DECODING THE SECRETS OF

ZAGREUS

ZAGREUS

Automate your IT

Table of Contents

1. Introduction 16
2. Zagreus as a whole system 18
2.1 Zagreus Server 19
2.1.1 Communication 19
2.1.2 Logging 19
2.1.3 Components 20
2.2 Zagreus Worker 22
2.2.1 Communication 22
2.2.2 ID of the Zagreus Worker 23
2.2.3 Logging 23
2.2.4 Statuses 24
2.2.5 Memory handling 24
2.3 Zagreus Worker-Controller 26
2.3.1 Communication 26
2.3.2 Logging 26
2.3.3 Statuses 27
2.3.4 Starting Zagreus Workers 27
2.3.5 Managing Zagreus Workers 30
2.3.6 Collecting and sending worker information 30
2.3.7 Suspended mode 31
2.4 Zagreus Client 32
2.4.1 Communication 32
2.5 Zagreus Monitor 33
2.5.1 Communication 33
2.6 Other Zagreus clients 34
2.6.1 Communication 34
2.6.2 Command-line tools 34
2.6.3 Zagreus HTML Application 34

2.7 Security 35
2.7.1 SSL Certificates 35
2.7.2 Encrypted passwords, cpassword 38
2.7.3 Users and password policy 38
3. Installation 39
3.1 Downloading Zagreus 40
3.2 Installation on Windows 42
3.2.1 Copying the installation files 42
3.2.2 Setting the configuration parameters 42
3.2.3 Setting up Zagreus Windows services 44
3.2.4 Opening ports in the firewall 45
3.2.5 Starting the Zagreus services 45
3.2.6 Starting the Zagreus clients 45
3.3 Installation on Linux 47
3.3.1 Creating the target installation folder 47
3.3.2 Unpacking the archive file 48
3.3.3 Editing the set_environment.sh file 48
3.3.4 Opening ports in the firewall 49
3.3.5 Starting and administering Zagreus 49
3.4 Sending the licence key 50
3.5 Standalone installation of the client modules 51
3.5.1 Copying the installation files 51
3.5.2 Setting the *JAVA_HOME* environment variable 52
3.5.3 Starting the Zagreus clients 52
3.6 Troubleshooting 53
3.6.1 Issues independent of the operating system 53
3.6.2 Issues on Windows 53
3.6.3 Issues on Linux 54
4. Configuration 55
4.1 Zagreus Server configuration 56

4.1.1 General properties 56
4.1.2 Server startup and shutdown properties 57
4.1.3 SSL properties 57
4.1.4 MySQL properties 58
4.1.5 Queue group properties 58
4.1.6 Password policy properties 59
4.1.7 Trigger and watcher properties 60
4.1.8 Miscellaneous properties 60
4.1.9 Server-level execution options 61
4.1.10 Server-level and queue-level variables 61
4.2 Zagreus Worker-Controller configuration 63
4.2.1 General properties 63
4.2.2 Worker-related properties 64
4.3 Zagreus Worker configuration 66
4.3.1 Property lists 66
4.3.2 Property declaration for specific Worker instances 66
4.3.3 Worker startup properties 67
4.3.4 Connection properties 68
4.3.5 Miscellaneous properties 69
4.3.6 Worker-level execution options 70
4.3.7 Worker-level variables 70
4.4 File paths 72
5. System setup and administration 74
5.1 Licencing 75
5.1.1 Content of a Zagreus Licence 75
5.1.2 Installing and listing a Zagreus Licence 76
5.2 Administrative scripts 79
5.2.1 Connections 79
5.2.2 Time schedules 80
5.2.3 Scripts 80

5.2.4 Error handling 81
5.3 Concepts of user and group management 82
5.3.1 Users in the Zagreus System 82
5.3.2 Groups in the Zagreus System 83
5.3.3 Ownership of Zagreus resources 84
5.3.4 Password policy 84
6. Resources 86
6.1 Resource types 87
6.2 Resource properties 90
6.2.1 List of resource properties 90
6.2.2 Resource properties in the Zagreus Client 92
6.3 Resource versioning 93
6.3.1 Version format 93
6.3.2 Current version 93
6.3.3 Resource *ID* and *version* 94
6.3.4 Versioning in the Zagreus Client 95
6.4 Resource storaging 98
6.4.1 Embedded MySQL database 98
6.4.2 Local filesystem in the Zagreus Server 98
6.4.3 Local filesystem in the Zagreus Worker 101
7. Queuing and jobs 104
7.1 Job properties 106
7.1.1 Caller and caller type 108
7.2 Job lifecycle 110
7.3 Queue 112
7.3.1 Queue groups 112
7.3.2 Queue-level variables 113
7.3.3 Priority and priority algorithm 113
7.4 Hidden jobs 115
7.4.1 The *job_monitoring* execution option 115

7.4.2 The *invisible* result message 115
7.5 Skipped jobs 117
7.5.1 Skipped jobs in the Zagreus Monitor 117
7.5.2 Skipped jobs in the Zagreus Client 118
7.5.3 Setting the tolerance 120
7.6 Cancellation 122
7.6.1 Manual cancellation 122
7.6.2 Multiple cancellation by job statuses 123
7.6.3 Cancellation by the *zs:cancel* action 124
7.6.4 Zagreus Server startup and shutdown cancellation 124
7.6.5 Zagreus Worker automatic restart cancellation 125
8. Scripts 126
8.1 Actions 127
8.1.1 Action groups and action name 127
8.1.2 Action attributes 127
8.1.3 Action content 130
8.2 Order of execution, result flow 134
8.2.1 Ordering numbers 134
8.2.2 Execution of an action 134
8.2.3 Result of an action 135
8.2.4 *result-message* of the script 135
8.2.5 Basic traversal of the actions 136
8.2.6 Special control flow statements 137
8.2.7 Parallel threads in the *z:foreach* action 144
8.2.8 Templates 146
8.2.9 Result flow 148
8.2.10 Result attributes 151
8.3 Includes 154
8.3.1 Including connections 154
8.3.2 Including templates 155

8.3.3 Including scripts 156
8.4 Error handling 158
8.4.1 on-error-next-sibling attribute 158
8.4.2 on-error-next-child attribute 159
8.4.3 z:on-error action 159
8.4.4 *errorMessage* and *errorTrace* variables 161
8.4.5 *z:raise* action 162
8.5 Variables 163
8.5.1 z:variable action 163
8.5.2 Variable scopes 164
8.5.3 Monitoring variables 166
8.5.4 Common attributes that create new variables 166
8.6 Engine expressions 168
8.6.1 Basic expressions 168
8.6.2 Operators 169
8.6.3 Lists, records and tables 170
8.6.4 Function calls 171
8.6.5 Expressions and statements 171
8.6.6 Data types 173
8.7 Script Logging 177
8.7.1 job-log file 177
8.7.2 *z:log* action 178
8.7.3 *log* attribute 179
8.7.4 Logging levels and loglevel 179
8.7.5 z:logfile action 182
8.7.6 *logfile* attribute 182
8.7.7 log-attributes attribute 183
8.7.8 log-result-attributes attribute 183
8.8 XML representation 185
9. Connections 187

9.1 Defining connections 189
9.1.1 Creating a connection resource 190
9.1.2 General connection attributes 192
9.2 Using connections 194
9.2.1 Test connection feature 194
9.2.2 Referencing to a connection 195
9.2.3 Inserting connections to a script 196
9.2.4 Closing a connection 198
9.2.5 Opening connections in the Zagreus browser 199
9.3 Secure connections 206
9.4 *zs* connection 207
9.5 Tips and tricks 208
9.5.1 Creating standalone connection resources 208
9.5.2 Using subfolders 208
9.5.3 Using meaningful names 208
9.5.4 Using resource versioning 208
9.5.5 Keeping connections up-to-date 209
10. Zagreus Client 210
10.1 Zagreus browser window 212
10.1.1 Toolbar 213
10.1.2 Basic navigation 216
10.1.3 Common resource management operations 218
10.1.4 Opening resources 225
10.1.5 Searching for resources 226
10.1.6 Drag-and-drop operations 229
10.1.7 Script-specific operations 230
10.1.8 Connection-specific operations 237
10.1.9 Operations for event-type resources 237
10.1.10 Showing dependent resources 238
10.1.11 *Send* context submenu 239

10.1.12 Context menu of the server definition node 239
10.1.13 Context menu of a user node 241
10.1.14 Context menu of a group node 242
10.1.15 Recycle bin 243
10.2 Editor area 244
10.2.1 Script Editor 244
10.2.2 Simple text editor 246
10.2.3 Other editors 247
10.3 Extension windows of the Script Editor 248
10.3.1 Outline window 248
10.3.2 Attributes window 249
10.3.3 Variables / Functions window 249
10.3.4 Breakpoints window 251
10.3.5 Watch window 251
10.4 Monitoring 253
10.4.1 Active jobs window 253
10.4.2 Active logs window 257
10.4.3 Execution engines window 260
10.4.4 Finished jobs window 265
10.4.5 Finished logs window 272
10.4.6 Skipped jobs window 273
10.5 Main menu bar 277
10.5.1 *File* menu 277
10.5.2 *Edit* menu 277
10.5.3 *Window* menu 278
10.5.4 *Tools* menu 279
10.5.5 *Help* menu 280
10.6 Main toolbar 281
10.6.1 Views 281
10.6.2 Open/close the Zagreus browser window 283

10.6.3 Open/close the Active jobs window 284
10.6.4 Open/close the Active logs window 284
10.6.5 Open/close the Engine status window 284
10.6.6 Open a new Finished jobs window 284
10.6.7 Open a new Finished logs window 285
10.6.8 Open/close the Skipped jobs window 285
10.6.9 Save resource to the server 285
10.6.10 Save as… resource to the server 285
10.6.11 Save a new version of the resource 287
10.6.12 Save and run resource 288
10.6.13 Run script 289
10.6.14 Resume 289
10.6.15 Step to the next action 289
10.6.16 Stop debugging 289
10.6.17 Create new resource 289
10.6.18 Zoom display 291
10.6.19 Zoom in and Zoom out 292
10.7 Options dialog 293
10.7.1 Graph Editor tab 293
10.7.2 Download / upload tab 298
10.7.3 Copy tab 300
10.7.4 *General behaviour* tab 301
10.7.5 Palette tab 301
10.8 Keybindings 303
11. Zagreus Monitor 304
11.1 Main menu bar 306
11.2 Sections of the Zagreus Monitor 308
11.2.1 Zagreus Server Connections 308
11.2.2 Timeline area 310
11.2.3 Filter area 319

11.2.4 Execution Engines window 323
12. Other Zagreus clients 325
12.1 Command-line tools 326
12.1.1 Executable files 326
12.1.2 Examples for Windows 327
12.1.3 Examples for Linux 329
12.2 Zagreus HTML application 331
12.2.1 *Run script and get info* tab 333
12.2.2 *Job info* tab 334
12.2.3 *Fire event* tab 335
12.3 Troubleshooting 337
13. Script Editor 338
13.1 Layout 339
13.1.1 Canvas 340
13.1.2 Palette 340
13.1.3 View selector tabs 341
13.2 Actions 343
13.2.1 View modes 343
13.2.2 Basic operations 347
13.2.3 Editing 352
13.3 Action help 359
13.4 Formatting 361
13.4.1 Alignment operations 361
13.4.2 Size operations 362
13.5 Additional displaying options 365
13.5.1 Outside displaying option for a child action 365
13.5.2 Showing sibling links 366
13.5.3 Opacity 366
13.5.4 Displaying goto expressions 367
13.5.5 Attribute as child element 370

13.6 Special operations 373
13.6.1 Find in script 373
13.6.2 Set breakpoint 374
13.6.3 Encrypt password 375
13.6.4 Paste path 376
13.6.5 Show path in status line 376
13.7 Configuration options 377
14. Debugging in the Zagreus Client 378
14.1 Features 379
14.2 Debugging concepts and terms 380
14.3 Starting a debug session 382
14.4 Debug Editor 384
14.4.1 Main toolbar 384
14.4.2 Debug Editor and the execution workflow 385
14.4.3 Action context menu 386
14.4.4 Breakpoints window 387
14.4.5 Watch window 388
14.5 Best practices 390
15. Initiating script execution 391
15.1 Overview 392
15.1.1 Manual execution 392
15.1.2 Execution by event-type resources 393
15.2 Execution options 395
15.2.1 Declaration levels 395
15.2.2 Precedence order for resolution 396
15.2.3 List of execution options 397
15.2.4 Prefixes 400
15.3 Start-up variables 401
15.3.1 Declaration levels 402
15.3.2 Precedence order for resolution 403

15.3.3 Prefixes 404
15.3.4 Automatically set start-up variables 405
15.3.5 List of resolved start-up variables 408
15.4 Subscriptions 410
15.4.1 Subscriptions from the perspective of scripts 410
15.4.2 Subscriptions from the perspective of event-type resources 415
15.5 Execution by event-type resources 417
15.5.1 Event schedule 417
15.5.2 Time schedule 419
15.5.3 Mail watcher 420
15.5.4 Database watcher 426
15.5.5 File trigger 433
15.5.6 Special events 435
15.5.7 Administrative tools for event-type resources 437
15.6 Manual script execution 443
15.6.1 Execution in the Zagreus Client 443
15.6.2 Execution with the *.sendscripts* file 444
15.6.3 Execution from the command-line client 446
15.6.4 Execution from the Zagreus HTML Application 446
15.6.5 Execution from external systems 447
15.7 Execution from a Zagreus script 451
15.8 Summary 452
15.9 Best practices 453
15.9.1 Choosing the appropriate event-type resource 453
15.9.2 Monitoring watchers and triggers 454
15.9.3 Quarterly settings for a time schedule 454
15.9.4 Using special subscription features 455
15.10 Troubleshooting 457
15.10.1 Practices for event-type resources 457
15.10.2 Command-line tools 458

15.10.3 The HTML application 458
16. Special features 459
16.1 Standalone Worker 460
16.1.1 How to use 460
16.1.2 Configuration 461
16.1.3 Licencing 461
16.1.4 Limitations 462
16.1.5 Notes 462
16.2 External script execution 464
16.2.1 How it works 464
16.2.2 Configuration 465
16.2.3 *tempfilename* attribute 465
16.2.4 *params* attribute 466
16.2.5 Synchronous and asynchronous execution 467
16.2.6 Examples 467
16.3 Document URL feature 472
16.3.1 How to use 472
16.3.2 *docurl* variable 473
16.3.3 *docurl_replace* variable 474
16.3.4 Examples 475
16.4 Bank holidays feature 479
16.4.1 Common European holidays 479
16.4.2 Specifying additional bank holidays 480
17. Server administration in the Zagreus Client 481
17.1 Administrator options 482
17.1.1 Group management 482
17.1.2 User management 485
17.1.3 Cancel all jobs 491
17.1.4 Stop / start server components 491
17.1.5 Manage certificates 491

17.1.6 Monitor watchers, triggers 493
17.1.7 Configuration testing 497
17.2 Get licence information 499
17.3 Server information 500

1. Introduction

IT departments and their managers face a challenging issue in the rapidly evolving field of information technology: satisfying rising expectations with limiting budgets. Organizations seek for success in a competitive market by shifting multiple processes to IT, with the goal of improving speed, precision, and cost-effectiveness. While investments in technological infrastructure continue to be approved, the need for more human resources sometimes falls on deaf ears. This shortage necessitates more ingenuity, which leads to what is frequently viewed as an outstanding solution: automation.

Automation, unquestionably recognized for its ability to save time and money, becomes a significant concern. However, the pressing question remains: how could automation be implemented successfully in the face of time and money constraints? The urgency of the issue frequently encourages the selection of the simplest and quickest choices, including new features via software products, open-source code, in-house development, or customized systems with restricted capability. Unfortunately, the quickest solution is not always the best. Although short-term savings may be realized, negative long-term consequences are more likely to arise.

To avoid falling into the automation trap — attaining short-term goals at the price of long-term issues — a fundamental change toward centralization is required. Sustainable automation is based on centralizing all processes within a dedicated system, which provides a variety of benefits such as process simplification, easy definition of interdependencies, cost-effective development and maintenance, integrated error handling with logging functionality, comprehensive monitoring at a central point, and unparalleled flexibility through scalability.

Enter Zagreus, a cutting-edge process platform precisely created for businesses of all sizes and types. Zagreus, with its extensive library of capability, enables the quick mapping of all digitally controllable business processes. This platform is distinguished by many significant advantages:

- *Low-code platform*
  Zagreus avoids the need for heavy programming by utilizing a low-code platform that allows users to develop and alter processes without considerable coding skills.

- *Utilization of existing system components*
  Zagreus readily interfaces with diverse data storage and operating systems, improving compatibility and lowering implementation hurdles.

- *Centralized administration*
  By streamlining administrative activities, Zagreus enables businesses to manage and monitor their automated operations from a single location, improving control and efficiency.

Zagreus stands out as a standard software solution with great flexibility, recognizing the individuality of each enterprise. Zagreus is a plug-and-play solution with rapid installation and quick utilization, avoiding the need for lengthy, costly projects. Old procedures are quickly changed, and new ones are flawlessly incorporated, showcasing Zagreus's expertise in this area.

Zagreus's flexible REST API interface expands its possibilities even further, allowing for smooth connectivity with a variety of systems. This versatility distinguishes Zagreus as a dynamic and future-ready solution that meets the changing demands of modern organizations.

In essence, Zagreus is more than a software solution: it is a strategic enabler that allows organizations to navigate automation issues with exceptional efficiency and agility. As more is learned about Zagreus, its disruptive impact on business process automation becomes clear, bringing in a new era of IT solutions.

2. Zagreus as a whole system

Zagreus is a modular system, consisting of several main parts that are constantly communicating with each other. This design has the following advantages:

- different modules are responsible for their own processes only,
- the modules can be installed on different hosts,
- the whole system contains indirectly interfering asynchronous tasks. The proper implementation for this needs separate main execution processes (Java Virtual Machines). Therefore, all Zagreus modules are implemented as Java applications running on separate JVMs.

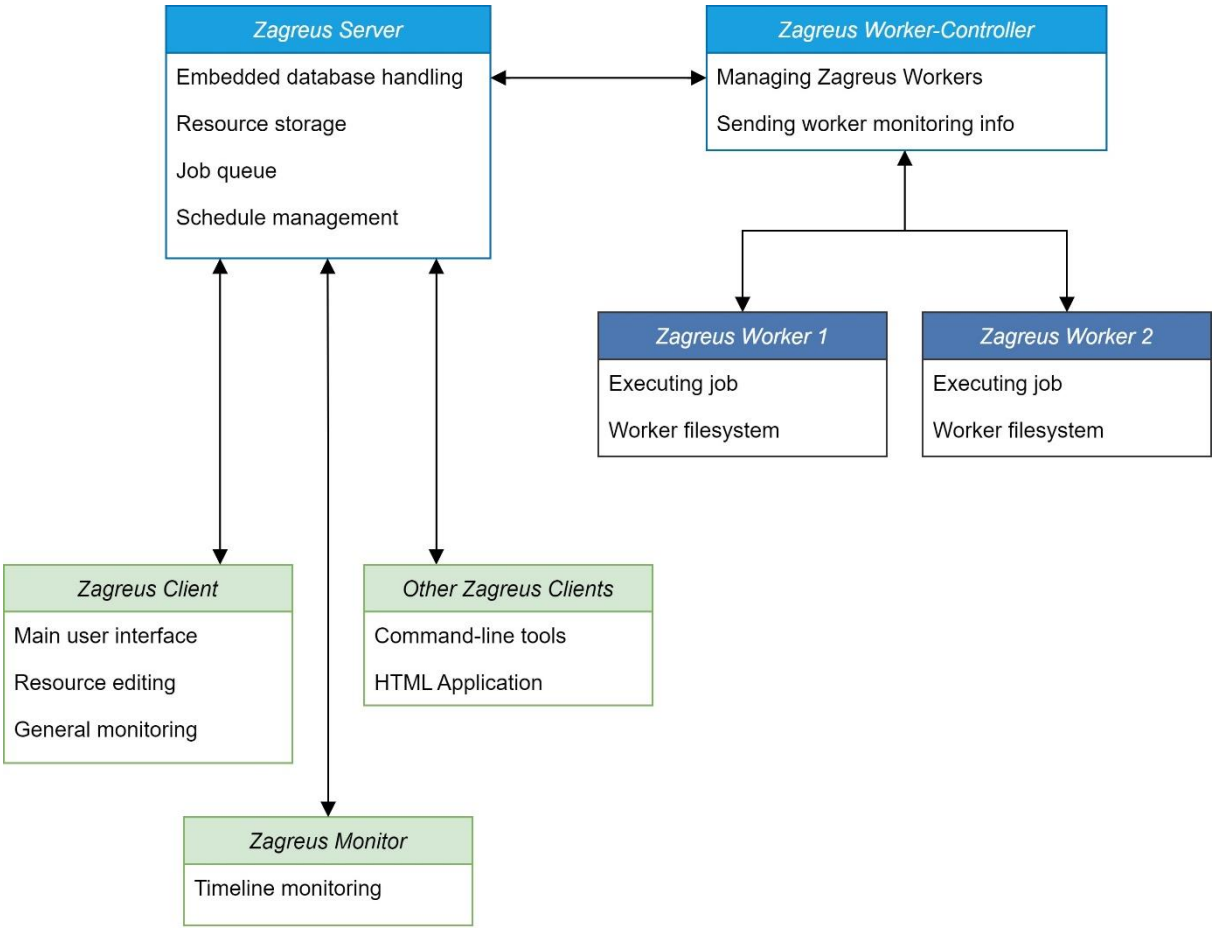


Figure 1 – The main structure of the Zagreus modules

In *Figure 1.*, the main modules are shown. In the following chapters, these modules are introduced and described in details.

2.1 Zagreus Server

The Zagreus Server is the main module of the Zagreus System. The Server module installation is neccessary in order to have a minimal working system, it maintains an embedded database (see → Embedded MySQL database) for its resources and job information (see → Job properties).

The Zagreus Server is performing the following tasks:

- embedded database management
- web server management
- user authentication and authorization
- resource management in database and in local filesystem
- job queue management
- maintaining connection for Zagreus Workers
- storing job-log files, results
- running time schedule events and subscriptions
- managing watcher and trigger type resource events

2.1.1 Communication

The Zagreus Server uses two different communication protocols:

- The Zagreus Client modules communicate with the Zagreus Server via HTTP / HTTPS webservice protocol. Therefore the clients and the Server can be installed anywhere throughout the Internet.
- The Zagreus Workers and Zagreus Worker Controller communicate with the server by Java RMI protocol, so these modules need to be installed on the same intranet, or preferably on the same machine.

The Zagreus Server is the central module to which all other modules connect. Therefore, it is recommended to start the Zagreus Server first , right before all other modules.

2.1.2 Logging

The Zagreus Server module generates log-files as it is running. They contain events and possible error messages as well as information about job execution and user

interaction. This is crucial for monitoring system performance, diagnosing issues, and ensuring security.

There are two different types of log-files:

- *server log-file*
  The server log-file contains information about the server module: starting up and shutting down, events in the job-queue, user logins and logouts. It is located in the `<zagreus_home>/server/log` folder, under the file name `srv`.
  The log-file is maintained by the `log4j` system, and it is archived regularly by rolling file appender feature: when the size of the log-file exceeds 10 megabytes, or a full day has passed, a new `.gzip` archive is automatically generated from the actual log-file. The content of the log-file will be stored in the archive, and a log-file itself will be reset to empty. The name of the archive `.gzip` file uses the format of `srv_<yyyyMMdd>.log.gz`, where the timestamp in the filename follows the creation date of the archive. Archived log-files older than 30 days are deleted automatically.

- *job-log files*
  For each job executed in the Zagreus System, a separate *job-log* file is generated. The file name is the ID of the job, and the file is located under `<zagreus_home>/server/log/job` unless it is set otherwise, see → General properties . For more information about job-logs, see → job-log file.

2.1.3 Components

There are several larger parts of the Zagreus Server that can be categorized as system components.

2.1.3.1 Job queue

The job queue is the main component in the Zagreus Server. It stores and manages the tasks (the so-called *jobs*) to be executed by the Zagreus Workers, see → Zagreus Worker. For further details, see → Queue.

2.1.3.2 Quartz scheduler

Quartz Scheduler is the component which manages the execution of recurring, time-based tasks. The Scheduler component registers *crontime*-based definitions (see →

Time schedule) and triggers events at the specified time points. These events can execute a Zagreus script (see → Execution by event-type resources) or check out a mail- or database watcher condition, see → Mail watcher and → Database watcher.

Zagreus System uses the Quartz Scheduler, which is a robust, open-source job scheduling library for Java applications, enabling flexible scheduling of jobs with simple, interval-based, or cron-like expressions.

The scheduler can be temporaliry switched off from the Zagreus Client (see → Stop / start server components) and from the Zagreus Monitor (see → Additional options) applications.

2.1.3.3 Embedded MySQL database

The Zagreus Server stores its resources and metadata information in an embedded MySQL database, which is starting and shutting down together with the Zagreus Server. For more details, see → Embedded MySQL database.

2.1.3.4 Local filesystem

Another important component of the Zagreus Server is the connection with the local filesystem, that allows managing files and folders on the local installation machine. For more details, see → Local filesystem in the Zagreus Server.

2.2 Zagreus Worker

The Zagreus Worker is responsible for executing a Zagreus script type resource, see → Scripts. Multiple Zagreus Worker instances can run at the same time, at least one is needed to allow script execution. Each Worker instance runs as a separate JVM (Java Virtual Machine), so they are independent processes on the OS level. The number of Zagreus Worker instances depends on the Zagreus licence (see → Licencing) and the configuration (see → Configuration).

The Zagreus Worker module is designed to be a constantly running process which is connecting to the Zagreus Server job queue (see → Queue) and waiting for a new job to execute. When there is no such job, it stays in idle mode and keeps waiting for a task.

The Zagreus Worker contains an execution processor called the Zagreus Execution Engine which provides the main functionality for executing Zagreus scripts.

The Zagreus Worker is performing the following tasks:

- In idle mode, it is constantly connecting to the Zagreus Server, waiting for a new job to execute.
- If there is a job to execute, it passes the script content of the job to the Zagreus Execution Engine.
- The Zagreus Execution Engine processes the script content.
- After the execution has been finished or failed, the status and the result of the job are sent to the Zagreus Server, and the Zagreus Worker switches back to idle mode, waiting for the next job to execute.

2.2.1 Communication

The Zagreus Worker communicates both with the Server and the Zagreus Worker-Contoller, and uses the Java RMI protocol:

- *The Zagreus Worker as a client connecting to the Zagreus Server*
  The Worker is constantly connected to the Zagreus Server job queue (see → Queue) via the Java RMI protocol. This RMI port is defined in server configuration (see → Zagreus Server configuration), but the same port has to be set in the Zagreus Worker configuration (see → Zagreus Worker configuration) in order to Worker could connect to the Server properly.

- *The Zagreus Worker behaves as a server when the Zagreus Worker-Controller is connecting to a Worker instance*
  The Worker-Controller also maintains a communication to the Zagreus Worker instances to be able to monitor and manage them. The Worker-Controller is using Java RMI communication. The RMI port is different for each worker instance and it is derived from the specified server RMI port and the Worker ID; so if the pre-defined Zagreus Server RMI port is 6666, then the RMI port is 6667 for Worker (ID: 1), 6668 for Worker (ID: 2), respectively.

Each job is assigned to the Zagreus Worker instances with round robin assignment, except when a specific Zagreus Worker group is selected (see → Queue groups).

2.2.2 ID of the Zagreus Worker

Each Zagreus Worker has an ID which is an numeric integer value starting from 1. When the Zagreus Worker-Controller starts the Zagreus Workers (see → Starting Zagreus Workers), the Worker IDs are consecutive numbers, such as *1, 2, … N* where *N* is the the maximum number of the allowed Workers defined in the Zagreus Licence, see → Licencing.

Zagreus Workers can be stopped, restarted by the user → Worker information tab, so the Worker IDs might not be consecutive any more. However, when the user manually starts a new Zagreus Worker, its ID needs to be unique and in the range of *[1, N]* where *N* is the maximum number of allowed Workers.

In some cases. a *fully qualified Worker ID* is neccessary which is specified in the form of:

```
<Worker-Controller ID>.<Worker ID>
```

This format can unambigously identify the given Workers even if they belong to different Worker-Controllers.

2.2.3 Logging

Each Zagreus Worker module instance generates a log-file as it is running. It contains events and possible error messages as well as information about job execution. This is crucial for monitoring system performance, diagnosing issues, and ensuring security.

The module log-file contains information about the actual Worker instance: starting up and shutting down, initiating and finishing job execution, as well as stacktraces of module-related errors. It is located in the `<zagreus_home>/worker-`

`controller/worker/log` folder, under the file name `worker_<worker-id>_<yyyyMMdd>.log`, where the timestamp in the filename follows the creation date of the log-file.

2.2.4 Statuses

The Zagreus Worker undergoes a life-cycle with distinct statuses. These statuses are the following:

- *Initializing*
  The Zagreus Worker is starting up. It lasts only for a few seconds.

- *Idle*
  The Zagreus Worker has successfully started and is waiting for a job to execute.

- *Busy*
  The Zagreus Worker is currently executing a job.

- *Shutting down*
  The Zagreus Worker is shutting down.

For most of the time, the Zagreus Worker switches between *Idle* and *Busy* statuses.

The status of the Zagreus Workers can be seen in the *Execution engines* window in the Zagreus Client application, see → Zagreus Client.

Aside from these statuses, there is another property which defines the availability of the particular Zagreus Worker. The *Enabled* property tells if the Worker can receive any job. By default, the *Enabled* property is true, but the user can manually set it to false in the Zagreus Client (see → Worker information tab). In the latter case the Zagreus Worker is in *Idle* status, but will not accept any job for execution.

2.2.5 Memory handling

The Zagreus Worker has a few settings in terms of memory handling. Each Worker instance manages its memory independently.

Because of the fact that a Zagreus Worker is a JVM, the memory settings are specified as standard Java options in the Zagreus Worker configuration, see → Worker startup properties.

There might be a need for restarting a Zagreus Worker if it has exceeded a pre-defined memory consumption after a job execution. For such cases, a special configuration property can be set, see → Miscellaneous properties.

The actual memory consumption of each Zagreus Worker can be monitored in the *Execution Engines* window in the Zagreus Client (see → Execution engines window) and in the Zagreus Monitor applications (see → Execution Engines window).

2.3 Zagreus Worker-Controller

The Zagreus Worker-Controller module is responsible for managing the instances of Zagreus Worker modules. Although the execution of a Zagreus script is performed by a Zagreus Worker module, the Zagreus Worker-Controller module installation is neccessary to start and manage the Zagreus Worker instances themselves.

The Zagreus Worker-Controller is performing the following main tasks:

- starting and managing Zagreus Workers
- canceling a running job on a Zagreus Worker
- collecting real-time information about Zagreus Workers and sending them to the Zagreus Server module

Besides these, it also handles minor tasks like managing SSL certificates (see → Manage certificates) and testing the integrity of both the configuration files of itself and of the Zagreus Worker (see → Configuration testing).

> ***Info:*** *When the Zagreus Worker-Controller module starts up, it starts the number of worker instances set in the configuration. When the Worker-Controller is shut down, it first stops all worker instances.*

2.3.1 Communication

The Zagreus Worker-Controller communicates both with the Server and the Zagreus Workers via the Java RMI protocol. The RMI port of the Zagreus Server can be defined in the configuration file, see → General properties.

2.3.2 Logging

The Zagreus Worker-Controller module generates a log-file as it is running. It contains events and possible error messages as well as information about job execution and user interaction. This is crucial for monitoring system performance, diagnosing issues, and ensuring security.

The module log-file contains information about the Worker-Controller module: starting up and shutting down, commands from the Zagreus Server and the OS

commands which start the Zagreus Worker processes. It is located in the `<zagreus_home>/worker-controller/log` folder, under the file name `wc`.

The log-file is maintained by the `log4j` system, and it is archived regularly by rolling file appender feature: when the size of the log-file exceeds 10 megabytes, or a full day has passed, a new `.gzip` archive is automatically generated from the actual log-file. The content of the log-file will be stored in the archive, and a log-file itself will be reset to empty. The name of the archive `.gzip` file uses the format of `wc_<yyyyMMdd>.log.gz`, where the timestamp in the filename follows the creation date of the archive. Archived log-files older than 30 days are deleted automatically.

2.3.3 Statuses

The Zagreus Worker-Controller undergoes a life-cycle with distinct statuses. These statuses are the following:

- *Starting*
  The Zagreus Worker-Controller is starting up. It lasts only for a few seconds.

- *Running*
  The Zagreus Worker-Controller has successfully started and is running.

- *Suspended*
  The Zagreus Worker-Controller can not establish an active connection to the Zagreus Server, see → Suspended mode.

- *Shutting down*
  The Zagreus Worker-Controller is shutting down.

The status of the Zagreus Worker-Controller can be seen in the *Execution engines* window in the Zagreus Client application, see → Execution engines window.

2.3.4 Starting Zagreus Workers

A Zagreus Worker is started by the Zagreus Worker-Controller module via a command-line command as a separate Java child process. The structure of this command is:

```
<Java executable> -cp <classpath> <Java options> <Worker class name>
<Worker ID> <Worker-Controller ID> <Server address> [<parameters>]
```

All of this is performed automatically by the Worker-Controller. However, it is useful for the end-user to know about this command structure in case there is a need to configure or add parameters for the starting Zagreus Workers, which can be done by editing the appropriate configuration properties (see → Worker startup properties).

The elements of this command are the following:

- *<Java executable>*
  The OS path of the Java executable, shipped with Zagreus.

- *<classpath>*
  The Java classpath of the Zagreus Worker.

- *<Java options>*
  The various Java options passed to the Zagreus Worker. Besides the user-configured values (see → Worker startup properties), further parameters are added:
  ```
  -Dworking.folder
  -Djava.folder
  -Dworker.id
  -Dworker.date
  -Dworker.timestamp
  ```

- *<Worker class name>*
  The fully qualified name of the main Java class of the Zagreus Worker.

- *<Worker ID>*
  The ID of the actual Zagreus Worker.

- *<Worker-Controller ID>*
  The ID of the Zagreus Worker-Controller.

- *<Server address>*
  The address of the Zagreus Server.

Info: *The actual command-line command used to start the Zagreus Worker instances can be seen in the log file of the Worker-Controller module.*

2.3.4.1 Number of Zagreus Workers

The Zagreus Worker-Controller is a standalone module, thus it is starting up in itself, and it tries to connect to the Zagreus Server as soon as possible. The number of possibly running Zagreus Worker instances is limited by the Zagreus license (see → Licencing), which is installed in the Zagreus Server module database. Before reaching the Zagreus Server, the Worker-Controller module is not aware of the maximum number of Zagreus Workers allowed. However, the Zagreus Worker-Controller can start the Zagreus Worker instances before connecting to the Server module. The number of started instances can be set in the Worker-Controller configuration (see → Zagreus Worker-Controller configuration) with the *workercontroller.defaultworkercount* property.

When the connection between the Zagreus Worker-Controller and the Zagreus Server is finally established, the limitation of the Zagreus Licence is automatically applied if the number of started Zagreus Worker instances is greater than the allowed number (it can only happen by misconfiguration), so some of the Workers are stopped by the Worker-Controller.

Setting the default number of starting Zagreus Worker instances, however, can be practical in the following cases:

- The user can manually configure the proper load-balance settings when multiple Worker-Controller instances are installed. In this case, the *total count* of Zagreus Worker instances managed by each Worker-Controllers cannot exceed the limit set in the Zagreus licence.
- When there is one Worker-Controller instance in the Zagreus ecosystem, this setting can manually be matched to the allowed number of Zagreus Worker instances defined in the Zagreus licence. In this case, all Worker instances are already running when the Worker-Controller establishes the connection to the Zagreus Server.

2.3.5 Managing Zagreus Workers

The Worker-Controller module is responsible for managing the worker instances by receiving commands from the Zagreus Server.

These command can be one of the following:

- *Start:* starts a new worker instance.
- *Stop:* stops an already running worker instance.
- *Restart a worker instance:* stops the worker instance, and then starts it again.
- *Disable a worker instance:* the worker instance is kept running, but does not accept any job from the Zagreus Server
- *Enable a worker instance:* enables the disabled Zagreus Worker
- *Cancel a job:* cancels a job running on a worker instance.

The Worker-Controller is also checking the availability of the worker instances:

- When a worker instance is not available (the connection is lost between the Worker-Controller and the worker instance), the Worker-Controller restarts the particular worker instance.
- When any worker instance has been exceeding a configurable maximum memory consumption limit for its lastly executed job, the Worker-Controller restarts that Worker in order to avoid any memory issues for its future jobs.

2.3.6 Collecting and sending worker information

The Worker-Controller module collects the actual status and further information about the worker instances (i.e. *worker information*), and forwards them to the Zagreus Server. The following information are collected:

- the ID of the worker instance
- whether the worker instance is enabled
- the status of the worker instance, see → Statuses
- the ID of the currently executed job (if any)
- the status of the currently executed job (if any), see → Job lifecycle
- the ID of the currently executed script (if any)
- the full path of the currently executed script (if any)
- the number of log lines produced by the currently executed job (if any)

- the time when the worker instance was started
- the actual time when the worker information was sent
- the maximum allowed memory of the worker instance
- the total memory of the worker instance
- the free memory of the worker instance
- the number of available processors

Furthermore, the Worker-Controller sends information about itself to the Server as well, consisting of:

- the ID of the Worker-Controller
- the status of the Worker-Controller
- the time when the Worker-Controller was started
- the actual time when the Worker-Controller information was sent
- the maximum allowed memory of the Worker-Controller
- the total memory of the Worker-Controller
- the free memory of the Worker-Controller

This information can be viewed in the Execution Engines view of the Zagreus Client, see → Execution engines window.

The frequency of collecting and forwarding this information can be defined in the configuration file, see → Zagreus Worker-Controller configuration.

2.3.7 Suspended mode

When the Worker-Controller is unable to send the worker information to the Zagreus Server multiple times in a row, it switches into *suspended mode*. When in suspended mode, the Worker-Controller module suspends all forms of communication with the Server: it does not send the worker and worker-controller information, does not accept (and forward) commands related to the Workers, and does not accept commands controlling its own behavior (e.g. shut down). The suspended Worker-Controller is trying to establish the lost connection regularly, and when it can be repaired, all suspended services are revoked.

The properties for timeouts and polling frequencies in suspended mode are configurable, see → Worker-related properties.

2.4 Zagreus Client

The Zagreus Client is the main user interface in the Zagreus System. The Zagreus Client is implemented only for Windows operating systems.

The end-user or administrator can use the following functionality with the Zagreus Client:

- creating and maintaining Zagreus Server connections
- browsing the embedded database and local OS filesystems (see → Resource storaging) and performing file operations (e.g. copy, move, create folder)
- viewing and editing Zagreus resources
- managing and browsing external connections (see → Opening connections in the Zagreus browser)
- creating, editing and debugging Zagreus scripts
- monitoring the active and finished jobs
- monitoring the Zagreus Worker-Controller and Zagreus Worker instances
- doing administrative tasks like user and group management (see → User management)

2.4.1 Communication

The Zagreus Client communicates with the Zagreus Server by using the HTTP / HTTPS webservice protocol.

2.5 Zagreus Monitor

The Zagreus Monitor is a standalone client application for monitoring script execution including finished and active jobs as well as scheduled estimations. The Zagreus Monitor is the secondary user interface in the Zagreus System and is implemented for Windows operating systems only.

The end-user or administrator can check the following in the Zagreus Monitor:

- the job executions on the timeline with user-defined filters
- the future estimations of subscribed scripts on the timeline, see → Timeline area
- the skipped jobs on the timeline, see → Skipped jobs in the Zagreus Monitor
- the properties of jobs including start-up variables, see → Start-up variables
- the monitoring variables, see → Monitoring variables
- the job-logs of any finished job, see → job-log file

2.5.1 Communication

The Zagreus Monitor communicates with the Zagreus Server by using the HTTP / HTTPS webservice protocol.

2.6 Other Zagreus clients

Apart from the Zagreus Client and Zagreus Monitor modules, there are a few other client applications in Zagreus. Due to the fact that there are several common functionalities of the Zagreus Server which are used quite often (initiating a script execution, firing an event, etc.), these functionalities can be accessed via multiple client applications.

2.6.1 Communication

All other client applications communicate with the Zagreus Server by using the HTTP / HTTPS webservice protocol.

2.6.2 Command-line tools

The Zagreus command-line tools consist of a collection of simple executable files. To support both Windows and Unix platforms, there is a `.bat` and an `.sh` script file available for every functionality supported. There are batch files and shell scripts for administrative tasks as well as for running scripts and firing events. For further details, see → Command-line tools.

2.6.3 Zagreus HTML Application

The Zagreus HTML Application is a web-based user interface for initiating script execution, triggering event schedules and get information about jobs. It is hosted by the Zagreus Server. For further details, see → Zagreus HTML application.

2.7 Security

Securing online communications is essential to protect sensitive information from cyber threats. This chapter focuses on the critical role of SSL certificates in ensuring secure connections over the internet. SSL certificates encrypt data transmitted between users and websites, establishing a trusted and secure environment for online transactions.

Zagreus uses several connection types between its modules as well as external connections from the system. Besides that, there is always a user who is using the connection. There are the following points where any security risk can emerge in terms of connections and users for the Zagreus System:

- *HTTP / HTTPS connections among modules*
  The connection between the Server and the clients are HTTP / HTTPS, see → Communication, which needs to be secured, specially if the client modules are installed on different hosts over the internet.
  The goal here is to protect the transmitted data by using a trusted SSL protocol.

- *HTTP / HTTPS external connections*
  In the Zagreus System, the user has the possibility to use external connections, such as IMAP, POP3, FTP, etc., see → Connections.
  In most of the cases, these external connections require an installed certificate on the client (Zagreus) side in order to open a secure connection, see → Secure connections.

- *Security on the user level, authentication and authorization*
  On a fine-tuned system, there are multiple users, each of them performing specific tasks. Users need to authenticate by their user name and password. The administrator user needs to authorize common resources (i.e. groups) for them (see → Groups in the Zagreus System) as well as applying password policy.
  The goal is to fine-tune the system properly and maintain user accessibility.

2.7.1 SSL Certificates

SSL is a protocol used to encrypt data transmitted over the internet, providing security and privacy. When SSL is combined with HTTP, it forms HTTPS (HyperText Transfer Protocol Secure). HTTPS is the secure version of HTTP, ensuring that all data

exchanged between a user's client application and a web server is encrypted and secure from eavesdropping or tampering.

Each module has a specific file location for storing the SSL certificates. These files are called *truststores*.

- *for Zagreus Server:*
  `<zagreus_home>\server\service\conf\ssl\client.trustedservers`

  In this truststore, both the Zagreus Server self-signed certificate (public and private key) and the certificates for external connections (see → Certificates for external connections) are installed.
  The truststore filename can be changed in the Zagreus Server configuration, see → SSL properties

- *for Zagreus Workers:*
  `<zagreus_home>\worker-controller\worker\conf\ssl\client.trustedservers`

  In this truststore, both the Zagreus Server self-signed certificate (public key) and the certificates for external connections (see → Certificates for external connections) are installed.
  The truststore filename can be changed in the Zagreus Worker configuration, see → Miscellaneous properties

- *for Zagreus Client:*
  `<zagreus_home>\gui\configuration\client.trustedservers`

  In this truststore, the Zagreus Server self-signed certificate (public key) is installed.

- *for Zagreus Monitor:*
  `<zagreus_home>\monitor\conf\client.trustedservers`

  In this truststore, the Zagreus Server self-signed certificate (public key) is installed.

- *for Zagreus command-line tools:*
  `<zagreus_home>\command-line\keystore\client.trustedservers`

In this truststore, the Zagreus Server self-signed certificate (public key) is installed.

- *for Zagreus HTML application:*
  By using the recommended HTTPS URL for the Zagreus HTML application (see → Zagreus HTML application), the connection is already trusted.

2.7.1.1 Certificates for communication between modules

There is a pre-configured SSL certification installed for the connection between the Zagreus Server and the client applications. This is a *self-signed* certificate, meaning that it is signed by the same entity whose identity it certifies, therefore the issuer and the subject of the certificate are the same. Self-signed certificates, in general, can safely be employed for internal use.

If the Zagreus System is installed in a way that the client applications are not on the same host as of the Zagreus Server is installed on, there is the possibility to install an official CA-signed certificate. In order to do this, the system administrator needs to perform the following steps:

- obtain the official CA certificate (public and private keys)
- connect to the Zagreus Server with the Zagreus Client, and open the *Manage certificates* dialog box, see → Manage certificates. This process can be done with the `keytool` command-line application as well
- delete the pre-installed self-signed Zagreus Server certificate (with the alias *zagreus_server*)
- install the official public and private key to Zagreus Server truststore file with a selected alias
- using the `keytool` command-line application, the user needs to install the official CA certificate (public key) into the truststore file of each client module

2.7.1.2 Certificates for external connections

Only the Zagreus Server and the Zagreus Worker modules use external connections. Therefore, the certificates for external connections need to be installed for these two modules. The most straightforward way to do this is via the *Manage certificates* dialog box, see → Manage certificates. This process can be done with the `keytool` command-line application as well; the locations of the truststore files were already listed above in this chapter.

2.7.2 Encrypted passwords, cpassword

Just like in any system, passwords are considered as sensitive data in the Zagreus System. User passwords are stored in the following locations:

- in the embedded MySQL database (encrypted in AES-256)
- in the connections.dat files for the Zagreus Client and Zagreus Monitor applications (encrypted as *cpasswords*)
- in the Zagreus script connections (can be encrypted as *cpasswords*)

The user needs to take care about the last occurences only. There is a way to hide the password values declarared in a Zagreus script by encrypting them as *cpassword*, see → username, password and cpassword attributes.

2.7.3 Users and password policy

It is the responsibility of the administrator user to create and manage the non-administrator users (see → Users in the Zagreus System) for the system. The password policy (see → Password policy) is an important aspect of the security. Besides their own home folder, each user has authorization for the groups they are the member of, see → Groups in the Zagreus System.

3. Installation

In this chapter the installation and configuration of specific Zagreus modules, the Zagreus Server, Zagreus Client and Zagreus Monitor are described.

The Zagreus Server and the Zagreus Worker-Controller module can be run both on Windows and Linux platforms, while the Zagreus Client and the Zagreus Monitor modules are available only for the Windows operation system.

3.1 Downloading Zagreus

The installation packages and instructions can be downloaded from the following web address: https://support.etixpert.com/zagreus/download-1.5.php , see *Figure 1*. To obtain a username / password pair for accessing the page content, please contact support@etixpert.com .

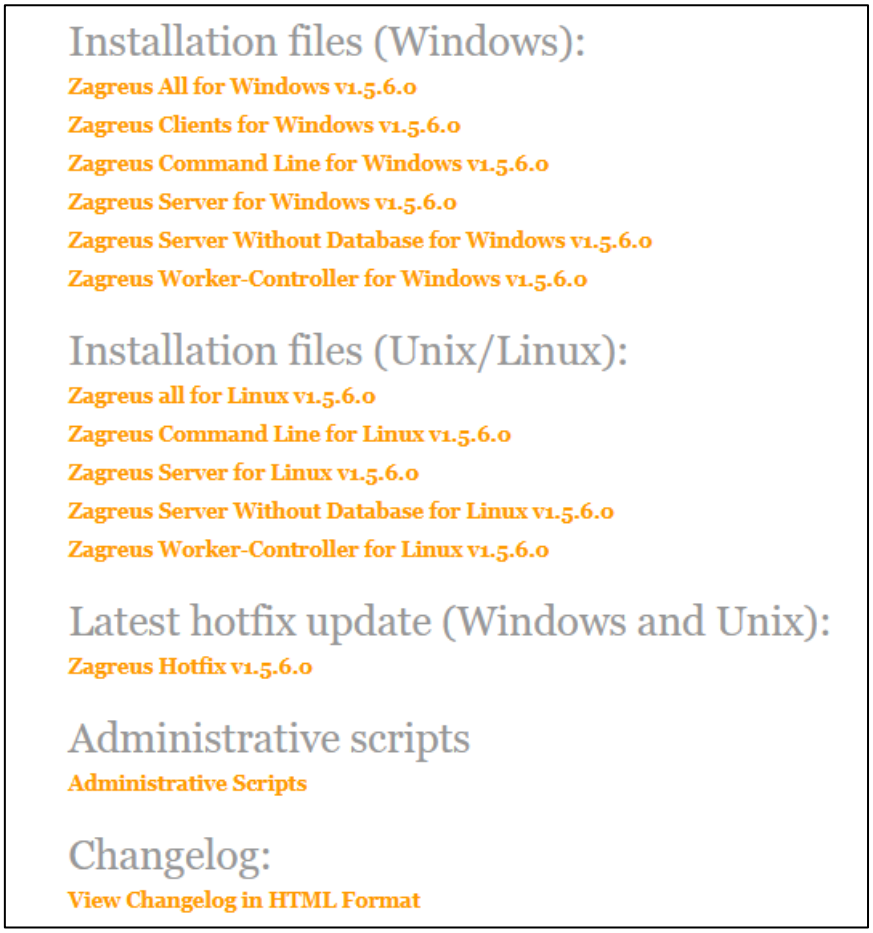


Figure 1 – The download page of the Zagreus Support site as of the writing of the documentation

On this site, the following archive types can be downloaded (the link title also contains the name of the operation system and the version of Zagreus):

- *All*
  Contains the Zagreus Server, Zagreus Client, Zagreus Monitor, Zagreus Commandline and Zagreus Worker-Controller (and the embedded Zagreus Worker) modules.

- *Clients*
  Contains the Zagreus Client and the Zagreus Monitor modules.

- *Command Line*
  Contains the Zagreus Command Line module.

- *Server*
  Contains the Zagreus Server module with the embedded MySQL database.

- *Server Without Database*
  Contains the Zagreus Server module without the embedded database.

- *Worker-Controller*
  Contains the Zagreus Worker-Controller and the embedded Zagreus Worker modules.

Since the Zagreus Client and the Zagreus Monitor modules are available only to Windows OS, not all of above-listed archive types can are available for Linux platform either.

3.2 Installation on Windows

Under the 'Installation files (Windows)' section, download the '*Zagreus All for Windows v1.5.x.x*' installation package. After downloading, the following steps are required to install the Zagreus System on a Windows environment:

- copying the files to the target installation folder
- setting the proper configuration parameters
- setting up the Windows services

Next, these steps are described in details.

3.2.1 Copying the installation files

First, the user has to select a folder which Zagreus will be installed in. By default, the OS path of this folder is `C:\Programme\zagreus` . All the contents of the downloaded installation .zip archive must be extracted to this target installation folder.

> ⚠ ***Warning:*** *When Zagreus is installed in a different folder than the default one (such as* `C:\Program Files\zagreus` *), the configuration files must be changed accordingly.*

3.2.2 Setting the configuration parameters

After copying the Zagreus files to the target installation folder, specific configuration parameters might be set. These parameters are pre-configured in the downloaded Zagreus installation .zip archive; editing them is required only when the Zagreus target installation folder path differs from the default one (i.e. `C:\Programme\zagreus`).

These particular settings are located in three files:

- the `setenvironment.bat` file in the installation root folder
- the `wrapper.conf` configuration file of the Server module
- the `wrapper.conf` configuration file of the Worker-Controller module

3.2.2.1 setenvironment.bat

First the file `setenvironment.bat` (located in the root folder of the Zagreus target installation folder) has to be reviewed: the variables `JAVA_HOME` and `ZAGREUS_HOME` must be defined accordingly, when the target installation folder path differs from the default value of `C:\Programme\zagreus` . For example, if the target installation folder path is `C:\Program Files\zagreus`, the edited part of the `setenvironment.bat` file should look like:

```
set JAVA_HOME="C:\Program Files\zagreus\java\openjre11"
set ZAGREUS_HOME="C:\Program Files\zagreus\server"
```

3.2.2.2 wrapper.conf (Server module)

The file `wrapper.conf` (located in the `server/service/conf` folder within the Zagreus target installation folder) also has to be reviewed: the properties `wrapper.java.command` and `wrapper.java.additional.1` must be defined accordingly, when the target installation folder path differs from the default value of `C:\Programme\zagreus` . For example, if the target installation folder path is `C:\Program Files\zagreus`, the new values should look like:

```
wrapper.java.command=”C:/Program Files/zagreus/java/openjre11/bin/java”
wrapper.java.additional.1=-Dworking.folder="C:/Program
Files/zagreus/server"
```

Note, that in the `wrapper.conf` file, the OS paths contain ‘/’ characters in the paths unlike in the `setenvironment.bat` previously.

3.2.2.3 wrapper.conf (Worker Controller module)

The file `wrapper.conf` (located in the `worker-controller/service/conf` folder within the Zagreus target installation folder) may also need to be reviewed: the properties `wrapper.java.command`, `wrapper.java.additional.1` and `wrapper.java.additional.2` must be defined accordingly, when the target installation folder path differs from the default value of `c:\Programme\zagreus` . For example, if the target installation folder path is `c:\Program Files \zagreus`, the new values should look like:

```
wrapper.java.command=”C:/Program Files/zagreus/java/openjre11/bin/java”
wrapper.java.additional.1=-Dworking.folder="c:/Program
Files/zagreus/worker-controller"
wrapper.java.additional.2=-Djava.folder=”c:/Program Files/zagreus/java”
```

Note, that in the `wrapper.conf` file, the OS paths contain ‘/’ characters in the paths unlike in the `setenvironment.bat` previously.

⚠ ***Warning:*** *If the installation path contains space characters, quotations must be used, as shown in the samples above.*

3.2.3 Setting up Zagreus Windows services

The two main modules of the Zagreus System, the Zagreus Server and the Zagreus Worker-Controller are designed to run as Windows services on the Windows OS system.

To install these two modules as Windows services, execute the following scripts in command line with administrator privileges (the paths are relative to the Zagreus target installation folder):

- `InstallApp-NT.bat` in the `server/service/bin` folder
- `InstallApp-NT.bat` in the `worker-controller/service/bin` folder

After installing the Windows services, the proper administrator rights for both services have to be set by the following steps:

1) Start the *Run* Windows application and type `services.msc` to run the *Services* system application.
2) If necessary, navigate to the *Extended* tab in the *Services* window, then right-click on the *Zagreus-Server* service and select the *Properties* menu item from the context menu.
3) Click on the *Log On* tab on the opened *Properties* dialog box and set up the Administrator account (or an account with administrator rights) of the current Windows installation, see *Figure 2*.

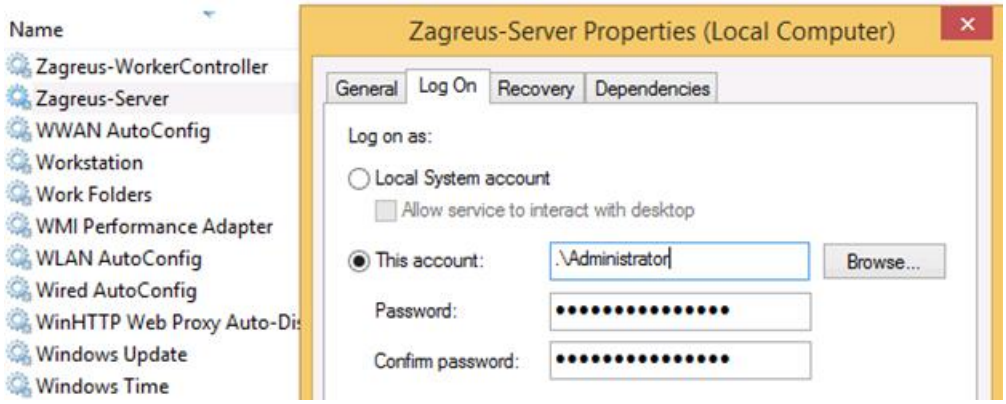


Figure 2 – Setting up administrator privileges for the *Zagreus-Server* service

4) Repeat steps 2) and 3) for the *Zagreus-WorkerController* service.

3.2.4 Opening ports in the firewall

To allow external connections to the Zagreus Server, its HTTP connection port must be allowed via the firewall. By default, the Zagreus Server module communicates over the port 7323 for HTTP and port 7443 for HTTPS connections, but these can be modified (see → General properties).

For instructions how to open specific ports on Windows, consult your system administrator.

3.2.5 Starting the Zagreus services

After installing Zagreus, the Zagreus Server and the Zagreus Worker-Controller modules can be started / stopped by starting / stopping the corresponding services (i.e. *Zagreus-Server* and *Zagreus-WorkerController*). The services can be configured to be automatically started when the Windows system boots up. Alternatively, these services can be started manually by the *Services* system application, accessible either from the *Task Manager* (and by selecting the *Services* tab) or via the *Run* application by starting the `services.msc` command.

3.2.6 Starting the Zagreus clients

To start the Zagreus Client or the Zagreus Monitor applications, the following batch files have to be executed from the target installation root folder:

- `startgui_en.bat` : Starts the Zagreus Client application (English version)
- `startgui_de.bat` : Starts the Zagreus Client application (German version)

- `startmonitor.bat` : Starts the Zagreus Monitor application

> ⚠ **Warning:** *If the Zagreus installation folder is located under the Windows default program folder (e.g.* `C:\Program Files` *) then client applications must run with administrator privileges* – Run as Administrator!

Note that the Zagreus Client and the Zagreus Monitor modules can be downloaded as standalone applications as well. For detailed instructions, see also → Downloading Zagreus.

3.3 Installation on Linux

Under the 'Installation files (Linux)' section, download the '*Zagreus All for Linux v1.5.x.x*' installation package to a temporary folder. Note: it is not possible to install the Zagreus Client and the Zagreus Monitor modules for Linux, as these modules support only the Windows operation system. After downloading, the following steps are required to install the Zagreus System on a Linux environment:

- creating the target installation folder
- unpacking the archive and copying the unpacked files to the target installation folder
- setting the proper configuration parameters

Next, these steps are described in details.

3.3.1 Creating the target installation folder

First, the root user has to create the folder which Zagreus will be installed in. The recommended OS path of this folder is `/home/zagreus`, which path is pre-configured in the shipped configuration files. For a different installation folder path, the steps must changed accordingly.

```
su root
mkdir /home/zagreus
```

Due to the limitations of MySQL (the embedded SQL database shipped with Zagreus), Zagreus cannot be started as the `root` user. Therefore, it is recommended to configure the `hostmaster` user to start the Zagreus System. The installation steps shown below are still performed by the `root` user though.

As the first step, the target installation folder needs to be created, and the ownership of this folder must be handed over to the `hostmaster` user and the `users` group.

```
chown hostmaster /home/zagreus
chgrp users /home/zagreus
```

> ⚠ ***Warning:*** *Zagreus cannot be started as the root user due to the limitations of the MySQL server!*

After setting up the correct privileges it is recommended to switch to the `hostmaster` user, otherwise the files that will be copied later will belong to the `root` user.

```
su hostmaster
```

3.3.2 Unpacking the archive file

Next, the downloaded archive (e.g. `zagreus_linux_install.tar.gz`) has to be unpacked into the temporary folder which it was downloaded to (assumed to be the current folder):

```
tar -xzvf zagreus_linux_install.tar.gz
```

The user then needs to copy the unpacked files (i.e. the contents of the `zagreus` folder) to the target installation folder:

```
cp -ra zagreus/* /home/zagreus
```

3.3.3 Editing the set_environment.sh file

After copying the Zagreus files to the target installation folder, the file `set_environment.sh` (located in the root folder of the Zagreus target installation folder) has to be reviewed: the variable `ZAGREUS_INSTALLATION_HOME` must be defined accordingly, when the target installation folder path differs from the default value of `/home/zagreus` . For example, if the target installation folder path is `/home/zagreus_installation`, the edited part of the `set_environment.sh` file should look like:

```
export ZAGREUS_INSTALLATION_HOME=/home/zagreus_installation
```

3.3.4 Opening ports in the firewall

To allow external connections to the Zagreus Server, its HTTP connection port must be allowed via the firewall. By default, the Zagreus Server module communicates over the port 7323 for HTTP and port 7443 for HTTPS connections, but these can be modified (see → General properties).

For instructions how to open specific ports on Linux, consult your system administrator.

3.3.5 Starting and administering Zagreus

The Zagreus Server and Zagreus Worker-Controller modules can be started, stopped and restarted by using the `zagreus.sh` script, located in the Zagreus target installation folder root. The user has to provide one of the following commands as the only command line parameter for the aforementioned script (e.g. issuing the command '`./zagreus.sh start`'):

- *start*: starts the Zagreus Server module
- *stop*: stops the Zagreus Server module except when there are queued or running jobs (about job statuses, see → Job lifecycle)
- *forcestop*: forces the Zagreus Server module to stop without checking any queued or running jobs
- *start-wc*: starts the Zagreus Worker-Controller module
- *stop-wc*: stops the Zagreus Worker-Controller module
- *status*: prints the status of the following processes: Zagreus Server, Zagreus Worker-Controller, Zagreus Workers and the embedded MySQL database

Warning: *When the Zagreus Server is installed in a Linux environment, the Zagreus Client and Zagreus Monitor applications have to be installed separately, in a Windows environment.*

3.4 Sending the licence key

The various features of Zagreus are accessible depending on the installed Zagreus licence. The properties of the Zagreus licence is managed via the Zagreus *licence key*, issued by Etixpert GmbH. By default, the downloaded Zagreus Server is shipped with a demo licence key to allow the user to perform the very basic operations. When the administrator user obtains a new custom licence key, this licence key needs to be sent to the Zagreus Server. This can be done in the Zagreus Client application: after right-clicking the server definition node, the user has to select the *Get licence information…* menu item from the appearing context menu, that opens the *Licence info* dialog box, see *Figure 3*.

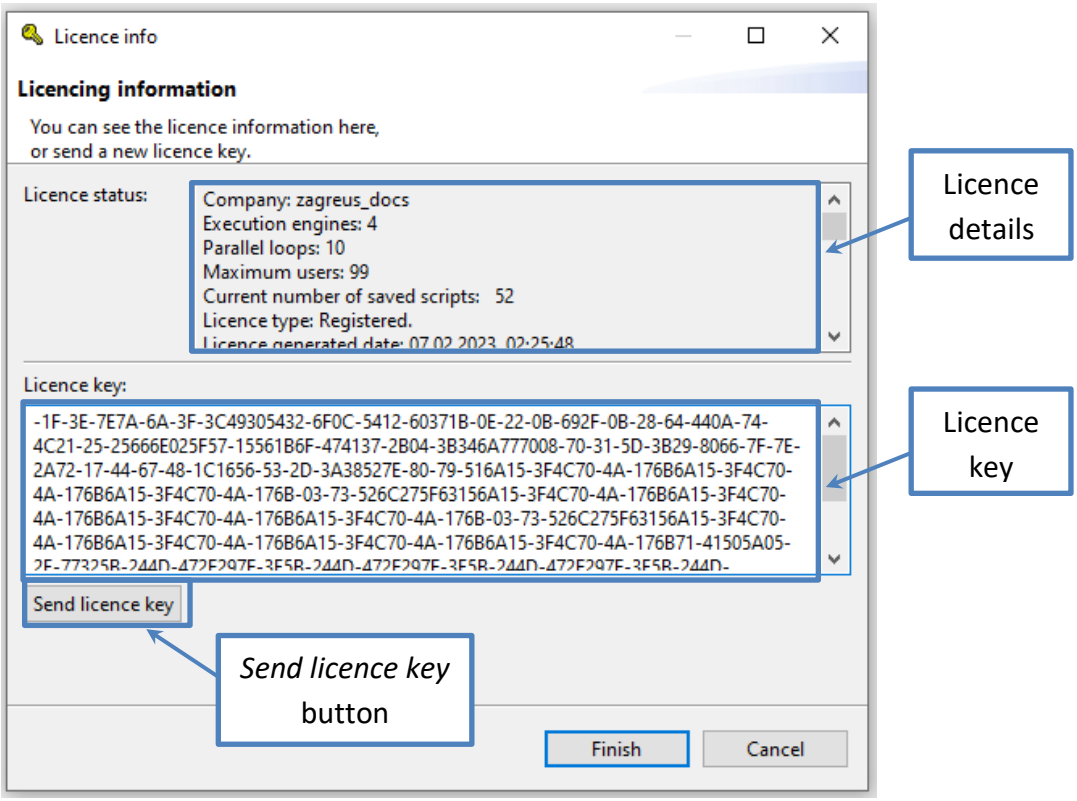


Figure 3 – The *Licence info* dialog box

The *Licence status* textbox contains the details of the currently installed (demo) licence. To install the obtained custom licence key, the user needs to paste the key into the *Licence key* text area and click on the *Send license key* button, see *Figure 3*.

For further details on the Zagreus licence features, see → Licencing.

3.5 Standalone installation of the client modules

The Zagreus Client and the Zagreus Monitor modules can be downloaded and installed separately as standalone applications in order to connect to a remotely installed Zagreus Server. This is useful when the Zagreus Server is installed on a Linux environment, or any of the Zagreus client applications is installed on multiple PCs, which need to access the same Zagreus Server remotely.

To install one or both of these client applications separately, the user first needs to navigate to the section '*Installation files (Windows)*' in the download page of the Zagreus Support site (see → Downloading Zagreus) and download the '*Zagreus Clients for Windows v1.5.x.x*' installation package. After downloading, the following steps are required to install the Zagreus client applications:

- copying the files to the target installation folder
- setting the proper `JAVA_HOME` environment variable (only for the Zagreus Client application)

Next, these steps are described in details.

Warning: *The Zagreus Client and Zagreus Monitor applications can be installed only on Windows operating system!*

3.5.1 Copying the installation files

First, the user has to select a folder which the Zagreus client applications will be installed in. All the contents of the downloaded installation .zip archive must be extracted to this target installation folder (e.g. `C:\Programme\Zagreus Clients`).

The installation .zip archive contains three folders:

- `gui`: contains the Zagreus Client application
- `monitor`: contains the Zagreus Monitor application
- `java`: contains the Java Runtime Environment (JRE) recommended for the Zagreus Client and Zagreus Monitor applications

If the user wants to install only one of the Zagreus client applications, it is enough to copy the content of the corresponding application folder and the `java` folder to the target installation folder.

3.5.2 Setting the *JAVA_HOME* environment variable

To run the Zagreus client applications, the value of the `JAVA_HOME` system variable has to be set properly in the `setenvironment.bat` file. For example, if the target installation folder path is `C:\Programme\Zagreus Clients`, the edited part of the `setenvironment.bat` file should look like:

```
set JAVA_HOME="C:\Programme\Zagreus Clients\java\openjre11"
```

Info: *When installed as standalone applications, the Zagreus Client and the Zagreus Monitor modules do not use the* `ZAGREUS_HOME` *setting, therefore the definition of this property can be ignored.*

3.5.3 Starting the Zagreus clients

After proper installation and configuration, the client applications can be started by executing the following batch files:

- `startgui_en.bat` : Starts the Zagreus Client application (English version)
- `startgui_de.bat` : Starts the Zagreus Client application (German version)
- `startmonitor.bat` : Starts the Zagreus Monitor application

Warning: *If the Zagreus installation folder is located under the Windows default program folder (e.g.* `C:\Program Files` *) then client applications must run with administrator privileges* – Run as Administrator!

3.6 Troubleshooting

The installation might enounter some specific problems. Next some suggestions will be described in details for these possible cases.

3.6.1 Issues independent of the operating system

3.6.1.1 RMI port conflict

In rare cases it can happen that Zagreus server cannot start up because RMI ports are already used. In this case, new RMI ports need to be defined for Zagreus. Please, check configuration section for further information – see also → Configuration

3.6.2 Issues on Windows

3.6.2.1 Enabling logging for the Java Service Wrapper

If Zagreus services cannot start up and there are no log files generated for the Server and Worker-Controller, the user can analyse the problem by enabling logging for the Java Service Wrapper by editing the following configuration files:

- *for Zagreus Server:*
  `<zagreus_home>\server\service\conf\wrapper.conf`

- *for Zagreus Worker-Controller:*
  `<zagreus_home>\worker-controller\service\conf\wrapper.conf`

Uncomment the logging-related variables in the configuration files, i.e.:

```
# in the Server wrapper.conf
wrapper.logfile=log/srv_YYYYMMDD.log

# in the Worker-Controller wrapper.conf
wrapper.logfile=log/wc_YYYYMMDD.log
```

After starting the services again, the following wrapper log files will be created:

- *for Zagreus Server:*
  `<zagreus_home>\server\log\srv.log`

- *for Zagreus Worker-Controller:*
  `<zagreus_home>\worker-controller\log\wc.log`

3.6.3 Issues on Linux

The content of the `set_environment.sh` file should be double-checked, see → Editing the set_environment.sh file.

4. Configuration

Zagreus consists of several modules (for an overview, see → Zagreus as a whole system). The user can configure the modules by editing specific configuration files for each module.

After installation, the configuration files of these modules are already prepared for a fully functional Zagreus System. However, further fine-tuning of these configuration files might be necessary, depending on the actual requirements.

Next the available parameters available in the three configuration files will be described in details.

4.1 Zagreus Server configuration

The OS path of the Zagreus Server module configuration file is `server/conf/conf.properties` (relative to the Zagreus target installation folder). In it, the following properties can be set:

4.1.1 General properties

- *server.listener.port*
  The HTTP port of the Zagreus Server module. This port must be specified for the client applications like for the Zagreus server definition node in the Zagreus Browser window in the Zagreus Client application. Note that this value applies only when the specified connection is not in *secure* mode; when the connection is secure, the value of the *ssl.listener.port* property is used.
  Default value: *7323*

- *server.rmi.port*
  The Java RMI port, which is used by the Zagreus Worker-Controller and Zagreus Worker instances when connecting to the Zagreus Server. This setting also has to be configured consistently in the Zagreus Worker-Controller and the Zagreus Worker configuration files (see → Zagreus Worker-Controller configuration and → Zagreus Worker configuration).
  Default value*: 6666*

- *filesystem.root*
  The OS path of the Zagreus local filesystem root. This path can be absolute, but it is recommended to use the *${working.folder}* subtitution string, see → File paths.
  Examples: (on Linux)

```
filesystem.root=/your/absolute/path/to/filesystem
filesystem.root=${working.folder}/filesystem
```

  Default value: *${working.folder}/filesystem*

- *job.logpath*
  The OS path of the folder where the job-log files are stored. This path can be absolute, but it is recommended to use the *${working.folder}* subtitution string, see → File paths.
  Default value*: ${working.folder}/log/job*

4.1.2 Server startup and shutdown properties

- *startscheduler*
  Sets whether the scheduler component starts up when the server is started. Setting this property to *false* is useful when the Zagreus System needs to be started without any automatic script execution, e.g. debugging.
  Possible values: *true* / *false*
  Default value*: true*

- *server.canceljobs.onstop*
  Sets if all running and queued jobs should be canceled when the server is shutting down.
  Possible values: *true* / *false*
  Default value*: true*

- *server.canceljobs.onstart*
  Sets if all running and queued jobs should be canceled when the server is starting. These can be useful when there are stuck jobs in the job queue.
  Possible values: *true* / *false*
  Default value*: false*

4.1.3 SSL properties

- *ssl.listener.port*
  The port for secure Zagreus HTTPS connection. This port must be specified for the client applications like for the Zagreus server definition node in the Zagreus Browser window in the Zagreus Client application. Note that this value applies only when the specified connection is in *secure* mode; when the connection is not secure, the value of the *server.listener.port* property is used.
  Default value*: 7443*

- *ssl.listener.cipher-suites*
  The list of accepted SSL cipher suites.
  Default value:
  SSL_RSA_WITH_RC4_128_MD5,SSL_RSA_WITH_RC4_128_SHA,SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

- *ssl.listener.algorithm*
  Sets the keystore file management algorithm for the embedded webserver.
  Default value: *ssl.listener.algorithm=SunX509*

- *trustedstore.filename*
  The name of the SSL truststore file. This is the file which the additional SSL certificates have to be installed into for secure external connections, see → Secure connections. The location of this file is always `/zagreus_home/server/conf/ssl`
  Default value*: client.trustedservers*

4.1.4 MySQL properties

The Zagreus Server is shipped with an embedded MySQL database for storing its internal data. The following properties can be used to configure the MySQL database.

- *mysql.port*
  The port of the MySQL database.
  Default value*: 3336*

- *mysql.basedir*
  The OS path of the root directory of MySQL database. This path can be absolute, but it is recommended to use the *${working.folder}* subtitution string, see → File paths.
  Default value*: ${working.folder}/mysql*

4.1.5 Queue group properties

It is possible to group workers into *queue groups* (see → Queue groups). The specific settings of grouping workers are the following:

- *queue.groups.num*

The number of worker groups for the queue.
Default value*: 1*

- *queue.groups.<WorkerGroupID>.worker*
  This property defines the worker identifiers for each queue group.
  The following example defines two worker groups with four and two workers, respectively:

```
queue.groups.1.worker=1.1, 1.2, 1.3, 1.4
queue.groups.2.worker=1.5, 1.6
```

4.1.6 Password policy properties

The following properties configure the settings of the password policy. Password policy can be switched on for any user in the Zagreus Client, see → Zagreus Client.

- *password.validity*
  The number of days after the password expires.
  Default value: *90*

- *password.expiration.reminder*
  There is a reminder for the user before the password expires. The user gets this reminder as a warning dialog box when conneting to the Zagreus Server in the Zagreus Client. The number of days before the reminder starts to show up can be configured by this property.
  Default value: *7*

- *password.reusable.after*
  The number of password changes after which the same password can be used again.
  Default value: *6*

- *password.minimum.length*
  The minimum length of the password.
  Default value: *10*

- *password.minimum.capital*

The minimum number of upper-case characters in the password.
Default value: *1*

- *password.minimum.numeric*
  The minimum number of numeric characters in the password.
  Default value: *1*

- *password.minimum.special*
  The minimum number of special characters in the password, which are: `& @ # ' " + % / = ( ) . , ; : ? ! * < > - _` and space.
  Default value: *1*

4.1.7 Trigger and watcher properties

- *filetrigger.double.trigger.limit*
  Sets the minimum time difference between two subsequent trigger events for the same file to treat them as two different events, see also → Server-side configuration. This value is defined in milliseconds.
  Default value*: 500*

- *watcher.counter.policy*
  Sets when the *actual value* property of a watcher is supposed to be decreased, see → Scheduling section . The possible values are:

  - *evaluate*: when the condition is evaluated
  - *condition_true*: when the condition is evaluated to true
  - *script_run*: when the condition is evaluated to true and at least one script is going to be executed

4.1.8 Miscellaneous properties

- *userecyclebin*
  Sets whether the server uses the recycle bin (see → Recycle bin) when deleting a resource.
  Possible values: *true* / *false*
  Default value*: true*

- *bankholidays.path*
  Defines the full path of the optional bank holidays descriptor file, see → Bank holidays feature.

- *variable.server.docurl*
  Sets the URL for the optional Document URL feature, see → Document URL feature.

- *variable.server.docurl_replace*
  Sets the additional substring replacements for the optional Document URL feature, see → docurl_replace variable.

- *queue.skippedjobs.threshold*
  Defines time threshold when a job must be marked as *skipped* in Zagreus Monitor, see → Skipped jobs in the Zagreus Monitor. The value is defined in milliseconds.
  Default value*: 5000 (5 seconds)*

4.1.9 Server-level execution options

The configuration file of the Zagreus Server can contain server-level options as well (see → Declaration levels). These are optional, and defined in the same `key=value` format as the aforementioned properties. The keys defining these options must start with the `option.server` prefix. The following example defines the execution option *running_timeout* set to *50000*:

```
option.server.running_timeout=50000
```

For more details on the option resolution precedence order, see → Precedence order for resolution.

4.1.10 Server-level and queue-level variables

The configuration file of the Zagreus Server can contain server-level and queue-level variables as well (see → Declaration levels). These are fully optional, and defined in the same `key=value` format as the aforementioned properties. The keys defining these variables must start with the `variable.server` or `variable.queue` prefix, respectively. The following example defines a server variable *x* with the value *2*:

```
variable.server.x=2
```

For more details on the start-up variables resolution precedence order, see → Precedence order for resolution.

4.2 Zagreus Worker-Controller configuration

The OS path of the Zagreus Worker-Controller module configuration file is `worker-controller/conf/workercontroller.properties` (relative to the Zagreus target installation folder). In it, the following properties can be set:

4.2.1 General properties

- *workercontroller.id*
  The ID of the Worker-Controller, which must be unique (to avoid conflict if there are multiple Worker-Controller instances installed).
  Default value*: 1*

- *server.host*
  The URL of the Zagreus Server host (accessed through Java RMI protocol).
  Default value*: rmi://localhost*

- *server.rmi.port*
  The Java RMI port of the Zagreus Server to which the Worker-Controller is connecting. This setting also has to be configured consistently in the Zagreus Server and the Zagreus Worker configuration files (see → Zagreus Server configuration and → Zagreus Worker configuration).
  Default value: *6666*

- *workercontroller.servercommandservicepolltimeout*
  Defines the timeout when the Worker-Controller is trying to reconnect to the Zagreus Server in suspended mode, see → Suspended mode. This value is defined in seconds.
  Default value: *60*

- *workercontroller.serverconnectioncheckfrequency*
  Defines the time period between two trials when the Worker-Controller is trying to reconnect to the Zagreus Server in suspended mode. This value is defined in milliseconds.
  Default value: *5000*

4.2.2 Worker-related properties

- *worker.rootfolder*
  The folder where the Zagreus Worker is installed. By default, it is located in the `/worker` subfolder in the Worker-Controller module installation root folder. This path can be absolute, but it is recommended to use the *${working.folder}* subtitution string, see → File paths.
  Default value*: ${working.folder}/worker*

- *workercontroller.defaultworkercount*
  Defines the number of workers which are started when the Worker-Controller module starts up, see → Number of Zagreus Workers. If it is set to -1, no Zagreus Workers are started by default before the connection to the Zagreus Server is established; after the connection is made, the maximum number of Zagreus Workers allowed in the Zagreus Licence will be started, see → Licencing.
  Default value*: -1*

- *workercontroller.autokillcommand*
  When the Worker-Controller module is starting up, it automatically checks if there is any running Zagreus Worker instance (accidentally stuck and could not be stopped). For safety reasons, the module can automatically execute a command-line command to kill any of these stuck Worker instances. This property defines this OS command.

  Recommended value on Windows:

```
workercontroller.autokillcommand=taskkill /f /im \"zagreus-worker.exe\"
```

  Recommended setting on Linux/Unix:

```
workercontroller.autokillcommand=killall -9 zagreus-worker
```

- *workercontroller.workerrestarttimeout*
  Defines the time threshold after which the Worker-Controller restarts a particular Worker in case when the connection between the Worker-Controller and any of the Zagreus Workers is lost. This value is defined in milliseconds.
  Default value: *30000*

- *workercontroller.softcanceltimeout*
  When a job is canceled, the Zagreus Server sends a cancel command to the Worker-Controller module. The Worker-Controller first tries to do a soft cancelation, and if it is not successful, it restarts the whole JVM of the particular Zagreus Worker instance. This property defines how long the Zagreus Worker-Controller waits for the soft cancelation. This value is defined in milliseconds.
  Default value: *1000*

- *workercontroller.workerpollingfrequency*
  Defines the frequency at which the Worker-Controller module obtains information from the Zagreus Worker instances, see → Collecting and sending worker information. This value is defined in milliseconds.
  Default value: *2000*

- *workercontroller.monitorinfofrequency*
  Defines the frequency at which the Worker-Controller module sends the worker information to the Zagreus Server module, see → Collecting and sending worker information. This value is defined in milliseconds.
  Default value: *2000*

4.3 Zagreus Worker configuration

The OS path of the configuration file of the Zagreus Worker-Controller module is `worker-controller/worker/conf/worker.properties` (relative to the Zagreus target installation folder).

4.3.1 Property lists

In the `worker.properties` configuration file, there are special properties which belong together as a parameter list, such as *worker.javaoptions* and *worker.parameters* . The general declaration format of these property lists is *worker.<property-name>.<property-index> ,* for example:

```
worker.javaoptions.1=-Xmx4096M
worker.javaoptions.2=-Xms256M
worker.javaoptions.3=-Djava.io.tmpdir=temp
```

4.3.2 Property declaration for specific Worker instances

Most property keys in the `worker.properties` file can be declared either in a general or in an instance-specific way. Instance-specific declaration allows the user to configure any particular Worker instance differently when needed. Configuring Worker instances with specific parameter values can be useful in combination with queue groups, see use case → Queue groups.

The general format is *worker.<property-name>*, while the instance-specific format contains the particular Worker ID: *worker.<worker-ID>.<property-name>* . Resolving the value of a specific Worker property is performed in the following order:

- instance-specific property declaration
- general property declaration
- default value of the general property

For example, the MSTR classpath of the Worker instances can be declared generally in the following way:

```
worker.classpath.mstr=lib/mstr/11.3.0760/*
```

For Worker ID=2, it can be overridden with another value:

```
worker.2.classpath.mstr=lib/mstr/11.2.0/*
```

For property lists, overriding a specific list element can be declared in the format *worker.<worker-ID>.<property-name>.<property-index>* . For example, overriding the second Java option for Worker ID=3 can be done with the following declaration:

```
worker.3.javaoptions.2=-Xms512M
```

4.3.3 Worker startup properties

When the Zagreus Worker-Controller starts the Zagreus Worker instances one by one (see → Starting Zagreus Workers), it reads the following property values from the `worker.properties` configuration file:

- *worker.javabin*
  Defines the local OS path for the Java executable (by default it is set to use the one shipped with Zagreus). The user should avoid using backslashes in the path. This path can be absolute, but it is recommended to use the *${working.folder}* or the *${java.folder}* subtitution string, see → File paths.

- *worker.classpath*
  Defines the main classpath for Zagreus Worker execution. This path can be absolute, but it is recommended to use the *${working.folder}* or the *${java.folder}* subtitution string, see → File paths.

⚠ **Warning:** *To set the* worker.classpath *property under Linux, use colons (':') instead of semicolons (';') as delimiters!*

- *worker.classpath.mstr*
  Defines the classpath specifically for MicroStrategy libraries. Since MicroStrategy external libraries change more often than other libraries, in such cases the user only needs to update the value of this property. This path can be absolute, but it is recommended to use the *${working.folder}* subtitution string, see → File paths. For example, to use the libraries for MicroStrategy 11.3.0760 version, specify the following :

  ```
  worker.classpath.mstr=lib/mstr/11.3.0760/*
  ```

On Linux/Unix systems, use the following format:

```
worker.classpath.mstr=${working.folder}/lib/mstr/11.3.0760/*
```

- *worker.javaoptions*
  Defines additional JVM command line options. This is a property list, see → Property lists, so different numbers must be used as postfixes. For example:

```
worker.javaoptions.1=-Xmx4096M
worker.javaoptions.2=-Xms256M
```

  Instance-specific way (see → Property declaration for specific Worker instances) of property declaration is also possible:

```
worker.1.javaoptions.5=-Dworker.self.id=1
worker.2.javaoptions.5=-Dworker.self.id=2
```

- *worker.classname*
  Defines the fully qualified name of the main Java class of the Zagreus Worker module. Instance-specific way of declaration is not accessible for this property. Default value: *com.etixpert.zagreus.worker.impl.Worker*

- *worker.parameters*
  Defines additional start-up command-line parameters for the Zagreus Worker module. This is a property list, see → Property lists, so different numbers must be used as postfixes. For example:

```
worker.parameters.1=additional_testparam1
worker.parameters.2=additional_testparam1
```

  Currently this property is not used.

4.3.4 Connection properties

- *worker.serverport*
  The http port of the Zagreus Server module. Must match the value of the *server.listener.port* property defined in the Zagreus Server configuration (see → Zagreus Server configuration). This port is used for creating the default `zs` local connection, see → zs connection.

Default value: *7323*

- *server.rmi.port*
  The Java RMI port of the Zagreus Server. Must match the value of the *server.rmi.port* property defined in the configuration file of the Zagreus Server (see → General properties) and the *server.rmi.port* property defined in the configuration file of the Zagreus Worker-Controller modules (see → General properties).
  Default value: *6666*

4.3.5 Miscellaneous properties

- *worker.maxmemorytorestart*
  After the job execution has finished, a Zagreus Worker module can restart itself when its memory allocation exceeds a certain limit. This property sets the memory limit of this feature. Value *-1* indicates that the given Worker should never be restarted, while value *0* indicates that the Worker must be restarted after each job execution. This value is expressed in bytes, but the following postfixes are recommended to use :

  - *k* or *K*: 1024 (i.e. kilobytes)
  - m or M: 1024 kilobytes (i.e. megabytes)
  - g or G: 1024 megabytes (i.e. gigabytes)

  For example, the following setting sets the memory limit to 500 megabytes:

  ```
  worker.maxmemorytorestart=500M
  ```

  Default value: *-1*

- *trustedstore.filename*
  The name of the SSL truststore file. This is the file which the additional SSL certificates have to be installed into for secure external connections, see → Secure connections. The location of this file is always `/zagreus_home/worker-controller/worker/conf/ssl`
  Default value*: client.trustedservers*

- *worker.filesystem.path*
  Specifies the worker filesystem path (see → Local filesystem in the Zagreus Worker) relative to the Zagreus Worker root folder.
  Default value: */filesystem*

- *worker.standalone.joblog.path*
  Specifies the joblog path that the Standalone Worker module (see → Standalone Worker) uses, relative to the Zagreus Worker root folder.
  Default value: */joblog*

4.3.6 Worker-level execution options

The configuration file of the Zagreus Worker can contain one specific worker-level option: *log_level*, see → List of execution options. This is optional, and it must start with the `option.worker` prefix. The following example defines the execution option *log_level* set to *debug*:

```
option.worker.log_level=debug
```

For more details on the option resolution precedence order, see → Precedence order for resolution.

4.3.7 Worker-level variables

The configuration file of the Zagreus Worker can contain worker-level variables as well (see → Declaration levels). These are fully optional, and defined in the same `key=value` format as the aforementioned properties. The keys defining these variables must start with the `variable.worker` prefix. The following example defines a worker variable *x* with the value *3*:

```
variable.worker.x=3
```

For more details on the variable resolution precedence order, see → Precedence order for resolution.

The worker variables can be declared in an instance-specific way, just like most of the other properties listed above. The following examples show different variable declarations for different worker instances:

```
variable.worker.1.x=3
variable.worker.2.x=6
```

4.4 File paths

In the various configuration files there are properties which need to specify a local OS path. If the user does not like to use an absolute OS path, there is the possibility to use a substitution for the working folder root of the corresponding module: if the string `${working.folder}` is present in the path definition, that will be substituted. For example, in the following example:

```
filesystem.root=${working.folder}/filesystem
```

If the working folder of the Zagreus Server module is `C:\Program Files\zagreus\server`, the value will be resolved as:

```
filesystem.root=C:\Program Files\zagreus\server\filesystem
```

The properties which support this substitution are the following:

- *filesystem.root* in the Zagreus Server module
- *job.logpath* in the Zagreus Server module
- *mysql.basedir* in the Zagreus Server module
- *worker.rootfolder* in the Zagreus Worker-Controller module
- *worker.classpath* in the Zagreus Worker module
- *worker.classpath.mstr* in the Zagreus Worker module
- *worker.javabin* in the Zagreus Worker module

Furthermore, for properties *worker.javabin* and *worker.classpath*, there is the possibility to refer to the OS path of the Java root folder in the Zagreus target installation root folder. If the string `${java.folder}` is present in the paths of these properties, that will be resolved as `<zagreus_home>/java` . For example, in the following example:

```
worker.javabin=${java.folder}/openjre11/bin/zagreus-worker.exe
```

If the working folder of the Zagreus Server module is `C:\Program Files\zagreus\server`, the value will be resolved as:

```
worker.javabin=C:\Program  Files\zagreus\server\java\openjre11\bin\zagreus-worker.exe
```

Info: *In the Zagreus System, Java executables appear with names like* zagreus-server*,* zagreus-wc*,* zagreus-worker *and* zagreus-monitor *(with .exe extension in Windows) for better maintainability.*

5. System setup and administration

After the installation and configuration has been successfully performed, Zagreus needs a proper system setup including licence installation and basic user management. Optionally, recurring administrative tasks can be configured for the Zagreus System. Next, these features will be discussed in detail.

5.1 Licencing

Zagreus has plenty of features and provides efficient solutions for many possible problems. Most of its functionality is offered by the large selection of actions and actions groups. The customer needs to buy a Zagreus Licence to be able to use the Zagreus System. The Zagreus Licence defines the feature-sets that the customer needs, expiration date and some other personalized information. The licence is sent as an encoded licence key to the customer and it needs to be installed on the already configured Zagreus System.

5.1.1 Content of a Zagreus Licence

The Zagreus Licence contains the following information:

- *Licence type*
  The type of the Zagreus licence. It can be one of the following:
  - *Registered:*
    This is the standard licence type. A company needs to request a registered licence to be able to use specific action groups and specific number of Zagreus Workers. The licence also contains settings for all the other listed features.
  - *Demo:*
    This type of licence allows the trying of Zagreus components for a short period of time. The downloadable Zagreus installation provides a demo licence by default.
- *Company*
  The name of the company that Zagreus is registrered for.
- *Licence generated date*
  The time the license was created.
- *Expiration date*
  The time the license will be expired.

- *Execution engines*
  The maximum number of Execution engines running at the same time (see → Number of Zagreus Workers).

- *Parallel loops*
  This specific property constraints the maximum number of parallel threads in the `z:foreach` action. This powerful feature allows the script to be executed efficiently in a parallel manner.

- *Maximum users*
  The maximum number of users that can be created on the Zagreus Server.

- *Enabled action groups*
  The list of action groups that are allowed to be executed.

5.1.2 Installing and listing a Zagreus Licence

The first time the user connects to a newly installed Zagreus Server, the Zagreus Client shows a notification dialog box that a licence key needs to be installed (see *Figure 1.*). Zagreus is deliberately shipped with an expired licence key to trigger this message.

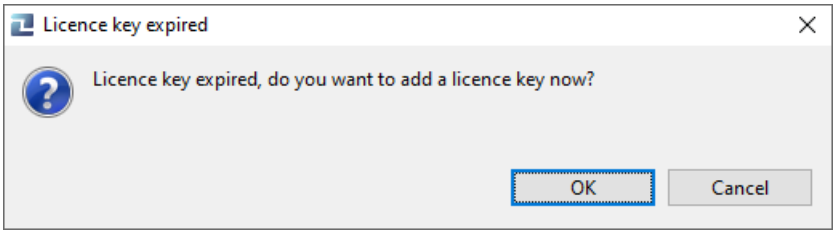


Figure 1 – The *Licence key expired* dialog box

After clicking on the *OK* button, the *Licence info dialog* box will appear and the new licence key can be inserted in the *Licence key* field, see *Figure 2*.

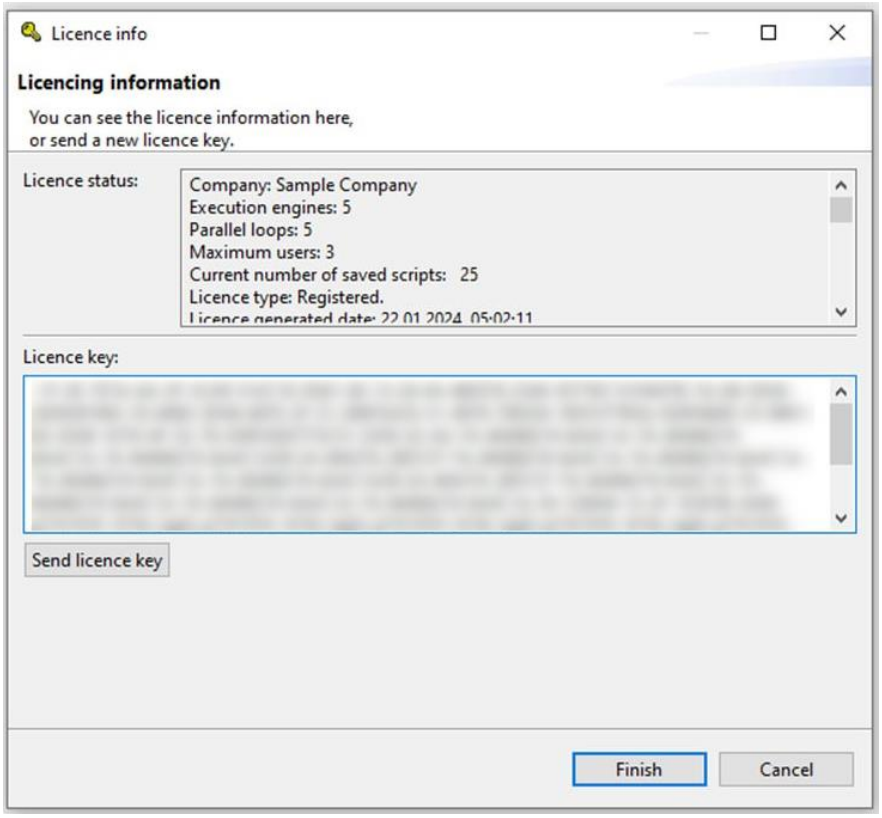


Figure 2 – The *Licence info* dialog box

The *Licence Info* dialog box serves two purposes:

- *Listing the contents of the currently installed licence*
  The *Licence status* textbox shows the details of the currently installed licence. When the customer needs to install a new licence key for the first time, the information in this textbox is not relevant. Otherwise, the important features of the licence are listed here, as listed in the previous chapter.

- *Providing a way to install a new licence key*
  The new licence key should be pasted into the *Licence key* textbox. After pressing the *Send licence key* button, a notification message reinforces the user that the licence key has been sent succesfully. The newly installed licence information is automaticcally refreshed in the *Licence status* textbox. The Zagreus Server does not need to be restarted.

Info: *The action groups enabled by the installed Zagreus Licence are listed in the* Licence status *textbox of the* Licence Info *dialog box in the Zagreus Client.*

5.2 Administrative scripts

The Zagreus System is shipped with pre-installed administrative scripts that help the administrator user to perform specific administrative tasks for system maintenance. These scripts are installed under *admin/administration* folder in the embedded database filesystem, see → Embedded MySQL database. The scripts use some external resources as well from the same folder.

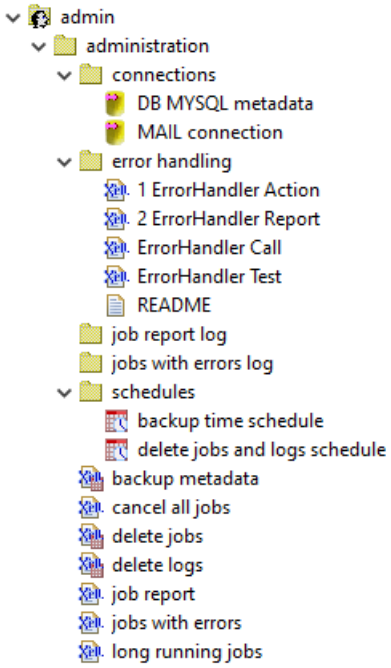


Figure 3 – The *administration* subfolder in the *admin* user home folder

5.2.1 Connections

There are two pre-installed connections in the *connections* subfolder:

- *DB MYSQL metadata*
  This is the database connection initialized to reach the local embedded MYSQL database, see → Embedded MySQL database. This is neccessary for the *delete jobs* script. When the *mysql.port* server configuration property is not set to the default 3336 value, this connection needs to be modified accordingly.

- *MAIL connection*
  This is a connection that is needed for sending error reports when necessary. This connection is not initialized (because the SMTP settings are installation dependent), so this must be configured properly!

5.2.2 Time schedules

There are two pre-installed time schedule resources in the *schedules* subfolder:

- *backup time schedule*
  This is the pre-installed time schedule for the *backup metadata* script, which is subscribed to this time schedule in the shipped Zagreus installation. The time schedule triggers the backup script every day at `3:00 AM`, but it can be modified to suit the needs of the actual system.

- *delete jobs and logs schedule*
  This is the pre-installed time schedule for the *delete jobs* and *delete logs* scripts, which are subscribed to this time schedule in the shipped Zagreus installation. The time schedule triggers the backup script every day at `1:00 AM`, but it can be modified to suit the needs of the actual system.

5.2.3 Scripts

The administrative scripts are located in the root of the *administration* folder:

- *backup metadata*
  This script performs a database backup of the embedded MYSQL database. The *DaysToKeepJobs* script variable defines the number of days to keep existing backups. The instructions are written in the first action of the script.

- *cancel all jobs*
  This script cancels all jobs that are in the queue (i.e. with statuses *running* or *queued*, see → Job lifecycle). The instructions are written in the first action of the script.

- *delete jobs*
  This script deletes the jobs from the embedded database metadata that are older than the value of the *DaysToKeepJobs* script variable (defined in days). The instructions are written in the first action of the script.

- *delete logs*
  This script deletes log files from different log folders that are older than the values defined in the appropriate script variables (defined in days). The detailed instructions are written in the first action of the script.

- *job report*
  This script generates a detailed job report and sends it in e-mail to a pre-defined address. A job report is a tabular data with special filters, similar to the *Active jobs* and *Finished jobs* windows in the Zagreus Client (see → Zagreus Client). The detailed instructions are written in the first action of the script.

- *jobs with errors*
  This script is a special case of the *job report* script: it filters only the jobs with status *error* and sends it in e-mail to a pre-defined address. The detailed instructions are written in the first action of the script.

- *long running jobs*
  This script is a special case of the *job report* script: it filters only jobs with status *finished* that have been running for a longer time than the *running_time* script variable (defined in seconds). The detailed instructions are written in the first action of the script.

5.2.4 Error handling

There are helper script 'snippets' in the *error handling* subfolder. These script behave as templates, their contents can be inserted into other administrative scripts. With the help of them, the administrator can further customize the existing administrative scripts, e.g. a specific error handler can easily be implemented, see → Error handling. The detailed instructions are written in the *README* text file in the same folder.

5.3 Concepts of user and group management

In the Zagreus System, a user is an entity (typically a person) that interacts with the Zagreus System via a client application to perform tasks, access resources and initialize script execution. The user *admin* is the default user in a Zagreus sytem.

A group is a shared folder, to which users can be assigned, therefore gaining access to shared resources. All users in the Zagreus System are assigned to the *public* group by default.

This organizational structure helps streamline the administration of user accounts and enhances security and efficiency.

5.3.1 Users in the Zagreus System

Zagreus provides the possibility for creating multiple users, which is necessary to build up a safe and efficient system. In Zagreus, there are two types of users: simple users and administrators.

The only existing user in a freshly installed Zagreus System is the user *admin*, which is a default user with administrator rights. This user has the default password *admin*, which is recommended to change as soon as possible, see → Password policy. The *admin* user can create new users to build up a multi-user system, giving each users only the necessary rights. Creating new users are described in details here → Create new user. Users can only be managed by an administrator user, except for changing their own password, see → Context menu of a user node.

Each user has its own dedicated home folder, in which it can store its own resources. The user home folder is located in the embedded database under */users/<user-name>* . The only exception is the home folder of the default *admin* user, whose path is */admin* .

Each user has its own dedicated filesystem root mapping as well, see → Mapping of the root folder. The entry point of this mapping is located in the home folder of the particular user.

Aside from the unique user name, each user is identified by a user ID as well. For the user *admin* it is 1, the ID-s of the newly created users start from 100.

There is the possibility to define variables on the user level, see → Declaration levels. It can be done in the Zagreus browser window in the Zagreus Client, see → Context menu of a user node.

5.3.1.1 User rights

Any user in a Zagreus sytem has the rights to:

- Browse all the resources located in its own user home folder.
- Browse all the resources located in the group home folders that the user is assigned to.
- Open, edit, copy, move, delete all these aforementioned resources.
- Execute the scripts it can access.
- Create and delete subscriptions for the scripts and schedules it can access.
- Manage the jobs it started and check out the logs of these jobs.

5.3.1.2 Administrator user rights

The administrator-type user (see → Administrator user rights) has the additional rights to:

- Access *all the resources* in the embedded database and in the local filesytem. The administrator user can open, copy, move, delete all of the resources of all other users.
- Execute any of the scripts in the Zagreus System.
- Create and delete subscriptions for any scripts and schedules.
- Manage all the jobs and check out the logs of any job.
- Manage all users and groups.
- Start and stop system components.
- Manage the SSL certificates of the Zagreus Server and Worker-Controller.
- Monitoring all watcher and trigger resources, see → Monitor watchers, triggers.

5.3.2 Groups in the Zagreus System

Zagreus users can be assigned to Zagreus groups. User group assignments can only be managed by an administrator user, see → Modify existing user.

Zagreus groups facilitate resource sharing and collaboration among members with similar responsibilities or tasks.

There is the possibility to define variables on the group level, see → Declaration levels. It can be done in the Zagreus browser window in the Zagreus Client, see → Context menu of a group node.

Each group has its own dedicated home folder, in which the shared resources are stored. The group home folder is located in the embedded database under */groups/<group-name>* .

Each group has its own dedicated filesystem root mapping as well, see → Mapping of the root folder. The entry point of this mapping is located in the home folder of the particular group.

5.3.3 Ownership of Zagreus resources

Each Zagreus resource has an ownership property (*owner*, see → Ownership of Zagreus resources), which governs the access to that particular resource. The ownership of a Zagreus resource depends on the folder where it is located. If the given resource is located in the home folder of a user, the resource is owned by that particular user; this means that only that user (and all the administrator users) is allowed to access it. If the given resource is located in the home folder of a group, the resource is owned by that particular group; this means that only users who are assigned to this group (and all the administrator users) is allowed to access it.

Info: *When the storing home folder is changed for a Zagreus resource (because e.g. the resource was moved), the ownership of the resource changes as well.*

Warning: *Ownership (*owner *property) and who created a resource (*created by *property) are two distinct properties!*

5.3.4 Password policy

The primary goal of a password policy is to enhance the security of user accounts by promoting the use of strong and secure passwords. In the Zagreus System, the password policy rules can be switched on for each particular user, see → Modify existing user. When the password policy is activated, the password must satisfy several conditions; the password must:

- be at least 10 characters long,
- contain at least one upper case letter,
- contain at least one digit,

- contain at least one special character (i.e. one of the following: `& @ # ' " + % / = ( ) . , ; : ? ! * < > - _` and space).

An active password policy also affects password expiration and reusability. Fine-tuning the properties of this feature can be set in the configuration of the Zagreus Server, see → Password policy properties.

- The user needs to upgrade its password after a certain time period (configurable). Otherwise the password will be expired and cannot be used for logging in.
- Whenever the user logs in via the Zagreus Client and the time of password expiration is approaching, a warning message is shown. The time period threshold of this warning is also configurable.
- The user cannot set a password used recently. The number of password changes after which the same password can be used again can be configured as well.

6. Resources

Resources are the basic blocks of related information. They are like files and folders in a filesystem. In Zagreus, resources are stored either in the embedded database or in the local filesystems (both on server or worker module side). Both are located on the server side. See → Resource storaging.

Automated workflows need a properly defined and configured underlying resource system which includes resources containing data, resources for executable content and also resources which are responsible for trigger mechanisms. See → Execution by event-type resources.

The easiest approach is to treat resources as files with specific role and behaviour in a complex automated workflow.

6.1 Resource types

There are several types of resources in the Zagreus System. Different types are used for different roles in the execution workflow. Resource types can be grouped in the following categories:

- *folder types:* They are mainly for organizing other resources.
- *types with general data storage:* Simple files with content that is unrelated with the inner mechanism of the Zagreus System. For example an uploaded pdf, xlsx or a binary file.
- *resources with executable content:* They store the content which the system can execute. Scripts, templates and the separated connection resource.
- *trigger types:* They are responsible for defining trigger mechanisms for script execution. They can be time-related or event-related.

Resources that are stored in the database can be all types of the following list, however resources in the local filesystems (server or worker) can only be a folder or a file (mapped from the operating system file system).

- *Folder*
  Folders are ’virtual locations’. They can help in storing and organizing other resources or folders. They have basically the same function as the operating system folders. All local filesystem folders are mapped to this type as well.

- *User home folder*
  This is a special folder type. It represents the home folder of a particular user. The path of the user home folder of the *’admin’* user is ’/admin’; for all other users it is *’/users/username’*.

- *Group home folder*
  This is a special folder type. It represents the home folder of a particular group. Its path is ’/groups/groupname’.

- *Filesystem root*
  This is also a special folder type. It represents the root of a particular user’s filesystem mapping. It is located in the root of the user home; the name is always

<-filesystem->. Its content is mapped to the user's server filesystem folder. See also → Mapping of the root folder

- *File*

  Files are used to store general-purpose data. They can contain binary / textual data. Also, all local filesystem (server or worker) files are mapped to this resource type, regardless of their purpose.

- *Script*

  Script is the resource type where the user can define what tasks should be executed. Scripts can be edited in the Graph Editor, and they contain the actions in a hierarchical structure. Script content can be viewed graphically or in an xml format. See also → Scripts.

  Script variables and execution options can be assigned to a script resource, see → Start-up variables and → Execution options.

- *Connection*

  Connections can be viewed in two ways:

  - a specific resource which stores connection parameters
  - a separate part of the script, containing the connection action for reusability

  Connections can also be edited in the Graph Editor, but they most have only one action, a connection type action (i.e. their name ending with 'connection').

  Some connections can be opened (like a folder) in the Zagreus Browser. See also → Connections and → Opening connections in the Zagreus browser

- *Template*

  Templates are separated, reusable parts of a script. They can be imported and executed from a script, with passing parameters as well. See also → Templates

- *Time schedule*

  Time schedule is an event-type resource, allowing automatic script execution (see also → Execution by event-type resources). It defines a Cron-type time-based trigger such as repeating time intervals, time ranges, specific times etc. It uses a Quartz Cron expression. See also → Time schedule

- *Event schedule*
  Event schedule is also am event-type resource. Events can be fired from multiple sources (manual, from a script, through a webservice call), and that is what is triggering it. See also → Event schedule

- *Mail watcher*
  A mail watcher is a special type of an event-type resource. It uses a mail connection that it is periodically checking and it triggers when its specified condition evaluates to true. See also → Mail watcher

- *Database watcher*
  A database watcher is another special type of an event-type resource. It uses a database connection that it is periodically checking and it triggers when its specified condition evaluates to true. See also → Database watcher

- *File trigger*
  A file trigger is checking a specified filesystem folder and triggers on specific events e.g. file creation or deletion. See also → File trigger

6.2 Resource properties

6.2.1 List of resource properties

All resources have specific properties in the Zagreus System. Resources stored in the database are slightly different from the ones are located in the local filesystem. Not all properties are accessible for the local filesystem resources. The following table shows the differences:

Property name	Database	Filesystem
Name	√	√
ID	√	√
Version	√	×
Type	√	√
Full path	√	√
Size	√	√
Owner name	√	√
Creation time	√	×
Modification time	√	√
Created by	√	×
Description	√	×

- *Name*
  The name of the resource. It cannot contain the character '/' and the maximum length is 100 characters.

- *ID*
  The identifier of the resource. For database resources it is a GUID (32 character long); for referencing resources, usually the version is concatenated at the end of the ID with delimiter '|&', see also → Embedded MySQL database.
  For server filesystem resources, the ID is generated from the operating system path in the form *fs/1/folder/filename.ext*, see also → ID and full path format
  For worker filesystem resources, the ID is also generated from the operating system path in the form *wfs/folder/filename.ext*, see also → ID and full path format
  Some special resources like the admin user home folder or the recycle bin folder have special, pre-defined IDs.

- *Version*
  The version of the database resource, for example '1.2.3.4'.
  Non-versionable resources like folders also have a version, but only the default version '1.0.0.0'. See also → Current version

- *Type*
  Resource type, see also → Resource types

- *Full path*
  The full path of the resource. It starts with a '/' and ends with the resource name. For filesystem resources, the path constructed of the filesystem root folder path (*'/users/username/<-filesystem->'*) and the mapped filesystem resource path (*'/fileSystemFolderName/localFileName'*).

- *Size*
  The size of the resource by bytes.

- *Owner name*
  The owner (user) name of the resource. See also → Ownership of Zagreus resources

- *Creation time*
  The timestamp when the resource was created.

- *Modification time*
  The timestamp when the resource was last modified.

- *Created by*
  The name of the user who has created the resource.

- *Description*
  The description of the resource, which is a short textual element attached to the resource. Maximal length is 255 characters.

6.2.2 Resource properties in the Zagreus Client

The resource properties can be checked in the Zagreus Client by right-clicking on the particular resource, and selecting *Resource information* from the context menu.

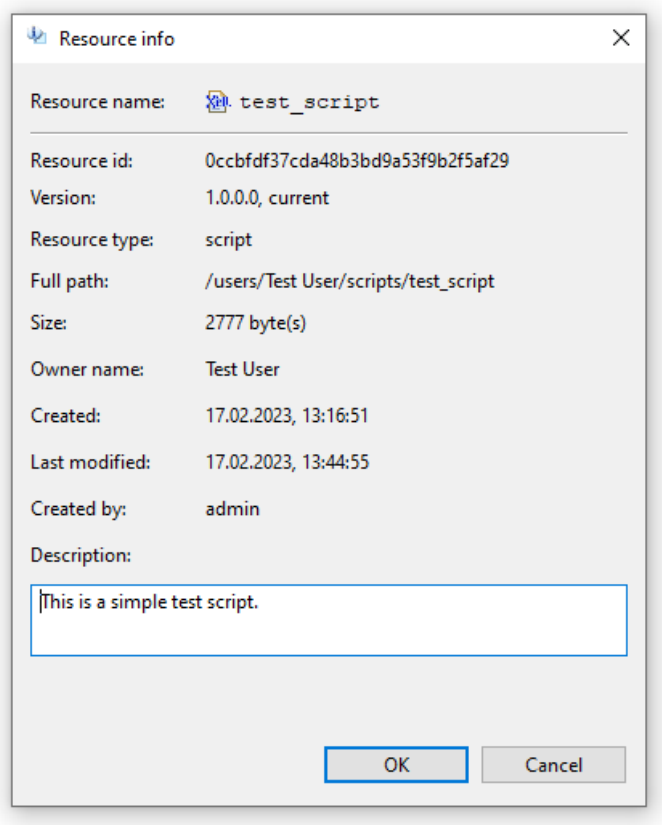


Figure 1 – Resource info dialog

Description property can only be set from this dialog for a resource.

> ***Info:*** *Not all properties are listed for all types. E.g. for folders the 'size' property is not shown, or for local filesystem file resources there is no version property.*

6.3 Resource versioning

Certain database resources can have multiple versions. There are several use cases when creating new versions of an existing resource is useful:

- *developing complex scripts*
  All new changes can be saved into a new (trial) version without losing any details of the already working script.

- *keeping track of the changes over time*
  Versions can be treated as revisions of the same resource.

- *multiple choices for tests:*
  Using multiple versions of the same connection resource (e.g. with different user credentials) helps in testing different use cases. Switching between test cases is easy by setting the current version of the resource.

> ***Info:*** *Only database resources can have versions, because local filesystem resources are mapped from the OS filesystem (which natively does not have built-in versioning).*

6.3.1 Version format

The format of the version is like '1.0.0.0', so it is a

`n.n.n.n`

format, where `n` is an unsigned integer number in the range of 0-99.

6.3.2 Current version

Current version is a special version among all the versions. There are built-in rules for the resource versioning and the current version:

- Each resource has an initial version (aka *default version*). This is always version `1.0.0.0`.

- Each resource has to have a *current version*. If there is only one (the default) version for a resource, that must also be the current version.
- Only one current version can exist for a resource. When there are multiple versions of a resource, all other versions are non-current.
- The current version can be freely selectable among versions. See also → Setting the current version
- The resource id contains a 32 character long GUID concatenated with the resource version by the delimiter `|&`
- A resource can be referenced by only its GUID and in this case it always refers to the *current version*

6.3.3 Resource *ID* and *version*

The resource ID and the version are stored together in the database in the following format:

```
323eab314a5749b68fd1830408d43619|&1.0.0.0
```

where the first part is the generated 32 character long GUID, the last part is the version, and they are concatenated by the special delimiter: `|&`. This fully qualified format is the truly unique identifier for a resource. When there are multiple versions created for a resource, their ID-s will have the same GUID part with different version parts like:

```
323eab314a5749b68fd1830408d43619|&1.0.0.0
323eab314a5749b68fd1830408d43619|&1.0.0.1
323eab314a5749b68fd1830408d43619|&1.0.0.2
```

This kind of ID structure assures that the versions of the same resource belong to each other (hence the same GUID) but differ from each other (by their version numbers).

> ***Info:*** *There is a useful simplification built-in to the Zagreus System: resource referencing can be done by only the GUID part of the whole ID, in this case it means the current version of the resource.*

6.3.4 Versioning in the Zagreus Client

6.3.4.1 Listing resource versions

Versioned resources are displayed in a parent-child structure in the Zagreus Client. If there is more than one version of a resource, the resource itself becomes a container node for all its version.

For example, in *Figure 2.* the script *test_script* has only the default version, but *test_script_with_versions* has multiple versions:

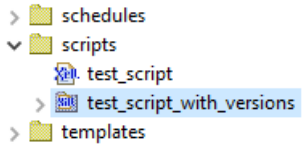


Figure 2 – Versioned resources are expandable tree nodes

When expanded (by clicking on the > sign), the versions are listed as child nodes, the version numbers are shown, and the current version is marked. The current version is always the first element of the list, all other versions are sorted by the version number.

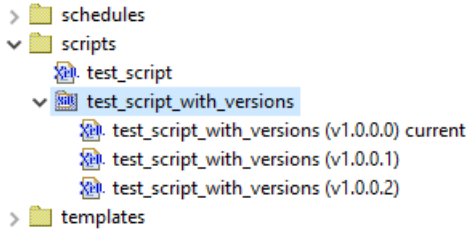


Figure 3 – All versions are listed as child nodes

6.3.4.2 Setting the current version

Setting the current version can be done by right-clicking on the particular (non-current) child node, and selecting *Set to current version*:

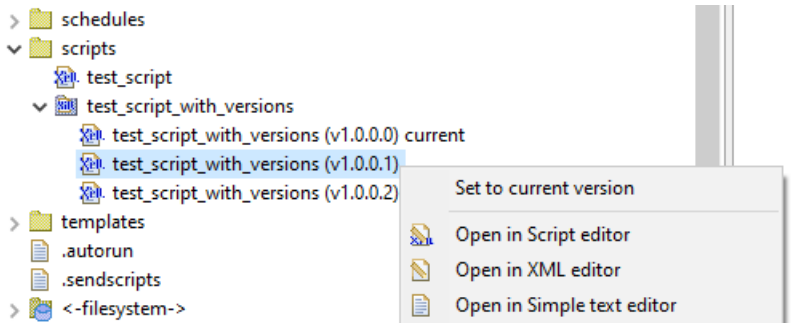


Figure 4 – Changing the current version

The child nodes are refreshed after this operation and the *current* mark is changed accordingly.

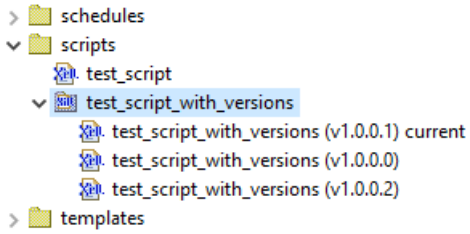


Figure 5 – The current version is 1.0.0.1

6.3.4.3 Creating a new version

Creating a new version can be done from the Script editor by clicking the icon tool (‘*Save a new version for the resource…*’) or by pressing the *CTRL+Shift+V* hotkey.

In the ‘*Set resource version*’ dialog box, the following parameters can be set:

- the new version number
- if the new version would be the current version
- a description for the resource, see also → Resource properties

A list of the currently saved versions are displayed at the bottom of the dialog box. The new version number must differ from any of the existing ones, and must satisfy the version format. See also → Version format

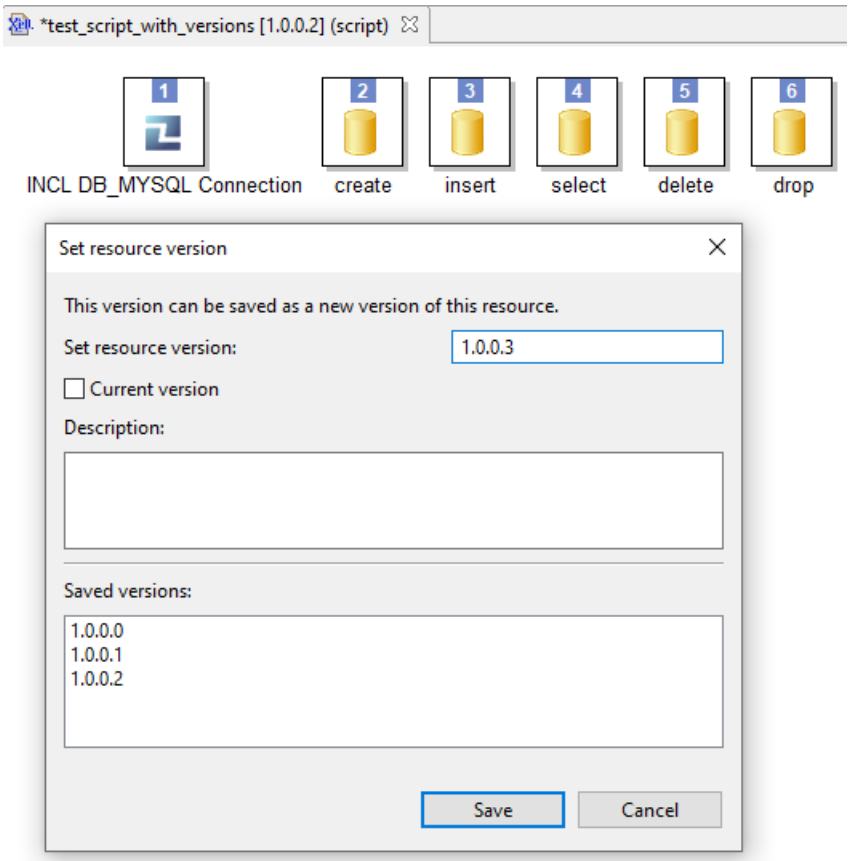


Figure 6 – Creating and saving a new version

6.3.4.4 Deleting a version

Deleting a version is only possible for a non-current version. The user needs to right-click on one of the versions of the resource and select the *Delete* menu item from the context menu, see *Figure 7.*

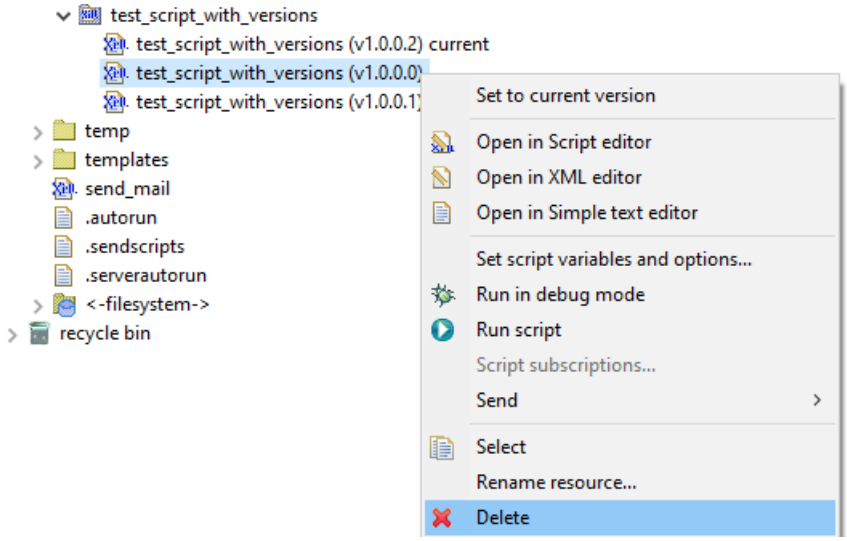


Figure 7 – Deleting a script version

6.4 Resource storaging

For flexible resource-handling, Zagreus supports three kinds of resource storage: database, server filesystem and worker filesystem (from Zagreus Version 1.5.6.0).

6.4.1 Embedded MySQL database

The database (as of Zagreus Version 1.5.6.0) is an embedded MySQL database which is managed by the Zagreus Server module. For database settings see → MySQL properties

The advantages of the database storage:

- Typed resources can be saved (e.g. scripts, connections, time schedules) for specific roles in the workflow.
- Standard filesystems do not support a number of resource-related Zagreus features: specific properties (e.g. description), variables and options for scripts, and versioning for resources.
- It is easier to include database resources for more complex workflows like creating a job when executing a script, filtering with properties when running a job report, etc.
- Using resource IDs intead of paths (that filesystems only have) opens more possibilites for synchronizing different systems (like test and production servers).
- Resources, triggers, jobs and automation are embedded into one system.

6.4.2 Local filesystem in the Zagreus Server

Each user and group has a filesystem root folder that does not overlap with filesystem folders of other users and groups. Symbolic links under Linux / shortcuts under Windows are supported.

The advantages of the local filesystem storage:

- Direct access for operating system files (easy access for externally generated file resources such as Excel, PDF, etc.).
- More efficient for handling large input/output files.

- Output files with an external source or target platform (.pdf, .xlsx, images) can be checked swifter from the local filesystem.
- Symbolic links or shortcuts for extension of the filesystem is easy.

6.4.2.1 Mapping of the root folder

When the user logs in by the Zagreus Client, the filesystem is visible in the home folder's root:

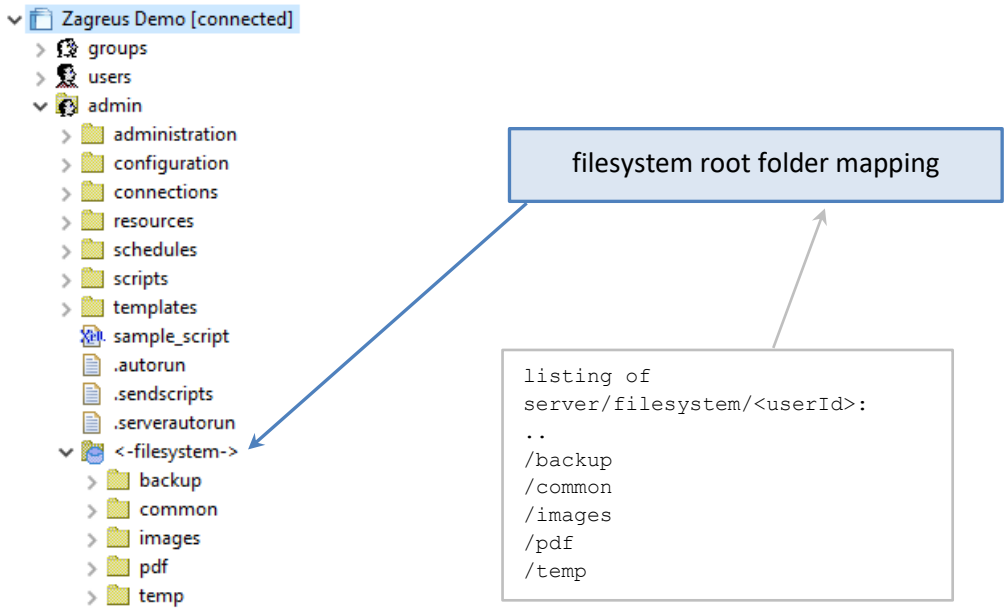


Figure 8 – Filesystem root folder is located in the user home folder

The OS filesystem is mounted under the *<-filesystem->* node, so the *<-filesystem->* folder node is the point where the local database folder structure and the OS filesystem are connected together.

The path of the OS folder which is mounted to this node is located in the server module under the path

```
/zagreus_home/server/filesystem/<userId>
```

The OS folder relative to the Zagreus Server module root folder can be configured in the configuration of the Zagreus Server by the `filesystem.root` setting, see → General properties.

The `<userId>` part is resolved by the unique user ID, see → Users in the Zagreus System.

6.4.2.2 ID and full path format

Since the external OS files and folders are mapped to Zagreus file and folder types (see → Resource types), the Zagreus System is generating IDs and full paths in its own format for these files and folders. Also, the OS files are identified by their OS path only, so a specific OS resource can be identified in multiple ways:

OS path:
`zagreus_home/server/filesystem/101/folder/file.txt`

Zagreus resource full path:
`/users/testuser/<-filesystem->/folder/file.txt`

Zagreus resource ID:
`fs/101/folder/file.txt`

The following terms explain the relation among the different identifiers:

- *Local OS filesystem root*
  The main mapping root point for all the user and group filesystems of the external operating system, see → Mapping of the root folder.

- ***User / group ID***
  The unique identifier of the particular user / group to which the filesystem belongs.

- *Local OS user / group filesystem root*
  The main mapping root point for the particular user or group filesystem. It is derivated from the local OS filesystem root and the **user / group ID** (the concatenation of the two).

- *Relative OS path*
  The path of the actual resource relative to the local OS user / group filesystem root.

- *Local database filesystem folder*
  The Zagreus full path to the filesystem mapping folder node, see → Mapping of the root folder.

- *'fs' prefix*
  In the case of filesystem resources, the Zagreus resource ID is generated from the relative OS path and the **user / group ID**. The 'fs' prefix indicates that it is an identifier. The ID is the concatenation of the three elements.

The Zagreus filesystem ID and full path can be used as general identifiers for resources in the scripts.

6.4.3 Local filesystem in the Zagreus Worker

From Zagreus Version 1.5.6.0, a further storage type is accessible: the worker filesystem. The worker filesystem is similar to the server filesystem, the main difference being that it is located on the worker module side. It does not appear in the Zagreus browser window, nor it is mapped under any browseable node.

The main advantages of using the worker filesystem:

- Large files (such as results of actions) can be stored immediately without transferring them to the server side storage systems. This results in faster and safer file I/O operations.
- This feature is neccessary for the Zagreus Standalone Worker (see → Standalone Worker) to be able to perform file operations.
- Worker filesystem is also essential for the External Script Execution feature (see → External script execution). Using locally stored files is inevitable for the external execution engines to read their inputs, and their result output files are also generated locally. They can be accessed only via the worker filesystem.

The Zagreus Execution Engine provides worker filesystem-related operations only via the `wfile` action group. In the actions of this action group, *worker filesystem paths* must be used, which are relative to the worker filesystem root folder, see below.

6.4.3.1 Mapping of the root folder

The location of the worker filesystem root folder can be set in the configuration of the Zagreus Worker (see → Zagreus Worker configuration): the `worker.filesystem.path` property specifies the file system root folder name relative to the root folder of the Worker module. If this property is not specified, the folder name *filesystem* will be used as default. Thus, in this case, the worker filesystem is under:

```
/zagreus_home/worker-controller/worker/filesystem
```

In this case, the worker filesystem path */folder/file.txt* will be translated to:

```
/zagreus_home/worker-controller/worker/filesystem/folder/file.txt
```

Similarly to the server filesystem, symbolic links or shortcuts can be used.

6.4.3.2 ID and full path format

Similarly to the server filesystem resource identifiers, worker filesystem resources also have Zagreus IDs and full paths. These can only be used in the dedicated `wfile` action group.

The Zagreus System is generating IDs and full paths in its own format for these files and folders. Also, the OS files are identified by their OS path only, so a specific OS resource can be identified in multiple ways:

- *OS path:*
  `zagreus_home/worker-controller/worker/filesystem/folder/file.txt`
- *Zagreus resource full path in the worker filesystem:*
  `/folder/file.txt`
- *Zagreus resource ID in the worker filesystem:*
  `wfs/folder/file.txt`

The following terms explain the relation among the different identifiers:

- *OS worker filesystem root*
  The main mapping root point for the worker filesystem of the external operating system, see → Mapping of the root folder.
- *Relative OS path*
  The path of the actual resource relative to the OS worker filesystem root. This is also the Zagreus worker filesystem resource full path.

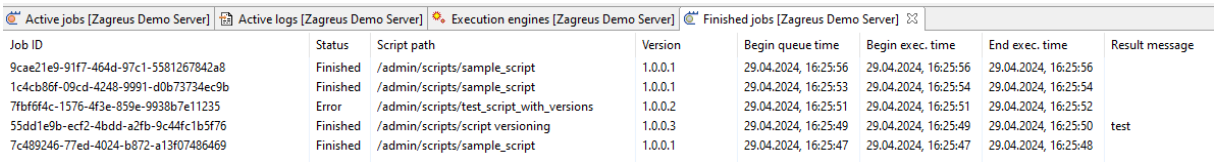
- *'wfs' prefix*
  In the case of worker filesystem resources, the resource ID is generated from the relative OS path. The 'wfs' prefix indicates that it is a worker filesystem identifier. The ID is the concatenation of the two elements.

7. Queuing and jobs

After initiating the execution of a script (see → Initiating script execution), Zagreus generates a *job*. A job represents a task to be executed.

The system submits this task into a *job queue* (basically a FIFO data structure: *first-in first-out*, so the first job which is put into the queue is the first one that is taken from the queue. this can be overridden by *priority* execution option, see → Execution options). Once a Zagreus Worker successfully takes a job from the queue, the execution of the job can finally be started by the Execution Engine of the Worker, see → Zagreus Worker.

It is important to see the difference between the script and the job. The term *script* can refer to the resource type as well as the resource content. On the other hand, the term *job* means the task for executing a particular script. For example, if a script is executed three times in a row, that results in generating three different jobs. Because of the fact that the execution of the script depends on the actual state of the environment (for example the content of a mailbox folder, or the actual Zagreus Worker occupancy), the results of the jobs can be very different from each other.


Active jobs [Zagreus Demo Server] | Active logs [Zagreus Demo Server] | Execution engines [Zagreus Demo Server] | Finished jobs [Zagreus Demo Server]

Job ID	Status	Script path	Version	Begin queue time	Begin exec. time	End exec. time	Result message
9cae21e9-91f7-464d-97c1-5581267842a8	Finished	/admin/scripts/sample_script	1.0.0.1	29.04.2024, 16:25:56	29.04.2024, 16:25:56	29.04.2024, 16:25:56	
1c4cb86f-09cd-4248-9991-d0b73734ec9b	Finished	/admin/scripts/sample_script	1.0.0.1	29.04.2024, 16:25:53	29.04.2024, 16:25:54	29.04.2024, 16:25:54	
7fbf6f4c-1576-4f3e-859e-9938b7e11235	Error	/admin/scripts/test_script_with_versions	1.0.0.2	29.04.2024, 16:25:51	29.04.2024, 16:25:51	29.04.2024, 16:25:52	
55dd1e9b-ecf2-4bdd-a2fb-9c44fc1b5f76	Finished	/admin/scripts/script versioning	1.0.0.3	29.04.2024, 16:25:49	29.04.2024, 16:25:49	29.04.2024, 16:25:50	test
7c489246-77ed-4024-b872-a13f07486469	Finished	/admin/scripts/sample_script	1.0.0.1	29.04.2024, 16:25:47	29.04.2024, 16:25:47	29.04.2024, 16:25:48	

Figure 1 – Finished jobs in the *Finished jobs* window in the Zagreus Client

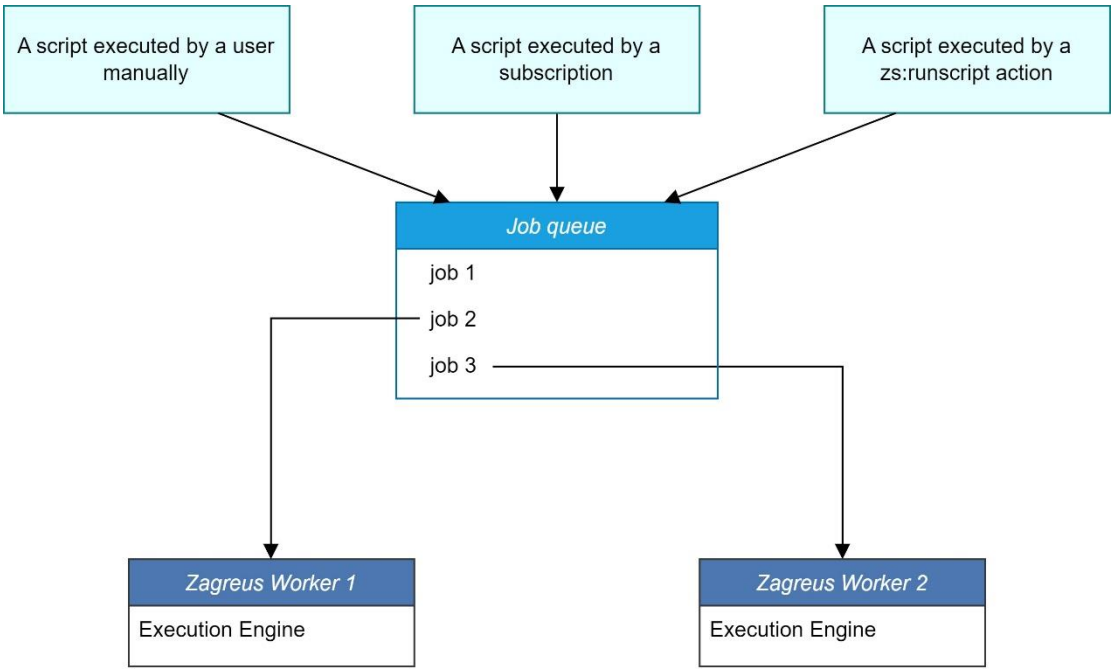


Figure 2 – The relation of the scripts, the queue and the Zagreus Workers

7.1 Job properties

The job can be considered as a data structure with properties. This properies are stored in the local database, the administrator can even check them using the local database connection (see → Connections)

- *job id*
  A unique identifier of the job. The format follows the GUID standard.

- *script id* and *script path*
  The ID and path properties of the script whose execution was initiated.

- *script content*
  The content of the script which execution was initiated. It is saved for the job, since the content of the script (i.e. the Zagreus resource) might change after job execution.

- *script start-up variables* and *execution options*
  The starting variables and options which were used for the script execution. There are many ways to specify these, such as the variables saved for the script itself in the Zagreus local database, or the ones defined for subscriptions, as well `zs:runscript` action parameters (see → Start-up variables and → Execution options).

- *begin queue time* and *end queue time*
  At the moment when the job is created, it is automatically put into the queue. This point of time is the *begin queue time* of the job. When the job is taken by a Zagreus Worker to be executed, it is no longer in the queuing state (see job statuses below). That point in time is the *end queue time* of the job.

- *begin execution time* and *end execution time*
  The point in time when the execution of the job started by the Zagreus Worker is the *begin execution time.* When the execution is finished, the *end execution time* is set.

- *job status*
  The status shows the current state of the job, see → Job lifecycle. It starts with status *queued* then changes accordingly to the execution stages of the job.

- *worker id and worker-controller id*
  When a Zagreus Worker gets the job to execute, its *worker id* is stored to the job. In default configuration, there is only one Worker-Controller set up to the system, which has *worker-controller id* = 1.

- *user id*
  The id of user who manually executed the script or whose subscription initiated the execution, see → Users in the Zagreus System

- *parent job id*
  If the execution was initiated from another script via a `zs:runscript` action, the id of that job is the *parent job id.*

- *caller* and *caller type*
  caller type describes the type of how the execution was initiated, see → Caller and caller type. The property *caller* stores additional information about the initiation.
  For more details on these two properties, see the following sub-chapter.

- *execution mode*
  describes the execution mode of the job. The possible values are: *direct, scheduled, fired* and *triggered.*

- *priority*
  It is an integer value (from 1 to 1000). The lower the value the higher the job priority is.

- *result and result-message of the execution*
  When the job successfully finished (the job status is *finished*), the result and result-message of the executed script is stored. See → Result flow

7.1.1 Caller and caller type

The *caller-type* job property defines the type of how the execution was initiated, see → Initiating script execution. Its value can be one of the following list elements:

- *scheduler*: initiated by a time schedule trigger
- *gui*: initiated by a user manually from the Zagreus Client
- *webservice*: execution initiated from the html pages or other external source, see → Execution from external systems
- *script*: initiated from a `zs:runscript` action using local zs connection, see → Execution from a Zagreus script
- *remote script*: initiated from a `zs:runscript` action using remote zs connection, see → Execution from a Zagreus script
- *autorun*: execution initiated by the autorun funcionality of Zagreus, see → Script execution by autorun configuration files
- *event*: the execution was initiated by an event schedule, see → Event schedule
- *file trigger*: a certain file event triggered a file trigger (see → File trigger) which initiated the execution of this script
- *db watcher*: execution was initiated by a database watcher whose condition was evaluated to true, see → Database watcher
- *mail watcher*: execution was initiated by a mail watcher whose condition was evaluated to true, see → Mail watcher

The *caller* job property provides additional information about the execution. Depending on the caller type, the value of the caller property can be:

Caller type	Caller property value
scheduler	the time schedule id : subscription id
gui	the client ip
webservice	custom text, in case of a html call, it is '*html*'
script	the caller script id
remote script	caller script id @ ip address
autorun	the constant '*ServerAutorun*'
event	event id : subscription id
file trigger	file trigger id : subscription id
db watcher	database watcher id : subscription id
mail watcher	mail watcher id : subscription id

The caller and the caller type properties can be checked out and seen both the in the *Job Properties* dialog in the Zagreus Monitor, see → Job properties dialog and in the Finished jobs window in the Zagreus Client, see → Finished jobs window.

7.2 Job lifecycle

Each job has a lifecycle from the creation of the job till the job ends. When the job is created, basic properties are filled like *job id, script id, script path, script content, user id, caller type, execution mode, priority* and *parent job id* (if there is any).

- The job lifecycle starts with the status '*queued*'. It means that the job is put into the Zagreus *job queue*, waiting for taken out by a Zagreus Worker for execution. The job property *begin queue time* is set at this point.

- If the job is waiting in the queue for too long (the time exceeds the queue timeout set for the script, see → General properties), the job status is set to '*queue timeout*'. The job is removed from the queue and it will not be executed. The job property *end queue time* is set at this point.

- When the job is successfully taken from the queue by a Zagreus Worker, its status is set to '*starting*' for a very short period of time, then it is set to '*running*'. The execution of the job is starting now. The job properties *end queue time* and *begin execution time* are set at this point.

- If the execution finished successfully, the job status is set to *'finished'*. The *result*, *result message* of the executed script (see also → Order of execution, result flow) is stored in the local database along with the property *end execution time*.

- If the execution does not finish successfully (an error has been thrown, or the *status* attribute of the `z:exit` action was set to *'error'*, see → z:exit and z:return actions), the job status is set to *'error'*. The *result message* property of the executed script is set only if it was specified by the `z:exit` action. The job property *end execution time* is set as well.

- If the job is running for too long (the time exceeds the running timeout set for the script, see → General properties), the job status is set to '*running timeout*'. The system interrupts the execution of the job, and the job property *end queue time* is set.

- A job can be cancelled in several ways, such as manual cancellation from the Zagreus Client, server shutdown automatic cancellation or the `zs:cancel` action

(see → Server startup and shutdown properties and → Cancellation by the zs:cancel action). The job status is set to '*cancelled*'. The system interrupts the execution of the job, and the job property *end queue time* is set.

There are two further, special statuses: *suspended* and *debugging*. These are available only if the script has been run in debug mode, see → Starting a debug session .

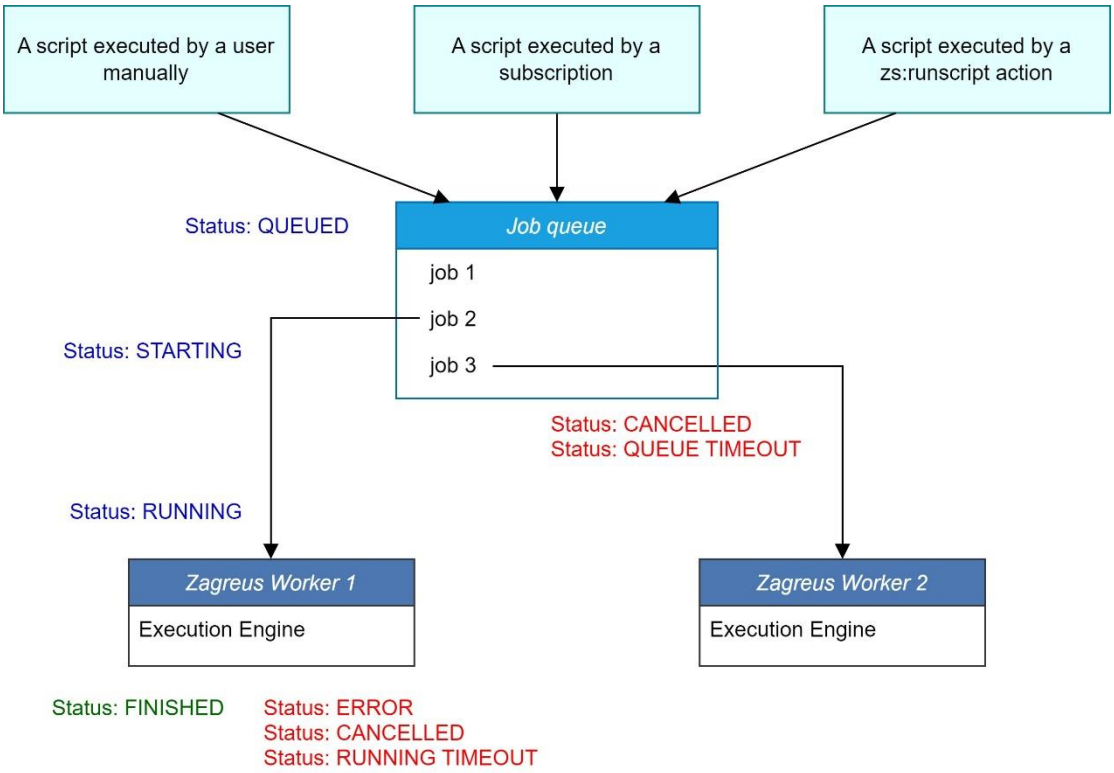


Figure 3 – The job statuses in the same structure as in Figure 2

7.3 Queue

The *queue* (or *job queue*) is a data structure managed by the Zagreus Server. It is basically a priority queue (see → Priority and priority algorithm) stored in the memory and in the local database as well. The Zagreus Workers (with their Execution Engines, see → Zagreus Worker) are connected to the queue, and are constantly trying to get a new job to execute. When a Zagreus Worker finally obtains a job, it starts to process it and the status of the Worker is turning to *busy*. The particular job is removed from the queue.

The job queue can be stopped from the Zagreus Client (see → Stop / start server components) and from the Zagreus Monitor (see → Additional options) applications. When the queue is stopped, it can not receive any new jobs and the currently queued jobs will not be passed to any Zagreus Workers.

7.3.1 Queue groups

All the Zagreus Workers are connected to the aformentioned job queue. Therefore, each given job can be assigned to any of the Workers. However, sometimes there is a need to override this default behavior of the dispatcher mechanism.

For example, there may be a need for a specific Worker to be configured using a different library path than the others (such as libraries of different MicroStrategy versions). Defining queue groups can be a solution for this scenario.

A queue group is a set of Zagreus Workers. By default, there is only one queue group in the system (the *default* group), and all Workers belong to this group. However, the user can define custom queue groups. This means that the Zagreus Workers will be split into groups.

Custom queue groups can be defined in the Zagreus Server configuration, see → Zagreus Server configuration.

Example: Suppose the Zagreus System allows to use up to five Zagreus Workers. Defining two queue groups would look like this in the configuration file:

```
queue.groups.num=2
queue.groups.1.worker=1.1,1.2,1.3
queue.groups.2.worker=1.4,1.5
```

This means that two different queue groups are defined. *Group 1* is associated with *Worker 1.1, Worker 1.2* and *Worker 1.3* while *Group 2* is associated with *Worker 1.4* and *Worker 1.5.* For fully qualified Worker numbers, see → ID of the Zagreus Worker .

Overlapping between two queue groups is not allowed, i.e. one worker can only be assigned to one worker group. All the workers need to be assigned to a group.

When there are two or more groups, the jobs can still be executed by any of the Workers until an additional setting is specified: assigning a script to a specific queue group. Since any group is associated with specific Workers, the script can only be executed by those Workers.

The execution option *queue_group_id* needs to be set in order to assign a script to a queue group, see → List of execution options.

In the example above, if the user wants a script to be assigned to Worker 1.4 or Worker 1.5, the execution option *queue_group_id* needs to be set to *2*.

7.3.2 Queue-level variables

Variables can be defined on the queue level. They must be declared in the configuration of the Zagreus Server. Variable resolution and their precedences are described in details, see → Precedence order for resolution.

Example:

```
# This is a queue-level variable (resolved as 'x' in the script)
variable.queue.x=1
```

7.3.3 Priority and priority algorithm

The job queue is a priority queue. It is very similar to a standard *FIFO* (first in – first out) queue but with an added feature: each element has a priority associated with it. Elements are dequeued based on their priority rather than just their order in the queue.

The priority of the job can be set by the *priority* execution option (see → List of execution options) which is a numeric value, set to the value *10* by default. If the default value of this option is not overriden, the queue will behave accordingly to the FIFO behaviour: the first queued job will be achievable to a Zagreus Worker first, then the second one and so on. But if there are jobs in the queue with different priority values, jobs with lower priority values will be taken out from the queue earlier (so the

queue is ordered by the priorities). This way, more important jobs can be put to the beginning of the queue.

> ⚠ ***Warning:*** *The word* priority *can be misleading, as* high-priority *jobs have* low *prioritiy values and vice versa.*

There is a case, though, which should be avoided. Assume that there are many jobs in the queue with different priorities, and high-priority jobs (i.e. those with low *priority* values) are regularly put into the queue. Because the number of Zagreus Workers is limited, these high-priority jobs will always occupy the Workers, so the low-priority jobs would be waiting for a very long time. To prevent this case, Zagreus offers a built-in feature called *priority algorithm*. The priority algorithm works in the following way: each new job passed to a Zagreus Worker automatically decreases the priority value of all the other, queued jobs. Therefore, over time, the relative importance of these jobs will be slowly increased against the newly queued jobs.

This algorithm can be switched on or off from the Zagreus Client (see → Stop / start server components) and from the Zagreus Monitor (see → Additional options) applications.

7.4 Hidden jobs

There may be cases when specific jobs are scheduled to run quite often, and this makes monitoring the other jobs in the Zagreus Client and in the Zagreus Monitor challenging. Zagreus offers two solutions for hiding these jobs from monitoring.

7.4.1 The *job_monitoring* execution option

Hidden jobs can be defined by setting the `job_monitoring` execution option. By default, it is *true*. When it is set to *false*, the executed job is only visible if the client applications explicitly ask for it.

For the Zagreus Client application, this explicit option can be set by the *Show hidden jobs* checkbox in the *Finished job report parameters* dialog box, see *Figure 4.*

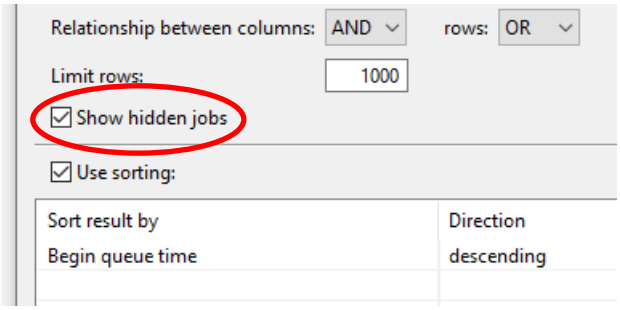


Figure 4 – The *Show hidden jobs* setting in the Zagreus Client

For the Zagreus Monitor application, this explicit option can be controlled by the *Show hidden jobs* checkbox on the main toolbar of the connected server pane:

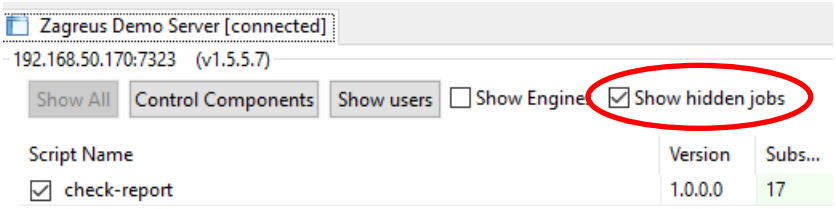


Figure 5 – The *Show hidden jobs* setting in the Zagreus Monitor

Execution options can be set in several ways in Zagreus, see → Declaration levels .

7.4.2 The *invisible* result message

The Zagreus Client also provides an application-level solution for the aforementioned problem. By default, the *Finished jobs* window also filters out jobs which have a specific result message: *invisible*. A predefined condition is set for the

Result message column in the *Finished job report parameters* dialog, as shown in *Figure 6*. Since it is an application-level solution, this filter can be changed (or solved by another filter) to match special user requirements.

This filter works only if the particular script has a result-message set to '*invisible'*, see → Result flow for understanding result-message and → z:exit and z:return actions for setting the result-message in `z:exit` action.

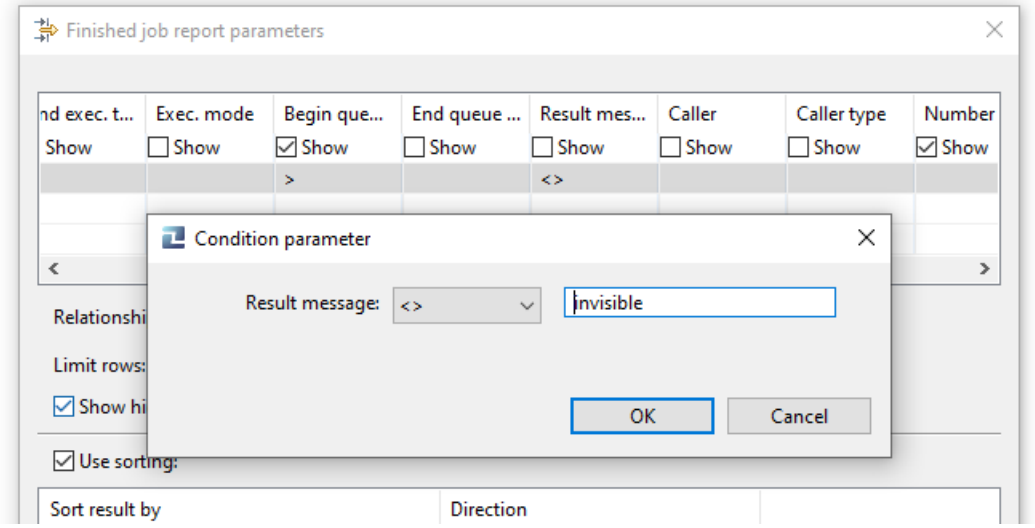


Figure 6 – The condition for the *Result message* column in the *Finished job report parameters* dialog

Figure 7. shows an example for using the `z:exit` action to set the result-message of the script to '*invisible'.*

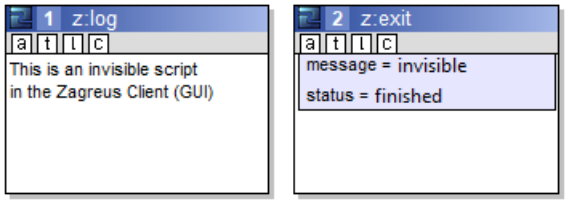


Figure 7 – The result-message *invisible* set by the `z:exit` action in a script

7.5 Skipped jobs

Skipped jobs is a concept in the Zagreus System: these are scheduled jobs which were not executed, mostly due to an unexpected server shutdown. The user can check these non-executed (i.e. skipped) jobs by the Zagreus Client and the Zagreus Monitor applications.

Since the Zagreus Server and its database were down, there are no possible database entries or log messages about which jobs were skipped. Because of this, the list of skipped jobs can only be estimated later on, when the Server has been restarted. The system examines all the calculated points in time according to script subscriptions and their underlying time schedule definitions, and tries to figure out if a job associated with the corresponding script has been queued at the given moment of time.

7.5.1 Skipped jobs in the Zagreus Monitor

In the Zagreus Monitor application, skipped jobs are automatically displayed on the Timeline (depending on the state of the *skipped* checkbox in the Status filter, see → Status filter). As it is seen in *Figure 8.*, skipped jobs are represented by red triangles.

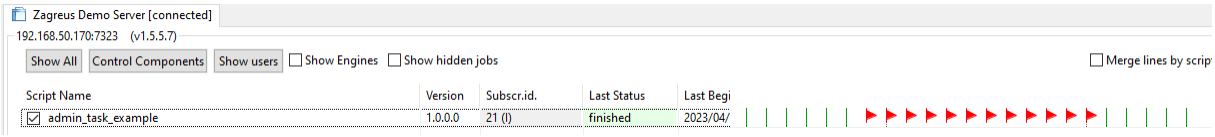


Figure 8 – Skipped jobs

By clicking on the red triangle icon, the *Skipped job* dialog opens, see *Figure 9.* All the fields which make sense are filled the same way as for the *Job properties* dialog (see → Job properties dialog). Note that the *Begin Queue Time* field shows the estimated scheduled time, indicating that the job has never actually run.

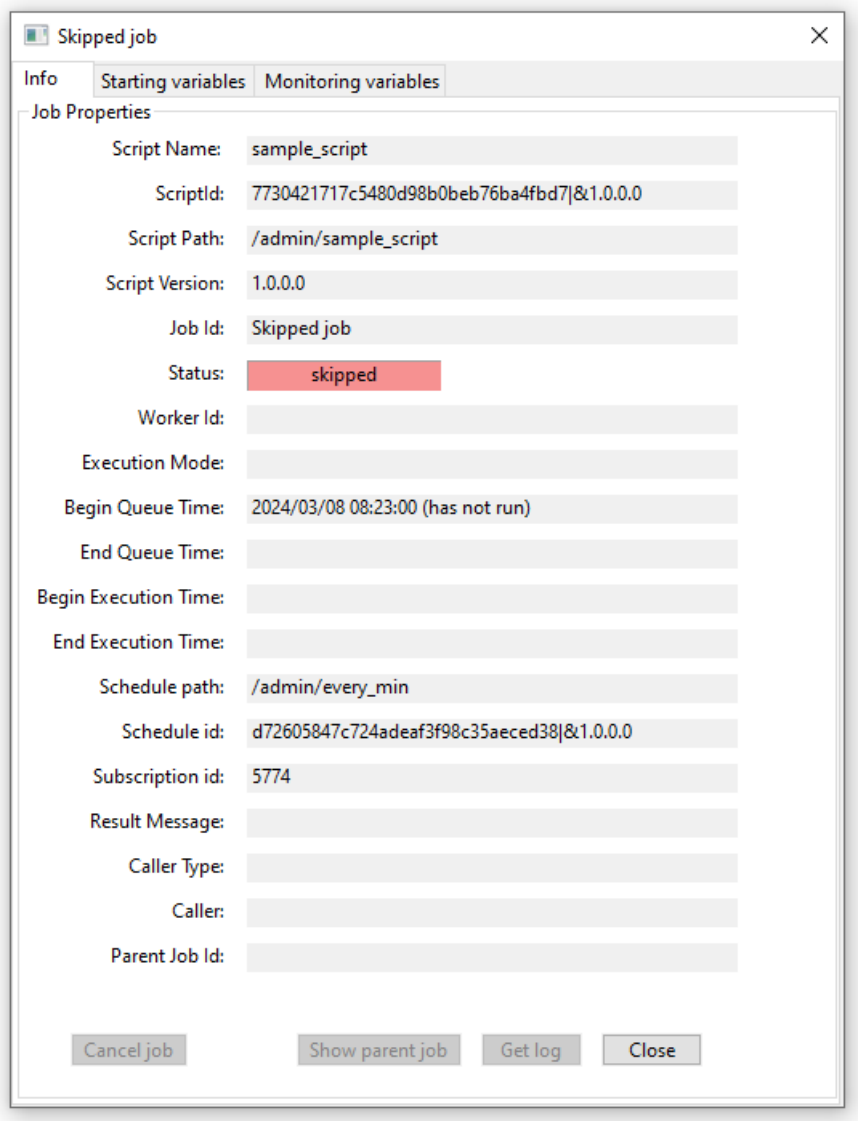


Figure 9 – The *Skipped job* dialog box in the Zagreus Monitor application

7.5.2 Skipped jobs in the Zagreus Client

In the Zagreus Client, a dedicated window can show the skipped jobs for a specified time interval. By default, this window is not open, the user needs to select the *Skipped jobs window* menu item from the main *Window* menu. The appearing *Skipped jobs* window (see *Figure 10.*) shows the list of the skipped jobs from the last 24 hours.

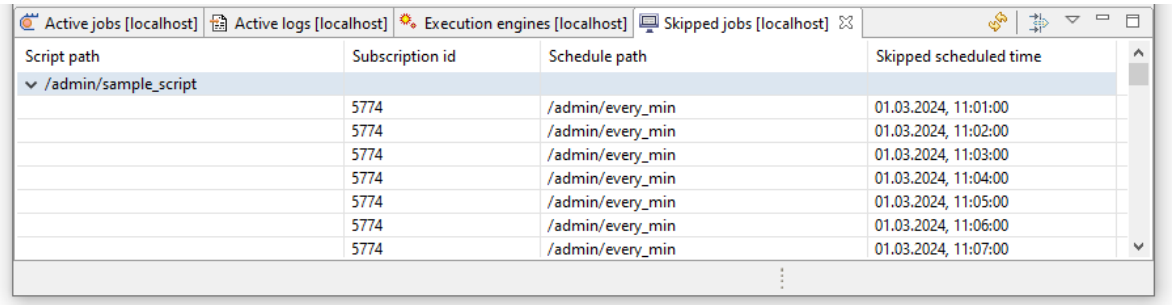


Figure 10 – Cancelling a running job by right-clicking on the job in *Active jobs* window

The skipped jobs are shown in a tree-table, where the skipped jobs belonging to the same script are grouped together. The script nodes can be expanded or collapsed. Each row then contains the following information in its columns:

- *Script path*: the path of the script that is associated to the skipped job
- *Subscription id*: the ID of the subscription that should have initiated the execution of the script
- *Schedule path*: the path of the time schedule that is associated to the subscription
- *Skipped scheduled time:* the time when the job should have been created

The default grouping and the time interval of the skipped jobs can be modified in the *Skipped jobs window preferences* dialog (see *Figure 11.*), which can be open by clicking on the  icon.

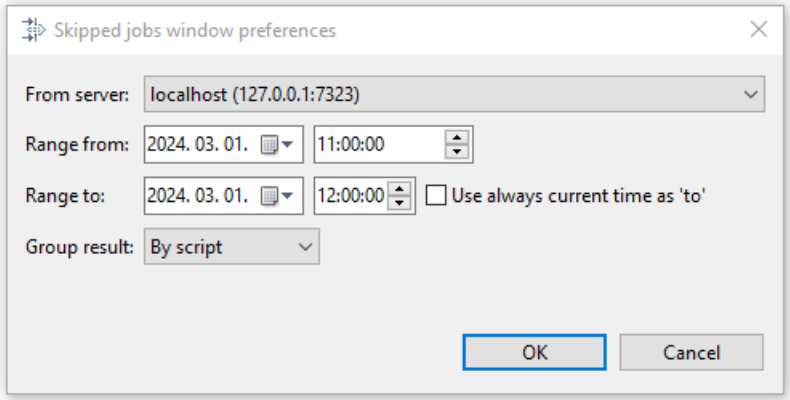


Figure 11 – Cancelling a running job by right-clicking on the job in *Active jobs* window

In the *Skipped jobs window preferences* dialog, the user can set the following properties for the skipped job estimation:

- *From server* drop-down list
  The user can select the Zagreus Server connection from the list of opened connections.

- *Range from* time selectors
  The user can set the start time for the skipped job estimation. By clicking on the  icon, the user can select the start date from the appearing date picker widget.

- *Range to* time selectors
  The user can set the end time for the skipped job estimation. By clicking on the  icon, the user can select the end date from the appearing date picker widget.

- *Use always current time as 'to'* checkbox
  If set, the skipped job estimation will always take the current time as the end date, ignoring the value in the *Range to* time selector.

- *Group result* drop-down list
  The user can select the field on which the skipped jobs will be grouped together. Possible values are: *By script*, *By subscription* and *By schedule*.

7.5.3 Setting the tolerance

There is a certain tolerance used for the calculation of the skipped job list, set by the `queue.skippedjobs.threshold` property in the configuration of the Zagreus Server, see → Miscellaneous properties. The default value is 5000 (in milliseconds).

This tolerance value is used when the system is checking if any jobs has been created at a given calculated point in time according to the subscriptions and time schedules.

Example:

A script called `script_1` was subscribed and scheduled to be executed at every hour on weekdays, and the server was down for two hours on a Friday from 10:55 to 13:10. Assuming that no further issues were experienced with Zagreus Server on this day, there are three skipped jobs for `script_1` (at 11:00, at 12:00 and at 13:00).

After the Zagreus Server has been restarted, and the client applications are connected again, there are ways to check if there was any skipped jobs during the shutdown period (see → Skipped jobs). Possible skipped jobs are always calculated in a user-defined time range. So, if the user likes to see the skipped jobs from e.g. on Friday, the system calculates points in time when `script_1` should have been queued: *00:00, 01:00, 02:00, … 11:00, 12:00, 13:00*, etc. To find out, the mechanism checks if there were any corresponding scheduled jobs queued around these times.

Because of the fact that queuing is not happening immediately, the checkout for finding these jobs needs to use a tolerance: if the tolerance is set to 5000 milliseconds, it means that a job corresponding to `script_1` (started by the corresponding subscription) queued at 09:00:04.100 is accepted as one started at 09:00, since the time difference is only 4100 milliseconds.

Example:

```
# This setting increases the tolerance to 10 seconds
queue.skippedjobs.threshold=10000
```

7.6 Cancellation

Job cancellation can be initiated by the user manually or the automatically by the Zagreus System. Cancellation is especially useful when:

- the job is running for an unreasonable long period of time
- the job execution is hanging and not responsive
- the job has been started accidentally
- the memory consumption of a Zagreus Worker exceeds the configured limit

7.6.1 Manual cancellation

The user can cancel a single job manually in several ways:

- in the Zagreus Client, via right-clicking on the job in the *Active jobs* window:

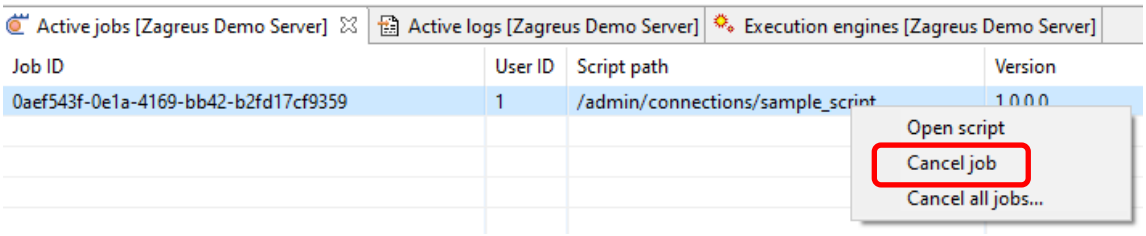


Figure 12 – Cancelling a running job by right-clicking on the job in *Active jobs* window

- in the Zagreus Client, via right-clicking the particular Worker row in the *Execution engines* window, on Worker Information tab:

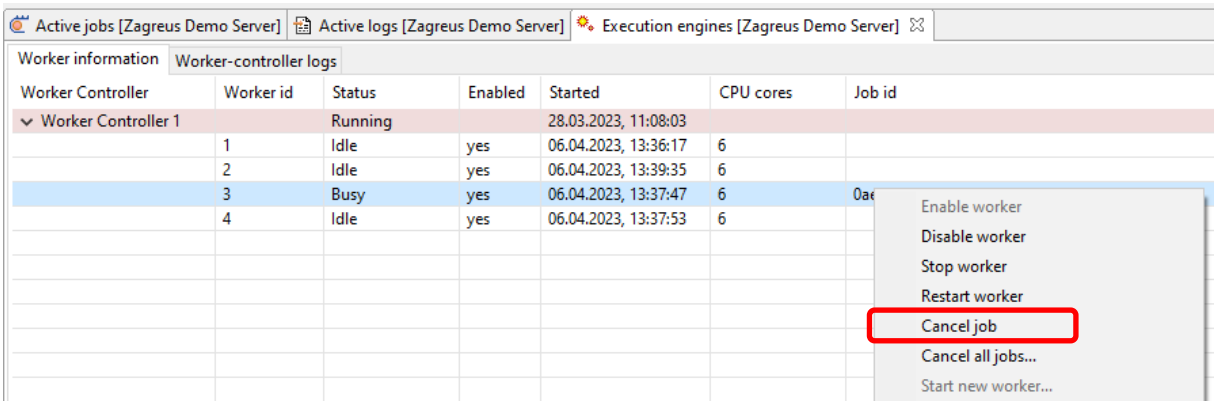


Figure 13 – Cancelling a running job by right-clicking on the job in *Execution engines* window

- in the Zagreus Monitor, clicking on the *Cancel job* button in the *Job properties* dialog box, see → Job properties dialog:

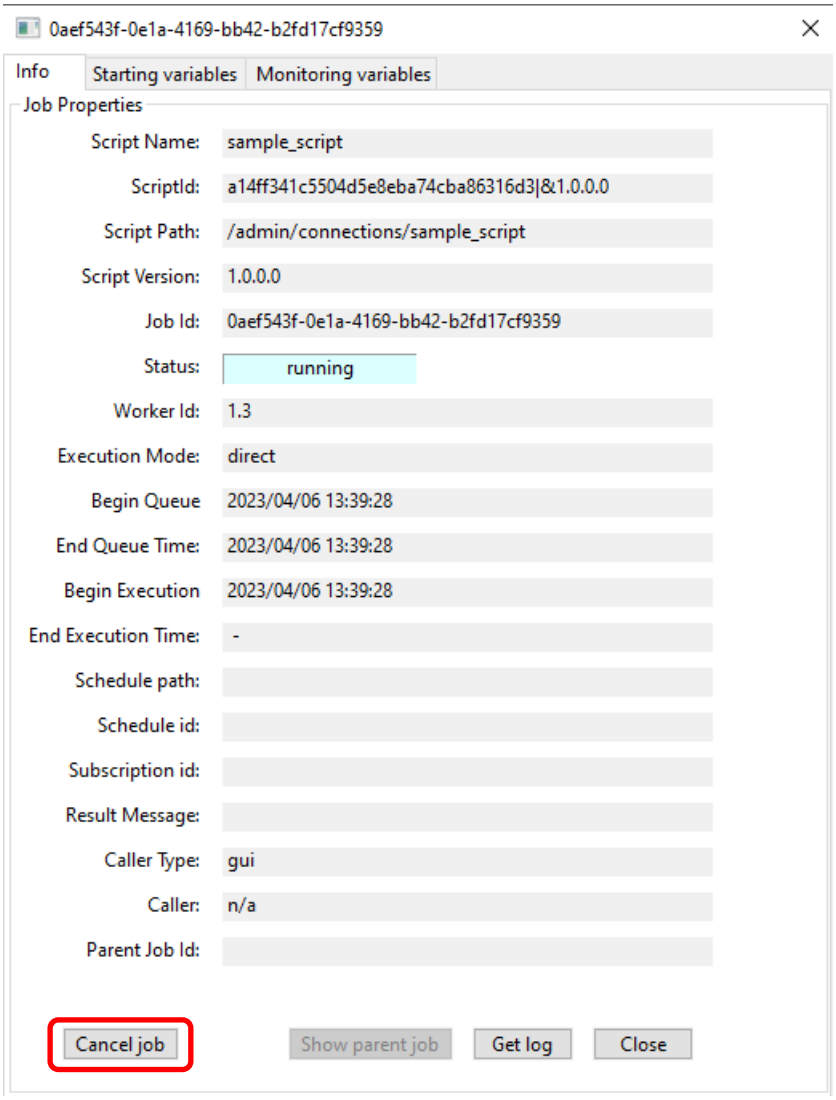


Figure 14 – Cancelling a job with the *Cancel job* button in Zagreus Monitor

7.6.2 Multiple cancellation by job statuses

In the Zagreus Client, there is a way to cancel multiple jobs at once. The *Cancel jobs* dialog box (see *Figure 15.*) can be opened by the following menu items:

- right-clicking the job in *Active jobs* window (see *Figure 12.*) and select *Cancel all jobs…*
- right-clicking the worker in *Execution engines* window (see *Figure 13.*) and select *Cancel all jobs…*
- right-clicking the server definition node and select *Administration options / Cancel all jobs…*

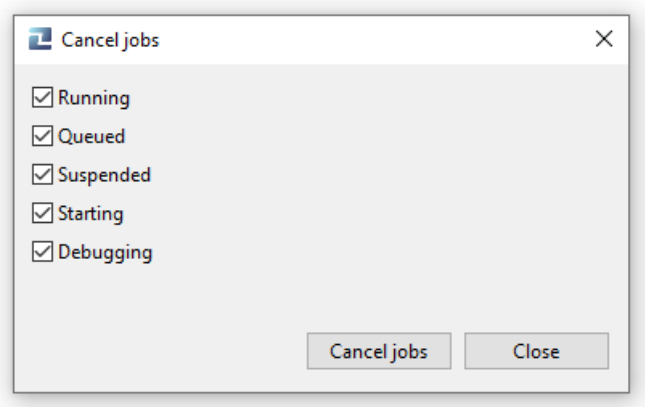


Figure 15 – The *Cancel jobs* dialog box

In the *Cancel jobs* dialog box, the user can filter which running statuses are going to be cancelled. For a list of job statuses, see → Job lifecycle.

7.6.3 Cancellation by the *zs:cancel* action

Cancelling can also be done using the `zs:cancel` action in a script. *Figure 16.* shows the case when the action cancels one particular job, identified by its job id.

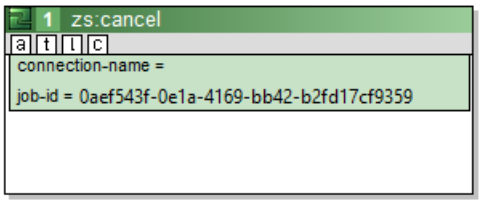


Figure 16 – Cancelling a job with the `zs:cancel` action by job-id

The `zs:cancel` action can also cancel multiple jobs just like using the Cancel jobs dialog in the client applications (see → Multiple cancellation by job statuses). *Figure 17.* shows an example for cancelling all the jobs with *queued* and *running* statuses.

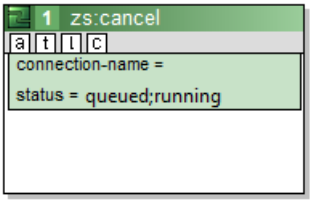


Figure 17 – Cancelling all jobs with the `zs:cancel` action by job statuses

7.6.4 Zagreus Server startup and shutdown cancellation

Jobs can be cancelled automatically on Zagreus Server startup at shutdown. This can be configured in the configuration of the Zagreus Server (see → Zagreus Server configuration). The following settings are configurable:

- *server.canceljobs.onstop*
  If this boolean setting is true, the server will cancel all jobs on shutdown that have one of the following job status: *queued*, *starting*, *running*, *suspended* and *debugging*. The default value is true.
  Cancelling the jobs during shutdown is recommended in order to prevent jobs remaining in an inconsistent state. Though a Zagreus Worker can still continue to execute a job when the Zagreus Server is shut down, the queue is not active on the server side, so the Server and the Worker cannot be synchronized anymore.

- *server.canceljobs.onstart*
  If this boolean setting is true, the server will cancel all jobs on startup that have one of the following job status: *queued*, *starting*, *running*, *suspended* and *debugging*. The default value is false.
  This is useful when there are jobs left in the queue (stored in the queue in the database when the Server was down) which the user does not want to execute. Practically most of these jobs are in an inconsistent state anyways, so cancellation of all jobs is highly recommended by using the *server.canceljobs.onstop* setting.

7.6.5 Zagreus Worker automatic restart cancellation

The job status can be *canceled* in the rare case when the communication between the Zagreus Worker-Controller and the Zagreus Worker has been lost. If the duration of communication outage exceeds the default 30 seconds value, the Worker-Controller automatically restarts the particular Zagreus Worker. If there was a running job on the given Worker, the status of this job will be set to *canceled*.

The default threshold of this behavior can be set in the Zagreus Worker-Controller configuration (see → Worker-related properties), with the *workercontroller.workerrestarttimeout* property.

8. Scripts

Scripts are the most important resources in the Zagreus System. They contain executable actions in a flexible structure which makes possible to define a proper execution order for the actions and the dataflow pipeline among the logical parts of the script.

Scripts use a hierarchical model to define a sequence of executable actions. *Actions* are the basic building blocks of the script, so the script resource is a container for its own actions. Scripts contains all the actions that belongs to the same logical workflow. Scripts are stored in a XML format (see → XML representation), but they are viewed and edited in the fully-featured Script Editor in the Zagreus Client, see → Script Editor.

Though templates are technically different resource types, they strictly belong to the script execution mechanism, so they are described in this chapter as well.

8.1 Actions

An action represents a basic function or feature that can be executed. It can transport or transform data as well as perform complex operations in the background.

8.1.1 Action groups and action name

Actions are divided into logical groups (Action groups, e.g. `file`, `mail` or `db`). Each action has a unique, fully qualified name containing two parts separated by the ":" character: the action group title and the action name like `file:read` or `db:sql`. All actions in the same group have the same action group qualifier. The basic form is then:

```
<actionGroupTitle>:<actionName>
```

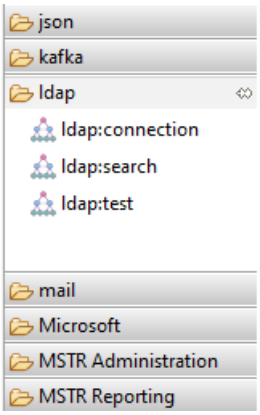


Figure 1 – Actions are listed in groups on the palette

Technically this is stored as a namespace-prefix and element name in the XML correspondingly. Fully-qualified names are useful to avoid collisions (for example, `db:connection` and `ldap:connection` actions are different actions). This name is always written on the action icon or container (depending on which view mode is used, see also → View modes)

8.1.2 Action attributes

Attributes are the input parameters of the action, they store name-value pairs, similary to the case of XML attributes. The user can set a parameter by selecting the attribute name and specifying its value.

Action documentation can be displayed by right-clicking on the action name in the palette (see also → Action help).

8.1.2.1 Predefined and common attributes

There are predefined attribute names for each action, which are action-specific inputs for the particular feature, and there are also common attributes which can freely be used for every action. These latter define some common functionality, e.g. the *output* common attribute is used for storing the result of the action in a file resource.

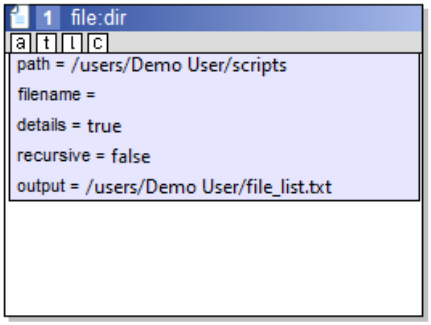


Figure 2 – `File:dir` action with a common attribute

For example, in *Figure 2.*, the `file:dir` action has four pre-defined attributes (*path, filename, details* and *recursive*), while *output* is a common attribute which was added manually.

The pre-defined attributes are shown in the Script Editor after drag-and-dropping an action from the palette (when the view mode is set to *Full*, see also → The full view mode). The common attributes can be drag-and-dropped to an action from the palette too.

Attributes can be edited in several ways in the Script Editor, see → Attributes.

8.1.2.2 Required and optional attributes

There are attributes which are required (mandatory) to be able to process the given action. These attribues must be filled by the user. Sometimes these values are already filled when the action is created in the Script editor. Required attributes are indicated in the Documentation dialog, see → Action help. If the value of a required attribute is left empty, an error will be thrown.

Other attribues are optional, i.e. they can be left empty. Optional attributes can still have a default value even when they are not specified, see also the Documentation dialog → Action help.

8.1.2.3 Attribute value types

Textual, numeric and date types are stored as textual values and checked only during the execution of the script.

There are some special types though, which can restrict the value selection:

- *elements from a list*
  there are attributes that expect their values from a predefined list. The user can select a list element from the *Attributes* dialog box, see → Attributes

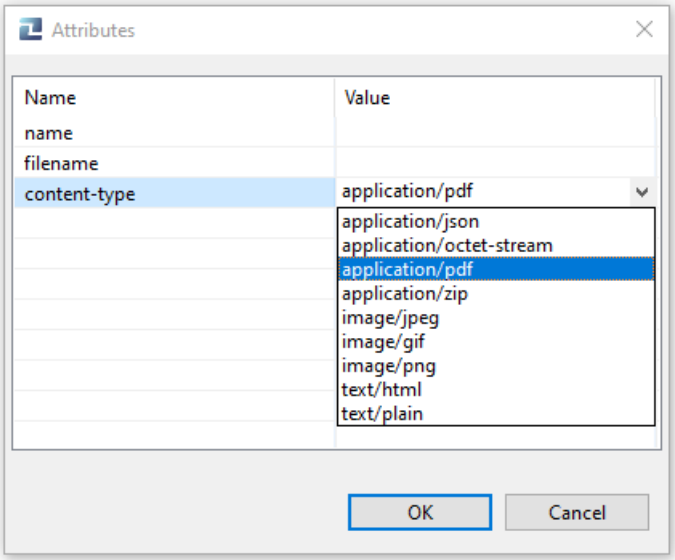


Figure 3 – An example for the list selection of an attribute in the *Attributes* dialog box

- *boolean*
  other attributes can have a boolean type logical value as *true*/*false*. This is also selectable from a combo box in the *Attributes* dialog box, see → Attributes

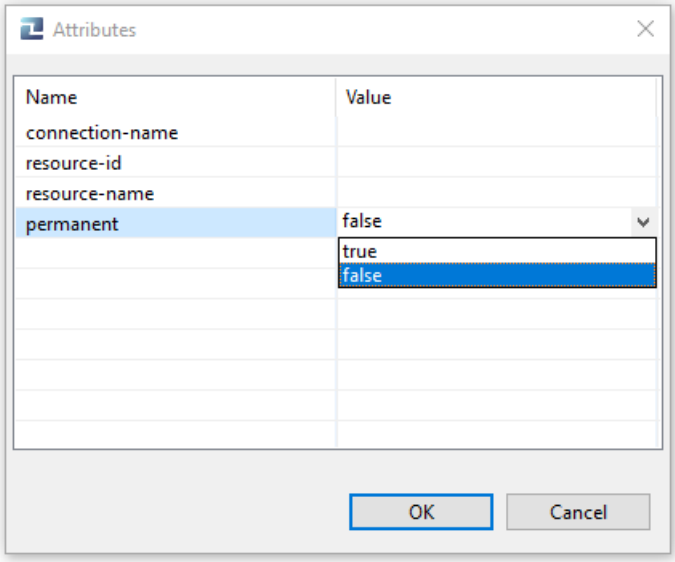


Figure 4 – An example for a boolean type attribute in the *Attributes* dialog box

8.1.3 Action content

Actions can contain further elements inside. In some cases, child content is needed for the proper action performance (for example, `mail:body` is needed as a child action of `mail:send` parent action), while in some other cases a dataflow pipeline is implemented by this way (a `mail:attachment` parent action contains a `file:read` child action, in order to attach a file to the email). Sometimes a simple textual content is enough as an input for the particular action (for example `z:log` action).

8.1.3.1 Textual content

An action can contain a simple textual content just like PCDATA in an XML element. Some actions expect text content only (e.g. `z:log`, `db:sql`), while others can process textual content and other child action content together (e.g. `rest:call`).

Textual content is processed in UTF-8 encoding.

Figure 5 – Textual content of a `z:log` action

Textual content can be edited in several ways in the Script Editor, see → Text content

8.1.3.2 Child action content

Actions can also contain child actions. They are special or general inputs for the given action. The ouput result of the child actions serve as the input of the parent action.

For the order of processing actions, see → Order of execution, result flow, for checking the result pipelines, check → Pipelining actions.

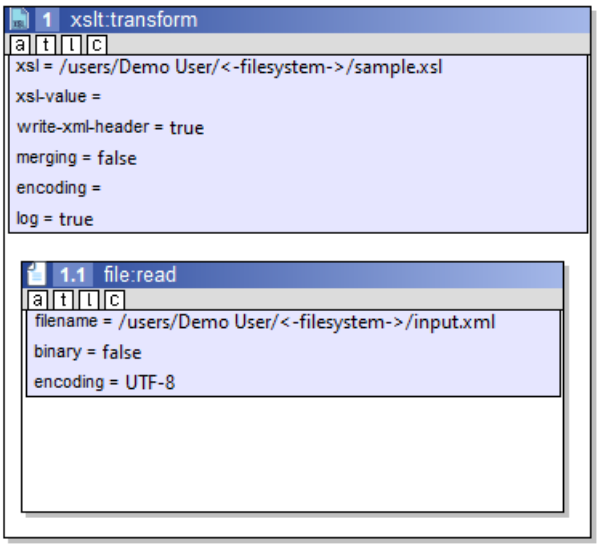


Figure 6 – An example for a child action serving as an input to the parent action

There are several types of child actions:

- *A child action which provides output for further usage*
  This is the most general case: a child action generates an output which is used as an input of the parent action. See result flow → Result flow .

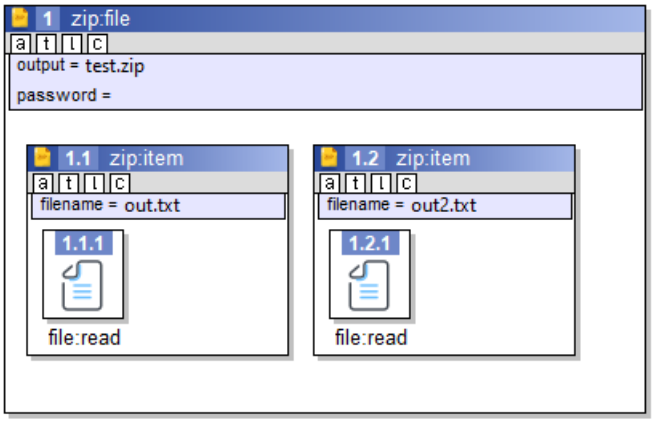


Figure 7 –A `zip:file` parent action uses the outputs of the `zip:item` actions.

They also pipeline data from their own `file:read` action children

- *Some child actions can be used with a particular parent*
  these child actions make sense only with the parent action together, and the parent actions may need them as a mandatory child action or as an optional one. For example, action `z:in` can and must be the child of action `z:foreach`, just as the relation between `mail:body` and `mail:send`:

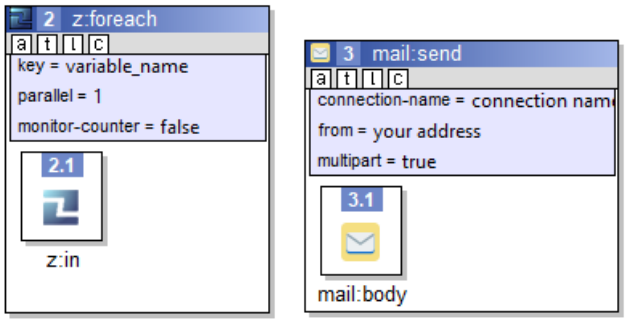


Figure 8 – Child actions belonging to particular parent actions

- *A child action that behaves as an attribute*
  In some rare cases, setting an attribute value is not possible or not convenient for an action. In these cases, an attribute can be changed to a child action which has the same name as the attribute was. There is a context menu to do it in the Script editor, see → Attribute as child element
  These are so-called pseudo-actions because they do not exist in the palette and there is no list of their possible names. Any predefined attribute can be converted to a child element. *Figure 9.* shows a use case where an attribute value would be read from a file. The first and second picture behave just like the same, the *pdf-name* attribute was converted to a `pdf:pdf-name` child action. Then in step 3, the original value was replaced for one that is set by a `file:read` action output.

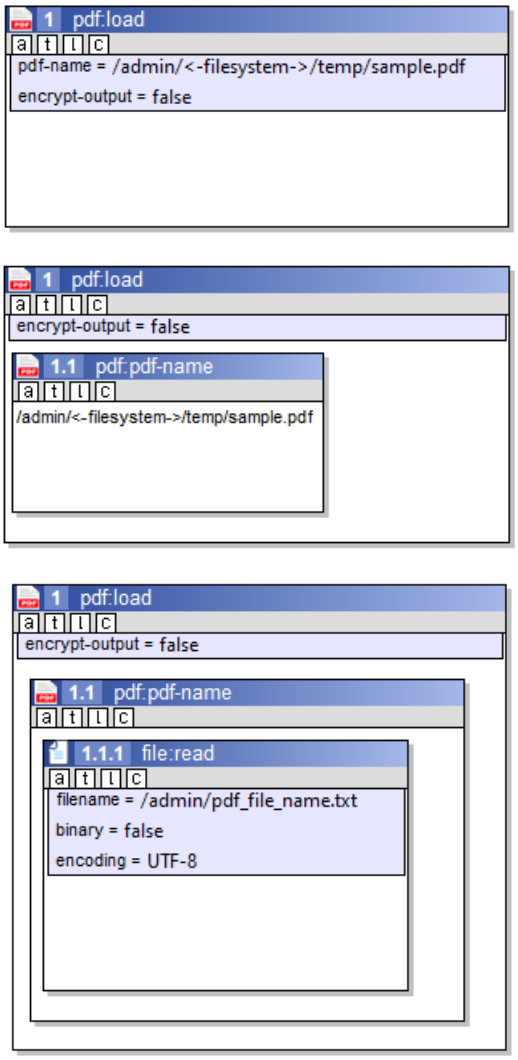


Figure 9 – Usage of an attribute converted to a child action

8.2 Order of execution, result flow

In Zagreus, script execution follows a comprehensive logical pattern in terms of execution order of actions and input-output data flow. The order of execution can be imagined horizontally (sibling actions are executed one after the other), but the data flow is going vertically, from inside to outside – from the child action to the parent action. This way, a wide variety of use cases can be implemented easily in a Zageus script.

8.2.1 Ordering numbers

Each action has an ordering number. This is visible before the action name in the Script editor.

On the root level (i.e. canvas) the ordering numbers are simple ordinal numbers like 1, 2, 3 etc. Following the hierarchical model, child actions are numbered with prefixes of the ordering number of the parent. Thus, the first child of Action 2 will have the ordering number of 2.1.

In *Figure 10.*, a general structure is demonstated:

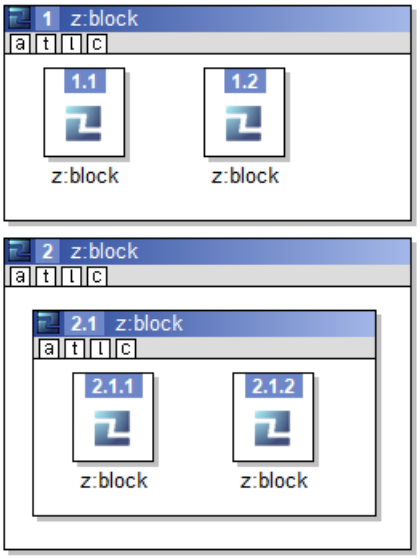


Figure 10 – Parent-child actions and siblings

8.2.2 Execution of an action

The execution of an action consists of three different steps:

1) Processing and evaluating the attributes
All attributes from the pre-defined list of the action are read by the processing engine. If needed, variable references in attribute values are resolved at this point.

2) Processing child elements
If the action contains any child elements (i.e. child actions or textual content), they are executed.

3) Processing the actual action
After processing all possible inputs for the action (i.e. attributes and child elements), the processing engine finally executes the actual action.

8.2.3 Result of an action

Executing an action does not only mean that the given action performs a specific task, but it can also produce an output called *action result*. For example, the result of a `file:read` action is the content of the specified file. The type of the action result is one of the Zagreus data types, see → Data types .

The result can be accessed in one of the following ways:

- *Implicitly propagated*
  If an action has a result, it is always implicitly propagated upwards to the parent action, see → Pipelining actions.

- alias *and* alias-global *attributes*
  The action result can manually be stored in a script variable (see → Variables) by the *alias* and *alias-global* attributes, see → alias and alias-global attributes.

- output, debug-output *and* worker-output *attributes*
  The action result can manually be stored in a simple file by the *output*, *debug-output* (see → output, debug-output and output-encoding attributes) and worker-output (see → worker-output attribute) attributes.

8.2.4 *result-message* of the script

The script itself can also have an output that is stored in a job property called *result-message* (see → Job properties). The result-message of the script is a textual value

which can provide useful feedback for the user about how the execution went. It can be set in the script by the `z:exit` action, or by the *exit-message* attribute of any action, see → exit and exit-message attributes. It can be read from the finished job by the Finished Jobs window in the Zagreus Client (see → Finished jobs window), the Job properties dialog of the Zagreus Monitor (see → Job properties dialog) or by the `zs:jobinfo` and `zs:joblist` actions.

For example, if the script ran successfully, the last action can be a `z:exit` action with a human-readable message that can be seen in the Finished Jobs window in the Zagreus Client. This way, not only the job status would inform the user about the status of the script, but also the result message can reinforce the successful execution. Another possible use case could be to distinguish multiple scenarios where the script execution was successful.

8.2.5 Basic traversal of the actions

Script execution always starts with executing `Action 1`. This is the entry point of the execution for the whole script.

Actions with child contents are processed in the following way:

- The parent action processes its own child actions first (if any) in the order of their ordering numbers. This happens in a recursive manner.
- Then the action itself is processed, i.e. it is performing its own function.

In the example of *Figure 10.*, the execution order is the following:

1) Execution starts with `Action 1`. This action has child actions, so it processes them first. `Action 1.1` and `Action 1.2` are processed one after the other.
2) Then `Action 1` itself is executed. If there were any outputs of its children, it can use it as its own input, see → Pipelining actions
3) Execution is followed by `Action 2`. It has a child action, `Action 2.1`, so `Action 2` starts to process its child first.
4) `Action 2.1` also has children, so it processes them first. `Action 2.1.1` and `Action 2.1.2` are processed respectively.
5) `Action 2.1` is executed. If there was any output from its child actions, it can use them as its own input, see → Pipelining actions
6) `Action 2` is executed last. It can also use the output of `Action 2.1` as its own input.

So the execution order is:

```
 Action 1.1
 Action 1.2
Action 1
  Action 2.1.1
  Action 2.1.2
 Action 2.1
Action 2
```

8.2.6 Special control flow statements

Control flow statements change the flow of execution. Following the logics of programming languages, Zagreus has several control flow statement actions and attributes. These structures are used in the script in order to control the flow of execution based on a condition or a loop definition.

8.2.6.1 *z:if* action

The `z:if` action is a conditional control flow action that allows to execute a different child action of the parent action, based on a specified condition. This condition is set in the *test-expr* attribute. The dedicated children elements are `z:then` and `z:else`. For example, the script displayed in *Figure 11.*, prints “less than five”.

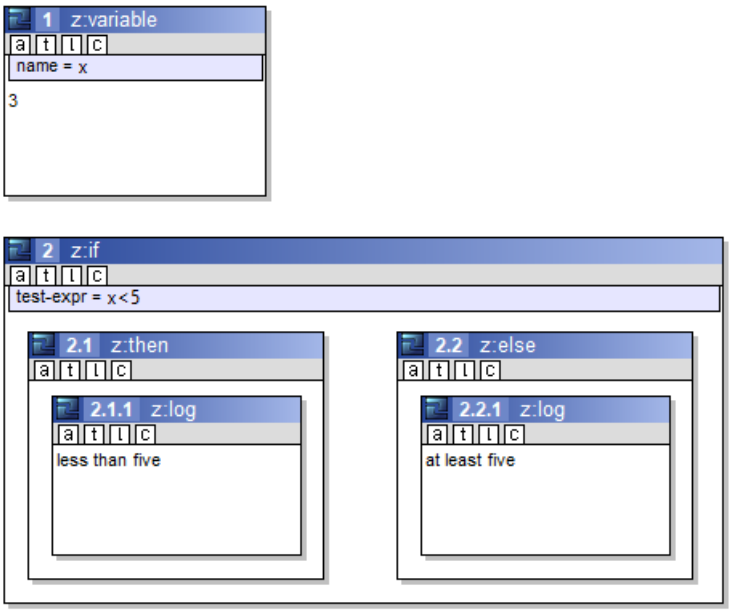


Figure 11 – Example for a `z:if` action

8.2.6.2 *test-expr* attribute

The *test-expr* attribute is also common attribute, i.e. it can be set for all actions, not just for `z:if` (see previous sub-chapter). The evaluation result of the attribute decides whether the action is to be executed or not. *Figure 12.* shows an example for the usage of this attribute: the `Action 4` will not be executed because the expression specified evaluates to false.

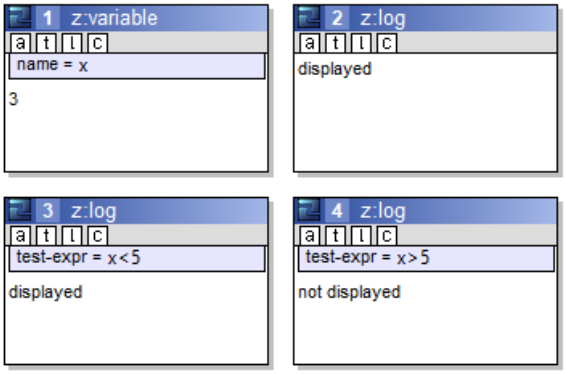


Figure 12 – Example for the *test-expr* common attribute

8.2.6.3 z:switch action

The `z:switch` action is a control flow action that allows one selection of several choices to be executed, based on a specified condition. This condition is set in the *test-expr* attribute. The dedicated child elements are `z:case` and `z:else`. Former sub-actions define what to execute on the matching condition, latter is the default / else branch.

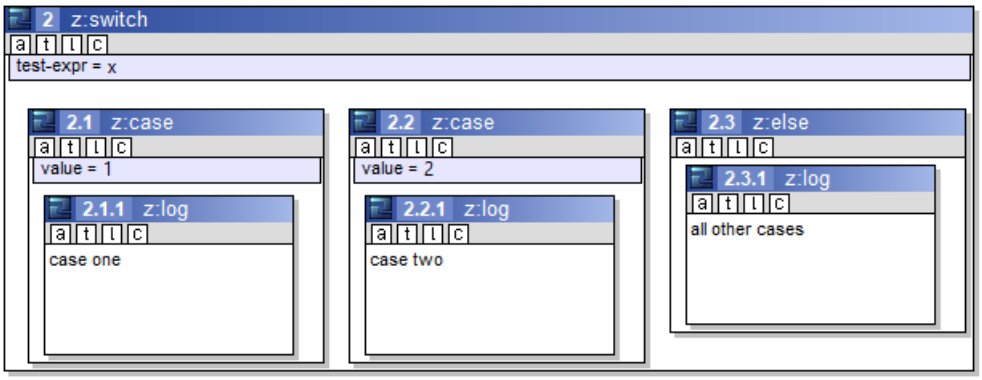


Figure 13 – Example of a `z:switch` action

8.2.6.4 *z:for* action

The `z:for` action is a control flow action that allows to iterate over a range of numbers. The content of the `z:for` action is executed within each iteration. For example, *Figure 14.* shows a `z:for` action which logs the numbers 1, 2, …, 10.

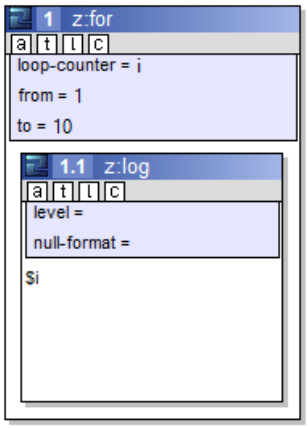


Figure 14 – Example of a `z:for` action

8.2.6.5 z:foreach action

The `z:foreach` action is a control flow action that allows an iteration over the elements of a pre-defined list. This pre-defined list has to be defined in the mandatory `z:in` first child element which can also be specified as the attribute *in*. In *Figure 15.*, the foreach loop logs the first ten prime numbers.

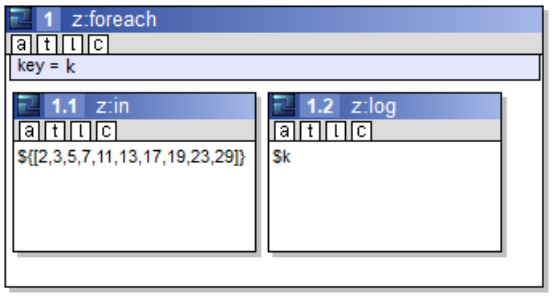


Figure 15 – Example of a z:foreach action

The `z:foreach` action has a special parallel thread feature, see → Parallel threads in the z:foreach action.

8.2.6.6 *z:while* action

The `z:while` action is a control flow action that allows to repeatedly execute the content of the action as long as a given condition is true. The condition is evaluated before each iteration.

In *Figure 16.*, a four-element list is defined as variable `$x`. In `Action 2`, another variable, `$j` is used as an index counter, which value is increased by 1 in every loop iteration in `Action 3`. The condition in the `z:while` loop checks if the next list element is not null, in this case the loop body (children `3.1` and `3.2`) is executed and the condition is checked again. The result of the while loop is the logged `$x` list elements one by one.

For better understanding of engine expressions, see → Engine expressions

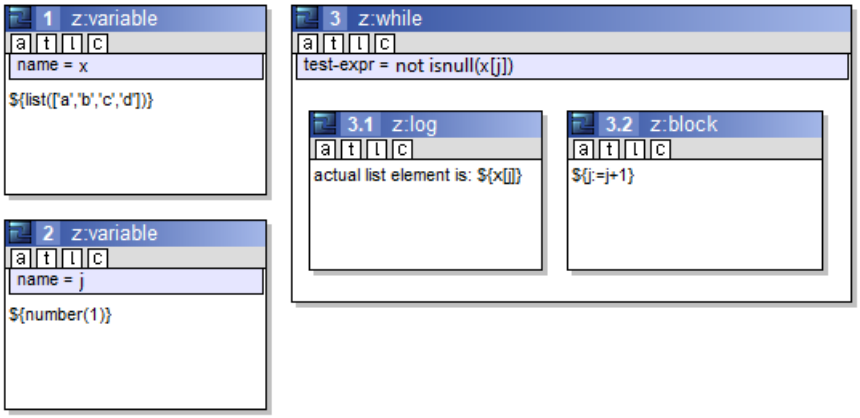


Figure 16 – Example of a `z:while` loop

8.2.6.7 z:do-while action

The `z:do-while` action is a control flow action that allows to repeatedly execute the content of the action as long as a given condition is true. The condition is evaluated after each iteration, so (in contrast with `z:while`) the *content of the action is executed at least once*.

8.2.6.8 Goto expressions

If a goto expression is defined in the Script Editor (see → Script Editor), it allows the execution of the script jump to a different action (on the same sibling level). For example, *Figure 17.* shows a goto expression set to `Action 2`. Since the goto condition is true, `Action 4` will be executed right after `Action 2`.

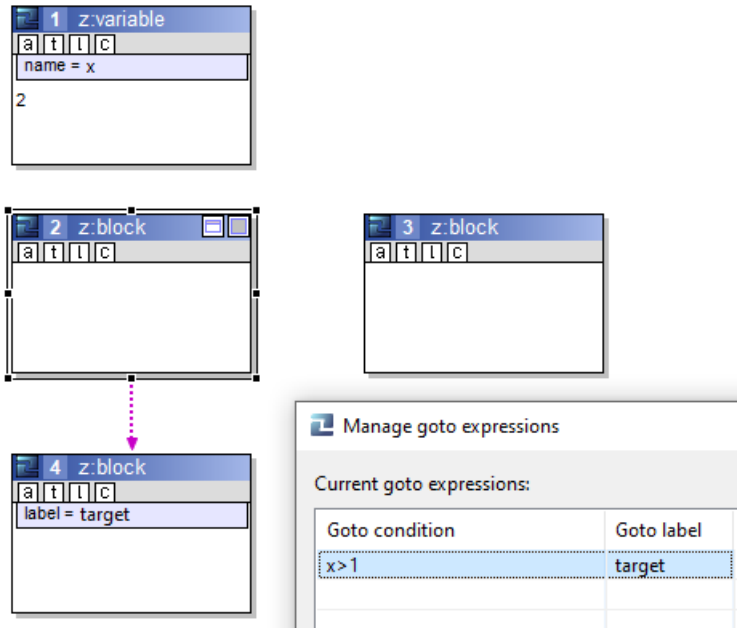


Figure 17 – Example of a *goto* expression

8.2.6.9 z:break and z:continue actions

For loop actions (`z:for`, `z:foreach`, `z:while`, `z:do-while`) there are two further actions in order to control the flow of execution. They need to be used inside the body of the loop.

- `z:break`
  It terminates the execution of the current iteration and the processing engine continues executing the next action after the loop. So it exits the loop early and skips any remaining iterations.
  In *Figure 18.*, a list `$x` is defined with mixed elements in it, the 5th element is a number, all the others are of text type. Inside the `z:for` loop, a `z:if` action is checking whether the current list element is a number. If so, it breaks the whole loop, until then it logs the corresponding elements of the list. The result would be *'a'*, *'b'*, *'c'* and *'d'*.

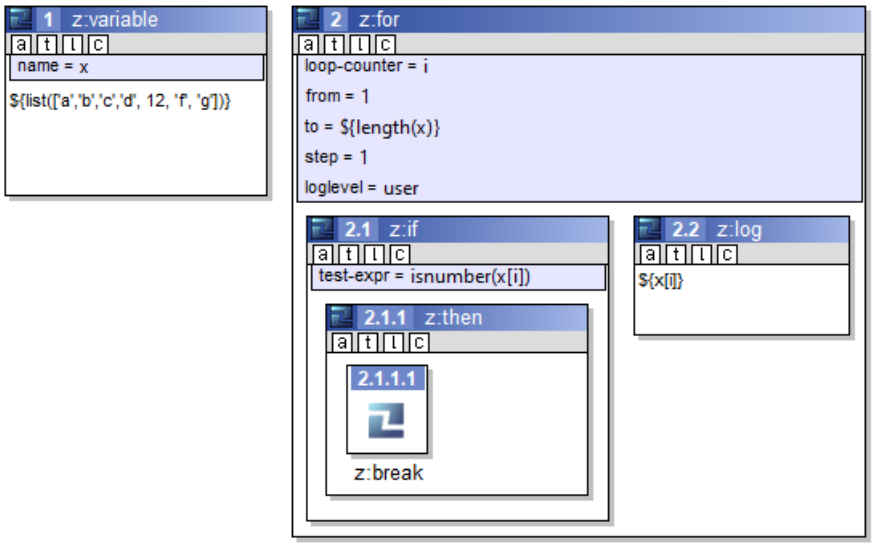


Figure 18 – Example for `z:break`: the first occurrence of a number breaks the loop

- `z:continue`
  It also terminates the execution of the current iteration, but then the engine jumps immediately to the next iteration. It does not exit the loop itself, only the current iteration.

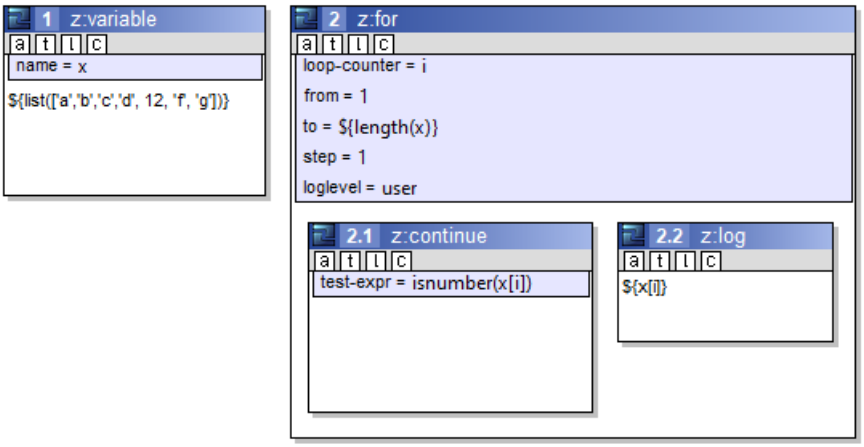


Figure 19 – Example for `z:continue`: numbers are not logged

In *Figure 19.*, a similar example is shown as in *Figure 18.*, but this time, a simple common attribute *test-expr* is used to evaluate if the `z:continue` is executed or not. In case of a number, the `z:log` is not executed, so the result is *'a'*, *'b'*, *'c'*, *'d'*, *'f'* and *'g'*.

8.2.6.10 z:exit and z:return actions

There are two special actions that immediately interrupt job execution: `z:exit` and `z:return`. Aside from the fact that they break the execution flow, they can set the *result* or the *result-message* of the script as well, see → Result flow.

- `z:exit` exits the execution flow and sets the result-message of the script as well as the job status (for example 'error' or 'finished', see → Job lifecycle). This action is useful in error handlers: *Figure 20.* shows an example where, after catching an error with a `z:on-error`, the `z:exit` action immediately interrupts script execution with *error* status (see → Job lifecycle) and a custom result message (see → result-message of the script).

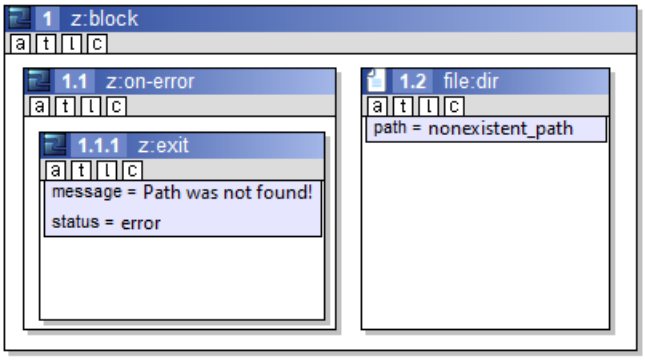


Figure 20 – `z:exit` action controls the script status and result-message

- `z:return` exits the execution flow and sets the result of the script. *Figure 21.* demonstrates an example for using `z:return`. It is highly recommended that the `z:return` action should be the last action of the execution flow, as any other actions after this would not be executed!

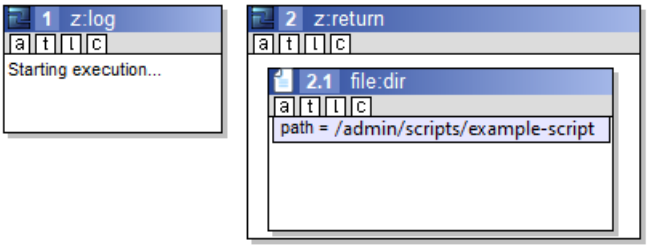


Figure 21 – An example for a `z:return` action

8.2.6.11 exit and exit-message attributes

The *exit* attribute immediately exits the execution flow, just like the `z:exit` action in the previous chapter. If the *exit-message* attribute is also filled along with the *exit* attribute, the result message will be set to the specified value.

8.2.7 Parallel threads in the *z:foreach* action

The action `z:foreach` has a special feature: it can use parallel processing for its iterations. In some important cases, it can make the execution much faster. For example, there is a long list of email addresses, any simple loop would send emails one by one which can take a lot of time. In this case, the order of the addresses (as well as the order of email sending) is not important, so parallel execution threads can do the job faster.

Normally, the execution of an action is processed in the *main thread* of the particular engine. The main thread is the first thread created by the Java Virtual Machine (JVM) when the engine starts running. All actions are executed one by one in this thread. The JVM thread id of the main thread is *1*. This is called single-threaded or sequential execution type.

In contrast, parallel execution is when multiple tasks are executed simultaneously. In order to do this, it is necessary to create and manage multiple threads. There is a special attribute *parallel* for `z:foreach`, it is set to *1* per default (sequential processing). When it is set to greater than one, the `z:foreach` action works as following:

- It creates a new child thread for all iterations (regardless of the *parallel* setting).
- It allows these threads to be executed at the same time but only according to the *parallel* setting. So when *parallel="3"* and there are 5 iterations, then each child thread will be executed in a controlled way, so that only 3 of them can be executed at the same time.
- Two variables can be obtained in the loops: `$threadId` (the JVM thread id) and `$parallelThreadId` (Zagreus managed parallel thread id).

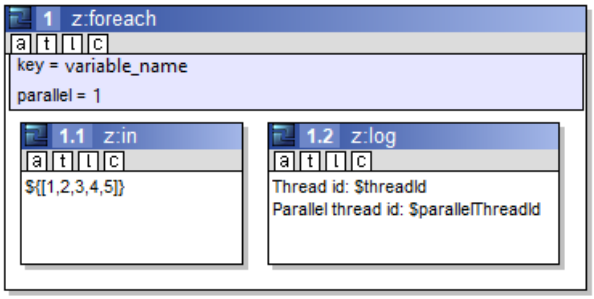


Figure 22 – Sequential processing in `z:foreach`

In *Figure 22.,* the sequential processing is shown with the *parallel="1"* setting. The result of the `z:foreach` loop in the logfile is:

```
<"1" z:foreach>
<"1.1" z:in>
</"1.1" z:in>
<"1.2" z:log>
Thread id: 1
Parallel thread id: 1
</"1.2" z:log>
<"1.2" z:log>
Thread id: 1
Parallel thread id: 1
</"1.2" z:log>
<"1.2" z:log>
Thread id: 1
Parallel thread id: 1
</"1.2" z:log>
<"1.2" z:log>
Thread id: 1
Parallel thread id: 1
</"1.2" z:log>
<"1.2" z:log>
Thread id: 1
Parallel thread id: 1
</"1.2" z:log>
</"1" z:foreach>
```

It means that there is a main thread (JVM thread id, `$threadId=1`) and a single-threaded processing (*parallel=1* setting, the `$parallelThreadId=1` as well).

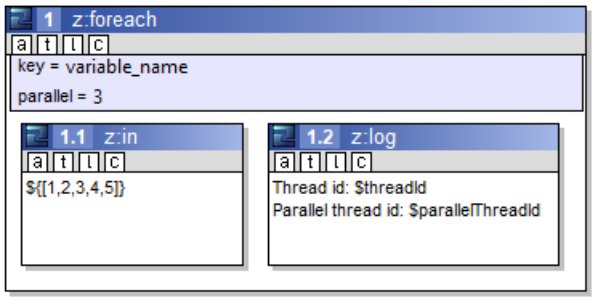


Figure 23 – Parallel processing in `z:foreach`

In *Figure 23.,* three parallel processing threads are set with *parallel=”3”*. The result of this `z:foreach` is the following:

```
<"1" z:foreach>
<"1.1" z:in>
</"1.1" z:in>
<"1.2" z:log>
<"1.2" z:log>
<"1.2" z:log>
Thread id: 1027
Parallel thread id: 1
</"1.2" z:log>
Thread id: 1028
Parallel thread id: 2
Thread id: 1029
Parallel thread id: 3
</"1.2" z:log>
</"1.2" z:log>
<"1.2" z:log>
<"1.2" z:log>
Thread id: 1031
Parallel thread id: 2
Thread id: 1030
Parallel thread id: 1
</"1.2" z:log>
</"1.2" z:log>
</"1" z:foreach>
```

It shows that 3 `z:log` actions started at the same time, and after they were finished, two others were started in parallel again. So the 5 iteration was executed in a way that maximum 3 of them were executed at the same time in parallel.

In this case, Zagreus maintains 3 parallel ‘virtual’ threads (numbered as *1,2,3*) and the actual JVM thread id is different for all the iterations (*1027, 1028, …, 1030*) due to the implementation of the Java threading system.

Info: *The parallel setting is limited by the licence setting as well.*

8.2.8 Templates

Templates define reusable parts of a script. The action `z:template` is used for this purpose. The child actions of `z:template` will be executed when the template is

called by the `z:call-template` action. Action `z:template` also can handle parameters with its optional child action `z:param`.

Figure 24. shows an example for using a template. `Action 1` defines the template and its parameter *x*, which has a default value *"3"*. (This default value will be applied if the given parameter is not specified at the `z:call-template` action.)

Both `Action 2` and `Action 3` call the template. Referencing the template is done by matching the *name* attributes, which is *"template_1"* in this example. `Action 2` calls the template without specifying any parameter, while `Action 3` is calling the template with *x="2"*.

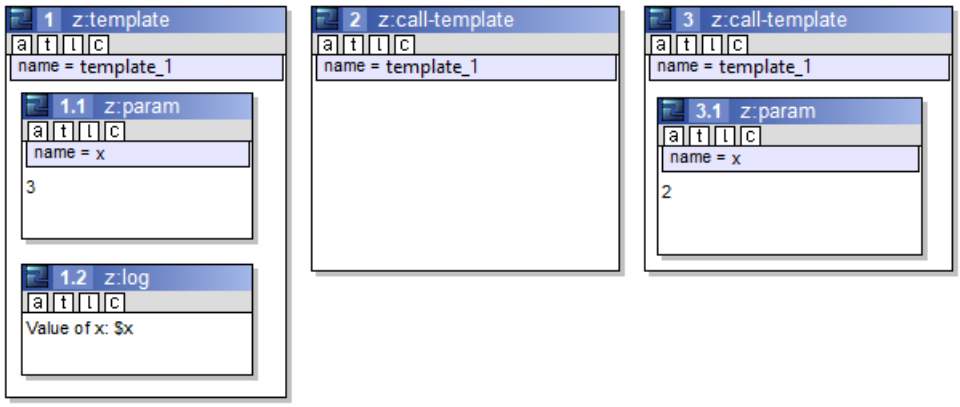


Figure 24 – Calling a template with and without a parameter

The result of executing this script is the following:

```
<"1" z:template>
</"1" z:template>
<"2" z:call-template>
<"1.2" z:log>
Value of x: 3
</"1.2" z:log>
</"2" z:call-template>
<"3" z:call-template>
<"1.2" z:log>
Value of x: 2
</"1.2" z:log>
</"3" z:call-template>
```

`Action 1` is not executed, hence it is just the definition of the template: without a `z:call-template` action it would never be executed. When `Action 2` and `Action 3` call the template, the content of the template is executed: Action `z:log 1.2` is logging the value of parameter *x*, which is different in the two calls.

8.2.9 Result flow

8.2.9.1 Pipelining actions

Most of the actions produce output data, which is called their *result*. This result is used as the input of the parent action. In this way, the user can create a pipeline-like processing of data. In *Figure 25.*, the output of the `file:read` action serves as an input of the `xslt:transform` action. Its output is the input of the `z:log` action.

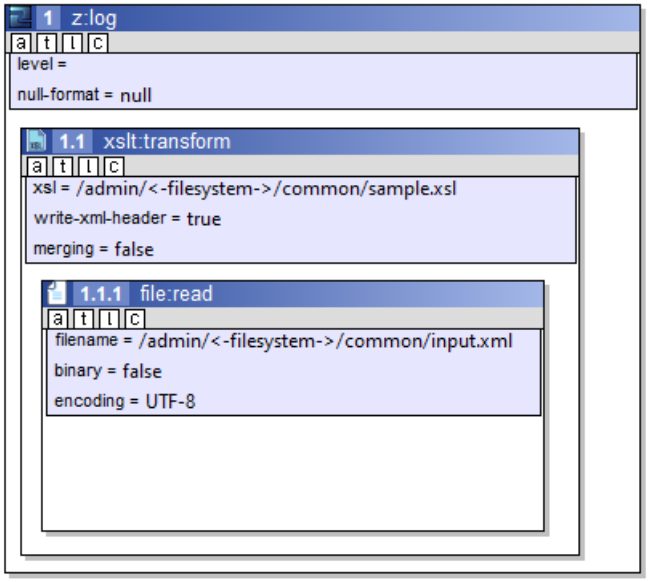


Figure 25 – Data flows from child to parent action

There can be multiple children within one parent action. In this case, it depends on the parent action implementation how it will be handled. For example, in *Figure 26.*, the `file:write` parent action can receive multiple chunks of input data from the corresponding `file:read` children action, and it can merge them together with the *append="true"* attribute.

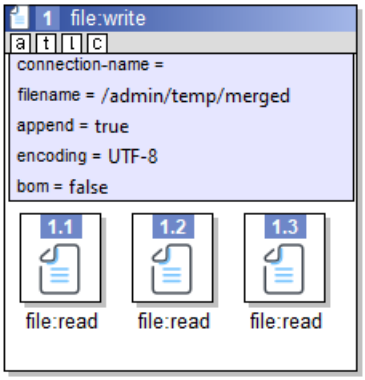


Figure 26 – Data flows from child to parent action

8.2.9.2 alias and alias-global attributes

Aside from the pipeline-like dataflow, there are other ways to pass the result data among actions. Many times it is not practical to produce data directly inside the action that would use it, or the script is more readable doing it otherwise.

One good solution is to use an *alias* attribute. It creates a variable (see also → Variables) out of the result of the action, and that variable can be used at any point in the script within the same local scope (see → Local scope). In this way, the pipeline-like result flow can be avoided, and a horizontal result usage can be implemented. In *Figure 27.*, the three-level pipeline is modified from *Figure 25.* with the use of the *alias="result_of_xslt"* attribute. It creates a `$result_of_xslt` variable in the background that is used in the `z:log` action later.

> ***Info:****The* alias *attribute creates a variable with a local scope. If you need a global scope variable, use the* alias-global *attribute!*

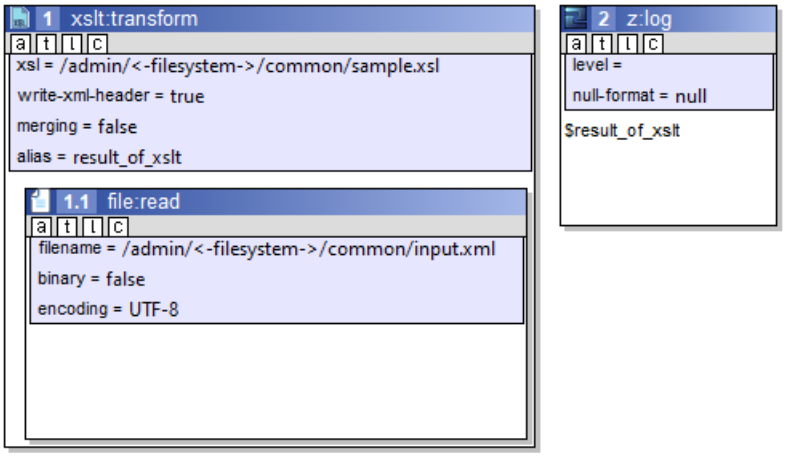


Figure 27 – Using an alias attribute for result re-use

8.2.9.3 output, debug-output and output-encoding attributes

When the result is simply needed to be written into a file, the *output* attribute can be the solution. It specifies a filename with a path where the output result data will be written. In *Figure 28.*, the output of the `xslt:transform` action is written to the specified *output.dat* file.

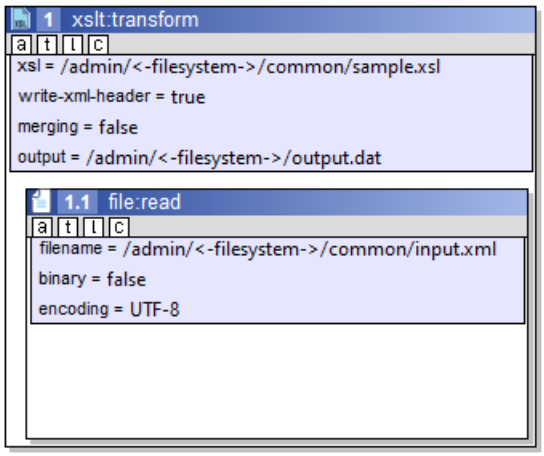


Figure 28 – Using an output attribute for storing the result

The *debug-output* attribute behaves in the same way as the output attribute does, but it is processed only when the script is executed in debug mode (see → Debugging in the Zagreus Client).

If the output is textual data, the output-encoding attribute can be used to set the encoding of the written file.

> ***Info:*** *Even when the* alias *and* output *attributes are set, the result of the action will still be propagated to the parent action.*

8.2.9.4 worker-output attribute

The *worker-output* attribute behaves the same way as the *output* attribute described in the previous chapter. The only difference is that the output is written into the Zagreus Worker filesystem (see → Local filesystem in the Zagreus Worker) instead of the embedded database or the local filesystem.

8.2.9.5 *no-result* attribute

In some cases it is needed to explicitly tell the script processor not to propagate the result of the action upwards to the parent action. For example, the output is too large to store it in the memory. In such cases the no-result attribute can be used. This does not affect the behavior of the output and debug-output attributes.

8.2.9.6 eval-output attribute

If the result of the given action is of a textual type, in some cases it might be useful to be evaluated as an engine expression (see also → Engine expressions). By using the eval-output=”true” setting, the result is evaluated before passing as an input to the parent action. In *Figure 29.*, the result of the file:read action is evaluated before the parent z:log action processes it, so the message “The value of y is: variable value” will be logged by resolving the $y string part to the value of variable y.

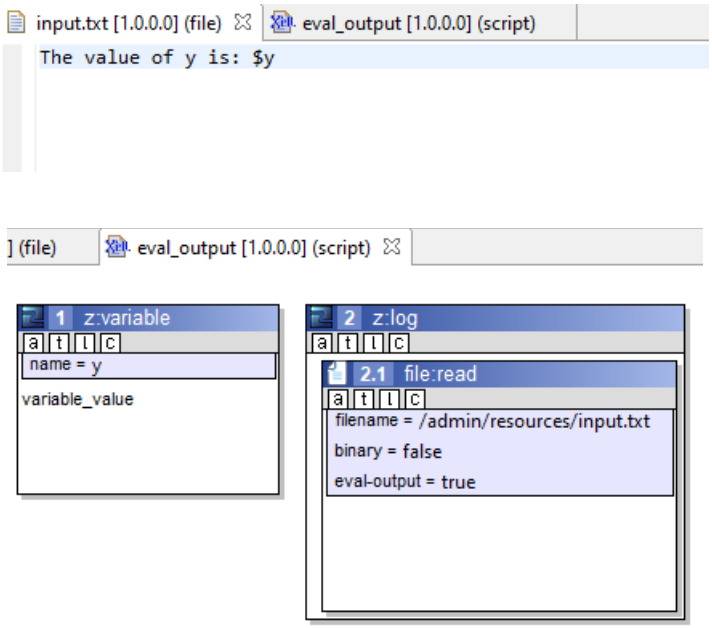


Figure 29 – Evaluating the file:read result

> ***Info:*** *Common attributes that are related to logging (like* log*,* log-result-attributes*) might be considered as a special type of result flow as well during the execution of the Zagreus script.*

8.2.10 Result attributes

In many cases, not just one single result is expected as an output of an action, but many other output parameters related to the result. For example, if a result is a list type data, it can be useful to know the number of rows of the result, or the number of processed lines of a complex action like `zs:migrate`. These additional output data attributes are set after the execution of a particular action and they are called *result attributes.*

For the specific list of all possible result attributes for an action, see → Action help about action documentation.

Contrary to the *result* data, one must explicitly refer to the result attributes in order to check their values. The result attributes are mapped together as an associative array with name-value pairs.

For example, for a `zs:list` action, there are two result attributes defined: *rowcnt* (the number of rows of the result) and *execution_time* (the execution time of the action in milliseconds). After the `zs:list` execution, a special associative array is generated with values like: `[rowcnt:16, execution_time:230]`. This whole array can be referenced as well as its keys inside (the result attributes). For associative arrays, see → Engine expressions

This reference can be done in one of the following ways:

- *Using the* result-attributes *attribute*
  This attribute can specify a variable name which will contain the result attributes after the execution of the current action. The attribute itself can be referenced as a whole array, and its keys as the result attributes.

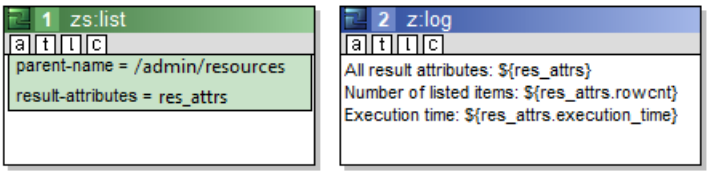


Figure 30 – Using *result-attributes* attribute to reference the result attributes

In *Figure 30.,* `$res_attrs` is used as the referencing variable name. The result of the `z:log` action is the following:

```
All result attributes: [rowcnt:16,execution_time:230]
Number of listed items: 16
Execution time: 230
```

- *Using special reference to the last executed action type*
  Instead of a named variable, Zagreus automatically maps the result attributes to a special variable name which is derived from the action name:

```
$nameSpace_actionName
```

This automatically generated variable always stores the result attributes of the *last executed action of the specified type*. It is useful to quickly check out some result attribute when testing or debugging.

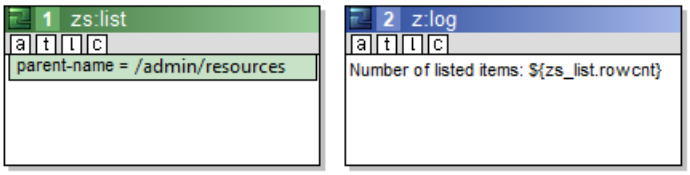


Figure 31 – Using the special variable name to reference result attributes

Info: *the* result-attributes *attribute sets a variable to a local scope. A* result-attributes-global *attribute can set a global scope variable if needed, just like the* alias *and* alias-global *attributes.*

Info: *The result attributes can also be logged with the* log-result-attributes *attribute.*

8.3 Includes

Scripts can include the content of other resources as well. A special `z:include` action is used for this purpose. The main point of using includes is the reusability of other resources:

- A connection resource, once defined and configured properly, can be used from many different scripts.
- A template (see → Templates) can be easily used from other scripts without modifying the template itself.
- Scripts that are doing simple (or repetitive) jobs can be simply included to other scripts, they do not need to be implemented all over again.

The easiest way to include a resource to the Script Editor is using drag-and-drop operation from the Zagreus Browser window, see → Drag-and-drop operations .

8.3.1 Including connections

Connection resources contain a specific connection action with properly set up parameters so that the Zagreus System can connect to a particular server (database, ftp, MSTR server, etc.), see → Connections

An already existing and configured connection resource can be included into a script that can use this connection in its actions. *Figure 32.* shows the content of a connection resource that is included into a script in *Figure 33*. The *filename* attribute in the `z:include` action can be a resource path or a resource id as well.

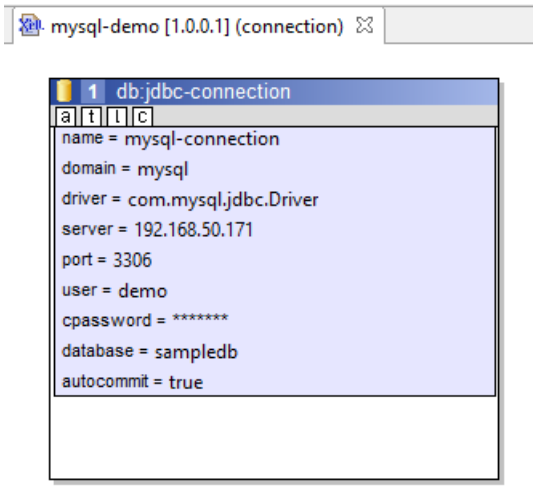


Figure 32 – A connection resource contains a single connection action with parameters

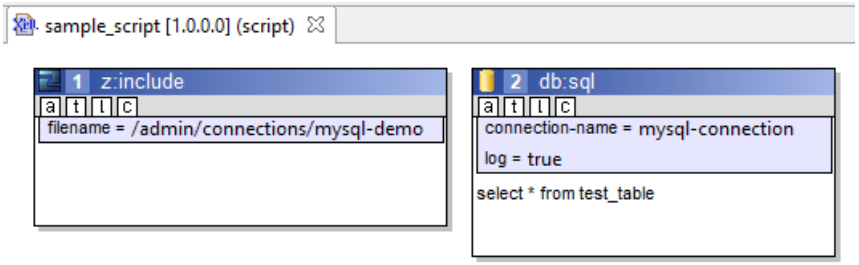


Figure 33 – The connection resource is included with a `z:include` action and used in the `db:sql` action

Actions that are using the included connections have to refer to the included connection action by its name. In *Figure 32.*, the name of the `db:jdbc-connection` action is *"mysql-connection".* In the script in *Figure 33.,* the `db:sql` action is referencing the connection action by the attribute *connection-name="mysql-connection".* For further details, see → Referencing to a connection .

8.3.2 Including templates

Template resources are used as containers for storing one or more `z:template` actions (see → Templates), so that they can be reused from multiple scripts as includes. The example of *Figure 24.* is changed to demonstrate this use case: *Figure 34.* shows the content a template resource, which is included by the `z:include` action in the script in *Figure 35*.

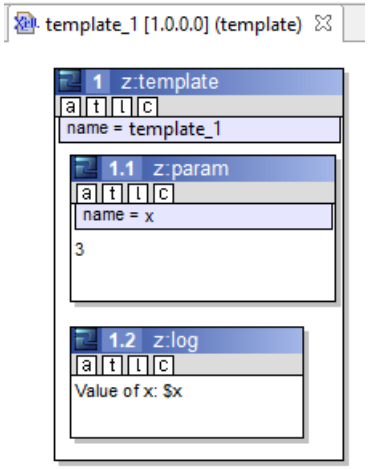


Figure 34 – The template resource, containing a `z:template` action

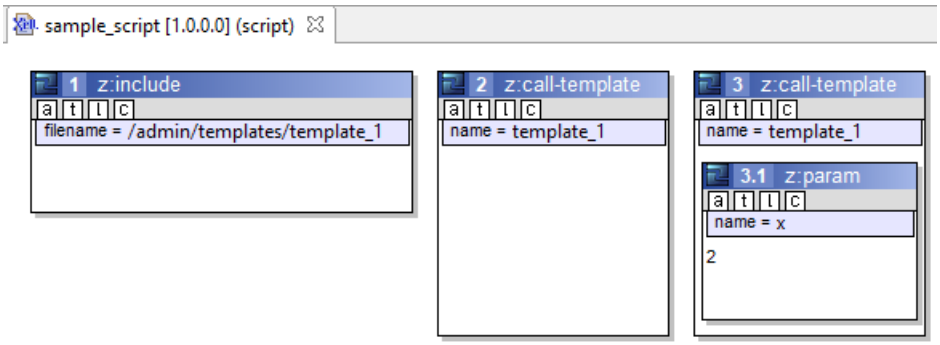


Figure 35 – The script uses the included `z:template`

8.3.3 Including scripts

Script also can be included into other scripts. Contrary to the cases of including connections or templates, included scripts are executed at the point of inclusion. The script in *Figure 36.* is included into the script shown in *Figure 37*.

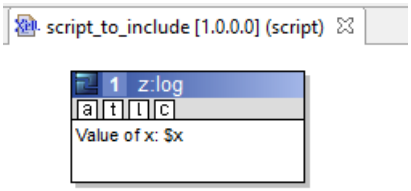


Figure 36 – The script to be included

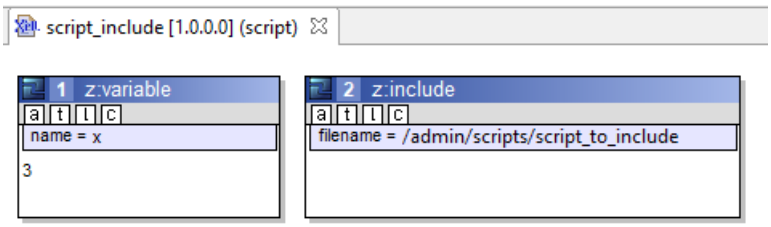


Figure 37 – The script including another script

The log result of executing the script in *Figure 37.* is the following:

```
<"1" z:variable>
</"1" z:variable>
<"2" z:include>
<"2.1" z:log>
Value of x: 3
</"2.1" z:log>
</"2" z:include>
```

The included `z:log` action was executed as the child element of the `z:include` action. This also means that variable *$x* in the `z:log` action of the *script_to_include* script was treated as a local variable in *script_include*.

Info: *The included action ordering numbers are changed to be matched to the ordering number of the corresponding* `z:include` *action. Included actions behave as direct children of the* `z:include` *action.*

8.4 Error handling

Errors (or technically: exceptions) are thrown from any action which has unexpected problems during execution. This includes missing attributes, runtime errors, broken connections, etc.

If the error is not handled in the action where it occurred, it will be propagated up to the parent action. This can continue for several levels. If the error is not handled at all, the script will be finished with an Error status (see also → Job lifecycle).

Zagreus has two ways to handle possible errors:

- specifying one of these attributes: on-error-next-sibling, on-error-next-child
- having a z:on-error action

In the following, we will demonstrate error handling by a file:dir action with an incorrect *path* attribute value, which throws an error.

8.4.1 on-error-next-sibling attribute

The *on-error-next-sibling* attribute instructs the Zagreus processing engine to continue script execution on the next sibling action when an error occurs. In *Figure 38.*, the error is thrown in `Action 1.1`, which has the *on-error-next-sibling* attribute set. Due to this, the next action to be processed will be `Action 1.2` (i.e. `z:log`), which will be followed by `Action 2`.

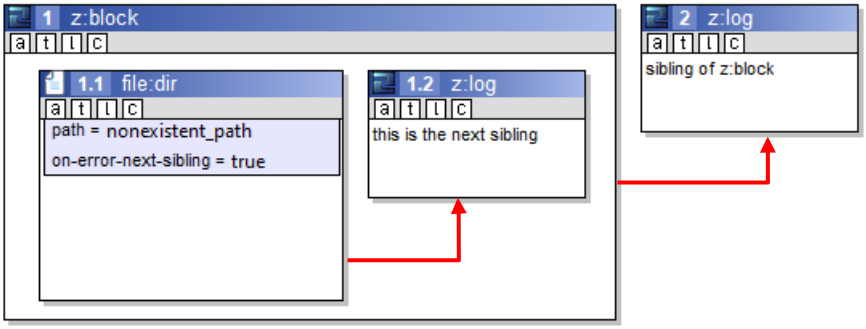


Figure 38 – An example for error handling with the *on-error-next-sibling* attribute

Recall that, if the error is not handled on the level it was thrown, it is passed to the parent action. In *Figure 39.*, the error is thrown in `Action 1.1`, but it will be handled by the parent action, i.e. `Action 1`. Because this action has the *on-error-next-sibling* attribute set, execution will continue with `Action 2`, and `Action 1.2` (along with all possible further child actions of `Action 1`) will be skipped.

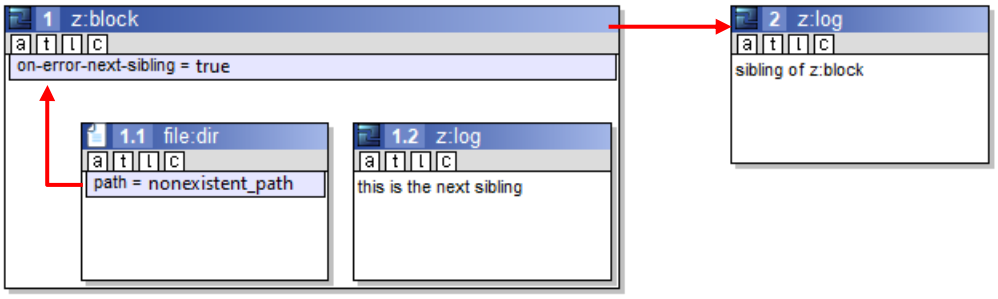


Figure 39 – Another example for error handling with the *on-error-next-sibling* attribute

8.4.2 on-error-next-child attribute

When the *on-error-next-child* attribute is specified, execution will continue on the next child action in the case of an error. This attribute is meant to be used in the parent action in contrast of using the *on-error-next-sibling* attribute. (This of course has no effect if the error occurred within the given action, as its child actions were already processed; for the order of action execution see → Order of execution, result flow). In *Figure 40.*, the error is thrown again in `Action 1.1`, but it will be handled by the parent action, i.e. `Action 1`. Because this action has the *on-error-next-child* attribute set, execution will continue with `Action 1.2`, being the next child action of `Action 1`. After this, `Action 2` will also be processed. In this case, the order of action execution is the same as it was in *Figure 36*. In general, setting the *on-error-next-child* attribute for the parent action is essentially the same as setting the *on-error-next-sibling* attribute for all its children.

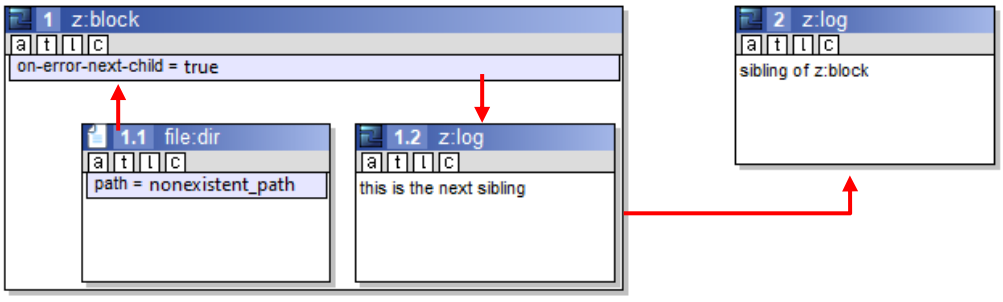


Figure 40 – An example for error handling with the *on-error-next-child* attribute

8.4.3 z:on-error action

The `z:on-error` action allows performing steps in the case of an error (e.g. logging, writing into a file, sending an e-mail etc.) by adding child actions to the `z:on-error` action. The error handling will be done on the parent level where the `z:on-`

`error` action was specified. Script execution will continue with the subsequent action on the parent level.

Figure 41. shows a simple example of error handling with `z:on-error`; similarly to the previous examples, the error is thrown from the `file:dir` action (now `Action 1.2`). In this case, the error is propagated up to and handled by `Action 1` (i.e. `z:block`). After logging the *"error caught!"* message, script execution will continue with `Action 2`, *"sibling of z:block"*.

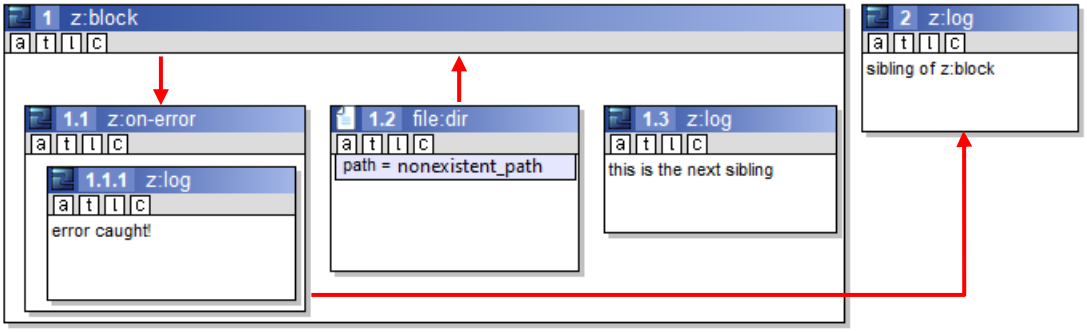


Figure 41 – Example for error handling with a `z:on-error` child action

Actions can handle their own exceptions as well: in *Figure 42.*, the `z:on-error` action is the child action of `Action 1.1`. So when an error occured in `Action 1.1`, it can handle it by its own `z:on-error` child action. Due to this, after logging the *"error caught!"* message (`Action 1.1.1.1` now), execution will continue with `Action 1.2` (and then with `Action 2`).

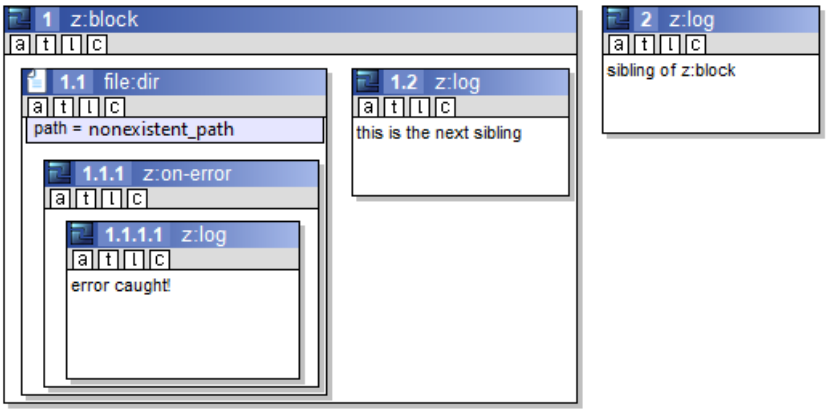


Figure 42 – log 1.1.1.1, then go on with 1.2 and 2

In *Figure 43.*, the `z:on-error` action is specified on the root level of the script. As the error is not handled by the `file:dir` action (i.e. `Action 2.1`), it will be passed up to the parent action (`Action 2`). As it is not handled there either, it will be passed up further to the root level, where it will be handled by the `z:on-error` child action.

In this case, both `Action 2.2` and `Action 3` will be skipped, so the script will finish after logging the *"error caught!"* message.

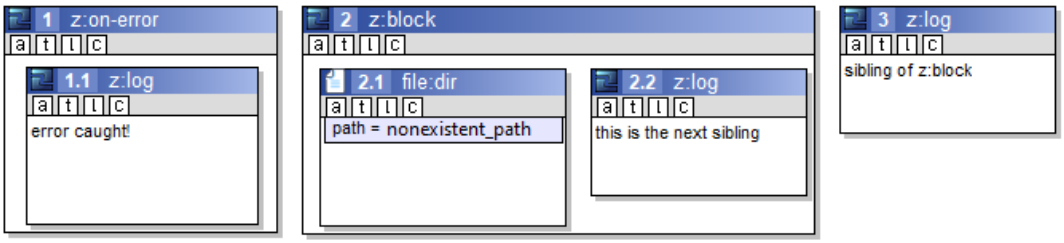


Figure 43 – Log "error caught" (in action 1.1), then finish the script

> ⚠ ***Warning:*** *The* `z:on-error` *action should be specified as the first action of a given level. If the action is not specified up to the point where an exception is thrown, the exception will not be caught, see Figure 44.*

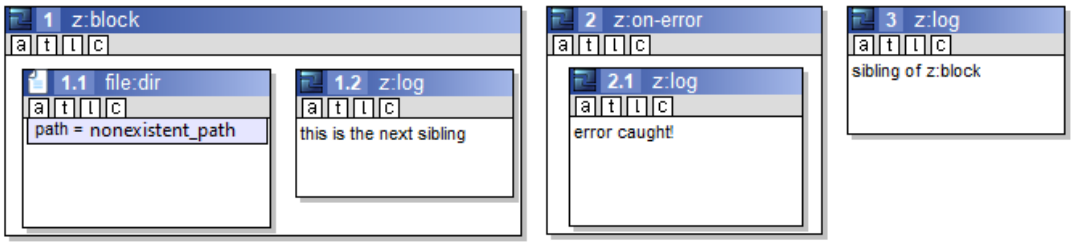


Figure 44 – The script is finished with error

> ⓘ ***Info:*** *The* `z:on-error` *action has an* error-name *attribute, which allows to filter exceptions based on the type of the error (e.g. AttributeRequired, IO, FileNotFound, JSON, MSTR).*

8.4.4 *errorMessage* and *errorTrace* variables

In case of any error, two variables are automatically set during the script execution: *errorMessage* and *errorTrace*. These two variables are both set in global scope, see → Global scope. They appear as result attributes as well, with the same names. *Figure 45.* demonstrates the logging of all these variables and result attributes.

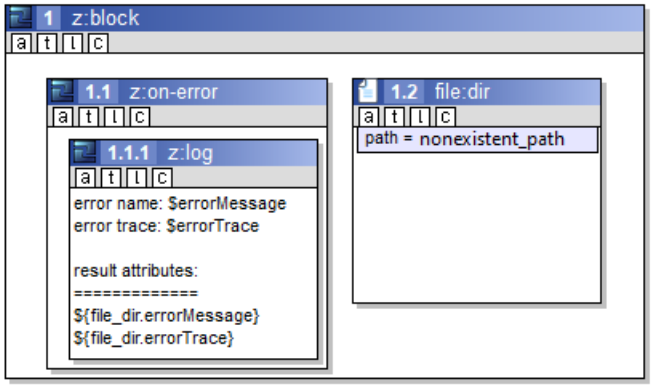


Figure 45 – Using *errorMessage* and *errorTrace*

8.4.5 *z:raise* action

There might be cases when the user intentionally wants to throw (or raise) an error. It works just as in programming languages: throwing an error stops the normal execution flow and an error handler (if any) can catch it and continues the execution. For an example, see *Figure 46*.

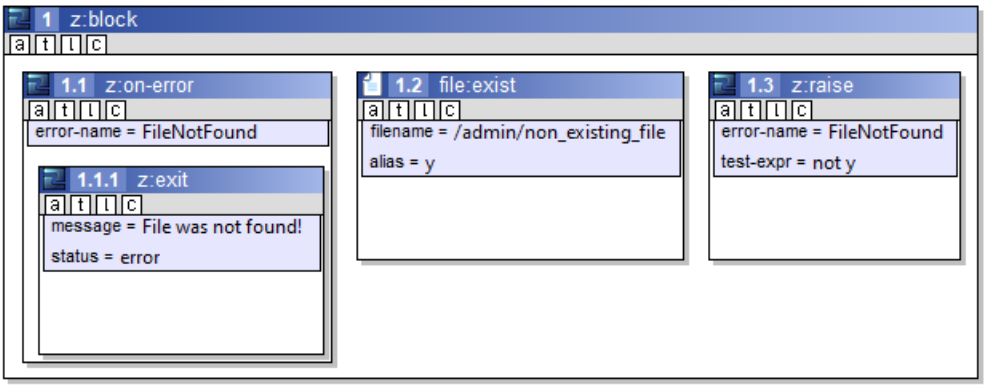


Figure 46 – Using the `z:raise` action

8.5 Variables

Variables are basic concepts of the Zagreus System, they are data values which can change or can be changed over time. The variables in Zagreus consist of two parts: *name* and *value*. Names are strings which are accepted as variables in the standard programming languages (i.e. a sequence of lower- and uppercase letters, underscore (”_”) and digits, where the first character cannot be a digit). Zagreus variable names are case sensitive, i.e. `a` and `A` refer to two distinct variables. Variables have also types, see → Data types

A variable can be defined on many different levels (see → Declaration levels), but almost all the variable references are finally resolved and processed during the execution of a script. The result of a script or the flow of execution in a script can be based on variables entirely.

In this chapter, only the script level variables are discussed. For variable scopes, see → Start-up variables. On how to set variables on different levels, see → Server-level and queue-level variables, → Setting script variables and options, → Context menu of a user node and → Context menu of a group node.

Variables used in the scipt level has types, see → Data types

8.5.1 z:variable action

The most common way to define a variable is the `z:variable` action. The variable name can be specified by the attribute *name*, and the value is inside the action. *Figure 47.* shows an example of such a definition and usage. Notice the variable reference `$x`, see → Engine expressions.

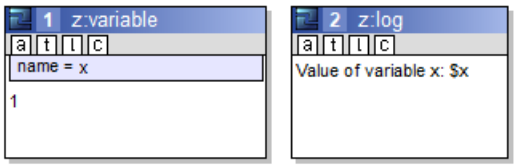


Figure 47 – Defining a variable with the `z:variable` action

Variables can also be used in action attributes. *Figure 48.* displays an example for referencing variable `x` in `Action 2`.

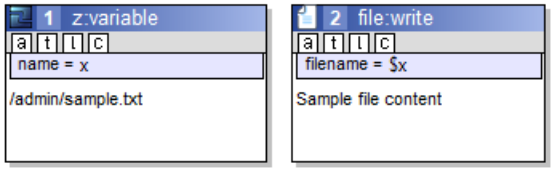


Figure 48 – Referencing a variable in an attribute

8.5.2 Variable scopes

Once a variable is specified, it can be referenced by its name. Variable scope is a specific region in the script where this reference is valid. Zagreus uses only two scopes: *local* and *global*. In action `z:variable`, scopes can be specified manually (by using the *scope* attribute).

Variables are mapped to the action within they are created. When a reference is used, Zagreus tries to resolve the variable by checking the already mapped variables of the parent action. If it does not succeed, it goes up to the parent action again in a recursive manner. In *Figure 49.*, variable x is mapped to `Action 1` (since z:variable is the child action of `Action 1`). When Zagreus is trying to resolve the variable reference in `Action 1.2.1.1.1`, it first checks `Action 1.2.1.1`, whether it has a mapped x variable. Because it is not the case, it goes up one level to `Action 1.2.1`, …, until it finds this variable mapped in `Action 1`.

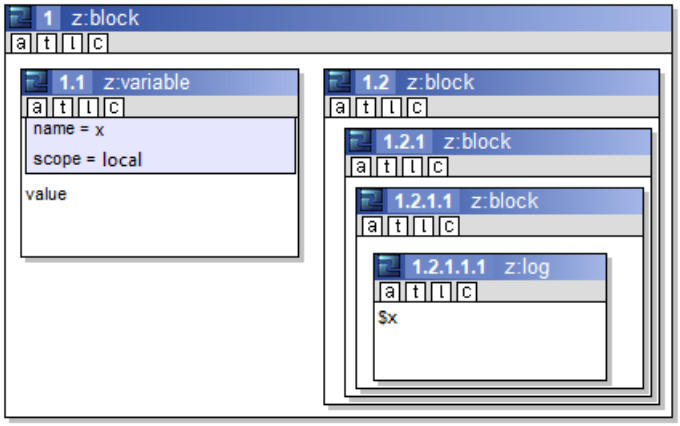


Figure 49 – Example for variable referencing

8.5.2.1 Local scope

Variables with local scope can be referenced only inside the container action the variable has been defined in. Outside of this parent action, the variable cannot be referenced. *Figure 50.* shows an example such a variable and an embedded reference to it.

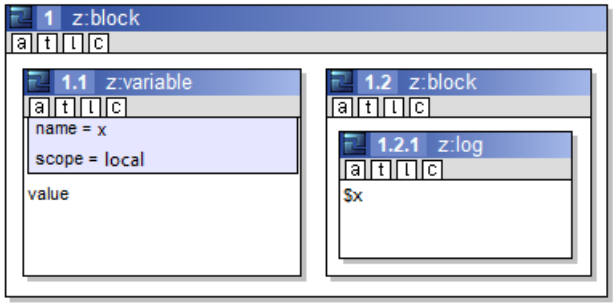


Figure 50 – Local scope variable can be referenced anywhere inside the same container action

A local scope can temporarily override another local scope which is located outside. In *Figure 51.*, two variables are specified with the name `x`, in different local scopes. Referencing for $x in `Action 1.2.2` will be resolved as *"second"*, while referencing in `Action 1.3` will be resolved as *"first"*. This is due to the fact that the variable defined in `Action 1.1` is mapped to `Action 1`, while the variable defined in `Action 1.2.1` is mapped to `Action 1.2`.

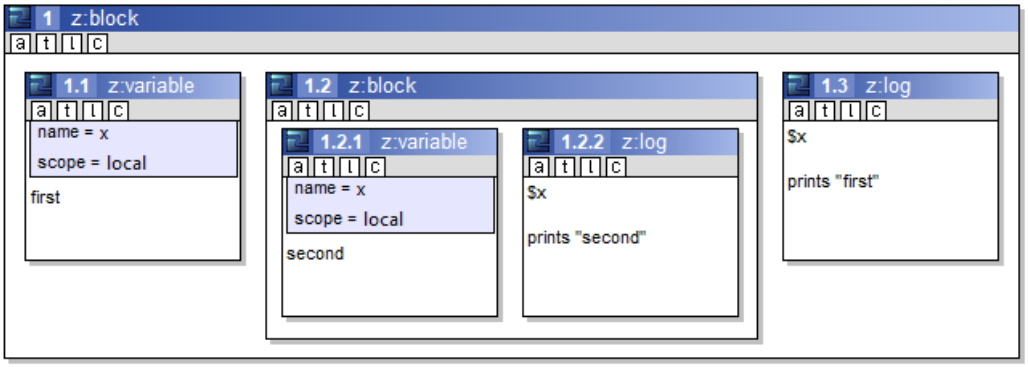


Figure 51 – Local scope variable can be referenced anywhere inside the same container action

8.5.2.2 Global scope

Variables with global scope can be referenced from all parts of the whole script after variable creation. Any variable defined outside of the script (e.g. server variables or script variables, see → Server-level and queue-level variables and → Setting script variables and options) are on the global scope per default. Inside the script, the *scope* attribute of the `z:variable` action can be used to specify a global scope variable. In *Figure 52.*, global scope variable `x` can be referenced outside of `Action 1`.

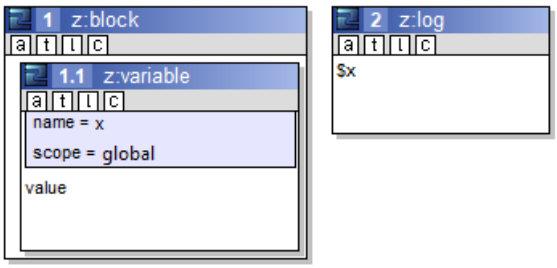


Figure 52 – Referencing a variable in an attribute

8.5.3 Monitoring variables

The `z:variable` action can have a special attribute *monitoring*. If this attribute is set to *TRUE*, the variable can be monitored in real time for running jobs. These *monitoring variables* can be checked on the *Monitoring variables* tab in the *Job properties dialog* in the Zagreus Monitor application, see → Job properties dialog.

8.5.4 Common attributes that create new variables

There are common attributes that also create variables in Zagreus System:

- *alias* and *alias-global*
  These attributes can control the result flow (see → Result flow) by storing the result of the action in the variable name (that was specified in the *alias* attribute).

- *result-attributes* and *result-attributes-global*
  These attributes belong to the result-flow monitoring (see → Result attributes) by storing the result attributes of the action in the variable name (that was specified in the *result-attributes* attribute).

- *action-attributes*
  This common attribute creates a variable with the specified name. After the action has been executed, this variable contains the original attribute values of the particular action, see *Figure 53*.

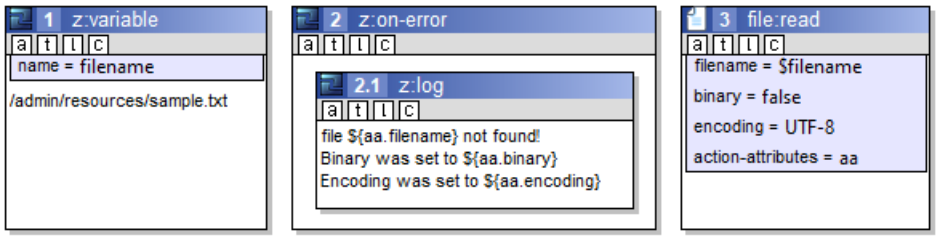


Figure 53 – Using the *action-attributes* attribute

8.6 Engine expressions

Engine expressions are special expressions in a Javascript-like language, which offer a wide range of processing functionalities. Every expression has an evaluated return value.

They can be located in two places: firstly, in the attributes of the actions, and secondly, in the text content of the actions. By default, these contents are handled simply as strings; for example, a `"hello"` text content inside a `z:log` action simply logs the text `hello`. There are three ways to interpret a string as an engine expression:

- By simply using a `$` sign before a variable name; i.e. `$a` means the value of variable `a`. For example the `"value of a=$a, value of b=$b"` text content inside a `z:log` action logs the values of variables `a` and `b`.
- By encapsulating the engine expression in a `${...}` block; i.e. `"${a + 10}"` will result in the value of the `a` numeric variable, increased by 10.
- By adding the `"-expr"` suffix to some specific attribute name (e.g. `test-expr` instead of `test`). Setting the `"test"` attribute to `"${a + 10}"` and setting the `"test-expr"` attribute to `"a + 10"` will lead to the same result.

Note: to add a `$` character to a simple text, `$$` should be used; e.g. `incoming tax: $value$$`.

Info: *In the following examples, remarks will be indicated with the `//` characters at the beginning of the lines. However, Zagreus does not handle these remarks in scripts.*

8.6.1 Basic expressions

Basic expressions can be the following:

- variables
- numeric literals, e.g.: `1`, `-2`, `0.6`, `-1.2e-30`
- string literals: `"hello"`, `'hello'`

Inside the string literals which are specified inside *double* quotes, it is possible to use special characters using the `\` escape character. The available constructions are:

- `\n`: new line
- `\r`: carriage return
- `\t`: tab character

- \': the ' character
- \": the " character
- \\: the \ character

For example, the literal "first\tsecond" leads to a tabular character between the two words, while the literal 'first\tsecond' leads to a backslash (i.e. \ ) and a t character between the words first and second.

8.6.2 Operators

Inside the engine expressions, the following operators can be used to construct more complex expressions:

- member access operators: . (dot), [] (for the details, see → Lists, records and tables)
- multiplicative operators: *, /
- additive operators: +, –
- relational operators: <, >, =, <=, >=, <> or !=, ~ (string matching regular expression), !~, in
- logical operators: and, or, not
- assignment operator: :=

The operators are processed in a standard priority order, i.e. * is processed before + and + is processed before <. Subexpressions using operators with the same priority are evaluated in direction from left to right, except the assignment operator (which is evaluated from right to left).

In the following examples all expressions are considered to be inside a ${...} block.

```
// variable "a" is assigned a value of 2.
a := 1 + 1
```

```
// variable "a" is assigned a text (containing a new line character).
a := "Zagreus\nAutomate your IT"
```

```
// the … block is executed if the logical condition is satisfied
if (a < b - 10 or a > b + 10) {
  …
}
```

8.6.3 Lists, records and tables

Lists (or indexed arrays) and records (or associative arrays, or maps) are data types which can contain any type of data. There is no type restriction for the stored elements, i.e. a list or record might contain numbers, texts and other lists at the same time.

Lists can be created by the `list([...])` function, and specific actions have lists as their result. Element access can be done by the `[...]` operator; e.g. `a[8]` refers to the 8th element of list `a`. Indexing is one-based.

```
// new list with different data types
l:=list([1, date, "apple"])
// displaying the current date
println(l[2]);
```

Records map its elements by keys. If no keys are defined, they are automatically created as numbers starting from 1. Records can be created either by the `record(...)` function, inline in an engine expression, and specific actions have records as their result. Element access is possible by specifying the corresponding key either by the `.` (dot) or the `[...]` operators; e.g. `a["x"]`, `a['x']`, `a.'x'`, `a."x"` and `a.x` all refer to the element of variable `a` mapped by the `"x"` key.

To create a record, use one of the following:

```
// empty record variable
a := [];
// referencing to a non-existing member will create this item
a.x := 1;
// new record, having keys of 1, 2 and 3
b := ["first", "second", "third"];
```

```
// new record with keys x, y
c := ["x":1,"y":2];
// accessing items of c:
cx1 := c.x;
cx2 := c['x'];
cy1 := c.y;
cy2 := c["y"];
```

```
// new record with function call: x:11, y:12, z:13, 1:14
d := record(['x','y','z'], [11,12,13,14])
```

Tables are special lists containing records with the same keys. One can see it as a two-dimensional matrix. Tables are returned by specific actions as their result (e.g. `z:parse`). For example, given a table `t`, `t[5].x` refers to the field `x` of row 5. Tables can be easily logged in a human-readable format, see → Script Logging.

8.6.4 Function calls

Engine expressions might include simple function calls. Parameters are passed within brackets (i.e. `(...)`), separated by commas (`,`). For example:

```
// a substring function call
s:=substring('this is a string', 6, 7)
// displaying "is"
println(s);
```

For a list of the supported functions, see → Variables / Functions window.

8.6.5 Expressions and statements

At its simplest terms, expressions are evaluated to produce a value. In contrast, statements are executed to make something happen. Statements can be also evaluated to return a value.

The following simple statements are supported in Zagreus:

- `if`: runs a statement if a given condition is true.
- `switch`: processes statements according to the value of a given expression.
- `for`: takes all items of the specified expression (e.g. list) and processes the given statement with these items (practically a foreach loop). When iterating through a record, it processes the values and ignores the keys.
- `while`: executes a statement while the given condition is true.
- `break`: stops the execution of the current `while`, `for` or `switch` statement.
- `continue`: skips the remaining part of the loop and goes to the beginning of the current `for` or `while` statement.

Examples:

```
// store the absolute value of x in x
if (x < 0) {
  x := x * -1;
}
```

```
// store the signum of x in x
if (x < 0) {
  x := -1;
} else {
  x := 1;
}
```

```
// store the text representation of x in y
switch (x) {
  case 1: y := ”one”; break;
  case 2: y := ”two”; break;
  case 3: y := ”three”; break;
  default: y := ”other”;
}
```

```
// add up the values stored in x
var act_record := [ 2, 4, 6, 8, 10];
sum := 0;
for (item in act_record) {
  sum := sum + item;
}
```

```
// example for a while loop
var act_record := [ ”a”: 2, ”b”: 4, ”c”: 6, ”d”: 8, ”e”: 10];
counter := 0;
while (act_record .b > 0) {
  act_record .b := act_record .b - 1;
  counter := counter + 1;
}
println(’iteration count: ’ + counter);
```

```
// printing even numbers
var counter := 0;
while (counter < 20) {
  counter := counter + 1;
  if (counter % 2 = 1) continue;
  println(counter);
}
```

```
// adding up numbers until they exceed 10 (i.e. 12)
var act_record := [ 2, 4, 6, 8, 10];
sum := 0;
```

```
for (item inlist act_record) {
  sum := sum + item;
  if (sum > 10) break;
}
println(sum);
```

```
// printing numbers until the first negative one
var act_record := [ 2, 4, -6, -8, 10];
for (item inlist act_record) {
  if (item < 0)
    break;
  println(item);
}
```

8.6.6 Data types

The values of variables or the result of the actions can be of different types. Most data types can also be created within engine expressions; others are just returned by specific Zagreus actions as their results or as their result attributes. The type of a Zagreus variable can be determined by using the `gettype(...)` function.

Data types also fall under some categories. These are:

- *Comparable*
  The comparator operators (e.g. `<`, `>`, `!=`, `<=` etc.) can be applied to them.

- *Concatable*
  The values can be concatenated, primarily by the `concat(...)` function. For example, concating the texts `"Zag"` and `"reus"` results in the text `"Zagreus"`, while concatenating two lists results in a list containing the contents of both original lists.

- *Formatable*
  They can be formatted with the `format(...)` function.

- *Mergeable*
  The contents can be merged by the `z:union` action.

- *Iterable*
  The elements stored inside the variable can be iterated over, for example by a `for(...)` engine expression (see → Expressions and statements) or a `z:foreach` or `z:for` action (see → z:for action and → z:foreach action).

8.6.6.1 Simple data types

Most data types in Zagreus are simple ones; next we describe them in details.

- *Text*
  { Comparable, Concatable, Formatable, Mergeable }
  Texts are string, i.e. character sequences. They can be created in engine expressions with the format `"Zagreus text"` or `'Zagreus text'`; the difference among the two syntaxes is that the former one resolves certain specific characters such as `\n` or `\t`. Data belonging to other types can be converted to text by using the `text(...)` function. The `istext(...)` function returns true iff the type of its parameter is text. Texts can be concatenated with the `+` operator (along with the straightforward `concat(...)` function). There are several text functions in Zagreus, the *Variables / Functions* extension window in the Zagreus

Client lists them and shows their help in the tooltip texts, see → Variables / Functions window .

- *Number*
  { Comparable, Formatable }
  Numbers hold numeric values; internally they are stored as floating-point numeric values (i.e. doubles). They can be created in engine expressions with the formats `1867`, `186.7` and `1.867e03` (i.e. scientific notation). Data of other types can be converted to number by using the `number(...)` function. The `isnumber(...)` function returns true iff the type of its parameter is number.

- *Boolean*
  { Comparable }
  Booleans are a binary type, they can either be `true` or `false`. They can be created in engine expressions in the formats `true` and `false`, and as a result of several comparison operators (e.g. `"dog"!="cat"` will result in the value `true`). In attributes expecting a boolean value, the forms `yes` and `no` can also be used. Data of other types can be converted to boolean by using the `boolean(...)` function; the number `0` will be converted to `false` and all other numeric values to `true`; the text `""` (i.e. empty string) will be converted to `false` and all other textual values to `true`. The `isboolean(...)` function returns true iff the type of its parameter is boolean.

- *Date*
  { Comparable, Formatable }
  Dates are represented internally as Unix timestamps (i.e. the number of milliseconds passed since 1st of January, 1970, 0:00 GMT). Data of other types can be converted to date by the functions `date` (representing the actual date at the function call), `date(datevalue)` and `date(datevalue, formatstring)`. If the parameter is a text, and no format string is specified, the format `"yyyy-MM-dd"` will be used. Besides text values and other dates, numbers can also be used (and will be interpreted as Unix timestamps). The `isdate(...)` function returns true iff the type of its parameter is date.

- *BinaryData*
  { Comparable, Concatable }
  Binary data are mainly returned by specific functions. Data of other types can be converted to binary by the `binary(...)` function. The `isbinary(...)` function returns true iff the type of its parameter is binary.

- *Null*
  { Comparable, Formatable }

Nulls are representing null objects. They can be created in engine expressions by the token `null`. The `isnull(...)` function returns true iff the type of its parameter is null.

- *XML*
  XMLs represent XML data. They can be created in engine expressions by the function `xmltype`: other than other XML data, only `text` input is supported at the moment. The `isxml(...)` function returns true iff the type of its parameter is XML.

8.6.6.2 Extended data types

Some data types in Zagreus are extended ones, which means that they inherit most of their properties of another Zagreus type. Besides sharing several details, the most important consequence is that they can also be considered as their basic data type; for example, since the *time* data type extends the *date* data type, the `isdate(...)` function will return true for a variable with a type of time as well.

- *Time*
  { Comparable, Formatable }
  Time data types extend over Date. The main difference is that their default format is of `"HH:mm:ss.SSS"`. Data of other types can be converted to time by the functions `time` (representing the actual time at the function call), `time(timevalue)` and `time(timevalue, formatstring)`. If the parameter is a text, and no format string is specified, the format `"HH:mm:ss.SSS"` will be used. Besides text values and other dates (including times), numbers can also be used (and will be interpreted as Unix timestamps). The `istime(...)` function returns true iff the type of its parameter is time; furthermore, as time extends the date type, the `isdate(...)` function also returns `true` if its input is of type time.

- *PDF*
  { Comparable, Concatable }
  PDFs represent binary data with a type of PDF. Due to this, they cannot be created inside engine expressions, they are only returned by specific actions.

- *Excel*
  { Comparable, Concatable }
  Excels represent binary data with a type of Excel workbooks. Due to this, they cannot be created inside engine expressions, they are only returned by specific actions.

8.6.6.3 Compound data types

The following, aforementioned data types contain instances of other data types. See also → Lists, records and tables.

- *Record*
  { Comparable, Concatable, Formatable, Iterable, Mergeable }

- *List*
  { Comparable, Concatable, Formatable, Iterable, Mergeable }

- *Table*
  { Comparable, Concatable, Formatable, Iterable, Mergeable }

8.7 Script Logging

In this chapter, the practices for logging inside a script is covered.

8.7.1 job-log file

Script log messages are stored in a file after (and during) execution. These files are called job-log files, their names are derived from the job ID of the execution, see → Job properties.

These log files can easily be checked in many ways: double-clicking the job in Finished jobs window (see → Finished jobs window), checking Active logs (see → Active logs window) or manually check the logfile in the server subfolder (see → General properties).

In the job-log file, there are messages from the execution engine itself and also, the user-initiated log messages are shown here. A typical log file starts with similar messages as below:

```
Execution started on script "test"
Zagreus version: 1.5.5.7
Job ID: eae00564-850b-4404-ac31-4bb25772e70e

Job starting variables:
variable_name     variable_value
callerType  gui
currentUserId     1
executingUserName admin
executionMode     direct
…
Default encoding: UTF-8
Default locale / country: United States
Default locale / language: English
```

The Execution Engine is listing the start-up variables of the script (see → Start-up variables). After the variables and the locale settings, the script execution messages are listed. For example:

```
<"1" z:log>
enter log message here
</"1" z:log>
<"2" z:if>
<"2.1" z:then>
</"2.1" z:then>
<"2.2" z:else>
</"2.2" z:else>
</"2" z:if>
<"3" z:list>
</"3" z:list>
Execution finished on script "test"
```

Per default, the Execution Engine logs all the action entering and exiting events. If the user does not want to see these messages, a *loglevel* setting can be changed to 'user' or 'error' or 'warning', see → Logging levels and loglevel.

The job-log messages are meant for being checked by the user who can see them in the *Active Log* window and in the *Finished Log* window in the Zagreus Client (see → Active logs window and → Finished logs window).

8.7.2 *z:log* action

The simplest way to log a message is to use the `z:log` action. In the text content of this action, variables, function calls and complex engine expressions can all be used. In *Figure 54.*, `Action 1` logs a simple message while `Action 2` is using the *$date* variable reference which is resolved into the current date in the resulting log message.

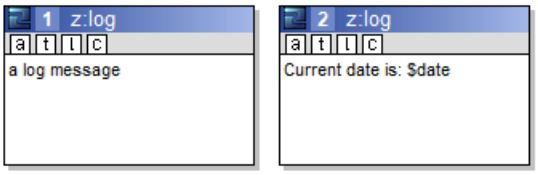


Figure 54 – Examples for the `z:log` action

The `z:log` action can also handle more complex inputs, such as the result of a child action. *Figure 55.* shows an example for logging the result of the `file:read` action, i.e. the content of *sample.txt*.

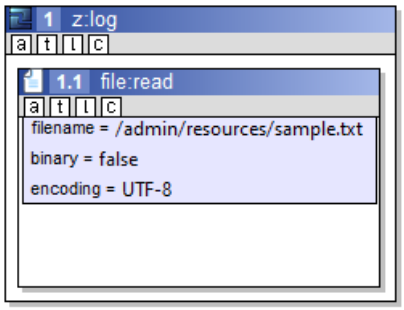


Figure 55 – Logging the result of a file:read action

8.7.3 *log* attribute

Since logging the results of an action is a very common use case, Zagreus provides a simple common attribute for this purpose. *Figure 56.* shows an equivalent example as in *Figure 55.*, using the *log* common attribute instead of a container `z:log` action.

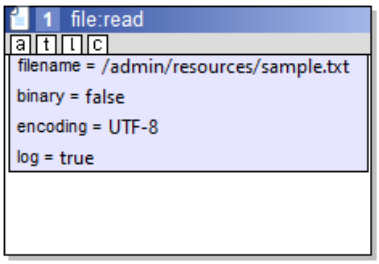


Figure 56 – Logging the result of a file:read action using the *log* common attribute

8.7.4 Logging levels and loglevel

It is important to distinguish between the concept of *logging level* and *loglevel*.

Logging levels are used to categorize log messages based on their severity. Different logging levels represent different levels of severity or importance of the logged messages. Zagreus defines the following logging levels:

- *user*
  user-intended messages such as a `z:log` action or the *log* common attribute.

- *error*
  error messages and the corresponding stack-traces generated by the Zagreus engine.

- *warning*
  warning messages generated by the Zagreus engine.

- *info*
  normal messages in the Zagreus engine. For example: entering and exiting actions, starting and finishing processes, opening connections.

- *debug*
  special debug messages.

Loglevel, on the other hand, is the setting which controls which log message categories will be written into the log file. The value of this setting is one of the logging level values listed above. Hence the list of the logging levels is a priority list as well, the *loglevel* setting works as the following:

- *loglevel*=user
  Only the user logging level log messages are logged into the logfile.

- *loglevel*=error
  User and error logging level log messages are logged into the logfile.

- *loglevel*=warning
  User, error and warning logging level log messages are logged into the logfile.

- *loglevel*=info
  User, error, warning and info logging level log messages are logged into the logfile.

- *loglevel*=debug
  Every log message is logged into the logfile.

The loglevel is set by an execution option called *log_level*. The default value of this option is *info*, but the user can override it by specifying the *log_level* option, see → List of execution options).

Inside the script the default *loglevel* can be changed with the `z:loglevel` action or with the *loglevel* attribute.

It is important to understand the difference between logging levels and *loglevel*. Logging levels are the categories for the messages and *loglevel* is a filter during execution that controls which messages are shown in the logfile.

Info: *The default logging level of user-logged messages is ‘*user*’. Only the* `z:log` *action can override it with the* level *attribute. So the user can also generate log messages on other levels than ‘*user*’, e.g. ‘*error*’.*

8.7.4.1 z:loglevel action

The `z:loglevel` action sets the *loglevel* of the script from that point of the execution flow onwards. In *Figure 57.*, the `z:loglevel` action switches the default ’info’ loglevel to ’user’. This means that, from that point on, only the ’user’ logging level messages are shown in the logfile. The message in the `z:log` is still visible, because the default level of `z:log` (and all other user-initiated log messages) is ’user’.

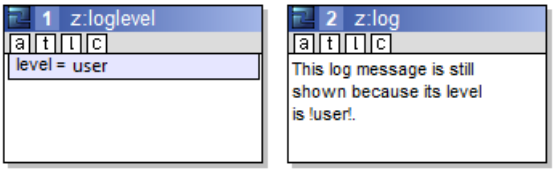


Figure 57 – Example for setting the loglevel to ‘user’ with the `z:loglevel` action

On the contrary, in *Figure 58.* the message in the `z:log` action is not shown, because the intended logging level of that is ‘info’, set by the *level* attribute. The *loglevel* is set to ‘user’, which means that the ‘info’ logging level messages are not shown in the logfile. See → Logging levels and loglevel.

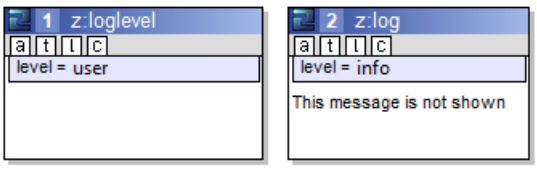


Figure 58 – Example for a log message that is not shown because of its logging level

8.7.4.2 *loglevel* attribute

The *loglevel* common attribute sets the loglevel for a specific action and its child actions. Unlike the `z:loglevel` action, the loglevel will be reset to the default loglevel (or the one set in the context) after this action is executed. In *Figure 59.*, the loglevel of the `z:block` action is set to ’user’, so only `Action 1.2` is logged according to the priority order of the logging levels (see → Logging levels and loglevel).

Figure 59 – Setting the loglevel for a `z:block` action with the *loglevel* attribute

8.7.5 z:logfile action

Zagreus job-logfiles are stored in a server subfolder, see → General properties. The user can redirect the log messages to another file at a certain point during the execution with a specific action `z:logfile`.

In *Figure 60.,* `Action 2` redirects the logging into a new file (*newlogfile.txt*), so the log message in `Action 3` will be present in this file. `Action 4` resets this redirection with the empty *filename* attribute, so the message in `Action 5` is present in the standard job-log file again.

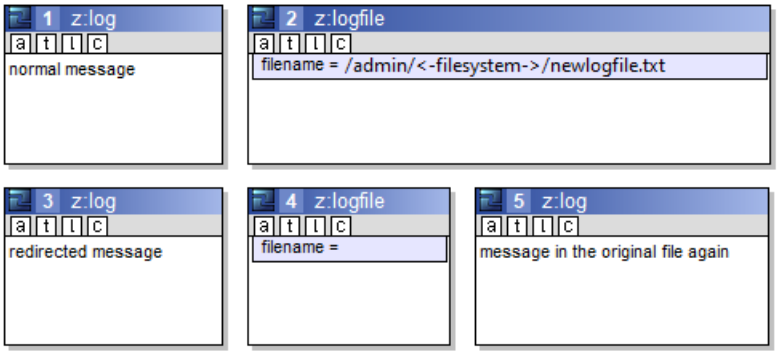


Figure 60 – Redirecting logging to another file temporarily with the z:logfile action

8.7.6 *logfile* attribute

The special attribute *logfile* also redirects the logging into another file, but unlike the `z:logfile` action, this attribute applies only for the given action and its sub-actions (no need to reset this redirection after the action).

In *Figure 61.*, this attribute is set for `Action 2`, and the logging for `z:block` and all of its child actions is redirected to the new logfile (*newlogfile.txt*).

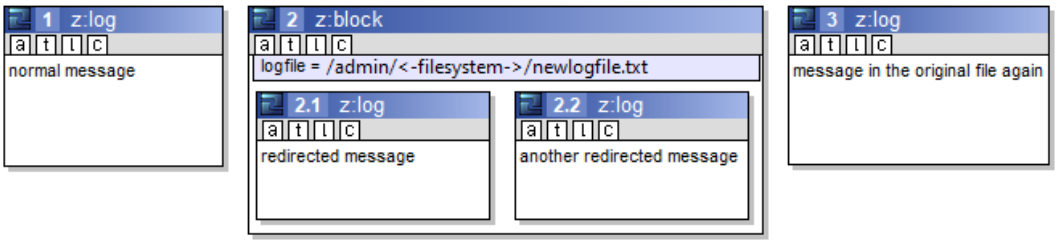


Figure 61 – Redirecting logging to another file temporarily with the *logfile* attribute

8.7.7 log-attributes attribute

The *log-attributes* attribute is used for logging all name-value pairs of the attributes of a particular action.

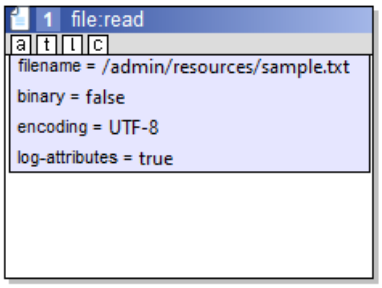


Figure 62 – Example for the *log-attributes* attribute

In the logfile, the attributes are listed like below:

```
<"1" file:read>
Attribute: filename, value: /admin/resources/sample.txt
Attribute: binary, value: false
Attribute: encoding, value: UTF-8
Attribute: log-attributes, value: true
Reading file "/admin/resources/sample.txt"
</"1" file:read>
```

8.7.8 log-result-attributes attribute

The *log-result-attributes* attribute logs all the result attributes (see → Result attributes) of the action after execution. In *Figure 63.*, the result attributes of the `z:for` action (which are: *loopcnt* and *execution_time*) are logged into the logfile shown as below:

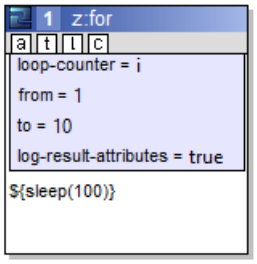


Figure 63 – Example for the *log-result-attributes* attribute

```
<"1" z:for>
[loopcnt:10,execution_time:1003]
</"1" z:for>
```

8.8 XML representation

Scripts are stored in XML format in the embedded MySQL database of the Zagreus Server. This XML format corresponds to the one shown in the fully-featured Script Editor, see → Script Editor. The user can check out the XML representation in the Zagreus Client in the following ways:

- selecting the XML view in the Script Editor, see → XML view
- opening the script by the XML Editor from the Zagreus browser window, see → Opening resources

> ⚠ ***Warning:*** *It is highly recommended not to manually edit the XML content of the script. The main purpose of checking out the XML representation is debugging.*

In the next example a simple script is shown:

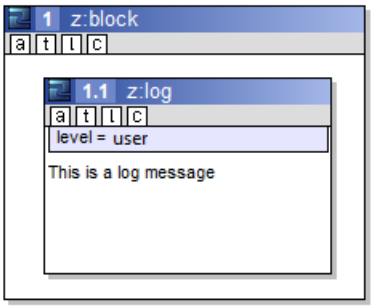


Figure 64 – An action containing another action

The XML representation of the script in *Figure 64.* is shown in the following snippet:

```
<z:block _z="0" _o="1" _x="28" _y="19" _w="231" _h="189" _v="3">
    <z:log _z="0" _o="1.1" _x="20" _y="15" _w="185" _h="128" _v="3"
level="user" null-format="">This is a log message</z:log>
</z:block>
```

Notice the following details in the XML representation:

- Actions are represented by XML elements.
- Action name is represented by the local name of the XML element.
- Action group is represented by the namespace of the XML element.

- Child actions are represented by additional XML elements as the content of the parent XML element (such as the `z:log` child action).
- Action attributes are represented by the attributes of the XML element (such as the *level* attribute of the `z:log` action).
- Textual content of an action is represented by the textual content (PCDATA) of the XML element (such as the *"This is a log message"* string).
- Attributes of the XML element starting with *"_"* are strictly for internal usage.

9. Connections

In Zagreus, a connection is the collection of information required to access a (typically remote) server, such as protocol, Internet address, IP address, port, username, password, operation folder etc. Using connections is one of the key steps in creating a Zagreus script. In general, in a Zagreus script the users can access various kinds of data sources, import data and perform various operations on them. In addition to gathering, manipulating, and forwarding data, Zagreus can leverage multiple connections in a single script.

Here are two examples for using different connection types:

- An Excel file, received via e-mail in attachment is downloaded from an e-mail inbox *(IMAP connection)*. The data stored in the Excel file is read, processed and changed; this new data is written into a database (SQL) table *(database connection)*. The new Excel file, containing the modified data, is copied to a network drive *(file connection)*.

- MicroStrategy (MSTR) reports are collected over the course of a week *(MSTR connection)*, they are exported and the files are zipped and sent to a remote File Transfer Protocol (FTP) server *(FTP connection)*. The latest report and its summary is uploaded to a Confluence page *(Confluence connection).* Finally, a status report e-mail is sent to the co-workers *(SMTP connection)*.

By using Zagreus connections, data which is stored on different servers can be accessed by using different protocols (e.g. IMAP, FTP, SQL). This allows Zagreus users to automate various types of jobs, and develop multiple workfolws – as demonstrated in the above examples.

Currently the following connection types are supported by Zagreus:

- *confluence:connection*: for connecting to a Confluence server and reach Confluence spaces
- *db:jdbc-connection*: for connecting to a database server using jdbc database connection. Supported database environments are: Oracle, Mysql, SQL-Server, Teradata, db2, SQLite, PostgreSQL, Exasol.
- *file:connection*: points to a file in the Zagreus external file system

- *ftp:connection*: for connecting to an FTP server to manage, upload and download files
- *http:connection*: for connecting to an HTTP server
- *jira:connection*: for connecting to a Jira server and reach its projects
- *kafka:connection*: for connecting to a Kafka cluster to produce and consume messages, list topics, etc.
- *ldap:connection:* for connecting to an LDAP server
- *mail:connection*: for connecting to an e-mail server. Can be configured as an SMTP, IMAP or POP3 connection
- *msft:connection*: for connecting to a Microsoft account in order to access OneDrive, mails, files, etc.
- *mstr:connection*: for calling MSTR functions with the MSTR Java Web API
- *mstrrest:connection*: for calling MSTR functions via REST API
- *rest:connection*: for connecting to a REST API provider
- *ws:connection*: for connecting to a server in order to execute web service calls
- *zs:connection*: for connecting to a Zagreus server

9.1 Defining connections

Many types of connections are available in Zagreus for making automated processes – database, FTP, IMAP, SMTP, MSTR, REST, etc. The term *connection* is used in Zagreus in two different meanings: as a Zagreus connection *action*, and as a Zagreus connection-type *resource*. In the first sense, the connection action can be handled as any other Zagreus action: it can be drag-and-dropped from the palette (see → Palette) to the canvas area of the Script Editor (see → Canvas), its attributes can be edited, etc. A Zagreus connection resource refers to a special type of Zagreus resource (see → Resource types), which contains only one action, which is a Zagreus connection action.

Although defining a Zagreus connection is possible as a standalone action (see → Creating a connection resource) within a Zagreus script, it is recommended to define each connection as a standalone connection resource; this way, it can be included into several different scripts. Would some property of the connection change in time (e.g. changing an expired password), connection management is more straightforward and easier this way than editing all affected scripts individually.

9.1.1 Creating a connection resource

Perhaps the most straightforward way of creating a Zagreus connection resource in the Zagreus Client, in the Script editor window (see also → Script Editor → Script Editor), in the Graph Editor (see → Graph view). Similarly to other actions, connection actions can be drag-and-dropped from the palette (see → Palette tab), which can be followed by specifying the content of the action attributes.

To create a Zagreus connection, select the menu item *File / New resource*... , see *Figure 1*.

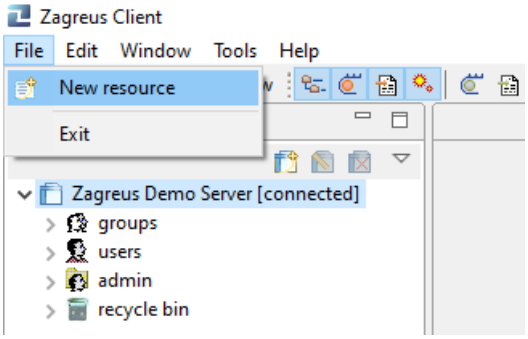


Figure 1 – Creating a new connection resource by using the main menu bar

Alternatively, the user can select the *Create new resource...* menu item from the context menu of the containing folder (see → Creating new resources), see *Figure 2.*

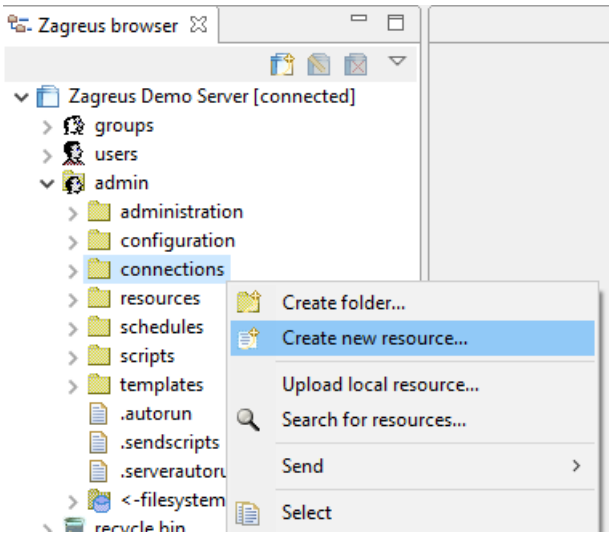


Figure 2 – Creating a new connection resource by using the context menu

Next, type of the new resource must be selected. The option *connection* can be found in the dropdown list, see *Figure 3.*

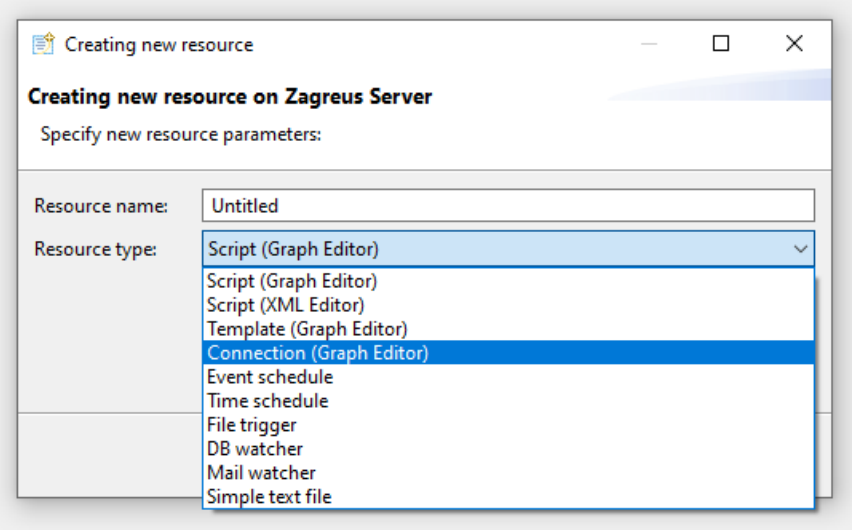


Figure 3 – Selecting the *connection* resource type from the dropdown list

After the new Script Editor tab is opened in the editor area, the particular connection action can be drag-and-dropped from the *palette* (→ Palette tab), see Figure 4.

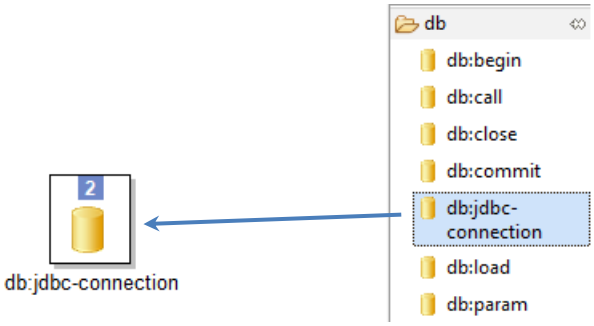


Figure 4 – Drag-and-drop the connection action from the palette

Finally, the attributes of the connection action must be filled; for a completed connection as an example, see *Figure 5.*

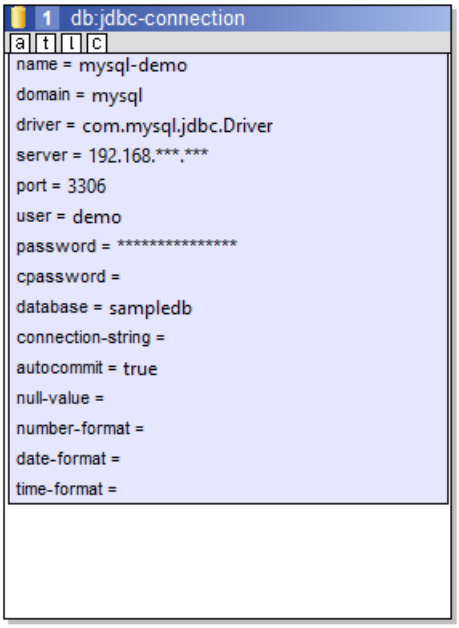


Figure 5 – An example of a completed connection

9.1.2 General connection attributes

Most of the connection attributes are unique and connection-specific; however, there are general attributes which are present for most connection actions (for example *name, server, username* or *password*). Next we will discuss the most frequent ones.

9.1.2.1 *name* attribute

The *name* attribute is used in all Zagreus connection action definitions. This will serve as a reference of the particular Zagreus connection for the other actions of the action group / namespace. It is recommended to specify it as a short, concise and easy-to-read expression. The *name* attribute is required to specify.

In most actions, the *connection-name* attribute is used to reference the given connection. As a notable exception, action `zs:migrate` (used to migrate resources across Zagreus servers) uses two Zagreus connections at once: *source-connection-name* and *target-connection-name*.

9.1.2.2 *server* attribute

This attribute is present for most Zagreus connection actions. It stands for the URL or the IP address of the target server. There are a few exceptions, which connections do not use the *server* attribute, such as the *Microsoft connection* (action

`msft:connection`), where the *authority* attribute identifies the target, or the *MSTR REST connection* (action mstrrest:connection), which has the *url* attribute for this purpose.

From this perspective, the *Zagreus Server connection* (action *zs:connection*) is a special connection. For this particular connection action, the *server* attribute can be left blank, which just indicates the *actual* Zagreus server, i.e. the hostname "localhost". However, if the local Zagreus server instance uses a different port than the default one (i.e. 7323), further configuration steps are required, see also → Connection properties.

9.1.2.3 *username*, *password* and *cpassword* attributes

The first step of a server-client communication is the authentication of the client who wants to perform some operations. Perhaps the most common way of authentication process is to use username and password, which values can be specified with the *username* and *password* attributes for most Zagreus connection actions, e.g. FTP, database, SMTP, etc.

If the user does not want to display the password as a readable data on screen (and store it in the Zagreus script as plain text), then encrypted password can be generated with the built-in *Password converter* tool (see also → Tools menu). In this case the attribute *cpassword* must be filled with this encrypted password, while the attribute *password* will be left empty.

9.1.2.4 Authenticating with tokens

In some cases user authentication is done by using tokens, i.e. by specific, generated identifiers, represented as character strings. For example, in the case of the Microsoft connection (action `msft:connection`), *Client id* and *Client secret* tokens have to be generated before creating the connection resource, and the values have to be copied into the attributes *client-id* and *client-secret* of the `msft:connection` action.

9.2 Using connections

The main benefit of using connection resources is the possibility to reuse them: several scripts can utilize the same connections, and the necessary changes (e.g. port, new password) have to be corrected in one file only. Zagreus connection resources have further advantages in terms of maintenance and administration (comparing to Zagreus connection actions defined inside scripts): the parameters of the connection can be verified easier, and connections can be drag-and-dropped from the Zagreus browser onto an action, allowing users to include the correct resource into the script with only a single mouse movement.

9.2.1 Test connection feature

It is recommended to *test* a Zagreus connection resource after it has been created, before using it. This practically means that the Zagreus server attempts to connect to the given server with the given parameters, defined in the corresponding Zagreus connection resource. This function can be accessed in the Browser Window of the Zagreus Client, by right-clicking the given Zagreus connection resource, and choosing the *Test connection* menu item from the context menu, see *Figure 6*.

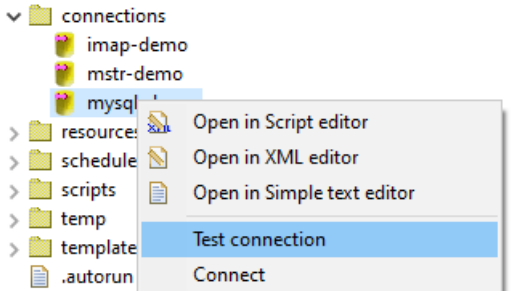


Figure 6 – Choosing the *Test connection* option

The result of a connection test can be *successful* or *failed*, and it appears in a pop-up window, see *Figure 7.*

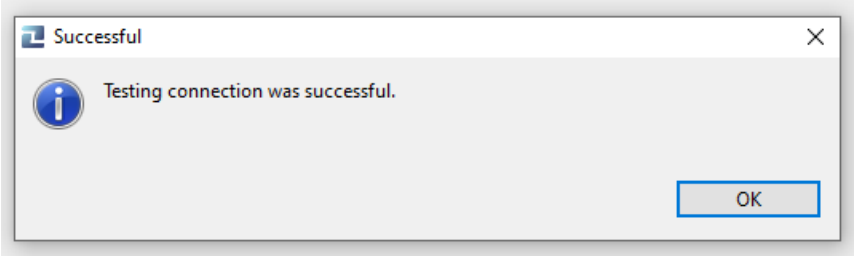


Figure 7 – Successful connection test message.

If a connection test fails, a brief error message informs the user about the cause of the failure; for an example, see *Figure 8*.

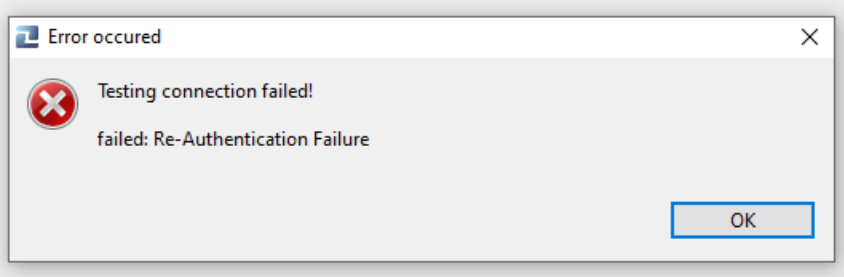


Figure 8 – Failed connection test with a specific error message.

When the server did not respond within 15 seconds during a connection test, the testing will fail as well (time out). That may indicates that connection definition contains an error, or the target server is unavailable. For an example, see *Figure 9*.

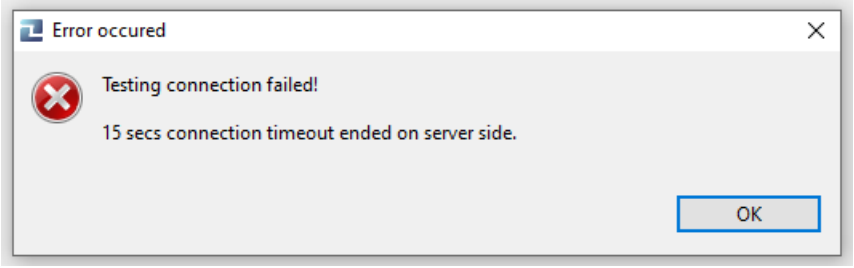


Figure 9 – Failed connection test due to time-out.

It is important to understand that the connection test is performed by the Zagreus Server module (see → Zagreus Server), while script execution (and therefore, actual connection usage) is done by a Zagreus Worker (see → Zagreus Worker). The configuration of these modules might be different, for example the firewall settings might affect the two differently (if they are installed on different machines), or they can differ in the SSL certificates installed (in case of secure connections, see → Manage certificates). If the connection test succeeds, but script execution fails for the same Zagreus connection resource, the reason is probably related to worker settings (see → Zagreus Worker configuration).

9.2.2 Referencing to a connection

If a script can use a Zagreus connection *action* (either because it was defined in the given script, or because it was included in the script via the `z:include` action (see → Includes), the corresponding actions can reference it. Of course, they must belong to the same action group: it makes no sense to reference an FTP connection from a

MicroStrategy-related action (e.g. `mstr:report`). Referencing is done by using the *connection-name* attribute in the referencing action. *Figure 10.* shows an example.

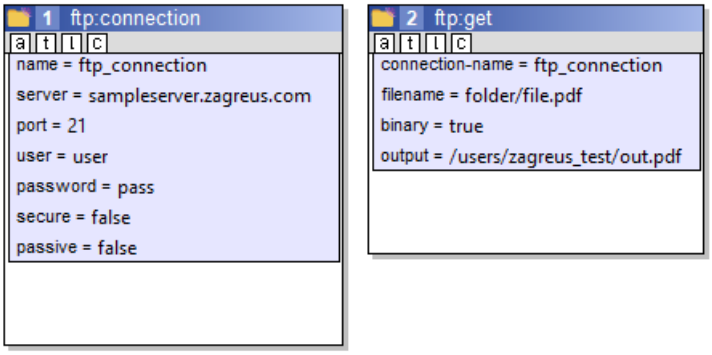


Figure 10 – Referencing a connection from another action

In this example, an FTP connection is defined in Action 1 (`ftp:connection`). Among the connection parameters, the name attribute has the value "ftp_connection", which will be the ID of the given FTP connection. Action 2 (`ftp:get`) uses this string in the value of the connection-name attribute to identify which connection action will be used in the actual FTP step.

> ***Info:*** *As a rule of thumb, a Zagreus connection action is just a definition of the given connection. Connecting to the specified server is performed when the first referencing action is executed.*

9.2.3 Inserting connections to a script

Instead of manually specifying a `z:include` action (see → Includes), connections can be included into a script with only a simple drag-and-drop event. This is done by dragging the resource in the Zagreus Browser window, and releasing it on the attribute *connection-name* of the action which should reference this connection (see *Figure 11.* and *Figure 12.*). This will have two effects: firstly, the Zagreus connection resource will be included in the given script (by adding a `z:include` action), and secondly, the connection-name attribute will contain the identifier of the connection. Connections are included by their resource ID.

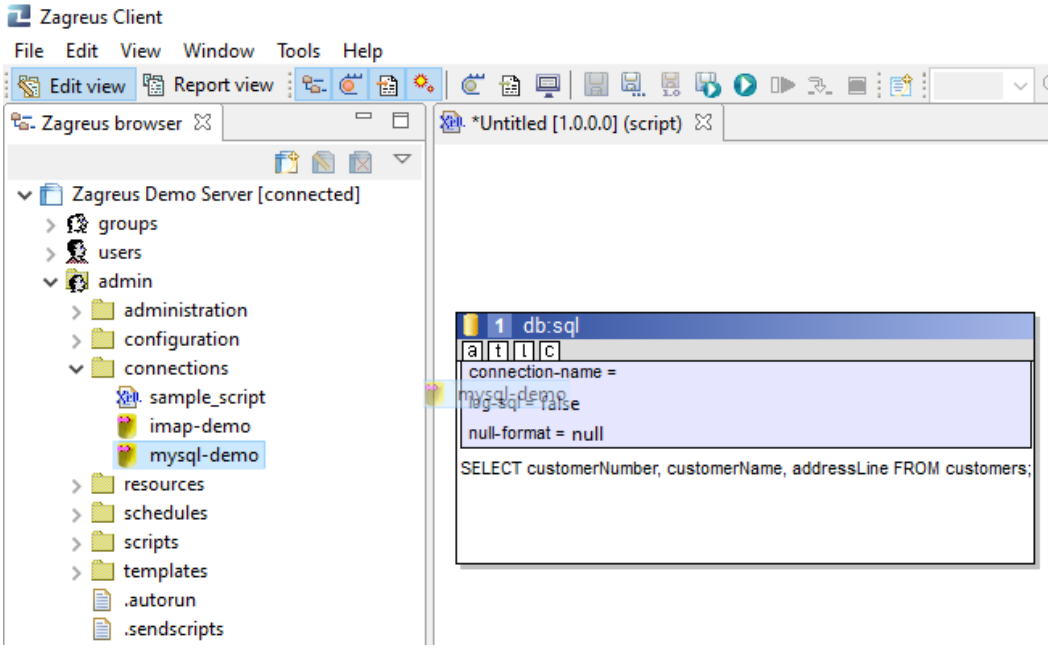


Figure 11 – Drag-and-drop event of a connection resource: it is pulled onto the attribute *connection name*

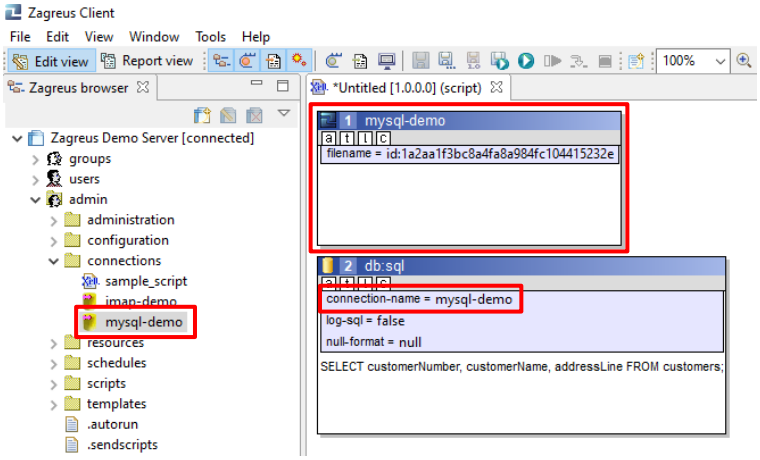


Figure 12 – Result of a connection include

For possible other referencing actions, the value of the *connection-name* attribute can be copied from one action to the other, or entered manually. Alternatively, drag-and-dropping the same connection multiple times also works – the connection resource will be included only once in this case.

Info: *The current version of the connection resource is included.*

9.2.4 Closing a connection

Some servers are quite sensitive to connections kept open; therefore, it is important to close a connection after it is used. In Zagreus, there are two ways to close a connection.

In the first one, the user can manually close a connection by using the appropriate *close* action. For example, a MicroStrategy connection (action *mstr:connection*) can be closed by the *mstr:close* action, which will close the connection to the given MicroStrategy server, when the script execution reaches this action (see → Order of execution, result flow). Of course, the *connection-name* attribute of the close action has to be filled with the name (identifier) of the connection. See *Figure 13.* for an example.

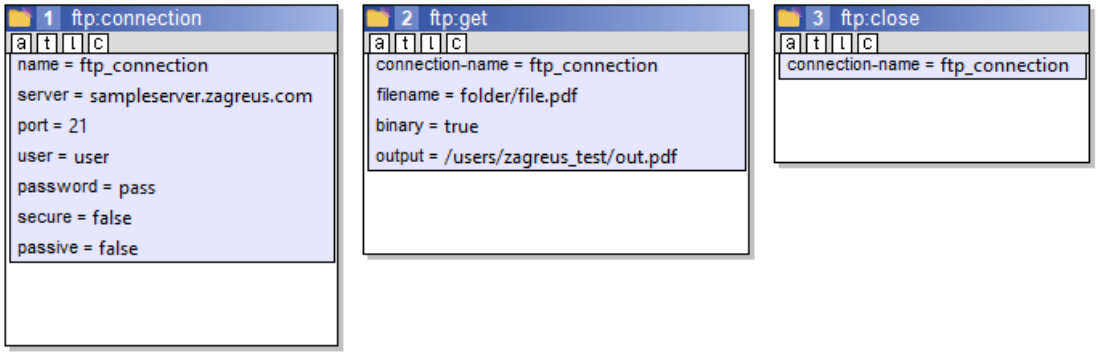


Figure 13 – Closing an FTP connection by the `ftp:close` action

In the other approach, the user can rely on Zagreus to automatically close the open connections. This will happen at the end of the execution of the containing block, which has the connection definition as its direct child. (For script structures, see → Actions, while for script execution, see → Order of execution, result flow.) See *Figure 14.* for an example: the FTP connection is defined in Action 1.1 (`ftp:connection`), but it is opened only in Action 1.2 (`ftp:get`). When script execution reaches the end of the containing action, i.e. Action 1 (`z:block`), this connection is closed automatically, therefore it will already be closed in Action 2 (`z:log`).

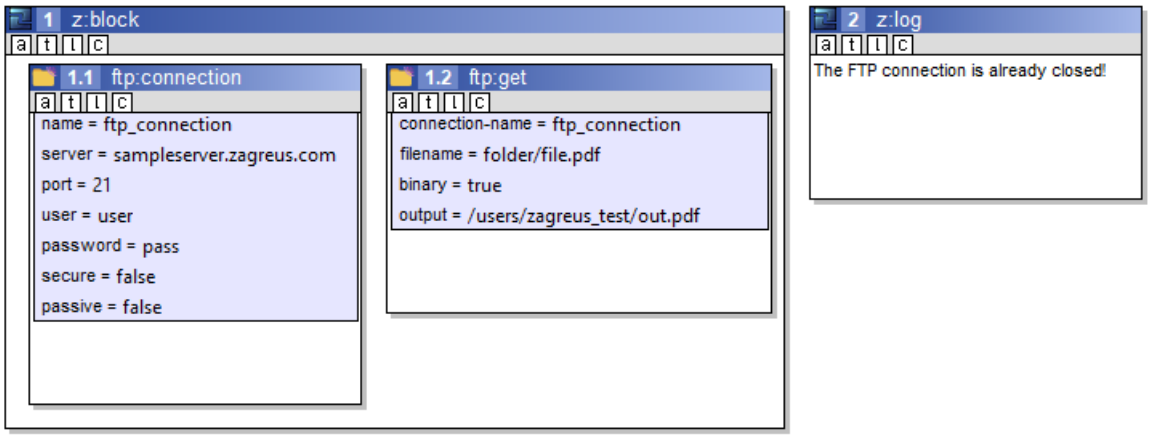


Figure 14 – Closing an FTP connection automatically, at the end of the defining block

Info: *Connections defined or included on the root level will be closed at the end of Zagreus script execution.*

9.2.5 Opening connections in the Zagreus browser

It is also possible to connect and browse directly in some types of connections in the Browser window of the Zagreus Client. For this, the user has to create a Zagreus connection resource. To connect to the given server, the user has to right-click on the given connection resource, and select the *Connect* menu item from the context menu, see *Figure 15*.

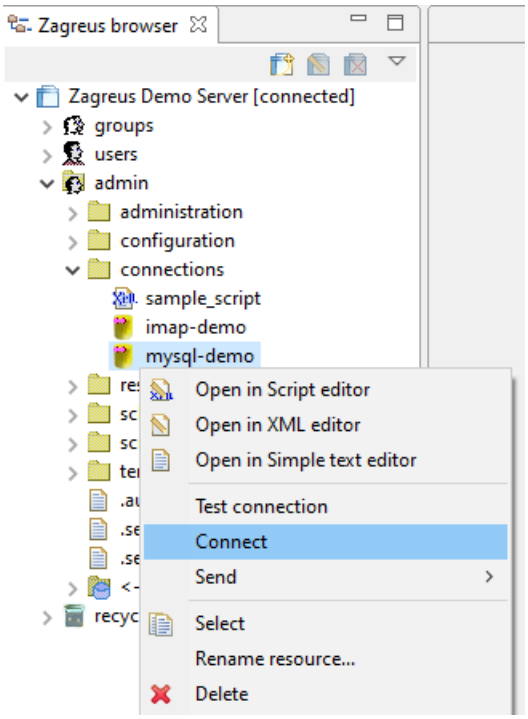


Figure 15 – Connecting to a database server (via a connection resource) in the Browser window

The browsing functionality is limited to specific connection types, and the functionality of browsing (i.e. items shown) is limited as well.

Zagreus can connect directly to the following server types:

Connection type	Available features
Database (i.e. SQL)	Get database information, list properties of tables and views, list table content
Mail (IMAP)	List folders of INBOX and check folder content, show e-mail information
Mail (POP3)	List e-mail information of INBOX
MicroStrategy	List project objects, list resource information, copy and paste resource path into the *name* attribute of *mstr actions.* Drag-and-drop MicroStrategy reports and documents onto the canvas of the graph editor.

9.2.5.1 Database connections

After connecting to a database server (see *Figure 15.*), listing items of a database connection can be done in the sub-tree of the opened connection. The browsing tree goes down to column definition level; for an example, see *Figure 16*.

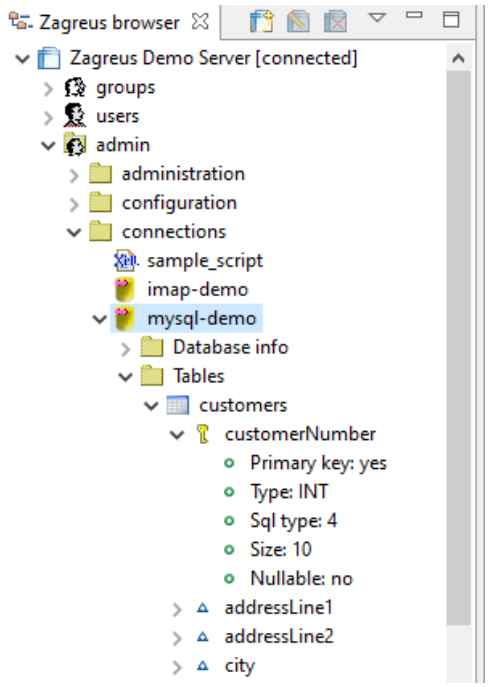


Figure 16 – Listing column information of a database table

It is also possible to check the content of a database table by right-clicking a table name and selecting the *Show data* menu item from the context menu, see *Figure 17.*

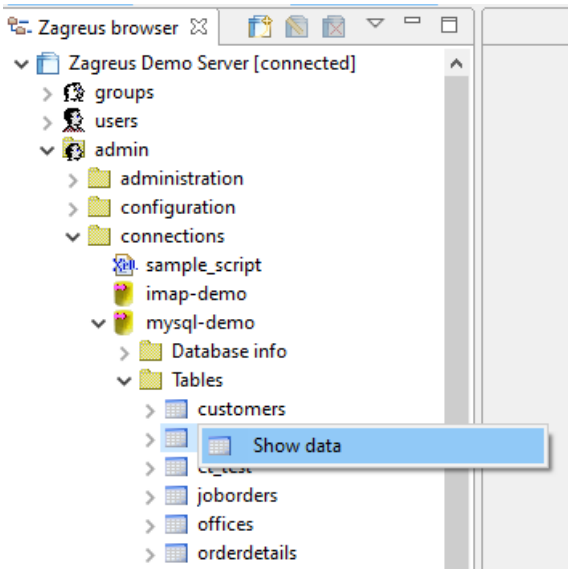


Figure 17 – Opening the content of a table

The table content will be shown in a separate view in the Zagreus Client; the name of the window will be the same as the name of the database table. For an example, see *Figure 18*.

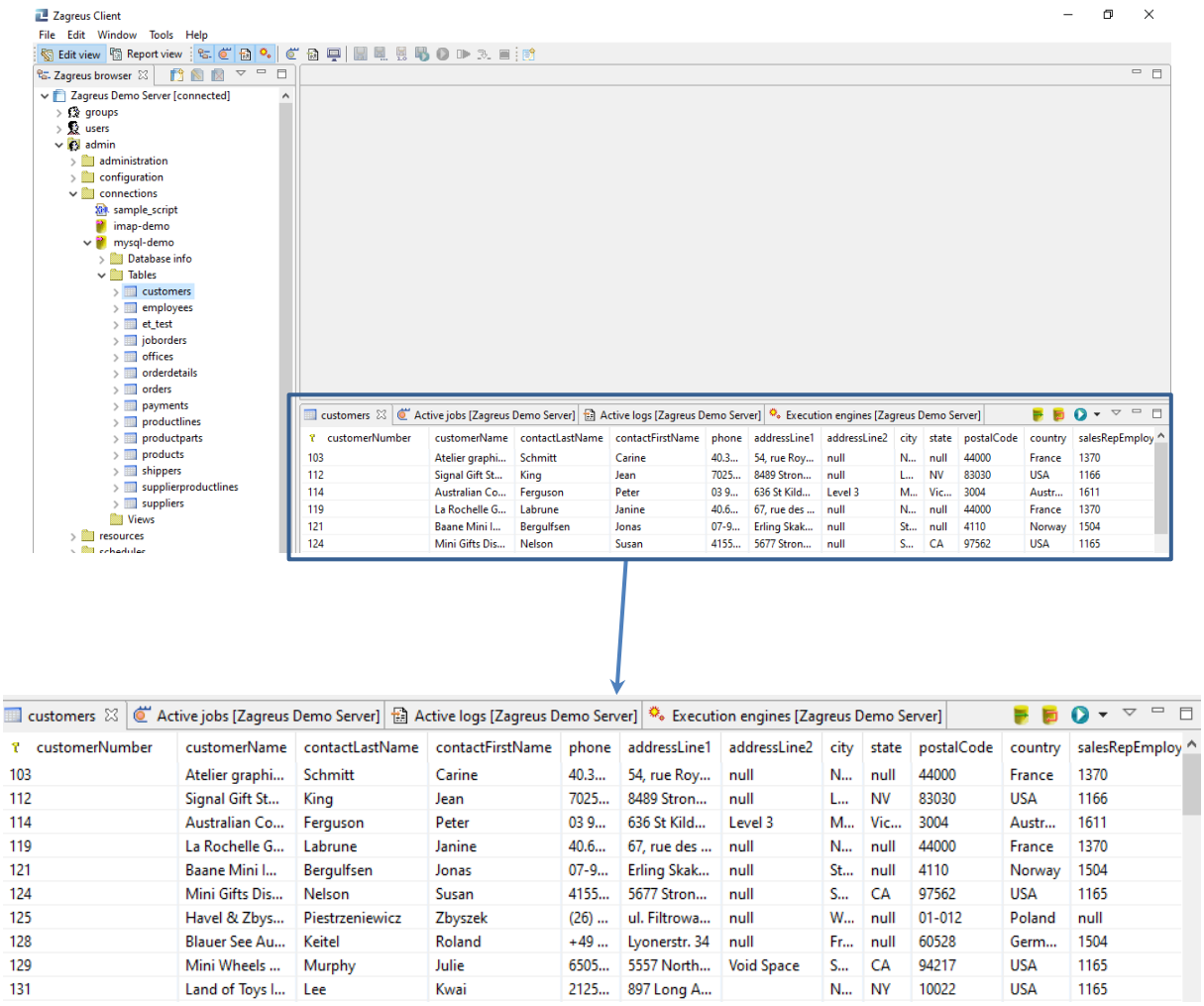


Figure 18 – Showing the content of a database table

9.2.5.2 Mail connections

It is also possible to connect to a *Mail* connection in the Zagreus browser, if the type of the mail connection is *IMAP*. In this case, the folders of the connection are listed along with the number of the contained e-mails; for an example, see *Figure 19*.

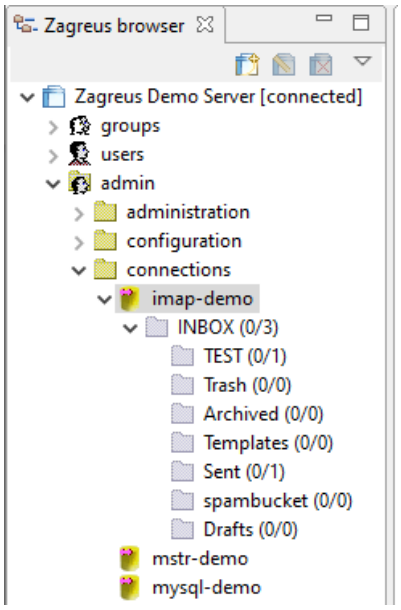


Figure 19 – Folders of a Mail (IMAP) connection are listed along with the number of e-mails

Similarly to the case of the database connections, it is possible to get information about the objects on the server. In this case, this means getting information about the mailboxes; for that, the user has to right click on the given mail folder, and select the *Show mail info* menu item from the context menu, see *Figure 20*.

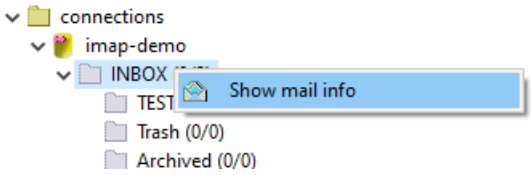


Figure 20 – Folders of a Mail (IMAP) connection are listed along with the number of e-mails

9.2.5.3 MicroStrategy connections

When a MicroStrategy connection is opened in the Browser window of the Zagreus Client, it is possible to list MicroStrategy resources of the given connection. It is also possible to get further information about specific MicroStrategy resources; for this, the user has to right click on the given resource and select the *Show resource information* menu item from the context menu. This will open the MicroStrategy object info window, which contains basic properties of the given object, see *Figure 21*.

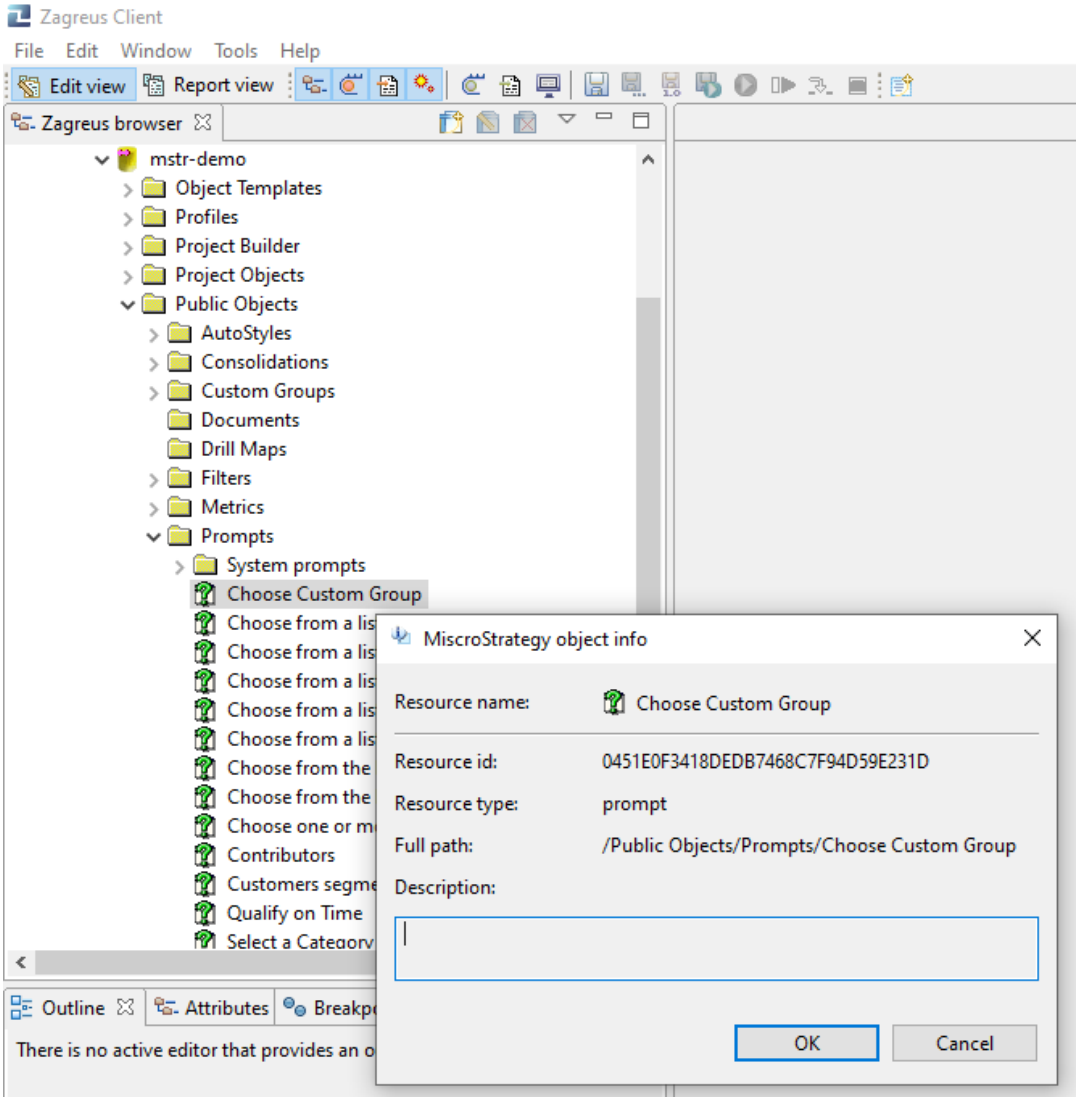


Figure 21 – Listing prompts of a MicroStrategy project and displaying object information

It is also possible to drag-and-drop MicroStrategy *reports* or *documents* onto the canvas of the Script Editor, see *Figure 22*. In this case, the corresponding Zagreus action (i.e. `mstr:report` or `mstr:document`) will be created, and the used connection will be included in the script automatically.

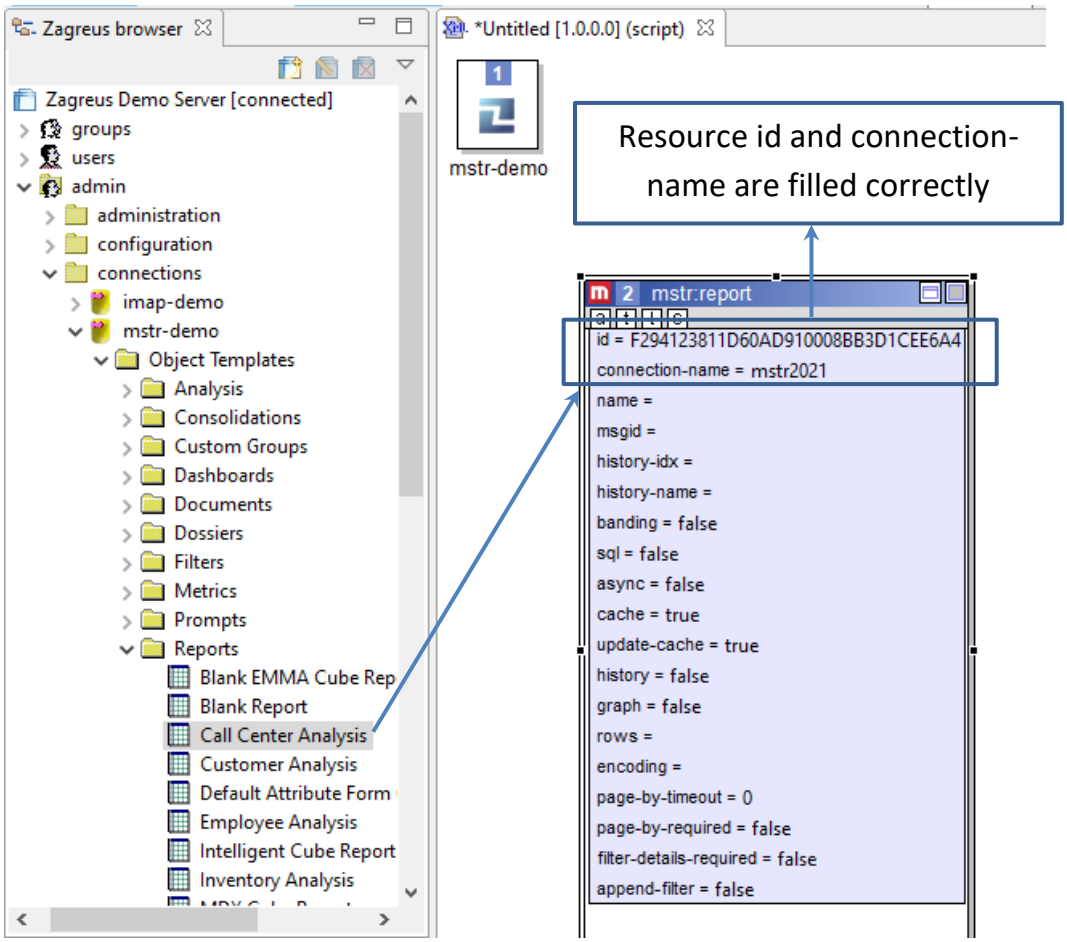


Figure 22 – Drag and drop a MicroStrategy report from the browser window. Notice that the connection is also included

9.3 Secure connections

For specific connection types in Zagreus (e.g. FTP, MicroStrategy), there is the possibility to use them in a *secure* way. To use such a secure connection, the corresponding certificate(s) must be imported into the keystore file of Zagreus (see → Manage certificates). Using a secured connection can be set differently in the case of each connection types. For instance, in the case of an LDAP connection, attribute *protocol* has to be set as *LDAPS,* while in the case of an FTP connection attribute *secure* must be set to *true*. Settings can be checked in the help of the given action, see also → Action help.

9.4 *zs* connection

The `zs:connection` and the `zs` action group (standing for Zagreus Server, marked with green headers in the Zagreus Client) are unique among the connection types supported by Zagreus.

The user can perform actions on a specific Zagreus server initiated from a Zagreus script using the actions in the `zs` action group. This is particularly useful for the following use cases:

- executing another script from the actual script
- obtaining information about jobs, subscriptions and other queue-related server-side components
- managing subscriptions
- performing file operations on the server side (see → Embedded MySQL database and → Local filesystem in the Zagreus Server)
- migrating scripts, users, subscriptions or other resources from one Zagreus Server to another by the `zs:migrate`, `zs:migrate-user-group` and `zs:migrate-subscription` actions

The `zs` connection is using the *http / https* communication protocol (i.e. they are calling webservice functions of the Zagreus Server).

It is a common use case to perform these actions on the local Zagreus Server (i.e. the one that the actual Zagreus Worker is connected to). To aid this, it is possible to reference the local Zagreus Server by leaving the *connection-name* attribute empty in in all the actions of the `zs` action group. In such cases, the so-called *local Zagreus connection* will be referenced implicitly.

⚠ ***Warning:*** *The local Zagreus Server runs on its default* http / https *port and the local zs connection is counting on these port values. If the Zagreus Server is not using the default* http / https *port, the administrator needs to adjust the* worker.serverport *property in the configuration of the Zagreus Worker, see → Connection properties*.

9.5 Tips and tricks

This chapter contains some tips for making connection administration and usage easier. Of course, these practices are merely suggestions, and they are not the only way of doing the maintenance of connection actions and resources.

9.5.1 Creating standalone connection resources

During the creation of a connection create it as a standalone resource – changes are easier to make. And changes tend to be necessary from time to time, either due to different firewall settings, relocated virtual machines or password expiration.

9.5.2 Using subfolders

It is recommended to collect connection resources into a separate folder. It can be a shared folder for each users or each user can create her/his own connection folder – depending on the environment.

9.5.3 Using meaningful names

It is advisable to give meaningful names both for the connection resources and for the connection actions, preferably also containing type of connection in the connection name, because connections are shown with the same icon in the Zagreus Graph Editor, and distinguishing them by a meaningful name is easier.

9.5.4 Using resource versioning

Versioning of connection is possible in the same way as, like, script versioning. When a connection is drag-and-dropped into a script, its current version is included. If the same connection would be used with minor changes – such as date format or collation –, then multiple versions of the same connection can be created. This can be a need as well when the different connection versions refer to the same server, but with different users (on the remote server). Switching between the connection configurations can be done easily by setting the *current version* property of the connection resource, see also → Resource versioning.

9.5.5 Keeping connections up-to-date

With some simple administrative steps, the usage of connections can be easier and some problems can be prevented.

Running connection tests periodically – even if the connection is not in use – can reveal changes in the configuration such as changes in server configuration or in the user of the connection. In such cases, problems can be revealed before connection include and script execution, and the necessary steps for fixing the connections can be performed.

10. Zagreus Client

Zagreus offers an intuitive graphical user interface (GUI - *Graphical User Interface*), which allows access to all of the functions of Zagreus Server. The interface of the Zagreus Client has several main parts, ensuring it is easy to maintain a good overview:

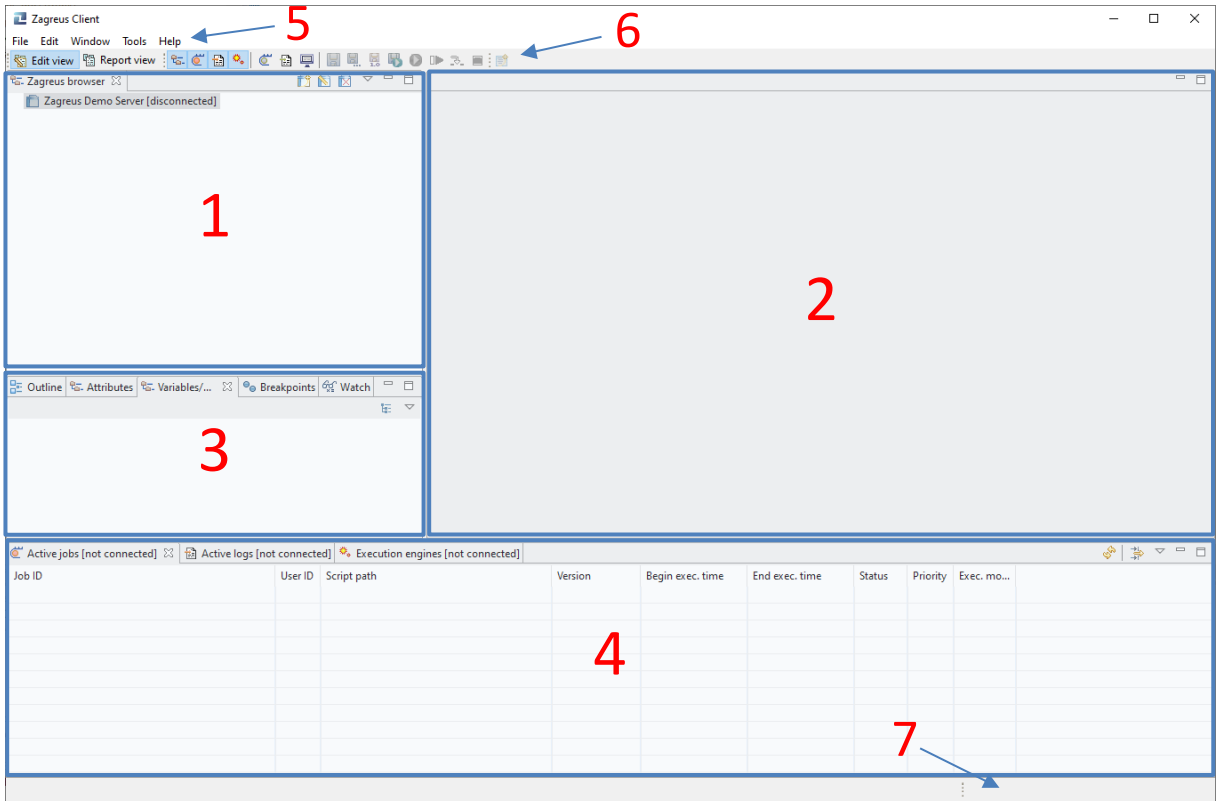


Figure 1 – The parts of the Zagreus Client

1) *Zagres browser window*

   Shows the folders and database / server filesystem resources. The user can open, manage resources as well as administer the server-side settings such as user and group management, starting and stopping server components.

2) *Editor area*

   Allows the editing of various types of resources.

3) *Extension windows for editor area*

   It is a container for additional windows related to the opened resource that is currently edited in the Editor area.

4) *Monitoring windows*
   Displays active services and their logging information, the status of execution engines and job reports.

5) *Main menu bar*
   The main menu bar of the Zagreus Client application.

6) *Main toolbar*
   The main application toolbar containing tools for managing windows, layouts in the Zagreus Client as well as tools for editing, saving and debugging the resource being edited.

7) *Status bar*
   Shows information about the ongoing processes.

10.1 Zagreus browser window

The user can navigate through a tree-like structure of database and server filesystem resources in the *Zagreus browser* window (for resource types, see → Resource types).

The root level contains one or more Zagreus Server definition nodes. Once the user has successfully connected to a server, the server definition node expands and the content of the server root level is appearing beneath the particular server definition node.

The Zagreus browser window has its own toolbar for adding, modifying and removing server definition nodes. This toolbar can be found in the upper right corner of the browser window, see *Figure 2*.

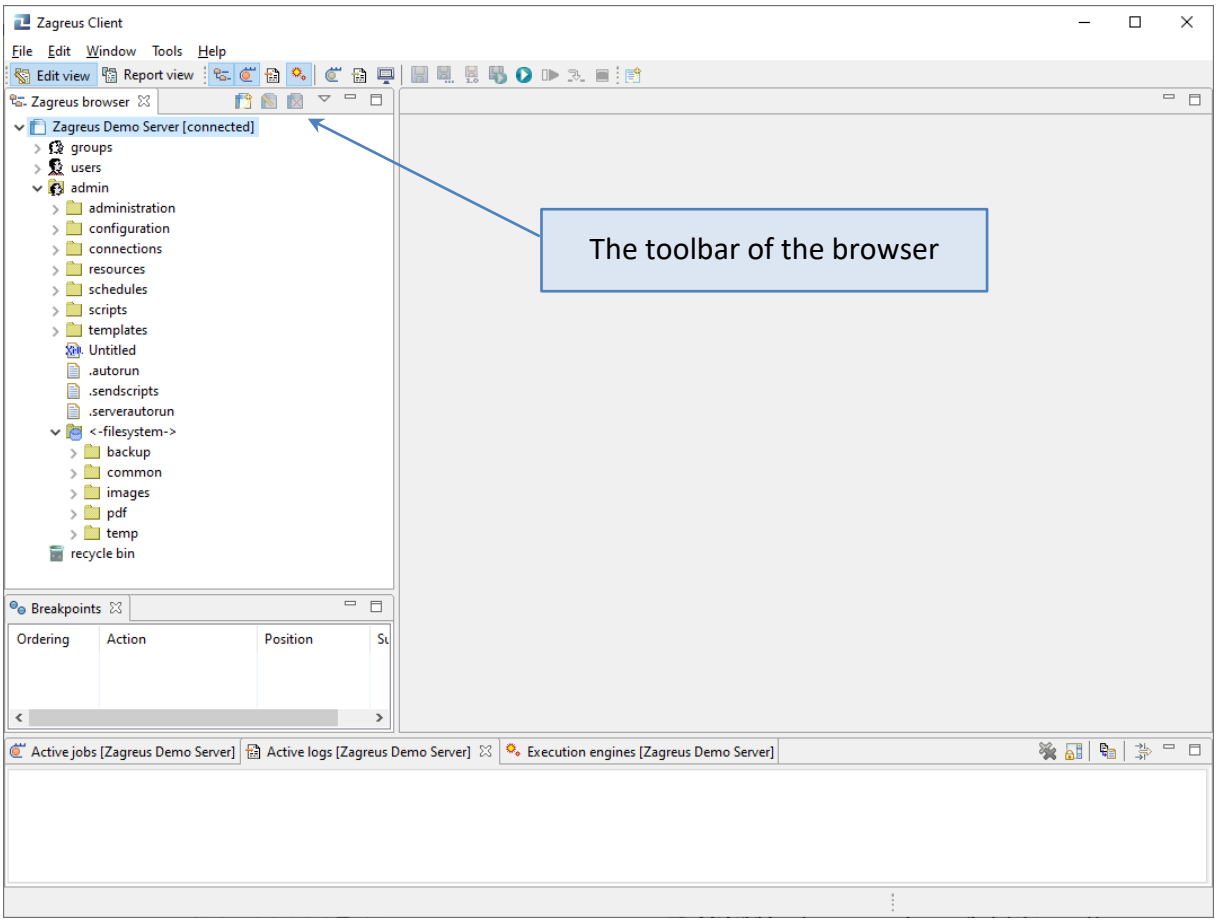


Figure 2 – The location of the Zagreus browser toolbar

Server definition nodes can be opened either by double-clicking on the definition node or by right-clicking on the node and selecting the *Connect to server* menu item in the context menu.

10.1.1 Toolbar

The Zagreus browser window has its own toolbar, which contains necessary server node management tools.

10.1.1.1 Adding a server definition node

To connect to a Zagreus Server, first a new server definition node needs to be created in the browser. This can be done by clicking on the *Add server definition* tool, see *Figure 3*.

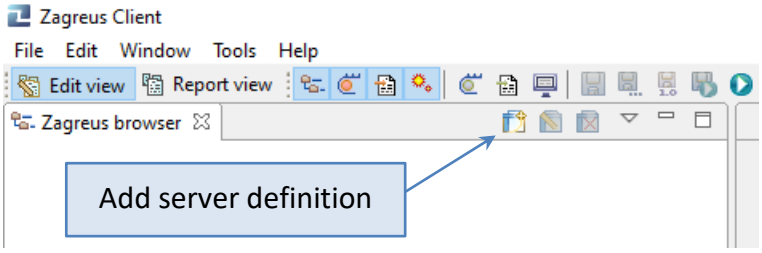


Figure 3 – The *Add server definition* tool

The following connection parameters must be specified in the appearing *Define Zagreus Server* dialog box, see *Figure 4.*:

- *Connection name*: the name of the server definition
- *Host*: the hostname or IP address of the server
- *Port*: the port of the server
- *Secure*: must be checked for secure (SSL) connection
- *User*: the user name
- *Password*: the password of the user
- *Save Password*: if checked, the Zagreus Client will store the password in an encrypted format. If the password is not saved, it will be required to be typed every time when the user attempts to connect.

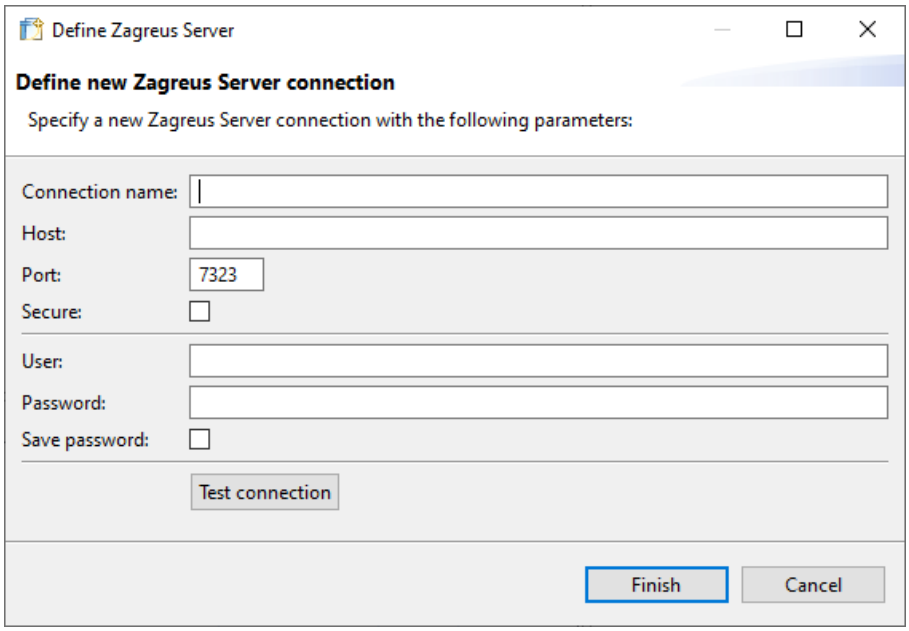


Figure 4 – The *Define Zagreus Server* dialog box

The connection can be tested by clicking on the *Test connection* button. The new server definition node will be created by clicking on the *Finish* button.

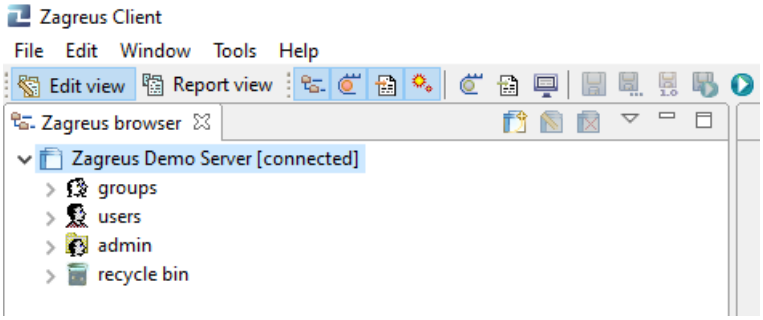


Figure 5 – An opened server definition node after a successfully established connection

10.1.1.2 Modifying a server definition node

For modifying an existing server definition node, the user needs to right-click on a closed server node and select the *Modify server definition* menu item in the context menu or select the *Modify server definition* tool in the browser window toolbar, see *Figure 6*.

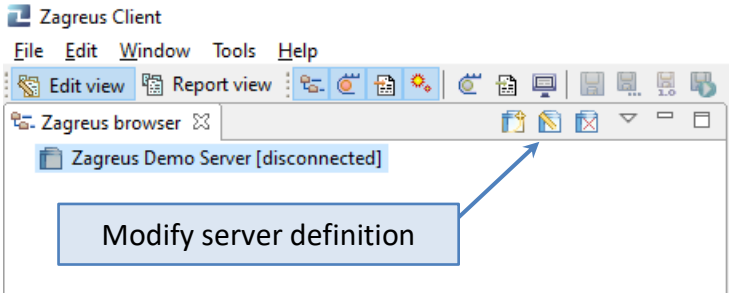


Figure 6 – The *Modify server definition* tool

All the aforementioned parameters of the server definition can be changed in the dialog box, see *Figure 7*.

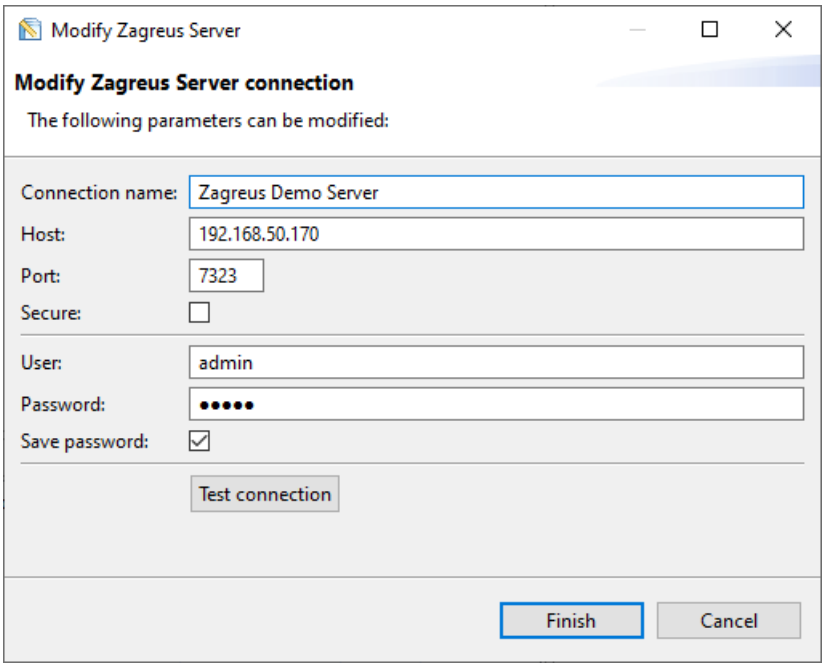


Figure 7 – The *Modify Zagreus Server* dialog box

10.1.1.3 Removing a server definition node

For removing an existing server definition node, the user needs to right-click on a closed server node and select the *Remove server definition* menu item in the context menu or select the *Remove server definition* tool in the browser window toolbar, see *Figure 8*. The server definition node will be removed after confirmation.

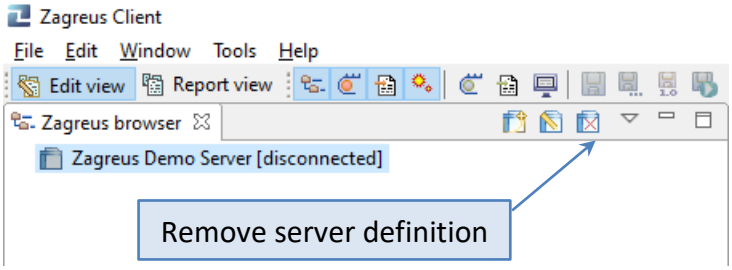


Figure 8 –The *Remove server definition* tool

10.1.2 Basic navigation

The Zagreus browser represents the files and folders of a Zagreus Server in a tree-like structure. The user can navigate in this structure by the basic operations described in the next subchapters.

10.1.2.1 Opening and closing a server definition node

Server definition nodes can be opened either by double-clicking on the definition node or by right-clicking on the node and selecting the *Connect to server* menu item in the context menu. If the password is not saved to the definition node, the user will be prompted to specify it (see *Figure 9.*).

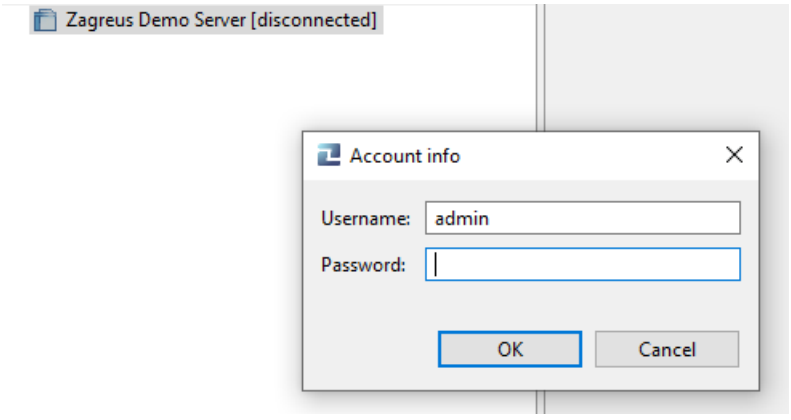


Figure 9 –The *Account Info* dialog box

If the connection is successful, the server definition node opens and the base content of the server root level is appearing under the server definition node (see *Figure 2.*). The title postfix of the server definition node is changing from *[disconnected]* to *[connected]*.

In addition, the *Active jobs*, *Active logs* and *Execution Engines* windows will change their respective header postfixes from *[not connected]* to the *[server_node_name]* and

show actual information from the server (see → Active jobs window, → Active logs window and → Execution engines window).

To close an open Zagreus connection, the user needs to right click on the server definition node and select the *Disconnect from server* menu item from the context menu.

10.1.2.2 Expanding and collapsing tree nodes

Once the server connection is established, the user can navigate through the tree-like folder structure. The first level of an opened server connection contains the *users*, *groups* folders and the *admin* user home folder (if the dedicated *admin* user logged in). If the usage of recycle bin is set (see → Recycle bin and → Miscellaneous properties), the *recycle bin* folder also appears at the end of the list.

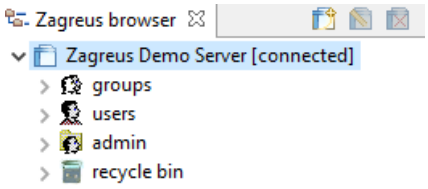


Figure 10 – Base level of a connected server node

These folders arrange the basic structure of the local database files and subfolders.

- *groups*
  This folder contains the available groups for the logged-in user. The group *public* is predefined in the shipped Zagreus versions. For group management, see → Groups in the Zagreus System and → Context menu of a group node.

- *users*
  This folder contains the home folder of the logged-in user (when a non-admin user is logged in) or the home folders of all the users (when an admin user is logged in). For user management, see → Users in the Zagreus System and → Context menu of a user node.

- *admin*
  The *admin* user home folder is separated from the home folders of other users. System-wide administrative scripts are mapped under the admin user dedicated folders, see → Administrative scripts.

- *recycle bin*
  The recycle bin is a system-wide folder that contains the deleted resources. See → Recycle bin.

During browsing, the folders can be opened either by double-clicking on the folder name, or by clicking on the > sign before the folder name. Closing an open folder can only be performed by clicking on the ∨ sign before the folder name.

Versioned resources behave similarly as folders. Versions are listed under a version parent resource node, see → Resource versioning.

10.1.2.3 Refreshing tree nodes

The refresh operation can be performed on folders, version parent resources and other resources as well. In case of refreshing a folder, the content of the folder will be retrieved from the Zagreus Server. The same applies when refreshing a version parent node. For other resources, the resource properties are updated.

> ***Info:*** *if a description is set for a resource, a tooltip is showing the description text when the user hovers the mouse over the resource name.*

10.1.3 Common resource management operations

The user can perform specific resource management operations, e.g. creating, deleting and copying selected resources.

10.1.3.1 Creating new resources

Creating new resources can be initiated by right-clicking on a particulat folder tree node and selecting either the *Create folder…* or the *Create new resource…* menu item from the context menu, see *Figure 11*. The latter can also be done by clicking on the *Create new resource…* icon on the main toolbar, see → Create new resource.

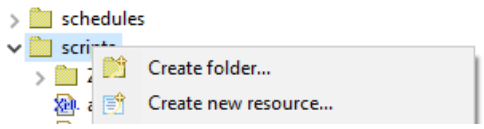


Figure 11 – Create new resources menu items

Selecting the *Create folder…* menu item, a *New Folder* dialog box appears. The name of the new folder has to be entered, see *Figure 12*.

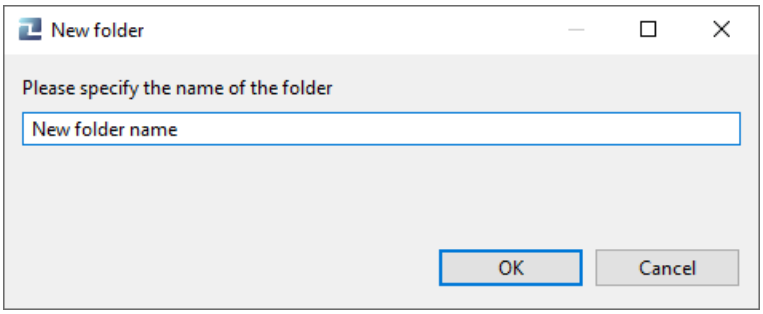


Figure 12 – The *New folder* dialog box

Selecting the *Create new resource…* menu item opens the *Creating new resource* dialog box, see → Create new resource.

10.1.3.2 Deleting resources

In order to delete a resource, the user has to do one of the following operations:

- right-click on the particular resource and select the *Delete* menu item from the context menu
- press the Delete key when the selection is on the resource to be deleted, or
- select the *Delete* menu item from the main *Edit* menu.

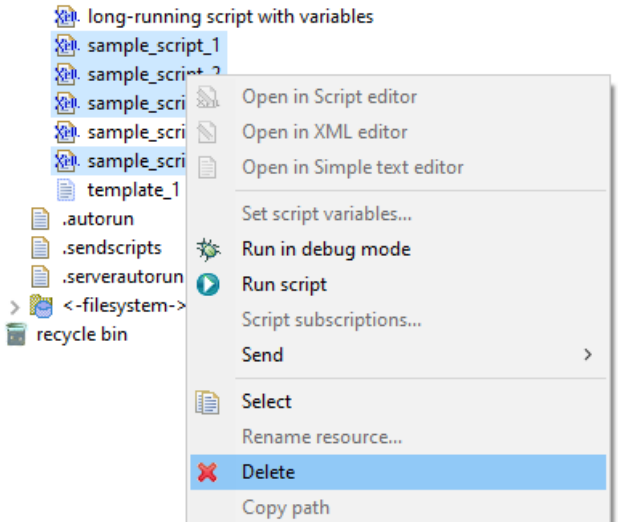


Figure 13 – Deleting multiple resources

The delete operation is then performed after clicking *OK* on the confirmation dialog box.

If the usage of the recycle bin is configured (see → Miscellaneous properties), the deleted resource is first moved into the recycle bin. In this case, permanent deletion is only accessible from the recycle bin context menu (see → Recycle bin).

10.1.3.3 Copying and moving resources

Copying and moving are performed in two steps in the Zagreus browser window. First, the resource (or resources) have to be selected, see *Figure 14*. In case of selecting multiple resources, the selected resources must be siblings (they must be within the same folder).

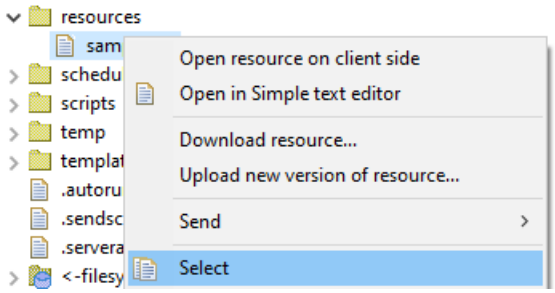


Figure 14 – Selecting a resource to be copied or moved

Then the user needs to click on the target folder (either on the same or on another connected Zagreus server), and select the *Copy selected resources* or *Move selected resources* menu item to perform the respected operation, see *Figure 15*.

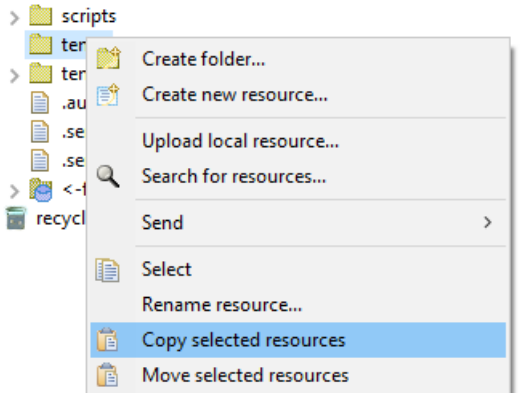


Figure 15 – Performing the copy resources operation

The behaviour of both operations depends on the settings which can be found in the *Copy* tab in the *Options* dialog box. It can be accessed by selecting the *Options…* menu item from the *Tools* main menu, see also → Copy tab.

> ***Info:*** *only scripts, folders and simple files are allowed to be copied or moved from the embedded database to the local filesystem. The script content will be a simple xml file in this case.*

10.1.3.4 Renaming resources

In order to rename a resource, the user has to do one of the following operations:

- Right-clicking on the particular resource and selecting the *Rename resource…* menu item from the context menu.
- Pressing the *F2* key when the selection is on the resource to be renamed.
- Selecting the *Rename resource…* menu item from the main *Edit* menu.

The user then needs to type in the new name of the resource, see *Figure 16.*

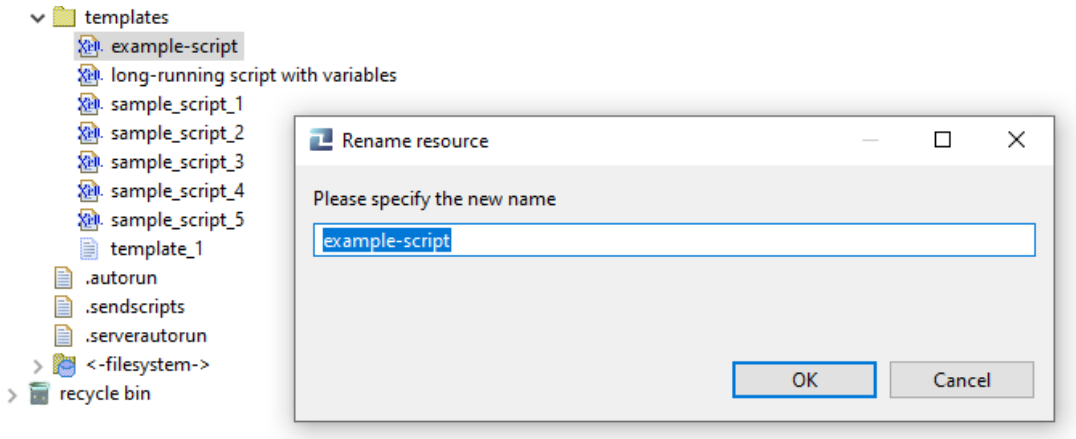


Figure 16 – The *Rename resource* dialog box

10.1.3.5 Uploading and downloading resources

Resources can be uploaded from an external source into the Zagreus embedded database or into the server filesystem. The *Upload local resource…* menu item is accessible for folders only, see *Figure 17*.

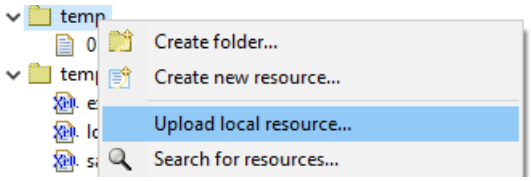


Figure 17 – The *Open* dialog box for selecting a resource to upload

The user can select a resource in the *Open* operating system dialog box. After clicking on *Open* button, the local resource will be uploaded into the selected target folder, see *Figure 18*.

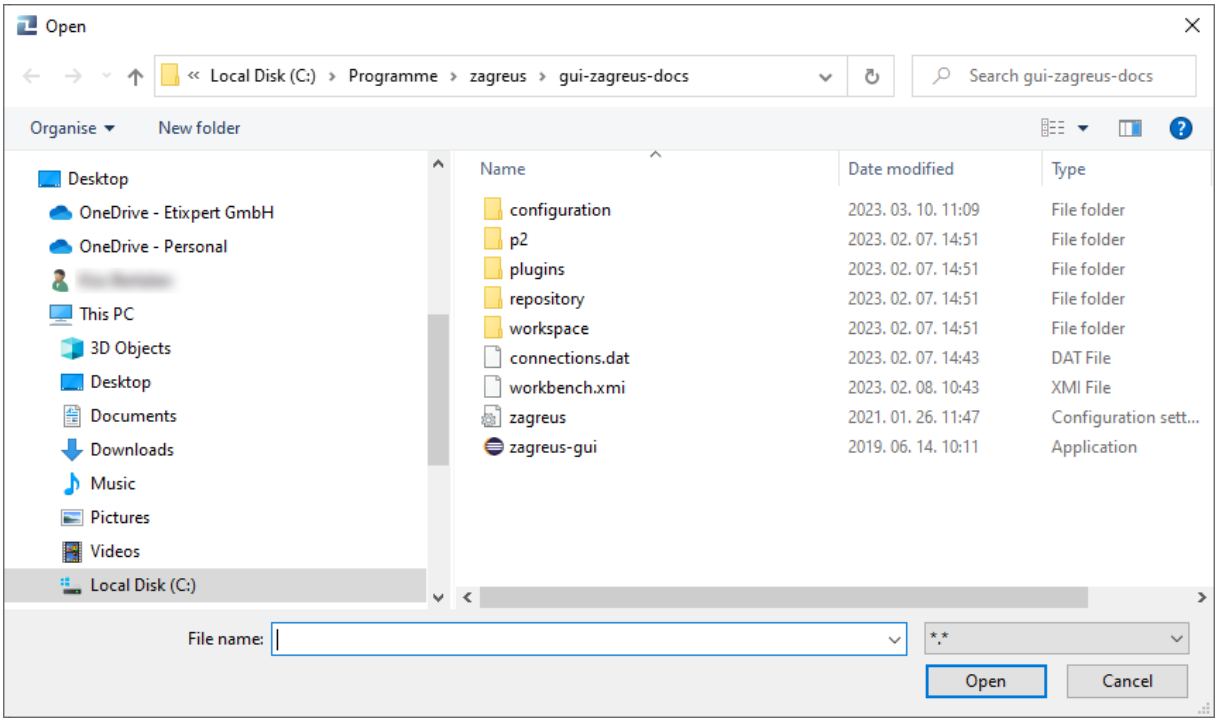


Figure 18 – The *Open* dialog box for selecting a resource to upload

Simple file resources can be downloaded into the operating system local filesystem from Zagreus Server. This can be performed by selecting the *Download resource…* menu item from the context menu, see *Figure 19.* The user first select a target folder in the appearing operation system *Save as…* dialog box.

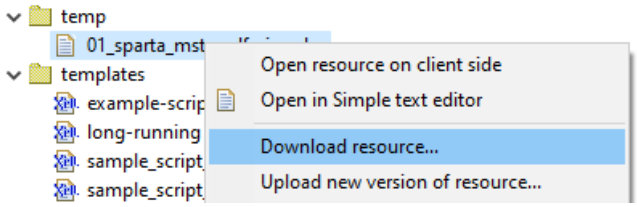


Figure 19 – The *Download resource…* menu item

Also, a new version of a selected simple file resource can be uploaded by right-clicking on the simple file resource and selecing the *Upload new version of the resource…* menu item in the context menu. After selecting a resource in the operating system *Open* dialog box, just like in case of simple upload operation, the user needs to specify the new version number for the resource, see *Figure 20.* In this dialog, the description of the new version can also be specified as well as the possibility to set it to the current version, see → Current version.

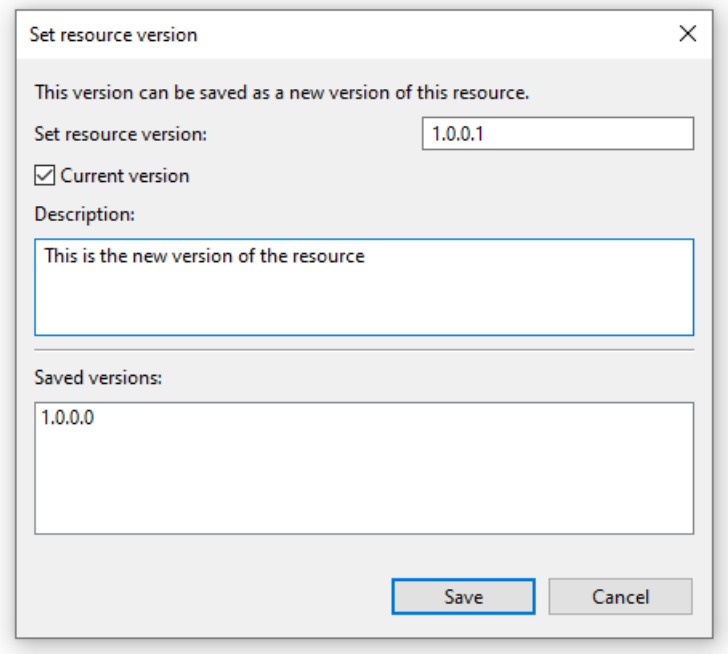


Figure 20 – The *Set resource version* dialog box

After clicking the *Save* button, the new version will be uploaded and merged to the existing version in the Zagreus browser list.

10.1.3.6 Resource information

A comprehensive set of resource properties can be displayed by right-clicking on the selected resource and selecting the *Show resource information* menu item from the context menu, see *Figure 21*. For the list of resource properties, see → Resource properties.

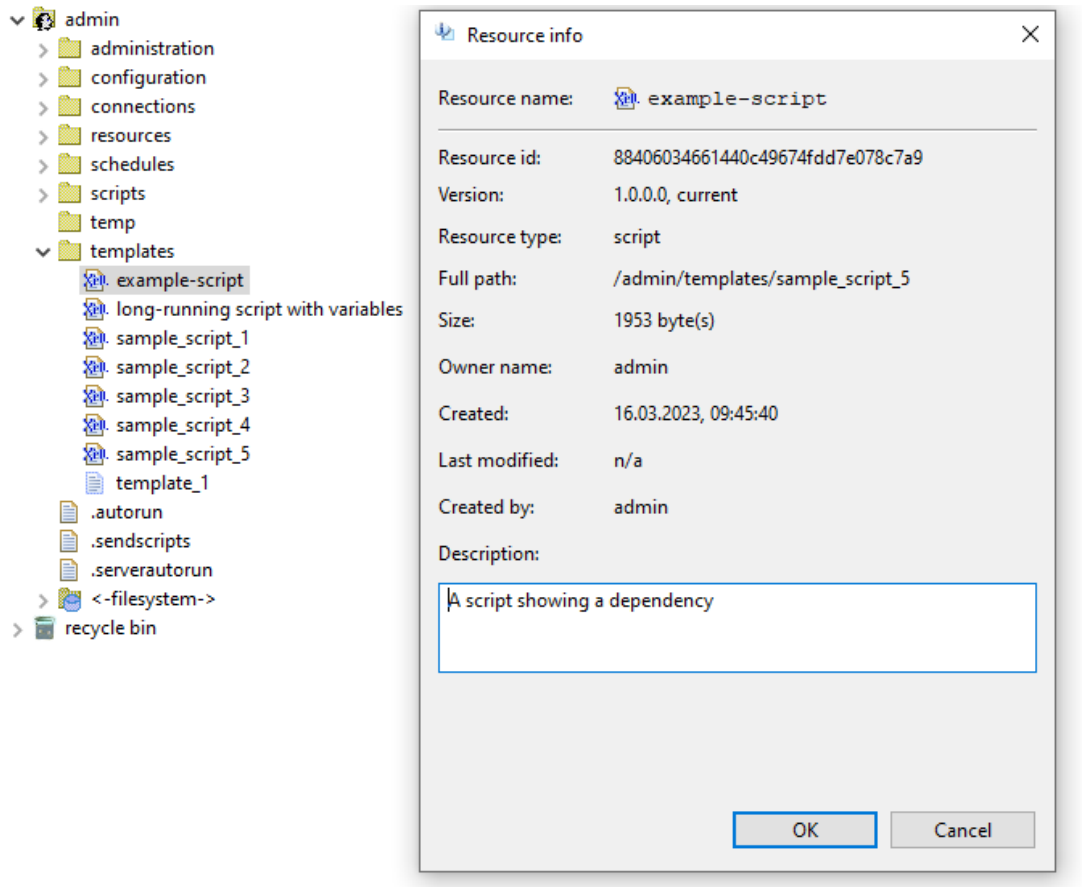


Figure 21 – The *Resource info* dialog box

10.1.3.7 Copying the resource path

The *Copy path* operation is accessible for all the resource types in the Zagreus browser. This operation allows the user to copy the resource full path into the Script Editor in a two-step manner: first, the user needs to select the *Copy path* menu item from the context menu (see *Figure 22.*), then the *Paste path* operation has to be performed in the Script Editor (see → Paste path).

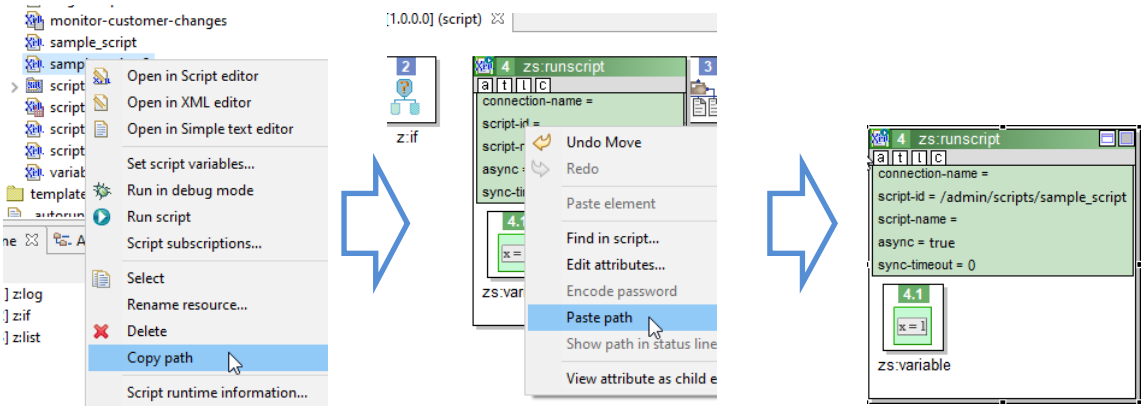


Figure 22 – The usage of the *Copy path* menu item

10.1.4 Opening resources

Resources can be opened in the editor area with their respective editor type. There are the following editor types in Zagreus:

- *Script editor*
  The default editor for scripts, templates and connections

- *XML editor*
  An alternative editor for XML-based resources (i.e. scripts, templates and connections)

- *Simple text editor*
  The default editor for simple text files, and an alternative editor for XML-based resource

- *Cron Time editor*
  The editor for time schedules

- *Event editor*
  The editor for event schedules

- *File Trigger editor*
  The editor for file triggers

- *DB Watcher editor*
  The editor for database watchers

- *Mail Watcher editor*
  The editor for mail watchers

The resource is opening in the default editor by simply double-clicking on it. An alternative way of opening the resource in the default editor is right-clicking on the resource and selecting the first menu item, i.e. *Open in <editor type>*, see *Figure 23.*

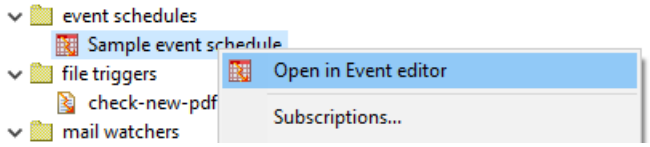


Figure 23 – Opening a resource in its default editor

For those resources that have an alternative editor, the user can select among multiple menu items, see *Figure 24.*

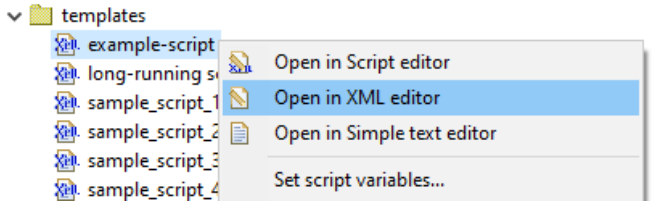


Figure 24 – Opening a resource in an alternative default editor

Furthermore, for simple file resources there is an option to open them in a selected external application, configured in the *Options* dialog, see → Download / upload tab. It can be performed by clicking on the *Open resource on client side* menu item, see *Figure 25*.

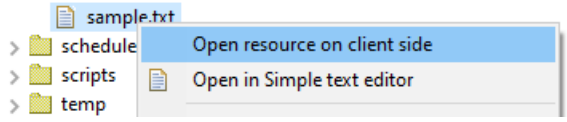


Figure 25 – Opening a simple file resource in an external application

10.1.5 Searching for resources

It is possible to search for resources by clicking on the *Search for resources…* menu item from the context menu. This menu is accessible only for folders. Then the *Search for resources* dialog box appears, see *Figure 26*.

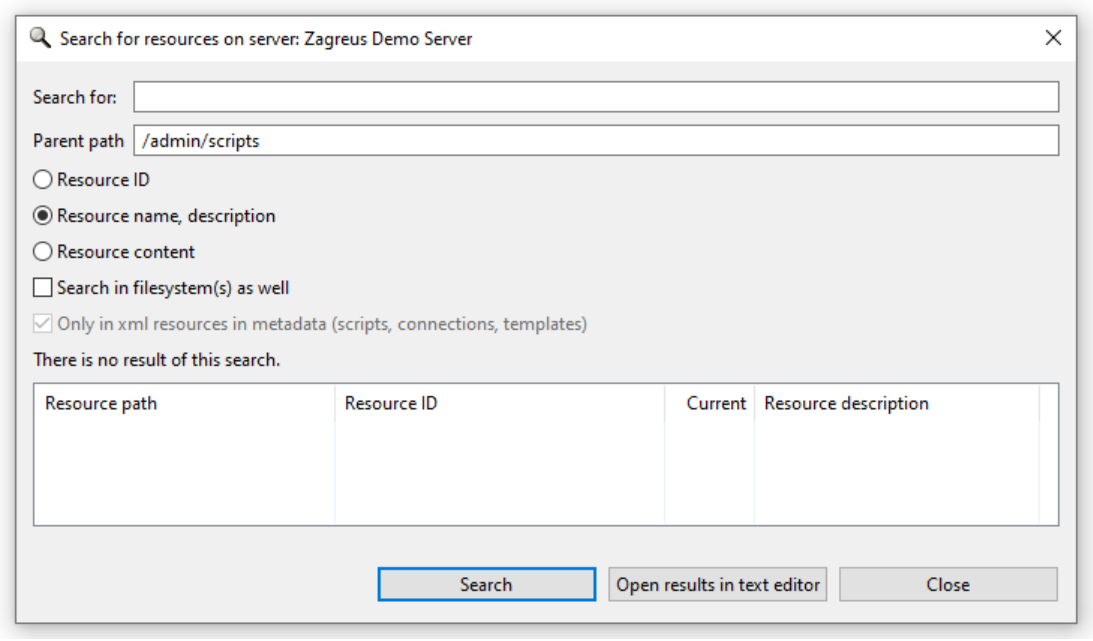


Figure 26 – Search for resources dialog box

The available search options in the dialog box are as follows:

- *Search for*
  The search filter. Can be applied for the ID, name, description or content of the resource; these options can be selected by selecting one of the radio buttons in this dialog box (see below).

- *Parent path*
  The path where the search will take place. It is the selected folder path by default.

- *Resource ID*
  When selected, the search filter will be applied for resource IDs.

- *Resource name, description*
  When selected, the search filter will be applied for resource names and descriptions.

- *Resource content*
  When selected, the search filter will be applied for resource content. Only XML-based (scripts, connections, templates) and/or textual files are included in the search, see the last setting in this list below-

- *Search in filesystem(s) as well*

the search will take place in the reachable filesystem as well (see → User rights and → Administrator user rights). It is switched off by default.

- *Only in xml resources in metadata (scripts, connections, templates)*
  it limits the content search for resources for XML-based resources: scripts, templates and connections in the embedded database

After clicking the *Search* button, the result is shown in the table at the bottom of the dialog box. If there is no result, a *There is no result of this search* message shown above the table and the table remains empty.

The result table colums display the following information, see *Figure 27.*:

- *Resource path:* the full path of the searched resource
- *Resource ID:* the ID of the searched resource
- *Current:* if the resource is a current version
- *Resource description:* the description of the resource

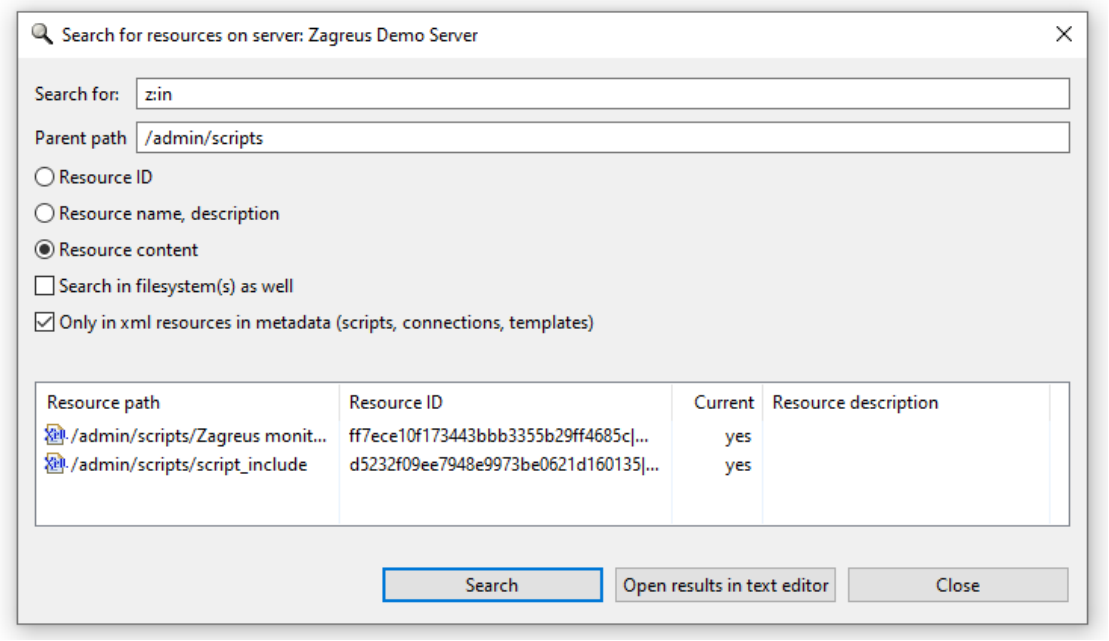


Figure 27 – The results shown in the table

Info: *Double-clicking on the table row of a resource opens the selected resource in the default editor.*

By clicking the *Open results in text editor* button, the result can be saved in the folder specified in the *Path where to save these files* field in the Options dialog (see → Download / upload tab), using the `export-result-<timecode>.txt` file name.

10.1.6 Drag-and-drop operations

There are some very useful functions that can be performed by drag-and-drop operations from the Zagreus browser window into an open script in the Editor area.

Source	Target	Key	Action
all	attribute	CTRL+SHIFT	the selected id is copied as the attribute value
script	editor area	-	`z:include` action created with the script id
script	editor area	CTRL	`z:block` action created with the contents of the selected script
connection	editor area	-	`z:include` action created with the connection id
connection	editor area	CTRL	the connection action is inserted
connection	*connection-name* attribute	-	the connection action is inserted to the script as the first action, and the *connection-name* attribute is filled properly
template	editor area	-	`z:include` action is created and the template is displayed on the palette bottom
template	editor area	CTRL	the template action is inserted
simple file	editor area	-	A `file:read` action is created with the path of the selected file

As it is displayed in the table above, the simple drag-and-drop operation creates a `z:include` action when it makes sense in terms of script execution (XML-based actions: script, connection, template), and for simple files it creates a `file:read` action.

The *Control* key + drag-and-drop operation makes a direct inclusion of the XML-based content. In case of a script, it is wrapped into a `z:block` action, because there can be several siblings of its root level.

10.1.7 Script-specific operations

There are several operations that can make only sense when a script is selected in the Zagreus browser tree.

10.1.7.1 Running a script

A manual script execution can be initiated by the *Run script* menu item in the context menu.

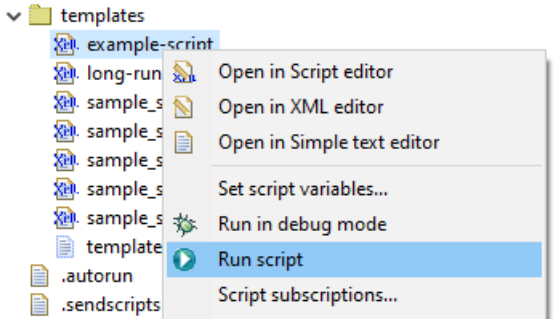


Figure 28 – Running a script manually

For different script execution modes, see → Job properties. This menu item initiates a script execution on the server side, in an asynchronous way. The script then goes into the queue that can be monitored in the *Active jobs* window, see → Active jobs window. Once the script execution has begun, the currently running script is displayed in the *Active jobs* and *Execution Engines* windows (see → Active jobs window and → Execution engines window), and the log messages are shown in the *Active logs* window, see → Active logs window.

After clicking on this menu item, a *Running has started on server* message appears in the status bar.

The following windows shows monitoring information about the running (or queued) script.

- *Active jobs window:* (see also → Active jobs window)

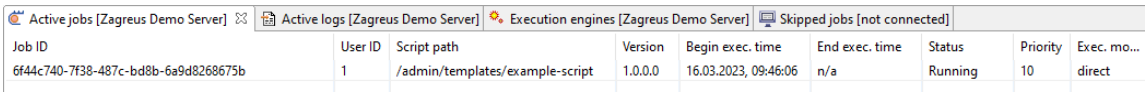


Figure 29 – A job appears in the *Active jobs* window

- *Active logs window:* (see also → Active logs window)

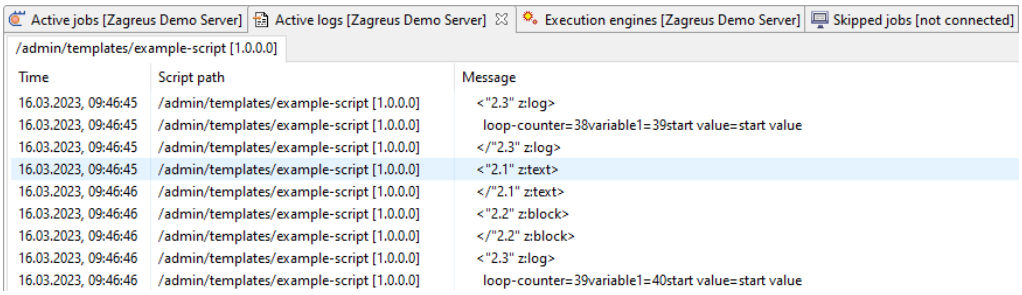


Figure 30 – Job-logs are displayed in the *Active logs* window

- *Execution engines window:* (see also → Execution engines window)

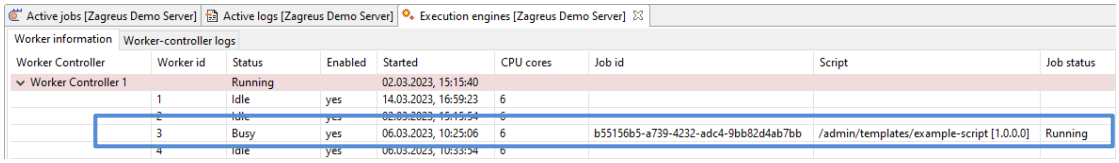


Figure 31 – A job appears in the *Execution engines* window

Further details of the finished scripts are available in the *Finished jobs* window (see also → Finished jobs window).

10.1.7.2 Running a script in debug mode

In order to run the script in debug mode (see → Debugging in the Zagreus Client), the user needs to select the *Run in debug mode* menu item from the context menu in the Zagreus browser.

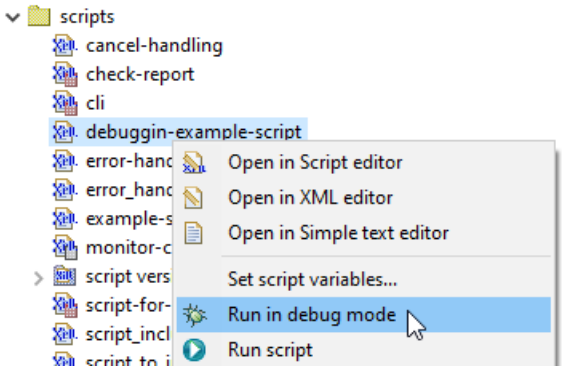


Figure 32 – The *Run in debug mode* menu item

For further information of running the script in debug mode, see → Debugging concepts and terms

10.1.7.3 Script subscriptions

Clicking on the *Script subscriptions…* (see *Figure 33.*) context menu item opens the *Subscriptions* dialog box, see → Subscriptions. Subscriptions for scripts can be created, modified and deleted here.

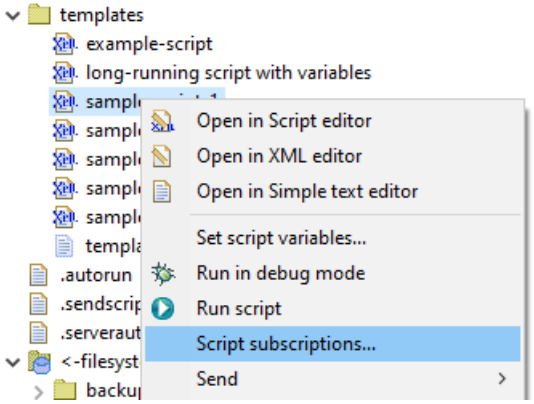


Figure 33 – The *Script subscriptions…* context menu

For understanding how subscriptions work, see → Subscriptions.

10.1.7.4 Setting script variables and options

By clicking on the *Set script variables and options…* context menu item (see *Figure 34.*), the *Script variables and options* dialog box will open (see *Figure 35.*).

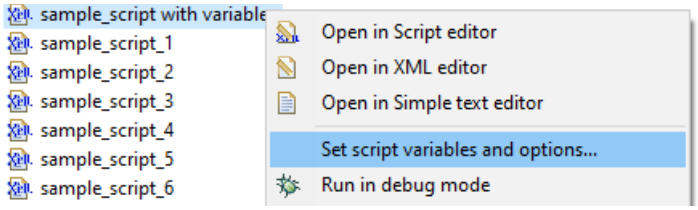


Figure 34 – The *Set script variables…* context menu

There are two separated tabs in this dialog box. The first one is the *Script variables* tab that is showing the script variables currently set for the particular script resource. For understanding variables and variable scopes, see → Variables.

The table is editable by double-clicking on the proper cells (*Name* or *Value* column) and entering the name or value respectively:

- A new name / value pair can be specified by double clicking on a cell in the first empty row in the table, entering the text then pressing *Enter* key.

- An existing name / value pair can be edited by double-clicking on an already filled cell, modifying the text and pressing the *Enter* key.

- A row can be deleted by simply deleting its *Name* entry, pressing *Enter* key. This name / value pair will not be saved after clicking the *OK* button.

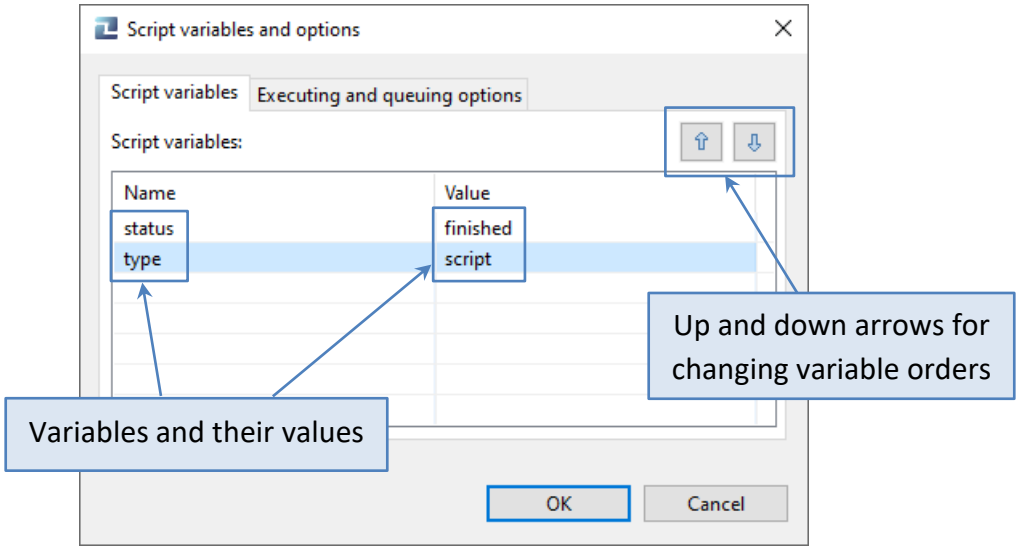


Figure 35 – The *Script variables* tab of the *Script variables and options* dialog

When there are multiple variables, they can be re-ordered by using the arrows in the top right corner of the tab. Variable order might be important for human readibility reasons e.g. adding a new important variable can be in the first place in the list this way.

After clicking *OK*, the changes will be saved.

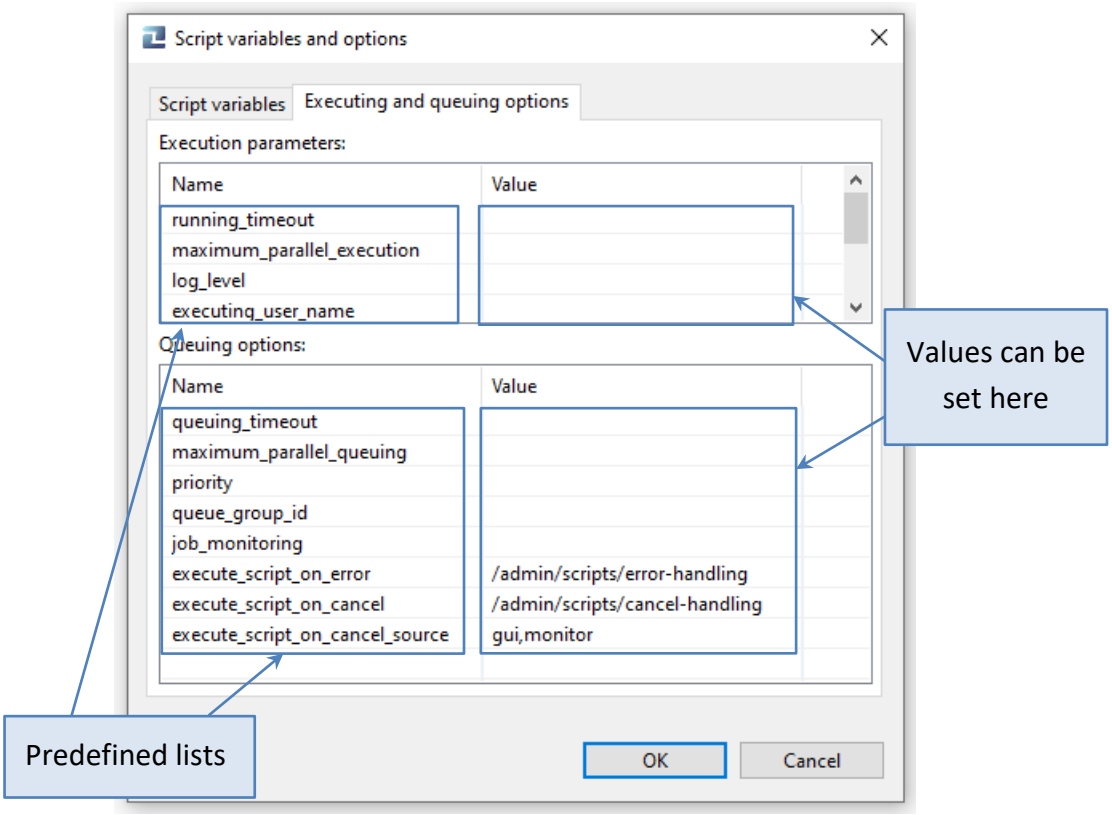


Figure 36 – The *Executing and queuing options* tab of the *Script variables and options* dialog

On the *Executing and queuing options* tab, two sets of predefined options are available. These options play a role when the script is being executed or queued, respectively. For a list of execution options, see → List of execution options.

> ***Info:*** *When the mouse cursor is moved over the options, additional help information will be shown in the tooltips.*

10.1.7.5 Script runtime information

The *Script runtime information…* context menu item (see *Figure 37.*) opens the *Script runtime info* dialog box (see *Figure 38.*), which displays details on the finished jobs of the selected script. This function is similar to the functionality of the *Finished jobs* windows, but limited to the selected script.

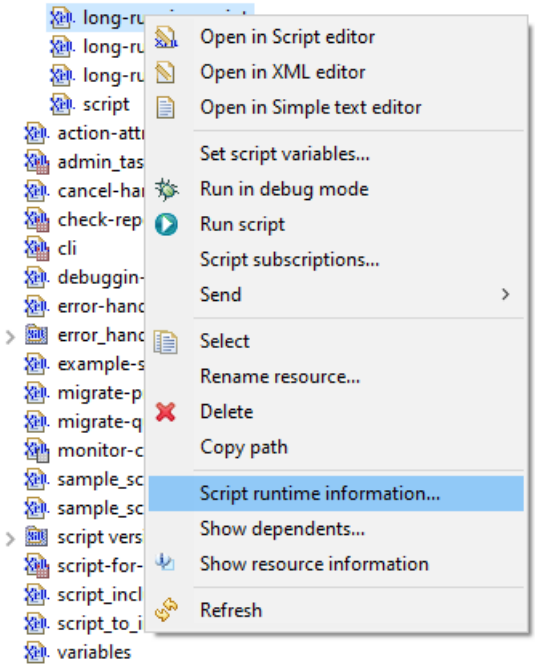


Figure 37 – The *Script runtime information...* menu item

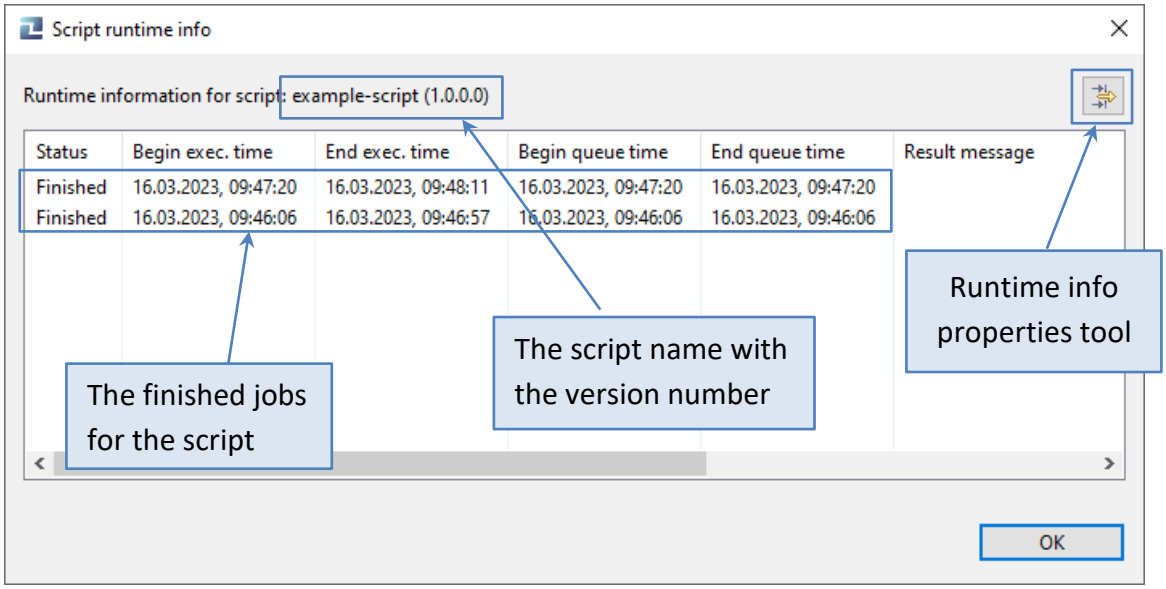


Figure 38 – The *Script runtime info* dialog box

It also gives the option to open one of the finished logs in the *Finished logs* window. The user needs to double-click one of the jobs to do this.

In order to change the selected columns for the displayed jobs that will be shown in the dialog box or change the condition filters for the result, the user needs to click on the *Runtime info properties* tool (the icon at the top-right corner). It opens another dialog, the *Runtime info properties*, see *Figure 39.*

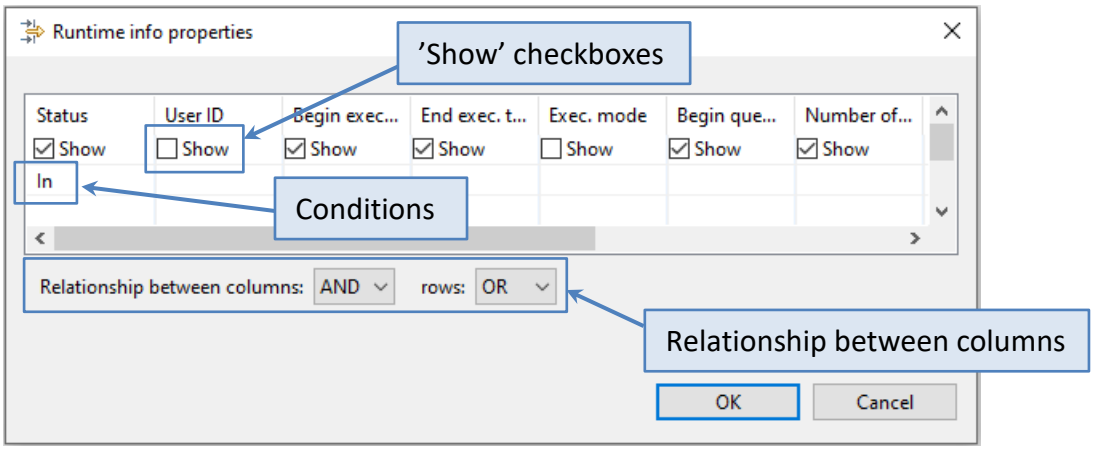


Figure 39 – The *Runtime info properties* dialog box

A *Show* checkbox located beneath each column header determines which columns will be shown from the pre-defined list of possible columns:

- *Status:* the status of the finished job
- *User ID:* the id of the user who initiated the execution of the script, see → Users in the Zagreus System
- *Begin execution time:* the timestamp when the execution of the script was started
- *End execution time:* the timestamp when the execution of the script ended
- *Execution mode:* the mode of the script execution.
- *Begin queue time:* the time when the script was queued
- *Number of lines:* the number of log lines of the job-log file, see → job-log file
- *End queue time:* the time when the script was removed from the queue and handed over for execution
- *Result message:* the output message of the script after the execution has finished. For further information, see → result-message of the script
- *Job ID:* the id of the job, this is always shown

For more details about job-related properties, see → Job properties .

The user has to click on one of the blank fields under the *Show* checkbox to specify a condition for a parameter. The *Condition parameter* window will appear after clicking on the *Add/modify condition* option. Based on the column where the user has clicked, a condition for the particular property can be set, see *Figure 40*. By clicking one of the conditions, the condition can be deleted by choosing the *Delete condition* option.

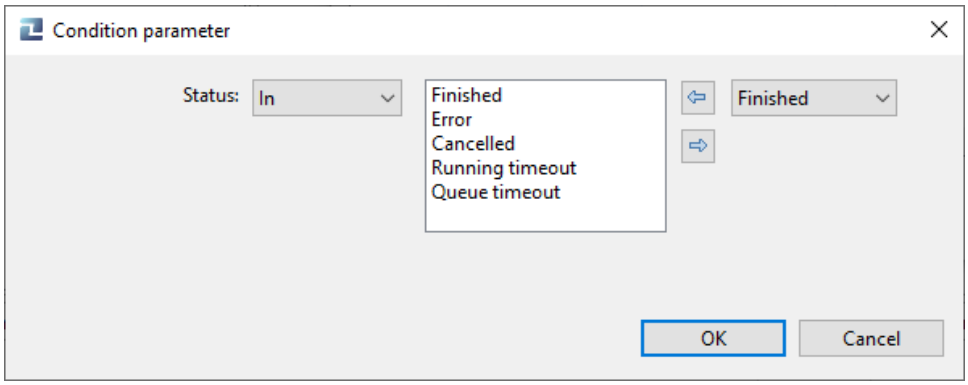


Figure 40 – The *Condition parameter* dialog box for the column 'Status'

The relationship between the condition columns and rows can be set under the table, see *Figure 39*.

This feature set is similar to the one that the *Finished jobs report parameters* provides, see →.Finished job report parameters dialog.

10.1.8 Connection-specific operations

There are several operations that can make only sense when a connection is selected in the Zagreus browser tree.

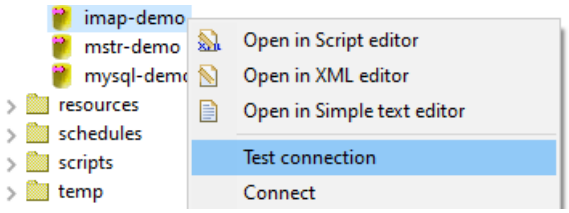


Figure 41 – The context menu of a connection resource

- *Test connection*
  It performs a connection testing on the server side, depending on the specific connection type, see → Test connection feature.

- *Connect*
  There are several connection types which support custom connection browsing inside the Zagreus browser window, see also → Opening connections in the Zagreus browser.

10.1.9 Operations for event-type resources

There are some specific operations for managing the event-type resources.

10.1.9.1 Subscriptions… menu item

As it was mentioned in the → Script subscriptions section, the *Subscriptions* dialog box can be opened by selecting the *Script subscriptions…* menu item for a selected script. The same dialog is can be opened by the *Subscriptions…* menu item for trigger-based resources (time schedule, event schedule, file trigger, mail watcher and database watcher), but with limited funcionality. The list in the dialog box displays the subscriptions related to the given trigger-based resource.

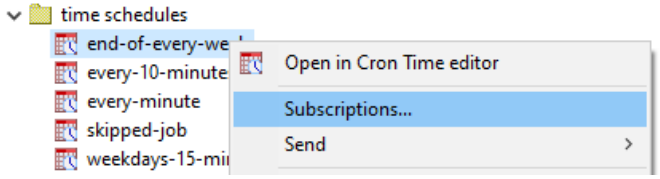


Figure 42 – The *Subscriptions…* menu item for trigger-based resources

10.1.9.2 Evaluate watcher condition… menu item

This menu item appears only for mail watchers and database watchers. They have a specific condition set up by the user who created them. This condition can be checked manually by the *Evaluate watcher conditions…* menu item from the context menu, see *Figure 43*. It performs the evaluation of the condition for the selected watcher, and shows the result in the *Evaluation results* dialog box, see → Evaluate watcher condition and → Evaluate watcher condition.

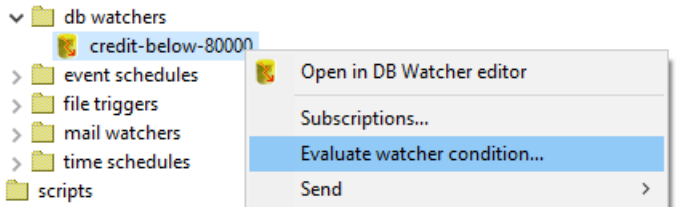


Figure 43 – The *Evaluate watcher conditions…* menu item for watchers

10.1.10 Showing dependent resources

This menu item lists all the resources that depend on the currently selected resource, see *Figure 44*. That is, such XML-based resources will be listed which contain the id or path of the selected resource in their content. So a resource has a dependent, if its id or path is present in the content of another XML-based resource.

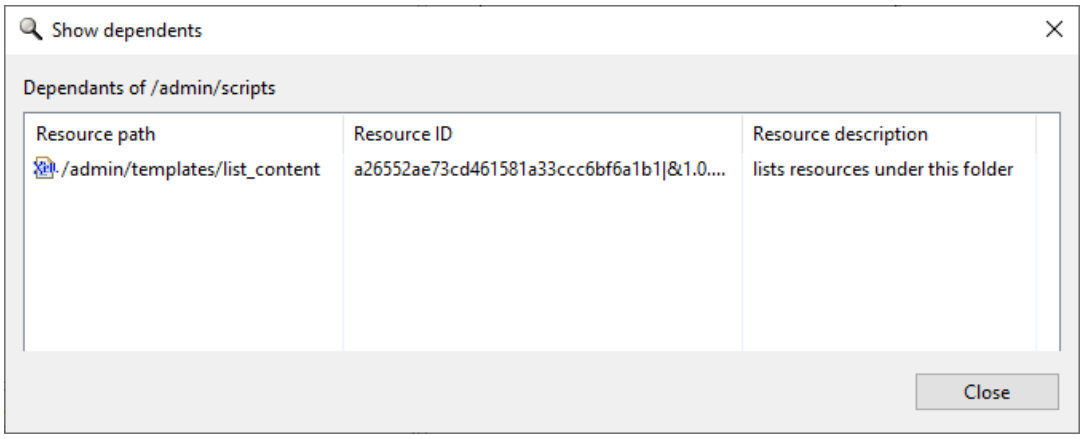


Figure 44 –The *Show dependents* dialog box

The *Show dependents* dialog box shows the following information:

- *Resource path:* the full path of the resource
- *Resource ID:* the ID of the resource
- *Resource description:* the description of the resource

By double-clicking on the resource, it can be opened in the Editor area.

10.1.11 *Send* context submenu

This context menu item is only accessible when the `.sendscripts` configuration file is set for the currently logged-in user, in the root of its home directory. In this configuration file, scripts can be specified for which certain parameters of a selected script can be sent for processing, see also → Execution with the .sendscripts file.

10.1.12 Context menu of the server definition node

For performing administrative tasks, the administrator user has several additional menu items shown in the context menu of the server definition node.

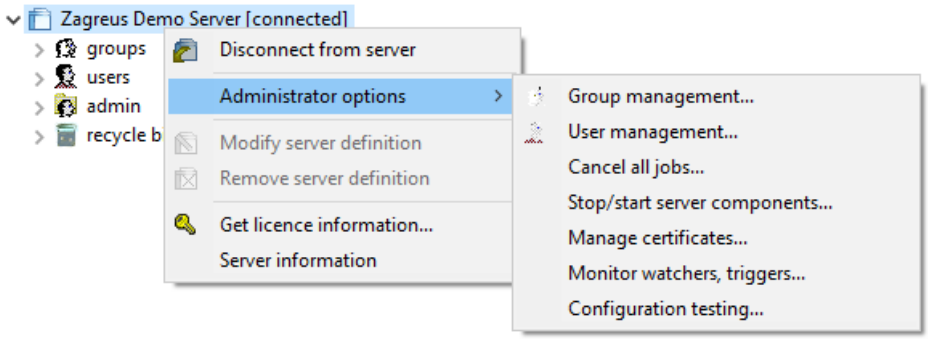


Figure 45 – The server context menu

- *Get licence information…*
  For checking and managing the licence information installed on the server, see → Licencing. This menu item is also shown when the server definition node is disconnected, see → Opening and closing a server definition node.

- *Server information*
  Displays the *Server information* dialog box, showing information about the server version and uptime, see → Server information.

Administrator options sub menus:

- *Group management…*
  It opens the *Group management for administrators* dialog box to manage groups, see → Group management.

- *User management…*
  It opens the *User management for administrators* dialog box to manage users, see → User management.

- *Cancel all jobs…*
  It opens a *Cancel jobs* dialog box for cancelling jobs by status, see → Cancel all jobs.

- *Stop / start server components*
  It opens the *Server component control* dialog for starting and stopping server components, see → Stop / start server components.

- *Monitor watchers, triggers…*
  It opens the *Monitoring watchers and triggers* dialog box for monitoring, see → Monitor watchers, triggers.

- *Configuration testing…*
  It executes the self-testing function on the server side, then opens the *Self-test result* dialog to show the results, see → Configuration testing.

10.1.13 Context menu of a user node

For performing user-related administrative tasks, the administrator user has several additional menu items shown in the context menu of the user home folder, see *Figure 46*.

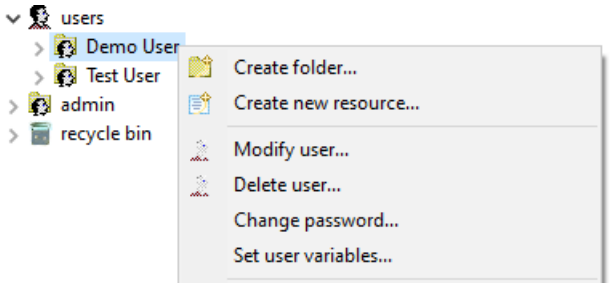


Figure 46 – The user context menu

- *Modify user…*
  It opens the *User management for administrators* dialog box, with the pre-selected *action* (*Modify existing user…*) and *user*, see → Modify existing user

- *Delete user…*
  It opens the *User management for administrators* dialog box, with the pre-selected *action* (*Delete existing user…*) and *user*, see → Delete existing user

- *Change password…*
  It opens the *Change password* dialog for the selected user, see → Changing password

- *Set user variables and options…*
  It opens the *User variables and options* dialog for the selected user. The dialog works identically as the *Script variables and options* dialog, see → Setting script variables and options

There is one more menu item which appears only in the *users* folder context menu (it only makes sense on that level): the *Create new user…* menu item (see *Figure 47.*). It opens the *User management for administrators* dialog box, with the pre-selected *action* (*Create new user…*), see → Create new user.

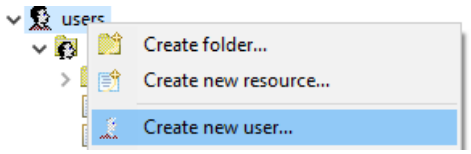


Figure 47 – The users context menu

10.1.14 Context menu of a group node

For performing group-related administrative tasks, the administrator user has several additional menu items shown in the context menu of the group home folder, see *Figure 48*.

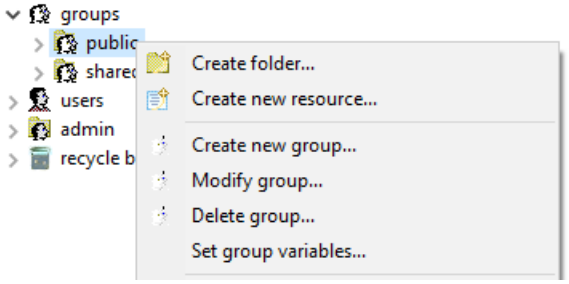


Figure 48 – The group context menu

- *Create new group…*
  It opens the *Group management for administrators* dialog box, with the pre-selected *action* (*Create new group…*). This option is for creating a sub-group, see → Create new group.

- *Modify group…*
  It opens the *Group management for administrators* dialog box, with the pre-selected *action* (*Modify existing group…*) and *group*, see → Modify existing group.

- *Delete group…*
  It opens the *Group management for administrators* dialog box, with the pre-selected *action* (*Delete existing group…*) and *group*, see → Delete existing group.

- *Set group variables and options…*
  It opens the *Group variables and options* dialog for the selected group. The dialog works identically as the *Script variables and options* dialog, see → Setting script variables and options.

There is one more menu item which appears also in the *groups* folder context menu: the *Create new group…* menu item (see *Figure 49.*). This option is for creating a base-level group. It opens the *Group management for administrators* dialog box, with the pre-selected *action* (*Create new group…*), see → Create new group.

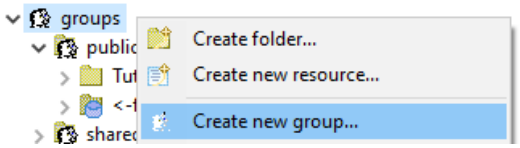


Figure 49 – The groups context menu

10.1.15 Recycle bin

The recycle bin is for storing the deleted resources temporarily. The resources can be restored or permanently deleted from the recycle bin.

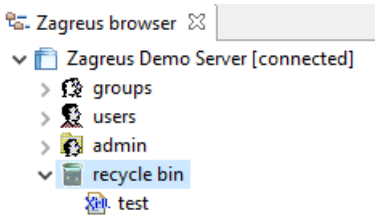


Figure 50 – A deleted resource in the recycle bin

By right-clicking on the recycle bin, a new menu item appears in the context menu: *Empty recycle bin.* By selecting this menu item, all the resources in the recycle bin will be permanently deleted.

Moreover, by right-clicking on one of the deleted resources in the recycle bin, the following menu items are available in the context menu:

- *Delete permanently:* deletes the selected resource(s) permanently
- *Recover resource:* restores the selected resource(s) to the location where it was deleted from

To check how to switch on/off the recycle bin, see → Miscellaneous properties.

10.2 Editor area

Various types of editors will open in the Editor area. The editor type corresponds to the type of the resource that the user wants to edit. The different editor types will be discussed in details in this chapter . All editors open on a tab in the editor area. Multiple editors can be open at the same time, they take as many editor tabs right next to each other as needed.

10.2.1 Script Editor

Zagreus scripts are developed by using the Script Editor. There are two main views of the Script Editor: the *Graph View* and the *XML View*. The contents of these two views are always synchronized with each other. Switching between the two views can be done by clicking on the correponding tab at the bottom of the Script Editor, see *Figure 51*. Whilst there is no functionality that is not covered by the Grapth View of the Script Editor, it can sometimes be useful to directly edit the XML code on the XML View tab.

10.2.1.1 Graph view

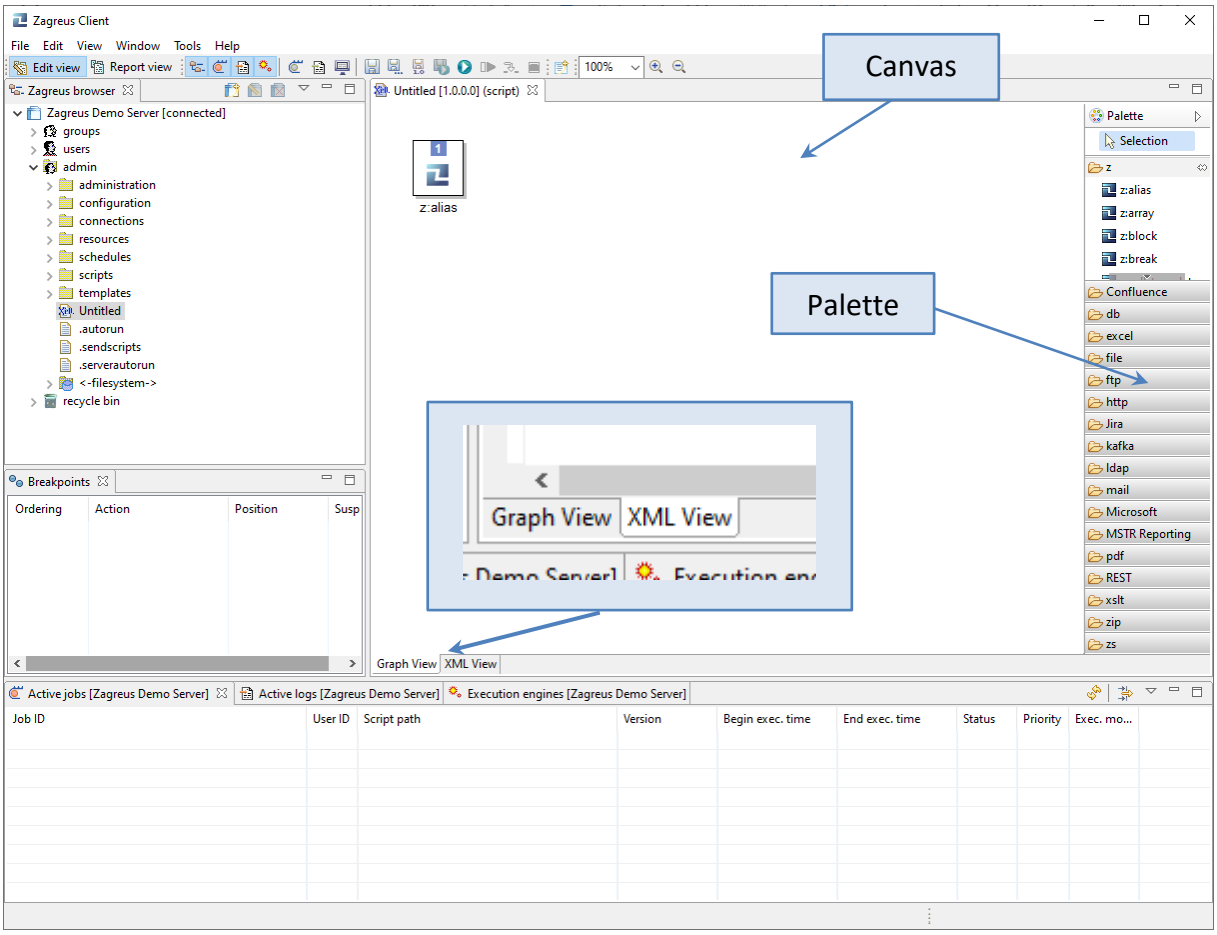


Figure 51 – The *Graph View* of the Script Editor

The Graph View is the default view when the user opens a script resource type. All functions of the Script Editor are present in the Graph View.

The functionality of the Script Editor is fully covered in a separate chapter, see → Script Editor.

10.2.1.2 XML view

The XML View can be selected by clicking on its tab at the bottom of the Script Editor right next to the Graph View tab. Basic text editing functions can be used like search (*Ctrl+F*), copy (*Ctrl+C*), paste (*Ctrl+V*), cut (*Ctrl+X*) and undo (*Ctrl+Z*).

To see the detailed description of the XML structure of a script, see → XML representation.

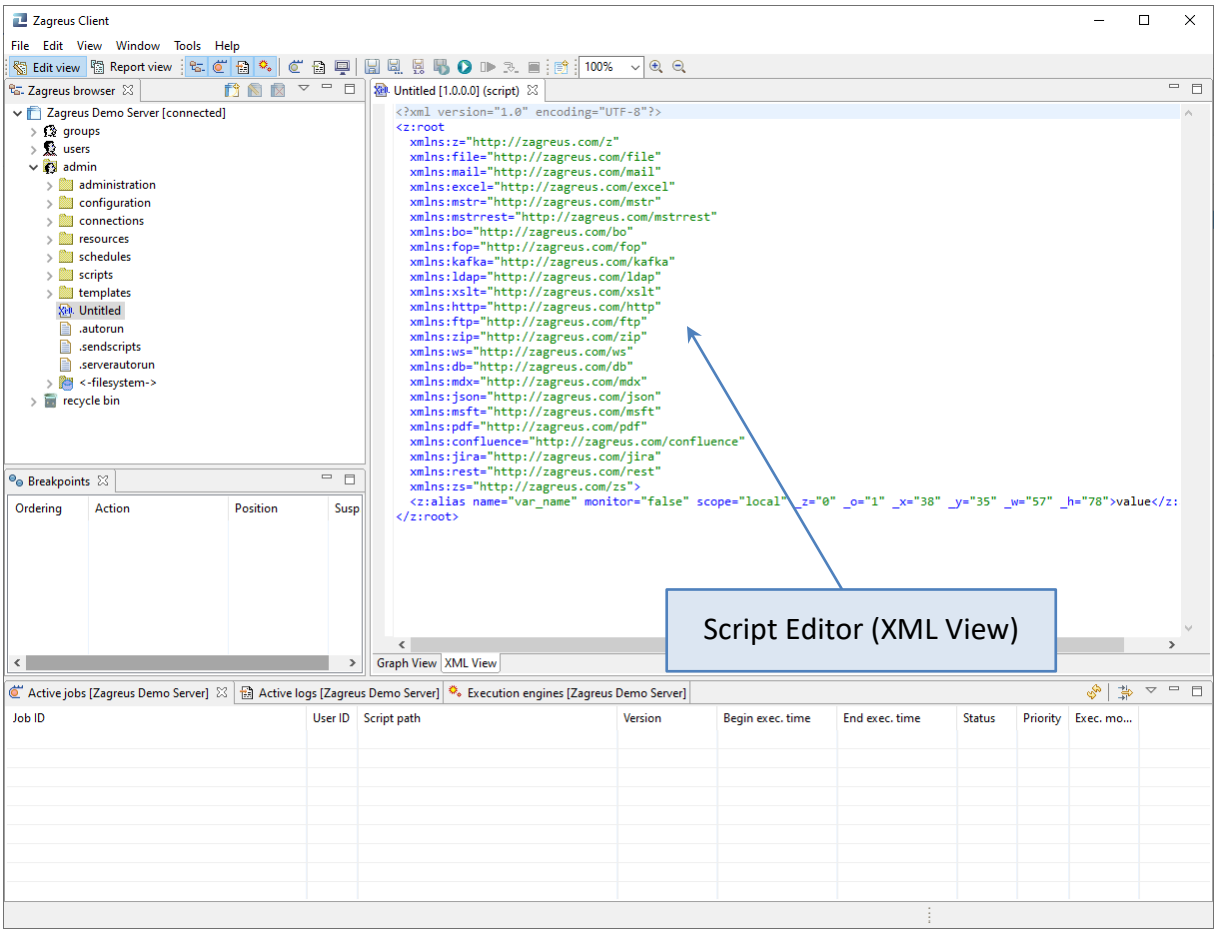


Figure 52 – The *XML View* of the Script Editor

10.2.2 Simple text editor

The Simple text editor serves to edit files with textual content. Basic text editing functions can be used like search (*Ctrl+F*), copy (*Ctrl+C*), paste (*Ctrl+V*), cut (*Ctrl+X*) and undo (*Ctrl+Z*).

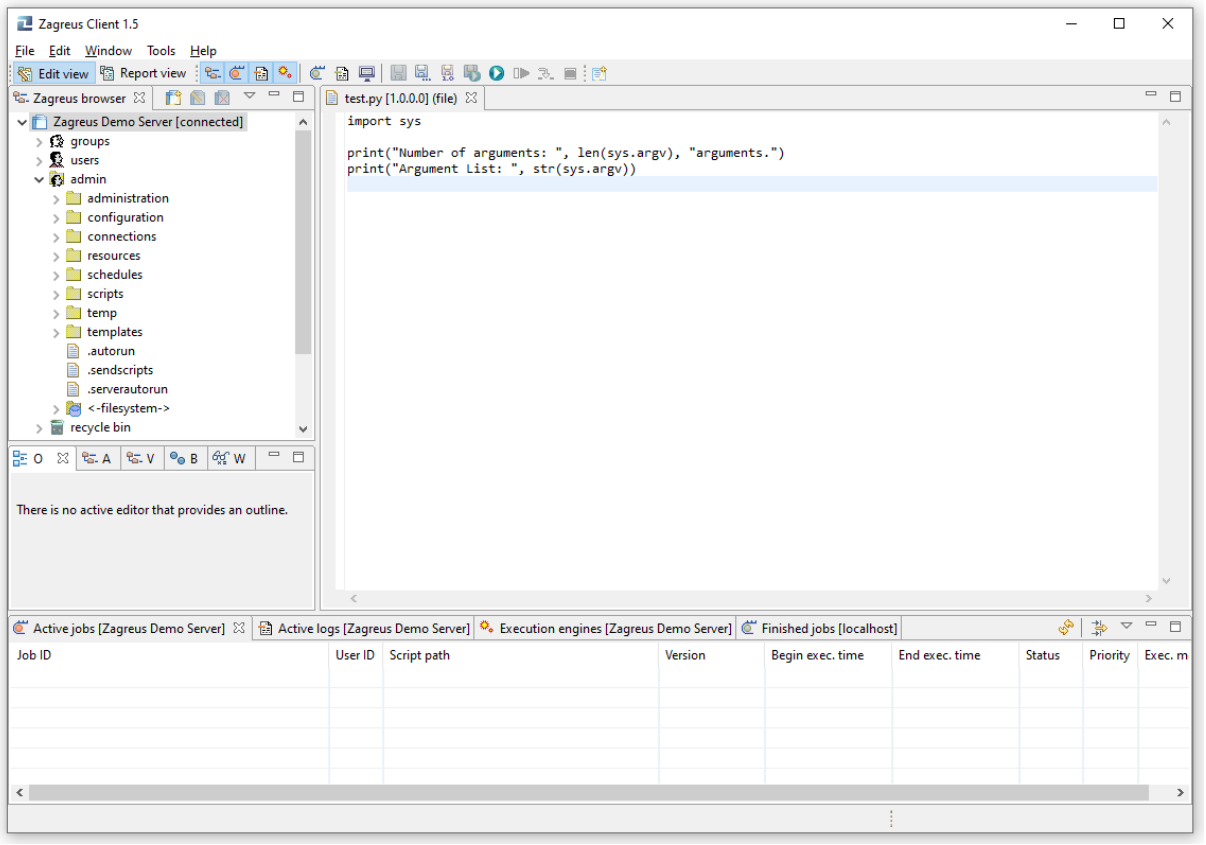


Figure 53 – The Simple text editor

10.2.3 Other editors

When the user creates a new resource or opens an existing one, a corresponding editor type opens in the editor area.

The following editor types are covered in details in other chapters:

- *Event schedule editor:* see → Event schedule
- *Time schedule editor:* see → Time schedule
- *File trigger editor:* see → File trigger
- *DB watcher editor:* see → Database watcher
- *Mail watcher editor:* see → Mail watcher

10.3 Extension windows of the Script Editor

There are a group of windows showing additional details about the script that is currently open in the Script Editor (and its tab is the currently the selected tab in the Editor Area if there are more than one script open). These windows are for overviewing the element structure of the whole script, the attributes, variables, etc. These windows only have content when any script is opened.

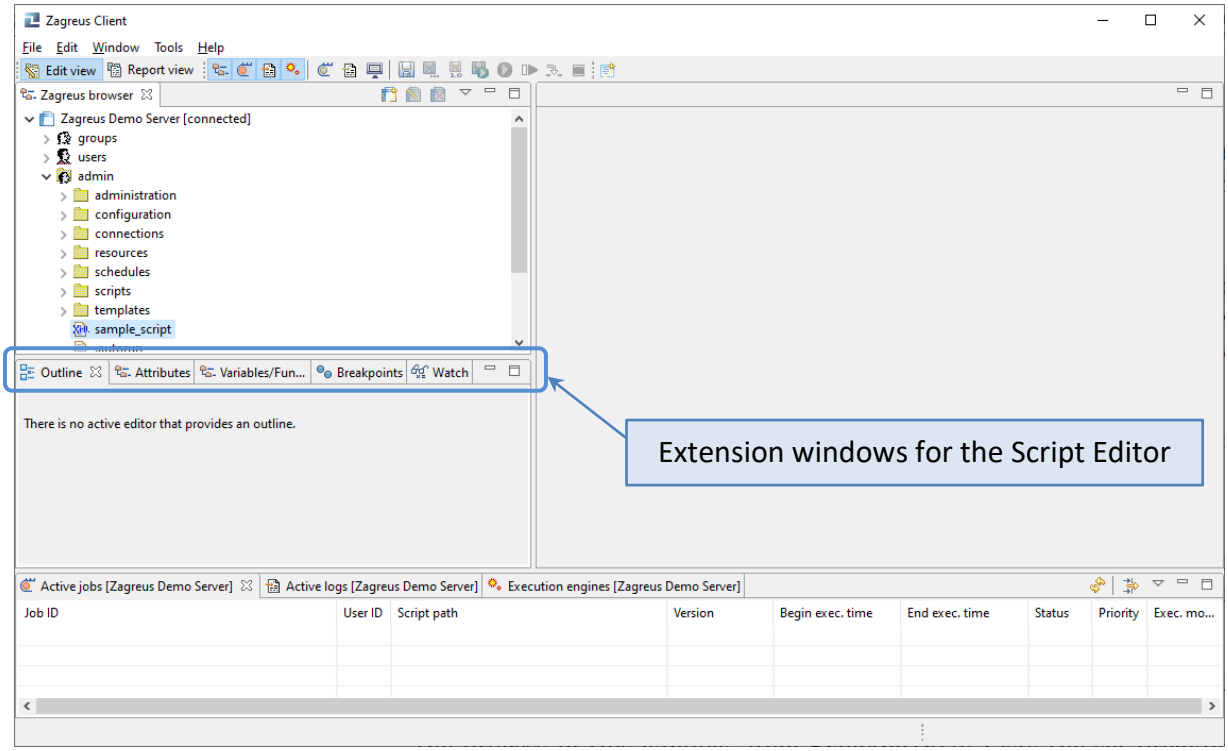


Figure 54 – Extension windows

10.3.1 Outline window

The *Outline window* makes it easier to view large scripts with deeply nested action structure. If this window is not shown, it can be opened by clicking on the *Outline window* option on the *Window* main menu bar. In the Outline window, the actions of the script are listed under each other sorted by the action order number. The actions listed here are expandable if there is any content of the particular action: text element or other child actions. So the action hierarchy of the original script is mapped in a tree-like structure in the Outline window, see *Figure 55*.

The Outline window is synchronized with the Script Editor: the selection in the Outline window is corresponding with that of the Script Editor. A context menu appears by right-clicking on one of the actions. This is the same context menu that is displayed by right-clicking on the same action in the Script Editor.

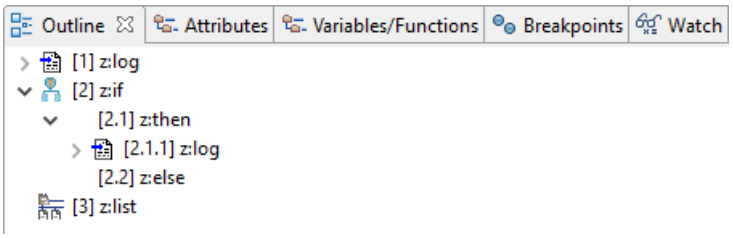


Figure 55 – The Outline window

10.3.2 Attributes window

The *Attributes window* displays the attributes of the selected action. If this window is not shown, it can be opened by clicking on the *Attributes window* menu item on the *Window* main menu bar. The attributes can be edited by double-clicking on the attribute name. If the currently selected action has no attribute at all or none of the actions are selected, there will be no attributes shown in the Attributes window.

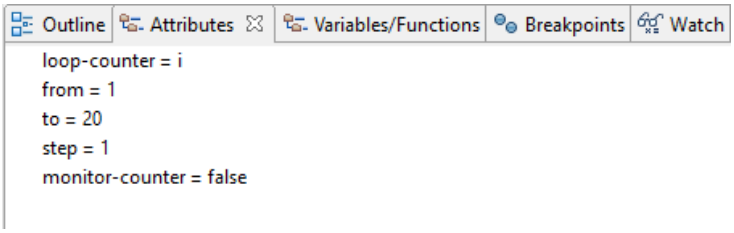


Figure 56 – The Attribute window

10.3.3 Variables / Functions window

The *Variables/Functions window* displays the generally usable engine system variables and functions (see → Function calls), the script starting variables along with the variables defined in the script. If this window is not shown, it can be opened by clicking on the *Variables window* item on the *Window* main menu bar. The items in this window are categorized into groups, and all these groups are expandable. Furthermore, tooltips help to give information about the usage of functions and variables.

Groups:

- *System variables*
  The user can check the list of the predefined system variables

- *System functions*
  The list of the predefined engine system functions. If the category toggle button is set to *Categories presentation*, the list of the functions are arranged into further categories (see below).

- *Script starting variables*
  The list of the variables for the script defined in the *Script variables and options* dialog box (see also → Setting script variables and options). When a script is running, other variables are actually reachable for the script, but because this window belongs to the Script Editor itself (so the script is likely not currently running), it can only know the closest scope of variables at this point in time, which is the script scope.

- *Script variables*
  It shows the variables defined in the script (see also → Start-up variables). It shows these variables according to the current point of selection. So if a variable is defined later than the actual selection, the variable is not shown in the list. This list also takes the variable scope (local or global) setting into account. For variable scopes, see → Variable scopes

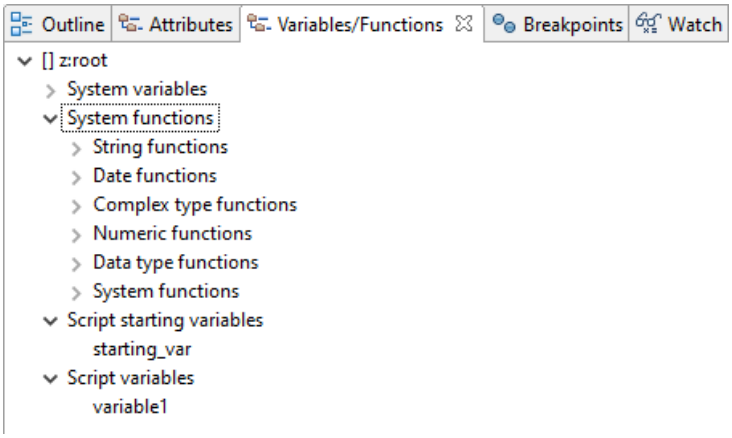


Figure 57 – Variables/Functions window, the categories of the engine system functions

It is possible to display the engine functions in *categories* or in *flat* (simple list) presentation. To switch between presentations, the user needs to click on the icon located in the upper right corner of this window, or select the corresponding tool menu item right next to the icon.

The categories presentation lists the following categories:

- String functions
- Date functions
- Complex type functions
- Numeric functions
- Data type functions
- System functions

Variables and functions can be added to the textual content of an action in the Script editor by drag and dropping the particular element onto the text content of that action. For this, the textual content of the target action should not be empty, see *Figure 58.* and *Figure 59*.

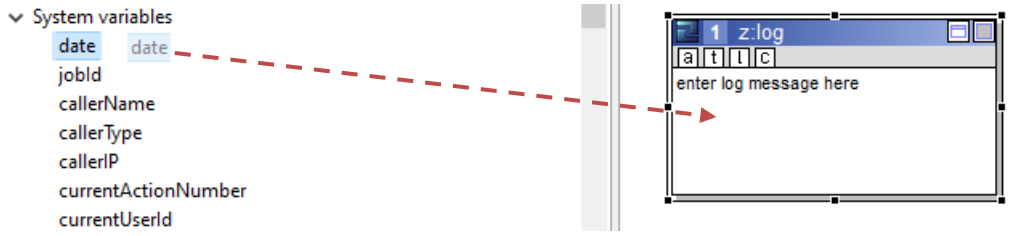


Figure 58 – The *date* engine system variable is started to drag to the `z:log` action…

Figure 59 – …and after dropping onto the text element of the `z:log` action

10.3.4 Breakpoints window

The *Breakpoints window* is designed for the case when the script is running in debug mode. This sort of running makes sense only if the script is opened in the Script Editor (debug mode) as long as the script is being executed. The Breakpoints window is linked to the debug mode Script Editor. Therefore, the usage of this window is described in details in the chapter about the debug mode, see → Starting a debug session

10.3.5 Watch window

The *Watch window* is designed for the case when the script is running in debug mode. This sort of running makes sense only if the script is opened in the Script Editor (debug mode) as long as the script is being executed. The Watch window is linked to

the debug mode Script Editor. Therefore, the usage of this window is described in details in the chapter about the debug mode, see → Starting a debug session.

10.4 Monitoring

Monitoring is an important part of the workflow in Zagreus. The monitoring options include the ability to follow script execution, checking the logs of the running script as well as seeing the actual states of the execution engines and the Worker Controller.

The monitoring windows are located at the bottom of the Zagreus Client in the default Edit view. The monitoring windows are the following:

- *Active jobs*
- *Active logs*
- *Execution engines*
- *Finished jobs*
- *Finished logs*
- *Skipped jobs*

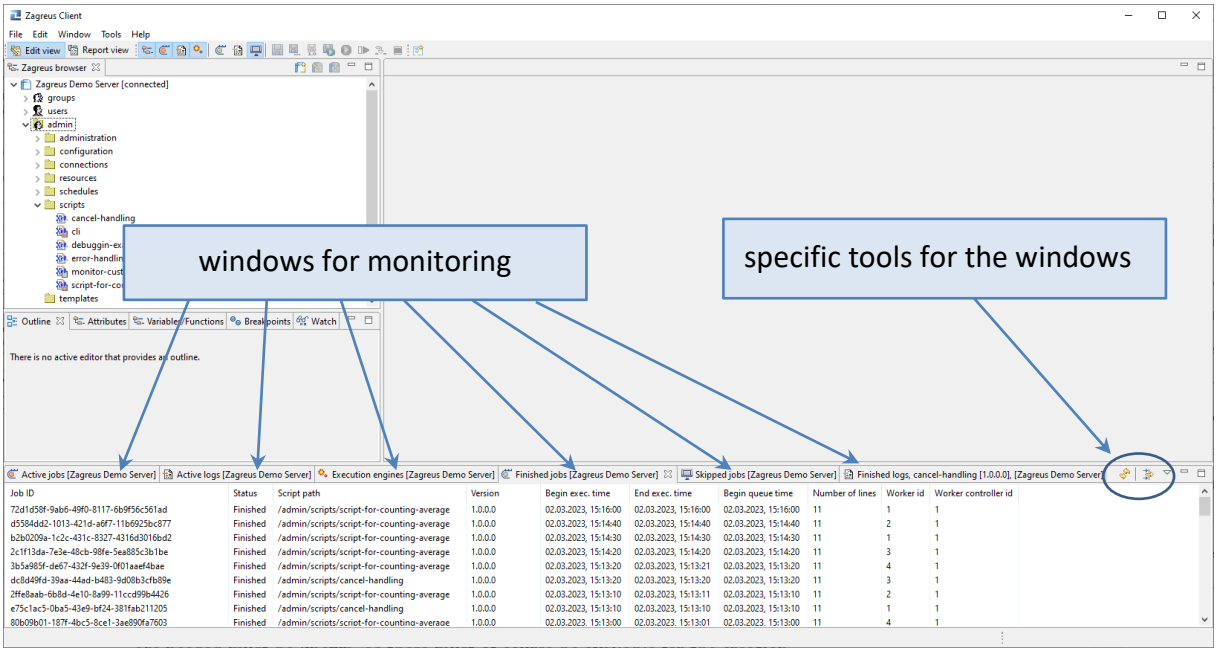


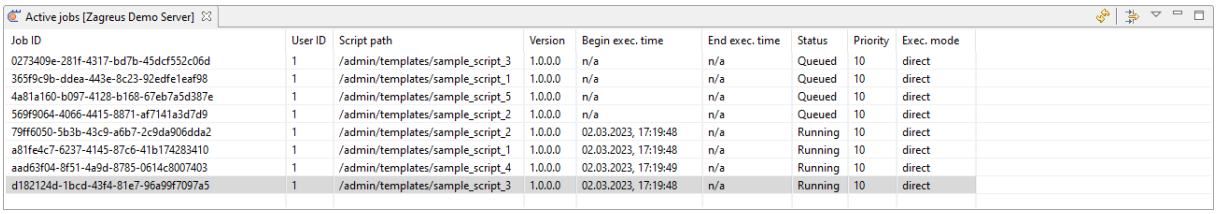
Figure 60 – The monitoring windows and their tools

10.4.1 Active jobs window

The jobs that are currently queued or running can be found in the *Active jobs* window. For the lifecycle of a job, see → Job lifecycle

The header of the Active jobs window has a postfix (the part in *[…]*). It shows the connected server name when there is an active open server connection (e.g. *Zagreus Demo Server* in *Figure 61.*). If there are multiple open connections, it shows the active

one which is selected by clicking on any of the tree sub-nodes of the particular server in the Zagreus Browser window, see → Zagreus browser window


Active jobs [Zagreus Demo Server]

Job ID	User ID	Script path	Version	Begin exec. time	End exec. time	Status	Priority	Exec. mode
0273409e-281f-4317-bd7b-45dcf552c06d	1	/admin/templates/sample_script_3	1.0.0.0	n/a	n/a	Queued	10	direct
365f9c9b-ddea-443e-8c23-92edfe1eaf98	1	/admin/templates/sample_script_1	1.0.0.0	n/a	n/a	Queued	10	direct
4a81a160-b097-4128-b168-67eb7a5d387e	1	/admin/templates/sample_script_5	1.0.0.0	n/a	n/a	Queued	10	direct
569f9064-4066-4415-8871-af7141a3d7d9	1	/admin/templates/sample_script_2	1.0.0.0	n/a	n/a	Queued	10	direct
79ff6050-5b3b-43c9-a6b7-2c9da906dda2	1	/admin/templates/sample_script_2	1.0.0.0	02.03.2023, 17:19:48	n/a	Running	10	direct
a81fe4c7-6237-4145-87c6-41b174283410	1	/admin/templates/sample_script_1	1.0.0.0	02.03.2023, 17:19:48	n/a	Running	10	direct
aad63f04-8f51-4a9d-8785-0614c8007403	1	/admin/templates/sample_script_4	1.0.0.0	02.03.2023, 17:19:49	n/a	Running	10	direct
d182124d-1bcd-43f4-81e7-96a99f7097a5	1	/admin/templates/sample_script_3	1.0.0.0	02.03.2023, 17:19:48	n/a	Running	10	direct

Figure 61 – The Active jobs window

The columns that are shown in the Active Jobs window are:

- *Job ID*
  The ID of the job

- *User ID*
  The ID of the user who initiated the execution of the script

- *Script path*
  The full path of the script

- *Version*
  The version number of the script, see → Resource ID and version

- *Begin exec. time*
  The begin execution time of the job.

- *End exec. time*
  The end execution time of the job.

- *Status*
  The current status of the job.

- *Priority*
  The priority number of the job. The smaller the number, the higher the priority is. The default priority number is 10, if is not set otherwise in the *priority* execution option, see → List of execution options

- *Exec. mode*
  The execution mode of the job.

- *Script name* (not shown by default)
  The name of the script.

- *Begin queue time* (not shown by default)
  The begin queue time of the job.

- *Number of lines* (not shown by default)
  The full number of log lines.

For the list of job properties, see → Job properties

Specific tools for the Active jobs window:

- *Refresh View*
  The Active jobs window updates its content every 5 seconds by default. It can be refreshed manually with this tool.

- *Active jobs window preferences*
  It opens the *Active jobs window preferences* dialog box, see below.

10.4.1.1 Active jobs window preferences dialog

This dialog allows the user to select the property columns which will appear in the Active jobs window, see *Figure 62*.

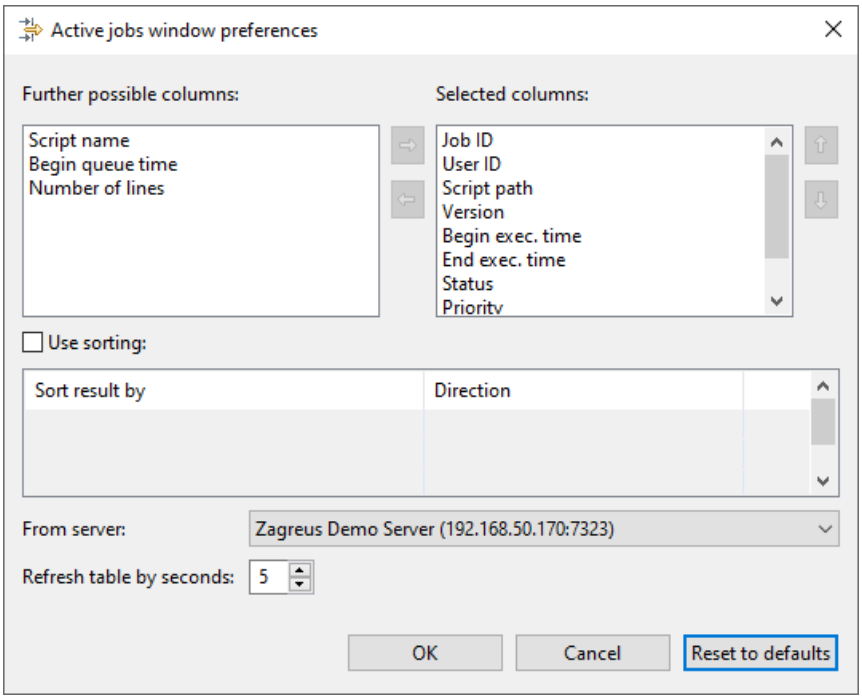


Figure 62 – The *Active jobs windows preferences* dialog box

The sections of the dialog are the following:

- *Selected columns*
  The list of the column names displayed in the Active jobs window. The order of the columns can be modified by selecting a column name and using one of the arrows to the right of the list box (up and down arrows).
  Removing a column is also possible by selecting a column name and using the left arrow icon, located to the left of the list box.

- *Further possible columns*
  The list of the column names which are not in the *Selected columns* list box, but possibly selectable.
  To select a possible column, select a column name and click on the right arrow icon, located to the right of the list box.

- *Use sorting*
  When this option is checked, the listed job results can be sorted by the specified column names. To specify such a sorting column, the user needs to select the column name from the dropdown box which appears by clicking on the first empty cell of the first column (*Sort result by*). The direction of the selected column (*ascending* or *descending*) can be set in the same manner in the second column (*Direction*), see *Figure 63*.

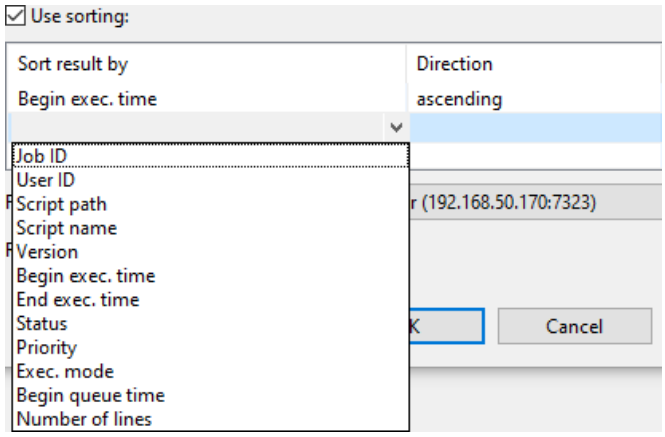


Figure 63 – Defining a sorting column name in the *Active jobs window preferences* dialog box

- *From server*
  If there are multiple open server connections, the target server can be selected from this dropdown list. After changing the server, by clicking on the OK button, the header of the Active jobs window changes correspondingly.

- *Refresh table by seconds*
  This is the polling rate that is used to retrieve data periodically from the target server. Default is 5 seconds.

The changes are applied after clicking on the *OK* button. In order to reset all the settings, the user has to click on the *Reset to defaults* button.

10.4.2 Active logs window

The *Active logs window* displays the actual real-time logging output during the execution of scripts, see *Figure 64*. If more than one script is running, multiple job-log outputs are displayed simultaneously in this window in different tabs.

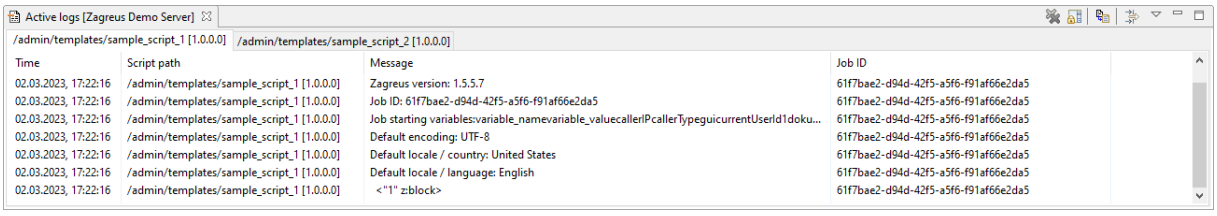


Figure 64 – The *Active logs* window

The log tabs are automatically opened when the first log message arrives from a job and stay open for ten more seconds after the last message.

The columns that are shown in the Active logs window:

- *Time*
  The timestamp of the log message in a <dd.MM.yyyy, HH:mm:ss> format.

- *Script path*
  The full path and version number of the script.

- *Message*
  The job-log message, see logging → job-log file

- *Job ID*
  The ID of the job.

- *User name* (not shown by default)
  The user ID who initiated the script execution

- *Job status* (not shown by default)
  The status of the job.

Specific tools for the *Active logs* window:

- *Close all log tabs* 
  Immediately closes all log tabs.

- *Scroll lock* 
  It stops the automatic scrolling down.

- *Show log result in text editor* 
  The content of the log tab will be opened in a Simple text editor. Only the selected tab content will be displayed and it will not be refreshed.

- *Active logs window preferences*
  It opens the *Active logs window preferences* dialog box, see below.

10.4.2.1 Active logs window preferences dialog

This dialog allows the user to select the property columns which will appear in the Active logs window, see *Figure 65*.

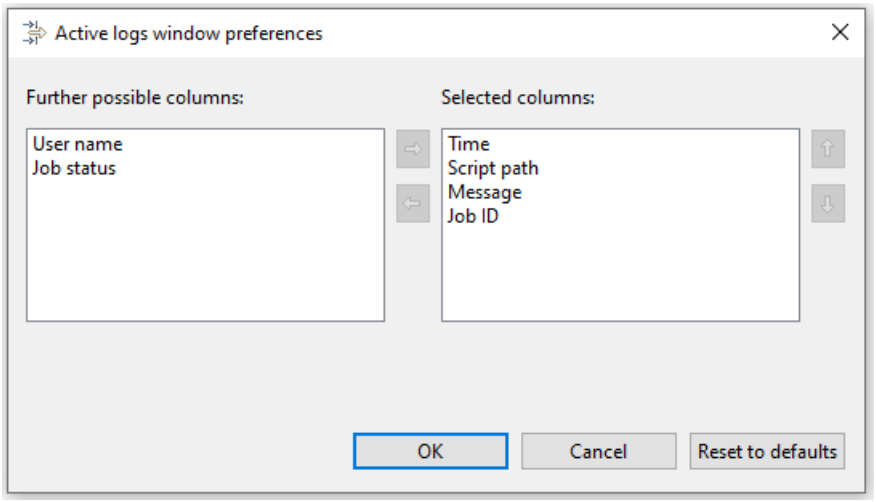


Figure 65 – The *Active logs window preferences* dialog

The sections of the dialog are the following:

- *Selected columns*
  The list of the column names displayed in the Active logs window. The order of the columns can be modified by selecting a column name and using one of the arrows to the right of the list box (up and down arrows).
  Removing a column is also possible by selecting a column name and using the left arrow icon, located to the left of the list box.

- *Further possible columns*
  The list of the column names which are not in the *Selected columns* list box, but possibly selectable.
  To select a possible column, select a column name and click on the right arrow icon, located to the right of the list box.

The changes are applied after clicking on the *OK* button. In order to reset all the settings, the user has to click on the *Reset to defaults* button.

10.4.3 Execution engines window

The *Execution engines window* displays the current status of all the Zagreus Workers (see → Zagreus Worker).

There are two tabs in this window: the *Worker information* tab and the *Worker-controller logs* tab. In the first one, worker controllers and workers are displayed while in the second one, the logs are shown from the Zagreus Worker Controller, see → Zagreus Worker-Controller.

10.4.3.1 Worker information tab

The *Worker information* tab is selected by default in the window. This tab offers a real-time monitoring about execution processes. The Zagreus Workers are shown in a table widget that has collapsable nodes (Zagreus Worker Controllers) apart from the usual table rows (Zagreus Workers). So the Zagreus Workers are batched together under a Worker Controller, see *Figure 66*. The number of workers is determined by the number of execution engines included in the license (see → Licencing).

Zagreus provides the usage of multiple Worker Controllers, though in most of the cases only one Worker Controller is configured in the Zagreus System.

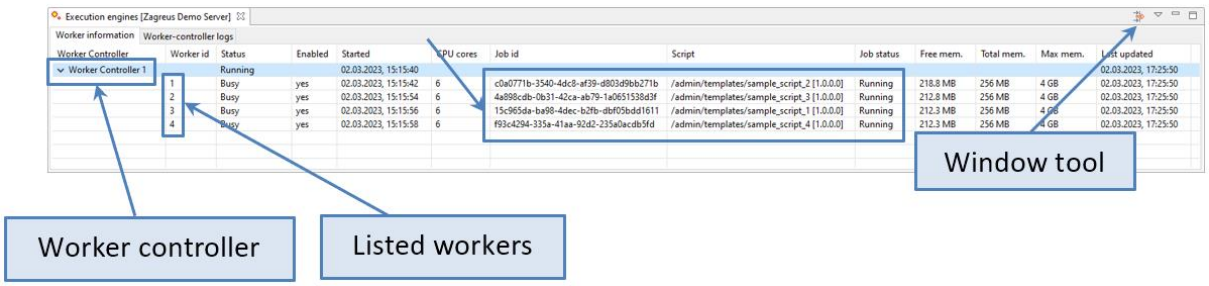


Figure 66 – The *Execution engines* window

The columns for the Worker information tab of the Execution engines window are:

- *Worker Controller*
  The Worker Controller instance with its ID, see → Zagreus Worker-Controller

- *Worker id*
  The id of the worker, see → ID of the Zagreus Worker

- *Status*
  The status of the Worker Controller or Worker (depending on the row). The status can be *Idle*, *Busy*, *Starting* and *Shutting down* in the case of Workers, and *Running*, *Starting*, *Suspended* and *Shutting down* in the case of the Worker Controller.

- *Enabled*
  Shows whether the Zagreus Worker is enabled or disabled

- *Started*
  Showing the timestamp when the Zagreus Worker has started.

- *CPU cores*
  The number of CPU cores of the host the Zagreus Worker JVM runs on.

- *Job id*
  The ID of the job that is currently executed on the given Zagreus Worker

- *Script*
  The name and version number of the script associated with the job that is currently executed on the given Zagreus Worker

- *Job status*
  The status of the job.

- *Free mem.*
  The currently free memory that the Zagreus Worker JVM can use. However, it does not mean this is the maximum available memory because the JVM allocates memory incrementally. So, if free memory is too small, JVM will allocate more memory until it reaches the maximum memory.

- *Total mem.*
  The actual memory that the Zagreus Worker JVM is using. It can be higher than expected even when the JVM is in *idle* status. JVM-s are using their own garbage collector mechanism to free up memory when it is needed. The total memory for a worker can be specified with the `-Xms` setting in the Zagreus Worker configuration (see also → Worker startup properties).

- *Max mem.*
  This is the maximum memory that the Zagreus Worker JVM can use. The total memory for a worker can be specified with the `-Xmx` setting in the Zagreus Worker configuration (see also → Worker startup properties).

- *Last updated*
  The timestamp when all information was last updated.

Right-clicking on one of the Worker Controllers will open a context menu, see *Figure 67.*

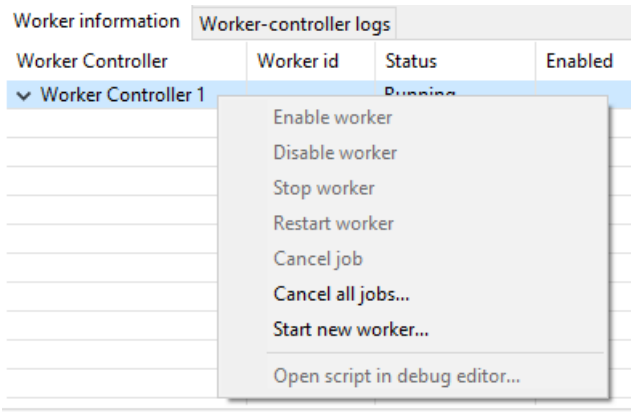


Figure 67 – The context menu of the Worker Controller node

Here, the user can perform one of the following operations:

- *Cancel all jobs…*
  It opens the *Cancel jobs* dialog box. This is the same dialog box that the *Administrator options / Cancel all jobs…* menu item opens by the server definition node context menu in the Zagreus browser window, see → Cancel all jobs.

- *Start new worker…*
  It opens the *Start new worker* dialog box, see *Figure 68.* A new worker can be started with the specified worker id. The worker id must be a unique number which is not among the currently running workers, see → ID of the Zagreus Worker.

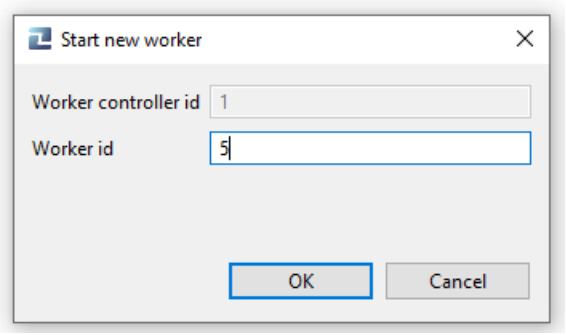


Figure 68 – The *Start new worker* dialog box

Right-clicking on one of the Workers opens another context menu, see *Figure 69.*

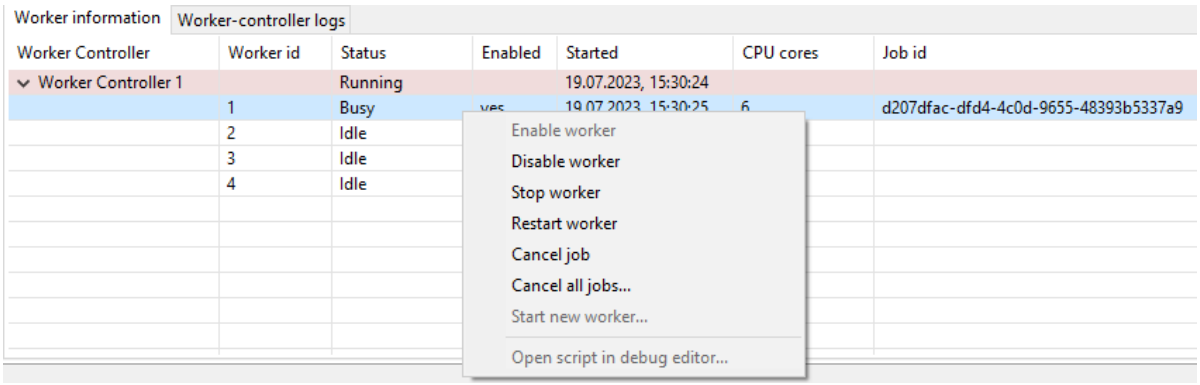


Figure 69 – The context menu of a Worker node

Here, the user can perform one of the following operations:

- *Enable worker*
  It enables the selected worker

- *Disable worker*
  It disables the selected worker

- *Stop worker*
  It stops the selected worker. After stopping, the selected worker is removed from the worker list. If there was a running job, this operation cancels the job first.

- *Restart worker*
  It restarts the selected worker. First, the selected worker is removed from the worker list, but shortly it appears again.

- *Cancel job*
  It cancels the currently executed job on the worker, see → Cancellation.

- *Cancel all jobs…*
  It opens the *Cancel jobs* dialog box. This is the same dialog box that the *Administrator options / Cancel all jobs…* menu item opens by the server definition node context menu in the Zagreus browser window, see → Cancel all jobs.

- *Open script in debug editor…*
  This menu item is only enabled when the status of the currently executed job is *Debugging*. It opens the currently executed job in a Script Editor (debug mode), see → Debug Editor.

For further information about Zagreus Worker management, see → Managing Zagreus Workers

There is one tool available for the Execution engines window: the *Engine status windows properties* icon. This opens the *Engine status window preferences* dialog, see *Figure 70*.

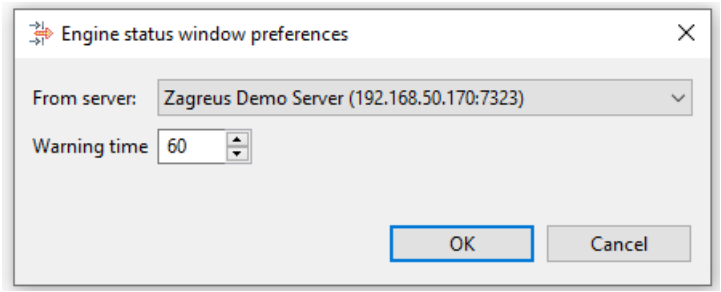


Figure 70 – The *Engine status window preferences* dialog

Here, the user can adjust the following parameters:

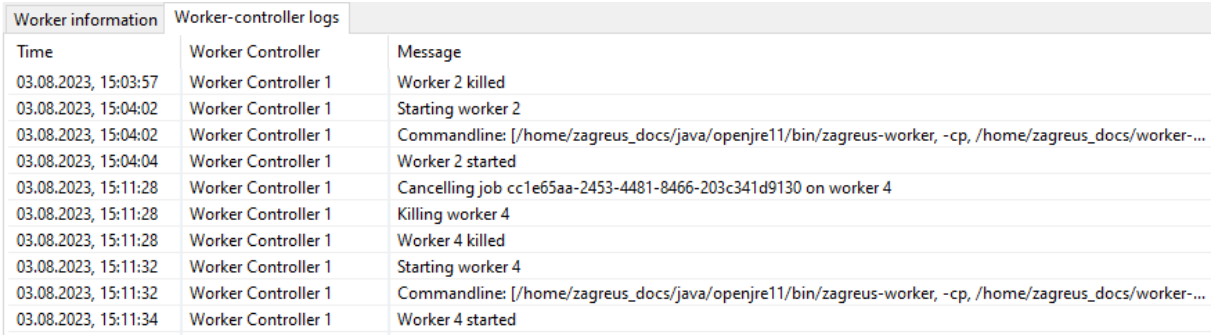
- *From server*
  If there are multiple open server connections, the target server can be selected from this dropdown list. After changing the server, by clicking on the OK button, the header of the Execution engines window changes correspondingly

- *Warning time*
  When the content of the Worker information tab is not updated within a time threshold (e.g. the connection is lost, the *Last updated* column is not changing

any more), the affected rows become red. This threshold can be set here, specified in seconds.

10.4.3.2 Worker-controller logs tab

This tab displays direct log messages from the Worker Controller, see *Figure 71*.


Worker information | Worker-controller logs

Time	Worker Controller	Message
03.08.2023, 15:03:57	Worker Controller 1	Worker 2 killed
03.08.2023, 15:04:02	Worker Controller 1	Starting worker 2
03.08.2023, 15:04:02	Worker Controller 1	Commandline: [/home/zagreus_docs/java/openjre11/bin/zagreus-worker, -cp, /home/zagreus_docs/worker-...
03.08.2023, 15:04:04	Worker Controller 1	Worker 2 started
03.08.2023, 15:11:28	Worker Controller 1	Cancelling job cc1e65aa-2453-4481-8466-203c341d9130 on worker 4
03.08.2023, 15:11:28	Worker Controller 1	Killing worker 4
03.08.2023, 15:11:28	Worker Controller 1	Worker 4 killed
03.08.2023, 15:11:32	Worker Controller 1	Starting worker 4
03.08.2023, 15:11:32	Worker Controller 1	Commandline: [/home/zagreus_docs/java/openjre11/bin/zagreus-worker, -cp, /home/zagreus_docs/worker-...
03.08.2023, 15:11:34	Worker Controller 1	Worker 4 started

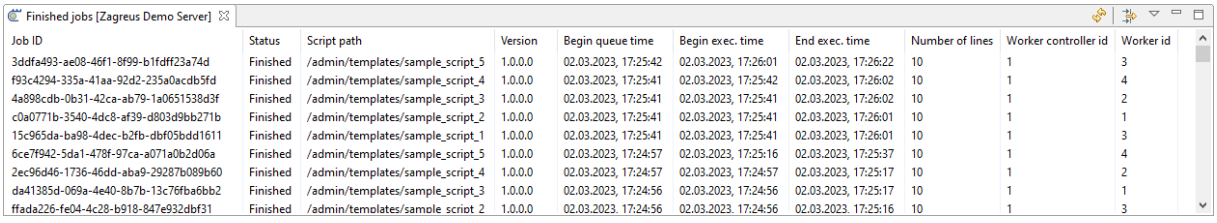
Figure 71 – The *Worker-controller logs* tab of the *Execution engines* window

The columns of this table are the following:

- *Time:* the timestamp of the log message.
- *Worker Controller:* the ID of the Worker Controller.
- *Message:* the message from the Worker Controller.

10.4.4 Finished jobs window

The *Finished jobs* window displays report-like information about completed jobs, see *Figure 72*. This window is not opened by default, it can be opened either by clicking on the *Open new Finished jobs window* tool in the main toolbar (see also → Open a new Finished jobs window), or pressing the *Alt+J* key combination, or accessed by selecting the *Finished jobs window* option from the *Windows* main menu bar.


Finished jobs [Zagreus Demo Server]

Job ID	Status	Script path	Version	Begin queue time	Begin exec. time	End exec. time	Number of lines	Worker controller id	Worker id
3ddfa493-ae08-46f1-8f99-b1fdff23a74d	Finished	/admin/templates/sample_script_5	1.0.0.0	02.03.2023, 17:25:42	02.03.2023, 17:26:01	02.03.2023, 17:26:22	10	1	3
f93c4294-335a-41aa-92d2-235a0acdb5fd	Finished	/admin/templates/sample_script_4	1.0.0.0	02.03.2023, 17:25:41	02.03.2023, 17:25:42	02.03.2023, 17:26:02	10	1	4
4a898cdb-0b31-42ca-ab79-1a0651538d3f	Finished	/admin/templates/sample_script_3	1.0.0.0	02.03.2023, 17:25:41	02.03.2023, 17:25:41	02.03.2023, 17:26:02	10	1	2
c0a0771b-3540-4dc8-af39-d803d9bb271b	Finished	/admin/templates/sample_script_2	1.0.0.0	02.03.2023, 17:25:41	02.03.2023, 17:25:41	02.03.2023, 17:26:01	10	1	1
15c965da-ba98-4dec-b2fb-dbf05bdd1611	Finished	/admin/templates/sample_script_1	1.0.0.0	02.03.2023, 17:25:41	02.03.2023, 17:25:41	02.03.2023, 17:26:01	10	1	3
6ce7f942-5da1-478f-97ca-a071a0b2d06a	Finished	/admin/templates/sample_script_5	1.0.0.0	02.03.2023, 17:24:57	02.03.2023, 17:25:16	02.03.2023, 17:25:37	10	1	4
2ec96d46-1736-46dd-aba9-29287b089b60	Finished	/admin/templates/sample_script_4	1.0.0.0	02.03.2023, 17:24:57	02.03.2023, 17:24:57	02.03.2023, 17:25:17	10	1	2
da41385d-069a-4e40-8b7b-13c76fba6bb2	Finished	/admin/templates/sample_script_3	1.0.0.0	02.03.2023, 17:24:56	02.03.2023, 17:24:56	02.03.2023, 17:25:17	10	1	1
ffada226-fe04-4c28-b918-847e932dbf31	Finished	/admin/templates/sample_script_2	1.0.0.0	02.03.2023, 17:24:56	02.03.2023, 17:24:56	02.03.2023, 17:25:16	10	1	3

Figure 72 – The *Finished jobs* window

The columns that are shown in the Finished jobs window are:

- *Job ID*

The ID of the job

- *Status*
  The status of the job

- *User ID* (not shown by default)
  The ID of the user who initiated the execution of the script

- *Script path*
  The full path of the script

- *Script name* (not shown by default)
  The name of the script

- *Version*
  The version number of the script, see → Resource versioning

- *Begin exec. time*
  The begin execution time of the job

- *End exec. time*
  The end execution time of the job

- *Exec. mode* (not shown by default)
  The execution mode of the job

- *Begin queue time*
  The begin queue time of the job

- *End queue time* (not shown by default)
  The end queue time of the job

- *Result message* (not shown by default)
  The result message of the job, see → result-message of the script

- *Caller* (not shown by default)
  The caller of the job

- *Caller type* (not shown by default)
  The caller type of the job

- *Number of lines*
  The full number of log lines of the job-log file, see → job-log file

- *Worker id*
  The ID of the Zagreus Worker where the job was executed on (see also → ID of the Zagreus Worker)

- *Worker controller id*
  The ID of the Zagreus Worker Controller which control the Zagreus Worker where the job was executed on (see also → Zagreus Worker-Controller)

- *Priority* (not shown by default)
  The priority number of the job. The smaller the number, the higher the priority is. The default priority number is 10, if is not set differently in the *priority* execution option, see → List of execution options

Specific tools for the Finished jobs window:

- *Refresh View* 
  Clicking on this icon tool refreshes the content of the Finished jobs window according to its report parameters. The content of the window can also be refreshed by clicking on the window header if the *Auto refresh window by clicking on its header* setting is checked in the *Options* dialog box, see → Options dialog

- *Setting report parameters…* 
  It opens the *Finished job report parameters* dialog box, see → Finished job report parameters dialog . This tool is also accessible from the tool menu opened by clicking the down-arrow icon right to the icon of this tool.

- *Setting log columns…* (accessible from the tool menu opened by clicking the down-arrow icon right to the *Setting report parameters…* icon)
  This menu item opens the same *Finished log report parameters* dialog that the *Setting log columns...* tool opens in the *Finished logs window,* See → Finished logs window

A context menu appears when the user is right-clicking on a job in the result list, see *Figure 73.*

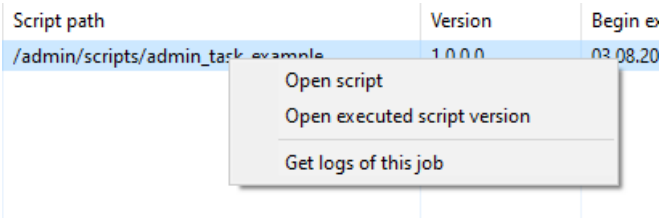


Figure 73 – The context menu of a job in the *Finished job* window

The following menu items are accessible:

- *Open script*
  It opens the actual script content from the local database that is associated with the job in the Script Editor.

- *Open executed script version*
  It opens the script content that was actually executed by the associated job (so it is a cached script version for the job) in the Script Editor.

- *Get logs of this job*
  Opens a new *Finished logs* window with the log messages of the selected job.

> ***Info:*** *It is also possible to open the associated log messages for a job by double-clicking on a selected row. It either opens the Finished logs window, or opens a Simple text editor with the log messages. The behaviour of this can be controlled with the appropriate setting in the Options dialog.*

10.4.4.1 Finished job report parameters dialog

This dialog allows the user to set the job report parameters for the Finished jobs window, see *Figure 74*.

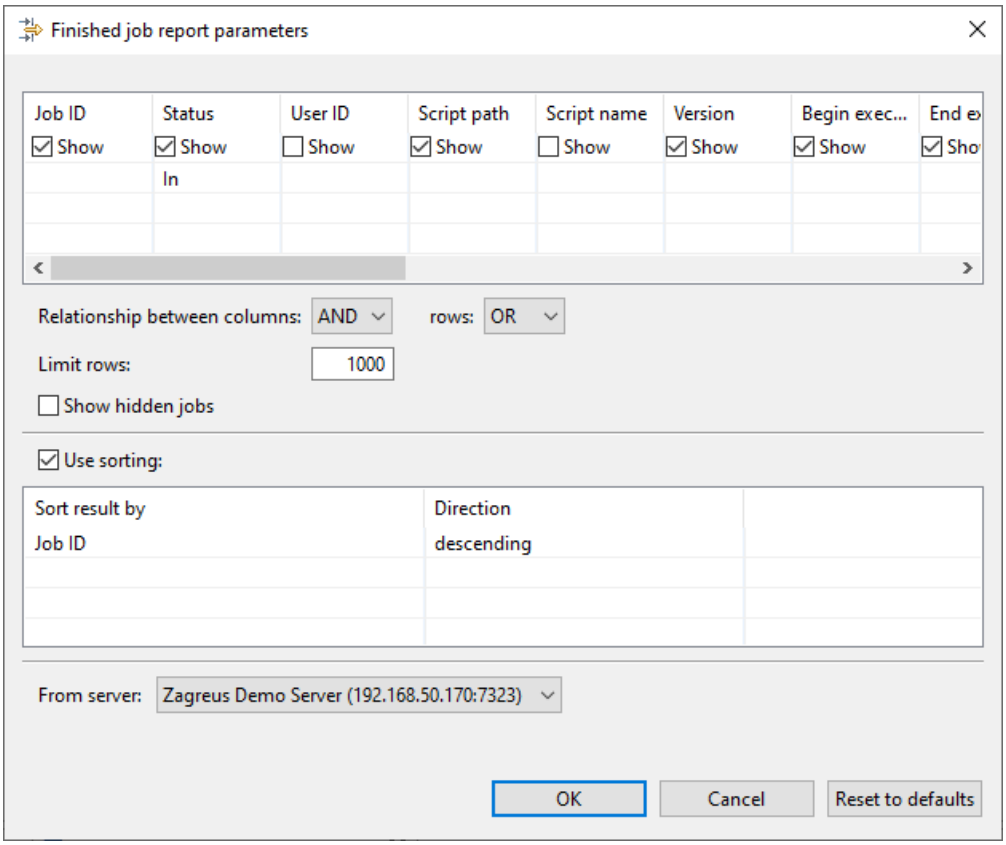


Figure 74 – The *Finished job report parameters* dialog box

In the table located in the upper-half of the dialog, the user can select the columns which will be displayed in the Finished jobs window. This can be done by checking the corresponding checkboxes (with the label *Show*) under the column names.

In this table the user can also specify conditions on the selected columns. The user has to click on one of the blank fields under the *Show* checkbox to specify a condition for a parameter. The *Condition parameter* window will appear after clicking on the *Add/modify condition* option. Based on the column where the user has clicked, a condition for the particular property can be set; for an example, see *Figure 75*. By clicking one of the conditions, the condition can be deleted by choosing the *Delete condition* option.

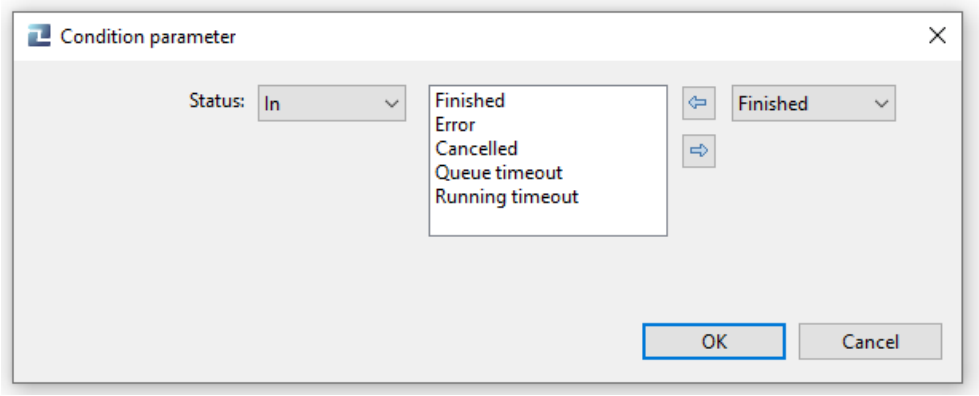


Figure 75 – The *Condition parameter* dialog box for the Status column

Based on the column type, the following operators are available:

- *For numeric types* (columns *User ID, Number of lines, Priority*):
  `=, <, >, <=, >=, <>, In, Between`

- *For date types* (columns *Begin exec. time, End exec. time, Begin queue time, End queue time*):
  `=, <, >, <=, >=, <>, Between`

- *For string types* (columns *Job ID, Script path, Script name, Version, Result message*):
  `=, <>, In, Between, Contains`

- *For pre-defined sets* (columns *Status, Caller type, Exec. mode*):
  `=, <>, In`

When the chosen operator requires two operands (like the Between operator), the dialog box changes its layout correspondingly.

Multiple conditions can be specified even for the same column. In this case the conditions have to be defined in separate rows under each other in the table.

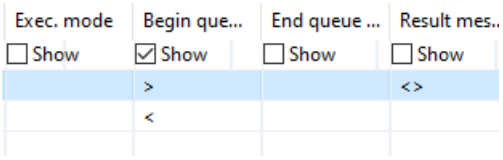


Figure 76 – Multiple conditions specified for *Begin queue time*

Under the table, the logical relationship between columns and rows can be specified, see *Figure 77*. The selectable values are *AND* and *OR*. These settings are evaluated in the following way: first the conditions of each separate row are grouped

(according to the *columns* setting), then the different such row-level conditions are grouped (according to the *rows* setting).

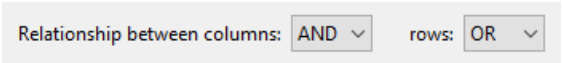


Figure 77 – The *Relationship between columns* and *rows* setting

The user can also set the maximum number of listed jobs by using the *Limit rows* setting, see *Figure 78*.

Figure 78 – The *Limit rows* setting

By checking the *Show hidden jobs* checkbox, the user can include the hidden jobs in the result list, see --> .

When the *Use sorting* option is checked, the listed finished job results can be sorted by the specified column names. To specify such a sorting column, the user needs to select the column name from the dropdown box which appears by clicking on the first empty cell of the first column (*Sort result by*). The direction of the selected column (*ascending* or *descending*) can be set in the same manner in the second column (*Direction*), see *Figure 79*.

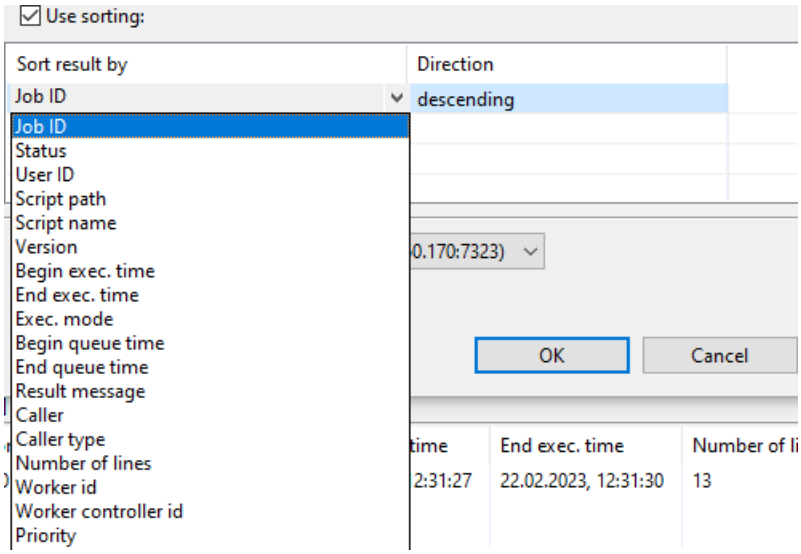


Figure 79 – Defining a sorting column name in the *Finished job report parameters* dialog box

If there are multiple open server connections, the target server can be selected from the *From server* dropdown list, see *Figure 80*. After changing the server, by clicking on the OK button, the header of the Finished jobs window changes correspondingly.

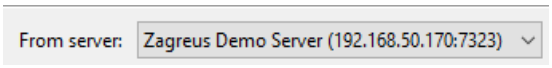


Figure 80 – The *From server* dropdown list

The changes are applied after clicking on the *OK* button. In order to reset all the settings, the user has to click on the *Reset to defaults* button.

10.4.5 Finished logs window

The *Finished logs window* displays the logging output for a certain completed job. This output can be accessed by using the *Get logs of this job* menu item in the Finished jobs window, see *Figure 81*. Also, it can be done by double-clicking on the selected job in the *Finished jobs* window, if the appropriate setting in the *Options* dialog allows this behaviour, see also → General behaviour tab.

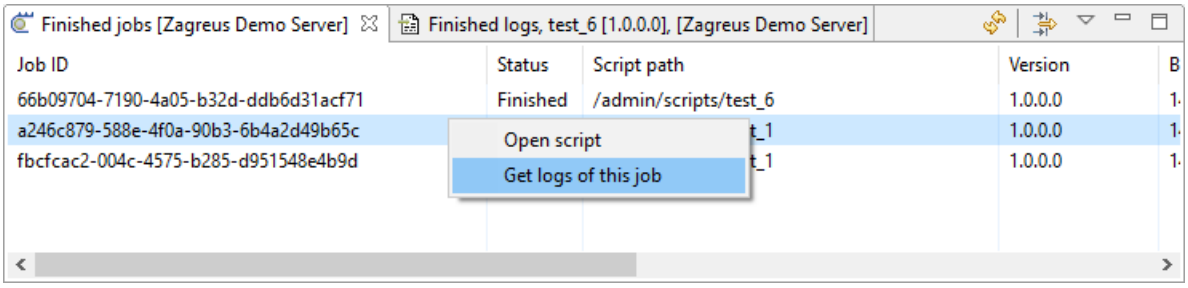


Figure 81 – Opening the log of a finished job in the *Finished jobs* window

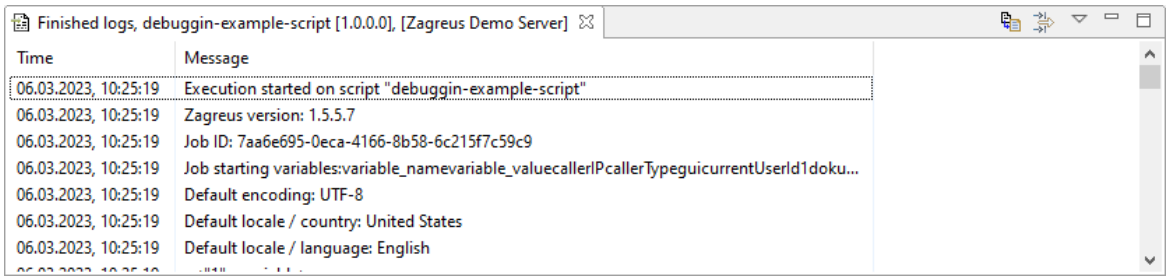


Figure 82 – A *Finished logs* window

The columns that appear in the Finished logs window:

- *Time*
  The timestamp of the log message.

- *Action*
  The fully qualified action name which created the log message, see → Action groups and action name

- *Action number* (not shown by default)

The action ordering number which created the log message, see → Ordering numbers

- *Thread id* (not shown by default)
  The ID of the thread in which the log message was created, see → Parallel threads in the z:foreach action

- *Message*
  The log message.

- *Level*
  The logging level of the message, see → Logging levels and loglevel

Specific tools for the Finished logs window:

- *Show log result in text editor*
  Displays the logging results in a Simple text editor in the Editor area.

- *Finished log report parameters*
  Opens the *Finished log report parameters* dialog, see *Figure 83.* The user can select the visible columns with the checkboxes in the dialog.

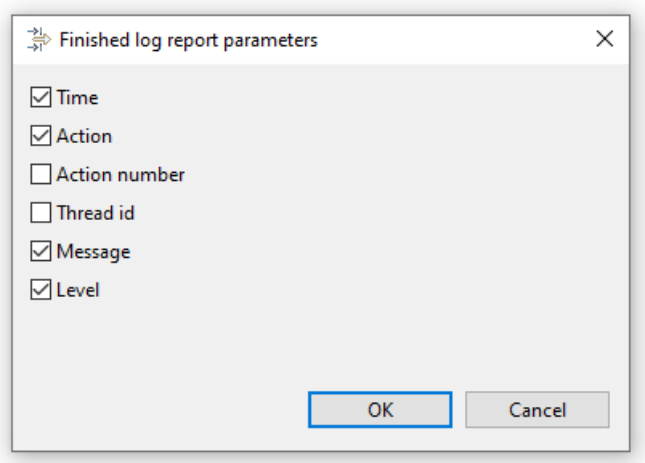


Figure 83 – The *Finished logs* report parameters dialog

10.4.6 Skipped jobs window

The *Skipped jobs window* displays all scripts with skipped execution in a specific time period. For more information about skipped jobs, see → Skipped jobs.

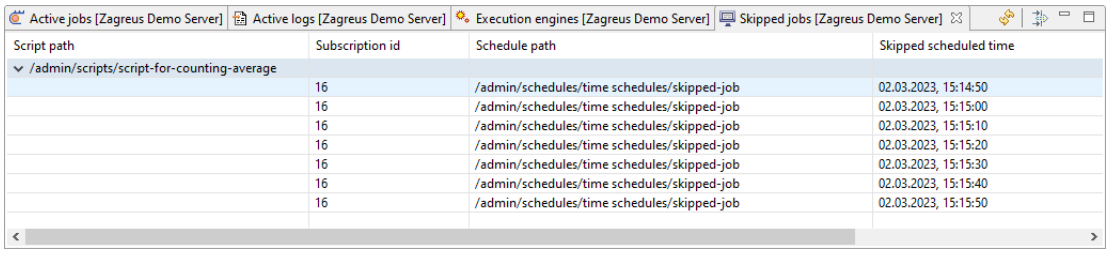


Figure 84 – The *Skipped jobs* window

This window is not open by default, it can be opened either by clicking on the *Open/close Skipped jobs window* tool in the main toolbar (see also → Open/close the Skipped jobs window), or accessed by selecting the *Skipped jobs window* option from the *Windows* main menu bar.

The results of the skipped job report appears as grouped elements by a certain criterion, see → Skipped jobs in the Zagreus Client. The parent nodes are expandable to see the individual skipped jobs.

The columns that are shown in the Skipped jobs window are:

- *Script path*
  The full path of the script associated with the skipped job

- *Subscription id*
  The id of the subscription associated with the skipped job, see → Subscriptions

- *Schedule path*
  The full path of the time schedule associated with the subscription above

- *Skipped scheduled time*
  The timestamp when the script should have been executed.

Specific tools for the Skipped jobs window:

- *Refresh view*
  It refreshes the content of the window according to the Skipped jobs window preferences dialog settings.

- *Skipped jobs window preferences tool*
  It opens the Skipped jobs window preferences dialog, see → Skipped jobs window preferences dialog

10.4.6.1 Skipped jobs window preferences dialog

This dialog allows the user to set the job report parameters for the Skipped jobs window, see *Figure 85*.

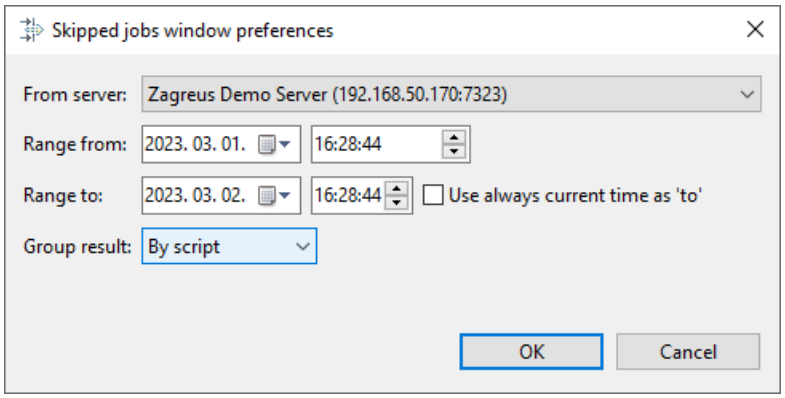


Figure 85 – The *Skipped jobs window preferences* dialog

The sections of the dialog are the following:

- *From server*
  If there are multiple open server connections, the target server can be selected from this dropdown list. After changing the server, by clicking on the OK button, the header of the Skipped jobs window changes correspondingly

- *Range from*
  It contains two widgets: a calendar for the date and a widget for the time setting. Both settings together specify the starting time for the skipped job report.

- *Range to*
  It contains two widgets: a calendar for the date and a widget for the time setting. Both settings together specify the end time for the skipped job report.
  Also, there is a checkbox *Use always current time as 'to'*. It automatically sets the current time as the end time for the skipped job report.

- *Group result*
  The skipped jobs can be grouped by different columns: *script*, *subscription* and *schedule*. For an example, see *Figure 86*.

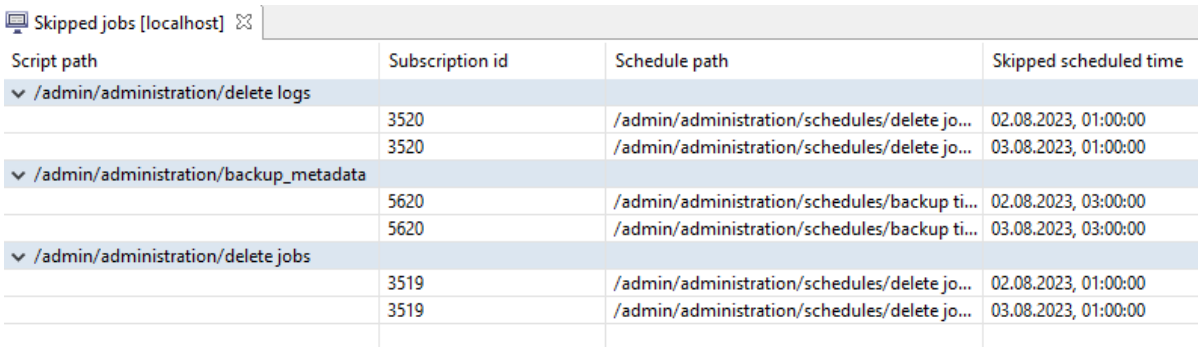


Figure 86 – The *Skipped jobs* window grouped by script path

10.5 Main menu bar

The main menu bar can be found at the top left corner of the Zagreus Client.

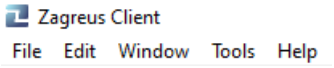


Figure 87 – The *Main menu bar*

The menu bar contains several menus with logically grouped menu items. Some of them can be accessed in another way as well but a few of them can only be accessed from the menu bar.

10.5.1 *File* menu

- *New resource*
  It creates a new resource. It is only active if there is an open Zagreus Server connection.

- *Exit*
  Terminates the Zagreus Client application. When there are open resources in the Editor Area, first, the Zagreus Client offers saving them before closing the main window.

10.5.2 *Edit* menu

- *Undo*
  Performs an undo operation for the script being edited in the currently opened Script Editor, see → Script Editor.

- *Redo*
  Performs a redo operation for the script being edited in the currently opened Script Editor, see → Script Editor.

- *Delete*
  This operation depends on the active selection in the main application window. It deletes the selected resource(s) if the *Zagreus browser* window is active, or the selected action(s) in the *Script Editor* if the focus is on the editor.

More than one resource can be selected by holding down the *Ctrl* or *Shift* buttons while selecting resources in the Zagreus browser window (see → Zagreus browser window) or actions in the Script Editor (see → Selecting multiple actions).

- *Rename resource*
  Performs the same operation as the *Rename resource...* menu item in the Zagreus Browser window, see → Renaming resources.

10.5.3 *Window* menu

- *Zagreus browser*
  To open or close the *Zagreus Browser* window (see → Zagreus browser window), the user needs to use this menu item or use the *Ctrl+Alt+Z* key combination. This option is also accessible by selecting the *Zagreus browser* icon tool from the main toolbar.

- *Active jobs window*
  To open or close the *Active jobs window* (see → Active jobs window), the user needs to use this menu item or the *Ctrl+Alt+J* key combination. This option is also accessible by selecting the *Active jobs window* icon tool from the main toolbar.

- *Active logs window*
  To open or close the *Active logs window* (see → Active logs window), the user needs to use this menu item or the *Ctrl+Alt+L* key combination. This option is also by selecting the *Active logs window* icon tool from the main toolbar.

- *Engine status window*
  To open or close the *Execution engines* window (see → Execution engines window), the user needs to use this menu item tool or the *Ctrl+Alt+E* key combination. This option is also accessible by selecting the *Execution engines* window icon tool from the main toolbar.

- *Finished jobs window*
  To open a new instance of the *Finished jobs window* (see → Finished jobs window), the user needs to use this menu item or the *Ctrl+J* key combination. Multiple *Finished jobs* windows can be opened simultaneously. This option is also accessible by selecting the *Finished jobs window* icon tool from the main toolbar.

- *Finished logs window*
  To open a new instance of the *Finished logs window* (see → Finished logs window), the user needs to use this menu item or the *Ctrl+L* key combination. Multiple *Finished logs* windows can be opened simultaneously. This option is also accessible by selecting the *Finished logs window* icon tool from the main toolbar.

- *Skipped jobs window*
  To open or close the *Skipped jobs window* (see → Skipped jobs window), the user needs to use this menu item. This option is also accessible by selecting the *Skipped jobs window* icon tool from the main toolbar.

- *Outline window*
  To open or close the *Outline window* (see → Outline window), the user needs to use this menu item or the *Ctrl+Alt+O* key combination.

- *Attributes window*
  To open or close the *Attributes window* (see → Attributes window), the user needs to use this menu item or the *Ctrl+Alt+A* key combination.

- *Variables window*
  To open or close the *Variables window* (see → Variables / Functions window), the user needs to use this menu item or the *Ctrl+Alt+V* key combination.

- *Breakpoints window*
  To open or close the *Breakpoints window* (see → Breakpoints window), the user needs to use this menu item or the *Ctrl+Alt+B* key combination.

- *Watch variables window*
  To open or close the *Watch variables window* (see → Watch window), the user needs to use this menu item or the *Ctrl+Alt+W* key combination.

10.5.4 *Tools* menu

- *Password converter*
  The user can convert passwords from plain text to a Zagreus-specific encrypted format. Encrypted passwords can be inserted in the *cpassword* attribute in several actions (see → username, password and cpassword attributes), mostly in

connection actions. In the Script Editor, the value of the *password* attribute can also be encrypted into a *cpassword* attribute by the *Encrypt password* menu item in the attribute context menu, see → Encrypt password.

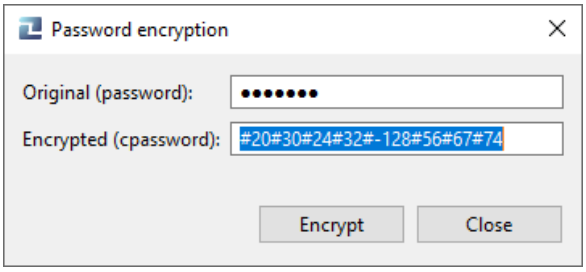


Figure 88 – The *Password encryption* dialog box

- *Options…*
  It opens the *Options* dialog box. There are a lot of settings for the Zagreus Client which are described in a separated chapter, see → Options dialog.

10.5.5 *Help* menu

- *About*
  It opens the *About Zagreus Client* dialog box. The version number can be seen at the upper part of the dialog.

Note: the *Installation Details* button opens another *Zagreus Client Installation Details* dialog which shows information on the Eclipse RCP (Rich Client Platform) application level. This is the underlying window manager framework for the Zagreus Client application, the details here are very technical and not directly related to the Zagreus System installation structure.

10.6 Main toolbar

The main toolbar provides quick access to tools and dialogs for some of the most common tasks in the Zagreus Client. It is located directly under the main menu bar, see *Figure 89*.

Figure 89 – The main toolbar

10.6.1 Views

Views are specific layouts for particular tasks. A view defines a structure of graphical placeholders for windows, areas, groups of other elements. A view can be treated as a comfortable arrangement of the different parts in the Zagreus Client.

10.6.1.1 *Edit* view

The default view in the Zagreus Client is the *Edit view*, which contains the following windows and areas:

Default:

- *Zagreus Browser* window, see → Zagreus browser window
- *Editor area*, see → Editor area
- *Active jobs* window, see → Active jobs window
- *Active logs* window, see → Active logs window
- *Execution engines* window, see → Execution engines window
- *Outline* window, see → Outline window
- *Attributes* window, see → Attributes window
- *Variables* window, see → Variables / Functions window
- *Breakpoints* window, see → Breakpoints window
- *Watch* window, see → Watch window

Optional:

- *Finished jobs* window(s), see → Finished jobs window
- *Finished logs* window(s), see → Finished logs window
- *Skipped jobs* window, see → Skipped jobs window

These are all the possibly displayable window types and the main editor area. This view is intented to use for the common browsing and editing operations.

The remaining space is used for the editors. The user can switch between the views by selecting the proper menu item from the main Window menu, or by clicking on the proper toggle-button on the main toolbar.

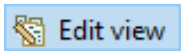


Figure 90 – The *Edit View* button in the toolbar

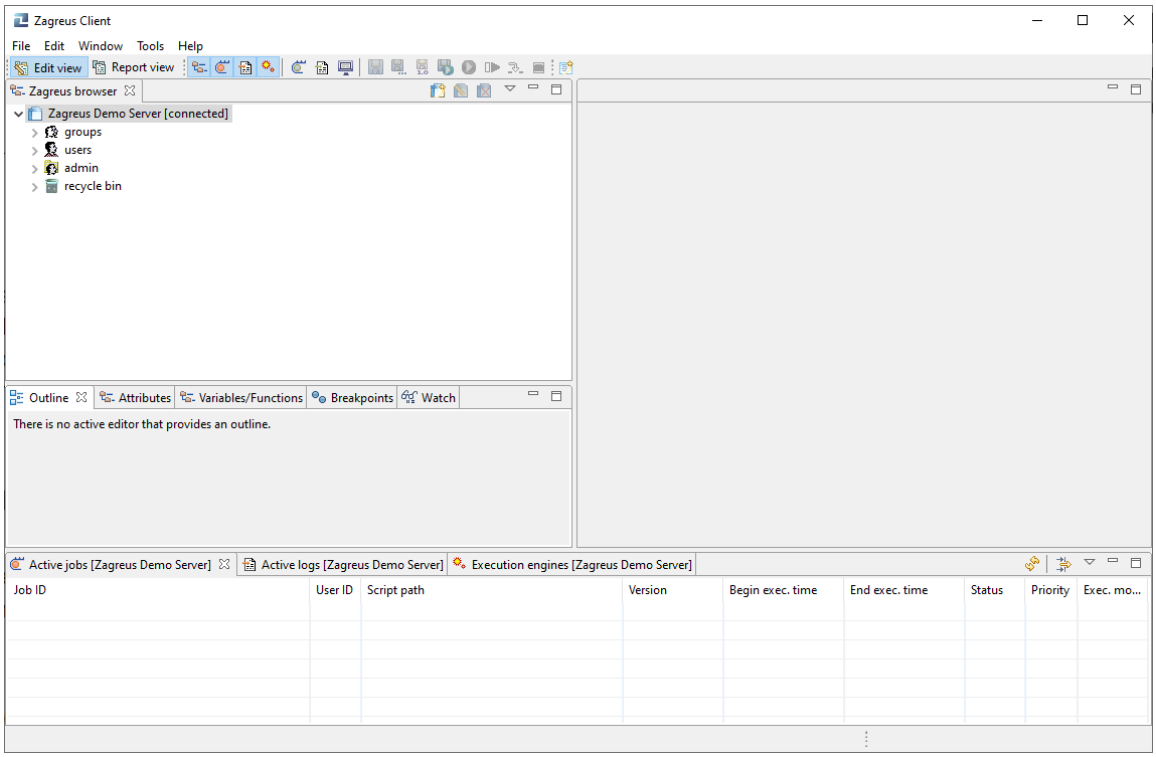


Figure 91 – The *Edit View* in the Zagreus Client

10.6.1.2 Report view

The *Report view* is a more comfortable arrangement of windows for monitoring job execution. The editor area is not present in this view, leaving more room for all the other windows.

The Report view in the Zagreus Client contains the following windows and areas:

Default:

- *Zagreus Browser* window, see → Zagreus browser window
- *Active jobs* window, see → Active jobs window
- *Active logs* window, see → Active logs window

- *Skipped jobs* window, see → Skipped jobs window
- *Execution engines* window, see → Execution engines window

Optional:

- *Finished jobs* window(s), see → Finished jobs window
- *Finished logs* window(s), see → Finished logs window

The view can be switched by clicking on the proper toggle-button on the main toolbar, or selecting the proper menu item from the `Window` main menu.

Figure 92 – The *Report View* button in the toolbar

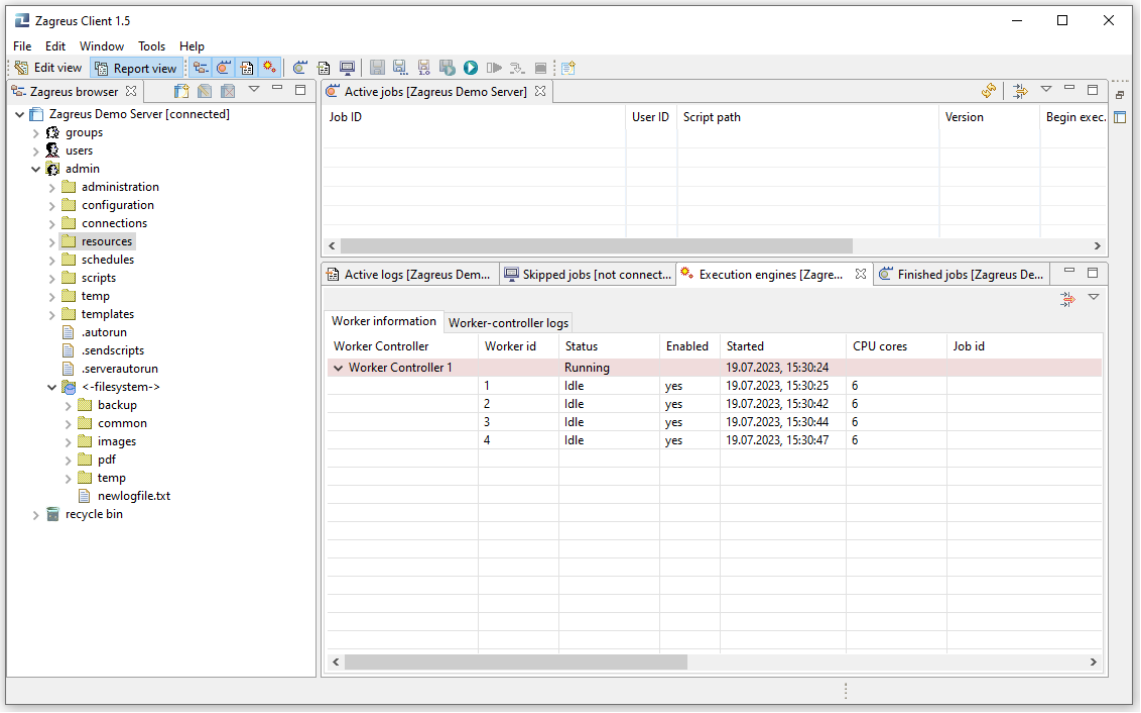


Figure 93 – The *Report view* in the Zagreus Client

10.6.2 Open/close the Zagreus browser window

To open or close the *Zagreus Browser window* (see → Zagreus browser window), the user needs to use this toggle button or the *Ctrl+Alt+Z* key combination. This option is also accessible by selecting the *Zagreus browser* menu item from the *Window* main menu.

Figure 94 – The *Open/close Zagreus browser window* icon tool

10.6.3 Open/close the Active jobs window

To open or close the *Active jobs window* (see → Active jobs window), the user needs to use this toggle button or the *Ctrl+Alt+J* key combination. This option is also accessible by selecting the *Active jobs window* menu item from the *Window* main menu.

Figure 95 – The *Open/close Active jobs window* icon tool

10.6.4 Open/close the Active logs window

To open or close the *Active logs window* (see → Active logs window), the user needs to use this toggle button or the *Ctrl+Alt+L* key combination. This option is also by selecting the *Active logs window* menu item accessible from the *Window* main menu.

Figure 96 – The *Open/close Active logs window* icon tool

10.6.5 Open/close the Engine status window

To open or close the *Engine status window* (see → Execution engines window), the user needs to use this toggle button tool or the *Ctrl+Alt+E* key combination. This option is also accessible by selecting the *Engine status window* menu item from the *Window* main menu.

Figure 97 – The *Open/close Engine status window* icon tool

10.6.6 Open a new Finished jobs window

To open a new instance of the *Finished jobs window* (see → Finished jobs window), the user needs to use this tool or the *Ctrl+J* key combination. Multiple *Finished jobs* windows can be opened simultaneously. This option is also accessible by selecting the *Finished jobs window* menu item from the *Window* main menu.

Figure 98 – The *Finished jobs window* icon tool

10.6.7 Open a new Finished logs window

To open a new instance of the *Finished logs window* (see → Finished logs window), the user needs to use this tool or the *Ctrl+L* key combination. Multiple *Finished logs* windows can be opened simultaneously. This option is also accessible by selecting the *Finished logs window* menu item from the *Window* main menu.

Figure 99 – The *Finished logs* window icon tool

10.6.8 Open/close the Skipped jobs window

To open or close the *Skipped jobs window* (see → Skipped jobs window), the user needs to use this toggle button. This option is also accessible by selecting the *Skipped jobs window* menu item from the *Window* main menu.

Figure 100 – The *Open/close Skipped jobs window* icon tool

10.6.9 Save resource to the server

When the developing of the resource is finished the resource can be saved by clicking on the *Save resource* button or by pressing the *Ctrl+S* hotkey.

Figure 101 – The *Save resource onto the server* icon tool

If the resource has not been saved yet, Zagreus will offer to save the resource by displaying the *Save resource as* dialog box (see below); otherwise, the resource will be overwritten without any warning.

10.6.10 Save as… resource to the server

This tool allows the user to perform one of the following operations:

- Save the resource for the first time if it is not saved yet.
- Save a copy of the currently edited resource to a different location / with different name.

Figure 102 – The *Save as…* resource icon tool

The tool is also accessible by pressing the *Ctrl+Alt+S* hotkey.

The tool opens the *Save resource as* dialog box, allowing the user to specify the location and name of the resource.

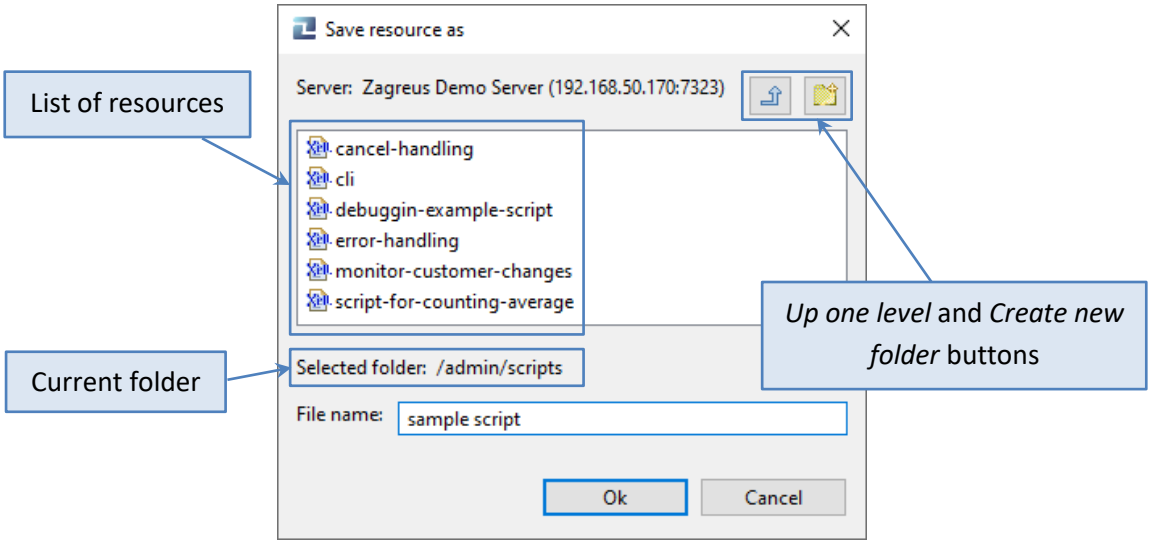


Figure 103 – The *Save resource as* dialog box

Information related to the location is also shown in the dialog box, see *Figure 103.*:

- *Server:* the currently selected server (if there are multiple connected servers, the user can select a different server by going upward to the server level in the dialog and choose another one)
- *List of resources:* contains all of the resources in the selected folder
- *Selected folder:* the path of the currently selected folder

The selected folder is set in the dialog box depending on various conditions:

- If the resource is not saved yet, and there is a folder or resource selected in the Zagreus browser window (in the embedded database, not in the server filesystem), then the folder itself or the parent folder of the selected resource is set by default.

- If the resource is not saved yet, and there is no selection in the Zagreus browser window (or a filesystem resource is selected), then the server root node is selected by default in the dialog box.

- If the resource is already saved, the original parent folder is selected here.

The user can use the *Up one level* button to navigate upwards in the folder structure, and the *Create new folder* button to create a new folder, see *Figure 103.*

The user has to specify the resource name in the *File name* field and then click on the *OK* button to save it.

10.6.11 Save a new version of the resource

The user can create a new version of the currently opened resource by using this tool or pressing the *Ctrl+Shift+V* key combination. Saving a new version creates a copy of the original resource with the same id but with a different version number, see → Resource versioning. The tool opens the *Set resource version* dialog box to set the parameters of this operation, see *Figure 104*.

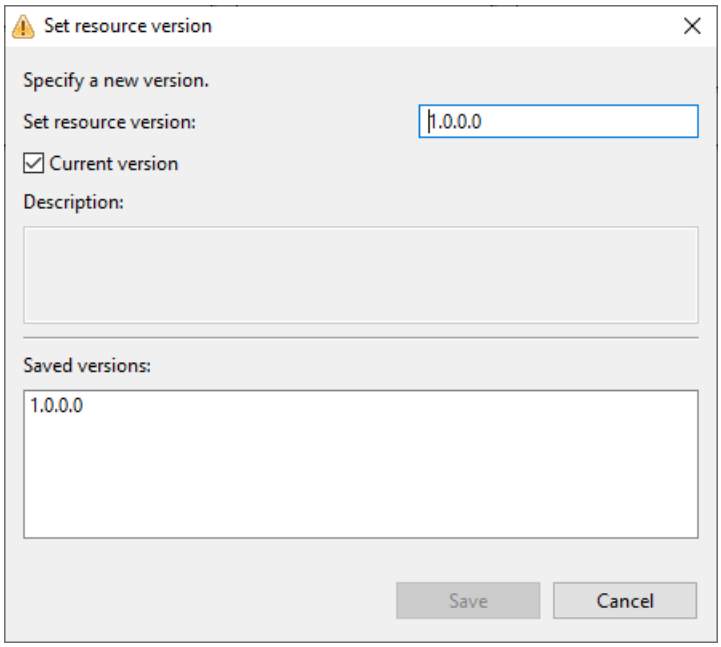


Figure 104 – The *Set resource version* dialog box

- *Set resource version*
  The user needs to enter the new version number for the resource. The text box is filled with the version of the edited resource, this must be changed in order to save a new one.

- *Current version*
  The newly created resource will be set as the current version. For more information about current version, see → Current version.
  Setting the current version can also be done by selecting the *Set to current version* option from the context menu in the Zagreus browser window. The user has to right-click on one of the versions grouped under a resource version parent node.

- *Description*
  Allows the user to specify a description for the newly created resource version. A resource description can also be specified by right-clicking on a resource, and select the *Show resource information* menu item from the context menu in the Zagreus browser window. Then, in the *Resource info* dialog box, the description can be added or changed.

- *Saved versions*
  The list of the available versions of the resource

It is recommended to use the version format specified → Version format. After saving the new resource, the new version is listed In the Zagreus browser, see *Figure 105.*

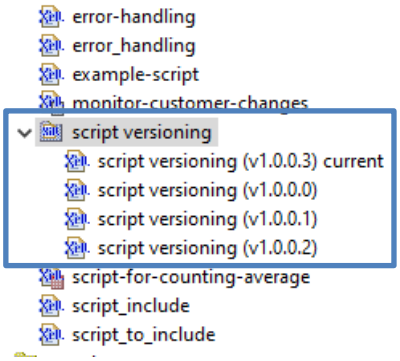


Figure 105 – Versions of a script

10.6.12 Save and run resource

It runs the resource after saving. The *Save resource as* dialog box will appear when the resource that is newly created has not been saved yet. After it has been saved, Zagreus initiates the execution of the resource, see → Initiating script execution.

Figure 106 – The *Save and run* resource icon tool

10.6.13 Run script

It runs the script that is currently opened and active in the Editor Area. If the script has not been saved yet, Zagreus will show a warning message that the script must be saved before running it.

Figure 107 – The *Run script* icon tool

10.6.14 Resume

This icon tool is only enabled for the debug mode of the Script Editor, see → Starting a debug session.

Figure 108 – The *Resume* icon tool

10.6.15 Step to the next action

This icon tool is only enabled for the debug mode of the Script Editor, see → Starting a debug session.

Figure 109 – The *Step to the next action* icon tool

10.6.16 Stop debugging

This icon tool is only enabled for the debug mode of the Script Editor, see → Starting a debug session.

Figure 110 – The *Stop debugging* icon tool

10.6.17 Create new resource

It creates a new resource. By clicking on this icon (see *Figure 111.*), the dialog box *Creating new resource* appears, see *Figure 112*. The resource name can be set (the default is *Untitled*) and the resource type must be selected. After clicking on the *OK* button, the proper editor based on the resource type will appear in the Editor Area.

Figure 111 – The *Create new resource…* icon tool

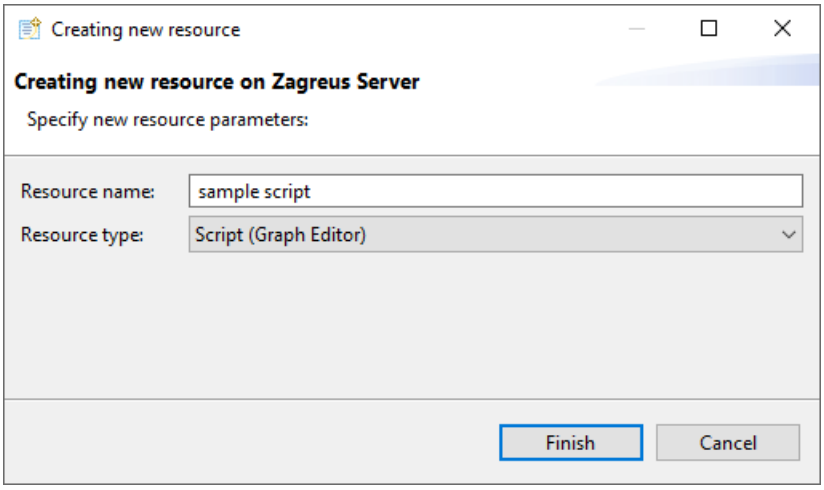


Figure 112 – The *Creating new resource* dialog box

The following resource types are listed:

- *Script (Graph Editor)*
  Creates a script resource that opens in a Script Editor, see also → Script Editor.

- *Script (XML Editor)*
  Creates a script resource that opens in an XML Editor, see also → Opening resources.

- *Template*
  Creates a template resource that opens in a Script Editor, see also → Script Editor.

- *Connection*
  Creates a connection resource that opens in a Script Editor, see also → Script Editor.

- *Event schedule*
  Creates an event schedule resource that opens in an Event editor, see also → Event schedule.

- *Time schedule*
  Creates a time schedule resource that opens in a Cron Time editor, see also → Time schedule.

- *File trigger*
  Creates a file trigger resource that opens in a File trigger editor, see also → File trigger.

- *DB watcher*
  Creates a database watcher resource that opens in a Database watcher editor, see also → Database watcher.

- *Mail watcher*
  Creates a mail watcher resource that opens in a Mail watcher editor, see also → Mail watcher.

- *Simple text file*
  Creates a simple text resource that opens in a Simple text editor, see → Simple text editor.

The created resource is stored only in the memory until it is saved properly to a Zagreus Server.

10.6.18 Zoom display

Zooming in and out of the Script Editor area is possible by selecting a percentage value from a predefined list or by typing in a unique value.

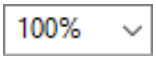


Figure 113 – The *Zooming* predefined list

The default setting is 100%. To select from one of the predefined sizes, the user needs to use the down arrow next to the number which is currently set. Apart from the pre-defined values, the option *Page*, *Width* and *Height* are also available in the list.

- *Page*: It resizes the content to fit the currently visible Script Editor area.
- *Width*: It resizes the content to fit the currently visible Script Editor area based on the width of the canvas.
- *Height*: It resizes the content to fit the currently visible Script Editor area based on the height of the canvas.

An accurate size can also be set manually by clicking on the displayed number and overwriting it.

10.6.19 Zoom in and Zoom out

These tools allow resizing the Script Editor area. The zoom steps come from the predefined list available in the zoom display. A more accurate size can be specified in the Zoom display tool, see above.

Figure 114 – The *Zoom in* and *Zoom out* icon tools

10.7 Options dialog

The *Options* dialog box can be opened by the *Options...* menu item from the *Tools* menu in the main menu bar. It provides a lot of settings that are grouped in different tabs. The settings are saved by clicking the *Ok* button at the bottom of the dialog and they are persisted after the Zagreus Client is closed and reopened.

10.7.1 Graph Editor tab

- *Element mode after double-clicking on an iconized state*
  This option sets the default element view mode after double-clicking on the iconized mode in the Script Editor, see → Iconized view mode.

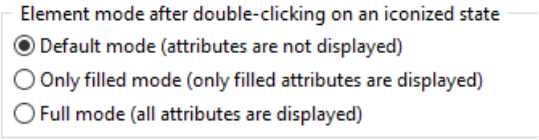


Figure 115 – The *Element mode after double-clicking...* setting

  - *Default mode*
    This sets the view mode when the attributes are not displayed for the action in the Script Editor. This is the default setting. This option can also be accessed in the Script Editor by right-clicking on one of the actions and selecting the *Default* menu item.

Figure 116 – Example for the *default* view mode of an action

  - *Only filled mode*
    This sets the view mode when only the attributes with filled values are displayed for the action in the Script Editor. This option can also be accessed in the Script Editor by right-clicking on one of the actions and selecting the *Only filled* menu item.

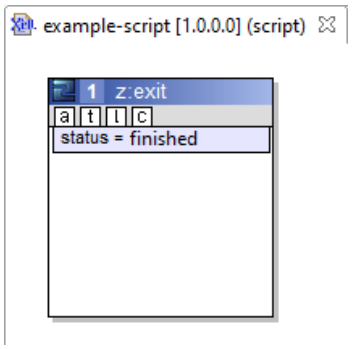


Figure 117 – Example for the *only filled* view mode of an action

- *Full mode*
  This sets the view mode when all the attributes are displayed for the action in the Script Editor. This option can also be accessed in the Script Editor by right-clicking on one of the actions and selecting the *Full* menu item..

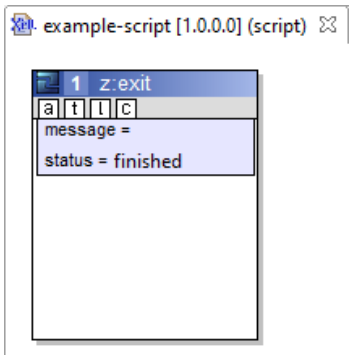


Figure 118 – Example for the *full* view mode of an action

- *Icons*
  The user can select if the new icons, old icons or a combination of the two should be used in the palette in the Script Editor. The old icons were designed for each individual actions, the new ones for only each action groups.

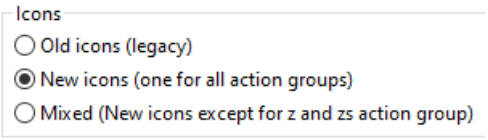


Figure 119 – The *Icons* settings

  - *Old icons*
    The legacy icons will be shown in the palette

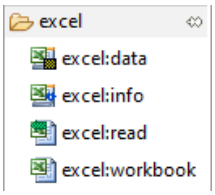


Figure 120 – Old icons in the palette

- *New icons*
  The new icons will be shown in the palette, one different icon for each action groups.

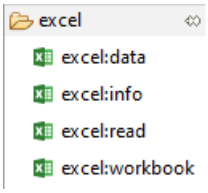


Figure 121 – New icons in the palette

- *Mixed*
  The new icons except for `z` and `zs` action groups will be displayed in the palette in the Script Editor

- *Siblings*
  The user can select the behavior of sibling arrows in case of reordering the affected actions.

  - *Keep sibling arrows after reordering actions*
    If the sibling link arrows are displayed among several sibling actions (see → Showing sibling links), reordering the actions (see → Changing the order of execution) can keep the arrows displayed or make them disappear according to this setting.

- *Resizing*
  The user can select the behavior of the automatic parent action resizing.

  - *Auto-resize the parents (if needed) when a new element inserted*
    when an element is inserted to the content of an action, the size of the parent action is adjusted accordingly when this option is set.

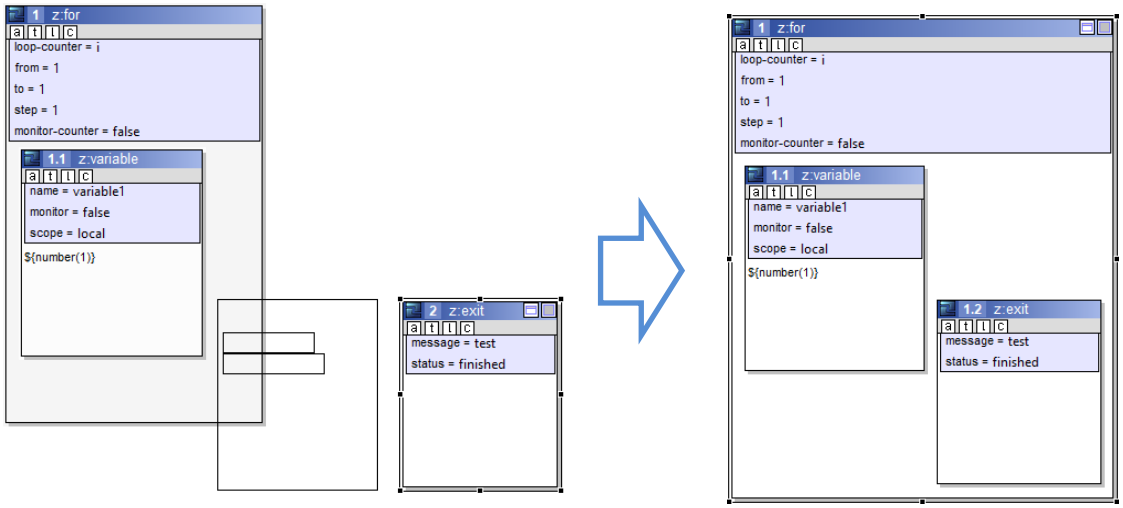


Figure 122 – Auto-resize the parent when a new child action is inserted

- *Auto-resize the parents (if needed) when a child element moved inside*
  when the position of a child element is changed inside the content of the parent action, the size of the parent action is adjusted accordingly if this option is set.

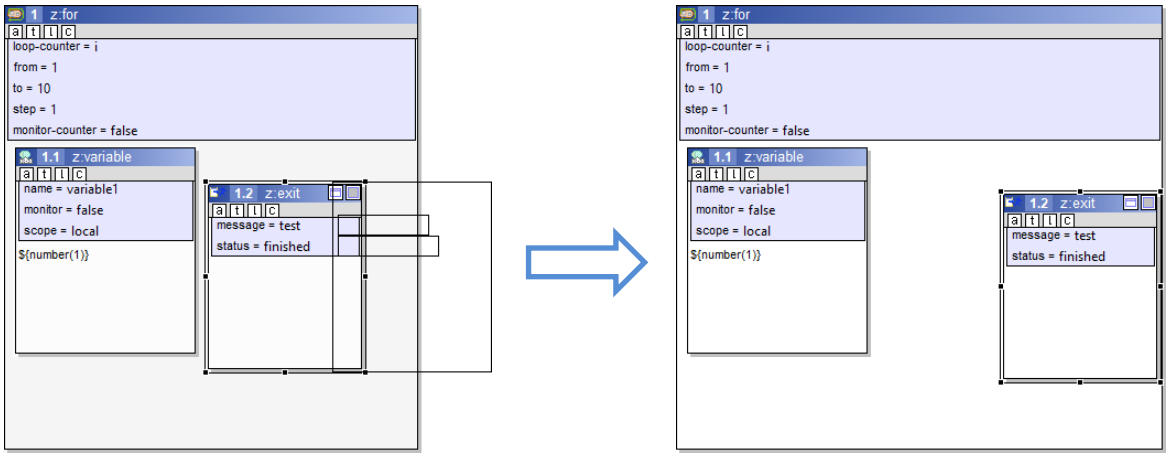


Figure 123 – Auto-resize the parent when a new child action is moved inside the content of the parent action

- *Prevent resizing parent smaller when its content would be covered*
  if this is set, the parent action cannot be resized to the point where any of its child actions are hidden or partially coverd

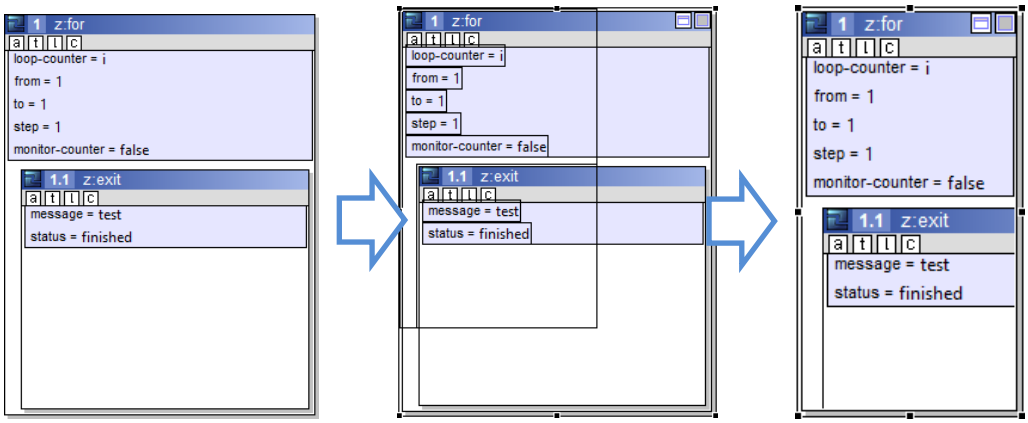


Figure 124 – When this option is not set, the parent can be resized smaller than its content

- *Transparency*
  The transparency of the elements that overlap can be specified here in percentages.

  - When the value is set to 100, there is no transparency, see *Figure 125.*

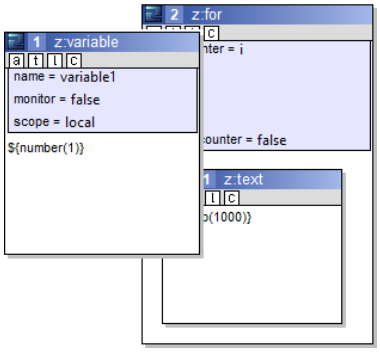


Figure 125 – No transparency for overlapping actions

  - When the value is set to 50, there is 50% transparency, see *Figure 126.*

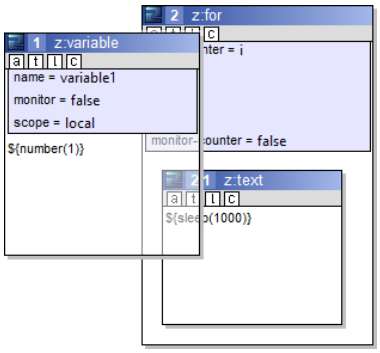


Figure 126 – 50% transparency for overlapping actions

- *Minimum sizes*
  Minimum height of the non-iconized element content area (resizing is not allowed for a smaller height value).

- When the value is set from 100 to 40, see *Figure 127.*

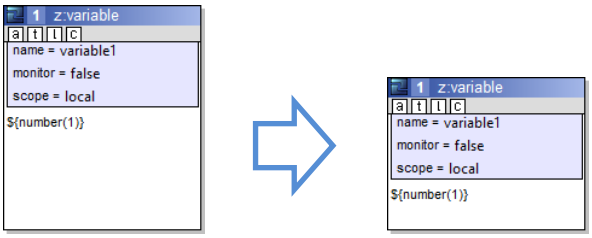


Figure 127 – The minimum height is changed from 100 to 40

- *Scroll*
  Reveals non-visible actions when clicked on its name in the *Outline window*

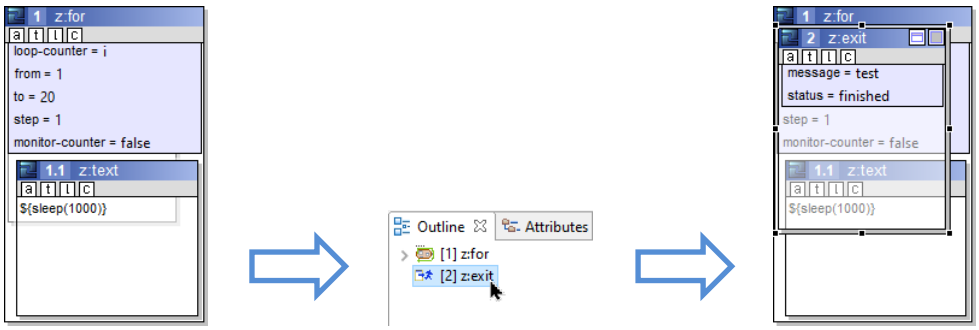


Figure 128 – Revealing a covered action when clicked on its name in the Outline window

10.7.2 Download / upload tab

- *Opening downloaded resources on client side*
  When a program path is set for a specific extension of a simple file, the downloaded resource will be opened after downloading, see → Uploading and downloading resources.

  - *Program path for opening .pdf files*
    The path to the program for opening `.pdf` extension files can be specified here.

  - *Program path for opening .xml files*
    The path to the program for opening `.xml` extension files can be specified here.

  - *Program path for opening .xsl files*
    The path to the program for opening `.xsl` extension files can be specified here.

- *Program path for opening .xls, .xlsx files*
  The path to the program for opening `.xls` and `.xlsx` extension files can be specified here.

- *Program path for opening .txt files*
  The path to the program for opening `.txt` extension files can be specified here.

- *Path where to save these files*
  The temporary saving folder for the Zagreus Client. The files are downloaded here in two cases:

  A. When the user clicks on the *Open resource on client side* menu item in the context menu in the Zagreus browser window (see → Opening resources). The file is first downloaded in the temporary saving folder, then opened by the external editor set in the previous settings.
  B. When the user clicks on the *Open results in text editor* button in the *Search for resources* dialog box, see → Searching for resources. The opened search result file is saved to this temporary saving folder.

- *Uploading:*

  - *Consider file extensions for uploading resources*
    Zagreus will save the uploaded resources as follows in its database if this setting is checked:

    A. A file with the *.scr* extension is being saved as a script resource after uploading.
    B. A file with the *.tmp* extension is being saved as a template resource after uploading.
    C. A file with the *.con* extension is being saved as a connection resource after uploading.
    D. Other files that have no extension listed above will be saved as simple text files.

10.7.3 Copy tab

- *Overwriting*
  The overwrite type for the copy operation can be set here.

  - *Automatically (overwrite with source)*
    The resource with the same name in the target folder will be overwritten with the copying resource. Zagreus will overwrite the target resource without warning.

  - *Use* `Copy of` *… prefix and numbering*
    If a resource with the same name already exists in the target folder, the copying resource will be prefixed with `Copy of`. If it is copied more than once, an ordering number will appear in addition to the prefix.

  - *Skipping (leave the target resource)*
    A resource with the same name is left unchanged in the target folder.

- *Versions*
  The version copy options can be set here.

  - *Copy only the current version*
    When a resource has multiple versions, only the current version is copied.

  - *Copy all versions together*
    All versions are copied together no matter which one was selected first.

- *To another server*
  - *Use the same id, name and version*
    this setting is for updating or merging different systems in different servers. Overwriting is done automatically, and all versions are copied together (the overwrite and version settings are ignored in this case). When it is checked the following options are available:

    A. *Preserve current target version*
       If the resource with the same id already exists in the target site, it will not be overwritten by the resource on the source side.

B. *Preserve current source version*
If the resource with the same id already exists in the target site, it will be overwritten by the resource on the source side.

10.7.4 *General behaviour* tab

- *Finished jobs window*
  These settings are related to the *Finished job window* (see also → Finished jobs window):

  - *Auto refresh window by clicking on its header*
    By checking this setting, the finished jobs will be refreshed in the *Finished jobs* window by clicking on its header; otherwise, the 'Refresh view' tool should be used for refreshing the content of the window.

  - *Double-click on job opens logs for jobs in text editor without log view window*
    This setting controls the behavior of the opened job-logs for a finished job. When the option is checked, the logs are opened in a simple text file in the Editor area (see → Simple text editor), otherwise they will be opened in a *Finished logs* window (see → Finished logs window).

  - *Automatic scroll-down to the end of the finished logs when opened by double-clicking (see above)*
    This setting also controls the behaviour of the opened job-logs for a finished job. If the job-log file is configured to be opened in a simple text file in the Editor area (see setting *Double-click on job opens logs for jobs in text editor without log view window* above) and this setting is also checked, the job-log will automatically scroll to the end of the job-log when opening.

- *Scaling*
  If there is font scaling in the operating system, this setting can help to properly set the dialog sizes accordingly to the OS scale setting.

10.7.5 Palette tab

- *Action groups visibility*
  Action groups that will be visible on the Palette in the Script Editor can be selected from this list. Scripts must be reopened for the changes to take effect.

Figure 129 – The *Action groups visibility* list on the *Palette* tab

10.8 Keybindings

By pressing the *Ctrl+Shift+L* key combination, the keybindings preferences page can be accessed, see *Figure 130.*

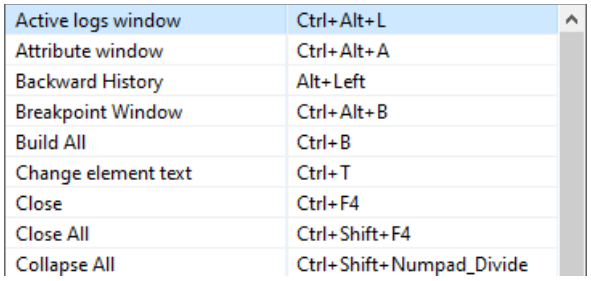


Figure 130 – The keybindings preferences page

11. Zagreus Monitor

The Zagreus Monitor is a standalone client application for monitoring script execution including finished and active jobs as well as scheduled estimations. The Zagreus Monitor can be started from the Zagreus root folder by the `$ZAGREUS_HOME/startmonitor.bat` file.

Like the Zagreus Client, see→ Zagreus Client, the Zagreus Monitor user interface is also split into sections as shown in *Figure 1.*

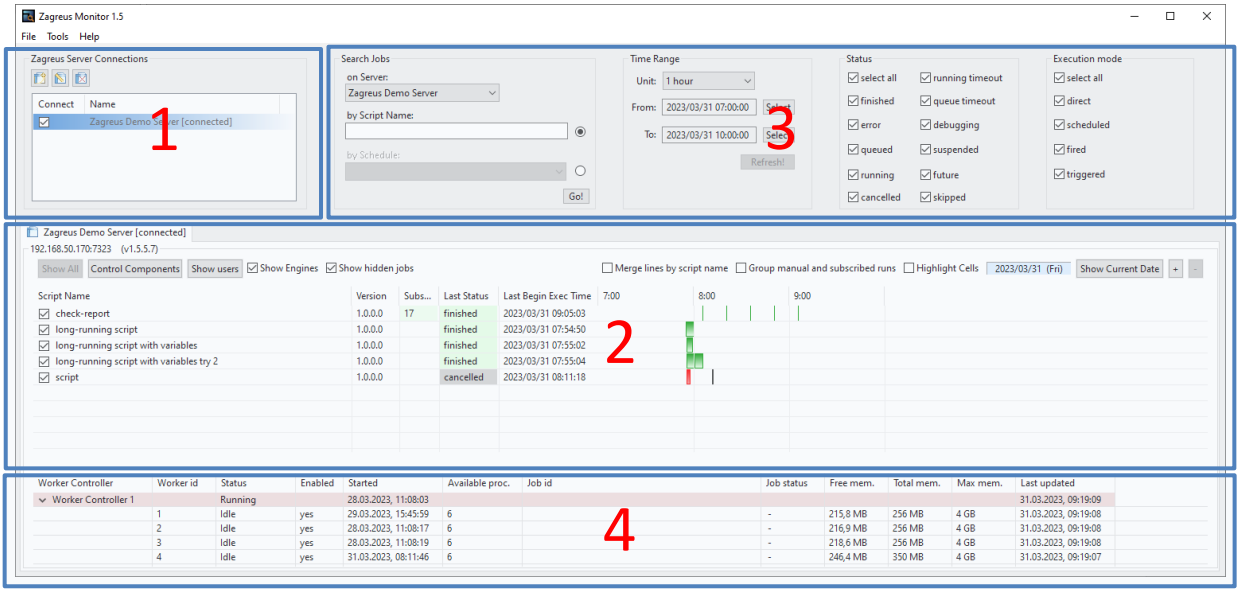


Figure 2 – Zagreus Monitor and its main sections

The sections are the following:

1) *Zagreus Server Connections*
   The area for administrating connections to one or more Zagreus servers, see → Zagreus Server.

2) *Timeline* window
   Graphical representation of scripts and their executed jobs, see → Queuing and jobs.

3) *Filter area*
   It contains various filters for the jobs shown in the timeline window.

4) *Execution Engine window*
   It shows the Worker Controller and its workers of the active server connection. This window is almost identical to the *Execution Engines* window in the Zagreus Client (see → Execution engines window).

11.1 Main menu bar

The main menu bar can be found at the top left corner of the Zagreus Monitor. The following menu items are available:

- *File / Exit*
  This menu item exits from the Zagreus Monitor application

- *Tools / Options*
  This menu item displays the *Options…* dialog with the server polling parameters (see *Figure 2.*).

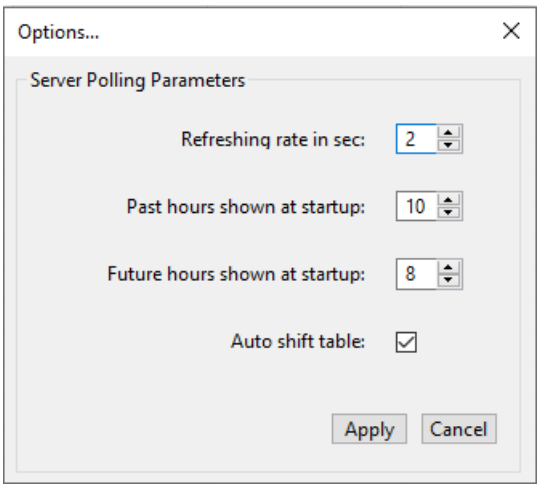


Figure 2 – Options dialog box

  - *Refreshing rate in sec*
    It sets the rate at which the jobs and scripts in the *Timeline* window are refreshed (specified in seconds).

  - *Past hours shown at startup*
    The number of hours prior to the startup time that are shown in the *Timeline* window. The setting takes effect after restarting the Zagreus Monitor.

  - *Future hours shown at startup*
    The future hours after the startup time that are shown in the *Timeline* window. The setting takes effect after restarting the Zagreus Monitor.

  - *Auto shift table*
    It shifts the *Timeline* window based on the current time in each hour.

- *Help / About*
  This menu item shows the Zagreus Monitor version and logo.

11.2 Sections of the Zagreus Monitor

Next, the aforementioned sections of the Zagreus Monitor application are described in details.

11.2.1 Zagreus Server Connections

The connections to the Zagreus servers can be specified here. Multiple Zagreus Server connections can be defined and opened at the same time. When a connection is established, all jobs and their associated scripts are displayed in the *Timeline* window. A host and port combination is allowed only once in the list of servers.

> ⚠ ***Warning:*** *Only users with administrator rights can use the Zagreus Monitor application. The non-admin users get a* Permission denied *error message when trying to connect.*

The three icons on the toolbar allow to manage server connections:

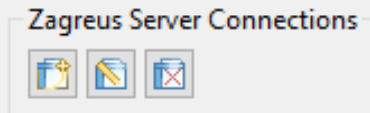


Figure 3 – The toolbar of the *Zagreus Server Connections* section

- *Add Server Connections* 
  By clicking on this icon, the *Add Server Connection* dialog box will be shown, see *Figure 4.*

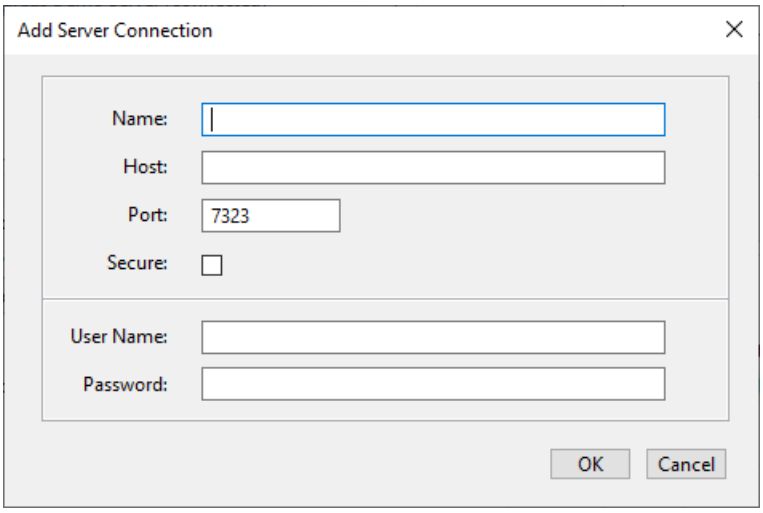


Figure 4 – The *Add Server Connection* dialog box

The following connection parameters must be specified:

- *Name*: the (human-readable) name of the remote connection
- *Host*: the hostname or IP address of the server
- *Port*: the port of the server
- *Secure*: if the connection uses a secure protocol (i.e. SSL)
- *User Name*: the user name
- *Password*: the user's password

- *Modify Server Connection* 
  By clicking on this icon, the *Modify Server Connection* dialog box will be shown, see *Figure 5.*

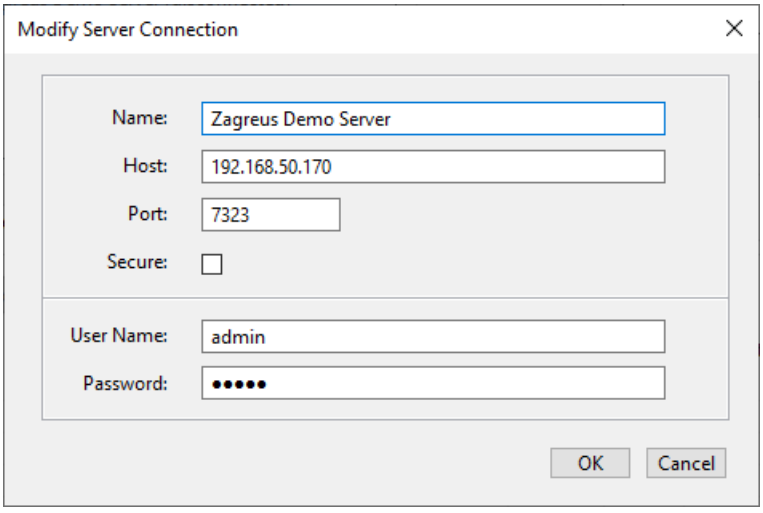


Figure 5 – The *Modify Server Connection* dialog box

- *Delete Server Connection* 
  It removes a server connection.

11.2.2 Timeline area

The *Timeline* area allows jobs and their associated scripts to be monitored. Each active server connection has its own separated Timeline area identified by the connection name and the server host:port in the corresponding tab header. *Figure 6.* shows one active server connection.

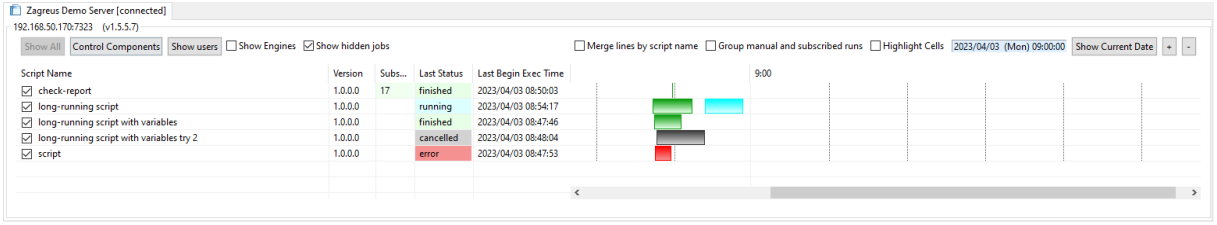
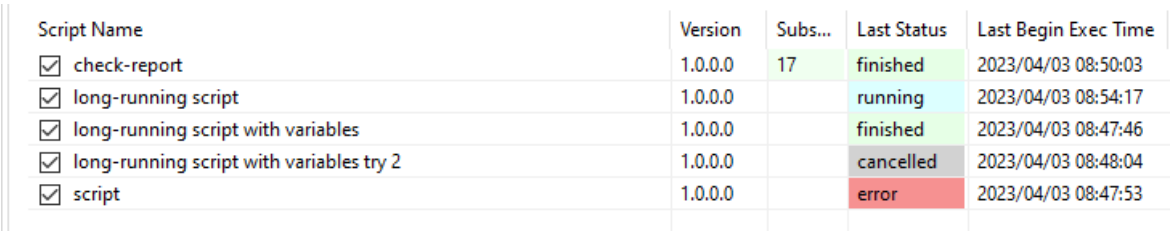


Figure 6 – The *Timeline* area

11.2.2.1 Left side of the area

The left side of the Timeline area shows the list of the monitored scripts, see *Figure 7.*



Script Name	Version	Subs...	Last Status	Last Begin Exec Time
check-report	1.0.0.0	17	finished	2023/04/03 08:50:03
long-running script	1.0.0.0		running	2023/04/03 08:54:17
long-running script with variables	1.0.0.0		finished	2023/04/03 08:47:46
long-running script with variables try 2	1.0.0.0		cancelled	2023/04/03 08:48:04
script	1.0.0.0		error	2023/04/03 08:47:53

Figure 7 – List of scripts

In this area, the following columns are displayed:

- *Script Name*
  The name of the script. Unchecking the checkbox before the script name removes the script from the list temporarily. It can be useful for specific monitoring purposes, e.g. too many scripts are shown in the list. For listing all the scripts again, the user needs to click on the *Show all* tool button, see → Additional options.

- *Version*
  The version of the script.

- *Subscription id*
  The subscription ID of the currently running or lastly executed job of the script (if any).

- *Last Status*
  For an actively running job, it is the current status (e.g. *running*, *debugging*, *suspended*); otherwise it is the status of the last job of the script (e.g. *finished*, *error*, *cancelled*).

- *Last Begin Execution Time*
  The last begin execution time of the script.

By double clicking on any of the lines in this list, the *Job properties* dialog box for the currently running or last executed job appears, see → Job properties dialog.

11.2.2.2 Right side of the area

The right side of the Timeline area shows an actual graphical representation of a timeline for the jobs that have been executed, are running or can be expected to be executed in the future (i.e. scheduled). The different lines correspond to the particular scripts listed in the left side of the Timeline area. Jobs are represented as rectangles over time: the width of the rectangle indicates the duration of execution. If the jobs are executed in a very short period of time or the timeline is zoomed out, the representation of the particular job can be seen as a simple vertical line.

When the time range does not fit to the currently displayed timeline, a horizontal scroll bar appears at the bottom of this window. See → Time Range filter.

Furthermore, to see the details of a job, clicking one of them opens the *Job properties* dialog box, see → Job properties dialog.

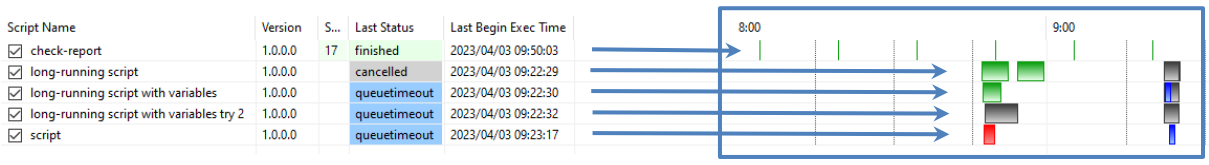


Figure 8 – Timeline for each script

Different colors represent the current status of the particular job:

- *green:* finished

- *gray:* cancelled
- *red:* error
- *light blue:* running
- *brown:* starting
- *purple:* debugging
- *blue:* queuetimeout

Zagreus Monitor also shows estimations in two further cases:

- *Scheduled scripts:*
  These are scripts that are subscribed to a time schedule, see → Subscriptions. If there are expected time events in the displayed time frame, they are shown as orange triangles in the Timeline, see *Figure 9.*

- *Skipped jobs:*
  A job is considered a skipped job if it had been scheduled but was not queued due to server shutdown or error, see → Skipped jobs. As *Figure 10*. shows, they are represented by red triangles.

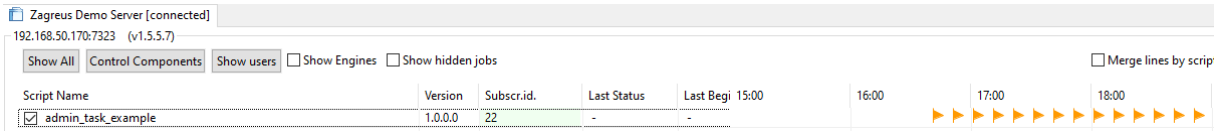


Figure 9 – Scheduled jobs

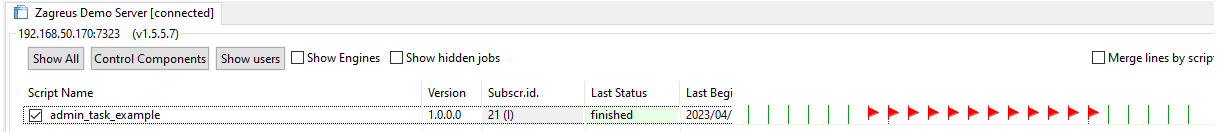


Figure 10 – Skipped jobs

Double-clicking on these virtual jobs also shows the Job properties dialog box with the corresponding information of the scheduled or skipped job.

11.2.2.3 Job properties dialog

This is the main dialog of the Zagreus Monitor application, showing lots of information of a selected job. The header of this dialog box displays the job id.

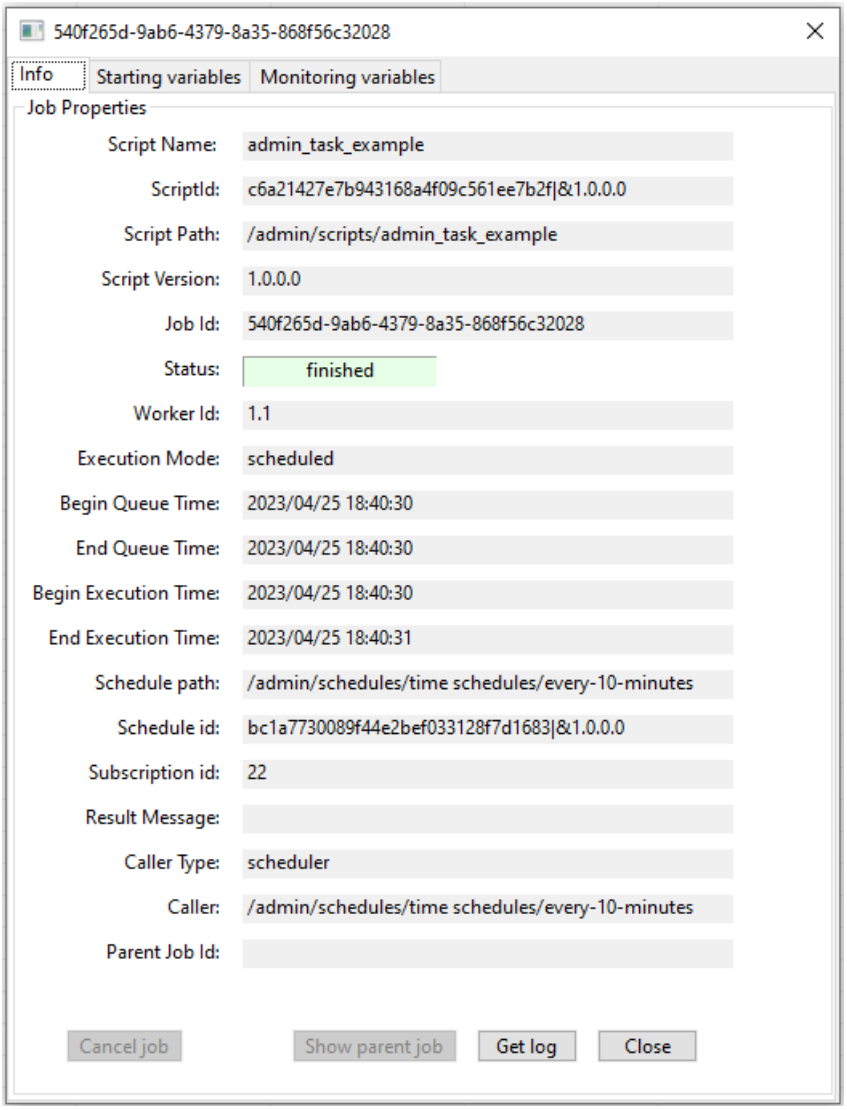


Figure 11 – The *Info* tab of the *Job properties* dialog box

There are three tabs on the pane of the Job properties dialog box: *Info*, *Starting variables* and *Monitoring variables*. The *Info* tab displays the general properties of the selected job:

- *Script Name*: the name of the script
- *Script Id*: the fully qualified id of the script (see → Resource ID and version)
- *Script Path*: the full path of the script
- *Script Version*: the version number of the script
- *Job Id*: the job id
- *Status*: the status of the job, using the same color codes as the Timeline window
- *Worker Id*: the id of the worker that executed the job
- *Execution Mode*: the execution mode of the job
- *Begin Queue Time:* the begin queue time of the job

- *End Queue Time:* the end queue time of the job
- *Begin Execution Time:* the begin execution time of the job
- *End Execution Time:* the end execution time of the job
- *Result Message:* the result message of the job, see → result-message of the script
- *Caller Type:* the caller type of the execution
- *Caller:* the additional caller information of the execution
- *Parent Job Id:* the parent job id, if the script execution was initiated by another script (i.e. by the `zs:runscript` action)

  For detailed descriptions about job properties, see → Job properties

The following properties are shown if the script execution was initiated by an event-type resource:

- *Schedule path:* the full path of the corresponding trigger
- *Schedule id:* the id and version of the corresponding trigger
- *Subscription id:* the id and version of the corresponding trigger

The bottom of the Info tab offers several buttons for the following functionalities:

- *Cancel job*
  Cancels the currently running job. Only available if the job status one of the following: *running*, *debugging*, *suspended*, *starting*. See also → Cancellation

- *Show parent job*
  This button switches the Job properties dialog to the parent job. Only available if the script execution was initiated by another script (i.e. by the `zs:runscript` action)

- *Get log*
  This button displays the log messages of the currently selected job in a new dialog *Finished Log* (see *Figure 12.*).

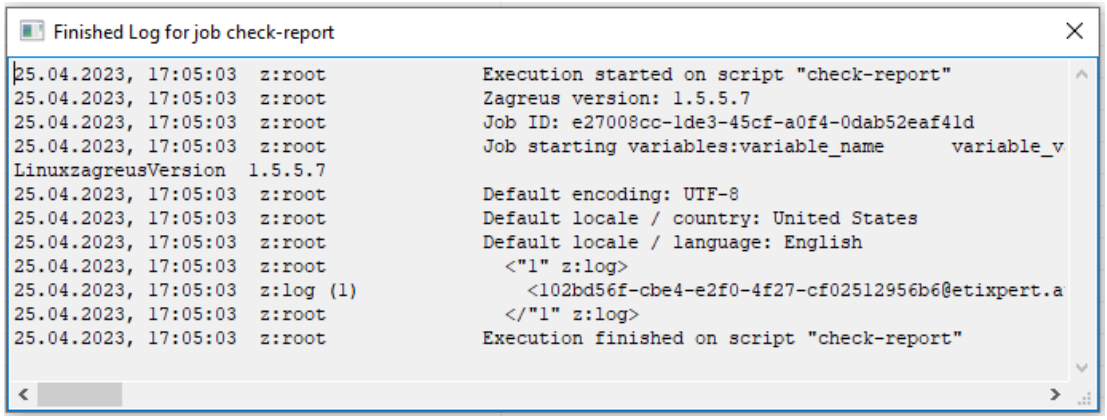


Figure 12 – Log messages of the selected job

- *Close*
  Closes the Job properties dialog

The *Starting variables* tab displays the starting variables of the selected job, see *Figure 13*. The *Advanced mode* checkbox shows the same list with the fully-qualified variable names according to their scopes, see also → Prefixes.

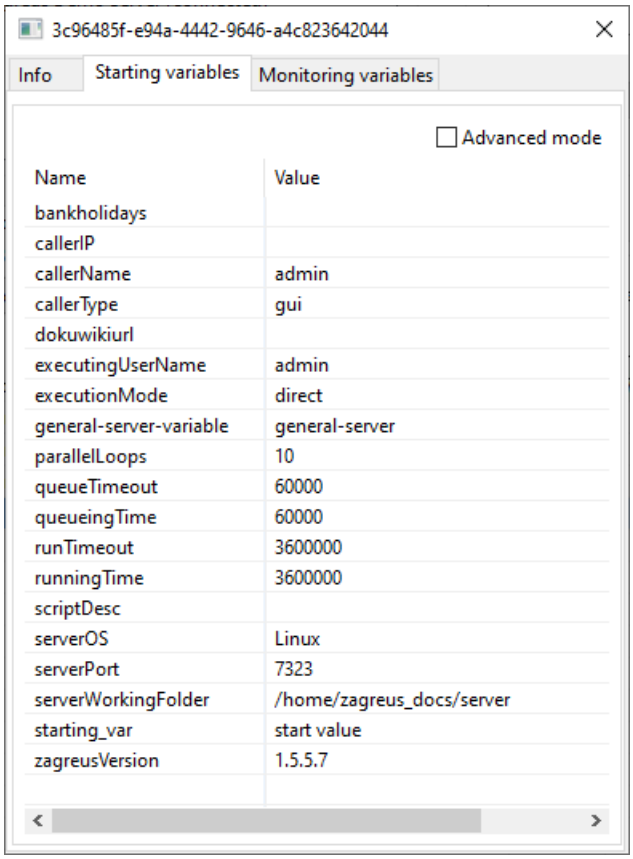


Figure 13 – Starting variables tab

The *Monitoring variables* tab displays special monitoring variables (see → Monitoring variables) of the selected job, see *Figure 14*.

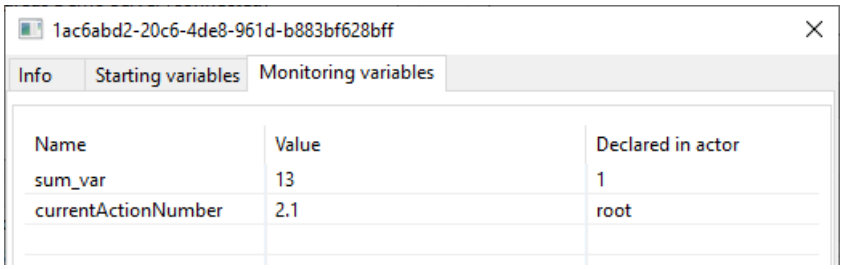


Figure 14 - See the changes in the value of the `sum_var` variable in the Zagreus Monitor

These are script variables with the attribute *monitor=true*, see *Figure 15*. Also, there is a built-in specific variable `currentActionNumber`, which is monitoring the ordinal number of the currently executed action, see also → Ordering numbers.

Aside from the name and value of the monitoring variables, the third column *Declared in action* shows the action ordinal number of variable declaration.

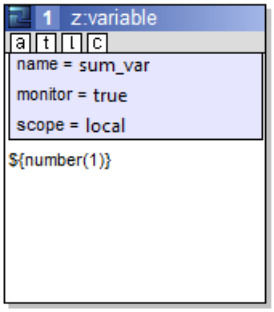


Figure 15 - Set monitor attribute to true for the `sum_var` variable in the Zagreus Client

> ***Info:*** *The content of the Job properties dialog box can be switched to another job either by clicking on the corresponding job in the right side of the Timeline window or double clicking on one of the listed scripts.*

11.2.2.4 Additional options

In this part of the area, additional options are available for managing server components, changing timeline view and sorting scripts.

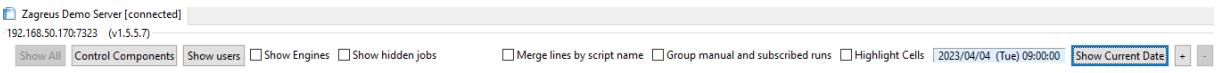


Figure 16 - The additional options of the Timeline window

The name of the connection, the hostname, the port and the server version are displayed in the header of the Timeline window.

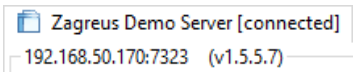


Figure 17 – The header of the Timeline window

On the toolbar of the tab, the following options are available:

- *Show All* button
  displays all scripts and their jobs that are executed within the time range specified. After removing some scripts from the list with the checkbox before the script name, it is the way to list them all again.

- *Control Components* button
  the *Server control* dialog box for the selected Zagreus server will appear, displaying the components:

  - *Scheduler*
    It starts / stops the Quartz scheduler of the Zagreus System, see → Quartz scheduler.

  - *Queue*
    It starts / stops the Zagreus job queue, see → Queue.

  - *Direct running*
    It starts / stops the possibility of manual job execution, see → Manual script execution.

  - *Filetriggers running*
    It starts / stops the file trigger functionality of the Zagreus server, see → File trigger.

  - *Watchers running*
    It starts / stops the mail watcher and database watcher functionality of the Zagreus server, see → Mail watcher and → Database watcher

- *Priority algorithm*
  It starts / stops the priority management of the Zagreus job queue, see → Priority and priority algorithm

All components can be enabled or disabled by right-clicking on the proper component, see *Figure 18.*

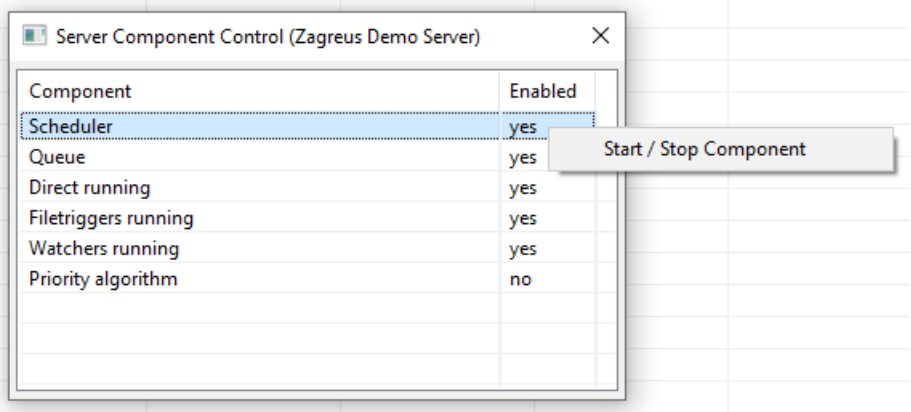


Figure 18 – The *Server control* dialog box

- *Show users* button
  The *Logged in users* dialog box will appear, displaying a list of currently logged-in users, see *Figure 19*.

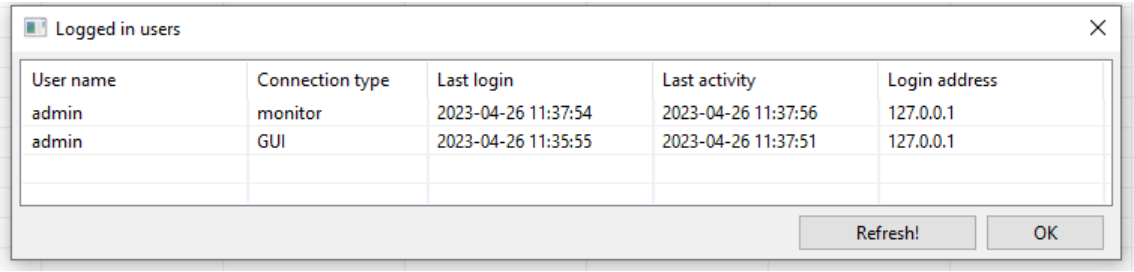


Figure 19 – The *Logged in users* dialog box

- *Show Engines* checkbox
  This checkbox displays the *Execution Engines* window, see → Execution Engines window

- *Show hidden jobs* checkbox
  Displays all hidden jobs as well, see → Hidden jobs

- *Merge lines by script name* checkbox
  When a script is executed both manually and by a triggered subscription, Zagreus Monitor displays them in separate lines in the script list of the Timeline window.

Furthermore, when the same script is executed by multiple subscriptions, they are also shown in separate lines. Using this checkbox, these lines are merged into a single line in the timeline.

- *Group manual and subscribed runs* checkbox
  It categorizes jobs based on whether they were executed manually or by a subscription. If set, the manually executed jobs are at the upper part of the script list.

- *Highlight Cells* checkbox
  If this checkbox is set, the hourly regions of scripts which have any job (including skipped and scheduled jobs) in the given region will be highlighted.

- *Date and Time display*
  It shows the date and time of the displayed starting position of the timeline.

- *Show Current Date* button
  It jumps to the current date and time on the timeline. This button is only available if the current date and time are within the range specified in the *Time Range* filter, see → Time Range filter.

- *Zoom In* and *Zoom Out* buttons
  These buttons allow the user to zoom in and out on the timeline.

11.2.3 Filter area

This area contains various filters for fine-tuning the displayed content of the Timeline window. It has four main parts. This is useful when the number of the executed scripts is relatively large, making the list of scripts hard to handle.

11.2.3.1 Search Jobs filter

By this filter the user can filter for script name or for a specific event-type resource associated with the scripts, see *Figure 20*.

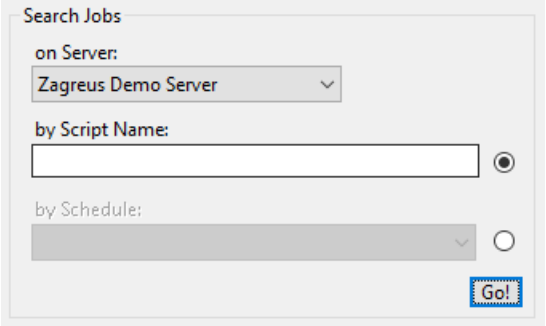


Figure 20 - Filtering by script name or schedules

- *on Server* dropdown
  The Zagreus Server for which the filter will be applied can be selected here (in case when multiple server connections are open).

- *by Script Name* textbox
  By selecting this option with the radio button to the right to the textbox, and by specifying the script name filter, only the scripts that contain the specified string will be shown in the script list of the Timeline window. Substrings can also be used, e.g. a value ”script” will result in showing the scripts *sample_script* and *example_script*, but hiding the *script sample_request*.

- *by Schedule* dropdown menu
  By selecting this option with the radio button to the right to the dropdown menu, and by choosing the full path of the event-type resource from the dropdown, the content of the Timeline window will be limited to the scripts that are subscribed to this event-type resource and were executed by those subscriptions.

The options above take effect after clicking on the *Go!* button.

11.2.3.2 Time Range filter

The user can use the Time Range filter to set the time range of the timeline displayed in the Timeline window, see *Figure 21*.

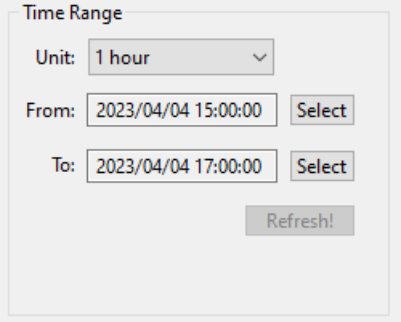


Figure 21 – The *Time Range* filter

- *Unit*
  the unit that the time range uses can be specified here. The available values are as follows: `1 hour` and `1 day`.

- *From*
  the starting date and time can be specified here. A calendar with the current date appears after clicking the `Select` button. Only hourly precision can be used here since the Timeline window units are hourly-based.

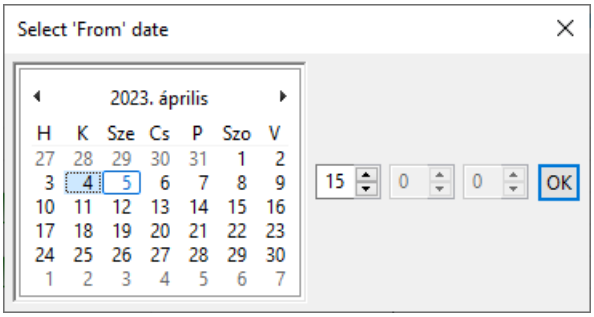


Figure 22 - Calendar for the starting date

- *To*
  The ending date and time can be specified here. A calendar with the current date appears after clicking the `Select` button. Only hourly precision can be used here since the Timeline window units are hourly-based.

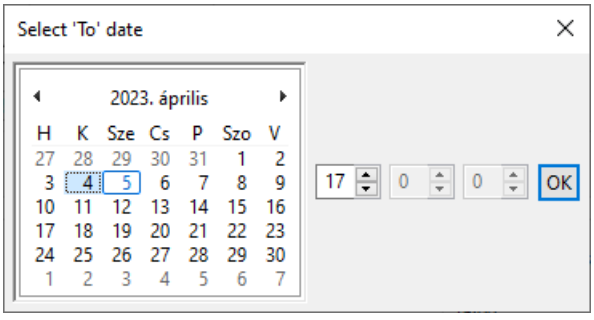


Figure 23 - Calendar for the ending date

Clicking the *Refresh!* button applies the changes on the Timeline.

11.2.3.3 Status filter

This filter selects the status of the jobs that will be displayed in the Timeline window. See → Job lifecycle

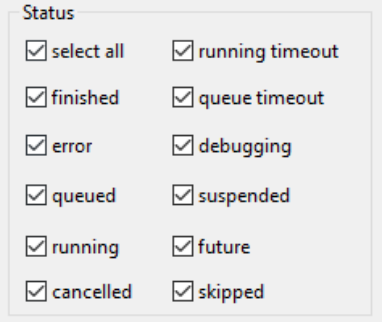


Figure 24 – The Status filter

Aside from the selectable statuses, the *select all* checkbox behaves as a toggle button, the user can select or deselect all statuses at once.

The timeline is updated right after any of the checkboxes has changed.

11.2.3.4 Execution mode filter

This filter selects the execution mode of the jobs that will be displayed in the Timeline window, see → Job properties.

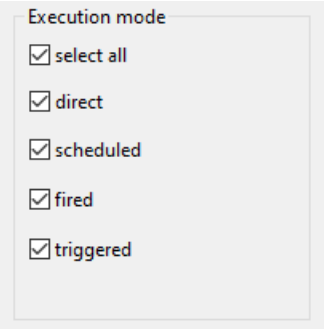


Figure 25 - Execution mode filter

Aside from the selectable execution modes, the *select all* checkbox behaves as a toggle button, the user can select or deselect all execution modes at once.

The timeline is updated right after any of the checkboxes has changed.

11.2.4 Execution Engines window

It displays the current status of all Execution Engines (i.e. Zagreus Workers) shown in a tree-table. The Zagreus Workers are located under the Worker Controller they belong to. The Worker Controller tree node is expandable and collapsable either by clicking on the arrow right next to the Worker Controller name or by double-clicking on the Worker Controller name. When a script is being processed, its job will be displayed under the *Job id* column next to the worker who is executing it. Furthermore, the job status will be displayed in the *Job status* column, allowing the current status of the job to be tracked. See the details of the columns below.

- *Worker Controller* (expandable node)
  Shows all Worker Controllers.

- *Worker id*
  The ID of the particular Zagreus Worker.

- *Status*
  The status of the Worker Controller / Zagreus Workers.

- *Enabled*
  Shows if the Zagreus Worker is enabled or disabled.

- *Started*
  Shows the date when the Zagreus Worker was started.

- *Available processors*
  The number of CPU cores in the Zagreus Server, reported by the JVM.

- *Job id*
  The identifier of the actual job (if there is a running job).

- *Job status*
  The current status of the job (if there is a running job).

- *Free memory*
  The currently free memory that the JVM can use, reported by the JVM, see → Memory handling

- *Total memory*
  The actual memory that the JVM is using, reported by the JVM, see → Memory handling

- *Max memory*
  This is the maximum memory that the JVM can use, reported by the JVM, see → Memory handling.

- *Last updated*
  The most recent time all information was updated.

12. Other Zagreus clients

Besides the Zagreus Client, there are further custom clients provided with Zagreus, which offer much more limited functionalities, but which are light-weight and simple to use. Both the *Zagreus Command-line tools* and the *Zagreus HTML application* are shipped with the Zagreus installation and work out-of-the-box. Additionally, command-line tools can be installed as a standalone package on other machines as well. In contrast, the Zagreus html application requires to be hosted on a Zagreus server. Both clients allow the user to initiate script execution and to fire event schedules. These clients improve the accessibility of the Zagreus infrastructure because the Zagreus server can be accessed from:

- a machine without a graphical user interface,
- a machine that does not have an installed Zagreus Client,
- or even from mobile devices (smartphones, iPads, etc.).

12.1 Command-line tools

The Zagreus command-line tools, shipped along with Zagreus installations as a standalone package, consist of a collection of simple executable files. To support both Windows and Unix platforms, there is a `.bat` and an `.sh` script file available for every functionality supported. There are batch files and shell scripts for administrative tasks as well as for running scripts and firing events.

The command-line package for the Windows environment consists of the following files: `backupdatabase.bat`, `configtester.bat`, `connectedusers.bat`, `fireevent.bat`, `restoredatabase.bat`, `runscript.bat`. Of course, the same files are provided with `.sh` extension, adapted for Unix environment.

The main purpose of the command-line package is to allow the user to initiate script execution, but it can also be used for administrative purposes. It can be useful when it is not possible to install a Zagreus Client application on the client machine, or when the particular administrative task has to be performed from another application.

12.1.1 Executable files

There are six executable files with the extension of `.bat` or `.sh` (Windows and Unix environments, respectively). Since they are batch files / shell scripts, they can be copied and edited according to the actual use case.

12.1.1.1 backupdatabase script

This command-line tool creates a backup file of the database instance of the Zagreus Server. By default, the output of the backup process will be saved in the `backup` folder within the Zagreus Server home folder (e.g. `c:\Programme\zagreus\server\` or `/home/zagreus/server/`). The output file will be assigned the name `backup-<yyyy-MM-dd_hhmmss>.sql`, where `<yyyy-MM-DD_hhmmss>` stands for the current timestamp (for example, `backup-2023-07-01_121314.sql`).

12.1.1.2 configtester script

This command-line tool checks the content of the configuration files of the Zagreus Server, Zagreus Worker and Zagreus Workercontroller components. It sends feedback

about property keys without value definition, and marks property keys which are deprecated.

12.1.1.3 connectedusers script

This command-line tool returns information about the users who are currently connected to the Zagreus Server. This data is displayed in a table format, for example:

```
User     Connection type   Last login          Last activity       Login address
admin    GUI               2023-06-07          2023-06-07          127.0.0.1
                           13:02:23            13:22:41
```

12.1.1.4 fireevent script

This command-line tool triggers the given event schedule, specified by its resource ID or by its resource path. The result can be checked in the Zagreus Client or in the Zagreus Monitor applications.

12.1.1.5 restoredatabase script

This command-line tool restores the Zagreus Server meta database from a backup file (created by e.g. the `backupdatabase` command-line tool).

12.1.1.6 runscript script

This command-line tool initiates the execution of the given script, specified by its resource ID or by its resource path. The result can be checked in the Zagreus Client or in the Zagreus Monitor applications.

Note: In a Windows environment, the command line window closes automatically after the end of the process.

12.1.2 Examples for Windows

Next we present some use-case examples for executing command line files and configuring their input parameters in a Windows environment. In these examples, the

folder path of the command line tools is `c:\Programme\zagreus\command-line\` (i.e. the home folder of the Zagreus Server is `c:\Programme\zagreus`).

12.1.2.1 Initiating script execution

Initiating script execution without parameters:

```
C:\Users\DemoUser> cd C:\Programme\zagreus\command-line
C:\Programme\zagreus\command-line> runscript.bat h=my.zagreus.server
-p=7323 -u=admin -pass=******** d0a26bbd10194bf69460f770b3d8d9ff
```

Initiating script execution with the parameter *secure*, passing script variables and using encrypted password (i.e. *cpassword*):

```
C:\Users\DemoUser>cd C:\Programme\zagreus\command-line
C:\Programme\zagreus\command-line>runscript.bat -h= my.zagreus.server
-secure -p=7443 -u=admin -cpass=#-3#67#-49#-********
-param=x:my-input-param -param=y:123 e275a121e9304a3290c00bfc6afcc756
```

General pattern for initiating script execution:

```
runscript.bat -h=host -p=port -u=user [-pass=password or -cpass=cpassword]
[-secure] [-param=name:value [-param=name:value ...]] <script_id>
```

12.1.2.2 Triggering an event schedule resource

Triggering an event schedule resource with the parameter *secure*, and using encrypted password (i.e. *cpassword*):

```
C:\Users\DemoUser>cd C:\Programme\zagreus\command-line
C:\Programme\zagreus\command-line>fireevent.bat -h= my.zagreus.server -
p=7443 -u=admin -cpass=#-3#67#-49#-********* -secure
7529d6d7efeb4160ab0d8cc5084b6d7f
```

General pattern for triggering an event schedule resource:

```
-h=host -p=port -u=user [-pass=password or -cpass=cpassword] [-secure]
```

> ***Info:*** *When using the* `-secure` *command-line parameter, the port must be changed to the Zagreus SSL port.*

12.1.3 Examples for Linux

Next we present some use-case examples for executing command line files and configuring their input parameters in a Linux environment. In these examples, the folder path of the command line tools is `/home/zagreus/command-line` (i.e. the home folder of the Zagreus Server is `/home/zagreus`).

12.1.3.1 Initiating script execution

Initiating script execution on localhost without parameters:

```
/home/zagreus/command-line/runscript.sh -h=localhost -p=7323 -u=admin -
pass******* e275a121e9304a3290c00bfc6afcc756
```

Initiating script execution on localhost with the parameter *secure*, passing script variables and using encrypted password (i.e. *cpassword*):

```
/home/zagreus/command-line/runscript.sh -h=127.0.0.1 -secure -p=7443
-u=admin -cpass=#-3#67#-49#-********** -param=x:my-param
-param=y:456 e275a121e9304a3290c00bfc6afcc756
```

General pattern for initiating script execution:

```
/home/zagreus/command-line/runscript.sh -h=host -p=port -u=user
[-pass=password or -cpass=cpassword] [-secure] [-param=name:value
[-param=name:value ...]] <script_id>
```

12.1.3.2 Triggering an event schedule resource

Triggering an event schedule resource with the parameter *secure*, and using encrypted password (i.e. *cpassword*):

```
/home/zagreus/command-line/fireevent.sh -h= my.zagreus.server -secure
-p=7443 -u=admin
-cpass=#-3#67#-49#********** 7529d6d7efeb4160ab0d8cc5084b6d7f
```

General pattern for triggering an event schedule resource:

```
/home/zagreus/command-line/fireevent.sh -h=host -p=port -u=user
[-pass=password or -cpass=cpassword] [-secure]
```

Info: *When using the* `-secure` *command-line parameter, the port must be changed to the Zagreus SSL port.*

12.2 Zagreus HTML application

The Zagreus HTML Application is a user interface for initiating script execution, triggering event schedules and get information about jobs. It is shipped with each Zagreus installation and hosted by the Zagreus Server.

This client can be accessed directly via a web browser. When defining the URL in the browser, it is necessary to define the port where the Zagreus Server publishes the HTML application. It is possible to reach Zagreus server on the standard or on the SSL port, see → General properties and → SSL properties.

Sample URL for reaching Zagreus HTML client with HTTPS protocol:

```
https://my-zagreus-server:7443/zagreus/html/zagreus.html
```

The Zagreus Server uses a self-signed certificate, therefore the Zagreus HTML application site has to be added to the security exceptions in the web browser used (see *Figure 1.*).

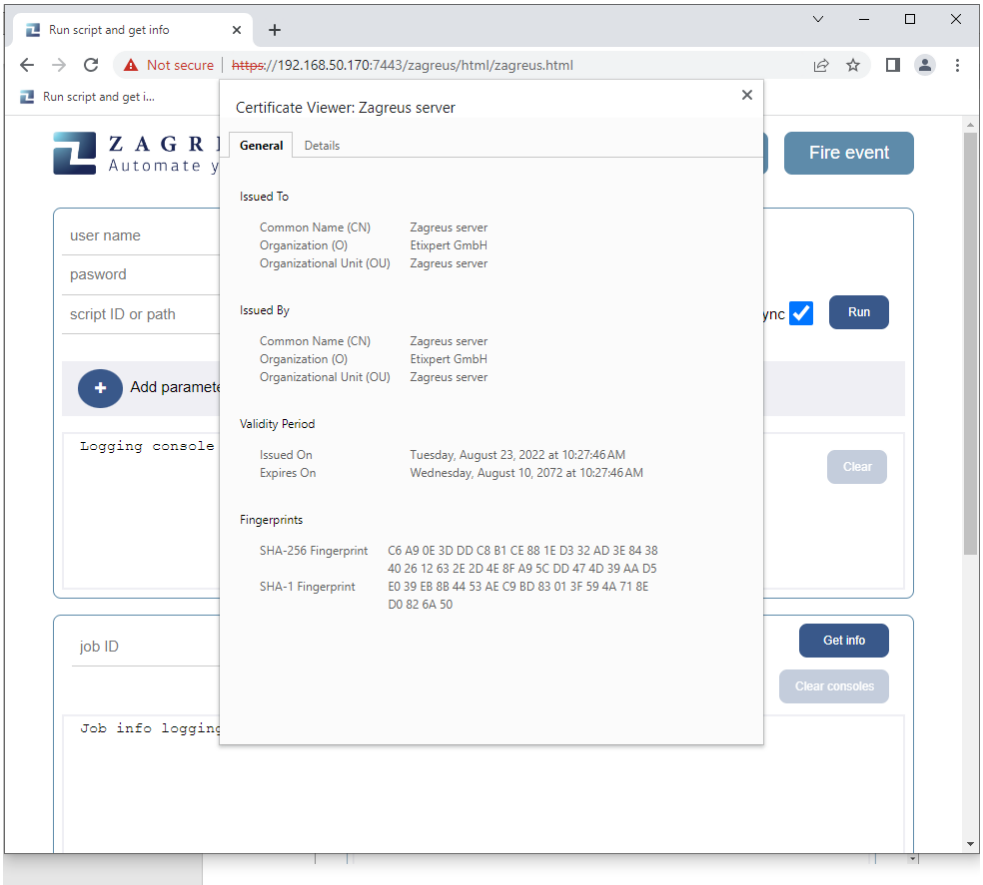


Figure 1 – The Zagreus Server certificate in a Chrome browser

After adding the Zagreus HTML application site as a security exception, it can be reached from the web browser.

For the structure of the Zagreus HTML application, see *Figure 2.* and *Figure 3*.

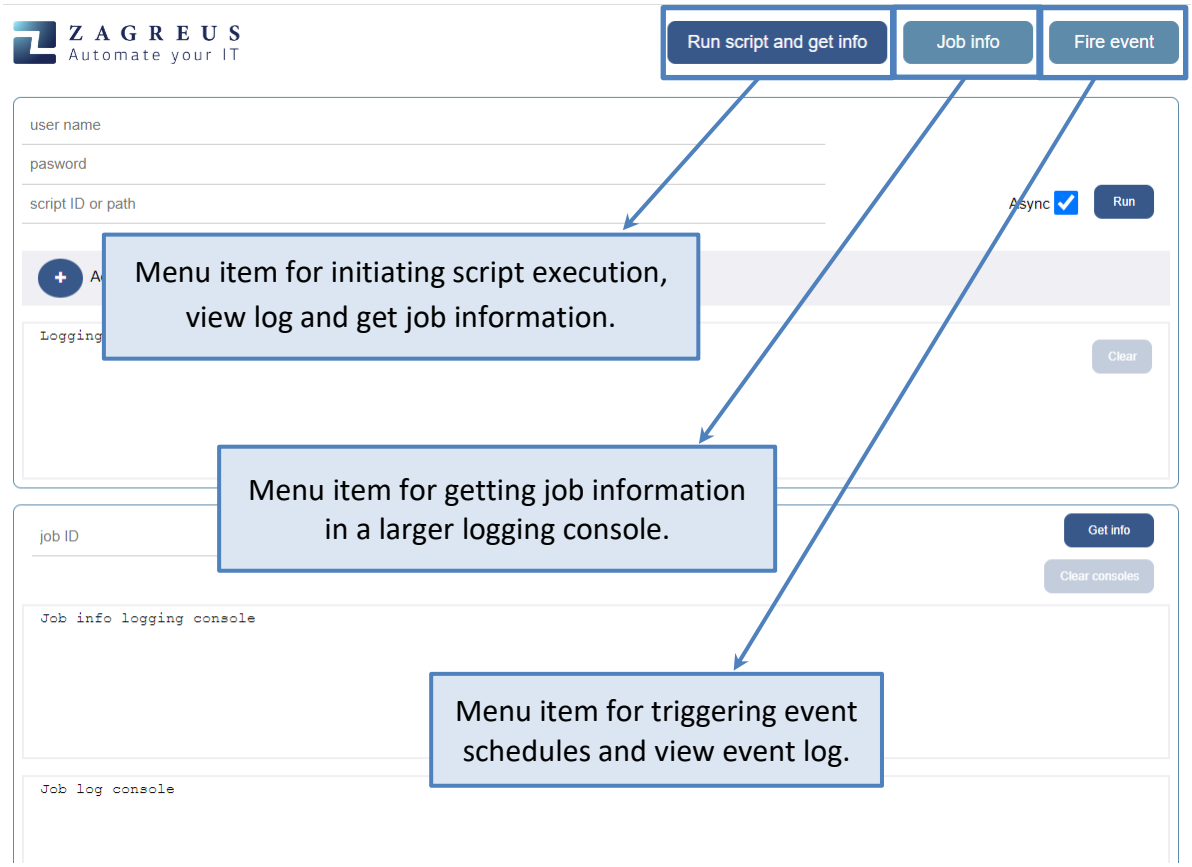


Figure 2 – The Opening screen of the Zagreus HTML application

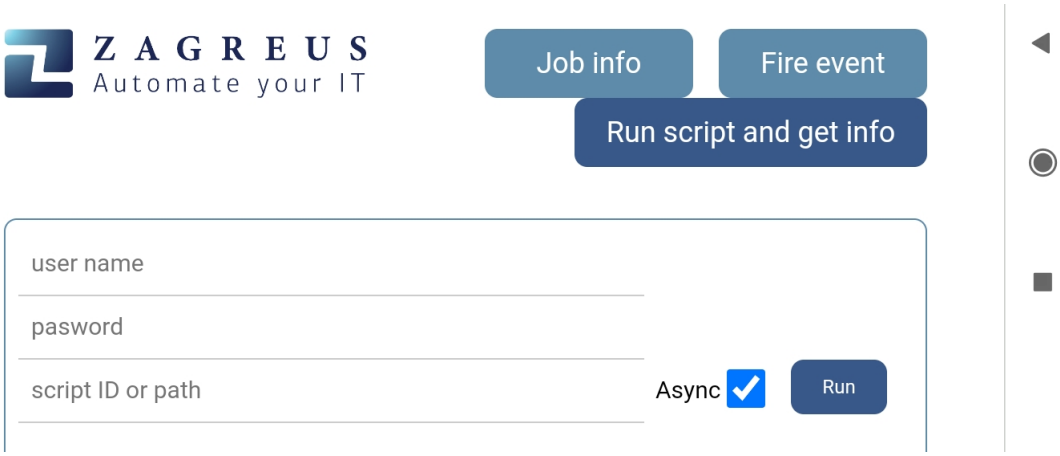


Figure 3 – The Zagreus HTML application opened in the browser of a smartphone.

12.2.1 *Run script and get info* tab

In this page the user can directly initiate script execution, define parameters for scripts, check logs, and get job information. There are three mandatory input fields:

- username
- password
- script ID / script path

Adding script parameters is possible by opening the *Add parameters* section. The result of the job is displayed in the *Logging console*. In the *Job info logging console* frame, the job-log can be viewed in the following way: copy the *job ID* from the *Logging console* to the *job ID* input field in the *Job info logging console* frame.

Scripts can be executed either in synchronous or in asynchronous mode, the default mode being asynchronous. By using the *Async* checkbox, the execution mode can be changed. Scripts can be executed by clicking the *Run* button. Consoles can be cleared with the *Clear / Clear console* button.

Figure 4 - Main controller items of the *Run script and get info* screen

12.2.2 *Job info* tab

This menu item is used for viewing the job-log entries. The number of displayed lines is configurable. In this page, there are three mandatory input fields:

- username
- password
- job ID

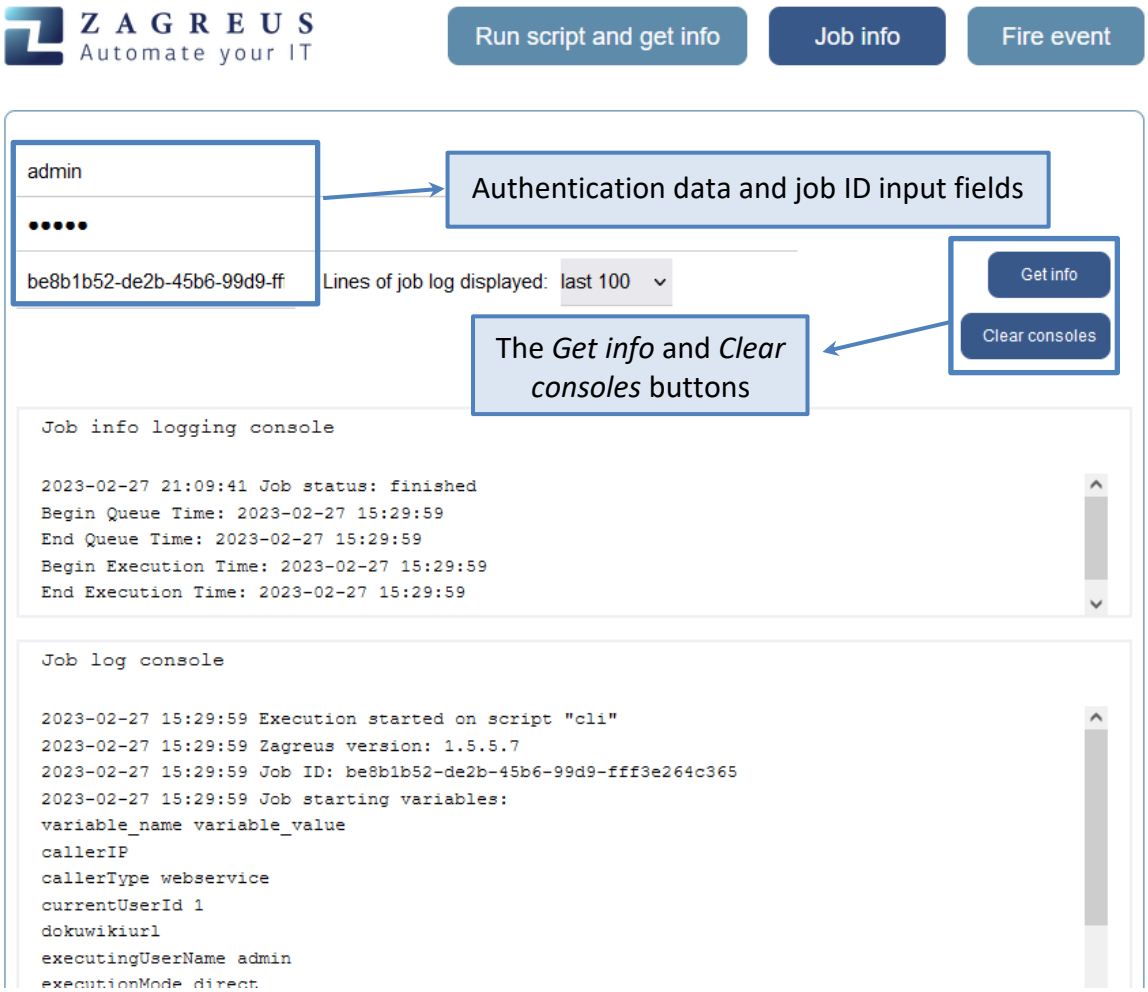


Figure 5 - Job info screen with job-log entries

12.2.3 *Fire event* tab

The user can trigger event schedules and check event logs on this screen. There are three mandatory input fields:

- username
- password
- event ID / event path

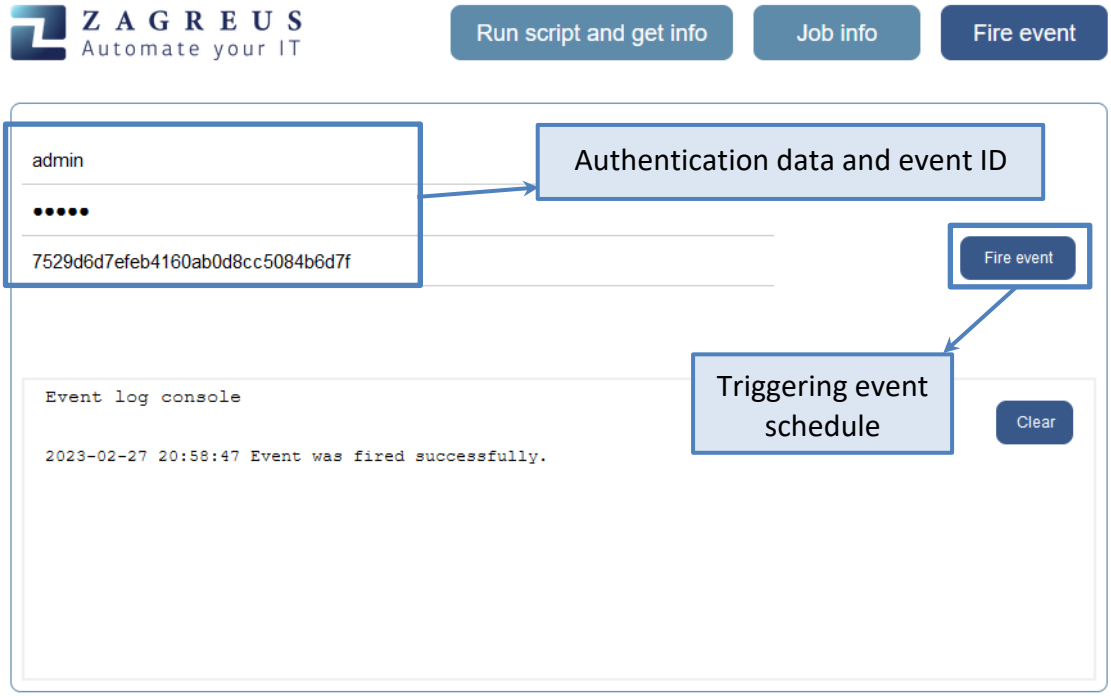


Figure 6 - Triggering an event schedule and viewing the result

12.3 Troubleshooting

When the execution of a script is initiated from command line, the result can be checked in the Zagreus Client, *Finished jobs* window, see → Finished jobs window. If the *Finished jobs* window does not contain the result, then the script was not executed. In such cases, it is recommended to inspect the following items:

- server definition in the command
- port definition in the command
- port definition in the Zagreus Server configuration, see → General properties and → SSL properties
- user authentication data
- server machine firewall settings
- server machine and router port forwarding settings

If one of the pages in the Zagreus HTML application does not load, it is recommended to inspect the following items:

- is the Zagreus Server running?
- port value in the URL
- port value in the Zagreus Server configuration, see → General properties and → SSL properties

Feedback message about script execution initiation is displayed immediately in the *Logging console* section, see → Run script and get info tab.

13. Script Editor

The Zagreus Script Editor is the tool for implementing the core of IT processes - creating Zagreus scripts. Scripts are composed from actions, see → Actions. Actions are able to exchange information with one another and also to embed other actions as sub-elements. The fundamental step in writing Zagreus scripts is making these links between actions. Additionally, Zagreus scripts can communicate with one another to build larger processes.

13.1 Layout

The Script Editor can be opened either by creating a new script resource (see → Scripts) or by double-clicking on an already existing script resource in the Zagreus Browser window.

The Script Editor consists of three main regions:

- Canvas
- Palette
- View selector

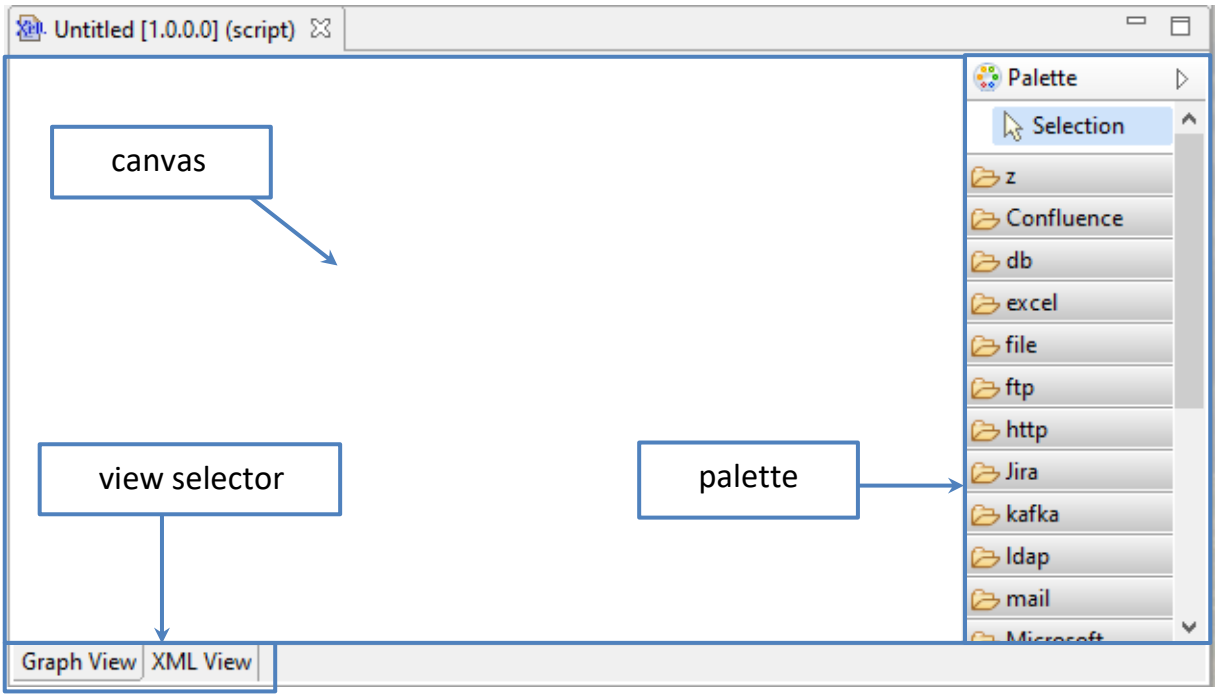


Figure 1 – The layout of the Script Editor

The *canvas* is the area for creating the content of Zagreus scripts. The *palette* is the container element of the available *action groups* and *actions,* see → Action groups and action name. Actions can be drag-and-dropped from the palette to the canvas. The user can choose between the *Graph View* and the *XML View* of the script via the *View selector*.

13.1.1 Canvas

The canvas represents the `z:root` parent action of the Zagreus script. In case of a new script the canvas is empty. Actions can be drag-and-dropped from the palette onto the canvas, they will become the direct children of the root element → XML representation. The canvas area is theoretically infinite; scrollbars appear when there is any content in the non-visible region. The content of the canvas can be zoomed in or out with the Zoom display and Zoom In / Out tools in the main toolbar, see → Zoom in and Zoom out.

Aside from the actions, the canvas can display specific relations between actions, like the *links from siblings* (see → Showing sibling links) and *child-content arrows* (see → Outside displaying option for a child action).

13.1.2 Palette

The palette is a container element of the available *action groups* and *actions*. The list of available action groups is controlled by the installed Zagreus licence (see → Licencing). An action group is a container element of particular actions with the same namespace, see → Action groups and action name. For example in *Figure 2.*, the action group *db* contains database related actions.

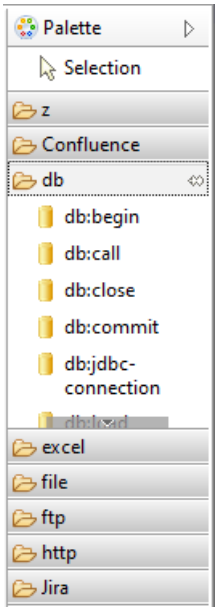


Figure 2 – The `db` action group on the palette

To open or close an action group in the palette, the user needs to click on its name. The action items can be scrolled with the scroll wheel of the mouse. Up and down keys

also can be used for navigation, while the *left* cursor key closes and the *right* cursor key opens an action group.

It is configurable which items are displayed on the palette, see also → Palette tab.

13.1.3 View selector tabs

It is possible to switch between *XML View* and *Graph view* by using the View selector tabs. The Graph View (see *Figure 3.*) contains the canvas and the palette, and the XML View (see *Figure 4.*) shows the XML representation of the Zagreus script, see → XML representation.

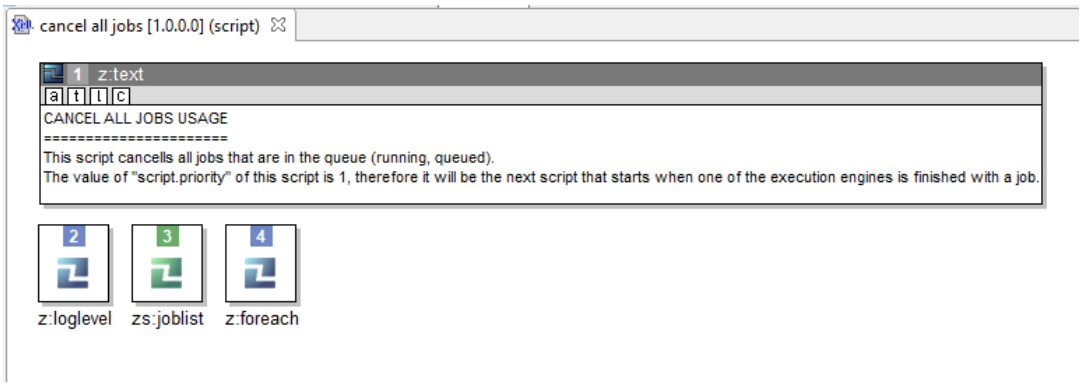


Figure 3 – A script in Graph View

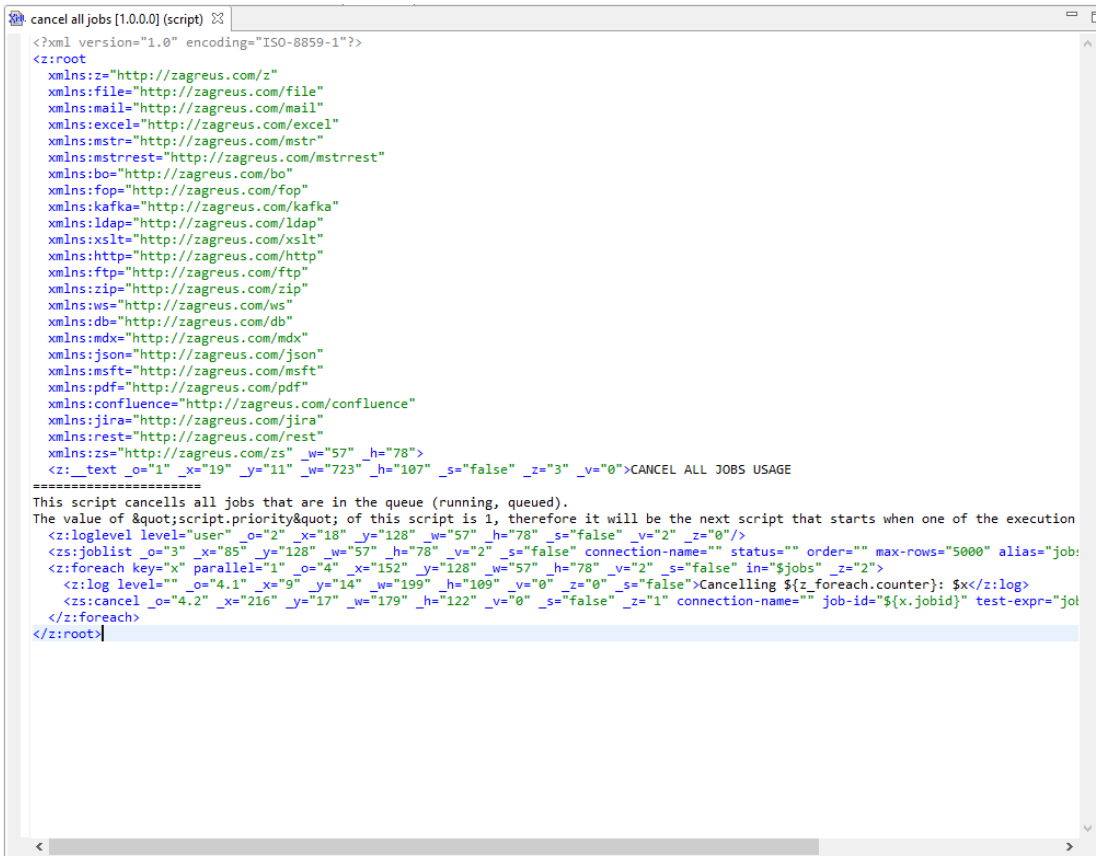


Figure 4 – The same script in XML View

Any changes are made on the XML View is applied on Graph View and vice versa.

13.2 Actions

Actions are the building blocks of Zagreus scripts. Actions are divided into logical groups – called action groups, e.g. `file`, `mail` or `db`. In full view mode, actions display their attributes and child content. Actions can be created, repositioned, deleted, copied etc. on the Graph View tab via specific user operations.

13.2.1 View modes

A view mode is the way of displaying an action. The view mode determines only the graphical representation of the action on the canvas, it does not affect either any operation performed on the action (e.g. copy, move, delete, comment) or the execution process of the script.

There are two types of view modes: 'closed' and 'open' ones. In a closed view mode (the *iconized* view mode), the sub-elements of the particular action such as child actions or text are hidden. In contrast, in an open view mode (all the other view modes) the sub-elements are fully visible.

13.2.1.1 *Iconized* view mode

The *iconized* view mode is a closed view mode and it has the smallest size among all the view modes. When an action is in iconized view mode, it hides all the sub-tree of the underlying script structure, so it can be regarded as a collapsed tree-node in a tree-like structure (like the script XML itself). It is a very useful way of hiding sub-elements of an action to provide a much cleaner overview for large scripts.

In the iconized view mode, only the following information are shown (see *Figure 5.*):

- the ordering number of the action (see → Ordering numbers)
- the icon of the action group
- the fully qualified name of the action (see → Action groups and action name)

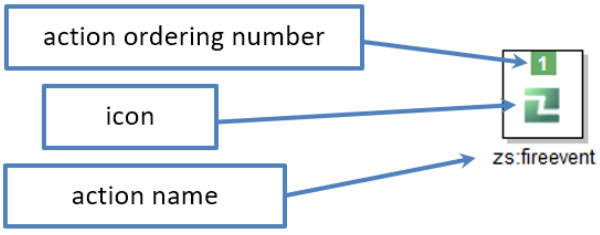


Figure 5 – The *iconized* view mode

> ***Info:*** *When a label is set for an action, the label is displayed instead of the action name.*

13.2.1.2 *Default* view mode

The *default* view mode is one of the 'open' view. This means that the contents of the action (such as child actions or text elements) are fully visible. Just because the particular action is in an open view mode, any of its children still can be iconized. In contrast with other open view modes, in the default view mode the attributes of the action are not visible.

In the default view mode, the following information are shown (see *Figure 6.*):

- the icon of the action group (or of the particular action, see → Action groups and action name)
- the ordering number of the action (see → Ordering numbers)
- the fully qualified name of the action (see → Action groups and action name)
- the toolbar of the action, containing specific tools for editing the properties of the action
- the content area, which shows the sub-elements of the action (see → Action content)

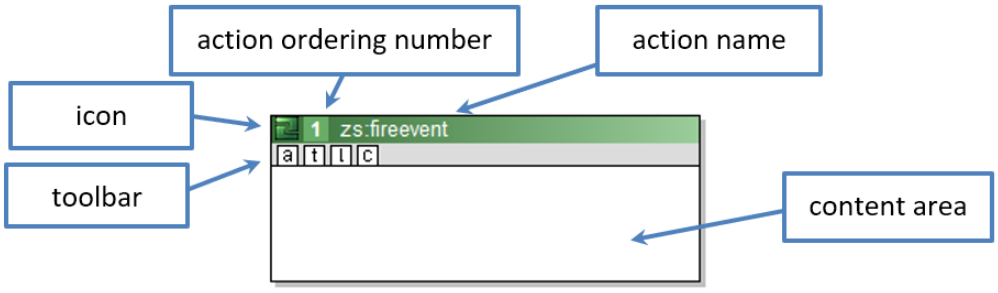


Figure 6 – The *default* view mode

13.2.1.3 Only filled view mode

The *only filled* view mode is another 'open' view mode. This means that the contents of the action (such as child actions or text elements) are fully visible. Just because the particular action is in an open view mode, any of its children still can be iconized. In this view mode the already filled attributes of the action are visible (the attributes with non-empty values).

In the only filled view mode, the following information are shown (see *Figure 7.*):

- the icon of the action group (or of the particular action, see → Action groups and action name)
- the ordering number of the action (see → Ordering numbers)
- the fully qualified name of the action (see → Action groups and action name)
- the toolbar of the action, containing specific tools for editing the properties of the action
- the list of the attributes of the action which have non-empty value (see → Action attributes)
- the content area, which shows the sub-elements of the action (see → Action content)

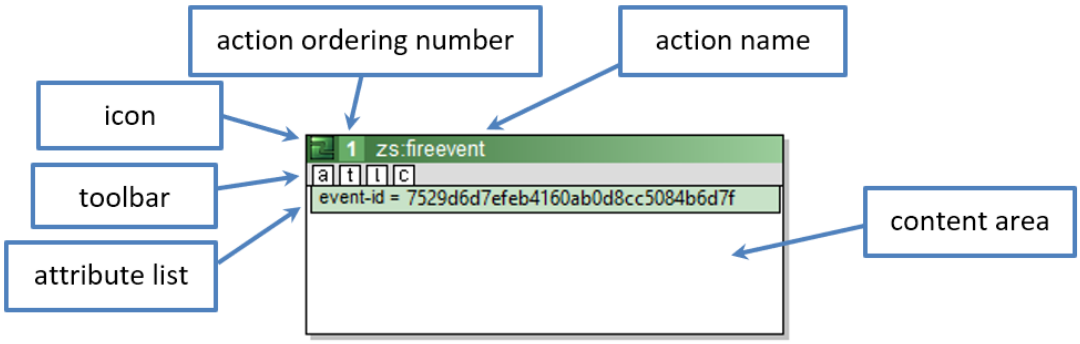


Figure 7 – The *only filled* view mode

13.2.1.4 The *full* view mode

The *full* view mode is the last 'open' view mode. This means that the contents of the action (such as child actions or text elements) are fully visible. Just because the particular action is in an open view mode, any of its children still can be iconized. In this view mode all the attributes of the action are visible.

In the full view mode, the following information are shown (see *Figure 8.*):

- the icon of the action group (or of the particular action, see → Action groups and action name)
- the ordering number of the action (see → Ordering numbers)
- the fully qualified name of the action (see → Action groups and action name)
- the toolbar of the action, containing specific tools for editing the properties of the action
- the list of all the attributes of the action (see → Action attributes)

- the content area, which shows the sub-elements of the action (see → Action content)

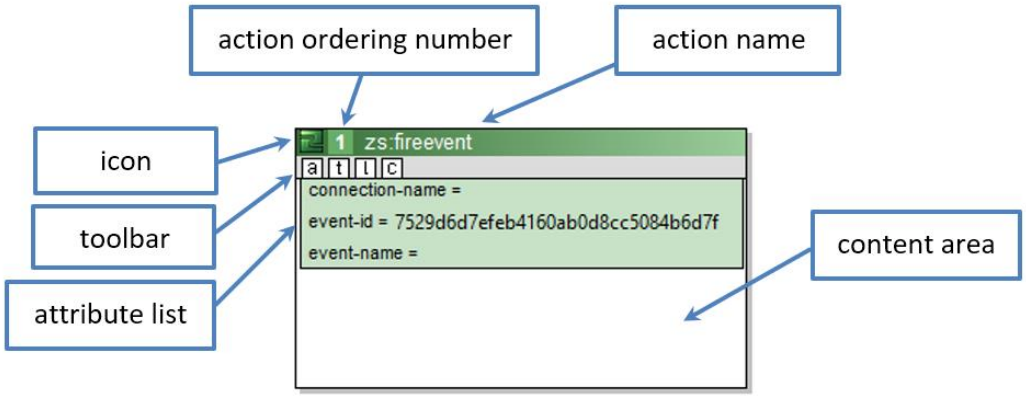


Figure 8 – The *full* view mode

This view mode provides the most information about an action.

13.2.1.5 Switching between view modes

Switching between the view modes are possible in the following ways:

- by selecting the target view mode in the action context menu (see *Figure 9.*)
- by double-clicking on the action header (in open view modes) or on the action icon (in iconized mode). This operation switches from any open view modes to the iconized view, while from the iconized view mode it switches to the open mode configured in the Zagreus Client options, see → Graph Editor tab.

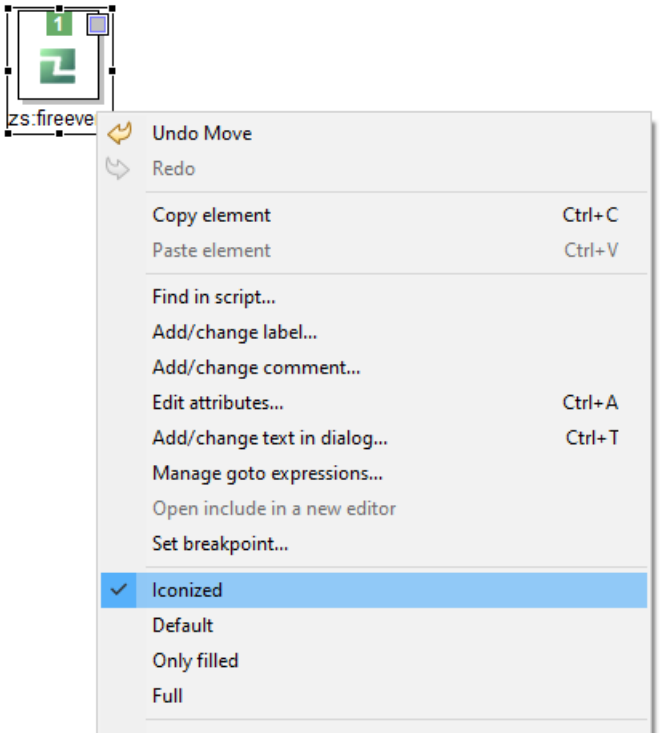


Figure 9 – Switching between view modes

13.2.2 Basic operations

Users can perform several operations in order to create and maintain Zagreus scripts. In this section, the operations related to the script structure are discussed.

13.2.2.1 Create

Drag-and-dropping an action from the palette is the simplest way to add one to a script, see *Figure 10*. When an action is created on the canvas, it is displayed in the *iconized* view, see also → Iconized view mode.

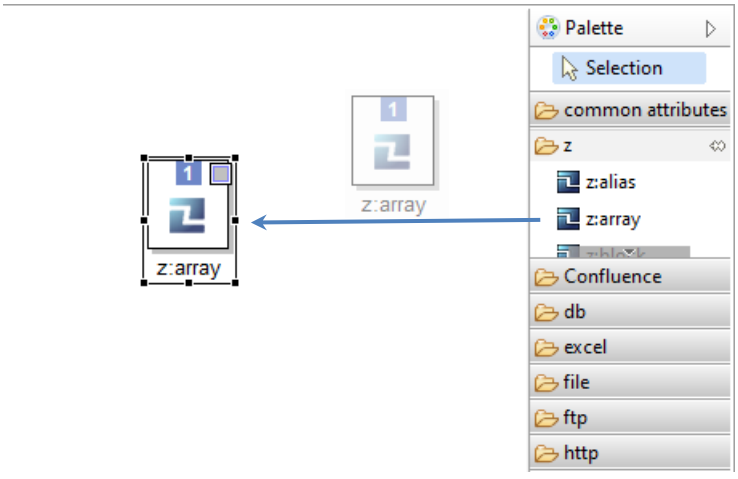


Figure 10 – Drag-and-dropping an action from the palette

13.2.2.2 Copy

Several ways are available for copying actions. When the user presses the hotkey *Ctrl+C*, and then the *Ctrl+V* combination, a new copy of the action is created in the top left corner of the canvas.

The second alternative for copying actions is the *Ctrl + drag* method: while holding down the Ctrl key, the user has to drag an action to another position on the canvas.

The third approach is right-clicking on the action and choosing the *Copy* item from the context menu (see *Figure 11.*). Then, by choosing the *Paste element* menu item from the context menu of the canvas (clicking on an empty area on the canvas, see *Figure 12.*), the copy of the action is created at the mouse position.

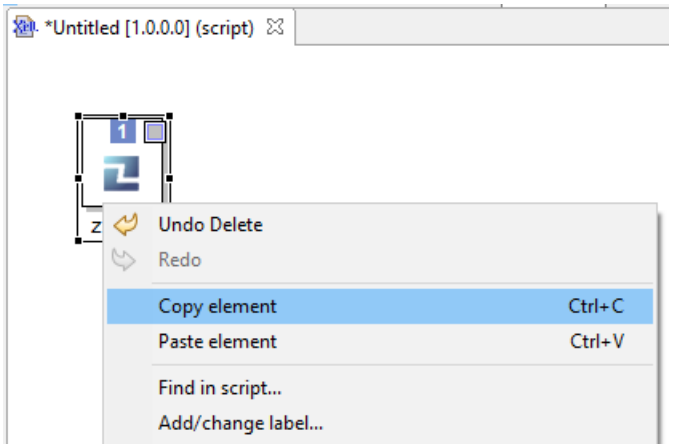


Figure 11 – Step 1: selecting the *Copy element* menu item from the action context menu

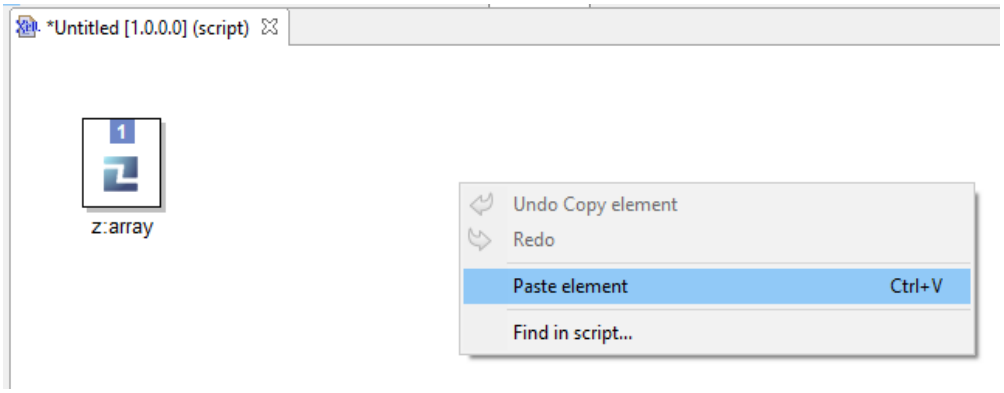


Figure 12 – Step 2: right-clicking on an empty space and selecting the *Paste element* menu item from the context menu

Altough It is possible to copy actions in the XML View from one script to another, but it is not recommended. Any kind of introduced syntax or semantic error prevents the user from switching back to the Graph View or saving the document.

Info: *The first and the third method can also be used between different scripts opened in the Editor Area*.

13.2.2.3 Move

Users can move actions by simply dragging them on the canvas. The simplest moving operation is when the action remains on the same level in the XML hierarcy, thus its action ordering number does not change.

There are two other cases of moving actions: *embedding* and *detaching*.

Embedding an action to a new parent: the user is dragging an action and dropping it inside the area of another action, see *Figure 13*. In this case the moved action becames the last child action of the other action. The action ordering number changes accordingly.

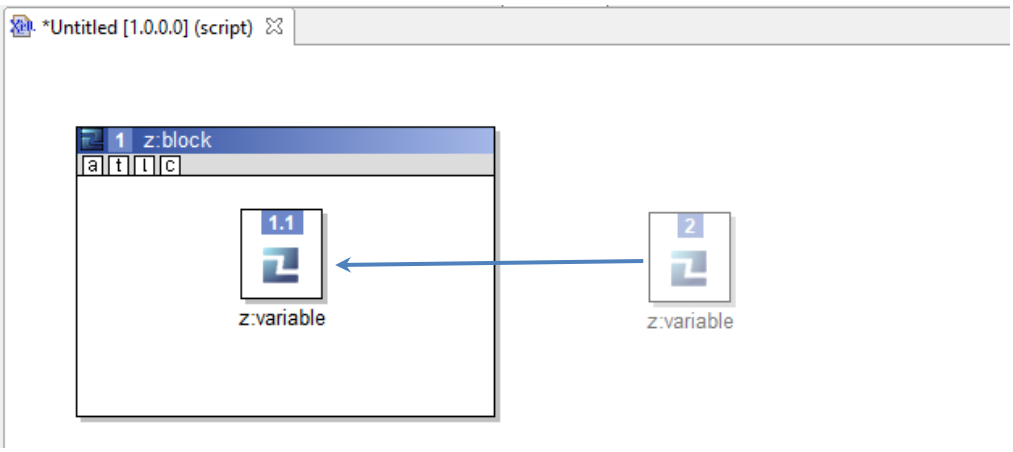


Figure 13 – Embeding Action 2 into Action 1. The ordering number becomes 1.1

Detaching a child action from its parent: the user is dragging the child action out of the area of its parent, see *Figure 14*. In this case the moved action becames the last child action of its new parent action. The action ordering number changes accordingly.

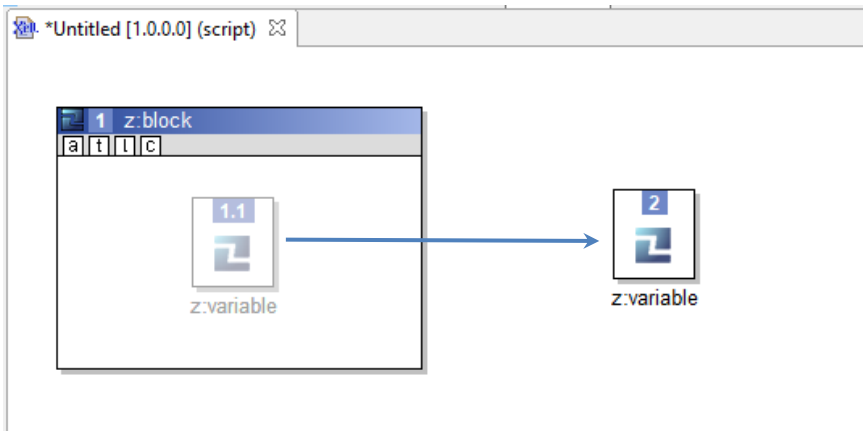


Figure 14 – Detaching Action 1.1 from its parent. The ordering number becomes 2.

13.2.2.4 Delete

There are several ways to delete actions in the Script Editor. When any actions are selected, pressing the *Delete* key deletes the selected actions from the script. Another way is to use the *Delete* menu item from the action context menu. Alternatively, the menu item *Delete* is accessible from the main menu bar, see → Edit menu.

13.2.2.5 Changing the order of execution

The action ordering number indicates the order of execution of the script, see → Order of execution, result flow. The graphical arrangement of the actions does not affect the order of execution, but it is highly recommended to arrange the actions in a way that represents the actual processing order.

The ordering number of an action can freely be changed among its siblings (i.e. only among actions with the same parent action), which operation does not change the position of the actions on the canvas. When an ordering number is increased or decreased, the ordering number of the adjacent action changes accordingly. This operation practically exchanges the ordering number of two adjacent actions.

There are several ways to perform this operation:

- by pressing the *PageUp* and *PageDown* keys
- by the *Move up* / *Move down* menu items in the action context menu

In *Figure 15.*, the actions are organized well visibly, but their ordering numbers are messed up, so script execution would result in error, e.g. the action ordering number of the `db:jdbc-connection` action should precede all other db-related actions.

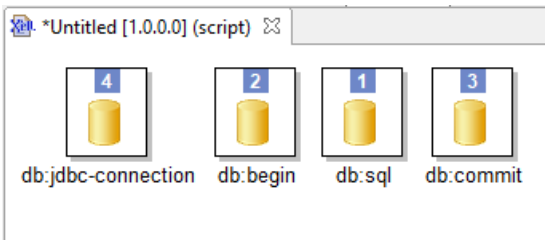


Figure 15 – Ordering numbers are messed up on the execution level

In *Figure 16.*, the same actions are ordered properly for exection.

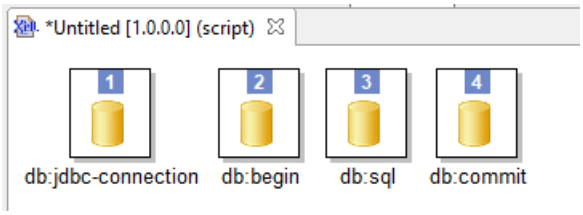


Figure 16 – Ordering numbers are set correctly

13.2.2.6 Comment / uncomment

The user can comment any action in order to omit it from the execution. (It is similar to temporarily commenting code lines in programming languages.) When an action is commented, its ordering number remains unchanged, but it is skipped from the execution flow. Commented status is indicated by grey color. The hotkey for commenting / uncommenting an action is `Ctrl+Shift+C`. The same operation can be accessible from the context menu of the action by selecting the *Comment / Uncomment* menu item.

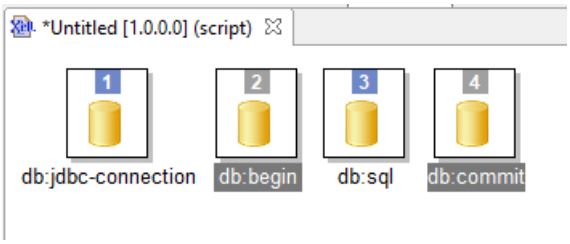


Figure 17 – `Action 2` and `Action 4` are commented, so only `Action 1` and `Action 3` will be executed

13.2.2.7 Selecting multiple actions

It is possible to select and manipulate multiple actions at the same time.

Figure 18 – `Action 1` and `Action 3` are selected

There are several ways to select multiple actions:

- *Using the Rectangle selection tool*
  Clicking on the canvas with the left mouse button, and by dragging the mouse pointer, a selection rectangle will be drawn. All actions inside the rectangle will be selected.

- *Using the Ctrl+left mouse click combination*
  Actions can be added / removed from the existing selection.

- *Using the Shift+left mouse click combination*
  Actions can be added to the existing selection.

- *Using the Shift + arrows key combination*
  Left clicking on the first action to be selected, then holding the *Shift* key and pressing an *arrow* key of the desired direction at the same time.

The following operations can be performed for multiple selected actions:

- Copy
- Move
- Delete
- Comment / uncomment

13.2.3 Editing

In this section, operations related to various action properties are discussed. The properties of an action are described in details in → Action attributes and → Action content.

13.2.3.1 Attributes

Action attributes can be edited in two different ways:

- by using inline attribute editing
- by opening the *Attributes* dialog

Inline editing can be performed by double clicking on the name of the attribute of the action, see *Figure 19*. In this case, an input text field appears. After typing the attribute value, it can be saved by pressing the *Enter* key, or cancelled by pressing the *Esc* key (in this case, the original value will be restored).

Inline editing is only available when the action is either in the *full* or in the *only filled* view modes, see → The full view mode and → Only filled view mode.

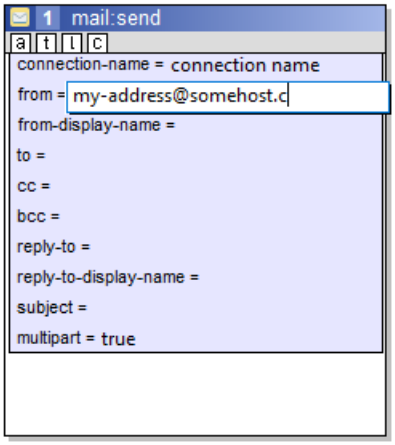


Figure 19 – Inline editing of an attribute value

The *Attributes* dialog provides a more convenient way to manage and edit attributes. It can be opened in the following ways:

- by selecting the *Edit attributes...* menu item from the action context menu
- by pressing *Ctrl + A*
- by clicking on the 'a' toolbar icon on the action toolbar

The *Attributes* dialog lists all the attributes of the action. Here, the user can create new attributes, edit the values of existing ones, or delete attribute name-value pairs.

- For creating a new attribute, the user needs to click on the subsequent empty table line and enter the attribute name-value pair in the corresponding columns.

- For editing an existing attribute, The user needs to click on the given attribute value and enter the new value, see *Figure 20*.

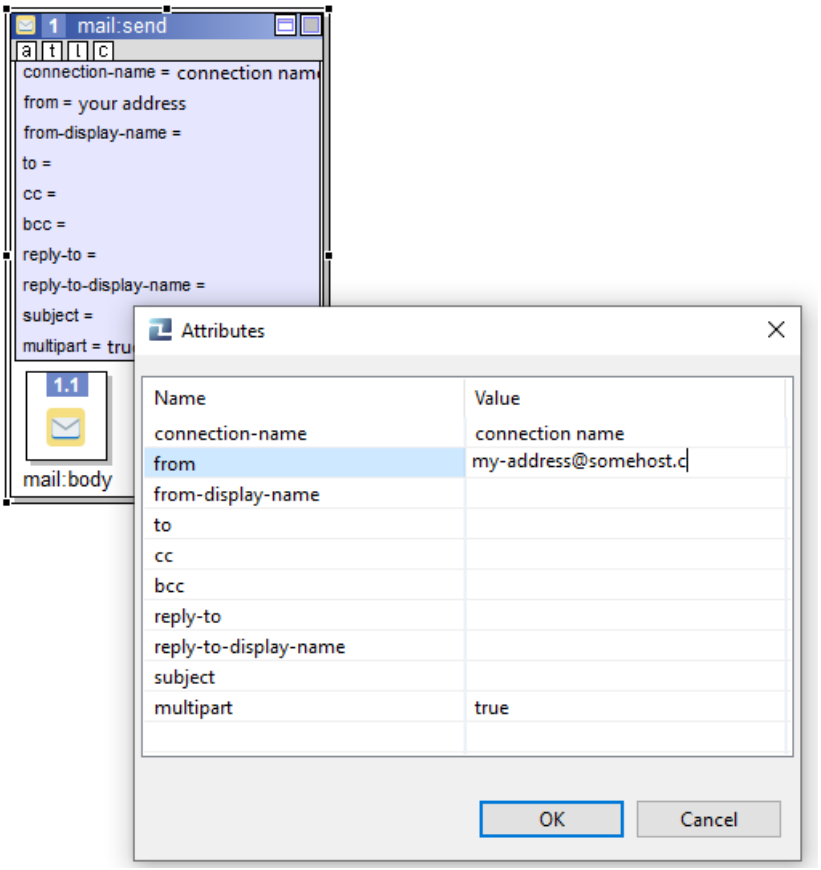


Figure 20 – Editing the value of an existing attribute

- For deleting an attribute, the user needs to click on the given attribute name and erase the it.

The changes take place after pressing the OK button.

Some attributes values can be selected from a pre-defined list, see → Predefined and common attributes. In those cases, the value can be chosen from the pre-defined values via a dropdown list, see *Figure 21*.

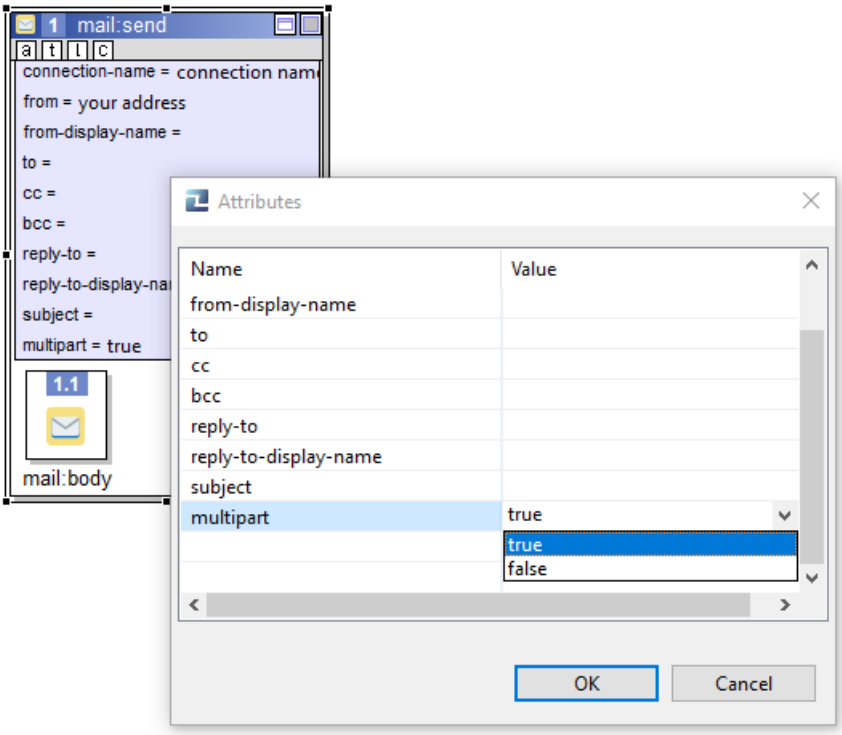


Figure 21 – Selecting the attribute value from a dropdown list

13.2.3.2 Text content

This property sets the textual content of an action, see → Textual content. The textual content can be edited in two different ways:

- by inline text editing
- by opening the *Set text* dialog

Inline editing can be performed by double clicking on the textual value of the action, see *Figure 22*. In this case, an input text field appears. After typing the new value, it can be saved by the *Ctrl+Enter* key combination or by clicking outside of the textbox, or cancelled by pressing the *Esc* key (in this case, the original value will be restored).

Inline editing is only available when the action is in an *open* view mode, see → View modes.

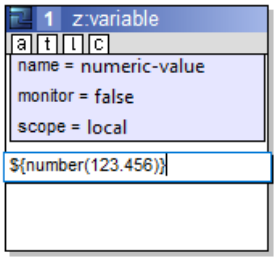


Figure 22 – Inline editing of the text content

The *Set text* dialog provides a more convenient way to edit the text content. It can be opened in the following ways:

- by selecting the *Add/change text in dialog…* menu item from the action context menu
- by pressing *Ctrl + T*
- by clicking on the 't' toolbar icon on the action toolbar

The *Set text* dialog shows the textual content of the action in a multi-line text editor, see *Figure 23*.

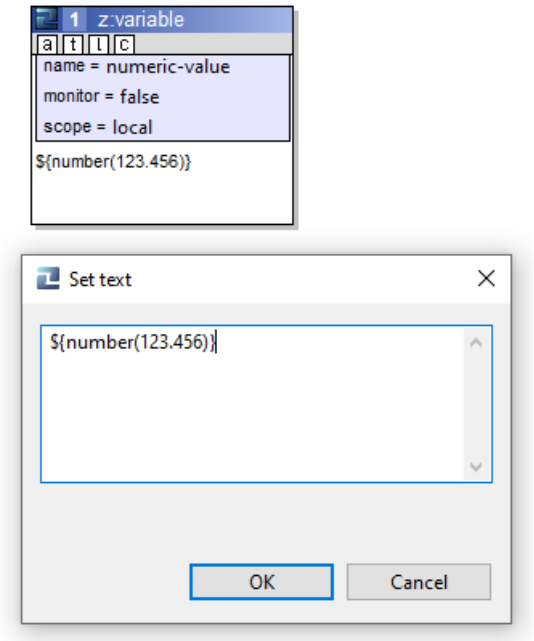


Figure 23 – Editing the text content in the *Set text* dialog

13.2.3.3 Label

An action can be labeled in order to provide short useful information about its behaviour. This is recommended for parent actions that bunch child actions that are doing a particular functionality in the script (e.g. connecting to a database and processing some results might be labeled as '*db query*'). A parent action can be a `z:block` action for this goal. Action labels are shown in the place and instead of the action name.

Action labels can be edited in the following ways:

- by selecting the *Add/change label…* menu item from the action context menu
- by clicking on the 'l' toolbar icon on the action toolbar

After the *Set label* dialog box appears, the user can specify the label, see *Figure 24.* The maximum length of the label is 256 characters, and line breaks cannot be used.

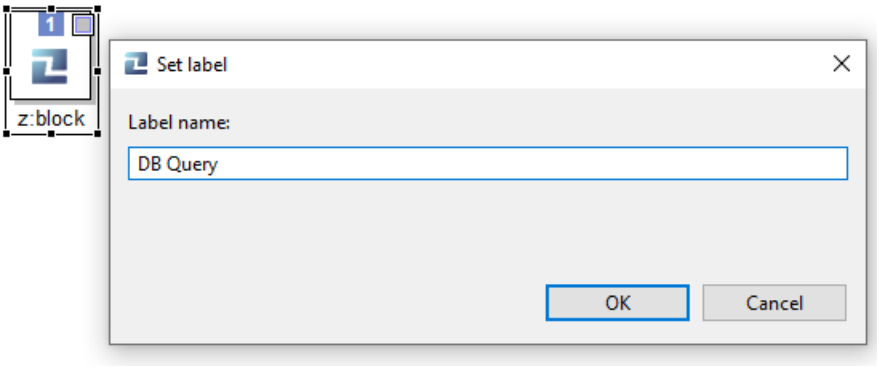


Figure 24 – Setting an action label via the *Set label* dialog box

After pressing the *Ok* button, the action label will be shown at the place of the original action name, see *Figure 25.*

Figure 25 – The action label is displayed instead of the action name after a label has been added

A label can be updated in the same ways as creating one. Deleting an action label is possible by setting an empty action label.

13.2.3.4 Comment

An action comment can be specified in order to provide short, useful information about its behaviour. Unlike the action label, the action comment appears as a tooltip when the mouse cursor hovers over the action.

Action comments can be edited in the following ways:

- by selecting the *Add/change comment…* menu item from the action context menu
- by clicking on the 'c' toolbar icon on the action toolbar

After the *Set comment* dialog box appears, the user can specify the comment, see *Figure 26.* The maximum length of the comment is 256 characters.

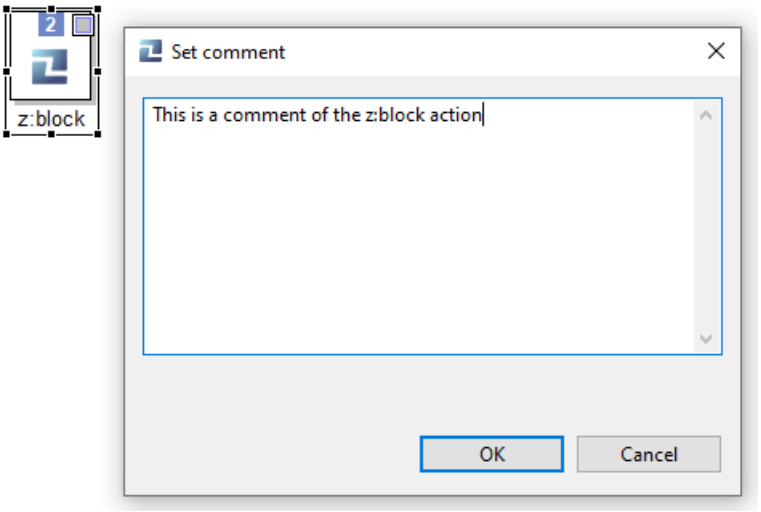


Figure 26 – The *Set comment* dialog box

After pressing the *Ok* button, the action comment will be shown as a tooltip when the mouse pointer is hovering over the action, see *Figure 27.*

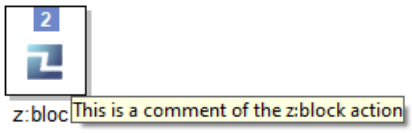


Figure 27 – The new comment appears as a tooltip

An action comment can be updated in the same ways as creating one. Deleting an action comment is possible by setting an empty comment.

⚠ ***Warning:*** *The* comment *action property can be confused with the commenting / uncommenting action operation. They are not the same!*

13.3 Action help

Each action has its own help page, where all action details are described and examples are listed. It is accessible from the palette (see → Palette) by right-clicking on the action name and clicking on the *help* menu item, see *Figure 28.*

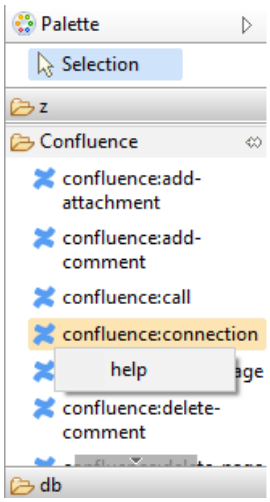


Figure 28 – Accessing connection help from the palette

An alternative is to right-click on an already existing action action in the Script Editor, and selecting the *Show help...* option in the context menu, see *Figure 29*.

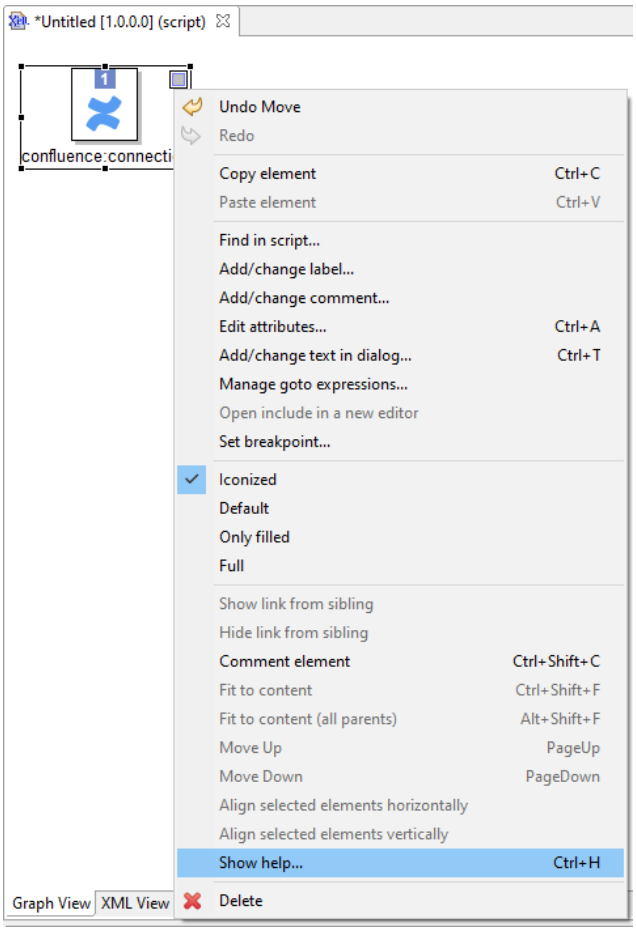


Figure 29 – Accessing connection help from the Script Editor

Either way, the *Documentation* window will be shown, see *Figure 30*.

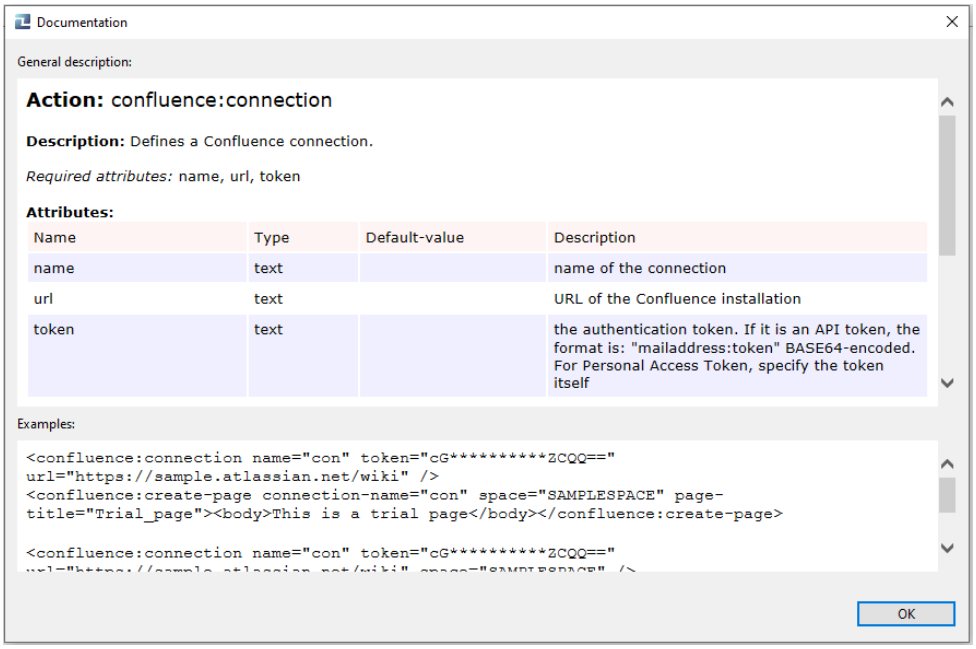


Figure 30 – The *Documentation* window of the `confluence:connection` action with sample configurations

13.4 Formatting

There are several operations related to the formatting of the actions. Some of them affect actions on the same level (actions with the same parent), while some of them change the size of the parent action according to its children positions.

13.4.1 Alignment operations

Multiple actions can be aligned both horizontally and vertically. Align options can be found in the context menu of multiple action selection (after selecting multiple actions, the user needs to right click on one of them).

13.4.1.1 Align selected elements horizontally

By choosing the *Align selected elements horizontally* menu item from the context menu, the selected actions will be aligned horizontally. For example, the actions shown in *Figure 31.* will be aligned as shown in *Figure 32.*

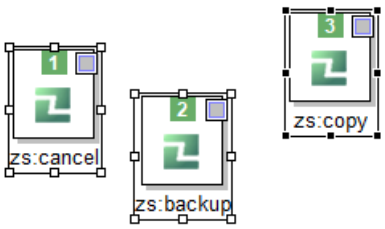


Figure 31 – Selecting multiple actions on the same level

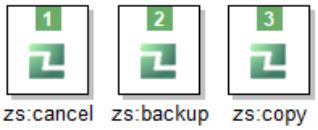


Figure 32 – Actions aligned horizontally

Note that the order of selecting multiple actions determines the alignment result. The action selected first will be the first action in the result, the subsequently selected action will be the second one, and the others respectively.

13.4.1.2 Align selected elements vertically

By choosing the *Align selected elements vertically* menu item from the context menu, the selected actions will be aligned vertically. For example, the actions shown in *Figure 31.* will be aligned as shown in *Figure 33.*

Figure 33 – Actions aligned vertically

Note that the order of selecting multiple actions determines the alignment result. The action selected first will be the top-most action in the result, the subsequently selected action will be the second one, and the others respectively.

13.4.2 Size operations

Action size operations affect the size of the particular action related to its content.

13.4.2.1 Fit to content

The *Fit to content* operation changes the action size up to the minimum size that the child actions take up. It makes sense only in open view modes, see → View modes.

The *Fit to content* operation can be performed in the following ways:

- by selecting the *Fit to content* menu item from the action context menu
- by pressing *Ctrl + Shift + F*

For example, the size of `Action 1` shown in *Figure 34.* will be changed as shown in *Figure 35.*

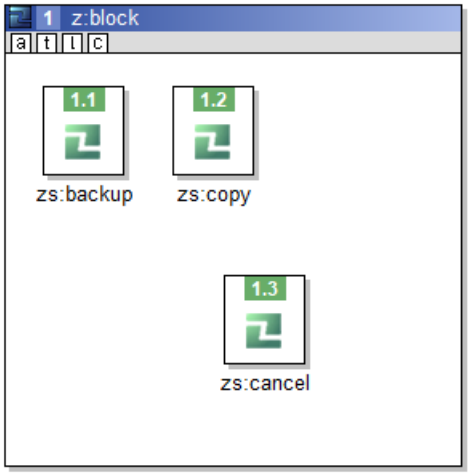


Figure 34 – Before performing a *Fit to content* operation on `Action 1`

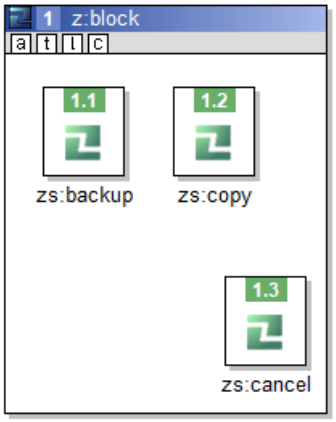


Figure 35 – After performing a *Fit to content* operation on `Action 1`

> ***Info:*** *When an action view mode is changed from a closed to an open view mode, the* Fit to content *operation is automatically performed.*

13.4.2.2 Fit to content (all parents)

This operation performs the same as the *Fit to content* operation (see above). The only difference is that it recursively resizes all the parents of the original parent action in the same manner, so the fitting the content is propagated up to the `z:root` action (the canvas level).

The *Fit to content (all parents)* operation can be performed in the following ways:

- by selecting the *Fit to content (all parents)* menu item from the action context menu
- by pressing *Alt + Shift + F*

For example, performing this operation on `Action 1.4` in *Figure 36.* will result in an arrangement shown in *Figure 37.*

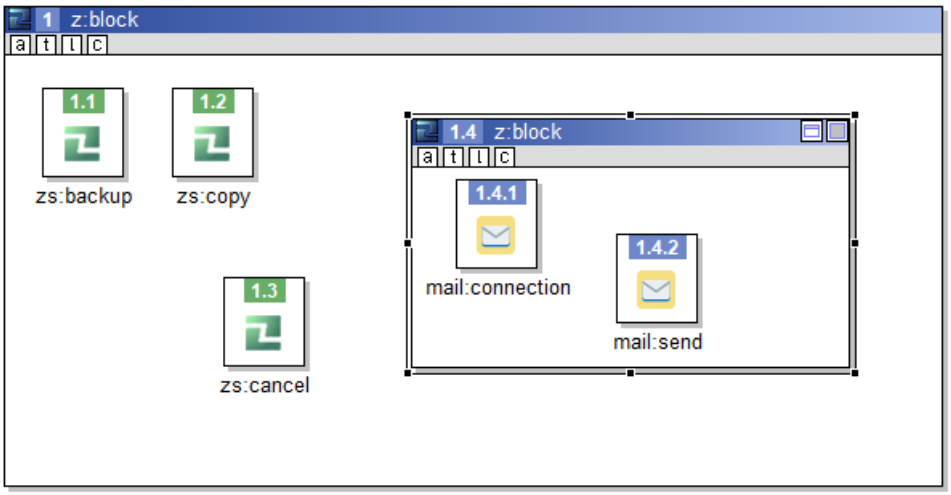


Figure 36 – Before performing a *Fit to content (all parents)* operation on `Action 1.4`

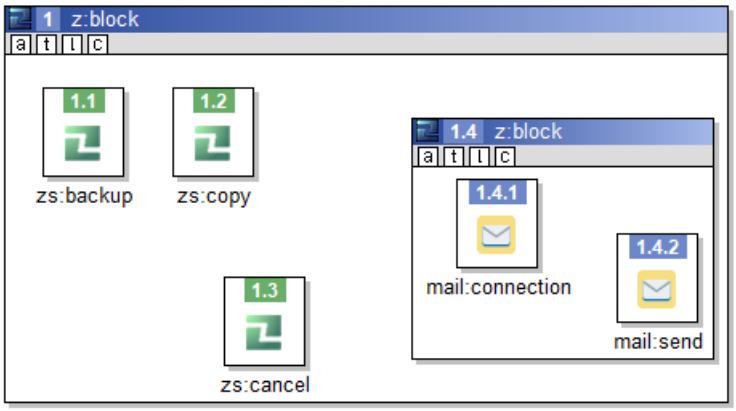


Figure 37 – After performing a *Fit to content (all parents)* operation on `Action 1.4`

13.5 Additional displaying options

There are special cases when action opacity or the relations of actions are displayed in a different way than usual. In this section, these displaying options are discussed in details.

13.5.1 Outside displaying option for a child action

By default, the parent-child action relationship is displayed as the child action being embedded into the content of the parent action. In some cases, switching between an open and a closed view mode (see → View modes) of a large child action inside the content area of the parent action is not convenient, see *Figure 38*. In this case, the alternative display mode of embedment can be useful, see *Figure 39*.

The child action is displayed outside of the parent action, while a small connector node indicates its original place in the content area of the parent action. These two are connected by a black arrow pointing from the child action to the connector node.

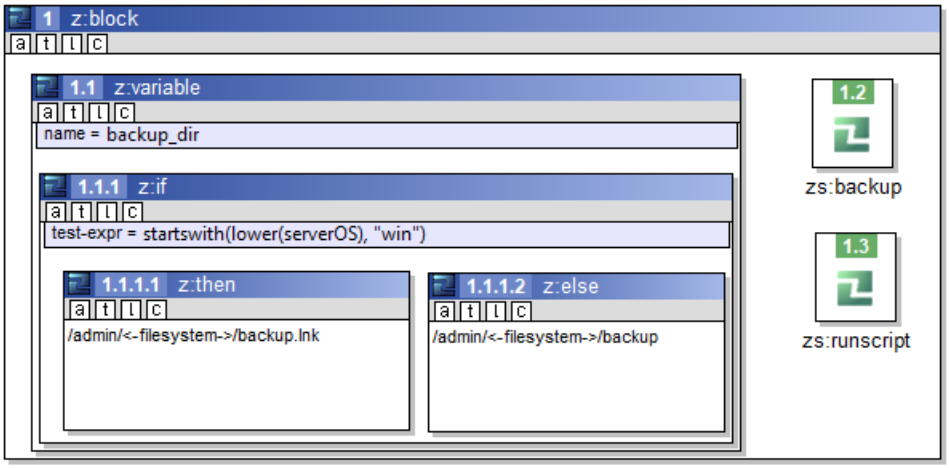


Figure 38 – A parent action having a large child action

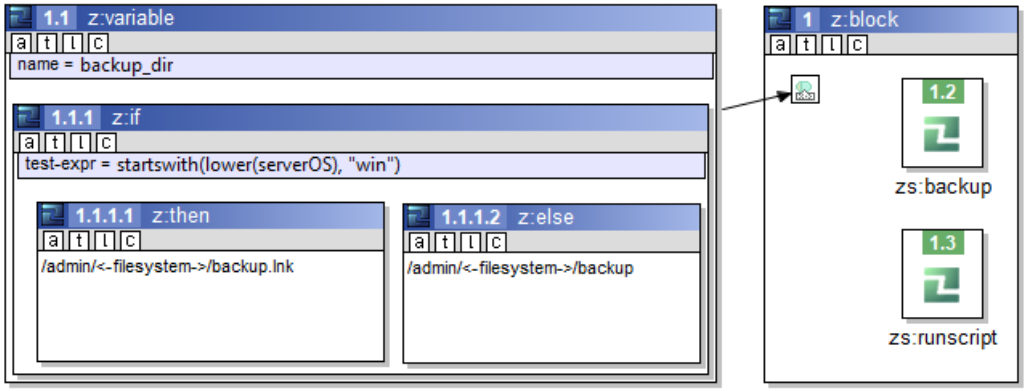


Figure 39 – `Action 1.1` is now displayed outside of `Action 1`

To achieve this alternative displaying option, the user needs to do the following steps:

1) select an action
2) right-click on the *embedment handle* at the top-right corner of the main selection frame (see *Figure 40.*)
3) drag the handle with the mouse pointer to an empty area of the canvas

Figure 40 – The embedment handle of the selected action

13.5.2 Showing sibling links

There is an option to display execution order between two sibling actions for visually emphasizing the execution flow, see → Order of execution, result flow. If the user select the *Show link from sibling* menu item from the action context menu, a red arrow is shown from the previous sibling action to the selected action, see *Figure 41*.

Figure 41 – Sibling link is shown between two subsequent actions

To hide a siblink link arrow, the user has to select the *Hide link from sibling* menu item from the action (the one where the arrow is pointing to) context menu.

There is an option for the behaviour of the arrows in case of reordering affected actions, see → Changing the order of execution.

13.5.3 Opacity

There is an option to set the opacity of overlapping actions. The opacity value can be set from 0 to 100, 0 meaning full transparency and 100 meaning full opacity. By default, the opacity value is set to 70.

Figure 42 – Opacity value is set to 50 (semi-transparent)

13.5.4 Displaying goto expressions

By using goto expressions, the user can change the order of execution (see → Order of execution, result flow). Goto expressions can be displayed in the Script Editor.

Next, adding goto expressions will be demonstrated through an example. The starting state of the sample script is shown in *Figure 43*.

Figure 43 – The script before adding goto expressions

One important condition to create a goto expression is having a *label* attribute at the target action. In *Figure 43.*, `Action 5` and `Action 6` will be used as target actions, therefore they have their *label* attributes set already.

By right-clicking on `Action 2` and selecting the *Manage goto expressions…* menu item from the action context menu, the *Manage goto expressions* dialog box appears. Clicking on the *Add…* button opens a new input dialog box (*Goto parameters*), see *Figure 44.* In this dialog box, the user has to specify the goto condition and select the target label of the goto expression to be created. Notice that only labels belonging to the same sibling level as the selected action are listed!

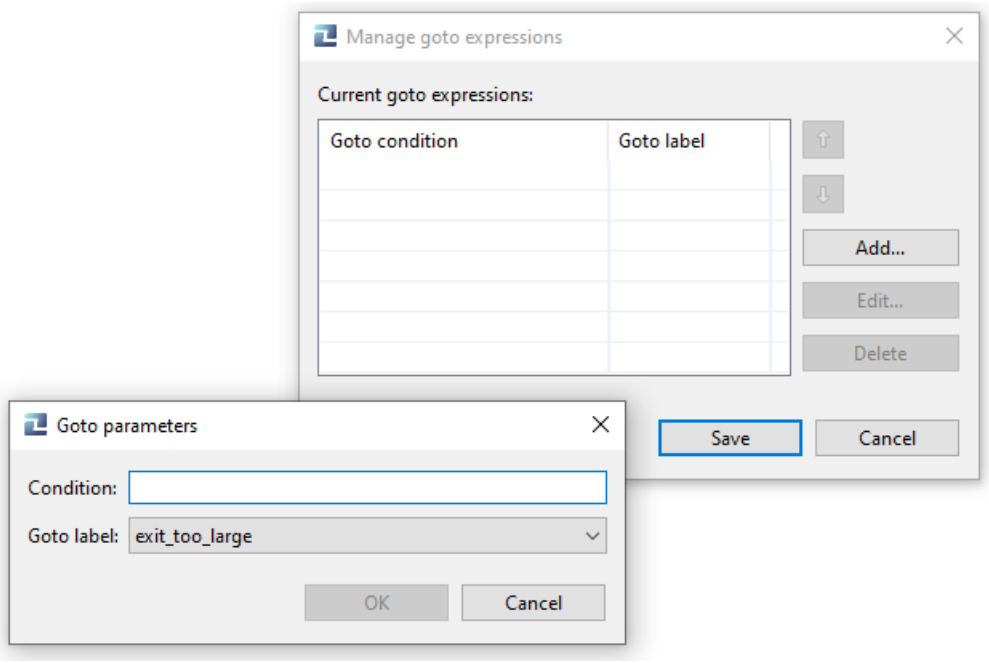


Figure 44 – Specifying a goto expression

Multiple goto expressions can be added for the same action, see *Figure 45.*

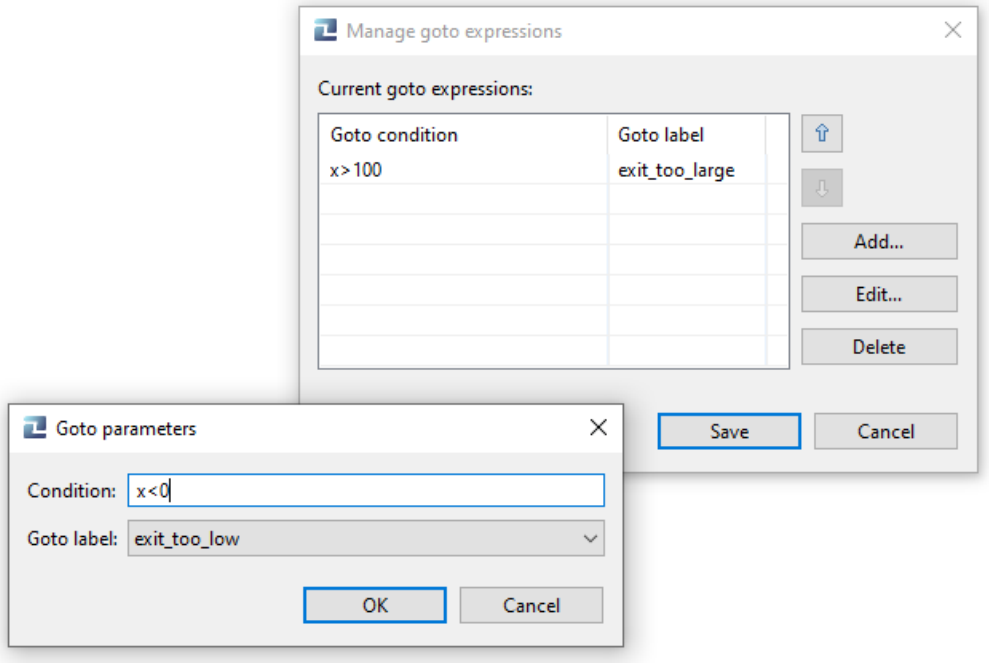


Figure 45 – Specifying the second goto expression

All the current goto expressions are displayed in the *Current goto expressions* table, see *Figure 46*. Already specified goto expressions can be edited or deleted by using the *Edit…* and *Delete* buttons, respectively, while they can be reordered by using the up and down arrows.

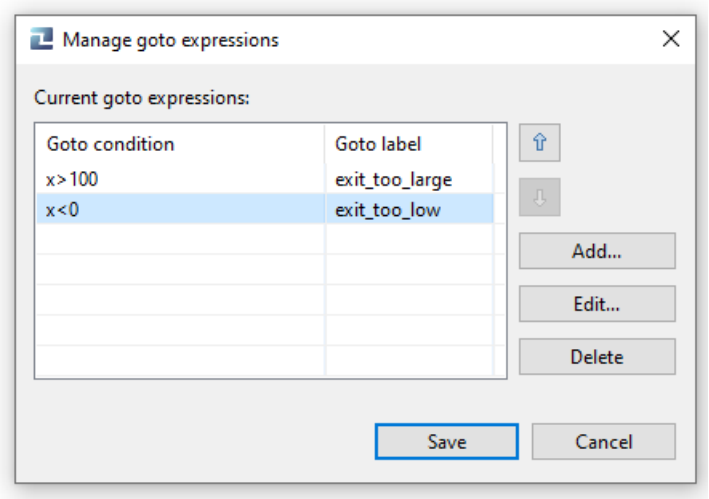


Figure 46 – Showing the list of goto expressions

After pressing the *Save* button, magenta colored arrows will be shown and indicate the direction of the newly created goto expressions, see *Figure 47*.

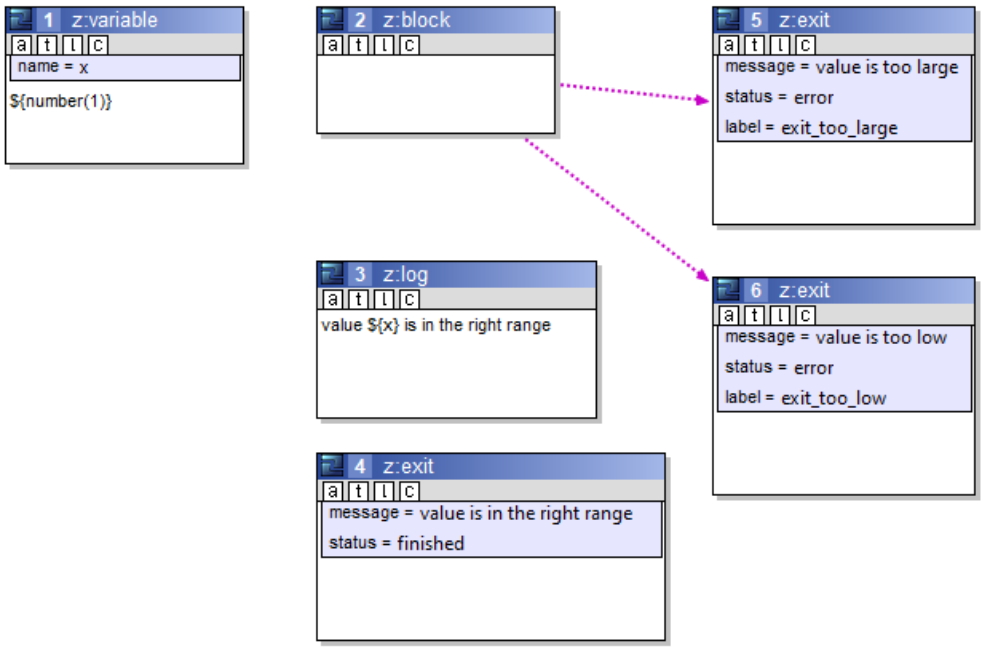


Figure 47 – The script after adding the goto expressions

For displaying the most comprehensive visual aid for the execution order of actions, sibling links (see → Showing sibling links) might be switched on for specific actions, see *Figure 48*.

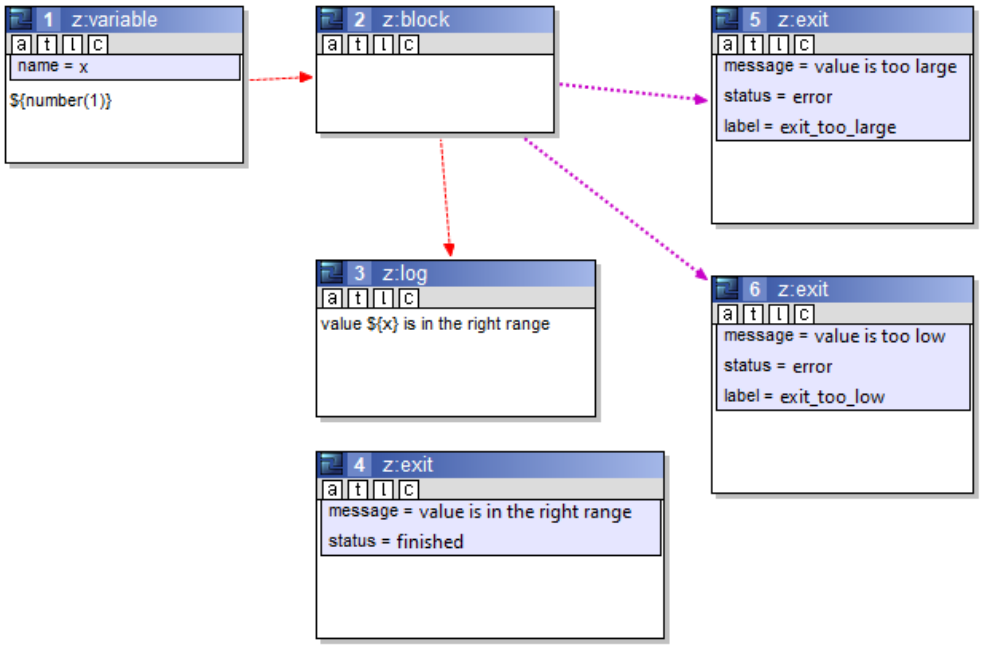


Figure 48 – The script showing the sibling links as well

13.5.5 Attribute as child element

There might be cases when a value of an attribute is too complex for a simple textual attribute value. There is an option for loading an attribute value as a child element of the parent action. *Figure 49.* shows a `z:foreach` action with an *in* attribute filled with a list of values.

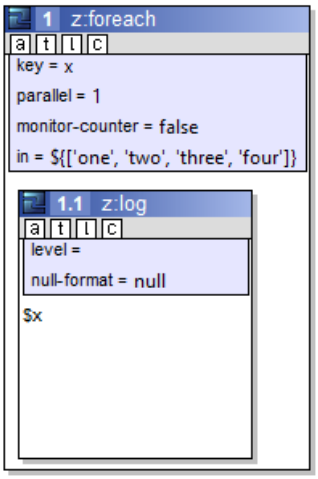


Figure 49 – An example for a `z:foreach` action with *in* attribute

If the user wants to use a lot of items in the list, reading them dynamically (e.g. from a file) could be a good solution. However, because of the fact that an attribute value can only be specified as a simple text value, it is not possible. One solution would be to use a `file:read` action with an *alias* attribute, and refer to the alias in the *in* attribute

of the `z:foreach` action. A more convenient way to do the same is to use the *View attribute as child element* menu item from the attribute context menu, see *Figure 50.*

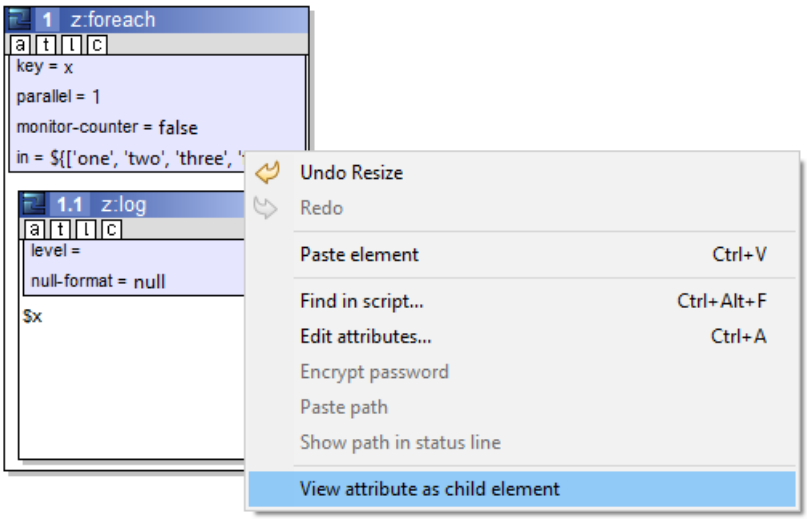


Figure 50 – The *View attribute as child element* menu item for the *in* attribute

This operation converts the *in* attribute into a `z:in` child action. As seen in *Figure 51.*, the original attribute value became the textual content of the child action.

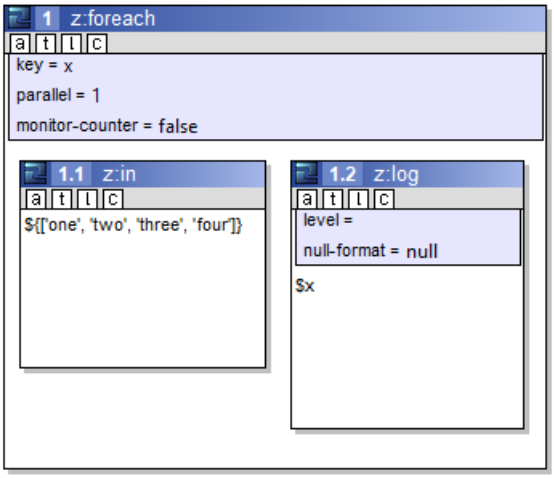


Figure 51 – The resulting `z:in` child action

The user now needs to simply delete the textual content and replace it with an action with a more complex output, e.g. a `file:read` and a `z:parse` action, see *Figure 52.*

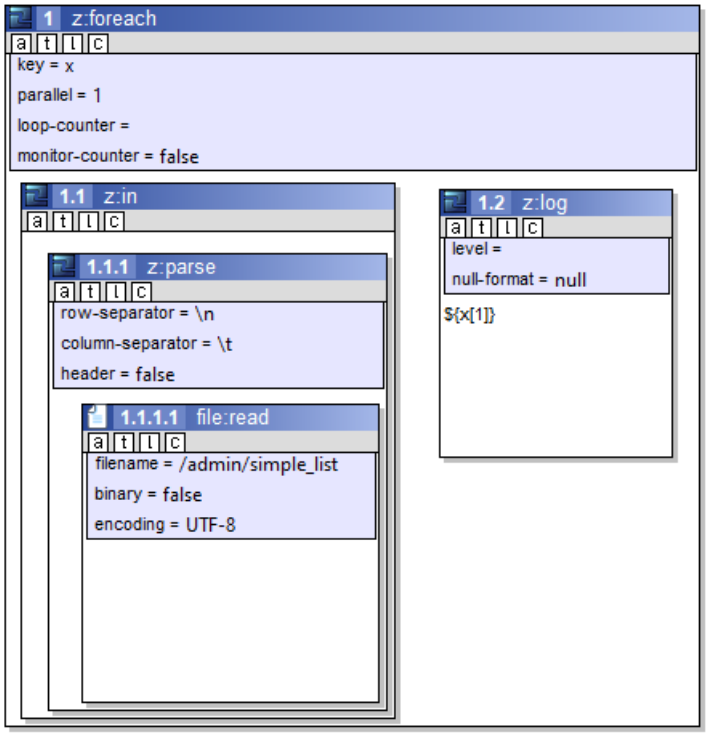


Figure 52 – An example for a more complex `z:foreach` action with `z:in` child element

The `z:foreach` action now iterates over the lines of the content of the `/admin/simple_list` file.

Note that the child actions generated this way can be converted back into attributes by selecting the *View element as attribute* menu item from the action context menu. However, using this is not recommended if the child action contains anything else than textual content, as only textual content can be copied back into the attribute value.

13.6 Special operations

13.6.1 Find in script

The user can search in the content of the script by selecting the *Find in script…* menu item from the action context menu. It opens the *Search in script content* dialog box, see *Figure 53.* The search will be applied for the selected action and all of its child actions recursively; the header of the dialog box also indicates the scope of the search.

If the user wants to search in the whole script, right-clicking on the canvas and selecting the same menu item from the context menu should be performed.

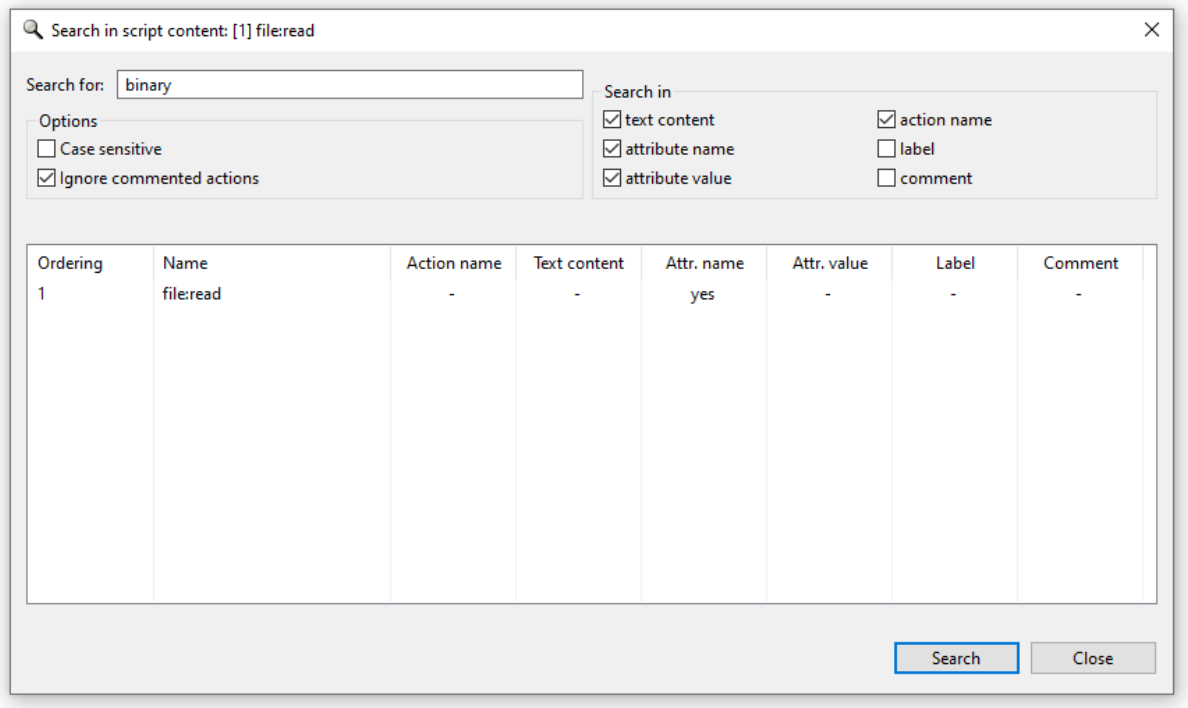


Figure 53 – The *Search in script content* dialog

The user needs to specify the search string in the *Search for* textbox.

In the *Search in* area, the scope of the search can be specified. The following checkboxes specify the parts of the script in which the search string should be looked after:

- *text content:* in the text content of the action, see → Textual content
- *attribute name:* in the name of the attributes, see → Action attributes
- *attribute value:* in the value of the attributes, see → Action attributes
- *action name:* in the fully-qualified name of the actions, see → Action groups and action name

- *label:* in the label of the actions, see → Label
- *comment:* in the comment of the actions, see → Comment

In the *Options* area, two search options can be specified:

- *Case sensitive:* when checked, a case sensitive search will be performed.
- *Ignore commented actions:* when checked, the commented actions (see → Comment / uncomment) are excluded from the search.

By pressing the *Search* button, the search results are shown in the table in the lower half of the dialog box. The table shows the following columns:

- *Ordering:* the ordering number of the action in which the search string was found, see → Ordering numbers
- *Name:* the fully-qualified name of the action in which the search string was found, see → Action groups and action name
- *Action name:* it indicates whether the search string was found in the action name
- *Text content:* it indicates whether the search string was found in the text content
- *Attr. name:* it indicates whether the search string was found in the attribute name
- *Attr. value:* it indicates whether the search string was found in the attribute value
- *Label:* it indicates whether the search string was found in the label
- *Comment:* it indicates whether the search string was found in the comment

13.6.2 Set breakpoint

Setting breakpoints for debugging can be useful before starting a debug session (see → Starting a debug session) in the Script Editor. This operation can be accessed by selecting the *Set breakpoint…* menu item from the action context menu, which opens the *Set breakpoint* dialog, see → Breakpoints window.

This way, the user does not need to set the breakpoints after each debug session start, but the breakpoints are already defined and saved into the Zagreus script.

The created breakpoints are displayed in the *Breakpoints* extension window, see → Breakpoints window. Just like during a debug session, the user can edit and delete the defined breakpoints in this window.

13.6.3 Encrypt password

The user can directly encrypt the value of a *password* attribute. This encryption is the same as achievable in the *Password encryption* dialog box, see → Tools menu. The *Encrypt password* menu item is accessible in the attribute context menu only for the *password* attribute, see *Figure 54.*, for all other attributes it is greyed out.

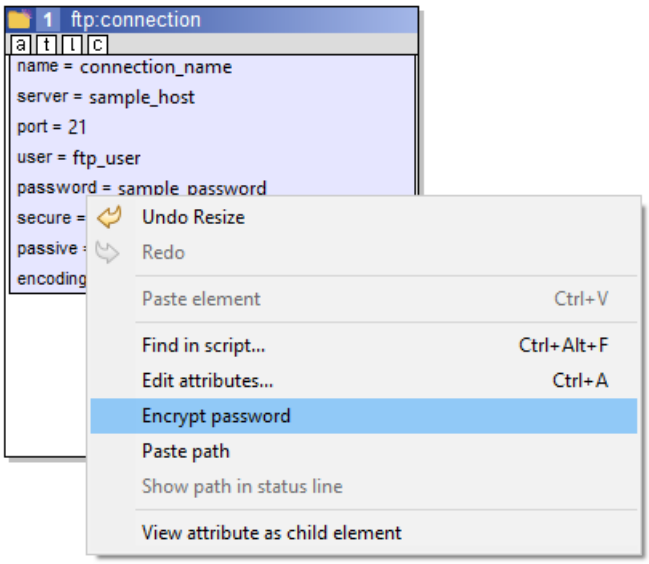


Figure 54 – The *Encrypt password* menu item in the attribute context menu

After the operation is performed, the encrypted password will appear in the value of the *cpassword* attribute, see *Figure 55.* The original *password* attribute will be empty.

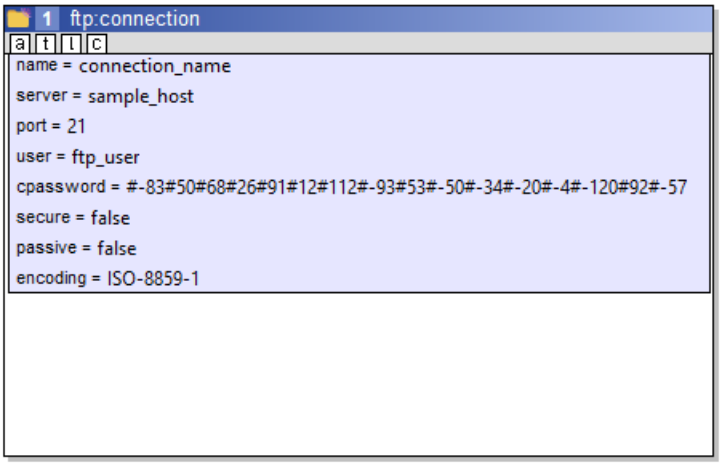


Figure 55 – The *cpassword* attribute holds the encrypted password value

13.6.4 Paste path

Once the user selected the *Copy path* operation in the Zagreus Browser window (see → Copying the resource path), the path copied to the clipboard can be pasted to an attribute value. For this, the user has to select the *Paste path* menu item in the attribute context menu, see *Figure 56.*

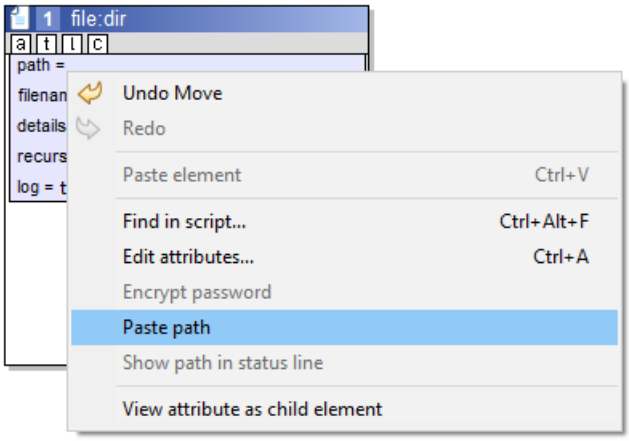


Figure 56 – The *Paste path* menu item in the attribute context menu

13.6.5 Show path in status line

There is a special feature for the attributes *filename* and *id*. When they contain the ID of a resource, the *Show path in status line* menu item is accessible in the attribute context menu, see *Figure 57.* Using this feature will show the full path of the resource in the status line of the Zagreus Client, see → Zagreus Client.

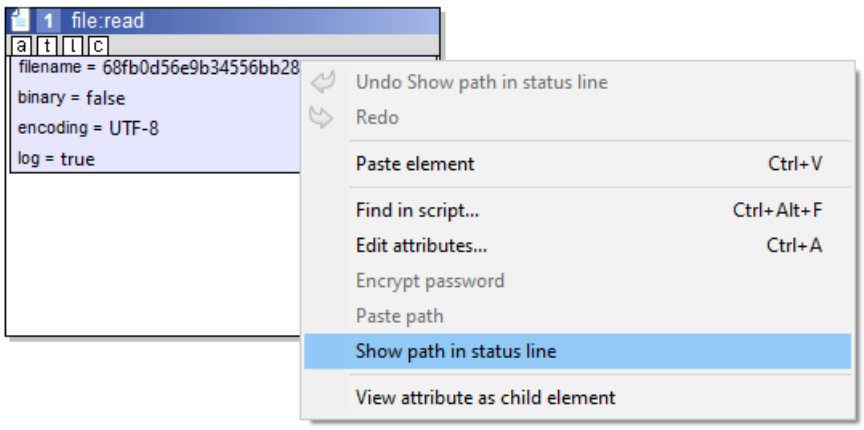


Figure 57 – The *Show path in status line* menu item in the attribute context menu

13.7 Configuration options

Various settings for the Script Editor can be changed in the *Options* dialog box (see → Options dialog), which can be opened by selecting the *Options…* menu item in the *Tools* menu in the main toolbar. Two tabs on the dialog contain settings related to the Script Editor: the *Script Editor* and the *Palette* tabs.

14. Debugging in the Zagreus Client

Just like any fully-featured programming language framework, Zagreus offers debugging functionality for script execution. For using this functionality, the user needs to start the script in *debug mode*, and open it in the *Debug Editor* in the Zagreus Client.

14.1 Features

The Debug Editor of the Zagreus Client provides various features for the user:

- *Controlling script execution*
  Controlling basic script execution by the *Resume*, *Step* and *Stop* tools

- *Monitoring actual state of script execution*
  Monitoring the actual point where the execution is suspended

- *Handling breakpoints*
  Specifying breakpoints for each action and fine-tune them. Managing the breakpoints is possible in the *Breakpoints* window

- *Watching script variables*
  Monitoring the actual value of script variables in the *Watch* window

14.2 Debugging concepts and terms

Debugging is the process in which the user can identify and fix problems for a script. During a debugging session, the execution of the script is often paused or suspended at so-called breakpoints in order to check variable values and the actual execution flow of the script in real-time.

Some basic concepts of debugging in the Zagreus System are:

- *Debug session*
  A debug session is a special type of script execution, in which the user can control the execution process and closely inspect the actual state of the job. In almost all cases the debug session is suspended at a breakpoint (see below). When the debug session is over, the status of the script is the same as if it was executed in a normal way.
  When a debug session is started, it immediately suspended before the first action – this is the only way the user can open it after initiating an execution in debug mode.

- *Running in debug mode*
  In Zagreus, all scripts can be started in the so-called *debug mode*. This execution type is very similar to the normal script execution (see → Initiating script execution), but the execution of the started job is suspended at the very beginning of the script. This allows the user to open the script in the Debug Editor, which provides controls over the debug session of the started script.
  Running in debug mode puts the job into the queue (see → Queue) just like during a normal initiation of execution, and then when a Zagreus Worker starts processing it, a *debug session* is automaticcally opened for the job.

- *Breakpoints*
  Breakpoints are specific points in the script where script execution is paused. A breakpoint is linked to a specific action. A breakpoint can be fine-tuned by setting its *position in action*. Its possible values are: *Before action processing, After attributes processing, After children actions processing* and *After action processing*.

- *Execution controlling action: Resume*
  From a suspended state of the debug session, the user can continue the execution of the script with the *Resume* tool. The job execution will suspend again at the next breakpoint (or the script will be executed till its end, if there is no any).

- *Execution controlling action: Stop*
  The user can immediately stop the debug session with the *Stop* tool. The status of the job will be *Cancelled* in this case.

- *Execution controlling action: Step to next*
  The *Step to next* tool is a special controlling tool, by which the user continues job execution, and suspends again at the next sibling action.

- *Inspecting or watching variable values*
  By using the *Watch* window, the user can real-time monitor the actual values of the specified watching variables. Checking these values is the main purpose of the debugging session itself, and it offers a truly useful possibility for identifying the problems of the particular script.

14.3 Starting a debug session

As it was mentioned earlier, the script should be executed in *debug mode* in order to start a debug session. The debug session can be considered as an additional layer over the normal script execution, the suspended-type execution and the real-time monitoring of the execution flow and the watching variables being the only differences. After the end of the debug session, the job will be found among finished jobs just as if it had been executed in the normal way.

In order to execute a script in debug mode, the user needs to right-click on the script in the Zagreus Browser and select the *Run in debug mode* menu item from the context menu, see *Figure 1*.

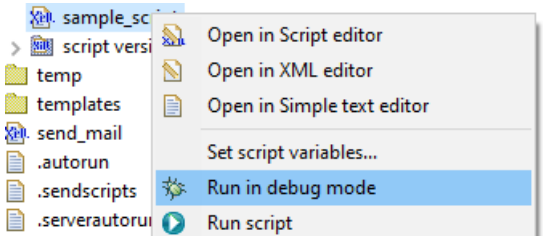


Figure 1 – The *Run in debug mode* menu item in the Zagreus Browser

Once the execution of the newly-created job was started, it is displayed in the Active Jobs window with the job status *Debugging*, see *Figure 2.*

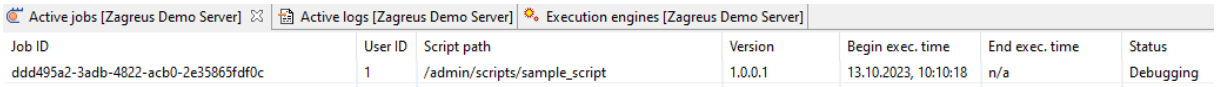


Figure 2 – The job of `sample_script` is in *Debugging* status in the *Active jobs* window

The job is also shown in the the Execution Engines window with the same job status, see *Figure 3*.

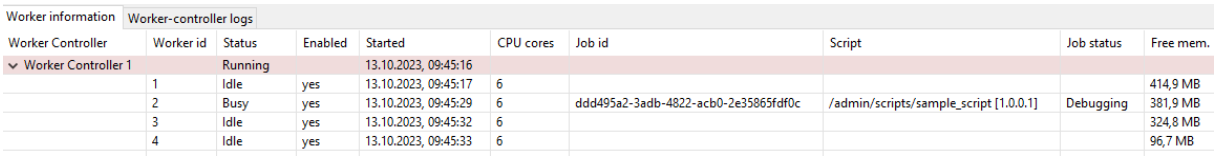


Figure 3 – The job of `sample_script` is in *Debugging* status in the *Worker information* window

The debug session has an implicit breakpoint before the first action, to allow the user to open it in the *Debug Editor* and to control the execution process of the job from the beginning. The job status *Debugging* is a suspended state: the processsing of the job is paused before the execution of the first action, and at all breakpoints in the further execution flow.

At this point, the user can open the job (the debug session) in the Debug Editor by right-clicking on the job in the Active jobs window or in the Execution Engines window, and then selecting the *Open script in debug editor…* menu item from the context menu, see *Figure 4.* and *Figure 5*.

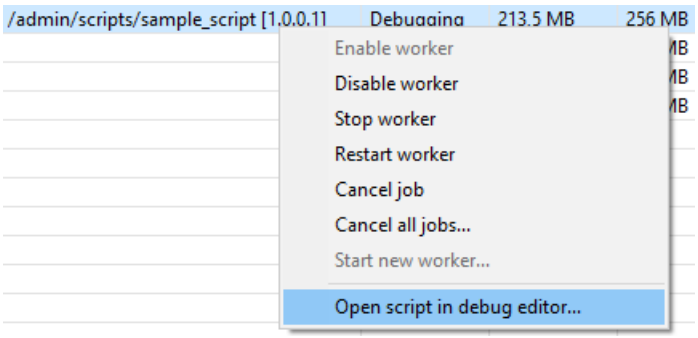


Figure 4 – The *Open script in debug editor…* menu item in the *Active jobs* window

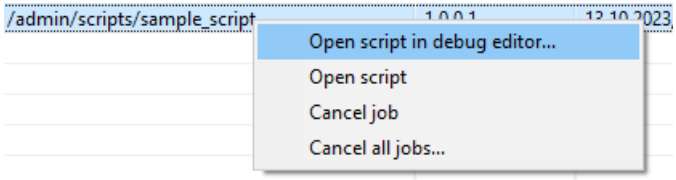


Figure 5 – The *Open script in debug editor…* menu item in the *Worker information* window

The script – the debug session associated with the job – is opened in the Debug Editor, see *Figure 6*. The functionality of this editor is discussed in details in the next chapter.

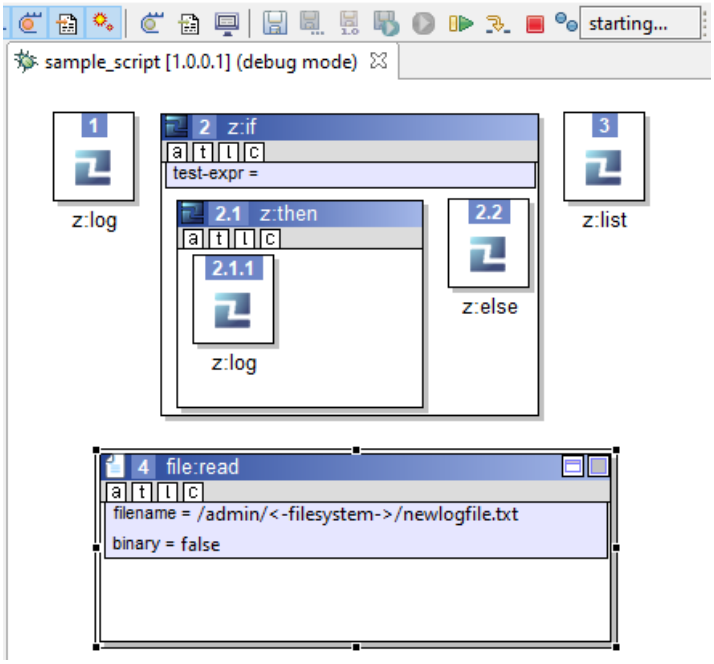


Figure 6 – The script / job opened in the *Debug Editor*

14.4 Debug Editor

The Debug Editor is basically a Script Editor with some further functionality to allow script debugging. This mode does not contain the palette, because the script should not be modified while it is under execution. On the other hand, it provides several additional tools and functions compared to the Script Editor.

One should not forget about the conceptual difference between the two editors: the Script Editor is designed strictly for editing scripts, while the Debug Editor is only for handling a debug session: the Debug Editor is showing the currently running (debugging) job for the particular script. The reason why the Debug Editor looks very similar to the Script Editor is that it is very convenient for the user to see the same format of the script for following the execution flow, monitoring the watch variables and setting the breakpoints.

The Debug Editor is not an 'editor' in the conventional way. It is an interface for a currently running debug session.

14.4.1 Main toolbar

The main toolbar of the Debug Editor is the same as of the Script Editor, but the usual tools (*Save*, *Save as…*, *Save new version*, etc.) are inactive. The following additional tools are, however, available in the main toolbar:

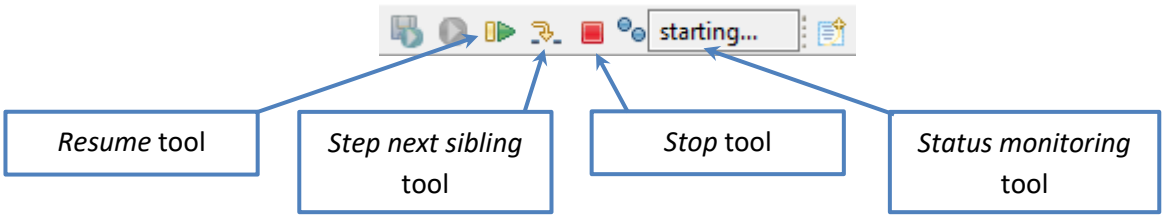


Figure 7 – The tools for the Debug Editor in the main toolbar

- The *Resume* tool:
  By using the *Resume* tool, the job execution can continue from a suspended state until the next breakpoint or the end of the script.

- The *Step to next sibling* tool:
  By using the *Step to next sibling* tool, the job execution continues from the currently suspended state to the next sibling and there it is paused again.

- The *Stop* tool:
  The *Stop* tool immediately stops the debug sesssion. The job will be canceled and no other tool can be used from this point onward.

- The *debugger status monitoring* tool:
  This is an active textbox tool, which monitors the actual execution state of the debug session. Its possible values are: *starting*, *finished*, and the current action number where the execution process is suspended.

14.4.2 Debug Editor and the execution workflow

The execution is paused most of the times during the debug session. When the session is suspended at a breakpoint, the action for which the particular breakpoint was set is marked by a red header, see *Figure 8*.

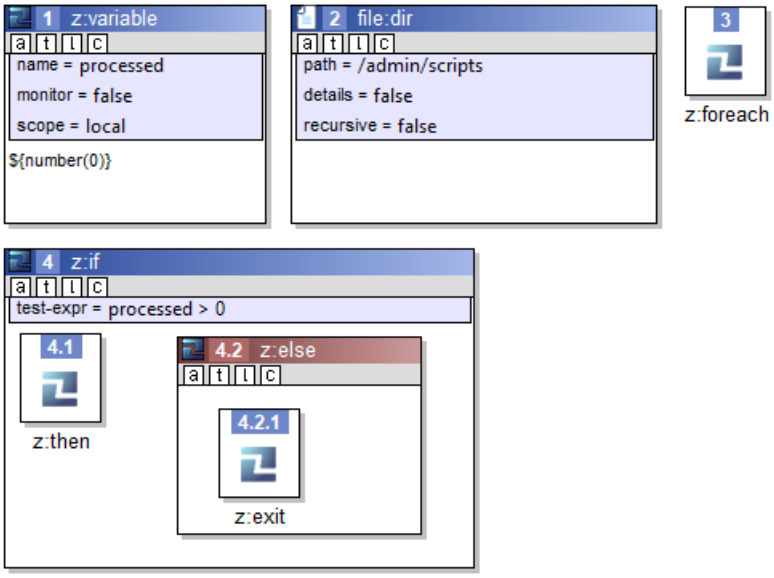


Figure 8 – The current action where the processing is suspended at is marked by a red header

Also, the action ordering number of this action is shown in the status monitoring tool on the main toolbar.

Figure 9 – The status monitoring tool is displaying the action ordering number

14.4.3 Action context menu

In the Debug Editor, only the *Set breakpoint..* menu item is displayed in the action context menu (which opens by right-clicking on an action).

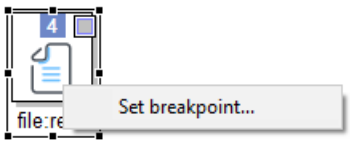


Figure 10 – The *Set breakpoint...* menu item in the action context menu

This menu item opens the *Set breakpoint* dialog box, see *Figure 11.*

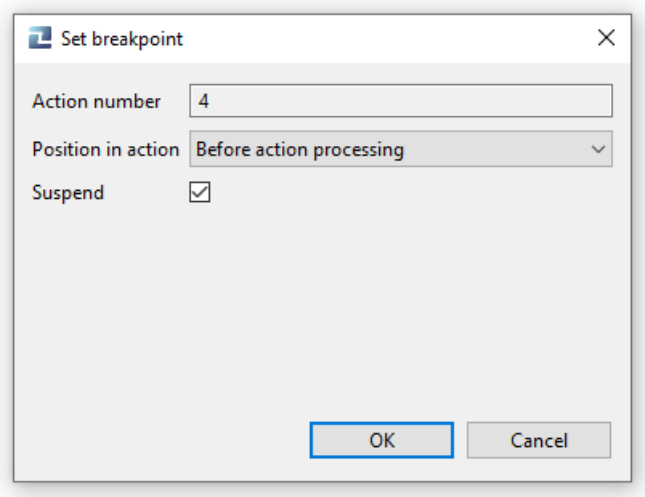


Figure 11 – The *Set breakpoint* dialog box

Here, the user can specify a breakpoint for the debug session. The following fields are available on the dialog:

- *Action number*
  The action ordering number for the breakpoint. It is non-editable.

- *Position in action*
  According to the process how an action is executed (see → Execution of an action), the breakpoint position can be fine-tuned here, see → Debugging concepts and terms. Possible values are: *Before action processing, After attributes processing, After children actions processing* and *After action processing*

- *Suspend*
  By default, job execution is paused at every breakpoint, and for these cases the *Suspend* checbox should be checked. However, if the breakpoint is in a loop, and after the first check it is not needed anymore (or not needed for a certain execution flow branch), it can be unchecked, so the breakpoint will not suspend the execution process.

After pressing the *OK* button, a new breakpoint is added to the action. Existing breakpoints can be viewed and deleted only in the *Breakpoints window*, see below.

14.4.4 Breakpoints window

The *Breakpoints window* is an extension window for the currently active Debug Editor (and the currently monitored debug session). It maintains a list of breakpoints specified for the current session, see *Figure 12*.

The following columns are defined for the table displaying the breakpoints:

- *Ordering*
  The action ordering number where the breakpoint was added to, see → Ordering numbers.

- *Action*
  The fully qualified name of the action, see → Action groups and action name.

- *Position*
  The breakpoint position within the action, see → Debugging concepts and terms.

- *Suspend*
  Whether the breakpoint pauses the execution of the job.

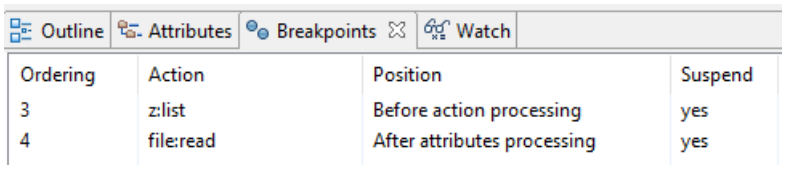


Figure 12 – The *Breakpoints* window

A context menu can be opened by right-clicking on a selected breakpoint, see *Figure 13.*

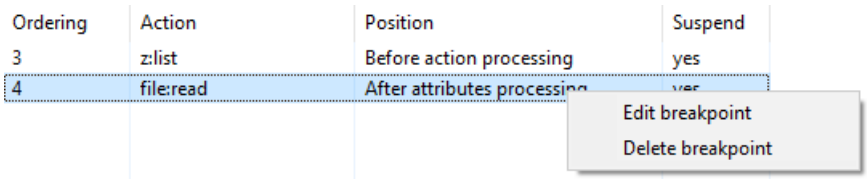


Figure 13 – The context menu of the items in the *Breakpoints* window

Two menu items are available in the context menu. If the user selects the *Edit breakpoint* menu item, the same *Set breakpoint* dialog box appears as by selecting the *Set breakpoint* menu item from the action context menu, see *Figure 11.* The *Delete breakpoint* menu item deletes the selected breakpoint.

14.4.5 Watch window

The *Watch window* is another extension window for the currently opened Debug Editor and the attached debug session, see *Figure 14*.

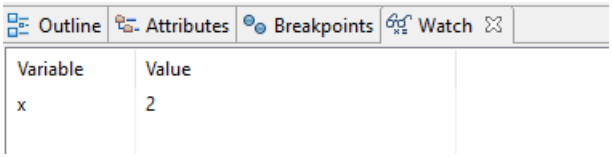


Figure 14 – The *Watch* window

The watch window is empty by default. The user needs to specify a variable or an engine expression in the *Variable* column by simply clicking on the first empty line in that column and type the expression or variable in the text field. After pressing *Enter,* the specified input is evaluated, and its result (if any) is displayed in the *Value* column.

Any kind of valid engine expression can be typed in, not just simple variable names. If the specified expression cannot be evaluated yet, the result remains empty until it can have a value. The `$` and the `${}` formats should not be used here, the processing engine expects the input format without these qualifiers.

For example, in *Figure 15.,* variable `x` is evaluated successfully at the current point in execution, but variable `y` does not have a value yet, so its value is empty. The expressions `2+2` and `date` can be properly evaluated at any point in the script, so the values are filled immediately after the user typed them in.

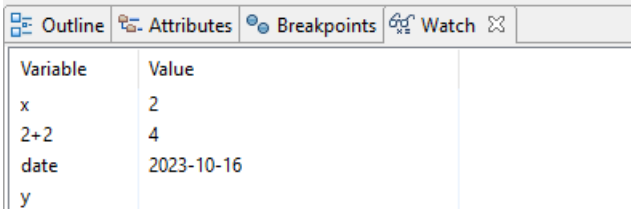


Figure 15 – Various engine expressions and variables can be specified in the *Watch* window

There are several use cases for monitoring the watch variables in a debug session:

- *Checking the value of the result of an action*
  When there is a result of an action which is used in the further execution process as a variable (e.g. created and named by an *alias* attribute), the user can type in the variable name after the action has been executed at a suspended breakpoint. It is an easy way to check a sub-result in the execution logic.

- *Checking a particular variable and the changes of its value during execution:*
  If there is a variable (in most of the cases defined by a `z:variable` action) that the script is using from the beginning till the end, or in a loop, the changes of the value of this variable can be monitored througout the whole execution. The user types the variable name right after the debug session is opened in the Debug Editor, and checks its changes when stepping the debugger from breakpoint to breakpoint (or from action to the next sibling).

14.5 Best practices

- *Using a reference to result attributes*
  It is possible to watch the result attributes of an action in the format of *namespace* + _ + *action name* (e.g. `file_dir`). The user can use this reference in the Watch window after an action has been executed, and it is valid until it is overwritten by another action of the same fully qualified name (e.g. another `file_dir` action).
  Example:

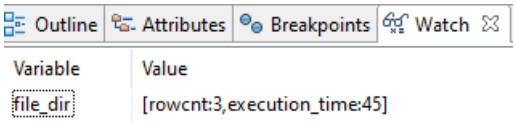


Figure 16 – The result attributes of the last executed `file:dir` action are shown in the *Watch* window

- *Setting the maximum running timeout*
  Because the debug session is a special case of normal script execution, the same timeout values are applied to the job of the debug session as if it was executed in a standard way. Hence it is recommended to set the maximum running timeout large enough, see → List of execution options.

15. Initiating script execution

Any Zagreus script can easily be executed from the Zagreus Client (see → Zagreus Client). In addition to this, Zagreus also has powerful automated processes and script executing features. Making sure that the IT operations are started at the proper time or are activated by the required event is crucial for the automation of such activities. Zagreus Subscriptions allow processes to be scheduled, started at a given time, or triggered by another task, depending on the requirements.

15.1 Overview

Besides manual execution, script executions can be triggered by specific *events* as well, such as specific time instances (e.g. every Monday at 10:00 AM), fire events or OS-based events such as the deletion of a file from a specific folder. To handle such events for script execution, *event-type resources* have to be created. Examples for event-based resources are *time schedules* (defining a set of time instances when script execution should be triggered), *database watchers* (monitoring database tables for satisfying specified conditions, such as the number of rows returned by a specific SQL query), or *file triggers* (triggered by certain file manipulations in a specified OS folder). Choosing the right event-based resource for an actual use case is a key step in automating processes.

Next, if the condition defined in the event-type resource (if any) is satisfied, it triggers the associated script or scripts. For this, in Zagreus the so-called *subscriptions* are used, which link the event-type resource with a script. Whenever the event-type resource is actived, the subscriptions linked to it will be active as well active, and the execution of the corresponding scripts will be triggered. The subscriptions are stored in the local database. In general, any subscription defines the link between a script and an event-type resource, allowing to realize a many-to-many relation between them: an event-type resource can start multiple scripts and a script can belong to multiple event-type resources as well.

There are also cases where automated script execution happens without a subscription. For example, script execution can be initiated by another script using the zs:runscript action, or special events such as ‘server autorun’. See also → Special eventsSpecial .

15.1.1 Manual execution

Zagreus offers several methods for manual script execution. Using the Zagreus Client is the most common way to execute scripts (see → Manual script execution). There are also command-line executables in Windows and Linux environments for firing events, running scripts, and other administrative tasks (see → Execution from the command-line client). Another convenient interface for running scripts is the Zargeus HTML application (see → Execution from the Zagreus HTML Application).

In addition to these, execution of Zagreus scripts is also possible by using web service calls from other platforms – such as Oracle, MSSQL database environments or the Niota application. (For web service calls, see → Execution from external systems).

15.1.2 Execution by event-type resources

The other type of script execution happens via using *event-type* resources. The types of event-type resources are as listed below:

- *event schedule*
  When an event schedule is activated, it simply triggers the execution of the subscribed scripts without evaluating any conditions. Triggering the event schedule can be done by using the action `zs:fireevent`, using the Zagreus command-line client (see → fireevent script), using the Zagreus HTML Application (see → Fire event tab) or manually in the Browser window of the Zagreus Client (see → Operations for event-type resources). Event schedules can be created and edited in the *Zagreus event schedule editor* of the Zagreus Client application, see → Event schedule

- *time schedule*
  A time schedule represents a set of time points at which the schedule is activated. Time schedules cannot be fired manually. Time schedules can be created and edited in the *Zagreus time schedule editor* of the Zagreus Client application, see → Time schedule.

- *mail watcher*
  A mail watcher regularly polls a target e-mail server to check the content of an inbox. It is triggered when the specified condition (usually the existence of a specified email) is evaluated as true. Mail watchers can be configured and edited in the *Zagreus mail watcher editor* of the Zagreus Client application, see → Mail watcher

- *database watcher*
  A database watcher regularly polls a target database server. It is triggered when a specified condition (typically referring the result of an SQL query) is evaluated as true. Database watchers can be configured and edited in the *Zagreus database watcher editor* of the Zagreus Client application, see → Database watcher

- *file trigger*
  A file trigger is continuously monitoring some part (e.g. a folder) of the Zagreus file system. It will be triggered when a specific file event (such as file creation,

modification or deletion) occurs. File triggers can be configured and edited in the *Zagreus file trigger editor* of the Zagreus Client application, see → File trigger

In addition, there are special triggers that are mostly used for error handling and administrative tasks.

The *execute_script_on_error* and *execute_script_on_cancel* are script execution options (see → List of execution options). If the particular script ends with *error* or *cancelled* status (see → Job lifecycle), the script specified by these options is executed.

Furthermore, scripts can be automatically executed when a particular user logs in with the Zagreus Client application, or when the server starts up. To configure these, the scripts for automatic execution have to be listed in the files `.autorun` and `.serverautorun`, respectively, see → Script execution by autorun configuration files.

15.2 Execution options

Script execution can be fine-tuned by so-called *execution options*, which are specific settings that control how execution should be performed, or specifying details for other execution features. Execution options are applied only for a specific job, not the script itself.

15.2.1 Declaration levels

In Zagreus, execution options can be declared on several different logical levels. These logical levels are:

- *server*
  This level defines the most general scope of option declarations. The user needs to set the server level options in the configuration of the Zagreus Server, see → Server-level execution options.

- *worker*
  This level is only for the execution option *log_level* (for compatibility reasons only, because of the former *engine.loglevel* setting).
  The user needs to set this worker level option in the configuration of the Zagreus Worker, see → Worker-level execution options.

- *owner*
  This level serves for option declarations specific to the user or group which owns the actual script whose execution is initiated. The user needs to set the owner level options by using the *Set user variables and options…* or *Set group variables and options…* menu items in the Zagreus browser window in the Zagreus Client, see → Context menu of a user node and → Context menu of a group node.

- *script*
  This level serves for option declarations of the actual script whose execution was initiated. Options on this level can be set as script options (see → Setting script variables and options). However, the options set for subscriptions (see → Creating a new subscription) override the options set for the script. The same option overriding also happens in case of any webservice calls (such as calls from the Zagreus HTML Application or from the Zagreus command line) or when the

user set execution options by `zs:option` child elements in a `zs:runscript` call.

15.2.2 Precedence order for resolution

The different levels that Zagreus options can be declared on have a specific precedence. The same option set on different declaration levels eventually will be evaluated, using a specific order of precedence. Then the evaluated value of the option will be used for the proper queuing mechanism for the job.

The order of resolution is the following (from the lowest to the highest priority):

- *default (implicit level)*
- server
- worker
- owner
- script

The first logical level (default level) is an implicit level meaning that all options have a default value.

An example for option resolution, when the *priority* execution option is defined on the following levels:

- on the server level *priority* is set to *50*
- on the owner level for the user *test*, *priority* is set to *30*
- on the owner level for the group *public*, *priority* is set to *70*
- on the script level for the script *sample_script*, *priority* is set to *10*

The value of the execution option *priority* is resolved as:

- *priority=10*
  for the script *sample_script*

- *priority=30*
  if user *test* is the owner of the script and it is not the script *sample_script*

- *priority=70*
  if group *public* is the owner of the script and it is not *sample_script*

- *priority=50*
  for all the scripts whose owners are different than user *test* and group *public* and the script is not *sample_script*

15.2.3 List of execution options

The Zagreus System defines quite a few execution options. The listed options were introduced from Zagreus version 1.5.6.1, but some of the options has already existed as script options, or properties in the Zagreus Server or Zagreus Worker configuration under different names, they are listed in the *compatibility* field. Zagreus 1.5.6.1 maintains backward compatibility with these old options and properties, but if an option is specified with the regularized new name, the latter takes precedence.

- *running_timeout*
  The maximum duration which the job is allowed to be executed. If the execution time exceeds the value of this option (specified in milliseconds), the job will be finished with *running timeout* status, see → Job lifecycle. The value -1 means unlimited, i.e. there is no running timeout for the job.
  Default value: 3600000 (i.e. 1 hour)
  Minimum value: -1
  Compatibility: *engine.maxrunning* on script level, *queue.runningtimeout* on server level

- *queuing_timeout*
  The maximum duration which the job is allowed to be queued. If the queuing time exceeds the value of this option (specified in milliseconds), the job will be removed from the queue with the *queue timeout* status, see → Job lifecycle. The value -1 means unlimited, i.e. there is no queuing timeout for the job.
  Default value: 60000 (i.e. 1 minute)
  Minimum value: -1
  Compatibility: *job.max_queuing_time* on script level, *queue. queuingtimeout* on server level

- *maximum_parallel_execution*
  Limits the number of times the given script can be executed at the same time. The value -1 means unlimited.
  Default value: -1

Minimum value: -1
Compatibility: *engine.maxparallelexecution* on script level, *queue.maximum.parallel.execution* on server level

- *maximum_parallel_queuing*
  Limits the number of times the given script can be queued at the same time. The value -1 means unlimited.
  Default value: -1
  Minimum value: -1
  Compatibility: *engine.maxparallelqueuing* on script level, *queue.maximum.parallel.queuing* on server level

- *log_level*
  The applied *loglevel* (see → Logging levels and loglevel) during execution. Available values are: 'user', 'info', 'warning', 'error' and 'debug'.
  Default value: info
  Compatibility: *engine.loglevel* on script level, *engine.loglevel* on worker level

- *executing_user_name*
  If filled, the script will be executed on the behalf of the user with the specified username. This option will only be applied if the user who originally initiated the script execution is an administrator.
  Compatibility: *executing.username* on script level

- *executing_user_id*
  If filled, the script will be executed on the behalf of the user with the specified ID. This option will only be applied if the user who originally initiated the script execution is an administrator.

- *priority*
  The priority of the job.
  Default value: 10
  Minimum value: 0
  Maximum value: 10000
  Compatibility: *script.priority* on script level, *queue.script.priority* on server level

- *queue_group_id*
  When it is set, only Zagreus Workers that are associated to the specified queue group ID are allowed to execute the script, see also → Queue groups. The value can be a single number or a comma separated list of numbers.
  Minimum value: 1
  Maximum value: 99
  Compatibility: *script.queueId* on script level, *queue.default.queueid* on server level, *queue.default.<username>.queueId* also on server level (now the same setting is *queue_group_id* option on owner level)

- *job_monitoring*
  If set to false, the job is not shown in the Active Job window and Finished Job window in the Zagreus Client (see → Finished jobs window) and in the timeline of the Zagreus Monitor (see → Timeline area).
  Default value: *true*
  Compatibility: *job.monitoring* on script level

- *execute_script_on_error*
  When this option is filled with a script fullpath, and the job execution is finished with an *error* status, the execution of the specified script in the path will be initiated, see → Script execution by script options.
  Compatibility: *error.runscript* on script level

- *execute_script_on_cancel*
  When this option is filled with a script fullpath, and the job execution is finished with a *cancel* status, the execution of the specified script in the path will be initiated, see → Script execution by script options.
  Compatibility: *cancel.runscript* on script level

- *execute_script_on_cancel_source*
  This option serves as an additional filter condition for the *execute_script_on_cancel* option, see → Script execution by script options. Possible values are 'gui' (referring to the Zagreus Client), 'monitor ', 'zs ' and 'server '. Multiple values can be listed separated by commas, e.g. 'zs,server'. When left empty, all cancel types trigger the execution.
  Compatibility: *cancel.runscript.source* on script level

15.2.4 Prefixes

Although options with the same name declared on different levels override each other, the original values are stored in the Zagreus System, and can be checked for debugging purposes. In order to distinguish these option values, Zagreus uses proper prefixes based on the level of declaration. Thus:

- `default.` for the default values of options
- `server.` for server level declarations
- `owner.` for owner level declarations (i.e. users and groups)
- `worker.` for worker level declarations
- `script.` for script level declarations

Option names with the proper prefixes are called *fully-qualified option name*. For example, *server.priority* is referring to the option *priority* declared on the server level, even though it was overridden during option resolution (e.g. by the *priority* option defined on the script level).

The full list of the execution options with fully-qualified names can be seen in the Zagreus Monitor application, by checking the *Starting options* tab (with the *Advanced mode* checkbox checked) of the *Job properties* dialog. *Figure 1.* shows the tab in normal and advanced modes.

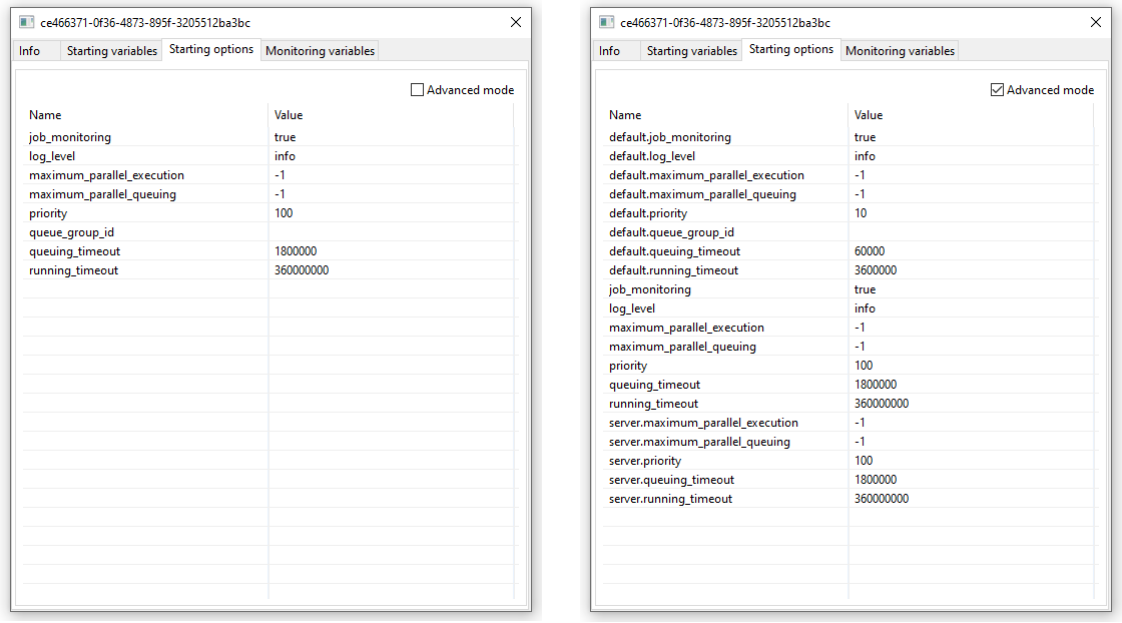


Figure 1 – The *Starting options* tab of the Job properties dialog (normal and advanced mode)

15.3 Start-up variables

Start-up variables are parameters that are (eventually) passed to the script when its execution is initiated. Variables allow the user to provide input to a script, enabling it to perform specific actions or calculations based on the given values.

Fundamentally, all the variables declared on all different levels (e.g. server, queue, user, group, script) eventually are passed to the script being executed. Additionally, new variables can be created inside the script, e.g. by using the `z:variable` action, see → z:variable action. Generally, using variables is a necessity to build up a flexible, efficient and scalable system.

15.3.1 Declaration levels

In Zagreus start-up variables can be declared on several different logical levels. These logical levels are:

- *server*
  This level defines the most general scope of variable declarations. When there are multiple Zagreus Server installations, these variables can be specific to the actual system, or they can simply represent general constant-like values. The user needs to set the server level variables in the configuration of the Zagreus Server, see → Server-level and queue-level variables.

- *queue*
  This level serves for variable declarations specific to the actual queue group. For more details, see → Server-level and queue-level variables.

- *worker*
  This level serves for variable declarations specific to all (or a particular) worker which executes the job. For more details, see → Worker-level variables.

- *owner*
  This level serves for variable declarations specific to the user or group which owns the actual script whose execution is initiated. The user needs to set the owner level variables by using the *Set user variables and options…* or *Set group variables and options…* menu items in the Zagreus browser window in the Zagreus Client, see → Context menu of a user node and → Context menu of a group node.

- *script*
  This level serves for variable declarations of the actual script whose execution was initiated. Variables on this level can be set as script variables (see → Setting script variables and options). However, the variables set for subscriptions (see → Creating a new subscription) override the variables set for the script. The same overriding also happens in case of any webservice calls (such as calls from the Zagreus HTML Application or from the Zagreus command line) or when the user set variables by `zs:variable` child elements in a `zs:runscript` call.

Declaring Zagreus variables on the proper level is important to build up and maintain an efficient and well-structured system. For example, in a multi-user system, the same

variable can hold different values for each different user, e.g. *emailaddress=user@host.com* . This variable is recommended to be declared on the *user* level. Another example is using a general variable *environment=PROD* , which indicates that the current Zagreus installation is in a production environment. This variable is then recommended to be declared on the *server* level.

15.3.2 Precedence order for resolution

The different levels that Zagreus variables can be declared on have a specific precedence. Variables with the same name (declared on different levels) eventually will be used as one single variable during the execution of the job, so these variables override each other.

The order of resolution is the following (from the lowest to the highest priority):

- server
- queue
- worker
- group
- user
- script
- *job (implicit level)*

The last logical level (job level) is an implicit level meaning that the user can not declare variables on this level. However, many job-related start-up variables are specified automatically by the execution system, see below the list of them.

An example for variable resolution, when an *email* variable is defined on the following levels:

- on the server level with a value *server@host.com*
- on the owner level for the user *test* with a value *test@host.com*
- on the owner level for the group *public* with a value *public@host.com*
- on the script level for the script *sample_script* with a value sample_script@*host.com*

The value of *email* variable is resolved as:

- *sample_script@host.com*

if the script is *sample_script*

- *test@host.com*
  If the owner of the script is user *test* and the script is not *sample_script*

- *public@host.com*
  If the owner of the script is group *public* and the script is not *sample_script*

- *server@host.com*
  for all the scripts whose owners are different than user *test* and group *public* and the script is not *sample_script*

15.3.3 Prefixes

Although variables with the same name declared on different levels override each other, the original values are stored in the Zagreus System, and can be checked for debugging purposes. In order to distinguish these variables, Zagreus uses proper prefixes based on the level of declaration. Thus:

- `server.` for server level declarations
- `queue.` for queue level declarations
- `worker.` for worker level declarations
- `owner.` for group level declarations
- `script.` for script level declarations
- `job.` for job level declarations

Variable names with the proper prefixes are called *fully-qualified variable name*. For example, *server.email* is referring to the variable *email* declared on the server level, even though it was overridden during variable resolution (e.g. by the *email* variable defined on the script level).

The full list of the start-up variables with fully-qualified names can be seen in the Zagreus Monitor application, by checking the *Starting variables* tab (with the *Advanced mode* checkbox checked) of the *Job properties* dialog. *Figure 2.* shows the tab in normal and advanced modes.

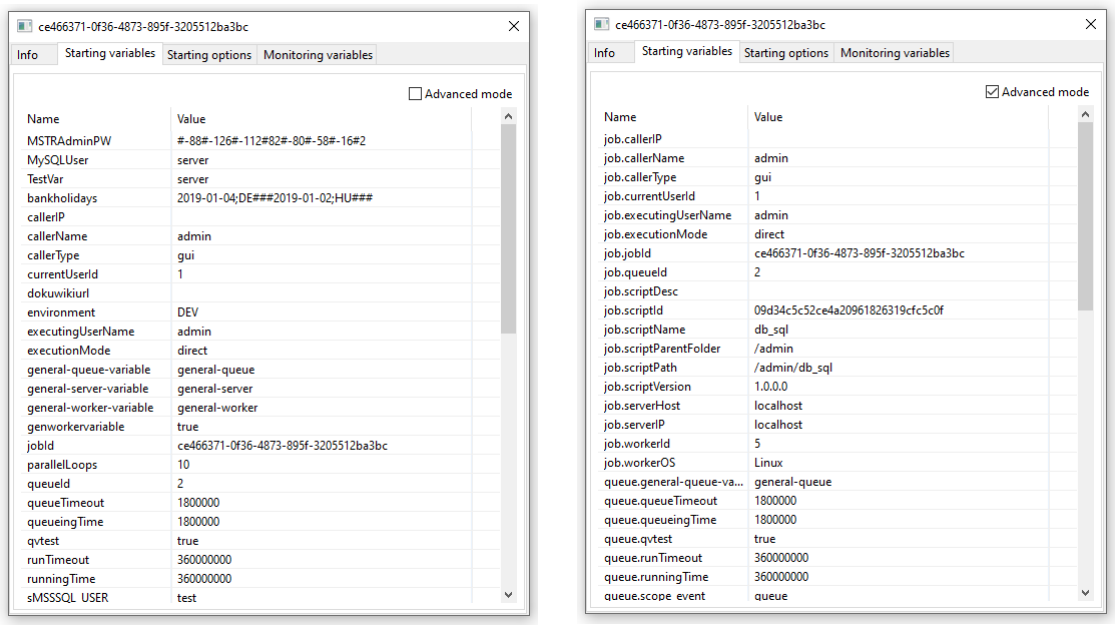


Figure 2 – The *Starting variables* tab of the Job properties dialog (normal and advanced mode)

15.3.4 Automatically set start-up variables

There are start-up variables that are automatically set when the job execution is starting. These variables provide information about the execution environment and how the execution has been initiated.

These variables also have a fully-qualified name with proper prefixes. Referencing them can be done in the same way as the user declared variables, both the resolved name and the fully qualified name work, see → Prefixes.

Variables with the prefix *job*:

- *job.jobId*
  the ID of the job

- *job.scriptId*
  the ID of the executed script

- *job.scriptVersion*
  the version of the executed script

- *job.scriptName*
  the name of the executed script

- *job.scriptPath*
  the full path of the executed script

- *job.scriptDesc*
  the description of the executed script

- *job.scriptParentFolder*
  the full path of the parent folder of the executed script

- *job.queueId*
  the queue group ID (see → Queue groups) of the job. Empty when there are no queue groups defined

- *job.currentUserId*
  the ID of the executing user

- *job.executingUserName*
  the name of the executing user

- *job.scheduleId*
  in case when the job execution was initiated by a subscription (see → Subscriptions), it is the ID of the associated time schedule or event-type resource

- *job.scheduleName*
  in case when the job execution was initiated by a subscription (see → Subscriptions), it is the name of the associated time schedule or event-type resource

- *job.schedulePath*
  in case when the job execution was initiated by a subscription (see → Subscriptions), it is the full path of the associated time schedule or event-type resource

- *job.subscriptionId*
  in case when the job execution was initiated by a subscription (see → Subscriptions), it is the ID of the subscription

- *job.home*
  the full path of the home folder of the scipt owner

- *job.callerType*
  the caller type of the job, see → Caller and caller type

- *job.executionMode*
  the execution mode of the job, see → Job properties

- *job.callerName*
  the caller name of the job, see → Caller and caller type

- *job.callerIP*
  the caller IP of the job, see → Caller and caller type

- *job.callerResourceId*
  the ID of the resource which initiated the execution of the script. When it was initiated by a subscription, it is the ID of the associated event-type resource (see → Execution by event-type resources). When it was a `zs:runscript` action, the value of this variable is the ID of the caller script. The ID does not contain the version, which is set in a variable *job.callerResourceVersion*

- *job.callerResourceVersion*
  the version of the resource which initiated the execution of the script. When it was initiated by a subscription, it is the version of the associated event-type resource (see → Execution by event-type resources). When it was a `zs:runscript` action, the value of this variable is the version of the caller script.

- *job.callerResourceName*
  the name of the resource which initiated the execution of the script. When it was initiated by a subscription, it is the name of the associated event-type resource (see → Execution by event-type resources). When it was a `zs:runscript` action, the value of this variable is the name of the caller script.

- *job.callerResourcePath*
  the full path of the resource which initiated the execution of the script. When it was initiated by a subscription, it is the full path of the associated event-type resource (see → Execution by event-type resources). When it was a

`zs:runscript` action, the value of this variable is the full path of the caller script.

- *job.serverHost*
  the host of the Zagreus Server to which the actual worker is connected

- *job.serverPort*
  the port of the Zagreus Server to which the actual worker is connected

- *job.workerId*
  the ID of the worker that is executing the job

- *job.workerOS*
  the operating system of the worker that is executing the job

Variables with the prefix *server*:

- *server.bankholidays*
  the full path of the optional bank holidays descriptor file, see → Bank holidays feature.

Variables with the prefix *worker*:

- *worker.parallelLoops:*
  The number of allowed parallel loops defined in the installed Zagreus Licence, see → Licencing.

15.3.5 List of resolved start-up variables

Resolving the start-up variables is done right before the execution of the job. The list of these variables is logged into the job-log file (see → job-log file), allowing the user to check the final variable resolution. This may prove to be useful e.g. in the following cases:

- Checking configuration when variables with the same name were declared on different levels (e.g. server, owner, script levels).

- Checking values of dynamically set start-up variables. For example, if there is a job that is started by a `zs:runscript` action from another script, and the `zs:runscript` action specifies some start-up variables dynamically. In this case, the values of those start-up variables can be checked.

- Checking caller properties for jobs whose execution were initiated automatically (e.g. by subscriptions or `zs:runscript` action). The caller properties are passed as automatically set start-up variables (e.g. *callerName*, *callerIP*, *subscriptionId*).

Figure 3. below shows a beginning of a sample job-log file:

```
Job starting variables:
callerResourceId    d72605847c724adeaf3f98c35aeced38
callerResourceName  every_min
callerResourcePath  /admin/every_min
callerResourceVersion   1.0.0.0
callerType  scheduler
currentUserId   1
docurl  ${server.docurl_1}
docurl_1    http://localhost/%resourceName
docurl_1_replace    ' ',,'+'
docurl_replace  ' ',,'+'
executingUserName   admin
executionMode   scheduled
group_var   10
jobId   3a8c9c25-82ec-4620-8f6a-007d955237d1
parallelLoops   5
queueId 1
queueTimeout    10000
queueingTime    10000
runTimeout  6000000
runningTime 6000000
scriptDesc
scriptId    7730421717c5480d98b0beb76ba4fbd7
scriptName  sample_script
scriptParentFolder  /admin
scriptPath  /admin/sample_script
scriptVersion   1.0.0.0
serverHost  localhost
serverOS    Windows 10
serverPort  7323
serverWorkingFolder c:/Programme/zagreus/server
subscriptionId  5774
test_var    true
uMailconn
uMySQLUser  root
uwusername  admin
workerId    1
workerOS    Windows 10
worker_execute_R_tempfolder /r
worker_execute_powershell_tempfolder    /powershell
worker_execute_python_tempfolder    /python
x   2
y   4
zagreusVersion  1.5.5.6P02
```

Figure 3 – The list of start-up variables in the job-log file of the finished job in the Zagreus Client

15.4 Subscriptions

In the Zagreus System, to allow automatic task execution, the scripts (see → Scripts) specify the tasks to be executed and the event-type resources (see → Execution by event-type resources) are responsible for the starting of the execution in the appropriate time or at the occurrence of the appropriate event. The scripts and the event-type resources have an *n:m* relation, which means that multiple scripts can be subscribed to the same event-type resource, and also one script can be subscribed to multiple event-type resources. The Zagreus System uses the *subscriptions* to establish the connection between one script and one event-type resource, helping to establish this *n:m* relation.

A subscription has the following properties:

- *id*: identifier of the subscription resource. It is an automatically assigned unique integer, and it cannot be changed.
- *alias*: a human-readable name of the subscription.
- *schedule path*: the path of the event-type resource, associated with the subscription
- *script id*: the id of the script resource which is associated with the subscription
- *active*: whether the subscription is active (i.e. it executes the subscribed scripts) or not
- subscription *variables*: variables (see → Start-up variables) can also be defined on subscription-level, which are passed to the executed script
- subscription *options*: options (see → Execution options) can also be defined on subscription-level, which are used during script execution

15.4.1 Subscriptions from the perspective of scripts

Subscriptions are usually maintained from the perspective of the scripts, however it is possible to check and remove subscriptions from the perspective of an event-type resource.

The *Subscriptions* window can be opened by right clicking on the particular script and selecting the menu item *Script subscriptions*... from the context menu (see *Figure 23.*).

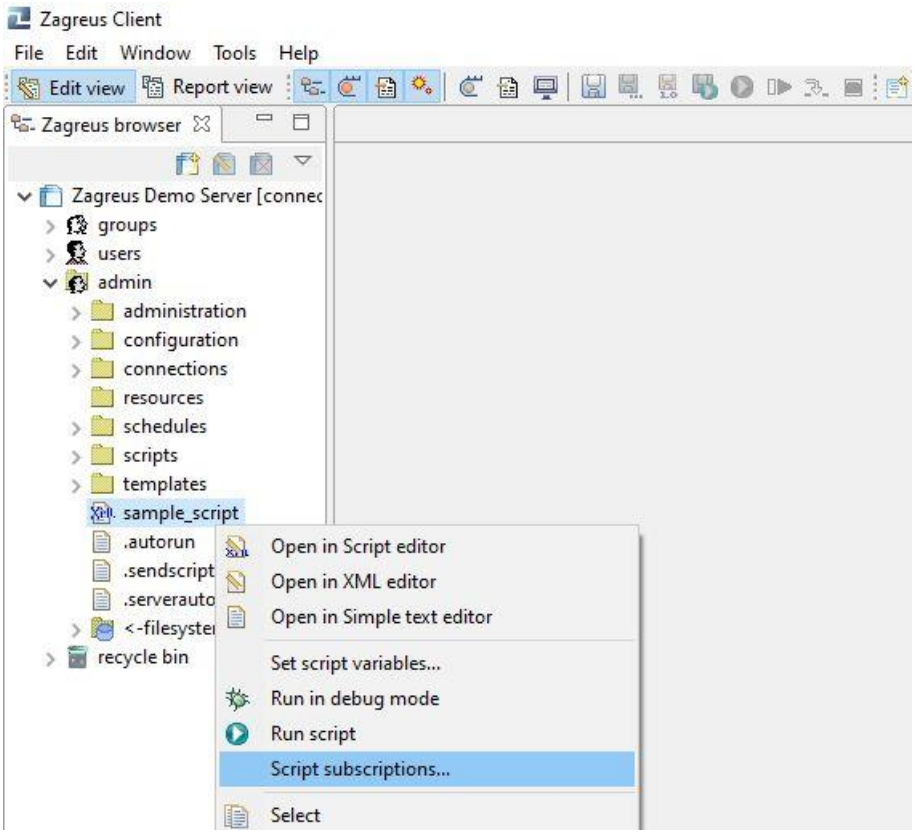


Figure 23 – Opening the *Subscriptions* window from the context menu of a script

In the *Subscriptions* window (see *Figure 24.*) one may check, edit, or remove subscriptions belonging to the particular script.

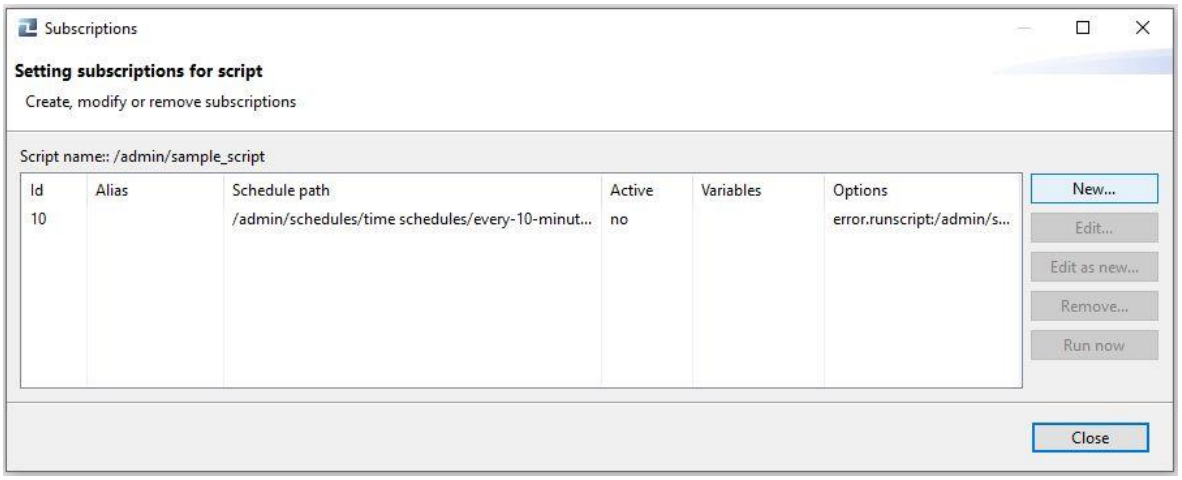


Figure 24 – The *Subscriptions* window

15.4.1.1 Creating a new subscription

A new subscription can be created by clicking on the *New* button in the *Subscriptions* window (see *Figure 24.*). The opening *Create new subscription* window has three tabs:

- *Subscribe to*: allows to set the general properties of the subscription
- *Script variables*: allows to set the script variables of the subscription
- *Executing and queuing options*: allows to set the options of the subscription

The main subscription settings can be edited under the *Subscribe to* tab, see *Figure 25*. Users can define here the alias of the schedule by entering the value in the *Alias* textbox. The active / inactive status can be toggled by the *Active* checkbox. The watcher or trigger can be chosen from the list of all watchers and triggers in the given Zagreus System (for these, the resource path is shown). To make this selection easier, it is also possible to filter on the type of the watcher or trigger by using the *Type* drop-down.

The new subscription can be saved with the *Create new subscription* button, while the subscription can be discarded by using the *Cancel* button.

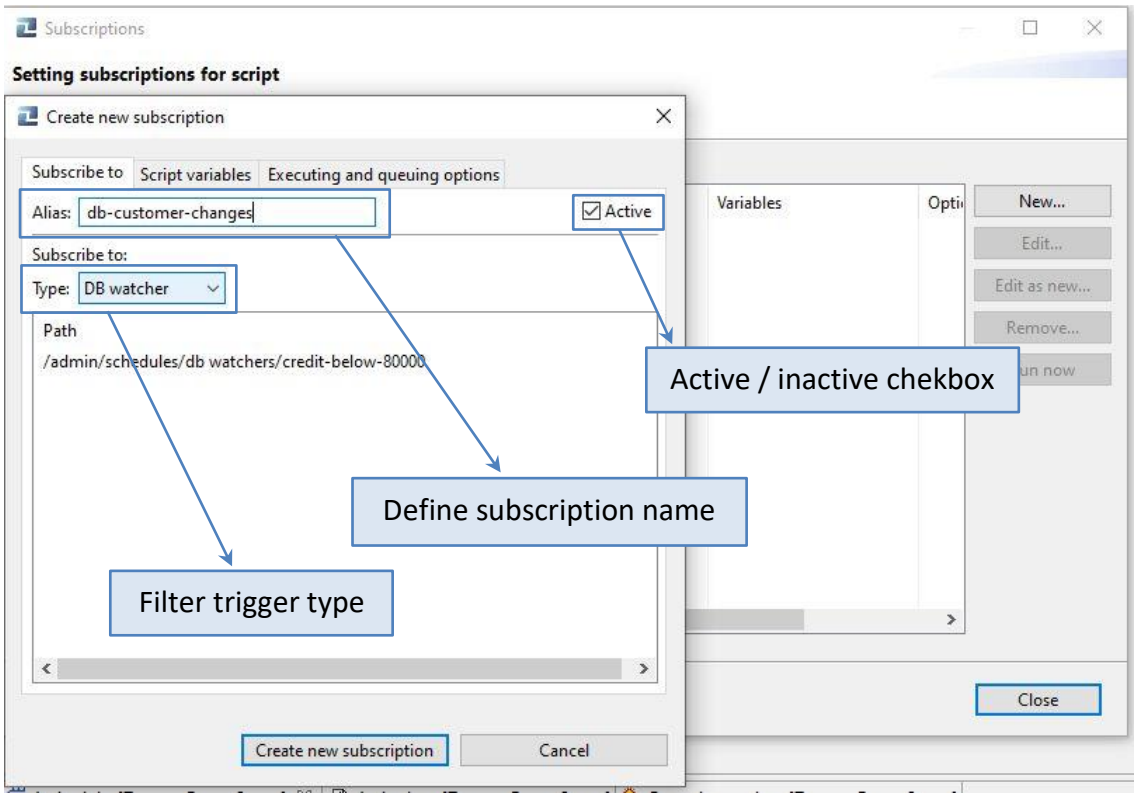


Figure 25 – Settings of the *Subscribe to* tab of the *Create new subscription* window

The *Script variables* tab allows the user to specify and edit the variables which can be used during script execution (see *Figure 26.*). Variables defined for the script (listed in the *Script variables and options* dialog, see → Setting script variables and options) are overridden with the variables defined here if they have the same name. By the *Reload from script* button, it is possible to delete the currently defined variables and to load the variable values which are defined in the *Set script variables* window (e.g. for further editing).

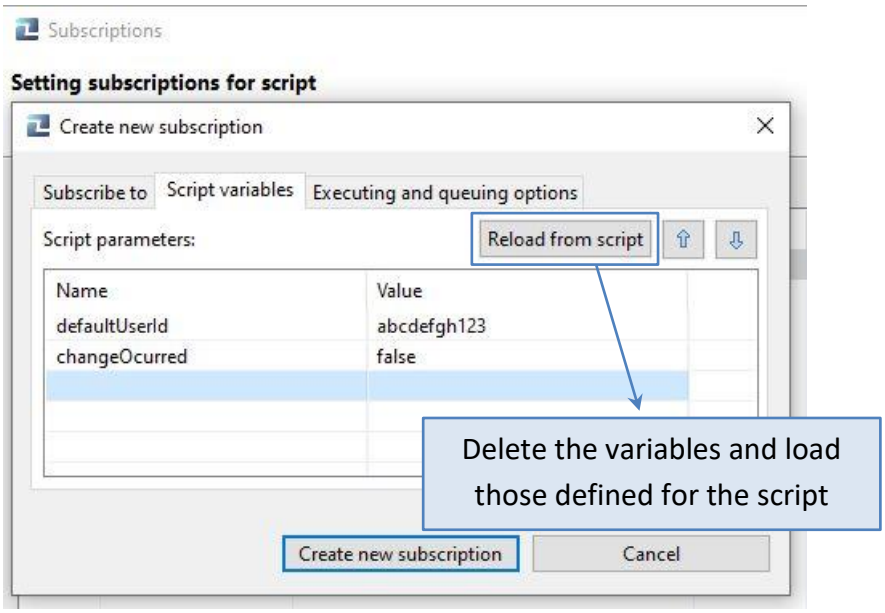


Figure 26 – The settings of the *Script variables* tab of the *Create new subscription* window

The layout and the options present in the *Executing and queuing options* tab are the same as in the window *Script variables and options* (see → Setting script variables and options). Similarly to the case of the variables, the executing and queuing options defined here override the settings of the script during execution – of course, only in the case when the given script is executed by the actual subscription.

> ***Info:*** *The settings of the* Script variables *and* Executing and queuing options *tabs extend or override the settings of the script – but only in the case when the script is executed by the subscription.*

When a subscription is created, the button *Run now* becomes active on the *Subscriptions* window (see *Figure 24.*). By clicking this button, the currently selected subscription is executed immediately. It might be a useful feature during fine-tuning and testing a subscription.

15.4.1.2 Editing a subscription

There are two ways to edit an already existing subscription, both accessible from the *Subscriptions* window (see *Figure 24.*). By clicking on the *Edit…* button, the currently selected subscription can be edited in the *Edit subscription* window. The

accessible tabs and settings of this window are the same as those of the *Create new subscription* window (see above).

The second way of editing a subscription is to use the *Edit as new* button (see *Figure 27.*): it creates a copy of the selected subscription in an edit window. After making the necessary changes, it can be saved as a new subscription object.

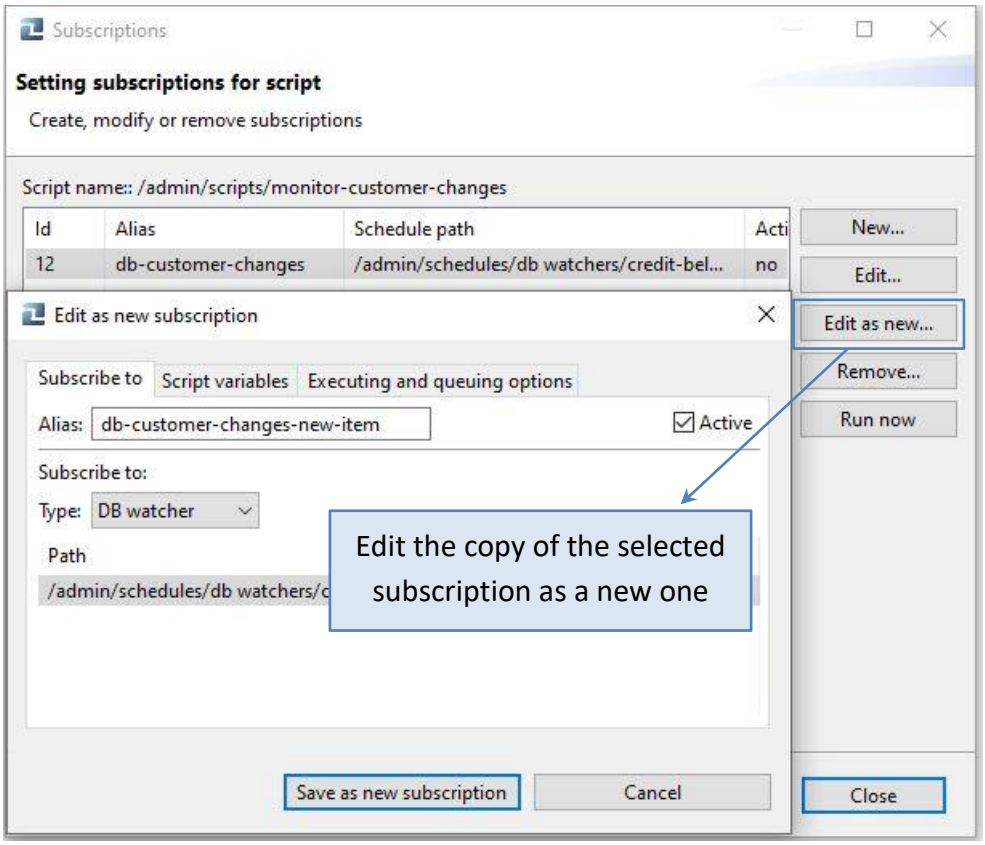


Figure 27 – Editing a subscription as a new one

15.4.1.3 Removing a subscription

The selected subscription can be deleted by clicking on the *Remove* button in the *Subscriptions* window (see *Figure 24.*). Subscriptions are not stored in the recycle bin (see → Recycle bin), but they are deleted permanently.

> ⚠ ***Warning:*** *Unlike all Zagreus resources, subscriptions are deleted permanently.*

15.4.2 Subscriptions from the perspective of event-type resources

Subscriptions associated with an event-type resource (i.e. event schedule, time schedule, mail watcher, database watcher and file trigger) can be displayed by right-clicking on the event-type resource and selecting the menu item *Subscriptions...* from the context menu (see *Figure 28.*).

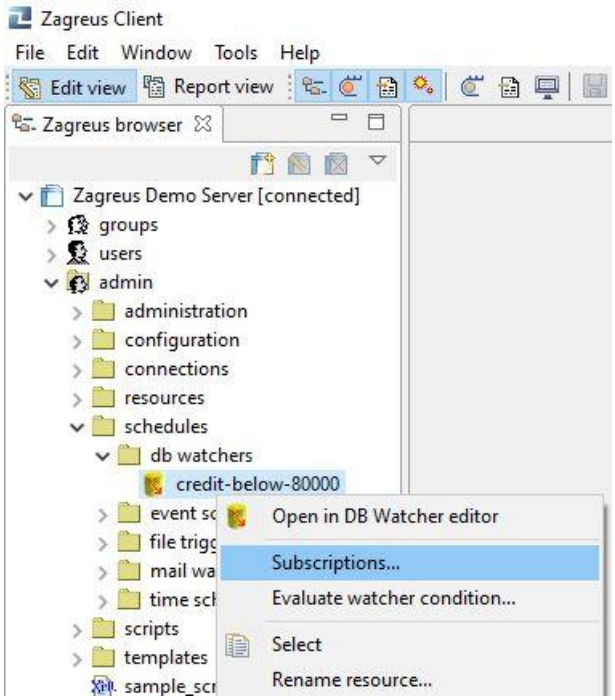


Figure 28 – Opening the *Subscriptions* window of a DB watcher.

In the *Subscriptions* window, the user can view the properties of the associated subscriptions (see *Figure 29.*). The only action currently supported by the Zagreus Client in this window is deleting a subscription, by clicking on the *Remove...* button.

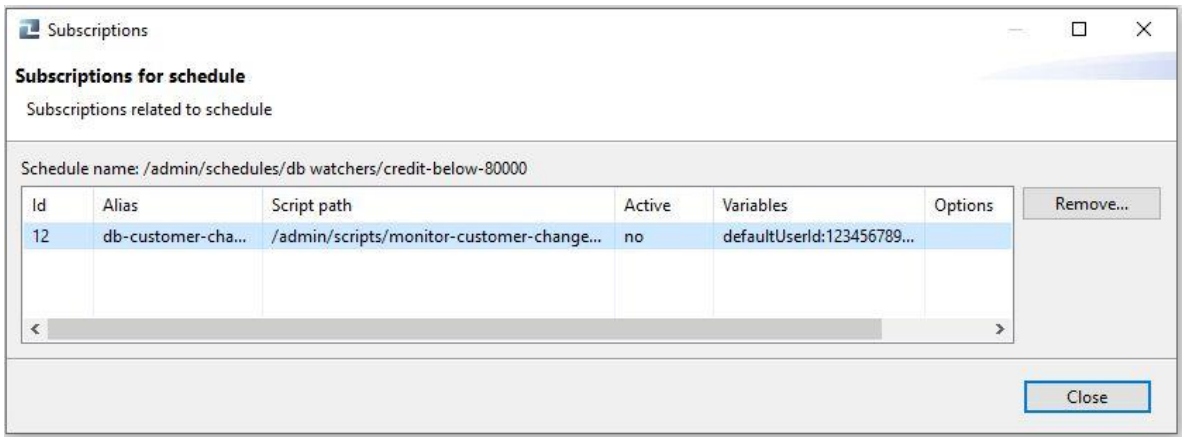


Figure 29 – The *Subscriptions* window of a database watcher

⚠ ***Warning:*** *Unlike all Zagreus resources, subscriptions are deleted permanently*

15.5 Execution by event-type resources

Executing Zagreus scripts via event-type resources is a key step for automation. First, the user has to find out which type of event-type resource fits the given task best. It is also possible to combine such resources: for example, a file watcher executes a script, and inside this script, an event schedule is fired when a condition is satisfied. Zagreus provides solutions both for handling various kinds of events and for performing iterated tasks automatically.

All event-type resources need subscriptions to be assigned to any given script, see → Subscriptions.

> ***Info:*** *To use event-type resources, the Scheduler component of the Zagreus Server must be enabled.*

15.5.1 Event schedule

Event schedules can be created and edited in the “Zagreus event schedule editor”, accessible from the menu item *File / New Resource… / Event schedule*. The only property that can be set for an event schedule is its description (see *Figure 4.*).

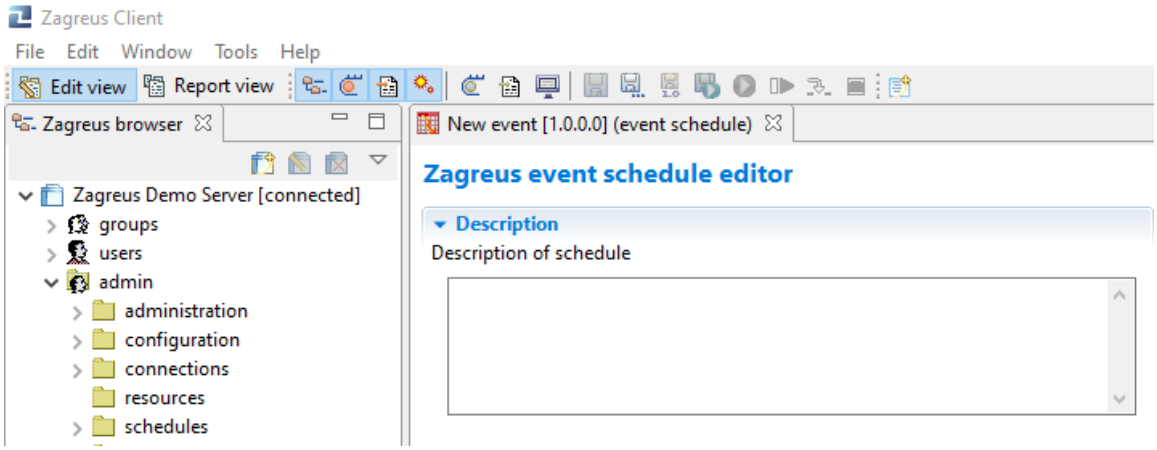


Figure 4 – Event schedule editor

All scripts subscribed to a specific event schedule will be started when the event schedule is fired. It can be done in the following ways:

- *manually in the Zagreus Client*

The user needs to right-click on the specific event schedule in the Zagreus browser window of the Zagreus Client and choose the *Fire event* menu item from the context menu (see *Figure 5.*)

- *manually in the Zagreus HTML application*
  The user needs to manually fire the event in the *Fire event* tab in the Zagreus HTML application (see → Fire event tab)

- *manually via the Zagreus command line application*
  The user needs to or use the fireevent command-line tool (see → fireevent script)

- *with the* `zs:fireevent` *action*
  Firing an event can be triggered with the `zs:fireevent` action in a Zagreus script.

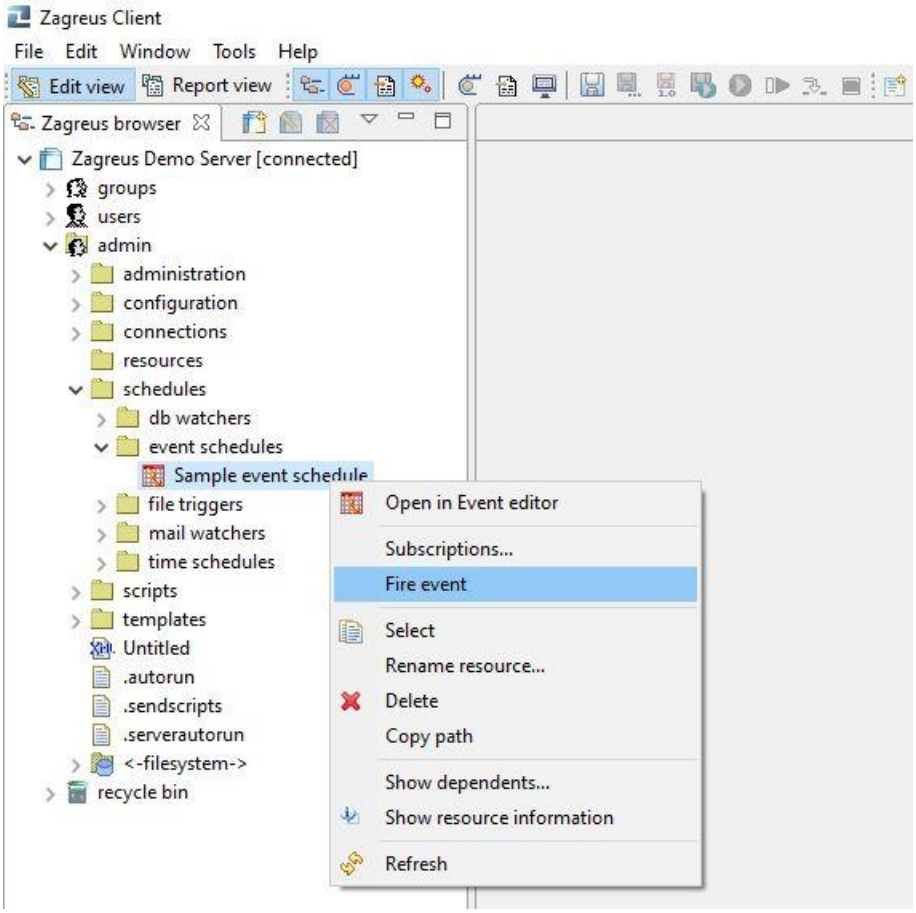


Figure 5 – Firing an event schedule manually in the Zagreus Client

15.5.2 Time schedule

Time schedules can be created and edited in the “Zagreus time schedule editor” (menu item *Menu / File / New Resource / Time schedule*). The properties of firing time and description can be defined for each time schedule object (see *Figure 6.*).

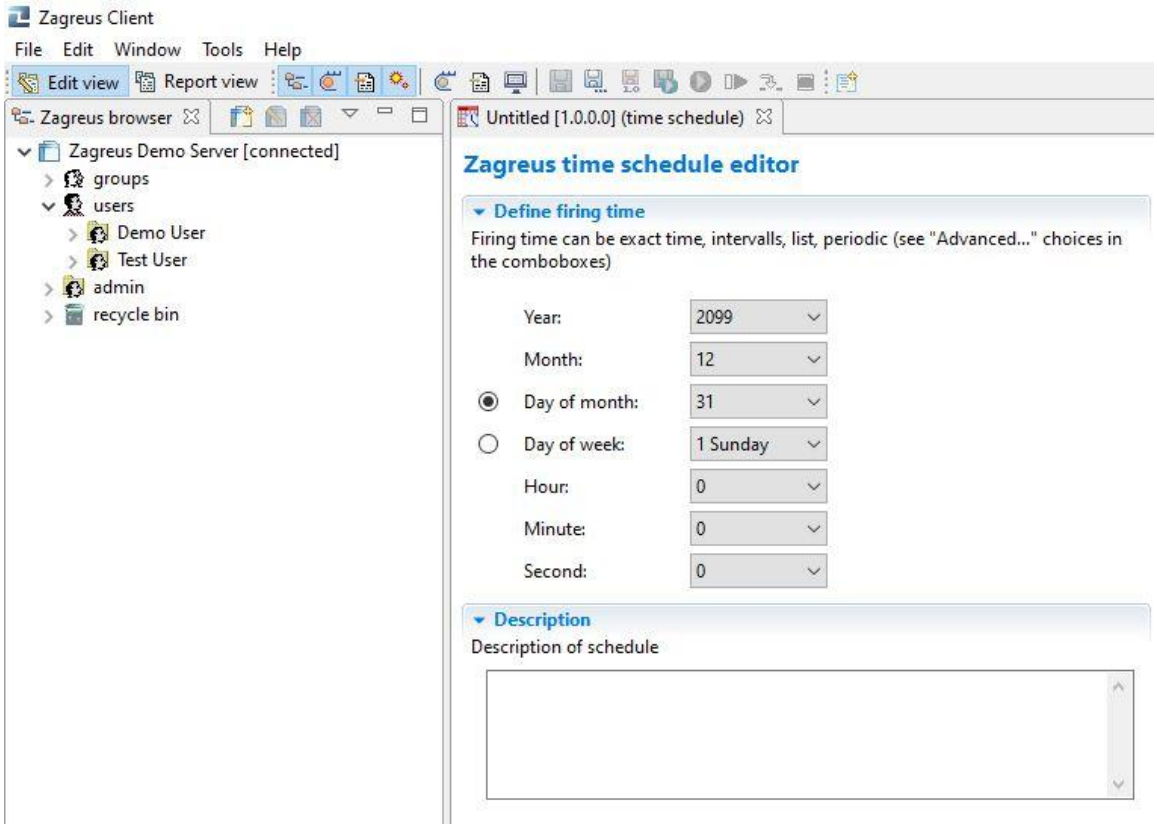


Figure 6 – Opening a time schedule editor tab, showing the default settings

The option to enter an exact value (for example, Year=2021) as well as the value "Every" is available for all fields. Complex patterns (following the Cron format) can be set for each field by choosing the option "Advanced".

Examples for advanced setting:

- Specifying an interval, e.g. *Hour*=9-17
- Specifying a list, e.g. *Hour*=9,10,11,12,13,14,15,16,17
- Specifying a period by using the pattern “<starting time>/<interval>”, e.g. *Minute*=5/15 . This time schedule will fire at the minutes 5, 20, 35 and 50 in every hour.

In *Figure 7.*, time schedule will be triggered on every weekday, between 8:00 and 17:00 at 5, 20, 35, and 50 minutes:

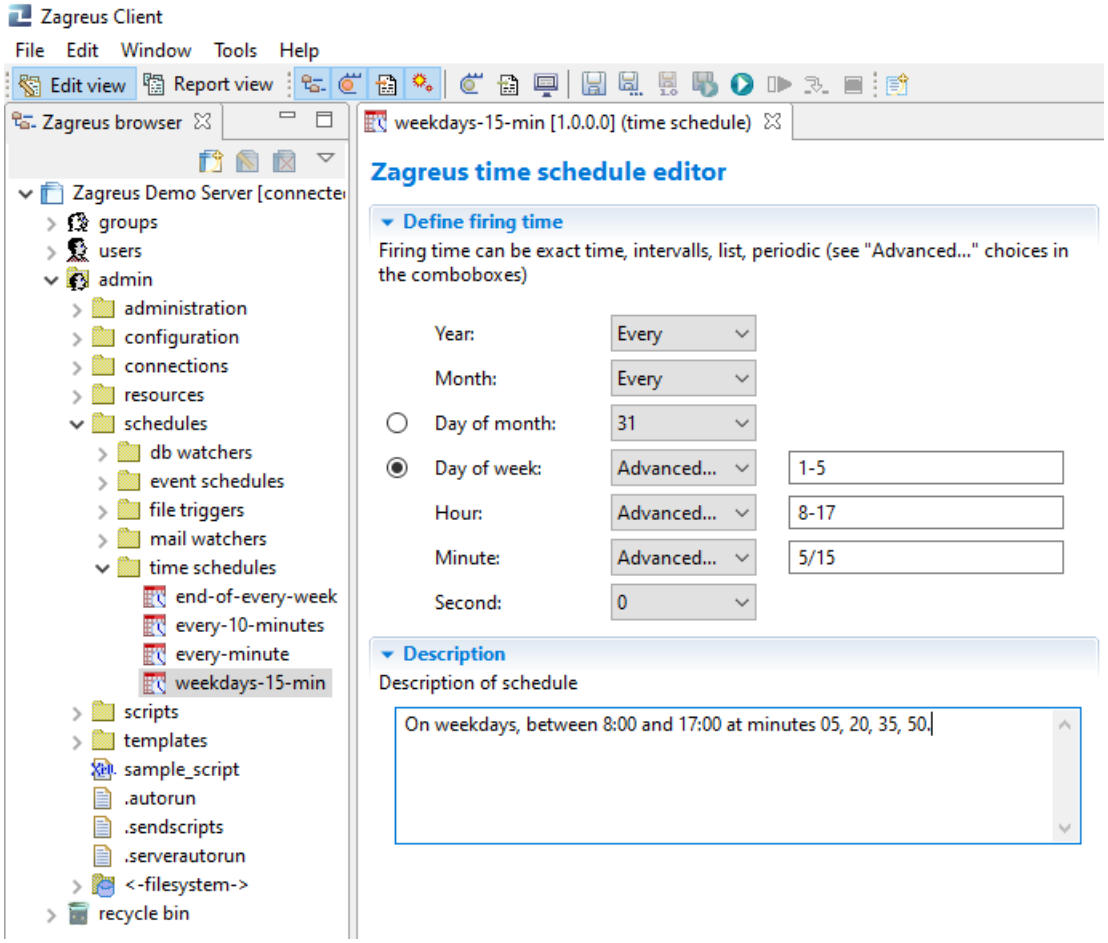


Figure 7 – A time schedule with advanced day, hour, and minute setting

15.5.3 Mail watcher

By using a mail watcher, the user can execute scripts depending on the results of the fetched messages from an e-mail inbox. Checking the mailbox content occurs periodically (relying on a time schedule, see → Time schedule), evaluating a specified logical condition based on the filters of the mail watcher. If this condition is evaluated as true, the mail watcher will be triggered, executing the subscribed script(s). Technically, a mail watcher can be considered as an additional filter inserted between a time schedule and triggering the script execution, where the filter is based on the content of an e-mail inbox. There is also a built-in *counter* value for fine-tuning the behaviour of the watcher.

15.5.3.1 Define filter section

The *filter* defines the condition which is evaluated for the contents of the specified e-mail inbox, accessible under the “Define filter” expandable section of the Zagreus

mail watcher editor. *Figure 8.* shows a sample filter defined in the Zagreus Client application, filtering on the subject and of the name of the attachments of the last 20 e-mails.

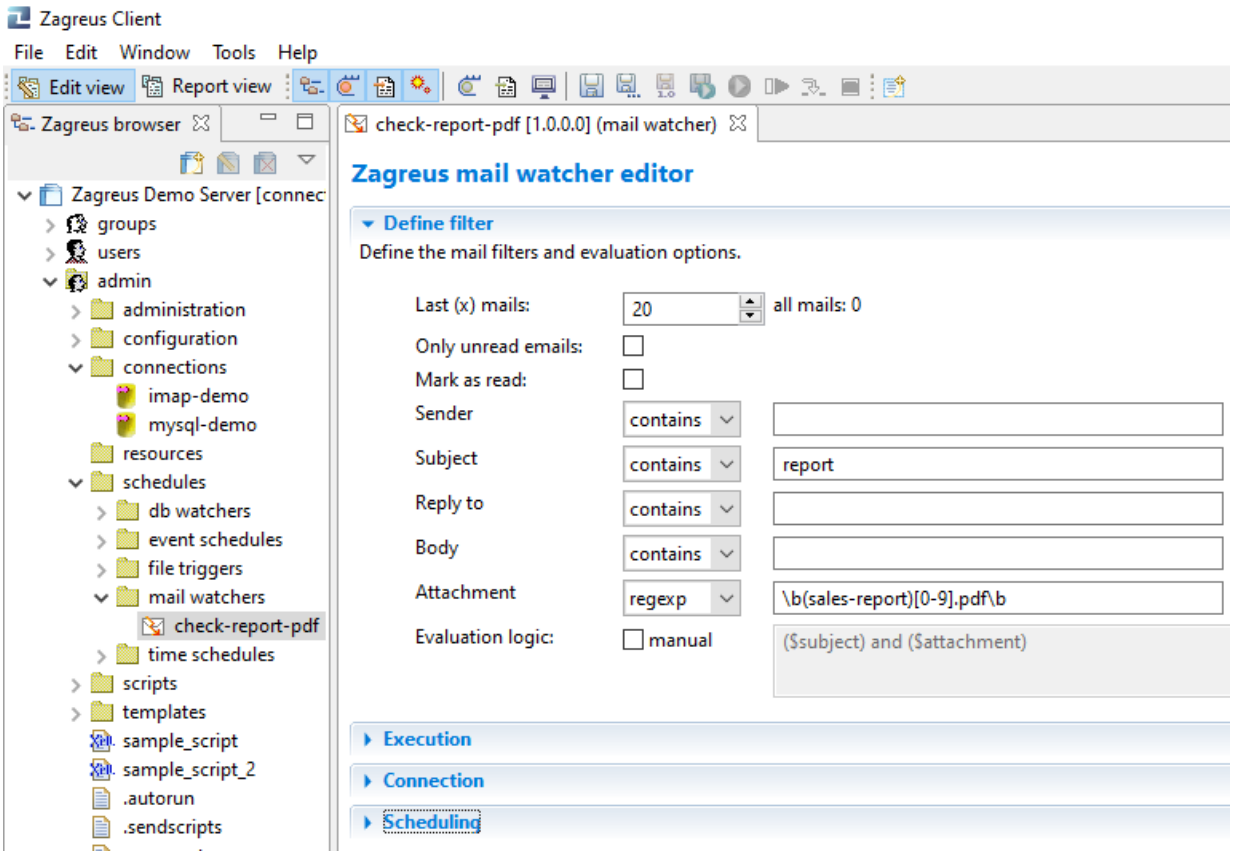


Figure 8 – Creating a new mail watcher resource

The following options control which e-mails are examined, i.e. they refer to the number and the status of the examined e-mails:

- *Last (x) mails*
  The latest *x* e-mails to be filtered. If it is set to 0, all e-mails are being examined.

- *Only unread emails*
  If this setting is checked, only unread e-mails will be examined. If this setting is unchecked, both read and unread e-mails are examined.

- *Mark as read*
  If this setting is checked, the mail watcher will set the status of *unread* e-mail messages to *read* after examination.

The next group of options define the condition which is evaluated for the e-mails, each option corresponding to a specific property of the e-mail. The following properties can be referred to:

- *Sender*: the sender of the e-mail
- *Subject*: the subject of the e-mail
- *Reply to*: the reply-to field of the e-mail
- *Body*: the body of the e-mail
- *Attachment*: the name of attached files of the e-mail

For all five properties, the user can choose from two behaviours: *"contains"* and *"regexp"*. "*contains"* means that the text element defined for the given property is expected to be a part of the given property of the e-mail, while with the *"regexp"* option one can define a regular expression, which is matched for the particular e-mail property. For example, by setting *"Sender"* to *"contains"* and setting the corresponding value to *"sender@test"*, the watcher will trigger the associated scripts when the sender property of an examined e-mail contains the specified value, for example *"testsender@testdomain.com"*. In the *"regexp"* behaviour, regular expressions can be used with tags like [A-Z, a-z]*. For example, the regular expression *"[Ss]ender[0-9].*"* will match for any property values starting with the parts "sender" or "Sender", followed by a digit, and ending with any string.

Finally, one might specify the relation between the examined properties. By default, all fields which were not left empty in the mail watcher have to be satisfied, being equivalent to an AND relation between them. By setting the *"manual"* "Evaluation logic" checkbox, more complex relations can be defined. The condition parts can be referred to by adding the "$" character; possible values are: `$sender`, `$subject`, `$replyto`, `$body`, `$attachment`. Over these parts, AND, OR and NOT logical operators can be used to define their relations, e.g. the value "`($sender) AND ($subject) AND NOT ($replyto)`" means that the conditions defined for the *"Sender"* and the *"Subject"* properties have to be true, while the condition defined for the *"Reply to"* property has to be false at the same time for the mail watcher to trigger the associated scripts.

15.5.3.2 Execution section

On the next expandable section ("Execution"), additional script execution options can be set, see *Figure 9*.

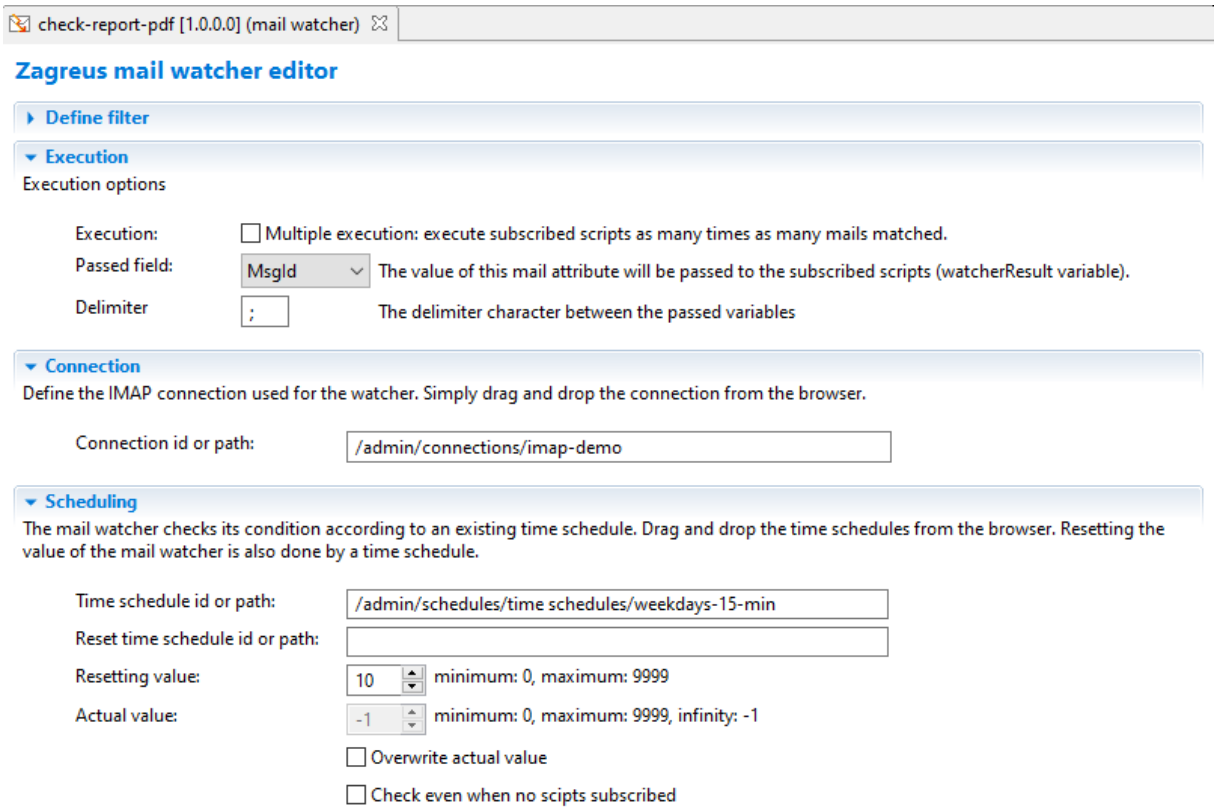


Figure 9 – Mail watcher *Execution*, *Connection* and *Scheduling* expandable sections. In this particular example, an IMAP connection is defined and a time schedule of 15 minutes at each weekday is used for scheduling to check the e-mail inbox.

The following execution options can be changed:

- *Execution*
  If the *"Multiple execution"* option is set, the mail watcher will be activated independently for each e-mail for which the filter matches; otherwise, it will be activeted only once (in the case that the filter matches for at least one e-mail).

- *Passed field*
  This field grants the option that the script(s) which execution will be triggered are provided information about the corresponding e-mail. For this, the value of the `$watcherResult` variable will be set accordingly; the dropdown box shows the property of the e-mail which will be stored in this variable. Currently only the option *"MsgId"* is supported.

- *Delimiter*
  If the *"Multiple execution"* option was not set, the mail watcher will fire only once, regardless of the number of the e-mails satisfying the specified filter. In this case, the value of the `$watcherResult` variable will contain the corresponding

information (e.g. sender) for all these e-mails, separated by some delimiter (e.g. ";"), which delimiter can be set here. It can be at most two characters long.

15.5.3.3 Connection section

The mail connection (i.e. a Zagreus recource with connection type, see also → Connections), which the mail watcher uses to check the incoming e-mails, can be set in the *"Connection"* expandable section. It can either be IMAP or POP3. It can be drag-and-dropped from the *Zagreus browser* (see → Drag-and-drop operations), or the corresponding resource ID or resource path can be copied from e.g. the *Resource information* page (see → Resource information).

15.5.3.4 Scheduling section

As we mentioned, the mail watcher can be considered as a further filter between a time schedule and script execution, as it evaluates its e-mail-based condition when a given time schedule fires. This time schedule resource (see also → Time schedule) can be set in the *Scheduling* expandable section along with further scheduling options.

These further options rely on the concept of the *counter*. To help fine-tuning the behaviour of the mail watcher, a maximum value can be defined at which the mail watcher triggers the subscribed scripts within a given time interval. This is implemented by assigning a *counter* value for a mail watcher; this value is decreased by one each time the mail watcher runs (even if there is no matching e-mail), and if it falls to zero, the mail watcher becomes inactive. The *counter* value can be reset by another time schedule. The -1 value of the *counter* means that this setting is inactive, and the mail watcher might trigger the subscribed scripts infinitely.

The following options are available in the *Scheduling* expandable section of the mail watcher editor in the Zagreus Client application:

- *Time schedule id or path*
  The mail watcher checks its condition according to the time schedule specified here. This field is mandatory.

- *Reset time schedule id or path*
  The time schedule which resets the *counter* value of the mail watcher resource can be set here.

- *Resetting value*
  The value which is set for the *counter* value when resetting. It can be a non-negative integer between 0 and 9999.

- *Initial value*
  The initial value of the *counter*. Accessible in the Zagreus Mail watcher editor only when the mail watcher is created; afterwards, the *initial value* will be replaced by the *actual* counter value.

- *Actual value*
  The actual value of the *counter*. By default, changing this value is inactive in the Zagreus Mail watcher editor. It can be an integer between 0 and 9999. The value -1 has a special meaning: in this case, there is no limitation on the number of triggering the execution of the subscribed scripts.

- *Overwrite actual value*
  To modify the actual value in the Mail watcher editor, this setting must be checked, otherwise the *actual value* input field will behave read-only. This is a safety feature to avoid unintentional changes in the *counter* value.

- *Check even when no script is subscribed*
  If this setting is checked, the mail watcher will run even if there are no scripts subscribed to it.

15.5.3.5 Evaluate watcher condition

The user can evaluate the condition of the mail watcher without affecting the actual counter value. This step can be very useful during the development and set-up period for the watcher configuration, i.e. the user can check if the mail connection is alive, and if the condition filter works properly.

The user needs to select the *Evaluate watcher condition…* menu item from the context menu in the Zagreus Browser window, see → Operations for event-type resources. This opens the *Evaluation results* dialog, which shows the result of condition evaluation. *Figure 10.* shows the result of a mail watcher with two messages that satisfied the filter condition (so the condition was evaluated as TRUE).

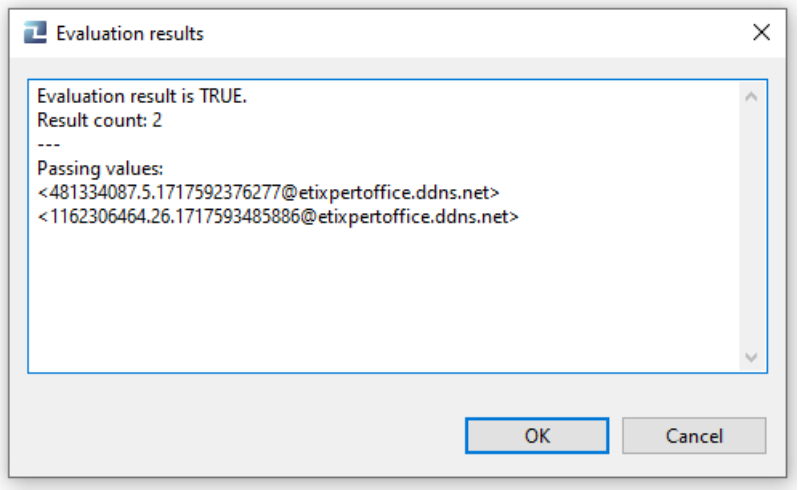


Figure 10 – The *Evaluation results* dialog box for a mail watcher

15.5.3.6 Server-side configuration

Most of the settings of a mail watcher are done on client side, in the Zagreus Client application. However, a property affecting general watcher behaviour is configured on the server side, where the way of evaluating the watcher conditions can be set by the *watcher.counter.policy* setting, see → Trigger and watcher properties. The possible values are:

- *evaluate*: the counter is decreased by 1 each time the condition is evaluated
- *condition_true*: the counter is decreased by 1 if the condition is evaluated to true. This is the default value of this setting.
- *script_run*: the counter is decreased by 1 if the condition is evaluated to true and any script is executed

⚠ ***Warning:*** watcher.counter.policy *is a general setting, which affects the behaviour of all of the watchers, i.e. both database watchers and mail watchers.*

15.5.4 Database watcher

The database watcher is technically very similar to the mail watcher. In the case of a database watcher, the triggering (and therefore the execution of the subscribed scripts) depends on the result of an SQL query. The query is run against a database server periodically (relying on a time schedule, see → Time schedule). Technically, a database watcher can be considered as an additional filter inserted between a time

schedule and triggering the script execution, where the filter is based on the content of some database table. There is also a built-in *counter* value for fine-tuning the behaviour of the watcher.

15.5.4.1 Define filter section

The filter of the database watcher can be set in the *Define filter* expandable section, see *Figure 11*. The SQL query has to be set in the *SQL query* field, and it is mandatory.

The *Condition type* radio button is used to set the interpretation of the result of the SQL query. The two options are:

- *Lines*
  By selecting this option, the database watcher examines the number of lines returned by the SQL query. If this value is larger than 0, the condition will be evaluated to true and the database watcher will trigger the execution of the subscribed scripts.

- *Scalar*
  By selecting this option, the database watcher examines the *value* of the result returned by the SQL query. If this value is not 0, the condition will be evaluated to true and the database watcher will trigger the execution of the subscribed scripts. A straightforward use case for this option is a '`SELECT COUNT(*) …`' SQL query.

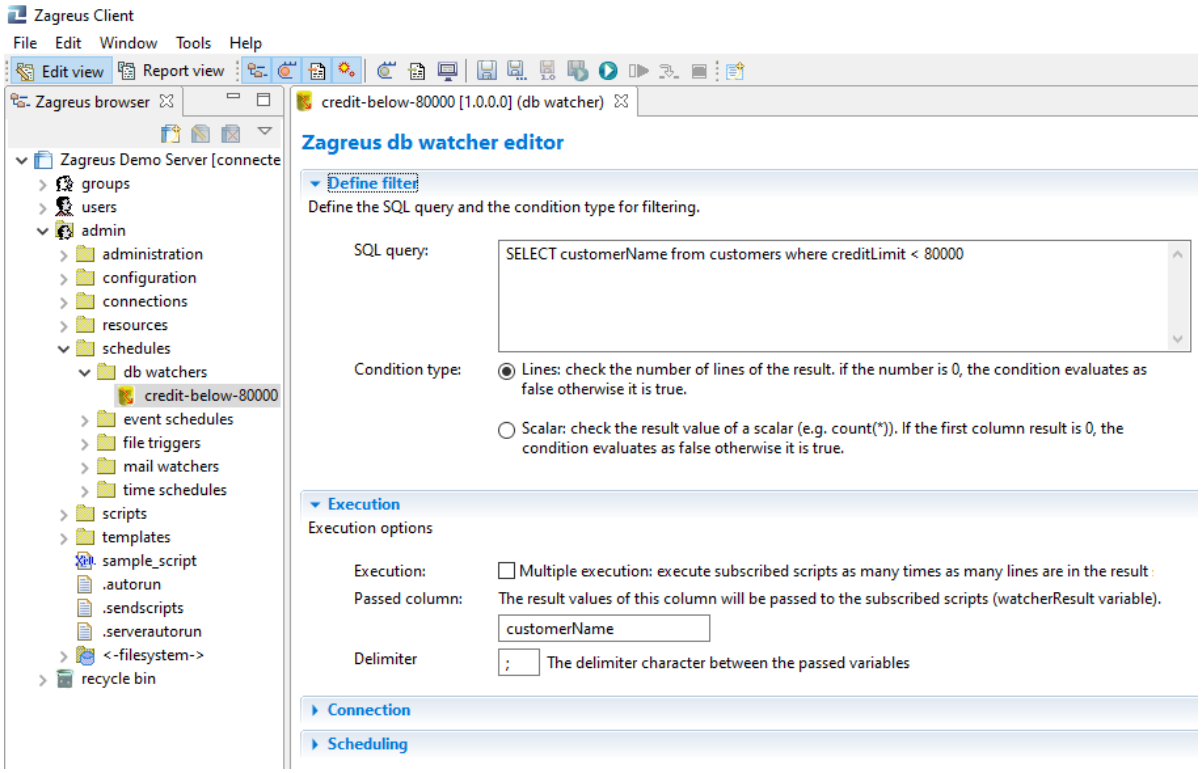


Figure 11 – Creating new db watcher resource. *Define filter* and *Execution* expandable sections

15.5.4.2 Execution section

On the next expandable section (*Execution*), additional script execution options can be set. The following execution options can be changed:

- *Execution*
  If the *Multiple execution* option is set, the database watcher will be activated independently for each line returned by the SQL query. Each executed script receives one line of result via the `${watcherResult}` variable. If this checkbox is not checked, the database watcher will trigger script execution only once.

- *Passed field*
  This field grants the option that the script(s) which execution will be triggered are provided information about the corresponding database content. For this, the value of the `$watcherResult` variable will be set to the result value(s) of the column defined in this *"Passed field"* textbox.

- *Delimiter*
  If the *Multiple execution* option was not set, the database watcher will fire only once, regardless of the number of lines returned by the SQL query. In this case, the value of the `$watcherResult` variable will contain the value of the specified

column for all the returned rows, separated by some delimiter (e.g. ";"), which delimiter can be set here. It can be at most two characters long.

> ***Info:*** *The value of only one result column can be passed to the script. This column must be defined in the* Passed column *setting.*

15.5.4.3 Connection section

The database connection (i.e. a Zagreus recource with connection type, see also → Connections), which the database watcher uses to check the database content, can be set in the *Connection* expandable section (see the upper half of *Figure 12*.). It can be drag-and-dropped from the *Zagreus browser*, or the corresponding resource ID or resource path can be copied from e.g. the *Resource information* page (see → Resource information).

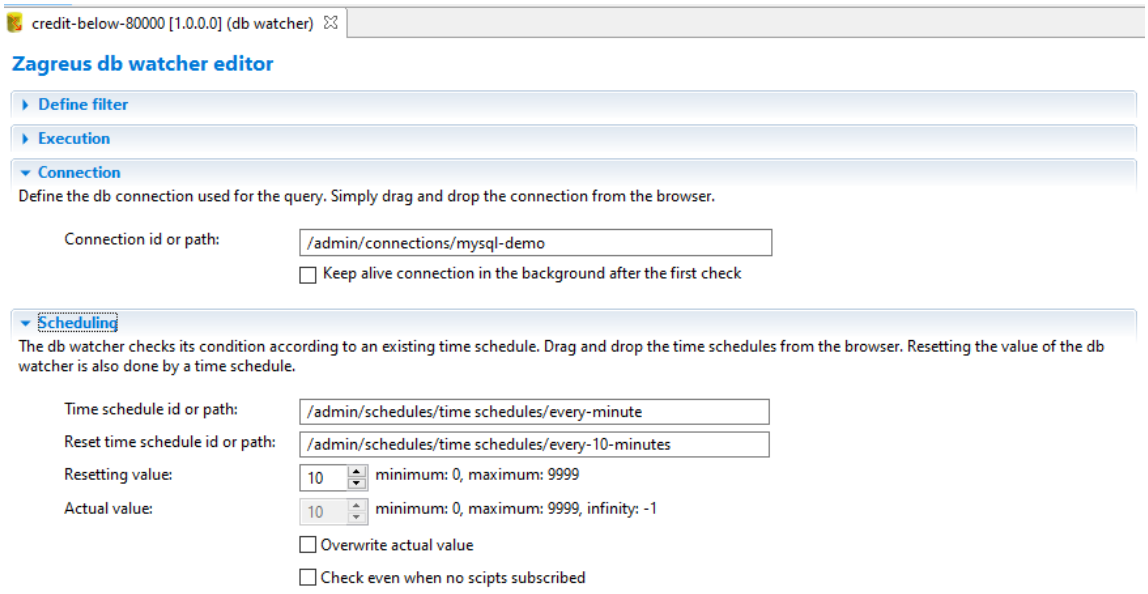


Figure 12 – Creating a new database watcher resource. *Connection* and *Scheduling* expandable sections

The other setting in the *Connection* tab (*Keep alive connection in the background after the first check*) refers to a specific connection keep-alive setting. When the database watcher runs for the first time, the connection used by the watcher is stored in a connection pool and re-used when necessary. If a connection is used frequently, it is worth keeping this connection open to save server resources, which behaviour is enabled by setting this option. If this option is not set, the database connection opens and closes each time when the database watcher is executed.

15.5.4.4 Scheduling section

As we mentioned, the database watcher can be considered as a further filter between a time schedule and script execution, as it evaluates its database-related condition when a given time schedule fires. This time schedule resource (see also → Time schedule) can be set in the *Scheduling* expandable section (see the lower half of *Figure 12*.) along with further scheduling options.

These further options rely on the concept of the *counter*. To help fine-tuning the behaviour of the database watcher, a maximum value can be defined at which the database watcher triggers the subscribed scripts within a given time interval. This is implemented by assigning a *counter* value for a database watcher; this value is decreased by one each time the mail watcher runs (even if there is no matching database content), and if it falls to zero, the database watcher becomes inactive. The *counter* value can be reset by another time schedule. The -1 value of the *counter* means that this setting is inactive, and the database watcher might trigger the subscribed scripts infinitely.

The following options are available in the *Scheduling* expandable section of the database watcher editor in the Zagreus Client application:

- *Time schedule id or path*
  The database watcher checks its condition according to the time schedule specified here. This field is mandatory.

- *Reset time schedule id or path*
  The time schedule which resets the *counter* value of the database watcher resource can be set here.

- *Resetting value*
  The value which is set for the *counter* value when resetting. It can be a non-negative integer between 0 and 9999.

- *Initial value*
  The initial value of the *counter*. Accessible in the Zagreus Database watcher editor only when the database watcher is created; afterwards, the *initial value* will be replaced by the *actual* counter value.

- *Actual value*
  The actual value of the *counter*. By default, changing this value is inactive in the Zagreus Database watcher editor. It can be an integer between 0 and 9999. The value -1 has a special meaning: in this case, there is no limitation on the number of triggering the execution of the subscribed scripts.

- *Overwrite actual value*
  To modify the actual value in the Database watcher editor, this setting must be checked, otherwise the *actual value* input field will behave read-only. This is a safety feature to avoid unintentional changes in the *counter* value.

- *Check even when no script is subscribed*
  If this setting is checked, the database watcher will run even if there are no scripts subscribed to it.

15.5.4.5 Evaluate watcher condition

The user can evaluate the condition of the database watcher without affecting the actual counter value. This step can be very useful during the development and set-up period for the watcher configuration, i.e. the user can check if the database connection is alive, and if the condition works properly.

The user needs to select the *Evaluate watcher condition…* menu item from the context menu in the Zagreus Browser window, see → Operations for event-type resources. This opens the *Evaluation results* dialog, which shows the result of condition evaluation. *Figure 13.* shows the result of a watcher with a *lines* condition type, listing all the values which were returned by the SQL query (if any). In this example, the condition is evaluated as TRUE since the query returned five values.

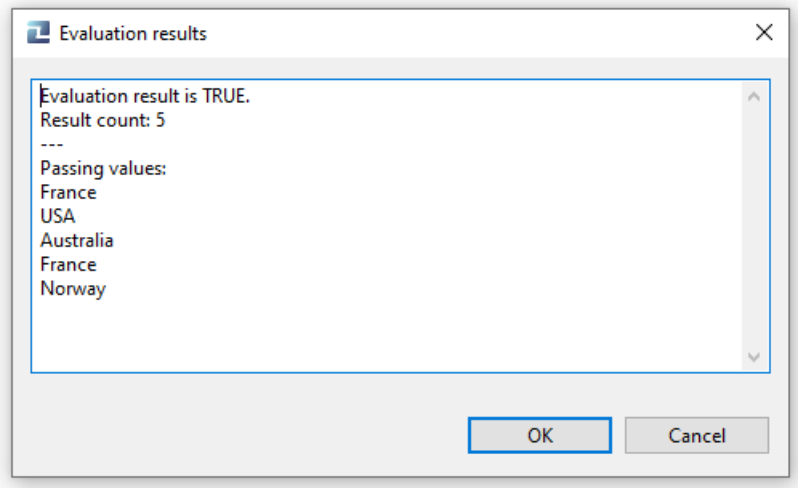


Figure 13 – The *Evaluation results* dialog box for *lines* condition type

Figure 14. shows the result of a watcher with a *scalar* condition type, showing the numeric return value of the SQL query (i.e. 30). In this example, the condition is evaluated as TRUE since the returned value is not zero.

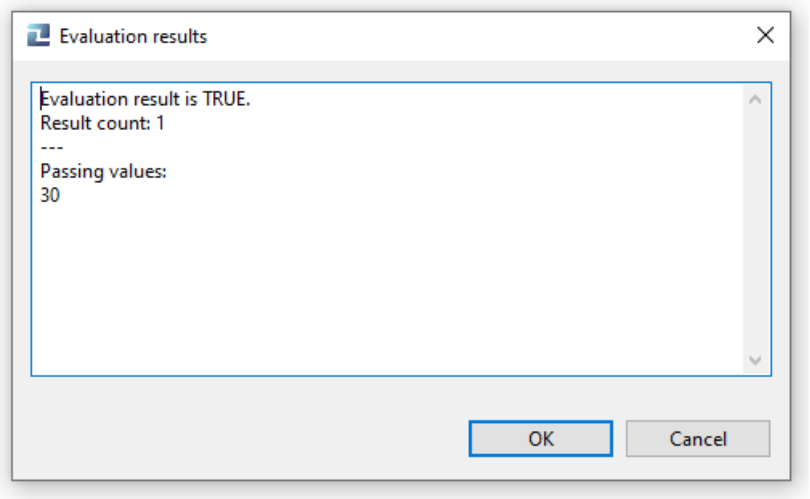


Figure 14 – The *Evaluation results* dialog box for *scalar* condition type

15.5.4.6 Server-side configuration

Most of the settings of a mail watcher are done on client side, in the Zagreus Client application. However, a property affecting general watcher behaviour is configured on the server side, where the way of evaluating the watcher conditions can be set by the *watcher.counter.policy* setting, see → Trigger and watcher properties. The possible values are:

- *evaluate*: the counter is decreased by 1 each time the condition is evaluated
- *condition_true*: the counter is decreased by 1 if the condition is evaluated to true. This is the default value of this setting.

- *script_run*: the counter is decreased by 1 if the condition is evaluated to true and any script is executed

> ⚠ ***Warning:*** watcher.counter.policy *is a general setting which affects the behaviour of all of the watchers, i.e. both database watchers and mail watchers.*

15.5.5 File trigger

File triggers allow the triggering of the execution of subscribed scripts when a specific file event (e.g. file modification or file deletion) occurs in the Zagreus filesystem.

The number of file trigger activations is related to the number of files which satisfy the condition defined in the file trigger. For example, when a particular file trigger checks the deletion of files in a specific folder, and two files are deleted from the folder, the subscribed scripts will be executed twice. File triggers can be created and edited in the *Zagreus File trigger editor*, accessible via the menu path *File / New Resource / File trigger*.

15.5.5.1 Define folder and filename pattern section

The filter of the file trigger can be set in the *Define folder and filename pattern* expandable section, see the upper half of *Figure 15*. The folder which will be watched can be set in the *Watch folder* field, and it is mandatory. Its content must be a Zagreus filesystem folder. The folder can also be drag-and-dropped from the Browser window of the Zagreus Client application, see → Zagreus Client.

The *Filename pattern* radio button is used to set the file mask for the file trigger filter. The two options are:

- *All files*
  The file trigger will fire for all the files in the specified watch folder.

- *textbox*
  By selecting this option, a filename filter can be specified in the textbox. In this filename mask, the wildcard character "`*`" can be used, for example: "`2021*.xls`".
  The checkbox below this option controls whether this filename mask should be interpreted as a regular expression or as a standard filename. For example, the regular expression "`[A-Z,a-z]+`" stands for file names which contain only uppercase and lowercase letters.

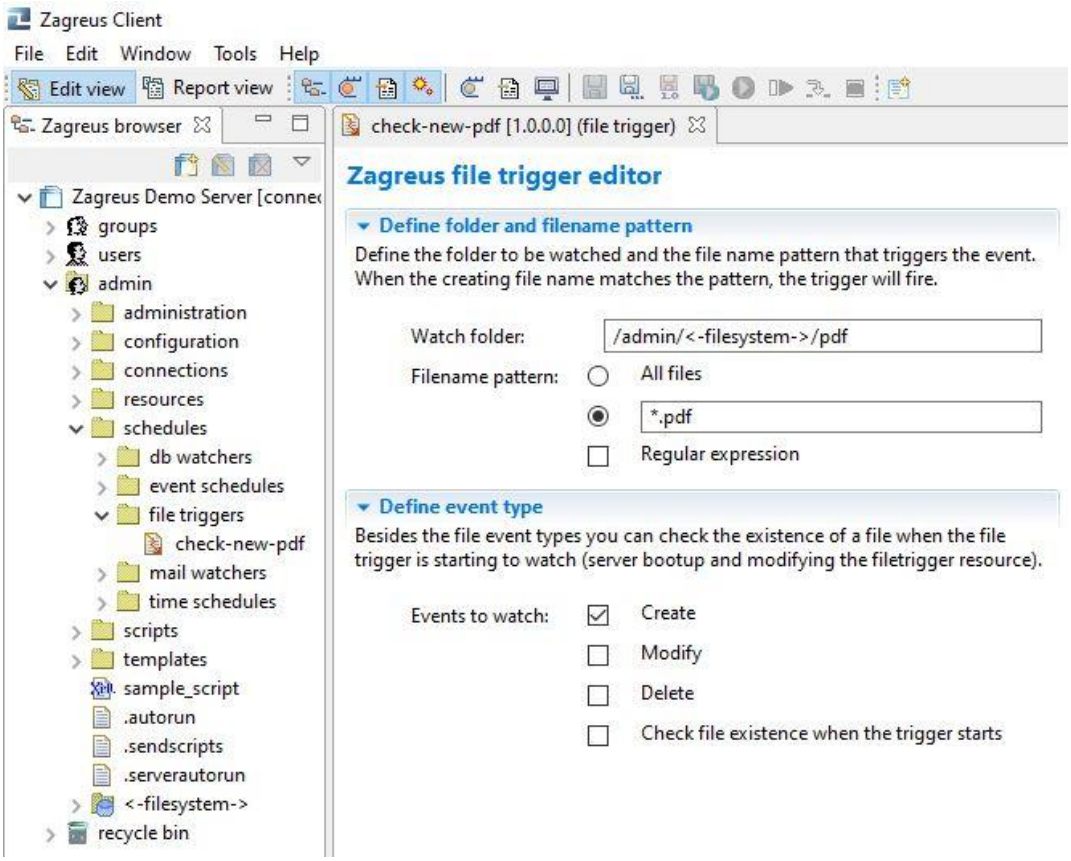


Figure 15 – Creating a new file trigger. This trigger fires on the creation of a file with .pdf extension

15.5.5.2 Define event type section

There are three types of file events which might trigger a script execution:

- *Create*: file creation
- *Modify*: file modification
- *Delete*: file deletion

These values can be selected in the *"Define event type"* expandable section (see the lower half of *Figure 15.*), it is possible to check only one or multiple events. By setting the *Check files existence when the trigger starts* checkbox, the existence of the specified file(s) will be checked by the file trigger when it starts to run – i.e. when the Zagreus server starts, or after saving the modified or newly created file trigger resource. If none of these check boxes is set, the file trigger remains active, but – as no file event may satisfy the specified condition – the execution of the subscribed scripts will not be triggered.

> ***Info:*** *The events (script creation, modification and deletion) are not watched by Zagreus, but they are detected by the given operation system (OS), and forwarded to Zagreus.*

15.5.5.3 Server-side configuration

Some behaviour setting of file triggers can be configured on the server side, see → Trigger and watcher properties. There, the setting *filetrigger.double.trigger.limit* controls the handling of a specific OS tendency. There might be unwanted cases that sometimes two events are created for one file event, while the user understandably expects only one script execution triggering. For example, modifying a file creates two events in the Windows OS. To avoid double script execution in such cases, and to filter these double file events (referring to the same file with a too short time difference between the two events), we can use this *filetrigger.double.trigger.limit* setting, which is used to define the minimal time difference between two events (in milliseconds). For example, if this setting is set to 250, a second event referring to the same file only after 220 milliseconds after the first one will be ignored; therefore, the file trigger will be activated by only the first one. However, a setting of 200 milliseconds would result in double triggering, and therefore double script executions. The default value for this setting is 500.

15.5.6 Special events

There are also special events in Zagreus which may trigger automatic script execution. Out of the four special such events, two are script options, while the other two are special files located in the Zagreus local database.

15.5.6.1 Script execution by script options

There are two special execution options: *execute_script_on_error* and *execute_script_on_cancel.* If they are set for a given script, they can initiate the execution of another script. If the execution of the actual script (i.e. which has any of these options set) ends with *error* on *cancelled* status (for job statuses, see → Job lifecycle), the execution of the script specified in the corresponding option will be triggered. The value of these options can be either a script ID or path.

Regarding the option *execute_script_on_cancel*, there is another script option to fine-tune its behaviour: the source of cancellation can also be specified. This option is an additional filter condition, and can be set with the option *execute_script_on_cancel_source*, with the possible values being *gui*, *monitor*, *zs* and *server*. It is also possible to specify multiple values separated by commas. *Figure 16.* shows an example setting for these options: in this particular example, if the execution of the given script (i.e. `sample_script`) ends with status *error*, the script `error-handling` will be executed. Similarly, if the execution of `sample_script` ends with status *cancelled*, where the source of cancellation was the Zagreus Client or the Zagreus Monitor application, the execution of the script `cancel-handling` will be triggered. However, if the source of cancellation was a `zs:cancel` action, no script execution will be triggered, as that would be handled by the *zs* value of the *execute_script_on_cancel_source* option.

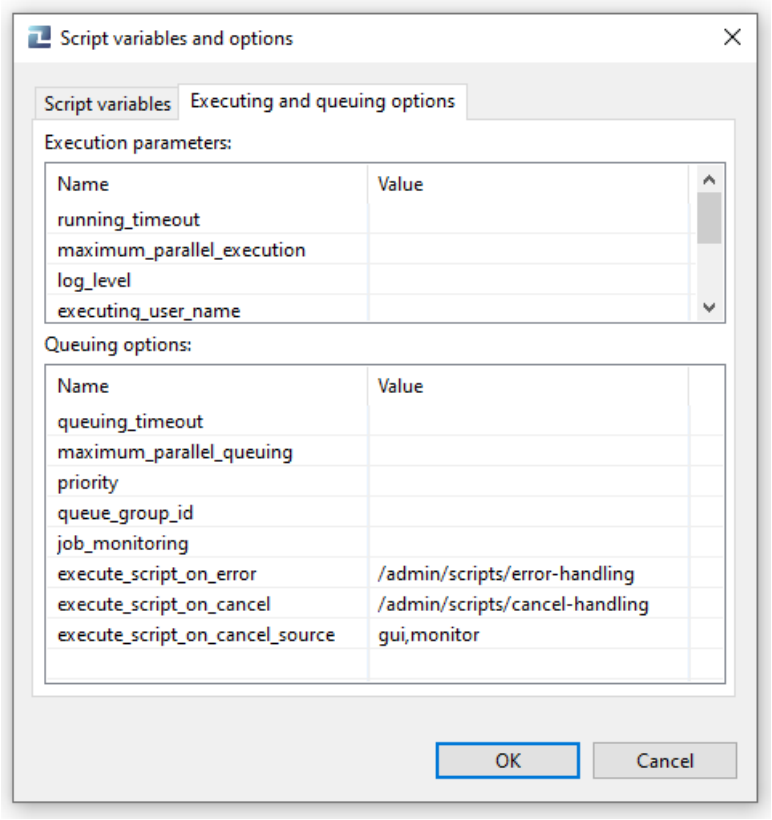


Figure 16 – Setting the `execute_script_on_error`, `execute_script_on_cancel` and `execute_script_on_cancel_source` options

15.5.6.2 Script execution by *autorun* configuration files

The special file `.autorun` is a special text file in the home folder of each Zagreus user. The execution of scripts (specified either by their resource ID or their path) listed in this file will be triggered right after the given user logs in to the Zagreus Server via the Zagreus Client application. In this file, one line defines one script to execute.

Administrator users can create another special file named `.serverautorun` in their Zagreus home folder. The execution of scripts listed in this file are will be triggered right after a Zagreus server start-up event. The format of this file is the same as that of the `.autorun` file.

15.5.7 Administrative tools for event-type resources

There are useful administrative tools built in the Zagreus Client applications to monitor and handle the behaviour of event-type. Additionally, there is the Zagreus Monitor application, also designed to monitor script executions. By these tools the users can check the status of automating components, monitor which scripts are started by different types of event-type resources, while administrators can also start and stop the operation of triggers, watchers and schedulers.

15.5.7.1 Monitoring watchers and triggers window

In the Zagreus Client, in the *Monitoring watchers and triggers* window, the administrator users can monitor the behaviour of the watchers and triggers on the given Zagreus server. This window can be opened by right-clicking on the server definition node in the Browser window of the Zagreus, then selecting the *Administrator options / Monitor watchers, triggers...* options from the context menu (see *Figure 17.*).

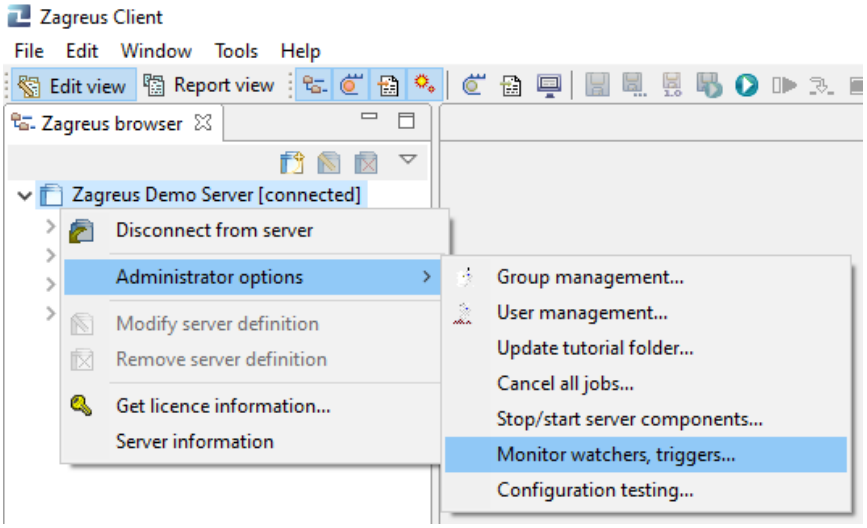


Figure 17 – Opening the *Monitor watchers, triggers...* window

The opening *Monitoring watchers and triggers* window consists of three tabs: *Watchers*, *File Trigger* and *DB Connection Pool*. The mail and database watchers are listed in the *Watchers* tab, while file trigger resources can be viewed on the *File Trigger* tab.

In the *Watchers* tab, the user can check the following details of the mail and database watchers (see *Figure 18.*):

- *Trigger*: the path of the watcher
- *Type*: the type of the watcher (i.e. *mail watcher* or *db watcher*)
- *Connection*: the path of the connection resource which the watcher uses
- *Schedule*: the path of the time schedule which the watcher uses
- *Reset schedule*: the path of the time schedule which the watcher uses for resetting its *counter* value
- *Last evaluation*: the date and time when the watcher was last evaluated
- *Last execution*: the date and time when the watcher last triggered script execution
- Actual value: the actual value of the *counter*

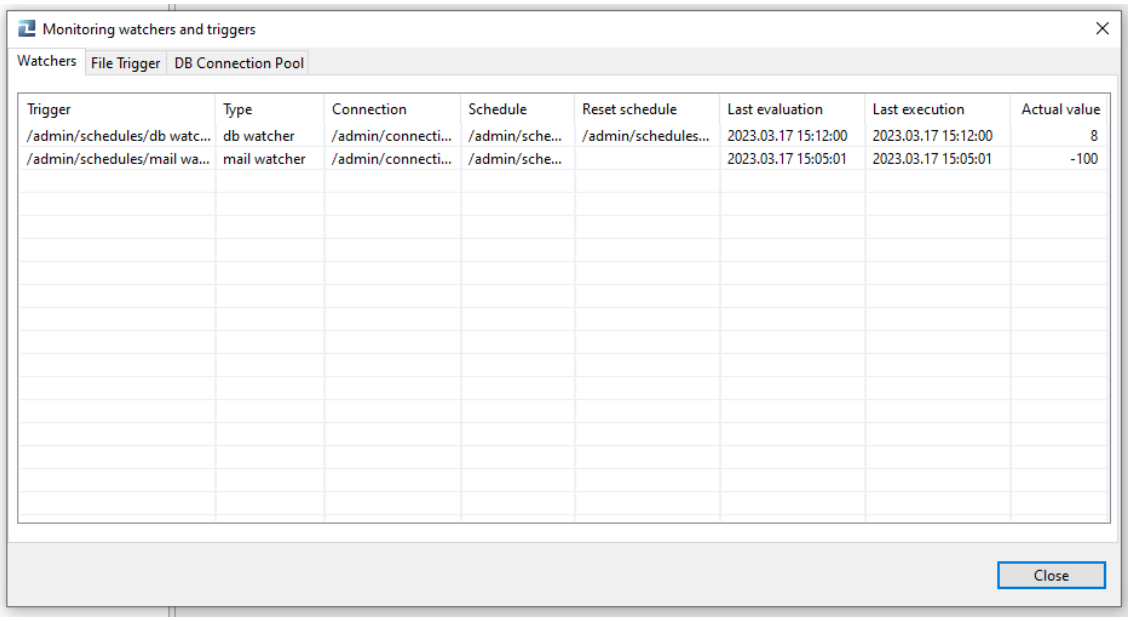


Figure 18 – *Monitoring watchers, triggers...* window

In the *File trigger* tab, the following details of the file trigger resources are listed:

- *Trigger*: the path of the trigger
- *Folder path*: the path of the folder the file trigger uses in its filter (see → Define folder and filename pattern section)
- *Last triggered*: the date and time when the file trigger was last triggered
- *Trigger filename*: the filename which caused the file trigger to trigger last
- *Event type*: the type of event which causes the file trigger to trigger last
- *Currently active*: whether the file trigger is currently active (*yes* or *no*)

The *DB Connection pool* tab lists the database connections which are used by database watcher resources having the setting *Keep alive connections in the background...* checked, i.e. they are open and kept in memory (see → Connection section). The following details are listed:

- *Connection*: the path of the connection resource which the watcher uses
- *Driver*: the name of the SQL driver which is used by this open connection
- *Host*: the host to which this DB connection is open
- *User*: the user which is used to log in by this DB connection

15.5.7.2 Stop / start server components

With this option in the Zagreus Client application, administrator users can turn on and off the Scheduler module (see → Quartz scheduler), the execution of watchers and triggers in a Zagreus server. This menu item can be found by right-clicking on the server connection node in the Zagreus browser window of the Zagreus Client, and selecting the *Administrative options / Stop/start server components* option in the context menu (see *Figure 19.*).

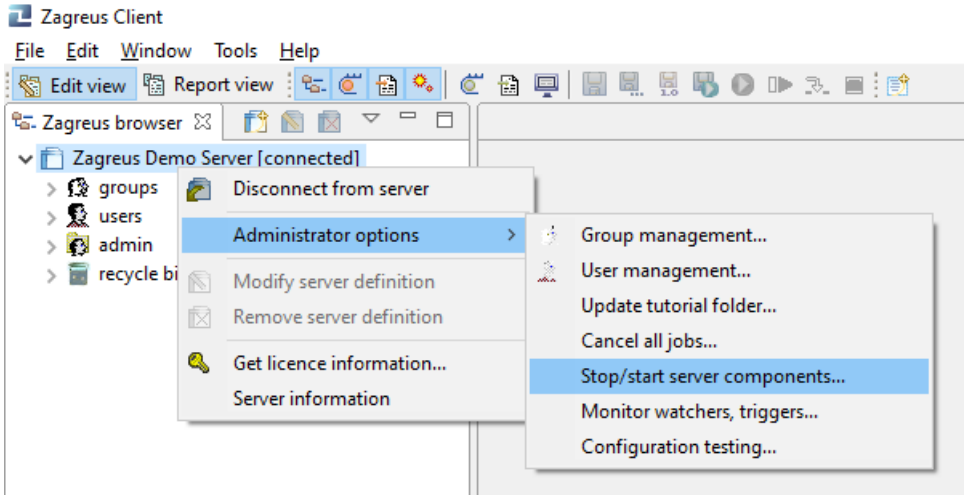


Figure 19 - Opening the Stop/start server components window

In the *Server Component Control* dialog, the user can enable or disable the selected component by right-clicking on it (see *Figure 20.*).

For further details on different server components, see → Components.

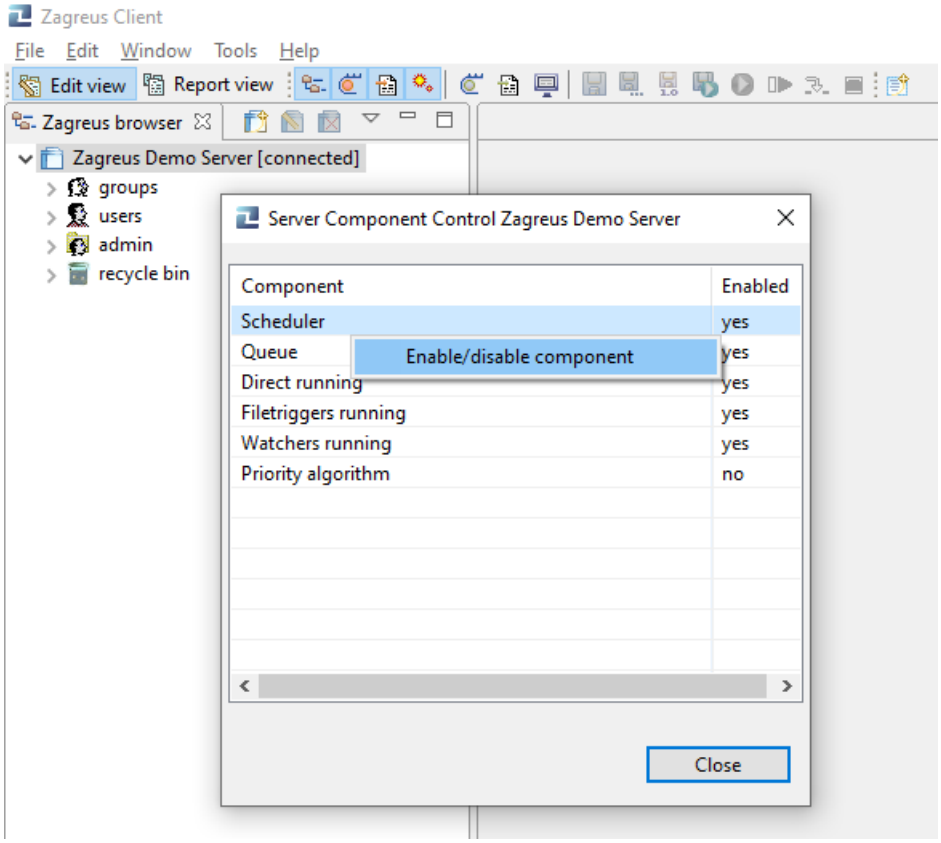


Figure 20 – Enable / disable server components

> ⚠ ***Warning:*** *If the scheduler module is disabled, the watchers and file triggers will not fire either, since they rely on time schedule resources.*

15.5.7.3 Administrative tools in Zagreus Monitor

The Zagreus Monitor application (see → Zagreus Monitor) is primarily designed to allow the administrators to monitor the job executions performed on the given Zagreus Server. It has functionality to filter scripts by different criteria like execution mode, status, etc. The events are displayed in a timeline, see *Figure 21*.

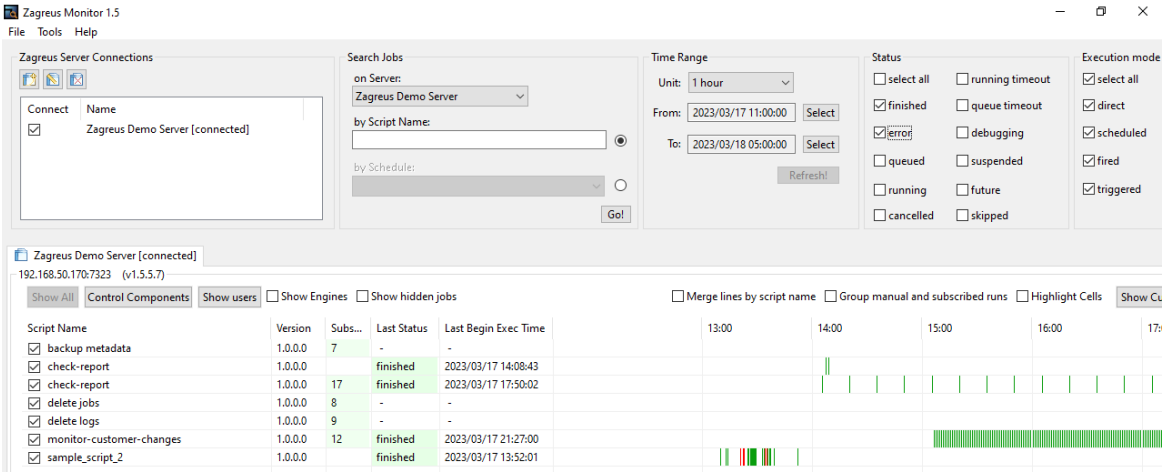


Figure 21 – A sample Zagreus Monitor window, showing jobs regardless of execution type

If the user restrains the execution modes to keep only the *scheduled* and *triggered* items, scripts with automated execution can be monitored, see *Figure 22*.

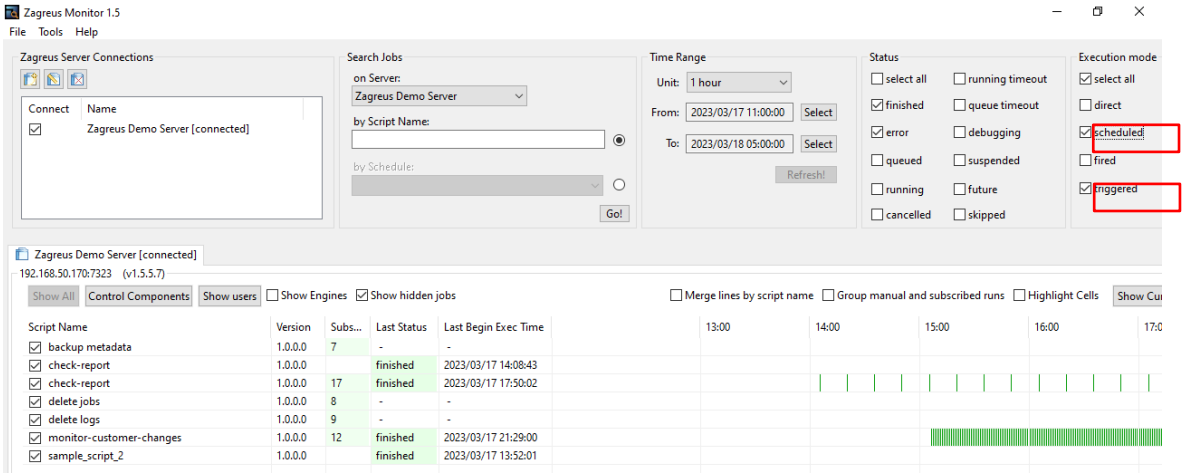


Figure 22 – A sample Zagreus Monitor window, showing only *scheduled* and *triggered* scipt executions

15.6 Manual script execution

Besides automated script execution, Zagreus also provides multiple ways of executing scripts manually. It is possible to initiate script execution from the Zagreus Client, from another script (by the `zs:runscript` action), while remote users can also initiate script execution by using Zagreus command line tools or the Zagreus HTML application. Furthermore, scripts can be executed from external systems – such as database environments – with SOAP requests.

15.6.1 Execution in the Zagreus Client

To manually execute a script in the Zagreus Client, right-click on the script in the Browser window of the Zagreus Client, and choose the *Run script* menu item from the context menu (see *Figure 30.*). By selecting multiple scripts, all of them can be executed at the same time by selecting the *Run script* item from the context menu after right-clicking on the selection.

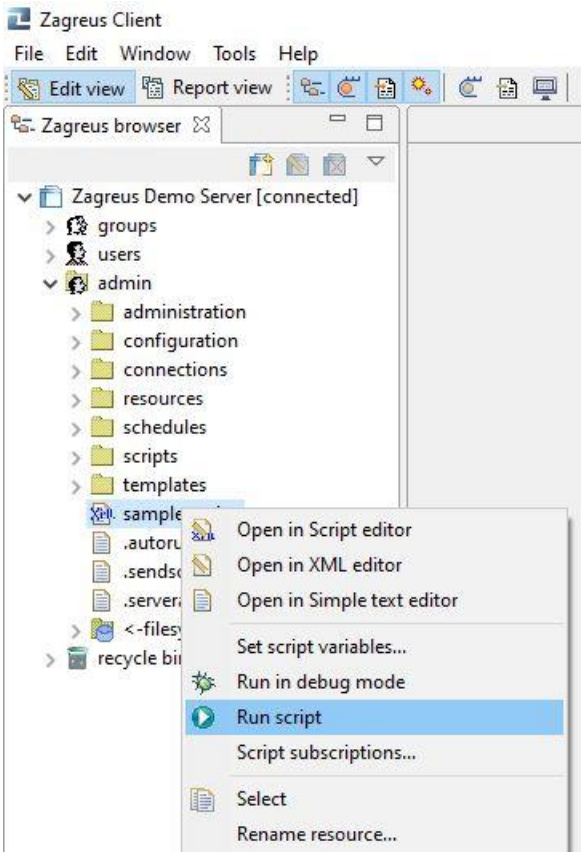


Figure 30 – Executing a script from the context menu in the Browser window of the Zagreus Client

When a script is opened for editing in the Zagreus Client, there are two icons in the top icon bar which serve to initiate script execution manually: *Save and run resource* and *Run script* (see *Figure 31.*). Scripts have to be saved before execution; clearly, the *Save and run resource* icon first saves the active script, and executes it afterwards.

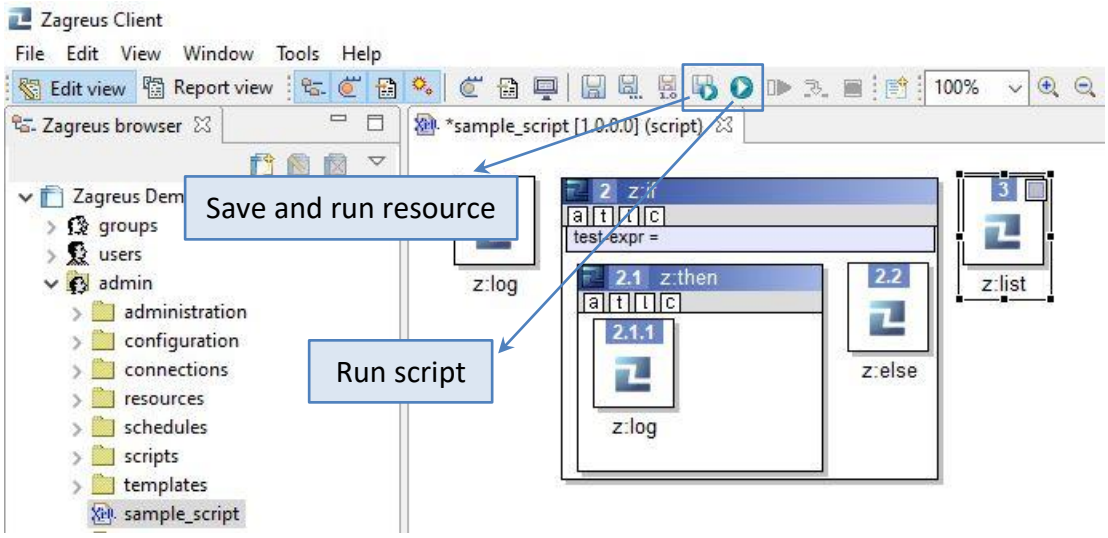


Figure 31 – Initiating the execution of a script which is opened for editing

15.6.2 Execution with the *.sendscripts* file

By using a special file called `.sendscripts`, the user can use a special type of manual script execution initiation. This file has to be placed in the Zagreus home folder of the user (see *Figure 32.*), and its content applies for the given user.

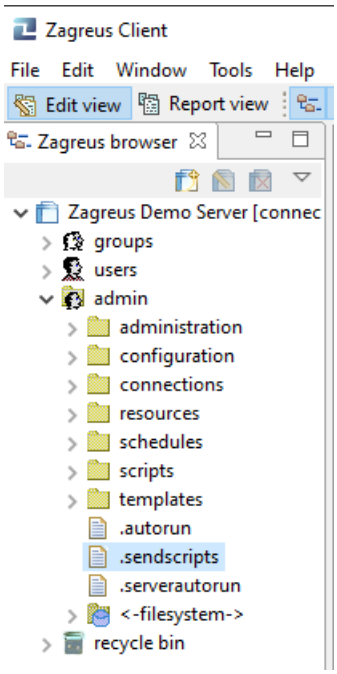


Figure 32 – The location of the `.sendscripts` file in the home folder of the admin user

In the `.sendscripts` file the user can list Zagreus scripts that can be executed from the Zagreus Client in a special way:

1) The user needs to select a resource that will be an input of the script that will be executed.

2) When the user right-clicks on the selected resource, a new context menu item *Send* will be shown, with the listed scripts from the `.sendscripts` file shown in a sub menu, see *Figure 33.*

3) By clicking on one of the possible script aliases, the given script will be executed and the properties of the originally selected resource will be passed as start-up script variables, see below.

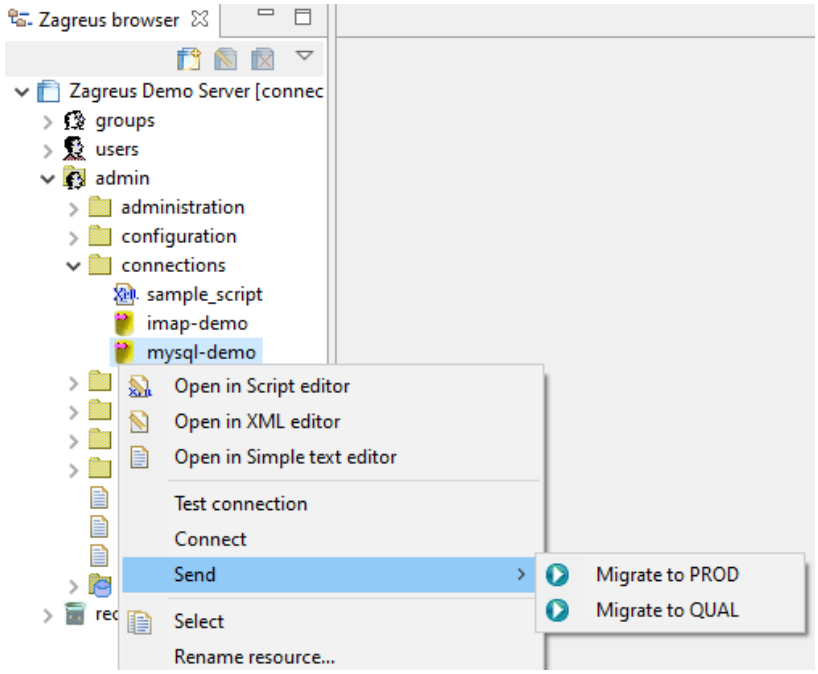


Figure 33 – `.sendscripts` example: using a MYSQL connection as an input parameter for a migration script

15.6.2.1 Format of the .sendscripts file

Each line of the file defines one case, containing the ID or the path of the script to be executed. A human-readable description can optionally be written to the same line, separated by a semicolon, which will be shown in the Zagreus Client in the *Send* context menu. Without adding description to an entry, the name of the script will be shown.

Empty lines and lines starting with the character # will be considered as remarks and will be ignored.

The keyword `import` can be used to include the contents of another file. The format of imported files must match the same rules as the original `.sendscript` file.

A sample `.sendscripts` file can be the following:

```
# here is the import
import /groups/common_group/shared_sendscript_definitions

# here come the new definitions
bd92555df4614090bfe1e31a45fe8f79; Migrate to PROD
/admin/scripts/migrate-qual; Migrate to QUAL
```

15.6.2.2 Passed start-up variables

If the user clicks on a custom menu item from the submenu, the execution of the given script (i.e. which was defined in the corresponding line of the `.sendscripts` file) will be initiated. For the execution, the following resource properties will be set as input parameters as script start-up variables:

- *$inputResourceId*: the id and version of the resource (concatenated)
- *$inputResourceName*: the name of the resource
- *$inputResourcePath*: the Zagreus path of the resource

> ***Info:*** *Changes will be applied immediately after saving the* `.sendscripts` *file – there is no need to restart the Zagreus Client or to disconnect from the Zagreus server.*

15.6.3 Execution from the command-line client

The user can run a Zagreus script from via the command-line client, the process is described here in details, see → runscript script.

15.6.4 Execution from the Zagreus HTML Application

The user can run a Zagreus script from via the Zagreus HTML Application, the process is described here in details, see → Zagreus HTML application.

15.6.5 Execution from external systems

Zagreus scripts can be executed by using SOAP requests from those platforms which supports this method. For example, in an Oracle database environment a SOAP call can be defined as a stored procedure, while in MSSQL Server a standalone binary has to be implemented and the `.dll` file must be imported into the database.

15.6.5.1 General SOAP format without script parameters

The SOAP request pattern for initiating script execution without any parameters: (the parts where the actual parameters have to be substituted are marked in orange)

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Header>
            <ns2:user
xmlns:ns2="http://ws.server.zagreus.etixpert.com/">MY_USERNAME</ns2:user>
            <ns2:password
xmlns:ns2="http://ws.server.zagreus.etixpert.com/">MY_BASE64_ENCODED_PASSWO
RD</ns2:password>
      </S:Header>

      <S:Body>
            <ns2:executeExt
xmlns:ns2="http://ws.server.zagreus.etixpert.com/">
                  <arg0>MY_SCRIPT_ID</arg0>
                  <arg1/>
                  <arg2/>
                  <arg3>3</arg3>
                  <arg4>html</arg4>
                  <arg5>false</arg5>
            </ns2:executeExt>
      </S:Body>
</S:Envelope>
```

15.6.5.2 General SOAP format with script parameters

The SOAP request pattern for initiating script execution with parameters: (the parts where the actual parameters have to be substituted are marked in orange)

```
<?xml version="1.0" encoding="UTF-8"?><S:Envelope
xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Header>
            <ns2:user
xmlns:ns2="http://ws.server.zagreus.etixpert.com/">MY_USERNAME</ns2:user>
            <ns2:password
xmlns:ns2="http://ws.server.zagreus.etixpert.com/">MY_BASE64_ENCODED_PASSWO
RD</ns2:password>
      </S:Header>
```

```
      <S:Body>
            <ns2:executeExt
xmlns:ns2="http://ws.server.zagreus.etixpert.com/">
                  <arg0>MY_SCRPT_ID</arg0>
                  <arg1>VARIABLE_NAME_1</arg1>
                  <arg1>VARIABLE_VALUE_1</arg1>
                  <arg1>VARIABLE_NAME_2</arg1>
                  <arg1>VARIABLE_VALUE_2</arg1>
                  ...
                  <arg1>VARIABLE_NAME_n</arg1>
                  <arg1>VARIABLE_VALUE_n</arg1>
                  <arg2/>
                  <arg3>3</arg3>
                  <arg4>html</arg4>
                  <arg5>false</arg5>
            </ns2:executeExt>
      </S:Body>
</S:Envelope>
```

Info: *When passing multiple input parameters for script execution, the variable names and variable values have to be put inside <arg1>...</arg1> tags; for example:*

```
<arg1>VARIABLE_NAME_1</arg1>
<arg1>VARIABLE_VALUE_1</arg1>
<arg1>VARIABLE_NAME_2</arg1>
<arg1>VARIABLE_VALUE_2</arg1>
...
<arg1>VARIABLE_NAME_n</arg1>
<arg1>VARIABLE_VALUE_n</arg1>
```

15.6.5.3 Execution from Oracle

A sample Oracle stored procedure for initiating script execution with encrypted password:

```
create or replace PROCEDURE "ZAGREUS_SAMPLE_CPWD" (p_server IN varchar2,
p_username IN varchar2, p_password IN varchar2, p_scriptid IN varchar2) AS
   http_req  UTL_HTTP.req;
   http_resp UTL_HTTP.resp;
   request_env varchar2(32767);
   response_env varchar2(32767);

BEGIN
    dbms_output.put_line('procedure started');

    request_env := '<?xml version="1.0" encoding="UTF-8"?>
    <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header>
    <ns2:user
xmlns:ns2="http://ws.server.zagreus.etixpert.com/">'||p_username||'</ns2:us
er>
    <ns2:password
xmlns:ns2="http://ws.server.zagreus.etixpert.com/">'||utl_raw.cast_to_varch
```

```
ar2(utl_encode.base64_encode(utl_raw.cast_to_raw(p_password)))||'</ns2:pass
word>
    </S:Header>
    <S:Body>
    <ns2:executeExt xmlns:ns2="http://ws.server.zagreus.etixpert.com/">
    <arg0>'|| p_scriptid ||'</arg0>
    ';


    request_env := request_env ||'<arg2></arg2>';
    request_env := request_env ||'<arg3>3</arg3>
    <arg4>html</arg4>
    </ns2:executeExt>
    </S:Body>
    </S:Envelope>
    ';

    http_req :=
utl_http.begin_request('http://'||p_server||'/zagreus/services/ws/zagreuswe
bbaseservice', 'POST', utl_http.HTTP_VERSION_1_1);
    utl_http.set_header(http_req, 'Content-Type', 'text/plain; charset=utf-
8');
    utl_http.set_header(http_req, 'Content-Length', length(request_env));
    utl_http.write_text(http_req, request_env);

    dbms_output.put_line('Request sent');

    http_resp := utl_http.get_response(http_req);
    utl_http.read_text(http_resp, response_env);
    utl_http.end_response(http_resp);

END ZAGREUS_SAMPLE_CPWD;
```

Execute the stored procedure with the following command:

```
call ZAGREUS_SAMPLE_CPWD('my.zagreus.server:7323', 'admin', '**********',
'0d020d9ceebb486998e56b1c24273253');
```

Note: encoding the password is done inside the stored procedure. As the input parameter for the stored procedure, the plain text password is required.

15.6.5.4 Execution from C#

To initiate script execution through the Zagreus web service from a C# code, at least the following includes must be used:

```
using System.Text;
using System.Threading.Tasks;
using System.Net.Http;
using System.Xml.Linq;
```

15.6.5.5 Execution from PHP

To initiate script execution through the Zagreus web service from a php code, the `soap` extension must be enabled in the php configuration file and `class SoapClient` must be used. The SOAP call can be performed by calling the method `__doRequest` of an instance of `SoapClient`.

A sample function for a Zagreus call from php:

```
public function doZagreusRequest($XMLStr) {
      $endpoint =
"http://localhost:7323/zagreus/services/ws/zagreuswebbaseservice";

$wsdlLocation = "http://localhost:7323/zagreus/zagreuswebbaseservice.wsdl";

$client = new SoapClient($wsdlLocation, array('encoding'=>'UTF-8',
'soap_version' => SOAP_1_2, 'trace' => 1, 'exceptions' =>1));

      $ret = $client->__doRequest($XMLStr, $endpoint, '', SOAP_1_2);

      return $ret;
      }
```

15.7 Execution from a Zagreus script

Initiating the execution of a Zagreus script from another running script (i.e. job) can be done by using the `zs:runscript` action. For the newly created job, the value of the *parent job id*, *caller*, *caller type* and *execution mode* attributes take the corresponding values. Of course, the script whose execution was initiated, might also contain a `zs:runscript` action, allowing the user to define execution chains.

Besides `zs:runscript`, there are further actions in the `ZS` action group which might initiate or affect script execution. The more important ones are the following:

- `z:fire-event`, `zs:fireevent`: fires an event schedule
- `zs:cancel`: cancels a running job
- `zs:wait`: waits for a running job to finish
- `zs:createevent`: creates an event schedule
- `zs:createschedule`: creates a time schedule
- `zs:createresource`: allows creating resources, including events and time schedules
- `zs:subscribe`, `zs:unsubscribe`: manages subscriptions

Of course, several further actions might initiate script execution in a more indirect manner; for example, the execution of a `file:copy`, `zs:copy` or `zs:delete` action might lead to the activation of a file trigger, a `mail:send` action might activate a mail watcher, while the user might use the `zs:setvariable` action to set the value of the `execute_script_on_cancel` option.

15.8 Summary

As described in this section, Zagreus provides a powerful infrastructure for automating IT and business processes. This flexibility extends to initiating Zagreus script execution as well, as the user can choose from a wide range of tools. Time schedules can be used for periodic maintenance of repetitive operations, while event schedules, file triggers, database watchers and mail watchers makes Zagreus able to respond to a variety of actions and events. These options offer Zagreus users a wide range of solutions to find the best setting for their particular automation problems.

A Zagreus installation contains three clients. The *Zagreus Client* is the most powerful application to connect to a Zagreus server, create resources, run scripts, fire triggers and check execution result (see also → Zagreus Client). With the *Zagreus HTML application*, a user might initiate script execution, fire events and check logs in a web browser of a PC, laptop, tablet, or a smart phone (see also → Zagreus HTML application). The *Zagreus Console applications* might also be used for initiating script execution and fire event schedules, however they also allow the user to perform some delegated administrative tasks (see also → Command-line tools). Using the `zs:runscript` and several further `ZS` actions, script execution can be initiated and affected on the local or even on remote Zagreus Servers, allowing building a powerful, automated system.

15.9 Best practices

Finally, we give some tips on configuring the script execution in the Zagreus System. There are practices for choosing the appropriate schedule, watcher and trigger type, also when it is worthwhile to invest time in creating an automated workflow rather than manually initiating script execution. There are also some hints to tune watchers, triggers, and check execution result.

15.9.1 Choosing the appropriate event-type resource

The main question is "*Does script execution depend from an event?*". As a rule of thumb, if a script should be executed If the answer likes the followings, then a mail watcher, a database watcher or a file trigger should be used: If a script should be executed if

- A report or a file is received: *mail watcher* (→ Mail watcher), *file trigger* (→ File trigger).
- Data is received from another department: *database watcher* (→ Database watcher), *file trigger* (→ File trigger).
- An update or a report about the status of a task or process is received: *mail watcher* (→ Mail watcher), *database watcher* (→ Database watcher).
- Data migration is performed: *database watcher* (→ Database watcher), *file trigger* (→ File trigger).
- Files are uploaded to some (other) server: *file trigger* (→ File trigger), *event schedule* (→ Event schedule).
- The execution of a job is finished: *event schedule* (→ Event schedule).
- The execution of a job ends in error: *special events* (e.g. *execute_script_on_error*) (→ Script execution by script options).
- The user logs into Zagreus: *special events* (→ Script execution by autorun configuration files).

For time-scheduled tasks, the main question is "*When should...*?". Answers for these kind of tasks are usually similar to the following:

- Before holidays.
- At the end of each weekday / weekend.
- At 8:00 am each weekday.

- At a special day in every year (i.e. anniversaries).
- Every 15 minutes (e.g. synchronize a folder).

15.9.2 Monitoring watchers and triggers

There are two ways for monitoring watchers and triggers: one is the *Monitor watchers and triggers* window in the Zagreus Client (see → Monitor watchers, triggers). This window is used only for monitoring the operation of watchers, triggers and the database connections kept open for the database watchers.

The other way of monitoring watchers and triggers is the Zagreus Monitor application. The job status and further corresponding data are displayed in a timeline view (see → Timeline area), while it is possible to filter on execution modes (triggered, scheduled, etc., see → Execution mode filter) as well as for job status (finished, error, running, etc., see → Status filter).

15.9.3 Quarterly settings for a time schedule

The following example (see *Figure 34.*) shows a setting for every quarter of a year. Instead of at the end of a quarter, the time schedule fires in the very first second of the next quarter to avoid the problem of months with 28, 30 and 31 days.

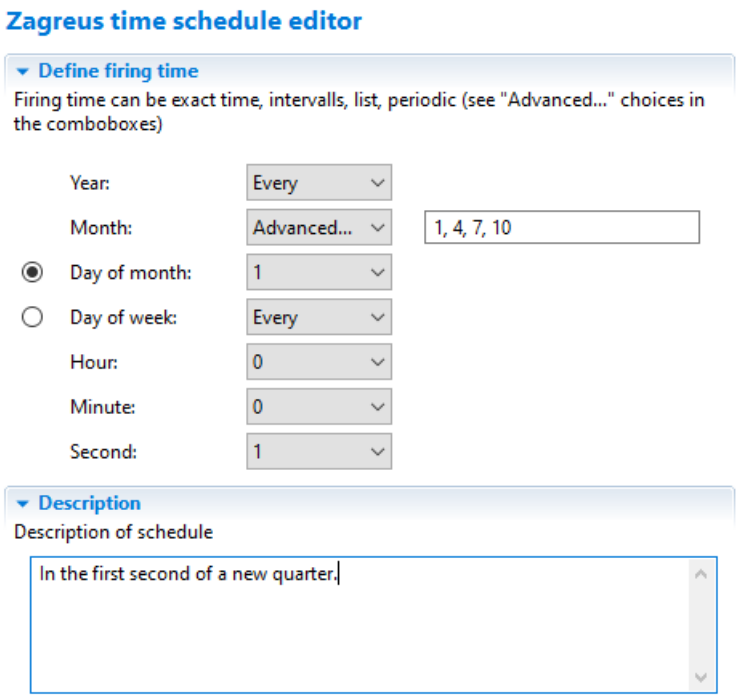


Figure 34 – Time schedule which fires at the end of each quarter

15.9.4 Using special subscription features

During the editing of subscriptions, the *Run now* button is useful for checking if the value of variables and options are set correctly. It triggers the execution of the subscribed script immediately. The settings can be checked in the job-log file, see *Figure 35*.

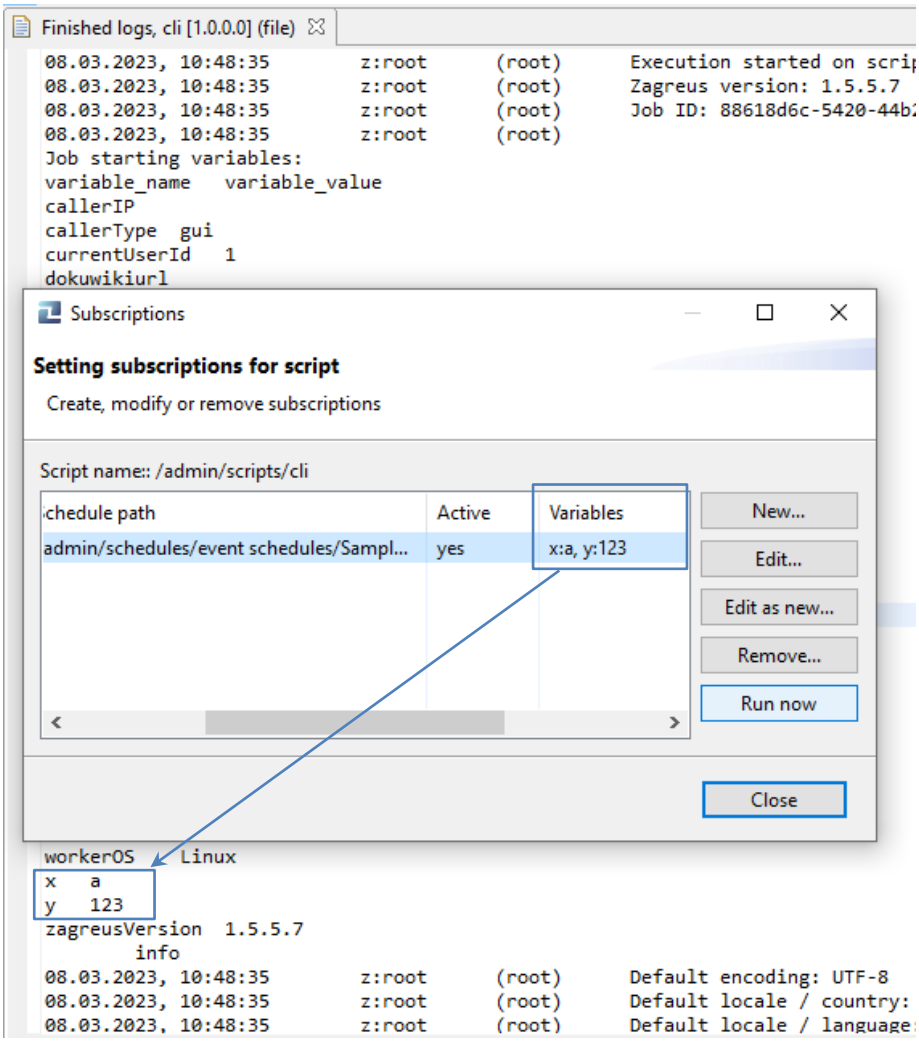


Figure 35 – Checking subscription variables in the job-log file after clicking the *Run now* button

The feature of the *Edit as new…* button is useful when the subscription variables and options are the same in multiple cases. If the user wants to subscribe the particular script to a new event-type resource with the same settings (or even with just smaller changes), this is an effective solution.

It is also recommended to think about when to delete and when to only deactivate a subscription. Since the subscriptions are deleted permanently, and therefore they cannot be recovered from the recycle bin, in most of the cases setting the *Active*

property of the corresponding subscription to false (i.e. deactivating it) could be a better solution.

15.10 Troubleshooting

15.10.1 Practices for event-type resources

In the followings some general practices and some specific cases will be described for the event-type resources, i.e. for *time schedules*, *mail watchers* and *database watchers*.

15.10.1.1 General practices

Make sure that the *Watchers* module of the Zagreus Server is running (see → Stop / start server components).

Since mail watchers and database watchers rely on a time schedule, it is also worth to see if the *Scheduler* module of the Zagreus Server is running as well (see → Quartz scheduler).

Instead of subscribing a heavy-weight, long-running script which operates on production resources, it is worth to experiment with a simple Zagreus script first, e.g. one with a simple `z:log` action.

15.10.1.2 Subscriptions

It is worth checking the *subscription* settings (see also → Editing a subscription). Variables and options set in the subscription override the existing script settings. The following steps should be performed to verify basic subscription settings:

- Check if the required subscription exists
- Check if the required subscription is active
- Check the variables and options of the subscription
- Use the *Run now* button to check the result of script execution triggered by subscription. In the job-log file, the actual values of the script starting variables can also be verified, see → Editing a subscription

15.10.1.3 Database watchers and mail watchers

Database watchers and mail watchers both rely on connection resources. It is recommended to test the database, POP3 or IMAP connection which is used in the watcher definition, see → Test connection feature.

Use the context menu item *Evaluate watcher condition...* to test the syntax and behavior of the filter condition, see → Evaluate watcher condition and → Evaluate watcher condition.

15.10.2 Command-line tools

When a script execution is initiated from the command line, the result can be verified in the *Finished jobs* window of the Zagreus Client (see also → Finished jobs window). If the result cannot be found in the *Finished jobs* window, then the initiation of the script execution has failed. In such a case, the following details are recommended to be checked:

- server definition in the command
- sort definition in the command
- port definition in the Zagreus Server configuration (see → General properties)
- user authentication data
- server machine firewall settings
- server machine and router port forwarding settings

About the command-line tools in details, see → Command-line tools.

15.10.3 The HTML application

In the case of Zagreus HTML application, if the appropriate HTML page does not load, the following items should be checked:

- Is the Zagreus Server running?
- Is the port definition in the URL correct?
- Is the port definition in the Zagreus Server configuration correct? (See → General properties)

Feedback messages about script execution are displayed in section *Logging console*, see → Run script and get info tab.

About the Zagreus HTML application in details, see → Zagreus HTML application.

16. Special features

There are some special features in Zagreus that are complex enough to be discussed in separate chapters. Next, these features are described in details.

16.1 Standalone Worker

Since Zagreus version 1.5.6.0, the Zagreus Worker module can be started as a standalone Java application ('Standalone Worker'). This allows the user to execute a script without the whole Zagreus System running. The script still can use a large portion of the Execution Engine functionality, but in the abscence of all other modules to which the Zagreus Worker is usually connected, some limitations are present.

Unlike the Zagreus Worker, which is a part of the whole Zagreus System, the Standalone Worker does not run continuously. It must be started by the user via command line with a specified script to be executed. When the script execution is finished, the Standalone Worker stops running.

16.1.1 How to use

The following list is the recommended way for the user to efficiently use the Standalone Worker module:

- The user first should develop the script to be executed in a standard Zagreus environment, being aware of the limitations of the Standalone Worker.

- The script then should be copied to the worker filesystem. It can be done, for example, by using a `file:read` action and the *worker-output* attribute.

- Since job execution cannot be monitored in the usual ways (e.g. Zagreus Client, job-log etc.), the script should be designed in a way that its output can be read from created files in the worker filesystem or in the Standalone Worker job-log files, see → Local filesystem in the Zagreus Worker.

- The user now can start the Standalone Worker with the provided command line script `zagreus_home/worker-controller/worker/standalone_worker.bat` (or `standalone_worker.sh` on Linux systems), using the path of the script and the optional script start-up variables as command line parameters. E.g.:

  ```
  standalone_worker.bat folder/script.xml x=1;y=2
  ```

  where the path of the script to be executed is relative to the worker filesystem root, see → Local filesystem in the Zagreus Worker, and the optional second command line parameter is of the format `key1=value1;key2=value2` …

- After the execution of the script has been finished (which is clearly visible in the command line window), the generated log files can be accessed in the specified log folder, see below. The default value is `zagreus_home/worker-controller/worker/joblog`.

- Any other files that were generated by the script execution can manually be inspected in the worker filesystem folder.

16.1.2 Configuration

Just like the Zagreus Worker, the Standalone Worker module is using the `worker.properties` configuration file, see → Zagreus Worker configuration. The following path properties are particularly important for the Standalone Worker module:

- *worker.filesystem.path*
  Defines the OS worker filesystem root folder, relative to the Worker module root directory.
  Default value: */filesystem*

- *worker.standalone.joblog.path*
  Defines the job output log folder, relative to the Worker module root directory.
  Default value: */joblog*

It is very important to set the correct `-Dworking.folder` and `-Djava.folder` parameter values inside the provided starting script.

In some cases, the user might need to manually edit the value of the Java classpath parameter (i.e. `cp`) in the starting script, e.g. when a new MicroStrategy library version is installed in the `/lib` folder of the Worker module, see → Worker startup properties.

16.1.3 Licencing

Because of the fact that the Standalone Worker module runs without a Zagreus Server connection, it has no access to the licence key which is installed on the server. Therefore it needs the licence key in its own folder structure. The user needs to put the licence key file into the `/conf` folder relative to the Worker module root under the name `key.txt`

16.1.4 Limitations

Because of the fact that the Zagreus Server is not accessible, the user needs to be aware that only resources stored only on the worker side are available. The following limitations should be considered when using the Zagreus Worker in standalone mode:

- The action group `file` cannot be used in the script. A substitute for using the `file` actions is the `wfile` action group.
- The *output* common attribute cannot be used. The user can use the *worker-output* common attribute instead, see → worker-output attribute.
- The `z:include` action does not work. The content of the included resource (in most cases a connection definition) must be directly copied into the executed script instead.
- The running of the Standalone Worker cannot be monitored from the customary Zagreus clients, e.g. Zagreus Client, Zagreus Monitor.

16.1.5 Notes

Here is a non-comprehensive list of actions and common attributes which should be avoided during the development of a script intended to be executed with a Standalone Worker:

- `file` action group
- `zs` action group
- `z:include` action
- `z:logfile` action
- *output* common attribute (use the common attribute *worker-output* instead)
- `ftp:lcd`, `ftp:mget`, `ftp:mput` actions
- *filename* attribute of `excel:read` action (use `wfile:read` child action instead)
- *template* attribute of `excel:workbook` action
- `pdf:to-image`, `pdf:extract-images` actions
- *attachments-target-path* attribute of the `msft:get-mail` action
- *xsl* attribute of the `xslt:transform` action (use *xsl-value* attribute instead)
- *filename* attribute of the `zip:read` and `zip:dir` actions (use `wfile:read` child action instead)
- *wildcard* attribute of the `zip:file` action

16.2 External script execution

One of the recent special features in Zagreus is the ability to integrate methods for external script execution. 'Scripts' in this context are not the Zagreus script resources but scripts from external programming languages and frameworks, like *Python*, *R*, *Powershell*, etc.

All this functionality is organized around the dedicated `z:execute` Zagreus action. In the following, the configuration and the usage of the external script execution is described in details.

16.2.1 How it works

External script execution works in the following way:

4) A Zagreus script is initiating an external script execution by the `z:execute` action inside the Zagreus script.

5) The `z:execute` action specifies an external script and its framework type (like *python*, *R*, *Powershell*, etc.) to be executed. This external script is stored in the local database or in the server filesystem as a simple file resource. An extension like `.py`, `.R`, `.ps1` is recommended to use.

6) When the Zagreus execution engine is processing the `z:execute` action, the following steps are taken:

   a) The external script is copied to a pre-defined temporary folder in the worker filesystem with a specified / generated name, see → Configuration

   b) A command line is constructed by using the executor binary, the copied temporary file name path and the parameters. The exact way of this construction is depending on the *tempfilename* and the *params* attributes, see → tempfilename attribute and → params attribute.

   c) The Zagreus Worker JVM starts a new external process to execute the constructed command line.

   d) Depending on the *async=false/true* attribute of the `z:execute` action, the Zagreus script processing waits for the end of the external process

(synchronous execution) or release it (asynchronous execution), see → Synchronous and asynchronous execution.

7) The Zagreus script continues to process its other actions and eventually ends.

8) The result of the externally executed script can be accessed in different ways, depending on the *async* attribute.

16.2.2 Configuration

Each script execution framework type has to be configured in the Zagreus Worker Configuration, see → Zagreus Worker configuration. First, a user-defined type label has to be chosen for the actual execution framework, e.g. 'python' for python executables. (In the following we will refer to this as '`<type>`'.) This label is very important since it will be referred to in the *type* attribute of the `z:execute` action.

Then the following properties have to be defined for each framework type:

- *worker.execute.<type>.bin*
  This property specifies the OS full path (in the format of the given OS) of the binary executable file of the selected framework, e.g. `C:\Python\python.exe`

- *worker.execute.<type>.tempfolder*
  This property specifies the path of the temporary folder relative to the root folder of the worker filesystem, where the temporary script files will be saved for execution, e.g. `/python-scripts`

16.2.3 *tempfilename* attribute

The copied temporary file name will be generated according to the following rules:

- If the *tempfilename* attribute is specified, the temporary file name will simply be the specified value.

- If the *tempfilename* attribute is not specified, then the temporary file name will be generated, with the actual time stamp (in the format 'yyyyMMddHHmmss'):

  - If the external script to be executed is defined by using the *name* attribute in the `z:execute` action, the file name of the specified Zagreus resource

will be used with the timestamp. For example, if the script path was */users/testuser/external_scripts/test.py*, the generated filename for the copied temporary file will be `test-20230901153030.py`

- If the external script to be executed is defined by any other way (using the *id* attribute or a child element, so the original file name cannot be retrieved), the name '*script*' will be used with the timestamp. For example, if the content of the (here, a python) script was inserted as a child element in the `z:execute` action, the generated filename for the copied temporary file will be `script-20230901153030.py`

16.2.4 *params* attribute

The command line is constructed according to the following rules, depending on the *params* attribute:

- If the *params* attribute is not specified, the command line is:
  `executor_binary temporary_filename_fullpath`

- If the *params* attribute is specified, the command line is using the value of this attribute as the parameter list, i.e.:
  `executor_binary temporary_filename_fullpath parameters`

- If the *params* attributes is specified and it contains the *#tempfilename* string, then the *#tempfilename* string will be substituted by the fullpath of the copied temporary file, and the command line will look like:
  `executor_binary resolved_parameters`
  This is the only workaround for the special cases where the temporary file fullpath is not the first parameter of the executor binary, see this example → #tempfilename substitution.

Note: the *params* attribute can contain multiple parameters divided by the space character. Also, depending on the operating system, if there is a space character inside one parameter value, double quotes can be used to force them to be created as one parameter. For example, the string

one two "three four" five

will be treated as only four parameters.

16.2.5 Synchronous and asynchronous execution

External scripts can be executed either in a synchronous or asynchronous way, specified by the *async* attribute in the `z:execute` action. In case of synchronous execution (*async="false"*), the Zagreus processing engine waits for the end of the external process, and only then it continues processing the subsequent Zagreus action. Also, the output of the externally executed script is accessible as the result of the `z:execute` action, as well as the exit value of the external execution is passed as the result attribute `exit_value`.

On the other hand, in case of asynchronous execution (*async="true"*), the Zagreus processing engine starts the external script execution process, but it immediately continues processing the subsequent Zagreus action. Since we release the external process at this point, the Zagreus script cannot reach its output or exit value. Therefore the output stream and the error stream of the external process are redirected into two separate files in the worker filesystem, right next to the copied external script. Their file names are the same as the original copied script, but with `.out` and `.err` extensions, respectively. The OS full path of these files are returned in the results attributes `output_filename` and `error_filename`.

16.2.6 Examples

In the `worker.properties` configuration file, the external framework type label 'python' has been chosen, and used in the following two properties:

```
worker.execute.python.bin=C:\python-embedded\python.exe
worker.execute.python.tempfolder=/python
```

In the Zagreus local database, an external python script *test.py* is saved in the folder */users/testuser/external_scripts*, see *Figure 1*.

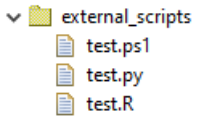


Figure 1 – The list of external scripts stored in the external_scripts folder

The content of *test.py* is the following:

```
import sys
```

```
print("Number of arguments: ", len(sys.argv), "arguments.")
print("Argument List: ", str(sys.argv))
```

Next we present some examples which consist of executing this python script via a `z:execute` action in different ways.

16.2.6.1 Synchronous execution

In order to execute the above python script synchronously, the following action has been created in a Zagreus script:

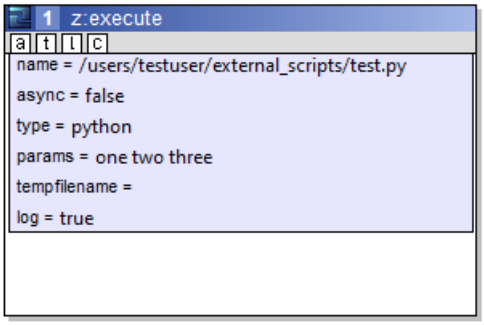


Figure 2 – A `z:execute` action referring to the *test.py* script with *async="false"*

The `z:execute` action in *Figure 2.* will perform the following steps:

1) It loads the content of the */users/testuser/external_scripts/test.py* file.

2) The content is saved in the */python* folder of the worker filesystem. Hence the *tempfilename* attribute is empty, the copied file name will be generated with a timestamp, such as `test-20230901153030.py`

3) The processing engine translates the worker file system path of the previously copied file into OS full path, such as `C:\Programme\zagreus\worker-controller\worker\filesystem\python\test-20230901153030.py`

4) The processing engine constructs the full command line using the specified executor binary, the translated OS filesystem path and the specified parameters:

```
C:\python-embedded\python.exe C:\Programme\zagreus\worker-controller\
worker\filesystem\python\test-20230901153030.py one two three
```

5) The processing engine starts an external process executing the previously constructed command line. Since the *async* attribute was set to *false*, it waits for

the end of the external process. The output of the external process is passed as the result of the `z:execute` action.

6) Since there is a *log="true"* attribute specified, the result of the `z:execute` action is logged to the job-log file:

```
Number of arguments:  4 arguments.
Argument List:  ['c:/work/eclipse_workspace_new/zagreus-worker-1.5.0/filesystem/python/test-20231115144950.py', 'one', 'two', 'three']
```

16.2.6.2 Asynchronous execution

In order to execute the *test.py* python script defined above asynchronously, the following action has been created in a Zagreus script:

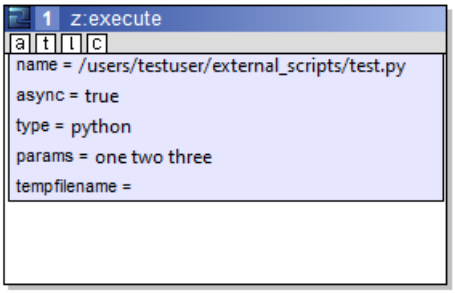


Figure 3 – A `z:execute` action referring to the *test.py* script with *async="true"*

The `z:execute` action in *Figure 3.* will perform the following steps:

1) It loads the content of the */users/testuser/external_scripts/test.py* file.

2) The content will be saved in the */python* folder of the worker filesystem. Hence the *tempfilename* attribute is empty, the copied file name will be generated with a timestamp, such as `test-20230901153030.py`

3) The processing engine translates the worker file system path of the previously copied file into OS full path, such as `C:\Programme\zagreus\worker-controller\worker\filesystem\python\test-20230901153030.py`

4) The processing engine constructs the full command line using the specified executor binary, the translated OS filesystem path and the specified parameters:

```
C:\python-embedded\python.exe C:\Programme\zagreus\worker-controller\
worker\filesystem\python\test-20230901153030.py one two three
```

5) The processing engine starts an external process executing the previously constructed command line. Since the *async* attribute was set to *true*, it immediately continues executing the further actions of the Zagreus script.

6) The external process is executing the specified command line independently of the Zagreus processing engine. It may even last longer than the execution of the Zagreus script.

7) Due to the asynchronous execution, the output and the error streams of the external process are redirected to two separate files in the */python* worker filesystem subfolder, with the `.out` and `.err` extensions, respectively, see *Figure 4*.

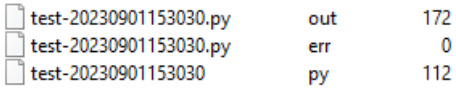


Figure 4 – The copied python script and the generated output and error files

16.2.6.3 Custom *tempfilename* attribute

The user can opt for specifying a custom name for the copied temporary file by using the *tempfilename* attribute. In the next example, the `z:execute` action is the same as seen in *Figure 3.*, but with a *tempfilename* attribute specified:

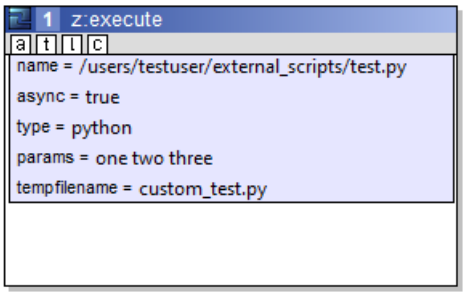


Figure 5 – A `z:execute` action specifying the *tempfilename* attribute

The steps of the execution are identical to those in the previous example, with the exception of the name of the copied file, which will be `custom_test.py` (without a timestamp). Therefore, the following files will be generated in the worker filesystem temporary folder:

Figure 6 – The copied python script and the generated output and error files with a custom name

16.2.6.4 *#tempfilename* substitution

In this example a Powershell script is executed with the `z:execute` action. In order to allow this, in the `worker.properties` configuration file, the type label 'powershell' has been chosen, and is used in the following two properties:

```
worker.execute.powershell.bin=powershell.exe
worker.execute.powershell.tempfolder=/powershell
```

Note: there is no need to specify the full path of the executor binary (i.e. `powershell.exe`), if the location of the binary is part of the OS system path.

In the Zagreus local database, an external Powershell script *test.ps1* is saved in the folder */users/testuser/external_scripts* (see *Figure 1*.) with the content:

```
Write-Host "You passed $($args.Count) arguments:"
$args | Write-Host
```

In *Figure 7.,* a properly configured `z:execute` action is shown:

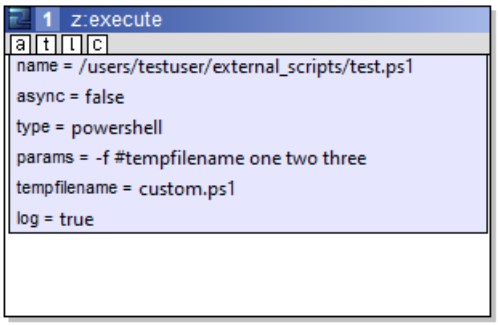


Figure 7 – A `z:execute` action specifying a powershell type

In the case of executing Powershell scripts with the powershell.exe binary, there is a special order of parameters: the script to be executed is not the first parameter of `powershell.exe` (as it is in case of most execution frameworks), but it has to be specified with the `-f` switch. Because of this, the user has to properly specify the whole parameter list in the *params* attribute, including the *#tempfilename* tag, which will be substituted with the copied custom temporary file name. Thus, the whole command line will be the following:

```
powershell.exe -f C:\Programme\zagreus\worker-controller\worker\
filesystem\powershell\custom.ps1 one two three
```

16.3 Document URL feature

In the Zagreus System, there is a possibility to open dynamically created external links from the Zagreus Client application. This is useful for externally stored documentation (e.g. Confluence pages) for individual resources.

For example, in Confluence, it is common for the URL of a page to include the title of the page. Confluence often generates URLs based on the title to create human-readable and user-friendly links. The title is usually converted to lowercase, spaces are typically replaced with + signs, in the following format:

```
https://your-confluence-site.com/display/SPACEKEY/My+Example+Page
```

Therefore, a resource-based documentation can easily be implemented by Confluence system using the Zagreus Document URL feature, which allows the user to use document-specific external links. These links open in the default browser of the installation environment.

The Zagreus Document URL feature works for the following resource types:

- script
- connection
- template
- time schedule
- event schedule
- mail watcher
- database watcher
- file trigger

Therefore, this feature can only be used for resources stored in the embedded database, and not to those stored in the local OS filesystem.

16.3.1 How to use

The Document URL needs pre-defined variables. These variables can be declared on several different levels, for example *server*, *owner*, *script* level (see also → Declaration levels). For further details about the variables, see the next chapter.

Once the Document URL feature is configured, the user can use it in the Zagreus Client application by opening the *Resource info* dialog box (see → Resource information). At the end of the first line of the dialog box, a new icon appears, see

Figure 8. By clicking on the icon, the default external browser opens the dynamically created document URL.

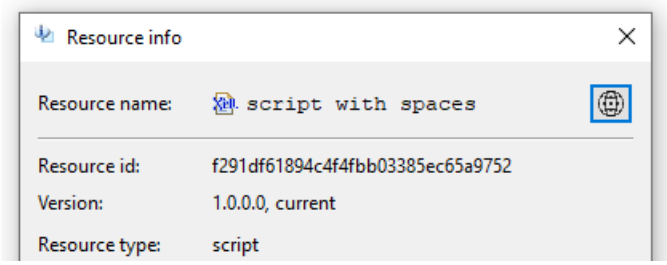


Figure 8 – The Document URL link icon in the *Resource info* dialog box

16.3.2 *docurl* variable

The *docurl* variable must be declared on any of the following levels: *server*, *owner* and *script.* The already discussed order of variable resolution (see → Precedence order for resolution) applies to this variable declaration, i.e. a variable *docurl* defined on the script level overrides the variable *docurl* defined on the server level. The value of the *docurl* variable defines a valid URL with the necessary *http://* or *https://* prefixes. For example, :

```
variable.server.docurl=https://my-confluence.com/display/SPACE/confluence-page
```

This example defines a static link, i.e. it will point to the same URL for any given resource. For the dynamic behaviour, there is a possibility to use substitution keys. When a substitution key is present in the value of the *docurl* variable, it will be resolved dynamically when the user opens the *Resource info* dialog box. The following substitution keys can be used:

- *%resourceId*
  refers to the ID of the resource

- *%resourceIdWithVersion*
  refers to the ID of the resource with the concatenated version

- *%resourceName*
  refers to the name of the resource

- *%resourcePath*
  refers to the full path of the resource

- *%resourceDescription*
  refers to the description of the resource until the first line break, see → Resource properties

For example, for the *docurl* variable using the *%resourceName* substitution key:

```
variable.server.docurl=https://confluence.com/display/SPACE/%resourceName
```

will be resolved as *https://confluence.com/display/SPACE/sample_script* for the resource named *sample_script*.

16.3.3 *docurl_replace* variable

Sometimes there is a need to replace particular characters in the dynamically created link. For example, in Confluence the URL of a page is derived from the title of the page. When the title contains spaces, these appear as plus signs in the page URL. In Zagreus, the only way to replace some special characters to others in the generated Document URL is the provided *docurl_replace* variable functionality. The usage of this variable is optional.

Just like the *docurl* variable, the *docurl_replace* variable can also be declared on the *server*, *group*, *user* and *script* levels. Variable resolution behaves the same way as well. The value of the *docurl_replace* variable needs to follow the following format:

```
variable.server.docurl_replace=' ',,'+'
```

This example defines a character pair: all occurrences of the first character (in this example, the space character in apostrophes) will be replaced by the second one (the plus sign). The two character definitions are separated by double commas (*,,*).

This functionality is not limited for replacing single characters, substrings can also be used.

Multiple replacements can be defined by using double semicolons (*;;*) as separators.

```
variable.server.docurl_replace=' ',,'+';;'ä',,'a'
```

This example replaces all space characters with plus signs as well as all a-umlaut (*ä*) characters with the letter '*a*'.

16.3.4 Examples

16.3.4.1 General *docurl* definition

When the user wants to use a general approach for the Document URL feature, only two variables need to be specified on the server level:

```
variable.server.docurl=https://confluence.com/display/SPACE/%resourceName
variable.server.docurl_replace=' ',,'+'
```

This allows the user to open the *Resource info* dialog for the supported resource types and to click on the appearing Document URL icon. The generated URL contains the name of the selected resource at the end, and the space characters are replaced by plus signs.

16.3.4.2 User-specific *docurl* definitions

When the goal is to use a general approach for the Document URL feature for most users, but a different approach for one specific user, first both variables need to be specified on the server level:

```
variable.server.docurl=https://confluence.com/display/SPACE/%resourceName
variable.server.docurl_replace=' ',,'+'
```

Then the following variables need to be defined on the user level, see *Figure 9.*:

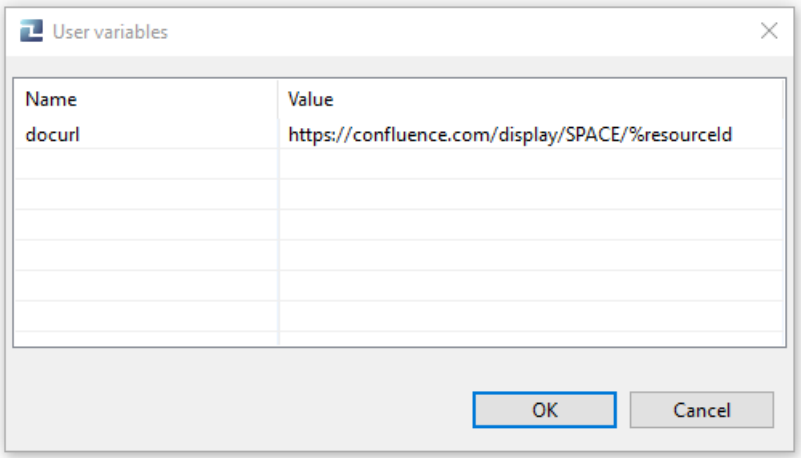


Figure 9 – The *docurl* variable set for the specific user

Now the Document URL behaves in the following way:

- For resources whose owner is not the specific user, the generated Document URL will follow the general approach declared on the server level.

- For resources whose owner is the specific user, the generated Document URL will follow the user-specific approach as seen in *Figure 9.*

16.3.4.3 A single script-level *docurl* definition

When the goal is to set a Document URL for one single script resource, the user simply needs to set the *docurl* variable for the specific script, see *Figure 10.*:

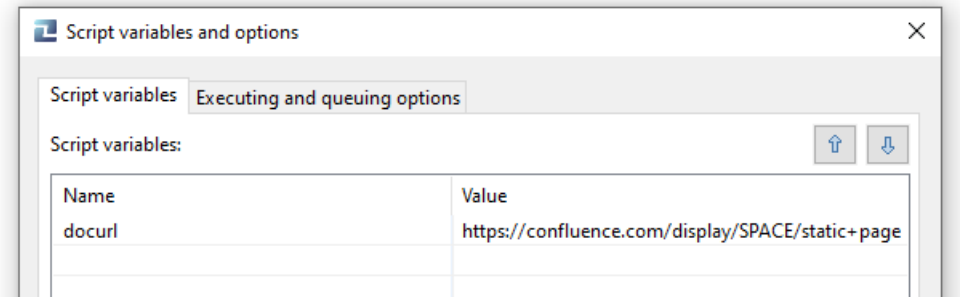


Figure 10 – The *docurl* variable set for the specific resource

Since the variable declared on the *script* level overrides all other levels where the *docurl* variable might be set, the specified Document URL will be assigned to the given script. Therefore, a static link is satisfactory without any substitution.

For the other resources, the Document URL will be determined based on the variables set on the other levels (i.e. *server*, *group* and *user*).

16.3.4.4 Defining the Document URL in the resource description

There is a special case when the resource description contains the Document URL for each resource. To do this, the user needs to specify the `%resourceDescription` as the value of the *docurl* variable (either on *server*, *user* or *group* level). The following example shows such a definition on the *server* level:

```
variable.server.docurl=%resourceDescription
variable.server.docurl_replace=' ',,'+'
```

Then, each resource description must contain the whole Document URL static link in the first line, see *Figure 11.*:

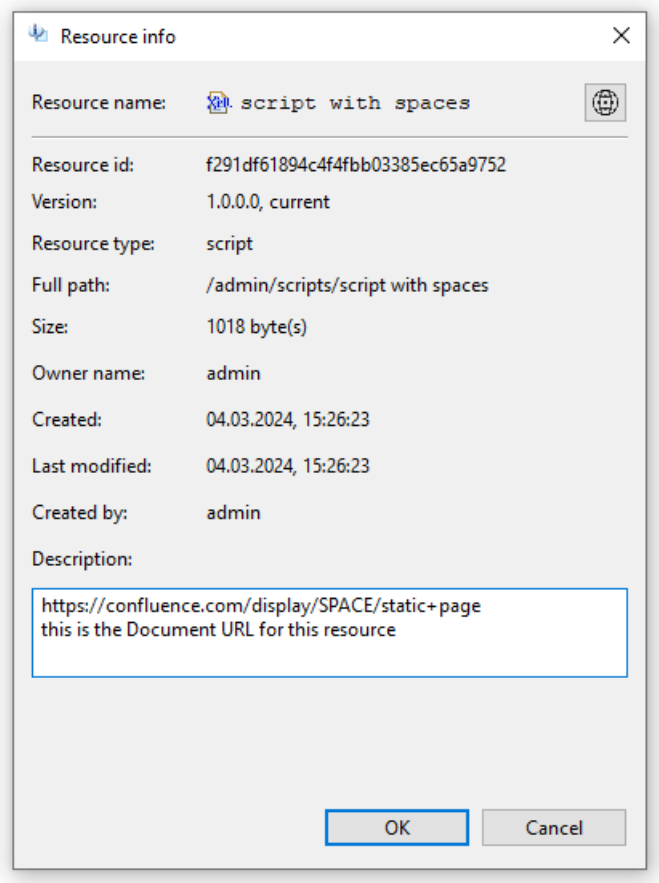


Figure 11 – The Document URL link is set in the resource description

As it was mentioned earlier, only the first line is substituted by the `%resourceDescription` substitution string (from the second line the user can add additional comments).

16.3.4.5 A more complex example with variable referencing

The docurl variable can also refer to other variables, declared on different levels. By using variable referencing, more complex examples can be built.

In the next example, aside from the usual *docurl* and *docurl_replace* variables, there are two further variables defined on the server level:

```
variable.server.docurl=https://confluence.com/display/SPACE/%resourceName
variable.server.docurl_replace=' ',,'+'

variable.server.docurl_alt_1=%resourceDescription
variable.server.docurl_alt_2=https://custom-server.com/%resourceId
```

The variables *docurl_alt_1* and *docurl_alt_2* will behave as optional alternatives against the default *docurl* variable.

For one specific user, the *docurl* variable can be overridden by referencing to the first alternative variable declared on the server level, see *Figure 12.*:

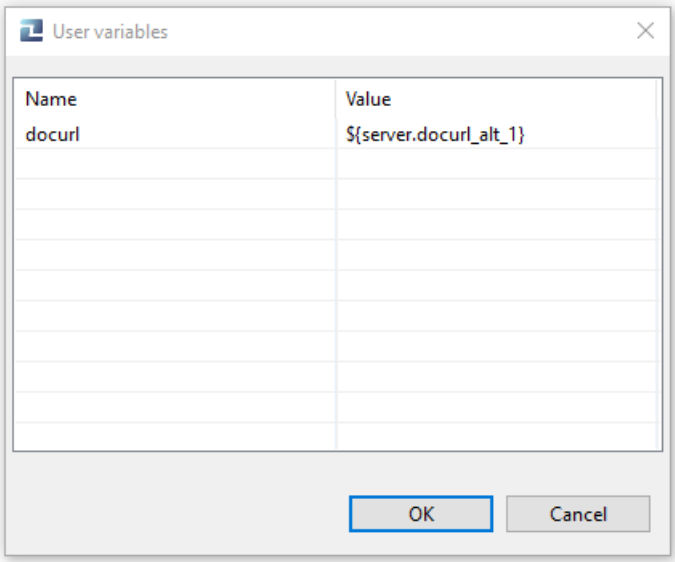


Figure 12 – The *docurl* variable set for a user is referencing to the *docurl_alt_1* server variable

For one specific script, the *docurl* variable can be overridden by referencing to the second alternative variable declared on the server level, see *Figure 13.*:

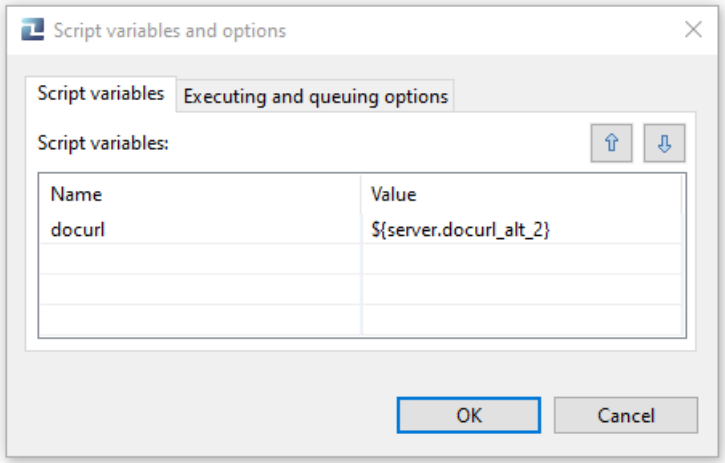


Figure 13 – The *docurl* variable set for a script is referencing to the *docurl_alt_2* server variable

In this example, the specific script will generate the Document URL derived from the value of the script ID (i.e. *docurl_alt_2*), while the Document URLs of all resources of the aforementioned specific user will use the *%resourceDescription* substitution string. All other resources will simply use the default *docurl* variable from the server level.

By using such a structure, the Document URL behavior can be controlled in a very flexible and user-friendly way.

16.4 Bank holidays feature

The Zagreus System maintains a list of common European holidays as well as a configurable, country-specific list of additional bank holidays. The list plays an important part when performing working day-related calculations with the following functions:

- *workingday(date)* and *workingday(date, locale)*
  Tells if the given date (with the optional locale) is a working day.
  Example:
  workingday('2023-05-01', 'AT') returns `false`
  workingday('2023-12-18', 'AT') returns `true`

- *workingdayofmonth(date)* and *workingdayofmonth(date, locale)*
  Tells which working day of the actual month is the given date.
  Example:
  workingdayofmonth('2023-12-04', 'AT') returns `2`

- *workingdaysafter(date, number)* and *workingdaysafter(date, number, locale)*
  Tells the date which is the specified 'number' workingdays later than the given date.
  Example:
  workingdaysafter('2023-12-04', 5, 'AT') returns `'2023-12-11'`

16.4.1 Common European holidays

This is a fixed list of common European holidays:

- *January 1*
- *May 1*
- *November 1*
- *December 25 1*
- *December 26*
- *Easter Monday*
- *Ascension Day*
- *Whit Monday*
- *Corpus Christi*

16.4.2 Specifying additional bank holidays

The Zagreus System provides a way for the user to specify additional bank holidays. It can be useful to set locale-specific bank holidays (i.e. dates which count as holidays only in the particular country).

The following steps need to be taken in order to set a list of additional bank holidays:

- *Specifying the full path of a bank holidays descriptor file*
  The user needs to specify the full path (either in the embeded database or in the local filesystem) which points to the simple text file which contains the list of additional bank holidays (see below). This can be set by the *bankholidays.path* property in the Zagreus Server configuration, see → Miscellaneous properties.

- *Creating the bank holidays descriptor file*
  The user needs to create a simple text file containing the list of additional bank holidays in the following format: *YYYY-MM-DD;<Country code>* in each line. Comments can be used by starting the line with the # character.
  Example:
  2023-01-06;AT
  2023-08-15;AT

17. Server administration in the Zagreus Client

There are a bunch of administrative options which are available in the Zagreus Client application. These options are grouped together under the server definition node context menu in the Zagreus Browser window, see *Figure 1*.

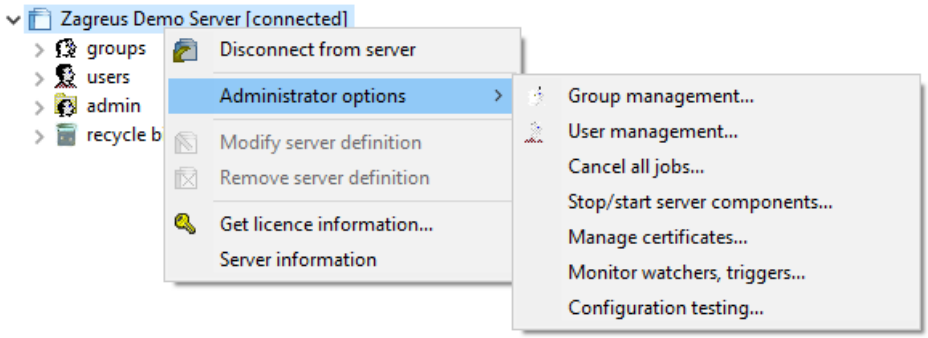


Figure 1 – The server definition node context menu

17.1 Administrator options

The menu item *Administrator options* contains several sub-menu items, which will be described in details in the further sections.

17.1.1 Group management

It opens the *Group management for administrators* wizard. Here, groups can be managed based on the options available in the *Select action* dropdown list, see *Figure 2.*

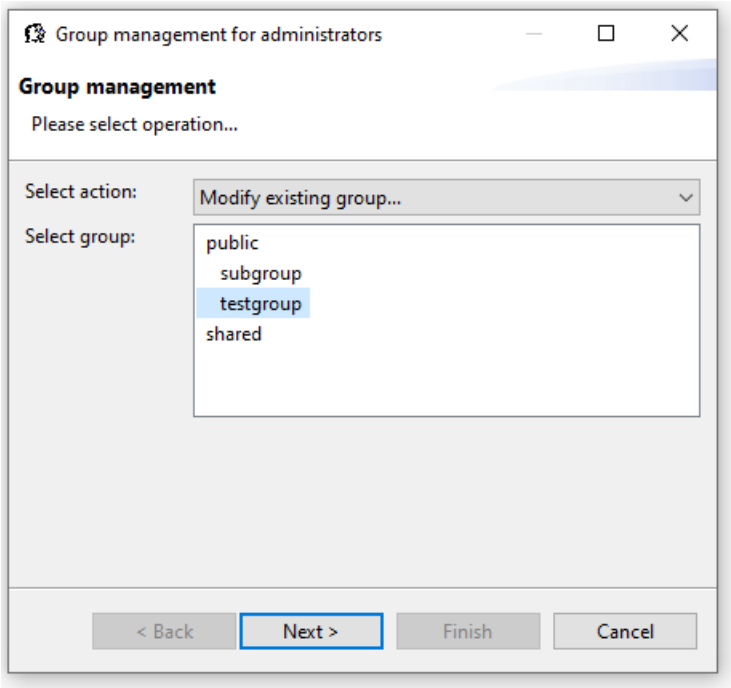


Figure 2 – The *Group management for administrators* wizard

The following options are available in the *Select action* dropdown list:

- *Create new group…*: for creating a new group
- *Modify existing group…*: for modifying a group
- *Delete existing group…*: for deleting a group

When the user selects the *Modify existing group…* or *Delete existing group…* action, a further list *Select group* appears for selecting a particular existing group.

By pressing the *Next* button, the wizard proceeds to the next wizard page to complete the selected action.

17.1.1.1 Create new group

By selecting this option, a new group can be created, see *Figure 3.*

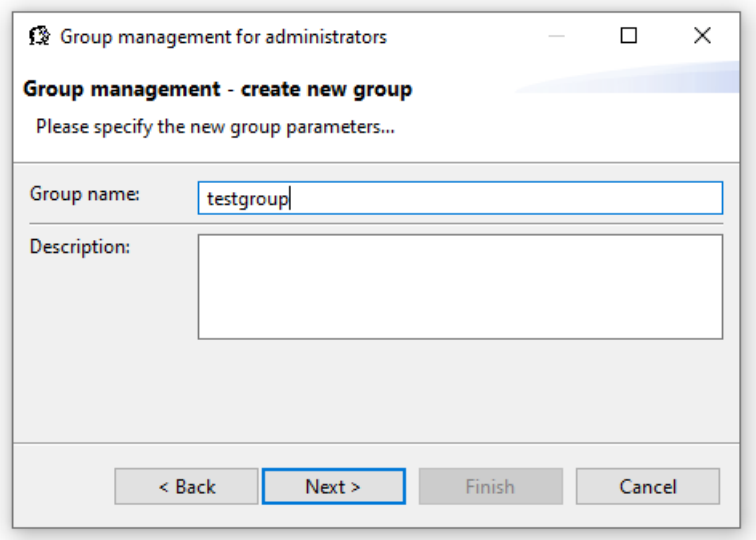


Figure 3 – Creating a new group with the *Group management* wizard

The following options are available on this wizard page:

- *Group name*
  The name of the group to be created.

- *Description*
  A description can be defined for the group.

After specifying the required information and pressing the *Next* button, the *Group management - summary* wizard page appears for the final confirmation, see *Figure 4*.

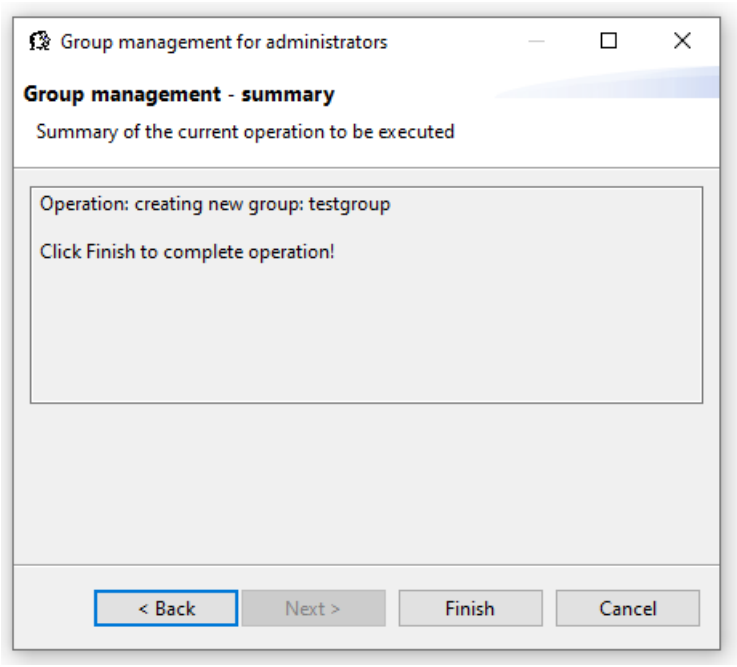


Figure 4 – Group management summary for final confirmation in the *Group management* wizard

17.1.1.2 Modify existing group

By selecting this option, an existing group can be modified, see *Figure 5.*

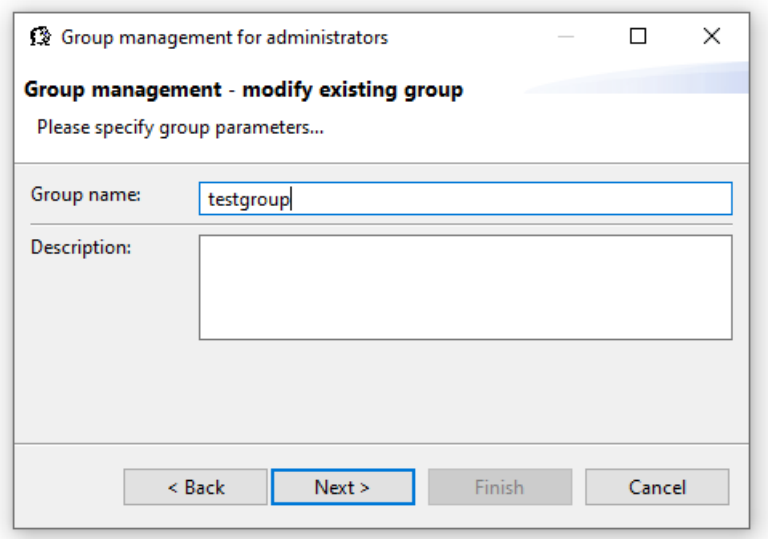


Figure 5 – Modifying a group with the *Group management* wizard

The following options are available on this wizard page:

- *Group name*
  The name of the group can be modified here.

- *Description*
  A description of the group can be modified here.

After specifying the required information and pressing the *Next* button, the *Group management - summary* wizard page appears for the final confirmation, see *Figure 4*.

17.1.1.3 Delete existing group

By selecting this option, an existing group can be deleted. The *Group management - summary* wizard page appears for the final confirmation, see *Figure 4*.

17.1.2 User management

It opens the *User management for administrators* wizard. Here, users can be managed based on the options available in the *Select action* dropdown list, see *Figure 6.*

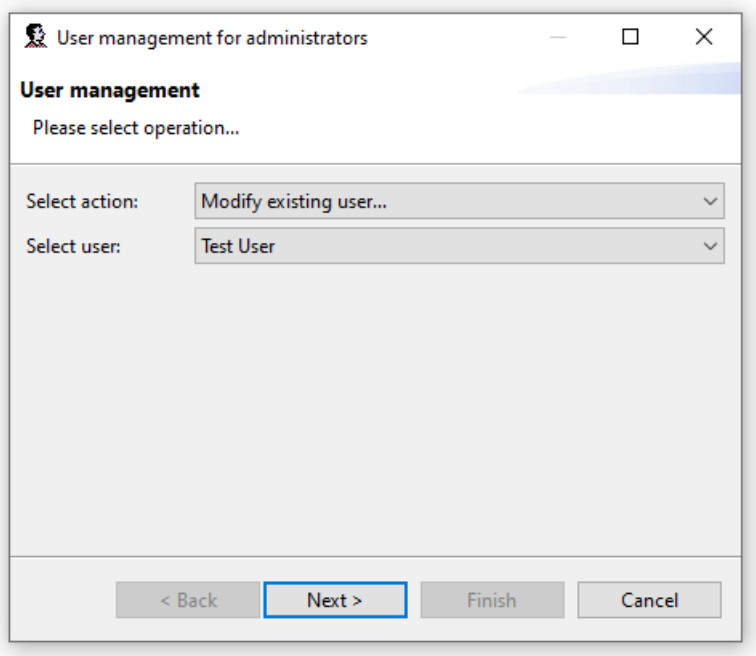


Figure 6 – The *User management for administrators* wizard

The following options are available in the *Select action* dropdown list:

- *Create new user…*: for creating a new user
- *Modify existing user…*: for modifying a user
- *Delete existing user…*: for deleting a user

When the user selects the *Modify existing user…* or *Delete existing user…* action, a further dropdown *Select user* appears for selecting a particular existing user.

By pressing the *Next* button, the wizard proceeds to the next wizard page to complete the selected action.

17.1.2.1 Create new user

By selecting this option, a new user can be created, see *Figure 3.*

To create a new user, select the *Create new user...* option and click Next to open *User management - create new user* wizard page.

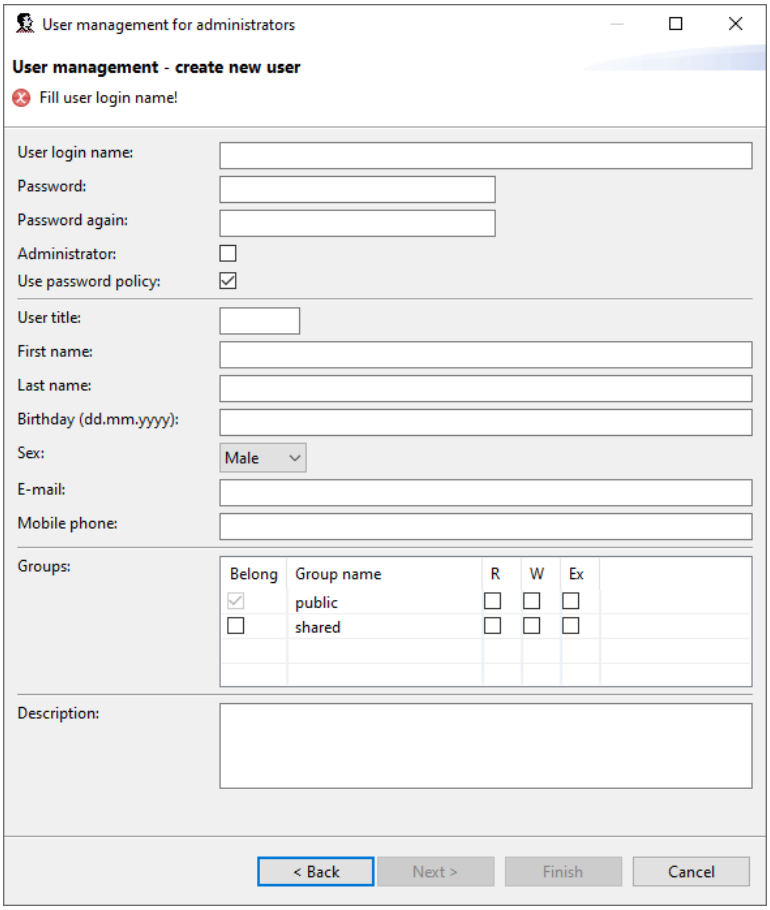


Figure 7 – Creating a new user with the *User management* wizard

The following options are available on this wizard page:

- *User login name*
  The name of the user to be created. This field is mandatory.

- *Password*
  The password for the user to be created. If the *Use password policy* option is checked, the password policy rules must be applied. This field is mandatory.

- *Password again*
  The user has to enter the password again to ensure that there are no typing errors. This field is mandatory.

- *Administrator*
  If set, the new user will have administrator rights, see → Administrator user rights.

- *Use password policy:*
  If set, the password policy is activated for the user, see → Password policy.

- *User title*
  The title of the user.

- *First name*
  The first name of the user.

- *Last name*
  The last name of the user.

- *Birthday (dd.mm.yyyy)*
  The birthday of the user. The date format is indicated inside the parentheses.

- *Sex*
  The sex of the user.

- *E-mail*
  The e-mail of the user.

- *Mobile phone*
  The mobile phone number of the user.

- *Groups*
  The groups associated with the new user. By default, the user is assigned to the *public* group. Additional groups can be selected by checking the checkbox under the *Belong* column. The *R*, *W* and *Ex* are indicators for *read*, *write* and *execute* user rights, respectively, for the particular group.

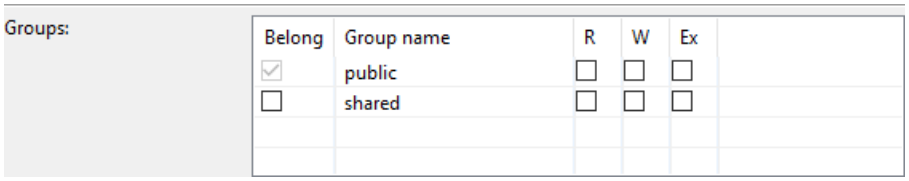


Figure 8 – The *Groups* table in the *User management* wizard

- *Description*
  A description for the user.

After specifying the required information and pressing the *Next* button, the *User management - summary* wizard page appears for the final confirmation, see *Figure 9*.

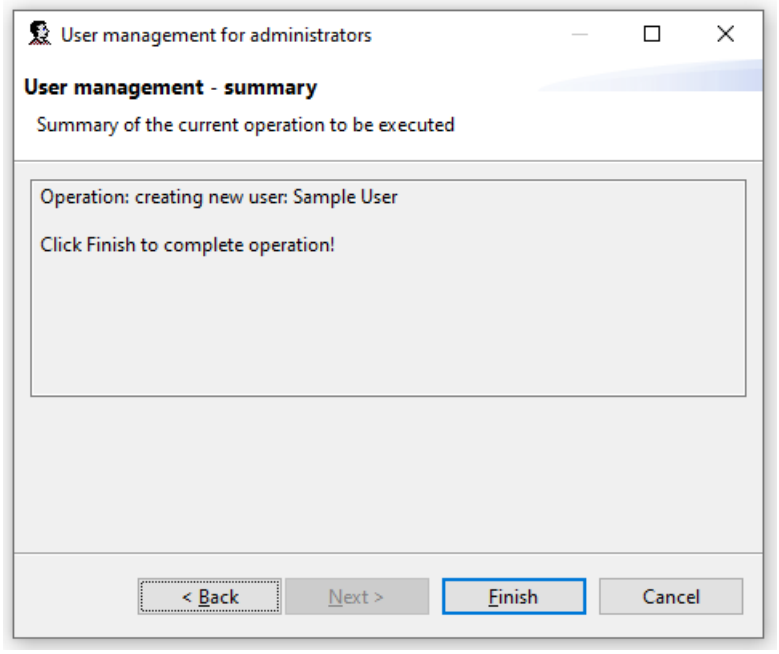


Figure 9 – User management summary for final confirmation in the *User management* wizard

17.1.2.2 Modify existing user

By selecting this option, an existing user can be modified, see *Figure 10.*

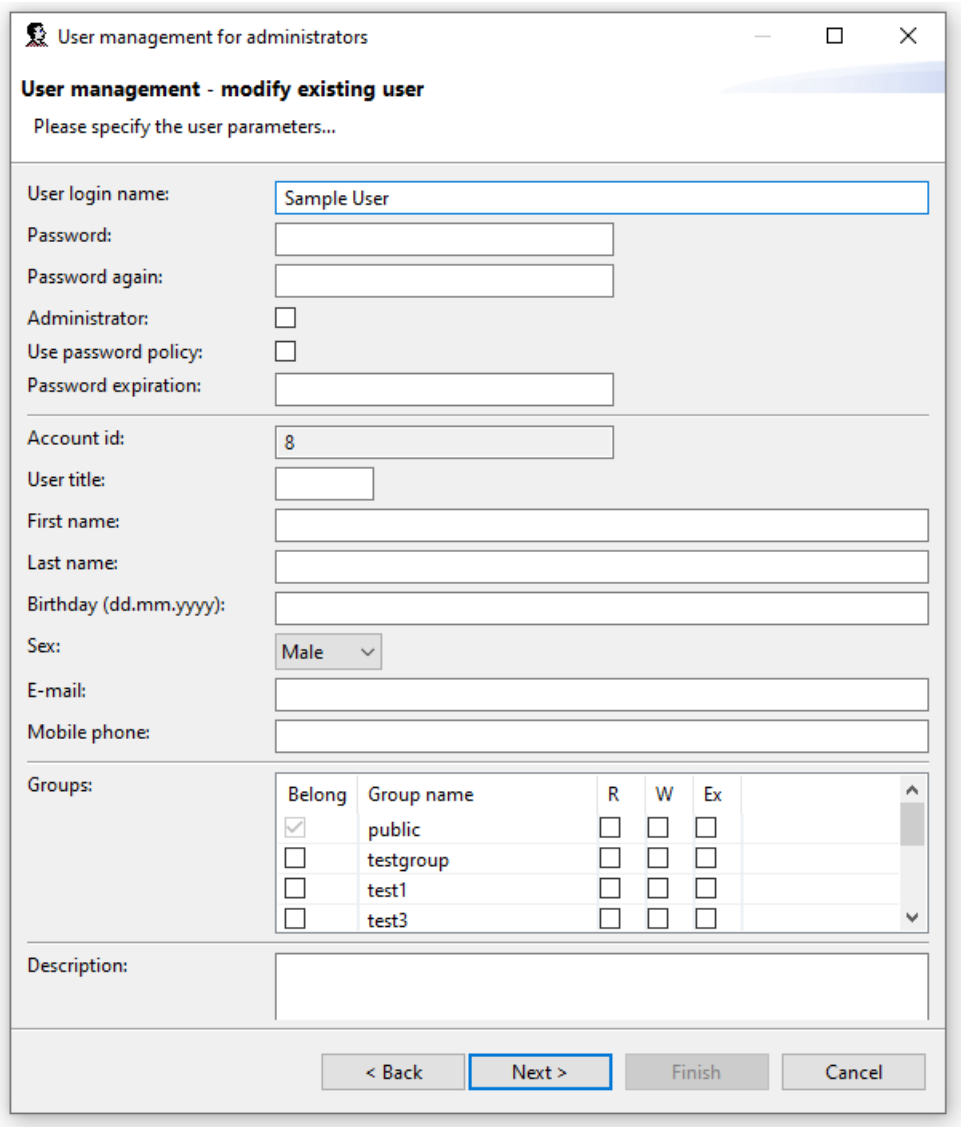


Figure 10 – Modifying a user with the *User management* wizard

The attributes are mostly identical to those of the user creating process in the previous chapter; next only the differences are listed:

the same as those available during the user creation process (See -->), with the exception of one when the password policy option is selected for the first time during the user modification.

- *Password expiration*
  If the password policy is activated for the user, a date is displayed here. (For password policy and password expiration, see → Password policy)

- *Account id*
  The unique identifier number of the user. This field cannot be changed.

After specifying the required information and pressing the *Next* button, the *User management - summary* wizard page appears for the final confirmation, see *Figure 9*.

17.1.2.3 Delete existing user

By selecting this option, an existing user can be deleted. The *User management - summary* wizard page appears for the final confirmation, see *Figure 9*.

17.1.2.4 Changing password

The user can change his/her own password by selecting the *Change password…* menu item from the user context menu (right-clicking his/her user home folder node). This opens the *Change password* dialog box, see *Figure 11.* The old password needs to be specified as well as the new one. The user can select the *Show* checkbox in order to see the plain-text password.

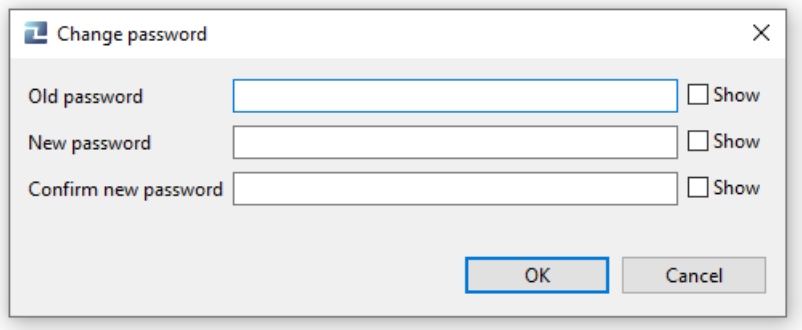


Figure 11 – The *Change password* dialog box

If the password policy is switched on for the given user (see → Password policy), the new password needs to satisfy the conditions of the password policy.

An administrator user can change the password for all the users. In this case, the old password of the given user does not need to be specified, see *Figure 12.*

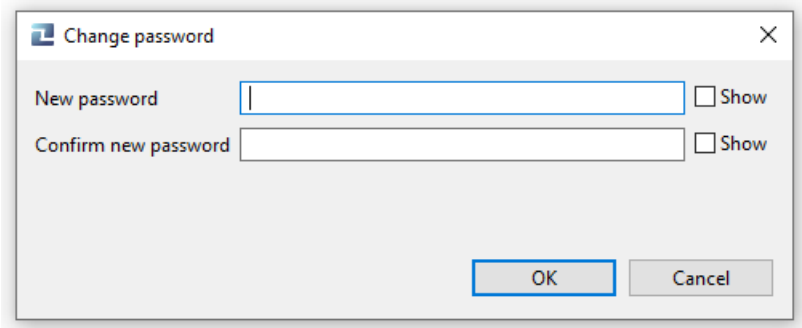


Figure 12 – The *Change password* dialog box for an administrator user

17.1.3 Cancel all jobs

It opens a *Cancel jobs* dialog box for cancelling jobs based on their status, see *Figure 13*.

For further information about job cancelation methods in Zagreus System, see → Cancellation.

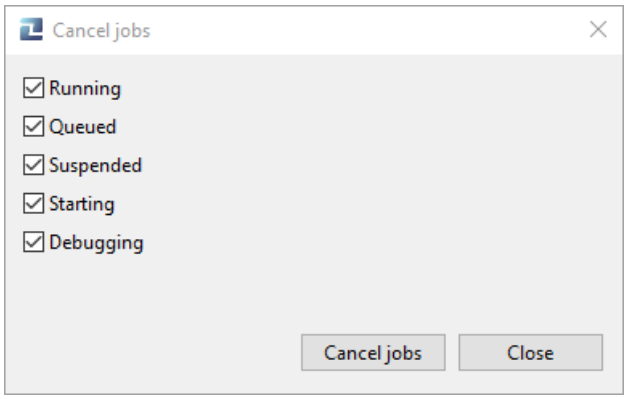


Figure 13 – The *Cancel jobs* dialog box

17.1.4 Stop / start server components

It controls components of the Zagreus server. This is the very same dialog as displayed in the Zagreus Monitor; for a detailed description, see → Additional options.

17.1.5 Manage certificates

It opens the *Manage certificates* dialog box, see *Figure 14*. With this dialog, the user can manage the SSL certificates installed on both the Zagreus server and the Zagreus Worker Controller side, see → Security. There are two tabs by default: *Zagreus Server* and *Zagreus Worker Contoller 1* (the current 1.5.6.0 version of Zagreus supports only one Worker Controller module). The two tabs behave in an identical way, the only difference is the location of the truststore file.

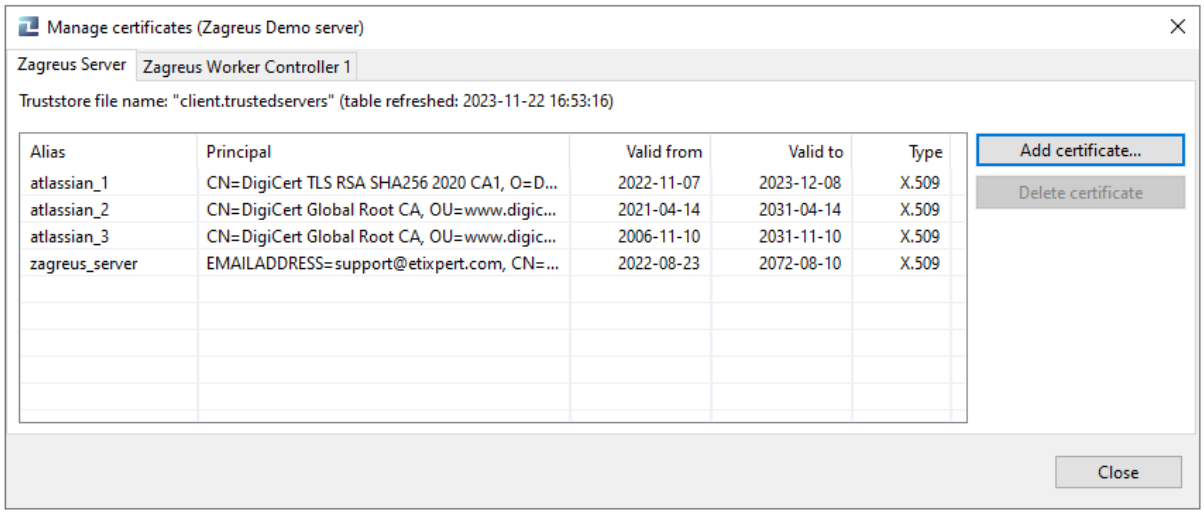


Figure 14 – The *Manage certificates* dialog box

On the top of the tab item, the truststore file name for the selected module is displayed. The most recent time when the truststore file was loaded is also indicated; after all certificate operations, it is properly refreshed.

The table displayed on the tab item contains the following columns:

- *Alias*
  The alias of the certificate, loaded from the truststore file.

- *Principal*
  The subject distinguished name of the certificate.

- *Valid from*
  The start date of the certificate validity.

- *Valid to*
  The end date of the certificate validity.

- *Type*
  The type of the certificate; in most cases it is X.509.

There are color codes for the table: red indicates an expired certificate, while orange warns that the certificate will expire within a month.

There are two operations that the user can perform via this dialog box:

- *Add certificate…*
  The user has to select a certificate file from the local OS filesystem via the opened file browser. This file can be BASE64-encoded or binary type; the extension of the binary file must be `.der`. Also, this file can contain multiple certificates (certificate chain).
  After selecting the certificate file, the needs to specify an alias in the *Select an alias* dialog box, see *Figure 15*. Be aware of the fact that if the certificate file contained a certificate chain, they will be installed separately, and the given alias will be used with different ordering number suffixes, see the *atlassian* alias in *Figure 14*.

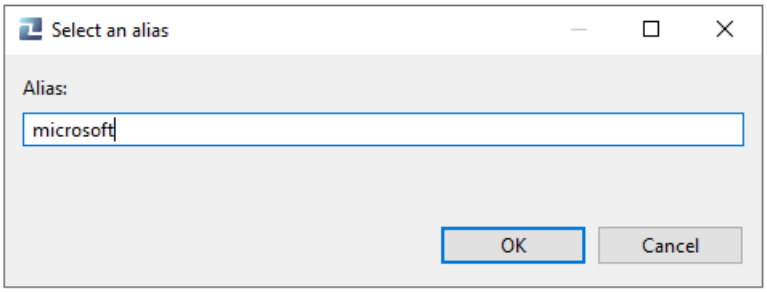


Figure 15 – The *Select an alias* dialog box

After pressing the *OK* button, the user needs to confirm the operaton, and the certificate will be installed on the selected module. The main table on the tab item will be refreshed accordingly. If there is any problem with a given certificate (technical issues or the certificate is already expired), an error message is displayed.

- *Delete certificate*
  The user first needs to select a particular certificate by selecting a table row, then needs to click on this button. After confirming the deletion operation, the selected certificate will be removed from the truststore. The main table on the tab item will be refreshed accordingly.

17.1.6 Monitor watchers, triggers

Clicking on this menu item opens the *Monitoring watchers and triggers* dialog box, where the user can monitor the watchers and triggers on the selected Zagreus server. There are three tabs on this dialog box: *Watchers*, *File Trigger* and *DB Connection Pool*.

17.1.6.1 Watchers tab

Mail and database watcher resources (see → Mail watcher and → Database watcher) together are listed on this tab, see *Figure 16*.

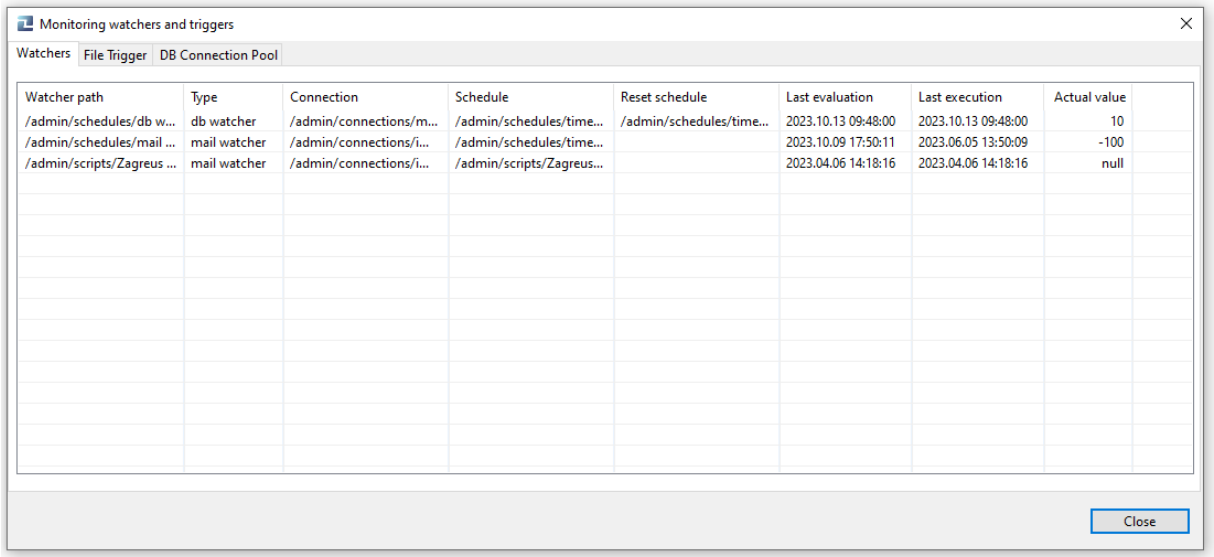


Figure 16 – The *Watchers* tab of the *Monitoring watchers and triggers* dialog box

The following columns are displayed in the main table:

- *Watcher path*
  The full path of the watcher.

- *Type*
  The type of the watcher. It can be *db watcher* or *mail watcher*.

- *Connection*
  The full path of the connection resource associated with the watcher.

- *Schedule*
  The full path of the time schedule resource associated with the watcher.

- *Reset schedule*
  If a *Reset time schedule* resource (see → Connection section) is set for the watcher, the full path of that time schedule resource is displayed.

- *Last evaluation*
  The most recent evaluation date of the watcher (if any).

- *Last execution*
  The most recent execution date of the watcher (if any).

- *Actual value*
  The actual value of the counter in the watcher. This value is also shown in the *Zagreus mail watcher editor* (see → Scheduling section) and *Zagreus DB watcher editor* (see → Scheduling section).

17.1.6.2 File Trigger tab

File trigger resources (see → File trigger) are listed on this tab, see *Figure 17*.

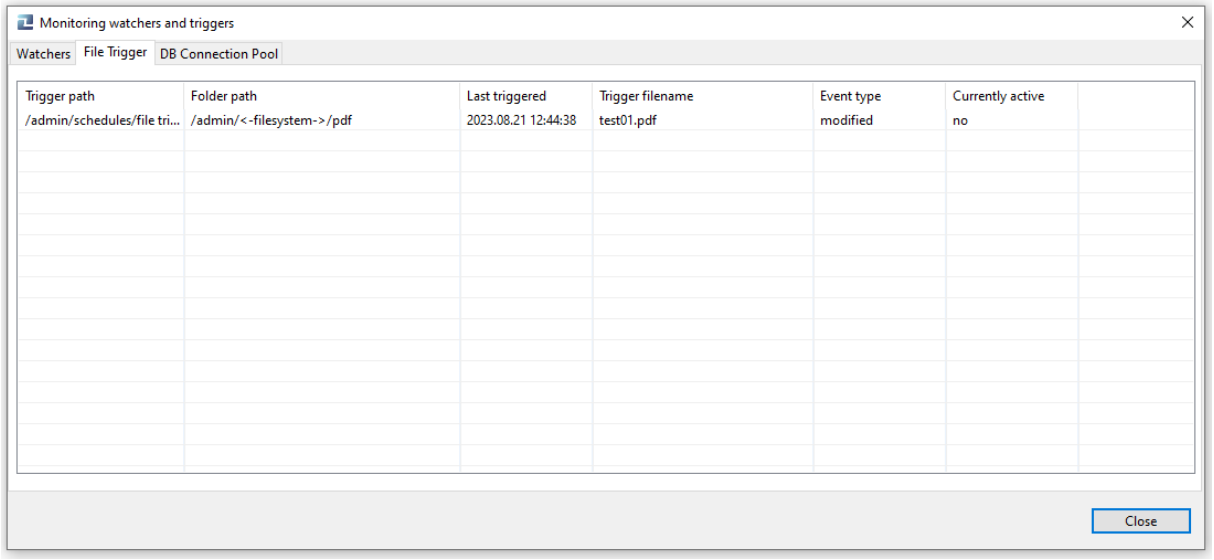


Figure 17 – The *File Trigger* tab of the *Monitoring watchers and triggers* dialog box

The following columns are displayed in the main table:

- *Trigger path*
  The full path of the file trigger.

- *Folder path*
  The full path of the folder monitored by the file trigger.

- *Last triggered*
  The most recent date on which the file trigger was activated.

- *Trigger filename*

The filename of the OS file that activated the file trigger most recently.

- *Event type*
  The type of event that activated the file trigger most recently. Can be *created*, *modified*, *deleted* and *exists*.

- *Currently active*
  If the file trigger is currently active.

17.1.6.3 DB Connection Pool tab

Database connections kept open by database watchers (with the *Keep alive connections in the background…* setting checked, see → Connection section) are listed on this tab, see *Figure 18*.

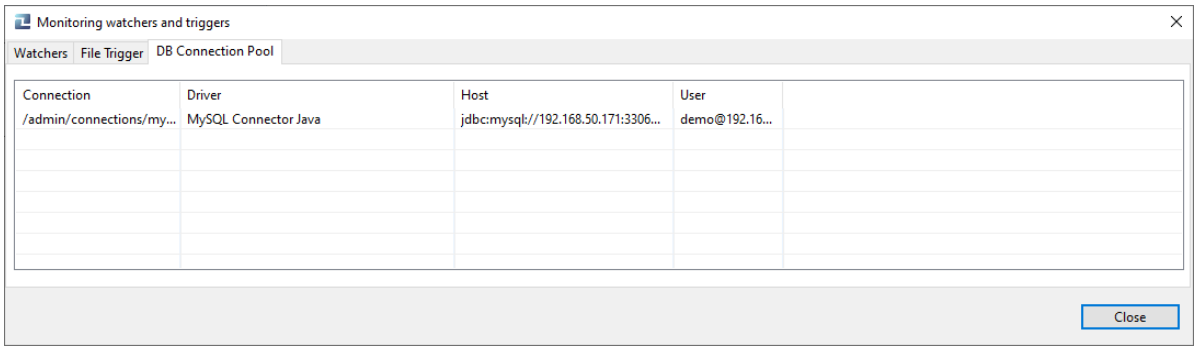


Figure 18 – The *DB Connection Pool* tab of the *Monitoring watchers and triggers* dialog box

The following columns are displayed in the main table:

- *Connection*
  The path of the connection resource associated with the database watcher.

- *Driver*
  The name of the database driver Java library of the open database connection.

- *Host*
  The JDBC connection string.

- *User*
  The database user of the open connection.

17.1.7 Configuration testing

The configuration settings of the main Zagreus modules such as the Zagreus Server, the Zagreus Worker Controllers and the Zagreus Workers can be tested by selecting the *Configuration testing…* menu item. The modules are able to self-test themselves by running a specific testing procedure, and the results of these procedures are displayed in the *Self-test results* dialog box.

The particular property files involved in the self-testing procedures are:

- *Zagreus Server*
  `zagreus_home/server/conf/conf.properties`
- *Zagreus Worker Controller*
  `zagreus_home/worker-controller/conf/workercontroller.properties`
- *Zagreus Workers*
  `zagreus_home/worker-controller/worker/conf/worker.properties`

In the result, errors and warnings may appear in case of any misconfiguration (e.g. unknown parameter or wrong value). An example for a self-test result for all three modules can be seen in *Figure 19*.

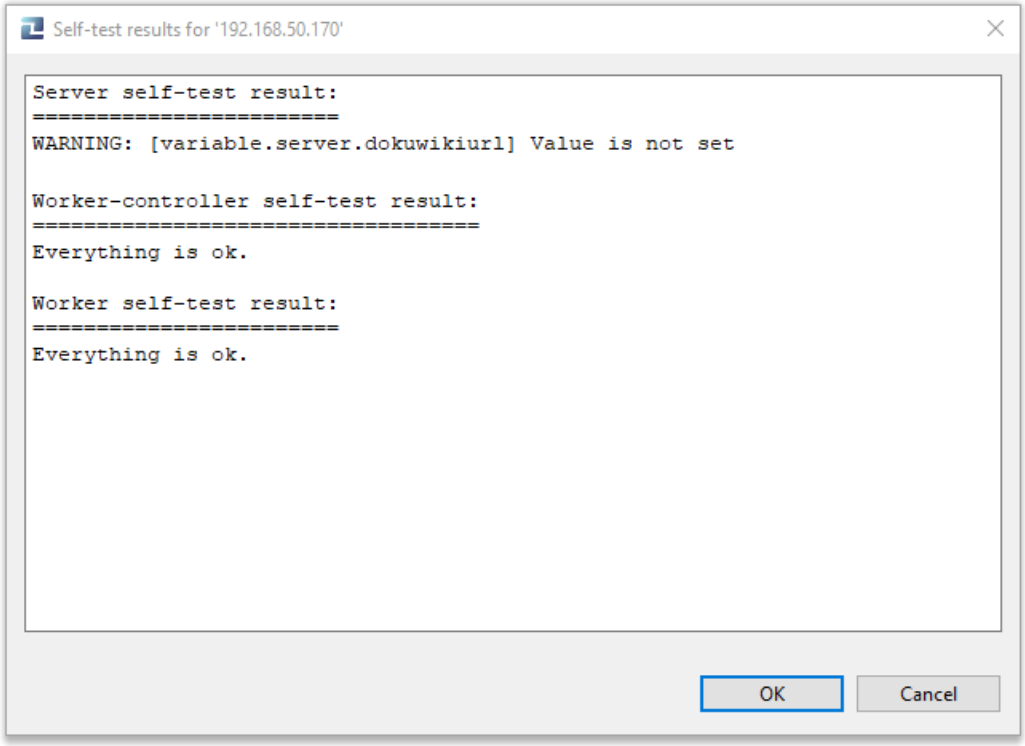


Figure 19 – The *Self-test results* dialog box

17.2 Get licence information

This menu item opens the *Licence info* dialog box, which shows information about the active licence installed on the Zagreus Server, see *Figure 20*. This dialog box can be opened even when the server is disconnected.

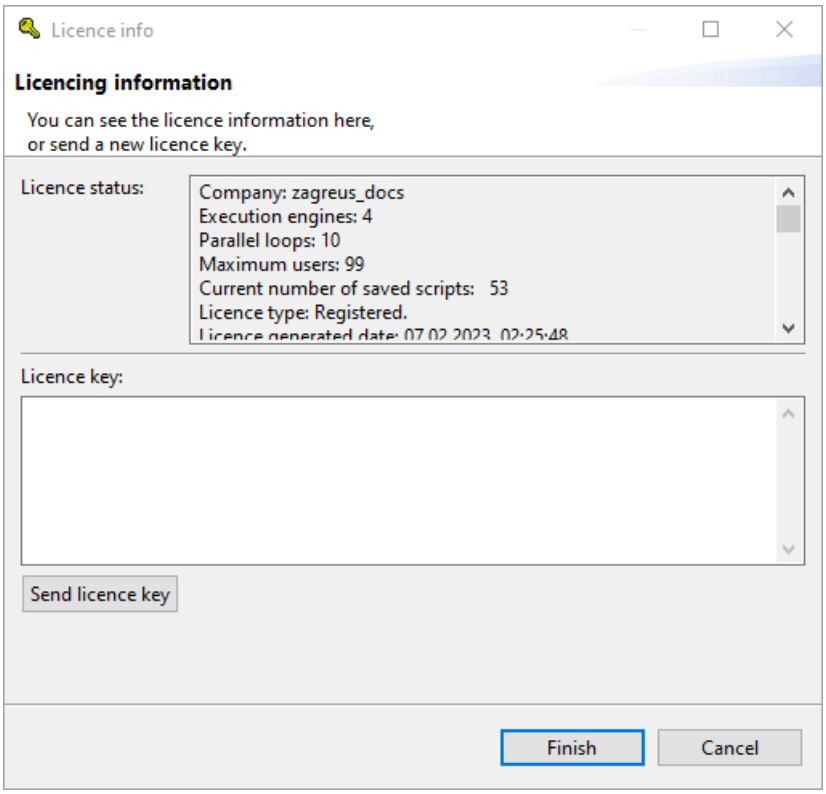


Figure 20 – The *Licence info* dialog box

The *Licence status* textbox contains information about the licence, see → Licencing.

The user can install a new licence key to the selected Zagreus Server by pasting it into the *Licence key* textbox and pressing the *Send licence key* button. After the new licence key has been installed on the server, the content of the *Licence status* textbox will be refreshed.

17.3 Server information

The *Server information* dialog box displays version and uptime information about the actual Zagreus Server, see *Figure 21*.

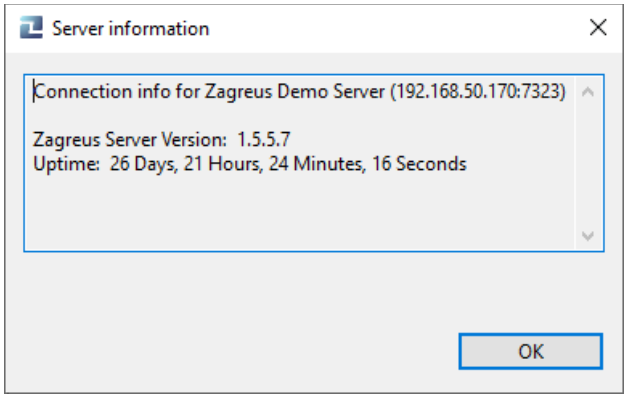


Figure 21 – The *Server information* dialog box