Documentation - Content

[HOME] [CONTENT] [NEXT (Operating-Modes)]

[HOME] [CONTENT] [NEXT (Operating-Modes)]

Embperl - Embed Perl in Your HTML Documents

Current Version

`Stable: 1.2.0`

`Beta: comming soon`

DEUTSCH

Embperl gives you the power to embed Perl code in your HTML documents. Using Perl means being able to use a very elaborate programming language, which is widely used for WWW purposes. You can also use hundreds of Perl modules which have already been written - including DBI - for database access to a growing number of database systems.

While this could also be done with other packages, Embperl has several features especially for HTML: dynamic tables, formfield-processing, escaping/unescaping and more.

Embperl is a server-side tool, which means it's browser-indpendent. It can run in various ways: Under mod_perl, as a cgi script, or offline.

If you would like to get an impression of Embperl's features, you can see them here. Read the introduction to get started. If you would like to know more about Embperl, here is the full documentation. A

News

20. Dec 99	Added Introduction to DBIx::Recordset
4. Dec 99	Added Page with WorldWide Mirrors
17. Nov 99	Embperl 1.2.0 released
4. Nov 99	Embperl 1.2b11 released
7. Oct 99	Embperl 1.2b10 released
10. Sept 99	Embperl 1.2b9 released
10. Sept 99	Embperl 1.2b9 released
9. Sept 99	Embperl 1.2b8 released
8. Sept 99	German Embperl Website is online
30. Aug 99	Embperl 1.2b7 released
19. Aug 99	Embperl 1.2b6 released
12. Aug 99	Article in german Unix magazine iX 9/99 about Embperl and DBIx::Recordset
monthly	Perlmonth has a column about Embperl
5. July 99	Embperl 1.2b5 released
1. July 99	On the O'Reilly Open Source Software Convention there are two talks about Embperl Embperl and DBIx::Recordset -- How to Easily Build a Database Driven Web Site New Stuff: Embedding Perl

FAQ list is also available and the history of Changes can also be found here. There are also useful examples which show exactly how Embperl works.

There is also a module for Database access within Embperl, look here for an introduction of DBIx::Recordset

---

Last Update $Id: index.html,v 1.61 1999/12/20 05:05:03 richter Exp $

Main features of HTML::Embperl 1.2

[HOME] [CONTENT] [PREV (Features - Content)]

- Lets you embed Perl code into HTML documents. Perl code is evaluated at the server side and the result is sent to the browser.
- Offers various meta-commands for conditional and loop processing of the HTML documents.
- Automatically generates dynamic HTML tables/lists from Perl arrays or function calls (e.g. DBI fetch)
- Form data send to your document is placed in a hash
- Automatically inserts data from the form hash into HTML input, textarea and select tags
- Understands HTML and URL escaping and unescaping
- Handles per-user and per-module persistent session data for you. All you need to do is store and retrieve them to and from a special hash.
- Allows you to build your web-site out of components. Frequently used elements can be defined once and included in every page.
- Gives you the possibility to build libraries with a set of frequently used Elements or components, and make them available to the whole web-site in the same way as Perl modules are - well, modularized.
- Supports debugging of pages by generating a very detailed log file and making it accessible via the browser with a single click.
- Generates verbose error pages in which every error can link to the log file. This is ideal for debugging.
- Offers a lot of options for maximum configurability according to your personal tastes and needs.
- Fully integrated into Apache and mod_perl to acheive the best performance. Can also run as a CGI-script, offline or called from another Perl program.
- The Perl module DBIx::Recordset offers high level, easy to handle database access for Embperl.
- If you are concerned about security you can configure Embperl to use Safe.pm. This way, your documents can run in a safe namespace which can allow or disallow certain Perl opcodes.

[HOME] [CONTENT] [PREV (Features - Content)]

Introduction - Content

[HOME] [CONTENT] [NEXT (What is Embperl?)]

[HOME] [CONTENT] [NEXT (What is Embperl?)]

INSTALLATION

[HOME] [CONTENT] [PREV (Installation - Content)]

- UNIX
- WIN 32
- Further Documentation (english)
- Further Documentation (german)

UNIX

- Unpack the archive

- perl Makefile.PL

You will be asked if you want to compile in support for Apache mod_perl. If you say yes, the source of Apache is searched (for headerfiles). If it can't found, please specify the basepath. Please make sure mod_perl is already installed. If you have installed mod_perl-0.96 or higher, Makefile.PL automatically suggests the same source path as was used to build mod_perl.

- make

- make test

- make install

NOTE 1: (only Embperl-1.2b1 or higher) For using session handling you need Apache::Session-0.17 or higher. If possible use Apache::Session 1.00 or higher.

NOTE 2: *Embperl* **runs without additional Perl modules, but the** `make test` **needs the following modules to work:**

- **URI**
- **MIME::Base64**
- **HTML::Parser**
- **HTML::HeadParser**
- **Digest::MD5**
- **libnet**
- **libwww**
- **libwin32 (only Win32)**

If you have already successfully build and tested `mod_perl` **all modules will already be there.**

WIN 32

I have tested the offline mode on Windows 95 with a Microsoft Visual C++ 4.2 compiler and I have succesfully run Embperl with apache_1.3b5-1.3.9 / mod_perl-1.12-1.21 / perl5.004_04-perl5.005 (without threads) on Windows NT with VC++ 5.0.

NOTE: It was necessary for me to compile perl5.004_04 with the VC++ on my own to get it to run with Apache and mod_perl, while I was able to use the binary distribution

(/authors/id/GSAR/perl5.00402-bindist04-bc.tar.gz) of Perl for offline testing under Win 95.

NOTE: You need mod_perl >= 1.12 on win32

On Windows 95/NT you have to do the following:

perl Makefile.PL

nmake (you must have the c compiler in your path)

if you get an error about compiling Embperl.c. Edit Embperl.c and change

-#line ``Embperl.xs''

to

+#line 1 ``Embperl.xs''

now re-run nmake and it should compile.

nmake test

NOTE: nmake test needs libwin32, libwww and HTML::Parser installed to work

nmake install

NOTE 1: (only Embperl-1.2b1 or higher) For using session handling you need Apache::Session-0.17 or higher. If possible use Apache::Session 1.00 or higher.

NOTE 2: *Embperl* **runs without additional Perl modules, but the** `make test` **needs the following modules to work:**

- **URI**
- **MIME::Base64**
- **HTML::Parser**
- **HTML::HeadParser**
- **Digest::MD5**
- **libnet**
- **libwww**
- **libwin32 (only Win32)**

If you have already successfully build and tested `mod_perl` **all modules will already be there.**

Further Documentation (english)

See "perldoc Features" for list of Embperls features

See "perldoc Intro" for an step by step introduction to Embperl.

See "perldoc Embperl" for complete documentation.

See the "eg/" directory for examples.

See "perldoc Faq" for Frequently Asked Questions.

or you can view it online on http://perl.apache.org/embperl/

Further Documentation (german)

See perldoc FeaturesD for list of Embperls features

See perldoc IntroD for an step by step introduction to Embperl.

See perldoc EmbperlD for complete documentation.

or you can view it online on http://www.ecos.de/embperl/

[HOME] [CONTENT] [PREV (Installation - Content)]

FAQ - Content

[HOME] [CONTENT] [NEXT (Downloading, Compiling & Installing)]

[HOME] [CONTENT] [NEXT (Downloading, Compiling & Installing)]

Examples for use of Embperl.

[HOME]

- if.htm
- loop.htm
- table.htm
- lists.htm
- config.htm
- dbi1.htm
- dbi2.htm
- recordset.htm
- input.htm
- upload.htm
- neu.htm
- Execute.pl

NOTE: All examples are just static pages, they just to show how the output looks like. There currently no real working examples here

---

if.htm

simple html file which shows how you can use the if, elsif, else and endif metacommands.

[Embperl source] [Full HTML source] [Result]

---

loop.htm

simple html file which shows how you can use the while, do/until and foreach matacommands to display arrays or hashs

[Embperl source] [Full HTML source] [Result]

---

table.htm

simple html file which shows how to use the dynamic table feature

[Embperl source] [Full HTML source] [Result]

---

lists.htm

shows you how to use dynamic lists and listboxes and dropdownboxes

[Embperl source] [Full HTML source] [Result]

---

config.htm

print out your perl config, like perl -V does

[Embperl source] [Full HTML source] [Result]

---

dbi1.htm

shows how to fetch data from a database and display them into a table using Embperl dynamic tables Please read the setup notes in the file itself.

[Embperl source] [Full HTML source] [Result]

---

dbi2.htm

shows how to fetch data from a database and display them into a table using Embperl loop metacommands Please read the setup notes in the file itself.

[Embperl source] [Full HTML source] [Result]

---

recordset.htm

shows how to fetch data from a database using the DBIx::Recordset module Please read the setup notes in the file itself.

[Embperl source] [Full HTML source] [Result]

---

input.htm

a input and confirmation form (including error checking), which data will be send via mail

[Embperl source] [Full HTML source] [Result Page 1] [Result Page 2] [Result Page 3]

---

upload.htm

Example how to implement file-upload with Embperl

[Embperl source] [Full HTML source]

---

neu.htm

this example shows many of the feature of Embperl, as embedding various code, conditional processing and form management (It's in German but I think it can be understand anyway). If called it works like a wizard known from windows 95 where you can enter data on some consecutive pages and walk forward or back at everytime preserving the input made so far. Note that the database interface is missing from this example.

[Embperl source] [Full HTML source]

Execute.pl

This is an perl script which shows how to use the function HTML::Embperl::Execute. It runs under Apache::Registry or offline.

[Perl source] [Result]

[HOME] [CONTENT] [PREV (Examples - Content)]

---

eg

Support

[HOME] [CONTENT] [PREV (Compatibility)] [NEXT (References)]

Feedback and Bug Reports

Please let me know if you use or test this module. Bugs, questions, suggestions for things you would find useful, etc., are discussed on the mod_perl mailing list.

>From the mod_perl README: The Apache/Perl mailing list (modperl@apache.org) is available for mod_perl users and developers to share ideas, solve problems and discuss things related to mod_perl and the Apache::* (and Embperl) modules. To subscribe to this list, send mail to majordomo@apache.org with the string ``subscribe modperl'' in the body.

There is a hypermail archive for this list available from:
http://outside.organic.com/mail-archives/modperl/

There is an Epigone archive for the mod_perl mailing list at
http://forum.swarthmore.edu/epigone/modperl

Commerical Support

You can get free support on the mod_perl mailing list (see above). If you need commercial support, ecos can provide it for you. We offer:

- **Consulting and assitance for you and your programmers**
- **Planning of your dynamic website**
- **Creating of parts or a whole website**
- **Fixing bugs in Embperl (also available for mod_perl)**
- **Adding new features**

You can reach us via http://www.ecos.de or info@ecos.de

[HOME] [CONTENT] [PREV (Compatibility)] [NEXT (References)]

References

[HOME] [CONTENT] [PREV (Support)] [NEXT (Author)]

- Information
- Download
- CVS

Information

mod_perl	http://perl.apache.org/
mod_perl FAQ	http://perl.apache.org/faq
Embperl	http://perl.apache.org/embperl/
Embperl (german)	http://www.ecos.de/embperl/
DBIx::Recordset	ftp://ftp.dev.ecos.de/pub/perl/dbi
apache web server	http://www.apache.org/
ben-ssl (free httpsd)	http://www.apache-ssl.org/
stronghold (commerical httpsd)	http://www.c2.net/
europe	http://www.eu.c2.net/
other Apache modules	http://perl.apache.org/src/apache-modlist.html

Download

mod_perl	http://www.perl.com/CPAN/modules/by-module/Apache
Embperl	ftp://ftp.dev.ecos.de/pub/perl/embperl
DBIx::Recordset	ftp://ftp.dev.ecos.de/pub/perl/dbi
Win NT/95/98 binarys Apache/perl/ mod_perl/Embperl	http://www.perl.com/CPAN/authors/Jeffrey_Baker/

CVS

The lastest developements are available from a CVS. Look at "perldoc CVS.pod" for a detailed description.

[HOME] [CONTENT] [PREV (Support)] [NEXT (Author)]

Sites running Embperl - What people say about Embperl

[HOME] [CONTENT] [PREV (Sites running Embperl - Content)]

- http://bilder.ecos.de - Gerald Richter [richter@ecos.de]
- http://www.iii.co.uk - Michael Smith [mjs@iii.co.uk]
- https://community.aeccom.com - Dirk Lutzebaeck [lutzeb@aeccom.com]
- http://www.uml.lodz.pl/infoe/index.html - Bartlomiej Papierski [bartek@atrix.com.pl]
- http://www.webpersonals.com - Steve Willer [willer@interlog.com]
- http://www.dejanews.com/ - Dejanews / france -
- http://www.americanspice.com/ - Todd R. Eigenschink [eigenstr@mixi.net]
- http://www.golfbids.com - Dave Paris [dparis@w3works.com]
- http://jobsite.com.au/ - Rick Welykochy [rick@praxis.com.au]
- Intranet site at Tivoli Systems - Jason Bodnar [jbodnar@tivoli.com]
- Intranet site at Motorola - Bryan Thale [thale@rsch.comm.mot.com]
- Intranet site John Fisher [fisher@jfisher.com]
- Intranet Site - Deutsche Bank / germany
- Intranet Site - Siemens AG / germany

If you like to see your site here or just a comment about Embperl, send a mail to richter@dev.ecos.de .

---

http://bilder.ecos.de - Gerald Richter [richter@ecos.de]

This was the site Embperl was originally written for. It's a picture database which contains pictures and tourist information for Rheinland-Pfalz/Germany. It's intended as press information, to reduce the need for sending photographs by real mail. You can view and search the pictures via the Internet or via a direct dial-in. Download is only available for valid users and via direct dial-in. It is also possible to maintain the database via the web and insert new pictures and descriptions and change or delete old ones.

http://bilder.ecos.de

---

http://www.iii.co.uk - Michael Smith [mjs@iii.co.uk]

We're now using embperl for iii's flagship portfolio product, and it's working like a dream. It's fast and simple, we're really happy. Embperl is probably responsible for over 200,000 pages per day now and getting more all the time.

http://www.iii.co.uk

---

https://community.aeccom.com - Dirk Lutzebaeck [lutzeb@aeccom.com]

This is a web based document and communication managment system for the construction industry in Germany. It has gateways to lots of other interfaces like fax, ftp, smtp, isdn/eurofiletransfer. It runs with Postgresql and lots of other perl modules. Nifty features like a browser file upload progress bar couldn't be implemented without the help of mod_perl/Embperl. The system has about 6000 lines of perl module code and 4000 lines of HTML/Embperl lines of code and is currently in the first release phase. I wouldn't know how I could have implemented this system better, faster and reliabler than with mod_perl/Embperl.

https://community.aeccom.com

---

http://www.uml.lodz.pl/infoe/index.html - Bartlomiej Papierski [bartek@atrix.com.pl]

It's a Tourist Guide of city of Lodz made as a part of The City od Lodz website. It contains useful information about hotels, restaurants, etc. in Lodz.

http://www.uml.lodz.pl/infoe/index.html

---

http://www.webpersonals.com - Steve Willer [willer@interlog.com]

Our site is primarily database-driven. We need to keep track of sessions, allow the user to set a lot of information about themselves, search other people's ads, mail each other through the system, talk interactively, etc. Having an algorithmic language embedded in HTML is a must because it's a lot easier to visualize what you're doing.

Our decision to port from PHP to Perl was based entirely on performance. mod_perl keeps compiled scripts and modules in memory, and Perl is 10-20 times faster for most operations even when you don't consider the benefit of not throwing away parsed data. My reasons for going with Embperl is because it's under active development, it seems to have lots of features, it probably doesn't perform poorly and most importantly it is capable of handling error situations gracefully. As you might expect, on a commercial site, you have to be very concerned about reliability and what happens when the inevitable bug creeps in.

http://www.webpersonals.com

---

http://www.dejanews.com/ - Dejanews / france - Doug Bagley [doug@dejanews.com]

We do use Embperl at Deja News, mostly to generate chunks of pages that share a common format. That's why the Execute function was so valuable to us.

http://www.dejanews.com/

---

http://www.americanspice.com/ - Todd R. Eigenschink [eigenstr@mixi.net]

Pretty much everything is Embperl. It's a commerce site for a company that makes spices, sauces, and all sorts of other related things. There's a little more info at http://www.americanspice.com/about.html .

http://www.americanspice.com/

---

http://www.golfbids.com - Dave Paris [dparis@w3works.com]

created by W3Works, LLC for the Golf Auction Network is a realtime auction environment for consumer-based golfing equipment and accessories. The site is comprised of Apache 1.3.X and Stronghold 2.X web servers and supported by an Oracle 8 database. The ``signup'', ``navigation'' and ``overview'' pages are wholly composed of Embperl documents which query the database in realtime. User state is maintained via a session identifier embedded in the URL and Embperl makes management of these identifiers a breeze.

The site has not been well marketed to-date, and the existing average of 1700 Embperl pages per day is well below our estimated capabilities of the system. Embperl has been effectively used in all forms (including secure signup forms sent by the Stronghold secure server), including the administrative sections. Our Embperl logs show a typical execution time range between 50 and 120ms per page. With 25 servers spawned and taking a high average of 100ms per Embperl page, our calculations say the combination should handle more than 250 requests/second - which would, of course, exceed the seek speed of the harddrives, so the hardware then becomes the limiting factor and Embperl performs flawlessly.

W3Works, LLC has been experimenting with the latest (final) release of Embperl and fully intends to make use of it wherever possible in our future projects (which include the re-development of the two domains ``fishing.com'' and ``flyfishing.com'' - which, when combined, currently generate over 190,000 pages and over 1.5M hits and per month - even in their ``cob-web'' state).

http://www.golfbids.com

Another Site from Dave Paris which uses Embperl is http://www.genebot.com

---

http://jobsite.com.au/ - Rick Welykochy [rick@praxis.com.au]

built ENTIRELY in mod_perl/MySQL/HTML::Embperl in a time period of about five weeks :-) BTW: I went on a six week vacation, and the Apache/mod_perl system ran solidly for the entire time without a single glitch or complaint. My compliments to a great Apache module! [I shudder to think how often my holidaying would have been interrupted if I had done all this on a Win/NT box using IIS!]

http://jobsite.com.au/

Intranet site at Tivoli Systems - Jason Bodnar [jbodnar@tivoli.com]

We use embperl at Tivoli Systems for web based discussion forums. It allows us to take a three tierd approach to the software (template pages, object modules, database). By having templates with embedded code we can easily add new features and launch multiple instances of the forums for our internal clients with the ability to customize them any way we like.

Intranet site at Motorola - Bryan Thale [thale@rsch.comm.mot.com]

I maintain a number of intranet web sites for Motorola Labs, the research arm of Motorola, Inc. We use the sites to distribute research papers and information internally amongst ourselves and to the many product groups here at Motorola. We use Embperl quite a bit for such things as generating lists and links to currently available research reports and results and for interfacing to our sites' search engine. The ability to use parsers to extract titles, descriptions, and authorship information directly out of the reports themselves means we can often add a new report complete with links, abstract, and contact information to a page simply by placing the report file in the sites' document tree and let Embperl build the on-screen link to it.

The function that probably gets the hardest workout is our mirror of Internet RFCs and Drafts which uses Embperl to provide easy access to all the information in those repositories. On the lighter side, my personal favorite is a little script to automatically update the new/newer/newest flag on links so that I don't have to remember to change the GIFs as the reports age. It's simple, but it saves me a lot of maintenance work.

Intranet site John Fisher [fisher@jfisher.com]

I've been working extensively with Web servers for several years. I've done considerable work with 'standard' CGI Perl scripts, Javascript, ASP, and PHP (2 & 3). I've even published a book about the first two topics (``The Webmaster's Handbook''). But, without a doubt, HTML::Embperl, in conjunction with mod_perl and the CGI library, is the coolest tool I've ever used.

I had to make an extremely complex intranet site in a short amount of time (isn't that the way it always goes?). Originally, the site was written with PHP3, mainly because I hadn't discovered mod_perl, and wanted to keep things very fast. But, after discovering mod_perl & HTML::Embperl, I rewrote *everything*, about 5,000 lines of code. Took little over a week, and some convincing with my boss, but well worth it. The performance and maintainability went up considerably, and I've found the code to be a lot more interesting (Perl offers a lot more flexibility than PHP3). The code has grown to 7,000 lines, and everyone is just amazed by how fast everything is. My only disappointment is that the latest Perl innovations don't seem to get enough attention in the press.

Intranet Site - Deutsche Bank / germany Christian Gau [CHRISTIAN.GAU@zentrale.deuba.com]

Our server runs a MySQL database, which gets its data from other in-house databases. This database contains aggregated management information stored in a historical format. With Embperl it's easy (and fast) to show this data in small web pages. Its also possible to change them interactively so everybody only sees what he/she wants. We have discovered that the people who supply the information and write

the web pages have learned Perl very quickly, as long as they don't have to leave their usual HTML editor.

Intranet Site - Siemens AG / germany Steffen Geschke [Steffen.Geschke@erlf.siemens.de]

eSupport: Distribution of Software, Documentation and Help Desk for customers and service technician.

[HOME] [CONTENT] [PREV (Sites running Embperl - Content)]

Recordset Introduction - Content

[HOME] [CONTENT] [NEXT (Embperl and DBIx::Recordset)]

[HOME] [CONTENT] [NEXT (Embperl and DBIx::Recordset)]

HTML::Embperl Mirrors
Organized by DNS Country Code

HTTP:

ac.uk - ac.uk - am - ar - at - at - au - au - au - au - be - be - be - bg - bo - br - br - br - ca - ca - ca - ca - ca - ca - ch - ch - cn - cn - co.uk - co.uk - co.uk - co.uk - co.uk - co.uk - co.uk - co.uk - cr - cz - cz - cz - de - de - de - de - dk - dk - dk - ee - es - es - es - es - es - es - fi - fi - fi - fi - fi - fr - fr - fr - gr - hk - hk - hk - hk - hr - hu - hu - hu - id - id - ie - il - il - in - is - is - it - it - it - it - it - it - it - jp - jp - jp - jp - jp - jp - jp - kr - kr - kr - lv - my - mz - nl - nl - nl - no - no - no - pl - pl - pl - pt - pt - pt - pt - pt - pt - ro - ro - ru - ru - ru - ru - ru - ru - se - se - se - sg - si - sk - sk - sk - su - th - th - tr - tw - tw - tw - tw - ua - ua - ua - ua - us - us - us - us - us - us - us - us - us - us - us - us - us - us - us - us - us - us - us - us - us - us - us - us - us - us - yu - za - za -

How do I become an Apache mirror site?

Embperl Home

Database access

[HOME] [CONTENT] [PREV (Debugging)] [NEXT (Security)]

Plain DBI

This is another example of using plain DBI within Embperl. In opposition to the example I gave in the chapter about dynamic tables, this example works with explicit loops.

```
[-
# connect to database
$dbh = DBI->connect($DSN) ;
# prepare the sql select
$sth = $dbh -> prepare ("SELECT * from $table") ;
# excute the query
$sth -> execute ;
# get the fieldnames for the heading in $head
$head = $sth -> {NAME} ;
-]
<table>
   <tr>
   [$ foreach $h @$head $]
       <th>[+ $h +]</th>
   [$ endforeach $]
   </tr>
   [$ while $dat = $sth -> fetchrow_arrayref $]
       <tr>
           [$ foreach $v @$dat $]
               <td>[+ $v +]</td>
           [$ endforeach $]
       </tr>
   [$ endwhile $]
</table>
```

DBIx::Recordset

DBIx::Recordset is a module for easy database access.

Search Example

```
[-*set = DBIx::Recordset -> Search ({%fdat,
                                     ('!DataSource'   => $DSN,
                                      '!Table' => $table,
                                      '$max'   => 5,)}) ; -]
<table>
 <tr><th>ID</th><th>NAME</th></tr>
 <tr>
   <td>[+ $set[$row]{id} +]</td>
   <td>[+ $set[$row]{name} +]</td>
 </tr>
</table>
[+ $set -> PrevNextForm ('Previous Records',
                         'Next Records',
                         \%fdat) +]
```

Search sets up a Recordset object

Search will take the values from `%fdat` and use them to build a SQL WHERE expression. This way, what you search for depends on what is posted to the document. For example, if you request the document with http://host/mydoc.html?id=5 the above example will display all database records where the field 'id' contains the value 5. =head2 Data can accessed as array or via the current record The result of the query can be accessed as an array (this does not mean that the whole array is actually fetched from the database). Alternative, you can directly access the current record just by accessing the fields.

```
    set[5]{id}   access the field 'id' of the sixth found record
    set{id}      access the field 'id' of the current record
```

Fields can be accessed by name

While normal DBI let you access your data by column numbers, DBIx::Recordset uses the field names. This makes your program easier to write, more verbose and independent of database changes.

PrevNextForm generates no/one/two buttons depending if there are more records to display

The PrevNextButtons function can be used to generate button for showing the previous record or the next records. PrevNextButton generates a small form and includes all necessary data as hidden fields. To get it to work, it's enough to feed this data to the next request to Search.

As for Search there are methods for Insert/Update/Delete

Example for Insert

If `%fdat` contains the data for the new record, the following code will insert a new record into the database.

```
[-*set = DBIx::Recordset -> Insert ({%fdat,
                                    ('!DataSource'   => $DSN,
                                     '!Table' => $table)}) ; -]
```

Database table can also tied to a hash

DBIx::Recordset can also tie a database table to a hash. You need to specify a primary key for the table, which is used as key in the hash.

```
    $set{5}{name}    access the name with the id=5
                     (id is primary key)
```

[HOME] [CONTENT] [PREV (Debugging)] [NEXT (Security)]

Features - Content

[HOME] [CONTENT] [NEXT (Main features of HTML::Embperl 1.2)]

[HOME] [CONTENT] [NEXT (Main features of HTML::Embperl 1.2)]

What is Embperl?

[HOME] [CONTENT] [PREV (Introduction - Content)] [NEXT (How to Embed Perl Code in HTML Documents)]

Embed Perl Code in HTML Documents

The main purpose of HTML::Embperl is to embed Perl code in HTML documents. While Embperl can also be used with non-HTML documents, it has several features that are specifically for HTML.

How does it compare to ASP, PHP, ePerl?

Embperl is not the only processor for embedded Perl code. ASP used with ActiveState Perl port provides this for Microsoft IIS, and ePerl is a solution which does this job very well for all sorts of ASCII files. There are other Perl solutions around as well. PHP is a well-known solution for easily building web pages with embedded code and database connections, but it's not Perl.

Additional HTML features

The main advantage of Embperl is the built-in HTML awareness. It provides features for handling form data and HTML tables, along with converting log files and error pages to HTML and linking them together. It also allows for escaping and unescaping.

Integration with Apache and mod_perl

Embperl can be used offline (as a normal CGI script or as a module from other Perl code), but its real power comes when running under mod_perl and Apache. It's directly integrated with Apache and mod_perl to achieve the best performance by directly using Apache functions and precompiling your code to avoid a recompile on every request.

For Usage With High Level HTML Editor

Embperl was designed to be used with a high-level HTML editor. The Perl code can be entered as normal text (the editor need not know any special HTML tags nor is it necessary to enter special HTML tags via uncomfortable dialogs); just enter your code as if it were normal text. Embperl takes care of unescaping the HTML codes and eliminates unwanted HTML tags (like
) which are entered into

your Perl code by the editor (e.g. because you like to have a line break for better readability).

[HOME] [CONTENT] [PREV (Introduction - Content)] [NEXT (How to Embed Perl Code in HTML Documents)]

How to Embed Perl Code in HTML Documents

[HOME] [CONTENT] [PREV (What is Embperl?)] [NEXT (Dynamic Tables)]

Perl code can be embedded in three ways:

1.) [- ... -] Execute code

```
[- $a = 5 -]  [- $b = 6 if ($a == 5) -]
```

The code between the [- and the -] is executed. No output will be generated. This is mainly for assignments, function calls, database queries, etc.

2.) [+ ... +]Output the result

```
[+ $a +]  [+ $array[$b] +] [+ "A is $a" +]
```

The code between the [+ and the +] is executed and the return value (the value of the last expression evaluated) is output (sent to the browser).

3.) [! ... !] Execute code once

```
[! sub foo { my ($a, $b) = @_ ; $a * $b + 7 } !]
```

Same as [- ... -], but the code is only executed for the first request. This is mainly for function definitions and one-time initialization.

Meta-Commands

Embperl support some meta commands to control the ``program flow'' within the Embperl document. This can be compared to preprocessor commands in C. The meta commands take the following form:

```
[$ <cmd> <arg> $]
```

---

if, elsif, else, endif

The if command is just the same as in Perl. It is used to conditionally output/process parts of the document. Example:

```
[$ if $ENV{REQUEST_METHOD} eq 'GET' $]
   This is a GET request
[$ elsif $ENV{REQUEST_METHOD} eq 'POST' $]
   This is a POST request
[$ else $]
   This is not GET and not POST
[$ endif $]
```

This will output one of the three lines depending on the setting of $ENV{REQUEST_METHOD}.

---

while, endwhile

The while command can be used to create a loop in the HTML document. For example:

```
[$ while ($k, $v) = each (%ENV) $]
   [+ $k +] = [+ $v +] <BR>
[$ endwhile $]
```

The above example will display all environment variables, each terminated with a line break.

---

do, until

The do until also create a loop, but with a condition at the end. For example:

```
[- @arr = (3, 5, 7); $i = 0 -]
[$ do $]
   [+ $arr[ $i++ ] +]
[$ until $i > $#arr $]
```

---

foreach, endforeach

Create a loop iterating over every element of an array/list. Example:

```
[$ foreach $v (1..10) $]
   [+ $v +]
[$ endforeach $]
```

---

var <var1> <var2> ...

By default, you do not need to declare any variables you use within an Embperl page. Embperl takes care of deleting them at the end of each request. Sometimes, though, you want to declare them explicitly. You can do this by using var:

```
[$ var $a @b %c $]
```

Has the same effect as the Perl code:

```
use strict ;use vars qw {$a @b %c} ;
```

---

hidden

hidden is used for creating hidden form fields and is described in the form field section below.

[HOME] [CONTENT] [PREV (What is Embperl?)] [NEXT (Dynamic Tables)]

Dynamic Tables

[HOME] [CONTENT] [PREV (How to Embed Perl Code in HTML Documents)] [NEXT (Form fields)]

A very powerful feature of Embperl is the processing of dynamic tables. This feature was designed mainly to display Perl arrays (one or two dimensional, regular and irregular), but can also be used in other ways.

Display a Perl Array

```
[- @a = ( 'A', 'B', 'C') ; -]
<TABLE BORDER=1>
  <TR>
       <TD> [+ $a[$row] +] </TD>
  </TR>
</TABLE>
```

The above example simply displays a table with three rows containing A, B and C.

The trick is done by using the magical variable `$row` which contains the row count and is incremented for every row. The table ends if the expression which contains `$row` returns <undef>. The same can be done with `$col` for columns and `$cnt` can be used if you need a table which wraps after a certain number of elements.

This works with table/select/menu/ol/dl/dir

Simple DBI Example

Here is a simple DBI example that displays the result of a query as a two dimension table, with field names as headings in the first row:

```
[-
# connect to database
 $dbh = DBI->connect($DSN) ;
# prepare the sql select
$sth = $dbh -> prepare ("SELECT * from $table") ;
# excute the query
$sth -> execute ;
# get the fieldnames for the heading in $head
$head = $sth -> {NAME} ;
#continues on the next page...
# get the result in $dat $dat = $sth -> fetchall_arrayref ;
-]
<table>
   <tr><th>[+ $head->[$col] +]</th></tr>
```

```
    <tr><td>[+ $dat -> [$row][$col] +]</td></tr>
 </table>
```

[HOME] [CONTENT] [PREV (How to Embed Perl Code in HTML Documents)] [NEXT (Form fields)]

Form fields

[HOME] [CONTENT] [PREV (Dynamic Tables)] [NEXT (Maintaining persistent (session) data)]

Posted form data available in %fdat/@ffld

The hash `%fdat` contains all values of form fields. The array @ffld contains the names in the order in which they were submitted.

Input/Textarea/Select tags take values from %fdat

If you do not specify a default value for an input tag and a value for that input tag is available in %fdat, Embperl will automatically insert this value and send it to the browser. This is similar to the behavior of CGI.pm. This means that if you post a form to itself, the browser will display the values you just entered.

[$ hidden $]

[$ hidden $] creates hidden form fields for all fields not in another input field. This can be used to transport data through confirmation forms. (For example, a wizard.)

A simple Text input / Confirmation form

The following example shows many of the possibilities of Embperl. It's a simple form where you can enter your name, your email address and a message. If you hit the send button, you see the data you just entered and can confirm the information by hitting the ``send via mail'' button, or you can go back to the input form to change the data. If you confirm your input, the data will be sent to a predefined e-mail address. The example also shows how you can implement error checking--if you miss your name or your e- mail address, you will get a corresponding error message and the input form is shown again.

The first part is the error checking; the second part the confirmation form; the third part sends the mail if the input was ok and is confirmed; the last part is the input form itself.

Depending on the values of $fdat{check}, $fdat{send} and if $fdat{name} and $fdat{email} contains data, the document decides which part to show.

```
[-  $MailTo = 'richter\@ecos.de' ;
 @errors = () ;
 if (defined($fdat{check}) || defined($fdat{send}))
```

```
    {
    push @errors, "**Please enter your name" if (!$fdat{name}) ;
    push @errors, "**Please enter your e-mail address" if (!$fdat{email}) ;
    }
-]
[$if (defined($fdat{check}) and $#errors == -1)$]
[-
 delete $fdat{input} ;
 delete $fdat{check} ;
 delete $fdat{send}
-]
<hr><h3> You have entered the following data:</h3>
<table>
 <tr><td><b>Name</b></td><td>[+$fdat{name}+]</td></tr>
 <tr><td><b>E-Mail</b></td><td>[+$fdat{email}+]</td></tr>
 <tr><td><b>Message</b></td><td>[+$fdat{msg}+]</td></tr>
 <tr><td align="center" colspan="2">
    <form action="input.htm" method="GET">
      <input type="submit" name="send"
             value="Send to [+ $MailTo +]">
      <input type="submit" name="input" value="Change your data">
      [$hidden$]
   </form>
   </td></tr>
</table>
[$elsif defined($fdat{send}) and $#errors == -1$]
[- MailFormTo ($MailTo,'Formdata','email') -]
<hr><h3>Your input has been sent</h3>
[$else$]
<hr><h3>Please enter your data</h3>
<form action="input.htm" method="GET">
 <table>
   [$if $#errors != -1 $]
     <tr><td colspan="2">
     <table>
   <tr><td>[+$errors[$row]+]</td></tr>
     </table>
     </td></tr>
   [$endif$]
   <tr><td><b>Name</b></td> <td><input type="text"
                                       name="name"></td></tr>
   <tr><td><b>E-Mail</b></td> <td><input type="text"
                                         name="email"></td></tr>
   <tr><td><b>Message</b></td> <td><input type="text"
                                          name="msg"></td></tr>
   <tr><td colspan=2><input type="submit"
                            name="check" value="Send"></td></tr>  </table>
</form>
[$endif$]
```

[HOME] [CONTENT] [PREV (Dynamic Tables)] [NEXT (Maintaining persistent (session) data)]

Maintaining persistent (session) data

[HOME] [CONTENT] [PREV (Form fields)] [NEXT (Breaking your code up into components)]

```
(Embperl 1.2 or above)
```

While hidden fields are useful when working with forms, it's often necessary to **store persistent data in a more general way. Embperl utilizes Apache::Session to do this job. Apache::Session is caple of storing persistent data in memory, in a textfile or in a database. More storage methods may supported in the future. While you can simply call Apache::Session from an Embperl page, Embperl can do it for you. All you need to do is to put your data in the hash %udat. The next time the same user requests any Embperl page** `%udat` **will contain the same data. You can simply use this to keep state information for the user. Depending on your expire settings, the state can also kept between mulitiple sessions. A second hash, %mdat, can be used to keep a state for one page, but for multiple users. A simple example would be a page hit counter:**

```
The page is requested [+ $mdat{counter}++ +] times
since [+ $mdat{date} ||= localtime +]
```

The above example counts the page hits and shows the date when the page is first requested. You don't need to worry about performance - as long as you don't touch `%udat` **or %mdat, no action is taken.**

[HOME] [CONTENT] [PREV (Form fields)] [NEXT (Breaking your code up into components)]

Breaking your code up into components

[HOME] [CONTENT] [PREV (Maintaining persistent (session) data)] [NEXT (Debugging)]

```
(Embperl 1.2 or above)
```

Subroutines

It is better to write subroutines than to keep placing repetetive pieces of code in your program many times. You can do this with Embperl too. As an example, if you have text input fields with labels, this may work better for you:

```
[$ sub textinput $]
   [- ($label, $name) = @_ -]
   [+ $label +]<input type=text name=[+ $name +]>
[$ endsub $]
<form>
   [- textinput ('Last Name', 'lname')  -]<p>
   [- textinput ('First Name', 'fname') -]<p>
</form>
```

The **sub metacommand starts the subroutine and the parameters are passed in the array @_. You can do anything in the subroutine that you would normally be able to do inside normal Embperl pages. Embperl lets you call this subroutine just like any other Perl subroutine: just write its name and, if necessary, the parameter list.**

Execute

If you are working on an entire site rather than just a few pages, you are well aware that there are always elements which occur in every page or across many pages. Instead of copying the source code to every page, you can include other Embperl pages in your page - so you have to write the source only once. Such an included page could be a header, a footer, a navigation bar, and so on. Embperl is not only capable of including such partial pages, you can also pass arguments - for example, to tell the navigation bar which of its own element to highlight:

```
Example for a simple navigation bar
[- @buttons = ('Index', 'Infos', 'Search') -]
<table><tr><td>
    [$if $buttons[$col] eq $param[0]$] <bold> [$endif$]
    <a href="[+ $buttons[$col] +].html"> [+ $buttons[$col] +] </a>
    [$if $buttons[$col] eq $param[0]$] </bold> [$endif$]
</td></tr></table>
<hr>
```

Now if you are on the ``Info'' page you can include the navigation bar this way:

```
[- Execute ('navbar.html', 'Infos') -]
```

This will include the navigation bar, which is stored in the file navbar.html, and pass as its first parameter the string 'Infos'. The navigation bar module itself uses a dynamic table to display one column - which contains the text and a link - for every item in the array @buttons. The text which matches that which is passed as the first parameter is displayed in bold. There is also a long form of the Execute call, which allows you to control all of the details of how the called page is executed.

Creating Component Libraries

Instead of creating a single file for every piece of HTML-code you wish to include, you can pack them together in just one library. To do this, split up every piece of code you want to include separately in one Embperl subroutine (sub-metacommand). Now, you can use the import parameter of the Execute function to import all of the subrountines defined in one file, into the namespace of the current page. Afterwards, you are able to call them just like any other Perl subroutine.

Moreover, if you wish to have some systemwide Embperl subroutines, you can put all the Embperl code in a normal Perl module (a foo.pm file), install it into your Perl system (or a private library path), and use it just like any other Perl module - just by saying

```
use mymodule;
```

[HOME] [CONTENT] [PREV (Maintaining persistent (session) data)] [NEXT (Debugging)]

Debugging

[HOME] [CONTENT] [PREV (Breaking your code up into components)] [NEXT (Database access)]

Embperl log file

The log file is the main source for debugging. It shows you what Embperl does while it processes your page. Depending on the debug flag settings, Embperl logs the following things:

Source

Environment

Form data

Evals (Source + Result)

Table processing

Input tag processing

HTTP headers

Embperl log file can be viewed via the browser

For debugging, you can tell Embperl to display a link at the top of each page to your log file. If you follow the link, Embperl will show the portion of the log file corresponding to that request. The log file lines are displayed in different colors to give a better overview.

Embperl error page contains links to the log file

If you have enabled links to the log file, every error displayed in an error page is a link to the corresponding position in the logfile, so you can easily find the place where something is going wrong.

[HOME] [CONTENT] [PREV (Breaking your code up into components)] [NEXT (Database access)]

Security

[HOME] [CONTENT] [PREV (Database access)] [NEXT (Escaping/Unescaping)]

- Safe namespaces
- Operator restrictions

When running under mod_perl, all Perl code shares the same interpreter. This means that every application can access data from every other application. Embperl maintains a separate namespace for every document, which is enough to avoid accidentally overwriting other applications data, but there is no real security. You can access anything you like if you explicitly specify a package name.

Safe namespaces

Therefore, Embperl incorporates Safe.pm, which will make it impossible to access any packages other than your own. This can be used, for example, to calculate something in a Perl module and then pass the results to an Embperl document. If the Embperl document runs in a safe namespace, it can access the data it has received from the browser, but can't access outside itself. Therefore, it's safe to let different people create the layouts for Embperl pages.

Operator restrictions

Safe.pm also permits the administrator to disable every Perl opcode. If you use this, you are able to decide which Perl opcodes are permitted to be used by the page creators.

[HOME] [CONTENT] [PREV (Database access)] [NEXT (Escaping/Unescaping)]

Escaping/Unescaping

[HOME] [CONTENT] [PREV (Security)]

- Input: unescaping
- Output: escaping

Input: unescaping

(disable via optRawInput)

- convert HTML escapes to characters (e.g. < to <) - remove HTML tags from Perl code (e.g.
 insert by high level editor)

Output: escaping

(disable via escmode) convert special characters to HTML (e.g. < to <)

[HOME] [CONTENT] [PREV (Security)]

Installation - Content

[HOME] [CONTENT] [NEXT (INSTALLATION)]

[HOME] [CONTENT] [NEXT (INSTALLATION)]

Embperl Examples - If Metacommands

The following displays an sentence depending on the method (GET, POST, other)

```
[$ if $ENV{REQUEST_METHOD} eq 'GET' $]
This is a GET request
[$ elsif $ENV{REQUEST_METHOD} eq 'POST' $]
This is a POST request
[$ else $]
This is not GET and not POST
[$ endif $]
```

The following displays an different sentence depending on the show parameter

```
[$ if $fdat{show} eq 'yes' $]
The show parameter is 'yes'
[$ elsif $fdat{show} eq 'no' $]
Nothing to show
[$ else $]
Request with if.htm?show=yes
of with if.htm?show=no
[$ endif $]
```

```
<html>
<head>
<title>Embperl Examples - If Metacommands</title>
</head>
<body background="../images/jazzbkgd.gif">

<h1>Embperl Examples - If Metacommands</h1>
<hr><h3>The following displays an sentence depending on the method (GET, POST, other)</h3>

[$ if $ENV{REQUEST_METHOD} eq 'GET' $]
This is a GET request
[$ elsif $ENV{REQUEST_METHOD} eq 'POST' $]
This is a POST request
[$ else $]
This is not GET and not POST
[$ endif $]


<hr><h3>The following displays an different sentence depending on the show parameter</h3>


[$ if $fdat{show} eq 'yes' $]
The show parameter is 'yes'
[$ elsif $fdat{show} eq 'no' $]
Nothing to show
[$ else $]
Request with if.htm?show=yes<BR>
of with if.htm?show=no<BR>
[$ endif $]

<p><hr>

<small>HTML::Embperl (c) 1997-1998 G.Richter</small>

</body>
</html>
```

Embperl Examples - If Metacommands

The following displays an sentence depending on the method (GET, POST, other)

This is a GET request

The following displays an different sentence depending on the show parameter

Request with if.htm?show=yes
of with if.htm?show=no

Embperl Examples - Loop Metacommands

This is a example of using the while metacommand in embperl to show the environement

```
[$ while ($k, $v) = each (%ENV) $]
[+ $k +] = [+ $v +]
[$ endwhile $]
```

This is a example of using the while metacommand to show the loaded modules, with using an index

```
[- $i = 0; @k = keys %INC -]
[$ while ($i < $#k) $]
[+ $k[$i] +] = [+ $INC{$k[$i]} +]
[- $i++ -]
[$ endwhile $]
```

This is a example of using the do .. until metacommand to show the array @arr = (3,5,7)

```
[- @arr = (3, 5, 7); $i = 0 -]
[$ do $]
[+ $arr[ $i++ ] +]
[$ until $i > $#arr $]
```

This is a example of using the foreach metacommand to show the list (1..10)

```
[$ foreach $v (1..10) $]
[+ $v +]
```

[$ endforeach $]

```
<html>
<head>
<title>Embperl Examples - Loop Metacommands</title>
</head>
<body background="../images/jazzbkgd.gif">

<h1>Embperl Examples - Loop Metacommands</h1>
<hr><h3>This is a example of using the while metacommand in embperl to show the environement</h3>

[$ while ($k, $v) = each (%ENV) $]
[+ $k +] = [+ $v +] <BR>
[$ endwhile $]

<hr><h3>This is a example of using the while metacommand to show the loaded
modules, with using an index</h3>

[- $i = 0; @k = keys %INC -]
[$ while ($i < $#k) $]
[+ $k[$i] +] = [+ $INC{$k[$i]} +]<BR>
[- $i++ -]
[$ endwhile $]

<hr><h3>This is a example of using the do .. until metacommand to show the array @arr = (3,5,7)</h3>

[- @arr = (3, 5, 7); $i = 0 -]
[$ do $]
[+ $arr[ $i++ ] +]
[$ until $i > $#arr $]


<hr><h3>This is a example of using the foreach metacommand to show the list (1..10)</h3>

[$ foreach $v (1..10) $]
[+ $v +]
[$ endforeach $]



<p><hr>

<small>HTML::Embperl (c) 1997-1998 G.Richter</small>

</body>
</html>
```

Embperl Examples - Loop Metacommands

This is a example of using the while metacommand in embperl to show the environement

SERVER_SOFTWARE = Apache/1.3.0 (Unix)
DOCUMENT_ROOT = /local/www/data
GATEWAY_INTERFACE = CGI-Perl/1.1
REMOTE_ADDR = 195.52.12.194
SERVER_PROTOCOL = HTTP/1.1
REQUEST_METHOD = GET
TestEnv = 1234
HTTP_USER_AGENT = Mozilla/4.0 (compatible; MSIE 4.01; Windows 95)
QUERY_STRING =
PATH = /bin:/usr/bin:/usr/ucb:/usr/bsd:/usr/local/bin
EMBPERL_VIRTLOG = /embperl-log
HTTP_CONNECTION = Keep-Alive
HTTP_ACCEPT = image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
REMOTE_PORT = 1065
HTTP_ACCEPT_LANGUAGE = de
MOD_PERL = 1.11
EMBPERL_OPTIONS = 0
SCRIPT_FILENAME = /usr/msrc/embperl/eg/x/loop.htm
HTTP_ACCEPT_ENCODING = gzip, deflate
SCRIPT_NAME = /embperl/x/loop.htm
SERVER_NAME = venus.gr.ecos.de
REQUEST_URI = /embperl/x/loop.htm
SERVER_PORT = 8765
HTTP_HOST = www.gr.ecos.de:8765
PATH_TRANSLATED = /usr/msrc/embperl/eg/x/loop.htm
SERVER_ADMIN = root@venus.gr.ecos.de

This is a example of using the while metacommand to show the loaded modules, with using an index

Apache/Status.pm = /usr/lib/perl5/site_perl/Apache/Status.pm
Fcntl.pm = /usr/lib/perl5/i586-linux/5.00404/Fcntl.pm
IO.pm = /usr/lib/perl5/IO.pm
Exporter.pm = /usr/lib/perl5/Exporter.pm
strict.pm = /usr/lib/perl5/strict.pm

overload.pm = /usr/lib/perl5/i586-linux/5.00404/overload.pm
vars.pm = /usr/lib/perl5/vars.pm
subs.pm = /usr/lib/perl5/subs.pm
Cwd.pm = /usr/lib/perl5/Cwd.pm
LWP/MemberMixin.pm = /usr/lib/perl5/site_perl/LWP/MemberMixin.pm
Safe.pm = /usr/lib/perl5/i586-linux/5.00404/Safe.pm
SelectSaver.pm = /usr/lib/perl5/SelectSaver.pm
Apache.pm = /usr/lib/perl5/site_perl/Apache.pm
IO/Seekable.pm = /usr/lib/perl5/IO/Seekable.pm
DynaLoader.pm = /usr/lib/perl5/i586-linux/5.00404/DynaLoader.pm
IO/File.pm = /usr/lib/perl5/IO/File.pm
HTML/Entities.pm = /usr/lib/perl5/site_perl/HTML/Entities.pm
Time/Local.pm = /usr/lib/perl5/Time/Local.pm
Apache/Registry.pm = /usr/lib/perl5/site_perl/Apache/Registry.pm
constant.pm = /usr/lib/perl5/constant.pm
HTTP/Headers.pm = /usr/lib/perl5/site_perl/HTTP/Headers.pm
FileHandle.pm = /usr/lib/perl5/i586-linux/5.00404/FileHandle.pm
HTTP/Date.pm = /usr/lib/perl5/site_perl/HTTP/Date.pm
AutoLoader.pm = /usr/lib/perl5/AutoLoader.pm
HTML/Parser.pm = /usr/lib/perl5/site_perl/HTML/Parser.pm
CGI.pm = /usr/lib/perl5/CGI.pm
IO/Handle.pm = /usr/lib/perl5/IO/Handle.pm
LWP/Protocol.pm = /usr/lib/perl5/site_perl/LWP/Protocol.pm
Apache/SIG.pm = /usr/lib/perl5/site_perl/Apache/SIG.pm
Data/Dumper.pm = /usr/lib/perl5/site_perl/Data/Dumper.pm
Config.pm = /usr/lib/perl5/i586-linux/5.00404/Config.pm
Symbol.pm = /usr/lib/perl5/Symbol.pm
HTTP/Request.pm = /usr/lib/perl5/site_perl/HTTP/Request.pm
Opcode.pm = /usr/lib/perl5/i586-linux/5.00404/Opcode.pm
HTTP/Response.pm = /usr/lib/perl5/site_perl/HTTP/Response.pm
Apache/DBI.pm = /usr/lib/perl5/site_perl/Apache/DBI.pm
HTML/HeadParser.pm = /usr/lib/perl5/site_perl/HTML/HeadParser.pm
Carp.pm = /usr/lib/perl5/Carp.pm
Apache/Symbol.pm = /usr/lib/perl5/site_perl/Apache/Symbol.pm
LWP/Debug.pm = /usr/lib/perl5/site_perl/LWP/Debug.pm
DBI/Form2DB.pm = /usr/lib/perl5/site_perl/DBI/Form2DB.pm
HTML/Embperl.pm = /usr/lib/perl5/site_perl/HTML/Embperl.pm
File/Basename.pm = /usr/lib/perl5/File/Basename.pm
Devel/Symdump.pm = /usr/lib/perl5/site_perl/Devel/Symdump.pm
Apache/Constants.pm = /usr/lib/perl5/site_perl/Apache/Constants.pm
DBD/mysql.pm = /usr/lib/perl5/site_perl/DBD/mysql.pm
DBI.pm = /usr/lib/perl5/site_perl/DBI.pm
DBD/Pg.pm = /usr/lib/perl5/site_perl/DBD/Pg.pm
LWP/UserAgent.pm = /usr/lib/perl5/site_perl/LWP/UserAgent.pm
URI/URL.pm = /usr/lib/perl5/site_perl/URI/URL.pm
LWP.pm = /usr/lib/perl5/site_perl/LWP.pm

HTTP/Status.pm = /usr/lib/perl5/site_perl/HTTP/Status.pm

This is a example of using the do .. until metacommand to show the array @arr = (3,5,7)

3 5 7

This is a example of using the foreach metacommand to show the list (1..10)

1 2 3 4 5 6 7 8 9 10

Embperl Examples - Dynamic Tables

This is a example of using the table tag to show the array @arr = ('A', 'B', 'C')

[- @arr = ('A', 'B', 'C') ; -]

[+ $arr[$row] +]

This is a example of using the table tag in embperl to show the environement

[- @k = keys %ENV -]

Using $row (one value per row)

Row	Var	Content
[+ $i=$row +]	[+ $k[$row] +]	[+ $ENV{$k[$i]} +]

Using $col (one value per column, only one row)

[-$maxcol=99-]

[+ $i=$col +]	[+ $k[$col] +]	[+ $ENV{$k[$i]} +]

Using $cnt and $maxcol (three values per row)

[-$maxcol=3-]

[+ $i=$cnt +]	[+ $k[$cnt] +]	[+ $ENV{$k[$i]} +]

Display an two dimensional array with one, two and three columns !

Please take a look at the source in your browser to see the difference

```
[-
$a[0][0] = '1/1' ;
$a[1][0] = '2/1' ;
$a[1][1] = '2/2' ;
$a[2][0] = '3/1' ;
$a[2][1] = '3/2' ;
$a[2][2] = '3/3' ;

$maxcol=99 ;
-]
$a[[0][[0] = '1/1' ;
$a[[1][[0] = '2/1' ;
$a[[1][[1] = '2/2' ;
$a[[2][[0] = '3/1' ;
$a[[2][[1] = '3/2' ;
$a[[2][[2] = '3/3' ;
```

$tabmode = default

```
[+ $a[$row][$col] +]
```

$tabmode=3 + 48 ; $maxcol = 4; $maxrow = 4

```
[- $tabmode=3 + 48 ; $maxcol = 4; $maxrow = 4 -]
[+ $a[$row][$col] +]
```

$tabmode=1 + 32 ;

```
[- $tabmode=1 + 32 ; -]
[+ $a[$row][$col] +]
```

```
<html>
<head>
<title>Embperl Examples - Dynamic Tables</title>
</head>
<body background="../images/jazzbkgd.gif">

<h1>Embperl Examples - Dynamic Tables</h1>
<hr><h3>This is a example of using the table tag to show the array @arr = ( 'A', 'B', 'C')</h3>

[- @arr = ( 'A', 'B', 'C') ; -]
<table border=1>
<tr>
<td> [+ $arr[$row] +] </td>
</tr>
</table>



<hr><h3>This is a example of using the table tag in embperl to show the environement</h3>

[- @k = keys %ENV -]

<h4>Using $row (one value per row)</h4>

<table>
<TR>
<TH>Row</TH>
<TH>Var</TH>
<TH>Content</TH>
</TR>
<tr>
<td>[+ $i=$row +] </td>
<td>[+ $k[$row] +] </td>
<td>[+ $ENV{$k[$i]} +] </td>
</tr>
</table>

<hr>
<h4>Using $col (one value per column, only one row)</h4>

[-$maxcol=99-]
<table>
<tr>
<td>[+ $i=$col +] </td>
<td>[+ $k[$col] +] </td>
```

```
<td>[+ $ENV{$k[$i]} +] </td>
</tr>
</table>


<hr><h4>Using $cnt and $maxcol (three values per row)</h4>

[-$maxcol=3-]
<table>
<tr>
<td>[+ $i=$cnt +] </td>
<td>[+ $k[$cnt] +] </td>
<td>[+ $ENV{$k[$i]} +] </td>
</tr>
</table>

<hr><h3>Display an two dimensional array with one, two and three columns !</h3>
<h4>Please take a look at the source in your browser to see the difference</h4>

[-
$a[0][0] = '1/1' ;
$a[1][0] = '2/1' ;
$a[1][1] = '2/2' ;
$a[2][0] = '3/1' ;
$a[2][1] = '3/2' ;
$a[2][2] = '3/3' ;

$maxcol=99 ;
-]
$a[[0][[0] = '1/1' ;<BR>
$a[[1][[0] = '2/1' ;<BR>
$a[[1][[1] = '2/2' ;<BR>
$a[[2][[0] = '3/1' ;<BR>
$a[[2][[1] = '3/2' ;<BR>
$a[[2][[2] = '3/3' ;<BR>

<h4>$tabmode = default </h4>

<table>
<tr>
<td>[+ $a[$row][$col] +] </td>
</tr>
</table>


<hr><h4>$tabmode=3 + 48 ; $maxcol = 4; $maxrow = 4 </h4>
```

```
[- $tabmode=3 + 48 ; $maxcol = 4; $maxrow = 4 -]

<table>
<tr>
<td>[+ $a[$row][$col] +] </td>
</tr>
</table>

<hr><h4> $tabmode=1 + 32 ; </h4>
[- $tabmode=1 + 32 ; -]

<table>
<tr>
<td>[+ $a[$row][$col] +] </td>
</tr>
</table>

<p><hr>

<small>HTML::Embperl (c) 1997-1998 G.Richter</small>


</body>
</html>
```

Embperl Examples - Dynamic Tables

This is a example of using the table tag to show the array @arr = ('A', 'B', 'C')

A
B
C

This is a example of using the table tag in embperl to show the environement

Using $row (one value per row)

Row	Var	Content
0	SERVER_SOFTWARE	Apache/1.3.0 (Unix)
1	DOCUMENT_ROOT	/local/www/data
2	GATEWAY_INTERFACE	CGI-Perl/1.1
3	REMOTE_ADDR	195.52.12.194
4	SERVER_PROTOCOL	HTTP/1.1
5	REQUEST_METHOD	GET
6	TestEnv	1234
7	HTTP_USER_AGENT	Mozilla/4.0 (compatible; MSIE 4.01; Windows 95)
8	QUERY_STRING	
9	PATH	/bin:/usr/bin:/usr/ucb:/usr/bsd:/usr/local/bin
10	EMBPERL_VIRTLOG	/embperl-log
11	HTTP_CONNECTION	Keep-Alive
12	HTTP_ACCEPT	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
13	REMOTE_PORT	1068
14	HTTP_ACCEPT_LANGUAGE	de
15	MOD_PERL	1.11
16	EMBPERL_OPTIONS	0
17	SCRIPT_FILENAME	/usr/msrc/embperl/eg/xtable.htm
18	HTTP_ACCEPT_ENCODING	gzip, deflate
19	SCRIPT_NAME	/embperl/xtable.htm
20	SERVER_NAME	venus.gr.ecos.de
21	REQUEST_URI	/embperl/xtable.htm
22	SERVER_PORT	8765
23	HTTP_HOST	www.gr.ecos.de:8765
24	PATH_TRANSLATED	/usr/msrc/embperl/eg/xtable.htm
25	SERVER_ADMIN	root@venus.gr.ecos.de

Using $col (one value per column, only one row)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Apache/1.3.0 (Unix)	/local/www/data	CGI-Perl/1.1	195.52.12.194	HTTP/1.1	GET	1234	Mozilla/4.0 (compatible; MSIE 4.01; Windows 95)		/bin:/usr/bin:/usr/ucb:/usr/bsd:/usr/local/bin	/embperl-log	Keep-Alive	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*	1068	de	1.11	0	/usr/msrc/embperl/eg/xtable.htm	gzip, deflate	/embperl/xtable.htm	venus.gr.ecos.de	/embperl/xtable.htm	8765	www.gr.ecos.de:8765	/usr/msrc/embperl/eg/xtable.htm	root@venus.gr.ecos.de

Using $cnt and $maxcol (three values per row)

0	SERVER_SOFTWARE	Apache/1.3.0 (Unix)	1	DOCUMENT_ROOT	/local/www/data	2	GATEWAY_INTERFACE	CGI-Perl/1.1
3	REMOTE_ADDR	195.52.12.194	4	SERVER_PROTOCOL	HTTP/1.1	5	REQUEST_METHOD	GET
6	TestEnv	1234	7	HTTP_USER_AGENT	Mozilla/4.0 (compatible; MSIE 4.01; Windows 95)	8	QUERY_STRING	
9	PATH	/bin:/usr/bin:/usr/ucb:/usr/bsd:/usr/local/bin	10	EMBPERL_VIRTLOG	/embperl-log	11	HTTP_CONNECTION	Keep-Alive
12	HTTP_ACCEPT	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*	13	REMOTE_PORT	1068	14	HTTP_ACCEPT_LANGUAGE	de
15	MOD_PERL	1.11	16	EMBPERL_OPTIONS	0	17	SCRIPT_FILENAME	/usr/msrc/embperl/eg/xtable.htm
18	HTTP_ACCEPT_ENCODING	gzip, deflate	19	SCRIPT_NAME	/embperl/xtable.htm	20	SERVER_NAME	venus.gr.ecos.de
21	REQUEST_URI	/embperl/xtable.htm	22	SERVER_PORT	8765	23	HTTP_HOST	www.gr.ecos.de:8765
24	PATH_TRANSLATED	/usr/msrc/embperl/eg/xtable.htm	25	SERVER_ADMIN	root@venus.gr.ecos.de			

Display an two dimensional array with one, two and three columns !

Please take a look at the source in your browser to see the difference

$a0[0] = '1/1' ;
$a1[0][0] = '2/1' ;
$a1[1] = '2/2' ;
$a2[0][0] = '3/1' ;
$a2[1] = '3/2' ;
$a2[2] = '3/3' ;

$tabmode = default

1/1
2/1 2/2
3/1 3/2 3/3

$tabmode=3 + 4B ; $maxcol = 4; $maxrow = 4

1/1
2/1 2/2
3/1 3/2 3/3

$tabmode=1 + 32 ;

1/1
2/1 2/2
3/1 3/2 3/3

[- @k = keys %INC -]
[- @v = values %INC -]

OL Tag

1. [+ $k[$row] +] = [+ $v[$row] +]

UL Tag

- [+ $k[$row] +] = [+ $v[$row] +]

Select Tag

If you request this document with list.htm?sel=x you can specify which module of the dropdownlist is initialy selected
eg. lists.htm?sel=Apache.pm

[+ $v[$row] +]

DL Tag

[+ $k[$row] +]
: [+ $v[$row] +]

MENU Tag

- [+ $k[$row] +] = [+ $v[$row] +]

DIR Tag

- [+ $k[$row] +] = [+ $v[$row] +]

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>

<head>
<meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1">
<title>Lists</title>
</head>

<body bgcolor="#FFFFFF">

<p>[- @k = keys %INC -]<br>
[- @v = values %INC -]<br>
</p>
<hr>
<P>OL Tag</P>
<ol>
<li>[+ $k[$row] +] = [+ $v[$row] +]</li>
</ol>

<hr>
<P>UL Tag</P>
<ul>
<li>[+ $k[$row] +] = [+ $v[$row] +]</li>
</ul>

<hr>
<form method="POST">
<P>Select Tag</P>

If you request this document with list.htm?sel=x you can specify which
module of the dropdownlist is initialy selected<BR>
eg. lists.htm?sel=Apache.pm

<p><select name="sel" size="1">
<option value="[+ $k[$row] +]">[+ $v[$row] +]</option>
</select></p>
</form>

<hr>
<P>DL Tag</P>
<dl>
<dt>[+ $k[$row] +]</dt>
<dd>[+ $v[$row] +]</dd>
</dl>
```

```
<hr>
<P>MENU Tag</P>
<menu>
<li>[+ $k[$row] +] = [+ $v[$row] +]</li>
</menu>

<hr>
<P>DIR Tag</P>
<dir>
<li>[+ $k[$row] +] = [+ $v[$row] +]</li>
</dir>
</body>
</html>
```

OL Tag

1. Apache/Status.pm = /usr/lib/perl5/site_perl/Apache/Status.pm
2. Fcntl.pm = /usr/lib/perl5/i586-linux/5.00404/Fcntl.pm
3. IO.pm = /usr/lib/perl5/IO.pm
4. Exporter.pm = /usr/lib/perl5/Exporter.pm
5. strict.pm = /usr/lib/perl5/strict.pm
6. DBIx/Compat.pm = /usr/lib/perl5/site_perl/DBIx/Compat.pm
7. overload.pm = /usr/lib/perl5/i586-linux/5.00404/overload.pm
8. vars.pm = /usr/lib/perl5/vars.pm
9. subs.pm = /usr/lib/perl5/subs.pm
10. Cwd.pm = /usr/lib/perl5/Cwd.pm
11. LWP/MemberMixin.pm = /usr/lib/perl5/site_perl/LWP/MemberMixin.pm
12. Safe.pm = /usr/lib/perl5/i586-linux/5.00404/Safe.pm
13. SelectSaver.pm = /usr/lib/perl5/SelectSaver.pm
14. Apache.pm = /usr/lib/perl5/site_perl/Apache.pm
15. IO/Seekable.pm = /usr/lib/perl5/IO/Seekable.pm
16. DynaLoader.pm = /usr/lib/perl5/i586-linux/5.00404/DynaLoader.pm
17. IO/File.pm = /usr/lib/perl5/IO/File.pm
18. HTML/Entities.pm = /usr/lib/perl5/site_perl/HTML/Entities.pm
19. Time/Local.pm = /usr/lib/perl5/Time/Local.pm
20. Apache/Registry.pm = /usr/lib/perl5/site_perl/Apache/Registry.pm
21. constant.pm = /usr/lib/perl5/constant.pm
22. HTTP/Headers.pm = /usr/lib/perl5/site_perl/HTTP/Headers.pm
23. FileHandle.pm = /usr/lib/perl5/i586-linux/5.00404/FileHandle.pm
24. HTTP/Date.pm = /usr/lib/perl5/site_perl/HTTP/Date.pm
25. AutoLoader.pm = /usr/lib/perl5/AutoLoader.pm
26. HTML/Parser.pm = /usr/lib/perl5/site_perl/HTML/Parser.pm
27. CGI.pm = /usr/lib/perl5/CGI.pm
28. IO/Handle.pm = /usr/lib/perl5/IO/Handle.pm
29. LWP/Protocol.pm = /usr/lib/perl5/site_perl/LWP/Protocol.pm
30. Apache/SIG.pm = /usr/lib/perl5/site_perl/Apache/SIG.pm
31. Data/Dumper.pm = /usr/lib/perl5/site_perl/Data/Dumper.pm
32. Config.pm = /usr/lib/perl5/i586-linux/5.00404/Config.pm
33. Symbol.pm = /usr/lib/perl5/Symbol.pm

34. HTTP/Request.pm = /usr/lib/perl5/site_perl/HTTP/Request.pm
35. Opcode.pm = /usr/lib/perl5/i586-linux/5.00404/Opcode.pm
36. HTTP/Response.pm = /usr/lib/perl5/site_perl/HTTP/Response.pm
37. Apache/DBI.pm = /usr/lib/perl5/site_perl/Apache/DBI.pm
38. DBIx/Recordset.pm = /usr/lib/perl5/site_perl/DBIx/Recordset.pm
39. HTML/HeadParser.pm = /usr/lib/perl5/site_perl/HTML/HeadParser.pm
40. Carp.pm = /usr/lib/perl5/Carp.pm
41. Apache/Symbol.pm = /usr/lib/perl5/site_perl/Apache/Symbol.pm
42. LWP/Debug.pm = /usr/lib/perl5/site_perl/LWP/Debug.pm
43. DBI/Form2DB.pm = /usr/lib/perl5/site_perl/DBI/Form2DB.pm
44. HTML/Embperl.pm = /usr/lib/perl5/site_perl/HTML/Embperl.pm
45. File/Basename.pm = /usr/lib/perl5/File/Basename.pm
46. Devel/Symdump.pm = /usr/lib/perl5/site_perl/Devel/Symdump.pm
47. Apache/Constants.pm = /usr/lib/perl5/site_perl/Apache/Constants.pm
48. DBD/mysql.pm = /usr/lib/perl5/site_perl/DBD/mysql.pm
49. DBI.pm = /usr/lib/perl5/site_perl/DBI.pm
50. DBD/Pg.pm = /usr/lib/perl5/site_perl/DBD/Pg.pm
51. LWP/UserAgent.pm = /usr/lib/perl5/site_perl/LWP/UserAgent.pm
52. URI/URL.pm = /usr/lib/perl5/site_perl/URI/URL.pm
53. LWP.pm = /usr/lib/perl5/site_perl/LWP.pm
54. HTTP/Status.pm = /usr/lib/perl5/site_perl/HTTP/Status.pm
55. HTTP/Message.pm = /usr/lib/perl5/site_perl/HTTP/Message.pm

---

UL Tag

- Apache/Status.pm = /usr/lib/perl5/site_perl/Apache/Status.pm
- Fcntl.pm = /usr/lib/perl5/i586-linux/5.00404/Fcntl.pm
- IO.pm = /usr/lib/perl5/IO.pm
- Exporter.pm = /usr/lib/perl5/Exporter.pm
- strict.pm = /usr/lib/perl5/strict.pm
- DBIx/Compat.pm = /usr/lib/perl5/site_perl/DBIx/Compat.pm
- overload.pm = /usr/lib/perl5/i586-linux/5.00404/overload.pm
- vars.pm = /usr/lib/perl5/vars.pm
- subs.pm = /usr/lib/perl5/subs.pm
- Cwd.pm = /usr/lib/perl5/Cwd.pm
- LWP/MemberMixin.pm = /usr/lib/perl5/site_perl/LWP/MemberMixin.pm
- Safe.pm = /usr/lib/perl5/i586-linux/5.00404/Safe.pm

- SelectSaver.pm = /usr/lib/perl5/SelectSaver.pm
- Apache.pm = /usr/lib/perl5/site_perl/Apache.pm
- IO/Seekable.pm = /usr/lib/perl5/IO/Seekable.pm
- DynaLoader.pm = /usr/lib/perl5/i586-linux/5.00404/DynaLoader.pm
- IO/File.pm = /usr/lib/perl5/IO/File.pm
- HTML/Entities.pm = /usr/lib/perl5/site_perl/HTML/Entities.pm
- Time/Local.pm = /usr/lib/perl5/Time/Local.pm
- Apache/Registry.pm = /usr/lib/perl5/site_perl/Apache/Registry.pm
- constant.pm = /usr/lib/perl5/constant.pm
- HTTP/Headers.pm = /usr/lib/perl5/site_perl/HTTP/Headers.pm
- FileHandle.pm = /usr/lib/perl5/i586-linux/5.00404/FileHandle.pm
- HTTP/Date.pm = /usr/lib/perl5/site_perl/HTTP/Date.pm
- AutoLoader.pm = /usr/lib/perl5/AutoLoader.pm
- HTML/Parser.pm = /usr/lib/perl5/site_perl/HTML/Parser.pm
- CGI.pm = /usr/lib/perl5/CGI.pm
- IO/Handle.pm = /usr/lib/perl5/IO/Handle.pm
- LWP/Protocol.pm = /usr/lib/perl5/site_perl/LWP/Protocol.pm
- Apache/SIG.pm = /usr/lib/perl5/site_perl/Apache/SIG.pm
- Data/Dumper.pm = /usr/lib/perl5/site_perl/Data/Dumper.pm
- Config.pm = /usr/lib/perl5/i586-linux/5.00404/Config.pm
- Symbol.pm = /usr/lib/perl5/Symbol.pm
- HTTP/Request.pm = /usr/lib/perl5/site_perl/HTTP/Request.pm
- Opcode.pm = /usr/lib/perl5/i586-linux/5.00404/Opcode.pm
- HTTP/Response.pm = /usr/lib/perl5/site_perl/HTTP/Response.pm
- Apache/DBI.pm = /usr/lib/perl5/site_perl/Apache/DBI.pm
- DBIx/Recordset.pm = /usr/lib/perl5/site_perl/DBIx/Recordset.pm
- HTML/HeadParser.pm = /usr/lib/perl5/site_perl/HTML/HeadParser.pm
- Carp.pm = /usr/lib/perl5/Carp.pm
- Apache/Symbol.pm = /usr/lib/perl5/site_perl/Apache/Symbol.pm
- LWP/Debug.pm = /usr/lib/perl5/site_perl/LWP/Debug.pm
- DBI/Form2DB.pm = /usr/lib/perl5/site_perl/DBI/Form2DB.pm
- HTML/Embperl.pm = /usr/lib/perl5/site_perl/HTML/Embperl.pm
- File/Basename.pm = /usr/lib/perl5/File/Basename.pm
- Devel/Symdump.pm = /usr/lib/perl5/site_perl/Devel/Symdump.pm
- Apache/Constants.pm = /usr/lib/perl5/site_perl/Apache/Constants.pm
- DBD/mysql.pm = /usr/lib/perl5/site_perl/DBD/mysql.pm
- DBI.pm = /usr/lib/perl5/site_perl/DBI.pm

- DBD/Pg.pm = /usr/lib/perl5/site_perl/DBD/Pg.pm
- LWP/UserAgent.pm = /usr/lib/perl5/site_perl/LWP/UserAgent.pm
- URI/URL.pm = /usr/lib/perl5/site_perl/URI/URL.pm
- LWP.pm = /usr/lib/perl5/site_perl/LWP.pm
- HTTP/Status.pm = /usr/lib/perl5/site_perl/HTTP/Status.pm
- HTTP/Message.pm = /usr/lib/perl5/site_perl/HTTP/Message.pm

Select Tag

If you request this document with list.htm?sel=x you can specify which module of the dropdownlist is initialy selected
eg. lists.htm?sel=Apache.pm

/usr/lib/perl5/site_perl/Apache/Status.pm

DL Tag

Apache/Status.pm
: /usr/lib/perl5/site_perl/Apache/Status.pm

Fcntl.pm
: /usr/lib/perl5/i586-linux/5.00404/Fcntl.pm

IO.pm
: /usr/lib/perl5/IO.pm

Exporter.pm
: /usr/lib/perl5/Exporter.pm

strict.pm
: /usr/lib/perl5/strict.pm

DBIx/Compat.pm
: /usr/lib/perl5/site_perl/DBIx/Compat.pm

overload.pm
: /usr/lib/perl5/i586-linux/5.00404/overload.pm

vars.pm
: /usr/lib/perl5/vars.pm

subs.pm
: /usr/lib/perl5/subs.pm

Cwd.pm
: /usr/lib/perl5/Cwd.pm

LWP/MemberMixin.pm
: /usr/lib/perl5/site_perl/LWP/MemberMixin.pm

Safe.pm
    /usr/lib/perl5/i586-linux/5.00404/Safe.pm
SelectSaver.pm
    /usr/lib/perl5/SelectSaver.pm
Apache.pm
    /usr/lib/perl5/site_perl/Apache.pm
IO/Seekable.pm
    /usr/lib/perl5/IO/Seekable.pm
DynaLoader.pm
    /usr/lib/perl5/i586-linux/5.00404/DynaLoader.pm
IO/File.pm
    /usr/lib/perl5/IO/File.pm
HTML/Entities.pm
    /usr/lib/perl5/site_perl/HTML/Entities.pm
Time/Local.pm
    /usr/lib/perl5/Time/Local.pm
Apache/Registry.pm
    /usr/lib/perl5/site_perl/Apache/Registry.pm
constant.pm
    /usr/lib/perl5/constant.pm
HTTP/Headers.pm
    /usr/lib/perl5/site_perl/HTTP/Headers.pm
FileHandle.pm
    /usr/lib/perl5/i586-linux/5.00404/FileHandle.pm
HTTP/Date.pm
    /usr/lib/perl5/site_perl/HTTP/Date.pm
AutoLoader.pm
    /usr/lib/perl5/AutoLoader.pm
HTML/Parser.pm
    /usr/lib/perl5/site_perl/HTML/Parser.pm
CGI.pm
    /usr/lib/perl5/CGI.pm
IO/Handle.pm
    /usr/lib/perl5/IO/Handle.pm
LWP/Protocol.pm
    /usr/lib/perl5/site_perl/LWP/Protocol.pm
Apache/SIG.pm

/usr/lib/perl5/site_perl/Apache/SIG.pm

Data/Dumper.pm

/usr/lib/perl5/site_perl/Data/Dumper.pm

Config.pm

/usr/lib/perl5/i586-linux/5.00404/Config.pm

Symbol.pm

/usr/lib/perl5/Symbol.pm

HTTP/Request.pm

/usr/lib/perl5/site_perl/HTTP/Request.pm

Opcode.pm

/usr/lib/perl5/i586-linux/5.00404/Opcode.pm

HTTP/Response.pm

/usr/lib/perl5/site_perl/HTTP/Response.pm

Apache/DBI.pm

/usr/lib/perl5/site_perl/Apache/DBI.pm

DBIx/Recordset.pm

/usr/lib/perl5/site_perl/DBIx/Recordset.pm

HTML/HeadParser.pm

/usr/lib/perl5/site_perl/HTML/HeadParser.pm

Carp.pm

/usr/lib/perl5/Carp.pm

Apache/Symbol.pm

/usr/lib/perl5/site_perl/Apache/Symbol.pm

LWP/Debug.pm

/usr/lib/perl5/site_perl/LWP/Debug.pm

DBI/Form2DB.pm

/usr/lib/perl5/site_perl/DBI/Form2DB.pm

HTML/Embperl.pm

/usr/lib/perl5/site_perl/HTML/Embperl.pm

File/Basename.pm

/usr/lib/perl5/File/Basename.pm

Devel/Symdump.pm

/usr/lib/perl5/site_perl/Devel/Symdump.pm

Apache/Constants.pm

/usr/lib/perl5/site_perl/Apache/Constants.pm

DBD/mysql.pm

/usr/lib/perl5/site_perl/DBD/mysql.pm

DBI.pm

/usr/lib/perl5/site_perl/DBI.pm

DBD/Pg.pm

/usr/lib/perl5/site_perl/DBD/Pg.pm

LWP/UserAgent.pm

/usr/lib/perl5/site_perl/LWP/UserAgent.pm

URI/URL.pm

/usr/lib/perl5/site_perl/URI/URL.pm

LWP.pm

/usr/lib/perl5/site_perl/LWP.pm

HTTP/Status.pm

/usr/lib/perl5/site_perl/HTTP/Status.pm

HTTP/Message.pm

/usr/lib/perl5/site_perl/HTTP/Message.pm

MENU Tag

- Apache/Status.pm = /usr/lib/perl5/site_perl/Apache/Status.pm
- Fcntl.pm = /usr/lib/perl5/i586-linux/5.00404/Fcntl.pm
- IO.pm = /usr/lib/perl5/IO.pm
- Exporter.pm = /usr/lib/perl5/Exporter.pm
- strict.pm = /usr/lib/perl5/strict.pm
- DBIx/Compat.pm = /usr/lib/perl5/site_perl/DBIx/Compat.pm
- overload.pm = /usr/lib/perl5/i586-linux/5.00404/overload.pm
- vars.pm = /usr/lib/perl5/vars.pm
- subs.pm = /usr/lib/perl5/subs.pm
- Cwd.pm = /usr/lib/perl5/Cwd.pm
- LWP/MemberMixin.pm = /usr/lib/perl5/site_perl/LWP/MemberMixin.pm
- Safe.pm = /usr/lib/perl5/i586-linux/5.00404/Safe.pm
- SelectSaver.pm = /usr/lib/perl5/SelectSaver.pm
- Apache.pm = /usr/lib/perl5/site_perl/Apache.pm
- IO/Seekable.pm = /usr/lib/perl5/IO/Seekable.pm
- DynaLoader.pm = /usr/lib/perl5/i586-linux/5.00404/DynaLoader.pm
- IO/File.pm = /usr/lib/perl5/IO/File.pm
- HTML/Entities.pm = /usr/lib/perl5/site_perl/HTML/Entities.pm
- Time/Local.pm = /usr/lib/perl5/Time/Local.pm
- Apache/Registry.pm = /usr/lib/perl5/site_perl/Apache/Registry.pm
- constant.pm = /usr/lib/perl5/constant.pm

- HTTP/Headers.pm = /usr/lib/perl5/site_perl/HTTP/Headers.pm
- FileHandle.pm = /usr/lib/perl5/i586-linux/5.00404/FileHandle.pm
- HTTP/Date.pm = /usr/lib/perl5/site_perl/HTTP/Date.pm
- AutoLoader.pm = /usr/lib/perl5/AutoLoader.pm
- HTML/Parser.pm = /usr/lib/perl5/site_perl/HTML/Parser.pm
- CGI.pm = /usr/lib/perl5/CGI.pm
- IO/Handle.pm = /usr/lib/perl5/IO/Handle.pm
- LWP/Protocol.pm = /usr/lib/perl5/site_perl/LWP/Protocol.pm
- Apache/SIG.pm = /usr/lib/perl5/site_perl/Apache/SIG.pm
- Data/Dumper.pm = /usr/lib/perl5/site_perl/Data/Dumper.pm
- Config.pm = /usr/lib/perl5/i586-linux/5.00404/Config.pm
- Symbol.pm = /usr/lib/perl5/Symbol.pm
- HTTP/Request.pm = /usr/lib/perl5/site_perl/HTTP/Request.pm
- Opcode.pm = /usr/lib/perl5/i586-linux/5.00404/Opcode.pm
- HTTP/Response.pm = /usr/lib/perl5/site_perl/HTTP/Response.pm
- Apache/DBI.pm = /usr/lib/perl5/site_perl/Apache/DBI.pm
- DBIx/Recordset.pm = /usr/lib/perl5/site_perl/DBIx/Recordset.pm
- HTML/HeadParser.pm = /usr/lib/perl5/site_perl/HTML/HeadParser.pm
- Carp.pm = /usr/lib/perl5/Carp.pm
- Apache/Symbol.pm = /usr/lib/perl5/site_perl/Apache/Symbol.pm
- LWP/Debug.pm = /usr/lib/perl5/site_perl/LWP/Debug.pm
- DBI/Form2DB.pm = /usr/lib/perl5/site_perl/DBI/Form2DB.pm
- HTML/Embperl.pm = /usr/lib/perl5/site_perl/HTML/Embperl.pm
- File/Basename.pm = /usr/lib/perl5/File/Basename.pm
- Devel/Symdump.pm = /usr/lib/perl5/site_perl/Devel/Symdump.pm
- Apache/Constants.pm = /usr/lib/perl5/site_perl/Apache/Constants.pm
- DBD/mysql.pm = /usr/lib/perl5/site_perl/DBD/mysql.pm
- DBI.pm = /usr/lib/perl5/site_perl/DBI.pm
- DBD/Pg.pm = /usr/lib/perl5/site_perl/DBD/Pg.pm
- LWP/UserAgent.pm = /usr/lib/perl5/site_perl/LWP/UserAgent.pm
- URI/URL.pm = /usr/lib/perl5/site_perl/URI/URL.pm
- LWP.pm = /usr/lib/perl5/site_perl/LWP.pm
- HTTP/Status.pm = /usr/lib/perl5/site_perl/HTTP/Status.pm
- HTTP/Message.pm = /usr/lib/perl5/site_perl/HTTP/Message.pm

DIR Tag

- Apache/Status.pm = /usr/lib/perl5/site_perl/Apache/Status.pm

- Fcntl.pm = /usr/lib/perl5/i586-linux/5.00404/Fcntl.pm
- IO.pm = /usr/lib/perl5/IO.pm
- Exporter.pm = /usr/lib/perl5/Exporter.pm
- strict.pm = /usr/lib/perl5/strict.pm
- DBIx/Compat.pm = /usr/lib/perl5/site_perl/DBIx/Compat.pm
- overload.pm = /usr/lib/perl5/i586-linux/5.00404/overload.pm
- vars.pm = /usr/lib/perl5/vars.pm
- subs.pm = /usr/lib/perl5/subs.pm
- Cwd.pm = /usr/lib/perl5/Cwd.pm
- LWP/MemberMixin.pm = /usr/lib/perl5/site_perl/LWP/MemberMixin.pm
- Safe.pm = /usr/lib/perl5/i586-linux/5.00404/Safe.pm
- SelectSaver.pm = /usr/lib/perl5/SelectSaver.pm
- Apache.pm = /usr/lib/perl5/site_perl/Apache.pm
- IO/Seekable.pm = /usr/lib/perl5/IO/Seekable.pm
- DynaLoader.pm = /usr/lib/perl5/i586-linux/5.00404/DynaLoader.pm
- IO/File.pm = /usr/lib/perl5/IO/File.pm
- HTML/Entities.pm = /usr/lib/perl5/site_perl/HTML/Entities.pm
- Time/Local.pm = /usr/lib/perl5/Time/Local.pm
- Apache/Registry.pm = /usr/lib/perl5/site_perl/Apache/Registry.pm
- constant.pm = /usr/lib/perl5/constant.pm
- HTTP/Headers.pm = /usr/lib/perl5/site_perl/HTTP/Headers.pm
- FileHandle.pm = /usr/lib/perl5/i586-linux/5.00404/FileHandle.pm
- HTTP/Date.pm = /usr/lib/perl5/site_perl/HTTP/Date.pm
- AutoLoader.pm = /usr/lib/perl5/AutoLoader.pm
- HTML/Parser.pm = /usr/lib/perl5/site_perl/HTML/Parser.pm
- CGI.pm = /usr/lib/perl5/CGI.pm
- IO/Handle.pm = /usr/lib/perl5/IO/Handle.pm
- LWP/Protocol.pm = /usr/lib/perl5/site_perl/LWP/Protocol.pm
- Apache/SIG.pm = /usr/lib/perl5/site_perl/Apache/SIG.pm
- Data/Dumper.pm = /usr/lib/perl5/site_perl/Data/Dumper.pm
- Config.pm = /usr/lib/perl5/i586-linux/5.00404/Config.pm
- Symbol.pm = /usr/lib/perl5/Symbol.pm
- HTTP/Request.pm = /usr/lib/perl5/site_perl/HTTP/Request.pm
- Opcode.pm = /usr/lib/perl5/i586-linux/5.00404/Opcode.pm
- HTTP/Response.pm = /usr/lib/perl5/site_perl/HTTP/Response.pm
- Apache/DBI.pm = /usr/lib/perl5/site_perl/Apache/DBI.pm
- DBIx/Recordset.pm = /usr/lib/perl5/site_perl/DBIx/Recordset.pm

- HTML/HeadParser.pm = /usr/lib/perl5/site_perl/HTML/HeadParser.pm
- Carp.pm = /usr/lib/perl5/Carp.pm
- Apache/Symbol.pm = /usr/lib/perl5/site_perl/Apache/Symbol.pm
- LWP/Debug.pm = /usr/lib/perl5/site_perl/LWP/Debug.pm
- DBI/Form2DB.pm = /usr/lib/perl5/site_perl/DBI/Form2DB.pm
- HTML/Embperl.pm = /usr/lib/perl5/site_perl/HTML/Embperl.pm
- File/Basename.pm = /usr/lib/perl5/File/Basename.pm
- Devel/Symdump.pm = /usr/lib/perl5/site_perl/Devel/Symdump.pm
- Apache/Constants.pm = /usr/lib/perl5/site_perl/Apache/Constants.pm
- DBD/mysql.pm = /usr/lib/perl5/site_perl/DBD/mysql.pm
- DBI.pm = /usr/lib/perl5/site_perl/DBI.pm
- DBD/Pg.pm = /usr/lib/perl5/site_perl/DBD/Pg.pm
- LWP/UserAgent.pm = /usr/lib/perl5/site_perl/LWP/UserAgent.pm
- URI/URL.pm = /usr/lib/perl5/site_perl/URI/URL.pm
- LWP.pm = /usr/lib/perl5/site_perl/LWP.pm
- HTTP/Status.pm = /usr/lib/perl5/site_perl/HTTP/Status.pm
- HTTP/Message.pm = /usr/lib/perl5/site_perl/HTTP/Message.pm

```
[- use Config qw (myconfig) ; -]
[- $c = myconfig () -]
[- $c =~ s/\n/\\\
/g -]
[+ $c +]
```

```
<html>
<head>
<title>Perl config (like perl -V)</title>
</head>
<body background="../images/jazzbkgd.gif">

[- use Config qw (myconfig) ; -]
[- $c = myconfig () -]
[- $c =~ s/\n/\\\<BR\\\>/g -]
[+ $c +]

</body>
</html>
```

Summary of my perl5 (5.0 patchlevel 4 subversion 4) configuration:
Platform:
osname=linux, osvers=2.0.30, archname=i586-linux
uname='linux venus 2.0.33 #1 sat mar 7 15:35:47 met 1998 i586 unknown '
hint=recommended, useposix=true, d_sigaction=define
bincompat3=y useperlio=undef d_sfio=undef
Compiler:
cc='cc', optimize='-O2', gccversion=2.7.2.1
cppflags='-Dbool=char -DHAS_BOOL -I/usr/local/include'
ccflags ='-Dbool=char -DHAS_BOOL -I/usr/local/include'
stdchar='char', d_stdstdio=define, usevfork=false
voidflags=15, castflags=0, d_casti32=define, d_castneg=define
intsize=4, alignbytes=4, usemymalloc=n, prototype=define
Linker and Libraries:
ld='cc', ldflags =' -L/usr/local/lib'
libpth=/usr/local/lib /lib /usr/lib
libs=-lgdbm -ldb -ldl -lm -lc
libc=/lib/libc.so.5.4.44, so=so
useshrplib=false, libperl=libperl.a
Dynamic Linking:
dlsrc=dl_dlopen.xs, dlext=so, d_dlsymun=undef, ccdlflags='-rdynamic'
cccdlflags='-fpic', lddlflags='-shared -L/usr/local/lib'

Embperl Examples - DBI access using Dynamic Tables

NOTE:
You must set the database and table to something which exists on your system
Maybe it's necessary to insert a PerlModule DBI into your srm.conf to get this working

---

```
[-

$DSN = 'dbi:mysql:test' ;
$table = 'dbixrs1' ;

use DBI ;

# connect to database
$dbh = DBI->connect($DSN) or die "Cannot connect to '$DSN'" ;

# prepare the sql select
$sth = $dbh -> prepare ("SELECT * from $table") or die "Cannot SELECT from '$table'" ;

# excute the query
$sth -> execute or die "Cannot execute SELECT from '$table'";

# get the fieldnames for the heading in $head
$head = $sth -> {NAME} ;

# get the result in $dat
$dat = $sth -> fetchall_arrayref ;

-]
```

[+ $head->[$col] +]
[+ $dat -> [$row][$col] +]

```
<html>
<head>
<title>Embperl Examples - DBI access using Dynamic Tables</title>
</head>
<body background="../images/jazzbkgd.gif">

<h1>Embperl Examples - DBI access using Dynamic Tables</h1>
<hr><b>NOTE:</b><br>
You must set the database and table to something which exists on your system<br>
Maybe it's necessary to insert a PerlModule DBI into your srm.conf to get this working<br>

<hr>

[-

$DSN = 'dbi:mysql:test' ;
$table = 'dbixrs1' ;

use DBI ;

# connect to database
$dbh = DBI->connect($DSN) or die "Cannot connect to '$DSN'" ;

# prepare the sql select
$sth = $dbh -> prepare ("SELECT * from $table") or die "Cannot SELECT from '$table'" ;

# excute the query
$sth -> execute or die "Cannot execute SELECT from '$table'";

# get the fieldnames for the heading in $head
$head = $sth -> {NAME} ;

# get the result in $dat
$dat = $sth -> fetchall_arrayref ;

-]

<table border=1>
<tr><th>[+ $head->[$col] +]</th></tr>
<tr><td>[+ $dat -> [$row][$col] +]</td></tr>
</table>

<p><hr>

<small>HTML::Embperl (c) 1997-1998 G.Richter</small>
```

```
</body>
</html>
```

Embperl Examples - DBI access using Dynamic Tables

NOTE:
You must set the database and table to something which exists on your system
Maybe it's necessary to insert a PerlModule DBI into your srm.conf to get this working

---

id	name	value1	addon
1	First Name	9991	Is
2	Second Name	9992	it
3	Third Name	9993	it ok?
4	New Name on id 4	4444	Or not??
5	Fivth Name	9995	Is
6	Sixth Name	9996	it
7	Seventh Name	9997	it ok?
8	Eighth Name	9998	Or not??
9	Ninth Name	9999	Is
10	Tenth Name	99910	it
11	Eleventh Name	99911	it ok?
12	Twelvth Name	99912	Or not??
1234	New rec 1234		
12345	New rec 12345		
123456	New rec 123456		
1234567	New rec 1234567		

Embperl Examples - DBI access using Loops

NOTE:
You must set the database and table to something which exists on your system
Maybe it's necessary to insert a PerlModule DBI into your srm.conf to get this working

---

```
[-

$DSN = 'dbi:mysql:test' ;
$table = 'dbixrs1' ;

use DBI ;

# connect to database
$dbh = DBI->connect($DSN) or die "Cannot connect to '$DSN'" ;

# prepare the sql select
$sth = $dbh -> prepare ("SELECT * from $table") or die "Cannot SELECT from '$table'" ;

# excute the query
$sth -> execute or die "Cannot execute SELECT from '$table'";

# get the fieldnames for the heading in $head
$head = $sth -> {NAME} ;

-]

[$ foreach $h @$head $] [$ endforeach $] [$ while $dat = $sth -> fetchrow_arrayref $]
[$ foreach $v @$dat $]
[$ endforeach $]
[$ endwhile $]
```

[+ $h +]
[+ $v +]

```
<html>
<head>
<title>Embperl Examples - DBI access using Loops</title>
</head>
<body background="../images/jazzbkgd.gif">

<h1>Embperl Examples - DBI access using Loops</h1>
<hr><b>NOTE:</b><br>
You must set the database and table to something which exists on your system<br>
Maybe it's necessary to insert a PerlModule DBI into your srm.conf to get this working<br>

<hr>

[-

$DSN = 'dbi:mysql:test' ;
$table = 'dbixrs1' ;

use DBI ;

# connect to database
$dbh = DBI->connect($DSN) or die "Cannot connect to '$DSN'" ;

# prepare the sql select
$sth = $dbh -> prepare ("SELECT * from $table") or die "Cannot SELECT from '$table'" ;

# excute the query
$sth -> execute or die "Cannot execute SELECT from '$table'";

# get the fieldnames for the heading in $head
$head = $sth -> {NAME} ;

-]

<table border=1>

<tr>
[$ foreach $h @$head $]
<th>[+ $h +]</th>
[$ endforeach $]
</tr>

[$ while $dat = $sth -> fetchrow_arrayref $]
<tr>
[$ foreach $v @$dat $]
```

```
<td>[+ $v +]</td>
[$ endforeach $]
</tr>
[$ endwhile $]
</table>

<p><hr>

<small>HTML::Embperl (c) 1997-1998 G.Richter</small>


</body>
</html>
```

Embperl Examples - DBI access using Loops

NOTE:
You must set the database and table to something which exists on your system
Maybe it's necessary to insert a PerlModule DBI into your srm.conf to get this working

---

id	name	value1	addon
1	First Name	9991	Is
2	Second Name	9992	it
3	Third Name	9993	it ok?
4	New Name on id 4	4444	Or not??
5	Fivth Name	9995	Is
6	Sixth Name	9996	it
7	Seventh Name	9997	it ok?
8	Eighth Name	9998	Or not??
9	Ninth Name	9999	Is
10	Tenth Name	99910	it
11	Eleventh Name	99911	it ok?
12	Twelvth Name	99912	Or not??
1234	New rec 1234		
12345	New rec 12345		
123456	New rec 123456		
1234567	New rec 1234567		

Embperl Examples - DBIx::Recordset

NOTE:
You must set the database and table to something which exists on your system. Also this example presumes that the table contains the fields id and name. If not you have to change the field-names in the table below
Maybe it's necessary to insert a PerlModule DBIx::Recordset into your srm.conf to get this working

You may specify search parameters: For example request this document with
recordset.htm?id=5
to get all records where the id = 5

```
[-

$DSN = 'dbi:mysql:test' ;
$table = 'dbixrs1' ;

use DBIx::Recordset ;

*set = DBIx::Recordset -> Search ({%fdat,
('!DataSource' => $DSN,
'!Table' => $table,
'$max' => 5,)}) ; -]
```

ID	NAME
[+ $set[$row]{id} +]	[+ $set[$row]{name} +]

[+ $set -> PrevNextForm ('Previous Records', 'Next Records', \%fdat) +]

```
<html>
<head>
<title>Embperl Examples - DBIx::Recordset</title>
</head>
<body background="../images/jazzbkgd.gif">

<h1>Embperl Examples - DBIx::Recordset</h1>
<hr><b>NOTE:</b><br>
You must set the database and table to something which exists on your system.
Also this example presumes that the table contains the fields <b>id</b> and
<b>name</b>. If not you have to change the field-names in the table below<br>
Maybe it's necessary to insert a PerlModule DBIx::Recordset into your srm.conf
to get this working<br>
<br>
You may specify search parameters: For example request this document with<br>
<b>recordset.htm?id=5</b><br>
to get all records where the id = 5

<hr>

[-

$DSN = 'dbi:mysql:test' ;
$table = 'dbixrs1' ;

use DBIx::Recordset ;

*set = DBIx::Recordset -> Search ({%fdat,
('!DataSource' => $DSN,
'!Table' => $table,
'$max' => 5,)}) ; -]
<table border=1>
<tr><th>ID</th><th>NAME</th></tr>
<tr>
<td>[+ $set[$row]{id} +]</td>
<td>[+ $set[$row]{name} +]</td>
</tr>
</table>
[+ $set -> PrevNextForm ('Previous Records', 'Next Records', \%fdat) +]

<p><hr>

<small>HTML::Embperl (c) 1997-1998 G.Richter</small>
```

```
</body>
</html>
```

Embperl Examples - DBIx::Recordset

NOTE:
You must set the database and table to something which exists on your system. Also this example presumes that the table contains the fields id and name. If not you have to change the field-names in the table below
Maybe it's necessary to insert a PerlModule DBIx::Recordset into your srm.conf to get this working

You may specify search parameters: For example request this document with recordset.htm?id=5
to get all records where the id = 5

ID	NAME
1	First Name
2	Second Name
3	Third Name
4	New Name on id 4
5	Fivth Name

Next Records

Embperl Examples - Input / Confirmation - Form

Please make sure you enter a valid email address in the source before testing the mail function

```
[-
$MailTo = 'richter@ecos.de' ;

@errors = () ;
if (defined($fdat{check}) || defined($fdat{send}))
{
push @errors, "**Please enter your name" if (!$fdat{name}) ;
push @errors, "**Please enter your e-mail address" if (!$fdat{email}) ;
}
-]

[$if (defined($fdat{check}) and $#errors == -1)$]
[- delete $fdat{input} ; delete $fdat{check} ; delete $fdat{send} -]
```

You have entered the following data:

Name	[+$fdat{name}+]
E-Mail	[+$fdat{email}+]
Message	[+$fdat{msg}+]

Send to [+ $MailTo +] Change your data

[$hidden$]

[$elsif defined($fdat{send}) and $#errors == -1$]

[- MailFormTo ($MailTo,'Formdata','email') -]

Your input has been sent

[$else$]

Please enter your data

[$if $#errors != -1 $]
[$endif$]

[+$errors[$row]+]

Name

E-Mail

Message

Send

[$endif$]

```
<html>
<head>
<title>Embperl Example - Input / Confirmation - Form</title>
</head>
<body background="../images/jazzbkgd.gif">

<h1>Embperl Examples - Input / Confirmation - Form</h1>

<b>Please make sure you enter a valid email address in the
source before testing the mail function</b>

[-
$MailTo = 'richter@ecos.de' ;

@errors = () ;
if (defined($fdat{check}) || defined($fdat{send}))
{
push @errors, "**Please enter your name" if (!$fdat{name}) ;
push @errors, "**Please enter your e-mail address" if (!$fdat{email}) ;
}
-]

[$if (defined($fdat{check}) and $#errors == -1)$]
[- delete $fdat{input} ; delete $fdat{check} ; delete $fdat{send} -]

<hr><h3> You have entered the following data:</h3>
<table>
<tr><td><b>Name</b></td><td>[+$fdat{name}+]</td></tr>
<tr><td><b>E-Mail</b></td><td>[+$fdat{email}+]</td></tr>
<tr><td><b>Message</b></td><td>[+$fdat{msg}+]</td></tr>
<tr><td align="center" colspan="2">
<form action="input.htm" method="GET">
<input type="submit" name="send" value="Send to [+ $MailTo +]">
<input type="submit" name="input" value="Change your data">
[$hidden$]
</form>
</td></tr>
</table>

[$elsif defined($fdat{send}) and $#errors == -1$]

[- MailFormTo ($MailTo,'Formdata','email') -]
<hr><h3>Your input has been sent</h3>

[$else$]
```

```
<hr><h3>Please enter your data</h3>

<form action="input.htm" method="GET">
<table>
[$if $#errors != -1 $]
<tr><td colspan="2">
<table>
<tr><td>[+$errors[$row]+]</td></tr>
</table>
</td></tr>
[$endif$]
<tr><td><b>Name</b></td> <td><input type="text" name="name"></td></tr>
<tr><td><b>E-Mail</b></td> <td><input type="text" name="email"></td></tr>
<tr><td><b>Message</b></td> <td><input type="text" name="msg"></td></tr>
<tr><td colspan=2><input type="submit" name="check" value="Send"></td></tr>
</table>
</form>

[$endif$]

<p><hr>

<small>HTML::Embperl (c) 1997-1998 G.Richter</small>

</body>
</html>
```

Embperl Examples - Input / Confirmation - Form

Please make sure you enter a valid email address in the source before testing the mail function

---

Please enter your data

Name

E-Mail

Message

Send

Embperl Examples - Input / Confirmation - Form

Please make sure you enter a valid email address in the source before testing the mail function

---

You have entered the following data:

Name	Gerald
E-Mail	richter@ecos.de
Message	Test

Send to richter@ecos.de | Change your data

Embperl Examples - Input / Confirmation - Form

Please make sure you enter a valid email address in the source before testing the mail function

---

Your input has been sent

Embperl Examples - File-Upload

The file you enter in the following text box will be uploaded to the /tmp directory if you hit the "upload file" button

[$ if !defined $fdat{ImageName} $]

Upload file

[$else$]

```
[-
open FILE, "> /tmp/file.$$";
print FILE $buffer while read($fdat{ImageName}, $buffer, 32768);
close FILE;
-]
```

Your file has been saved to [+ "/tmp/file.$$" +]

Upload new file

[$endif$]

```
<html>
<head>
<title>Embperl Examples - File-Upload</title>
</head>
<body background="../images/jazzbkgd.gif">

<h1>Embperl Examples - File-Upload</h1>

<hr><h3>The file you enter in the following text box will be uploaded to the /tmp directory if you hit the
"upload file" button</h3>


[$ if !defined $fdat{ImageName} $]

<FORM METHOD="POST" ENCTYPE="multipart/form-data">
<INPUT TYPE="FILE" NAME="ImageName">
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Upload file">
</FORM>

[$else$]

[-
open FILE, "> /tmp/file.$$";
print FILE $buffer while read($fdat{ImageName}, $buffer, 32768);
close FILE;
-]

Your file has been saved to [+ "/tmp/file.$$" +]<br>

<FORM METHOD="GET">
<INPUT TYPE="SUBMIT" NAME="Submit" VALUE="Upload new file">
</FORM>


[$endif$]

<p><hr>

<small>HTML::Embperl (c) 1997-1998 G.Richter</small>

</body>
</html>
```

```
[- $b = defined($fdat{back})-][- $c = defined($fdat{cont})-][-$kn = $fdat{Kundennummer} eq 'Ja'-][-$p
= $fdat{page}
-][$if $p==0 or ($p==2 and $b) or ($p==4 and $b and !$kn)$]
```

Eintragen eines neuen Objekts

Um eine neues Objekt welches Sie anbieten, mieten oder kaufen möchten einzutragen benötigen wir zuerst Ihre persönlichen Daten.

Haben Sie bereits eine Kundennummer bei uns?

- ○ Ja, ich habe bereits eine Kundennummer
- ◉ Nein, ich habe keine Kundennummer

<< Zurück | Weiter >> | Abbrechen [$hidden$]

[$elsif $kn and (($p==1 and $c) or ($p==3 and $b))$]

Geben Sie bitte Ihre Kundennummer ein, wenn Sie diese nicht zur Hand haben können Sie auch nach Ihrere Kundennummer suchen, indem Sie die Felder Name und/oder Firma ausfüllen.

Kundennummer: [ ]

Name: [ ]

Firma: [ ]

<< Zurück | Weiter >> | Abbrechen [$hidden$]

[$elsif $kn and (($p==2 and $c) or ($p==5 and $b))$]

Es wurden mehere Personen gefunden die auf Ihrer Angaben zutreffen. Bitte wählen Sie die richtige aus.

[- $AdrData[0]{Kdnr} = 123; $AdrData[0]{Name} = 'Mr. X'; $AdrData[0]{Ort} = 'Somewhere'; $AdrData[1]{Kdnr} = 234; $AdrData[1]{Name} = 'Mr. Y'; $AdrData[1]{Ort} = 'Anywhere'; $AdrData[2]{Kdnr} = 345; $AdrData[2]{Name} = 'Note: Database query is missing'; $AdrData[2]{Ort} = ''; $AdrData[3]{Kdnr} = 456; $AdrData[3]{Name} = 'This are only '; $AdrData[3]{Ort} = 'dummys'; -]

○ [+$AdrData[$row]{Name}+] [+$AdrData[$row]{Ort} +]

<< Zurück | Weiter >> | Abbrechen [$hidden$]

[$elsif !$kn and (($p==1 and $c) or ($p==5 and $b))$]

Geben Sie bitte hier persönnlichen Daten ein.

Firma:			
Anrede:			
Name:		Vorname:	
Straße:		Postfach:	
Plz:		Ort:	
Telefon:		Telefax:	
E-Mail:		WWW Adresse:	

<< Zurück | Weiter >> | Abbrechen [$hidden$]

[$elsif ($p==4 && $c) || ($p==3 && $c)$]

Wenn Sie jetzt auf **Abspeichern klicken, werden folgende Daten von Ihnen gespeichert:**

Firma:	[+$fdat{Firma}+]		
Anrede:	[+$fdat{Anrede}+]		
Name:	[+$fdat{Name}+]	**Vorname:**	[+$fdat{Vorname}+]
Straße:	[+$fdat{Strasse}+]	**Postfach:**	[+$fdat{Postfach}+]
Plz:	[+$fdat{Plz}+]	**Ort:**	[+$fdat{Ort}+]
Telefon:	[+$fdat{Telefon}+]	**Telefax:**	[+$fdat{Telefax}+]
E-Mail:	[+$fdat{EMail}+]	**WWW Adresse:**	[+$fdat{WWW}+]

<< Zurück | Abspeichern | Abbrechen **[$hidden$]**

[$endif$]

```
<!DOCTYPE HTML PUBLIC "-//W3O/DTD HTML//EN">

<html>
<head>
<title>Example for creating a wizard with Embperl</title>
</head>
<body background="../images/jazzbkgd.gif">
<p>[- $b = defined($fdat{back})-][- $c =
defined($fdat{cont})-][-$kn = $fdat{Kundennummer} eq 'Ja'-][-$p =
$fdat{page}<br>
-][$if $p==0 or ($p==2 and $b) or ($p==4 and $b and !$kn)$]</p>
<h1 align="center">Eintragen eines neuen Objekts</h1>
<div align="center"><center>
<table border="2" cellpadding="3" cellspacing="6" width="500">
<tr>
<td width="100%">Um eine neues Objekt welches Sie
anbieten, mieten oder kaufen möchten einzutragen
benötigen wir zuerst Ihre persönlichen Daten.
<p>Haben Sie bereits eine Kundennummer bei uns? </p>
<form action="neu.htm" method="GET">
<input type="hidden" name="page" value="1">
<blockquote>
<p align="left"><input type="radio"
name="Kundennummer" value="Ja">Ja, ich habe
bereits eine Kundennummer<br>
<input type="radio" checked name="Kundennummer"
value="Nein">Nein, ich habe keine Kundennummer</p>
</blockquote>
<p align="center"><input type="submit" name="back"
value="<< Zurück"> <input type="submit"
name="cont" value="Weiter >>"> <input
type="submit" name="cancel" value="Abbrechen">[$hidden$]</p>
</form>
</td>
</tr>
</table>
</center></div>
<p> </p>
<p> [$elsif $kn and (($p==1 and $c) or ($p==3 and $b))$] </p>
<div align="center"><center>
<table border="2" cellpadding="3" cellspacing="6" width="500">
<tr>
<td width="100%">Geben Sie bitte Ihre Kundennummer ein,
wenn Sie diese nicht zur Hand haben können Sie auch
nach Ihrere Kundennummer suchen, indem Sie die Felder
Name und/oder Firma ausfüllen.
```

```
<form action="neu.htm" method="POST">
<input type="hidden" name="page" value="2">
<blockquote>
<p align="left"> </p>
</blockquote>
<table width="100%">
<tr>
<td width="50%">
<p align="right">Kundennummer: </p>
</td>
<td width="50%"><input type="text" size="20"
maxlength="20" name="kdnr"></td>
</tr>
<tr>
<td width="50%">
<p align="right">Name: </p>
</td>
<td width="50%"><input type="text" size="20"
maxlength="20" name="Name"></td>
</tr>
<tr>
<td width="50%">
<p align="right">Firma: </p>
</td>
<td width="50%"><input type="text" size="20"
maxlength="20" name="Firma"></td>
</tr>
</table>
<p align="center"><input type="submit" name="back"
value="<< Zurück"> <input type="submit"
name="cont" value="Weiter >>"> <input
type="submit" name="cancel" value="Abbrechen">[$hidden$]</p>
</form>
</td>
</tr>
</table>
</center></div>
<p> </p>
<p> </p>
<p> [$elsif $kn and (($p==2 and $c) or ($p==5 and $b))$]</p>
<p> </p>
<div align="center"><center>
<table border="2" cellpadding="3" cellspacing="6" width="500">
<tr>
<td width="100%">Es wurden mehere Personen gefunden die
auf Ihrer Angaben zutreffen. Bitte wählen Sie die
```

```
richtige aus.
<form action="neu.htm" method="POST">
<input type="hidden" name="page" value="3">
<blockquote>
<div align="center"><center>
[- $AdrData[0]{Kdnr} = 123; $AdrData[0]{Name} = 'Mr. X'; $AdrData[0]{Ort} = 'Somewhere';
$AdrData[1]{Kdnr} = 234; $AdrData[1]{Name} = 'Mr. Y'; $AdrData[1]{Ort} = 'Anywhere';
$AdrData[2]{Kdnr} = 345; $AdrData[2]{Name} = 'Note: Database query is missing'; $AdrData[2]{Ort}
= '';
$AdrData[3]{Kdnr} = 456; $AdrData[3]{Name} = 'This are only '; $AdrData[3]{Ort} = 'dummys'; -]
<table width="90%">
<tr>
<td width="100%"><input type="radio"
name="kdnr"
value="[+$AdrData[$row]{Kdnr}+]">
[+$AdrData[$row]{Name}+]
[+$AdrData[$row]{Ort} +]</td>
</tr>
</table>
</center></div>
</blockquote>
<p align="center"><input type="submit" name="back"
value="<< Zurück"> <input type="submit"
name="cont" value="Weiter >>"> <input
type="submit" name="cancel" value="Abbrechen">[$hidden$]</p>
</form>
</td>
</tr>
</table>
</center></div>
<p> </p>
<p> </p>
<p> [$elsif !$kn and<br>
(($p==1 and $c) or<br>
($p==5 and $b))$]</p>
<div align="center"><center>
<table border="2" cellpadding="3" cellspacing="6" width="500">
<tr>
<td width="100%">Geben Sie bitte hier persönnlichen
Daten ein.
<form action="neu.htm" method="POST">
<input type="hidden" name="page" value="4"><div
align="center"><center>
<table width="100%">
<tr>
<td align="right" width="15%">Firma: </td>
```

```
<td colspan="3" width="35%"><input
type="text" size="70" maxlength="70"
name="Firma"></td>
<td align="right" width="15%">
<p align="right"> </p>
</td>
<td width="35%"></td>
</tr>
<tr>
<td align="right" width="15%">Anrede: </td>
<td width="35%"><input type="text" size="30"
maxlength="40" name="Anrede"></td>
<td align="right" width="15%">
<p align="right"> </p>
</td>
<td width="35%"></td>
</tr>
<tr>
<td align="right" width="15%">Name: </td>
<td width="35%"><input type="text" size="30"
maxlength="40" name="Name"></td>
<td align="right" width="15%">
<p align="right">Vorname: </p>
</td>
<td width="35%"><input type="text" size="30"
maxlength="40" name="Vorname"></td>
</tr>
<tr>
<td align="right" width="15%">Straße: </td>
<td width="35%"><input type="text" size="30"
maxlength="40" name="Strasse"></td>
<td align="right" width="15%">
<p align="right">Postfach: </p>
</td>
<td width="35%"><input type="text" size="30"
maxlength="40" name="Postfach"></td>
</tr>
<tr>
<td align="right" width="15%">Plz: </td>
<td width="35%"><input type="text" size="30"
maxlength="40" name="Plz"></td>
<td align="right" width="15%">
<p align="right">Ort: </p>
</td>
<td width="35%"><input type="text" size="30"
maxlength="40" name="Ort"></td>
```

```
</tr>
<tr>
<td align="right" width="15%">Telefon: </td>
<td width="35%"><input type="text" size="30"
maxlength="40" name="Telefon"></td>
<td align="right" width="15%">
<p align="right">Telefax: </p>
</td>
<td width="35%"><input type="text" size="30"
maxlength="40" name="Telefax"></td>
</tr>
<tr>
<td align="right" width="15%">E-Mail: </td>
<td width="35%"><input type="text" size="30"
maxlength="40" name="EMail"></td>
<td align="right" width="15%">
<p align="right">WWW Adresse:</p>
</td>
<td width="35%"><input type="text" size="30"
maxlength="40" name="WWW"></td>
</tr>
</table>
</center></div>
<p align="center"><input type="submit" name="back"
value="<< Zurück"> <input type="submit"
name="cont" value="Weiter >>"> <input
type="submit" name="cancel" value="Abbrechen">[$hidden$]</p>
</form>
</td>
</tr>
</table>
</center></div>
<p> </p>
<p>[$elsif ($p==4 && $c) || ($p==3 && $c)$]</p>
<div align="center"><center>
<table border="2" cellpadding="3" cellspacing="6" width="500">
<tr>
<td width="100%">Wenn Sie jetzt auf <strong>Abspeichern</strong>
klicken, werden folgende Daten von Ihnen gespeichert:<br>
<div align="center"><center>
<table width="100%">
<tr>
<td align="right" width="15%"><strong>Firma:</strong> </td>
<td colspan="3" width="35%">[+$fdat{Firma}+]</td>
<td align="right" width="15%">
<p align="right"> </p>
```

```
</td>
<td width="35%"></td>
</tr>
<tr>
<td align="right" width="15%"><strong>Anrede:</strong> </td>
<td width="35%">[+$fdat{Anrede}+]</td>
<td align="right" width="15%">
<p align="right"> </p>
</td>
<td width="35%"></td>
</tr>
<tr>
<td align="right" width="15%"><strong>Name:</strong> </td>
<td width="35%">[+$fdat{Name}+]</td>
<td align="right" width="15%">
<p align="right"><strong>Vorname:</strong> </p>
</td>
<td width="35%">[+$fdat{Vorname}+]</td>
</tr>
<tr>
<td align="right" width="15%"><strong>Straße:</strong> </td>
<td width="35%">[+$fdat{Strasse}+]</td>
<td align="right" width="15%">
<p align="right"><strong>Postfach:</strong> </p>
</td>
<td width="35%">[+$fdat{Postfach}+]</td>
</tr>
<tr>
<td align="right" width="15%"><strong>Plz:</strong> </td>
<td width="35%">[+$fdat{Plz}+]</td>
<td align="right" width="15%">
<p align="right"><strong>Ort:</strong> </p>
</td>
<td width="35%">[+$fdat{Ort}+]</td>
</tr>
<tr>
<td align="right" width="15%"><strong>Telefon:</strong> </td>
<td width="35%">[+$fdat{Telefon}+]</td>
<td align="right" width="15%">
<p align="right"><strong>Telefax:</strong> </p>
</td>
<td width="35%">[+$fdat{Telefax}+]</td>
</tr>
<tr>
<td align="right" width="15%"><strong>E-Mail:</strong> </td>
<td width="35%">[+$fdat{EMail}+]</td>
```

```
<td align="right" width="15%">
<p align="right"><strong>WWW Adresse:</strong></p>
</td>
<td width="35%">[+$fdat{WWW}+]</td>
</tr>
</table>
</center></div>
<form action="neu.htm" method="POST">
<input type="hidden" name="page" value="5">
<p align="center"><input type="submit" name="back"
value="<< Zurück"> <input type="submit"
name="cont" value="Abspeichern"> <input type="submit"
name="cancel" value="Abbrechen">[$hidden$]</p>
</form>
</td>
</tr>
</table>
</center></div>
<p> </p>
<p>[$endif$] </p>
<p> </p>
<p> </p>
</body>
</html>
```

```
#
# Example for using HTML::Embperl::Execute
#
# run this under mod_perl / Apache::Registry
# or standalone
#


use HTML::Embperl ;

my($r) = @_;

$HTML::Embperl::DebugDefault = 811005 ;


$tst1 = '<P>Here is some text</P>' ;

$r -> status (200) ;
$r -> send_http_header () ;

print "<HTML><TITLE>Test for HTML::Embperl::Execute</TITLE><BODY>\n" ;
print "<H1> 1.) Include from memory</H1>\n" ;

HTML::Embperl::Execute ({input => \$tst1,
mtime => 1,
inputfile => 'Some text',
req_rec => $r}) ;


print "<H1> 2.) Include from memory with some Embperl code</H1>\n" ;

HTML::Embperl::Execute ({input => \'[- @ar = (a1, b2, c3)
-]<table><tr><td>[+$ar[$col]+]</td></tr></table></P>',
mtime => 1,
inputfile => 'table',
req_rec => $r}) ;

print "<H1> 3.) Include from memory with passing of variables</H1>\n" ;


$MyPackage::Interface::Var = 'Some Var' ;

HTML::Embperl::Execute ({input => \'<P>Transfer some vars [+ $Var +] !</P>',
inputfile => 'Var',
mtime => 1,
'package' => 'MyPackage::Interface',
```

```
req_rec => $r}) ;

print "<H1> 4.) Change the variable, but not the code</H1>\n" ;

$MyPackage::Interface::Var = 'Do it again' ;

# code is the same, so give the same mtime and inputfile to avoid recompile
# Note you get problems is you change the code, but did not restart the server or
# change the value in mtime. So make sure if you change something also change mtime!

HTML::Embperl::Execute ({input => \'<P>Transfer some vars [+ $Var +] !</P>',
inputfile => 'Var2',
mtime => 1,
'package' => 'MyPackage::Interface',
req_rec => $r}) ;


print "<H1> 5.) Use \@param to pass parameters</H1>\n" ;


HTML::Embperl::Execute ({input => \'<P>Use \@param to transfer some data ([+ " @param " +])
!</P>',
inputfile => 'Param',
req_rec => $r,
param => [1, 2, 3, 4] }
) ;


print "<H1> 6.) Use \@param to pass parameters and return it</H1>\n" ;


my @p = ('vara', 'varb') ;

print "<H3> \$p[0] is $p[0] and \$p[1] is $p[1]<H3>" ;

HTML::Embperl::Execute ({input => \'<P>Got data in @param ([+ "@param" +]) !</P>[- $param[0] =
"newA" ; $param[1] = "newB" ; -]<P>Change data in @param to ([+ "@param" +]) !</P>',
inputfile => 'Param & Return',
req_rec => $r,
param => \@p }
) ;

print "<H3> \$p[0] is now $p[0] and \$p[1] is now $p[1]<H3>" ;

print "<H1> 7.) Presetup \%fdat and \@ffld</H1>\n" ;
```

```
my %myfdat = ('test' => 'value',
'fdat' => 'text') ;

my @myffld = sort keys %myfdat ;

HTML::Embperl::Execute ({input => \'<P><table><tr><td>[+ $k = $ffld[$row] +]</td><td>[+
$fdat{$k} +]</td></tr></table></P>',
inputfile => 'fdat & ffld',
req_rec => $r,
fdat => \%myfdat,
ffld => \@myffld}
) ;


print "<H1> 8.) Inculde a file</H1>\n" ;


HTML::Embperl::Execute ({inputfile => '../inc.htm',
req_rec => $r}) ;


print "<H1> 9.) Inculde a file and return output in a scalar</H1>\n" ;

my $out ;

HTML::Embperl::Execute ({inputfile => '../inc.htm',
output => \$out,
req_rec => $r}) ;


print "<H3>$out</H3>\n" ;

print "<H1> 10.) Done :-)</H1>\n" ;


print "</body></html>\n";
```

1.) Include from memory

Here is some text

2.) Include from memory with some Embperl code

a1 b2 c3

3.) Include from memory with passing of variables

Transfer some vars Some Var !

4.) Change the variable, but not the code

Transfer some vars Do it again !

5.) Use @param to pass parameters

Use \@param to transfer some data (1 2 3 4) !

6.) Use @param to pass parameters and return it

$p[0] is vara and $p[1] is varb

Got data in @param (vara varb) !

Change data in @param to (newA newB) !

$p[0] is now newA and $p[1] is now newB

7.) Presetup %fdat and @ffld

fdat text
test value

8.) Inculde a file

9.) Inculde a file and return output in a scalar

10.) Done :-)

Downloading, Compiling & Installing

[HOME] [CONTENT] [PREV (FAQ - Content)] [NEXT (Common Problems)]

For basics on downloading, compiling, and installing, please see the INSTALLATION in the Embperl documentation. Please be sure to load Embperl at server startup - if you do not, various problems may result. An exception to that rule is when you have compiled mod_perl with **USE_DSO. In this case you must not load Embperl at server statup, neither via an use in your startup.pl file, nor via PerlModule from your httpd.conf.**

Is there a binary distribution of Embperl for Unix?

No.

Is there a binary distribution of Embperl for Win32?

Win NT/95/98 binarys for Apache/perl/mod_perl/Embperl are available from
http://www.perl.com/CPAN/authors/Jeffrey_Baker/

I want to run Embperl with mod_perl under Apache. In what order should I do the compiling?

First mod_perl and Apache, then Embperl.

I'm getting:

../apache_1.3.0/src/include/conf.h:916: regex.h: No such file or directory

Try compiling Embperl again, like this:

```
make DEFS=-DUSE_HSREGEX
```

I'm trying to build HTML::Embperl, and while running 'make' i get:

cc: Internal compiler error: program cc1 got fatal signal 11 make: * [epmain.o] Error 1**

GCC croaking with signal 11 frequently indicates hardware problems. See http://www.bitwizard.nl/sig11/

I have a lot of errors in 'make test' from mod_perl when using Embperl

Try recompiling Perl and all modules -- this can sometimes make those annoying error messages disappear!

Running 'make test' fails with an error message at loading of Embperl (even though mod_perl compiled and tested cleanly!)

This can happen when symbols in the Apache binary can not be found or are not being resolved correctly.

Some OS do this (for instance bsdos), and it can also happen if your Apache binary is set to strip symbol information out from binaries.

Try:

1. **) make clean**
2. **) perl Makefile.PL**

   NOTE: answer _no_ to mod_perl support. (This is important!)
3. **) make test**

If that works, it means that your installation of Embperl is OK, but is having problems resolving symbols with Apache.

Try rebuilding Apache and mod_perl from scratch, and make sure you do not strip symbols out of either.

If you don't succeed with this approach, try statically linking Embperl to Apache/mod_perl (please see the next question for step-by-step instructions on how to do this).

How can I build a statically-linked copy of Embperl with mod_perl support?

1. **) go to your mod_perl directory, change to src/modules/perl and edit the Makefile so that it contains the line**

   ```
   #STATIC_EXTS = Apache Apache::Constants HTML::Embperl
   ```
2. **) add a definition for EPDIR and change the ONJ= line so that it looks like this:**

EPDIR=/usr/msrc/embperl OBJS=$(PERLSRC:.c=.o) $(EPDIR)/Embperl.o $(EPDIR)/epmain.o $(EPDIR)/epio.o (EP DIR)/epeval.o $(EPDIR)/epcmd.o $(EPDIR)/epchar.o $(EPDIR)/eputil.o

3. **) go to the mod_perl directory and run**

```
perl Makefile.PL
```

4. **) go to the Embperl directory and do**

```
make clean
perl Makefule.PL
make
```

(to compile in mod_perl support)

5. **) go back to the mod_perl directory and remake Apache by typing**

```
make
```

Now you have successfully built a httpd with statically-linked Embperl.

NOTE: If you want to stop here, you can skip to step 11. and run a 'make install' in the Embperl directory to finish.

But if you want to run Embperl tests and/or if you want to be able to use Embperl in offline or ``vanilla'' CGI mode, we need to continue:

6. **) go back to the Embperl directory**
7. **) backup the file test/conf/config.pl**
8. **) now build Embperl again but _without_ mod_perl support**

```
make clean
perl Makefile.PL
make
```

9. **) restore your saved config.pl to test/conf/config.pl**

(without this step, only the offline mode would be tested)

10. **) run 'make test' for Embperl**
11. **) do 'make install' for Embperl**

NOTE: You should do it in this order, or it may not work.

NOTE: It seems to be necessary to load Embperl at server startup, either by PerlModule or in a PerlScript. See next question on how to do this.

How do I load Embperl at server startup?

You can load Embperl at server startup by PerlModule or in a startup.pl:

1. **) edit your srm.conf file to read:**

```
PerlModule HTML::Embperl
```

2. **) edit your startup.pl file to read:**

```
use HTML::Embperl
```

NOTE 1: Either of these approaches can often 'fix' SIGSEVs in any mod_perl handler, not just Embperl.

NOTE 2: When mod_perl is compiled as loadable module (i.e. with USE_DSO) you must not load Embperl at server startup time!

---

make test fails with a SIGxxxx, how can I obtain a stack bactrace from gdb?

The eaiest way is

```
make install            -> if Embperl is installed, it's easier
gdb perl                -> start the debugger with perl binary
set args test.pl        -> set the arguments for perl
r                       -> start the program
-> Here you should receive the signal
share                   -> makes sure all symbols are really loaded
bt                      -> show the backtrace
```

To get some more information it would be a good idea to compile Embperl with debugging infomation enabled. Therefor do

---

How do I build Embperl with debugging informations

edit the Makefile

search for the line starting with 'CC = ' add the -g switch to the end of the line

search for the line starting with 'LDDFLAGS = ' add the -g switch to the end of the line

type make to build Embperl with debugging infomation

now start the gdb as decribed before.

---

make test fails with SIGXFSZ

This may occur when the filesize limit for the account, either test is running as or the test httpd, is too small. Embperl make test generates a really large logfile! Yu must increase the filesize limit for that accounts.

Embperl on SCO Unix

>From Red Plait

My OS is SCO Unix 3.2v4.2, Apache 1.3.4, perl 5.004_4, mod_perl 1.18 and Embperl-1.1.1

I done following:

1. **)**

   I made HTML-Embperl-1.1.1 with no mod_perl support (when I builded it with mod_perl 1.18 I can`t link it because it don`t finds ap_XXX functions. When I manually insert src/main/libmain.a from Apache 1.3.4 I got message ``Symbol main is multiple defined in /src/main/libmain.a. and perlmain.o'' ). Then I ``make test'' - all tests was O`k. After this I ``make clean'', ``perl Makefile.pl'' with mod_perl support and ``make install''

2. **)**

   I installed mod_perl and ``perl Makefile.PL'', then ``make''

3. **)**

   because I have`nt dynamical loading ( very old and buggy OS ) I had to manually change src/modules/perl/perlxsi.c to insert bootstraps function`s and it`s invocations and also /src/Makefile to manually insert libXXX.a libraries

   In access.conf I insert code:

```
     PerlModule HTML::Embperl
 <Directory /my_dir>
  SetHandler perl-script
  PerlHandler HTML::Embperl::handler
 </Directory>
```

Embperl and mod_perl on AIX

(from Jens-Uwe Mager)

This was tested on AIX 4.1.5 with Apache 1.3.9 and the latest (post 1.21) modperl from CVS. Earlier versions of modperl would only work after massaging the modperl build process by hand. To build Apache with modperl as a DSO I had to apply the following patch to modperl (the original assumed the apache source is in /usr/local/apache/src):

```
Index: apaci/configure
===================================================================
--- apaci/configure.orig       Wed Aug 18 16:54:07 1999
+++ apaci/configure    Wed Aug 18 16:57:12 1999
@@ -73,6 +73,7 @@
 my_apxs_sourcedir="`$my_apxs -q PREFIX`"
 my_apxs_cflags="`$my_apxs -q CFLAGS`"
 my_apxs_includes="-I`$my_apxs -q INCLUDEDIR`"
+my_apxs_libexec="`$my_apxs -q LIBEXECDIR`"

 #   friendly header
 echo "Configuring mod_perl for building via APXS" 2>&1
@@ -95,6 +96,7 @@
 echo "RANLIB=ranlib" >>$my_makefileconf
 echo "LIBEXT=so" >>$my_makefileconf
 echo "APACHEEXT=$my_apxs_sourcedir/src" >>$my_makefileconf
+echo "APACHELIBEXEC=$my_apxs_libexec" >>$my_makefileconf
 echo "BASEEXT=mod_perl" >>$my_makefileconf
 echo "APXS=$my_apxs" >>$my_makefileconf
 echo "#   own special stuff" >>$my_makefileconf
Index: apaci/mod_perl.config.sh
===================================================================
--- apaci/mod_perl.config.sh.orig      Wed Aug  4 03:17:52 1999
+++ apaci/mod_perl.config.sh   Wed Aug 18 16:55:46 1999
@@ -116,7 +116,7 @@
 perl_lddlflags="`$perl_interp -MConfig -e 'print $Config{lddlflags}'`"

 case "$os_version" in
-    aix*)  perl_lddlflags="$perl_lddlflags -bI:\$(APACHEEXT)/support/httpd.exp" ;;
+    aix*)  perl_lddlflags="$perl_lddlflags -bI:\$(APACHELIBEXEC)/httpd.exp" ;;
     * )    ;;
 esac

Index: apaci/mod_perl.exp
===================================================================
--- apaci/mod_perl.exp.orig    Wed Aug 18 17:26:50 1999
+++ apaci/mod_perl.exp Wed Aug 18 17:58:49 1999
@@ -1 +1,3 @@
+#!
 perl_module
+mod_perl_sent_header
```

A second patch was needed for Apache 1.3.9 itself, as two symbols were forgotton to export in the base release:

```
Index: src/support/httpd.exp
===================================================================
--- src/support/httpd.exp.orig Wed Aug 18 17:07:34 1999
+++ src/support/httpd.exp      Wed Aug 18 17:08:20 1999
@@ -9,6 +9,7 @@
 ap_SHA1Update
 ap_add_cgi_vars
 ap_add_common_vars
```

```
 +ap_add_file_conf
  ap_add_module
  ap_add_named_module
  ap_add_per_dir_conf
 @@ -308,6 +309,7 @@
  ap_server_root_relative
  ap_set_byterange
  ap_set_callback_and_alarm
 +ap_set_config_vectors
  ap_set_content_length
  ap_set_etag
  ap_set_file_slot
```

After that you can simply build Embperl by using the supplied Makefile. As for the question where mod_perl is located, I gave the answer /usr/local/apache/libexec, as modperl is installed as a DSO.

Embperl does not write to the logfile, because of missing permissions of the user Apache runs as.

The apache server is started as root, then set the effective uid to user ``www'', who can then write to the embperl logfile (owned by root) file handle that is passed along. However, if this log file handle is later accidentally closed, then reopen, the www user would have problem writing to it?

The reopen is only done when the logfile name changes. As log as you don't change the name on the logfile, the logfile will stay open.

The problem (in this case) is, that Embperl init function ,(Init in epmain.c) calls OpenLog will an second argument of zero. Which will only save the filename. The log will actually opened on the first write to it (or at the start of the first request). At this time your Apache has alreay switch to user www. This is done to allow to change the logfile name before an request, but after the init is already called (which is done when you or Apache ``use'' the module)

The current solutions is to write something to the log, before Apache changes it's user (i.e. in the startup.pl)

[HOME] [CONTENT] [PREV (FAQ - Content)] [NEXT (Common Problems)]

Common Problems

[HOME] [CONTENT] [PREV (Downloading, Compiling & Installing)] [NEXT (Common Questions)]

The most common problems of all involve Escaping and Unescaping. They are so common, that an entire section on Escaping & Unescaping is devoted to them.

Why doesn't the following line work?

```
[+ $var . "<b>". $foo . "</b>". $bar +]
```

See what we mean? This is an Escaping & Unescaping problem for sure. You need to escape <b> as ' <b&gt ' and you probably also need to read the section on Escaping & Unescaping...

I'm getting: "Glob not terminated at ..."

This might be a problem with Escaping & Unescaping as well.

My HTML is getting stripped out.

Sounds like a problem with Escaping & Unescaping again!

Unless, of course, you have already read the section on Escaping & Unescaping, and it is still happening... Like if you are using optRawInput and your HTML is _still_ being stripped out...

I _am_ using optRawInput, and my HTML _is_ still being stripped out!

Aha! Well that's different! Never mind..

It can be easy to accidentally set optRawInput too late in your code...

Try setting it in an extra Perl block ([- $optRawInput = 1 -]) earlier in the code, or in the server config, and see if that doesn't solve the problem... (optRawInput must be set before the block that uses it begins, as the block which uses it shouldn't be translated).

Help! I got a SIGSEGV! Ack!

If Embperl is not compiled at server startup, it can cause error messages, SEGfaults, core dumps, buffer overflow, etc - especially if you are using another module inside an Embperl page. As far as anyone can tell, this seems to be a Perl/mod_perl problem - but maybe not. If you have any ideas, let me know.

To see the steps for loading Embperl at server startup, please see the section Downloading, Compiling & Installing.

NOTE: When mod_perl is compiled with **USE_DSO it behaves vice versa and you may get SIGSEGVs when Embper is loaded at server startup time.**

I am having troubles with using Embperl in combination with Apache::Include inside a Apache::Registry script.

This is a known problem, but it is a problem with mod_perl rather than with Embperl. It looks like mod_perl clears the request_rec after the first subrequest, so that it later doesn't know which subrequest was intended (unless it's explicitly specified). Try using:

```
Apache::Include->virtual("test.epl", $r);
```

(instead of just Apache::Include->virtual(``test.epl''); where `$r` is the apache request rec)

I can't get PerlSendHeader to work under Embperl?

You don't need PerlSendHeader when using Embperl - Embperl always sends its own httpd header.

But how do I customize the header that Embperl is sending?

You'll find the answer to this and many other header issues in the Common Questions section.

I can't figure out how to split a 'while' statement across two [- -] segments

That isn't surprising, as you cannot split Perl statements across multiple [- -] blocks in Embperl :) You need to use a metacommand for that. The [$while$] metacommand comes to mind... :)

For a list of all possible metacommands, see the section on Meta-Commands in the Embperl documentation.

```
[$ while $st -> fetch $]
        #some html or other Embperl blocks goes here
[$ endwhile $]
```

Newer Embperl versions (1.2b3 and above) supports the [* *] which can be used for such purposes.

```
[* while ($st -> fetch) { *]
        #some html or other Embperl blocks goes here
[* } *]
```

While the later can use all Perl control structures, the first seems to me more readable and is better debugable, because Embperl controls the execution of the control structure it can do a quite better job in debug logging.

My HTML tags like '<' '>' and '"' are being translated to <, > !!!

Hey! Not you again!? I thought we already sent you to the Escaping & Unescaping section of the FAQ?!?! ;)

Netscape asks to reload the document

If you have something like this in your source, it may be the problem:

```
<META HTTP-EQUIV="content-type" CONTENT="text/html;charset=iso-8859-1">
```

Netscape seems to have a problem in such cases, because the http header is only content-type text/html, while the META HTTP-EQUIV has an additional charset specified. If you turn optEarlyHttpHeader off, Embperl will automatically set the http header to be the same as the META HTTP-EQUIV.

[HOME] [CONTENT] [PREV (Downloading, Compiling & Installing)] [NEXT (Common Questions)]

Common Questions

[HOME] [CONTENT] [PREV (Common Problems)] [NEXT (Escaping & Unescaping)]

The most common questions of all deal with Escaping & Unescaping - they are so common that the whole next section is devoted to them. Less common questions are addressed here:

How can I get my HTML files to be converted into Perl code which, as a whole, could then be compiled as function so that I could, for instance, fetch Perl docs from the Formatter table and compile them the way AUTOLOAD does.

Embperl cannot covert your HTML into one piece of Perl-code, but you can wrap the call to Execute into a Perl function and let AUTOLOAD call it.

I have an HTML page which is dynamically generated at runtime and should be post-processed by Embperl. How can I do this?

1. **) Generate the page within a normal CGI/Apache::Registry script and put the result into a scalar - then you can call HTML::Embperl::Execute to post-process your document.**

Execute can either send the document to the browser or put it into another scalar for further processing.

2. **) Use EMBPERL_INPUT_FUNC (1.1b1 and above). With this configuration directive, you can specify a custom input function which reads the HTML source from the disk or even from a database. Embperl also provides the function ProxyInput, which allows you to get input from another web server altogether.**
3. **) Look at the module Apache::EmbperlChain, which is able to chain multiple modules, including Embperl, together.**

How can I customise the header that Embperl is sending?

You can write it as

```
<meta http-equiv="Content-Type" content="text/html">
```

(Embperl will automatically insert all meta http-equiv tags into the http header)

or use `%http_headers_out`

```
[- $http_headers_out{'Content-Type'} = 'text/html' -]
```

or (only when running under mod_perl) you can use

```
[- $req_rec -> content_type ('text/html') -]
```

Can I use Embperl to send cookies?

Yes. Embperl sends its own headers, so all you have to do to send cookies is to remember to print an additional header.

Example Code:

1. **) in documents, add**

```
<META HTTP-EQUIV="Set-Cookie" CONTENT="[+ $cookie +] = [+ $value +]">
```

2. **) or use %http_headers_out**

```
[- $http_headers_out{'Set-Cookie'} = "$cookie=$value" -]
```

3. **) or - using mod_perl's functionality - use**

```
[- $req_rec ->  header_out("Set-Cookie" => "$cookie=$value"); -]
```

NOTE: You make also take a look at Embperls (1.2b2 and above) ability to handle sessions for you inside the `%udat` **and** `%mdat` **hashes.**

Can I do a Redirect with Embperl?

The following way works with mod_perl and as cgi:

```
[- $http_headers_out{'Location'} = "http://www.ecos.de/embperl/"; -]
```

the status of the request will automaticly set to 301.

or use the mod_perl function Apache::header_out.

Example Code:

```
[-
```

```
use Apache;
use Apache::Constants qw(REDIRECT);
$req_rec->header_out("Location" => "http://$ENV{HTTP_HOST}/specials/");
$req_rec->status(REDIRECT);
-]
```

If there is nothing more to do on this page, you may call `exit` directly after setting the status.

Can I serve random GIFs with Embperl? (Will Lincoln Stein's GD.pm module work with Embperl??)

As always, there is more than one way to do this - especially as this is more of a question of how you are coding your HTML than how you are coding your Embperl.

Here are some ideas:

1. **) You could include an IMG tag which points to your cgi-bin, where a regular CGI script serves the graphics.**
2. **) You could be running Apache::Registry, which can generate on-the-fly GIFs using GD. (This is just the same as if you were including the GD image from a static page or from another CGI script, but it allows all of the appropriate logic to live in a single document, which might be appropriate for some Embperl users).**

If you think of another way, or come up with some sample code, I'd love to hear from you, so that I could add it to the FAQ...

Can I use Embperl as a template for forms? Can I make form values persist (like with "vanilla" CGI)? Does Embperl rewrite my template file so that parameters of things like INPUT/TEXTAREA/SELECT persist?

Yes. Your page design staff should just be able to say <input name=``foo''> and let the default attributes of ``foo'' be defined elsewhere - for instance in a settings file. In this case, `%fdat` should be pre-set with your default values. Setting $fdat{foo} = ``abc'' will cause Embperl to change the above code to <input name=``foo'' value=``abc''>.

Does Embperl automatically add HIDDEN fields?

The [$hidden$] metacommand creates hidden fields for every entry in `%fdat` which was not used by any other input tag so far.

You can also try something like this:

```
[-
$fdat{foo} = "abc" ;
$fdat{bar} = "xyz" ;
-]
```

```
<input name="foo">
[$hidden$]
```

and Embperl will create:

```
<input name="foo" value="abc">
<input type=hidden name="bar" value="xyz">
```

For a list of all possible metacommands, see the section on Meta-Commands in the Embperl documentation.

What about security? Is Embperl Secure?

Just like anything else, Embperl is as secure as you make it. Embperl incorporates Safe.pm, which will make it impossible to accidentally access other Packages - it also permits the Administrator to disable Perl opcodes, etc.

For more on security, please see (Safe-)Namespaces and opcode restrictions in the Embperl documentation.

Is there any plan to make Embperl an Object so someone could subclass it and override certain of its methods? (For example, I'd like to let it parse the file for me, but then let me control the manipulation of the form tags.)

Embperl is going to be an Object from version 1.2b1. This, among other things, make it re-entrant, so that you will be able to call Execute from within an Embperl page. It will also mean that Embperl will come with hooks, which will allow you to alter or change the way Embperl processes code. The details have not all been worked out yet, but I'm working on it... :)

Are Embperl routines currently pre-compiled or even cached, or are only fragments cached?

All embedded Perl code is compiled the first time it is executed and cached for later use. The second time the code is executed, only the precompiled p-code is called.

Every code block is compiled as a single subroutine. The HTML text between the Perl block is still read from the file.

Why are Perl blocks broken up into single subroutines?

1. **) It makes it easier to process the HTML tags between the Perl blocks - this gives you more control over what's happening**
2. **) If you compiled _everything_ to Perl, you would hold all of the HTML text in memory, and**

your Apache child processes would grow and grow... But often-accessed documents are still held in memory by your os disk cache, which is much more memory-efficient.

3. **) There is only so far that you can go with precompiling until you reach the point of diminishing returns. My guess is that converting dynamic tables and other HTML processing to Perl at this point in Embperl's development would actually slow down operation.**

Can I pass QUERY_STRING information to an HTML::Embperl::Execute call?

With Embperl 1.0 and higher, you can do this. QUERY_STRING is set as $ENV{QUERY_STRING} by default. Alternatively, you can use the fdat parameter to pass values to %fdat.

How to include other files into Embperl pages?

I am using embedded Perl on my site and am curious if I can use it for server side includes. I want to embed the contents of file x.html into file y.html such that whenever I change x.html, displaying y.html will also reflect this change. How do I do it using embedded perl?

You need Embperl 1.2b4 or above. Then you can say inside of y.html:

```
[- Execute ('x.html') -]
```

EmbPerl iteration without indexing

I have a rather large table in a database which I'd like to display using EmbPerl. All of the examples show a process of fetching all the data first, then iterating through it using `$row` and $col, like this:

```
[-
$sth = $dbh -> prepare ("select * from $comptbl order by SubSystem");
$sth -> execute;
$dref = $sth -> fetchall_arrayref;
-]
<TABLE>
   ... $dref -> [$row][0] ...
</TABLE>
```

I'd prefer to fetch the data one row at a time, how can I do this?

For solution 1 you may write

```
<table>
[$while $rref = $sth -> fetch $]
    <tr>....</tr>
[$endwhile$]
</table>
```

Solution 2 should work like this

```
<table>
   <tr> [- $dummy = $row ; $rref = $sth -> fetch -]
            ....
   </tr>
</table>
```

The table ends when the expression where `$row` is used in some way returns <undef>. So also there is no releation between `$row` and the fetch, both conditions are met.

How to display arrays with undef values in it?

```
I'm doing a search on a table where some of the columns have NULL and
non-NULL values.  DBIx::Recordset has no problem reading this values
The problem is that I then tried to print these values
out in a table using Embperl's  table feature, like this.

 <TABLE>
 <TR>
 <TD>$set[$row]{column_name1}</TD>
 <TD>$set[$row]{column_nameN}</TD>
 </TR>

 The problem is that I got 5 rows instead of the 15 that I was expected.  I
 have been  trying all kinds of tweaks to the arguments to the Search
 function and getting nowwhere, until I re-read the Embperl docs.  Embperl
 will not print out a table row if one of the columns has an
 expression that
 is undefined.  This is a problem since DBIx::Recordset (and DBI)
 natually uses undef to represent a NULL value for a column.  So I made a
 slight modification to my embperl code.

 <TABLE>
 <TR>
 <TD>defined($set[$row]{column_name1}) ? $set[$row]{column_name1} :
 "UNDEF"</TD>
                                    .   .   .
 <TD>$set[$row]{column_nameN} ? $set[$row]{column_nameN} : "UNDEF"</TD>
 </TR>

Now all 15 rows appear as expected, with "UNDEF" representing the NULL
values in the database.
```

Another way top solve you problem may be:

```
 <TABLE>
 <TR>
 [- $r = $set[$row] -]
 <TD> [+ $r -> {column_name1} +] </TD>
                                    .   .   .
 <TD> [+ $r -> {column_nameN} +] </TD>
 </TR>
```

This will only refer one time to `$row` and the expression is defined, as long as the row could be fetched from the db. All NULL fields will be displayed as empty table cells.

[HOME] [CONTENT] [PREV (Common Problems)] [NEXT (Escaping & Unescaping)]

Escaping & Unescaping

[HOME] [CONTENT] [PREV (Common Questions)] [NEXT (Debugging)]

Escaping & Unescaping Input

By default, Embperl removes all HTML tags from the Perl source. It does this because many high-end WYSIWYG HTML Editors (like MS Front Page) insert HTML tags like <FONT> and <COLOR> in rather random places (like in the middle of your Perl code). This Embperl feature keeps things like

```
[- $var  = 1; <br>
    $foo = 2 -]
```

permissable, so that you can enter Perl code while you mark up pages in an editor, all at once. In this example, Embperl would remove the unnecessary
 tag and, therefore, make Perl happy. And if Perl is happy, we are all happy.

It is not difficult to change this behavior, if you are the kind of person who codes HTML in an ascii editor (like vi or emacs).

If you use a high-level HTML editor, you shouldn't have any problems with input escaping, because the editor will, for example, write a '<' as '<' in the HTML code. Embperl translates this back to '<' and therefore it knows that this wasn't an HTML tag which should be removed.

Problems with input escaping only occur if you use an ascii editor. Then you will need to escape input (see the next section for details on how to do this).

To see the exact steps taken by Embperl to process a Perl-laden document, please see the section Inside Embperl in the Embperl documentation.

Ways To Escape Input:

1. **Escape it -> \<H1>**

   NOTE: Inside double quotes you will need to use \\ (double backslash), since Perl will remove the first Escape itself.

   Example: In most cases '\<tr>' but inside double-quotes ``\\<tr>"
2. **Turn off Escaping for all input by setting the optRawInput in EMBPERL_OPTIONS**
3. **Learn to avoid using HTML tags inside Perl code. Once you get the hang of it, you'll love it.**

Here is one example of how to do it:

```
[- $output = "<bold>Hello world</bold>" -]
[+ $output +]
```

write

```
[- $output = "Hello world<bold>" -]
```

this outputs

```
Hello world
```

or

```
<bold>[+ $output +]</bold>
```

this outputs

```
<bold>Hello world</bold>
```

And here is another example of how to do it:

```
[-
@a = ('a', 'b', 'c') ;
foreach $i (0..2)
    {
    $output. = "<tr><td>Row $a[$i]</td></tr>" ;
    }
-]
<table>
[+ $output +]
</table>
```

The output here would be:

```
<table>Row aRow bRow c</table>
```

The Embperl version is

```
[-
@a = ('a', 'b', 'c') ;
-]
<table>
<tr><td>Row [+ $a[$row] +]</td></tr>" ;
</table>
```

The output will be

```
<table>
<tr><td>Row a</td></tr>" ;
<tr><td>Row b</td></tr>" ;
<tr><td>Row c</td></tr>" ;
</table>
```

And another: This elegant solution shows you how to take advantage of Embperl's ability to create dynamic tables:

```
[-
use DBI;
my $dbh =
DBI->connect("DBI:mysql:database:localhost","Username","Password") ||
die($!);
$hstmt = $dbh->prepare("select ID, Heading from Shops order by Heading");
$hstmt->execute();
$dat = $hstmt->fetchall_arrayref() ;
$hstmt->finish();
$dbh->disconnect();
-]
<table border=1>
<tr><td>[+ $$dat[$row][$col] +]</td></tr>
</table>
```

This HTML code will then display the contents of the whole array.

Escaping & Unescaping Output

Embperl will also escape the output - so <H1> will be translated to <H1>

To see the exact steps taken by Embperl to process a Perl-laden document, please see Inside Embperl in the Embperl documentation.

Ways To Escape Output:

1. **) Escape it -> \\<H1>**

```
(You need a double backslash \\, because
the first one is removed by Perl and the second by Embperl.
```

2. **) set $escmode = 0 -> [- $escmode = 0 ; -]**
3. **) set SetEnv EMBPERL_ESCMODE 0 in your srm.conf**

[HOME] [CONTENT] [PREV (Common Questions)] [NEXT (Debugging)]

Debugging

[HOME] [CONTENT] [PREV (Escaping & Unescaping)] [NEXT (Customizing)]

I am having a hard time debugging Embperl code

Have you, umm, checked the error log? ;)

Have you tried setting debug flags higher by resetting EMBPERL_DEBUG in the server config files? (And still higher? :)

dbgMem isn't usually very useful as it always outputs a lot of allocation. dbgFlushLog and dbgFlushOutput should be used if (and only if) you are debugging SIGSEGVs.

For easy debugging, you can tell Embperl to display a link at the top of each page to your log file. Then every error displayed in an error page is a link to the corresponding position in the logfile, so you can easily find the place where something is going wrong

For more on using HTML links to the Embperl error log, see EMBPERL_DEBUG in the Embperldocs.

Embperl is running slow.

There are some debugging settings which may cause Embperl to drastically slow down. If you are done with debugging, set debugging bits back to normal.

Also, using dbgFlushLog and dbgFlushOutput will make execution much slower. These are only intended for debugging SIGSEGVs.

Never set all debugging bits!

How can I improve Embperl's performance?

1. **) Load Embperl at server startup. This will cause UNIX systems to only allocate memory once, and not for each child process. This reduces memory use, especially the need to swap additional memory.**
2. **) Disable all unneeded debugging flags. You should never set dbgFlushLog dbgFlushOutput, dbgMem and dbgEvalNoCache in a production environment.**
3. **) You may also want to take a look at the available options you can set via EMBPERL_OPTIONS. For example optDisableChdir, will speed up processing because it avoid the change directory before every request.**

[HOME] [CONTENT] [PREV (Escaping & Unescaping)] [NEXT (Customizing)]

Customizing

[HOME] [CONTENT] [PREV (Debugging)] [NEXT (Optimizing & Fine-Tuning)]

How can I fiddle with the default values? How can I override or alter this or that behavior?

Usually, defaults are set in a way that is likely to make most sense for a majority of users. As of version 1.0, Embperl allows much more flexibility in tweaking your own default values than before. Take a look at EMBERPL_OPTIONS.

I'd like to (temporarily) disable some of Embperl's features. What can be customized?

1. **) Use optDisableHtmlScan to disable processing of html tags. If this is set, Embperl will only pay attention to these types of constructs:**

   [+/-/!/$ .... $/!/-/+]
2. **) optDisableTableScan, optDisableInputScan and optDisableMetaScan can be used to disable individual parts of HTML processing.**

   You may set these flags in your server config, or at runtime:

```
[+ $optDisableHtmlScan = 1 +]
<table> foo </table>
[+ $optDisableHtmlScan = 0 +]
```

How can I disable auto-tables?

Set optDisableTableScan in EMBPERL_OPTIONS

How can I change predefined values like $escmode from my Toolbox module?

```
$HTML::Embperl::escmode = 0 ;
```

Predefined values in Embperl are simply aliases for $HTML::Embperl::foo (for instance, $escmode is an

alias for $HTML::Embperl::escmode)

How can I customize the header that Embperl is sending?

You'll find the answer to this and many other header issues in the Common Questions section.

How can I use a different character set? ASCII values over 128 are showing up as ? (question marks)!

This is caused by the translation of characters to HTML escapes. Embperl translates them to escapes which are then sometimes not understood by the browser, which may display a ``?'' instead, because it is using the wrong character set.

If you want to use the escaping features of Embperl in this case, you have to adapt the file `epchar.c` to your character set.

The distribution contain already an `epchar.c.iso-latin-2` from Jiri Novak which is an replacement for epchar.c for the iso-8859-2 (iso-latin-2) character set. If you want to use iso-latin-2, simply renmae `epchar.c.iso-latin-2` to `epchar.c`.

This file contains three tables:

Char2Html [] Convert characters to html escape

Char2Url [] Convert characters to url escapes (do not change this one!!)

Html2Char [] Convert html escapes to characters

You need to change the first and the last tables. Do not change the second table!!

Please make sure Char2Html contains one entry (and only one entry) for each of the 256 ascii codes (with none left undefined) in the right order, and that Html2Char is sorted by html escape.

If somebody generates new tables for national character sets, please send a copy to the author, so it can be included it in future versions of Embperl.

[HOME] [CONTENT] [PREV (Debugging)] [NEXT (Optimizing & Fine-Tuning)]

Optimizing & Fine-Tuning

[HOME] [CONTENT] [PREV (Customizing)] [NEXT (Additional Help)]

How can I be sure that Embperl is re-compiling my page template (and the Perl blocks contained in it) only when needed, and not each time?

As long as your input file's time stamp stays the same, Embperl will only compile the script the first time it's called. When you use the Execute function, Embperl will recompile the script only if the input file and mtime paramenters have changed since the last time the script was called.

You can verfiy this by setting dbgDefEval. Now, every time a Perl block is compiled, Embperl logs a line starting with DEF:. You will see this line only on the first request. The cached Perl blocks are stored as a set of subroutines in the namespace of the document. (HTML::Embperl::DOC::_<n> is the default) Look at the logfile to see the actual name.

How can I pre-compile pages, so that each httpd child doesn't have to have its own separate copies of the pre-compiled pages?

To pre-compile pages, just call Execute once for every file at server startup in your startup.pl file.

In what namespace does Embperl store pre-compiled data?

The cached Perl blocks are stored as a set of subroutines in the namespace of the document. (HTML::Embperl::DOC::_<n> for default) Look at the logfile to see the actual name.

I have both Embperl and ordinary Perl processes running. The docs say that Embperl uses a CGI.pm instance in its own internal processing, but they don't say how to control it. How can I get Embperl to use *my* CGI.pm object instead of creating its

own?

Embperl only creates a CGI objects to process multipart form data (from fileupload). In all other cases Embperl doesn't use CGI.pm. There is no way to change this behaviour, or access the internal CGI object in case of file-uploads.

[HOME] [CONTENT] [PREV (Customizing)] [NEXT (Additional Help)]

Additional Help

[HOME] [CONTENT] [PREV (Optimizing & Fine-Tuning)] [NEXT (AUTHOR)]

Where can I get more help?

You can get free support on the mod_perl mailing list. If you need commercial support (with a guarantee for response time or a solution) for Embperl, or if you want a web site where you can run your Embperl/mod_perl scripts without setting up your own web server, please send email to info@ecos.de.

Please also see the section Support in the Embperl documentation.

[HOME] [CONTENT] [PREV (Optimizing & Fine-Tuning)] [NEXT (AUTHOR)]

AUTHOR

[HOME] [CONTENT] [PREV (Additional Help)]

Gerald Richter <richter@ecos.de>

Edited by Nora Mikes <nora@radio.cz>

[HOME] [CONTENT] [PREV (Additional Help)]

SYNTAX

[HOME] [CONTENT] [PREV (Runtime configuration)] [NEXT (Variable scope and cleanup)]

Embperl understands two categories of commands. The first one are special Embperl commands, and the second category consists of some HTML tags which can trigger special processing. Embperl commands can span multiple lines and need not start or end at a line boundary.

Before the special Embperl commands are processed, and for the VALUE attribute of the INPUT tag (see below), all HTML tags are removed and special HTML characters are translated to their ASCII values (e.g., `<' is translated to `<'). You can avoid this behavior by preceding the special character or HTML tag with a backslash. This is done in case your favorite (WYSIWYG) HTML editor inserts tags like line breaks or formatting into your Embperl commands where you don't want them.

VERY IMPORTANT NOTE: If you do use an ASCII editor to write your HTML documents, you should set the option optRawInput so that Embperl does not preprocess your source. You can also HTML-escape your code (i.e. write `<' instead of `<'), to avoid ambiguity. In most cases it will also work without the optRawInput and HTML-escaping, but in some cases Embperl will detect an HTML tag were there isn't one.

If you have any trouble with your code, especially with HTML tags or filehandles in your Perl code, be sure to understand input- and output- escaping and unescaping. Read the section "Inside Embperl" to see what's going on!!

All Embperl commands start with a `[' and end with a `]'. To get a real `[' you must enter `[['.

Embperl does not use SGML comments (i.e., <! ... !> or similar things) because some HTML editors can't create them, or it's much more complicated. Since every HTML editor takes (or should take) `[' and `]' as normal text, there should be no problem.

[+ Perl code +]

Replace the command with the result you get from evaluating the Perl code. The Perl code can be anything which can be used as an argument to a Perl eval statement. (See (Safe-)Namespaces and opcode restrictions below for restrictions.) Examples:

```
[+ $a +]        Replaces the [+ $a +] with the content of
                the variable $a
[+ $a+1 +]      (Any expression can be used)
[+ $x[$i] +]    (Arrays, hashes, and more complex
                expressions work)
```

NOTE: **Whitespace is ignored. The output will be automatically HTML-escaped (e.g., `<' is translated to `<') depending on the value of the variables $escmode. You do not have to worry about it.**

[- Perl code -]

Executes the Perl code, but deletes the whole command from the HTML output.

Examples:

```
[- $a=1 -]            Set the variable $a to one.
                      No output will be generated.
[- use SomeModule -]  You can use other modules.
[- $i=0; while ($i<5) {$i++} -]  Even more complex
                                 statements or multiple
                                 statements are possible.
```

NOTE: **Statements like if, while, for, etc., must be contained in a single Embperl command. You cannot have the if in one command block and the terminating `}' or else in another.**

NOTE: **To define subroutines, use [! Perl Code !] (see below) instead of [- ... -] to avoid recompilation of the subroutine on every request.**

[! Perl Code !]

Same as [- Perl Code -] with the exception that the code is only executed at the first request. This could be used to define subroutines, or do one-time initialization.

[* Perl code *]

(only version 1.2b2 or higher)

This is similar to [- Perl Code -], the main difference is, while [- Perl Code -], has always it's own scope, all [* Perl code *] blocks runs in the same scope. This gives you the possibilty to define ``local'' variables with a scope of the whole page. Normaly you don't need to use local, because Embperl takes care of separate namespaces of different documents and cleanup after the request is finished, but in special cases it's necessary. For example if you want recursivly call an Embperl document via Execute.

There is a second reason to use the [* Perl code *] instead of the [- Perl Code -]. If you like to use perl's control structures. Perl's if, while, for etc. can not span mulitple [- Perl Code -] blocks, but it can span multiple [* Perl Code *].

```
Example:
[* foreach $i (1..10) { *]

  [- $a = $i + 5 -]
  loop count + 5 = [+ $a +] <br>
[* } *]
The following B<won't> work:
[- foreach $i (1..10) { -]
  some text here <br>
[- } -]
```

The same can be done with Embperl [meta commands](#) (see below)

```
[$ foreach $i (1..10) $]

  [- $a = $i + 5 -]
  loop count + 5 = [+ $a +] <br>
[$ endforeach $]
```

NOTE 1: [* ... *] blocks _must_ always end with a ;,{ or }

NOTE 2: [* ... *] cannot apear inside a html tag that is interpreted by Embperl (unless you disable the interpretation of such tags like table, input etc.)

NOTE 3: Since the execution of [- ... -] and metacommands is controlled by Embperl, there is a much better debugging output in the logfile for this two ones. Also no restriction where they can be used apply to meta-commands. You can use them anywhere even inside of html tags that are interpreted by Embperl.

[# Some Text #] (Comments)

(only version 1.2b2 or higher)

This is a comment block. Everything between the [# and the #] will be removed from the output.

NOTE 1: The [* ... *] blocks are interpreted before the comment block, so they are executed also inside a comment.

NOTE 2: Everything (except [* ... *]) is really removed from the source, so you can also use the [# ... #] block to take a part out of your document.

[$ Cmd Arg $] (Meta-Commands)

Execute an Embperl metacommand. Cmd can be one of the following. (Arg varies depending on <Cmd>).

if, elsif, else, endif

> **Everything following the if metacommand until the else, elsif, or endif is only output if the Perl expression given in Arg is true. else and elsif work similarly.**
>
> **Example:**
>
> ```
> [$ if $ENV{REQUEST_METHOD} eq 'GET' $]
> Method was GET<BR>
> [$ else $]
> Method other than GET used<BR>
> [$ endif $]
> ```
>
> **This will send one of the two sentences to the client, depending on the request method used to retrieve the document.**

while, endwhile

> **Executes a loop until the Arg given to while is false.**
>
> **Example: (see eg/x/loop.htm)**
>
> ```
> [- $i = 0; @k = keys %ENV -]
> [$ while ($i < $#k) $]
> [+ $k[$i] +] = [+ $ENV{$k[$i]} +]<BR>
> ```

```
[- $i++ -]
[$ endwhile $]
```

This will send a list of all environment variables to the client.

NOTE: The `<' is translated to `<' before calling Perl eval, unless optRawInput is set.

do, until

Executes a loop until the Arg given to until is true.

```
Example:
[- $i = 0 -]
[$ do $]
    [+ $i++ +] <BR>
[$ until $i > 10 $]
```

foreach, endforeach

Executes a loop for each element of the second Arg, setting the first Arg accordingly.

```
Example:
[- @arr = (1, 3, 5) -]
[$ foreach $v @arr $]
    [+ $v +] <BR>
[$ endforeach $]
```

hidden

Arg consists of zero, one or two names of hashes (with or without the leading %) and an optional array as third parameter. The hidden metacommand will generate hidden fields for all data contained in the first hash but not in the second hash. The default used for the first hash is `%fdat`**,** `%idat` **is used for the second.**

If the third parameter is specified, the fields are written in the order they appear in this array. That is, all keys of the first hash must be properly sorted in this array. This is intended for situations where you want to pass data from one form to the next, for example, two forms which should be filled in one after the other. (Examples might be an input form and a second form to review and accept the input, or a Windows-style ``wizard''). Here you can pass along data from previous forms in hidden fields. (See eg/x/neu.htm for an example.) If you use just the 'hidden' command without parameters, it simply generates hidden fields for all form fields submitted to this document which aren't already contained in another input field.

Example:

```
<FORM ACTION="inhalt.htm" METHOD="GET">
    <INPUT TYPE="TEXT" NAME="field1">
[$ hidden $]
</FORM>
```

If you request this with http://host/doc.htm?field1=A&field2=B&field3=C

the output will be

```
<FORM ACTION="inhalt.htm" METHOD="GET">
    <INPUT TYPE="TEXT" NAME="feld1" VALUE="A">

<INPUT TYPE="HIDDEN" NAME="field2" VALUE="B">
<INPUT TYPE="HIDDEN" NAME="field3" VALUE="C">
</FORM>
```

`NOTE:` **This should only be used for a small amount of data, since the hidden fields are sent to the browser, which sends it back with the next request. If you have a large amount of data, store it in a file with a unique name and send only the filename in a hidden field. Be aware of the fact that the**

data can be change by the browser if the user doesn't behave exactly as you expect. Users have a nasty habit of doing this all of the time. Your program should be able to handle such situations properly.

var

The var command declares one or more variables for use within this Embperl document and sets the strict pragma. The variable names must be supplied as space-separated list.

Example: [$var $a %b @c **$]**

This is the same as writing the following in normal Perl code:

```
use strict ;
use vars qw($a %b @c) ;
```

NOTE 1: `use strict' within an Embperl document will only apply to the block in which it occurs.

sub

(Only Embperl 1.2b5 and above)

Defines a Embperl subroutine. Example:

```
[$ sub foo $]
  <p> Here we do something </p>
[$ endsub $]
```

You can call this subroutine either as a normal Perl subroutine

```
[- foo -]
```

or via the HTML::Embperl::Execute function.

```
[- Execute ('#foo')           # short form -]
[- Execute ({ sub => 'foo'})  # long form  -]
```

The difference is that the Execute function will reset the internal states of Embperl like they were before the subrountine call, when the subroutine returns. Also Execute could handle recursive call, which currently not work when calling it as a Perl subroutine.

You may also pass Parameters to the subroutine:

```
[$ sub foo $]
  [- $p = shift -]
  <p> Here we show the first parameter [+ $p +]</p>
[$ endsub $]

[- foo ('value') -]
```

If you have a couple of commonly used subroutines you can define then in one file and import them into the modules where they are neccesary:

```
[- Execute ({ inputfile => 'mylib.htm', import => 1 }) -]
```

This will import all subroutines from the file *mylib.htm* **into the current page where they could call just as a normal Perl subroutine.**

HTML Tags

Embperl recognizes the following HTML tags specially. All others are simply passed through, as long as they are not part of a Embperl command.

TABLE, /TABLE, TR, /TR

Embperl can generate dynamic tables (one- or two-dimensional). You only need to specify one

row or column.

Embperl generates as many rows or columns as necessary. This is done by using the magic variables $row, $col, and $cnt. If you don't use $row/$col/$cnt within a table, Embperl does nothing and simply passes the table through.

Embperl checks if any of $row, $col, or `$cnt` is used. Embperl repeats all text between <table> and </table>, as long the expressions in which `$row` or `$cnt` occurs is/are defined.

Embperl repeats all text between <tr> and </tr>, as long the expressions in which `$col` or `$cnt` occurs is/are defined.

See also $tabmode (below) for end-of-table criteria.

Examples: (see eg/x/table.htm for more examples)

```
[- @k = keys %ENV -]
<TABLE>
    <TR>
        <TD>[+ $i=$row +]</TD>
        <TD>[+ $k[$row] +]</TD>
        <TD>[+ $ENV{$k[$i]} +]</TD>
    </TR>
</TABLE>
```

This will show all entries in array `@k` (which contains the keys from %ENV), so the whole environment is displayed (as in the while example), with the first column containing the zero-based index, the second containing the content of the variable name, and the third the environment variable's value.

This could be used to display the result of a database query if you have the result in an array. You may provide as many columns as you need. It is also possible to call a 'fetch' subroutine in each table row.

TH, /TH

The <TH> tag is interpreted as a table heading. If the whole row is made up of <TH> </TH> instead of <TD> </TD>, it is treated as a column heading. Everything else will be treated as row headings in the future, but are not now: everything else is ignored in the current version.

DIR, MENU, OL, UL, DL, SELECT, /DIR, /MENU, /OL, /UL, /DL, /SELECT

Lists and dropdowns or list boxes are treated exactly as one- dimensional tables. Only $row, $maxrow, $col, $maxcol and $tabmode are honored. `$col` and `$maxcol` are ignored. See eg/x/lists.htm for an example.

OPTION

Embperl checks if there is a value from the form data for a specific option in a menu. If so, this option will be pre-selected.

Example:

<FORM METHOD=``POST''> <P>Select Tag</P>

```
  If you request this document with list.htm?SEL1=x
  you can specify that the element which has a value
  of x is initially selected
  <P><SELECT NAME="SEL1">
     <OPTION VALUE="[+ $v[$row] +]">
        [+ $k[$row] +]
     </OPTION>
  </SELECT></P>
</FORM>
```

INPUT

The INPUT tag interacts with the hashes `%idat` und `%fdat`. If the input tag has no value, and a key exists with the same text as the NAME attribute of the input tag, Embperl will generate a VALUE attribute with the corresponding value of the hash key. All values of <INPUT> tags are stored in the hash `%idat`, with NAME as the hash key and VALUE as the hash value. Special processing is done for TYPE=RADIO and TYPE=CHECKBOX. If the VALUE attribute contains the same text as the value of the hash the CHECKED attribute is inserted, else it is removed.

So if you specify as the ACTION URL the file which contains the form itself, the form will be redisplayed with same values as entered the first time. (See eg/x/neu.htm for an example.)

TEXTAREA, /TEXTAREA

The TEXTAREA tag is treated exactly like other input fields.

META HTTP-EQUIV=

<meta http-equiv= ... > will over-ride the corresponding http header. This keeps Netscape from asking the user to reload the document when the content-type differs between the http header and the meta http-equiv.

This can also be used to set http headers. When running under mod_perl http-headers can also be set by the function header_out

```
Example of how to set a http header:
<META HTTP-EQUIV="Language" CONTENT="DE">
This is the same as using the Apache function
[- $req_rec ->  header_out("Language" => "DE"); -]
```

A, EMBED, IMG, IFRAME, FRAME, LAYER

The output of perl blocks inside the `HREF` attribute of the A Tags and the `SRC` attribute of the other Tags are URL escaped instead of HTML escaped. (see also *$escmode*). Also when inside such a URL, *Embperl* expands array refernces to URL paramter syntax. Example:

```
[-
%A = (A => 1, B => 2) ;
@A = (X, 9, Y, 8, Z, 7)
-]
<A HREF="http://localhost/tests?[+ [ %A ] +]">
<A HREF="http://localhost/tests?[+ \@A +]">
```

is expanded by *Embperl* to

```
<A HREF="http://localhost/tests?A=1&;B=2">
<A HREF="http://localhost/tests?X=9&;Y=8&Z=7">
```

[HOME] [CONTENT] [PREV (Runtime configuration)] [NEXT (Variable scope and cleanup)]

(Safe-)Namespaces and opcode restrictions

[HOME] [CONTENT] [PREV (Session handling)] [NEXT (Utility Functions)]

Since most web servers will contain more than one document, it is necessary to protect the documents against each other. Embperl does this by using Perl namespaces. By default, Embperl executes every document in its own namespace (package). This will prevent documents from accidentally overriding the other's data. You can change this behavior (or simply the package name) with the configuration directive **EMBPERL_PACKAGE. NOTE: By explicitly specifying a package name, you can access data that is used by another document.**

If Embperl is used by more then one person, it may be neccessary to really protect documents from each other. To do this, Embperl gives you the option of using safe namespaces. Each document runs in its own package and can't access anything outside of this package. (See the documentation of Safe.pm for a more detailed discussion of safe namespaces.)

To make a document run in a safe namespace, simply add optSafeNamespace to EMBPERL_OPTIONS. The default package name used is the same as in normal operation and can be changed with EMBPERL_PACKAGE. NOTE: From the perspective of the document being executed, the code is running in the package main!

A second option to make Embperl more secure is the use of the opcode restriction mask. Before you can use the opcode mask, you must set up a safe compartment.

```
B<$cp = HTML::Embperl::AddCompartment($name);>
```

This will create a new compartment with a default opcode mask and the name $name. (The name is used later to tell Embperl which compartment to use.) Now you can change the operator mask. For example:

```
B<$cp->deny(':base_loop');>
```

In your configuration you must set the option optOpcodeMask in EMBPERL_OPTIONS and specify from which compartment the opcode mask should be taken by setting EMBPERL_COMPARTMENT.

```
Example (for use with mod_perl):
   B<srm.conf:>
   PerlScript startup.pl
   SetEnv EMBPERL_DEBUG 2285
   Alias /embperl /path/to/embperl/eg
   <Location /embperl/x>
   SetHandler perl-script
   PerlHandler HTML::Embperl
   Options ExecCGI
   PerlSetEnv EMBPERL_OPTIONS 12
   PerlSetEnv EMBPERL_COMPARTMENT test
   </Location>
   B<startup.pl:>
   $cp = HTML::Embperl::AddCompartment('test');
   $cp->deny(':base_loop');
```

This will execute the file startup.pl on server startup. startup.pl sets up a compartment named `test', which will have a default opcode mask and additionaly, will have loops disabled. Code will be executed in a safe namespace.

NOTE: The package name from the compartment is NOT used!

Look at the documentation of Safe.pm and Opcode.pm for more detailed information on how to set opcode masks.

[HOME] [CONTENT] [PREV (Session handling)] [NEXT (Utility Functions)]

Inside Embperl - How the embedded Perl code is actually processed

[HOME] [CONTENT] [PREV (Input/Output Functions)] [NEXT (Performance)]

If Embperl encounters a piece of Perl code **([+/-/!/$ .... $/!/-/+]) it takes the following steps.**

1. **Remove anything which looks like an HTML tag**
2. **Translate HTML escapes to their corresponding ASCII characters**
3. **Remove all carriage returns**
4. **Eval the Perl code into a subroutine**
5. **Call the subroutine**
6. **Escape special characters in the return value**
7. **Send the return value as output to the destination (browser or file)**

Steps 1-4 take place only the first time the Perl code is encountered. Embperl stores the eval'ed subroutine, so all subsequent requests only need to execute steps 5-7.

Steps 6 and 7 take place only for code surrounded by [+ ... +].

What does this mean?

Let's take a piece of code like the following:

```
[+ <BR>
$a = "This '&gt;' is a greater-than sign"
<BR> +]
```

---

1. Remove the HTML tags. Now it looks like

```
[+
$a = "This '&gt;' is a greater-than sign"
+]
```

**The
s were inserted by some WYSIWYG HTML editor (e.g., by hitting return to make the source more readable. Also, such editors often generate ``random'' tags like <FONT>, etc.). Embperl removes them so they don't cause syntax errors.**

There are cases where you actually want the HTML tag to be there. For example, suppose you want to output something like

```
[+ "<FONT COLOR=$col>" +]
```

If you write it this way, Embperl will just remove everything, leaving only

```
[+ "" +]
```

There are several ways to handle this correctly.

```
a. <FONT COLOR=[+$col+]>
   Move the HTML tag out of the Perl code. This is the best way, but
   it is not possible every time.
b. [+ "\<FONT COLOR=$col>" +]
   You can escape the opening angle bracket of the tag with `\'.
c. [+ "&lt;FONT COLOR=$col&gt;" +]
   You can use the HTML escapes instead of the ASCII characters.
   Most HTML editors will automatically do this.  (In this case,
   you don't have to worry about it at all.)
d. Set optRawInput (see below).
   This will completely disable the removal of HTML tags.
```

NOTE: In cases b-d, you must also be aware of output escaping (see below).

You should also be aware that Embperl will interpret the Perl spaceship operator (<>) as an HTML tag and will remove it. So instead of

```
[- $line = <STDIN>; -]
```

you need to write either

```
a. [- $line = \<STDIN>; -]
b. [- $line = &lt;STDIN&gt;; -]
```

Again, if you use a high-level HTML editor, it will probably write version (b) for you automatically.

---

2. Translate HTML escapes to ASCII characters

Since Perl doesn't understand things like `$a < $b`**, Embperl will translate it to** `$a < $b`**. If we take the example from earlier, it will now look like**

```
[+
$a = "This '>' is a greater sign"
+]
```

This step is done to make it easy to write Perl code in a high-level HTML editor. You do not have to worry that your editor is writing > instead of > in the source.

Again, sometimes you need to have such escapes in your code. You can write them

```
a. \&gt;
   Escape them with a `\' and Embperl will not translate them.
b. &amp;gt;
   Write the first `&' as its HTML escape (&amp;).  A normal HTML
   editor will do this on its own if you enter &gt; as text.
c. Set optRawInput (see below)
   This will completely disable the input translation.
```

Since not all people like writing in a high level or WYSIWYG HTML editor, there is an option to disable steps 1 and 2. You can use the optRawInput in EMBPERL_OPTIONS to tell Embperl to leave the Perl code as it is. It is highly recommended to set this option if you are writing your HTML in an ASCII editor. You normally don't want to set it if you use some sort of high level HTML editor.

You can also set the optRawInput in your document by using $optRawInput, but you must be aware that it does not have any consequences for the current block, because the current block is translated

before it is executed. So write it in separate blocks:

```
[- $optRawInput = 1 -]
[- $line = <FILEHANDLE> -]
```

---

3. Remove all carriage returns

All carriage returns (\r) are removed from the Perl code, so you can write source on a DOS/Windows platform and execute it on a UNIX server. (Perl doesn't like getting carriage returns in the code it parses.)

---

4. Eval perl code into a subroutine

The next step generates a subroutine out of your Perl code. In the above example it looks like:

sub foo { `$a = ``This '>' is a greater sign'' }`

The subroutine is now stored in the Perl interpreter in its internal precompiled format and can be called later as often as necessary without doing steps 1-4 again. Embperl recognizes if you request the same document a second time and will just call the compiled subroutine. This will also speed up the execution of dynamic tables and loops, because the code inside must be compiled only on the first iteration.

---

5. Call the subroutine

Now the subroutine can be called to actually execute the code.

If Embperl isn't executing a [+ ... +] block we are done. If it is a [+ ... +] block, Embperl needs to generate output, so it continues.

---

6. Escape special characters in the return value

Our example returns the string:

``This '>' is a greater sign''

The greater sign is literal text (and not a closing html tag), so according to the HTML specification it must be sent as > to the browser. In most cases, this won't be a problem, because the browser will display the correct text if we send a literal '>'. Also we could have directly written > in our Perl string. But when the string is, for example, the result of a database query and/or includes characters from national character sets, it's absolutely necessary to send them correctly-escaped to the browser to get the desired result.

A special case is the <A> HTML tag. Since it includes a URL, the text must be URL-escaped instead of HTML-escaped. This means special characters like `&' must be sent by their hexadecimal ASCII code and blanks must be translated to a `+' sign. If you do not do this, your browser may not be able to interpret the URL correctly.

Example:

```
<A HREF="http://host/script?name=[+$n+]">
```

When `$n` **is ``My name'' the requested URL, when you click on the hyperlink, will be**

```
http://host/script?name=My+name
```

In some cases it is useful to disable escaping. This can be done by the variable $escmode.

Example: (For better readability, we assume that optRawInput is set. Without it, you need to cover the Embperl pre-processing described in steps 1-3.)

```
[+ "<FONT COLOR=5>" +]

This will be sent to the browser as &lt;FONT COLOR=5&gt;, so you
will see the tag on the browser screen instead of the browser
switching the color.

[+ local $escmode=0 ; "<FONT COLOR=5>" +]

This will (locally) turn off escaping and send the text as a plain
HTML tag to the browser, so the color of the output will change.

NOTE: You cannot set $escmode more than once inside a [+ ... +] block.
Embperl uses the first setting of $escmode it encounters inside the block.
If you need to change $escmode more than once, you must use muliple
[+ ... +] blocks.
```

---

7. Send the return value as output to the destination (browser/file)

Now everything is done and the output can be sent to the browser. If you haven't set dbgEarlyHttpHeaders, the output is buffered until the successful completion of document execution of the document, and is sent to the browser along with the HTTP headers. If an error occurs, an error document is sent instead.

The content length and every <META HTTP-EQUIV=...> is added to the HTTP header before it is sent. If Embperl is executed as a subrequest or the output is going to a file, no http header is sent.

[HOME] [CONTENT] [PREV (Input/Output Functions)] [NEXT (Performance)]

Runtime configuration

[HOME] [CONTENT] [PREV (Operating-Modes)] [NEXT (SYNTAX)]

The runtime configuration is done by setting environment variables, either on the command line (when working offline) or in your web server's configuration file. Most HTTP servers understand

SetEnv <var> <value>

If you are using Apache and mod_perl you can use

PerlSetEnv <var> <value>

The advantage of PerlSetEnv over SetEnv is that it can be used on a per directory/virtual host basis.

EMBPERL_FILESMATCH

If specified, only files which match the given **perl regular expression will be processed by Embperl, all other files will be handled by the standard Apache handler. This can be useful if you have Embperl documents and non Embperl documents (e.g. gifs) cohabitating in the same directory. EMBPERL_FILESMATCH works only under mod_perl.**

```
Example:
# Only files which end with .htm will processed by Embperl
PerlSetEnv EMBPERL_FILESMATCH \.htm$
```

EMBPERL_ALLOW (only 1.2b10 and above)

If specified, only files which match the given perl regular expression will be processed by Embperl, all other files will return FORBIDDEN. Especialy in a CGI environenemt this can be usefull to make a server more secure.

EMBPERL_COMPARTMENT

Gives the name of the compartment from which to take the opcode mask. (See the chapter about (Safe-)Namespaces and opcode restrictions for more details.)

EMBPERL_ESCMODE

Specifies the initial value for $escmode (see below).

EMBPERL_LOG

Gives the location of the log file. This will contain information about what Embperl is doing. How much information depends on the debug settings (see EMBPERL_DEBUG below). The log output is intended to show what your embedded Perl code is doing and to help debug it. The default is /tmp/embperl.log.

NOTE: When running under mod_perl you need to use PerlSetEnv for setting the logfile path, and mod_perl >= 1.07_03 if you load Embperl at server startup (with PerlScript or PerlModule).

EMBPERL_PACKAGE

The name of the package where your code will be executed. By default, Embperl generates a unique package name for every file. This ensures that variables and functions from one file can not affect those from another file. (Any package's variables will still be accessible with explicit package names.)

EMBPERL_VIRTLOG

Gives a virtual location where you can access the Embperl logfile with a browser. This feature is disabled (default) if EMBPERL_VIRTLOG is not specified. See also EMBPERL_DEBUG and dbgLogLink for an Example how to set it up in your srm.conf.

EMBPERL_OPTIONS

This bitmask specifies some options for the execution of Embperl. To specify multiple options, simply add the values together.

optDisableVarCleanup = 1

> **Disables the automatic cleanup of variables at the end of each request.**

optDisableEmbperlErrorPage = 2

Tells Embperl to not send its own errorpage in case of failure, instead shows as much of the page as possible. Errors are only logged to the log file. Without this option, Embperl sends its own error page, showing all the errors which have occurred. If you have dbgLogLink enabled, every error will be a link to the corresponding location in the log file. This option has no effect if optReturnError is set.

optReturnError = 262144

With this option set Embperl sends no output in case of an error, instead it returns the error back to Apache or the calling programm. When running under mod_perl this gives you the chance to use the Apache *ErrorDocument* **directive to show a custom error-document.**

optSafeNamespace = 4

Tells Embperl to execute the embedded code in a safe namespace so the code cannot access data or code in any other package. (See the chapter about (Safe-)Namespaces and opcode restrictions below for more details.)

optOpcodeMask = 8

Tells Embperl to apply an operator mask. This gives you the chance to disallow special (unsafe) opcodes. (See the Chapter about (Safe-)Namespaces and opcode restrictions below for more details.)

optRawInput = 16

Causes Embperl not to pre-process the source for a Perl expression. (The only exception is that carriage returns will be removed, as Perl does not like them.) This option should be set when you are writing your code with an ASCII editor.

If you are using a WYSIWYG editor which inserts unwanted HTML tags in your Perl expressions and escapes special charcaters automatically (e.g., `<' appears as `<' in the source), you should not set this option. Embperl will automatically convert the HTML input back to the Perl expressions as you wrote them.

optEarlyHttpHeader = 64

Normally, HTTP headers are sent after a request is finished without error. This gives you the chance to set arbitrary HTTP headers within the page, and gives Embperl the chance to calculate the content length. Also Embperl watches out for errors and sends an errorpage instead of the document if something goes wrong. To do this, all the output is kept in memory until the whole request is processed, then the HTTP headers are sent, and then the document. This flag will cause the HTTP headers to be sent before the script is processed, and the script's output will be sent directly.

optDisableChdir = 128

Without this option, Embperl changes the currect directory to the one where the script resides. This gives you the chance to use relative pathnames. Since directory-changing takes up some millisecs, you can disable it with this option if you don't need it.

optDisableFormData = 256

This option disables the setup of `%fdat` and @ffld. Embperl will not do anything with the posted form data.

optDisableHtmlScan = 512

When set, this option disables the scanning of all html-tags. Embperl will only look for [+/-/!/$ ... $/!/-/+]. This will disable dynamic tables, processing of the input data and so on.

optDisableInputScan = 1024

Disables processing of all input-related tags. (<INPUT><TEXTAREA><OPTION>)

optDisableTableScan = 2048

Disables processing of all table-related tags. (<TABLE><TH><TR><TD><MENU><OL><SELECT><UL>)

optDisableMetaScan = 4096

Disables processing of all meta tags. (<META HTTP-EQUIV>)

optAllFormData = 8192

This option will cause Embperl to insert all formfields in `%fdat` and @ffld, even if they are empty. Empty formfields will be inserted with an empty string. Without this option empty formfields will not be insert in `%fdat` and @ffld.

optRedirectStdout = 16384

Redirects STDOUT to the Embperl output stream before every request and resets it afterwards. If set, you can use a normal Perl print inside any Perl block to output data. Without this option you can only output data by using the [+ ... +] block, or printing to the filehandle OUT.

optUndefToEmptyValue = 32768

Normally if there is no value in `%fdat` for a specific input field, Embperl will leave it untouched. When this option is set, Embperl will handle the field as if an empty string was stored in `%fdat` for the field.

optNoHiddenEmptyValue = 65536 (only 1.2b2 and above)

Normally, if there is a value defined in `%fdat` for a specific input field, Embperl will output a hidden input element for it when you use hidden. When this option is set, Embperl will not output a hidden input element for this field when the value is a blank string.

optAllowZeroFilesize = 131072 (only 1.2b2 and above)

Normaly Embperl reports NOT_FOUND (404) if a file of length zero is requested. With this option set Embperl will return an empty document.

optKeepSrcInMemory = 524288 (only 1.2b5 and above)

Tells Embperl to keep the source file in memory and not reload it on every request. (The precompiled Perlcode is always kept in memory, regardless of this flag)

optKeepSpaces = 1048576 (only 1.2b5 and above) = 0x100000,

Disable the removing of spaces and empty lines from the output. This is usefull for other sources then HTML.

optOpenLogEarly = 2097152 (only 1.2b5 and above)

This option causeses Embperl to open the logfile as soon as it is loaded into memory. You can use this when you load Embperl via PerlModule under Apache, to open the log as root instead of the non-privileged user Apache runs as. =item optUncloseWarn = 4194304 (only 1.2b6 and above)

Disable the warnings about unclosed `if`, `while`, `table` etc. at the end of the file.

EMBPERL_DEBUG

This is a bitmask which specifies what should be written to the log. To specify multiple debugflags, simply add the values together. The following values are defined:

dbgStd = 1

Show minimum information.

dbgMem = 2

Show memory and scalar value allocation.

dbgEval = 4

Show arguments to and results of evals.

dbgCmd = 8

Show metacommands and HTML tags which are processed.

dbgEnv = 16,

List every request's environment variables.

dbgForm = 32

Lists posted form data.

dbgTab = 64

Show processing of dynamic tables.

dbgInput = 128

Show processing of HTML input tags.

dbgFlushOutput = 256

Flush Embperl's output after every write. This should only be set to help debug Embperl crashes, as it drastically slows down Embperl's operation.

dbgFlushLog = 512

Flush Embperl's logfile output after every write. This should only be set to help debug Embperl crashes, as it drastically slows down Embperl's operation.

dbgAllCmds = 1024

Logs all commands and HTML tags, whether or not they are really excuted or not. (It logs a `+' or `-' to tell you if they are executed.)

dbgSource = 2048

Logs the next piece of the HTML source to be processed. (NOTE: This generates a lot of output!)

dbgFunc = 4096

This is only anvailable when Embperl is compiled with -DEPDEBUGALL, and is normally only used for debugging Embperl itself. Records all function entries to the logfile.

dbgLogLink = 8192

Inserts a link at the top of each page which can be used to view the log for the current HTML file. See also EMBPERL_VIRTLOG.

Example:

```
SetEnv EMBPERL_DEBUG 10477
SetEnv EMBPERL_VIRTLOG /embperl/log
<Location /embperl/log>
SetHandler perl-script
PerlHandler HTML::Embperl
Options ExecCGI
</Location>
```

dbgDefEval = 16384

Shows every time new Perl code is compiled.

dbgCacheDisable = 32768

Disables the use of the p-code cache. All Perl code is recompiled every time. (This should not be used in normal operation as it slows down Embperl dramatically.) This option is only here for debugging Embperl's cache handling. There is no guarantee that Embperl behaves the same with

and without cache (actually is does not!)

dbgHeadersIn = 262144

Log all HTTP headers which are sent from the browser.

dbgShowCleanup = 524288

Show every variable which is undef'd at the end of the request. For scalar variables, the value before undef'ing is logged.

dbgProfile = 1048576 (only 1.2b4 and above)

When set every source line in the log file will also display the time until request was started. (dbgSource must also be set)

dbgSession = 2097152 (only 1.2b4 and above)

Enables logging of session transactions.

dbgImport = 4194304 (only 1.2b5 and above)

Show how subroutines are imported in other namesapces

A good value to start is `2285` or `10477` if you want to view the logfile with your browser. (Don't forget to set EMBPERL_VIRTLOG.) If Embperl crashes, add `512` so the logfile is flushed after every line is written and you can see where Embperl is when it crashes.

EMBPERL_INPUT_FUNC

This directive gives you the possiblity to specify a non-standard way of fetching input. Normally, Embperl reads its input (source) from a file (or gets it from a scalar if you use `Execute`). Here, you can give the name of a Perl function which is called instead of reading the input from a file. The function must look like the following:

```
InputFunc ($r, $in, $mtime, additional parameters...) ;
```

$r

Apache Request Record (see perldoc Apache for details)

$in

a reference to a scalar, to which the input should be returned.

```
Example:
open F, "filename" ;
local $\ = undef ;
$$in = <F> ;
close F ;
```

$mtime

a reference to a scalar, to which the modification time should be returned.

```
Example:
$$mtime = -M "filename" ;
```

You can give additional parameters (which must be comma-separated) to EMBPERL_INPUT_FUNC which will then pass them as a string.

```
Example:
PerlSetEnv EMBPERL_INPUT_FUNC "InputFunc, foo, bar"
will call
InputFunc ($r, $in, $mtime, 'foo', 'bar') ;
```

```
  to get the input.
```

EXAMPLE for an input function which does just the same as Embperl

```
 sub Input
    {
    my ($r, $in, $mtime) = @_ ;
    open F, $r -> filename or return NOT_FOUND ;
    local $\ = undef ;
    $$in = <F> ;
    close F ;
    $$mtime = -M $r -> filename ;

    return 0 ;
    }
```

See also ProxyInput below, for an input function which comes with Embperl.

NOTE: There are also two modules (*HTML::EmbperlChain* and *Apache::EmbperlFilter*) which provides you with the possibility to chain *Embperl* and other modules together.

EMBPERL_OUTPUT_FUNC

This directive allows you to specify a non-standard way of dealing with output. Normally, Embperl sends its output (source) to a file/the browser (or to a scalar if you use `Execute`). Here, you can give the name of a Perl function which is called instead of sending the output to a file/the browser. The function must look like the following:

```
 OutputFunc ($r, $out, additional parameters...) ;
```

$r

> **Apache Request Record (see perldoc Apache for details)**

$out

> **a reference to a scalar, which contains the output from Embperl**

You can give additional parameters (which must be comma-separated) to EMBPERL_OUTPUT_FUNC, which will then pass them as a string.

```
  Example:
  PerlSetEnv EMBPERL_OUTPUT_FUNC "OutputFunc, foo, bar"
  will call
  OutputFunc ($r, $out, 'foo', 'bar') ;
  for output.
```

EXAMPLE for an ouput function which does just the same as Embperl

```
 sub Output
    {
    my ($r, $out) = @_ ;
    $r -> send_http_header ;
    $r -> print ($$out) ;
    return 0 ;
    }
```

See also LogOutput below, for an output function which comes with Embperl.

NOTE: There are also two modules (*HTML::EmbperlChain* and *Apache::EmbperlFilter*) which provides you with the possibility to chain *Embperl* and other modules together.

EMBPERL_MAILHOST

Specifies which host the MailFormTo function uses as SMTP server. Default is localhost.

EMBPERL_MAIL_ERRORS_TO

If set all errors will be send to the email adress given.

EMBPERL_COOKIE_NAME

(only 1.2b4 or above) Set the name that Embperl uses when it sends the cookie with the session id. Default is EMBPERL_UID.

EMBPERL_COOKIE_DOMAIN

(only 1.2b4 or above) Set the domain that Embperl uses for the cookie with the session id. Default is none.

EMBPERL_COOKIE_PATH

(only 1.2b4 or above) Set the path that Embperl uses for the cookie with the session id. Default is none.

EMBPERL_COOKIE_EXPIRES

(only 1.2b4 or above) Set the expiration date that Embperl uses for the cookie with the session id. Default is none.

EMBPERL_SESSION_CLASSES

Space separted list of object store and lock manager for Apache::Session (see Session handling)

EMBPERL_SESSION_ARGS

List of arguments for Apache::Session classes (see Session handling) Example:

```
  PerlSetEnv EMBPERL_SESSION_ARGS "DataSource=dbi:mysql:session UserName=www
Password=secret"
```

[HOME] [CONTENT] [PREV (Operating-Modes)] [NEXT (SYNTAX)]

Compatibility

[HOME] [CONTENT] [PREV (Bugs)] [NEXT (Support)]

I have tested Embperl succesfully

---

on Linux 2.x with

perl5.004_04
perl5.005_03
apache_1.2.5
apache_1.2.6
apache_1.3.0
apache_1.3.1
apache_1.3.2
apache_1.3.3
apache_1.3.4
apache_1.3.5
apache_1.3.6
apache_ssl (Ben SSL)
Stronghold 2.2
Stronghold 2.4.1
Apache_1.3.3 with mod_ssl 2.0.13
Apache_1.3.6 with mod_ssl 2.3.5

I know from other people that it works on many other UNIX systems

---

on Windows NT 4.0 with

perl5.004_04
perl5.005
apache_1.3.1
apache_1.3.6

---

on Windows 95/98 with

perl5.004_02 (binary distribution, only Offline Mode)
perl5.005_02 + apache_1.3.6

[HOME] [CONTENT] [PREV (Bugs)] [NEXT (Support)]

Bugs

[HOME] [CONTENT] [PREV (Performance)] [NEXT (Compatibility)]

None known.

Under perl5.004 there are memory leaks. This is not an Embperl bug, but can cause your httpd to grow endlessly when running under mod_perl. Please upgrade to perl5.004_04 to fix this. You should also upgrade to a mod_perl version higher than 1.07_01 as soon as possible, because until 1.07_01 there is a memory leak in Apache->push_handler.

[HOME] [CONTENT] [PREV (Performance)] [NEXT (Compatibility)]

Author

[HOME] [CONTENT] [PREV (References)]

G. Richter (richter@dev.ecos.de)

[HOME] [CONTENT] [PREV (References)]

CVS - Content

[HOME] [CONTENT] [NEXT (CVS - Access to the Embperl CVS development tree)]

[HOME] [CONTENT] [NEXT (CVS - Access to the Embperl CVS development tree)]

Sites running Embperl - Content

[HOME] [CONTENT] [NEXT (Sites running Embperl - What people say about Embperl)]

[HOME] [CONTENT] [NEXT (Sites running Embperl - What people say about Embperl)]

Embperl and DBIx::Recordset

[HOME] [CONTENT] [PREV (Recordset Introduction - Content)] [NEXT (Basic Example)]

This introduction gives an overview how to use *DBIx::Recordset* **together with** *HTML::Embperl***. Since there are only a few** *Embperl* **specific things herein, it should be also usefull for non** *Embperl* **users.**

Overview

It is often very difficult to layout and design the output of normal CGI scripts, because you are dealing with HTML-sourcecode which spans multiple prints, and it isn't possible to use some sort of HTML-editor. Embperl takes a different approach to this problem. With Embperl, you can build your HTML-pages with any tool you like, and you can embed fragments of code directly in the page. This makes it much easier for non-programmers to use, because they are able to use their usual tools and they see the fragments of code as normal text. This indroduction will deal with the Perl Modules *HTML::Embperl* **and** *DBIx::Recordset***, with a focus on database access.**

Embperl

In brief, the purpose of Embperl is to execute code that is embedded in HTML-pages as the page is requested from the server. There are two ways to do this with Embperl. The first way is to embed the code between [- and -] tags. This will cause Embperl to execute the code and remove it from the source before sending the page. The second way is to use [+ and +] as the delimiter, in which case the code will be executed and the result of the execution is send to the browser in place of the code. All database access is done via the module *DBIx::Recordset***, which simplifies a lot of common tasks when accessing a database via DBI.**

[HOME] [CONTENT] [PREV (Recordset Introduction - Content)] [NEXT (Basic Example)]

Basic Example

[HOME] [CONTENT] [PREV (Embperl and DBIx::Recordset)] [NEXT (Multiple tables)]

The following example shows the basic functions of both modules. It shows the contents of a table whose name is passed as a parameter:

<body> <h1>Contents of table ``[+ $fdat{'!Table'} +]''</h1>

```
  [-
  use DBIx::Recordset ;

  $fdat{'!DataSource'} = 'dbi:mysql:test' ;
  *set = DBIx::Recordset -> Search(\%fdat) ;
  $names = $set -> Names ;
  -]
  <table>
    <tr>
      <th>[+ $names -> [$col] +]</th>
    </tr>
    <tr>
      [- $rec = $set[$row] -]
      <td>[+ $rec -> {$names->[$col]} +]</td>
    </tr>
  </table>
</body>
```

To show the contents of the table `address` you may call it with:

`http://www.domain.com/path/to/example1.htm?!Table=address`

All query parameters are placed in the hash `%fdat` by Embperl. In our example, `$fdat{'!Table'}` would contain the value `address`. Additionally, Embperl replaces the code between `[+` and `+]` with the result, so the headline of the page would be 'Contents of table ``address'''.

The following [- -] block will be executed by Embperl. No trace of it will show up in the page that is sent to the browser. The first line sets the database which should be accessed. The syntax is the same as for the DBI connect call. If you omit the line, you must additionally send the databasename as a query parameter - but for security reasons, that isn't a very good idea.

Search

Next we call the method `Search` of *DBIx::Recordset*, **where we have the choice between the object and the class-method. This applies to a lot of other methods as well. When we call it as a class method, as we do in our example, it constructs a new** *DBIx::Recordset* **object and uses the passed parameters to query the database. It's also possible to divide these two steps and call** `Setup` **to first construct the object and then** `Search` **with this object to execute the Search. In the example above, we do not pass any query parameters -- so** `Search` **will return the contents of the whole table. (***DBIx::Recordset*

converts the call internally to the SQL statement `SELECT * FROM address`**).**

The last line of the [- -] block retrieves the fieldnames of the table. Here we can see a special feature of *DBIx::Recordset***, which we will discuss in detail later on. The constructor returns a typeglob (**`*set`**), but the call to** `Names` **uses a scalar (**`$set`**). By returning a typeglob,** *DBIx::Recordset* **is able to return a scalar, an array and a hash at the same time. (If you don't like the idea of using typeglobs, you can also construct all three with different methods).**

Display the table

At first glance, the following might appear to be a simple HTML-table. But *Embperl* **expands it, so that the full contents of the database table is shown. Let us first look at the header, which should show the fieldnames of the database-table:** `$names` **contains a reference to an array which contains the fieldnames.** *Embperl* **gives us the magical variable** `$col`**.** `$col` **will be automatically incremented as long as the result of the expression which contains** `$col` **doesn't return undefined. At the same time,** *Embperl* **repeats the surrounding** `<th>` **or** `<td>` **tags. If we have a table with the three columns** `name`**,** `firstname` **and** `town`**, the output would look like this:**

```
<th>name</th><th>firstname</th><th>town</th>
```

Now the header is ready and we can start to output the contents. Here we use the array part of the typeglob that is returned by `Search`**. Access to the results of the SQL-query is done via the array** `@set`**, and every row of the array ``contains'' one row of the database-table. It does not really contain the row, but** *DBIx::Recordset* **will fetch the row from the databases for you if you access the corresponding array row. The rows are stored as a hash, where the fieldnames are the hashkeys. This is the same mechanism that helped us to expand the columns of the header, but it's at work here in a two-dimensional manner.** `$row` **contains the row-count and** `$col` **contains the column-count.**

Supplying query parameters

But our small example can do even more: If we supply more query parameters in our request, we can decide which parts of the table should be selected (and therefor, shown). If we request the page with

`http://wwww.domain.com/path/to/example1.htm?!Table=address&;town=Berlin`

Embperl **will not only place** `!Table` **in the hash** `%fdat`**, but also** `town`**. Since** `town` **corresponds to a fieldname in our table,** *DBIx::Recordset* **interprets it as a parameter for the** `WHERE` **part of the** `SELECT` **command.** *DBIx::Recordset* **will generate the following SQL-query:**

```
SELECT * FROM address WHERE town='Berlin' ;
```

The programmer doesn't have to pay attention to datatypes or quoting, this is done automatically by *DBIx::Recordset***.**

Also, complex queries are easy to implement: if, for example, the user wants to be able to search for a name or for a town, it would be possible to use the following form:

```
<form action="/path/to/example1.htm" method=GET >
  <input type=text name="+name|town">
  <input type=hidden name="!Table" value="address">
  <input type=submit>
</form>
```

If the user enters ``Richter'' to the input field and presses the submit button, the following SQL-query will be generated:

```
SELECT * FROM address WHERE name='Richter' OR town='Richter' ;
```

Just by varying the parameters, it is possible to create simple or complex queries. In this way, you can use the same page with different parameters to create different sorts of queries.

[HOME] [CONTENT] [PREV (Embperl and DBIx::Recordset)] [NEXT (Multiple tables)]

Multiple tables

[HOME] [CONTENT] [PREV (Basic Example)] [NEXT (Modify the Database)]

- DBIx::Database
- Sub-Objects

Until now, we only have worked with one table. In real life, you often have to deal with mulitple tables. For this reason, *DBIx::Recordset* **helps you to reduce the expense associated with dealing with multiple tables. The simplest way to do this is to use the parameters** `!TabJoin` **and** `!TabRelation` **to tell** *DBIx::Recordset* **to create an SQL-join between two or more tables. This will link the tables together and the result looks just like one great table.**

More interesting is the possibility to create "links". As an example, we'll take the same table we used above and divide it into two tables: one table for the names and one table for the towns. As a link we add an id-field. If the fields are following some naming convention, *DBIx::Recordset* **is able to find this link automatically. If fields are named in another way, you have to tell** *DBIx::Recordset* **manually how the tables belong together.**

```
Table name:       firstname, name, town_id
Table town:       id, town
```

Here, every name has exactly one town and every town has a number of names assigned. With a simple modification of our first example, we could get the same result as above (except that we are now dealing with two tables instead of one):

```
[-
use DBIx::Recordset ;
$db = DBIx::Database -> new ('dbi:mysql:test') ;
$db -> TableAttr ('ort', '!NameField', 'town') ;
$fdat{'!DataSource'} = $db ;
$fdat{'!LinkName'} = 3 ;
*set = DBIx::Recordset -> Search(\%fdat) ;
$names = $set -> Names ;
-]
```

And the request would be:

`http://www.domain.com/path/to/example2.htm?!Table=name`

DBIx::Database

The new thing here is the `DBIx::Database` **object. It gathers meta-information about the database and stores it for later use. Because of the names of the fields the object can detect that the field** `town_id` **in the table** `name` **points to field** `id` **in the table** `town`**. Additionally, we tell the** `DBIx::Database` **object which** `column(s)` **contain the human-readable name of the table** `town`**. These initialisations only have to be executed once. If you use** *mod_perl*, **for example, you should be able to move these lines into a common startup file.**

Also new is the parameter `!LinkName`**. It tells** *DBIx::Recordset* **to return the human-readable name (in our example,** `town`**) instead of the field which links the two tables together (**`town_id` **in our example). Internally,** *DBIx::Recordset* **generates an SQL-join, so there is only one** `SELECT` **command necessary**

and the result is just the same as in the last example.

Sub-Objects

But what to do if we have the `id` of a town and want to display all the names that belongs to it? An *Embperl*-page that does this job might look something like this:

<body> [- use DBIx::Recordset ; $fdat{'!DataSource'} = 'dbi:mysql:test' ; $fdat{'!Table'} = 'town' ;
`*set` **= DBIx::Recordset ->** `Search(\%fdat)` **; -]**

```
  town: [+ $set{town} +]<br>
  <table>
    <tr>
      <th>name</th><th>firstname</th>
    </tr>
    <tr>
      [- $rec = $set{-name}[$row] -]
      <td>[+ $rec -> {name} +]</td><td>[+ $rec -> {firstname} +]</td>
    </tr>
  </table>
</body>
```

A request to that page might look like this:

`http://www.domain.com/path/to/example3.htm?id=5`

In this example, we specify the name of the table directly inside the page, so it can't be overwritten from outside. The call to `Search` returns the town for the given query parameters. In our example, it will select the town with the `id` 5. The command `[+ $set{town} +]` shows the value of the field town in the current record. After the call to `Search`, this is the first selected record. Next, we need to display all the names. This is very easy using the special field `-name`. `-name` contains a sub-object for the table `name`. The query parameters for this sub-object are set by *DBIx::Recordset* in such a way that it contains all names which meet the link-condition. We just wrap it in a table and we are already done.

[HOME] [CONTENT] [PREV (Basic Example)] [NEXT (Modify the Database)]

Modify the Database

[HOME] [CONTENT] [PREV (Multiple tables)]

Up to this point, we have only discussed the retrieval and display of data. But of course it's also possible to modify data. The simplest way to do this is to assign new values to the result of a `Search` call. For example, you may write `$set{town} = 'Frankfurt'` to change the name of the town. *DBIx::Recordset* **converts this into a vaild SQL-Update-command.**

While this is very useful in normal Perl scripts, you probably won't use it very often in a cgi script. The methods `Insert`/`Update`/`Delete` **will probably prove more useful. Just like** `Search`**, these directly accept query parameters posted to the page. The method** `Execute` **combines all four of these together, making it possible to control the type of action via the CGI-parameters.**

Here is an example:

<html> <head> <title>Database Access with HTML::Embperl and DBIx::Recordset</title> </head> <body>

```
  [-
  ### Database-parameter ###
  use DBIx::Recordset ;
  $fdat{'!DataSource'} = 'dbi:mysql:test' ;
  $fdat{'!Table'} ||= 'town' ;
  $fdat{'!PrimKey'} = 'id' ;
  $fdat{'$max'}     = 10 ;
  ### Execute action according to the query parameters ###
  *set = DBIx::Recordset -> Execute (\%fdat) ;
  -]
  [$if $DBI::errstr $]
        <h1>Database Error [+ $DBI::errstr +]</h1>
  [$else$]
    [-$names = $set -> AllNames ; -]
    [$if $set[0] && $set -> MoreRecords $]
      [### We found more then one record ###]
      [### -> display as a table         ###]
      <table>
        <tr>  [### Display header ###]
          <th>[+ ucfirst ($names -> [$col]) +]</th>
        </tr>
        <tr>  [### Display record -> Table will be expanded by Embperl ###]
          [- $rec = $set[$row] -]
          <td>
            [- $name = $names -> [$col] -]
            [$if $name eq $fdat{'!PrimKey'} $]
              [### Generate HTML link to edit this record ###]
              <a href="example4.htm?!Table=[+ $fdat{'!Table'} +]&[+ $fdat{'!PrimKey'}
+]=[+ $rec ->{$fdat{'!PrimKey'}} +]">[+ $rec -> {$name} +]</a>
            [$elsif $set -> Link4Field($name) $]
              [### Link to other table -> generate HTML link ###]
              [- $link = $set -> Link($set -> Link4Field($name)) -]
              <a href="example4.htm?!Table=[+ $link -> {'!Table'} +]&[+ $link ->
{'!LinkedField'} +]=[+ $rec -> {$link -> {'!MainField'}} +]">[+$rec -> {$name}+]</a>
            [$else$]
              [### Display contents of field ###]
              [+ $rec -> {$names->[$col]} +]
            [$endif$]
            </td>
        </tr>
```

```
      </table>
      [+ $set -> PrevNextForm ('\<\<Prev', 'Next\>\>', \%fdat) ; +]
      <hr>
      <a href="example4.htm?!Table=[+ $fdat{'!Table'} +]&%3dempty=1">Search</a>
record in table '[+ $fdat{'!Table'} +]'
    [$else$]
      [### We found no/one record(s) ###]
      [### -> Display form          ###]
      <form>
        <table>
        <tr>
          [- $name = $names -> [$row] -]
          <td> [### Display fieldname ###]
            [+ ucfirst ($name) +]
          </td>
          <td> [### Display content of field ###]
            <input type=text name="[+ $name +]" value="[+ $set{$name} +]">
            [$if $set -> Link4Field($name) $]
              [### Link to other table -> generate HTML link ###]
              [- $link = $set -> Link($set -> Link4Field($name)) -]
              <a href="example4.htm?!Table=[+ $link -> {'!Table'} +]&[+ $link ->
{'!LinkedField'} +]=[+ $set{$link -> {'!MainField'}} +]">Show record from table '[+
$link -> {'!Table'} +]'</a>
            [$endif$]
          </td>
        </tr>
        </table>
        [### Buttons for the different actions, the "name" attribute determinates
###]
        [###  which action should be taken
###]
        <input type=submit name="=search" value="Search">
        <input type=submit name="=empty"  value="New">
        <input type=submit name="=insert" value="Add">
        <input type=submit name="=update" value="Update">
        <input type=submit name="=delete" value="Delete">
        <input type=hidden name="!Table"  value="[+ $fdat{'!Table'} +]">
      </form>
    [$endif$]
  [$endif$]
</body>
</html>
```

When you first request this page, it will show the contents of the preset table. Alternatively, you can supply a tablename with the parameter `!Table`**. The link, which is shown at the bottom of the page, leads you to an imput form. There, you can fill in one or more fields and press the Search button. This invokes the page itself and** `Execute` **will be instructed by the parameter** `=search` **(Name of the button ``Search'') to retrieve all records which match the entered values.**

If the query finds more then one record, a table with all records found will be shown. If there are more records than specified by the parameter `$max`**, only** `$max` **records are displayed. If this is the case, the** `PrevNextForm` **method adds a ``Previous'' and a ``Next'' button to the page, allowing you to browse through the whole table. In the example above, we assume that every table has a primary key, which is passed to** *DBIx::Recordset* **by the line** `$fdat{'!PrimKey'} = 'id' ;`**. The column which contains this primary key will be displayed as an HTML link containing the parameters to execute a search for just this record. As you can see in example4.htm, this can be used to display a form which includes some of the data from the found record (see below). Columns which are links to other tables will also be shown with an HTML-link. A click on that link will open the linked table or record.**

If the search only selects one record, the same form is shown, but with the data from the record filled in. Now it's possible to change the content. The changes are written to the database when you press the button Update

(parameter `=update`). A new, empty form could be shown with the button New (parameter `=empty`) and if you have written data into this empty form, you can add it as a new record with the Add button (parameter `=insert`). Last but not least, there is a Delete button (parameter `=delete`). In all of these cases, the content of the form is sent to the page itself, and the `Execute` method at the start of the page executes the desired action.

More comments can be found inside the source ([# #] blocks).

[HOME] [CONTENT] [PREV (Multiple tables)]

Operating-Modes

[HOME] [CONTENT] [PREV (Documentation - Content)] [NEXT (Runtime configuration)]

Embperl can operate in one of four modes:

Offline

Converts an HTML file with embedded Perl statements into a standard HTML file.

embpexec.pl [-o outputfile] [-l logfile] [-d debugflags] htmlfile [query_string]

embpexec.bat [-o outputfile] [-l logfile] [-d debugflags] htmlfile [query_string]

Use embpexec.pl on Unix systems and embpexec.bat on Win32 systems.

htmlfile

> **The full pathname of the HTML file which should be processed by Embperl.**

query_string

> **Optional. Has the same meaning as the environment variable QUERY_STRING when invoked as a CGI script. That is, QUERY_STRING contains everything following the first ``?'' in a URL. <query_string> should be URL-encoded. The default is no query string.**

-o outputfile

> **Optional. Gives the filename to which the output is written. The default is stdout.**

-l logfile

> **Optional. Gives the filename of the logfile. The default is /tmp/embperl.log.**

-d debugflags

> **Optional. Specifies the level of debugging (what is written to the log file). The default is nothing. See EMBPERL_DEBUG for exact values.**

As a CGI script

Instead of a file being sent directly by the web server, the document is processed by the CGI script and the result is sent to the client.

embpcgi.pl

embpcgi.bat

Use embpcgi.pl on Unix systems and embpcgi.bat on Win32 systems.

If `embpcgi.pl/embpcgi.bat` **is invoked without any parameters and the environment variable PATH_TRANSLATED is set, it runs itself as a CGI script. This means that form data is taken either from the environment variable QUERY_STRING or from stdin, depending on whether or not CONTENT_LENGTH is set. (This will be set by the web server depending on whether the request method is GET or POST). Input is taken from the file pointed to by PATH_TRANSLATED and the output is send to stdout. The logfile is generated at its default location, which is configurable via the environment variable EMBPERL_LOG.**

To use this mode you must copy embpcgi.pl to your cgi-bin directory. You can invoke it with the URL http://www.domain.xyz/cgi-bin/embpcgi.pl/url/of/your/document.

The /url/of/your/document will be passed to Embperl by the web server. Normal processing (aliasing, etc.) takes place before the URI makes it to PATH_TRANSLATED.

If you are running the Apache httpd, you can also define embpcgi.pl as a handler for a specific file extention or directory.

Example of Apache `srm.conf`**:**

```
<Directory /path/to/your/html/docs>
Action text/html /cgi-bin/embperl/embpcgi.pl
</Directory>
```

NOTE 1: Out of security reasons, embpexec.pl must not be used anymore as CGI script!

NOTE 2: CGI Scripts are not so secure. You should consider using *EMBPERL_ALLOW* **to restrict the access to the rights documents.**

From mod_perl (Apache httpd)

This works like the CGI-Script, but with the advantage that the script is compiled only once at server startup, where other one-time actions (such as opening files and databases) can take place. This will drastically reduce response times for the request. To use this you must compile `Apache httpd` **with** `mod_perl` **and add** `HTML::Embperl` **as the** `PerlHandler`**.**

Example of Apache `srm.conf`**:**

```
SetEnv EMBPERL_DEBUG 2285
Alias /embperl /path/to/embperl/eg
<Location /embperl/x>
SetHandler  perl-script
PerlHandler HTML::Embperl
Options     ExecCGI
</Location>
```

Another possible setup (for Apache 1.3bX see below) is

```
SetEnv EMBPERL_DEBUG 2285
<Files *.epl>
SetHandler  perl-script
PerlHandler HTML::Embperl
Options     ExecCGI
</files>
AddType text/html .epl
```

Don't forget the AddType. In this setup, all files ending with .epl are processed by Embperl.

NOTE: **Since <Files> does not work the same in Apache 1.3bX as it does in Apache 1.2.x, you need to use <FilesMatch> instead.**

```
<FilesMatch ".*\.epl$">
SetHandler  perl-script
PerlHandler HTML::Embperl
Options     ExecCGI
</FilesMatch>
```

See the section EMBPERL_DEBUG (dbgLogLink and EMBPERL_VIRTLOG) to find out how you can configure Embperl so you can view the log file with your browser!

NOTE: When mod_perl is compiled as loadable module (i.e. with USE_DSO) you must not load Embperl at server startup time!

By calling HTML::Embperl::Execute (\%param)

Execute can be used to call Embperl from your own modules/scripts (for example from a Apache::Registry or CGI script) or from within another Embperl page (only 1.2b1 or higher) to nest multiple Embperl pages (for example to store a common header or footer in an different file).

There are two forms you can use for calling Execute. A short form which only takes an filename and optional additional parameters or a long form which takes a hash reference as its argument. This gives it the chance to vary the parameters according to the job that should be done.

(See eg/x/Excute.pl for more detailed examples)

```
Execute($filename, $p1, $p2, $pn) ;
```

This will cause Embperl to interpret the file with the name `$filename` **and, if specified, pass any additional parameters in the array @param (just like** `@_` **in a perl subroutine). The above example could also be written in the long form:**

```
Execute ({inputfile => $filename,
          param     => [$p1, $p2, $pn]}) ;
```

The possible items for hash of the long form are:

inputfile

> **The file which should be used as source. If input is also specified, this parameter should be given a unique name to identify the source. Every time Embperl sees the same text in inputfile, it assumes that you compiled the same source - that means that Embperl uses the same package name as in your last call, and only recompiles the code if mtime has changed or is undefined.**

sub

> **Call the Embperl subroutine named by this** `parameter(see` **also "[$ sub $]"). Currently the subroutine must be in the same file or if it's in another file, the other file has to be imported first.**

input

> **Reference to a string which contains the source. inputfile must also be specified to give a name for the source. The name can be any text.**

mtime

> **Last modification time of member input. If undef the code passed by input is always recompiled, else the code is only recompiled if mtime changes.**

outputfile

> **File to which the output should be written. If neither outputfile nor output is specified, ouput is written to stdout.**

output

Reference to a scalar where the output should be written to.

import

A value of zero tell's Embperl not to execute the page, but define all subrountines found inside. This is neccessary before calling them with Execute by the sub parameter or for an later import.

A value of one tell's Embperl to define to subrountines inside the file (if not already done) and to import them as perl subroutines into the current namespace.

See [$ sub $] metacommand and section about subroutines for more infos.

req_rec

NOTE: The req_rec parameter isn't necessary anymore in versions >= 1.2b2

If used under mod_perl, you should set the req_rec parameter to the Apache request record object provided by mod_perl.

cleanup

This value specifies if and when the cleanup of the package should be executed. (See Variable scope and cleanup below for more information on cleanup)

cleanup = -1

Never cleanup the variables

cleanup = 0 or not specified

If running under mod_perl, cleanup is delayed until the connection to the client is closed, so it does not lengthen the response time to the client. If the Execute function is called more the once before the end of the request, all cleanups take place after the end of the request and not between calls to Execute.

If running as a CGI or offline, cleanup takes place immediately.

cleanup = 1

Immediate cleanup

param

Can be used to pass parameters to the Embperl document and back. Must contain a reference to an array.

```
Example:
    HTML::Embperl::Execute(..., param => [1, 2, 3]) ;
    HTML::Embperl::Execute(..., param => \@parameters) ;
```

The array @param in the Embperl document is setup as an alias to the array. See eg/x/Excute.pl for a more detailed example.

ffld and fdat

Could be used to setup the two Embperl predefined variables.

firstline

Specifies the first linenumber of the sourcefile (Default: 1)

options

Same as EMBPERL_OPTIONS (see below), except for cleanup.

NOTE: You should set the optDisableFormData if you have already read the form data from stdin while in a POST request. Otherwise Execute will hang and try to read the data a second time.

debug

Same as EMBPERL_DEBUG (see below).

escmode

Same as EMBPERL_ESCMODE (see below).

package

Same as EMBPERL_PACKAGE (see below).

virtlog

Same as EMBPERL_VIRTLOG (see below). If virtlog is equal to uri the logfile is sent.

uri

The URI of the request. Only needed for the virtlog feature.

compartment

Same as EMBPERL_COMPARTMENT (see below).

input_func

Same as EMBPERL_INPUT_FUNC (see below). Additionaly you can specify an code reference to an perl function, which is used as input function or an array reference, where the first element contains the code reference and further elements contains additional arguments passed to the function.

output_func

Same as EMBPERL_OUTPUT_FUNC (see below). Additionaly you can specify an code reference to an perl function, which is used as output function or an array reference, where the first element contains the code reference and further elements contains additional arguments passed to the function.

cookie_name

Same as EMBPERL_COOKIE_NAME (see below).

cookie_path

Same as EMBPERL_COOKIE_PATH (see below).

cookie_domain

Same as EMBPERL_COOKIE_DOMAIN (see below).

cookie_expires

Same as EMBPERL_COOKIE_EXPIRES (see below).

errors

Takes a reference to an array. Upon return the array will contain a copy of all errormessages, as long as there are any.

Helper functions for Execute

HTML::Embperl::Init ($Logfile, $DebugDefault)

This function can be used to setup the logfile path and (optional) a default value for the debugflags, which will be used in further calls to Execute. There will always be only one logfile, but you can use Init to change it at any time.

NOTE: You do not need to call Init in version >= 0.27. The initialization of Embperl takes place

automatically when it is loaded.

HTML::Embperl::ScanEnvironement (\%params)

Scans the %ENV and setups %params for use by Execute. All Embperl runtime configuration options are recognized, except EMBPERL_LOG.

EXAMPLES for Execute:

```
# Get source from /path/to/your.html and
# write output to /path/to/output'
HTML::Embperl::Execute ({ inputfile  => '/path/to/your.html',
                          outputfile => '/path/to/output'}) ;
# Get source from scalar and write output to stdout
# Don't forget to modify mtime if $src changes
$src = '<html><head><title>Page [+ $no +]</title></head>' ;
HTML::Embperl::Execute ({ inputfile  => 'some name',
                          input      => \$src,
                          mtime      => 1 }) ;
# Get source from scalar and write output to another scalar
my $src = '<html><head><title>Page [+ $no +]</title></head>' ;
my $out ;
HTML::Embperl::Execute ({ inputfile  => 'another name',
                          input      => \$src,
                          mtime      => 1,
                          output     => \$out }) ;
print $out ;
# Include a common header in an Embperl page,
# which is stored in /path/to/head.html

[- Execute ('/path/to/head.html') -]
```

[HOME] [CONTENT] [PREV (Documentation - Content)] [NEXT (Runtime configuration)]

Variable scope and cleanup

[HOME] [CONTENT] [PREV (SYNTAX)] [NEXT (Predefined variables)]

The scope of a variable declared with **my or local ends at the end of the enclosing [+/- ... -/+] block; the [+/- ... -/+] blocks act much like Perl's { ... } in that regard.**

Global variables (everything not declared with my or local) will be undef'ed at the end of each request, so you don't need to worry about any old variables laying around and causing suspicious results. This is only done for variables in the package the code is eval'ed in -- every variable that does not have an explicit package name. All variables with an explicit package name (i.e., in modules you use) will stay valid until the httpd child process dies. Embperl will change the current package to a unique name for every document, so the influence between different documents is kept to a minimum. You can set the name of the package with EMBPERL_PACKAGE. (See also (Safe-)Namespaces and opcode restrictions.)

Since a CGI script is always a process of its own, you don't need to worry about that when you use Embperl as a CGI script.

If you need to declare variables which need to live longer than just one HTTP request (for example, a database handle), you must either put it's name in the hash %CLEANUP or declare them in another package (i.e., $Persistent::handle instead of $handle).

If you want to use the strict pragma, you can do this by using the var metacommand to declare your variables.

NOTE: **Bacause Apache::DBI has its own namespace, this module will work together with Embperl to maintain your persistent database connection.**

You can disable the automatic cleanup of global variables with EMBPERL_OPTIONS or the cleanup parameter of the Execute function.

You can define expections to the cleanup rule with the hash %CLEANUP.

If you like to do your own cleanup you can define a subroutine CLEANUP in your document. This will be called right before the variables are cleaned up, but after the connection to the client is closed.

EXAMPLE:

```
[! sub CLEANUP { close FH ; } !]
```

[HOME] [CONTENT] [PREV (SYNTAX)] [NEXT (Predefined variables)]

Predefined variables

[HOME] [CONTENT] [PREV (Variable scope and cleanup)] [NEXT (Session handling)]

Embperl has some special variables which have a predefined meaning.

%ENV

Contains the environment as seen from a CGI script.

%fdat

Contains all the form data sent to the script by the calling form. The NAME attribute builds the key and the VALUE attribute is used as the hash value. Embperl doesn't care if it is called with the GET or POST method, but there may be restrictions on the length of parameters using GET -- not from Embperl, but perhaps from the web server, especially if you're using Embperl's CGI mode -- it is safer to use POST.

Embperl also supports ENCTYPE multipart/form-data, which is used for file uploads. The entry in `%fdat` corresponding to the file field will be a filehandle, as with CGI.pm. (Embperl uses CGI.pm internally to process forms encoded with multipart/form-data.)

File upload example:

```
HTML page:
```

```
  <FORM METHOD="POST" ENCTYPE="multipart/form-data">
    <INPUT TYPE="FILE" NAME="ImageName">
  </FORM>
Embperl ACTION:
  [- if (defined $fdat{ImageName}) {
       open FILE, "> /tmp/file.$$";
       print FILE $buffer
         while read($fdat{ImageName}, $buffer, 32768);
       close FILE;
     }
  -]
```

When you have installed CGI.pm 2.46 or above you may also retrieve the filename (local filename, as it was on the browser side) and the informations provied by the CGI.pm uploadInfo function. To get the filename just print out the value of the correspondig `%fdat` entry, instead of using it as a filehandle. To get the **uploadInfo use the fieldname with a dash in front of it:**

```
Example:
# ImageName is the NAME of the field, you must replace it with whatever
# name is given in your HTML code
Filename:      [+ $fdat{ImageName} +] <br>
Content-Type:  [+ $fdat{-ImageName} -> {'Content-Type'} +] <br>
```

NOTE: The way uploadInfos are accessed before 1.2b11 are not supported anymore.

@ffld

Contains all the field names in the order in which they were sent by the browser. This is normally -- but not necessarily -- the order in which they appear in your form.

%idat

Contains all the values from all input tags processed so far.

%udat (only 1.2b1 or higher)

You can use %udat to store per user data. As long as you don't use %udat, nothing happens, but as soon as you write anything to %udat, Embperl creates a session id and sends it via a cookie to the browser. The data you have written to `%udat` is stored by Apache::Session. The next time the same user request an Embperl page, the browser sends the cookie with the session id back and Embperl fill the `%udat` hash from Apache::Session with just the same values as you have stored for that user. (See also Session handling)

%mdat (only 1.2b2 or higher)

You can use %mdat to store per module/page data. As long as you don't use %mdat, nothing happens, but as soon as you write anything to %mdat, Embperl creates a session id and stores the data via Apache::Session. The next time any user hits the same Embperl page, Embperl fill the `%mdat` hash from

Apache::Session with just the same values as you have stored within the last request to that page. (See also Session handling)

$row, $col

Row and column counts for use in dynamic tables. (See "HTML tag table".)

$maxrow, $maxcol

Maxium number of rows or columns to display in a table. To prevent endless loops, `$maxrow` defaults to 100 and `$maxcol` to 10. (See "HTML tag table".)

$cnt

Contains the number of table cells displayed so far. (See "HTML tag table".)

$tabmode

Determines how the end of a dynamic table is detected:

end of table

$tabmode = 1

> **End when one of the expressions with `$row` becomes undefined. The row containing the undefined expression is not displayed. Only those expression are observed which contains an access to the varibale $row.**

$tabmode = 2

> **End when an expression with `$row` becomes undefined. The row containing the undefined expression is displayed.**

$tabmode = 4

> **End when `$maxrow` rows have been displayed.**

end of row

$tabmode = 16

> **End when one of the expression with `$col` becomes undefined. The column containing the undefined expression is not displayed. Only those expression are observed which contains an access to the varibale $col.**

$tabmode = 32

> **End when an expression with `$col` becomes undefined. The column containing the undefined expression is displayed.**

$tabmode = 64

> **End when `$maxcol` columns have been displayed.**

The default is 17, which is correct for all sort of arrays. You should rarely need to change it. The two values can be added together.

$escmode

Turn HTML and URL escaping on and off. The default is on ($escmode = 3).

$escmode = 3

> **The result of a Perl expression is HTML-escaped (e.g., `>' becomes `>') in normal text and URL-escaped (e.g., `&' becomes `%26') within of A, EMBED, IMG, IFRAME, FRAME and LAYER tags.**

$escmode = 2

> **The result of a Perl expression is always URL-escaped (e.g., `&' becomes `%26').**

$escmode = 1

> **The result of a Perl expression is always HTML-escaped (e.g., `>' becomes `>').**

$escmode = 0

> **No escaping takes place.**

$req_rec

This variable is only available when running under control of mod_perl. It contains the request record needed to access the Apache server API. See perldoc Apache for more information.

LOG

This is the filehandle of the Embperl logfile. By writing `print LOG ``something''' you can add lines to the logfile. NOTE: The logfile line should always start with the pid of the current process and continue with a four-character signature delimited by a ':', which specifies the log reason.

Example: print LOG ``[$$]ABCD: your text\n'' ;

If you are writing a module for use under Embperl you can say

```
tie *LOG, 'HTML::Embperl::Log';
```

to get a handle by which you can write to the Embperl logfile.

OUT

This filehandle is tied to Embperl's output stream. Printing to it has the same effect as using the [+ ... +] block. (See also optRedirectStdout)

@param

Will be setup by the 'param' parameter of the Execute function. Could be used to pass parameters to an Embperl document and back. (see Execute for further docs)

%http_headers_out (only 1.2b10 and above)

You can put any http headers you want to send into this hash. If you set an location header Embperl will automaticly set the status to 301 (Redirect). Example:

```
[- $http_headers_out{'Location'} = "http://www.ecos.de/embperl/"; -]
```

see also META HTTP-EQUIV=

$optXXX $dbgXXX

All options (see EMBPERL_OPTIONS) and all debugging flags (see EMBPERL_DEBUG) can be read and set by the corresponding variables.

```
Example:
  [- $optRawInput = 1 -] # Turn the RawInput option on

  Now write something here
  [- $optRawInput = 0 -] # Turn the RawInput option off again
  [+ $dbgCmd +] # Output the state of the dbgCmd flag
```

There are a few exceptions, where the variables can only be read. Setting of such options must be done via the config-files. Read-only variables are:

$optDisableVarCleanup

$optSafeNamespace

$optOpcodeMask

$optDisableChdir

$optEarlyHttpHeader

$optDisableFormData

$optAllFormData

$optRedirectStdout

$optAllowZeroFilesize

$optKeepSrcInMemory

%CLEANUP

Embperl cleanups up only variables with are defined within the Embperl page. If you want Embperl to cleanup addtional variables you can add them to the hash %CLEANUP, with the key set to the variable name and the value set to one. The other way round you could prevent Embperl from cleaning up some variables, by adding them to this hash, with a values of zero.

[HOME] [CONTENT] [PREV (Variable scope and cleanup)] [NEXT (Session handling)]

Session handling

[HOME] [CONTENT] [PREV (Predefined variables)] [NEXT ((Safe-)Namespaces and opcode restrictions)]

From 1.2b1 and higher Embperl is able to handle per user sessions for you. You can store any data in the *%udat* **hash and if the same user request again an Embperl document, you will see the same values in that hash again.**

From 1.2b2 and higher Embperl is able to handle per module/page persitent data for you. You can store any data in the *%mdat* **hash and if any request comes to the same Embperl document, you will see the same values in that hash again.**

To configure *Embperl* **to do session management for you, you must have installed** *Apache::Session* **(1.00 or higher) and tell Embperl which storage and locker classes you would like to use for** *Apache::Session***. This is done by setting the environement variable** `EMBPERL_SESSION_CLASSES`**. You may have a startup.pl for your httpd which looks like this:**

```
BEGIN
    {
    $ENV{EMBPERL_SESSION_CLASSES} = "DBIStore SysVSemaphoreLocker" ;
    $ENV{EMBPERL_SESSION_ARGS}    = "DataSource=dbi:mysql:session UserName=test" ;
    } ;
use HTML::Embperl ;
```

For Solaris it's neccessary to set the `nsems` **Argument if you use SysVSemaphoreLocker**

```
$Apache::Session::SysVSemaphoreLocker::nsems = 16;
```

You may also put this in the httpd/srm.conf:

```
PerlSetEnv EMBPERL_SESSION_CLASSES "DBIStore SysVSemaphoreLocker"
PerlSetEnv EMBPERL_SESSION_ARGS "DataSource=dbi:mysql:session UserName=test"
PerlModule HTML::Embperl ;
```

`EMBPERL_SESSION_ARGS` **is a space separated list of name/value pairs, which gives additional arguments for Apache::Session classes.**

NOTE: The above configuration works only with *Embperl* **1.2b11 or above. The way** *Apache::Session* **was used in earlier versions still works, but I have removed the documentation to avoid confusion. Changes are, that you don't need to load** *Apache::Session* **anymore on your own and that** *Apache::Session 1.00* **takes totaly different arguments then** *Apache::Session 0.17***.**

Now you are able to use the `%udat` **and** `%mdat` **hashs for your user/module sessions. As long as you don't touch** `%udat` **or** `%mdat` **Embperl will not create any session, also Apache::Session is loaded. As soon as you store any value to %udat, Embperl will create a new session and send a cookie to the browser to maintain it's id, while the data is stored by Apache::Session. (Further version may also be able to use URL rewriting for storing the id). When you store data to** `%mdat` **Embperl will store the data via Apache::Session and retrieves it when the next request comes to the same page.**

[HOME] [CONTENT] [PREV (Predefined variables)] [NEXT ((Safe-)Namespaces and opcode restrictions)]

Utility Functions

[HOME] [CONTENT] [PREV ((Safe-)Namespaces and opcode restrictions)] [NEXT (Input/Output Functions)]

AddCompartment($Name)

Adds a compartment for use with Embperl. Embperl only uses the opcode mask from it, not the package name. AddCompartment returns the newly- created compartment so you can allow or deny certain opcodes. See the Safe.pm documentation for details of setting up a compartment. See the chapter about (Safe-)Namespaces and opcode restrictions for details on how Embperl uses compartments.

Example:

```
$cp = HTML::Embperl::AddCompartment('TEST');
$cp->deny(':base_loop');
```

MailFormTo($MailTo, $Subject, $ReturnField)

Sends the content of the hash `%fdat` in the order specified by @ffld to the given **$MailTo addressee, with a subject of $Subject. If you specify** `$ReturnField` **the value of that formfield will be used as Return-Path. Usually, this will be the field where the user enters his e-mail address in the form.**

If you specifiy the following example code as the action in your form

```
<FORM ACTION="x/feedback.htm" METHOD="POST"
      ENCTYPE="application/x-www-form-urlencoded">
```

The content of the form will be mailed to the given e-mail address.

MailFormTo uses EMBPERL_MAILHOST as SMTP server or localhost if non given.

Example:

```
<HTML>
<HEAD>
<TITLE>Feedback</TITLE>
</HEAD>
<BODY>
      [- MailFormTo('webmaster@domain.xy',
                    'Mail from WWW Form', 'email') -]
      Your data has been sccesfully sent!
</BODY>
</HTML>
```

This will send a mail with all fields of the form to webmaster@domain.xy, with the Subject 'Mail form WWW Form' and will set the Return-Path of the mail to the address which was entered in the field with the name 'email'.

NOTE: You must have Net::SMTP (from the libnet package) installed to use this function.

---

exit

exit will override the normal Perl exit in every Embperl document. Calling exit will immediately stop any further processing of that file and send the already-done work to the output/browser.

NOTE 1: If you are inside of an Execute, Embperl will only exit this Execute, but the file which called the file containing the exit with Execute will continue. NOTE 2: If you write a module which should work with Embperl under mod_perl, you must use Apache::exit instead of the normal Perl exit (just like always when running under mod_perl).

[HOME] [CONTENT] [PREV ((Safe-)Namespaces and opcode restrictions)] [NEXT (Input/Output Functions)]

Input/Output Functions

[HOME] [CONTENT] [PREV (Utility Functions)] [NEXT (Inside Embperl - How the embedded Perl code is actually processed)]

ProxyInput ($r, $in, $mtime, $src, $dest)

```
 B<USAGE in srm.conf:>
 <Location /embperl/ifunc>
 SetHandler perl-script
 PerlHandler HTML::Embperl
 Options ExecCGI
 PerlSetEnv EMBPERL_INPUT_FUNC "ProxyInput, /embperl/ifunc,
http://otherhost/otherpath";
 </Location>
```

This input function will request the source from another URL instead of reading it from the disk. In the above USAGE Example, a request to /embperl/ifunc/foo.html, will first fetch the URL http://otherhost/otherpath/foo.html, and then it will process this document by Embperl and then it will send it to the browser.

This could be used to process documents by mod_include **and Embperl, so in one document there can be both Server-Side Includes and Embperl Commands.**

```
 Example B<srm.conf> for B<SSI> and B<Embperl>:
 <Location /embperl>
 SetHandler perl-script
 PerlHandler HTML::Embperl
 Options ExecCGI
 PerlSetEnv EMBPERL_INPUT_FUNC "ProxyInput, /embperl, http://localhost/src";
 </Location>
 <Location /src>
 SetHandler server-parsed
 Options +Includes
 </Location>
```

The source files must be in the location /src, but they will be requested via the URI /embperl. Every request to /embperl/foo.html will do a proxy-request to /src/foo.html. The file /src/foo.html will be processed by mod_include and then sent to Embperl, where it can be processed by Embperl before being sent to the browser. It would be also possible to use two httpd's on different ports. In this configuration, the source and the URI location could be the same.

LogOutput ($r, $out, $basepath)

```
 B<USAGE in srm.conf:>
 <Location /embperl/ofunc>
 SetHandler perl-script
```

```
PerlHandler HTML::Embperl
Options ExecCGI
PerlSetEnv EMBPERL_OUTPUT_FUNC "LogOutput, /usr/msrc/embperl/test/tmp/log.out"
</Location>
```

LogOutput is a custom output function. It sends the output to the browser and writes the output to a unique file. The filename has the the form ``$basepath.$$.$LogOutputFileno''.

[HOME] [CONTENT] [PREV (Utility Functions)] [NEXT (Inside Embperl - How the embedded Perl code is actually processed)]

Performance

[HOME] [CONTENT] [PREV (Inside Embperl - How the embedded Perl code is actually processed)] [NEXT (Bugs)]

To get the best performace from Embperl, it is necessary to restrict logging to a minimum. You can drastically slow down Embperl if you enable all logging options. (This is why `make test' takes a while to run.) You should **never enable dbgFlushOutput, dbgFlushLog or dbgCacheDisable in a production environment. More debugging options are useful for development where it doesn't matter if the request takes a little bit longer, but on a heavily-loaded server they should be disabled. Addtionaly the options** *optDisableChdir***,** *optDisableHtmlScan***,** *optDisableCleanup* **have consequences for the performance.**

Also take a look at mod_perl_tuning.pod for general ideas about performance.

[HOME] [CONTENT] [PREV (Inside Embperl - How the embedded Perl code is actually processed)] [NEXT (Bugs)]
