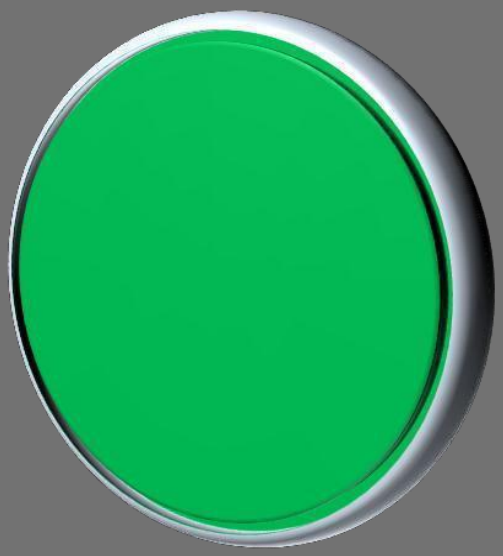
2020

TARTARUS

Empowering Agents

# Tartarus
# -Users' Manual

***Tackle the Real World and***
***Say Bye to Simulations!***



Robotics Lab.,
Dept. of Computer Science & Engineering, Indian Institute of Technology Guwahati.

## Foreword

Embedding intelligence on real decentralized and distributed systems calls for an agent platform that features tools for programming intelligence. It was thus thought that the creation of such a platform using an AI language would best suit the purpose. The first attempt to realize this, at the Robotics Lab. of the Dept. of Computer Science and Engineering, Indian Institute of Technology Guwahati, culminated in the creation of ***Typhon***, a mobile agent platform. ***Typhon*** was built over the Chimera (static) agent system ported with LPA Prolog[1]. While ***Typhon*** was used to realize and demonstrate a variety of decentralized applications, it had several issues that could not be sorted out due to its proprietary compiler. An alternative became mandatory. ***Typhon*** was thus revamped and re-coded using SWI-Prolog and bundled with better and enhanced features and christened Tartarus. This new platform supports a host of features and unlike its predecessor, can also be used in conjunction with embedded systems such as the Raspberry Pi and Lego NXT robots. Contrary to ***Typhon***, agents in Tartarus agents run as threads and thus provide for better concurrency. Agents in Tartarus can also carry more payload than its predecessor and can be run on both Windows, Linux (Ubuntu) and Raspbian operating systems.

The use of Tartarus running on Raspberry Pi boards, nicknamed ***AgPi***, increases the dimensionality of its applications. Systems programmed using Tartarus can now be organized and connected to form either an **Internet of Things (IoT)** or a **Cyber Physical System (CPS)**. The current version can be used for a plethora of intelligent applications that encompass the areas of networking, robotics, computing architectures, big data and the like. Since the computing requirements are minimal, it is hoped that Tartarus will encourage researchers to graduate from mere simulations to emulations of the systems they endeavour to build and come up with real workable decentralized and distributed algorithms and their associated real-world applications.

This manual assumes that the reader is aware of the use of SWI Prolog[2] and has the same installed in the system on which s/he intends to run Tartarus.

Cheers! And let's say bye to simulations.

The Tartarus Team

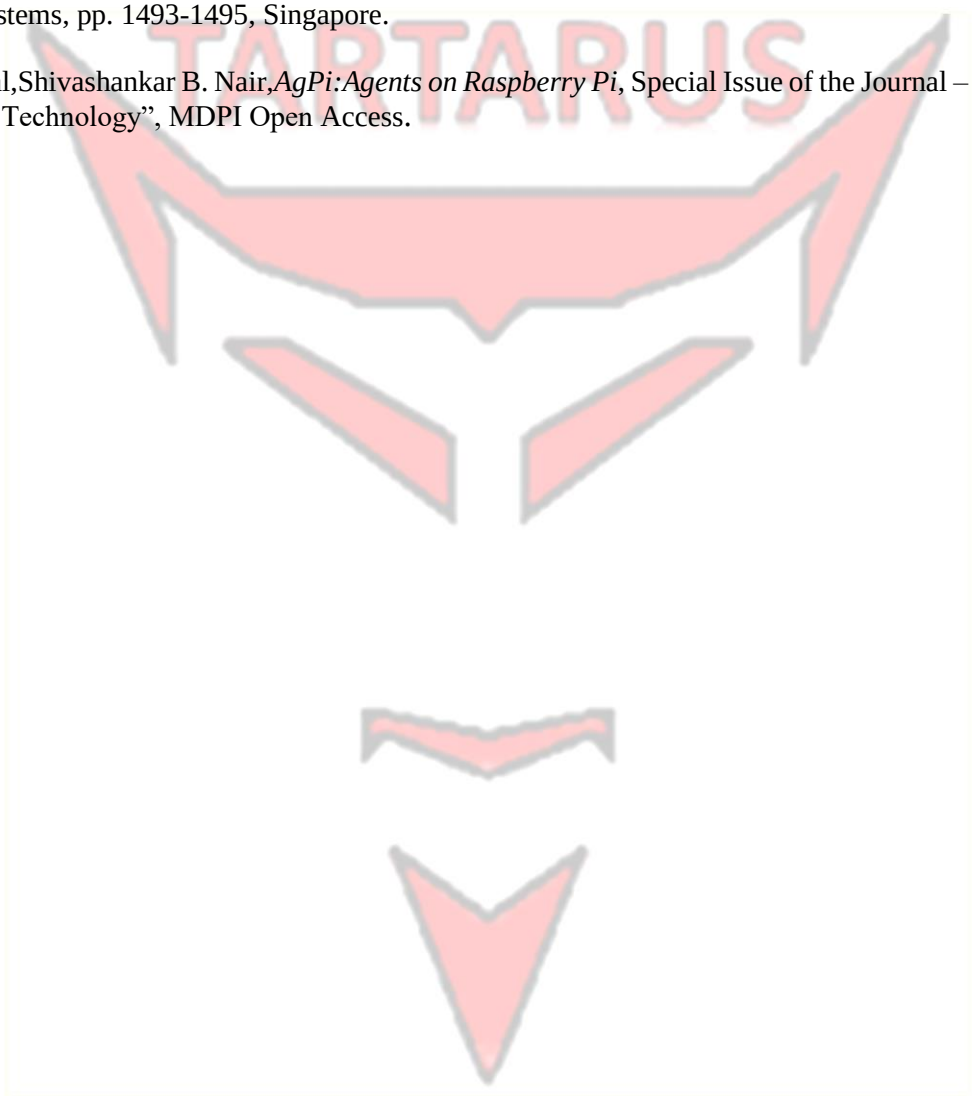
The names of those who prominently contributed to the making of Tartarus:
Mentor: Shivashankar B. Nair
Current: Divya Kulkarni, Suraj Pandey, Menaxi J Bagchi, Tushar Semwal
Past: Vivek Singh, Manoj Bode

[1] www.lpa.uk
[2] www.swi-prolog.com

Authors using this software and desirous of publishing their work are obliged to cite the following papers:

Tushar Semwal, Nikhil S., Shashi Shekhar Jha, Shivashankar B. Nair, TARTARUS*: A Multi Agent Platform for Bridging the gap between Cyber and Physical Systems*, Proceedings of the AAMAS-2016 Autonomous Agents and Multi Agent Systems Conference, International Foundation for Autonomous Agents and Multiagent Systems, pp. 1493-1495, Singapore.

Tushar Semwal,Shivashankar B. Nair,*AgPi:Agents on Raspberry Pi*, Special Issue of the Journal – Electronics: "Raspberry Pi Technology", MDPI Open Access.



# Chapter 1
# Getting Started

Since Tartarus runs over SWI-Prolog the first step is to ensure that you have the latter installed in your system. SWI-Prolog can be downloaded from www.swi-prolog.org. If you wish agents to express using speech you will need to install ***espeak*** too.

**Note**: All through this manual, the terms Tartarus and *Platform* have been used interchangeably.

## 1.1 Installing and Loading the Tartarus platform

### (i) For Windows OS

Follow the steps below to install Tartarus on your Windows based machine:

1. Download and install the SWI-Prolog on your system. SWI-Prolog can be downloaded from http://www.swi-prolog.org/download/stable.
2. Run the Tartarus installer file which can be downloaded from the link: http://www.iitg.ernet.in/cse/robotics/?page_id=2302
3. There are two ways to load the Tartarus platform on SWI-Prolog:
   a. If you are new to SWI-Prolog, just double click on the Tartarus icon. This will open an SWI-Prolog instantiation with Tartarus pre-loaded onto it. You can now go ahead to use Tartarus.
   b. For others who use SWI-Prolog, just **consult (load)** the **platform.pl** file resident in the Tartarus directory.

### (ii) For Debian based OS (Ubuntu, Raspbian Jessie)

1. Install SWI-Prolog using the command:
**sudo apt-get install swipl**
2. Download the tar ball from the link: http://www.iitg.ernet.in/cse/robotics/?page_id=2302
3. Untar the tar ball to any desired location in your system. Use the following command to untar the tar ball: tar -xvf <tar ball> -C <desired location>
4. In a new command line terminal, type: **swipl**
5. This will start a SWI-Prolog session. Now to load the Tartarus platform, enter the command:
**consult('/home/<Installed Tartarus directory>/platform.pl').**

### (iii) For espeak

**Windows:**
Please execute the windows installer provisioned at "http://espeak.sourceforge.net/download.html" to install espeak in C drive.

**Ubuntu/Raspbian:**
Use the command "sudo apt-get install espeak"

## 1.2 AgPi - An interface for Tartarus and Raspberry Pi

WiringPi is a C library which provides APIs to access the GPIO pins and other peripherals of Raspberry Pi. Through this wrapper, Tartarus agents can access the different peripherals of a Raspberry Pi, such as GPIO pins, PWM, I2C, etc. The steps to install and load the Wiring-Pi interface can be found in the github repository of Tartarus - https://github.com/roboticslab-cseiitg/ProjectTartarus/wiki/AgPi:-Agents-on-Raspberry-Pi. It also contains few of the sample programs of AgPi.

**Note: The github link for Tartarus is given below:**
https://github.com/roboticslab-cseiitg/ProjectTartarus

# Chapter 2
# Tartarus Built-in Predicates

## 2.1 Introduction

This chapter describes the various predicates supported by Tartarus. These are generic predicates which can be used when running the Tartarus platforms on Windows, Linux or the Raspberry Pi. For each predicate the arity is represented by a number /n after the predicate name in the form -

**predicate_name/n**

The description of each of these arguments is provided next as **<DATA_TYPE, X>** where X indicates the manner in which the data passed to the predicate needs to appear and can assume three symbols viz. ?, + and -. Their meanings are conveyed below -

'?' indicates that the value can be passed to the predicate; if not, the same will be assigned after a call to the predicate.

'+' indicates that the value needs to be passed to predicate (**mandatory**).

'- ' indicates that value will be assigned by a call to the predicate.

The predicates listed below have been bunched based on their approximate functionalities.

Tartarus provides a range of predicates to be used in conjunction with agent programming. They have been categorized below for convenience.

| Platform predicates | Agent predicates | Log Server predicates | Miscellaneous predicates | RaspberryPi predicates |
|---|---|---|---|---|
| start_tartarus/3 | create_mobile_agent/4 | set_log_server/2 | get_new_name/2 | ledOn/1 |
| close_tartarus/0 | create_mobile_agent/3<br>create_static_agent/4 | send_log/2 | get_new_name_alpha/1 | ledOff/1 |
| reset_tartarus/0 | purge_agent/1 | | tts/1 | mpu6050/2 |
| print_tartarus_status/0 | execute_agent/3 | | tts_on/0 | mpu6050/1 |
| assert_file_to_tartarus/1 | execute_agent /4 | | tts_off/0 | servo/4 |
| get_tartarus_details/2 | add_payload/2 | | subt_off/0 | |
| set_token/1 | remove_payload/2 | | subt_on/0 | |
| load_tartarus_state/1 | add_token/2<br>clone_agent/3 | | in_hop_time/2,<br>in_hop_time/5 | |
| save_tartarus_state/1 | move_agent/2 | | out_hop_time/2,<br>out_hop_time/5 | |
| | save_agent/2 | | in_hop_time_stat/3 | |
| | isexist_agent/3 | | out_hop_time_stat/3 | |
| | list_ agents/0 | | overall_hop_time_stat/3 | |
| | post_agent/3 | | in_hop_times_details/1 | |
| | | | out_hop_times_details/1 | |
| | | | in_hop_times_info/1 | |
| | | | out_hop_times_info/1 | |

Tartarus also provides special predicates to control Lego NXT Mindstorms which are presented and explained in a separate manual provided with along with the software. When used in conjunction with Raspberry Pi, the Tartarus equivalent AgPi supports the various predicates to access the hardware on- board. These predicates and their functionalities have also been presented as a separate manual.

## 2.2 Predicates pertaining to the Tartarus platform

### (i) start_tartarus/3

start_tartarus(IP, Port,Token). % Starts a Tartarus platform or instance. %

**Note:-** The IP should be given either as "localhost" or the IP address of the system in the dotted decimal format. For e.g., '192.168.2.3'.

**Arity types:**
IP: <atom + >
Port: <integer +>
Token:<integer +>

**Description:**
This predicate is used for starting a Tartarus platform. It initializes all global variables and stack and starts the platform on the given IP address and Port number. It also sets the platform token. The IP address is the address of the system on the network. This predicate must be executed before creating any static or mobile agents. One can start multiple platforms on the same IP provided the Port numbers are different.

**Example:**
?- start_tartarus (localhost,6000,1).

If the first predicate above is successful, the SWI-Prolog terminal will display the following message:

**********
**Welcome to Tartarus Multi-Agent Platform**
**(Ver.2020.2.2, 12th October 2020)**

**Copyrights 2017: Robotics Lab.**
**Dept. of Computer Science & Engg.**
**Indian Institute of Technology Guwahati**

**(For any query contact: tartarus.iitg@gmail.com)**
**********

==========
**Tartarus Platform launched!**
**IP = localhost, Port = 11111**
==========

**CAUTION: If you intend to run several platforms in the same machine ensure to use different ports else the system will throw an error.**

### (ii) close_tartarus/0

close_tartarus. % Shuts down a Tartarus platform. %

**Arity Types:**
NA

**Description:**
When run within a platform, this predicate terminates all the agents and removes all code running on the platform. It closes all open sockets before closing the platform. After closing platform, you can once again start a fresh platform on different IP and/or Port.

**Example:**
?- close_tartarus.
After successful closing of platform, the SWI-Prolog terminal will display following message:

Platform has been closed.

### (iii) reset_tartarus/0

reset_tartarus. % Resets the existing Tartarus platform within which the predicate is executed. %

**Arity Types:**
NA

**Description:**
This predicate is used for resetting the platform to the state when it was first started. Unlike *close*_tartarus/0 which merely closes the platform, this predicate closes the platform and restarts it afresh using the same IP and Port.

**Example**:
?- reset_tartarus.

After successful reset, the SWI-Prolog terminal will display the message:

-------restarting platform in 2 seconds------
**********
Welcome to Tartarus Multi-Agent Platform
(Ver.2020.2.2, 12th October 2020)

Copyrights 2017: Robotics Lab.
Dept. of Computer Science & Engg.
Indian Institute of Technology Guwahati

(For any query contact: tartarus.iitg@gmail.com)
**********

==========
Tartarus Platform launched!
IP = localhost, Port = 11111
==========

-------platform restarted sucessfully-------
true.

### (iv) print_tartarus_status/0

print_tartarus_status. %Prints the status and parameters of the platform on the console. %

**Arity Types:**
NA

**Description:**

This predicate can be used for debugging. It prints all the values asserted in Tartarus platform, all the agent codes and threads currently associated with the platform, payloads carried by agents, token of the platform and other information related to platform.

**Example:**
?- print_tartarus_status.

Executing this predicate will list the platform details as follows:

platform_Ip : localhost platform_port : 11111
---
platform_token: 1
---
true.

### (v) assert_file_to_tartarus/1

assert_file_to_tartarus(File_name.pl). % Asserts a file with the name File_name.pl on the Tartarus platform.

**Arity types:**
File_name: <atom + >

**Description:**
This predicate is used for asserting the handler codes of the concerned agents by specifying the name of the file viz. *File_name* (variable) which contains the agent code. The handler predicates listed in the file-*File_name* are asserted as generic predicates. When the agents are created (see agent_create) these generic predicates are converted into agent-specific predicates.

**Example:**
?- assert_file_to_tartarus('handler_file.pl').

### (vi) get_tartarus_details/2

get_tartarus_details(IP,Port). % The predicate provides details of the IP, and Port of the Tartarus platform. %

**Arity types:**
IP: <atom ->
Port: <integer ->

**Description:**
It gives the IP, Port number of platform set using start_tartarus/3. The predicate comes handy when an agent or program requires to know the details of the platform where it is executing.

**Example:**
?- get_tartarus_details(Platform_IP, Platform_Port).
The output will be as follows:
**Platform_IP = localhost, Platform_Port = 6000.**

Here, we will the get IP address, and Port number of that specific platform in *Platform_IP, and Platform_Port* respectively.

### (vii) set_token/1

set_token(Token) % Sets a token in the current platform. %

**Arity types:**
Token: <integer + >

**Description:**
Whenever a platform is started a token value is assigned to it. This is used as a ticket to enter the platform. Unless an agent has this token within itself, it is not permitted entry into the platform. Agents can thus migrate to only those platforms whose tokens they carry within themselves. Tokens thus can be used to restrict /allow entry of agents to specific platforms and constitute a level of security.

**Example:**
?- set_token(9595).
Any incoming mobile agent should have 9595 as one of its tokens else it will not be allowed to enter this platform.

### (viii) save_tartarus_state/1

save_tartarus_state(FileName) % Saves the current state of the platform in a file %

**Arity types:**
FileName: <atom + >

**Description:**
This predicate saves the current state of platform into a file with name **FileName**. All associated asserted code along with facts and running agents are stored in this file. The state of platform can be resumed by calling another predicate **load_tartarus_state/1**.

**Example:**
?- save_tartarus_state('E:\\platform_state.txt').

On successful execution of the above predicate, a file is created in the specified location containing the state of platform. (In above example, .txt extension has been used. However, one may use any file format).

## (ix) load_tartarus_state/1

load_tartarus_state(FileName) % Loads the state of the platform saved in a file. %

**Arity types:**
FileName: <atom +>

**Description:**
This predicate is used in conjunction with **save_tartarus_state/1**. The state of the platform can be resumed by calling this predicate. It asserts all the associated predicates within the file FileName and asserts all the agents and related handler code. It also restarts all agents which were running at the time of **save_tartarus_state/1** predicate was called.

**Example:**
?- load_tartarus_state('E:\\platform_state.txt').

## 2.3 Predicates pertaining to Agents

### (i) create_mobile_agent/4

create_mobile_agent(Agent_name, (IP, Port), Handler,Token_list).

% Creates a mobile agent within the platform specified by the IP and Port, confers it a name and allocates its handler and a list of tokens. %

**Arity types:**
Agent_name: <atom? >
IP:<atom +>
Port: <integer +>
Handler: <atom + >
Token_list:<integer+>

**Description:**
This predicate creates a mobile agent with the name *Agent_name* given by the user. It allocates the handler (which is a predicate specified by the user) *Handler*. It is also given a list of tokens which is required to enter a platform. The agent is created in the platform running at the IP and Port specified within. This predicate automatically creates the agent specific predicates and asserts them within the specified platform. If the platform already contains an agent with the same name, it overwrites the same with this new agent having the specified name and handler.

In case the name of the agent is not specified by the user, then the predicate automatically calls get_new_name_alpha/1 and then assigns a new name to this agent.

**Example:**
Let the handler predicate for an agent be ***agent handler***.

agent_handler(guid,(IP,Port),main):- writeln("hello!!"),
writeln("I am the handler").

1. create_mobile_agent (myagent, (localhost, 6000), agent_handler,[1,2]).

This command will create an agent with name “myagent” as specified by the user carrying code specified by the handler “agent_handler”. On successful execution of the predicate, the prolog terminal will display following result:

Agent with name myagent created.

2. create_mobile_agent (Agent_name, (localhost, 6000), agent_handler,[1,2]).
In this, the *Agent_name* is generated by calling get_new_name_alpha/1. Platform will automatically generate a name for agent and then with that name agent is created. On successful creation of agent, the predicate will display following result:

Agent with name cvfzap created Agent_name = cvfzap.
where ‘cvfzap’ is the randomly generated agent name.

## (ii) create_mobile_agent/3

create_mobile_agent(Agent_name, Handler,Token_list).

% Creates a mobile agent within the current working platform by conferring it a name and allocating its handler and a list of tokens. %

**Arity types:**
Agent_name: <atom ? >
Handler: <atom + >
Token_list:<integer+>

**Description:** This is a spin-off from the earlier used create_mobile_agent/4 with less number of arguments. Through this predicate, the mobile agent is created within the current working platform.

## (iii) create_static_agent/4

create_static_agent(Agent_name,(IP,Port),Handler,Token_list)

%Creates a static agent at the specified IP and Port and allocates its handler and a list of tokens.%

**Arity types:**
Agent_name: <atom? >
IP:<atom +>
Port: <integer +>
Handler: <atom + >
Token_list:<integer+>

**Description:**
This predicate creates a static agent with the name *Agent_name* given by the user. It allocates the handler (which is a predicate specified by the user) *Handler*. It also adds a token list to the agent which might be required by the agent for cloning to another platform. The agent is created in the platform running at the IP and Port specified within. This predicate automatically creates the agent specific predicates and asserts them within the specified platform. If the platform already contains an agent with the same name, it overwrites the same with this new agent having the specified name, handler.

**Example:**
1. ?- create_static_agent(myagent, (localhost, 4545), function,[1,2]).
2. ?- create_static_agent (Agent_name, (localhost, 4545), function,[1,2]).

In 1., the predicate creates all the agent-specific predicates and asserts them within the platform running in the localhost at port 4545. The agent name is *myagent*. In the second example, the name of the agent is unbound. The predicate automatically calls get_new_name_alpha/1 and then assigns a new name to this agent.

### (iv) add_token/2

add_token(Agent_name, Token List) % adds a list of tokens to an agent %

**Arity types:**
Agent_name: <atom +>
Token List: <List +>

**Description**:
This predicate is used to add a list of tokens to a given agent. When a mobile agent enters a platform, the platform first checks the members of this list of tokens with the token of its own platform. If a match if found, the agent is allowed to enter the platform otherwise a negative acknowledgement is given to the sender platform.

**Example:**

?- add_token(myagent,[1111,2222,3333]).

?- add_token(Agent_name,[7878]).

### (v) purge_agent/1

purge_agent(Agent_name).

% Terminates the agent and its associated predicates within the current platform.%

**Arity types:**
Agent_name: <atom +>

**Description:**
This predicate is used to kill and retract all associated agent code from the current platform. Th e Agent_name is the name of agent to be terminated. purge_agent /1 purges all the threads associated with the specified agent after which all agent-specific code is retracted.

**Example:**
**?-** purge_agent(myagent).

### (vi) execute_agent/3

?- execute_agent(Agent_name,(IP, Port), Handler) % Executes the *default* handler code of the agent at the specified platform. %

**Arity types:**
Agent_name: <atom+>
IP: <atom + >
Port: <integer +>
Handler: <atom +>

**Description:**

This predicate is used for starting the executing of an agent which has already been created (and is not executing) on the specified platform. The predicate calls the associated handler predicate with the default 'main' keyword within its arguments. Before execution it removes any running agent with the same name in the platform. The **Agent_name** is the agent name of agent on the platform, **IP** is the IP address and **Port** is port number of platform where the **Handler** is to be executed.

**Example:**

Assume that an agent named *myagent* is already created. Let the agent's handler code be as below:

```
agent_handler(guid,(IP,Port),main):-
writeln('Agent's main function called!!'), N is 0,
writeln('Value of N is '), writeln(N).
```

```
?- execute_agent(myagent, (localhost,6000),agent_handler).
```

On successful execution of the above predicate, the result will be:

Agent's main function called!! Value of N is 0

**Note:** The *guid* atom as the first argument to the handler code is replaced by the name of the agent whenever agent_create/3 is executed. Hence, in agent_execute/3, only name of the agent to be executed is provided.

## (vii) execute_agent/4

execute_agent( Agent_name, (IP, Port), Handler, Start_function).

% Executes a *specified* handler code of the agent at the specified platform. %

**Arity types:**

IP: <atom + > Port:<integer+>

Handler: <atom + > Start_function: <atom+>

**Description:**

This predicate is similar to agent_execute/3 but instead of calling the default handler predicate with keyword *main* as a parameter, it will execute the one with *Start_function* as a parameter.

**Example:**

```
?- agent_execute(myagent, (localhost,6000), agent_handler, initialize).
```

Here, **Start function** is set to **initialize.** Thus, it will call the following handler predicate instead of default 'main' function:

```
agent_ handler(myagent,(localhost,9595),initialize):- <Your agent behaviour code goes here>
```

## (viii) add_payload/2.

add_payload(Agent_name, PredList) % Adds the list of predicates *PredList* whose clauses need to be carried by the specified agent as it migrates. %

**Arity types:**
Agent_name: <atom+>
AddList: <atom list +>

**Description:**
add_payload /2 add the payload to the agent. The payload predicate should have *guid* as one of the argument.

For Example: *ability(guid,X):-print(X).*

Here *ability(guid,X)* is a payload having arity 2 with guid as the first argument. Thus when command add_payload/2 is invoked, the *guid* is replaced with the actual identifier/name of the agent. *AddList* is list of predicate name and arity as *[(<predicatename1>,<number of arity>),(<predicatename2>,<number of arity>),…].*

**Example:**
Let square_coordinate/2 and point/1 be predicates defined as follows:

?- square_coordinate(0,0).

?- square_coordinate(0,10).

?- square_coordinate(10,10).

?- square_coordinate(10,0).

?- point(1).

Suppose the payload to be carried by an agent are the clauses for square_coordinate/2 and point/1. The following add_payload predicate ensures that the clauses for the above two predicates are carried with the agent named myagent as and when it migrates to another platform.
add_payload(myagent, [(square_coordinate,2), (point,1)]).

Note that the second argument is a list comprising a tuple that contains the name of the predicate and its arity. Thus in a generalized way, we can define the **add_payload** predicate as follows: add_payload(AgentName, [(<predicate1>,<Arity>), (<predicate1>,<Arity>), …]).

**CAUTION**: All predicates within the **PredList** have to be declared **dynamic**.

## (ix) remove_payload/2

remove_payload(Agent_name, RemovePredList) % Removes all clauses for the predicates specified in the RemovePredList %

**Arity types:**
Agent_name: <atom +>
RemovePredList: <atom list>

**Description:**
This predicate is used to remove the payload associated with the agent named **Agent_name**. The **RemovePredList** is the list of payload predicates which are to be removed. The structure of **RemovePredList** the same as specified in **PredList** of the **add_payload** predicate.

**Example:**
?- remove_payload(myagent,[(square_coordinate,2),(point,1)]).

In a generalized way, we can define the remove_payload predicate as follows: remove_payload(Agent_name, [(<predicate1>,<Arity>), …]).

## (x) clone_agent/3

clone_agent(Agent_name, (Destination_IP, Destination_Port), Clone_name)
% Clones the designated agent at a specified platform and confers it a new name. %

**Arity types:**
Agent_name1: <atom +>
Destination_ip: <atom + >
Destination_port: <integer +>
Agent_name2: <atom ->

**Description:**
This predicate is used for cloning of agents. The agent named **Agent_name** is cloned at **Destination_IP** and **Destination_Port** address specified by the user. The clone's name is returned in the variable **Clone_name**.

**Example:**
?- clone_agent(myagent,(localhost,5656),Clone_name). Clone_name = myagent.

## (xi) save_ agent/2

save_agent(GUID, FileName) % Saves the predicates associated to the specified agent into a file. %

**Arity types:**
Agent_name: <atom +>
FileName: <atom + >

**Description:**
This predicate is used to save all the predicates, payload and facts associated with the agent named *Agent_name* specified by user. The definitions are stored in the file with name *FileName*.

**Example:**
Let the agent handler code be:

agent_handler(guid,(IP,Port),mai n):- writeln("I am the handler"), N is 1, writeln(N).

?- save_agent(myagent,'E:\\save_agent_state.txt').

On successful execution of above predicate, a text file is created in the specified location containing agent code. The **save_agent_state .txt** file will save the agent state as:

agent_handler(myagent,(_,_),main):- writeln("I am the handler"),
N is 1, writeln(N).

The user can use the desired file type (eg. pl.txt).

## (xii) list_agents/0

list_agents. % Lists all the agent in the current platform. %

**Description:**
This predicate lists all agents existing in the current platform along with details of the agent viz. - its name, handler and port number on which agent resides. In case of a mobile agent the port number will be same as the platform port number in which it currently resides.

**Example:**
?- list_agents.

==========Agent list==========

Agent Name: agent1, handler name: mobileagent1, residing on port:5658 Agent Name: agent2, handler name: mobileagent2, residing on port:5658 Agent Name: agent3, handler name: staticagent1, residing on port:5659
==========

true.

## (xiii) isexist_agent/3

isexist_agent(Agent_name,(IP,Port),Handler) % The predicate can be used to check the existence of a specific agent in a platform. %

**Arity types:**
Agent _name: <atom ?>
IP: <atom ?>
Port: <integer ?>
Handler: <atom ?>

**Description:**
This predicate checks if any agent exists with the name provided within the predicate. If so, it provides details regarding its IP, Port number and handler. The predicate fails otherwise.

**Example:**
?- isexist_agent(agent1,(IP,Port),Handler).
?- IP = localhost,

?- Port = 5658,

?- Handler = mobileagent.

?- isexist_agent(agent1,(IP,5658),Handler).

?-IP = localhost,

?- Handler = mobileagent.

## (xiv) move_agent/2

move_agent(Agent_name,(Receiver_ip, Receiver_port))

**Arity types:**
Agent_name: <atom + >
Receiver_ip: <atom +>
Receiver_port: <integer +>

**Description:**
This predicate is used for mobility of agents the agent named *Agent_name* is moved to *Receiver_ip* and *Receiver_port* specified by user. The predicate tries to move the agent if the movement fails then agent is restarted at the same platform otherwise if movement is successful then the agent code is retracted from platform.

**Example:**
?- move_agent(myagent, (localhost,6767)).

## (xv) post_agent/3

post_agent(platform, (Receiver_ip, Receiver_port), Predicate_list )

**Arity types:**
platform: <keyword>
Receiver_ip: <atom +>
Receiver_port: <integer +>
Predicate_list: <List+>

**Description:**
This predicate is used for sending messages across platforms or between agents. '*platform*' is key word which used to tell that it is handler of platform. *Receiver_ip* is the IP address of destination platform where we want to send message, *Receiver_port* is Port number of destination platform. The *Predicate_list* is the list with handler name followed by handler parameters in sequence. The using list the predicate is created and called at destination platform.

**Caution**: Too many concurrent post_agent/3 executions may cause socket errors especially in Windows environment.

**Example:**
?- post_agent( platform, (localhost,4545),[ handler,myagent, (localhost,4545), sum(45,67)]).

Here *Predicate_list* is *[ handler,myagent, (localhost,4545), sum(45,67)]* then predicate '*handler(myagent,(localhost,4545),sum(45,67)*' will be called on destination platform.

## 2.4 Predicates pertaining to Log Server

### (i) set_log_server/2

set_log_server(IP , Port) % Sets the IP and Port number of the log server. %

**Arity types:**
IP: <atom+>
Port: <atom +>

**Description:**
This predicate is used for setting the IP address and Port number of a log server. The IP and Port address of the log server are asserted on the platform where this predicate is invoked so that all agents within this platform can use this information to send messages to the log server using send_log/2.

**Example:**
?- set_log_server (localhost, 6666).

### (ii) send_log/2

send_log (Agent_name, Message) % It causes the specified agent to send a message to the log server.

**Arity types:**
Agent_name: <atom +>, Message: <atom +>

**Description:**
This predicate is used by agents to send log messages to log server. The log server displays these messages on its terminal. The **Agent_name** is the agent sending the message to the log server.

**CAUTION**: The IP address and port number of log server has to be known to the concerned platform.
Do not forget to set the log server (**set_log_server/2**) for the same.

**Example:**
?send_log(myagent, 'Hello log server'). The log file format is as below:
<Time in seconds> <Agent name> <IP address of the platform from which log was sent> <port number of the platform from which log was sent> <Message>

Example log file format:

*1406530660.314867 hlxyll localhost 5656Hello logserver 1406530660.321872 hlxyll localhost 5657Hello logserver 1406530660.331881 hlxyll localhost 5658Hello logserver 1406530660.337885 hlxyll localhost 5659 Hello log server.*

## 2.5 Miscellaneous predicates

### (i) get_new_name/2

get_new_name(Agent_name, New_agent_name) % Generates a new name for an agent %

**Arity types:**
Agent_name: <atom + >
New_agent_name: <atom - >

**Description:**
This predicate is used for generating a unique name for an agent. The names generated by this Predicate are of form <Agent_name><random number>_<IP>_<Port>. The returned value is assigned in the variable New_agent_name.

**Example:**
?- get_new_name(agenta,X)
?- X = agenta1_localhost_5658.

### (ii) get_new_name_alpha/1

get_new_name_alpha(Agent_name) % The predicate provides a new alphabetic name that can be used to christen an agent. %

**Arity types:**
Agent _name: <atom - >

**Description:**
This predicate returns the 6-character random alphabetic name to be used for an agent.

**Example:**
?-get_new_name_alpha(New_Name).
?- New_Name = wqcfsd.

### (iii) tts/1

tts(Text) % The predicate utters the text fed to it as the argument. %

**Arity Type:**
Text: <atom +/Variable+ >

**Description:**
This predicate converts text fed to it into speech. **The text should either not contain any white space or if so, then should been closed in single quotes.** Prior to usage, kindly install the 'espeak' library as follows:

Windows:
Please execute the windows installer provisioned at "http://espeak.sourceforge.net/download.html" to install espeak in C drive.

Ubuntu/Raspbian:
Use the command "sudo apt-get install espeak"

**Note:**

1. While using 'tts', irrespective of the presence of 'espeak' library; an option for provisioning subtitles is asked when creating the tartarus platform via ('start_tartarus(_,_,_)'). If one requires the subtitles i.e. text to be displayed along with the utterance, then '1' has to be given as an option else '0' in the case otherwise.

2. 'tts' can work coherently across platforms while maintaining the choice of subtitles made for each platform. Following agent transfer scenarios have been fully tested successfully with 'tts':
a) Ubuntu to Raspbian
b) Ubuntu to Windows
c) Windows to Windows
d) Windows to Ubuntu
e) Windows to Raspbian
f) Raspbian to Windows
g) Raspbian to Ubuntu

3. If 'tts' is being used without installation of 'espeak', then the error is caught by displaying the absence of 'espeak' files at the concerned path. The subtitle choice is kept as it is and is reflected in the rest of the program. However, 'espeak' installation is mandatory for speech to be enabled.

**Example:**
?-tts(hi).
?-tts('tartarus just spoke for you').

## (iv) tts_off/0

tts_off % The predicate is used to disable the text to speech capability of the platform.%

**Arity types:**
NA

**Description:**
This predicate disables the voice output for the **tts** predicate. This does not influence the choice of subtitles made for the platform.

**Example:**
?-tts_off.

## (v) tts_on/0

tts_on % The predicate is used to enable the text to speech capability of the platform.%

**Arity types:**
NA

**Description:**
This predicate enables the voice output for the **tts** predicate if the 'espeak' library is already present. This does not influence the choice of subtitles made for the platform.

**Example:**
?-tts_on.

## (vi) subt_off/0

subt_off % The predicate is used to disable the subtitles provided for the **tts** predicate.%

**Arity types:**
NA

**Description:**
This predicate disables the subtitles provided for the audio output of the **tts** predicate. Unlike in **start_tartarus()**, this predicate can be used throughout the lifetime of a platform to affect the subtitle facility.

**Example:**
?subt_off.

## (vii) subt_on/0

subt_on % The predicate is used to enable the subtitles provided for the **tts** predicate.%

**Arity types:**
NA

**Description:**
This predicate enables the subtitles provided for the audio output of the **tts** predicate. Unlike in **start_tartarus()**, this predicate can be used throughout the lifetime of a platform to affect the subtitle facility.

**Example:**
?subt_on.

## (viii) in_hop_time/2

in_hop_time(AgentName,Time) % This predicate gives the hop time information of the requested incoming agent on the destination side. In case if the first argument is a variable, it will list the information for all the incoming agents for the platform where it is executed.%

**Arity types:-**
AgentName: <atom +>/<Variable ->
Time: <Variable ->

**Description: -**
The hoptime information of the requested agent is given in seconds.

**Example:-**
?- in_hop_time(agent1,Time).
Time = 0.015548944473266602.

## (ix) out_hop_time/2

out_hop_time(AgentName,Time) % This predicate gives the hop time information of the requested outgoing agent on the source side. In case if the first argument is a variable, it will list the information for all the outgoing agents for the platform where it is executed.%

**Arity types:-**
AgentName: <atom +>/<Variable ->
Time: <Variable ->

**Description:-**
The hoptime information of the requested agent is given in seconds.

**Example**:-
?- out_hop_time(agent1,Time).
Time = 0.015548944473266602.

## (x) in_hop_time/5

in_hop_time(AgentName, (SrcIP,SrcPort), (DestIP,DestPort), (Date,StartTime), TimeConsumed) % This predicate gives information regarding the source and destination IP and port numbers, the date and the start time of journey and the hop time information of the incoming agent on the destination side. In case if the first argument is a variable, it will list the information for all the incoming agents for that platform.%

**Arity types:-**
AgentName: <atom +>/<Variable ->
(SrcIP,SrcPort): <Variable ->
(DestIP,DestPort):<Variable ->
(Date,StartTime):<Variable ->
TimeConsumed:<Variable ->

**Description:-**
The hoptime information of the requested agent is given in seconds.

**Example:-**
?- in_hop_time(agent1,B,C,D,E).
B = (localhost,12345),
C = (localhost,123456),
D = (‘6/9/2020’, ‘22:25’),
E = 0.015548944473266602.

## (xi) out_hop_time/5

out_hop_time(AgentName, (SrcIP,SrcPort), (DestIP,DestPort), (Date,StartTime), TimeConsumed) % This predicate gives information regarding the source and destination IP and port numbers, the date and the start time of journey and the hop time information of the outgoing agent on the source side. In case if the first argument is a variable, it will list the information for all the outgoing agents for that platform.%

**Arity types:-**
AgentName: <atom +>/<Variable ->
(SrcIP,SrcPort): <Variable ->
(DestIP,DestPort):<Variable ->
(Date,StartTime):<Variable ->
TimeConsumed:<Variable ->

**Description:-**
The hoptime information of the requested agent is given in seconds.

**Example:-**
?- out_hop_time(agent1,B,C,D,E).
B = (localhost,12345),
C = (localhost,123456),
D = ('6/9/2020', '22:25'),
E = 0.015548944473266602.

## (xii) in_hop_time_stat/3

in_hop_time_stat(Maximum, Minimum, AverageInHopTime) % This predicate gives the maximum, minimum and the average hop time statistics of the incoming agents for the platform where it is executed.%

**Arity types:-**
Maximum: <Variable ->
Minimum: <Variable ->
AverageInHopTime: <Variable ->

**Description:-**
The hoptime information is given in seconds. The Maximum, Minimum and AverageInHopTime represent the maximum, minimum and average hop times for all the incoming agents for that particular node.

**Example:-**
?- in_hop_time_stat(Maximum, Minimum, AverageInHopTime).
Maximum = 0.015629053115844727,
Minimum = 0.009842157363891602,
AverageInHopTime =0.01367338498433431.

## (xiii) out_hop_time_stat/3

out_hop_time_stat(Maximum, Minimum, AverageOutHopTime) % This predicate gives the maximum, minimum and the average hop time statistics of the outgoing agents for the platform where it is executed.%

**Arity types:-**
Maximum:<Variable ->
Minimum:<Variable ->
AverageOutHopTime:<Variable ->

**Description:-**
The hoptime information is given in seconds. The Maximum, Minimum and AverageOutHopTime represent the maximum, minimum and average hop times among all the outgoing hoptimes for that particular node.

**Example:-**
?- out_hop_time_stat(Maximum, Minimum, AverageOutHopTime).
Maximum = 0.015629053115844727,
Minimum = 0.009842157363891602,
AverageOutHopTime =0.01367338498433431.

## (xiv) overall_hop_time_stat/3

overall_hop_time_stat(OverallMaximum, OverallMinimum,OverallAverageHopTime) % This predicate gives the maximum, minimum and the average hop time statistics of all the incoming and the outgoing agents of a particular node where it is executed.%

**Arity types:-**
OverallMaximum: <Variable ->
OverallMinimum: <Variable ->
OverallAverageHopTime: <Variable ->

**Description:-**
The hoptime information is given in seconds. The OverallMaximum, OverallMinimum and OverallAverageHopTime represent the maximum, minimum and average hop times of all the agents for that particular node.

**Example:-**
?- overall_hop_time_stat(OverallMaximum, OverallMinimum, OverallAverageHopTime).
OverallMaximum = 0.015629053115844727,
OverallMinimum = 0.009842157363891602,
OverallAverageHopTime =0.004557794994778103.

## (xv) in_hop_times_details/1.

in_hop_times_details(ListName) % This predicate gives information regarding the source and destination IP and port numbers, the date and the start time of journey and the hop time information of all the incoming agents for a particular node.%

**Arity types:-**
ListName: <Variable ->

**Description:-**
The output is given in the form of a list.

**Example:-**
?- in_hop_times_details(S).
S = [(agent3, (localhost, 12345), (localhost, 22222), (‘25/9/2020’,’16:1’), 0.02234196662902832) (agent1, (localhost, 12345), (localhost, 22222), ('25/9/2020', '16:3'), 0.014158010482788086)].

## (xvi) in_hop_times_info/1

in_hop_times_info(ListName) %This predicate gives the hop time information of all the incoming agents for a particular node.%

**Arity types:-**
ListName: <Variable ->

**Description:-**
The output is given in the form of a list.

**Example:-**
?- in_hop_times_info(A).
A = [(agent3, 0.02234196662902832), (agent1, 0.014158010482788086)].

## (xvii) out_hop_times_details/1.

out_hop_times_details(ListName) % This predicate gives information regarding the source and destination IP and port numbers, the date and the start time of journey and the hop time information of all the outgoing agents for a particular node.%

**Arity types:-**
ListName: <Variable ->

**Description:-**
The output is given in the form of a list.

**Example:-**
?- out_hop_times_details(S).
S = [(agent10, (localhost, 12345), (localhost, 123456), ('25/9/2020', '16:1'), 0.03461813926696777), (agent2, (localhost, 12345), (localhost, 123456), ('25/9/2020', '16:1'), 0.03356289863586426)].

## (xviii) out_hop_times_info/1.

out_hop_times_info(ListName) % This predicate gives the hop time information of all the outgoing agents for a particular node.

**Arity types:-**
ListName: <Variable ->

**Description:-**
The output is given in the form of a list.

**Example:-**
?- out_hop_times_info(A).
A = [(agent10, 0.03461813926696777), (agent2, 0.03356289863586426)].

## 2.6 RaspberryPi predicates

### (i) ledOn/2

ledOn(Pin,Status) % The predicate is used to turn an LED on.%

**Arity Types:**
Pin: <integer + >
Status: <string ->

**Description:**

This predicate is specifically created to be executed on a Raspberry Pi. It feeds a high signal to the provided 'Pin' number, thereby turning on the LED connected to the corresponding pin. Upon successful execution, it returns the 'Status' of the LED as 'led on'.

**Example:**
?-ledOn(11,Result).

**Note:**
1. 'ledOn' and 'ledOff' are facilitated as plug and play predicates that may be used to interact with external entities in a Raspberry Pi. Hence before their usage, please ensure proper and safe connection of the hardware entity being used.
2. A separate Prolog file has been facilitated for both the predicates along with the python modules. Hence prior to the usage, the files ('led.pl' and 'led.py') have to be consulted in the platform.

### (ii) ledOff/2

ledOff(Pin,Status) % The predicate is used to turn an LED off.%

**Arity types:**
Pin: <integer + >
Status: <string - >

**Description:**

This predicate is specifically created to be executed on a Raspberry Pi. It feeds a low signal to the provided 'Pin' number, thereby turning off the LED connected to the corresponding pin. Upon successful execution, it returns the 'Status' of the LED as 'led off'.

**Example:**
?-ledOff(11,Result).

### (iii) mpu6050/2

mpu6050(reading,Result) % The predicate is used to get the readings through an mpu6050 sensor.%

**Arity types:**
Pin: <string + >
Result: <string -/ list - >

**Description:**

This predicate is specifically created to be executed on a Raspberry Pi. It is used to get the readings through an mp6050 sensor connected to the Raspberry Pi. The 'reading' argument can have the following values:

'roll'/'yaw'/'pitch'/'all'/'temp'. 'Roll', 'yaw' and 'pitch' give back the corresponding reading of the sensor through 'Result' parameter. 'temp' gives back the temperature reading while 'all' gives back a list consisting of each of these readings as a tuple.

**Example:**
?-mpu6050(roll,Result).
?-mpu6050(all,Result).

**Note:**
1. 'mpu6050' is facilitated as a plug and play predicate that may be used to interact with the mpu6050 sensor in a Raspberry Pi. Hence before its usage, please ensure proper and safe connection of the sensor being used. The I2C address of the sensor is automatically read within the predicate if it is connected to the Raspberry Pi properly.

2. A separate Prolog file has been facilitated for the predicate along with the python modules. Hence prior to the usage, the files ('mpu6050.pl' and 'mpu6050.py') have to be consulted in the platform.

## (iv) mpu6050/1

mpu6050(Result) % The predicate is used to get the values of the registers of an mpu6050 sensor.%

**Arity types:**
Result: <list - >

**Description:**
This predicate is specifically created to be executed on a Raspberry Pi. It is used to get the values of the registers of an mp6050 sensor connected to the Raspberry Pi. Upon execution, the values of the 8 registers containing the acceleration and the temperature data (0x3B, 0x3D, 0x3F, 0x41 and their corresponding adjacent registers that contain the lower byte) of the sensor is returned back as a list to the 'Result' parameter.

**Example:**
?-mpu6050(Result).

## (v) servo/4

servo(Pin,Fr,Start,Result) % The predicate is used to control a servo motor through a Raspberry Pi%

**Arity types:**
Pin: <integer + >
Fr: <integer +>
Start: <float + >
Result:<string- >

**Description***:*
This predicate is specifically created to be executed on a Raspberry Pi. It is used to control a servo motor connected to the Raspberry Pi. It uses PWM method for the control. Given the pin number ('Pin'), the frequency ('Fr') and the starting duty cycle ('Start'); the motor turns to the appropriate position and returns the string 'moved' on successful execution of the predicate.

**Example:**
?-servo(11,50,2.5,Result): This generally makes the servo turn to its neutral position and returns string 'moved' to Result.

# Chapter 3

# Sample programs

## Sample 1

Create static agent which will be sitting on platform and when executed prints “Agent executed”.

Steps:

1. Open SWI-PROLOG terminal

2. Consult (load) platform.pl file
?-consult(‘path to platform.pl file’).

Platform.pl can also be directly consulted by opening platform.pl file through SWI-PROLOG terminal.
3. Start the platform on specific IP and Port using
?-start_tartarus(‘172.112.23.4’,6000,1). Or
?- start_tartarus(localhost,6000,1).

4. Load the handler file for the agent. Suppose the handler file is in agent.pl file. Then consult (load) agent.pl.

?- consult('agent.pl').

?- platform_tartarus_file('agent.pl').

5. Create static agent using create_static_agent/*4* predicate.

?- create_static_agent(agent1,(localhost,5657),myagent,[1,2]).

It will create static agent agent1 on IP address localhost and port number 5657 with agent handler name myagent.

6. To execute agent use

?- execute_agent(agent1,(localhost,5657),myagent).

true.

==========

Agent code executed IP:localhost Port:6000

==========

It will start execution from myagent handler. It calls myagent(guid,(IP,Port),main) predicate.

## Sample 2

Create mobile agent which moves from first platform to second platform.

Steps:

1. Using steps 1-4 as mentioned in Sample 1 start two Tartarus platforms.

Load mobileagent.pl file on one of the platform.
2. Create mobile agent using
?- create_mobile_agent(myagent,(localhost,5656),mobileagent,[1,2]).

Here agent with name *myagent* is created. Third parameter *mobileagent* is name of the handler.
%*** mobileagent.pl - Contains mobile agent's handler code ***%
:-dynamic mobileagent/3.

mobileagent(guid,(IP,Port),main):- writeln('Agent has reached!').

3. Start executing mobile agent using
?- move_agent(myagent,(localhost,6565)).

The execution will start from mobileagent(guid,(IP,Port),main).

2. After step-5 the other Tartarus platform will show:
?- Agent has reached!

## Sample 3

Create mobile agent that will carry a payload and will move from one platform to other.

Steps:
1. Use Step 1-5 to start two platform (If you are starting both platform on same system then make sure that the port of both platform should be different otherwise it will through socket_error exception) (For our case, platforms are started on port 6001 and 6002).

2. Load agent_handler.pl file on both platforms.

**agent_handler.pl code**:

:-dynamic agent_handler/3.
agent_handler(guid,(IP,Port),main):-
writeln('I am the handler'), (Port=6001-> current(guid,1,X,Y), writeln(X), writeln(Y), write('On platform 6001'), move_agent(guid,('1.1.1.1','2.2.2.2'))
;
writeln('Not on platform 6001'),
current(guid,2,X,Y), writeln(X),writeln( Y)).

3. Load main.pl file on one of the platform.
**main.pl**

```
current(guid,1,150,650).
current(guid,2,250,350).
start :-
get_tartarus_details(IP, Port),
create_mobile_agent(referee, (IP, Port), agent_handler,[6001,6002]),
add_payload(referee, [(current, 4)]),
execute_agent(referee, (IP, Port), agent_handler).
```

4. Start execution by typing
?- start.

(This will start executing all the statements within the predicate start in main.pl)
1. If main.pl is consulted at platform 6001 then output will be

**At platform 6001**:
I am the handler
150
650
On platform 6001

**At platform 6002:**

Not on platform 6001
250
350

And, if the main.pl file is consulted at platform 6002 then output will be:

At platform 6002:
Not on platform 6001
250
350

At platform 6001:

(No output-as agent will not go there)

(**Note:** Notice the fact that the current/4 predicate is defined only on main.pl and consulted only once on one platform but still as we are attaching it as payload with agent so when agent will move from one platform to other platform then the payload(In this example, current/4) will also move and is accessible to the destination platform).

## Error Messages

| Error Number | Error Messages |
|---|---|
| t1 | Second argument should be the port number which is an integer. |
| t2 | Last argument should be the platform token which is an integer. |
| t3 | Second argument should be the port number which is a positive integer. |
| td1 | First argument should be the IP which should be a variable here. |
| td2 | Last argument should be the Port number which should be a variable here. |
| sa1 | Third argument should be the port number which is an integer. |
| sa2 | Third argument should be the port number which is a positive integer. |
| sa3 | Fourth argument should be the handler name which is an atom. |
| sa4 | Last argument should be a list of tokens. |
| sa5 | Last argument should be a list of tokens which is an integer. |
| ea2 | Third argument should be the port number which is an integer. |
| ea3 | Last argument should be the handler name which is an atom. |
| ea4 | Third argument should be the port number which is a positive number. |
| ea5 | First argument should be the agent name which is an atom. |
| ea6 | Third argument should be the port number which is an integer. |
| ea7 | Fourth argument should be the handler name which is an atom. |
| ea8 | Last argument is the name of the parameter in the handler instead of "main" which is an atom. |
| ea9 | Third argument should be the port number which is a positive integer. |
| ma21 | Second argument should be the name of the handler which is an atom. |
| ma22 | Last argument which is a list of tokens should be in the form of a list. |
| ma23 | Last argument should be a list of tokens which is an integer. |
| ma1 | Third argument should be the port number which is an integer. |
| ma2 | Fourth argument should be the handler name which is an atom. |
| ma3 | Last argument should be a list of tokens. |
| ma4 | Last argument should be a list of tokens which is an integer. |
| ma5 | Third argument should be the port number which is a positive integer. |
| mov2 | Last argument should be the port number which is an integer. |
| mov3 | Last argument should be the port number which is a positive integer. |
| ca3 | Last argument should be the new name of the agent which is a variable. |
| ca4 | Third argument should be the port number which is a positive number. |
| ap1 | First argument which is the name of the agent should be an atom. |
| ap2 | Last argument which is the payload list should be a list. |
| rp1 | First argument which is the name of the agent should be an atom. |
| rp2 | Last argument which is the payload list should be in the form of a list. |
| sav1 | First argument which is the name of the agent should be an atom. |
| sav2 | Last argument which is a file name should be an atom. |
| gnma1 | The argument should be a variable representing the name of an agent. |
| tf1 | The argument should be the file name which is an atom. |

TARTARUS