HMGGEN App Generator

Introduction

HMGGEN is an application generator for Windows applications, based on Harbour.
The environment is simple, logical, intuitive and straightforward, just like the language Harbour and HMG, the Harbour MiniGui developed by Roberto Lopez.

HMGGEN App generator has been build on top of the HMG.
HMGGEN generates:
- workflow processes
- objects
- forms

In addition HMGGEN enables you to define and maintain business rules outside the code.

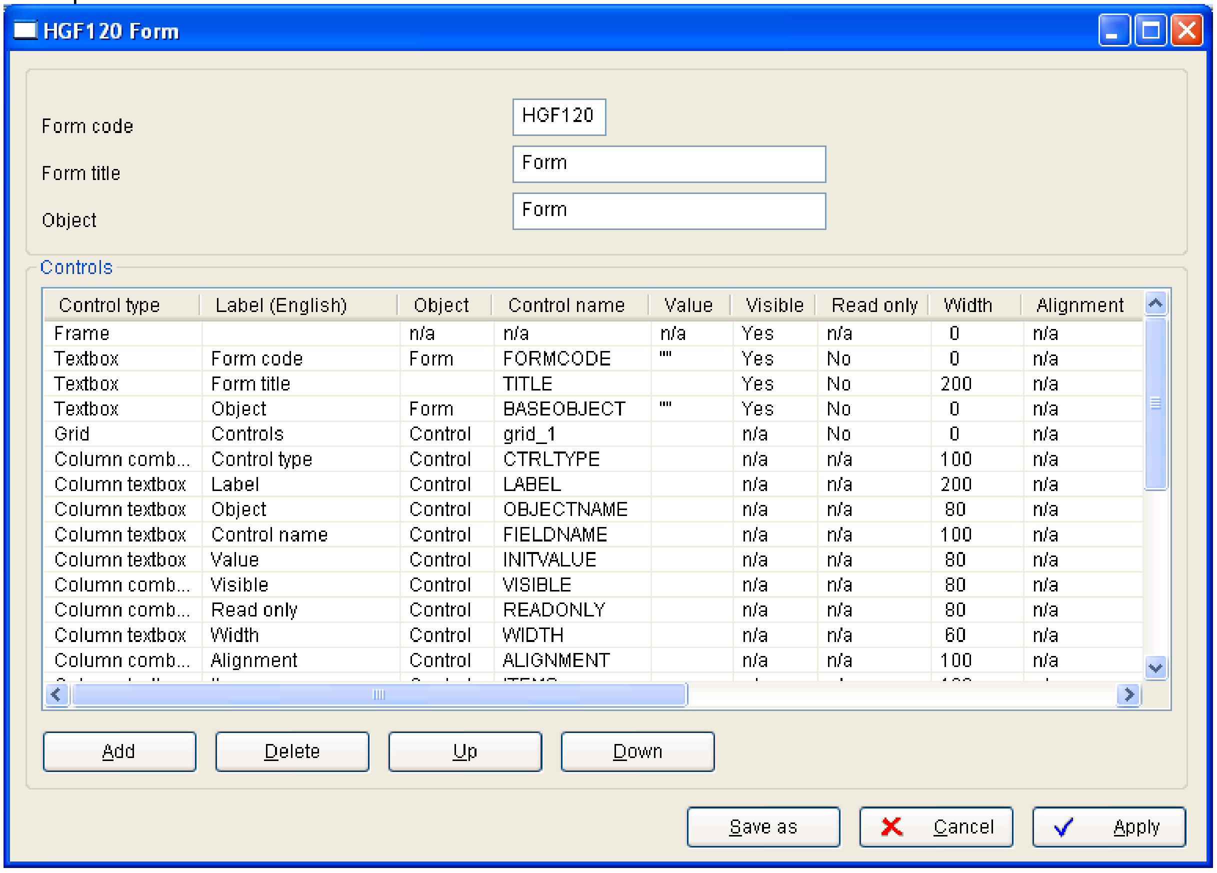
Why use HMGGEN ?

In the old days Clipper enabled us to develop client/server applications in a fast way. Later on the complexity of systems increased dramatically and we felt back on 3GL languages like Java and C/C++.
Nowadays the learning curve for new Java and .NET developers is enormously and companies choose for packages instead of internal development of systems. But is that a good idea?
If you can use a package out of the box as intended by the supplier the answer is yes, it does.
But if your business is specific, or a package fits for less than 80% percent, in house development can be a good alternative.

The generator enables you to focus on the real work that has to be done, the generator takes care of implementation details like presentation and retrieval and storage of objects. The form generator uses a sort of cascading stylesheet (for examples refer to the appendix), so all forms have the same look and feel. Also all objects and processes behave in the same predictable way.
The generator also hides implementation details of non functional requirements for usability, reliability (including confidentiality, integrity and availability), performance and supportability.
In this way you can prototype and elaborate the prototype “under architecture” from scratch.

HMGGEN has been generated *using* HMGEN, so you might examine HMGGEN to learn the concepts behind the generator. When you are familiar with these concepts, feel free to tailor the generator to your own specific needs.

Example:



HGF120 Form

Form code	HGF120
Form title	Form
Object	Form

Controls

Control type	Label (English)	Object	Control name	Value	Visible	Read only	Width	Alignment
Frame		n/a	n/a	n/a	Yes	n/a	0	n/a
Textbox	Form code	Form	FORMCODE	""	Yes	No	0	n/a
Textbox	Form title		TITLE		Yes	No	200	n/a
Textbox	Object	Form	BASEOBJECT	""	Yes	No	0	n/a
Grid	Controls	Control	grid_1		n/a	No	0	n/a
Column comb...	Control type	Control	CTRLTYPE		n/a	n/a	100	n/a
Column textbox	Label	Control	LABEL		n/a	n/a	200	n/a
Column textbox	Object	Control	OBJECTNAME		n/a	n/a	80	n/a
Column textbox	Control name	Control	FIELDNAME		n/a	n/a	100	n/a
Column textbox	Value	Control	INITVALUE		n/a	n/a	80	n/a
Column comb...	Visible	Control	VISIBLE		n/a	n/a	80	n/a
Column comb...	Read only	Control	READONLY		n/a	n/a	80	n/a
Column textbox	Width	Control	WIDTH		n/a	n/a	60	n/a
Column comb...	Alignment	Control	ALIGNMENT		n/a	n/a	100	n/a

This form for defining a form has been generated with HMGGEN. Under “Controls” you see the specifications of the form controls labeled “Form code”, “Form title” and “Object”. They specify the controls you see in the upper part of form. Under ”Controls” you also see the column controls “Control type”, “Label”, “Object, “Control name” and so on. They specify the columns you see in the grid.
Of course there are a lot of conventions and semantics. Many of them are intuitive, but some are not, refer to the appendix.

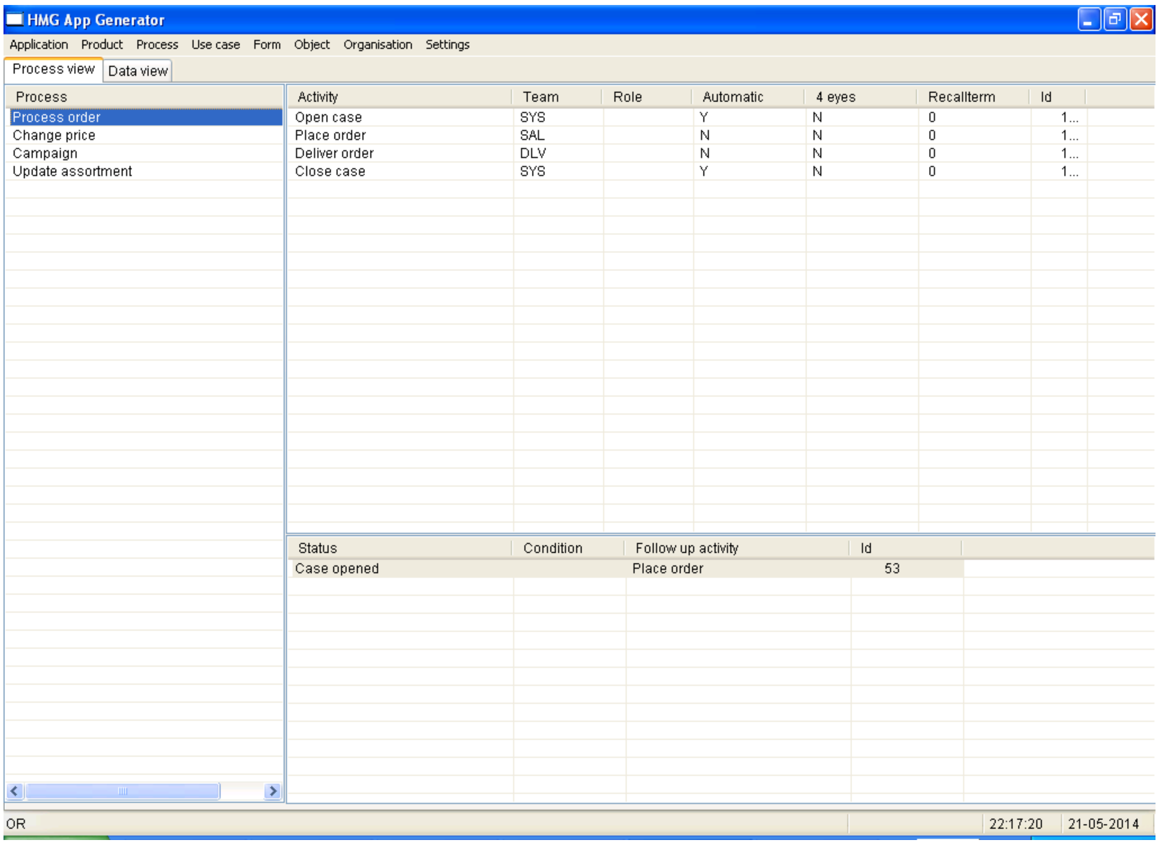
HMGGEN contains a rules engine and a workflow engine. You can run and examine the demo app Order to get familiar with these engines.

HMGGEN has intentionally been made for *prototyping* under architecture. You can set up an application in a fast way (against 3 hours per function point). The chosen architecture has been based on:
- workflow support
- 3 tier model
- fulfilment of architectural principles (e.g. separation of concerns like isolation of business rules)
- fulfilment of non functional requirements

I placed the code on the web cause HMGGEN is just a starting point. The chosen architecture makes it possible to replace the standard dbf files by for instance a MySql database, or to choose for another presentation channel, for instance a web client or an android mobile telephone. So I hope in the future, with the help of the community, HMGGEN or its successor can be evolve to a model driven generator for mission critical business applications.

Brief description HMGGEN

The menu shows the following choices : Application, Product, Process, Use case, Form, Object, Organisation and Settings.
The GUI has been based on the concept “selection before action”, so you start with the selection of an app. Open HMGGEN with Application>Open. The statusbar at the bottom shows the current selected app.
The main form has two tabs with two views, a process view and a data view.
The process view shows the app’s workflow processes at the left side and their activities and follow up activities on the right side.


HMG App Generator
Application Product Process Use case Form Object Organisation Settings
Process view Data view

Process
Process order
Change price
Campaign
Update assortment

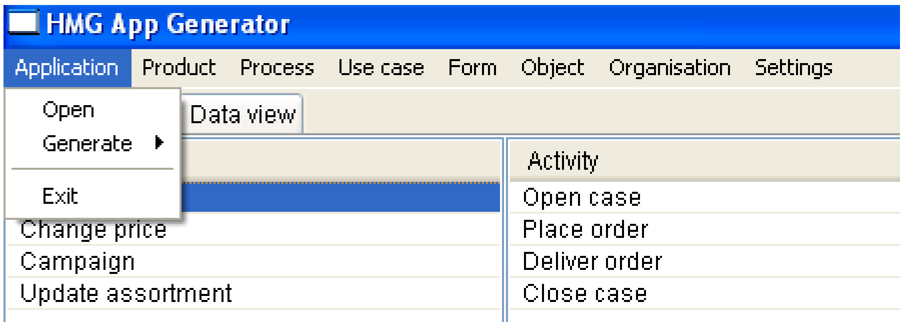
Activity	Team	Role	Automatic	4 eyes	Recallterm	Id
Open case	SYS		Y	N	0	1...
Place order	SAL		N	N	0	1...
Deliver order	DLV		N	N	0	1...
Close case	SYS		Y	N	0	1...

Status	Condition	Follow up activity	Id
Case opened		Place order	53

OR 22:17:20 21-05-2014

The data view shows the objects, in fact the classes, on the left side and their attributes and rules on the right side.

From a functional perspective you start with product configuration and processes. Beware that not all processes need workflow support. From a data perspective you start with the business object model.



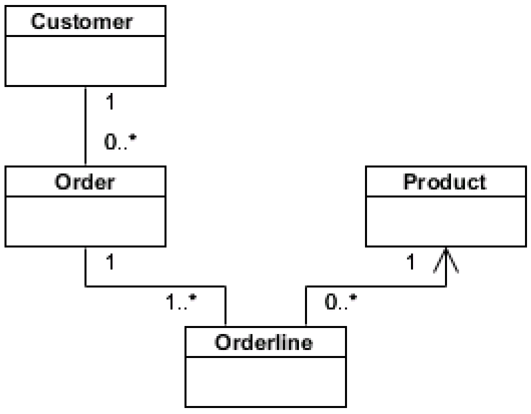
Using the menu you always have to work from right to left, since a form refers to objects, a use case to forms and a process activity to a use case. So you start with setting up the datamodel and the database, since an object must refer to a database table. Keep in mind that a change of the database causes a change of an object and a change of an object causes a change of a form and so on, so it is important to set up the database and the objectmodel properly.

When you need a dynamic choicelist in a form (a combobox control), you can make a sidestep to the menuchoice Product to define the choicelist.

When an app has a lot of dynamic rules (computations an inference rules) that change over time, you can specify them per class. The rules engine interprets those rules at run time. Refer to the specific user's guide.

Setting up the objectmodel

The objectmodel shows objects, in fact classes, and their associations. For the demo app the following objectmodel has been used.



Note that each association is bidirectional, except the association from Orderline to Product.
This means that you can navigate from a customer to his or hers order and vice versa from an order to a customer, but in this objectmodel you can not navigate from a product to orderlines.
However, on database level it is still possible to query all orderlines of a certain product.

Each object is mapped on a database table. A convention is that the tablename is a mnemonic of 4 uppercases. So Order can be mapped on ORDR.DBF. Using the standard HMG IDE the file looks like this:

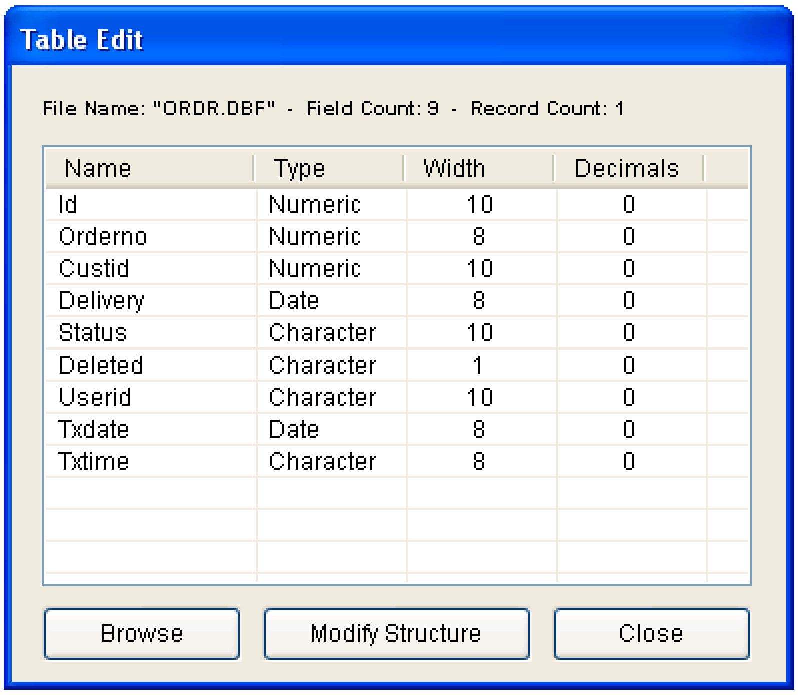


Table Edit

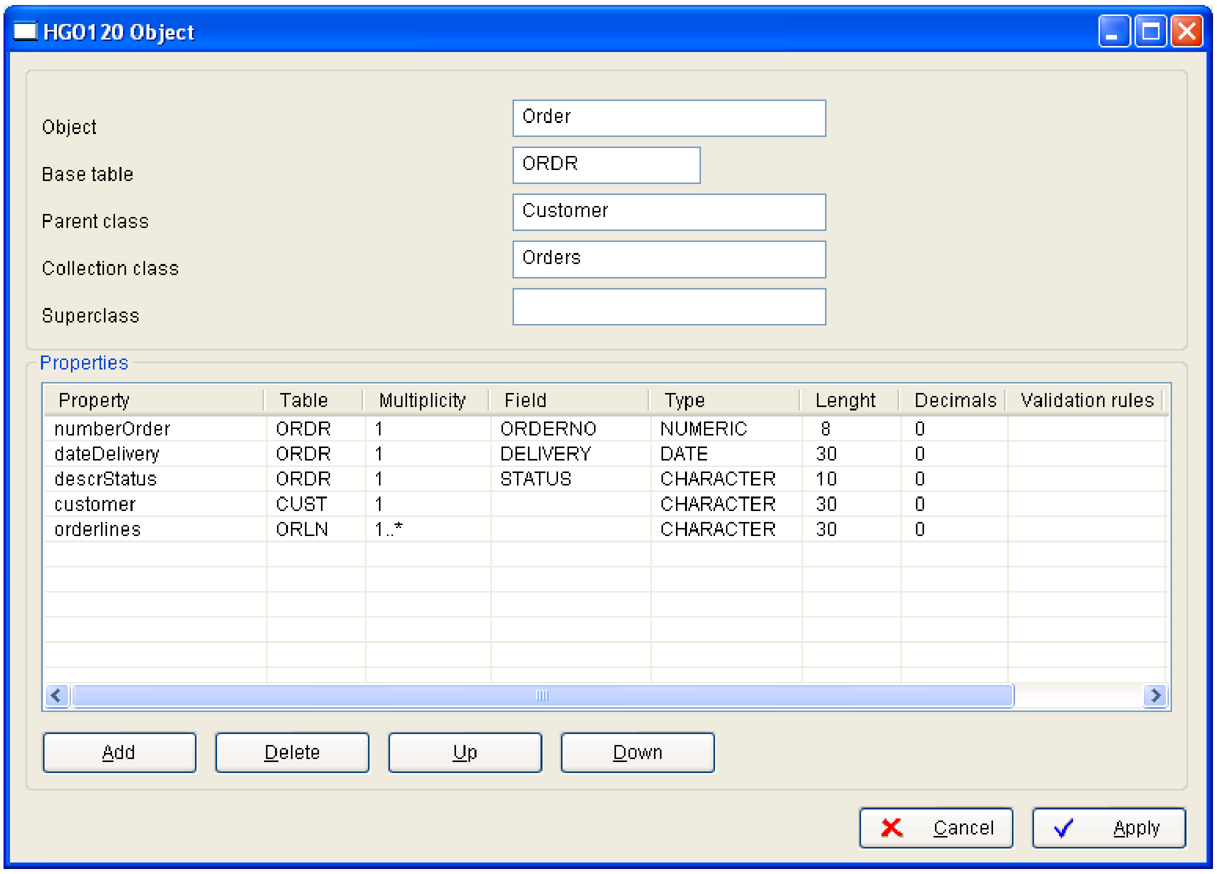
File Name: "ORDR.DBF" - Field Count: 9 - Record Count: 1

Name	Type	Width	Decimals
Id	Numeric	10	0
Orderno	Numeric	8	0
Custid	Numeric	10	0
Delivery	Date	8	0
Status	Character	10	0
Deleted	Character	1	0
Userid	Character	10	0
Txdate	Date	8	0
Txtime	Character	8	0

Note that there are some conventions. The fields Id, Deleted, Userid, Txdate and Txtime are standard mandatory fields, used by hmggen. And the name of a foreign key like Custid must be composed of the table name (CUST) and the suffix id.

Setting up an object in HMGGEN

The next step is setting up the objects in hmggen. The object Order looks like this:



Object	Order
Base table	ORDR
Parent class	Customer
Collection class	Orders
Superclass	

Properties

Property	Table	Multiplicity	Field	Type	Lenght	Decimals	Validation rules
numberOrder	ORDR	1	ORDERNO	NUMERIC	8	0	
dateDelivery	ORDR	1	DELIVERY	DATE	30	0	
descrStatus	ORDR	1	STATUS	CHARACTER	10	0	
customer	CUST	1		CHARACTER	30	0	
orderlines	ORLN	1..*		CHARACTER	30	0	

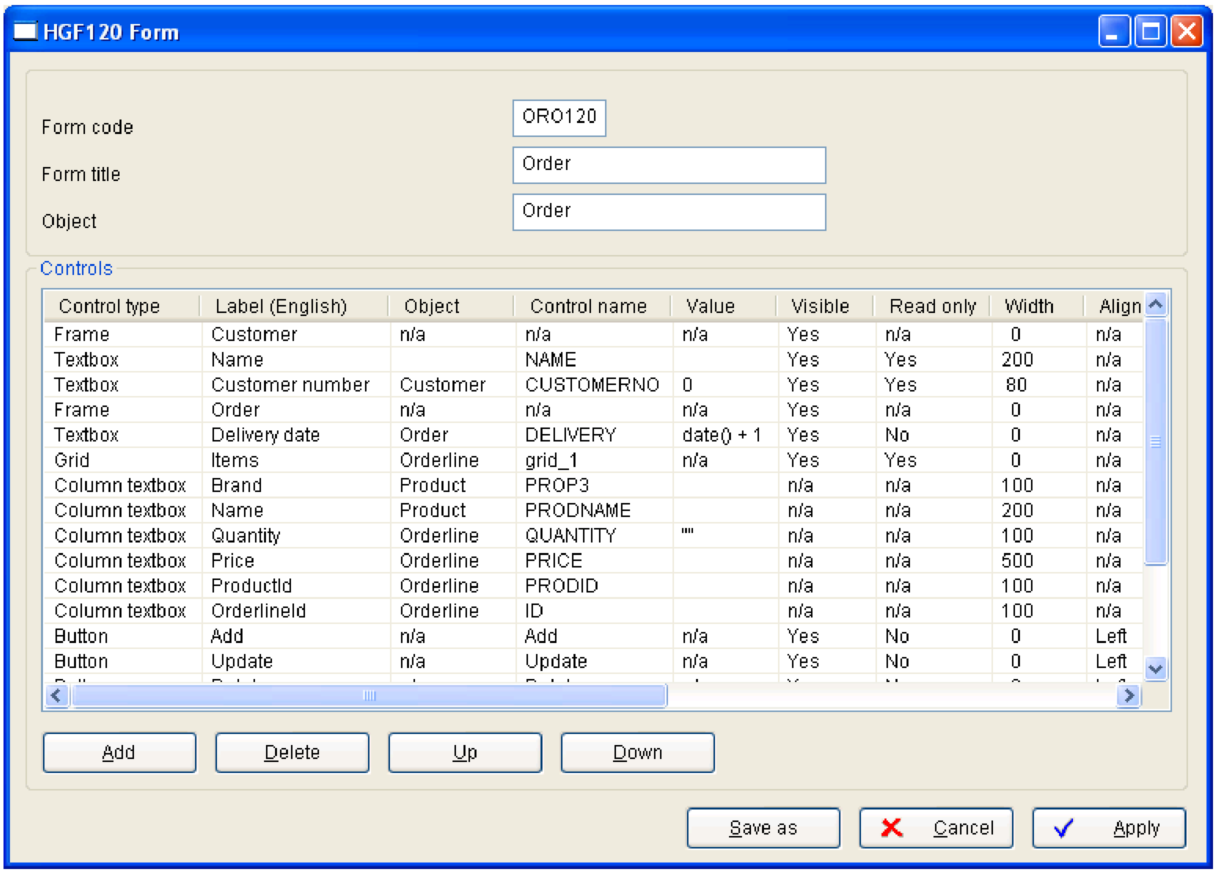
You see that Order has been mapped on the table ORDR and Order has a parent named Customer. You also have to specify the collection class for orders that represents the order table ORDR. The name of a collection class is the plural form of the name of the object.
Order has no superclass and doesn't inherit anything, so you can leave this field empty.

The customer also has been added as a property, associated with the parent table CUST with multiplicity 1.
An order has 1..* orderlines. This association has been added as a property associated with the table ORLN with multiplicity 1..*.

Note that the properties for Id, Deleted, Userid, Txdate and Txtime are missing. You don't have to specify them, hmggen takes care for that.

Setting up a form in HMGGEN

The next step is setting up a form in hmggen. The Order form looks like this:



Form code	ORO120
Form title	Order
Object	Order

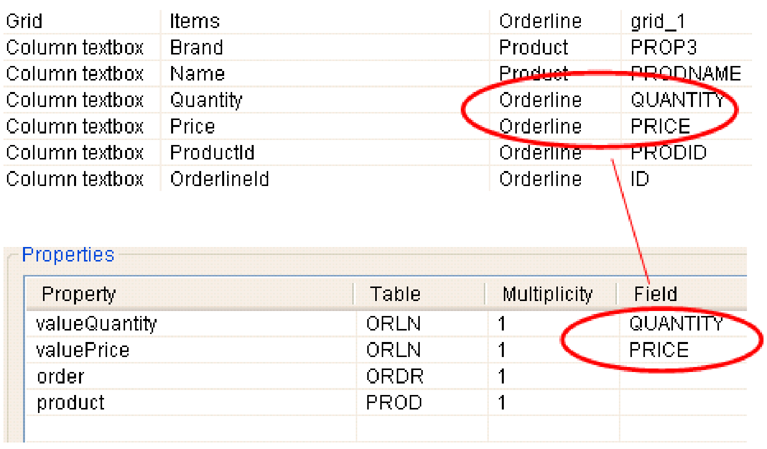
Controls

Control type	Label (English)	Object	Control name	Value	Visible	Read only	Width	Align
Frame	Customer	n/a	n/a	n/a	Yes	n/a	0	n/a
Textbox	Name		NAME		Yes	Yes	200	n/a
Textbox	Customer number	Customer	CUSTOMERNO	0	Yes	Yes	80	n/a
Frame	Order	n/a	n/a	n/a	Yes	n/a	0	n/a
Textbox	Delivery date	Order	DELIVERY	date() + 1	Yes	No	0	n/a
Grid	Items	Orderline	grid_1	n/a	Yes	Yes	0	n/a
Column textbox	Brand	Product	PROP3		n/a	n/a	100	n/a
Column textbox	Name	Product	PRODNAME		n/a	n/a	200	n/a
Column textbox	Quantity	Orderline	QUANTITY	""	n/a	n/a	100	n/a
Column textbox	Price	Orderline	PRICE		n/a	n/a	500	n/a
Column textbox	ProductId	Orderline	PRODID		n/a	n/a	100	n/a
Column textbox	OrderlineId	Orderline	ID		n/a	n/a	100	n/a
Button	Add	n/a	Add	n/a	Yes	No	0	Left
Button	Update	n/a	Update	n/a	Yes	No	0	Left

The textboxes Name and Customer number are placed in a frame, just like Form code, Form title and Object. The textbox for Delivery date is placed in a frame as well.
The orderlines of the order are places in a framed grid, just like thet headers “Control type”, “Label”, “Object”, “Control name” and so on. Since the grid buttons Add, Update and Delete have been aligned “left” they are placed within the frame of the grid. Buttons at the bottom outside any frame must have alignment “Bottom left” or “Bottom right”.

The order controls have been mapped on Order and the repetitive orderline controls on Orderline. A convention is that for an object bound control the technical control name must correspond with the name of the database field.

Example:



Grid	Items	Orderline	grid_1
Column textbox	Brand	Product	PROP3
Column textbox	Name	Product	PRODNAME
Column textbox	Quantity	Orderline	QUANTITY
Column textbox	Price	Orderline	PRICE
Column textbox	ProductId	Orderline	PRODID
Column textbox	OrderlineId	Orderline	ID

Properties

Property	Table	Multiplicity	Field
valueQuantity	ORLN	1	QUANTITY
valuePrice	ORLN	1	PRICE
order	ORDR	1	
product	PROD	1	

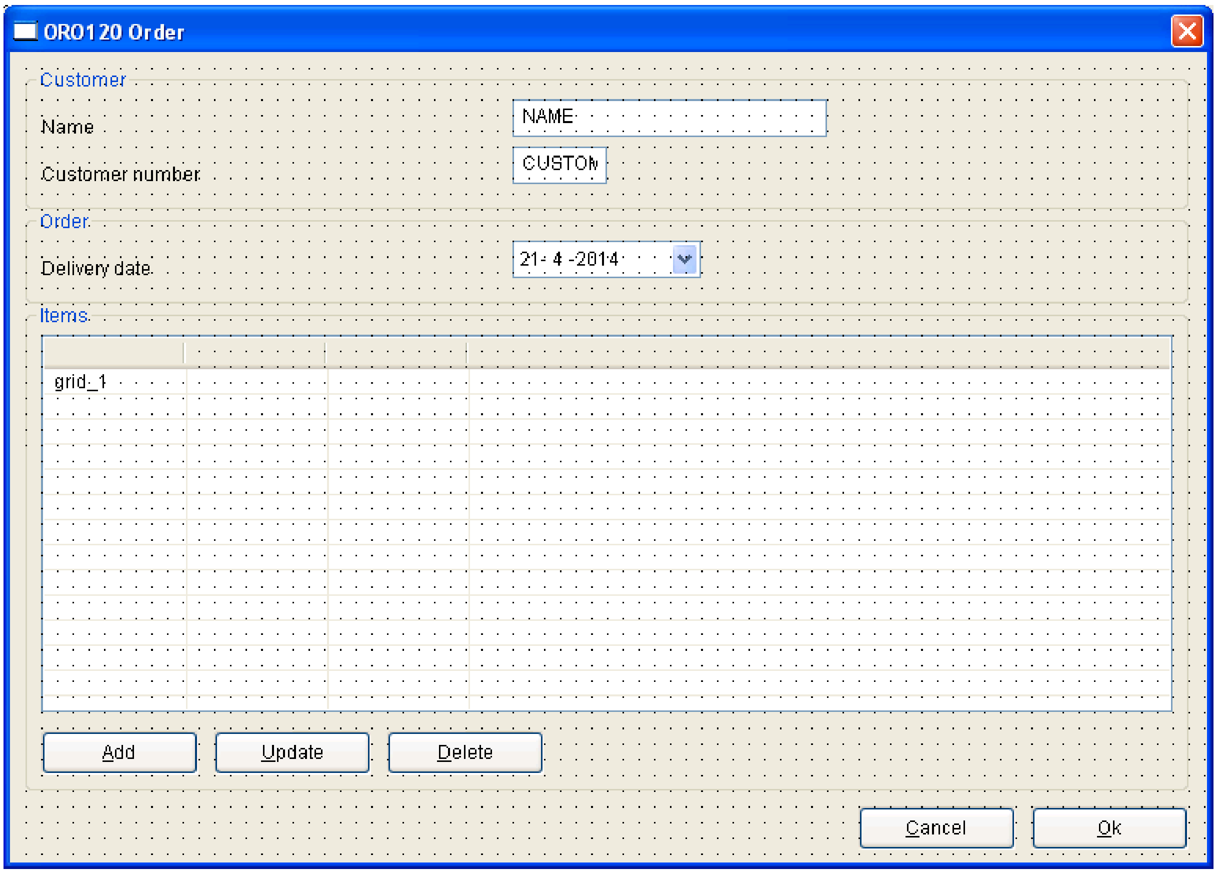
Note that the Name of the customer has not been mapped on the object Customer. This is because the NAME is composed from FIRSTNAME and LASTNAME.

You might give a property a default initial value. The value may be the returnvalue of a function as well, like for instance date() + 1.
When you use a dynamic combobox, the standard initial value can be overruled here.

Generation of code in HMGGEN

You can generate code for forms and objects. Since forms depend on objects, always generate code for objects first.
When you generate code for the object Order, hmggen generates the file cordr.prg.
When you generate code for the form Order, hmggen generates the files oro120.fmg and oro120.prg.

If you open the form oro120.fmg with HMG IDE, the form design looks like this:


ORO120 Order

Customer

Name NAME

Customer number CUSTOM

Order

Delivery date 21- 4 -2014

Items

grid_1

Add Update Delete

Cancel Ok

You may customize this form as you wish. If you do so, place some comments in the file oro120.prg, because after each time you regenerate this form you have to do the changes again.
If you have structural improvements, feel free to adjust the generator (file hga210.prg) and inform the forum about it.

A fragment of cordr.prg:

```
#include <hmg.ch>
#include "hbclass.ch"
/* BEGIN */
/* END */

class Orders
      Method Init() constructor
```

```
       Method Find(custid, orders, refdat)
       Method Get(id, order, requestor, refdat)
       /* BEGIN */
       Method GetLatestSeqno(txdate)
       /* END */
endclass

…

// put here your udf's

/* BEGIN */
Method GetLatestSeqno(txdate) class Orders
       local old := select(), nNumber := 0
       set deleted off
       select 0
       use ordr index ordrb
       seek dtos(txdate)
       do while .not. eof() .and. ordr->DELIVERY = txdate
              nNumber++
              skip
       enddo
       use
       select(old)
       set deleted on
return(nNumber)
/* END */
```

This code fragment shows that you can add your own code, like the method GetLatestSeqno(). Each time you regenerate a form or an object, your own code will be merged into the new code. When a code fragment can't be merged anymore, for instance because a button has been removed, that code fragment will be inserted between the last markers at the end of the file.

Don't insert code outside the BEGIN and END markers!

Next a fragment from the generated file oro120.prg:

```
#include "hmg.ch"
#include "apprefdata.ch"

declare window ORO120
/* BEGIN */
declare window OR0000
declare window ORO130
declare window ORO140
/* END */

function ORO120_OnInit()
    local n
    /* BEGIN */
    local aItem
    if OR0000.USECASE.Value = "Place order"
       ORO120.STARTDATE.Value := dtoc(date())
       ORO120.STARTTIME.Value := time()
    endif
    /* END */
    if order:id = 0
       ORO120.CUSTOMERNO.value := 0
       ORO120.DELIVERY.value := date() + 1
    else
       ORO120.CUSTOMERNO.value := customer:numberCustomer
       ORO120.DELIVERY.value := order:dateDelivery
       for n := 1 to len(order:orderlines)
```

```
        ORO120.grid_1.AddItem({ ;
            ComboLookupDescr(“BRAND”, order:orderlines[n]:product:codeBrand), ;
            order:orderlines[n]:product:nameProduct, ;
            str(order:orderlines[n]:valueQuantity), ;
            order:orderlines[n]:valuePrice, ;
            order:orderlines[n]:product:id, ;
            order:orderlines[n]:id ;
        })
        next
    endif
    /* BEGIN */
    ORO120.NAME.Value := customer:nameFirst + " " + customer:nameLast
    ORO120_grid_1_Refresh()
    ORO120.DELIVERY.SetFocus
    /* END */
return(nil)

function ORO120_OnUpdate(order)
    /* BEGIN */
    /* END */
    order:dateDelivery := ORO120.DELIVERY.value
    /* BEGIN */
    /* END */
return(nil)
```

There are some interesting phrases in this code fragment.

sub fragment 1:

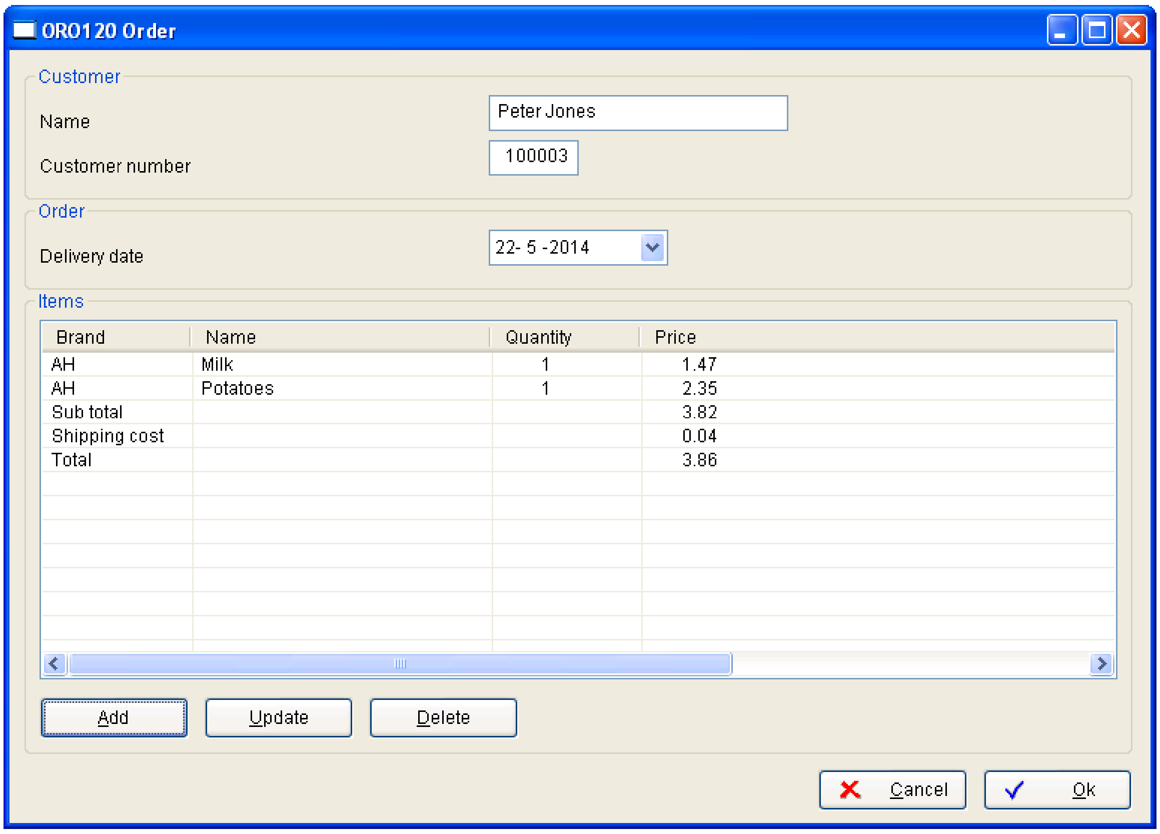
```
    /* BEGIN */
    local aItem
    if OR0000.USECASE.Value = "Place order"
       ORO120.STARTDATE.Value := dtoc(date())
       ORO120.STARTTIME.Value := time()
    endif
    /* END */
```

Form OR0000 has a hidden control named USE CASE, form ORO120 has two hidden controls for the STARTDATE and STARTTIME of the use case. When you use the workflow engine (it’s an option), the startdate and time is stored in these variables. When the use case is finished, the use case informs the workflow engine about the startdate and time and the enddate and time of the use case.

sub fragment 2:

```
        str(order:orderlines[n]:valueQuantity), ;
```

In fact valueQuantity is a numeric property, but with an empty string as default value. In this way the value is presented as a string. This is done to suppress the value 0 in total lines as shown in the example below.



Customer

Name: Peter Jones

Customer number: 100003

Order

Delivery date: 22- 5 -2014

Items

Brand	Name	Quantity	Price
AH	Milk	1	1.47
AH	Potatoes	1	2.35
Sub total			3.82
Shipping cost			0.04
Total			3.86

sub fragment 2:

```
        ComboLookupDescr(”BRAND”, order:orderlines[n]:product:codeBrand), ;
```

The description of codeBrand is fetched from a dynamic choicelist named BRAND.
For a simple static combobox you can use the column “Items” to define the fixed choices, as you do using the standard HMG IDE, for example {“NL”,”UK”}. The next chapter illustrates how to work with dynamic choicelists like BRAND.

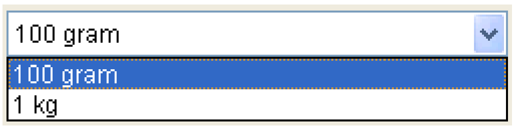
Product choices

Each product has product characteristics. The values of these characteristics determine a concrete and unique product .
For simple products like food and non-food supermarket products all characteristics already have a value, you just take them off the shelf. Food products have characteristics like product type, brand, product ingredients, packaging unit, price per packaging unit and possibly a best before date.

Sometimes a supermarket offers a travel insurance as well.
For this kind of financial products the values of the product characteristics are determined at buy time. A travel insurance has characteristics like departure date, duration, family composition, destination, and additional coverage (money, medical expenses, cancellation, accidents and so on)

To keep the administration cost low this kind of products are standardized by restricting the possible values of product characteristics.
Usage of predefined choice lists with default values restrict the freedom of choice,

Example:
Packaging unit



In this example the choice values depend on the chosen type of product. Possibly that chosen type of product is a meat product.

It could be that a customer has a specific product requirement. In such a case a commercial decision can be a solution.
When a product choice value has been marked as “commercial decision” only an authorized user can see and use that choice value.
When needed a product characteristic can only be made visible for an authorized user like a sr. sales representative.

Authentication and authorization

Hmggen offers a simple but adequate authentication and authorization mechanism. Under “Organisation” you can add a new user. A user has a userid of 3 characters and is assigned to a usergroup (or team). When a user belongs to more usergroups, the user needs an account (= userid) for each usergroup.

When a user makes him/herself known to the system, by means of the credentials userid and password, his/hers userprofile is retrieved. A user acts on a certain authorization level, a number of two digits.

Level	Permission
10-19	Read permission
20-29	Update permission
30-99	System permission

The first digit indicates vertical function separation. When a user has update permission, the user has read permission implicitly. When a user has system permission, the user has update and read permission as well. The system self has permission 99.
When the second digit is greater than 0, the second digit is used for horizontal function separation. So when a user has role 22, that user has only permission for use cases assigned to roles 10-19 or 20, but not for uses cases assigned to role 21 or 23-29.

<u>Roles</u>
A user has a (one) role. When a user has more roles the user needs as much accounts as roles.
For the demo app a user can have the following roles:

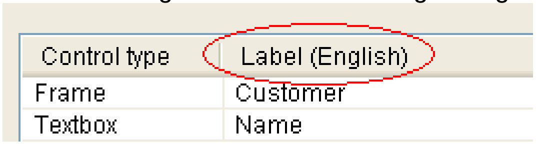
Role	Level
sales representative	21
sr. sales representative	22
Storekeeper	26
Application manager	30
System	99

How does it work?
The requested role for the use cases Place order and Adjust order is “sales representative” (role 21), only users with this role or with a role within a higher range (30-99) can place and adjust an order. The requested role for the use case Deliver order is “storekeeper”, so only users with the role “storekeeper” (role 26) or with a role in a higher range (30-99) can do this.

The requested role for the use case Change price is “sr. sales representative” (level 22), only users with this role or with a role in a higher range (30-99) can change the price of a product. Maybe the sr. sales representative must be able to place orders as well. For this purpose you can assign role 20 to use case Place order. But then the storekeeper with role 26 can place an order as well. Alternatively you can make an extra account for the sr. sales representative with role 21.

Multilingual

Under “Settings” you can change the language. The default is English. When you chose for you own native language, keep in mind that each time you start hmggen you have to set the language to your native language again.
When the language setting is English, all form titles, form control labels and choicelists must be defined in English. When the setting is English, you see it in the column headers.



Control type	Label (English)
Frame	Customer
Textbox	Name

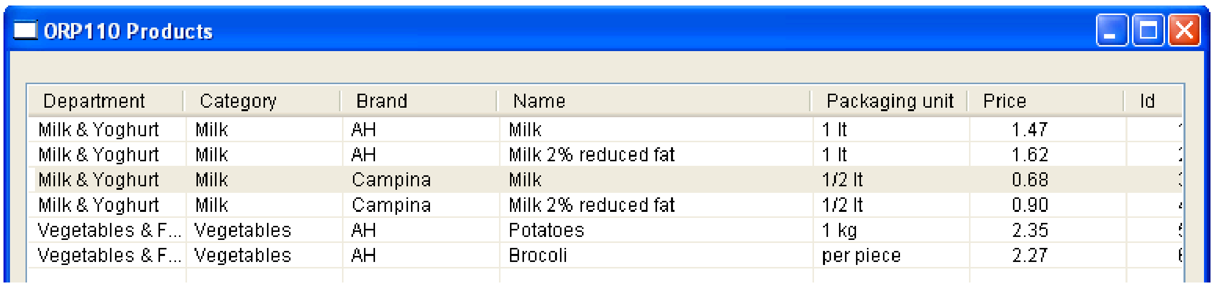
Faultmessages, in English or native language, must be maintained outside hmggen. For this you have to manipulate the table MSSG directly.

Keeping track of historical data

The generator makes use of a general tracking mechanism for historical data. Each time a record is updated[1] the old values are saved in the table CLOG. Only the values of the properties that have changed are saved. The database always represents the actual data, all history is kept separately.

When a transaction has a startdate in the future, the transaction must wait until that startdate.
If you use the workflow engine, the transaction data is kept in xml format within the case dossier.

So when you change the price of Campina Milk ½ lt from € 0,69 to € 0,68 with startdate 22-05-2014, what happens?



Department	Category	Brand	Name	Packaging unit	Price	Id
Milk & Yoghurt	Milk	AH	Milk	1 lt	1.47	
Milk & Yoghurt	Milk	AH	Milk 2% reduced fat	1 lt	1.62	
Milk & Yoghurt	Milk	Campina	Milk	1/2 lt	0.68	
Milk & Yoghurt	Milk	Campina	Milk 2% reduced fat	1/2 lt	0.90	
Vegetables & F...	Vegetables	AH	Potatoes	1 kg	2.35	
Vegetables & F...	Vegetables	AH	Brocoli	per piece	2.27	

The change is stored in the memofield DOSSIER of the table CASE in xml format.
The xml looks like this:

```
<Pricechange>
    <producttypes>
       <Producttype>
          <price>0.68</price>
          <productcode>AH000054</productcode>
       </Producttype>
    </producttypes>
    <startdate>22-05-2014</startdate>
</Pricechange>
```

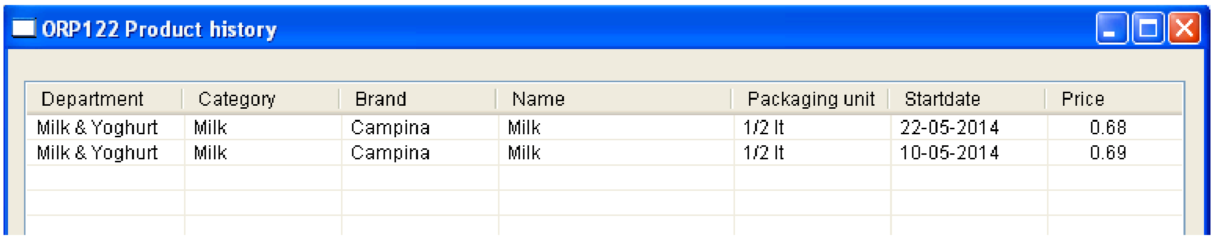
When the price change has been approved and applied by a colleague, according to the 4 eyes principle, the database is updated and just before the update the old price is logged in CLOG as follows:

ID	TXDATE	TXTIME	ENDDATE	ENDTIME	CLASS	FIELD	REFID	VALUE	USERID

[1] A deletion is also considered as an update.

40	22-05-2014	14:23:11	22-05-2014	00:00:00	PROD	PRICE	3	0.69	SA3

If you click on “History” you see:


ORP122 Product history

Department	Category	Brand	Name	Packaging unit	Startdate	Price
Milk & Yoghurt	Milk	Campina	Milk	1/2 lt	22-05-2014	0.68
Milk & Yoghurt	Milk	Campina	Milk	1/2 lt	10-05-2014	0.69

In many situations the state of objects must be frozen at the end of a period. The period can be a day, a month or longer. In such cases for each new period a new record is added. Since the logging mechanism only logs updates and deletions (a deletion is also an update) in such situations no logging records have to be added in CLOG.

Under “Settings” you can disable logging. When logging is set to “No” during code generation no logging is invoked.

Time travelling

Since hmggen keeps track of historical data, you might travel back in time If you add the parameter “refdate” to a Find or Get message, hmggen will reconstruct the state of the data on that date.

The mechanism also offers possibilities to handle transactions with retrospective effect.
For instance a salary change can have a startdate in the past. When this happens
- the effected data of the employee must be rolled backed until that startdate,
- the salary change must be applied, and
- the transactions[2] from the startdate must be applied again.

Suppose pension premiums must be recalculated for this employee. What do you have to do?
You have to roll back the data needed for the calculation of the pension premium as well, because pension rates and conditions can change over time. And then you have to replay all premium collections since the startdate. And the previous premium collections must be made undone first.

So you see this is complicated stuff and it’s far beyond the goal of this document to go in depth here, but the hmggen architecture makes it possible to fulfil this kind of requirements as well.

[2] The transactions that cause the state transitions of the rolled back data must be applied again. Note that for transactions that cause outgoing data that data must be made undone first.

APPENDIX: Installation HMGGEN

Prerequisites:
- Windows machine (Windows XP or higher)
- installation of hmg (the latest, at the moment 3.1.5)

Make a directory hmggen under c:\hmg.1.3.5\samples
Unzip the files under c:\hmg.1.3.5\samples\hmggen
start ide (the ansi version !)
open project hmggen
run
open project ordersystem
run

APPENDIX: HMGGEN Conventions

Product
- The collection of values of properties of a product determine a concrete product(object)
- A product property can be specified as a choicelist, presented as a combobox
- The items of a dynamic combobox can be made dependent of another choice

Process
- A process consists of a list of predefines activities
- A use case can be linked to a predefined activity
- A use case contains a number of forms

Form
- The code for a form consists of the app abbreviation (2 characters), a menu letter and a sequence number (3 digits)
- The insert order of a control determines the tab order. The order can be changed with Up and Down buttons.
- Data entry controls are placed in the preceding frame
- A Grid is standard placed in a frame
- Grid column controls are placed in the preceding grid control
- The last grid column control must be the id of the object that is shown
- Grid buttons (indicated with alignment Left) are placed in the preceding grid frame
- Other buttons (indicated with alignment Bottom left or Bottom right) are placed at the bottom of the form
- Visible bottom aligned buttons overlap invisible bottom aligned buttons
- The fieldname of an object bound control must correspond with the database fieldname where the object property is stored
- When the language has been set to native language, the form title, control labels and choicelists must be specified in the native language, otherwise in English

Object
- The base table of the object must refer to the database file where the object is stored.
- The name of an object, in fact a class, must start with an uppercase, e.g. Order
- The name of a property must start with a lowercase, e.g. dateDelivery
- The name of a 1-to-1 link property is the name of the target object, e.g. customer
- The name of a 1-to-many link property is de the collection name of the target objects, e.g. orderlines
- A property must refer to a database fieldname
- A database fieldname is indicated by uppercases
- The object generator always add the following properties to an object:
  - id
  - deleted
  - userid
  - txdate
  - txtime

Relationship between prefixes for property names and form controls:

Prefix	Meaning	Control	
code	code	combobox	items are retrieved from choicelist
abbrev	Abbreviation	tesxbox	type character
name	name	textbox	type character
descr	description	textbox	type character
indic	indication	combobox	“Yes”, “No”
date	date	datepicker	type date
number	number	textbox	type numeric
amount	amount	textbox	type numeric
value	value	textbox	type numeric
perc	percentage	textbox	type numeric

id	identification	textbox	type numeric
Others		textbox	type character

Database table
- The name of the base table must be a mnemonic of 4 uppercases, e.g. ORDR
- A database fieldname is limited to 10 characters
- A database fieldname is specified in uppercases
- The primary index must have the same name as the table name, with the extension .ntx
- The first secondary index, with suffix A, e.g. ORDRA, must refer to the main parent, e.g. CUSTID
- Each extra secondary index must have the suffix “B”, “C” or “D”
- The database table must always include the following fields:
  - ID
  - DELETED
  - USERID
  - TXDATE
  - TXTIME

O/R mapping
- Each object is mapped on one database table
- When a superclass (or supertype or generalization) has subtypes (or specializations), the supertypes and the subtypes are mapped on one table

Rules
- Hmggen distinguishes four kinds of business rules
  1. data constraints
  2. process rules
  3. compuations
  4. inference rules
- examples:
  data constraints:
  BR001, BR002 and so on, refer to form field validation of form Object
  process rules (workflow rules):
  If order has been submitted by the customer, forward the case to the delivery team
  computations:
  _amountOrderline := orderline.valueQuantity * orderline.product.valuePrice
  inference rules
  if customer.address.zipcode starts with 1000, _shippingcost := $ 5,00

Validations
- Hmggen distinguishes four levels of validations (data constraints)
  1. content of a form field
  2. form fields in relation to each other
  3. key’s
  4. referential integrity
- 1 and 2 must be evaluated in the front-end AND in the back-end (the front end in my opinion is “tissueware”, throw away software)
- 3 and 4 must be evaluated in the back-end

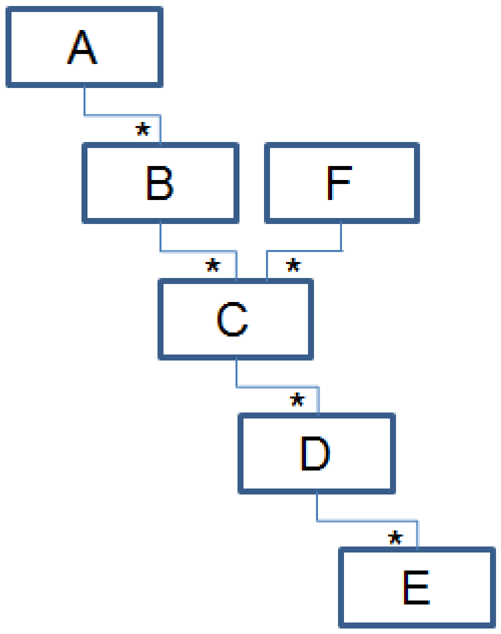
Methods

Find
- A find request returns a collection of objects
- A standard find request returns the child objects of the main parent
  example: Orders:Find(custid, {})
- When the request contains a reference date, the method returns the state of the objects on that date (all transactions are rolled back until that date)
  example: Orders:Find(custid, {}, ctod(“01-01-2014”))

Get

- A get request returns the requested object including its parent(s) and its children (cascading)
- A parent is only retrieved whether the requestor of the request is not that parent
- The children are only retrieved when the requestor is not such a child object

  Examples



CollectionA():Get(id, A(),requestor=”a”) returns requested object a, its objects of B, C, D and E, plus for each object c parent f.
CollectionB():Get(id, B(),requestor=”b”) returns requested object b, its objects of C, D and E plus parent a and for each object c parent f.
CollectionB():Get(id, B(),requestor=”a”) returns requested object b, its objects of C, D and E and for each object c parent f, but not parent a.
CollectionC():Get(id, C(),requestor=”c”) returns requested object c, its objects of D and E plus parent b and its ultimate parent a and parent f.
CollectionC():Get(id, C(),requestor=”b”) returns requested object c, its objects of D and E plus parent f, but not parent b and its ultimate parent a.
CollectionC():Get(id, C(),requestor=”d”) returns requested object c, plus parent b and its ultimate parent a and parent f, but not objects of D and E.

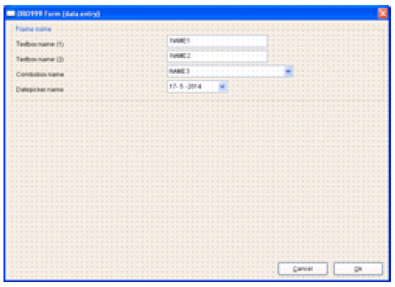
- When the request contains a reference date, the method returns the state of the objects on that date (all transactions are rolled back until that date)

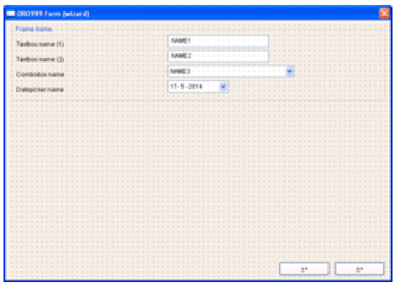
Save

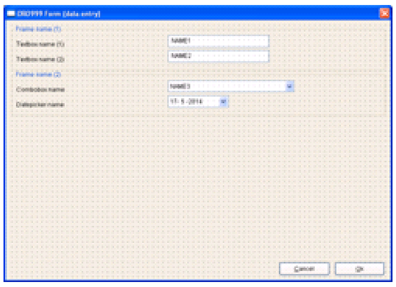
- A save request persists the state of the object into a database file
- When the corresponding database record has been updated or deleted by another user, the save request is not performed (optimistic locking)
- When one or more validations don’t succeed, the request is not performed and a list of broken rules is returned
- Just before an update the former state of an object property value is logged into a specific log file

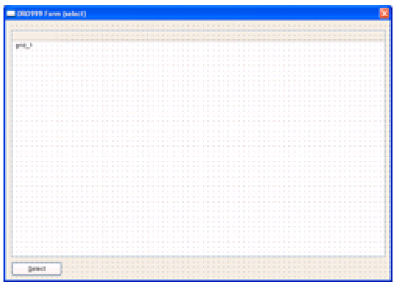
APPENDIX: Form styles

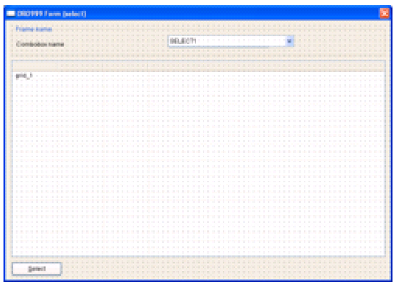
The following examples show the possible styles. Feel free to adjust styles or to add new styles, but be careful.

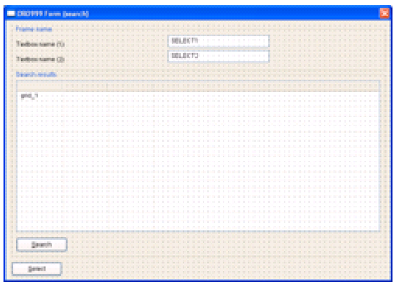

DBD999 Form (data entry)
Frame name
Textbox name (1)
NAME1
Textbox name (2)
NAME2
Combobox name
NAME3
Datepicker name
17- 5 -2014
Cancel
Ok

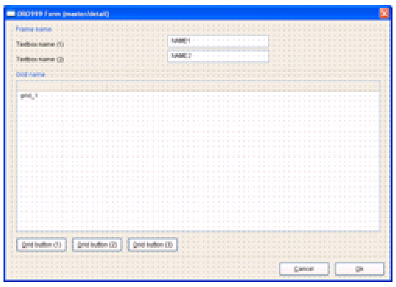

DBD999 Form (wizard)
Frame name
Textbox name (1)
NAME1
Textbox name (2)
NAME2
Combobox name
NAME3
Datepicker name
17- 5 -2014

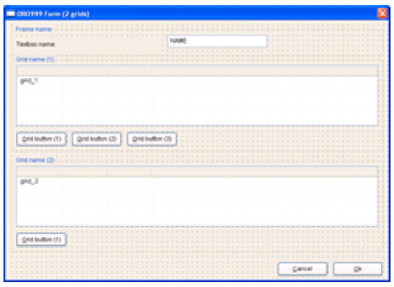

DBD999 Form (data entry)
Frame name (1)
Textbox name (1)
NAME1
Textbox name (2)
NAME2
Frame name (2)
Combobox name
NAME3
Datepicker name
17- 5 -2014
Cancel
Ok


DBD999 Form (select)
grid_1
Select


DBD999 Form (select)
Frame name
Combobox name
SELECT1
grid_1
Select


DBD999 Form (search)
Frame name
Textbox name (1)
SELECT1
Textbox name (2)
SELECT2
Search results
grid_1
Search
Select


DBD999 Form (master/detail)
Frame name
Textbox name (1)
NAME1
Textbox name (2)
NAME2
Grid name
grid_1
Grid button (1)
Grid button (2)
Grid button (3)
Cancel
Ok


DBD999 Form (2 grids)
Frame name
Textbox name
NAME
Grid name (1)
grid_1
Grid button (1)
Grid button (2)
Grid button (3)
Grid name (2)
grid_2
Grid button (1)
Cancel
Ok