GIMS AUTOMATION MANAGER

РАБОЧАЯ ДОКУМЕНТАЦИЯ

ИНСТРУКЦИЯ РАЗРАБОТЧИКА

Москва 2022

СОДЕРЖАНИЕ

1 ОПИСАНИЕ ЛОГИЧЕСКИХ СУЩНОСТЕЙ

1.1 Сценарий автоматизации

1.1.1 Назначение сценариев автоматизации

«Сценарии автоматизации» - набор инструкций на языке программирования Python 3, предназначенный для реализации заданной бизнес-логики. Инициализация запуска сценария автоматизации выполняется с помощью активаторов. Процесс разработки сценария включает в себя обычные этапы написания скрипта на языке Python. Процесс разработки сценария выполняется на языке Python с использованием всех стандартных библиотек и методов.

1.1.2 Использование сценариев автоматизации

Разработка сценария автоматизации выполняется непосредственно в интерфейсе GIMS в разделе "Сценарии автоматизации".

В процесс разработки система позволяет использовать стандартные библиотеки Python. Если требуется импортировать какие-либо библиотеки, их необходимо установить в соответствующий инстанс среды Python.

На время отладки сценария можно инициировать его ручной запуск с помощью соответствующей кнопки на панели инструментов редактора. В этом случае таймаут выполнения задается автоматически и равен 3600 секунд.

1.1.3 Входные и выходные параметры

При запуске сценария из активатора можно передать входные параметры, которые в сценарии автоматизации будут доступны под теми же именами, которые были указаны при вызове.

Входные параметры типа «Источник данных» передаются в виде списка (List) имен источников данных. Для использования этих источников требуется их предварительно инициализировать с помощью функции: **load_data_sources.**

Для возврата в активатор результата выполнения сценария используется системная переменная: **_return_result**.

1.1.4 Специализированные функции

1) load_data_sources – инициализировать источники данных.

load_data_sources(ids_list=none, names_list=none, like_name=none, type_names_list=none, like_type_name=none)

где:

- *ids_list* - массив (list) идентификаторов ИД модели данных.
- *names_list* - массив строк полных имен ИД.
- *like_name* - строка содержащая часть имени ИД, необходимых ИД.
- *type_names_list* - массив строк полных имен типов ИД.
- *like_type_name* - строка содержащая часть имени типа ИД, необходимых ИД

Пример:
ds_array = load_data_sources(names_list=var_name)

2) set_cache – сохранить значение в кэш кластера GIMS Automation.

set_cache(<param_name>, <value>)

где:

- param_name – имя сохраняемого параметра
- value – значение параметра

Пример:
set_cache('last_id')

3) get_cache – получить значение из кэша кластера GIMS Automation.

get_cache(<param_name>)

где:

- param_name – имя сохраняемого параметра
- value – значение параметра.

Пример:
last_id = get_cache('last_id')

1.2 Типы источников данных

1.2.1 Назначение типов источников данных

«Источники данных» используются для подключения к внешним системам, с которыми осуществляется обмен данными. Это могут быть различные базы данных, API и т.д. Для настройки источника данных требуется наличие разработанного адаптера, называемого в GIMS «Тип источника данных».

1.2.2 Использование типов источников данных

Тип источника данных представляет собой описание объекта Python, а именно его свойств и методов.

1.2.2.1 Свойства типа источника данных

Свойства описываются в разделе «Свойства» соответствующего типа источника данных. Значения этих свойств задаются при создании источника данных, либо используются значения, заданные по умолчанию.

К свойствам можно обратиться непосредственно из кода сценария автоматизации или типа активатора: *<ds_name>.<property_name>*

Где:

- *ds_name* – имя соответствующего источника данных;
- *property_name* – имя свойства;

Пример:
faultdb.ip_address

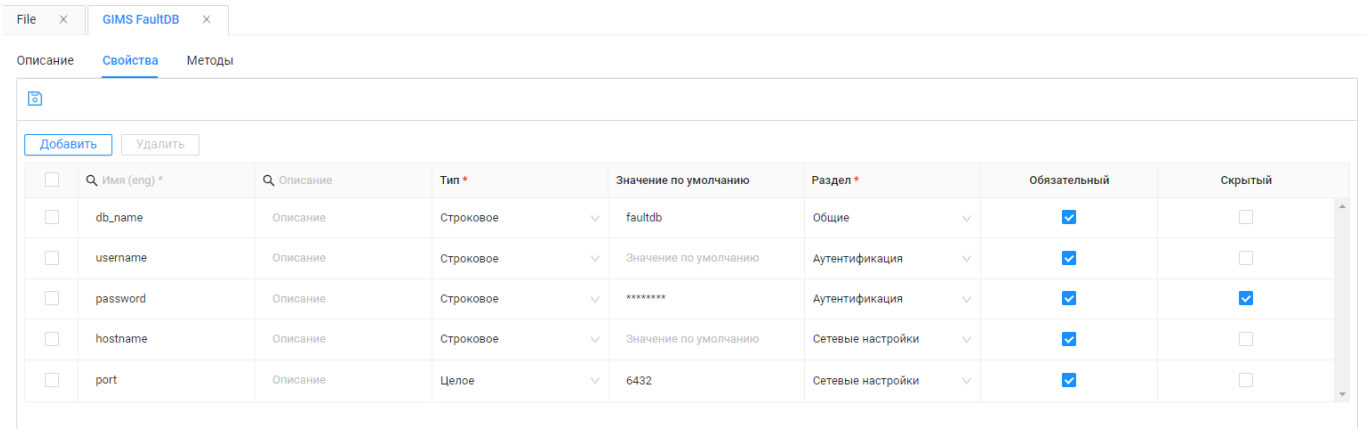

Рисунок 1 Пример описания свойств типа источника данных

При настройке свойств типов источников данных необходимо задать:

- Имя – имя свойства. Это имя будет использоваться при обращении к свойству.
- Описание – дополнительное описание.
- Тип – тип данных.
- Значение по умолчанию – значение, назначаемое соответствующему свойству источника данных в случае, если оно явно не было задано.
- Раздел – раздел в редакторе настроек источника данных. В этом разделе будет расположено свойство.
- Обязательный – признак обязательности заполнения свойства в редакторе соответствующего источника данных.
- Скрытый – скрывать значение свойства при редактировании. Используется при настройке конфиденциальных свойств, таких, как пароли или ключи.

1.2.2.2 МЕТОДЫ ТИПА ИСТОЧНИКА ДАННЫХ

Методы описываются в разделе «Методы» соответствующего типа источника данных. Для описания методов необходимо задать входные и выходные параметры и программный код этого метода на языке Python.

Методы можно вызвать непосредственно из кода сценария автоматизации или типа активатора: *<ds_name>.<method_name>(param1=value1, param2=value2,… paramN=valueN)*

Где:

- *ds_name* – имя соответствующего источника данных;

– *method_name* – имя метода;
– *paramN* – имя передаваемого параметра;
– *valueN* – значение передаваемого параметра.

Пример:
faultdb.execute(sql=my_query, limit=100)

1.2.2.3 ПАРАМЕТРЫ МЕТОДОВ

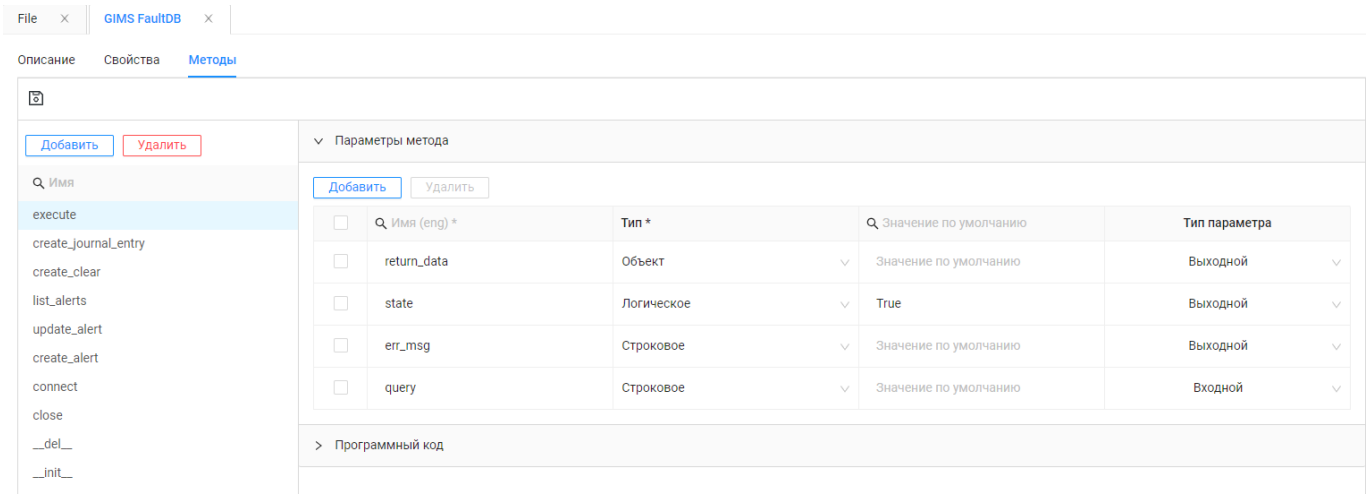

Рисунок 2 Пример описания параметров метода типа источника данных

Параметры методов задаются в разделе «Параметры метода» соответствующего метода.

При настройке перечня параметров необходимо задать:

– Имя – имя параметра. Это имя будет использоваться при вызове соответствующего метода из кода сценария автоматизации или типа активатора.
– Тип – тип данных.
– Значение по умолчанию – значение, назначаемое соответствующему параметру в случае, если оно явно не было задано.
– Тип параметра – Входной/Выходной.

1.2.2.4 Программный код метода

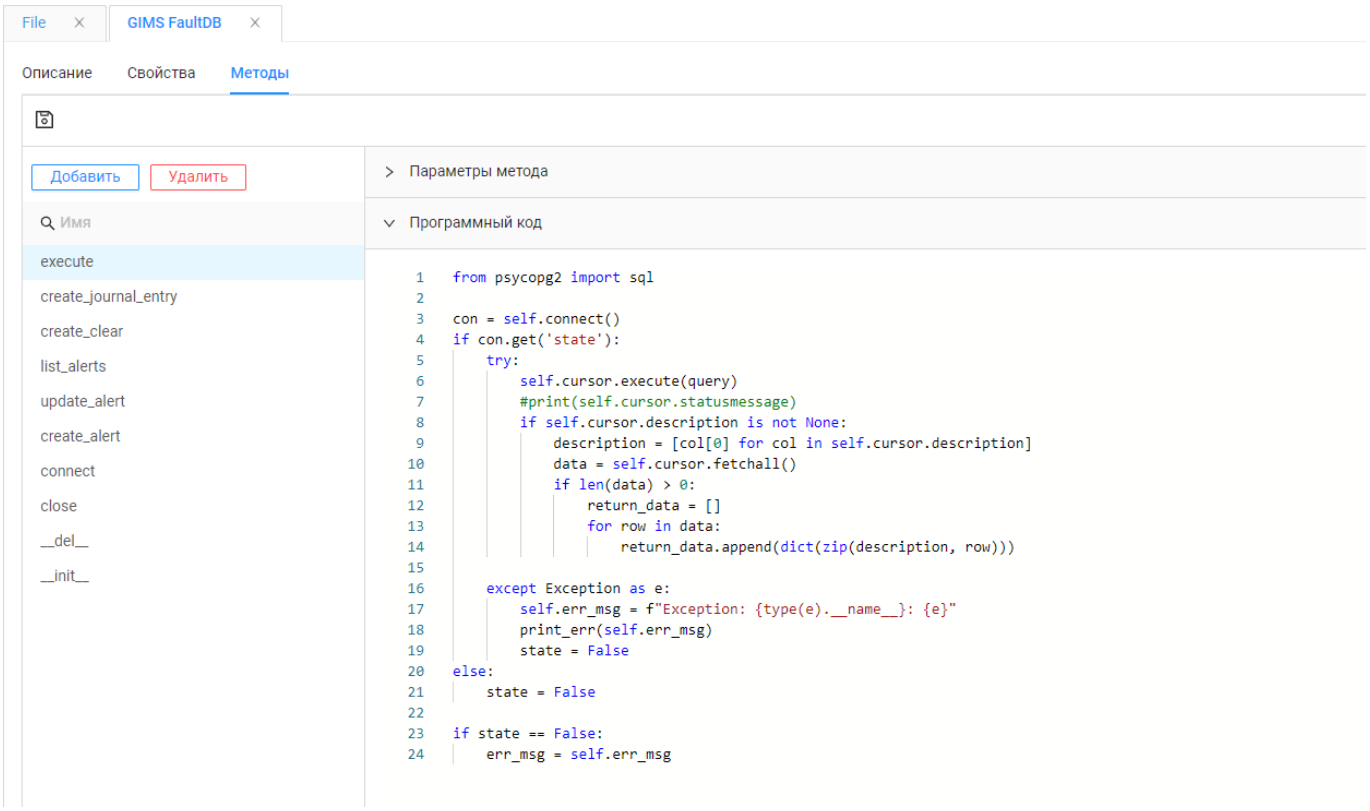

Рисунок 3 Пример кода метода типа источника данных

В программном коде метода должно выполняться какое-либо действие с использованием входных параметров. Результат выполнения должен быть записан в один или несколько выходных параметров, если таковые описаны. Входные и выходные параметры доступны под их непосредственным именем.

В программном коде метода можно обращаться к свойствам типа источника данных с помощью ссылки на объект self. С помощью этой же ссылки можно вызывать другие методы этого типа источника данных.

Пример:

Например, метод, приведенный выше (см. Рисунок 2 и Рисунок 3), принимает на вход параметр query, а результат возвращает в параметр return_result. Внутри метода выполняется вызов другого метода self.connect(). Также используется обращение в свойству self.err_msg.

1.3 Типы активаторов

1.3.1 Назначение типов активаторов

«Активаторы» — это программные модули, запускаемые в качестве демона. Из активатора может выполняться обращение к источникам данных, а также вызываться сценарии автоматизации. Для настройки активатора требуется наличие разработанного адаптера, называемого в GIMS «Тип активатора».

1.3.2 Использование типов активаторов

Тип активатора представляет собой скрипт Python, запускаемый в качестве демона, а также набор его свойств, задаваемых при создании экземпляра активатора данного типа.

1.3.3 Свойства типа активатора

Свойства описываются в разделе «Свойства» соответствующего типа активатора. Значения этих свойств задаются при создании экземпляра активатора данного типа, либо используются значения, заданные по умолчанию.

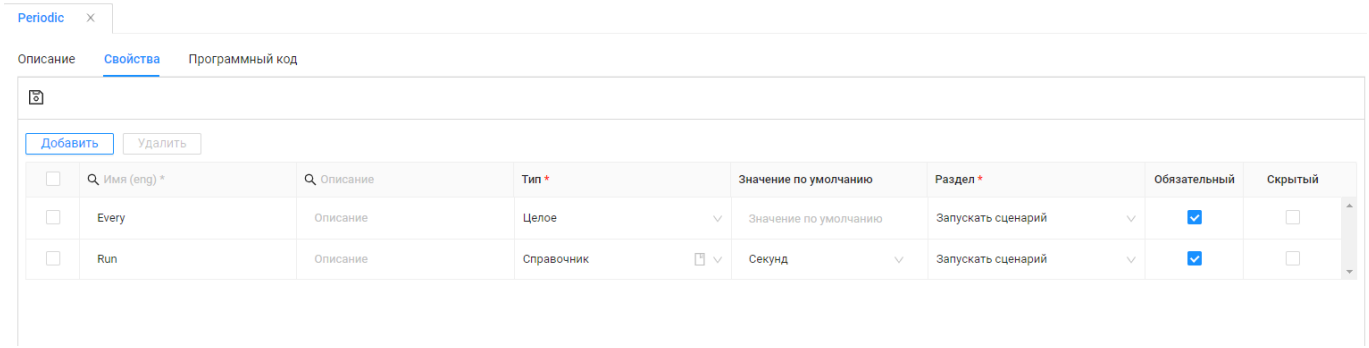

Рисунок 4 Пример свойств типа активатора

При настройке свойств типов источников данных необходимо задать:

– Имя – имя свойства. Это имя будет использоваться при обращении к свойству в коде типа активатора.

– Описание – дополнительное описание.
– Тип – тип данных.
– Значение по умолчанию – значение, назначаемое соответствующему свойству в случае, если оно явно не было задано при создании активатора данного типа.
– Раздел – раздел в редакторе настроек источника данных. В этом разделе будет расположено свойство.
– Обязательный – признак обязательности заполнения свойства в редакторе соответствующего активатора.
– Скрытый – скрывать значение свойства при редактировании. Используется при настройке конфиденциальных свойств, таких, как пароли или ключи.

1.3.4 Программный код метода

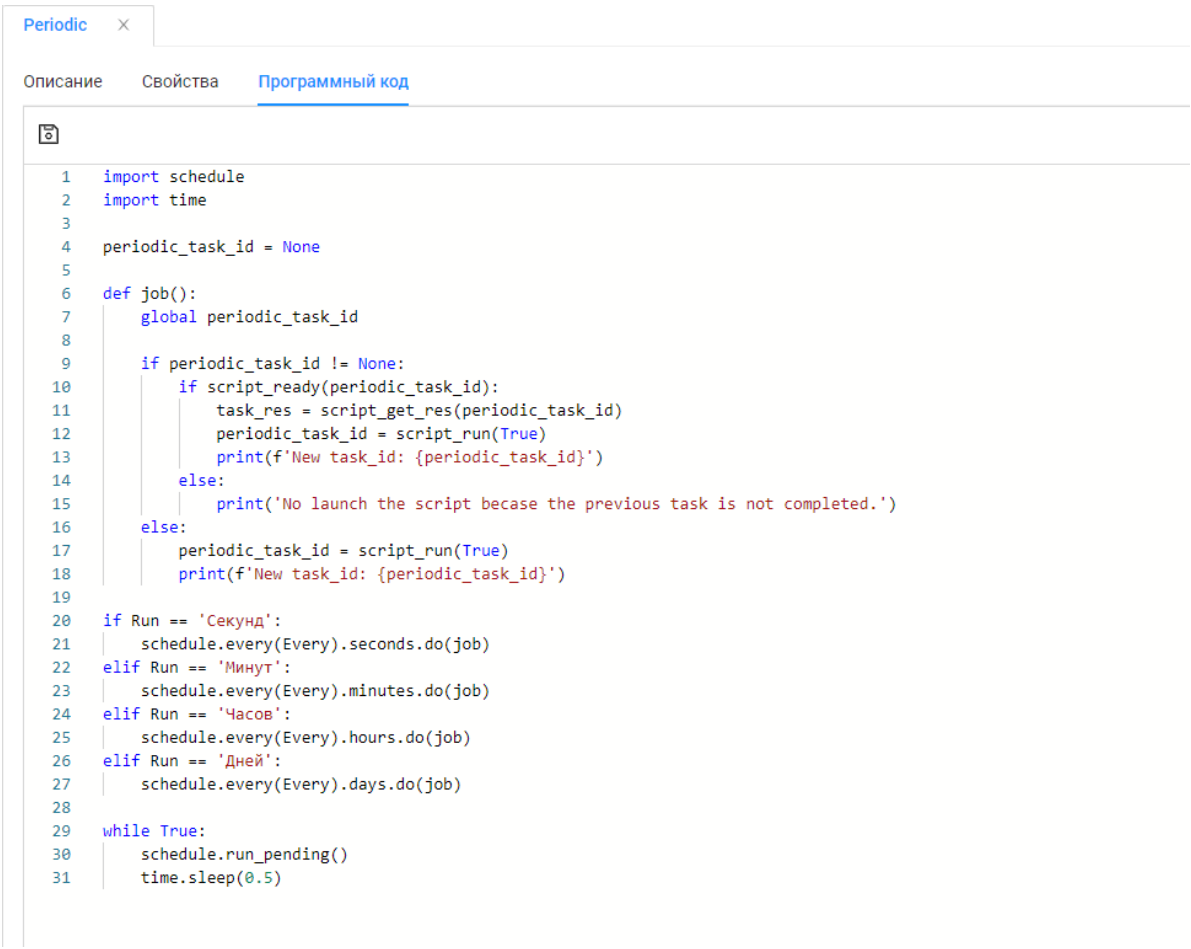

Рисунок 5 Пример программного кода типа активатора

В программном коде типа активатора описывается низкоуровневая обработка получаемых данных, которая не зависит от бизнес-задачи (например, прием snmp-trap или syslog сообщений, генерация периодических сигналов (crontab), сервер HTTP и т.п.). Дальнейшая обработка в зависимости от бизнес-требований рекомендуется реализовывать в сценариях автоматизации и вызывать их из типа активатора.

Для вызова сценария автоматизации используется метод:

task_id = script_run(type=True|False, params=qwargs**

Где:

- type – типа вызова. True – с возвратом, False – без возврата.
- params – передаваемые параметры, значения которых необходимы для дальнейшей обработки в сценарии.
- task_id – идентификатор вызова (по этому идентификатору можно получить результат выполнения сценария, если это требуется).

Если выполняется вызов с возвратом, то из кода активатора необходимо явно выполнить проверку наличия возвращенного значения, а затем получить сам результат. Для этого используются следующие методы:

is_ready = script_ready(task_id)

где:

- task_id – идентификатор вызова, полученный при выполнении вызова с помощью функции script_run;
- is_ready – True – вызванный сценарий завершил выполнение и можно получить результат его выполнения.

task_res = script_get_res(task_id)

где:

- task_id – идентификатор вызова, полученный при выполнении вызова с помощью функции script_run;
- task_res – результат выполнения сценария автоматизации.