OML Scripting Guide

Introduction to OML Scripting 3
Definition of Terms 3

Defining an OML Script 5
Using the Pick Lists 6

Variables in OML Scripts 7
Local Variables 7
Global Variables 7

DQL in OML Scripts 8
Processing Commands 9
Sorting and Grouping Operators 9
Control Commands 10
Procedural Commands 10
Functions 11
Symbols and comparison operators 11
Relational Statistical Operators 12
Conditional Logic 12

Displaying Changes 13

Lost Scripts and Error Messages 14

Introduction to Events 15
What are Events? 15
The Event Return 15
Event Strings 16

The Event List 17
Keyboard Events 20
Introduction to Methods 21
The Method List 21

Introduction to Properties 23
The Property List 23

Introduction to Objects 27
The Object List..... 27

Class Properties 50
Font 51
Color 55
Fill 57
Border..... 59
3D Class Properties 61
Shine 63
Shade 65
Rect 67
RectBorder..... 68
RectFill 70
ScrollBorder 72
ScrollFill 74
Child 76
Next 77
Prev 78

Example OML Scripts
Input Validation with PostEdit 79
ValueLoaded Event 80
Report Totals 82
Realtime Data Processing 83
Conditional Subform Display 85

Alphabetical List of OML Scripting Commands … 88

Index 89

Introduction to OML Scripting

DataEase 6 introduces the ability to define OML Scripts, which can be attached to form objects. An "Event" might be a mouse-click, or a field value changing. When an Event takes place, the relevant OML script is triggered and runs like a piece of DQL.

OML Scripts can alter the appearance and contents of any object on your document. As an extreme example, you could write an OML Script which - when a button is pressed - alters the size, shape, colour, font, screen co-ordinates, and contents of every single object on your document.

Additionally, OML Scripts can run a sub-set of DQL, so they can enter/modify/delete data, call programs and procedures, and carry out most other tasks which are usually assigned to a DQL procedure. (In fact, OML Scripts are part of DQL, but because the syntax is different, it is convenient to refer to OML Scripts and DQL as if they were separate scripting languages).

Definition of Terms

Throughout this documentation you will repeatedly come across various object-oriented programming terms. If you consulted six different technical works on Object Oriented Programming, you would probably find six different definitions of such basic terms as "Object" and "Method". The definitions given below are regarded as the most pertinent to DataEase.

Class

The prototype of an object, registered with DataEase or Windows. Sometimes called a template.

Instantiation

The creation of an actual object from a prototype (the class) that defines what the object will be.

Object

An actual example of an object.

…….so when you create an edit box field on screen, you're actually **instantiating** an **object** of the EditBox **Class**.

Control

Windows documentation frequently refers to both **objects** and **classes** as "controls". This documentation shall simply refer to them as "objects", because version 7 of DataEase will introduce ActiveX controls - hence the word 'control' is reserved for describing them.

Method

A function in an object which can be called externally – typically some sort of process relevant to the object, such as "refresh the object's screen display".

Event

Events are the means by which the object sends information back to DataEase - specifically, the news that something has happened which might require attention. (More properly the object reports the news to Windows, which then reports to DataEase).

Event/Methods

It is common practise to give a method fired by an event the same name as the triggering event. Hence an Event/Method consists of both an Event and a Method which, because of their relationship, share the same name.

Property

A variable characteristic (for example Font Size, or Colour) of an object that can be accessed by the DataEase developer – typically something which changes the behaviour or appearance of the object.

Object Properties can be modified in one of two ways:

a) Via the object's various design-time Dialog Boxes used to create or modify aspects of the object's Definition, Display, Font, Action, and Layout.

b) Via OMLt Scripts, which can change all the above values.

Data Model

The term **Data Model** is applied to the physical structure of the DataEase document(s) you are currently working with - the Tables and their Relationships. (Open the QBM Dialogue to see your current Data Model). The term **schema** describes the entire Data Model - every Table and Relationship in the database.

Multiview

The term **MultiView** describes a run-time snapshot of the actual data held in your current Data Model.

Simple Properties

A **Simple** property defines a single characteristic. For example, the **Visble** property defines when an object is visible (MyObject.Visible := 1) or invisible (MyObject.Visible := 0).

Class Properties

A **Class** Property does not directly define properties. Instead it uses either Simple Properties or other Class Properties to define properties.

For example, the Class Property **Fill** uses the Simple Properties **Hatch** and **Style**. But it also uses the Class Property **Color**. If you wanted to use **Fill** to change the **Style** of an object, the syntax would be: MyObject.fill.style := 3 .

But if you wanted to use **Fill** to change the colour of an object, the syntax would be:

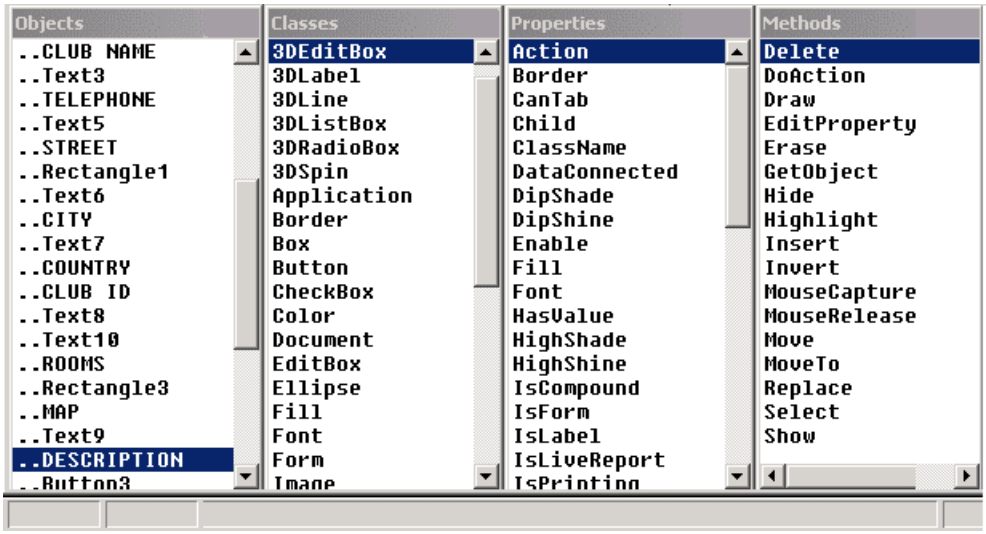
```
MyObject.fill.color.red := 128 .
MyObject.fill.color.green := 100 .
MyObject.fill.color.blue := 0 .
```

…because **Color** is itself a Class Property, defining three simple properties - **Red**, **Blue** and **Green**.

Defining an OML Script

You define an OML Script using the DQL/Event Editor - this being the normal DQL Editor, which has now been extended to include Event Scripting Pick Lists. These consist of Events, Objects, Classes, Properties and Methods. (If you're writing a procedure, you may want to shrink the Event Scripting Pick Lists, to leave more room for your query script).

Four of the new Pick Lists are shown below. The missing Pick List (Events) is at the top of the Script Editor screen. (It's shown on the next page).



The script editor is accessed by right-clicking on an object, and selecting "script" from the pop-up Express Menu. Once in the Script Editor, you can define scripts for as many objects as you like.

Each object can have several OML Scripts….in fact one for each Event that the object is capable of handling. Different Objects respond to different Events.

An Action Button object contains no data, and so it has no Events dependant on a change in data value, such as "ValueChange". An Edit Box has a large number of possible events, because there are a large number of ways in which the user can interact with it.

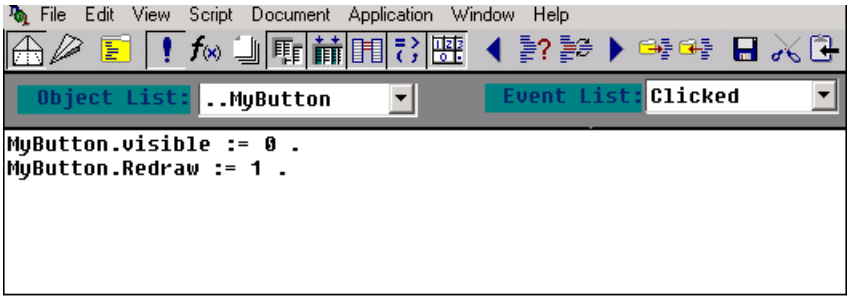
So, writing a very simple OML Script would involve just three steps:

1. In the Script Editor select (from the Object Pick List) a form object - say, a button.
2. Select (from the Event Pick List) an Event which will trigger your script - say "clicked".
3. Select the action which will take place when the Event is triggered - say, hide the button. This actually requires two lines of code. The first line sets the object's "visible" property to "0" - which means invisible. The second line sets the "Redraw" property to "1" - which forces the object to be immediately redrawn. This is necessary, because the "visible" property simply defines how the object will behave when it is next drawn - it doesn't cause the object to be redrawn.

The two lines of code are shown below:

```
MyButton.visible := 0 .
MyButton.Redraw := 1 .
```

…so the complete OML script would look like the screenshot shown below.



Syntax: Look at the first line and note the syntax. First we have the object name "MyButton" followed by the word "visible" ("Visible" is a Button Property). These two words are separated by a ".". Dot notation is used throughout the OML Script Editor. To set the property value we use the ":=" symbol, followed by the new value itself. Finally, the instruction is terminated by a "dot". (Note that there must be a space between the value and the closing dot).

You may have noticed that the Object List is displayed twice - once at the top of the screen, and once in a Pick List at the bottom of the screen. The 'top' Object List is positioned opposite the Event Pick List. You highlight an Object from the Object List, and then select the Event which will trigger your script. So you can think of these as **Object>>Event** pairs. You then write your script for the **Object>>Event** pair in the usual script area.

The 'bottom' Object Pick List is an aide-mémoire, showing you which objects are available for modification in your script.

Every Object can have a separate Event Script for **each** Event that applies to it…so you can write an awful lot of Event Scripts for each document.

Using the Pick Lists

- Click on an Object in the Object Pick List and the object's Events, Properties, and Methods will appear in the relevant Pick Lists. **But** be aware that the Event Pick List may sometimes contain Events which are inherited from higher Classes, and are not usable by the current object.
- Double-click on any item in the Pick List and the item will be pasted into the Script Editor at the current cursor position.
- DataEase contains both **Simple** and **Class** Properties. When you wish to define a Simple Property, you select it from the Property Pick List. When you wish to define a Class Property, you select it from the Class Pick List, whereupon the Class's Simple Properties will be displayed in the Properties Pick List.

  See the **Properties Introduction** for a full explanation of Simple and Class Properties.

Variables in Scripts

Local Variables

In DQL, you would manipulate variables by defining and assigning them as either Temporary (operates within the procedure only) or Global (operates outside the procedure) Variables, as shown in the examples below:

```
Define Temp "MyLocalCounter" number .

Define Global "GeneralCounter" number .
```

In OML Scripting, you assign a local variable using the syntax:

```
Define "MyLocalCounter" number .
```

…note that they keyword "temp" is not used. You can shorten your definition further, by using the syntax:

```
number MylocalCounter .
```

..instead of:

```
Define "MyLocalCounter" number .
```

..both examples work, but the longer method probably makes it easier for a maintenance programmer to recognise at a glance.

Global Variables

DataEase Global variables are not supported within OML Scripts, however you can create globally active variables by using CDF's, such as SetArray and GetArray. These CDF's ship with DataEase 6.x.

GetArray and SetArray are very easy to use. To use them, you must first register their CDF Library - CDFS2 - with DataEase. Once the library is loaded, you can use GetArray and SetArray whenever you want. They are briefly described below.

The SetArray funtion establishes a small ten element array in memory. The elements are numbered from 1 to 10, and each element holds a character string of up to 255 characters in length. You place values into the array with a command such as;

```
SetArray (1, "520") .

SetArray (4, "Somettext") .
```

Having created these values, you access them by using the GetArray CDF. The syntax for Getarray is;

```
MyField := GetArray(4).

MyNumberField := GetArray(1)
```

…which would retrieve the text value "SomeText" and place it into the field called MyField, and place the number "520" in the number field called MyNumberField.

Note that there are other CDFs available which define larger arrays.

DQL in OML Scripts

Before you start writing OML Scripts, you must fully understand the difference between the **MultiView** and the **Data Model**. We defined them earlier, but will repeat the definition now.

- The term **Data Model** is applied to the physical structure of the document(s) you are currently working with - the Tables and their Relationships. (Open the QBM Dialogue to see your current Data Model).
- The term **MultiView** describes a run-time snapshot of the actual data held in your current Data Model.

When writing an OMLt Script, you **CAN** modify the **MultiView**, but can **NOT** modify the **Data Model**. In other words you are allowed to enter, modify and delete data. But you are not allowed to modify the structure of that data - so you can't create new tables, or ad-hoc relationships, or permanent relationships between tables.

In addition to the OML Script commands themselves, a large subset of DQL commands and structures can be used within an OML Script. The following tables describe which statements and structures **CAN** and can **NOT** be used.

DQL can be broken down into the following categories:

Processing Commands

Sorting and Grouping Operators

Control Commands

Procedural Commands

Functions

Symbols and comparison operators

Relational Statistical Operators

…each of which is described in the following pages.

Processing Commands

Most Processing commands can be used inside an OML Script, but the "**For**" statement can **NOT** be used, so it is **NOT** possible to use structures such as;

```
For FORM (with some criteria)
  list records
  modify records
  enter a record
  delete records
```

…however you **CAN** use processing commands without the 'For' command, as in;

```
modify a record in
enter a record in
delete records in
```

The full Processing Commands rules are:

For	can **NOT** be used
List Records	can **NOT** be used
Input Using	can **NOT** be used
Export	can **NOT** be used
Lock	OK
Unlock	OK
Query Selection	OK
Modify records	OK - outside a For loop.
Enter a Record	OK - outside a For loop.
Delete a Record	OK - outside a For loop.

Sorting and Grouping Operators

Sorting and Grouping Operators are not allowed inside an OML Script.

In order	can **NOT** be used
In groups	can **NOT** be used
In reverse	can **NOT** be used
In groups with group totals	can **NOT** be used

Control Commands

Control Commands are all listed below. They can all be used from within an OML Script.

Run Procedure	OK
Call Menu	OK
Call Program	OK
Record Entry	OK
Import	OK
Reorganize	OK
Db Status	OK
Backup DB	OK
Restore DB	OK
Lock DB	OK
Unlock DB	OK
Install Application	OK
Application Status	OK
Lock	OK

Procedural Commands

Most (but not all) Procedural Commands **can** be used in an OML Script.

Output	Can **NOT** be used.
Message	OK
If	OK
Else	OK
End	OK
While	OK
Break	OK
Exit	OK
Case	OK
Value	OK
Others	OK
Define	OK, but only with a local OML Script Variable - not **Temp** or **Global**.
Assign	OK, but only with a local OML Script Variable - not **Temp** or **Globa**l

Note: Although Temporary and Global variables can not be used in an OML Script, OML Scripts have their own 'local' variables, which are essentially identical to Temporary variables. Global variables can be created by using the CDF's getarray, setarray and memarray32.

See the section "**Variables in Scripts** " for a full explanation.

DQL Functions

All 58 DQL Functions **CAN** be used. The list (by category) is;

Text	all OK
Date	all OK
Time	all OK
Spell	all OK
Text	all OK
If	is OK
Math	all OK
Financial	all OK
Scientific	all OK
Trigonometric	all OK

DQL symbols and comparison operators

All DQL symbols and comparison operators **CAN** be used. These consist of;

	(addition)	OK
-	(subtraction)	OK
/	(division)	OK
*	(multiplication)	OK
*	(asterisk)	OK
?	(question mark)	OK
~	(tilde)	OK
:	(colon)	OK
()	(parentheses)	OK
.	(period)	OK
;_	(semicolon)	OK
""	(quotation marks)	OK
:=	(assignment operator)	OK
<	(less than)	OK
=	(equals)	OK
>	(greater than)	OK
>=	(greater than or equal to)	OK
--	(comment)	OK

Relational Statistical Operators

All Relational Statistical Operators CAN be used in a Script. Note that you must use an existing relationship to retrieve summary data - you can **NOT** define an ad-hoc relationship in your Script.

Count of	OK
Highest of	OK
Sum of	OK
Mean of	OK
Lowest of	OK

Conditional Logic

You can apply conditional logic to your OML Scripts by using normal DQL commands. The example below contains a Rectangle object and a Button object. When the **Button>>Clicked Event** takes place, the script is run. An **if…then…else** construction toggles the Rectangle's Red colour on and off.

```
if Rectangle.Fill.Color.Red = 0  then
Rectangle.Fill.Color.Red := 255 .
rectangle.redraw := 1 .
else
Rectangle.Fill.Color.Red := 0 .
rectangle.redraw := 1 .
end .
```

Displaying Changes

When you change the visual characteristics of an object on screen, there are three different ways in which you can cause the object to be re-displayed.

The **Draw()** method ..MyObject.Draw() .

The **Show()** method ..MyObject.Show() .

The **ReDraw** property ..MyObject.redraw := 1 .

Each has a slightly different way of redrawing the object.

The **Draw()** Method draws the object over the top of the old object, and over the top of all other objects that might be stacked beneath or above it - so if the object was previously concealed by another object, then it is no longer concealed.

The **Show()** Method draws the object over the top of the old object, but maintains the object's position in a stack of objects - so if the object was previously concealed by another object, then it remains concealed.

The **ReDraw** Property deletes the old object, and then draws it again, but maintains the object's position in a stack of objects - so if the object was previously concealed by another object, then it remains concealed.

The difference between **ReDraw** and **Show()** is that **ReDraw** deletes the old object before redrawing it.

For example: Imagine you have a Rectangle Object, and you write an OML Script which reduces the object's size.

If you redisplay it by using the **Show()** Method - nothing will appear to happen. The object's new size is not apparent, because the old 'large' version is still displayed on screen.

If you set the **ReDraw := 1** property instead, then the object's new size will be immediately visible.

These three options allow you to pick the specific 'type' of screen repainting which suits your current requirement.

Lost Scripts

There are two ways in which your OML Scripts can be 'lost'.

- If you write OML Scripts on a Form - call it MyForm - and then make MyForm a subform of another form. The subform version of MyForm does **not** retain the original MyForm OML Scripts. However, MyForm itself does still possess the OML Scripts. This allows you to create new scripts for every occurrence off MyForm as a subform in other forms.
- If you create a form over an existing form, then any scripts belonging to the original will be duplicated in the copy. **Unless** you alter the layout of the 'copied' form before saving it. So if you open a form, move things around, then save it as a new form - expect your OML Scripts to be lost.

Error Messages

There are three Error Messages specific to OML Scripts. These error messages may appear if different OML Scripts conflict with each other at Runtime.

For example, you might write a Script which deleted an object. This script would compile correctly and would work correctly.

You might write a script which changed the colour of an object. This script would compile correctly and work correctly…..unless the Event which triggered it came **after** the event which deleted it.

The following three error messages deal with such conflicts.

- **Runtime Error. Attempt to use uninitialised variable xxxxxx**

This occurs when a variable returns NULL unexpectedly - testing for next object when you are the last object on the screen would be an example. Normally the name of the variable will be supplied at **xxxxx**.

- **Runtime Error. Attempt to use unavailable function xxxxx**

This appears whenever an attempt is made to call a function of any type for which the address cannot be found. Calling a function in an object that has been deleted at runtime would be an example. If available, the function name will be listed at **xxxxx**. Often, however, in those circumstances the name of the function is garbled, so the error message shown below may be substituted.

- **Runtime Error. Attempt to access object with unknown class.**

Again, most likely to occur if an object has been deleted at runtime and an attempt is made to call its functions or set its properties. This error message is

Is less informative, but avoids printing garbage characters instead of function names.

Introduction to Events

Events are the means by which the object sends information back to DataEase - specifically, the news that something has happened which might require attention. (More properly the object reports the news to Windows, which then reports to DataEase).

Not all Events are available to all Objects. For example, a "Line" Object is just a piece of window-dressing, used to improve the visual appearance of your document. It contains no value, and hence Events such as "PreEdit" and "ValueChange" are irrelevant to it, since it has no value to edit or change.

Since an understanding of Events is fundamental to writing OML Scripts, a lengthier description of Events is given below.

What are Events?

Pete Tabord, Sapphire's Head of Development, defines them from a DataEase perspective

“It is inherent in the Windows architecture that, unlike DOS, the machine essentially sits dormant until an event occurs. One does not run a program so much as register a program with Windows that will be handed control from Windows when certain things occur. So, in contrast to DOS, a programmer does not own the machine even when his program is ‘running’. He merely supplies code that will be accessed when certain external events occur.

The mechanism for this is the message. One creates message handlers that are fired up in response to Windows generating a message that an event has occurred. DataEase is not a monolithic program. It is composed of a database engine, a user interface, and a large number of objects, some internal (e.g. forms and reports) and some external DataEase object libraries, such as button, line, etc.

Many messages are handled by DataEase objects. Not all objects handle all events. DataEase 6 supplies the ability, in certain circumstances, for the application developer to add his own event handler in screen objects. You can only attach an OML Script to an event if the object in question has a handler for that event”.

The Event Return

Some Events - such as PostEdit, KeyInput and ValueChange - can effectively be run through a loop, by writing code which either accepts or rejects the Event. You do this by setting the **Event Return** value.

Example: Using the **Reservations** form in Club ParaDease, we write a script in the **KeyInput Event** on the **Rooms Required** field, which will provide us with a conditional Prevent data-entry. The script below will reject any user input which begins with "P".

```
if CLUB NAME.Value="P*"
then
message "You cannot enter data here" window .
return(2). /* Fail - do not allow input */
else
return(1). /* Success - allow input
```

Note: DataEase is unusual in that the event **return()** does not hold a Boolean value, and moreover treats the values "**0**" and "**1**" as both meaning **True.** Any result other than **0** or **1** means **False**. So if you are testing for a **False** condition, write **return(2)**. DataEase will recognise the **2** as being a False value.

..so the above piece of code would continually loop through the field's data-entry phase, until the condition was met.

Event Strings

Several events return a new value in the object's **string** property. String is a text property. If you wish to carry out mathematical operations or comparisons on **string**, then you need to work through a temporary variable.

Example: The code below is placed in a number field's PostEdit Event.

```
Define "MyVariable" number .
MyVariable := string .
if
MyOtherObject.value = "Discount"
then
MyVariable := MyVariable * 0.35 .
string := MyVariable .
redraw := 1 .
else
end .
```

...which conditionally (if MyOtherObject = discount) reduces the field's numerical content to 35% of its previous value.

List of Events

All the Events supported by DataEase 6 are listed and described below.

Event Name	Parameters	Description
Clicked	(Number ButtonID, Number xLoc, Number yLoc)	Fired when you perform a mouse click in the object. ButtonID: This indicates which button was pressed. 0 is left; 1 is right. The xLoc and yloc parameters (described below) are not available to the developer. xLoc the X co-ordinate in pixels of the mouse pointer when the event occurred. yLoc the Y co-ordinate in pixels of the mouse pointer when the event occurred. (0 , 0) is the top left-hand corner of the form.
DblClicked	(Number ButtonID, Number xLoc, Number yLoc)	Fires when you double-click your mouse in an object. ButtonID: This indicates which button was pressed. 0 is left; 1 is right. The xLoc and Yloc parameters (described below) are not available to the developer. xLoc the X co-ordinate in pixels of the mouse pointer when the event occurred. yLoc the Y co-ordinate in pixels of the mouse pointer when the event occurred. (0 , 0) is the top left-hand corner of the form.
DownArrowClicked	()	This Event occurs if the **Down Arrow** in a SpinBox or 3DSpinBox is clicked.
GotFocus	()	Fires when a field is entered. **Note**: If you put this event on the object which receives focus when a form is opened, it will **NOT** be triggered when the form opens. Technically the object already HAS focus, before the GotFocus Event can be evaluated.
PreEdit	()	Although PreEdit appears in the DataEase 6.0 Event Pick List, it is only available as a Method. This event/method occurs immediately after a field gets the focus and before a user enters data in it. The PreEdit event/method gets the field ready to be edited. For example, it allows the cursor to appear in the field, and highlights the text of existing values.

PostEdit	(Text String)	Fires after the field has been edited, whereupon the **String** parameter contains the new value. The object's **value** property still contains the old value. **Note**: PostEdit fires before the ValueChange Event. This means that it fires before the object's derivation formula (if any) is evaluated. So if the user types in "10", and the derivation formula multiplies the input value by 5, then **String** will hold "10", and not "50". So to modify the object's value, modify the **string** parameter. Use **Event Return** to either accept or reject the string value.
ValueLoaded	()	Triggered when a change is made to the field's value. This can result from a user editing the field, or from the action of a script or derivation, or simply pressing the F3 key. Note that the ValueLoaded Event fires twice for every record.
ValueChange	(Text String)	Fired when field loses focus and the field's value differs from the stored (originally loaded from the Table) value. The **String** parameter holds the new value, after all derivations and lookups have been performed. You can **not** modify a field's **value** property using this event - because it occurs after the field's derivations have been carried out. However you can check **value**, and either accept or reject it, by using the **Event Return** Flag.
ValueRequired	()	Can not be customised.
LostFocus	()	Fires when you exit from an object which previously held focus.
MouseDown	(Number ButtonID, Number xLoc, Number yLoc)	Fires when mouse button pressed. ButtonID: This indicates which button was pressed. 0 is left; 1 is right. The xLoc and Yloc parameters (described below) are not available to the developer. xLoc the X co-ordinate in pixels of the mouse pointer when the event occurred. yLoc the Y co-ordinate in pixels of the mouse pointer when the event occurred. (0 , 0) is the top left-hand corner of the form.

MouseUp	(Number ButtonID, Number xLoc, Number yLoc)	Fires when mouse button released. ButtonID: This indicates which button was pressed. 0 is left; 1 is right. The xLoc and Yloc parameters (described below) are not available to the developer. xLoc: The X co-ordinate in pixels of the mouse pointer when the event occurred. yLoc: The Y co-ordinate in pixels of the mouse pointer when the event occurred. (0 , 0) is the top left-hand corner of the form.
MouseMove	(Number xLoc, Number yLoc)	Fires when mouse moved in a visible object. The xLoc and Yloc parameters (described below) are not available to the developer. xLoc: The X co-ordinate in pixels of the mouse pointer when the event occurred. yLoc: The Y co-ordinate in pixels of the mouse pointer when the event occurred. (0 , 0) is the top left-hand corner of the form.
MouseOver	()	Fires when mouse passes over a visible object.
MouseEnter	()	Fires when mouse enters a visible object.
MouseExit	()	Fires when mouse exits a visible object.
KeyDown	(Number KeyValue)	Fires when an object has the focus and a keyboard key is pushed down. KeyValue: The ANSI number of the key pressed. (Introduced in 6.5).
KeyInput	(Number KeyValue)	Fires when a user presses a key that inputs a value while an object has the focus. KeyValue: The ANSI number of the key pressed. (Introduced in 6.5).
KeyUp	(Number KeyValue)	Fires when a user releases a key while an object has the focus. KeyValue: The ANSI number of the key pressed. (Introduced in 6.5).
UpArrowClicked	()	This Event occurs if the **Up Arrow** in a SpinBox or 3DSpinBox is clicked.

Keyboard Events

Usually the order of events is: **KeyDown**, **KeyInput** and **KeyUp**. However, there are certain keys for which there is no KeyInput event, and there are even a few keys that only trigger the KeyUp event.

Each event has an associated KeyValue event parameter, which is the ANSI number of the key concerned. The KeyValue for KeyDown and KeyUp is not modified by pressing other keys at the same time; the KeyInput KeyValue is modified by other keys. Principally, this means that, if the user holds down the Shift key and then presses 7 (ANSI 55), the KeyValue for both KeyDown and KeyUp will be 55, whereas KeyInput's KeyValue will be 38, which is the '&' character that appears above the '7' on that key.

For alphabetical characters, pressing 'A' for KeyDown and KeyUp gets a value of 65, which is the upper case ANSI value, where KeyInput has 97, the lower case value for 'a'.

All the following characters trigger KeyDown/KeyUp events only. Their KeyValue value is given in brackets:

Caps lock (20); Shift (16); Ctrl (17); Ctrl+Alt (18) (Alt on its own has no value); Windows button on left-hand side of keyboard (91); Windows button on right-hand side (92); Context button (93); ScollLock (145); Pause (19); Insert (45); Home (36); Page up (33); Delete (but not backspace) (46); End (35); Page Down (34); Up Cursor (38); Down Cursor (40); Left Cursor (37); Right Cursor (39); NumLock (144).

The behaviour of the function keys requires even more care. Generally, they are KeyDown/KeyUp events only, and have values from 112 for F1 to 123 for F12. If the function key has an action that is not for some reason disabled, or has no associated action, the KeyDown and KeyUp events will fire. But if, for example, F2-Save is disabled because the usage of the form is prevent data-entry, then only a KeyUp event occurs.

F10 triggers no events. Alt also usually has no value returned. Print Screen only fires a KeyUp event (with a KeyValue of 44).

Introduction to Methods

A Method is a function which is internal to an object. This function can be called externally.

Methods typically perform some sort of process relevant to the object, such as "refresh the object's screen display".

It is common practice to give a method fired by an event the same name as the triggering event. Hence an Event/Method consists of both an Event and a Method which, because of their relationship, share the same name.

List of Methods

Method Name	Parameters	Description
Delete		Deletes an object from the form. Should be used with caution! Events and derivation formula which refer to a deleted object will trigger error messages.
Draw	()	Draws an Object as the front-most object. (see **Show**).
DoAction		Can not be customised.
EditProperty		Can not be customised.
Erase	()	Erases an Object.
ExecuteVerbList		Can not be customised.
GetObject		Can not be customised.
GetVerbList		Can not be customised.
Hide	()	Hides an object. Removes an object from the display, setting its **visible** property to false (0)
Highlight	()	Removes the 'Highlight' border which normally surrounds an Object which has focus.
Insert		Can not be customised.
Invert		Can not be customised.
MouseCapture	()	Captures the mouse.
MouseRelease	()	Releases the mouse.
Move	(Number x, Number y)	Can not be customised. Moves an object **relative** to its current origin. X The distance in pixels to move horizontally. Y The distance in pixels to move vertically. Use positive numbers to move right and down, use negative numbers to move left and up. The object will be redrawn in front of any objects it overlaps, so you do not need to add a "show" instruction.
MoveTo	(Number xLoc, Number yLoc)	Can not be customised. Moves an object to an **absolute** location. X The pixel position on screen to move horizontally to. Y The pixel position on screen to move vertically to.

		This moves an object directly to a position on the current form. Negative numbers will move the control partly or completely off screen. The object will be redrawn in front of any objects it overlaps. Any objects the moved object overlaps will also be redrawn.
Replace		Can not be customised.
Select	()	Not implemented in 6.0. When implemented, Select will select an object - essentially gives focus to that object.
Show	()	Shows an object. **Show** retains an object's position in a stack of overlapping objects, while **draw** redraws the object as the front-most object. See **Displaying changes** for a lengthier description of the differences between Show and Draw.

Introduction to Properties

A property is a variable characteristic - for example the **Font Size**, or **Colour** - of an object that can be accessed by the DataEase developer – typically something which changes the behavior or appearance of the object.

In DataEase there are two basic types of property - the **Simple** Property, and the **Class** Property. (There is actually a third category - System Classes - consisting of **Application**, **Document**, **SuperString** and **VObject** - but this class is not available to the DataEase application developer).

A **Simple** property defines a single characteristic. For example, the **Visible** property defines when an object is visible (MyObject.Visible := 1) or invisible (MyObject.Visible := 0).

A **Class** Property does not directly define properties. Instead it uses either Simple Properties or other Class Properties to define properties.

For example, the Class Property **Fill** uses the Simple Properties **Hatch** and **Style**. But it also uses the Class Property **Color**. If you wanted to use **Fill** to change the **Style** of an object, the syntax would be:

```
MyObject.fill.style := 3 .
```

But if you wanted to use **Fill** to change the colour of an object, the syntax would be:

```
MyObject.fill.color.red := 128 .
MyObject.fill.color.green := 100 .
MyObject.fill.color.blue := 0 .
```

…Because **Color** is itself a Class Property, defining three simple properties - **Red**, **Blue** and **Green**.

Object Properties can be modified in one of two ways:

a) Via the object's various design-time Dialog Boxes used to create or modify aspects of the object's **Definition**, **Display**, **Font**, **Action,** and **Layout**.

b) Via OML Scripts, which can change all the values mentioned above, but **ONLY** while the object's owning form is open. If the form is closed and then re-opened, the object will return to its original defined properties.

This section of the Guide describes Simple Properties. Class Properties are described in the following section.

Property List

The table below summarizes the properties available in DataEase. The majority of these properties are Boolean, and hold a single Bit of information - a "0" or a "1", with "1" meaning **True** and "0" meaning **False**. A few of the properties hold Number or Text values.

Some properties shown in the table are **Class Properties**, meaning that they hold no values of their own. Instead they will define several **Simple Properties** - which do hold values. Class Properties are identified by having **(C)** after their name.

Properties can be either **Read Only** or **Read/Write**. As you will have guessed, you can not change the value of a **Read Only** property.

Properties	Read Only or Read/Write	Format	Description
Action	Read Only	0 or 1	Can not be customised..
Border (C)	Read/Write	-	See the **Border** Class Property.
Backpoint	Read/Write		This is a property of the 3DLine Object. If True (1) places an Arrow at the back of the line. (This is only really visible if you thicken the line). If False (0) no arrow is placed at the back of the line.
CanTab	Read Only	0 or 1	An object which can be tabbed (e.g. is capable of gaining focus), is True (1). An object which cannot be tabbed - such as an ellipse - is False (0).
Child (C)	Read Only	-	Most objects have a Parent - the Record Object. The Child property is true (1) for the first child of a record, and false (0) for subsequent children. Other children can be found by accessing the **Next** and **Prev** properties of the child. For example, a Record may contain a number of fields and other objects.. To work through the objects displayed within the record, start with the (first) child of the record. For each child, locate the **Next** control until no more controls are found.
ClassName	Read Only	Text	Returns the Class Name of the object - SpinBox, Rectangle, Ellipse, etc.
Delay	Read/Write	Number	Sets the spin rate in a spin box. Measured in 100's of a second.
DataConnected	Read Only	0 or 1	This is True (1) if the object can contain data - such as an Edit Box - and False (0) if the object contains no data - such as an Ellipse.
DipShade (C)	Read/Write	-	See the Dipshade Class Property

DipShine (C)	Read/Write	-	See the Dipshine Class Property
Enable	Read/Write	0 or 1	Turns OML Scripting for an object on and off, with True (1) meaning that OML Scripts on the object will be run, and False (0) meaning that OML Scripts will be disabled.
Fill (C)	Read/Write	-	See the **Fill** Class Property.
Font (C)	Read/Write	-	See the **Font** Class Property.
FrontPoint	Read/Write	0 or 1	This is a property of the 3DLine Object. If True (1) places an Arrow at the front of the line. (This is only really visible if you thicken the line). If False (0) no arrow is placed at the front of the line.
HasValue	Read Only	0 or 1	If True (1), the object is capable of storing a Value, and if False (0) the object is not capable of storing a value. Similar to data-connected property.
HighShade (C)	Read/Write	-	See the **HighShade** 3D Class Property
HighShine (C)	Read/Write	-	See the **HighShine** 3D Class Property.
IsCompound	Read Only	0 or 1	Reads True (1) if the object has a child, and False (0) if the object has no children. So a Form with a Subform would read True.
IsForm	Read Only	0 or 1	True (1) means the object is a form, False (0) means it isn't.
IsLabel	Read Only	0 or 1	True (1) means the object is a label, False (0) means it isn't.
IsLiveReport	Read Only	0 or 1	True (1) means the object is a live report, False (0) means it isn't.
IsPrinting	Read Only	0 or 1	True (1) means the object is printing, False (0) means it isn't.
IsReport	Read Only	0 or 1	True (1) means the object is a report, False (0) means it isn't.
IsRecord	Read Only	0 or 1	True (1) means the object is a record, False (0) means it isn't.
IsSelected	Read Only	0 or 1	When True (1), this object has the focus.
IsTableView	Read Only	0 or 1	Returns True (1) if the document is in Table View, and False (0) if it is in Form View.
Justify	Read/Write	0 or 1	Used in the Label and 3D Label objects. True (1) justifies text, and False (0) leaves it unjustified.
LowShade (C)	Read/Write	-	See the 3D Class Property **LowShade**
LowShine (C)	Read/Write	-	See the 3D Class Property **LowShine**
Name	Read Only	Text	Contains the name of the object. (Read Only, to avoid chaos!).
Next (C)	Read Only	0 or 1	See the **Next** Class Property

Parent (C)	Read Only	-	See the **Parent** Class Property
Prev (C)	Read Only	0 or 1	See the **Prev** Class property
PropertyList	Read Only	Text	Not implemented.
Rect (C)	Read/Write	-	See the **Rect** Class Property
RectBorder (C)	Read/Write	-	See the **RectBorder** Class object.
RectFill (C)	Read/Write	-	See the **RectFill** Class Object
ReDraw	Read/Write	0 or 1	If True (1) forces the object to be deleted and then redrawn on screen.
RimShade (C)	Read/Write	-	See the 3D Class Property **RimShade**.
RimShine (C)	Read/Write	-	See the 3D Class Property **RimShine**
RoundedCorners	Read/Write	0 or 1	When True (1) the corners of a rectangle are rounded, and when False (0) they are squared off.
Shadow (C)	Read/Write	-	See the **Shadow** 3D Class Property
Shade (C)	Read/Write	-	Can not be customised..
Shine (C)	Read/Write	-	Can not be customised..
ScrollBorder	Read/Write	-	See the **ScrollBorder** Class Property.
ScrollFill	Read/Write	-	See the **ScrollFill** Class property.
Taborder	Read Only	Number	Holds the Tab Order position of this object. Returns **-1** if no tab position has been set.
TextString	Read/Write	Text	The actual text within a Text Label
TextMargin	Read/Write	Number	Sets the text margin in a 3D field.
TextSlantinDegrees	Read/Write	Number	The angle of slant in a 3D Text object.
TextShade (C)	Read/Write	-	See the **TextShade** 3D Class Property
TextShine (C)	Read/Write	-	See the **TextShine** 3D Class Property
Thickness	Read/Write	Number	Defines the thickness (in pixels) of a 3D Line Object.
Type	Read Only	Number	Returns the Class Number of an object. (Each Class Object has a Number as well as a name).
Value	Read/Write	Text	Contains the Value held in the object. Only data-connected objects have a Value.
View	Read/Write		Can not be customised..
Visible	Read/Write	0 or 1	When Visible is True (1) The object can be seen. When False (0) the object is hidden.
Wrap	Read/Write	0 or 1	If True (1), wraps the text in a Label or 3DLabel object. If False (0), does not wrap the text.

Introduction to Objects

An object is something you place on a DataEase document - a field, a box, a picture, a summary variable, and so on.

Every object responds to certain types of Events - the actual list of Events varies from one object to another. Similarly, each object possesses Methods and Properties which can be used with it.

All the objects used in DataEase are listed below. Click on any object to see a description of its Events, Methods and Properties.

Some, but not all, objects are available in both "normal" and "3D" versions. They behave identically, but the 3D versions have additional display characteristics.

Objects have many associated properties - size, colour, position, content, and so on. These properties are initially specified at design time, but most properties can subsequently be changed by an OML Script.

List of Objects

All the objects used in DataEase are listed below. Click on any object to see a description of its Events, Methods and Properties.

3DcheckBox	**Checkbox**	**Application Variable**
3DeditBox	**EditBox**	**OLE**
3Dline	**Ellipse**	**RadioBox**
3DlistBox	**Image**	**3DradioBox**
ImageField	**Rectangle**	**Spin**
3Dspin	**Label**	**3Dlabel**
Line	**Button**	**ListBox**

3DcheckBox (3DRadioButton)

A 3DCheckBox displays choices in a single box that toggles between Yes (selected) and No (deselected). The field contains a blank value (default) until you click on the check box. When you create a CheckBox field, DataEase displays the field name as the caption to the right of the check box.

The 3D CheckBox behaves in exactly the same way as a CheckBox, except that it contains enhanced Display properties to make it more attractive or eye catching.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**KeyDown**	**KeyInput**
	KeyUp	**ValueLoaded**	**ValueChange**
	ValueRequired		
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**(C)Border**	**CanTab**
	Child	**ClassName**	**DataConnected**
	(C)DipShade	**(C)DipShine**	**Enable**
	(C)Fill	**(C)Font**	**HasValue**
	(C)HighShade	**(C)HighShine**	**IsCompound**
	IsForm	**IsLabel**	**IsReport**
	IsPrinting	**IsRecord**	**IsSelected**
	IsTableView	**(C)LowShade**	**(C)LowShine**
	Name	**Next**	**Parent**
	Prev	**PropertyList**	**(C)Rect**
	ReDraw	**(C)RimShade**	**(C)RimShine**
	RoundedCorners	**(C)Shadow**	**Taborder**
	TextString	**TextMargin**	**TextSlantinDegrees**
	(C)TextShade	**(C)TextShine**	**Type**
	Value	**View**	**Visible**

3DeditBox

A 3D EditBox displays alphanumeric values in a rectangular field. It is identical to an Editbox, except that it contains enhanced Display properties to make it more attractive or eye catching.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**PostEdit**	**PreEdit**
	KeyDown	**KeyInput**	**KeyUp**
	ValueLoaded	**ValueChange**	**ValueRequired**
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**(C)Border**	**CanTab**
	Child	**ClassName**	**DataConnected**
	(C)DipShade	**(C)DipShine**	**Enable**
	(C)Fill	**(C)Font**	**HasValue**
	(C)HighShade	**(C)HighShine**	**IsCompound**
	IsForm	**IsLabel**	**IsReport**
	IsPrinting	**IsRecord**	**IsSelected**
	IsTableView	**(C)LowShade**	**(C)LowShine**
	Name	**Next**	**Parent**
	Prev	**PropertyList**	**(C)Rect**
	ReDraw	**(C)RimShade**	**(C)RimShine**
	RoundedCorners	**(C)Shadow**	**Taborder**
	TextString	**TextMargin**	**TextSlantinDegrees**
	(C)TextShade	**(C)TextShine**	**Type**
	Value	**View**	**Visible**

3Dlabel

A 3DLabel - also known as a Text object - is used for placing Text on a document. The text is attached to the Document, not to individual records in the document, and is used for Headings, descriptive field labels, and so on. A 3Dlabel is identical to a Label, except that it contains enhanced Display properties to make it more attractive or eye catching.

Events:	**MouseDown**	**MouseUp**	**MouseMove**
	MouseOver	**MouseEnter**	**MouseExit**
	KeyDown	**KeyInput**	**KeyUp**
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**(C)Border**	**CanTab**
	Child	**ClassName**	**DataConnected**
	(C)DipShade	**(C)DipShine**	**Enable**
	(C)Fill	**(C)Font**	**HasValue**
	(C)HighShade	**(C)HighShine**	**IsCompound**
	IsForm	**IsLabel**	**IsReport**
	IsPrinting	**IsRecord**	**IsSelected**
	IsTableView	**(C)LowShade**	**(C)LowShine**
	Justify	**Name**	**Next**
	Parent	**Prev**	**PropertyList**
	(C)Rect	**ReDraw**	**(C)RimShade**
	(C)RimShine	**RoundedCorners**	**(C)Shadow**
	Taborder	**TextString**	**TextMargin**
	TextSlantinDegrees	**(C)TextShade**	**(C)TextShine**
	Type	**Value**	**View**
	Visible	**Wrap**	

3Dline

A 3Dline object is a simple graphical shape - in this case a line - placed on a document. A 3DLine is identical to a Line, except that it contains enhanced Display properties to make it more attractive or eye catching.

Events:	**MouseDown**	**MouseUp**	**MouseMove**
	MouseOver	**MouseEnter**	**MouseExit**
	KeyDown	**KeyInput**	**KeyUp**
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**(C)Border**	**Backpoint**
	CanTab	**Child**	**ClassName**
	DataConnected	**(C)DipShade**	**(C)DipShine**
	Enable	**(C)Fill**	**(C)Font**
	FrontPoint	**HasValue**	**(C)HighShade**
	(C)HighShine	**IsCompound**	**IsForm**
	IsLabel	**IsReport**	**IsPrinting**
	IsRecord	**IsSelected**	**IsTableView**
	(C)LowShade	**(C)LwShine**	**Name**
	Next	**Parent**	**Prev**
	PropertyList	**(C)Rect**	**ReDraw**
	(C)RimShade	**(C)RimShine**	**RoundedCorners**
	(C)Shade	**(C)Shine**	**(C)Shadow**
	Taborder	**TextString**	**TextMargin**
	TextSlantinDegrees	**(C)TextShade**	**(C)TextShine**
	Thickness	**Type**	**Value**
	View	**Visible**	

3DlistBox

The ListBox displays choices in a drop-down list. The 3D ListBox is identical to ListBox, except that it contains enhanced Display properties to make it more attractive or eye catching.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**PostEdit**	**KeyDown**
	KeyInput	**KeyUp**	**ValueLoaded**
	ValueChange	**ValueRequired**	
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**(C)Border**	**CanTab**
	Child	**ClassName**	**DataConnected**
	(C)DipShade	**(C)DipShine**	**Enable**
	(C)Fill	**(C)Font**	**HasValue**
	(C)HighShade	**(C)HighShine**	**IsCompound**
	IsForm	**IsLabel**	**IsReport**
	IsPrinting	**IsRecord**	**IsSelected**
	IsTableView	**(C)LowShade**	**(C)LowShine**
	Name	**Next**	**Parent**
	Prev	**PropertyList**	**(C)Rect**
	ReDraw	**(C)RimShade**	**(C)RimShine**
	RoundedCorners	**(C)Shadow**	**Taborder**
	TextString	**TextMargin**	**TextSlantinDegrees**
	(C)TextShade	**(C)TextShine**	**Type**
	Value	**View**	**Visible**

3DradioBox

The RadioBox displays choices as radio buttons. The 3DradioBox is identical to RadioBox, except that it contains enhanced Display properties to make it more attractive or eye catching.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**KeyDown**	**KeyInput**
	KeyUp	**ValueLoaded**	**ValueChange**
	ValueRequired		
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**(C)Border**	**CanTab**
	Child	**ClassName**	**DataConnected**
	(C)DipShade	**(C)DipShine**	**Enable**
	(C)Fill	**(C)Font**	**HasValue**
	(C)HighShade	**(C)HighShine**	**IsCompound**
	IsForm	**IsLabel**	**IsReport**
	IsPrinting	**IsRecord**	**IsSelected**
	IsTableView	**(C)LowShade**	**(C)LowShine**
	Name	**Next**	**Parent**
	Prev	**PropertyList**	**(C)Rect**
	ReDraw	**(C)RimShade**	**(C)RimShine**
	RoundedCorners	**(C)Shadow**	**Taborder**
	TextString	**TextMargin**	**TextSlantinDegrees**
	(C)TextShade	**(C)TextShine**	**Type**
	Value	**View**	**Visible**

3Dspin

A 3DSpinBox displays numeric values. Type in a value or use the vertical arrow controls to display a value. A 3D Spinbox is identical to a Spinbox, except that it contains enhanced Display properties to make it more attractive or eye catching.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**PostEdit**	**PreEdit**
	KeyDown	**KeyInput**	**KeyUp**
	ValueLoaded	**ValueChange**	**ValueRequired**
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**CanTab**	**Child**
	ClassName	**Delay**	**DataConnected**
	(C)DipShade	**(C)DipShine**	**Enable**
	(C)Fill	**(C)Font**	**HasValue**
	(C)HighShade	**(C)HighShine**	**IsCompound**
	IsForm	**IsLabel**	**IsReport**
	IsPrinting	**IsRecord**	**IsSelected**
	IsTableView	**(C)LowShade**	**(C)LwShine**
	Name	**Next**	**Parent**
	Prev	**PropertyList**	**(C)Rect**
	RectBorder	**RectFill**	**ReDraw**
	(C)RimShade	**(C)RimShine**	**RoundedCorners**
	ScrollBorder	**ScrollFill**	**(C)Shade**
	(C)Shine	**(C)Shadow**	**Taborder**
	TextString	**TextMargin**	**TextSlantinDegrees**
	(C)TextShade	**(C)TextShine**	**Type**
	Value	**View**	**Visible**

Application Variable

An Application Variable object displays system-generated information (e.g. current date, current time, current page number), rather than displaying any data from the database itself.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**KeyDown**	**KeyInput**
	KeyUp		

Methods: **-**

Properties: **Name**

Button

A Button is a graphic object that performs an action (such as displaying or hiding the Toolbar) when it is clicked in user View.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**KeyDown**	**KeyInput**
	KeyUp		
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**(C)Border**	**CanTab**
	Child	**ClassName**	**DataConnected**
	Enable	**(C)Fill**	**(C)Font**
	HasValue	**IsCompound**	**IsForm**
	IsLabel	**IsReport**	**IsPrinting**
	IsRecord	**IsSelected**	**IsTableView**
	Name	**Next**	**Parent**
	Prev	**PropertyList**	**(C)Rect**
	ReDraw	**RoundedCorners**	**Taborder**
	TextString	**TextMargin**	**TextSlantinDegrees**
	Type	**Value**	**View**
	Visible		

Checkbox

A CheckBox displays choices in a single box that toggles between Yes (selected) and No (deselected). The field contains a blank value (default) until you click on the check box. When you create a CheckBox field, DataEase displays the field name as the caption to the right of the check box.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**KeyDown**	**KeyInput**
	KeyUp		
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**CanTab**	**Child**
	ClassName	**DataConnected**	**Enable**
	HasValue	**IsCompound**	**IsForm**
	IsLabel	**IsReport**	**IsPrinting**
	IsRecord	**IsSelected**	**IsTableView**
	Name	**Next**	**Parent**
	Prev	**PropertyList**	**(C)Rect**
	ReDraw	**Taborder**	**Type**
	Value	**View**	**Visible**

EditBox

An EditBox displays alphanumeric values in a rectangular field.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**PostEdit**	**PreEdit**
	KeyDown	**KeyInput**	**KeyUp**
	ValueLoaded	**ValueChange**	**ValueRequired**
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**Border**	**CanTab**
	Child	**ClassName**	**DataConnected**
	Enable	**(C)Fill**	**(C)Font**
	HasValue	**IsCompound**	**IsForm**
	IsLabel	**IsReport**	**IsPrinting**
	IsRecord	**IsSelected**	**IsTableView**
	Name	**Next**	**Parent**
	Prev	**PropertyList**	**(C)Rect**
	ReDraw	**Taborder**	**Type**
	Value	**View**	**Visible**

Ellipse

An Ellipse object is a simple graphical shape - in this case an ellipse - placed on a document.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**KeyDown**	**KeyInput**
	KeyUp		
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**CanTab**	**Child**
	ClassName	**DataConnected**	**Enable**
	HasValue	**IsCompound**	**IsForm**
	IsLabel	**IsReport**	**IsPrinting**
	IsRecord	**IsSelected**	**IsTableView**
	Name	**Next**	**Parent**
	Prev	**PropertyList**	**(C)Rect**
	ReDraw	**Taborder**	**Type**
	Value	**View**	**Visible**

Image (Picture)

An Image field (or Picture field) is a background picture that appears the same on each record, because it is attached to the Form, and not to individual Records within the form. Picture Objects can have Actions assigned to them, so that they behave like Buttons.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**KeyDown**	**KeyInput**
	KeyUp	**ValueLoaded**	**ValueChange**
	ValueRequired		
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**CanTab**	**Child**
	ClassName	**DataConnected**	**Enable**
	HasValue	**IsCompound**	**IsForm**
	IsLabel	**IsReport**	**IsPrinting**
	IsRecord	**IsSelected**	**IsTableView**
	Name	**Next**	**Parent**
	Prev	**PropertyList**	**(C)Rect**
	ReDraw	**Taborder**	**Type**
	Value	**View**	**Visible**

ImageField

An Imagefield accepts an image filename and displays graphic data in User View. ImageFields are tied to individual records in a form, and are used for Staff Records, Part Inventories, and so on.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**KeyDown**	**KeyInput**
	KeyUp	**ValueLoaded**	**ValueChange**
	ValueRequired		
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**CanTab**	**Child**
	ClassName	**DataConnected**	**Enable**
	HasValue	**IsCompound**	**IsForm**
	IsLabel	**IsReport**	**IsPrinting**
	IsRecord	**IsSelected**	**IsTableView**
	Name	**Next**	**Parent**
	Prev	**PropertyList**	**(C)Rect**
	ReDraw	**Taborder**	**Type**
	Value	**View**	**Visible**

Label

A Label - also known as a Text object - is used for placing Text on a document. The text is attached to the Document, not to individual records in the document, and is used for Headings, descriptive field labels, and so on.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**KeyDown**	**KeyInput**
	KeyUp		
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditPropert**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**Border**	**CanTab**
	Child	**ClassName**	**DataConnected**
	Enable	**(C)Fill**	**(C)Font**
	HasValue	**IsCompound**	**IsForm**
	IsLabel	**IsReport**	**IsPrinting**
	IsRecord	**IsSelected**	**IsTableView**
	Name	**Next**	**Parent**
	Prev	**PropertyList**	**(C)Rect**
	ReDraw	**Taborder**	**Type**
	Value	**View**	**Visible**

Line

A Line object is a simple graphical shape - in this case a line - placed on a document.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**KeyDown**	**KeyInput**
	KeyUp		
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**CanTab**	**Child**
	ClassName	**DataConnected**	**(C)DipShade**
	HasValue	**IsCompound**	**IsForm**
	IsLabel	**IsReport**	**IsPrinting**
	IsRecord	**IsSelected**	**IsTableView**
	Name	**Next**	**Parent**
	Prev	**PropertyList**	**(C)Rect**
	ReDraw	**Taborder**	**Type**
	View	**Visible**	

ListBox

The ListBox displays choices in a drop-down list.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**KeyDown**	**KeyInput**
	KeyUp	**ValueLoaded**	**ValueChange**
	ValueRequired		
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**CanTab**	**Child**
	ClassName	**DataConnected**	**Enable**
	HasValue	**IsCompound**	**IsForm**
	IsLabel	**IsReport**	**IsPrinting**
	IsRecord	**IsSelected**	**IsTableView**
	Name	**Next**	**Parent**
	Prev	**PropertyList**	**(C)Rect**
	ReDraw	**Taborder**	**Type**
	Value	**View**	**Visible**

OLE

An OLE Object is generated by an external software application, such as Microsoft Excel, and either embedded or linked in a container document, such as a DataEase form.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**KeyDown**	**KeyInput**
	KeyUp		
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**CanTab**	**Child**
	ClassName	**DataConnected**	**Enable**
	HasValue	**IsCompound**	**IsForm**
	IsLabel	**IsReport**	**IsPrinting**
	IsRecord	**IsSelected**	**IsTableView**
	Name	**Next**	**Parent**
	Prev	**PropertyList**	**(C)Rect**
	ReDraw	**Taborder**	**Type**
	View	**Visible**	

RadioBox (RadioButton)

The RadioBox displays choices as radio buttons.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**KeyDown**	**KeyInput**
	KeyUp	**ValueLoaded**	**ValueChange**
	ValueRequired		
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**CanTab**	**Child**
	ClassName	**DataConnected**	**Enable**
	HasValue	**IsCompound**	**IsForm**
	IsLabel	**IsReport**	**IsPrinting**
	IsRecord	**IsSelected**	**IsTableView**
	Name	**Next**	**Parent**
	Prev	**PropertyList**	**(C)Rect**
	ReDraw	**Taborder**	**Type**
	Value	**View**	**Visible**

Rectangle (Box Class Object)

A Rectangle object is a simple graphical shape - in this case a rectangle - placed on a document. Do not confuse it with the **Rect** Class property.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**KeyDown**	**KeyInput**
	KeyUp		
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**Border**	**CanTab**
	Child	**ClassName**	**DataConnected**
	Enable	**(C)Fill**	**HasValue**
	IsCompound	**IsForm**	**IsLabel**
	IsReport	**IsPrinting**	**IsRecord**
	IsSelected	**IsTableView**	**Name**
	Next	**Parent**	**Prev**
	PropertyList	**(C)Rect**	**RoundedCorners**
	ReDraw	**Taborder**	**Type**
	View	**Visible**	

Spin (Spin Button)

A SpinBox (also called a SpinButton) displays numeric values. Type in a value or use the vertical arrow controls to display a value.

Events:	**Clicked**	**DownArrowClicked**	**DblClicked**
	GotFocus	**LostFocus**	**MouseDown**
	MouseUp	**MouseMove**	**MouseOver**
	MouseEnter	**MouseExit**	**KeyDown**
	KeyInput	**KeyUp**	**UpArrowClicked**
	ValueLoaded	**ValueChange**	**ValueRequired**
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**CanTab**	**Child**
	ClassName	**DataConnected**	**Delay**
	Enable	**(C)Font**	**HasValue**
	IsCompound	**IsForm**	**IsLabel**
	IsReport	**IsPrinting**	**IsRecord**
	IsSelected	**IsTableView**	**Name**
	Next	**Parent**	**Prev**
	PropertyList	**(C)Rect**	**RectBorder**
	RectFill	**ReDraw**	**Taborder**
	Type	**Value**	**View**
	Visible		

Summary

A Summary Variable object displays summary information (e.g. page totals and running totals) based on the application data. You can create Summary Variable objects in Headers and Footers only.

Events:	**Clicked**	**DblClicked**	**GotFocus**
	LostFocus	**MouseDown**	**MouseUp**
	MouseMove	**MouseOver**	**MouseEnter**
	MouseExit	**KeyDown**	**KeyInput**
	KeyUp		
Methods:	**Delete**	**DoAction**	**Draw**
	Erase	**EditProperty**	**GetObject**
	Hide	**Highlight**	**Insert**
	Invert	**MouseCapture**	**Move**
	MoveTo	**MouseRelease**	**Replace**
	Select	**Show**	
Properties:	**Action**	**CanTab**	**Child**
	ClassName	**DataConnected**	**Enable**
	HasValue	**IsCompound**	**IsForm**
	IsLabel	**IsReport**	**IsPrinting**
	IsRecord	**IsSelected**	**IsTableView**
	Name	**Next**	**Parent**
	Prev	**PropertyList**	**(C)Rect**
	ReDraw	**Taborder**	**Type**
	View	**Visible**	

Class Properties

A **Class** Property does not directly define properties. Instead it uses either Simple Properties or other Class Properties to define properties. There are fifteen general Class Properties in DataEase 6, plus a further eleven Class Properties which are specific to 3D objects. These twenty six Class Properties are described in the following pages.

Font (Class Property)

Objects which display text - either as labels or textual and numeric values in fields - all have a Font Class Property. By changing Font settings, you can alter a piece of text's size, style, and other such details.

The Font Class Property has seven Simple Properties, namely;

Bold	Size	CharSet	Under	Italic	Name	Strike

Two other properties - **Width** and **Height** - may appear in the Properties Pick List. These are not currently used, and should be ignored.

As with all Class Properties, you set them by specifying the Object name, followed by the Class Name, followed by the Simple Property Name, followed by the value, e.g.

```
MyTextLabel.Font.Bold := 1 .
```

When you change a Font property, the change is not displayed until the object is redrawn, either as a result of a user action, or through a script that uses the Show or Draw method.

The seven Simple Properties are now described.

Bold

This read/write flag determines if text in a font is displayed in bold.

Type

Number (Boolean).

Details

If the Bold property is true (1), text displayed using this font object appears in boldface; if false (0), it does not.

Example

```
MyTextLabel.Font.Bold := 1 .
```

Size

This read/write property is the size of a font, expressed in points.

Type

Number.

Details

The value must be a whole number; half-point and other fractions are not allowed. If you try to set a fractional size via a script, the actual size is rounded up to the next integer.

You can choose any positive integer; you are not restricted to those sizes that appear on the pre-set Size picklist in the Fonts dialog. Sizes above 400 points are not recommended for on-screen use for practical reasons.

Negative values are converted to positive values. If the Size is set larger than the space in the control allows, the text is clipped.

Example

```
MyTextLabel.Font.Size := 12 .
```

Italic

This read/write flag determines whether text in a font is displayed in italics.

Type

Number (Boolean).

Details

If the Italic property is true (1), text displayed using this Font object appears in italics; if false (0), it does not.

Example

```
MyTextLabel.Font.Italic := 1 .
```

Name

This read/write property is the name of a font.

Type

Text.

Details

This identifies the name of the font (typeface) of the Font object. The Font properties of an object are initially set when you define the object, but your script can subsequently modify them - perhaps to add visual emphasis to an object.

The name is the name as it appears in the Font Name picklist of the Fonts dialog box.

Note that fonts are treated as belonging to one of two sets: characters or symbols. Character fonts are most of the fonts you will use, such as Times New Roman and Arial: Symbols are fonts like Symbol and WindDings.

You can only change fonts from the same type of set. For example, you can switch from Arial to Arial Narrow, but not from Arial to Symbol.

If the font name is not recognized (for example, it is mis-typed or not installed on the computer in use), DataEase substitutes Times if the font was previously character-based, and Symbol if it was previously symbol-based. It will also switch to these fonts if you try to swap between the two sets, e.g. change from WingDings to Courier.

Example

MyTextLabel.Font.Name := Arial .

Strike

This read/write flag indicates whether text in a font is displayed with strikethrough.

Type

Number (Boolean).

Details

If the Strike property is true (1), text displayed using this font object appears with a strikethrough; if false (0), it does not.

Example

```
MyTextLabel.Font.Strike := 0 .
```

Under

This read/write flag determines if text in a font is underlined.

Type

Number (Boolean)

Details

If the Under property is true (1), text displayed using this font object appears with underlining; if false (0), it does not.

Example

```
MyTextLabel.Font.Under := 0 .
```

CharSet

Sets the **Character Set** to be used.

Type

Number

Details

There are three possible values: 0, 1 or 2.

0 …means use the standard Western Character Set. (Default value).

1 …means use the System Default Character Set.

2 …means use a Symbol Character Set.

Example

MyField.font.Charset := 2 .

MyField.Font.Name := Wingdings .

Note: Selecting the Symbol Character Set is only half the job. You then have to specify which **Font** you want to use. In the above example we have selected a Symbol Character Set, and then asked for the Wingdings font.

Color (Class Property)

The Color class property is used to change the colour of an object, such as a box, a line, or a piece of text. You change the colour by setting values for Blue, Red, and Green - the three RBG values from which all colours are derived in computer graphics. Values are in the range 0-255.

For example, setting Green=255, Red=0 and Blue=0 would produce a green colour. Setting Green=255, Red=255 and Blue=255 would produce white, while Green=0, Red=0 and Blue=0 would produce black.

The Color Class is never called on its own, so you can not write a script such as:

```
CreditBox.Color.Red := 255 .
```

Instead, the Color Class is always called by other Object Classes - such as the **Border** and **Fill** Class Properties.

Imagine you had a "Credit Worthy" box, normally displayed in green. You could create a script which checked a client's credit rating, and if the rating was poor, the script could alter the "Credit Worthy" Box to a Red display. The syntax would be:

```
If any CLUBS Credit_Rating = "No Way!" then
CreditBox.Fill.Color.Red := 255 .
CreditBox.Fill.Color.Green := 0 .
CreditBox.Fill.Color.Blue := 0 .
CreditBox.Redraw := 1 .
else
CreditBox.Fill.Color.Red := 0 .
CreditBox.Fill.Color.Green := 255 .
CreditBox.Fill.Color.Blue := 0 .
CreditBox.Redraw := 1 .
End .
```

…the "Redraw" command is necessary, to force the colour change to be displayed on screen.

With a little extra work, you could - depending on the client's exact credit worthiness - gradually shade the Credit Worthy Box from green to red, by decreasing and increasing the relevant colours.

As mentioned above, the three simple properties associated with the Color Class Property are:

Blue

Green

Red

Examples of their use are given overpage.

Blue
This property specifies the saturation of blue coloring.

Type
Number.

Details:
Set this property to a value between 0 and 255. 0 specifies minimum saturation, 255 specifies maximum.

Example

```
CreditBox.fill.Color.Blue := 0 .
```

Green
This property specifies the saturation of green coloring.

Type
Number.

Details:
Set this property to a value between 0 and 255. 0 specifies minimum saturation, 255 specifies maximum.

Example

```
CreditBox.fill.Color.Green := 10 .
```

Red
This property specifies the saturation of red coloring.

Type
Number.

Details:
Set this property to a value between 0 and 255. 0 specifies minimum saturation, 255 specifies maximum.

Example

```
CreditBox.fill.Color.Red := 128 .
```

Fill (Class Property)

Most objects use the Fill Class Property to set the following three properties of an object.

Color (Color is itself a Class Property - see **Color**

Style

Hatch

A fill object may be made transparent by setting its Style property to 1. An object can have a solid fill of a particular color, a fill of a specified pattern, or no fill.

The effect of changing an object's Fill property is not displayed until the object is redrawn, either as the result of a user action or through a script that redraws the object.

Style

Style holds a number indicating whether the object is transparent, opaque, or hatched. If Style is hatched, then Hatch is a number indicating how the control face is painted (such as solid, diagonal lines, or vertical lines).

Type

Number:

Details

0 = Solid

1 = No Fill (transparent)

2 = Hatch (patterned)

Example

```
MyObject.fill.style := 2 .
```

Hatch

This read/write property controls the type of pattern to use for a fill.

Type

Number:

Details

0 = Horizontal Lines1 = Vertical Lines2 = Diagonal from top left to bottom right3 = Diagonal from top right to bottom left4 = Grid Hatch5 = Cross Hatch

Example

```
MyObject.fill.hatch := 3 .
```

Remember that you can only set the hatch property if the Style is set to Hatch (2), so the full example would be:

```
MyObject.fill.style := 2 .
MyObject.fill.hatch := 3 .
MyObject.redraw := 1 .
```

Color

Color is itself a Class Property. See Color.

Border (Class Property)

The Border class is used to define borders around the perimeters of on-screen objects. The Border is drawn immediately inside an object's rectangle. The border of an object is separate from the three-dimensional look of the object – which is governed by the 3D special effect properties.

The Border properties of an object are normally specified through the Display dialog box, but you may use a script to change them in order to visually highlight a special condition.

The Border Class has three properties:

Color (Color is itself a Class Property - see **Color**

Style

WidthX

A fourth property **WidthY** - may appear in the Property Pick List, but this property is not implemented and should be ignored.

Style

Style determines how the border is drawn, but it will only take effect if **WidthX** (the line thicknesses of the horizontal and vertical drawing strokes) is greater than 0.

The effect of changing a Border property of an object will not appear until the object is redrawn, either as a result of a user action or through a script that redraws the object.

Type

Number

Details

The number represents a style for the border, depending on the type of object and the width of the border. As long as the border is set to either 0 or 1, the styles for **Line** objects are as follows:

0 = Solid

1 = Long Dashed

2 = Short Dashed

3 = Alternating dash and dot

4 = Alternating dash and double dot

5 = None

6 = InsideFrame (same as Solid)

Any other positive number gives a line style of ‘none’; all negative numbers give a solid line.

For **all other** objects which use Borders, the styles are:

0 = Solid

1 = Solid

2 = Short Dashed (same as for a Line object)

All other numbers – positive or negative – give a solid line.

WidthX

Usually initialized by highlighting the Border option in the in the Display Properties Dialog and then selecting an option from the **Line Type** picklist, this read/write property indicates the thickness of an object's border.

The effect of changing a Border property of an object will not appear until the object is redrawn, either as a result of a user action or through a script that redraws the object.

Type

Number.

Details

WidthX is the thickness of the border (on all sides). The thickness of a border increases inward, remaining inside the defined rectangle.

The WidthX property of an object's border is designed using the Display dialog box, and fixed after that, but it can be modified through a script to visually highlight a special condition. As with all OML Scripted visual changes to an object, these are run-time changes, and the display properties will return to their original values when the form is closed.

Note that a WidthX value of anything other than 0 or 1 makes the border style solid, no matter what the value for its Style property is.

Example

```
MyBoxObject.border.Widthx := 40
```

...which would set an extremely wide border inside the object.

3D Class Properties

When you define a 3D object you are given the opportunity to define a number of special visual effects. These are:

DipShade	**LowShine**
DipShine	**RimShade**
HighShade	**RimShine**
Shadow	**HighShine**
TextShade	**LowShade**
TextShine	

These eleven Class Properties all provide special colour effects on 3D objects, and all are called with exactly the same syntax.

Each of these eleven 3D properties calls the Red, Green and Blue colour properties. Examples are shown below.

```
My3Dobject.HighShade.red := 255 .
My3Dobject.HighShade.green := 255 .
My3Dobject.HighShade.blue:= 255 .
My3Dobject.HighShine.blue := 0 .
My3Dobject.HighShine.red := 255 .
My3Dobject.HighShine.green := 0 .
```

..and so on.

The three color class properties are used to change the colour of an object, such as a box, a line, or a piece of text. You change the colour by setting values for Blue, Red, and Green - the three RBG values from which all colours are derived in computer graphics. Values are in the range 0-255.

For example, setting Green=255, Red=0 and Blue=0 would produce a green colour. Setting Green=255, Red=255 and Blue=255 would produce white, while Green=0, Red=0 and Blue=0 would produce black.

The **Red**, **Green** and **Blue** properties are described below.

Blue

This property specifies the saturation of blue coloring.

Type

Number.

Details:

Set this property to a value between 0 and 255. 0 specifies minimum saturation, 255 specifies maximum.

Example

```
My3DEditBox.lowshine.Blue := 0 .
```

Green

This property specifies the saturation of green coloring.

Type

Number.

Details:

Set this property to a value between 0 and 255. 0 specifies minimum saturation, 255 specifies maximum.

Example

```
My3DEditBox.highshine.Green := 10 .
```

Red

This property specifies the saturation of red coloring.

Type

Number.

Details:

Set this property to a value between 0 and 255. 0 specifies minimum saturation, 255 specifies maximum.

Example

```
My3DEditBox.dipshine.Red := 128 .
```

Shine (Class Property)

The Button Shine effect is not currently implemented.

The Shine Class Property is used to set the colour of the 'shine effect' in Button objects. You change the colour by setting values for Blue, Red, and Green - the three RBG values from which all colours are derived in computer graphics. Values are in the range 0-255.

For example, setting Green=255, Red=0 and Blue=0 would produce a green colour. Setting Green=255, Red=255 and Blue=255 would produce white, while Green=0, Red=0 and Blue=0 would produce black.

Shine sets three Simple Properties, namely:

Green

Blue

Red

Red

This property specifies the saturation of red coloring.

Type

Number.

Details:

Set this property to a value between 0 and 255. 0 specifies minimum saturation, 255 specifies maximum.

Example

```
CreditBox.shine.Red := 255 .
```

Blue

This property specifies the saturation of blue coloring.

Type

Number.

Details:

Set this property to a value between 0 and 255. 0 specifies minimum saturation, 255 specifies maximum.

Example

```
CreditBox.shine.Blue := 128 .
```

Green

This property specifies the saturation of green coloring.

Type

Number.

Details:

Set this property to a value between 0 and 255. 0 specifies minimum saturation, 255 specifies maximum.

Example

```
CreditBox.shine.Green := 10 .
```

Shade (Class Property)

The Button Shade effect is not currently implemented.

The Shade Class Property is used to set the colour of the 'shade effect' in Button objects. You change the colour by setting values for Blue, Red, and Green - the three RBG values from which all colours are derived in computer graphics. Values are in the range 0-255.

For example, setting Green=255, Red=0 and Blue=0 would produce a green colour. Setting Green=255, Red=255 and Blue=255 would produce white, while Green=0, Red=0 and Blue=0 would produce black.

Shade sets three Simple Properties, namely:

Blue

Red

Green

Red

This property specifies the saturation of red coloring.

Type

Number.

Details:

Set this property to a value between 0 and 255. 0 specifies minimum saturation, 255 specifies maximum.

Example

```
CreditBox.shade.Red := 128 .
```

Blue

This property specifies the saturation of blue coloring.

Type

Number.

Details:

Set this property to a value between 0 and 255. 0 specifies minimum saturation, 255 specifies maximum.

Example

```
CreditBox.shade.Blue := 0 .
```

Green

This property specifies the saturation of green coloring.

Type

Number.

Details:

Set this property to a value between 0 and 255. 0 specifies minimum saturation, 255 specifies maximum.

Example

```
CreditBox.shade.Green := 10 .
```

Rect (Class Property)

The **Rect** Class Property describes the on-screen rectangular area occupied by an object. (Don't confuse it with the Rectangle, which is just a graphic object). Objects which use Rect include Edit Boxes, CheckBoxes, Rectangles, and Text Labels. The four properties of a rectangle object define the X and Y co-ordinates that fully describe the object's display area.

The four properties are:

Top

Bottom

Left

Right

You can determine the object's current location by reading its four Rect properties.

Example

Type the following script on a **Button>>Clicked** event.

```
Rect.Top := Rect.Top  10 .
Rect.Bottom := Rect.Bottom  10 .
Rect.Left := Rect.Left  10 .
Rect.Right := Rect.Right  10 .
Hide() .
Show() .
Draw() .
```

..which will shift the button to the right and down every time you click on it. If you assign different values to these four properties - say, 10, 30, 10, 30 - then both the position and **size** of the object will be changed.

Limitations

If the Top coordinate is greater than or equal to the Bottom, or the Left coordinate is greater than or equal to the Right, no rectangle is defined, and the object will disappear from view.

Top

Position of rect top. Increasing the value moves the top border edge down. Decreasing it moves the edge up.

Bottom

Position of rect bottom. Increasing the value moves the bottom border edge down. Decreasing it moves the edge up.

Left

Position of rect left. Increasing the value moves the left border edge right. Decreasing it moves the edge left.

Right

Position of rect right. Increasing the value moves the right border edge right. Decreasing it moves the edge left.

RectBorder (Class Property)

Most objects use the Border Class Property to select the style, color and width of the border which surrounds the object itself.

The SpinBox and 3DspinBox (also called SpinButtons) have two separate 'border' areas - the data-area, and the 'spin arrows' themselves. So to define these areas they need two Class Properties instead of one. These Class Properties are RectBorder and ScrollBorder.

The RectBorder Class Property is used to define borders around the perimeters of the spin-box's **data-area**. The Border is drawn immediately inside the data-area's rectangle. (Note that this border of is separate from the three-dimensional look of a 3D SpinBox – which is governed by the 3D special effect properties).

The RectBorder properties of are normally specified through the Display dialog box, but you may use a script to change them in order to visually highlight a special condition.

The RectBorder Class has three properties:

Color (Color is itself a Class Property - see **Color**

WidthX

A fourth property **WidthY** - may appear in the Property Pick List, but this property is not implemented and should be ignored.

Style

Style determines how the border is drawn, but it will only take effect if **WidthX** (the line thicknesses of the horizontal and vertical drawing strokes) is greater than 0.

The effect of changing a RectBorder property of an object will not appear until the object is redrawn, either as a result of a user action or through a script that redraws the object.

Type

Number

Details

The number represents a style for the border. The styles are:

0 = Solid

1 = Solid

2 = Short Dashed

All other numbers – positive or negative – give a solid line.

WidthX

Usually initialized by highlighting the Border option in the in the Display Properties Dialog and then selecting an option from the **Line Type** picklist, this read/write property indicates the thickness of an object's border.

The effect of changing a RectBorder property of an object will not appear until the object is redrawn, either as a result of a user action or through a script that redraws the object.

Type

Number.

Details

WidthX is the thickness of the border (on all sides). The thickness of a border increases inward, remaining inside the defined rectangle.

The WidthX property of an object's border is designed using the Display dialog box, and fixed after that, but it can be modified through a script to visually highlight a special condition. As with all OML Scripted visual changes to an object, these are run-time changes, and the display properties will return to their original values when the form is closed.

Note that a WidthX value of anything other than 0 or 1 makes the border style solid, no matter what the value for its Style property is.

Example

```
My3DSpinBox.rectborder.Widthx := 40
```

...which would set an extremely wide border inside the object.

RectFill (Class Property)

This Class property applies to the SpinBox and 3DspinBox objects only.

Most objects use the Fill Class Property to set the following three properties of an object.

Color (Color is itself a Class Property - see **Color**

Style

Hatch

...but the Spin Box and 3DspinBox (spinbuttons) have two separate components - the 'data-area' and the 'spin-button' area - so they require two sets of commands to distinguish between these areas. These two Class Properties are **RectFill** and **BorderFill**.

RectFill sets the Color, Style and Hatch of the **data-area**, while BorderFill sets the color, style and hatch of the **spin-button** area.

Using Rectfill, an object's data-area may be made transparent by setting its Style property to 1. An object can have a solid fill of a particular color, a fill of a specified pattern, or no fill.

The effect of changing an object's RectFill property is not displayed until the object is redrawn, either as the result of a user action or through a script that redraws the object.

Style

Style holds a number indicating whether the object is transparent, opaque, or hatched. If Style is hatched, then Hatch is a number indicating how the control face is painted (such as solid, diagonal lines, or vertical lines).

Type

Number:

Details

0 = Solid

1 = No Fill (transparent)

2 = Hatch (patterned)

Example

```
My3DSpinBox.Rectfill.style := 2 .
```

Hatch

This read/write property controls the type of pattern to use for a fill.

Type

Number:

Details

0 = Horizontal Lines1 = Vertical Lines2 = Diagonal from top left to bottom right3 = Diagonal from top right to bottom left4 = Grid Hatch5 = Cross Hatch

Example

```
MySpinButton.Rectfill.hatch := 3 .
```

Remember that you can only set the hatch property if the Style is set to Hatch (2), so the full example would be:

```
MySpinButton.Rectfill.style := 2 .
MySpinButton.Rectfill.hatch := 3 .
MySpinButton.redraw := 1 .
```

Color

Color is itself a Class Property. See **Color**

ScrollBorder(Class Property)

Most objects use the Border Class Property to select the style, color and width of the border which surrounds the object itself.

The SpinBox and 3DspinBox (also called SpinButtons) have two separate 'border' areas - the data-area, and the 'spin arrows' themselves. So to define these areas they need two Class Properties instead of one. These Class Properties are RectBorder and ScrollBorder.

The ScrollBorder Class Property is used to define borders around the perimeters of the spin-box's **spin-buttons**. The Border is drawn immediately inside the spin buttons' rectangle. (Note that this border of is separate from the three-dimensional look of a 3D SpinBox – which is governed by the 3D special effect properties).

The ScrollBorder properties of are normally specified through the Display dialog box, but you may use a script to change them in order to visually highlight a special condition.

The ScrollBorder Class has three properties:

Color (Color is itself a Class Property - see **Color**

Style

WidthX

A fourth property **WidthY** - may appear in the Property Pick List, but this property is not implemented and should be ignored.

Style

Style determines how the border is drawn, but it will only take effect if **WidthX** (the line thickness of the horizontal and vertical drawing strokes) is greater than 0.

The effect of changing a ScrollBorder property of an object will not appear until the object is redrawn, either as a result of a user action or through a script that redraws the object.

Type

Number

Details

The number represents a style for the border. The styles are:

0 = Solid

1 = Solid

2 = Short Dashed

All other numbers – positive or negative – give a solid line.

WidthX

Usually initialized by highlighting the Border option in the in the Display Properties Dialog and then selecting an option from the **Line Type** picklist, this read/write property indicates the thickness of an object's border.

The effect of changing a ScrollBorder property of an object will not appear until the object is redrawn, either as a result of a user action or through a script that redraws the object.

Type

Number.

Details

WidthX is the thickness of the border (on all sides). The thickness of a border increases inward, remaining inside the defined rectangle.

The WidthX property of an object's border is designed using the Display dialog box, and fixed after that, but it can be modified through a script to visually highlight a special condition. As with all OML Scripted visual changes to an object, these are run-time changes, and the display properties will return to their original values when the form is closed.

Note that a WidthX value of anything other than 0 or 1 makes the border style solid, no matter what the value for its Style property is.

Example

```
My3DSpinBox.Scrollborder.Widthx := 40
```

...which would set an extremely wide border inside the object.

ScrollFill(Class Property)

This Class property applies to the SpinBox and 3DspinBox objects only.

Most objects use the Fill Class Property to set the following three properties of an object.

Color (Color is itself a Class Property - see **Color**

Style

Hatch

...but the Spin Box and 3DspinBox (spinbuttons) have two separate components - the 'data-area' and the 'spin-button' area - so they require two sets of commands to distinguish between these areas. These two Class Properties are **RectFill** and **BorderFill**.

RectFill sets the Color, Style and Hatch of the **data-area**, while BorderFill sets the color, style and hatch of the **spin-button** area.

Using Scrollfill, an object's spin-button may be made transparent by setting its Style property to 1. An object can have a solid fill of a particular color, a fill of a specified pattern, or no fill.

The effect of changing an object's ScrollFill property is not displayed until the object is redrawn, either as the result of a user action or through a script that redraws the object.

Style

Style holds a number indicating whether the object is transparent, opaque, or hatched. If Style is hatched, then Hatch is a number indicating how the control face is painted (such as solid, diagonal lines, or vertical lines).

Type

Number:

Details

0 = Solid

1 = No Fill (transparent)

2 = Hatch (patterned)

Example

```
My3DSpinBox.Scrollfill.style := 2 .
```

Hatch

This read/write property controls the type of pattern to use for a fill.

Type

Number:

Details

0 = Horizontal Lines1 = Vertical Lines2 = Diagonal from top left to bottom right3 = Diagonal from top right to bottom left4 = Grid Hatch5 = Cross Hatch

Example

```
MySpinButton.Scrollfill.hatch := 3 .
```

```
Parent(Class Property)
```

The Parent Class Property is used to query the current object's Parent, just as the Child Class Property is used to query a Form's children.

Currently Parent and Child apply only to Forms and Subforms. A SubForm has a Parent, and a form with a subform has a Child.

The Parent Class Property has two simple properties:

Name : returns the name (the form name) of the Parent Form.

ClassName : returns the class of the Parent - always a Form.

Example:

```
Myvariable = MySubform.parent.name .
```

...which would place the name of the subform's Parent Form in the variable Myvariable.

Child(Class Property)

The Child Class Property is used to query the current object's Children, just as the Parent Class Property is used to query a Form's parent.

Currently Child and Parent apply only to Forms and Subforms. A SubForm has a Parent, and a form with a subform has a Child.

The Child Class Property has two simple properties:

Name : returns the name (the form name) of the Child Form.

ClassName : returns the class of the Child - always a Subform.

Example:

```
Myvariable = MySubform.child.name .
```

...which would place the name of the form's first Child Form in the variable Myvariable.

Next(Class Property)

The Next Class Property is used to identify the next object in a list of form objects.

There are two simple properties in Next, namely:

Name : Returns the Name of the object.

ClassName : Returns the ClassName of the object - Box, Ellipse, etc.

..both of which are read only properties.

Example:

You have a form with two objects - a field called "Customer" and a field called "Phone". Place the following script in the Customer object.

```
MyVariable = myformname.next.name
```

…which would place the name of the next object (Phone) in the form in the variable called MyVariable.

Note that objects in a Form are listed from top left to bottom right of the form.

To reverse the search, use the **Prev** Class Property. This works in exactly the same way as Next, but searches upwards through the list of objects, rather than downwards.

Prev(Class Property)

The Prev Class Property is used to identify the previous object in a list of form objects.

There are two simple properties in Prev, namely:

Name : Returns the Name of the object.

ClassName : Returns the ClassName of the object - Box, Ellipse, etc.

..both of which are read only properties.

Example:

You have a form with two objects - a field called "Customer" and a field called "Phone". Place the following script in the Phone object.

```
MyVariable = myformname.prev.name
```

…which would place the name of the previous object (Customer) in the form in the variable called MyVariable.

Note that objects in a Form are listed from top left to bottom right of the form.

To reverse the search, use the **Next** Class Property. This works in exactly the same way as Next, but searches downwards through the list of objects, rather than upwards.

Example Scripts: Input Validation with PostEdit

This script demonstrates how the PostEdit Event can be used to perform dynamic validation, by automatically testing values as they are typed in by the user. This example script rejects any input which contains spaces in the value.

Select the event to script as: **MyEditBox>> PostEdit**.

…where **MyEditBox** is the name of the EditBox you wish to validate.

Note that the PostEdit **String** parameter contains the user's input data, while the original (retrieved from the Table) value is held in the MyEditBox **Value** property.

Now write the following script:

```
if textpos( String, " " ) > 0 then
  message "Please re-input without spaces!"
   window .
else
  return(1) .
end
```

Result: If the user types in a space character, and then leaves the field, a warning message is displayed on screen. Focus remains in the field.

Note: DataEase is unusual in that the event parameter **return()** does not hold a Boolean value, and moreover treats the values "**0**" and "**1**" as both meaning **True.** Any result other than **0** or **1** means **False**.

So if you are testing for a **False** condition, write **return(2)**. DataEase will recognise the **2** as being a False value.

Example Scripts: ValueLoaded Event

The following two examples show how the ValueLoaded Event can be used to carry out a record-by-record check. When the check-condition is triggered, the field being checked changes colour.

The first example shows the script working in a form document, and the second example shows the script working in a report.

Example 1: On the Form Document

Select the event to script as: **MyEditBox>> ValueLoaded**, then type in the following script.

```
if salary.value not= blank then
  if salary.visible = 1 then
    if abs(salary.value) > 30000 then
      fill.color.green := 0 .
      fill.color.blue := 0 .
    else
      fill.color.green := 255 .
      fill.color.blue := 255 .
    end
    salary.visible := 0 .
  else
    salary.visible := 1 .
  end
  redraw := 1 .
  return(1) .
end
```

Result: Whenever a record is opened, the script will change the Salary Field to Red, if the salary is greater than 30,000.

Note: Because the ValueLoaded Event fires twice for each record, the script uses a 'dummy' outer loop - `if salary.visible = 1 then` - to ignore the first Event.

Example 2: On a Report Document

Note: To work as shown, the report **mus**t be WYSIWYG and the field **must** have a solid fill).

1. Select the event to script as: **MyEditBox>> ValueLoaded**.
2. Type in the following script.

```
if abs(salary.value) > 40000 then
     salary.fill.color.green := 0 .
     salary.fill.color.blue := 0 .
     salary.fill.color.red := 255 .
  redraw := 1 .
end
```

Result: If the record being reported on has a salary greater than 40,000, then the salary field will be displayed in red.

Note: Since the report only triggers the ValueLoaded event once, there is no need for a dummy outer loop, as was shown in the form document example.

Example Scripts: Report Totals

This example shows how a report total can be used to calculate a further total - say, a fixed postage and packing charge added to an invoice total. The example below assumed a fixed postage of £2.40.

Place your script in the **SummaryField>>ValueLoaded** Event. You will also need to create a text label named LblFinalResult, and place this beneath the SummaryField.

```
number pos .
text tAnswer .

tAnswer := concat(abs( Value ) 2.40, blank) .
pos := textpos( tAnswer ,"." ) .
if pos = 0 then
  tAnswer := concat( tAnswer, ".00" ) .
else
  if pos= length( tAnswer ) - 1 then
    tAnswer := concat( tAnswer, "0" ) .
  else
    tAnswer := firstc( tAnswer, pos  2 ) .
  end
end

LblFinalResult.TextString := tAnswer .
```

Result: The LbLFinalResult label displays the invoice value plus £2.40. Notice that the DQL TextPos command is used to correctly format the final answer, adding trailing "0"s as required so that "15" becomes "15.00" and "26.3" becomes "26.30".

Example Scripts: Realtime Data Processing

Discussion

The following example assumes the existence of a table called STOCK_MASTER, to which there exists a related table named DAILY_SALES.

Entries in DAILY_SALES must instantly reduce stock levels in STOCK_MASTER for that Stock_Code, and simultaneously archive the sales data into a table called HISTORY.

The relationship from either HISTORY or DAILY_SALES to STOCK_MASTER has been created and is called rMASTER.

Historically, DataEase developers would create the DAILY_SALES screen either as a DQL procedure's data-entry form, or use the 'input using' DQL statement.

Here is it done by putting a real-time processing button on the DAILY_SALES screen and suppressing via the menu and toolbar any ability for the user to create records in DAILY_SALES.

The script for the button clicked event is....

```
-- Process realtime

define "tCODE" Numeric String 3 .
define "tQTY" Number .
define "dummy" Number .
define "tPassedInfo" text 255 .

assign tCode := Stock_Code.Value .
assign tQTY := Qty_Sold.Value .

tPassedInfo := concat( tCODE, "^", tQTY ) .
dummy := SetArray( 1, tPassedInfo ) .

run procedure "MakeHistory" .

message "Input has been processed..!" window .

tQTY := RecordRestore() .

-- end of script
```

This script passes the data entered onto the screen to a DQL procedure in a string using the **SetArray** CDF (supplied with DataEase in the CDFS2.DLL library). This CDF has to be registered in the Custom Functions form, as does another CDF - **RecordRestore** - which is supplied in the DFWACTS.DLL CDF library.

RecordRestore is used to prevent DataEase from asking "save the changes?" when the screen is cleared in DAILY_SALES . It then calls a separately written DQL process which reads the values from the passed global string using **GetArray** (the matching part of SetArray), then carries out the modification and creation routines.

the MakeHistory DQL reads....

```
define "tCODE" Numeric String 3 . define "tQTY"
  Number .
define "dummy" Number .  define "tPassedInfo" text
  255 .

tPassedInfo := GetArray(1) .
if textpos( tPassedInfo , "^" ) not = 4 then
  exit .
end

tCode := firstc( tPassedInfo, 3 ) .
tQTY := midc ( tPassedInfo, 5, 255 ) .

modify records in STOCK_MASTER named "match"
                    with ( Stock_Code = tCode )
  Qty_Held := Qty_Held - abs(tQTY) .

enter a record in HISTORY
  Stock_Code := tCode ;
  Qty_Sold :=  tQTY ;
  Date := current date .

dummy := SetArray ( 1, "" ) .
```

Example Scripts: Conditional Subform Display

Subform Display is User Choice

Here is an example of how you can create conditionally displaying subforms.

The code is based on Club ParaDease, and will make the subform CLUB ACTIVITIES hidden when the form CLUBS opens. CLUB ACTIVITIES can then be displayed by clicking on a button, and hidden again by clicking another button.

1. Open CLUBS and Create a Text Label called **HideFlag**.
2. Type the value **ON** into HideFlag.
3. Put the following code in the ValueLoaded event on CLUB NAME.

```
If HideFlag.TextString = "ON" then
HideFlag.TextString := "OFF" .
HideFlag.Hide() .
CLUB ACTIVITIES.Hide() .
else
end
```

4. Create a button called ActivitiesOn.
5. In the clicked event on this button put the following code.

```
CLUB ACTIVITIES.Show() .
```

6. Create a button called ActivitiesOff.
7. In the clicked event on this button put the following code.

```
CLUB ACTIVITIES.Hide() .
```

Subform display is automatic

While the above example allows the user to hide or display the Subform at will, the example below requires just one OML Script to automatically hide the subform is there is no matching data, and display if it there IS matching data.

This example uses the RESERVATIONS form of Club ParaDease.

1. Open RESERVATIONS and put the following code in the **ValueLoaded** event on the **Last Name** field.

```
if count of RESERVATION DETAIL = 0 then
RESERVATION DETAIL.Hide() .
else
RESERVATION DETAIL.Show() .
end
```

This will hide the subform if there are no matching records, and display it if there are 1 or more matching record. When the form is initially displayed, there are no matching records, so the subform will be hidden.

Additions in 6.5 OML

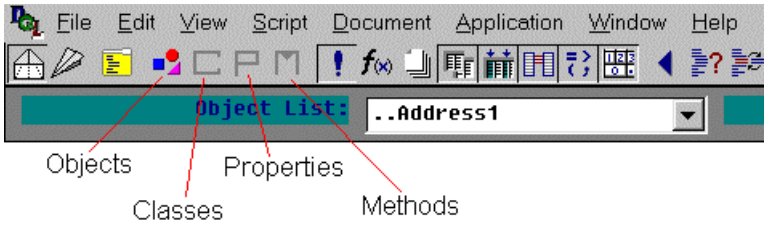
DataEase 6.5 introduced a number of improvements to OML Scripting. The major differences are listed below.

New Toolbar Icons

Previous versions of DataEase 6 did not have facilities to switch the OML Scripting Pick-Lists on or off, which made the Edit Screen a little crowded for people who were writing DQL.

This oversight is corrected in 6.5, which contains four Toolbar Icons which control the OML-specific Pick-Lists, namely Objects, Classes, Properties and Methods.

Clicking one of the icons shown below will toggle the appropriate Pick-List on or off.



Note that when you close and re-open the form, the DQL/OML Script Editor will default to displaying all the Pick-Lists.

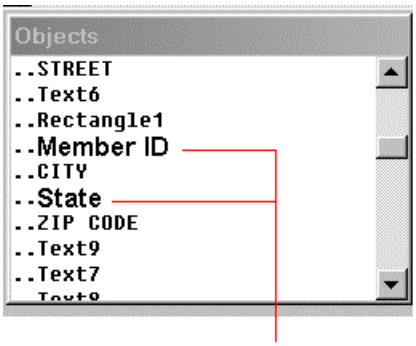
Enter Key and Button Focus

In previous versions of DataEase a Button Object's clicked event OML script was fired when the button was clicked – but not when the Enter key was pressed. This was incorrect - since in Windows the action of 'clicking' a button and hitting the enter key while the focus is on a button – should be the same.

This inconsistent behavior has now been corrected. In 6.5, a Button Object's clicked event OML script will be fired if the button is clicked, or if the Enter key is pressed while the button has focus.

Bold Pick Lists

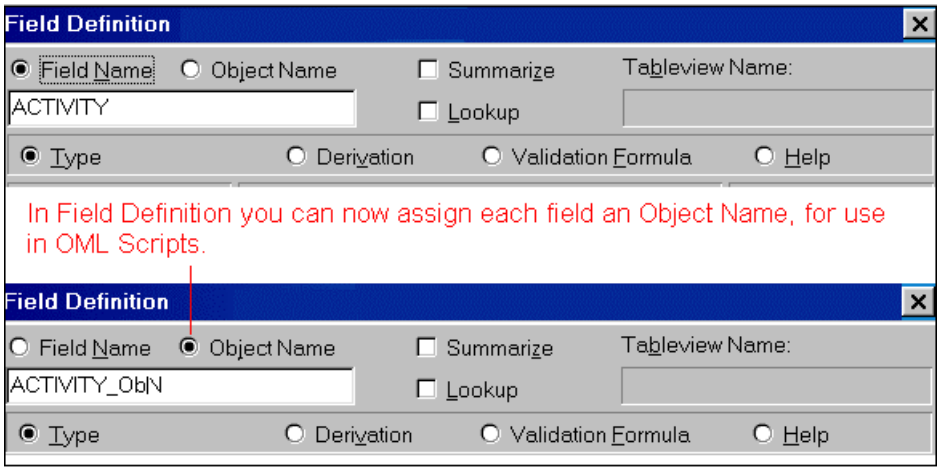
Objects and Methods that have OML script attached are now shown in **bold** typeface in the Script Editor, so you can tell at a glance where your OML scripts are in place.



Notice that two of the Objects shown in the Object List Box are in bold print, indicating that they have OML scripts attached to them.

Optional Object Names

DataEase 6.5 allows you to assign Object Names to Field objects and Summary Fields, so that they can be manipulated by OML scripts.



In previous versions you were not able to make DataEase differentiate in scripts between a table field and a field object (edit box, or list box, etc.), which is used to display data in a table, because they had the same names. Similarly, all summary fields were named 'Summary', which posed particular limitations: for example, it was impossible to change the background colour of summary fields because fields were not objects.

The object-oriented approach implemented in DataEase 6.5 removes these constraints: now you can optionally give Field objects and Summary fields Object Names that will be unique, the default Object names coinciding with the Field Names.

So a data-related control (or field object) can be processed in scripts (OML) as soon as it has got its own unique Object name.

New Event Parameters

The following events' parameters now return data: See **Keyboard Events**.

- KeyInput (number KeyValue),
- KeyDown (number KeyValue),
- KeyUp (number KeyValue),

Alphabetical Command List

Events

Clicked	**DblClicked**	**DownArrowClicked**
GotFocus	**LostFocus**	**MouseDown**
MouseUp	**MouseMove**	**MouseOver**
MouseEnter	**MouseExit**	**PostEdit**
PredEdit	**KeyDown**	**KeyInput**
KeyUp	**UpArrowClicked**	**ValueLoaded**
ValueChange	**ValueRequired**	

Methods

Delete	**DoAction**	**Draw**
Erase	**EditProperty**	**GetObject**
Hide	**Highlight**	**Insert**
Invert	**MouseCapture**	**Move**
MoveTo	**MouseRelease**	**Replace**
Select	**Show**	

Properties

Simple Properties are listed by their Name only, while Class properties are proceeded by the letter **(C)**.

Action	**(C)Border**	**Backpoint**	**CanTab**
Child	**ClassName**	**DataConnected**	**Delay**
(C)DipShade	**(C)DipShine**	**Enable**	**(C)Fill**
(C)Font	**FrontPoint**	**HasValue**	**(C)HighShade**
(C)HighShine	**IsCompound**	**IsForm**	**IsLabel**
IsLiveReport	**IsPrinting**	**IsRecord**	**IsReport**
IsSelected	**IsTableView**	**Justify**	**(C)LowShade**
(C)LowShine	**Name**	**(C)Next**	**(C)Parent**
(C)Prev	**PropertyList**	**(C)Rect**	**(C)RectBorder**
(C)RectFill	**ReDraw**	**(C)RimShade**	**(C)RimShine**
RoundedCorners	**(C)ScrollBorder**	**(C)ScrollFill**	**(C)Shade**
(C)Shine	**(C)Shadow**	**Taborder**	**TextString**
TextMargin	**TextSlantinDegrees**	**(C)TextShade**	**(C)TextShine**
Thickness	**Type**	**Value**	**View**
Visible	**Wrap**		

Index

3D Class Properties, 61
3DcheckBox (3DRadioButton), 28
3DeditBox, 29
3Dlabel, 30
3Dline, 31
3DlistBox, 32
3DradioBox, 33
3Dspin, 34
Action, 24
Application Variable, 35
Backpoint, 24
Blue, 56
Bold, 51
Border, 24
Border (Class Property), 59
Bottom, 67
Button, 36
CanTab, 24
CharSet, 53
Checkbox, 37
Child, 24
Child(Class Property), 76
Class, 3
Class Properties, 4, 50
ClassName, 24
Clicked, 17
Color, 58
Color (Class Property), 55
comparison operators, 11
Conditional Logic, 12
Conditional Subform Display, 85
Control, 3
Control Commands, 10
Data Model, 4, 8
DataConnected, 24
DblClicked, 17
Defining an Event Script, 5
Delay, 24
Delete, 21
DipShade, 24
DipShine, 25
Displaying Changes, 13
DoAction, 21
DownArrowClicked, 17
DQL Functions, 11
DQL in Scripts, 8
DQL symbols, 11
Draw, 21
Draw() method, 13
EditBox, 38
EditProperty, 21
Ellipse, 39
Enable, 25
Erase, 21
Error Messages, 14
Event, 3
Event Strings, 16
Event/Methods, 4
Events, 15
ExecuteVerbList, 21
Fill, 25
Fill (Class Property), 57
Font, 25
Font (Class Property), 51
FrontPoint, 25
GetObject, 21
GetVerbList, 21
Global Variables, 7
GotFocus, 17
Green, 56
HasValue, 25
Hatch, 57
Hide, 21
Highlight, 21
HighShade, 25
HighShine, 25

Image (Picture), 40
ImageField, 41
Input Validation, 79
Instantiation, 3
Invert, 21
IsCompound, 25
IsForm, 25
IsLabel, 25
IsLiveReport, 25
IsPrinting, 25
IsRecord, 25
IsReport, 25
IsSelected, 25
IsTableView, 25
Italic, 52
Justify, 25
KeyDown, 19
KeyInput, 19
KeyUp, 19
Label, 42
Left, 67
Line, 43
List of Events, 17
List of Methods, 21
List of Objects, 27
ListBox, 44
Local Variables, 7
Lost Scripts, 14
LostFocus, 18
LowShade, 25
LowShine, 25
Method, 3
Methods, 21
MouseCapture, 21
MouseDown, 18
MouseEnter, 19
MouseExit, 19
MouseMove, 19
MouseOver, 19
MouseRelease, 21
MouseUp, 19
Move, 21
MoveTo, 21
Multiview, 4
MultiView, 8
Name, 25, 52
Next, 25
Next(Class Property), 77
nsert, 21
Object, 3
Objects, 27
OLE, 45
Parent, 26
PostEdit, 18
PreEdit, 17
Prev, 26
Prev(Class Property), 78
Procedural Commands, 10
Processing Commands, 9
Properties, 23
Property, 4
Property List, 24
PropertyList, 26
RadioBox (RadioButton), 46
Realtime Data Processing, 83
Rect, 26
Rect (Class Property), 67
Rectangle (Box Class Object), 47
RectBorder, 26
RectBorder (Class Property), 68
RectFill, 26
RectFill (Class Property), 70
Red, 56
ReDraw, 26
ReDraw property, 13
Relational Statistical Operators, 12
Replace, 22
Report Totals, 82
Right, 67
RimShade, 26
RimShine, 26
RoundedCorners, 26
Script Syntax, 6
ScrollBorder, 26

ScrollBorder(Class Property), 72
ScrollFill, 26
ScrollFill(Class Property), 74
Select, 22
Shade, 26
Shade (Class Property), 65
Shadow, 26
Shine, 26
Shine (Class Property), 63
Show, 22
Show() method, 13
Simple Properties, 4
Size, 52
Sorting and Grouping Operators, 9
Spin (Spin Button), 48
Strike, 53
Style, 57
Style (Border), 59
Summary Variable, 49
Taborder, 26
TextMargin, 26
TextShade, 26
TextShine, 26
TextSlantinDegrees, 26
TextString, 26
The Event Return, 15
Thickness, 26
Top, 67
Type, 26
Under, 53
UpArrowClicked, 19
Using the Pick Lists, 6
Value, 26
ValueChange, 18
ValueLoaded, 18
ValueLoaded Event, 80
ValueRequired, 18
Variables in Scripts, 7
View, 26
Visible, 26
WidthX (Border), 60
Wrap, 26