

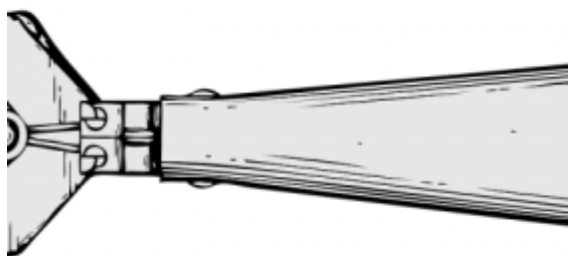
Тотальная автоматизация для 1С-разработчика. Как автоматизировать типовые задачи с помощью OneScript

Понедельник, 2 октября 2017 г.

Рубрика: [1С:Предприятие](#) -> [Программирование](#) -> [Статьи](#) -> [Журнал Системный Администратор](#)

Метки: [1С:Предприятие](#) | [OneScript](#) | [OpenSource](#) | [программирование](#) | [системный администратор](#)

Просмотров: 19261



Для автоматизации рутинных операций в операционной системе разработчики с системными администраторами давно привыкли использовать встроенные средства. CMD, PowerShell, VSScript, JS, BASH – прекрасные инструменты для разработки автоматизирующих сценариев. Минус у них лишь один – обязательность изучения еще одного диалекта.

Это не сильно большая проблема для профессионального разработчика, но согласитесь, хорошо, когда в экосистеме есть специализированный инструмент, решающий стандартные задачи на знакомом языке. За примерами далеко ходить не нужно. PHP, JavaScript, Python разработчики используют язык разработки для написания сценариев автоматизации. Они пишут на языке каждый день, и написать на нем кода для, скажем, резервного копирования разрабатываемого проекта – дело нескольких часов.

Здорово ведь, что не требуется переучиваться и разбираться с очередным синтаксисом вместо написания кода для решения определенной проблемы. Именно этого и не хватает 1С-разработчикам – инструмента, понимающего код на встроенном языке 1С и умеющий делать различные системные вещи.

К счастью, такой инструмент существует, причем уже не первый год и сегодня мы внимательно посмотрим на него с разных практических сторон. Итак, коллеги, знакомьтесь, OneScript – системная палочка-выручалочка для 1С-разработчиков.

Свободный, независимый OneScript

Начнем с главного, к созданию OneScript, компания 1С не имеет никакого отношения. Это полностью независимая кросс-платформенная реализация

виртуальной машины для исполнения сценариев на встроенном языке платформы «1С:Предприятия».

Решение создавалось специально для 1С-разработчиков, поэтому синтаксически это чистый язык 1С. Циклы, коллекции, условия и т.д. – все точно такое же, как мы привыкли использовать при разработке под платформу «1С:Предприятие».

«OneScript» предоставляет 1С-разработчику создавать сценарии автоматизации на привычном языке без привязки к платформе. Проще говоря, чтобы исполнить сценарий, написанный на OneScript, вам не требуется наличие установленной платформы «1С:Предприятие». Достаточно установленного интерпретатора «OneScript» и наличие в системе .NET Framework. Причем первое условие вполне реально обойти, но об этом чуть позже.

Другая важная особенность OneScript – кросс-платформенность. Платформа «1С:Предприятия» давно существует под разные платформы и «OneScript» придерживается той же идеологии. Проект создан с использованием .NET Framework, а значит, ничего не мешает применять «OneScript» на Linux, macOS. Проект Mono (реализация .NET под Linux/macOS) и ваши сценарии заработают в альтернативной среде.

Третья из приятных особенностей «OneScript» – расширяемость. Это значит, что разработчик при желании может расширить возможности «OneScript» за счет написания дополнительных библиотек. Под словом «библиотека» подразумевается не просто сборник жизненно-необходимых функций, а реализация полноценных объектов, которые могут быть использованы в сценариях и поставляться посредством менеджера пакетов OneScript Package Manager (opm).

«OneScript» развивается не первый год и за это время успел обзавестись полезными библиотеками от независимых авторов. Тем самым показав, что ставка на бесшовную расширяемость была сделана не напрасно.

Области применения

Первая очевидная ниша для «OneScript» – автоматизация типовых задач 1С-разработчика/администратора, связанных с сопровождением информационных баз. За примерами далеко ходить не нужно. Все мы (надеюсь, что все) написали свои «лучшие» сценарии для автоматизации резервного копирования, завершения работы пользователей в живой базе и другие знакомые всем вещи. Все выше перечисленные вещи наверняка создавались на одном из озвученных выше инструментов (CMD, PowerShell и т.д.).

«OneScript» поможет сделать все то же самое, но на «родном» языке программирования. Перечисленные задачи – идеальный полигон для «OneScript».

Хорошо, с обслуживанием все понятно. А для чего еще его можно применять? Тут все зависит от фантазии и задач, стоящих перед разработчиком. «OneScript»

прекрасно подойдет для создания консольных приложений. Вам необязательно думать о «OneScript», как о части экосистемы платформы «1С:Предприятие». Правильней думать о нем как об отдельном инструменте со знакомым языком. Тогда полезных сценариев применения удастся придумать значительно больше.

Одной из таких идей может стать добавление «OneScript» к другим проектам, где требуется предоставление возможности расширения за счет плагинов. Представим, что вы занимаетесь разработкой узкоспециализированной системой учета. У вас есть база довольных пользователей с постоянным запросом новых функций.

Одни предложения по доработке функционалы интересны и могут быть полезными широкому кругу пользователей, а другие узкие, решающие единичные проблемы. Тратить время на такие вещи нецелесообразно и вот тут становится очевидной реализация системы плагинов. Если ваш проект создан под платформу «1С:Предприятие», то особых трудностей нет. 1С-разработчиков много и кто-нибудь, да возьмется за доработки. Совсем иная ситуация с не столь распространенным технологическим стеком.

Да, вы без труда сможете предоставить API, но тем самым создадите необходимость изучения нового языка разработки. Некоторые разработчики таких подобных проектов понимают сложность изучения языков вроде C#/Java и предоставляют прикладным разработчикам собственный синтаксис (такие, мини языки программирования). Они думают, что несут пользу, но на самом деле создают дефицит разработчиков и еще больше проблем.

Утверждение может показаться спорным, но те, кто на практике столкнулся с сопровождением учетных систем вроде «Инфо-бухгалтер, Парус и т.д.» прекрасно понимают боль, связанную со сложностью применения нового синтаксиса/парадигмы для доработки решения.

С 1С-разработчиками ситуация противоположная. Их много и если позволить им создавать для продукта расширения на понятном им языке, то от этого выиграют все стороны. Для таких задач, «OneScript» подходит как нельзя идеально. Вам требуется организовать интеграцию «OneScript» со своим решением, а дальше эстафетную палочку примут прикладные разработчики.

Применений для «OneScript» множество, главное не бояться пробовать новый инструмент на практике и решать с его помощью реальные повседневные задачи.

Установка OneScript

Выше я упомянул о кросс-платформенности «OneScript». На официальном сайте ([1]) доступны пакеты для Windows (zip, exe, msi) и Linux (rpm, deb). Актуальная версия на момент написания статьи – 1.0.16. Все рассмотренные далее примеры воспроизводились в Windows. Если вы решили проверить «OneScript» на Linux, то будьте внимательны. Некоторые примеры из-за особенностей ОС не будут работать.

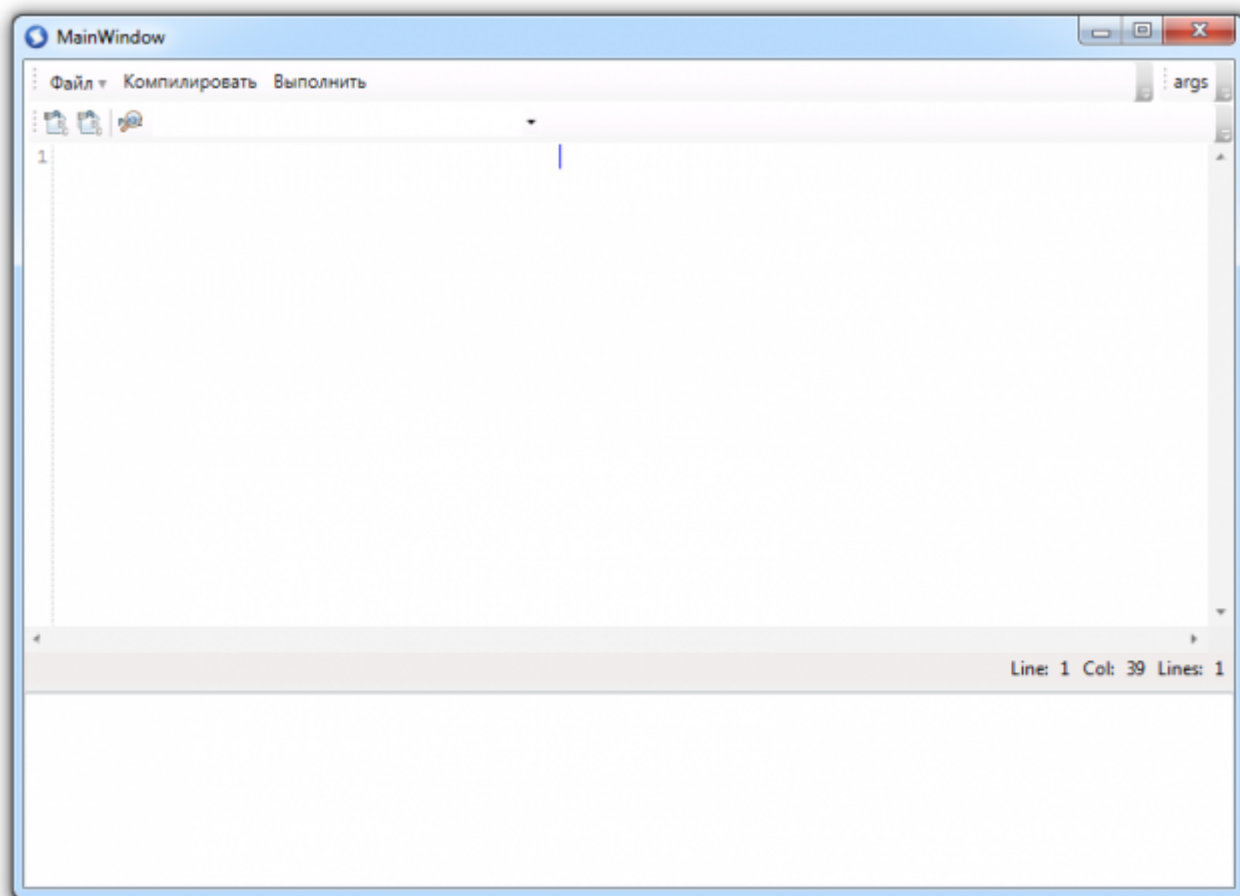
Установка «OneScript» выполняется стандартным образом – с помощью инсталлятора. В окне мастера установки рекомендую вам отметить все компоненты (по умолчанию они не ставятся). На начальном этапе изучения проекта они пригодятся.

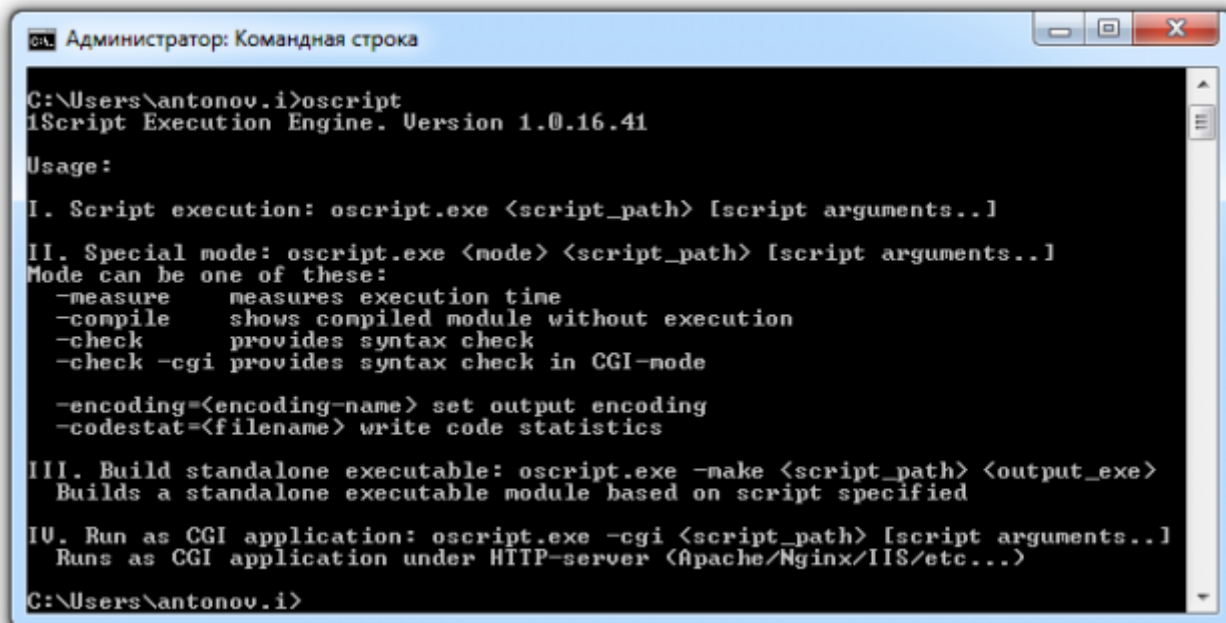
Перед установкой обратите внимание на одну зависимость – версия фреймворка .NET. Она должна быть не ниже 4.0 (или 4.4. в случае с Mono) . Поэтому проверьте и при необходимости обновите установленную на вашем компьютере версию .NET Framework.

После установке, в меню пуск появится ссылка на запуск консоли для разработки и тестирования сценариев (рисунок 1). Для созданных сценариев из командной строки, рекомендую сразу изменить системную переменную PATH.

Для этого запустите апплет панели управления «Система». Далее в дополнительных свойствах выберите «Параметры среды», найдите переменную PATH и добавьте в ее значение (через точку с запятой) путь к директории с установленным «OneScript». После этого перезагрузите систему.

После перезагрузке запустите консоль CMD и выполните команду `oscript`. Результат корректного выполнения показан на рисунке 2.





```
Администратор: Командная строка

C:\Users\antonov.i>oscript
iScript Execution Engine. Version 1.0.16.41

Usage:

I. Script execution: oscript.exe <script_path> [script arguments...]

II. Special mode: oscript.exe <mode> <script_path> [script arguments...]
Mode can be one of these:
-measure      measures execution time
-compile      shows compiled module without execution
-check        provides syntax check
-check -cgi   provides syntax check in CGI-mode

-encoding=<encoding-name> set output encoding
-codestat=<filename> write code statistics

III. Build standalone executable: oscript.exe -make <script_path> <output_exe>
Builds a standalone executable module based on script specified

IV. Run as CGI application: oscript.exe -cgi <script_path> [script arguments...]
Runs as CGI application under HTTP-server (Apache/Nginx/IIS/etc...)

C:\Users\antonov.i>
```

Где писать код

Код сценариев можно писать как в консоле (поставляется в комплекте), так и в вашем любимом редакторе кода. В консоле, доступной из коробки, работать с кодом не очень удобно, поэтому рекомендую посмотреть в сторону профессиональных редакторов – Atom, VSCode, Brackets и т.д.

Помимо программирования под платформу «1С:Предприятие», ваш покорный слуга занимается фронтендом, поэтому для меня универсальным редактором давно стал Visual Studio Code ([2]).

Технологически он похож на распиаренный Atom от GitHub, но выгодно отличается от него производительностью. Редактор в разы быстрее стартует и глюков у него на порядок меньше. К тому же, для разработки на «OneScript» для VSCode есть готовый плагин – Language 1C (BSL). Для загрузки пройдите в официальный маркет-плейс ([3]) или установить прямо из редактора, выполнив команду:

```
ext install language-1c-bsl
```

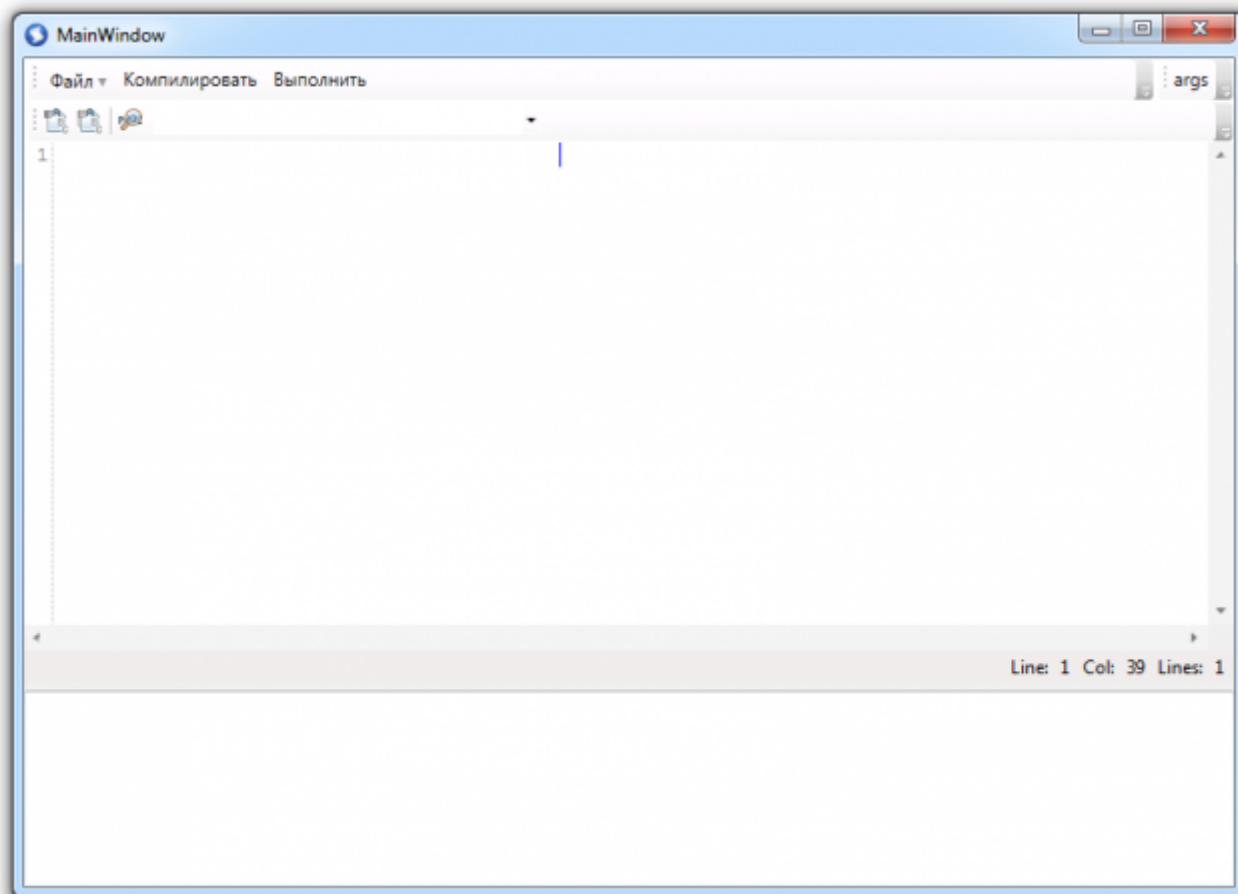
Будем считать, что с установкой и настройкой редактора для работы с OneScript вы справились. Я буду ориентироваться, что вы сделали выбор в пользу VSCode (поверьте, он действительно удобный). Создайте где-нибудь каталог, в котором будете размещать файлы примеров и откройте его в редакторе. Перед тем как добавлять новые файлы сценариев, нажмите F1 и в появившейся командной строке выполните: Language 1C (BSL): Create tasks.json for current workspace.

Тем самым вы создадите конфигурационный файл (tasks.json), описывающий настройки по запуску сценариев. Не будем погружаться в его содержимое, просто запомним, что для запуска сценария требуется нажать сочетание клавиш – ctrl + shift + b.

Убедимся в правильности настроек, попробуем создать новый сценарий (helloWorld.os) и напишем в нем традиционную фразу:

```
Сообщить("Hello, world");
```

Жмем ctrl + shift + b и если все работает корректно, в окне терминала вы видите вывод первой фразы программиста (см. рисунок 3).



Очищаем клиентский кеш

В рамках статьи хочется привести ряд полезных примеров, но они достаточно объемные и просто так код их не уместить. Поэтому здесь мы посмотрим решение задачи по очистке клиентского кеша (помните, таинственные папки в профиле пользователя?), а затем я познакомлю вас с готовыми библиотеками, на основе которых вы сможете написать полезные для себя сценарии.

До сих пор в составе платформы «1С:Предприятие» нет инструмента для быстрой очистки кэша. Необходимость в очистке кэша возникает не часто, но иногда все же приходится выполнять эту операцию.

Некоторые глюки, возникающие после обновления конфигурации элегантно решаются только очисткой кеша. Что подразумевается под этой процедурой? Нужно пройти по нескольким скрытым каталогам в директории профиля пользователя и удалить файлы.

Задача простая, но когда доходит дело до объяснения этого квеста пользователю, к которому нет возможности подключиться удаленно, трижды вспоминаешь о необходимости специализированного инструмента.

В разное время для решения этой проблемы я пользовался небольшим собственным приложением. Удобно, но не очень универсально. В первом листинге я привел решение этой задачи на «OneScript».

Листинг №1. Очистка клиентского кеша на OneScript

```
#Использовать logos
Перем ЖурналЛогирования;
Процедура ВыполнитьОчисткуКэша(Параметры)
    Для Каждого ДиректорияОбработки Из Параметры.ДиректорииДляОбработки
        ЖурналЛогирования.Информация("----- Очистка каталога: " + ДиректорияОбработки.РабочийКаталог + " Новый Файл(ДиректорияОбработки);

        Если НЕ РабочийКаталог.Существует() Тогда
            ЖурналЛогирования.Ошибка("!!! Директория " + ДиректорияОбработки.РабочийКаталог + " не существует. Продолжить;
        КонецЕсли;
        ФайлыДляУдаления = НайтиФайлы(ДиректорияОбработки, "*", Лож);
        Для Каждого ОбнаруженныйФайл Из ФайлыДляУдаления Цикл
            Если ОбнаруженныйФайл.ЭтоКаталог() И Параметры.Исключен(ОбнаруженныйФайл.ПолноеИмя) Тогда
                ЖурналЛогирования.Информация("----- Совпало исключение. Продолжить;
            КонецЕсли;
            Попытка
                УдалитьФайлы(ОбнаруженныйФайл.ПолноеИмя);
            Исклучение
                ЖурналЛогирования.Ошибка("!!! Не удалось удалить файл: " + ОбнаруженныйФайл.ПолноеИмя);
            КонецПопытки;
        КонецЦикла;
    КонецЦикла;
КонецПроцедуры
Функция ПодготовитьПараметры()
    СистемнаяИнформация = Новый СистемнаяИнформация;
    ДиректорияПрофиляПользователя = СистемнаяИнформация.ПолучитьПервыйРезультат = Новый Структура();
    Результат.Вставить("ДиректорииДляОбработки", Новый Массив);
    Результат.Вставить("Исключения", Новый Массив);
    Результат.ДиректорииДляОбработки.Добавить(ДиректорияПрофиляПользователя);
    Результат.ДиректорииДляОбработки.Добавить(ДиректорияПрофиляПользователя);
    Результат.ДиректорииДляОбработки.Добавить(ДиректорияПрофиляПользователя);
```

```
Результат.ДиректорииДляОбработки.Добавить(ДиректорияПрофиляПоль
Результат.Исключения.Добавить("ExtCompT");
```

```
Возврат Результат;
```

```
КонецФункции
```

```
ЖурналЛогирования = Логирование.ПолучитьЛог("oscript.app.cleaner");
```

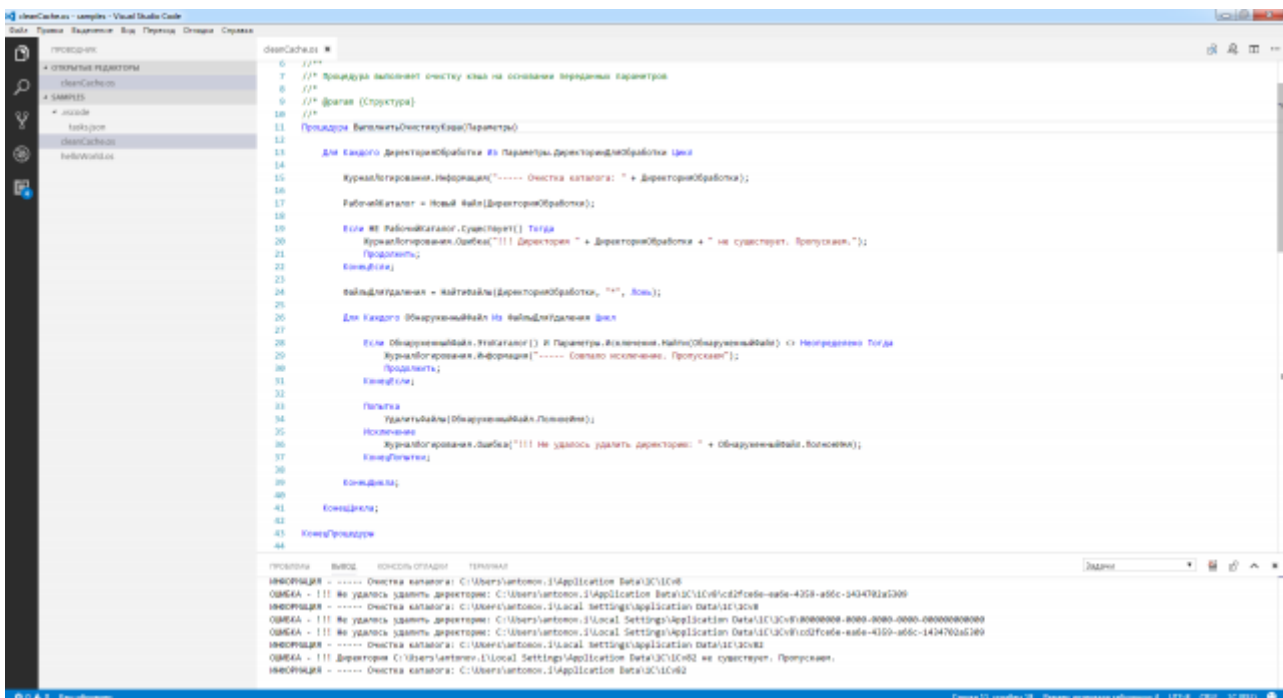
```
Параметры = ПодготовитьПараметры();
```

```
ВыполнитьОчисткуКэша(Параметры);
```

Код получился достаточно тривиальным и если вы хоть раз писали обработку для работы с файлами, то сразу разберетесь с происходящим. Помимо стандартного кода на 1С, вы могли заметить применение директивы «#Использовать». В «OneScript» она применяется для подключения к сценарию библиотеки. В примере такой библиотекой выступает – logos. Это отличный инструмент для организации в коде логирования и он применяется практически во всех разрабатываемых сценариях.

Чтобы воспользоваться в сценарии сторонней библиотекой, ее требуется сначала установить с помощью пакетного менеджера (opm), а затем подключить к сценарию директивой “#Использовать <ИмяБиблиотеки>”. Если вы просто скопировали код листинга 1 и попробовали его выполнить, то вместо очистки кеша получите ошибку «Библиотека logos» не найдена. Выполните ее установку, а затем запустите сценарий (ctrl + shift + b):

```
opm install logos
```



Больше библиотек

Как я уже говорил: «OneScript расширяется за счет сторонних библиотек». Таких библиотек уже не мало и о нескольких полезных мы сейчас поговорим. Полный

список библиотек доступен в репозитории ([4]), а здесь я упомяну о трех, постоянно используемых мной в практике.

cmdline

Большинству сценариев, так или иначе, требуется взаимодействовать с передаваемыми параметрами командной строки. Считать файл, загрузить файл по указанному пути – все эти данные попадают сценарию при помощи параметров. Для работы с параметрами командной строки, которые получает сценарий, в «OneScript» доступна коллекция (массив) – «АргументыКоманднойСтроки». Из нее разработчик может вытащить все параметры, которые передали сценарию. Однако, работать с этой коллекцией не сильно удобно.

Представим, что у нас есть сценарий, который умеет считывать параметры “-firstname” и “-lastname”. Цель скрипта банальна до безобразия – сформировать строку имени и фамилии на основании данных из перечисленных параметров. Если мы запустим наш сценарий командой:

```
oscript ourscript.os -firstname Игорь -lastname Антонов
```

А затем попробуем прочитать коллекцию «АргументыКоманднойСтроки» в цикле, то увидим никак не связанный из себя набор элементов:

- firstname
Игорь
- lastname
Антонов

Чтобы сотворить из списка удобную для работы структуру придется написать мини-парсер. Может показаться, что это не слишком сложно, но не стоит забывать, что параметров может быть несколько и пользователь имеет возможность указывать их в произвольном порядке. Библиотека *cmdline* такие проблемы решает максимально эффективно. Во втором листинге представлен код простого приложения, демонстрирующий применение *cmdline* на практике.

Листинг 2. Эффективная работа с параметрами командной строки

```
#Использовать cmdline
Процедура КакТебяЗовут()
    ПарсерКоманднойСтроки = Новый ПарсерАргументовКоманднойСтроки()
    ПарсерКоманднойСтроки.ДобавитьИменованныйПараметр("-firstname")
    ПарсерКоманднойСтроки.ДобавитьИменованныйПараметр("-lastname");
    РазобранныеПараметры = ПарсерКоманднойСтроки.Разобрать(Аргумент
Сообщить("Тебя зовут:" + РазобранныеПараметры["-firstname"] + " " +
```

```
КонецПроцедуры  
КакТебяЗовут( )
```

При использовании `cmdline`, получить значение разобранных команд, сводится к обращению к одноименным ключам полученной на выходе структуры.

oscript-yadisk

В нашей компании мы активно используем облачный сервис «Яндекс.Диск». У нас полно автоматизирующих сценариев, написанных еще на PHP, выполняющих подготовку и миграцию различных данных в облако. Наша команда была приятно удивлена, что для «OneScript» существует отличная готовая библиотека (`oscript-yadisk`), позволяющая без лишних сложностей взаимодействовать (загружать файлы, получать информацию, скачивать файлы и т.д.) с популярным сервисом от Яндекс.

При ее использовании, работа с «Яндекс.Диск» сводится к вызову одноименных команд. В официальной репозитории есть подробные примеры использования библиотеки. Библиотека устанавливается стандартным способом через `orm`.

json

Из коробки в OneScript не реализована поддержка популярного формата JSON, поэтому для работы с ним придется воспользоваться одноименной библиотекой. Библиотека основана на модуле «1С:JSON Александр Переверзева», который долгое время был стандартом для работы с JSON в платформе «1С:Предприятие» (пока не появились нативные методы).

v8runner

Интересное решение, упрощающее взаимодействие с конфигуратором. По факту это обертка для широкого списка параметров командной строки, которые принимает конфигуратор. Рассмотрим пример. Вы помните весь список ключей, при помощи которых конфигуратор выполнит загрузку конфигурации из файла? Если помните, то значит у вас отличная память. Посмотрите, как эта задача решается с помощью модуля `v8runner`:

```
#Использовать v8runner  
Конфигуратор = Новый УправлениеКонфигуратором();  
Конфигуратор.УстановитьКонтекст("/IBConnectionString" "Srvr=server:1  
Конфигуратор.ЗагрузитьКонфигурациюИзФайла("C:uh.cf");
```

На основе `v8runner` можно написать множество типовых сценариев для автоматизации. Мы используем библиотеку для организации резервного копирования, быстрого восстановления баз и т.д. Пользователь `gitHub'a` –

BlackDrak0n, пошел дальше и на основе библиотеки создал универсальный сценарий для обновления типовых конфигураций.

Его проект `oscript-AutoUpdateIB` ([5]) умеет автоматически загружать обновления с официального сайта поддержки пользователей, выполнять резервное копирование ИБ, а затем накатывать загруженные обновления. Незаменимая вещь при сопровождении большого количества типовых решений.

Модульное тестирование в массы

Одним из ограничений «OneScript» на начальном этапе освоение будет отсутствие встроенного отладчика. 1С-разработчик применяет пошаговую отладку кода постоянно, поэтому при освоении скриптового инструмента возникнут небольшие сложности. Первой альтернативой классическому отладчику однозначно станет логирование, реализуемое библиотекой `logos`. По мере роста исходного кода сценариев одного логирования будет недостаточно, и тут на помощь готовы прийти модульные тесты.

«OneScript» поддерживает тестирование из коробки. Загружаем с официального сайта сценарий `testrunner.os`. Как видно из названия он предназначен для последовательного запуска тестов. Дальше создаем директорию для хранения тестов к сценарию и добавляем файл с описанием тестов (листинг 3).

Листинг 3. Пример файла с тестами

```
Функция ПолучитьСписокТестов(ЮнитТестирование) Экспорт
    юТест = ЮнитТестирование;
    ВсеТесты = Новый Массив;

    ВсеТесты.Добавить("ТестДолжен_ПроверитьУмножениеЧисле2и2");
    сообщить(ЮнитТестирование);
    Возврат ВсеТесты;
КонецФункции
Процедура ТестДолжен_ПроверитьУмножениеЧисле2и2() Экспорт
    Утверждения.ПроверитьРавенство(4, ВыполнитьУмножение());
КонецПроцедуры
Функция ВыполнитьУмножение() Экспорт
    Возврат 2*2;
КонецФункции
```

Во втором листинге приведен код с тривиальным тестом. Практического смысла в нем нет (мы знаем, что $2 * 2$ будет четыре), но он прекрасно подходит для понимания основ тестирования. Итак, в одном файле может быть объявлено несколько разных тестов. Все они (имена функций с тестами) должны быть перечислены в массиве, который возвращает функция «ПолучитьСписокТестов()». Во

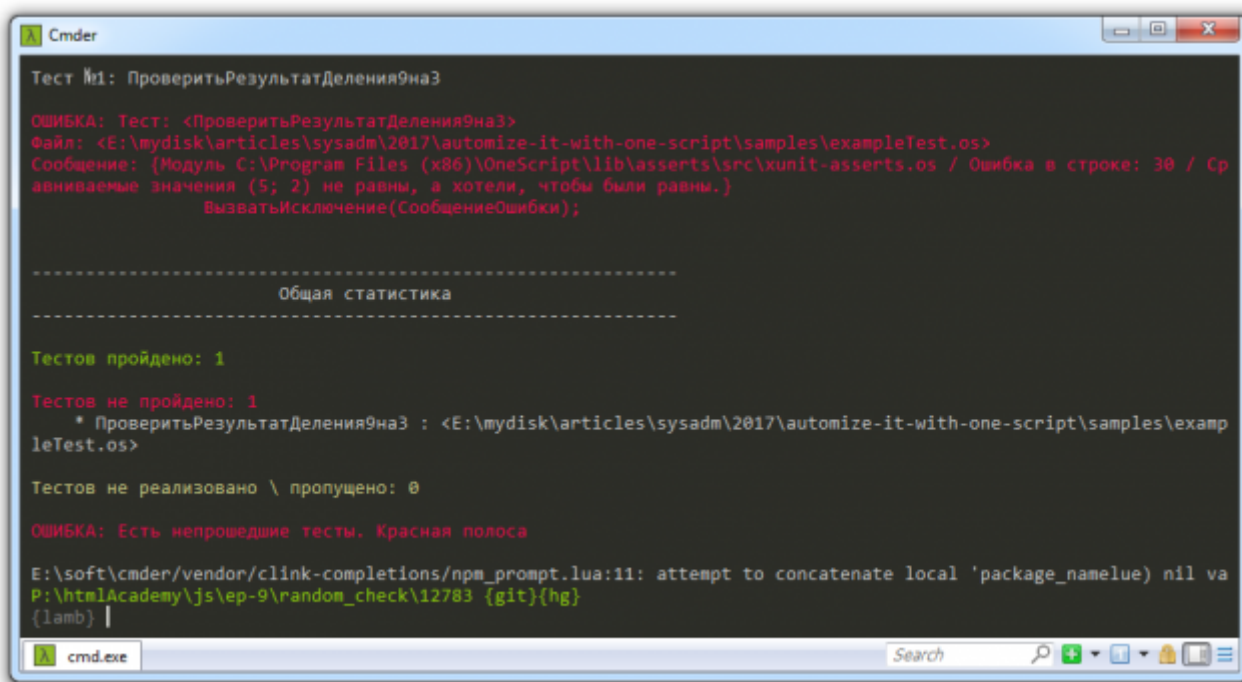
втором листинге описывается один тест – «ПроверитьУмножениеЧисел2и2». Имена процедур с тестами могут быть любимыми. Главное чтобы сами процедуры были описаны как экспортные.

Описание самой процедуры с тестом выполняется стандартным образом. Внутри пишете произвольный код и выполняете проверку утверждения с помощью метода «ПроверитьРавенство». В приведенном примере мы проверяем утверждение, что вызов функции умножить равняется 4. Если функция «ВыполнитьУмножение()» вернет не 4, то тест считается проваленным и надо разбираться почему это произошло.

Для запуска созданного теста в консоле требуется выполнить:

```
oscript "C:\Program Files (x86)\OneScript\lib\1testrunner\testrunner
```

По окончании тестирования результаты будут выведены в консоль, но вы можете настроить сохранение результатов в отдельный файл для последующего детального анализа. На рисунке 5 показан результат выполнения двух тестов.



Может показаться, что написание тестов пустая трата времени и сил – по факту кода приходится писать в два раза больше. Для небольших и одноразовых сценариев выгода от тестов будет не столь очевидна. Она проявится при работе с большими, активно сопровождаемыми проектами, когда внесение нового функционала может привести к неработоспособности существующего.

Больше, чем сценарий

В самом начале статьи я обмолвился возможностью запуска сценариев даже без наличия интерпретатора «OneScript». Такая возможность действительно существует. Любой сценарий вы можете преобразовать в полноценный

запускаемый файл и распространять по типу обычных приложений. Единственной зависимостью в этом случае будет .NET Framework 4. Собрать из сценария запускаемый файл можно командой:

```
oscript -make <ваш_сценарий.os> <имя_файла.exe>
```

Сценарий выполнен

«OneScript» - прекрасный инструмент для автоматизации, который стоит взять на заметку всем, чья работа связана с разработкой и сопровождением решений на базе платформы «1С:Предприятие». Эффективно решайте задачи на привычном языке, вместо траты лишнего времени на изучение синтаксиса очередного интерпретатора.

Дополнительные материалы

- <http://oscript.io>. Официальный сайт проекта OneScript.
- <https://code.visualstudio.com>. Бесплатный универсальный редактор Visual Studio Code от Microsoft.
- <https://marketplace.visualstudio.com/items?itemName=xDrivenDevelopment.language-1c-bsl>. Плагин для работы с OneScript из редактора VSCode.
- <https://github.com/xDrivenDevelopment>. Репозиторий готовых библиотек для OneScript.
- <https://github.com/BlackDrak0n/oscript-AutoUpdateIB>. Автоматизация загрузки и установки официальных обновлений

Статья опубликована в журнале "Системный администратор" (<http://samag.ru/>). Май 2017 г.