Behind the scene of isc-tar project

Dmitriy Maslennikov 18 March 2019

Development Environment, DevOps, Docker, VSCode, InterSystems IRIS

I am just recently announced my project isc-tar. But sometimes it is not less interesting what's behind the scene: how it was built, how it works and what happens around the project. Here is the story:

- How to develop this project
- How to test it
- How to release new versions for publishing
- And finally how to automate all above
- Continuous integration

So, I would like to tell all about it.

Development

My favourite tools in recent times are Docker and VSCode. I use Docker to start any of my projects in its own environment and VSCode just the best editor ever, which I use together with the extension vscode-objectscript. In any new project, I use at least both of these tools. I'm also a macOS user, so all the next steps tested and should perfectly work in macOS and maybe on Linux as well, but could be different for Windows.

- First of all, clone our project to some place

  ```
  $ git clone git@github.com:daimor/isc-tar.git
  ```

- I have the command code in my shell, so I can easily open any folder from my terminal with the code. If you want this feature as well, you can set it from the command palette. You have to reload your shell, to get this command available

- Open editor with just cloned repository

  ```
  $ code isc-tar
  ```

- VSCode has the option of an integrated terminal. So you can now open it and you will not lose your terminal, which will be very close all the time. And we can use the terminal to setup project environment with the following two commands:

  ```
  $ docker-compose build --no-cache
  $ docker-compose up -d
  ```

it may take some time.

- And you already able to code, this repository contains settings for VSCode, so, it is already preconfigured to use this IRIS.

Testing

The project has a couple of tests, so, you are able to run it:

- in the terminal go to IRIS session

```
$ docker-compose exec iris iris session iris

USER>do ##class(%UnitTest.Manager).RunTest()
```

BTW, have you noticed 3 ‘iris’ in the command? I'm not that IRIS fun who likes to repeat it: and you really have to put iris three times to launch IRIS terminal shell. First is for service name in docker-compose, next it is the command iris inside the container (replacement of ccontrol which was with Caché), and the last one is for instance name. I wish InterSystems will add a shorter command `session`, to replace this unnecessary repetition of iris word. And get something like this

```
docker-compose exec iris session
```

Publish

As I said in the announce, this tool may work in earlier versions of Caché and Ensemble too. But not all of those versions support UDL as the import format. So, we need XML which can be used everywhere on IRIS or Caché/Ensemble. Fortunately, I have only one class and can do a simple export.

```
do $system.OBJ.Export("%zUtils.FileBinaryTar.cls", "/opt/zUtils.FileBinaryTar.xml", "
/diffexport")
```

Unfortunately, for unknown reasons, I can't use a useful qualifier /exportversion=2010.1

```
USER>do $system.OBJ.Export("%zUtils.FileBinaryTar.cls", "/opt/zUtils.FileBinaryTar.xml", "/diffexport/exportversion=2010.1")

Exporting to XML started on 03/16/2019 09:18:35
Exporting class: %zUtils.FileBinaryTar
ERROR #5126: XML export version '2010.1' not supported, supports 2010.1 and onwards.
Errors detected during export.
```

In exported XML I need to replace Export header to make it recognizable for Caché as well.

```
<Export generator="IRIS" version="26">
```

And I use command sed for it.

```
sed -i.bak 's/^<Export generator="IRIS" .*$/<Export generator="Cache" version="25">/g' /opt/zUtils.FileBinaryTar.xml
```

Now it can be used on Caché and IRIS both.

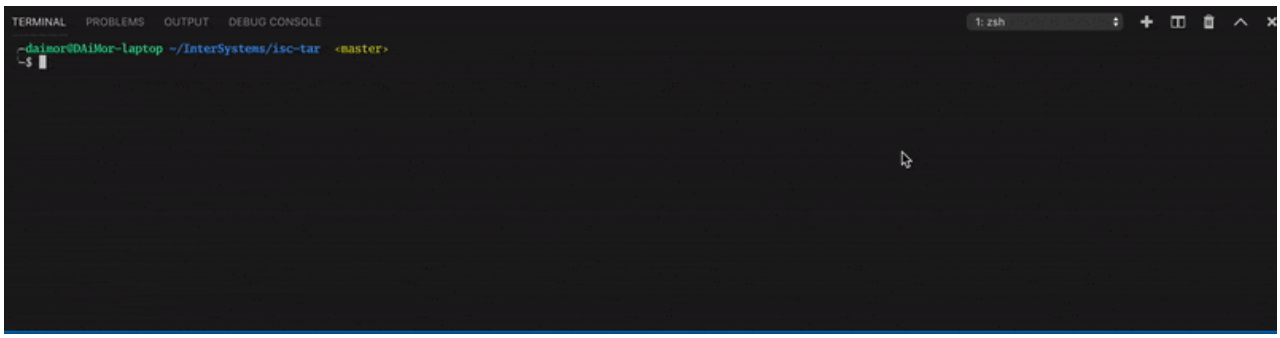
Automation

Running tests and building releases needs you to remember and call long commands, so it would be nice to simplify it as much as possible. And Makefile perfectly suitable for it.

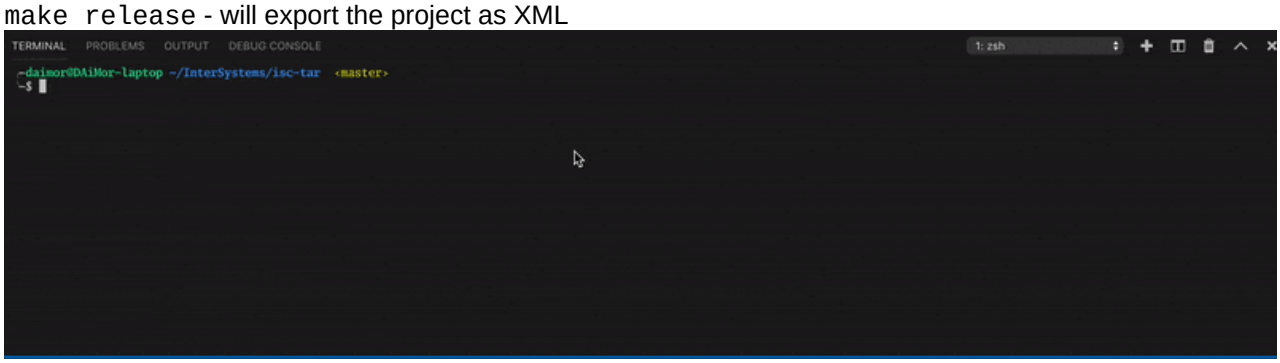
```
APP_NAME = isc-tar
IMAGE ?= daimor/$(APP_NAME)

SHELL := /bin/bash

.PHONY: help build test release

help: ## This help.
    @awk 'BEGIN {FS = ":.*?## "} /^[a-zA-Z_-]+:.*?## / {printf "\033[36m%-30s\033[0m %s\n", $$1, $$2}' $(MAKEFILE_LIST)

.DEFAULT_GOAL := help

build: ## Build the image
    docker build -t $(IMAGE) .

test: build ## Run UnitTests
    docker run --rm -i -v `pwd`/tests:/opt/tests -w /opt --entrypoint /tests_entrypoint.sh $(IMAGE)

release: clean build ## Export as XML
    docker run --rm -i -v `pwd`/out:/opt/out -w /opt --entrypoint /build_artifacts.sh $(IMAGE)

clean:
    -rm -rf out
```

My Makefile is quite simple, I have two primary recipes

- `make test` - will run Unit Tests



- `make release` - will export the project as XML



Both of these recipes depend on build which builds the IRIS image with the project inside.

Continuous Integration

Of course, my article would not complete without it but stay tuned for the next part, I'm sure you will be surprised how it was done.

- + 2
- 2
- 1
- 317
- 0

Dmitriy Maslennikov 18 March 2019

Reply

Source URL: https://community.intersystems.com/post/behind-scene-isc-tar-project