

WireLess Mobile V1.x **AOBE** **Applicative** **Transitions**



WireLess**Mobile**

Relative documents :

- [WireLess Mobile 1.0 Channel Designer User's Manual](#)
- WireLess Mobile - Sample Channel Description

1	Introduction	3
2	Solution Architecture Design.....	3
2.1	The Applicative Transitions Model	3
2.2	A complete example : the List Selector.....	4
2.3	Resources Storage	4
2.4	Notation for Diagramming Applicative Transitions Systems.....	5
3	Standard Mechanisms Design	6
3.1	Applicative transitions.....	6
3.1.1	The "call://" and "exec://" protocols	6
3.1.2	The "retpage" and "go" arguments	7
3.1.3	Examples.....	7
3.1.4	Standard Applicative Transitions.....	8
3.2	HTML pages generation.....	9
3.2.1	HTML Templates	9
3.2.2	HTML Templates Interpretation	9
3.2.3	Templates Sections, CHtmlPageMaker and Applicative Transitions	10
3.3	Database	11
3.3.1	Schema	11
3.3.2	The DB Query Processor.....	11
3.4	Gobal simplification	13

1 Introduction

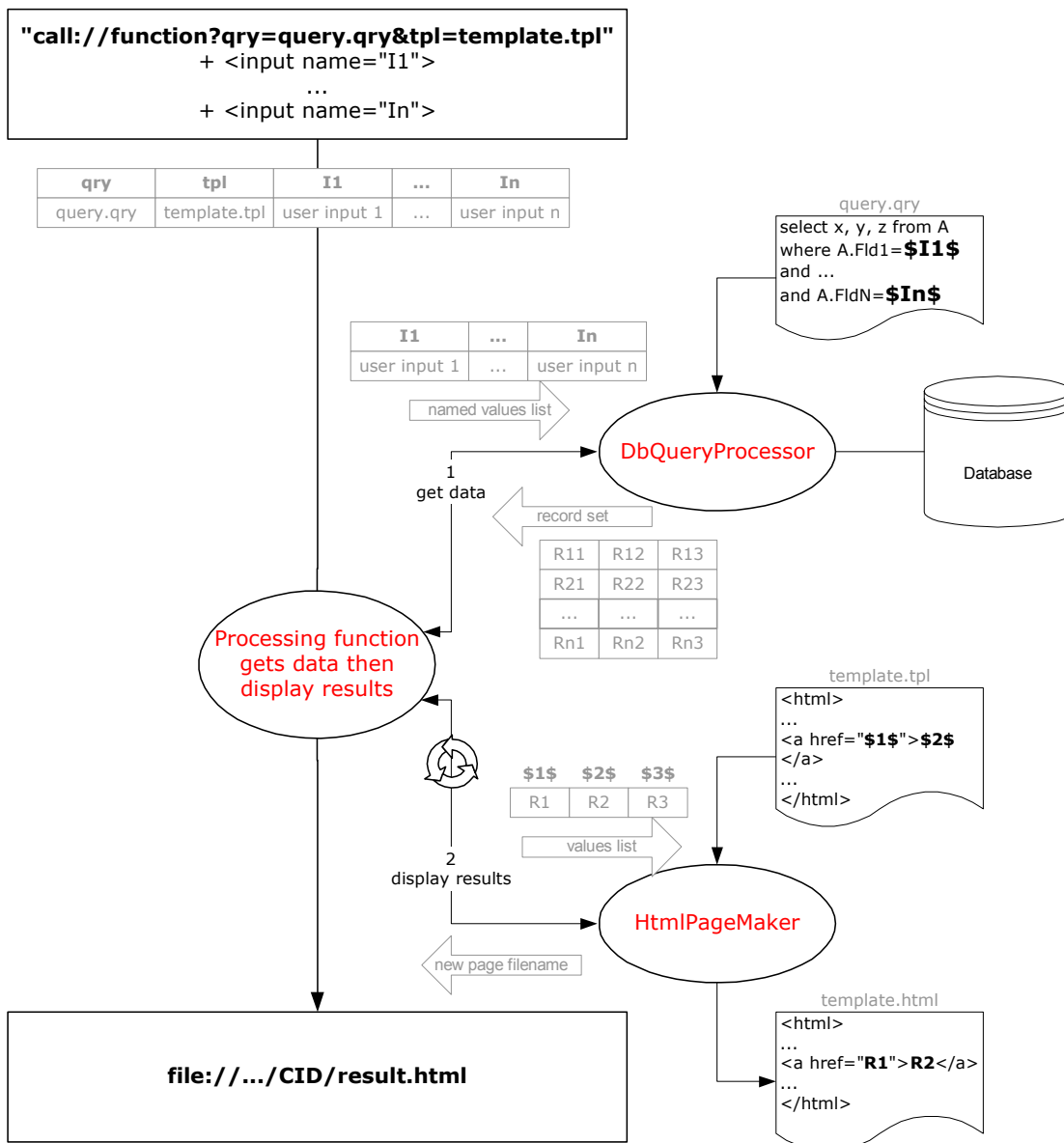
This document describes the AOBE Applicative Transitions technology and framework and its use to implement the Data base support in AOBE for PocketPC.

2 Solution Architecture Design

2.1 The Applicative Transitions Model

The Applicative Transitions Model is the heart of the solution :

- **"call://function"** URLs start the transition. All args and formdata inputs are collected into a list of named arguments and passed to an internal function.
- If the **"qry"** arg is present, the DbQueryProcessor is called. It replaces variables (\$name\$) by values, performs the request and returns a recordset.
- if the **"tpl"** arg is present, an HtmlPageMaker object is initialized with the template file. Its methods are called to generate the output HTML page. Each record from the recordset is translated to a StringList before processing.



2.2 A complete example : the List Selector

Here is how the List Selector looks like :

It is called from a link such as :

```
<a href="call://DisplayList@ qry=SelLst.qry&tpl=SelLst.tpl ">Mes
listes</a>
```

Here is the HTML code of this page :

```
<html><body><font face="verdana">
<font size="3"><p align="center"><b>Select Active List</b></p></font>
<hr width="75%">
<form method="POST"
action="call://SetActLst?qry=ActLst.qry&tpl=ActLst.tpl">
  <input type="radio" name="LSTNAM" value="Liste 1">Liste 1<br>
  <input type="radio" name="LSTNAM" value="Liste 2">Liste 2<br>
  <input type="radio" name="LSTNAM" value="Liste 3">Liste 3<br>
  <center><input type="submit" name="BTN_ACT" value="Select">
</center>
</form>
<hr width="75%">
<a href="CIDHP.html"><font face="Verdana" size="1">Home</font></a>
</font></body></html>
```

When the Select button is clicked -let's assume that the second radio button is checked- this generates the following named arguments list:

Tpl	qry	LSTNAM	BTN_ACT
ActLst.tpl	ActLst.qry	Liste 2	Select

Then the ActLst.qry DB query script file is loaded :

```
SELECT Items.REF, Items.NAM
FROM (ListItems RIGHT JOIN Lists ON ListItems.LSTnum=Lists.num)
LEFT JOIN Items ON ListItems.ITMnum=Items.num
WHERE Lists.NAM='$LSTNAM$';
```

In this situation, \$LSTNAM\$ is replaced by "Liste 2" and the query will select its contents. The result is a record set where each record has 2 components : the item's reference and the item's name.

The final page is generated using the following template :

```
<!--header-->
<html><body><font face="verdana">
<!--title-->
<hr width="75%" align="center"/>
<font size="3"><p align="center"><b>Liste 2</b></p></font>
<!--line:pre-->
<ul><font size="2">
<!--line:do-->
<li><a href="I-Liste 2.html">Item 1</a></li>
<!--line:post-->
</ul></font></ul>
<!--backlinks-->
<br /><hr width="75%">
<a href="CIDHP.html"><font face="Verdana" size="1">Home</font></a>
<!--footer-->
</font></body></html>
```

2.3 Resources Storage

The #RES# directory contains all DB queries and HTML templates plus the Database. <CID> subdirectories are available for channel specific resources.

Search path: <BaseDir>/<CID> → <BaseDir>/#FSUB#<CID> → <BaseDir>/#FSUB#

3 Standard Mechanisms Design

3.1 Applicative transitions

An applicative transition is an HTML navigation (either hyperlink or form submission) that corresponds to an Aladdino specific protocol : call:// or exec:// and is handled and interpreted by the AOBÉ application to perform something specific.

3.1.1 The "call://" and "exec://" protocols

- **"call://function?args_list"** : identifies internal applicative navigations and submissions. This is applicable either in hyperlinks, form actions and buttons:

```
<a href="call://function?arg1=value 1&arg2=value 2">text</a>
<form method="post" action="call://function?arg1=value 1&arg2=value 2">
  ... inputs ...
  <input type="submit" name="call://fnc1?arg1=value+1&arg2=value+2" value="Execute fnc1"/>
</form>
```

- the target URL identifies the internal function to call.
 - If present, the URL's arguments are collected into a Named Arguments List and passed to the referenced internal function.
 - In a form, the inputs' values are added to the arguments list.
 - Also, the called function and its arguments may be changed by naming the submit button "call://function" (URL encoding to take in account). Note that the form's action attribute MUST be either a "call://" or "exec://" one otherwise nothing happens.
 - "call://" navigations always ends up with a regular HTML navigation. The target HTML page may be either online, offline or dynamic.
- **"exec://app.exe cmd-args"**: identifies external applications invocations. This is applicable either in hyperlinks, form actions and buttons:

```
<a href="exec://app.exe /n name /u login /p password">text</a>
<form method="post" action="exec://app.exe /n name /u login /p password ">
  ... inputs ...
  <input type="submit" name="exec://program files\appdir\app.exe" value="Launch App"/>
</form>
```

This is the same as for "call://" except that :

- the target URL is interpreted as an entire command line with arguments and switches and is used to invoke an external application. It is just URL-decoded but not interpreted.
 - Form arguments (if any) are written down into the "exec_args.afd" file in the channels directory (any previous contents overwritten). The invoked application may use the values it contains, but it must be programmed accordingly.
 - Submit inputs can be named "exec://" . Same rules apply as for "call://".
 - "exec://" navigations does not perform any navigation. The external application is launched and AOBÉ looks like it is in a "wait state". Only a dialog box "launch failed" appears in case of error.
- **URL encoding** : in HTML source files, it is not required to URL-encode HREF, action or name attributes values. The only exceptions concern the following characters :
 - " replaced by %22
 - & replaced by %26
 - = replaced by %3D
 - % replaced by %25

3.1.2 The "retpage" and "go" arguments

These special arguments only concern the "call://" protocol.

- **SetRetpage=<qualification>** : the "retpage" argument allows to assert that the page to return "when appropriate" is the current page. That is the page that contains the "call://function?...&retpage=static&..." link. The qualification value may be :
 - **dynamic** : in this case, if the current page is dynamic, then it will be rebuilt upon backward navigation.
 - **static** : in this case, the html page is simply redisplayed and not rebuilt. Note that the name of a dynamic page is the name of its template file with the html extension.
- **go=<pagename>** : the "go" argument allows to navigate to the retpage. It can be used to navigate to other pages in the same channel. Here are the rules :
 - **go overrides tpl** : one cannot navigate to 2 targets at a time...
 - **go=Getretpage** : retpage is the keyword that asks to navigate back to the retpage that has been set by the latest retpage=... encountered directive.
 - **go=<Url_Encoded>** : **Url_Encoded is an AOBÉ Url (xxxx://) who is encoded in order to avoid a confusion with the Url wich contain the "Go" argument.**

3.1.3 Examples

- **"call://" Href**

```
<a href="call://DisplayList?qry=ActLst.qry&tpl=ActLst.tpl">Bookmarks</a>
```

With this URL, the navigation will call the DisplayList function which will perform the DB query specified in ActLst.qry and then display the results using the ActLst.tpl. The "sys-ActLst.html" file will be generated. Note that no real argument is passed to the function.

- **Single "call://" Form**

```
<form method="post" action="call://DisplayList?qry=SearchByName.qry&tpl=SearchItems.tpl">
Search : <input name="PATTERN"/><br>
<center><input type="submit" value="OK"/></center>
</form>
```

This also calls the DisplayList function which will perform the SearchByName DB query and build the resulting Html page using the SearchItems.tpl Html template. Both the DB query and the template are parameterized by the PATTERN argument.

- **Multiple "call://" Form**

```
<form method="post"
action="call://DisplayList?qry=SearchItems.qry&tpl=SearchItems.tpl&SearchField=NAM">
Find <input name="PATTERN"/><br>
<left><input type="submit" name="BTN_ACT" value=" Name"/></left>
<right><input type="submit"
name="call://DisplayList?qry=SearchItems.qry&tpl=SearchItems.tpl&SearchField=OTHER"
value="Princip."/></right>
</form>
```

This form corresponds to the Channel Home Page's Find Form that allows to search items by name or by principle (stored in the OTHER DB field). In this example, for both buttons, the same function will be called with the same script (SearchItems.qry) and HTML template. The only difference is the value of the **SearchField** argument. The DB query script is supposed to use this argument (and also the PATTERN argument of course).

Note that clicking "Princip." button will result in the following formdata argument :

```
call%3A%2F%2FDisplayList1%3Fqry%3DSearchItems%2Eqry%26tpl%3DSearchItems%2Etpl%26SearchField%3DOTHER=Princip%2E,
```

the name of the button has to be URL-decoded back to :

```
call://DisplayList?qry=SearchItems.qry&tpl=SearchItems.tpl&SearchField=OTHER
```

- **"call://" using retpage and go arguments**

Here is the link from the ActLst.html page :

[Change List](call://DisplayList?qry=ActLst.qry&tpl=ActLst.tpl&SetRetpage=dynamic)

And here is the generated List Selector page's form :

```
<form method="post" action="call://DoAction?qry=SetActLst.qry&go=GetRetpage ">
<input type="radio" name="LSTNAM" value="Liste 1"/> Liste 1<br>
...
<input type="radio" name="LSTNAM" value="Liste N"/> Liste N<br>
<left><input type="submit" name="BTN_ACT" value="Select List"/></left>
</form>
```

The first link tells that one wants to return to the current page "when appropriate", and the second one (form's action) tells that after performing "DoAction" this will be the appropriate time to return to the retpage.

As asserted in the first link (SetRetpage=dynamic), the page will be rebuilt upon return.

- **"exec://" navigation**

```
<form method="post" action="exec://app.exe /n name /u login /p password">
<input type="checkbox" name="CHECK1">choix 1</input><br>
Précision<input name="TED1" size="14"/></br>
<center><input type="submit" value="OK"/></center>
</form>
```

Upon validation of the OK button in the form above (we assume it is located in the CID1 directory), Aladdino will create the <basedir>\CID1\exec_args.afd file containing the following two lines :

```
CHECK1=choix 1
TED1=input text
```

and then invoke : app.exe /n name /u login /p password. This is URL-decoded from : exec://app.exe%20%2Fn%20name%20%2Fu%20login%20%2Fp%20password

3.1.4 Standard Applicative Transitions

These functions are described more in details in the CappMgr class (implementation section).

- **DisplayList** : this function is made to display the result of a select DB query.
- **ConfirmAction** : this function is made to display a confirmation page before or after a create/insert/delete DB request.
- **DoAction** : this function is made to perform a query that will result in the redisplay of an existing page. Therefore, the "go" argument will have to be used.

3.1.5 Writing Querys on Win32 and WinCE

Because of ADOCE (WinCE) and ADO (Win32) need different syntaxes, Querys must not be exactly the same on WinCE and win32.

Each string must be delimited by "" on WinCE and by ' ' on Win32.

Numbers doesn't have to be delimited.

HTML pages generation

The ideal way to generate HTML in a flexible manner is to use XSL style sheets. But this is not possible in our context as we do not have any XML documents yet. This is why we introduce Aladdino HTML Templates.

3.1.6 HTML Templates

The HTML templates are a privileged way to generate HTML pages for AOBÉ and other Aladdino WinCE software.



Here are 2 examples of the same list with two different looks.



and here are their corresponding HTML templates

<pre><!--header--> <html> <body> <!--image--> <p align="center"></p></pre>	<pre><!--header--> <html> <body> <!--image--> <p align="center"></p> <!--title--></pre>
<pre><!--line:post--> <!--backlinks-->
<hr width="75%"> Home <!--footer--> </body></html></pre>	<pre><!--line:post--> </table></pre>
	<pre><!--backlinks-->
<hr width="75%"> Home <!--footer--> </body></html></pre>

3.1.7 HTML Templates Interpretation

An HTML template is a viewable HTML document that is internally split into tagged sections that have the following profile : **<!--function:step -->**

- **"function"** is the name of the HTML formatting function that is concerned by this section of the template.
- **"step"** is one of the 3 possible values :
 - pre : initiates a sequence of successive calls to the formatting function.
 - do : implements the formatting call itself
 - post : terminates a sequence of successive calls.

The step may be omitted, its default value is "do"

Precisions :

- the Pre- and Post-steps are **not mandatory**.

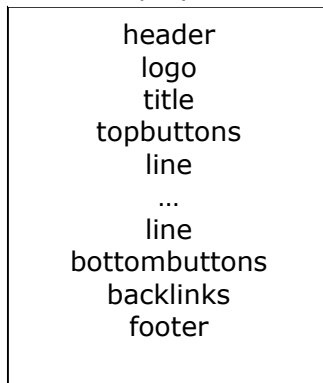
- the pre- and post-steps are **automatically executed**. Pre: the first time the formatting function is called and post: since any other formatting function is called or when the document is closed (footer).
- When processing a section, all occurrences of \$name\$ variables are replaced by their corresponding value as found in the passed Named Arguments list.
- For ease of use, several "**system variables**" are available. Here is the list :
 - **\$prvpage\$** : URL of the previous page
 - **\$curpage\$** : URL of the current page
 - **\$cid\$** : channel ID (for programming)
 - **\$itemref\$** : current item's reference (if applicable)
 - **\$prvitemref\$** : previous item's reference (if applicable)

3.1.8 Templates Sections, CHtmlPageMaker and Applicative Transitions

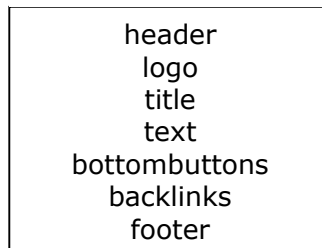
When the system processes a "call:/" applicative transition, it is lead to calling one of the DisplayList, ConfirmAction or DoAction internal functions.

The DisplayList and ConfirmAction functions call the CHtmlPageMaker section output methods in a particular order as illustrated by the following schemes. Note that an individual template may not implement all of the sections. The only mandatory sections are the header and footer.

DisplayList



ConfirmAction

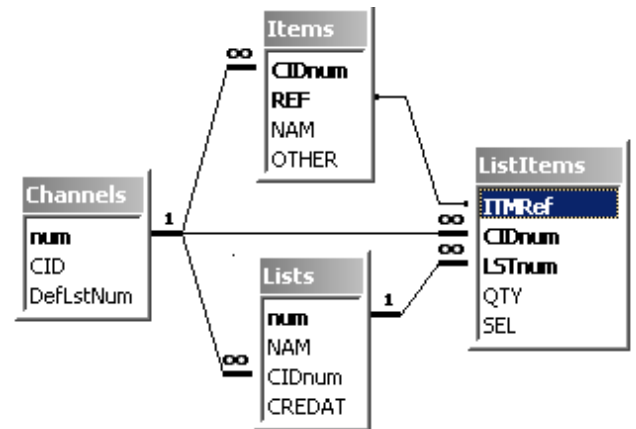


3.2 Database

3.2.1 Schema

The schema is made of 4 tables. All items & lists are stored together regardless of the channel they belong to. It is designed to be stable, flexible and to save space.

- **Channels** : all channels installed on the PDA. Must be updated together with the CHNINFO text file when adding and removing channels. For removal, DB constraints must be satisfied (other tables to cleanup first).
 - **num** : automatic number, primary key.
 - **CID** : channel ID, that is the internal name of the channel.
- **Items** : all items from all channels.
 - **num** : automatic number, primary key.
 - **CIDnum** : pointer to the channel the item belongs to.
 - **REF** : Item's reference. 64 characters max.
 - **NAM** : Item's name. 255 characters max.
 - **OTHER** : complementary field. Likely the FNC. Any additional information we'd like to use for item searching.
- **Lists** : all list names (from all channels)
 - **num** : automatic number, primary key.
 - **NAM** : list name, 64 characters max.
 - **CIDnum** : pointer to the channel the list belongs to.
 - **CREDAT** : list's creation date
- **ListItems** : the contents of all the lists. A record is an occurrence of an item in a particular list.
 - **ITMnum** : pointer to the item
 - **LSTnum** : pointer to the list this occurrence belongs to
 - **QTY** : quantity to order (an integer, for future use)
 - **SEL** : Boolean that tells if the item is selected in this list.



3.2.2 The DB Query Processor

The DB Query processor is relatively simple. Basically the sequence of operations is :

- load the SQL query file and scans its contents in order to get a list of sections introduced by a `.#.SQL` (SQL query) or `.#.EVAL` (variables valuation) line.
- opens the Aladdino Database
- For each section in sequence
 - if `.#.SQL` :
 - all `$<name>$` variables are replaced by the corresponding value in the named values list
 - the SQL queries are executed. The resulting record set is kept for further processing.
 - if `.#.EVAL` :
 - each `(SET <varname> $<n>$)` line is processed in order to complement the named args list with the new `(<varname>,<n>th` value in current record). If `<varname>` already exists, it is updated.

- each (DBUPDATE \$<n>\$ <varname>) line is processed in order to updates the <n>th value of the recordset with the <varname> value from the named args list.
- each (INCREMENT <varname>) line is processed in order to increment the <varname> value of the named args list.
- each (EXITIF <varname> <operatortag> <testvalue>) line is processed in order to evaluate boolean expression. If evaluation returns TRUE, script processing is interrupted.
 <varname> corresponds to any variables from the CNamesArgs collection. A specific <varname> value is managed : NBRESULT. It corresponds to the number of records in the resulted recordset.
 <operatortag> corresponds to any following arithmetical operator :
 - EQ corresponding to ==
 - NE corresponding to !=
 - GE corresponding to >=
 - GT corresponding to >
 - LE corresponding to <=
 - LT corresponding to <
 <testvalue> corresponds to either an integer or a string value. In case of string, comparison is case unsensitive.
NOTE: The NBRESULT variable is unavailable. It always returns a null value.

- returns the recordset and a completion status.

The great thing about the DB Query Processor is that it allows to adapt to any DB schema. With the same application exe, you just change the queries text and its done...
 The **only constraint** is that all variables mentioned in the query's script are actually defined by the source HTML page, either as arguments or as form inputs.

The following script file lists the contents of a List given its name. It contains only one section with no introduction line and is therefore implicitly a SQL query :

```
SELECT Items.REF, Items.NAM
FROM (ListItems RIGHT JOIN Lists ON ListItems.LSTnum=Lists.num) LEFT JOIN
Items ON ListItems.ITMnum=Items.num
WHERE Lists.NAM="$LSTNAM$"
```

This one creates a new list :

```

.#.SQL
SELECT Channels.num FROM Lists INNER JOIN Channels ON
Lists.CIDnum=Channels.num WHERE Channels.CID="$CID$" AND
Lists.NAM="$NEWLST$"
.#.EVAL
(EXITIF NBRESULT NE 0)
.#.SQL
SELECT Channels.num FROM Channels WHERE Channels.CID="$CID$"
.#.EVAL
(SET CIDNUM $1$)
.#.SQL
SELECT num FROM Lists ORDER BY num DESC
.#.EVAL
(SET LSTNUM $1$)
(INCREMENT LSTNUM)
.#.SQL
INSERT INTO Lists (num, NAM, CIDnum, CREDAT)
VALUES ($LSTNUM$, "$NEWLST$", $CIDNUM$, '10/10/2001')
```

This one updates the default list for a channel :

```

.#.SQL
```

```

SELECT Lists.num FROM Lists INNER JOIN Channels ON
Lists.CIDNum=Channels.num WHERE NAM="$SELLST$" AND CID="$CID$" ORDER BY
Lists.num
.#.EVAL
(SET LSTNUM $1$)
.#.SQL
SELECT DefLstNum FROM Channels WHERE CID="$CID$"
.#.EVAL
(DBUPDATE $1$ LSTNUM)

```

3.3 Goba! simplification

The "SWITCH to internal function" is a classical switch to one of the legal internal functions :

DisplayList uses a record-set for FindItems, DisplayActLst, DisplayListSelector

ConfirmAction only uses the first record of the record set (if any) for ConfirmDeleteList, AddItemInList.

DoAction does not use CHtmlPageMaker object. It is supposed to navigate to a specified URL (retpage or else).

