE is

    /*
    ** Take a copy of the arguments because we need to pass them
    ** IN OUT to icd_popen, but we really don't want people to have
    ** to call our routines in the same way.
    */
    cmd varchar2(1024) := command;
    cmode varchar2(1) := ctype;

    stream PIPE;
BEGIN
    if (cmode not in (READ_MODE, WRITE_MODE))
```

```
       or (cmode is NULL)
       or (cmd is NULL)
    then
        raise VALUE_ERROR;
    end if;

    stream.file_handle := icd_popen(fh_popen, cmd, cmode);
    stream.is_open := TRUE;
    stream.read_write_mode := ctype;
    return(stream);
END popen;


/* ------------ PROCEDURE PCLOSE ----------- */
/*
** Procedure PCLOSE -- Close a pipe
*/
PROCEDURE pclose(stream in out PIPE) is
BEGIN
    icd_pclose(fh_pclose, stream.file_handle);
    stream.is_open := FALSE;
END pclose;


/* ------------ FUNCTION GET_LINE ----------- */
/*
** Function GET_LINE -- Get a line of text into a buffer
** from the read mode pipe.
*/
FUNCTION get_line(stream in out PIPE,
                  s in out VARCHAR2, n in PLS_INTEGER)
RETURN BOOLEAN is
    buffer ORA_FFI.POINTERTYPE;
BEGIN
    if (n <= 0)
       or (stream.is_open = FALSE)
       or (stream.is_open is NULL)
       or (stream.read_write_mode <> READ_MODE)
    then
        raise VALUE_ERROR;
    end if;

    /*
    ** Initialise the Buffer area to reserve the correct amount of space.
    */

    s := rpad(' ', n);
```

```
        buffer := icd_fgets(fh_fgets, s, n, stream.file_handle);

        /*
        ** Determine whether a NULL pointer was returned.
        */
        return (ora_ffi.is_null_ptr(buffer) = FALSE);
    END get_line;


    /* ------------ PROCEDURE PUT_LINE ----------- */
    /*
    ** Procedure PUT_LINE -- Put a line of text into a a write mode pipe.
    */
    PROCEDURE put_line(stream in out PIPE,
                       s in VARCHAR2) is
        buffer varchar2(2048) := s;
    BEGIN
        if (stream.is_open = FALSE)
           or (stream.is_open is NULL)
           or (stream.read_write_mode <> WRITE_MODE)
        then
            raise VALUE_ERROR;
        end if;

        icd_fputs(fh_fputs, buffer, stream.file_handle);
        buffer := chr(10);
        icd_fputs(fh_fputs, buffer, stream.file_handle);
    END put_line;


    /* ------------ FUNCTION IS_OPEN ----------- */
    /*
    ** Function IS_OPEN -- Determines whether a pipe is open.
    */
    FUNCTION is_open(stream in PIPE)
    RETURN BOOLEAN is
    BEGIN
        return(stream.is_open);
    END is_open;

BEGIN
    /*
    ** Declare a library handle as libc.  (Internal so NULL,NULL)
    */
    lh_libc := ora_ffi.load_library(NULL, NULL);
    if ora_ffi.is_null_ptr(lh_libc) then
```

```
    raise VALUE_ERROR;
end if;

/*
** Register the popen function, it's return type and arguments.
*/
fh_popen := ora_ffi.register_function(lh_libc, 'popen');
if ora_ffi.is_null_ptr(fh_popen) then
    raise VALUE_ERROR;
end if;
ora_ffi.register_return(fh_popen, ORA_FFI.C_DVOID_PTR);
ora_ffi.register_parameter(fh_popen, ORA_FFI.C_CHAR_PTR);
ora_ffi.register_parameter(fh_popen, ORA_FFI.C_CHAR_PTR);

/*
** Register the pclose function, it's return type and arguments.
*/
fh_pclose := ora_ffi.register_function(lh_libc, 'pclose');
if ora_ffi.is_null_ptr(fh_pclose) then
    raise VALUE_ERROR;
end if;
ora_ffi.register_return(fh_pclose, ORA_FFI.C_VOID);
ora_ffi.register_parameter(fh_pclose, ORA_FFI.C_DVOID_PTR);

/*
** Register the fgets function, it's return type and arguments.
*/
fh_fgets := ora_ffi.register_function(lh_libc, 'fgets');
if ora_ffi.is_null_ptr(fh_fgets) then
    raise VALUE_ERROR;
end if;
ora_ffi.register_return(fh_fgets, ORA_FFI.C_DVOID_PTR);
ora_ffi.register_parameter(fh_fgets, ORA_FFI.C_CHAR_PTR);
ora_ffi.register_parameter(fh_fgets, ORA_FFI.C_INT);
ora_ffi.register_parameter(fh_fgets, ORA_FFI.C_DVOID_PTR);

/*
** Register the fputs function, it's return type and arguments.
*/
fh_fputs := ora_ffi.register_function(lh_libc, 'fputs');
if ora_ffi.is_null_ptr(fh_fputs) then
    raise VALUE_ERROR;
end if;
ora_ffi.register_return(fh_fputs, ORA_FFI.C_VOID);
ora_ffi.register_parameter(fh_fputs, ORA_FFI.C_CHAR_PTR);
ora_ffi.register_parameter(fh_fputs, ORA_FFI.C_DVOID_PTR);
```

```
END ora_pipe_io;
```

6.3 Form Builder アプリケーションの作成と修正を行うためのオープン API の使用

この項では、Form Builder アプリケーションを作成および変更するための非インタラクティブなプログラミング方法を説明します。この項では次の項目について説明します。

- 6.3.1 項「オープン API」
- 6.3.2 項「オープン API 使用のガイドライン」
- 6.3.3 項「オープン API の使用」
- 6.3.4 項「オープン API の例」

6.3.1 オープン API

オープン API は、非インタラクティブな環境においてフォーム・モジュールの作成または変更を行うための機能と柔軟性を求める C/C++ 開発者のための Form Builder 拡張機能です。

注意 : オープン API を使用する前に、Form Builder オブジェクトおよびそのプロパティとリレーションについて完全に理解してください。

6.3.1.1 オープン API を利用する場合

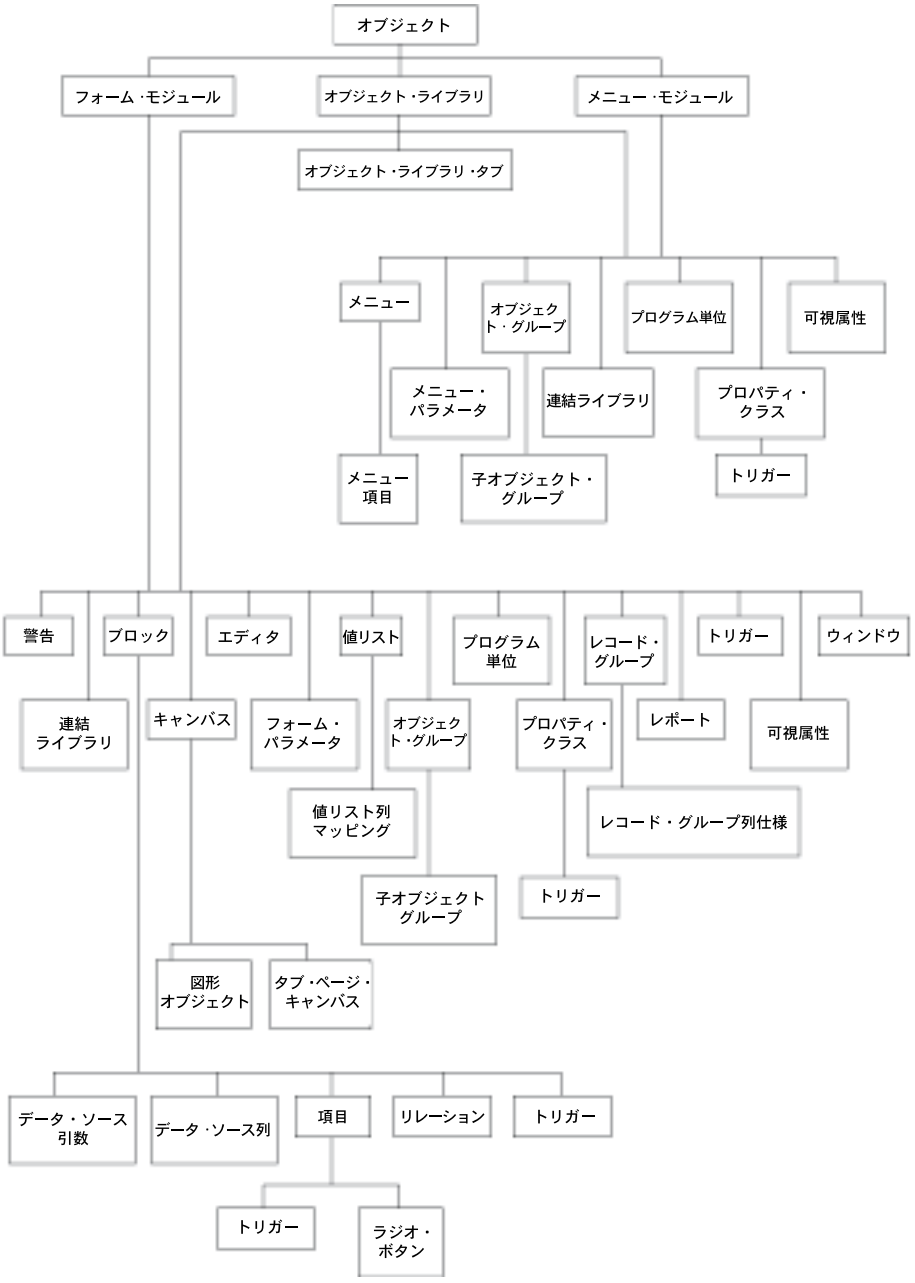
開発の変更内容を多数のフォーム・モジュールにすばやく反映させたい場合は、オープン API を使用します。たとえば、オープン API を使用して、アプリケーションのルック & フィールを現行の企業規格に更新します。この場合、フォーム・モジュールの更新は数百にのぼることもあります。

オープン API には他にも次のような使用方法があります。

- 一連のフォームをコンパイルします。
- 依存性情報を収集します。
- 独自のドキュメントを書き込みます。

6.3.1.2 オープン API ヘッダー・ファイル

オープン API は、1 つの Form Builder オブジェクトに対して 1 つの C ヘッダー・ファイルから構成されています。Form Builder オブジェクトは 34 個あります（図を参照）。これらのオブジェクトは、設計時に使い慣れている Form Builder オブジェクトと対応しています。各ヘッダー・ファイルには、Form Builder オブジェクトの作成および操作に使用する関数やマクロが複数含まれています。


オブジェクト
フォーム ·モジュール
オブジェクト ·ライブラリ
メニュー ·モジュール
オブジェクト ·ライブラリ ·タブ
メニュー
オブジェクト・グループ
プログラム単位
可視属性
メニュー・パラメータ
連結ライブラリ
プロパティ・クラス
メニュー項目
子オブジェクト・グループ
トリガー
警告
ブロック
エディタ
値リスト
プログラム単位
レコード・グループ
トリガー
ウィンドウ
連結ライブラリ
キャンバス
フォーム・パラメータ
オブジェクト ·グループ
プロパティ・クラス
レポート
可視属性
値リスト列マッピング
レコード・グループ列仕様
子オブジェクトグループ
トリガー
図形オブジェクト
タブ ·ページ · キャンバス
データ・ソース引数
データ ·ソース列
項目
リレーション
トリガー
トリガー
ラジオ・ボタン

6.3.1.3 オープン API プロパティ

オープン API では、オブジェクト・プロパティを設定して Form Builder オブジェクトを操作します。

オープン API プロパティには、D2FP_FONT_NAM をはじめとする一意の名前があります。これらのプロパティは、設計時に使い慣れている Form Builder プロパティと対応しています。

プロパティには次のものがあります。Boolean、Text、Number、Object、または Blob です。

次の表では、共通項目プロパティおよびそれらと対応するオープン API 等価プロパティをリストします。

オープン API プロパティ	Form Builder（設計時）プロパティ
D2FP_ACCESS_KEY	アクセス・キー
D2FP_BEVEL_STY	凹凸
D2FP_CNV_NAM	キャンバス
D2FP_ENABLED	変更可
D2FP_FONT_NAM	フォント名
D2FP_HEIGHT	幅 / 高さ
D2FP_X_POS	X 位置
D2FP_Y_POS	Y 位置

6.3.1.4 オープン API ファンクションとマクロ

オープン API ファンクションおよびマクロを使用して、オブジェクト・プロパティの作成、破棄、複製、サブクラス化、取得、設定を行うことができます。

たとえば、項目のフォント・サイズを決めるには、D2FITMG_FONT_SIZ マクロを使用します。

```
d2fitmg_font_siz(ctx, obj, val);
```

このマクロは、項目オブジェクトのフォント・サイズ・プロパティの値をタイプ番号として戻します。

テキスト項目プロパティを設定するには、D2FITMST_SETTEXTPROP ファンクションを使用します。

```
d2fitmst_SetTextProp(d2fctx *pd2fctx, d2fitm *pd2fitm, ub2 pnum,text *prp );
```

このファンクションは、指定の項目テキスト・プロパティの値を設定します。ポインタを pd2fctx の内容に、項目を pd2fitm に、プロパティ番号を pnum に、ハンドルを prp のテキスト値に指定します。

6.3.2 オープン API 使用のガイドライン

オープン API を動作させるとき、これらのガイドラインを考慮してください。

項目	推奨事項
ファイル・バックアップ	オープン API は非インタラクティブです。妥当性チェックおよびエラー・チェックはサポートされていません。オープン API を使用する前に、フォーム・モジュール（.FMB）をバックアップしてください。
リレーション・オブジェクトの作成	リレーション・オブジェクトの作成時に、次のことを行う必要があります。 ■ オブジェクトを作成します。 ■ リレーション・オブジェクト・プロパティを設定します。 ■ d2frelup_Update ファンクションをコールして、オブジェクトのインスタンスを生成します。

6.3.3 オープン API の使用

この項では、オープン API を使用して Form Builder モジュールを作成および変更するための詳細な手順を説明します。

6.3.3.1 オープン API を使用したモジュールの作成と変更

Form Builder モジュールを作成または変更するには、次のようにします。

1. 適切な C ヘッダー・ファイルを C ソース・コードに挿入します。
2. C ソース・コードにある任意の API をコールします。
   - コンテキスト構造を初期化します。
   - ロード・ファンクション・コールを行って、既存のフォーム・モジュール、メニュー・モジュール、またはオブジェクト・ライブラリをオープンします。
   - 所定のオープン・フォーム API ファンクション・コールを行って、既存のデータベースへの接続をはじめとする任意の操作を、必要に応じて行います。
   - 適切な CompileFile() ファンクションを使用して、.FMX または .MMX コンパイル・フォームを生成します。
   - 必須のファンクション・コールを行って、対応モジュール（たとえば、フォーム・モジュールの d2ffmdsv_Save()、メニュー・モジュールの d2fmmdsv_Save()、またはオブジェクト・ライブラリの d2folbsv_Save() など）を保存します。
   - 最後に、コンテキスト破棄ファンクション d2fctxde_Destroy() をコールして、オープン・フォーム API コンテキストを破棄します。このファンクションのコールは最後に行う必要があることに注意してください。

3. ソース・ファイルをオープン API ライブラリ（ ifd2f60.lib）にリンクします。
4. ファイルをコンパイルして実行プログラム（.EXE ファイル）を作成します。
5. 実行プログラムを実行してフォーム・モジュール（.FMB）の作成または変更を行います。

6.3.4 オープン API の例

この項では、複数の例を示しながらオープン API の使用方法を説明します。

6.3.4.1 オープン API を使用したモジュールの変更

```
/*
This example determines if the Form Builder object is a subclassed object and
returns the file path of the parent to NULL if the object is subclassed. This sample
only processes the following object types:  form level triggers, alerts, blocks,
items, item level triggers, radio buttons, and block level triggers.  Use a similar
method to process other object types.
*/
#include <stdio.h>
#include <string.h>
#include <windows.h>
#include <d2ferr.h>
#include <d2fctx.h>
#include <d2ffmd.h>
#include <d2fblk.h>
#include <d2fitm.h>
#include <d2falt.h>
#include <d2ftrg.h>
#include <d2frdb.h>
#define BUFSIZE 128
int WINAPI WinMain(HANDLE hInstance,
                   HANDLE hPrevInstance,
                   LPSTR lpszCommandLine,
                   int cmdShow)
{
    d2fctx*   pd2fctx;
    d2ffmd*   pd2ffmd;
    d2fblk*   pd2fblk;
    d2fitm*   pd2fitm;
    d2fctxa   d2fctx_attr;
    d2fstatus status;
    d2falt*   pd2falt;
    d2ftrg*   pd2ftrg;
    d2frdb*   pd2frdb;
    int counter;
```

```
char buf[BUFSIZE];
char* form_name=(char*)0;
/* Get the form name from the command line */
strncpy(buf, lpszCommandLine, BUFSIZE);
form_name = strtok(buf, ".");
/* Initialize the attribute mask */
d2fctx_attr.mask_d2fctxa = 0;
/* for MS Windows-only attributes */
d2fctx_attr.d2fihnd_d2fctxa = hInstance;
d2fctx_attr.d2fphnd_d2fctxa = hPrevInstance;
d2fctx_attr.d2fcmsh_d2fctxa = cmdShow;
/* Create the API context */
status = d2fctxcr_Create(&pd2fctx, &d2fctx_attr);
/* Load the form */
status = d2ffmdld_Load(pd2fctx, &pd2ffmd, form_name, FALSE) ;
if (status == D2FS_D2FS_SUCCESS)
{
  /*** Process Form Level Trigger Objects ***/
  for(status = d2ffmdg_trigger(pd2fctx,pd2ffmd,&pd2ftrg);
      pd2ftrg != NULL;
      status = d2ftrgg_next(pd2fctx,pd2ftrg,&pd2ftrg))
  {
    if (d2ftrgis_IsSubclassed(pd2fctx,pd2ftrg) == D2FS_YES)
        d2ftrgs_par_flpath(pd2fctx,pd2ftrg,NULL);
  }
  /*** Process Alert Objects ***/
  for(status = d2ffmdg_alert(pd2fctx,pd2ffmd,&pd2falt);
      pd2falt != NULL;
      status = d2faltg_next(pd2fctx,pd2falt,&pd2falt))
  {
    if (d2faltis_IsSubclassed(pd2fctx,pd2falt) == D2FS_YES)
        d2falts_par_flpath(pd2fctx,pd2falt,NULL);
  }
  /*** Process Block Objects ***/
  for(status = d2ffmdg_block(pd2fctx,pd2ffmd,&pd2fblk);
      pd2fblk != NULL;
      status = d2fblkg_next(pd2fctx,pd2fblk,&pd2fblk))
  {
    if (d2fblkis_IsSubclassed(pd2fctx,pd2fblk) == D2FS_YES)
       d2fblks_par_flpath(pd2fctx,pd2fblk,NULL);
  }
  /* Process Item Objects */
  for(status = d2fblkg_item(pd2fctx,pd2fblk,&pd2fitm);
      pd2fitm != NULL;
      status = d2fitmg_next(pd2fctx,pd2fitm,&pd2fitm))
  {
    if (d2fitmis_IsSubclassed(pd2fctx,pd2fitm) == D2FS_YES)
```

```
      d2fitms_par_flpath(pd2fctx,pd2fitm,NULL);
   /* Process Item Level Trigger Objects */
    for(status = d2fitmg_trigger(pd2fctx,pd2fitm,&pd2ftrg);
        pd2ftrg != NULL;
        status = d2ftrgg_next(pd2fctx,pd2ftrg,&pd2ftrg))
    {
        if (d2ftrgis_IsSubclassed(pd2fctx,pd2ftrg)==D2FS_YES)
        {
          d2ftrgs_par_flpath(pd2fctx,pd2ftrg,NULL);
          printf("item trigger is Subclassed¥n");
        }
        else if  (d2ftrgis_IsSubclassed(pd2fctx,
                                        pd2ftrg)==D2FS_NO)
        printf("item trigger is NOT Subclassed¥n");
    }
 /* Process Radio Button Objects *
   for(status = d2fitmg_rad_but(pd2fctx,pd2fitm,&pd2frdb);
       pd2frdb != NULL;
       status = d2frdbs_next(pd2fctx,pd2frdb,&pd2frdb))
   {
       if (d2frdbis_IsSubclassed(pd2fctx,pd2frdb)==D2FS_YES
       {
         d2frdbs_par_flpath(pd2fctx,pd2frdb,NULL);
         printf("radio button is Subclassed¥n");
       }
       else if (d2frdbis_IsSubclassed(pd2fctx,
                                      pd2frdb)==D2FS_NO)
       printf("radio button is NOT Subclassed¥n");
  }
 }
 /* Process Block Level Trigger Objects */
 for(status = d2fblkg_trigger(pd2fctx,pd2fblk,&pd2ftrg);
     pd2ftrg != NULL;
     status = d2ftrgg_next(pd2fctx,pd2ftrg,&pd2ftrg))
 {
     if (d2ftrgis_IsSubclassed(pd2fctx,pd2ftrg) == D2FS_YES)
     {
       d2ftrgs_par_flpath(pd2fctx,pd2ftrg,NULL);
       printf("block trigger is Subclassed¥n");
     }
     else if (d2ftrgis_IsSubclassed(pd2fctx,
                                    pd2ftrg)==D2FS_NO)
     printf("block trigger is NOT Subclassed¥n");
 }

/* Save out the form */
d2ffmdsv_Save(pd2fctx, pd2ffmd, (text *)0, FALSE) ;
```

```
     /* Generate the forms executable (fmx) */
     d2ffmdcf_CompileFile(pd2fctx, pd2ffmd ) ;
     /* Destroy the API Context */
     d2fctxde_Destroy(pd2fctx) ;
    }
}
```

6.3.4.2 オープン API を使用したモジュールの作成

```
/*
This example creates a master-detail form based on the dept and emp database tables
owned by the user scott.  The master contains the following fields: empno, ename,
job, sal, and deptno.  The detail contains the following fields deptno, dname, and
loc.  The join condition is deptno.
*/
#include<stdio.h>
#include<string.h>
#include<windows.h>
#include<d2fctx.h>
#include<d2ffmd.h>
#include<d2ffpr.h>
#include<d2fob.h>
#include<d2fcnv.h>
#include<d2ftrg.h>
#include<d2blk.h>
#include<d2fitm.h>
#include<d2fwin.h>
#include<d2frel.h>
#define D2FS_SUCCESS 0
#define FAIL 1
#define BUFSIZE 128
#define WBP_TXT "null;¥n"
int WINAPI WinMain(HANDLE hInstance,
                   HANDLE hPrevInstance,
                   LPSTR lpszCommandLine,
                   int cmdShow)
{
d2fctx  *pd2fctx;
d2ffmd  *pd2ffmd;
d2fcnv  *pd2fcnv;
d2fwin  *pd2fwin;
d2fblk  *pempblk;
d2fblk  *pdeptblk;
d2frel  *pd2frel;
d2fitm  *pEempnoitm;
d2fitm  *pEenameitm;
d2fitm  *pEjobitm;
```

```
d2fitm  *pEsalitm;
d2fitm  *pEdeptnoitm;
d2fitm  *pDdeptnoitm;
d2fitm  *pDdnameitm;
d2fitm  *pDlocitm;
text  *name = (text *)0;
text  *form_name = (text *)0;
d2fctxa  d2fctx_attr;
d2fstatus retval;
char buf[BUFSIZE];
/* Get form name */
strncpy(buf, "empdept", BUFSIZE);
form_name = (text*)strtok(buf, ".");
/* Initialize the attribute mask */
d2fctx_attr.mask_d2fctxa = 0;
/* for MS Windows-only attributes */
d2fctx_attr.d2fihnd_d2fctxa = hInstance;
d2fctx_attr.d2fphnd_d2fctxa = hPrevInstance;
d2fctx_attr.d2fcmsh_d2fctxa = cmdShow;
/* Create the API context */
status = d2fctxcr_Create(&pd2fctx, &d2fctx_attr);
/* Create the context */
d2fctxcn_Connect(pd2fctx, (text*)"scott/tiger@test");
/* Create the form */
d2ffmdcr_Create(pd2fctx, &pd2ffmd, form_name);
/* Create a window */
d2fwincr_Create(pd2fctx,pd2ffmd,&pd2fwin,(text*)"MYWIN");
/*** Create Canvas and set canvas-related properties ***/
/* Create a canvas */
d2fcnvcr_Create(pd2fctx, pd2ffmd, &pd2fcnv, (text*)"MYCANVAS");
/* Set viewport width */
d2fcnvs_vprt_wid(pd2fctx, pd2fcnv, 512);
/* Set viewport height */
d2fcnvs_vprt_hgt(pd2fctx, pd2fcnv, 403);
/* Set window */
dwfcnvs_wnd_obj(pd2fctx, pd2fcnv, pd2fwin);
/* Set viewport X-position */
d2fcnvs_vprt_x_pos(pd2fctx, pd2fcnv, 0);
/* Set viewport Y-position */
d2fcnvs_vprt_y_pos(pd2fctx, pd2fcnv, 0);
/* Set width */
d2fcnvs_width(pd2fctx, pd2fcnv, 538)
/* Set height */
d2fcnvs_height(pd2fctx, pd2fcnv, 403)
/*** Create Emp block and set block-related properties ***/
/* Create block */
d2fblkcr_Create(pd2fctx, pd2ffmd, &pempblk, (text*)"EMP");
```

```
/* Set to database block */
d2fblks_db_blk(pd2fctx, pempblk, TRUE);
/* Set query data source to Table */
d2fblks_qry_dat_src_typ(pd2fctx, pempblk, D2FC_QRDA_TABLE);
/* Set query data source name to EMP table */
d2fblks_qry_dat_src_nam(pd2fctx, pempblk, "EMP");
/* Set DML data source type to Table */
d2fblks_dml_dat_typ(Pd2fctx, pempblk, D2FC_DMDA_TABLE);
/* Set DML data source name to EMP table */
d2fblks_dml_dat_nam(pd2fctx, pempblk, (text*)"EMP");
/*** Create Dept block and set block-related properties ***/
/* Create block */
d2fblkcr_Create(pd2fctx, pd2ffmd, &pdeptblk, (text*)"DEPT");
/* Set to database block */
d2fblks_db_blk(pd2fctx, pdeptblk, TRUE);
/* Set query data source to Table */
d2fblks_qry_dat_src_typ(pd2fctx, pdeptblk, D2FC_QRDA_TABLE);
/* Set query data source name to EMP table */
d2fblks_qry_dat_src_nam(pd2fctx, pdeptblk, "DEPT");
/* Set DML data source type to Table */
d2fblks_dml_dat_typ(Pd2fctx, pdeptblk, D2FC_DMDA_TABLE);
/* Set DML data source name to EMP table */
d2fblks_dml_dat_nam(pd2fctx, pdeptblk, (text*)"DEPT");
/*** Create empno item and item-related properties ***/
/* Create item */
d2fitmcr_Create(pd2fctx, pempblk, &pEempnoitm, (text*)"EMPNO");
/* Set item type */
d2fitms_itm_type(pd2fctx, pEempnoitm, D2FC_ITTY_TI);
/* Set Enable property */
d2fitms_enabled(pd2fctx, pEempnoitm, TRUE);
/* Set item (keyboard) navigable property */
d2fitms_kbrd_navigable(pd2fctx, pEempnoitm, TRUE);
/* Set item Data Type property */
d2fitms_dat_typ(pd2fctx, pEempnoitm, D2FC_DATY_NUMBER);
/* Set item Max Length property */
d2fitms_max_len(pd2fctx, pEempnoitm, 6);
/* Set item Required property */
d2fitms_required(pd2fctx, pEempnoitm, TRUE);
/* Set Distance Between Records property */
d2fitms_dist_btwn_recs(pd2fctx, pEempnoitm, 0);
/* Set Database block(Database Item) property */
d2fitms_db_itm(pd2fctx, pEempnoitm, TRUE);
/* Set Query Allowed */
d2fitms_qry_allowed(pd2fctx, pEempnoitm, TRUE);
/* Set Query Length */
d2fitms_qry_len(pd2fctx, pEempnoitm, 6);
/* Set Update Allowed */
```

```
d2fitms_updt_allowed(pd2fctx, pEempnoitm, TRUE);
/* Set Item Displayed (Visible) */
d2fitms_visible(pd2fctx, pEempnoitm, TRUE);
/* Set Item Canvas property */
d2fitms_cnv_obj(pd2fctx, pEempnoitm, pd2fcnv);
/* Set Item X-position */
d2fitms_x_pos(pd2fctx, pEempnoitm, 32);
/* Set Item Y-position */
d2fitms_y_pos(pd2fctx, pEempnoitm, 50);
/* Set Item Width */
d2fitms_width(pd2fctx, pEempnoitm, 51);
/* Set Item Height */
d2fitms_height(pd2fctx, pEempnoitm, 17);
/* Set Item Bevel */
d2fitms_bevel(pd2fctx, pEempnoitm, D2FC_BEST_LOWERED);
/* Set item Hint */
d2fitms_hint(pd2fctx, PEempnoitm, (text*)"Enter value for :EMPNO");
/*** Create Ename item and item-related properties ***/
/* Create item */
d2fitmcr_Create(pd2fctx, pempblk, &pEenameitm, (text*)"ENAME");
/* Set item type */
d2fitms_itm_type(pd2fctx, pEenameitm, D2FC_ITTY_TI);
/* Set Enable property */
d2fitms_enabled(pd2fctx, pEenameitm, TRUE);
/* Set item (keyboard) navigable property */
d2fitms_kbrd_navigable(pd2fctx, pEenameitm, TRUE);
/* Set item Data Type property */
d2fitms_dat_typ(pd2fctx, pEenameitm, D2FC_DATY_CHAR);
/* Set item Max Length property */
d2fitms_max_len(pd2fctx, pEenameitm, 10);
/* Set Distance Between Records property */
d2fitms_dist_btwn_recs(pd2fctx, pEenameitm, 0);
/* Set Database block(Database Item) property */
d2fitms_db_itm(pd2fctx, pEenameitm, TRUE);
/* Set Query Allowed */
d2fitms_qry_allowed(pd2fctx, pEenameitm, TRUE);
/* Set Query Length */
d2fitms_qry_len(pd2fctx, pEenameitm, 10);
/* Set Update Allowed */
d2fitms_updt_allowed(pd2fctx, pEenameitm, TRUE);
/* Set Item Displayed (Visible) */
d2fitms_visible(pd2fctx, pEenameitm, TRUE);
/* Set Item Canvas property */
d2fitms_cnv_obj(pd2fctx, pEenameitm, pd2fcnv);
/* Set Item X-position */
d2fitms_x_pos(pd2fctx, pEenameitm, 83);
/* Set Item Y-position */
```

```
d2fitms_y_pos(pd2fctx, pEenameitm, 50);
/* Set Item Width */
d2fitms_width(pd2fctx, pEenameitm, 77);
/* Set Item Height */
d2fitms_height(pd2fctx, pEenameitm, 17);
/* Set Item Bevel */
d2fitms_bevel(pd2fctx, pEenameitm, D2FC_BEST_LOWERED);
/* Set item Hint */
d2fitms_hint(pd2fctx, PEenameitm, (text*)"Enter value for :ENAME");
/*** Create JOB item and item-related properties ***/
/* Create item */
d2fitmcr_Create(pd2fctx, pempblk, &pEjobitm, (text*)"JOB");
/* Set item type */
d2fitms_itm_type(pd2fctx, pEjobitm, D2FC_ITTY_TI);
/* Set Enable property */
d2fitms_enabled(pd2fctx, pEjobitm, TRUE);
/* Set item (keyboard) navigable property */
d2fitms_kbrd_navigable(pd2fctx, pEjobitm, TRUE);
/* Set item Data Type property */
d2fitms_dat_typ(pd2fctx, pEjobitm, D2FC_DATY_CHAR);
/* Set item Max Length property */
d2fitms_max_len(pd2fctx, pEjobitm, 9);
/* Set Distance Between Records property */
d2fitms_dist_btwn_recs(pd2fctx, pEjobitm, 0);
/* Set Database block(Database Item) property */
d2fitms_db_itm(pd2fctx, pEjobitm, TRUE);
/* Set Query Allowed */
d2fitms_qry_allowed(pd2fctx, pEjobitm, TRUE);
/* Set Query Length */
d2fitms_qry_len(pd2fctx, pEjobitm, 9);
/* Set Update Allowed */
d2fitms_updt_allowed(pd2fctx, pEjobitm, TRUE);
/* Set Item Displayed (Visible) */
d2fitms_visible(pd2fctx, pEjobitm, TRUE);
/* Set Item Canvas property */
d2fitms_cnv_obj(pd2fctx, pEjobitm, pd2fcnv);
/* Set Item X-position */
d2fitms_x_pos(pd2fctx, pEjobitm, 160);
/* Set Item Y-position */
d2fitms_y_pos(pd2fctx, pEjobitm, 50);
/* Set Item Width */
d2fitms_width(pd2fctx, pEjobitm, 70);
/* Set Item Height */
d2fitms_height(pd2fctx, pEjobitm, 17);
/* Set Item Bevel */
d2fitms_bevel(pd2fctx, pEjobitm, D2FC_BEST_LOWERED);
/* Set item Hint */
```

```
d2fitms_hint(pd2fctx, PEjobitm, (text*)"Enter value for :JOB");
/*** Create SALARY item and item-related properties ***/
/* Create item */
d2fitmcr_Create(pd2fctx, pempblk, &pEsalitm, (text*)"SAL");
/* Set item type */
d2fitms_itm_type(pd2fctx, pEsalitm, D2FC_ITTY_TI);
/* Set Enable property */
d2fitms_enabled(pd2fctx, pEsalitm, TRUE);
/* Set item (keyboard) navigable property */
d2fitms_kbrd_navigable(pd2fctx, pEsalitm, TRUE);
/* Set item Data Type property */
d2fitms_dat_typ(pd2fctx, pEsalitm, D2FC_DATY_NUMBER);
/* Set item Max Length property */
d2fitms_max_len(pd2fctx, pEsalitm, 9);
/* Set Distance Between Records property */
d2fitms_dist_btwn_recs(pd2fctx, pEsalitm, 0);
/* Set Database block(Database Item) property */
d2fitms_db_itm(pd2fctx, pEsalitm, TRUE);
/* Set Query Allowed */
d2fitms_qry_allowed(pd2fctx, pEsalitm, TRUE);
/* Set Query Length */
d2fitms_qry_len(pd2fctx, pEsalitm, 9);
/* Set Update Allowed */
d2fitms_updt_allowed(pd2fctx, pEsalitm, TRUE);
/* Set Item Displayed (Visible) */
d2fitms_visible(pd2fctx, pEsalitm, TRUE);
/* Set Item Canvas property */
d2fitms_cnv_obj(pd2fctx, pEsalitm, pd2fcnv);
/* Set Item X-position */
d2fitms_x_pos(pd2fctx, pEsalitm, 352);
/* Set Item Y-position */
d2fitms_y_pos(pd2fctx, pEsalitm, 50);
/* Set Item Width */
d2fitms_width(pd2fctx, pEsalitm, 70);
/* Set Item Height */
d2fitms_height(pd2fctx, pEsalitm, 17);
/* Set Item Bevel */
d2fitms_bevel(pd2fctx, pEsalitm, D2FC_BEST_LOWERED);
/* Set item Hint */
d2fitms_hint(pd2fctx, PEsalitm, (text*)"Enter value for :SAL");
/*** Create DEPTNO item and item-related properties ***/
/* Create item */
d2fitmcr_Create(pd2fctx, pempblk, &pEdeptnoitm, (text*)"DEPTNO");
/* Set item type */
d2fitms_itm_type(pd2fctx, pEdeptnoitm, D2FC_ITTY_TI);
/* Set Enable property */
d2fitms_enabled(pd2fctx, pEdeptnoitm, TRUE);
```

```
/* Set item (keyboard) navigable property */
d2fitms_kbrd_navigable(pd2fctx, pEdeptnoitm, TRUE);
/* Set item Data Type property */
d2fitms_dat_typ(pd2fctx, pEdeptnoitm, D2FC_DATY_NUMBER);
/* Set item Max Length property */
d2fitms_max_len(pd2fctx, pEdeptnoitm, 4);
/*Set item Required property */
d2fitms_required(pd2fctx, pEdeptnoitm, TRUE);
/* Set Distance Between Records property */
d2fitms_dist_btwn_recs(pd2fctx, pEdeptnoitm, 0);
/* Set Database block(Database Item) property */
d2fitms_db_itm(pd2fctx, pEdeptnoitm, TRUE);
/* Set Query Allowed */
d2fitms_qry_allowed(pd2fctx, pEdeptnoitm, TRUE);
/* Set Query Length */
d2fitms_qry_len(pd2fctx, pEdeptnoitm, 4);
/* Set Update Allowed */
d2fitms_updt_allowed(pd2fctx, pEdeptnoitm, TRUE);
/* Set Item Displayed (Visible) */
d2fitms_visible(pd2fctx, pEdeptnoitm, TRUE);
/* Set Item Canvas property */
d2fitms_cnv_obj(pd2fctx, pEdeptnoitm, pd2fcnv);
/* Set Item X-position */
d2fitms_x_pos(pd2fctx, pEdeptnoitm, 493);
/* Set Item Y-position */
d2fitms_y_pos(pd2fctx, pEdeptnoitm, 50);
/* Set Item Width */
d2fitms_width(pd2fctx, pEdeptnoitm, 30);
/* Set Item Height */
d2fitms_height(pd2fctx, pEdeptnoitm, 17);
/* Set Item Bevel */
d2fitms_bevel(pd2fctx, pEdeptnoitm, D2FC_BEST_LOWERED);
/* Set item Hint */
d2fitms_hint(pd2fctx, PEdeptnoitm, (text*)"Enter value for :DEPTNO");
/*** Create DEPTNO item and item-related properties ***/
/* Create item */
d2fitmcr_Create(pd2fctx, pdeptblk, &pDdeptnoitm, (text*)"DEPTNO");
/* Set item type */
d2fitms_itm_type(pd2fctx, pDdeptnoitm, D2FC_ITTY_TI);
/* Set Enable property */
d2fitms_enabled(pd2fctx, pDdeptnoitm, TRUE);
/* Set item (keyboard) navigable property */
d2fitms_kbrd_navigable(pd2fctx, pDdeptnoitm, TRUE);
/* Set item Data Type property */
d2fitms_dat_typ(pd2fctx, pDdeptnoitm, D2FC_DATY_NUMBER);
/* Set item Max Length property */
d2fitms_max_len(pd2fctx, pDdeptnoitm, 4);
```

```
/*Set item Required property */
d2fitms_required(pd2fctx, pDdeptnoitm, TRUE);
/* Set Distance Between Records property */
d2fitms_dist_btwn_recs(pd2fctx, pDdeptnoitm, 0);
/* Set Database block(Database Item) property */
d2fitms_db_itm(pd2fctx, pDdeptnoitm, TRUE);
/* Set Query Allowed */
d2fitms_qry_allowed(pd2fctx, pDdeptnoitm, TRUE);
/* Set Query Length */
d2fitms_qry_len(pd2fctx, pDdeptnoitm, 4);
/* Set Update Allowed */
d2fitms_updt_allowed(pd2fctx, pDdeptnoitm, TRUE);
/* Set Item Displayed (Visible) */
d2fitms_visible(pd2fctx, pDdeptnoitm, TRUE);
/* Set Item Canvas property */
d2fitms_cnv_obj(pd2fctx, pDdeptnoitm, pd2fcnv);
/* Set Item X-position */
d2fitms_x_pos(pd2fctx, pDdeptnoitm, 32);
/* Set Item Y-position */
d2fitms_y_pos(pd2fctx, pDdeptnoitm, 151);
/* Set Item Width */
d2fitms_width(pd2fctx, pDdeptnoitm, 38);
/* Set Item Height */
d2fitms_height(pd2fctx, pDdeptnoitm, 17);
/* Set Item Bevel */
d2fitms_bevel(pd2fctx, pDdeptnoitm, D2FC_BEST_LOWERED);
/* Set item Hint */
d2fitms_hint(pd2fctx, PDdeptnoitm, (text*)"Enter value for :DEPTNO");
/*** Create DNAME item and item-related properties ***/
/* Create item */
d2fitmcr_Create(pd2fctx, pdeptblk, &pDdnameitm, (text*)"DNAME");
/* Set item type */
d2fitms_itm_type(pd2fctx, pDdnameitm, D2FC_ITTY_TI);
/* Set Enable property */
d2fitms_enabled(pd2fctx, pDdnameitm, TRUE);
/* Set item (keyboard) navigable property */
d2fitms_kbrd_navigable(pd2fctx, pDdnameitm, TRUE);
/* Set item Data Type property */
d2fitms_dat_typ(pd2fctx, pDdnameitm, D2FC_DATY_CHAR);
/* Set item Max Length property */
d2fitms_max_len(pd2fctx, pDdnameitm, 14);
/* Set Distance Between Records property */
d2fitms_dist_btwn_recs(pd2fctx, pDdnameitm, 0);
/* Set Database block(Database Item) property */
d2fitms_db_itm(pd2fctx, pDdnameitm, TRUE);
/* Set Query Allowed */
d2fitms_qry_allowed(pd2fctx, pDdnameitm, TRUE);
```

```
/* Set Query Length */
d2fitms_qry_len(pd2fctx, pDdnameitm, 14);
/* Set Update Allowed */
d2fitms_updt_allowed(pd2fctx, pDdnameitm, TRUE);
/* Set Item Displayed (Visible) */
d2fitms_visible(pd2fctx, pDdnameitm, TRUE);
/* Set Item Canvas property */
d2fitms_cnv_obj(pd2fctx, pDdnameitm, pd2fcnv);
/* Set Item X-position */
d2fitms_x_pos(pd2fctx, pDdnameitm, 70);
/* Set Item Y-position */
d2fitms_y_pos(pd2fctx, pDdnameitm, 151);
/* Set Item Width */
d2fitms_width(pd2fctx, pDdnameitm, 102);
/* Set Item Height */
d2fitms_height(pd2fctx, pDdnameitm, 17);
/* Set Item Bevel */
d2fitms_bevel(pd2fctx, pDdnameitm, D2FC_BEST_LOWERED);
/* Set item Hint */
d2fitms_hint(pd2fctx, PDdnameitm, (text*)"Enter value for :DNAME");
/*** Create LOC item and item-related properties ***/
/* Create item */
d2fitmcr_Create(pd2fctx, pdeptblk, &pDlocitm, (text*)"LOC");
/* Set item type */
d2fitms_itm_type(pd2fctx, pDlocitm, D2FC_ITTY_TI);
/* Set Enable property */
d2fitms_enabled(pd2fctx, pDlocitm, TRUE);
/* Set item (keyboard) navigable property */
d2fitms_kbrd_navigable(pd2fctx, pDlocitm, TRUE);
/* Set item Data Type property */
d2fitms_dat_typ(pd2fctx, pDlocitm, D2FC_DATY_CHAR);
/* Set item Max Length property */
d2fitms_max_len(pd2fctx, pDlocitm, 13);
/* Set Distance Between Records property */
d2fitms_dist_btwn_recs(pd2fctx, pDlocitm, 0);
/* Set Database block(Database Item) property */
d2fitms_db_itm(pd2fctx, pDlocitm, TRUE);
/* Set Query Allowed */
d2fitms_qry_allowed(pd2fctx, pDlocitm, TRUE);
/* Set Query Length */
d2fitms_qry_len(pd2fctx, pDlocitm, 13);
/* Set Update Allowed */
d2fitms_updt_allowed(pd2fctx, pDlocitm, TRUE);
/* Set Item Displayed (Visible) */
d2fitms_visible(pd2fctx, pDlocitm, TRUE);
/* Set Item Canvas property */
d2fitms_cnv_obj(pd2fctx, pDlocitm, pd2fcnv);
```

```
/* Set Item X-position */
d2fitms_x_pos(pd2fctx, pDlocitm, 173);
/* Set Item Y-position */
d2fitms_y_pos(pd2fctx, pDlocitm, 151);
/* Set Item Width */
d2fitms_width(pd2fctx, pDlocitm, 96);
/* Set Item Height */
d2fitms_height(pd2fctx, pDlocitm, 17);
/* Set Item Bevel */
d2fitms_bevel(pd2fctx, pDlocitm, D2FC_BEST_LOWERED);
/* Set item Hint */
d2fitms_hint(pd2fctx, PDlocitm, (text*)"Enter value for :LOC");
/*** Create Relations and relations-related properties ***/
/* Create Relation */
d2frelcr_Create(pd2fctx, (d2fob *)pdeptblk, &pd2frel, (text*)"DEPT_EMP");
/* Set Relation Detail block */
d2frels_detail_blk(pd2fctx, pd2frel, (text *)"EMP");
/* Set Master Deletes property */
d2frels_del_rec([pd2fctx, pd2frel, D2FC_DERE_NON_ISOLATED);
/* Set Deferred property */
d2frels_deferred(pd2ctx, pd2frel, FALSE);
/* Set Auto Query property */
d2frels_auto_qry(pd2ctx, pd2frel, FALSE);
/* Set Prevent Masterless property */
d2frels_prvnt_mstrless_ops(pd2ctx, pd2frel, FALSE);
/* Set Join Condition property */
d2frels_join_cond(pd2ctx, pd2frel, (text*)"DEPTNO");
/* Instantiate Relation: creates master-detail triggers */
d2frelup_Update(pd2fctx, pd2frel);
/* Save Form */
d2ffmdsv_Save(pd2fctx, pd2ffmd, (text*)0, FALSE, TRUE);
/* Compile Form */
d2ffmdcf_CompileFile(pd2fctx, pd2ffmd);
/* Destroy Context */
d2fctxde_Destroy(pd2fctx);
}
```

6.4 ODBC データソースに対して実行するアプリケーションの設計

企業内のデータは、複数の異種データソースに常駐することがよくあります。たとえば、データの一部が Oracle データベースに保存され、別の一部が Informix データベースに保存されている場合があります。単一でそのすべてのデータソースにアクセスできるアプリケーションの作成は、難しい作業になりかねません。

しかし、Forms Developer および Reports Developer のオープン・データソース・サポートを利用すれば、すべての ODBC 準拠データソースに対して透過的に実行される汎用アプリケーションを構築することができます。

この項では、オープン・データソース・サポートについて説明します。この項では次の項目について説明します。

- 6.4.1 項「Oracle Open Client Adapter(OCA)」
- 6.4.2 項「オープン・データソースのガイドライン」
- 6.4.3 項「ODBC データソースに対して実行するアプリケーションの設定」

6.4.1 Oracle Open Client Adapter(OCA)

ODBC データソースに接続する場合、Oracle Open Client Adapter (OCA) を使用します。OCA は ODBC レベル 2 に準拠したユーティリティで、Microsoft Windows 95、Windows NT 環境の Forms Developer および Reports Developer では、これを使用することにより ODBC ドライバを介して ODBC 準拠データソースにアクセスできます。

OCA は、Forms Developer および Reports Developer に含まれています。OCA をインストールするには、Oracle Installer を使用します。

6.4.1.1 OCA を利用する場合

アプリケーションで非 Oracle データソースにアクセスする必要がある場合は、必ず OCA を使ってください。Forms Developer および Reports Developer アプリケーションでは、すべての ODBC 準拠データソースに自動的にアクセスできます。ODBC データソースへの接続の詳細は、オンライン・ヘルプを参照してください。

6.4.1.2 OCA アーキテクチャ

Oracle Open Client Adapter の構成は次のとおりです。

コンポーネント	説明
Forms Developer または Reports Developer アプリケーション	処理を実行して ODBC ファンクションをコールし、SQL 文を送信して結果を取り出します。
Oracle Open Client Adapter	Oracle データベース・コールを ODBC コールに変換します。
ドライバ・マネージャ	アプリケーション用の ODBC ドライバをロードします。

コンポーネント	説明
ODBC ドライバ	ODBC ファンクション・コールを処理し、SQL 要求を指定のデータソースに送信して、結果をアプリケーションに戻します。
データソース	ユーザーがアクセスを望むデータ、それに対応付けられたオペレーティング・システム、DBMS および（もしあるならば）DBMS へのアクセスに使用されるネットワーク・プラットフォームからなります。

6.4.1.3 DBC 接続の確立

ODBC データソースに接続するには、接続ダイアログ・ボックスに次の接続文字列を入力します。

```
[user[/password]]@ODBC:datasource[:dbname]
```

たとえば、Sybase System 10 に接続する場合は、次のように入力します。

```
scott/tiger@ODBC:sybase_ds
```

6.4.1.4 ODBC ドライバ

ODBC データソースに接続する場合、ODBC ドライバを使用してデータソースと通信します。Forms Developer および Reports Developer には、サポートしている各データソースに対応した ODBC ドライバがすでにバンドルされています。これらのドライバは ODBC レベル 1 準拠ですが、パフォーマンスを向上させるためにレベル 2 の機能性を一部提供しています。

6.4.1.5 OPENDB.PLL

OPENDB.PLL は、OCA 付属の PL/SQL ファンクション・ライブラリです。アプリケーションで OPENDB.PLL を使用すると、次のことができます。

- 実行時にフォームおよびデータ・ブロック・プロパティを自動調整してデータソースを整えます。
- アプリケーションのメイン接続のほかに、その他のデータソースへの補助接続を開きます。
- 接続での任意の SQL 文およびストアド・プロシージャ・コールを実行します。
- 結果を Oracle 以外のストアド・プロシージャから取り出します。
- 任意の接続の DBMS および ODBC ドライバ名およびバージョンを取得します。

OPENDB.PLL の詳細は、ORACLE_ HOME¥TOOLS¥DOC20 ディレクトリの OCA_INFO.PDF を参照してください。

6.4.2 オープン・データソースのガイドライン

複数のデータソースを処理する場合は、次のガイドラインを考慮してください。

項目	推奨事項
アプリケーションを最適化して、複数のデータソースのフォームを実行します。	システムに固有の機能を利用するために特定のデータソースを指定するのでなければ、複数のデータソースに対して実行するためにアプリケーションを最適化する必要はありません。
ODBC データソースと併用する PL/SQL の作成	PL/SQL プログラム単位に埋め込まれている SQL 文は、接続対象データソースの Oracle SQL および SQL ダイアレクトの両方に適合する必要があります。Oracle と適合しない文の場合、PL/SQL コンパイル・エラーとなります。同様に、サポートされていない構文を使用する文の場合、実行時にエラーとなります。 SYSDATE ファンクションと USER ファンクションのみが、この制限の例外です。この 2 つは Oracle 固有のファンクションです。OCA によってこれらのファンクションが対応する ODBC ファンクションに変換されるため、これらのファンクションはすべてのデータソースに対応することができます。 データソース固有でも、Oracle 構文と競合する SQL 文を発行する場合は、EXEC_SQL パッケージを使用します。
複数のデータソースからの表の参照	多くのデータソースでは、データベース、所有者および表を指定すれば（たとえば、database.owner.tablename のように）、他のデータソースにある表にアクセスできます。 しかし、PL/SQL ではこのような 3 つの要素で表を指定する構文が認識されないため、クライアント側のプログラム単位またはトリガーはコンパイルされません。 この制限に対処するには、まず 3 部構成の表名を二重引用符で囲み、後に適切な OPENDB ファンクションをコールし、二重引用符を削除します。

項目	推奨事項
制限	■ 非 Oracle7 データソースを処理する場合は、アプリケーション・モジュール（フォーム、レポート、図表）をファイル・システムに保存する必要があります。非 Oracle7 データソースは、アプリケーション・モジュールを保存するためのリポジトリとして使用することはできません。 ■ 列名のトリガー情報へのアクセスは、オブジェクト・ナビゲータ（データベース・オブジェクト・ノード）からは行えません。 ■ ストアド・プロシージャ・テキストを表示できるのは、Oracle ALL_SOURCE 表をエミュレートするデータソースのみに限られます。データベース・ストアド・プロシージャ・テキストは編集できません。 ■ PL/SQL プログラム単位をクライアントから非 Oracle7 データソースにドラッグ・アンド・ドロップすることはできません。 ■ Forms Developer および Reports Developer では、マスター / ディテール・リレーションのデフォルト選択を行うために主キーおよび外部キーの制約情報を使用することはできません。これらのリレーションは、必要に応じて、直接指定する必要があります。 ■ オプティマイザ・ヒント（/*hint*/ の形式のコメント）は、OCA を使用して接続するデータソースでは無視されます。

トラブルシューティング	OCA 発行の SQL 文、および ODBC ドライバまたはデータベース生成のメッセージを表示するには、次のようにします。
	1. 次のエントリが、Windows NT および Windows 95 の場合はレジストリに、Windows3.1 の場合は `ORACLE.INI` に設定されていることを確認します。
	`UB=ORACLE_HOME¥OCA20`
	2. エラーが解決できない場合は、サポート担当に連絡してください。
	3. 次のエントリを、Windows NT および Windows 95 の場合は `SOFTWARE¥ORACLE` のレジストリに、Windows3.1 の場合は `ORACLE.INI` に追加します。
	`OCA_DEBUG_SQL=TRUE`
	`OCA_DEBUG_ERROR=TRUE`
	4. ODBC データソースに対してアプリケーションを実行して、SQL 文またはエラー・メッセージをデバッグ・ウィンドウに表示します。デバッグ・ウィンドウをクローズして処理を続行する場合は、「OK」をクリックしてください。

項目	推奨事項
デバッグ上のヒント	デバッグ情報を表示するには、OCA_DEBUG_SQL および OCA_DEBUG_ERROR 設定変数を TRUE に設定します。 これらの環境変数は、OCA データソースに対して Forms Developer または Reports Developer を使用する際の SQL エラーの特定作業に役立ちます。 OCA_DEBUG を TRUE に設定すると、ODBC ドライバに送られる SQL 文が送信前に表示されます。 OCA_DEBUG_ERROR を TRUE に設定すると、ODBC ドライバから戻されたエラーは、Forms Developer または Reports Developer に戻される前にダイアログに表示されます。

6.4.3 ODBC データソースに対して実行するアプリケーションの設定

ODBC 準拠データソースに対して実行するためにアプリケーションを設定するには、オンライン・ヘルプのトピック「非 Oracle データソースへのアクセス」を参照してください。

用語集

CGI ー コモン・ゲートウェイ・インタフェース（Common Gateway Interface）

Web サーバー上でアプリケーションを実行する業界標準の技術。Web サーバーから取り出した標準の HTML 文書が、静的（毎回まったく同じテキストが取り出される）であるのに対し、CGI は、Web サーバー上で稼働しているプログラムが、別のコンピュータと通信し、ユーザーの入力した情報に応じて " 動的 "HTML 文書を生成することが可能になる。

GUI ー グラフィカル・ユーザー・インタフェース（Graphical User Interface）

プログラムの入出力を示すために、言葉以外に絵を利用すること。GUI 付きのプログラムは、ウィンドウ・システム（X Window、Microsoft Windows、Apple Macintosh など）で稼動する。GUI プログラムでは、スクリーン上のウィンドウにアイコンやボタンなどが表示される。ユーザーは主にスクリーン上のポインタ（通常、マウスで制御）を動かして、GUI プログラムを制御する。

HTML ー ハイパーテキスト・マークアップ言語（HyperText Markup Language）

コンテンツおよびインターネットの WWW サーバー上にある他の文書への、ハイパーテキスト・リンクの指定に使用するタグ・ベースの ASCII 言語。Web ブラウザを使用するエンド・ユーザーは、HTML 文書を参照し、リンクをたどって他の文書を表示する。

HTTP ー ハイパーテキスト転送プロトコル（HyperText Transfer Protocol）

WWW ブラウザ・コンピュータとアクセスされている WWW サーバー間の WWW 通信量の転送に使用されるプロトコル。

IP（インターネット・プロトコル）アドレス（Internet Protocol Address）

それぞれ 3 桁までの数字から成る 4 つの部分で構成される番号で、インターネット上のコンピュータを一意に識別する。

JAR ー Java アーカイブ（Java ARchive）

多数のファイル（Java クラス・ファイル、イメージなど）を 1 つのファイルにまとめるために使用されるファイル。

Java

プラットフォームに依存しない“ アプレット ”の形で、インターネット用のプログラム作成をサポートするコンピュータ言語。

ORACLE_HOME

Oracle7 Server コード・ツリーのルートを示す環境変数。

PDF ― ポータブル・ドキュメント形式（Portable Document Format）

文書の作成に使用されたオリジナルのアプリケーション・ソフトウェア、ハードウェア、およびオペレーティング・システムに依存しない方法で、文書を表示するための（Adobe Acrobat 固有の）ファイル形式。PDF ファイルでは、デバイスと解像度に依存しない形式で、テキスト、図形、およびイメージの任意の組合せを含む文書を記述できる。

PL/SQL

Oracle 独自の SQL 言語の拡張版。アプリケーションの記述に適したものにするために、SQL にプロシージャ構成体とその他の構成体を追加する。

RDBMS ― リレーショナル・データベース管理システム（Relational Database Management System）

データ構造の定義、保存と検索操作、および整合性制約が可能なデータベース。このようなデータベースでは、データおよびそれらデータ間の関係は表にまとめられる。

TCP ― 転送制御プロトコル（Transmission Control Protocol）

クライアントと Web サーバー間で HTTP 要求を交換するための、基本通信プロトコル。

URL ― ユニフォーム・リソース・ロケータ（Uniform Resource Locator）

WWW サーバーとホーム・ページの指定に使用する“ アドレス ”。次はその例である。

```
http://www.acme.com/
```

これは、ホストのアドレスが `www.acme.com` であることを示している。

URL は、ほとんどの場合ファイル名（その URL までの長いパス名が含まれる場合もあり、PC-DOS ファイル名の場合は、通常、`.HTML` または `.HTM` の拡張子が付いている）。

Web カートリッジ（Web cartridge）

WRB を介して Web サーバー上で実行されるプログラム。

Web サーバー（Web server）

Web サイトで稼動しているサーバー・プロセス（HTTP デーモン）のことで、リモートの Web ブラウザからの HTTP 要求に応じて、Web ページを送り出す。

Web ブラウザ（Web browser）

エンド・ユーザーが、（Web サーバーによりサービスを提供している）コンピュータに保存されている HTML 文書およびプログラムの読込みに利用するプログラム。

WRB ー Oracle Web Request Broker

Web 用アプリケーションの開発および利用のための、強力な分散ランタイム環境を提供する。WRB ランタイム・プラットフォームにより、アプリケーション開発者は Web サーバーに依存せず、多数の Web サーバーを処理するアプリケーションを作成できる。

WWW ー World Wide Web

インターネット上のサーバーのネットワーク。各サーバーには 1 つ以上のホーム・ページがあり、情報や、そのサーバー上と（通常は）他のサーバー上にある他の文書へのハイパーテキスト・リンクを提供する。

アクション（action)

Project Builder では、Project Builder またはユーザーによって指定され、任意の形式のファイルに適用されるコマンド文字列のこと。アクションは、1 つのファイル形式に限定されない。たとえば、アクションの " COMPILE（コンパイル）" をフォームと C ソース・ファイルの両方に定義した場合、メニュー項目から「すべてをビルド」を選択すれば、適切なツールを使用して、すべての FMB および .C ファイルがコンパイルされる。「事前定義済みのアクション」、「ユーザー定義アクション」も参照。

アプレット（applet）

Web ページやアプリケーションに、必要に応じて動的にインポートできる小さなプログラムを表す Java 用語。

暗号化（encryption）

目的の受信者のみがデータをスクランブル解除（解読）して読めるように、データにスクランブルをかける（暗号化する）こと。

暗黙的項目（implied item）

Project Builder では、通常は自動生成の結果であるプロジェクト項目のことで、Project Builder によって認識され、プロジェクト・ナビゲータに自動的にエントリが作成される。たとえば、Project Builder では、C ソース・ファイルのコンパイルで即時ステップとして生成された `.OBJ` ファイルを認識し、プロジェクト・ナビゲータにそれらのエントリを作成する。暗黙的項目のプロパティはリセットできないという制限があるが（次のコンパイルで変更が破壊される）、暗黙的項目は「編集」、「表示」、「印刷」といったアクションによって調べられるので、役に立つ場合がある。「アクション」、「項目」も参照。

依存性表示（dependency view）

Project Builder では、プロジェクト・ナビゲータ内で、ファイルを相互に依存する順序で表示する表示形式のこと。階層の最上位はプロジェクト・ノードで、その下にターゲット・ノード、さらにその下にそれらのターゲットの構築可能なコンポーネントが続く。たとえ

ば、実行可能フォームの下には、フォームが依存する .fmb ファイルが続き、.fmb ファイルの下にはファイルが依存するユーザー・イグジットに使用されるライブラリが続く、というようになる。「プロジェクト表示」、「ターゲット」も参照。

インターネット（Internet）

世界規模の TCP/IP ベースのコンピュータ・ネットワーク。

イントラネット（Intranet）

企業または組織内の個人に（ファイアウォールを介して）アクセスを限定した内部 TCP/IP ネットワーク。イントラネットではインターネット類似のサービスが組織内で提供されるが、インターネットと接続されているとは限らない。イントラネットの一般的な例は、企業が社内で情報あるいはアプリケーションを配布するために、内部ネットワーク上に 1 つ以上の Web サーバーを設置するというもの。

インポート（import）

Project Builder では、プロジェクト情報を含むファイルを読み込むこと。これはプロジェクト共有のために推奨されている方法。「エクスポート」、「エクスポート・ファイル」も参照。

ウィンドウ（window）

グラフィカル・ユーザー・インタフェース（GUI）環境におけるスクリーン。ウィンドウは、構成要素が描かれる面を囲むフレーム。

エクスポート（export）

Project Builder では、プロジェクト、タイプ、アクション、マクロの定義を個別に、あるいはすべて含むファイルを、移植可能な形式に書き、異種プラットフォーム上で作業する他のユーザーに配布するプロセスのこと。「エクスポート・ファイル」、「インポート」も参照。

エクスポート・ファイル（export file）

Project Builder では、プロジェクトのエクスポートにより作成された、共有可能で、移植可能なファイルのこと。エクスポート・ファイルのデフォルトの拡張子は .UPX。「エクスポート」、「インポート」も参照。

エンティティ（entity）

ユーザーにとって重要なもの。エンティティの例としては、‘ 割当て ’ や ‘ 売上発注書 ’ がある。1 つのエンティティは、‘ 売上レポート ’、‘ 割当て制限 ’ および ‘ 地域 ’ といった複数のブロックから構成されている場合がある。

仮想ディレクトリ（virtual directory）

仮想ファイル・システムを、ホスト・マシンのオペレーティング・システムによって維持しているファイル・システムに保存されているファイルにマップすること。

仮想ファイル・システム（virtual file system）

URL で使用されているパス名を、ホスト・マシンのオペレーティング・システムによって維持しているファイル・システムに関連付けるマッピング。

キャラクタ・セット (character set)

各文字を異なるバイナリ値によって表すエンコード法。たとえば、ISO8859-1 は、40 以上の西欧言語をサポートする拡張ラテン・キャラクタ・セット。

キャンバス（canvas）

インタフェースの項目やプロンプトが表示される面。キャンバスはウィンドウに表示される。

グループ（group)

Project Builder では、ランチャから離れたサブメニューを介して使用できる関連項目の集合のこと。グループにより、ユーザーはランチャを Windows95 の「スタート」メニューのように設定し、任意の " グループ " がポップアップして、他の項目やグループを表示できる。

グローバル・レジストリ（Global Registry）

インストールされた Forms Developer または Reports Developer 全体に共通の情報を保存する Project Builder レジストリ。この情報は、タイプの定義とそれらに関連付けられたアクション、事前定義済みまたはユーザー定義のプロパティに限られる。グローバル・レジストリの使用はオプションで、そのファンクションは、個々のユーザーのレジストリで実行できる。「レジストリ」、「ユーザー・レジストリ」も参照。

継承（inherit）

Project Builder では、依存性ツリーの上位ノードからアクション、タイプ、マクロ、またはプロパティの定義の情報を取得すること。上位ノードに関連の属性があれば、それらは継承される。このため、フォームや文書のようなファイルシステム項目には、サブプロジェクト、プロジェクト、ユーザー・レジストリ、またはグローバル・レジストリからアクションの定義が継承され、プロジェクトには、ユーザー・レジストリまたはグローバル・レジストリからタイプの定義が継承される、というようになる。

言語環境変数（language environment variable）

ユーザーの環境用に設定された言語、領域およびキャラクタ・セットを指定する環境変数。言語環境変数は、次のうちのいずれかになる。NLS_LANG、DEVELOPER_NLS_LANG、または USER_NLS_LANG。

項目（item）

Project Builder では、フォームまたはレポートといった、プロジェクトに関連付けられたファイル・システム内のオブジェクトのことで、プロジェクト・ナビゲータ内のノードによって指定または表される。

事前定義済みのアクション（pre-defined action）

Project Builder に付属しているアクションで、メニュー項目またはツールバー・ボタン、あるいはその両方によって自動的にユーザーが使用できるようになる。事前定義済みのアクションには、「ビルド」および「配布」の他、いくつかのソース制御オプションがある。ある事前定義済みのアクションが、サポートされているファイル形式に定義されている場合、ユーザーが Project Builder からコールすると、そのファイル形式の任意の選択項目によって、そのアクションが起動される。「アクション」、「ユーザー定義アクション」も参照。

書式マスク（format mask)

フィールドの値の表示を定義する設定。たとえば、書式マスクは、通貨の金額や日付の表示方法の指定に使用される。

スナップ・ポイント（snap point）

その位置を示す（X、Y）位置に対応したウィジェットのポイント。

ソケット（socket）

1 つの IP アドレスと 1 つのポート番号の組合せ。

ターゲット（target）

Project Builder では、依存性ツリーの中にある任意の項目のこと。たとえば、実行ファイルはライブラリの（コンパイル）ターゲット、ライブラリはオブジェクト・グループのターゲット、オブジェクトはソース・ファイルのターゲットである。「入力項目」も参照。

ダイアログ・ボックス（dialog box）

特定のアクションを完了するために必要な情報入力に使用されるウィンドウ。ユーザーは、続行する前にこのウィンドウで対話する必要がある。

タイプ（type）

Project Builder では、フォームや文書などのファイル形式の説明のことで、タイプの名前や説明といった情報が含まれる。タイプはアクションやマクロを定義する基礎となる。

多言語アプリケーション（multilingual application）

複数の言語で実行でき、ローカル規則に従ってデータを表示するアプリケーション。

ツールバー（toolbar）

「リスト」や「保存」といった一般的なアクションを実行する一連のアイコン・ボタン。

入力項目（input item）

Project Builder では、ターゲットの作成に使用されるファイルのこと。たとえば、.FMB は .FMX の入力項目。“ ソース ” とも呼ばれる。

ハイパーテキスト（hypertext）

Web ブラウザによって、読者が 1 つの文書から別の文書へ簡単に移動できるようにするクロス・リファレンスを含む文書の集合。

ハイパーリンク（hyperlink）

1 つのハイパーテキスト文書のある点から、別の文書（のある点）あるいは同じ文書内の別の場所への参照（リンク）。Web ブラウザでは、通常、ハイパーリンクをなんらかの目立つ方法（別の色、フォントまたはスタイル）で表示する。ユーザーがハイパーリンクをアクティブにする（マウスでクリックする）と、ブラウザにそのリンクのターゲットが表示される。

配布（deliver）

Project Builder では、分配し、利用するために、完成したアプリケーションを準備し、提供すること。

ビルトイン・マクロ（built-in macro）

Project Builder では、Project Builder に付属しているマクロのこと。「マクロ」も参照。

ファイアウォール（firewall)

ローカル・エリア・ネットワーク（LAN）で外部からのコンピュータへのアクセスや、LAN 内からの外部コンピュータへのアクセスを制限するコンピュータ。

フィールド（field）

ユーザーに対して情報を表示する、またはユーザーからの入力を受け入れる（あるいはその両方を行う）インタフェースの構成要素。フィールドの例としては、テキスト項目、チェック・ボックス、ポップリストがある。‘ ウィジェット（widget）’ または ‘ 項目（item）’ としても知られている。

プロジェクト（project）

Project Builder によって作成される基本データ構造体。プロジェクトは、ユーザーのファイル・システムにあるファイルの場所を示すポインタの集合。プロジェクトには、ユーザーが指定したプロジェクトに適用すると考える動作についての情報も含まれる。例えば次のどの形式のファイルを編集するにも特定のエディタを起動するといったもの。.CPP、.H および .TXT。プロジェクト・ファイルは、プラットフォーム全体でエクスポートも共有も可能。「エクスポート」、「プロジェクト定義ファイル」、「プロジェクト項目」も参照。

プロジェクト・ウィザード（Project Wizard）

Project Builder では、ユーザーが新規のプロジェクトまたはサブプロジェクトを作成する際、必要なステップを完了できるように案内するダイアログのこと。

プロジェクト項目（project item）

Project Builder では、接続文字列や、含まれている項目の暗黙リストなど、プロジェクト固有の情報が保存される項目のこと。ここではタイプの定義は行われないが、プロジェクトで参照できるすべてのタイプに対するアクションとユーザー定義マクロの定義または上書き、あるいはその両方ができる。「項目」、「プロジェクト」、「プロジェクト定義ファイル」も参照。

プロジェクト定義ファイル（project definition file）

Project Builder では、プロジェクト項目とそれらのプロパティから成るプロジェクト・データを保存するファイルのこと。デフォルトでは、ファイルごとに 1 つのプロジェクト項目が含まれ、このプロジェクトをそのファイルの “ ルート ” またはマスター・プロジェクトとみなすことができる。ユーザーは、このファイル内で必要なだけサブプロジェクトを作成できる。サブプロジェクトは、マスター・プロジェクトの下にある項目で、これによりユーザーは項目のサブグループを収集して、親（サブプロジェクト）レベルで、それらのプロパティを変更できる。プロジェクト・ファイルのデフォルトの拡張子は .UPD。「プロジェクト」、「プロジェクト項目」も参照。

プロジェクト・ナビゲータ（Project Navigator）

Project Builder では現行セッションのプロジェクト項目の階層型リストが含まれるウィンドウのこと。リストは略図の形で表示され、ユーザーは、オブジェクトの作成、編集、改名および削除といった、いくつかの作業を完了できる。いつでも参照できるスキーマは 1 つのみだが、ユーザーは 2 つの異なるスキーマから、オブジェクトの編成に使用するスキーマを選択できる。「依存性表示」、「プロジェクト表示」も参照。

プロジェクト表示（project view）

Project Builder のプロジェクト表示では、プロジェクト・ナビゲータ内のオブジェクトがタイプやプロジェクト / サブプロジェクトのリレーションごとに編成されて、表示される。プロジェクトは、まずプロジェクト・ファイルごとに、次にカテゴリごとにアルファベット順に編成されている。「依存性表示」、「プロジェクト・ナビゲータ」も参照。

ブロック（block）

フォーム上のエンティティを表したもの。

プロンプト（prompt）

項目を一意に識別するラベル。‘ 販売員 ’ や ‘ 項目説明 ’ がプロンプトの例。

ポート（port）

TCP で特定のプログラムとの送信データのやりとりに使用される番号。

マクロ（macro）

Project Builder では、アクションのカスタマイズおよび拡張に使用できるタイプ固有の変数のこと。マクロは定数でも簡単な式（これには、さらに定数または式、あるいはその両方が含まれる場合がある）でもよい。マクロの使用により、コマンド・オプションの発行や、1

つのプロパティ定義の変更によって一連のコマンドをユーザーが変更できるようにするといった面で自由度が高まる。

マスター・ディテール（master-detail）

情報の階層を示す 2 つのエンティティ間のリレーション。たとえば、「受注」は「（受注）データ」エンティティと「（受注）明細」エンティティから成り、「（受注）データ」が「（受注）明細」のマスターで、「（受注）明細」が「（受注）データ」のディテールとなる。

モーダル（modal）

アプリケーションの操作を継続する前に、ユーザーが特定の情報を提供しなければならない状態。

ユーザー定義アクション（user-defined action）

Project Builder では、Project Builder ユーザーによって定義されたカスタム・アクションのこと。このようなアクションは、1 つのファイル形式に適用しても、すべてのファイル形式に適用してもよい。「アクション」、「事前定義済みのアクション」も参照。

ユーザー定義マクロ（user-defined macro）

Project Builder では、Project Builder のユーザーによって定義されたカスタム・マクロのこと。このようなマクロは、事前定義済みおよびユーザー定義のアクションの変更に使用できる。「アクション」、「ビルトイン・マクロ」、「マクロ」、「事前定義済みのアクション」も参照。

ユーザー・レジストリ（user registry）

Project Builder では、ユーザーごとに構成情報を保存するプロジェクト・ファイルのこと。これにより、ユーザーは各自の開発環境をカスタマイズできる。ユーザー・レジストリには、グローバル・レジストリからタイプ情報が継承され、グローバル・レジストリで定義されたタイプの一部を変更するのみでなく、新規のタイプを定義することもできる。ユーザー・レジストリには、ユーザーの優先接続文字列、UI 設定、およびプロジェクトの一般情報といった、環境や作業環境情報も保存される。「グローバル・レジストリ」、「レジストリ」も参照。

ランチャ（Launcher）

Project Builder では、プロジェクト・ナビゲータの左に（デフォルトで）合体される二次的なツールバーのこと。簡単な編成機能およびアプリケーション起動機能を提供する。

領域（region）

1 つのエンティティ内の一連の関連項目。たとえば、Purchase Order Header エンティティには、「レート」、「種類」および「日付」フィールドから構成される「通貨情報」の領域が含まれる。

両方向サポート（bidirectional support)

中東や北アフリカの言語のように、通常右から左に書く言語のサポート。

レジストリ（registry）

Project Builder では、プロジェクトの定義や環境情報の保存に使用される、グローバルまたはユーザー固有、あるいはその両方の構成ファイルのこと。「グローバル・レジストリ」、「ユーザー・レジストリ」も参照。

索引

M

N

O

P

R

S

U

V

W

あ

い

う

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

は

ひ

ふ

へ

ほ

ま

み

め

も

ゆ

ら

り

れ

ろ

わ