

При прохождении лица с одноразовым пропуском в пустое помещение генерировать сигнал тревоги

Обработка событий доступа: *Отслеживать количество людей в помещении и при прохождении лица с одноразовым пропуском в пустое помещение генерировать сигнал тревоги.*

1. Создайте фильтр по метке #ПроходСовершен [[Фильтры](#)].
2. В блоке **Параметры сигнала** выберите **По событию**, в поле **Фильтр по событиям** выберите из раскрывающегося списка созданный на предыдущем этапе фильтр.
3. Скопируйте приведённый ниже код скрипта.

```
import extensions.automation.scripts.AutomationActionScript
import extensions.automation.signals.{AutomationSignal,
NeyrossEventAutomationSignal}
import models.common.{Event, EventExtensions, FunctionalElement}
import models.neyross.Pass
import proto.neyross.PassProto
import utils.common.{EventBuilder, Tags}
import scala.collection.mutable.TreeMap
import scala.concurrent.{ExecutionContext, Future}
import scala.concurrent.duration._
import java.time.OffsetDateTime
import models.pacs.PacsRoomComponent
import slick.basic.DatabaseConfig
import slick.jdbc.JdbcProfile
import akka.actor.ActorRef

class AdvancedRoomControl extends extensions.automation.scripts.
AutomationActionScript {
    private implicit val ec: ExecutionContext = ctx.executionContext
    private implicit val dbConfig: DatabaseConfig[JdbcProfile] = ctx.dbConfig

    private val scheduler = ctx.injector.instanceOf[utils.bootstrap.Scheduler]
    private val eventTransmissionActor: ActorRef = ctx.injectNamedActor(actors.
common.EventTransmissionActor.Name)

    private val rooms = new PacsRoomComponent()
    private val roomIds = Seq(1l, 45l, 123l, 56l)
    private val enter2lastRegularPassTime = TreeMap.empty[String, Option
[OffsetDateTime]]
    private var passCount = 0

    override def init: Future[Unit] = {
        val dbAction = for {
            rooms <- rooms.findInIds(roomIds)
        } yield rooms.flatMap(_.enterAccessPoints)
```

```

ctx.dbRun(dbAction).map(tokens => {
  enter2lastRegularPassTime.addAll(
    tokens.map(token => (token, None))
  )
})
}

private def reportIfUnaccompanied(token: String): Future[Unit] = {
  passCount = passCount + 1
  val currentLastRegularPassTime = enter2lastRegularPassTime(token)
  Future.successful(
    scheduler.executeOnce(10 seconds)({
      if (enter2lastRegularPassTime(token) == currentLastRegularPassTime) {
        val warningEventBuilder = new EventBuilder
        val event = warningEventBuilder.setHeadline("Проход без
сопровождения")
          .setDescription(s"Проход пользователя с одноразовым пропуском
без сопровождения через точку доступа $token")
          .addEventTags(Tags.Alarm, Tags.PACS)
          .build()
        eventTransmissionActor ! event
      }
    })(s"pass_check_$passCount")
  )
}

private def handleAccessTaken(pass: Pass, token: String): Future[Unit] = {
  if (enter2lastRegularPassTime.get(token).nonEmpty) {
    pass.`type` match {
      case Some(PassProto.Pass.Type.REGULAR) =>
        Future.successful(
          enter2lastRegularPassTime.addOne((token, Some(OffsetDateTime.
now())))
        )
      case Some(PassProto.Pass.Type.ONETIME) =>
        enter2lastRegularPassTime(token) match {
          case Some(lastTime) if lastTime.plusSeconds(10).isBefore
            (OffsetDateTime.now()) =>
            reportIfUnaccompanied(token)
          case None =>
            reportIfUnaccompanied(token)
          case _ =>
            Future.unit
        }
      case _ =>
        Future.unit
    }
  }
}

```

```

    } else {
      Future.unit
    }
}

override def onSignal(signal: AutomationSignal): Future[Unit] = {
  signal match {
    case NeyrossEventAutomationSignal(eventDto) =>
      val isAccessTakenEvent = eventDto.event.eventTags.getOrElse(Nil).
contains(Tags.AccessTaken)
      if (isAccessTakenEvent) {
        val passAndTokenOpt = for {
          passExtension: AnyRef <- eventDto.extensions.get(EventExtensions.
EventPass)
          feExtension: AnyRef <- eventDto.extensions.get(EventExtensions.
FunctionalElement)
        } yield (passExtension.asInstanceOf[FunctionalElement],
          feExtension.asInstanceOf[Pass])
        passAndTokenOpt match {
          case Some((pass, token)) =>
            handleAccessTaken(pass, token)
          case _ =>
            Future.failed(new IllegalArgumentException("unable to get pass or
functional element token"))
        }
      } else {
        Future.unit
      }
    case _ =>
      Future.unit
  }
}
new AdvancedRoomControl

```

4. В блоке **Параметры действия** выберите **Пользовательский скрипт** и вставьте код в поле **Скрипт**. Для открытия большого окна редактора, нажмите на кнопку 
5. Нажмите на кнопку **Сохранить задание**.

Если в помещении нет людей, по факту получения события **ПроходСовершен** по пропуску типа [Разовый], будет формироваться тревожное событие «Проход без сопровождения», которое будет отправлено в ленту событий **АРМ НЕЙРОСС Центр**, также его можно будет отследить в отчёте **Журнал событий**.

События (32)



11:31 Проход без сопровождения
10.1.31.94, Платформа НЕЙРОСС

Дата и время	Узел-источник	Элемент-источник	Событие
28.10.20 11:31:53	10.1.31.94, Платформа НЕЙРОСС	10.1.31.94, Платформа НЕЙРОСС	Проход без сопровождения