Article
Eduard Lebedyuk · Jul 13, 2022 7m read

Continuous Delivery of your InterSystems solution using GitLab - Part X: Beyond the code

After almost four years on hiatus, my CI/CD series is back! Over the years, I have worked with several InterSystems clients, developing CI/CD pipelines for different use cases. I hope the information presented in this article will be helpful to someone.

This series of articles discusses several possible approaches toward software development with InterSystems technologies and GitLab.

We have an exciting range of topics to cover: today, let's talk about things beyond the code - namely configurations and data.

Issue

Previously we discussed code promotions, and that was, in a way, stateless - we always go from a (presumably) empty instance to a complete codebase. But sometimes, we need to provide data or state. There are different data types:

- Configuration: users, web apps, LUTs, custom schemas, tasks, business partners, and many more
- Settings: environment-specific key-value pairs
- Data: reference tables and such often must be provided for your app to work

Let's discuss all these data types and how they can be first committed into source control and later deployed.

Configuration

System configuration is spread across many different classes, but InterSystems IRIS can export most of them into XMLs. First of all, is a Security package containing information about:

- Web Applications
- DocDBs
- Domains
- Audit Events
- KMIPServers
- LDAP Configs
- Resources
- Roles
- SQL Privileges
- SSL Configs
- Services
- Users

All these classes provide Exists, Export, and Import methods, allowing you to move them between environments.

A few caveats:

- Users and SSL Configurations might contain sensitive information, such as passwords. It is generally NOT recommended to store them in source control for security reasons. Use Export/Import methods to facilitate one-off transfers.
- By default, Export/Import methods output everything in one file, which might not be source control friendly. Here's a utility class that can export and import Lookup Tables, Custom Schemas, Business Partners, Tasks, Credentials, and SSL Configuration. It exports one item per file, so you get a directory with LUT, another directory with Custom Schemas, and so on. For SSL Configurations, it also exports files: certificates and keys.

Also worth noting that instead of export/import, you can use %Installer or Merge CPF to create most of these. Both tools also support the creation of namespaces and databases. Merge CPF can adjust system settings, such as global buffer size.

Tasks

%SYS.Task class stores tasks and provides ExportTasks and ImportTasks methods. You can also check the utility class above to import and export tasks one by one. Note that when you import tasks, you can get import errors (ERROR #7432: Start Date and Time must be after the current date and time) if StartDate or other schedule-related properties are in the past. As a solution, set LastSchedule to 0, and InterSystems IRIS would reschedule a newly imported task to run in the nearest future.

Interoperability

Interoperability productions contain:

- Business Partners
- System Default Settings
- Credentials
- Lookup Tables

The first two are available in Ens.Config package with %Export and %Import methods. Export Credentials and Lookup Tables using the utility class above. In recent versions, Lookup Tables can be exported/imported via $system.OBJ class.

Settings

System Default Settings - is a default interoperability mechanism for environment-specific settings:

> The purpose of system default settings is to simplify the process of copying a production definition from one environment to another. In any production, the values of some settings are determined as part of the production design; these settings should usually be the same in all environments. Other settings, however, must be adjusted to the environment; these settings include file paths, port numbers, and so on.
>
> System default settings should specify only the values that are specific to the environment where InterSystems IRIS is installed. In contrast, the production definition should specify the values for settings that should be the same in all environments.

I highly recommend making use of them in production environments. Use %Export and %Import to transfer system default settings.

Application Settings

Your application probably also uses settings. In that case, I recommend using System Default Settings. While it's an interoperability mechanism, settings can be accessed via: %GetSetting(pProductionName, pItemName, pHostClassName, pTargetType, pSettingName, Output pValue) (docs). You can write a wrapper which would set the defaults you don't care about, for example:

```
ClassMethod GetSetting(name, Output value) As %Boolean [Codemode=expression]
{
##class(Ens.Config.DefaultSettings).%GetSetting("myAppName", "default", "default", ,
name, .value)
}
```

If you want more categories, you can also expose pItemName and/or pHostClassName arguments. Settings can be initially set by importing, using System Management Portal, creating objects of Ens.Config.DefaultSettings class, or setting ^Ens.Config.DefaultSettingsD global.

My main advice here would be to keep settings in one place (it can be either System Default Settings or a custom solution), and the application must get the settings using only a provided API. This way application itself does not know about the environment and what's left is supplying centralized setting storage with environment-specific values. To do that, create a settings folder in your repository containing settings files, with file names the same as the environment branch names. Then during CI/CD phase, use the $CI_COMMIT_BRANCH environment variable to load the correct file.

```
DEV.xml
TEST.xml
PROD.xml
```

If you have several settings files per environment, use folders named after environment branches. To get environment variable value from inside InterSystems IRIS use $System.Util.GetEnviron("name").

Data

If you want to make some data (reference tables, catalogs, etc.) available, you have several ways of doing it:

- Global export. Use either a binary GOF export or a new XML export. With GOF export, remember that locales on source and target systems must match (or at least global collation must be available on the target system). XML export takes more space. You can improve it by exporting global into an xml.gz file, $system.OBJ methods automatically (un)archive xml.gz files as required. The main disadvantage of this approach is that data is not human-readable, even XML - most of it is base64 encoded.
- CSV. Export CSV and import it with LOAD DATA. I prefer CSV as it's the most storage-efficient human-readable format, which anything can import.
- JSON. Make class JSON Enabled.
- XML. Make class XML Enabled to project objects into XML. Use it if your data has a complex structure.

Which format to choose depends on your use case. Here I listed the formats in the order of storage efficiency, but that's not a concern if you don't have a lot of data.

Conclusions

State adds additional complexity for your CI/CD deployment pipelines, but InterSystems IRIS provides a vast array of tools to manage it.

Links

- Utility Class
- %Installer
- Merge CPF
- $System.OBJ
- System Default Settings

#Continuous Delivery #InterSystems IRIS

Source URL:https://community.intersystems.com/post/continuous-delivery-your-intersystems-solution-using-gitlab-part-x-beyond-code