By Marco Menardi mmenaz@mail.com
TECHNICAL INFORMATION SHEET - (DRAFT, LGPL license) - 2003-09-06

(Following Richard Stallman ideals about Free Software, I would like that this document, if useful, wil be improved by everyone that wants to do so. Feel free to use this document and, if you improve it, give me the enhanced version so I will merge the improvements so everyone will benefit).

RAD with IBO native components

Introduction
Well, Ibo native components (TIB_xxx) have a lot of useful properties that let you save a lot of time in developing your program, reduce errors, increase user interface consistency. These notes are based upon my experience, and are intended to show you one way for doing things. They don't pretend to teach "the best way", but you can consider them a starting point.

Part one: data entry and fields attributes

One of the most tedious and inconsistency prune problems in the client side of the application is when data entry of fields of the same type are managed differently, or some database requirements are not checked or imposed when the user enters data, like uppercase or date format. I've often had some data entry forms where name was automatically uppercased, and other where it was not, since I forgot to set the field editmask the correct way... Or dates formatted in different ways in different parts of the program.
In the standard VCL components, you can create Fields components for each field of your tables (persistent fields) and then set the attributes to each of them (like editmask, displayformat, etc.) that are used by TDBEdit or other databound components.
In IBO you don't have persistent fields, all is created at runtime, but you can set fields (columns) attributes in the query editor, that I find more convenient. But this is a tedious way of doing things like the VCL one! And you can easily forget to set some of them, or set them differently from query to query. And what about if you change your mind and, for instance, change a field from case insensitive to uppercase? You should search it on all your queries and change the CharCase field property attribute.
In addition, you have plenty of fields in your program, but their kinds are much less. I mean, you can have 50 fields for date storage, and they are all of date type, so the display format and the edit mask is the same. The "Zip Code" field is the same in your customer table or in your supplier one, in the contact table or in the Christmas greetings one. This is the main reason in Interbase we use DOMAINS. With domains, you can define a "kind" of field, and set a field to be of that kind. Oh, how useful would be at client level associate data entry and other field properties to domains... And that's exactly what IBO let's you do!

Centralization settings: IB_Connection
As you can see, TIB_Connection has a lot of properties related to fields: FieldsAlignment, FieldsCharCase, FieldsDisplayLabel, FieldsGridLabel, FieldsDisplayFormat, FieldsDisplayWidth, FieldsEditMask, FieldsReadOnly, FieldsTrimming, FieldsVisible. There are also ColumnAttributes and DefaultValues properties. They are the same of the TIB_Query component, and this is not casual.
In fact, if you define a field property here, every time that field is used by a query, the property, if not "overridden" (defined) at query level, is taken from the connection component!
So, if you enter in FieldsCharCase the value "ADDR_NAME=UPPER", each field of name "ADDR_NAME" in whatever query will behave like you had defined "Uppercase" property for it in the query.
This alone would be a big saving, since you could enter the various properties of all the fields of your database in the connection properties and have your program use their attributes consistently in every place. But IBO can do even better, much better!

We must look at the IB_Connecion FieldEntryTypes property. Let's concentrate upon fetDomainName true/false value of that property. If set to True, then IBO considers the various field related properties seen above as related to DOMAINS. So if we have defined a domain "PURCASE_DATE_DM" (I always postfix domains names with "_DM"), we could set, for instance:
FieldsDisplayFormat: PURCASE_DATE_DM=mm/dd/yyyy ddd
FieldsEditMask: PURCASE_DATE_DM=99/99/99
This way, ALL fields in our program that are defined in the database of domain "PURCASE_DATE_DM" will

have an entry mask like "01/07/02" and will be displayed like "01/07/2002 Friday". Is that magic? Consider that if you later want to change, for instance, the display format, all you have to do is change the related property in the IB_Connection.FieldsDisplayFormat property, and your program as a whole will be updated!
And if I have CustomerID of domain CustomerID_DM (varchar(6), uppercase), to have it always uppercase, and that can be searched incrementally even entering lowercase in the search edit, I can define:
FieldsCharCase: CustomerID_DM=UPPER
ColumnAttributes: CustomerID_DM=NOCASE=CustomerID;BLANKISNULL
This tells IBO that every field of domain CustomerID_DM must be entered as Uppercase, and that if you search upon a field of that domain, the search must be no case sensitive and performed upon the CustomerID columns of the query. Fabulous!
For booleans, I've two kinds: YESNO_DM and NOYES_DM, one having "Y" (yes) as default, the other "N" (no). So I've:
In database:
CREATE DOMAIN YESNO_DM AS CHAR (1) DEFAULT 'Y' NOT NULL CHECK (VALUE IN ('Y', 'N'));
CREATE DOMAIN NOYES_DM AS CHAR (1) DEFAULT 'N' NOT NULL CHECK (VALUE IN ('Y', 'N'));

In IB_Connection:

FieldsCharCase:
YESNO_DM=UPPER
NOYES_DM=UPPER

ColumnAttributes:
YESNO_DM=BOOLEAN=Y,N
NOYES_DM=BOOLEAN=Y,N

DefaultValues:
YESNO_DM=Y
NOYES_DM=N

That's all I need for booleans in my application!

Default values
Default values is a problematic issue. The database applies defaults only if a insert does not specify a field value, that is if it does not include it in the insert SQL string.
IBO, when building automatically the insert sql statement, uses all the fields of the related select, and if has no specific value for a field when inserting, it assigns "null" to it. The database considers null a value, so it writes into the table "null" and not the default value for that field.
If you want defaults values, you must have them defined at the client level. The use of then IB_Connection DefaultValues property is a good idea for fields that always use the default of their domain (like booleans in the above example), but this is not always the case, since it can be always redefined in the column declaration.
Also, having the same value duplicated at the database level and client level is not a good idea. If you one day will change the default in the database and not in the client, you will have that data entered in some ways have certain default (i.e. data imported from a file, where that field is not used in the "insert"), while other in other way (i.e. data entry forms) have a different one.
It would be great if the client could retrieve the defaults from the server and use them... and that's what IBO can do for you! You just have to set the query property "GetServerDefaults" to True and it works! Time ago this was often discouraged because the default values were retrieved from the server each time the query was prepared for an insert, with some performance penalty. I used them without noticing any problem, but with slow connection sure there is a lot of more data travelling. Recent versions have optimised it, so now defaults are retrieved for all fields in the database as soon as the first query needs one of them, and then these values are used without further database interrogation.

Searching with "starting with"
You know that IBO native components have a very useful search state, where you can enter values in the

various fields in your data entry form, and then click the "first" navigation bar button to have a selected subset of the table that meets the required values. But 99% of the times, I don't know the exact values to match! For instance, I recall how the last name of a customer begins, and want to have a list of them to find the right one. What I want is a search that acts as "starting with" and not "=". To do so, you have to set "No trailing" to the field in the columns attributes of each query, or enter ">" in front of the field value (i.e. ">WHARTON"). So tedious! And I want it for almost every field of my tables. To do this, I set IB_Connection property "DefaultNoTrailing" to true, and that's all! In the property help you can read: "This property can be used to default all string/text fields to being searched as though field=NOTRAILING was specified in the column attributes. This default can be overridden in the column attributes by the use of field=YESTRAILING.". Fabulous again! So if we want exact match for some particular field, we can set YESTRAILING for it's domain in the connection ColumnAttributes property. Great.

Avoiding unnecessary data fetch
If you came from the Paradox/DBIII word, you think in terms of "tables" rather than "datasets". This means that if you want to work upon a certain database table, you "open" the table, and then you start working navigating in it (usually with locate).
In a Client/Server approach, this is a suicide, since the "open" tells the server "prepare and start giving me all the data of that huge, multi million records table, please". Using IB_Query does not make you change that, since a "select * from customers" will bring you back to your client ALL the customers (if your query stays open for long, you will see an IBO message appear telling that IBO is automatically fetching the rest of the table...). You can set MaxRows property, but this does not reduce the server's work to prepare the whole data you requested, just limits the number of rows the client gets back to a fixed number (you can't have more rows if you need to navigate further).
But how can I build an user friendly data editing form, where the user can insert new records, or search for old ones? If you start with a closed dataset, the IB_Edit controls are greyed and the user can enter nothing on them...
The best solution is to put the dataset in search state, and provide some IB_Bars (IB_SearchBar, IB_NavigationBar and IB_UpdateBar, or mine IB_UpdateWorkBar). This way, the dataset stays closed but the user can enter search criteria into the visual controls and retrieve the data he wants to edit. If he wants to insert some data, he just presses "+" (add) button and a new record is inserted.
In the fields you can also use wildchars like "*" or "?", or also comparisons operators like ">" and "<" (i.e. Enter "> 10"), and also a value list using IN keyword (i.e. "IN(12,50,65)") to perform more powerful search. If, instead of standard exact match, you want that the text will be used with the SQL "STARTING WITH", set the column attribute "NO TRAILING" to True. If you want this to be the default for all your application, set the IB_Connection.DefaultNoTrailing to True.

Use meaningful field names
If you use meaningful field names, you can save a lot of time while building a prototype or also the final version of your program. In fact, IB_Grid and AutoLabels (of IB_Edit and a lot of other IBO components) use field names to produce the corresponding label. If you use the underscore as a separator, it's transformed in a space, making the label even more "professional" (i.e. CUSTOMER_NAME will produce the label "CUSTOMER NAME"). If you want a different name, you can use DisplayLabel field property for tailoring IB_Grids, or enter the label text you want in the autolabel Caption property. But please, with Firebird don't use those bad field names that always need to be enclosed into double quotes. Prefer the "not so beauty" ALL UPPER, no spaced, standard field names that will make your SQL programming a lot easier!

DML cache
----- Next time.

Further topics roadmap:
- SQLLock and LOCK_FLG
- Field names with '_' that autolabel or IB_Grdid translates into space (es. CUSTOMER_NAME)
- What to do to refresh dataset. Events BeforeOpen and OnPrepareSQL, use of close/open/invalidateSQL, refresh, etc.
- Short note about the use of SUSPEND in stored procedures. Always use it if the procedure will be called

with SQL SELECT instead of EXECUTE.