



SAFR Actions

Event to Action Tool

SAFR Mobile Application Platform User Guide

Documentation Version = 1.0

Publish Date = December 15, 2022

Copyright © 2022 RealNetworks, Inc. All rights reserved.

SAFR® is a trademark of RealNetworks, Inc. Patents pending.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

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.
- **SAFR Actions:** Only available on macOS and Windows. SAFR Actions is a GUI tool that makes editing the SAFRActions.config file much easier. It presents the JSON information of the config file in a visual and easy to understand manner and offers drop-down menus so you can quickly and easily see what values are available and valid. SAFR Actions makes the JSON element hierarchies easy to understand, and ensures that your changes will validate against the SAFRActions.config JSON schema.

1.2 SAFRActions.config Overview

The SAFRActions.config file stores the rules for triggering actions. Rules are made up of events (conditions) and the respective triggers (actions). Below is a high level example of a SAFR Actions config file.

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

- rules
 - 1 or more rules can be included.
 - When an event (person detection, badge detection, smile) occurs each rule is checked to see if it matches the event
 - A rule matches an 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 time of day conditions match (unless triggerId is identical in which case only the 1st is fired)
 - Each trigger has one or more actions.
 - Actions are either:
 - shell command or a batch/shell script to be executed
 - Email send command in syntax of: @emailSend <value of emailDef.label>
 - SMS send command in syntax of: @smsSend <value of smsDef.label>
 - All actions are run asynchronously unless a conditionalReply is specified in which case the 1st rule is run synchronously (and the return code of that used for the conditionalReply) and 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 SMS message attributes (message, maxPrice)

1.2.1 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 #l - #S.
 - ...

Log all events to CSV and send one type of email for known person and another for threat.

- 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\\" ..."
- emailDef
 - 1 (label=knownEmail, subject, message, etc)
 - 2 (label=threatEmail, subject, message, etc)

Notes:

- If editing config file, escape backslash or quotes with another backslash. In SAFR Actions App no escaping is needed)
- File 'log_event.bat' should be placed in C:\Program Files\RealNetworks\SAFR\ares\scripts or /Library/RealNetworks/SAFR/ares/scripts

1.3 Quoting and Escaping Arguments in SAFRActions.config

When using any arguments with spaces such as file names, the argument must be quoted, and quotes need to be escaped using a backslash (e.g. \") within the SAFRActions.config file. Backslashes should also be escaped with another backslash (e.g. \\).

For example:

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

Within the SAFR Actions GUI the same entry appears without the escape (\) as follows:

▼ actions	(1 item)
Item 1	python C:\\Program Files\\RealNetworks\\SAFR\\ares\\test.py

Also, any arguments that may be empty should be quoted. For example, including age and gender in the argument list must be quoted because they may be empty for some events.

For example:

```
"actions": [
  "python \\"#A\\" \\"#G\\""
```

In SAFR Actions GUI, this appears without the escape characters as in: python "#A" "#G"

▼ actions	(1 item)
Item 1	python "#A" "#G"

2 Actions Relay Event Service (ARES)

ARES is a cross-platform Java application that acts as an event listener which dispatches configured actions (i.e. macros) in response to events. The recommended Java version is 9.0.4 or later. ARES can dispatch replies on any event detected by your SAFR system 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 Options

You can manually start your ARES service by running the following command.

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

The command line supports the following options:

- **-u <UserId>**: Provides the SAFR account user ID. The user ID that is passed into ARES via the command line overrides whatever user ID is configured within *SAFRActions.config*.
- **-p <Password>**: Provides the SAFR account password. The password that is passed into ARES via the command line overrides whatever password is configured within *SAFRActions.config*.
- **-s**: Saves the user ID and/or password to *SAFRActions.config*. If a password is saved, it's encrypted before it's saved.
- **-q**: Turns on quiet mode, which suppresses most console output.

Linux users should use the **-s** command line option to save their **password** to *SAFRActions.config* so it will be stored encrypted. Windows users should either use the **-s** command line option or the SAFR Actions GUI tool to save their **password** to *SAFRActions.config* so it will be stored encrypted.

2.3 Reconfiguration

- ARES dynamically applies any changes to the *SAFRActions.config* file without restarting:
- ARES examines the *SAFRActions.config* file every 2 seconds for any changes.
- When a change is detected, ARES reads the change and automatically reconfigures itself accordingly. (Event polling is suspended briefly and then promptly resumed after reconfiguration is complete.)
- The reconfiguration action is indicated in the log:

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

2.4 Console Output

- When started, ARES displays any errors or warnings based on the contents of the *SAFRActions.config* file.
- ARES displays all received events, triggered actions, and replies issued unless it was given the -**q** (quiet) option when it was started.

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.int2.real.com" for integration environment
    "https://cvos.real.com" for production environment
    - if specified Config is retrieved from Cloud using
    following address: <configServer>/obj/ares/<aresId> >
  userId : "string", <optional>
  userPwd : "string", <optional, encrypted or open text>
  directory : "string", <required>
  site : "string", <optional>
  source : "string", <optional>
  aresId : "string", <optional>
  maxEventLatency: <long>, <optional, in milliseconds, default = 8000>
  logActionResponses: <bool>, <optional, default = false>
  rules: [
    {
      event : {
        type: [ "string", ... , "string" ],
          <optional, values=(person, tag, action or recognizedObject),
          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>
directionId: [ "string", ... , "string" <"left", "right", "up", "down"> ],
    <optional, default = all, "" = no directionId>
ended: <boolean>,
    <optional, default = false>
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>
hasFaceId: <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>
    accessClearance: [ "string", ... , "string" ],
        <optional, default = all, "" = no company>
    accessClearanceLevel: {
        <optional, default = all>
        min: <long>,
        max: <long>
    }
    accessCardId: [ "string", ... , "string" ],
        <optional, default = all, "" = no company>
    accessFacilityId: [ "string", ... , "string" ],
        <optional, default = all, "" = no company>
    accessCardFormat: [ "string", ... , "string" ],
        <optional, default = all, "" = no company>
    groups: [
        [ "string", ... , "string" ],
        ...
        [ "string", ... , "string" ]
    ]
        <optional, default = all>
    srcUserId: [ "string", ... , "string" ],
        <optional, default = all, "" = no company>
    statusType: [ "string", ... , "string" ],
        <optional, default = all, "" = no company>
    feed: [ "string", ... , "string" ],
        <optional, default = all, "" = no company>
    processor: [ "string", ... , "string" ],
        <optional, default = all, "" = no company>
    mode: [ "string", ... , "string" ],
        <optional, default = all, "" = no company>
    longitude: [
        <optional, default = all>
        {
            min: <float>,
            max: <float>
        }
    ],
    ...

```

```

],
latitude: [
  <optional, default = all>
  {
    min: <float>,
    max: <float>
  },
  ...
],
altitude: [
  <optional, default = all>
  {
    min: <float>,
    max: <float>
  },
  ...
],
srcHeading: [
  <optional, default = all>
  {
    min: <float>,
    max: <float>
  },
  ...
],
srcHeadingPitch: [
  <optional, default = all>
  {
    min: <float>,
    max: <float>
  },
  ...
],
srcOrientation: [
  <optional, default = all>
  {
    min: <float>,
    max: <float>
  },
  ...
],
srcOrientationPitch: [
  <optional, default = all>
  {
    min: <float>,
    max: <float>
  },
  ...
],
srcSpeed: [
  <optional, default = all>
  {
    min: <float>,
    max: <float>
  },
  ...
]
}
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",
  ...
],
triggerFrequencyLimit : {
  <optional, default = unlimited>
  "minSeparationInterval" : long,
    <optional, in milliseconds, default = 0>,
  "spanRootPersonIds": bool,

```



```

    },
    ...
  ],
  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>
    },
    ...
  ],
}

```

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 **personType** array would have to have all 3 specified **personTypes** for it to match the rule.

- **personTags**: All elements in one of the sub-arrays need to exist in the event's **personTags** array to match the rule.
- **ended**: Indicates if an event has completed. It defaults to false for all events, but it's set to true when an event completes.
- 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.

3.2.2 trigger

- Event (id) can trigger actions only once (albeit multiple triggers can be activated simultaneously).
- Event (id) can trigger replies only once per reply context (triggered, notTriggered). Multiple 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.

Structure of the reply:

```

{
  triggered : boolean,
               <required - either true or false>
  replyTime : <epoch time>,
               <required - milliseconds since start of epoch>
  eventStartTime: <epoch time>,
               <required, indicates time of the start of the event being replied to>
  replies: [
    <optional, default = no replies>
    {
      replyDelay : long,
                   <optional, in milliseconds, default = 0>
      message: "string",
                <optional, default = no message>
      disposition: double,
                   <optional - [-1 .. 1]>
      tags: [ "tag1", ..."tagN" ],
              <optional, default = no tags>
      actions: [ "action1", ..."actionN" ]
                 <optional, default = no actions>
    }
  ]
}

```

- actions returns the values passed in the actions list

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 (particularly 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 of passing arguments to scripts, see Passing Script Arguments below.
- 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 to the end of name)
#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
#h - Base64(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)
#X - access clearance
#x - access clearance level
```



```
#Q - access card id
#q - access facility id
#f - access card format
#g - groups (comma separate list of groups)
#j - source user id
#w - status type
#K - feed
#P - processor
#W - mode
#1 - longitude
#2 - latitude
#3 - altitude
#4 - source heading
#5 - source heading pitch
#6 - source orientation
#7 - source orientation pitch
#8 - source speed
#9 - source position time (milliseconds since epoch)
#0 - source position date/time (local time)
```

The `\` character is used as an escape character. Thus, if you wanted the string "Welcome, Ms. #N!" to appear as a reply, you would need to have the following in your config file:

```
"reply" : {
  "message" : "Welcome, Ms. \#N!"
},
```

3.3.1 Passing Script Arguments

When passing in simple arguments, you simply pass in the arguments with white space separating them from the script and from each other. For example:

```
"reply" : {
  "python ./scripts/write_result.py Jamie"
},
```

When a single argument contains whitespace, or when you want to use one of the `#<letter>` tokens listed above on a Windows machine, use escaped `""` characters. For example:

```
"actions": [
  "python ./scripts/write_result.py \"Jamie Rodriguez\""
],
```

or

```
"actions": [
  "python .\\scripts\\write_result.py \"#N\""
],
```

Note the need to escape the path separator character `\` in the Windows example above.

When you want to use one of the #<letter> tokens listed above on a Linux machine, use escaped " characters. For example:

```
"actions": [
  "python ./scripts/write_result.py \'#N\'"
],
```

3.4 N-factor Actions

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

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

When the action starts, the following occurs:

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

Response codes for nFactorStart action:

- 0 = nFactor monitoring for action started successfully

nFactorStart-ed action 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 action just as any other action is first resolved for escape sequences.
 - Each factor can resolve at most one not yet resolved factor requirement.
- Response codes for nFactorResolve action:
 - 0 = resolved last unresolved factor
 - Executed action response will supersede
 - >=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 (not defined in nFactorDef) factor_name.

@nFactorStartOrResolve combines starting and resolving into one action. Usually used for generating pseudo events from monikers.

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

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

```
{
  "nFactorDef" : [ {
    "name": "string",
    "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 and macOS 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 and macOS 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. +12065551313)
 - **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.

4 SAFR Actions

SAFR Actions is a GUI tool to aid users in editing the `SAFRActions.config` configuration file. It comes already installed with SAFR Platform and SAFR Desktop.

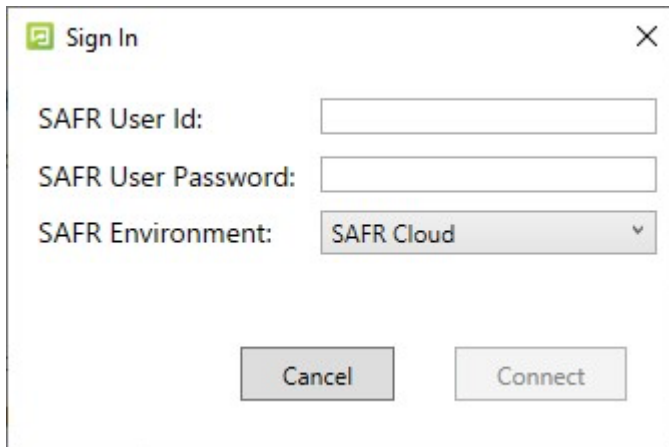
SAFR Actions is generally very light in terms of system resource usage but can be burdensome if the rate of events requiring handling is high (for example, hundreds of cameras with significant activity) and actions scripts are computationally or I/O intensive. However, this is not a common occurrence.

4.1 Configure Email Server

Only available on Windows.

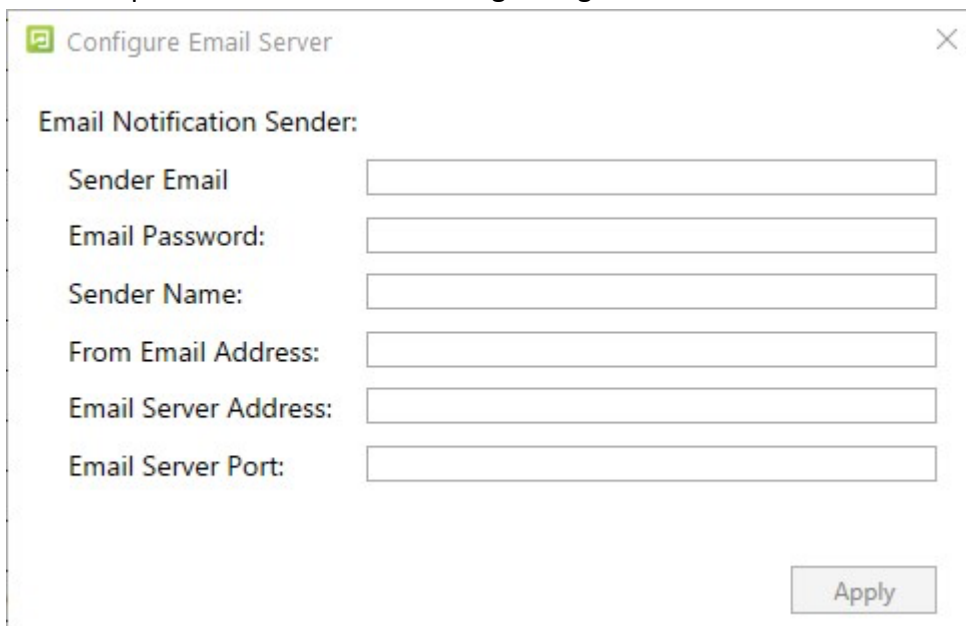
Configure SAFR so that it can use an email server to send email messages by doing the following:

1. Get an SMTP Server account you can use for sending emails.
2. Within the SAFR Actions GUI, select **Configure Email Server...** from the **Tools** drop-down menu at the top.
3. You'll be prompted to enter your SAFR credentials.



A dialog box titled "Sign In" with a close button (X) in the top right corner. It contains three input fields: "SAFR User Id:" with a text box, "SAFR User Password:" with a text box, and "SAFR Environment:" with a dropdown menu currently showing "SAFR Cloud". At the bottom, there are two buttons: "Cancel" and "Connect".

- **SAFR User Id:** Your SAFR account user ID.
 - **SAFR User Password:** Your SAFR account password.
 - **SAFR Environment:** If you're using a cloud license, select *SAFR Cloud* from the drop-down menu. If you're using an on-premises license, select *SAFR Local*.
4. You'll be presented with the following dialogue. Fill out all the fields of the dialogue.



A dialog box titled "Configure Email Server" with a close button (X) in the top right corner. It contains six input fields under the heading "Email Notification Sender:": "Sender Email", "Email Password:", "Sender Name:", "From Email Address:", "Email Server Address:", and "Email Server Port:". At the bottom right, there is an "Apply" button.

- **Sender Email:** The email username of the SMTP account. (e.g. Susan.Johnson@gmail.com)
- **Note:** If you use 2-Step-verification with a gmail account, you need to create and use an App Password on your Google Account. For more information, see <https://support.google.com/mail/answer/185833?hl=en>.

- **Email Password:** The password for the SMTP account.
- **Sender Name:** The display name on the "From" line. (e.g. Susan Johnson)
- **From Email Address:** The email address that will appear on the "From" line. This feature isn't supported by all email servers; if this field isn't used then the *Sender Email* value is used for the "From" line.
- **Email Server Address:** The address of the SMTP email server.
- **Email Server Port:** The email server port. The default port for SMTP is 587.

5. Click **Apply**.

4.2 Configure SMS Sender

Only available on Windows.

To configure SAFR so that it can send short message service (SMS) messages, do the following:

1. Sign up for Amazon SNS (Simple Notification Service), as described on Amazon's site here: <https://aws.amazon.com/sns/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-newcards.sort-order=desc>.
2. Within the SAFR Actions GUI, select **Configure SMS Sender...** from the **Tools** drop-down menu at the top.
3. You'll be prompted to enter your SAFR credentials.

The image shows a 'Sign In' dialog box with the following fields and controls:

- SAFR User Id:** A text input field.
- SAFR User Password:** A password input field.
- SAFR Environment:** A dropdown menu with 'SAFR Cloud' selected.
- Buttons:** 'Cancel' and 'Connect' buttons at the bottom.

- **SAFR User Id:** Your SAFR account user ID.
- **SAFR User Password:** Your SAFR account password.
- **SAFR Environment:** If you're using a cloud license, select *SAFR Cloud* from the drop-down menu. If you're using an on-premises license, select *SAFR Local*.

4. You'll be presented with the following dialogue. Fill out all the fields of the dialogue.

Configure SMS Notifications

SMS Configuration:

SMS Provider: Amazon_SNS

Access Key:

Secret Key:

Region:

Sender Id:

Apply

Send Test Message:

Phone Number:

Message: SAFR Actions Configuration Verification id: 626928

Send

- **SMS Provider:** The SMS provider that you're using. Leave this value at its default: Amazon_SNS.
- **Access Key:** Your Amazon SNS Access Key.
- **Secret Key:** Your Amazon SNS Secret Key.
- **Region:** The region of your Amazon SNS.
- **Sender Id:** The name that will be used to send the SMS messages.
- **Send Test Message:** Configure the test message that will be sent after you finish setting up SMS.
- **Phone Number:** The phone number to which the test message will be sent. It should be in the E.164 format. (e.g. +12065551313)
- **Message:** The text message that will be sent to the phone number specified above.

5. Click **Apply**, then click **Send**

5 SAFR Actions Scripts

3 Python scripts are included with your Actions Relay Event Service (ARES) installation. Each script is designed to work with a different third party relay device that can electronically lock or unlock doors. The scripts are located here:

- On Windows: C:\Program Files\RealNetworks\SAFR\ares\scripts
- On macOS: /Library/RealNetworks/SAFR/ares/scripts
- On Linux: /opt/RealNetworks/SAFR/ares/scripts

To run these scripts from the command line, do the following:

1. If it's not already installed, install the latest version of Python 3.X. The Python installer can be downloaded from here: <https://www.python.org/downloads/>.
2. Run the script in question. **On Windows:**
 1. Open *Command Prompt (Admin)* or *Windows PowerShell (Admin)* by right clicking on the Windows Start menu (located in the bottom left corner of the screen) and selecting the appropriate entry.
 2. Navigate to C:\Program Files\RealNetworks\SAFR\ares\scripts.
 3. Run python <script syntax>.

On Linux:

1. Open Terminal.
2. Run sudo python /opt/RealNetworks/SAFR/ares/scripts/<script syntax>.

On macOS:

1. Open Terminal.
2. Run python /Library/RealNetworks/SAFR/ares/scripts/<script syntax>.

To run these scripts from with SAFR actions, do the following:

1. If it's not already installed, install the latest version of Python 3.X onto the machine where ARES is installed. The Python installer can be downloaded from here: <https://www.python.org/downloads/>.
2. Within the **SAFRActions.config** file, set the **rules** -> **Item X** -> **triggers** -> **Item 1** -> **action** -> **Item 1** key to the following value:
 - On Windows: python ./scripts/<script syntax>
 - On Linux: sudo python ./scripts/<script syntax>
 - On macOS: python ./scripts/<script syntax>

5.1 relay-x410 Script

This script uses the X-410 Relay to lock and unlock doors. The syntax to use the script is as follows: Usage:

```
relay-x410.py <relayHost> <relayNumber>
```

- **relayHost:** The URL of the X-410 relay.
- **relayNumber:** The channel on the X-410 relay to trigger.

Example:

```
relay-x410.py 192.168.1.2 1
```

5.2 relay-numato Script

This script uses a Numato USB Relay to lock and unlock doors. The syntax to use the script is as follows:

```
Usage: relay-numato.py <relayNumber> <delay> [tty.device]
```

- **relayNumber**: The channel on the Numato relay to trigger.
- **delay**: The number of seconds the relay is to be kept closed after triggering. This translates to the number of seconds the door will be unlocked.

Optional arguments:

- **tty.device**: The text telephone (TTY) device to use.

Example:

```
relay-numato.py 0 5 or
```

```
relay-numato.py 0 5 /dev/tty.usbmodem1411
```

5.3 relay-lcus Script

This script uses an LCUS USB relay to lock and unlock doors. The syntax to use the script is as follows:

```
usage: relay-lcus.py [-h] [-p PORT] [-b {9600}] [-r {1,2,3,4}] [-d] [-q] <delay>
```

- **delay**: The number of seconds the relay is to be kept closed after triggering. This translates to the number of seconds the door will be unlocked.

Optional arguments:

- **-h**: Shows the script help message.
- **-p PORT**: Serial port device. (e.g. COM1, /dev/tty.usb-serial, etc.)
- **-b {9600}**: Serial port baud rate.
- **-r {1,2,3,4}**: Relay device number.
- **-d**: Enables DEBUG level logging.
- **-q**: Suppresses non-error output.

5.4 Samples

- Simple: [ares_simple.config](#)
- nFactor: [ares_nFactor.config](#)
- Cloud: [ares_cloud.config](#)
- Smile to Unlock: [ares_smile.config](#)

