

# Index

[Index](#)

[Overview](#)

[Metadata Section](#)

[Goal Representation](#)

[Player Palette](#)

[Layout Section](#)

[Colors Section](#)

[Directions Section](#)

[Components Section](#)

[Execution Section](#)

[Player Section](#)

[File Validation](#)

[ME command line interface](#)

[Comments and Revisions](#)

[2016-04-12 Removed multiple of 3](#)

[2016-04-22 Workflow](#)

[2016-04-25 Strict JSON](#)

[2016-04-29 Goal structure details](#)

[2016-05-03 Player palette](#)

[2016-05-06 Goal structure details and speed](#)

[2016-05-10 Color updated](#)

[2016-05-17 Added customs, updated examples](#)

[2016-05-20 Several Changes](#)

[2016-05-24 Added arrow Component](#)

[2016-06-03 Rewrote execution section](#)

[2016-07-15 Added intermediate points in execution section](#)

[2016-09-06 Colors, events, simulation](#)

[2016-09-23 Several changes](#)

[2016-10-01 Event for exchange points](#)

[2016-10-13 Added -1 to diverters](#)

[2016-10-19 Added "Empty" to diverters](#)  
[2016-10-19 Added optional "thread\\_id to goal\\_struct](#)  
[2017-01-16 Added denominator for delivery component](#)  
[2017-02-20 Deprecated components](#)  
[2017-05-07 Updated Execution and Player sections](#)  
[2017-08-28 Added events of type F](#)

## Overview

This document describes the file format used to exchange information between the clients (e.g., Unity) and the backend containing the level generation (PCG), malevolence engine (ME including the model checker, planner and simulator) and the player modeling component (PM). The current document describes multiple sections of a single file although potentially these may be broken across different files.

The exchange file uses plain text structure with tab separators to facilitate readability and embedded JSON to represent structured components. Unless specifically noted, the values outside JSON (including all strings) are unquoted and types must be casted accordingly when parsing the file. Each section in the file is preceded by its section name in uppercase and followed by a blank line.

Example (excerpt, see complete [Examples](#) at the end of this document):

```
METADATA
level_id      3
player_palette {"semaphore":{"count":3},"signal":{"count":3},"painter":{"count":0}}
board_width   27
...

LAYOUT
-----
-FEEGEEEEEC-
-J--J-----J-
...

COLORS
-----
-           -
-a--d----- -
...

COMPONENTS
0      thread      13      1      S      L      {"color":0,"initial_direction":"West"...
5      semaphore   2       4      P      E      {"value":1}
6      pickup      1       1      S      L      {"type":"Conditional","color":2}
...

EXECUTION
M      7           1       1       1       {"speed":0.45}
```

```

E      7      6      1      1      {"picked":1}
E      7      1      1      1      {"payload":[2]}
...

PLAYER
(always empty, uses a separate file)

```

## Metadata Section

Describes the goal and general properties of the board.

This section uses tab separated lines with two columns, the first column is the property key (string) and the second the value. The following list enumerates valid properties and their type:

- `level_id` (int)
- `level_title` (string, human readable text)
- `goal_string` (string, human readable text, double backslash `\\` for new lines)
- `goal_struct` (json string): the requirements for completing the level, see below
- `player_palette` (json string): the components available for the player, see below
- `time_pickup_min` (int): the minimum time it takes to pick up a package by default
- `time_delivery_min` (int): the minimum time it takes to drop off a package by default
- `time_exchange_min` (int): the minimum time it takes to exchange packages by default
- `time_pickup_max` (int): the time maximum it takes to pick up a package by default
- `time_delivery_max` (int): the maximum time it takes to drop off a package by default
- `time_exchange_max` (int): the maximum time it takes to exchange packages by default
- `board_width` (int)
- `board_height` (int)
- `time_efficiency` (float): the average time for the comparison against the player's solutions

See the next subsections for more details on the json string structures for the goal and player palette.

**Example:** This is a minimal but complete example for the metadata section.

```

METADATA
level_id      3
level_title   Mutex
goal_string    Deliver 4 packages. Extra bonus for no starvation and no race conditions.
goal_struct   {"required":[{"id":1002,"type":"delivery","property":"delivered","value":4,"condition":"eq"},
,}
player_palette
{"semaphore":{"count":3},"signal":{"count":3},"painter":{"count":0},"colors":{"count":1},}
time_pickup_min      0
time_pickup_max      0
time_delivery_min     1
time_delivery_max     5
time_exchange_min     0
time_exchange_max     0
board_width          27
board_height         21

```

## Goal Representation

Used to define the goals of the level to check completion and allow for the evaluation of the player's performance (i.e., the points/stars awarded). The goal struct is a collection of lists of assertions (an assertion is a condition) of the values of the properties of several components (see components section). The goal struct is a JSON object with the following structure:

- The goal struct is a JSON object with the keys "required" and "desired". The value for each of the keys is a list of assertions.
- Each of the assertions is a JSON object with the following keys:
  - id (int): the identifier of the component from the components section in the file this assertion refers to
  - type (string): the type of the component, each with different properties described in the components section of this document
  - property (string): the property of the component to check
  - value (integer for now): the value to of the component to check
  - condition (string) the comparison to perform, one of:
    - eq: equal
    - gt: greater than
    - lt: less than
    - ne: not equal
  - thread\_id (int): the identifier of a thread, this allows to filter the number of interactions to the specified thread only (0 for any thread). Currently only implemented for the delivery property.

**Goal example 1:** The following example defines the goal of a level for the number of delivered packages to be equal to 4. Then the desired conditions (for extra bonus or to evaluate the quality of a solution) define that there have been no race conditions when delivering packages (missed equal to 0), that there was no starvation (each thread has delivered at least 1 package) and ideally it was well balanced by each thread delivering the same number of packages (equal to 2).

```
{
  "required": [
    {"id":1002,"type":"delivery","property":"delivered","value":4,"condition":"eq"},
  ],
  "desired": [
    {"id":1002,"type":"delivery","property":"missed","value":0,"condition":"eq"},
    {"id":2000,"type":"thread","property":"delivered","value":1,"condition":"gt"},
    {"id":2001,"type":"thread","property":"delivered","value":1,"condition":"gt"},
    {"id":2000,"type":"thread","property":"delivered","value":2,"condition":"eq"},
    {"id":2001,"type":"thread","property":"delivered","value":2,"condition":"eq"},
  ],
}
```

**Goal example 2:** The following goals require each vehicle to deliver exactly one package to one specific delivery point.

```
{ "desired": [], "required": [ { "condition": "eq", "property": "delivered", "id": 3001, "type": "delivery", "value": 1, "thread_id": 1001 }, { "condition": "eq", "property": "delivered", "id": 3001, "type": "delivery", "value": 1, "thread_id": 1002 } ] }
```

**Range of goals that can be defined:** notice this is a very flexible way to represent goals, which can be defined using the properties of any component in the game. For example:

- **Number of loops threads complete:** this can be defined checking the property "passed" of the semaphores.
- **Number of packages delivered:** this can be defined checking the property "delivered" of the delivery point or the threads.
- **Avoid having missed packages:** this can be defined checking the property "missed" of the delivery point or the threads.
- **Make a thread reach a certain point of the level:** this can be defined checking the property "passed" of a component in that certain point (e.g. a non-strict customs component or the old non-consuming delivery points).
- **Pickup all the packages in a level:** this can be defined checking the property "picked" of a pickup point

## Player Palette

Note: this is currently not used.

~~The player palette defines what components are available for the player to place and the properties the player will be able to select in order to introduce component in the tutorial (e.g. won't allow selecting all the different 6 colors in the beginning) and provide more expressiveness for more complex levels. Additionally, a value of -1 implies an unlimited availability of a component.~~










**Player palette example:** ~~The following example enables the player to use an unlimited number of semaphores but only 3 signals and no painters. Additionally, there is only one color available to link the signals and semaphores.~~








```
+
  "semaphore": { "count": -1 },
  "signal": { "count": 3 },
  "painter": { "count": 0 },
  "color": { "count": 1 },
+
```

## Layout Section

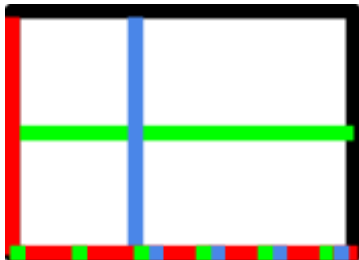
This section describes the layout of the board (roads) as is generated by the PCG backend or manually created from existing level designs.

This section uses a collection of 16 characters to encode different configurations of a subtitle (not all currently in use) as defined by the following mapping (part is a mnemonic nickname, an asterisk implies a rotation on the mnemonic, the four directions will be allowed later for each kind of tile when a 1 is present):

key	part	North	East	South	West	tile
-		0	0	0	0	
A		0	0	0	1	
B		0	0	1	0	
C	L*	0	0	1	1	
D		0	1	0	0	
E	-	0	1	0	1	
F	L*	0	1	1	0	
G	T	0	1	1	1	
H		1	0	0	0	

I	L*	1	0	0	1	
J	I	1	0	1	0	
K	T*	1	0	1	1	
L	L	1	1	0	0	
M	T*	1	1	0	1	
N	T*	1	1	1	0	
O	+	1	1	1	1	

The characters are used to encode the layout of a board in a matrix of `board_width` x `board_height` with rows separated `\n`. Future references to the board in subsequent sections will use zero-based indexes to columns (x) and rows (y) assuming the origin at the left-top corner.  
Example (12x9 tiles)



### Example:

```
LAYOUT
-----
-FEEGEEEEEC-
-J--J-----J-
-J--J-----J-
-NEEOEEEEEK-
-J--J-----J-
-J--J-----J-
-LEEMEEEEEI-
-----
```

## Colors Section

This section defines the colors that the road tiles will be painted with indicating which vehicles are allowed in each road segment.

This section uses characters that define a bitmask of the colors of a segment where 0 bits indicates it can be traversed by any vehicle. The bitmask is obtained by the index of the character minus the index of the space character. This allows the use of 6 colors. The color identifiers used below are as follows:

Color index	Color
0	No color (meaning no restrictions on roads, vehicles, packages or delivery)
1	Color 1 (green in-game, red in the example above)
2	Color 2 (yellow in-game, green in the example above) v
3	Color 3 (orange in-game, blue in the example above)
4	Color 4 (magenta in-game)
5	Color 5 (red in-game)
6	Color 6 (purple in-game)

Excerpt of the bit mask table, see ASCII table [here](#).

	Color 1	Color 2	Color 3	Color 4	Color 5	Color 6
(space)	0	0	0	0	0	0
!	1	0	0	0	0	0
"	0	1	0	0	0	0



#	1	1	0	0	0	0
\$	0	0	1	0	0	0
%	1	0	1	0	0	0
&	0	1	1	0	0	0
'	1	1	1	0	0	0
(	0	0	0	1	0	0
...						

Example:

COLORS

```

-----
-           -
-!--$----- -
-!--$----- -
-#" "&" " " " " -
-!--$----- -
-!--$----- -
-###" ' ' ' ' ' ' -
-----

```

## Directions Section

This section defines the directions of the road tiles (used for aesthetics only). The directions are computed along with the ME output so it currently required a solution to be provided. It uses 6 characters:

- A: North or up
- >: East or right
- V: South or down
- <: West or left
- X: More than one direction
- (whitespace): No travel direction

Example:

DIRECTIONS

```

>>>V           V<<<
A  V           V  A
A  V           V  A
A  >>>>>>V<<<<<<  A
A           V       A
A           V       A
A           V       A

```

```

A      V      A
A      V      A
A      V      A
A      V      A
A      V      A
A      V      A
A      V      A
A<<<<<<<<<X>>>>>>>>>A

```

## Components Section

This section describes the components (vehicles, semaphores...) that are placed on top of the board (roads) as generated by the PCG backend or created manually. Likewise this is also used to export components added by the player interactively at play time for the ME to compute an execution.

This section uses tab separated lines (one line per component) with seven columns, the following list enumerates valid columns with their purpose and their type:

```

Component id      int
Component type     string
Component x position  int (column starting with 0)
Component y position int (row starting with 0)
Placed by         [S|P] (placed initially by the system or at play time by the player)
Editable          [E|L] (whether the player can edit/move/delete the component or is
locked)
Configuration     string_json

```

The x and y positions use zero-based indices with the origin (0,0) in the left/top corner.

The following table enumerates valid component types and their configuration parameters (an associative array encoded in JSON, all lowercase); note that some of the parameters (highlighted in blue) may be used in the metadata or execution sections to define the goals or transitions in the state.

\*Components with an asterisk have been discussed but are currently NOT used, NOT fully implemented in either the ME or the client or outdated. ~~Strikeout~~ properties are currently NOT considered.

type	description	configuration
thread	A spawn point for threads (a vehicle in the current metaphor) that traverse the board (travels on roads).	<ul style="list-style-type: none"> <li>color (int): what road or color is it allowed to go through (note that the painter component can override this property)</li> <li>initial_direction (North East South West): meaning one of North, East, South or West</li> <li>capacity (int): how many packages it can hold, -1 for unlimited</li> <li>delay (int): how long will it delay before spawning the first thread</li> <li>time_pickup_min (int): thread</li> </ul>

		<p>specific values, optional, if not specified, it will use the default specified in the metadata section</p> <ul style="list-style-type: none"> <li>time_delivery_min (int): idem</li> <li>time_exchange_min (int): idem</li> <li>time_pickup_max (int): idem</li> <li>time_delivery_max (int): idem</li> <li>time_exchange_max (int): idem</li> <li>picked (int): the number of packages that have been picked</li> <li>delivered (int): number of packages successfully delivered</li> <li>missed (int): number of packages missed</li> </ul>
spawnner*	A spawn point for threads (vehicles in the current metaphor) that traverse the board (travels on roads).	<ul style="list-style-type: none"> <li>color (int): what road or color is it allowed to go through (note that the painter component can override this property)</li> <li>initial_direction (North East South West): meaning one of North, East, South or West</li> <li>capacity (int): how many packages it can hold, -1 for unlimited</li> <li>delay (int): how long will it delay before spawning the first thread</li> <li>number (int): how many vehicles will be spawned from this point</li> <li>frequency (int): how long will it delay between vehicle spawns</li> </ul>
signal	Linked to a semaphore (light) or conditional (arrow) to switch its state to active (1 in case of semaphores) or toggle between the conditional's directions	<ul style="list-style-type: none"> <li>link (int): the semaphore id that will be switched from inactive (0, red) to active (1, green) or the arrow that will be toggled</li> <li>passed (int): number of vehicles or threads that passed by this point</li> </ul>
semaphore	A light. <b>This is linkable to a signal.</b>	<ul style="list-style-type: none"> <li>value (int): the initial state of the semaphore where 0 is red or inactive and 1 is green or active.</li> <li>passed (int): number of vehicles or threads that passed by this point</li> </ul>
pickup	Original location where a package will be spawned	<ul style="list-style-type: none"> <li>type (Unconditional Conditional Limited)</li> </ul>

		<p>: the kind of package that will be spawned</p> <ul style="list-style-type: none"> <li>• color (int): the color of the package (used internally for linking)</li> <li>• passed (int): number of vehicles or threads that passed by this point</li> <li>• picked (int): the number of packages that have been picked</li> </ul>
delivery	A delivery location	<ul style="list-style-type: none"> <li>• <del>color (int): the color of the delivery point, 0 for no color (i.e., it will accept any package)</del></li> <li>• accepted_types (array of string with values possible: Unconditional Conditional Limited): the types of packages that will be accepted, empty list for any type of package (and will then check the color/kind)</li> <li>• accepted_colors (array of int): the colors or "kinds" of packages accepted, empty list for all colors (and will check the type instead)</li> <li>• consumer (int): how many packages are consumed from the vehicle (1 for cars, &gt;1 for trucks, -1 for all the packages possible) or 0 if the package is not removed/returned to the vehicle after passing)</li> <li>• <del>strict (int): whether (1) or not (0) a delivery point will indefinitely stop a car without the appropriate package's color</del></li> <li>• passed (int): number of vehicles or threads that passed by this point</li> <li>• delivered (int): number of packages successfully delivered</li> <li>• missed (int): number of packages missed</li> <li>• denominator (int): number to show as the denominator in the UI. Missing property defaults to -1 and indicates NO denominator shown.</li> </ul>
customs*	Sits in any road tile. Checks that vehicles have the corresponding	<ul style="list-style-type: none"> <li>• color (int): the color of the customs point, 0 for no color (so it can</li> </ul>

	<p>package (of the color of the customs) and prohibits passage to vehicles that don't have it. Used to implement key-and-lock puzzles. Note that this is a specialized version of a non-consuming delivery location. Note: This can be reimplemented using diverters (as seen in Level 3).</p>	<p>check that a vehicle has at least a package)</p> <ul style="list-style-type: none"> <li>• consumer (int): always 0 for customs (will not consume packages)</li> <li>• strict (int): always 1 for customs (will always check packages)</li> <li>• passed (int): number of vehicles or threads that passed by this point</li> </ul>
diverter	<p>Sits always in an intersection. Checks that vehicles have the corresponding package (of the color/kind/type specified) and diverts them to one road or the other. Used to implement conditionals (without the painter). Note that this is a specialized version of the intersection below. Implementation note: The diverter will divert based on the first matching package. It will iterate over each package and each the lists of type/color in North, East, South, West order. Otherwise it will not change direction.</p>	<ul style="list-style-type: none"> <li>• <del>color (int): the color of the diverter, 0 for no color (so it can check that a vehicle has at least a package)</del></li> <li>• <del>direction_condition ({North East South West}): the direction towards which vehicles that do satisfy the condition are diverted to.</del></li> <li>• <del>direction_default ({North East South West}): the direction towards which vehicles that don't satisfy the condition are diverted to.</del></li> <li>• directions_types (array of arrays of strings): an array with 4 elements (one for each of the 4 directions in this order: West, South, East, North), where each element is a list of the package types diverted toward that direction (Unconditional Conditional Limited Empty), an empty list implies nothing can go to that direction and the package type "Empty" indicates vehicles without a package.</li> <li>• directions_colors (array of arrays of ints): an array with 4 elements (one for each of the 4 directions in this order: West, South, East, North), where each element is an empty list or a list of the package colors/kinds diverted toward that direction and -1 indicates vehicles without a package. An empty list implies nothing can go to that direction. When not empty,</li> </ul>

		<p>supersedes the direction_types.</p> <ul style="list-style-type: none"> <li>passed (int): number of vehicles or threads that passed by this point</li> </ul>
exchange	Each one of a pair of two exchange points	<ul style="list-style-type: none"> <li>link (int): the id of the exchange point that this is connected to</li> <li>color (int): the color of the exchange point, 0 for no color</li> <li>strict (int): whether (1) or not (0) an exchange point will indefinitely stop a car without the appropriate package's color</li> <li>stop (int): whether (1) or not (0) an exchange point will stop a vehicle until the a vehicle in the other end (link) is present</li> <li>passed (int): number of vehicles that have passed through</li> <li>exchanged (int): number of times a successful exchange has happened</li> </ul>
painter*	Paints vehicles based on the color of the package they are carrying	<ul style="list-style-type: none"> <li>passed (int): number of vehicles that have passed through</li> <li>painted (array[int]): number of times a vehicle has been painted in, where the color is the array index</li> </ul>
conditional	An arrow, also called direction switch, that diverts traffic. <b>This is linkable to a signal.</b>	<ul style="list-style-type: none"> <li>directions (array[North East South West]): an array of the available directions that this arrow switches between</li> <li>current (int): index of the current direction for the arrow</li> <li>passed (int): number of vehicles that have passed through</li> </ul>
arrow*	Static arrow that diverts traffic	<ul style="list-style-type: none"> <li>direction (North East South West)</li> <li>passed (int): number of vehicles that have passed through</li> </ul>
intersection*		<ul style="list-style-type: none"> <li>directions (array[North East South West]): the direction towards which vehicles with a certain color turn at an intersection (T or X). The array of</li> </ul>

		length equal to number of colors (currently 6) plus 1 for vehicles without color (index 0), vehicles will read the array index by their color to choose a direction <ul style="list-style-type: none"> <li>passed (int): number of vehicles that have passed through</li> </ul>
line*	A "finish line" that counts things?	<ul style="list-style-type: none"> <li>passed (int): number of vehicles that have passed through</li> </ul>

### Example:

#### COMPONENTS

```

0      thread      13      1      S      L      {"color":0,"initial_direction":"West"...
1      thread      12      1      S      L      {"color":1,"initial_direction":"West"...
2      signal      1      1      P      E      {"link":5}
3      signal      4      1      P      E      {"link":4}
4      semaphore   2      1      P      E      {"value":0}
5      semaphore   2      4      P      E      {"value":1}
6      pickup      1      1      S      L      {"type":"Conditional","color":2}
...
100     thread 1      1      S      L
{"color":0,"initial_direction":"North","capacity":1,"delay":0,"picked":0,"delivered":0,"missed":0}
101     spawner    13      1      S      L
{"color":0,"initial_direction":"West","capacity":1,"delay":1,"number":2,"frequency":1,"picked":0,"delivered":0,"missed":0}
102     delivery   13      13      S      L
{"color":0,"consumer":-1,"strict":0,"passed":0,"delivered":0,"missed":0}
103     delivery   13      13      S      L
{"color":1,"consumer":0,"strict":1,"passed":0,"delivered":0,"missed":0}
104     exchange   10      13      S      L
{"link":1005,"color":0,"strict":0,"stop":0,"passed":0,"exchanged":0}
105     exchange   16      13      S      L
{"link":1004,"color":0,"strict":0,"stop":0,"passed":0,"exchanged":0}
106     painter    16      16      S      L      {"passed":0,painted:[0,0,0,0,0,0]}
107     conditional 16      19      S      L
{"directions":["East","West"],"current":0,"passed":0}
108     intersection 19      19      S      L
{"directions":["North","South","East","South","East","East","East"]}

```

### Example 1:

Have a delivery point that accepts only 2 kinds of packages:

```

101     pickup      1      1      S      L      {"type":"Conditional","color":1}
102     pickup      1      2      S      L      {"type":"Conditional","color":2}
103     pickup      1      3      S      L      {"type":"Conditional","color":3}
104     pickup      1      4      S      L      {"type":"Conditional","color":4}
201     delivery    13      13      S      L
{"consumer":-1,"passed":0,"delivered":0,"missed":0,"accepted_colors":[1,3]}

```

### Example 2:

Have a diverter that diverts vehicles with packages of kind 1 to the left, kind 2 to the right and the rest to the top (remember the order is North, East, South, West:

```
101 pickup 1 1 S L {"type":"Conditional","color":1}
102 pickup 1 2 S L {"type":"Conditional","color":2}
103 pickup 1 3 S L {"type":"Conditional","color":3}
104 pickup 1 4 S L {"type":"Conditional","color":4}
201 diverter 13 13 S L
{"passed":0,"directions_types":[],"directions_colors":[[3,4],[2],[],[1]]}
```

## Execution Section

This section describes an execution behavior computed by the ME as needs to be visualized by the client and the execution results.

This section uses tab separated lines (one line per event) with the following columns:

- Event type (M, S, E or D or F): Type of event (see below)
- Time step (int): The current time
- Component id (int)
- Component x position (int)
- Component y position (int)
- Status (string\_json): A json object representing a map of key-value pairs.

The event types are only:

- Type M for each unit, with its position at each time step where there is either an interaction with a component, a turn, or a start/stop event. This can be used to interpolate unit positions over time.
- Type S for each unit and with speed as the only property set in the status
  - The speed is expressed in tiles over units of time. The maximum speed currently is 1, which indicates a vehicle will traverse the distance between the centers of two tiles in one unit of time (a Unity second or whatever multiplier is used for maximum speed).
- Type E: some event happened, the json includes any of the properties defined in the components section that have changed at this point in time (so audio/visual feedback can be given to the player).
- Type D: a summary for deliveries and exchanges (redundant), see example below
- Type E at the very end includes information about the end-state of the simulation, includes:
  - final\_condition is a bitmask with the following indications:
    - RESULT\_DEADEND = 1
    - RESULT\_PARTIAL = 2 (only the required goals are satisfied)
    - RESULT\_PROBLEMATIC = 4
    - RESULT\_SUCCESSFUL = 8 (includes all desired and required goals)
    - RESULT\_PROBLEMATIC\_MISSED = 16
    - RESULT\_PROBLEMATIC\_STARVATION = 32
    - RESULT\_PROBLEMATIC\_INFINITE\_LOOP = 64\*



- RESULT\_PROBLEMATIC\_LOOPY\_HASH = 128\*
- RESULT\_PROBLEMATIC\_REGRESSION = 256\*
- RESULT\_PROBLEMATIC\_WRONG\_PATH = 512\*
- In other words: 2, 8 and 10 are good. 20, 36, 22 and 38 are bad, any other is probably bad as well. 0 means it didn't finish, either not enough budget or it went into a loop (shouldn't happen). \*The last 4 are relatively new and I'm still making sure they work fine.
- goal\_state an array of boolean values corresponding to the goals of the level and indicating whether each has been satisfied or not.
- goal\_descriptions an array of strings with a code and short description corresponding to the goals of the level the codes are:
  - s01 This required goal was completed successfully.
  - s02 This optional goal was completed successfully.
  - e00 There was no delivery by a specific arrow to a specific delivery point.
  - e12 An arrow didn't deliver the exact number of packages.
  - e13 An arrow delivered more packages than it was supposed.
  - e14 An arrow did not deliver all the packages it was supposed.
  - e15 An arrow delivered the wrong number of packages.
  - e22 An arrow didn't deliver the exact number of packages.
  - e23 An arrow delivered more packages than it was supposed.
  - e24 An arrow did not deliver all the packages it was supposed.
  - e25 An arrow delivered the wrong number of packages.
  - e36 Another required goal was not met.
  - e37 Another desired goal was not met.
  - e48 Couldn't evaluate the goal.
- time\_efficiency a float indicating whether the difference between the average time required to complete the current solution and the reference solution. If time is  $\leq 0$  it's a good solution (and the player should receive an extra star) otherwise if it's  $> 0$  it's a worse solution (and the player should not receive a star).
- Type F: when a unit goes over a delivery point but there is no delivery.

### Example 1:

M	0	1001	6	3	{"speed":1.0}
M	16	1001	7	14	{"speed":0.14285715}
M	23	1001	5	3	{"speed":1.0}
M	53	1001	7	3	{"speed":0.5}
M	55	1001	8	3	{"speed":1.0}
M	56	1001	9	9	{"speed":0.2413793}
M	85	1001	9	12	{"speed":1.0}
M	91	1001	7	14	{"speed":0.14285715}
M	98	1001	5	3	{"speed":1.0}
M	128	1001	6	3	{"speed":0.0}
M	0	1002	12	3	{"speed":1.0}
M	1	1002	12	3	{"speed":0.0}
M	14	1002	10	3	{"speed":1.0}
M	88	1002	12	3	{"speed":0.0}

M	89	1002	10	3	{"speed":1.0}
M	128	1002	9	13	{"speed":0.0}
E	128	0	0	0	{"final_condition":36}

This example illustrates: At time 0 the vehicle 1001 starts moving at speed 1.0 until time 16 when changes speed to 0.1428... until time 23 where changes back to 1.0, etc.

At the same time, vehicle 1002 starts moving at 1.0 but at time 1 it stops until time 14 where it resumes at 1.0 and stops again at time 89.

Both vehicles stop at time 128; at the end of the simulation. At the end there has been some starvation condition indicated by code 36.

### Example 2:

S	0	1005	14	25	{"speed":1.0}
M	0	1005	13	25	
S	18	1005	19	34	{"speed":0.0}
M	18	1005	19	34	
S	76	1005	20	34	{"speed":1.0}
M	76	1005	19	34	
S	77	1005	20	34	{"speed":0.0}
M	77	1005	20	34	
...					
M	6	1005	19	25	
M	15	1005	19	34	
M	77	1005	20	34	
...					

This example illustrates: At time 0 the vehicle 1005 starts moving at speed 1.0. At time 6 there is a corner. At time 15 there is another corner. At time 18 it stops. At time 76 it resumes at speed 1.0. At time 77 it stops at a corner.

### Example 3:

Vehicle 1001 picks package at point 2001 at time 94

E	94	2001	9	9	{"picked":3}
E	94	1001	9	9	{"payload":[2001]}

Then vehicle 1001 drops the package at 3001 which completes one of the "optional" goals

E	98	3001	9	12	{"delivered":3}
E	98	3001	9	12	{"passed":3}
E	98	1001	9	13	{"payload":[]}
E	98	1001	9	13	{"delivered":2}
E	98	0	0	0	{"goals_completed":1}

This is some vehicle pressing the button 3002 which changes the semaphore 4002 to "green" and immediately afterward a vehicle that was waiting passes and the semaphore goes back to "red"

E	100	3002	9	14	{"passed":3}
E	100	4002	10	3	{"value":1}
E	101	4002	10	3	{"value":0}

There are a couple examples about events in

[https://github.com/josepvalls/ParallelProg/blob/master/levels/play\\_out\\_normal\\_solved.txt](https://github.com/josepvalls/ParallelProg/blob/master/levels/play_out_normal_solved.txt)

```
E 14 3001 9 12 {"delivered":1}
E 14 3001 9 12 {"passed":1}
E 14 1001 9 12 {"payload":[]}
E 14 1001 9 12 {"delivered":1}
D 14 1001 9 12 {"delivered_to":3001,"missed_items":[2001],"delivered_items":[]}
```

The D event type example which summarizes the previous 4 events.

Example 4:

Thread 1002 picks up package 2001

```
E 21 1002 12 5 {"payload":[2001]}
```

And it uses exchange points 2010 and 2011 to exchange the package with thread 1001

```
E 24 2001 0 3 {"passed":1}
E 24 2010 0 6 {"exchanged":1}
E 24 2011 6 6 {"exchanged":1}
E 24 2011 6 6 {"value":0}
E 24 1001 6 6 {"payload":[2001]}
E 24 1002 0 3 {"payload":[]}
D 24 1001 6 6 {"exchange_between_b":1002,"exchange_between_a":1001}
```

The D event type, also redundant has the IDs of the two threads that exchange (in arbitrary order, note that they both can have packages).

## Player Section

This section is currently not used and will be blank. Instead, the player trace is recorded on a separate file that has references to the different runs of the ME. The file is located in the [persistentDataPath](#) and is named with the prefix "Session".

## File Validation

Writing these files is NOT trivial so I found it useful to build a little validator. Not all the errors are caught yet. The additional verbose argument will parse a file and export the parsed interpretation for comparison. Depends on gson and jdom.

```
OSX:dist josepvalls$ java -cp PCGMC4PP.jar:lib/gson-2.6.2.jar support.validate
Usage: java support.validate file.txt [Verbose]
```

```
OSX:dist josepvalls$ java -cp PCGMC4PP.jar:lib/gson-2.6.2.jar support.validate
../../level-3-prototype-errors.txt
Loading: ../../level-3-prototype-errors.txt
Tile bitmask mismatch on line 22, column 9 expected J found O.
Empty line 47 ignored when parsing file
Invalid initial_direction component property EAST in line 49
Cannot create component on line 54 error: java.lang.NumberFormatException: For input string:
"North"
EXECUTION Section ignored when parsing file.
```

*Note, on Windows, use PCGMC4PP.jar;lib/gson-2.6.2.jar instead of PCGMC4PP.jar:lib/gson-2.6.2.jar (the colon is replaced by semicolon).*

## ME command line interface

The command line interface uses the class support.ME in the PCGMC4PP.jar. It is documented in the [Play Button Activity](#) in the [Workflow and Activities](#) document.

## Comments and Revisions

### 2016-04-12 Removed multiple of 3

Removed the multiple of 3 restriction.

Suggestion for a future version of the file format:

It may be useful and more expressive to encode the layout and colors sections in the same format of the components section and encode the colors along with any other attributes for the road pieces that we may need in the future.

The current format would be then redundant but could be used to show the layout of a level at a glance. Take the following excerpt:

```
LAYOUT
-----
-FEEGEEEEEC-
-J--J-----J-
-NEEOEEEEEK-

COLORS
-----
-@@@@@@@@@@-
-A--D-----@-
-CBBFBBBBB@-
```

Example using a list of colors:

```
COMPONENTS
1000  F      1      1      S      L      {"colors": [0]}
1001  E      2      1      S      L      {"colors": [0]}
1002  E      3      1      S      L      {"colors": [0]}
1002  G      4      1      S      L      {"colors": [0]}
...
1010  J      1      2      S      L      {"colors": [1]}
1011  J      4      2      S      L      {"colors": [3]}
1012  J     10      2      S      L      {"colors": [3]}
1013  N      1      3      S      L      {"colors": [1,2]}
1014  E      2      3      S      L      {"colors": [2]}
1015  E      3      3      S      L      {"colors": [2]}
1016  O      4      3      S      L      {"colors": [2,3]}
...
```

Alternative example using a bitmask of colors:

COMPONENTS						
1000	F	1	1	S	L	{"colors":0}
1001	E	2	1	S	L	{"colors":0}
1002	E	3	1	S	L	{"colors":0}
1002	G	4	1	S	L	{"colors":0}
...						
1010	J	1	2	S	L	{"colors":1}
1011	J	4	2	S	L	{"colors":4}
1012	J	10	2	S	L	{"colors":4}
1013	N	1	3	S	L	{"colors":3}
1014	E	2	3	S	L	{"colors":2}
1015	E	3	3	S	L	{"colors":2}
1016	O	4	3	S	L	{"colors":6}
...						

## 2016-04-22 Workflow

The workflow has been moved to a dedicated document per Anna's request.

[Workflow and Activities.](#)

## 2016-04-25 Strict JSON

Updated examples to use strict JSON and match the documentation.

## 2016-04-29 Goal structure details

As per Joe Jalbert's request, expanded the documentation on the goal structure in the metadata section.

I also updated the examples. These were based off "Level 3 Mutex.ai" by Evan Freed but the colors of the roads caused unexpected results.

## 2016-05-03 Player palette

A count of -1 for a component can be used to enable unlimited access to a component. For the time being player palette counts will be 0 or -1. Updated the times for pickup/delivery/exchange to be ranges and added thread specific ranges (updated metadata and components/thread sections).

## 2016-05-06 Goal structure details and speed

Added some structure to the metadata section and examples of the goals. Added the computed speed to the execution section.

## 2016-05-10 Color updated

In order to properly handle the 6 colors for the dining philosopher there is a small change on the file format for the colors section. Instead of starting from the '@' sign we start from the ' ' (space character).

## 2016-05-17 Added customs, updated examples

Added the customs component to the components section. Added an \* to the components that are not considered in the current implementation (including the painter).

Updated all 3 comprehensive examples to the current specification of the document.

Also added a ME interface link for convenience.

## 2016-05-20 Several Changes

Major:

Colors section is completely ignored for now. Instead of road colors, intersection components will be used.

Added the diverter component to replace the painter and complement the customs.

Minor:

Clarified definitions for the intersection, customs and diverter components.

Updated explanation of the speed in the execution section so the maximum is 1. The speed will be normalized to 1 before exporting from ME so it can be consistently multiplied for visualization.

## 2016-05-24 Added arrow Component

Added arrow component.

Minor:

Marked customs component as not in the current prototype.

Removed stopped property (currently not updated and not in use anyway).

## 2016-06-03 Rewrote execution section

Rewrote with the only properties we currently use and documented the final condition codes in case we want feedback in the client.

## 2016-07-15 Added intermediate points in execution section

Updated events so there is M that reports all intermediate positions at a time step in order to interpolate the position of the units. Currently this is not sorted and could be cleaned up for intersections without a turn. Also it may report multiple events at a single time step if there is a component at an intersection or a stop/start event at either a component or an intersection.

Documentation for execution section updated and added Example 2.

I took a look at Jal's code and I updated it so the old event M is renamed to S (although it's all the same right now). I see he is already sorting it there.

## 2016-09-06 Colors, events, simulation

Colors are back in, execution exports all events, simulation command line option, see [https://docs.google.com/document/d/1sZ48ufEntpJYwj24I980WppXXXWoLHofE\\_lwFU8s-Wc/edit](https://docs.google.com/document/d/1sZ48ufEntpJYwj24I980WppXXXWoLHofE_lwFU8s-Wc/edit)

## 2016-09-23 Several changes

New properties for delivery: `accepted_types` (array of string) and `accepted_colors` (array of int) that supercede `color` (int) and `strict` (int).

New properties for diverter: `directions_types` (array for 4 arrays of strings) and `directions_colors` (array of 4 arrays of ints) that supercede the `color` (int) and `directions`.

New section `directions` with information for the Unity client to display directions on the tracks.

Events in execution are now exported at the time the event is complete before moving (so the vehicles delivering are still at the delivery point).

Added type of event D with a summary for each delivery.

Document review, some other minor changes.

## 2016-10-01 Event for exchange points

Added type of event D with a summary for each exchange.

## 2016-10-13 Added -1 to diverters

Can be used to divert traffic that doesn't carry any package.

## 2016-10-19 Added "Empty" to diverters

Can be used to divert traffic that doesn't carry any package.

## 2016-10-19 Added optional "thread\_id to goal\_struct

This enables the goal struct to specify conditions like deliver X to Y with Z such as in smokers so each thread delivers to each delivery point. Note, currently this only works for delivery.

Example below:

```
goal_struct
{"desired":[], "required":[{"condition":"eq", "property":"delivered", "id":3001, "type":"delivery", "value":1, "thread_id":1001}, {"condition":"eq", "property":"delivered", "id":3001, "type":"delivery", "value":1, "thread_id":1002}]}
```

## 2017-01-16 Added denominator for delivery component

Added: denominator (int) to delivery

## 2017-02-20 Deprecated components

We decided not to use arrows and intersections anymore.

## 2017-05-07 Updated Execution and Player sections

Updated documentation to reflect a change in the events of type D in the execution section.

From:

```
D    14    1001    9    12    {"delivered_to":3001,"missed":[2001],"delivered":[]}
```

To:

```
D    14    1001    9    12    {"delivered_to":3001,"missed_items":[2001],"delivered_items":[]}
```

Added remark on the Player section, not used anymore and instead saved on a separate file.

Some other minor updates.

## 2017-08-28 Added events of type F

Added the third line as seen in this example:

```
E      3      3001    2      0      {"passed":1}
E      3      1001    2      0      {"payload":[2001]}
F      3      3001    2      0
```

## 2017-11-21 Added info for end screen

Added time\_efficiency in both metadata and execution sections and additional information to event type F.