SAFR from realnetworks

Actions Relay Event Service (ARES)

Documentation Version = 3.017

Publish Date = April 11, 2021

Contents

1 Actions Overview

In SAFR an action is essentially a script or macro that communicates a desired action in a language or protocol the receiving device or system understands. It can be written in any language supported by the computer where Actions Relay Event Service (ARES) is installed. It only needs to be invocable as an executable directly or through the use of another executable (usually a script interpreter such as Python).

1.1 Actions Components

These are the principle components involved with actions:

- **Actions Relay Event Service (ARES)**: ARES is a cross-platform Java application that acts as an event listener that dispatches configured actions in response to events, as defined in the SAFRActions.config file. ARES can provide replies on any event to be handled by the client originating the event and is normally installed as a service by either the SAFR Platform or SAFR Desktop installers. It is constantly active and is automatically started by the operating system on power-up.
- **SAFRActions.config**: The SAFRActions.config file defines which events will trigger specified actions. It also can specify additional condition constraints before the action(s) will trigger.

1.2 SAFRActions.config Overview

```
<name: value connection attributes>
rules: [
  {
    event: {  },
    triggers: [
        <time of day and week properties>
        actions: [  ],
        reply: {  },
        conditionalReply: {  },
    ],
    excludeDates: [  ]
  }
]
noTriggerReply: {  }
nFactorDef: [ {  }, { }, ... ]
emailDef: [  {  }, {  }, ... ]
smsDef: [  {  }, {  }, ... ]
```

- **rules**:
  - 1 or more rules can be defined.
  - When an event occurs each rule is checked to see if any of its events match.
  - A rule's event matches an occurring event when:
    - All attributes rules[i].events match the event.
  - Each rule has 1 or more triggers.
    - Each trigger inside a matching rule is fired as long as the time of day conditions match. **Exception**: If 2 *triggerId*s are identical only the first trigger is fired.
  - Each trigger has one or more actions.
    - Actions are either:
      - A shell command or a batch/shell script to be executed.
      - A send email command that has the syntax of: `@emailSend <value of emailDef.label>`
    - All actions are run asynchronously unless a *conditionalReply* is specified in which case the first rule is run synchronously (and the return code of that rule is used for the conditionalReply) while all other rules are run asynchronously.
- *noTriggerReply* is used to perform a reply if none of the triggers are fired.

- *nFactorDef* can define 2 or more conditions that must occur within the specified time window.
- *emailDef* defines one or more email message attributes (subject, from, message, etc).
- *smsDef* defines one or more Short Message Service (SMS) messages.

Examples:

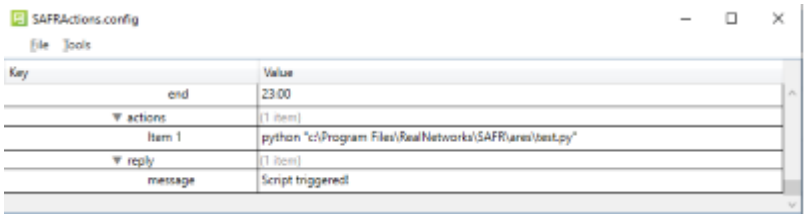
- Send email when visitor arrives during work hours
  - rules
    - Rule 1
      - event (hasPersonId=false)
      - trigger (day/hours: 8-5, M-F)
        - action: @emailSend visitorEmail
  - emailDef
    - label=visitorEmail
    - subject="Visitor Arrived"
    - message="A visitor has arrived at #I - #S."
    - ...
- Log all events to a CSV and send one type of email for a known person event and another for a threat event.
  - rules
    - Rule1 (known person email)
      - event (hasPersonId=true, idClass=No-Concern)
      - trigger
        - action: @emailSend knownEmail
    - Rule 2 (threat email)
      - event (hasPersonId=true, idClass=[Threat, Concern])
      - trigger
        - action: @emailSend threatEmail
    - Rule 3 (log)
      - trigger
        - action: ".\scripts\log_event.bat "#D" "#N" "#F" ..."
          - If editing config file, escape backslash or quotes with another backslash. (In SAFR Actions no escaping is needed.)
          - The file 'log_event.bat' should be placed in `C:\Program Files\RealNetworks\SAFR\ares\scripts` (for Windows) or `/Library/RealNetworks/SAFR/ares/scripts` (for macOS).
  - emailDef
    - 1 (label=knownEmail, subject, message, etc)
    - 2 (label=threatEmail, subject, message, etc)

1.2.1 Long File Names

- When using long file names for actions on Windows machines, the file names need to be escaped correctly:

```
"actions": [
            "python \"c:\\Program
               Files\\RealNetworks\\SAFR\\ares\\test.py\""
          ]
```

- Within the SAFR Actions GUI the same entry appears as follows:


SAFRActions.config
File Tools
Key
Value
end
23:00
actions
(1 item)
Item 1
python "c:\Program Files\RealNetworks\SAFR\ares\test.py"
reply
(1 item)
message
Script triggered!

2 Actions Relay Event Service (ARES)

ARES is a cross-platform Java application that acts as SAFR Platform event listener that dispatches configured actions (macros) in response to events. The recommended Java version is 9.0.4 or later. ARES can provide replies on any event to be handled by the client originating the event and is normally installed as a service by either SAFR Platform or SAFR Desktop installers. It is constantly active and is automatically started by the operating system on power-up.

2.1 ARES Installation Locations

- For Windows: `C:\Program Files\RealNetworks\SAFR\ares`
- For macOS: `/Library/RealNetworks/SAFR/ares`
- For Linux: `/opt/RealNetworks/SAFR/ares`

2.2 Command Line Start

```
java -jar Ares.jar
```

Command line supports the following options:

```
-u <UserId>       - provides RealCV account User Id
-p <Password>     - provides RealCV account password
-q                - turns on quiet mode which suppress most console output
```

Command line UserId/Password override those configured in SAFRActions.config.

2.3 Re-configuration

- ARES dynamically applies any changes to config file without restarting:
  - ARES monitors config file for any changes.
    - ARES examines config file for modifications every 2 seconds
- When a change is noticed, ARES reads and reconfigure atomically (event polling is to suspend briefly and then promptly resumed after reconfiguartion).
- Reconfiguration action is indicated in the log:

  ```
  --- RECONFIGURED at <date>
  ```

2.4 Console Output

- At start, ARES displays any errors or warning based on contents of the config file.
- ARES displays all received events, triggered actions, and replies issued unless it was given -q (quiet) option at start.

Tip: In the Mac terminal or in the Windows Cygwin shell, the `tail -f ares.log` command is a convenient way to monitor the SAFR Action service in real time.

3 SAFRActions.config

The SAFRActions.config file defines which events will trigger specified actions. You can also specify additional condition constraints before the action(s) will trigger. It also contains basic configuration information so that ARES can communicate with other SAFR components, such as the Event Archive.

3.1 SAFRActions.config JSON Schema

```
{
   environment : "string",
                 <optional,
                 - values: "LOCAL", "DEV", "INT2", "PROD", "Custom"
                 - if not specified assumed PROD >
   eventServer : "string",
                 <optional,
                 - required in case of Custom environment
                 - only affects Custom environment>
   replyServer : "string",
                 <optional,
                 - only affects Custom environment>
   coviServer : "string",
                 <optional,
                 - only affects Custom environment>
   reportServer : "string",
                 <optional,
                 - only affects Custom environment>
   configServer : "string",
                 <optional, "https://cvos.real.com" for cloud environment
                 - if specified config is retrieved from the cloud using
                    the
                  following address: <configServer>/obj/ares/<aresId> >
   userId :      "string", <optional>
   userPwd :     "string", <optional>

   directory :   "string", <required>
   site :        "string", <optional>
   source :      "string", <optional>

   aresId :      "string", <optional>

   maxEventLatency: <long>,  <optional, in milliseconds, default = 8000>

   rules: [
      {
        event : {
          type: [ "string", ... , "string" ],
                 <optional, values=(person, badge, action or object),
                    default = all>
          personType: [ "string", ... , "string" ],
                 <optional, default = all, "" = no personType>
          personTags: [
                 [ "string", ... , "string" ],
                 ...
                 [ "string", ... , "string" ]
```

```
]
       <optional, default = all>
tagType: [ "string", ... , "string" ]
       <optional, values=(april), default = all, "" = no
          tagType>
tagId: [ "string", ... , "string" ],
       <optional, values=(Ids of tagType) default = all, "" =
          no tagId>
actionType: [ "string", ... , "string" ],
       <optional, values=(smileToActivate) default = all, "" =
          no actionType>
actionId: [ "string", ... , "string" ],
       <optional, default = all, "" = no actionId>
name: [ "string", ... , "string" ],
       <optional, default = all, "" = no name>
company: [ "string", ... , "string" ],
       <optional, default = all, "" = no company>
moniker: [ "string", ... , "string" ],
       <optional, default = all, "" = no moniker>
personId: [ "string", ... , "string" ],
       <optional, default = all, "" = no personId>
hasPersonId: <boolean>,
       <optional, default = all>
hasName: <boolean>,
       <optional, default = all>
hasMoniker: <boolean>,
       <optional, default = all>
hasRootEventId: <boolean>,
       <optional, default = all>
gender: [ "string", ... , "string" ],
       <optional, default = all>
age: [
       <optional, default = all>
   {
      min: <float>,
      max: <float>
   },
   ...
],
smile: <boolean>,
       <optional, default = all>
avgSentiment: [
       <optional, default = all>
   {
      min: <float>,
      max: <float>
   },
   ...
],
liveness: {
       <optional, default = all>
   min: <float>,
   max: <float>
},
```

```
      livenessConfirmed: <boolean>,
            <optional, default = all>
      mask: <boolean>,
            <optional, default = all>
      similarityScore: {
            <optional, default = all>
         min: <float>,
         max: <float>
      },
      occlusion: {
            <optional, default = all>
         min: <float>,
         max: <float>
      },
      site: "string",
            <optional if specified at the root>
      source: "string",
            <optional if specified at the root>
      idClass: [ "string", ... , "string" ],
            <optional, default = all, "" = no idClass>
      directGazeDuration: {
            <optional, default = all>
         min: <long>,
         max: <long>
      }
      objectType: [ "string", ... , "string" ]
            <optional, default = all, "" = no objectType>
      objectId: [ "string", ... , "string" ],
            <optional, default = all, "" = no objectId>
   }
   triggers : [
      {
         triggerId : "string",
             <optional>
         daysOfWeek: ["Mon","Tue","Wed","Thu","Fri","Sat","Sun"],
             <optional, default = all>
         timesOfDay: [
             <optional, default = all>
            {
               start: "11:00",                  <required>
               end: "17:00"                     <required>
            },
            ...
         ],
         actions: [
             <required - can be empty (no actions)>
            "string",
            ...
         ],
         reply: {
             <optional, default = no reply>
            "replyDelay": long,
                 <optional, in milliseconds, default = 0>
            "message": "string",
```

```
                    <optional, default = no message>
               "disposition": double,
                    <optional, range [-1 .. 1], default = 1>
               "tags": [ "tag1", ... "tagN" ]
                    <optional, default = no tags>
          },
          conditionalReply: [
              <optional, default = no conditional reply>
              {
                   "actionResponse": [ integer, ..., integer ],
                        <required>
                   "replyDelay": long,
                        <optional, in milliseconds, default = 0>
                   "message": "string",
                        <optional, default = no message>
                   "disposition": double,
                        <optional, range [-1 .. 1], default = 1>
                   "tags": [ "tag1", ... "tagN" ]
                        <optional, default = no tags>
              }
              ...
          ],
       },
       ...
    ],
    excludeDates : [
              <optional, default = none>
              "7/4",
              "12/25",
              "4/10/2017",
              ...
    ]
   }
   ...
],
noTriggerReply: {
              <optional, default = no reply>
   "replyDelay": long,
                    <optional, in milliseconds, default = 0>
   "message": "string",
                    <optional, default = no message>
   "disposition": double,
                    <optional, range [-1 .. 1], default = -1>
   "tags": [ "tag1", ... "tagN" ]
                    <optional, default = no tags>
},
nFactorDef: [
   {
      "name": string,
         <required>
      "failOnMismatch": string,
         <optional: "delayed"/"immediate"/"none", default = "delayed">
      "maxDelay": <milliseconds>,
         <optional, default = 60000 (1min)>
```

```
         "factors": [
            "<factor_name>|<factor_value>",
            ...
         ],
         "actions": [
            "<action_command>",
            ...
         ]
      },
      ...
   ],
   emailDef: [
      {
         "label": string,
            <required>
         "recipients": [ "recipient1", ... "recipientN" ],
            <required, escape sequences can be used>
         "subject": string,
            <required, escape sequences can be used>
         "cc": [ "cc1", ... "ccN" ],
            <optional, escape sequences can be used>
         "bcc": [ "bcc1", ... "bccN" ],
            <optional, escape sequences can be used>
         "message": string,
            <optional, escape sequences can be used>
         "attachments": [ "attachment1", ... "attachmentN" ],
            <optional, escape sequences can be used
             http://, https://, cvos:// url schemes are supported>
      },
      ...
   ]
   smsDef: [
      {
         "label": string,
            <required>
         "recipients": [ "recipient1", ... "recipientN" ],
            <required, escape sequences can be used, phone numbers using
               the the E.164 format required>
         "maxPrice": string,
            <optional>
         "message": string,
            <optional, escape sequences can be used>
      },
      ...
   ],
}
```

- Events that are older than maxEventLatency will be ignored. Event time is defined as the difference between the time the event was generated - as measured by the SAFR Cloud (or machine Platform is running) and the time the event is processed – as measured on the machine the SAFR Actions app is running.

3.2 rules

3.2.1 event

- For rules.events that allow arrays, the new event must contain all the specified array elements to match. For example, if a config file specified rules.events.personType as follows:

```
personType: [
    "staff",
    "admin",
    "guest"
],
```

  Then the new event's personTags array would have to have all 3 specified personTypes for it to match the rule.

- personTags: all elements in one of sub-arrays need to exist in event's personTags array to match the rule.

3.2.2 trigger

- Event (id) can trigger actions only once (albeit multiple triggers can be activated simultaneusly).
- Event (id) can trigger replies only once per reply context (triggered, notTriggered). Multiply replies can be triggered simultaneously (one reply per triggered action).
- triggerId - ID Unique within the triggers array used in rare case where you want only 1 trigger to fire. If triggerId is same on 2 or more, only 1st of all matching get triggered.
- Useful if date filters are overlapping and during overlap times only wish to actions from single trigger.

3.2.3 conditionalReply and reply

- disposition refers to how the reply should be perceived by the recipient:
  - Replies with disposition in range [-1 .. 0 > are interpreted as negative replies and can thus be expected to be presented (color, sound, voice) in manner consistent with rejection.
    - Value of 0 is a neutral reply and can thus be expected to be presented in a neutral manner (color, sound, voice).
    - Replies with disposition in range <0 .. 1] are interpreted as positive replies and can thus be expected to be presented (color, sound, voice) in manner consistent with acceptance.
- When conditional reply is specified, non-conditional reply is used only as catch-all if none of the action response codes match.
- When conditional reply is specified, execution of the FIRST action in trigger will occur in blocking manner to enable retrieval of the response code from that FIRST action.
  - If any other actions are specified, they will be performed in non-blocking manner and their response codes will not be retrieved or used.
- When conditional reply is not specified, execution of all actions will occur in non-blocking manner.
- A reply is generated as follows:
  - One or more matching conditionalReply entries are sent
  - In addition, either the reply or noTriggerReply is sent
- URL used to post the reply: `<replyServer>/stream/reply.<Base64(event Id)>`
  - By default the reply is posted to the CVOS server (replyServer)
  - POST is a file of the following format.
  - The reply object (JSON file) can be obtained by querying the CVOS server after some delay after the event was fired

3.2.4 actions

- Each action is a command string that will be executed.
- Commands are executed asynchronously unless conditionalReply is set
- If conditionalReply is set, the first command is executed synchronously.
- Some Windows programs (particular Windows programs that do not have a message pump) may not run in background and block until the command returns.
- If multiple actions are defined, each action is executed in sequence.
- For information on the syntax for emails, see Email Actions below.
- For information on the syntax for SMS notifications, see SMS Actions below.

3.3 Action and Reply Message Escape Sequences

```
#N - name
#F - first name (name prefix up to first white-space)
#U - surname (name postfix: staring after first white-space sequence to
   the end of name string)
#T - person type
#S - source
#I - site
#D - person id
#R - root person id
#E - person external id
#G - gender
#A - age       (###)
#M - sentiment (#.##)
#L - smile     (true/false)
#V - event type
#v - event id
#B - tag type
#C - action type
#b - tag id
#c - action id
#k - direction id
#s - event start time (milliseconds since epoch)
#r - event start date/time (local time)
#p - validation phone
#e - validation email
#H - home location
#t - personTags (comma separate list of personTags)
#O - company
#m - moniker
#<d>m - moniker substring (delimited by white-space)
        indexed by single decimal digit 0-9 .  E.g.:  #0m or #3m
#l - similarityScore (#.####)
#a - idClass
#Z - directGazeDuration
#o - objectType
#d - objectId
#u - occlusion (#.##)
#i - liveness (#.##)
#n - livenessConfirmed (true/false)
#z - mask (true/false)
```

3.4 N-factor Actions

- nFactor actions are started via internal @nFactorStart actions within the standard trigger actions array:

```
{
   triggerId : "string",
   ...
   actions: [
      "@nFactorStart <name>",
      ...
   ],
   reply: {
      ...
   },
   conditionalReply: [
      ...
   ]
}
```

When the action starts, the following occurs:

- @nFactorStart actions are first resolved for escape sequences
- factors (names and values) defined in the corresponding nFactorDef are also resolved for escape sequences
- actions defined in the corresponding nFactorDef are also resolved for escape sequences
- eventStartTime is retrieved from the triggering event

Response codes for nFactorStart actions:

- 0 = nFactor monitoring for action started successfully

nFactorStart-ed actions are resolved via nFactorResolve commands. When all factors needed for the actions are resolved, actions are executed:

```
{
   triggerId : "string",
   ...
   actions: [
      "@nFactorResolve <name> <factor_name>|<factor_value>",
      ...
   ],
   reply: {
      ...
   },
   conditionalReply: [
      ...
   ]
}
```

- At the time of resolving the following occurs:
  - @nFactorResolve actions are first resolved for escape sequences.
  - Each factor can be resolved by at most one not yet resolved factor requirement.
- Response codes for nFactorResolve actions:
  - 0 = resolved last unresolved factor
    - Executed action response supersedes
  - >=1 resolved other than last unresolved factor
  - -1 = no matching <Site>/<Source>/<name>

- -2 = <mismatched factor - ignored since failOnMismatch = none>
- -3 = <matches but already resolved>
- -4 = <matches but too late to resolve>
- -5 = <mismatched factor - error since failOnMismatched = delayed/immediate>
- -6 = unknown (i.e. not defined in nFactorDef) factor_name

- @nFactorStartOrResolve combines starting and resolving into one action. It's usually used for generating pseudo events from monikers.

```
{
   triggerId : "string",
   ...
   actions: [
      "@nFactorStartOrResolve <name> <factor_name>|<factor_value>",
      ...
   ],
   reply: {
      ...
   },
   conditionalReply: [
      ...
   ]
}
```

@personEventFromMoniker action generates a pseudo person event from moniker created by combining all the resolved factor values (separated by space) in the order listed in factors array. The generated event is of type *person* which is populated with the meta-data of person with moniker matching the assembled moniker value.

```
{
   nFactorDef : [ {
      factors : [
            "moniker|**",
            "moniker|1**",
            "moniker|2**",
            "moniker|3**"
      ],


      actions : [
            "@personEventFromMoniker"
      ]
      }
   ]
}
```

3.5 Email Actions

To send emails using actions, you must do the following:

1. Obtain an SMTP server account that you can use to send emails.
2. Configure SAFR so that it's ready to use your SMTP server account to send emails. You can do this from the Status page of the Web Console. On Windows machines, you can also do this via **Tools -> Configure Email Server** in SAFR Actions.

3. Configure the emailDef section of the SAFRActions.config, as described below. Note that your emailDef section can define multiple emails, each one being identified by the `label` field.

```
emailDef: [
    {
        "label": string,
            <required>
        "recipients": [ "recipient1", ... "recipientN" ],
            <required, escape sequences can be used>
        "subject": string,
            <required, escape sequences can be used>
        "cc": [ "cc1", ... "ccN" ],
            <optional, escape sequences can be used>
        "bcc": [ "bcc1", ... "bccN" ],
            <optional, escape sequences can be used>
        "message": string,
            <optional, escape sequences can be used>
        "attachments": [ "attachment1", ... "attachmentN" ],
            <optional, escape sequences can be used
                http://, https://, cvos:// url schemes are supported>
    },
]
```

- **label**: The label used to identify this particular email.
- **recipients**: One or more email addresses where the email will be sent.
- **subject**: The text that will appear in the email's subject line.
- **cc**: List of email addresses that will be cc'ed on the email.
- **bcc**: List of email addresses that will be bcc'ed on the email.
- **message**: The text that will be the body of the email.
- **attachments**: The location of any attachments you want to attach to the email.

4. In the `actions` field of SAFRActions.config, enter a string with the following syntax: "@emailSend <label>", where <label> = the label of whichever email within your SAFRActions.config that you want to use.

3.6 SMS Actions

To use Short Message Service (SMS) notifications within actions, you must do the following:

1. Obtain an AWS account which is configured for your region so it can send SMS messages.
2. Configure SAFR so that it's ready to use your AWS account to send SMS notifications. You can do this from the Status page of the Web Console. On Windows machines, you can also do this via **Tools -> Configure SMS Sender** in SAFR Actions.
3. Configure the smsDef section of the SAFRActions.config, as described below. Note that your smsDef section can define multiple SMS messages, each one being identified by the `label` field.

```
smsDef: [
   {
      "label": string,
         <required>
      "recipients": [ "recipient1", ... "recipientN" ],
         <required, escape sequences can be used, phone numbers using the
            the E.164 format required>
      "maxPrice": string,
         <optional>
```

```
      "message": string,
         <optional, escape sequences can be used>
   },
]
```

- **label**: The label used to identify this particular SMS message.
- **recipients**: The list of recipients to receive the SMS message, formatted using the E.164 format. (e.g. +2065551313)
- **maxPrice**: The maximum amount in USD that you are willing to spend to send the SMS message. Amazon SNS will not send the message if it determines that doing so would incur a cost that exceeds the maximum price. See the description of the `AWS.SNS.SMS.MaxPrice` attribute here for more information about this field.
- **message**: The text message to be sent.

4. In the `actions` field of SAFRActions.config, enter a string with the following syntax: "@smsSend <label>", where <label> = the label of whichever SNS message within your SAFRActions.config that you want to use.