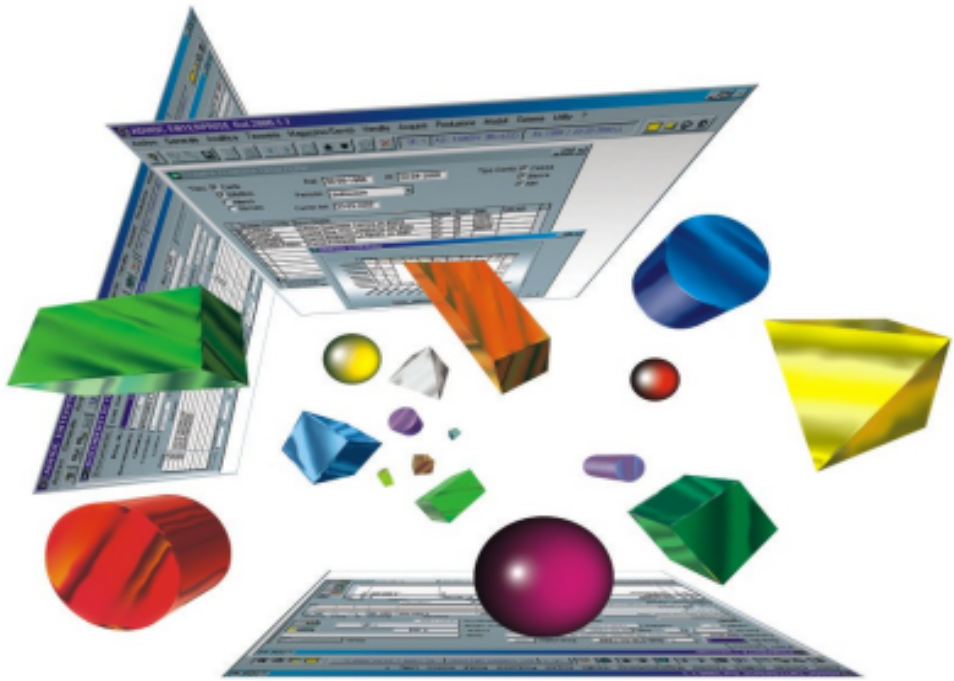


CODEPAINTER REVOLUTION

Component Guide

ZUCCHETTI TOOLS S.r.l. SOFTWARE TECHNOLOGY
PADOVA - BELLARIA - RIMINI
E-mail: clabrn@codelab.it
Indirizzi Sito Web:
http:\\www.zucchettitools.com
http:\\www.codepainter.com
http:\\www.codelab.it

Sommario

Capitolo 1

Introduction

The 'Component Guide' guides you through the functionalities of CODE**PAINTER REVOLUTION** Visual Objects.

The next chapter describes all available objects, its characteristics and how these objects can be used.

WARNING

Class titles and property strings in CodePainter are currently still in Italian. This manual therefore maintains the Italian names and strings. You will always find the English translation next to the titles and under the strings.

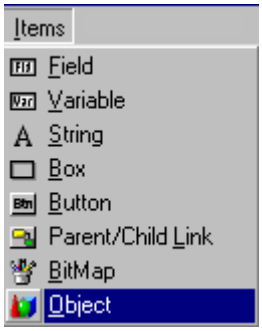
We apologize for any inconvenience this may cause to you and we commit amending the problem as soon as possible.

Capitolo 2

The Component Guide

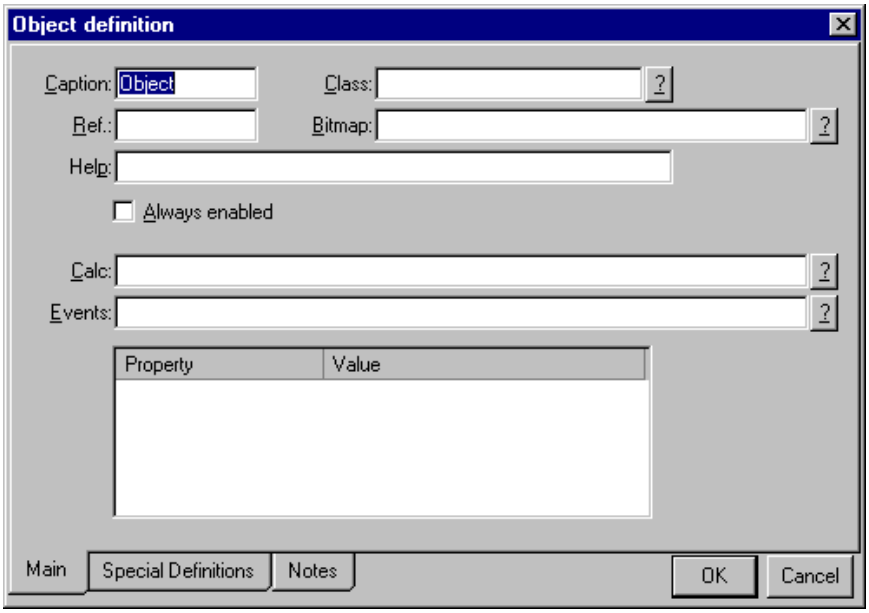
2.1 Objects Options

Visual Objects can be added to dialog windows of your application. Go to CodePainter Front End and open the Painter corresponding to the desired entity (Master File, Detail File, Master/Detail, Dialog Window, Routine Painters). Open the 'Item' menu and select 'Object'.



You can also add a Visual Object clicking the 'Object' button on the 'Painter Tools' toolbar.

Once the object has been added you need to define its parameters in the 'Object Definition' dialog window.



CodePainter has a set of predefined Visual Objects that belong to predefined classes. When you select a Visual Object some simple guidelines help you defining the object. These guidelines are taken from a list of predefined classes.

The class list is stored in the OBJECTS directory of CODE**PAINTER REVOLUTION**. Objects are defined in the **CP_CLASS.PRG** routine, which is stored under the **VFCSIM** directory of CODE**PAINTER REVOLUTION**.

A new class can be defined creating a subdirectory to OBJECTS, which will be named after the class. The new class must be also defined in the routine.

In each directory that identifies a class a file with the extention '.CPL' is required. This file can be created/modified using a Text Editor, such ad NotePad, MS WinWord, etc. Each .CPL file contains a standard command ('GenericObject….') as well as the initialization of class properties (in the "Init()" method).

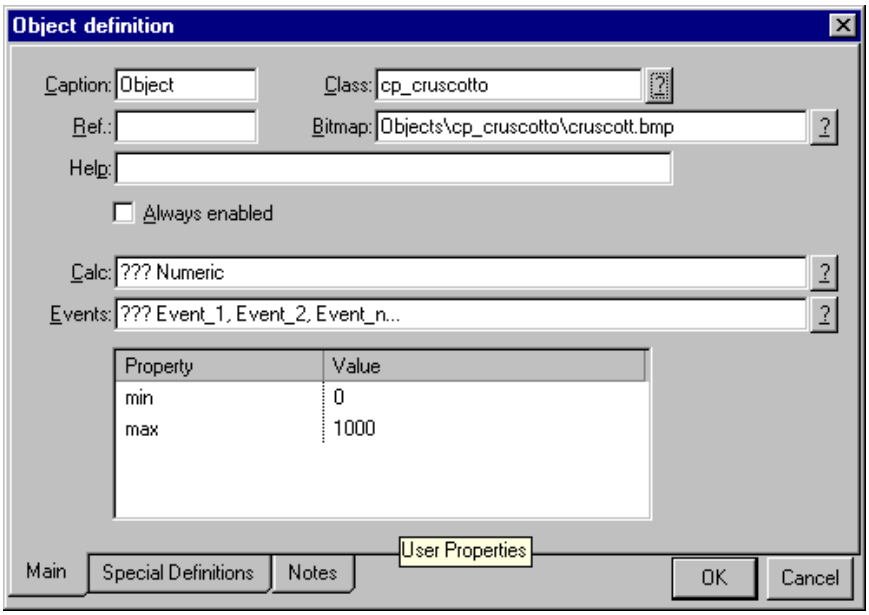
Example of a CPL File for an additional class

```
from GenericObject import Object

class cp_<nome classe>(Object)
  proc Init()
    description:="<Class Description>"
    bitmap:="immagine.bmp"
    w:=<initial width >
    h:=<initial hight >
    <additional property >:=<additional property value>
    ...
    userproperties["<property>"]:='<value>'
  end
end
```

For more information on object class definition please refer to manuals of the used programming language under the heading CP_CLASS.PRG.

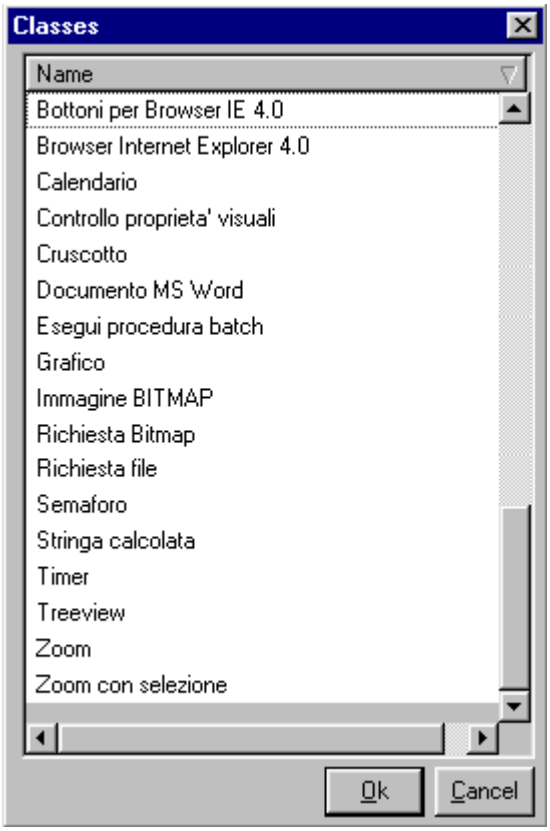
To enter an object in the Class field click the '? Button: a pick-list containing predefined and additional classes is opened. Basing on the class type selected, some fields within the 'Object Definition' window are valorized. Typically, valorized fields are 'Calc' and 'User Def'". For more details on the meaning of fields please refer to the User Reference Guide.



To add Visual Objects select the '?' button next to the **Class** field. A pick list is opened and a set of class types shown. Depending on the class type selected, fields in the dialog window are automatically entered. The class values are entered in the 'Calc.' field, and the field classes characteristics in the 'Property' area . For the meaning and use of other fields in the dialog window please refer to the 'User Reference Guide'.

2.2 Available Objects

Let us now see the list of available objects.



2.2.1 Cruscotto (Dashboard)

The object 'Cruscotto' displays a dashboard that changes depending on the value of a field or of a numeric variable. To add the dashboard you need to define the following parameters:

Calc

The 'Calc' field defines the numeric value ('??? Numeric' defines the parameter type) to which the arrow position changes. The syntax is:*'??? Numerico'*

Translation:'??? Numeric'

Property

The 'Property' area contains the parameters required for the value definition. The syntax is: *Min=0,Max=100*

Parameter Name	Description
Min	Minimum Dashboard Value.
Max	Maximum Dashboard Value.

To display the weighting of the item price defined in the field PRZART you need to define the price working variable in the 'Calc' field, namely:

W_PRZART

If the price value can be minimum 0 and maximum 1000 you need to change the 'Property' in:

Min=0,Max=1000.

2.2.2 Esegui procedura batch (Execute Routine)

The 'Esegui Procedura Batch' object allows executing a routine when one or more defined events are triggered. An event is a specific happening in the program execution.

The *'NotifyEvent'* procedure is defined in each program and allows notifying these happenings so that they can be managed by the '*Event*' procedure defined in Visual Objects.

CodePainter supports predefined events that are notified at fixed points of each procedure and cannot be changed. Some of these events are INIT, DONE, INSERT START, INSERT END, etc.; e.g. the 'Update End' event is added after a record update to start a routine performing specific tasks.

To add the 'Execute Routine' object you need to define the following parameters:

Events

The 'Events' field defines the event or the list of events (divided by commas) that trigger the routine procedure. The field is optional. Clicking the '?' button next to the field all available events are listed. Events are stored in the EVENTS.CPL file under the CLASSES directory.

```
The syntax used is:'??? Evento_1, Evento_2, Evento_n...'


Translation'??? Event_1, Event_2, Event_n...'
```

Property

```
The 'Property' field contains the name of the routine procedure that
must be executed. The syntax used is:Prg='??? nome del programma da
eseguire'


TranslationPrg='??? procedure name that must be executed'
```

To execute the routine procedure EXPL_DIS after the current record is saved, the 'Property' field must be defined as follows:

```
Prg='expl_dis'
```

In the 'Events' field you need to type the following event:

```
Update end
```

2.2.3 Calendario (Calendar)

The 'Calendario' object displays a calendar and gives you the possibility to select a date. To use this object you need to define the following parameters:

Calc

```
In the 'Calc' field define a 'date' type variable. The syntax used
is:'??? Data'


Translation'??? Date'
```

Property

In the 'Property' field define a 'date' type variable in which you want to input the selected date. The date is selected double clicking it on the calendar.

```
The syntax used is: Var='w_???'
```

To display the calendar with the date stored in the field 'DATAMOV' in the 'Calc' field type

```
W_DATAMOV
```

In the 'Property' field type:

```
var='W_DATMOV'
```

2.2.4 Semaforo (Traffic Lights)

The 'Semaforo' object displays green or red traffic lights, depending on the logical expression associated to the object. To use traffic lights you need to define the parameters as follows:

Calc

```
In the 'Calc' field define the logical expression that influences the
traffic lights color. The syntax used is: '??? logico (.t.=verde
.f.=rosso)'
```

```
Translation '??? logic (.t.=green .f.=red)'
```

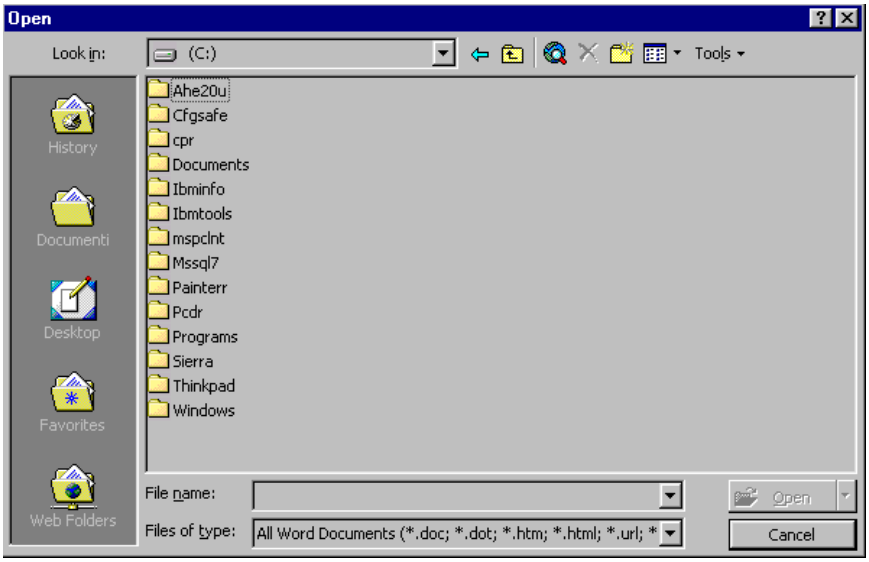
To display green traffic lights when the price in the field PRZART is greater than 100 you need to define the following parameter:

In the 'Calc' field type the expression:

```
W_PRZART>100
```

2.2.5 Richiesta file (Ask File)

The 'Richiesta File' object displays the 'Open' dialog window so that a file can be selected. A file name and its path can be associated to a working variable (of a field or variable). This class can be used for example to associate a MS Word document to be opened with the class 'Word Document'.



To use this object you need to define the following parameters:

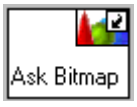
Property

```
In the 'Property' field you need to add the working variable
containing the path of the selected file. The syntax used is:
Var='w_???'
```

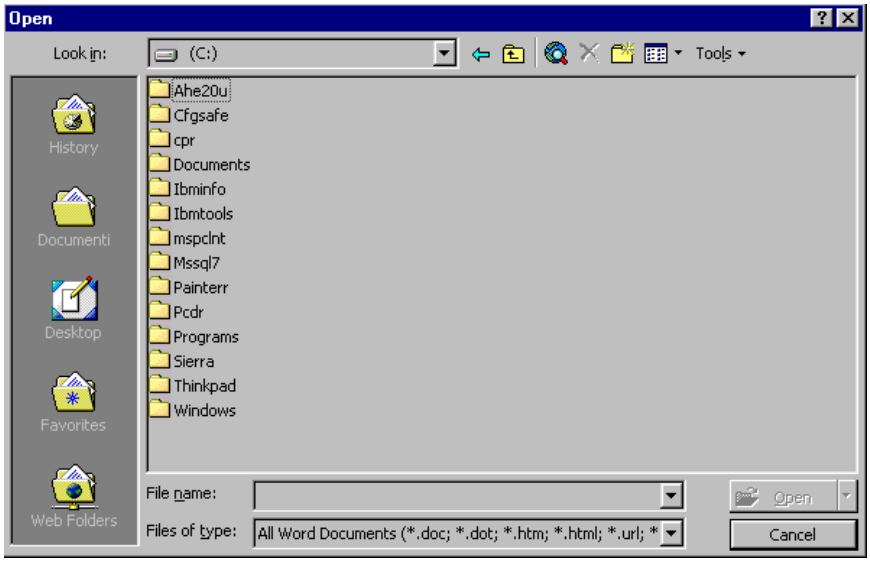
Must contain the path and the MS Word Document name that must be opened in the 'Property' area. E.g. if you want to ask for DOCFILE , you need to define the following:

```
var='W_DOCFILE'
```

2.2.6 Richiesta BitMap (Ask Bitmap)



The 'Richiesta Bitmap' object displays a button to call the 'Open' dialog window so that a Bitmap file can be selected. The object is associated to the 'Bitmap Images' class so that images can be displayed as Bitmaps or Icons (*.ICO)



To use this object you need to define the following values:

Property

In the 'Property' area you need to define the working variable containing the name and path of the selected Bitmap image. The syntax used is: *Var='w_???'*

2.2.7 Immagine Bitmap (Display Bitmap Image)

The 'Immagine Bitmap' object displays a Bitmap image or Icon. To use this object you need to define the following:

Calc

In the 'Calc' field define the working variable containing the file name and path. The syntax used is: *'??? nome bitmap'*

Translation '??? bitmap name'

Property

In the 'Property' field define the name and path of the Bitmap image or Icon that must be displayed. The syntax used is: *'bitmap di default'*

Translation 'default bitmap'

2.2.8 Documento MS Word (MS Word Document)

The 'Documento MS Word' object creates/opens a MS Word document. To use this object you need to define the parameters as follows:

Calc

In the 'Calc' field you need to define the name of the MS Word document that must be created or opened. The name must be typed between apexes. You can also define a working variable containing the file name and path. In this latter case the apexes are NOT required. The syntax used is: *'??? nome documento'*

Translation '??? document name'

2.2.9 Zoom

The 'Zoom' object adds integrated zooms in dialog windows. To use this object you need to define the parameters as follows:

Calc

In the 'Calc' field add the variable, field, or expression that triggers the re-execution of the query. Clicking the '?' button next to the field you can select the file from a pick-list.

If the dialog window has the variable/field 'DESART' and you want to re-execute the query every time that 'DESART' is changed you need to define 'w_DESART' in the 'Calc' field.

The syntax used is:'al variare di questa espressione riesegue la query'

Translation'when this expression changes re-execute the query'

Events

The 'Events' field defines the event or the list of events divided by commas that trigger the routine procedure. This field is optional. Clicking the '?' button next to the field the list of available events is displayed. The list of events is stored in the EVENTS.CPL file under the CLASSES directory. The syntax used is: *'??? Evento_1, Evento_2, Evento_n...'*

Translation '??? Event_1, Event_2, Event_n...'

Property

The 'Property' field contains the parameters to define values basing on the following syntax:*bAdvOptions=.t.,bReadOnly=.t.,cTable='tabella',cZoomFile="",bOptions=.t.*

TranslationbAdvOptions=.t.,bReadOnly=.t.,cTable='table',cZoomFile="",bOptions=.t.

Parameter Name	Description
bAdvOptions=.t.	Flag that activates the configuration options button ('Options').
bReadOnly=.t.	Determines whether zoom rows must be edited or not.
cTable='nome_tabella'	Table name on which the zoom is executed.
cZoomFile=.f.	Configuration Name
bOptions=.t.	Flag that activates various zoom buttons, namely 'Ask Parameters', 'Configuration', Execute Report, etc.

The 'cTable' variable contains the table name on which the zoom is executed, i.e. the name of the table that in the configuration file comes before '_VZM'. To work on the configuration file of the zoom *'MyConfiguration.<Transactions>_vzm'*, the value 'Transactions' must be associated to the 'cTable' variable. You therefore need to type cTable='Transactions'.

N. B.

If you want to use a memory cursor in MS Visual FoxPro you can simply type the cursor name in the 'cTable' variable. The cursor must not be stored in the application databases. The system does not find the cursor in any database and therefore searches in the memory for an active cursor having the same name .

The 'cZoomFile' variable contains the configuration name that must be used. The default value .f. allows using the configuration of the default zoom that has been created for the table defined in 'cTable'. The system will search for the configuration *<Table_Name>_vzm'*.

You can also use a custom zoom configuration typing the zoom configuration name between apexes next to the variable 'cZoomFile='. To associate the configuration file *'MyConfiguration.<Table_Name>_vzm'* to the zoom you need to type 'MyConfiguration' next to 'cZoomFile='. If the zoom is based on a query the system will search for query parameters in the dialog window from which the zoom is executed. The zoom result is saved in the memory cursor "**cCursor**", which contains the extracted records

Let us assume you integrated a zoom in 'Items' that filters all transactions of the item you are working on. The filter parameter for the Visual Query is 'w_ARCODART'. The variable is defined in the 'Items' window and in the query in the 'Field Name' of the 'Filter Parameter' tabstrip. The 'Example' area in the 'Filter' tabstrip has the variable '?w_ARCODART'. The Visual Query will not use the working variable defined in 'Items'.

You can use values defined in the zoom associating a name, i.e. reference, to the object. Adding a unique identifier in the 'Ref' field a working variable representing the object is created. You can thus create reference zoom fields using the **GetVar().** function.

To carry over values of zoom fields in dialog window fields or variables, you need to type 'MyZoom' in the 'Ref. Field'. The variable 'w_MyZoom' is automatically created and values are read with the 'GetVar' function using the following syntax: *w_Myzoom.getvar('<FieldName>')*. To associate the zoom field value to the dialog window field or variable, you need to define the field or variable as 'Calculate' and in the 'Cal/Init/Def' field type: w_Myzoom.getvar('<FieldName>'), whereby the *<FieldName>* is the name of the zoom field that must be carried over.

You can edit and change zoom rows setting the parameter bReadonly to .f. Changes to zoom rows affect the temporary file and are not saved.

2.2.10 Zoom con selezione (Zoom With Selection)

The 'Zoom con selezione' object adds integrated zooms having a selection checkbox. To use this object you need to define the parameters as follows:

Calc

In the 'Calc' field add the variable, field, or expression that triggers the re-execution of the query. Clicking the '?' button next to the field you can select the file from a pick-list.

If the dialog window has the variable/field 'DESART' and you want to re-execute the query every time that 'DESART' is changed you need to define 'w_DESART' in the 'Calc' field.

The syntax used is:'al variare di questa espressione riesegue la query'

Translation'when this expression changes re-execute the query'

Events

The 'Events' field defines the event or the list of events divided by commas that trigger the routine procedure. This field is optional. Clicking the '?' button next to the field the list of available events is displayed. The list of events is stored in the EVENTS.CPL file under the CLASSES directory. The syntax used is: *'??? Evento_1, Evento_2, Evento_n...'*

Translation '??? Event_1, Event_2, Event_n...'

Property

The 'Property' field contains the parameters to define values basing on the following syntax:*bAdvOptions=.t.,bReadOnly=.t.,cTable='tabella',cZoomFile=" ",bOptions=.t.*

TranslationbAdvOptions=.t.,bReadOnly=.t.,cTable='table',cZoomFile=" ",bOptions=.t.

Parameter Name	Description
bAdvOptions=.t.	Flag that activates the configuration options button ('Options').
bReadOnly=.t.	Determines whether zoom rows must be edited or not.
cTable='nome_tabella'	Table name on which the zoom is executed.
cZoomFile=.f.	Configuration Name
bOptions=.t.	Flag that activates various zoom buttons, namely 'Ask Parameters', 'Configuration', Execute Report, etc.

The 'cTable' variable contains the table name on which the zoom is executed, i.e. the name of the table that in the configuration file comes before '_VZM'. To work on the configuration file of the zoom *'MyConfiguration.<Transactions>_vzm',* the value 'Transactions' must be associated to the 'cTable' variable. You therefore need to type cTable='Transactions'.

N. B.

If you want to use a memory cursor in MS Visual FoxPro you can simply type the cursor name in the 'cTable' variable. The cursor must not be stored in the application databases. The system does not find the cursor in any database and therefore searches in the memory for an active cursor having the same name.

The 'cZoomFile' variable contains the configuration name that must be used. The default value .f. allows using the configuration of the default zoom that has been created for the table defined in 'cTable'. The system will search for the configuration *<Table_Name>_vzm'*.

You can also use a custom zoom configuration typing the zoom configuration name between apexes next to the variable 'cZoomFile='. To associate the configuration file *'MyConfiguration.<Table_Name>_vzm'* to the zoom you need to type 'MyConfiguration' next to 'cZoomFile='. If the zoom is based on a query the system will search for query parameters in the dialog window from which the zoom is executed.

Let us assume you integrated a zoom in 'Items' that filters all transactions of the item you are working on. The filter parameter for the Visual Query is 'w_ARCODART'. The variable is defined in the 'Items' window and in the query in the 'Field Name' of the 'Filter Parameter' tabstrip. The 'Example' area in the 'Filter' tabstrip has the variable '?w_ARCODART'. The Visual Query will not use the working variable defined in 'Items'.

Records selected with the Zoom are saved in the temporary cursor '**cCursor**'. The cursor has as many fields '**XCHK**' as records defining whether the record has been selected (Value 1) or not (Value 0).

You can use values defined in the zoom associating a name, i.e. reference, to the object. Adding a unique identifier in the 'Ref' field a working variable representing the object is created. You can thus create reference zoom fields using the **GetVar().** function.

```
Using the object reference you can pass on the cursor name and the
selected records to a routine procedure. For example, if you defined
the 'Ref' field as 'MyZoom' and you want to pass on the cursor name to
the routine procedure, you need to recall the procedure using the
following syntax: <RoutineName>(w_Myzoom.cCursor)
```

You can edit and change zoom rows setting the parameter bReadonly to .f. Changes to zoom rows affect the temporary file and are not saved.

The 'Zoom con selezione' object notifies four different events. These events allow defining complex procedures and are defined in the following table:

EVENT	DESCRIPTION
"<ZoomName> row checked"	When a row is selected
"<ZoomName> row unchecked"	When a row is deselected
"<ZoomName> after query"	Before the query is executed
"<ZoomName> before query"	After the routine procedure has been executed

2.2.11 Grafico (Graph)

The 'Grafico' object adds graphs linked to Visual Query data. To use this object you need to define the parameters as follows:

Calc

In the 'Calc' field define the field, variable, or expression that triggers the query re-execution. Clicking the '?' button next to the field you can select the required element from a pick-list. If e.g. you want to re-execute the query when the field 'ARDESART' in the dialog window is updated you need to type 'w_ARDESART' in the 'Calc' field. The syntax used is:*'al variare di questa espressione riesegue la query'*

Translation'when this expression changes re-execute the query'

Events

The 'Event' field defines the event or the list of events divided by commas, that trigger the routine procedure. The field is optional. Clicking the '?' button next to the field the list of available events is opened. The list is read from the EVENTS.CPL file under the CLASSES directory. The syntax used is: *'??? Evento_1, Evento_2, Evento_n...'*

Translation '??? Event_1, Event_2, Event_n...'

Property

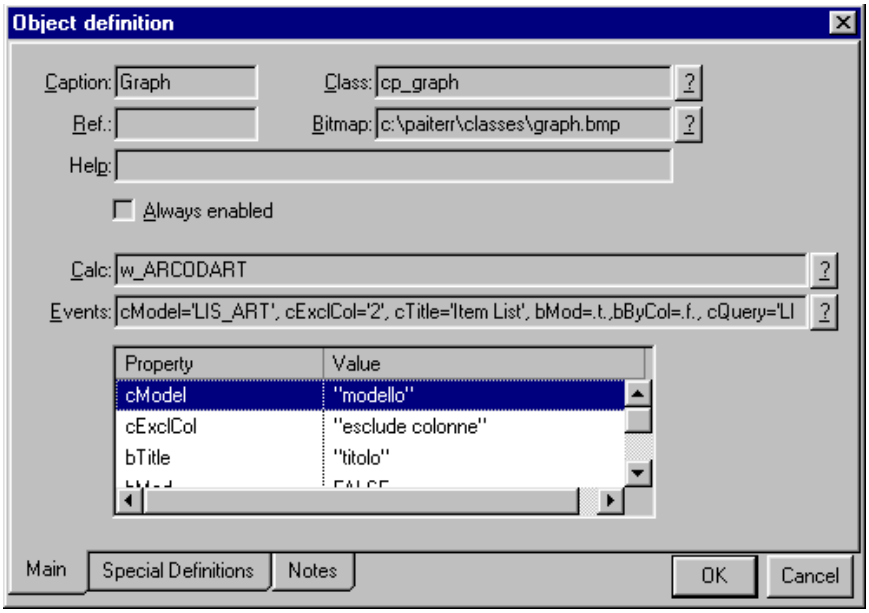
The 'Property' area contains parameters for the definition of values. The syntax used is:cModel='modello', cExclCol='esclude colonne', bTitle='titolo', bMod=.f., bByCol=.f., cQuery='query/cursore'

TranslationcModel='model', cExclCol='exclude columns', bTitle='title', bMod=.f., bByCol=.f., cQuery='query/cursor'

Parameter Name	Description
cModel='modello' ('model')	Graph model name with or without the extention 'VGR'.

cExclCol='esclude colonne' ('exclude columns')	Columns of source data (Query) that must be excluded from the graph. The format used is: <Column Number>, [<Column Number>]. Defining for example cExclCol='1,5,8' will exclude the columns 1 5 and 8.
cTitle='titolo' ('title)	Graph title. It will be ignored if a title has been already given to the graph model.
bMod=.f.	When the parameter is defined as FALSE the graph is 'display only'. When the parameter is defined as TRUE the graph can be modified double clicking on it.
bByCol=.f.	When the parameter is defined as FALSE the graph is created 'by row'. When the parameter is defined as TRUE the graph is created by column, which is useful when dealing with queries that have no Pivot functionalities.
cQuery='query/cursore' ('query/cursor')	Name of the Query or of the Memory Cursor on which the graph creation is based. Defining the extention as 'VQR' it will be interpreted as Visual Query. If the extention is left blank it will be interpreted as Cursor.

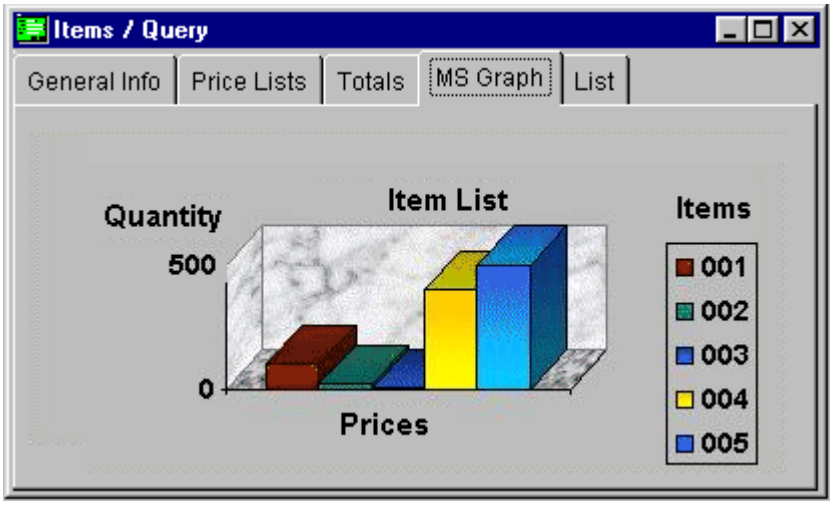
To display the graph 'LIS_ART' in 'Items' you need to add a 'Grafico' object in the 'Items' dialog window and define the parameters as shown in the following picture:



In particular the 'Property' area has been defined as:

```
cModel='LIS_ART', cExclCol='2', cTitle='Item List',
bMod=.t.,bByCol=.f., cQuery='LIS_ART.VQR'
```

When the field w_ARCODART is updated the query LIS_ART.VQR is re-executed and data is passed on to the graph having the same name. The column 2 is excluded (cExclCol='2'), the graph title is the one defined in cTitle, the graph can be changed (bMod=.t.) and has been created by row (bByCol=.f.). The end result should look like the following picture:



2.2.12 TreeView

The Issue

Structured data can not always be easily displayed because it is usually complex. You need to identify the components of structured data and again the components of components, etc. down to the lowest level. Data is typically organized using two structures linked by a Parent/Child relationship, no matter if the link is between two entities (a Master File and a Detail File) or within a Master Detail Entity.

The Solution

The *'Treeview'* object has been developed to graphically display structured data. When the object is integrated into a dialog window, it displays data that is passed on using the treeview. It is therefore fundamental that the data has a clearly defined structure. To achieve this you need to define the level of each component. If you think e.g. of a Personal Computer, the PC will be the first level, the mouse the second level and the mouse ball the third level as it is a component of the mouse.

When data is saved the level of the component is still unknown. This is why the 'Treeview' object is supported by a routine procedure that structures data. The routine gathers all components in a cursor and to each component it adds a field defining the level. The 'Treeview' object processes the cursor and produces the graphical organization of data.

Solution Implementation

Routine

The routine procedure supporting the 'Treeview' object is made of two main instructions:

- the creation of a temporary cursor containing data that can be processed and then displayed;
- the creation of a cursor that contains a field that defines the treeview level at which the record must be placed.

To create the temporary cursor you need to define a query for the data selection (using the 'Query Painter') and to execute with a routine the following 'external program':

```
VQ_EXEC WITH "<query_name>", THIS, "<cursor_name>"
```

The query must always work on the 'Parent' database, which is selected depending on the result required. The cursor containing the Visual Query result must be passed on to one of the following external programs: **cp_ExpDB**, or **cp_Level**, whereby **cp_ExpDB** is called to display data of Bills Of Materials and **cp_Level** to display the structured data as treeview.

These two routines do not necessarily need to be integrated in the 'Treeview' object. They can be also used to create cursors that process data and order it in levels.

Bills Of Materials (BOMs) are a typical component of commercial/business applications. BOMs are complex and structured on multiple levels. Developing BOMs is usually costly in terms of time and resources. In CodePainter BOMs are readily available simply implementing the routine procedure **'cp_ExpDB'**.

The 'cp_ExpDB' routine creates cursors that can be used to treeview BOMs or to cancel possible BOMs balances. The routine basically adds two fields to the cursor:

- **LvlKey**: character field defining the level of the record within BOMs. This field is used when the cursor is displayed in the 'Treeview' object.
- **QtaComp**: numeric field defining the sum of the table's field containing the total. This field has been defined in the routine as parameter. E.g. to create a BOM for warehouse items, the table must detail the item components. The total number of components making up the item could be one of the total fields.

The call to the routine procedure **cp_ExpDB** must be integrated:

- In the 'Exec' command of the Routine Painter or of procedures having the 'External Prg.' option.
- In CodePainter Painters where the call to a 'Program' object is defined.
- In the manual areas.

The parameters of the 'cp_ExpDB' class are the following:

Parameter Name	Description
cOutCursor	Output cursor name that will contain the data.
cInpCursor	Input cursor name that is opened. Usually the cursor is the result of a Visual Query executed on the 'Parent' object. The cursor fields are a subset of the fields in the reference table 'cRifTable'. Therefore the cursor can not be the result of a join between other tables.
cRifTable	Name of the referenced table. Usually it is the 'Parent' table defined in the 'Parent/Child' relationship.
cRifKey	List of reference table key fields involved in the link between the 'Parent/Child' and the table that must be opened. The syntax used is: '<campo chiave 1>, ..., <campo chiave n>'. Translation '<key field 1>, ..., <key field n>
cExpTable	Name of the table that must be opened. This table contains fields and data required to open the BOM. Usually it is the 'Child' table defined in the 'Parent/Child' relationship.
cExpKey	List of key fields of the table that must be opened, involved in the link between the 'Parent/Child' and the reference table. The syntax used is: '<campo chiave 1>, ..., <campo chiave n>'. Translation '<key field 1>, ..., <key field n>

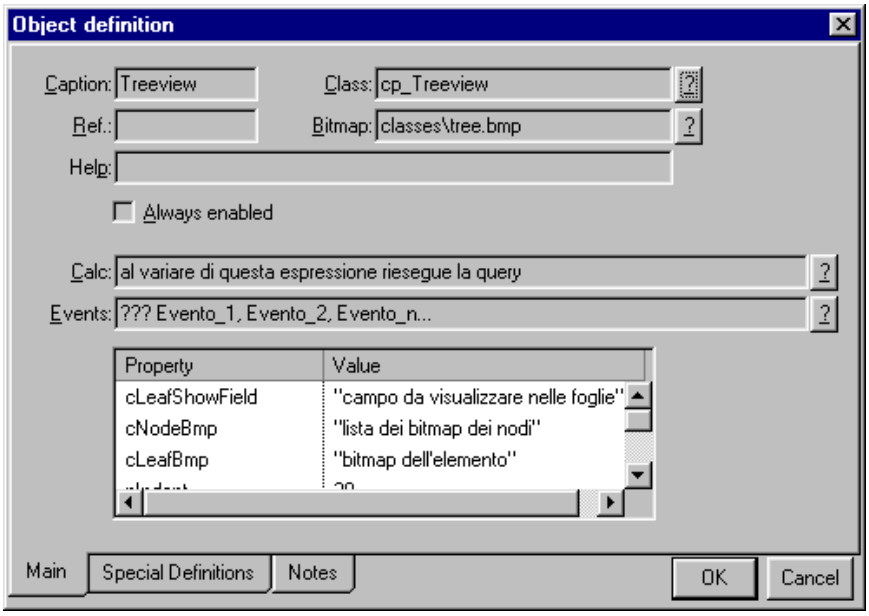
cRepKey	List of repeated key fields of the table that must be opened, referring to the the 'Parent' table key fields defined in the parameter 'cRifKey'. Usually these fields are involved in a 'Relationship' link with the referenced table. The syntax used is: '<campo chiave 1>, ..., <campo chiave n>". Translation '<key field 1>, ..., <key field n>
cExpField	Name of the total field of the table that must be opened. The field must be numeric. The value contained in the field is multiplied by the value contained in the same field of the record which is one level above. The total is displayed in the output cursor field 'QtaComp'.
cOtherField	Name of the fields of the table that must be opened. These fields must be included in the cursor result 'cOutCursor'.

In routines supporting the 'Treeview' object you need to define the call to an 'external Program' and name it **"cp_Level"**. This class returns a cursor where the character field 'LvlKey' has been added. This field defines the unique level of the record. The cursor is displayed in the 'Treeview' object as input parameter. The parameters of the class 'cp_Level' are the following:

Parameter Name	Description
cCursor	Name of the cursor that creates the field that defines the level.
cCursorNodes	List of fields of the cursor 'cCursor', separated by commas, having a node functionality.
cTotField	List of node total fields, separated by commas.

The Treeview Object

No matter whether you are defining treeviews or BOMs, when you add a 'Treeview' object the 'Objects Definition' window must be always defined as shown in the next picture. You can distinguish the use of the 'Treeview' object when you define the 'External Program' ('cp_ExpDB' or 'cp_Level').



In the 'Object Options' you need to define the parameters as follows:

In the 'Calc' field you need to define the variable, field, or expression that triggers the query re-execution. Clicking the '?' button next to the field the list of available working variables is displayed. For example if you want that the query is re-executed when the field 'ARDESART' in the dialog window is up-dated, in the 'Calc' field you need to define the variable 'w_ARDESART'. The syntax displayed is:*'al variare di questa espressione riesegue la query'*

Translation'when this expression is changed re-execute the query'

In the 'Events' field define the event or list of events divided by commas, that trigger the routine execution. This field is optional. Clicking the '?' button the list of available events stored in the CLASSES.CPL file are displayed. The syntax used is:*'??? Evento_1, Evento_2, Evento_n...'*

Translation'??? Event_1, Event_2, Event_n...'

The 'Property' area contains the parameters that must be passed on to the template. The syntax used is:*cCursor='cursore',cShowFields='lista dei campi da visualizzare'*

TranslationcCursor='cursor',cShowFields='list of fields to be displayed'*,cNodeShowField='campo da visualizzare nei nodi',cLeafShowField='campo da visualizzare nelle foglie'*

Translation,cNodeShowField='field to be displayed in the nodes',cLeafShowField='field to be displayed in the subdirectories'*,cNodeBmp='lista dei bitmap dei nodi',cLeafBmp='bitmap dell'elemento'*

Translation,cNodeBmp='list of the node bitmaps',cLeafBmp='element bitmap'

Parameter Name	Description
cCursor	Name of the cursor created, i.e. 'cp_ExpDB' or 'cp_Level'. In both cursors there must always be an indexed level field named 'LvlKey' that has the following syntax: <nodo 1>. <nodo 2>. ... .<nodo n>. Translation <node 1>. <node 2>. ... .<node n>.

cShowFields	String containing the list of cursor fields divided by commas that must be displayed in the various levels of the treeview. If there are n levels and m fields the level x will display the field having the sequence number calculated with the following formula: x=n module m.
cNodeShowField	String containing the name of the cursor field that must be displayed in the nodes together with fields defined in the parameter 'cShowFields'.
cLeafShowField	String containing the name of the cursor field that must be displayed in the leafs, together with fields defined in the parameter 'cShowFields'.
cNodeBmp	List of bitmap files (<bitmap name>.bmp) divided by commas, that will be added to the treeview nodes having elements. If there are n levels and m bitmaps, the level x will display the bitmap having the sequence number calculated with the following formula: x=n module m.
cLeafBmp	Name of the bitmap file (<bitmap name>.bmp) that must be displayed in the treeview leafs.

The field name that is added to the treeview parameter must be a valid MS Visual FoxPro instruction containing the file name and other data. For example:

```
cNodeShowField="arcod_art"


cNodeShowField=""Item "+Trim(arcod_art)+"
.Price:"AllTrim(Str(prz_art))"
```

The second half of the parameter definition in the example above completes the information on the node that can thus be read.

```
cShowFields='" "+ardes_art'

cNodeShowFields='"Code: "+arcod_art'

cNodeShowFields='"Price:"+Trim(Str(prz_art))'
```

Using this definition the fields of the parameters 'cNodeShowField' and 'cShowField' are displayed in the subelement nodes. The fields of the parameters 'cLeafShowField' and 'cShowField' are displayed in the leafs.

Using the Treeview in the application is very easy and similar to the standard MS Windows Explorer in Windows 98. The structure looks like a tree. When the Treeview is opened only the main nodes are displayed (roots). You can extend them clicking on the '+' next to the nodes. The node detail can have other nodes that again can be expanded in the same way, or final elements (leafs) that can no longer be expanded. When you expand a node the '+' next to it turns into a '-'. Clicking on the '-' you can close the node's detail.

The Treeview 'GetVar ()Method' allows building references on field values included in the Treeview. Open the 'Object Options' window and define a unique identifier in the 'Ref.' field. A working variable representing the object is created. Through this variable you can access the 'GetVar()' and other element's methods and properties. Using the syntax *w_<rif_treeview>.getvar('<FieldName>')* you can access the field values of the element you selected in the treeview.

Treeview objects generate two kind of events. The first is called **NodeClick** and the second **NodeRightClick**. The 'NodeClick' event is generated double clicking an element. The 'NodeRightClick' is generated right clicking an element.

The event notification is transparent to the application execution. To have some impact on the application events need to be associated to objects that can answer the notification, like e.g. the *Esegui procedura Batch (Execute Routine)*. In the 'Object Options' window of the answering object you need to define the treeview reference name (in capital letters) followed by the name of the event. The syntax is: *RIF_TREEVIEW EventName*

Examples

Here to follow you will find the example of an application that gives you a complete overview on the 'Treeview' object.

Bill Of Materials

To build the Bill Of Materials (BOMs) you need to define two entities: a Master File named 'Items' and a Detail File named 'Components'. Define two links: a 'Relationship' link and a 'Parent/Child Relationship'. The 'Relationship' link connects the primary key (arcod_art) of the 'ArtDB' table in 'Items' and the repeated primary key (art_com) of the 'CompDB' table in 'Components'. The 'Parent/Child Relationship' links the primary key (arcod_art) of the 'ArtDB' table and the unrepeated primary key (arcod_com) of the 'CompDB' table.

The BOM 'Treeview' dialog window will be made of the objects 'Treeview' and 'External Program'.

The 'External Program' object calls a routine when the dialog window is opened (triggered by the Event **Init**) and creates a cursor containing the data that must be displayed. The routine associated to the object is as follows:

```
vq_Exec with "BOMQ", this, "BOM"

cp_ExpDB with
"View","BOM","ArtDB","arcod_art","CompDB","cod_com","art_cod","qta_com
","qta_com"

If used("BOM")

   select BOM

   use

End if
```

The first instruction executes the **External Program** and creates the cursor named 'View' that contains the fields of the query 'BOMQ'. The 'BOMQ' query selects all fields of the table 'ARTDB'. The SQL Sentence used is:

```
SELECT artDB.cod_art, artDB.des_art, artDB.prz_art FROM artDB
```

The cursor is created using a query that has been build with the Query Painter and is therefore transparent to all kinds of databases.

'cp_ExpDB' is the 'External Program' execution that creates the cursor 'View' containing detailed data and a further field detailing the record level. The parameters are:

- **View:** name of the temporary cursor containing detail data. The cursor has been previously created with the **SELECT** instruction.
- **BOM:** name of the cursor in which data must be entered.
- **ArtDB:** name of the basic table from which data is selected.
- **cod_art:** primary key of the basic table.
- **CompDB:** name of the table from which detail data is selected.
- **cod_com** and **art_cod:** fields making up the key of the detail table ('CompDB').
- **qta_com:** name of the detail file of the detail table (optional).
- **qta_com:** quantity field name of the components table (optional).

The **If** instruction checks whether the 'BOM' cursor is opened. If so the instruction selects and closes it.

After having created the cursor that will contain the data that must be viewed you can define the 'Treeview' parameters.

Add a 'Treeview' object and in the 'Property' area define the following parameters:

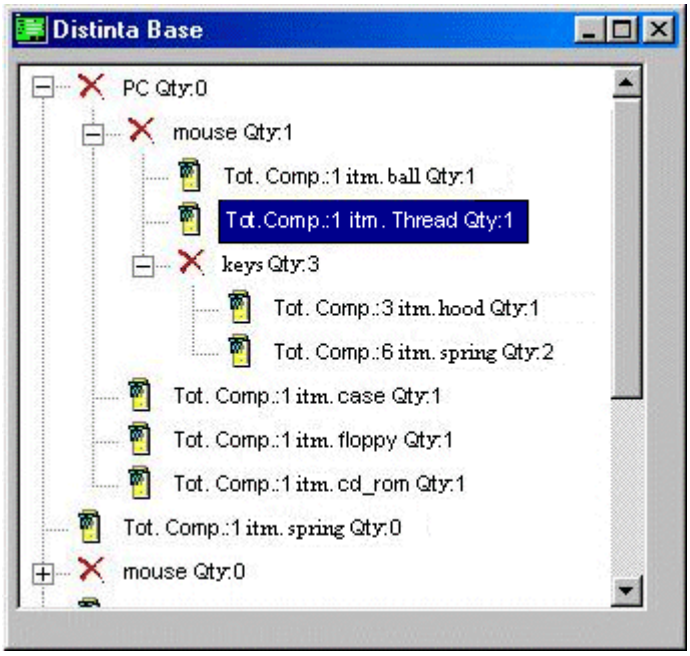
'cCursor' as 'View'.

'cShowField as 'trim(ardes_art)+" Qta:"+alltrim(str(qta_com))', which is the string with no blanks containing the values displayed on the different levels. The 'qta_com' field refers to the quantity field of the 'Components' table.

cLeafShowField as 'Tot. Comp.:"+alltrim(str(qtacomp))+" art.', which is the string with no blanks containing the values that will be added to strings of the various 'Treeview' levels.

The 'qta_com' field refers to the field of the external procedure 'cp_ExpDB' containing the overall total of the 'Components' table.

The bitmap associated to the parameter 'CbmpNode' is 'esc.bmp' and the one associated to 'cBmpElement' is 'word.bmp'. The final result should look like the following picture.

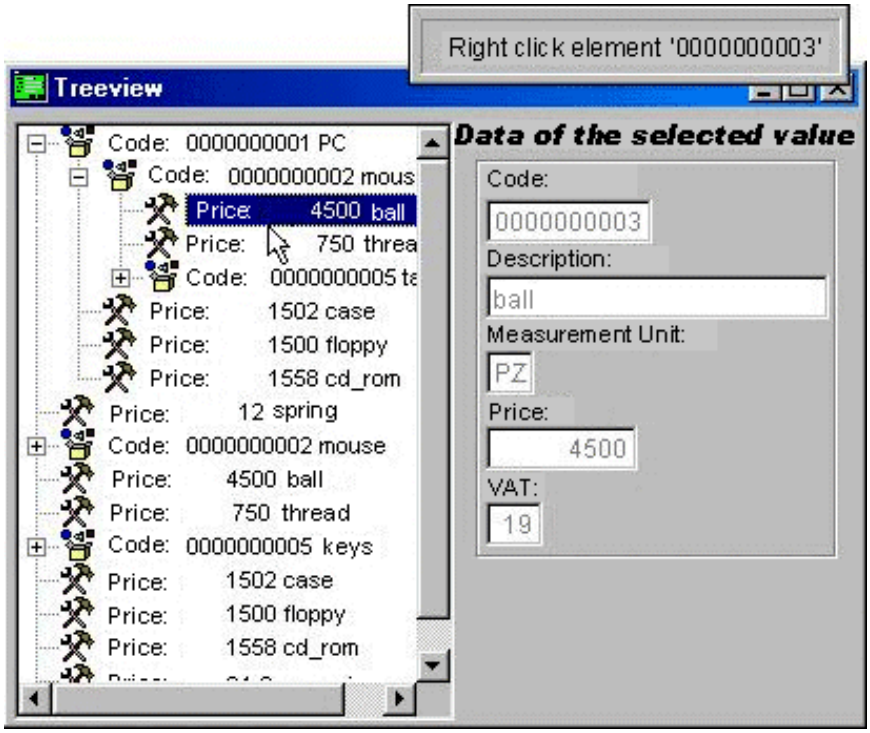


Now change the example to use the *GetVar()* method of the *NodeRightClick* event.

In the 'Treeview Object Options' window type 'TreeArt' in the 'Ref.' field. The working variable *w_TreeArt* is thus created. The field values of the selected object are read in the 'Treeview' using the syntax: w_<rif_treeview>.getvar('<FieldName>'). This means that in this example the read values are: "arcod_art", "ardes_art", "udm_art", "prz_art", "iva_art", following the syntax w_TreeArt.getvar('<FieldName>')

To manage the answers to the events you need to add a *Esegui procedura Batch (Execute Routine)* object which executes a procedure when the event **NodeRightClick** is triggered. In the 'Execute Routine' object option window add the string *TREEART NodeRightClick* in the **Events** field. In the 'Property' area in the parameter *prg* type the procedure name 'RClick'.

The *RClick* procedure opens the warning message used by the Visual FoxPro instruction *wait wind "Right mouse key of the selected element"*. As you can see in the next window the bitmap parameters has changed as well as the **cleafShowField** parameter that now displays the price instead of the quantity.



Basic Treeview

You are now required to display the quantity of items ordered by various customers divided by order form. In the application design define three entities, namely two Master Files 'Customers' and 'Items' and a Master/Detail 'Orders'. The three entities are linked by two 'Relationship' links. The first link is between the primary key (cod_cli) of the 'CliDB' database ('Customers') and the customer field (cli_ord) in the 'OrdDB_m' table ('Orders'). The second link is between the primary key (cod_art) of the 'ArtDB' table and the 'Items' field (ord_art) in the 'OrdDB' table.

The dialog window of the 'Basic Treeview' is made of the objects 'Treeview' and 'External Program'.

The 'External Program' routine calls a procedure when the dialog window is opened and creates a cursor that contains the data that must be displayed. The routine procedure associated to the object is:

```
vq_Exec with "ViewQ", this, "View"


cp_Level with "View", "art_ord, cli_ord", ""


The first instruction executes the External Program that creates the
cursor View containing the fields of the ViewQ query previously
created with the Query Painter. The 'ViewQ' query selects the same
fields from the tables using the following SQL sentence:SELECT
ordDB_m.cod_ord, ordDB_m.cli_ord, cliDB.cli_des, ordDB.art_ord,
artDB.des_art, SUM(ordDB.qta_ord) AS qta_ord, FROM (((ordDB RIGHT
OUTER JOIN ordDB_m ON ordDB_m.cod_ord=ordDB.cod_ord) LEFT OUTER JOIN
artDB ON artDB.cod_art=ordDB.art_ord) LEFT OUTER JOIN cliDB ON
cliDB.cod_cli=ordDB_m.cli_ord) GROUP BY ordDB.art_ord,
ordDB_m.cli_ord, ordDB_m.cod_ord
```

The parameters of the **vq_Exec** procedure are:

- **ViewQ** name of the query from which the cursor is created.
- **this** is the pointer to the 'Routine' entity.
- **View** is the name of the cursor that will contain detailed data.

'cp_Level' is the execution of an **External Program** that creates the cursor 'View'. The cursor contains the detailed data and the **LvlKey** field defining the record level. The parameters used are:

- **View** is the name of the temporary cursor previously created by the **'vq_Exec'** procedure containing the detailed data.
- **art_ord, cli_ord** is the list of fields of the created cursor. They are the nodes of the treeview.
- **""** no field is passed through as this nodes 'total' parameter is optional.

Now that the cursor has been defined you can define the 'Treeview' parameters. In the 'Object Definition' window go to the 'Property' area and in **cCursor** type the cursor name **View**. In the **cShowField** parameter type the three values that define the three levels separated by commas, namely:

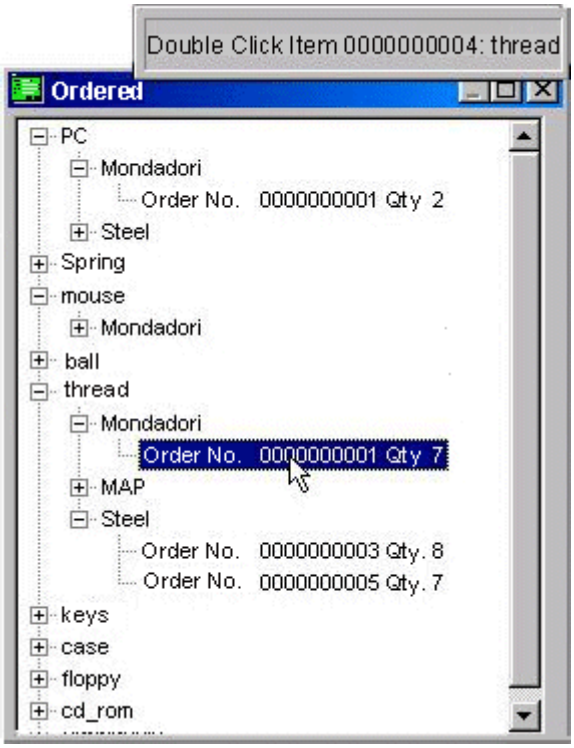
des_art, i.e. the item description - first level,

des_cli, i.e. the customer description - second level, and

'Order No.' '+cod_ord+' Qty '+ alltrim(str(qta_ord)), i.e. the string associated to the third level defining the order number in which the item and the ordered quantity are stored.

The other parameters are left empty.

The added *Esegui procedura Batch (Execute Routine)* object is triggered by the event **NodeClick**. In the *Treeview* object you need to define the 'Ref.' field in the 'Object Options' window as 'DblClick'. In the 'Object Definition' window of the 'Execute Routine' object you need to type the string *DBLCLICK NodeClick* in the 'Events' field, and the procedure name 'DoubleClick' in the *prg* parameter in the 'Property' area. The *DoubleClick* procedure opens the Visual FoxPro warning message *wait wind "Double Click on the selected item"*. The result should look like the following picture.



Technical Notes

The *Treeview* object instances the 'cp_Treeview' class, which is stored in the file 'cp_class.prg.' The dialog window where you add this object will include a 'Treeview' control.

2.2.13 Timer

Business/Commercial Software Applications sometimes need to manage time-controlled activities. For example the application may have to automatically close down a dialog window after a given time-period, or it may have to start routine procedures at a given time every morning/ evening.

To facilitate the job to Software Developers, CodePainter introduces the **Timer** object. When this object has been integrated into a dialog window it notifies events basing on defined dates and times or in time-lapses. The main activities performed by this object are:

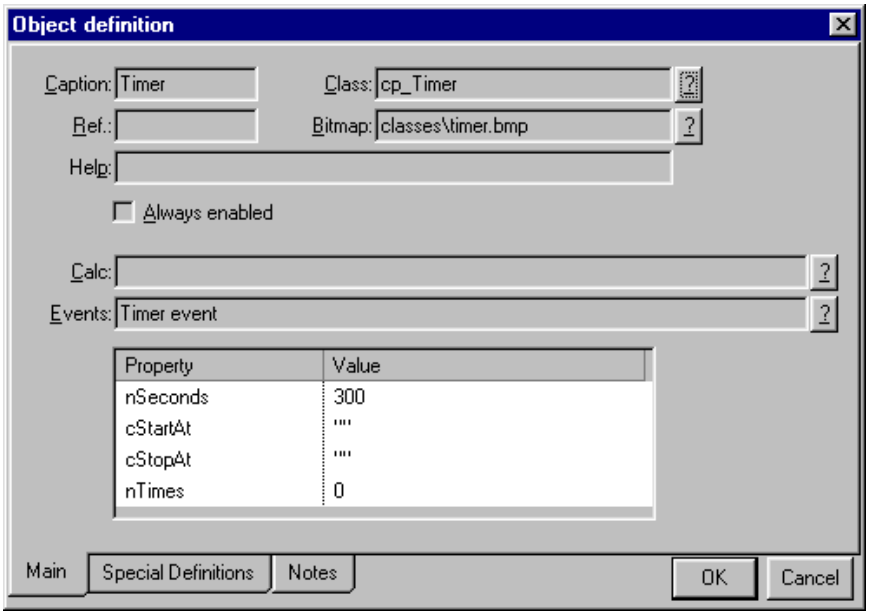
- checking whether this event must be notified or not;
- notifying the event.

To define the trigger of the event there are four parameters defininig intervals (in seconds) in which the objects must check whether the event must be notified or not; the number of times the event must be notified; the date and time during which the event must no longer be notified. The first parameter defines the deadline for checking the other parameters. If these latter ones are fulfilled the event is notified.

The 'Timer' object simply notifies an event and must therefore be associated to another object that answers to this event.

Implementation Options

The **'Timer'** Object Options window is shown in the following picture:



To use the 'Timer' object you need to define the following parameters.

Events

In the 'Events' field define the name of the event that must be notified by the 'Timer' object. The syntax used is:*Timer Event*

Property

In the 'Property' area you need to define the intervals in which parameters must be passed on to the template to notify the event. The syntax used is: *nSeconds=300,cStartAt='',cStopAt='',nTimes=0*

Parameter Name	Description
nSeconds	Time intervals expressed in seconds on which the control on other parameters must be executed. If this parameter is set to '0' the control is initialized every 300 seconds but will never be executed.
cStartAt	Date and time from which the object must start notifying the event.
cStopAt	Date and time by which the object must stop notifying the event.
nTimes	Number of times that the event must be notified. If this parameter is set to '0' the event will always be notified.

The 'Timer' object is executed only when the dialog window in which it is integrated is opened. Every time the dialog window is closed the associated counters are set to zero. Let us suppose that you define the parameters so that the event is notified only once. If the dialog window at that time is closed the event is not notified. If the dialog window is left opened for over a day, the event is notified only once. Indeed the control on the notification is made also on the event itself. If the event has been already notified the defined number of times, the event is no longer generated. To notify the event every day you need to define also the time when the object must stop checking: counters are set to zero.

Examples

The 'Timer' object is not displayed in the application. The End-user will only see the effects it has on the other associated objects. Here to follow you will find two examples on how the 'Esegui procedura batch (Execute Routine Procedure)' can be associated to a 'Timer' object.

Dialog Window Close Down

To automatically close down the dialog window after 300 seconds you need to add two objects: *Timer* and *Esegui procedura batch (Execute Routine)*.

The 'Timer' must notify the *TimeOut* event only once after 5 minutes. The **Object Options** window must therefore be defined as follows: In the **Events** field type the string *TimeOut*. In the **Property** area type *300* next to **nSeconds**; *''''* next to **cStartAt**; *''''* next to **cStopAt**, and *1* next to **nTimes**.

The 'Execute Routine' must execute the 'CloseWindow' routine when the 'TimeOut' event is notified. The **Object Options** window must therefore be defined as follows: in the **Events** field define the string *TimeOut*. In the **Property** area type the name of the procedure that must be executed between apexes (*'CloseWindow'*) next to **Prg**. The **CloseWindow** routine closes down the window using the *ecpQuit* method of the *Parent.* object.

```
//5 minutes have passed


parent.ecpQuit()
```

Backup

You are now required to activate the back-up procedure every day between 6:00 p.m. and 7:00 p.m.

The **'Timer'** Object Options window must therefore be defined as follows: In the **Events** field type the string *Backup* . In the 'Property' area type *300* next to **nSeconds=** , *'18:00'* next to **cStartAt=**, *'19:00'* next to **cStopAt=** , and *1* next to **nTimes=**.

The 'Execute Routine' object executes the 'RunBakcup' procedure when the 'Backup' event is notified. In the fields **cStartAt** and **cStopAt** no dates have been defined, but only times. When the StopDate is reached the counters are set to zero. The next day the event will be again notified between 6:00 p.m. and 7:00 p.m. Defining specific dates the counters will not be set to zero daily.

The 'Timer' object is always enabled when the application dialog window is in the Load or Change mode. This applies to Master Files, Master/Detail and Detail Files. If you want that the object is enabled also in the 'Query' mode you need to activate the 'Always Enabled' flag.

2.2.14 Displaying The Internet Explorer Browser

Many Business/Commercial applications require a link to a Web site in order to receive information or to download real-time data as e.g. currencies exchange rates.

To facilitate the Software Developers' task, CodePainter introduces two objects:

- "Browser Internet Explorer 4.0".
- "Bottoni per Browser IE 4.0" (Browser IE 4.0 Buttons).

These two objects allow integrating the Internet Explorer Browser 4.0 in your application. The first object will display all documents that the browser can support. The second allows the user to browse the documents that have been already downloaded once. You can integrate these objects in any dialog window, access Internet Explorer and browse on static addresses or addresses that can be taken from a table, from a calculation result or directly from within the application.

Browser IE 4.0

When the *Browser Internet Explorer 4.0* object is added to a dialog window it displays the window of the Internet Explorer 4.0 Browser. You can define a default site on which the window must be opened in the 'Calc' field. The use of this object requires the following values:

The 'Calc' field defines the string variable containing the 'URL' address. The syntax used is:*'Al variare di questa espressione riesegue la navigazione'*

Translation'When this expression changes re-execute the browsing'

The 'Events' field defines the event or the list of events separated by commas. When the events are triggered a routine procedure must be executed. This field is optional. Clicking the '?' button next to the field you can select the event from a pick-list. The list of events is stored in the 'EVENTS.CPL' under the directory 'CLASSES'. The syntax used is. *'??? Evento_1, Evento_2, Evento_n...'*

Translation '??? Event_1, Event_2, Event_n...'

The *Browser Internet Explorer 4.0* object generates the event **NavigationComplete** which is triggered by ending the browsing activity. The event notification is transparent to the application execution. You need to associate the object to another object that answers to the event notification, such as the *Esegui procedura Batch (Execute Routine)* object.

In the 'Object Options' window of the answering object you need to define in the **Events** field the name of the reference object, i.e. the *Browser Internet Explorer 4.0* object using capital letters plus the name of the event. The syntax used is: *<Object Name> NomeEvento*

Translation <Object Name> EventName

In the *Browser Internet Explorer 4.0* object the *cURL* property has been defined, which returns the address loaded by the browser. This property returns the current address to which the object is pointing.

Bottoni per Browser IE 4.0 (Browser IE 4.0 Buttons)

When the *Bottoni per Browser IE 4.0 (Browser IE 4.0 Buttons)* object is added to a dialog window it displays four buttons that allow to browse the Internet using the **"Browser Internet Explorer 4.0"** object. To use this object you need to define the following values.

```
The 'Property' area contains the parameter 'cBrowserName' to which the
name of the "Browser Internet Explorer 4.0" object is associated.
Defining this field you can use the object's buttons as simulators of
the standard Internet Explorer 4.0 buttons. The button functionalities
are Back, Forward, Stop and Refresh. The syntax used
is:cBrowserName='variabile che identifica il Browser IE4.0'


TranslationcBrowserName='variable identifying the IE 4.0 Browser.'
```

Examples

You are now required to add the two Browser objects into one window. You need to define a dialog window and a routine.

HTML Document Dialog Window

The dialog window must have four elements for each browser:

- a string variable
- a *BrowserButtons* object
- a *Browser;* object
- an *External Program.* object.

The variables work as bar where browser addresses are defined. In these variables you can define an http or ftp address to access the desired Web page.

There are two string variables that are initialized an http address. The first variable initializes at an Internet address (e.g.www.codepainter.com), the second an Intranet address (e.g. / / spdnt).

The *Browser* objects contain the Web pages you are accessing.

The **Object Options** windows must be defined as follows: in the **Ref** field define the reference names for the browsers ('browser1' and 'browser2'). In the **Calc** field define the variables containing the URL address to which you want to connect ('address1' and 'address2').

The *BrowserButtons* objects allow browsing the pages that have been already accessed at least once using the functionalities **Forward** and **Back**. Further they allow re-loading a page in the browser using the functionality **Refresh** or to stop the downloading using the functionality **Stop**.

The **Object Options** window are defined as follows. In the 'Property' area define the reference to the *Browser* object next to the parameter **cBrowserName**: *'w_browser1'* and *'w_browser2'*.

```
The 'External Program' object returns the current browser name in the
address bar and is executed when the Browser object has completed the
page download. The Object Options windows must be defined as follows:
in the  Events field define the event NavigationComplete generated by
the first and second browser. In the first object add the string:
BROWSER1 NavigationComplete


and in the second object add the string BROWSER2 NavigationComplete
```

In the **Property** area type *Browse* next to **prg**, i.e. the name of the procedure that must be executed: **prg** *='BROWSE(w_browser1,1)'* and **prg** *='BROWSE(w_browser2,2)'*.

The parameters passed on to the procedure are: the reference variable of the object *Browser*, and the reference to the *Browser* that notifies the event.

The Browse Procedure

The 'Browse' procedure is called when the *Browser* object has completely downloaded the Web page. The Browser returns the current address to the variable of the browser that has called the address using the property *cURL*. The routine is:

```
//Parameter Definition

oBrowser(O)

nBrowser(N,1,0)

//Global Variables Definition

w_address1(C,60)

w_address2(C,60)

If nBrowser=1

   w_address1=<=oBrowser.cURL

Else

   w_address2=<=oBrowser.cURL

End If
```

The result of this example is shown in the following picture.

2.2.15 Controllo Proprieta' Visuali (Check Visual Properties)

The 'Controllo Proprietà Visuali' (Check Visual Properties) object allows to set an object property and a value if required.

To use this 'Controllo Proprietà Visuali' object you need to define the following parameters:

Calc

In the 'Calc' field define the property value. Clicking the '?' button next to the field all available working variables are listed. The syntax used is:*valore per la proprieta'*

Translationproperty value

Property

The 'Property' area contains the variable name (**cObj**) and the property (**cProp**). The syntax used is:*cObj="nome variabile"cProp="propieta'"*

TranslationcObj="variable name"*cProp="property"*

2.2.16 Stringa Calcolata (Calculated String)

The 'Stringa Calcolata' (Calculated String) object allows adding a label to the dialog window.

To use this object you need to define the following parameters:

Calc

In the 'Calc' field define the text, text color and background. The syntax used is:*testo','colore testo (num)','sfondo (num)*

Translationtext','text color (num)','background (num)

Property

In the 'Property' area define the Caption. The syntax used is:*caption= testo della label*

Translationcaption = label text