Уведомление ответственного лица о факте прохода через точку доступа

Часто бывает необходимо информировать о факте прохода/проезда кого-либо, например:

- руководителя — о приходе подчинённого на работу и об отработанном им времени;
- PR-менеджера — о приходе соискателя;
- организатора выставки — о важном посетителе.

> При этом выдача пропуска может осуществляться удалённо, в виде QR-кода или пин-кода, например. Также вы можете настроить доступ по распознаванию лиц [Настройка биометрической верификации | Биометрия по лицам и отпечаткам пальцев, термометрия].

Для решения таких задач мы разработали скрипт «Персональные Telegram уведомления о доступе», который осуществляет рассылку телеграмм-уведомлений. Текст уведомлений настраивается и может содержать следующую информацию:

1. информация о владельце пропуска — любые данные владельца и пропуска, отработанное время:
2. информация о точке доступа, время прохода;
3. фотоматериалы — скриншот с камеры видеонаблюдения, «привязанной» к точке доступа.

> **ПОДСКАЗКА**
>
> Скрипт также может осуществлять информирование самого посетителя [Отправка инструкций и напоминаний посетителю или сотруднику по факту прохода через точку доступа посредством телеграмм-уведомлений]. Под запрос может быть выполнена доработка скрипта с целью информирования по другим каналам связи. Также возможно дополнение уведомления любыми данными и более сложная логика работы скрипта.

Для обеспечения работы скрипта вам потребуется:

1. В форме пропуска указать идентификатор (ID) телеграмм-канала ответственного лица и подключится к каналу НЕЙРОСС БОТ [Настройка формы владельца пропуска].
2. Настроить фильтр со списком точек доступа, по факту прохода через которые требуется уведомлять ответственное лицо [Настройка фильтра].
3. Импортировать задание автоматизации, настроить текст уведомления [Настройка задания автоматизации].

4. При необходимости расчёта времени нахождения на предприятии в течение текущих суток, требуется указать, какие точки доступа являются входом на предприятие, а какие — выходом из него [Добавление меток входа/выхода].

Настройка формы владельца пропуска

Для информирования ответственного лица или самого владельца пропуска необходимо указать идентификаторы телеграмм-каналов данных лиц. Для удобства мы подготовили готовую форму ввода с требуемыми полями.

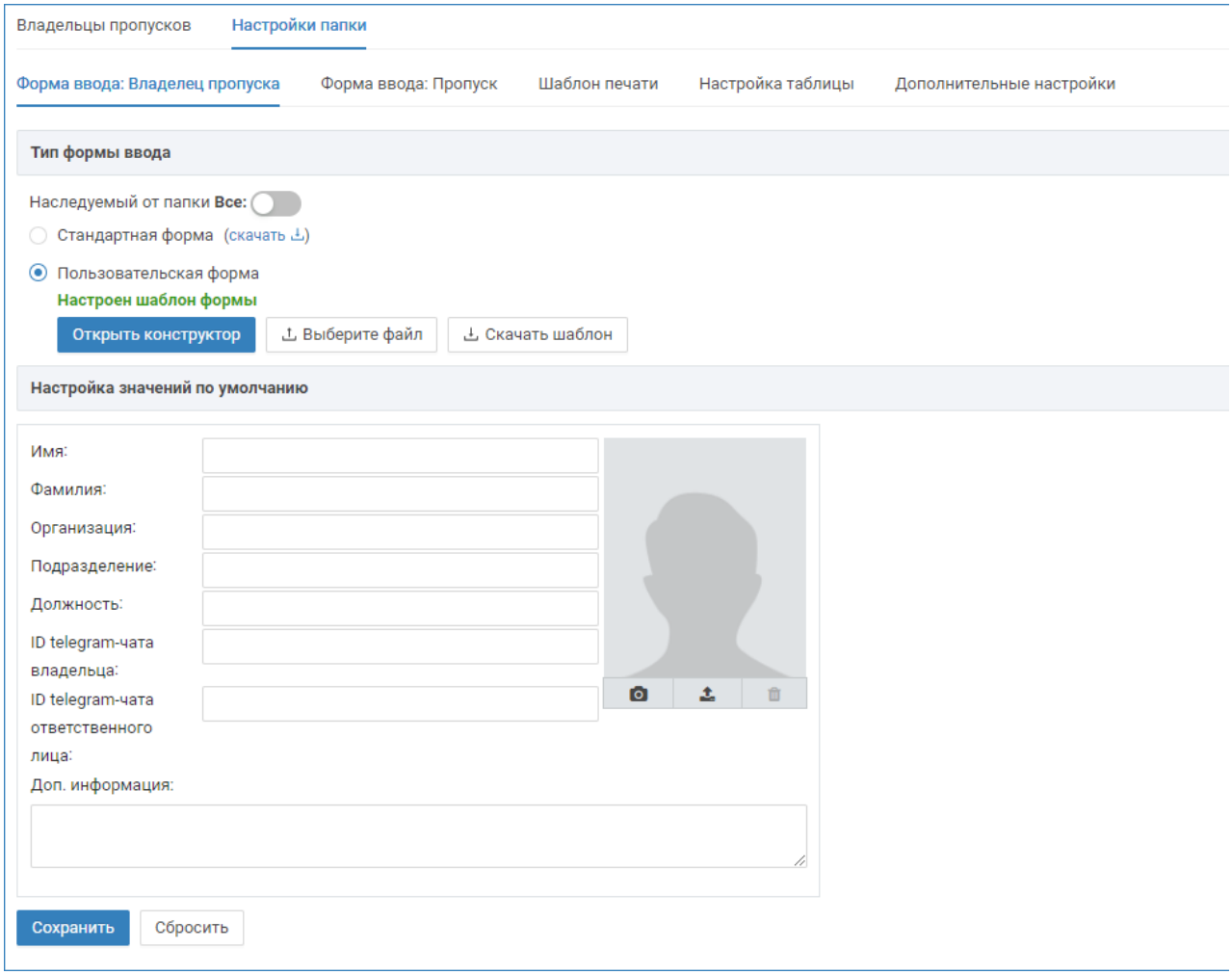
Скопируйте тест в поле ниже и создайте файл с произвольным именем и расширением **JSON**, например: *форма_ввода_владельца_пропуска_с_telegram_полями.json*. Для этого удобно использовать простые текстовые редакторы типа Блокнот или Notepad++. Вы также можете обратиться к специалистам компании ИТРИУМ, мы вышлем подготовленные файлы.

{"type":"object","title":"Title form","keysFields":["person.name","person.surname","person.organization","person.division","person.post","person.properties.neyross:automation:personTelegramChatId","person.properties.neyross:automation:responsibleTelegramChatId","person.photo","person.properties.extra"],"properties":{"person.name":{"type":"string","title":"Имя","format":"single-line","property":{"type":"searchableText","title":"Имя","schema":{"type":"string","format":"single-line"},"dataType":"TEXT","property":"person.name","columnType":"searchableText","searchAlias":"name","propertyType":"STANDART"}},"person.surname":{"type":"string","title":"Фамилия","format":"single-line","property":{"type":"searchableText","title":"Фамилия","schema":{"type":"string","format":"single-line"},"dataType":"TEXT","property":"person.surname","columnType":"searchableText","searchAlias":"surname","propertyType":"STANDART"}},"person.properties.neyross:automation:personTelegramChatId":{"type":"string","title":"ID telegram-чата владельца","format":"single-line","property":{"id":4,"title":"ID telegram-чата владельца","schema":{"type":"string","format":"single-line"},"dataType":"TEXT","property":"person.properties.neyross:automation:personTelegramChatId","propertyType":"USER"}},"person.properties.neyross:automation:responsibleTelegramChatId":{"type":"string","title":"ID telegram-чата ответственного лица","format":"single-line","property":{"id":3,"title":"ID telegram-чата ответственного лица","schema":{"type":"string","format":"single-line"},"dataType":"TEXT","property":"person.properties.neyross:automation:responsibleTelegramChatId","propertyType":"USER"}},"person.photo":{"title":"Фотография","type":"string","format":"photo","property":{"propertyType":"STANDART","property":"person.photo","title":"Фотография","columnType":"photo","type":"photo","dataType":"PHOTO","schema":{"type":"string","format":"photo"}}},"person.post":{"title":"Должность","type":"string","format":"glossary","glossaryType":"post","property":{"propertyType":"STANDART","property":"person.post","title":"Должность","columnType":"searchableInGlossary","searchAlias":"post","type":"searchableInGlossary","dataType":"GLOSSARY","schema":{"type":"string","format":"glossary","glossaryType":"post"}}},"person.organization":{"title":"Организация","type":"string","format":"glossary","glossaryType":"organization","property":{"propertyType":"STANDART","property":"person.organization","title":"Организация","columnType":"searchableInGlossary","searchAlias":"organization","type":"searchableInGlossary","dataType":"GLOSSARY","schema":{"type":"string","format":"glossary","glossaryType":"

organization"}}},"person.division":{"title":"Подразделение","type":"string","format":"glossary","glossaryType":"division","property":{"propertyType":"STANDART","property":"person.division","title":"Подразделение","columnType":"searchableInGlossary","searchAlias":"division","type":"searchableInGlossary","dataType":"GLOSSARY","schema":{"type":"string","format":"glossary","glossaryType":"division"}}},"person.properties.extra":{"title":"Доп. информация","type":"string","format":"multi-line","property":{"propertyType":"STANDART","property":"person.properties.extra","title":"Доп. информация","dataType":"TEXT","schema":{"type":"string","format":"multi-line"}}}},"description":"Description form"}

В **АРМ НЕЙРОСС Доступ** выберите папку пропусков, выберите **Настройки папки > Форма ввода: владелец пропуска > Пользовательская форма**, укажите путь к подготовленному файлу [Настройка форм ввода данных]. Форма будет загружена и доступна к предпросмотру.

При необходимости, задайте значение по умолчанию — ID Telegram-чата отв. лица. Это значение будет задаваться для всех новых пропусков в этой папке, но может быть изменено.



⚠ Если идентификатор канала не указан, рассылка по факту прохода данного лица не производится. Указывайте идентификатор канала отв. лица только в

случае, если требуется уведомлять ответственное лицо. Указывайте идентификатор канала владельца пропуска только в случае, если требуется информировать его об отработанном им времени, либо выдавать инструкции /напоминания.

Чтобы узнать идентификатор своего чата (канала) телеграмм и в дальнейшем получать уведомления, наведите камеру смартфона на QR-код ниже или перейдите по ссылке https://t.me/neyross_bot и подключитесь к каналу НЕЙРОСС БОТ. Рекомендуем распечатать и разместить этот QR-код в помещении бюро пропусков. Также по запросу мы можем поместить его в форму ввода данных владельца пропуска.

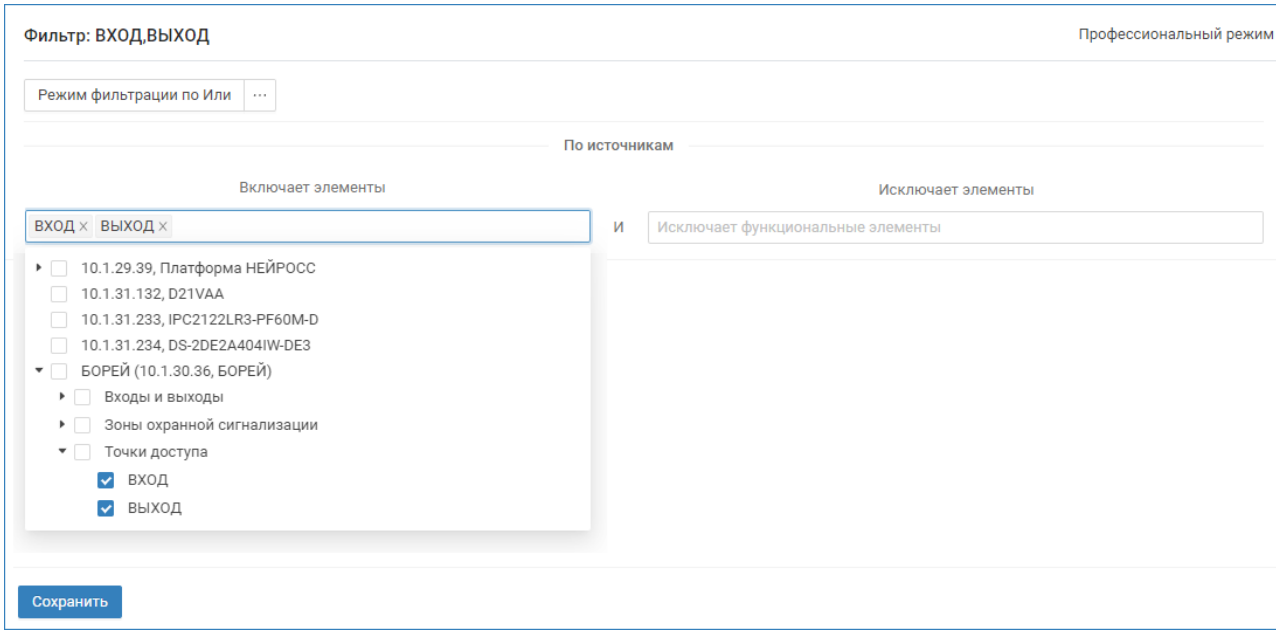
После перехода по ссылке подключитесь к каналу НЕЙРОСС Бот. Вы получите сообщение вида:

Ваш ID: 1055492440
User name: логин
Имя: имя
Фамилия: фамилия

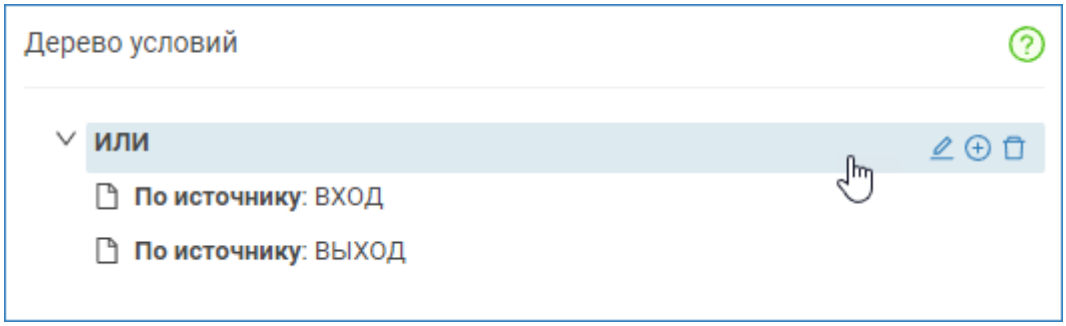
Где цифры между стрелками — идентификатор вашего канала. На примере это 1055492440. Этот идентификатор и должен быть указан в форме пропуска.

Настройка фильтра

Перейдите в раздел **Фильтры** и создайте простой фильтр [Фильтры]. В блоке По источникам в поле **Включает элементы** выберите из раскрывающегося списка точки доступа, событие прохода по которым будут обрабатываться скриптом рассылки уведомлений. Количество точек доступа не ограничено. Вы можете указать точки доступа нескольких контроллеров.



Либо вы также можете воспользоваться профессиональным режимом и задать перечень точек доступа.



Сохраните изменения.

Настройка задания автоматизации

Для удобства мы подготовили готовый код задания автоматизации. Вам потребуется импортировать задание и указать фильтр, подготовленный на шаге [Настройка фильтра].

Скопируйте тест в поле ниже и создайте файл с произвольным именем и расширением **JSON**, например: *Персональные_telegram_уведомления_о_доступе.json*. Для этого удобно использовать простые текстовые редакторы типа Блокнот или Notepad++. Вы также можете обратиться к специалистам компании ИТРИУМ, мы вышлем подготовленные файлы.

```
{
    "signalKey": "neyrossEvent",
    "filterConfig": {
        "eventFilter": 0
    },
    "stats": {
        "lastHandleTimestamp": "2023-05-04T15:58:35+03:00",
        "processedSignals": 2,
```

"failedSignals": 0
},
"enabled": true,
"action": {
"key": "userScript",
"config": {
"body": " \n import beans.common.EventBean\n import beans.pacs.{PassBean, PersonBean}\n import dto.common.EventDto\n import extensions.automation.signals.NeyrossEventAutomationSignal\n import models.common.{Event, EventExtensions, FunctionalElement, FunctionalElementComponent}\n import models.neyross.{Pass, PassComponent, Person}\n import services.neyross.NeyrossEmbedApiService\n import services.pacs.PassService\n import services.telegram.TelegramService\n import utils.common.Tags\n import beans.vmc.EventMediaInfoBean\n import models.media.GetEventMediaInfoRequest\n import scala.collection.mutable.ListBuffer\n\n import java.time.OffsetDateTime\n import extensions.automation.signals.AutomationSignal\n import services.common.MessageTemplateService\n import services.logging.web.{LoggerWithWeb => Logger}\n import slick.basic.DatabaseConfig\n import slick.jdbc.JdbcProfile\n import services.automation.AutomationMessageDataService\n\n import scala.concurrent.{ExecutionContext, Future}\n\n case class WorkTimeDto(workedMinutes: Long,\n idleMinutes: Long)\n\n class TelegramPersonalPacsDataAutomationScript extends extensions.automation.scripts.AutomationActionScript {\n val logger = Logger(\"TelegramPersonalPacsDataAutomationScript\")\n private implicit val ec: ExecutionContext = ctx.executionContext\n private implicit val dbConfig: DatabaseConfig[JdbcProfile] = ctx.dbConfig\n private implicit val neyrossApi: NeyrossEmbedApiService = ctx.injector.instanceOf[NeyrossEmbedApiService]\n private val personBean: PersonBean = ctx.injector.instanceOf[PersonBean]\n private val eventMediaInfoBean: EventMediaInfoBean = ctx.injector.instanceOf[EventMediaInfoBean]\n private val eventBean: EventBean = ctx.injector.instanceOf[EventBean]\n private val passService: PassService = ctx.injector.instanceOf[PassService]\n private val telegramService: TelegramService = ctx.injector.instanceOf[TelegramService]\n private val messageTemplateService: MessageTemplateService = ctx.injector.instanceOf[MessageTemplateService]\n private val functionalElements = new FunctionalElementComponent()\n private val passes = new PassComponent()\n private val messageDataService: AutomationMessageDataService = ctx.injector.instanceOf[AutomationMessageDataService]\n\n private val responsibleChatIdPropertyKey = \"neyross:automation:responsibleTelegramChatId\"\n private val personalChatIdPropertyKey = \"neyross:automation:personTelegramChatId\"\n private val workTimeTemplateString = \"${workTime}\"\n private val greetingTemplateString = \"${greeting}\"\n\n // @parameter { \"key\": \"notifyResponsible\", \"type\": \"boolean\", \"title\": \"Информировать ответственное лицо\", \"description\": \"При наличии у владельца пропуска дополнительного свойства с ключом neyross:automation:responsibleTelegramChatId\" }\n val notifyResponsible = true\n\n // @parameter { \"key\": \"responsibleMessageTemplate\", \"type\": \"string\", \"title\": \"Шаблон сообщения для

ответственного лица\", \"description\": \"Помимо стандартных полей, вы можете использовать дополнительно: workTime - подсчёт рабочего времени за сутки (для событий прохода через точки доступа, помеченные меткой ТочкаДоступаВыход)\" }\n val responsibleMessageTemplate = \"${person.surname} ${person.name}, проход через '${element.name}' совершён. ${workTime}\"\n\n // not implemeted now\n // parameter { \"key\": \"snapshotsForResponsible\", \"type\": \"boolean\", \"title\": \"Отправлять кадры связанных медиаисточников для ответственного лица\", \"description\": \"В случае, если к точке доступа, через которую осуществляется проход, привязаны медиаисточники, в сообщении будут отправлены кадры с данных медиаисточников в момент прохода\" }\n val snapshotsForResponsible = false\n\n // @parameter { \"key\": \"notifyPerson\", \"type\": \"boolean\", \"title\": \"Информировать владельца пропуска\", \"description\": \"при наличии у владельца пропуска дополнительного свойства с ключом neyross:automation:personTelegramChatId)\" }\n val notifyPerson = true\n\n // @parameter { \"key\": \"personMessageTemplate\", \"type\": \"string\", \"title\": \"Шаблон сообщения для владельца пропуска\", \"description\": \"Помимо стандартных полей, вы можете использовать два дополнительных: greering - персонализированное приветствие / прощание, workTime - подсчёт рабочего времени за сутки (для событий прохода через точки доступа, помеченные меткой ТочкаДоступаВыход)\" }\n val personMessageTemplate = \"${greeting} ${workTime}\"\n\n override def init: Future[Unit] = {\n   Future.unit\n }\n\n override def onSignal(signal: AutomationSignal): Future[Unit] = {\n   signal match {\n   case t: NeyrossEventAutomationSignal =>\n   if (isValidEvent(t.event)) {\n processEvent(t)\n   } else {\n   logger.info(s\"got event without AccessTaken event tags; ignoring\")\n   Future.unit\n   }\n   case _ =>\n   Future.unit\n   }\n }\n\n // ---\n\n private def isValidEvent(event: EventDto): Boolean = {\n   event.event.eventTags.getOrElse(Nil).contains(Tags.AccessTaken) &&\n   event.event.credentialToken.nonEmpty\n }\n\n private def processEvent(signal: NeyrossEventAutomationSignal): Future[Unit] = {\n logger.debug(s\"processing event ${signal.event.event.headline}\")\n   for {\n (personOpt, passOpt) <- getPersonAndPass(signal.event.event.credentialToken.get.toString)\n   _ <- if (personOpt.isDefined && notifyPerson) {\n performNotifyPerson(signal, personOpt.get)\n   } else {\n   Future.unit\n }\n   _ <- if (personOpt.isDefined && notifyResponsible) {\n performNotifyResponsible(signal, personOpt.get)\n   } else {\n   Future.unit\n   }\n   } yield ()\n }\n\n private def performNotifyPerson(signal: NeyrossEventAutomationSignal, person: Person): Future[_] = {\n   logger.debug(s\"notifying person (if able to) for person ${person.uuid}\")\n   person.getPropertyValue(personalChatIdPropertyKey) match {\n   case Some(chatId) =>\n   logger.debug(s\"notifying person to chat with id = $chatId\")\n   for {\n   msg <- getMessageForPerson(signal, person)\n   _ <- {\n   logger.debug(s\"ready to send message = $msg\")\n   telegramService.sendMessage(chatId, msg)\n   }\n   } yield ()\n   case _ =>\n   logger.debug(s\"not found personal chat id property ($personalChatIdPropertyKey) \" +\n   s\"in person ${person.uuid}; not notifying person\")\n   Future.unit\n   }\n\n }\n\n private def getMessageForPerson(signal: NeyrossEventAutomationSignal, person: Person): Future[String] = {\n   logger.debug(s\"getting message for person\")

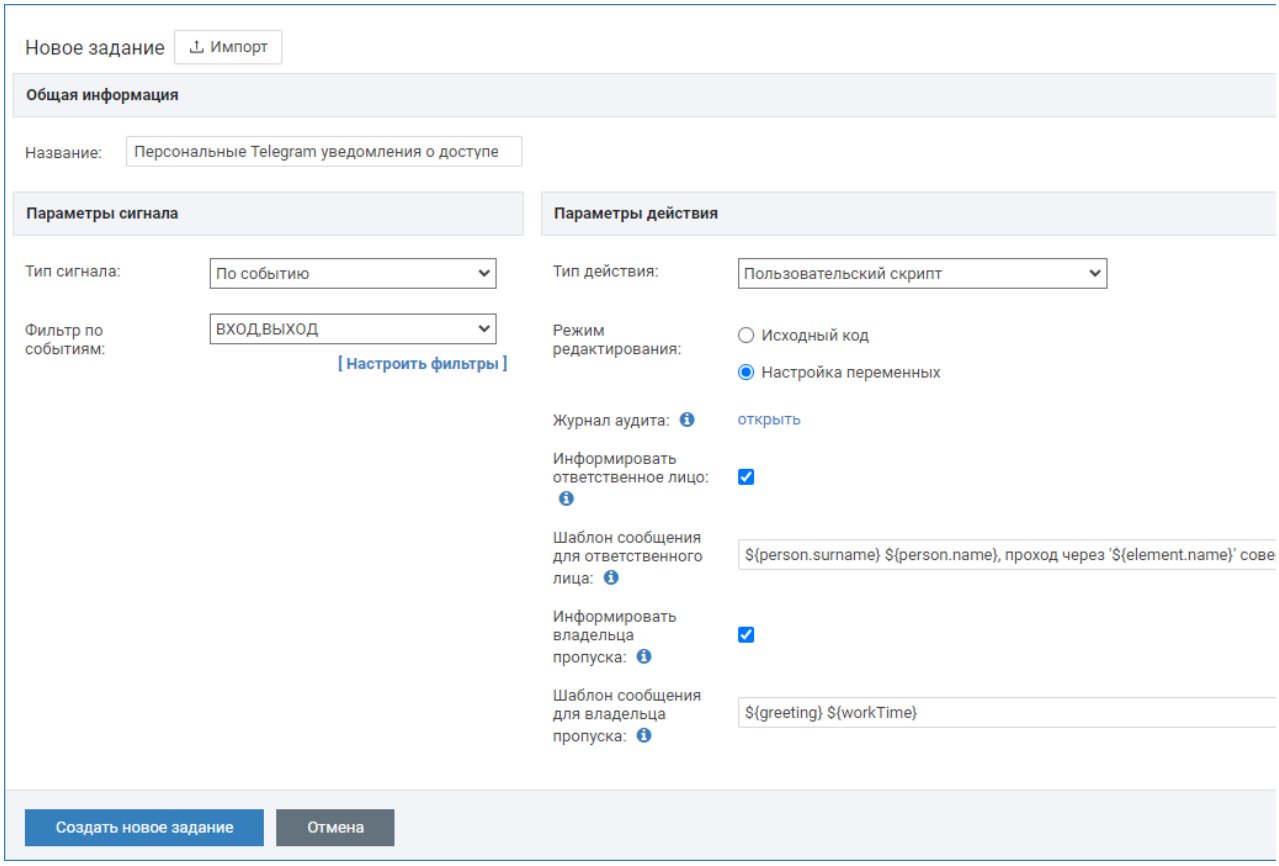
```
\n        for {\n          workTimeOpt <- if (personMessageTemplate.contains
(workTimeTemplateString)) {\n            logger.debug(s\"message template contains
${workTimeTemplateString}, calculating work time\")\n            getWorkTime(signal.
event, person)\n          } else {\n            logger.debug(s\"message template not contains
${workTimeTemplateString}, not calculating work time\")\n            Future.successful
(None)\n          }\n          patchedTemplateString = personMessageTemplate.replace
(workTimeTemplateString, workTimeOpt.getOrElse(\"\"))\n          greetingOpt = if
(personMessageTemplate.contains(greetingTemplateString)) {\n            logger.debug
(s\"message template contains ${greetingTemplateString}, forming greeting\")\n
Some(getPersonGreeting(signal, person))\n          } else {\n            logger.debug(s\"
message template not contains ${workTimeTemplateString}, not calculating work
time\")\n            None\n          }\n          secondPatchedTemplateString =
patchedTemplateString.replace(greetingTemplateString, greetingOpt.getOrElse(\"\"))
\n          compiledTemplate = {\n            logger.debug(s\"ready to compile template
(patched template string is $secondPatchedTemplateString)\")\n
messageTemplateService.compileTemplate(secondPatchedTemplateString) match
{\n              case scala.util.Success(t) =>\n                t\n              case scala.util.Failure
(exception) =>\n                logger.error(s\"cannot compile message template. Reason:
$exception\")\n                throw exception\n            }\n          }\n          message <-
messageDataService.setDataToTemplate(compiledTemplate, signal)\n        } yield
message\n      }\n\n      private def getPersonGreeting(signal:
NeyrossEventAutomationSignal, person: Person): String = {\n        signal.event.
extensions.get(EventExtensions.FunctionalElement).map(_.asInstanceOf
[FunctionalElement]) match {\n          case Some(fe) =>\n            if (isEntryAccessPoint
(fe)) {\n              s\"Добро пожаловать, ${person.name.getOrElse(\"\")}!\"\n            }
else if (isExitAccessPoint(fe)) {\n              s\"Всего доброго, ${person.name.getOrElse
(\"\")}!\"\n            } else {\n              s\"Выполнен проход через точку доступа '${fe.
name}'.\"\n            }\n          case _ =>\n            s\"Выполнен проход через точку
доступа.\"\n        }\n      }\n\n      private def performNotifyResponsible(signal:
NeyrossEventAutomationSignal, person: Person): Future[_] = {\n        logger.debug(s\"
notifying responsible (if able to) for person ${person.uuid}\")\n        person.
getPropertyValue(responsibleChatIdPropertyKey) match {\n          case Some(chatId)
=>\n            logger.debug(s\"notifying responsible to chat with id = $chatId\")\n
for {\n              msg <- getMessageForResponsible(signal, person)\n
attachments <- getAttachmetsForResponsible(signal, person)\n              _ <-
{\n                logger.debug(s\"ready to send message = $msg\")\n
telegramService.sendMessage(chatId, msg)\n              }\n              _ <- {\n                if
(attachments.nonEmpty) {\n                  logger.debug(s\"got non-emtpy attachments;
sending them\")\n                  telegramService.sendMediaGroup(chatId, attachments)
\n              } else {\n                Future.unit\n              }\n              }\n            } yield ()
\n          case _ =>\n            logger.debug(s\"not found responsible chat id property
($responsibleChatIdPropertyKey) \" +\n              s\"in person ${person.uuid}; not
notifying responsible\")\n            Future.unit\n        }\n      }\n\n      private def
getAttachmetsForResponsible(signal: NeyrossEventAutomationSignal, person:
Person) = {\n        if (snapshotsForResponsible) {\n          val request =
GetEventMediaInfoRequest(\n            signal.event.event.functionalElement,\n
signal.event.event.sent,\n            withVideo = false,\n            withSnapshots = true,
```

\n videoIntervalDuration = 0\n)\n eventMediaInfoBean.getForEvent
(request).map(response => {\n logger.debug(s\"got event media info:
$response\")\n response.snapshots\n })\n } else {\n Future.
successful(Nil)\n }\n }\n\n private def getMessageForResponsible(signal:
NeyrossEventAutomationSignal, person: Person): Future[String] = {\n logger.debug
(s\"getting message for responsible\")\n for {\n workTimeOpt <- if
(responsibleMessageTemplate.contains(workTimeTemplateString)) {\n logger.
debug(s\"message template contains ${workTimeTemplateString}, calculating work
time\")\n getWorkTime(signal.event, person)\n } else {\n logger.
debug(s\"message template not contains ${workTimeTemplateString}, not calculating
work time\")\n Future.successful(None)\n }\n patchedTemplateString
= responsibleMessageTemplate.replace(workTimeTemplateString, workTimeOpt.
getOrElse(\"\"))\n compiledTemplate = {\n logger.debug(s\"ready to
compile template (patched template string is $patchedTemplateString)\")\n
messageTemplateService.compileTemplate(patchedTemplateString) match {\n
case scala.util.Success(t) =>\n t\n case scala.util.Failure(exception)
=>\n logger.error(s\"cannot compile message template. Reason: $exception\")
\n throw exception\n }\n }\n message <-
messageDataService.setDataToTemplate(compiledTemplate, signal)\n } yield
message\n }\n\n private def getWorkTime(dto: EventDto, person: Person): Future
[Option[String]] = {\n logger.debug(s\"calculating work time for person $person and
event ${dto.event.headline}\")\n dto.extensions.get(EventExtensions.
FunctionalElement).map(_.asInstanceOf[FunctionalElement]) match {\n case
Some(fe) if isExitAccessPoint(fe) =>\n for {\n eventsWithElements <-
getAccessEventsForToday(person)\n workTimeDto =
getWorkMinutesForToday(eventsWithElements)\n } yield {\n val parts:
ListBuffer[String] = new ListBuffer[String]()\n if (workTimeDto.workedMinutes >
0) {\n parts.addOne(s\"отработанное время: ${workMinutesToHuman
(workTimeDto.workedMinutes)}\")\n }\n if (workTimeDto.idleMinutes >
0) {\n parts.addOne(s\"перерыв: ${workMinutesToHuman(workTimeDto.
idleMinutes)}\")\n }\n if (parts.isEmpty) {\n None\n }
else {\n Some(parts.mkString(\", \"))\n }\n }\n case _
=>\n Future.successful(None)\n }\n }\n\n private def localizeHours
(hours: Int): String = {\n val str = hours.toString\n var postfix = \"часов\"\n
if (str.endsWith(\"1\")) {\n postfix = \"час\"\n }\n if (str.endsWith(\"2\") || str.
endsWith(\"3\") || str.endsWith(\"4\")) {\n postfix = \"часа\"\n }\n s\"$str
$postfix\"\n }\n\n private def localizeMinutes(minutes: Int): String = {\n val str
= minutes.toString\n var postfix = \"минут\"\n if (str.endsWith(\"1\")) {\n
postfix = \"минута\"\n }\n if (str.endsWith(\"2\") || str.endsWith(\"3\") || str.
endsWith(\"4\")) {\n postfix = \"минуты\"\n }\n s\"$str $postfix\"\n }
\n\n private def workMinutesToHuman(minutes: Long): String = {\n val parts:
ListBuffer[String] = new ListBuffer[String]()\n val hours = scala.math.floor(minutes /
60).toInt\n val residualMinutes = scala.math.abs(minutes % 60).toInt\n if
(hours > 0) {\n parts.addOne(localizeHours(hours))\n }\n if (minutes > 0)
{\n parts.addOne(localizeMinutes(residualMinutes))\n }\n parts.mkString
(\", \")\n }\n\n private def getWorkMinutesForToday(eventsWithElements: List
[(Event, Option[FunctionalElement])]): WorkTimeDto = {\n var workedMinutes:

```
Long = 0\n        var idleMinutes: Long = 0\n        var previousEvent: Option[(Event,
Option[FunctionalElement])] = None\n        if (eventsWithElements.nonEmpty) {\n
eventsWithElements.foreach(eventWithElement => {\n            if (previousEvent.
isEmpty) {\n                // do nothing\n            } else {\n                logger.debug(s\"current
event is ${eventWithElement._1.sent}, prev is $previousEvent\")\n                val
diffInMinutes = scala.math.floor((eventWithElement._1.sent.toEpochSecond -
previousEvent.get._1.sent.toEpochSecond) / 60).toLong\n                if
(isEntryAccessPoint(previousEvent.get._2.get) && isExitAccessPoint
(eventWithElement._2.get)) {\n                logger.debug(s\"prev is enter, now is exit;
adding $diffInMinutes minutes to work time\")\n                workedMinutes +=
diffInMinutes\n                } else if (isExitAccessPoint(previousEvent.get._2.get) &&
isEntryAccessPoint(eventWithElement._2.get)) {\n                logger.debug(s\"prev is
exit, now is enter; adding $diffInMinutes minutes to idle time\")\n                idleMinutes
+= diffInMinutes\n                } else if (isExitAccessPoint(previousEvent.get._2.get) &&
isExitAccessPoint(eventWithElement._2.get)) {\n                logger.debug(s\"prev is exit,
now is exit; consider this as second exit without prev enter; adding $diffInMinutes
minutes to work time\")\n                workedMinutes += diffInMinutes\n                } else if
(isExitAccessPoint(previousEvent.get._2.get) && isExitAccessPoint(eventWithElement.
_2.get)) {\n                logger.debug(s\"prev is enter, now is enter; do nothing\")
\n                }\n            }\n            previousEvent = Some(eventWithElement)\n        })
\n        }\n        WorkTimeDto(workedMinutes, idleMinutes)\n    }\n\n    private def
getAccessEventsForToday(person: Person): Future[List[(Event, Option
[FunctionalElement])]] = {\n        for {\n            accessPointElements <- ctx.dbRun( //
getting all access points, that marked as Entry and Exit\n
functionalElements\n                .findByTypeTag(Tags.AccessPoint)\n                .map
(elements => {\n                elements.filter(element => {\n                isEntryAccessPoint
(element) || isExitAccessPoint(element)\n                })\n                })\n            )\n
foundPasses <- ctx.dbRun(passes.findByPerson(person.uuid))\n            foundEvents <-
{\n            val searchFrom = OffsetDateTime\n                .now()\n                .withHour(0)
\n                .withMinute(0)\n            val searchTo = OffsetDateTime\n                .now()
\n            logger.debug(s\"searching AccessTaken events from $searchFrom to
$searchTo, access points are ${accessPointElements.map(_.name)}\")\n
eventBean\n                .searchInRange(\n                beans.common.EventSearchFilter
(\n                includeEventTags = Some(Seq(Tags.AccessTaken)),\n                None,
\n                credentialTokens = Some(foundPasses.map(_.uuid)),\n
functionalElements = Some(accessPointElements.map(_.id.get))\n                ),
\n                searchFrom,\n                searchTo,\n                None\n                )\n            }
\n        } yield {\n            logger.debug(s\"found ${foundEvents.size} events\")\n
foundEvents.map(event => {\n            (event, accessPointElements.find(_.id.contains
(event.functionalElement)))\n            })\n            .sortBy(_._1.sent)\n            .
toList\n\n        }\n    }\n\n    private def getPersonAndPass(credentialToken: String):
Future[(Option[Person], Option[Pass])] = {\n        for {\n            passOpt <- passService.
findByUuid(credentialToken)\n            personOpt <- passOpt match {\n            case
Some(pass) if pass.person.isDefined =>\n                personBean.getByUuid(pass.person.
get)\n            case _ =>\n                Future.successful(None)\n            }\n        } yield
(personOpt, passOpt)\n    }\n\n    private def isEntryAccessPoint(fe:
FunctionalElement): Boolean = {\n        (fe.typeTags.getOrElse(Nil) ++ fe.userTags.
```

```
getOrElse(Nil)).contains(Tags.AccessPointEntry)\n      }\n\n      private def
isExitAccessPoint(fe: FunctionalElement): Boolean = {\n        (fe.typeTags.getOrElse
(Nil) ++ fe.userTags.getOrElse(Nil)).contains(Tags.AccessPointExit)\n      }\n\n\n    }
\n\n    new TelegramPersonalPacsDataAutomationScript"
        }
    },
    "name": "Персональные Telegram уведомления о доступе"
}
```

В разделе **Автоматизация** нажмите на кнопку **Добавить новое задание**, нажмите на кнопку **Импорт** и укажите путь к подготовленному на предыдущем этапе файлу [Автоматизация].



В блоке **Параметры сигнала** в поле **Фильтр по событиям** выберите из раскрывающегося списка фильтр, подготовленный на шаге [Настройка фильтра].

В блоке **Параметры действия** настройте поля согласно таблице ниже.

Поле	Диапазон значений	Комментарий
Информировать ответственное лицо	Да/Нет, логическое поле	Снимите флаг, если не требуется отправлять уведомление ответственному лицу. По умолчанию флаг установлен, если в форме владельца пропуска указан

		идентификатор телеграмм-канала ответственного лица, будет отправлено сообщение, текст которого определён в поле **Шаблон сообщения для ответственного лица** .
Шаблон сообщения для ответственного лица	Произвольный текст и переменные	Определяет текст информирования ответственного лица. Может содержать переменные: • $(person.surname) — фамилия владельца пропуска; • $(person.name) — имя владельца пропуска; ⓘ Вы можете использовать уникальные ключи других свойств владельца или пропуска. Они указаны в разделе Настройки СКУД АРМ НЕЙРОСС Доступ [Добавление пользовательских свойств]. • $(element,name) — имя точки доступа, через которую осуществлён проход. • ${workTime} — время нахождения на предприятии (с учётом перерывов).
Информировать владельца пропуска	Да/Нет, логическое поле	Снимите флаг, если не требуется отправлять уведомление владельцу пропуска. По умолчанию флаг установлен, если в форме владельца пропуска указан идентификатор телеграмм-канала владельца, будет отправлено сообщение, текст которого определён в поле **Шаблон сообщения для владельца пропуска**.
Шаблон сообщения для владельца пропуска	Произвольный текст и переменные	Определяет текст информирования владельца пропуска. Может содержать тот же набор переменных, как в описании шаблона сообщения для ответственного лица, а также приветствие, заданное в переменной: • $(greeting). Переменная задаётся в коде скрипта [Изменение текста приветствия]

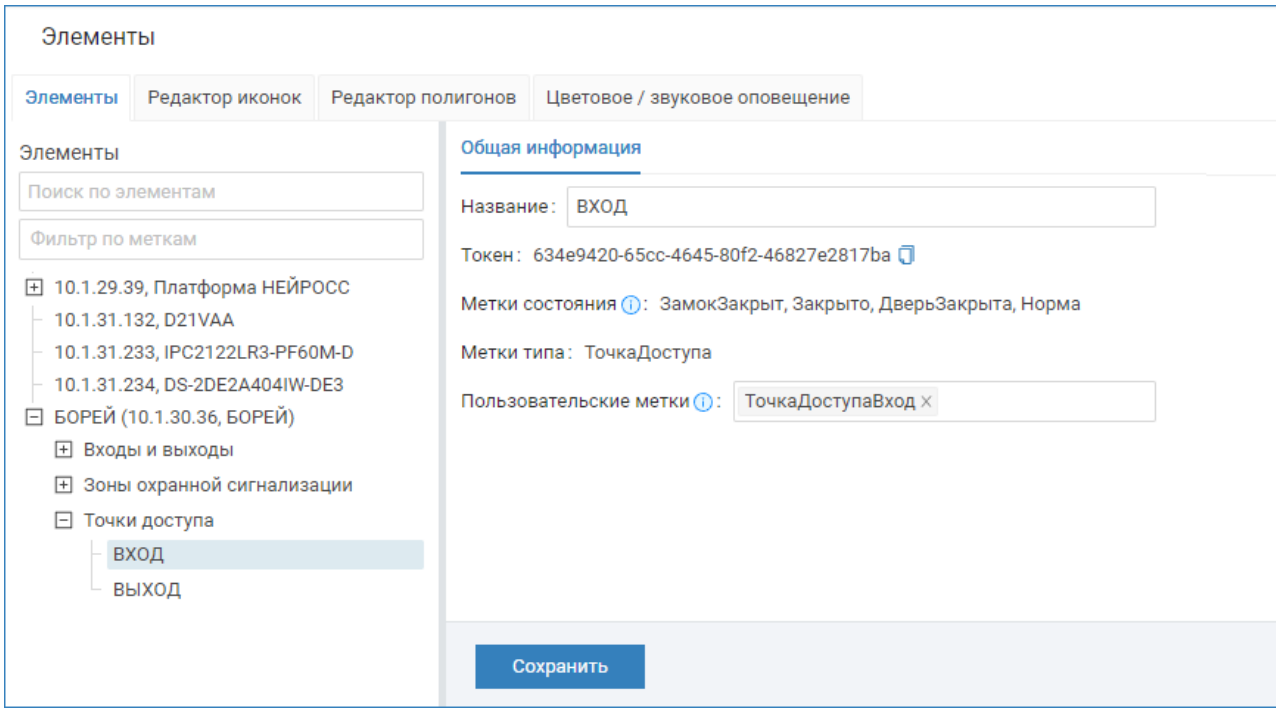
Добавление меток входа/выхода

При необходимости расчёта времени нахождения на предприятии в течение текущих суток, требуется указать, какие точки доступа являются входом на

предприятие, а какие — выходом из него. Для этого используется механизм пользовательских меток.

Перейдите в раздел **Элементы** [Элементы]. На вкладке **Элементы** выберите точку доступа контроллера, в поле **Пользовательские метки** задайте:

- Метку *ТочкаДоступаВход* для точек, являющихся входом на предприятие, сохраните изменения.
- Метку *ТочкаДоступаВыход* для точек доступа, являющихся выходом с предприятия, сохраните изменения.



Изменение текста приветствия

Переменная $(greeting) определяет приветствие владельца пропуска и зависит от метки точки доступа, через которую выполнен проход.

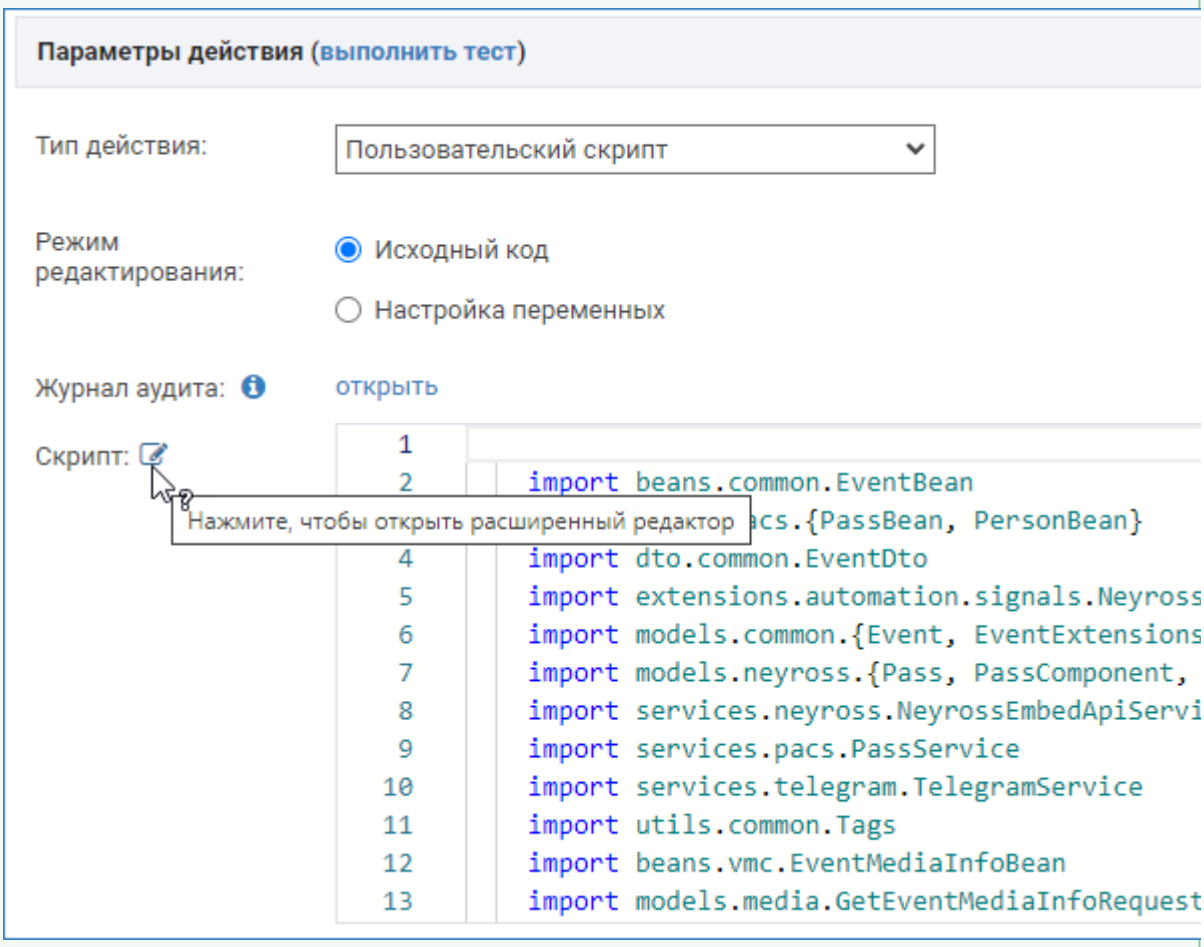
```
private def getPersonGreeting(signal: NeyrossEventAutomationSignal, person:
Person): String = {
      signal.event.extensions.get(EventExtensions.FunctionalElement).map(_.
asInstanceOf[FunctionalElement]) match {
        case Some(fe) =>
          if (isEntryAccessPoint(fe)) {
            s"Добро пожаловать, ${person.name.getOrElse("")}!"
          } else if (isExitAccessPoint(fe)) {
            s"Всего доброго, ${person.name.getOrElse("")}!"
          } else {
            s"Выполнен проход через точку доступа '${fe.name}'."
          }
        case _ =>
```

```
        s"Выполнен проход через точку доступа."
    }
  }
```

- При проходе через точку доступа, помеченной меткой *isEntryAccessPoint* (русифицированной как *ТочкаДоступаВход*), выводится сообщение: «*Добро пожаловать, [Имя владельца пропуска]*».
- При проходе через точку доступа, помеченной меткой *isExitAccessPoint* (русифицированной как *ТочкаДоступаВыход*), выводится сообщение «*Всего доброго, [Имя владельца пропуска]*».
- При проходе через точку доступа, не имеющей перечисленных меток, выводится сообщение «*Выполнен проход через точку доступа [Имя точки доступа]*».

ПОДСКАЗКА

Вы можете изменить текст приветствия прямо в коде скрипта, для этого выберите режим редактирования **Исходный код**, откройте расширенный редактор скриптов, с помощью поиска найдите нужный текст кода, откорректируйте тест и сохраните изменения.



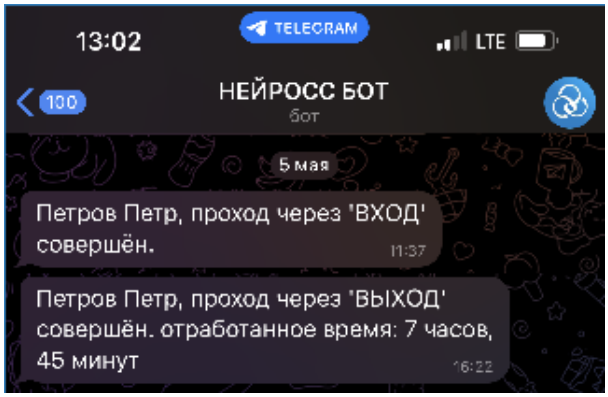
Отказ от меток и переменной $(greeting)

Если не требуется информировать об отработанном в текущие сутки времени вы можете вовсе отказаться от меток и переменной $(greeting), создать несколько

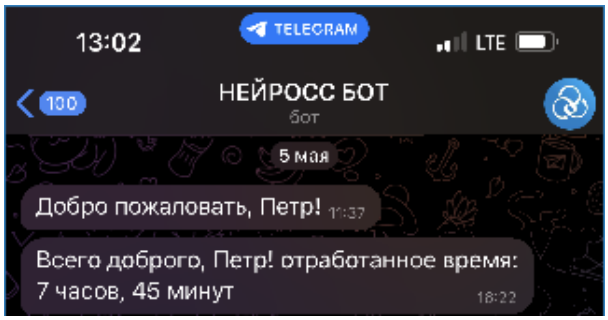
отдельных фильтров на разные точки доступа и для каждого фильтра создать отдельное задание автоматизации, указав в поле **Шаблон сообщения** собственный текст.

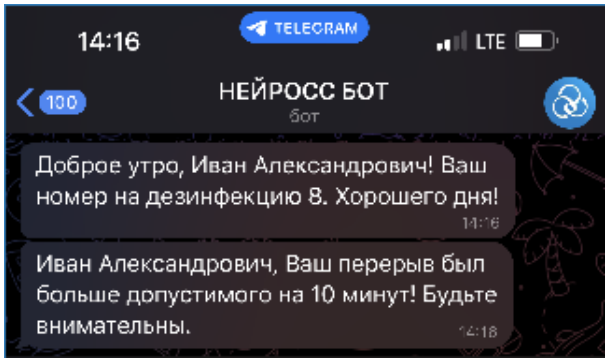
Пример работы скрипта автоматизации

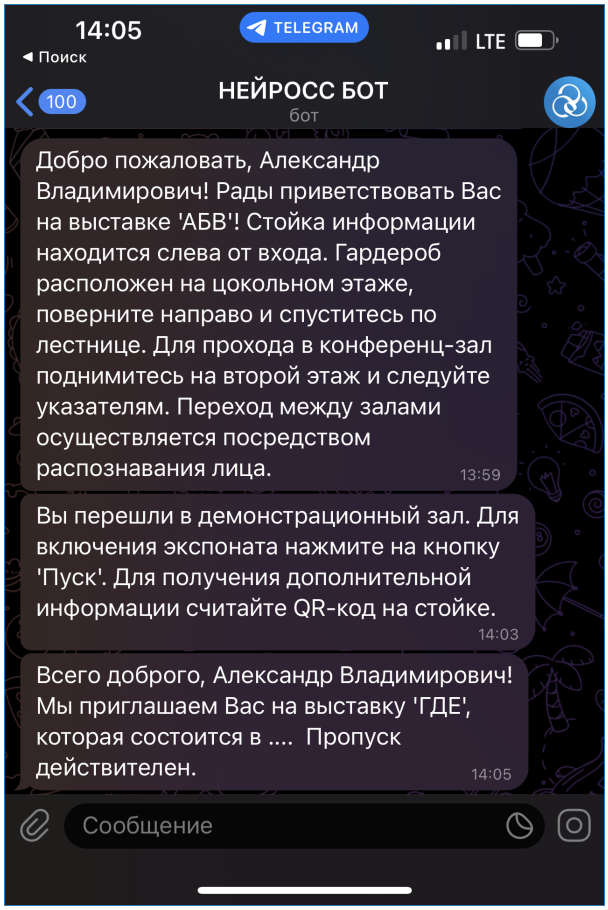
Уведомление ответственного лица:



Уведомление владельца пропуска:







Вы вольны задавать любые формулировки приветствия и тексты сообщений, а также доработать код скрипта для выполнения требуемых вам действий. Вы также можете обратиться к специалистам компании ИТРИУМ для разработки требуемого скрипта.