ASP-DB INTERFACE

TABLE OF CONTENTS

I. Installation

A. Requirements

Server-side requirements:

- IIS v4 or above,
- VBScript v5 or above,
- MS Excel 97 or above (required if the *exportExcel* function is used).

Client-side requirements:

- Browser I.E 4 or above,
- Scripting languages and cookies enabled.

B. Package installation

Server-side :

?? Unzip the archive Dbtools in the web site folder.

?? Copy the capcomDCT.dll file into \inetpub\Scripts and type at command prompt *regsvr32 capcomdct.dll*. This installs a free COM component that is necessary to have a dictionary object at application level.

?? Set the ROOT constant to the current web folder. DBTOOLS can now be used as an include file in the scripts.

C. PIMS (MMIHND) example installation

Server-side :

?? Unzip the archive PIMS,

?? Create an ODBC connection versus an Oracle Database called FM.

?? Share this folder as a web folder called PIMS.

II. Features

It allows to develop the MMIHND part of the Product Information Management System in a configurable way. The standard ASP pages (view and select data of a table / edit a record / delete a record / create a record) can be written without hard coding any SQL request in the edition pages.
It is done by providing high level function (see Package functions) that are then used in the pages. These functions rely on a configuration file. Thus, a configuration process is needed to make the scripts work properly (see Configuration).
It must be stressed that this package does NOT constitute in any way a constraint for the development ; the use of the provided functions is not necessary all over the web site.

A. View page configurable items :

- Table outlook (colors, fonts, margins…)
- Number of columns and their names in the database.
- Columns captions.
- Number of buttons (like Cancel, Edit, Ingredients…) for each record line in the table and their pictures.
- Fields, which can be used for querying (dates, numbers, strings, list boxes, check box, radio buttons).

B. Update/Add pages configurable items :

- Fields than can be edited.
- Fields, which value is printed but not user-editable.
- Fields “AutoNumber”.
- Fields, which values are given by list-boxes, calendars, radio buttons or check boxes.
- Restricted access (one user at a time)

C. Other features implemented

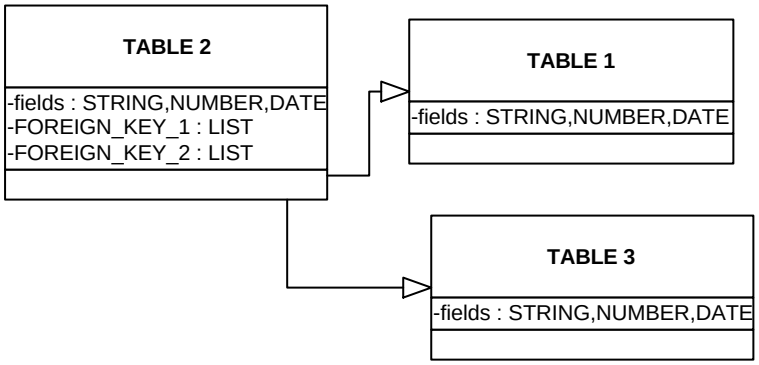
- Supports three RDBMS : Oracle, MSJET and mySQL.
- Deletion of records.
- Secured access by login/password (via checkAccess).
- Log of user actions and errors (requires Comune.inc -> WriteLog).
- All the strings (returned to the user or into the log file) can be translated.
- Transactions on multiple pages.
- Import / Export of record sets: either via CSV native encoding functions, either via Excel (then it supports other formats like DBF, XLS, text etc.).

D. Limitations

?? Any user action (delete, insert and update) will update only one table.

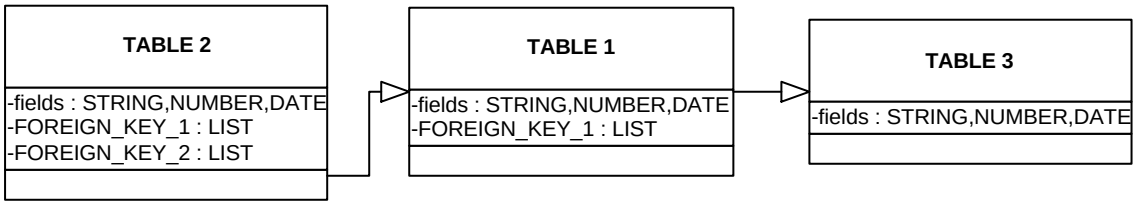
?? Only one level of foreign keys can be used for the update / insert / query fields. However, there is no limitation on the number of foreign keys.

In other words, this example is legal:



(The user will see two list boxes of the selected table1 and table3 fields. The table 2 foreign keys will be updated on user actions).

This example is **not** legal:



FOREIGN_KEY_1 and FOREIGN_KEY_2 cannot appear in *insertFields*, *updateFields*, *queryFields* strings (see next section for explanations). However, it is possible to display TABLE3 fields.

III. Configuration

The entire table configuration process in done in the file **global.asa**. For each table used by the scripts, we create a Caprock.Dictionary object into the Application object. This dictionary (fully compatible with the standard VBScript.Dictionary) has several keys, described below.

A. Dictionary keys

Tables are divided into two categories, “simple” ones that aggregate data from only one table in the database, and “composite” ones that aggregate data from two or more tables in the database.

STANDARD TABLE DICTIONARY KEYS

Key name	Type	Description
name	string	The table name as it appears in the database, or a "*" character if it is a composite table
MAIN PAGE		
visibleFields	string like (elem1)(elem2) ... (elemN)	The columns names as they appear in the database. If it is a composite table, they must be preceeded by the table source number (or by the last 10 chars of the table's name).
visibleFieldsNames	string like (elem1)(elem2) ... (elemN)	Columns titles as they will be displayed on the user's screen. **If it begins by "LED"**, it is supposed to be a CHECK type which value will be represented by a green or red led.
visibleColumnsImages	string like (elem1)(elem2) ... (elemN)	Pictures used as links to action scripts. Will be prefixed with the root constant. E.g. (/Images/Delete.gif)
visibleColumnsParameters	string like (elem1)(elem2) ... (elemN)	Database fields names (for composite tables, prefixed by the last 10 characters of the source table) which value will be passed to the action script, via URL-encoding. E.g. : (ID)
visibleColumnsScripts	string like (elem1)(elem2) ... (elemN)	Action scripts names. E.g. : (delete.asp)
EDIT PAGE		
updateFields	string like (elem1)(elem2) ... (elemN)	Columns of the table that will appear on the edition page. For a composite table, that is a subset of table 2 fields. E.g : (ID)(NAME)
updateFieldsType	string like (elem1)(elem2) ... (elemN)	Each element must be one of these : DATE, STRING, NUMBER, AUTO, LIST:<valuesSource>:<descriptionSource>, CHECK, RADIO:<groupName>
updateFieldsNames	string like (elem1)(elem2) ... (elemN)	Names of the fields as printed on the edition page. E.g. : (Ident)(Product name)
updateDisabledFields	string like (elem1)(elem2) ... (elemN)	Subset of updateFields : fields which value is displayed, without allowing the user to edit them. E.g. : (ID)
INSERT PAGE		
insertFields	string like (elem1)(elem2) ... (elemN)	Columns of the table that will appear on the "Add new..." page. For a composite table that is a subset of table 2 fields. Must contain all the not-null constrained database fields.
insertTypeFields	string like (elem1)(elem2) ... (elemN)	Each element must be one of these : DATE, STRING, NUMBER, AUTO, LIST:<valuesSource>:<descriptionSource>, CHECK, RADIO:<groupName>
insertFieldsNames	string like (elem1)(elem2) ... (elemN)	Names of the fields as printed on the "Add new..." page.
QUERY MASK		
queryFields	string like (elem1)(elem2) ... (elemN)	Columns of the table that appear in the query box. For composite table, they must begin by the table number they come from.
queryTypeFields	string like (elem1)(elem2) ... (elemN)	Each element must be one of these : DATE, STRING, NUMBER, AUTO, LIST:<valuesSource>:<descriptionSource>, CHECK, RADIO:<groupName>
queryFieldsName	string like (elem1)(elem2) ... (elemN)	Names of the fields as they will appear in the query box.
ACCESS CONTROL		
accessControl	Boolean	If set to true, only one user at a time can edit / cancel / add a record to the table
currentlyUsed	Boolean	Must be set to false in Application_OnStart
currentUser	Integer	Must be set to 0 in Application_OnStart

COMPOSITE TABLE SPECIFIC DICTIONARY KEYS

Key name	Type	Description
`numTables`	Integer	Number of database tables involved.
`Table1`	String	The table name as it appears in the database.
`Table2`	String	The table name as it appears in the database.
`Table3..9`	String	Needed only if numTables >2. The table name as it appears in the database.
`allFields`	string like (elem1)(elem2) … (elemN)	All the columns names of the database tables that will be used later on.
`allFieldsSourceTables`	string like (tableNumber) … (tableNumber)	indicates from which table comes the corresponding element in allFields.
`condition1,condition2,condition12`	String	SQL WHERE statements that will be used to build the main page. They are all connected by AND by the package. E.g. "PRODUCTS.ID=BATCHES.PRO_ID"

B. Types description

The building of SQL statements and query inputs depends on the fields' types. Moreover, they involve different validation checks. Here is explained the differences between the types.

- ***String***: the user will have a textbox to edit/query/add a record. The entry will be translated into 'user_entry' in SQL. The quote character in the user entry is forbidden (so as to avoid SQL syntax problems and security holes).

- ***Date****:* the user will have a textbox followed by a calendar icon to help him to enter the date (in edit/add pages). In the query form, he will see two textboxes (lower and upper bound) with calendars. The dates will be translated in SQL using the *to_date* function of Oracle. The date format string argument of this function can be set in DBTOOLS.INC.

- ***Number*****:** the user will have two textboxes (Min and Max) in the query form. In the other pages, only one textbox will be seen. The user entry will be evaluated before action: if its value is 0, an error message will appear.

- ***List:<valueSource>:<textSource>*** : it is used to implement the foreign keys. Will be displayed as a listbox, which items are the values of the *textSource* field.

e.g :

The table FLOWS_LOGS has a field FLO_ID which type is <LIST:FLOWS.ID:FLOWS.DESCRIPTION>

It will build a list box with the DESCRIPTION FIELD. Internally, each item in the list box is associated to its ID value in the FLOWS table. When the user performs an insertion/modification, the FLO_ID is updated with the ID value, though the user never sees or edits the ID value.

Please note that the table referred by *valueSource* and *textSource* must be the same.

?? ***Check****:* in the edition or Add new… page the user will have a tick box. In the query mask, there will be a list box with three entries: empty, YES and NO. In the table, the user will see YES or NO to describe the value of the field, except if the name of the column (element of *visibleFieldsNames*) begins by "LED" : in this case, a green or red LED will be displayed. In any action, the database field will be filled will 'YES' or 'NO' strings.

?? ***Radio:<groupName>* :** in the edition or Add new… page, the user will have a radio button. For each field with the same group name, only one button can be selected at a time. In the table, the user will see YES or NO for any field of the group. In any action, the database field will be filled with 'YES' or 'NO'. This type is not supported into query masks.

C. Special tags in GLOBAL.ASA

So as to allow the automatic configuration of global.asa, some special comment tags are used : they identify the blocks to be generated by the global configuration program.
They all look like:

' DO NOT REMOVE THIS LINE <sectionName>

…

' DO NOT REMOVE THIS LINE </sectionName>

D. Other points of configuration

?? Tables outlook can be configured editing the style sheet ocrim.css.

?? Strings can be translated changing the value of the STRINGS dictionary keys.

?? Root directory of the web site: change the ROOT constant in Comune.inc (by default : "\pims").

?? The SQL generated strings can be Oracle-compatible, MSJet-compatible (useful when wanting to use an access database for debugging) or mySQL-compatible : just change the value of the RDBMS constant.

IV.Package functions

A. List of functions and procedures

The basic functions of the scripts: displaying a table, making specific queries, edit and cancel records are encapsulated into high-level functions.

1. HTML rendering

- **BuildInsertForm(connection,tableName,actionScript) :** displays a table with lists/input boxes for the insertFields

- **buildQueryForm(connection,tableName,actionScript)** : given a table name, generates a query form with a Submit button. Note this programming information: it calls *Response.Flush(),* so no *Response.Redirect* can be executed after a call to *BuildQueryForm* (it allows the user to see the query mask when the table is being built, which could be useful on very large and complex tables).

- **buildSimplePage(tableName, pageName, addButton, addLink, printButton)** : build a simple view page with a query mask and, on request, add/print buttons. Manages the DB connection, calls *buildQueryForm,record2tablePaginated.*

- **BuildTable(recordSet,tableName,urlNavigation,currentPageNumber,NMAXPERPAGE)** : given a table name and a record set, builds a paginated table showing only the VisibleFields (in order of enumeration in the string).

- **BuildUpdateForm(connection,tableName,actionScript,currentValues,currentID) :** displays a table with lists/input boxes for the table's updateFields.

2. I/O on database

- **DeleteRecord(connection,tableName,ID) :** cancels the ID record of the given table. It logs eventual errors and their description. Returns true if it succeeded.

- **DeleteAllFromTable(connection,tableName)** : tries to delete all records in the given table. It logs eventual errors and their description. Returns true if it succeeded.

- **GetCurrentValues(connection,tableName,ID)** : performs a request on the database and returns a string containing the field values of the current edited record (used to build the update form).

- **GetAllFromTable** ?**Where**? ?**Order**?**:** returns a record set containing all the fields of the given table name matching the condition. Can use an order condition. Can gather only distinct values.

The names of the record set fields are as the names of the source field if the table is not composite, otherwise it is <last ten characters of the source table name ><name of the source field> (e.g. PRODUCTSID). Only the 10 last characters are used so as not to exceed the maximum length of an Oracle identifier.

- **getOneFromTable(Where) :** returns a record set containing the specified field of the given table name matching the condition. Can be ordered. Works transparently with composite tables.

- **GetNextID(connection,tableName)** : returns an integer that can be used as primary key for the given table.

- **InsertInto(connection,tableName,form)** : inserts into the given table a new record with the values inside the form object. It logs eventual errors and their description. Returns true if it succeeded.

- **UpdateTableFromForm(connection, tableName, ID, form) :** copies into the database the form values given by BuildUpdateTable. Returns true if the update succeeded.

- **rollBack(objectName)** : rolls back the transaction then destroys the object and its key in Session.Contents. If an error is encountered during the commitment (it should not), returns false. It logs eventual errors and their description.

- **commit(objectName)** : commits the transaction then destroys the object and its key in Session.Contents. If an error is encountered during the commitment (it may happen with concurrent transactions), returns false. It logs eventual errors and their description.

- **importFromCSV(connection, tablename, fileInput) :** tries to import the CSV file into the database table by creating new records. The CSV file should contain all fields in the same order as the table (even the ID). Returns true if everything proceeded well.

- **exportToCSV(recordset, fileOutput)** : write all the records in the recordset in a CSV file. The first line is filled with the column names. The values of the records should not contain any comma to preserve the CSV integrity. Returns true if everything went well.

- **exportExcel(recordset, fileOutPut, format, headers)** : this function will use an Excel.Application object to do the export. The output format is one of Excel's recognised ones. Of course, it needs Excel (v. 97 or above) on server-side. If *headers* is set to true, it will add the columns names as first line of the spreadsheet.

3. Other

- **CheckInsertForm(tableName, form) :** verifies whether the user's entries are correct or no (use it before to call *updateTableFromForm*).

- **CheckUpdateForm(tableName, form) :** verifies whether the user's entries are correct or no (use it before to call *insertInto*).

- **GetSessionVarsFromForm(tableName)** : updates the session variables values with the form values (used for querying)

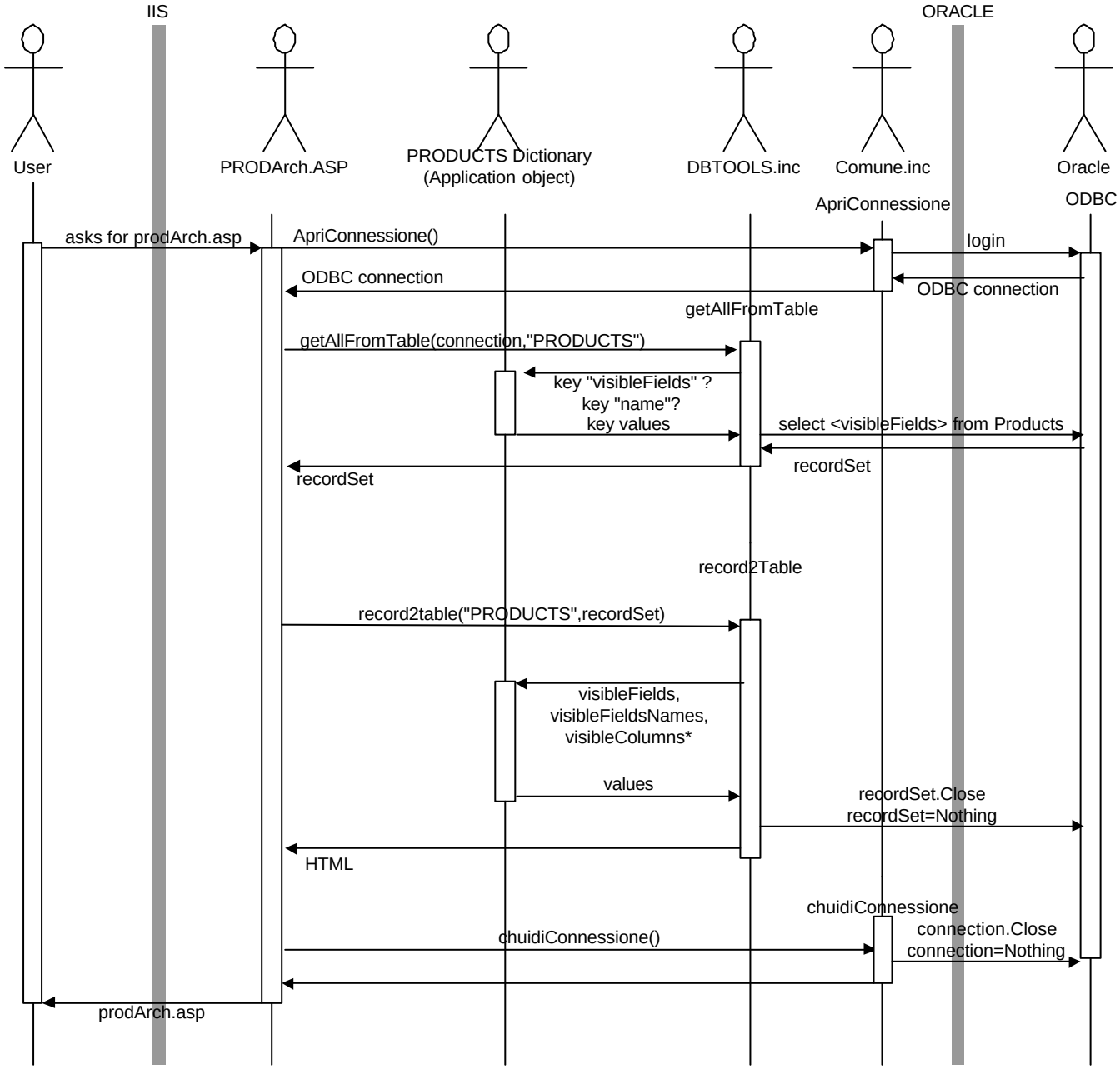
- **GetWhereStatement(tableName)** : given the data typed by the used, builds the SQL condition.

- **CreateSessionConnection(objectName)** adds an opened ODBC connection to the Session.Contents dictionary. It logs eventual errors and their description.

- **SessionConnectionPresent(objectName)** : returns true if the connection at session scope still exists.

B. Use case : view page

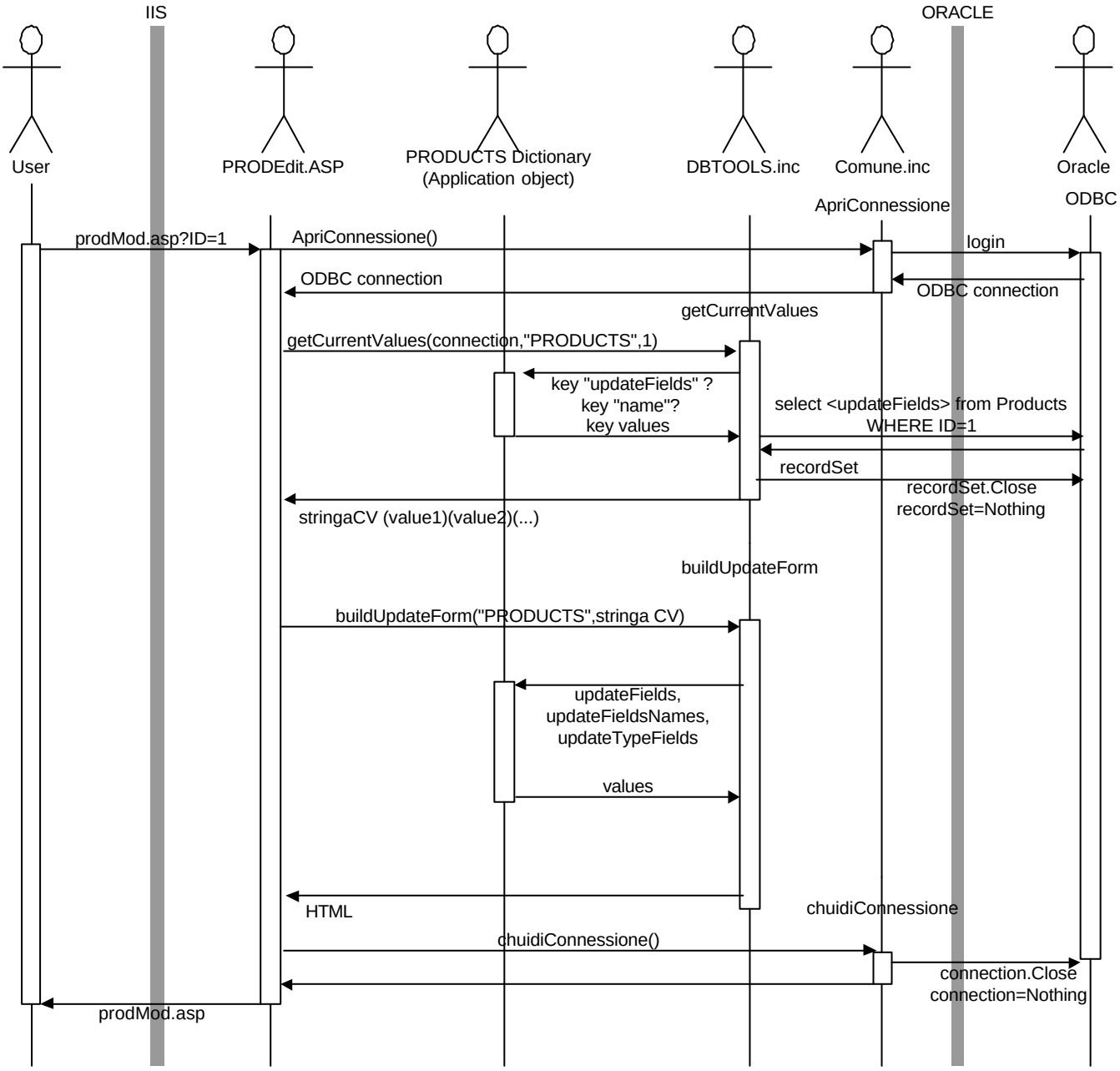
In this example, the call to ***BuildQueryMask*** has been omitted: since it involves the Session object to store values during post-back trips, it would have led to an unnecessary complex schema. It can been noticed that ProdArch.ASP makes no direct call to the ODBC layer.



C. Use case : edit page

One user asks for the edition page, by clicking the picture “Edit” of one record, in the main page.

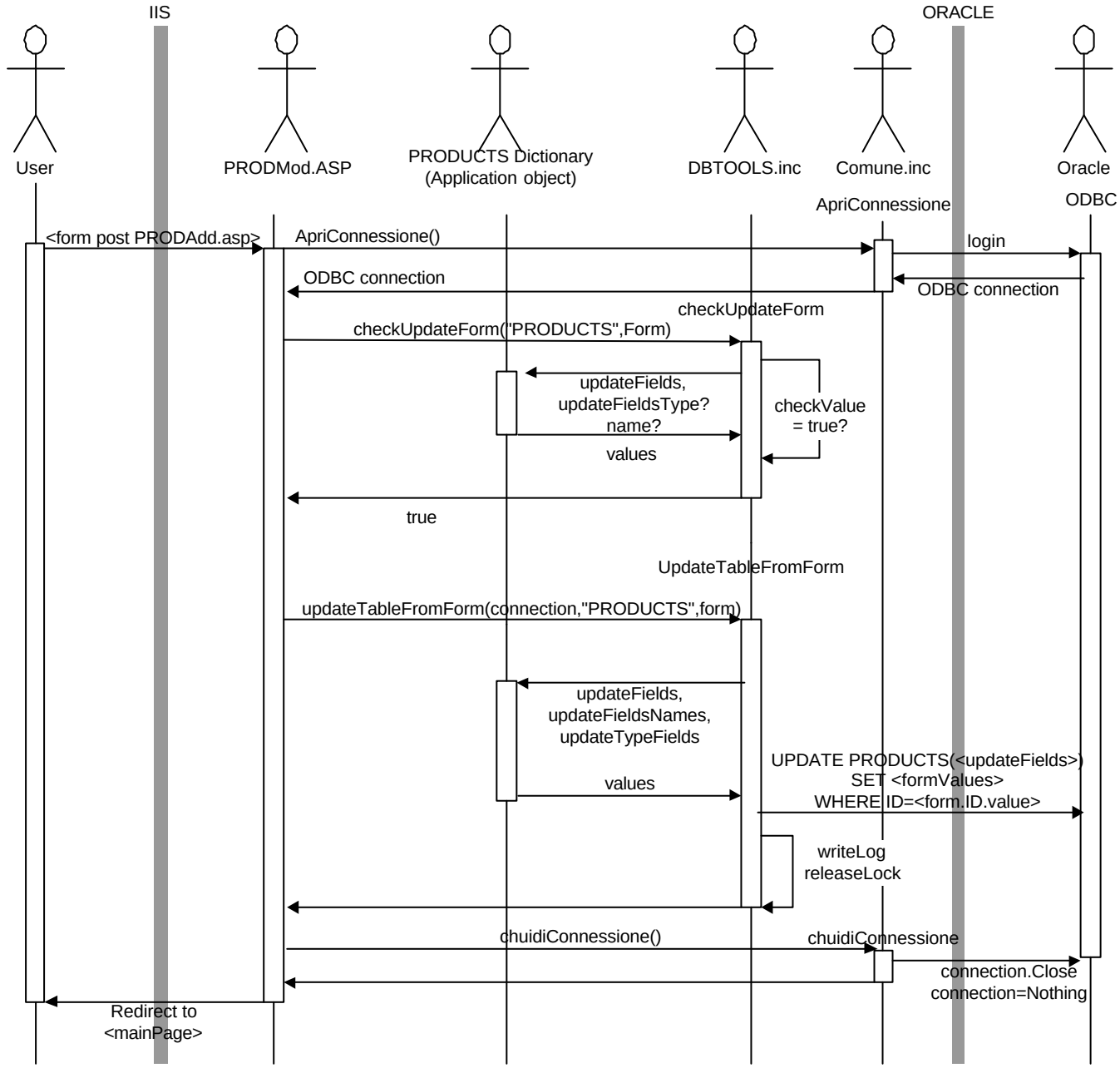
In this diagram is omitted the call to ***CanAccess*** by ***BuildUpdateForm***, which checks if another user is modifying the table products): for more details about it, see VI.D : Use case.



D. Use case : form post on the "Add new…" page

The user posts a form (by clicking the submit button), which values are all correct. The database is updated and he is redirected to the main page.

Note the call to ***ReleaseLock*** and ***writeLog*** inside ***UpdateTableFromForm***.



V. Internal functions

A. Configuration strings management

GetField(source,n) : return the n-th element into in the source string (enclosed into parenthesis)
GetIndice(source,elem) : returns the index of the elem in source. Case insensitive.
NumF(source) : returns the number of fields in the source string (separated by a parenthesis).
NumSF(source) : returns the number of fields in the source string (separated by ":").
ParseString(string, number, delimiter) : generic function that parses a string, used by getField e getSF.
GetSF(source,elem) : returns the n-th element into in the source string (delimited by ":").

B. Form manipulation

CheckValue(value,type,name) : checks is the value is coherent with the type, if not displays an error message. It detects illegal characters.

C. HTML rendering

Array2ListBox : builds a listbox from two arrays (texts and values).
Tag2listBox : translates a <LIST:…:…> tag into two arrays and calls array2ListBox

D. Table locks

CanAccess : returns true and sets a lock on the table if the user can access the table. Otherwise returns false.
SetLock : called by CanAccess, sets a lock on the table.
ReleaseLock (DBTOOLS.inc) : called by insertInto, deleteRecord, updateTableFromForm once done the update.
ReleaseLock (releaseLock.asp) : called remotely by the client at the page unload event. Cancels the lock eventually owned by the client.

E. Other

toMYSQLDate(string) : translates a "DD/MM/YYYY" string in a "YYYY-MM-DD" string.

VI. Locking system

Using the *accessProtected* key of a table dictionary, it is possible to limit the access of some crucial pages to one user at a time.

A. Locking rules

When a user requests a page (insert/delete/update/import…), the **canAccess** function is called. Since the **canAccess** call is inserted into the standard functions **buildUpdateForm**, **buildInsertform** etc., no modifications are needed in the standard ASP pages.
If the table is not protected (*accessProtected*=FALSE), it returns true.
If it is protected and someone else uses it, **canAccess** returns false and an error message is displayed.
If it is protected and not used by anyone, **canAccess** calls **setLock**, which sets a lock on the table for the user.

B. Unlocking rules

- The user posts the form: the **releaseLock** function is called automatically after database update. Even if the update fails (because of data integrity rules, for example), the lock will be released.
- The user goes to another page: the Unload event of the protected page calls a server function that removes the lock (if this user had really one lock on the given table).
- The user stays inactive: after 5 minutes, he is redirected to the main page, which triggers the **UnLoad** event of the page that releases the lock.

In case of abnormal client browser termination, network failure, etc., the **TimeOut** event of the Session object removes all the locks owned by the user, after 5 minutes of interrupted communication with the browser thread. Since they are not saved on disk, a server failure removes all the locks.

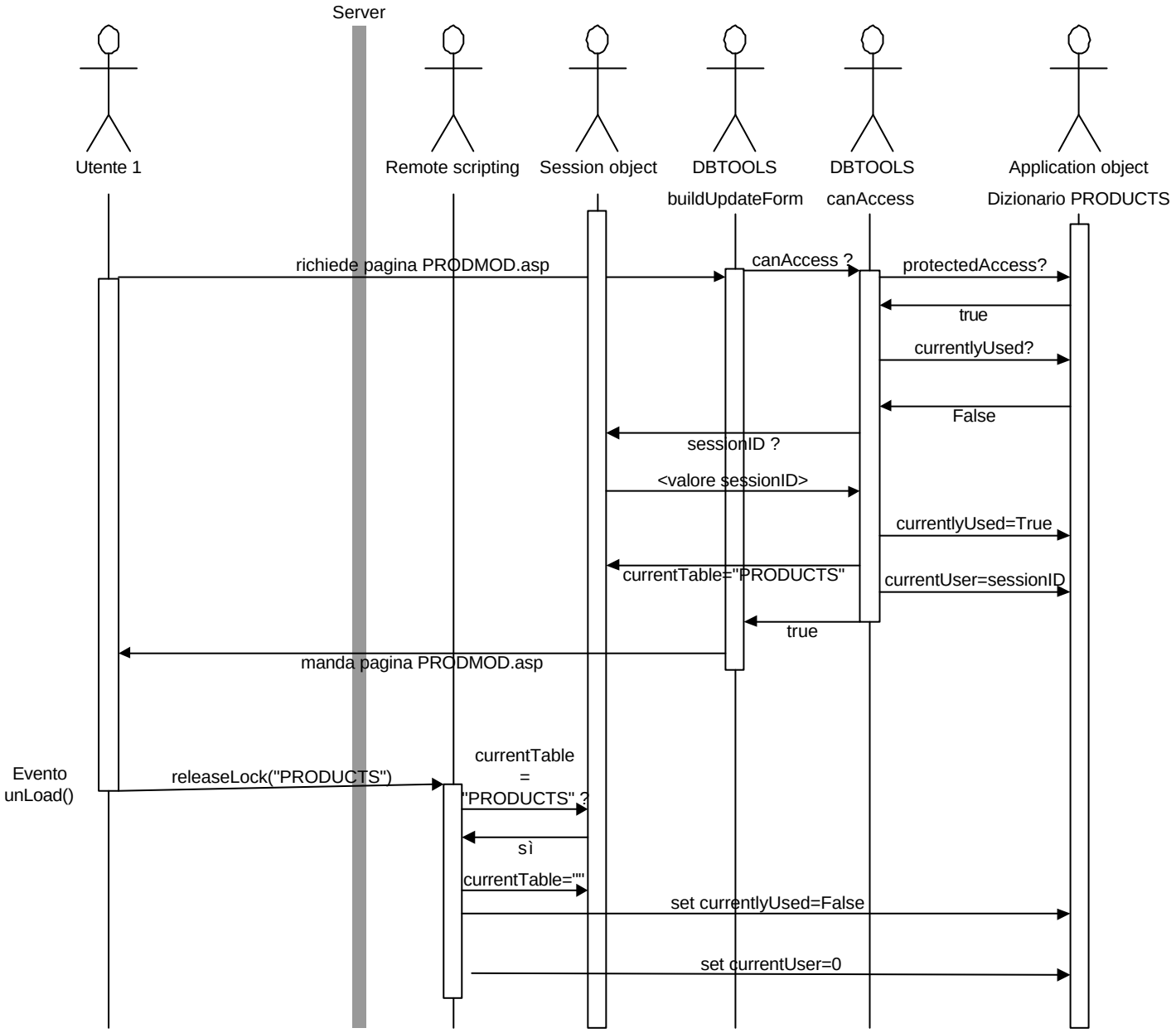
C. Implementation

At session scope, *tableUsed* contains the name of the table locked by the user. If it is an empty string, the user has no lock.

- At application scope, each table dictionary has one flag to indicate whether it is locked and one *SessionID* to store the user identification.
- To allow a call on a server-side function at **Unload** event on client-side, the remote scripting library is used.

D. Use case

One authorised user asks for the “PRODMOD.ASP” page, which is initially not used by anyone. Then he closes the browser. Here is what happens:



?? On Session.TimeOut event, the same routine as page.Unload is triggered.

VII. Error trapping

It is implemented with *On Error* statements in the package functions. It uses the *Errors* collection of the *ODBC Connection* object too.

A. Deletion of a foreign key referenced record

If the user attempts to delete a record, which is referenced by another table, the deletion will **not** occur.
An error message will be returned (by default: "This record could not be cancelled because of integrity rules.") and an entry will be added to the log file (by default : “error encountered when trying to execute “ + SQL statement + error description).

B. Edition

A check is made on types *STRING, NUMBER, DATE* before update. Since the other types are managed internally, they all should be correct.
If an error occurs when executing the SQL statement, (if some extra checks rules are implemented in the database, it can happen), the user will see a generic error message, the details of the error will be logged, and he will be invited to go to the last visited page.

C. Add New

A check is made on types *STRING, NUMBER, DATE* before update. Since the other types are managed internally, they all should be correct.
If an error occurs when executing the SQL statement, the user will see a generic error message, the details of the error will be logged, and he will be invited to go to the last visited page.

VIII. Transactions

A. Requirements

- Oracle Drivers for ODBC (not tested with Microsoft ones).
- Oracle 8i or more database (note that MS Access or mySQL do not support transactions).
- Like for the standard connection, the calling pages have to manage the session connection life cycle: the object is **not** monitored by the package functions (except in *session.OnEnd*). However I provide an example that works with the FeedMill Master Recipes pages.

B. Concurrent transactions

- If two users have pending transactions involving the same record in a table, the last user will be stopped until the first user has committed or rolled back his query (standard Oracle driver behaviour). If this delay exceed the IIS script time out delay, the user will have an error message. A way to prevent this would be to use the locking system provided below.
- Two users can work on the same table (but different records) without blocking.

C. Transaction life time

- Transaction time out should be set to five minutes (ODBC driver parameter).
- Session.OnEnd rolls back any opened connections at session level.

IX.Possible improvements

A. *Data formatting*

It is possible to add a special field type (like “HTML”), which would mean that the field name will be directly sent to the document, allowing to insert some HTML fragments into the form building and improving the look of the forms.
It is also possible to create in the file *ocrim.css* new styles that will be used by *buildInsertForm*, *buildUpdateForm* etc., again for a better formatting.

B. *Import*

Excel files could be used as an import format, if for any database table there is a worksheet in the document with the same name. By using the Excel object, it is quite easy to implement (but pay attention to its life cycle, since it consumes 8 Mb on the server!).
In spite of creating new records during importation, it should be possible to update the existing ones.

C. *Data retrieval speed*

To print the table on the screen, only NMAX_RECORDS are needed. However the whole table records are retrieved. It should be possible to create a function *getRecordPageFromTable(pageNumber, table, connection)* which would select only the required records.

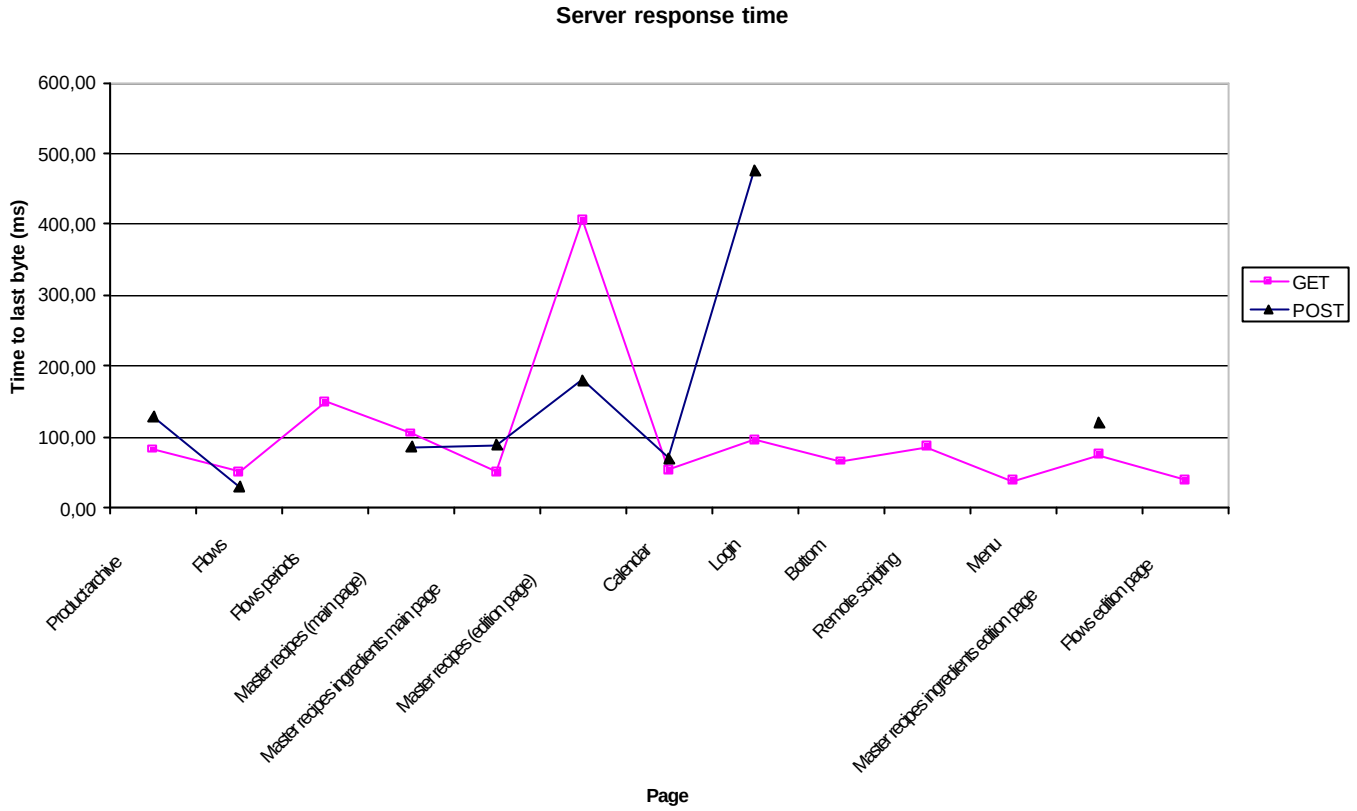
X. Conclusion

A. *Benchmark*

Nine clients simulated (via MS Web Application Stressing Tool).
Average time between each client’s requests: one second.
Testing duration: 10 minutes.
Server configuration: PII 500 Mhz, 512 Mo RAM, with IIS 5 and an Oracle 9R2 database accessed through ODBC (pooling active).

Every page in the demo was requested, some queries were made on the main view / update pages. A transaction was committed in the Feedmill recipes pages.

The slowest computation occurs for *Login.asp* : looking through the code, I noticed that one record set object was not set to nothing after use (so the garbage collecting process needs more time). It underlines the interest of this package that encapsulates low-level details, avoiding these problems.
The master recipe edition page is quite slow in reason of a great number of *updateFields*.



On this diagram, the GET serie represents the latency to get a page, and POST the time for the web server to compute the user's form.

B. Contact

I can be reached via e-mail: brunot.pascal@wanadoo.fr, if more explanations are needed.