DB Mobile Workflow Agent

Configuration and Installation

Version : 0.3

Stefan Marx

4/3/2017

1 Introduction 3

1.1 Function and Purpose 3

1.2 Overall Architecture 3

2 Installation 4

2.1 Machine Agent Installation 4

2.2 Extension installation 5

2.3 Configuration and Test (Extension) 7

2.4 Enable the Agent 8

3 Function and Implementation 10

3.1 Breadcrumbs 10

3.2 Processing 10

3.3 Metrics Table 11

4 Apendix A: Sample Log output explained 12

# Introduction

At the DB Mobile Project, we needed to create a custom solution to allow monitoring of Async workflows that happen inside the mobile agent (IOS and Android) without any backend common execution.

## Function and Purpose

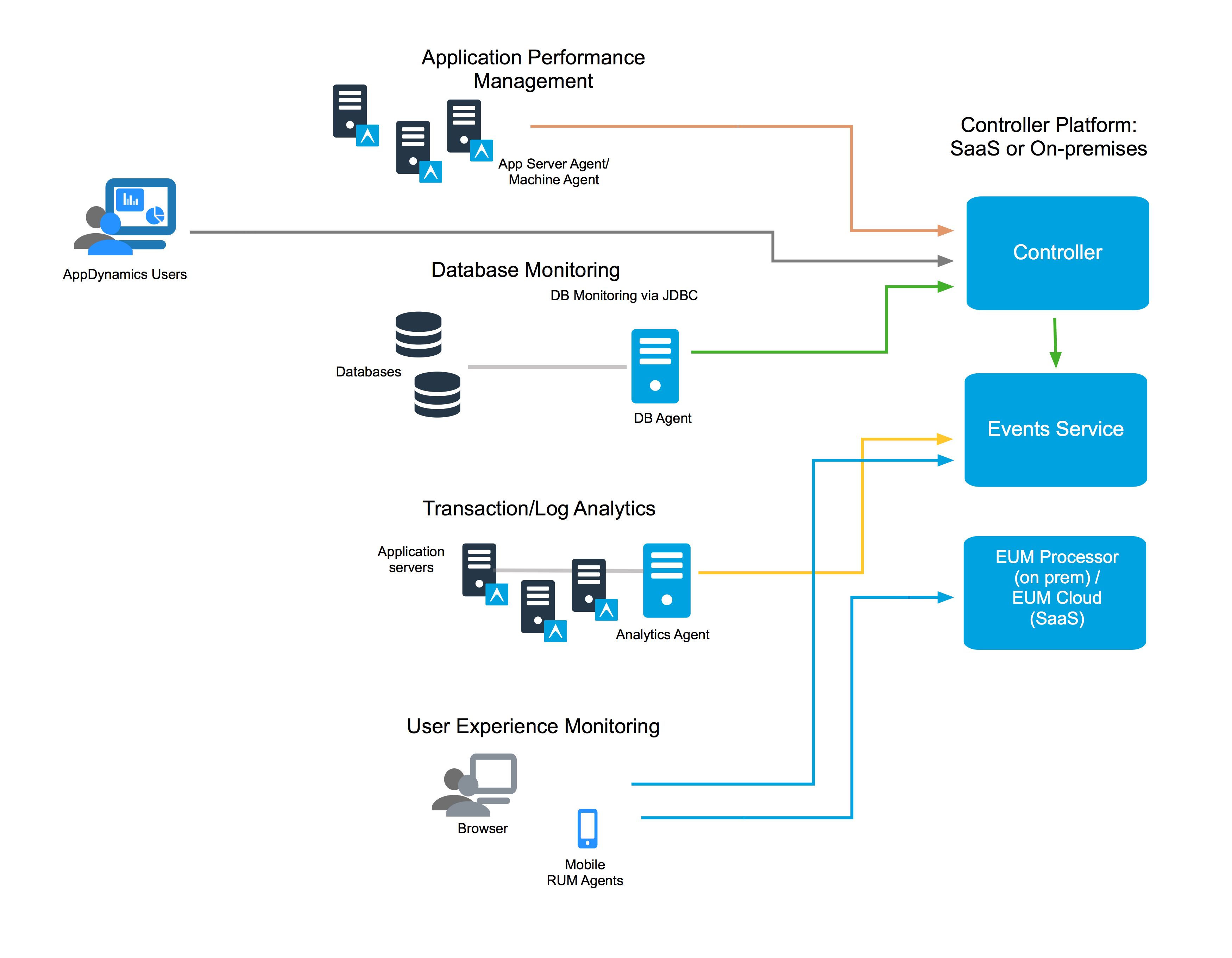
The Agent relies on Analytics data and is using the mobile session data created by the mobile agents to collect, interpret and report status and performance of async workflow executions. The workflows, possible states and error conditions can be configured. The Agent uses the machine agent extension interface to connect to the controller and the event service api to connect to the analytics data store.

<https://docs.appdynamics.com/display/PRO43/Machine+Agent+Metric+Collection>

<https://docs.appdynamics.com/display/PRO43/Extensions+and+Custom+Metrics>

<https://docs.appdynamics.com/display/PRO42/Analytics+Events+API>

## Overall Architecture



The DB Installation utilizes the AppDynamics controller on premise but uses the EUM events service and collector in the SaaS cloud. The Mobile Workflow extension is installed using a standalone machine agent which require access to the on premise controller (http/https) AND access to the SaaS cloud event service (https). Both connections are outbound and can use proxies to establish a secure connection.

# Installation

## Machine Agent Installation

The Machine agent is a pre requisite to run the Extension therefor a standalone machine agent installation is the first step in installing the mobile workflow extension. The Machine Agent needs to be installed following this guide (<https://docs.appdynamics.com/display/PRO42/Standalone+Machine+Agent)> and it need to be configured to point to the on-premise Controller.

In case Linux is used as the basic system the machine agent should either be installed with its own jre:

<https://docs.appdynamics.com/display/PRO42/Linux+Install+Using+ZIP+with+Bundled+JRE>

or it can also be installed using an installation package:

<https://docs.appdynamics.com/display/PRO42/Linux+Install+Using+the+RPM+Package>

It is suggested to define a static Application and tier name for the machine agent as it would not be able to pick up an Application name and tier from an AppServer agent running on the same node. The Application name should either be the same application that the corresponding mobile application uses for its data OR it should be a dedicated Application only handling the mobile workflow data.

<https://docs.appdynamics.com/display/PRO43/Standalone+Machine+Agent+Configuration+Property+Reference>

The Agents memory requirements are depending on the number of active sessions to trace and the number of workflows configured. The suggestion is to start with the standard requirements and watch the machine agent process. If the memory seems to become a bottleneck the –xMx parameter should be increased.

<https://docs.appdynamics.com/display/PRO42/Standalone+Machine+Agent+Requirements+and+Supported+Environments#StandaloneMachineAgentRequirementsandSupportedEnvironments-JVMMemoryRequirements>

The Machine Agent should be started and the status should be verified before continuing to the next step. After this verification the machine Agent should be stopped again!

<https://docs.appdynamics.com/display/PRO43/Verify+the+Standalone+Machine+Agent+Installation>

In case the Server Metrics (infrastructure data) is not allowed or not required, the infrastructure monitor can be turned off after verification.

(in agent-install/monitors/JavaHardwareMonitor/monitor.xml change enable to false)

<monitor>

<name>SigarHardwareMonitor</name>

<type>managed</type>

<enabled>false</enabled>

<enable-override os-type="linux">false</enable-override>

<enable-override os-type="solaris">false</enable-override>

<enable-override os-type="sunos">false</enable-override>

<enable-override os-type="windows">true</enable-override>

<description>Monitors system resources - CPU, Memory, Network I/O, and Disk I/O, Disk Space.

</description>

<monitor-configuration>

</monitor-configuration>

<monitor-run-task>

<type>java</type>

<execution-style>scheduled</execution-style>

<java-task>

<impl-class>internal.hardware.monitor</impl-class>

<factory-method></factory-method>

<invoke-method></invoke-method>

</java-task>

</monitor-run-task>

</monitor>

## Extension installation

The current version of the extensions should be downloaded and unzipped. The Machine agent supports multiple metric collections and maintain all active and installed extension in the base path of its install location.

.

├── bin

├── conf

│   └── logging

├── extensions

│   ├── AgentServer

│   ├── CrashGuard

│   └── ServerMonitoring

├── lib

├── local-scripts

├── monitors

│   ├── HardwareMonitor

│   ├── JavaHardwareMonitor

│   ├── MobileWorkflowAgent

│   └── analytics-agent

├── monitorsLibs

└── scripts

The unzipped Extension needs to be copied into this directory. The filepath should look like this after the installation.

MobileWorkflowAgent

├── MobileWorkflowAgent.jar

├── conf

│   ├── dbSample.yml

│   ├── sample.yml

│   └── sample2.yml

├── lib

│   ├── Client-0.4.1.jar

│   ├── commons-codec-1.9.jar

│   ├── commons-lang3-3.5.jar

│   ├── commons-logging-1.2.jar

│   ├── gson-2.7.jar

│   ├── httpclient-4.5.3.jar

│   ├── httpcore-4.4.6.jar

│   ├── jackson-annotations-2.8.0.jar

│   ├── jackson-core-2.8.6.jar

│   ├── jackson-databind-2.8.6.jar

│   ├── jackson-dataformat-yaml-2.8.6.jar

│   ├── slf4j-api-1.7.18.jar

│   └── snakeyaml-1.17.jar

└── monitor.xml

There are two critical config files in this directories. The monitor.xml describes how the machine agent loads and starts the extension. The \*.yml config describes how the extension should login to the event service, which data it should receive and how it should interpret the data.

## Configuration and Test (Extension)

The central config file of the extension is a yaml file that will be loaded and referenced by the monitor.xml file when the machine agent loads the extension (http://www.yaml.org/start.html). The Following is a simple example which will be used to discuss the different config Options.

---

loginConfig:

passwd: "XXXX-XXXX-XXXX"

user: "DeutscheBankXXXXX"

proxyHost: "localhost"

proxyPort: 8080

apps:

- appName: "com.db.pwcc.dbmobile.dev.INT"

appKey: "AD-AAB-AAD-FYB"

workflowInstances:

- flowName: "PaymentTransaction"

shouldLogTimeoutSessions: true

timeoutMs: 60000

startStates:

- "MopayApduDataReceived"

validStates:

- "MopayApduDataPassedToLibrary"

- "MopayApduDataReceivedFromLibrary"

- "MopayTransactionAuthorized"

endStates:

- "MopayTransactionCompletedSuccess"

errorStates:

- "MopayTransactionSuspendedPinIsRequired"

- "MopayTransactionCompletedApplicationAborted"

- flowName: "CardTokenization"

shouldLogTimeoutSessions: true

timeoutMs: 60000

startStates:

- "MopayExecuteTokenizationStart"

endStates:

- "MopayExecuteTokenizationSuccess"

errorStates:

- "MopayExecuteTokenizationFailure"

- flowName: "CardDelete"

shouldLogTimeoutSessions: true

timeoutMs: 60000

startStates:

- "MopayDeleteCardStart"

endStates:

- "MopayDeleteCardSuccess"

errorStates:

- "MopayDeleteCardError"

The first section contains the login information. This Data is global for the config file and can be retrieved using the active controller license screen. This is the login information used by the controller to connect to the event service API

The next section (apps) is a list of application definition. The Extension is able to monitor workflows within multiple mobile application as long as all using the same controller/event api login.

The Application config needs to use the same appName and appKey then the actual monitored application. This information is used to select the application Data and session information.

Each Application includes a list of workflowInstances. Any instance can be configured with a name (which becomes part of the workflow data path in the AppDynamics metrics browser) a potential timeout value (ms) and a configuration whether actually timeouts and stale sessions should be monitored. The difference is that a stale session is a workflow execution that doesn’t end before the mobile session was closed, while a timeout section is a workflow that didn’t reach a proper end or error state before the end of the timeoutMS value (time between workflow start and potential timeout state in ms)

To define the possible session flows each section also contains a list of potential startStates and a list of potential endStates or errorStates. A Workflow definition may also contain a list of valid states that can be reached during the execution of a workflow. All states are delivered using the mobile agent SDK and the breadcrumb api calls!

The Extension would track the start of each workflow and looks out for any end or error state. It will then register the total amount of Executions per state and the average time spend in the workflow. Those metrics are then send to the controller as 1-min aggregates and used to show the current state of the workflows executed in a dashboard. They could also be used to display alarms.

A Proxy definition may be added to the yaml file and it’s a global setting!

To test a configuration before using in the real extension a command line utility is available that would execute an config file the same way the agent does, but instead sending data to the controller it will log to the console. An active java jre or jdk is required to execute the cli.

/MobileWorkflowAgent -cfg build/sandbox/agent/monitors/MobileWorkflowAgent/conf/dbSample.yml -start 2017-04-01 -end 2017-04-22

TODO: receive a real live example and add to this doc

This cli is very helpful while testing and changing a new configuration or when adding a new Workflow.

## Enable the Agent

The custom monitoring extensions itself is configured and enabled using the monitor.xml file located in the extension directory.

<monitor>

<name>MobileWorkflowAgent</name>

<type>managed</type>

<description>DB Mobile Workflow Agent</description>

<monitor-configuration />

<monitor-run-task>

<execution-style>periodic</execution-style>

<execution-frequency-in-seconds>60</execution-frequency-in-seconds>

<name>MobileWorkflowAgent</name>

<display-name>MobileWorkflowAgent</display-name>

<description>DB Mobile Workflow Agent</description>

<type>java</type>

<execution-timeout-in-secs>120</execution-timeout-in-secs>

<task-arguments>

<!-- Config file to read from -->

<argument name='config' default-value='./conf/dbSample.yml' is-required='true' />

<!-- newest sessions to retrieve in seconds (default:15 minutes old) -->

<argument name='delay' default-value='900' is-required='true' />

<!-- Session window (oldesSession - newestSession in seconds) -->

<argument name='windowSize' default-value='300' is-required='true' />

<!-- How long to keep old sessions in cache, should be similar to the oldes possible Session. -->

<argument name='purgeSessions' default-value='1200' is-required='true' />

<!-- Custom Metrics prefix.

Use 'Server|Component:<id>' with the ID of the tier that this Agent bounds to

to make this a tier private metric. -->

<argument name='metricPrefix' default-value='Custom Metrics' is-required='true' />

</task-arguments>

<java-task>

<classpath>./lib/:MobileWorkflowAgent.jar</classpath>

<load-jars-in-classpath-dirs>true</load-jars-in-classpath-dirs>

<impl-class>com.appdynamics.ace.custom.agent.mobileworkflowagent.DBWorkflowAgent</impl-class>

</java-task>

</monitor-run-task>

</monitor>

With this default configuration the custom collector is executed once every 60 seconds. The Collector specific config are within the task arguments

<task-arguments>

<!-- Config file to read from -->

<argument name='config' default-value='./conf/dbSample.yml' is-required='true' />

<!-- newest sessions to retrieve in seconds (default:15 minutes old) -->

<argument name='delay' default-value='900' is-required='true' />

<!-- Session window (oldesSession - newestSession in seconds) -->

<argument name='windowSize' default-value='300' is-required='true' />

<!-- How long to keep old sessions in cache, should be similar to the oldes possible Session. -->

<argument name='purgeSessions' default-value='1200' is-required='true' />

<!-- Custom Metrics prefix.

Use 'Server|Component:<id>' with the ID of the tier that this Agent bounds to

to make this a tier private metric. -->

<argument name='metricPrefix' default-value='Custom Metrics' is-required='true' />

</task-arguments>

|  |  |  |
| --- | --- | --- |
| Name | Default | Description |
| config |  |  |
| delay | 900 | Time window end to look for. |
| windowSize | 300 | Time window to look for |
| purgeSessions |  | Max age of sessions before purged from the session cache |

The config setting is a relative or absolut path pointing to the config.yml file to be used.

The time settings describe the timerange of events that the agent will retrieve on each execution. With the default settings the agent will look for sessions that have been closed between now()-(900 seconds +300 seconds) and now-( 900 seconds). If the session timeout value in AppDynamics EUM is changed to a custom value, those config values may need to be changed. The Window size should be big enough to capture all relevant sessions BUT small enough to prevent too many sessions to be falsely retrieved while being processed in the previous run. Sessions will only be processed once because the session cache will store session ids that already has been processed and ignore those in the next execution. This is an in memory cache, frequent machine-agent restart may count some workflows twice in the first execution after restart!!!

The Session Purge value is less critical, it should be at least as big as the windowSize and delay values combined. Bigger numbers have a **small** impact on the memory footprint of the custom collector extension.

It’s worth to mention the Custom Collector will only operate on closed sessions.

# Function and Implementation

The workflow progress monitor uses eum session information to scan and follow workflow executions. Workflows are heavily async so the mobile applications use custom data points (breadcrumb) to log the steps within the workflow execution

<https://docs.appdynamics.com/display/PRO42/Use+the+APIs+of+the+iOS+SDK+to+Customize+Your+Instrumentation>

<https://docs.appdynamics.com/display/PRO42/Use+the+APIs+of+the+iOS+SDK+to+Customize+Your+Instrumentation#UsetheAPIsoftheiOSSDKtoCustomizeYourInstrumentation-iosbread>

## Breadcrumbs

Breadcrumbs allow you to situate a crash in the context of your user's experience. Set a breadcrumb when something interesting happens. If your application crashes at some point in the future, the breadcrumb will be displayed along with the crash report.

There are two ways of leaving breadcrumbs:

* Crash Reports Only
* Modal

Using this method means that breadcrumbs are reported in crash reports only.

+ (void)leaveBreadcrumb:(NSString \*)breadcrumb

Using this method lets you fine tune where the breadcrumbs are reported, either only in crash reports or in crash reports and sessions.

+ (void)leaveBreadcrumb:(NSString \*)breadcrumb mode:(ADEumBreadcrumbVisibility)mode

Where mode is either:

* ADEumBreadcrumbVisibilityCrashesOnly
* ADEumBreadcrumbVisibilityCrashesAndSessions

*If the breadcrumb is over 2048 characters, it is truncated. If it is empty or nil, no breadcrumb is recorded. Each crash report displays the most recent 99 breadcrumbs.*

## Processing

The Agent retrieves closed session within the configurated time window and process those session data (breadcrumbs only)

Multiple workflow definitions are processed in parallel. Each session start state will create a workflow instance (if no instance of this workflow is already started), or log a warning (if there is already an instance started of this same workflow. Multiple workflow configurations may share the same start state, multiple instances will be created.

Once started, a workflow instance tracks its execution using the defined workflow steps. In case it reaches an end or error state the end or error count metrics are updated, and the average execution time metric is updated for this configuration.

In case a workflow reaches the timeout age (TIMEOUT) or it is not in a known end or error state at the end of the eum session (STALE) the collector will update the state counter for stale or timeout executions. It will also update another metric that reports the last known state of this execution so that counters can be used to define the most common last state BEFORE the timeout or stale of a particular workflow.

It will also produce a log output that will contain diagnostic information such as the unique session ID and timing. This information can be used to open the session details in AppDynamics and have the full picture on this particular session with all connected information and potentially all snapshots collected during this session.

## Metrics Table

All Metrics follow a common prefix:

customMetric|<**appName**>|<**FlowName**>

|  |  |
| --- | --- |
| Metric Path | Description |
| normalExecutions | # of **normal** Executions per minute |
| errorExecutions | # of **error** Executions per minute |
| timeoutExecutions | # of **timeout** Executions per minute |
| staleExecutions | # of **stale** Executions per minute |
| totalExecutions | # of **total** Executions per minute |
| responseTimeMS | Avg duration of **normal** executions (1 minute average) in ms |
| states|timeout|<stateName> | # of **timeout** executions where the last state was <stateName> |
| states|error|<stateName> | # of **error** executions where the last state was <stateName> |
| states|stale|<stateName> | # of **stale** executions where the last state was <stateName> |

# Apendix A: Sample Log output explained

TODO: Run a test run using a proper DB config file and explain the log output