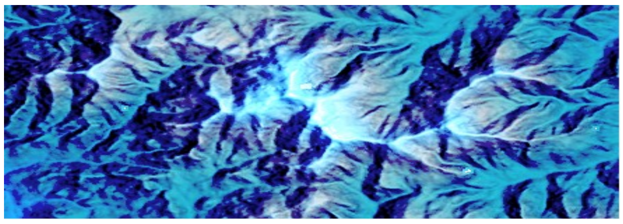


SPECTO®

Internet/intranet application monitoring

User's manual
& tutorial

Release 1.98

MATHESIS

SPECTO

collection of manuals
(***usage*** and ***tutorial***)

for product release 1.98

Product information

`www.nls.de#gotospecto`
`SPECTO@NLS.DE`
+49 6321 968540

Ordering information

product key: spman198
downloading: `www.nls.de/downloads/SpectoManual.pdf`

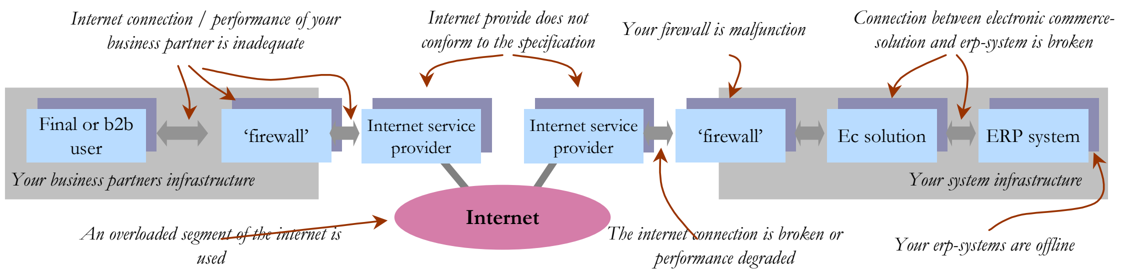
CONTENT

Overview

Problem description

Companies are beginning to use the internet for their business. Common areas like advertising, product information, sales and service are enhanced with internet solutions. In some areas they even begin to replace the traditional concepts.
At the same time, companies more and more rely on services delivered via the internet or intranet for their own internal operation.

Above services are based on relative new and in some cases complex technologies. In those interconnected environments even malfunctions in minor components can decrease the availability or quality of the service.


Internet connection / performance of your business partner is inadequate
Internet provide does not conform to the specification
Your firewall is malfunction
Connection between electronic commerce-solution and erp-system is broken
Final or b2b user
'firewall'
Internet service provider
Internet service provider
'firewall'
Ec solution
ERP system
Your business partners infrastructure
Your system infrastructure
Internet
An overloaded segment of the internet is used
The internet connection is broken or performance degraded
Your erp-systems are offline

Though it is possible, and already done in many cases, to monitor all components used in an e-commerce solution, this is not sufficient to guarantee the functionality from an user's point of view.

Idea

Only a continuous end-to-end monitoring, which accesses the whole implementation from the user to the last involved system can allows fast detection of problems and provide hints to emerging failures.

Implementation

SPECTO behaves like a real user would. It constantly processes predefined transactions on the web applications to monitor and, based on the evaluated results, takes actions like informing support organizations and signaling system management applications.

Also SPECTO's summarizing reports allow retrospection of the monitored applications behavior to detect upcoming problems.

Architecture

SPECTO engine

Is a highly focused monitor for web based transactions using html (user based web) or XML (b2b based web) transactions.
SPECTO can be used in-house and over the internet; for details about SPECTO scenarios see 'www.nls.de#gotospecto'.

This manual describes the usage and installation of SPECTO; it also includes a tutorial to ease becoming familiar with SPECTO configuration and operation.

The SPECTO system

A SPECTO system consists of :

- An active SPECTO instance
- One or multiple backup SPECTO instances
- The SPECTO web server
- The SPECTO (relational) database

From the operating system the following services are required (SPECTO base system) :

- TCP/IP protocol
- SMTP protocol and SMTP Server name

SPECTO Net

Multiple SPECTO instances can be tied together forming a SPECTO net. All SPECTO instance of a SPECTO net share the same data (configuration and results).

Multiple SPECTO nets can coexist without interference.

All communication between the SPECTO instances of a SPECTO net is based on the SMTP (email) protocol and is therefore guaranteed to work across firewalls.

All data communication is formatted in XML.

SPECTO web recorder

The web recorder is a client software for windows PCs to record user sessions with the Microsoft Internet Explorer, prepare the recorded information and sent it to the SPECTO engine for advanced editing and playback.

The SPECTO data model

SPECTO ‘clients’

A SPECTO instance provides an unlimited number of independent ‘clients’. For any client exactly one account (userid / password) is maintained and checked against during login; the authorization is restricted to the assigned client; it is not possible to change or view information of other clients in the SPECTO instance.
Aside this client-tied accounts there is one instance-wide (super-user) account which may switch between the different clients without login in again. Also, this super-user has access to the system-wide configuration parameters and commands.

SPECTO ‘chains’

‘URL’ or ‘web page’ are standard internet acronyms, SPECTO additionally uses the ‘chain’ keyword. A chain describes a ‘session’ which consists of a set of URLs which will be performed in a given sequence and share a common environment (like a user-login).
The ‘session’ concept is not part of the www specification. Web application designer invented this to allow the implementation of applications like those in a classical edp environment.

Start of a chain

From a user’s view a chain would always begin with an informational page (because the URL is called without parameters) which will show the initial page; and be followed with a series of transactional pages. SPECTO can ignore the initial informational page (because the URL of the first transactional page and all its parameters are known).

Next page of a chain

SPECTO has two ways to select the next URL of a chain :
- ***‘hard wired’*** : The URL is specified in the URL definition. However, through the content evaluation, which of the defined URLs of a chain will be used as next URL may be variable. See chapter ‘FollowUp computation’ for details.
- ***‘dynamic’*** : If in the URL definition no URL string is specified, SPECTO accepts the ‘current’ URL (which is the URL sent by the web browser in response to the previous transactional URL).

SPECTO supports all known technologies to implement sessions (see ‘session management’ for details).

SPECTO ‘URLs’

The web provides ‘informational’ and ‘transactional’ pages. ‘Informational’ pages don’t provide any user interaction, they just display information and, usually, allow the user, via links to switch to another page. ‘Transactional pages’ provide for some kind of user

interaction (usually via input fields and buttons); 'transactional pages' require server-side processing and are the basis of web-based ('e-commerce') applications.
In SPECTO, any URL definition with parameters is considered an 'transactional' page and any URL without defined parameters is considered an 'informational' page.

A SPECTO URL specifies a web page and consists of :

- an address ('uniform resource locator', URL); e.g. 'www.nls.de'
- a set of input parameters. They define the input a user would normally enter into the page.
- a set of content tags which are checked against the output of the web page. The evaluation of the different tags can be combined logically to compute the page 'result' and the 'follow up page'.
- a set of technical attributes used by SPECTO to process the page. E.g. the 'timeout' parameter which may override the chains 'timeout' setting.

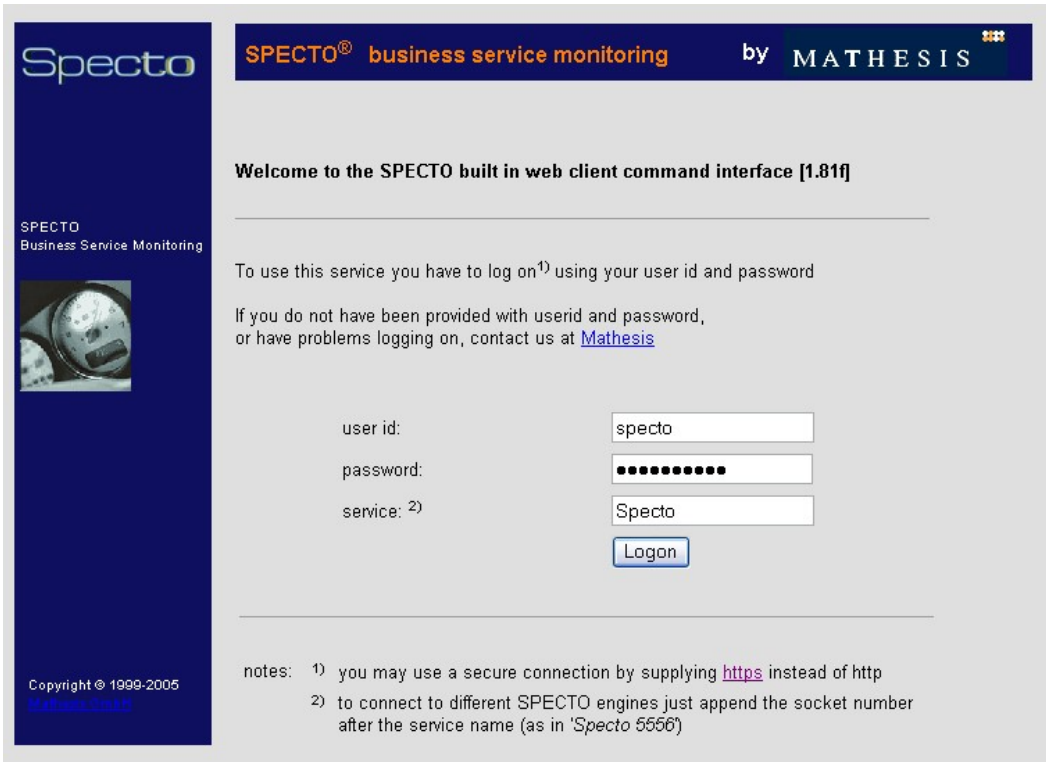
To ease setting up a URL configuration SPECTO provides a mechanism (see 'page analysis' for details) to analyze web pages in order to find out about parameters and configurations used in the web pages construction.

Using SPECTO

Logging in

Start page of SPECTO's web server is the login page. This page may be accessed using ‚http.//xxx.xxx.xxx.xxx' URL in the browser with 'xxx.xxx.xxx.xxx' being the IP address which has been assigned to the SPECTO instance. (For demonstration use 'www.nls.de', then 'customer area' / 'Kundenbereich').

In the login page, the company's name is used as user id which is identical to the initial password. The service name is 'Specto'.


Specto
SPECTO® business service monitoring by MATHESIS
SPECTO
Business Service Monitoring
Copyright © 1999-2005

Welcome to the SPECTO built in web client command interface [1.81f]

To use this service you have to log on[1] using your user id and password

If you do not have been provided with userid and password,
or have problems logging on, contact us at Mathesis

user id: specto
password: ●●●●●●●●●●●
service: [2] Specto
Logon

notes: [1] you may use a secure connection by supplying https instead of http
[2] to connect to different SPECTO engines just append the socket number after the service name (as in '*Specto 5556*')

Select the 'Logon' button to log on. Once logged on, the password may be changed using the *'pw <newpassword> <newpassword>'* command. Note that the password is stored unencrypted and so is visible for the SPECTO administrator.

Notes:

1. The same panel is used to access the NLS online support database; just replace the service name 'Specto' with 'support'. Your userid/password is also valid for the support application.
2. In custom installations the 'Service:' field is not available; the service will default to 'SPECTO'.
3. The URL '`http://`*`machine-name`*`/servlet/SpectoHome`' may be used to log on to SPECTO in a separate browser (without having the left hand side NLS/SPECTO' frame).
4. Multiple instances are accessed by adding the port number to the service name (see 'Execution and Termination', 'Advanced topics').

Initial screen

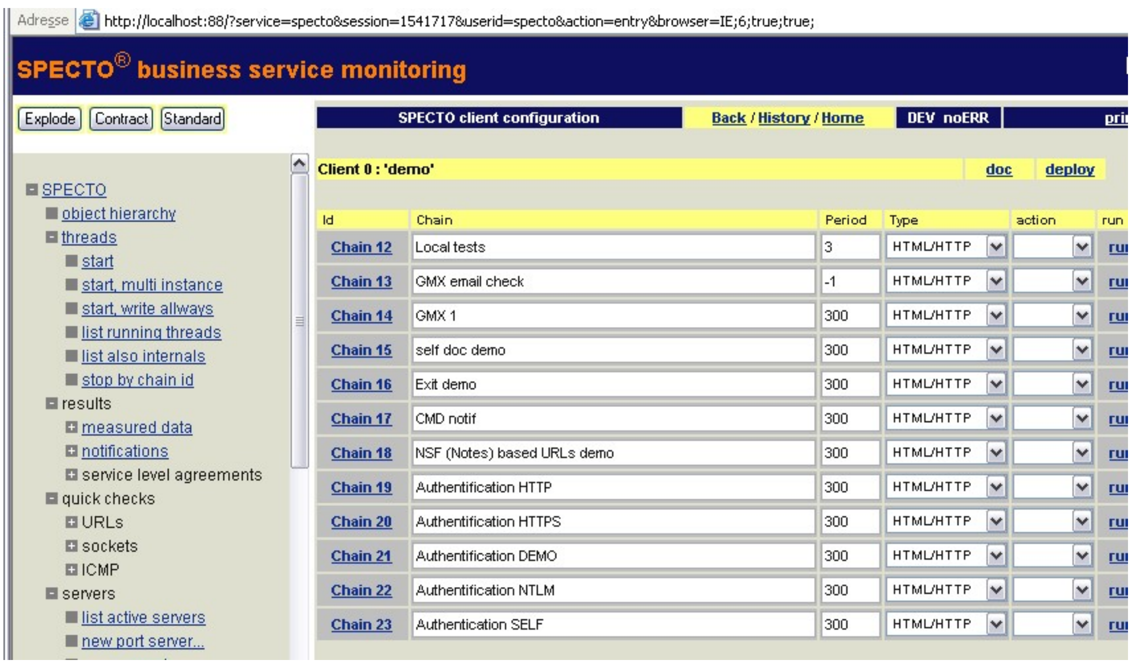
In most environments (recent browser, scripting enabled) usage of the (default) advanced layout as described here is recommended. Other, less demanding layouts may be also selected (see next chapter).

Advanced layout

This layout consists of the following sub screens :

- Title area
- Navigation menu (on the left side, with command buttons on top)
- Main screen
- Command area (on the bottom)

Each of the sub screens can be scrolled, sized and printed individually, and are refreshed independent from each other by the SPECTO engine. Pressing the 'refr.' (refresh) button in the command area forces a repaint of all sub screens.


Adresse http://localhost:88/?service=specto&session=1541717&userid=specto&action=entry&browser=IE;6;true;true;

SPECTO® business service monitoring

Explode Contract Standard

- SPECTO
  - object hierarchy
  - threads
    - start
    - start, multi instance
    - start, write allways
    - list running threads
    - list also internals
    - stop by chain id
  - results
    - measured data
    - notifications
    - service level agreements
  - quick checks
    - URLs
    - sockets
    - ICMP
  - servers
    - list active servers
    - new port server...

SPECTO client configuration | Back / History / Home | DEV noERR | pri

Client 0 : 'demo' doc deploy

Id	Chain	Period	Type	action	run
Chain 12	Local tests	3	HTML/HTTP		ru
Chain 13	GMX email check	-1	HTML/HTTP		ru
Chain 14	GMX 1	300	HTML/HTTP		ru
Chain 15	self doc demo	300	HTML/HTTP		ru
Chain 16	Exit demo	300	HTML/HTTP		ru
Chain 17	CMD notif	300	HTML/HTTP		ru
Chain 18	NSF (Notes) based URLs demo	300	HTML/HTTP		ru
Chain 19	Authentification HTTP	300	HTML/HTTP		ru
Chain 20	Authentification HTTPS	300	HTML/HTTP		ru
Chain 21	Authentification DEMO	300	HTML/HTTP		ru
Chain 22	Authentification NTLM	300	HTML/HTTP		ru
Chain 23	Authentication SELF	300	HTML/HTTP		ru

Navigation menu

In the navigation menu individual branches can be open or closed; the open/close status is preserved over logoff times (stored in a cookie). The buttons above the navigation area allow the minimizing and maximizing of the navigation tree or to apply the standard configuration.

All navigation menu entries are resolved to commands of the SPECTO engine (and therefore appear in the log and the history list). For commands with parameters a selection screen for the parameters will be displayed before the command is executed (this is indicated by three dots (…) after the menu entry). The set of commands displayed in the navigation menu is dependent of the rights (see chapter 'user management) of the actual logged on user.

Main area

The content of the main screen at logon time (initial executed command) can be changed by setting the 'FirstCommand' or 'First_<user>' attribute. System default is the 'logo' command, but 'el', the root of the object hierarchy (as shown above), or 'lc' for the client selection screen, are recommendable alternatives.

The main screen is titled by a header line displaying the current screen title, a 'back' link for re-executing the last command, a 'history' link for a list of the most latest executed commands, and the 'home' link to return to the first page. Also available are links to the net/node/client hierarchy.
Below the navigation screen, the footer line displays the user command links, the command line field, and links to the SPECTO home directory on the web and the NLS support application.

Command area

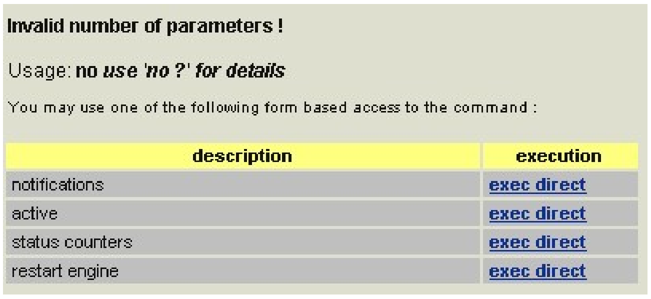
The user commands are user definable links to SPECTO commands; they can be maintained using the '`uc`' user commands console, or directly via the '`ua`', '`ud`' and '`ul`' commands.

The 'Command' field is used to directly enter SPECTO commands (see chapter 'commands'. If, directly after the logon, the 'Execute' button is clicked, or the 'enter' key is pressed, SPECTO will display a summary of the most important commands. At other times, executing with an empty action field will repeat the last command issued.

A list of the most recent executed commands is available using the 'history' link in the command line or the 'hi' command. The commands displayed there can be re-executed, copied into the command line (for editing before re-execution) or saved as a user defined command.

Normal operation of SPECTO will seldom require the usage of the command line; relevant activities may be selected graphically and often used commands can be individually tied to links.

If a command is entered with an incorrect set of parameters a message and a short help is displayed.
Also the menu is searched for matching entries. Such entries are presented in a list for direct execution (see example at the right).



Invalid number of parameters !

Usage: **no *use 'no ?' for details***

You may use one of the following form based access to the command :

description	execution
notifications	exec direct
active	exec direct
status counters	exec direct
restart engine	exec direct

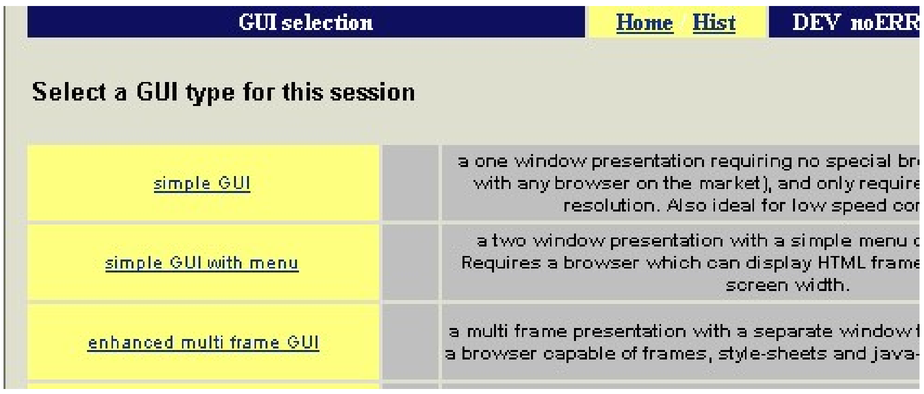
Screen refresh

Usually only those screen areas relevant for a command are redrawn. The '`refr.`' (refresh) button forces a redraw of alls sub screens (frames).

Alternate layouts

For limited environments (physical screen size, browser capabilities, etc.) a set of alternate screen layouts is available. :

Switching to one of the alternate user interface is accomplished by entering the '`gui`' command and then selecting an appropriate layout from the resulting menu (The standard screen, as featured in the chapter before, corresponds to the entry **'graphical multi frame GUI')** :

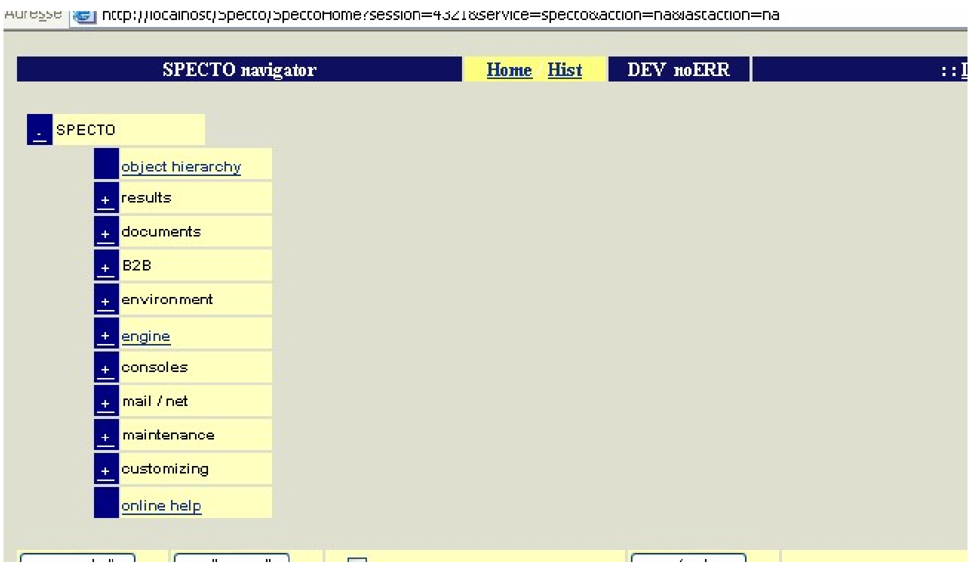


GUI selection | Home Hist | DEV noERR

Select a GUI type for this session

simple GUI	a one window presentation requiring no special br with any browser on the market), and only require resolution. Also ideal for low speed co
simple GUI with menu	a two window presentation with a simple menu Requires a browser which can display HTML frame screen width.
enhanced multi frame GUI	a multi frame presentation with a separate window a browser capable of frames, style-sheets and java-

Basic layout



Adresse http://localhost/specto/spectoHome?session=4321&service=specto&action=na&lastaction=na

SPECTO navigator | Home Hist | DEV noERR | ::

- SPECTO
  - object hierarchy
  - + results
  - + documents
  - + B2B
  - + environment
  - + engine
  - + consoles
  - + mail / net
  - + maintenance
  - + customizing
  - online help

The most basic layout shown above is optimized to work with physical screen resolutions of 800 vs. 600 pixels and browsers without frames, java script and cookies.

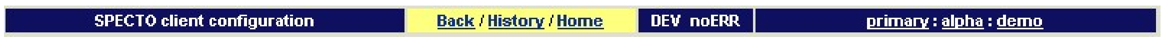
The navigation tree as shown in the picture above and located in the left frame of the 'multiframe GUI', is displayed by command '`na`'.

Note that the basic layout may not be supported in coming releases of the SPECTO engine!

SPECTO user interface

Though the SPECTO engine is command driven it provides a web browser based graphical user interface.

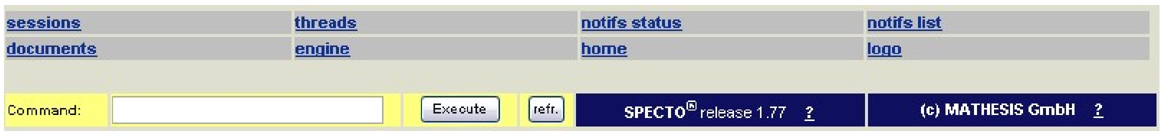
The header block



The header line shows :

- the current screen title. Also outstanding operator messages are displayed in the right corner of this area.
- a 'back' link for re-execution of the last command
- a 'history' link for a list of all recent commands
- the 'home' link, a click here displays the SPECTO start screen, usually the root of the object hierarchy.
- some status information, at the moment if the instance is a development configuration and the error status of the engine
- links to the 'net / node / client' hierarchy.

The footer block



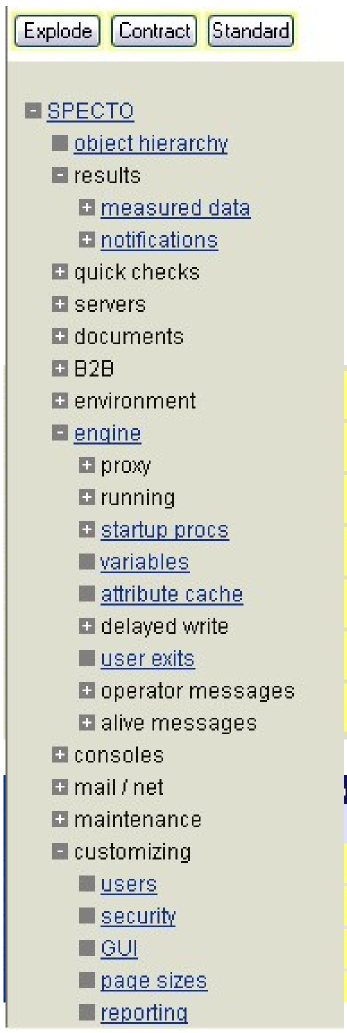
The footer line displays the

- user definable command links (see paragraph 'User defined commands' in section 'Commands in detail'),
- the command line entry field, and
- links to the SPECTO web home and NLS support.

The command line allows the direct entering of commandos to the SPECTO engine (see section 'Commands in detail'). A command is submitted by pressing the 'Execute' button; if the button is pressed with an empty command line, the last command will be executed again.

The navigation screen

Though SPECTO can be used entirely by commands it is advised to use the graphical 'navigation' interface. The navigation' screen can be called from any other screen by issuing the 'na' command; also (if not configured different) it is the screen called when following the 'home' link of the header line.

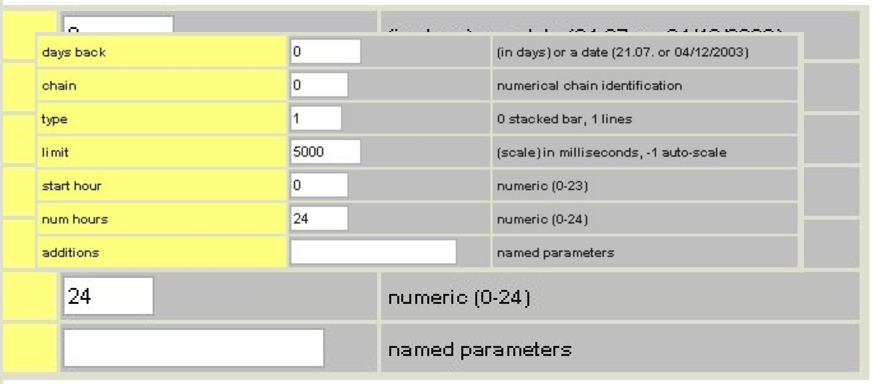


Explode Contract Standard

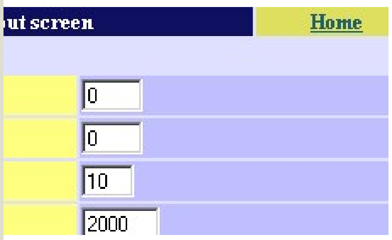
- SPECTO
  - object hierarchy
  - results
    - measured data
    - notifications
  - quick checks
  - servers
  - documents
  - B2B
  - environment
  - engine
    - proxy
    - running
    - startup procs
    - variables
    - attribute cache
    - delayed write
    - user exits
    - operator messages
    - alive messages
  - consoles
  - mail / net
  - maintenance
  - customizing
    - users
    - security
    - GUI
    - page sizes
    - reporting

The navigation screen displays the SPECTO functionally in a tree view which can be expanded and collapsed by clicking the 'arrow' links left of the inner nodes.

The displayed functionality can be started by a click on the function's name. Functions with a highlighted background (as 'table dynamic and 'graphic dynamic' in the example) will display a dynamic parameters screen before the function execution :



days back	0	(in days) or a date (21.07. or 04/12/2003)
chain	0	numerical chain identification
type	1	0 stacked bar, 1 lines
limit	5000	(scale) in milliseconds, -1 auto-scale
start hour	0	numeric (0-23)
num hours	24	numeric (0-24)
additions		named parameters
	24	numeric (0-24)
		named parameters



ut screen Home

0

0

10

2000

.

Basic layout :

Using the basic layout the behavior of the navigation screen can be switched to always displaying only the necessary nodes (preserving vertical screen space) by selecting the 'expand only active branch' option (the refresh key must be used to refresh the display after switching the behavior).

The navigation screen does only display the nodes to which the current logged on user has the right to execute them; therefore it may look different from user to user.

Language support

The SPECTO systems primary user interface language is English. Since release 1.90 a limited support for German and French user interface language has been inserted.

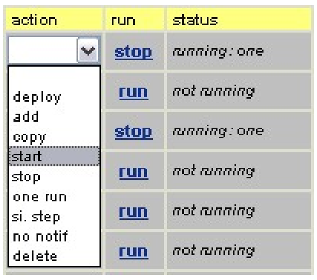
The language can be selected using the ‘lang’ command.

Starting and stopping of monitoring ('chain execution')

SPECTO's basic execution unit is the 'chain' of sequentially executed web or b2b pages. Any chain may be 'not executing', 'executing in one thread' or 'executing in multiple threads'.

Manual start

Chains are usually started (and stopped) using the 'run' / 'stop' links of the chain list ('client configuration screen', reached via the 'el' command or the 'object hierarchy' link in the navigation frame) or the equivalent entry in the drop down box of the 'action' column.
A chain may also be started or stopped using the 'start' button in the chain configuration screen (this screen is reached by selecting a chain in the above client configuration screen.



action	run	status
	stop	running: one
	run	not running
	stop	running: one
	run	not running
	run	not running
	run	not running

A chain may be started in several instances; the stop command stops all instances.
For debugging a chain may be executed for exactly one run (action 'one run').

On command level, any chain may also be started using the '`sc <chain-id>`' command. The chain's processing will be submitted to a separate process with a name of '`thread<nnn>`'. Any process can be stopped using the '`kt <thread-name>`', '`ktf <thread-name>`' ('force'), '`kti <chain-name>`', '`kta true [ <nice> ]`' (all chains, 'nice' is `true` : wait until stop of chain execution) or '`kp <thread-name>`' ('hard core stop') commands. Processes can be stopped and restarted without loss of elder measurement data.

The chain execution period can be dynamic using 'ramp mode', see chapter 'ramp mode' for details.

The '`lt`' ('list threads') command can be used to inspect the running threads.

Notifications resulting from errors in the monitored object can be deleted by using the 'no notif' entry or the '`no d`...' command set. Note that existing notifications are NOT deleted by a chain stop command!

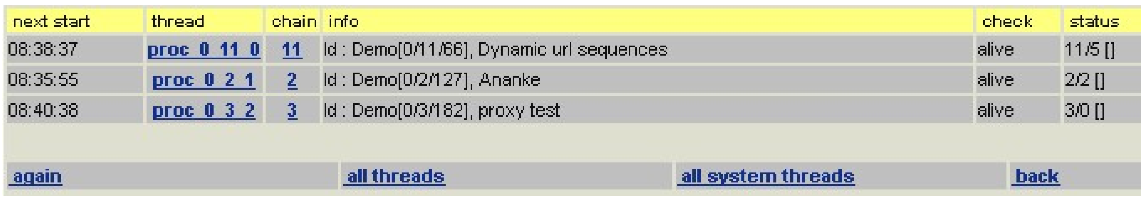
Automatic start

By default no threads will be started automatically at SPECTO launch.

Any configuration of currently running thread may be set as startup configuration using the '`st save`' command. The '`st list`' command displays the list of currently defined start up threads; '`st delall`' deletes all startup entries; and '`st start`' starts all defined threads.

Operations at Runtime

A list of currently running chains ('threads') is available via menu 'threads' – 'list running threads' (command 'lt') :

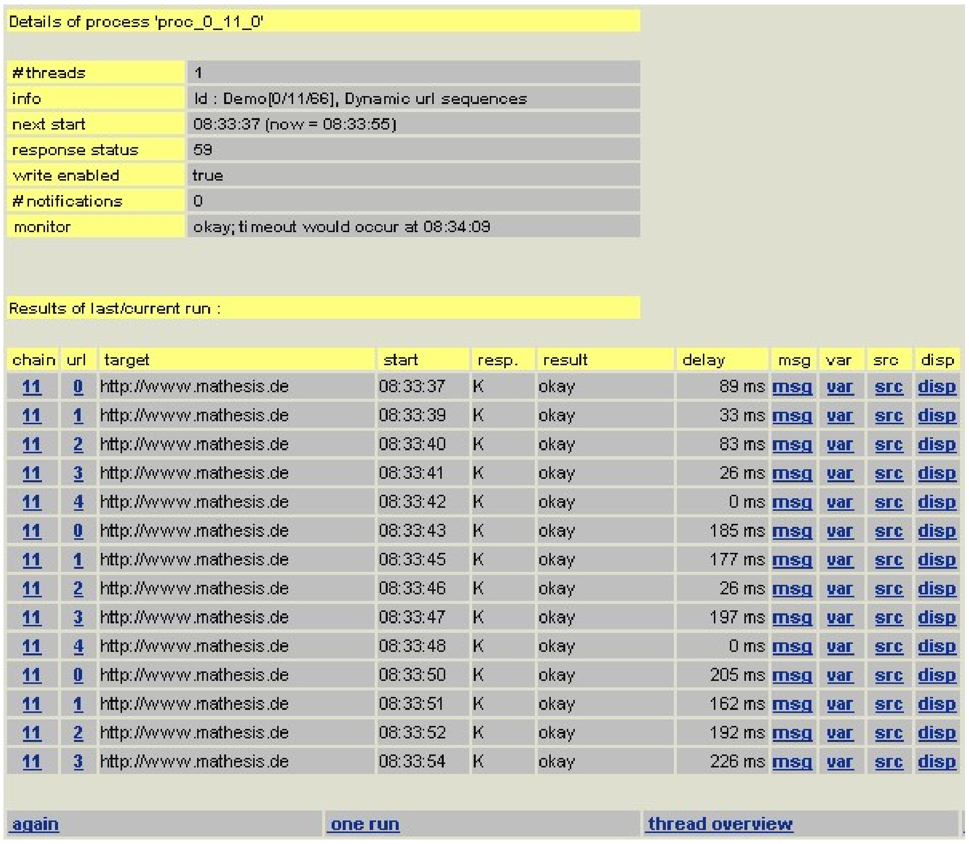


next start	thread	chain	info	check	status
08:38:37	proc_0_11_0	11	Id : Demo[0/11/66], Dynamic url sequences	alive	11/5 []
08:35:55	proc_0_2_1	2	Id : Demo[0/2/127], Ananke	alive	2/2 []
08:40:38	proc_0_3_2	3	Id : Demo[0/3/182], proxy test	alive	3/0 []

again all threads all system threads back

The status column identifies the current step (chain/url) of the running thread.
The 'lt p' variant of the 'lt' command list all currently running processes (for all clients and the SPECTO's engine internal processes).

During a chain is running the current/least status and results can be inspected by following the 'proc…' link in the threads list, or the 'runs:' link in the chain screen:



Details of process 'proc_0_11_0'

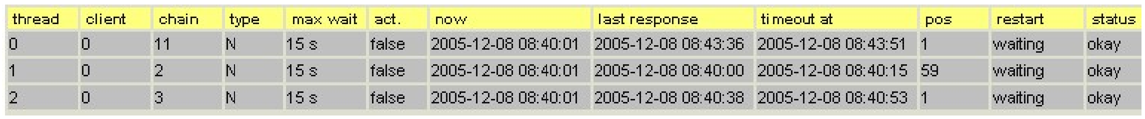
#threads	1
info	Id : Demo[0/11/66], Dynamic url sequences
next start	08:33:37 (now = 08:33:55)
response status	59
write enabled	true
#notifications	0
monitor	okay; timeout would occur at 08:34:09

Results of last/current run :

chain	url	target	start	resp.	result	delay	msg	var	src	disp
11	0	http://www.mathesis.de	08:33:37	K	okay	89 ms	msg	var	src	disp
11	1	http://www.mathesis.de	08:33:39	K	okay	33 ms	msg	var	src	disp
11	2	http://www.mathesis.de	08:33:40	K	okay	83 ms	msg	var	src	disp
11	3	http://www.mathesis.de	08:33:41	K	okay	26 ms	msg	var	src	disp
11	4	http://www.mathesis.de	08:33:42	K	okay	0 ms	msg	var	src	disp
11	0	http://www.mathesis.de	08:33:43	K	okay	185 ms	msg	var	src	disp
11	1	http://www.mathesis.de	08:33:45	K	okay	177 ms	msg	var	src	disp
11	2	http://www.mathesis.de	08:33:46	K	okay	26 ms	msg	var	src	disp
11	3	http://www.mathesis.de	08:33:47	K	okay	197 ms	msg	var	src	disp
11	4	http://www.mathesis.de	08:33:48	K	okay	0 ms	msg	var	src	disp
11	0	http://www.mathesis.de	08:33:50	K	okay	205 ms	msg	var	src	disp
11	1	http://www.mathesis.de	08:33:51	K	okay	162 ms	msg	var	src	disp
11	2	http://www.mathesis.de	08:33:52	K	okay	192 ms	msg	var	src	disp
11	3	http://www.mathesis.de	08:33:54	K	okay	226 ms	msg	var	src	disp

again one run thread overview

The monitor (command 'mo l') may be used to get information about failures of running threads :

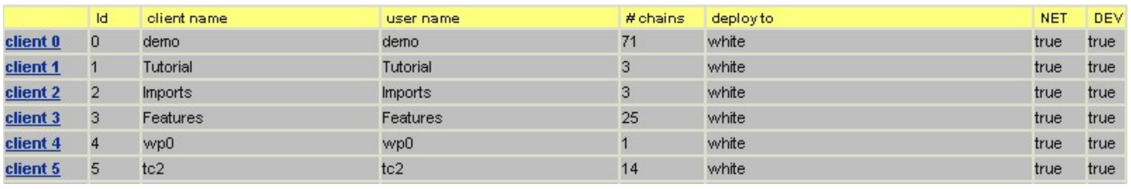


thread	client	chain	type	max wait	act.	now	last response	timeout at	pos	restart	status
0	0	11	N	15 s	false	2005-12-08 08:40:01	2005-12-08 08:43:36	2005-12-08 08:43:51	1	waiting	okay
1	0	2	N	15 s	false	2005-12-08 08:40:01	2005-12-08 08:40:00	2005-12-08 08:40:15	59	waiting	okay
2	0	3	N	15 s	false	2005-12-08 08:40:01	2005-12-08 08:40:38	2005-12-08 08:40:53	1	waiting	okay

Object hierarchy – 'the clients'

Within any SPECTO instance, data is divided into independent 'clients'. Any operation (with exception of some 'cross-client' actions) happens only within the scope of the current client.

The administrative user can view/select clients using the 'select client' entry of the navigation screen or by issuing the '`lc`' command:

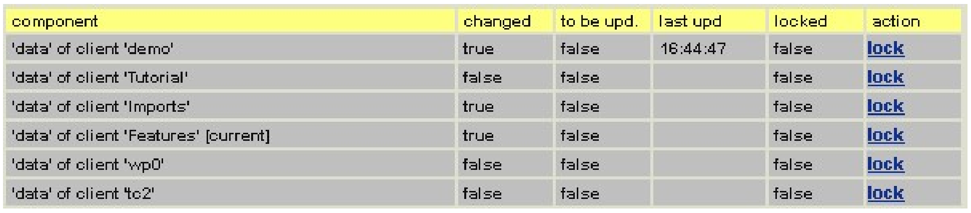


	Id	client name	user name	# chains	deploy to	NET	DEV
client 0	0	demo	demo	71	white	true	true
client 1	1	Tutorial	Tutorial	3	white	true	true
client 2	2	Imports	Imports	3	white	true	true
client 3	3	Features	Features	25	white	true	true
client 4	4	wp0	wp0	1	white	true	true
client 5	5	tc2	tc2	14	white	true	true

The client list shows the define clients name, primary user name, the number of defined chains, the list of deployed to SPECTO instances (if this instance is a deployment in stance), whether this client is part of a SPECTONET configuration and whether the client is enabled for development.

The '`sm <clientid>`' command can be used to switch directly between clients.

More technical information about the clients status can be obtained by the '`lock`' command :



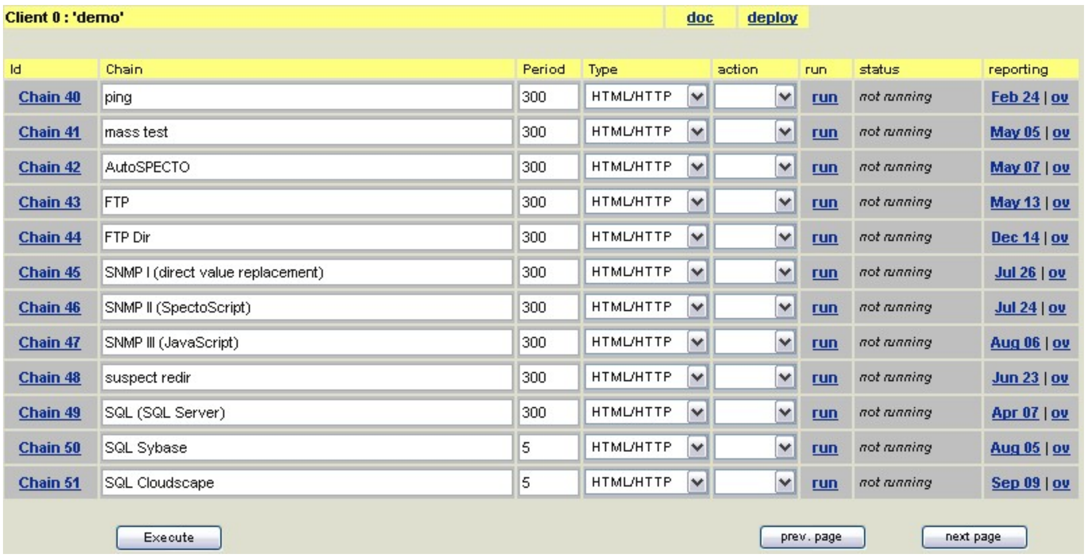
component	changed	to be upd.	last upd	locked	action
'data' of client 'demo'	true	false	16:44:47	false	lock
'data' of client 'Tutorial'	false	false		false	lock
'data' of client 'Imports'	true	false		false	lock
'data' of client 'Features' [current]	true	false		false	lock
'data' of client 'wp0'	false	false		false	lock
'data' of client 'tc2'	false	false		false	lock

Client maintenance is available under menu '`maintenance - clients`'. It is possible to create new clients, rename clients and to delete clients.

SPECTO clients are not the root of the SPECTO object hierarchy. SPECTONet is available to connect multiple SPECTO engines to a network appearing as one large, distributed SPECTO entity. The SPECTO engines within a SPECTONet are called 'nodes' in the object hierarchy.

The object hierarchy – 'the current client'

At logon the actual client for the user is determined according to the user configuration. The 'client definition' screen shown below is the entry point of the object hierarchy (reached by selection 'object hierarchy' from the navigation screen or entering the 'el' command on the command line) and displays all configured chains of the current client.



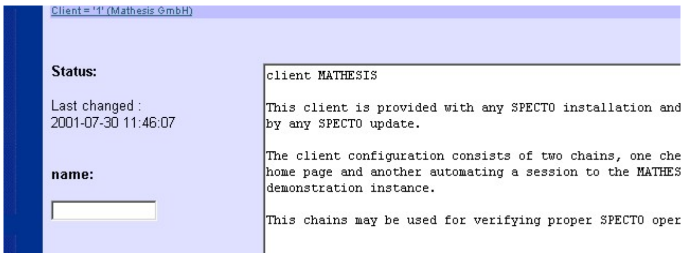
Client 0 : 'demo' doc deploy

Id	Chain	Period	Type	action	run	status	reporting
Chain 40	ping	300	HTML/HTTP		run	not running	Feb 24 | ov
Chain 41	mass test	300	HTML/HTTP		run	not running	May 05 | ov
Chain 42	AutoSPECTO	300	HTML/HTTP		run	not running	May 07 | ov
Chain 43	FTP	300	HTML/HTTP		run	not running	May 13 | ov
Chain 44	FTP Dir	300	HTML/HTTP		run	not running	Dec 14 | ov
Chain 45	SNMP I (direct value replacement)	300	HTML/HTTP		run	not running	Jul 26 | ov
Chain 46	SNMP II (SpectoScript)	300	HTML/HTTP		run	not running	Jul 24 | ov
Chain 47	SNMP III (JavaScript)	300	HTML/HTTP		run	not running	Aug 06 | ov
Chain 48	suspect redir	300	HTML/HTTP		run	not running	Jun 23 | ov
Chain 49	SQL (SQL Server)	300	HTML/HTTP		run	not running	Apr 07 | ov
Chain 50	SQL Sybase	5	HTML/HTTP		run	not running	Aug 05 | ov
Chain 51	SQL Cloudscape	5	HTML/HTTP		run	not running	Sep 09 | ov

The 'client definition' screen consists of the areas :

- Client name, links to its documentation ('doc') and to deploy the client via SPECTONet ('Deploy', only visible if the SPECTO engine has deployment enabled)
- Editable fields of the individual chains :
    - Links to the chain configuration screens
    - Chain name
    - Execution period (in seconds)
    - The 'actions' drop down box
    - Quick start/stop
    - the chain execution status
    - Link to the reporting data for the chain; also showing the last timestamp (or date if no results of today are available) of a measurement. The 'ov' link selects an overview of all measurements of the last fourteen days.
- Execution and scrolling buttons: 'Execute' to execute an action selected in the 'action' column (do not disturb this button with the 'Execute' button of the command area!); and, there appropriate, buttons to scroll on a page-wise to the list of chains.

The 'docu' link switches to the documentation screen. Documentation can be maintained for every level of the SPECTO object hierarchy.
It is also used by PDF based reporting and SLA generation.



Client = '1' (Mathesis GmbH)

Status:

Last changed :
2001-07-30 11:46:07

name:

```
client MATHESIS

This client is provided with any SPECTO installation and
by any SPECTO update.

The client configuration consists of two chains, one che
home page and another automating a session to the MATHES
demonstration instance.

This chains may be used for verifying proper SPECTO oper
```

Editable fields of the chains

The name of the chain, the period of executions (in seconds) and the default type (which may be overwritten in the chain definition screen) can be changed by editing the values in the this text fields.
The 'action' column provides shortcuts to selected commands for the chain.
Changes in these fields only become active when they are committed using the 'Execute' button.
Chain entries may be repositioned on the screen using 'drag & drop' (browser dependent).

Links to the chain configuration screens

The links in the 'Id' column switch to the according chain definition screen.

Chain execution status

This column displays if a chain is running and in how many instances. If notifications are active, their number is displayed as the second number within the parenthesis. Warning or error status of a running chain is represented as yellow or red coloring of the status text.

Link to the reporting data for the chain

Selecting the '`ov`' link will switch to the 'reporting overview' screen of the chain and display the results summary of the last two weeks of operation. The link left to the 'ov' link shows and links to the last date for which reporting data is available.

Scrolling buttons

Larger lists of chains may be separated into pages (see page size command. If this is enabled page scroll buttons will appear below the listed chains. Using the '`el <chainid>`' variant of the '`el`' command, it is possible to navigate to a certain area of the chain list without using the scroll buttons.

Saving data

Any change in a text field must be committed using the 'Execute' button before it will be saved to the database. Failing to do so, e.g. following a link or using the browsers 'back' button will discard all entries made in the screen.

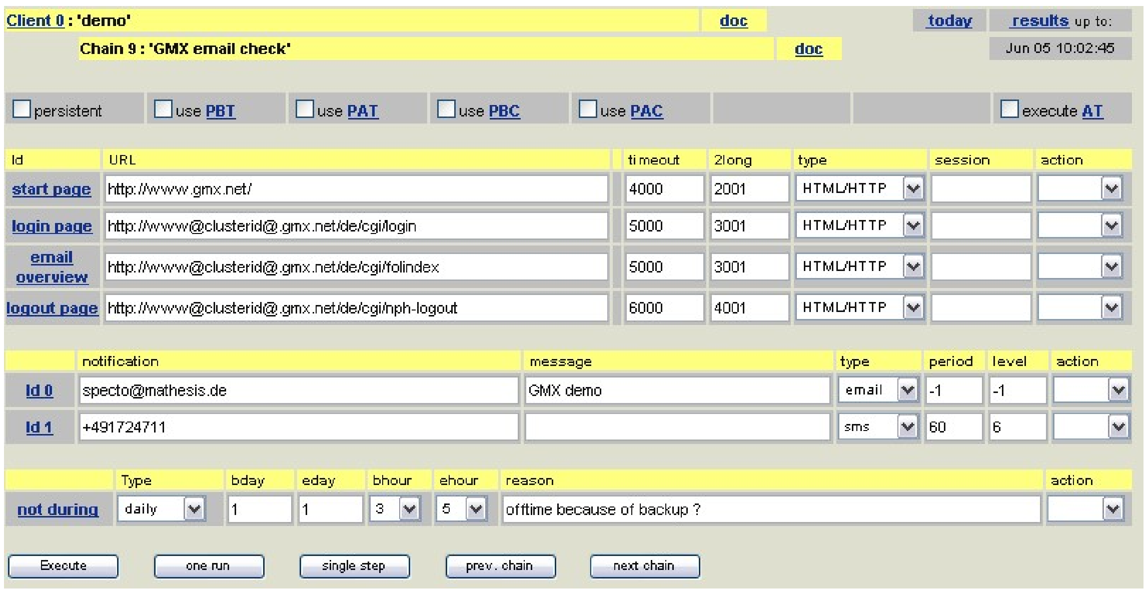
The object hierarchy – 'chain'

Any chain exists of a number of URLs (web processing) or XMLs (b2b processing). For any such entry two time parameters (timeout and 'too long' warning) can be configured. For every chain a number of notifications (see chapter 'notifications') can be configured.

The 'chain configuration' page is used to specify one chain of a client. It consists of the areas :

- Object hierarchy navigation (allows fast access to the higher level client)
- Option flags
- List of URLs / pages
- List of notifications
- List of off-times
- Buttons

Editing is done by changing the values in the fields and pressing the 'Execute' button directly below. Actions are performed by selecting the action in the 'action' field and pressing the same 'Execute' button. Only one action (but multiple changes are allowed for one button execution.


Client 0 : 'demo' doc today results up to:
Chain 9 : 'GMX email check' doc Jun 05 10:02:45

☐ persistent ☐ use PBT ☐ use PAT ☐ use PBC ☐ use PAC ☐ execute AT

Id	URL	timeout	2long	type	session	action
start page	http://www.gmx.net/	4000	2001	HTML/HTTP		
login page	http://www@clusterid@.gmx.net/de/cgi/login	5000	3001	HTML/HTTP		
email overview	http://www@clusterid@.gmx.net/de/cgi/folindex	5000	3001	HTML/HTTP		
logout page	http://www@clusterid@.gmx.net/de/cgi/nph-logout	6000	4001	HTML/HTTP		

	notification	message	type	period	level	action
Id 0	specto@mathesis.de	GMX demo	email	-1	-1	
Id 1	+491724711		sms	60	6	

	Type	bday	eday	bhour	ehour	reason	action
not during	daily	1	1	3	5	offtime because of backup ?	

(This screen was generated using the 'ec 1' command, alternatively by following the 'chain 1' link in the client configuration screen.)

Area 'option flags' :

'Persistent'	Data (e.g. variables and applet instances) of a run of the chain will be preserved to the next run.
'use PBC'	the 'process before chain' script will be executed before every execution of the chain.
'use PAC'	the 'process after chain' script will be executed after every execution of the chain.

Area 'URLs / pages' :

For any URL the URN (the address) and the default timeout and time-warning ('2long') have to be entered. Also the name of a session id parameter identifying the session of the URL sequence can be entered ('cookies' are handled automatically). Those parameters values (multiple may be separated using a ';') will be learned in the first page of the chain which has them specified and will be applied to the consecutive pages.

URLs can be edited in detail (parameters and content) by following the link 'Page_x'; new URLs can be inserted using the 'add' action or deleted using the 'delete' action. Any URL can be analyzed ('analyze' action; same as the 'pa' command; for details refer to the 'commands in detail' section) or executed once ('test' action).

URL entries may be repositioned on the screen using 'drag & drop' (browser dependent).

Commands :

Instead of URLs also commands can be used; in this case the type field has to be set to 'command'. The following commands area available :

Command	description	Example
Gosub <target>	continue at the specified location and prepare to return	gosub Home:0
Goto <target>	continue at the specified location	Goto Home:StartPage
Repeat	repeat the current URL	Repeat
Return	return to the location from where this chain was called	Return
Break	Stop the chain processing of this run	Break
Quit	Stop the chain processing complete.	Quit

See chapter 'Page specification' / 'follow up computation' for a description of the location specification.

Comments :

Any URL entry (even commands) can be appended with a comment; comments are identified by a leading ' //' (note the space).

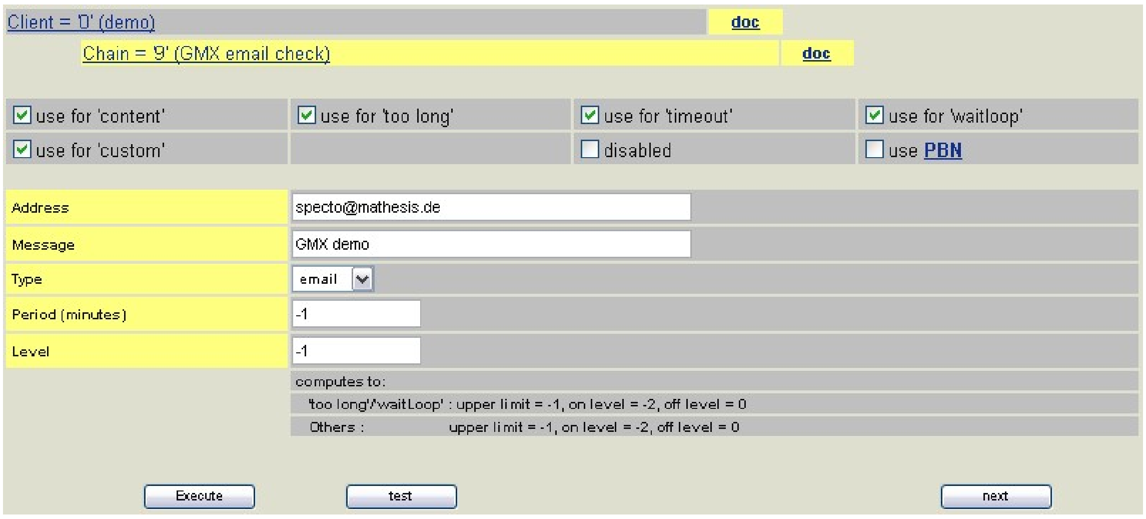
Example : `localhost/index.html // main page`

Area 'notifications' :

In case of problems during the chain execution notifications are generated. Notifications can be edited, added and deleted. Any notification can be supplied with an optional message, this message may contain variables. Variables may also be used to specify the notification address itself.

In order for the different types to work correctly some prerequisites may be required; see chapter ‘notification’.

Notifications may be specified in more detail by following the ‘Id…’ links in the chain configuration screen :


Client = '0' (demo) doc
Chain = '9' (GMX email check) doc

✓ use for 'content'	✓ use for 'too long'	✓ use for 'timeout'	✓ use for 'waitloop'
✓ use for 'custom'		☐ disabled	☐ use PBN

Address	specto@mathesis.de
Message	GMX demo
Type	email
Period (minutes)	-1
Level	-1

```
computes to:
  'too long'/'waitLoop' : upper limit = -1, on level = -2, off level = 0
  Others :                upper limit = -1, on level = -2, off level = 0
```

Here, a notification may be tied to specific error types, be completely disabled (without changing any of the other parameters), and a ‘process before notification’ (‘pbn’) script may be specified.

According to the specified ’Level’ parameter, the maximum level and the ‘on’ and ‘off’ thresholds are displayed.

Using the ‘Test’ button a notification may be submitted immediately without having an appropriate error condition.

Notification entries may be repositioned on the screen using ‘drag & drop’ (browser dependent).

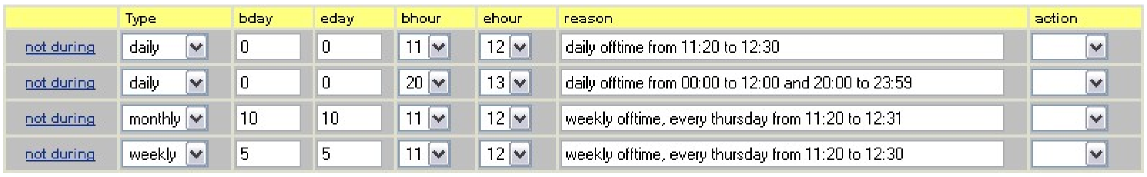
Area ‘off-times’ :

The processing of a chain happens per default always with the specified period. In the ‘off times’ section of the ‘chain configuration’ page, time periods in which no processing will occur can be specified (edited, added deleted).
Note: the ‘reason’ field is for comments only and is not processed.

Off times may be defined on a daily, weekly or monthly base. The days within a week are specified from sunday (1) to Saturday (7), the days within a month are specified from ‘1’ to the number of the last day of the month.

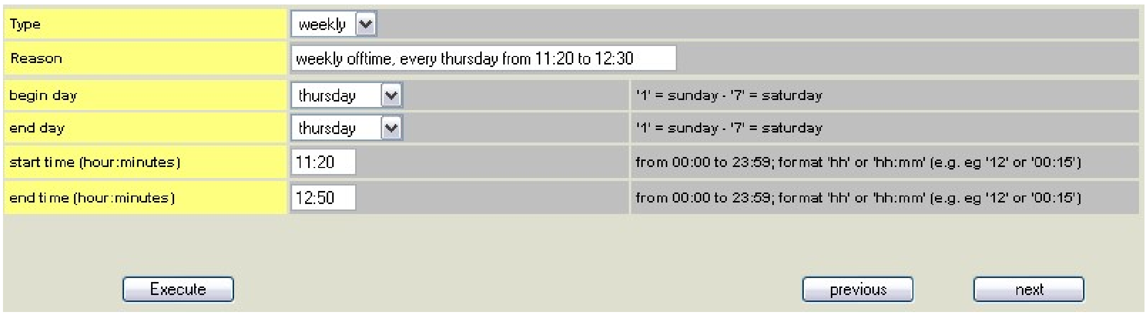
If for daily off-times the begin time is specified to a value after the end time (as you can see in the second entry of below example) the resulting off time is from the beginning of the day to the end time and from the begin time to the end of day.

Off-time entries may be repositioned on the screen using 'drag & drop' (browser dependent).



	Type	bday	eday	bhour	ehour	reason	action
not during	daily	0	0	11	12	daily offtime from 11:20 to 12:30	
not during	daily	0	0	20	13	daily offtime from 00:00 to 12:00 and 20:00 to 23:59	
not during	monthly	10	10	11	12	weekly offtime, every thursday from 11:20 to 12:31	
not during	weekly	5	5	11	12	weekly offtime, every thursday from 11:20 to 12:30	

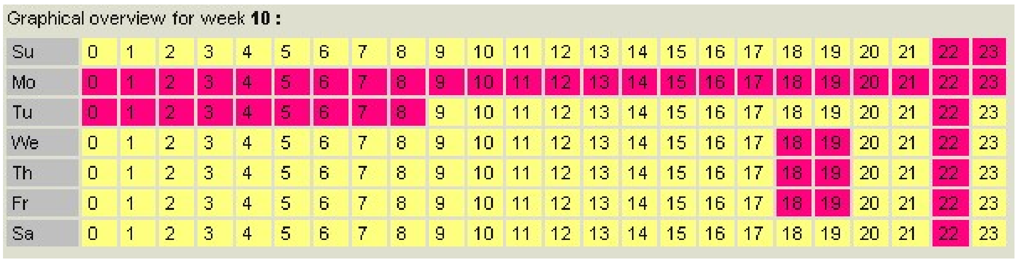
Following the 'not during' link, an off-time entry can be specified in more detail (see the example below). Especially it is possible to specify the begin- and end times on a minute base, and the selection of days within a week is more comfortable.



Type	weekly	
Reason	weekly offtime, every thursday from 11:20 to 12:30	
begin day	thursday	'1' = sunday - '7' = saturday
end day	thursday	'1' = sunday - '7' = saturday
start time (hour:minutes)	11:20	from 00:00 to 23:59; format 'hh' or 'hh:mm' (e.g. eg '12' or '00:15')
end time (hour:minutes)	12:50	from 00:00 to 23:59; format 'hh' or 'hh:mm' (e.g. eg '12' or '00:15')

Using the 'previous' and 'next' buttons it is possible to scroll through the off time entries.

The 'check' link in the left/top cell of the off-times section it is available to generate a graphical representation of off-times for a selectable time period :



Graphical overview for week **10 :**

Su	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	**22**	**23**
Mo	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**
Tu	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	9	10	11	12	13	14	15	16	17	18	19	20	21	**22**	23
We	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	**18**	**19**	20	21	**22**	23
Th	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	**18**	**19**	20	21	**22**	23
Fr	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	**18**	**19**	20	21	**22**	23
Sa	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	**22**	23

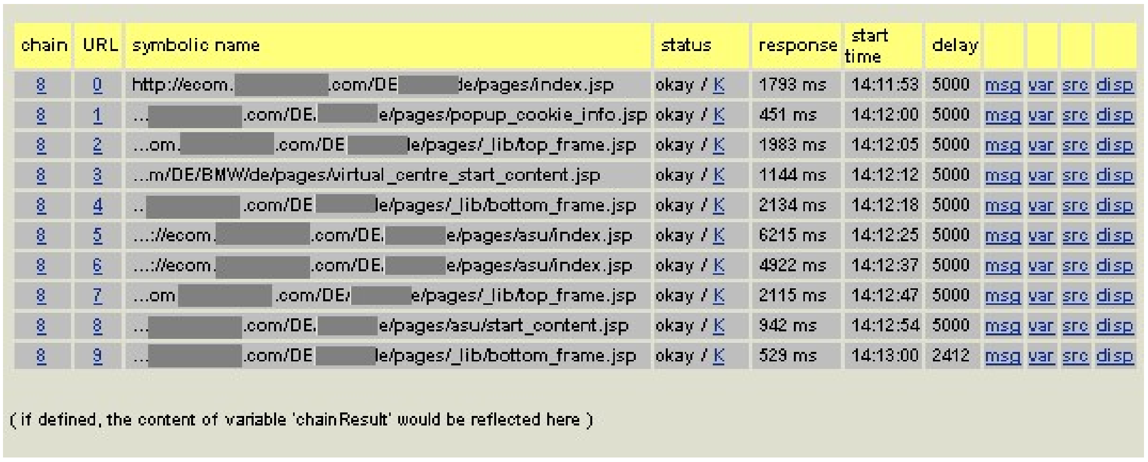
Area 'Buttons' :

The 'Execute' button saves the changed values and/or performs the action chosen in an action field.
Other Buttons allow the scrolling to previous and next chains and the 'one time' execution of the chain.

'One run' execution :

For debugging purposes it is possible to execute only one run of the defined chain. Additional information will be gathered and made available after the processing has finished. 'One run' execution can also be initiated from several other places at SPECTO.

After starting a 'one run' execution the 'one run' screen is displayed. In this screen processed URLs are shown together with the major measurement results. This is screen is automatically refreshed.



chain	URL	symbolic name	status	response	start time	delay				
8	0	http://ecom.[redacted].com/DE[redacted]de/pages/index.jsp	okay / K	1793 ms	14:11:53	5000	msg	var	src	disp
8	1	...[redacted].com/DE.[redacted]e/pages/popup_cookie_info.jsp	okay / K	451 ms	14:12:00	5000	msg	var	src	disp
8	2	...om.[redacted].com/DE[redacted]le/pages/_lib/top_frame.jsp	okay / K	1983 ms	14:12:05	5000	msg	var	src	disp
8	3	...m/DE/BMW/de/pages/virtual_centre_start_content.jsp	okay / K	1144 ms	14:12:12	5000	msg	var	src	disp
8	4	..[redacted].com/DE[redacted]le/pages/_lib/bottom_frame.jsp	okay / K	2134 ms	14:12:18	5000	msg	var	src	disp
8	5	...://ecom.[redacted].com/DE.[redacted]e/pages/asu/index.jsp	okay / K	6215 ms	14:12:25	5000	msg	var	src	disp
8	6	...://ecom.[redacted].com/DE.[redacted]e/pages/asu/index.jsp	okay / K	4922 ms	14:12:37	5000	msg	var	src	disp
8	7	...om [redacted] .com/DE/[redacted]e/pages/_lib/top_frame.jsp	okay / K	2115 ms	14:12:47	5000	msg	var	src	disp
8	8	...[redacted].com/DE.[redacted]e/pages/asu/start_content.jsp	okay / K	942 ms	14:12:54	5000	msg	var	src	disp
8	9	...[redacted].com/DE[redacted]le/pages/_lib/bottom_frame.jsp	okay / K	529 ms	14:13:00	2412	msg	var	src	disp

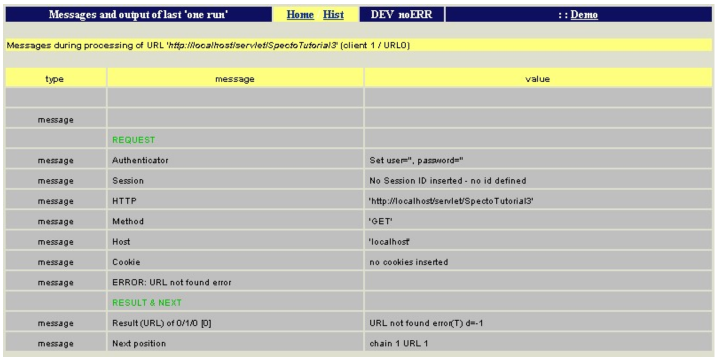
(if defined, the content of variable 'chainResult' would be reflected here)

During or after a 'one run' , the messages, variables, results screen content and result screen view can be displayed for any step. The display of that data is always in a separate browser window; from which multiple can be opened (they have to be closed manually).

Note that following one of the 'chain', 'URL' or 'status' links will terminate the refreshing of the status screen but will not terminate the 'oen run' processing. The status screen may be reactivated using the 'pc w' command from the command line.

Display of messages:

The message window is used to display all messages, including all HTTP parameters and script-outputs of the processed URL.

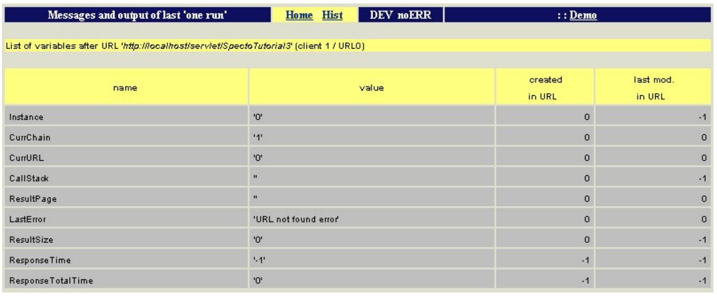


Messages during processing of URL '*http://localhost/servlet/SpectoTutorial3*' (client 1 / URL0)

type	message	value
message		
	REQUEST	
message	Authenticator	Set user='', password=''
message	Session	No Session ID inserted - no id defined
message	HTTP	'http://localhost/servlet/SpectoTutorial3'
message	Method	'GET'
message	Host	'localhost'
message	Cookie	no cookies inserted
message	ERROR: URL not found error	
	RESULT & NEXT	
message	Result (URL) of 0/1/0 [0]	URL not found error(T) d=-1
message	Next position	chain 1 URL 1

This separate window has to be closed manually when it is not used any more.

Display of variables:

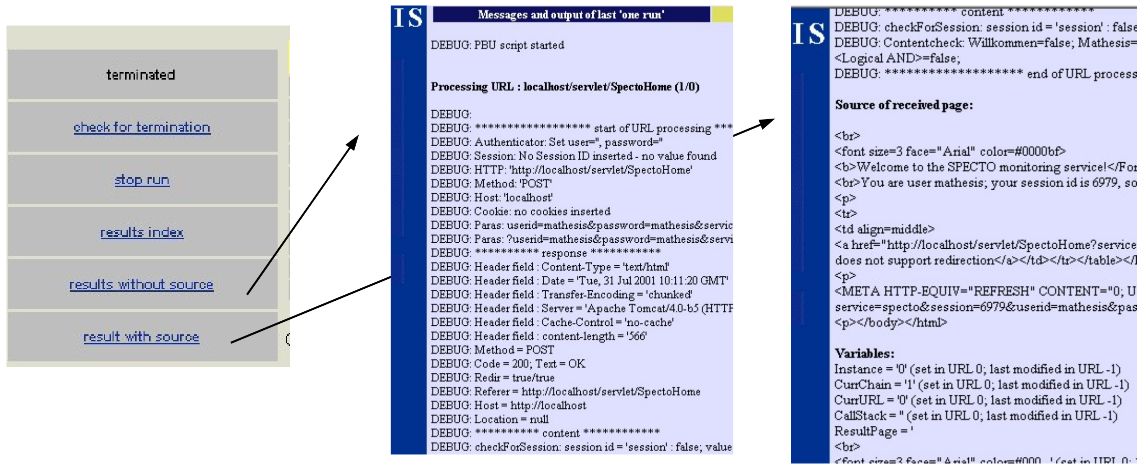


List of variables after URL '*http://localhost/servlet/SpectoTutorial3*' (client 1 / URL0)

name	value	created in URL	last mod. in URL
Instance	'0'	0	-1
CurrChain	'1'	0	0
CurrURL	'0'	0	0
CallStack	''	0	-1
ResultPage	''	0	0
LastError	'URL not found error'	0	0
ResultSize	'0'	0	-1
ResponseTime	'-1'	-1	-1
ResponseTotalTime	'0'	-1	-1

Controlling 'one run' execution

With the link selection in the block on the left side of the 'one run' execution screen it is possible to check and stop the 'one run' execution. Also it is possible to get an index to the complete results , view the complete result (with or without the retrieved html sources).



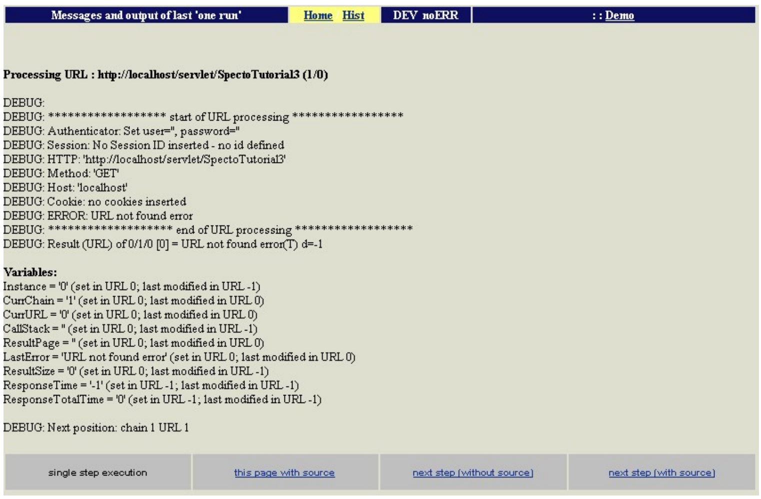
```
DEBUG: PBU script started

Processing URL : localhost/servlet/SpectoHome (1/0)

DEBUG:
DEBUG: ****************** start of URL processing ***
DEBUG: Authenticator: Set user='', password=''
DEBUG: Session: No Session ID inserted - no value found
DEBUG: HTTP: 'http://localhost/servlet/SpectoHome'
DEBUG: Method: 'POST'
DEBUG: Host: 'localhost'
DEBUG: Cookie: no cookies inserted
DEBUG: Paras: userid=mathesis&password=mathesis&servic
DEBUG: Paras: ?userid=mathesis&password=mathesis&servi
DEBUG: ********** response ***********
DEBUG: Header field : Content-Type = 'text/html'
DEBUG: Header field : Date = 'Tue, 31 Jul 2001 10:11:20 GMT'
DEBUG: Header field : Transfer-Encoding = 'chunked'
DEBUG: Header field : Server = 'Apache Tomcat/4.0-b5 (HTTP
DEBUG: Header field : Cache-Control = 'no-cache'
DEBUG: Header field : content-length = '566'
DEBUG: Method = POST
DEBUG: Code = 200; Text = OK
DEBUG: Redir = true/true
DEBUG: Referer = http://localhost/servlet/SpectoHome
DEBUG: Host = http://localhost
DEBUG: Location = null
DEBUG: ********** content ************
DEBUG: checkForSession: session id = 'session' : false; value
```

```
DEBUG: ********** content *************
DEBUG: checkForSession: session id = 'session' : false
DEBUG: Contentcheck: Willkommen=false; Mathesis=
<Logical AND>=false;
DEBUG: ******************* end of URL process

Source of received page:

<br>
<font size=3 face="Arial" color=#0000bf>
<b>Welcome to the SPECTO monitoring service!</For
<br>You are user mathesis; your session id is 6979, so
<p>
<tr>
<td align=middle>
<a href="http://localhost/servlet/SpectoHome?service
does not support redirection</a></td></tr></table></
<p>
<META HTTP-EQUIV="REFRESH" CONTENT="0; U
service=specto&session=6979&userid=mathesis&pas
<p></body></html>

Variables:
Instance = '0' (set in URL 0; last modified in URL -1)
CurrChain = '1' (set in URL 0; last modified in URL -1)
CurrURL = '0' (set in URL 0; last modified in URL -1)
CallStack = '' (set in URL 0; last modified in URL -1)
ResultPage = '
<br>
```

'Single step' execution:

A variant of the 'one run' execution explained in above is 'single step' execution. During 'single step', execution stops after each URL and waits for user interaction.

'Single step' execution will be integrated into 'one run' in a later release.



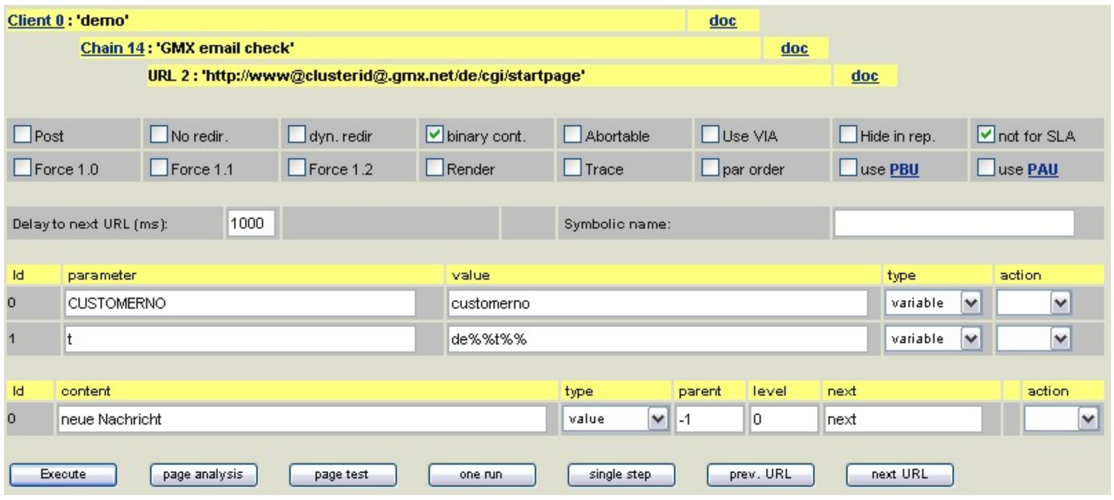
```
Processing URL : http://localhost/servlet/SpectoTutorial3 (1/0)

DEBUG:
DEBUG: ****************** start of URL processing *****************
DEBUG: Authenticator: Set user='', password=''
DEBUG: Session: No Session ID inserted - no id defined
DEBUG: HTTP: 'http://localhost/servlet/SpectoTutorial3'
DEBUG: Method: 'GET'
DEBUG: Host: 'localhost'
DEBUG: Cookie: no cookies inserted
DEBUG: ERROR: URL not found error
DEBUG: ****************** end of URL processing ******************
DEBUG: Result (URL) of 0/1/0 [0] = URL not found error(T) d=-1

Variables:
Instance = '0' (set in URL 0; last modified in URL -1)
CurrChain = '1' (set in URL 0; last modified in URL 0)
CurrURL = '0' (set in URL 0; last modified in URL 0)
CallStack = '' (set in URL 0; last modified in URL -1)
ResultPage = '' (set in URL 0; last modified in URL 0)
LastError = 'URL not found error' (set in URL 0; last modified in URL 0)
ResultSize = '0' (set in URL 0; last modified in URL -1)
ResponseTime = '-1' (set in URL -1; last modified in URL -1)
ResponseTotalTime = '0' (set in URL -1; last modified in URL -1)

DEBUG: Next position: chain 1 URL 1
```

The object hierarchy – 'URLs'

The 'URL configuration' page is used to specify one URL of a chain. It consists of the areas :

- Object hierarchy navigation
- Option flags
- Attributes
- Parameters
- Content
- Buttons

Editing is done by changing the values in the fields and pressing the 'Execute' button directly below. Actions are performed by selecting the action in the 'action' field and pressing the same 'Execute' button. Only one action (but multiple changes are allowed for one button execution.



Client 0 : 'demo' doc

Chain 14 : 'GMX email check' doc

URL 2 : 'http://www@clusterid@.gmx.net/de/cgi/startpage' doc

☐ Post	☐ No redir.	☐ dyn. redir	✓ binary cont.	☐ Abortable	☐ Use VIA	☐ Hide in rep.	✓ not for SLA
☐ Force 1.0	☐ Force 1.1	☐ Force 1.2	☐ Render	☐ Trace	☐ par order	☐ use PBU	☐ use PAU

Delay to next URL (ms): 1000 Symbolic name:

Id	parameter	value	type	action
0	CUSTOMERNO	customerno	variable	
1	t	de%%t%%	variable	

Id	content	type	parent	level	next	action
0	neue Nachricht	value	-1	0	next	

(This screen was generated using the 'eu 14 2' command, or by following the 'Page 2' link in the chain config. screen)

Area 'option flags' :

'Post'	Using HTTP 'POST' instead of 'GET'
'no redir'	Do not use automatic redirection
'dyn. Redir'	Automatic redirection will be performed by SPECTO (must be used together with 'no redir').
'binary content'	the content will not be analyzed as text.
'Abortable'	Allow aborting by monitor..
'Use VIA'	Do not execute this URL but forward it to a SPECTO 'VIA' instance.
'Hide in rep.'	Do not show this URL in reporting.
'not for SLA'	Ignore this URL for SLA ('service level agreement') processing.
'Force…'	Specifies required HTTP protocol levels
'Render'	Perform an HTML rendering (to measure render time).
'par order'	perform automatic parameter ordering based on the preceding page.
'use PBU'	the 'process before URL' script will be executed before every execution of the URL.
'use PAU'	the 'process after URL' script will be executed after every execution of the URL.

Area ‘Attributes’

The ‘delay to next URL’ field accepts a wait time (in milliseconds) before the execution of the next URL. If not entered or entered as ‘0’ a default of 5000 milliseconds will be used.

A symbolic name can be entered in the ‘Symbolic name’; this can be used as reference instead of the URL sequence number.

Areas ‘Parameters’ and ‘Content’

In the URL configuration any number of parameters (which will go into the ‘params’ section of the requesting URL) can be specified. If a session id is specified in the corresponding chain configuration, this will be included automatically.

Also any number of content tags can be specified and computed to form a result and (optionally) a follow up page.

See chapter ‘Page specification’ for details.

Area ‘Buttons’

The ‘Execute’ button saves the changed values and/or performs the action chosen in an action field.
‘Page analysis’ retrieves the URL and analyzes the content.
‘Page test’ calls the page with the specified parameters and applies the content check.
‘one run’ starts a ‘one run’ processing of the corresponding chain.

Page specification

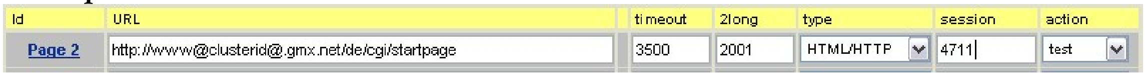
URL specification

URLs are specified according to the syntax of the W3C.

The 'type' field is only used if the URL does not specify the type by itself. In that case, if the type field is 'default' the type of the chain is used.

SPECTO variables can be used in URLs, there they have to be framed with '@' characters (as in …@clusterid@.. in the example below).

Example :



Id	URL	timeout	2long	type	session	action
Page 2	http://www@clusterid@.gmx.net/de/cgi/startpage	3500	2001	HTML/HTTP	4711	test

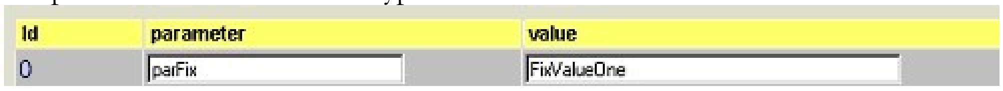
note: It may be necessary to experiment with omitting or specifying the leading 'http://' and a trailing '/'.

URL parameter specification

Any web page which allows for user input (graphical elements like buttons, checkboxes and input fields) communicates with the web server thru the use of parameters. Any parameter has a name and an associated value.
SPECTO allows for the definition of fixed and variable parameters. The parameters are supplied to the target page in the order they are specified; this may be important for pages consisting of multiple forms.
A parameter may refer to a specific form within a page using the colon format : <form>:<parameter>.

Fixed parameters

If a parameter is defined with type 'direct' its value is the text entered in the 'value' column.

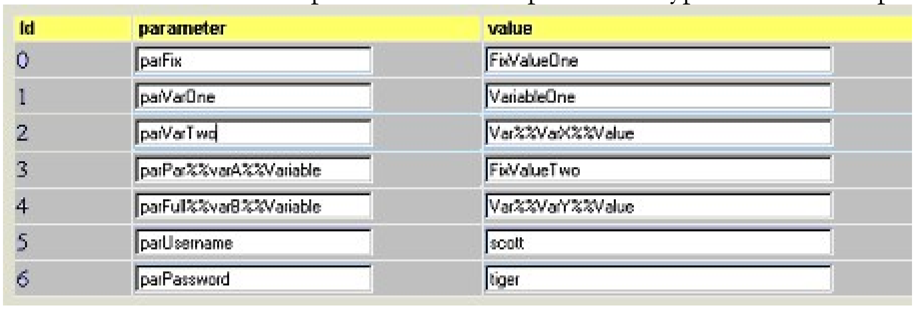


Id	parameter	value
0	parFix	FixValueOne

In the above example the URL is supplied with the parameter 'parFix' which has the value 'FixValueOne'.

Dynamic parameters

Parameter values (and the names) can be computed dynamically by the use of variables. Variables can be used in parameters if the parameter type 'variable' is specified.



Id	parameter	value
0	parFix	FixValueOne
1	parVarOne	VariableOne
2	parVarTwo	Var%%VarX%%Value
3	parPar%%varA%%Variable	FixValueTwo
4	parFull%%varB%%Variable	Var%%VarY%%Value
5	parUsername	scott
6	parPassword	tiger

In the simplest case the 'value' part is taken as the variable's name and will be substituted at runtime with the variable's value (example: Id 1).

If the parameter value consists of fix and variable (even multiple) parts, the variable name(s) have to be enclosed in '%%' tags. (example: Id 2; if the value of variable 'VarX' is 'iable' then the parameter 'parVarTwo' is submitted with the value 'VariableValue' to the URL.

Variables can also be used to specify the parameter names (example: Id 3; here, if the value of variable 'varA' is 'ameter', a parameter 'parParameterVariable' with the value 'FixValue' is submitted to the URL.)
Variables can be used in parameters names and parameter values at the same time (example: Id 4).

Normally variables are local to the chain. If a variable is preceeded with an underscore '_' it is considered global within the client and commands 'vw', 'vr' and 'vl' can be used to set and read it.

The following variables are (among others) supplied SPECTO automatically :

• '`currURL`'	address of the current URL
• '`resultSize`'	number of characters in the result page
• '`resultPage`'	the content of the result page

For web pages using the HTTP authorization mechanism ('WWW-Authenticate' header) the special parameter types 'username' and 'password' are available. (examples: Id. 5 and Id. 6).

External parameter computation

Parameter values can be computed by an external routine written in the Java programming language and called by SPECTO's 'exit' mechanism (see also chapter 'exits'). The syntax for an 'exit' type parameter is a set of three subparts divided by colons : 'class:method:parameter', the subparts may also be or include variables. (See also the next chapter for a description of 'exit' programming).

Special parameter formats

Besides the typical parameter formats like ‘text/html’ or ‘application/x-www-form-urlencoded’ which are supplied automatically, special formats can be selected using the ‘contentType’ property. (see also section ‘Advanced settings’).
The following content types are supported :
‘multipart/form-data boundary=--xxx’
all parameters will be enclosed with the specified boundary and the parameter name is transmitted as a ‘name=”<parameter name>”’ construct.

Parameter ordering

Parameters are supplied to the URL in the order as they are defined in the URL configuration page. If the flag ‘par order’ id set in the URL configuration screen SPECTO tries to compute the parameter order from the last page accessed. (‘automatic parameter ordering’).

Parameters not found in this process are moved to the end (by preserving their original order).

Any parameter can be excluded from this processing by preceding the parameter name with ‘noorder:’.

If automatic parameter ordering is necessary only for a small subset of the specified parameters, this parameters can be scheduled for automatic ordering by preceding them with ‘toorder:’. In this case the flag ‘par order’ must not be selected.

Page status computation

The status of a processed URL or XML request is dependent of the content of the received page/xml-object. To compute the page status an arbitrary number of content specifying tags (‘content tags’) can be specified, and, by the use of ‘meta tags’, can be computed using logical ‘and’, ‘or’ and ‘not’ operators.

Content check

Any content tag of type ‘value’ is considered a text string; its value is true if the text can be found in the web page. This comparison is case-insensitive.

Logic

Logical expressions can be constructed using meta tags. The following meta tags are available :
- AND type = 0
- OR type = 1
- NOT type = 2

For the example in 'editing a web-page configuration' (page 9) the following formulas are specified :

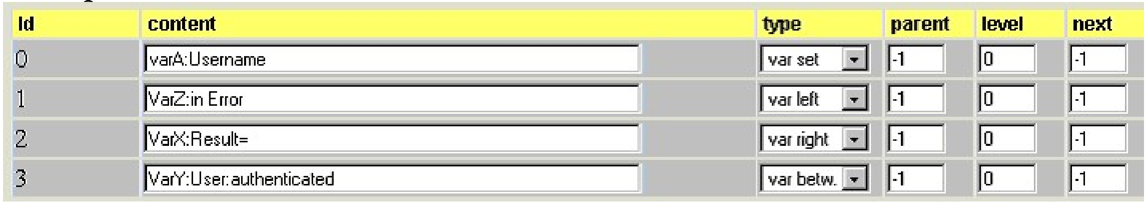
```
If  ( page contains 'welcome' or 'willkommen' )
   and page contains 'SPECTO'
   then continue with the next page of the chain.
If  page contains 'nicht gültig'
   then continue with page 2 of the chain.
```

Usage of variables

During processing of a chain variables can be used to transmit information dynamically from one URL to the next URL.
Variables can be set fix, computed according to a formula or extracted from selected parts of the page returned from the current URL.

- var set/cmp content = "variable_name : variable_value". The 'variable_value' can be a constant or computed (see preceeding chapter 'variables in parameters). Instead of assignments also comparisons can be made using '<', '>', '=' and '#' operators. During comparison first a numeric evaluation of the two components will be tried; if that fails an alphanumeric comparison will be made.
- var left content = "variable_name : search_text". Extracts the text left from 'search_text' and stores it in the variable 'variable_name'.
- var right content = "variable_name : search_text". Extracts the text right from 'search_text' and stores it in the variable 'variable_name'.
- var betw. content = "variable_name : search_text_left : search_text_right". Extracts the text between 'search_text_left' and 'search_text_right' and stores it in the variable 'variable'.
- Custom types 100 to 999

Examples :



Id	content	type	parent	level	next
0	varA:Username	var set	-1	0	-1
1	VarZ:in Error	var left	-1	0	-1
2	VarX:Result=	var right	-1	0	-1
3	VarY:User:authenticated	var betw.	-1	0	-1

Custom type (Exits)

Custom types are submitted together with the content values to the customer supplied java class :

```
   e.g. 'SpectoCustom',
```

method :

```
   e.g. 'String computeContentLogic(String id, String param)'
```

The Parameter 'Id' will be supplied automatically.

Then the content entry could be (using variable 'SearchText') :
SpectoCustom:computeContentLogic:%%SearchText%%

The method has to separate the page result from the supplied parameter. Both are concatenated, separated by a binary zero ('\u0000') and are supplied in the parameter 'param'). The content check is assumed okay if the method returns the string 'true'.

The demonstartion method shown below checks if the supplied parameter (the content of variable 'SearchText' in the demonstration entry) is part of the page returned by the URL.

```
// content check method
   public static String computeContentLogic
           (String id, String param) {
   String  result  = "";
   String  page    = "";
   int     index0  = 0;

           if ((index0=param.indexOf("\u0000")) >= 0) {
                  page    = param.substring(index0+1);
                  param   = param.substring(0, index0);
           }
           if (page.toLowerCase().indexOf(param) >= 0)
                  result = "true";
           return result;
   }
```

Advanced settings (‘set’ command)

The processing of URLs can be fine tuned by a variety of properties. Those properties can only be set in ‘process …’ scripts. The script command ‘set’ is used for setting properties. (See also chapter ‘SPECTO Script’).

Available properties :

Property name	Description	Allowed values / examples
defaultFollowUp	Base directory of SPECTO installation	
deleteEmptyCookies	cookies with an empty content will not be transmitted to the target	true, false (default is ‘true’)
deleteExpiredCookies	cookies which are expired will not be transmitted to the target	true, false (default is ‘true’)
deleteCookiesWoExp	cookies without an expiration date will be ignored	true, false (default is ‘false’)
loadIncludedObjects	All objects specified in the accessed HTML page (graphics, frames, etc.) are also loaded and their load time becomes part of the page load time	true, false (default is ‘false’)
maxIncludedObjects	Maximum number of objects to load.	Default is ‘25’.
monitorWaitTime	time in milliseconds the monitor process waits before a waiting chain thread is reported as an error (‘waitloop’ error)	numeric (default is ‘30000’)
appletInitWaitTime	time in milliseconds the URL is delayed after an applet was init	numeric (default is ‘5000’)
limitMaxAutoRedir	during auto redirection this property specifies whether the maximum number of redirections will be limited (to 10).	true, false (default is ‘false’)
notificationEnable	controls generation of notifications, e.g. in sub-chains.	true, false (default is ‘true’)
notificationAddError	controls whether any or only the first error within a chain will increase the error counter.	true, false (default is ‘true’)
notificationSubject	the subject used for a notification (where applicable, e.g. email, SMS, fax). The subject is extended with ‘remember’ / ‘release’ flags and the client/chain/url hierarchy.	any text, including variables. An empty setting displays the default text.

allowGzipEncoding	allow the web server to return result in zip format.	true, false (default is 'false')
allowCompressEncoding	allow the web server to return result in compress format.	true, false (default is 'false')
allowDeflateEncoding	allow the web server to return result in deflate format.	true, false (default is 'false')
contentType	the HTTP 'content-type' header entry. Several headers (e.g. the 'multipart' family) get special treatment.	GET: text/html POST: application/x-www-form-urlencoded
binaryRead	Binary read instead of ASCII/Text read	true, false (default is 'true')
HTTPMethodVersion	Not yet used	-
noParamSeparators	The parameters specified for an URL are not separated by the '&' character (useful e.g. for applets)	true, false (default is 'false')
SOAPAction	The optional HTTP 'SOAPAction' header field.	
socketEarlyClose	During socket operations the sending socket is closed directly after the content has been transmitted.	true, false (default is 'false')
socketMaxWaitTime	The maximum time (specified in milliseconds) the system waits for an answer on the receiving socket.	A positive number. Default is 10000.
nativePollWait	If, after issuing a native command, the system will wait for the result.	true, false (default is 'true')
nativeMaxWaitTime	The maximum time (specified in milliseconds) the system waits for an answer on an issued native command.	GET: text/html POST: application/x-www-form-urlencoded
dispLenVar	The number of characters shown of a variables content during 'single step' or 'one run'.	A positive number. Default is 60.
userAgent	the HTTP 'user-agent' header entry.	default: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT; DigExt)
maxResultSize	Only during 'binary reads' : Limit the size of the result page read from the web server; the rest is discarded. SPECTO does not wait for the end of the transmission.	in bytes (default is '1000000')
writeSubResults	save also the timing results of chains called via GOSUB.	true, false (default is 'false')
delayToNextURL	time in milliseconds to wait before the execution of the	any positive number

	next chain; this overwrites the GUI '**Delay to next URL**' parameter.	

Network protocol properties

Following the 'net.' Prefix the following networking properties can be set (example : 'set net.http.ProxyHost myProxy.myCompany.com') :

java.net.preferIPv6Addresses	Use IP6 addresses if available.	true, false (default is 'false')
http.proxyHost	Network name (not ip address) of the HTTP proxy	any positive number
http.proxyPort	Port of the HTTPS proxy server	any positive number between 0 and 65535. Default is 80.
http.nonProxyHosts	Network domains or names (not IP addresses) not to be accessed using the proxy.	List of names, separated by a '|' character. Example : *.foo.com|localhost
https.proxyHost	Network name (not ip address) of the HTTPS proxy server machine.	Any dns name.
https.proxyPort	Port of the HTTPS proxy server	any positive number between 0 and 65535. Default is 80.
https.nonProxyHosts	Network domains or names (not IP addresses) not to be accessed using the proxy.	List of names, separated by a '|' character. Example : *.foo.com|localhost
ftp.proxyHost	Network name (not IP address) of the ftp proxy	any positive number
ftp.proxyPort	Port of the ftp proxy server	any positive number between 0 and 65535. Default is 80.
ftp.nonProxyHosts	Network domains or names (not IP addresses) not to be accessed using the proxy.	List of names, separated by a '|' character. Example : *.foo.com|localhost
socks.proxyHost	Network name (not IP address) of the http proxy	any positive number
socks.proxyPort	Port of the proxy server	any positive number between 0 and 65535. Default is 1080.

B2B properties

Also special b2b properties are available; they are explained in chapter 'B2B'.

A list of all properties with their default values can be displayed using command '`properties`'.

Follow up computation

The page status computation may also return the number of the next page to be executed (column 'next').

The following values are supported :

Value	Function
Positive number	go to the URL with that index
-1 ('next')	the next URL of the chain will be used (sequential order, this is the default processing).
-2 ('break')	terminate this run of the chain processing
-3 ('onError')	if no error occured, go to the next URL of the chain (same as –1), otherwise terminate this run.
-4 ('byVar')	then, if specfied, go to the URL specified in variable 'nextURL' otherwise go to the next URL (like -1)
-5 ('return')	Return to the chain and URL of the last call
-6 ('quit')	Quit the processing of this chain thread.
-7 ('repeat')	Repeat this URL
between -100 and –999	the number is interpreted as -xyy with x being the error class+1 and yy the URL index which will be switched to if the preceeding URL ended in an error of the specified error class. (error classes: 0=timeout, 1=toolong, 2=content, 3=custom, 4=waitloop). eg. -202 = if error was 'tooLong' then go to URL index 2
between -1000 and –99999	the number is interpreted as –xx0yy with xx being the chain index + 1 and yy the URL index which will be switched to. e.g. –4002 go to URL 2 of chain 3
any other case	(which normally should not happen) just go to the next URL (like -1)

Instead of the above numeric notation the chain and URL may also be specified with their name / symbolic name in the 'colon' format :

```
chain_name:URL_symbolic_name
```

Example :

If the chain name is 'Auto-Specto' and one of its URLs has the symbolic name 'StartPage' then 'Auto-Specto:StartPage' is a legal follow-up value. Instead of the URL's symbolic name also its Id may be used; in the example here, if the URL 'StartPage' is the first URL : 'Auto-Specto:0'.

Working with text

Treatment of special characters

Especially when specifying URLs or their parameters it may become necessary to work with special characters which cannot be displayed by the SPECTO web front end and/or cannot be stored within a database without conversion. Examples are all ascii values below 32 and the ', " and " characters.

SPECTO supplies a syntax to specify any character as its hexadecimal representation using following syntax :

\u*nnnn* where *nnnn* is the hexadecimal specified Unicode value of the character.

Abbreviations :

\\ is equal to the backslash character
\r is equal to **\u000d**
\n is equal to **\u000a**

Special characters in the above notation may be used almost at any place where SPECTO allows the entry of text.

Wherever text can be entered, SPECTO allows for a linkage to ***text areas*** (see next section for a discussion of text areas) by using the \l<linkname>\l syntax. The content of the specified text area will replace the linkage at run time. The link name can be numeric id of the text area, or its symbolic name.

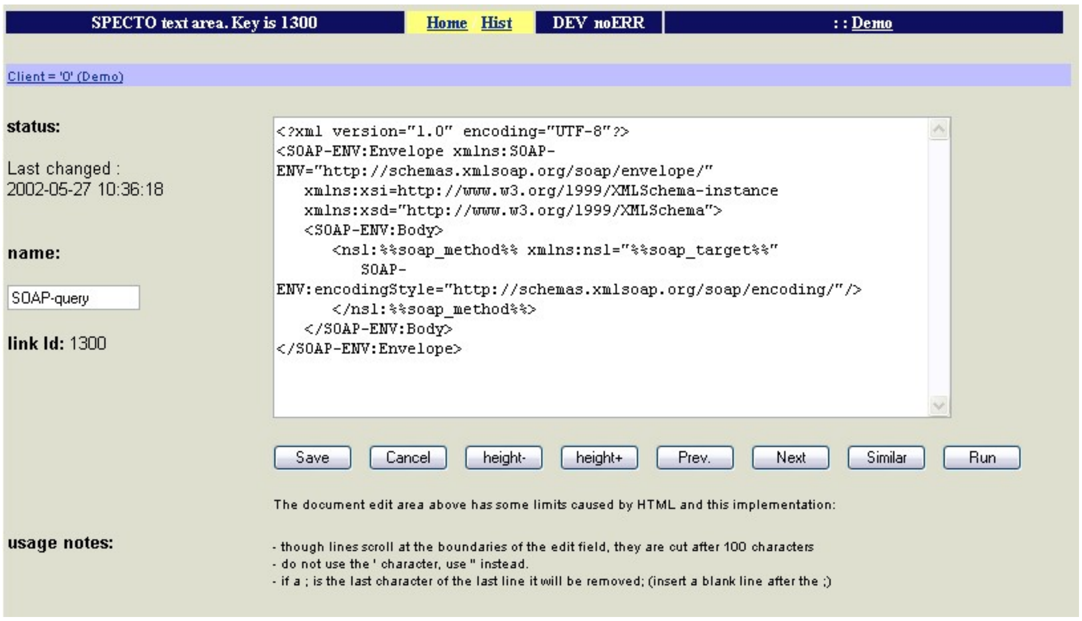
Example :

If the text area 'company' would consist of the text 'my company at my home' and it a text field has the content 'this is /lcompany/l' then the resulting content of the text field (at runtime) would be 'this is my company at my home'.

Text areas

In certain situations a constant text is used in multiple places, or a SPECTO text field may be too small to hold a specific text. For such cases SPECTO provides ***text areas*** which are persistent and unlimited in size. Text areas are maintained using the 'ta <linkname>' command. Links to text areas can be inserted at any place where special characters are permitted and can be nested. An irresolvable text area link will not insert any text.

The link name may be numeric or a symbolic name.



Client = '0' (Demo)

status:

Last changed : 2002-05-27 10:36:18

name: SOAP-query

link Id: 1300

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi=http://www.w3.org/1999/XMLSchema-instance
   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
      <nsl:%%soap_method%% xmlns:nsl="%%soap_target%%"
         SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </nsl:%%soap_method%%>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The document edit area above has some limits caused by HTML and this implementation:

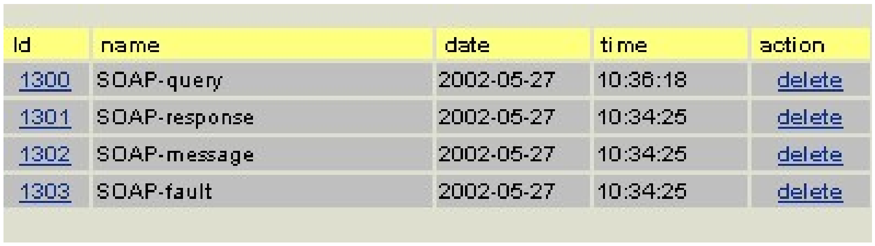
usage notes:

- though lines scroll at the boundaries of the edit field, they are cut after 100 characters
- do not use the ' character, use " instead.
- if a ; is the last character of the last line it will be removed; (insert a blank line after the ;)

Text areas refer to SPECTO variables using the '%%<variable_name>%%' syntax. They may include other text areas using the 'link' syntax.

The text area above was displayed using the '`ta SOAP-query`' command. The 'Prev' and 'Next' buttons allow for navigation within the defined text areas.

The defined text areas may be listed using the 'tal [prefix] command. For example the command '`tal SOAP`' will display a list similar to the one displayed below:



Id	name	date	time	action
1300	SOAP-query	2002-05-27	10:36:18	delete
1301	SOAP-response	2002-05-27	10:34:25	delete
1302	SOAP-message	2002-05-27	10:34:25	delete
1303	SOAP-fault	2002-05-27	10:34:25	delete

Following the 'Id' link will go to the edit screen with the selected text area loaded and selecting the 'delete' link will remove the text area from the database.

Creating a new text area :

To create a new text area it is first required to create the text area with a new, unique id. To find an unused id, use the '`tal`' command. Then create the new text area using '`ta <new_id>`'. The above editing screen will appear, and the content of the text area and a symbolic name (which may be used instead of the id) can be entered. Then use 'Save' to save the new text area to the database.
Command '`ta <new_name> create`' may be sued to create a text area with a symbolic name, in this case the id will be computed automatically (id range above 10000).

Text areas are cached by SPECTO; you may use command '`ca`' to review the cache status.

Page analysis

The ‘`pa <url>`‘ command (example: ‘`pa www.altavista.de`’) is provided to analyze an existing web page. The information returned may be used to create a SPECTO parameter entry.
Analysis is done for :

- Frames
- Title and other unique information which may be used for identification
- References (‘links’)
- Forms
- Scripts (e.g. ‘javascript’ / ‘jscript’)
- Applets
- Unknown constructs

Example result returned for ‘www.altavista.de’ :

SPECTO Command output:

```
Analysis of page 'http://www.altavista.de' :
Titles :
Line 2 :    altavista.de® - deutschland - suche
Frames :
- none -
Forms :
Line 72 :   action="/cgi-bin/query" name=mfrm
Line 78 :   action=/cgi-bin/domainame method=post
References :
Line 42 :   ="/index.html"
Line 68 :   ="http://ad.de.doubleclick.net/jump/homepgtable…
Line 71 :   ="http://ad.de.doubleclick.net/jump/homepgtable…
Line 72 :   ="http://ad.de.doubleclick.net/jump/homepgtable…
Line 73 :   ="http://www.spiegel.de/sport/fussball/0,1518,alt…
Line 74 :   ="http://www.spiegel.de/wissenschaft/0,1518,alt…
Line 75 :   ="/smart"
Line 76 :   ="http://www.zdnet.de/news/artikel/2000/09/090…
Line 77 :   ="http://www.zdnet.de/news/artikel/2000/09/090…
Line 78 :   ="/doc/help/h_se_image_help_001.html"
Scripts :
Line 12 :   language="javascript1.2"
Line 27 :   language="javascript"
Input fields :
Line 72 :   =hidden name=pg value=q
Applets :
- none -
Ambiguous :
- none -
```

Session management

General

There are two session managing technologies used in the (normally stateless) HTTP environment: ***session-ids*** and ***cookies.***

Session Id

Theory of operation:
(After a login) the web server provides the session identification as a name/value pair in a hidden input value of a form. When the user selects a component of the form (e.g. a button) the name/value pair will be automatically inserted in the URL by the browser.

Specto operation:
Specto requires the name of the session id (often 'session') to be specified in the chain configuration. Whenever the session id is recognized in the page submitted by the web server its value is stored. The session ids name and its stored value will then be included (as first parameter) of every URL sent by SPECTO.

Cookies

Theory of operation:
The web server sends (always / once ?) a cookie (represented as an entry in the page header with the tag 'Set-Cookie' and the value in the form 'name=value' to the browser. In every communication with that web server the browser transmits the cookie(s) in the request header.

Specto operation:
Whenever a cookie is detected in the response header sent from the web server Specto extracts and stores both name and value. When the same page is accessed again, Specto adds the cookie(s) as fields (tag name 'Cookie') of the request header.
Cookie operation in Specto is automatic, no configuration is required.

SPECTO maintains session ids in the 'session' field of a URL configuration line. If multiple session ids are required, they are concatenated in this field separated by ';'.

Reference measurements

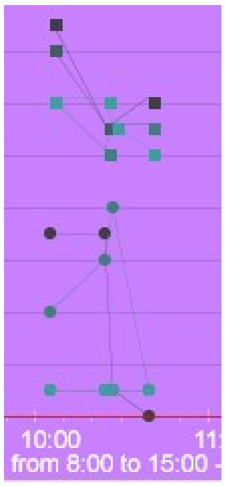
General

Measurements of a chain can be set in relation to a reference. SPECTO supports two types of references***: reference-URLs*** and ***URL grouping***.

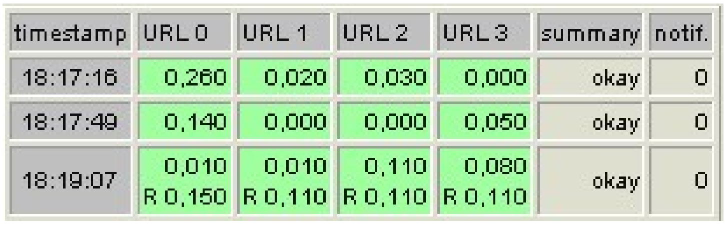
Reference URLs

A reference URL is a single URL specified using properties 'referenceURLChain' and 'referenceURLurl'. Whenever 'referenceURLChain' is set (usually during a PBT or PBC event), for every URL of the chain a measurement of the specified reference URL is also made. If 'referenceURLurl' is set (usually during a PBU event) this setting overrides the 'referenceURLchain setting for the next URL.

Example: `set referenceURLchain "http://www.yahoo.de";`



Reference measurements are output during reporting in graphical and tabular form in the daily views (commands 'rg' and 'rh'). In the tabular form reference values are shown in addition to the original measurement as a separate line prefixed by an 'R'. In the graphical view reference measurements are displayed as square points in the same color as the referenced original measurements.



timestamp	URL 0	URL 1	URL 2	URL 3	summary	notif.
18:17:16	0,260	0,020	0,030	0,000	okay	0
18:17:49	0,140	0,000	0,000	0,050	okay	0
18:19:07	0,010 R 0,150	0,010 R 0,110	0,110 R 0,110	0,080 R 0,110	okay	0

Reference measurements are also visible in the reporting list (command 'rl'), marked by the 'reference' identifier.

URL grouping

Standard reporting which displays all measurements of all URLs of a chain can be enhanced in two ways :

- hiding of selected URLs. This is done using the flag 'Hide in rep.' in the URL configuration form (this only affects the display of measurement results, the measurements are taken nevertheless)
- adding various other chain/URL measurements (URL grouping)

URL grouping allows specifying a set of URL measurements of other chains which are displayed in addition to the URLs of the selected chain. The additional URLs are specified as trailing parameter of 'rg'/'rh' commands : '{ chainId/URLId, }'. The syntax is detailed in chapter 'Reporting'.

Non-HTML contents

Overview

Standard communication and content formatting in the web is based on standards of the W3C and relies on the HTTP protocol and the HTML format. Due to certain limitations, over the time extensions have emerged. The most common are :

- Applets by Sun corporation
- The PDF document format by Adobe corporation
- Flash by Macromedia corporation

Also special applications may communicate using the HTTP protocol directly.

HTTP direct

Special solutions may be constructed by directly accessing the HTTP protocol. The necessary logic has to be maintained in a 'java script' program.
The supplied functions are :

- HEAD (all four types are supported)
- POST
- PUT
- GET
- DELETE
- OPTIONS
- CONNECT
- VALUE=<value>

The HTTP direct commands are entered in URL pages in the form : 'http-direct::<function>:header-entries:content.

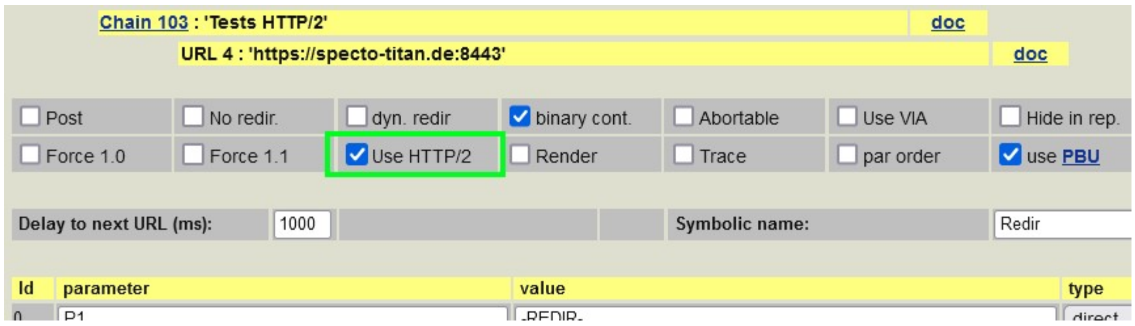
Multiple header entries must be separated by semi-colons (';').
Content may also be supplied by the usage of parameters.
Variables, text-areas and content-check are available without restrictions

Examples :
http-direct::POST:content-type=proprietary;content-length=32:logon
http-direct::VALUE=\u0027::%%result%%

HTTP Protocol Version 2

The version 2 of the HTTP protocol, commonly referred to as HTTP/2 can be used instead of HTTP version 0.9, 1.0 or 1.1.

The usage of HTTP/2 can be specified on an URL basis in the GUI:



Chain 103 : 'Tests HTTP/2' doc

URL 4 : 'https://specto-titan.de:8443' doc

☐ Post	☐ No redir.	☐ dyn. redir	✓ binary cont.	☐ Abortable	☐ Use VIA	☐ Hide in rep.
☐ Force 1.0	☐ Force 1.1	✓ Use HTTP/2	☐ Render	☐ Trace	☐ par order	✓ use PBU

Delay to next URL (ms): 1000 Symbolic name: Redir

Id	parameter	value	type
0	P1	-REDIR-	direct

Or it can be defaulted (logical 'OR'ed with the GUI based specification) in a `PBx` script by setting the '`defaultUseHttp2`' property.

Example script:
```
  specto.setProperty("defaultUseHttp2", true);
```

All common HTTP/1.x functionalities like Redirects, Cookies, Proxy, Authorization, certificates work identically when using the HTTP/2 protocol version. Especially the certificate stores (for client and server certificates) is shared among the two protocol families.
However note that, because the SPECTO HTTP/2 implementation is internally base don a completely different protocol stack, the messages during '`OneRun`' debugging are slightly different.

HTTP/2 Server Push

SPECTO URL access using the HTTP2 protocol understands 'PUSH_PROMISE' frames and automatically integrates them in the response content.
To give the server enough time to issue a PUSH, the SPECTO wait time before ending the request can be set using the '`pushWaitTimeInMS`' property (default 0 milliseconds, indicating that no server PUSH is expected).

Example script:
```
  specto.setProperty("pushWaitTimeInMS", 1000);
```

Web Sockets

Web Sockets are an extension of the classic HTTP protocol to allow a bidirectional, message orientated communication between the client (e.g. a browser, or a SPECTO instance) and a Web Socket enabled server.

The usage of the web socket protocol is indicated by using '`ws:`' or '`wss:`' in the scheme part of the URI:

ws`://specto-rhodium.de:8444`	for unencrypted communication

or

wss`://specto-rhodium.de:8443`	for encrypted communication

The SPECTO engine then automatically performs the defined Web Socket mechanism:

- Establish a HTTP or HTTPS connection to the Web Socket host
- Inquire an '`Upgrade`' to '`websocket`' via the 'Upgrade' HTTP header
- Supplies a key in the "`sec-websocket-key`" parameter
- Checks the server response and verifies the 'Accept' token supplied in the "`sec-websocket-accept`" header field.

Then the communication is switched from the client/server HTTP to the bidirectional Web Socket communication.

The following SPECTO properties (which may be set in a `PBx` script using the '`.setProperty()`' method) further detail the communication:

`wssUseHttp2`	Whether the communication shall start using the HTTP/1.x (default) or HTTP/2 protocol version.
`wssWaitTimeInMS`	The time (in milliseconds) to wait for server responses (the Web Socket protocol allows for multiple response) after sending the message to the server. Default is 500 milliseconds.
`wssParamsAsSepMsgs`	Whether each URL parameter shall be communicated as a separate Web Socket message, or (default) the parameters will be concatenated and transmitted as a single message

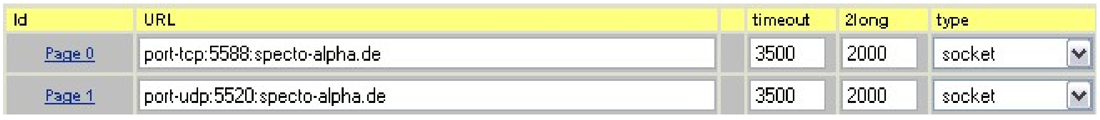
Example script:
```
  specto.setProperty("wssWaitTimeInMS", 1000);
  specto.setProperty("wssUseHttp2", true);
  specto.setProperty("wssParamsAsSepMsgs", true);
```

Socket / Port access

Above methods are correlated with socket services, described in chapter **'Port / socket services'.**

Port/Socket requests use the following formats:
Port-tcp:*port-number:url-or-ip* *and* Port-udp:*port-number:url-or-ip*

Example:



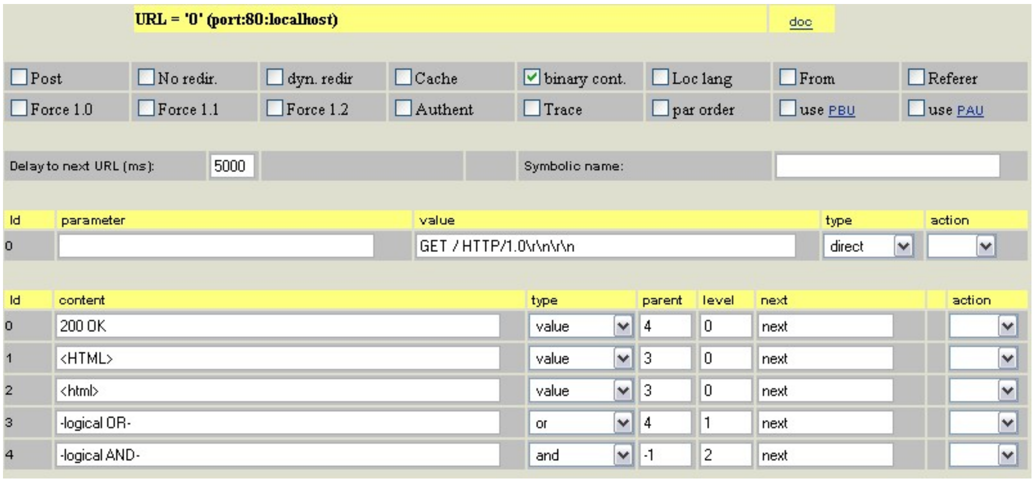
Id	URL	timeout	2long	type
Page 0	port-tcp:5588:specto-alpha.de	3500	2000	socket
Page 1	port-udp:5520:specto-alpha.de	3500	2000	socket

Parameter and result processing is identical to other URL-types.

Above methods only check the availability of the socket service; the answer from the SPECTO engine is 'okay' if the connection could be established, or an error message if not.

For scenarios where a more detailed communication has to be established the 'port:' (alternate name 'socket:') method can be used. It fully features SPECTO parameters and content check options. The syntax is identical to that of 'port-tcp:'.

Example :



URL = '0' (port:80:localhost) doc

☐ Post	☐ No redir.	☐ dyn. redir	☐ Cache	✓ binary cont.	☐ Loc lang	☐ From	☐ Referer
☐ Force 1.0	☐ Force 1.1	☐ Force 1.2	☐ Authent	☐ Trace	☐ par order	☐ use PBU	☐ use PAU

Delay to next URL (ms): 5000 Symbolic name:

Id	parameter	value	type	action
0		GET / HTTP/1.0\r\n\r\n	direct	

Id	content	type	parent	level	next	action
0	200 OK	value	4	0	next	
1	<HTML>	value	3	0	next	
2	<html>	value	3	0	next	
3	-logical OR-	or	4	1	next	
4	-logical AND-	and	-1	2	next	

There are two definable TCP-timeouts for Port access : '`portConnectTimeout`' and '`portSoTimeout`'; both are available as attributes and as script-properties. Their values are in milliseconds; the defaults are 5000 and 2500.

Ping

SPECTO features an ICMP protocol implementation allowing for ping requests.

Syntax :
ping:<hostname>

The resulting output is similar to :
Source:
```
192.168.73.200 : delay=10
```

Telnet / SSH access

SPECTO supports the automation of command dialogs using traditional ***telnet*** or secure ***ssh*** protocols as it is defined in RFC 854.

SSH supports **IDEA, Blowfish** and **RSA/PKCS#1** encryption.

Syntax :

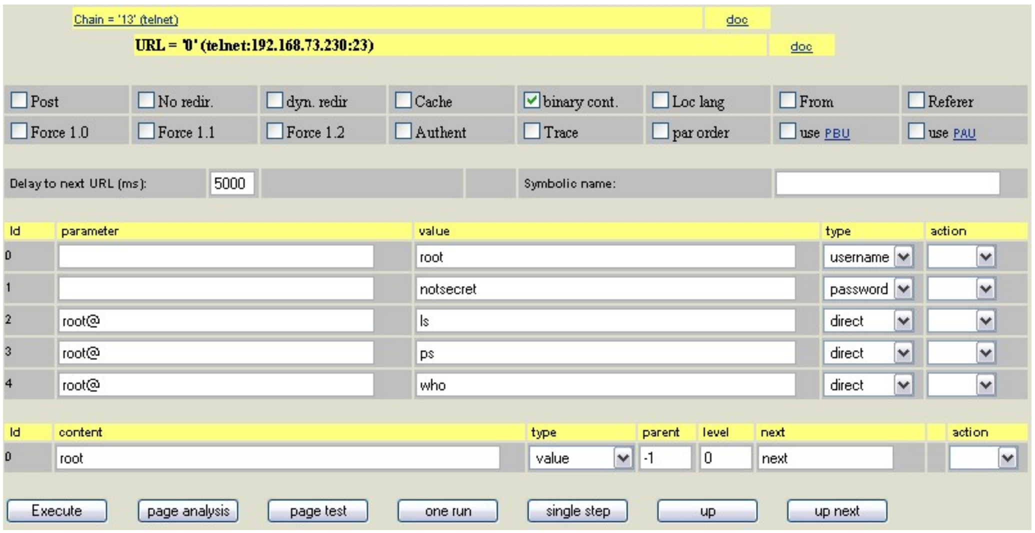
telnet:<hostname>[: <port>]	default port number is 23
ssh:<hostname>[: <port>]	default port number is 23

Parameters :

Username and password have to be supplied with the standard SPECTO mechanism (types 'username' and 'password').
In any additional parameter entry the 'value' field is used as a command which is send to the target. Before the sending of a new command, the received output is parsed for the occurrence of the pattern specified in the 'parameter' field, to assure that the system has responded completely to the previous command (the logon is considered a command also). Usually the command line prompt of a system is used for that check. UNIX systems regularly use 'username'@'systemname' as prompt.
Note: If the specified value does not match the string returned by the target the SPECTO engine will wait until a socket timeout will occur (usually after about 30 seconds).

The common SPECTO content check mechanisms are available to parse the output returned from the target.

Example :


Chain = '13' (telnet) doc
URL = '0' (telnet:192.168.73.230:23) doc

☐ Post	☐ No redir.	☐ dyn. redir	☐ Cache	✓ binary cont.	☐ Loc lang	☐ From	☐ Referer
☐ Force 1.0	☐ Force 1.1	☐ Force 1.2	☐ Authent	☐ Trace	☐ par order	☐ use PBU	☐ use PAU

Delay to next URL (ms): 5000 Symbolic name:

Id	parameter	value	type	action
0		root	username	
1		notsecret	password	
2	root@	ls	direct	
3	root@	ps	direct	
4	root@	who	direct	

Id	content	type	parent	level	next	action
0	root	value	-1	0	next	

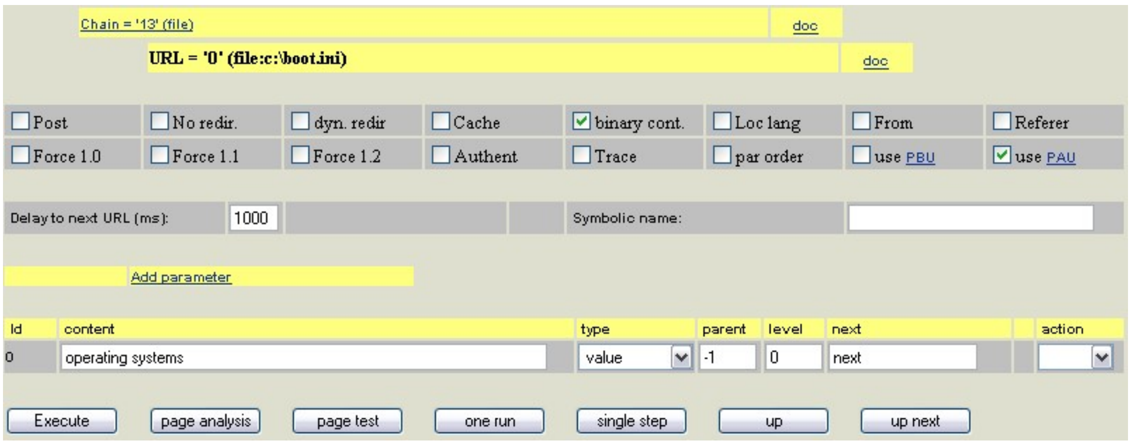
In the above example the user 'root' with password 'notsecret' is using telnet protocol to log in to system 192.168.73.230 on port 23. Then the three commands 'ls', 'ps' and 'who' are executed. Before the execution of the commands it is waited for the string 'root@' to appear in the returned answer.

File access

Files in text or binary formats can be read from the SPECTO engine's target machine's file system or any remote file system mounted by NFS, SMB/Samba.

Syntax :
file: [<share>] <filename>

Example :



Chain = '13' (file) doc
URL = '0' (file:c:\boot.ini) doc

☐ Post	☐ No redir.	☐ dyn. redir	☐ Cache	✓ binary cont.	☐ Loc lang	☐ From	☐ Referer
☐ Force 1.0	☐ Force 1.1	☐ Force 1.2	☐ Authent	☐ Trace	☐ par order	☐ use PBU	✓ use PAU

Delay to next URL (ms): 1000
Symbolic name:

Add parameter

Id	content	type	parent	level	next	action
0	operating systems	value	-1	0	next	

The content of the fetched file is available as SPECTO content and can be used for content checks.

SNMP ('mib')

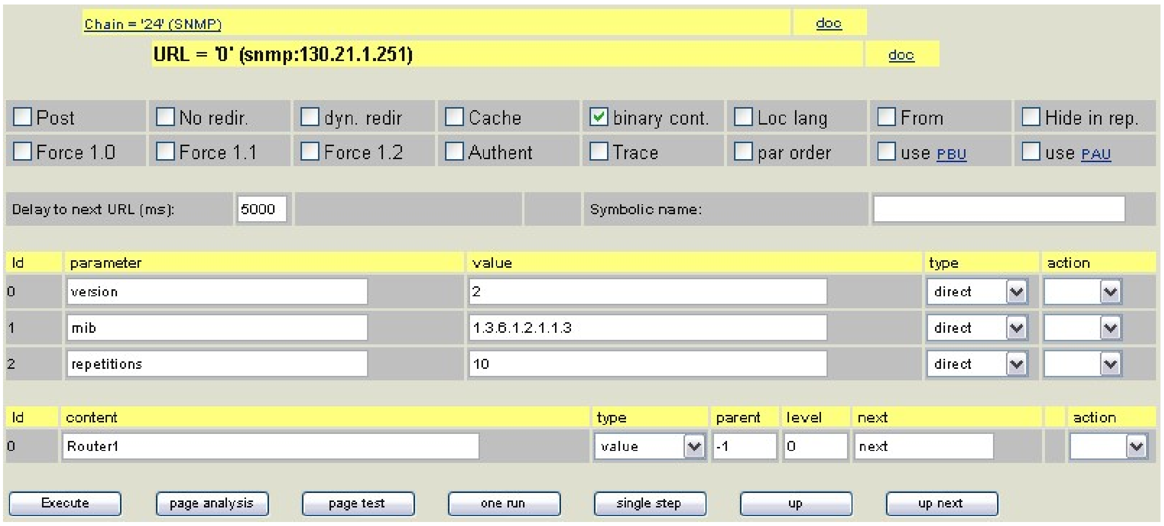
MIB variables can be read using SNMP protocol in versions 1, 2c and 3.

Syntax :
snmp:<hostname> [: <mib_variable>]

The parameter section is used to specify properties of the access. The following properties are available :

Name	description	example
version	SNMP protocol version. Valid are 1, 2 and 3.	2
port	The port for the SNMP protocol. Default = 161.	161
mib	The (first) mib variable to be fetched. This overwrites a mib variable specified in the URL	1.3.6.1.2.1.1.3
community	The SNMP community	public
repetitions	The number of successive mib variables to be fetched (not available for version 1)	10
context	Context (only available for version 3)	public
engine	Engine Id (only available for version 3)	010000a1d41e4946
contact	Contact (only available for version 3)	
user	Username for access (only available for version 3)	guest
password	The password (only available for version 3)	insecure
authprotocol	The protocol for authentication. ('MD5' or 'SHA1'). Default = 'MD5'.	SHA1
debug	Level of console messages (0 to 15) 0 denotes no console messages. Default = 0	0

Example :


Chain = '24' (SNMP) doc
URL = '0' (snmp:130.21.1.251) doc

☐ Post	☐ No redir.	☐ dyn. redir	☐ Cache	☑ binary cont.	☐ Loc lang	☐ From	☐ Hide in rep.
☐ Force 1.0	☐ Force 1.1	☐ Force 1.2	☐ Authent	☐ Trace	☐ par order	☐ use PBU	☐ use PAU

Delay to next URL (ms): 5000 Symbolic name:

Id	parameter	value	type	action
0	version	2	direct	
1	mib	1.3.6.1.2.1.1.3	direct	
2	repetitions	10	direct	

Id	content	type	parent	level	next	action
0	Router1	value	-1	0	next	

The values of the fetched mib variables are available as SPECTO content; they can be used for content checks.

File transfer protocol (FTP)

There are two specifications for file transfer using the tcp/ip file transfer protocol :
- rfc 959 (the older)
- rfc 1738 (the newer, URL based)

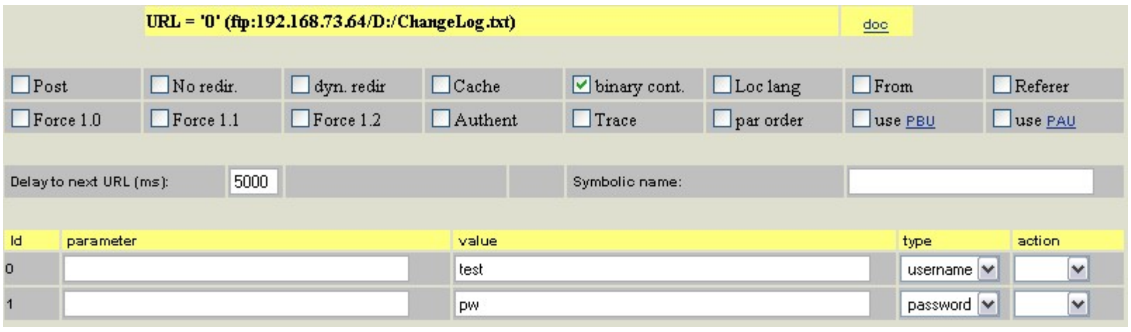
Both specifications are supported by SPECTO.

RFC 959

Syntax :
ftp:<hostname>/<filename>

You have to supply username and password as parameters (types 'username' and 'password)

Example :



URL = '0' (ftp:192.168.73.64/D:/ChangeLog.txt) doc

☐ Post	☐ No redir.	☐ dyn. redir	☐ Cache	✓ binary cont.	☐ Loc lang	☐ From	☐ Referer
☐ Force 1.0	☐ Force 1.1	☐ Force 1.2	☐ Authent	☐ Trace	☐ par order	☐ use PBU	☐ use PAU

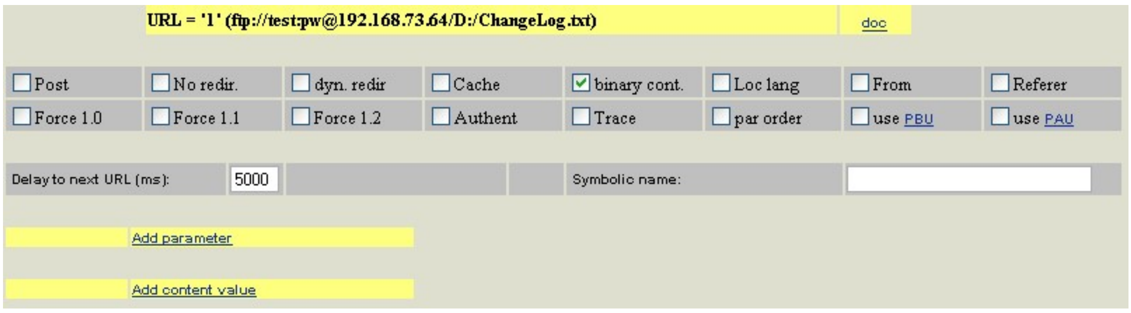
Delay to next URL (ms): 5000 Symbolic name:

Id	parameter	value	type	action
0		test	username	
1		pw	password	

RFC 1738

Syntax :
ftp://<user>:<password>@<hostname>/<filename>

Example :



URL = '1' (ftp://test:pw@192.168.73.64/D:/ChangeLog.txt) doc

☐ Post	☐ No redir.	☐ dyn. redir	☐ Cache	✓ binary cont.	☐ Loc lang	☐ From	☐ Referer
☐ Force 1.0	☐ Force 1.1	☐ Force 1.2	☐ Authent	☐ Trace	☐ par order	☐ use PBU	☐ use PAU

Delay to next URL (ms): 5000 Symbolic name:

Add parameter

Add content value

The content of the fetched file is available as SPECTO content and can be used for content checks.

SQL (relational database) access

SPECTO supports full database DML and DDL as supported by available JDBC database drivers. It is possible to register additional vendor supplied database drivers. The following databases are supported directly :

• ODBC	(Windows platform only)	driver id 0 / n.a.
• Borland JDataStore		driver id 3 / n.a.
• Oracle		driver id 5 / 6 (UNIX)
• MS SQL server		driver id 10 / n.a.
• SAP SAPDB		driver id 11 / 12 (UNIX)
• Sybase Adaptive Server		driver id 13 / 14 (UNIX)
• mySQL		driver id 15 / 16 (UNIX)
• Hypersonic		driver id 17
• Cloudscape (releases >= 10)		driver id 19

Syntax :
sql: [<database_driver_id> |] database_URL

Parameters :

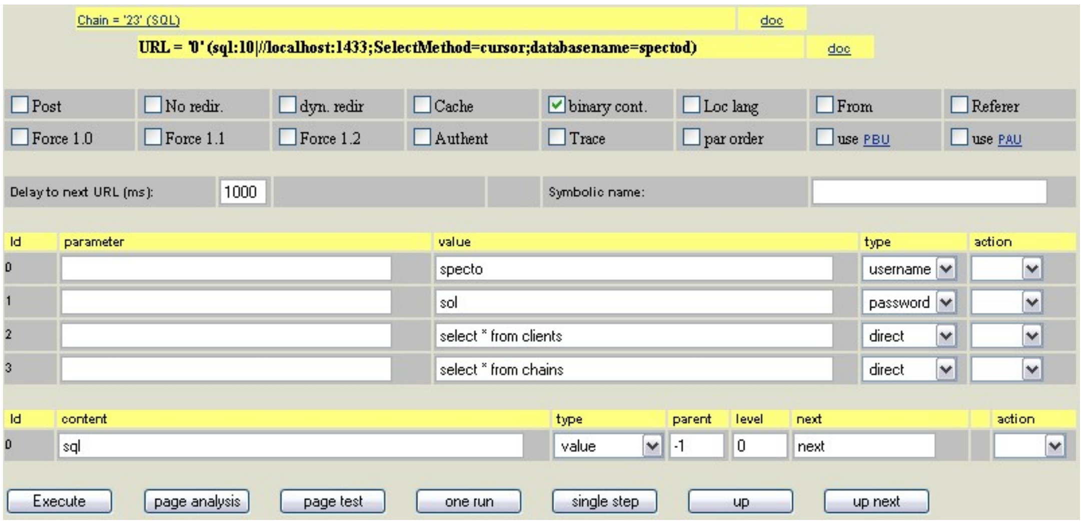
Username and password have to be supplied with the standard SPECTO mechanism (types 'username' and 'password').
In any additional parameter entry the 'value' field is used as a SQL command which is send to the target.

The parameter field denotes the type of access made to the database:
If empty a SQL *query* is performed, if 'statement' a SQL *statement* (statements are SQL commands which are not expected to return results, e.g. an insert) is executed, if 'prepared', a SQL *prepared statement* is defined and executed (e.g. to execute a stored procedure; also with some databases DDL statements also have to be executed as prepared statements).

The generated outputs includes all rows separated by CR/LF and is prefixed with a row of the column names.
The common SPECTO content check mechanisms are available to parse the output returned from the target.

For database specific information see also chapter 'Databases'.

Example :



Chain = '23' (SQL) doc

URL = '0' (sql:10|//localhost:1433;SelectMethod=cursor;databasename=spectod) doc

Post	No redir.	dyn. redir	Cache	✓ binary cont.	Loc lang	From	Referer
Force 1.0	Force 1.1	Force 1.2	Authent	Trace	par order	use PBU	use PAU

Delay to next URL (ms): 1000 Symbolic name:

Id	parameter	value	type	action
0		specto	username	
1		sol	password	
2		select * from clients	direct	
3		select * from chains	direct	

Id	content	type	parent	level	next	action
0	sql	value	-1	0	next	

In the above example the MS SQLServer database 'spectod' on machine 'localhost', port 1433 is accessed using username 'specto' with password 'password'.

Two SQL selects ('select * from clients' and 'select * from chains') are executed in sequence.

LDAP (directory) access

SPECTO supports LDAP v3 protocol to read LDAP ('lightweight directory access protocol') directories :

Syntax :
ldap: [<version> |] directory_URL [:port]

Parameters :

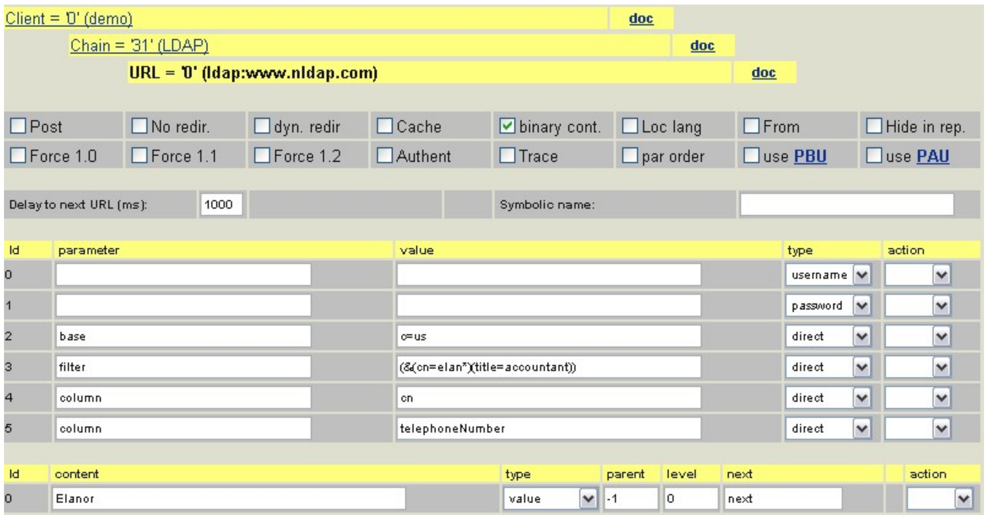
Username and password have to be supplied with the standard SPECTO mechanism (types 'username' and 'password'). If not specified or empty no authorization is performed. The search base (may be empty for the root), search string (in LDAP filter format.), and the columns to be returned, are specified using parameters 'base', 'filter' and 'column'. Multiple columns can be specified, if none specified all columns are returned.

The generated output includes all rows separated by CR/LF. Every row starts with the 'distinguished name', followed by ' : ' and then the list of columns. Every column's value is separated by ' = ' from the column's name.

Attributes 'ldap.maxWait' and 'ldap.maxItems' (menu 'customizing – network – protocols') allow to restrict the wait time and the number of returned items.

Example :

This example queries a public available LDAP directory (by Novell Inc.) for all entries in the 'us' branch with the name starting with 'ela' and the tile being 'accountant'. Returned are the columns 'cn' and 'telephoneNumber'. A check against 'Elanor' is performed.


Client = '0' (demo) doc
Chain = '31' (LDAP) doc
URL = '0' (ldap:www.nldap.com) doc

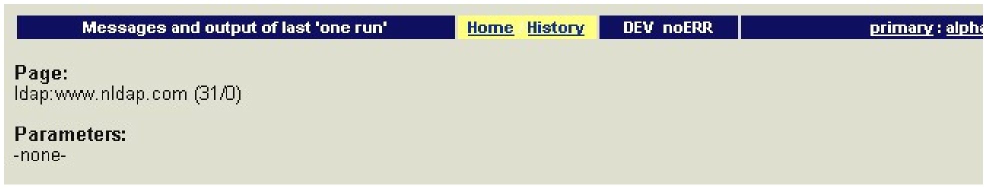
☐ Post	☐ No redir.	☐ dyn. redir	☐ Cache	☑ binary cont.	☐ Loc lang	☐ From	☐ Hide in rep.
☐ Force 1.0	☐ Force 1.1	☐ Force 1.2	☐ Authent	☐ Trace	☐ par order	☐ use PBU	☐ use PAU

Delay to next URL (ms): 1000 Symbolic name:

Id	parameter	value	type	action
0			username	
1			password	
2	base	c=us	direct	
3	filter	(&(cn=elan*)(title=accountant))	direct	
4	column	cn	direct	
5	column	telephoneNumber	direct	

Id	content	type	parent	level	next	action
0	Elanor	value	-1	0	next	

Above example LDAP query returns the following output (obtained via 'one run') :


Messages and output of last 'one run' Home History DEV noERR primary : alpha

Page:
ldap:www.nldap.com (31/0)

Parameters:
-none-

SOAP (web services) access

SPECTO supports SOAP version 0.9, 1.0 and 1.1 to access web services : SPECTO supplies three different implementations of SOAP access :

- Based on the 'Apache' SOAP framework
- Based on the SUN webservices 'JAXM / SAAJ' SOAP framework
- Based on the SPECTO internal XML capabilities and SOAP document library

'Apache based' SOAP access :

The reason for usage of the Apache SOAP library is to supply the 'defacto' SOAP standard.

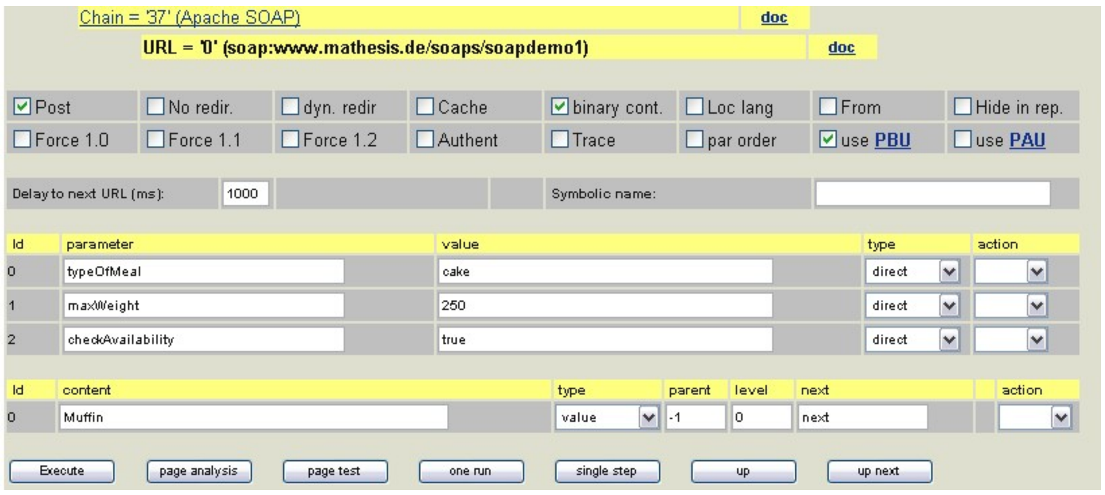
Syntax :
soap: [<version> |] web_service_URL

Parameters :

Parameters are specified with their parameter name and an optional Type (= java class) separated by a ':' character. If no parameter type is specified SPECTO auto detects Text, numerical and Boolean parameters.
The common SPECTO content check mechanisms are available and are applied to the textual representation (Java '.toString()' method) return value of the called SOAP method.

A WSDL specification may be checked against the supplied parameters by specifying the (script) property 'checkSOAPforWSDL <WSDL>'.

Example :


Chain = '37' (Apache SOAP) doc
URL = '0' (soap:www.mathesis.de/soaps/soapdemo1) doc

✓ Post	No redir.	dyn. redir	Cache	✓ binary cont.	Loc lang	From	Hide in rep.
Force 1.0	Force 1.1	Force 1.2	Authent	Trace	par order	✓ use PBU	use PAU

Delay to next URL (ms): 1000
Symbolic name:

Id	parameter	value	type	action
0	typeOfMeal	cake	direct	
1	maxWeight	250	direct	
2	checkAvailability	true	direct	

Id	content	type	parent	level	next	action
0	Muffin	value	-1	0	next	

In the above example the web service function 'soapdemo1' is executed on host 'www.nls.de/soaps'.

JAXM/SAAJ based SOAP access :

Syntax :
soapj: function | web_service_URL

Function :
The function is a single name or an aggregate of the form 'function ; namespace ; urn'.
Note : If the function is preceded with a '!' character the request is not really executed but the generated SOAP request is returned as response (for debugging purposes).

Parameters :

Parameters are specified with their parameter name and optional attributes separated by a '|' character. Any attribute is of the form 'name=value'; the name is specified like the function name.
Complex parameter contents should be prepared using a script during PBU and transferred via a variable. See chapters 'SpectoScript' (page 96) and 'JavaScript' (page 123) for details.

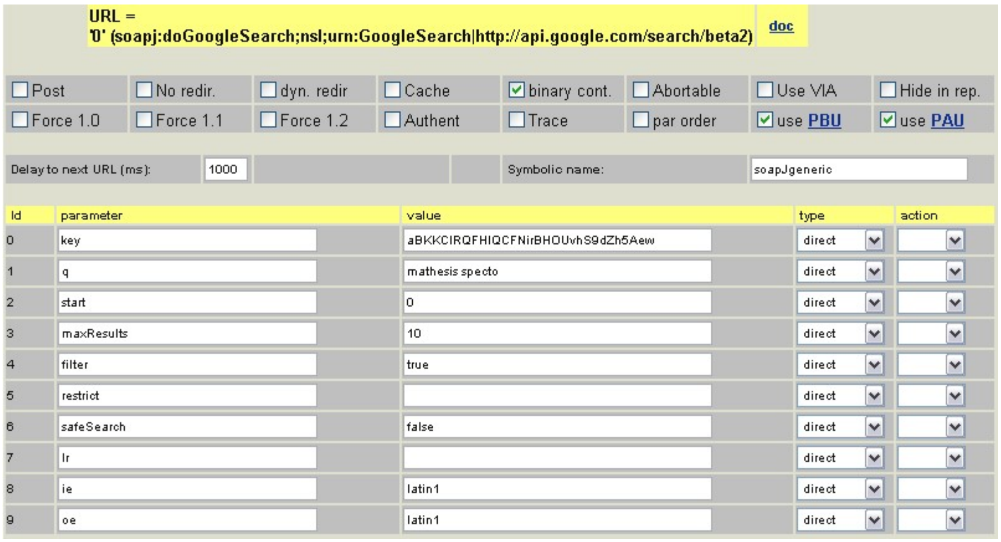
If a parameter is prefixed with a '.', it and following parameters will be positioned within the parameter before; if a parameter is prefixed wit '..' it and the following parameters will be positioned within the parent of the current parameter.

If a parameter name is prefixed with '!', it is inserted as a SOAP attachment instead of a element within the request.

The common SPECTO content check mechanisms are available to parse the output returned from the target. A PAU script may be used to extract content into variables to be used during content check.

Example :

Web search via Google's web API :



URL =
'0' (soapj:doGoogleSearch;nsl;urn:GoogleSearch|http://api.google.com/search/beta2) doc

☐ Post	☐ No redir.	☐ dyn. redir	☐ Cache	✓ binary cont.	☐ Abortable	☐ Use VIA	☐ Hide in rep.
☐ Force 1.0	☐ Force 1.1	☐ Force 1.2	☐ Authent	☐ Trace	☐ par order	✓ use PBU	✓ use PAU

Delay to next URL (ms): 1000
Symbolic name: soapJgeneric

Id	parameter	value	type	action
0	key	aBKKCIRQFHIQCFNirBHOUvhS9dZh5Aew	direct	
1	q	mathesis specto	direct	
2	start	0	direct	
3	maxResults	10	direct	
4	filter	true	direct	
5	restrict		direct	
6	safeSearch	false	direct	
7	lr		direct	
8	ie	latin1	direct	
9	oe	latin1	direct	

In the above example the Google web service function 'doGoogleSearch' with parameters 'key' to 'oe' is executed on host 'http://api.google.com/search/beta2'.

In that example the following XML request is generated :

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV= "http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <nsl:doGoogleSearch xmlns:nsl="urn:GoogleSearch">
         <key>8D22DSBQFHK/eS/HHRO925CbTRVwmjMH</key>
         <q>NLS specto</q>
         <start>0</start>
         <maxResults>10</maxResults>
         <filter>true</filter>
         <restrict/>
         <safeSearch>false</safeSearch>
         <lr/>
         <ie>latin1</ie>
         <oe>latin1</oe>
      </nsl:doGoogleSearch>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

returning this result :

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/1999/XMLSchema">
   <SOAP-ENV:Body>
      <ns1:doGoogleSearchResponse xmlns:ns1="urn:GoogleSearch"
            SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <return xsi:type="ns1:GoogleSearchResult">
         <directoryCategories
               xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/"
               xsi:type="ns2:Array"
               ns2:arrayType="ns1:DirectoryCategory[0]">
            </directoryCategories>
            <documentFiltering xsi:type="xsd:boolean">false</documentFiltering>
            <endIndex xsi:type="xsd:int">10</endIndex>
            <estimateIsExact xsi:type="xsd:boolean">false</estimateIsExact>
            <estimatedTotalResultsCount xsi:type="xsd:int">17</estimatedTotalResu…
            <resultElements xmlns:ns3="http://schemas.xmlsoap.org/soap/encoding/"
                  xsi:type="ns3:Array" ns3:arrayType="ns1:ResultElement[10]">
               <item xsi:type="ns1:ResultElement">
                  <URL xsi:type="xsd:string">http://www.NLS.de/Specto/</URL>
                  <cachedSize xsi:type="xsd:string">3k</cachedSize>
                  <directoryCategory xsi:type="ns1:DirectoryCategory">
                     <fullViewableName xsi:type="xsd:string">Top/Science</fullViewable…
                     <specialEncoding xsi:type="xsd:string"></specialEncoding>
                  </directoryCategory>
                  <directoryTitle xsi:type="xsd:string"><b&gt;NLS</b&gt;</dire…
                  <hostName xsi:type="xsd:string"></hostName>
                  <relatedInformationPresent xsi:type="xsd:boolean">true</related…
                  <snippet xsi:type="xsd:string">Optimization</snippet>
                  <summary xsi:type="xsd:string">Business monitoring</summary>
                  <title xsi:type="xsd:string"></title>
               </item>
               <item xsi:type="ns1:ResultElement">
               ...
            </resultElements>
            <searchComments xsi:type="xsd:string"></searchComments>
            <searchQuery xsi:type="xsd:string">NLS Specto</searchQuery>
            <searchTime xsi:type="xsd:double">0.182448</searchTime>
            <searchTips xsi:type="xsd:string"></searchTips>
            <startIndex xsi:type="xsd:int">1</startIndex>
         </return>
      </ns1:doGoogleSearchResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

For more complex SOAP requests it is possible to use a PBU script to define the request and a PAU script to analyze the response. The SPECTO JAXM based SOAP implementation is fully integrated into the SPECTO Javascript script language and allows the unlimited usage of all JAXM methods.

Getting a JAXM reference : `<soapj> = specto.getSOAPJ();`

The reference retrieved with above function 'specto.getSOAPJ()' includes the following predefined objects :

```
SOAPConnectionFactory      scf;
SOAPConnection             connection;
MessageFactory             mf;
SOAPMessage                request;
SOAPMessage                reply;
SOAPPart                   part;
SOAPEnvelope               envelope;
SOAPHeader                 header;
SOAPBody                   body;
SOAPElement                soapFunc;
```

They can be maintained with all methods of the JAXM and SAAJ packages. Description about that packages can be downloaded from SUN microsomputer's Java web site for SOAP processing ('http://java.sun.com/xml/saaj/index.jsp').

A sample PBU script to define a SOAP request :

```
// script=js;
// demo for JAXM based SOAP pre-processing

// make and get the SOAP JAXM environment
soapj = specto.getSOAPJ();

// helper function for parameter insertion
function addc(name, value) {
  ce = soapj.soapFunc.addChildElement(
    soapj.envelope.createName(name));
  ce.addTextNode(value);
}

// the main routine
if (true) {
  specto.message("constructing SOAPJ request");
// create the function
  func = "doGoogleSearch;nsl;GoogleSearch";
  fName = soapj.createName(soapj.envelope, func);
  soapj.soapFunc =
    soapj.body.addChildElement(fName);
// add the parameters
  addc("key", "aBKKClRQFHIQCFNirBHOUvhS9dZh5Aew");
  addc("q", "NLS specto");
  addc("start", "0");
  addc("maxResults", "8");
  addc("filter", "true");
  addc("restrict", "");
  addc("safeSearch", "false");
  addc("lr", "");
  addc("ie", "latin1");
  addc("oe", "latin1");
// note that parameters defined on the screen
// are added to (after) those defined here
}
```

A sample PAU script to analyze the returned SOAP response :

```
//script=js;
// example of PAU for JAXM based SOAP access

// a helper function for node hierarchy
function walk (tChild) {
var children, currChild, numC;
  specto.message("child: " + tChild.getNodeName() +
tChild.getNodeValue());
  if (tChild.hasChildNodes()) {
    children = tChild.getChildNodes();
    numC = children.getLength();
    for(i=0; i<numC; i++) {
      currChild = children.item(i);
      specto.message("xxx : " + currChild.getNodeName());
      walk(currChild);
    }
  }
}

// get the instance of the SOAP JAXM class
soapj = specto.getSOAPJ();

// errors occured ?
specto.message("errors : " + soapj.body.hasFault());

// the children
mainChildren = soapj.body.getChildElements();
while (mainChildren.hasNext()) {
  child = mainChildren.next();
  walk(child);
}

// info about attachments
numa=soapj.reply.countAttachments();
specto.message("Number of attachments : " + numa);
if (numa > 0) {
  attachments = soapj.reply.getAttachments();
  for(i=0; i<numa; i++) {
    specto.message(i + " : " +
      attachments.next().getContent().toString());
  }
}
```

SOAPJ PAU processing is also possible using SPECTOScript it provides the following functions for that :

SpectoScript :
```
<reply>  = soapj.get "reply";
<countA> = soapj.numAttachments;
<valueA> = soapj.getAttachment <index>;
<sizeA>  = soapj.getAttachmentSize <index>;
```

Generic SOAP access :

Syntax :
http: web_service_URL

Parameters :

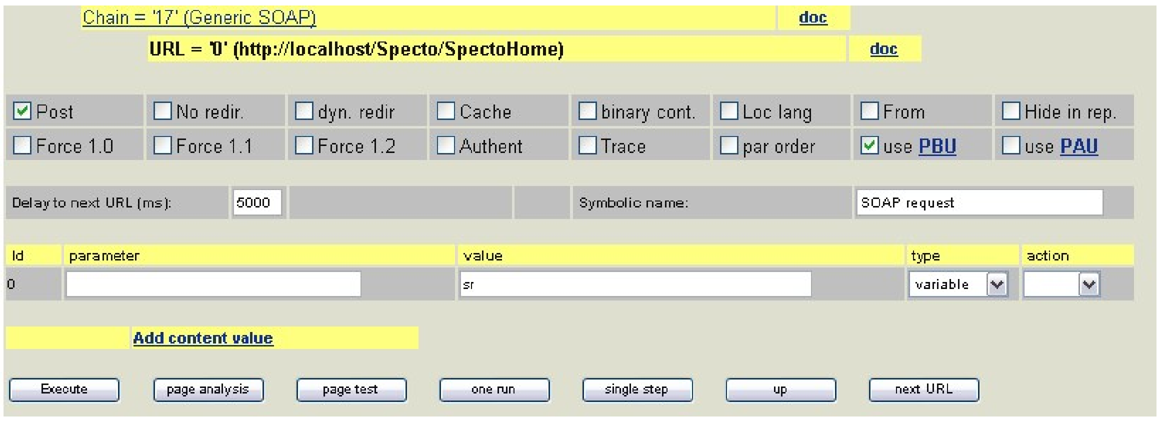
Parameters are specified with their parameter name and an optional Type (= java class) separated by a ':' character. If no parameter type is specified SPECTO auto detects Text, numerical and Boolean parameters.

SPECTO includes a number of predefined attributable documents for the various SOAP formats. See chapter 'business-to-business' for details.

The common SPECTO content check mechanisms are available to parse the output returned from the target.

Example :

```
//script=js

var soap_request =
"<?xml version=\"1.0\" encoding=\"utf-8\"?>" +
"<soap:Envelope xmlns:xsi=\"http://www.w3.org/" +
"2001/XMLSchema-instance\" " +
"xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\"> " +
"   <soap:Body>" +
"      <MetB><Par>1</Par><Par>2</Par></MetB>" +
"   </soap:Body>" +
"</soap:Envelope>";

// save to variables which is later used as an URL parameter
specto.setVariable("sr", soap_request);
```


Chain = '17' (Generic SOAP) doc
URL = '0' (http://localhost/Specto/SpectoHome) doc

✓ Post	No redir.	dyn. redir	Cache	binary cont.	Loc lang	From	Hide in rep.
Force 1.0	Force 1.1	Force 1.2	Authent	Trace	par order	✓ use PBU	use PAU

Delay to next URL (ms): 5000 Symbolic name: SOAP request

Id	parameter	value	type	action
0		sr	variable	

Add content value

In the above example the web service function 'MetB' with parameters '1' and '2' is executed on host 'localhost'.

WebDAV (internet file) access

SPECTO supports WebDAV :

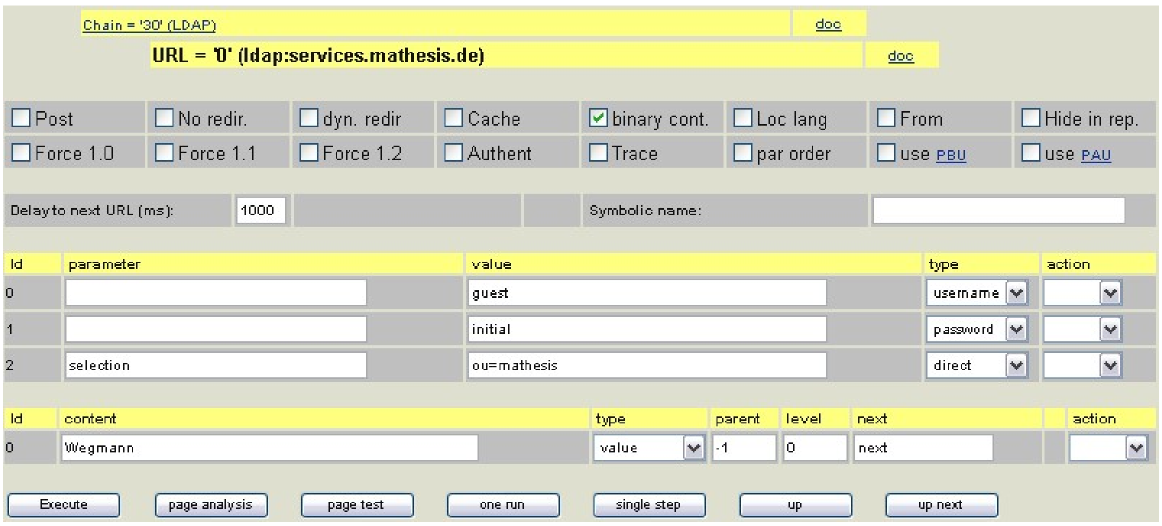
Syntax :
webdav: directory_and_file_URL

Parameters :

Username and password have to be supplied with the standard SPECTO mechanism (types 'username' and 'password').

The generated outputs includes all rows separated by CR/LF and is prefixed with a row of the column names.
The common SPECTO content check mechanisms are available to parse the output returned from the target.

Example :



Chain = '30' (LDAP) doc

URL = '0' (ldap:services.mathesis.de) doc

☐ Post	☐ No redir.	☐ dyn. redir	☐ Cache	☑ binary cont.	☐ Loc lang	☐ From	☐ Hide in rep.
☐ Force 1.0	☐ Force 1.1	☐ Force 1.2	☐ Authent	☐ Trace	☐ par order	☐ use PBU	☐ use PAU

Delay to next URL (ms): 1000 Symbolic name:

Id	parameter	value	type	action
0		guest	username	
1		initial	password	
2	selection	ou=mathesis	direct	

Id	content	type	parent	level	next	action
0	Wegmann	value	-1	0	next	

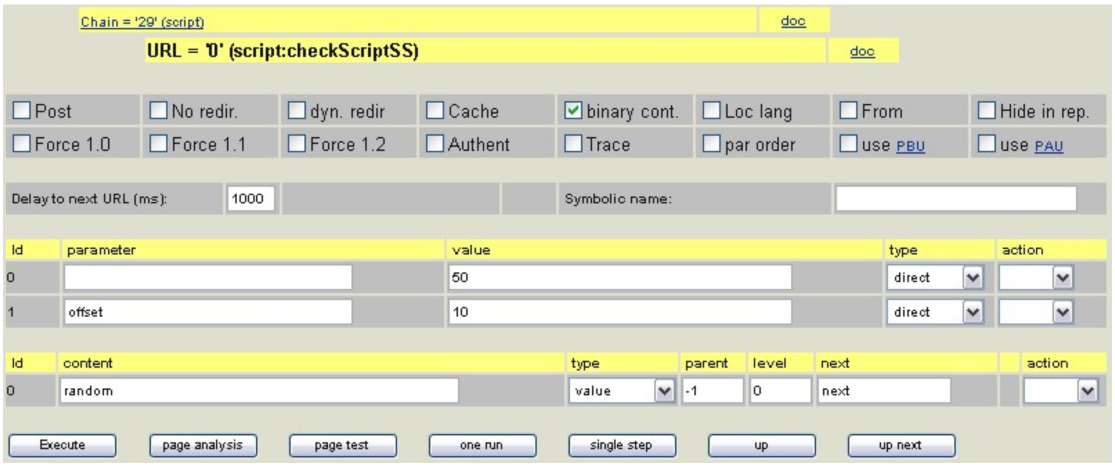
In the above example the web-dav server on host " is queried for file ".

Programmed checks (scripts)

Instead of one of the built in access methods a custom script may be executed.

Syntax :
script: <scriptname>

Example :



Chain = '29' (script) doc

URL = '0' (script:checkScriptSS) doc

☐ Post	☐ No redir.	☐ dyn. redir	☐ Cache	✓ binary cont.	☐ Loc lang	☐ From	☐ Hide in rep.
☐ Force 1.0	☐ Force 1.1	☐ Force 1.2	☐ Authent	☐ Trace	☐ par order	☐ use PBU	☐ use PAU

Delay to next URL (ms): 1000 Symbolic name:

Id	parameter	value	type	action
0		50	direct	
1	offset	10	direct	

Id	content	type	parent	level	next	action
0	random	value	-1	0	next	

Execute page analysis page test one run single step up up next

Script of the above example :

```
// script=ss
// compute a random number
tron;
result = "0";
numargs = args.length;
if numargs ge 1;
   maxr = args 0;
   result = random maxr;
   if numargs ge 2;
      option = args 1;
      index = find$ option "offset=";
      if index ge 0;
         offset = mid$ option 7 9999;
         result = result + offset;
         result = "random: " + random;
      endif;
   endif;
endif;
troff;
return result;
```

The content of the script is available as SPECTO content and can be used for content checks.

Applets

Applets are applications developed in the Java programming language which are executed in the browser. Usually (though not required) applets communicate with the server from which they are loaded.

There must be differentiated between several ways of applet/server communication :

1. The applet is embedded in an HTML page and communicates within the browser by reading HTML parameters. The requests sent to the server have the identical syntax as requests submitted by the browser as the result of an HTML '<form>' construct. The results sent back by the server is another HTML page which will be displayed by the browser, no direct communication to the applet can appear.
2. The applet establishes an direct, bidirectional communication channel to the server. The communication is based on the HTTP protocol. The data transmitted may be HTML encoded or in an proprietary form, but usually ASCII encoded (variant 2a); or generated by serialization of a java class (variant 2b).
3. The applet establishes an direct, bidirectional communication channel to the server. The communication is based on a socket communication (which also may be RMI ('remote method invocation') and therefore is 'low level'. Data formatting is usually proprietary. This variant is unusual because it is normally not possible to be used through proxies/firewalls.

SPECTO assistance : applet emulation

Variant 1 does not need special treatment; it can be handled using standard SPECTO mechanisms. In this case the applet is not loaded/executed by the SPECTO engine.

For variant 2 SPECTO uses special switches ('post' and 'binary') and the 'http' script command for http header fields to program the HTTP communication.
In this case the applet is not loaded/executed by the SPECTO engine.

For variant 3 SPECTO uses a set of commands to program the socket communication.
In this case the applet is not loaded/executed by the SPECTO engine.
The URL must be type-specified as 'RAW' and a socket number has to provided. The supplied parameters are concatenated without '?' and '&' separators and will not be 'URL encoded'.
Example : **`raw://socket-host.com:1204`**

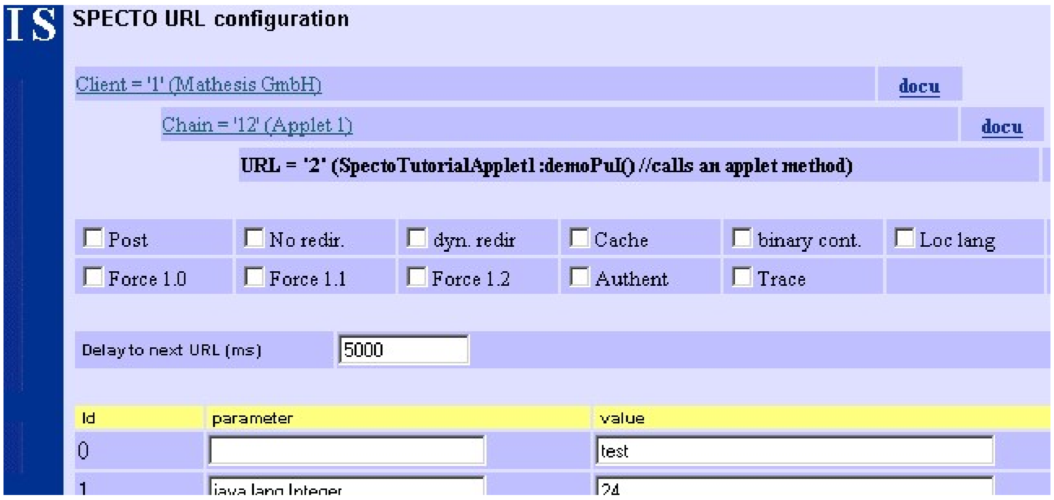
SPECTO assistance : applet automation

As an extension to the mechanisms described above, especially variants 2b and 3, SPECTO provides a mechanism to 'automate' the original applet. Here SPECTO does not directly communicate with the web/application server any more, but calls methods of the applet which then handles the communication with the server. The advantage is that details of the complex applet-servlet communication need not be known by the SPECTO engine.

In this case the applet is loaded and executed by the SPECTO engine and remains active (in a separate thread) during the life cycle of a chain (optionally of the thread). All public methods and public data of the applet are made available to be executed by SPECTO URL definitions.

The syntax for the URL is :
To call a method : **`AppletName:MethodName()`**
To get a field content :**`AppletName:FieldName`**

SPECTO's URL parameter mechanism (field 'value') is used for the method parameters value specification. The 'parameter' field may be used to specify the type of the parameter. If the type is not specified, Java type 'String' is assumed.
All capabilities of SPECTO parameters and content check specification are available.


SPECTO URL configuration

Client = '1' (Mathesis GmbH) docu

Chain = '12' (Applet 1) docu

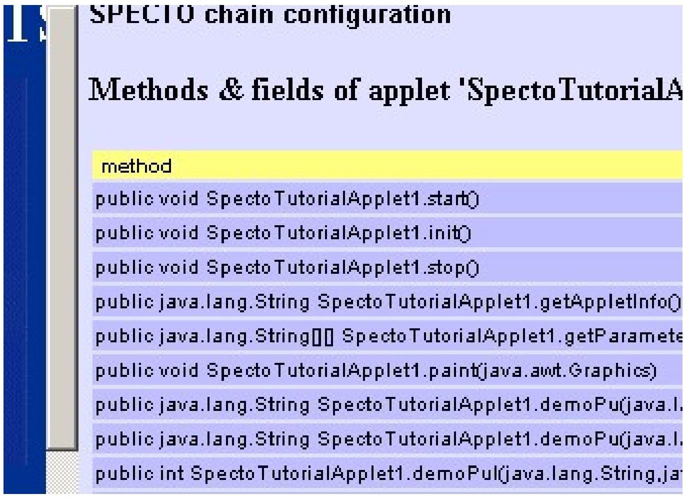
URL = '2' (SpectoTutorialApplet1:demoPuI() //calls an applet method)

☐ Post	☐ No redir.	☐ dyn. redir	☐ Cache	☐ binary cont.	☐ Loc lang
☐ Force 1.0	☐ Force 1.1	☐ Force 1.2	☐ Authent	☐ Trace	

Delay to next URL (ms) 5000

Id	parameter	value
0		test
1	java.lang.Integer	24

To assist in evaluating the methods and data available in an applet the 'paa <AppletName>' command (or the 'analyze' function in the URL specification screen) is provided.


SPECTO chain configuration

Methods & fields of applet 'SpectoTutorialA

method
public void SpectoTutorialApplet1.start()
public void SpectoTutorialApplet1.init()
public void SpectoTutorialApplet1.stop()
public java.lang.String SpectoTutorialApplet1.getAppletInfo()
public java.lang.String[][] SpectoTutorialApplet1.getParamete
public void SpectoTutorialApplet1.paint(java.awt.Graphics)
public java.lang.String SpectoTutorialApplet1.demoPu(java.l.
public java.lang.String SpectoTutorialApplet1.demoPu(java.l.
public int SpectoTutorialApplet1.demoPuI(java.lang.String,ja

As part of the Specto tutorials the applet 'SpectoTutorialApplet1' is provided.

PDF documents

There is no specific support for PDF documents in SPECTO. The availability, load time and content of PDF documents can be evaluated using standard SPECTO mechanisms. The advanced navigation features available in PDF since version 4 cannot be automated by SPECTO.

Flash

There is no specific support for Flash applications in SPECTO. The availability, load time and content of Flash scripts can be evaluated using standard SPECTO mechanisms.

Scalable Vector Graphics (SVG)

Pages coded using Scalable Vector Graphics are XML documents and may be processed with SPECTO standard methods for XML documents.

The SPECTO engine has a SVG generator which is accessible via the script languages.

Advanced Topics

Proxy coverage

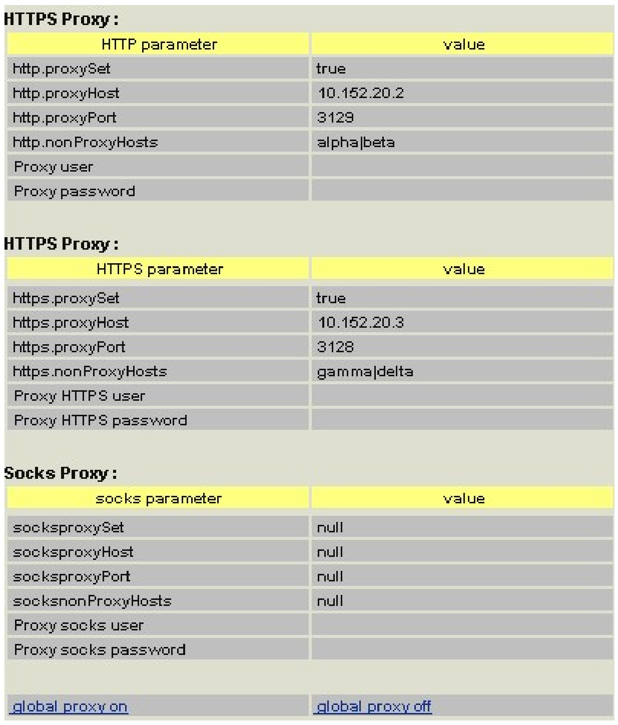
Overview

Usually intranets are separated from the internet by 'firewalls'. 'firewalls' allow certain protocols (e.g. http and smtp) to tunnel through themselves by the use of 'proxy servers'. Also 'proxy servers' are used by some internet providers for load balancing, caching and filtering.
Beside the usual 'http' proxies there are 'socks' proxies which allow for highly secure tunneling of applications through the firewall. So currently not much in use, 'socks' proxies may spread when electronic commerce applications gain wider usage.
A proxy may require a client to authenticate itself using a userid/password scheme.

SPECTO fully supports proxy based infrastructure.

Commands :

It is recommended to use command 'wp' (see screen shot below) for a graphical proxy maintenance.



HTTPS Proxy :

HTTP parameter	value
http.proxySet	true
http.proxyHost	10.152.20.2
http.proxyPort	3129
http.nonProxyHosts	alpha|beta
Proxy user	
Proxy password	

HTTPS Proxy :

HTTPS parameter	value
https.proxySet	true
https.proxyHost	10.152.20.3
https.proxyPort	3128
https.nonProxyHosts	gamma|delta
Proxy HTTPS user	
Proxy HTTPS password	

Socks Proxy :

socks parameter	value
socksproxySet	null
socksproxyHost	null
socksproxyPort	null
socksnonProxyHosts	null
Proxy socks user	
Proxy socks password	

Other commands :

'wp 0'	turn proxy assistance off.
'wp 1'	enable global proxy for http, https and socks.

Configuration

It is recommended to use SPECTO customizing (menu 'customizing') for proxy configuration.
Proxy configuration is via SPECTO attributes 'ProxyHost', 'ProxyPort', 'ProxySocksHost', 'ProxySocksPort', ProxyUserId' and 'ProxyPassword'. The default proxy usage (as maintained using the ***'wp'*** command) is configured in attribute 'ProxyDefault'; if 'ProxyDefault' is not set, '0' (no proxy usage) is assumed.
If the proxy configuration is changed during SPECTO operation; applying the ***'wp'*** without parameters command will read the new configuration.

Example:

'aw Proxyhost proxy.nls.de', *'aw ProxyPort 8080'* and '*aw ProxyDefault 1*' prepare SPECTO to direct all URLs to the proxy server 'proxy.nls.de', port 8080.

Notes :

- Proxy configuration is common for a SPECTO instance. If different URLs have to be accessed with and without using a proxy or using different proxies, then those proxies have to specified dynamically in the 'PBC' or 'PBU' event of the affected chain or URL using the 'Set' command (see also chapter '*Advanced settings ('set' command)*' for a list of proxy related properties and chapter 'Dynamic proxy setting' below).

Dynamic proxy setting

Example (Javascript) :

```
// script=js;
specto.setProperty("net.http.proxyHost", "localhost");
specto.setProperty("net.http.proxyPort", "82");
specto.setProperty("net.https.proxyHost", "localhost");
specto.setProperty("net.https.proxyPort", "447");
```

Secure communication

SPECTO supports HTTP-S and SSL secure transfer and authentication by certificates.

HTTP-S

Secure socket layer is fully supported in SPECTO. An URL specified with the 'https:' protocol or typed as 'HTML/HTTPS' in the type field will be accessed using the HTTP-S protocol.

Secure socket layer ('ssl')

Secure socket layer is fully supported in SPECTO. An URL specified with the 'https:' protocol or typed as 'HTML/HTTPS' in the type field will automatically accessed using ssl encryption.

Authentication by certificates

SPECTO supports authentication by certificates. In the default configuration all certificates will be accepted (acknowledged positive), an user exit allows for filtered acknowledges. If the 'keytool' program is available for SPECTO, all received certificates will be stored in the key store.
The key store is located in file 'specto' in the root of the file system; its username and password is 'specto'.

Location of the certificates keystore

The JAVA keystore is *usually* implemented in file `cacerts` located in the `security` folder of folder `lib` in the java home directory. The java home directory is available via property `JAVA_HOME` and can be determined by the SPECTO '`pr`' command.

Processing the certificates keystore

All processing is performed by the command line utility `keytool` located in the JAVA `bin` directory (usually not in the computer's search path).

Official documentation of `keytool`:
`http://docs.oracle.com/javase/7/docs/technotes/tools/windows/keytool.html`

Most common parameters:

Parameter	Description
A command like	`-list`, `-genkeypair` or `-importcert`
-keystore	specifies the keystore, for SPECTO `cacerts` in the JRE's

	lib/security folder should be used (JAVA often also uses the file .keystore in the user's home directory).
-storepass	Password of the keystore. The default password is 'changeit', SPECTO does not change this password nor requires a different password, it id howevere recommended to use a secure password.

keytool -list -keystore ..\lib\security\cacerts

Examples (assuming a windows environment, with the command shell in the JAVA HOME directory):

Command	Explanation
`bin\keytool -list -storepass changeit -keystore lib\security\cacerts`	List all certificates in the store. Option -v would yield a more detailed list.
`bin\keytool -printcert -file lib\security\<filename>.cer`	Print the content of a certificate.

JAVA-Configuration of SSL and certificates

Properties:
`javax.net.ssl.keyStore` : Location of the Java keystore file containing an application process's own certificate and private key. On Windows, the specified pathname must use forward slashes, /, in place of backslashes.

`javax.net.ssl.keyStorePassword`: Password to access the private key from the keystore file specified by `javax.net.ssl.keyStore`. This password is used twice: To unlock the keystore file (store password), and to decrypt the private key stored in the keystore (key password).

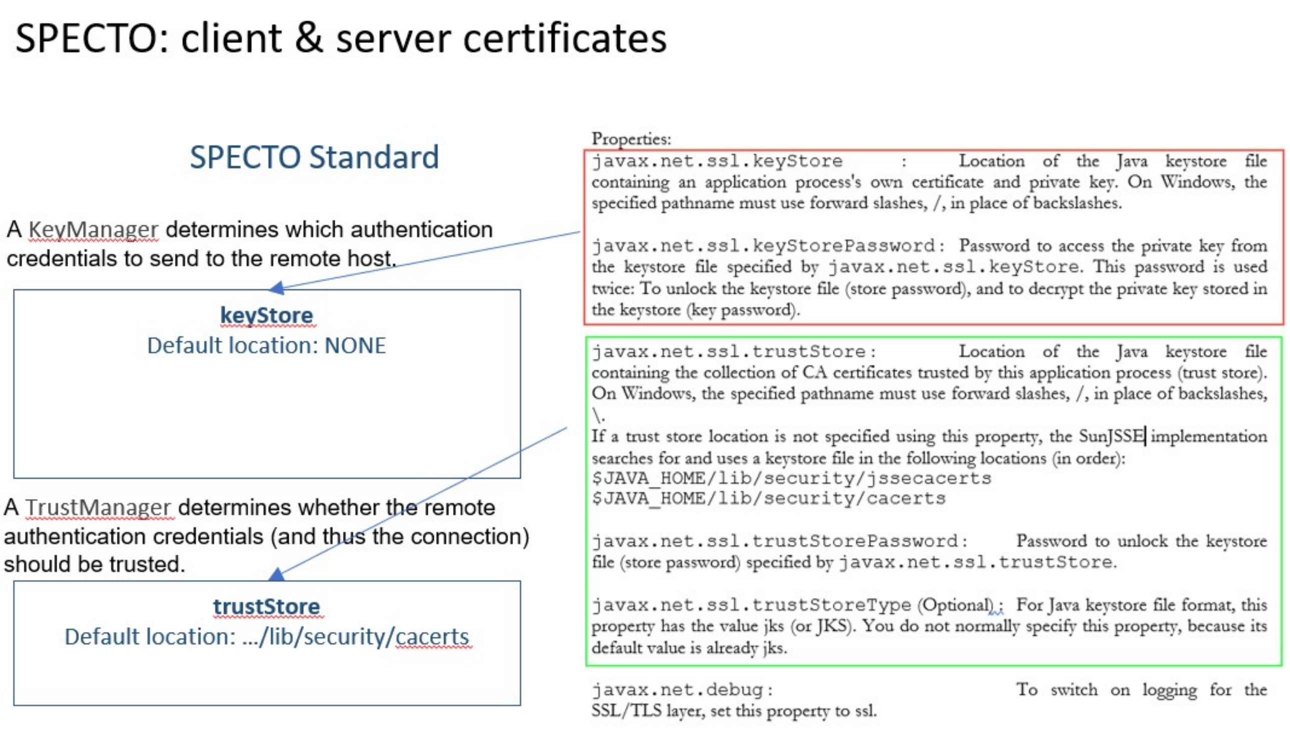
`javax.net.ssl.trustStore`: Location of the Java keystore file containing the collection of CA certificates trusted by this application process (trust store). On Windows, the specified pathname must use forward slashes, /, in place of backslashes, \.
If a trust store location is not specified using this property, the SunJSSE implementation searches for and uses a keystore file in the following locations (in order):
`$JAVA_HOME/lib/security/jssecacerts`
`$JAVA_HOME/lib/security/cacerts`

`javax.net.ssl.trustStorePassword`: Password to unlock the keystore file (store password) specified by `javax.net.ssl.trustStore`.

`javax.net.ssl.trustStoreType` (Optional) : For Java keystore file format, this property has the value jks (or JKS). You do not normally specify this property, because its default value is already jks.

`javax.net.debug`: To switch on logging for the SSL/TLS layer, set this property to ssl.

Link to official documentation:
http://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html#Customization



SPECTO: client & server certificates

SPECTO Standard

A KeyManager determines which authentication credentials to send to the remote host.

keyStore
Default location: NONE

A TrustManager determines whether the remote authentication credentials (and thus the connection) should be trusted.

trustStore
Default location: .../lib/security/cacerts

Properties:

`javax.net.ssl.keyStore` : Location of the Java keystore file containing an application process's own certificate and private key. On Windows, the specified pathname must use forward slashes, /, in place of backslashes.

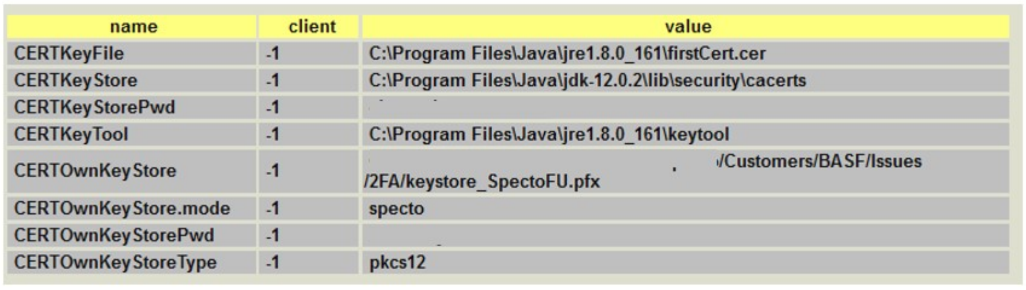
`javax.net.ssl.keyStorePassword`: Password to access the private key from the keystore file specified by `javax.net.ssl.keyStore`. This password is used twice: To unlock the keystore file (store password), and to decrypt the private key stored in the keystore (key password).

`javax.net.ssl.trustStore`: Location of the Java keystore file containing the collection of CA certificates trusted by this application process (trust store). On Windows, the specified pathname must use forward slashes, /, in place of backslashes, \.
If a trust store location is not specified using this property, the SunJSSE implementation searches for and uses a keystore file in the following locations (in order):
`$JAVA_HOME/lib/security/jssecacerts`
`$JAVA_HOME/lib/security/cacerts`

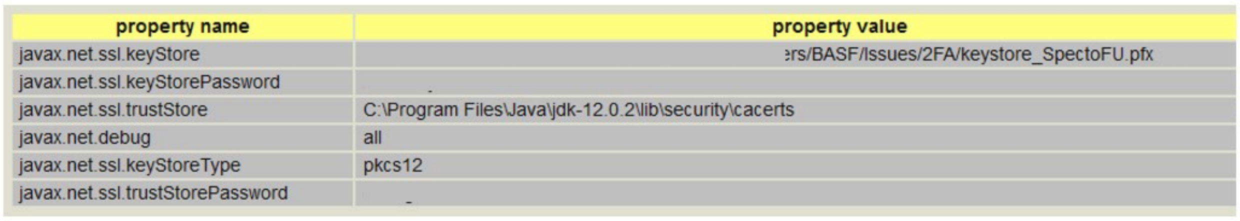
`javax.net.ssl.trustStorePassword`: Password to unlock the keystore file (store password) specified by `javax.net.ssl.trustStore`.

`javax.net.ssl.trustStoreType` (Optional) : For Java keystore file format, this property has the value jks (or JKS). You do not normally specify this property, because its default value is already jks.

`javax.net.debug`: To switch on logging for the SSL/TLS layer, set this property to ssl.



name	client	value
CERTKeyFile	-1	C:\Program Files\Java\jre1.8.0_161\firstCert.cer
CERTKeyStore	-1	C:\Program Files\Java\jdk-12.0.2\lib\security\cacerts
CERTKeyStorePwd	-1	[redacted]
CERTKeyTool	-1	C:\Program Files\Java\jre1.8.0_161\keytool
CERTOwnKeyStore	-1	[redacted] /Customers/BASF/Issues /2FA/keystore_SpectoFU.pfx
CERTOwnKeyStore.mode	-1	specto
CERTOwnKeyStorePwd	-1	[redacted]
CERTOwnKeyStoreType	-1	pkcs12



property name	property value
javax.net.ssl.keyStore	[redacted] ers/BASF/Issues/2FA/keystore_SpectoFU.pfx
javax.net.ssl.keyStorePassword	[redacted]
javax.net.ssl.trustStore	C:\Program Files\Java\jdk-12.0.2\lib\security\cacerts
javax.net.debug	all
javax.net.ssl.keyStoreType	pkcs12
javax.net.ssl.trustStorePassword	[redacted]

Multiple network interfaces

SPECTO supports direction of network traffic through different network interfaces on the host system. This is accomplished by using the ‘route’ capabilities of the host system.

Configuration

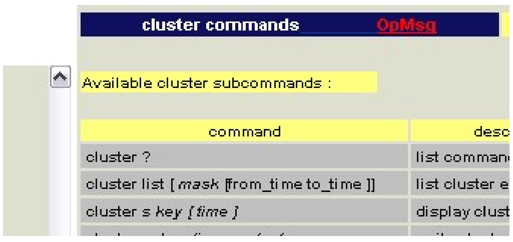
The SPECTO ‘`ni`’ command shows the available network interfaces of the host system.
The SPECTO ‘`route`’ command shows and maintains the current routing configuration.

‘Clustered’ storage

SPECTO supports permanent storage of any data (especially of any size and structure) under a key and an optional time key in the SPECTO cluster.
The clusters are primarily used via the SPECTO scripting languages. E.G. a PAU script may write the complete HTML of a monitored site to the cluster for later inspection.

Configuration and maintenance

The SPECTO ‘`cluster`’ is used for cluster management.

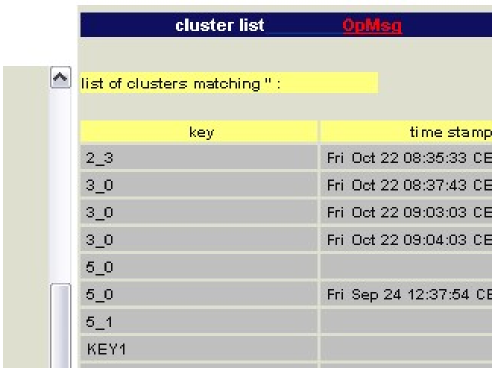


cluster commands OpMsg

Available cluster subcommands :

command	desc
cluster ?	list comman
cluster list [*mask* [from_time to_time]]	list cluster e
cluster s *key [time]*	display clust

The available clusters are listed using the ‘`cluster list`’ command :



cluster list OpMsg

list of clusters matching '' :

key	time stamp
2_3	Fri Oct 22 08:35:33 CE
3_0	Fri Oct 22 08:37:43 CE
3_0	Fri Oct 22 09:03:03 CE
3_0	Fri Oct 22 09:04:03 CE
5_0	
5_0	Fri Sep 24 12:37:54 CE
5_1	
KEY1	

Script

The clustered storage is available for SPECTO scripts through the methods listed below :

```
Object[] dirCluster (String key, long timeFrom, long timeTo)
String readCluster (String key, long time)
String readCluster (String key)
int writeCluster (String key, long time, String data)
int writeCluster (String key, String data)
int writeClusterAtNow (String key, String data)
int deleteCluster (String key, long time)
int deleteCluster (String key)
```

Sample code

```
// script=js
specto.writeClusterAtNow(“demo”, “test”);
```

SAP R/3 URLs

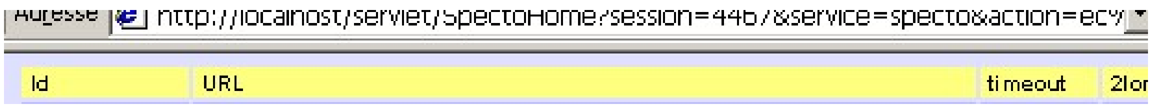
Instead of an web server it is possible to monitor a SAP R/3 system. Most RFC capable function module can be called. Simple import and export parameters are supported as parameters; internal tables are not supported.

The syntax for the URL is
with server logon:
```
r3:<hostname>:<systemnr>:<userid>:<password>:<client>:<language>/<function>
```

with group logon:
```
r3:<hostname>:<systemnr>:<userid>:<password>:<client>:<language>:<message
server>:<system name>:<logon group>/<function>
```

Userid and password can also be transmitted using the SPECTO parameters of type 'username' and 'password'; in this case the corresponding above fields may be left empty.

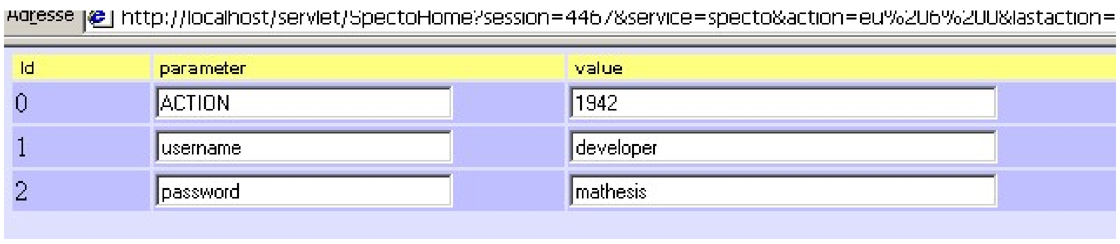
Example :


Adresse http://localhost/servlet/SpectoHome?session=4467&service=specto&action=ec

Id	URL	timeout	2lor

The import parameters of the function are defined as SPECTO parameters; the export parameters of the function can be checked using SPECTO's content mechanism. The form of then content field is '[type:]parameter_name;value' (e.g. 'INT:RESULT;00003884'). 'type' may be any of BCD, BYTE, CHAR, DATE, FLOAT, INT, ITAB, NUM, TIME, WCHAR and WSTRING; the default is CHAR. For the parameters the complete SPECTO functionality (variables and external computed values) is available.

Example :


Adresse http://localhost/servlet/SpectoHome?session=4467&service=specto&action=ed%206%200&lastaction=

Id	parameter	value
0	ACTION	1942
1	username	developer
2	password	mathesis

The example shown in the example above calls the (implemented for this) function 'ZZZSPECTO_PING' on system 192.168.73.230, system number 17 in client 000 after logging on with user 'developer' and password 'nls' using English as dialogue language.

The return parameter 'RESULT' is checked against the value '00003884'.

The primary intent of the SPECTO R/3 functionality is not to monitor R/3 implementations, but to check if, as a result of a chain executed on an electronic commerce implementation, an appropriate result has appeared in the attached R/3 system.

Message services

In complex application environments components are often coupled asynchronously by using message queues. SPECTO can access and generate messages in any message queue system with a JMS front-end. There is additional support for the following message queue service implementations :

- PROGRESS software 'sonicMQ'
- IBM 'MQseries'

SPECTO supports 'point-to-point' and 'publish and subscribe' models. In receiving messages, 'reply mode' is possible.

Format

The syntax for the URL is :
```
jms[(<implementation>)]-<model>://<hostname>/<queuename>:          <message-
mask>:<accesstype>[:<username>[:<password>]]
```

with 'implementation' currently being 'sonic' or 'mqseries'; 'model' being 'p2p' or 'pas'; access type' being 'browse' (non changing); 'retrieve' (removing the message from the queue) or 'set' (writing a message into the queue).
Username ans password may also be supplied as parameters (parameter types 'username' and 'password').

Reading / retrieving messages

Messages are returned as one block of XML formatted text with the following title lines added at the beginning by SPECTO :

```
<from> sender identification </from>
<message-id> message id </message-id>
<sent> timestamp </sent>
<jms-message>
```

and the following line added at the end :

```
</jms-message>
```

If no message is available, an empty result is returned. If multiple messages are matching the message mask, the message with the highest priority and then the earliest date returned.

Note: applications should parse for the '`<jms-message>`' tag since it is likely that more title tags will be introduced in future releases.

Sending messages

Messages are sent using SPECTO parameter mechanism. The following parameters, according to the JMS specification are available :

`'type'`	message type (eg. 'PERSISTENT')

`'priority'`	priority (0-9, 4 as default)
`'from'`	sender identification
`'id'`	message identification
`'expiration'`	message expiration (in milliseconds, 0='forever')
`'content'`	message content

Any other parameters are set as JMS properties of type text.

Extensions for PROGRESS software 'sonicMQ'

- Hierarchical name spaces are supported
- Distributed transactions are supported.). This is available with SPECTO rel. 1.62.
- The management api and the special 'dead message' queue (DMQ) is supported.

Management API

By setting the attribute 'JMSSonicMEnable' to 'true' a separate listener is started which collects any event entries from the sonic system and the sonic DMQ and makes it available in the variables '_JMSSonicEvents' and '_JMSSonicDMQ'.

Extensions for IBM 'MQseries'

Non-JMS queue types can be specified (queue names according to mq-series specification). This is available with SPECTO rel. 1.62.

Examples

jms(mqseries)-p2p://central/order-queue:order4711:browse

Email fetch

In b2b environments components are often not coupled directly to the ERP system but create and send emails with the reulst of their processing (e.g. an order entered by a customer using the web application is sent as an email to the order entry department). SPECTO can access email mailboxes and check for or retrieve mails matching certain selection criteria.
In addition SPECTO can generate mails.

Retrieving emails

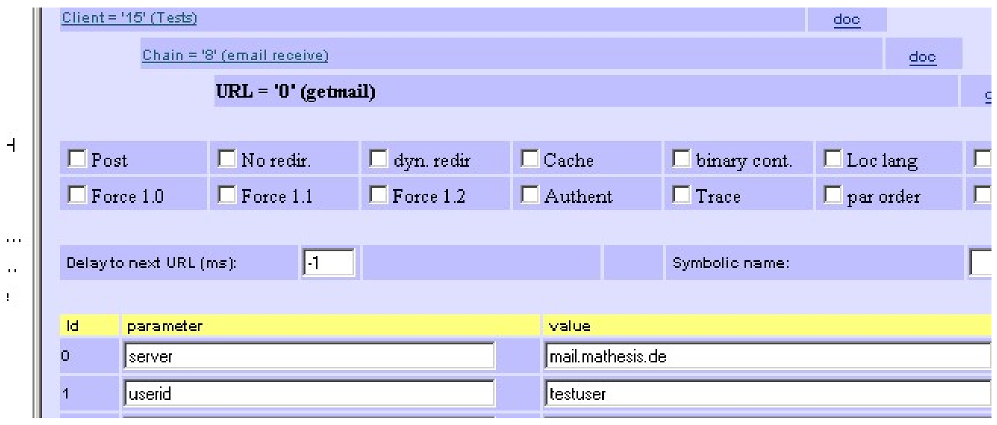
The syntax of the URL for receiving of emails is :
`smtp:read://<mailhost>/<subject>:<accesstype>[:<username>[:<password>]]`
(as an alternative instead of an URL the keyword 'getmail' can be used.)

With 'accesstype' being 'browse' or 'retrieve'. Username and password may also be supplied as parameters (parameter types 'username' and 'password').

The following parameter types are supported :

`'server'`	the email server name
`'title'`	the prefix of the subject to be searched for

Example :



Client = '15' (Tests) doc

Chain = '8' (email receive) doc

URL = '0' (getmail)

☐ Post ☐ No redir. ☐ dyn. redir ☐ Cache ☐ binary cont. ☐ Loc lang
☐ Force 1.0 ☐ Force 1.1 ☐ Force 1.2 ☐ Authent ☐ Trace ☐ par order

Delay to next URL (ms): -1 Symbolic name:

Id	parameter	value
0	server	mail.mathesis.de
1	userid	testuser

Sending emails

Sending of emails is equivalent to the retrieving of emails. The syntax of the URL for sending of emails is :
`smtp://<mailhost>/<subject>:<accesstype>[:<username>[:<password>]]`

The following additional parameter types are supported :

`'receivers'`	a list of email addresses, separated by a ';'
`'ccreceivers'`	a list of email addresses, separated by a ';', to be sent as 'cc:'
`'content'`	the body of the message

Note: SPECTO can access email mailboxes using the POP3 or IMAP protocols, emails are sent using the SMTP protocol. In order for receiving and sending of emails to work, the corresponding attributes must be configured accordingly (see 'Notifications' and 'SPECTONet').

Ramp mode

For load measurements it may be required to start a chain processing smoothly to allow the measured system to 'warm up'. SPECTO assists this using 'ramp mode'. 'ramp mode' is activated for a chain by specifying a negative period for the chain repetition time.
If 'ramp mode' is activated the interval at this the chain is repeated starts with ten times the (positive) value defined in the period parameter and is decreased to 75% of its value after each chain processing until it reaches the defined period.

An example : If the chain's period is defined as 10 seconds and ramp mode is activated (ergo –10 is entered in the period field) there will be a 100 second wait time between the first and the second iteration of the chain, followed by a 75 second wait time, then 56 seconds, 42 seconds, 32 seconds, 24 seconds, 18 seconds, 13 seconds and finally 10 seconds.
The 75% divider can be modified by setting the 'RampIndex' Attribute. The divider is defined as RampIndex/(RampIndex+1); the default for 'RampIndex' is '3'. (75% = 3 / (3+1)).

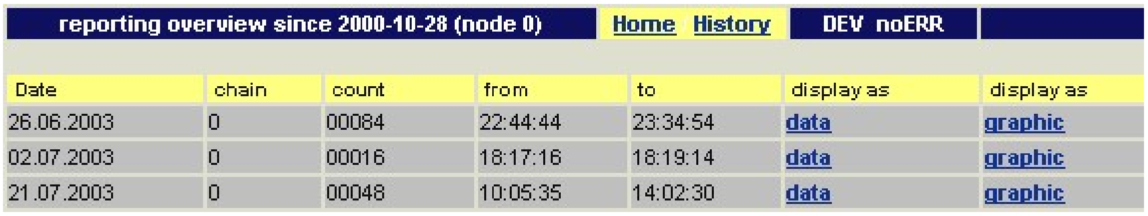
Reporting

Built in reporting

SPECTO supports a textual and a graphical display of the results of its operation. (see chapter 'commands' for a detailed description of the commands parameters).

Navigating within reporting data :

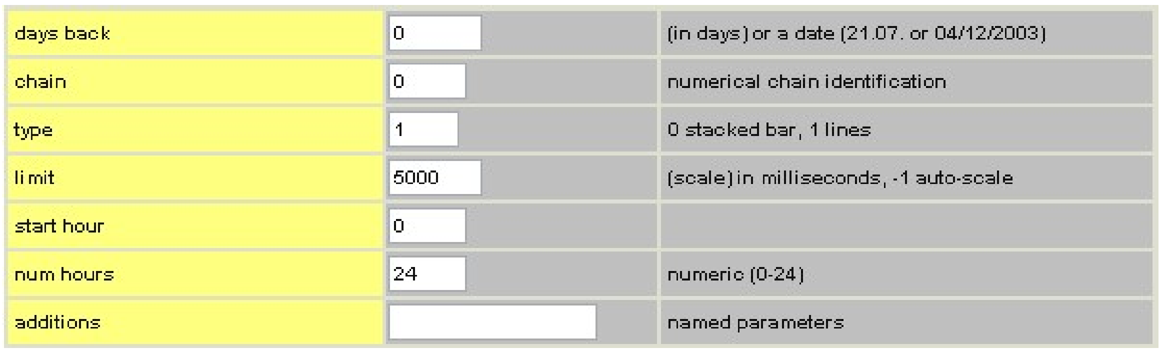
The reporting overview ('**`ro <days-back> [ <chained> ]`**') command provides an entry point into SPECTO reporting. Following the 'data' and 'graphic' links display the corresponding data accordingly using standard settings.



reporting overview since 2000-10-28 (node 0) Home History DEV noERR

Date	chain	count	from	to	display as	display as
26.06.2003	0	00084	22:44:44	23:34:54	data	graphic
02.07.2003	0	00016	18:17:16	18:19:14	data	graphic
21.07.2003	0	00048	10:05:35	14:02:30	data	graphic

For having more control about the selection and display of the data it is recommended to use the navigation menu, entries 'results – measured data – xxx dynamic' and specifying the query as shown below :



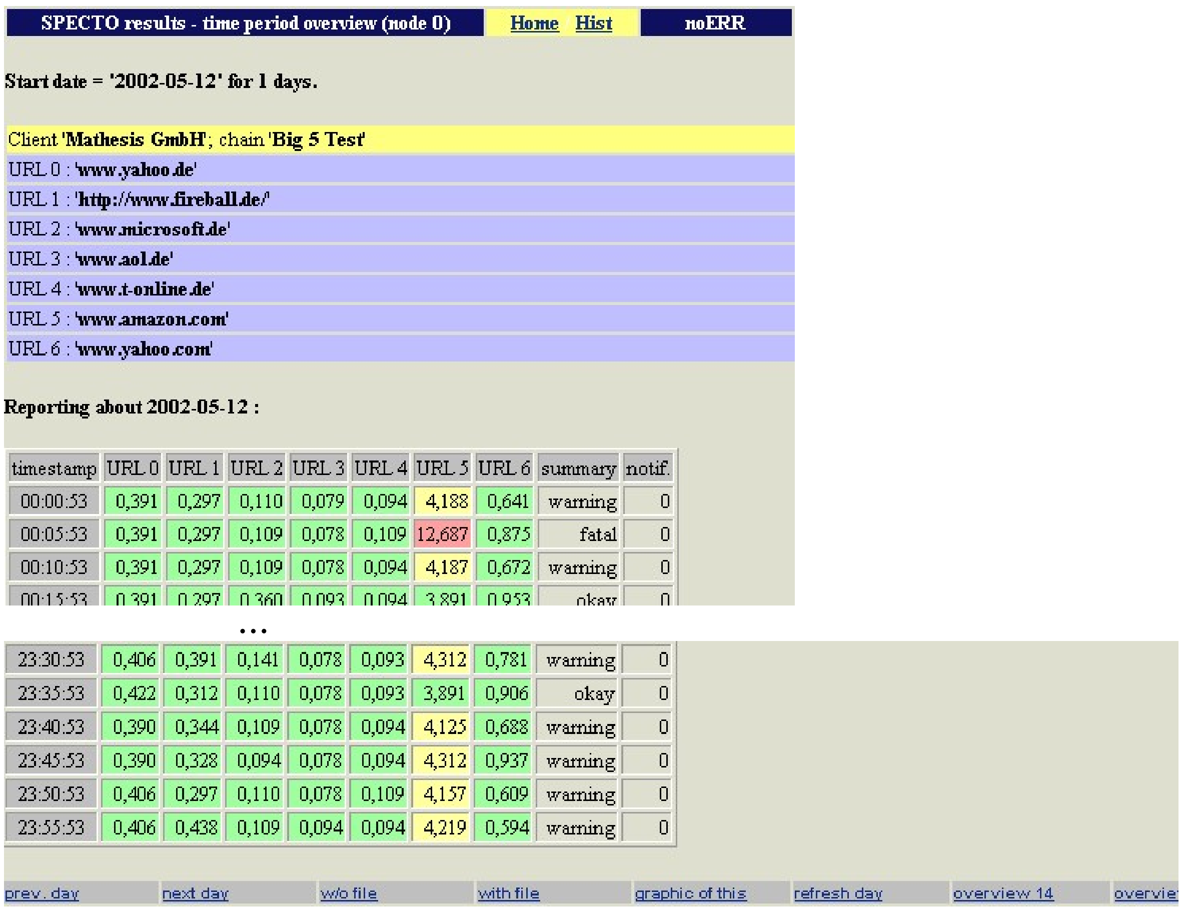
days back	0	(in days) or a date (21.07. or 04/12/2003)
chain	0	numerical chain identification
type	1	0 stacked bar, 1 lines
limit	5000	(scale) in milliseconds, -1 auto-scale
start hour	0	
num hours	24	numeric (0-24)
additions		named parameters

It is possible to enter a date value ('12.05.', '13.05.2002', '04/12/2002', …) instead of the 'days back' count. If type div 10 is not zero, then a download link will be displayed.

The exact parameters of the underlying commands ('**rg**' and '**rh**') are described in chapter 'commands.

Textual reporting :

Example with one measurement per cell :



SPECTO results - time period overview (node 0) | Home Hist | noERR

Start date = '2002-05-12' for 1 days.

Client 'Mathesis GmbH'; chain 'Big 5 Test'
URL 0 : 'www.yahoo.de'
URL 1 : 'http://www.fireball.de/'
URL 2 : 'www.microsoft.de'
URL 3 : 'www.aol.de'
URL 4 : 'www.t-online.de'
URL 5 : 'www.amazon.com'
URL 6 : 'www.yahoo.com'

Reporting about 2002-05-12 :

timestamp	URL 0	URL 1	URL 2	URL 3	URL 4	URL 5	URL 6	summary	notif.
00:00:53	0,391	0,297	0,110	0,079	0,094	4,188	0,641	warning	0
00:05:53	0,391	0,297	0,109	0,078	0,109	12,687	0,875	fatal	0
00:10:53	0,391	0,297	0,109	0,078	0,094	4,187	0,672	warning	0
00:15:53	0,391	0,297	0,360	0,093	0,094	3,891	0,953	okay	0
…									
23:30:53	0,406	0,391	0,141	0,078	0,093	4,312	0,781	warning	0
23:35:53	0,422	0,312	0,110	0,078	0,093	3,891	0,906	okay	0
23:40:53	0,390	0,344	0,109	0,078	0,094	4,125	0,688	warning	0
23:45:53	0,390	0,328	0,094	0,078	0,094	4,312	0,937	warning	0
23:50:53	0,406	0,297	0,110	0,078	0,109	4,157	0,609	warning	0
23:55:53	0,406	0,438	0,109	0,094	0,094	4,219	0,594	warning	0

prev. day | next day | w/o file | with file | graphic of this | refresh day | overview 14 | overvie

(This output was generated using a "**`rh '12.5.' 1 10 0 23`**" command.)

The links ('prev. day', 'next day', …) may be used to navigate within the results without to return to the reporting screen overview.

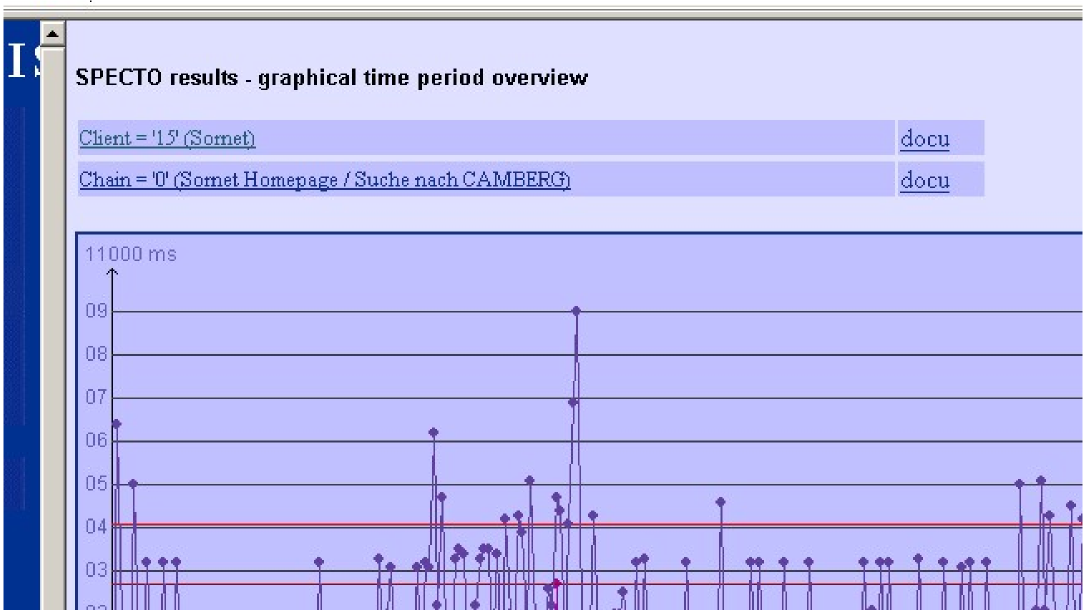
Textual reporting may be enhanced using the following 'named' parameters :

name	description	example
dispref	Whether to display measured reference values. Allowed values are 'true' and 'false'; the default is 'true'	dispref=false
colmin	Additional column which displays the minimum result of the line. Allowed values are 'true' and 'false'; the default is 'false'.	colmin=true
colmax	Additional column which displays the maximum result of the line. Allowed values are 'true' and 'false'; the default is 'false'.	colmax=true
colsum	Additional column which displays the sum of all result of the line. Allowed values are 'true' and 'false'; the default is 'false'.	colsum=true
colavg	Additional column which displays the average result of the line. Allowed values are 'true' and 'false'; the default is 'false'	colavg=true

Example : **`rh 0 0 10 dispref=false colavg=true`**

Graphical reporting :

Overview of a day. It is possible to navigate within the data area in daily steps or switch to the data display using the buttons below the graphic.

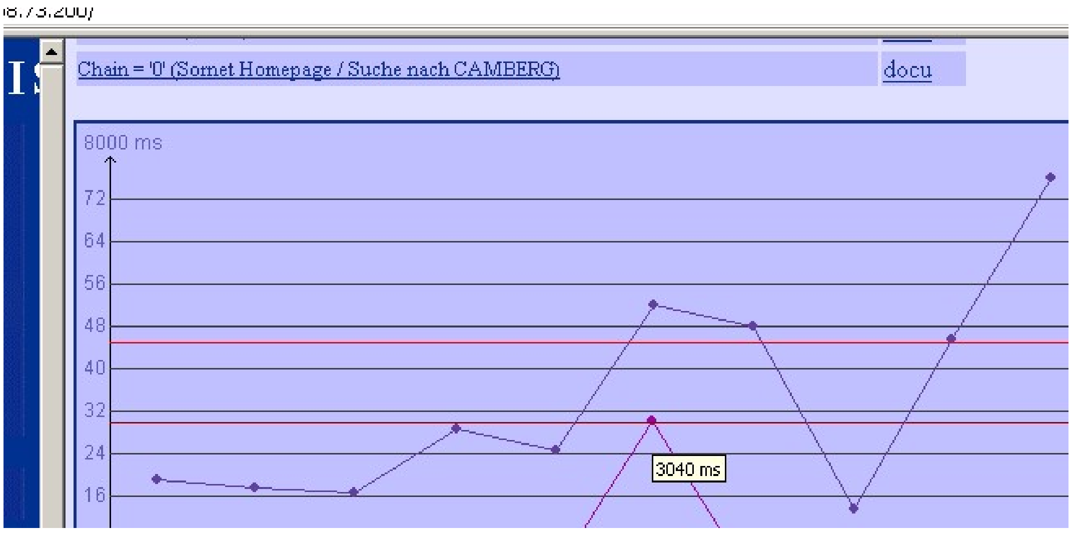

SPECTO results - graphical time period overview

Client = '15' (Sornet) docu

Chain = '0' (Sornet Homepage / Suche nach CAMBERG) docu

A click in into the line area zooms into the selected hour (see picture below), a click into the horizontal scale area zooms in the corresponding 'five-hour' area.

Detailed view (on hour base) :


Chain = '0' (Sornet Homepage / Suche nach CAMBERG) docu

Pointing on an measure mark displays the corresponding value. Clicking the 'refresh whole' button returns to the display of the whole day.

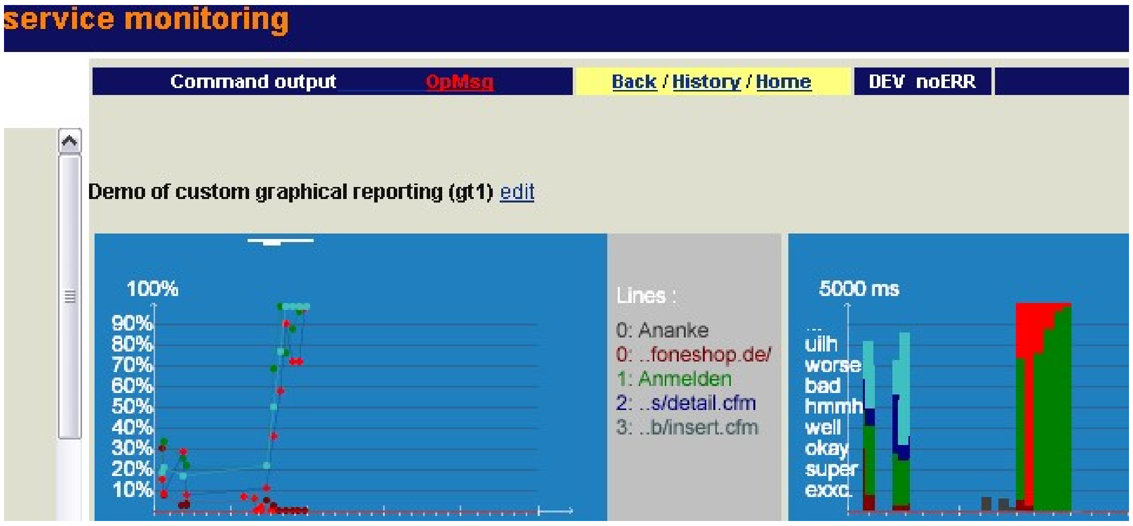
Active notifications are displayed as a horizontal bar on the top end of the image; the thickness of the bar indicates the number of active notifications (not visible in the examples above).

Script based reporting :

The reporting engine is also available for SPECTO scripts. This allows for highly customized graphical reporting.

Variable elements :

- Number of graphics
- Height and width of the graphics
- X and y axis legends
- Combination of URLs of different chains
- Computed (vs. measured data)



The configuration is done within a script by specifying the parameters of a structure (type 'ReportConfigEntry', combining multiple such structures to an arryay, one or multiple calls to the 'specto.reportGra()' method.
Sample scripts are available within the SPECTO on-line support system.

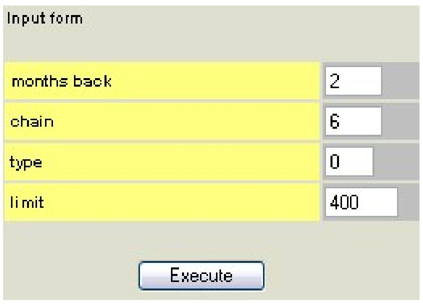
Extracts of sample:

```
…
var gras   = new Array(2);
gra = specto.getReportConfigEntry();
gra.chainId  = chainId;
gra.daysBack = 7;
gra.sizeX    = 395;
gra.sizeXLegend = 100;
gra.sizeY    = 200;
gra.yMax = "100%";
gra.colorBkGnd = 0x7f7f7f;
gra.colorBkGndLegend = 0x1f7fbf;
gra.colorLines = new Array(0x3f3f3f, 0);
gra.arrowSize  = 2;
gra.markerRadius = 2;
gra.chartBorder = 40;
gra.footer   = "Block1";
gras[0] = gra;
…
page += specto.reportGra(-1, 1, 5000, 10, 4, gras, false);
…
```

Summary reports :

Overview of a month/week :

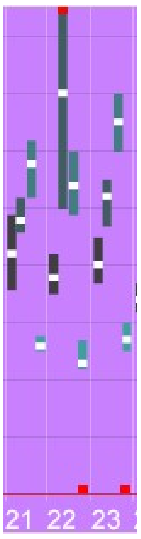
A month/week-wise overview of measured results can be generated for any chain using the 'rm' / 'rw' commands (rm <months_back> <chain> [[<type>] limit], rw <weeks_back> <chain> [[<type>] limit) or the 'monthly overview' / 'weekly overview' entries in the 'results – measured data' section of the menu. Using the menu, the command parameters have to be specified in the dialog shown on the right. The selected month is specified in relation to the current month. Currently only type 0 is implemented.


Input form

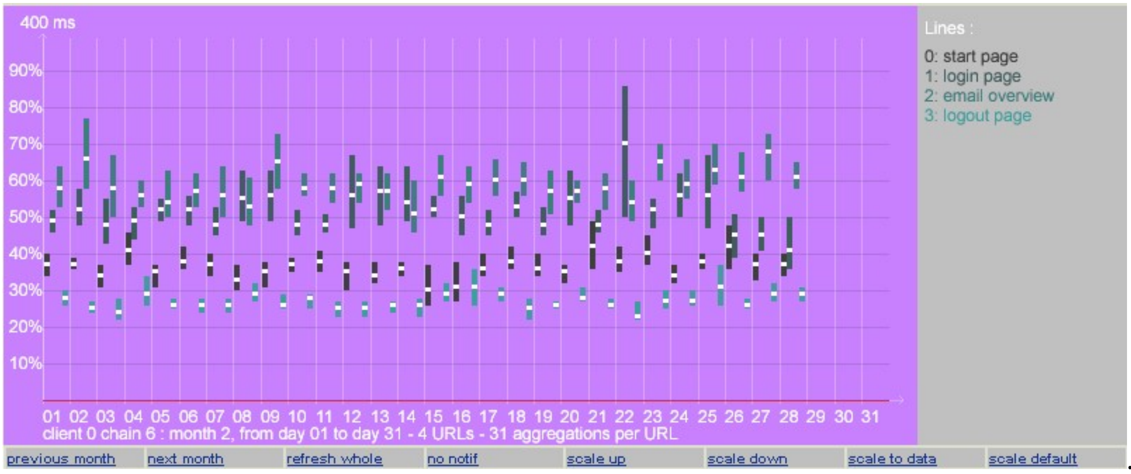
months back	2
chain	6
type	0
limit	400

The resulting display consists of a bar for every URL on every day of the month indicating the range of measured delay times, and a white spot showing the average delay time.
Data exceeding the selected scale will be marked with a red indicator on the top; and measurements with errors are marked with a red indicator on the bottom of the bar.

The display may be scaled and exported (in SVG format) using a right mouse button click in the display area.



Example (month based overview) :

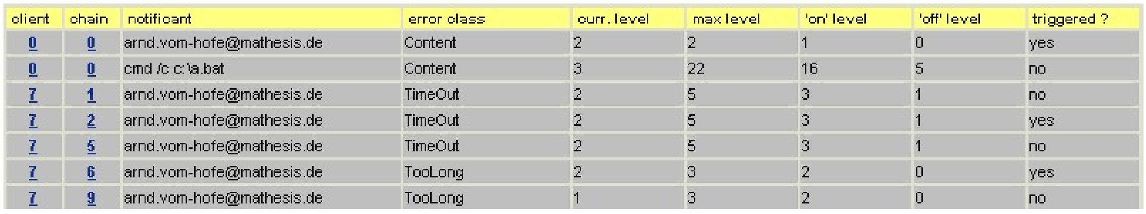


With the links below the graphic it is possible to navigate on a month base and to scale the displayed data.

Reporting of notifications :

Status counters

The status of the notification counters is displayed using menu entry 'Results' – 'Notifications' – 'status counters' or by using command 'no s' :

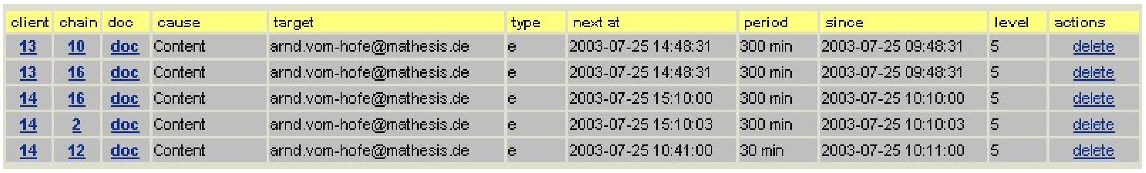


client	chain	notificant	error class	curr. level	max level	'on' level	'off' level	triggered ?
0	0	arnd.vom-hofe@mathesis.de	Content	2	2	1	0	yes
0	0	cmd /c c:\a.bat	Content	3	22	16	5	no
7	1	arnd.vom-hofe@mathesis.de	TimeOut	2	5	3	1	no
7	2	arnd.vom-hofe@mathesis.de	TimeOut	2	5	3	1	yes
7	5	arnd.vom-hofe@mathesis.de	TimeOut	2	5	3	1	no
7	6	arnd.vom-hofe@mathesis.de	TooLong	2	3	2	0	yes
7	9	arnd.vom-hofe@mathesis.de	TooLong	1	3	2	0	no

For any entry the status counters, the threshold limits and the activation status is shown. Direct links allow for navigation to the client or chain.

Active notifications

Currently active notifications are displayed using menu entry 'Results' – 'Notifications' – 'active', or by using command 'no l' :

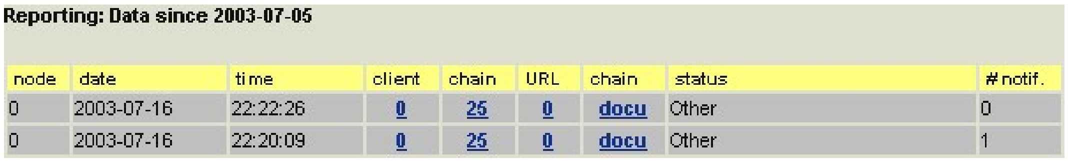


client	chain	doc	cause	target	type	next at	period	since	level	actions
13	10	doc	Content	arnd.vom-hofe@mathesis.de	e	2003-07-25 14:48:31	300 min	2003-07-25 09:48:31	5	delete
13	16	doc	Content	arnd.vom-hofe@mathesis.de	e	2003-07-25 14:48:31	300 min	2003-07-25 09:48:31	5	delete
14	16	doc	Content	arnd.vom-hofe@mathesis.de	e	2003-07-25 15:10:00	300 min	2003-07-25 10:10:00	5	delete
14	2	doc	Content	arnd.vom-hofe@mathesis.de	e	2003-07-25 15:10:03	300 min	2003-07-25 10:10:03	5	delete
14	12	doc	Content	arnd.vom-hofe@mathesis.de	e	2003-07-25 10:41:00	30 min	2003-07-25 10:11:00	5	delete

Direct links allow for navigation to the client or chain, the documentation, and to delete the notification.

Past notifications

Historic Notification events are displayed using menu entry 'Results' – 'Notifications' – 'History', or by using command 'rn <days_back> <maxlines> [<chain>]' :



Reporting: Data since 2003-07-05

node	date	time	client	chain	URL	chain	status	# notif.
0	2003-07-16	22:22:26	0	25	0	docu	Other	0
0	2003-07-16	22:20:09	0	25	0	docu	Other	1

Direct links allow for navigation to the client or chain, and the documentation.

Generation of reporting documents

For off-line reporting to non-technical persons SPECTO supports the generation of predefined PDF-formatted documents. Such documents may be generated and viewed on-line, or can be automatically generated and distributed in the background when coupled with SPECTO script and batch processing.

On-line generation of PDF documents

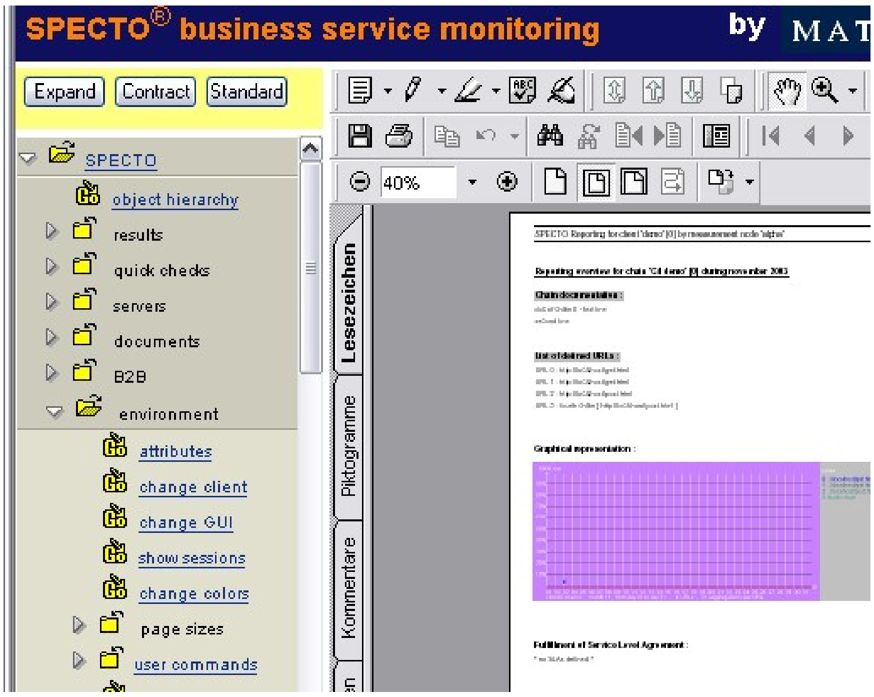
The command **`pdf`** ***`<type> <subtype> <chain> <elements_back> <limit>`*** … is available to generate and view PDF based reports online.
Type : 0=month, 1=week, 2=day
SubType : 0=overview summary, 1=all measurements

Background generation of PDF documents

Both SPECTO script implementations allow the generation of PDF formatted reports analog to the 'pdf' command described above (see the script chapters for more details).

Example

Execution ('**`pdf 0 0 0 0`**') results in a 'link' screen, following the link displays the PDF document in the browser :



SPECTO Reporting for client 'demo' [0] by measurement node 'alpha'

Reporting overview for chain 'C4 demo' [0] during november 2003

Chain documentation :

List of defined URLs :

Graphical representation :

Fulfillment of Service Level Agreement :

* no SLAs defined *

Depending on the plug-in used, the generated pdf document can be printed or saved to local disk from here.

User reporting

SPECTO stores all results in tables of the underlying relational database. There they can be accessed with common methods and the data can be formatted using office (eg. MS Excel) or tools for statistical analysis (e.g. SAS).

Example database ODBC definitions for MS Access are provided together with the SPECTO package.

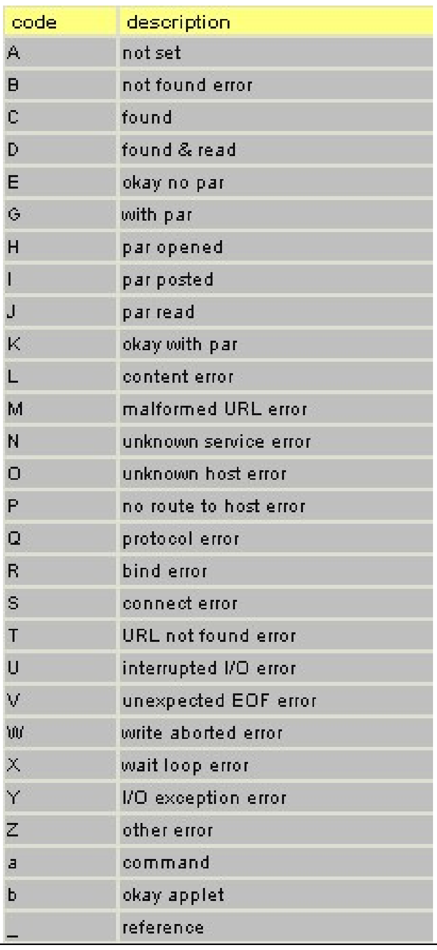
The layout of the primary reporting table (table 'specto.results') is :

```
"node_id"           integer not null,
"specto_id"         integer not null,
"client_id"         integer not null,
"chain_id"          integer not null,
"url_id"            integer not null,
"datestamp"         date not null,
"timestamp"         time not null,
"result"            char(1),
"delay"             integer,
```

The 'delay' column is in milliseconds. Negative numbers denote problems; they are further explained in the 'result' column.

The client_id, chain_id and URL_id columns are internal keys and are referenced in tables 'clients', 'chains' and 'urls'.

The result column is coded as :



code	description
A	not set
B	not found error
C	found
D	found & read
E	okay no par
G	with par
H	par opened
I	par posted
J	par read
K	okay with par
L	content error
M	malformed URL error
N	unknown service error
O	unknown host error
P	no route to host error
Q	protocol error
R	bind error
S	connect error
T	URL not found error
U	interrupted I/O error
V	unexpected EOF error
W	write aborted error
X	wait loop error
Y	I/O exception error
Z	other error
a	command
b	okay applet
_	reference

If the result column contains a number, the current entry is not a measurement but a message from the notification engine and the result column contains the number of active notifications at that moment.

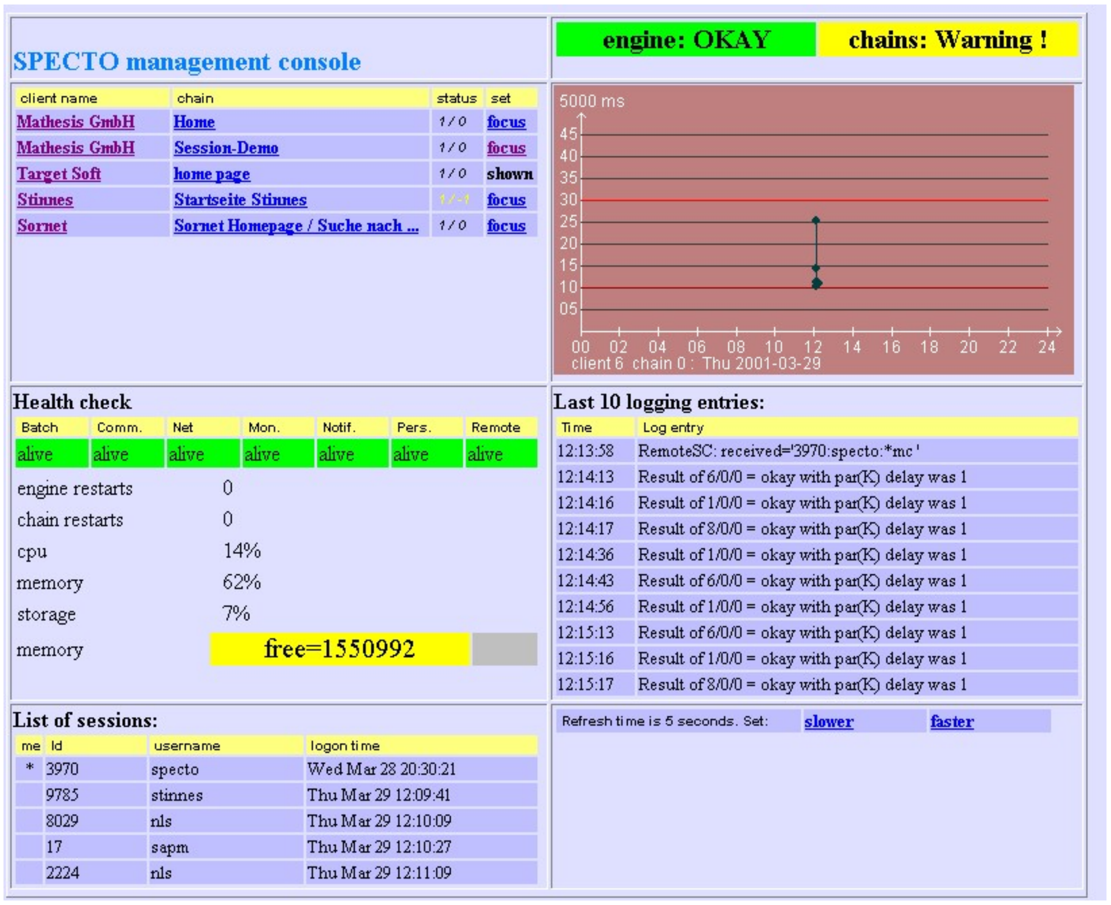
Notes :

- A message starting with 'okay' denotes a successful URL fetch.
- A message ending with 'error' denotes an URL fetch resulting in an error.
- The 'command' message denotes the execution of a SPECTO command (e.g. 'goto')
- The 'reference' message denotes an additional measurement to a reference URL (see chapter reference measurements).

Master console

The master console is a highly configurable summary of all major SPEPTO data. It refreshes at selectable intervals and is started using the navigation menu or using the 'mc' command :

Example view :



SPECTO management console

engine: OKAY | chains: Warning !

client name	chain	status	set
Mathesis GmbH	Home	1 / 0	focus
Mathesis GmbH	Session-Demo	1 / 0	focus
Target Soft	home page	1 / 0	shown
Stinnes	Startseite Stinnes	1 / -1	focus
Sornet	Sornet Homepage / Suche nach ...	1 / 0	focus

Health check

Batch	Comm.	Net	Mon.	Notif.	Pers.	Remote
alive	alive	alive	alive	alive	alive	alive

engine restarts	0
chain restarts	0
cpu	14%
memory	62%
storage	7%
memory	free=1550992

Last 10 logging entries:

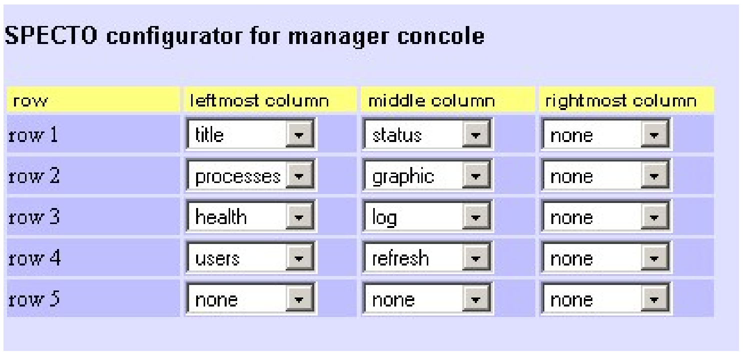
Time	Log entry
12:13:58	RemoteSC: received='3970:specto:*mc '
12:14:13	Result of 6/0/0 = okay with par(K) delay was 1
12:14:16	Result of 1/0/0 = okay with par(K) delay was 1
12:14:17	Result of 8/0/0 = okay with par(K) delay was 1
12:14:36	Result of 1/0/0 = okay with par(K) delay was 1
12:14:43	Result of 6/0/0 = okay with par(K) delay was 1
12:14:56	Result of 1/0/0 = okay with par(K) delay was 1
12:15:13	Result of 6/0/0 = okay with par(K) delay was 1
12:15:16	Result of 1/0/0 = okay with par(K) delay was 1
12:15:17	Result of 8/0/0 = okay with par(K) delay was 1

List of sessions:

me	Id	username	logon time
*	3970	specto	Wed Mar 28 20:30:21
	9785	stinnes	Thu Mar 29 12:09:41
	8029	nls	Thu Mar 29 12:10:09
	17	sapm	Thu Mar 29 12:10:27
	2224	nls	Thu Mar 29 12:11:09

Refresh time is 5 seconds. Set: slower faster

The console can be configured using the 'mcc' command :



SPECTO configurator for manager concole

row	leftmost column	middle column	rightmost column
row 1	title	status	none
row 2	processes	graphic	none
row 3	health	log	none
row 4	users	refresh	none
row 5	none	none	none

The three columns and five rows can be customized with the several modules of the master console.

Available modules :

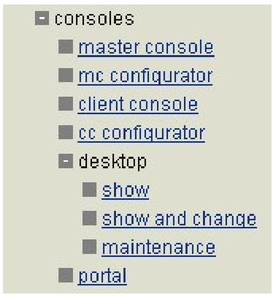
Module	Function
Title	Displays the SPECTO title
Time	Displays the current date and time (of the SPECTO engine)
Health	Displays performance statistics about the SPECTO engine
Processes	Displays running threads (multiple 'process' windows may be opened)
Graphic	Displays graphical view of one process. The process may be selected in the 'process' module
Status	A two field, color coded quick check of the status of SPECTO engine and running processes.
Users	A list of current logged on users and sessions
Refresh	A form to adjust the console refresh rate
Log	The last 10 log entries
Script	One of three Scripts which will be executed at each iteration of the console. The result of the script is displayed in the window.
Variable	One of three predefined variables which's content is output in the window.

Client console

The customer console provides for a highly configurable 'portal' for a user. It consists of a subset of the functionality of the 'master console.

The client console may be accessed via the navigation menu or using command 'cv'.

Desktop

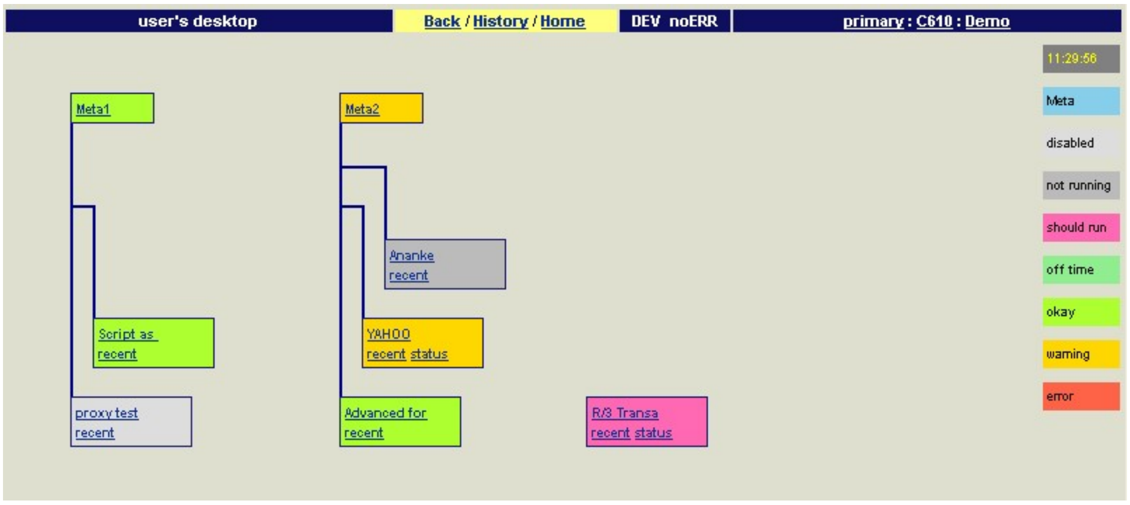


The desktop is a graphical, self refreshing display of the current status of selected chains, and is activated using the navigation menu or by issuing a 'desk' command. (Note: Currently only IE >= 5.5 with ActiveX enabled is supported).

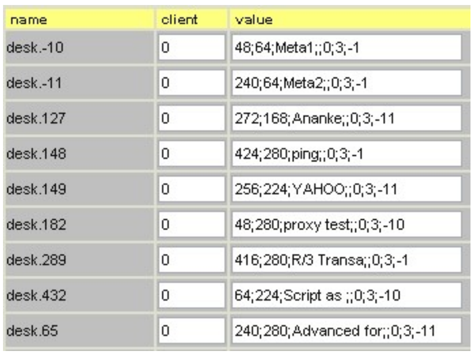
The desktop can be used in 'active' mode (menu '*show and change*' / command '`desc s`'), where the icons may be dragged to new positions, or in 'view only' mode (menu '*show*' / command '`desc v`') in which it is not possible to rearrange the icons.

On the screen larger icons represent chains; for chains of the current client to be shown on the desktop they have to be enabled via the '*for desk*' checkbox ☑ for desk in the chain configuration screen. The status of every chain is identified by a color according to the color legend shown on the right hand side of the screen.

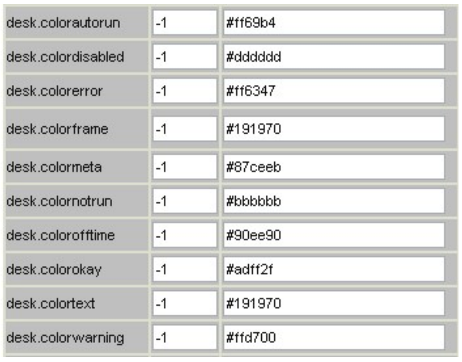
Aside the chain-based icons, 'meta' icons can be defined and linked to groups of chain-icons. Meta icons inherit the status of the linked chain icons. (In the example below the meta icon 'Meta2' is colored yellow ('warnings') because there is at least one yellow icon linked to it). Chain icons may be dragged on top of each other to minimize desk space; the meta icon will still reflect the over all status of this chain group.



The desktop's configuration and appearance is customized via attributes (command '`cu desk`'); a more elegant solution will be provided in a future release.



name	client	value
desk.-10	0	48;64;Meta1;;0;3;-1
desk.-11	0	240;64;Meta2;;0;3;-1
desk.127	0	272;168;Ananke;;0;3;-11
desk.148	0	424;280;ping;;0;3;-1
desk.149	0	256;224;YAHOO;;0;3;-11
desk.182	0	48;280;proxy test;;0;3;-10
desk.289	0	416;280;R/3 Transa;;0;3;-1
desk.432	0	64;224;Script as ;;0;3;-10
desk.65	0	240;280;Advanced for;;0;3;-11



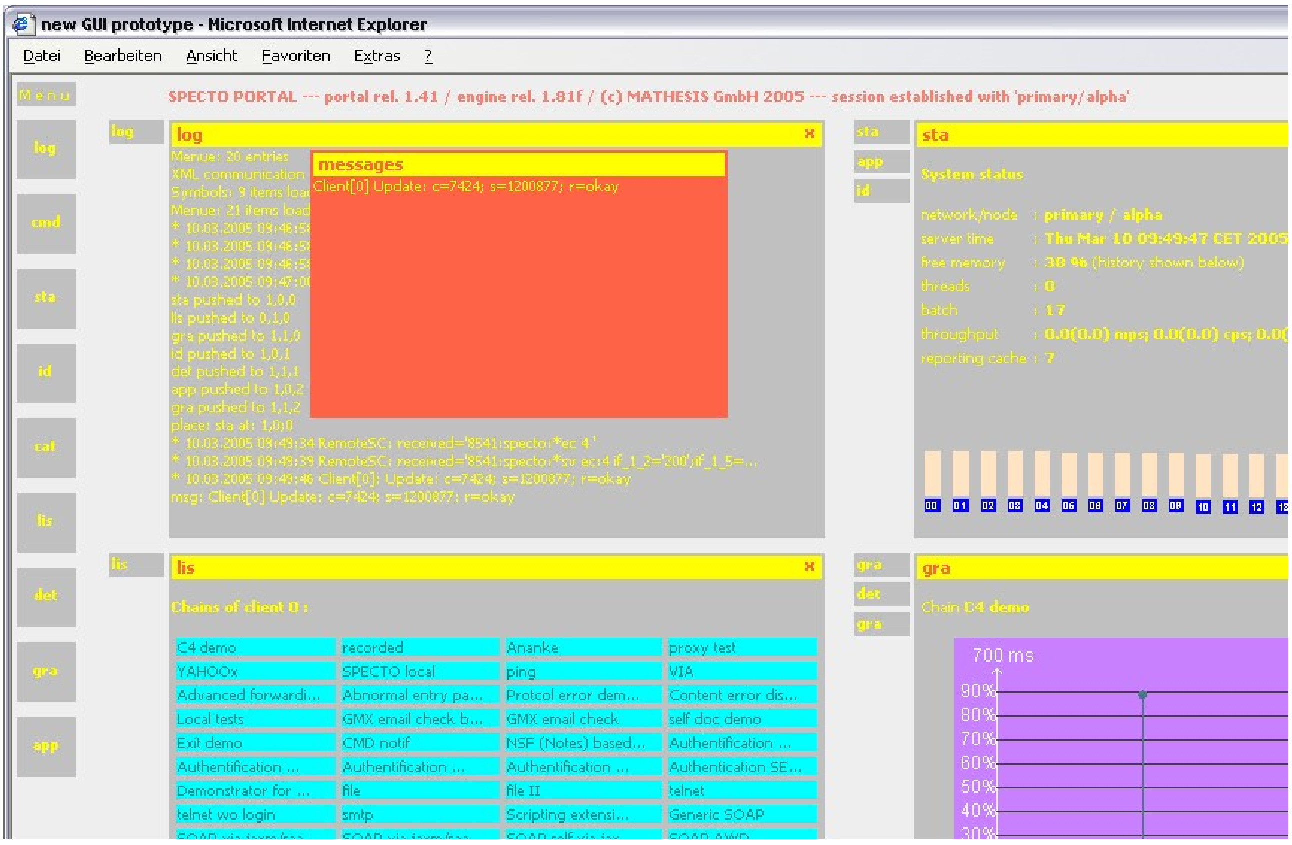
desk.colorautorun	-1	#ff69b4
desk.colordisabled	-1	#dddddd
desk.colorerror	-1	#ff6347
desk.colorframe	-1	#191970
desk.colormeta	-1	#87ceeb
desk.colornotrun	-1	#bbbbbb
desk.colorofftime	-1	#90ee90
desk.colorokay	-1	#adff2f
desk.colortext	-1	#191970
desk.colorwarning	-1	#ffd700

Portal

The SPECTO Portal is a successor to the current consoles and is available using the '`portal`' command.

Notes about the SPECTO Portal :

- The portal was introduced to the SPECTO gui with release 1.81 as a preview; it is still under development and should be considered as a 'preliminary'.
- The portal implementation relies on 'avant garde' capabilities of the web browser; only the latest releases of Internet explorer and Netscape/Mozilla will be able to support the portal correctly. (Apple Safari will be adapted later).
- The 'portal' command differs from other SPECTO commands in that it opens a new browser window.
- The task of the portal is to retain an overview about that is going on in a SPECTO instance; maintenance has still to be done using the 'classic' gui.



SPECTO PORTAL --- portal rel. 1.41 / engine rel. 1.81f / (c) MATHESIS GmbH 2005 --- session established with 'primary/alpha'

```
Menue: 20 entries
XML communication
Symbols: 9 items loa
Menue: 21 items load
* 10.03.2005 09:46:5
* 10.03.2005 09:46:5
* 10.03.2005 09:46:5
* 10.03.2005 09:47:0
sta pushed to 1,0,0
lis pushed to 0,1,0
gra pushed to 1,1,0
id pushed to 1,0,1
det pushed to 1,1,1
app pushed to 1,0,2
gra pushed to 1,1,2
place: sta at: 1,0;0
* 10.03.2005 09:49:34 RemoteSC: received='8541:specto:*ec 4 '
* 10.03.2005 09:49:39 RemoteSC: received='8541:specto:*sv ec:4 if_1_2='200';if_1_5=...
* 10.03.2005 09:49:46 Client[0]: Update: c=7424; s=1200877; r=okay
msg: Client[0] Update: c=7424; s=1200877; r=okay
```

```
Client[0] Update: c=7424; s=1200877; r=okay
```

System status

```
network/node    : primary / alpha
server time     : Thu Mar 10 09:49:47 CET 2005
free memory     : 38 % (history shown below)
threads         : 0
batch           : 17
throughput      : 0.0(0.0) mps; 0.0(0.0) cps; 0.0(
reporting cache : 7
```

Chains of client 0 :

C4 demo	recorded	Ananke	proxy test
YAHOOx	SPECTO local	ping	VIA
Advanced forwardi...	Abnormal entry pa...	Protcol error dem...	Content error dis...
Local tests	GMX email check b...	GMX email check	self doc demo
Exit demo	CMD notif	NSF (Notes) based...	Authentification ...
Authentification ...	Authentification ...	Authentification ...	Authentication SE...
Demonstrator for ...	file	file II	telnet
telnet wo login	smtp	Scripting extensi...	Generic SOAP

Chain C4 demo

The basic arrangement of the portal consists of :

- The 'views' (left hand side squares, e.g. 'log')
- The 'windows', arranged in 'places' (four in the example above)
- The data 'categories' (type 'cat')

Operation within the portal is primarily via drag&drop:

- Drag a view or a window to a place
- Drag a object to a window

Service level agreements

Overview

SPECTO supports definition and execution of service level agreements ('SLA') on a chain base. SLAs are formatted as PDF type documents and may be generated manually on request or automatically and distributed via electronic mail.

SLAs for chain measurements are built of sets of rules for computing 'fullfillment' percentages. The rules are mainly weighting factors for individual aspects as which URL 'time of day', 'day of week', 'degree of deviation'.

Based on the rules a ***SLA contract*** can be generated. Such a contract may be used as formal document e.g. between provider and customer of a service.

Based on the rules and the executed measurements a ***SLA report*** can be generated for a certain timeframe (a week or a month). The report may be enhanced with comments to detail a particular event.
The report summarizes the selected measurements graphically and by numbers. Also a daily and a total fulfillment is computed. Optionally, based on the fulfillment result, penalties may be computed.
Such a report may be used for regular quality meetings e.g. between provider and customer of a service.

Enabling management of service level agreement :

Within customizing menu 'service level agr.' Set the flag '`sla.cust.enabled`' to '`true`' for all (-1) or selected clients. There is no restart required.

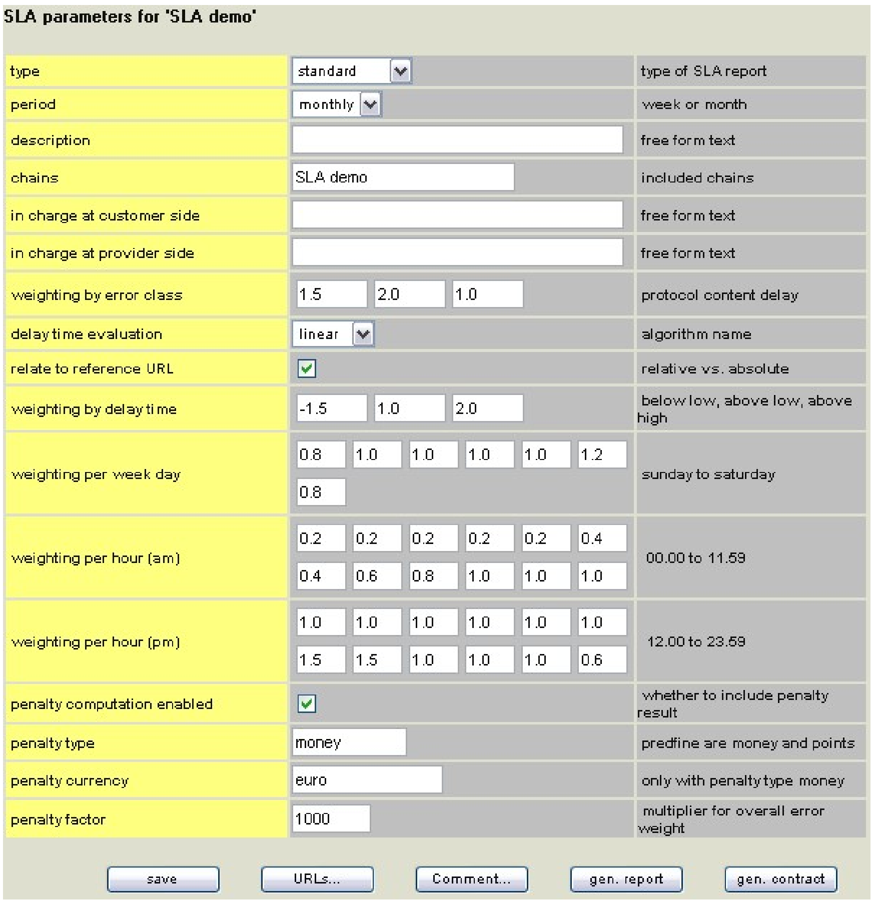
Customizing of service level agreement management :

The look and feel of SLA reports and contracts may be customized using standard SPECTO customizing : Menu '`customizing - PDF formatting`'. (attribute grouping '`pdf.`').

Note : To ease using SPECTO's SLA module, an 'explanation by example' is included at the end of this chapter.

Definition of a service level agreement :

The SLA editor is available via command ('`sla e <chained>`') or the '`sla`' link in the chain configuration screen (right hand side of the 'doc' link).



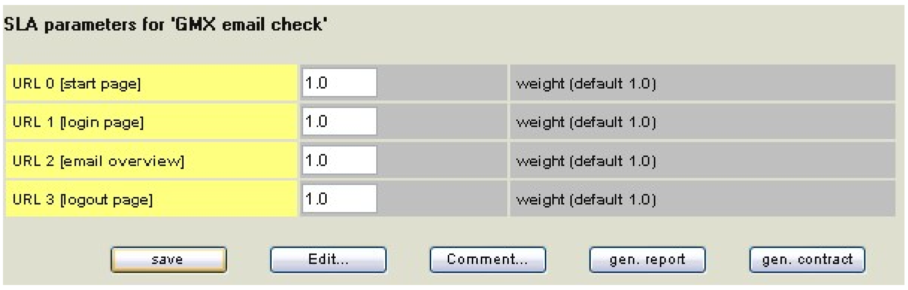
SLA parameters for 'SLA demo'

type	standard	type of SLA report
period	monthly	week or month
description		free form text
chains	SLA demo	included chains
in charge at customer side		free form text
in charge at provider side		free form text
weighting by error class	1.5 2.0 1.0	protocol content delay
delay time evaluation	linear	algorithm name
relate to reference URL	✓	relative vs. absolute
weighting by delay time	-1.5 1.0 2.0	below low, above low, above high
weighting per week day	0.8 1.0 1.0 1.0 1.0 1.2 0.8	sunday to saturday
weighting per hour (am)	0.2 0.2 0.2 0.2 0.2 0.4 0.4 0.6 0.8 1.0 1.0 1.0	00.00 to 11.59
weighting per hour (pm)	1.0 1.0 1.0 1.0 1.0 1.0 1.5 1.5 1.0 1.0 1.0 0.6	12.00 to 23.59
penalty computation enabled	✓	whether to include penalty result
penalty type	money	predfine are money and points
penalty currency	euro	only with penalty type money
penalty factor	1000	multiplier for overall error weight

Values are stored on execution of the 'Save' button.

The 'Comment…' button switches to the commenting form (described later).
The 'gen. report' and 'gen. contract' buttons switch to PDF generation of reports and contracts (described later).

The 'URLs…' button switches to the maintenance of URL based weighting :

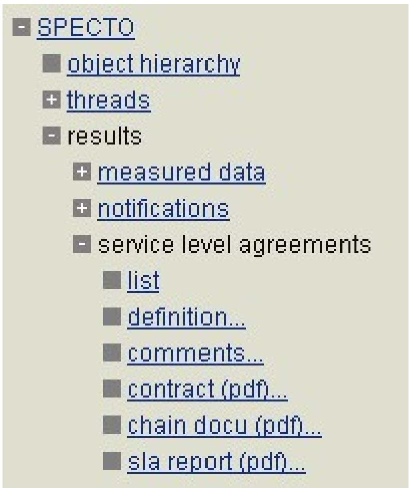


SLA parameters for 'GMX email check'

URL 0 [start page]	1.0	weight (default 1.0)
URL 1 [login page]	1.0	weight (default 1.0)
URL 2 [email overview]	1.0	weight (default 1.0)
URL 3 [logout page]	1.0	weight (default 1.0)

Working with service level agreements :

The SLA functionality is available via the 'service level agreements' menu in the 'results section or by using the 'sla' command ('sla ?' for help).



The common steps are :

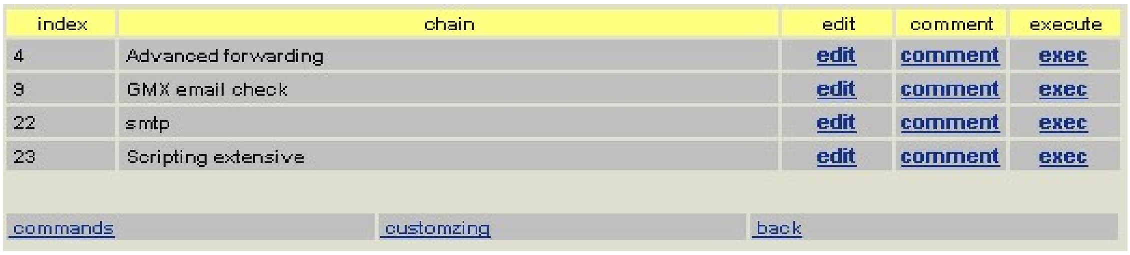
i. Defining an SLA and generating the associated SLA contract (once).
ii. Commenting SLA performance and generating SLA reports (per time period)

Optionally (as an alternative to step ii. from above) :

i. Automating the report generation and distribution by the SPECTO batch processor and an associated script program (see the NLS support area for sample scripts).

Listing all defined service level agreements :

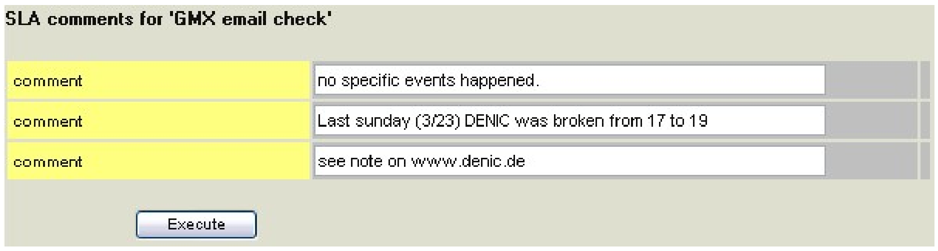
A list of he defined SLAs within the current client is available via command ('**sla l**') or the 'list' entry within menu 'service level agreements'.



index	chain	edit	comment	execute
4	Advanced forwarding	edit	comment	exec
9	GMX email check	edit	comment	exec
22	smtp	edit	comment	exec
23	Scripting extensive	edit	comment	exec

Maintaining comments to the current status of a service level agreement :

The SLA comment editor is available via command ('**sla c <chained>**'), the 'comments' entry within menu 'service level agreements' or via the 'list' screen (see above).



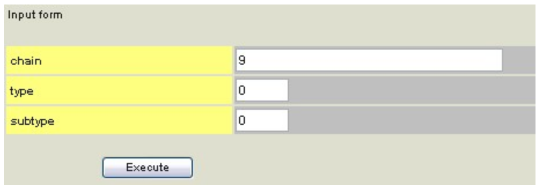
SLA comments for 'GMX email check'

comment	no specific events happened.
comment	Last sunday (3/23) DENIC was broken from 17 to 19
comment	see note on www.denic.de

Note : Previous comments are not stored. Any SLA report is generated with the current comment set.

Generating a service level agreement report :

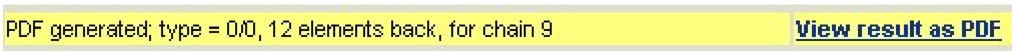
SLA reports are generated via command (‘**`sla p <chained> <type> <subtype> <time_back`**’), the ‘sla report (PDF)’ entry within menu ‘service level agreements’ or via the ‘list’ screen (see above).



Input form

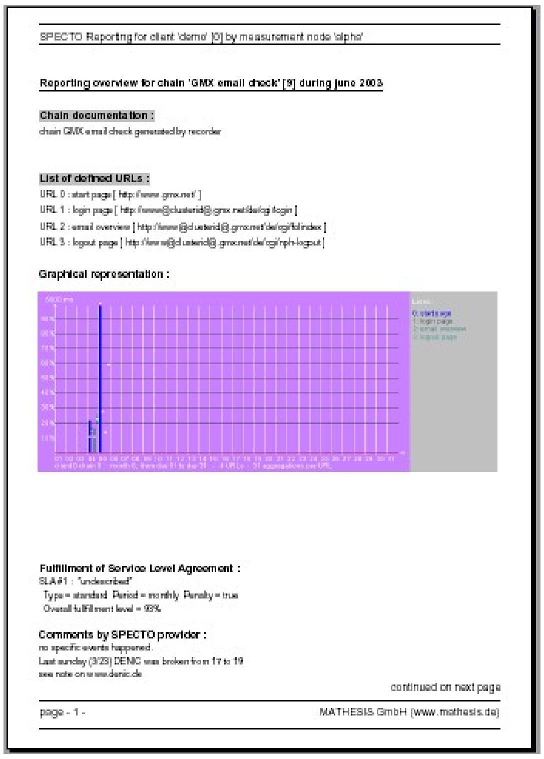
chain	9
type	0
subtype	0

After the ‘Excecute’ button the SLA is computed and formatted as a PDF document. A link to the PDF document is generated :



PDF generated; type = 0/0, 12 elements back, for chain 9

The document may be reviewed online or printed or saved locally.



SPECTO Reporting for client 'demo' [0] by measurement node 'alpha'

Reporting overview for chain 'GMX email check' [9] during june 2003

Chain documentation :
chain GMX email check generated by recorder

List of defined URLs :
URL 0 : start page [http://www.gmx.net/]
URL 1 : login page [http://www@clusterid@.gmx.net/de/cgi/login]
URL 2 : email overview [http://www@clusterid@.gmx.net/de/cgi/folindex]
URL 3 : logout page [http://www@clusterid@.gmx.net/de/cgi/nph-logout]

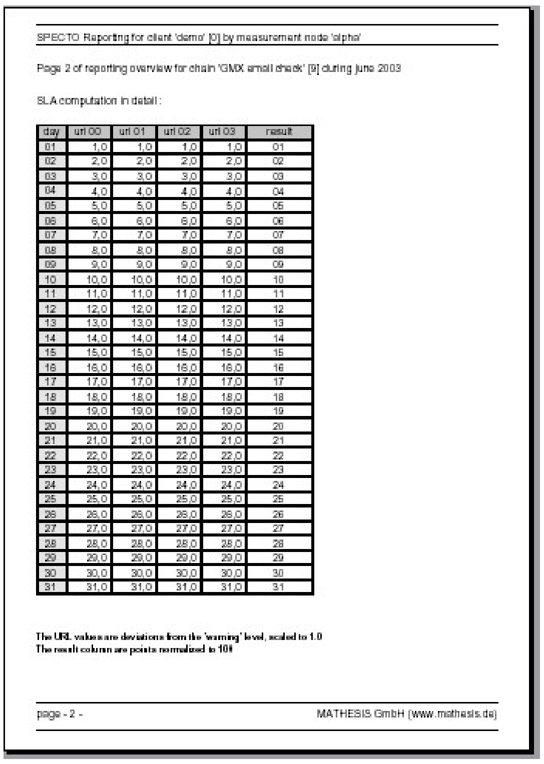
Graphical representation :

Fulfillment of Service Level Agreement :
SLA #1 : "undescribed"
Type = standard Period = monthly Penalty = true
Overall fulfillment level = 93%

Comments by SPECTO provider :
no specific events happened.
Last sunday (3/23) DENIC was broken from 17 to 19
see note on www.denic.de

continued on next page

page - 1 - MATHESIS GmbH (www.mathesis.de)



SPECTO Reporting for client 'demo' [0] by measurement node 'alpha'

Page 2 of reporting overview for chain 'GMX email check' [9] during june 2003

SLA computation in detail :

day	url 00	url 01	url 02	url 03	result
01	1,0	1,0	1,0	1,0	01
02	2,0	2,0	2,0	2,0	02
03	3,0	3,0	3,0	3,0	03
04	4,0	4,0	4,0	4,0	04
05	5,0	5,0	5,0	5,0	05
06	6,0	6,0	6,0	6,0	06
07	7,0	7,0	7,0	7,0	07
08	8,0	8,0	8,0	8,0	08
09	9,0	9,0	9,0	9,0	09
10	10,0	10,0	10,0	10,0	10
11	11,0	11,0	11,0	11,0	11
12	12,0	12,0	12,0	12,0	12
13	13,0	13,0	13,0	13,0	13
14	14,0	14,0	14,0	14,0	14
15	15,0	15,0	15,0	15,0	15
16	16,0	16,0	16,0	16,0	16
17	17,0	17,0	17,0	17,0	17
18	18,0	18,0	18,0	18,0	18
19	19,0	19,0	19,0	19,0	19
20	20,0	20,0	20,0	20,0	20
21	21,0	21,0	21,0	21,0	21
22	22,0	22,0	22,0	22,0	22
23	23,0	23,0	23,0	23,0	23
24	24,0	24,0	24,0	24,0	24
25	25,0	25,0	25,0	25,0	25
26	26,0	26,0	26,0	26,0	26
27	27,0	27,0	27,0	27,0	27
28	28,0	28,0	28,0	28,0	28
29	29,0	29,0	29,0	29,0	29
30	30,0	30,0	30,0	30,0	30
31	31,0	31,0	31,0	31,0	31

The URL values are deviations from the 'warning' level, scaled to 1.0
The result column are points normalized to 10#

page - 2 - MATHESIS GmbH (www.mathesis.de)

The first page consists of a graphical view similar to the normal reporting but enhanced with the SLA overview and the comments section. The second page displays the week/month based detail table of SLA performance.

Within the table, the day entries have the following meaning:

- first line is error weight and fulfillment percentage;
- second line is number of time limit deviations, number of errors, overall number of measurements.

Error weight and percentage computation

A SLA report displays two values :

- Error weight. Weighted sum up of all errors over the reported period. Ranging from -10.0 to +10.0; the higher, the worse. 0 stands for 'reaches the norm'.
- Fulfillment percentage. Range from 0 to 200%, the higher the better.

The SLA computation consists of two steps.

- First all the raw measurement data is aggregated on a per url/hour basis, and the corresponding error weight and fulfillment percentage is computed. (this intermediate data may be displayed using command '`sla r 0` *`chain monthsBack`*' command, but this is not necessary during normal work).
- The second step aggregates the above data on day-by-url, day-by chain and week or month basis. This data is shown in the SLA report.

Error weight computation

The error weight is computed based on the amount of communication errors (e.g. protocol), the amount of content errors and the deviations from the expected delay time.

Basic to the understanding of the SLA computation of time deviations is the concept of the 'norm'; this is a delay which is considered as 'value to be achieved'; it is computed by a percentage (default is 50%, customizable with attribute '`sla.cust.normprc`') of the urls '`toolong`' value (default '`toolong`' is 2000 milliseconds).

There are several algorithms ('Steps', 'Linear', 'Custom') for delay time based error weight computation :

If the measured delay is below norm a negative error weight is computed by :

```
ErrorWeight('Linear') = ( Norm-Delay ) / Norm * WeightBelowLow
ErrorWeight('Steps')  = 0.5 * WeightBelowLow
```

If the measured delay is between norm and toolong no error weight is set.

```
ErrorWeight('Linear') = 0.0
ErrorWeight('Steps')  = 0.0
```

If the measured delay is between toolong and timeout :

```
ErrorWeight('Linear') = (Delay-TooLong ) / TooLong * WeightAboveLow
ErrorWeight('Steps')  = 0.5 * WeightAboveLow
```

If the measured delay is above timeout :

```
ErrorWeight('Linear') = ( TimeOut-TooLong ) / TooLong * WeightBelowLow
                      + ( Delay-Timeout ) / TimeOut * WeightAboveHigh
ErrorWeight('Steps')  = 0.5 * WeightBelowLow
                      + 0.5 * WeightAboveHigh
```

The individual error weights from the communication, content and delay error weights are merged together by multiplying them with the appropriate '*error class*' weights.

This hour-based error weights are weighted by the WeightByHourOfDay and form the day-by-url error weight and fulfillment percentage.

Fulfillment percentage computation

The *error weight* multiplied with the 'w2p' ('weight to percentage') factor (default is 25%, customizable with attribute 'sla.cust.w2p'), subtracted from 100% and then limited to 0 – 200%.

'Result of Day' computation

The days *error weights* of the individual URLs are weighted according the WeightByURL and form the overall day error weight and fulfillment percentage.

'Result of Week / Month' computation

The individual days *error weights* are weighted according to the WeightByDayOfWeek and form the overall week or month error weight and fulfillment percentage.

Penalty computation

The *overall fulfillment percentage* is (optionally) used to compute a penalty. Penalties may be based on money (then the currency value is used) or on any other crediting scheme (e.g. 'points'). Penalty values may be transferred from one reporting period to the next. The penalty value is computed by multiplying the overall error weight with the penalty factor.

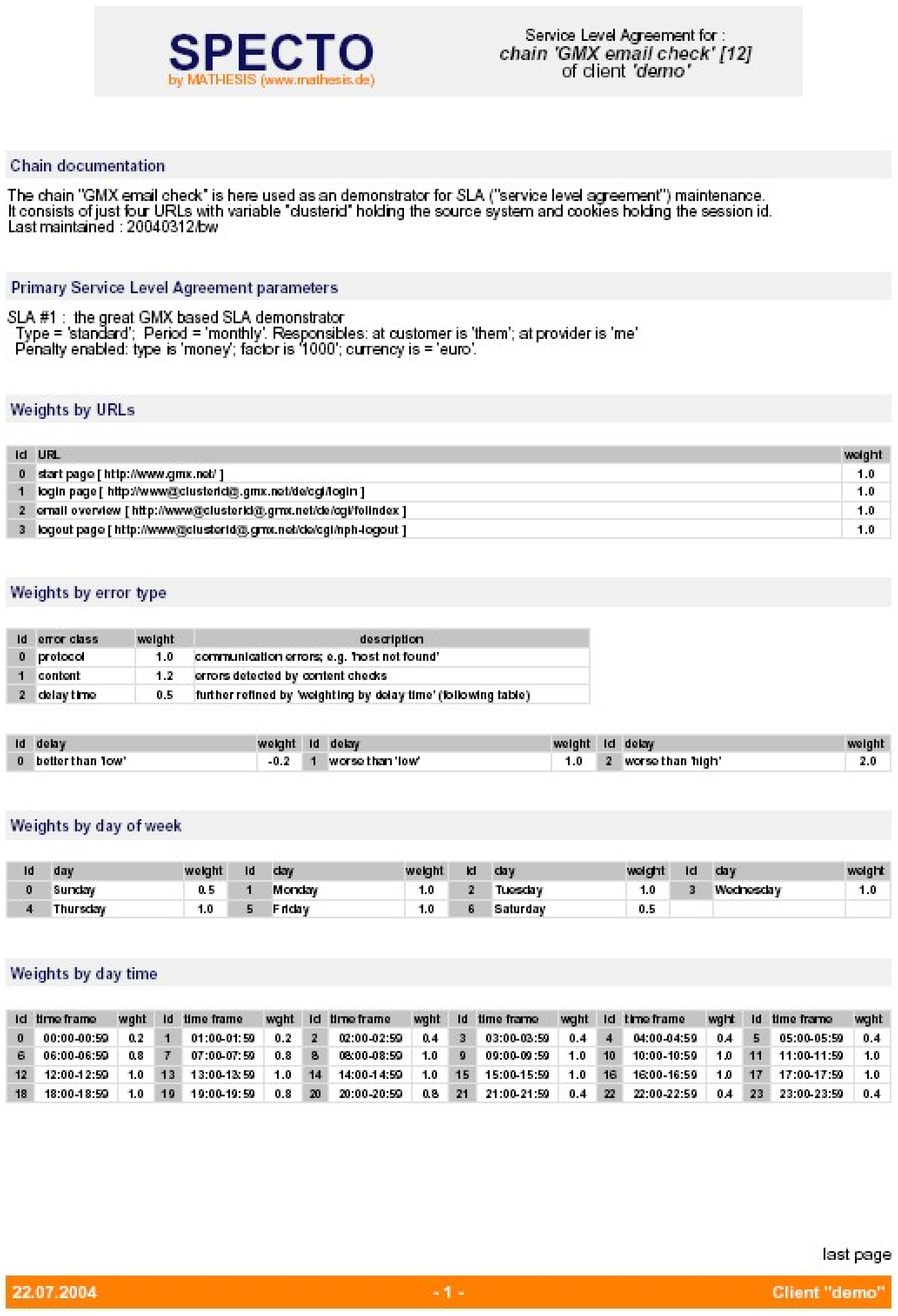
Custom computation of error weights, fulfillment percentages and penalty values

The algorithms described above may be overridden by custom computations; e.g. to take response time during incidents into account.

Such algorithms have to be developed using the SPECTO's engine Javascript language and are called by the SLA modules exit interface. Several sample scripts are available from NLS upon request.

Generating a service level agreement contract :

SLA reports are generated via command (‘**`sla o <chained> <type> <subtype> <time_back`**’), the ‘sla report (PDF)’ entry within menu ‘service level agreements’ or via the ‘list’ screen (see above)
Sample :



SPECTO
by MATHESIS (www.mathesis.de)

Service Level Agreement for :
chain 'GMX email check' [12]
of client *'demo'*

Chain documentation

The chain "GMX email check" is here used as an demonstrator for SLA ("service level agreement") maintenance.
It consists of just four URLs with variable "clusterid" holding the source system and cookies holding the session id.
Last maintained : 20040312/bw

Primary Service Level Agreement parameters

SLA #1 : the great GMX based SLA demonstrator
Type = 'standard'; Period = 'monthly'. Responsibles: at customer is 'them'; at provider is 'me'
Penalty enabled: type is 'money'; factor is '1000'; currency is = 'euro'.

Weights by URLs

id	URL	weight
0	start page [http://www.gmx.net/]	1.0
1	login page [http://www@clusterid@.gmx.net/de/cgi/login]	1.0
2	email overview [http://www@clusterid@.gmx.net/de/cgi/folindex]	1.0
3	logout page [http://www@clusterid@.gmx.net/de/cgi/nph-logout]	1.0

Weights by error type

id	error class	weight	description
0	protocol	1.0	communication errors; e.g. 'host not found'
1	content	1.2	errors detected by content checks
2	delay time	0.5	further refined by 'weighting by delay time' (following table)

id	delay	weight	id	delay	weight	id	delay	weight
0	better than 'low'	-0.2	1	worse than 'low'	1.0	2	worse than 'high'	2.0

Weights by day of week

id	day	weight	id	day	weight	id	day	weight	id	day	weight
0	Sunday	0.5	1	Monday	1.0	2	Tuesday	1.0	3	Wednesday	1.0
4	Thursday	1.0	5	Friday	1.0	6	Saturday	0.5			

Weights by day time

id	time frame	wght	id	time frame	wght	id	time frame	wght	id	time frame	wght	id	time frame	wght	id	time frame	wght
0	00:00-00:59	0.2	1	01:00-01:59	0.2	2	02:00-02:59	0.4	3	03:00-03:59	0.4	4	04:00-04:59	0.4	5	05:00-05:59	0.4
6	06:00-06:59	0.8	7	07:00-07:59	0.8	8	08:00-08:59	1.0	9	09:00-09:59	1.0	10	10:00-10:59	1.0	11	11:00-11:59	1.0
12	12:00-12:59	1.0	13	13:00-13:59	1.0	14	14:00-14:59	1.0	15	15:00-15:59	1.0	16	16:00-16:59	1.0	17	17:00-17:59	1.0
18	18:00-18:59	1.0	19	19:00-19:59	0.8	20	20:00-20:59	0.8	21	21:00-21:59	0.4	22	22:00-22:59	0.4	23	23:00-23:59	0.4

last page

22.07.2004 - 1 - Client "demo"

Custom SLA reports and SLA contracts :

During creation of SLA documents the design and the computation can be enhanced through a user maintained script which is called from several places during the document creation process. The name of the script follows the convention '**`sla.chain.<physical_chain_id>.script`**' , e.g. '**`sla.chain.64.script`**'. The physical chain id may be determined using the '**`dpi <chain_id`**>' command.
If for a given SLA no script is found, the default values and computation algorithms are used.

Any such callback ('event') is identified by an event name which is passed as the first parameter of the script call. Depending on the event, other arguments are included in addition.
The script may return values through an XML formatted record.

<u>Sample script :</u>

```
//script=js

var persist = specto.getSessionPersistence();
var sla     = specto.getSLA();
var event   = specto.args[0];

// log parameters, event and first parameter
specto.log("sla: " + specto.args.length + ", " +
  event + ", " + specto.args[1]);


// 'start' event
if (event=="start") {
// save a parameter to be used by other events
  persist.put("file", specto.args[1]);
// set the contract penalty currency
  sla. slaPenaltyCurrency = "Rubel";
// return new values for center title and subtitle
  "<title_center>Jri</title_center><subtitle>Kaana</subtitle>";
}


// before the URL list is processed
else if (event=="beforeURLlist") {
// demo how to get parameters set by other events
  specto.log("sla: back=" + persist.get("file"));
// returns nothing
  "y";
}


// before the SLA values are output
else if (event=="CompTotal") {
// output the pre-computed day values weights
  specto.log("sla: CompDay: " + "a");
  for(i=0; i<30; i++) {
    specto.log(i + ":" + sla);
  }// return new total value
  "<total_percentage>123%</ total_percentage>";
}
```

The SLA structure

The method '**`specto.getSLA()`**' is available to get a reference to the current instance of the SLAManager. This can be used to set further details of the SLA computation process, e.g. the weighting factors.

The exact structure of the instance can be inspected by the '**`struct`**' command.

List of available events for SLA enhancement

SLA callback event	Description
`start`	At start of the SLA generation before anything is written into the PDF document.
`beforeURLList`	Before the list of URLs is written into the document. Used to modify the list itself and its appearance
`SLAParams`	Before SLA specific parameters are output.
`CompDay`	After computation of one day (but before output)
`CompTotal`	After total computation (but before output)

SLA event 'start'

Input parameter	Description
1	Name of the PDF file to be generated
2	Type of the report
3	Subtype of the report
4	Primary chain id
5	Elements (days, weeks, months) back
6	Limit for graphical display (-1 is automatic computation)

Output parameters	Description
`<title_left>`	Left title of the header line of each page
`<title_center>`	Center title of the header line of each page
`<title_right>`	Right title of the header line of each page
`<author>`	'Author' attribute of the PDF document
`<creator>`	'Creator' attribute of the PDF document
`<title>`	
`<subject>`	

SLA event 'beforeURLList'

Input parameter	Description
1	Array of selected chain names
2	Array of selected URL Ids
3	Array of selected URL names

Output parameters	Description

`<add>`	Add an URL to the list
`<remove>`	Remove an URL from the list
`<rename id=…>`	Rename an URL

SLA event 'SLAParams'

Input parameter	Description
1	Settings modified (true/false)

Output parameters	Description
%to be specified%	Several available

SLA event 'CompDay'

Input parameter	Description
1	numURLs
2	numDays

Output parameters	Description
`<day_percentage>`	

SLA event 'CompTotal'

Input parameter	Description
1	numURLs
2	numDays

Output parameters	Description
`<total_percentage>`	The overall fulfillment percentage

The following structures are available within the SLAManager instance; they are used for the SLA report table and can be set during events '`CompDay`' and '`CompTotal`' events :

```
SLAEntry slaEntries [numURLs][24][ numDays];
SLAEntry slaUDEntries [numURLs][ numDays];
SLAEntry slaDEntries [numDays];
SLAEntry slaOverallEntry;
```

The type SLAEntry is defined :

```
   int   delay;
   int   delayReference;
   int   count;
   int   countE;
   int   countT;
   float weight;
   float percentage;
```

Generating a chain documentation :

A complete documentation of a chain may be generated via command (‘**`sla o <chained> <type> <subtype> <time_back`**’), the ‘sla report (PDF)’ entry within menu ‘service level agreements’ or via the ‘list’ screen (see above)

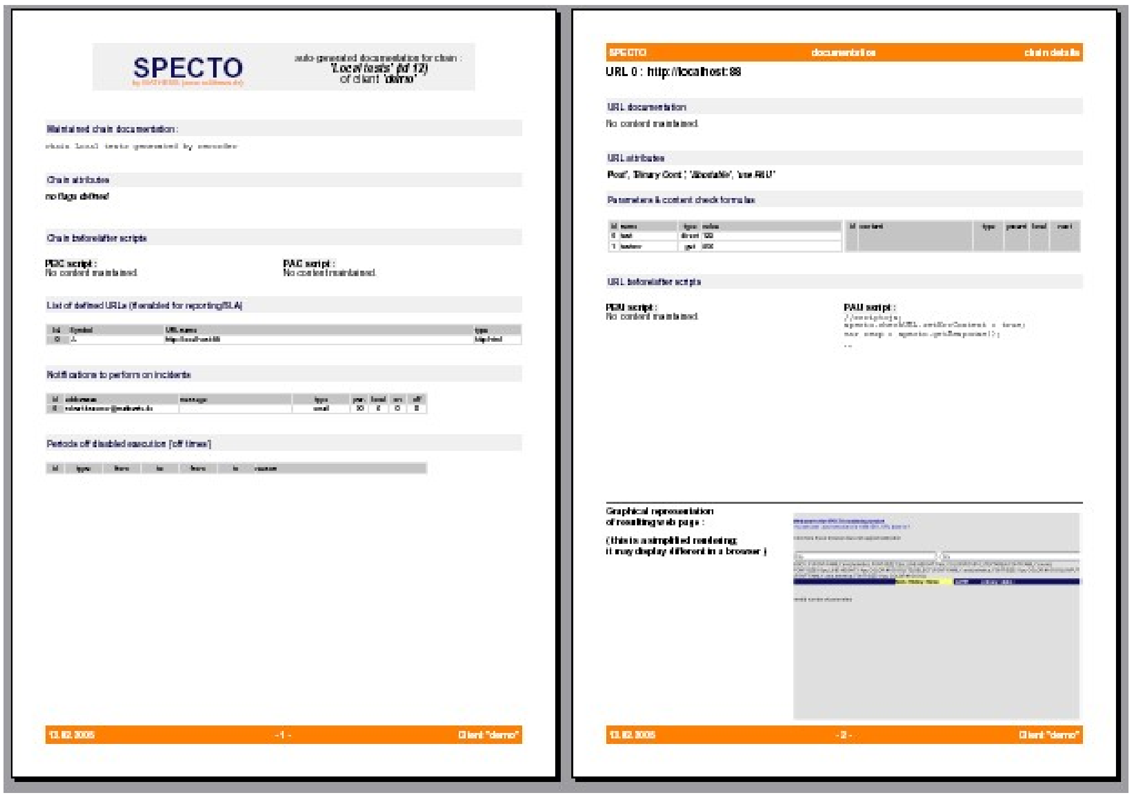
The chain documentation is intended as addendum for an SLA contract but may also be used independent from SLA management for other documentation purposes.

The chain documentation consists of one title page describing the chain, and one detail page for every URL.

He chain page consist of the chain documentation, the PBT, PBC and PAC scripts and the defined notifications.

The URL pages consist of the URL parameters, content check definitions, the documentation, and the PBU and PAU scripts. Also an image of the page is included (note that the image generated internal HTML renderer can only give a rough view of the page).

Sample :



SPECTO

auto generated documentation for chain :
'Local tests' (id 12)
of client 'demo'

Maintained chain documentation :

```
chain local tests generated by [illegible]
```

Chain attributes

no flags defined

Chain before/after scripts

PBC script :
No content maintained.

PAC script :
No content maintained.

List of defined URLs (if enabled for reporting/SLA)

id	Symbol	URL name	type
0	[illegible]	http://localhost:88	http/html

Notifications to perform on incidents

id	addressee	message	type	[illegible]	[illegible]	[illegible]	off
0	[illegible]		email	30	[illegible]	0	0

Periods off disabled execution [off times]

id	type	from	to	from	to	reason

13.02.2005 - 1 - Client "demo"

SPECTO documentation chain details

URL 0 : http://localhost:88

URL documentation

No content maintained.

URL attributes

'Post', 'Binary Cont.', 'Abortable', 'use PAU'

Parameters & content check formulas

id	name	type	value
0	[illegible]	direct	100
1	[illegible]	get	[illegible]

id	content	type	[illegible]	[illegible]	[illegible]

URL before/after scripts

PBU script :
No content maintained.

PAU script :

```
//[illegible]
specto.[illegible] = true;
var resp = specto.getResponse();
...
```

Graphical representation
of resulting web page :
(this is a simplified rendering;
it may display different in a browser)

13.02.2005 - 2 - Client "demo"

Note that the design of the report may be modified in the SLA customizing.

Introduction by example to service level agreements

This chapter explains SLA management by an example. To be able to work with the example it is necessary to define a sample chain and import the supplied sample data.

Generating the sample chain :

Define a chain with a meaningful chains name (e.g. '**`SLA demo`**'), one dummy url (e.g. '**`http://www.nls.de`**'), retaining the default parameters (too long of 2000 milliseconds and timeout of 3500 milliseconds. Maintain some useful documentation.
This chain is only used as a container for the test data, it will not be run.

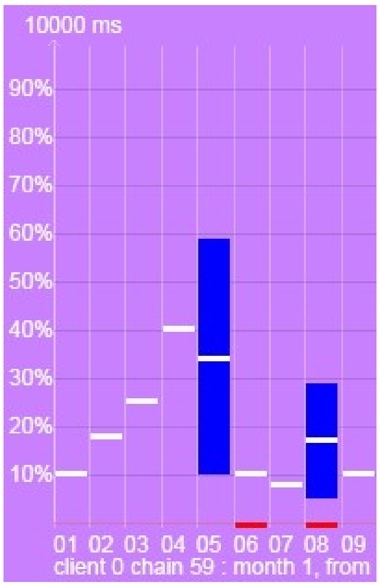
Import the demonstration data :

Transfer the supplied file '**`slatest.csv`**' to the SPECTO file area using menu '*`Maintenance - Imp./exp. (XML) - upload to Specto…`*'.

Import the file using the command **`csvi slatest.csv`** ***`chainid_or_name`***. (e.g. `csvi slatest.csv 'SLA demo'`). The sample data will be loaded into January 2005 of the specified chain.

Note: The file contains data for one month (January 2005) with 100 equally spread entries per day. Most days consist of equal entries with a delay of 1000 milliseconds, but days two to eight contain different values to show how SLA weights are being computed. You' re encouraged to look at the `slatest.csv` file to get familiar with the scenario.

Verify the loaded test-data :



Via menu '*`Results - monthly overview`*' the selection form is shown. Enter the correct '*months back*' value to reach January 2005 (e.g. 1 if in February), your chain name or chain id, and 10000 as limit. The resulting graphic should look like the one shown on the right (picture only partially shown).

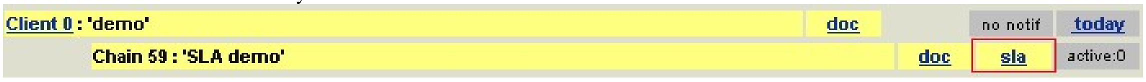
The first, the ninth day and all days after, show the exact 1000 milliseconds measurements in those days. The second to fourth day also show the constant values at 1800, 2500 and 4000 milliseconds respectively and fifth day has the 1000 to 6000 milliseconds range with the average around 3500. Sixth day is similar to the first day but has 10 % content errors (indicated by the red bar at the x-axis). Seventh day has measurements below 1000 milliseconds and eight day is a combination of everything.

Customize SLA :

Check that SLA is enabled in your client: Via menu '*Customizing - Service level agr.*'; the attribute '`sla.cust.enabled`' should read '`true`'.

Edit chain's SLA definition :

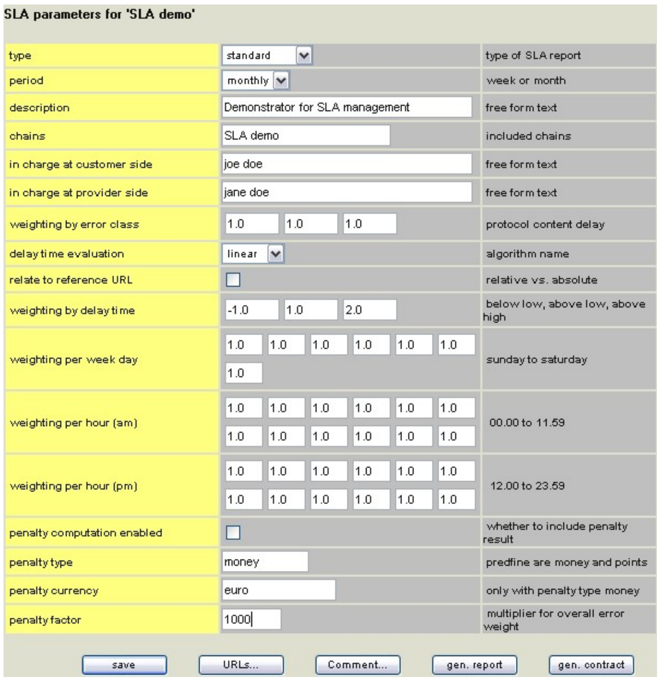
Go to the definition of your chain and follow the 'SLA' link:



Client 0 : 'demo'	doc	no notif	today
Chain 59 : 'SLA demo'	doc	sla	active:0

The SLA definition panel for the chain opens and shows the default entries for new chains.

For a first evaluation, please set all the weights in the 'weighting per week day' and 'weighting per hour' blocks to '1.0'. Also enter some meaningful data in the 'description' and 'in charge…' fields; then save the configuration.



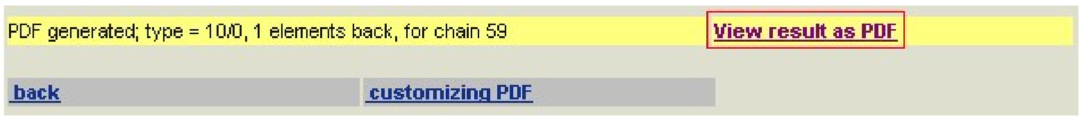
SLA parameters for 'SLA demo'

type	standard	type of SLA report
period	monthly	week or month
description	Demonstrator for SLA management	free form text
chains	SLA demo	included chains
in charge at customer side	joe doe	free form text
in charge at provider side	jane doe	free form text
weighting by error class	1.0 1.0 1.0	protocol content delay
delay time evaluation	linear	algorithm name
relate to reference URL	☐	relative vs. absolute
weighting by delay time	-1.0 1.0 2.0	below low, above low, above high
weighting per week day	1.0 1.0 1.0 1.0 1.0 1.0 1.0	sunday to saturday
weighting per hour (am)	1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0	00.00 to 11.59
weighting per hour (pm)	1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0	12.00 to 23.59
penalty computation enabled	☐	whether to include penalty result
penalty type	money	predfine are money and points
penalty currency	euro	only with penalty type money
penalty factor	1000	multiplier for overall error weight

Generate a SLA report :

From the definition form, execute the 'gen. report' button. (note: the report is always executed for the previous month; because we want to see the January 2005 data, the 'gen. report' buttons only works in February 2005. Otherwise use command '`sla p chain 10 0` *`monthsBack`*' (e.g. `sla p 'SLA demo' 10 0 2` if in march).

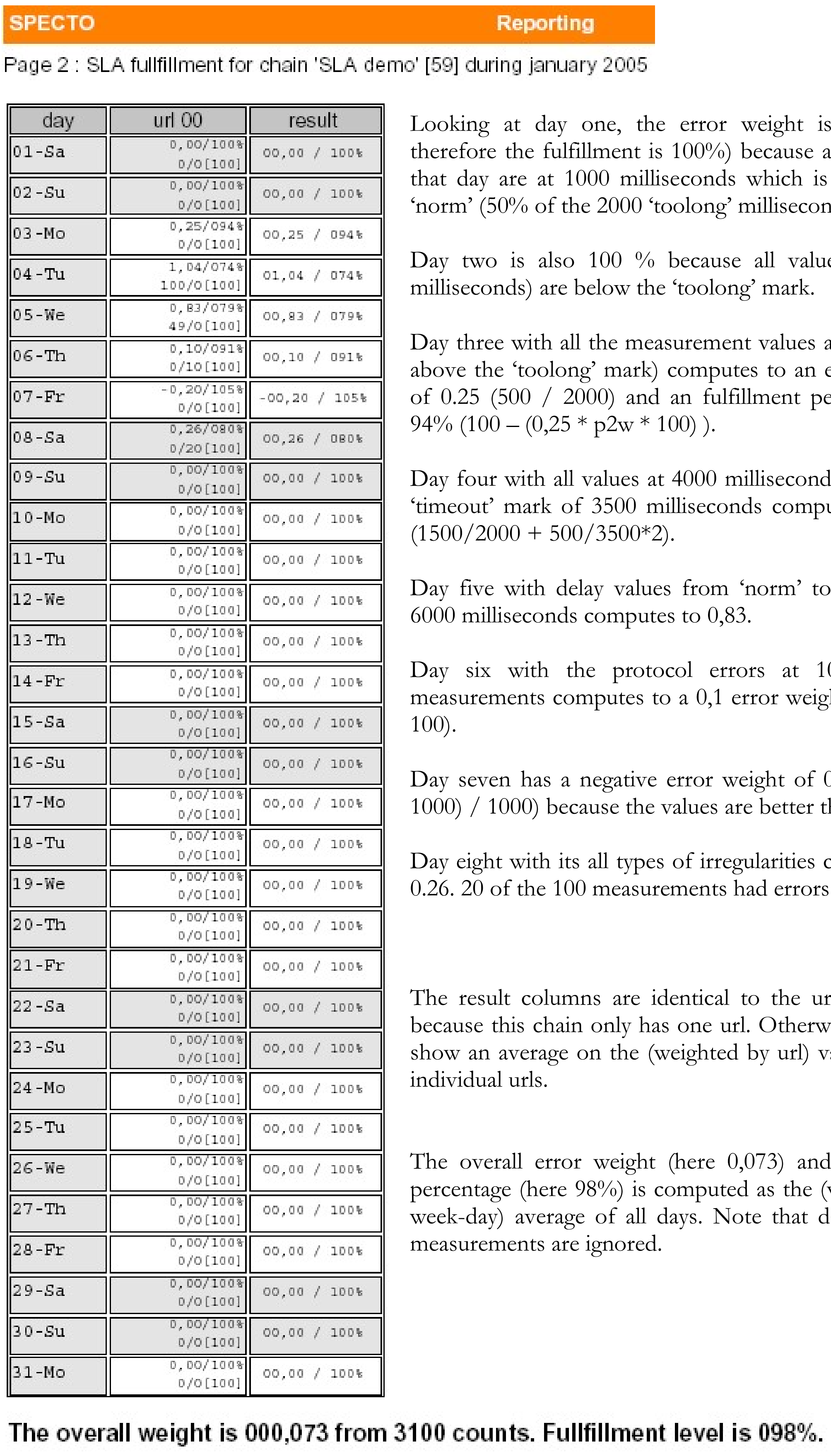
The PDF representation of the SLA report will be created.



PDF generated; type = 10/0, 1 elements back, for chain 59

Follow the 'View result as PDF' link; the PDF will display in the browsers PDF plug-in. The report consists of two pages; the first page displaying the chain's master data, the summary graphics and the SLA comments.

The SLA weight and percentage computation is shown on the second page:



SPECTO Reporting

Page 2 : SLA fullfillment for chain 'SLA demo' [59] during january 2005

day	url 00	result
01-Sa	0,00/100% 0/0[100]	00,00 / 100%
02-Su	0,00/100% 0/0[100]	00,00 / 100%
03-Mo	0,25/094% 0/0[100]	00,25 / 094%
04-Tu	1,04/074% 100/0[100]	01,04 / 074%
05-We	0,83/079% 49/0[100]	00,83 / 079%
06-Th	0,10/091% 0/10[100]	00,10 / 091%
07-Fr	-0,20/105% 0/0[100]	-00,20 / 105%
08-Sa	0,26/080% 0/20[100]	00,26 / 080%
09-Su	0,00/100% 0/0[100]	00,00 / 100%
10-Mo	0,00/100% 0/0[100]	00,00 / 100%
11-Tu	0,00/100% 0/0[100]	00,00 / 100%
12-We	0,00/100% 0/0[100]	00,00 / 100%
13-Th	0,00/100% 0/0[100]	00,00 / 100%
14-Fr	0,00/100% 0/0[100]	00,00 / 100%
15-Sa	0,00/100% 0/0[100]	00,00 / 100%
16-Su	0,00/100% 0/0[100]	00,00 / 100%
17-Mo	0,00/100% 0/0[100]	00,00 / 100%
18-Tu	0,00/100% 0/0[100]	00,00 / 100%
19-We	0,00/100% 0/0[100]	00,00 / 100%
20-Th	0,00/100% 0/0[100]	00,00 / 100%
21-Fr	0,00/100% 0/0[100]	00,00 / 100%
22-Sa	0,00/100% 0/0[100]	00,00 / 100%
23-Su	0,00/100% 0/0[100]	00,00 / 100%
24-Mo	0,00/100% 0/0[100]	00,00 / 100%
25-Tu	0,00/100% 0/0[100]	00,00 / 100%
26-We	0,00/100% 0/0[100]	00,00 / 100%
27-Th	0,00/100% 0/0[100]	00,00 / 100%
28-Fr	0,00/100% 0/0[100]	00,00 / 100%
29-Sa	0,00/100% 0/0[100]	00,00 / 100%
30-Su	0,00/100% 0/0[100]	00,00 / 100%
31-Mo	0,00/100% 0/0[100]	00,00 / 100%

The overall weight is 000,073 from 3100 counts. Fullfillment level is 098%.

Looking at day one, the error weight is 0,00 (and therefore the fulfillment is 100%) because all entries of that day are at 1000 milliseconds which is exactly the 'norm' (50% of the 2000 'toolong' milliseconds).

Day two is also 100 % because all values (at 1800 milliseconds) are below the 'toolong' mark.

Day three with all the measurement values at 2500 (500 above the 'toolong' mark) computes to an error weight of 0.25 (500 / 2000) and an fulfillment percentage of 94% (100 – (0,25 * p2w * 100)).

Day four with all values at 4000 milliseconds above the 'timeout' mark of 3500 milliseconds computes to 1,04 (1500/2000 + 500/3500*2).

Day five with delay values from 'norm' to just below 6000 milliseconds computes to 0,83.

Day six with the protocol errors at 10% of the measurements computes to a 0,1 error weight (10 * 1 / 100).

Day seven has a negative error weight of 0,20 ((800 – 1000) / 1000) because the values are better than 'norm'.

Day eight with its all types of irregularities computes to 0.26. 20 of the 100 measurements had errors.

The result columns are identical to the url 0 column because this chain only has one url. Otherwise it would show an average on the (weighted by url) values of the individual urls.

The overall error weight (here 0,073) and fulfillment percentage (here 98%) is computed as the (weighted by week-day) average of all days. Note that days without measurements are ignored.

Fine tuning the report by modifying the SLA definition :

The above configuration only uses weights of 1.0 and the standard algorithm. The following configurations are common with SLA definitions :

Fine-tuning weights

Above example uses equal weights of 1.0 for every weighting. SPECTO standard is a reduced weight for weekends.

Action: Apply 0.5 for saturday and Sunday and rerun the report. The error weight on day 02 will decrease because of the reduced weight of that day.

Specifying another 'norm' level

Using attribute '`sla.cust.normprc`' the 'norm' level is specified as a percentage of the url's 'toolong' lower limit. The default is 50%; corresponding to an '50' entry.

Action: Customize 75 and rerun the SLA report: The error weights will decrease because the norm is now set to 1500 milliseconds instead of 1000 milliseconds as before.

Specifying another algorithm

The weight computation described above is the standard algorithm ('***linear***', field 'algorithm type'). The following other algorithms are available :

'***step***': The delay-time weights are not computed linear within the ranges (below 'norm', between 'norm and 'toolong', between 'toolong' and 'timeout' and above 'timeout) but are assigned the values -0.2, 0, 0.2, 0.4 (customizable).

'***script***': During the computation an user defined script is called which computes the weights with a custom algorithm

SPECTO script

Overview

SPECTO script is a small scripting language used in several places of the SPECTO engine to automate processes. Especially, scripts are used to specify the before/after processing of URLs ('pbu'/'pau'), chains ('pbc'/'pac'), threads ('pbt') and notifications ('PBN').

Also SPECTO scripts can be defined in text areas (see chapter 'working with text') and executed from there using the 'es <script_name>' command.

Syntax

A script consists of one or multiple commands; each terminated with a semi colon (';'). Any command is a simple command, an assignment or a conditional command.
Simple command are of the form <command> { <parameter> }
Commands which return values are of the form <variable> '=' <command> { <parameter> }
Assignments are of the form <variable> '=' <parameter> { <opcode> <parameter> }

Parameters are constants, enclosed in "" characters, or variables (global variables have to be prefixed with an underscore '_'). A variable name prefixed with '&' is considered a pointer to another variable.

Tracing

Several levels of tracing may be activated using trace keywords.

Trace keyword	function
Tron	display the current command.
TronT	as above, including parameters
TronV	displays assignments to variables
TronT	displays the parsed tokens
Troff	turn off all traces

Control structures

Conditional execution

'if', 'else', 'endif' are used to separate blocks and execute them dependent from a condition. Conditions are specified as two parameters and an comparison operator.

```
If <par1> <op> <par2> ;
  commands…
{ else ;
```

```
  commands…
}
endif ;
```

Supported comparison operators :

'eq' / 'ne'	equal / not equal
'lt' / 'le'	lower than / lower or equal than
'gt' / 'ge'	greater than / greater or equal than

The comparison operator works numerical if possible (both parameters are numeric) or alphanumerical.

Repeated execution

All statements within a 'Loop' 'EndLoop' Block are executed continuous until a 'break' command is issued.

```
loop;
  commands…
  if <par1> <op> <par2> break;
  commands…
endloop;
```

Functions

Functions can be defined using the 'function' and 'endfunction' keywords. Parameters are defined with their formal names following the 'function' keyword; a result may be returned using the 'return' keyword'.

Example :

```
// function demo

function myFkt par1 par2;
   debug "inside myFKT";
   debug "first parameter : " par1;
   debug "second parameter: " par2;
   return par1 & "+" & par2;
endfunction;

// main program
debug "start" " of prog";
varf   = "bbb";
result = myFkt "aaa" varf;
debug "This Result:" result;
```

Includes

Script modules can include other script modules using the 'include' keyword. It is advised (so currently not inhibited) to use included scripts only for the definition of functions (as libraries) and not to execute code within them.

Example :

```
// include demo

include "fktlib1";

// main program
// function extFkt is defined in script fktlib1;
extFkt "ccc" "ddd";
```

Available commands

A command is the basic execution unit of a SPECTO script. Commands may return values which can be assigned to variables. Also, after the execution of a command the predefined variable 'subrc' reflects the execution status. An execution status of 0 denotes that the execution performed without errors. Command names are case insensitive.

log { <*value*> }	outputs a console log message.
debug { <*value*> }	outputs a console message during URL 'one run' processing.
<*var*> = ***attribute*** <*name*>	reads a SPECTO attribute. For security reasons a '_' prefix is set before the attribute name (this prevents script commands from reading standard customizing attributes.
<*var*> = ***var*** { <*value*> }	concatenate the values and treat this as a variable name; returns the content of this variable.
<*var*> = ***perform*** <*value*>	execute the script stored in text area <value>.
<*var*> = ***inline*** <*value*>	execute the script specified in <value>.
<*var*> = ***call*** <*class:method:param*>	Call an external java class. (see also chapter 'customer exits').
<*var*> = ***random*** <*maximum*>	Compute a random number within 0 and <maximum>.
<*var*> = ***getSessionId***	Get the session id of the execution unit. Empty if not attached to a session.
<*var*> = ***makeLink*** <*command*>	Construct a HTML link ('anchor').
<*var*> = ***getTimeMs***	Get number of milliseconds since 1.1.1970.

<*var*> = ***getDate*** <*dateformat*> <*timeformat*> <*language*> <*country*>

Get a formatted date time. <dateformat> and <timeformat> are of 'none', 'short', 'medium', 'long' and 'full'; <language> and <country> are according the ISO formats.
Example: GetDate "short" "none" "en" "us"

set *<parameter> <value>*	Set a property (see chapter 'URL configuration' for available values)
<var> = ***get*** *<parameter>*	Read a properties value.
properties	List all properties (see also 'Set' command) and their current values (for debugging purposes).
http *<name> <value>*	Add an URL HTTP Request header
<var> = ***httpGet*** *<field>*	Get the value of the HTTP response field <field> into variable <variable>.
<var> = ***httpGetAll***	Get all HTTP response fields.
nextURL *<chain> <URL>*	Override the next chain/URL to process. If <chain> is below 0, the current chain is used.
notify *<value1> <value2> <set> <timeout> <toolong> <content><custom> <waitloop>*	Create a notification event if <value1> is larger than <value2>. If <set> is 'true' the notification is increased, otherwise decreased.
notify_clear *<noti.> <msg.>*	clears all notification stati. If 'noti.' is 'true' also active notifications are deleted; if <msg.> is 'true' a release message will be send.
<var> = ***len*** *<string>*	Compute the length of <string>.
<var> = ***trim$*** *<string>*	remove leading and trailing blanks of <string>.
<var> = ***mid$*** *<string> <from> <to>*	Compute a substring of <string> starting at position <from> and ending before position <to>. The index values are zero based. Example: *mid$ "ABCD" 1 3* will return *'BC'.*
<var> = ***left$*** *<string> <len>*	Compute the left part of <string>. Example: *left$ "ABCD" 2* will return 'A*B'.*
<var> = ***right$*** *<string> <len>*	Compute the right part of <string>. Example: *right$ "ABCD" 1* will return *'D'.*
<var> = ***find$*** *<string> <string2> [<starting> [<ending>]]*	Find the position of <string2 in <string>; -1 if not found. This function is case sensitive. The result is zero based and indicates

the position from the beginning of the string (not from the <starting> position).
Example: *find$ "ABCD" "C"* will return 2.
Example: *find$ "ABCD" "c"* will return -1.
Example: *find$ "ABCD" "C"* 3 will return -1.

<var> = ***between$*** *<string> <string1> <string2> [<starting>]*
Extract the substring between <string1> and <string2> of <string>. This function is case sensitive.
Example: *between$ "ABCD" "B" "D"* will return *'C'*.

<var> = ***replace$*** *<string> <string1> <string2> [<with> [<starting]]*
Replace the substring between <string1> and <string2> of <string> with <with> starting at position <starting>. This function is case sensitive.
Example: *replace$ "ABCD" "B" "D" "xxx"* will return *'ABxxxD'*.

<var> = ***replaceI$*** *<string> <with> <starting> [<ending>]*
Replace the substring between position <starting> and position <ending> of <string> with <with>.
Example: *replaceI$ "ABCD" 1 3 "xxx"* will return *'AxxxD'*.

<var> = ***replaceTag$*** *<string> <tag> <with> [<starting> <ending>]*
Replace all occurrences of <tag> in <string> with <with>.
Example: *replaceT$ "ABCB" "B" "xx"* will return *'AxxCxx'*.

<var> = Field <array> <index> Extract the <index> element of an array <array> of elements. The separator of the elements is the first character in <array>. The index is zero based.

Delay <milliseconds> Wait (at least) for the specified number of milliseconds.

Exit Terminate script. (Scripts automatically terminate at the end of the script).

Quit Terminate script. (see also command 'Exit').

Return { <par> } Terminate script and return the concatenated list of parameters. The return value of a PBU script is used as a command instead of the URL defined in this page.
Example : 'return 'quit';' would stop the chain processing.

Script objects

Script supports various objects. Their members are accessed using standard 'dot' convention.

Members of object 'specto.client *[<member>]* ***' :***
name
id
username
password
numchains

Members of object 'specto.chain *<chain> [<member>]'* ***:***
name
id
type
sequence
flags
period
numurls
definednotifications
activenotifications

Members of object 'specto.chain.url *<chain> <url> [<member>]'* ***:***
url
symbolicName
id
type
sequence
flags
waitBetweenUrls
sessionId
tooLong
timeOut
numPars
numContents

Members of object 'specto.chain.notification *<chain> <notification> [<member>]'* ***:***
address
type
message
source
deltaMinutes

Members of object 'specto.thread *<thread> [<member>]* ***' :***
name
type
chain
position
status

Members of object 'specto.notification *[<member>]* ***:***

address	
type	
status	(0=first, 1=reminder, 2=release)
subject	

source
typeError
message
deltaMinutes
level

notes:
Setting ‘type’ to “” disables execution of the notification.

Members of object ‘document’ :

get <document>	read document content
add <document> <value>	store <value> into document <document>.

Members of object ‘file’ :

append <filename> <value>	append <value> to file <filename>.
read <filename>	return content of file <filename>.
remove <filename>	remove file <filename>.
list <mask>	return a list of files matching <mask>.

Members of object ‘mail’ :

send <to> <subject> <message> [<attachment>]
send an email message.

Members of object ‘’response’ :

resultContentCheck	get/set ‘content check’ value (from measurement).
resultContentCheckBad	get/set ‘content check bad’ value (from measurement).
timePassed	get/set time passed value (from measurement).
responseCode	get/set ‘response code’ value (from measurement).
URLCheck	get/set ‘URLCheck’ value (from content check).

Members of object ‘results’ :

toFile <chain> <daysBack> [<beginhour> [<endhour>]]
Generate a result file. Returns the filename.
toGraphic <chain> <daysBack> [[[<beginhour> [<endhour> [<limit>]]]
Generate a graphical result file. Returns the filename.
toOverview <type> <chain> <daysBack> [<limit>]
Generate a graphical overview file. Returns the filename.
Type : 0=monthly, 1=weekly, 2=daily.
toHTML <chain> <daysBack> [<beginhour> [<endhour>]
Generate a result in HTML format.
toPDF <type> <subType> <chain> <elements_back> [<limit>]
Generate a result in PDF format. Returns the filename.

Members of object ‘cookie’ :

get <name> <path>	The cookie’s value.
getPath <name> <path>	The cookie’s path.
getDomain <name> <path>	The cookie’s domain.
getExpires <name> <path>	The cookie’s expiration date/time.
getAll <name> <path>	The cookie’s combined information.

Members of object 'xml' :

Attribute.get <text> <taglist>	Get an XML attribute.
Attribute.set <text> <taglist> <value>	Set an XML attribute.
Attribute.add <text> <taglist> <value>	Add an XML attribute.
Attribute.remove <text> <taglist> <value>	Remove an XML attribute.
Element.get <text> <taglist>	Get an XML element.
Element.set <text> <taglist> <value>	Set an XML element.
Element.remove <text> <taglist> <value>	Remove an XML element.

Special commands :

R3 'spec' 'input_parameters' 'output_parameters' with <spec> of the form : 'hostname:system:userid:password:client:language'

Example 1 (overview)

Reading attribute '_order' :

```
order = attribute "order";
```

Setting a HTTP header during an 'PBU' event :

```
http "Accept-Language" "en" ;
```

The same using a global variable :

```
_language = "en";
http "Accept-Language" _language;
```

Initializing global numerical variable '_bcount' to 1 when a chain is started for the first time:

```
if _firstRun eq "";
   _firstRun = "no";
   _bcount = 1;
endif;
```

Extract the first half of a string :

```
text = "abcdefgh";
lll = len text;
lll = lll / 2;
htext = mid$ text 0 lll;
log lll " - " htext;
```

Indirection (a kind of pointer mechanism), variant 1 :

```
pv1 = "_firstRun";
log &pv1;
```

Indirection (a kind of pointer mechanism), variant 2 :

```
pv2 = var "_first" "Run" "4";
log pv2;
```

loop over an array content :

```
index = 0;
loop;
   el = field ";first;second;last", index;
   ll = len el;
   if len eq 0; break; endif;
   index = index + 1;
endloop;
```

Example 2 (language features)

```
/*
 demonstration of SPECTO scripting capabilities
 rel. 1.0
*/

// output
debug "Output to debugging screen";
log   "Output to console and into Log table";

// variables and assignments
varInt = -1204;
varString1 = "Test";
varString2 = "Test\u0022Test";
debug varInt varString1 varString2;

// arithmetic operations
sum = 3 + 4 * 5;
debug "sum 1 : " sum;
sum = 1 + sum / 2;
debug "sum 2 : " sum;

// String operations
xxx = " to ";
xxxt1 = trim$ xxx;
xxxt2 = trim$ " to ";
debug "*" xxx "*" xxxt1 "*" xxxt2 "*";
s = "from the begin" & xxx & "the end";
sl = left$ s 2;
sm = mid$ s 6 1;
sr = right$ s 1;
sall = sl & sm & sr;
debug "PartStrings: " sl "," sm "," sr;
debug "Together   : " sall;
found1 = find$ s " to ";
found2 = find$ s "notthere";
debug "Founds at " found1 " and " found2;

// file operations
filename = "test.txt";
debug filename;
file.remove filename;
file.append "test.txt" "XXX";
r = file.append "test.txt" "XXX";
debug r;
file.append filename "XXX" vvv;
r = file.append filename "XXX" vvv;
debug r;
file.append filename "Robert" "Krämer" "\r\n";
r = file.append filename "line 2" "\r\n";
debug r;
file.append filename "last line" "\r\n";
ff = file.list ".";
debug ff;
ft = file.trew;
fi = file.read filename;
debug fi;

// control structures - conditional execution
if sum eq 18;
debug "sum is okay";
else;
debug "sum is not okay: " sum;
endif;
```

```
// control structures - loops
i = 1;
loop;
   i = i + 1;
   if i gt 3;
      break;
   endif;
   debug "loop: " i;
endloop;

z1 = specto.chain currChain "name";
z2 = specto.url currChain currURL "symbolicName";

set notificationEnable true;
strInline = "set notificationEnable false;";
debug strInline;
x = inline strInline;
debug x;
strSet = Get "notificationEnable";
debug strSet;

x = inline "set notificationEnable false;";
debug x
strSet = Get "notificationEnable";
debug strSet;
```

Example 3 (XML)

SPECTO Script has a XML library, allowing to access and modify attributes and content of selected elements of a XML document.

```
// an example of XML processing caps.
//

// get from a persistent document
nXML = "\lxmldemo\l";

// read/write attribute
a1 = xml.attribute.get nXML "id";
xml.attribute.set nXML "id" "2";

// read/write tag
t1 = xml.element.add nXML "value" "124";
```

Example 4 (notification)

```
// script=ss
// reading and modifying the current notification
// used in a PBN script

// read an attribute of this notification
a = specto.notification "address";
log "address is : " m;

// set an attribute of this notification
specto.notification "message" "modified";
```

JavaScript

Overview

As an alternative to SPECTO Script JavaScript (ECMA script) can be used wherever SPECTO script is possible.

Features are :

- Compliant to JavaScript 1.5
- Usable wherever SPECTO script may be used
- Full access to SPECTO object-hierarchy

Selection of scripting engine

By default SPECTO uses SPECTO script.
This default may be changed by setting the (client dependent) attribute 'DefaultScriptEngine' to 'js' for JavaScript or 'ss' for SPECTOScript.

Using the 'es' command to execute a script via command line or the batch processor, the variants 'esj' or 'ess' start scripts as JavaScript or SPECTOScript respectively, overriding the default or the setting of attribute 'DefaultScriptEngine'.

In the first line of a script the tag '//script=js' or '//script=ss' may be used to define the scripting engine for that script, overriding any other configuration.

Syntax of JavaScript

The SPECTO JavaScript implementation fully adheres to JavaScript 1.5. Therefore any book about JavaScript can be used as a reference or tutorial.
Note that this is a 'server side' implementation of JavaScript; common objects of 'client side' implementations in web browsers like 'screen', 'window' or 'frame' are not available here.

Returning results

A JavaScript program may return results to the SPECTO engine by terminating with any JavaScript expression. E.g. the script '4 + 5;' returns '9'. If a JavaScript is executed direct via the 'es' command, the result of the script is output to the screen; if a JavaScript is called via another JavaScript, the result is stored in the result variable (if specified) of the calling script.

Using the SPECTO object hierarchy

A JavaScript program has access to selected internal objects and methods of the SPECTO engine by several predefined objects in the JavaScript engine.

This SPECTO object hierarchy is the same hierarchy used for accessing SPECTO objects via external JAVA applications. The complete reference to the object hierarchy is made available as a JavaDoc file.
Also JavaScript's reflection methods may be used to find out more about SPECTO objects (see example 5).

Available SPECTO objects

A JavaScript program has access to the internals of the SPECTO engine by several SPECTO objects.

'specto'	predefined object; includes the chains and URLs hierarchy of the current client (see example 1) and the execution engine.
'script'	Predefine object; allows execution of other scripts.

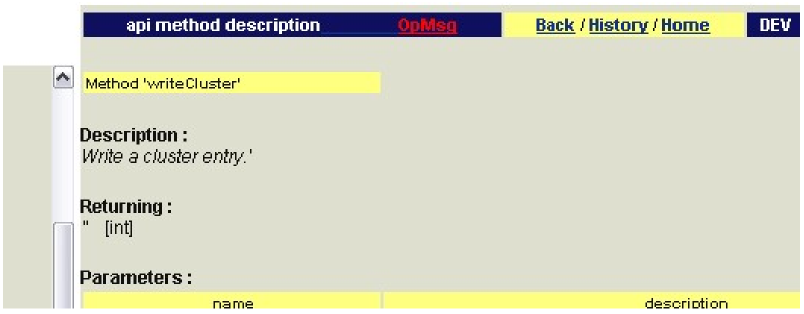
All other objects described below are dynamically created and need not be defined in the JavaScript program.

The methods and structures (=classes) of the SPECTO engine available for javascript programs are available via the '`api`' and '`struct`' commands (menu: 'development').

The following list is only an excerpt of the available commands.

A list of methods is displayed by '`api list`'; any method is described by the '`api describe` <method>' command.

Example (command '`api describe writeCluster`'):



Method 'writeCluster'

Description :
Write a cluster entry.'

Returning :
" [int]

Parameters :

name	description

Where appropriate, it is possible to navigate to the structure definition from a method's parameter list.

List of Properties and Methods of object 'specto' :

Void ***message***(text)
 Output a message to the screen (if executed online) or to the log.

Void ***log***(text)
 Output a message to the log.

Void ***getSessionId***()
 Get the session id of the current selection (empty if no session connected to the scripts execution unit (e.g. batch scripts or pb/pa scripts)).

Void ***makeLink***(command)
 Construct a HTML link (an anchor) to the specified SPECTO engine command. Gets the session information from '*getSessionId()*'.

AttributesChain[] ***getChains***()
 Returns a collection of the chains in the current client. Any chain is represented as an 'AttributesChain' object.

Void ***setUpdate***()
 Mark the current client to be scheduled for update to the database. To be used after changes have been made to the chain/URL object hierarchy.

String ***getAttribute***(attribute_name)
 Return the value of a SPECTO attribute. The attributes name is automatically prefixed with an underscore ('_') to prevent (for security reasons) access to normal SPECTO attributes.

Void ***setAttribute***(attribute_name, attribute_value);
 Set the value of a SPECTO attribute. The attributes name is automatically prefixed with an underscore ('_') to distinguish it from the normal SPECTO attributes.

String ***getVariable***(variable_name);
 Return the value of a SPECTO local or global variable.

Void ***setVariable***(variable_name, variable_value);
 Set a SPECTO local or global variable.

String ***getProperty***(property_name);
 Get the property value of an executing URL. Only makes sense for 'process...' scripts.

Void ***setProperty***(property_name, property_value);
 Set the property value of an executing URL. Only makes sense for 'process...' scripts.

AddressEntry[] ***getNotifications***(chain_id)
 Return a collection of defined notifications of the specified chain.

NotificationEntry[] ***getActiveNotifications***(chain_id)
 Return a collection of currently active notifications of the specified chain.

NotificationEntry ***getNotification***()
Return the current notification. To be used inside a PBN script.
Setting '.typeNotification' to "" disables execution of the notification.

Boolean ***startThread***(chainId)
Start one instance of the specified chain.

Boolean ***stopThreads***(chainId)
Stop all instances of the specified chain.

Boolean ***stopAllThreads***()
Stop all chain-instances of this client.

ThreadEntry[] ***getThreads***(chainId)
Get a collection of all chain-instances of the specified chain.

ThreadEntry[] ***getAllThreads***()
Get a collection of all chain-instances of this client.

Boolean ***sendMail***(address, title, message,[, attachment])
Send an email.

String ***html*** (format [, convert_flag])
Create an html format (see examples 6 and 7).
The following are valid tags for *format* : normal, bold, italic, bold-italic,blink, newline, small, medium, large, space, tableon, tableoff, tron, troff, tdon, tdoff, gt, lt.

The optional *convert_flag* must be set to *false* if used for a console script (see example 7).

String ***reportPDF(type, subtype, chained, elementsBack, limit)***
String ***reportPDF(type, subtype, chained, elementsBack)***
Generates a PDF formatted report. Returns the filename.

String ***readFile***(file_name);
Get the content of a file on the local machine.
Returns null if an error occurred.

String ***writeFile***(file_name, content);
Write a file on the local machine.
Returns null if the write was successful, an error description otherwise.

String ***deleteFile***(file_name);
Delete a file on the local machine.
Returns null if the delete was successful, an error description otherwise.

String ***execCommand***(command)
Execute a SPECTO command.

GregorianCalendar ***getGregorianCalendar***()

Return a new instance of a (Java) GregorianCalendar.

String ***getLatestError***();
Returns a textual description of the latest error.

Properties and Methods of object 'script' :

String ***exec***(script);
Execute a JavaScript document.

Properties and Methods of object 'AttributesChain' :

Id	Id
sequence	Sequence number within chain
type	Type
name	Name
flags	Set of flags, coded as an integer
period	period of execution (in seconds)
timeout	'timeout' limit (in milliseconds)
toolong	'too long' limit (in milliseconds)
lastReportingUpdate	Time of last entry into reporting

AttributesURL[] ***getURLs***()
Returns a collection of URLs of this chain.

Properties and Methods of object 'AttributesURL :

Id	Id
sequence	Sequence number within chain
waitBetweenURLs	Time to wait before execution of next URL
type	Type
symbolicName	Symbolic name
sURL	URL
sessionId	Session Id
flags	Set of flags, coded as an integer
timeout	'timeout' limit (in milliseconds)
toolong	'too long' limit (in milliseconds)

Properties and Methods of object 'AddressEntry' :

typeNotification	type of notification
address	Address
message	Message to be included
flags	Set of flags, coded as an integer
source	the initiating client/chain
deltaMinutes	time between executions

level	level of invocation

Properties and Methods of object 'NotificationEntry' :

typeNotification	type of notification
chainId	id of the associated chain
URLId	id of the URL triggering the notification
status	(0 = first, 1 = reminder, 2 = release)
adress	Address
subject	subject (e.g. of email)
message	Message to be included
flags	Set of flags, coded as an integer
source	the initiating client/chain
deltaMinutes	time between executions
level	level of invocation
startTimestamp	Time of first execution of this notification
nextNotification	Time of next execution of this notification

Properties and Methods of object 'ThreadEntry' :

clientId	Id
chainId	Sequence number within chain
name	Name

Integer ***getLastResult***()
Returns the latest result code.

String ***getLastResponse***()
Returns the Timestamp of the last execution.

The SPECTO JavaScript XML object :

SPECTO provides several ways to work with XML formatted objects:

- The SAX Parser
- The SPECTO XML library

The SPECTO XML library

The SPECTO XML library is available for scripts written in JavaScript and for exit implementations written in Java.

xml ***getXMLObject*** (String_source);

The provided xml-formatted text ('source') is stored within a new created xml object. No validation is performed on 'source'.

SPECTO engine objects

The complete SPECTO engine modules are also available via the SPECTO java script implementation. Usage of this APIs requires specific knowledge of the internal structure of SPECTO and is not recommended without assistance by NLS.

Class of returned instance	method	description
SpectoBackGround[]	getBkgEntries ()	
SpectoBatch	getBatchInstance ()	
SpectoDelayedWrite	getDelayedWriteInstance ()	
SpectoHTTPServer	getHTTPServerInstance ()	
SpectoLocal	getLocalInstance ()	
SpectoLogging	getLoggingInstance ()	
SpectoMail	getMailInstance ()	
SpectoMaster	getMasterInstance ()	
SpectoMonitor	getMonitorInstance ()	
SpectoNameService	getNameServiceInstance ()	
SpectoNet	getNetInstance ()	
SpectoNotification	getNotificationInstance ()	
SpectoPersistence	getPersistenceInstance ()	
SpectoReporting	getReportingInstance ()	
SpectoServices	getServicesInstance ()	
SpectoUsers	getUsersInstance ()	

Example

The following example lists some details of the currently active monitors (information similar to the command 'mo l') :

```
// script=js

// list active monitor entries

var monitor = specto.getMonitorInstance();
var entries = monitor.getMonitorEntries();

  for(i=0; i<entries.length; i++) {
  var entry = entries[i];

    if (entry)
      specto.message("Pos " + i + ": thread=" +
        entry.clientId + "/" + entry.chainId + " : " + entry.status);
  }
```

Example 1 (accessing SPECTO objects: chain/url hierarchy)

The following example outputs some information about the chains and their URLs of the current client to the screen. Also it increases the value of the 'tooLong' parameter of every url by one and saves enables the changes to be saved back to the database.

```
//script=js
//example for access of SPECTO objects

specto.log("JavaScript Demo: js1");
specto.message("\n\nThe chains in this client (\"" +
   specto.getClientName() + "\") :");
var chains = specto.getChains();
for (i in chains) {
   specto.message("[" +
      chains[i].Id + "] " +
      chains[i].name + " (" +
      chains[i].numURLs + " URLs)");
   var urls = chains[i].getURLs();
   for(u in urls) {
      specto.message("- " +
         ((urls[u].symbolicName == "")
            ? urls[u].sURL : "[" + urls[u].symbolicName + "]"));
      urls[u].toolong += 1;
   }
}
specto.setUpdate();
```

Example 2 (accessing SPECTO objects: execution)

The following example shows access to the execution methods of the 'specto' object.

```
//script=js
//example of chain execution
var thisChain = 10;

function listThreads(chain) {
var threads = specto.getThreads(chain);
   specto.message("\nActive threads:");
   for(it in threads)
      specto.message(threads[it].name +
         " [" + threads[it].clientId + "]");
}

// main
listThreads(thisChain);
specto.startThread(thisChain);
listThreads(thisChain);
```

Example 3 (accessing SPECTO objects: others)

The following example shows access to several other parts of the 'specto' object.

```
//script=js
//example for access to SPECTO objects
var thisChain = 10;
```

```
specto.message("\nDefined notifications:");
var adrs = specto.getNotifications(thisChain);
for(iadrs in adrs)
   specto.message(adrs[iadrs].adress +
           " / " + adrs[iadrs].deltaMinutes);

specto.message("\nActive notifications:");
var notifs = specto.getActiveNotifications(thisChain);
if (notifs.length <= 0)
   specto.message("no active notifications " +
      "for chain " + thisChain);
else
   for(inotifs in notifs)
      specto.message(adrs[iadrs].adress +
           " / " + adrs[iadrs].message);

specto.message("Calling javascript \"js1\" :");
specto.message(script.exec("js1"));

//returning a message
"Script \"js2\" terminated successfully";
```

Example 4 (documents)

The following example shows how to access (read & store) SPECTO documents (use SPECTO command ‘tal’ to get an overview on available documents.

```
//script=js
//example for reading/storing of documents
var doc   = "testdoc1";
var itext = "doc to save";

specto.message("Saving document : " + doc);
var lId   = specto.storeDoc(doc, itext);
specto.message("used/generated link Id : " + lId);

specto.message("\nReading document: " + doc);
var text = specto.readDoc(doc);
   specto.message(((text == null)
     ? "document " + doc + " not found"
     : "content of " + doc + " :\n" + text)
   + "\n");
```

Example 5 (reflection of SPECTO objects)

The following example shows how to use JavaScript reflection to get information about SPECTO objects.

```
//script=js
//example of inspection of SPECTO objects
function members (obj) {
   for (m in obj) {
   var t = typeof obj[m];

     specto.message(t + " : " + m);
   }
}
```

```
specto.message("\nThe members in \"specto\" :");
members(specto);

specto.message("\nThe members in \"AttributesChain\" :");
members((specto.getChains())[0]);

specto.message("\nThe members in \"AttributesURL\" :");
members((((specto.getChains())[0]).getURLs())[0]);
```

Example 6 (formatting output)

The following example shows how to format output for programs intended to be run in the foreground using the ‘es’ command :

```
// script = js

var chain = 1;

specto.message(
   specto.html("italic") +
   "There are currently " +
   specto.html("bold-italic") +
   specto.getActiveNotifications(chain).length +
   specto.html("normal") +
   " notifications for chain " +
   chain + ".");
```

Example 7 (console plug-in)

The following example shows a plug-in for the master or client console. It has to be saved as ‘console_1’, ‘console_2’ or ‘console_3’. For this demo to work, the console has to be configured (‘mcc’ or ‘cvc’) to display the script in one frame and variable ‘_console1’ in another frame :

```
// script = js

// a demonstration for a console plug in

// setting one of the three console variables
var c1;
   if ((c1=specto.getVariable("_console1")) == "")
      c1 = 0;
   c1 = parseInt(c1);
   c1 += 1;
   specto.setVariable("_console1", c1);

// setting a HTML formatted result text
   specto.html("newline", false) +
   "current client is " +
   specto.html("bold-italic", false) +
   "\"" + specto.getClientName() + "\"" +
   specto.html("newline", false);
```

Example 8 (Low level system access)

Accessing the runtime environment of the SPECTO engine:.
These are potentially very dangerous functions.

```
// System properties
var pr = "user.country";
  specto.setSystemProperty(pr, "DE");
var pv = specto.getSystemProperty(pr);
  specto.message("Property " + pr + " : '" + pv + "'");

// System environment
var en = "PATH";
var ev = specto.getSystemEnvironment(en);
  specto.message("Environment " + en + " : '" + ev + "'")

// System runtime
var rt = specto.getRuntime();
var prog = "notepad.exe";
  //rt.exec(prog);
```

Example 9 (Session persistence)

There are two separate scripts who exchange data via the 'session persistence' object::

```
//
// Demo for session based persistence, part I
// Name : persist_1
//
var per = specto.getSessionPersistence();
var key = "abcde";

  if (per != null) {
    per.put(key, "xxx");
    specto.execScript("persist_2");
  }



//
// Demo for session based persistence, part II
// Name : persist_2
//
var per = specto.getSessionPersistence();
var key = "abcde";
var val;

  if (per != null) {
    val = per.get(key);
    specto.message(val);
  }
```

Example 10 (class/instance references)

Working with SPECTO JAVA classes and instances of JAVA classes: :

```
var clna = "java.util.GregorianCalendar";
var cl = specto.classForName(clna);
  specto.message("Class ref '" + clna + "' : " + cl);
var clin = specto.newInstance(cl);
  specto.message("Class ins '" + clin);

var greg = specto.getGregorianCalendar();
  specto.message("Greg.Cal.Inst. : " + greg);
var gregCl = specto.getClass(greg)
  specto.message("Greg.Cal.Class : " + gregCl);

  clna = "kernel.ExtendedCalendar";
var cl = specto.classForName(clna);
  specto.message("Class ref '" + clna + "' : " + cl + "'");
  specto.message("Class val '" + cl.getName() + "'");
  clin = specto.newInstance(cl);
  specto.message("Class val '" + clin.APRIL + "'");
  specto.message("Class val '" + clin.getWeekYear() + "'");
```

Example 11 (Working with JSON formatted data: Parser)

Parsing JSON formatted data into JavaScript objects:

```
// A function to parse JSON into a JS object using
// the SPECTO JSON preprocessor
function parseJSON (json) {
var jsonPrepped = specto.prepJSON(json);
  specto.message(jsonPrepped.length());
// Parse based on (clean ?) 'Function' approach
var obj = Function('return (' + jsonPrepped + ')')();
// Alternate (dirtier, 'eval()' based) version of parse
  //obj = eval("(" + jsonPrepped + ")");
  return obj;
} // parseJSON()


// Example run
//  specto.message("Source: " + jsonfile);
var obj = parseJSON(jsonfile);
  specto.message("Obj : " + obj + " / " + obj.name + " / " +
    obj.periodDuration + " / " + obj.order);

for (var prop in obj) {
  specto.message(prop);
}
```

Example 12 (Codings: BASE64, Deflate, Zip)

Decoding and encoding data using BASE64, Decode/Encode and ZIP.
Also shows determining and changing the character set:

```
var from = "aA./bb";

// Used character set
  specto.message("Charset  : " + specto.charsetS);
  //specto.charsetS = "ISO-8859-1";
  specto.message("Charset  : " + specto.charsetS);
  specto.message(" ");

// Base 64
var b64  = specto.btoa(from);
  specto.message("to B64   : " + b64);
  specto.message("from B64 : " + specto.atob(b64));
  specto.message("EQ : " +
    ""+specto.atob(specto.btoa(from)) == from);
  specto.message("TO : " +
    typeof (""+specto.atob(specto.btoa(from))) + typeof from);
  specto.message(" ");

// URL Encode/Decode
var enc  = specto.encodeURL(from);
  specto.message("to encoded   : " + enc);
  specto.message("from encoded : " + specto.decodeURL(enc));
  specto.message(" ");

// Deflate/ZIP
var deflated =
  "nZNNb9swDIb/iqC7vxO0EOIUaYJiAbrNS9wddpNluhVgSZ4op92/n+w4aQ5L" +
  "DrsYAvny6yG9ePhQLTmARWl0TpMwpgS0MLXUrzl9KZ+Ce/qwXCBXbdqxVe/e" +
  "9A5+94CO+ECN7OjJaW81MxwlMs0VIHOC7Vdfn1kaxqyzxhlhWkpWiGCdL7U2" +
  "GnsFdg/2IAW87J5z+uZchyyKKo5N6OWtrCz3DxVRsvEVpeZu7PIkbKAGO9rC" +
  "KUZFWtZdNDYVIRpKnowVMPad04a3CJRsNzkVTZUk84oHVVaJYDbzn/tZDUE1" +
  "v6vidF7PeMW9EguOKA/wGYvYw1aj49rlNI3TJEjiILkrk4xlKUvjMEuyX5QU" +
  "08iPUh9R3uJTHUXIvpRlERTf9yUlP08r8QI6LYCN1e0l+duJ+Qk3XV6Hu4gu" +
  "k593/c1n224K00rxh6za1ryvLXDnUTjbwwhWcXe9fhImo0XWQTNKWa+xAyEb" +
  "CTUl+2Io8KPn7WCwN7cfnZuabg/qcaP+iBx8OLI2quNW4oBLSS1Vr87ILoXr" +
  "1gPZQfNfAG/KBBNDbm8eDubd2Ho4ABC+09JyP7ex7sT5Xx0tJ+eVCT/dl3/g8i8=";

//  deflated = "S3TU009KAgA="; // = deflated(from)
  deflated =  specto.deflate(from);
  deflated += ""; // Convert from Object to String
  specto.message("deflated      : " + deflated);
  specto.message("deflated type : " + typeof deflated);
  specto.message("deflated len  : " + deflated.length);
  specto.message("from deflated : " + specto.inflate(deflated));
  specto.message(" ");
```

Example 13 (Working with OS files)

Finding and deleting files:

```
// retrieve a list of files from the
// engine or the web directory
// the pattern must be at the beginning of the filename
// or within the filename if '* is the first character
// true/false indicates whether engine or web directory

var files = specto.getFileNames("del", true);

  if (files) {
    specto.message("number found: " + files.length);
    for(var iF=0; iF<files.length; iF++) {
    var filename = files[iF];
      specto.message(filename);
    // optionally delete the file (only in engine dir)
      if (true) {
      var result = specto.deleteFile(filename);
        if (result)
          specto.message("Failed:" + result);
        else
          specto.message("Deletion of " + filename + " okay");
      }
    }
  }
  else
    specto.message("No matching files found");
```

Example 14 (Sending emails)

Sending an email to one or more recipients and optional attachments (file):

```
specto.sendMail( "specto@mathesis.de",
  "Title", "Body Line1\nLine2");

specto.sendMail( "specto@mathesis.de",
  "Title", "Body Line1\nLine2", "report.pdf");

specto.sendMails( [ "specto@mathesis.de", "specto@nls.de" ],
  "Title", "Body Line1\nLine2", "report.pdf");
```

Example 15 (Creation and Distribution of reports with attached PDF)

```
// generate a day-based summary report with a PDF attachment
// and email it to a list of recipients
//
// demo script - MATHESIS (c) 2003/11/02
//
// this script is intended to be started by the batch processor

// init
var report = specto.getReportingInstance();
var adrs   = new Array(
             "xxx@customer.de", "yyy@customer.de",
             "admin.specto@mathesis.de"
             );
var i;
var st     = " style='font-size:10pt; font-family:Tahoma'";
var tt     = "<table border=1>";
var ttc    = "</table>";
var tr     = "<tr>";
var trc    = "</tr>";
var td     = "<td" + st + ">";
var tdc    = "</td>";
var th     = "<th align='left'" + st + ">";
var thc    = "</th>";
var nl     = "<br>";
var now    = specto.getGregorianCalendar();
var text   =
    "<div" + st + ">" + nl +
    "<b><u>SPECTO Tagesbericht f?r " +
    specto.formatDate(now, "de", true, true) + "</u></b>" + nl + nl;
var status;

// within the current client, loop over the running chains
var threads = specto.getAllThreads();
  text += "<b>Laufende Monitore:</b>" + nl + nl;
  text += tt + tr + th + "Prozess" + thc + th + "Letzter Lauf" + thc +
trc;
  for (i=0; i<threads.length; i++) {
  var thread = threads[i].iThread;

    text += tr + td + thread.ac.name + " [" + thread.Name + "]" +
               tdc + td + specto.formatDate(thread.getLastResponse(),
               "de", true, true) + tdc + trc;
  }
  text += ttc;


// check open notifications
var notifs     = specto.getActiveNotifications(-1);

  text += nl + nl + "<b>";
  if (notifs.length == 0) {
    text += (status="Keine aktuellen St?rungen") + "</b>" + nl;
  }
  else {
    text += (status="Aktuelle St?rungen") + " :</b>" + nl + nl;
    text += tt + tr + th + "Prozess" + thc +
      th + "Stoerungsbeschreibung" + thc + th + "Nachricht" + thc +
      th + "seit" + thc + trc;
    for (i=0; i<notifs.length; i++) {
    var notif = notifs[i];
      text += tr + td + specto.getChains()[notif.chainId].name + tdc +
              td + notif.descriptionOfError + tdc +
              td + notif.message + tdc +
              td + specto.formatDate(notif.startTimestamp,
                        "de", true, true) + tdc + trc;
    }
    text += ttc;
  }

// check recent notifications
var rnotifs    = specto.getReportedNotifications(-1, 1);

  text += nl + nl + "<b>";
  if (rnotifs.length == 0) {
    text += "Keine St?rungen am Vortag" + "</b>" + nl;
  }
  else {
    text += "St?rungen am Vortag" + " :</b>" + nl + nl;
    text += tt + tr +
      th + "Zeitpunkt" + thc + th +"Prozess" + thc +
      th + "Status" + thc + th + "Adressat" + thc + trc;
    for (i=rnotifs.length; i>0;) {
    var rnotif = rnotifs[--i];
      text += tr + td + rnotif.timest.substring(11, 19) + tdc +
              td + specto.getChains()[rnotif.chainId].name + tdc +
              td + rnotif.status + tdc +
              td + rnotif.text + tdc + trc;
    }
    text += ttc;
  }


// additional remarks
  text += nl + nl +
    "<i>In Anlage eine graphische Darstellung des Verlaufs des " +
    "vergangenen Tags (PDF Format)</i>" + nl;

var mys = specto.getNetInstance();
  text += nl + "<hr><i>Erzeugt durch SPECTO Instanz : </i>" +
    mys.thisNetwork + "." + mys.thisNode + "." +
    specto.getClientName() + nl + nl;



//
// PDF generation
//
var filePDF = "attn2.pdf";
  now.add(now.DAY_OF_MONTH, -1);
var today   = specto.formatDate(now, "de", true, false);
var limit   = 5;
var leftM   = 36;

// additional footer (not the pdf page footer!)
function footer () {
  specto.writeAtPos(
    "Die Graphiken sind i.W. auf " + limit + " Sekunden skaliert. " +
    "Balken am oberen Bildrand kennzeichnen Bereiche mit Fehlern.",
    0, leftM, 70, 0);
  specto.writeAtPos("created by SPECTO", 0, 580, 70, 90);
}


// definition of automatic header and footer
var pe = specto.getPageEvent("image50.0;50.0;:logoCustomer.gif",
    "process monitoring", "customer.de");
  specto.setPageEvent(pe);

// open PDF stream
  specto.outputPDFFile(filePDF);
  specto.outputMode(false, false, true);

// title
  specto.format("size 12");
  specto.writeAtPos("Tagesverlauf customer.de' am " + today,
    0, leftM, 720, 0);
  specto.format("medium");

// loop thru the running threads
var currX = leftM, currY = 570;
  for (i=0; i<threads.length; i++) {
  var thread  = threads[i].iThread;
  if (!thread.ac) continue;
  var chainId = thread.ac.sequence;
  var climit = limit;
    if ((chainId == 5) || (chainId == 12))
      climit = 2 * limit;
    specto.message(i + " : " + thread.Name + " : " + thread.ac.name);
// prepare graphic description block
    gra = specto.getReportConfigEntry();
    gra.clientId     = specto.clientId;
    gra.chainId      = chainId;
    gra.daysBack     = 1;
    gra.sizeX        = 395;
    gra.sizeXLegend  = 100;
    gra.sizeY        = 200;
    gra.yMax         = limit + " Sekunden";
    gra.colorBkGnd   = 0x7f7f7f;
    gra.colorBkGndLegend = 0xff0000;
    gra.colorLines   = new Array(0x3f3f3f, 0);
    gra.arrowSize    = 2;
    gra.markerRadius = 1;
    gra.chartBorder  = 20;
    gra.chartBorderL = 25;
    gra.fontName     = "tahoma";
    gra.fontSize     = 9;
    gra.fontStyle    = 1;
    gra.footer       = thread.ac.name;
  // set position and scale of next graphic
    specto.reportGraPos(currX, currY, 2, 60, 60);
  // generate the graphic
    specto.reportGra(-1, 0, limit*1000, 0, 24, new Array(gra), false);
    specto.writeAtPos("Prozess '" + thread.ac.name + "' [" + chainId +
      "/" + thread.Name + "]", 0, currX, currY-10, 0);

  // list of recent notifications of this chain
    rnotifs    = specto.getReportedNotifications(chainId, 1);
    if (rnotifs.length <= 0)
      specto.writeAtPos("Keine St?rungsmeldungen",
        0, currX, currY-20, 0);
    else {
    var cno = 2;
      specto.format("size 8");
      for (var ino=rnotifs.length; ino>0;) {
      var rnotif = rnotifs[--ino];
      var text   = rnotif.type + " Fehler seit " +
          rnotif.timest.substring(11, 19) + " : " +
          rnotif.status + ", " + rnotif.text;
        if (text.length > 70)
           text = text.substring(0, 70);
        specto.writeAtPos(text, 0, currX, currY-(cno*10), 0);
        if (cno++ > 5) break;
      }
      specto.format("medium");
    }

  // go to next column, row, page
    if ((currX+=280) > 400) {
      currX = leftM;
      if ((currY-=200) < 100) {
        currY = 570;
        footer();
        specto.writePageBreak();
        specto.writeAtPos("Fortsetzung des " + today, 0,
               leftM, 720, 0);
      }
    }
  }
  footer();
  specto.outputMode(false, false, false);

// send the email with attached PDF
  if (true)
    specto.writeFile("c:\\t.html", text);
  else
    specto.sendMails(adrs, "SPECTO Status: " + status, text,
         filePDF);

// Return the generated text
   text;
```

Business-to-business ('b2b')

Overview

In addition to the common human-to-machine communication made possible by a browser displaying HTML formatted documents, the internet is also used for machine-to-machine communication exchanging business documents (mostly) formatted in an XML syntax. This process is normally fully automated.

Typical for the exchanged documents is that they are :
- Represented using UNICODE characters,
- formatted in extensible markup language ('XML'),
- structured according to an document type definition ('DTD') or a schema.
- include at least a sequence number and an transaction code

The transfer of B2B documents may happen using :
- HTTP ('www') protocol
- SMTP ('email') protocol

Above functionality is implemented in SPECTO.

Certain organizations and companies are working on standardization of structured document types for b2b exchange.
They are among others :
- Electronic business XML initiative ('ebXML') by UN/EDIFACT
- Common XML business library ('xCBL') by Commerce One Inc.
- ECo framework by Commerce.net

SPECTO will support above recommendations/implementations when they become significant.

Usage

SPECTO handles B2B requests like URLs. Instead of the URL a pair of address and DTD has to be applied, separated by a ';' character. For HTTP transport the address is an URL, for SMTP transport the address is an email address. The DTD may be specified by its name or id.

The creation and analysis of XML formatted documents is assisted by separate documents formally describing structure and possible content of the business documents. Those formal specifications are maintained using spcification languages like

- DTD ('document type definition')
- XML schema
- Proprietary specifications (Microsoft 'XDR', commerce one 'SOX', etc.)

Up to release 1.2 SPECTO's XML processing was based on specifying DTDs which then were transformed to XML formatted business documents at runtime using variable defined in the SPECTO URL specification. It was planned to replace/enhance the DTD mechanism by XML schema (which is generally considered to be the successor to DTD).

Since then our view towards using XML business documents has changed, in fact has been made a lot of easier, by not using the specification level any more but relying on parameterized XML skeletons of existing business documents which are value substituted at run time.
The advantages are :

- Allows easy usage of recorded or vendor supplied business documents samples
- Fits into SPECTO's existing parameter/variables/content-checking model
- No problems with inconsistent / non-existing schema definitions (…)
- Much easier to use

Therefore SPECTO does not support handling of schemas any more but will be supplied with a set of XML business samples of leading e-commerce players (Ariba's 'cXML', Commerce one's 'xCBL, UN/CEFACT 'ebXML', the BME's 'BMEcat').

In a future release Specto will be able to automatically import documents from an XML repository and will allow version and delta checking of schema files to detect changes which may have consequences on the skeleton business documents being used.

The mechanisms described here are also used for data formatting of SOAP messages. There the resulting XML document is the included into an (also XML) SOAP frame.

The b2b document templates available in SPECTO are separated into the groups :

- ***Transfers*** for documents used for specification of the transport of douments, SOAP is the most common example.
- ***Frameworks*** describing mechanisms to construct business documents.
- ***Functions*** business documents already specified for concrete business scenarios
- ***Verticals*** predefined documents for very specific applications.

Details

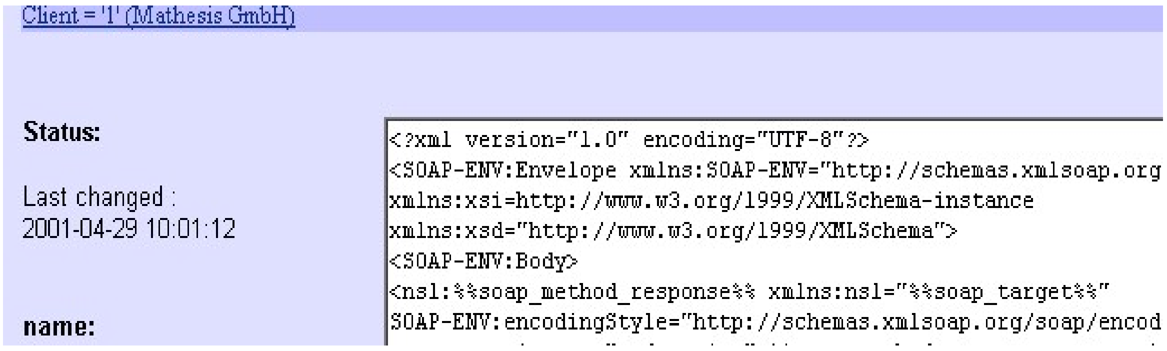
Transport: SOAP

SOAP does not specify any business content but the transport mechanism and therefore is usually used together with a b2b framework or function.
SOAP always formats the business content in XML (they have their own schemes and namespaces). Transport may be via HTTP and SMTP. SOAP is basically a RPC (‚remote procedure call') mechanism used to call application functions in remote systems and return their results. Therefore SOAP provides XML frames for requests, messages and responses.

SPECTO supports SOAP via predefined frames (‚query', ‚answer', ‚message' and ‚fault') which may be parameterized in the usual way, and some predefined variables (‚SOAP-action', ‚SOAP-encoding', ‚SOAP-namespace', ...).

In the response the new http status 500 is made available in variable ‚SOAP-response-http500', then the SOAP XML frame is stripped, and the result of the remote function call handled over to the SPECTO ‚content check'.

Example:


Client = '1' (Mathesis GmbH)

Status:

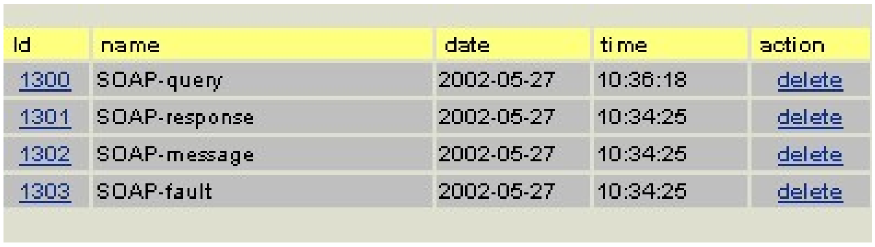
Last changed :
2001-04-29 10:01:12

name:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org
xmlns:xsi=http://www.w3.org/1999/XMLSchema-instance
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
<nsl:%%soap_method_response%% xmlns:nsl="%%soap_target%%"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encod
```

Note : The %%xxx%% denotes the access to the variable xxx. For details see chapter ‚Working with text' (page 45).

All SOAP frames have symbolic names starting with 'SOAP'; they can be listed using the command 'tal SOAP' :



Id	name	date	time	action
1300	SOAP-query	2002-05-27	10:36:18	delete
1301	SOAP-response	2002-05-27	10:34:25	delete
1302	SOAP-message	2002-05-27	10:34:25	delete
1303	SOAP-fault	2002-05-27	10:34:25	delete

Function: xCBL

XCBL is Commerce one's 'common business library' :

Example of a parametrized business document :

The following SPECTO business document skeleton (in xCBL format) :

```
…
<orderheader>
   <poissuedate>%%idate%%</poissuedate>
   <requesteddeliverydate>%%ddate%%</requesteddeliverydate>
</orderheader>
<listoforderdetail>
   <orderdetail>
          %%Onicount:0:2%%
          <baseitemdetail>
                 <lineitemnum>%%icount%%</lineitemcount>
                 <supplierpartnum>%%partnum:icount%%</supplierpartnum>
                 <qunatity>%%quantity:icount%%</quantity>
                 <notes>/l124+icount/l</notes>
          </baseitemdetail>
          %%ENDON%%
   </orderdetail>
</listoforderdetail>
<ordersummary
</ordersummary>
…
```

will at runtime be transformed using this parameters :

```
idate            = 2001-01-01
ddate            = 2001-01-15
partnum          = ;4711;1204;9999;
quantity         = ;2;4;11;
textarea 8       = 'handle with care'
textarea 124     = ''
textarea 125     = 'ship sea packed'
textarea 126     = 'print "/l8/l" on package'
```

to the following final document :

```
<orderheader>
   <poissuedate2001-01-01</poissuedate>
   <requesteddeliverydate>2001-01-15</requesteddeliverydate>
</orderheader>
<listoforderdetail>
   <orderdetail>
          <baseitemdetail>
                 <lineitemnum0</lineitemcount>
                 <supplierpartnum>4711</supplierpartnum>
                 <qunatity>2</quantity>
                 <notes></notes>
          </baseitemdetail>
          <baseitemdetail>
                 <lineitemnum0</lineitemcount>
                 <supplierpartnum>1204</supplierpartnum>
                 <qunatity>4</quantity>
                 <notes>ship sea packed</notes>
          </baseitemdetail>
          <baseitemdetail>
                 <lineitemnum0</lineitemcount>
                 <supplierpartnum>9999</supplierpartnum>
                 <qunatity>11</quantity>
                 <notes>print 'handle with care' on package</notes>
          </baseitemdetail>
   </orderdetail>
</listoforderdetail>
<ordersummary
```

```
</ordersummary>
```

Properties

Properties are set in scripts using the 'set' command. Here, only the properties specific for b2b are described, see chapter 'Object hierarchy – URLs' for other properties.

Property name	Description	Allowed values / examples
b2bMode	set b2b mode, 0 = no b2b specific behaviour	numeric, (default is '0')
b2bTransferEnvelope	empty = none, symbolic name of text-area else. (link name in frame = '%%link%%)	symbolic name (default is empty)
b2bContentEnvelope	empty = none, symbolic name of text-area else. (link name in frame = '%%link%%)	symbolic name (default is empty)
b2bAsyncResponse	the response is transmitted asynchronously and should be waited for	true, false (default is 'false')

E-mail server

The SPECTO engine can act as a server incoming for e-mails. B2B applications may send e-mail messages to the SPECTO engine which processes them by a custom script.
The script specified in attribute 'EmailInScript[src]' is executed for every incoming message in the context of the client specified in attribute ‚EmailInClient[src]'.
Setting the attribute ‚EmailInDebug[src]' to ‚true' enables the generation of detailed messages during the script processing.
The specified script is called with three parameters containing the sender, the subject and the message body.
The E-mail server processing must be enabled by setting attribute ‚EmailInEnabled[src]' to ‚true'. The E-mails are read from post offices specified in attributes EmailInSource[src] ('src' being a unique identifier, valid examples are 'EmailInSource', 'EmailInSourceFirst'); with the value structured as '`hostname[;inbox]:username:password`' (examples: 'www.gmx.de:ussp345:secret', 'mail.nls.de;in01:gspecto:guest').

Example E-mail server script :

```
// script=js
// Server for incoming e-mails

if (specto.args.length != 3) {
   specto.log("E-mail: args=" + specto.args.length +" ?");
}
else {
   specto.log("Processing e-mail from " + specto.args[0] +
       "; subject: " + specto.args[1]);
}
```

XML server

The SPECTO engine can act as a server for XML queries. B2B applications may send XML messages to the SPECTO engine which processes and returns them.
The script specified in attribute 'XMLInScript' is executed for every incoming message in the context of the client specified in attribute ‚XMLInClient'.
Setting the attribute ‚XMLInDebug' to ‚true' enables the generation of detailed messages during the script processing.
The specified script is called with one parameter containing the complete XML message.
The result of the script is returned to the sender of the XML message
The XML server processing must be enabled by setting attribute ‚XMLInEnabled' to ‚true'.

Example XML server script :

```
// script=js
// XML incoming calls

var result = "";
if (specto.args.length != 1) {
   specto.log("XML: argus=" + specto.args.length +" ?");
}
else {
var xmlBody       = specto.getXMLObject(specto.args[0]);
var xmlComponent  = xmlBody.getFirst("");
var numComponents = 0;

   while (xmlComponent != null) {
      specto.log(xmlComponent.getContent());
      xmlComponent = xmlBody.getNext("");
      numComponents++;
   }
   result="<answer>XML answer from script \"xml_in\": " +
      numComponents + " components</answer>";
}

// return content of variable result
result;
```

SOAP server

The SPECTO engine can act as a server for SOAP queries. B2B applications may request SOAP queries from the SPECTO engine which processes and answers them. The processing is similar to the XML server (see chapter before), the script specified in attribute 'SoapInScript' is executed for every incoming message in the context of the client specified in attribute ‚SoapInClient'.
Setting the attribute ‚SoapInDebug' to ‚true' enables the generation of detailed messages during the script processing.
The specified script is called with three parameters, the first parameter is the content of the SOAPAction HTTP header, the second parameter is the complete SOAP Header and the third parameter is the complete SOAP body. The result of the script is packaged into a valid SOAP response and returned to the sender of the SOAP query.
The SOAP server processing must be enabled by setting attribute ‚SoapInEnabled' to ‚true'.

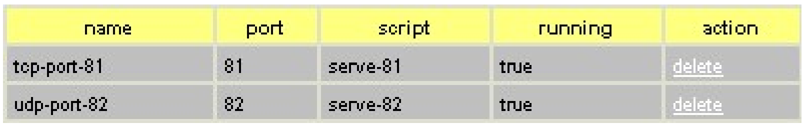
Example SOAP server script :

```
// script=js
// SOAP incoming calls

var result = "";
if (specto.args.length != 3) {
   specto.log("SOAP: args=" + specto.args.length +" ?");
}
else {
var soapAction = specto.args[0];
var xmlHead    = specto.getXMLObject(specto.args[1]);
var xmlBody    = specto.getXMLObject(specto.args[2]);
var xmlMet     = xmlBody.getFirst("");
var numMethods = 0;

   while (xmlMet != null) {
      specto.log(xmlMet.getContent());
      xmlMet = xmlBody.getNext("");
      numMethods++;
   }
   result = "SOAP answer from script \"soap_in\": " +
      numMethods + " method calls.";
}

// return content of variable result
result;
```

Port/socket services and port/socket server

The SPECTO engine can act as a server for incoming socket/port calls. Multiple port services may be active at the same time. It is also possible to generate TCP port connections, TCP port connections with transmissions, and UDP transmissions.
Any incoming port/socket request is handled by a script; the script name is specified during the definition of a port service.

The 'pl' command group is used to manage port services and port requests, e.g. 'pl l' lists currently defined services :



name	port	script	running	action
tcp-port-81	81	serve-81	true	delete
udp-port-82	82	serve-82	true	delete

SNMP trap server

The SPECTO engine can act as a server for incoming SNMP traps. Multiple SNMP trap services may be active at the same time. Any incoming port/socket request is handled by a script; the script name is specified during the definition of a port service.

Advanced B2B features

(provided as 'experimental' with release 1.60; will be operational with release 1.62 of SPECTO).

WSDL ('web services description language')

WSDL ('web services description language') is an XML format for describing interface, protocol bindings and the deployment details of network services as a set of endpoints operating on messages containing document-oriented or procedure-oriented information.

SPECTO maintains WSDL documents as textareas and provides special script commands to modify them :

<value> = wsdl.tModel.getTag <wsdl-doc> <tag>
<lastValue> = wsdl.tModel.setTag <wsdl-doc> <tag> <setValue>
<value> = wsdl.tModel.getAttribute <wsdl-doc> <tag> <attribute>
<lastValue> = wsdl.tModel.setAttribute <wsdl-doc> <tag> <attribute> <setValue>

Procedure for communicating WSDL information :

- copy from a WSDL-schema textarea to a variable
- modify the variable's content using wsdl… commands
- transmit using SOAP with SOAP wrapper 'SOAP-wsdl'
- analyze result

SPECTO can access WSDL endpoints using SOAP 1.1, HTTP and MIME. Within SPECTO WSDL is primarily used for access of UDDI registries.

UDDI ('universal description, discovery and integration')

UDDI ('universal description, discovery and integration') is a specification for accessing registries using SOAP and XML, eventually via WSDL.

SPECTO UDDI service is implemented for UDDI 2.0, featuring 'inquiry' and 'publishing'. It can be used in various situations :

- Monitoring of a public or private UDDI operator site
- Alerting on changes in specific UDDI entries
- Actualizing information necessary for a monitor from an UDDI registry

Features new with UDDI 2.0 support :

- Definition of relationship data

- validated classification and identification

Format

The syntax for the URL is :
`uddi:<URL>:<accesstype>`

with <accesstype> being 'soapSPEC' or 'wsdlSPEC'.

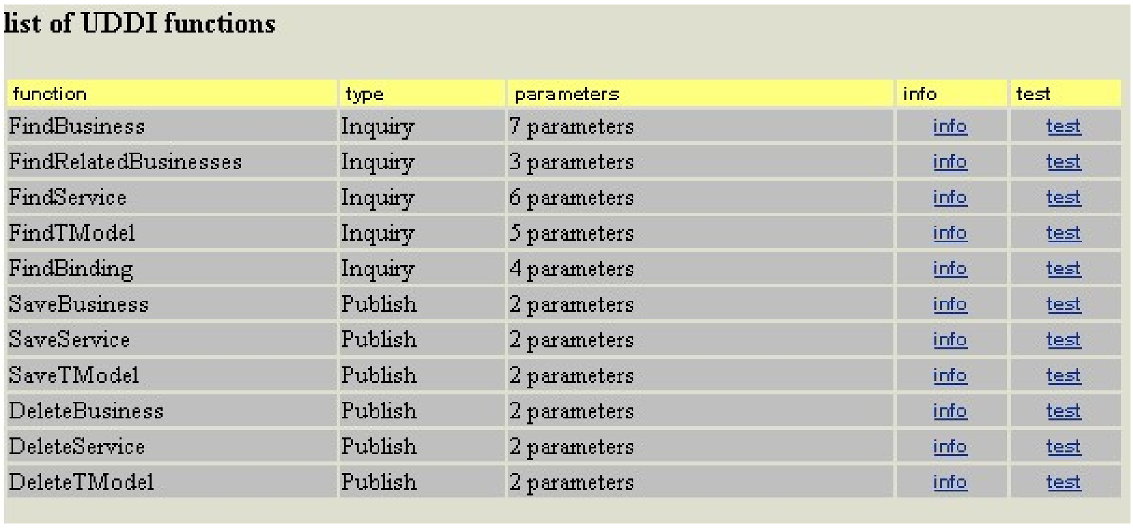
The following parameters, according to the JMS specification are available :

`'type'`	UDDI transaction type ('find', 'register')
`'object'`	UDDI object type ('business entity', 'service', 'binding', 'tModel')
`'content'`	WSDL document (XML format)

SPECTO's UDDI implementation may use HTTP and HTTPS as transport protocol.

The command 'uddi' gives an overview of the available UDDI subcommands.

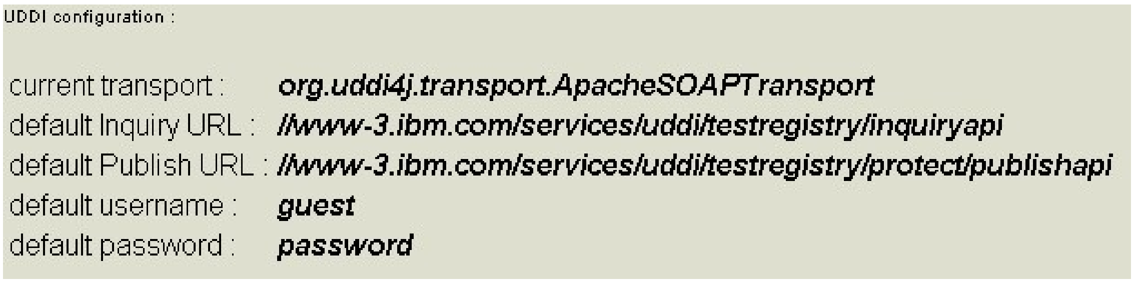
Command 'uddi l' lists the currently defined functions :



list of UDDI functions

function	type	parameters	info	test
FindBusiness	Inquiry	7 parameters	info	test
FindRelatedBusinesses	Inquiry	3 parameters	info	test
FindService	Inquiry	6 parameters	info	test
FindTModel	Inquiry	5 parameters	info	test
FindBinding	Inquiry	4 parameters	info	test
SaveBusiness	Publish	2 parameters	info	test
SaveService	Publish	2 parameters	info	test
SaveTModel	Publish	2 parameters	info	test
DeleteBusiness	Publish	2 parameters	info	test
DeleteService	Publish	2 parameters	info	test
DeleteTModel	Publish	2 parameters	info	test

Command 'uddi s' displays the current configuration :



UDDI configuration :

current transport :	*org.uddi4j.transport.ApacheSOAPTransport*
default Inquiry URL :	*//www-3.ibm.com/services/uddi/testregistry/inquiryapi*
default Publish URL :	*//www-3.ibm.com/services/uddi/testregistry/protect/publishapi*
default username :	*guest*
default password :	*password*

Notification

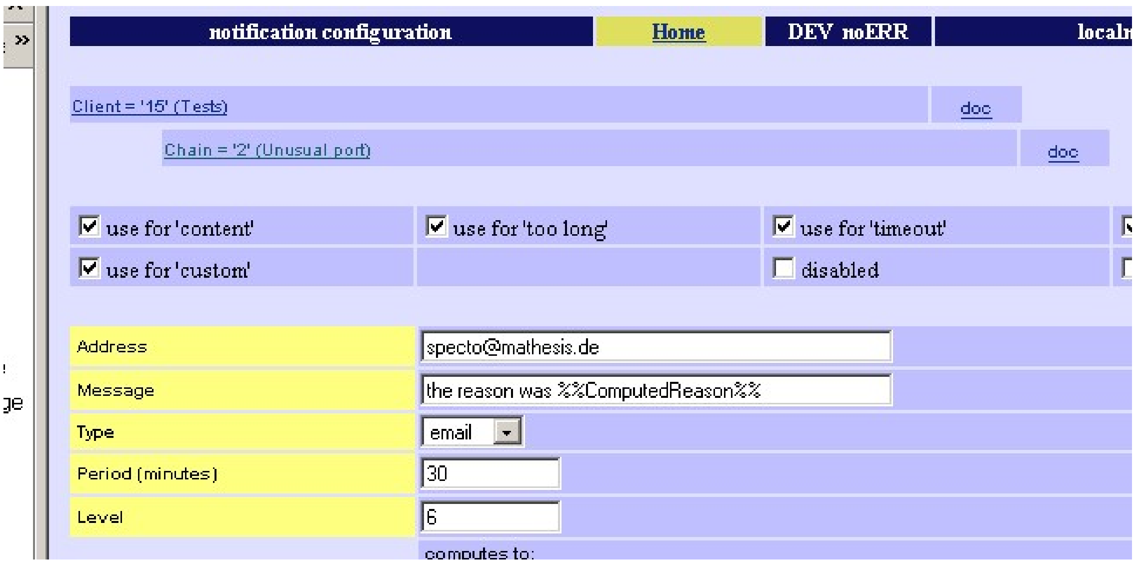
Whenever a HTML/XML is executed, a status is computed in the areas :

- Timeout
- TooLong
- Content
- User defined
- Wait loop

Any status area is then used to compute the corresponding error level. The status' error levels, minimum and maximum levels and the upper limit can be reviewed using the 'no s' command.

Whenever the maximum level of a status area is reached, a notification is added to the notification list. When the minimum level of a status area is reached (coming from a higher level) the corresponding notifications are removed from the notification list. Any entry in the notification list is executed at regular intervals.

Notifications can be edited in detail following the notification entry link from the chain configuration screen :



Client = '15' (Tests)

Chain = '2' (Unusual port)

✓ use for 'content'	✓ use for 'too long'	✓ use for 'timeout'
✓ use for 'custom'		☐ disabled

Address	specto@mathesis.de
Message	the reason was %%ComputedReason%%
Type	email
Period (minutes)	30
Level	6
	computes to:

For any notification the error type for which the notification will be triggered. Also a script ('PBN') can optionally be executed before the notification will be executed. In this script the notification can be cancelled using the 'notificationEnable' property.

Notifications are processed in the separate notification engine. The following commands are available :

'no s'	List all current status counters (status with count 0 are not shown)
'no l'	List all current notifications
'no d' <chain-id>'	Delete a notification
'no i' <chain-id> <error-type>	Add a notification manually (for test purposes)

Available types of notifications :

Email notification

Whenever a notification is active an email to the supplied address will be send. The notification type for this interface 'E', the configuration is "command_name"; the parameter string is of the format "clients-name@clients-host".
Sending of emails requires SPECTO to have an email connection. The underlying operating system has to be enabled for email processing (separate client software) and the attribute 'MailHost' must be set to valid SMTP server. The senders address can be configured in the 'MailFrom' attribute.
In case of problems with the processing of emails the 'MailDebug' attribute may be set to 'true' (otherwise 'false') to turn on advanced reporting during email processing.

FAX notification

Whenever a notification is active a operating system command may be configured. The notification type for this interface 'F', the parameter string is a fax number.
FAX notifications are implemented using a smtp or web interface to the FAX service supplier.

FAXType	Type of the FAX service
	0 = NLS via SMTP
	1 = NLS via web
	10 = Promedia via SMTP (the default)
FAXProvider	Id (e.g. email-address) of the FAX service provider
FAXUsername	Logon name for the FAX service
FAXPassword	Logon password for the FAX service

SMS notification

Whenever a notification is active a operating system command may be configured. The notification type for this interface 'S', the configuration is "command_name"; the parameter string is the telephone number of a mobile phone".
SMS notifications are implemented using a web or smtp interface to the SMS service supplier. The following attributes are available to configure SMS service :

SMSType	Type of the SMS service
SMSProvider	0 = NLS via SMTP
	1 = NLS via web
	10 = Promedia via SMTP (the default)
SMSProvider	Id (e.g. email-address) of the SMS service provider
SMSUsername	Logon name for the SMS service
SMSPassword	Logon password for the SMS service

Telephone notification

Whenever a notification is active a operating system command may be configured. The notification type for this interface 'T', the configuration is "command_name"; the parameter string is of the format "client:chain:delay:result".
Telephone notification requires a programmed voice-type modem attached to port 'com1:' (Windows NT) or '/dev/com0' (Unix).

Native command notification

Whenever a notification is active a operating system command may be configured. The notification type for this interface 'O', the configuration is "command_name"; the parameter string is of the format "client:chain:delay:result".

Java class notification

Whenever a notification is active, a configurable Java method in a class (loaded by 'ClassByName') may be called. The notification type for this interface 'J', the configuration is "class:method:"; the result string is of the format "client:chain:delay:result".

SNMP notification

Whenever a notification is active, a SNMP trap (versions 1, 2 or 3) may be send to a network management application on the machine specified in the 'notification' address field. The trap may be configured using a set of attributes which have to be entered in the 'message' part of the notification. The format is :
`'attribute-name'='attribute-value'`; the individual attribute specifications separated by a space character. Example: '`version=2 community=germany generic=2`'

Supported attributes :

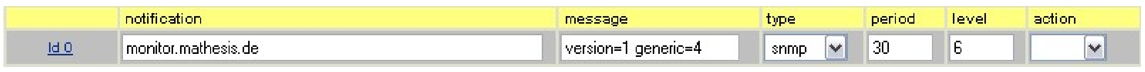
Attribut	Values	Example	Description
port	numeric	163	**Listening port on the target machine (default = 162)**
version	1, 2, 3	2	**SNMP version (default = 1)**
enterprise	String	1.1	**SNMP enterprise (default = 1.1)**
specific	0, 1, …	0	**Type of trap in version 1 (default = 0)**
generic	1, 2, …	4	**Type of generic trap in version 1 (default = 1)**
mib	'dotted' mib - format	1.3.1.6	**Enterprise specific mib; overrides 'generic' setting**
uptime	Numeric (milliseconds)	1000	**Adds an 'uptime' (1.3.6.1.2.1.1.3(.0)) entry**
contact	String	contact	**Adds an 'contact' (1.3.6.1.2.1.1.4.0) entry**
community	String	ec	**SNMP community (default = public)**
context	String	ctx1	**SNMP context (version 3 only) (default = blank)**
protocol	MD5, SHA1	SHA1	**Protocol (version 3 only) (default =**

			MD5)
user	String	User	**Version 3 authentication user name**
password	String	Pwd	**Version 3 authentication user password**
privacy	**String**	**Privacy**	**Version 3 authentication privacy password**

The following 'generic' values are implemented :

Id	description	Mib
0	cold start	**1.3.6.1.6.3.1.1.5.1**
1	warm start	**1.3.6.1.6.3.1.1.5.2**
2	link down	**1.3.6.1.6.3.1.1.5.3**
3	link up	**1.3.6.1.6.3.1.1.5.4**
4	authentication failure	**1.3.6.1.6.3.1.1.5.5**
5	egpNeighborLoss	**1.3.6.1.6.3.1.1.5.6**
6	**enterprise specific (use attribute 'mib' to specify the value)**	**1.3.6.1.6.3.1.1.5.7**

Example :

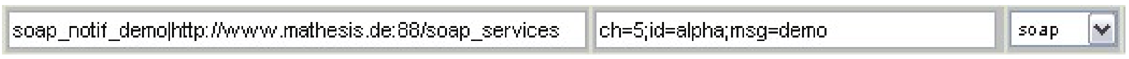


	notification	message	type	period	level	action
Id 0	monitor.mathesis.de	version=1 generic=4	snmp	30	6	

SOAP notification

Whenever a notification is active, a configurable SOAP message may be initiated.
The SOAP target (system and function) is configured in column 'notification' :
'`function:hostname`' or '`function|hostname`'.
The SOAP function's parameters are configured in column 'message' :
'`parameter1=value1;parameter2=value2;`…'.

Example :



soap_notif_demo|http://www.mathesis.de:88/soap_services	ch=5;id=alpha;msg=demo	soap

SAP R/3 notification

Whenever a notification is active, a configurable R/3 (RFC capable) function module may be called. This requires the SAP R/3 Java JRFC library installed on the SPECTO system.
The notification type for this interface = 'R', the configuration is 'hostname:systemnumber:userid:password:clientid:functionname:import_parameter"; the result string is of the format "client:chain:delay:result".

Example :

Adresse http://localhost/servlet/SpectoHome?session=4467&service=specto

notification

HP OpenView/VantagePoint integration

SPECTO can notify a HP openview/vantagepoint system management instance using the openview 'opcmsg' interface. The vantage point 'a' parameter is set to 'specto', the 'o' parameter to 'WWW'.
The 'msg_text' parameter consists of the field/value pairs :

- 'Problem' with 'report' if a problem exists, or 'release' if a problem has disappeared.
- 'for' identifies the object in client/chain/url notation
- 'type' error type; values are 'Timeout', 'TooLong', 'Content' or 'Custom'
- 'at' the time when the event happened
- 'note' the 'notification' field of the SPECTO chain configuration screen

SAP R/3 CCMS integration

Overview

The SPECTO to CCMS interface allows the monitoring of various status objects of SPECTO instance(s) and the customization of selected SPECTO objects.

Release 1.0 : view of notification status
Release 1.5 : view of SPECTO customizing
Release 2.0 : change of selected SPECTO customizing

SPECTO assistance : notifications

The SPECTO notification object attributes, active notifications, and status counters of the notification engine are made available.

SPECTO assistance : object hierarchy

The SPECTO object hierarchy is made available for reading to the CCMS interface.

SPECTO assistance : commands

The SPECTO command interpreter can be accessed via CCMS.

Node hierarchy

SPECTO	SPECTO	root node

SPECTONet	net	??
engine	engine	summary node
clients[]	client[]	summary node
chains[]	chain[]	summary node
notifications[]	notifications	node
status[]	status	node
measuredData[]	data	performance node

Operation

Preconditions

The attribute “R3Interfaces” must be set to “true” to enable SPECTO-R3 functionality.

Creation of CCMS objects

Before the CCMS integration is operable and whenever attributes are changed, the relevant CCMS objects must be created using command ‘it ci’.
note: CCMS integration should not be activated before/during this operation

Continous operation

CCMS integration is activated by setting attribute ‘CCMSActivation’ to ‘true’ followed by command ‘it cr’ or a SPECTO engine restart. During operation some parameters can be changed by modifying the corresponding attributes and rereading them using command ‘it cr’.

Status & Logging

CCMS status can be inquired using command ‘it cs’. CCMS log verbosity can be set using command ‘it cv <verbosity>’, with verbosity ranging from ‘0’ (no messages) to ‘3’ (all messages), default verbosity is ‘1’.

Technical implementation

The SPECTO CCMS interface is based on JMonAPI, a java based front for CCMS clients, supplied by SAP. The interface type is active, any changes of object values will be transmittedt to the jmon client at regular (default 5000 milliseconds, customizable with attribute ‘CCMSRefreshPeriod’).

Defined objects :

object	Value	parent	description
AnalysisMethod	SPECTO		
RootNode	SPECTO	-	
SummaryNode			
ObjectNode			

StatusLine			

Installation

CCMS integration is a standard part of the SPECTO product. To activate it, the following additional measures have to be taken :
installation of the R/3 client protocol
installation of the R/3 JRFC libraries
installation of the CCMS client
installation of the JmonApi java runtime
inclusion of the JmonApi libraries into the SPECO CLASSPATH
customizing of the CCMS parameters in SPECTO
customizing of the CCMS parameters in R/3

1. Installation of the R/3 client protocol

By installation of the SAPGUI component. (maybe not necessary, but a working SAPGUI connection is a good proof of a correct connection).

2. Installation of the R/3 JRFC libraries

By execution of the SAPGUI's JRFC installation.

3. Installation of the CCMS client

Windows nt/2k/xp :
The library '`jmonslib.dll`' must be copied to the windows system directory.
The application '`sapccmsr.exe`' must be installed and tested according to the SAP documentation.

UNIX :
n/a

4. Installation of the JmonApi java runtime

The following java libraries are accessed by SPECTO's CCMS interface :
```
jmonapi.jar
logging.jar
```
They must be copied to the SPECTO installation.

5. Inclusion of the JmonApi libraries into the SPECO CLASSPATH

The following libraries are accessed by SPECTO's CCMS interface :
```
com.sap.rfc.*
com.sap.rfc.exception.*
com.sap.mona.api.*
```

This can be accomplished by copying the libraries into the existing SPECTO classpath or (preferred) by adding their location to the SPECTO classpath during SPECTO startup (script 'init'). Example:

```
set CLASSPATH=%CLASSPATH%d:\jdk1.3\lib\jmonapi.jar;
set CLASSPATH=%CLASSPATH%d:\jdk1.3\lib\logging.jar;
```

6. Customizing of the CCMS parameters in SPECTO

The following attributes specify SPECTO's CCMS interface :

attributename	default	'it cr' aware	description
`CCMSActivation`	`false`	yes	must be 'true' for CCMS integration to be activated.
`CCMSR3Host`		no	the R/3 hostname or IP address
`CCMSR3System`	`00`	no	the R/3 system number
`CCMSR3Client`	`100`	no	the R/3 client number
`CCMSRefreshPeriod`	`5000`	yes	the refresh period of SPECTO object status changes to the CCMS client. In milliseconds
`CCMSRootNode`	`SPECTO`	yes	name of the SPECTO root node in the CCMS hierarchy

The attributes may be client-specific.

7. Customizing of the CCMS parameters in R/3

By usage of transaction RZ21 :

Forwarding of notifications / NLS notification services

Most SPECTO notifications are based on emails. However, when an email infrastructure (SMTP/POP3/IMAP4) is not present at the client's location, email-bound notifications may be forwarded to another SPECTO instance using SOAP messages via standard HTTP protocol.

Notification forwarding is enabled by ('Customizing'-'mail transport') setting attributes 'MailViaSOAP' to 'true', and specifying the URL of the serving SPECTO engine in attribute 'MailHostSOAP'. The serving SPECTO engine must have incoming SOAP enabled (see chapter 'SOAP server').

Notification forwarding via SOAP may also be used as a backup for SMTP based mail transport: If attribute ‘MailBackupViaSOAP’ is set to true, whenever a mail sent using SMTP protocol fails, an alternate access via SOAP is executed.

If there is no SPECTO instance with access to an email infrastructure, forwarding may also occur to one of NLS’ hosted SPECTO instances.
In this case the demo URL is : `http://www.NLS.de/Specto/SpectoHome`

User management

Standard

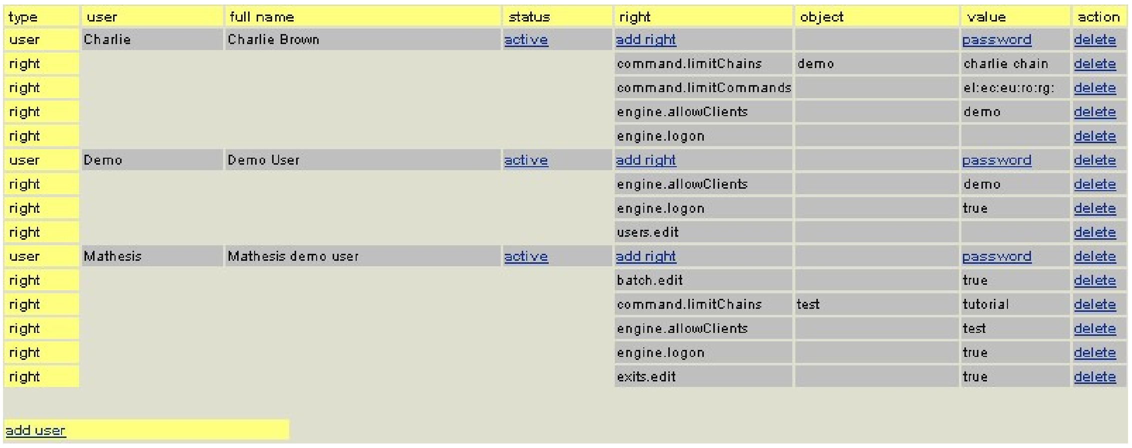
The SPECTO admin user account's name and password is maintained in attributes 'MasterUser' (the logon name, default is 'specto'), 'MasterPassword' (the logon password) and 'MasterClient' (for the active client after logon, default is '0'). The admin account is not, and cannot be limited in rights within the SPECTO engine.

For any client a client specific user entry is automatically created. The default password is the same as the username and the client name. The password may be changed using command :
'pw <new_password> <new_password> [<client_id>]'.
If a client's name is changed (using command *'cn <new_client_name>'*), the password is also set to the client's new name.
A client user has only access to 'his' client, it may be disabled for changes by setting the (client specific) attribute 'ReadOnly' to 'true' (menu: 'customizing' – 'security'). A client user cannot access the standard attributes; any attributes operations of a client user will automatically prefix the attribute name with a '_' character, so that the client user may store private information using SPECTO's attribute mechanism but cannot read or modify the standard attributes.

Enhanced

In addition to the standard user management the SPECTO engine features an enhanced user management allowing for a much grainer specification of users and rights as the standard user management. Enhanced user management must be enabled by setting attribute 'UsersEnable' to true (menu: 'customizing' – 'users'); (requires engine restart).

Users and rights are maintained using command 'us' (menu: 'maintenance' – 'users'). Recommended usage is the command 'us l' (menu entry 'list users & rights'); the picture below shows a sample 'us l' result screen layout :



type	user	full name	status	right	object	value	action
user	Charlie	Charlie Brown	active	add right		password	delete
right				command.limitChains	demo	charlie chain	delete
right				command.limitCommands		el:ec:eu:ro:rg:	delete
right				engine.allowClients		demo	delete
right				engine.logon			delete
user	Demo	Demo User	active	add right		password	delete
right				engine.allowClients		demo	delete
right				engine.logon		true	delete
right				users.edit			delete
user	Mathesis	Mathesis demo user	active	add right		password	delete
right				batch.edit		true	delete
right				command.limitChains	test	tutorial	delete
right				engine.allowClients		test	delete
right				engine.logon		true	delete
right				exits.edit		true	delete

add user

Users may be added and deleted by using the relevant links; for a new user its name, password and a description have to be entered. By default, a new user has no rights within the SPECTO engine; at least the 'engine logon' right has to be assigned to the user.
Assigning rights is done via the 'add right' link in the user row. In the following panel the right has to be selected from a list, and the 'object' and the right's 'value' concerning the object has to be entered. 'object' and 'value' are specific to the right, so they have to be correctly entered and cannot be selected from a list.
An 'object' is always a part of the SPECTO engine, e.g. clients, chains, the notification engine, the batch processor. The object column allows for lists (items separated by comma ',') and '*' as a selector for all objects.
The 'value' describes the level of access within the right which is granted to the object.

Several entries (rows) may be added for a right;/object in that case the 'value' fields are concatenated. The following table lists the currently available rights and their 'object' and 'value' specification :

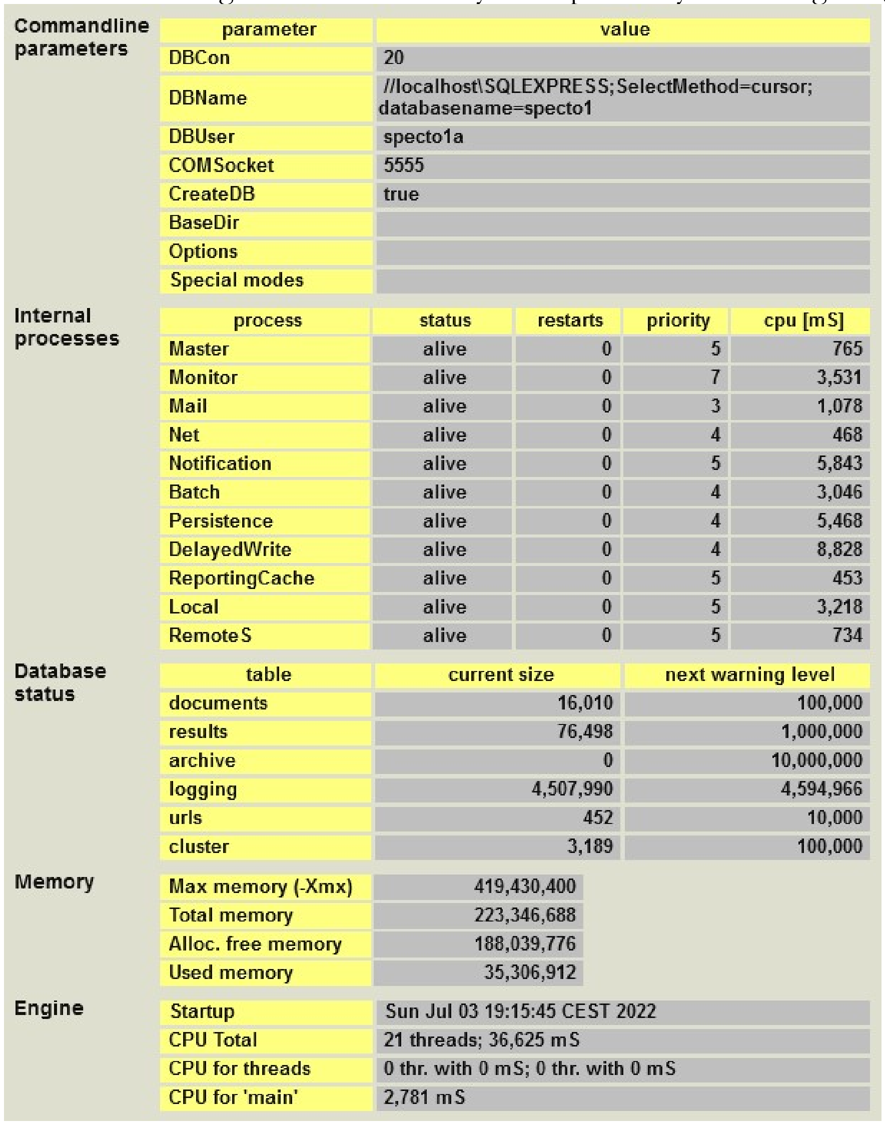
Right	**description**	**object**	**value**	**example (object, right)**
Client.access	* not yet implemented *			
Engine. allowClients	Enabled clients.		A ':' separated list of client names; or '*' to allow all clients	**demo:test:**
Engine.logon	Logon via browser (the most basic right, usually required for all users)	-	-	
Command. allowAdmin	Allowing commands normally reserved for the administrator		A ':' separated list of the allowed commands	**st:lc:**
Command. limitChains	Restricting the access to the specified chains (If not specified all chains of the client are allowed)	A ':' separated list of client names	A ':' separated list of chain names	**firstChain: thirdChain:**
Command. limitCommands	Restricting access to the specified commands (If not specified, all non-administrator commands are allowed)		A ':' separated list of the allowed commands	**rg:rh**
Users.edit	Access to the user management		true / false (default)	
Exits.edit	Access to the exits management		true / false (default)	
Batch.edit	**Access to the batch engine**		**true / false (default)**	

Any assigned or removed user and right comes in effect immediately (no engine restart required.

Engine management

Health check

The SPECTO engine's own status may be inspected by menu 'engine' (command 'cs') :



Commandline parameters

parameter	value
DBCon	20
DBName	//localhost\SQLEXPRESS;SelectMethod=cursor; databasename=specto1
DBUser	specto1a
COMSocket	5555
CreateDB	true
BaseDir	
Options	
Special modes	

Internal processes

process	status	restarts	priority	cpu [mS]
Master	alive	0	5	765
Monitor	alive	0	7	3,531
Mail	alive	0	3	1,078
Net	alive	0	4	468
Notification	alive	0	5	5,843
Batch	alive	0	4	3,046
Persistence	alive	0	4	5,468
DelayedWrite	alive	0	4	8,828
ReportingCache	alive	0	5	453
Local	alive	0	5	3,218
RemoteS	alive	0	5	734

Database status

table	current size	next warning level
documents	16,010	100,000
results	76,498	1,000,000
archive	0	10,000,000
logging	4,507,990	4,594,966
urls	452	10,000
cluster	3,189	100,000

Memory

Max memory (-Xmx)	419,430,400
Total memory	223,346,688
Alloc. free memory	188,039,776
Used memory	35,306,912

Engine

Startup	Sun Jul 03 19:15:45 CEST 2022
CPU Total	21 threads; 36,625 mS
CPU for threads	0 thr. with 0 mS; 0 thr. with 0 mS
CPU for 'main'	2,781 mS

Statistics

The 'statistics' link (see screen above) displays absolute and average values of key parameters of the engine (requests, measurements, commands, incoming calls, database).

Monitor

The 'checks' link (see screen above) displays details about the current configuration of the database table size checks and allows for their customization.

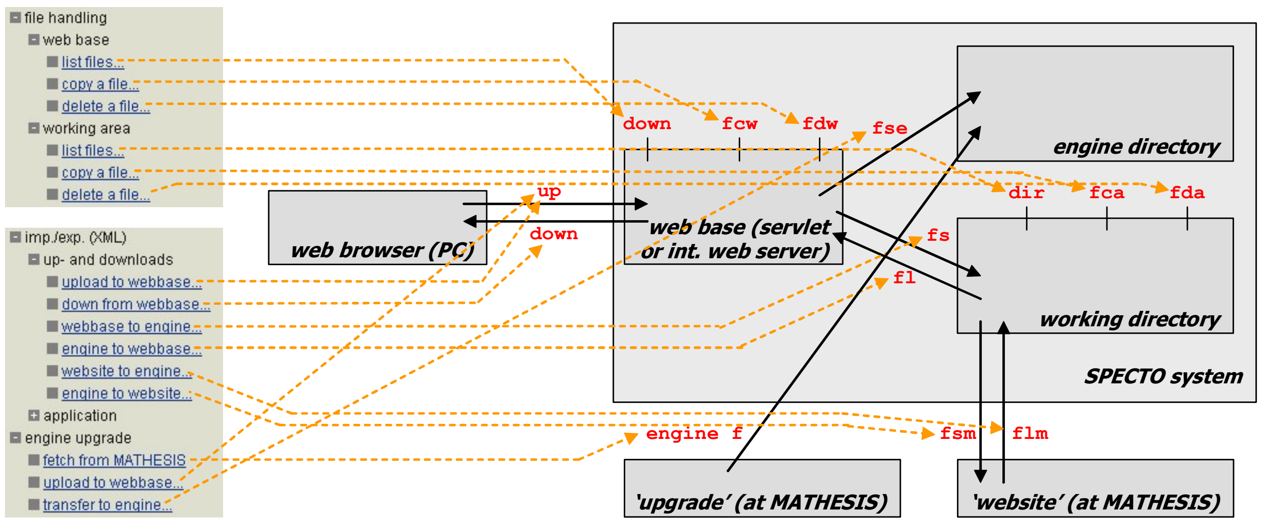
Working with files

Though all SPECTO data is stored in a relational database, for reasons of data transfer and upgrades there is some need to handle files in a SPECTO environment.

Using files some care has to be taken to differentiate between the various locations where the SPECTO system may access files:

- Work station of the user
- SPECTO system web area
- SPECTO engine working area
- SPECTO engine code area (for engine upgrades)
- Central (NLS hosted) SPECTO upgrade area
- Central (NLS hosted) SPECTO transfer area

The diagram below shows the architecture of possible file locations, the commands to transfer files between them, and the menu access to the commands :



Beside the necessary commands to *transfer* files there are additional commands ('fc', 'fd') to *copy* or *delete* files within the locations.

Above commands work identically with the usage of the servlet ('tomcat') based or SPECTO internal web-servers.

Database management

The SPECTO engine features an adaptive database interface with caching and recover mechanisms.

Database connections and recovery

It is recommended to use command *'db'* or menu entry *'maintenace – database'* to maintain the database engine.
The according customizing is done using menu entry *'customizing – database'*.

Delayed write

Database writes to result and logging tables may be (recommended) cached by the database interface.
It is recommended to use command '*dw*' or menu entry '*maintenace – delayed write*' to maintain the database engine. The according customizing is done using menu entry '*customizing – database*'.

Attribute cache

All attributes are cached by the database interface.

Command '*ac*' or menu entry '*maintenace – attribute cache*' may be used to maintain the caching status of attributes.

Document cache

Documents (also used for all scripts including ('PBC', 'PAC', …) are cached by the database interface.

Command '*ca*' or menu entry '*maintenace – documents*' may be used to maintain the caching status of documents.

Operator messages

On any incident occurring within the SPECTO engine operator messages are created. The current list of operator messages can be viewed using menu entry 'engine' – 'operator messages'.
Open operator messages are identified by a red link within the title of the main sub screen.

Operator messages may be forwarded to a supervisory system using the same interfaces as ‘notifications’ or the ‘am alive’ processor.

Operator messages are customized (menu ‘customizing’ – ‘operator messages’).

‘am alive’ messages

The SPECTO engine may forward periodic ‘am alive’ messages to a supervisory system using the same interfaces as ‘notifications’ or the ‘operator messages’.

‘Am alive’ messages are customized (menu ‘customizing’ – ‘am alive’).

Time base / NTP

The SPECTO engine features a configurable time base with optional synchronization from a hierarchy of NTP servers.

Command ‘*nt*’ or menu entry ‘*maintenace – time base*’ may be used to maintain the time base. The according customizing is done using menu entry ‘*customizing – time base / NTP*’.

Limits

The SPECTO engine periodically checks the sizes of several database tables and gernates operator messages if configured limits are reached..

The table size limits are customizable (menu ‘customizing’ – ‘limits’).

Auto export

The SPECTO engine allows for automatic daily export of all or changed client configurations.

‘Auto export’ is customizable (menu ‘customizing’ – ‘auto export’).

SPECTO execution extensions

Customer exits

Introduction

The SPECTO engine provides for user exits at every major event of its execution. The user exits are realized as dynamic java functions called by the 'class.forName()' mechanism of Java.

To simplify the method invocation, currently every exit has exactly two parameters of type 'String' and always returns a 'String' parameter. The first parameter denotes the Specto engine id as it is defined in the attribute 'SNThisNode'; the second parameter is specific to the type of exit, normally is compound by several sub parameters which are separated by colons.
Therefore the definition of a valid method should be like :

```
   public static String demo (String id, String param) …
```

Note that any exit function must be part of the Specto package, hence must include the '`package Specto;`' statement.

Exits can be tested using the 'te <class> <method> <parameter>' command.

Sample code

The following example can be used as a starter; and should be placed in a file named 'SpectoExitDemo.java'. The compiled .class file has to be at the same place as the other SPECTO engine .class files. A precompiled version is already included in the SPECTO engine.

The example can be tested using 'te Specto.SpectoExitDemo demo x'.

```
package Specto;

public class SpectoExitDemo {

// constructor
   SpectoExitDemo () {
          System.out.println("Initialize 'SpectoExitDemo'");
   }

// demonstration method
   public static String demo (String id, String param) {
   String  result;

          result = "From: " + id + "; got: " + param;
          return result;
   }
} // end of class SpectoExitDemo
```

Exits with customer supplied class/methods

In some cases the class and its method can be specified by the customer. E.g. custom notifications are realized using this feature. The exit has to be specified using the syntax :

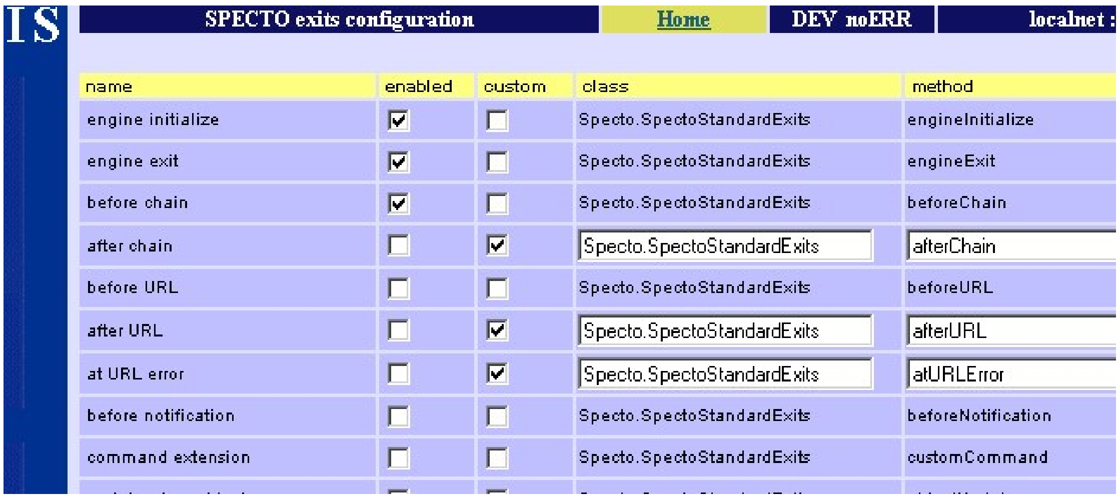
'class.method:id'. The id is optional; if supplied it will be appended to the parameter String during the method invocation.

Examples : `Specto.myClass.myMethod`
`Exits.genericExitClass.primaryExit:homeURL`

Summary of exits available in the SpectoEngine

The exit technology is a key feature of SPECTO and therefore NLS is keen in including exits wherever it is of interest for our customers. Just contact us if the above list does not fulfill your requirements.

The exits are maintained using command 'ex' :



name	enabled	custom	class	method
engine initialize	✓		Specto.SpectoStandardExits	engineInitialize
engine exit	✓		Specto.SpectoStandardExits	engineExit
before chain	✓		Specto.SpectoStandardExits	beforeChain
after chain		✓	Specto.SpectoStandardExits	afterChain
before URL			Specto.SpectoStandardExits	beforeURL
after URL		✓	Specto.SpectoStandardExits	afterURL
at URL error		✓	Specto.SpectoStandardExits	atURLError
before notification			Specto.SpectoStandardExits	beforeNotification
command extension			Specto.SpectoStandardExits	customCommand

Any exit can be individually enabled and instead of the default class/method an own class/method can be configured.

Example of an exit (coded in the java programming language) extracted from the SpectoStandardExits.java source model (available on demand) :

```
/*
 * application  SPECTO
 * module       standard exits
 *
 * date              author              reason
 * -------------------------------------------------------
 * 03.01.2001     -   creation
 * 21.01.2001     -   callbacks 'getData()', 'executeXML()'
 * 23.01.2001     -   exit 'heartbeat()'
 * 22.05.2001     -   inclusion of 'clientId' for client
 *                    specific exits.
 *
 */


//
// SPECTO exits - standard delivery
//
// (c) NLS GmbH 2001
//
// this source must be compiled in the same directory (...\Specto\) as the Specto
// .class files because javac will have to read them to resolve some of the
// methods used here.
//
// SPECTO provides for custom exits also. They can be included here, but it is
// recommended to use a separate source file instead.
//


package Specto;

public class SpectoStandardExits {
```

```
// constants
public        static    final     int       EXITS_NOLOG                                     = 0;
public        static    final     int       EXITS_LOGTOCONSOLE            = 1;
public        static    final     int       EXITS_LOGTOCONSOLEANDDB       = 2;

// class variables
private       static    int                 levelLog  = EXITS_LOGTOCONSOLE;         // default: log to console


// constructor
    SpectoStandardExits () {
              SpectoMain.Log.Log("Initialize 'SpectoStandardExits'");
              setLogExit(EXITS_LOGTOCONSOLE);
    }


// set console/log-logging
    public static int setLogExit (int levelLog) {
              return SpectoStandardExits.levelLog = levelLog;
    }


// write log entry
    private static void writeLogEntry (String text) {
              switch(levelLog) {
              case EXITS_LOGTOCONSOLE :
                        SpectoMain.Log.LogNoDB(text);
                        break;
              case EXITS_LOGTOCONSOLEANDDB :
                        SpectoMain.Log.Log(text);
                        break;
              }
    }


//
// SPECTO exits start here.
//
// It is recommended not to include your extensions into the exit methods
// itself but to implement separate methods and call them from the exits.
// This eases adaption of exit definitions changed by NLS...
//
// In this default implementation none of the exits 'does' anything
// besides writing log-entries and returning a mirror of the input data.
//

// the 'id' paramter holds the name of the instance.
// the 'param' parameter contains a concatenation of exit-specific
// values.

// Within any exit the following SPECTO methods (all can, but only those
// are 'guaranteed') may be called:
// int                        SpectoMain.getClient();
// SpectoData  SpectoMain.getData(int clientId);
// String               SpectoMain.Command.executeXML(String command);
//


// called during intialize phase of engine (happens at several places)
    public static String engineInitialize (String id, String param) {
    String    result;

              result = "EngineInitialize From: " + id + "; got: " + param;
              writeLogEntry(result);
              return result;
    }
```

...

SAP R/3 command line interface (CLI)

R/3 transaction 'YSC1' provides a command line interface to a SPECTO instance. CLI is available for R/3 rel. 4.0b and above; it does not support graphical and formatted (HTML) representations of SPECTO commands.
This requires the SAP R/3 Java JRFC library installed on the SPECTO system.

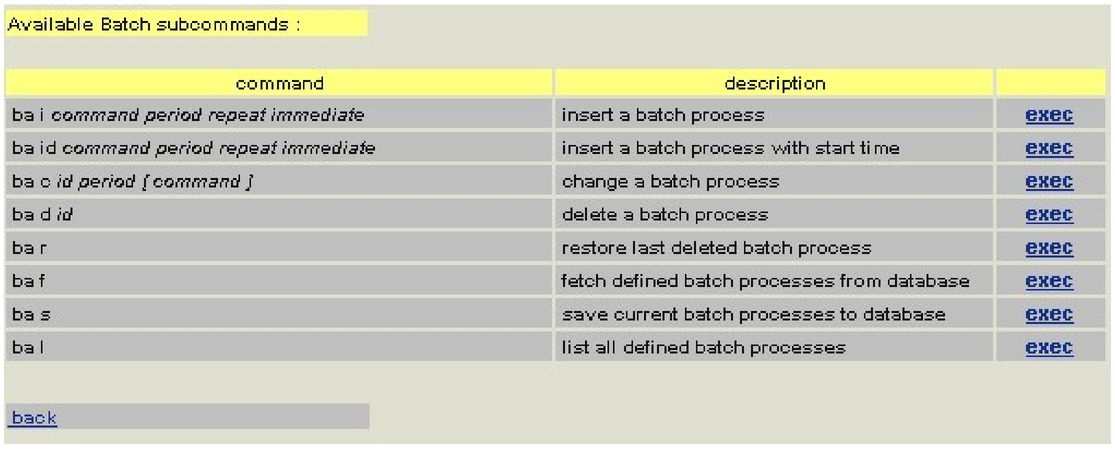
Remote execution sample code

SPECTO commands may be submitted using a TCP/IP socket connection. Sample code to demonstrate this capability is available upon request.

Batch execution

The SPECTO engine can execute commands in batch mode; Batch mode commands may be scheduled for specified times and may be repeated at specified intervalls. Batch type processing can be controlled by hand (e.g. for maintenance tasks like deletion of logs) and is used by the chain engine for delayed execution.

Batch subcommands (following 'ba') :

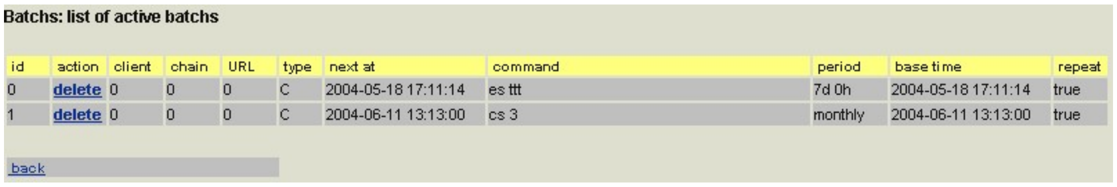


Available Batch subcommands :

command	description	
ba i *command period repeat immediate*	insert a batch process	exec
ba id *command period repeat immediate*	insert a batch process with start time	exec
ba c *id period [command]*	change a batch process	exec
ba d *id*	delete a batch process	exec
ba r	restore last deleted batch process	exec
ba f	fetch defined batch processes from database	exec
ba s	save current batch processes to database	exec
ba l	list all defined batch processes	exec

back

View of current batch processes ('ba l') :



Batchs: list of active batchs

id	action	client	chain	URL	type	next at	command	period	base time	repeat
0	delete	0	0	0	C	2004-05-18 17:11:14	es ttt	7d 0h	2004-05-18 17:11:14	true
1	delete	0	0	0	C	2004-06-11 13:13:00	cs 3	monthly	2004-06-11 13:13:00	true

back

Example :

```
  ba i 'ld 20' 120 true true
```

Above example inserts the command 'ld 20' for repeated execution every 120 minutes starting immediately.

The result of statements run in the background can be examined using the 'show' link.

Script

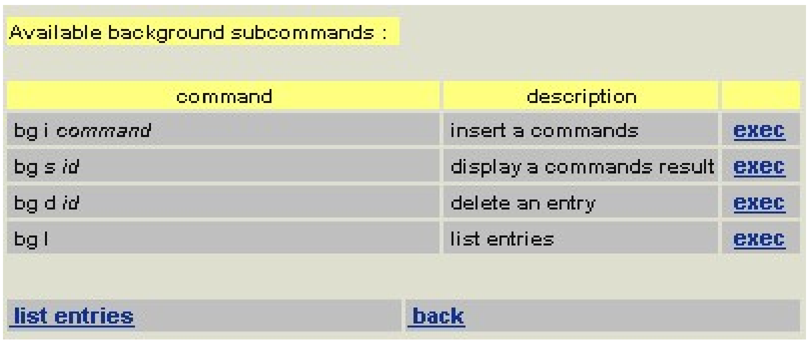
The batch processor is available for SPECTO scripts as object '`batch`'; a reference to the single instance may be retrieved by the method '`getBatchInstance()`'.
Sample code is available on request.

Background execution

Note: Any SPECTO command will continue execution still it is finished. If a new command is started via a command line entry or by a click on a link during a previous command is still executing, the output of previous commands will not be visible.

Therefore SPECTO can schedule commands to be executed in the background. Run status of background commands may be inspected and after being finished the output may be shown.

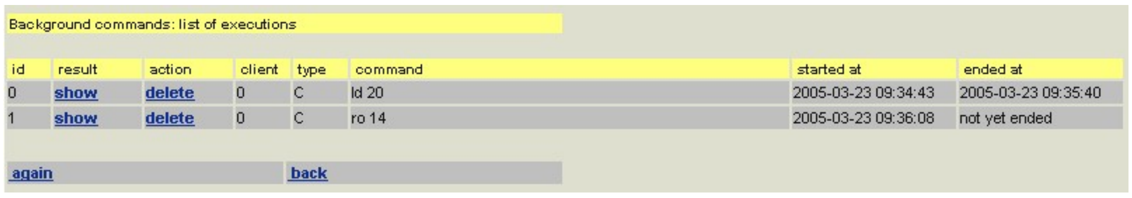
Background subcommands (following 'bg') :


Available background subcommands :

command	description	
bg i *command*	insert a commands	exec
bg s *id*	display a commands result	exec
bg d *id*	delete an entry	exec
bg l	list entries	exec

list entries back

View of current batch processes ('bg l') :


Background commands: list of executions

id	result	action	client	type	command	started at	ended at
0	show	delete	0	C	ld 20	2005-03-23 09:34:43	2005-03-23 09:35:40
1	show	delete	0	C	ro 14	2005-03-23 09:36:08	not yet ended

again back

Example :
```
  bg i ld 20
```

Above example schedules the command '`ld 20`' (deletion of log entries elder than 20 days) for background execution. After scheduling is done a list of previous and current background commands is shown.

Script

The background processor is available for SPECTO scripts as objects of class '`SpectoBackground`'; a reference to the list of instances can be retrieved by the method '`getBkgEntries()`'.
Sample code is available on request.

SPECTO NET

Overview

SPECTO instances are connected using SPECTO NET. SPECTO NET relies on the email ('SMTP') transport layer to be able to work across firewalls. All instances of a SPECTO NET share the same configuration data; all SPECTO NET masters hold the summary of all SPECTO instances results data.. Events like notifications can be forwarded within a SPECTO NET.

Architecture

Any SPECTO NET must consist of exactly one master and an arbitrary number of clients. Clients can be of types 'send only' or 'send and receive'. (because 'send and receive' clients need a email mailbox which is sometimes more effort to configure, the 'send only' client is included).
Any communication takes place between any client and the master, no client to client communication will happen. Data distribution takes place in two phases; during phase one any data is send from the clients to the master; in phase two the master sends, after consolidation, the changed data to all clients known to him.
The master 'auto learns' about his clients; any client that has sent any data to the master will be included into the list of clients.

Configuration

Preconditions & basic configuration

Any configuration is done using SPECTO attributes (see chapter 'commands' for administration of attributes).

Any node must have a unique name (attribute 'SNThisNode'), eg. 'SPECTO12' (maximum 12 characters.
Any node must have an email account (sending for non-master nodes; sending & receiving for master nodes), (attribute 'SNAccount').
Non master nodes must have a 'net master node' entry (attribute 'SNMasterNode'), this has to be set to 'master' for a 'net master node'.

Transfer cycles

The period of transfers can be configured in seconds (attribute 'SNPeriod'); if not maintained 30 minutes is assumed.

Transfer types

There are 'full' and 'delta' transfers. Full transfers transmit the complete configuration of a node; they can only be initiated manually. Delta transfers transmit the changed data since the last transfer; they are initiated automatically by SPECTO.

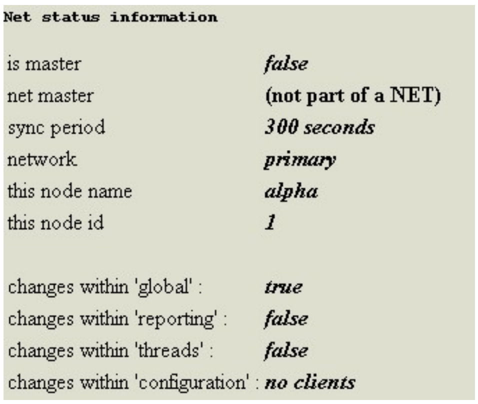
Operation

Activation

SPECTO NET operation may be activated or deactivated. At startup attribute ‘SNStartUp’ (values ‘0’ or ‘1’) define if SPECTO NET will be activated.

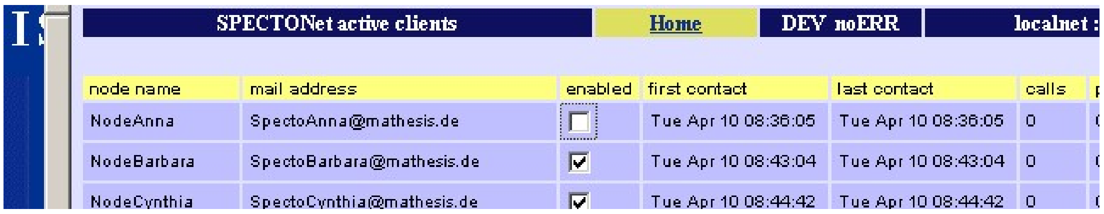
Status

Current SPECTONet status may be viewed using the ‘`snn i`’ command :



```
Net status information

is master                           false
net master                          (not part of a NET)
sync period                         300 seconds
network                             primary
this node name                      alpha
this node id                        1

changes within 'global' :           true
changes within 'reporting' :        false
changes within 'threads' :          false
changes within 'configuration' : no clients
```

A list of current nodes is available using ‘snn l’.



node name	mail address	enabled	first contact	last contact	calls
NodeAnna	SpectoAnna@mathesis.de	☐	Tue Apr 10 08:36:05	Tue Apr 10 08:36:05	0
NodeBarbara	SpectoBarbara@mathesis.de	✓	Tue Apr 10 08:43:04	Tue Apr 10 08:43:04	0
NodeCynthia	SpectoCynthia@mathesis.de	✓	Tue Apr 10 08:44:42	Tue Apr 10 08:44:42	0

Additional commands

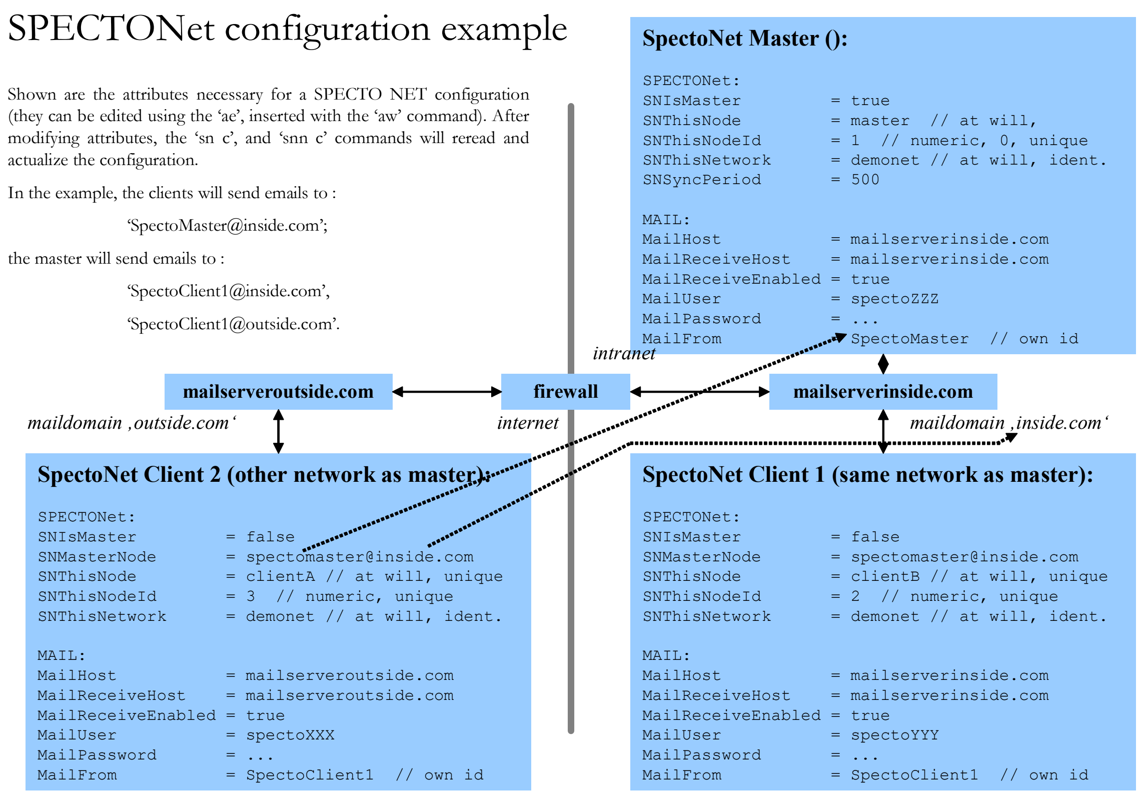
‘`sn on`’	Enable SPECTO NET
‘`sn off`’	Disable SPECTO NET
‘`sn ft`’	Initiate SPECTO NET full transfer
‘`sn dt`’	Initiate SPECTO NET delta transfer
‘`sn st`’	Display SPECTO NET status

Remote notification

Some SPECTO notification require additional infrastructure (software libraries or hardware) which may not be available to all nodes within a SPECTONet. Remote notification allows any client of a SPECTONet to forward its notification to the SPECTONet master which will execute it. Remote notification is available to all notification types and is selected by preceding the notification ‘level’ with a minus (‘-‘) sign.

Sample configuration

The sample configuration shown below is a basic SPECTO NET with two clients and one master. One of the clients is considered to be outside of a firewall and therefore needs a separate SMTP mailer.



SPECTONet configuration example

Shown are the attributes necessary for a SPECTO NET configuration (they can be edited using the 'ae', inserted with the 'aw' command). After modifying attributes, the 'sn c', and 'snn c' commands will reread and actualize the configuration.

In the example, the clients will send emails to :

'SpectoMaster@inside.com';

the master will send emails to :

'SpectoClient1@inside.com',

'SpectoClient1@outside.com'.

SpectoNet Master ():

```
SPECTONet:
SNIsMaster          = true
SNThisNode          = master  // at will,
SNThisNodeId        = 1  // numeric, 0, unique
SNThisNetwork       = demonet // at will, ident.
SNSyncPeriod        = 500

MAIL:
MailHost            = mailserverinside.com
MailReceiveHost     = mailserverinside.com
MailReceiveEnabled  = true
MailUser            = spectoZZZ
MailPassword        = ...
MailFrom            = SpectoMaster  // own id
```

SpectoNet Client 2 (other network as master):

```
SPECTONet:
SNIsMaster          = false
SNMasterNode        = spectomaster@inside.com
SNThisNode          = clientA // at will, unique
SNThisNodeId        = 3  // numeric, unique
SNThisNetwork       = demonet // at will, ident.

MAIL:
MailHost            = mailserveroutside.com
MailReceiveHost     = mailserveroutside.com
MailReceiveEnabled  = true
MailUser            = spectoXXX
MailPassword        = ...
MailFrom            = SpectoClient1  // own id
```

SpectoNet Client 1 (same network as master):

```
SPECTONet:
SNIsMaster          = false
SNMasterNode        = spectomaster@inside.com
SNThisNode          = clientB // at will, unique
SNThisNodeId        = 2  // numeric, unique
SNThisNetwork       = demonet // at will, ident.

MAIL:
MailHost            = mailserverinside.com
MailReceiveHost     = mailserverinside.com
MailReceiveEnabled  = true
MailUser            = spectoYYY
MailPassword        = ...
MailFrom            = SpectoClient1  // own id
```

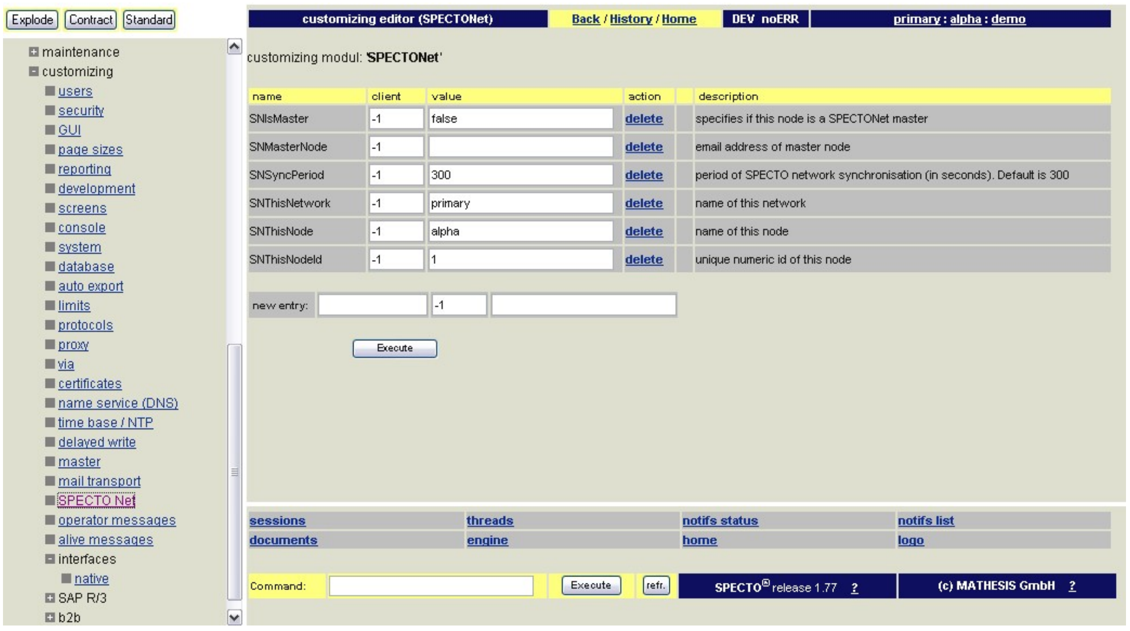
Typically a SPECTO instance hosted by NLS will be included in such a scenario.

SPECTO Customizing

Attributes

SPECTO is customized using attributes. Attributes can be set for the engine or for selected clients.

The attributes are grouped according their functionality; and can be accessed via the menu. Selecting a customizing group will open the attribute editor with the corresponding selection of attributes and their description in the main page.



customizing modul: 'SPECTONet'

name	client	value	action	description
SNIsMaster	-1	false	delete	specifies if this node is a SPECTONet master
SNMasterNode	-1		delete	email address of master node
SNSyncPeriod	-1	300	delete	period of SPECTO network synchronisation (in seconds). Default is 300
SNThisNetwork	-1	primary	delete	name of this network
SNThisNode	-1	alpha	delete	name of this node
SNThisNodeId	-1	1	delete	unique numeric id of this node
new entry:	-1			

Attributes may be defined globally, for all clients (entry 'client' is set to -1) or for specific clients. During attribute evaluation the SPECTO engine first tries to read the attribute value for the current client. If there is none defined the engine tries to read the global instance of the attribute. If there is even no global value the engine uses the default value.

Note that at every start of the SPCETO engine all missing attributes are recreated using their default values. So in case an attribute was deleted without intention, and the correct attribute name or value is not clear, it is sufficient to restart the SPECTO engine.

Attributes may also be created using the 'aw' command (`aw AttributeName Value [client]`), or maintained using the 'ae' command (`ae [ name_prefix ]`). Attribute names are not case sensitive, attribute values may be case sensitive.

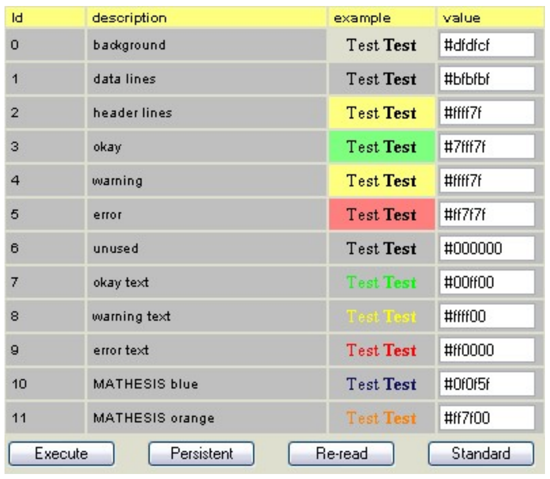
The current attributes may be exported/imported as XML formatted files using menu entries group 'maintenance' – 'imp./ex.(xml)' – 'base data'.

Attribute maintenance is only available to the administrative user.

Graphical User Interface

Coloring

The coloring of the SPECTO GUI can be customized using the ‘color definition’ screen (available via command ‘co’ or the menu (‘environment’ – ‘change colors’) :



Id	description	example	value
0	background	Test **Test**	#dfdfcf
1	data lines	Test **Test**	#bfbfbf
2	header lines	Test **Test**	#ffff7f
3	okay	Test **Test**	#7fff7f
4	warning	Test **Test**	#ffff7f
5	error	Test **Test**	#ff7f7f
6	unused	Test **Test**	#000000
7	okay text	Test **Test**	#00ff00
8	warning text	Test **Test**	#ffff00
9	error text	Test **Test**	#ff0000
10	MATHESIS blue	Test **Test**	#0f0f5f
11	MATHESIS orange	Test **Test**	#ff7f00

All color values are entered in hex (identified by the preceding ‘#’ character) and consist of the three two-digit values for the red, green and blue components of the color. The higher the value, the more intense is the part of the selected color. (#000000 is white, #ffffff is black).

Changes are activated using the ‘Execute’ button and come into effect immediately. If made persistent using the ‘Persistent’ button they are saved as attributes GUIColor<Id>. The current configuration can be read from the attributes using the ‘re-read’ button.

These color definitions also define the coloring of the text-base reporting. The graphical based reporting uses attributes ‘GColor<Id>’ for coloring and is describes with the ‘rg’ command in the ‘Commands in detail’ section of this manual.

Page sizes

In the ‘client configuration’ and ‘chain configuration’ screens the number of lines displayed on page can be set using the ‘page size’ (ps <clint> <chain>) command, by setting the attributes ‘pageSizeClient’ and ‘pageSizeChain’) or using the menu entry ‘environment’ – ‘page sizes’.

If paging is enabled page scroll buttons will appear automatically.

Sessions

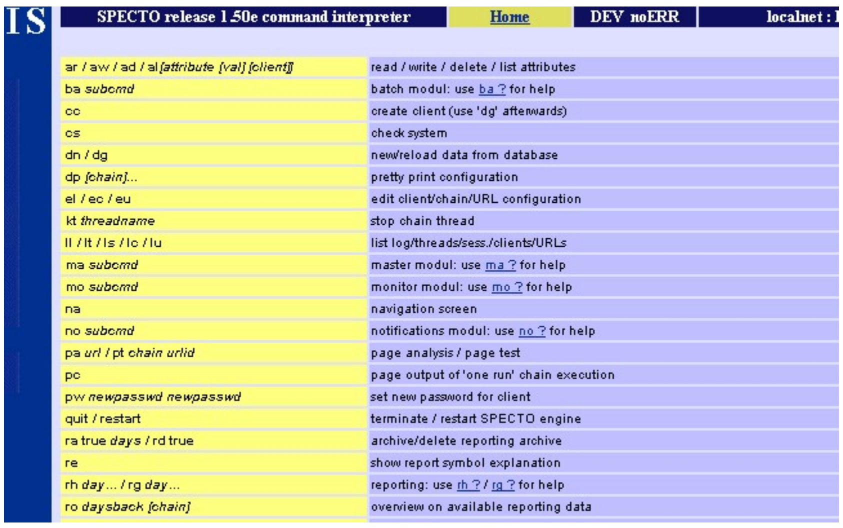
The users currently connected to a SPECTO engine can be verified using the ‘sl’ command.
Note: The web interface for the SPECTO engine is implemented as a ‘servlet’ which is separate to the SPECTO engine. This servlet may service several SPECTO engines and therefore also has to have some session information. The servlets session information may be requested using the ‘ls’ command.

Commands in detail

General

The SPECTO engine is command driven. Commands are usually created by the user interface but can also be entered directly in the command line (some special features are currently only available via the command line.
Guidelines for the command line :

- 'h' or '?' provides a command summary. (see screen shot below)
- Some commands can be called with the ' -?' option to explain themselves in detail.
- Submitting an empty command repeats the last command
- Avoid usage of special characters like " or '; they may result in strange behavior.



ar / aw / ad / al *[attribute [val] [client]]*	read / write / delete / list attributes
ba *subcmd*	batch modul: use ba ? for help
cc	create client (use 'dg' afterwards)
cs	check system
dn / dg	new/reload data from database
dp *[chain]...*	pretty print configuration
el / ec / eu	edit client/chain/URL configuration
kt *threadname*	stop chain thread
ll / lt / ls / lc / lu	list log/threads/sess./clients/URLs
ma *subcmd*	master modul: use ma ? for help
mo *subcmd*	monitor modul: use mo ? for help
na	navigation screen
no *subcmd*	notifications modul: use no ? for help
pa *url* / pt *chain urlid*	page analysis / page test
pc	page output of 'one run' chain execution
pw *newpasswd newpasswd*	set new password for client
quit / restart	terminate / restart SPECTO engine
ra true *days* / rd true	archive/delete reporting archive
re	show report symbol explanation
rh *day...* / rg *day...*	reporting: use rh ? / rg ? for help
ro *daysback [chain]*	overview on available reporting data

Overview of SPECTO commands (using command 'h')

Navigation

Command 'na' displays the tree-view guide to the SPECTO functionality. This is the preferred way to work with the SPECTO engine.

Object hierarchy

The 'el' command; which displays a list of all chains of the current client, is the best starting point for navigation through a SPECTO configuration.
Then the browser-links can be used to navigate within the chains and URLs. Alternatively the commands ***'ec <chain-id>'*** and ***'eu <chain-id> <URL-id'*** are available for navigation.

Create a new client

SPECTO is client aware. During login, the current client is selected by the user id. All commands related to client management require administrator privileges.

Available commands :

'cc' <client name> Creates a new client with the specified name. The new client already contains an initial chain. Also a user id / password (both with the same name as the client) are created. Currently a ***'dg'*** (reload client) command must be executed after the 'cc' command !

'sm' <client id> Switch to the specified client.

'lc' Shows a list of all defined clients.

'cn' <name> Rename a client (including its username and password).

Configure the client

SPECTO is client aware. During login, the user id selects the current client. The ***'el'*** command displays the current client's configuration, basically a set of chains. In the 'action' part of any chain, commands can be entered; valid commands are :

- add a new chain after the selected chain.
- delete the current chain
- analyze the page
- test the page configuration

Available commands :

'el' selects the list of chains for configuration.

Configure a chain

Any chain exists of an ordered set of URLs and an not ordered set of notifications. In the 'action' part of any URL or notification, commands can be entered; valid commands are :

'a' add a new URL after the selected URL.
'd' delete the current URL

Available commands :

'ec' <chain> selects the chain for configuration.

Configure a web page

In the 'action' part of any URL parameter/content, commands can be entered; valid actions are :

- adding a new chain after the selected chain.
- delete the current chain
- starting a chain processing as a new process

- stopping all processes of the chain

Available commands :
'eu' <chain> <URL> selects the URL for configuration.

Copying items

Clients, chains and URLs can be copied. For every item the target ('<to>') must exist and must have the name 'copy'. If a copy is issued from the web front end's 'action' field, the rename of the target to 'copy' must not be in the same transaction as the copy itself.

Available commands :

Copying a client
'dc' 'cl' <from> <to> copy client <from> into client <to>.

Copying a chain
'dc' 'ch' <from> <to> copy chain <from> into chain <to> within the current client.

Copying an URL
'dc' 'url' <fromChain> <fromURL> <toChain> <toURL> copy URL <fromURL> of chain <fromChain> into URL <toURL> of chain <toChain>.

Start/Stop processing a chain

Available commands :
'sc' <chain id> … starts processing of one or multiple chains. Multiple chains may be started with one 'sc' command by applying their ids separated by spaces.
'kt' <chain id> stops a chain process
'lt' list all active chain processes

'st save' store the current process list as startup (all processes which are currently active will be automatically started at next SPECTO start).
'st list' list the currently defined startup processes.
'st start' start the currently defined startup processes.
'st delall' delete all startup processes (reserved for administrator).

Reporting

SPECTO provides some basic reporting mechanisms and an archiving function.

Available commands :
'ra' <days-back>moves any data older than <days-back> from the reporting tables into the archive.
'rd' deletes all entries of the reporting archive
're' display the status codes definition
'ro' <days-back>' gives an overview on available data.

'rh' <days-back> <chain> [<type> [<limit> [<start-hour> [<num-hours>]]]]
]
provides a tabular representation in with one chain measurement per line; (the ***'rhl'*** variant provides a denser view). If the last digit of <type> is '1' a download option is provided; if 'type' is larger than 9 'type' div 10 is the number of days (default is one) to report. Instead of <days-back> a date in American or German syntax can be used; such dates must be quoted.

Example 1 : 'rh 1 0' reports yesterday's result of chain 0.
Example 2 : 'rh '23.12.00' 2 30' reports result of chain 2 from 12/23 to 12/25.

'rg' <days-back> <chain> <type> [<limit> [<start-hour> [<num-hours>]]]
gives a graphical representation. 'type' 0 is bar, 1 is line type display. If 'type' is larger than 9 'type' div 10 is the number of days (default is one) to report. Instead of <days-back> a date in American or German syntax can be used; such dates must be quoted.
<limit> overrides the automatically computed vertical axis with the supplied value. With <start-hour> / <num-hours> a part of the day may be selected.

Example 1 : 'rg 0 2 0' generates a drawing of today's result in 'bar' type format.
Example 2 : 'rg 7 1 31' generates drawings from 7 to 5 days back in 'line' type format.
Example 3 : 'rg '6/28/00' 0 1 5000 11 3' generates drawing of June 28th, chain 0 from 11:00 to 14:00 o'clock.

The colors of the generated graphs can be set using the 'Gcolor<Id>' attributes, where <Id> is : 0=Background left, 1=background right, 2=grid, 3=grid text, 4=line base, 5=line offset. The color is specified with <red>,<green>,<blue>.
To activate the new color setting for the current instance, the 'rg c' command is used.

Example : The following commands set the backgrounds to a light purple and a grey :
'aw GColor0 191,127,127'
'aw GColor1 127,127,127'
The color identifying an URLs data is read from attributes 'Gcolor<URLId+10>; if such an attribute entry is not defined, the color is computed by 'GColor4' + <URLId> * 'GColor5'.

The named parameter 'group' is used to specify a selection of URLs instead of using all URLs of the specified primary chain.
The syntax is : ***group={ [<chain>/]url; }***
If a url of -1 is specified the average of all urls is computed.

Example : The following commands set the backgrounds to a light purple and a grey :
'rg 1 0 1 group=0;1;2;1/6;'Yesterday: Urls 0,1 and 2 of chain 0, and url 6 of chain 1.
'rg 1 0 1 group=-1;' The average of chain 0 of yesterday.
'rg 1 0 1 group=-1;0;1/-1;'Yesterday: Average and url 0 of chains 0 and average of chain 1.

'rl' <days-back> <max-count> <chain>
List reporting entries of the last <days-back> days with max <max-count> entries of chain <chain>.

'rm' <months-back> <chain> <type> [<limit>]
gives a graphical summary of a complete month of measurement of a chain. This report is only available in SVG ('scalable vector graphics') format. If the browser is not enabled for SVG a download link for a (the Adobe) plug-in is automatically shown.

Analyzing or check a web page or whole chain

'pa' <URL> <include-source> analyses the page in reference to frames, forms, scripts, applets, links, fields and buttons. Also generates warning for unsupported constructs. See chapter 'Page analysis' for details.
If <include-source> is 'true' or '1' the source of the result page is included in the output.
'pt' <chain id> <url id> <include-source> check a page including all parameters and contents.
If <include-source> is 'true' or '1' the source of the result page is included in the output.
'sco' <chain id> run once a complete chain of URLs and preserve additional information (for debugging purposes).
'pc' display the result of the least chain 'run once'.

Attributes

Overall configuration of SPECTO is accomplished using attributes which are stored in the database. Attribute names are case insensitive but are stored bewaring case to allow for better formatting. The attributes are maintained using the following commands :

Available commands :

'aw' <name> <value> [<client>]	adds/replaces an attribute
'ar' <name> [<client>]	reads an attribute value
'al' [<client>]	list all attributes

Variables

During a chain execution local and global variables are used. Global variables must be preceded with an underscore '_'.

Available commands for global variables :

'vw' <name> <value>	adds/replaces a global variable
'vr' <name>	reads a global variable
'vd' <name>	deletes a global variable
'vl'	list all global variables

User defined command

Any parameterized command can be assigned a name and will be displayed as an entry in the 'user commands' section of SPECTO.

Available commands :

'ua' <name> <command>	adds a user specified command
'ud' <name>	removes the user specified command
'ul'	displays all user defined commands

User management

SPECTO contains two types of users :

- normal, to a specific client bound users, and the
- 'master' user.

Logon id and password of the master user are maintained in the attributes 'MasterUser' and 'MasterPassword'. The master user is not bound to a specific client, he can be attached to any client using the 'sm' command. The client which is initially attached to the master is specified in the attribute 'MasterClient'.

Available commands :

'lc'	list all clients.
'sl'	list current user sessions.
'sm' <client-id>	set master's client.
'pw' <passw.> <passw.>	set the logon password (new password must be supplied twice).

Notifications

Based on the status reported by the individual chains during processing, SPECTO maintains high/low watermark controlled counters of errors. Whenever the upper level is reached notifications are generated and remain in processing until the low watermark is reached.

Available commands (***'no ?'*** for online help) :

'no s'	list all error counters.
'no l'	list all active notifications.
'no a <chain> <err_type>	add a notification
'no d <chain>	remove all notifications of a chain

Import / Export

The configuration of a complete client can be exported and imported to/from files formatted in XML.

Available commands :

'xc <filename>' Exports the current client into file <filename>. The file is created in directory above the SPECTO install directory (e.g. if SPECTO is installed in '../projects/specto/' the file is created in '../projects/'.

'xci <filename> <clientNameInFile>'
Imports the specified file into the current client. <clientNameInFile> must match the client name specified in the import file (xml tag <clientName>).
Note: *The existing content of the current client will be overwritten by this command; it is advisable to export (command 'xc') the existing client before importing. Especially the administrator should check if he is in the correct client (command 'sm') before importing.*

'xg <filename>'
Exports the SPECTO global data into file <filename>

Other commands

'cs' reports the technical status of the system. Any erroneous state is flagged accordingly. In general such a state will require a restart of SPECTO.

'll <daysback> <from> <to> <lines> <tag>' displays the last entries written to SPECTO's log.

'lb [<lines>] displays the last entries written to SPECTO's log from a special buffer in memory (->faster than 'll').

'ca' cache statistics and management for SPECTO documents (scripts, text areas, documentation).

'ms' <to> <subj.> <msg.> send a email message (test case)

web recorder I

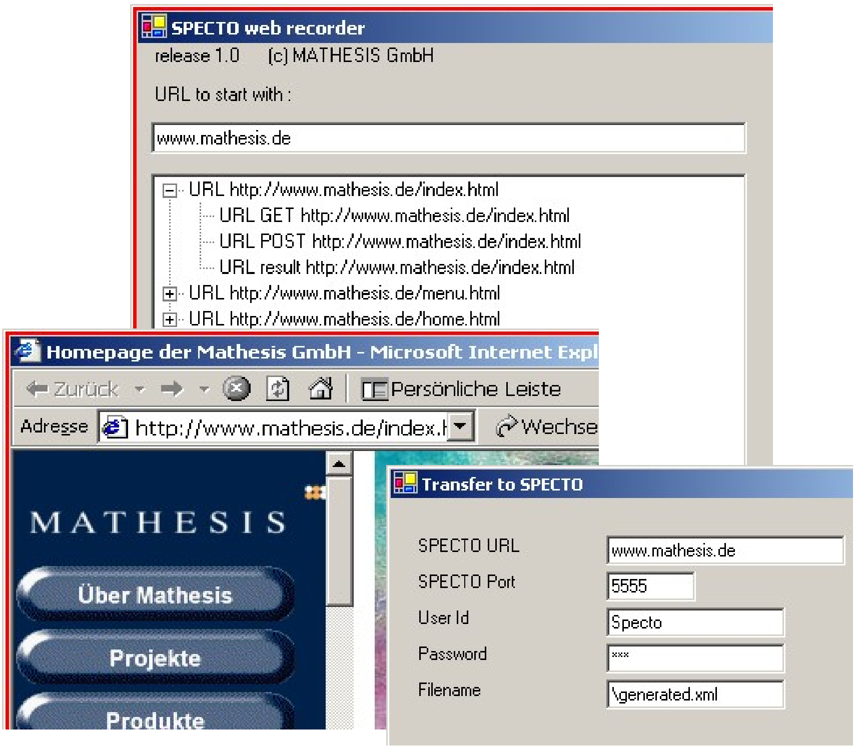
Overview

Web recorder I is not supported any longer, please refer to the SPECTO Interceptor.
The separate SPECTO web recorder is available to assist in recording sessions of user interaction on the web. The results are stored in the SPECTO configuration XML format and can be imported in any SPECTO chain.

The SPECTO web recorder is only available for the windows platform.

Usage

The SPECTP web recorder monitors an Microsoft Internet Explorer (rel. 5.0 or higher). No special treatments are necessary.



www.mathesis.de

```
URL http://www.mathesis.de/index.html
    URL GET http://www.mathesis.de/index.html
    URL POST http://www.mathesis.de/index.html
    URL result http://www.mathesis.de/index.html
URL http://www.mathesis.de/menu.html
URL http://www.mathesis.de/home.html
```

http://www.mathesis.de/index.h

SPECTO URL	www.mathesis.de
SPECTO Port	5555
User Id	Specto
Password	***
Filename	\generated.xml

Documentation

- There is a separate document for installation and usage of the SPECTO web recorder

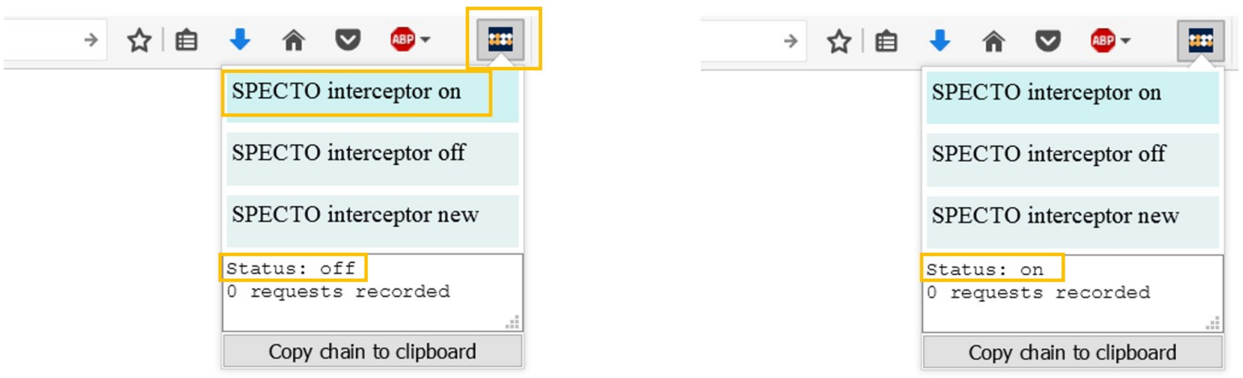
SPECTO ‘Interceptor’ web recorder

Overview

The separate SPECTO ‘Interceptor’ web recorder is a plug-in for common web browsers and allows the recording of a session with a website, filter the recording and transfer them into a SPECTO ‘chain’.
There is a separate documentation for installing and using the SPECTO ‘Interceptor’.

Usage

After installation the SPECTO Interceptor is accessible within the browser’s title frame by its own icon:



```
Status: off
0 requests recorded
```

```
Status: on
0 requests recorded
```

In the browser, by clicking on the interceptor icon, open the interceptor main menu.
The status should display as ‘off’
Select the ‘SPECTO interceptor on’ menu item
The status should change to ‘on’ with 0 recorded items.

Document release 2.2

During a web session the Interceptor may be turned on and off at will; also, it can be reset, clearing the recordings up to now.
After the session is completed, the recorded URLs can be filtered (e.g. to remove CSS and web fonts) and the remaining information, formatted in SPECTO’s chain configuration XML format, exported into the clipboard.

Installation

Your SPECTO system is shipped completely installed

You have to provide :

- An IP address and, if available, a DNS name
- A SMT server name or address

Configuration of the IP address :

Windows NT platform

Change to :

1. Windows NT control panel
2. Network
3. Protocol stack
4. TCP/IP protocol
5. Properties

Set the IP-address, IP-mask and gateway address according to your environment.

UNIX platforms

Set the IP-address, IP-mask and gateway address according to your environment.

Configuration of the Java keystore :

The Java keystore is (among others) used to provide a certificate for incoming SSL connections to the SPECTO internal web server (see command 'ws'). A keystore has to be generated once; as an easy approach use in the operating systems shell the command (from the java home directory) :

```
bin\keytool -genkey -keyalg "RSA" -keystore keystore -storepass 123456
```

The SPECTO customizing may have to adapted using the SPECTO command: **`cu cert`**

Databases

Relational databases play two roles within SPECTO. They are used for persistent storage of SPECTO configuration and reporting data, and they can be targets of ('sql:')-URLs. In both cases access of the database is via JDBC drivers.

This chapter details configuration information for the different databases supported by the SPECTO engine.

ORACLE

Overview

ORACLE is the market leader in relational databases. SPECTO supports all ORACLE editions (including 'personal edition' and 'Oracle lite') and versions. Installation of an ORACLE database is not described here; the supplied standard installation procedures shall be used.

Executing SPECTO on ORACLE databases

Start the specto engine using the following command :

```
java -Xms<memory> -cp Specto\Specto.jar Specto.SpectoMain <CreateDB-
option> DBCon=<db_type> DBUser=<user> DBPassword=<password>
DBName="thin:@<server>:<port>:<database>"
COMSocket=<socket_to_servlet>
```

With <db_type> :

- 5 : ORACLE on windows
- 6 : ORACLE on UNIX
- 7 : ORACLE lite on windows
- 8 : ORACLE lite on UNIX

Example:

```
java -Xms4000000 -cp Specto\Specto.jar Specto.SpectoMain CreateDB
DBCon=5 DBUser="usr" DBPassword="pwd"
DBName="thin:@localhost:1521:orcl" COMSocket=5555
```

The 'CreateDB' Parameter is only required during the first execution on a newly created database. It may (and should) be omitted on later executions.

Microsoft SQL Server

Overview

SPECTO supports the commercial and the (free) MSDE release of SQL Server. MSDE (Microsoft SQL Server 2000 Desktop Engine (MSDE 2000), Version A) is available as a package named '`<Country>_MSDE2000A.exe`' (e.g. '`GER_MSDE2000A.exe`' for German language) from Microsoft's web site (search for MSDE in the download section).

Installation of MSDE

During installation of the server the administrative user '`sa`' will be assigned a password ('specto') and, besides 'window authentication' also 'sql authentication' will be enabled:

- Download the ..._`MSDE2000A.exe` file
- Execute the file. Execution will self extract the installation files into a new (not the program) directory. This directory may be deleted after the installation.
- Using a command window change into the above directory and execute the following command : `setup SAPWD="specto" SECURITYMODE=SQL`
- Configure network by '`svrnetcn`.exe' in the '`80\Tools\binn`' subdirectory of the program directory of the installed MSDE; add protocol TCP/IP
- Start sql server by executing '`sqlmangr`.exe' in the '`80\Tools\binn`' subdirectory of the program directory of the installed MSDE.

Database creation using '`osql`'

After installation the tool '`osql.exe`' located in the '`80\Tools\binn`' subdirectory of the installed MSDE, or '`Enterprise Server Manager`' of the commercial SQL Server release can be used to create the SPECTO database and account.
Using '`osql.exe`' any of the following commands has to be executed in a command window by prefixing the command with '`osql.exe -U sa -P specto -Q` '.

- Create the SPECTO database : '`create database specto`'
  (enter: '`osql.exe -U sa -P specto` –Q "`create database specto`").

If the standard user ('`sa`') is not to be used, the following steps are required:

- Change to the new database : '`use specto`'
- Create an account : '`sp_addlogin 'specto', 'sol', 'specto'`'
- Grant access : '`sp_grantdbaccess 'specto', 'specto'`'
- Add to owner role : '`sp_addrolemember 'db_owner', 'specto'`'
- Add to ddl role : '`sp_addrolemember 'db_ddladmin', 'specto'`'

Executing SPECTO on SQL Server databases

Start the SPECTO engine (usually via script) using the following command :

```
java -Xms<memory> -cp Specto\Specto.jar Specto.SpectoMain <CreateDB-option>
DBCon=10 DBUser=<user> DBPassword=<password> DBName="//<server>:<port>;
SelectMethod=cursor;databasename=<database>" COMSocket=<socket_to_servlet>
```

Example:

```
java -Xms4000000 -cp Specto\Specto.jar Specto.SpectoMain CreateDB DBCon=10
DBUser="sa" DBPassword="specto"
```

```
DBName="//localhost:1433;SelectMethod=cursor; databasename=specto"
COMSocket=5555
```

Sybase Advanced Server

Overview

Sybase Advanced Server is a widely distributed database engine which also has been the root of Microsofts SQL server. Installation of a Sybase ASE database is not described here; the supplied standard installation procedures shall be used.

Executing SPECTO on Advanced Server databases

Start the specto engine using the following command :

```
java –Xms<memory> -cp Specto\Specto.jar Specto.SpectoMain <CreateDB-
option> DBCon=13 DBUser=<user> DBPassword=<password>
DBName="//<server>:<port>/<database>" COMSocket=<socket_to_servlet>
```

Example:

```
java -Xms4000000 -cp Specto\Specto.jar Specto.SpectoMain CreateDB
DBCon=13 DBUser="usr" DBPassword="pwd"
DBName="//localhost:1527/specto" COMSocket=5555
```

The 'CreateDB' Parameter is only required during the first execution on a newly created database. It may (and should) be omitted on later executions.

MySQL

Overview

MySQL is the best known open source database. SPECTO supports all MySQL releases in URL ('sql:') checking. As its underlying database SPECTO requires MySQL in release 4 because of the support for transactions. Installation of a MySQL database is not described here; the supplied standard installation procedures shall be used.

Executing SPECTO on cloudscape databases

Start the specto engine using the following command :

```
java –Xms<memory> -cp Specto\Specto.jar Specto.SpectoMain <CreateDB-
option> DBCon=15 DBUser=<user> DBPassword=<password>
DBName="//<server>:<port>/<database>" COMSocket=<socket_to_servlet>
```

Example:

```
java -Xms4000000 -cp Specto\Specto.jar Specto.SpectoMain CreateDB
DBCon=15 DBUser="usr" DBPassword="pwd"
DBName="//localhost:1527/specto" COMSocket=5555
```

The 'CreateDB' Parameter is only required during the first execution on a newly created database. It may (and should) be omitted on later executions.

SAP DB / MAX DB

Overview

-.

Executing SPECTO on cloudscape databases

Start the specto engine using the following command :

```
java -Xms<memory> -cp Specto\Specto.jar Specto.SpectoMain <CreateDB-
option> DBCon=<db_type> DBUser=<user> DBPassword=<password>
DBName="//<server>:<port>/<database>" COMSocket=<socket_to_servlet>
```

With <db_type> :

- 11 : SAP DB on windows
- 12 :SAP DB on UNIX

Example:

```
java -Xms4000000 -cp Specto\Specto.jar Specto.SpectoMain CreateDB
DBCon=12 DBUser="usr" DBPassword="pwd"
DBName="//localhost:1527/specto" COMSocket=5555
```

The 'CreateDB' Parameter is only required during the first execution on a newly created database. It may (and should) be omitted on later executions.

Hypersonic

Overview

Hypersonic is a relational database implemented entirely in JAVA. If used as the underlying SPECTO database there are some minor limitations concerning SPECTO's automatic upgrade of database tables.

Executing SPECTO on cloudscape databases

Start the specto engine using the following command :

```
java –Xms<memory> -cp Specto\Specto.jar Specto.SpectoMain <CreateDB-
option> DBCon=17 DBUser=<user> DBPassword=<password>
DBName="//<server>:<port>/<database>" COMSocket=<socket_to_servlet>
```

Example:

```
java -Xms4000000 -cp Specto\Specto.jar Specto.SpectoMain CreateDB
DBCon=17 DBUser="usr" DBPassword="pwd"
DBName="//localhost:1527/specto" COMSocket=5555
```

The 'CreateDB' Parameter is only required during the first execution on a newly created database. It may (and should) be omitted on later executions.

Cloudscape

Overview

Cloudscape is a relational database implemented entirely in JAVA. Originally developed by Informix, with the buy of Informix by IBM it moved to IBM and was recently (with release 10) made 'open source' by IBM and given to the Apache group for future maintenance and development.

Database engine

The cloudscape database is delivered with an installation application. But the installation of the pure database engine can also be accomplished manually by executing the following steps :

1. Download the most recent cloudscape distribution from :
   http://www-306.ibm.com/software/data/cloudscape/ (will be apache soon)
2. Unpack it into a temporary location (may be discarded later)
3. Create a new directory (which will hold the database(s))
4. Copy the library files ‚cs.jar' and ‚csnet.jar' into any directory (e.g. the one created in the previous step, or the JDK/JRE lib directory)
5. Assure that the JDK/JRE 'bin' directory is in the systems execution path
6. Start the cloudscape server using the ‚java' command with specifying the java class path to the two library (.jar) files copied from the download and executing ‚**`com.ihost.cs.drda.NetworkServerControl start`**' (see sample startup script below).

Sample startup script (windows environment) :

```
rem startup for cloudscape database server
set CSLIBS=C:\Programme\IBM\Cloudscape_10.0\lib\
java -cp %CSLIBS%cs.jar;%CSLIBS%csnet.jar com.ihost.cs.drda.NetworkServerControl start
```

Note: If the database server shall also be available via the network (as started above it is only available for clients on the same machine) the parameter (**`-h 0.0.0.0`**) has to be appended to above start command.

Note: Above script, with 'start' replaced by 'shutdown' may be used to stop the cloudscape server.

Enabling SPECTO for access of Cloudscape databases

Copy the java library files (from the download of the chapter before) 'cs.jar', csnet.jar', 'db2jcc.jar' and 'db2jcc_license_c.jar' to the directory where the other SPECTO additional libraries are located (usually the JRE's lib/ext directory).

Accessing of Cloudscape databases with SPECTO URLs

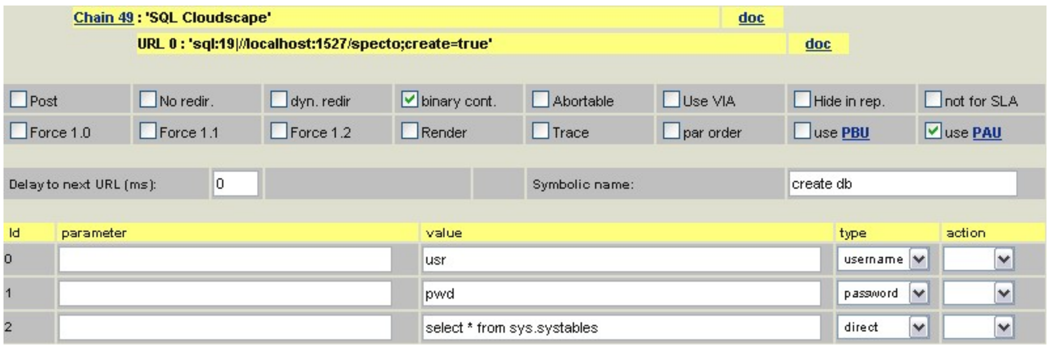
SPECTO provides the 'sql:' identifier to execute SQL statements on relational databases (see chapter 'sql (relational database) access' for details).
The 'sql:' id for cloudscape servers is '19'; the complete 'sql:' format is :
`sql:19|<servername>:<port>/<database>{options}`

Example : `sql:19|//localhost:1527/sample`

Creating a cloudscape database for SPECTO

A cloudscape database is represented as a subdirectory below the cloudscape directory (the directory where *startup* was executed).
Though SPECTO is able to initially create all its required tables and content during startup (parameter 'CreateDB'), the database itself has to be created before. Such an empty cloudscape database (the subdirectory structure) can be requested from NLS, or created by an (already running) SPECTO engine. For the later, the following steps are required :

1. The cloudscape database engine must be installed and running (see 'Database engine')
2. Within SPECTO create a new chain with one URL of the format : **`sql:19|//localhost:1527/specto;create=true`** ('specto' is the name of the new database, this may be changed). Also supply parameters for username, password and an SQL command to be executed on the newly created database as shown in the screen dump below :



Chain 49 : 'SQL Cloudscape' doc

URL 0 : 'sql:19|//localhost:1527/specto;create=true' doc

☐ Post	☐ No redir.	☐ dyn. redir	✓ binary cont.	☐ Abortable	☐ Use VIA	☐ Hide in rep.	☐ not for SLA
☐ Force 1.0	☐ Force 1.1	☐ Force 1.2	☐ Render	☐ Trace	☐ par order	☐ use PBU	✓ use PAU

Delay to next URL (ms): 0 Symbolic name: create db

Id	parameter	value	type	action
0		usr	username	
1		pwd	password	
2		select * from sys.systables	direct	

3. Execute the chain once (using 'one run'). The 'source' should show a listing similar to :

```
TABLEID TABLENAME TABLETYPE SCHEMAID LOCKGRANULARITY
80000010-00d0-fd77-3ed8-000a0a0b1900 SYSCONGLOMERATES S 8000000d-00d0-fd77-3ed8-000a0a0b1900 R
80000018-00d0-fd77-3ed8-000a0a0b1900 SYSTABLES S 8000000d-00d0-fd77-3ed8-000a0a0b1900 R
8000001e-00d0-fd77-3ed8-000a0a0b1900 SYSCOLUMNS S 8000000d-00d0-fd77-3ed8-000a0a0b1900 R
80000022-00d0-fd77-3ed8-000a0a0b1900 SYSSCHEMAS S 8000000d-00d0-fd77-3ed8-000a0a0b1900 R
8000002f-00d0-fd77-3ed8-000a0a0b1900 SYSCONSTRAINTS S 8000000d-00d0-fd77-3ed8-000a0a0b1900 R
…
```

4. Inspect the cloudscape base directory to ensure that there has been a new subdirectory 'specto' created.

Executing SPECTO on cloudscape databases

Start the specto engine using the following command :

```
java -Xms<memory> -cp Specto\Specto.jar Specto.SpectoMain <CreateDB-
option> DBCon=19 DBUser=<user> DBPassword=<password>
DBName="//<server>:<port>/<database>" COMSocket=<socket_to_servlet>
```

Example:

```
java -Xms4000000 -cp Specto\Specto.jar Specto.SpectoMain CreateDB
DBCon=19 DBUser="usr" DBPassword="pwd"
DBName="//localhost:1527/specto" COMSocket=5555
```

The 'CreateDB' Parameter is only required during the first execution on a newly created database. It may (and should) be omitted on later executions.

Running SPECTO

Running the database

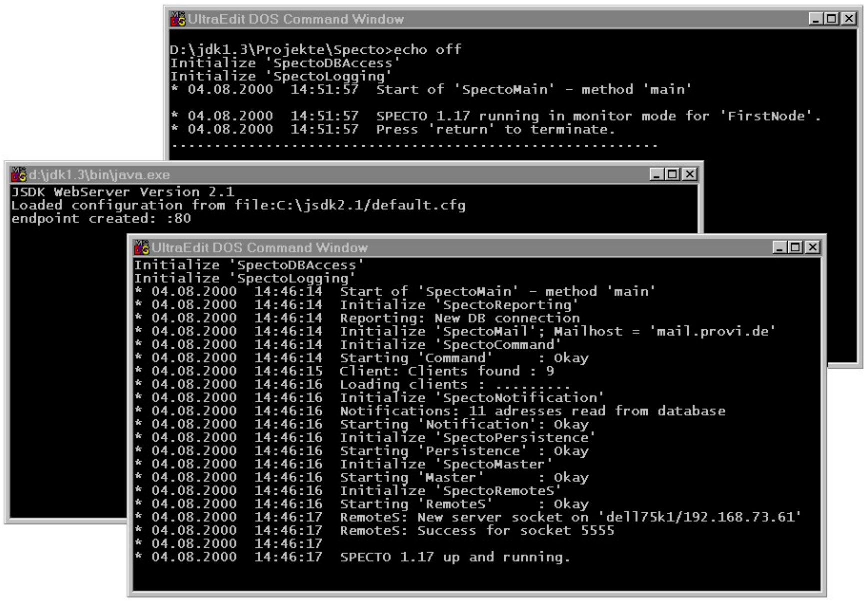
Start/Stop procedures of the database used by the SPECTO engine depends on the type of the database used. Please see the vendors manual for instructions. SPECTO related database specific issues are explained in the preceding chapter.

Starting the external ('tomcat') web server / servlet container

The external 'tomcat' web server is started by double clicking the 'Start WWW Server' icon on the desktop. A command window will appear. After some seconds; another window with the lines 'JSDK WebServer...endpoint created: :80)' will appear; the first window will disappear.
The 'JSDK WebServer' window will stay; it may be iconized.

Starting SPECTO

SPECTO is started by double clicking the 'SPECTO' icon on the desktop. A command window will appear and SPECTO will be executed in that window. If another SPECTO instance is already loaded, the new SPECTO will go into Backup mode for the running SPECTO.



```
D:\jdk1.3\Projekte\Specto>echo off
Initialize 'SpectoDBAccess'
Initialize 'SpectoLogging'
* 04.08.2000  14:51:57  Start of 'SpectoMain' - method 'main'

* 04.08.2000  14:51:57  SPECTO 1.17 running in monitor mode for 'FirstNode'.
* 04.08.2000  14:51:57  Press 'return' to terminate.
.............................................................
```

```
JSDK WebServer Version 2.1
Loaded configuration from file:C:\jsdk2.1/default.cfg
endpoint created: :80
```

```
Initialize 'SpectoDBAccess'
Initialize 'SpectoLogging'
* 04.08.2000  14:46:14  Start of 'SpectoMain' - method 'main'
* 04.08.2000  14:46:14  Initialize 'SpectoReporting'
* 04.08.2000  14:46:14  Reporting: New DB connection
* 04.08.2000  14:46:14  Initialize 'SpectoMail'; Mailhost = 'mail.provi.de'
* 04.08.2000  14:46:14  Initialize 'SpectoCommand'
* 04.08.2000  14:46:14  Starting 'Command'      : Okay
* 04.08.2000  14:46:15  Client: Clients found : 9
* 04.08.2000  14:46:16  Loading clients : .........
* 04.08.2000  14:46:16  Initialize 'SpectoNotification'
* 04.08.2000  14:46:16  Notifications: 11 adresses read from database
* 04.08.2000  14:46:16  Starting 'Notification': Okay
* 04.08.2000  14:46:16  Initialize 'SpectoPersistence'
* 04.08.2000  14:46:16  Starting 'Persistence' : Okay
* 04.08.2000  14:46:16  Initialize 'SpectoMaster'
* 04.08.2000  14:46:16  Starting 'Master'       : Okay
* 04.08.2000  14:46:16  Initialize 'SpectoRemoteS'
* 04.08.2000  14:46:16  Starting 'RemoteS'      : Okay
* 04.08.2000  14:46:17  RemoteS: New server socket on 'dell75k1/192.168.73.61'
* 04.08.2000  14:46:17  RemoteS: Success for socket 5555
* 04.08.2000  14:46:17
* 04.08.2000  14:46:17  SPECTO 1.17 up and running.
```

If started manually it may be useful to increase the process priority of the SPECTO engine (using task manager on Windows 32 or the command shell on UNIX).

Startup parameters

The SPECTO engine may be parameterized using startup parameters on the command line. Usually this is done in a startup script ('init.cmd' in a Windows, 'init' in an UNIX environment, the startup script is located in the SPECTO home directory).

The command line to start SPECTO consists of :

- the java virtual machine name
- parameters to the java virtual machine (may be empty)
- the SPECTO main class file name
- the startup parameters (may be empty)

Available startup parameters concerning database and front end connection (the parameters are case sensitive, they have to be applied exactly as shown in the table) :

Parameter	**Description**
`DBCon`	Connection type to database : 0 = JDBCviaODBC, 1/2 = JDBC-SOLID (Windows/UNIX), 3/4 = JDBC-JDataStore, 5/6 = JDBC-ORACLE, 7/8 = JDBC-ORACLE Lite 10 = MS SQL Server (MS JDBC driver)
`DBName`	Database name (ODBC name if DBCon is 0)
`DBUser`	UserId to connect to the database
`DBPassword`	Password for above UserId
`ComSocket`	TCP/IP port number on which the SPECTO engine will listen to commands. Default is 5555; if the specified port is used the next higher ports (up to 20) are tried.

If startup parameters are not supplied the following defaults are assumed by the SPECTO instance :

Parameter	**Description**
`DBCon`	0
`DBName`	
`DBUser`	specto
`DBPassword`	sol
`ComSocket`	5555

Example (SPECTO on MS SQL server) :

```
java  -Xms4000000  Specto.SpectoMain  DBCon=10  DBUser="spectod"  DBPassword="sol"
DBName="//localhost:1433;SelectMethod=cursor;databasename=spectod" COMSocket=5555
```

The current configuration of a SPECTO engine can be read by using the '`cs`' command.

The SPECTO engine's operations mode can be tailored using additional optional command line parameters at startup (the parameters are also case sensitive, they have to be applied exactly as shown in the table) :

Parameter	**Description**
`CreateDB`	In the underlying database create all tables needed by the SPECTO engine. Existing tables are not re-created, existing data is not modified. This parameter is usually used to create SPECTO tables at first invocation of the engine; it should but need not be removed after successful creation of the tables. The built-in SPECTO mechanism for updating of table structures during release changes is not affected by this parameter. See database specific issues in the preceding chapter.
`BaseDir`	Initial setting of the base directory used for data exchange with the web server. If applied also the '`WebBaseDir`' and '`TempBaseDir`' values are derived from this value. All xxxDir values are saved into the equivalent attributes (overwriting existing values).
`Options`	A list of character size entries determining minor SPECTO issues. Currently implemented : 'c' : disable console input 'h' : AWT 'headless' mode (Linux without X11)
`Continue`	Activates 'continue' mode. During 'continue' mode the new SPECTO engine is starting up in silent mode. After start up the run image (threads, notifications, sessions, global data) stored by the last SPECTO engine run is read (from cluster entry '`specto.runimage`' in client 0) and restored. 'continue' mode is automatically exited (switching to normal SPECTO operation) when the run image is fully operational.
`ClearContinue`	Clears a pending 'continue' mode (a 'continue' mode request which was set at exit of the last engine run).
`Mirror`	Activates 'mirror' mode. During 'mirror' mode a second SPECTO engine instance synchronizes with an already running SPECTO engine instance. The mirroring node works on the same database and starts the same threads as the mirrored node but does not write log or result entries into the database. 'mirror' mode is automatically exited (switching to normal SPECTO operation) when the mirrored engine instance is going down; it may be turned off manually using command '`mirror off`'.
`Maintenance`	Activates 'maintenance' mode. During 'maintenance' mode a SPECTO engine instance operates on the same database as another already running SPECTO instance. During 'maintenance' mode no log or result entries are written to the database. Also no Reporting cache pre-fetching, no auto start scripts, no startup threads and no batch processes are activated.
`NoNotif`	Disables activation of notifications. If `NoNotif` is specified no notification status nor notifications are computed during the amount of seconds specified. Notification processing can be manually reactivated using command '`no activate true`'.
`NoStartup`	Disables starting of defined startup threads.
`WebServer`	forces generation of an active internal web server instance on port 80. This is usually used during initial start of a new SPECTO engine without an external ('tomcat') web server. This parameter also creates a matching entry in the attributes list so that the parameter is not needed for consecutive starts of the SPECTO engine.

Stopping/restarting SPECTO

'SPECTO' is stopped by issuing the '`quit true`' or '`restart true'` commands. The SPECTO engine notifies about the success of stopping the individual services. Note that waiting for threads to terminate and open changes to be written to the database may take some time.

If the '`quit`' or '`restart`' command is issued with the optional parameter '`nice`' then the engine waits for all chain threads to finish the current chain measurement.

If the '`quit`' or '`restart`' command is issued with the optional parameter '`silent`' then the SPECTO shutdown messages are not written into the database log table.

If the '`restart`' command is issued with the optional parameter '`continue`' then the restart of the SPECTO engine will be in 'continue' mode (see chapter *Startup parameters*). This also enforces 'silent' mode.
Note: The startup parameter '`ClearContinue`' may be used to override an accidentally set 'continue' mode.

High availability operation (mode I : 'shared hardware')

In this availability scenario two 'SPECTO' instances are running on the same hardware and are using the same (remote) database and servlet ('tomcat') engine. The instances are named 'active' and 'passive'. The active instance is operated identically to a standard SPECTO instance; the passive instance is started in 'mirror' mode (see above parameter description).

Before takeover :

- Verify that the two instances are in sync (if in doubt save the runimage on the active instance and then restart the passive instance)
- verify that the results are prefetched

At takeover :

- shut down the active instance

The previous active instance may now be stopped.

High availability operation (mode II : 'shared database')

In a high availability scenario two 'SPECTO' instances are running on different hardware and are using the same (remote) database. The instances are named 'active' and 'passive'. The active instance is operated identically to a standard SPECTO instance; the passive instance is started in 'mirror' mode (see above parameter description).

Before takeover :

- Verify that the two instances are in sync (if in doubt save the runimage on the active instance and then restart the passive instance)
- verify that the results are prefetched

At takeover :

- shut down the active instance

The previous active instance may now be stopped.

High availability operation (mode III : 'unshared')

In this high availability scenario two 'SPECTO' instances are running on different hardware and databases. The instances are named 'active' and 'passive'. The active instance is operated identically to a standard SPECTO instance; the passive instance is started in 'mirror' mode (see above parameter description).

Release upgrade in a HA environment

In a high availability environment a 'SPECTO' downtime is not tolerable. Therefore release upgrades have to be performed using two instances, named 'C' (*current*) and 'I' (*intermediate*).

Note : The intermediate instance is assumed in a different directory or on a different machine to prevent problems with locked files of the code base.

Preparation :
Check for running 'maintenance' and 'backup' instances and shut them down (may block access to the new code base otherwise).

The recommended *manual* procedure is :

I: (Is assumed running) Prepare for the newest code release : '**`engine f`**'
C: Save current internal status : '**`runimage s`**'
I: Restart with new code base in 'continue' and 'mirror' mode : '**`restart true continue mirror`**'.
Note: Caches are not used chains are running 30 s shifted
I: Wait until running stable / Check functionality of new code base
C: Cleanly stop all running chains : '**`kta nice`**'
I: Terminate mirror mode (measurements will go into the database, notifications will execute) : '**`mirror off`**'
I: Save current image : '**`runimage s`**'
C: Prepare for new code base : '**`engine f`**'
C: Restart in 'continue' and 'mirror' mode : '**`restart true continue mirror`**'
Note: Caches are filled, chains are running unshifted
C: Wait until running stable
I: Cleanly terminate intermediate instance : '**`quit true nice`**'
C: Terminate mirror mode : '**`mirror off`**'

The optional *automatic* procedure is (beta yet) :

I: Check feasibility : '**`engine upgrade check`**'
Note : Verifies that the *current* node is reachable
C: Enable external controlled upgrade: '**`engine upgrade allow`**'
I: Upgrade *current* instance : '**`engine upgrade true`**'

Post operation :
Restart 'maintenance' and 'backup' instances

Stopping the web server

Double click the 'Stop WWW Server' icon on the desktop. A command window will appear. After same seconds the existing 'JSDK WebServer' window will show a termination message and disappear; then the initial command window will disappear. If the above procedure does not work, it is necessary to select the 'JSDK WebServer' window and (multiple times) press control-C.

Stopping the database

See the database vendors manual for instructions on how to safely shut down the database. There are no SPECTO specific issues with that.

Advanced topics

Running multiple SPECTO instances on one machine

Multiple instances of a SPECTO engine can be run on one computer system without disturbing each other. This may be helpful if a development system is required or if (because a native threaded java vm is not available) load distribution is required on a multiple cpu system.

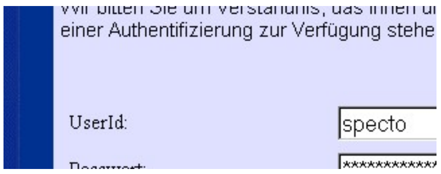
Database :
For every SPECTO instance a separate database engine is required. Sample configuration files are available upon request.

SPECTO engine :
Every SPECTO instance must be connected to its own database instance and must listen to a unique port.
This configuration is done using startup parameters in the startup script ('init.cmd' or 'init' in the SPECTO home directory) :
Example for two SPECTO instances using JDBC/ODBC interfaces and listening on ports 5555 (the default) and 5554 :

```
java Specto.SpectoMain DBName="ShMem Solid" COMSocket=5555
java Specto.SpectoMain DBName="Solid2 via net" COMSocket=5554
```

Web server :
Only one 'SPECTO web server' is required. Login on to a SPECTO other than the default is accomplished by specifying the connection socket after (separated by a space) the service name.

Example : 'specto 5554' would connect to a SPECTO instance listening on port 5554.


einer Authentifizierung zur Verfügung stehe

UserId: specto

****

Duplicating a SPECTO instance into foreign databases

With the 'be <filename>' command, the content of a SPECTO database can be exported as SQL 'INSERT' statements, which can be used to import the data in any other relation database.
'be' does not export the reporting, archive and logging data. If this is required, add an 'r', 'a' or 'l' character to the be command ('beral.' would export all data).

Attributes summary

SPECTO is configured by attributes. Attributes are permanent (stored in the database) and can be set using the '*aw <name> <value> [<client>]*' command or with the 'attribute editor' (commands '*ae*' or '*ae <prefix>*').

The SPECTO standard attributes (list below) are usually set during customizing. Therefore it is preferred to use the customizing modules from the menu (section 'customizing').

Some attributes require the SPECTO engine to be restarted.

It is permitted to add own attributes. Attributes can also be read from the SPECTO script languages (for security reason a '_' will automatically be prefixed to the attribute name).

Name	description	examples, comments
ActionOnError	Name of an operating system command to be executed whenever an URL runs into an error	tracert >tracert.log
AmAlive	If 'true' send an 'am alive' signal.	false
AmAliveAddress	Address to which to send the 'am alive' signal	specto@NLS.de
AmAliveType	Type of transport of 'am alive' message; like notification type, e.g. 'e' is email.	e
BaseDir	SPECTO installation base directory	
CCMSActivation	See chapter 'SAP R/3 CCMS'.	false
CCMSClient	See chapter 'SAP R/3 CCMS'.	000
CCMSHost	See chapter 'SAP R/3 CCMS'.	Sapr3.de
CCMSSystem	See chapter 'SAP R/3 CCMS'.	59
CERTKeyFile	See section 'authentication by certificates'.	
CERTKeyStore	See section 'authentication by certificates'.	
CERTKeyStorePwd	See section 'authentication by certificates'.	specto
CERTKeyTool	See section 'authentication by certificates'.	
CSVSeparator	The character used as separator during generation of CSV ('comma separated values') files.	,
ConsoleFormat	Format of messages on the SPECTO console (0=only message, 1 = time + message, 2 = date + time + message)	2
ConnectTimeout	Maximum wait time for opening of connections. In milliseconds	50000
DefaultScriptEngine	SpectoScript 'ss', JavaScript ('js').	js
DefaultUpperLimit	Default high value (in milli-seconds)	5000

	for graphical represen-tation of results	
DoLog	Enables writing of log messages into the permanent log	true (default)
ExpandActiveBranch	Preset of the ‘expandActive-Branch’ option in the navigation screen	false (default)
ExportBaseFilename	Prefix of the filename used for automatic exports.	AutoExport
ExportHour	Begin hour of export.	0
FAXPassword	Password for FAX service provider	
FAXProvider	Email address of provider	fax@NLS.de
FAXUsername	Username for FAX service provider	
FirstCommand	SPECTO command to be executed after logon.	el
FormMethod	HTTP method used to submit HTML forms	POST
Gcolor0 – 5	Permanent storage for coloring of graphics (see ‘Reporting’). 0/1 = backgrounds 2 = arrows, 3 = text, 4 = line base, 5 = line delta. Format = red,green,blue	128,64,0
GUIColor0 – 9	Permanent storage for coloring of the GUI (see command ‘co’). Format is #rrggbb (hex)	#804000
GUILinkButtons	Use of HTML buttons instead of links.	false
GUITextRows	Number of rows in the document window	18
GUIType	1 = single, 2=frames, 3=frames+javascript	1
HostsBaseDir	Directory of the ‘hosts’ file	C:\windows\system32\drivers\etc\
LastReceivedMessage	Used internally to remember the timestamp of the last received SpectoNet message	
LimitWriteToFirst	If multiple instances of a chain are started, if ‘true’ then only the first one writes entries into the result tables.	true
LogLineSize	Maximum number of characters in a log line	79
MailDebug	Enables debug messages to the console within the mail module	false (default)
MailFrom	Sending account for email Messages	
MailHost	SMTP mailhost used for sending and receiving of messages	mail.NLS.de
MailHostSOAP	Address of system providing SOAP	http://www.NLS.de/Specto

	based email service.	/SpectoHome
MailLastUpdate	Used internally to remember the timestamp of the last SpectoNet message poll	
MailMbox	Name of the mailbox	INBOX (default)
MailPassword	Password for authorization against the mail server	specto
MailPopBeforeSmtp	If ‘true’ send any SMTP action is prefixed with a POP read (for authentification)..	false
MailProtocol	Mail protocol used for reading of mail messages	pop3 (default)
MailReadPeriod	Period of fetching new mail from the mail server (in seconds)	300
MailReceiveEnabled	Enables receiving of SMTP messages	false (default)
MailReceiveHost	SMTP mailhost for receiving of messages (if different from ‘MailHost’	
MailReceivePassword	Password for the mail inbox (if different from ‘MailPassword’)	
MailReceiveUser	Username for the mail inbox (if different from ‘MailUser’)	
MailViaSOAP	If ‘true’ outgoing email is forwarded via a SOAP service instead of using a SMTP access.	false
MailBackupViaSOAP	If ‘true’ outgoing email is forwarded via a SOAP service in case the primary SMTP email service fails.	false
MailUser	Username for authorization against the mail server	spectomaster@NLS.de
MasterClient	Default client for the administrative account.	1
MasterPassword	Password of the administrative account	Specto
MasterUser	Name of the administrative account	Specto
MCConfiguration	Configuration of the master console. Should not be altered manually, use ‘mcc’ command.	tsnpgnpunhlnnnn
MonitorDisable	Specifies if the monitor functionality for checking running chains is disabled	False (default)
NetDebug		
NotificationAddError	If any URL failure within a chain will increase the error counter (‘true’) or only the first (‘false’).	Default = ‘true’
NotificationSubject	Text used as subject for notifications.	Default = "Specto Notification"
PageSizeChain	Number of URLs shown on one	99 (default)

	page of the chain configuration screen.	
PageSizeClient	Number of chains shown on one page within the client configuration screen.	99 (default)
PingEnabled	If 'true' 'ping' access is enabled. Note that ping services require an external library).	false
PlattformsToDeploy	Targets for deployment. Targets are separated by ';'; each specified as 'name:port:password'. (password section is optional). If 'PlattformsToDeploy' is specified then an entry 'deploy' is available as an action in the client configuration screen.	ProdSpecto.NLS.de:5555; 196.132.100.47:5554:pwd
ProxyDefault	Configuration of the proxy mechanism (see command 'wp')	
ProxyExcludedHosts	Hosts which shall not be accessed using the proxy. '*' may be used as wildcard; several entries are separated with '|'.	Example : *.NLS.de|localhost
ProxyHost	Name of the proxy host	Proxy.NLS.de
ProxyPort	Port number of HHTP access for proxy	8080
AmAlive	If 'true' send an 'am alive' signal.	false
ProxySHost	Name of the proxy host for HTTP-S connections	ProxyS.NLS.de
ProxySPort	Port number for HTTP-S access of proxy	8081
SocksExcludedHosts	See 'ProxyExcludedHosts'.	
SocksHost	Name of the proxy host for socks connections	socks.NLS.de
SocksPort	Port number for socks access of proxy	1080
R3CommandListener	If 'true' the R/3 RFC receiver process is enabled.	false
R3Interfaces	Specifies enabling of the R2Interfaces for incoming messages.	false (default)
R3Register	Command line parameters to register against an R/3 application server	-g 192.168.73.230 -a execcommand -x sapgw17
RampIndex	Default ramp index (see 'Ramp mode')	10
ReadOnly	Specifies if users other than the administrator are only allowed read access	false (default)
ReadTimeout	Maximum wait time for reading	25000

	from connections. In milliseconds.	
RemoteExec	Password for incoming commands. If the passwords do not match the command is rejected.	
Reporting.UseSVG	Enables SVG ('scalable vector graphics') for graphical reporting.	True / false (default)
Running	The name of the instance using a SPECTO database. Used internally as a lock for the database	
SMSPassword	Password for SMS service provider	
SMSUsername	Username for SMS service provider	
SNIsMaster	Specifies if this node acts as a SpectoNet master.	false (default)
SNMasterNode	Address of the SpectoNet master node	
SNSyncPeriod	Period (in seconds) for polling the message handler (e.g. mail host)	600 (default)
SNThisNetwork	SpectoNet name of this network.	SpectoDemoNet
SNThisNode	SpectoNet name of this instance.	WestCoast12
SNThisNodeId	The unique numeric id of this node	1
SoTimeout	If 'true' send an 'am alive' signal.	false
SoapInClient	See chapter 'SOAP server'	0
SoapInDebug	See chapter 'SOAP server'	false
SoapInEnabled	See chapter 'SOAP server'	false
SoapInScript	See chapter 'SOAP server'	Soap_in
TakeOverAdressee	If a failed SPECTO instanced is taken over by a standby instance, a message can be send to that address. (No message will be send if empty).	
TakeOverType	The type of message to be generated on take over (email, fax, sms).	e, f, s.
TempBaseDir	Directory in the file system to share generated graphic files between SPECTO engine and web server.	C:\tomcat4.0\ webapps\ROOT\ Specto\
ThreadNamePrefix	Prefix of the name of the chain processes.	proc
TempBaseDir	The directory for temporary files	
UDDITransport	The java library used for transport of UDDI messages.	org.uddi4j.transport.Apache SOAPTransport
UsersEnable	If 'true' activates enhanced user management.	false
Verbosity	Verbosity level of console messages (0=no messages).	2
XMLInClient	See chapter 'XML server'	0
XMLInDebug	See chapter 'XML server'	false
XMLInEnabled	See chapter 'XML server'	false
XMLInScript	See chapter 'XML server'	xml_in

Troubleshooting

Problems connecting to SPECTO

The login page does not appear

The web server is not started.
The web server (servlet) process is not configured for port 80.
The web server (servlet) could not connect to port 80 because this port is already in use or the process is not allowed to (‘Administrator’ / ‘root’ authorization required).
→ Check the web server’s startup log. Try using the local web browser with URL ‘http://localhost’.

No responses are returned from SPECTO

SPECTO is not started.
SPECTO could not connect to socket port 5555 (maybe has connected to another port because the initial port (5555 if not configured otherwise) is already in use.
→ check the startup log.
Your SPECTO instance is not defined for the default port (5555).
→ at the logon panel you have to supply the port number after the ‘specto’ service name, separated by a blank (e.g. ‘specto 5556’).

Problems during SPECTO execution

During start of SPECTO a class library cannot be found

The ‘CLASSPATH’ variable is not set correctly within the ‘init’ (‘init.cmd’, ‘init.sh’) script.

During start of SPECTO a huge number of Java JDBC exceptions occur

The database is not started, the SPECTO data is not installed on the database (maybe the data was not installed with the user account used for SPECTO), or the ODBC/JDBC driver is not installed.
SPECTO is not supplied with the correct startup parameters (DB-Type, -user etc.).

During a notification no emails are sent

The ‘MailHost’, ‘MailUser’ or ‘MailPassword’ variables are not properly set. Use the ‘al’ command to verify the configuration. You may use the mail send (‘sn <address> <subject> <content>’) command to verify proper configuration.

Incorrect/strange behavior

The cursor is not positioned correctly in SPECTO forms

The web browser used to access the SPECTO service does not support or has turned of Java script.

The display does not fit

SPECTO uses the browser's default font configuration; may be the font size has to be decreased/increased.

Web pages from SPECTO are only partial displayed, an HTTP 4xx error occurs or other 'strange' messages

Verify (message at startup) that the 'Tomcat 4.x' web server is in use, earlier versions (especially the 2.1) had problems with large HTML forms.
Also verify that the attribut 'FormMethod' is set to 'POST'.

Known problems

timeout

If computation of a command takes very long (e.g. graphical reports with multiple pages) it may occur that the web server times out before Specto delivers the result. In this case the return screen consists only of a new command line field.
Try to submit a command which requires less computational time or submit the command during a phase of less load on SPECTO.

database inconsistency

It may happen that the SPECTO persistence layer cannot write a changed configuration because of key violations. In such a case SPECTO is still functional but the last and any further changes are not reflected into permanent storage.
SPECTO indicates this by a red error line on any screen submitted to a user. You may try to undo the change which created the error, or you may reload the client in question using the ‘dg’ command.

application dump

If a program error occurs in SPECTO no answer will be returned to the web server and this will time out. In this case a dump occurs in the SPECTO console window. You are encouraged to make a screen shot of the dump and send it to ‘specto@NLS.de’.

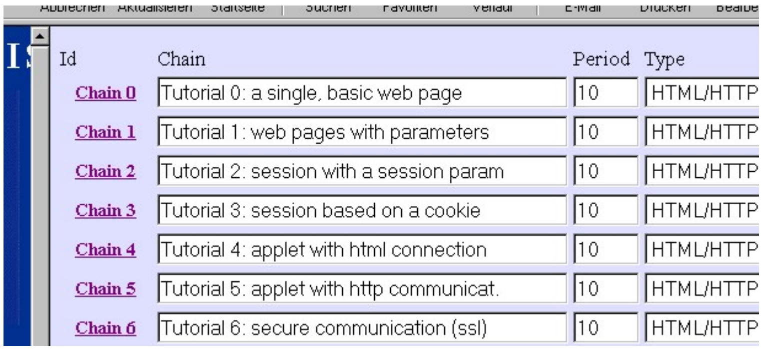
Tutorial

Overview

This tutorial consists of several sections, each of them showing a certain aspect of efficient SPECTO usage. The necessary web-pages to execute the examples are content of any SPECTO installation; also the instances on the NLS SPECTO demonstration instance ('www.NLS.de') can be used. Some examples access dynamic web-pages based on 'servlets', in this case the URL is case sensitive and must be entered exactly as shown. (you may copy the URLs from the PDF-version of this document).

The tutorials assumes that you are logged on to SPECTO and have selected the correct client (which is the case normally). To get familiar with their behavior it is recommended to play a little bit with the tutorials using a web browser before applying them to a SPECTO definition.

The screen print below shows the list of chains as they are delivered in the SPECTO demonstration client :



Id	Chain	Period	Type
Chain 0	Tutorial 0: a single, basic web page	10	HTML/HTTP
Chain 1	Tutorial 1: web pages with parameters	10	HTML/HTTP
Chain 2	Tutorial 2: session with a session param	10	HTML/HTTP
Chain 3	Tutorial 3: session based on a cookie	10	HTML/HTTP
Chain 4	Tutorial 4: applet with html connection	10	HTML/HTTP
Chain 5	Tutorial 5: applet with http communicat.	10	HTML/HTTP
Chain 6	Tutorial 6: secure communication (ssl)	10	HTML/HTTP

(You may use this client as a reference but you are encouraged to configure the tutorials by yourself).

The tutorials are enhanced continuously, please refer to the support application for the latest available tutorials.

A simple homepage monitor (tutorial 0)

In our first step we will set up a simple SPECTO system which just monitors a single web site.

Creating a new client

After logging on to SPECTO you are automatically attached to your client. You can implement the tutorials within this client or create a new client for the tutorials. (see the 'create client' section in the commands' chapter for details).

Creating a new chain

Use the 'el' command to display the current list of chains in your client. In the 'action' listbox of one of the chains select the 'add' entry and then select the 'Execute' button or press enter.
The new chain should appear in the list with a '* new chain *' title, which should be replaced with something more meaningful (like 'tutorial 0').
(If you have created a new client there is already a new chain created; which you can use).

Creating new URL, notification and off-time.

Clicking on the newly created chain link to change to the chain configuration screen into which one URL is already defined ('* new URL *'). Replace this URL with the NLS homepage ('www.NLS.de'), and tune the 'timeout' and 'too long' parameters to your needs.

Checking, testing, starting and stopping the chain.

Select the 'analyse' entry from the list box in the 'action' column; then press enter. SPECTO will read the defined web page once and displays some analytical information about the page.
Then select the 'test' entry to check the web page once; also press enter. SPECTO will test the page once and will display result information.

If the above results are okay, you may start the continuous page monitoring. To do that, go back to the chain definition page (using the 'el' command or the 'Client =…' link below the title); there select the 'start' entry of the list box and press enter. Note that the status column of the selected line should read 'running (one)'. Let the monitoring running for a while (some minutes at least) before going on to the next step. You may, but need not, stop the monitoring using the 'stop' entry.

Checking the results.

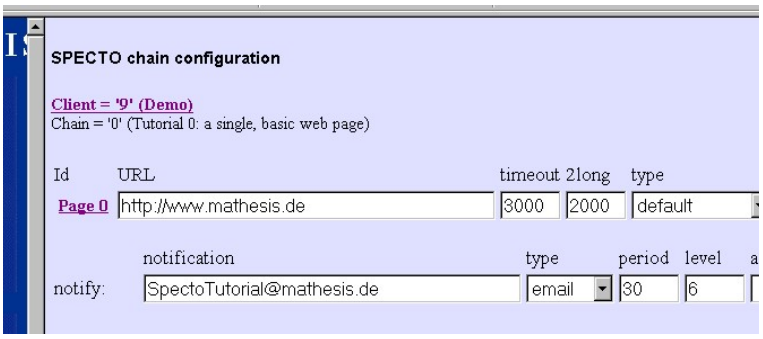
Use the 'ro 0' command to display the actual number of measurements for all chains of the current client today.

To display a graphical representation use the 'rg 0 0 1' command (assuming chain 0); for a textual view, use the 'rh 0 0 10' command. Using 'rg'/'rh' with the -? parameter gives a short explanation of the command.

Adding ‘notifications’ and ‘off times’.

Add a notification entry by clicking on the ‘add notification’ link and set the first field (‘notification’) to a valid email address (yours) and select ‘email’ in the ‘type’ field. The notification repeat period (30 minutes) and warning level (6) can also be adjusted.
Then add a ‘off time’ (a period of time in which no monitoring should happen) and adjust it to daily, from 21 pm to 3 am. (Assuming you won’t do the tutorial at that time…).

The chain configuration should now look like the screen print below :



SPECTO chain configuration

Client = '9' (Demo)
Chain = '0' (Tutorial 0: a single, basic web page)

Id	URL	timeout	2long	type
Page 0	http://www.mathesis.de	3000	2000	default

	notification	type	period	level
notify:	SpectoTutorial@mathesis.de	email	30	6

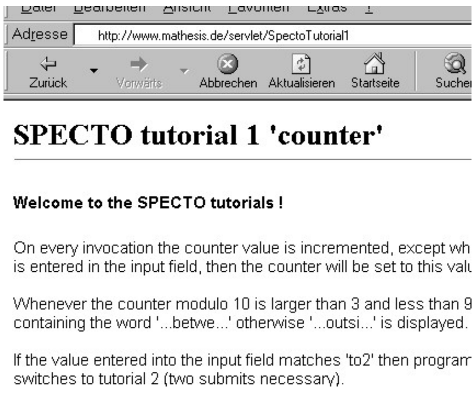
Note: Whenever the configuration of a chain is changed, the changes are NOT inherited to already running processes. To achieve this, the processes have to be stopped and restarted.

To test the notification adjust the timeout period to a low level (5 ms, to simulate a too slow response) and restart the process.
After a certain time (as soon as the error trigger level is reached) you will get an email notification. You may use the ‘no s’ command to monitor the error level rising and the ‘no l’ command to see the list of active notifications.

You may also experiment with an invalid URL instead of the short timeout value.

Monitoring a sequence of pages (tutorials 1 to 3)

In this session, monitoring will be set up for a complete sequence of web pages. You will learn about sequences of chains, the computation of page values, and the ‘session’ concept.
This tutorial is based on the preceding tutorial, you are recommended to process this before starting here. This tutorial, like the following ones are based on a set of similar web applications (‘servlets’); the screen pictured below gives an impression of their look :


Datei Bearbeiten Ansicht Favoriten Extras ?

Adresse http://www.mathesis.de/servlet/SpectoTutorial1

Zurück Vorwärts Abbrechen Aktualisieren Startseite Suchen

SPECTO tutorial 1 'counter'

Welcome to the SPECTO tutorials !

On every invocation the counter value is incremented, except wh
is entered in the input field, then the counter will be set to this valu

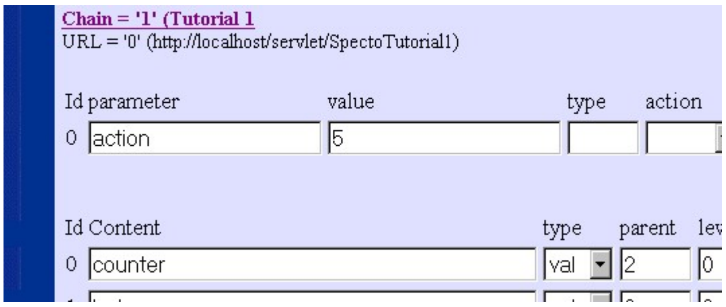
Whenever the counter modulo 10 is larger than 3 and less than 9
containing the word '...betwe...' otherwise '...outsi...' is displayed.

If the value entered into the input field matches 'to2' then prograr
switches to tutorial 2 (two submits necessary).

The source code of the tutorial applications is available on request.

Creating a new chain (checking page content)

As in the preceding example, create a new chain ('Tutorial 1') and, within this chain, create a new URLs and name it 'http://localhost/servlet/SpectoTutorial1'.
Change to the URL configuration screen (using the 'Page 0' link) and specify a parameter 'action' with a value of '5'; and three content entries; two values 'counter' and 'between' and one logical 'counter and between' and connect them by specifying the parent of the values as '2' and the level of the logical entry with '1'.
The URL configuration screen should now look like :

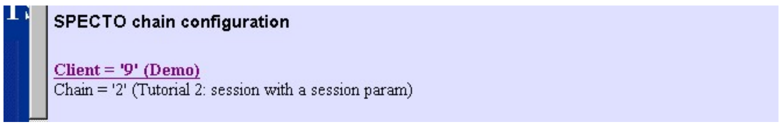

Chain = '1' (Tutorial 1
URL = '0' (http://localhost/servlet/SpectoTutorial1)

Id	parameter	value	type	action
0	action	5		

Id	Content	type	parent	lev
0	counter	val	2	0

Creating a new chain (sessions based on URL-included session parameters)

As in the preceding example, create a new chain ('Tutorial 2') and, within this chain, create a new URL and name it 'http://localhost/servlet/SpectoTutorial2'. This tutorial adds session management by usage of a session parameter in the URL. The session's parameter name is 'session', its value comes from the application, is constant during the session and will be read by SPECTO at its first appearance and then applied to every preceding page. The only difference to the preceding tutorial is that you have to define the session parameters name (here 'session') into the session field in the chain configuration screen.


SPECTO chain configuration

Client = '9' (Demo)
Chain = '2' (Tutorial 2: session with a session param)

Start the chain processing with the ‘start’ action. You may verify the session processing by applying part of the session recognition output as a content value.

Creating a new chain (session based on cookies)

SPECTO’s cookie handling is automatic, you do not have to specify anything. Note : SPECTO is serious about the session concept; every time it starts with the first URL of a chain, all cookies will be erased! You can mimic this behavior, by restarting the browser.
As in the preceding example, create a new chain (‘Tutorial 3’) and, within this chain, create two new URLs and name both of them ‘http://localhost/servlet/SpectoTutorial3’.
For both URL define the usual parameter ‘counter’ and an appropriate value, also, for the second URL specify a content variable ‘SpectoTutorialContent’.

Pages containing applets (tutorials 4 and 5)

This tutorial will show how to work with web applications using applets.
This tutorial is based on the preceding tutorials, you are recommended to process them before beginning with this tutorial.

As in the preceding example, create a new chain (‘Tutorial 4’) and, within this chain, create a new URLs and name it “http://localhost/servlet/SpectoTutorial4’.

Pages using secure communication (tutorials 6 and 7)

This tutorial will show how to work with web applications using secure communication based on HTTP-S and/or SSL.
This tutorial is based on the preceding tutorials, you are recommended to process them before beginning with this tutorial.

* to be done *

Monitoring a b2b site (tutorials 8 and 9)

This tutorial will guide you thru all the steps required to work with XML based ‘business-to-business’ (‘b2b’) implementations.
This tutorial is based on the preceding tutorials, you are recommended to process them before beginning with this tutorial.

* to be done *

Monitoring from different regions

This tutorial will guide you thru all the steps required to set up a distributed SPECTO configuration.

This tutorial is based on the preceding tutorials, you are recommended to process them before beginning with this tutorial.

SPECTO instances can be joined to a SPECTOnet. Any change made to a SPECTO configuration will be propagated to all members of the net within a short amount of time (usually some minutes).

Defining a SPECTOnet configuration

* to be done *

Propagating a configuration change

* to be done *

Joining results

* to be done *

INDEX