Article
Nikita Savchenko · Feb 12, 2019 12m read

How to Develop InterSystems Applications in Your Favorite IDE

" This is one of my articles which was never published in English. Let's fix it!

Hello! This article is about quite a practical way of *developing InterSystems solutions without using the integrated tools like Studio or* Atelier. All the code of the project can be stored in the form of "traditional" source code files, edited in your favorite development environment (for example, Visual Studio Code), indexed by any version control system and arbitrarily combined with many external tools for code analysis, preprocessing, packaging and so on.

The approach described in this article is suitable for any types of projects on top of InterSystems products. In my case, I developed a couple of my applications (WebTerminal, Visual Editor, Class Explorer) using this approach. This article demonstrates a development cycle which is not traditional for InterSystems, but rather is the practical one, which you may prefer to use for some of your developments.

TL;DR *Here are some examples of projects that utilize the approach described in this article:* WebTerminal, Class Explorer, Visual Editor, Entity Browser *(possibly, some other projects have picked up this idea - comment below!). If you want to check the file structure of these projects, click "Open" located right after "Repository" label, and you'll be redirected to GitHub. I've been developing these projects completely without Studio/Atelier!*

Below is a description of several simplest ways to organize such project development technique. Each method can be modified and expanded to a full-fledged tool for importing, assembling, or even debugging projects in InterSystems Caché, however, the purpose of this article is to provide the basics only and to show that it can work.

The described approach to development feature the following:

- The entire project (its source code) is located in the file system, with any arbitrary directory structure.
- The project directory is indexed by the Git version control system, has readme file, configs and scripts required for importing/compiling the project.

- The source code of the classes is in CLS format (as it appears in Studio/Atelier).
- Work on the project is carried out entirely in the file system, code writing - in any external text editor or IDE.
- The main feature of this approach is that you can connect any additional tools, for example, code preprocessing (like stub replacements at the compilation stage), front end assembly and so on.

This article will not cover ObjectScript routines, CSP and other files but only ObjectScript class files. The work with the first ones can be organized in the same way as with ObjectScript classes. When necessary, by analogy with the presented example, you can implement the support of importing ObjectScript routines yourself. Regarding CSP files, these are just files on the disk, so you don't need to import them at all. To make CSP files work with your InterSystems application, just copy them to the directory of your application.

The method described in the article does not require any additional tools and platforms, except for the installed version 2016.2+ of InterSystems Caché (Ensemble, Healthcare) or InterSystems IRIS. Additional assembly and preprocessing of the client code in this article is build with Node.JS, however, you can use any other technology you like. NodeJS is an open source and easy to use platform, which is chosen here because there are many ready-to-go packages already built for the tasks we are about to perform.

Motivation Behind Development in Non-InterSystems IDEs

The question arises, why not just continue to develop in the Studio, or switch to a "new studio", Atelier? What is the point of not using these IDEs at all?

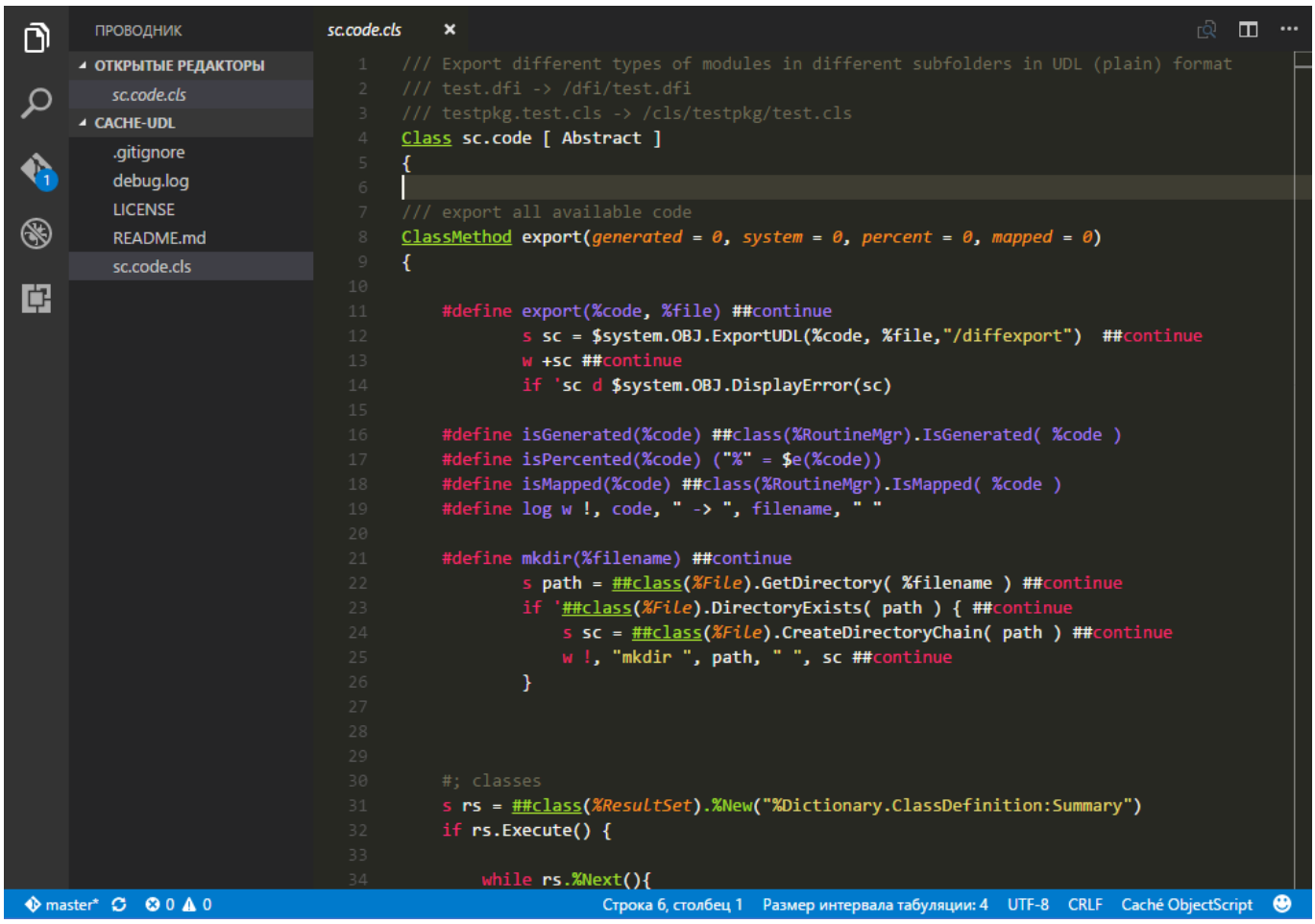
The ObjectScript programming language is very different from other common languages such as C#, Java, JavaScript, Python, Golang and others. The key difference here is that the language is "closed" by itself. Out-of-the box, many tools come directly from InterSystems, which is slowly changing with the introduction of InterSystems Open Exchange, a collection of community-created applications and tools for InterSystems products, and the politics of the corporation to make InterSystems more open. On my opinion, these changes are necessary to make ObjectScript a world-class player in the list of programming languages.

Moreover, historically, ObjectScript programs, as well as their source code, are stored directly in the DBMS itself. Before the UDL support was introduced in InterSystems Caché 2016.2 (or CDL in version 2013.2 - read below), in order to extract the source code from the database, it was necessary to write a considerable program to export plain text sources to the files, and put even more effort into getting the code back into DBMS. Now, exporting and importing plain text source code is possible with just a single command, so that you can easily organize a "traditional" model for solutions development: editing source code **files** — compiling — getting results.

Before Atelier, it wasn't simply possible to develop InterSystems applications on Linux / MacOS without a VM, since Caché Studio was supported only for Windows. Now that Atelier is based on Eclipse IDE, you can develop on any platform supported by Eclipse. However, the method described in the article is completely cross-platform.

Some projects have many other sources and files besides ObjectScript classes. The question here is how to properly organize the source code of the entire project. Today, the next development cycle is used for projects using InterSystems Tech: you work on sources in Studio/Atelier, and then you can do export XML/CLS files to a vcs-indexed file system with a help of embedded tools. These exported files are not intended for modifications.
In case of Atelier, the development cycle is designed around Atelier only, and each and every extension has to be supported by the IDE. There is a little support of external tools, build tools, code analyzers, preprocessors, as well as there is no support for an arbitrary project structure and so on. To sum up, mostly what was designed initially is supported only.

Finally, the most important motivation, taking into account all of the above, is to open ObjectScript Programming Language for the whole world. This has already began: InterSystems introduced InterSystems Open Exchange, an ObjectScript support was developed for Sublime Text Editor, Atom and Visual Studio Code, and so on. See? That's what it is about!



Syntax highlighting in Visual Studio Code - an "external" IDE for ObjectScript

Introduction

Exporting ObjectScript program sources to UDL (Universal Definition Language) format landed completely just in InterSystems Caché 2016.2. In previous versions, starting with InterSystems Caché 2013.2, the class code export was also possible, using the methods of the class %Compiler.UDL.TextServices. It is also worth mentioning that, starting from version 2016.2, the Atelier REST API is also available for importing/exporting class definitions for InterSystems products.

Before UDL, it was only possible to export and import the XML representation of classes, which was just a big mess for version control systems. Through XML class definition also had a plaintext code you could edit, you weren't able to see clean commit diffs (example: one of my projects which still has some XMLs in it), do merge requests and so on. UDL cleans this up and opens new possibilities for projects development on top of InterSystems products.

The result of this article is the simplest possible project organized entirely in the file system, and several scripts that create a single command to build and import the whole thing into DBMS.

Prerequisites

Let's assume that we have an ObjectScript project, which consists of class definitions (as well as, possibly, some routine code and a static front end). This is a necessary and sufficient condition to start applying the development method described in this article to an existing or a new project.

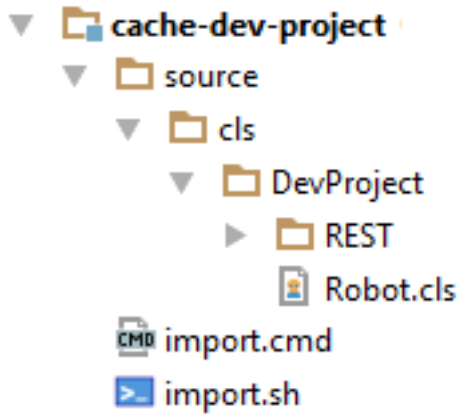
It is assumed that the machine you work on has a locally installed DBMS IRIS/Caché/Ensemble/HealthShare version 2016.2+. To implement this method of development in earlier versions

of InterSystems Caché (starting with 2013.2), you will need to adapt the suggested examples using the %Compiler.UDL.TextServices methods. If you don't have any InterSystems' products installed, you can try one out here. During the installation, specify the Unicode encoding instead of 8-bit, and leave all the other items suggested by the installation wizard unchanged.

The article uses the Git version control system. If you do not have Git installed, you have to install it.

Creating a Project

The directory structure of the demonstration project is as follows:



Wherein:

1. The source code of the project is located in the “source” directory, and, in the corresponding “cls” subdirectory, there is a tree of packages and classes. In the screenshot of the project structure above, as an example, the DevProject package is located, along with Robot class (DevProject.Robot) and REST subpackage.
2. The import.* Script imports the project into the DBMS.

The project code shown above is available on GitHub. It is suggested to clone the project to the local machine by following the instructions below:
git clone https://github.com/ZitRos/cache-dev-project

The project contains the source/cls directory, which contains the usual package hierarchy. For the demonstration, a simplest class was created containing the Message class-method, which displays the message “Welcome, Anonymous!”:

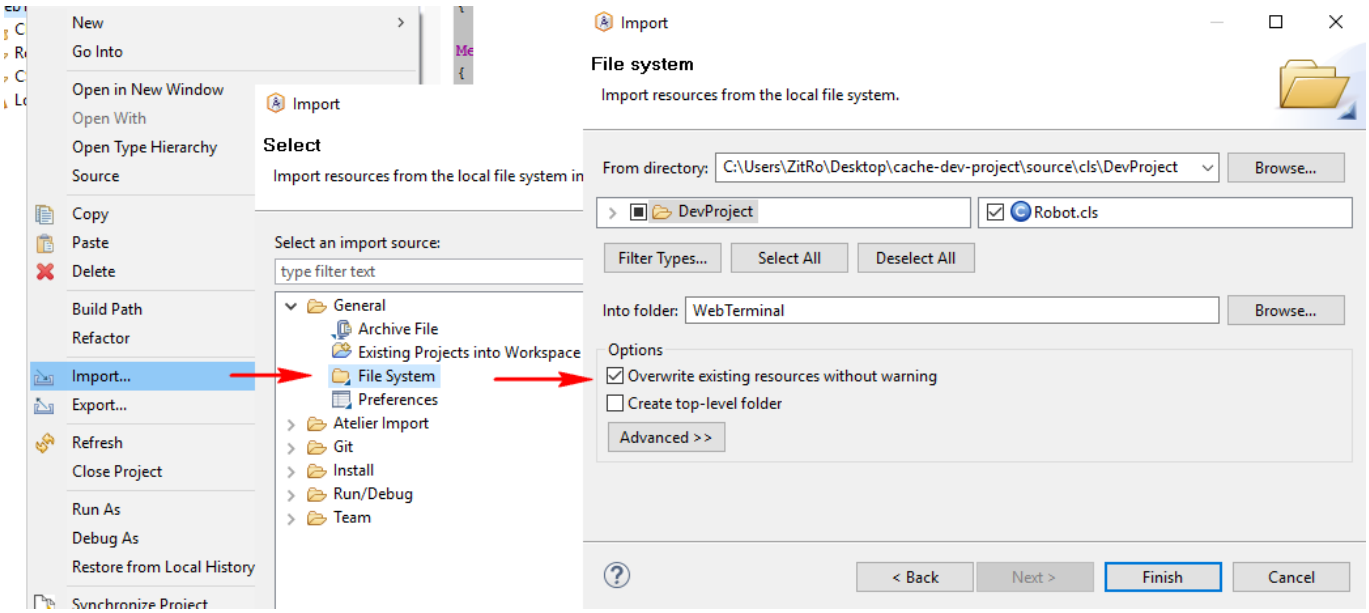
```
Class DevProject.Robot
{

ClassMethod Message(name As %String = "Anonymous")
{
   write "Welcome, ", name, "!"
}

}
```

To import this and other classes into DBMS, you can use one of the following ways:

1. Use Atelier:



It doesn't make sense to perform all these steps each time we would like to test our project. Hence, we are going to automate this.

2. Execute the following command in the terminal window:
do $system.OBJ.ImportDir("D:/Path/To/the/Project/source/cls",,"ck /checkuptodate=all",,1)

This command recursively loads all files from the D:/Path/To/the/Project/source/cls directory into the current namespace, and also compiles those classes that have changed since the last import. Thus, reloaded classes without changes will not take time to compile.

The second option is also isn't the most convenient solution - every time a project starts, you need to open a Caché terminal, enter a login-password pair (on instances with a normal protection level enabled), switch to the desired namespace and finally execute the command saved somewhere in a notebook. Finally, it is possible to automate this using the third option.

3. Create a script to automate all the routine stuff and use just this:
import

Calling the latter command in the case of development in almost any external IDE can be simplified even more, to the click of a single button or running a program which will watch files and re-import each time something changes.

Thus, the entire project is in the file system, work is being done with the plaintext files, and if necessary, just a single command imports and compiles the whole project without a hassle.

The Import Script

Let's take a closer look onto a script that imports a project into DBMS. In order to do this, it needs some additional information about your InterSystems instance, namely, it's install location, import namespace, as well as the username and password to log in to the system. This data is coded directly into the script, however, it could be separated to a config file.

The source code of the script is available on GitHub for Windows and *nix systems. All that needs to be done is several variables change in the script once before starting work on the project.

The script executes the cache.exe executable file, which is located in the /bin/ directory of the installed DBMS, and passes two arguments to it: the database directory and the namespace. Then, the script sends a user name, a password, and a few simple ObjectScript commands to the input interface via the terminal interface, importing classes and reporting a successful import or error.

Thus, the user gets all the necessary information about the import and compilation of the classes, as well as any errors that may have occurred during the compilation process. Here's the example of the output of the import.bat script:

```
Importing project...
Node: DESKTOP-ILGFMGK, Instance: ENSEMBLE20162

USER>
Load of directory started on 06/29/2016 22:59:10

Loading file C: \ Users \ ZitRo \ Desktop \ cache-dev-project \ source \ cls \ DevProject \ Robot.cls as udl
Loading file C: \ Users \ ZitRo \ Desktop \ cache-dev-project \ source \ cls \ DevProject \ REST \ Index.cls as udl

Compilation started on 06/29/2016 22:59:10 with qualifiers 'ck /checkuptodate=all'
Class DevProject.REST.Index is up-to-date.
Compiling class DevProject.Robot
Compiling routine DevProject.Robot.1
Compilation finished successfully in 0.003s.

Load finished successfully.
IMPORT STATUS: OK
```

Now we can ensure that the project was indeed imported:

```
USER > do ##class(DevProject.Robot).Message()
Welcome, Anonymous!
```

More Complex Example

To maximize the benefits of developing a project using InterSystems technologies, we will try to do something more attractive by adding a graphical interface and building the project with the use of NodeJS platform and Gulp task runner. The result is a web page which image is shown below.

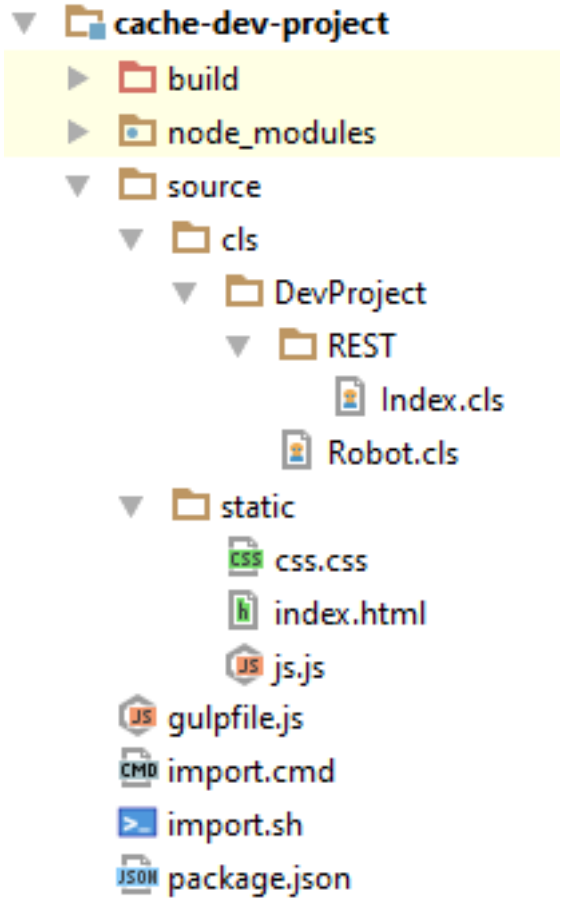
Emphasis will be placed on how it is possible to organize the development of such a project. First let's took at

architecture of the suggested solution.

The project consists of static client code (HTML, CSS, JavaScript), a class on the server that describes the REST interface, and one persistent class.

A client with a GET request gets a list of robots that are located on the server. Also, when you click on the "Spawn a new robot!" button, the client sends a GET request to the server, as a result of which a new instance of the Robot class is created and added to the display list. *(note: robot creation request should actually be a POST request, but we won't complicate things much in this example)*

Technical implementation of the project can be viewed on GitHub (in "extended" branch). In the article, further attention will be paid to the method of developing such projects.



Here, unlike the previous example, the client part of the application is added, which is located in the source/static directory, and the project is built using Node.JS and Gulp.

For example, besides the other code in the project there you can find some special comments like these:

v<!-- @echo package.version -->

When building a project, this comment will be replaced with the project version, which is listed in the package.json file. The build script also minimizes CSS and JavaScript code, and copies all the processed code into the `build` directory.

In the import script, unlike the previous example, the following changes were added:

1. Before importing, the project is assembled (bundled).
2. Import files are now imported from the build directory, as they pass through the preprocessor.
3. The files are copied from the build/static directory to the Caché's csp/user directory with CSP files. Thus, after importing the application it immediately becomes available.

Detailed instructions for installing and running this project are available in the description of the repository.

The result is a project that needs to be set up only once, by modifying several variables in the import.* file.

The considered development cycle is used in several of my own projects: WebTerminal, Class Explorer, Visual Editor, Entity Browser. Soon it may be used in other projects, including your own ones :)

IDE and Debugging

This development method does not provide any debugging utilities, it is simple as that. If you use more comprehensive debugging tools for your ObjectScript code rather than simply logging something to globals, you still have to use integrated debugging tools in InterSystems products.

However, besides of that, the described development method has a big advantage: the ability to use your favorite development environment for writing ObjectScript code, whether this is vim or a simple notebook, MacOS or *nix or any other programming language is used - you get the same workflow. But on the other hand, ObjectScript does not feature such a comprehensive support outside the Studio/Atelier. This means that even syntax highlighting is currently not quite well handled by external editors, let alone autocompletion. But all this is about to change in the near future, as more and more effort is being put into the open-source initiatives.

In the meantime, you can use elementary syntax highlighting by keywords that some IDEs offer, such as IntelliJ or Visual Studio Code:

```
/// Sample REST method
ClassMethod Index() As %Status
{
    set result = {}
    set resultSet = ##class(%ResultSet).%New("DevProject.Robot:Extent")
    do resultSet.Execute()
    while resultSet.Next() {
        set obj = ##class(DevProject.Robot).%OpenId(resultSet.Get("ID"))
        set $PROPERTY(result, obj.%Id()) = {
            "Name": obj.Name,
            "Energy": obj.Energy,
            "Message": obj.GetMessage()
        }
    }
    do resultSet.Close()
    write result.$toJSON()
    return $$$OK
}

ClassMethod Spawn() As %Status
{
    set status = ##class(DevProject.Robot).Spawn(%request.Get("number", 1))
    write { "status": status }.$toJSON()
    return status
}
```

In the case of IntelliJ IDEA is your favorite IDE, you can try it right now - here is the settings file you need to import using the File -> Import Settings menu. Highlighting is quite simple and incomplete, any additions are welcome.

Conclusion

The purpose of this article is to introduce something new into the world of development of InterSystems applications, present another version of development to the public and contribute to the spread of ObjectScript as a programming language as a whole. Any feedback, ideas and discussions are very welcome!

Thank you!

#Continuous Delivery #Continuous Integration #Development Environment #JavaScript #JSON #Node.js #Object Data Model #ObjectScript #Performance #UI Development #Frontend #SOAP #Atelier #Caché #Ensemble #HealthShare #InterSystems IRIS #InterSystems IRIS for Health #Open Exchange

Source URL: https://community.intersystems.com/post/how-develop-intersystems-applications-your-favorite-ide