EP

United Nations
Environment
Programme

UNEP(DEC)/MED WG.231/Inf.11
17 April 2003

ENGLISH

MEDITERRANEAN ACTION PLAN

Meeting of the MED POL National Coordinators

Sangemini, Italy, 27 - 30 May 2003

MEDITERRANEAN POLLUTANT RELEASE AND TRANSFER REGISTER (PRTR) PILOT PROJECT:

DEVELOPMENT OF DATABASE AND WEB SITE

UNEP
Athens, 2003

TABLE OF CONTENTS

INTRODUCTION

This white paper contains the description of the software written for the PRTR and specifically the data access application. The application described in this white paper and developed within the framework of PRTR project is a Web application.

The task of the PRTR project and therefore of the software developed is to memorize data on the emission of toxic agents and to centralize the repository, in order to be able to process data and generate reports.

To centralize the data means to put at disposal a system to which all the interested parties can connect in order to insert the data and to request the reports.

These two concepts lead to the logical consequence of creating an Internet site that provides these services.

The Internet site has to recognize the single user, in order to allow only his own data entry or modification and the report requests. Therefore, appropriate web-forms are needed to allow user's identification as well as data entry or reports generation. This idea could be put into practice if all the interested parties had an Internet connection. Unfortunately this is not the actual situation in the interested region. Consequently, in order to obtain the widest possible data collection, the Internet solution is not sufficient. A stand-alone program is needed to satisfy the lack of connection. It should allow the users to input the data in a local database, which should be afterwards sent to the central location of the PRTR project, that would provide the update of the central repository.

This macro-analysis depicts the need of the following three applications:

1. A web-site that can provide web-forms connected to the central database for users identification and for data entry;
2. A stand-alone application that replicates the data entry web forms (as per point 1) connected to a local database, which is a simplified copy of the central database;
3. A system for the reports generation.

SEARCH FOR THE SOLUTION

In the previous paragraph the necessity of the development of two applications, one web-based and one stand-alone has been evidenced.

The scope of these applications is to provide a tool which is both user-friendly and performing in terms of installation and maintenance. It means that:

1. PRTR operators should be able to easily modify the graphical layout as well as the contents of both the applications;
2. The users should not notice any difference between the stand-alone and the web-based application, this in order to facilitate the work continuity at the moment the user moves from the stand-alone to the web-site application.

According to point 1:

Modifications of a web-site connected to a database involve some insurmountable difficulties, even if the front-end can be simply modified with development tools. On the opposite, the modification of a layout in a stand-alone application can be problematical, because to this purpose application source code has to be available and then modifiable.

The development of a web-based system that can work also as a stand-alone application could present the solution to the matter. In this way, the two individual applications' creation could be ignored and the requirement mentioned in the above point 2 could be satisfied.

This web site should be installed to the parties that lack Internet connection. The web site forms should aim to a "local and simplified" version of the central database. In this case, the application would be a single-user solution; therefore the user identification forms wouldn't be necessary. It could be identified as a "light" version of the site.

Such solution implies the installation of a web server. The installation of a web server can cause a fall in the computer performances, as well as objective difficulties in the creation of the installation program for the local web site, because different versions of web servers are available for different operating systems (note that on Windows platforms only there are: personal web servers for win9x and IIS for NT, 2000 and XP) and there are also different installation procedures, comprising service packs or patches for the web server.

The "light" web-site is a valid solution. However, a methodology is needed to "replace" the web server, or rather to simulate in a single application all the interactions that run through Internet between the user, the browser and the web site, as well as the major number of implemented web server functionalities, in order to achieve a unique environment for application execution and application use.

ANALYSIS OF THE STAND-ALONE SOLUTION

In general, the interaction between the user's browser and the web site is based on the following concept: the user who needs to visualize one page of a web site sends through the browser an HTTP request to the web site. This request is received by a web server, which is actually the service that reacts on HTTP requests, analyzes the requested URL and, based on this, generates a result which is sent via HTTP to the user's browser.

An example of the functionality logic of a site, published using Internet Information Service, is described hereafter:

Request	**Action**
HTML Page	IIS returns the page immediately in HTML format.
ISAPI extension	IIS loads the ISAPI DLL (if it is not already running) and the request is sent to the extension through the **Extension_Control_Block** data structure.
File name extension mapped to a particular ISAPI extension	IIS loads the appropriate DLL file and presents the request through the **Extension_Control_Block** data structure. The .asp extension, for example, is mapped to Asp.dll, so that all requests for files with an .asp extension will be directed to Asp.dll.

In graphical terms:

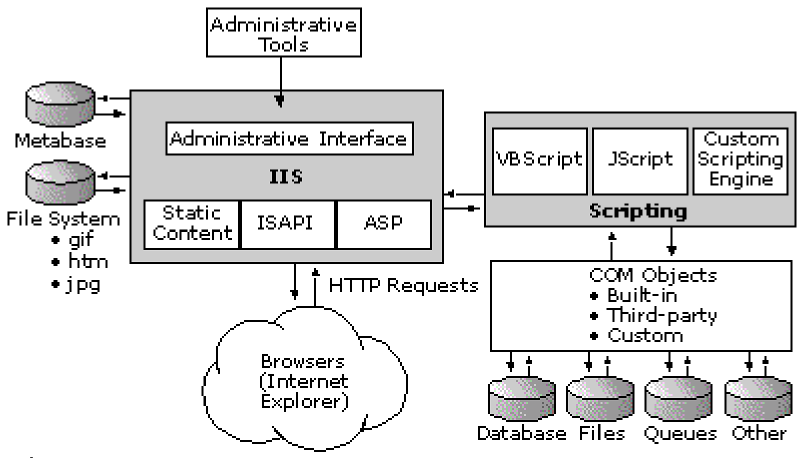
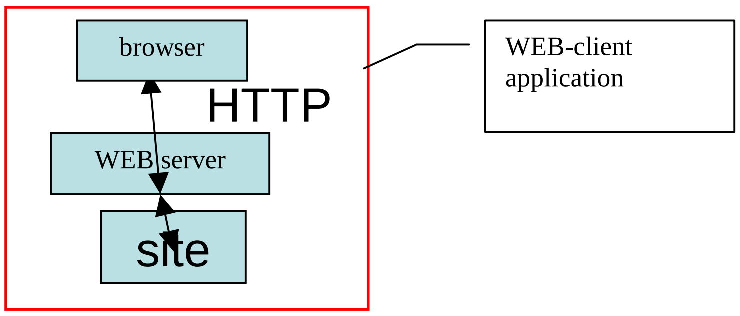


Figure 1

As illustrated in *Figure 1*, in order to simulate a web environment, the following has to be implemented:

1) Replication of a browser;
2) Replication of an HTTP request;
3) Capacity to react on a HTTP request;
4) Capacity of analyzing the URL;
5) Capacity of generating dynamic web pages if necessary, thus executing programs;
6) Capacity of sending a page to the browser.

With the creation of an application that we call "web-client", which satisfies the above 6 points, it is possible to make a "dynamic" web site inside a stand-alone application without actually installing a real web server (see *Scheme 1*).



Scheme 1

Scheme 1 shows that in any case the application can be divided in 3 parts:

1) UI, that simulates the browser;
2) A stream that simulates the HTTP;
3) An object that simulates the web server.

These 3 parts depend on the chosen technologies in the sense that the conceptual solution is strictly connected to the technical solution.

This suggests that the analysis of the program could be performed after the development technology is chosen. The local version of the final application consists of the "web-client" application and the "light" site application. According to this logic, the final application is executed on the "light" site only, just like in the on-line version. This is much simpler when compared to a recompilation of source-code in a standard stand-alone application.

ANALYSIS OF THE WEB SITE IMPLEMENTATION

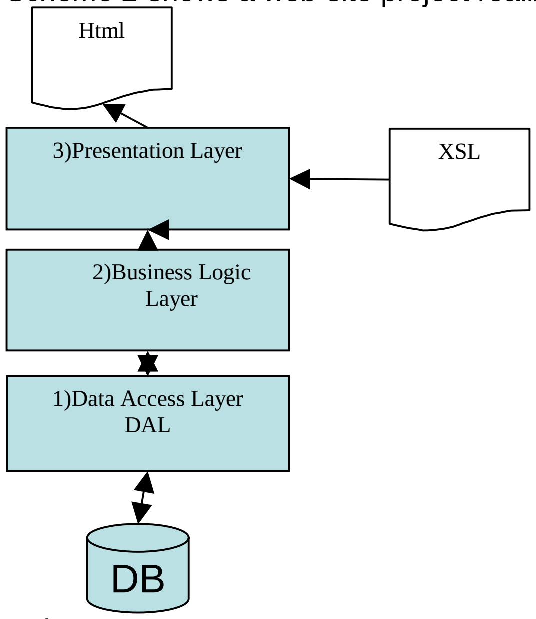
In the previous paragraph, the criteria for the development of a stand-alone solution have been defined. The stand-alone solution consists of a "web-client" application that represents the runtime environment for the "light" web site.

From the above considerations, some features that the web site should provide have been pointed out:

1) The "web-client" and the "real web" execution environments should be distinguished because of the following reasons:
    a. In the "real web" environment, the site should contain a part that "recognizes the user", while in the "web-client" environment it is useless;
    b. Depending on the execution environment, the site should generate the connection strings that would point to the proper data source: the "local database" in the "web-client" case, and the "central database" in "real web" case.
2) The site should provide forms that could be easily modified, and form content which should be as much as possible independent from the graphical layout.

These features can be implemented by the use of the "multi-tier" development technique. The functionality is divided in different homogeneous layers that enable the partial modification of the product.

Scheme 2 shows a web-site project realized with the use of these techniques.



Scheme 2

As shown in the scheme, all functionalities are allocated in separated blocks:
1) DAL – Data Access Layer: this "logic" block contains all the functions that allow the interaction with the database, i.e. opening the connections, data retrieve or data modification;

2) BLL – Business Logic Layer: this block contains all the functions that define "how and where to read or write the data". It also provides other functions such as user authentication;
3) PL – Presentation Layer: this block
    a. Contains all the functions that transform data to XML form, and afterwards to HTML form;
    b. Provides all the methods and properties for the "use" of subordinate blocks.

The construction of the blocks actually does not follow precise rules. This technique instead suggests a plan to be followed in the development process, but it does not impose strict limitations.

The described technology satisfies the above mentioned features. The first feature is fulfilled through the use of two BLLs, one for the "light" version and another one for the "on-line" version. The second feature is fulfilled through the use of a form as an XML document and a graphic layout as a XSL document in the PL.

ANALYSIS OF THE SITE ENGINE IMPLEMENTATION

In the previous paragraph the main functionality of the system has been identified as well as the development approach. It is necessary to analyze how to construct the engine which manages the web forms and the site.

In the engine construction, two approaches have been considered:

The first one assumes the construction of an "ad-hoc" engine on the PRTR database, using the multi-tier technology. This means that constructed BLL will be strictly connected to the structure of the PRTR database.

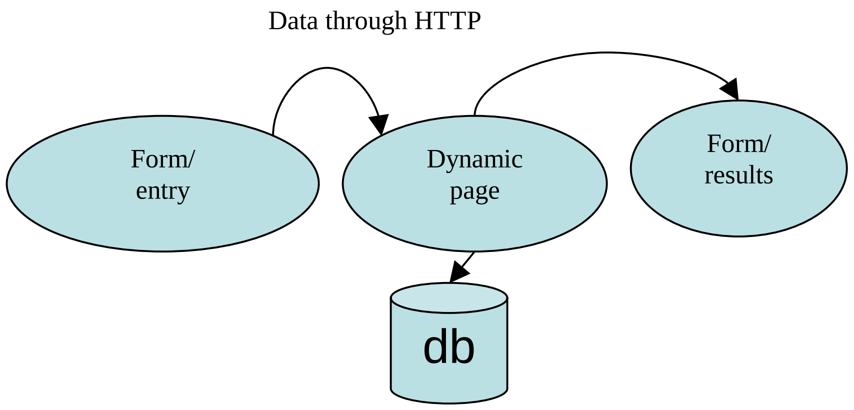
Using this methodology, results in a system designed and implemented for the solution of one single problem. Such system is very per formant as one single form accesses to several tables. However, the system re-usability is limited. This approach has been abandoned in favour of the second one, which assumes the construction of a "generic" engine that should be able to adapt itself to a certain number of possible cases, among which is also PRTR. In this way, the software becomes reusable.

It has been decided that the "engine" should have the potential to manage "automatically" databases with at least the one-to-many relation. To understand how to develop this idea, a program for the management of relational databases, like Microsoft Access is taken as an example. Besides the RDBMS functions, Access can generate the forms connected to the tables and queries for the addition and modification of database records. In case the display of data from tables with one-to-many relations becomes necessary, it is possible to create a form (named "main") connected to the primary table, which holds another form (named "subform"). The subform visualizes data from the related table. The main form and the subform are connected in a way that makes the subform show only the records related to the current record in the main form.

The above mentioned example indicates the type of navigation that can be used to simulate the form/subform concept. This navigation dictates moving from the grid forms, used for visualizing table records, to:

a. A "single form" for data insertion or modification. In case of modification, it is first necessary to select the record in the grid;
b. Other "grid forms" for the visualization of the records related to the selected table.

At this point, the application of these concepts to the web environment is to be considered. All the values in the web form fields are usually obtained by using get and post methods, from one “dynamic” page (could be also the same page that sends them), which collects them and instantiates the object used for reading/writing from/to the database.



Scheme 3

Examining *Scheme 3* and the navigation modes considered in order to simulate the main/subform forms, it can be noticed that everything resembles a certain similarity with one sequential or state machine.

The Business Logic Layer (BLL) therefore should manage:
1. The “state machine” for the navigation;
2. The mapping fields - query parameters;
3. The mapping of the fields in the forms. It is clear that every single state generates one form.

Besides, BLL should be configurable. Therefore it is necessary to create a BLL connected to an external database called “workflow”, which should allow the configuration of the previous points.

The structure of the database is given as follows:

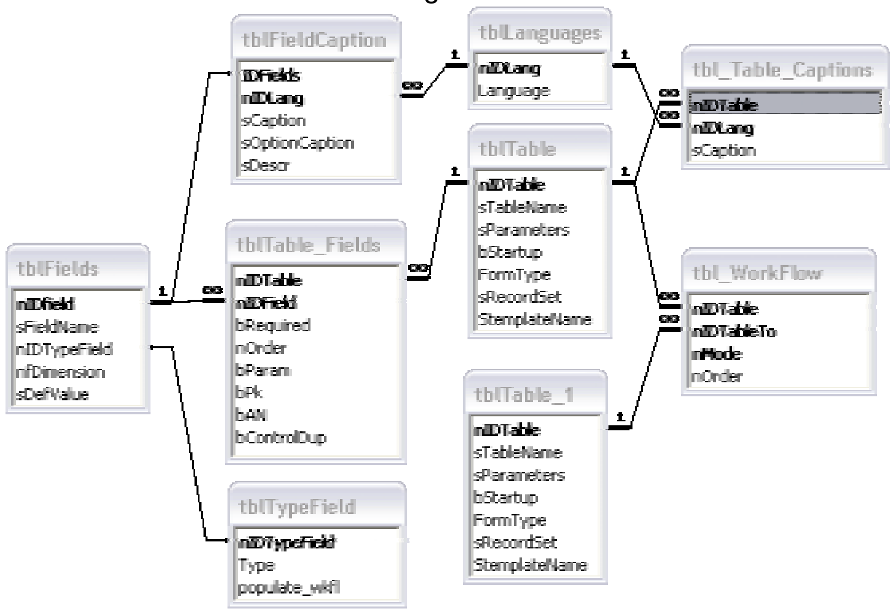


Figure 2

Table 1: Description of objects in db BLL

tblTable	Maps all the forms as states	
	sTableName	The name of the query which identifies the form. The query can be parametric.
	FormType	Distinguishes between “single form” and “grid form”.
	sRecordset	The name of the table/view on which insert/edit should be performed.
	stemplateName	Name of the XSL template to be applied when the form is generated.
	bStartup	Reserved for future use.
	sParameters	Reserved for future use.
tblTable_Fields	Each form contains fields and here it is indicated which fields represent the parameters for the query indicated in “sTableName” field of tblTable; which are the table key fields on which the form is constructed; and which fields are required to be populated.	
	bRequired	0,1 – Field required or not required
	bParam	0,1 – Represents the parameter of the query indicated in “sTableName”
	bPK	0,1 - Serves for the identification of the primary key used in “grid forms” specifically for the record selection or for the creation of the update queries by the engine.
	bAN	0,1 – The primary key of type “autonumber” serves as an indicator to the engine to exclude the field from the insert operations.
	nOrder	Numeric order of presentation of the fields in the form.
Tblfields	Each form has the fields mapped here, because one field can appear in more than one table.	
	sFieldName	Field name.
	nIDTypeField	Identifies the field type, the text box, the combo, etc. It is used by XSL to obtain the final HTML presentation.
	nDimension	Defines the field size.
	sDefValue	The default value of the field. It could be also a query (combo).
tblTypeField	The fields can have multiple “lookups” like text box, combo or data fields. This table maps every possible field type.	
	nIDTypefield	Denotes the fields.
	Populate_wkfl	0,1 – Indicates that the field values are obtained by a query.
tbl_Workflow	Manages the state machine	
	nIDTable	Initial state.
	nIDTableto	Final state.
	nMode	The navigation entry corresponds to Curr_nMode of the scheme. Values can be -1, 0, 1, 2

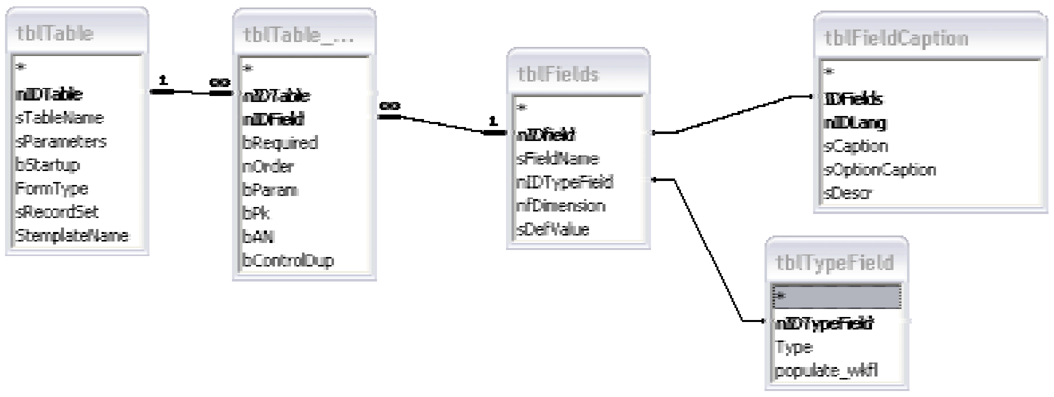
		and are defined by the navigation based on the number of tables to be interrogated.
	nOrder	Order of entry appearance in the form.

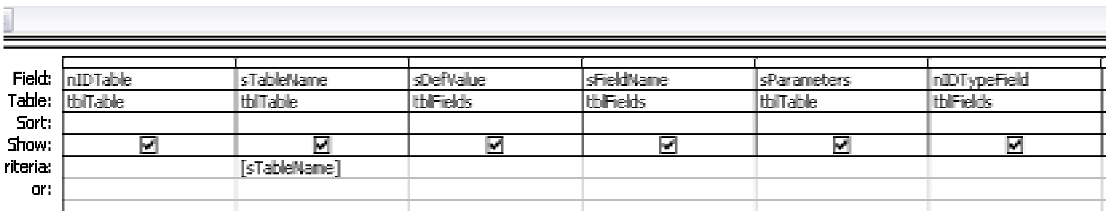
The Languages, tblfieldCaption, tblTableCaption tables manage respectively the “multilingual” captions of the forms and fields.

The queries that serve for the extraction of forms configuration and the state machine “workflow” are standard and are always the same:

1. “wp_qjTableFields”

   Extracts the fields of a form starting from the name of the query inserted in the field STableName of the associated table tblTable.

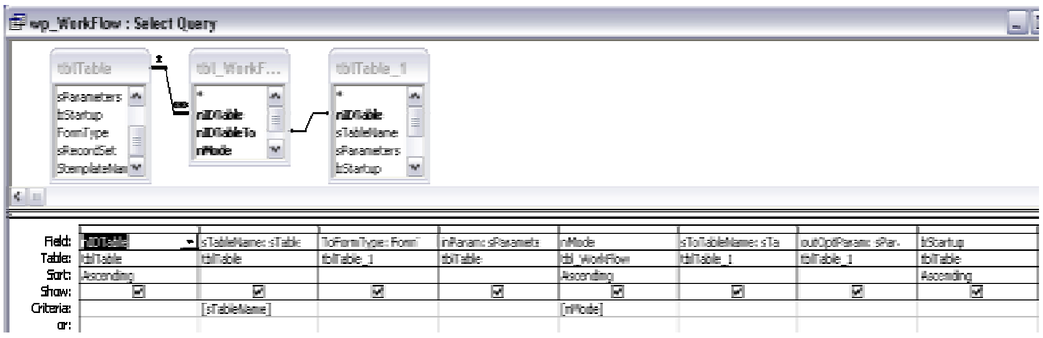




Field:	nIDTable	sTableName	sDefValue	sFieldName	sParameters	nIDTypeField
Table:	tblTable	tblTable	tblFields	tblFields	tblTable	tblFields
Sort:						
Show:	✓	✓	✓	✓	✓	✓
riteria:		[sTableName]				
or:						

2. “wp_workflow”

   Extracts the final state “curr_sTablename” according to the initial state “prev_sTableName”, and the entry “curr_nMode” chosen by the user.

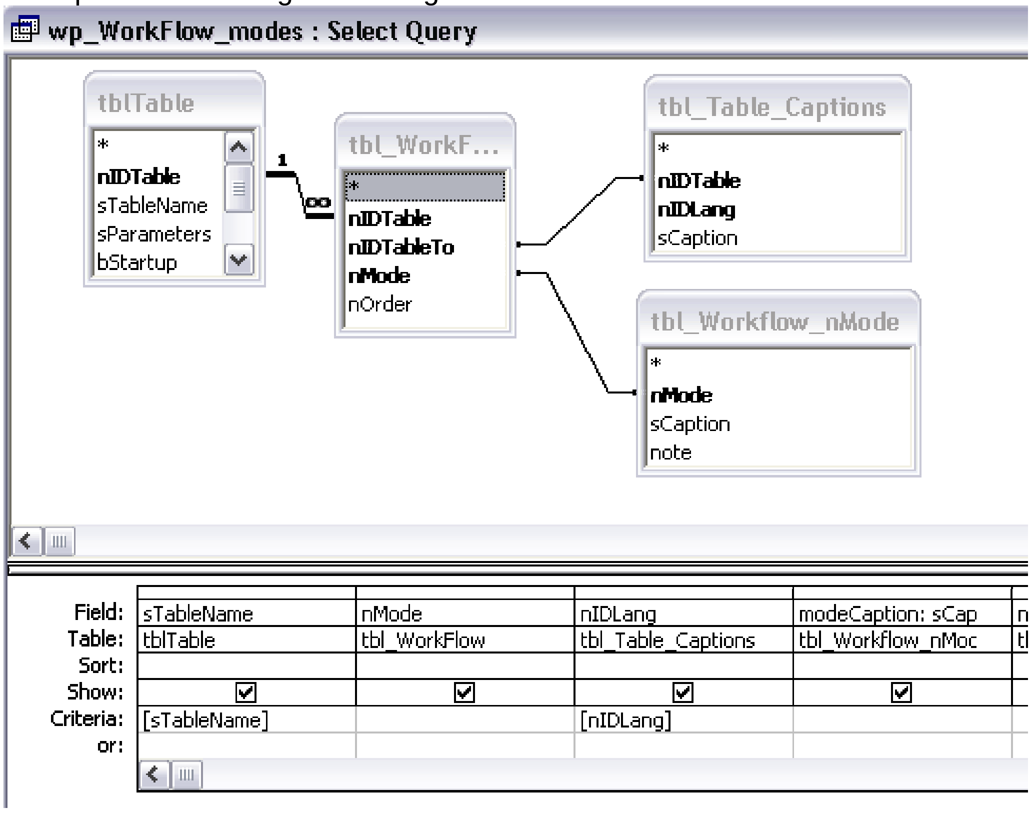


wp_WorkFlow : Select Query

Field:	nIDTable	sTableName: sTable	ToFormType: Form	inParam: sParamete	nMode	sToTableName: sTa	outOptParam: sPar.	bStartup
Table:	tblTable	tblTable	tblTable_1	tblTable	tbl_WorkFlow	tblTable_1	tblTable_1	tblTable
Sort:	Ascending				Ascending			Ascending
Show:	✓	✓	✓	✓	✓	✓	✓	✓
Criteria:		[sTableName]			[nMode]			
or:								

3. “wp_workflow_Modes”

Extracts the acceptable entries for the current form. It defines towards which forms is possible to navigate starting from the current one.



Field:	sTableName	nMode	nIDLang	modeCaption: sCap	n
Table:	tblTable	tbl_WorkFlow	tbl_Table_Captions	tbl_Workflow_nMoc	tl
Sort:					
Show:	✓	✓	✓	✓	
Criteria:	[sTableName]		[nIDLang]		
or:					

BLL therefore should only connect to this repository and read the contents; this is translated in a connection string and the names of the 3 previous queries passed to it as parameters.

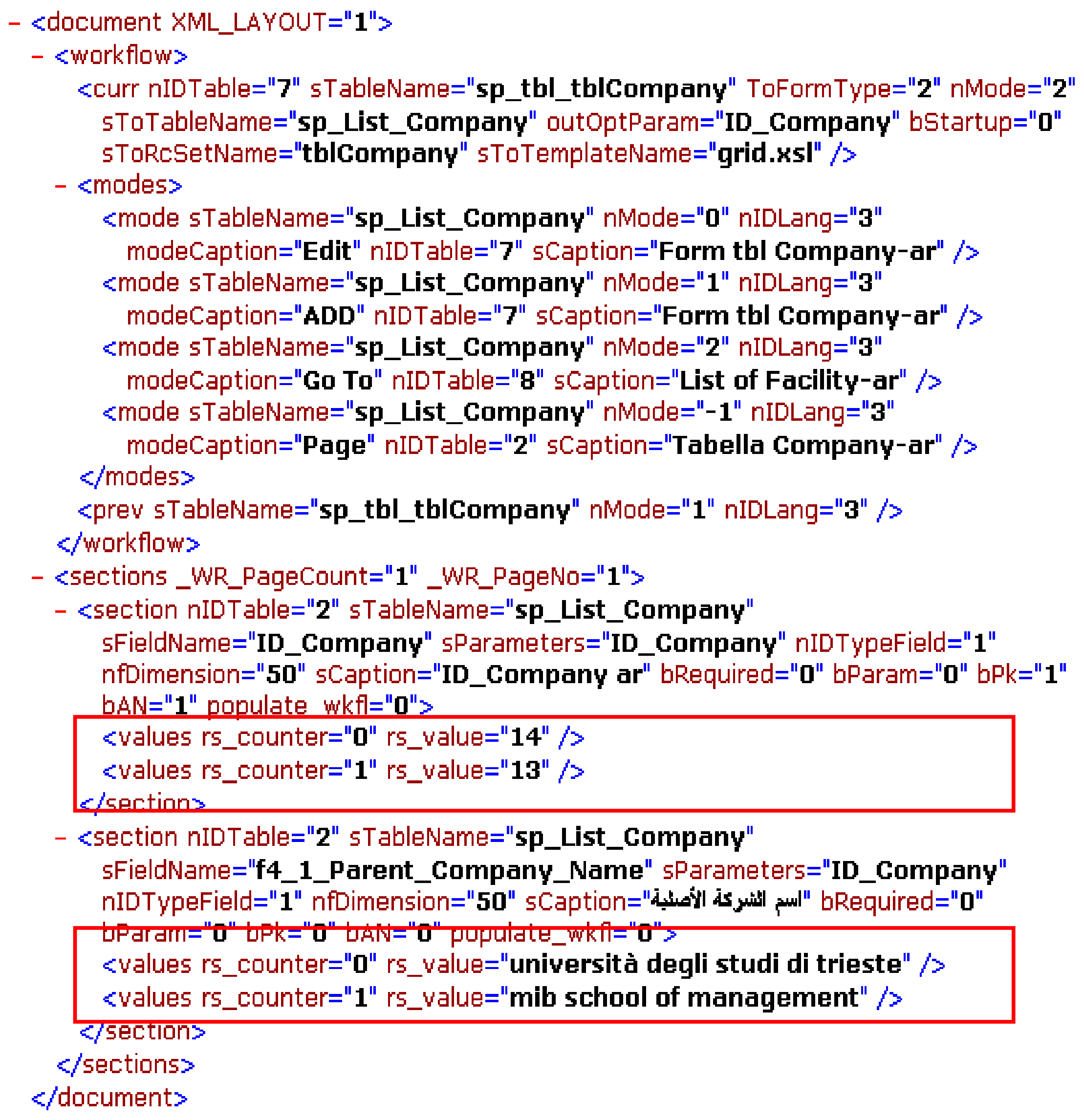
All the configuration parameters obtained by the queries form an XML document to which the field values can be added whenever requested:

1. Grid-form: the fields should be already populated in case the selected recordset is not empty.
2. Single-form: in edit mode.

The Grid-form principally reveals the necessity for a grid population, which means that for every field there is a requirement to insert as many values as the number of records that should be visualized, while the point 2 considers single form only. To overcome such circumstance, it is necessary to generate 2 XML documents: one for the Single-form and another for the grid-form.

Normally, the XML document should contain nodes that serve for the state machine, nodes that define the form, and nodes that contain the field values. The two XML documents are illustrated below:

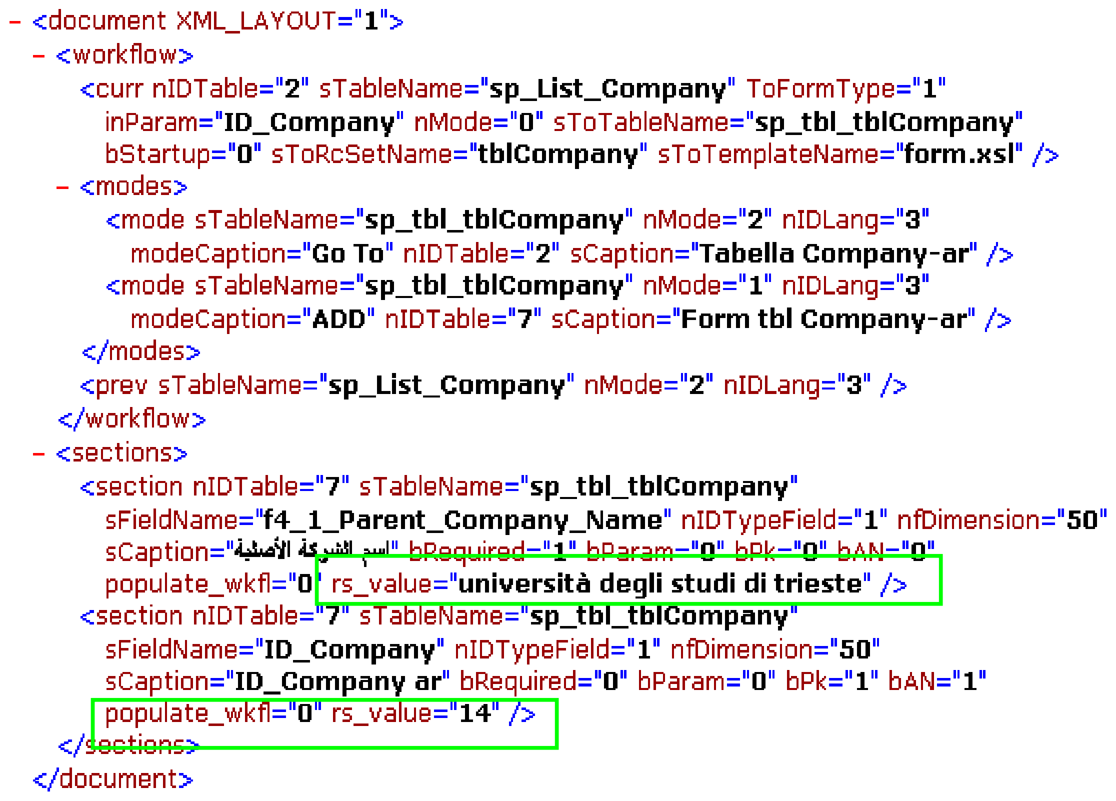
“Grid-form” document:



```
- <document XML_LAYOUT="1">
  - <workflow>
      <curr nIDTable="7" sTableName="sp_tbl_tblCompany" ToFormType="2" nMode="2"
        sToTableName="sp_List_Company" outOptParam="ID_Company" bStartup="0"
        sToRcSetName="tblCompany" sToTemplateName="grid.xsl" />
    - <modes>
        <mode sTableName="sp_List_Company" nMode="0" nIDLang="3"
          modeCaption="Edit" nIDTable="7" sCaption="Form tbl Company-ar" />
        <mode sTableName="sp_List_Company" nMode="1" nIDLang="3"
          modeCaption="ADD" nIDTable="7" sCaption="Form tbl Company-ar" />
        <mode sTableName="sp_List_Company" nMode="2" nIDLang="3"
          modeCaption="Go To" nIDTable="8" sCaption="List of Facility-ar" />
        <mode sTableName="sp_List_Company" nMode="-1" nIDLang="3"
          modeCaption="Page" nIDTable="2" sCaption="Tabella Company-ar" />
      </modes>
      <prev sTableName="sp_tbl_tblCompany" nMode="1" nIDLang="3" />
    </workflow>
  - <sections _WR_PageCount="1" _WR_PageNo="1">
    - <section nIDTable="2" sTableName="sp_List_Company"
        sFieldName="ID_Company" sParameters="ID_Company" nIDTypeField="1"
        nfDimension="50" sCaption="ID_Company ar" bRequired="0" bParam="0" bPk="1"
        bAN="1" populate_wkfl="0">
        <values rs_counter="0" rs_value="14" />
        <values rs_counter="1" rs_value="13" />
      </section>
    - <section nIDTable="2" sTableName="sp_List_Company"
        sFieldName="f4_1_Parent_Company_Name" sParameters="ID_Company"
        nIDTypeField="1" nfDimension="50" sCaption="اسم الشركة الأصلية" bRequired="0"
        bParam="0" bPk="0" bAN="0" populate_wkfl="0">
        <values rs_counter="0" rs_value="università degli studi di trieste" />
        <values rs_counter="1" rs_value="mib school of management" />
      </section>
    </sections>
  </document>
```

Figure 3

“Single-form” document:



```
- <document XML_LAYOUT="1">
  - <workflow>
      <curr nIDTable="2" sTableName="sp_List_Company" ToFormType="1"
        inParam="ID_Company" nMode="0" sToTableName="sp_tbl_tblCompany"
        bStartup="0" sToRcSetName="tblCompany" sToTemplateName="form.xsl" />
    - <modes>
        <mode sTableName="sp_tbl_tblCompany" nMode="2" nIDLang="3"
          modeCaption="Go To" nIDTable="2" sCaption="Tabella Company-ar" />
        <mode sTableName="sp_tbl_tblCompany" nMode="1" nIDLang="3"
          modeCaption="ADD" nIDTable="7" sCaption="Form tbl Company-ar" />
      </modes>
      <prev sTableName="sp_List_Company" nMode="2" nIDLang="3" />
    </workflow>
  - <sections>
      <section nIDTable="7" sTableName="sp_tbl_tblCompany"
        sFieldName="f4_1_Parent_Company_Name" nIDTypeField="1" nfDimension="50"
        sCaption="اسم الشركة الأصلية" bRequired="1" bParam="0" bPk="0" bAN="0"
        populate_wkfl="0" rs_value="università degli studi di trieste" />
      <section nIDTable="7" sTableName="sp_tbl_tblCompany"
        sFieldName="ID_Company" nIDTypeField="1" nfDimension="50"
        sCaption="ID_Company ar" bRequired="0" bParam="0" bPk="1" bAN="1"
        populate_wkfl="0" rs_value="14" />
    </sections>
  </document>
```

Figure 4

The following table denotes the main nodes:

document	/	Document root.
Workflow	Document/workflow	Entails all configura-tions of the workflow and therefore the actual entries, the previous state memo-ry, and the parameters for the form construc-tion.
modes	Document/workflow/modes	Each state (form) allows a certain number of entries, which are enclosed in this node.
mode	Document/workflow/modes/mode	Single entry.
sections	Document/sections	List of nodes with the field definitions.
Section	Document/sections/section	Single field.

The difference between documents is mainly seen at the most basic level. The red frame in *Figure 3* (“grid-form”) depicts the inclusion of the node list “values” in the document: the list is added for each field in the form (and therefore to each node “section”). It memorizes the values of the records in the current recordset. *Figure 4* represents a document in the “single-form”. Each field has an “rs_value” attribute (see the green frames) in which the value of the field in the current recordset is inserted. The current recordset contains in this case one single record.

As previously mentioned, every form has fields of different types: text box, hidden, etc. This is defined in the workflow database by the “nIDTypefield” value in the “tblfields” table (all configuration parameters in the database are listed in the XML document which describes the single form). The XSL style sheet should interpret the meaning of the value, which implies that it must have the templates of the types of the fields interested in the typeField table.

For instance, a field “X” with nIDTypeField =1, that could represent the “textBox” type, should select the template “1” in the style sheet, in order to get as final result a form, in which field “X” is represented by a “textbox”.

Special attention should be paid to the fields that commonly have an obligatory list of values to choose from (like combo and radio buttons). There are 2 types of lists:

1. Fixed list of values;
2. List of values extracted from a table.

Two solutions are given in order to deal with such situations:

Referring to the fixed list of values, it is simply possible to insert the values, separated by a pipe (I), in sDefValue in the Fields table. The XSL style sheet should parse the values later.

Referring to the list of values extracted from a table, let’s suppose a form in which the value of a certain field “X” is chosen from a list of a combo box. Such list presents the result of a query performed on a table. The combo box can be taken as a form with a unique field “X”. The form is furthermore incorporated in the main form and linked through “X”. In this regard, it is possible to use the part of the structure which is created for the form generation, to link the values of “combo” form to the field “X”. Since it is a form, it should be inserted as a record in the tblTable table. This record includes the name of the query to which it is linked, and the field which acts as a form that is supposed to be linked to “X”. In the tblFields table, the value sDefValue of the record corresponding to “X” denotes the name of the form’s query. It is meant that “X” should have the appropriate field type in order to be recognized and correctly configured by the “engine”.

From the above contemplations it can be concluded that the task of the XSL style sheet is to transform everything in HTML form, bearing in mind all mentioned characteristics as well as further features. One of these could be the case in which, when a field is required (bRequired=“1”), the transformation should generate the “client side” scripts to take care of the configuration detail.

Personalisation are “extracted” and located in 2 precise points:

- On the level of BLL, by modifying the database;
- Such modifications should be made only in XSL files and therefore in the PL.

The state machine is transparent to any modification made in the above 2 points.

In conclusion, in the case of the “on-line” version, the site engine should consist of a BLL containing the system for user identification. The choice of the BLL depends on the development technologies and on the state machine illustrated in this paper. In the case of a “light” site version, the BLL should consist of one machine only, as the 2 blocks are virtually independent

PRTR STRUCTURE

PRTR database was designed by ICS-UNIDO team in Alexandria and Trieste within the framework of PRTR project.

In order to determine its structure at the beginning of the venture, the very first steps have been made by modifying an existing database according to a “Reporting” module standard, chosen as a paper device for facility reporting. The Repository has been developed by the Canadian Environmental Protection Act.

An UI stand-alone application has been developed in Visual Basic. Its work is based on Microsoft Access database with the following scheme not showing the relationships:

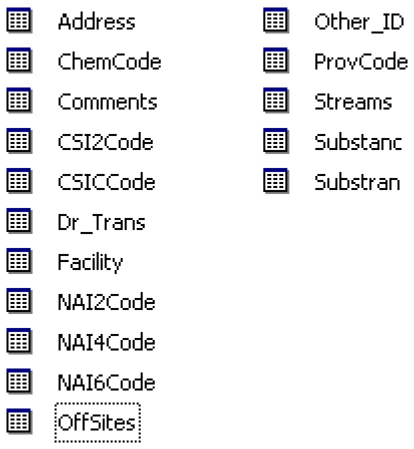

Address
ChemCode
Comments
CSI2Code
CSICCode
Dr_Trans
Facility
NAI2Code
NAI4Code
NAI6Code
OffSites
Other_ID
ProvCode
Streams
Substanc
Substran

Figure 5

However, modifications could not be carried out easily in this case. Therefore, the module has been analyzed and a new relational database has been constructed and tightly correlated to the “Reporting” module. Figure 6 shows the analysis result.

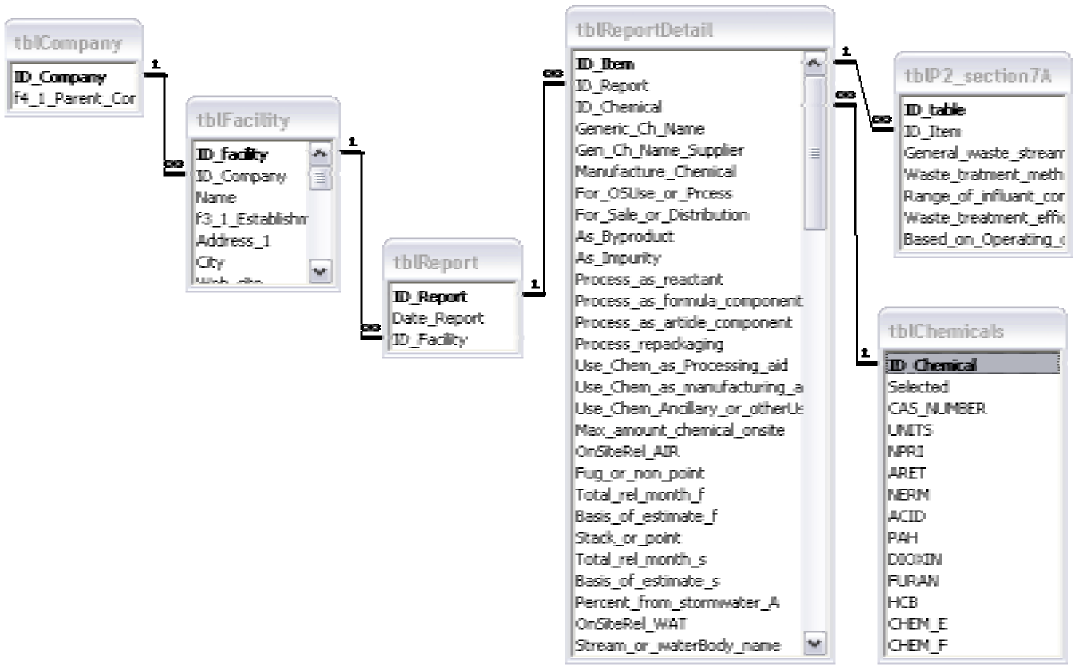


Figure 6

One “company” (tblCompany) can have many “Facilities” (tblFacility) which should be able to compile several reports (tblReport) (monthly). A report can contain many details (tblReportDetail), one for each chemical substance (ID_Chemical obtained from table tblChemicals); every Report detail can have several sections 7a (tblP2_section7a represents the recycling methods for every single waste material in the production process).

The table of the chemicals (tblChemicals) is already compiled since, for the sake of data integrity, there is no need for users to insert different names for the same substance.

The scheme shows tables with one-to-many relation (in effect, the relation between tblReport and tblChemicals is many-to-many and it is represented by table tblReportDetail, but since tblChemicals is a read-only table, this many-to-many relation can be omitted). Now it is feasible to apply the state machine mentioned in previous chapter, in order to construct the applications. The database actually corresponds to a simplified version of the “central” database, and the tables for user identification can be eventually added to it.

PLATFORMS

In this section the development platform and the data repository will be considered.

For the stand-alone version itself, the platform is connected to the operating system installed on the computers that lack Internet connection. In a study performed by ICS-UNIDO, it has been shown that for each “Facility” there is at least one PC with Windows operating system installed.

This means that the application should be developed using technologies such as **com** or the new **.net** (DOT NET). Also, the application is based on a dynamic web site, therefore the use of the same technologies is obvious. Consequently, for the “on-line” version, a site based on asp-Com or Asp.net would be necessary, and the web server that should be emulated is thus the IIS Internet Information Server.

Now, the development platform can be chosen.

The .net solution is not considered because of the following reasons:

1. For executing a program developed in this technology, it is necessary to install also the CLR Common Language Runtime. Actually, CLR is functioning on operating systems such as Windows XP and Windows 2000.
2. Even if the technologies interoperability allows the integration with com objects, the performance of such system is bad, unless a .net object is present in the framework.
3. Being a new technology, **.net** is still not able to guarantee stable functioning.

Our orientation towards the use of **com** technologies is based on several different motivations. **Com** technology today represents the standard for Windows systems, and it is stable. There are RAD tools, such as Visual Basic, that allow the application development in a very simple and fast way. One of the main motives for using **com** technology is the fact that there are many reusable components that can simplify the solution of problems such as:

In the stand-alone solution there are several requirements:

- To emulate the browser; Internet Explorer/version 3.0 offers an ActiveX **(com)** WebBrowser Control, which is in fact a browser itself. This allows the development of applications where the UI is based on DHTML and XML. It can be noticed that there is no similar object **.net** framework.

- To emulate the IIS; this means that the ISAPI Active Server Page filter has to be emulated. These pages can be programmed using a script language such as VBScript or Jscript. The page, before creation, is analyzed by an engine which then executes the code and generates the final result. A Microsoft Script Control **com** component is able to perform the task of a filter. Particularly, this component is an ActiveX Control without user interface and it can be interfaced to any Windows Script interface. Concerning language support, it is possible to choose between 2 scripting languages (VBScript and Javascript). The component allows the adjustment of the other components to the namespace of the script through the use of AddObj method.

In the site engine construction:

- XML support; the site engine should be able to generate the forms described in XML documents. Microsoft released version 4 of its XML sdk. Inside sdk there is a set of components which support XML parsing and XSLT transformations;
- Connections to data sources: the forms and the XML documents are the result of processing of the recordsets extracted from the db. The technology that provides such connections is ADO, and it is also based on **com**.

As a matter of fact, **.net** supports natively both XML and Database connecting through ADO.net, and therefore it is equivalent to the **com** technology. The **com** technology is chosen because of the problems in the development of the “stand-alone” version, particularly in the implementation of “web-client”.

Special attention should be paid to the use of RDBMS for both implementations: in stand-alone version the database should be installed to each single computer. In a Windows-based environment, the most used management platform is Microsoft Access. It offers an SQL engine, the possibility to construct forms and reports, and everything is contained in one single file. Instruments such as “Microsoft upsizing Tools” enable copying and updating of the structure and the data from “Access” database format directly to SQL Server. SQL Server version 2000 includes an analysis tool named “Analysis Service” that allows the development of “Data Warehouse” which is useful for the report generation. These simple considerations guided to the choice of repositories: the database in “Microsoft Access” format should be used for the “stand-alone” version, and the “Microsoft SQL Server 2000” should be utilized for the “on-line” version.

In conclusion, the following tools and components are to be selected:

- Microsoft Visual Basic as the development environment, because of its RAD technology and its simple integration with **com**;
- WebBrowser Control in order to emulate a browser;
- Microsoft Script Control in order to emulate the script engine within ISAPI Active Server Page filter;
- XML 4 framework;
- ADO for database access.

Results of the implementation can be seen in the White paper “Alexandria PRTR Project: Database development” specifically under the paragraph “The user Interface”. More details and source code of the program developed are available from ICS upon request.