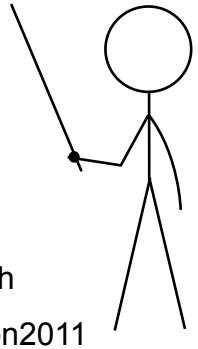
# Getting Eclipse Preferences Under Control In Teams



Michael Pellaton
michael.pellaton@netcetera.ch

downloads: https://contrails.ch/econ2011

EclipseCon 2011 - Santa Clara, CA

## Agenda

- README
- Introduction
  - Preferences in Teams
  - Configuration Everywhere
- Solutions
  - Basic Configuration Approach
  - The Wiki Way
  - The EPF Way
  - Eclipse Team Etceteras
  - Workspace Mechanic
  - Bug 334016
- Comparison
  - Feature Comparison
  - When to Use Which?
- Links and Resources

# README

## Whoami

Michael Pellaton <michael.pellaton@netcetera.ch>
Senior Software Engineer at Netcetera Zürich, CH

## Handout

https://contrails.ch/econ2011

## Disclaimer

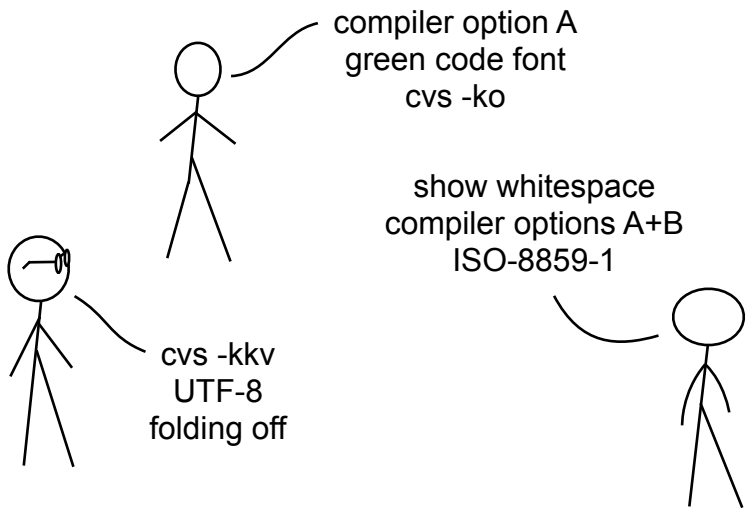
The speaker is the maintainer of the Eclipse Team Etceteras project and might therefore be biased.

Only freely available solutions were considered.

## Legal

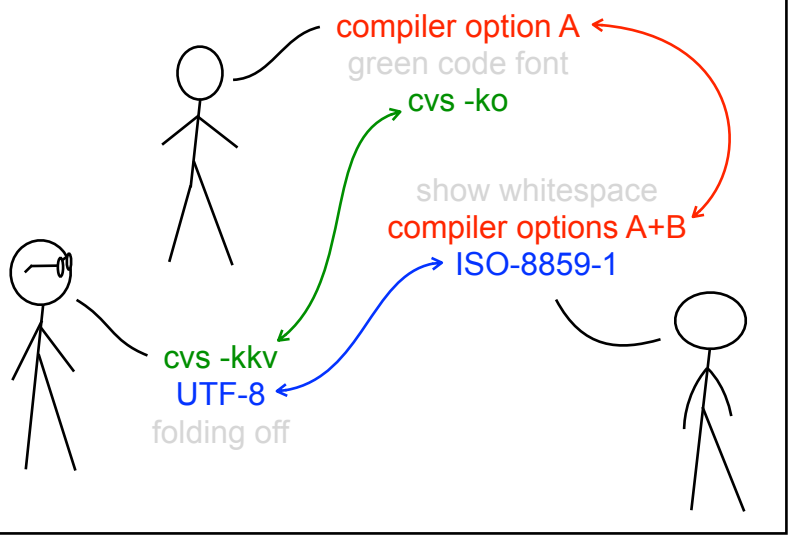


# Preferences in Teams




In a team and even among multiple workspaces, one tends to have very different preferences. Basically, that's perfectly fine and in most cases the users may configure Eclipse to their liking.

## Preferences in Teams




However, there are a number of settings that need to be coordinated to make sure the project doesn't run into technical troubles and inconsistencies. Examples for that type of preferences are:
- file encoding
- SCM settings like (content types, auto properties, ...)
- compiler options (same code might behave differently on different machines)
- …

Settings that adapt/hook up Eclipse to the local development environment are best coordinated for efficiency. Examples for that group of preferences are:
- SCM repositories
- Mylyn task repositories
- file templates
- update sites
- …

## Eclipse Configuration Everywhere

**Eclipse distribution**

```
eclipse
 |- eclipse.ini
 |- configuration
    |- config.ini
    |- …
```

**Equinox Secure Storage**

```
$home/.eclipse
```

**Preferences**

```
$workspace
  |- ./metadata/…
  |- $project/.settings/…
```

**Runtime options**

```
eclipse -data … -vm … -vmargs …
```

**This talk is about preferences only!**

There is not a single configuration file for Eclipse and the configuration is spread over several files and scopes. There are even several different mechanisms and APIs for different types of configuration and in some cases configuration on one place overrides the same configuration in another place (e.g. workspace vs project level; command line arguments vc eclipse.ini). The most common places where Eclipse can be configured in some way are:
- eclipse.ini: system properties, JVM options and program arguments
- preference store: the majority of preferences is stored here. This is a workspace level store.
- secure store: store for passwords and other sensitive information. If install directory is writable, it is stored in the eclipse directory
- some extensions have their own mechanisms

## Basic Configuration Approach

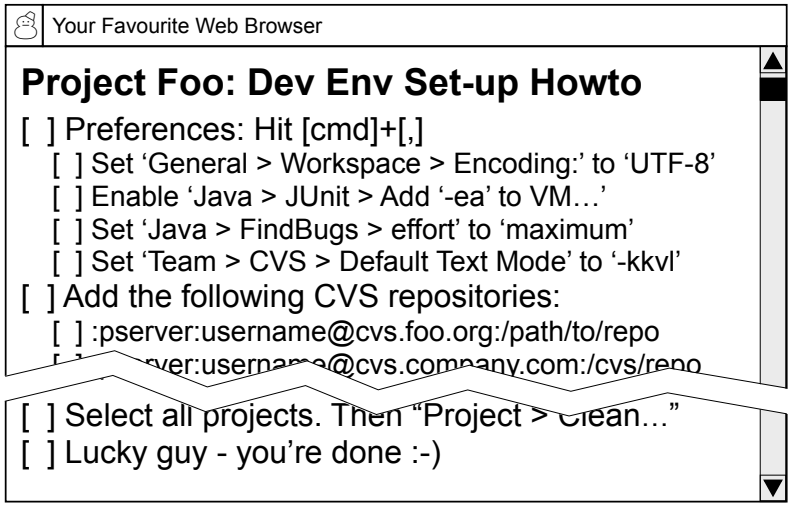
**Configure as much as possible on project level and ‘somehow’ manage the configurations that can’t be put into the project.**

**Let’s have a look at how ‘somehow’ could be...**

Put as much as possible on the project level:
- configuration is managed in the SCM
- updates are distributed to developers as normal SCM updates
- different projects may have different configurations (java version, different license headers, different coding styles for different customers) and still be checked out in the same workspace
- different project types may have different configurations
- tools run during the continuous/release build can be configured to use the same config (e.g. FindBugs, Checkstyle, PMD)

Disadvantages of putting preferences into the project:
- Changes in multi module projects must be replicated to the projects (If you use Maven, the Maven eclipse plugin can help here)
- Some developers complain about the number of .-files in the projects

## The Wiki Way



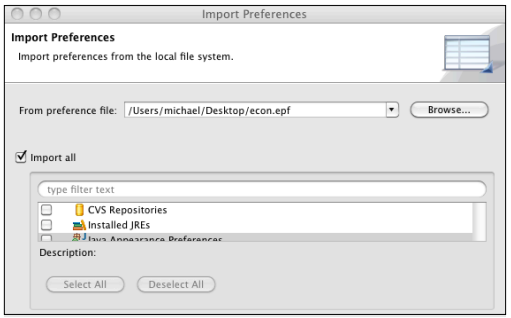

I am sure you know at least one project having one of these famous "How to set up the devenv" wiki pages or readme files.

One one hand, they solve the problem of preference synchronization. On the other hand they are broken for some reasons:
- People don't know where the document is
- Going through an endless checklist is a mundane task and people tend to do errors
- Keeping this document up-to-date is a pain
- Delta-checklists are needed if there are changes

## The EPF Way

### Shared Eclipse Preference File (EPF)

### Let users import the EPF manually using the import wizard



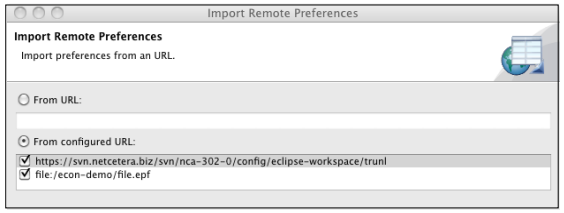

Eclipse offers a preference import / export mechanism that stored all preferences from the preference store in a so called Eclipse Preference File (EPF).

Providing a common EPF file on a shared file system simplifies the configuration for the users compared to the checklist. The problems about this solution are:
- people don't know where the file is
- the import must be done manually
- after changes, the developers need to re-import the file
- on each import, the user needs to know the location of the file
- complicated to maintain in SCM

## Eclipse Team Etceteras

### Automatic and manual EPF import over HTTP

### Preference transfer between workspaces




Eclipse Team Etceteras is a set of small enhancements to existing Eclipse preference functionality that ease the use:
- import EPF files over HTTP
- preconfigure the EPF URLs so that the users don't need to find the file
- remind developers when they create a new workspace to apply the default configuration
- Ad-hoc EPF import over HTTP (paste URL into import wizard)
- transfer preferences from one workspace to another

Disadvantages:
- limited to what EPFs can do
- no enforcement capabilities
- pre-configuration is done using a fragment bundle: need to deploy an own configuration plug-in along with Eclipse

## Workspace Mechanic

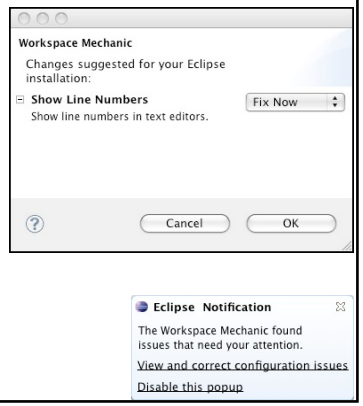
**Task oriented configuration engine**

**A task can be**

- a Java class
- a Groovy script
- an EPF file
- a Plug-in (extension point)

**Frequent checks**

**(File system based)**




Workspace Mechanic is a very powerful task oriented tool to administer Eclipse configurations. A task can be a Java class, a Groovy script, a Plug-in or an EPF file. Importing EPF files is just one of many possible tasks. You can extend Workspace Mechanic by providing new task and new task types using extension points. Tasks are picked up from shared file system locations by Workspace Mechanic in a configurable frequency. If anything needs to be done, the user is prompted to apply a so called 'Fix' in a notification.

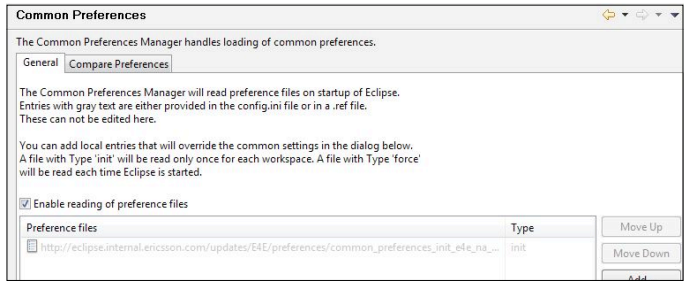
Workspace Mechanic offers some nice tools that make the lives of the administrators easier: creating a EPF file for a task is very simple thanks to the preference recorder.

Disadvantages:
- no enforcements
- a shared file system is required (as of now), HTTP capability is committed but not yet released
- pre-configuration is done in plugin_customization.ini or an EPF file: need to deploy that file along with Eclipse or being imported

## Bug 334016

**Automatic EPF import over HTTP**

**Preference lock-down and enforcement capability**




In Bug 334016 Ericsson donated a preference management tool. As there seems to be no official name, it is referred to under it's Bugzilla ID. At the time of the writing of these slides it was unclear whether, when and how the contribution will be included into Eclipse (according to C. Aniszczyk). There is no update site to install this functionality from - interested users need to get the source out of the attachments of this bug and build/release the plug-in themselves.

The features offered:
- import EPFs over HTTP
- lock-down of preferences
- automatic imports upon workspace startup (enforcement)

Disadvantages:
- not easily available from an update site or Eclipse Market Place
- limited to what EPFs can do

## Feature Comparison

| | ETE | Ws Mech | 334016 |
|---|---|---|---|
| EPF imp. over HTTP | ✔ | (✔) | ✔ |
| More than EPF | | ✔ | |
| Change Detection | | ✔ | ✔ |
| Enforcement/Lock | | | ✔ |
| Admin Tool Support | (✔) | ✔ | ✔ |
| On Marketplace | ✔ | | |
| License | EPL | EPL | EPL |

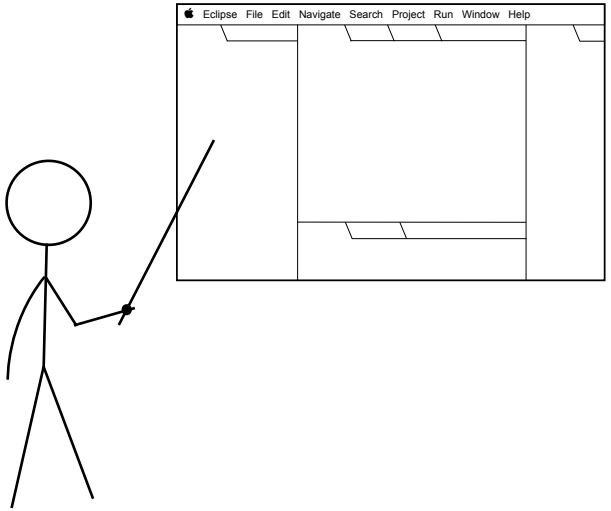
## When to Use Which?

**If EPF import over HTTP is all you need: Use ETE**

**If you need more complex stuff than EPF imports: Use Workspace Mechanic**

**If you need preference lock-down and enforcement: Use Bug 334016**

## ETE Demo



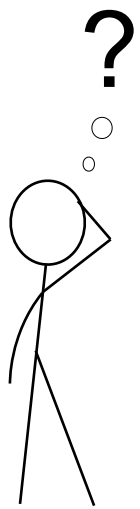

Demo of Eclipse Team Etceteras:

1. Start Eclipse with new workspace
--> ETE new workspace detection and preference import

2. Change workspace name (General > Workspace > Name)

3. File > Switch Workspace… (to a new workspace including preferences transfer)
--> in new workspace: show transferred preference (workspace name)

## Questions



# Links and Resources

**Presentation and other Resources**

https://contrails.ch/econ2011

**Discussed Projects**

Eclipse Team Etceteras

Workspace Mechanic

Bug 334016

**Eclipse Configuration and Preferences Resources**

User Settings: FAQ

Eclipse Wiki: Eclipse.ini

Eclipse Help: Provisioning Actions and Touchpoints

Eclipse Help: The Eclipse runtime options

Eclipse Preferences - Tutorial (by Lars Vogel)

Eclipse Help: Equinox Secure Storage

Extension Point Description: Preference Transfer

EclipseCon 2010 Talk: Eclipse in the Enterprise: Lessons from Google