【セッションNo.2】

コンポーネント有効活用で開発効率向上！
～コンポーネント活用テクニックのご紹介～

株式会社ミガロ
システム事業部 システム1課　主任
小杉　智昭

アジェンダ

1）既存コンポーネントを利用する方法
- ① コンポーネントとは
- ② コンポーネントの配布形態

2）コンポーネントを拡張する方法
- ① 継承とカスタムコンポーネント
- ② コンポーネントの拡張手順

3）*ピックアップ！* MaskEditコンポーネントの拡張
- ① 拡張する機能の紹介
- ② MaskEditコンポーネントの拡張手順

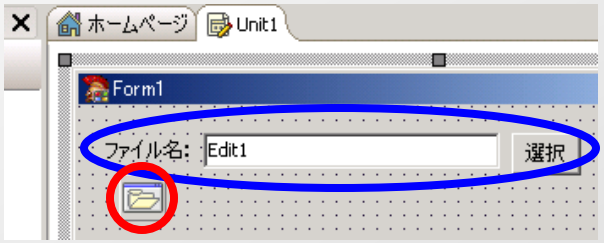
1）既存コンポーネントを利用する方法

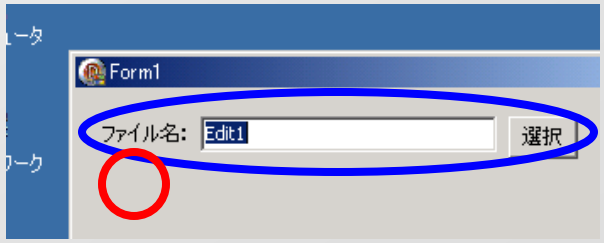
①コンポーネントとは

＜コンポーネントとは＞

・一般のソフトウェア開発の現場においては特定の機能を持ったプログラム用の部品
　－ 再利用性を考慮し汎用的な機能を持たせたプログラム
　　→ 他のプログラムと組み合わせて必要な機能を実現、ないしは追加する

・Delphiにおいてはオブジェクトの一種
　－ プログラミングする際に利用可能な「ビジュアル」なオブジェクト
　　・実行時も「ビジュアル」でユーザインターフェース要素となり得るもの
　　→ **ビジュアルコンポーネント**・・・TLabel、TEdit、TButton等
　　・設計時のみ「ビジュアル」でユーザインターフェース要素となり得ないもの
　　→ **非ビジュアルコンポーネント**・・・TDatabase、TOpenDialog、TTimer等





1）既存コンポーネントを利用する方法

①コンポーネントとは

＜コンポーネントを利用するメリット＞

・利用者側からのメリット
- 詳細な処理手順を知らなくても機能を利用できる
  → FTPの詳細を知らなくてもFTPコンポーネントを使えばファイル転送が使える、等
- コードの重複を避けることができる
  → コード記述の量を減らせる

・開発者側からのメリット
- コードの独立性が高い
  → 利用者側から不用意に内部動作を変更されない
- 再利用性が高い
  → 一度開発すれば大きな仕様変更がない限り利用し続けることができる

→ 作成したい内容に応じたコンポーネントを開発／利用することで、開発効率を劇的に向上させることができる

1）既存コンポーネントを利用する方法

②コンポーネントの配布形態

＜コンポーネントの配布形態＞

Ⅰ．バイナリ形式のパッケージ

主なファイル：

設計時パッケージ（dcp）、実行時パッケージ（bpl）、コンパイル済ユニット（dcu）

組込手順：

1．メニューの[コンポーネント|パッケージのインストール]を選択
2．「パッケージのインストール」ダイアログで追加ボタンを押下
3．対象の実行時パッケージファイルを選択

※配布元から導入手順が提示されていることが多いので、その場合は指示に従って導入して下さい。

注意事項：

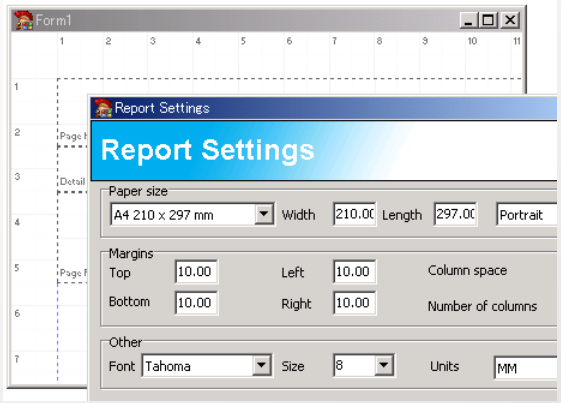
バージョン毎にほぼ専用のファイルが必要

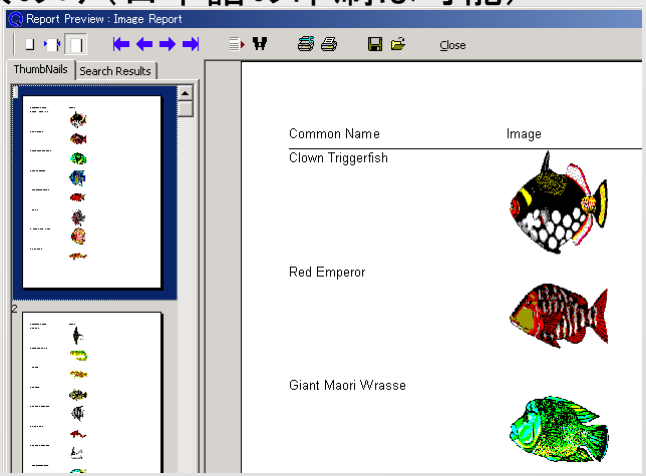
1）既存コンポーネントを利用する方法（操作例）

Ⅰ.バイナリ形式のパッケージ取込例（QuickReport）

※QuickReportとは．．．

- Delphiでほぼ標準のレポートツールとして長く添付されていたレポートツール
  → Delphi 6までは標準で登録、Delphi 7は追加登録可能
  Delphi 2006／2007はモジュールをwww.codegear.comより無償で入手可能
- Delphiで開発されているため、Delphiとの親和性が非常に高い
- ソースコードも付いた高機能版を有償で購入可能
- 現在入手可能なバージョンは英語版のみ（日本語の印刷は可能）





1）既存コンポーネントを利用する方法（操作例）

Ⅰ.バイナリ形式のパッケージ取込例（QuickReport）

エンバカデロ・テクノロジーズ（CodeGear）の以下のページから、QuickReport 4 Standard（QR4StdD2006W32Install.zip）を入手します。

http://cc.codegear.com/item/25002

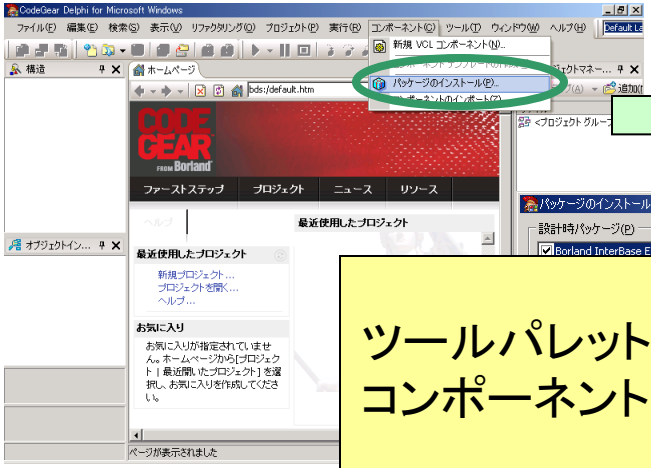
入手したQR4StdD2006W32Install.zipを展開し、QR4StdD2006W32.EXEを実行します。
実行後は画面指示に従って進めていきますが、インストール先の初期値はBDS 2006を前提にしたパスとなっているため、以下のパスに変更します。

C:¥Program Files¥Borland¥BDS¥4.0¥QRStandard
↓
C:¥Program Files¥CodeGear¥RAD Studio¥5.0¥QRStandard

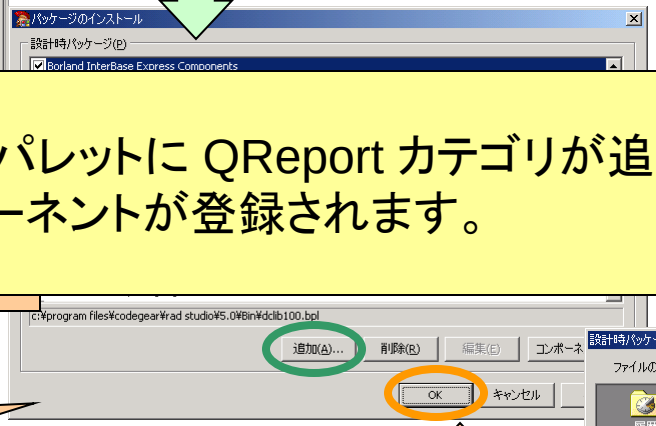
インストールが完了したらDelphi 2007を実行し、パッケージの登録を行います。（次頁）

1）既存コンポーネントを利用する方法（操作例）

Ｉ.バイナリ形式のパッケージ取込例（QuickReport）

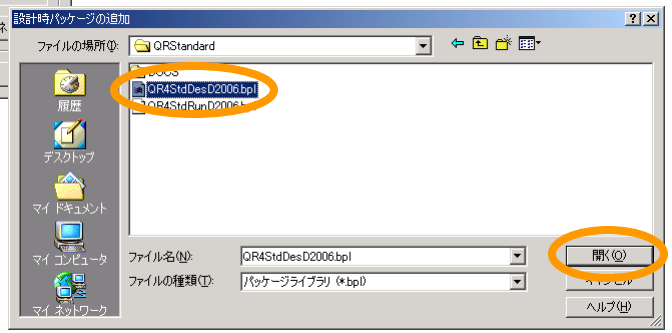


[コンポーネント|パッケージのインストール]を選択



追加ボタンを押下

ツールパレットに QReport カテゴリが追加され、
コンポーネントが登録されます。



OKボタンを押下

以下のファイルを選択
C:¥Program Files¥CodeGear¥RAD Studio¥5.0
¥QRStandard¥QR4StdDesD2006.bpl

1）既存コンポーネントを利用する方法

②コンポーネントの配布形態

＜コンポーネントの配布形態＞

Ⅱ．ソース形式のパッケージ

主なファイル：

パッケージソース（dpk）、ソースコード（pas）

組込手順：

1．メニューの[ファイル|プロジェクトを開く]を選択
2．「プロジェクトを開く」ダイアログで対象のパッケージソースを選択
3．プロジェクトマネージャ上でパッケージを右クリック
4．ポップアップメニューのインストールを選択

注意事項：

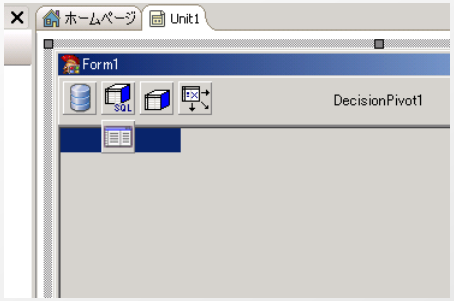
（必要に応じて）コンポーネントをコンパイル可能な環境を整える必要がある
バージョン依存する場合、修正が必要となることがある

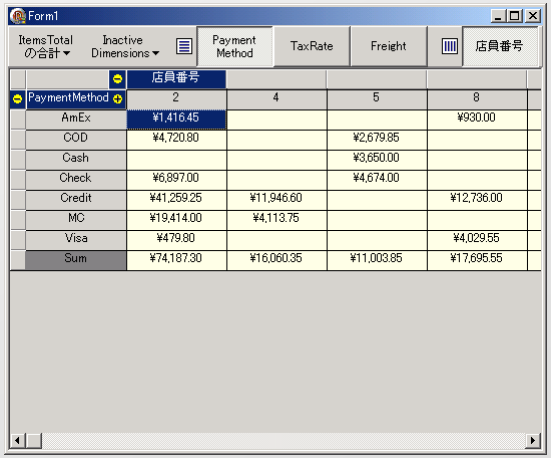
1）既存コンポーネントを利用する方法（操作例）

Ⅱ.ソース形式のパッケージ取込例（DecisionCube）

※DecisionCubeとは．．．

- 多次元解析のツールとして標準で添付されていたコンポーネント
  → Delphi 7までは標準で登録
  Delphi 2005以降はインストールされるが登録はされない
  Delphi 2007はインストールの際に一部修正が必要
- Delphiで開発されているため、Delphiとの親和性が非常に高い
- ソースコードが付属している





PaymentMethod	店員番号 2	4	5	8
AmEx	¥1,416.45			¥930.00
COD	¥4,720.80		¥2,679.85	
Cash			¥3,650.00	
Check	¥6,897.00		¥4,674.00	
Credit	¥41,259.25	¥11,946.60		¥12,736.00
MC	¥19,414.00	¥4,113.75		
Visa	¥479.80			¥4,029.55
Sum	¥74,187.30	¥16,060.35	¥11,003.85	¥17,695.55

1）既存コンポーネントを利用する方法（操作例）

Ⅱ.ソース形式のパッケージ取込例（DecisionCube）

DecisionCubeはやや古いコンポーネントで統合開発環境と共にインストールされますが、ツールパレットには登録されません。
2種類のパッケージで構成されていて、それぞれのパッケージソースを利用します。

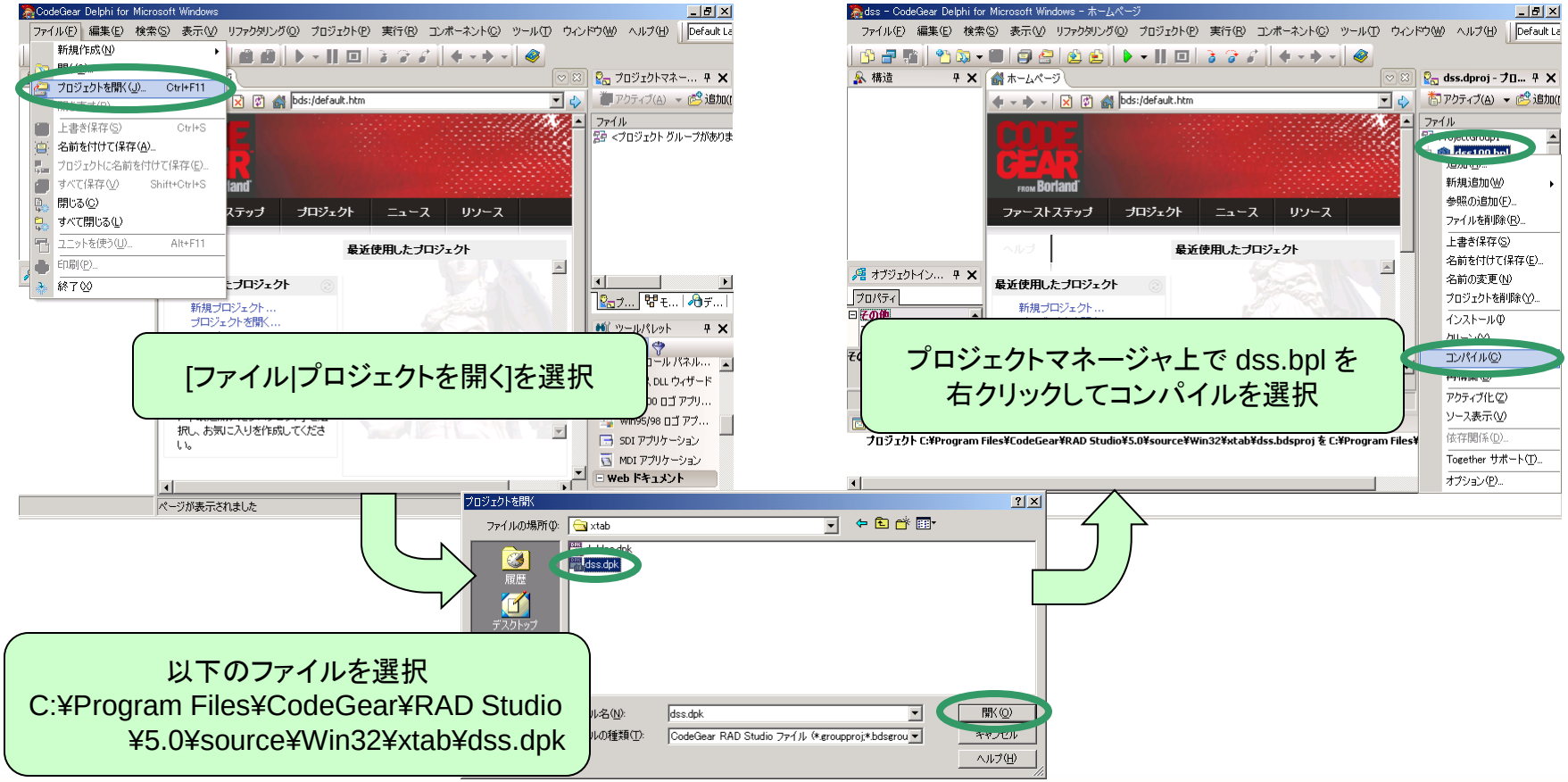
- 実行時用パッケージ（dss.dpk）
  登録時は先に開き、コンパイルのみ行います
- 設計時用パッケージ（dcldss.dpk）
  実行時パッケージの後で開き、インストールを行います
  ※Delphi 2007からは dclbde.dcp への参照を明示的に行う必要があります。

最初は実行時用のパッケージをコンパイルします。
以下のパッケージファイルを開き、コンパイルします。（次頁）

C:¥Program Files¥CodeGear¥RAD Studio¥5.0¥source¥Win32¥xtab¥dss.dpk

1）既存コンポーネントを利用する方法（操作例）

Ⅱ.ソース形式のパッケージ取込例（DecisionCube）



1）既存コンポーネントを利用する方法（操作例）

Ⅱ.ソース形式のパッケージ取込例（DecisionCube）

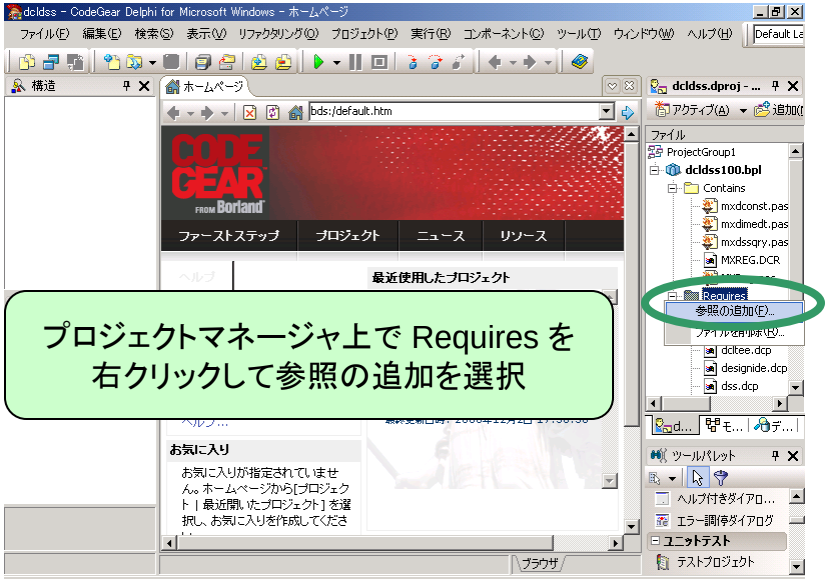
続いて設計時パッケージをインストールします。先程のプロジェクトが開かれたままになっているなら、一度全て閉じます。
その後、以下のパッケージファイルを開きます。

C:¥Program Files¥CodeGear¥RAD Studio¥5.0¥source¥Win32¥xtab¥dcldss.dpk

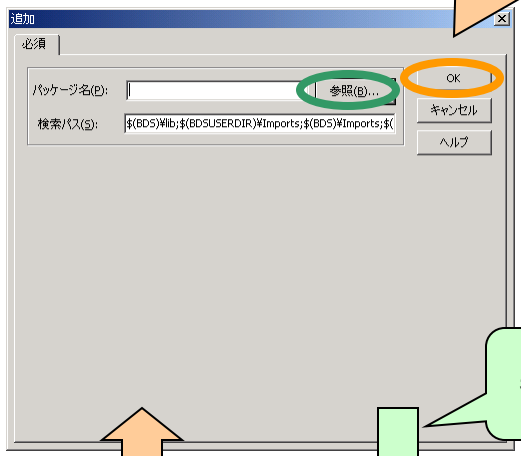
パッケージが開かれたなら、プロジェクトマネージャを使って dclbde.dcp を必須パッケージに追加します。
その後、以下のパッケージファイルを開き、インストールします。（次頁）

1）既存コンポーネントを利用する方法（操作例）

Ⅱ.ソース形式のパッケージ取込例（DecisionCube）



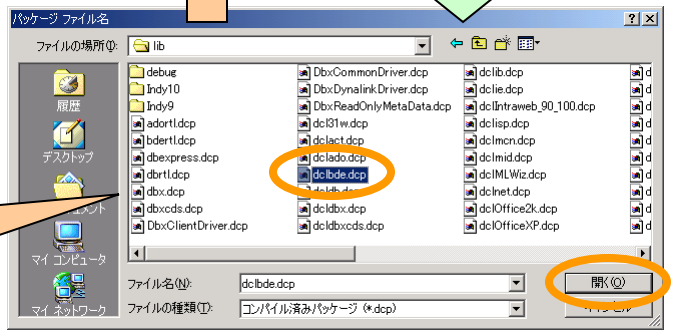
プロジェクトマネージャ上で Requires を
右クリックして参照の追加を選択



OKボタンを押下

次ページへ

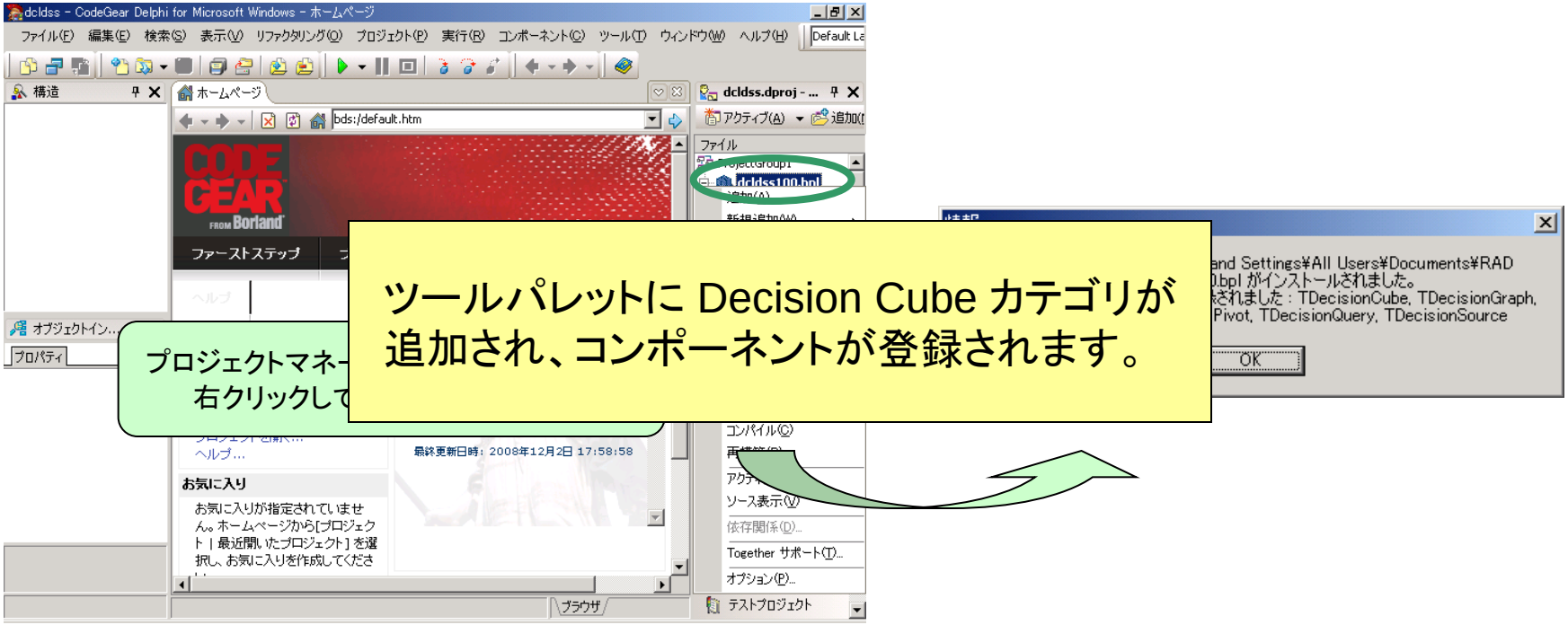
参照ボタンを押下



以下のファイルを選択
C:¥Program Files¥CodeGear¥RAD Studio
¥5.0¥lib¥dclbde.dcp

1）既存コンポーネントを利用する方法（操作例）

Ⅱ.ソース形式のパッケージ取込例（DecisionCube）



1）既存コンポーネントを利用する方法（操作例）

Ⅱ.ソース形式のパッケージ取込例（DecisionCube）

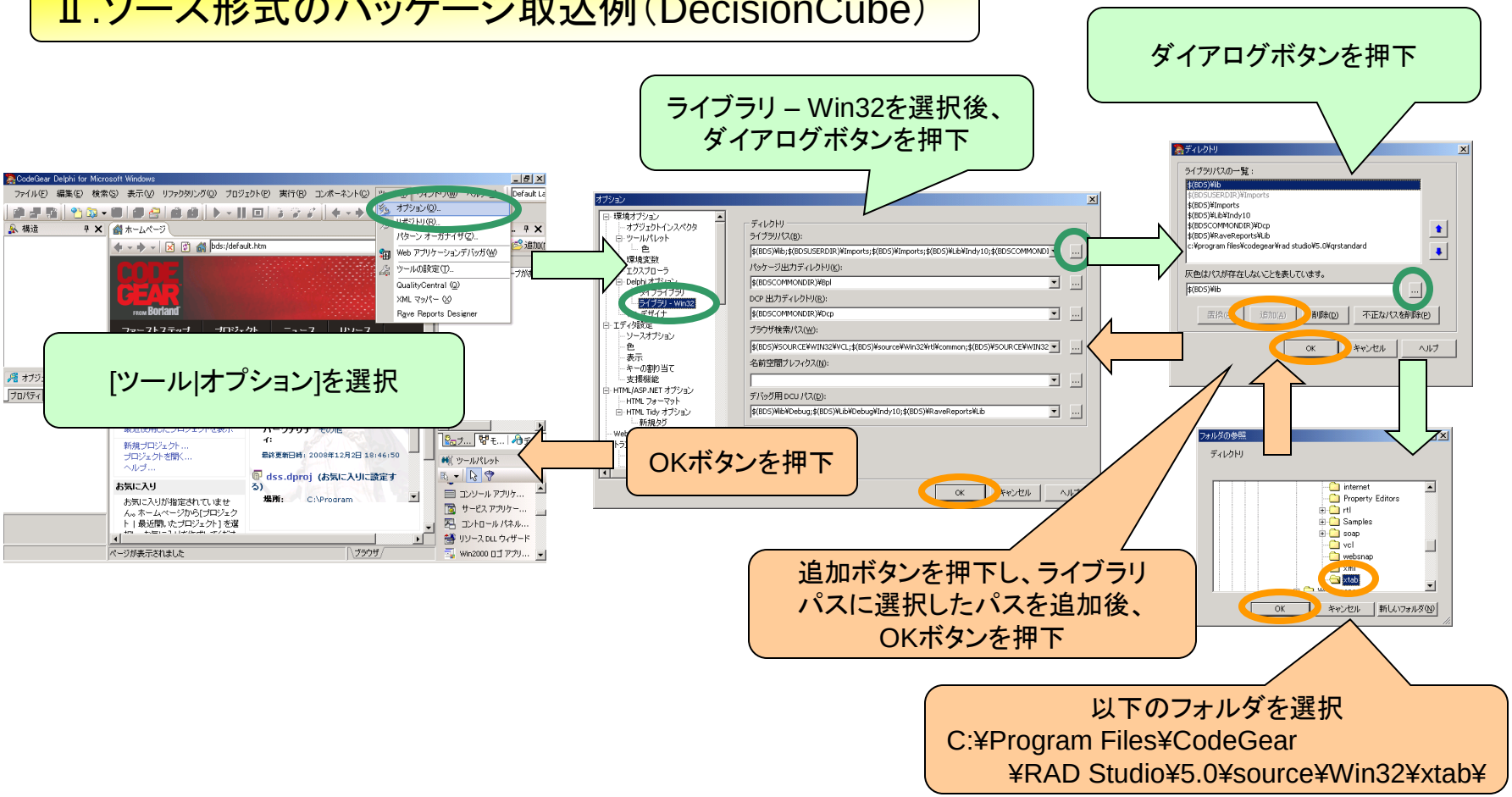
ここまでの操作でコンポーネントがツールパレットに配置され、設計画面で利用できるようになりますが、これらのコンポーネントを使ったアプリケーションをコンパイルしようとすると、ユニットが見つからないとのエラーが発生します。
エラーを解消するには、各コンポーネントソースから生成されたコンパイル済ユニット（dcu）を始めとした必要ファイルにパスを通す必要があります。

簡単な方法として、統合開発環境のライブラリパスにコンパイル済ユニットがあるパスを追加します。
メニューから[ツール|オプション]を開き、以下のパスを追加します。（次頁）

C:¥Program Files¥CodeGear¥RAD Studio¥5.0¥source¥Win32¥xtab¥

1）既存コンポーネントを利用する方法（操作例）

Ⅱ.ソース形式のパッケージ取込例（DecisionCube）



1）既存コンポーネントを利用する方法

②コンポーネントの配布形態

＜コンポーネントの配布形態＞

Ⅲ. バイナリ／ソース形式のコンポーネント単体

主なファイル：

ソースコード（pas）、コンパイル済ユニット（dcu）

組込手順：

1. 新規パッケージを作成するか、既存パッケージを開く
2. プロジェクトマネージャ上でパッケージを右クリック
3. ポップアップメニューの追加を選択
4. 「ユニットファイル名」ダイアログでコンポーネントを選択
   ※コンパイル済ユニットを選択する場合は、絞込みの解除が必要

以降はソース形式のパッケージ組み込み手順と同様になります。

注意事項：

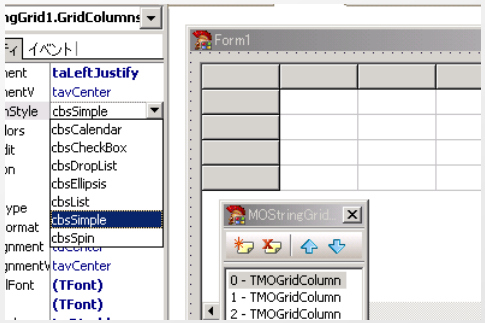
libフォルダ内に dclusr.dpk という空のパッケージファイルが予め用意されており、ユーザが自由に利用できます。

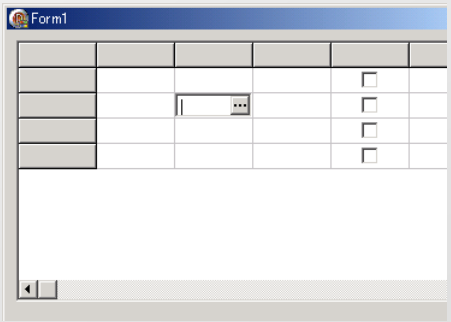
1）既存コンポーネントを利用する方法（操作例）

Ⅲ.単体のコンポーネント取込例（MOStringGrid）

※MOStringGridとは．．．

- Delphi 6のCompanion Tools CDに添付されていたコンポーネントで、標準のTStringGridを拡張し、各列毎にボタンやチェックボックス等を表示可能にした鈴木 政志さん作成のコンポーネント
- インターネット上で公開されていたソースは設計関係のコードが分離されておらず、Delphi 6以上に登録するには修正が必要
- Companion Tools CDに添付されているソースはリソースファイルが不足しているため、その箇所のみコメントアウトするか、リソースを用意する必要がある





1）既存コンポーネントを利用する方法（操作例）

Ⅲ.単体のコンポーネント取込例（MOStringGrid）

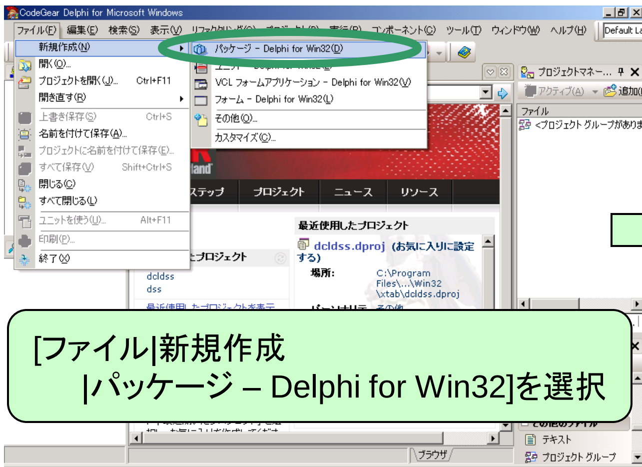
コンポーネント単体で配布されているケースでは、既存パッケージを利用するか
パッケージを新規作成し、コンポーネントをそのパッケージに登録します。（次頁）

後はソース形式のパッケージ取込と同じ手順を踏むことで、コンポーネントの登録が
可能です。
Delphiには予めユーザ向けのパッケージとして、dclusr.dpkが用意されていますので、
初めはこのパッケージを利用するのも良いでしょう。

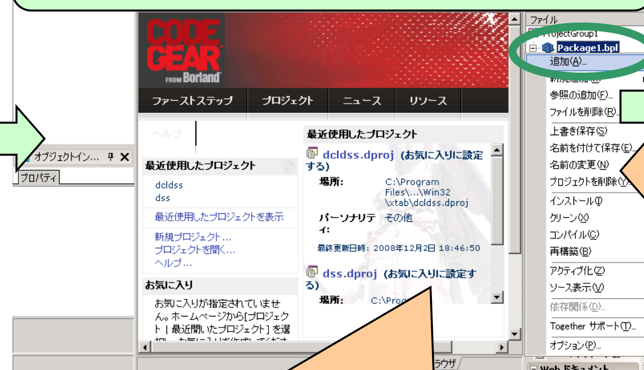
バイナリ形式（dcu）のコンポーネントは、コンポーネントをコンパイルした環境と登録・
利用する環境でバージョンを合わせる必要があります。
ソース形式の場合は、バージョンに依存するような記述をしていない限りはこの制限は
ありません。
但し、Delphi 6以降では設計時と実行時のソースの分離が厳密に必要になり、その
加減で修正が必要になることがあります。

1）既存コンポーネントを利用する方法（操作例）

Ⅲ.単体のコンポーネント取込例（MOStringGrid）

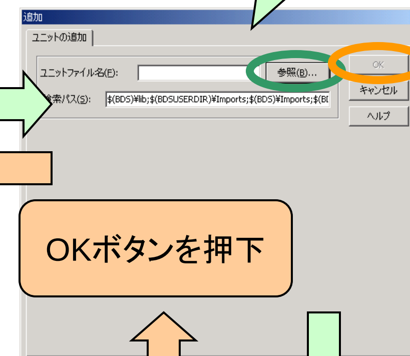


[ファイル|新規作成
|パッケージ – Delphi for Win32]を選択

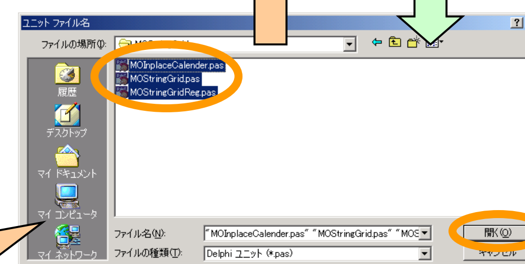


プロジェクトマネージャ上で Package1.bpl
を右クリックして追加を選択

参照ボタンを押下



OKボタンを押下



パッケージに追加後は、ソース形式の
パッケージを取り込む例と同様になります。

必要ファイルを選択
複数同時選択が可能です。バイナリ形式のファイルの場合、
ファイルの種類を全てのファイルにしてからdcuファイルを選択します。

1）既存コンポーネントを利用する方法

②コンポーネントの配布形態

＜配布形態の違いによるメリット・デメリット＞

それぞれの配布形態について、利用者の立場から見たメリット・デメリットには以下のようなものがあります。

■バイナリ配布
- ・統合開発環境の組み込みが比較的簡単である
- ・動作の詳細について考える必要が無く、一利用者に徹することができる
- ・利用したいDelphiのバージョンに応じたバイナリが必要である
- ・修正や変更、バージョンアップ対応等は作者側次第となる

■ソース配布
- ・複数のDelphiバージョンで利用可能である
- ・処理内容を確認できるため、安心感がある
- ・修正や仕様変更を自らできる
- ・利用する際にコンパイル環境を整え、バイナリを作成しなければならない
- ・利用者側であっても、ある程度の知識・技術が求められることがある

2）コンポーネントを拡張する方法

①継承とカスタムコンポーネント

＜継承と派生＞

オブジェクト指向プログラミングでは、あるクラスの属性や機能を完全に引き継ぐことを「継承」といい、新しい属性や機能を付加して新たなクラスとして派生させることができます。継承の際、元になったクラスを上位クラス（superclass）、派生してできたクラスを下位クラス（subclass）と呼びます。
Delphiにおいて全てのクラスはTObjectを直接、または間接的に継承しています。同様にコンポーネントはTComponentを継承しています。

＜アクセス制御＞

ユニットやコンポーネントには外部から利用されたくないメンバ（プロパティやメソッド）を隠蔽することができます。逆に外部から利用させるためにメンバを公開することもでき、これらを設定することをアクセス制御と呼びます。

2）コンポーネントを拡張する方法

①継承とカスタムコンポーネント

＜クラスのアクセス制御＞

Delphiではクラスに対してメンバ単位でアクセス制御を行うことが可能です。
スコープ（可視性）と呼ばれるアクセス制御のレベルは以下の4段階存在します。

可視性	内容
private	そのクラスが宣言されたユニット内でのみアクセス可能。 外部から変更されたくないメンバを定義します。
protected	そのクラスが宣言されたユニット及び下位クラスからアクセス可能。 派生された先でも利用する可能性のあるメンバを定義します。
public	そのクラスが宣言されたユニット外からもアクセス可能。 コンポーネントのプロパティやメソッドとして利用するメンバを定義します。
published	可視性としてはpublicと同様だが、ここで定義したプロパティやイベントは フォームファイルへ保存可能になる。 オブジェクトインスペクタに表示したいプロパティやイベントを定義します。

上位クラスで設定したスコープを下位クラスで広げることは可能ですが、逆に狭めることはできません。また、privateで宣言された場合、下位クラスであってもアクセスできないため、再定義できません。

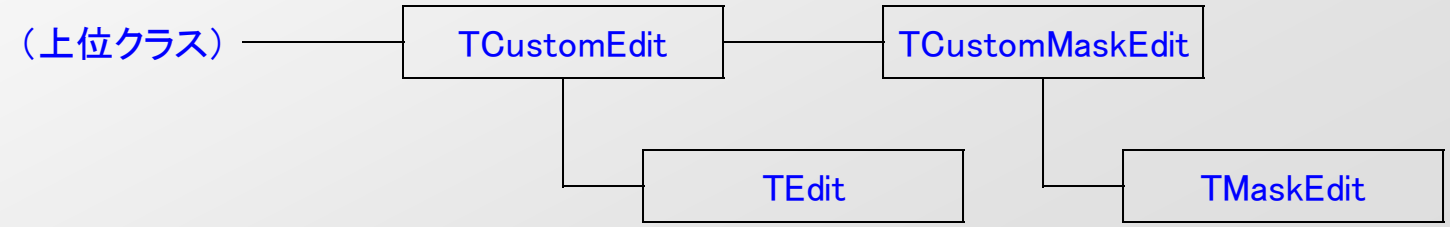
2）コンポーネントを拡張する方法

①継承とカスタムコンポーネント

＜カスタムコンポーネント＞

スコープは上位クラスで広げてしまうと下位クラスで狭めることができません。
そこで、スコープは狭めたまま機能だけ実装したクラスとそのクラスを継承し、スコープを広げたクラスを作成することで、再利用性が向上します。このようなスコープを狭めたままの状態のコンポーネントをカスタムコンポーネントと呼び、主に上位クラスとして利用されています。
標準のTEditやTCheckBoxにもそれぞれTCustomEditやTCustomCheckBoxといったクラスが用意されています。

例）TEditとTMaskEditの継承関係



2）コンポーネントを拡張する方法

②コンポーネントの拡張手順

＜コンポーネントの拡張手順＞

Delphiで用意されているコンポーネントはTObjectから段階を追って派生しており、
それぞれの段階で作成されたクラスを自由に利用することが可能です。そのため、
完全に新規でコンポーネントを作成することは少なく、何らかの上位クラスを元に
継承し機能拡張するのが通常です。
仕様や動作が決定した後、実際にコンポーネントを作成する際の手順を以下に
まとめます。

Ⅰ．パッケージを作成または選択します。
Ⅱ．作成したいコンポーネントの上位クラスを選択します。
Ⅲ．ユニットを作成し、機能を実装します。
Ⅳ．コンパイル・テストし、動作を確認します。

コンポーネント開発の場合、コンポーネントの機能毎にテスト方法を考慮しなければ
なりませんので、テストのやり方を色々と工夫する必要があります。

3）ピックアップ！MaskEditコンポーネントの拡張

①拡張機能の紹介

＜拡張機能の紹介＞

コンポーネントの機能拡張の具体例として、第1回テクニカルセミナーで配布しました
TMGRMaskEditを題材に拡張手順と実際の操作を紹介したいと思います。
まず、TMGRMaskEditの機能紹介を行います。

[TMGRMaskEditの拡張機能]

Ⅰ．Alignmentプロパティ
→ 入力文字列の横方向の配置を指定します。
左寄せ、右寄せ、中央寄せを選択できます。

Ⅱ．CharaSetプロパティ
→ 入力文字列の種類を指定します。
指定なし、半角文字のみ、全角文字のみを選択できます。

3）ピックアップ！MaskEditコンポーネントの拡張

①拡張機能の紹介

＜拡張機能の紹介＞

[TMGRMaskEditの拡張機能]

Ⅲ. EnterNextプロパティ

→ Enterキーの押下でフォーカスの移動を行います。
移動する、移動しないを選択できます。

Ⅳ. PageCodeプロパティ

→ CCSID（CodePage）に応じて半角英小文字の制御を行います。
JP-1相当（半角英小文字を大文字に変換）、JP-2相当（入力内容をそのまま）を選択できます。

以上の4つプロパティの追加を行い、関連機能の調整を行います。

※このサンプルは理解しやすいことを優先しており、本来想定される動作とは異なる動作の箇所も許容しております。

3）ピックアップ！MaskEditコンポーネントの拡張

②MaskEditコンポーネントの拡張手順

＜MGRMaskEditの作成手順概要＞

Ⅰ．新規パッケージとしてMGRComponentsを作成します。
Ⅱ．MGRMaskEditコンポーネントの上位クラスとして
　TCustomMaskEditを選択します。（参考ソース：Step01）
Ⅲ．MGRMaskEditユニットを作成し、各機能を実装します。
　Ⅲ－１．Alignmentプロパティの実装（参考ソース：Step02）
　Ⅲ－２．CharaSetプロパティの実装（参考ソース：Step03）
　Ⅲ－３．EnterNextプロパティの実装（参考ソース：Step04）
　Ⅲ－４．PageCodeプロパティの実装（参考ソース：Step05）
　Ⅲ－５．アクセス制御レベル等の調整（参考ソース：Step06）
Ⅳ．パッケージをインストールし、動作を確認します。（参考ソース：Step07）

3）ピックアップ！MaskEditコンポーネントの拡張

②MaskEditコンポーネントの拡張手順

＜パッケージの作成＞

新たなコンポーネントを作成する際、最初に行うことはパッケージを作成することです。
既に幾つかのコンポーネントを作成している場合は既存のパッケージに追加することも可能ですし、Delphi標準としてユーザ向けに利用可能なパッケージが用意されています。
今回は新規パッケージMGRComponentsを作成します。

＜上位クラスの選定とユニットの作成＞

続いて上位クラスを指定し、ユニットを作成します。
スコープを考慮した場合、より自由度の高いカスタムコンポーネントから選択することが多くなると思われます。
今回はTMaskEditを拡張したTMGRMaskEditを作成しますが、実際にはTMaskEditの上位クラスであるTCustomMaskEditを上位クラスとして選択します。

3）ピックアップ！MaskEditコンポーネントの拡張

②MaskEditコンポーネントの拡張手順

＜統合開発環境の操作：プロパティ追加＞

private～publishedの各宣言部に「property プロパティ名： データ型；」まで記述します。
記述後、Ctrl + Shift + C を押下することで、コード補完機能が働き、プロパティ値を保存するためのフィールド（変数）、プロパティ書込時に実行されるメソッドが作成されます。

＜統合開発環境の操作：コンストラクタ／デストラクタ追加＞

public宣言部で Ctrl + Space を押下することでコード補完リストが表示されます。
このリスト中にテンプレートの形でコンストラクタの作成があります。また、デストラクタは既存メソッドのオーバーライド候補として表示されます。
※Delphi 2006ではコンストラクタもデストラクタと同様の形式でリストされます。
※コンストラクタは初期化処理に用いられる特別なCreateメソッドです。同様にデストラクタは終了処理に用いられる特別なDestroyメソッドです。

3）ピックアップ！MaskEditコンポーネントの拡張

②MaskEditコンポーネントの拡張手順

＜Alignmentプロパティの実装ポイント＞

左右中央寄せを実現するには、CreateParamsメソッドをオーバーライドし、パラメータを変更するだけです。詳細はWindows APIについての知識が必要となりますので、MSDN等でご確認下さい。

※この実装方法はWin95等の古いOS上では正しく動作しませんが、それらのOSが既にサポート対象外となっているため、特に問題は無いと思われます。

＜CharaSetプロパティの実装ポイント＞

全角、半角等の文字種別を実現するには、Changeメソッドをオーバーライドし、入力モードに不適切な入力を除外します。このサンプルでは併せてOS/400上の文字長を考慮した入力制限も追加します。

こちらで利用する「半角文字を全角文字に変換する」機能と「OS/400上の文字長を考慮して文字列の調整を行う」機能は関数化し、ユニット内から利用可能にします。

3）ピックアップ！MaskEditコンポーネントの拡張

②MaskEditコンポーネントの拡張手順

＜EnterNextプロパティの実装ポイント＞

Enterキー押下時のフォーカス移動を実現するには、KeyDownメソッドをオーバーライドし、フォーカスの移動メッセージを親フォームに対して送ります。フォーカスを移動させる方法はさまざまなやり方が存在しますので、色々とお試しいただくと良いでしょう。
尚、Windows標準のフォーカス移動キーはTabキーであることから、Enterキーの押下をTabキーの押下に変更するといった方法でも可能です。

＜PageCodeプロパティの実装ポイント＞

半角英小文字を対象に半角英大文字へ変換する機能を実現するには、CharaSetプロパティ同様、Changeメソッドで制御する方法があります。但し、Changeメソッドは一文字単位の入力で発生しますので、ここでの処理はできるだけ軽いものにしておく必要があります。
そこで、今回のサンプルではKeyPressメソッドで制御を行います。
この方式の場合、クリップボードからの貼り付けのような処理に対して制御漏れが発生してしまうことがあります。今回のサンプルでは対応していませんので、どのように制御を追加すれば良いか、検討してみて下さい。

3）ピックアップ！MaskEditコンポーネントの拡張

＜MGRMaskEdit 完了時点ソース＞

```
(*****************************************************************************
  << AS/400入力用MaskEdit >>
                                                2008.12.10 株式会社ミガロ

   標準のTMaskEditに対し下記機能拡張を行っています。

   Alignmentプロパティ: 文字列の横方向の配置を指定
                         taLeftJustify - 左寄せ
                         taRightJustify - 右寄せ
                         taCenter - 中央寄せ
   CharCaseプロパティ：文字列のセット仕様変更のため、機能削除
   CharaSetプロパティ：入力文字列の属性指定
                         csNone - 属性指定なし
                         csSBCSOnly - 半角文字列のみ入力可能
                         csDBCSOnly - 全角文字列のみ入力可能
   EnterNextプロパティ：Enterキー押下による項目移動の設定
   PageCodeプロパティ：CCSIDにあわせた文字列属性を指定
                         pcJP1 - CCSID=5026系(半角英小文字使用不可)
                         pcJP2 - CCSID=5035系(半角英小文字使用可)
   MaxLengthプロパティ：シフト文字を含む文字長の指定
*****************************************************************************)
unit MGRMaskEdit;

interface

uses
  Windows, Messages, SysUtils, Classes, Controls, Forms, StdCtrls, Mask;

type
  TCharaSet = (csNone, csSBCSOnly, csDBCSOnly);
  TPageCode = (pcJP1, pcJP2);

  TMGRMaskEdit = class(TCustomMaskEdit)
  private
    { Private 宣言 }
    FAlignment: TAlignment;
    FCharaSet: TCharaSet;
    FEnterNext: Boolean;
    FPageCode: TPageCode;
    procedure SetAlignment(const Value: TAlignment);
```

```
  protected
    { Protected 宣言 }
    procedure Change; override;
    procedure CreateParams(var Params: TCreateParams); override;
    procedure KeyDown(var Key: Word; Shift: TShiftState); override;
    procedure KeyPress(var Key: Char); override;
  public
    { Public 宣言 }
    constructor Create(AOwner: TComponent); override;
  published
    { Published 宣言 }
    property Alignment: TAlignment read FAlignment write SetAlignment
                                                        default taLeftJustify;
    property CharaSet: TCharaSet read FCharaSet write FCharaSet default csNone;
    property EnterNext: Boolean read FEnterNext write FEnterNext default True;
    property PageCode: TPageCode read FPageCode write FPageCode default pcJP1;
    property Align;
    property Anchors;
    property AutoSelect;
    property AutoSize;
    property BevelEdges;
    property BevelInner;
    property BevelOuter;
    property BevelKind;
    property BevelWidth;
    property BiDiMode;
    property BorderStyle;
//    property CharCase;
    property Color;
    property Constraints;
    property Ctl3D;
    property DragCursor;
    property DragKind;
    property DragMode;
    property Enabled;
    property EditMask;
    property Font;
    property ImeMode;
    property ImeName;
    property MaxLength;
```

3）ピックアップ！MaskEditコンポーネントの拡張

＜MGRMaskEdit 完了時点ソース＞

```
    property ParentBiDiMode;
    property ParentColor;
    property ParentCtl3D;
    property ParentFont;
    property ParentShowHint;
    property PasswordChar;
    property PopupMenu;
    property ReadOnly;
    property ShowHint;
    property TabOrder;
    property TabStop;
    property Text;
    property Visible;
    property OnChange;
    property OnClick;
    property OnDblClick;
    property OnDragDrop;
    property OnDragOver;
    property OnEndDock;
    property OnEndDrag;
    property OnEnter;
    property OnExit;
    property OnKeyDown;
    property OnKeyPress;
    property OnKeyUp;
    property OnMouseActivate;
    property OnMouseDown;
    property OnMouseEnter;
    property OnMouseLeave;
    property OnMouseMove;
    property OnMouseUp;
    property OnStartDock;
    property OnStartDrag;
  end;

procedure Register;

implementation
```

```
procedure Register;
begin
  RegisterComponents('MIGARO', [TMGRMaskEdit]);
end;

{*****************************************************************************
 目的: 半角文字列を全角文字列に変換
 引数: Str - 文字列(全半角混在可)
 戻値: 変換後文字列(全角文字列のみ)
*****************************************************************************}
function SingleToDoubleByteString(const Str: String): String;
begin
  //全角文字列に変換
  SetLength(Result, LCMapString(LOCALE_SYSTEM_DEFAULT, LCMAP_FULLWIDTH,
    PChar(Str), Length(Str), nil, 0));
  SetLength(Result, LCMapString(LOCALE_SYSTEM_DEFAULT, LCMAP_FULLWIDTH,
    PChar(Str), Length(Str), PChar(Result), length(Result)));
  //"¥"マークはパス区切り記号(半角のバックスラッシュと共用)で
  //Windowsの仕様上変換対象外になるため、個別に処理
  Result := StringReplace(Result, '¥', '￥', [rfReplaceAll]);
end;

{*****************************************************************************
 目的: シフト文字を考慮した文字長の文字列を取得
 引数: AText - 対象文字列、AMaxLength - シフト文字を含む文字長
 戻値: シフト文字を考慮して指定文字長に収まる文字列
*****************************************************************************}
function GetLengthText(AText: String; AMaxLength: Integer): String;
  // シフト文字を考慮した文字長取得
  function GetEbcdicLength(const s: String): Integer;
  var
    i: Integer;
    InDBCS: Boolean;
  begin
    InDBCS := False;
    Result := Length(s);

    for i := 1 to Length(s) do
    begin
      case ByteType(s, i) of
```

3）ピックアップ！MaskEditコンポーネントの拡張

＜MGRMaskEdit 完了時点ソース＞

```
        mbSingleByte:  //ASCII 文字もしくは半角カタカナ
          InDBCS := False;
        mbLeadByte:    //全角文字の先頭バイト
          if InDBCS = False then
          begin
            InDBCS := True;
            Result := Result + 2;
          end;
      end;
    end;
  end;
begin
  if (AMaxLength <= 0) or (GetEbcdicLength(AText) <= AMaxLength) then
    Result := AText
  else
  begin
    repeat
      AText := Copy(WideString(AText), 1, Length(WideString(AText)) - 1);
    until (GetEbcdicLength(AText) <= AMaxLength);
    Result := AText;
  end;
end;

{ TMGRMaskEdit }

procedure TMGRMaskEdit.Change;
var
  bSelFlg: Boolean;
  i, iSel: Integer;
  sText: String;
begin
  inherited;

  bSelFlg := False;
  iSel := 0;

  //カーソル位置の保持(実行時のみ)
  if (not (csDesigning in ComponentState)) and (0 < Length(Text))then
    if SelStart <> Length(Text) then
    begin
      iSel := SelStart;
      bSelFlg := True;
    end;

  //マスク指定されていない場合
  if not IsMasked then
  begin
    case FCharaSet of
      csSBCSOnly:  //半角文字限定の場合、全角文字を消去
        begin
          i := 1;
          sText := Text;
          while i < Length(sText) do
            if byteType(sText,i) = mbLeadByte then
              Delete(sText, i, 2)
            else
              Inc(i);
          if Text <> sText then Text := sText;
        end;
      csDBCSOnly:  //全角文字限定の場合、半角文字を全角文字に変換
        Text := SingleToDoubleByteString(Text);
    end;

    //最大文字列長内に収める
    sText := GetLengthText(Text, MaxLength);
    if Text <> sText then Text := sText;

    //カーソル位置の調整(実行時のみ)
    if (not (csDesigning in ComponentState)) and bSelFlg then
    begin
      if FCharaSet = csDBCSOnly then
        SelStart := iSel + 1
      else
        SelStart := iSel;
    end
    else
      SelStart := Length(Text);
  end;
end;
```

3）ピックアップ！MaskEditコンポーネントの拡張

＜MGRMaskEdit 完了時点ソース＞

```
constructor TMGRMaskEdit.Create(AOwner: TComponent);
begin
  inherited;

  ControlStyle := ControlStyle - [csSetCaption];
  FAlignment := taLeftJustify;
  FCharaSet := csNone;
  FEnterNext := True;
  FPageCode := pcJP1;
end;

procedure TMGRMaskEdit.CreateParams(var Params: TCreateParams);
begin
  inherited;

  case FAlignment of
    taRightJustify: Params.Style := Params.Style or ES_RIGHT;  //右寄せ
    taCenter:       Params.Style := Params.Style or ES_CENTER; //中央寄せ
  end;
end;

procedure TMGRMaskEdit.KeyDown(var Key: Word; Shift: TShiftState);
begin
  if FEnterNext and (Key = VK_RETURN) then
  begin
    if (Shift = []) or (Shift = [ssShift]) then
    begin
      //フォーカス移動処理(何も押されていない場合次へ、
      //                    Shiftのみ押されている場合前へ)
      if Shift = [] then
        SendMessage(GetParentForm(Self).Handle, WM_NEXTDLGCTL, 0, 0)
      else
        SendMessage(GetParentForm(Self).Handle, WM_NEXTDLGCTL, 1, 0);
      Key := 0;
    end;
  end;
  inherited;
end;
```

```
procedure TMGRMaskEdit.KeyPress(var Key: Char);
begin
  inherited;

  if FEnterNext and (Key = Chr(VK_RETURN)) then Key := #0;

  if (FPageCode = pcJP1) and (FCharaSet <> csDBCSOnly) then
    if ByteType(Key, 1) = mbSingleByte then Key := UpCase(Key);
end;

procedure TMGRMaskEdit.SetAlignment(const Value: TAlignment);
begin
  if FAlignment <> Value then
  begin
    FAlignment := Value;
    RecreateWnd;
  end;
end;

end.
```

ご静聴ありがとうございました