WebX

User's Guide

(dim)

v.2.0

April 1999

Preface

This software I've developed for my personal pleasure. It's a compilation of some my ideas which I wanted to see realized and used in Dynamic Web and Data Base developing. The goal of this product is to make our life more easy :) I say it every time when I develop something, but is it true for WebX − it's up to you...

So, I always wanted to have a kind of program which can be started with same interfaces on my Web server/browser and on my screen in shell mode. Second, I want to avoid any problems with design and result presentation of this program. The most simple and popular language today is HTML, and it's always more sample (in idea) to add several dynamic commands into existing page, than to put the whole page description into the program code. That's why I've decided to write a program capable to add an HTML extension into already existing or completely new pages, to have more softness in development.

And today you can find a lot of product with the same idea and sometimes doing the same thing... So, what is different? REALIZATION! :−) I propose you my realization, which is started from the begin of 1995 and continues its life. For the moment I've ported it on INFORMIX, ORACLE, MySQL and PostgreSQL, of course it turns on any UNIX, no comments for NT. :))

Installation

If you've downloaded the full binary pre−installed version you have just to follow the included installation notes, install the product and skip this section. In other way, if you want to personalize your installation or modify it − don't skip.

So, before to do anything you have to be sure what do you want:

- if you want to use WebX on your Web server you need first to have HTTP server already installed;
- if you want to have an access to the database via WebX you need to have already installed Server or Client software for this RDBMS;
- you have nothing to do if you want to use WebX as a shell tool, why not...

Now you can install WebX:

- First, you need to create /WebX directory

```
$ mkdir /WebX
$ cd /WebX
```

- Second, you have to create your environment file named "x.env" and which contains all environment values you wish to be useful to set during WebX execution. The format of this file is very sample: each line represent a value setting follows syntax NAME=VALUE.

For example:

```
$ cat x.env
PATH=/usr/bin:/sbin:/informix/bin:/oracle/bin
TMP_DIR=/tmp
INFORMIXDIR=/informix
INFORMIXSERVER=ol_goldgate
ORACLE_HOME=/oracle
ORACLE_SID=dim
MYSQL_HOST=goldgate

$
```

Note: the TMP_DIR is the reserved name and it's used to specify the WebX temporary directory. The default value is "/tmp".

- The last you need to create your WebX config file named "x.config". Each line in this file represents the WebX instance configuration for one HTTP server. The line format is follow:

  ServerName:Port RootDirectory DefaultDataBaseName [hold]

- ServerName and Port − your HTTP server name and its port number. For using WebX in shell mode you have to put here shell:00
- RootDirectory − the name of the root directory for WebX script files, so only files from this directory and its subdirectories can be executed for the current server configuration. As WebX is a kind of page interpreter this restriction protect your site against "hacking" interpretations of any other files from your hard disk (like /etc/passwd for example).
- DefaultDataBaseName − the default database name uses for any database connection, can be replaced by "−" if the database name is taken from environment variable (like ORACLE_SID)
- hold − is optional value that protect the database name changement for the current server configuration.

For example:

```
$ cat x.config
goldgate:80 /WebX/goldgate −
goldgate:81 /WebX/admin db_admin hold
shell:00 /WebX/shell −
$
```

Note: you can change the default "/WebX" directory by setting the **X_HOME** value in your environment for personal use and development.

Now you can copy WebX executable files to your server cgi−bin directory or in /usr/bin for using from shell (WebX − without db connections, WebX.IFX − to communicate with INFORMIX, WebX.ORA − with ORACLE, WebX.pSQL − with PostgreSQL, WebX.mySQL − with MySQL)

First View...

So, every page files you places into the WebX RootDirectory for your server you can execute via standard cgi−bin URL adding your page name:

/cgi−bin/WebX/pagename **or** /cgi−bin/WebX?X_PAGE=/pagename

where X_PAGE is reserved variable for pagename settings.

From shell you can start it as:

```
$WebX pagename
```

Your page will be interpreted by WebX and the output will be placed to your browser or to your screen if you are in shell mode. Any additional parameters can be passed using standard GET or POST method (HTTP server) or as command line parameters in shell mode. For example :

HTTP GET: http://yourhost/cgi−bin/WebX/pagename?name1=value1

HTTP GET: http://youthost/cgi−bin/WebX/pagename?name1=value1&name2=value2

Shell: $WebX pageame −name1 value1 −name2 value2

Within your page you can access to your variables directly or via multiple functions, for example just put $(name1) to show in this place the current value of the name1.

You can use also: $(name $(name2)) , $($(name1)) , $($($..$(name)..))
In case of $($(name1)) will be printed value of the variable which name is a value of name1.

Standard WebX commands

Each line from your page which starts with SPACES or TABS follows with "<@" WebX will try to interpret as a WebX command... If your command is too long you can cut it into several slices ended by back−slash ('\').

This chapter will present the standard set of commands which you can use on any page managed by WebX.

SET

Syntax:

```
<@set> Name value
<@Name>= value
<@Name>++
<@Name>−−
<@Name>+= value
<@Name>−= value
<@Name>*= value
<@Name>/= value
<@Name>%= value
```

Action: Set value to variable Name, if Name doesn't exist it'll be created. Any symbols of set: *{}[]*``~~''^^"" can be used as begin/end marker of the value string. The extensions ++, −−, *etc.* add the set of sample arithmetic operation under variables.

Examples:

```
<@Year>= 1995
<@Name>= ~Dimitri~
<@Name>= "$(Name) (dim)"
<@Year>++
<title> WebX $(Name) (c) $(Year) </title>
```

INCLUDE

Syntax:

```
<@include> pagename
```

Action: Include and interpret by WebX another page. You can have till 10 recursive includes. Every variables are common for all pages, during executing the reserved variable **DIR** keep always the name of the current directory for the current page.

Examples:

```
<@include> $(DIR)/x.include
```

SHELL

Syntax:

<@shell> shell−command

Action: execute a shell command (via system(2)).

Examples:

```
<@shell> env | sort
<@shell> ls −l $(Filename)
```

IF

Syntax:

```
<@if> condition <@WebXcommand> ...

<@if> condition
...
<@end>

<@if> condition
...
<@else>
...
<@end>
```

Action: no comments :−)

Conditions :

icmp X **op** Y − integer comparison between X and Y
cmp X **op** Y − string comparison between X and Y
X **op** Y − string comparison between X and Y

X,Y − Values
op: >,<,>=,<=,==,!=, like, !like
for like you can use any pattern as: "*?[a−z]"

fmt NAME format − check format of NAME value

format: int − check for integer
float − check for float

filled NAME − check variable NAME exists and filled

exist Filename − check than file Filename exist

EOF − check if EOF reached during database search

Note : !icmp, !cmp, !filled, !EOF, !script, !fmt, !exist can be used for **NOT** conditions...

Examples:

```
<@if> $(number) == 0 <@number>= 100
<@if> icmp $(count) < 0 <@count>= 0
...

<@if> !filled Name
<@Name>= UNKNOWN
<@Profile>= "−"
<@end>
...

<@if> $(Name) == ADMIN
<h1> Welcome to the Administration Service </h1>
<@call> welcome
<@else>
<h1> Access Forbidden </h1>
<@end>
```

QUIT

Syntax: <@quit>

Action: stop WebX execution

Examples:

```
<@if> $(MAX) == 1000
<h1> Reached MAX value... </h1>
<@quit>
<@end>
```

ERROR

Syntax:

```
<@error> message
```

Action: stop WebX execution with Error Message. This action will invoke the Error Page execution, the name of this page is stored in the reserved variable **X_ERROR** (default value : x.error). The contents of this page is free, the error message can be shown as **X_ERRMSG** value. If the X_ERROR is not set default format will be used.

Examples :

```
<@if> !fmt MAX int <@error> Invalid MAX value...
<@if> !fmt AGE int <@error> Invalid AGE value...
```

Example of x.error :

```
<h1> ERROR ! ! ! <h1>
<hr>
<h3> During execution the follow error has been occurred : </h3> <br>
<i> $(X_ERRMSG) </i> <br>
<hr>
```

WHILE

Syntax:

```
<@while> [condition]
...
<@break>
...
<@continue>
...
<@end>
```

Action: making a loop :)

Conditions :

<@while> − unlimited loop till <@break> or <@quit>

<@while> **values** NAME − loop for all values of multi−value NAME

<@while> **words** NAME − loop for all words from NAME value separated by SPACE

<@while> **words** NAME **SEP** −loop for all words in NAME value separated by any symbols from string SEP

Note: during a loop for values or words the NAME value will keep one of values or words at time.

Examples:

WHILE: Unlimited loop

```
<@State>= "Ok"

<@while>
...
<@if> $(State) != "Ok" <@break>
...
<@end>
```

WHILE: Multivalues

```
INPUT FORM
...
<form action=/cgi−bin/WebX method=POST>
...
<b> City included for search: </b>
<select name=City size=10 multiple>
<option> Paris
<option> Kiev
<option> St.Petersburg
<option> Moscow
...
</select>
...
<input type=hidden name=X_PAGE value=result_page>
<input type=submit name=GO value=" Start Search ">
...
</form>
```

```
result_page:
...
<@query>= "select * from Country;"
<@Where>= ""
<@OR>= ""

<h3> Search for City(s) :</h3><hr>
<@while> values City

$(City) <br>
<@Where>= "$(Where) $(OR) City = '$(City)' "
<@OR>= "or"

<@end>

<hr>

<@if> filled Where <@query>= "select * from Country where $(Where);"
...
```

Result: if it was selected Kiev and Paris you'll have in "query":
select * from Country where City = 'Kiev' or City = 'Paris';

WHILE : Words

```
<@Count>= "1;2;3;4;5;10;12;56;100"

<@while> words Count ";"
$(Count) ** 2 = $(expr : $(Count) * $(Count)) <br>
<@end>

<@if> filled SearchWords
<@query>= "select * from hotels where keyword in( "
<@comma>= ""

<@while> words SearchWords

<@query>= "$(query)$(comma) '$(SearchWords)' "
<@comma>= ","

<@end>

<@query>= "$(query) );"
<@end>
```

UNLINK

Syntax:

```
<@unlink> Filename
```

Action: delete the file

Examples:

```
<@unlink> /tmp/myfile
```

FILE

Syntax :

```
<@file> Filename
...
<@end>
```

Action: add all WebX output from the <@file> to the <@end> to the file named Filename.

Examples :

```
<@unlink> "/tmp/page.$(No)"
<@file> "/tmp/page.$(No)"
...
<@end>
...

<@include> "/tmp/page.$(No)"
<@unlink> "/tmp/page.$(No)"
```

COOKIE

Syntax:

```
<@cookie> Name
```

Action: get the Netscape Cookie value to the WebX variable (with the same name)

Note : the feature of the setting a cookie value is inactivated for the moment

Examples:

```
<html>
...
<@cookie> USER_ID

<@if> !filled USER_ID
<@include> ./login.html
<@quit>
<@end>

<@include> ./welcome−$(USER_ID).html
</html>
```

FUNCTION & CALL

Syntax :

```
<@function> FunctionName
...
<@return>
...
<@return> $(value)
...
<@end>


<@function> FunctionName( parameters )
...
<@end>


<@call> FunctionName
<@call> FunctionName( parameters )

$(call:FunctionName)
$(call:FunctionName( parameters ))
```

Action: function using

Note : function can be recursive

Examples:

```
<@function> Power2( val )
<@return> $(expr: $(val) * $(val) )
<@end>

<@function> Fact( val )

<@if> val == 1 <@return> 1
<@return> $(expr: $(val) * $(call:Fact( $(expr: $(val) −1) ) ) )

<@end>

<@function> Reset( v1,v2,v3 )

<@set> $(v1) 0
<@set> $(v2) 1
<@set> $(v3) 2

<@end>
...
```

```
$(numb) ** 2 = $(call:Power2( $(numb) )
$(numb)! = $(call:Fact( $(numb) )

Old values : A= $(A), B= $(B), C= $(C)
<@call> Reset( A, B, C )
New values : A= $(A), B= $(B), C= $(C)
```

REPLACE

<u>Syntax:</u>

<@replace> Name OldString NewString

<u>Action:</u> replace all parts of the old text by the new one in the Name value

<u>Examples:</u>

```
<@replace> Name "'" "''"
<@replace> Address "'" "''"

<@query>= "insert into Client( Name, Address ) values ( `$(Name)', `$(Address)' ) ;"
<@replace> Country "FR" "France"
```

FILE I/O

<u>Syntax:</u>

<@fopen> fpID Filename mode state − open a file
<@fgets> fpID ValName size state − read a line from file
<@fputs> fpID Value state − write a line to file
<@fclose> fpID state − close an opened file

<u>Action:</u> File I/O interface

<u>fpID</u> − any unique name uses to name file descriptor (*stdin*, *stdout* and *stderr* are reserved names for std−I/O)
<u>mode</u> − any legal fopen(3C) value ("w", "r", "a", etc.)
<u>state</u> − name of value where will be stored the operation status

<u>Examples:</u>

```
<@fopen> fpIN "input.txt" "r" sst
<@if> $(sst) != 0 <@error> Cannot open input file...
<@fopen> fpOUT "$(outputFilename)" "w" sst
<@if> $(sst) != 0 <@error> Cannot open output file: $(outputFilename)

<@while>

<@fgets> fpIN buff 200 sst
<@if> $(sst) != 0 <@break>
<@if> "$(buff)" like "*Comment*" <@continue>
<@replace> buff "Windows" "UNIX"
<@fputs> fpOUT "$(buff)$(\n)" sst


<@end>

<@fclose> fpOUT
<@fclose> fpIN
```

PROCESS I/O

Syntax:

```
<@popen> ppID Command mode state − open a process stdin/stdout
<@fgets> ppID ValName size state − read a line from process stdout
<@fputs> ppID Value state − write a line to process stdin
<@pclose> ppID state − close an opened process
```

Action: Process I/O interface

ppID − any unique name uses to name file descriptor
mode − any legal popen(3C) value ("w" or "r")
state − name of value where will be stored the operation status

Examples:

```
<@popen> pp "ls −l" "r" sst
<@if> $(sst) != 0 <@error> Cannot open "ls" command...

FILE SIZE OWNER
=================================
<@while>

<@fgets> pp buff 200 sst
<@if> $(sst) != 0 <@break>
<@if> $(word:buff:1) like "d*" <@fputs> stdout "/"
$(word:buff:9) $(word:buff:5) $(word:buff:3)

<@end>
=================================

<@pclose> pp
```

SLEEP

Syntax:

```
<@sleep> sec
```

Action: suspend execution for a number of seconds (sleep :))

Examples:

```
***tail of the file: $(filename)

<@fopen> fp "$(filename)" "r" sst
<@if> $(sst) != 0 <@error> Cannot open file: "$(filename)" ...

<@while>

<@fgets> fp buff 300 sst
<@if> $(sst) != 0
<@sleep> 1
<@continue>
<@end>
$(buff)

<@end>
```

CHDIR

Syntax:

<@chdir> directory

Action: change the current directory

Note: when WebX started the current directory is changed to the root directory for the current config (see x.config)

Examples:

```
<@chdir> $(DIR)

<@file> output.txt
...
<@end>
```

COMPILE

Syntax:

<@compile> ON/OFF

Action: set on/off value translation (default − ON)

Examples:

```
<@name>= "Unknown"
$(name) −> Unknown
<@compile> OFF
$(name) −> $(name)
```

HTML

Syntax:

<@html> ON/OFF

Action: set on/off translation accentuated letters to the HTML equivalents (default ON)

Examples:

```
<@name>= "Hélène"
$(name) −> H&acute;l&egrave;ne
<@html> OFF
$(name) −> Hélène
```

COMMENTS

Syntax:

<@−− Any text −−@>

Action: put some comments :))

Examples:

```
<@−− Follow lines are comments
and next command will never work:

<@shell> rm −rf *

but it's very dangerous to put it after end of comments... −−@>
```

In−Line Values and Functions

today − reserved variable with the current date in format YYYY−MM−DD

Examples:

```
Today is $(today).
<@set> query "select * from News where NEWS_DATE = '$(today)';"
```

X_DT − reserved variable with the current datetime in seconds since 1970 (UNIX time)

Examples:

```
Last news for the last hour...
<@query>= "select * from News where TIMESTAMP between $(X_DT)−360 and $(X_DT);"
```

dt2str:Name:Format, str2dt:Name:Format − internal functions to convert UNIX time to string and back, Format can be represented by any character pattern contains DD, MM, HH, etc.

Examples:

```
Now is $(dt2str:X_DT:hh:mi:ss dd−mm−yyyy).
<@fputs> stdout "Enter date format: "
<@fgets> stdin Format 100
<@fputs> stdout "Enter date: "
<@fgets> stdin Date 100

<@query>= "update News set TIMESTAMP= $(str2dt:Date:$(Format));"
```

cut:len:NAME − function to cut the value of NAME variable to len characters (it can protect your code if at place of 10 letters user sends to you 10.000 ...)

Examples:

```
<@Name>= "$(cut:20:Name)"
<@Address>= "$(cut:60:Address)"
```

len:NAME − function to give the length of the NAME value

Examples:

```
<@if> icmp $(len:Name) > 20

WARNING : The entered Name is longer than 20 characters, so it'll be truncated <br>

<@end>

<@Name>= "$(cut:20:Name)"
```

str:begin:end:NAME − function to get the substring from the NAME value

Examples:

```
<@fgets> fp buff 200 sst
<@query>= "insert into address( Street, No, City ) \
values( '$(str:1:40:buff)', $(str:41:45:buff), '$(str:46:60:buff)' );"
```

word:NAME:no − function to get the word number no from the NAME value

Examples:

```
<@popen> pp "ls −l" "r" sst
<@if> $(sst) != 0 <@error> Cannot open "ls" command...

FILE SIZE OWNER
================================
<@while>

<@fgets> pp buff 200 sst
<@if> $(sst) != 0 <@break>
<@if> $(word:buff:1) like "d*" <@fputs> stdout "/"
$(word:buff:9) $(word:buff:5) $(word:buff:3)

<@end>
================================

<@pclose> pp
```

trim:NAME − function to drop insignificant spaces in the NAME value

Examples:

```
Title = $(Title) ==> " Hello, Mister Dr. Brown!!! "
trim Title = $(trim:Title) ==> "Hello, Mister Dr. Brown !!!"
```

crypt:NAME_val:NAME_passwd − function to crypt/uncrypt any value by password

Examples:

```
<@passwd>= "Very secret password!"
Value: "$(value)" is crypted to: "$(crypt:value:passwd)"...
<@value>= "$(crypt:value:passwd)"
...
Crypted Value: "$(value)" is uncrypted to: "$(crypt:value:passwd)"...
...

<@cookie> Login
<@if> !exist /home/$(crypt:Login:passwd)/welcome.html
<@error> "Wrong User Name!"
<@end>

<@include> /home/$(crypt:Login:passwd)/welcome.html
```

txt2url:NAME, url2txt:NAME − functions to convert text to URL format and back

Examples:

```
Title = $(Title) ==> Story of L'Oreal
URL Title = $(txt2url:Title) ==> Story%20of%20L%27Oreal
```

so the correct URL is :

```
<a href=/cgi−bin/WebX/x.search?Title=$(txt2url:Link)> $(Title) </a>
```

uppercase:NAME, lowercase:NAME − functions to convert text to upper/lower case

Examples:

```
<@if> « $(uppercase:City) » == « PARIS » <@call> parisLINE

...

<@call> $(lowercase:City)LINE
```

\n, \r, \t − the reserved values to print the <NL>, <CR>, <Tab>

Examples:

```
Result : $(\t)$(number) $(\t)$(score) $(\n)
<@replace> String `$(\t)` ` `
<@replace> Text `$(\n)` `<br>`
$(Text)
```

sh:NAME − function to escape shell command from the NAME/value (security reasons)

Examples:

```
if $(Name) is: "nobody; cd /; rm −rf *"
(sh:Name) is: "nobody\; cd /\; rm −rf \*"
<@shell> /bin/script −login "$(sh:Name)" 2>&1
```

int:format:VALUE − function to present integer value (decimal) as output of C function "printf"

Examples:

```
<@Numb>= 100
Numb (%05d) = $(int:%05d:$(Numb)) <br>
Numb (%05o) = $(int:%05o:$(Numb)) <br>
Numb (%05h) = $(int:%05h:$(Numb)) <br>
```

expr:VALUES − expression − function to calculate arithmetical expression

Examples:

```
<@A>= 1
<@while>
...
<@TOTAL>= $(expr: $(TOTAL) + $(X)/$(Y))
...
<@A>++
<@if> icmp $(A) >= 5 <@break>
<@end>
```

Note: $(expr: 20/3) => 6; $(expr: 20.0/3) => 6.66666...

X_ERRMSG − reserved variable with the last WebX error message

Examples:

```
<@if> $(X_SQLCODE) != 0
<h1> Database Error Message: $(X_ERRMSG) </h1>
<@quit>
<@end>
```

X_ROOT − reserved variable keeps the WebX root directory name for the current config

Examples:

```
<@include> "$(X_ROOT)/Lib/x.draw"
```

DIR − reserved variable keeps the WebX current directory name

Examples:

```
<@include> "$(DIR)/init.html"
<@shell> $(DIR)/../bin/checkuser "$(Login)"
<@chdir> $(DIR)
```

X_PID − reserved variable keeps the WebX process id (pid)

Examples:

```
<@file> "/tmp/.out.$(X_PID)"
```

X_PROG − reserved variable keeps the WebX process name

Examples:

```
<@if> $(X_PROG) like "*.IFX" <@X_BASE>= informix
<@if> $(X_PROG) like "*.ORA" <@X_BASE>= oracle
```

env:NAME − function to take the environment value

Examples:

```
PATH=$(env:PATH) <br>
SCRIPT_NAME=$(env:SCRIPT_NAME) <br>
USER=$(env:REMOTE_USER) <br>
Called URL=$(env:HTTP_REFERER) <br>
```

Database Interface

It's very easy... :)

First, before you run any database queries you need to give values for follows variables:

X_BASE − database name (if you don't use the default database name)
X_USER and **X_PASSWD** − connection user name and password (if you don't use the default (login) connection)

For the immediate execution query you can use:

<@sqlexec> SQLquery

For the query which return some values you can use:

<@sql> SQLquery
...

<@break>
...
<@continue>

...
<@end>

Note: in the second case any query may be executed, even for immediate execution, so if you know nothing about nature of your query (user input, for example) − use second kind of SQL call.

The body between <@sql> − <@end> will be executed if the SQL query is finished correctly. For the SELECT query the body will be executed for every successful FETCH. Values fetched from the database become the normal WebX variables with the same names as the column names in SELECT. The reserved variable **X_sqlnames** keeps a name list of all selected columns (names are separated by **';'**). You can switch the lower (default) to upper case for the column names using:

<@sqlnames> uppercase/lowercase

WebX can keep till 10 (default value) active SQL queries on the same time, so no problem for the recursive selects.

By default, if it takes place some errors during query execution WebX will finish with SQL Error Message. If you prefer to personalize the reaction of your application on errors you can force WebX to ignore any SQL errors via <@sqltrap> call and manage them yourself. The reserved variable **X_SQLCODE** keeps the state after last SQL execution (0 == OK).

<@sqltrap> ON/OFF

Also, if time to time you change the database mode to support or unsupport transactions you can force WebX to ignore any SQL errors during BEGIN/COMMIT/ROLLBACK execution:

<@sqltrap> OFF_TRANSACTIONS/ON_TRANSACTIONS

If you execute INSERT statements using autoincrement/serial fields, the reserverd variable **X_INSERT_ID** will keep the last inserted value (of course if this kind of feature enabled in the database).

Let show some examples with SQL queries...

SELECT

SELECT simple

```
...
<@X_BASE>= dim
<@X_USER>= dim
<@X_PASSWD>= WebX

<!−− skip first 10 records and show next 40 −−!>
```

```
<table border=3>
<tr><th> No. </th><th> Author </th><th> Title </th><th> Year </th></tr>

<@no>= 0

<@sql> select author, title, year from book order by year;

<@no>++
<@if> icmp $(no) < 10 <@continue>
<@if> icmp $(no) > 50 <@break>

<tr><td align=right> $(no). </td>
<td> $(author) </td>
<td> $(title) </td>
<td> $(year) </td>
</tr>

<@end>

</table>
```

<u>SELECT Recursive</u>

```
<!−−− Print a tree from book_catalogue table:
=======================================
[ parent ] [ ref ] [ name ] [ print_date ]
=======================================
Note : parent & ref makes the father/son relation

Like:
0, 1, `INFORMIX User Manual', '1996−09−17'
1, 2, 'Part I', '1996−09−17'
2, 3, 'SQL intro', '1996−09−17'
...
1, 4, 'Part II', '1996−09−17'
1, 5, 'Part III', '1996−09−17'
...
0, 6, 'INFORMIX Administration', '1996−09−17'
−−−>
...
<@function> PrintTree( parentID )
<ul>

<@sql> select * from book_catalogue where parent = $(parentID);

<li> <i> $(name) </i> Modified: $(print_date)
<@call> PrintTree( $(parent) )

<@end>

</ul>
<@end>

...

<h1> The Catalogue Tree </h1>
<hr>
<@call> PrintTree( 0 )
<hr>
...
```

INSERT

```
...
<@sqlexec> insert into book( author, title, year ) values( '$(Author)', '$(Title)', '$(Year)' );
```

```
...
<@sqltrap> OFF
<@sql> insert into CAT_USERS( login, passwd ) values( '$(Login)', '$(crypt:passwd:passwd)' );

User: $(Login) successfully inserted to the database

<@end>
```

UPDATE

```
...

<@sqlexec> update book set author = '$(Author_NEW)' \

where author = '$(Author_OLD)';
```

DELETE

```
...
<@sqlexec> delete from book where author = '$(Author)' ;
```

Other features

There are some other functions and possibilities in WebX which are not documented and not described, like WebX transparently extending by new commands and functions, dynamic code executing (stored in the database, imported to any value, etc.), the file uploading from client to server, Netscape cookie management, etc. I did not describe them here, because I don't use them very often...

Thank you very mach to arrive till the end :−)