# Ambrosia Documentation

***Release 0.9.0***

**Wolfgang Ettlinger**

April 18, 2015

# CONTENTS

Ambrosia is a framework that takes the information from ANANAS reports, applies several operations (time adjustments, correlations, etc) and presents this data in a viewable form.

CHAPTER

# ONE

# CONTENTS

## 1.1 Ambrosia Client Documentation

### 1.1.1 Built-In Namespace _global_

#### Methods

##### Class

`Class(name, p1[, p2])`

**Arguments**

- **name** (*String*) – the fully qualified name of the new class
- **p1** (*String|Object*) – if two parameters are passed: the object containing class members, else the superclass
- **p2** (*Object*) – the obj object containing class members

**Returns class** the newly created class

creates a class

### 1.1.2 Namespace ambrosia_web

#### Constructor

class `ambrosia_web()`

#### Methods

##### init

`ambrosia_web.init()`

initialize Ambrosia

#### redraw

`ambrosia_web.redraw()`

Redraws all views of the application

## 1.1.3 Namespace ambrosia_web.entity

### Constructor

**class** `ambrosia_web.entity()`

### Methods

#### enrich

`ambrosia_web.entity.enrich(el)`

**Arguments**

- **el** (*object*) – the deserialized data

Receives an object containing the deserialized data from the server and returns an instance of the class `ambrosia_web.entity.Entity()`

### Attributes

#### onSelectHandler

**`onSelectHandler`**

contains all handlers for selecting entities. Any part of the application may listen to those events (i.e. add a function to this array). If the user select an entity the interface can adapt to this (e.g. the `ambrosia_web.view.entityview.EntityView()` shows details about this entity).

## 1.1.4 Class ambrosia_web.entity.Entity

The client side counterpart for an entity

**See also:**

`ambrosia.model.Entity`

### Constructor

**class** `ambrosia_web.entity.Entity()`

#### Methods

##### getLink

`ambrosia_web.entity.Entity.`**`getLink`**`()`

**Returns jQuery** the link

Returns a jQuery element containing a link that, when clicked, selects the entity.

##### resolveReference

`ambrosia_web.entity.Entity.`**`resolveReference`**`()`

resolves all references

**See also:**

`ambrosia.model.Event.to_serializeable()`

##### select

`ambrosia_web.entity.Entity.`**`select`**`()`

This method should be called when the user selects an entity.

### 1.1.5 Namespace ambrosia_web.entity.entities

#### Constructor

**class** `ambrosia_web.entity.`**`entities`**`()`

### 1.1.6 Class ambrosia_web.entity.entities-App

Represents `ambrosia.model.entities.App`

#### Constructor

**class** `ambrosia_web.entity.`**`entities-App`**`()`

### 1.1.7 Class ambrosia_web.entity.entities-File

Represents `ambrosia.model.entities.File`

#### Constructor

**class** `ambrosia_web.entity.`**`entities-File`**`()`

### 1.1.8 Class ambrosia_web.entity.entities-ServerEndpoint

Represents `ambrosia.model.entities.ServerEndpoint`

#### Constructor

**class** `ambrosia_web.entity.`**`entities-ServerEndpoint`**`()`

### 1.1.9 Class ambrosia_web.entity.entities-Task

Represents `ambrosia.model.entities.Task`

#### Constructor

**class** `ambrosia_web.entity.`**`entities-Task`**`()`

### 1.1.10 Namespace ambrosia_web.event

#### Constructor

**class** `ambrosia_web.`**`event`**`()`

#### Methods

##### clearSelect

`ambrosia_web.event.`**`clearSelect`**`()`

unselect all events

##### enrich

`ambrosia_web.event.`**`enrich`**`(`*el*, *parent*`)`

**Arguments**

- **el** (*object*) – the deserialized data
- **parent** (*ambrosia_web.event.Event*) – the events parent event (if exists)

Receives an object containing the deserialized data from the server and returns an instance of the class `ambrosia_web.event.Event()`

##### reset

`ambrosia_web.event.`**`reset`**`()`

Resets the default `A.layout.BlockLayoutManager()`

### Attributes

#### BLOCK_MARGIN_X

**BLOCK_MARGIN_X**

The horizontal space Ambrosia should keep between two adjacent event

#### BLOCK_MARGIN_Y

**BLOCK_MARGIN_Y**

The vertical space Ambrosia should keep between two adjacent event

#### BLOCK_PADDING

**BLOCK_PADDING**

The horizontal space Ambrosia should keep between the borders of a child event and its parent (in pixel)

#### BLOCK_WIDTH

**BLOCK_WIDTH**

The minimum width of a block (in pixel)

#### DEFAULT_BLOCK_HEIGHT

**DEFAULT_BLOCK_HEIGHT**

The default height in seconds for an event

#### DEFAULT_BLOCK_LAYOUT_MANAGER

**DEFAULT_BLOCK_LAYOUT_MANAGER**

The default `ambrosia_web.layout.BlockLayoutManager()` that is used on the top level

#### onSelectHandler

**onSelectHandler**

contains all handlers for selecting events. Any part of the application may listen to those events (i.e. add a function to this array). If the user select an entity the interface can adapt to this (e.g. the `ambrosia_web.view.detailsview.DetailsView()` shows details about this event).

#### onUnSelectHandler

**`onUnSelectHandler`**

contains all handlers for unselecting events. Any part of the application may listen to those events (i.e. add a function to this array). If the user unselect an entity the interface can adapt to this (e.g. the `ambrosia_web.view.detailsview.DetailsView()` shows details about this event).

## 1.1.11 Class ambrosia_web.event.BlockEvent

Base class for all events that are drawn as a block.

### Constructor

**class** `ambrosia_web.event.`**`BlockEvent`**`()`

Bases: `ambrosia_web.event.Event()`

### Methods

#### calcDimensions

`ambrosia_web.event.BlockEvent.`**`calcDimensions`**`(`*blockLayoutManager*`)`

**Arguments**

- **blockLayoutManager** (*ambrosia_web.layout.BlockLayoutManager*) – the block layout manager to use

Calculates the dimensions of the visualisation (for block events). The top level events are drawn using the default block layout manager. Each event that has visible children creates a new block layout manager that is used to position the children (the children's calcDimensions method is called). The block layout manager that was used to position the children holds the width and height that is required to draw all children. Afterwards (using this width/height) the parent event is drawn.

#### draw

`ambrosia_web.event.BlockEvent.`**`draw`**`(`*xOffset*`)`

**Arguments**

- **xOffset** (*int*) – (optional) if this is a child object, the x position of the parent

draws the event

## 1.1.12 Class ambrosia_web.event.Event

The client side counterpart for an event

**See also:**

`ambrosia.model.Event`

### Constructor

**class** `ambrosia_web.event.`**`Event`**`()`

### Methods

#### calcDimensions

`ambrosia_web.event.Event.`**`calcDimensions`**`(`*blockLayoutManager*`)`

**Arguments**

- **blockLayoutManager** (*ambrosia_web.layout.BlockLayoutManager*) – the block layout manager to use

Calculates the dimensions of the visualisation (for block events). Should be called second when drawing. events.

#### calcVisible

`ambrosia_web.event.Event.`**`calcVisible`**`()`

This is the first method called when drawing events. It calculates if an element should be shown and also considers the visibility of the child elements (a child can force it's parent to show)

#### draw

`ambrosia_web.event.Event.`**`draw`**`()`

Draw the event. Should be called third when drawing. Must be implemented by subclass.

#### getLink

`ambrosia_web.event.Event.`**`getLink`**`()`

**Returns jQuery** the link

Returns a jQuery element containing a link that, when clicked, selects the event.

#### select

`ambrosia_web.event.Event.`**`select`**`()`

This method should be called when the user selects one event.

#### selectAdd

`ambrosia_web.event.Event.`**`selectAdd`**`()`

This method should be called when the user adds an event to a selection.

#### unselect

`ambrosia_web.event.Event.unselect()`

This method should be called when the user unselects one event.

## 1.1.13 Class ambrosia_web.event.LineEvent

Base class for all events that are drawn as a horizontal line across the main view.

### Constructor

**class** `ambrosia_web.event.LineEvent()`

Bases: `ambrosia_web.event.Event()`

### Methods

#### draw

`ambrosia_web.event.LineEvent.draw()`

draws the line

## 1.1.14 Namespace ambrosia_web.event.events

### Constructor

**class** `ambrosia_web.event.events()`

## 1.1.15 Class ambrosia_web.event.events-ANANASAdbShellExec

Represents `ambrosia_plugins.lkm.events.ANANASAdbShellExec`

### Constructor

**class** `ambrosia_web.event.events-ANANASAdbShellExec()`

## 1.1.16 Class ambrosia_web.event.events-ANANASAdbShellExecEvent

Represents `ambrosia_plugins.lkm.events.ANANASAdbShellExecEvent`

### Constructor

**class** `ambrosia_web.event.events-ANANASAdbShellExecEvent()`

### 1.1.17 Class ambrosia_web.event.events-ANANASEvent

Represents `ambrosia_plugins.events.ANANASEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-ANANASEvent`**`()`

### 1.1.18 Class ambrosia_web.event.events-APKInstallEvent

Represents `ambrosia_plugins.lkm.events.APKInstallEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-APKInstallEvent`**`()`

### 1.1.19 Class ambrosia_web.event.events-AndroidApicall

Represents `ambrosia_plugins.apimonitor.AndroidApicall`

#### Constructor

**class** `ambrosia_web.event.`**`events-AndroidApicall`**`()`

### 1.1.20 Class ambrosia_web.event.events-AndroidApicallEvent

Represents `ambrosia_plugins.apimonitor.AndroidApicallEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-AndroidApicallEvent`**`()`

### 1.1.21 Class ambrosia_web.event.events-AnonymousFileEvent

Represents `ambrosia_plugins.lkm.events.AnonymousFileEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-AnonymousFileEvent`**`()`

### 1.1.22 Class ambrosia_web.event.events-CallLogAccess

Represents `ambrosia_plugins.apimonitor.CallLogAccess`

#### Constructor

**class** `ambrosia_web.event.`**`events-CallLogAccess`**`()`

### 1.1.23 Class ambrosia_web.event.events-CallLogAccessEvent

Represents `ambrosia_plugins.apimonitor.CallLogAccessEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-CallLogAccessEvent`**`()`

### 1.1.24 Class ambrosia_web.event.events-CommandExecuteEvent

Represents `ambrosia_plugins.lkm.events.CommandExecuteEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-CommandExecuteEvent`**`()`

### 1.1.25 Class ambrosia_web.event.events-ContactAccessEvent

Represents `ambrosia_plugins.apimonitor.ContactAccessEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-ContactAccessEvent`**`()`

### 1.1.26 Class ambrosia_web.event.events-ContactsAccess

Represents `ambrosia_plugins.apimonitor.ContactsAccess`

#### Constructor

**class** `ambrosia_web.event.`**`events-ContactsAccess`**`()`

### 1.1.27 Class ambrosia_web.event.events-CreateDir

Represents `ambrosia_plugins.lkm.events.CreateDir`

#### Constructor

**class** `ambrosia_web.event.`**`events-CreateDir`**`()`

### 1.1.28 Class ambrosia_web.event.events-CreateDirEvent

Represents `ambrosia_plugins.lkm.events.CreateDirEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-CreateDirEvent`**`()`

### 1.1.29 Class ambrosia_web.event.events-DeleteFileEvent

Represents `ambrosia_plugins.lkm.events.DeleteFileEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-DeleteFileEvent`**`()`

### 1.1.30 Class ambrosia_web.event.events-DeletePathEvent

Represents `ambrosia_plugins.lkm.events.DeletePathEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-DeletePathEvent`**`()`

### 1.1.31 Class ambrosia_web.event.events-ExecEvent

Represents `ambrosia_plugins.lkm.events.ExecEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-ExecEvent`**`()`

### 1.1.32 Class ambrosia_web.event.events-FileEvent

Represents `ambrosia_plugins.lkm.events.FileEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-FileEvent`**`()`

### 1.1.33 Class ambrosia_web.event.events-JavaLibraryLoadEvent

Represents `ambrosia_plugins.lkm.events.JavaLibraryLoadEvent`

### Constructor

**class** `ambrosia_web.event.`**`events-JavaLibraryLoadEvent`**`()`

## 1.1.34 Class ambrosia_web.event.events-LibraryLoad

Represents `ambrosia_plugins.lkm.events.LibraryLoad`

### Constructor

**class** `ambrosia_web.event.`**`events-LibraryLoad`**`()`

## 1.1.35 Class ambrosia_web.event.events-LibraryLoadEvent

Represents `ambrosia_plugins.lkm.events.LibraryLoadEvent`

### Constructor

**class** `ambrosia_web.event.`**`events-LibraryLoadEvent`**`()`

## 1.1.36 Class ambrosia_web.event.events-MemoryMapEvent

Represents `ambrosia_plugins.lkm.events.MemoryMapEvent`

### Constructor

**class** `ambrosia_web.event.`**`events-MemoryMapEvent`**`()`

## 1.1.37 Class ambrosia_web.event.events-PhoneCall

Represents `ambrosia_plugins.apimonitor.PhoneCall`

### Constructor

**class** `ambrosia_web.event.`**`events-PhoneCall`**`()`

## 1.1.38 Class ambrosia_web.event.events-PhoneCallEvent

Represents `ambrosia_plugins.apimonitor.PhoneCallEvent`

### Constructor

**class** `ambrosia_web.event.`**`events-PhoneCallEvent`**`()`

### 1.1.39 Class ambrosia_web.event.events-SMSAccess

Represents `ambrosia_plugins.apimonitor.SMSAccess`

#### Constructor

**class** `ambrosia_web.event.`**`events-SMSAccess`**`()`

### 1.1.40 Class ambrosia_web.event.events-SMSAccessEvent

Represents `ambrosia_plugins.apimonitor.SMSAccessEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-SMSAccessEvent`**`()`

### 1.1.41 Class ambrosia_web.event.events-SendSignal

Represents `ambrosia_plugins.lkm.events.SendSignal`

#### Constructor

**class** `ambrosia_web.event.`**`events-SendSignal`**`()`

### 1.1.42 Class ambrosia_web.event.events-SendSignalEvent

Represents `ambrosia_plugins.lkm.events.SendSignalEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-SendSignalEvent`**`()`

### 1.1.43 Class ambrosia_web.event.events-SocketAccept

Represents `ambrosia_plugins.lkm.events.SocketAccept`

#### Constructor

**class** `ambrosia_web.event.`**`events-SocketAccept`**`()`

### 1.1.44 Class ambrosia_web.event.events-SocketAcceptEvent

Represents `ambrosia_plugins.lkm.events.SocketAcceptEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-SocketAcceptEvent`**`()`

### 1.1.45 Class ambrosia_web.event.events-SocketEvent

Represents `ambrosia_plugins.lkm.events.SocketEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-SocketEvent`**`()`

### 1.1.46 Class ambrosia_web.event.events-StartTaskEvent

Represents `ambrosia_plugins.lkm.events.StartTaskEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-StartTaskEvent`**`()`

### 1.1.47 Class ambrosia_web.event.events-SuperUserRequest

Represents `ambrosia_plugins.lkm.events.SuperUserRequest`

#### Constructor

**class** `ambrosia_web.event.`**`events-SuperUserRequest`**`()`

### 1.1.48 Class ambrosia_web.event.events-SuperUserRequestEvent

Represents `ambrosia_plugins.lkm.events.SuperUserRequestEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-SuperUserRequestEvent`**`()`

### 1.1.49 Class ambrosia_web.event.events-SyscallEvent

Represents `ambrosia_plugins.lkm.events.SyscallEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-SyscallEvent`**`()`

### 1.1.50 Class ambrosia_web.event.events-UnknownFdEvent

Represents `ambrosia_plugins.lkm.events.UnknownFdEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-UnknownFdEvent`**`()`

### 1.1.51 Class ambrosia_web.event.events-ZygoteForkEvent

Represents `ambrosia_plugins.lkm.events.ZygoteForkEvent`

#### Constructor

**class** `ambrosia_web.event.`**`events-ZygoteForkEvent`**`()`

### 1.1.52 Namespace ambrosia_web.filter

#### Constructor

**class** `ambrosia_web.`**`filter`**`()`

#### Methods

##### handleLogicalOperation

`ambrosia_web.filter.`**`handleLogicalOperation`**`(`*ex1*, *rest*`)`

**Arguments**

- **ex1** – an expression
- **rest** – an array containing a logical operation and a second expression or undefined

**Returns ***

Helper function for the parser.

#### Attributes

##### addFilterHandler

**`addFilterHandler`**

contains all handlers for adding filters to an event class. Any part of the application may listen to those events (i.e. add a function to this array). If the user select an entity the interface can adapt to this.

#### removeFilterHandler

**`removeFilterHandler`**

contains all handlers for removing filters from an event class. Any part of the application may listen to those events (i.e. add a function to this array). If the user select an entity the interface can adapt to this.

### 1.1.53 Class ambrosia_web.filter.BlacklistFilter

A blacklisting filter

#### Constructor

**class** `ambrosia_web.filter.`**`BlacklistFilter`**(*rule*, *description*, *enabled*)

**Arguments**

- **rule** (*String*) – the condition for the filter
- **description** (*String*) – a string describing the filter
- **enabled** (*bool*) – (optional) whether the filter is effective

### 1.1.54 Class ambrosia_web.filter.Comparison

A comparison. Used by the parser.

#### Constructor

**class** `ambrosia_web.filter.`**`Comparison`**(*p1*, *op*, *p2*)

**Arguments**

- **p1** – the first value that is compared
- **op** – the compare operation
- **p2** – the sencond value

### 1.1.55 Class ambrosia_web.filter.Filter

A Filter represents a single condition (either entered by the user or a default condition).

The following shows example for the filter syntax:

#### Examples

```
!(test == 1.2 || (test > 2 && p.bar != "foobar") || true ) && !false
```

The logical operations "&&" and '!!' as well as the unary logical operation "!" are allowed. Parentheses may be used to change the default precedence of the operations.

These logical operations manage "comparisons". A "comparison" may compare two values using the operators "==", "!=", ">=", "<=", "<", "~" (the first value matches a regex defined by the second value), ":" (the second value is an array and the first element is contained in the second one) and "!:" (the first value is not contained in the second value).

A value may be a string in the form of "string", a number in the form of 1.0 or 1, true or false or a property. A property is a string describing an attribute of an event (e.g. abspath, successful). Moreover a property may also match a specific reference (e.g. r.process.pid, r.file.abspath). The reference defined in a property may be a specific reference (like r.file or r.process). Moreover the string "*" may be used to get all values (e.g. r.*.id). Since multiple values are returned, the value must be treated as an array (Array operations ":" and "!:" must be used). A general filter (that is applied to all events regardless of their type) can therefore be used to find all events related to a certain entity (e.g. "someidofanentity" : r.*.id).

### Constructor

**class** `ambrosia_web.filter.`**`Filter`**`()`

### Methods

#### evaluate

`ambrosia_web.filter.Filter.`**`evaluate`**`()`

**Returns bool** true if the event matches

Evaluate if an an event matches this filter

#### isEnabled

`ambrosia_web.filter.Filter.`**`isEnabled`**`()`

**Returns bool** true if enabled

Checks whether this filter is enabled

#### setDescription

`ambrosia_web.filter.Filter.`**`setDescription`**`(`*d*`)`

**Arguments**

- **d** (*String*) – the description

set the description

#### setEnabled

`ambrosia_web.filter.Filter.`**`setEnabled`**`(`*b*`)`

**Arguments**

- **b** (*bool*) – whether the filter should be enabled

enable or disable the filter

#### setRule

`ambrosia_web.filter.Filter.`**`setRule`**(*r*)

**Arguments**

- **r** (*String*) – the new rule in filter syntax

replaces the current rule with a new one

### 1.1.56 Class ambrosia_web.filter.LogicalOperation

Logical operations like “&&” and ”!!”. Used by the parser

#### Constructor

**class** `ambrosia_web.filter.`**`LogicalOperation`**(*p1*, *op*, *p2*)

**Arguments**

- **p1** – the first expression
- **op** – the operation
- **p2** – the second expression

### 1.1.57 Class ambrosia_web.filter.Property

A property used in a filter. Used by the parser.

#### Constructor

**class** `ambrosia_web.filter.`**`Property`**(*s*)

**Arguments**

- **s** – the property string

### 1.1.58 Class ambrosia_web.filter.UnaryOperator

Unary operators. Used by the parser

#### Constructor

**class** `ambrosia_web.filter.`**`UnaryOperator`**(*op*, *expression*)

**Arguments**

- **op** – the operation e.g. NOT
- **expression** – the expression the operator is applied to

### 1.1.59 Namespace ambrosia_web.layout

#### Constructor

**class** `ambrosia_web.`**`layout`**`()`

### 1.1.60 Class ambrosia_web.layout.BlockLayoutManager

The block layout manager is used to position event block in the main view.

**See also:**

`ambrosia_web.layout.BlockLayoutManager.fitBlock()` for details.

Note: in order for the block layout manager to properly work, the events have to be fitted in ascending order (x position)

#### Constructor

**class** `ambrosia_web.layout.`**`BlockLayoutManager`**`()`

#### Methods

##### fitBlock

`ambrosia_web.layout.BlockLayoutManager.`**`fitBlock`**`(`*dim*, *margin_x*, *margin_y*`)`

**Arguments**

- **dim** (*ambrosia_web.layout.Dimensions*) – the dimensions of the block (may overlap other events)
- **margin_x** (*int*) – the horizontal margin that should be left
- **margin_y** (*int*) – the vertical margin that should be left

**Returns ambrosia_web.layout.Dimensions** the new dimensions of the non-overlapping block

Takes a `ambrosia_web.layout.Dimensions()` object and tries to fit it considering the previously fitted blocks.

##### getEndY

`ambrosia_web.layout.BlockLayoutManager.`**`getEndY`**`()`

**Returns number**

position bottom border of the block layout manager (considering all fitted events)

##### getWidth

`ambrosia_web.layout.BlockLayoutManager.`**`getWidth`**`()`

**Returns number**

get the width of the whole block layout manager (considering all fitted events)

### 1.1.61 Class ambrosia_web.layout.Dimensions

Helper class that represents the dimensions of a block

#### Constructor

**class** `ambrosia_web.layout.`**`Dimensions`**(*x*, *y*, *width*, *height*)

**Arguments**

- **x** – the x position
- **y** – the y position
- **width** – the width
- **height** – the height

### 1.1.62 Namespace ambrosia_web.util

#### Constructor

**class** `ambrosia_web.`**`util`**()

#### Methods

##### assert

`ambrosia_web.util.`**`assert`**(*b*)

**Arguments**

- **b** (*bool*) –

Simple helper function that raises an exception when false is passed

##### deserialize

`ambrosia_web.util.`**`deserialize`**(*obj*, *objs*)

**Arguments**

- **obj** – the obj from Ambrosia
- **objs** – the objs from Ambrosia

deserialize results from Ambrosia

### 1.1.63 Class ambrosia_web.util.Log

The class that handles logging

#### Constructor

**class** `ambrosia_web.util.`**`Log`**()

### Methods

#### D

`ambrosia_web.util.Log.`**`D`**`(`*str*`)`

**Arguments**

- **str** (*String*) – the message to log

shortcut for debug logging

#### E

`ambrosia_web.util.Log.`**`E`**`(`*str*`)`

**Arguments**

- **str** (*String*) – the message to log

shortcut for error logging

#### I

`ambrosia_web.util.Log.`**`I`**`(`*str*`)`

**Arguments**

- **str** (*String*) – the message to log

shortcut for info logging

#### log

`ambrosia_web.util.Log.`**`log`**`(`*str*, *level*`)`

**Arguments**

- **str** (*String*) – the message to log
- **level** (*String*) – the level: DEBUG, INFO, WARN, ERROR

log an event

#### W

`ambrosia_web.util.Log.`**`W`**`(`*str*`)`

**Arguments**

- **str** (*String*) – the message to log

shortcut for warn logging

### 1.1.64 Namespace ambrosia_web.view

#### Constructor

**class** `ambrosia_web.`**`view`**`()`

#### Methods

##### hideAllPanels

`ambrosia_web.view.`**`hideAllPanels`**`()`

hide all panels

### 1.1.65 Class ambrosia_web.view.Panel

Base class for all panels (DetailsView, EntityView, FilterView)

#### Constructor

**class** `ambrosia_web.view.`**`Panel`**(*name*, *element*)

**Arguments**

- **name** (*String*) – the caption of the panel
- **element** (*jQuery*) – the element to draw the panel into

### 1.1.66 Namespace ambrosia_web.view.detailsview

#### Constructor

**class** `ambrosia_web.view.`**`detailsview`**`()`

### 1.1.67 Class ambrosia_web.view.detailsview.DetailsView

Implements a simple view that shows details about the last event that has been selected

#### Constructor

**class** `ambrosia_web.view.detailsview.`**`DetailsView`**(*element*)

**Arguments**

- **element** (*jQuery*) – the jQuery element the view should be located

#### Methods

##### setup

`ambrosia_web.view.detailsview.DetailsView.`**`setup`**`()`

set up the details view

### 1.1.68 Namespace ambrosia_web.view.entityview

#### Constructor

**class** `ambrosia_web.view.`**`entityview`**`()`

### 1.1.69 Class ambrosia_web.view.entityview.EntityView

Implements a simple view that shows details about the selected entity

#### Constructor

**class** `ambrosia_web.view.entityview.`**`EntityView`**`(`*element*`)`

**Arguments**

- **element** (*jQuery*) – the jQuery element the view should be located

### 1.1.70 Namespace ambrosia_web.view.filterview

#### Constructor

**class** `ambrosia_web.view.`**`filterview`**`()`

### 1.1.71 Class ambrosia_web.view.filterview.FilterView

Implements a view that allows to view and modify filters

#### Constructor

**class** `ambrosia_web.view.filterview.`**`FilterView`**`()`

#### Methods

##### redraw

`ambrosia_web.view.filterview.FilterView.`**`redraw`**`()`

redraws the filter view

#### setup

`ambrosia_web.view.filterview.FilterView.`**`setup`**`()`

sets up the filterview

### 1.1.72 Namespace ambrosia_web.view.mainview

#### Constructor

**class** `ambrosia_web.view.`**`mainview`**`()`

#### Attributes

##### EXTRA_WIDTH

**`EXTRA_WIDTH`**

the extra horizontal space that should be left after the last event

##### X_OFFSET

**`X_OFFSET`**

the x offset where events may be drawn

### 1.1.73 Class ambrosia_web.view.mainview.MainView

the main view showing all events in a timeline

#### Constructor

**class** `ambrosia_web.view.mainview.`**`MainView`**`()`

#### Methods

##### getHeight

`ambrosia_web.view.mainview.MainView.`**`getHeight`**`()`

**Returns number** the height

get the height of the main view

##### getWidth

`ambrosia_web.view.mainview.MainView.`**`getWidth`**`()`

**Returns number** the width

get the width of the main view

#### redraw

`ambrosia_web.view.mainview.MainView.`**`redraw`**`()`

redraw the main view

#### setup

`ambrosia_web.view.mainview.MainView.`**`setup`**`()`

set up the main view

#### setWidth

`ambrosia_web.view.mainview.MainView.`**`setWidth`**`(`*val*`)`

**Arguments**

- **val** (*number*) – the width

set the width of the main view

### 1.1.74 Overview

This section gives a short overview of the internal workings of Ambrosia Web. For a detailed description please see the documentation for the packages.

The function `ambrosia_web.init()` loads the serialized data (specified after the hash symbol in the URL), enriches the events and entities (`ambrosia_web.entity.enrich()` and `ambrosia_web.event.enrich()`) and resolves all references of the entities (`ambrosia_web.entity.Entity.resolveReferences()`). Afterwards all views are set up (`ambrosia_web.view.mainview.MainView()`, `ambrosia_web.view.entityview.EntityView()`, `ambrosia_web.view.detailsview.DetailsView()`, :js:class:‘ambrosia_web.view.filterview.FilterView).

The main view shows all events on a timeline. The method `ambrosia_web.view.mainview.MainView.redraw()` uses the following methods to draw the events:

- `ambrosia_web.event.Event.calcVisible()`: calculates if the event should be drawn at all (i.e. whether it is filtered)
- `ambrosia_web.event.Event.calcDimensions()`: calculates where the event should be drawn and how big it should be
- `ambrosia_web.event.Event.draw()`: draws the element.

Each of theses methods may call the corresponding methods on child events (e.g. a parent event needs to know about the positions of the children to decide how big it should be).

Ambrosia defines two types of events:

- a `ambrosia_web.event.BlockEvent()` is drawn as a block in the main view
- a `ambrosia_web.event.LineEvent()` is drawn as a line across the main view (children are not drawn)

In order for a block event to decide where it should be drawn the `ambrosia_web.layout.BlockLayoutManager()` is used. This class remembers the relevant block events that have already been drawn and allows an event to find a position where enough free space is available. the block layout manager is used on the top level and to position children of an event. Each event with children creates a new block layout manager.

Events and entities can be selected (see `ambrosia_web.event.Event.select()` `ambrosia_web.event.Event.selectAdd()`, `ambrosia_web.event.Event.unselect()`. `ambrosia_web.event.clearSelect()`, `ambrosia_web.entity.Entity.select()`). Any part of the application may select an entity or an event and all parts of the application may register to select and unselect events (see `ambrosia_web.event.onSelectHandler`, `ambrosia_web.event.onUnSelectHandler`, `ambrosia_web.entity.onSelectHandler`). Multiple events may be selected but only one entity can be selected.

Each event class specifies filters. For an event all filters have to match for the event to be shown. General filters are applied to all events (see `ambrosia_web.event.Event()`). Those the rules for these filters follow a specific syntax (see `ambrosia_web.filter.Filter()`).

## 1.2 Ambrosia Server Documentation

==

### 1.2.1 ambrosia package

#### Subpackages

##### ambrosia.clocks package

**Module contents**

**class** `ambrosia.clocks.`**`ClockSyncer`**(*context*)

Bases: `object`

Used to synchronize all events.

This class manages the **translate_table**. This Array has the following structure:

```
[
    (time, error)
]
```

where * *time* is a timestamp (datetime.datetime) when the emulator time has changed (in **emulator time**) and * *error* is the datetime.timedelta of how much the emulator time is in the future

The entries have to be sorted by *time*.

**Warning:** This class assumes that when the emulator is started, the times are synchronized.

**Warning:** This class assumes that the emulator clock is always turned ahead (and never back). Currently this is the case since ANANAS tries to trigger behaviour that occur when the sample has been installed for a while.
This also poses a theoretical issue e.g. if the emulator time is 17:00 at boot and at 17:02 the clock is turned back to 17:00. An event occurring at 17:01 can either have happened at 17:01 or 17:03.

**Warning:** This class assumes that all timestamps have the same time zone (local time).

**Parameters** **context** (*ambrosia_web.context.AmbrosiaContext*) – The current context.

**emu_time**(*t*)
Calculate host time from a given emulator timestamp.

The method goes through all entries and finds the first entry where the given emulator timestamp is greater than the *time*. This means that the timestamp occurs after this emulator clock change. If no such entry is found, the emulator clock is assumed to be in sync with the host clock.

### ambrosia.config package

#### Module contents

**class** `ambrosia.config.`**Config**(*configfile*)
Bases: `ConfigParser.SafeConfigParser`

Allows simple access to the configuration file (currently not used or implemented)

### ambrosia.context package

#### Module contents

**class** `ambrosia.context.`**AmbrosiaContext**(*configfile*)
Bases: `object`

Objects of this class hold all relevant information for **one** run of Ambrosia:

- *config* (`ambrosia_web.config.Config`): the configuration
- *db* (`ambrosia_web.db.AmbrosiaDb`): the database (currently not used)
- *analysis* (`ambrosia_web.model.Analysis`): the object containing the Analysis results.
- *clock_syncer* (`ambrosia_web.clocks.ClockSyncer`): used to syncronize clocks (emulator <-> host)
- *plugin_manager* (`ambrosia_web.plugins.PluginManager`): the object holding information about the Ambrosia plugins

**Parameters** **configfile** (*str*) – path to configuration file

### ambrosia.db package

#### Module contents

**class** `ambrosia.db.`**AmbrosiaDb**(*context*)
Bases: `object`

For future use: persistently store objects in Memory using ZODB.

Currently the memory-footprint of Ambrosia is reasonable. However, Ambrosia is designed to be stored in ZODB. This database allows transparent storage to disk if memory becomes scarce. ZODB also uses certain data structures optimized (e.g. BTree module). Ambrosia already uses these data structures. The following classes are already designed to be stored in ZODB:

- `ambrosia_web.model.Analysis`
- `ambrosia_web.model.Entity`
- `ambrosia_web.model.Event`

**Parameters** **context** (*ambrosia_web.context.AmbrosiaContext*) – The current context.

## ambrosia.model package

### Submodules

#### ambrosia.model.entities module

**class** `ambrosia.model.entities.`**`App`**`(`*context*, *package*`)`

Bases: `ambrosia.model.Entity`

**static** **`find`**`(`*context*, *entities*, *identifier_btree*, *package*`)`

**`get_serializeable_properties`**`()`

**class** `ambrosia.model.entities.`**`File`**`(`*context*, *abspath*`)`

Bases: `ambrosia.model.Entity`

Represents file (existing or not) on the emulator.

**Parameters**

- **context** (*ambrosia_web.context.AmbrosiaContext*) – the current context
- **abspath** (*str*) – the absolute path of the file

**static** **`find`**`(`*context*, *entities*, *identifier_btree*, *abspath*`)`

**`get_serializeable_properties`**`()`

**`matches_entity`**`(`*abspath*`)`

**static** **`unknown`**`(`*context*`)`

Get the file representing unknonw files

**Parameters** **context** (*ambrosia_web.context.AmbrosiaContext*) – the current context

**class** `ambrosia.model.entities.`**`ServerEndpoint`**`(`*context*, *protocol*, *address*, *port=None*`)`

Bases: `ambrosia.model.Entity`

Represents a server endpoint i.e. a server and port.

**Parameters**

- **context** (*ambrosia_web.context.AmbrosiaContext*) – the current context
- **protocol** (*str*) – the network protocol used (e.g. TCP)

**static** **`find`**`(`*context*, *entities*, *identifier_btree*, *protocol*, *address*, *port*`)`

**`get_serializeable_properties`**`()`

**class** `ambrosia.model.entities.`**`Task`**`(`*context*, *pid*, *start_ts*, *end_ts*`)`

Bases: `ambrosia.model.Entity`

Represents a process or thread running on the emulator.

**Parameters**

- **context** (*ambrosia_web.context.AmbrosiaContext*) – the current context
- **pid** (*int*) – the PID/TID of the task
- **start_ts** (*datetime.datetime*) – the timestamp the task started or *None* if unknown
- **end_ts** (*datetime.datetime*) – the timestamp the task ended or *None* if unknown

**static** **`find`**`(`*context*, *entities*, *identifier_btree*, *pid*, *start_ts*, *end_ts*`)`

**`get_serializeable_properties`**`()`

**is_process**
whether this task is a process rather than a thread

**Module contents**

**class** `ambrosia.model.`**`Analysis`**
Bases: `persistent.Persistent`

An Analysis object (and the referenced objects) stores all information the Ambrosia analysis found out.

Analysis also manages all (top-level) Events and Entities (see `ambrosia_web.model.Event`, `ambrosia_web.model.Entity`) and tries to optimize for searching performance.

**`add_entity`**(*context*, *cls*, **args*)
Add an entity (alias for `Analysis.get_entity()`)

**`add_event`**(*evt*)
Add event and generate indices

**Parameters** **evt** (*Event*) – the event to add

**Warning:** The indexed attributes of an event may not be altered after the event has been added (otherwise the indices are out of date). This means that only static values may be indexed.

**`adjust_times`**(*context*)
Goes through all events and calls adjust_times on all Events

**`del_event`**(*evt*)
Delete event and update indices

**Parameters** **evt** (*Event*) – the event to remove

**`get_entity`**(*context*, *cls*, **args*)
Search for a specific entity, if it does not exist, create a new one

**Parameters**

- **context** (*ambrosia_web.context.AmbrosiaContext*) – the current context
- **cls** (*class*) – the class of the entity we are looking for
- ***args** – the arguments that would construct an entity

This method uses `Entity.find()` (of the specific entity class) to search for entities. This method receives a List of all known entitites of that class as well as a `BTrees.OOBTree.BTree` also containing all entities (indexed by their *primary_identifier* to allow more efficient searching). Moreover this method relieves the *args argument.

The *args argument contains all information that identifies a certain entity. This could be e.g. the IP address and the port of a server. Those values are passed to the find method. If the server is already known, the entity representing it is returned. If no such server entity exist a new one is created using those two parameters.

This behaviour makes sure that multiple event referencing the same entity all have references to the exact same entity in memory.

**`iter_all_events`**(*context*, *key=None*, *min_value=None*, *max_value=None*, *value=None*)

**`iter_entities`**(*context*, *cls*)
iterate all known entities of a specific class.

**Parameters**

- **context** (*ambrosia_web.context.AmbrosiaContext*) – the current context
- **cls** (*class*) – the class of the entity we are looking for

**`iter_events`**(*context*, *cls=None*, *key=None*, *min_value=None*, *max_value=None*, *value=None*)
Iterates over all events matching specific conditions in an efficient manner.

**Parameters**

- **context** (*ambrosia_web.context.AmbrosiaContext*) – the current context
- **cls** (*class*) – the class of the events we are looking for
- **key** – the key we are searching for
- **min_value** – the minimum value
- **max_value** – the maximum value
- **value** – the specific value (to search for exactly one value)

This method uses an internal indizes to efficiently select specific events. Each event class defines attributes that should be indexed (`Event`.indices). This class makes sure that those attributes can be searched for very fast.

The method accepts the following combinations of argument: * nothing: return all events * *cls*: return all events of a specific class (inefficient) * *cls*, *key*, *min_value* and/or *max_value*: search for all events of a specific class where the attribute *key* is within the defined value constraints * *cls*, *key*, *value*: search all events of a specific class where the attribute *key* has the value *value*

**`to_serializeable`**()
Returns all results in a serializable form

**class** `ambrosia.model.`**`Entity`**(*primary_identifier*)
Bases: `persistent.Persistent`

An Entity represents a static element without a timestamp e.g. a file or a server.

**Parameters** **primary_identifier** (*str*) – A identifier that identifies the entity. This does not have to be unique (e.g. PID).

**static `find`**(*context*, *entities*, *identifier_btree*, **args*)
Should find and return an entity based on the *args. Must be implemented by subclass.

**Parameters**

- **entities** (*list*) – all entities known
- **identifier_btree** (*BTrees.OOBTree.BTree*) – a binary tree where the keys are the primary identifier and the values are a list containg the matching entity.
- ***args** – the arguments identifying the entity. Must be identical to the constructor parameters.

**`get_serializeable_properties`**()
Should return all information relevant about the specific entity. Must be implemented by subclass.

**`primary_identifier`**
Returns the primary identifier for the entity.

**`primary_key`** = **None**
A generated unique key

**`to_serializeable`**()
Returns a dict containing all relevant information about the entity.

**class** `ambrosia.model.`**`Event`**(*start_ts=None*, *end_ts=None*)

Bases: `persistent.Persistent`

Event represents any event with a start-time and/or end-time

**Parameters**

- **start_ts** (*datetime.datetime*) – the time the event began
- **end_ts** (*datetime.datetime*) – the time the event ended

**`add_child`**(*c*)

Add child to this event. Also checks whether new child already has a parent (this is not allowed in a tree structure) and updates timestamps.

**`adjust_times`**(*context*)

Adjust times (e.g. emulator time -> system time)

**Parameters** **context** (*ambrosia_web.context.AmbrosiaContext*) – the current context

**`children`**

Iterates over all children.

**`cmp_by_time`**(*other*)

Compares two events by start timestamp

**Parameters** **other** (*Event*) – the other event

**`end_ts`**

The end timestamp if set else the start timestamp (assuming that start timestamp = end timestamp)

**Returns** The end timestamp

**`get_serializeable_properties`**()

This method is used for serialisation, has to be implemented by the subclass. Should return a dict with all important information about the event.

**`indices`** **= set([])**

This set contains all attributes that can be searched for; these attributes MUST NOT be CHANGED after the event has been added

**`sort`**()

Sort events by start timestamp

**`start_ts`**

The start timestamp if set else the end timestamp (assuming that start timestamp = end timestamp)

**Returns** The start timestamp

**`to_serializeable`**()

Returns a dict that can be used for serialization.

The primary keys of entities this entity refers to (e.g. parent process) are stored in the attribute “references”. This way any entity only has to be transmitted once, when the entity is referenced only the primary key is used.

### ambrosia.plugins package

#### Module contents

**class** `ambrosia.plugins.`**`PluginInfoTop`**

Bases: `object`

The base class to all PluginInfo classes. Every plugin must define a class named *PluginInfo* in the base module of the plugin.

**static** `correlators()`
Should return a list with tuples containing a `ambrosia_web.Correlator` and the priority (int)

**static** `parsers()`
Should return a list with all defined `ambrosia_web.ResultParser` classes.

**class** `ambrosia.plugins.`**`PluginManager`**
Bases: `object`

Manages all Ambrosia plugins

**`correlators`**`()`
Iterate all correlators (sorted by priority)

**`find`**`()`
Finds all plugins and gathers information about them.

**`parsers`**`()`
Returs a set with all parsers

### ambrosia.util package

#### Submodules

#### ambrosia.util.log module

**class** `ambrosia.util.log.`**`AmbrosiaFormater`**(*use_colors*)
Bases: `logging.Formatter`

A custom log formatter that can use colors

**`color_mapping`** **= {'INFO': '1;35', 'CRITICAL': '1;31', 'WARN': '1;33', 'WARNING': '1;33', 'ERROR': '1;31', 'DEB**

**`format`**(*record*)

`ambrosia.util.log.`**`init_logging`**(*log_level*)
Initialize logging to stderr

**Parameters** **log_level** (*str*) – the minimum log level

#### Module contents

**exception** `ambrosia.util.`**`SerializationError`**
Bases: `exceptions.Exception`

Indicates that something went wrong during serialization

`ambrosia.util.`**`classname`**(*cls*)
Returns the full class name of a class

`ambrosia.util.`**`get_class`**(*name*)

`ambrosia.util.`**`get_logger`**(*o*)
Create a logger for a object.

**Parameters** **o** (*object*) – the *self* reference of a object

`ambrosia.util.`**`join_command`**(*lst*)
Convert a list of arguments (argv) to a command line

`ambrosia.util.`**`js_date`**(*date*)
Converts a datetime.datetime to a float timestamp for javascript

`ambrosia.util.`**`obj_classname`**`(`*o*`)`

Returns the full class name of an object

`ambrosia.util.`**`serialize_obj`**`(`*obj*, *fp*`)`

Serialize an object

**Parameters** **obj** (*object*) – the object to serialize

Returns a JSON-string containing the "hollow" object and a list with objects. All actual data is striped from the object and appended to the objects list.

For example this function converts the dict:

```
{
    'test': [None, 1, 'test']
}
```

into the following "hollow" object:

```
{
    1: [0, 2, 1]
}
```

and the following objects list: .. code-block:: python

[None, 'test', 1]

All the data in the "hollow" object references data in the objects list. E.g. *1* references 'test'.

This type of is used for compression. Since Ambrosia generates a lot of data containing the same string multiple times this serialization should reduce the size of the serialized data (since a string only has to be stored once in the objects list. E.g. in the example above the string 'test' is contained two times in the original data but only once in the objects list.

`ambrosia.util.`**`unique_id`**`()`

Generates a uniqe id

## Module contents

**class** `ambrosia.`**`Ambrosia`**`(`*root*, *configfile*`)`

Bases: `object`

This class is the main class that performs starts all actions

**Parameters**

- **root** (*xml.etree.Element*) – The document root of the XML report
- **configfile** (*str*) – the config file path

Upon object creation the report is being parsed. General information (such as the APK filename) as well as Plugin-specific values are obtained. Plugin-specific values are parsed using `ResultParser` instances.

**`adjust_times`**`()`

This method adjusts the timestamps of all events.

Since the emulator clock and the clock of the analysis machine may be different (e.g. when the simulation plugin turns time ahead) the timestamps of several Events (with timestamps comming from the emulator) have to be adjusted (to the clock of the analysis machine). See `ambrosia_web.clocks.ClockSyncer`.

This method should be called right after the `ambrosia_web.Ambrosia` class has been created.

**correlate**()

Correlates the events

This method finds all Correlaters (see `ambrosia_web.plugins.PluginManager`) and starts them.

A `Correlator` searches for specific events (at top level) and wraps them into other events. E.g. a open(), read() and close() SyscallEvents are wrapped into a FileEvent. The `Correlator` can also do several passes (e.g. wrap 3 events of type A into a event B, then wrap several B events and wrap them into a C event).

Should be called after `Ambrosia.adjust_times()`.

**serialize**(*fp*)

Serialize Events into a compact text format (see `ambrosia_web.util.serialize_obj()`).

Should be called after `Ambrosia.correlate()`.

**Returns** the serialized string

**class** ambrosia.**Correlator**(*context*)

Bases: `object`

Base class for Correlators.

A Correlator is called after all primitive events (like Syscalls, API calls etc.) have been acquired. The Correlator is responsible to find matching primitive events (or events generated by other Correlators) and wrap them into higher-level Events.

The `ambrosia_web.plugins.PluginInfoTop` specifies a priority for each Correlator. This allows to force a specific order in which the Correlators are called (e.g. if a Correlator relies on Events generated by another Correlator).

**correlate**()

**Must** be implemented by the specific class.

**update_tree**()

This method may be used by subclasses to update the result event tree.

If the subclass uses the `ambrosia_web.model.Event.iter_events()` in a loop it may not add or remove events from the event tree. Otherwise events may be skipped or processed twice. Therefore the subclass may use the *to_add* and *to_remove* attributes to store events that should be added and removed from the top level of the event tree. Afterwards this method can be used to process the pending adds/removes.

**class** ambrosia.**ResultParser**

Bases: `object`

Allows a plugin to implement parsers for the results in the XML report (Abstract base class).

When the *result* section of a report is parsed **all** ResultParsers of all plugins are called for each result section. Each ResultParser may generate primitive events from the supplied XML Element.

**finish**(*context*)

Called after all parsing has been done

**Parameters** **context** (*ambrosia_web.context.AmbrosiaContext*) – The current context.

**parse**(*name*, *el*, *context*)

The actual parsing routine **must** be implemented by the specific class.

**Parameters**

- **name** (*str*) – The name of the tag (child of the *results* element).
- **el** (*xml.etree.Element*) – The result element to parse.

- **context** (*ambrosia_web.context.AmbrosiaContext*) – The current context.

**prepare** (*context*)

Called before any parsing is done by any ResultParser. **May** be overwritten by specific class.

**Parameters** **context** (*ambrosia_web.context.AmbrosiaContext*) – The current context.

static **start_parsers** (*el*, *context*)

Starts all ResultParsers registered in the `ambrosia_web.plugins.PluginManager`.

**Parameters** **context** (*ambrosia_web.context.AmbrosiaContext*) – The current context.

### 1.2.2 ambrosia_plugins package

#### Subpackages

##### ambrosia_plugins.apimonitor package

**Module contents**

class `ambrosia_plugins.apimonitor.`**AndroidApicallEvent** (*api*, *method*, *params*, *returnval*, *start_ts*)

Bases: `ambrosia.model.Event`

Represents an API call of the App

**Parameters**

- **api** (*str*) – the class referenced by this API call
- **method** (*str*) – the method called
- **returnval** (*str*) – the return value
- **start_ts** (*datetime.datetime*) – the time the API call occurred (emulator clock)

**adjust_times** (*context*)

**get_serializeable_properties** ()

**indices** = {}

class `ambrosia_plugins.apimonitor.`**ApiCallCorrelator** (*context*)

Bases: `ambrosia.Correlator`

Goes through all API calls and wraps known API calls into higher-level events.

**Parameters** **context** (*ambrosia_web.context.AmbrosiaContext*) – the current context.

**correlate** ()

class `ambrosia_plugins.apimonitor.`**ApimonitorPluginParser**

Bases: `ambrosia.ResultParser`

The plugin parser that parses the apimonitor tag

**finish** (*context*)

**parse** (*name*, *el*, *context*)

class `ambrosia_plugins.apimonitor.`**CallLogAccessEvent**

Bases: `ambrosia.model.Event`

App accesses call logs

**get_serializeable_properties** ()

**indices** = {}

class ambrosia_plugins.apimonitor.**ContactAccessEvent**

Bases: ambrosia.model.Event

App accesses contacts

**get_serializeable_properties**()

**indices** = {}

class ambrosia_plugins.apimonitor.**PhoneCallEvent**

Bases: ambrosia.model.Event

App calls someone

**get_serializeable_properties**()

**indices** = {}

class ambrosia_plugins.apimonitor.**PluginInfo**

Bases: ambrosia.plugins.PluginInfoTop

static **correlators**()

static **parsers**()

class ambrosia_plugins.apimonitor.**SMSAccessEvent**

Bases: ambrosia.model.Event

App accesses SMS

**get_serializeable_properties**()

**indices** = {}

### ambrosia_plugins.events package

#### Module contents

class ambrosia_plugins.events.**ANANASEvent**(*name*, *timestamp*, *params*)

Bases: ambrosia.model.Event

Represents an event generated by the ANANAS analysis system itself. Any action performed by ANANAS that affects the emulator (e.g. execution of a command) is recorded in a ANANASEvent.

**Parameters**

- **name** (*str*) – the type of the event
- **timestamp** (*datetime.datetime*) – the time the event occurred (host clock)
- **params** – additional parameters (any serializable data structure)

**get_serializeable_properties**()

**indices** = set(['start_ts', 'name'])

class ambrosia_plugins.events.**EventParser**

Bases: ambrosia.ResultParser

**parse**(*name*, *el*, *context*)

class ambrosia_plugins.events.**PluginInfo**

Bases: ambrosia.plugins.PluginInfoTop

static **correlators**()

**static parsers** ()

## ambrosia_plugins.lkm package

### Submodules

#### ambrosia_plugins.lkm.events module

**class** `ambrosia_plugins.lkm.events.`**`ANANASAdbShellExecEvent`** (*process*)

Bases: `ambrosia.model.Event`

Represents a command that has been executed by ANANAS

**`get_serializeable_properties`** ()

**`indices`** **= set([])**

**class** `ambrosia_plugins.lkm.events.`**`APKInstallEvent`** (*file*, *process*)

Bases: `ambrosia.model.Event`

**`get_serializeable_properties`** ()

**`indices`** **= set([])**

**class** `ambrosia_plugins.lkm.events.`**`AnonymousFileEvent`** (*description*, *process*, *context*, *successful=True*)

Bases: `ambrosia_plugins.lkm.events.FileEvent`

Represents an operation that happens on a file without a name (e.g. an unnamed pipe)

**`get_serializeable_properties`** ()

**`indices`** **= set(['process'])**

**class** `ambrosia_plugins.lkm.events.`**`CommandExecuteEvent`** (*path*, *command*, *process*, *execfile*)

Bases: `ambrosia.model.Event`

Represents the execution of a command (including fork, exec, library loads, etc.)

**`get_serializeable_properties`** ()

**`indices`** **= set(['process'])**

**class** `ambrosia_plugins.lkm.events.`**`CreateDirEvent`** (*start_ts*, *end_ts*, *process*, *successful*, *file*)

Bases: `ambrosia.model.Event`

Represents an mkdir() syscall

**`get_serializeable_properties`** ()

**`indices`** **= set(['process'])**

**class** `ambrosia_plugins.lkm.events.`**`DeletePathEvent`** (*start_ts*, *end_ts*, *successful*, *file*, *process*)

Bases: `ambrosia.model.Event`

Represents an unlink() syscall

**`get_serializeable_properties`** ()

**`indices`** **= set([])**

**class** `ambrosia_plugins.lkm.events.`**`ExecEvent`** (*start_ts*, *end_ts*, *path*, *argv*, *env*, *process*)

Bases: `ambrosia.model.Event`

Represents an execve() syscall

**get_serializeable_properties**()

**indices = set(['process'])**

**class** ambrosia_plugins.lkm.events.**FileDescriptorEvent**(*process*, *successful*)
Bases: ambrosia.model.Event

The base event for all file descriptor related events

**get_serializeable_properties**()

**indices = set(['process'])**

**class** ambrosia_plugins.lkm.events.**FileEvent**(*file*, *flags*, *mode*, *process*, *successful*)
Bases: ambrosia_plugins.lkm.events.FileDescriptorEvent

Represents a normal file operation on a file, directory or pipe

**get_serializeable_properties**()

**indices = set(['process', 'abspath'])**

**mode_flags = {'O_DSYNC': 4096, 'O_DIRECTORY': 65536, 'O_LARGEFILE': 32768, 'O_CREAT': 64, 'O_PATH': 2**

**class** ambrosia_plugins.lkm.events.**JavaLibraryLoadEvent**(*file*, *process*, *successful*, *system_library_load*)
Bases: ambrosia.model.Event

Represents dalvik library load operation

**get_serializeable_properties**()

**indices = set(['process'])**

**class** ambrosia_plugins.lkm.events.**LibraryLoadEvent**(*file*, *process*, *successful*)
Bases: ambrosia.model.Event

Represents mmap() operations on a library file

**get_serializeable_properties**()

**indices = set(['process'])**

**class** ambrosia_plugins.lkm.events.**MemoryMapEvent**(*flags*, *fd*, *address*, *process*, *return_value*, *start_ts*, *end_ts*)
Bases: ambrosia.model.Event

Represents a call to mmap(). It's parent normally is a ambrosia_plugins.lkm.events.FileDescriptorEvent

**get_serializeable_properties**()

**indices = set(['process'])**

**mmap_flags = {'MAP_NONBLOCK': 65536, 'MAP_EXECUTABLE': 4096, 'MAP_SHARED': 1, 'MAP_GROWSDOW**

**class** ambrosia_plugins.lkm.events.**SendSignalEvent**(*start_ts*, *end_ts*, *number*, *process*, *target_process*)
Bases: ambrosia.model.Event

Represents a kill() syscall

**get_serializeable_properties**()

**indices = set([])**

**class** `ambrosia_plugins.lkm.events.`**`SocketAcceptEvent`**(*process*, *successful*)

Bases: `ambrosia_plugins.lkm.events.FileDescriptorEvent`

Represents a successful accept() on a socket

This event's parent normally is a `ambrosia_plugins.lkm.events.SocketEvent` and it is a `ambrosia_plugins.lkm.events.FileDescriptorEvent` and therefore itself is a file descriptor operation.

**`get_serializeable_properties`**()

**`indices`** **= set(['process'])**

**class** `ambrosia_plugins.lkm.events.`**`SocketEvent`**(*process*, *successful*)

Bases: `ambrosia_plugins.lkm.events.FileDescriptorEvent`

Represents an operation on a socket

**`address_families`** **= {0: 'AF_UNSPEC', 1: 'AF_UNIX', 2: 'AF_INET', 3: 'AF_AX25', 4: 'AF_IPX', 5: 'AF_APPLE**

**`get_serializeable_properties`**()

**`indices`** **= set(['process'])**

**`sock_types`** **= {1: 'SOCK_STREAM', 2: 'SOCK_DGRAM', 3: 'SOCK_RAW', 4: 'SOCK_RDM', 5: 'SOCK_SEQPAC**

**class** `ambrosia_plugins.lkm.events.`**`StartTaskEvent`**(*start_ts*, *end_ts*, *process*, *child_pid*, *spawned_child*)

Bases: `ambrosia.model.Event`

Represents a fork()-like syscall

**`get_serializeable_properties`**()

**`indices`** **= set([])**

**class** `ambrosia_plugins.lkm.events.`**`SuperUserRequestEvent`**(*start_ts*, *end_ts*, *process*)

Bases: `ambrosia.model.Event`

Indicates that the process tried to run "su"

**`get_serializeable_properties`**()

**`indices`** **= set([])**

**class** `ambrosia_plugins.lkm.events.`**`SyscallEvent`**(*context*, *props*, *time*, *monotonic_ts*, *process*, *idx*, *spawned_child=None*)

Bases: `ambrosia.model.Event`

Represents a system call from lkm

**`get_serializeable_properties`**()

**`indices`** **= set(['index', 'name'])**

**class** `ambrosia_plugins.lkm.events.`**`UnknownFdEvent`**(*process*, *fd_number*, *successful*)

Bases: `ambrosia_plugins.lkm.events.FileDescriptorEvent`

Represents a fd event where no syscall opening the fd has been found.

**`get_serializeable_properties`**()

**class** `ambrosia_plugins.lkm.events.`**`ZygoteForkEvent`**(*process*)

Bases: `ambrosia.model.Event`

**`get_serializeable_properties`**()

**`indices`** **= set(['process'])**

**Module contents**

**class** `ambrosia_plugins.lkm.`**`AdbCommandCorrelator`**(*context*)

Bases: `ambrosia.Correlator`

Find command executions that happen because of ANANAS (through ADB)

**`correlate`**()

**class** `ambrosia_plugins.lkm.`**`CommandExecuteCorrelator`**(*context*)

Bases: `ambrosia.Correlator`

Finds events that form the execution of a command.

- `ambrosia_plugins.lkm.events.StartTaskEvent`: indicate the creation of a new process
- `ambrosia_plugins.lkm.events.ExecEvent`: commands are started using a fork-and-exec
- **`ambrosia_plugins.lkm.events.LibraryLoad`: shortly after a fork indicates that a library is loaded that** is essential to run the command.
- `ambrosia_plugins.lkm.events.FileEvent`: several file events happen at the begin of a command execution

**`_find_file_events`**(*process*, *evt*, *start_ts*, *matches*)

**`_find_java_library_loads`**(*process*, *evt*, *start_ts*)

**`_find_library_loads`**(*process*, *evt*, *start_ts*)

**`_find_mkdir_events`**(*process*, *evt*, *start_ts*)

**`correlate`**()

**class** `ambrosia_plugins.lkm.`**`FileEventCorrelator`**(*context*)

Bases: `ambrosia.Correlator`

Finds library load events (mmap to *.so files)

**`correlate`**()

**class** `ambrosia_plugins.lkm.`**`InstallCorelator`**(*context*)

Bases: `ambrosia.Correlator`

**`correlate`**()

**class** `ambrosia_plugins.lkm.`**`LkmPluginParser`**

Bases: `ambrosia.ResultParser`

Parses the *process* and *syscalltrace* elements of the result set.

**`finish`**(*context*)

Calculate additional information for each process.

This method is executed after all processes have been parsed. This allows to reliably reference other processes (E.g. when the first process is being parsed no other proccess is known, therefore no other process can be referenced). The method sets the tg_leader and the parent. Moreover, it copies the reference to *fds* from the parent for all threads (in linux a thread *normally* shares FDs with its thread group leader).

**`parse`**(*name*, *el*, *context*)

Does the actual parsing.

- *process* element: All processes reported by the LKM/ANANAS are parsed and `ambrosia_web.model.entities.Task` entities are created. Moreover, the attributes * *ananas_id* (id in the ANANAS db) * *parent_id* (the ANANAS db id of the parent task) * *comm* (description of the process in thekernel) * *path* (of the executable) * *type* (the type of the task

ANANAS figured out) * *fds* (a dict of all file descriptors and the path during LKM load) * *tdgid* (the PID of the task group leader) * *tg_leader_id* (The ANANAS db id of the thread group leader)

- *syscalltrace* element: A `ambrosia_plugins.lkm.events.SyscallEvent` event is create for each syscall using all the information ANANAS provides. Moreover the `ambrosia_web.clocks.ClockSyncer`.translate_table attribute is filled. ANANAS records two timestamps for each syscall. There is a *normal* timestamp (which is the system time when the syscall returned) and the *monotonic* timestamp (which is the time that passed since the system booted). When the system clock is not changed, the *monotonic* and the *normal* clock are in sync (e.g. if 10 seconds pass on one clock 10 seconds pass on the other clock). Therefore the *normal* clock is ahead of the *monotonic* clock (a constant offset = the time the emulator booted). By calculating the *normal* clock minus the *monotonic* clock we always get this offset. When this offset changes, the system clock has been altered.

  **This algorithm is implemented using the following variables:**

  – boot_time: the actual time the emulator is booted (calculated *normal* - *monotonic* time on the first syscall = when emulator time and host time are still in sync)

  – error: how much the expected offset (boot_time) is off from the acutal offset (*normal* - *monotonic*). This is also the error of the emulator clock (compared to the host clock)

  – adjtime: the adjusted time (the captured *normal* time - error).

  – lasterror: the error of the last syscall. If the error of two consecutive syscall changes, we know that the system clock has been altered (and we need to make an entry in `ambrosia_web.clocks.ClockSyncer`.translate_table). The comparison sees two errors that are at a maximum of 1 second apart as a clock change. This is because the error is not absolutely precise (the *monotonic* and *normal* timestamps are not captured at exactly the same time, even a context switch may happen in between).

**class** `ambrosia_plugins.lkm.`**`PluginInfo`**

Bases: `ambrosia.plugins.PluginInfoTop`

**static** **`correlators`**`()`

**static** **`parsers`**`()`

**class** `ambrosia_plugins.lkm.`**`SyscallCorrelator`**`(`*context*`)`

Bases: `ambrosia.Correlator`

Wraps primitive events into higher-level events

**`_check_syscall`**`(`*evt*`)`

Wraps a single syscall event into a higher-level event

**Parameters** **evt** (*ambrosia_plugins.lkm.events.SyscallEvent*) – the syscall event

**`_generate_start_fd_directory`**`()`

Generates the initial fd directory.

Before the correlation is started the fd directory is filed with file descriptor events of processes that existed before the LKM was loaded.

**`_get_del_fd_event`**`(`*fd*, *process*, *success*, *logname*, *clazz=None*`)`

Gets an fd event from the fd directory and deletes it.

**Parameters**

- **fd** (*int*) – the file descriptor number we are searching for
- **process** (*ambrosia_web.model.entities.Task*) – the task the fd belongs to
- **clazz** (*class*) – (optional) only return an event of this type

- **process** – the task the fd belongs to

**`_get_dup`**(*evt*, *oldfd*, *newfd*, *process*)

Duplicate an fd (dup and dup2 syscalls)

**Parameters**

- **evt** (*ambrosia_web.model.Event*) – the dup syscall event
- **oldfd** (*int*) – the old file descriptor number
- **newfd** (*int*) – the new file descriptor number

**`_get_fd_event`**(*fd*, *process*, *success*, *logname*, *clazz=None*, *default_start_ts=None*)

Get an fd event from the a fd directory entry.

The fd directory (*fd_directory*) is a dict in the form of

```
{
    pid: {
        fd_number: fd_event,
        ...
    },
    ...
}
```

The fd directory represents all file descriptors of the emulator **at a specific point in time**. This means that the fd directory is constantly changed as syscalls are being processed (e.g. open() creates an entry, close removes an entry).

If (for some reason) the fd is not found, this method returns an `ambrosia_plugins.lkm.events.UnknownFdEvent`.

**Note:** One value of the fd dictionary dict may be stored under multiple pid keys since tasks (especially threads) may share file descriptors.

**Parameters**

- **fd** (*int*) – the file descriptor number we are searching for
- **process** (*ambrosia_web.model.entities.Task*) – the task the fd belongs to
- **clazz** (*class*) – (optional) only return an event of this type
- **default_start_ts** (*datetime.datetime*) – if this fd is unknown, return an event with this start timestamp

**`_parse_addr_str`**(*addrstr*, *socket_evt*)

**`correlate`**()

`ambrosia_plugins.lkm.`**`_timedelta_diff`**(*td1*, *td2*)

### ambrosia_plugins.network package

#### Module contents

**class** `ambrosia_plugins.network.`**`PluginInfo`**

Bases: `ambrosia.plugins.PluginInfoTop`

This plugin is not implemented. Implement as soon as ANANAS properly supports network traffic analysis.

#### Module contents

### 1.2.3 processor module

`processor.main()`
The main method

### 1.2.4 Overview

This section gives a short overview of the internal workings of Ambrosia. For a detailed description please see the documentation for the modules.

The main function for the Ambrosia server side part is located in `processor`. The following shows the usage of the processor:

```
usage: processor.py [-h] [--config CONFIG]
                    [--loglevel {FATAL,ERROR,WARN,INFO,DEBUG}]
                    [--output OUTPUT]
                    [--output-type {serialized,none,tree,interactive}]
                    report

process ANANAS report for Ambrosia

positional arguments:
  report                the XML report input

optional arguments:
  -h, --help            show this help message and exit
  --config CONFIG       the config file
  --loglevel {FATAL,ERROR,WARN,INFO,DEBUG}
                        the log level for stderr
  --output OUTPUT       the output file, default is stdout
  --output-type {serialized,none,tree,interactive}
                        define what should be printed
```

The processor initializes logging (see `ambrosia.util.log.init_logging()`), reads the XML report and creates an `ambrosia.Ambrosia` instance. It adjusts the timestamps of events coming from the emulator (see `ambrosia.Ambrosia.adjust_times()`), correlates the events (`ambrosia.Ambrosia.correlate()`) and serializes them (`ambrosia.Ambrosia.serialize()`).

All the results are stored in an `ambrosia.model.Analysis` instance. The main two types of entries in the result are events (`ambrosia.model.Event`) and entities (`ambrosia.model.Entity`). All events and entities are managed by the `ambrosia.model.Analysis` class. All entities are defined in `ambrosia.model.entities`. The events are defined by each plugin.

#### Plugins

All plugins are defined in the `ambrosia_plugins` module. A plugin has to specify a `ambrosia.plugins.PluginInfoTop` class called "PluginInfo". This class should return all `ambrosia.Correlator` and `ambrosia.ResultParser` classes defined by the plugin.

A `ambrosia.ResultParser` is used to extract events from the report. A `ambrosia.Correlator` can be used to correlate and consolidate events.

#### `ambrosia_plugins.apimonitor`

The `ambrosia_plugins.apimonitor.ApimonitorPluginParser` generates events from the report. The `ambrosia_plugins.apimonitor.ApiCallCorrelator` is used to then find known API calls and wrap them into higher-level events.

#### `ambrosia_plugins.events`

This simple plugin parses all events created by ANANAS itself.

#### `ambrosia_plugins.lkm`

The lkm plugin handles events originating from the lkm. After the ambrosia_plugins.lkm.LkmPluginParser has parsed all the lkm related information from the report, the `ambrosia_plugins.lkm.SyscallCorrelator` wraps primitive syscall events into higher-level event (like `ambrosia_plugins.lkm.events.SendSignal` or `ambrosia_plugins.lkm.events.CreateDir`). The `ambrosia_plugins.lkm.FileEventCorrelator` is used to classify file events (to `ambrosia_plugins.lkm.events.LibraryLoad`). The `ambrosia_plugins.lkm.CommandExecuteCorrelator` then finds command executions. The `ambrosia_plugins.lkm.AdbCommandCorrelator` is then used to find command executions that have been caused by ANANAS itself.

#### `ambrosia_plugins.network`

This plugin is currently not implemented.