cpl-c Module

Bogdan-Andrei Iancu
FhG FOKUS

Edited by
Bogdan-Andrei Iancu

cpl-c Module

Edited by and Bogdan-Andrei Iancuand Bogdan-Andrei Iancu

Revision History

Revision $Revision: 1.2 $ $Date: 2003/09/15 15:16:59 $

Table of Contents

1. User's Guide **1**
- 1.1. Overview 1
- 1.2. Dependencies 1
  - 1.2.1. SER Modules 1
  - 1.2.2. External Libraries or Applications 1
- 1.3. Exported Parameters 1
  - 1.3.1. `cpl_db` (string) 1
  - 1.3.2. `cpl_table` (string) 2
  - 1.3.3. `cpl_dtd_file` (string) 2
  - 1.3.4. `log_dir` (string) 2
  - 1.3.5. `proxy_recurse` (int) 3
  - 1.3.6. `proxy_route` (int) 3
  - 1.3.7. `nat_flag` (int) 3
  - 1.3.8. `lookup_domain` (int) 3
- 1.4. Exported Functions 4
  - 1.4.1. `cpl_run_script(type,mode)` 4
  - 1.4.2. `cpl_process_register()` 5
- 1.5. Installation & Running 5

2. Developer's Guide **6**

3. Frequently Asked Questions **7**

List of Examples

1-1. Set `cpl_db` parameter 2
1-2. Set `cpl_table` parameter..... 2
1-3. Set `cpl_dtd_file` parameter 2
1-4. Set `log_dir` parameter..... 2
1-5. Set `proxy_recurse` parameter..... 3
1-6. Set `proxy_route` parameter 3
1-7. Set `nat_flag` parameter..... 3
1-8. Set `lookup_domain` parameter..... 4
1-9. `cpl_run_script` usage..... 5

Chapter 1. User's Guide

1.1. Overview

cpl-c modules implements a CPL (Call Processing Language) interpreter. Support for uploading/downloading/removing scripts via SIP REGISTER method is present.

IMPORTANT Since CPL-C is not fully releasted in this SER version, before start compiling, please apply to TM module the patch from modules/cpl-c/tm.patch. From sip_router directory do "patch -p0<modules/cpl-c/tm.patch".

1.2. Dependencies

1.2.1. SER Modules

The following modules must be loaded before this module:

- *TM (Transaction) module- used for proxying/forking requests*
- *SL (StateLess) module - used for sending stateless reply when responding to REGISTER request or for sending back error responses*
- *USRLOC (User Location) module - used for implementing lookup("registation") tag (adding into location set of the users's contact)*

1.2.2. External Libraries or Applications

The following libraries or applications must be installed before running SER with this module loaded:

- *libxml2 and libxml2-devel - on some SO, these to packages are merged into libxml2. This library contains an engine for XML parsing, DTD validation and DOM manipulation.*

1.3. Exported Parameters

1.3.1. `cpl_db` (string)

A SQL URL have to be given to the module for knowing where the database containing the table with CPL scripts is locates. If required a user name and password can be specified for allowing the module to connect to the database server.

This parameter is MANDATORY!

Example 1-1. Set `cpl_db` parameter

```
...
modparam("cpl_c","cpl_db","mysql://user:passwd@host/database")
...
```

1.3.2. `cpl_table` (string)

Indicates the name of the table that store the CPL scripts. This table must be locate into the database specified by "cpl_db" parameter. For more about the format of the CPL table please see modules/cpl-c/init.mysql.

This parameter is MANDATORY!

Example 1-2. Set `cpl_table` parameter

```
...
modparam("cpl_c","cpl_table","cpltable")
...
```

1.3.3. `cpl_dtd_file` (string)

Points to the DTD file describing the CPL grammar. The file name may include also the path to the file. This path can be absolute or relative (be careful the the path will be relative to the starting directory of SER).

This parameter is MANDATORY!

Example 1-3. Set `cpl_dtd_file` parameter

```
...
modparam("cpl_c","cpl_dtd_file","/etc/ser/cpl-06.dtd")
...
```

1.3.4. `log_dir` (string)

Points to a directory where should be created all the log file generated by the LOG CPL node. A log file per user will be created (on demand) having the name username.log.

If this parameter is absent, the logging will be disabled without generating error on execution.

Example 1-4. Set `log_dir` parameter

```
...
modparam("cpl_c","log_dir","/var/log/ser/cpl")
...
```

1.3.5. `proxy_recurse` (int)

Tells for how many time is allow to have recurse for PROXY CPL node If it has value 2, when doing proxy, only twice the proxy action will be re-triggered by a redirect response; the third time, the proxy execution will end by going on REDIRECTION branch. The recurse feature can be disable by setting this parameter to 0

Default value of this parameter is 0.

Example 1-5. Set `proxy_recurse` parameter

```
...
modparam("cpl_c","proxy_recurse",2)
...
```

1.3.6. `proxy_route` (int)

Before doing proxy (forward), a script route can be executed. All modifications made by that route will be reflected only for the current branch.

Default value of this parameter is 0 (none).

Example 1-6. Set `proxy_route` parameter

```
...
modparam("cpl_c","proxy_route",1)
...
```

1.3.7. `nat_flag` (int)

Sets the flag used for marking calls via NAT. Used by lookup tag when retriving a contact behind a NAT (this flag will be set).

Default value of this parameter is 6.

Example 1-7. Set `nat_flag` parameter

```
...
modparam("cpl_c","nat_flag",4)
...
```

1.3.8. `lookup_domain` (int)

Tells if the lookup tag should use or not the domain part when doing user_location search. Set it to a non zero value to force also domain matching.

Default value of this parameter is 0.

Example 1-8. Set `lookup_domain` parameter

```
...
modparam("cpl_c","lookup_domain",1)
...
```

1.4. Exported Functions

1.4.1. `cpl_run_script(type,mode)`

Starts the execution of the CPL script. The user name is fetched from new_uri or requested uri or from To header -in this order- (for incoming execution) or from FROM header (for outgoing execution). Regarding the stateful/stateless message processing, the function is very flexibile, being able to run in different modes (see below the"mode" parameter). Normally this function will end script execution. There is no guaranty that the CPL script interpretation ended when ser script ended also (for the same INVITE ;-)) - this can happen when the CPL script does a PROXY and the script interpretation pause after proxying and it will be resume when some reply is received (this can happen in a different process of SER). If the function returns to script, the SIP server should continue with the normal behavior as if no script existed. When some error is returned, the function itself haven't sent any SIP error reply (this can be done from script).

Meaning of the parameters is as follows:

- *type* - which part of the script should be run; set it to "incoming" for having the incoming part of script executed (when an INVITE is received) or to "outgoing" for running the outgoing part of script (when a user is generating an INVITE - call).
- *mode* - sets the interpreter mode as stateless/stateful behaviour. The follwoing modes are accepted:
  - *IS_STATELESS - the current INVITE has no transaction created yet. All replys (redirection or deny) will be done is a stateless way. The execution will switch to statefull only whena proxy is done. So, if the function returns, will be in stateless mode.*
  - *IS_STATEFUL - the current INVITE has already a transaction associated. All signalling operations (replys or proxy) will be done in stateful way.So, if the function returns, will be in stateful mode.*
  - *FORCE_STATEFUL - the current INVITE has no transaction created yet. All signalling operations will be done is a stateful way (on signalling, the transaction will be created from within the interpreter). So, if the function returns, will be in stateless mode.*

  HINT: is_statefull is vary difficult to manage from the routing script (script processing can continue in statefull mode); is_stateless is the fastes and less resources consumer (transaction is created only if proxying is done), but there is minimal protection against retransmisions (since replies are send stateless); force_statefull is a good compromise - all signalling is done statefull (retransmission protection) and in the same time, if returning to script, it will be in stateless mode (easy to continue the routing script execution)

Example 1-9. `cpl_run_script` usage

```
...
cpl_run_script("incoming","force_statefull");
...
```

1.4.2. `cpl_process_register()`

This function MUST be called only for REGISTER requests. It checks if the current REGISTER request is related or not with CPL script upload/download/ remove. If it is, all the needed operation will be done. For checking if the REGISTER is CPL related, the function looks fist to "Content-Type" header. If it exists and has a the mime type set to "application/cpl+xml" means this is a CPL script upload/remove operation. The distinction between to case is made by looking at "Content-Disposition" header; id its value is "script;action=store", means it's an upload; if it's "script;action=remove", means it's a remove operation; other values are considered to be errors. If no "Content-Type" header is present, the function looks to "Accept" header and if it contains the "*" or "application/cpl-xml" the request it will be consider one for downloading CPL scripts. The functions returns to script only if the the REGISTER is not related to CPL. In other case, the function will send by itself the necessary replies (stateless - using sl), including for errors.

1.5. Installation & Running

Notes about installation and running.

Chapter 2. Developer’s Guide

The module does not provide any sort of API to use in other SER modules.

Chapter 3. Frequently Asked Questions

1. Where can I find more about SER?

Take a look at http://iptel.org/ser.

2. Where can I post a question about this module?

First at all check if your question was already answered on one of our mailing lists:

- http://mail.iptel.org/mailman/listinfo/serusers
- http://mail.iptel.org/mailman/listinfo/serdev

E-mails regarding any stable version should be sent to `<serusers@iptel.org>` and e-mail regarding development versions or CVS snapshots should be send to `<serdev@iptel.org>`.

If you want to keep the mail private, send it to `<serhelp@iptel.org>`.

3. How can I report a bug?

Please follow the guidelines provided at: http://iptel.org/ser/bugs