Knowledge Base

Fabasoft app.telemetry Knowledge Base

Contents

1 Known Issues

This chapter lists general known issues by Fabasoft app.telemetry encompassed with the recommended solution.

1.1 Connection to Agent Refused

Issue:

The status of a Fabasoft app.telemetry agent is displayed as "*not connected*" in the agent view of the app.telemetry client.

The error message text is similar to the following:

```
Connection refused because of multiple active connections
```

This situation is caused by multiple connections to a single Fabasoft app.telemetry agent. Multiple connections to a single Fabasoft app.telemetry agent might occur because one of the following configuration issues:

1. More than one Fabasoft app.telemetry server connect to the same Fabasoft app.telemetry agent.
2. One Fabasoft app.telemetry agent is configured multiple times within one Fabasoft app.telemetry server installation.

Solution:

To resolve the situation, ensure that following criteria apply:

1. Each Fabasoft app.telemetry agent is dedicated to one specific Fabasoft app.telemetry server.
2. All agents configured in the Fabasoft app.telemetry client connect to dedicated Fabasoft app.telemetry agents (e.g. no duplicate "*network address*" properties).

1.2 Turn on Crash Dumps on Linux

Issue:

The Fabasoft app.telemetry Linux services support as any other Linux process to produces a crash dump. With the default configuration this feature is disabled but can be enabled with a simple configuration file.

Solution:

To turn on crash dumps for app.telemetry Linux services (Agent, Server) follow the described steps below:

1. Open and edit the *sysconfig* configuration file for the `apptelemetryagentd` or the `apptelemetryserverd` in order to change the setting for the specified process. The files are self-explaining. Just uncomment or change the appropriate lines and restart the processes by means of using the init script.
   - `/etc/sysconfig/apptelemetryagentd`
   - `/etc/sysconfig/apptelemetryserverd`
2. 'Enable core files' by setting the core file limit to unlimited or any other reasonable value.
   - for Red Hat-based Linux systems: activate/uncomment the following line:
   - `DAEMON_COREFILE_LIMIT='unlimited'`
3. Set the core file location (optional):
   - 3.1. default (if not set explicitly) the system will write the core files two one of the following locations (depending on the startup kind of the process - during system boot or manually)
     - Processes started during system boot will be dumped to the root file system folder "`/`" (`/core.1234`)
     - Processes started manually by a user using the init script will be dumped to the current folder
     - In some cases the dump file may be placed next to the binary file (`/opt/app.telemetry/bin`)
   - 3.2. You can explicitly set the dump file location via the kernel `core_pattern` setting
     - Create the desired target directory and make it world writeable:
       ```
       mkdir /tmp/dumps
       chmod 777 /tmp/dumps
       ```
     - set the core file pattern in the kernel variable (file)
       ```
       echo "/tmp/dumps/core" > /proc/sys/kernel/core_pattern
       ```
       the default pattern is "`core`" which will place a file starting with the name "`core`" in the default location described above.
4. Restart the processes by means of using the init script
   - `/etc/init.d/apptelemetryagentd restart`
   - `/etc/init.d/apptelemetryserverd restart`

Finding core files somewhere on your Linux system:

If you don't know where your core files are placed and you know there must be a core file somewhere on your system, you could try finding the core file with the following find statement:

```
find / -mount -type f -regex ".*/core\.[0-9]+$"
```

1.3 SNMP Counter not Available

Issue:

It may occur that some Fabasoft app.telemetry checks return the message "*SNMP-Counter (...) not available*". This message indicates that the desired SNMP counter could not be checked by the Fabasoft app.telemetry agent.

The numeric value in brackets in the message (for example: `.1.3.6.1.4.1.2021.4.15.0`) is the SNMP OID of the defined counter check that is queried to get the resulting value. If there are more than one of such messages listed, the counter value is calculated based on a formula consisting of several SNMP values.

The case that SNMP-Counters are not available can have different problem causes:

1. no SNMP daemon installed on target system (package net-snmp is missing)
2. SNMP daemon not running on target system
3. SNMP configuration invalid to get that counter
   - 3.1. security restrictions: SNMP community string mismatch
   - 3.2. security restrictions: SNMP security view does not allow accessing that counter
4. app.telemetry counter check configuration invalid for that target system – counter or instance does not exist

Note: many app.telemetry counter checks for unavailable SNMP counters with a short update interval may decrease the app.telemetry agent performance, so check your counter configuration carefully!

Solution:

To resolve this problem follow the solution tips for the appropriate problem item (item number from issue list above) and test the configuration as described below.

1. If no SNMP daemon is installed, install the software-package "net-snmp-5.x.<arch>.rpm"
2. If SNMP daemon is not running, start the daemon by means of using the init script "`/etc/init.d/snmpd`".
   Additionally you should set the SNMP daemon to autostart at system startup "`chkconfig --level 345 snmpd on`"
3. Check the SNMP configuration / security restrictions ... (`/etc/snmp/snmpd.conf`)
   - 3.1. Configure SNMP authentication: configure SNMP community string on both sides. The SNMP community string set in the `snmpd.conf` configuration (`rocommunity <string>`) has to match the defined community string in the app.telemetry agent configuration (app.telemetry web client > edit mode > target agent > SNMP community
   - 3.2. Configure SNMP view restrictions: you should include all required SNMP counters in your SNMP configuration
     - 3.2.1. The simplest configuration is to disable all access control configuration lines in the configuration file and only set the SNMP community string with the following configuration line: "`rocommunity <string>`"
     - 3.2.2. Another possibility is to include all counters and do not restrict any counters by means of using the following configuration line: "`view systemview included .1`"
     - 3.2.3. The most secure configuration is to include only the required SNMP subtrees for your desired counters, but then you need to find out all required SNMP OIDs.

To test the SNMP configuration and the availability of the SNMP counters you need to have the additional SNMP software package "`net-snmp-utils`" installed on one of your Linux systems that could connect to the problematic target system (for example you could install these utilities on the app.telemetry server).

All you have to do is run the following commands with your concrete values set:

Linux Shell – Test SNMP Example

```
snmpwalk -v 1 -c <COMMUNITY-STRING> <TARGET-AGENT-IP> 1.3.6.1.2.1.25.1
snmpwalk -v 1 -c <COMMUNITY-STRING> <TARGET-AGENT-IP> 1.3.6.1.4.1.2021

# example ...
snmpwalk -v 1 -c public 10.20.30.40 1.3.6.1.2.1.25.1
snmpwalk -v 1 -c public 10.20.30.40 1.3.6.1.4.1.2021
```

1.3.1 SNMP Counter hrSystemProcesses not Available

Issue:

It may occur that some Fabasoft app.telemetry checks for getting the number of running system processes return the message "*SNMP-Counter .1.3.6.1.2.1.25.1.6.0 not available!”*. This may be a predefined app.telemetry counter check named “*System: Running Processes*” or a self-defined SNMP counter check for that counter.

This message indicates that the desired SNMP counter could not be checked by the Fabasoft app.telemetry agent because the counter is potentially really not available from net-snmp.

Cause:

You have upgraded your RHEL/CentOS installation from version 6.5 to version 6.6 and therefor got an updated net-snmp package version (`net-snmp-5.5-49.el6_5.2` or `net-snmp-5.5-49.el6_5.3`) containing a bug described in the following bug reports causing the SNMP counter `HOST-RESOURCES-MIB::hrSystemProcesses.0` to be not available:

- https://bugzilla.redhat.com/show_bug.cgi?id=1134335
- http://bugs.centos.org/view.php?id=7608

Solution:

Check the status of the bug reports mentioned above and look for an updated net-snmp RPM package solving that bug.

Or downgrade to the older net-snmp RPM package version (`<= net-snmp-5.5-49.el6_5.1`) from RHEL/CentOS 6.5.

1.4 app.telemetry Client not Working after Installation

Issue:

After installation of Fabasoft app.telemetry on a new Linux system and trying to access the Fabasoft app.telemetry client with your web browser the client may show an info dialog with the message that something is not working properly.

The following reasons could prevent the Fabasoft app.telemetry client from working properly:

1. Fabasoft app.telemetry server is not running
2. SELinux is running in “enforcing” mode and the app.telemetry SELinux policies have not been installed or applied correctly.
3. The app.telemetry connection/listening ports of server/webserver are misconfigured

Solution:

The solutions for the reasons above (same numbered item) are listed below:

1. Start the Fabasoft app.telemetry server service/daemon and check if it is set to automatic startup
2. SELinux Troubleshooting
    - 2.1. If you don’t really need SELinux, just turn it off or to permissive mode
        - Check SELinux state: `sestatus`
        - Change to permissive mode: `setenforce 0`
        - Change SELinux configuration mode for startup (persistent) – modify `/etc/selinux/config`
    - 2.2. Check the setup output of the Fabasoft app.telemetry RPM installation and see if any Warnings/Errors occurred during setup /var/log/app.telemetry/<server|agent|webapi>-rpm.log
        - On installation errors you might have not installed the required RPM packages for SELinux policy file management (`policycoreutils-python`) – install the missing packages and reinstall the app.telemetry RPM files
    - 2.3. Maybe you have to restart the app.telemetry daemons to get the SELinux policies be applied
    - 2.4. Check the SELinux audit log and grep for denied log entries `cat /var/log/audit/audit.log | grep “AVC”`
3. Check if the app.telemetry server and agent are running and listening on their configured ports by means of using the `netstat` tool (apptelemetryserver default port 10000, apptelemetryagent default port 10001)

1.5 Security Warning/Restriction for End-2-End Instrumentation

Issue:

The Fabasoft app.telemetry end-2-end instrumentation for websites via the JavaScript SDK is based on data communication POST-requests between the web site and the Fabasoft app.telemetry web service (WebAPI).

Web browsers have security limitations for POST-requests invoked from JavaScript for different locations (different protocol, target, port) so you may receive warning messages or even all the data transfer with the WebAPI is blocked and no end-2-end instrumentation is possible.

The following table shows the different access possibilities for the example base page:
`http://store.company.com/dir/page.html`

URL	Result	Reason
`http://store.company.com/dir2/other.html`	Success	
`http://store.company.com/dir/inner/another.html`	Success	
`https://store.company.com/secure.html`	Failure	Different protocol
`http://store.company.com:81/dir/etc.html`	Failure	Different port
`http://news.company.com/dir/other.html`	Failure	Different host

Source: https://developer.mozilla.org/en/Same_origin_policy_for_JavaScript

So if you have any troubles with denied POST-requests to the configured app.telemetry WebAPI check and compare the used URLs and follow the hints listed below.

Solution:

In order to use Fabasoft app.telemetry end-2-end instrumentation (via JavaScript SDK) ensure that the instrumented web site is allowed to send the data (POST-requests) to the configured WebAPI url.

1. The simplest solution is to install the app.telemetry WebAPI on the same host (also same port and protocol) that also provides your instrumented web page.
2. Using a Load-Balancer to route the requests correctly via a sub directory.
3. Server-side routing of the requests using a local WebAPI url inside your instrumented web page and some special server-side logic capturing and forwarding all telemetry requests to the WebAPI.
4. Using Apache mod_alias, mod_proxy or mod_rewrite to route the requests.
5. Using an iframe with script content from the target domain and specific client-side code to handle the data.
6. Using a web browser with support for cross-origin resource sharing (https://dvcs.w3.org/hg/cors/raw-file/tip/Overview.html - e.g.: Mozilla Firefox 3.5). This may also require some code modifications.
7. Using „Signed Scripts for Mozilla Firefox“ (http://www-archive.mozilla.org/projects/security/components/signed-scripts.html).

1.6 Feedback Dialog shows two Screenshot Properties

Issue:

When using the Fabasoft app.telemetry feedback dialog (report dialog) via the app.telemetry API and your application is running in a Fabasoft application context with available Fabasoft plugin there are different screenshots provided:

- A native screenshot provided by the Fabasoft plugin with a native screenshot preview opened in a desktop application (this type is also called the legacy native screenshot)
- An HTML5 screenshot (which may also fetch the screenshot data from the Fabasoft plugin if available) providing a full-featured screenshot inline preview with mark/crop/blackout features.

If you can see those 2 different screenshot attachments in your feedback dialog you are still using a legacy configuration when calling the `createDialog`-API-call.

Solution:

In order to disable the legacy native screenshot you have to set the `screenshot` property object to “`enabled:false`” when passing to the `createDialog`-API-call.

1.7 Bar Chart Text Overlapping

Issue:

In some situations your dashboard charts may look improperly (e.g. long bar chart label texts overlap).

Depending on your web browser window size, your dashboard settings (number of columns to display) and your chart settings (height of chart) the chart content may fit into the available space or not.

Solution:

Beside the basic features of reducing the number of dashboard columns, increasing the chart height or reducing the label length (of your service checks) some new improvements may help you solve your problems.

With Fabasoft app.telemetry 2012 Spring Release the chart capabilities have been improved the following way:

- Chart legend labels are limited in their maximal length (depending on the chart width) - anyway you should display the legend only if you need it (for bar charts this information is most times redundant)
- Gauge chart scaling improved
- Bar Charts axis labels improved:
    - Horizontal bar chart: long axis label texts now wrap around (depending on the chart size)
    - Vertical bar chart: the axis label texts can now be rotated to display them diagonal or even vertical to prevent overlapping texts ... for this purpose you can define the axis label rotation angle in the chart properties (edit view) with a value between -90° and 90° (negative angle values (e.g.: -30) are displayed left of the bar and positive values (e.g.: 45) are displayed right of the bar, a value of 0 displays the labels horizontal as before)

2 Feature Details

This chapter contains technical and informational descriptions for features of Fabasoft app.telemetry not described in the “Installation Guide” or somewhere else.

2.1 Virtual Host Detection (VMware ESX)

With Fabasoft app.telemetry it is possible to get information about the location of a virtualized system which uses VMware ESX-Server virtualization technology.

Note from "*vSphere Basic System Administration Guide*": (ESX/ESXi 4)

> ESX/ESXi includes an SNMP agent embedded in hostd that can both send traps and receive polling requests such as GET requests. This agent is referred to as the embedded SNMP agent.
>
> Versions of ESX prior to ESX 4.0 included a Net-SNMP-based agent. You can continue to use this Net-SNMP-based agent in ESX 4.0 with MIBs supplied by your hardware vendor and other third-party management applications. However, to use the VMware MIB files, you must use the embedded SNMP agent.
>
> By default, the embedded SNMP agent is disabled. To enable it, you must configure it using the vSphere CLI command vicfg-snmp.

Note: VMware does not officially support SNMP (GET requests) with ESXi products but since ESXi 4 it is possible to obtain the required SNMP counters.

The Fabasoft app.telemetry virtual host detection feature requires the VMware SNMP counters provided by the embedded VMware SNMP agent.

Enable the VMware SNMP support in one of the following ways:

- using the embedded VMware SNMP agent only:
  - use the vSphere remote CLI interface and turn on the embedded VMware SNMP (using the default port and setting a SNMP community)
  - for details see the VMware guide "*vSphere Basic System Administration*"
- or using the embedded VMware SNMP agent in combination with the Net-SNMP agent
  - for details about this configuration method see the VMware technical note "*Configuring the Net-SNMP Agent on ESX Hosts*"

To turn on SNMP support for VMware ESXi 4 use the vSphere remote CLI interface and turn on the embedded VMware SNMP agent which provides the required SNMP counters for VMware products

Test SNMP Configuration for VMware ESX/ESXi 4.0:

To test the correct SNMP configuration of VMware vSphere (ESX 4.0) / ESXi 4.0 servers check if the following SNMP counters are available from a remote system (especially from the app.telemetry proxy agent used for the VMware ESX server VM host detection):

- .1.3.6.1.4.1.6876: VMware root MIB
- .1.3.6.1.4.1.6876.1: VMware product information MIB (for product name and version)
- .1.3.6.1.4.1.6876.2: VMware VM information MIB (for VM detection)

Check the existence of the SNMP counters on the VMware ESX server from your proxy agent system by means of:

```
snmpwalk -v 1 -c <your_community> <ESX_server_IP> .1.3.6.1.4.1.6876
```

```
Example: snmpwalk -v 1 -c public 10.20.30.40 .1.3.6.1.4.1.6876
```

2.2 Special Notification Channels (Command Line)

The Fabasoft app.telemetry notification system is based on a notification channel defining the way how to send a notification and several notification accounts which are notified of status changes. The following notification channel types are supported:

- **SMTP** (E-Mail server)
  - Microsoft Windows: directly via SMTP
  - Linux: indirectly via local `sendmail` process
- **Command Line**: using any defined command line to send a notification

To set up the notification system correctly you have to create a notification channel first and then create sub elements of type notification account inside the notification channel.

Command Line Notification Channel:

Create a notification channel, choose "*Command Line Notification*" and define the notification command line to be executed on any status change. The command line consists of the absolute path of the command or script (on the Fabasoft app.telemetry server) and additional parameters passed to the command or script. Parameters with spaces must be quoted ("). The following variables can be used to pass concrete values to the notification command:

- **%FILE** … will be replaced with the temporary filename of the notification template filled with the current values of the current notification.
  - Example: "/tmp/app.telemetry/128920567310000000_1247583131572241.tmp"
- **%TO** … will be replaced with the "TO"-address of every configured notification account.
  - Example: "0664123456789"
- **%SUBJECT** … will be replaced with the current notification subject value.
  - Example: "*Service Check \"crond availability check\" changed to ok*"
- **%AGENTHOSTNAME** … will be replaced with the hostname of the agent where the status change came from (if possible).

An example of such a command line looks like:

Example: Command Line Notification

```
/opt/app.telemetry/bin/echonotification.sh %FILE %TO %SUBJECT
%AGENTHOSTNAME
```

… this will result in the following call:

```
/opt/app.telemetry/bin/echonotification.sh
  "/tmp/app.telemetry/128920567310000000_1247583131572241.tmp"
  "0664123456789"
  "Service Check \"crond availability check\" changed to ok"
  "examplehostname"
```

Set Timing Options for Notifications:

For some situations or in special installations you may need to tune some timing options for the notification system:

By default notification status is processed every 10 seconds starting 60 seconds after the app.telemetry Server service started. You may modify these settings by adding attributes to the respective *NotificationChannel* element in the `infra.xml`. To modify the notification interval, add an attribute `scheduleSeconds` with a value between 1 and 600 in seconds as the interval. To modify the time between the service start and the first notification, add an attribute `delayOnStartSeconds` with a number of seconds to wait between 0 and 3600. These two parameters cannot be changed at runtime

Example: Notification Channel Configuration

```
<NotificationChannel id="100123" name="Mailserver" status="0" type="smtp"
delayOnStartSeconds="300" scheduleSeconds="20">

    <Param key="Authenticate" value="Anonymous"/>
    <Param key="SendEmailAddress" value="notifications@domain.com"/>
    <Param key="Server" value="10.20.30.40"/>
    <Param key="ServerPort" value="25"/>

</NotificationChannel>
```

... this configuration will delay the notification processing for 5 minutes after server start (instead of 1 minute default) and process the notifications every 20 seconds (instead of default every 10 seconds).

2.3 Notification Templates

Fabasoft app.telemetry allows customization of notification templates. The notification template files are located at following location:

- Microsoft Windows: `<PROGRAMDATA>\Fabasoft app.telemetry\*template.txt`
  - ... the default files installed with the Setup package are named `*template.txt.template` - to customize those files make a copy of the files and store them without the `.template` suffix.
- Linux: `/etc/app.telemetry/*template.txt`
  - ... Linux manages these files as RPM configuration files, so they won't get overwritten on software upgrade if changed by user.

The following template files exist:

- for status-change notifications:
  - `commandlinetemplate.txt` ... used for notifications sent via command line notification channel
  - `mailtemplate.html` ... used for notifications sent via e-mail notification channel
- for feedback notifications:
  - `escalationcommandlinetemplate.txt` ... used for notifications sent via command line notification channel
  - `escalationmailtemplate.html` ... used for notifications sent via e-mail notification channel

Those template files support some substitution variables that will be replaced with the current value for that notification. These variables are written in the templates with following escape-syntax: "`<%VARIABLENAME%>`" (e.g.: `<%TIMESTAMP%>`).

Status-Change Notification Templates:

The following substitution variables exist for status-change notification templates:

- `NOTIFICATIONCLASS`: element type responsible for notification change (Service Group, Service or Service Check)
- `NAME`: name of element responsible for notification change
- `STATUS`: current status of element type responsible for notification change
- `PREVIOUS_STATUS`: previous status before this status change (status changed from `PREVIOUS_STATUS` -> to `STATUS`)
- `LOCALTIMESTAMP`: timestamp when the status change happened (in local time - time zone set on app.telemetry server)
- `TIMESTAMP`: timestamp when the status change happened (in UTC)
- `SUBNODES`: hierarchical structure of nodes affected with this status change ... for details see example below

Example: Status Change Notification Template (`commandlinetemplate.txt`)

```
<%NOTIFICATIONCLASS%> "<%NAME%>" changed to <%STATUS%>

Fabasoft app.telemetry notification Message

The status of <%NOTIFICATIONCLASS%> "<%NAME%>" changed from
<%PREVIOUS_STATUS%> to <%STATUS%>.

Date: <%LOCALTIMESTAMP%>, <%TIMESTAMP%>
```

```
<%SUBNODES%>

Reason

<%GROUP%>  Service group "<%NAME%>" reported status <%STATUS%>

<%/GROUP%><%SERVICE%>  Service "<%NAME%>" on agent <%HOSTNAME%> reported
status <%STATUS%>

<%/SERVICE%><%CHECK%>  Service Check "<%NAME%>" reported <%VALUE%>
<%MESSAGE%>

<%/CHECK%><%SERVICEPOSTFIX%>  --

<%/SERVICEPOSTFIX%><%GROUPPOSTFIX%>

---

<%/GROUPPOSTFIX%><%/SUBNODES%>
```

Escalation/Feedback Notification Templates:

The following substitution variables exist for status-change notification templates:

- `SUBJECT`: "Feedback Notification for Application <*LOGPOOL*> from <*USER*>" ... contains the user (name/email) of the user who submitted the feedback and the application/logpool where the feedback belongs to

Since product version 2014 Fall Release you can also customize the notification e-mail subject by means of replacing the predefined <*SUBJECT*>-tag with a custom template value consisting of raw text in combination with any desired other property value.

Here is an example based on feedback forms having a field with name “Message”:

```
<title>Feedback via form <%FORMNAME%> from user <%FROM%> with message:
<%PROPERTY%>Message<%/PROPERTY%></title>
```

- `MESSAGE`: the message entered by the user with his feedback
- `FROM`: "name <email>" using feedback field "*name*" and field "*email*" to fill this values
- `URL`: the web page URL the feedback was sent from
- `FILTER`: session filter that was used for the feedback session
- `LOCALTIMESTAMP`: timestamp when feedback was submitted (in local time - time zone set on app.telemetry server)
- `TIMESTAMP`: timestamp when feedback was submitted (in UTC)
- `LOGPOOLNAME`: name of logpool/application where the feedback belongs to
- `LOGPOOLID`: internal ID of logpool/application where the feedback belongs to
- `SESSIONID`: session ID of this feedback session (in order to open session directly via link in client - requires hyperlink URL hash properties: "#swt-view!logType:4!logId:<%SESSIONID%>")
- `APPLICATIONNAME`: appname of registered application which triggered the feedback session
- `APPLICATIONID`: appid of registered application which triggered the feedback session
- `APPLICATIONTIER`: apptiername of registered application which triggered the feedback session
- `FORMNAME`: name of feedback form which triggered the notification
- `SESSION_PROPERTY_LIST`: a list of all additional feedback property fields
- `PROPERTY`: to obtain the value of any desired session property using *<%PROPERTY%>property-name<%/PROPERTY%>*

- `ADD_FILES_AS_ATTACHMENTS`: adds all in the feedback containing files (screenshot, systeminfo) as e-mail attachment to the mail notification. The location of this tag in the template is not relevant as it is not inlined but added as attachment.

Example: Escalation/Feedback Notification Template (`escalationmailtemplate.html`)

```
<html>
  <head>
    <meta name="generator" content="Fabasoft app.telemetry" />

    <title><%SUBJECT%></title>

    <style type="text/css">
      BODY { font-family: Arial; font-size: 10pt; background-color:
#F5F5F5; }
          h1 { font-size: 13pt }
          h2 { font-size: 11pt; margin-bottom: 0.3em; }
      div, a { margin-left : 10px; }
    </style>

  </head>

  <body>

    <h1>Fabasoft app.telemetry feedback notification</h1>

    <h2>Message</h2>
    <div><%MESSAGE%></div>

    <h2>Date</h2>
    <div><%LOCALTIMESTAMP%></div>
    <div><%TIMESTAMP%></div>

    <h2>Application</h2>
    <div><b>Logpool: </b><%LOGPOOLNAME%></div>
    <div><b>Application Tags: </b><%APPLICATIONNAME%> - <%APPLICATIONID%>
- <%APPLICATIONTIER%></div>

    <h2>Sent by</h2>
    <div><%FROM%></div>

    <h2>Sent from</h2>
    <div><b>URL: </b><%URL%></div>
    <div><b>Filter: </b><%FILTER%></div>

    <h2>Feedback Infos</h2>
    <div><%SESSION_PROPERTY_LIST%></div>

    <h2>Open/View Feedback Session</h2>
    <div>Click the link below to view more details of this feedback in
your Fabasoft app.telemetry client:</div>
    <a href="http://apmserver.mydomain.com/apptelemetry/index.html#swt-
view!logType:4!logId:<%SESSIONID%>">Open Feedback Session</a><br/>

  </body>
</html>

<%ADD_FILES_AS_ATTACHMENTS%>
```

2.4 Fabasoft Folio Object Address Resolution

Fabasoft Folio object addresses (a.k.a. *COO-addresses*) are exact but quite meaningless in respect to the character of the object they represent. It is mainly for the sake of optimization that the Fabasoft app.telemetry instrumentation of Fabasoft Folio uses the 64-bit integer representation of the addresses to pass object identity information. Whereas the conversion to the "COO-Address" format has been coded into Fabasoft app.telemetry, a more user friendly way of presenting Fabasoft Folio objects is still available.

Mapping of addresses to Names and References:

Providing an XML file containing a mapping from object address to names or references Fabasoft app.telemetry can represent Fabasoft Folio addresses in human readable format to help users to interpret recorded request information more easily.

Generating the mapping file:

In order to generate the mapping file the "*Integration for app.telemetry Software-Telemetry*" Software Component provides the XSL Transformation file `FSCAPPTELEMETRY@1.1001:GenerateLogAnalyzerData`. Calling this XSL Transformation by a script or by a Fabasoft Expression you receive an XML file containing the addresses and names of the following object classes:

- Software Product
- Software Component
- Component Object (and all derived object classes)
- User
- Group
- Domain

To generate the mapping file start a command line (`cmd.exe` (Microsoft Windows) or `bash` (Linux)) on a Fabasoft Folio Server of your domain with a Fabasoft Folio service user account having permissions to read all objects, set the `HOST` and `PORT` variable to point to the Fabasoft Folio backend service and execute the following command (call the `fsceval` command in one line):

On Linux systems the default service user account is `fscsrv` and the default port of the Fabasoft Folio backend service is *18070*. The `fsceval` binary is located under `/opt/fabasoft/bin/` but should already be available via the PATH-variable without an absolute path.

Run `fsceval` (on Linux) to generate address resolution mapping file.

```
su - fscsrv

HOST=localhost PORT=18070 fsceval -eval
"coouser.COOXML@1.1:XSLTransformObject(coouser,
FSCAPPTELEMETRY@1.1001:GenerateLogAnalyzerData, 'fscdata.xml')"
```

On Microsoft Windows systems you should be logged in with an administrative account (of Fabasoft Folio). Setting the `HOST` (default: *localhost*) and `PORT` (default: *18070*) environment variables is optional and not required for a default installation.

Run `fsceval` (on Microsoft Windows) to generate address resolution mapping file.

```
fsceval.exe -eval "coouser.COOXML@1.1:XSLTransformObject(coouser,
FSCAPPTELEMETRY@1.1001:GenerateLogAnalyzerData, 'fscdata.xml')"
```

Note: In earlier Fabasoft Folio or Fabasoft eGov-Suite installations the component name was `FSCAPPLSTRUDL@1.1001` instead of `FSCAPPTELEMETRY@1.1001`.

The result is a generated fscdata.xml for your domain:

Syntax of `fscdata.xml` mapping files

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<Objects>

    <Object id="COO.1.1.1.2500" name="Fabasoft Folio/Base"
reference="ComponentBase@1.1"/>

    <Object id="COO.1.1.1.9285" name="Fabasoft Folio/Folio"
reference="ComponentFolio@1.1"/>

    ...

</Objects>
```

In earlier versions of the XSL-transformation script shipped with the app.telemetry Software Component some more additional properties (`<Attributes>`-sublist and `<Methods>`-sublist below the `<Object>`-tags) have been generated which are not used for the address resolution and name mapping and can be skipped. The only required entries in the mapping file for name- and reference-resolution are the `<Object …/>`-tags. To remove those not needed old sublist elements you can use grep to exclude all `<Attributes>`-sublist and `<Methods>`-sublist entries:

Exclude not used attributes from mapping file (optional)

```
grep -v "<Attribute" fscdata.xml | grep -v "</Attribute" | grep -v
"<Method"  | grep -v "</Method" > fscdata-small.xml
```

Set up Fabasoft Folio address resolution for Fabasoft app.telemetry:

The Fabasoft app.telemetry web browser client receives the formatted values from the Fabasoft app.telemetry web service, which is therefore responsible for the formatting of the addresses. This implies that the mapping file has to be stored in the configuration folder on the web service under the following path:

- Linux: `/etc/app.telemetry/fscdata.xml`
  (ensure that the apptelemetryworker user can access/read the file)
- Microsoft Windows: `%PROGRAMDATA%\Fabasoft app.telemetry\fscdata.xml`
  (ensure that the app.telemetry Worker user can access/read the file)

Restart the Fabasoft app.telemetry Worker service to read the new content of `fscdata.xml`.

The decision whether the COO-address is mapped to the objects name or reference is defined in the log definition of the corresponding log pool by the column “format” with the values:

- `format="fsc:address"` … pretty-print the COO-address
- `format="fsc:reference"` … print the reference value of the object if found in mapping file (fall-back: COO-address)
- `format="fsc:name"` … print the name value of the object if found in mapping file (fall-back: COO-address)

2.4.1 Merging of Multiple Mapping Files

In special situations one Fabasoft app.telemetry server may be used to monitor multiple Fabasoft Folio domains (e.g. test domain and production domain). Currently the app.telemetry server only supports one global address resolution mapping file (as described in the main chapter).

The solution to get Fabasoft Folio object addresses of different domains resolved together is to merge the separate mapping files (generated for each Fabasoft Folio domain) into one single mapping file.

1. Generate the separate mapping files for each Fabasoft Folio domain
    - 1.1. Resulting in several `fscdata.xml` files (e.g.: `fscdata1.xml, fscdata2.xml, fscdata3.xml, fscdata4.xml`)
2. Remove the basic XML-file header (line 1) and surrounding `<Objects>`-list XML-tags (line 2 and last line)
    - 2.1. First file (`fscdata1.xml`): remove the end tag `</Objects>` from last line of the first file
    - 2.2. File 2 … <n-1> (`fscdata2.xml, fscdata3.xml`): remove XML-header and `<Objects>`-start-tag and </Objects>-end-tag
    - 2.3. File n (`fscdata4.xml`): from the last file only remove the XML-header and `<Objects>`-start-tag
3. Concatenate all files in the same order: `”cat fscdata1.xml fscdata2.xml fscdata3.xml fscdata4.xml > fscdata.xml”`

Or just copy all plain `<Object>`-entries without any surrounding container-tags into 1 single file with the syntax shown in the following example:

Syntax of `fscdata.xml` mapping files

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Objects>
    <Object id="COO.1.1.1.2500" name="Fabasoft Folio/Base"
reference="ComponentBase@1.1"/>
    <Object id="COO.1.1.1.9285" name="Fabasoft Folio/Folio"
reference="ComponentFolio@1.1"/>
    ...
</Objects>
```

Note: Be careful with the file encoding – ensure to edit and save the file with valid encoding (UTF-8).

Duplicate `<Object>`-mapping definitions may occur in the merged file but doesn’t matter. The first `<Object>`-definition for an id (*COO-address*) is used to resolve the entry.

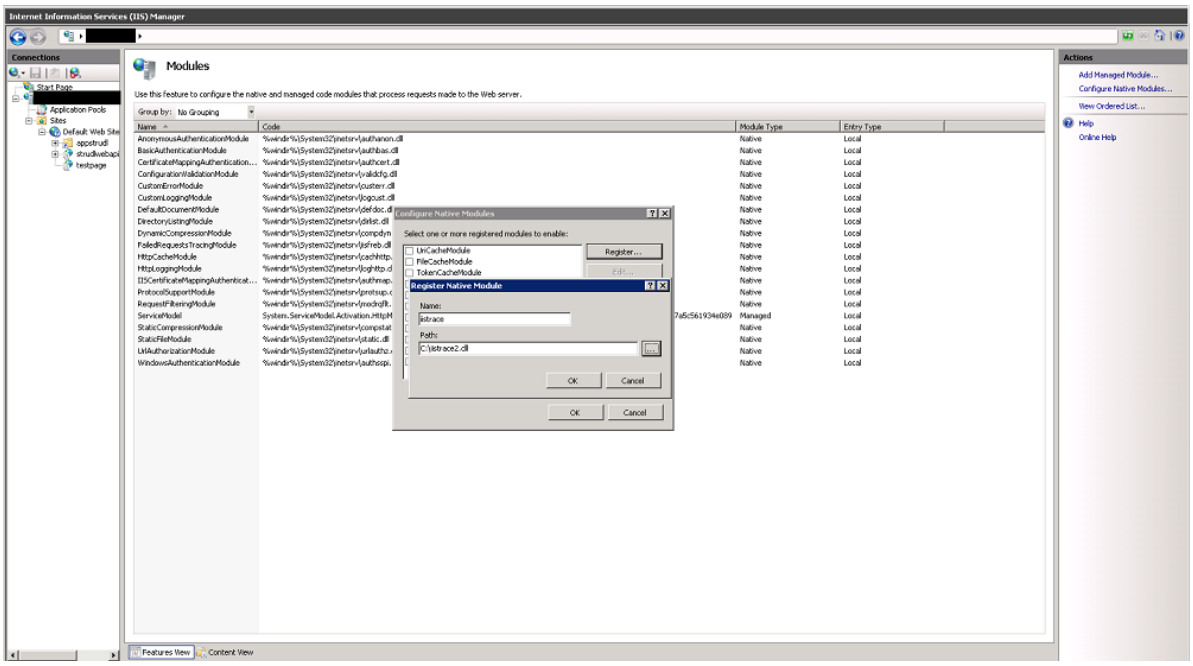
2.5 Configure Software-Telemetry Module for Microsoft IIS

With the native Software-Telemetry module for Internet Information Services (IIS) Fabasoft app.telemetry can log each HTTP-request with some important parameters to a separate Software-Telemetry log pool or the data is shown as extension module in the requests of an involved log pool of another web application. The module shows the start-, and end-time of each request, the time needed for authorization and execution and additional request parameters

Configuration:

To install the module on Microsoft IIS7 web server follow these steps:

1. Get the module files from the app.telemetry installation media from the directory “`Telemetry-Modules\WINDOWS_Xxx`”
2. Place the `telemetryiis7.dll` in any folder on your file system and give it the needed permission settings (the configured user of the web applications application pool must have read and execute access to the file).
3. Open IIS Manager and move to the root node of your web server
4. Select and enter the "*Modules*" feature.
5. Click on "*Configure native module*" inside the action pane on the right edge of IIS manager.
6. Click on "*Register*" in the new configuration dialog.
7. Fill in a custom name and the path of the `.dll` file into the required fields and commit by clicking OK.
8. Disable the module on the root level.



Use this feature to configure the native and managed code modules that process requests made to the Web server.

Name	Code	Module Type	Entry Type
AnonymousAuthenticationModule	%windir%\System32\inetsrv\authanon.dll	Native	Local
BasicAuthenticationModule	%windir%\System32\inetsrv\authbas.dll	Native	Local
CertificateMappingAuthentication...	%windir%\System32\inetsrv\authcert.dll	Native	Local
ConfigurationValidationModule	%windir%\System32\inetsrv\validcfg.dll	Native	Local
CustomErrorModule	%windir%\System32\inetsrv\custerr.dll	Native	Local
CustomLoggingModule	%windir%\System32\inetsrv\logcust.dll	Native	Local
DefaultDocumentModule	%windir%\System32\inetsrv\defdoc.dl	Native	Local
DirectoryListingModule	%windir%\System32\inetsrv\dirlist.dll	Native	Local
DynamicCompressionModule	%windir%\System32\inetsrv\compdyn	Native	Local
FailedRequestsTracingModule	%windir%\System32\inetsrv\iisfreb.dll	Native	Local
HttpCacheModule	%windir%\System32\inetsrv\cachhttp.	Native	Local
HttpLoggingModule	%windir%\System32\inetsrv\loghttp.d	Native	Local
IISCertificateMappingAuthenticat...	%windir%\System32\inetsrv\authmap.	Native	Local
ProtocolSupportModule	%windir%\System32\inetsrv\protsup.c	Native	Local
RequestFilteringModule	%windir%\System32\inetsrv\modrqflt.	Native	Local
ServiceModel	System.ServiceModel.Activation.HttpM 7a5c561934e089	Managed	Local
StaticCompressionModule	%windir%\System32\inetsrv\compstat	Native	Local
StaticFileModule	%windir%\System32\inetsrv\static.dll	Native	Local
UrlAuthorizationModule	%windir%\System32\inetsrv\urlauthz.	Native	Local
WindowsAuthenticationModule	%windir%\System32\inetsrv\authsspi.	Native	Local

Select one or more registered modules to enable:

☐ UriCacheModule
☐ FileCacheModule
☐ TokenCacheModule

Name: iistrace

Path: C:\iistrace2.dll

Enable the module for your web application:

1. Navigate to your web application in IIS manager and enter the "*Modules*" feature.
2. Click on "*Configure native module*" inside the action pane on the right edge of IIS manager.
3. Enable the module in the configuration dialog.

Note: You can enable the module only for your web application or for the complete web site, but be careful not to enable the module on two configuration layers because this will lead to a duplicate error.

Enable context transitions for Browser-Telemetry:

The IIS-Software-Telemetry module supports End-2-End Software-Telemetry by providing a context in a session-cookie. Use the following steps to enable this feature:

1. Configure the IIS schema
    - 1.1. stop IIS (iisreset /stop)
    - 1.2. copy `apmmodule_schema.xml` from the "`Telemetry-Modules\WINDOWS_Xxx`" directory to `%windir%\system32\inetsrv\config\schema`
    - 1.3. edit `%windir%\system32\inetsrv\config\applicationHost.config` to include the "`apmModule`" section under the "`system.webServer`" sectionGroup:

      ```
      <sectionGroup name="system.webServer">
            <section name="apmModule" overrideModeDefault="Allow" />
      …
      ```
    - 1.4. start IIS (iisreset /start)
2. Use Internet Information Service (IIS) Manager to configure the Fabasoft app.telemetry IIS module
    - 2.1. select the respective web site or virtual directory where you want to enable context transitions (you should enable this feature on virtual directories providing instrumented html files)
    - 2.2. double-click "*Configuration Editor*" from the "*Features View*"
    - 2.3. select "`system.webServer/apmModule`" in the "*Section*" property
    - 2.4. change the value of "*provideContext*" to "*True*"
    - 2.5. click "*Apply*"

2.6 Generic Log Definition Columns

Since Fabasoft app.telemetry 2012 Spring Release log pools and log definition columns have been extended to be more powerful and flexible than before.

Log definition columns can be defined as generic/dynamic columns based on other columns obtaining their value by means of evaluating a calculation formula.

Possible calculation types are:

- **categorize** value
- **split** value (regular expression)

Categorize Value

To categorize an existing log definition column value, decide which parent/base column you want to split up into reasonable value categories. This column is defined in the new column definition as `parent`-attribute containing the name of the chosen existing column. You can also choose internal columns like the duration column.

The next step is to define the split points how to separate the value into the categories using the `calculation`-attribute. If you define 3 split points, you will get 4 categories: below the 1st split point, between 1st and 2nd, between 2nd and 3rd and above the 3 split point.

Then you can define textual labels for the categories using the format-attribute containing the keyword "`enum:`" followed by the number of the category, a colon (`:`) and the label text separated with a semi-colon (`;`) from the next category.

Last but not least set the flags of the new defined column including the flag for a `CALC_VALUE_CATEGORY = 0x10000` (decimal=65536). If you want to define the column to be a dimension-column you also have to include that flag (0x0100 / decimal=256).

Syntax for Categorize Value

```
calculation="x1;x2;x3"
format="enum:1:0-x1;2:x1-x2;3:x2-x3;4:&gt;x3"
parent="name of parent column"
name="Category Column Label"
flags="65792"
```

Split Value (Regular Expression)

To split an existing log definition column value into sub parts, decide which parent/base column you want to split up. This column is defined in the new column definition as `parent`-attribute containing the name of the chosen existing column.

The next step is to define the regular expression splitting up the existing string into a new value using the `calculation`-attribute:

- Enclose the desired text part in the regex with grouping braces “`( … )`”
- Define the desired group match number with the `paramid`-attribute (first/only 1 matching group ... `paramid="1"`).

Last but not least set the flags of the new defined column including the flag for a `CALC_REGEXP = 0x20000` (decimal=131072). If you want to define the column to be a dimension-column you also have to include that flag (0x0100 / decimal=256).

Example: Web Timing Dynamic Columns

```
<APMLogAnalyzerEntry  name="Response Time Category"  parent="duration"
  calculation="10000000;40000000;100000000;300000000;600000000"
  format="enum:1:0-1s;2:1-4s;3:4-10s;4:10-30s;5:30-60s;6:&gt;60s"
  flags="65792"  paramid="0"  eventid="0"  module=""
  columndatatype="1" displaypriority="0" displaywidth="50"
tablename="browser" tabletype="1" type="1"/>


<APMLogAnalyzerEntry  name="Protocol"  parent="Page URL (referer)"
  calculation="([\w]+):"  paramid="1"  flags="131328"  eventid="0"
  module=""  columndatatype="3" columnlength="100"
  displaypriority="0" displaywidth="50"  tablename="browser"
tabletype="1" type="3"/>


<APMLogAnalyzerEntry  name="URL Path"  parent="Page URL (referer)"
  calculation="[\w]+://[^/]*([^\?]+)"  paramid="1"  flags="131328"
  eventid="0"  module=""  columndatatype="3" columnlength="200"
  displaypriority="0" displaywidth="120"  tablename="browser"
tabletype="1" type="3"/>
```

2.7 Log Statistics (Daily Statistics)

Since Fabasoft app.telemetry 2012 Spring Release a new database-based statistic feature is available which allows you calculate defined statistics on the available data at a defined time.

For example you can calculate and summarize all the requests from the last day every night and generate significant statistic charts.

Feature Details:

The new log statistics are based on the following components:

- An application *log pool* defining which data is recorded.
- A *database* selected in the log definition storing the basic data (full base data set) and the statistic tables.
- A *log statistic* defining which data is calculated at which time in which way.
- A *chart* based on that log statistic representing the summarized/calculated data.

The *Log Statistic* object defines the following key facts:

- A *time resolution* (in minutes) defining the interval when the statistic is calculated.
- A *time schedule* defining the delayed offset (in minutes) after the interval when the statistic is calculated
- A database table extension (*tableext*) that is appended to the new created statistic database tables (belonging to the database assigned in the log pool). This value must be unique for the statistics selected for a single logpool.
- A list of dimension fields. The statistic result may be grouped by the given dimensions (see "*group by*" in the chart).
- Optional *database filter* defining which requests should not be included for the statistic calculation (the filter has to be defined in SQL syntax, e.g. "`duration < 600000000`")

The Top-X Logpool Statistic Chart defines the following key facts:

- The *log pool* where the base data is taken from
- The *log statistics* where the calculated chart data is taken from
- A *time range* defining how many time intervals will be calculated
- *group by* field. Select the dimension from the Log Statistic dimension field by which to group the values by.
- Optional *database filter* defining which requests should not be included for that chart (the filter has to be defined in SQL syntax and may only cover dimensions included in the Log Statistic)
- A list of displayed *measures* defining which measures will be shown in the chart
- In Chart Data Mode you can specify the time column format (“date”, “time”, “datetime”) if the chart type is “Table”, use “summary” to get a single summary record.

Configuration Details

The basic configuration can be done via the app.telemetry client (GUI) interface on the edit view.

1. Check if your application log pool has a database assigned
2. Create a new log statistics object and define all required fields
3. Create a new chart - select "*Top-X Logpool Statistics*" as data source - and define all required fields

For web applications instrumented with the web-timing API you can simplify the configuration by means of using the web-timing options wizard.

Therefor you also have to define your basic log statistic object to be used for all your web applications. Afterwards you have to configure that log statistic object as default one with the following server property to be added in your infrastructure configuration file: `infra.xml` ... (change the ID values to the one corresponding to your infrastructure)

Example: default properties in `infra.xml`

```
<ServerInfo id="0" name="ServerInfo" ... >
  <Property key="defaultchannel" value="10255"/>
  <Property key="defaultdatabase" value="10259"/>
  <Property key="defaultstatistic" value="10275"/>
</ServerInfo>
```

Remember to restart your apptelemetryserver and webserver after editing the `infra.xml` file manually!

2.8 TCP Transport for Agent-Library Communications

Since Fabasoft app.telemetry 2012 Summer Release a new transport channel for app.telemetry agent/library communication is available.

Before this feature was introduced telemetry data could only be sent to the app.telemetry agent via a native library using shared-memory communication which limits application instrumentation on the supported platforms of the app.telemetry agent.

In order to extend the support for other platforms (for application instrumentation) we have introduced the TCP transport channel which can be used to transport the telemetry data from any Java platform (also with other hardware architecture).

Feature Details:

TCP transport channel is available for following app.telemetry libraries:

- Software-Telemetry C/C++ library (on supported app.telemetry Agent platforms)
- Software-Telemetry Java library (on any Java 1.5 or higher platform)

Configuration Details app.telemetry Agent

First of all you have to enable the TCP transport channel for any app.telemetry Agent in your infrastructure (default the agent does not listen for any TCP data).

Define the network port the agent should listen on for telemetry data in the app.telemetry agent configuration:

- Linux: `/etc/app.telemetry/agent.conf`: "`TelemetryPort 10002`"
- Microsoft Windows: Registry Key: `\HKLM\SOFTWARE\Fabasoft app.telemetry\Agent\TelemetryPort = 10002` (DWORD - decimal)

Restart the app.telemetry agent daemon/service.

Configuration Details for C/C++ Library

In order to tell the native C/C++ Software-Telemetry library to communicate via TCP transport (instead of shared memory) with an app.telemetry agent, start the instrumented application with the following environment variable:

- `APM_TRANSPORT=tcp://<agent-IP>:<port>`
  - for example: `APM_TRANSPORT=tcp://localhost:10002`

Configuration Details for Java Library

In order to tell the Java Software-Telemetry library to communicate via TCP transport (instead of communicating with the native library on the local system) with an app.telemetry agent, start the instrumented application with the following configuration parameters:

- either as Java system property at the startup command line of your instrumented application:
  - `-Dcom.apptelemetry.apm.transport=tcp://<agent-IP>:<port>`
- or as environment variable:
  - `APM_TRANSPORT=tcp://<agent-IP>:<port>`
- or as app.telemetry config property set via the app.telemetry Config tool:
  - `java -jar softwaretelemetry.jar`
  - to get basic help how to use this config tool call it with the param "`help`"
    - `java -jar softwaretelemetry.jar help`
  - to get more extensive help about the possible configuration parameters call it with the param "`info`"

        - `java -jar softwaretelemetry.jar info`
    - setting the transport as config property:
        - `java -jar softwaretelemetry.jar set configs/<myconfig> transport tcp://<agent-IP>:<port>`
    - and start the application with the config name with the Java property:
        - `-Dcom.apptelemetry.apm.config=<myconfig>`
    - or if using JVMTI start the application with the `javaagent` and the config name:
        - `-javaagent:softwaretelemetry.jar=config=<myconfig>`

Optionally you can define a log file for debugging purpose as Java system property:

- `-Dcom.apptelemetry.apm.logfile=<logfile-name.log>`
- `-Dcom.apptelemetry.apm.loglevel=debug|trace|default`
    - `debug` will cause many debug log messages
    - `trace` will result in a huge amount of trace messages where almost everything is logged (for developer)
    - any other *loglevel* value will result in normal logging (info, warning, error)

2.9 Dynamic Java Instrumentation (JVMTI)

With the 2011 Winter Release a Java instrumentation framework was introduced, which allows analyzing of Java applications based on instrumentation points dynamically added to the compiled java code on runtime. Using a definition stored at the Fabasoft app.telemetry Server, the application can be analyzed on a coarse or detailed level depending on the telemetry points added to the definition and by the level of detail you select to record.

2.9.1 Technical Background

Java provides an interface named "*Java Virtual Machine Tool Interface*" (*JVMTI*), which provides certain methods and callback entry points to access data and to intercept processing of applications running on Java virtual machines. Fabasoft app.telemetry provides a native code Java agent library, which may be loaded at startup of the Java runtime.

Instrumentation

The Fabasoft app.telemetry JVMTI Library will intercept the load mechanism of Java classes to insert code fragments into those methods defined in the "*Dynamic Instrumentation*" section of the log pool definition. When being executed these code fragments will call the Fabasoft app.telemetry library to record telemetry information in combination with the specified parameters and return values.

This instrumentation approach provides a moderate effort during startup time due to instrumentation of the methods selected to be instrumented, and a small overhead during execution time calling the telemetry library, which depends highly on the count of telemetry points recorded.

Sampling Mode

In addition to the instrumentation part, the Fabasoft app.telemetry JVMTI library provides a "`sampling`" mode, which is designed to provide a first chance overview over what takes long during application processing. This sampling mode analyzes the call stacks of all Java threads on a regular basis (default 10 milliseconds, configurable by command line parameters) to determine method executions that take long. Matching the call stack from the root to the previous interval, the sampling mode analyzer assumes that common entries will suggest a long runtime in the particular function. Although sampling may never be accurate, it provides a starting point for instrumentation and also a call hierarchy of method invocations suggested to be instrumented.

Application Registration

Fabasoft app.telemetry distinguishes application instances by application registration parameters provided by the application. These application registrations parameters will be used in ways to identify matching log pools or service objects and to clearly display the information source for particular instrumentation points during analysis. In conjunction with dynamic Java instrumentation the first usage of registration parameters is to determine the log pool where to take the dynamic instrumentation points from.

Fabasoft app.telemetry defines 4 Parameters to register an application with:

- Application name
- Application ID
- Application tier name
- Application tier ID

The "*application name*" and "*application ID*" should identify the system providing a specific service, whereas the "*application tier name*" allows distinguishing between multiple types of subservices and the "*application tier ID*" allows identifying individual service instances in an environment, where multiple instances of similar services are configured. Log pools are selected by a string match of any combination of these 4 parameters.

Analog to the mechanism of selecting the right log pool for to provide the dynamic instrumentation definition, the requests recorded by the instrumented application will show up in the request list of the particular log pool. See the configuration section on details how to set the application registration parameters.

Request Context

Services are commonly used in form of requests, either from a user or from other services. These requests may be as big as business transactions or as small as an RPC call to as base service. Fabasoft app.telemetry uses the term "*request*" as the unit of work performed on a single service. A request in the dynamic Java instrumentation is represented by the execution of one or more specifically marked functions. So the request starts at the beginning of the function and ends when the function returns. All instrumentation points executed on this particular thread will be recorded and displayed in the context of this request.

Instrumentation Point Definition

An instrumentation point definition represents all information, which the *telemetryjvmti* library needs to instrument a single method.

Each instrumentation point has a name, which should be recognizable by the user analyzing the recorded requests.

The "*Package/Class*" and "*Method/Parameter*" properties specify which method the library should apply the instrumentation to. Instead of the class name you may provide an interface name so the definition will be used for all classes implementing this interface.

Use the "*Instrumentation Module*" parameter to assign the instrumentation point to a module. Grouping instrumentation points into modules has the advantage, that you may easily identify the software layer (e.g. Database, Authentication, …) which is causing problems.

On runtime you select which level of information detail you want to record. A more detailed recording level will provide more detailed information on the cost of more data traffic and more impact on the performance of the application. With the "*Log-Level*" property you select in which detail level you choose to record this instrumentation point with. Every defined instrumentation point will always be inserted into the code, no matter what the current recording level is, because you may change the recording level in runtime or you may start a telemetry session with a more detailed recording level. But the first check the telemetry library does when a telemetry point has been triggered is to validate the log-level against the current recording level of the request, so the performance impact is minimized.

There are a number of flags, which can be set on each recorded instrumentation point:

- `Wait`: The Wait flag indicates (in combination with the "*Duration*" measurement) that the time in this method is spent waiting for another resource. The time is therefore displayed with a small bar in the request overview.
- `Process Info`: For to trace processor and memory usage, an additional "*Session Info*" entry will be recorded whenever this instrumentation point is triggered.
- `Warning/Error`: To mark an instrumentation point as being a warning or error message, set the according flag and the event will show up in the request analysis in a separate tab. Error events with log-level "*Log*" will increase the "*Errors*" count in the request list.

During the instrumentation process of a method, code fragments may be inserted at the beginning and before the return codes of the function. The measurement flags declare which fragments to be used in which place. When no measurement is selected, only one instrumentation point will be inserted at the beginning of the method. If only the "*Return Value*" parameter is selected, an instrumentation point will be inserted at the end of the method with the return value logged as a parameter. To measure the duration two code fragments need to be inserted, one at the beginning and on at the end of the method. Selecting the "*Duration*" measurement will provide this.

As described above (chapter “*Request Context*”) one or more methods need to be flagged as a “*Request Context*” to tell the Software-Telemetry about the scope of a request. Additional code fragments are inserted to trigger a “*CreateContext*” at the beginning of the method and a “*ReleaseContext*” at the end when the “*Start & Stop a request here*” flag is selected.

You may select any input parameter of the function as “*Method Parameters*” so the value of is will be added as a parameter to the instrumentation point at the beginning of the method.

2.9.2 Configuration of Dynamic Instrumentation / JVMTI

Load Fabasoft app.telemetry Java Agent Library

In order to activate Fabasoft app.telemetry for a Java application, the Java runtime has to be started with an additional command line parameter:

Syntax for Java Agent Startup

```
-agentlib:telemetryjvmti=<mode>
```

where the `<mode>` parameter may have one of the following values:

- `dynamicInstrument`
- `sampling`

For many Java applications this can be set via the environment variable `JAVA_OPTS` which is regarded by many startup scripts (sometimes it is better to extend that environment variable instead of overwriting it):

Syntax for Java Agent Setup via `JAVA_OPTS`

```
<set/export> JAVA_OPTS='-agentlib:telemetryjvmti=<mode>'
```

Additionally you can turn on logging for the app.telemetry JVMTI Java Library by setting the Java startup paramter:
`-Dcom.apptelemetry.apm.logfile=<your-logfile-pathname>`

Using “`dynamicInstrument`” will instruct the library to apply the dynamic instrumentation definition to the application so the application will be instrumented.

With “`sampling`” the library will – in addition to dynamic instrumentation – provide information gathered by analysis of thread call stacks on timed basis. Additional parameters may be provided in a comma separated way:

- `appname=<application name>`
- `appid=<application id>`
- `apptiername=<application tier name>`
- `apptierid=<application tier id>`
- `interval=<sampling interval in milliseconds>`

The first 4 options provide the application registration parameters and the interval option modifies the sampling interval if the library operates in sampling mode.

Usage Example for Java Agent Startup

```
java -jar myApp.jar
-agentlib:telemetryjvmti=dynamicInstrument,appname=”SampleApp”,
appid=”1.5”,apptiername=”backend”
```

Application Registration

Each application registers itself using the 4 registration parameters

- `Application name`
- `Application ID`
- `Application tier name`
- `Application tier ID`

You may provide theses parameters either as command line parameters in the `-agentlib` parameter or as environment variables

- `APM_APPNAME=<Application name>`
- `APM_APPID=<Application ID>`
- `APM_APPTIERNAME=<Application tier name>`
- `APM_APPTIERID=<Application tier ID>`

The environment parameters are considered only if none of the registration parameters is specified on the command line. The Application name is the only parameter that must be defined and it has a default value of "Fabasoft app.telemetry JVM TI Integration". All other parameters are optional and can be defined according to your services structure. For every registered application a service object will be created automatically in the Fabasoft app.telemetry infrastructure, which is uniquely identified by the 4 registration parameters and the infrastructure id of the Fabasoft app.telemetry agent. As soon as an application has successfully registered, the registration parameters are selectable as filter values in the log pool definition dialog.

Sampling Mode

When sampling mode is active by specifying the "`sampling`" mode parameter, the Fabasoft app.telemetry telemetryjvmti library will analyze the state of the application every sampling interval (default 10ms, configurable by the "`interval`" parameter in the `-agentlib` command line parameter).

Once every minute the cumulative statistic will be transferred to the Fabasoft app.telemetry server. This statistic is available through the "*Edit Log Definition*" dialog on the "*Dynamic Instrumentation*" tab.

Pressing the "*Sampling…*" button will show a list of all methods that have been determined to take time. Select any meaningful entry and press "*Add*" to add an instrumentation point for that method.

There is another way to access sampling mode statistics. Select an existing instrumentation point from "*Instrumentation Points for Dynamic Instrumentation*" and press "*Browse*" will show a dialog that presents the context of the instrumentation point. The list "*Current Method Invoked by the Following Methods (Callers)*" contains all methods which have seen to be callers of the selected function during the sampling time. And underneath the "*Selected Method*" there is the "*Current Method Calls Following Methods (Callees)*" list, which contains the methods, which have been called by the selected method. Both lists may not contain the complete list of callers/callees since sampling is a statistic and not an exact measuring method. Navigate through the call stacks by double-clicking methods in either the callers or callees list and add instrumentation points using the "*Add*" button.

Edit the properties of the new instrumentation points by selecting the respective entry and pressing the "*Edit*" button.

2.9.3 Predefined Instrumentation Definitions

Tomcat

The Tomcat instrumentation is based on Tomcat version 6 and covers the following interfaces:

- `org.apace.coyote`

    - Adapter
    - Request
    - Response
- `org.apace.catelina`
    - Valve
- `javax.servlet`
    - GenericServlet
    - ServletRequest
    - ServletResponse
    - Filter

The method `Adapter.service` is the main request context. During request processing the request is handled by a number of configured *Valves*, each calling the “`invoke`” method of the next Valve before leaving.

Liferay

The Liferay instrumentation definition has been developed for a Liferay version 5.2 on top of a Tomcat and defines additional instrumentation points covering the processing of the Liferay servlets.

The following classes will be instrumented:

- `com.liferay.portal.util`
    - PortalUtil
- `com.liferay.portal.events`
    - EventsProcessor
- `com.liferay.portal.kernel.events`
    - Action
- `com.liferay.portal.struts`
    - PortletAction
- `com.liferay.portal.action`
    - LayoutAction
- `com.liferay.portal.servlet`
    - MainServlet
- `org.apache.struts.action`
    - ActionServlet
    - RequestProcessor

JDBC

The JDBC instrumentation definition includes methods of the interfaces, which any JDBC driver implements. The instrumentation points cover the following interfaces:

- `java.sql`
    - DriverManager
    - Driver
    - Statement
    - PreparedStatement
    - ResultSet
    - RowSet

- SQLException

The methods used to establish a connection to the driver and to execute SQL statements are instrumented at “*Normal*” level whereas the access to the ResultSet is instrumented at “*Detail*” level. Access to the data fields is not instrumented at all because of performance and security reasons.

2.10 File System Counter Checks for Fabasoft Folio Internal Data

Fabasoft app.telemetry counter checks are a very powerful possibility to monitor arbitrary values of different (foreign) systems. One of those (foreign) systems (from the app.telemetry point of view) is Fabasoft Folio and the internal data structures.

With shell scripts you have still the possibility to obtain some internal data from Fabasoft Folio and with app.telemetry counter checks you can monitor those values obtained by some scripts writing the results into text files.

Some of the internal data of Fabasoft Folio can be obtained by executing the utility program `fsceval` on a Fabasoft Folio backend server (running as Fabasoft Folio service user).

1. Write your Fabasoft Folio Kernel expression printing out the desired value at the end of the script into a text file `<getvaluescript.coo>`.
2. Test your expression script with the tool `fsceval` running on a Fabasoft Folio backend server on behalf of a Folio service user (fscsrv): `fsceval -nologo -file <getvaluescript.coo>`.
   You may have to specify the Fabasoft Folio host (`HOST=localhost`) and the port of the backend service (`PORT=18070`).
   The script execution should print the value defined in the expression file.
3. “Grep” for the resulting value in the output and store the plain value into a text-file stored in the app.telemetry status-file folder readable for the app.telemetry agent service user:
   `fsceval -nologo -file <getvaluescript.coo> | grep value | awk '{print substr($3, 2, length($3) - 2)}' > /var/opt/app.telemetry/status/<fsc-value.txt>`.
4. Create a new Fabasoft app.telemetry service check of type “Counter check using file system” and select the status file defined in your scripts. You should also harmonize the update interval of your file counter (`cron` job) with the check interval.

Note: Running these scripts as cron job may require some special environment handling:

- Ensure that the tool `fsceval` is in the `PATH` of the cron job script or call it with full path (`/opt/fabasoft/bin/fsceval`).
- Ensure that the LD_LIBRARY_PATH is set correctly
  `LD_LIBRARY_PATH=/opt/app.telemetry/lib64:/opt/fabasoft/share/eval:/opt/fabasoft/share/eval/INSTALLDIR/Domain_1_1:/opt/fabasoft/share/eval/INSTALLDIR/Domain_1_1001`
- The expression file <`getvaluescript.coo`> has also to be passed with absolute path.

This may sound a little bit complex but the following two examples will help you understand and use this powerful feature:

2.10.1 Check Remaining Object Addresses in a Fabasoft Folio COO-Store

To monitor the count of free object addresses in a Fabasoft Folio COO-Store you may use the following expression and scripts:

1. Write the expression and save it to a file (objinfo.coo).

`objinfo.coo`: Expression for free addresses in COO-Store (example: COO-Store 2)

```
@svcs = coort.SearchLocalObjects3(cootx, "COOService");
@objremaining = 0;

for (@i = 0; @i < count(@svcs); @i++) {

  @objIds = @svcs[@i].coosrvinfo.cooinfmaxobjids;
  for (@j = 0; @j < count(@objIds); @j++) {

    @objId = @objIds[@j];

    if (@objId.cooinfmaxcoost && @objId.cooinfmaxcoost.coostid == 2) {
      @objremaining = @objId.cooinfavailobjids;
      break;
    }
  }
}

@objremaining;
```

2. Write a script to get and update the value, save it as shell script and test it (running as Folio service user on a backend server).

Test expression using `fsceval`

```
su - fscsrv
HOST=localhost
PORT=18070

fsceval -nologo -file objinfo.coo
```

3. Extend your script by means of “grepping” for the desired value in the output and storing the result into an app.telemetry status file and configure a `cron`-job to call this update script periodically.

Update shell script (`update-objinfo.sh`)

```
#!/bin/bash

HOST=localhost
PORT=18070

LD_LIBRARY_PATH=/opt/app.telemetry/lib64:/opt/fabasoft/share/eval:/opt/fabasoft/share/eval/INSTALLDIR/Domain_1_1:/opt/fabasoft/share/eval/INSTALLDIR/Domain_1_1001

export HOST PORT LD_LIBRARY_PATH

/opt/fabasoft/bin/fsceval -nologo -file /home/fscsrv/objinfo.coo
  | grep value | awk '{print substr($3, 2, length($3) - 2)}'
  > /var/opt/app.telemetry/status/coost2.txt
```

This will result in a status file containing the current count of free object addresses of that particular COO-Store object.

4. Create a new app.telemetry counter check to monitor the value from the status file and define the update interval and the warning/critical limits (for example: set a warning level for below 1000000 and a critical limit for below 100000) to be notified when the COO-Store is low on free object addresses.

2.10.2 Check Remaining Days until Fabasoft Folio License Expires

In order to get notified before your Fabasoft Folio license expires just follow this example.

1. Write an expression like the following to get the days until your license will expire and save it to a file (`fsclicense.coo`).

`fsclicense.coo`: Expression to check days until license expires

```
@lics = coort.GetCurrentDomain().COOSWCLM@1.1:domainlicenses;
@expiryday = 1000;

for (@i = 0; @i < count(@lics); @i++) {
  @lic = @lics[@i];
  if (@lic.COOSWCLM@1.1:keyexpirydate) {

    @exp = (@lic.COOSWCLM@1.1:keyexpirydate -
coort.GetCurrentDateTime(coouser)) / 3600 / 24;

    if (@exp < @expiryday) {
      @expiryday = @exp;
    }
  }
}

@expiryday;
```

Warning: In some situations you may not rely on the accuracy of the `COOSWCLM@1.1:domainlicenses` property of your current domain.

2. Write a script to get and update the value, save it as shell script and test it (running as Folio service user on a backend server).

Test expression using `fsceval`

```
su - fscsrv
HOST=localhost
PORT=18070

fsceval -nologo -file fsclicense.coo
```

3. Extend your script by means of “grepping” for the desired value in the output and storing the result into an app.telemetry status file and configure a `cron`-job to call this update script periodically.

Update shell script (`update-license-expiration.sh`)

```
#!/bin/bash

HOST=localhost
PORT=18070

LD_LIBRARY_PATH=/opt/app.telemetry/lib64:/opt/fabasoft/share/eval:/opt/fabasoft/share/eval/INSTALLDIR/Domain_1_1:/opt/fabasoft/share/eval/INSTALLDIR/Domain_1_1001

export HOST PORT LD_LIBRARY_PATH

/opt/fabasoft/bin/fsceval -nologo -file /home/fscsrv/fsclicense.coo
  | grep value | awk '{print substr($3, 2, length($3) - 2)}'
  > /var/opt/app.telemetry/status/fsc-lic-expiry.txt
```

4. Create a new app.telemetry counter check to monitor the value from the status file and define the update interval and the warning/critical limits (e.g. warning below 30 days and critical below 5 days).

2.11 Data Retention Strategies for app.telemetry

Fabasoft app.telemetry provides different strategies for managing data retention.

Larger amount of data is stored by the app.telemetry Server continuously by the following services:

- **Software-Telemetry request data** … is stored in a database table for each log pool (1 record/request). Cleanup rules can be configured in the log pool configuration dialog.
- **Software-Telemetry request detail data** … is stored as rawdata on the file system in "daily"-directories (…`/server/telemetry/<yyyy-mm-dd>`). This kind of data is consuming much more disk space than the request records on the database.
  Note: Do not delete the directory of the current day (without a full restart of all server processes). Instead you should use the automatic cleanup rules from the "Server Properties" dialog.
- **Precalculated log pool statistics** … are stored beside the request database tables for each log pool.
- **Software-Telemetry sessions** … are stored on the file system containing request data and request detail data (rawdata). They are created on demand by a user or when a feedback is sent.
  - **Started/Stopped Server sessions** are read from rawdata so they are available until rawdata of the respective time is being deleted. To keep the data permanently make sure to download the session as a zip file.
  - **Feedbacks** will be automatically extracted from rawdata into zip files so that the feedback are available even after the rawdata have been cleaned up. Session zip files are deleted when the corresponding feedback is permanently deleted from the inbox.
  - **Uploaded sessions** are stored as complete ZIP-archives containing all required data on the worker and are already independent from the rawdata directories.
- **Counter check data** can also be stored in a database table (1 record/check).
- **Recorded status change records** can be configured to be automatically deleted after x days via the global "Server Properties" (since version 14.2 configurable – before it was defined to be 1 month).
- **SLA-relevant availability data** information will never be deleted from the defined database.
- **Activity statistic data** … since version 2015 all telemetry requests are analyzed and statistical data (module and event activity statistics) is stored on the file system (…`/server/telemetry/activitystats/`). Cleanup strategies can be configured in the global "Server Properties" dialog.

In order to handle the increasing amount of data and prevent the disk from running out of free space you can configure automatic retention time periods within the app.telemetry client. For more details read the sub chapter "Cleanup Strategies".

Note: Before starting to delete any data you should export Software-Telemetry server sessions and feedbacks by a special automatic app.telemetry server task (configuration setting) in order to access the request details of such sessions later on.

2.11.1 Automatic Export of Feedback Sessions

In most situations feedbacks should be available for a much longer time period than standard request detail data which will be held for post-problem analysis for some time. Therefore you can split off the detail data required for the feedbacks from the normal rawdata directories.

This feature will be automatically enabled after updating to version 2014 Spring Release or later unless it is explicitly disabled in the server configuration file. If you want to change this setting or the session export path stop the app.telemetry server daemon then open the configuration file of the app.telemetry server (`/etc/app.telemetry/server.conf`) and setup and activate the configuration parameter “`SoftwareTelemetrySessionPath`”:

```
# The SoftwareTelemetrySessionPath property defines the target location
# for extracting reported telemetry sessions from the raw data files. (optional)
# The default path is: /var/opt/app.telemetry/server/sessions
# To disable automatic session extraction uncomment the line below (set to empty)
SoftwareTelemetrySessionPath /var/opt/app.telemetry/server/session
```

After this configuration is activated you can start the app.telemetry Server daemon again.

After a while the server will start processing all available existing telemetry sessions and export them to the configured directory. This process may take some time depending on your infrastructure and on the number and size of reported sessions/feedbacks. You can watch the progress of that action by the increasing directory content size on the file system. This process is an ongoing process that will also export new incoming feedbacks a couple of minutes after they have been fully completed.

2.11.2 Cleanup Strategies

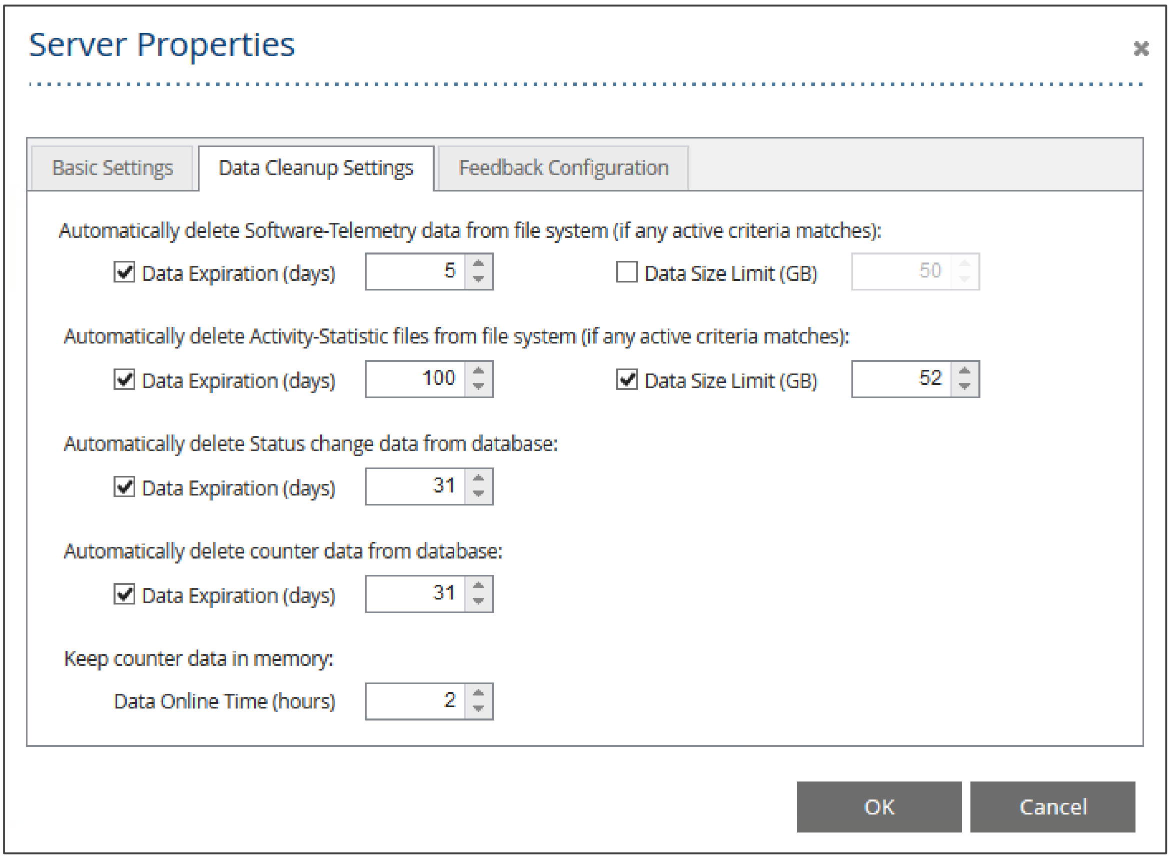
Within the Fabasoft app.telemetry client you can define different automatic data deletion rules in order to keep essential data for a defined time period but prevent filling up the disk with old data not required any more.

The most cleanup settings can be configured within the global “Server Properties” dialog (in edit view at the top of the infrastructure tree).

You can either **limit** the retention of the data **by time** as number of days. If you activate this cleanup rule, any data that is older than the defined time range will be deleted a non-deterministic time span later (please be patient after applying the changes and give the server some time to process the cleanup).

The other possibility to limit the amount of data (only available for file system based data) is to set a **data size limit** in gigabytes (GB). But be careful this limit is only an estimated size limit and can vary a little bit.

You can define a single type of limit for every kind of data or even both limits, which mean if your data match one of the two criteria the cleanup will be triggered to reduce the amount of data to fit the all criteria again.


Server Properties

Basic Settings | Data Cleanup Settings | Feedback Configuration

Automatically delete Software-Telemetry data from file system (if any active criteria matches):

☑ Data Expiration (days) 5 ☐ Data Size Limit (GB) 50

Automatically delete Activity-Statistic files from file system (if any active criteria matches):

☑ Data Expiration (days) 100 ☑ Data Size Limit (GB) 52

Automatically delete Status change data from database:

☑ Data Expiration (days) 31

Automatically delete counter data from database:

☑ Data Expiration (days) 31

Keep counter data in memory:

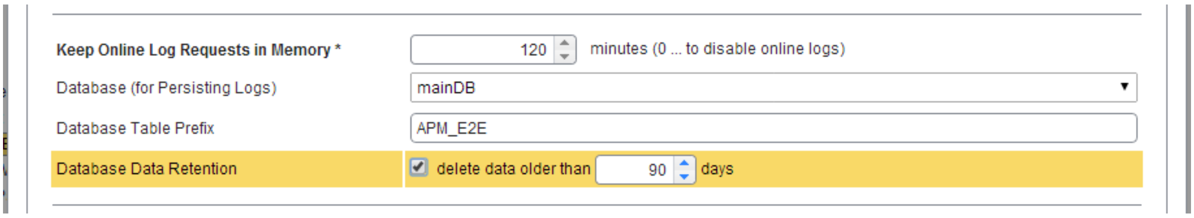
Data Online Time (hours) 2

The data retention for the **Software-Telemetry request detail data** (rawdata on filesystem) is used to reduce the big-sized data of old requests. It is your choice how long you want to keep request detail data for a detailed problem cause analyze (request overview, request details, request statistics, request grades) and depends on the amount of available disk space.

Warning: you should have exported the reported telemetry sessions/feedbacks as described in the last chapter otherwise those session details will not be accessible!

For long-term analysis of your applications you could still use the **activity statics** if you keep those data for a longer period than the request detail data.

Additionally you can configure a **database retention** time range for all **log pools** as parameter on every log pool configuration dialog:


Keep Online Log Requests in Memory * 120 minutes (0 ... to disable online logs)

Database (for Persisting Logs) mainDB

Database Table Prefix APM_E2E

Database Data Retention ☑ delete data older than 90 days

If you activate the database data retention for a given time range (in days) the following data will be automatically deleted from database tables belonging to that log pool:

- Software-Telemetry request data
- Precalculated statistics

The **Status change data** is required to show a table containing the history of all status changes of every service/counter-check. This history data is available for the time range defined in the cleanup settings of the “Server Properties”.

Service checks with a defined SLA-definition are persisted independently and unlimited in a database defined within the SLA-definition.

Counter checks with a defined database for persisting the counter results are also stored on the defined database. On the “Data Cleanup Settings” page of the “Server Properties” you may specify a **“Data Expiration (days)”** value to cleanup all counter values that are older than the given number of days (this option is available with Fabasoft app.telemetry 2015 Rollup 2).

2.12 Analyze Requests using Rules

Analyzing performance issues in requests – especially in distributed applications like Fabasoft Folio – is a time consuming task requiring a lot of application specific knowledge. To simplify this process, a set of rules is being created to identify common issues.

There are common rules applying to any Fabasoft app.telemetry instrumented application and rules specifically written for dedicated Products like Fabasoft Folio. In those rules common problems are identified by the telemetry data included in the request. Designing those rules is an evolving process where analyzing steps originally processed manually are being formalized and automated.

In order to use these rules for analyzing telemetry requests, open a request on the telemetry view and select the new “Grades” tab in the bottom analyzer area.

2.12.1 Analyze Entirely Completed Requests

This rule applies to any request generated by Fabasoft app.telemetry instrumented applications and simply checks, if all processing threads, which occur in this requests were correctly terminated.

There are several reasons why a request could be detected not to be finished:

- A request may be opened while still being processed. You may detect what the request is currently processing by looking at the end of the unfinished threads.
- A request may be too large to be processed. Try to reduce the amount of telemetry points per request by lowering the instrumentation level of the log pool or session.
- A process may have stopped working. The problem may be near to the last telemetry point of the unfinished thread.
- The instrumented application may not handle a failure (e.g. exception) correctly and so no ReleaseContext has been recorded.

2.12.2 Reduce RPCs to Fabasoft COO Service

The Fabasoft Web service communicates with the Fabasoft Backend Services using Remote Procedure Calls (RPCs). Each call requires at least one network roundtrip and the allocation of a backend service thread. Issuing too many calls will result in a delay mainly caused by the network latency. Replacing many small RPCs by fewer larger ones will save roundtrip time and management overhead on client and server side.

The grade of the rule will reflect the potential benefit of an improvement based on the fraction of time consumed by RPC requests in proportion to the total request time.

Thus the main info provided is the count and the duration of all RPCs executed. In addition the duration is split in the communication time and the execution time base on the time difference between the requests on the Fabasoft Kernel side and the execution on the Fabasoft COO Service side.

Especially when a high communication time is indicated, the COO Service RPCs are worth being further analyzed. Assuming that the Web Server and Backend Server are located in reliable and fast network infrastructure high communication time results most likely from a high number of RPCs. Each RPC takes at least half a millisecond overhead for the network to transfer the request, the COO Service to schedule the request to an available worker thread and to transfer the result back to the Web Service. So a high number of RPC requests directly lead to bad performance without having a bottleneck in any single application tier.

In the details section an RPC statistic based on RPC type is being provided indicating how the different RPC types contribute to the total RPC time and count.

The most common problem in this area is the so called **Typewriter**, which can be determined by a high “Request Count” in the “`COOSTAttrSel`” RPC, which is the RPC requesting object information from the COO Service. The typical source for that situation is a loop iterating over a list of objects without previously loading the required attributes of all these objects in a single call. So any access to an object will require the Kernel to load the object one by one. While this will produce the correct result, it will lead to multiple RPC requests and therefore to bad performance. To optimize the Typewriter scenario requires a call to `coort.LoadAllAttributes/LoadSpecificAttributes` providing the list of all objects being iterated.

The list of the "Top 10 events" issuing RPCs" may help you identifying the method containing the loop. This can be identified by the last Event before the count drops to a low value. Clicking on the Event will lead you to the detail view showing the instrumentation points recorded while processing this method.

2.12.3 Improve Object Attribute Reads

This rule analysis how many object attributes are read and how much time is consumed to do this. The instrumentation points required for this analysis are only recorded in *Debug* mode. In less detailed recording levels this rule will show up as "N/A".

As the `GetAttribute` variants are the usual way to access Fabasoft Folio objects, it is not an error to do so, but if accessing information takes a reasonable fraction of the processing time, it is still a good starting point for further investigations. So use the list of "Top events accessing object attributes" to identify the method, in the context of which many attributes are being accessed.

You may optimize data access either by caching or by calling `GetAttribute` once instead of iterating an attribute using `GetAttributeValue`. Also a call to `HasAttributeValue` with a subsequent call to `GetAttributeValue` can often be replaced by a single `GetAttributeValue` saving at least the overhead of an access check.

When looking at the "Attribute Access Statistics" you can determine the duration for a specific type of data access. Most interesting here is the fraction between the "Duration" and the "Self Time" where a high "Self Time" indicates, that the duration mainly results from the count of data accesses, whereas a low "Self Time" compared to the duration indicates cache misses either on the object itself or on objects required for the access check. Cache misses will result in RPCs fetching object data from the COO Service.

Click on the Call name to go to the "Selected Statistics Events" tab, sort by duration and try to solve the performance problem, when attribute accesses lead to COO Service requests.

2.12.4 Optimize HTTP Requests

The Fabasoft Client communicates with the Fabasoft Web services using http requests. Each call requires at least one network roundtrip and the allocation of a web service thread. Issuing too many calls will result in a delay mainly caused by the network latency.

The "Optimize HTTP Request" rule helps you determining why a request on the Fabasoft Web Browser Client is slow.

- The number of HTTP requests should be low. Each http request requires a network roundtrip which can be very expensive when the device is connected via mobile data connections.
- In addition browsers limit the number of concurrent connections to a server so otherwise parallel requests will be serialized limiting the number of parallel requests.
- The "Client Request Time" indicates the time required to send the request until the response has been fully received by the browser. The "Client Processing Time" represents the time needed to process the result and to make the required changes to the page being displayed. A large "Client Processing Time" indicates a high rate of changes to the DOM model of the page, which is most likely the result of large lists being rendered. Older browsers (e.g. Internet Explorer 7 or 8) have significantly slower engines which results in higher "Client Processing Time".

    - The “Network Time” is calculated from the difference between the “Client Request Time” and the “Server Processing Time” and depends mainly on the connection speed, connection latency and the amount of data being transferred. The size is being indicated here by the “Request Size” and “Response Size”.

Based on that analysis you can focus on the part of the request, which has most influence on the request time.

2.12.5 Optimize Database Statements

Fabasoft COO-Services read and write object data from/to a relational database. Reading data is required in case of queries and when objects are currently not in the COO-Service cache. Writing data occurs every time objects are being created, changed or deleted. Object lock information is also persisted on the database.

The way how to optimize queries depends on the type of database statement:

Reading Objects (`COOSTAttrSel`):

- Reading objects only occurs when objects are not in the COO-Service cache. So try to keep all active objects in the COO Service cache.
- Minimize the working set of objects being accessed by the user.
- Preload objects together before iterating a list of objects to reduce the count of database queries.

Queries (`COOSTQuery`): Processing queries is executed in several phases:

- Query for the object ids matching the properties given.
- Load all properties of all matching objects which are not in cache.
- With this information the kernel applies the security model and filters out those objects that may not be found by the user.

2.13 How to Configure Certificate Authentication on Linux

The default authentication method for Fabasoft app.telemetry web browser client users is "Basic Authentication". Since 2011 Winter Release you may use https with client certificates as an alternative login method. The following guide explains how to configure “Certificate Authentication” for Fabasoft app.telemetry using an Apache webserver on a Linux system.

Prerequisites:

- Server Certificate (pem and key file)
- CA certificate(s) of the client certificates

Configuration:

- Install mod_ssl (`mod_ssl-<version>.<os>.<arch>.rpm`)
- Copy the server certificate (e.g. to `/etc/pki/tls/certs/`)
- Copy the server certificate key file (e.g. to `/etc/pki/tls/private/`)
- Use `chmod/chown` to provide access to the certificate files to the apache user
- Configure Apache SSL and access restrictions (e.g. `/etc/httpd/conf.d/ssl.conf`)
  - Set the "`ServerName`" to the name used in the server certificate
  - Set the "`SSLCertificateFile`" to /etc/pki/tls/certs/<your certificate>.pem
  - Set the "`SSLCertificateKeyFile`" to /etc/pki/tls/private/<your certificate>.key
  - Set the "`SSLVerifyClient`" to `require`
  - Set the "`SSLVerifyDepth`" to a value of your chioce (e.g. 10)
  - Add the `SSLOptions +StdEnvVars`
- Import the client CA certificate
  - `openssl x509 -in cacert.cer -text >> /etc/pki/tls/certs/ca-bundle.crt`
- Add the common names (cn) of the users to the groups provided in `/etc/app.telemetry/htgroup`
- Restart the Apache web server (you have to provide the passphrase for the server certificate key file when restarting the web service).

Details / Hints:

- To extract the pem/key files from a pkcs12 certificate you may use the following commands:
  - `openssl pkcs12 -clcerts -nokeys -in vmcentos.pfx -out server.pem`
  - `openssl pkcs12 -nocerts -in vmcentos.pfx -out server.key`