製品版、評価版付属のサンプルプログラムデータを使用した

LLL/.net 開発環境評価手引き

株式会社アセンディア

パーシモン事業部

■■■---------- 目次 ----------■■■

１．はじめに ---------- 3
２．評価をはじめる前の準備作業 ---------- 5
３．LLL/.netの起動とサンプルシステムの動作確認 ---------- 6
４．簡単な単票形式データ入力プログラム ---------- 13
（テストプログラムの作成　その１）
５．複雑な単票形式データ入力プログラム---------- 18
（テストプログラムの作成　その２）
６．明細伝票形式データ入力プログラム---------- 29
（テストプログラムの作成　その３）
７．LLL/.netアプリケーションの動作構造 ---------- 43
８．フォームテンプレートと生成コード ---------- 45
〈フリーフォーマット（PAA100）テンプレート〉 45
（テストプログラムの作成　その４）
９．複雑な検索用ウインドウの作成 ---------- 72
〈多段階コード検索〉 72
〈複数フォーム使用のプログラム〉 74
（テストプログラムの作成　その５）
１０．スマートクライアント対応機能 ---------- 85
＜LLL/.net が実現するスマートクライアントとは＞ 85
＜Web 経由のＤＢアクセスを実現する XML Web サービス＞ 87
＜スマートクライアント型アプリケーションの作成＞ 88
＜デプロイメント（Web 経由のプログラム配信と自動アップデート）＞ 90
１１．サーバーコンポーネント ---------- 93
＜サンプル PG の説明とデータセットの扱い＞ 93
＜コンポーネントテンプレート）＞ 103
１２.帳票設計オプション---------- 107
＜LLL/.net 帳票設計オプションの概要＞ 107
＜簡単な帳票プログラムの作成＞ 112
（テストプログラムの作成　その６）
＜合計明細伝票印刷プログラムの作成＞ 121
（テストプログラムの作成　その７）
１３．おわりに ---------- 143

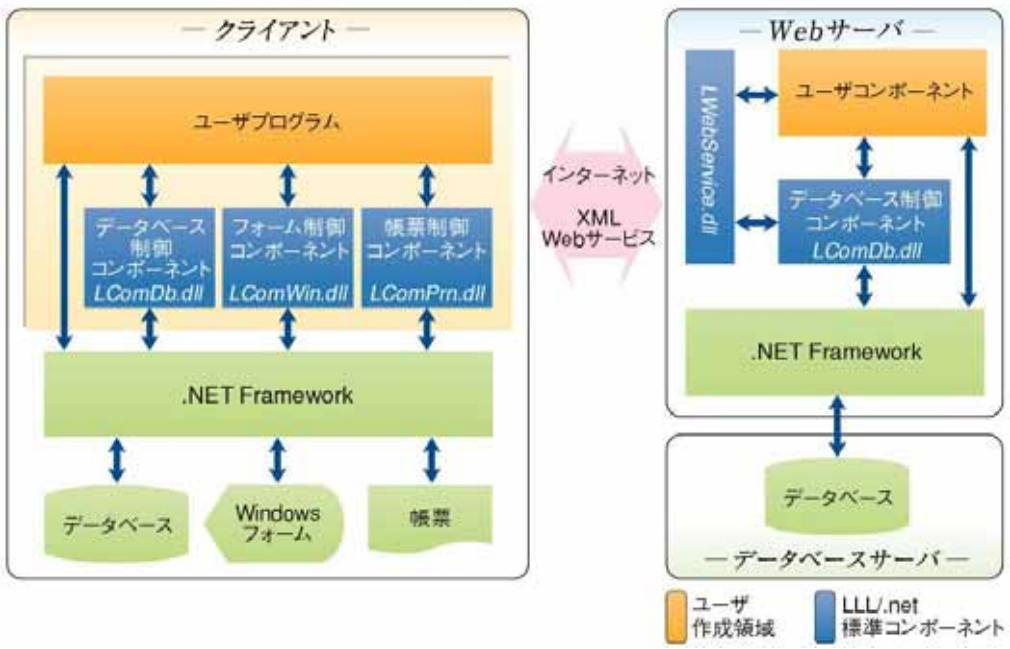
１．はじめに

本書では、LLL/.net の製品版、または評価版に付属するサンプルプログラム、およびデータベースを使用して、実際にプログラムを作成しながら LLL/.net を解説します。
したがって、極力LLL/.netがインストール済みで使用可能な環境をご用意ください。製品版をお持ちでない方は弊社まで評価版をご請求ください。

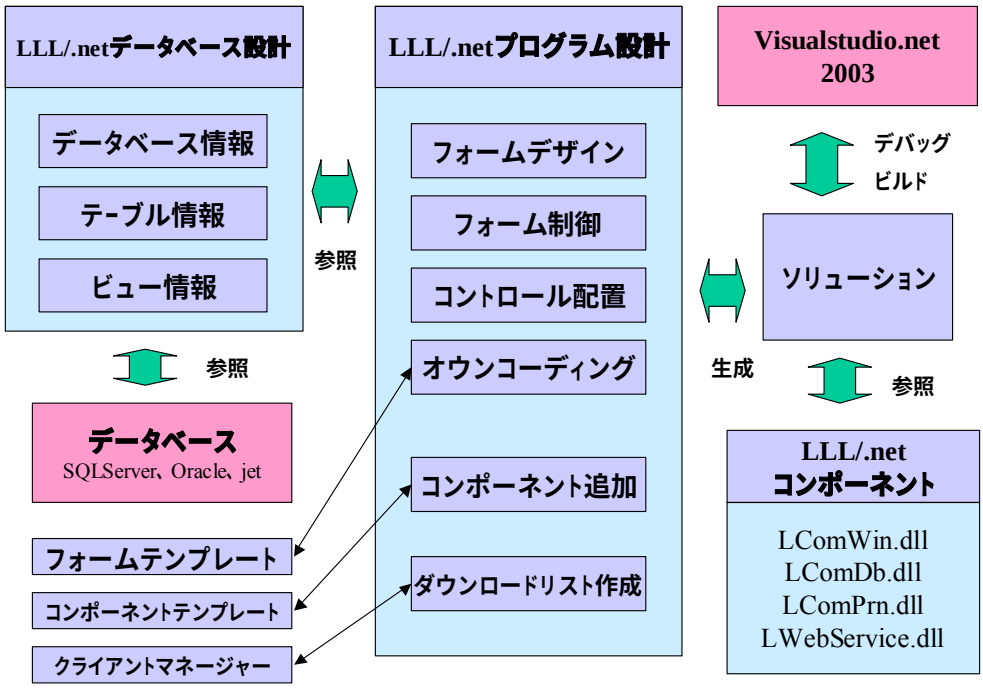
本書をお読みいただくにあたって、何らかの手段によるDBアクセスを行うWindowsアプリケーションの開発経験程度の知識が必要ですが、.NetFrameworkやOOP等の経験は問いません。そのため、経験の深い方にとては説明が冗長に感じられる点が多々あります。そう感じられるところは、読み飛ばしてお進みください。
前半の６章までは、開発環境の操作説明とコーディング例が主で、まずLLL/.netの操作に慣れていただき、とりあえず理屈抜きでプログラムを作ってみられることが目的です。構造や仕組みを理解し、応用力を高めるための情報は７章以降を中心にご覧ください。

< LLL/.net （トリプルエル ドットネット） >

**LLL/.netは、.NET Framework 対応のアプリケーション開発支援ツールです。
VisualStudio.net と組み合わせて使用し VB.NET と C# をサポートします。
WindowsフォームによるＣ／Ｓ型と、Windowsフォーム +XML/WEB サービスによるスマートクライアント型のアプリケーションを簡単に開発でき、独自実装したデプロイメントツールによりプログラムやファイルをWEB経由で自動配布、自動更新できるなどの機能提供します。
ドラッグ＆ドロップ操作で、テーブル項目とリンクしたフィールドを持つフォームを簡単に作成できます。そのフォームを制御し使い勝手の良い操作性を提供したり、データベースアクセスは、LLL/.net標準提供のコンポーネントが担います。
コンポーネントを使用する処理の流れは、改変や自作も自由に行え汎用性の高いテンプレートが生成するプログラムが制御します。
そしてLLL/.netのプログラム設計機能が、これらのリソースを統合的に扱い
VisualStudio.netでデバッグや修正が行えるソリューションを自動生成します。
その修正情報を反映させてLLL/.net に戻り、設定変更を加えて再生成するなど
LLL/.netとVS.netを何度でも行き来できるスパイラルな開発環境を実現しています。
操作性に優れたシステム構築の工数削減と標準化に威力を発揮するツールです。**



LLL/.netアプリケーションの動作構造図



LLL/.net開発環境の全体構成図

２．評価をはじめる前の準備作業

LLL/.netまたは評価版をインストールしていただいたルートディレクトリに **Manual.chm** というファイルがありますが、これがLLL/.netのマニュアルです。
これのショートカットをデスクトップに作成し、そこから起動しておいてください。
このマニュアルは、LLL/.netの開発環境メインメイニューの**「ヘルプ」**や、ログイン画面の**「ヘルプ」**ボタンを押すことで表示されるものと同じですが、これらから呼び出した場合、同期起動されます。このようにショートカットを作っておいていただくと開発環境とは非同期でいつでも見られて便利だと思います。
ショートカットを作りたくない場合は、スタートメニューから**「すべてのプログラム」->「LLL/.net開発キット」->「ドキュメント」**を選択しても非同期起動できます。
マニュアルと併行してサンプルの動作を確認していただくことで御理解が容易になります。

マニュアルの**「システム概要」->「セットアップ手順」->「セットアップ」**をご覧ください。そこに**「インストール先のディレクトリ構造」**が出ていますので、インストールしていただいたディレクトリがそこに示す構造になっていることご御確認ください。

ルートディレクトリの下にいくつかのサブディレクトリがあり、その中に **Sample**というものがあります。
Exploreで、そのディレクトリを開くと、**SampleCS.exe** と **SampleVB.exe** という２つの自己解凍形式の圧縮ファイルが入っています。ファイル名の後ろがVBとなっているのがVisualBasic.net、CSとなっているのがC# 用のそれぞれサンプルです。
見たい方のexeを実行して解凍してください。ただし紙面の都合上本書ではこれ以降のコーディングの殆どをVisualBasic.netを例として説明しています。

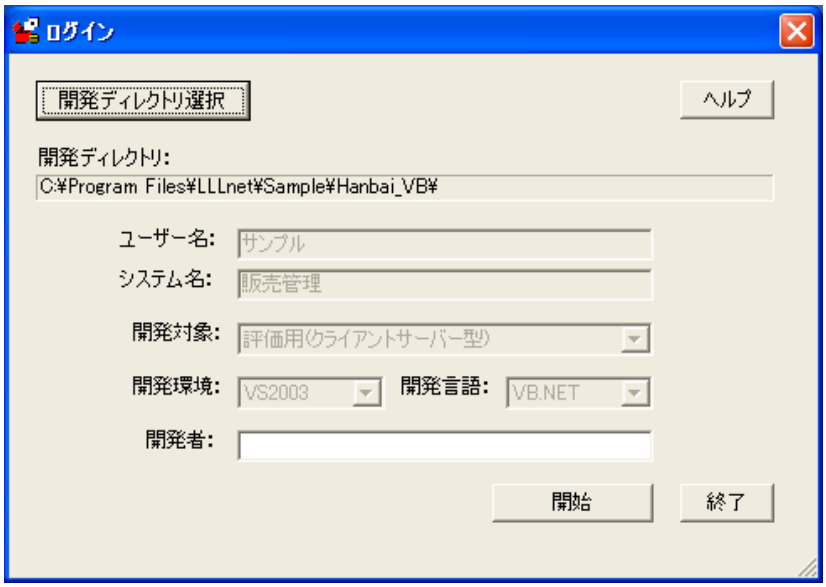
もちろん.NetFrameworkは言語依存性が低いのが特徴です。LLL/.netにおいても提供するプログラム設計機能やコンポーネントなどのツールや、できあがったプログラムの動作構造などは、VisualBasic.netとC#で共通ですので、すこし読み替えていただくことで本書はどちらの言語をご利用の方にもお使いいただけます。

例えば**SampleVB.exe** の方を解凍すると、デフォルト状態では **Sample** ディレクトリの下に **Hanbai_VB** といサブディレクトリを作成し、その下にサンプルに必要なフォルダーとファイル一式を展開します。この **Hanbai_VB** がLLL/.netの**「開発ディレクトリ」**になっています。**SampleCS.exe**の方では、**Hanbai_VC** という名前になります。

これでサンプルの準備ができましたので、LLL/.net開発環境を立ち上げましょう。

３．LLL/.netの起動とサンプルシステムの動作確認

インストール時にデスクトップ上に作成されるLLL/.netのショートカットアイコンをダブルクリックするとLLL/.netが起動します。最初に出てくるのはログイン画面です。

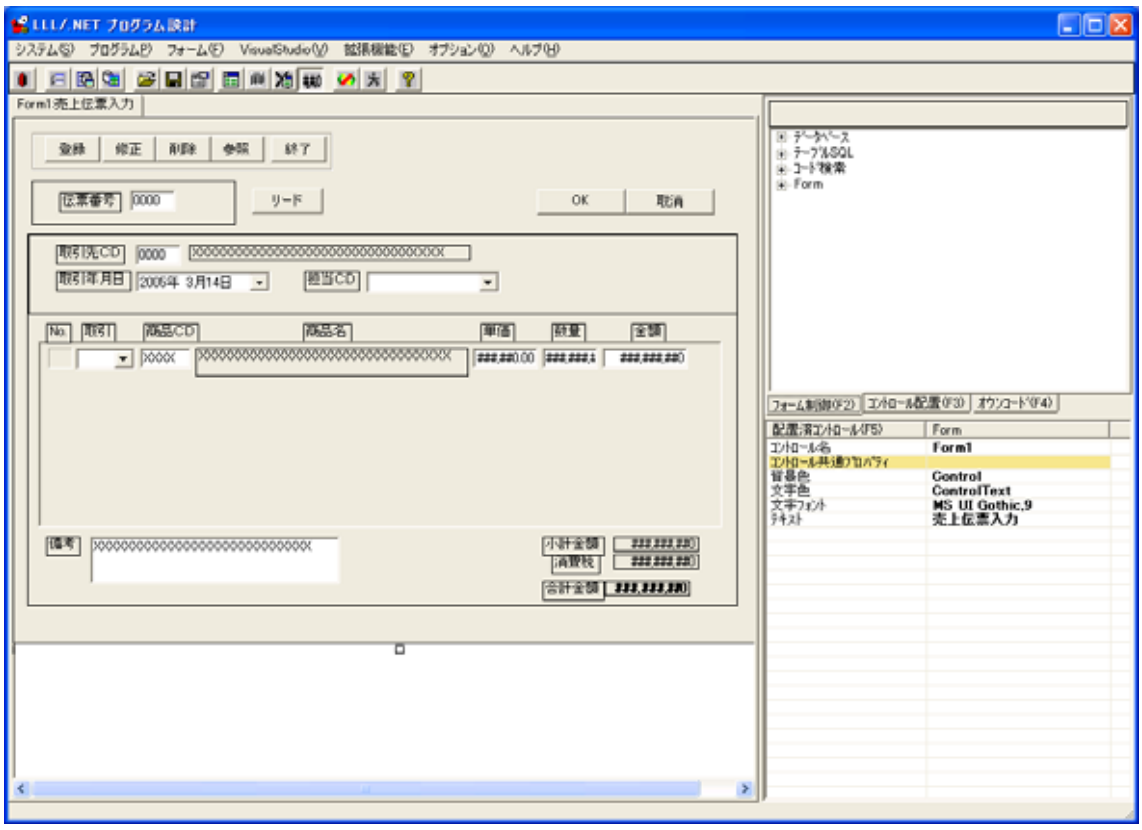


ここでどの開発ディレクトリにログインするかを入力しますが、最初は今解凍したサンプルの開発ディレクトリしかありませんので、**「開発ディレクトリ選択」**ボタンを押して、そのディレクトリ（**Hanbai_VB**）を選択してログインします。

「開発ディレクトリ」は、開発する複数のプログラム（ソリューション）をグループ化してまとめて保管できる場所で、LLL/.net開発環境の管理対象単位です。
開発環境へのログインの単位であり、開発対象がクライアントサーバー型かスマートクライアント型かも、この「開発ディレクトリ」毎に指定することになります。

次に**「プログラム選択」**という画面が表示されサンプルで提供されているプログラムの一覧が出てきます。
まずはそこでLLL/.netが提供する多くの機能を使用するという意味で、最も代表的なサンプルプログラムである　**「HAD010　売上伝票入力」**を見てみましょう。
これを選択して**「オープン」**ボタンを押してください。

すると**「LLL/.netプログラム設計」**というタイトルの付いた画面が出てきます。
これがLLL/.net開発環境のメイン画面です。今回は**「HAD010 売上伝票入力」**プログラムが既に設計済みの情報として表示されています。

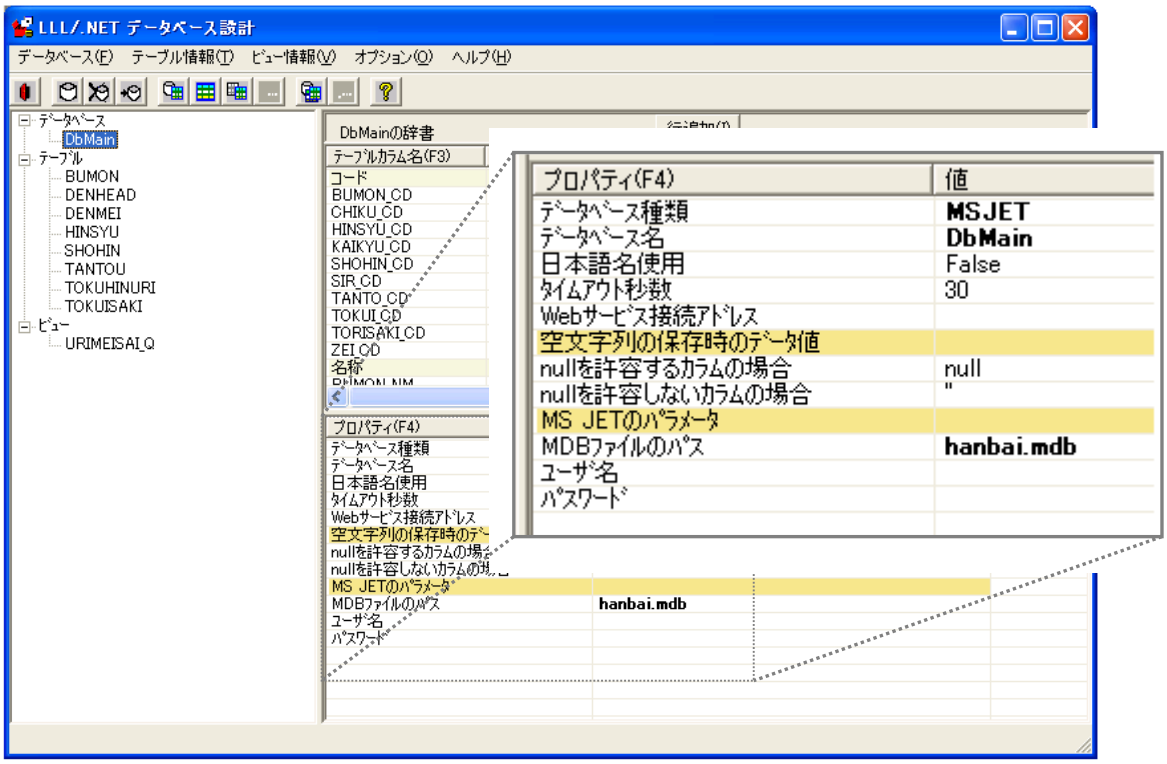


ではマニュアルを参照しながらそこに書かれた事と、実際に設計されたサンプルの情報を照らし合わせながら、LLL/.net開発環境の機能の確認をしていきましょう。

マニュアル->**「システムの概要」->「プログラム開発手順の概要」->「(1)開発システムのディレクトリ作成」**を見てください。開発ディレクトリを新規作成する時の手順と出来上がったディレクトリ構造の解説が出ています。サンプルの開発ディレクトリは既にできていますので、実物を確認しましょう。エクスプローラーで **Hanbai_VB** ディレクトリを開き、マニュアルどおりにディレクトリ構造が展開されていることを確認してください。

「システムの概要」->「プログラム開発手順の概要」->「(2)データベース設計」にデータベースを登録する手順があります。これもサンプルの実物を確認しましょう。
「LLL/.netプログラム設計」のメニューバーから**「システム」->「データベース設計」**を選択すると**「LLL/.netデータベース設計」**という画面が出てきます。

画面左のツリービューにサンプルが使用するデータベース情報として、**データベース**、**テーブル**、**ビュー**の各一覧が表示されます。**「データベース」**の中に、**DbMain**というアイテム名が表示されています。これがLLL/.netプログラムが認識するデータベースの名前です。マウスでこれを選択すると、画面右側に**DbMain**のプロパティが表示されます。



上段は、**DbMain**用に登録されている項目辞書の一覧、下段にはそれ以外のプロパティとしてデータベースの種類や接続情報が出ています。
サンプルではAccessのデータベースである**Jetエンジン**を使用していますので、その情報が表示されています。このデータベースの実体は **hanbai.mdb** というファイルですのでその名前がMDBパスに入っています。
これがORACLEや、SQL-SERVERになると接続するための情報の種類が変わります。
下段プロパティの最上行**「データベース種類」**が現在は**MSJET**になっていますが、試しに**oracle**や、**sql-server**に変更してみてください。その下に出てくる項目がデータベースにあわせて変化することを確認していただけます。
ただしこの操作を行うと、既にサンプル用として設定されていたMSJET用の情報が一部消えてしまいますので更新保存しないでください。もっともしまちがって更新してしまっても、サンプルの自己解凍ファイルを再解凍していただければ元に戻せます。
その点だけ注意していただき、その他の各項目もチェックしてみてください。

LLL/.netは下流CASEツールであり、データベースの設計支援などの上流CASEに分類されるような機能はありません。
データベース側で予めデータベースやテーブルを作成しておいていただき、LLL/.netではそれに接続してその情報取り込んで使用するのが、一般的な使い方になります。
既存のデータベースからの情報の取り込みも、この**「LLL/.netデータベース設計」**から行っていただきますが、ここでは説明を省略します。
データベースの設計の詳細については、マニュアルの**「システムの操作」->「データベース設計」**をご参照ください。

先ほど選択した売上伝票入力プログラムでは、hanbai.mdb の中の **DENHEAD（伝票デッダ）**、**DENMEI（伝票明細）**、**TOKUISAKI（得意先マスタ）**、**SHOHIN（商品マスタ）**、**TANTOU（担当者マスタ）** の各テーブルを使用します。
DENHEAD は、伝票ヘッダーテーブルで、このプログラムが使用するフォームテンプレート（画面デザインと処理の流れの雛形）のメインテーブル（ **main** ）です。DENMEI は伝票明細テーブルで、このプログラムにおけるフォームテンプレートの明細テーブル（ **subs** ）です。1件のメインテーブルレコードに複数件のDENMEIレコードが関連付けられて動作します。この２つのテーブルデータがこのフォームテンプレートにおけるデフォルトでの更新対象であり、自動的にひとつのトランザクションとして処理されます。
他のテーブルは、それぞれ得意先、商品、担当者のマスターテーブルで、このサンプルでは更新は行わず、参照にのみ使用しています。

では、これらが**「LLL/.netプログラム設計」**においてどのように定義され、設計されてい

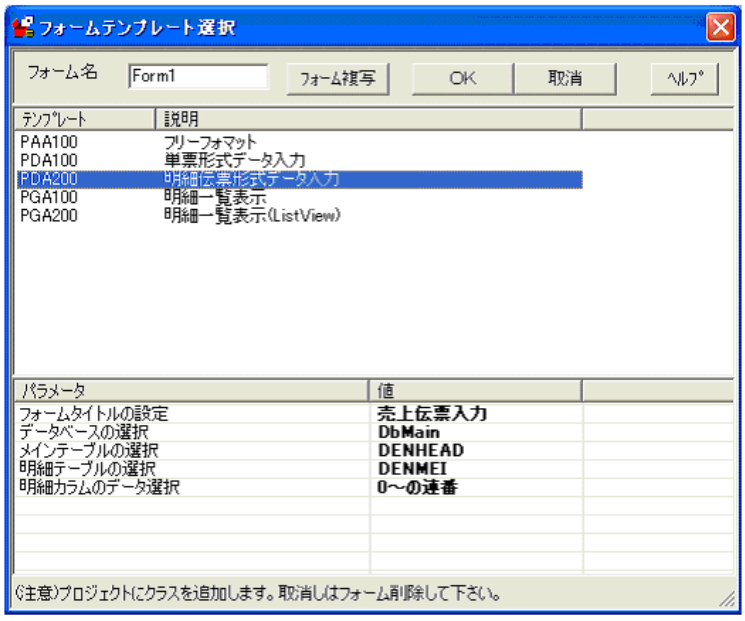
るのかを見てみましょう。

マニュアルの**「システムの概要」->「プログラム開発手順の概要」->「(3)プログラム設計」**に移ります。
ここには、プログラム設計機能が使用する**フォームテンプレート**、**コントロール制御**、**エリア制御**という３つの機能とその概念についての説明がなされています。

フォームテンプレートは、マニュアルにもある通り、汎用的なビジネスアプリケーションの処理や操作方法をいくつかの雛型としてまとめたものです。
製品出荷時に標準添付されるフォームテンプレートは下記の５つです。

１．PAA100 フリーフォーマット
２．PDA100 単票形式データ入力
３．PDA200 明細伝票形式データ入力
４．PGA100 明細一覧表示
５．PGA200 明細一覧表示(ListView)

数が少ないと思われるかもしれませんが、この標準添付のテンプレートは特に汎用性を重視しています。そしてテンプレートはお客様で改造を行っていただくことができます。
改造した結果は別名で保存することができ、そうすることによってテンプレートの種類を自由に増やせます。したがってこの標準添付テンプレートはお客様が自分に合ったテンプレートを作成していただくための雛形の雛形であるともいえます。
もちろん必ずしも改造を前提としている訳ではなくこのままでもお使いいただけます。

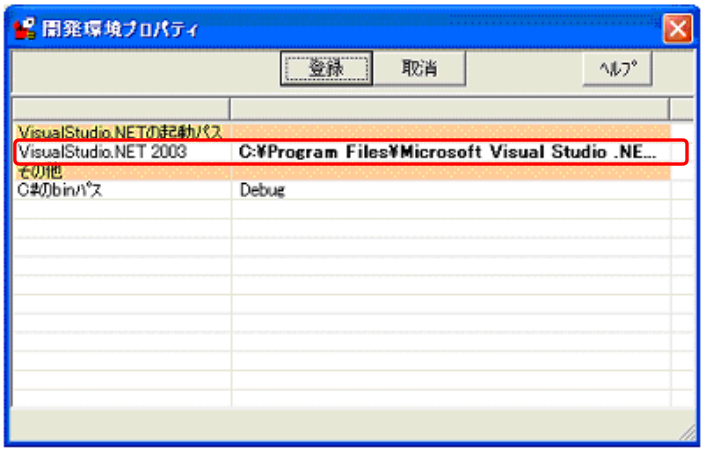


今回のHAD010「売上伝票入力」プログラムは、フォームテンプレート3番の**PDA200 「明細伝票形式データ入力」**を使用しています。

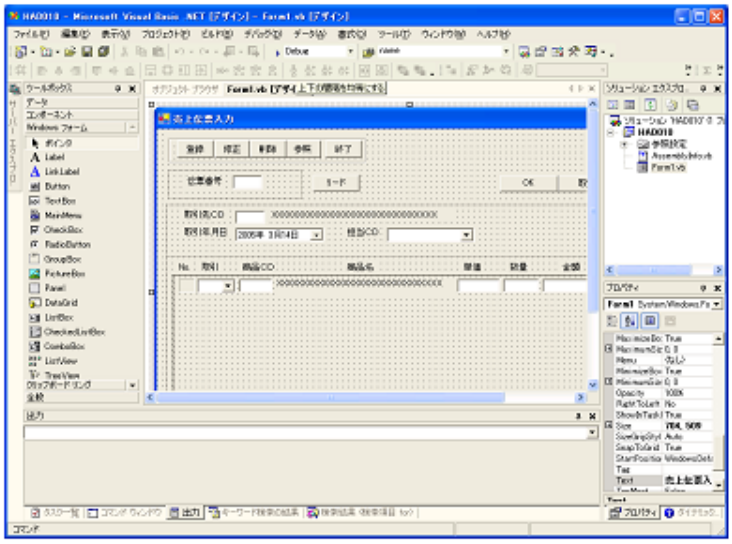
テンプレートについては、本書でも後で詳しく説明しますが、マニュアルでは**「テンプレート」**に詳細情報があります。こちらも参照してください。

新規にプログラムを作成する場合は、まずどのテンプレートを選択するかを決めます。
その上で、そのテンプレートが必要とするパラメータ（基本情報）を入力します。
売上伝票入力サンプルは、すでに開発が完了していますので、上記画面は今は実際には確認できません。上記はこれを作成開始する段階で入力した画面を再現したものです。売上伝票入力では**「PDA200 明細伝票形式データ入力」**を選択していますので、データベースとして、**DbMain**、メインテーブルすなわち伝票のヘッダーとなるテーブルとして　**DENHEAD**、明細テーブルとして　**DENMEI**　がこの段階で選択されています。
明細テーブルに明細番号を保存するキーカラムが存在する場合、テンプレートの自動生成ロジックはテーブルの書き込み時に自動連番を振ります。それを 0 起算とするか 1 起算とするか、またはテーブルにそのようなカラムが存在しない場合は、どのフィールドをキーカラムとするかを選択します。「HAD010　売上伝票入力」では、0 起算が選択されています。

サンプルでは当然のことながら設計は終わっていますが、コントロール制御、エリア制御、オウンコーディングなどを行った上で、そこまでの情報を元に VisualStudio.net のソリューション（プロジェクト）を生成し、VisualStudio.net に引き渡します。
設計作業の詳細は、後でマニュアルの**「システムの操作」->「ログイン」**以降をご覧頂くとして、ここではまず VisualStudio.net を起動してみましょう。



「LLL/.net プログラム設計」のメニューバーの**「オプション」->「環境設定」**で開発環境プロパティ画面を立ち上げ、VisualStudio.net2003 の **devenv.com** のパスを指定してあるか確認し、されていなければ指定してください。
デフォルトでは、C:¥Program Files¥Microsoft Visual Studio .NET 2003¥Common7¥IDE にあるはずです。この確認をした上で、「LLL/.net プログラム設計」のメニューバーの**「VisualStudio」->「VisualStudio の起動」**を選択すると、VisualStudio.net2003 の IDE が立ち上がり　**ソリューション HAD010(1 プロジェクト)**　が開きます。

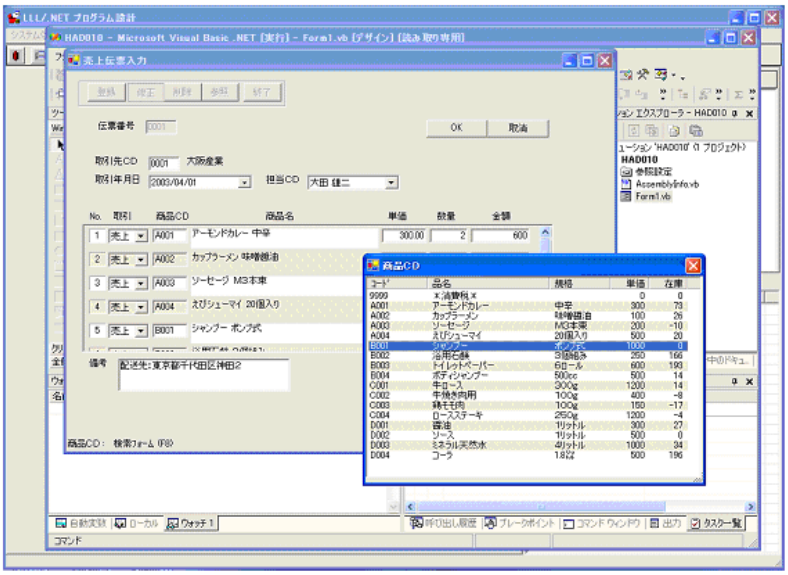


そこで IDE のメニューから**「デバッグ」->「開始」**を選択すると、売上伝票入力プログラムが起動します。

マニュアルの**「システム概要」->「作成したプログラムの概要」->「プログラムの操作」**を参照し、そこに書かれている操作をお試しください。

もちろんこの段階で、IDE から出来上がっているプログラムのソースやフォームデザインの確認、ビルドして EXE を作成することや、ブレイクポイントを設定してデバッグしてみることもできます。他のサンプルも同様の手順で試せます。

このように、LLL/.net のサンプルは、VisualStudio.net のソリューション（プロジェクト）を作成するまでに必要な LLL/.net 開発環境（プログラム設計、データベース設計）での設定情報（設定済み）と、データベースから構成されています。

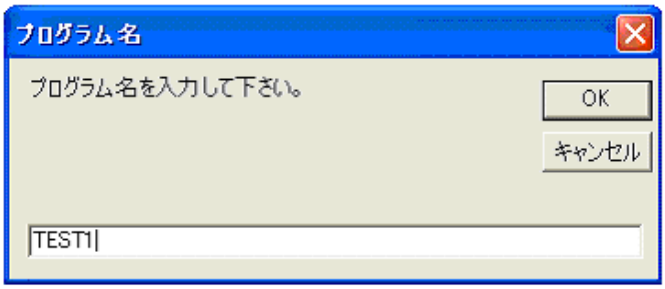


ひととおり、動きが確認できたら次は実際にプログラムを作って試してみましょう。

４．簡単な単票形式データ入力プログラム
（テストプログラムの作成その１）

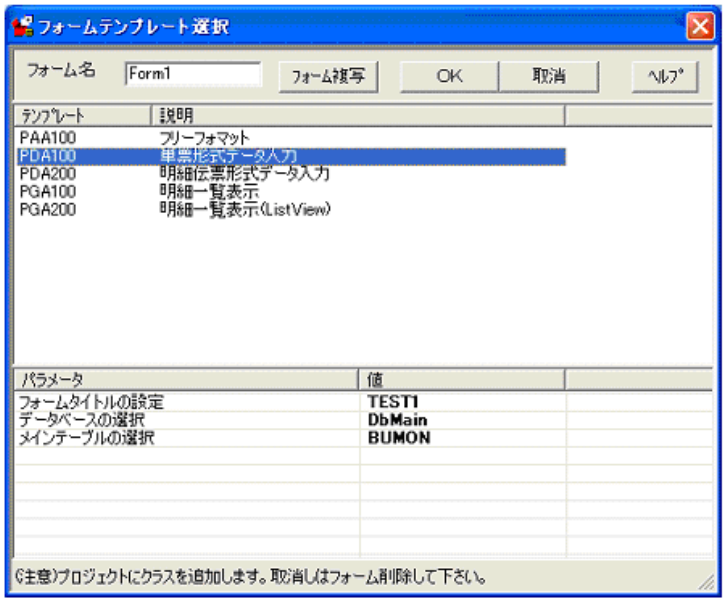
まずは、伝票よりも単純なフォームテンプレート　**「PDA100 単票形式データ入力」**　を使って簡単なマスターメンテナンスプログラムを作ってみたいと思います。
特に最初ですので、サンプルに付属するデータベースの中でも、テーブル構造が最も簡単な**部門マスター**のテーブルを使って作ってみましょう。

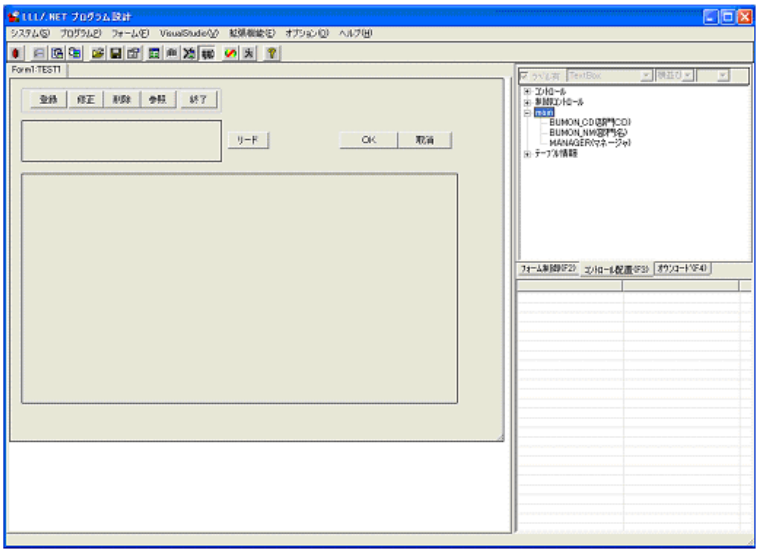
VisualStudio.net2003のIDEを終了しますと、制御がLLL/.net開発環境に戻ります。
その時、プログラム選択画面が表示されていると思いますので、**「新規プログラム」**を選びますと、今から作るプログラムの名前を入力するダイアログが表示されます。
名前は、サンプルと重複していなければ何でも構わないのですが、例えばTEST1と入力してください。



フォームテンプレート選択画面が表示されますので、リストの上から2番目にある
「PDA100 単票形式データ入力」　を選択してください。

下段のListViewに**「フォームタイトルの設定」、「データベースの選択」、「テーブルの選択」**というパラメータ項目名が表示されています。
タイトルの付け方は自由ですが、とりあえずプログラム名と同じ、TEST1、データベースとメインテーブルはサンプルで用意されている DbMain と　BUMONを選択します。



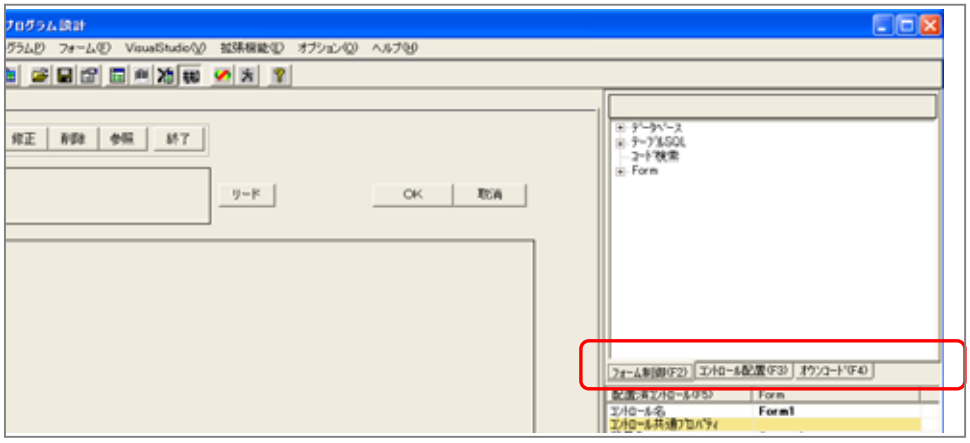


「OK」ボタンを押すと画面が**「LLL/.netプログラム設計」**に切り替わり、フォームの雛形が表示されます。

ここで**「システム」->「データベース設計」**で**LLL/.netデータベース設計**を立ち上げ、今回使用する部門テーブル「BUMON」のテーブル構造を確認しておきましょう。

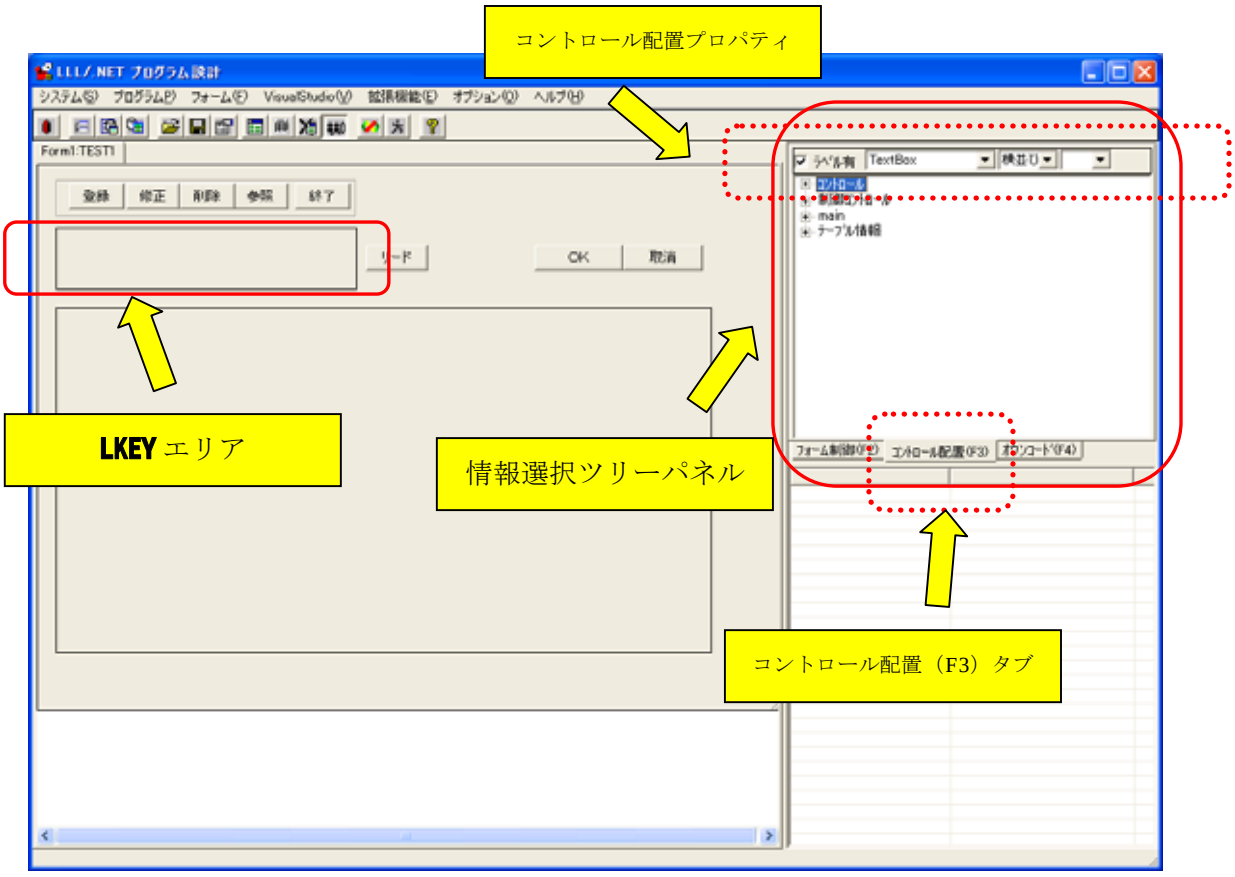
確認できたら、**LLL/.netデータベース設計**を閉じて　**LLL/.netプログラム設計**に戻ってください。

先ほどの設計済みの売上伝票とは異なり、いくつかのボタンとエリアの枠が表示されているだけで、入力や表示のフィールドは存在しません。
LLL/.netプログラム設計画面の右の方には、上段に**「情報選択ツリーパネル」**と呼ぶTreeViewを含むタブパネル、下段には空のプロティリストが表示されています。
このタブコントロールには**「フォーム制御(F2)」、「コントロール配置(F3)」、「オウンコード(F4)」**の３つのタブがあり、選択するとそれぞれページが切り替わります。

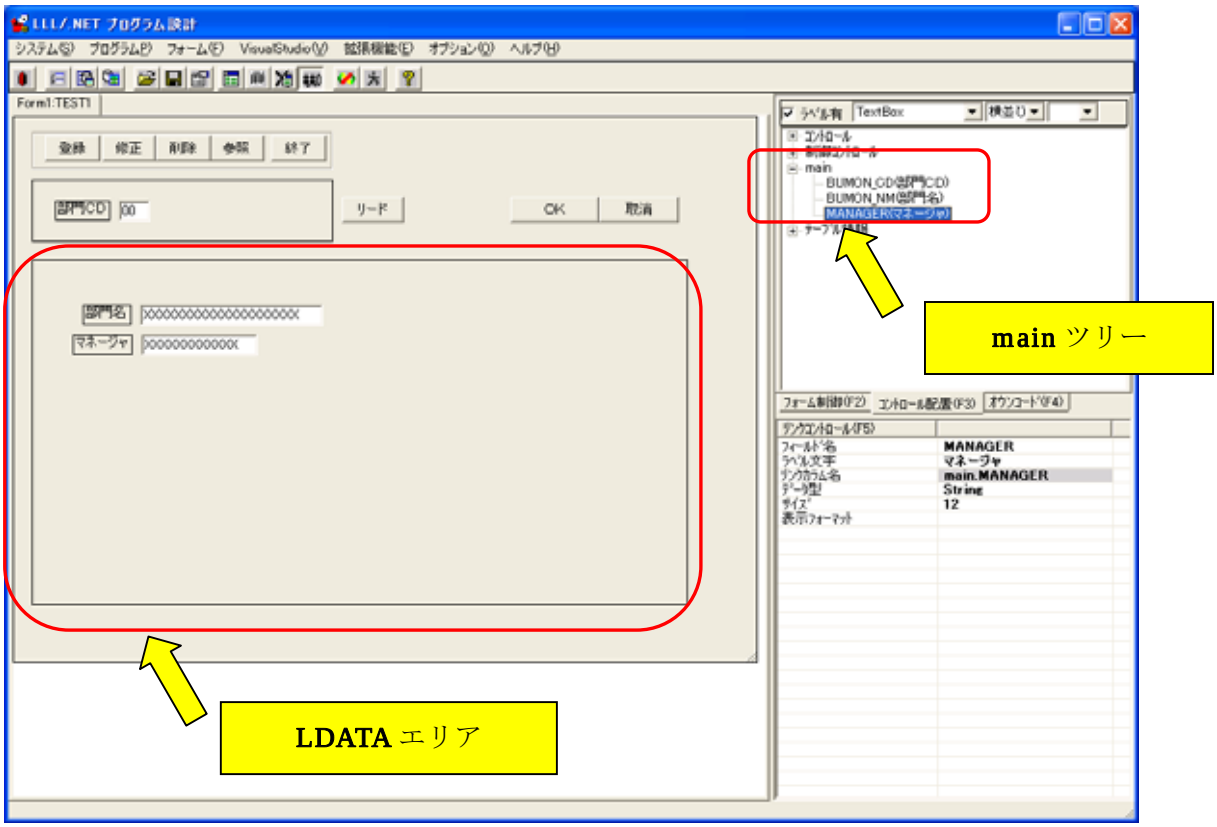


「フォーム制御(F2)」ページを**「コントロール配置(F3)」**に切り替えてください。

「コントロール配置（Ｆ３）」タブが選択されている状態の時、**「情報選択ツリーパネル」**の上枠には、**「ラベル」**というチェックボックス、**「TextBox」**が選択されているコントロール選択ボックス（ドロップダウンリスト）、**「横並び」**が選択されている方向選択ボックス（ドロップダウンリスト）、空白の、列数ボックス（ドロップダウンリスト）といったコントロール配置プロパティが表示されています。これらの詳細については、マニュアル**「システムの操作」->「プログラム設計」->「コントロール配置」**をご参照ください。

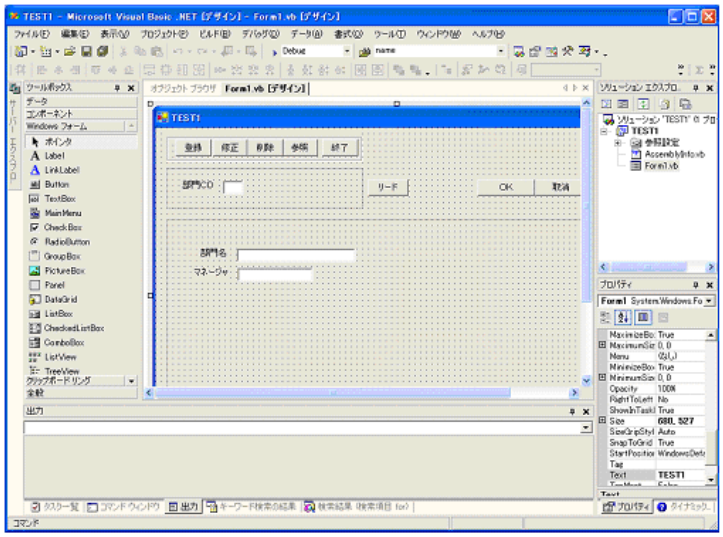


コントロール配置（F3）パネルの TreeView->**「main」->「BUMON_CD（得意先コード）」**をマウスでドラッグして左の **FORM** パネルまで移動し、上図の赤枠で囲った **LKEY** エリアの適当な位置にドロップしてください。
そこに、LABEL の見出し付きで、その見出しとは横並びになる TextBox が作成されます。
コントロール選択ボックスでコントロールを選択しなおすと、新たに選択されたコントロールでフィールドを作成することができます。
同様に BUMON_NM(部門名)、MANAGER（マネージャ）は **LKYE**（キーエリア）ではなく、その下の大きな四角　**LDATA**（データエリア）にドロップして、コントロールを配置します。
このようにしてコントロールを配置すると、同時にこれらを LLL/.net のフォーム制御コンポーネントがプログラムで制御するためのフィールド（変数）が 1 対 1 の関係で作成されています。それが情報選択ツリーパネル**「コントロール配置（F3）」**タブの隣の**「フォーム制御（F2）」**タブパネルのツリービューに存在する同名のアイテムです。



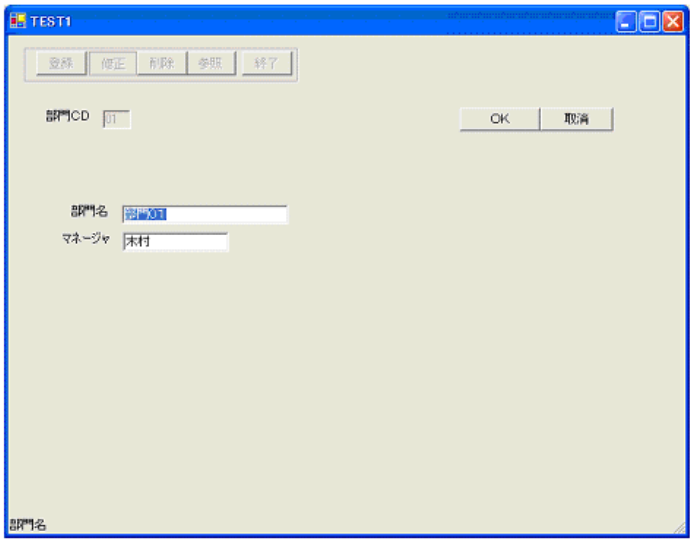
main 配下の３つのアイテム全てを配置すると出来上がりです。
BUMON_CD、BUMON_NM、MANEGER という３つの項目を、それぞれ**「情報選択ツリーパネル」->「コントロール配置（F3）」->main** から選択して配置しましたが、これら３つの項目は、**「情報選択ツリーパネル」->「コントロール配置（F3）」->「テーブル情報」-> BUMON** の

中にも存在します。しかし最初にメインテーブルとして **BUMONN** が選択された段階で、自動的に BUMONN を操作するための SQL 文が **main** という名前で作成されています。



これを**「テーブル SQL」**と呼びますが、テンプレートは、その「テーブル SQL」を使用してテーブル操作を行います。したがって、操作対象となるテーブル項目は、テーブル情報からではなく、必ず「テーブ SQL 」**名の main**のツリーからドラッグして作成しなければなりません。

メニュー**「VisualStudio」->「VisualStudio 起動」**を選び、LLL/.net での設計情報を引き渡して VisualStudio を起動します。**「デバッグ」->「開始」**で動作します。これで、部門マスターメンテナンスプログラムの完成です。



「修正」ボタンを押してから部門コードに　‘1’を入力して**「リード」**ボタンを押してください。部門 01 の部門名とマネージャ名が表示されます。

ここで例えばマネージャ「木村」を「田村」に変更し、「OK」ボタンを押します。

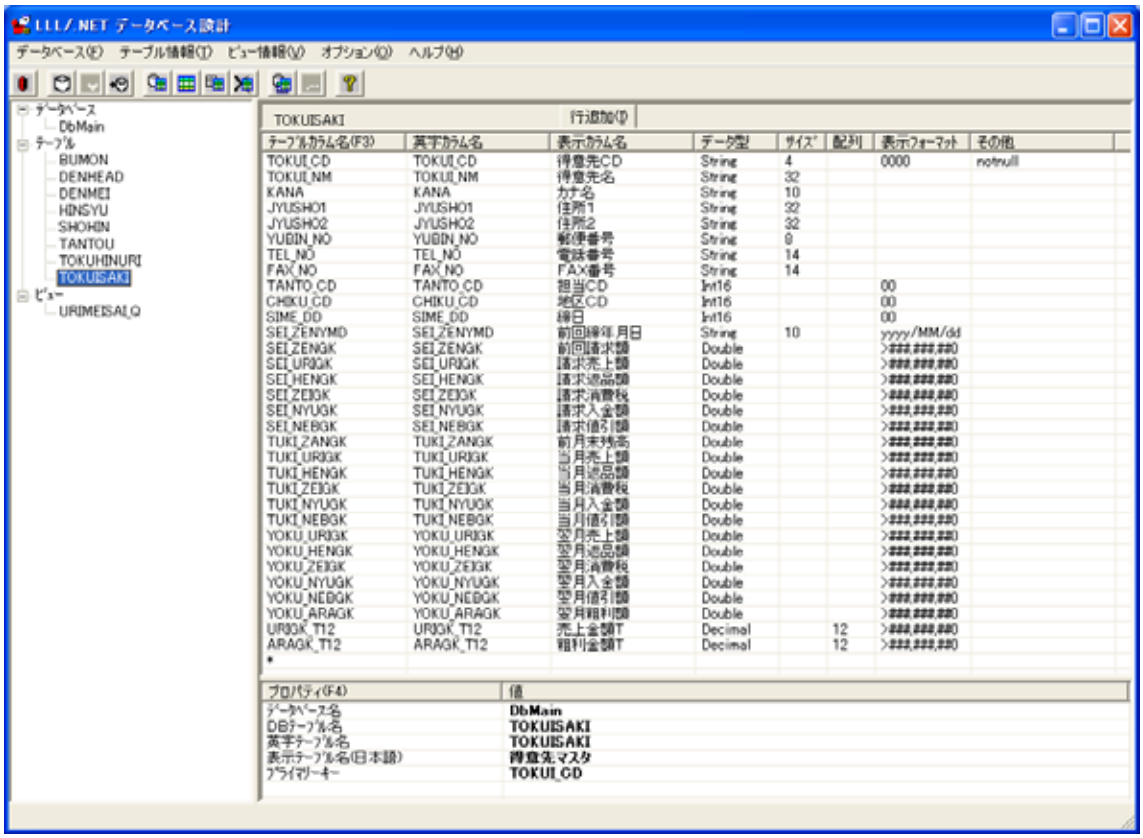
「データ修正しますか?」というメッセージボックスが表示されますので、「はい」を選んで押します。すると「データ修正を完了しました」というメッセージボックスに切り替わり確認待ちになります、「OK」ボタンを押して確認します。

修正モードでの初期状態に戻り、画面が初期化されますので、再度‘1’を入力して「リード」ボタンを押すと、今度はマネージャ名に「田村」と表示されています。

再度「木村」に修正して登録すると元に戻ります。これで、一応マスターメンテプログラムのできあがりです。「登録」「修正」「削除」「参照」の全ての動作の確認を行えます。

５．複雑な単票形式データ入力プログラム
（テストプログラムの作成その２）

もう一度、同じく単票形式データ入力（PDA100）のテンプレートを使用しますが、部門マスターより遥かに多い項目を持つ、**TOKUISAKI** テーブルを使用して、得意先マスターメンテナンスプログラムを作成します。

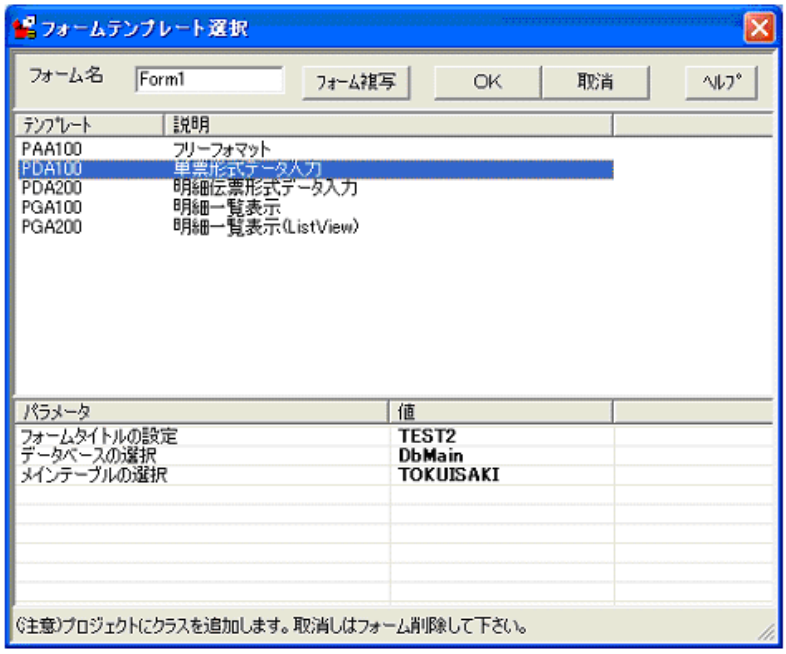


上記は、**LLL/.net データベース設計**　で見た **TOKUISAKI** テーブル です。

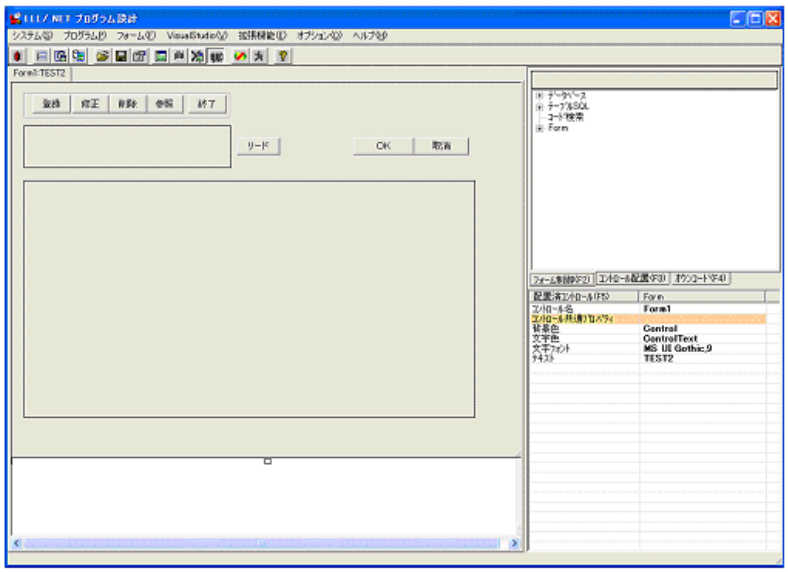
LLL/.net プログラム設計のメニュー**「プログラム」->「プログラム選択」**でプログラム選択ダイアログを表示し、**「新規プログラム」**ボタンを押し、プログラム名を TEST2 として入力し、フォームテンプレート選択画面に移ります。

テーブルは、サンプルに付属する得意先マスター **TOKUISAKI** ということで、先ほどと同じく　**単票形式データ入力（PDA100）**のテンプレートを使用します。

フォームテンプレートのパラメータは、フォームタイトルを TEST2、データベースを DbMain、メインテーブルの選択を TOKUISAKI として、OK ボタンを押してください。

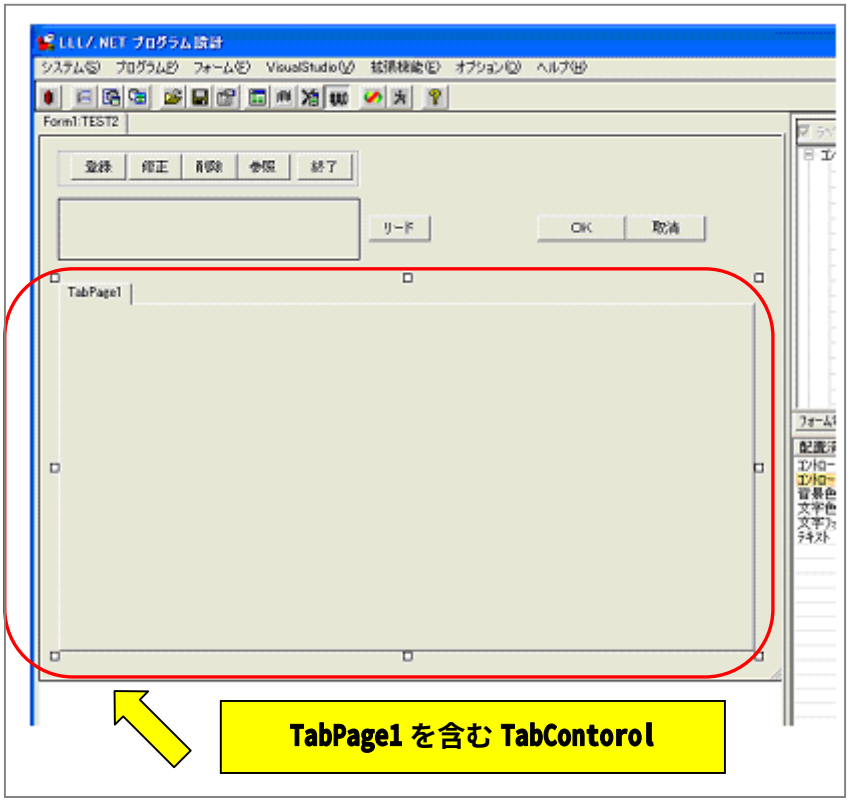


先の TEST1 の時と同様に単票形式データ入力の雛形が表示されます。
しかし今回のテーブルは項目が多いので、雛形のデータエリア **LDATA** に全てを配置することは困難です。そこで今回は **LDATA** にタブコントロールを使用し、複数のページに分けてフィールドを配置したいと思います。
FORM の雛形から **LDATA** のパネルを選択し、DEL キーを押して一旦削除してください。



その上で、**「情報選択ツリーパネル」**の**「コントロール配置(F3)」**タブを開きツリービュー最上段のトップレベルアイテム**「コントロール」**のサブアイテムにある**「TabContorol」**を選択しドラッグして**LDATA**のパネルがあったところにドロップします。

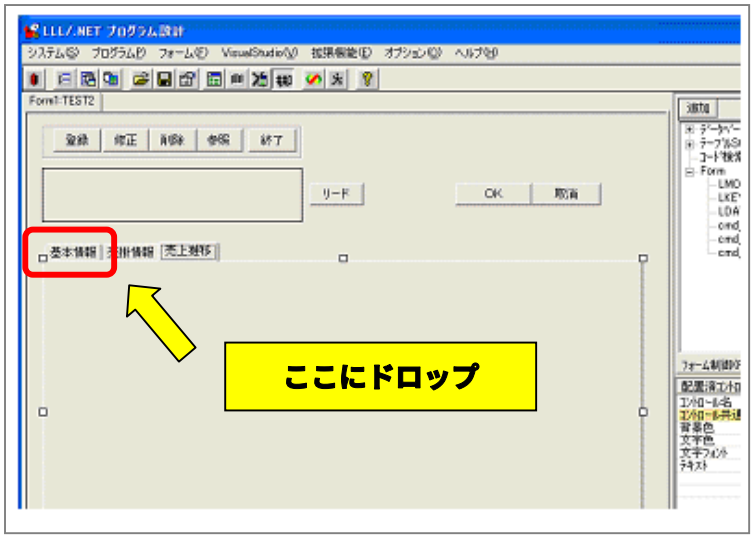
「TabContorol」はドロップされた時点で**「TabPage」**を１ページ含んでいます。改めて**「TabContorol」**を選択し、右下角をドラッグしてもとの LDATA と同じぐらいの大きさに拡大します。このとき、**「TabPage」**を選択してドラッグしないように注意してください。



この**「TabContorl」**に更に２ページ追加し、３ページで構成するようにします。

「情報選択ツリーパネル」の**「コントロール配置(Ｆ３)」**のツリービューの**「コントロール」**の下の**「TabPage」**を選択し、**「TabContorl」**にドラッグ＆ドロップします。

この時**「TabPage１」**と書かれたタブにドロップして追加してください。他の場所へのドロップはエラーになります。

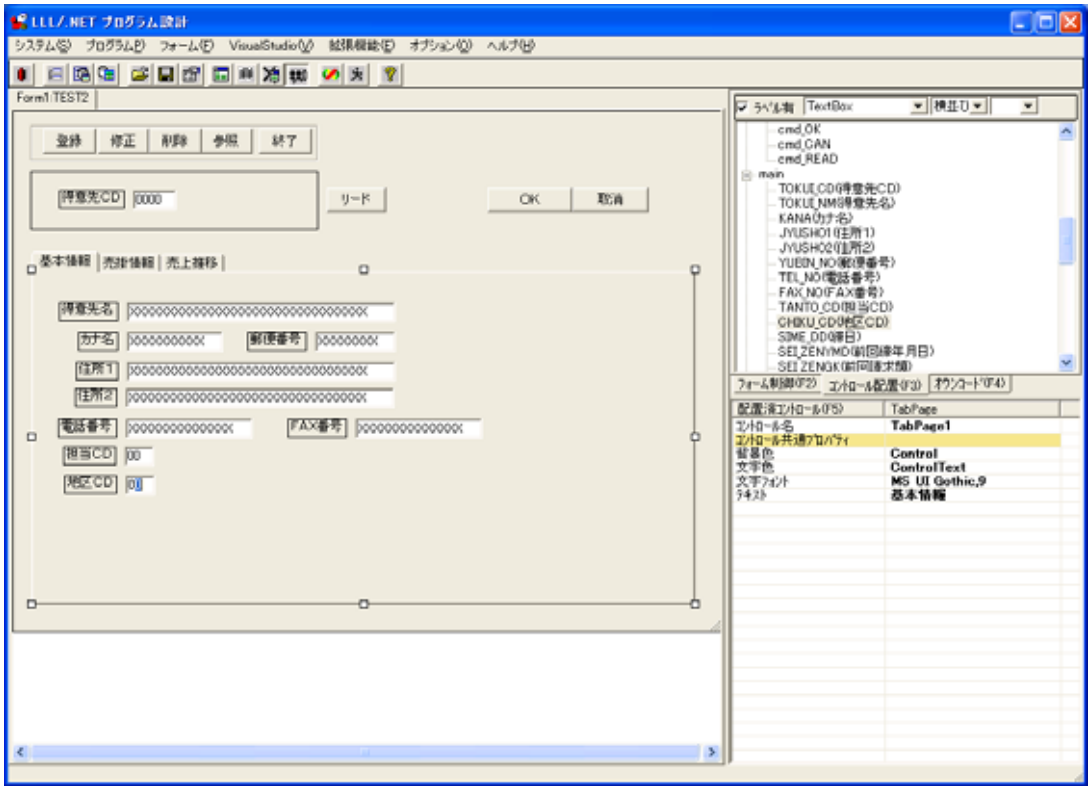


その上で改めて**TabContorl**を選択してください。確実に選択されたことを**「情報選択ツーリーパネル」**下のプロパティリストで配置コントロールが**TabControl**、コントロール名が**TabControl1**と表示されていることで確認してください。

コントロール名が、**TabPage1**や、**Form1**になっていた場合は、**TabControl1**が選択できるまでやり直してください。そして**TabControl1**の名前を、**LDATA**に変更します。

これで、雛形では Panel コントロールとして用意されていたエリア **LDATA** は、今作成した **TabControl** に置き換えられました。このように、エリアは「コンテナ制御」が可能なコントロールへなら置き換えが可能です。
さらに各 TabPage のプロパティ再下段の **「テキスト」** の、**TabPage1**、**TabPage2**、**TabPage3** を、それぞれ **「基本情報」**、**「売掛情報」**、**「売上推移」** に変更しておいてください。

次に FORM 上にフィールドを配置していきます。
まず **LKEY** エリアにこのテーブルの **プライマリーキー** である **TOKUI_CD**(得意先コード)を **「情報選択ツリーパネル」** の **main** から選んでドロップします。
ツリーの main の下部にある「テーブル情報」の TOKUISAKI の TOKUI_CD ではありません。
メインテーブルの項目は、全て **main** から選んで配置する必要があることにご注意ください。
次に **LDATA** エリアの項目を配置します。
まず **「基本情報」** タブページを選択してください。プロパティリストのテキスト欄が **「基本情報」** と表示されていることを確認してください。うまく選択できない場合は、一旦目的のタブをクリックしてからページのボディ部をクリックしてみてください。

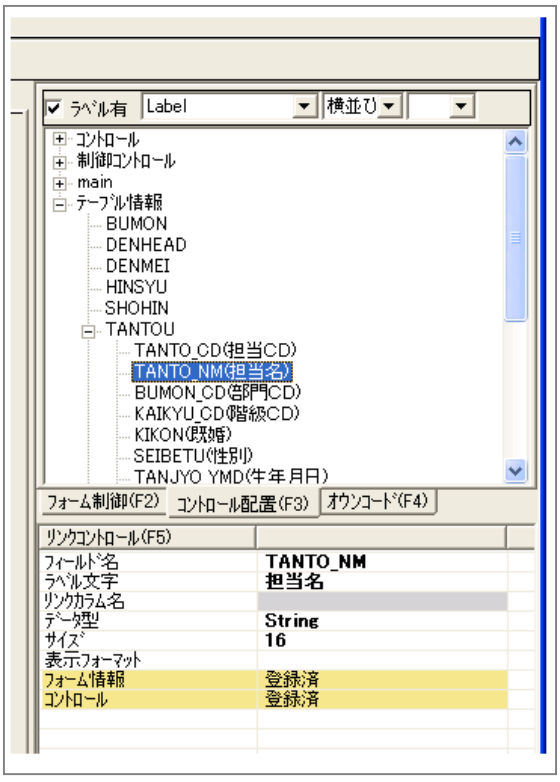


基本情報タブページへ **「情報選択ツリーパネル」** の **main** から、TOKUI_NM（得意先名）から CHIKU_CD(地区コード)までの全ての項目を順番に選択して配置してください。
配置場所はタブページ内であれば自由ですが、後で **main** テーブルが持たない情報である担

当名の表示を追加しますので、TANTO_CD（担当 CD）の近くにそのための場所を確保しておいてください。

では、担当名の表示を追加します。
「情報選択ツリーパネル」から **main** の各テーブル項目をドラッグ＆ドロップしていきましたが、自動的に見出しがラベルで作成され、その右横にその項目を入力するためのテキストボックスが並べて配置されていました。
お気付きかもしれませんが、これは**「情報選択ツリーパネル」**上枠に配置されている**「コントロール」**の設定によって変化します。

これらのコントロールについての説明は、マニュアル->**「システムの操作」->「プログラム設計」->「メイン画面」->「コントロール配置」**に出ていますので、ご確認ください。



では、先ほど言いました担当者名を表示するコントロールを配置します。
担当者名は、**main** テーブルにはありません。**「情報選択ツリーパネル」**の **main** と同一階層で、**main** の下に**「テーブル情報」**というアイテムがあります。ここから dbMain に含まれる全てのテーブルが参照できます。これらはテンプレートが自動的に操作を行う対象ではありません。ここの項目を使用する場合は、必要に応じて処理、または設定を追加定義する必要があります。
担当者名は、このサンプルデータベース **dbMain** では、TANTOU テーブルに TANTO_NM(担当名)として保持しています。この TANTOU テーブルのプライマリーキーは、TANTO_CD(担当コード)です。
そこで、先ほど画面上に配置済みである **main** テーブルの TANTO_CD をキーとして TANTOU テーブルをから合致する担当名を抽出し、画面上の TANTO_CD(担当コード)の横に表示するようにします。

まずは、実際に表示するためのコントロールを画面上に配置します。
「情報選択ツリーパネル」上枠の左から 2 番目に**「TextBox」**がデフォルトで選択されてい

るドッロプダウンリストスタイルのコンボボックスコントロールがあります。
これが、コントロール選択ボックスです。
「TextBox」だと入力ができてしまいますが、今回の担当者名はあくまでも参考に表示することが目的ですので、あえて入力できな**「Label」**コントロールを選択しておきます。
その上で、「情報選択ツリーパネル」の**「テーブル情報」->「TANTOU」->「TANTO_NM(担当名)」**をドラッグ＆ドロップしてForm上に配置してください。

これでフィールドの配置はできましたが、前述したようにテンプレートが自動的に操作してくれるのは、**main**テーブル、すなわち今回のケースではTOKUISAKIテーブルの項目のみです。そこで担当者名表示のための設定を追加します。

この担当者名は、画面上の担当CDをキーとして検索します。
そこで「情報選択ツリーパネル」のページを今までの**「コントロール配置(F3)」**から左隣のタブをクリックして**「フォーム制御（F2)」**に切り替えます。
その上でツリービュー上の**「Form」->「LDATA」->「TANTO_CD」**を選択します。**「TANTO_NM」**ではありませんので、念のため。

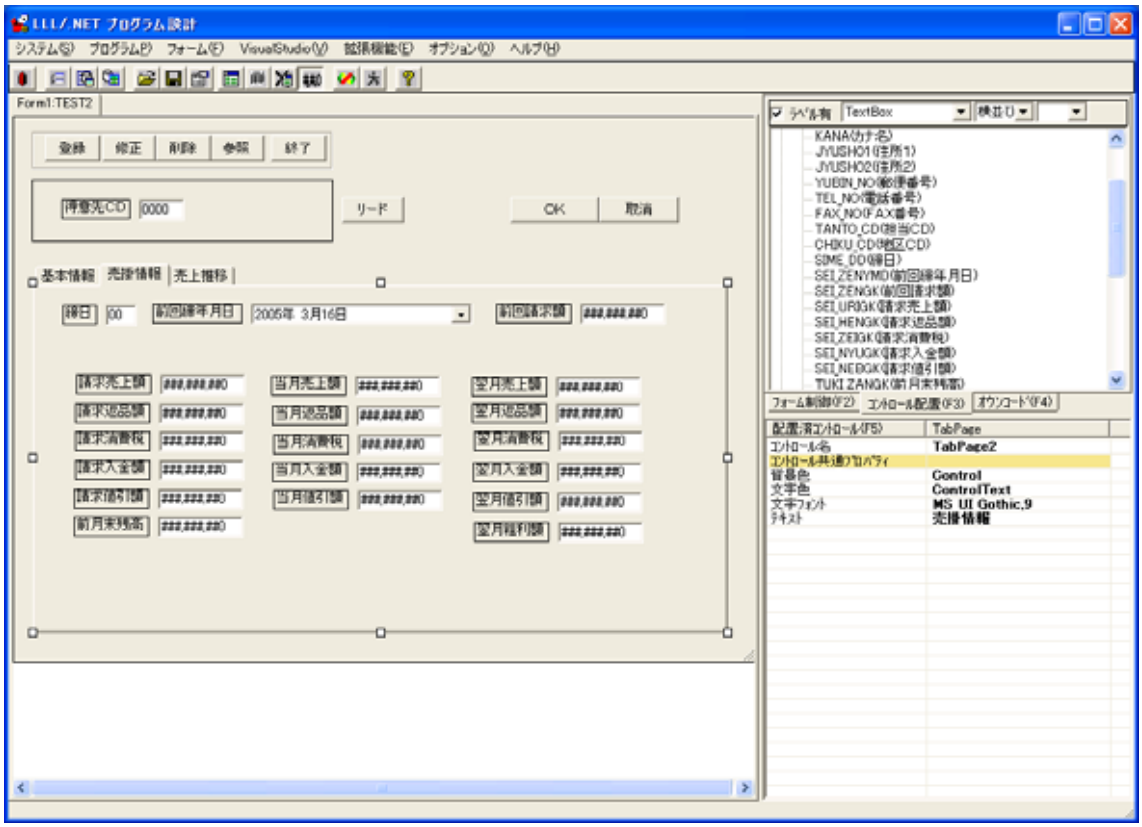
フィールド制御プロパティの先頭のフィールド名が、TANTO_CDになっていることを確認した上で、プロパティの「データ表示の設定」カテゴリの中に**「テーブル参照」**という項目があることを確認して選択し、そこにフォーカスすると現れるプッシュボタンを押して**「テーブル参照の編集」**ダイアログを表示させてください。

そのダイアログに、参照するテーブル名（TANTOU）と、その中の参照カラム（TANTO_NM）と参照した結果を表示する画面上の設定フィールド（TANTO_NM）を指定してください。
これで、画面上のTANTO_CDに値がセットされると、自動的にTANTOUテーブルを参照して該当するTANTO_NMの値を、画面上のTANTO_NMに表示させるというロジックを追加したことになります。

このダイアログについての説明は、マニュアル->**「システムの操作」->「プログラム設計」->「プロパティ編集画面」->「テーブル参照の編集」**に詳しく出ています。

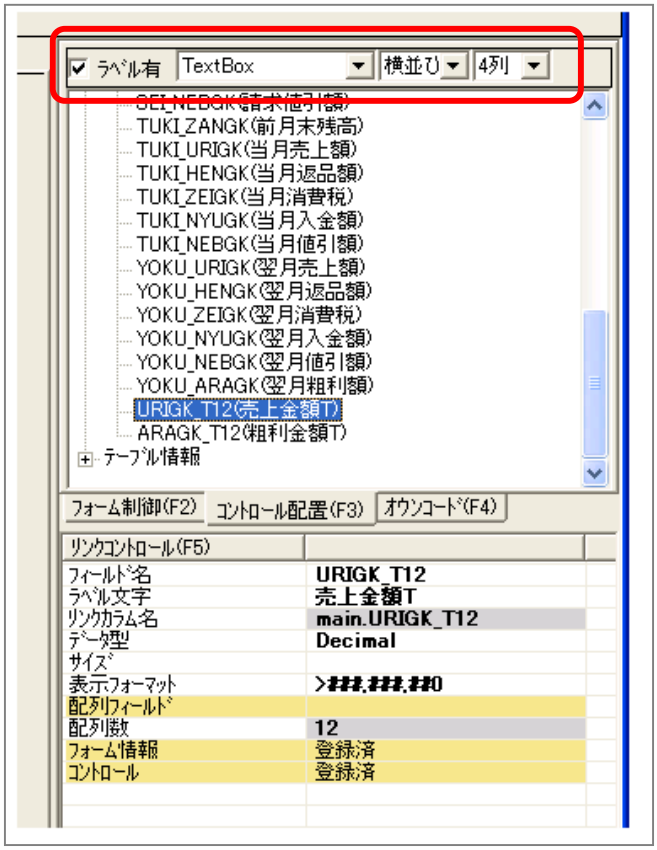
LDATA の残ったタブページの作成を行います。
Form の「売掛情報」タブをクリックしてからタブページのボディを再度クリックして「売掛情報」ページが選択されていることを確認してください。プロパティリストのコントロール名が、「TabPage2」、テキストが「売掛情報」と表示されていれば正解です。

「情報選択ツリーパネル」->「コントロール配置（F3)」のツリービューの **main** から、SHIME_DD(締日)から YOKU_ARAGK(翌月粗利額)までを順番にドラッグ＆ドロップして配置します。SEI_ZENYMD(前回締年月日)は、テキストボックスではなくカレンダーコントロールである DateTimePicker を使うと面白いと思います。



最後に３ページ目の売上推移を作成します。先ほどと同様 Form の「売上推移」タブをクリックしてからタブページのボディを再度クリックして「売上推移」ページが選択されていることを確認してください。プロパティリストのコントロール名が、「TabPage3」、テキストが「売上推移」と表示されていれば正解です。

最後のページには、main テーブルで残った２つの項目 URIAGE_T12(売上金額 T)と ARAGK_T12(粗金額 T)を配置します。実はこの２つの項目は、今までのテーブル項目とは少し違います。
これは２つの項目ではなく、それぞれが RDB 上では 12 個のカラムとして定義されており、これを LLL/.net が配列として取り扱うという特殊な項目です。

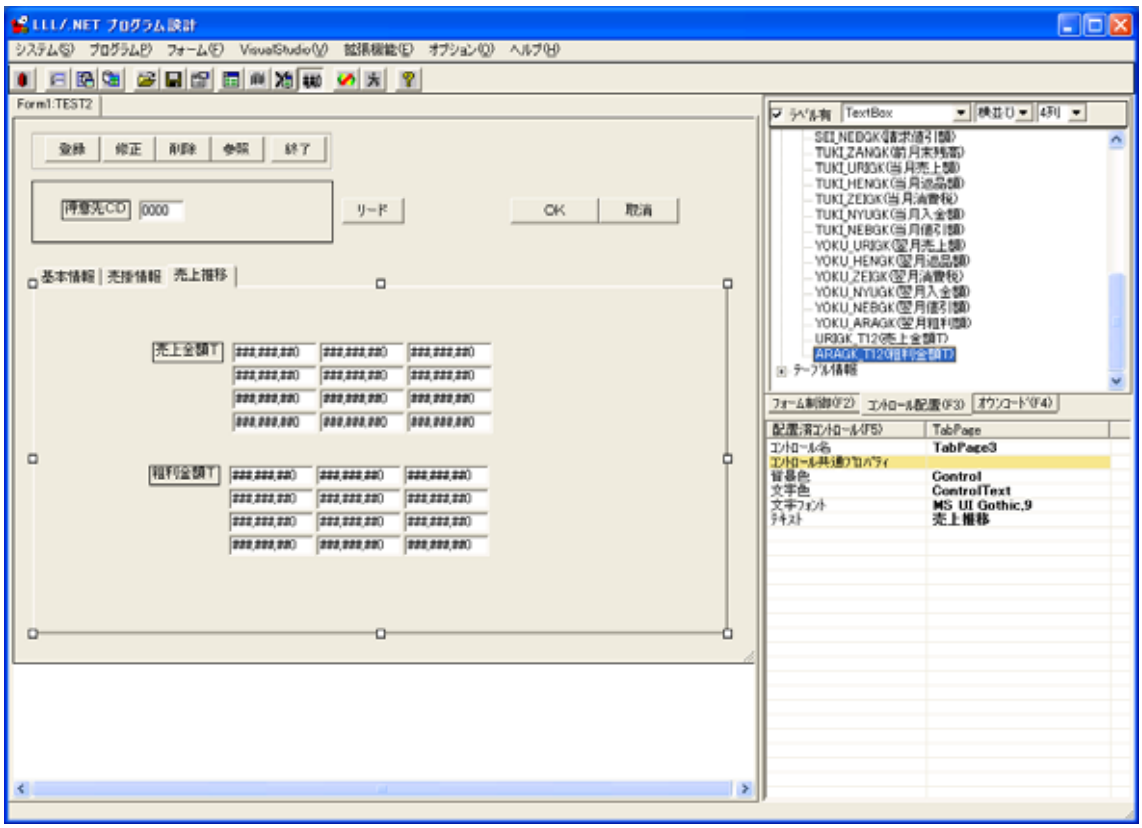


したがって、これをそのままドラッグして FORM に貼り付けようとすると見出しラベルの横に 12 個のテキストボックス 1 列に並べられ FORM をはみ出してしまう事態が発生します。

これでは困りますので、ドラッグを開始する前に、とりあえずマウスでクリックして選択だけ行ってください。すると「情報選択ツリーパネル」上枠の右端にある今まで空白だったドロップダウンスタイルコンボボックスの中に**「1 列」**と表示されます。「表」の概念からいうと、これは「列」ではなく「行」と表現すべきかという気もしますが、とにかく 1 列（1 行）では FORM に収まりきりませんので、これを**「4 列」**に選択しなおします。

これをドラッグすることで 1 行 3 個のテキストボックスを 4 行たてに繰り返す形で 12 個のテキストボックスを配置できます。

とりあえずこれで完成です。



これで基本的な機能は完成しました。
「修正」モードで、得意先 CD 欄に’1’を入力して「リード」ボタンを押すと、コード’1’の得意先情報が TabContorol 上の各フィールドに表示され、自由に編集できます。

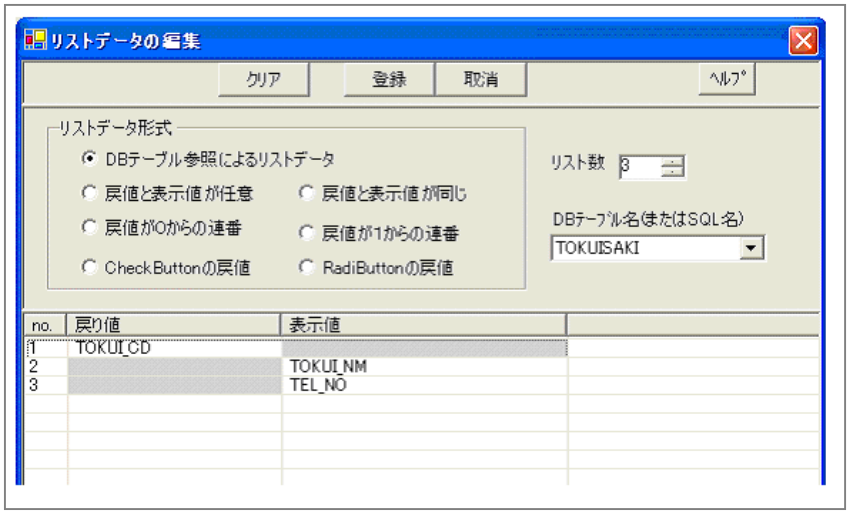
ただ、これでは目的の得意先コードが何番か分かっていないと操作できません。
そこで、得意先コードを入力する代わりに、得意先コードの一覧表を別ウインドウとして表示し、そこで得意先の名前から選択できると便利だと思われます。
このような検索機能を持ったウインドウは良く使われる機能ですが、LLL/.net でこれを簡易に提供するのが、**「コード検索」**という機能です。

実際に作ってみます。「情報選択ツリーパネル」のページが**「フォーム制御（F2)」**になっていることを確認してください。
ツリービューの第一階層のアイテムの上から 3 番目に**「コード検索」**というアイテムがあります。マウスでこれを選択しておき、その状態で「情報選択ツリーパネル」の上枠に表示されている**「追加」**というボタンを押します。
プロパティリストに「コード検索」のプロパティが表示されます。

コード検索名は Win1 となっています。この名前を変更せずにコード検索をさらに追加していくと自動的に Win2、Win3 という名前が付けられていきます。

Win1 では何のコード検索か分かりにくいので Win1 を、例えば TOKUI という名前に変えます。

そしてこのプロパティの「検索データの設定」カテゴリの先頭項目である**「リストデータの編集」**を行います。この項目をクリックして選択し、表示されるボタンを押してください。



リストデータ編集ダイアログが表示されたら、**「リストデータ形式」**で**「DB 参照によるリストデータ」**を選択します。リスト数はデフォルトでは‘２’ですが、‘３’に変更しておきます。これで３つまでの項目をリスト表示できるようになりました。これは必要に応じて変更してください。

「DB テーブル名（または SQL 名)」に TOKUISAKI を選択して入力します。

この段階で、**TOKUISAKI** の**プライマリーキー**である **TOKUI_CD** が自動的に戻り値としてセットされます。必要があればここで選択しなおせますが、今回はこのままにしておきます。

次に戻り値以外に検索ウインドウ内に表示させたい項目を「表示値」として設定します。

今回は、リスト数を３としましたので、戻り値以外に後２つの表示項目を設定できます。

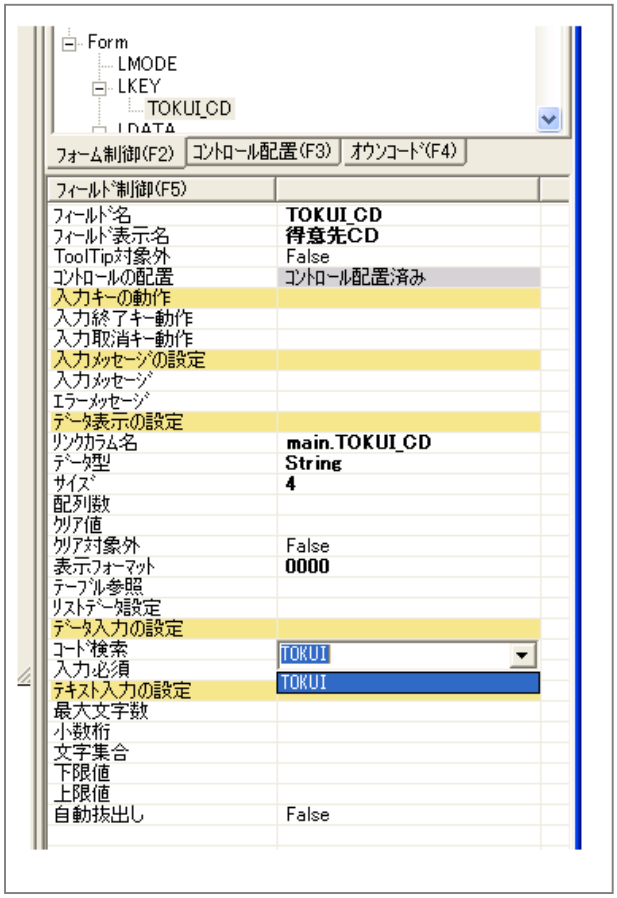
ここでは、TANTOU_NM と TEL_NO を選択しておきます。

これで検索ウインドウの用意ができましたので、これを呼び出す設定を行います。

この検索ウイドウは画面上の TOKUI_CD　で使用しますので、**「情報選択ツリーパネル」->**

「フォーム制御（F2)」のツリービューから、**Form->LKEY->TOKUI_CD** を選択し、これのプロパティを表示させてください。

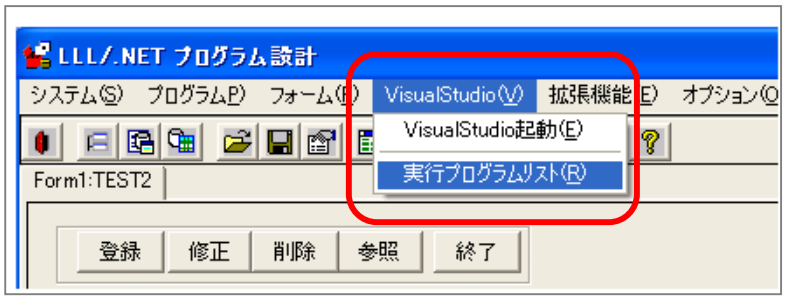
「テーブル参照」の少し下に「データ入力の設定」カテゴリがありその中に、**「コード検索」**という項目があります。この項目を選択すると、候補がドロップダウンリストで確認できます。コード検索を多く作成していると、一覧になって表示されるのですが、今回は TOKUI しか作成していませんので、これだけが表示されます。**TOKUI** を選択します。これで完了です。



同様に、「基本情報」タブの担当者名も、担当 CD 欄で、コード検索できると便利です。もう一度確認の意味で、手順だけ下に記します。

TANTOU という「コード検索」を追加し、リストデータの編集で、DB テーブル参照によるリストデータを選択し、リスト数２、DB テーブル名（または SQL 名）に TANTOU を指定、戻り値は自動的に TANTOU_CD が表示されるのでそのまま、表示値に TANTOU_NM を指定するとコード検索用ウインドウができます。

そして、「情報選択ツリーパネル」->**「フォーム制御（F2)」**のツリービューから、**Form->LDATA->TANTOU_CD** を選択し、これのプロパティ「コード検索」に TANTOU を指定すると完了です。コード検索設定の詳細については、マニュアル->**「システムの操作」->「プログラム設計」->「プロパティリスト」->「コード検索情報」**をご参照ください。

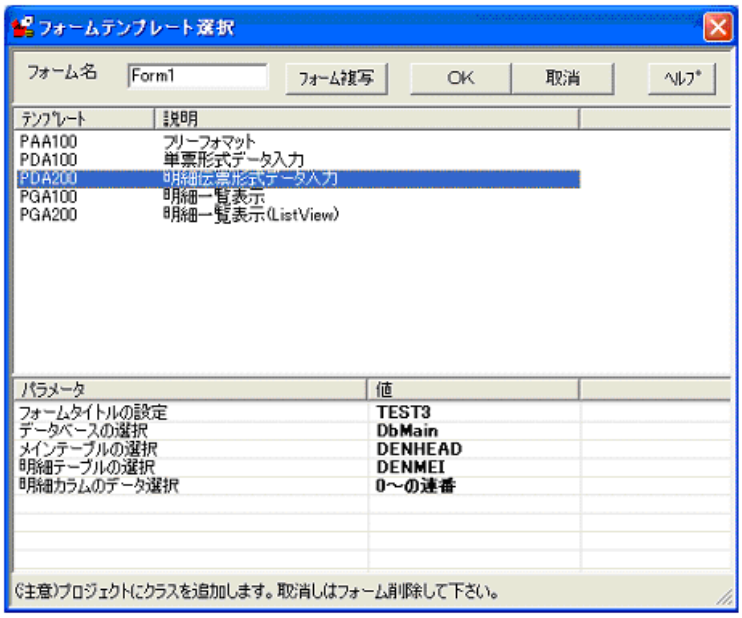


VisualStudio.net2003 を起動してデバッグで動作させてみてください。ビルド(B) して、LLL/.net プログラム設計に戻ると、「実行プログラムリスト(R)」からも起動できるようにできます。

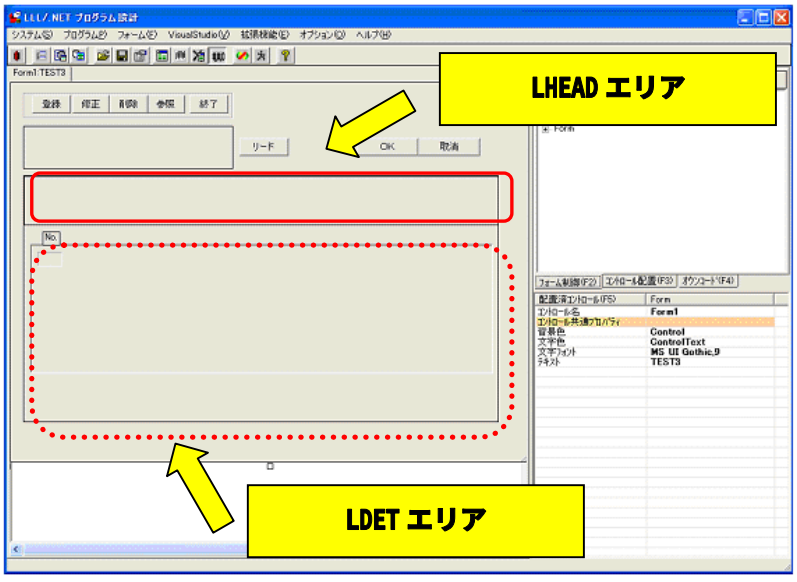
６．明細伝票形式データ入力プログラム
（テストプログラムの作成その３）

つづいて、さらに複雑な **明細伝票形式データ入力（PDA200）** フォームテンプレートを使用して、最初に見た売上伝票入力のようなプログラムを作ります。

LLL/.net プログラム設計のメニュー **「プログラム」->「プログラム選択」** でプログラム選択ダイアログで **「新規プログラム」** ボタンを押し、プログラム名を TEST3 として入力し、フォームテンプレート選択画面に移り、**「PDA200 明細伝票形式データ入力」** を選択します。



フォームテンプレートのパラメータとしては、フォームタイトル=TEST3、データベースの選択=DbMain、メインテーブルの選択=DENHEAD、明細テーブルの選択=DENMEI、明細カラムのデータ選択=0からの連番　と入力(又は選択)します。
これで「OK」を押し、プログラム設計を立ち上げます。

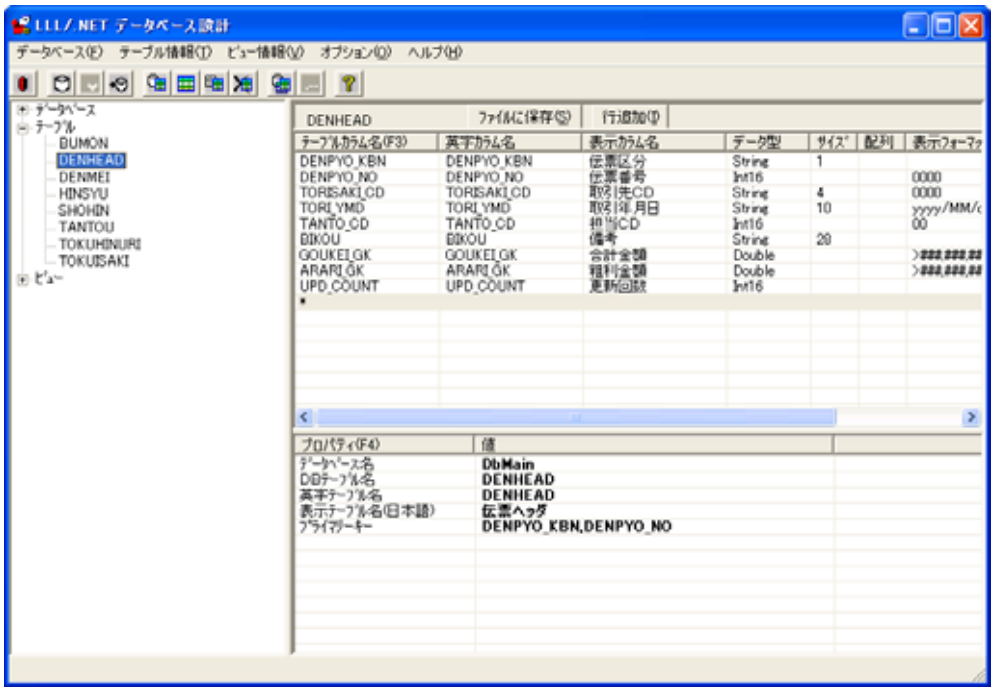


明細伝票形式データ入力（PDA200）フォームテンプレートの雛形FORMには、**LKEY**、**LDATA**、というエリアがあります。ここまでは先ほどの単票形式データ入力（PDA100）と同じですが、LDATAの中にさらに **LHEAD**、**LDET** という子エリアが含まれています。

これらのエリアは、複数配置されたコントロールを一括制御するための仕組みです。
エリアの詳細は、マニュアル**「プログラム開発手順の概要」->「(3)プログラム設計」->「エリア制御」**、ならびに**「システムの操作」->「プログラム設計」->「プロパティリスト」->「エリア情報」**をご覧ください。

LKEY に、メインテーブルのキー項目を配置します。情報選択ツリーパネルの「コントロール配置（F3)」タブで表示されるツリーから**main**を開きキー項目を**LKEY**にドラッグします。

今回 main に指定された伝票ヘッダーテーブル DENHEAD には、伝票区分と、伝票番号からなる複合キーがプライマリキーとして定義されています。両方とも **LKEY** に配置します。
プライマリーキーについては、LLL/.net データベース設計でも確認できます。

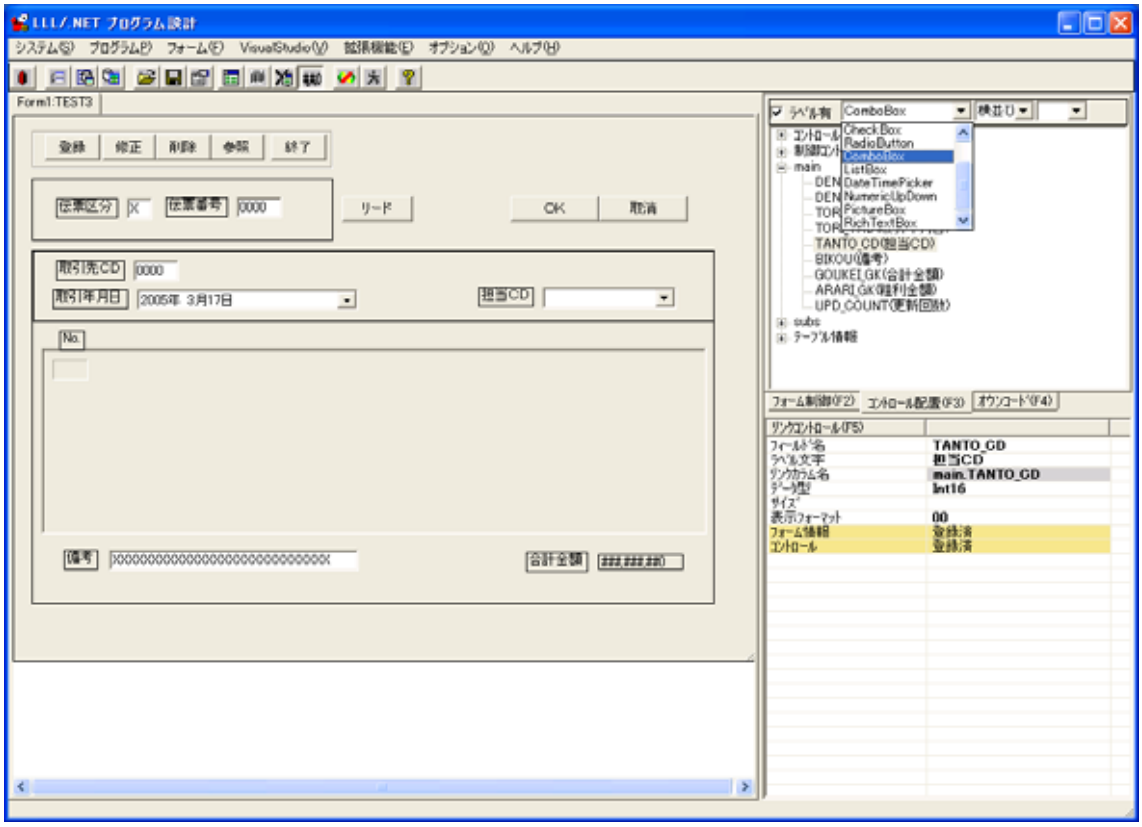


次に、キー以外のヘッダー情報を **LHEAD** エリアに配置します。
テキストボックスばかりでは､面白くありませんので､TORISAKI_CD(取引先 CD)は TextBox、

TORI_YMD(取引年月日)は DateTimePicker、TANTO_CD(担当 CD)は、ComboBox で作成しましょう。

繰り返しになりますが、これらは「情報選択ツリーパネル」->「コントロール配置 (F3)」のツリービューの上枠左から 2 番目の「コントロール選択ボックス」で切り替えができます。目的のコントロールに切り替えてから、ツリービューの main からアイテムをドラッグ＆ドロップして配置します。

特にフッターを制御するエリアはありません。BIKOU(備考)と GOUKEI_GK(合計金額)は LDATA エリアの中の LDET(明細)エリアの下の空き領域に配置します。
合計金額は、単独で入力されると整合性がとれませんので、Label で配置してください。



取引先コードの横には、このコードをキーとして TOKUISAKI テーブルを参照し、得意先名を表示するようにします。
そのため、コントロールとして Label を選択した上で、**「テーブル情報」->「TOKUISAKI」->「TOKUI_NM」**をドラッグ&ドロップして配置します。

引き続き、明細を作成します。明細は、テンプレートで指定した DENMEI を使用します。

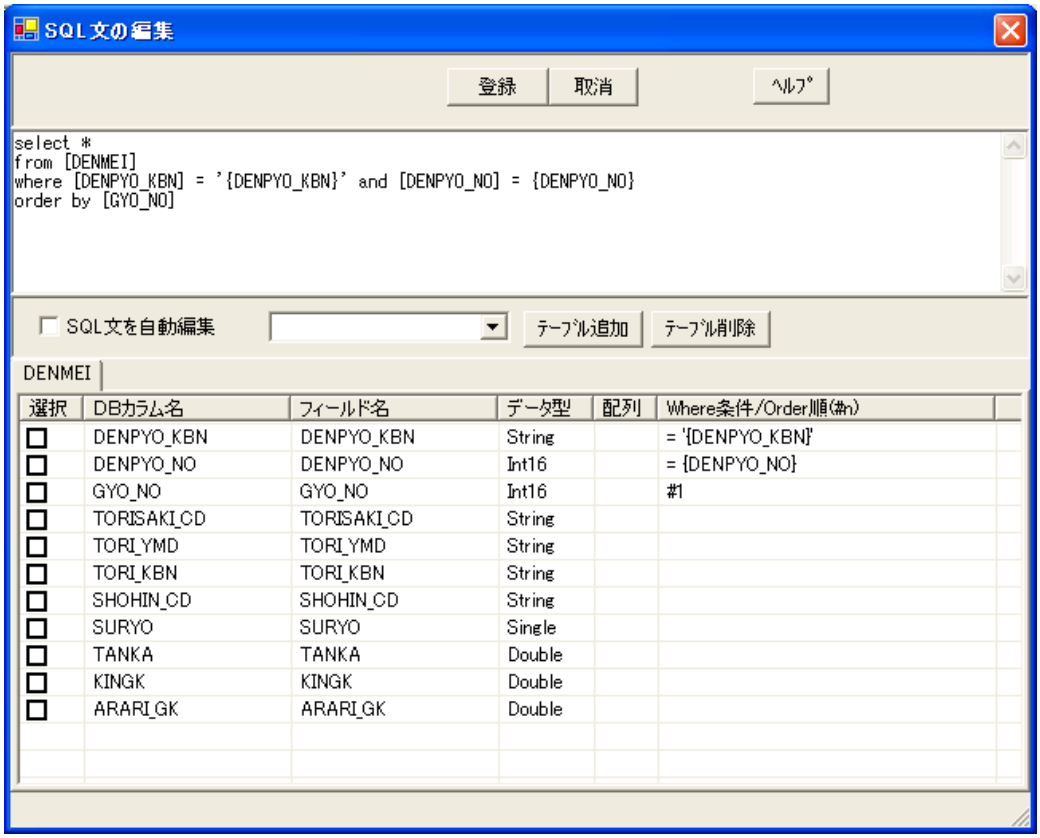
明細伝票形式データ入力（PDA200）では、開発の開始時にメインテーブル以外に、明細テーブルを必須条件として選択しています。
メインテーブルすなわちヘッダーテーブルと、明細テーブルの複数レコードで 1 枚の伝票が構成されるのですから当然ですが、必須条件となっています。

今回の TEST3 では、明細テーブル DENMEI(伝票明細)が選択されています。
メインテーブルを選択すると、main という SQL 文が生成されるというのは、単票形式データ入力（PDA100）でも同じですが、伝票形式データ入力では main 以外に、subs という名前の SQL 文が作成されます。

main、subs などテンプレートの指定で SQL 文を簡易的に自動作成している機能は下記のようになっています。

main：プライマリキーを where 条件にして SQL 文作成。
subs：プライマリキーの下位キーを除いて where 条件にし、
　　下位キーを ″order by″ にして SQL 文作成。
query：″select * from テーブル″

この内、明細伝票形式データ入力（PDA200）では、main、と subs の 2 種類をデフォルトで使用します。
こうして作成された SQL 文は、「テーブル SQL」として保存されており、「情報選択ツリーパネル」->「フォーム制御（F2)」のツリービューで確認できます。
「情報選択ツリーパネル」->「フォーム制御（F2)」のツリービューに、テーブル SQL と同名の main、subs というアイテムがあります。



これらは、main､subs というテーブル SQL が DENHEAD や、DENMEI を Select した結果と同じです。

Test1 でも main について簡単に説明しましたが、テンプレートが自動的に行うテーブル操作は、このテーブル SQL に基づいて行われています。したがって対象となる画面上のフィールド、例えば商品コードを明細エリア（LDET）に配置する場合、**「コントロール配置（F3)」->「テーブル情報」->DENMEI->SHOHIN_CD**、ではなく、必ず**「コントロール配置（F3)」->subs ->SHOHIN_CD** をドラッグ＆ドロップしなければならないということにご注意ください。

明細エリア(LDET)へのフィールド配置は、他エリアと少しだけ様子が違うことがあります。明細エリア(LDET)以外のエリアにコントロール配置を行った場合、マウスでドロップした場所、すなわちマウスの左ボタンから手を放した場所に、指定のコントロールが作成され

ます。しかし、明細エリア（LDET）では、エリア内のどこにドロップしても、エリアの左上隅から順番に最後に配置したコントロールの右隣へコントロールを配置します。

これは、明細行は、一般に表のような表現で、行は繰り返して複数表示されるという前提で考えた時に便利なように行っている意図的な操作です。
しかし、そのルールからあえてはずれた配置を行いたいということもあるかもしれません。その場合は、一旦上記ルールで自動配置されたコントロールを改めてマウスで選択してすきな場所に移動できます。こうして移動されたコントロールを LLL/.net が再度勝手に移動することは行いません。

では、まず明細の最初として**「情報選択ツリーパネル」->「コントロール配置（F3)」->subs ->SHOHIN_CD** を選択し、**LDET** 内にドロップしてください。

テーブル上の次の項目は、数量ですが、商品コードの横には、やはり商品名を表示したいのでこれを商品マスターからドラッグします。
ヘッダーへ得意先名を配置したのと同じ要領ですが、入力されると困るので、コントロールとして Label を選択した上で、**「情報選択ツリーパネル」->「コントロール配置（F3)」->テーブル情報 ->SHOHIN ->SHOHIN_NM(商品名)**をドロップしてください。

この商品名は、**subs** に含まれるデータではなく、フォームテンプレートのテーブル自動操作対象ではありません。単なる参照ですがそれでもテンプレートが関与しない以上、設定を行わなければ、自動的に表示されたりするわけではありません。
「情報選択ツリーパネル」->「フォーム制御（F2)」で Form->「LDATA」->「LDET」->SHOHIN_CD を選択し、プロパティ「テーブル参照」で DB テーブル名=SHOHIN、参照カラム名＝SHOHIN_NM、設定カラム名= SHOHIN_NM、と設定してください。

次に**「情報選択ツリーパネル」->「コントロール配置（F3)」->subs** にもどり、SURYO(数量)、TANKA（単価)、KINGK（金額）と順番に配置します。
数量と単価は入力可能なので TextBox、金額は入力不可にしたいので Label を選択してから配置してください。

次に、Test2 でも作成しましたが、得意先コードと、商品名コード欄にそれぞれ得意先名と商品名を検索できる「コード検索」を作成しておきましょう。説明は省略します。

ヘッダーの担当者 CD 欄は、コンボボックスで作成しましたが、これの設定を説明します。

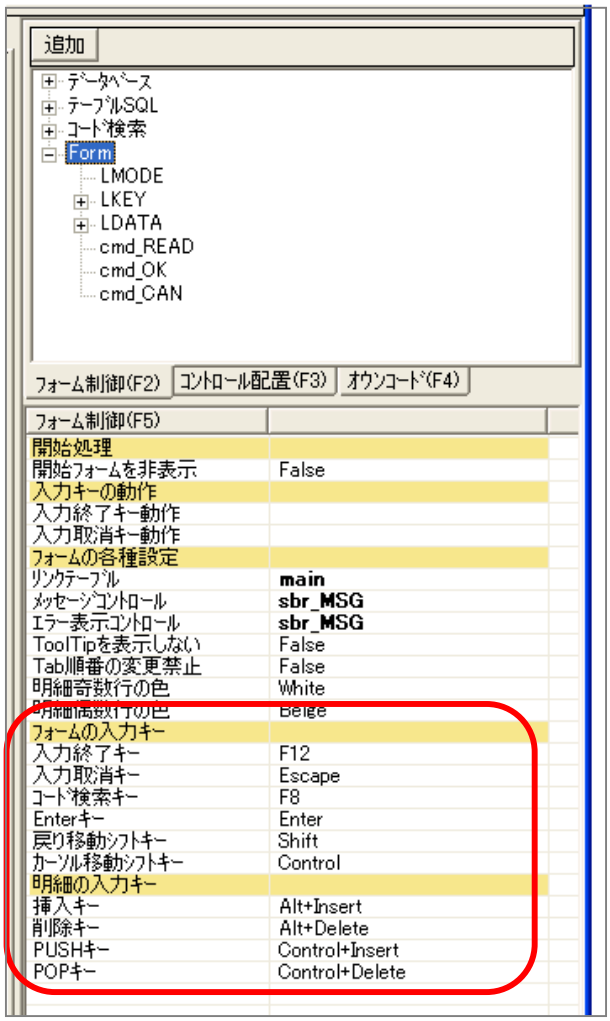
「情報選択ツリーパネル」->「フォーム制御（F2）」->Form->「LDATA」->「LHEAD」->TANTO_CD 選択し、プロパティの「データ表示の設定」カテゴリの一番下の方に**「リストデータ設定」**という項目があります。ここを選択すると表示されるボタンを押して、下記の「リストデータの編集」ダイアログを出してください。

そして、リストデータ形式＝ＤＢテーブル参照によるリストデータ、リスト数＝２ DB テーブル名（または SQL 名）＝TANTOU、戻り値＝TANTO_CD、表示値＝TANTO_NM、と設定してください。
当然ですが、この設定はリスト属性を持つコントロールにしか行えません。ではここまでで、一度実際に動かしてみましょう。
まだ計算ロジックは入れていませんので、当然ながら計算は行いません。
「LLL/.net プログラム設計」 ->「VisualStudio」 ->「VisualStudio の起動」、で VisualStudio.net を起動し、**「デバッグ」->「開始」**でプログラムを開始してくださいます。

ちなみに、明細エリア(LDET)での固有のキーボード操作として、LLL/.net では下記の操作が自動的にサポートされます。

１．カーソル移動キー（Ctrl＋矢印キー）による明細行でのフォーカスの上下左右移動
２．挿入削除キー（Alt＋Insert、Alt＋Delete）による明細行の挿入削除
３．挿入削除キー（Ctrl＋Insert、Ctrl＋Delete）による明細行データの上下シフト

これらの操作は、割り当てキーを変更したり、無効にすることも可能です。
TEST3 の実行と、VisualStudio.net を一旦終了し、LLL/.net プログラム設計に戻り、TEST3 を再度開いてください。



「情報選択ツリーパネル」->「フォーム制御（F2)」->Form を選択して、Form のプロパティを開くと、「明細の入力キー」というカテゴリーがあり、そこにそれぞれ定義されています。
「明細の入力キー」カテゴリーの上にある「フォームの入力キー」カテゴリーの設定は、明細エリア（LDET）を含む全てのエリア上で有効なキー操作の設定です。

操作を無効にする場合は、割り当てキーを **None** と設定してください。空白ではありません。

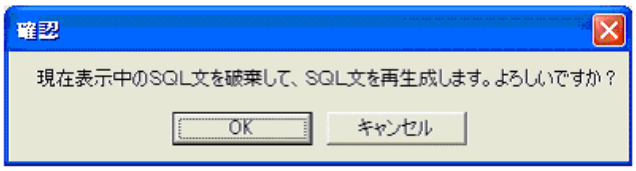
次に、テーブル SQL の編集を試してみましょう。
テーブル SQL は、プログラム内でテーブルを操作するための SQL 文です。
フォームテンプレートで定められた必須テーブル(main、subs)に関しては、予め LLL/.net により自動的に簡易作成されています。

「情報選択ツリーパネル」->「フォーム制御（F2)」->「テーブル SQL」->main を選択し、さらにプロパティ「SQL 文の編集」を選択して表示されるボタンを押して「SQL 文の編集」ダイアログを表示させ、簡易作成済みの SQL 文を見てみましょう。TEST3 の main の場合、下記のような SQL 文が作られています。

```
select *
from [DENHEAD]
where [DENPYO_KBN] = {DENPYO_KBN} and [DENPYO_NO] = {DENPYO_NO}
```

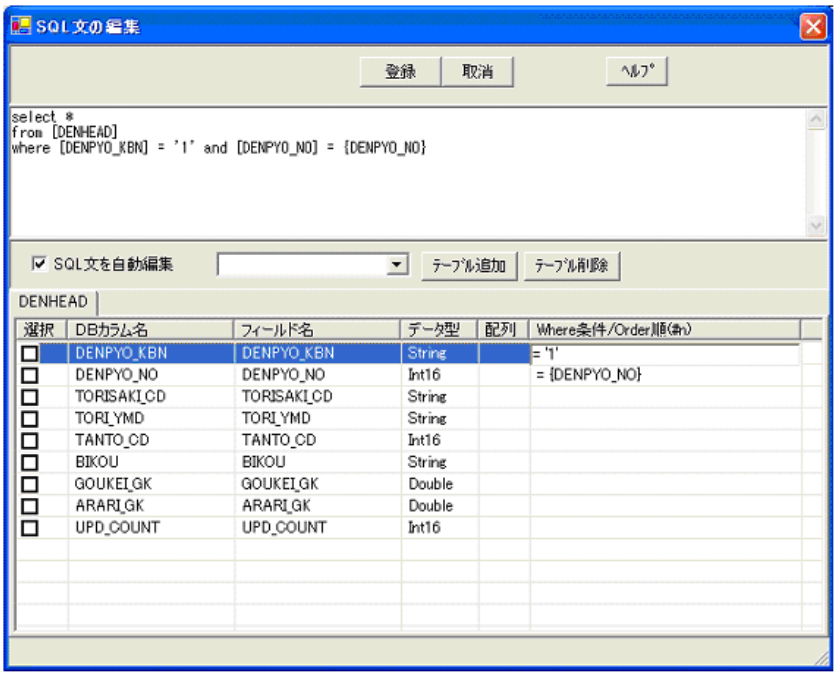
この SQL 文は、画面上の DENPYO_KBN、ならびに DENPYO_NO　というフィールドから入力された値を取得しこれを複合キーとして、DENHEAD テーブルの検索を行い合致したレコードの全カラムを取得することを意味します。

この SQL 文に手を加えるためには、**「SQL 文を自動編集」**というチェックボックスをチェックします。すると



という警告メッセージが表示されますが、[OK]ボタンを押します。
これで編集が可能になりました。

例えば、SELECT 対象となるカラムを全カラムではなく必要なものだけ指定したければ、
ListView に表示されているカラム名の左隣にある「選択」というチェックボックスから必要なものをチェックして選択すると、select 句が自動編集されます。

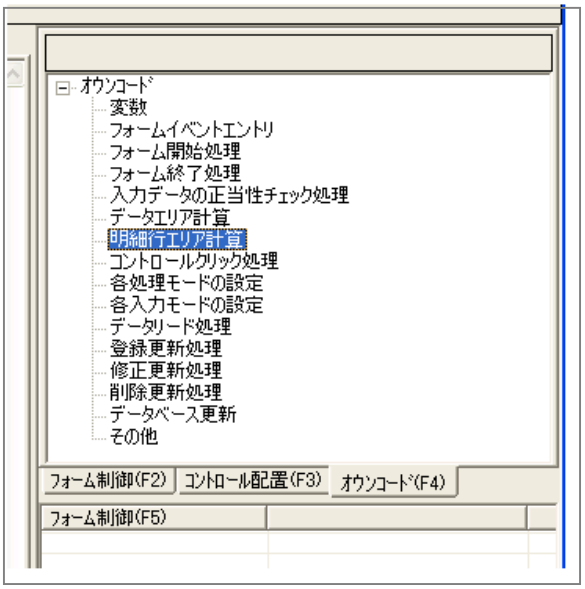


DENHEAD のプライマリーキーは、DENPYO_KBN　、　と DENPYO_NO からなる複合キーですが、これを両方とも画面入力させるのではなく、DENPYO_KBN に関しては固定値で’1’を持たせるという変更であれば、カラムリストの DENPYO_KB の「Where 条件/Order 順(#n)」欄を‘{DENPYO_KBN}’から’1’　に変更します。
これで、DENPYO_KBN　には無条件に’1’が入りますので、画面上から取り込む必要がなくなります。**main** をこのように改造したのであれば、当然 **subs** にも同様の変更が必要です。

SQL 文をこのように変更した場合は、画面上の LKEY エリアから **DENPYO_KBN** を削除してください。さらに**「FORM 制御（F2)」->「FORM」->「LKEY」**配下のツリーアイテム　**DENPYO_KBN**

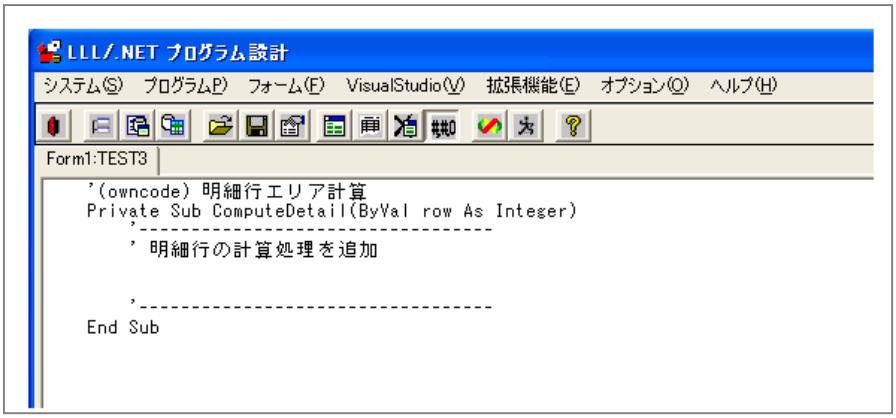
も削除してください。これはフォーム制御オブジェクトの制御対象となる「フィールド」です。フィールドは通常はフォーム上のコントロールに対応しますが、対応するコントロールを持たない work フィールドも存在できます。したがって自動的には削除されません。作成するときは、「コントロール配置（F3)」ツリービューからの一度のドラップで自動的に両方が同時に作成されますが、**削除は別々に行う必要がある**ことに注意してください。

このようにテーブル SQL を修正する場合、カラム名は両端を「[]」でくくります。また、画面上で入力された値を条件に含める場合、そのフィールド名の両端を「{}」でくくるということになっています。



では、次に計算ロジックの追加を行います。
ここで初めて VsiualBasic.net（または C#) によるコーディングが必要になります。

ロジックの追加変更は、オウンコードとして行います。**「情報選択ツリーパネル」->「オウンコード（F4)」**を選択します。ツリービューにオウンコードというアイテムが表示されます。そのメンバーとして、テンプレートが作成した各プロシージャが一覧表示されます。
その中の「明細行エリア計算」を選択してください。



フォームテンプレートが作成済みの「明細行エリア計算」のロジックが表示されます。

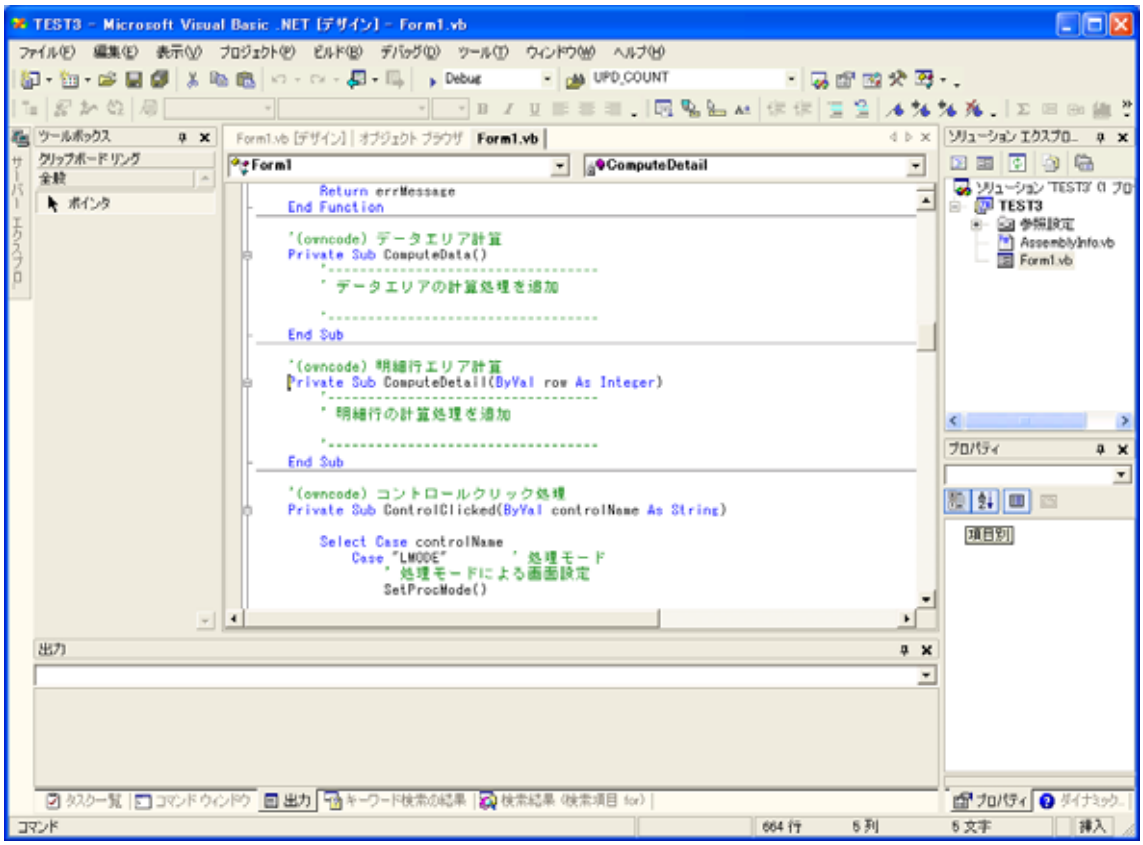
```
'(owncode) 明細行エリア計算
Private Sub ComputeDetail(ByVal row As Integer)
    '--------------------------------
    ' 明細行の計算処理を追加

    '--------------------------------
End Sub
```

このプロシージャは、フォームテンプレートの**フォームイベントエントリー**というプロシージャから呼び出されます。
そこの仕組みや詳細については、後で理解していただくとして、これはデータエリア計算プロシージャとともに、LLL/.net ではエリアイベントと分類されるプロシージャであり、このプロシージャが当該エリア、またはその下位エリアのコントロールのデータに変化があった場合に必ず呼び出されるプロシージャであると認識してください。

つまり、明細行において、金額＝数量×単価というロジックを埋め込みたいとすれば、このプロシージャにコーディングを追加しておけば良いことになります。
デフォルトではデータエリア計算プロシージャには、中身がありません。
どのような処理を必要とするかは、プログラムの目的によって異なるからです。

ちなみに、TEST3 では、例で出した正に、金額＝数量×単価という計算を行いたいわけですから、このコーディングを行います。
今表示されているソースコードに直接手を入れることも可能ですが、LLL/.net がソースを表示している部分は単なるテキストエディターです。スペルのチェックやデバッグを行うことはできません。そこで、実際にコードの追加はそのような機能の充実している VisualStudio.net の IDE 上で行う方が賢明といえます。

ということで、プログラムを保存し、VisualStudio.net を起動してください。
VisualStudio.net で、Form1 のソースを表示させると、先ほど LLL/.net プログラム設計の**「情報選択ツリーパネル」->「オウンコード（F4)」**で確認したのと同じコードが表示されています。
LLL/.net が、LLL/.net で定義された情報を元に、VisualStudio.net のソリューションならびにプロジェクトを生成し、その情報を引き渡して VisualStudio.net を起動しているのですから当然です。



それでは実際にコーディングを追加します。

まず、数量、単価、金額を扱う変数を定義します。

```
Dim tanka As Double
Dim kingk As Double
Dim suryo As Single
```

つぎに、今用意した変数に、画面上から値を取り込みます。明細のどの行から値を取り込むかなどは気にする必要はありません。それは LLL/.net の用意するフォーム制御コンポーネントが殆自動的に処理してくれます。実際のコーディングは下記のようになります。

```
tanka = LForm.GetDataDouble("TANKA", row)
suryo = LForm.GetDataSingle("SURYO", row)
```

これは、tanka と suryo に、今画面上で値が変化した明細行の TANKA 欄、SURYO 欄からデータを取得してそれぞれ代入するという命令を、LLL/.net のフォーム制御コンポーンネント

が提供する　**GetData データ型**　メソッドを使用して書いた例です。

フォーム制御コンポーネントは、**LForm**　という Public 変数に保存されます。従って、処理コードでフォーム制御コンポーネントのメソッドを実行する場合、**LForm.メソッド名**　というコーディングになります。
フォーム制御コンポーネントについては、マニュアル->「**ランタイムコンポーネント」->「ランタイムコンポーネント概要**」、ならびにマニュアル->**「ランタイムコンポーネント」->「フォーム制御コンポーネント」**を御参照ください。

また、これを使ったコーディングという意味では、マニュアル->**「プログラミング」->「プログラミングの概要」**、ならびにマニュアル->**「プログラミング」->「フォーム制御パラメータ情報」**、を御参照ください。
とりあえず、コーディングを続けます。

```
    kingk = tanka * suryo
```

これは、単純に、金額=単価×数量　を VisualBasic.net で書いただけです。
LLL/.net 特有のコーディングではありません。

```
    LForm.PutData("KINGK", row, kingk)
```

これは、先ほどの GetData と対を成すメソッドで、変数の値を画面上の指定したフィールドに表示するメソッドです。

全体を見るとこうなります。

```
'(owncode) 明細行エリア計算
Private Sub ComputeDetail(ByVal row As Integer)
    '--------------------------------
    ' 明細行の計算処理を追加
    Dim tanka As Double
    Dim kingk As Double
    Dim suryo As Single

    tanka = LForm.GetDataDouble("TANKA", row)
    suryo = LForm.GetDataSingle("SURYO", row)    '(owncode) 明細行計算
```

```
        kingk = tanka * suryo
        LForm.PutData("KINGK", row, kingk)

        '--------------------------------
    End Sub
```

これで終わりです。明細行の計算ができるようになりました。
次に、縦計を行って合計金額を表示できるようにします。

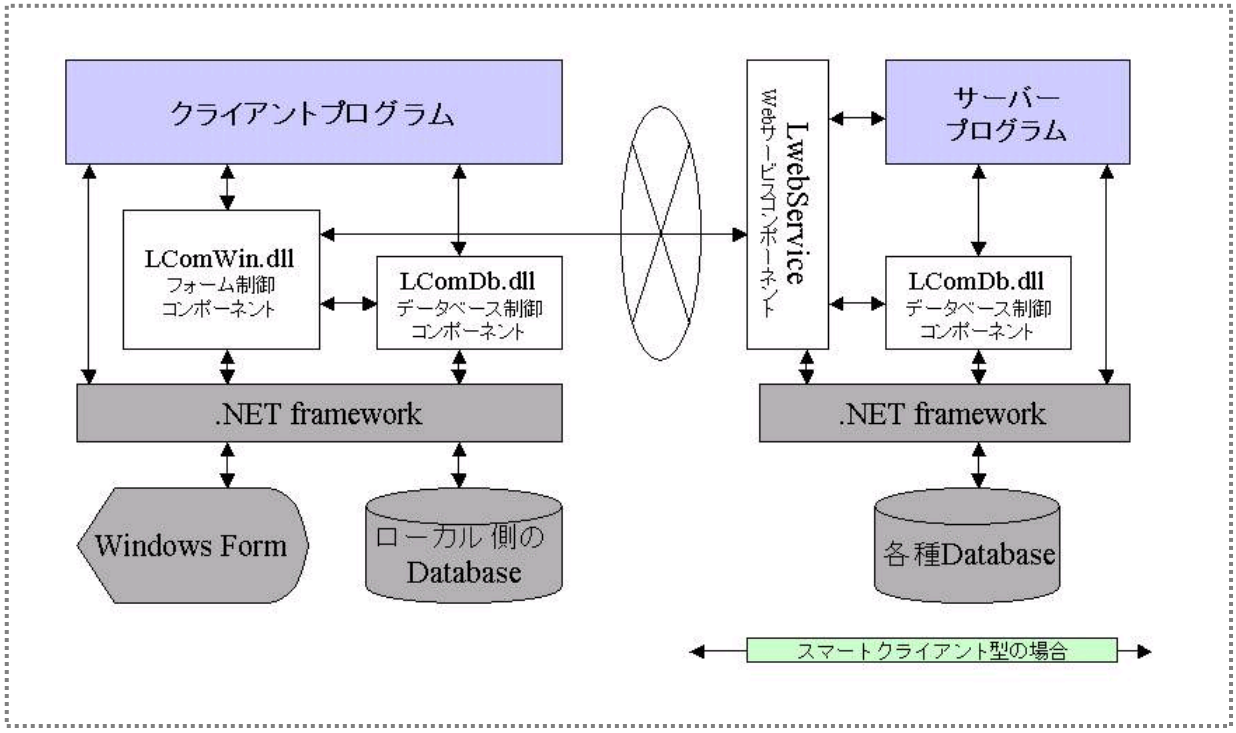
```
    '(owncode) データエリア計算
    Private Sub ComputeData()
        '--------------------------------
        ' データエリアの計算処理を追加
        Dim goukei As Double
        goukei = CType(LForm.RowSum("KINGK"), Double)
        LForm.PutData("GOUKEI_GK", goukei)
        '--------------------------------
    End Sub
```

LForm.RowSum()は、明細フィールドの合計値を算出するLLL/.netフォーム制御コンポーネントのメソッドです。 これで合計の表示も行えるようになりました。

早速、デバッグ->開始、で動作させてみましょう。
これで完成しました。

７．LLL/.netアプリケーションの動作構造

下は、マニュアルの「システムの概要」->「作成したプログラムの概要」->「ランタイムコンポーネント」にある、LLL/.net ランタイム・コンポーネントの全体関連図です



図の左半分だけで構成されるアプリケーションは「スタンドアロン型」です。この構成から「ローカル側の Database」のみを LAN を経由して別のマシン（サーバー）に配置すると「クライアント／サーバー型」になります。さらに右側はインターネットを経由した先に存在する Web サーバーですが、これを含む構成になると「スマートクライアント型」ということになります。

今まで作成してきた「単票形式データ入力」や、「伝票形式データ入力」のプログラムは全て「スタンドアロン型」または「クライアントサーバー型」として動作します。
これらを「スマートクライアント型」として動作させるのは、右側の仕組みを追加してやれば良いだけなので簡単です。
スマートクライアントについては、後述します。

まずは左半分を見てください。
ここに、**LComWin.dll** と、**LComDb.dll** という２つの dll ファイルが存在します。

これが **LLL/.net ランタイム・コンポーネント**です。

LLL/.net ランタイムコンポーネントは、LLL/.net で開発したアプリケーションにとって、.NetFramwork の CLR(**Common Language Runtime)**とともに動作の上で必須となるエンジンで、製品としての LLL/.net の中核をなすものです。
言い換えると、LLL/.net ランタイムコンポーネントの使用を前提とするアプリケーションが LLL/.net アプリケーションであるとも言えます。
製品としての LLL/.net が提供する開発環境（プログラム設計）やテンプレートなどはこのランタイムコンポーネントを利用したコーディング作業を支援する仕組みです。

LComWin.dll　は、**「フォーム制御コンポーネント」**です。
フォーム制御コンポーネントは、フォーム制御情報に基づくフォーム動作の制御をおこないます。

LComDb.dll　は、**「データベース制御コンポーネント」**です。
データベース制御コンポーネントは、データベースアクセスの各種メソッドが用意されています。フォーム制御コンポーネントからのデータベースアクセスに使用されますが、ユーザプログラムからも使用することができるとなっています。

しかし通常は「データベース制御コンポーネント」は、「フォーム制御コンポーネント」が内部から利用するものですので、LLL/.net を利用する開発者の方が必ず意識していただく必要があるのは、「フォーム制御コンポーネント（**LComWin.dll**）」の方だけだということになります。

コンポーネントのさらなる詳細については、マニュアルの「ランタイムコンポーネント」を御参照ください。

8. フォームテンプレートと生成コード

〈フリーフォーマット（PAA100）テンプレート〉

（テストプログラムの作成　その４）

ではここまでの話を踏まえて、さらに LLL/.net の仕組みの理解を深めていただくために、フリーフォーマット（PAA100）テンプレートを使ってテストプログラムも作成し、ソースを解析しながら LLL/.net アプリケーションの動作の仕組みを見て生きたいと思います。

フリーフォーマットは、データ入力や、明細一覧表示などのロジックを持たないテンプレートです。それらは必要であればオウンコーディングとして書き足していただきます。
もちろん単票形式や、伝票形式のデータ入力や、明細一覧表示のテンプレートが自動的に提供してくれる機能と同じものを実装するのであれば、始めからそれらのテンプレートを使用した方が簡単であることは言うまでもありません。

しかし、それらの機能をあえて自作してみていただくことで LLL/.net の仕組みに対する理解を深めることもできます。

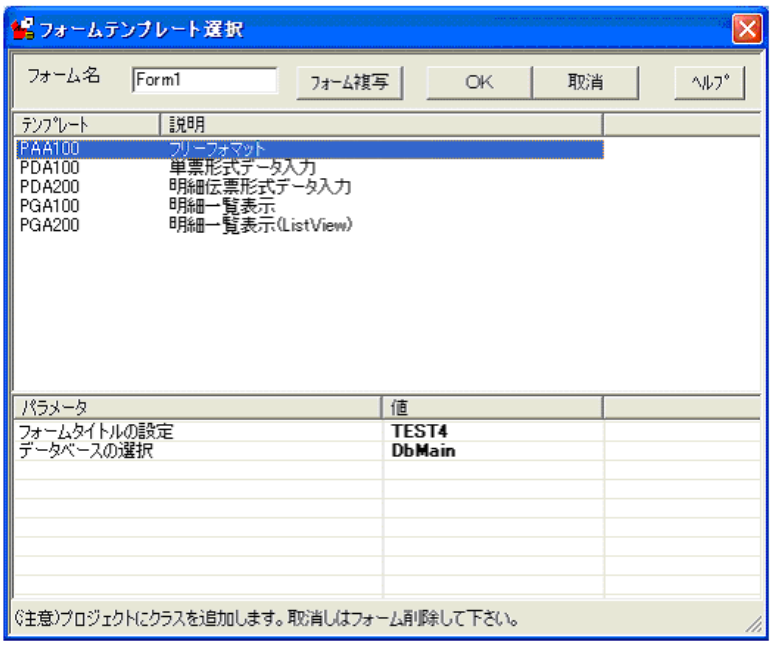
フリーフォーマットのテンプレートが何故存在するかというと、本来は既存のテンプレートが提供するのと全く異なるロジックを実装したり、既存のテンプレートの改造では却って手間がかかるような場合に使用します。
しかし、実はフリーフォーマットこそがテンプレートの基本形であり、LLL/.net が標準提供する他のテンプレートは、このフリーフォーマットテンプレートをカスタマイズすることで派生的に作り出された応用テンプレートのサンプルであるとお考えいただいても良いと思います。
そのため、フリーフォーマットがテンプレート一覧の先頭にあります。

フリーフォーマットテンプレートを使うためには、テンプレートの基本構造とコンポーネントの仕組みを理解していただく必要があります。
逆の言い方をすると、これらが理解できるとテンプレートの改造や自作も可能になり、LLL/.net の活用範囲が飛躍的に拡がります。
そのため研究の題材として、フォームテンプレートの基本形であるフリーフォーマットテンプレートは最適と言えます。

ここから先は、コーディングのかなり詳細に踏み込んでの説明になります。

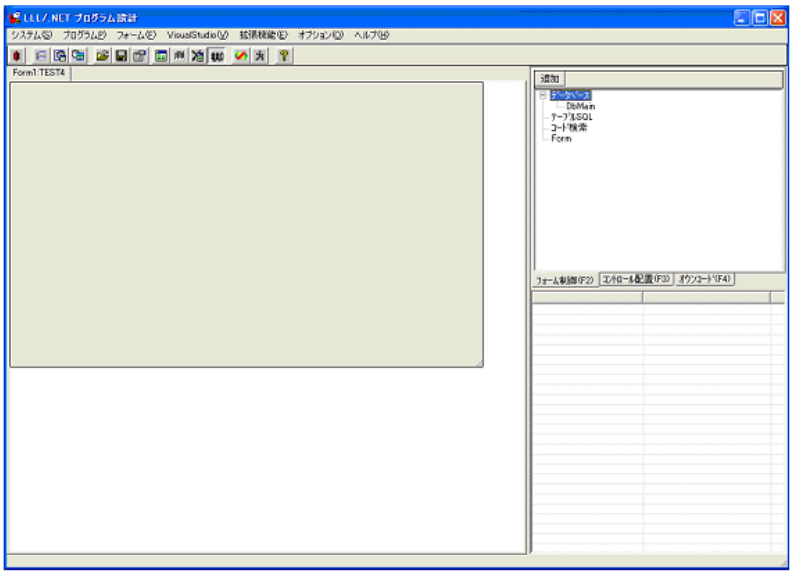
では、再度新規プログラムの作成から行っていきましょう。

今回は、フォームテンプレート選択パネルのリスト先頭（**PAA100 フリーフォーマット**）を選びます。「フォームタイトル」は TEST4 とし、「データベースの選択」には、DbMain を選択しておきます。フリーフォーマットですから、データベースアクセスを前提としないプログラムも作成可能です。ただし、LLL/.net では「データベースの選択」は項目として必須になっていることにご注意ください。例えダミーであってもこれを入力しないと先へ進めません。繰り返しますが、フリーフォーマットには「データ入力」や、「明細一覧表示」のロジックは予め実装されていません。



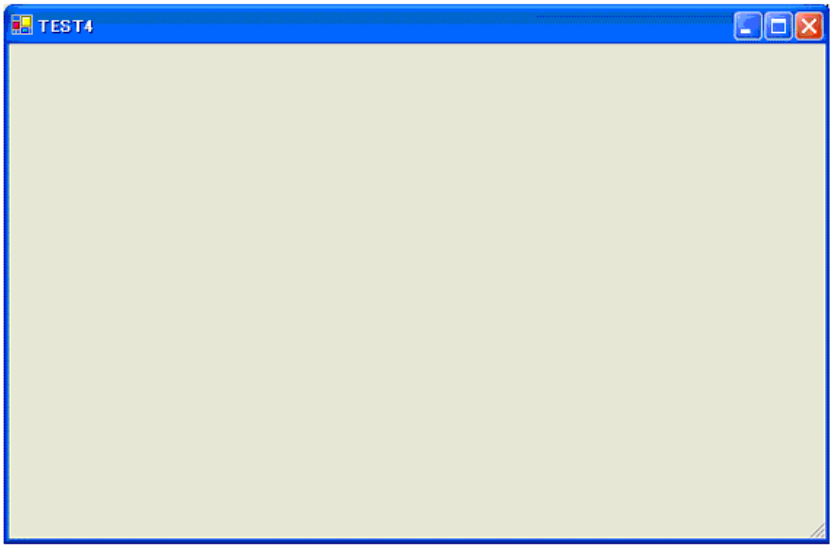
プログラム設計で表示される初期画面（雛形フォーム）も下記のように非常にシンプルなものとなっています。

では、フリーフォーマットテンプレートは、何をどの程度サポートしてくれるのでしょうか。フリーフォーマットであれば、そもそも LLL/.net など使用せず VisualStudio だけで開発しても同じでしょうか。そんなことはありません。



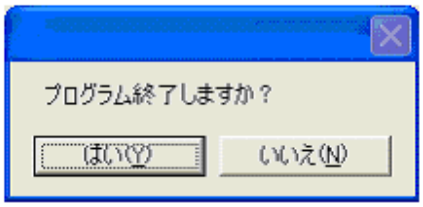
とりあえずコントロール配置も何もせずこのまま VisualStudio.net2003 に情報を引き渡して動かしてみましょう。
例によって、LLL/.net プログラム設計->メニュー->VisualStudio->VisualStudio の起動、で VisualStudio.net を起動し、「デバッグ」->「開始」で起動します。

プログラム開始中を示すダイアログの表示に続き、これまた非常にシンプルではありますが、下のような画面が表示されます。



ボタンもフィールドもありませんので、これ以上の入力や表示はできません。
タイトルバー右上の[×]ボタンを押して終了させましょう。

下のメッセージボックスが表示されます。



これで「はい」を押すとプログラムは終了します。
これでは、VisualStudio.net2003 で Windows アプリケーションのプロジェクトを新規作成し、そのまま動かした場合と比べると、見た目ではプログラム開始と終了のメッセージボックスが出てこない以外は殆ど同じに見えます。
しかし、LLL/.net フリーフォーマットテンプレートで作成したプログラムの方には、実は前述の LLL/.net ランタイムコンポーネントなど LLL/.net が提供する各種機能を利用するための準備が既にいろいろと施されています。

では、ViusalBasic.net を例にしてこのプログラムのソースを確認していきましょう。
VisualStudio.net2003 のデバッグを中止し、ソリューションエクスプローラーのメニューにある**「コードの表示」**ボタンを押し Form1.vb のソースを表示させます。

今回は、フリーフォーマットテンプレートに対し、フィールド追加なども一切行わずデフォルトのまま生成していますので、ソースとしては単純で理解しやすいと思います。

LLL/.net で作成したプログラムはテンプレートの種類によらず、以下のような構成になっています。

```
Imports LLLNET.LComWin
Imports LLLNET.LComDb

Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows フォーム デザイナで生成されたコード "
        フォーム デザイナで生成されたコード
    #End Region

    #Region " LLL/.NET フォーム制御パラメータ情報"
        フォーム制御パラメータ情報
    #End Region

    フォーム処理コード

End Class
```

フォーム処理コードが、アプリケーションロジックであり、それ以前の部分は、処理コードを実行するための前準備部です。
前準備部の内容は、選択したテンプレートや、フォームデザインによって異なりますが、この構成そのものに基本的な違いはありません。

フリーフォーマットの**フォーム処理コード**は、以下の要素で構成されています。

１．変数定義領域：　処理コードで使用する変数を定義します。

２．フォームイベントエントリ
３．フォーム開始・フォーム終了
４．入力データの正当性チェック処理
５．コントロールクリック処理

これらは、LLL/.net プログラム設計の**「情報選択ツリーパネル」->「オウンコード（F4)」**で表示されるプロシージャに対応しています。
フリーフォーマットには存在しませんが、上記に**「エリアイベント処理」**を加えたものをLLL/.net の**共通プロシージャ**と呼びます。
（※参照　マニュアル**「テンプレート」->「プログラミング」->「プログラミング概要」**）

では、具体的に先ほどデフォルト設定で作成したソースの中身を見ていきましょう。
冒頭は、下記の通りです。

```
'--------------------------------
Imports LLLNET.LComWin
Imports LLLNET.LComDb
```

VisualBasic6.0を使ってこられたプログラマーの方には、いきなり見慣れない構文ですが、最初は　Imports　キーワードから始まります。
これは、このプログラムがこの文で指定される名前空間を使用することを意味します。
名前空間は、クラスの所属する空間の名前を表しており、同じ名前空間の中に同名のクラスを持つことはできません。言い方を変えると名前空間はクラスやメソッドを特定するために存在するグループの名前のようなもので、クラスを呼び出す時に、本来はクラス名に先立って記述します。
しかし、こう記述することでこの名前空間に所属するクラスやメソッドを呼び出す時に名前空間の記述を省略することが可能になります。

```
----------------------------
Public Class Form1
    Inherits System.Windows.Forms.Form
```

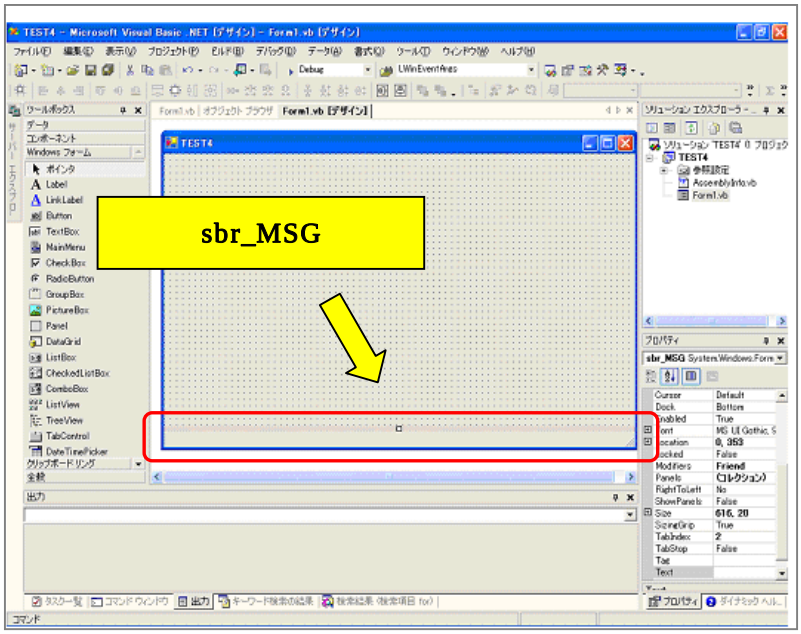
ここからForm1というクラス、すなわち今回の目的の画面（Form）プログラムのコーディングが始まります。LLL/.netが作成するアプリケーションは、.NetFrameworkのWindowsフォームアプリケーションです。

Windowsフォームアプリケーションは、デスクトップに表示されるGUIフォームを持ちます。このForm1というクラスは、.NetFramework CLR の　System.Windows.Forms名前空間に所属するFormというクラスを継承していることを示しています。

Windows フォーム デザイナで生成されたコード

これは、#Region（リージョン）キーワードにより折りたたまれたコードです。
#Regionは、次に出現する#End Region までのコードを折りたたんで格納しています。
左端にある[+]マークをマウスでクリックすると中身が展開されます。展開後　#Region キーワードの左隣の[－]マークをクリックすると再び折りたたまれます。

この中身は、フォームデザイナでデザインされたフォームの定義をコード化したり、Windowsアプリケーションが必要とする基本的な設定を行っています。
LLL/.net開発環境の「プログラム設計」、またはVisualStudio.netのフォームデザイナで設計したビジュアルに関する内容が自動的にコードとして反映される部分ですので、原則的に直接これを手で修正してはいけません。



ちなみにフリーフォーマットテンプレートからデフォルト設定で生成したフォームは、「フォーム制御オブジェクト」がメッセージを出力するためのステータスバー（StatusBar）コントロールが　sbr_MSG　という名前で一つだけフォームの最下段に貼り付けられている、というスタイルになっており、その定義もコードで書かれています。

その後に再び#Regionで折りたたまれたコードがあります。

今回の#Region はLLL/.net開発環境で定義した各種設定（フィールドプロパティ、テーブルSQL、コード検索等々）がコードとして格納されています。

プログラム設計の**「フォーム制御」情報ツリー**に登録した各種情報やプロパティデータが、フォームコードの#Regionで囲まれた部分に生成されます。このコードで、フォーム制御コンポーネントの初期設定が行われます。

この#Region内を編集したい場合は、プログラム設計の**「フォーム制御」情報ツリー**の設定内容を変更してください。この部分も原則的に直接手で修正してはいけません。

その意味で本来はこのまま折りたたんでおいていただければ良いのですが、今回は中身を簡単に紹介しますので、左端にある[+]をクリックして、#Regionを展開してください。

```
#Region " LWinForm フォーム制御パラメータ情報"
    ' 制限 : LWinForm フォーム制御パラメータ情報 は、特別に許可した場合を除いて
    ' プログラム設計を使って変更してください。

    ' フォーム制御オブジェクト
    Public LForm As LWinForm
```

変数　**LForm**　を「フォーム制御コンポーネント（LComWin.dll）」に含まれる　「フォーム制御オブジェクト（**LWinFormクラス**）」型として宣言しています。
実は、LComWin.dllの中には、LWinForm以外にも多くのクラスなどのメンバーが含まれています。ただしそれらは通常はLWinFormが内部から呼び出して使用するものです。
マニュアルに記載しているフォーム制御コンポーネントのメソッドというのも、全てLWinFormクラスの持つメソッドです。
したがってフォーム制御オブジェクト（LWinFormクラス）という言葉が示す機能の範囲は、ほぼ「フォーム制御コンポーネント（LComWin.dll）」と同じものとお考えください。
マニュアルにも「フォーム制御オブジェクト」と「フォーム制御コンポーネント」という言葉が混在して出てきますが、結果的に同じことを指していると解釈してください。

```
    Private Sub LForm_Initialize()
```

```
' フォーム制御オブジェクトの生成
LForm = New LWinForm(Me)
```

フォーム制御オブジェクト（LWinFormクラス）は、このように　LForm　というPublic変数に保存（すなわちインスタンス化）されています。従って、これ以降の処理コードでフォーム制御オブジェクトのメソッドを実行する場合、　**LForm.メソッド名**　というコーディングになります。

まず最初にこの#Region内で、２つのメソッドが実行されています。
LForm.SetDatabase　と　**LForm.SetForm**　です。

```
' データベース
LForm.SetDatabase("DbMain", "")
' テーブル
' コード検索
' フォーム設定
LForm.SetForm("Form", "msg=sbr_MSG|errmsg=sbr_MSG")
LForm.SetForm("/Form")
```

これらは、前述とおり、LForm　すなわち「フォーム制御オブジェクト」に含まれるメソッドですが、マニュアル**「ランタイムコンポーネント」->「フォーム制御コンポーネント」->「フォーム制御メソッド」**にそのリファレンスは存在しません。
LComWin.DLL全体だけではなく、LFormWin のメソッドもユーザーコーディングのために全てが公開されている訳ではありません。この２つはともにLLL/.net開発環境が自動生成するコードの中で使用するメソッドです。
名前から、およそどのような機能かは想像していただけると思いますが、LLL/.netアプリケーションが「フォーム制御コンポーネント」の各種機能を使って動作できるように、LLL/.net開発環境、**「プログラム設計」->「フォーム制御（F2）」**で定義されたフォームやデータベースに関する設定をフォームオブジェクトに対して行っているメソッドです。したがってダイレクトにここに手を加えてはいけません。
ここを変更する時は、**「プログラム設計」->「フォーム制御（F2）」**に設定してください。
今回はフリーフォーマットの初期設定のままの画面ですので、この部分のコードは少ないですが通常は、他に**LForm.SetTable** 、**LForm.SetArea**、や多くの **LForm.SetField**といったメソッドが並ぶことになります。

ちなみに、 **LForm.SetForm("Form", "msg=sbr_MSG|errmsg=sbr_MSG")** に出てくる **sbr_MSG** ですが、前述したフリーフォーマットフォームにデフォルトで貼り付けられているステータスバーコントロールです。実行時に「フォーム制御コンポーネント」が自動的に出力するメッセージの出力先の設定を行っています。

次の記述は重要です。

```
        ' フォームイベントハンドラーの設定
         AddHandler LForm.LWinEvent, AddressOf LForm_Event

    End Sub

#End Region
```

〈**AddHandler「オブジェクト.イベント」, AddressOf「イベント処理プロシージャ」**〉

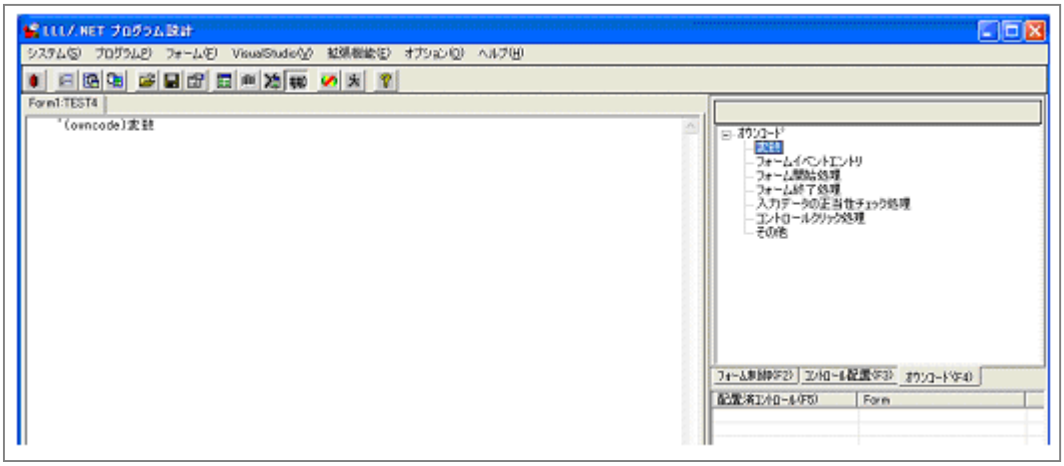
というコードは、**AddHandler** で指定されたオブジェクトイベントの処理を **AddressOf** で指定された処理プロシージャに委譲することを意味しています。
これは、デリゲートと呼ばれる仕組みを用いて実現している機能です。
LForm.LWinEvent の **LWinEvent** は、**LForm** の**メソッド**ではなく**イベント**です。

LFormすなわちフォーム制御オブジェクト（LWinForm）は、**「プログラム設計」->「フォーム制御（F2）」**で設定されたフィールドやエリアに対しForm上で対応するコントロールが発生させたイベント受け、フォーム制御オブジェクトとしての処理を行い、結果を**LLLWinEvent**という独自のイベントとしてアプリケーションに通知します。
通知されたイベントの処理を一手に引き受けるイベント処理の入り口が、この後始まる処理コードである**「フォームイベントエントリ」**です。
そしてこのブロックの最後に #End Region という記述があります。
すなわちここまでが **「LWinForm フォーム制御パラメータ情報」**として折りたたまれていた範囲です。

この下から、いよいよアプリケーションの処理コードに入ります。
通常、LLL/.netを使ってプログラムを作成するということは、ここから下のコーディングを行うことを指します。

```
'(owncode)変数
```

まず最初に上記のコメントがありますが、コーディングの中身はありません。
ここは、このプログラムが全体で使用する変数を定義する場所です。
ここが空白であるということは、フリーフォーマットテンプレートには、最初から用意されたプログラムが全体で使用する変数は無いということです。
これはもちろんテンプレートによって異なります。
'(owncode)　というコメントで始まるコーディングは、LLL/.net 開発環境の「情報選択ツリーパネル」->「オウンコード(Ｆ4)」で確認や修正が行えます。



今回は、空白ですので「情報選択ツリーパネル」->「オウンコード(Ｆ4)」の「変数」アイテムを選択しても、上図の通り左のパネルにコードは表示されていません。
「情報選択ツリーパネル」->「オウンコード(Ｆ4)」に表示されている処理の名前は、ソースコード上で、'(owncode)　で始まるコメントとして書かれたものです。
処理コードに'(owncode)　で始まるコメントを書き足すことで、自由に増やすことができます。
ツリーパネル上の処理の名前をクリックすると、次の'(owncode)　で始まるコメントまでのコーディングを左のパネル上にひとかたまりのものとして表示します。そのパネル上でソースを修正することも可能です。
ただしこのパネルは単なるテキストエディターです。ここでは確認にとどめ、実際のソースの修正は VisualStudio.net2003 を立ち上げて行っていただく方が良いと思います。

```
'(owncode) フォームイベントエントリ
Public Sub LForm_Event(ByVal we As LWinEventArgs)
```

フォーム制御コンポーネントで発生するイベントのイベントエントリとして、LForm_Event プロシージャが使用されます。
前述の　AddHandler LForm.LWinEvent, AddressOf LForm_Event　でそう宣言しているからです。

従って、処理コードの　LForm_Event プロシージャは必須となります。
フリーフォーマットテンプレートのデフォルトでは以下のようなコーディングになっています。

```
    Select Case we.Kind
        Case LEventKind.FormLoad
            ' フォーム開始処理
            FormLoad()

        Case LEventKind.FormClosing
            ' メインフォームの場合、プログラム終了前の確認
            If LForm.IsMainForm Then
                If MessageBox.Show("プログラム終了しますか？", "", _
                        MessageBoxButtons.YesNo) = DialogResult.No Then
                    we.Cancel = True
                    Return
                End If
            End If
            ' フォーム終了処理
            FormUnload()

        Case LEventKind.Validation
            ' 入力データの正当性チェック処理
            we.ErrorMessage = ValidationData(we.Name)

        Case LEventKind.ControlClick
```

```
            ' コントロールクリック処理
            ControlClicked(we.Name)

    End Select

End Sub
```

もう、お分かりいただけると思いますが、ここで　**we.Kind**　に返ってくるイベントの種類に応じて必要なユーザーコーディングを加えていくことでプログラムが作成できます。

フリーフォーマットテンプレートでは、必要最低限度のイベントしかデフォルトでは扱っていません。
実際には　**we.Kind**　に返ってくるイベントすなわち **LEventKind 列挙体のメンバー**には下記のものがあります。
FormLoad、FormClosing、FormClosed、ControlEnter、ControlClick、
ControlLeave 、ControlDoubleClick、ControlTextChanged、
ControlSelectedIndexChanged、 ControlCheckedChanged、AreaEvent、Validation

詳細は、マニュアル「ランタイムコンポーネント」->「フォーム制御コンポーネント」->「フォーム制御イベント」->「フォーム制御イベントの説明」をご参照ください。

以下は、上記のイベントエントリにより、振り分けられたイベント毎の具体的な処理を記述する場所です。

```
'(owncode) フォーム開始処理
Private Sub FormLoad()
    '--------------------------------
    ' フォーム開始処理を追加

    '--------------------------------
End Sub

'(owncode) フォーム終了処理
Private Sub FormUnload()
```

```
        '--------------------------------
        ' フォーム終了処理を追加

        '--------------------------------
    End Sub

    '(owncode) 入力データの正当性チェック処理
    Private Function ValidationData(ByVal fieldName As String) As String
        Dim errMessage As String = ""
        '--------------------------------
        ' データチェック処理を追加。エラーがある場合、エラーメッセージを設定します。

        '--------------------------------
        Return errMessage
    End Function

    '(owncode) コントロールクリック処理
    Private Sub ControlClicked(ByVal controlName As String)
        '--------------------------------
        ' コントロールのクリックイベントが発生したときの処理を追加

        '--------------------------------
    End Sub

    '(owncode)その他

End Class
```

フリーフォーマットのデフォルト状態ですから、実際には具体的な中身はありません。
これに必要な処理を書き加えていくと、プログラムが出来ていきます。

殆どの部分を VB.NET だけを例をとして説明していますが、C#の場合も基本的に考え方やプログラムの構造は同じです。
LComWin.dll や、LComDb.dll といったコンポーネントは、もちろん **VB.NET、C#で共通**です。

ちなみにこれらのコンポーネントのソースはC#で書かれています。

ここまでがご理解いただけましたら、再度今までに作成した単票形式や伝票形式のデーター入力プログラムのソースも見てください。
今まで、ソースを見てみ今ひとつピンと来なかったという人も、プログラム全体の流れを良くお分かりいただけるのではないでしょうか。

では、フォームにコントロールを追加しデータベースのテーブルアイテムの表示や更新が行えるようにしてみましょう。

今までは、テーブルデータの読み込みや更新はかなりの部分をテンプレートの機能に頼っていました。フリーフォーマットではどうなるのでしょうか。
ご心配は不要です。
データ入力や、一覧表示の処理コードのあるテンプレートも、フリーフォーマットのテンプレートも使用しているランタイムコンポーネントは同じです。
前述したとおり、デフォルトでは機能を使用していないだけで、フリーフォーマットであっても他のテンプレートが実現している機能は、基本的に使用可能です。

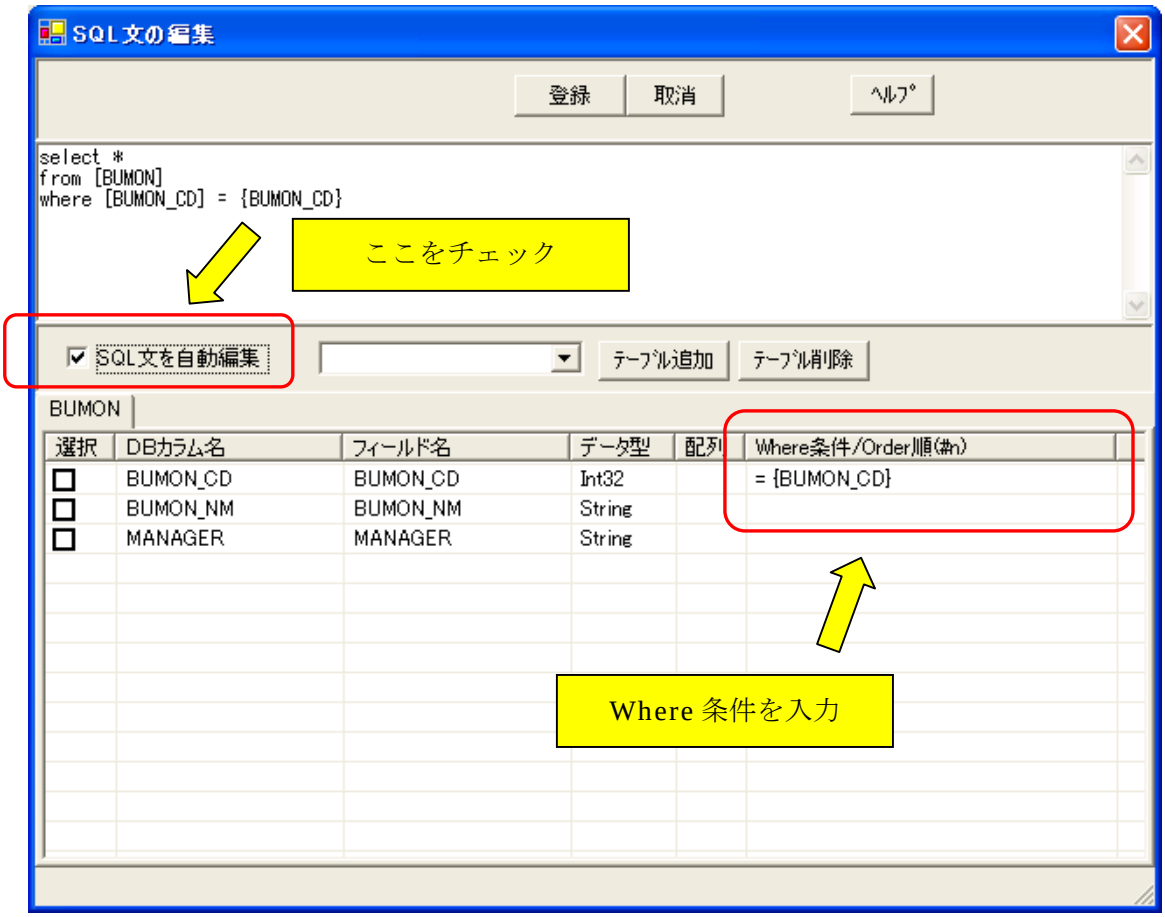
では、最初に単票形式データ入力テンプレートで作成した、部門テーブルのメンテナンスプログラムに相当するプログラムをフリーフォーマットテンプレートから作成してみましょう。

フリーフォーマットテンプレートは、テーブルアクセスを前提としていません。
そのため、単票形式データ入力テンプレートでは、開始時の設定から勝手に作成されていた **main** という「テーブル SQL」 も存在しません。これらは、必要に応じて自作していただくことになります。

では、「プログラム設計」に戻って「テーブルSQL」作ってみましょう。

「情報選択ツリーパネル」->「フォーム制御 (F2)」で**「テーブル SQL」**を選択した状態で、「追加」ボタンを押します。すると DbMain に含まれるテーブル一覧が表示されますので、目的のテーブルBUMONを選択します。
自動的に Tabel1 というサブアイテムが「テーブルSQL」の配下に作成され、プロパティリストにその内容が表示されます。そこでプロパティの上から3つ目にある**「SQL 文の編集」**を選択し、下のパネルを開きます。
最初は、Select * from [BUMON] というSQL文が生成されています。

これに、画面上の BUMON_CD をキーとして合致するレコードを表示するために使用できるように WHERE 句を追加します。

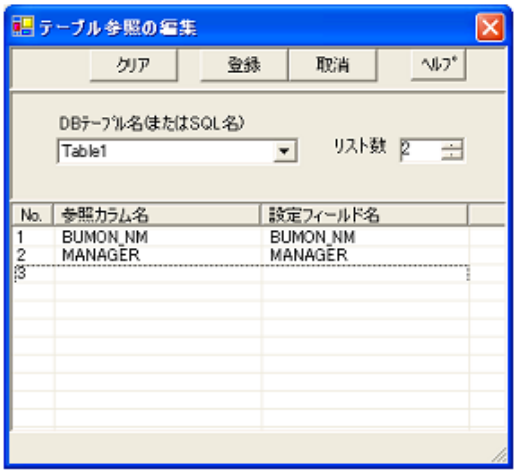


「SQL 文を自動編集」にチェックを付けた上で、下段の「Where 条件/Order 順（#n)」欄に「　=　{BUMON_CD}　」と入力します。上段の SQL 文が自動的に追従して編集されていることを確認して「登録」します。
そしてこの情報を使ってフォーム上にフィールドの配置を行います。

「情報選択ツリーパネル」->「コントロール配置（F3)」のツリーに　Table1　というアイテムが追加されていますので、そこから BUMON_CD(部門ＣＤ)、BUMON_NM(部門名)、MANAGER(マネージャ)を順番に選択してフォーム上にドロップ配置していきます。

単票形式データ入力テンプレートを使用していれば、とりあえずこれでデータの「登録」、「修正」、「削除」、「参照」が行なえるプログラムが動作しましたが、フリーフォーマットテンプレートではそうはいきません。

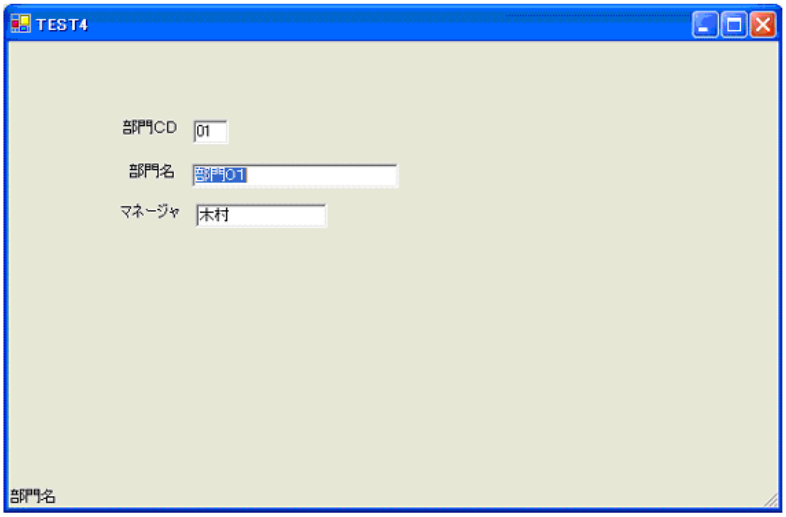
単純に BUMON_CD(部門ＣＤ)が入力されたら、対応する BUMON_NM(部門名)、MANAGER(マネージャ)を画面に表示させるだけでも簡単内容ではありますがそのための設定が必要です。具体的には、BUMON_CD(部門ＣＤ)に対して「テーブル参照」の設定を追加します。



「フォーム制御（F2)」->Form->BUMON_CD のプロパティ**「テーブル参照」**を選択し、「テーブル参照の編集」ダイアログを表示させてください。

そこで、テーブル名（または SQL 名）で「Table1」を選択し、リスト数を２以上に設定して、NO.１　参照カラム名 =BUMON_NM、
設定フィールド名 = BUMON_NM
NO.２　参照カラム名 = MANAGER、
設定フィールド名 = MANAGER、
と設定し「登録」してください。

そして「プログラム設計」メインメニューから「VisualStudio」->「VisualStudio 起動」を選び VisualStudio に制御を渡し、そのまま「デバッグ」->「開始」で実行してみます。



画面が立ち上がりましたら、「部門ＣＤ」に’1’を入力し、「ENTER キー」を押します。
すると、上記のように対応する「部門名」と「マネージャ」が表示されます。
画面上でのデータ編集もできますが、まだデータベースに対する更新などは行えません。一旦、右上隅の[×]ボタンを使って終了してください。

このようにフリーフォーマットでも、LLL/.net のランタイムコンポーネントの機能は使える準備が完了していますので、簡単な設定や処理を加えるだけでプログラムを作っていけ

ることがお分かりいただけたと思います。

では、せっかくですから更新も行ってみたいと思います。更新に関してはもう少し手続きが必要です。今度はフォーム制御オブジェクトのメソッドを使用する必要があります。
ここで、LLL/.net のデータベース制御のプログラミングの仕組みを少しご説明します。

LLL/.net では、データベースアクセスに **ADO.NET** を使用しています。
ADO.NET は、VisualBasic6.0 や以前の C++などでも使用されていた ADO の後継機能ですが、同様の機能を担当するため、名前が同じ ADO となっていますが、中身は完全に新しく作り直された別物と言われています。
ADO.NET はインターネット経由での不特定多数からのアクセスに耐えられる、**「非接続型」**モデルを採用しています。

インターネットのようなサービスでは、当然ながら最大同時接続数に見合うだけのリソースを用意することを前提とする**「接続型」**のモデルは利用し難いと言えます。必要とされるのは、データが読み込みや変更が必要になったときにだけデータベースへの接続を行い、かつ、その接続回数を減らすためにいったん取得したデータを可能な限り使い回すというモデルです。これを「接続型」モデルと対比させて、「非接続型」のモデルと呼びます。

そしてこのモデルでの接続を実現するために.NetFramework が ADO.NET の一部として用意した汎用的なクラス構造が**「データセット」**です。
データセットを構成するクラスは、いずれも System.Data 名前空間に所属し、代表的なものとして、**DataColumn クラス**、**DataRow クラス**、**DataTable クラス**、**DataSet クラス**などがあります。DataTable は、DataColumn と DataRow を、DataSet は DataTable を包含します。

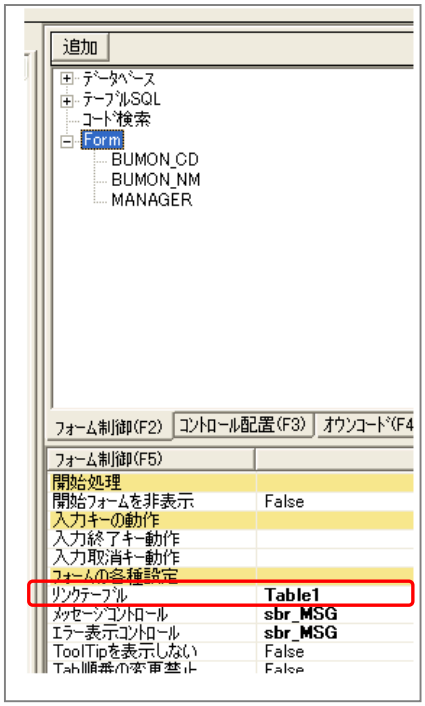
実際のデータベースの接続は「接続型」モデルである NET データ・プロバイダが提供するデータアダプタークラスなどで行います。
データセットは、データアダプタ等により取得されたレコードを格納するための単なる入れ物であり、データセットには実はデータベースに直接アクセスする機能は備わっていないというのがポイントです。

そう簡単では無さそうにも聞こえますが、LLL/.net ではこの仕組みをさらに簡単に使える機能を提供しています。
LLL/.net における ADO.NET を使用したデータベースアクセスは、フォーム制御オブジェクト（LWinForm）に用意されたいくつかの簡単な関連メソッドを使って行ないます。

それは、以下のメソッドです。

１．ReadTable フォーム制御情報に登録したテーブル SQL 情報の SQL 文で、データベースクエリーを実行します。クエリー結果として、DataTable オブジェクトが戻り値になります。
２．ReadSql SQL 文によるデータベースクエリーを実行します。クエリー結果として、DataTable オブジェクトが戻り値になります。
３．GetDataTable 指定された DataTable オブジェクトに、フォームのリンクコントロールのデータを保存します。
４．PutDataTable 指定された DataTable オブジェクトから、フォームのリンクコントロールに、データ表示します。
５．UpdateTable フォーム制御情報のテーブル SQL 情報で取得した DataTable オブジェクトに対する変更内容をデータベース更新します。

これらのさらなる詳細については、マニュアル**「ランタイムコンポーネント」->「フォーム制御コンポーネント」->「フォーム制御メソッド」**をご参照ください。



では、これらを使って修正更新機能をオウンコーディングしてみましょう。

上記メソッドの中で、GetDataTable や、PutDataTable といったフォームのリンクコントロールにアクセスするメソッドを使用する場合、予め LLL/.net プログラム設計で、**リンクテーブル名**を指定しておく必要があります。

他のテンプレートを使用した場合は、予め main 等の**「テーブル SQL」**が指定されています。

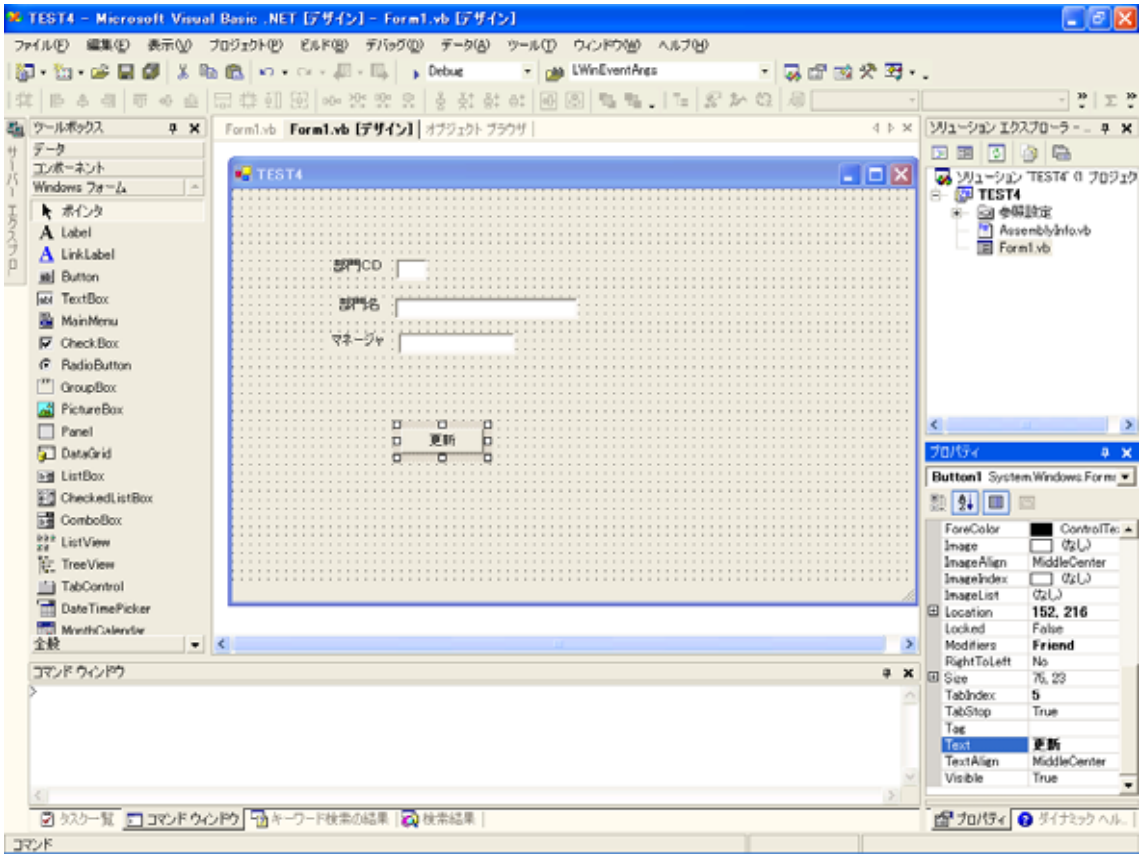
今回はフリーフォーマットを使用していますので、ここは自分で設定しなければなりません。「情報選択ツリーパネル」->「フォーム制御（F2)」の FORM プロパティで「Table1」を設定します。これで準備 OK です。

無意識に更新されてしまっても困りますので、オペ

レータの明確な意思を反映させるため更新コマンド用のボタンを画面に貼り付けます。
本来 LLL/.net ではテンプレートがサポートしていないボタンについても、フィールドとして登録し一括してフォーム制御オブジェクト経由でハンドリングします。
しかし、それは絶対ではなく、独自にコーディングすることも可能です。

今回は、あえてそういうケースとして試してみましょう。
フォーム制御オブジェクト経由での操作を前提としないコントロールについては、LLL/.net の「プログラム設計」で定義する必要もありません。

「プログラム設計」メインメニューから「VisualStudio」->「VisualStudio 起動」を選び VisualStudio を起動します。



ツールボックスの「Windows フォーム」から Button を選択し、フォーム上に配置します。
勝手に Button1 という名前のプッシュボタンが出来上がります。
表示テキストも Button1 では、あんまりなのでプロパティウィンドウで Text プロパティを「更新」に変更しておきます。
そして、このプッシュボタンのクリックイベントハンドラーとして、更新ロジックを実装

します。フォーム上で、更新ボタンをダブルクリックしてください。

```
'(owncode)その他
```

ソースの最下部、「その他」というオウンコーディングコメントのさらに下に VisualBasic6.0 と同じように、空の Button1 のクリックイベントハンドラーが出来上がります。この中に必要なコードを書いていけば良いわけです。

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click

End Sub
```

ところで、生成された**イベント・ハンドラー**ですが、VisualBasic6.0 の時とは少し様子が変わっています。VB6.0 では、下記のようなコードが生成されていました。

```
Private Sub Command1_Click()

End Sub
```

ずいぶんスッキリしています。
これに比べて VisualBasic.NET では、ずいぶんと複雑になりました。
イベント・ハンドラーは、IDE が勝手に作成してくれるので気にしなくても良いといえばそうなのですが、1 点だけ注目しておいてください。

```
Handles Button1.Click
```

という記述です。
VB6.0 ではイベント・ハンドラーの名前は、コントロールとイベントの名前に一致していました。しかし VisualBasic.net ではハンドラーの名前は自由です。このハンドラーがどのコントロールの何のイベントに対応したものかは、上記 Handles キーワードによって決まっています。
このことにより 1 つのイベントから複数のイベント・ハンドラーを呼び出したり、逆に複数のイベントのどれが発生しても、1 つのイベント・ハンドラーを呼び出すようにすることができます。あまり意味がなさそうですが異なるクラスから同じイベントに対する処理を行えるなどオブジェクト指向では有用な機能です。

この機能のおかげで、LLL/.net のフォーム制御オブジェクト（LWinForm）のイベント処理と、独自に直接記述した今回のようなイベント・ハンドラーも共存できます。

現在、部門コードに値を入れると、合致するコードの部門名とマネージャ名が表示されるプログラムというのができています。
また、画面上でデータを修正することもできています。ただしデータベースの更新が行えないということです。

そこで、今回は「更新」というボタンを追加しましたので、このボタンが押されたタイミングで画面上の各フィールドのデータを取り込み、データベースに書き込み更新するという機能を付加します。

最終的にデータベースを更新するのは、前述のフォーム制御オブジェクトの **UpdateTable メソッド**です。その前に画面上のデータを取ってくるのが、**GetDataTable** です。
そしてこれらのメソッドが、更新したり、取ってきたデータを格納しておく操作の対象物が DataTable **オブジェクト**です。

したがって、まず DataTable オブジェクト変数を宣言します。次に、DataTable オブジェクトのインスタンス化を行います。
ReadTable メソッドを用い実際にテーブルデータを読み込むことで、テーブルオブジェクトとテーブル SQL「Table1」を関連付けます。
その上で、**DataTable オブジェクト**に **GetDataTable** メソッドを使用して画面上の現在の値を読み込み、そのデータを使ってデータベースを **UpdateTable で**更新します。
これを実際のコーディングで現すと以下のようになります。

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click

    Dim DtMain As DataTable
    DtMain = LForm.ReadTable("Table1")
    LForm.GetDataTable(DtMain)
    LForm.UpdateTable(DtMain)

End Sub
```

このコーディングを VisualStudio 上で行い、「デバッグ」->「開始」で動かしてください。

部門 CD に　‘１’と入力して ENTER キーを押すと、部門名とマネージャー名が表示されます。マネージャー名を修正し、「更新」ボタンを押してください。
画面上に変化はありませんが、データベースは更新されています。
一度違う部署を呼び出してから、再度部門コードに‘1’を入力してください。先ほど修正した通りのデータが表示されます。
一見うまく動作したかのように見えますが、実は明らかな問題点があります。

最初の表示は「テーブル参照」の設定で行いましたが、更新のためには異なる準備が必要でした。今作成した更新機能は、後付けで作成したものです。

言い方を変えますと、このプログラムでは最初に部門コードを入力して、部門名とマネージャー名を表示したロジックと、「更新」ボタンを押してその時に表示されていた部門データを修正更新するロジックは、全く別物として動作しています。
このことは、1 台のクライアントしか存在しないスタンドアロンシステム（すなわち処理は暗黙の内にシリアライズされる）であれば、特に問題ではありません。しかし複数台のクライアントを前提とする C/S 型やスマートクライアント型アプリケーションにおいては問題です。

前述したように LLL/net が採用している ADO.NET データベースアクセスは、「非接続型」モデルです。「非接続型」モデルにおいて参照と、更新を無関係に行うと、オペレータがデータの読み込み後、表示されているデータが、現在の最新データであるという保障が得られません。表示後に他のユーザー（クライアント）によって既に修正が行われているかもしれないからです。
非接続型モデルではデータの読み込み時にテーブルやレコードのロックは行いませんが、更新タイミングで読み込み済みデータと、現在のデータに相違がないかを確認してから更新するという、擬似的な排他制御アクセスを行う必要があります。
ADO.net にこれを行ってもらうためには、読み込みと、更新を同じテーブルオブジェクトに対して行う必要があります。

今回のプログラムでは、「更新」ボタンが押された段階で、

```
DtMain = LForm.ReadTable("Table1")
```

テーブルからその時点のレコードを読み、すなわちテーブルオブジェクトを取得しています。そしてその後

```
        LForm.GetDataTable(DtMain)
        LForm.UpdateTable(DtMain)
```

によって、画面データの取得と、そのデータを使ったテーブルオブジェクトの更新を行っています。
しかしこのテーブルオブジェクトは、最初に部門コードによるテーブルデータの読み込みを行った時に使用したテーブルオブジェクトではありません。
ここに問題があります。

部門コード入力による、部門名とマネージャー名の表示は、LLL/.net プログラム設計「フォーム制御（F2)」の「FORM」->「BUMON_CD」->「テーブル参照」に対する設定で指示しました。ここでの設定内容は、自動生成されたソースの

```
#Region " LWinForm フォーム制御パラメータ情報"
    Private Sub LForm_Initialize()
```

内の‘フォーム設定’において、下記のように記述されたことにより実現されています。

```
    LForm.SetField("BUMON_CD", "caption=部門ＣＤ|data=Int32|format=00
    |map=Table1.BUMON_CD|ref=Table1,BUMON_NM/BUMON_NM,MANAGER/MANAGER")
```

ここでの記述は、実行時に「フォーム制御オブジェクト（LWinForm)」内で、勝手にテーブルオブジェクトを作って処理されます。ここで使用されたテーブルオブジェクトをユーザーアプリケーションが取得することはできません。

つまり、LLL/.net プログラム設計**「フォーム制御（F2)」**の**「FORM」->「BUMON_CD」->「テーブル参照」**に対する設定とは、当初から「参照」だけを目的とした機能であり、その結果がそのまま更新されることは想定されていないのです。

今回の処理目的を達成するためには、プログラム設計の「テーブル参照」を使用するのではなく、このタイミングから自力でテーブルオブジェクトを取得し、読み込んだレコードを画面に表示し、さらにそのテーブルオブジェクトを使用してボタンが押されたらこの時点の画面データを取り込んでデータベースを書き換えるという処理をオウンコードとして記述する必要があります。

ということで、再度プログラムを改造して問題点をつぶします。
部門コードが入力されたら、画面上からそのコードを取得してデータベースにアクセスします。該当するレコードのデータを取得し、画面上に表示します。
これは以下のように記述します。

```
'(owncode) 入力データの正当性チェック処理
Private Function ValidationData(ByVal fieldName As String) As String
    Dim errMessage As String = ""
    '---------------------------------
    ' データチェック処理を追加。エラーがある場合、エラーメッセージを設定します。

    Select Case fieldName
        Case "BUMON_CD"

            DtMain = LForm.ReadTable("Table1")
            LForm.PutDataTable(DtMain)

    End Select
    '---------------------------------
    Return errMessage
End Function
```

つぎに、更新というボタンが押されたら画面上のデータを取り込みます。
これは、先ほどはあえて

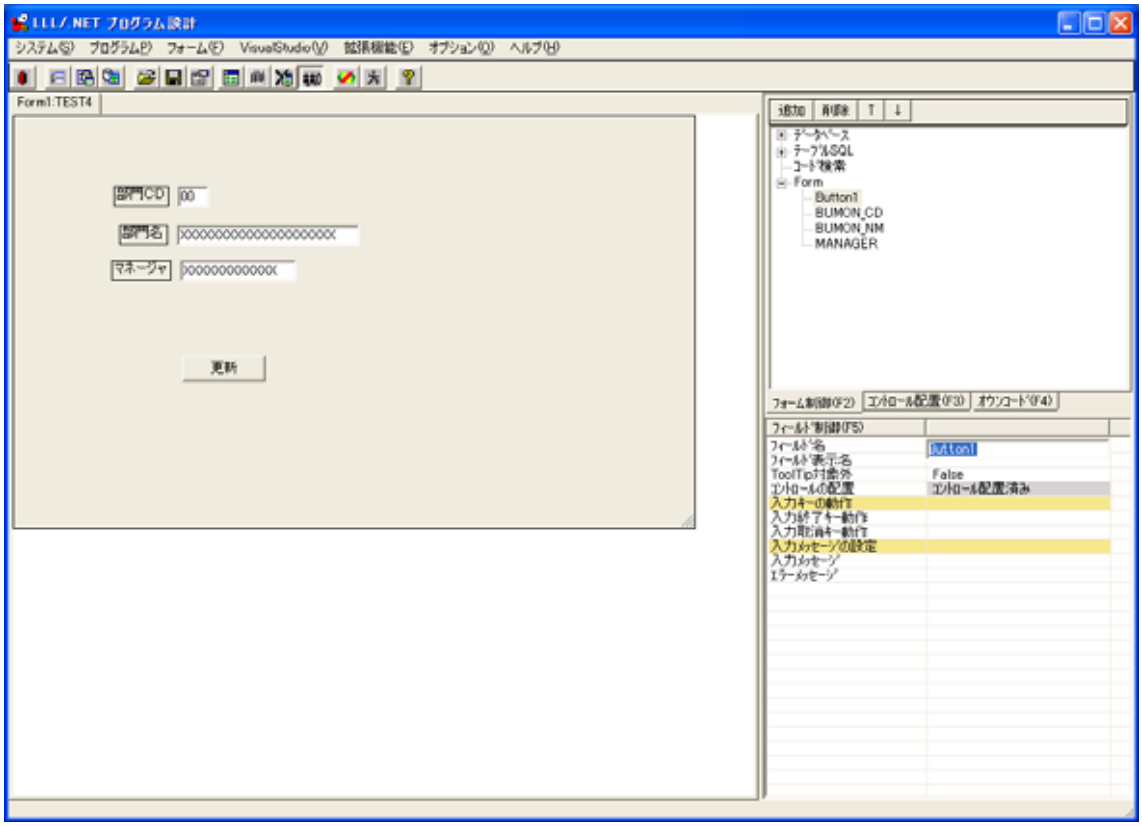
```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
```

に記述していましたが、今回は LLL/.net アプリケーションらしく LLL/.net の**フォームイベントエントリー**経由で動作するように記述し直します。

そのためには、まず LLL/.net に Button1　をフォーム制御オブジェクトが管理する**「フィールド」**として認識させる必要があります。一旦処理を LLL/.net「プログラム設計」に戻してください。

そして、「フォーム制御（F2)」で情報選択ツリービューの FORM を選択した状態で、**「追加」->「フィールド」**を行います。そして追加したフィールドの名前を **Button1**　に変更します。これで、Button1 は、フォーム制御オブジェクト（LWinForm）の制御対象フィールドになりました。

「フォーム制御（F2)」の**「フィールド」**には、こういう重要な意味があります。



これで準備 OK です。再度 VisualStudio.net2003 に戻ります。
そして以下のコーディングを行ってください。

```
'(owncode) コントロールクリック処理
 Private Sub ControlClicked(ByVal controlName As String)
    '---------------------------------
    ' コントロールのクリックイベントが発生したときの処理を追加

    Select Case controlName
        Case "Button1"
```

```
            LForm.GetDataTable(DtMain)
            LForm.UpdateTable(DtMain)

        End Select

        '--------------------------------
    End Sub
```

テーブルオブジェクトDtMainを使用するプロシージャが２箇所に分散しましたので、
Dtmainの宣言は、処理コード冒頭の　　'(owncode)変数　領域に移動します。

```
    '(owncode)変数

    Dim DtMain As DataTable
```

プログラム末尾の

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles Button1.Click
```

は、不要になりましたので、中身ごと全部削除してください。

これで完成です。[デバッグ]->「開始」で動作を確認してください。ビルドして EXE を作成すれば擬似排他制御が試せます。
EXE を２つ起動し、それぞれの画面から同じ部門を呼び出し、修正を行います。
テーブルからの Read 後に、もうひとつの EXE から同じ部門データを呼び、先に修正してしまうと、先に Read していた方の修正ができなくなります。

ここまででは、修正だけが行えるプログラムです。
マスターメンテナンスプログラムとしては、さらに新規登録や、削除が行える機能が必要になりますが、その実装方法はおよそ想像していただけるものと思います。

いかがでしょうか。今回はフリーフォーマットテンプレートを使って、簡単な入力プログラムを作成してみました。

このフリーフォーマットテンプレートには、他のテンプレートにあるような「エリア」という概念は、デフォルトでは前提としていません。つまり、「エリア」は LLL/.net にとって必須の概念ではありません。もちろん必要であれば追加できます。
「明細エリア」に関してはさらに有用で特殊な機能を持ちますが、その他のエリアは主にフォーム上のフィールドグループに対する汎用的なフォーカス制御に使用しています。

エリアは、テンプレートの汎用化を進める、道具のひとつです。
また、標準提供の入力系のテンプレートには、最初から配置されている「登録」、「修正」、「削除」、「参照」、「終了」といったモード選択ボタンや、「リード」、「OK」、「取消」などのボタンも最初は存在していません。つまりこれらも、LLL/.net としては必須のものではないということです。テンプレートのフォームの雛形に最初から用意されているこれらのコントロールは、そのテンプレートが必須としているコントロールであるというだけです。

どういうコントロールにどういう機能を持たせるか、ある程度決まった処理の流れを持たせるのか、高いインタラクティブ性を求めるのか、またどの程度テンプレートの汎用化を進めるかはテンプレート作成者の自由です。

本章の冒頭で、実はフリーフォーマットこそがテンプレートの基本形であり、LLL/.net が標準提供する他のテンプレートは、このフリーフォーマットテンプレートをカスタマイズすることで派生的に作り出された応用テンプレートのサンプルとも言えると申し上げました。その意味がお分かりいただけましたでしょうか。
フリーフォーマットには無く、他のテンプレートに存在するプロシージャの意味や、役割もソースを見ていただければご理解いただけるものと思います。

最も汎用性が高いテンプレートは、フリーフォーマットです。
しかし、データ入力や一覧表示を目的とするプログラムであれば、それら用のテンプレートを使用した方が開発はさらに早くなります。それと引き換えに汎用性は損なわれます。
テンプレートが標準で提供する操作手順や、機能に不満足であればそのテンプレートを改造するか、フリーフォーマットをベースに新たにテンプレートを作成してください。

実際のテンプレートの作成方法については、マニュアル　**「テンプレート」->「フォームテンプレートの概要」**ならびに**「テンプレート」->「フォームテンプレートの作成」**をご参照ください。

９．複雑な検索用ウインドウの作成
〈多段階コード検索〉

TEST2、TEST3 では、フォーム制御オブジェクト（LWinForm）が提供する簡単に設定できる「コード検索」機能を紹介して使用しました。

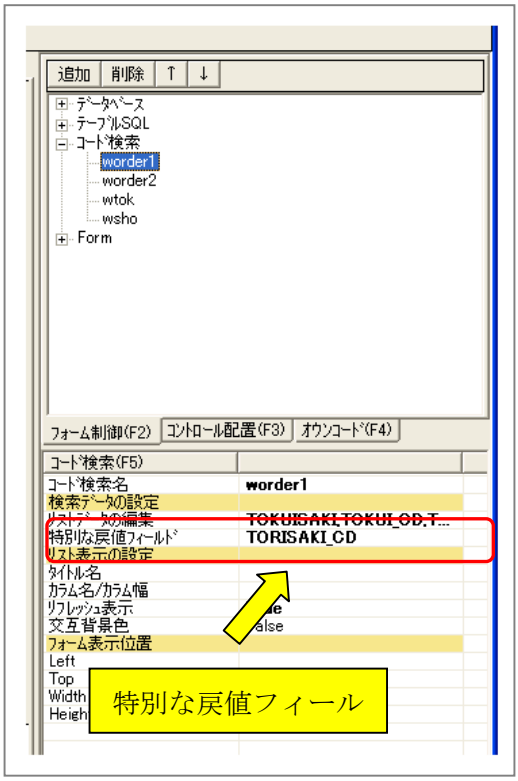
コード検索ウインドウは、表示対象の存在するテーブル名（参照テーブル）と、表示項目（カラム）を対話型で設定するだけで使えますので大変便利な機能です。
しかし、対象データが多い場合このような全項目一覧表示型の検索ウインドウでは、表示のためのデータ読み込みに時間やネットワークの負荷が大きくなる上、スクロール量が膨大になり、目的データの探索時間がかかるなど、対応しきれない場合もあります。

このような場合に有効なのがデータの絞り込みです。
これを実現する方法はいくつか考えられますが、簡単なのは LLL/.net の標準「コード検索」を複数組み合わせる多段階検索です。サンプルに標準添付されている売上伝票入力 HAD010 プログラムで伝票番号の検索ウインドウとして使用しています。実物を見てみましょう。

LLL/.net プログラム設計で、**「プログラム」->「プログラム選択」**で**[HAD010 売上伝票入力]**を選択して設計情報を表示してください。
このプログラムではコード検索を４つ使用しています。**「フォーム制御（F2)」->「コード検索」**を見てください。**worder1**、**worder2**、**wtok**、**wsho** の４つが確認できます。
このうち、wtok、wsho の２つは、それぞれ得意先と、商品のコード検索で、TEST3 で作成したコード検索と同じです。
worder1、と **worder2** が伝票番号検索に使用している 2 段階検索用の「コード検索」です。全伝票を一覧表示をすると前述とおり、件数が膨大で大量のスクロールを強いられる可能性があります。そこでこのサンプルでは一旦得意先コードで絞込みを行っています。
その為の「コード検索」が worder1 です。そして得意先で絞られたデータの中からさらに最終目的の伝票を探し出すのが worder2 です。サンプルで動作を確認してください。

LLL/.net プログラム設計で **worder1** の定義を見てください。実は得意先のコード検索である wtok と殆ど同じ定義内容です。それも当然で実は得意先のコード検索だからです。
Wtok と異なるのは、コード検索定義のプロパティに worder1 の方にだけ**「特別な戻値フィールド」**が設定されていることです。

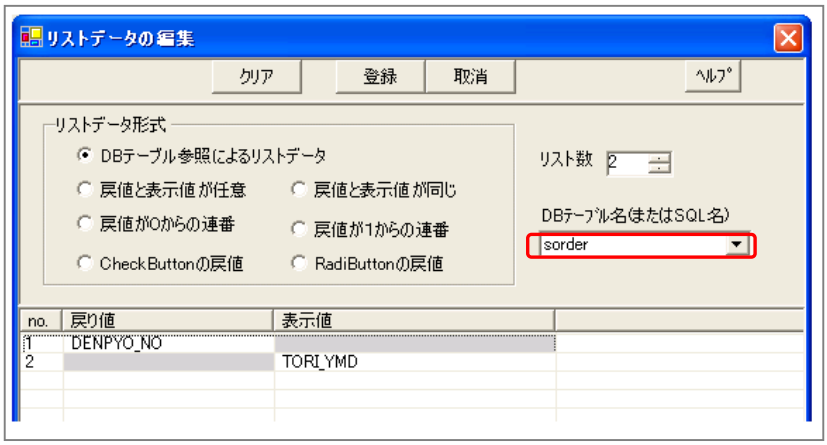
コード検索では、「特別な戻値フィールド」に設定がなければ、選択結果を呼び出し元のコントロール（フィールド）へ返します。



しかし「特別な戻値フィールド」に設定、すなわちフィールドが指定されていた場合は、そのフィールドに値を返します。この例では呼出し元は伝票番号（DENPYO_NO）ですが、戻り値は取引先ＣＤ（TORISAKI_CD）である。ということになります。さらにこの取引先ＣＤを使って伝票番号の絞込みを worder2 が行います。

このサンプルではもうひとつ特殊な処理を行っています。伝票番号の絞込みというのはテーブル DENHEAD　のコード検索を作れば良いということなのですが、TEST3 のテーブル SQL、main、subs の設定で御説明したように DENHEAD のプライマリーキーは DENPYO_KBN と DENPYO_NO の複合キーとして構成されています。複合キーを「コード検索」では一度に指定できません。そこで登場するのが、**「テーブル SQL」**です。

worder2 の「リストデータの編集」では、「DB テーブル名（または SQL 名）」に対し、DENHEAD ではなく、sorder という名前が設定されています。



これが**テーブル SQL** で、そこに定義されている SQL 文は下記の通りです。

```
select [DENPYO_NO], [TORI_YMD]
from [DENHEAD]
where [DENPYO_KBN] = '1' and [TORISAKI_CD] = '{TORISAKI_CD}'
```

この WHERE 句の'{TORISAKI_CD}'に、worder1 で選択した TORISAKI_CD が入ります。
コード検索の詳細情報は、マニュアル**「システムの操作」->「プログラム設計」->「プロパティリスト」->「コード検索情報」**をご参照ください。

〈複数フォーム使用のプログラム〉

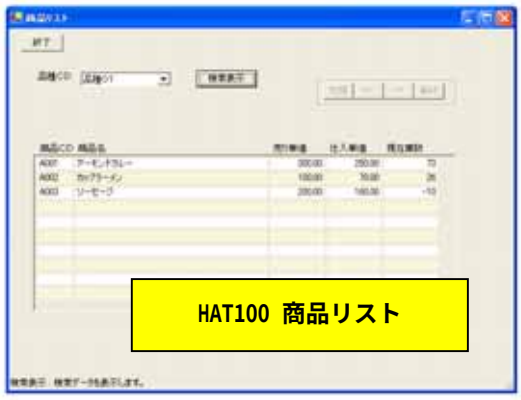
（テストプログラムの作成　その５）

このように LLL/.net 提供の機能でも組み合わせを使うことで、複雑なコード検索が可能ですが、コード検索画面そのものにもっと機能を持たせたいとか、フォームのデザインそのものをもっと自由に変更したいなどという場合には、どうすれば良いのでしょうか。
そのような場合は、さすがにこのような簡単な対話型設定だけでは実現できません。
コード検索のフォームを呼び出し元である親フォームとは別に設計する必要があります。

「別に設計」といっても一切 LLL/.net を使用せずに作成しようというのではありません。そのフォームを独立したフォーム、すなわち別のテンプレートで作成し連携させます。これに使える良いサンプルプログラムが LLL/.net に付属しています。

今回は、前述した５つの標準フォームテンプレートの中から、まだ説明していない、
４．PGA100 明細一覧表示、５．PGA200 明細一覧表示(ListView)、の２つのテンプレートのどちらかを使用して作成したプログラムを利用する方法をご説明します。

「４．PGA100 明細一覧表示」と**「５．PGA200 明細一覧表示(ListView)」**は、何が違うかというと、名前の上では　(ListView)　が付いているかいないかの違いだけですが、実際に作ったプログラムの画面を見ていただければ違いは一目瞭然です。



サンプルの HAT100 商品リスト　が「４．PGA100 明細一覧表示」の例、 同じく HAT110 商品リスト（ListView）が「５．PGA200 明細一覧表示(ListView)」の例です。
この２つのサンプルプログラムは、どちらも商品を一覧表として画面に表示するものです。
HAT100 商品リスト では､一覧の表示項目を複数の Label コントロールで作成しています。
HAT110 商品リスト（ListView）では、それらをまとめて１つの ListView コントロールで

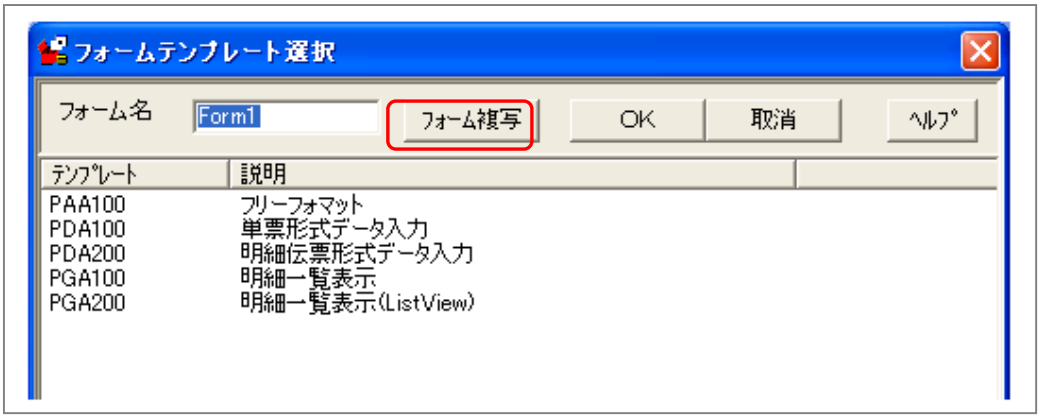
表示させています。ただそれだけの違いで機能的に大きくは変わりません。
表現可能な情報制限はありますが、デザイン的にはListViewの方がスッキリ見えるのではないでしょうか。今回はHAT110 商品リスト（ListView）を採用したいと思います。

商品リストは、品種コードをコンボボックスのドロップダウンリストから選択し、絞り込んでから表示させています。このプログラムを検索ウインドウとして使えれば、先にご紹介したように絞込みに「コード検索」を多重使用する必要がなくなります。「コード検索」の多重使用が悪いのではありませんがこういう選択もできればさらに自由度が高まります。

ということで、このプログラム（HAT110 商品リスト（ListView））を、TEST3 で作成した売上伝票入力プログラムと同じようなプログラムの商品検索用ウインドウとして使用する方法についてご説明します。

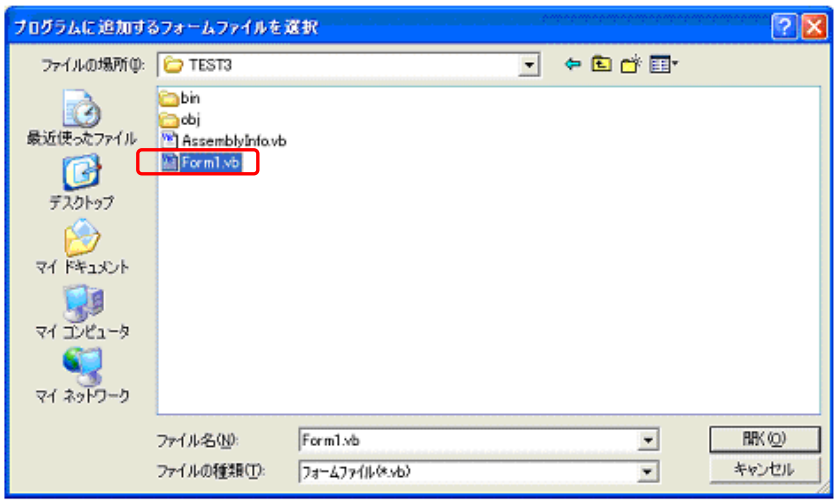
では、まず親画面となる伝票入力プログラムを作成しますが、「TEST3で作成した売上伝票入力プログラムと同じようなプログラム」であればもう一度作るのは面倒ですね。
このような場合、LLL/.net では資産の使い回しが簡単に行えますので、この機能もここで紹介しておきます。

「プログラム選択」->「新規プログラム」で名前を **TEST5** と入力してください。
ここまでは今までと同じですが次が違います。フォームテンプレートの選択ではテンプレートを選ばず、その代わりに**「フォーム複写」**ボタンを押してください。



すると、ファイル選択ダイアログが出現し、現在の開発ディレクトリが表示されます。
その中には、最初から添付されているサンプルプログラムと、今までに作成した TEST1 から TEST4 までのプログラム名のフォルダーがあります。

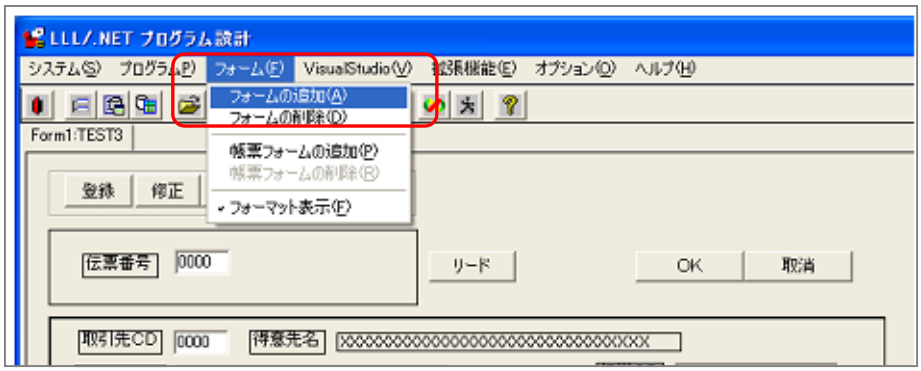
今回は、「TEST3 のような」プログラムを作りたいわけですから、TEST3 のフォルダーをダブルクリックしてその中の Form1.vb を選択してください。



これで、フォームすなわち、TEST3 プログラムの複製が TEST5 という名前でできたことになります。

そしてこの TEST5 プログラムに対して、絞込み機能付きコード検索ウインドウとして使用する商品リストプログラムを追加します。

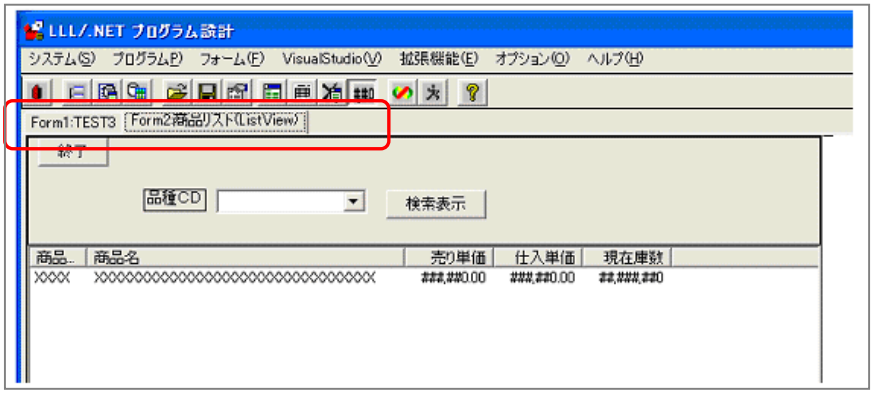
そのやり方は次の通りです。まず「プログラム設計」メニューから、「フォーム」->「フォームの追加」を選びます。



すると再びテンプレートの選択画面が出ます。

そこで先ほど TEST3 プログラムを複写したのと同じ方法で、商品リストプログラムのフォームを複写して追加します。

前述のとおり、サンプルの商品リストプログラムには 2 種類ありました。どちらでも良いのですが、今回はスッキリしたスタイルの HAT110 商品リスト（ListView）を採用したいと思いますので、そちらの Form1.vb を複写してください。



フォームパネルタブが追加され、「Form1:TEST3」の右横に「Form2:商品リスト(ListView)」が出来上がりました。

これで TEST5 プログラムは、Form1 と Form2 という２つのフォームを持つプログラムになりました。

しかし、商品リスト（ListView）は、もともと HAT110 として独立して動作することが前提のプログラムであり、単純に複写して追加されたからといってこれだけで TEST5 の商品検索ウインドウとして使用できるわけではありません。
Form1 からは、Form2 を呼び出す仕組みと Form2 から返される選択結果を受け取る仕組み、また Form2 では選択された結果を Form1 に返す仕組みを組み込む必要があります。

まずは、Form2 側にその機能を組み込みたいと思います。
では、VisualStudio.net へ制御を移します。
ソリューションエクスプローラーで Form2.vb を選択し、ソースを表示してください。
そしてソースコードの最下部　　**'(owncode)その他**　のさらに下の部分をみてください。

```
'-----------------------------------------------------------------
' このプログラムを、コード検索に使用する場合に追加するサンプルロジックです。
'-----------------------------------------------------------------
```

というコメントがあり、さらにコメント化されたコードが続いています。
この部分の各行頭の ’ を１つづつ消してコメントをコード化してください。
下記のようになります。

```
'(owncode)その他


'-----------------------------------------------------------------
' このプログラムを、コード検索に使用する場合に追加するサンプルロジックです。
```

```
    '-----------------------------------------------------------------
    '(owncode) 明細エリアの選択処理
    Private Sub ListData_DoubleClick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ListData.DoubleClick
        ListSelected()
    End Sub

    Private Sub ListData_KeyPress(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles ListData.KeyPress
        If e.KeyChar = Microsoft.VisualBasic.ChrW(13) Then   ' Enter
        ListSelected()
            ElseIf e.KeyChar = Microsoft.VisualBasic.ChrW(27) Then  ' Esc
        LForm.Focus("LKEY")
            End If
    End Sub

    Private Sub ListSelected()
        ' 選択アイテムがない場合、Exit
        If Me.ListData.SelectedItems.Count <= 0 Then Exit Sub
        ' 選択アイテム取得
        Dim selectItem As ListViewItem = Me.ListData.SelectedItems(0)

        ' 選択アイテムのデータを戻すロジックを追加してください。

        If Not LForm.IsMainForm Then
            Me.DialogResult = DialogResult.OK
            Me.Close()
        End If
    End Sub
    '-----------------------------------------------------------------

End Class
```

以下の３つのプロシージャが追加されたことになります。

１．ListData_DoubleClick

２．ListData_KeyPress

３．ListSelected

この HAT110.VB（HAT100.VB も同様）は、当初よりこれを検索ウインドウとして使用する場合を想定し、このように必要なコードサンプルを含んでいます。
このコードは、PGA100 明細一覧表示、や　PGA200 明細一覧表示(ListView)　のテンプレートが持っているのではなく、あくまでサンプルプログラムHAT110.VB、HAT100.VB　のサンプルオウンコードです。

追加されたプロシージャのソースの中を見ていただければおよそ検討をつけていただけると思いますが、Form2 のリストデータが、ダブルクリックされると
Handles ListData.DoubleClick　が定義された　ListData_DoubleClick　が呼び出されます。またキーが押下された場合には Handles ListData.KeyPress　が定義されている ListData_KeyPress　が呼ばれ、最終的にはさらにそれぞれが、ListSelected　を呼び出しています。ListSelected では選択されたアイテムを取得するところまでを行っています。
ただし、 ' 選択アイテムのデータを戻すロジックを追加してください。
のコメントがあり、この機能は自分で実装していただかなくてはなりません。

フォームからフォームへのデータの引継ぎをする手段についての考え方を説明します。
サブフォーム **Form2** はメインフォーム **Form1** からオブジェクトとして呼び出します。
Form2 の値を Form1 に引き渡す方法はいくつか考えられますが、今回はオブジェクト変数として Form1 から参照するようにしたいと思います。
そこで、外部から参照可能な Public 変数を Form2 に作成し、これに対して ListSelected で取得した選択結果をセットして終了するようにします。

```
'(owncode) 変数
' 明細行テーブル
Private DtSubs As DataTable
' リードレコード数
Private RecordCount As Integer

' 戻り値
Public strRetCode As String
```

上記のように、　**'(owncode) 変数**　部に赤の破線で囲った部分を追加します。

Public キーワードを宣言することにより外部から参照可能な変数となります。これが戻り値の格納場所です。
そして、ListSelectedで、取得した選択アイテムをこれにセットします。
ListSelected　は、下記のようになります。

```
Private Sub ListSelected()
    ' 選択アイテムがない場合、Exit
    If Me.ListData.SelectedItems.Count <= 0 Then Exit Sub
    ' 選択アイテム取得
    Dim selectItem As ListViewItem = Me.ListData.SelectedItems(0)

    ' 選択アイテムのデータを戻すロジックを追加してください。

    Me.strRetCode = selectItem.Text

    If Not LForm.IsMainForm Then
        Me.DialogResult = DialogResult.OK
        Me.Close()
    End If
End Sub
```

追加するのは、赤の破線部のみです。
ちなみに、**ListData.SelectedItems(0)**　は、選択されたリストアイテム行の先頭カラムすなわち 商品ＣＤ です。　したがってそのテキストプロパティ値を持つ変数 **strRetCode** は選択されたレコードの商品コードの値を保持していることになります。これで呼び出し元で選択結果を参照するための準備が完了しました。

次は、データの受け渡しには関係ないのですが、もともと独立したプログラムのサンプルである HAT110 商品一覧（ListView）では、終了時に本当にプログラムを終了させて良いかどうかの確認メッセージボックスを出力しています。
コード検索が閉じる毎にこのダイアログが出現するとおかしいので、これを消しておきましょう。

これは以下のようにフォームイベントエントリの　LEventKind.FormClosing　のイベント処理に書かれています。その該当部分を削除するかコメントアウトしてください。
実は該当箇所を見ていただくと、そのようにコメントで注釈されています。

```
'(owncode) フォームイベントエントリ
Public Sub LForm_Event(ByVal we As LWinEventArgs)
    Select Case we.Kind
        Case LEventKind.FormLoad
            ' フォーム開始処理
            FormLoad()

        Case LEventKind.FormClosing
'       ' プログラム終了前の確認(メインフォーム以外では、これを削除して下さい)
'           If MessageBox.Show("プログラム終了しますか？", "",
        MessageBoxButtons.YesNo) = DialogResult.No Then
'               we.Cancel = True
'               Return
'           End If

            ' フォーム終了処理
            FormUnload()
```

本来は、Form2 の実装はこれで完了です。ただしサンプル HAT110 の欠陥とも言えますが、このプログラムは、コンボボックスの品種コードを未選択のまま「検索」するとエラーを生じます。ついでに修正しておきましょう。FormLoad 時に初期値を設定しておきます。

```
'(owncode) フォーム開始処理
Private Sub FormLoad()
    ' リストデータのクリア
    LForm.Clear("ListData")
    '--------------------------------
    ' フォーム開始処理を追加

    HINSYU_CD.Text = "品種０１"

    '--------------------------------
End Sub
```

上記赤の破線部を追加します。
ではつづいて、Form1 での Form2 の呼び出し部を実装しましょう。
Form1 からは、今まで使用していた「コード検索」と同様に、Form2 も明細の商品コード欄がダブルクリックされるか、商品コード欄にフォーカスがある時にファンクションキー[F8]が押されると呼び出されるように実装します。
ということは、キーダウンや、ダブルクリックイベントをハンドリングする必要があります。しかし明細一覧表示（ListView）テンプレートには、これらのイベント処理は予め用意はされていませんので、自分で追加していただきます。

```
'(owncode) フォームイベントエントリ
Public Sub LForm_Event(ByVal we As LWinEventArgs)

        '<<――――――――省略――――――――>>

        Case LEventKind.ControlKeyDown
            ' コントロールキーダウン処理
            ControlKeyDown(we)

        Case LEventKind.ControlDoubleClick
            ' コントロールダブルクリック処理
            ControlDoubleClicked(we)

    End Select
End Sub
```

ControlKeyDown(we)、と ControlDoubleClicked(we)　の中身は、それぞれ

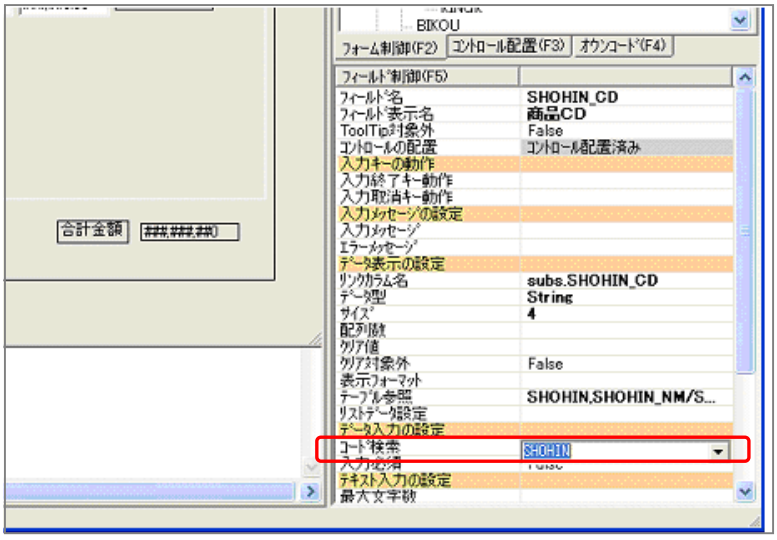
'(owncode)その他　にコーディングします。

ちなみに **LEventKind.ControlKeyDown**　は、マニュアルの　LEventKind 列挙体に紹介されていませんが、実際には使用できます。

'(owncode)その他　というコメント以外は全部新規のコーディングです。

```
'(owncode)その他

Private Sub ControlKeyDown(ByVal we As LWinEventArgs)
    Dim keyev As KeyEventArgs = CType(we.e, KeyEventArgs)
    If keyev.KeyCode = Keys.F8 And we.Name = "SHOHIN_CD" Then
        Dim sItem As New Form2
        If sItem.ShowDialog(Me) = DialogResult.OK Then
            LForm.PutData("SHOHIN_CD", we.Row, sItem.strRetCode)
        End If
    End If
End Sub

Private Sub ControlDoubleClicked(ByVal we As LWinEventArgs)
    If we.Name = "SHOHIN_CD" Then
        Dim sItem As New Form2
        If sItem.ShowDialog(Me) = DialogResult.OK Then
            LForm.PutData("SHOHIN_CD", we.Row, sItem.strRetCode)
        End If
    End If
End Sub
```

ControlKeyDown　ではファンクションキーF8 が押された時、その時点でのフォーカスが **SHOHIN_CD**(商品ＣＤ)にあれば　Form2　を sItem としてインスタンス化します。
ControlDoubleClicked　では、ダブルクリックされたのが **SHOHIN_CD であれば、**　Form2　を sItem としてインスタンス化します。最初の判断式が異なるだけで後半は同じコードです。

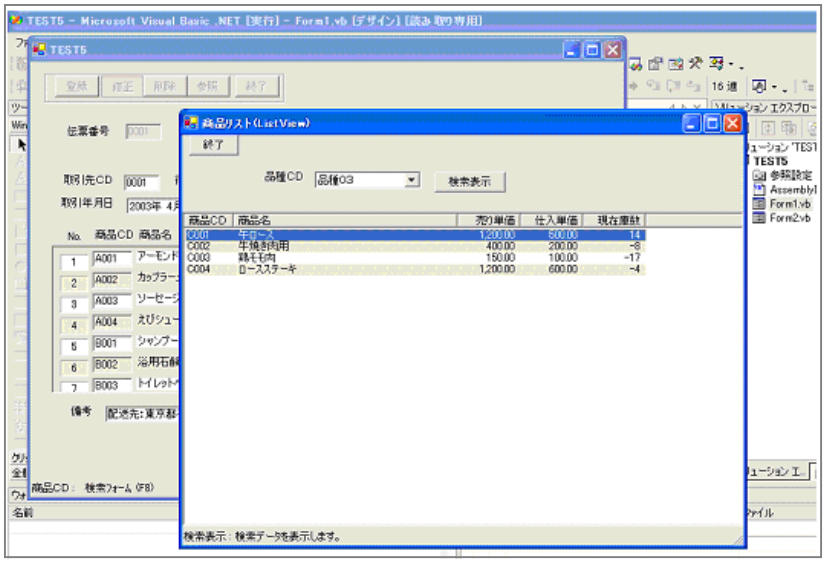


ここで、一旦 LLL/.net プログラム設計に戻ってください。Form1:TEST3 の「フォーム制御(F2)」->FORM->LDATA->LDET->SHOHIN_CD、すなち商品 CD フィールドの「データ入力の設定」->「コード検索」には、SHOHIN　というコード検索が設定され

ています。これは TEST3 を作成するときに指定したものですが、今回はこれの代わりに Form2:商品リスト（ListView）を使用しますのでこの設定を消してください。これを消去しないと、2 種類の検索ウインドウが表示されることになります。ツリービュー上の「コード検索」->SHOHIN ももちろん不要ですが、これは上記設定の消去で実害は無くなります。

次に、先ほど FORM2 のソース上でプログラムの終了時の確認メッセージを消去しましたが、もうひとつプログラムの開始時に開始処理中であることを示すメッセージボックスが表示されます。これはコーディングではなく、LLL/.net プログラム設計でのフォーム制御オブジェクトに対する設定で指定されています。これも直して（消して）おきましょう。フォームデザインパネルの Form2:商品リスト（ListView）タブに切り替え、「フォーム制御（f2）」->「FORM」のプロパティの先頭にある「開始フォームを非表示」を　True　に設定してください。これで開始フォームも出なくなりました。

さらについでですが、LLL/.net プログラム設計のフォームデザインパネル上のタブに FORM1 の名前が TEST3 と表示されています。これは、もともとこのフォームが TEST3 を複写したからです。これは LLL/.net プログラム設計では変更できません。
再度 VisualStudio.net に制御を戻し Form1 のプロパティを見てください。ここにある Text というプロパティが TEST3 になっています。これを TEST5 に変更しておいてください。次回 LLL/.net プログラム設計を開くとフォームデザインパネルのタブが Form1:TEST5 に変わっているはずです。では TEST5 を動かしてみましょう。



商品コードの検索ウインドウとして商品リスト（ListView）が使用できるようになりました。今回は商品リスト（ListView）の機能は選択結果を返せるようにした以外は殆ど変更しておりませんが、目的に応じて絞り込み条件の指定方法などいろいろな工夫により応用範囲が広がります。

９．スマートクライアント対応機能

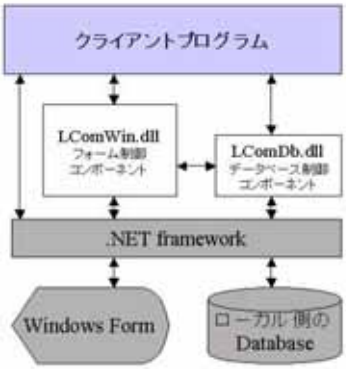
＜LLL/.net が実現するスマートクライアントとは＞

ここからは、Web サーバー（IIS）を使用するマイクロソフトのリッチクライアント Web アプリケーション、「スマートクライアント」の開発についてご説明します。
尚、本章でご説明する機能は、製品版のみ実際に動作できます。評価版では動作させてお試しいただくことはできません。
ただしお読みいただくことで、考え方としてはご理解いただけると思います。

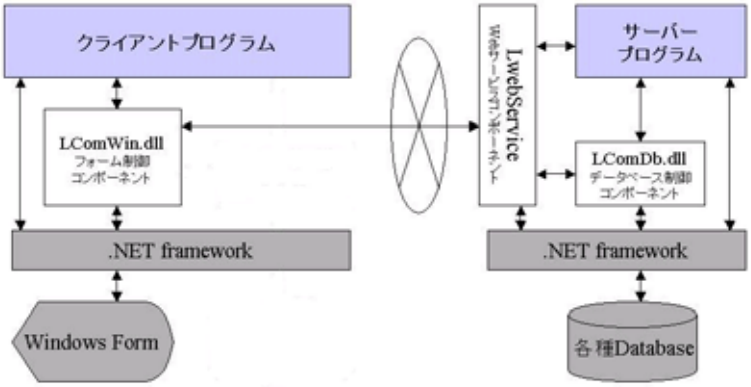
一般的に、インストールなどクライアントの環境作り、メンテナンスなど管理面においては WEB アプリケーションなどの「シンクライアント」が、また視認性、操作性、画面応答性などユーザービリティにおいてはクライアント/サーバーアプリケーションなどの「ファットクライアント」が有利とされています。スマートクライアントは、「シンクライアント」と、「ファットクライアント」のメリットを併せ持ち、「シン」でも「ファット」でもない、「スマート」なクライアントというものです。マイクロソフト社の造語ですが、その意味はかなり広い範囲を指しており実装方法も多々あります。

LLL/.net が実現するスマートクライアント型アプリケーションは、Windows フォームによるクライアントアプリをWeb サーバーから配信して実装し、プログラム更新時には自動アップデートします。クライアントからの DB アクセスなどサーバーとの通信は XML WEB サービスを持ちいて行うという仕組みを持ちます。

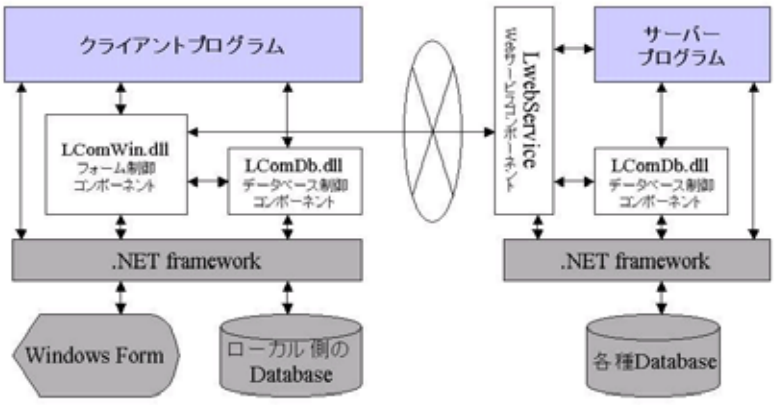
要するに今までに本書で作成手順を解説した各プログラム（TEST1～TEST5）や付属サンプルなどを、そのまま WEB のネットワーク環境で動作させようというものです。
スマートクライアントは、WEB サーバーを使用する WEB アプリですが、クライアントには WEB ブラウザは必要なく、画面が HTML に置き換わるわけでもありません。
クライアントではWindows フォームアプリがローカルプログラムとしてそのまま動作し、クライアントのローカル資源（DB や、プリンター等）のアクセスも自由に行えます。そのためクライアントには.NetFramework の CLR（ Common Language Runtime ／ 共通言語ランタイム）が必要になります。
LLL/.net が提供するスマートクライアント型アプリケーションと、クライアント/サーバー型アリケーションは、WEB サーバーアクセス用の XML WEB サービスコンポーネント（LwebService.dll）を使用したアクセスをするかどうかということ以外、基本的に殆ど同様の構造で動作します。殆どは運用環境の構築手段の違いであり、LLL/.net での開発作業に関しては、両者において殆ど違いはありません。



クライアント/サーバー型
（スタンドアロン型）構成



スマートクライアント型
構成



スマートクライアント型
複合型構成

■　LLL/.net アプリケーションの構成パターン図　■

当然ですがクライアント/サーバー型と、スマートクライアント型の、複合構成も可能です。通常は、Web を介して本社の DB をアクセスして業務を行っている営業所のノート PC を、営業マンが、お客様先や自宅に持ち込んだ場合は、ローカルディスクへアクセスし、それが保持しているデータを使用して業務を行うといったオフライン運用のシナリオの構築が可能です。
Web アプリは、ネットワークがつながっていないと一切の業務が行えないというのは、大きなデメリットと言われています。LLL/.net 作成したアプリでも、もちろんネットワークが切れていると WEB 経由の DB アクセスはできません。しかしそのような場合は Web を使用しない運用策を用意できるというのは大きなメリットです。

ただし、自動的に DB 接続を切り替えるようなロジックを搭載したテンプレートがあるわけではありません。そのような機能が必要であれば自分で実装していただく必要があります。

＜Web 経由のＤＢアクセスを実現する XML Web サービス＞

LLL/.net が提供するスマート クライアントの重要なポイントのひとつは、XML Web サービスです。XML Web サービスとは、本来はプロバイダーが提供した プログラムを、あらかじめその存在を知らないクライアントが利用でき、複数の異種システムを単一のコンピュータ ネットワークとして連携させることもできるという仕組みです。
その実現のために、 XML Web サービスレジストリなどサービス センターのような場所や、特定の XML Web サービスを検索するためには Web サービス記述言語 (WSDL: Web Services Description Language) が用意され、また通信用の主要プロトコルとしては SOAP が採用されています。

しかし、LLL/.net の XML Web サービスは、第三者に使用させることを目的としておりませんので、実際には前述の話はあまり意識していただく必要はありません。
仕組みとして XML Web サービスを使用しますが、それが WEB サーバーに配置していただくコンポーネント LWebService.DLL です。つまり実装はすでに終わっています。

LLL/.net の XML Web サービスは、クライアントが、サーバーに配置された LLL/.net のデータベース制御コンポーネント（LComDb.DLL）または自作のサーバーコンポーネントと連携して動作できる仕組みを LLL/.net が用意したものです。開発ディレクトリのシステムプロパティで「スマートクライアント型」を選択すると、標準コンポーネントとして提供される LWebService.DLL 等、WEB サービスとして必要な機能は自動的に使用可能になります。WEB サービスのクライアント側機能も、特に意識して開発する必要はありません。

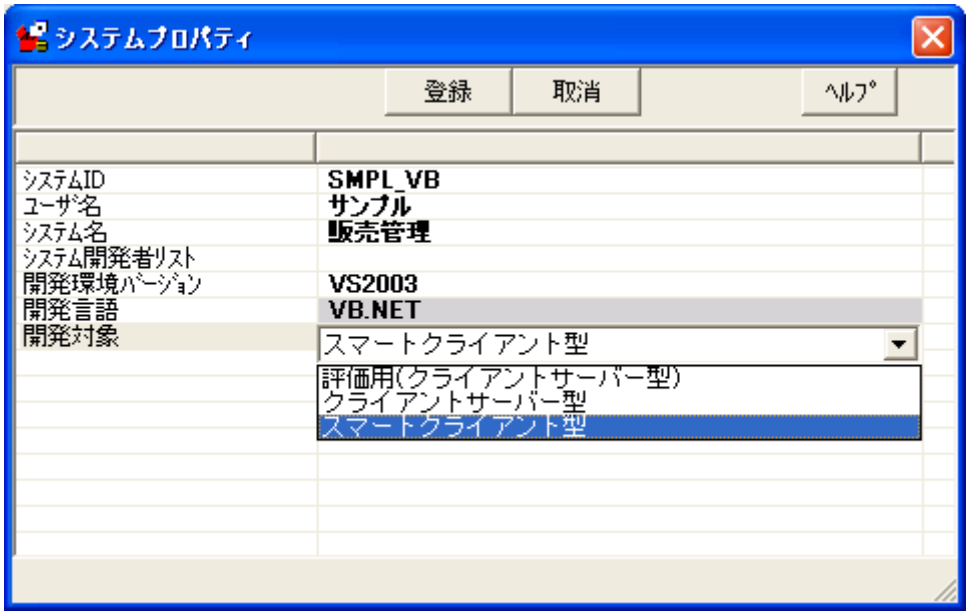
クライアント/サーバー開発でも使用する LComWin.Dll、LComDb.DLL がすでに機能を実装しています。
したがって XML Web サービスというと、よく出てくる、XSD スキーマ、UDDI、WSDL、SOAP、HTTP GET、HTTP POST、というような用語もとりあえず意識する必要はありません。

逆の言い方をすると、LLL/.net が標準で提供しているサービス以外の新しい独自の XML Web サービスを自由に作成することを支援する仕組みなどは、LLL/.net では提供していませんので、その点はご注意ください。
そのような機能が必要であれば、別の手段かご自分で実装していただく必要があります。

この仕組みにより、ブラウザベースの Web アプリケーションと同様に遠隔地のクライアントに配置された Windows フォームアプリケーションは、http または https プロトコルのみでサーバーとの通信が行え、問題なくファイアーウォールを越えることもできます。
この点はフレームリレーや専用線の回線を必要とした、従来のクライアント/サーバーアプリケーションにおけるリモートクライアントとは根本的に異なります。
現時点では公式にサポートしていませんが、この仕組みの採用により、マイクロソフト社製以外のサーバーをサポートできる可能性があります。

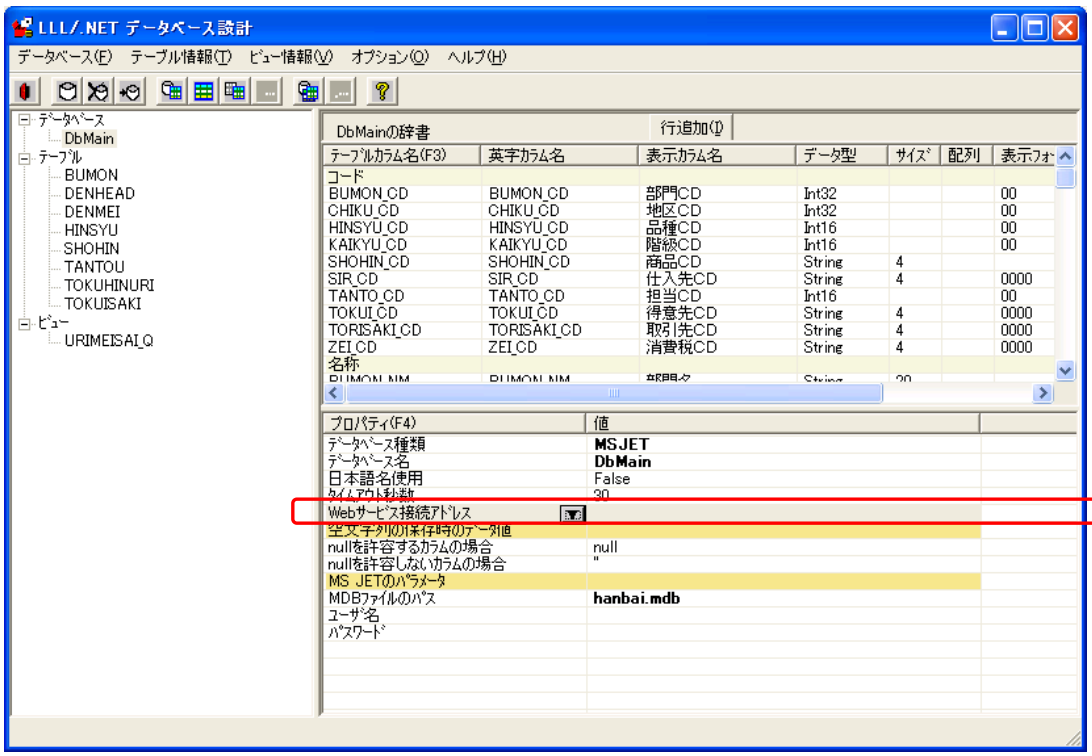
＜スマートクライアント型アプリケーションの作成＞

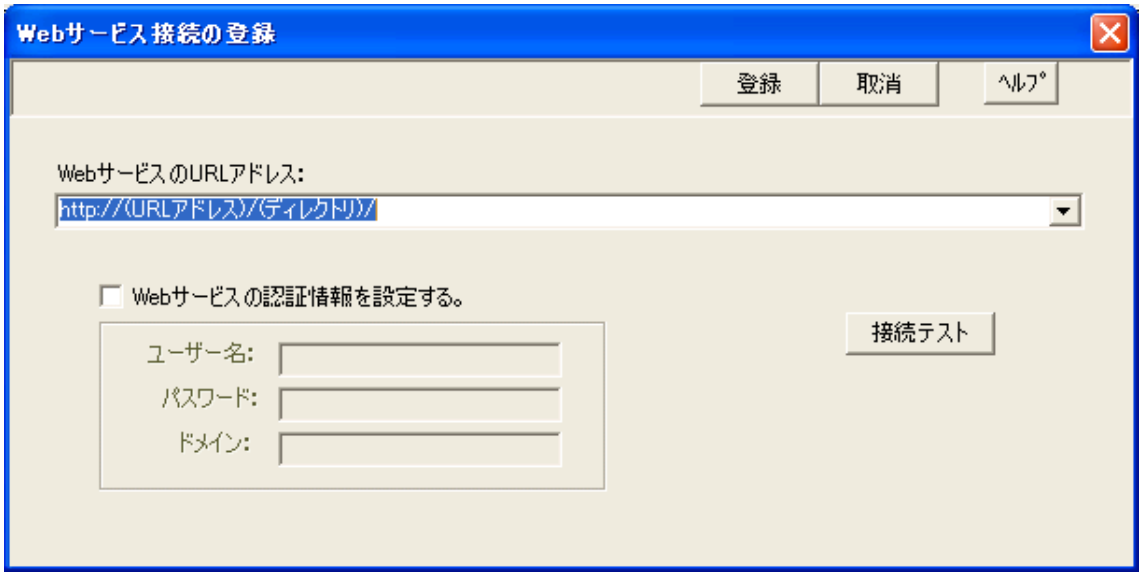
LLL/.net では開発対象がスマートクライアント型アプリケーションが、クライアント/サーバー型アリケーションかで、開発方法に殆ど違いはないと前述しましたが若干設定が異なります。
開発対象の指定は、プログラム単位ではなく開発ディレクトリ単位で行います。
今から開発するアプリケーションがスマートクライアント型である場合、LLL/.net プログラム設計の、「システム」->「システムプロパティ」で「開発対象」を「スマートクライアント型」に設定変更する必要があります。



この操作により、開発ディレクトリに対して XML Web サービスに必要となる、[bin]フォルダーや、[LWebService.dll]コンポーネントが追加され利用可能になります。

さらに XML/WEB サービスを使用した DB アクセスを行うためには、「LLL/.NET データベース設計」で、Web サービス接続アドレスの設定を行います。





この準備だけ行っておけば、プログラムの開発はクライアント/サーバー型でも同じです。WEB サーバー立ち上げ、公開ディレクトリに完成したアプリケーションをアップロードしてください。
WEB サーバーの設定については、マニュアルの「スマートクライアント」->「WEB サーバーの設定」をご参照ください。

＜デプロイメント（Web 経由のプログラム配信と自動アップデート）＞

デプロイメントとは、インストールとほぼ同じ意味合いですが、主に Web アプリケーションなどをネットワーク経由でクライアントから利用可能な状態にすることを指します。

クライアント/サーバー型アプリを利用可能にするためには、サーバーからクライアントへのプログラム配布の問題だけでなく、クライアントでレジストリキー登録や DLL 競合がつきまとうなどの問題があり、この点が Web アプリにアドバンテージを与えているとされていました。しかしこの問題は、.NetFramework の登場で大きく状況が変わりました。

.NetFramework の Common Language Runtime（共通言語ランタイム：CLR）が監視して実行する、Managed Code（日本語表記では、マネージドコード、またはマネージコード）の基本的なビルディングブロックをアセンブリと呼びます。そのアセンブリはマニフェストとモジュールから構成されています。マニフェストは、ファイルなどアセンブリ構成の詳細情報を保持しています。 さらに、モジュールは、IL（Intermediate Language（中間言語））と呼ばれる CLR が実行可能として認識するコードと、その実行可能コードが含む型、変数、

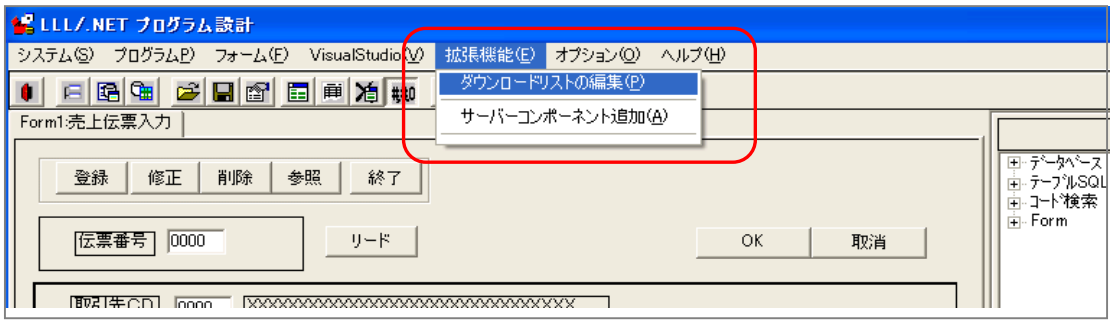
メソッドの引数などの詳細情報が記述されたメタデータから構成されています。

つまり Win32 の EXE や DLL からこのような形に進化したマネージドコードの強力な自己記述性は、複数のアプリケーションでの共有を目的とする.DLL の競合問題を回避し、レジストリにも依存しない XCOPY コマンドによるプログラムインストールを可能にしました。

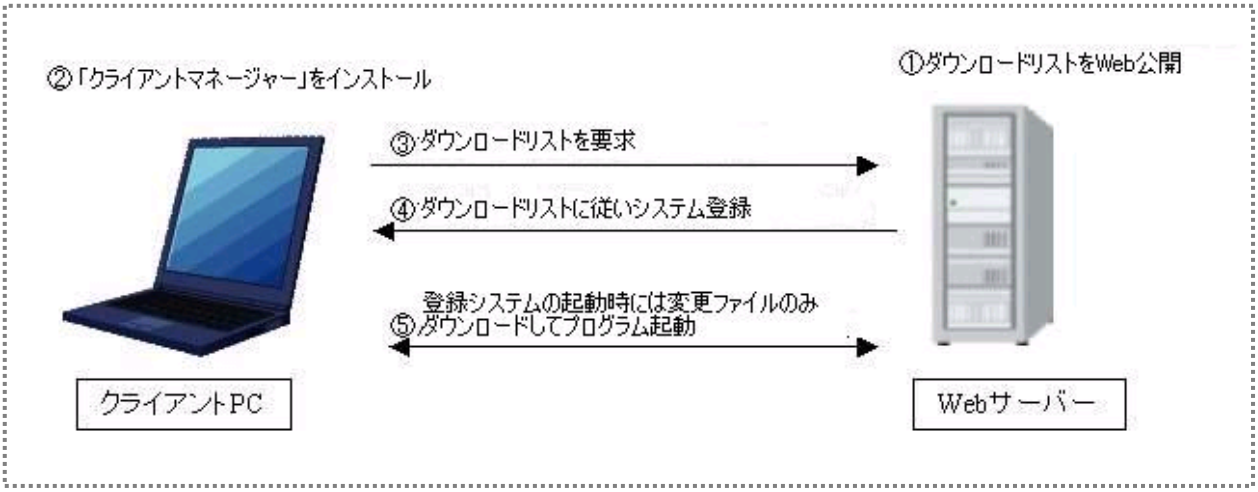
.NetFramework マネージドコードアプリケーションは、Web サーバーに配置してプログラムへのリンクを作成すれば、クライアントのダウンロードキャッシュ上でコード・アクセス・セキュリティによる部分信頼レベルでアプリを利用できます。部分信頼は安全のためには望ましいとのですが、社内業務アプリケーションでは不便な場合もあります。
そのような場合には、サーバーからプログラムをダウンロードし、ローカル環境へ完全配置（ディスクへの COPY）を行ってからプログラムを起動するダウンローダーとランチャーを実装することで、完全信頼による制限の無いローカル資源活用と、プログラムの自動アップデートも可能になります。
LLL/.net におけるその実装が、標準提供している LLL/.net デプロイメントツールです。

LLL/.net デプロイメントツールが提供するファイル配布機能は、.Net のアセンブリ情報に依存しません。したがってアンマネージドコードプログラムやデータファイルでもクライアントに配布可能です。ただし配布されたプログラムが、クライアントでセットアップを必要とせず動作できるかどうかは、マネージドコードでない場合はその実装に依存します。

LLL/.net デプロイメントツールは、配布対象となるファイルのリストを作成するための**「ダウンロードリストエディター」**と、そのリストに基づき WEB サーバーからプログラムをダウンロードしてクライアントのローカルディスクへコピーし、さらに指定された開始プログラムを起動するランチャーでもある**「クライアントマネージャー」**で構成されます。
デプロイメントツールの配布対象は、LLL/.net で作成したプログラムに限りません。そのためこのツールだけを単独で別売も行っています。
「ダウンロードリストエディター」は、LLL/.net のプログラム設計メインメニューの「拡張機能」->「ダウンロードリストの編集」から起動できます。（※　製品版のみ）



「クライアントマネージャー」は、LLL/.net をインストールしたディレクトリの直下にできる ClientManager というサブディレクトリの中の CM_Setup.EXE という自己解凍形式の圧縮ファイルにインストーラパッケージが入っていますので、それを使用するクライアントマシンへメディアかネットワークを通じて持って行き、事前にインストールしておいていただきます。



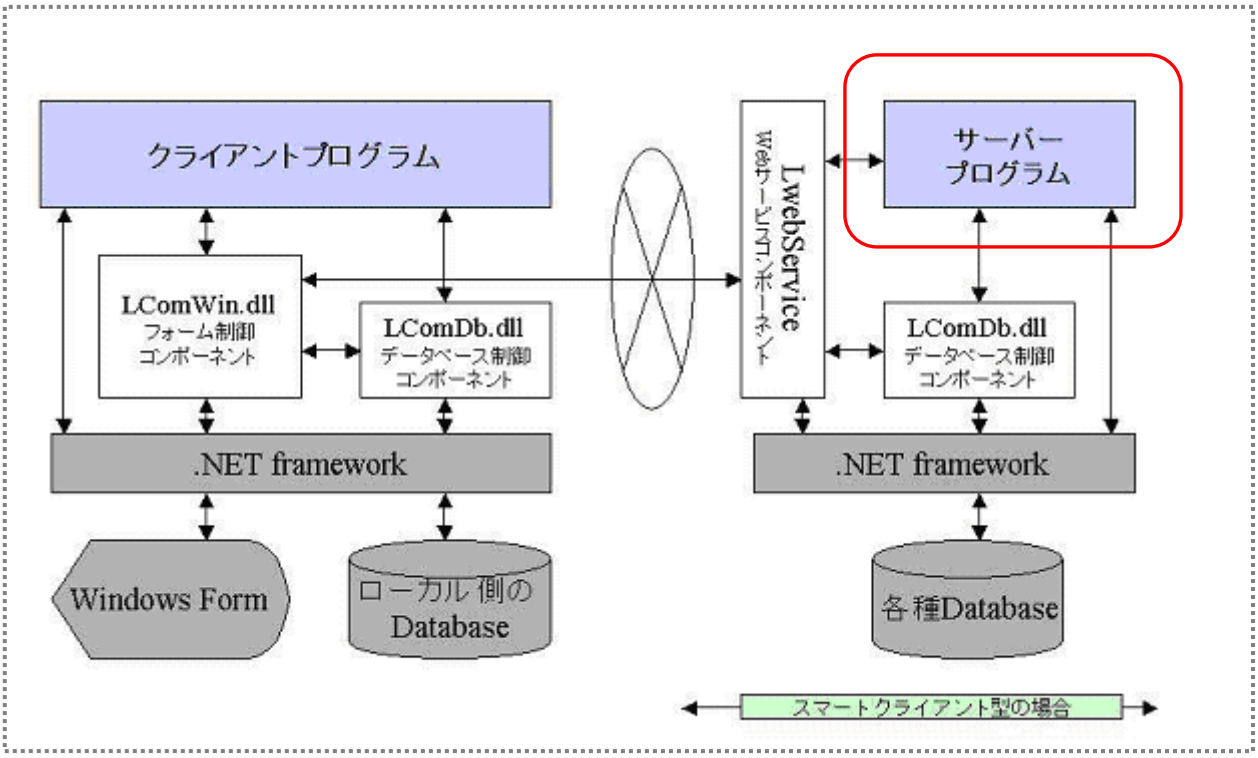
スマートクライアントのデプロイメントという意味では、サーバーからクライアントへのダウンロードに先立ち、本来は Web サーバーへのアプリケーションの配置という作業も発生しますが、LLL/.net デプロイメントツールにはサーバーへのアップロード機能などは特に用意しておりません。
LLL/.net における、スマートクライアント型アプリケーションは、プログラム配信とデータベースアクセスに WEB サーバーや WEB サービスを使用しますが、クライアントで動作しているプログラム自体は、Windows フォームのローカルプログラムであり、これはスタンドアロン型やクライアント／サーバー型におけるクライアントプログラムと同じ仕組みのものです。したがってスマートクライアント型プログラムの開発は、基本的にクアライント/サーバー型の開発と同じです。今まで本書で説明してきた、TEST プログラム 1～５の説明は、スマートクライアントの開発においても全て当てはまります。

本章に、スマートクライアントとしての特有のコードの説明や、サンプル作成が無いのはそのためです。しかし、スマートクライアントならではの開発（コーディング）というものが全く無い、または出来ないのかというと、そうではありません。
Web サーバー連携を行う場合、LLL/.net が提供する標準コンポーネントとは別に、Web サーバー上にユーザープログラムを分散させるというアプリケーションの構築方法があります。これにはメインプログラムとは別のコンポーネント（DLL）を作成してサーバーに配置し、Web サービスを経由して連携させるという方法で対応します。
これをサーバーコンポーネントと呼びますが、これについての説明は次章で行います。

１１．サーバーコンポーネント

＜サンプルＰＧの説明とデータセットの扱い＞

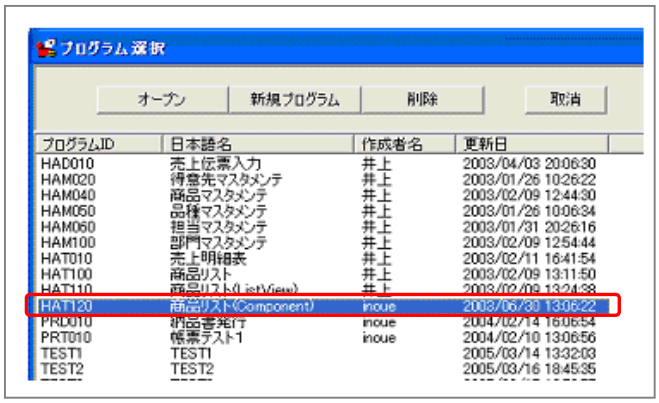
LLL/.net では、サーバーコンポーネントを作成する機能も提供します。
サーバーコンポーネントとは、Web サービスサーバーで動作するコンポーネントで、スマートクライアント型のアプリケーションで利用できます。データベース処理などの共通処理をコンポーネントとして Web サーバーに登録し処理の効率化やプログラミングの標準化に役立てられます。本章で説明する機能も評価版では実際にお試しいただけません。
下図の「サーバープログラム」と書いた部分がそれに相当します。



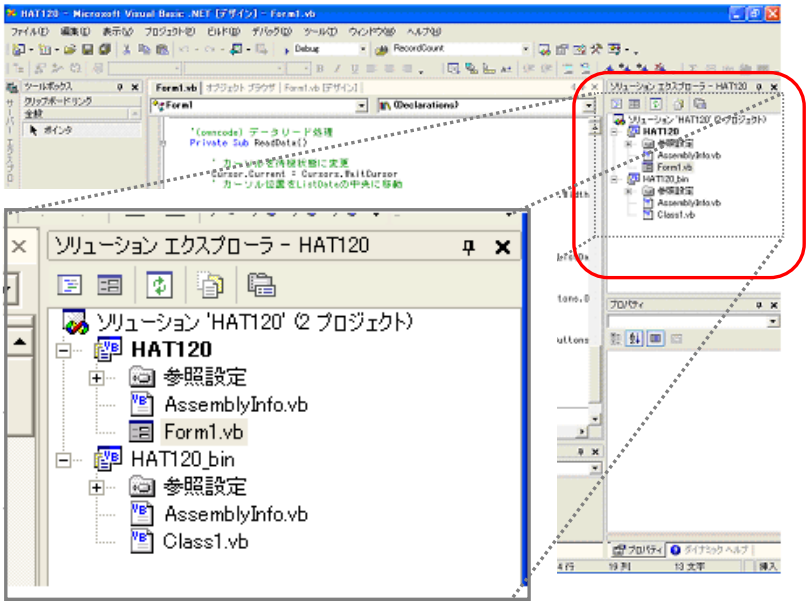
サーバーコンポーネントを公開 Web サイトディレクトリの bin サブディレクトリに配置すると、Web サービス経由で動作します。
このコンポーネントはクライアントのローカルディスクに配置して使用するローカルコンポーネントとして作成することも可能です。そこでスマートクライアントとは独立した章として説明します。

サンプルの中に、コンポーネントを使用した例がありますので、見てみましょう。
これは、評価版でもご覧いただけます。**HAT120 商品リスト（Component）**がそれです。

このプログラムは品種コードをドロップダウンリストで選択した後「検索表示」ボタンを押すこと該当する品種に属する商品一覧をリストビューコントロールに表示するというものです。外見上はTEST4でも使用した　**HAT110 商品リスト（ListView）**と同じですが、SHOHINテーブルから品種をキーとしてデータを取得する部分をコンポーネント（DLL）として切り出し、WEBサーバー上で動作するロジックとして実装したサンプルです。では、このプログラムの設計情報を見てみましょう。

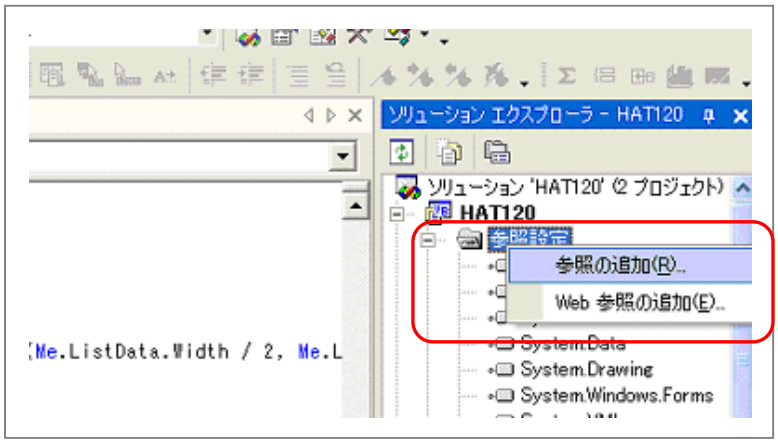


LLL/.net プログラム設計で、HAT120 商品リスト（Component）を呼び出して、そのままVisualStudio.net を起動し、ソースを確認してみましょう。



ソリューションエクスプローラーを見ると、このソリューション HAT120 は、２つのプロジェクト

で構成されていることが分かります。HAT120 は、**「PGA200 明細一覧表示(ListView)」**テンプレートで作成された呼び出し元のプロジェクト、HAT120_bin が呼び出されるサーバーコンポーネントのプロジェクトです。プロジェクト HAT120 の「参照設定」のマウス右クリックか、メニューの「プロジェクト」から「参照の追加」を選び HAT120_bin プロジェクトを追加していただければデバッグで動作させられます。



では、呼び出し元である HAT120 が、実際に HAT120_bin を呼んでいる部分**「'(owncode) データリード処理」**のコーディングを確認してみましょう。コーディングの冒頭は、

```
        ' カーソルを待機状態に変更
        Cursor.Current = Cursors.WaitCursor
        ' カーソル位置をListDataの中央に移動
        Cursor.Position = Me.ListData.PointToScreen(New Point(Me.ListData.Width / 2,
Me.ListData.Height / 2))
```

となっています。これは表示するデータ量やマシンスペックによっては、クライアントとコンポーネント間の処理に時間がかかる可能性もあることから一旦カーソルを待機状態（砂時計）にセットし、位置をリスト中央に移動させています。これは**「PGA200 明細一覧表示(ListView)」**テンプレートに最初から書かれているコードです。コンポーネント呼び出しには直接関係ありません。
前述ような考慮が不要の場合は削除することも可能です。

```
        ' リストデータをクリア
        LForm.Clear("ListData")
        ' フォームデータを取得
        Dim cmdDs As DataSet = LForm.GetFormData()
         ' コンポーネント実行
        Dim retDs As DataSet = LForm.ExecuteDLL("HAT120_bin", "Class1.GetListData",
cmdDs)
```

そして、リストを初期化した上で、**LForm.GetFormData** でフォームデータを取得しその値であるデータセット　**cmdDs**　を引数としてコンポーネントを呼び出しています。実際にコンポーネントに引き渡したいのは絞り込み用「品種コード」ですが、品種コードだけではなくフォーム上の全フィールドデータをデータセットにしてこれを引数として使用します。

そしてコンポーネントは、引数で指示された条件によって実行した結果のデータセットを戻り値　**retDs**　として返してきます。
コンポーネントを呼び出した **LForm.ExecuteDLL**　は、サーバーコンポーネントを実行するフォーム制御オブジェクト　**LWinForm**　のメソッドです。
通常はメインデータベース dbMain の Web サービスサイト上のサーバーコンポーネントを実行します。引数でデータベース名を指定すれば、メインデータベース以外のサーバーコンポーネントを実行することも可能です。引数としてアセンブリ名・クラス名・メソッド名を指定することで実行するコンポーネントメソッドを指定します。
そしてエラーがなければ、**LForm.PutFormData**　で戻り値をフォームに表示しています。

```
    '  リストデータを表示
    Dim errmsg As String = retDs.ExtendedProperties("Error")
    If Not errmsg Is Nothing Then
        MessageBox.Show(errmsg, "コンポーネントエラー", MessageBoxButtons.OK)
    Else

        LForm.PutFormData(retDs, "ListData")

        If LForm.GetWinArea("ListData").RowCount = 0 Then
            MessageBox.Show("データありません。", "確認", MessageBoxButtons.OK)
        End If
    End If

    ' カーソルを通常に戻す
    Cursor.Current = Cursors.Default
```

表示完了後、カーソル形状を元（矢印）に戻して完了しています。この部分もテンプレートが予め用意したコードです。コンポーネント呼び出しのエラー確認は データセットオブジェクト retDs の　拡張プロパティ **ExtendedProperties("Error")**　で確認しています。
ExtendedProperties　は、カスタム情報をオブジェクトと共に格納できる機能で、後で出てきますが呼び出されたコンポーネント内でデータセットに値を追加しています。

コンポーネントを呼び出した LForm.ExecuteDLL メソッド実行で引数として使用するデータは、ひとつの DateSet に設定しています。

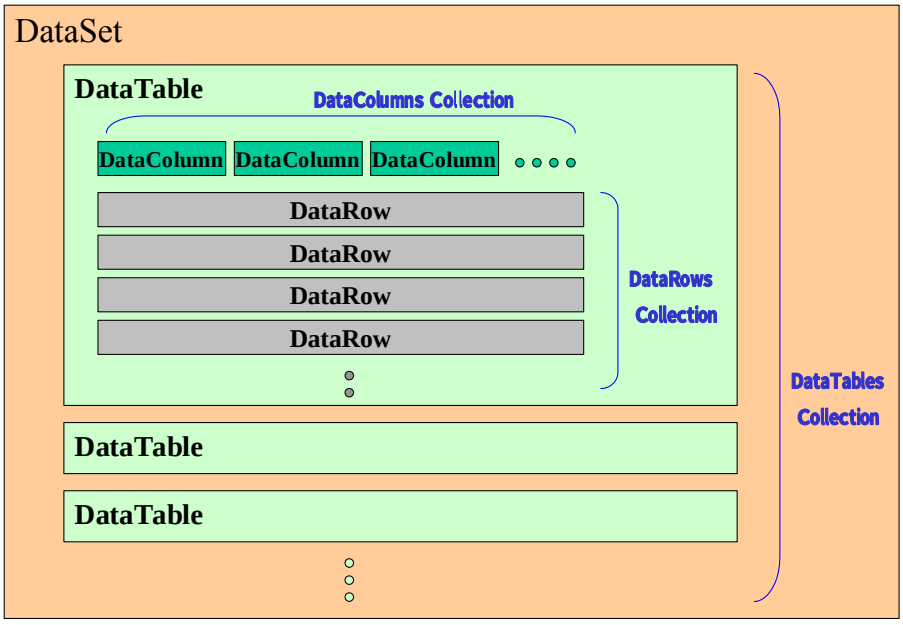
LForm.ExecuteDLL の詳細は、マニュアル**「ランタイムコンポーネント」->「フォーム制御コンポーネント」->「フォーム制御メソッド」->「ExecuteDll」**をご参照ください。

繰り返しになりますが、このようにコンポーネントは DataSet を引数として実行し、結果も DataSet で返してきます。これは DataSet が XML 表現を作成できることから WEB サービスの利用に適しているためです。

DataSet に関しては、フリーフォーマットテンプレートのところでも簡単に説明しましたが、ここで再度確認しておきましょう。

データセットとは、ADO.NET が標準で用意した非接続型モデルの汎用データ構造クラスです。

データセットの実体は、System.Data 名前空間にある DataSet クラスに関連した一連のクラスのことで、主なクラスとして次の４つがあり、図のような包含関係になっています。



DataColumn クラス　：　テーブルの列（名前、データ型など）

DataRow クラス　：　列の数だけの実データを持つ行

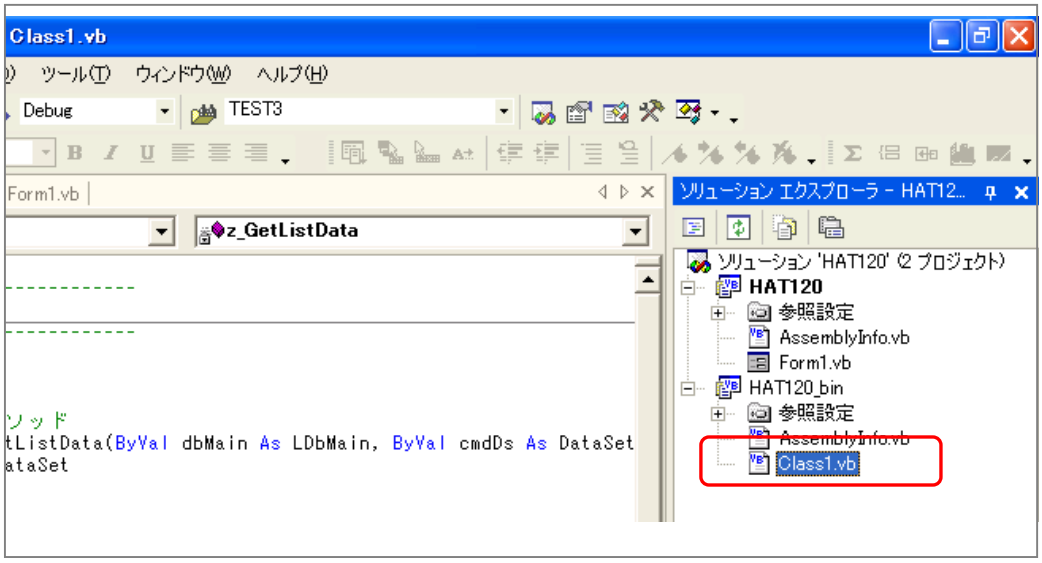
DataTable クラス　：　複数の列と行からなるテーブル

DataSet クラス　：　複数のテーブルからなるデータベース

etc.

データセットの構成

DataSet は、他にもテーブル間のリーレーションを表す　DataRelation クラス　など多くのメンバーで構成されています。
DataSet　は前述の通り、「非接続型」であり実際にデータベースにアクセスする機能は備わっていません。データベースに接続してテーブルのレコードを取得し、値をデータセットに設定するのは　.NET データ・プロバイダに含まれるクラスであるデータアダプタなどの役割です。LLL/.net　ではデータアダプタの操作を、データベース制御コンポーネント（LComDb.DLL）が内部で行っています。
次は呼び出されるコンポーネントのソースコードの方を見てみましょう。
ソリューション HAT120 のプロジェクト HAT120_bin のソース Class1.vb を開いてください。



HAT120_bin プロジェクトの、ソースファイルは Class1.vb という名前です。
サーバーコンポーネントは、文字通りサーバーで動作することが想定されており、クライアントに表示されるフォームは持っていません。
そのコードは以下のようになっています。

```
        '--------------------------------
        Imports LLLNET.LComDb
        '--------------------------------
        Public Class Class1
            ' コンポーネントメソッド
    Public Function GetListData(ByVal dbMain As LDbMain, ByVal cmdDs As DataSet) As
DataSet
        Dim retDs As DataSet
```

前述の通り、引数　ByVal cmdDs As DataSet　は　cmdDsが　データセット型であることを表し、戻り値も　Dim retDs As DataSet　のようにデータセット型として用意しています。

```
        Try
            ' リストデータを取得
            retDs = z_GetListData(dbMain, cmdDs)

        Catch ex As Exception
            ' エラーメッセージ
            retDs = New DataSet
            retDs.ExtendedProperties("Error") = "[HAT120_bin].[Class1.GetListData]" &
ex.Message()
        End Try

        Return retDs

    End Function
```

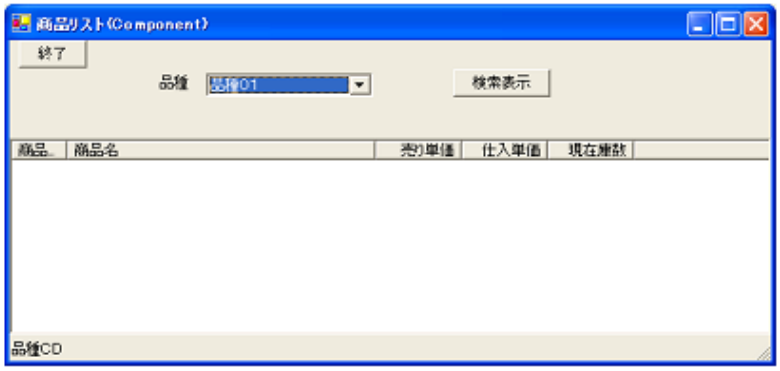
Try、Catch　キーワードは、.Net Common Language Runtime リリースによって、VisualBasicでも使用可能になった構造化例外処理を示します。これの詳細はMSDN等でご確認ください。
retDs.ExtendedProperties("Error") = "[HAT120_bin].[Class1.GetListData]" & ex.Message()では、呼び出し側で参照していたretDs.ExtendedPropertiesの　"Error"　にエラーメッセージを格納しています。これもLLL/.netならではの機能ではありません。

z_GetListData(dbMain, cmdDs)　で処理の本体を呼び出し、Return　retDs　でデータセットを使って戻り値であるリストデータを返しています。以下がリストにデータをセットするコーディングの本体です。

```
    Private Function z_GetListData(ByVal dbMain As LDbMain, ByVal cmdDs As DataSet) As
DataSet
        ' フォームデータ取得
        Dim dtMain As DataTable = cmdDs.Tables("_all")
        Dim dtList As DataTable = cmdDs.Tables("ListData")
        dtList.Rows.Clear()
```

まず呼び出し元から渡ってきたデータセットに格納されているテーブル情報を、ここで宣言した DataTable 型 の dtMain と dtList 上で保存した上で、dtList の行（Rows）の値を一旦クリア（初期化）しています。

"_all" と "ListData" がデータテーブルの名前です。
データテーブルと言っても、これは dbMain すなわちサンプルの Hanbai.mdb の「商品テーブル」や「得意先テーブル」ではありません。
コンポーネントの引数として渡したデータセット cmdDs に格納されているフォーム上のフィールドデータを収めたテーブル名すなわち、フォームに作成したエリアの名前です。
"ListData" は、ここではリストビューのことで商品の一覧を表示するエリアのことです。
"_all" は、マニュアルに記載がありませんが、フォームそのものに直接貼られ、特定のエリアに所属しないフィールド指す場合の仮想エリア名です。



現実にはこのフォーム上にリストデータ以外のデータは「品種」しかありません。
cmdDs.Tables("_all") を代入したことにより dtMain にはフォーム上の品種コードが保存されたことになります。

```
' データベースリード
Dim hincd As String = dtMain.Rows(0)("HINSYU_CD")
```

次に品種コードを文字列変数 **hincd** に保存します。
前述とおり、データテーブル dtMain には実際には「品種コード」しか格納されていませんが、それは今回の結果論であり、dtMain はDataTable構造を持ったオブジェクトですので、そこから特定のデータである品種コードを抜き出すというコーディングがなされています。それが、式の = の右部分です。 DataTableオブジェクトに含まれる DataRowCollection のindex 0番の Item すなわち DataRow の中で、さらに **"HINSYU_CD"** という ColumnName を持つItemの値を保存しています。これは省略形の記述方法を採用していますが、丁寧にすると下のようにも書けます。

```
        Dim hincd As String = dtMain.Rows.Item(0).Item("HINSYU_CD")
```

こう書いた方が、意味が分かりやすいかもしれません。

```
        Dim dt As DataTable = dbMain.ReadTable("SHOHIN", "select * from SHOHIN where
HINSYU_CD = " & hincd)
```

次に　dt という DataTable オブジェクトを宣言し、データベース制御コンポーネント（LComDb.dll）の ReadTable　メソッドを使用して上記で取得した品種コードに該当する商品を取得して代入します。

dbMain.ReadTable("SHOHIN", "select * from SHOHIN where HINSYU_CD = " & hincd)

dbMain は LDbMain 型のオブジェクトとして宣言されていますが、これは LLL/.net データベース制御コンポーネント（LComDb.Dll）に定義されているデータ型で接続情報を含むデータベース情報の集まりです。サンプルで使用している Hnbai.Mdb の情報を表します。
前述通り、LLL/.net では LComDb.Dll 内でデータアダプターオブジェクトを作成し、データベースに接続を行っています。

すなわちこのコードは、Hanbai.Mdb の SHOHIN テーブルに対し、
" select * from SHOHIN where HINSYU_CD = " & hincd
とういSQL を実行し、戻り値をデータテーブル dt にセットしたことになります。

別の言い方をすると、hincd　には画面で選択した「品種コード」が入っていますので、その品種に属する全商品のリストを　dt という名前で作成したということです。

```
        '  フォームデータ設定
        For Each dr As DataRow In dt.Rows
            Dim drList As DataRow = dtList.NewRow()
            drList("SHOHIN_CD") = dr("SHOHIN_CD")
            drList("SHOHIN_NM") = dr("SHOHIN_NM")
            drList("TANKA_URI") = dr("TANKA_URI")
            drList("TANKA_SIIRE") = dr("TANKA_SIIRE")
            drList("GEN_ZAISU") = dr("GEN_ZAISU")
```

```
            dtList.Rows.Add(drList)
        Next
        ' フォームデータを戻す
        Return cmdDs
    End Function

End Class
```

次に、For Each ～ In ～ Next 構文を使いループ処理で cmdDs に含まれるリストデータのテーブル dtList の更新を行っています。

まず、SQL文で select した結果のデータテーブル dt の RowCollectionのItemのであるDataRow型オブジェクト、すなわち1件(行)分のレコードを取り出し 別に用意したDataRow型オブジェクト dr に格納します。

そして **dtList.NewRow()** によってdtListのRowCollection のItemである DatatRow型オブジェクト **drList** を新たに宣言し各カラム名に該当するItemに dr の同名カラムデータを代入します。
"SHOHIN_CD"、"SHOHIN_NM"、"TANKA_URI"、"TANKA_URI"、"TANKA_SIIRE"、"GEN_ZAISU"の全てについて代入が完了したら１件（行）分の drList の完成です。
これを dtList.Rows.Add(drList) で dtListのRowCollection のItem として追加（Add）します。

これを dt の Rows の全 Item すなわち前述の SQL 文で select された全行数分のレコードに対して繰り返します。
先に **dtList.Rows.Clear()** で dtList の RowCollection は一旦初期化されていますので、今回新たに取得したデータにリストデータは置き換えられたことになります。
そしてこの新たなリストデータを含む データセット cmdDs を呼び出し元にリターンし Class1 のコーディングは終了しています。

select した結果のデータテーブルをそのまま dtList に代入できれば、もっとコーディングがすっきりして分かりやすいのですが、使用した SQL 文は 「select * from SHOHIN・・・」となっており SHOHIN マスターから条件に該当する全レコードの全カラムを取得しています。それに対し代入を行っている先のデータテーブルは画面フォーム上の ListData エリアのリストテーブルであるため、両方とも DataTable オブジェクトではあり

ますが、双方のテーブル構造が一致している保証はありません。そのためにこのサンプルでは各レコードの各カラムのデータ毎に 1 件づつ代入してリストを作成しているということに留意してください。

＜コンポーネントテンプレート＞

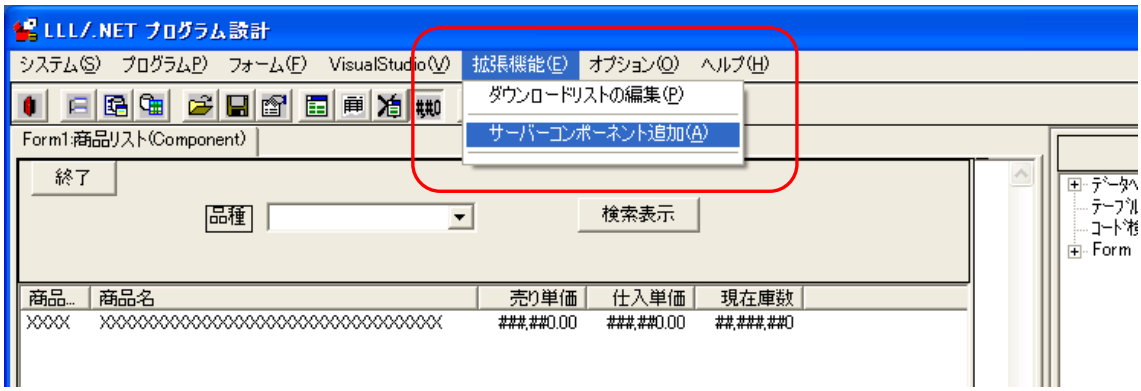
このコンポーネントサンプルの処理コードの大部分はオウンコードとして書かれたものですが、一部はテンプレートが自動的に用意しています。実はコンポーネント用にも**テンプレート**があります。
「プログラム選択」で表示されるフォームテンプレートの一覧に出てくるものではありません。このテンプレートは「サーバーコンポーネント追加」を行う時に選択できるもので、**コンポーネントテンプレート**と呼び、**フォームテンプレート**とは区別して扱っています。

LLL/.net をインストールしていただたディレクトリの下の「Enterprise」->「VS2003」->「Component」-> 「vb」->「Template」の中の　PZA100.vb　がテンプレートです。
その中身は、以下の通りです。

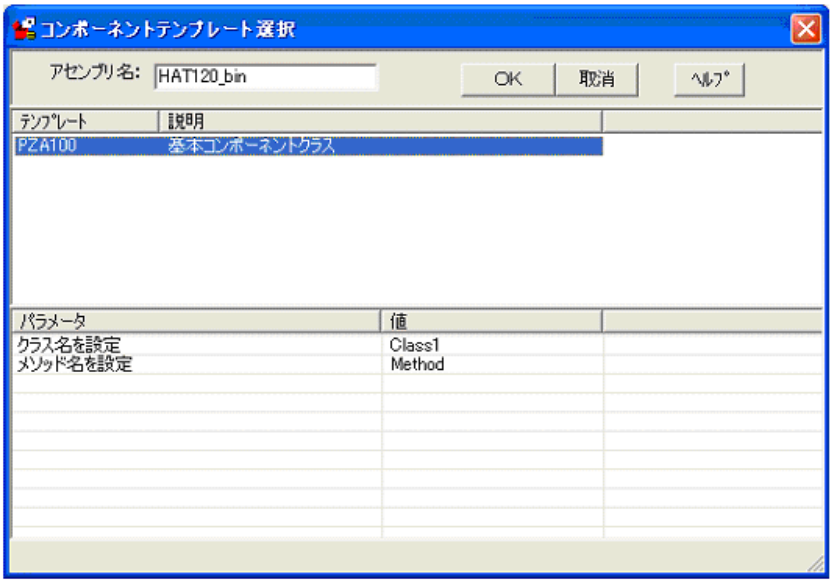
```
' [Template]
' Template#caption=基本コンポーネントクラス
' [Data]
' class#caption=クラス名を設定|default=Class1|input=text
' method#caption=メソッド名を設定|default=Method|input=text

  '--------------------------------
  Imports LLLNET.LComDb
  '--------------------------------

  Public Class {*class}

    ' コンポーネントメソッド
    Public Function {*method}(ByVal dbMain As LDbMain, ByVal cmdDs As DataSet) As DataSet
       Dim retDs As DataSet
```

```
        Return retDs
    End Function

End Class
```

サーバーコンポーネントの作成は、LLL/.net プログラム設計のメインメニューに「拡張機能」->「サーバーコンポーネント追加（A)」で行います。(※　評価版では「拡張機能」は選択できません)



これを選択すると、まず「コンポーネントテンプレート」選択ダイアログが表示されます。
製品御購入時には、PZA100「基本コンポーネントクラス」しかありません。
これを選択すると、VisualStudio.netのフォーム名ソリューションに　フォーム名_bin という名前のプロジェクトが追加され、class1.vb　が作成されます。



LLL/.net「プログラム設計」ではコンポーネントの追加作業だけをここで行い、後はVisualStudio.netで必要な処理コードをオウンコーディングとして追加

記述していきます。
「コンポーネントテンプレート」は「フォームテンプレート」と同様にカスタマイズや追加が可能です。

LLL/.net が標準提供しているフォームテンプレートでは、ビジネスロジックはクライアントアプリケーションとして実装しており、特にコンポーネントテンプレートとの連動を前提としたものはありません。
しかしそういう前提のテンプレートを作成し、ビジネスロジックの大部分をサーバーに実装するというスタイルのアプリケーションの構築に利用することも可能です。

というのは、スマートクライアントのクライアントプログラムは、基本的にユーザインタフェース機能の提供に専念すべきであるという考え方もあるからです。
スマートクライアントがサポートする範囲は、ユーザインタフェースとユーザインターフェースプロセスの部分とし、ビジネスロジックは Web サービスなどの形式で実装し、クライアントは Web サービスと通信をすることにより、アプリケーション機能を実現するべきという考え方です。
このような実装には、まさにサーバーコンポーネントを活用が有効です。

サーバーコンポーネントに実装したビジネスロジックは、バージョンアップが発生してもクライアントにダウンロードし直す必要がないというメリットもあります。
ただし、クライアントを利用する環境が必ずしもオンラインであるとは限りません。この場合を想定し、オフラインでも利用する形態も考慮できるのもスマートクライアントのメリットです。ここを重視する場合はある程度クライアントへのビジネスロジックの実装も止むを得ないとも言えます。

ビジネスロジックの実装をクライアントサイド、サーバーサイドのどちらへ行うか、また混在させる場合にはどのような基準で切り分けるのかはケースバイケースで判断する必要があります。実装形態の違いによるメンテンナンス性に与える影響は LLL/.net が提供するスマートクライアント型アプリケーションの仕組みにより、ほとんど無くなっています。

Web に限らずネットワークを使用するアプリケーションでは、ネットワーク上を流れるデータの量と頻度が極力少なくなるように実装することは運用効率上重要です。
実情に応じて適切に判断してください。

サーバーコンポーネントのアセンブリ名・クラス名・メソッド名は、後で変更したり追加することは自由にできます。しかし、パブリックメソッドの引数や戻り値は決まった設定

にする必要があります。引数や戻り値に使用する DataSet のデータ構成は自由にきめることができます。基本コンポーネントクラスでは、メソッド内の処理は全てコーディングする必要があります。

サーバーコンポーネントの作成の詳細は、マニュアルの**「スマートクライアント」->「サーバーコンポーネント」->「サーバーコンポーネントの作成」**をご参照ください。

HAT120 のソリューションをビルドすると、HAT120.exe と HAT120_bin.dll ができあがります。
今、LLL/.net プログラム設計の「システム」->「システムプロパティ」->「開発対象」が「クライアント　サーバー型」、または「評価用クライアント サーバー型」のままでお試しであれば、HAT120.exe と HAT120_bin.dll を hanbai.mdb が参照できる¥Sample¥Hanbai_VB ディレクトリにコピーしてください。そして HAT120.exe を起動すると動作を確認できます。

この場合、HAT120_bin.dll は、HAT120.exe と同じディレクトリにありますので実際にはサーバーコンポーネントではなく、Web サービスを経由せずにローカルコンポーネントとして動作しています。
サーバーコンポーネントとして動作させるためには、開発ディレクトリのシステムプロパティの「開発対象」を「スマートクライアント型」として開発し、できあがった dll は、Web サーバーに配置して公開しておく必要があります。

その他、スマートクライアントの詳細については、マニュアル「スマートクライアント型」をご参照ください。また、スマートクライアントについては、マイクロソフト他、インターネット上の多くのサイトが解説していますのでご参照ください。

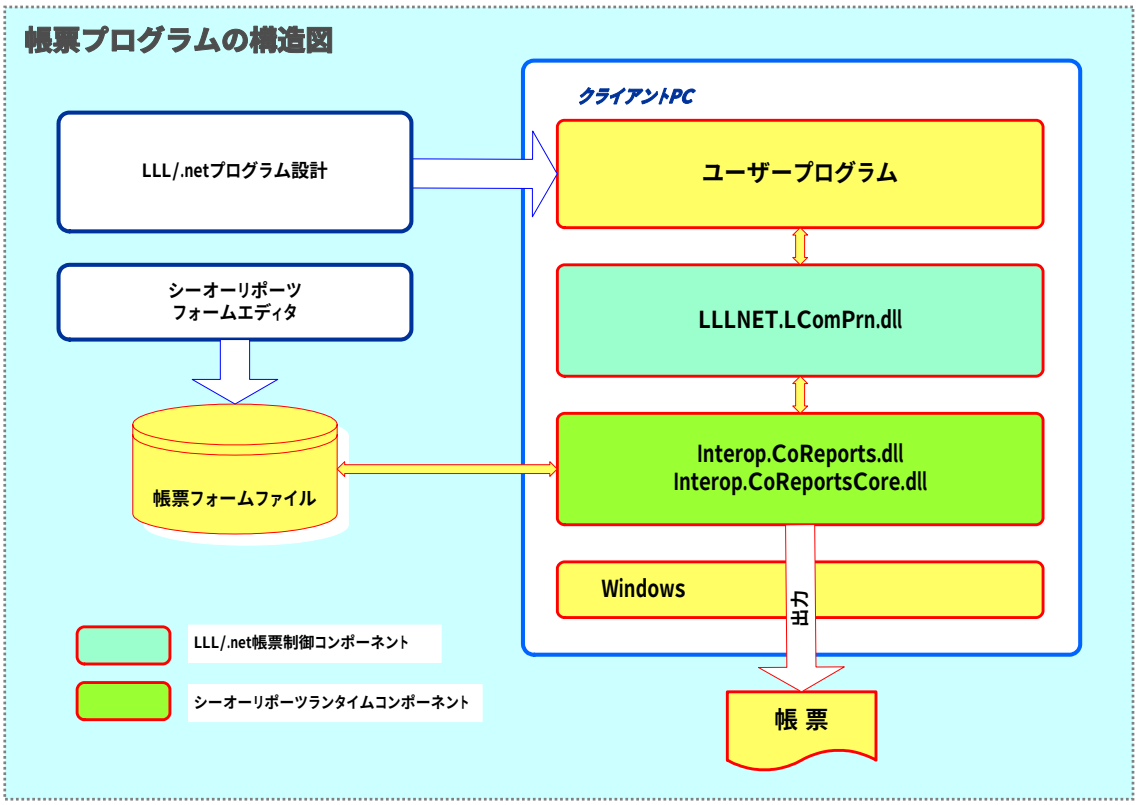
１２.帳票設計オプション

＜LLL/.net 帳票設計オプションの概要＞

次に、プリンターに帳票を出力するプログラムについて説明します。
VisualBasic.netや、C#といったプログラム言語の機能のみでは、複雑な帳票を作成することは難しいため、通常は何がしかの帳票作成ツールを組み合わせて実現します。
LLL/.net が帳票プログラム開発を支援する機能として、**「LLL/.net 帳票設計オプション」**という別売りのプロダクトがあります。このオプションを追加していただくと、**シーオーリポーツ**（CoReports 8.5 StandardEdition）を帳票エンジンとして使用する、帳票出力プログラムを楽に作成できます。

LLL/.net アプリケーションから帳票を出力させる場合、「帳票設計オプション」やシーオーリポーツ以外の帳票ツールを使用してはいけないということではありません。
自分でコーディングされる限り、どのようなツールを使用されるかは自由です。
したがって「帳票設計オプション」は必須ではなくオプションという扱いになっています。

帳票設計オプションは、**帳票制御コンポーネント（LComPrn.DLL）**、**帳票テンプレート**、そして　**CoReports 8.5 StandardEdition** によって構成されます。



< シーオーリポーツ Ver.8.5 Standard Edition >

シーオーリポーツ Ver.8.5 は、株式会社 エイチ・オー・エス殿の製品で、帳票システムを効率よく開発するための支援ツールです。Visual Basic や Delphi などでは手間のかかる罫線の多い複雑なレイアウトの帳票を簡単に作成することができます。QR コード、PDF417 バーコード、EAN128（コンビニバーコード）にも標準で対応しています。シーオーリポーツ Ver.8.5 では、PDF ファイルでの帳票出力が可能となった「Standard Edition」と Web サービス向けの帳票システム開発を強力にサポートする「Enterprise Edition」の 2 製品があります。

シーオーリポーツ Ver.8.5 Standard Edition は、専用フォームエディタ、ActiveX 描画コントロール、専用ビュアー、ActiveX ビューコントロール、PDF ライブラリ、二次元コード QR コードライブラリ、二次元コード PDF417 バーコードライブラリ、Version4.X 互換 DLL、コード生成ウィザードなどで構成されています。

詳しくは、下記のエイチ・オー・エス社のサイトをご覧ください。
http://www.hos.co.jp/

前述のとおり「LLL/.net 帳票設計オプション」で作成する帳票プログラムは、実行時に帳票エンジンとしてシーオーリポーツ Ver.8.5 Standard Edition を使用します。
フォームデザイン時や実行時には、この製品の帳票フォームエディターやランタイムが必要になります。

シーオーリポーツ Ver.8.5 には、**「Standard Edition」**と**「Enterprise Edition」**の 2 製品がありますが、「LLL/.net 帳票設計オプション」で使用するのは**「Standard Edition」**の方です。
「Enterprise Edition」はサーバー製品であり、XML 出力やサーバーでの印刷、ロギングに対応しています。
スマートクライアントで運用するなら、「Enterprise Edition」が必要と思われるかもしれませんが「LLL/.net 帳票設計オプション」で使用する場合に限っては必要ありません。
LLL/.net におけるスマートクライアントアプリケーションでは、デプロイメントツールが「Standard Edition」の実行エンジンをサーバーからクライントに配布して実行させます。
また XML/Web サービスによるクライアント/サーバー間の通信は LLL/.net のコンポーネン

トが行ないます。
したがってLLL/.netではシーオーリポーツ「Enterprise Edition」が持つそれらの機能を必要としません。シーオーリポーツはサーバーで動作させるのではなく、あくまでもローカルのクライアントプログラムとして動作しますので、「Standard Edition」を使用します。
また、エイチ・オー・エス社の説明によると「Enterprise Edition」は、「Standard Edition」を包含する製品ではありません。すなわち、**「Enterprise Edition」を既にお持ちであってもこれは使用できません。**「LLL/.net帳票設計オプション」においては、**必ず別途「Standard Edition」が必要**であることに御注意ください。

シーオーリポーツVer.8.5は、.NetFrameworkのマネージドコードプログラムではありません。すなわちスマートクライアントのノータッチデプロイメントや、ClickOnceではクライアントに配置できないプログラムです。
そこでアセンブリ情報に依存せずあらゆるタイプのファイルの配信に使用できる LLL/.net デプロイメントツールが役に立ちます。シーオーリポーツ Ver.8.5 の実行エンジン（ランタイムコンポーネント）は、初回配布後にクライアントで 1 回だけインストール作業が発生します。しかしそれさえ完了してしまえば、以降の帳票フォームの変更や追加は、全て自動更新で行われ、スマートクライアントアプリケーションとして動作します。

LLL/.net「帳票設計オプション」による帳票プログラムは以下のような機能を持つことができます。

１．　画面フォームのデータを取得して帳票出力。
２．　ＳＱＬで取得したテーブルデータを帳票として出力。
３．　明細行データが複数ページになる場合、２ページ目以降の帳票フォームを変更。
４．　ページ内の明細フィールドを２～９の段組構成で出力。
５．　明細フィールドの見出しや合計などのグループ処理機能。
６．　帳票フィールドのテーブル参照やデータ変換などの各種編集機能。
７．　帳票データのプレビュー表示機能。
８．　帳票ドキュメントのファイル出力機能。
９．　サブ帳票オブジェクトを使用することで複数帳票フォームによる帳票出力可能。

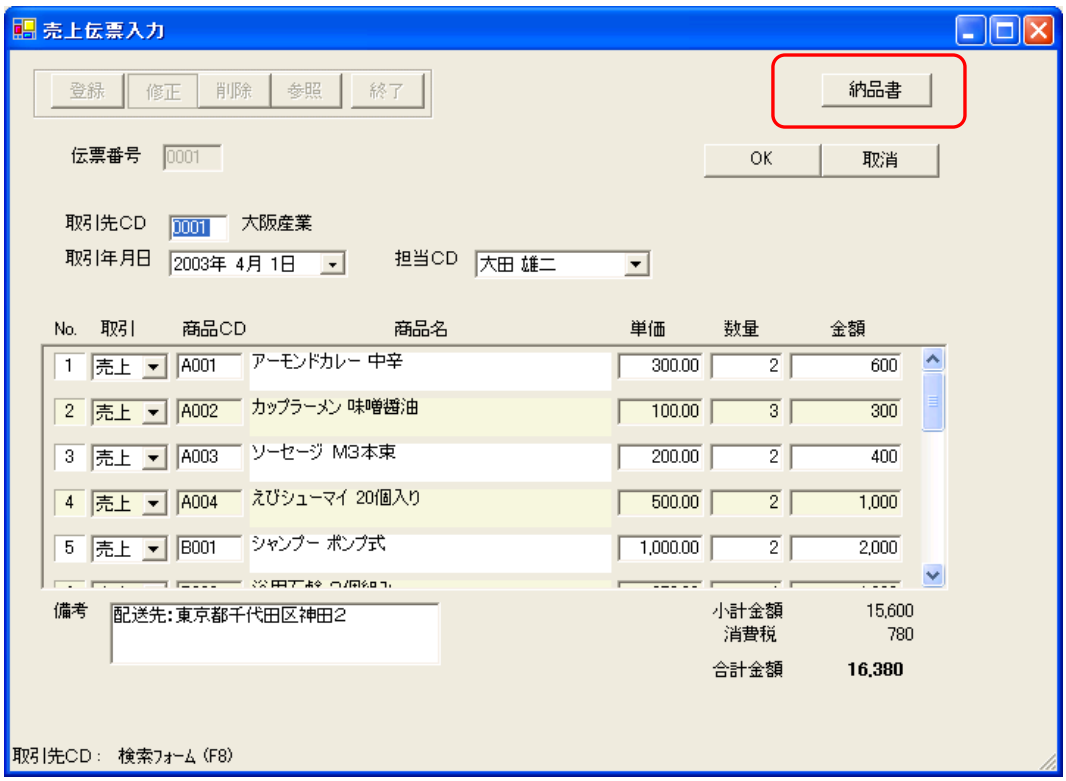
LLL/.net「帳票設計オプション」は、シーオーリポーツフォームエディターで作成された帳票フォームに対し、LLL/.net で作成されたプログラムから上記の動作を指示できる機能として、帳票テンプレートと LLL/.net 帳票制御コンポーネント（LComPrn.DLL）を提供します。

帳票出力プログラムの設計手順は、概ね次ぎようになります。

１．シーオーリポーツのフォームエディターで使用する帳票フォームをデザインします。
２．LLL/.net プログラム設計で帳票出力を実行する画面フォームをデザインします。
３．LLL/.net プログラム設計で「フォーム」->「帳票フォームの追加」から、テンプレートとして使用する帳票フォームクラスを選択して追加します。
４．帳票フォームクラスのテンプレートパラメータに、使用する帳票フォームファイルを選択指定します。
５．LLL/.net 帳票フォーム設計で各種制御プロパティを登録し、帳票フォームクラスを作成します。
６．画面フォームプログラムに、帳票フォームクラスを使用して帳票出力するためのオウンコードをコーディングします。

このようにして作成したプログラムが付属サンプルの**「PRD010 納品書発行」**と**「PRT010　帳票テスト１」**です。では、実際に作成された帳票プログラムの動きを見てみましょう。

プログラム選択で**「PRD010 納品書発行」**を選択し、VisualStudio.net を起動して「デバッグ」->「開始」実行してみてください。



「納品書」というボタンがある以外は、以前に見た売上伝票入力のサンプルプログラム**「HAD010 売上伝票入力」**とそっくりな画面が立ち上がります。
実際にこのサンプルプログラムは、「HAD010 売上伝票入力」を複写し、画面データを印刷する機能を付加して作成したプログラムです。「修正」か「参照」をクリックした上で、適当な伝票データを表示させて「納品書」ボタンを押してみてください。

下記のようなプレビューウィンドウが表示されます。

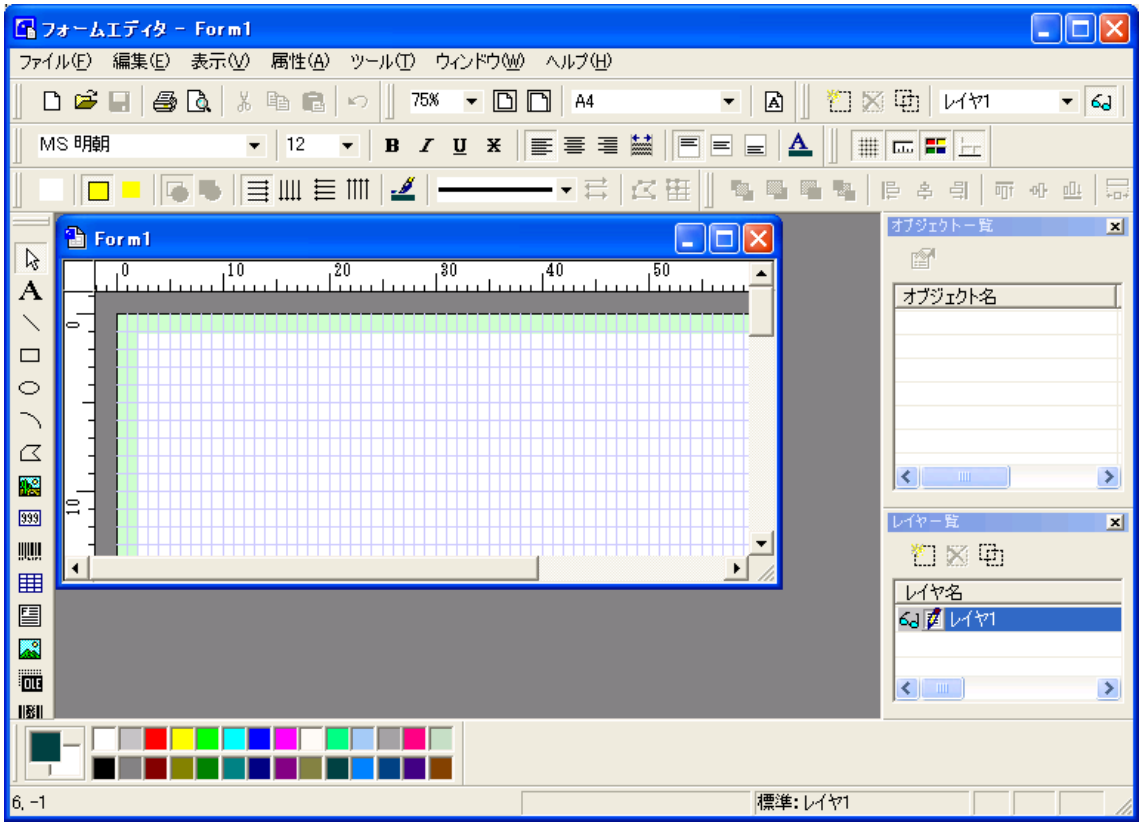
この印刷プレビューウィンドウは、LLL/.net 帳票制御コンポーネント（LComPrn.DLL）が作成するウィンドウで、シーオーリポーツが提供するビューアーとは別物です。
印刷プレビューウィンドウのツールバーの「印刷」ボタンを押してここからプリンターに印刷させることもできます。
また、このプレビューウィンドウを表示させることなく直接プリンターに出力させることも可能です。

＜簡単な帳票プログラムの作成＞

（テストプログラムの作成　その６）

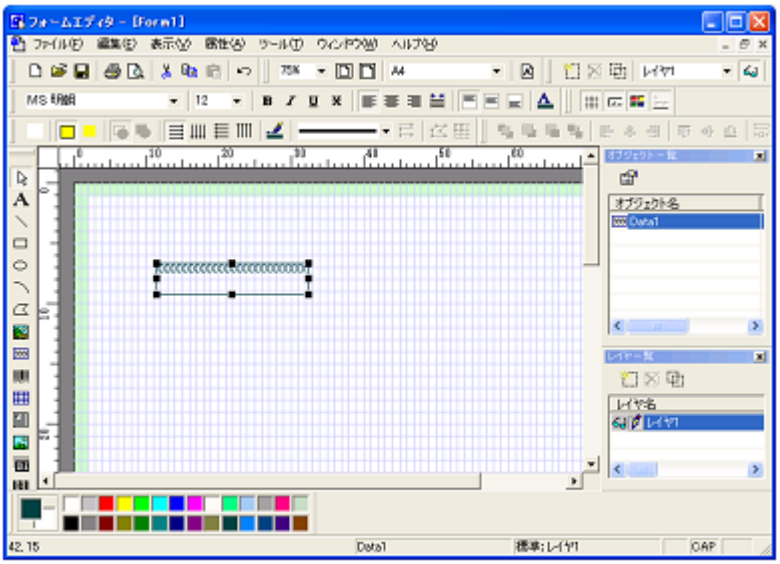
では、実際にプログラムを作成してみましょう。
ここから先の操作は、**シーオーリポーツ Ver.8.5 Standard Edition** がインストールされていることが前提となります。製品版をお持ちでなければ、エイチ・オー・エス社のサイトにアクセスし、シーオーリポーツ Ver.8.5 Standard Edition の体験版を入手してインストールしてください。（ HOS 社ホームページ　http://www.hos.co.jp/package/ ）

まずはシーオーリポーツのフォームエディタを使用して帳票フォームをデザインします。
シーオーリポーツがインストール済みであれば、Windows の**「スタート」->「すべてのプログラム（P)」->「CoReports Ver.8.5」->「フォームエディタ」**から帳票フォームエディタを起動できます。



最初は、上記のような画面が起動されます。
エディタの編集画面左端のツールボックスより配置したい目的の「描画オブジェクト」をマウスで選択し、編集画面で最初にクリックした場所を起点（オブジェクトの左上点）と

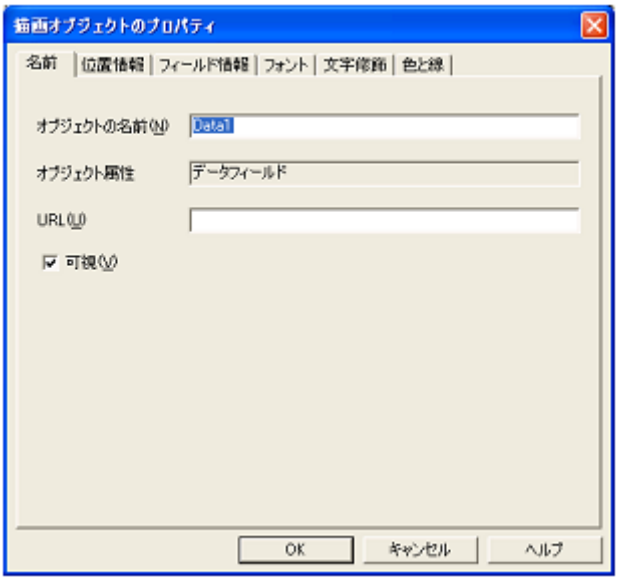
して描画が始まります。マウスボタンをクリックしたまま編集画面上を移動させ、マウスボタンから手を離した場所が終点（オブジェクトの右下点）として描画を終了し、オブジェクトの配置が完了します。



例えば最初に　ツールボックスから 999 （データフィールド）を選択し、まず１個だけこれの配置を行なったとすれば、左図のような状態になり、自動的にオブジェクト名として **Data1** という名前が付けられます。

もちろん２個目は　**Data2** です。
対応する、画面フォーム上のフィールドやテーブル項目があるのであれば、その名前と同じにしておくと分かりやすくなります。

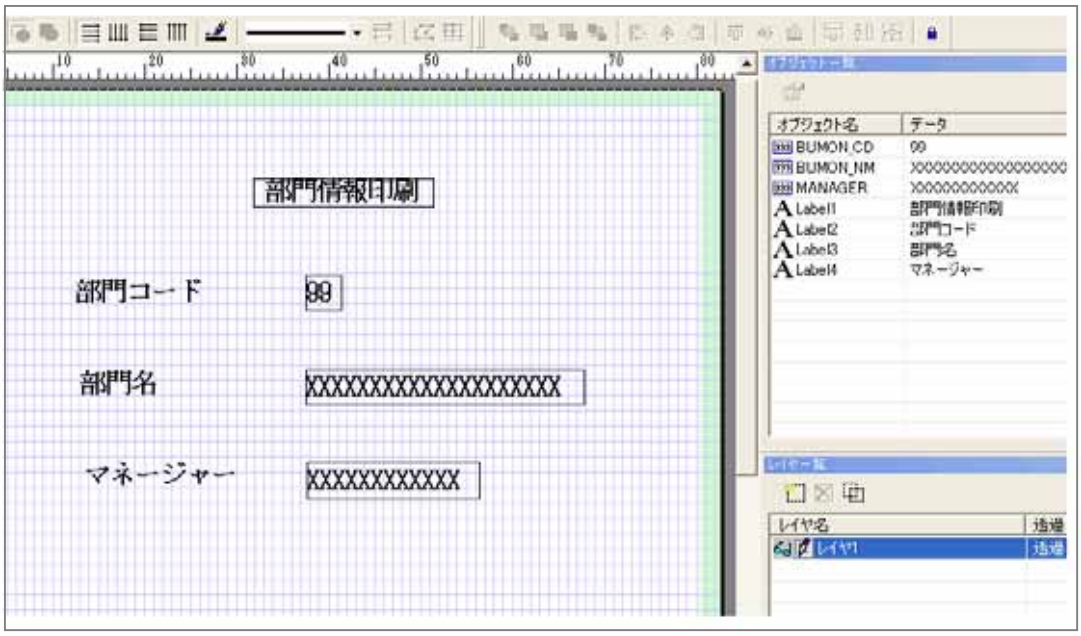
名前の変更や、フィールドの書式設定などは、目的のオブジェクト上でマウスの右ボタンクリックを行ってポップアップウィンドウを出力させ、その中にあるオブジェクトのプロパティを設定することで行なえます。



ツールボックスの A （ラベル）から下が描画オブジェクトです。さらにその内 999 （データフィールド）より下がプログラムから実行時に可変値を出力するフィールドです。それより上のピクチャボックスまでが固定値項目で帳票フォームエディタでデザインが確定する部分になります。

このエディタによる帳票デザインの方法と注意点の詳細については、LLL/.net マニュアルの**「システムの操作」->「帳票フォーム設計（帳票オプション）」->「帳票フォームデザイン」、並びに**シーオーリポーツフォームエディタの**「ヘルプ」**ご参照ください。

とりあえず部門マスタメンテナンス（TEST1）の画面情報を印刷するプログラム作ってみましょう。
シーオーリポーツフォームエディタを起動し、まずは帳票やフィールドの見出しとなる固定文字をラベル（ A ）として作成します。次にプログラムから値を出力するフィールドをデータフィールド（ 999 ）として作成します。
各項目の枠線のあるなしを含む修飾や、データフィールドの出力編集条件などはフォームエディタ上で配置済みコントロールをマウス右クリックしてポップアップを出力させ、その中からオブジェクトのプロパティを選んで「描画オブジェクトのプロパティ」パネルを呼び出して設定行います。



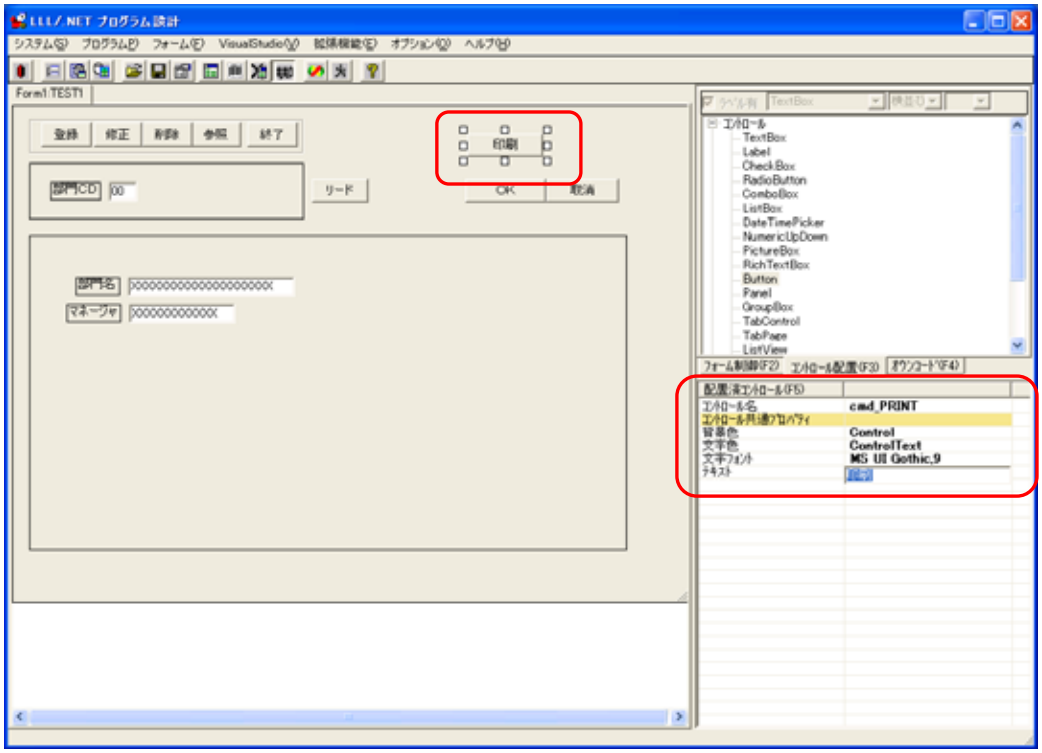
例えば上記のようなイメージになります。
ここではプログラムの組み方をまず習得していただくことが目的ですので、デザインに凝る必要はありません。
これに「TEST6.crf」という名前を付けて、開発ディレクトリのルートに保存します。

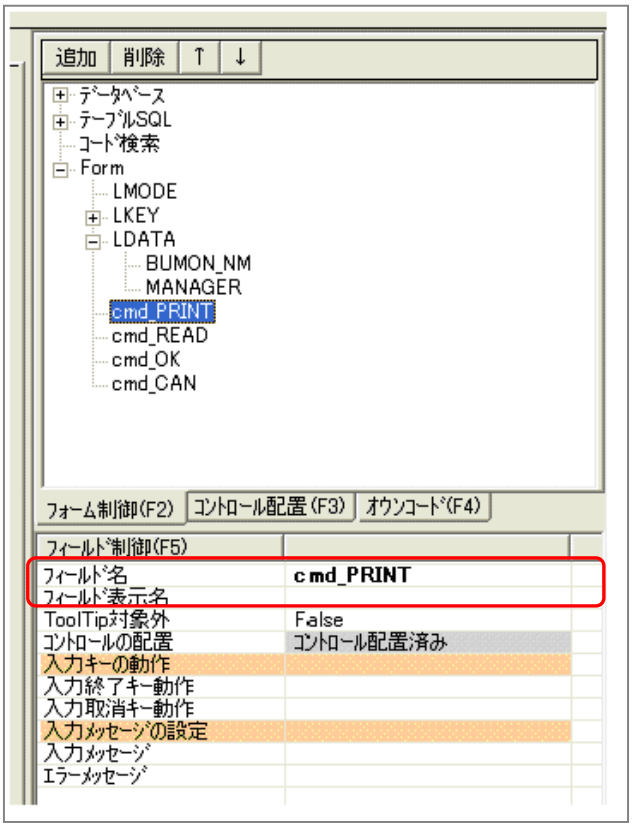
次に、この帳票を起動する呼び出し元プログラムを作成します。

部門マスター情報の印刷ですから、基本的には最初に「簡単な単票形式データ入力プログラム」として作成した TEST1 プログラムのフォームに印刷開始を指示するボタンを付加すれば、そのまま使用できると思います。

ということで、そういうイメージのテストプログラム TEST6 を作成しましょう。
新規プログラム作成で、プログラム名を TEST6 とし、フォーム複写で、TEST1 の Form1 を複写してください。

そして、印刷を実行させるためのボタン（cmd_PRINT）を配置します。
「情報選択ツリーパネル」->「コントロール配置（F3)」->「コントロール」から **Button** を選択し、フォームの OK ボタン（cmd_OK）の上あたりにでも配置してください。
配置済みのボタンを選択しプロパティパネルの**「コントロール名」**を **cmd_PRINT**、**「テキスト」**を**印刷**に設定します。



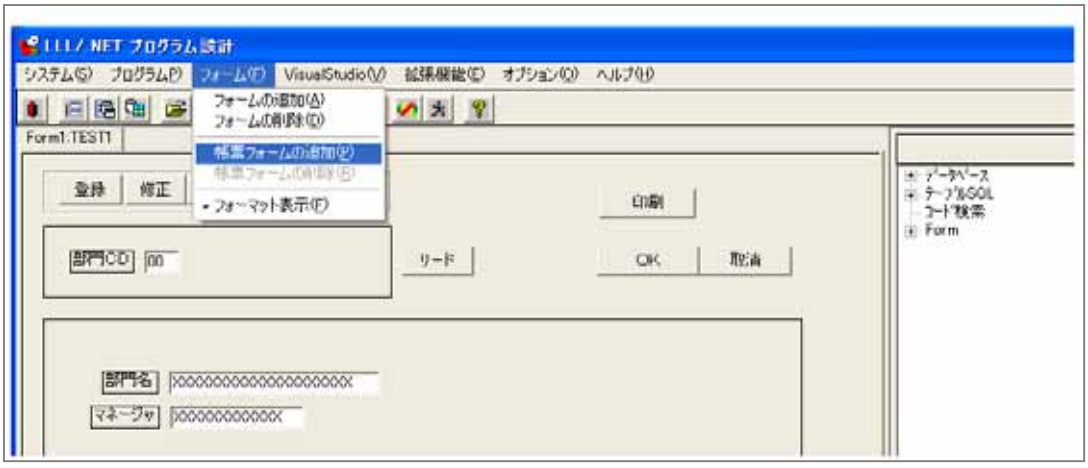


次にこのボタンを LLL/.net の制御対象に加えるため、**「情報選択ツリーパネル」->「フォーム制御（F２）」->「Form」->「LDATA」**を選び**「追加」->「フィールド」**を行います。

追加したフィールドのプロパティ**「フィールド名」**を **cmd_PRINT** に変更します。
これで呼び出し元となるフォームのデザインは完了です。

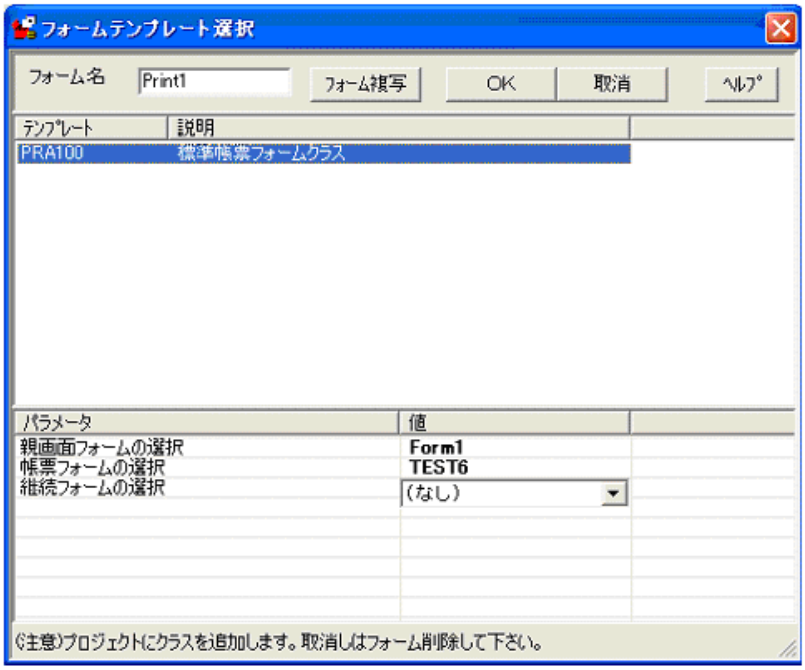
そして、このフォームプログラム TEST6 に対し、使用する帳票フォームクラスを追加します。

LLL/.NET プログラム設計->「フォーム（F)」->「帳票フォームの追加(P)」を選びフォームテンプレート選択ダイアログを表示させます。



LLL/.net ご購入時に最初から添付される帳票フォームテンプレートは１つしかありません。したがってこの画面には１つのテンプレートしか表示されません。

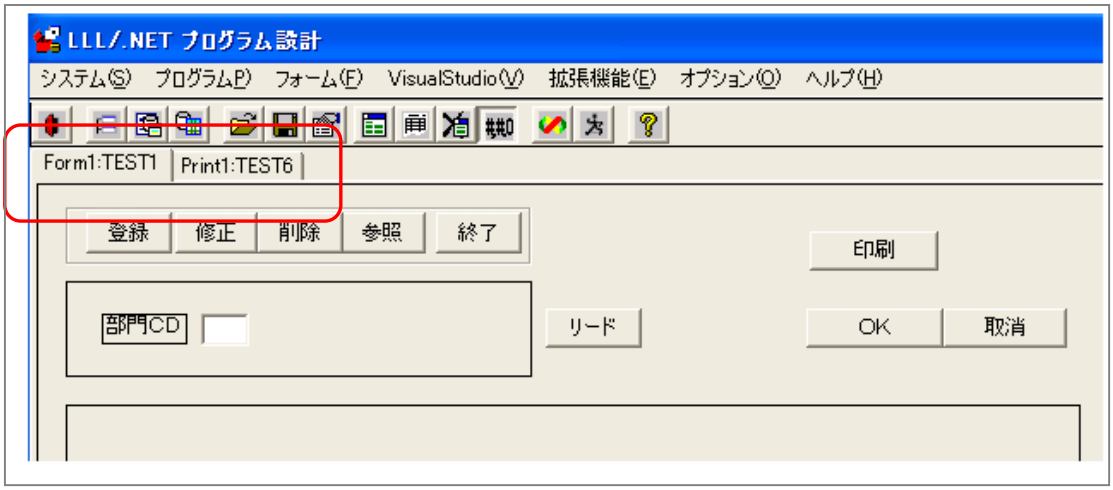
もちろんこれもフォームテンプレート同様、御自分で追加していただくことは可能です。



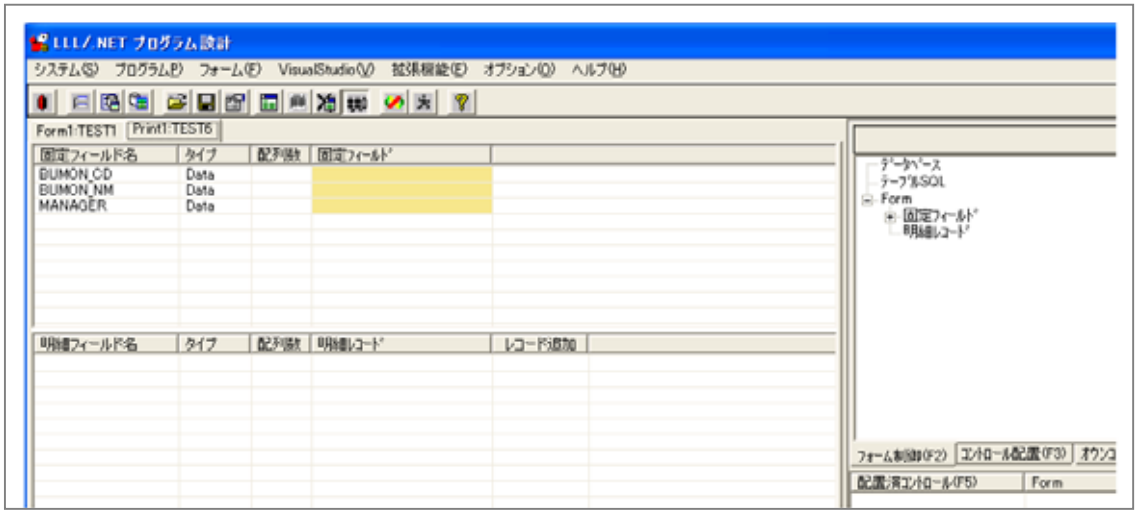
今は、唯一の選択肢である **PRA100 標準帳票フォームクラス** を選択してください。フォーム名は、デフォルトの **Print1** のままで構いません。

パラメータは、**親画面フォームの選択 = Form1**、**帳票フォームの選択 = TEST6**、**継続フォームの選択 = (なし)**、と設定してください。

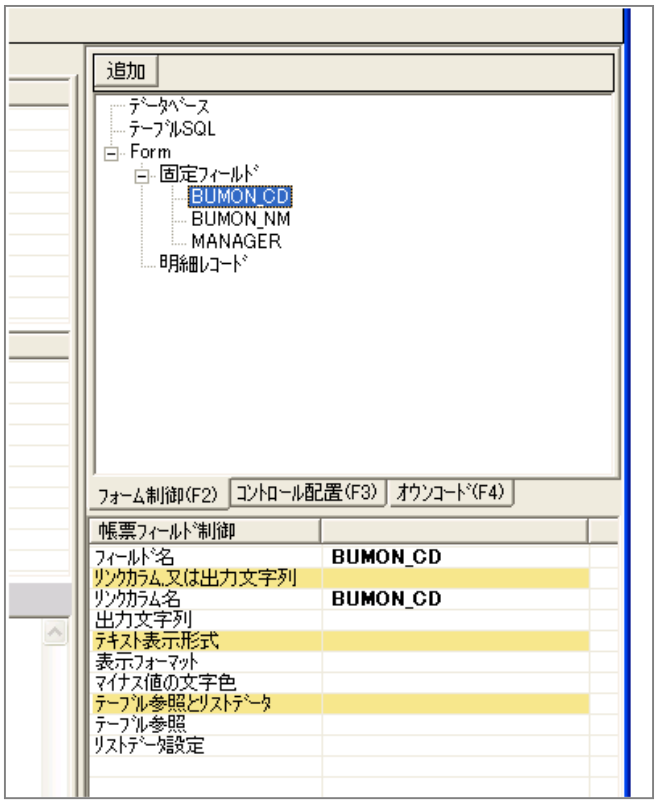
これで **OK ボタン**を押します。



今までフォームデザインタブページには、**Form1:TEST1** というタブしか存在しませんでしたが、その隣に **Print1:TEST6** というタブが追加されました。早速中を見てみたいと思います。**Print1:TEST6** タブをクリックしてください。下のようなページが表示されます。



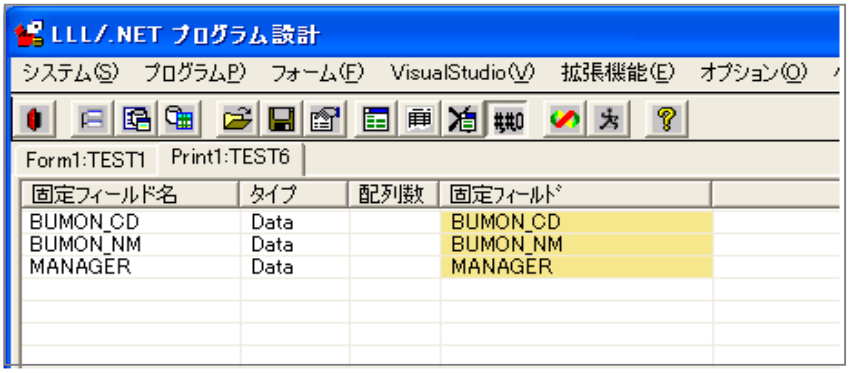
これは、帳票フォームデザインパネルです。
先に帳票フォームエディターで作成したフォーム上に配置したデータフィールドの名前が固定フィールド名として一覧表示されています。今回の帳票フォームには、固定フィールドしか存在しませんが、伝票フォームなどを作成した場合には明細フィールド名も表示されることになります。帳票フォームから自動的に読み取れる情報はここまでです。このパネルを使用して設定をさらに追加していきます。



固定フィールドに対して表示（印刷）させたい、値を指定します。今回は、画面上に存在する同名のフィールドの値を取り込んで印刷させたいので固定フィールドのプロパティで「リンクカラム名」を指定します。

フォーム制御（Ｆ２）情報ツーリビューから目的のフィールドを選び、プロパティ「リンクカラム名」をクリックすると選択候補として画面フォーム上のフィールド名がドロップダウンリストで一覧表示されますので、同名のフィールドを選びます。BUMON_CD には BUMON_CD、BUMON_NM には BUMON_NM、MANAGER には MANAGER、とそれぞれ指定してください。

この指定が完了すると、フォームデザインパネルの「固定フィールド」欄に設定した値が表示されます。



今回は使用しませんが、テーブル参照も使えますのでテーブルそのものや、帳票用のテーブル SQL を作成してそのカラムの値を印刷させたりすることを指定することもできます。
このあたりの機能の詳細については、マニュアルの**「システムの操作」->「帳票フォーム設計（帳票設計オプション)」->「帳票フォームの設計操作」**をご参照ください。

では、呼び出し元の Form1 に、「印刷」ボタン押下で印刷を行うコーディングを行ないます。
LLL/.NET プログラム設計->「VisualStudio(V)」->「VisualStudio 起動(E)」で、VisualStudio を起動します。
先ほど、画面フォーム上で「印刷」ボタン（cmd_PRINT）を配置し、LLL/.net 制御対象となるフィールドとして登録するところまでを行なっていますので、実際にボタンが押された後の処理をコードで記述すれば完成です。追加するコードは、Form1.VB 内の

```
'(owncode) コントロールクリック処理
Private Sub ControlClicked(ByVal controlName As String)
    Select Case controlName
```

に対し、印刷ボタン(cmd_PRINT)クリック時の処理として

```
            '印刷機能の呼び出し
        Case "cmd_PRINT"     ' 印刷ボタン
            PrintOut()
```

さらに、

```
'(owncode)その他
```

に、上記 **PrintOut()**の中身として、

```
Private Sub PrintOut()
    ' プリントオブジェクトPrint1のインスタンス化
    Dim pr As New Print1(Me.LForm)
    ' PrintOutメソッドの.Previewモード実行
    pr.PrintOut(LLLNET.LComPrn.LPrnMode.Preview)
End Sub
```

これだけです。後は細かいことですがこのまま実行すると、キー入力後に「リード」ボタンではなく、「印刷」ボタンにいきなりフォーカスされたりするので、「OK」や「取消」ボタンと同じタイミングで VisibleOn/Off の操作をしてあげれば良いと思います。

```
'(owncode) 各入力モードの設定
Private Sub SetInputMode(ByVal inputMode As String)
    Select Case inputMode
        Case "キー入力"          ' キー入力モード
```

の部分に対し、

```
            LForm.VisibleOff("cmd_PRINT")
```

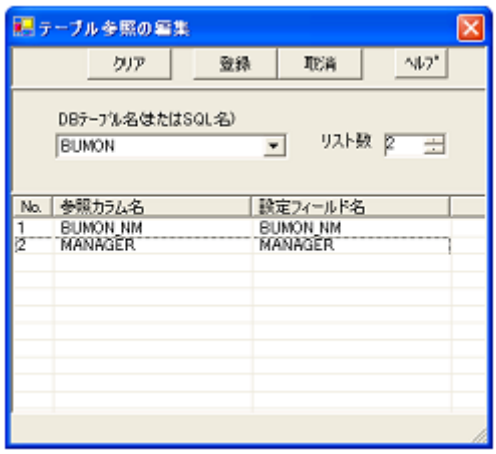
そして、

```
        Case "データ入力"          ' データ入力モード
```

の部分に対し、

```
            LForm.VisibleOn("cmd_PRINT")
```

を、それぞれ"cmd_CAN"に対する指示の直後あたりに追加してください。



これで、「印刷」ボタン押下時点の画面データをプレビュー表示させるプログラムができました。

今回は、帳票フォーム上の全フィールドに対し、「リンクカラム名」として画面フィールドを指定しましたが、キーを除く BUMON_NM、MANAGER、を、BUMON_CD の「テーブル参照」設定で左図のようにテーブルカラムを指定することにより、その時点の画面データではなくテーブルカラムデータをプレビューさせることもできます。

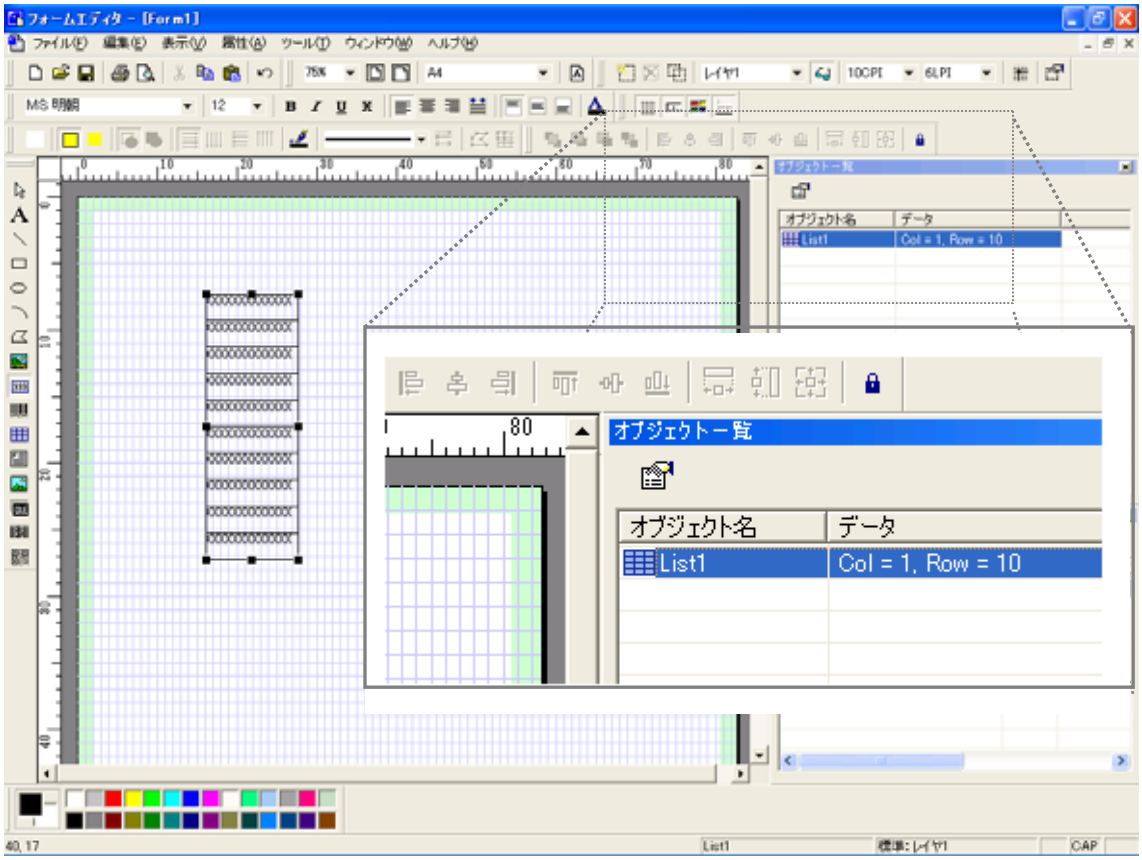
＜合計明細伝票印刷プログラムの作成＞

（テストプログラムの作成　その７）

次に明細フィールドを持つ、伝票型帳票の印刷プログラムを作りましょう。
さらに TEST6 のように単純に画面上のデータを印刷するのではなく、DB テーブル上のデータを集計するなど加工しながら印刷するタイプのプログラムにします。

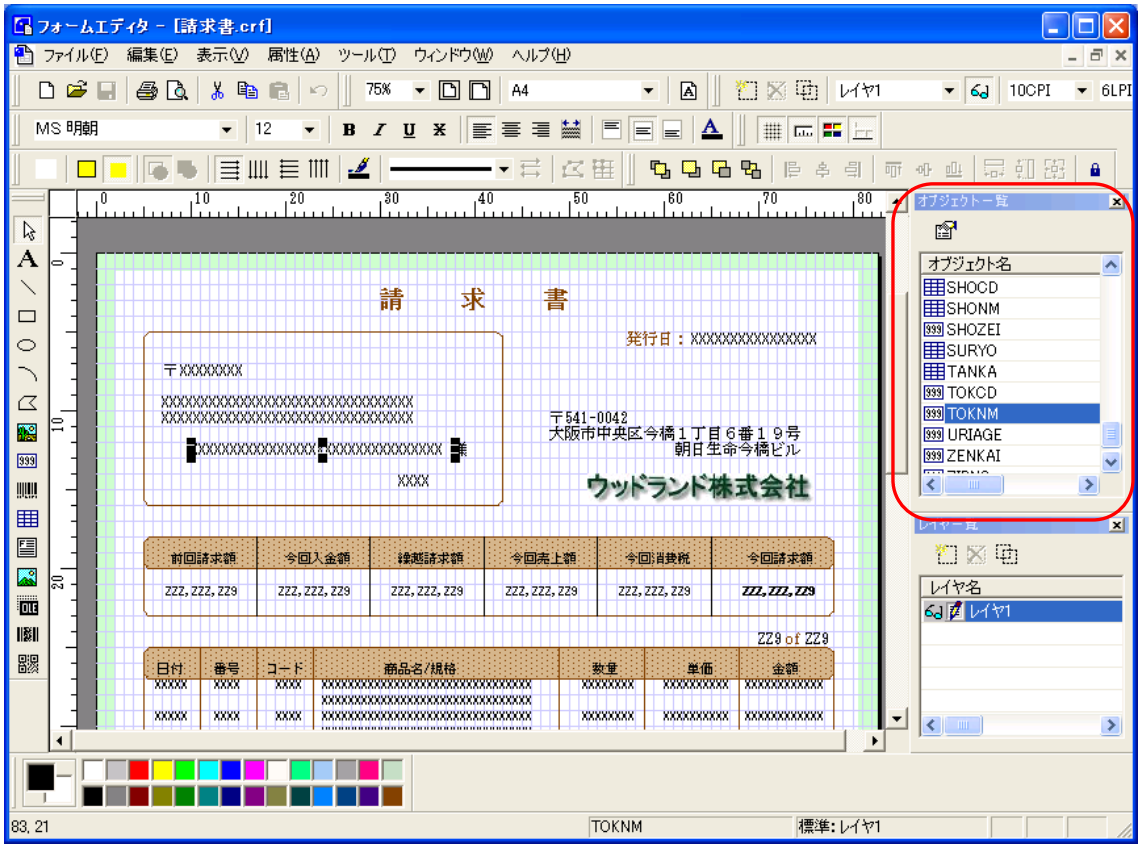
帳票フォーム上の明細フィールドは、帳票フォームエディターのツールボックスから「リストフィールド」を選択して作成します。

明細フィールドのひとつのカラム（Col=1）をひとつのリストフィールドとして定義し、必要な行数（Row）をフォーム上で貼り付けたコントロールをドラッグして定義します。



「商品コード」、「商品名」、「数量」、「単価」、「金額」という５つのカラムで構成される明細行を定義する場合は、「リストフィールド」は５つ作成することになります。
固定フィールドや、データフィールドの作成方法は、前述のテストプログラム（TEST6）での説明と同じです。

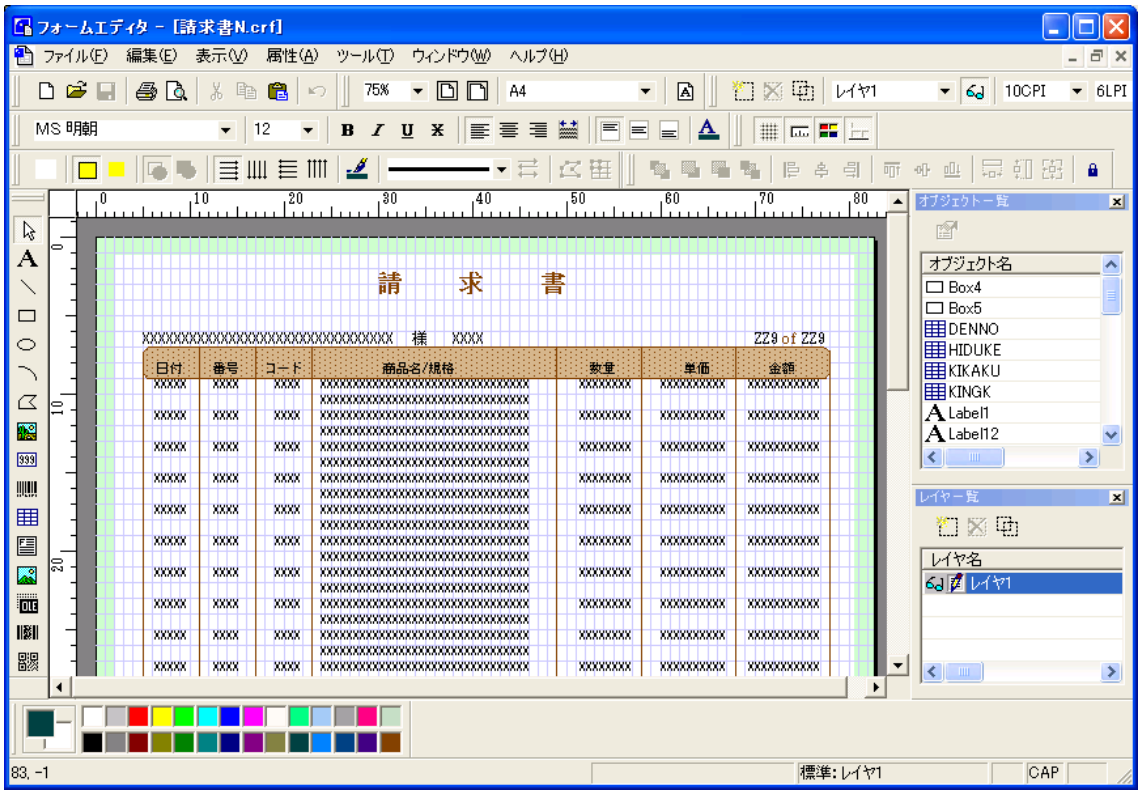
このようにして定義した、作成済みの帳票フォームサンプルがサンプルディレクトリにありますので、今回はそれを使用します。フォームエディタのメニュー**「ファイル(F)」->「開く(O)」** で「ファイルの場所」をサンプルの開発ディレクトリ「Hanbai_VB」(C#の方は「Hanbai_CS」)に合わせ、その中にある**「請求書.crf」**を選択して開いてください。
下記のように表示されます。



オブジェクト一覧に、フォーム上に配置されたオブジェクト名が表示されています。
リストフィールドに対しても、TEST6 で説明したようにオブジェクトプロパティの操作で名前を設定しています。

このサンプルは「合計明細請求書」と呼ばれるタイプの請求書で、合計請求書（鑑）と明細請求書から構成されています。便利なフォームですが明細が多い場合、2 枚（ページ）目以降にも合計請求欄があると無駄です。このような場合、2 枚目意向は明細請求書のみとしたいところですが、帳票プログラムからすれば一枚目と 2 枚目以降で帳票フォームを変化させなければならず結構厄介です。LLL/.net 帳票設計オプションでは、このような要求にも簡単に応えられます。

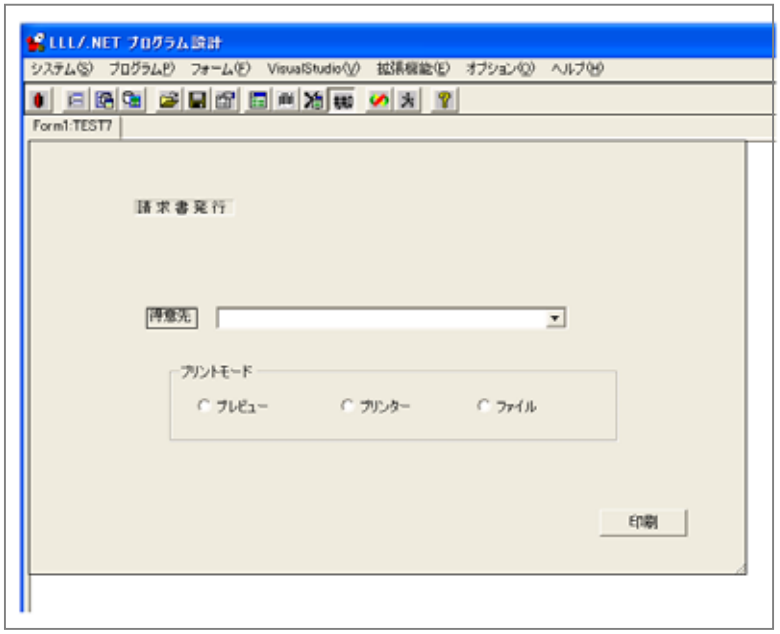
この場合、フォームそのものは 2 種類（１ページ目用と、２ページ目以降用）作成しておく必要があります。サンプルの**「請求書 N.crf」**がそのため（２ページ目以降用）のフォームサンプルです。今回はこれも使用します。



このように、必要なフォームファイル　**.crf**　が用意できましたら、この帳票を呼び出す親となるプログラムを用意します。

親プログラムは、請求書発行のメニュープログラムです。画面上で得意先を選んで、その得意先宛の請求書を発行するプログラムとします。
今回は、最終的には付属サンプルの「PRT010 帳票テスト１」の「２．請求書」だけに対応したようなテストプログラムを作成することになります。

「新規プログラム」でプログラム名を TEST7 とし、フォームテンプレートは、「PAA100　フリーフォーマット」を選択し、フォームタイトル = TEST7、データベースの選択 = DbMain と設定し「OK」ボタンを押してスタートしてください。
フォームのデザインは、あまり凝っていただく必要はありません。
得意先を選択するドロップダウンリストと、印刷モードを選択するラジオボタン、それに実際の印刷を指示するプッシュボタンがあれば十分でしょう。

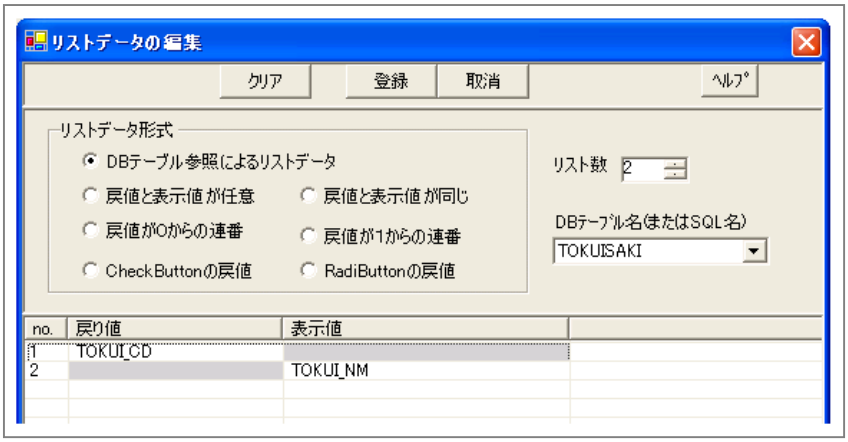


テーブル操作ですが、このフォームプログラムでは、テーブルの更新などは行いません。
得意先の選択で得意先の一覧表示をするために、得意先マスターテーブル（TOKUISAKI）の参照だけを行ないます。

「情報選択ツリーパネル」->「コントロール配置（F3)」でコントロールとしてComboBoxを選んだ上で、**「テーブル情報」->「TOKUISAKI」->「TOKUI_CD(得意先コード)」**をドラッグしてフォームに貼り付けてください。

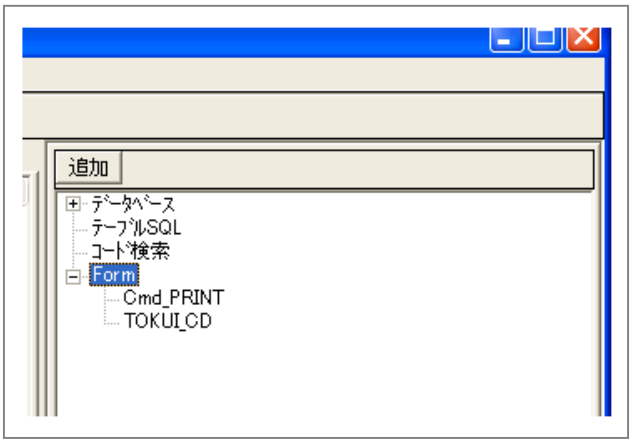
ドラッグしたカラムは得意先コードですが、実際の表示は得意先名にしたいのでラベルのテキストから「CD」を消しておきましょう。

「フォーム制御（F2)」->「Form」->「TOKUI_CD」を選び、プロパティの「リストデータ設定」を行ないます。



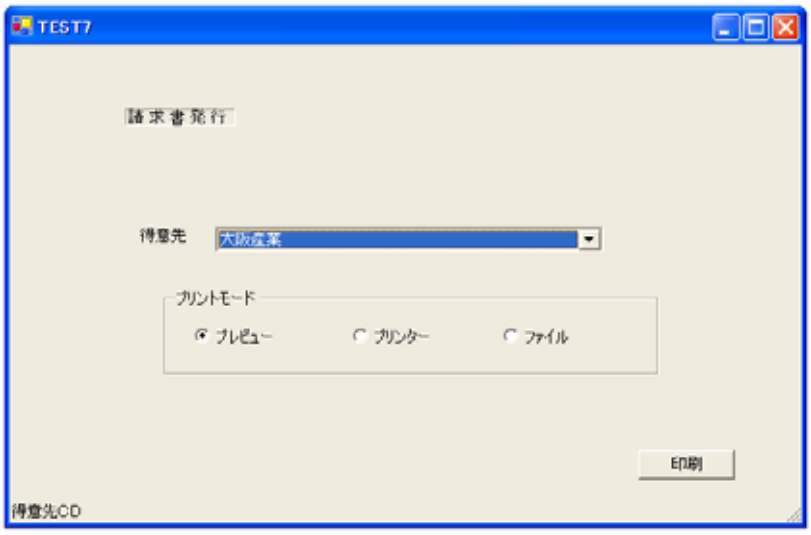
リストデータ形式は、「DBテーブル参照によるリストデータ」を選び、リスト数は２のまま、DBテーブル名（または SQL名）で TOKUISAKI を選択した上で、戻り値を TOKUI_CD(デフォルト)、表示値は TOKUI_NM と設定して「登録」します。
次に、印刷モードを選択できるようにラジオボタンを３つ作成します。
１グループしかありませんが、グループであることを明確にするためにグループボックスの中に作成しておきましょう。

グループボックス自体の操作は行ないませんので、名前は**GroupBox1**（デフォルト）のままで構いません。テキストプロパティを「プリントモード」とでもしておいてください。
今回のラジオボタンは、LLL/.net のフィールドである必要はありません。
３つ作成し、**Rad_Preview**、**Rad_Printer**、**Rad_File** とでも名前をつけておいてください。
それぞれのテキストプロパティを、「プレビュー」、「プリンター」、「ファイル」と設定し、後から VisualStudio で Rad_Preview の checked プロパティを True にしておいて下さい。



最後に、印刷を指示するプッシュボタンを１つ作成します。
名前を**Cmd_PRINT**、テキストを「印刷」と設定します。
これの押下は、フォームテンプレートのフォームイベントエントリで認識させますので LLL/.net のフィールドとして定義してください。

今回のフォームでは、Cmd_PRINT と、前述の TOKUI_CD だけが、LLL/.net のフォームイベントエントリのハンドリング対象であるフィールドになります。

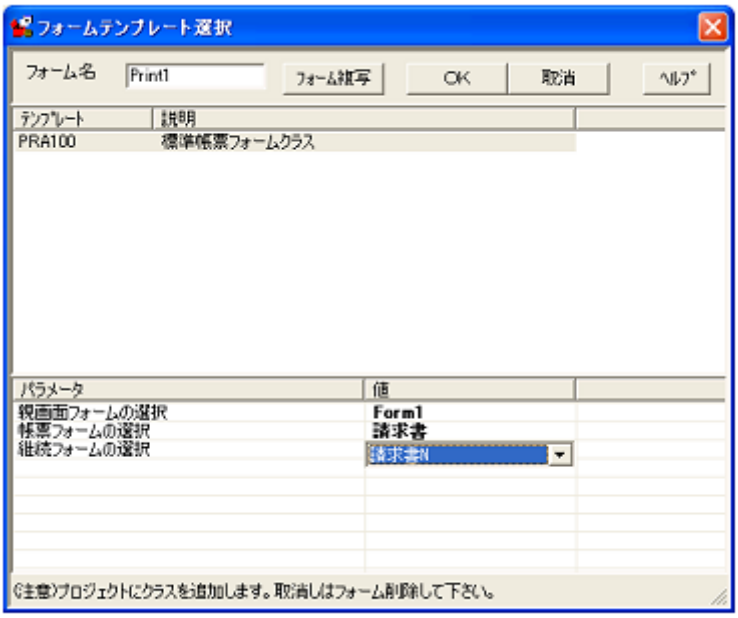


フォームの実行イメージは、こんな感じです。
まだ印刷はできませんが、得意先の表示と選択はすでに行えます。

ちなみに、「請求書発行」という表示は、単なるラベルで作成した見出しです。

次に、LLL/.net プログラム設計でこの画面フォームプログラムに対して、先ほどの帳票フォーム（請求書.crf、請求書N.crf）の追加を行います。
LLL/.net プログラム設計メニュー「フォーム」->「帳票フォームの追加」から、テンプレ

ートとして使用する「PR100 標準帳票フォームクラス」を選択して追加します。

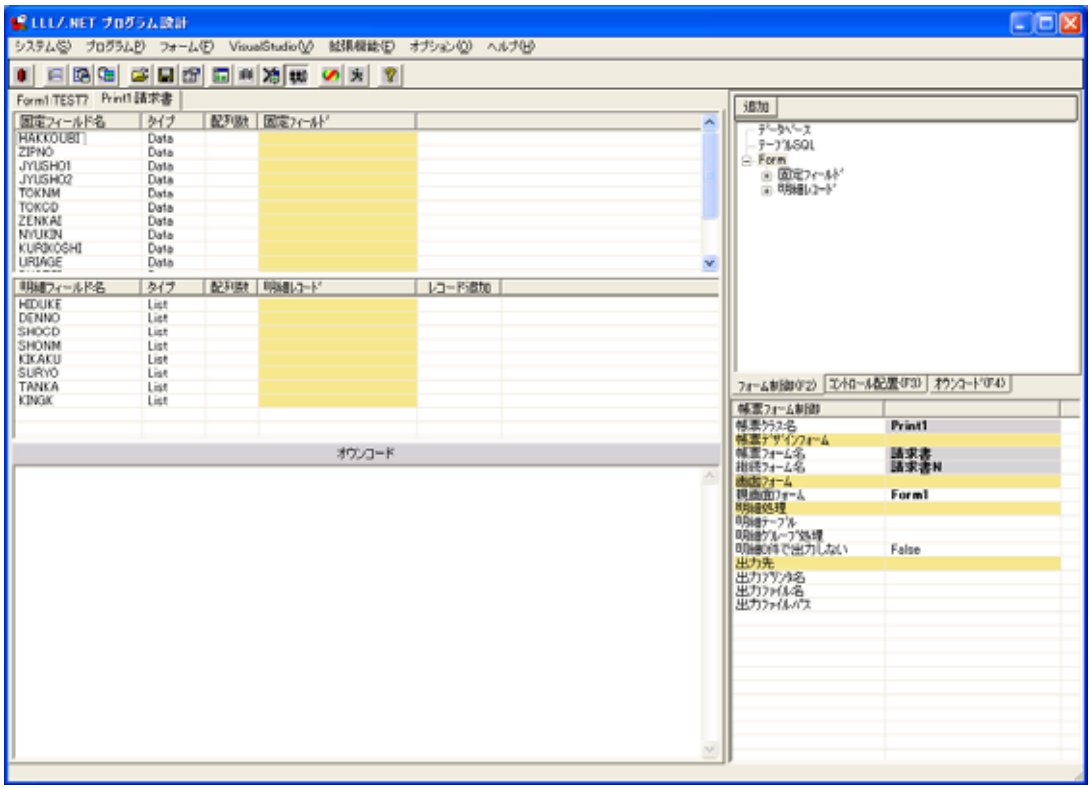


帳票フォームクラスのテンプレートパラメータに、親画面フォーム = Form1、帳票フォームの選択 = **請求書**、継続フォームの選択 = **請求書N**と設定して「OK」ボタンを押してください。
継続フォームとは、明細行が２ページ以上になる場合に、２ページ目以降に使用する帳票フォームです。
１ページ目と、２ページ目以降のフォームを変更できます。
そのような必要が無い場合は、「(なし)」と指定します。

フォームデザインパネルに、帳票フォームクラス「PRINT1:請求書」のタブが追加されました。次にこのパネルの設定を行ないます。
このタブページを表示させてください。今回使用する帳票フォーム（請求書.crf、請求書N.crf）には多くのフィールドが配置済みです。

今回は、継続フォームを使用しますので、帳票フォームファイルは、請求書.crf、請求書N.crf お２種類ですが、帳票フォームクラスは、あくまでも　Print1　１つです。
１つの帳票フォームクラスが、ページによって帳票フォームファイル（.crf）を使い分けるということです。

では、固定フィールドから設定していきます。TEST6 では、リンクカラム名に対応する画面フォーム上のフィールドを設定していきましたが、前述のとおり今回の画面フォームには得意先コードしかありません。



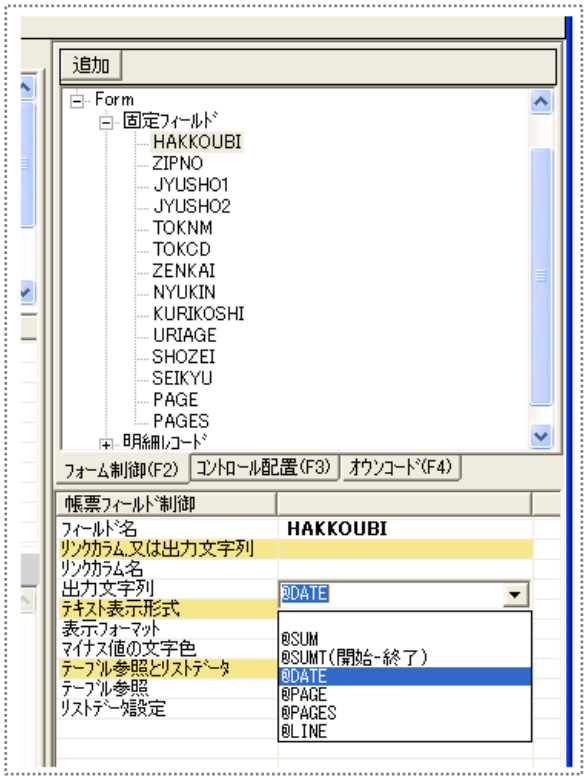
まずは、この得意先コード（TOKUCD）の「リンクカラム名」に画面フォームフィールドの得意先コード（TOKUI_CD）を指定して画面から取り込みます。
そしてこの TOKUI_CD をキーとして得意先マスターテーブル（TOKUISAKI）を参照し、その他の情報を設定していきます。

ZIPNO（郵便番号）以下の、JYUSYO1（住所１）、JYUSYO2（住所２）、TOKUNM(得意先名)、一つ飛ばして ZENKAI（前回請求額）が、得意先マスターテーブルから参照可能な項目です。（他にも TOKUISAKI には、当月売上や当月入金というカラムがありますが、値が入っていないので使えません）
「テーブル参照」プロパティを選択し、「テーブル参照の編集」ダイアログが表示させます。

そこで、DB テーブル名（または SQL 名）で「TOKUISAKI」テーブルを選択し、リスト数を「５」に増やした上で、参照カラム YUBIN_NO、JYUSYO1、JYUSYO2、TOKUI_NM、TUKI_ZANGK とそれぞれの設定フィールド名を ZIPNO、JYUSYO1、JYUSYO2、TOKUNM、ZENKAI というふうに設定して「登録」します。参照カラムがテーブルカラム、設定フィールドはここでは帳票フォーム上のフィールドになります。

一旦登録して「テーブル参照の編集」ダイアログを再度呼び出すと、リスト数は追加入力が行いやすいように自動的に現在登録済みのリスト数＋１、つまり「６」となっています。

では次に、HAKKOUBI（発行日）を設定しましょう。これはフォームからのオペレータの手入力や、DB テーブルカラム参照によらず、PC のシステム日付を使用したいとします。



そのような場合にも便利な機能があります。帳票フィールド制御プロパティの、「出力文字列」を使用します。TEST6 で使用した「リンクカラム名」は空白のままとし、プロパティリストの「出力文字列」を選択してください。ここで使用できる LLL/.net のマクロコマンドが一覧表示されます。
その中に、**@DATE** というコマンドがありますのでこれを指定します。
これは、指定されたフィールドに PC のシステム日付を表示させるというコマンドです。

同様にマクロコマンドが使用できる項目として、現在ページと、総ページの項目があります。これは、フィールド PAGE と PAGES で、それぞれコマンド **@PAGE** と **@PAGES** が使用できますので、そのように設定してください。

他にもまだ固定フィールドが残っています。NYUKIN(今回入金額)、KURIKOSHI(繰越請求額)、URIAGE(今回売上額)、SHOZEI(今回消費税)、SEIKYU(今回請求額)です。これは請求書鑑の合計項目です。これらは TOKUISAKI(得意先マスター)にはカラムまたは値が存在しませんの

で、伝票明細を計算して算出する必要があります。これらについては後でそのためのオウンコーディングを行ないます。

先に、明細フィールドの定義を続けて行ないましょう。
明細フィールドは売上伝票の明細データそのものです。これはテーブルから読み込みます。
そのためには、帳票フォームクラスに**テーブル SQL** を作成して登録する必要があります。

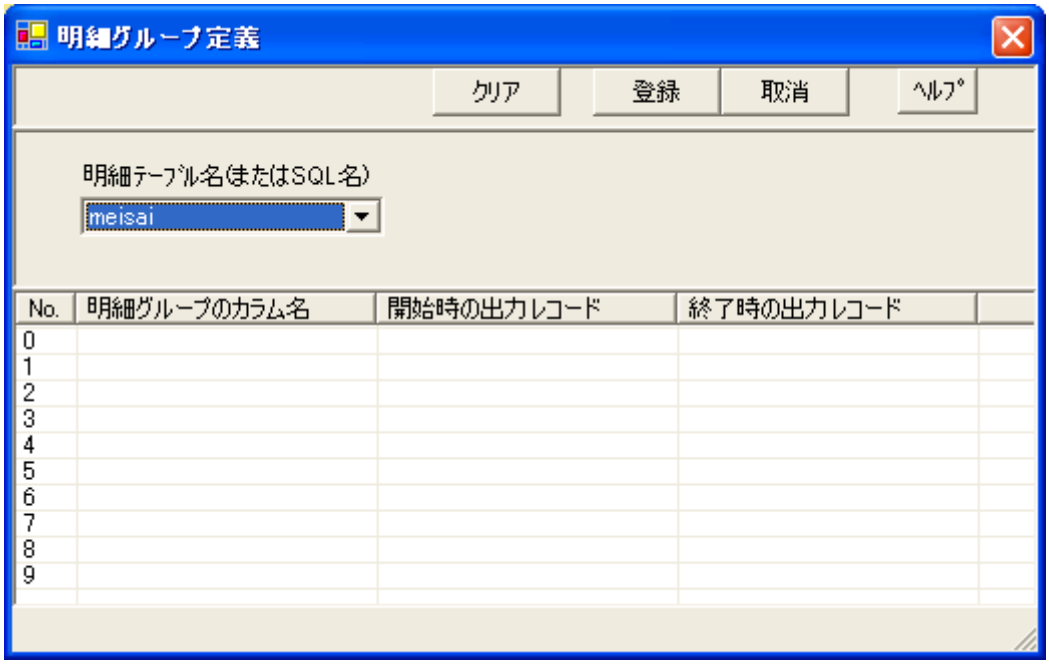
Print1:請求書タブページを開いた状態で、**「フォーム制御（F2)」->「テーブル SQL」**を選択し、「追加」ボタンを押し、**DENMEI**（伝票明細テーブル）を選択します。
Table1 というテーブル SQL が作成されますので、名前を **meisai** に変更しておきます。
伝票明細テーブルの構造は、LLL/.net データベース設計で確認しておいてください。

「SQL 文の編集」パネルを表示させ、「SQL 文を自動編集」にチェックした上で、画面フォーム上の TOKUI_CD をキーとして明細テーブル情報を取得し、取引日付、伝票区分、伝票番号、行番号の順でソートされるように、where 句と order by 句を追加します。

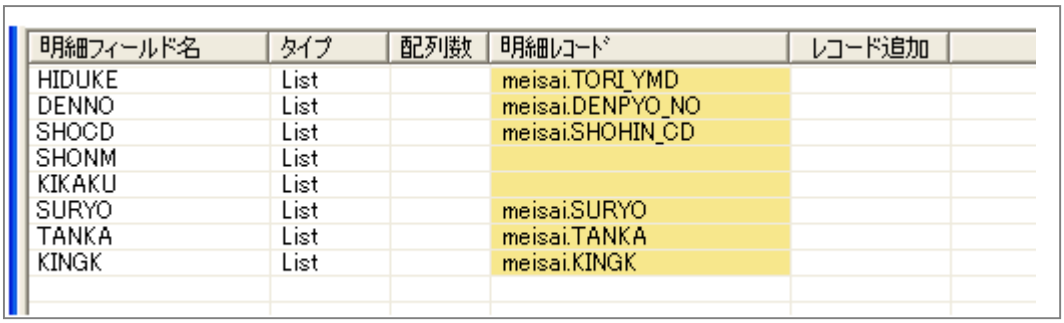
これで、「登録」します。

ちなみに情報選択ツリービュー先頭の「データベース」は、自動的に親である画面フォームプログラムの設定、すなわち **DbMain** が引き継がれていますので、ここで追加する必要はありません。

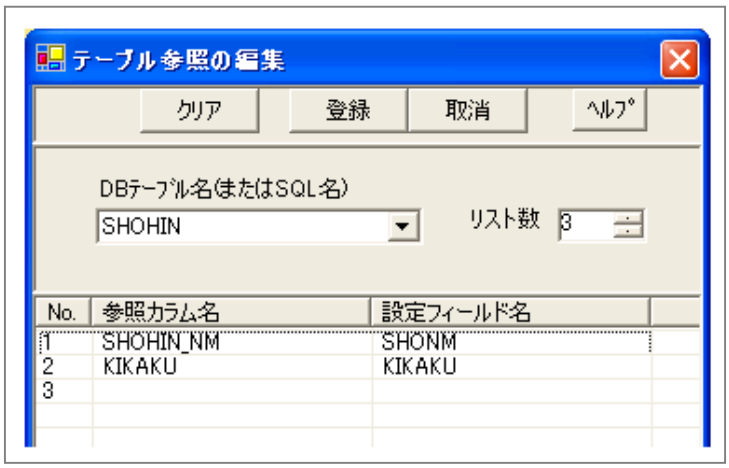
次に、「フォーム制御（F2)」->「Form」を選択して、これのプロパティリストの「明細テーブル」を選び、「明細グループ定義」パネルの「明細テーブル名（または SQL 名)」にテーブル SQL　**meisai** を指定して「登録」してください。



これで、テーブル DENMEI　の各項目を「リンクカラム名」に指定できるようになります。すなわち、画面上に存在しないテーブルデータの印刷指示が行えるようになります。



実際に設定してみましょう。SHONM(商品名)、KIKAKU(規格)以外は、すべて DENMEI テーブルが保持しているデータです。では SHONM、KIKAKU は、というとこれは SHOHIN テーブル（商



品マスタ）が保持していますので、先ほどの得意先情報と同様に、SHOCD をキーとして SHOHIN をテーブル参照することで簡単に設定できます。

DB テーブル名（または SQL 名）を SHOHIN とした上で、帳票フォーム上の SHONM、KIKAKU の両フィールドに対し、参照カラム名をそれぞれ SHOHIN_NM、KIKAKU と設定します。

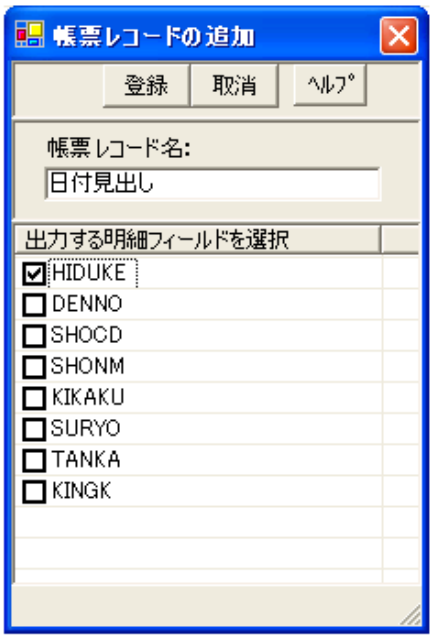
これで明細フィールド定義は出来上がりとしても良いのですが、これでは明細が単純に羅列されてしまい伝票毎の区切りも分かりません。また日付などは同じデータが繰り返し各行で印刷されてしまうことにもなります。

そこでまた便利な機能があります。「明細テーブルのカラムデータのブレイクによるグループ処理」がそれです。明細フィールドにおける見出しや合計の出力に使用できます。

マニュアル->「システムの操作」->「帳票フォーム設計（帳票設計オプション）」->「帳票フォームの設計操作」->「プロパティリスト」以下の各項目をご参照ください。

まずブレイク処理で出力させたいレコードを定義します。

帳票フォームデザインパネルの明細明細フィールド定義リストビューの「明細レコード」カラムの右隣に「レコード追加」というカラムがありますので、これをクリックしてください。

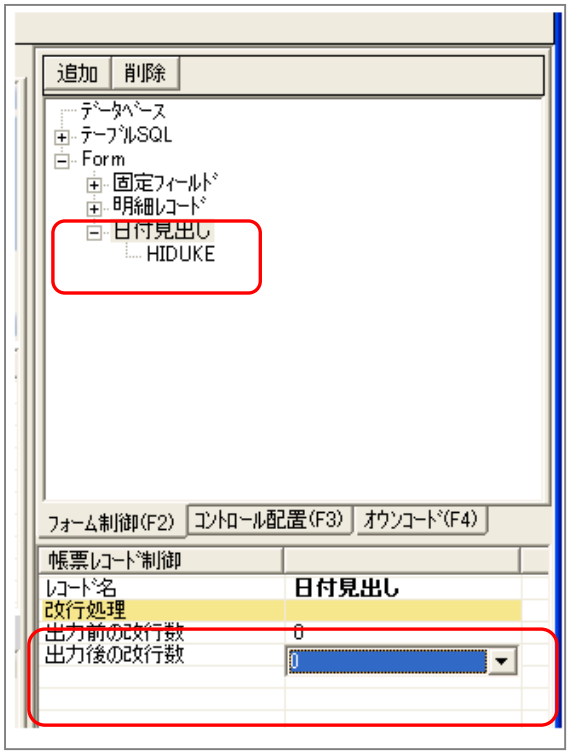


すると、「帳票レコードの追加」というパネルが表示されます。そこでまず帳票レコードの名前を付けます。
ここでは、「日付見出し」としておきましょう。

そして「出力する明細フィールドを選択」に対し、HIDUKE（日付）をチェックしておきます。

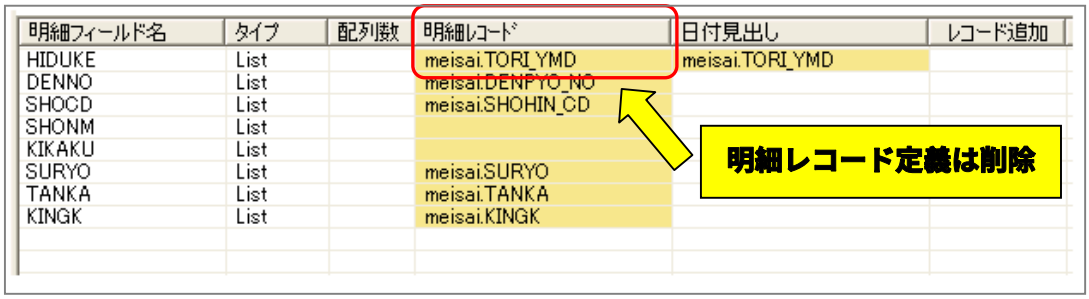
これは、日付が変化したら出力されるレコードとして HIDUKE を出そうということです。

「登録」ボタンを押すと、情報選択ツリービューの Form の下に HIDUKE フィールドという子アイテムを持つ「日付見出し」というレコードが追加されています。



レコードの追加は、この方法の他、情報選択ツリービューで **Form** を選択した状態で「追加」ボタンを押して一旦空のﾚｺｰﾄﾞを作成し、さらに追加されたレコードを選択した状態で「追加」ボタンを押してフィールドを追加作成するという方法でも実現できます。
日付見出しのプロパティを見ると、デフォルトで「出力後の改行数」は「1」となっていますが、今回は改行すると間延びしますので、値を「0」に設定しなおします。

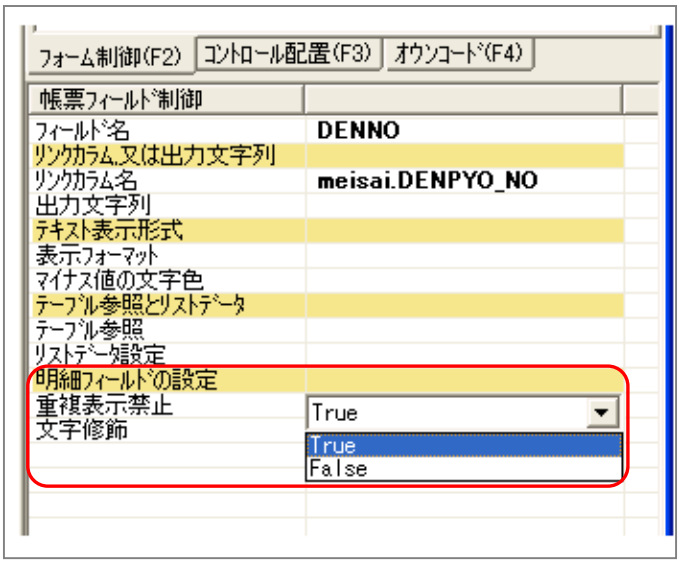
次にツリービューで HIDUKE を選択し、これのプロパティでリンクカラム名に「meisai.TORI_YMD」を指定します。
ついでに表示フォーマットも設定しておきましょう。**MM/dd** と指定すると、年は表示されません。
フォーマットは画面フォームでの設定と同じく.NET Framework のカスタム書式指定文字列の形式で設定できます。詳細は、マニュアル->「システムの操作」->「プログラム設計」->「プロパティ編集画面」->「表示フォーマットの編集」をご参照ください。



次に、今の状態では HIDUKE フィールドは、この「日付見出し」レコードと「明細レコード」の２つのレコードで重複定義されていますので、明細レコードの「リンクカラム名」の設定は削除してください。

これで、日付は変化が起こらない限り印刷されないことになります。

そういう意味では、次の伝票番号も同じ番号が続いて印刷される可能性があります。



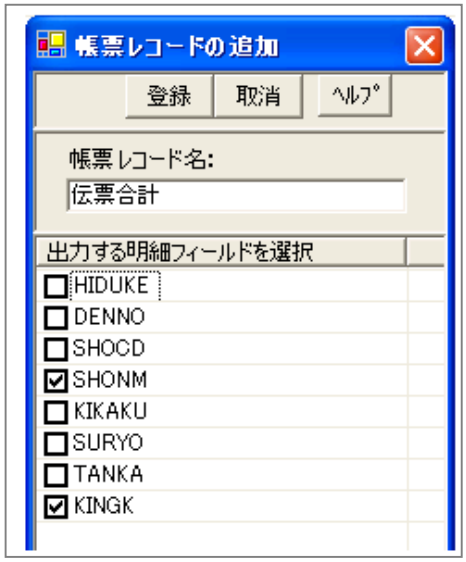
このような重複印刷を避ける方法としては、もう一つの方法があります。

帳票フィールド制御プロパティの**「重複表示の禁止」**を **True** に設定するという方法です。（デフォルトは False）これを設定してください。

これで、伝票番号も新しい値に変わらない限り印刷されないという定義がなされたことになります。

続いて、もうひとつブレイクによるグループ処理として伝票合計を明細中に印字する定義を行いたいと思います。

再度「レコード追加」ボタンを押し「帳票レコードの追加」ダイアログを表示させます。



今回は、「伝票合計」という名前のレコードとしましょう。そして SHONM(商品名)欄に「伝票合計」という見出し、KINGK(金額)欄に伝票の合計金額を印刷させたいと思いますので、SHONM、と KINGK にチェックを付けて登録します。

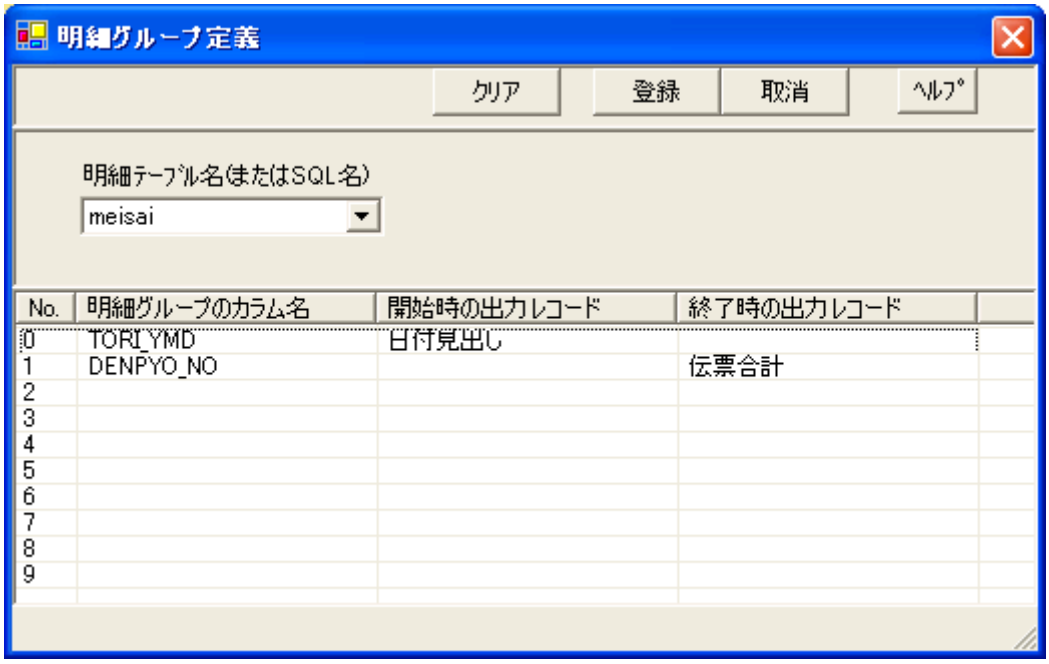
そして追加された SHOHIN を情報選択ツリービューで選択し、プロパティ出力文字列を「【伝票合計】」、文字修飾を横倍角「**w**」に設定します。KINGK に対しては、出力文字列にマクロコマンド **@SUM** を指定してください。
自動的に、伝票の金額欄が集計されて合計として印字されます。また、表示フォーマットは、**>###,###,##0** とし、マイナス値の文字色を **red**、文字修飾を太字下線「**bu**」としておきましょう。

その他の明細上の全ての数値フィールドにも表示フォーマット設定を行なってください。
ちなみに、表示フォーマットの指定は、ここで行なう方法と帳票フォームエディターのフィールドオブジェクトプロパティで行う方法があります。添付サンプルの帳票フォームでも両方のやり方での例がありますのでご参照ください。

この表示フォーマットや、前述の、文字修飾、出力文字列で使用可能なマクロコマンドの説明など**帳票フィールド情報プロパティ**の詳細については、**マニュアル->「システムの操作」->「帳票フォーム設計（帳票設計オプション)」->「帳票フォームの設計操作」->「プロパティリスト」->「フィールド情報のプロパティ」**をご参照ください。

次に作成した帳票レコードを、明細テーブルのどのカラムデータグループのブレイクによるグループ処理として、開始、終了のどちらのタイミングで使用するかを定義します。

「フォーム制御（F2)」->Form　のプロパティ「明細グループ処理」を設定してください。



明細グループ定義画面が表示されます。これは先ほど テーブル SQL　**meisai** を明細テーブルに指定した時に表示されたのと同じ画面です。
したがって「明細テーブル名（または SQL 名）」にはすでに **meisai** と入力されています。

読み込んだ　**meisai** のどのカラムをグループとして扱い、そのカラム値のブレイク開始時、と終了時にどの**帳票レコード**を出力するかを指定します。
今回は、TORI_YMD(取引日付)の開始時に、日付見出しを出力し、DENPYO_NO(伝票番号)の終了時に伝票合計を出力するという設定になります。
ちなみに、この明細グループの定義を行なったことで、オウンコードのグループ開始処理とグループ終了処理のイベントが発生し、必要なオウンコードを記述することが可能になります。

この機能の詳細については、**マニュアル->「システムの操作」->「帳票フォーム設計（帳票設計オプション)」->「帳票フォームの設計操作」->「プロパティリスト」->「明細グループ処理の設定」**をご参照ください。

これで、帳票のフォームに対する設定は終わりました。
つづいて帳票フォームクラスにオウンコーディングを追加し、ロジックを完成させていきます。

ここで、帳票フォームクラスのコーディングについて簡単に説明します。
帳票フォームクラスは、テンプレートとして提供します。
LLL/.net ご購入時に最初から付属するテンプレートは、**PRA100　標準帳票フォームクラス**という一種類のみです。

帳票フォームクラスは、以下のような構成になっています。

```
------------------------------------------------------------------------
Imports LLLNET.LComWin      ' フォーム制御コンポーネント
Imports LLLNET.LComDb       ' ＤＢ制御コンポーネント
Imports LLLNET.LComPrn      ' 帳票制御コンポーネント

Public Class Print1
    Inherits LPrnForm

    ' コンストラクタ（引数:親画面フォームのオブジェクト）
    Public Sub New(ByVal parentForm As LWinForm)
        MyBase.New(parentForm)
        LPrnForm_Initialize()  ' 帳票フォーム情報の初期設定
    End Sub

    #Region " LPrnForm 帳票フォーム情報"
        帳票フォーム制御パラメータ情報
    #End Region

    帳票フォーム処理コード

End Class
------------------------------------------------------------------------
```

先頭で帳票制御コンポーネント LLLNET.LcomPrn.dll を　Import し、帳票制御コンポーネントの LPrnForm クラスを継承してクラスを定義します。コンストラクタの引数として帳票出力を実行するフォーム制御オブジェクトが必要です。
#Region " LPrnForm 帳票フォーム情報"内は、画面プログラムの　**LWinForm フォーム制御パラメータ情報**　と同様、プログラム設計で登録した帳票情報を元に自動生成される部分ですので、通常は直接手で編集していただく必要はありません。
ここまでが前準備で、これに続く帳票フォーム処理コードがプログラムの中身になります。

PRA100 標準帳票フォームクラスの**帳票フォーム処理コード**は、以下の要素で構成されています。

1．変数定義領域：　処理コードで使用する変数を定義します。
2．帳票イベントエントリ
3．プリンタ OPEN・CLOSE
4．グループ開始・終了処理
5．明細処理
6．レコード出力前・出力後処理
7．ページ開始・終了処理
8．ページ出力処理

変数定義領域に続く「帳票イベントエントリ」は、必須であり削除できません。「帳票イベントエントリ」には、「帳票イベントオブジェクト(**LPrnEventArgs**)」が渡され、イベント種類やイベント名を取得することができます。帳票編集時の各種イベントを処理するために、イベント種類ごとのプロシージャを実行します。
帳票イベントエントリからコールするプロシージャには、必要に応じてオウンコードを追加します。オウンコードでは、「帳票フォームクラスのプロパティとメソッド」を使用することができます。

ここの詳細については、**マニュアル->「システムの操作」->「帳票フォーム設計（帳票設計オプション)」->「帳票フォームの設計操作」->「帳票フォームクラスの構成」**をご参照ください。

では、今回の請求書発行用のプログラム作成を続けましょう。
帳票フォーム上では、合計請求書（鑑）の ZENKAI(前回請求額)、以外の NYUKIN(今回入金額)、KURIKOSHI(繰越請求額)、URIAGE(今回売上額)、SHOZEI(今回消費税)、SEIKYU(今回請求額)の各項目が手付かずのフィールドとして残っています。

これらを算出するために、まず演算用項目を’(owncode)変数　に定義します。

```
'(owncode) 変数
' 合計演算用項目
Private g_NYUKIN As Decimal      ' 今回入金額
Private g_URIAGE As Decimal      ' 今回売上額
Private g_SHOZEI As Decimal      ' 今回消費税
```

次に明細レコード１件毎の出力前に行なう処理を’(owncode)明細処理　に記述します。

```
'(owncode) 明細処理
Private Sub LProcDetail()
    '--------------------------------
    Dim KINGK As Decimal = Me.DetailRow("KINGK")
    '
    If Me.DetailRow("DENPYO_KBN") = "1" Then        ' 売上
        If Me.DetailRow("SHOHIN_CD") = "9999" Then
            g_SHOZEI += KINGK       ' 今回消費税
        Else
            g_URIAGE += KINGK       ' 今回売上額
        End If
    Else                                            ' 入金
        g_NYUKIN += KINGK           ' 今回入金額
        Select Case Me.DetailRow("TORI_KBN")
            Case "1"
                Me.SetData("SHONM", "入金 現金")
            Case Else
                Me.SetData("SHONM", "入金 その他")
        End Select
        Me.SetData("SURYO", Nothing)
        Me.SetData("TANKA", Nothing)
    End If
    '--------------------------------
End Sub
```

明細レコードは、売上区分を持っており、１=売上、2=入金となっています。
売上データには、商品コード 9999(消費税)が含まれますので、これの集計を行ないます。
また、入金データは集計するとともに、取引区分によって、1 は、「入金 現金」、その他は「入金 その他」、という文字列を帳票フォームの　SHONM（商品名）に出力します。
入金に SURYO、TANKA はありませんので、値 Nothing　セットします。

そして、明細全体の終了時に行なう処理を ’(owncode)グループ終了処理 に記述します。
これはもちろん合計請求（鑑）の出力になります。

```
'(owncode) グループ終了処理
Private Sub LGroupEnd(ByVal groupName As String)
    '--------------------------------
    If groupName = "*" Then
        ' 前回請求額を取得
        Dim ZENKAI As Decimal = Decimal.Parse(Me.GetData("ZENKAI"))
        ' 今回入金額
        Me.SetData("NYUKIN", g_NYUKIN)
        ' 繰越請求額
        Me.SetData("KURIKOSHI", ZENKAI - g_NYUKIN)
        ' 今回売上額
        Me.SetData("URIAGE", g_URIAGE)
        ' 今回消費税
        Me.SetData("SHOZEI", g_SHOZEI)
        ' 今回請求額
        Me.SetData("SEIKYU", ZENKAI - g_NYUKIN + g_URIAGE + g_SHOZEI)
        ' 合計演算の初期化
        g_NYUKIN = 0
        g_URIAGE = 0
        g_SHOZEI = 0
    End If
    '--------------------------------
End Sub
```

グループ名として使用した　"*"　は、明細全体を指すLLL/.netの予約語です。
グループ終了処理で使用した場合は、明細全体の終了時を指します。

最後に細かい定義ですが、明細全体の開始時にページ番号の初期化が必要です。
@PAGE や **@PAGES**　を使用する場合には必要になります。

```
'(owncode) グループ開始処理
Private Sub LGroupStart(ByVal groupName As String)
    '--------------------------------
    If groupName = "*" Then
        Me.PageNumber = 1 ' ページ番号を初期化
    End If
    '--------------------------------
End Sub
```

以上で、帳票フォームクラスは完成です。
ちなみに、今回加えたオウンコーディングは、全てサンプル　**PRT010 帳票テスト１**　の「Print2：請求書」という帳票フォームクラスのオウンコーディングと同じです。
タイプが面倒であれば、「Print2：請求書」からコピー&ペーストしてください。

後は、画面フォームプログラムにこの帳票フォームクラスを呼び出すロジックを加えます。

追加すべき画面フォームプログラムのロジックは以下の通りです。

「印刷」ボタン（Cmd_Print）がクリックされたら、帳票フォームクラス（Print1）のプリントオブジェクトを作成し、その時点で選択されているプリントモードを指定して帳票出力メソッドを呼び出します。ただし、得意先コードが選択されていなかった場合は、実行しません。

ここまでが、画面フォームプログラム側の役割になります。

‘(owncode)コントロールクリック処理　に対し、「印刷」ボタンクリック時の処理としてオウンコードを記述します。

VisualStdio で Form1.vb のソースを開いてください。

下記のように記述します。

```
'(owncode) コントロールクリック処理
Private Sub ControlClicked(ByVal controlName As String)
```

```
    '--------------------------------
    ' コントロールのクリックイベントが発生したときの処理を追加
    Select Case controlName
        Case "Cmd_PRINT"

    If Me.TOKUI_CD.SelectedItem Is Nothing Then Exit Sub
    ' 帳票出力オブジェクトを作成
          Dim pr As Print1 = New Print1(Me.LForm)
    ' 帳票出力のメソッド実行
    pr.PrintOut(z_GetPrintMode())

    End Select
    '--------------------------------
End Sub
```

そして、プリントモード取得ロジックの本体を’(owncode)その他　に記述します。

```
'(owncode)その他

' プリントモードを取得
Private Function z_GetPrintMode() As LLLNET.LComPrn.LPrnMode
    '
    Dim mode As LLLNET.LComPrn.LPrnMode = LLLNET.LComPrn.LPrnMode.Print
    If Me.Rad_Preview.Checked Then
        mode = LLLNET.LComPrn.LPrnMode.Preview
    ElseIf Me.Rad_File.Checked Then
        mode = LLLNET.LComPrn.LPrnMode.File
    End If
    Return mode

End Function
```

これで完成です。「デバッグ」->「開始」または、ビルドして実行してみましょう。
大阪産業を選択し、プレビューで実行するとこのようなイメージになります。

プレビューツールバーで次ページを表示すると、ちゃんと合計明細の無い請求書Nのフォームで表示されます。

１３．おわりに

他の付属サンプルなども動作させ動きを確認しながら、今回使用しなかったコンポーネントのメソッドなどを実際に使ってみるプログラムなどを作ってみていただくと、更に理解が深まります。
テンプレートの構造が理解できれば、テンプレートを自作でき、自由にテンプレートの種類が増やせます。
自分のスタイルにあったテンプレートを追加していっていただくことで、更に LLL/.net の使い勝手を進化させていっていただくことができます。

ここではご紹介できなかった、または十分に説明できなかった LLL/.net の機能がまだまだあります。
それぞれに関するヒントは、サンプルやマニュアルの中にもありますので、よろしければ作成にチャレンジしてみてください。

LLL/.net を活用いただければ幸いです。

参考文献 （情報サイト）

○ マイクロソフト ＭＳＤＮ

スマートクライアントデベロッパーセンター

http://www.microsoft.com/japan/msdn/smartclient/

.NET Framework

http://www.microsoft.com/japan/msdn/netframework/

Visual Studio

http://www.microsoft.com/japan/msdn/vstudio/

ADO.NET 入門

http://www.microsoft.com/japan/msdn/sqlserver/sql2000/sql_adonetprimer.asp

○ @IT アットマークアイティ

特集　ファットからスマートへ進化する企業システムのクライアント

http://www.atmarkit.co.jp/fdotnet/special/smartcli/smartcli_01.html

解説　インサイド .NET Framework ［改訂版］

http://www.atmarkit.co.jp/fdotnet/technology/idnfw11_01/idnfw11_01_01.html

ADO.NET 基礎講座

http://www.atmarkit.co.jp/fdotnet/basics/adonet_index/index.html

実例で学ぶ ASP.NET プログラミング

（第 3 回「実プロ流」ASP.NET データベース操作術）

http://www.atmarkit.co.jp/fdotnet/aspexp/aspexp03/aspexp03_01.html

連載　改訂版 プロフェッショナル VB.NET プログラミング

http://www.atmarkit.co.jp/fdotnet/vb6tonet2/index/index.html

○ IT 用語辞典　e-Words

http://e-words.jp/

- 本書に記載しているプログラム名、機器名は、一般に各メーカーの（登録）商標です。
- 製品の仕様は予告無く変更される場合があります。

［発行履歴］

◆ 第一版　　2005 年 4 月 22 日　1 章～5 章

◆ 第二版第一刷　2005 年 6 月 24 日　6 章以降を追加、また全般に図や補足説明を強化するなど、大幅に加筆訂正。

◆ 第二版第二刷　2005 年 8 月 3 日　8 章テスト PG においてテーブル参照設定の説明が欠落している不具合他を修正。

◆ 第三版第一刷　2005 年 9 月 14 日　図表配置や文章表現を見直し全体のページ数を圧縮した上で、12 章として帳票設計オプションの解説とテストプログラム２本を追加。

◆ 第三版第二刷　2006 年 10 月 24 日　アセンディアへの社名変更対応

発行　2006 年 10 月 24 日
第三版　（第二刷）

株式会社アセンディア パーシモン事業部

〒541-0042　大阪市中央区今橋 1 丁目 6 番 19 号　カーニープレイス北浜
TEL ： 06-6204-0618 (事業部代表)　　FAX ： 06-6204-0626(事業部直通)
事業部メールアドレス ： persimmon@ascendia.jp
LLL/.net 公式ホームページ ：
http://www.per.woodland.co.jp/products_pages/lllnet_pages/lllnet_main.html