IAOP 1.1 Specifications

Table of Contents

Overview 3
Architecture 3
Functionality 4
Underlying Protocol 4
Request Format 5
Reply Format 7
Agent Typical Flow Chart 9
Agent register request 10
Keep-alive request 11
Alarms update request 12
  alarm 12
  clear 13
  event 14
Fetch configuration request 15
Set parameter request 17
Get binary request 18

Overview

IAOP (I-Am-On Protocol) is the protocol that allows R-U-ON agents to communicate with the R-U-ON servers and pass information regarding the health of the resources it is monitoring.

This document describes the functionality of the R-U-ON protocol, its underlying layers and its packet structure.

The document is intended for programmers who are looking to implement an IAOP stack. Alternatively, there are freely available APIs for C, C# and Java (see http://www.r-u-on.com/ctrl?action=developers).

This document assumes general understanding of data communication and experience with HTTP and XML

Architecture

The R-U-ON monitoring solution is a web hosted, agent based system. It lets users run compact monitoring agents on their network that report to the R-U-ON managements system. The management system stores this information and lets users access the information using tools such as web browsers, email notification and RSS (Real Simple Syndication).

Through out this document the R-U-ON agent will be referred to as “the agent” and the R-U-ON server will be referred to as “the server”.

Functionality

The IAOP protocol supports the following functions:

1. Agent registration – When an agent is installed it gets a unique id from the R-U-ON server that identifies it through its entire life-cycle, until it is uninstalled.
2. Keep alive – The agent notifies the server that it is up and running.
3. Alarm-up notification – The agent notifies the server that a new alarm has occurred.
4. Alarm down notification – The agent notifies the server that an existing alarm state is cleared.
5. Alarm snapshot – The agent sends the server a full list of alarms. Alarms that are not listed are assumed to be cleared.
6. Upgrade – The server can send the client an indication that it is running an outdated version and supports a request for acquiring a new version of the agent binary.

Underlying Protocol

IAOP is based on HTTP(S) 1.0. This ensures maximum firewall compatibility and a high security level (if SSL is used).
Using HTTP means that all communication is initiated from the Agent side, usually residing behind a firewall.

Requests from the agent to the server are expressed in a XML file sent as the payload of a POST command. The size of the post payload cannot exceed 10KB (10240 bytes). If it does, the request will be ignored.
Request results are expressed in a XML file sent as the payload of the result file.

All requests are sent to the URL: http(s)://agent.r-u-on.com/iaop1
More explicitly:
Host: agent.r-u-on.com
HTTP Port: 80
HTTPS Port: 443
HTTP Resource: /iaop1

The server ignores the Content-Type header field. Do not URL-Encode the message payload as that will create an illegal XML document.
You are encouraged to use HTTPS and check the certificate to make sure you are communicating with agent.r-u-on.com.

Request Format

Every XML sent from the agent has a root node that is called <iaop>.
The node contains the following attributes:

pver: The protocol version. This should have the value 1. This field denotes the protocol's major version. It will only change when a non backward compatible change is introduced.

atype: The type of the agent. Shared R-U-ON agents have a naming convention of ruon.NAME. When creating a user agent (an agent that is not to be published on the R-U-ON server) you should not use this prefix to assure name uniqueness.

aver: The current version of the agent. This is used both for display purposes and to initiate an automatic upgrade if the version does not match the latest version. Naturally, only official R-U-ON agents can be upgraded by the R-U-ON server.

os: Operating system name. This is a free text field and is not limited to a selection of systems.

id: The agent unique id. This attribute is ignored if the XML is a registration request (since id is still unknown). In the agent register request, this attribute denotes the customer id and not the agent id (since the agent id has not been obtained yet).

ip: The IP of the machine the agent resides on.

host: The host name of the machine the agent resides on. If the machine does not have a host name you can reuse the machine's IP.

time: The current time on the agent, using the agent's time zone and day light saving settings.
The format for the time is: yyyy-MM-dd hh:mm:ss
Hour should be expressed in 24 hour based (example 00:00:00 is midnight, 13:00:00 is 1pm, etc.).

uptime: The time in seconds the agent process has been up.

iter: The number of times the agent has successfully made contact with the server. This field is used to determine if an agent has rebooted since the last contact. The counter should be incremented every time you receive a valid, non-error, IAOP response.

The specific request node resides inside the <iaop> node. There can only be one request per request response session.

Sample:

```
<iaop pver="1" atype="MyAgent" aver="1.01" os="Linux"
 ip="192.168.0.1" host="server1" id="SMPLAGENTID"
 time="2006-07-25 13:23:04" uptime="3421" iter="5">
 <keepalive />
</iaop>
```

The request XML must conform to all the formatting rules governing the XML format. The R-U-ON server uses a standard DOM parser to parse the request.

Reply Format

Replies are returned in a <iaop> node that has no attributes.

A successful reply will in most cases be a single directive to be executed by the agent. The directive resides in a <directive> node inside the <iaop> node.

Sample:

```
<iaop>
   <directive>sleep:285</directive>
</iaop>
```

Directives always have the format verb:X where verb is the action to perform and X is a modifier that changes in meaning, depending on the verb.

Possible directives are:

sleep: Indicates the length of time, in seconds, the agent has to wait before making contact with the server. The format for the sleep directive is “sleep:X” where X indicates the number of seconds. When the agent reports an alarm a new sleep value might be returned and it overrides the old one. Reporting alarms is equivalent to a keep-alive request. When alarms need to be reported, the agent may occasionally ignore the directive and report before its scheduled time.

configure: Tells the agent that the user changed the configuration. As a result, the agent should invoke the “Fetch Config” request.
X is the number of seconds till the next keep-alive is required (exactly like the sleep directive).

uninstall: Tells the agent a user has requested the agent to uninstall. The agent is removed from the server side. A typical agent would clean all its files and resources from the system it resides on and stop running. Even if a full uninstall is not possible, the server does not expect the agent to make contact again. The agent id is no longer valid and should be deleted. X is a message describing the reason for this directive.

die: Indicates the agent should die. This is a rare occurrence giving a chance for the server to stop an agent that has been found delinquent. The format of the die directive is “die:X” where X is the reason the server has requested the agent’s death. The agent should stop reporting but does not have to uninstall – its record is still active on the server side. Typically this would require user intervention to recover from.

upgrade: Tells the agent it needs to perform an upgrade. A normal agent behavior would be to execute an upgrade request. The server will not send the upgrade directive to agents that do not support an upgrade.
X contains the version number to upgrade to (for logging purposes only; it is not required for the actual upgrade as the server will always deal the latest).

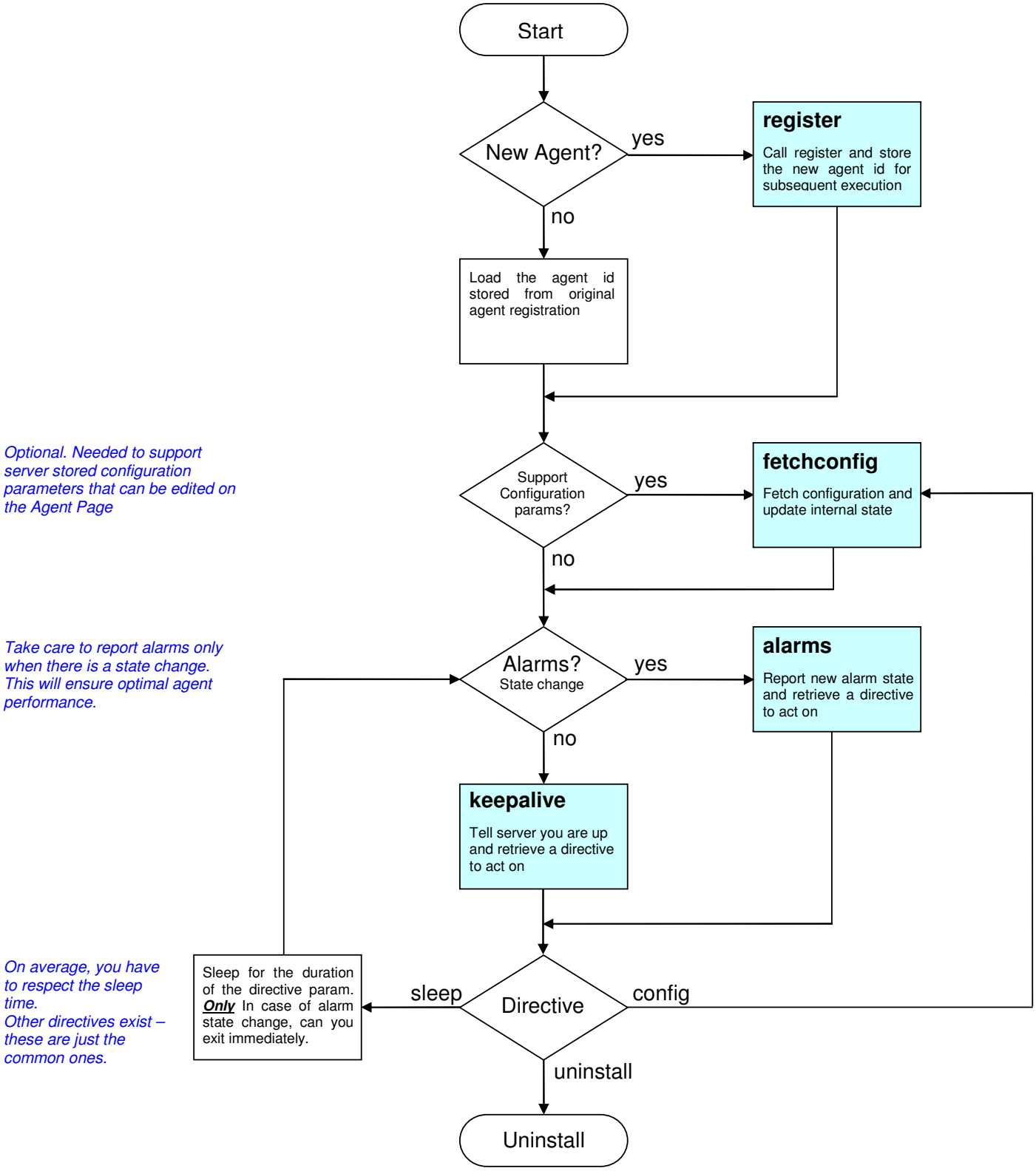
If an error occurs it will reside in an <error> node within the <iaop> node.
Sample:

```
<iaop>
   <error>Unkown agent id</error>
</iaop>
```

For some requests the expected reply is not a directive. See each request's documentation for specifics.
If the server returns a reply that does not start with <iaop> it should be ignored by the agent. This will happen when the server is initializing and is not ready to receive IAOP requests.

Replies are guaranteed to be sent with no whitespaces between the XML nodes.

Agent Typical Flow Chart



Agent register request

Until the agent obtains a unique identifier, it should only try and issue the register request. The ID obtained from this request should be used for every subsequent request, until the agent is uninstalled. The id should not be obtained each time the agent process is run but only once per agent installation.

The register request resides inside a <iaop> node, with one difference. The id tag denotes the customer id and not the agent id. With user agents this information is supplied by the user who can obtain it by logging into his account and locating the information in the account settings page. With shared R-U-ON agents this information is embedded in the executable during the download process (and therefore does not need to be supplied by the user).

The agent register request node is called <register> and resides within a <iaop> node. It has one optional attribute.
skaa: Suppress-Keep-Alive-Alarms. Setting skaa to true will create the agent with the keep-alive-alarms option turned off (by default it is turned on). The user can always change this setting manually from the agent page.

Sample:

```
<iaop pver="1" atype="MyAgent" aver="1.01" os="Linux"
 ip="192.168.0.1" host="server1" id="CUSTIDGOESHERE"
 time="2006-07-25 13:23:04" uptime="3421" iter="5">
 <register skaa="true"/>
</iaop>
```

A successful reply to this request has the newly created id in it inside an <id> node.

Sample:

```
<iaop>
   <id>NEWAGENTID</id>
</iaop>
```

Keep-alive request

The keep-alive request is issued by the agent periodically to notify the server that the agent is healthy. If the agent misses a keep-alive request, the server will assume the agent is down and notify the user accordingly. A typical response to a keep-alive request is a directive telling the agent how long it should wait (in seconds) until the next keep alive request (other directives are possible). This number contains in it necessary slack to account for a slow network. The agent should try not to contact ahead of time as it may be declared delinquent by the server.

The keep-alive request is represented by a node named <keepalive> contained within a <iaop> node.

Sample:

```
<iaop pver="1" atype="MyAgent" aver="1.01" os="Linux"
  ip="192.168.0.1" host="server1" id="SMPLAGENTID"
  time="2006-07-25 13:23:04" uptime="3421" iter="5">
  <keepalive />
</iaop>
```

The reply is a directive to be executed by the agent (see Reply Format).

If the agent cannot achieve a connection to the server, it should retry every few seconds (three is a good number) for the next 30 seconds. This will assure that a short network glitch will not be interpreted as a failed agent.
However, after a period of short retries the agent must increase the interval between retries to at least 60 seconds. This ensures that incase of connection loss, the agents do not overwhelm the R-U-ON server farm.

Alarms update request

The alarms update request allows the agent to report a problem in the resource the agent is monitoring. The server will note the alarm and inform the user.

The server supports two alarms update modes, incremental and snapshot. In incremental mode, the agent can report additions or deletions to and from the list of currently active alarms on the server. In snapshot mode, the agent gives the full list of active alarms. Any alarm that exists in the server and is not reported within the snapshot is cleared.
Using a snapshot request with no alarms in it will remove all active alarms for the particular agent.

An alarm update request is indicated by a node called <alarms> within the <iaop> node containing alarm state notification change. The <alarms> node has one attribute called “incremental” that can have a value of “true” or “false”. If the value is “true” alarms within the <alarms> node are treated as state changes. If it is “false” the list of alarms is treated as the current snapshot of active alarms for the agent. When “incremental” is “false” it does not make sense to include alarm clear notifications (although the server will still treat them).

Inside the <alarms> node there can be several alarm state notifications. The three types are “alarm”, “clear” and “event”. “alarm” is used to report a new alarm or change the fields of an existing alarm and “clear” is used to clear an alarm. “event” is used to indicate an alarm that should be reported but not maintained as active. It won’t appear as an active alarm but will be notified to the user and will appear in the alarm history report.

alarm

“alarm” is indicated by an <alarm> node inside the <alarms> node. It contains the following attributes:

resource: The resource identifies the module that is not functioning within the monitored entity. The resource uniquely identifies the element that is malfunctioning. If the resource is uniquely identified by a tree (for instance a port that belongs to card) it is recommended to use a slash (‘/’) as a delimiter. For example if monitoring ports inside cards a good way to indicate the resource is “card12/port3”.

The resource is a characters string with a maximum length of 255 characters.

id: The unique id of the alarm (per agent and resource). The id uniquely identifies the alarm from the set of alarms for a specific agent/resource combination. Subsequent calls to raise an alarm with the same id and resource will result in field changes to the alarm already up. An alarm down indication will use this id to indicate which alarm to clear. The id is a string of up to 255 characters. The id cannot contain characters that will compromise the XML structure (<.>,", etc.).
The id usually includes the reason the problem is occurring on the specified resource as long as that preserves uniqueness. For instance if an agent is monitoring cards inside a server and one of the cards stop reporting, the id can be "OFF" and the resource contain.

severity: The alarm severity can have one of the following values "C", "M", "m" standing for Critical, Major and Minor respectively.

In addition the <alarm> node contains a CDATA node with the alarm description. The description field cannot exceed 1KB (1024 characters). It cannot contain the string "]]>" as it will compromise the XML document integrity.

Sample:

```
<iaop pver="1" atype="MyAgent" aver="1.01" os="Linux"
 ip="192.168.0.1" host="server1" id="SMPLAGENTID"
 time="2006-07-25 13:23:04" uptime="3421" iter="5">
 <alarms incremental="false">
    <alarm resource="CARD1/PORT3" id="ETHOFF" severity="M">
       <![CDATA[Ethernet card 1, Port 3 is on fire]]>
    </alarm>
 </alarms>
</iaop>
```

clear

"clear" is indicated by a <clear> node within the <alarms> node. The node has the following attributes to uniquely identify the alarm to be cleared:

resource: Indicates the resource the alarm was reported on.

id: Indicates the id of the original alarm.

In addition, the <clear> node can optionally contain a CDATA node with the alarm clear description. If this node exists it is used for user notification. The node cannot exceed 1KB (1024 characters). It cannot contain the string "]]>" as it will compromise the XML document integrity.

Sample:

```
<iaop pver="1" atype="MyAgent" aver="1.01" os="Linux"
 ip="192.168.0.1" host="server1" id="SMPLAGENTID"
 time="2006-07-25 13:23:04" uptime="3421" iter="5">
 <alarms incremental="true">
    <clear resource="CARD1/PORT3" id="ETHOFF">
       <![CDATA[Fire has been extinguished]]>
    </clear>
 </alarms>
</iaop>
```

event

"event" is indicated by the <event> node within the <alarms> node. The node has the exact same attributes and options as the <alarm> node.

Sample:

```
<iaop pver="1" atype="MyAgent" aver="1.01" os="Linux"
 ip="192.168.0.1" host="server1" id="SMPLAGENTID"
 time="2006-07-25 13:23:04" uptime="3421" iter="5">
 <alarms incremental="false">
    <event resource="CARD1/PORT3" id="ETHOFF" severity="m">
       <![CDATA[The port had a small glitch]]>
    </event>
 </alarms>
</iaop>
```

A successful reply to this request is a directive to be carried out by the agent (see Reply Format).

Fetch configuration request

The configuration request is used by agents that need input from the user.
It should be sent when the agent goes up to get its configuration and as a response to the “configure” directive to load configuration changes made by the user.

The request includes the full list of fields (metadata) that the agent expects. This enables user agents to have a configuration without a separate configuration of metadata.

The request is indicated by a <fetchconfig> node within the <iaop> node.
<fetchconfig> has two sections in it, the first for agent parameters and the second for resource parameters.

Agent parameters are parameters that affect the way the agent works. For instance: a general timeout configuration, a threshold, etc. The resource parameters are the parameters that refer to a managed-resource. A managed resource is used when the agent manages a list of resources that is provided by the user. PingOn is an example of such an agent, where the resource list is a list of ip/hosts that will be pinged periodically.

The agent parameters are contained within an <agent> node and the resource inside a <resources> node. Each contains a list of parameters metadata.

Each item in the list is a <param> node that has the following properties:

name: The name of the parameters as will be visible to the user and referred to when a result is received. The name can contain only alphanumeric characters.

type: The type of the variable. The possible values are:
Integer: Negative and positive integer.
Boolean: Either “true” or “false”.
String: Single line string
Text: Multiple line string
It is possible to indicate the unit to be displayed to the user in parenthesis, after the type, such as “Integer(%)”.

default: The default value for this parameter when not supplied by the user.

secret: Indicates if the value should be displayed on screen. Possible values are “true” and “false”. This parameter is optional.

Request sample:

```
<iaop pver="1" atype="MyAgent" aver="1.01" os="Linux"
 ip="192.168.0.1" host="server1" id="SMPLAGENTID"
 time="2006-07-25 13:23:04" uptime="3421" iter="5">
 <fetchconfig>
     <agent>
          <param name="threshold" type="Integer" default="85"/>
          <param name="timeout" type="Integer" default="300"/>
     </agent>
     <resources>
          <param name="ip" type="String" default=""/>
          <param name="port" type="Integer" default="8080"/>
     </resources>
 </fetchconfig>
</iaop>
```

As a result the agent will get a <config> message which has in it agent parameters and a list of resources and their respective parameters.
Directly underneath the config node is a list of nodes, each one representing an agent variable. The variable name is the node name and its value the node value (example: <name>value</name>).
After the list of agent variables there is a list of resources inside a <resources> node.
Each resource is inside a <resource> node. Each <resource> node contains a list of nodes where the node name is the parameter name and its value the parameter value.

Result sample:

```
<iaop>
 <config>
    <resources>
       <resource>
          <ip>197.168.1.100</ip>
          <port>8080</port>
       </resource>
       <resource>
          <ip>197.168.1.107</ip>
          <port>8000</port>
       </resource>
    </resources>
 </config>
</iaop>
```

Set parameter request

The set-param request is issued by the agent in order to set parameters stored by R-U-ON. This allows agents to change settings from their own interface when the use case dictates it.

The request is indicated by a <setparam> node within the <iaop> node. Inside the node is a list of parameter names and their values. The values can be surrounded by a CDATA section.

Currently the following parameters are open for agent control:
alias: The name of the agent in the R-U-ON user interface.
group: The group the agent belongs to. If the group does not exist it will be created.
notif: Notification severity. Possible values: Critical, Major, Minor, none (case sensitive).
skaa: Suppress-Keep-Alive-Alarms. Possible values: true, false. Setting skaa to true will turn the keep-alive-alarms option off (by default it is turned on).

Names and values are case-sensitive.
The user can always override these settings from the web interface.

Request sample:

```
<iaop pver="1" atype="MyAgent" aver="1.01" os="Linux"
 ip="192.168.0.1" host="server1" id="SMPLAGENTID"
 time="2006-07-25 13:23:04" uptime="3421" iter="5">
 <setparam>
     <alias>The Agent</alias>
     <notif>Critical</notif>
 </setparam>
</iaop>
```

The reply is a directive to be executed by the agent (see Reply Format).

Get binary request

As a response to the upgrade directive, the agent should issue a request to get the binary and use it to upgrade itself. Only shared agents (agents that were downloaded from the R-U-ON system) can get the upgrade directive. If you are developing an agent for your own personal use, you do not need to support this request.

The request is represented by a <getbinary> node inside the <iaop> node. The node has no attributes.

Sample:

```
<iaop pver="1" atype="MyAgent" aver="1.01" os="Linux"
 ip="192.168.0.1" host="server1" id="SMPLAGENTID"
 time="2006-07-25 13:23:04" uptime="3421" iter="5">
 <getbinary />
</iaop>
```

In response, the agent receives the binary and an authentication id.
The first thing the agent should do is compare the authentication id with the one embedded in it during the first download (in the embedded variables section). This ensures that the agent is communicating with the official R-U-ON site.

If the id matches, the agent should use the binary to upgrade itself. The binary is base64 encoded in compliance with section 5.2 of RFC 1341. After the binary has been decoded, it should be installed and run by the agent. This operation is agent specific.

The first level of the response is the <iaop> node. Inside it, resides the <binary> node with the following attribute:
authentic: The secret id embedded in the executable. This should be used to make sure the binary is authentic.

The content of the <binary> node is the binary file, encoded in base64 (so as not to risk integrity of the XML document).

```
<iaop>
   <binary authentic="UNIQAUTHID">BASE64 ENCODED BINARY</binary>
</iaop>
```