UNIVERSITE DE TECHNOLOGIE DE COMPIÈGNE
Département de Génie Informatique

OMAS - Postman Connection

Jean-Paul Barthès

BP 349 COMPIÈGNE
Tel +33 3 44 23 44 23

Email: barthes@utc.fr

N276 v1.0
September 2012

Warning

This document discusses how to use a postman interface. It applies to OMAS version 9.0.3 and up)

Keywords

Postman, Transfer Agent

Revisions

Version	**Date**	**Author**	**Remarks**
1.0	Sep 12	Barthès	First Issue

Contents

1 Introduction

The postman interface has been revisited to allow easier deployment of platforms with some parts outside a firewall and some part on different loops inside a firewall. In addition an HTTP protocol has been implemented using the AllegroServe component. The document presents the new postman from a user point of view first, then gives some inside information.

2 Role of a Postman or Transfer Agent

A postman has two main roles:

- to connect agents of the same coterie located on different network loops;
- to interface external systems or devices.

In the first case the postman transfers all messages from an OMAS loop to another OMAS loop using either a TCP protocol or an HTTP protocol. Note that for a given application a single protocol must be chosen. Because in this case transfers are well defined, OMAS provides default skills and the programmer needs only specify a restricted number of parameters to be able to send messages to the postman of the target loop:

- the target postman key, e.g. :JPB
- the name or IP of the machine hosting the target postman
- the type of protocol :TCP or :HTTP
- its site name, e.g. :UTC

A site is defined as a set of loops inside a firewall. It has a name, e.g. :UTC or :TECPAR.
The parameters are given when the postman is created, e.g.

```
(defpostman :UTC-HTTP
            :site :UTC
            :Server T
            :internal-name "MIKONOS"
            :internal-IP "172.17.130.153" ; fixed IP
            :external-name "nat-omas.utc.fr"
            :external-IP "195.83.154.22"
            :Known-Postmen
               ((:Cit nil "219.166.183.59" :CIT :tcp)
                (:Tecpar nil "200.183.132.15" :TECPAR :tcp)
                (:notebook "jean-paulbaC4F6" nil :UTC :tcp))
            :proxy "proxyweb.utc.fr:3128"
            )
```

The example specifies that :UTC-HTTP is a server, on the :UTC site, the name of the machine on which it executes is MIKONOS, the internal IP of the machine (when addressed within the firewall) is ”172.17.130.153”, its external name (viewed from outside the firewall) is ”nat-omas.uts.fr”, and its external IP is ”195.83.154.22” it uses a TCP protocol (default) and knows three other postmen, :CIT, :TECPAR, :NOTEBOOK. The two first ones are defined by their IP and are external, the third one is defined by the name of the supporting machine. The two first ones are external, the last one is internal.

The :server option is used to launch the postman as a server automatically as soon as it is created. The proxy is unused for a TCP connection.

In the second case, the postman can be used to connect another platform or a different system (e.g. a voice component or a sensor). The parameters are then different.

```
(defpostman :UTC-VOICE
            :raw t
            :Known-Postmen
               ((:VOICE nil "127.1" :UTC :tcp))
            )
```

The :raw parameter means that we will define the :CONNECT, :DISCONNECT, and :SEND skills ourselves. The :known-postmen option is only used to post information in the postman window.

3 Technical Note

This section describes the processes provided by default. It describes first how the postman is created, then the TCP protocol, then the HTTP protocol.

3.1 Postman Creation

A postman is created by using the defpostman macro.

Parameter	value	role
name	key	name of the postman (a key)
	options	
external-ip	dotted-string	IP of the machine seen externally as a server
external-name	a string	name of the machine hosting the server
hide	t or nil	if t the agent will be hidden to the user
http	t or nil	if t indicates an HTTP protocol
http-port	a number	HTTP port number (default 80)
internal-ip	a dotted string	IP of the machine as seen inside the firewall
internal-name	a string	name of the machine as seen inside the firewall
known-postmen	a list	a list of remote postmen descriptions
proxy	a dotted string	web proxy
raw	t or nil	if t means that we provide our own skills
receiving-fcn	function name	currently unused
server	t or nil	if t starts the server immediately
site	a keyword	site name
tcp-port	a number	port for the TCP connection (default 52008)

When the postman is created, it has all the characteristics of a regular service agent and if :raw is nil has default skills: :CONNECT, :DISCONNECT, :SEND, :STATUS, :RESET, :STATUS, :SHUT-DOWN, :START.

3.2 TCP Protocol

This section describes the different skills and the corresponding actions. The skills can be invoked by using a request or inform message.

3.2.1 :START

A start action starts the server, i.e. creates a process with a receiving function waiting messages on the TCP port (default 52008). This skill is not really necessary since the CONNECT skill will start the server if needed.

3.2.2 :CONNECT

A connect action with args a remote postman description, tries to open a socket for connecting to the remote postman and produces either a success or a failure. In case of success the postman description is added to the list of active postmen info. If the server is not active, it is started automatically.

3.2.3 :DISCONNECT

A disconnect action simply removes the target agent from the list of active agent info.

3.2.4 :SEND

SEND is the crucial skill. It is called automatically when a new message appears on the local coterie LAN. The postman filters all messages addressed to it and all system messages and transfers the rest to the remote postmen.

3.2.5 :RESET

A reset action simply resets the TCP connection, closing the receive socket and restarting the receiving process.

3.2.6 :STATUS

A status action prints the status of the postman (not very useful).

3.3 Main Functions

The important functions are the receiving and the sending functions.

3.3.1 Receiving Function

Whenever the receiving process receives a message it calls the processing function.

Processing the message The message, a string, is first converted to an OMAS object. If the conversion fails, processing is simply abandoned. Then, the postman checks if the message has already been received by comparing its ID with the list of recorded IDs of the last 100 messages. If it has already been received, processing is abandoned. If the message is new to the postman, then the postman checks if the destination is itself. If so, it puts it into its own mailbox. Otherwise, the postman checks the identity of the sending agent.

Checking the identity of the sender The postman creates a new postman description with the information contained in the message and add it to the connected-postmen-info list, replacing the previous value. this guarantees that the info is up to date.

Dispatching the message Then the postman puts the message on the local coterie loop.

3.3.2 Sending Function

The postman first checks if the message is form itself or if the message is a system message. If so, it does not send it. Otherwise, it then filters the possible targets by removing all agents through which the message has already passed from the list of connected postmen. If some agents are left the message is sent to each target in turn using the postman description for each target agent. It creates a socket and sends the message to the remote postman.

3.4 HTTP Protocol

The HTTP protocol uses the same skills than the TCP one, with the exception of :SHUT-DOWN.

3.4.1 :START

A start action starts the ACL Allegroserve server. It creates a receiving address, e.g. ”nat-omas.utc.fr/omascc/80” to receive connections.

This skill is not really necessary since the CONNECT skill will start the server if needed.

3.4.2 :CONNECT

A connect action with args a remote postman description, tries to open a socket for connecting to the remote postman and produces either a success or a failure. In case of success the postman description is added to the list of active postmen info. If the server is not active, it is started automatically.

3.4.3 :DISCONNECT

A disconnect action simply removes the target agent from the list of active agent info.

3.4.4 :SEND

SEND is the crucial skill. It is called automatically when a new message appears on the local coterie LAN. The postman filters all messages addressed to it and all system messages and transfers the rest to the remote postmen.

3.4.5 :RESET

A reset action simply resets the TCP connection, closing the receive socket and restarting the receiving process.

3.4.6 :STATUS

A status action prints the status of the postman (not very useful).

3.4.7 :SHUT-DOWN

A shut-down action closes the HTTP server, terminating the receiving processes.

3.5 Additional OMAS Message Fields

Some additional fields have been added to OMAS messages to let postmen obtain the necessary information for answering messages or for avoiding sending messages in a loop. The fields are:

Field	typical value	role
id	115023273	a random tag identifying the message
sender-ip	”198.162.1.124” or ”pegasos”	IP or name of the machine sending the message
sender-site	(:UTC :JPB)	a list giving the site key and the key of the sending postman
thru	(:UTC :TECPAR)	a list of sites (posmen ids) through which the message already went

4 Examples of Inter-coterie Connections

This section describes the example of the set up at UTC, showing the different postmen being used in the different connections inside and outside the firewall (Fig.1).

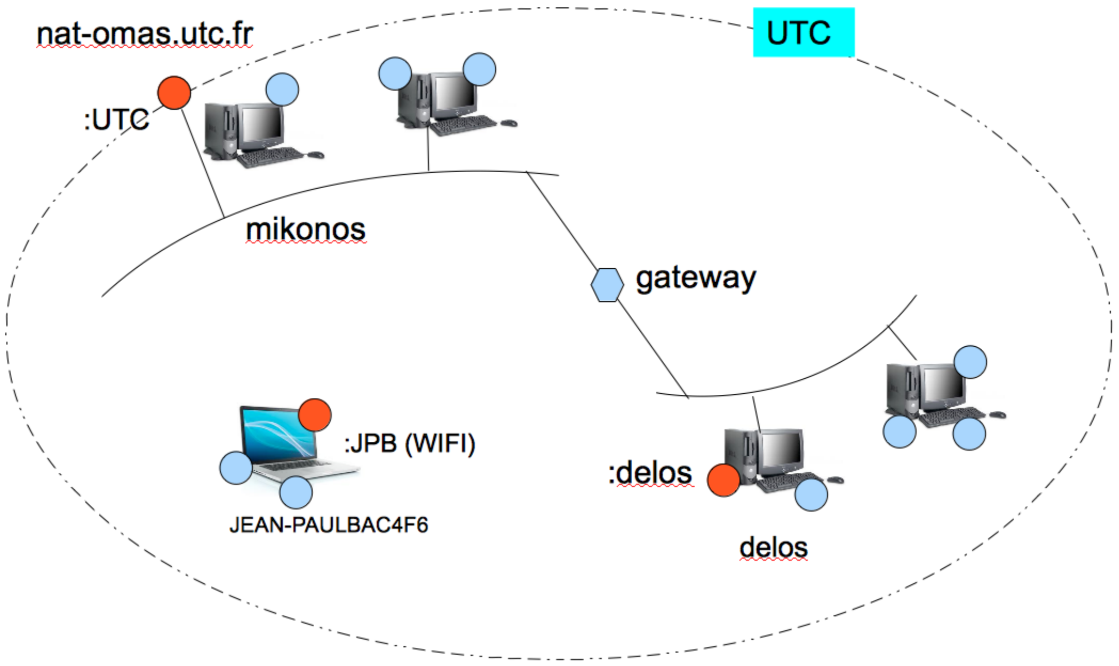

Figure 1: UTC site

One can see three postmen on Fig.1: :UTC, :JPB, and :DELOS respectively hosted by machines named: mikonos, jean-paulbac4f6, and delos. The whole site is named :UTC. The role of the postmen is the following:

- :UTC deals with internal connections and external connections to other sites, e.g. :CIT or :TECPAR;
- :DELOS deals with connections between internal loops within the firewall (expressed by the dashed line);

- :JPB deals with the internal connection within the firewall implemented through WiFi.

The different postmen can be declared as follows starting with the simpler ones.

4.1 Internal Connections

:DELOS and :JPB take care of internal connection. The content of the corresponding files will appear as follows:

```
;;;-*- Mode: Lisp; Package: "DELOS" -*-
;;;===========================================================================
;;;12/09/15
;;;                           AGENT POSTMAN :DELOS
;;;
;;;===========================================================================

(defpackage :DELOS (:use :moss :omas :cl #+MCL :ccl))
(in-package :DELOS)

(omas::defpostman :DELOS
    :server t
  :site :UTC  ; requested
  :internal-name "delos"
  :known-postmen  ((:UTC "mikonos" "172.17.130.153" :TCP :UTC))
  )

;;; uses default skills
:EOF
```

The postman parameters indicate that the postman name is :DELOS, it is a server (meaning that it will be started to receive messages as soon as the declaration is executed), its site is :UTC (inside the UTC firewall), the name of the host machine is delos, and it knows a postman named :UTC located on a machine named mikonos the IP of which is 172.17.130.153, the connection is done on a direct TCP mode on default port 52008 (default). Note that either mikonos or its IP are needed. If the IP is not given it will be computed from the name (the DNS will be asked).

There is essentially no difference for a WiFi connection with the notebook as shown here.

```
;;;-*- Mode: Lisp; Package: "JPB" -*-
;;;===========================================================================
;;;10/08/12
;;;                           AGENT POSTMAN :JPB
;;;
;;;===========================================================================

(defpackage :JPB (:use :moss :omas :cl #+MCL :ccl))
(in-package :JPB)

(omas::defpostman :JPB
    :server t
  :site :UTC  ; requested
  :internal-name "JEAN-PAULBAC4F6"
  :known-postmen  ((:UTC "mikonos" "172.17.130.153" :TCP :UTC))
  )

;;; uses default skills
:EOF
```

4.2 External Connection

Postman :UTC deals both with internal and external connections. For internal connections, the hosting machine is known as MIKONOS. For external connections the machine is known as nat-omas.utc.fr, e.g. when it is addressed from CIT or from TECPAR. The corresponding code is shown here.

```
;;;-*- Mode: Lisp; Package: "UTC" -*-
;;;===============================================================================
;;;12/09/15
;;;                          AGENT POSTMAN :UTC
;;;
;;; Postman to transfer messages from :UTC to remote OMAS platforms
;;;===============================================================================

(defpackage :UTC (:use :moss :omas :cl #+MCL :ccl))
(in-package :UTC)

(omas::defpostman :UTC
  :server T
  :site :UTC
  :internal-name "mikonos"
  :internal-IP "172.17.130.153"
  :external-name "nat-omas.utc.fr"
  :external-ip "195.83.154.22"
  :known-postmen ((:cit nil "219.166.183.59" :TCP :CIT)
                  (:tecpar nil "200.183.132.15" :TCP :TECPAR)
                  (:jpb "jean-paulbaC4F6" nil :tcp :UTC))
  :proxy "proxyweb.utc.fr:3128"
  )

;; uses default skills
:EOF
```

Note that we have here additional parameters: external-name, external-ip, and proxy, although this last information is not needed for a direct TCP connection. Note that the internal-name and internal-ip are not really necessary, the IP can be computed if needed.

If the exchanges are to be done using the HTTP protocol, then the parameter (:http t) must be specified. Note that in that case it is recommended that all exchanges be done using the HTTP protocol.

5 Controlling an External Device

The postman mechanism can be used for controlling an external device, e.g. a particular sensor. In that case one must provide the necessary skills and at least the SEND and CONNECT skills. The idea is whenever an agent sends a message to :TEMPCONTROL, the message will be automatically transferred to the device.

The following code would control a temperature controller defined as a pseudo agent named TEMPCONTROL. We assume that the CONNECT skill initiates the receiving function, and that the exchanges with the controller occur through sockets and use a TCP protocol.

```
;;;-*- Mode: Lisp; Package: "CONTROL" -*-
;;;===============================================================================
;;;12/09/15
;;;                          AGENT POSTMAN :CONTROL
;;;
;;; Postman for connecting a temperature controller
;;;===============================================================================

(defpackage :CONTROL (:use :moss :omas :cl #+MCL :ccl))
(in-package :CONTROL)

(omas::defpostman :UTC
  :site :UTC
  :raw t
  :known-postmen ((:TEMPCONTROL "tempcontrol"  <tempcontrol-ip> :TCP :UTC))
  )
;;; "tempcontrol" is the name of the temperature control host

;;;===============================================================================
;;;                                  globals
;;;===============================================================================

(defparameter *output-port* 7000 "output socket port")
(defparameter *input-port* 7001 "input socket port")
(defparameter *tempcontrol-connected* nil "flag keeping track of connection")
(defparameter *tempcontrol-ip* <temperature controller IP>)

;;;===============================================================================
;;;                                   Service functions
;;;===============================================================================

(defUn postman-receiving (agent port)
  "function used by the process that waits for incoming messages.
We assume that the incoming message is a string.
Arguments:
   agent: postman
   port: receiving port
Return:
   never returns, wait in a loop for messages"
  (unwind-protect
      ;; create a passive socket, listen to port "port" on localhost
      (let ((p (socket:make-socket :connect :passive :local-port port :backlog 50))
            message-string message s)
        ;; record socket object
        (setf (receive-socket agent) p)
        (format t "~&; ~S/ receiving/ passive socket created: ~&;  ~S"
          (key agent) p)
       ;; record that we are connected
       (setq *tempcontroller-connected* t)

        (loop
          ;; create stream for connection requests
          (setq s (socket:accept-connection p))
          ;***** see if we can get IP of sending host from the socket structure here
          ; by using (socket:remote-host s)
          ;; get incoming message, returning nil if message is empty (:eof reached)
          (setq message-string (read s nil nil))
          ;; print trace into Lisp console
          (format t "~&<<=== ~S/ receiving/ incoming message: ~&  ~S"
            (key agent) message-string)
          ;; close connection
          (close s)

          ;; if message-string is nil, then the message was empty, give up, go wait
          ;; for the next one (this is an unlikely case). Otherwise go process it
          (when message-string
            ;; process message converting it to an OMAS object and broadcasting it
            (user-process-message agent message-string))
          ))

    ;;=== unwind-protect clause, used when the receiving process is aborted, or the
    ;; plateform exits to do some clean up
    (progn
      ;; close passive socket
      (if (receive-socket agent) (close (receive-socket agent)))
      (setf (receive-socket agent) nil)
      )
    ))

;;; the following function could for example build an OMAS message and broadcast it.
(defun user-process-message (agent message-string)
   "user defined function for processing the input from the temperature controller"
   ...
   )
```

```
;;;===============================================================================
;;;                                   Skills
;;;===============================================================================

;;;========================================================================= skill
;;;                                  CONNECT
;;;===============================================================================

(defskill :connect :control
  :static-fcn connect-static)

(defun connect-static (agent message)
   "set up the receiving process"

(defUn connect-static (agent message)
  "the CONNECT skill installs the receiving function."
  (declare (ignore message)(special *input-port*))
  ;; if controller already connected, give up
  (if *tempcontol-connected*
      (return-from connect-static (static-exit agent :done)))

  ;; otherwise create a receiving process, and record it in the agent structure
  (setf (omas::receiving-process agent)
    (mp:process-run-function
     (format nil "~S Receiving" (omas::key agent)) #'postman-receiving agent *input-port*))

  ;; paint the connection pane green for 1/10s showing OK
  (omas::pw-show-success agent)

  ;; refresh postman window, which will show connections
  (omas::agent-display (omas::%agent-from-key (omas::key agent)))
  (static-exit agent :done)))

;;;========================================================================= skill
;;;                                  SEND
;;;===============================================================================

(defskill :send :control
   (static-fcn static-send)

(defun static-send (agent in-message message-string message)
  "skill that sees every message and filters those for the temperature controller.
arguments:
   agent: postman
   in-message: incoming message
   message-string: message to send, a string (ignored)
   message: message to send, an object"
  (declare (ignore in-message message-string)(special *tempcontrol-ip* *ouput-port*))
  (let (text socket-id)
    ;; check if message is for voice
    (unless (eql :tempcontrol (omas::to! message))
       (return-from static-send (static-exit agent)))

    ;; yes transfer the content to the temperature controller
    ;; create socket for sending and try to catch socket errors
    ;; the remote host must listen on the TCP port!
    (multiple-value-setq (test errno)
      (ignore-errors
       ;; create a socket to connect with the remote host
       (setq socket-id
             (socket:make-socket :remote-host *tempcontrol-ip* :remote-port *output-port*))
       ))

    ;; when socket is non nil, we can send
    (when socket-id
      ;; write message to stream
      (format socket-id "~S" (format nil "~S" value))
      ;; make sure the buffer is emptied
      (force-output socket-id)
      ;; close the socket before leaving
      (close socket-id)
      ;; print trace into the Lisp console
      (format t
          "~&===>> ~S/ send-remote/ message (ID: ~S) sent to: ~A:~S. Message:~&  ~S"
        (omas::key agent) (omas::id message) ip *output-port*  (format nil "~S" value))
      )
    (static-exit agent :done)))
:EOF
```

Warning The above functions have not been tested (lack of temperature controller), therefore they might be buggy...