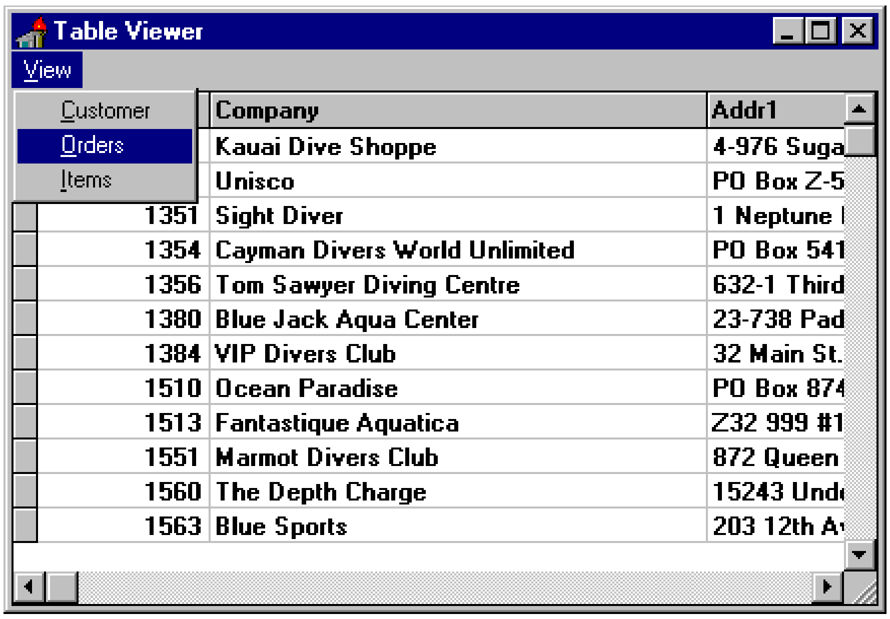
Typecasting Explained: Part 3

by Brian Long



	Company	Addr1
	Kauai Dive Shoppe	4-976 Suga
	Unisco	PO Box Z-5
1351	Sight Diver	1 Neptune I
1354	Cayman Divers World Unlimited	PO Box 541
1356	Tom Sawyer Diving Centre	632-1 Third
1380	Blue Jack Aqua Center	23-738 Pad
1384	VIP Divers Club	32 Main St.
1510	Ocean Paradise	PO Box 874
1513	Fantastique Aquatica	Z32 999 #1
1551	Marmot Divers Club	872 Queen
1560	The Depth Charge	15243 Und
1563	Blue Sports	203 12th A

This article doesn't really cover anything more on typecasting than has been covered previously: it's more targeted at taking a small task through several stages of development, making use of a variety of useful Object Pascal techniques, which include safe object typecasting. The task involves having an application that views tables and more particularly has a number of menu items that load up particular tables. The end result is shown in the screen shot on the right.

There are a number of ways to tackle the problem of how to code this in Delphi. We'll start by looking at the obvious ones and move along to the less apparent options. After having designed the form, one approach would be to make individual event handlers for each of the three menu items that look rather like Listing 1. Notice the `try...finally` statement that ensures the cursor is changed back to normal, regardless of whether there is an exception or not.

Immediately we can save some typing, and simultaneously make the code more efficient, by employing a `with` statement for both the `Screen` and `Table1` references (see project RTTI1.DPR on the disk), as shown in Listing 2.

The `with` statement used here causes the compiler to look at each term in the `with` block and check if it would be sensible to prefix it with `Table1.` and if it is, it implicitly does so. If it is not sensible, it then checks to see if it would be sensible to prefix it with `Screen.` and again, if it is, it does, and if it isn't, it doesn't (still with me?). If you are going to use multiple variables in a `with` statement, look very carefully at the order in which you list them. The precedence goes from right to left, and it is easy for the compiler to access a field of the wrong variable if you are not careful. I'll come back to this later.

The main problem with having three event handlers much like the one above is that we are duplicating most of the code. If the logic needs changing, we need to change it in three places. Instead, let's try another approach, using some code sharing to share an event handler between the three menu items. We'll consider that we are starting again from scratch, for the benefit of the description.

Bring up the menu designer for the main menu component and double click the first item to manufacture an event handler for its `OnClick` event. The event handler has been named after the menu item, `Customer1Click` in my case, which sort of implies, by its name, that it is intended for use just by the `Customer1` menu item. To make it sound a bit more generic, go to the `Events` page of the `Object Inspector` (`F11`, `Ctrl+Tab` will do this, for those of you not addicted to your mouse buttons). Over the top of `Customer1Click`, type a new name, `TableMenuClick`, and press `Enter`. You'll notice that this changes the name of the event handler in the source window.

Now get the menu designer back – you can do this by pressing `Alt+0` or choosing `View | Window List` to get the window list and choosing the window with the caption `Form1.MainMenu1`. Select the second menu item (that's a single click this time) and then go back to the `Object Inspector` (`F11`). Instead of making a new event handler, use

➤ *Listing 1*

```
procedure TForm1.CustomerClick(
  Sender: TObject);
begin
  Screen.Cursor := crHourGlass;
  try
    Table1.DisableControls;
    Table1.Close;
    Table1.TableName :=
      'CUSTOMER';
    Table1.Open;
    Table1.EnableControls;
  finally
    Screen.Cursor := crDefault;
  end;
end;
```

➤ *Listing 2*

```
procedure TForm1.CustomerClick(
  Sender: TObject);
begin
  with Screen, Table1 do begin
    Cursor := crHourGlass;
    try
      DisableControls;
      Close;
      TableName := 'CUSTOMER';
      Open;
      EnableControls;
    finally
      Cursor := crDefault;
    end;
  end;
end;
```

the drop down arrow (Alt+down arrow) to list the event handlers and choose TableMenuClick. After doing this again for the third menu item we're now ready to implement this common event handler.

One approach would be to check that a menu item caused the event to happen, by examining Sender, and then set Table1.TableName based on the Caption property of the menu item. The trouble with this is that menu item captions have a habit of being fiddled with, and so the reliability of the code would not necessarily be high. Instead, what we'll do is use the Tag property of each menu item to identify it. First we need to set the Tag property to a unique value for each menu item of interest, via the Properties page of the Object Inspector. We'll use 1, 2 and 3 respectively. Now we can employ the run-time type information operators to do some safe typecasting, and get code that looks like Listing 3 (see project RTTI2.DPR). You may notice that I am checking for a condition (non-empty table name) and then performing an action, where the general approach is to perform the action and react to any exceptions using a try...except block. The only reason I am checking is for brevity.

We check that the component that caused the event to happen is a TMenuItem and then go into a with statement, this time on three object references. This time, the order of the references is important. We refer in the code to the Tag property, which is common to all components. Because we actually want to refer to the menu item's Tag property, the menu item must appear last in the list (ie *first* in the precedence order). If it were to appear elsewhere, we would be referring to either the Tag of the table or of the Screen object.

The code still has room for improvement. Nested if statements which compare ordinal values can be replaced by a case statement. This changes the inside of the with block to the code shown in Listing 4, as found in project RTTI3.DPR.

To take it to the next stage, we need to know about a convenient, but strange, facility in Object Pascal. Normally, variables are not initialised to any value *(except object data fields which are all zeroed)*, but we can get initialised variables using typed constants (which aren't really constants at all). They have initial values stored in the executable file, and are given them at program startup. If the value is modified any time after that, they keep that new value. This is true regardless of whether they are local or non-local. The reworking of TableMenuClick (Listing 5) is in project RTTI4.DPR.

Notice that I have stopped comparing the TableName to an empty string to decide whether to open the table. Again, I am checking rather than exception handling. There is a good reason this time.

Listing 3

```
procedure TForm1.TableMenuClick(Sender: TObject);
begin
  if Sender is TMenuItem then
  with Screen, Table1, TMenuItem(Sender) do begin
    { Ensure menu item is last so the Tag references go to it,
      not the table }
    Cursor := crHourGlass;
    try
      DisableControls;
      Close;
      if Tag = 1 then TableName := 'CUSTOMER'
      else if Tag = 2 then TableName := 'ORDERS'
      else if Tag = 3 then TableName := 'ITEMS';
      if TableName <> '' then Open;
      EnableControls;
    finally
      Cursor := crDefault;
    end;
  end;
end;
```

Listing 4

```
Cursor := crHourGlass;
DisableControls;
Close;
case Tag of
  1: TableName := 'CUSTOMER';
  2: TableName := 'ORDERS';
  3: TableName := 'ITEMS';
end;
if TableName <> '' then Open;
EnableControls;
Cursor := crDefault;
```

Listing 5

```
procedure TForm1.TableMenuClick(Sender: TObject);
const
  Tables: array[1..3] of String[8] = ('CUSTOMER', 'ORDERS', 'ITEMS');
begin
  if Sender is TMenuItem then
  with Screen, Table1, TMenuItem(Sender) do begin
    { Ensure menu item is last so the Tag references go to it,
      not the table }
    Cursor := crHourGlass;
    try
      DisableControls;
      Close;
      if Tag in [1..3] then begin
        TableName := Tables[Tag];
        Open;
      end;
      EnableControls;
    finally
      Cursor := crDefault;
    end;
  end;
end;
```

Out of range array indexing will only cause an exception if a compiler option (range checking) is set on (not the default). If it is off, there is a good chance of a terminal failure. This time I am using a set operator on a set of values specified using a subrange. The reason for the change is that if another menu item, with a Tag property outside of our range, were to trigger the event, the index used in the array would be out of range: it only has elements from 1 to 3. So I check that it is in range, and if it is, index the array and open the table.

We're nearly there now, but there's one thing left to do to make this a Windows application taking advantage of Windows facilities. The last version (RTTI5.DPR) uses a Windows string table resource to store the table names. This allows easy modification of the application, if the tables change, without the need for recompilation. To do this using only the tools supplied with Delphi, we need a resource script, which is a text file with a .RC extension.

When we have entered all the resource information, we can use the command-line tool BRC.EXE to compile it into a binary .RES file. To link it into the executable, we will use a $R compiler directive, in just the same way as Delphi does in each form file to link the form resources in. Beware though, Delphi automatically generates a .RES file for each project, containing the program icon. This file has the same root name as the project, so ensure your own custom resources go in a differently named file, otherwise Delphi will happily wipe them out.

Since the project is called RTTI5.DPR and the form unit is called RTTI5U.PAS, I will call the resource file RTTI5U.RC. Although the Delphi form resource is in RTTI5U.DFM, there will not be a problem – the DFM file is already compiled. RTTI5U.RC is shown in Listing 6.

The syntax for string table resources is given in the Windows API help file supplied with Delphi. Select Help | Windows API and then Resource Statements and finally STRINGTABLE. Although the RC file syntax was developed for C compilers, Borland's resource compiler allows you to include Pascal constant definition units, to facilitate easy referencing of resources in your application. The file RTTI5U2.PAS that you see being included is very simple:

```
unit Rtti5u2;
interface
const TablesOffset  = 100;
implementation
end.
```

It just has one constant in it. This constant is shared between the resource and the Delphi program: the resources are defined using the constant, the program references the resources using the same constant. So long as the RTTI5U unit has RTTI5U2 in its uses statement, we can re-code the TableMenuClick event handler as shown in Listing 7.

All that is left now is to compile the RC file into a RES file:

```
\DELPHI\BIN\BRC -r RTTI5U.RC
```

The -r flag tells the resource compiler to just compile the resource and not attempt to bind it into the executable. We have a $R compiler directive to do that second step. One point to mention. Since the resource file has the same root name as the unit, you do not need to say {$R RTTI5U.RES} unless you want to, {$R *.RES} suffices.

And there we have it. Several development cycles and we have a reasonably efficient, un-duplicated implementation whose data can be modified without recourse to the compiler (you can relink a changed resource file into the executable either with BRC RTTI5U.RC or, if it is already compiled, with RLINK RTTI5U.RES RTTI5.EXE). Along the way we have seen event handler sharing and renaming, with statements over multiple variables, run-time typing information operators, subranges, typed constants, set operators, string tables and command-line resource compilers.

Brian Long is an independent consultant and trainer specialising in Delphi. His email address is 76004.3437@compuserve.com

➤ *Listing 6*

```
#include <rtti5u2.pas>
STRINGTABLE
BEGIN
  TablesOffset + 1 "CUSTOMER.DB"
  TablesOffset + 2 "ORDERS.DB"
  TablesOffset + 3 "ITEMS.DB"
END
```

➤ *Listing 7*

```
{$R *.RES} { Bind in compiled version of RTTI5U.RC }
procedure TForm1.TableMenuClick(Sender: TObject);
begin
  if Sender is TMenuItem then
  with Screen, Table1, TMenuItem(Sender) do begin
    { Ensure menu item is last so the Tag references go to it,
      not the table }
    Cursor := crHourGlass;
    try
      DisableControls;
      Close;
      if Tag in [1..3] then begin
        TableName := LoadStr(TablesOffset + Tag);
        Open;
      end;
      EnableControls;
    finally
      Cursor := crDefault;
    end;
  end;
end;
```