Cordon

License Control Operator

4.1.0

User Guide

ArSysOp [1]

March 16, 2025

Contents

Chapter 1

Install, launch and maintain LCO

ArSysOp Cordon License Control Operator issues licenses for products, protected by *Eclipse Passage* technology [4]. Besides, the tool does all the work around this task:

- helps *Licensing Operator* to declare all the data necessary to define a license, facilitates storing and indexing for the data;
- keeps and manages issued licenses and cryptography keys;
- provides analysis and reporting instruments.

1.1 Get distributive

ArSysOp Cordon LCO product usually shipped as simple archive, accessible for download at ArSysOp Cordon LCO official site [2]. Distributive is available for Windows and Linux operating systems.

To install *ArSysOp Cordon LCO* unzip the archive to any installation directory on a local file system. For instance, it can be (Windows example)

> D:/Program Files/ArSysOp/Cordon LCO 4.1.0/

1.2 Launch LCO

ArSysOp Cordon LCO can run

- as a regular standalone application - use *lco.exe* in the root of your installation directory

- as an application with console output - use *lcoc.exe*.

1.3 Get license for your LCO

ArSysOp Cordon LCO itself is a product protected by Eclipse Passage licensing system, so it demands a license in order to be used. To get one

- launch LCO
- see *License Status dialog*, which informs whether or not the installation is licensed properly and offers options to proceed.
- press **Request** button to gather hardware information necessary to request a license. Your workstation's hardware is assessed and the results are delivered as a file of **.assessment_xmi* format.
- write a letter to *support@arsysop.ru* with a license request, it should contain the following input:
  - product name: *ArSysOp Cordon LCO*,
  - product version: 4.1.0,
  - assessment results: attach the file you've just gained by hardware assessment process.
- use **Import...** to apply gained license to the installation of *ArSysOp Cordon LCO*. Point to a directory where the license resides.
- use **Accept...** option to read and accept *ArSysOp Cordon LCO End User License Agreement*

1.4 Support

In case of any trouble with the program contact *support@arsysop.ru*, describe the issue and attach the logs.

ArSysOp Cordon LCO keeps logs in *<lco-installation-directory>/workspace/.metadata/.log* file and optionally shares them in console, if launched with it.

There is also an option to open a issue ticket under *ArSysOp Cordon Support* repository [3].

Chapter 2

Operator Workspace

Operator Workspace is the main tool for *Licensing Operator*. It facilitates creation, evolution, management and maintenance for all the data required for license issuing in some *global licensing scope*.

In the aspect of User Interface it consists of three parts

- left: **License Projects Navigator** called *License Projects* exposes file-based structure of your data. A project file encapsulates data objects.
- center: **Editor** works with a particular file: shows data objects inside the file, facilitate changes.
- right: **Property Sheet** gives grained control over each manageable fragment of a data object.

Operator Workspace distinguishes two substantial parts to any licensing work:

- definitions (**meta**) to describe evolution of a product as a whole and as a set of features. It talks in the following concepts:
  - **Product**s under licensing;
  - **Feature**s to these **Product**s, that are to be protected by license;
  - **Version**s for both **Product**s and **Feature**s to reflect evolution and to represent an actual functionality shipped in scope of the product distributive.

- operative information (**data**), which keeps track on physical artifacts and surrounding context for licenses issuing process and outcome:
  - manages license templates;
  - accounts your **User**s;
  - facilitates license issuing for **User**s;
  - keeps issued licenses in a structured way.

This perspective demands for your workspace to be of particular structure. Start with creating two **Project**s, named after your *global licensing scope*. Most often this scope is named after company that owns a product under licensing.

2.1 Meta

Use *File/New/Project -> General/Project*, name the project **mycompany-meta**. This *-meta* postfix is mandatory. Into this project create two folders:

- *products* to store **Product Line**s filled with **Product**s and theirs **Product Version**s,
- *features* to define **Feature Set**s filled with **Feature**s and **Feature Version**s.

Use popup menu *New/Folder* action

2.2 Data

Use *File/New/Project -> General/Project*, name the project **mycompany-data**. We'll be needing the following folders structure:

- *users*
- *licenses*
- *keys*
- *issued*
- *agreements*

Chapter 3

What is to be protected by license

3.1 Product Line

For management purposes **Product**s are organized into collectives named **Product Line**s. Each **Product Line** is stored in its own file under local file system. Use this original packaging strategy to facilitate easy data exchange, version control invocation and to keep tabs on granularity.

To create use *File/New/Product Line...* main menu action. Select *mycompany-meta/products* project folder to persist the file, name it correspondingly to the planned products. For instance, if to use popular area of making useful programs - reporting, so **Product Line**'s file name could be *mycompary.reporting.products_xmi*.

Now open the **Product Line** file in **Editor** (double click on the corresponding file in **License Projects Navigator**), and fill its properties in **Property Sheet**.

3.2 Product

Product data object stays for a physical product in it entirety that is going to be developed, evolve to particular versions, each of which is to be delivered to end user. To create a **Product**

- open its owning **Product Line** in **Editor**;
- in **Editor** use context menu for the **Product Line**then call *New Child/Product*;
- fill properties for the **Product** in **Property Sheet**.

3.3 Product Version

Product Version data object represents some particular manure state in life of a **Product**. **Product Version** actually has a distributive, which is delivered to end user and, subsequently, demands from the user to have a license for it.

To create a **Product Version**

- open matured **Product** in **Editor**,
- in **Editor** use context menu for the **Product** then call *New Child/Product Version*
- fill properties for the **Product Version** in **Property Sheet**.

As **Product** code base implements actual license protection, each **Product Version** needs to have a dedicated pair of cryptography **Key**s. Generate this **Key Pair** in **Property Sheet** for **Product Version**: use *Generate Keys* button. Fresh **Key**s will automatically end up in your *mycompany-data* project, under *keys* folder.

Public key is to be transferred to the product development department as it should be included into the corresponding **Product Version** distributive.

3.4 Feature Set

Final target for license protection is a **Product Feature** - some valuable functionality in the product.

Features are grouped in **Feature Set**s, and each of those is to be persisted as a separate file under workspace *mycompany-meta* project, to *features* folder. For instance, such a file could be named as *mycompany.reporting.features_xmi*

To create new **Feature Set** use *File/New/Feature Set...* main menu action. Then open it in **Editor** and then fill its properties in **Property Sheet**.

3.5 Feature

Feature represents substantial functionality of a **Product** that is to be protected by license as a single entity.

To model a **Feature**, open its owning **Feature Set** in **Editor**, use context menu for **Feature Set**, then call *New Child/Feature* action, open the data object in **Property Sheet** and fill its properties.

Feature is mainly represented by its *identifier*, which is the second piece of data to be shared with the product development. This identifier is to be implanted into the product's code base in the part of license protection implementation.

3.6 Feature Version

Feature at some point of its evolution achieves release state and thus can be shipped to end user as a part of **Product Version**. Such a state of a **Feature** is represented with **Feature Version**.

To version a **Feature** use context menu in **Editor** for owning **Feature** and call *New Child/Feature Version*. Fill the properties: *Version* value has critical meaning as, again, is to be in synch with the development perspective. It is recommended to practice *Semantic Versioning* [6] for any version value through all *Passage* [4] related products management.

3.7 Product Version Feature

Product Version Feature reflects a **Feature Version** in a **Product Version** and is the main subject for runtime license protection. Fill **Product Version** with features to make it clear what functionality of which versions is included in this release and under which conditions it is to be used.

To add a licensed functionality to your **Product Version**, use context menu for **Product Version** in **Editor**, call *New Child/Product Feature Version*, fill its properties in **Property Sheet**.

Great deal of responsibility lies on the *Feature Identifier*, which should match existing **Feature**, defined under accessible **Feature Set**. Use *Edit...* button to select one.

Second important property is *Restriction Level*. It literally defines the way license protection engine treats lack of permission grant for this feature at runtime of the product.

- **fatal** means that if there is no license coverage for this **Feature Version**, then the product itself is to be stopped. Use this *restriction level*

for an identity-feature of a **Product** or for critical functionality;

- **error** prohibits the functionality exploiting, but does not affect the product as a whole, so it can be used further;
- **warn** allows running the function, but makes sure end user is notified of lack of permissions by emitting some restriction or disturbance, like *Insufficient License Coverage* warning dialog;
- **info** allows to use the feature freely, can be applied by the product for logging or other purposes.

Agreements as to which *restriction level* is to be applied to which feature must be conveyed to development department in order to have proper responses be implemented in the product code base.

Chapter 4

Who can get a license

To keep track on who receives licenses for **Product**s, fill and maintain your **User** base.

4.1 User Origin

Users can grow numerous with time, so it's easier to manage them if they are organized in packs. A pack is called **User Origin**, each origin is persisted to a separate file under the *users* folder of *-data* project.

To create a **User Origin**, use *File/New/User Origin...* main menu action. Select proper folder to store the file, name it according to the origin identity, fill the properties in **Property Sheet**.

4.2 User

User, been properly configured, represents a licensee whatever that could mean under real circumstances. That could be a person, or a company's server workstation, or whichever else entity can be authenticated.

To create a **User**, open its **User Origin** in **Editor** and then use origin's context menu *New Child/User*. Then edit its properties in **Property Sheet**.

4.2.1 User Identifier

The most critical value has the *Identifier* field, as it is used on license issuing, which, among other consequences, affects any analysis built over licenses,

issued for this particular **User** or its **User Origin**. This *Identifier* must be unique across the whole *Operator Workspace*. Classic way to assign the *Identifier* is to use e-mail address for the **User** (in case it represents a real person), but there are no limitations to the form if this information.

4.2.2 User Contact

The next important group of **User**'s fields is *Contact* group. Fill actual contact information here that will help you to connect to the other party regarding any aspect of license issuing or related matter. Contact information is not obliged to be unique, but most often it refers to a particular **User** personal contact.

4.2.3 User Authentication

License issued for a **User** must contain enough information for product to physically identify the **User**. One of the most popular way is to bind the **User** with its hardware environment (workstation), on which the **User** is planning to run the product.

A **User** can have several workstations declared for licensing, so *Licensing Operator* can issue one license that will work on each of them.

Authentication section on **User**'s **Property Sheet** allows to create and maintain this part of **User** information.

To become a licensee (id est to get ability to ever gain a license) **User** must declare at least one **Host** (workstation). To model one for a **User** press *Create and link new Environment Host*. Fill the properties for the **Host**.

Host Assessment

A **Host**, to reside a license, must be detailed by physical aspects of its hardware environment. Those aspects and characteristics are to be assessed by the product on first startup, when the **User** gathers hardware information to request a license.

Currently results of such **Assessment** can be delivered to *Licensing Operator* by the **User** in two ways

- textually - it's a legacy format. Old versions of Passage expose hardware **Assessment** details to **User** in text form, which looks like this:

computer
manufacturer: MANUFACTURER-ID
model: MODEL-IDENTIFIER
serialnumber: SOME/SERIAL/NUMBER
system
vendor: COMPUTER VENDOR
processorid: PROCESSOR-IDENTIFIER
name: MODEL-NAME
family: MODEL-FAMILY
model: MODEL-NO
... Such a text is intended to be sent to *Licensing Operator* and used to authenticate the **User** on license issuing.

- in the form as **.assessment_xmi* file, which contains the same information in a structural form.

Assessment is literally a set of simple tokens (**Assessment Token**), each token contains

- name of an abstract property, describing hardware in general,
- actual value for this property on this **Host**.

If hardware **Assessment** details are received from **User** in the legacy format, use *Create and link new Environment Assessment* button, paste supplied text and press *OK*.

If **User** sent hardware **Assessment** in the form of a file, use *Link Environment Assessment* button and select the file on you local file system.

A **Host** can be assessed multiple times during its lifetime. Each **Assessment** is to be kept in conjunction with the report-date. If some **Assessment** is reported again, new report-date is automatically added to the **Assessment** already assigned to the **Host**.

Among all **Assessment**s the one is to be used for license issuing is the last reported. Such an **Assessment** also must have valid **Authentication Expression** defined.

Authentication Expression

All this authentication data gathering and structuring serves the only purpose: authenticate a **User** on product runtime. To do so, at the end of the

day *Licensing Operator* needs a valid **Authentication Expression**, associated with each active **Host** of the **User**.

Authentication Expression is a subset of **Assessment** tokens, significant enough to identify this particular **Host**. Until such an expression is defined for the **Host** it is not eligible for license issuing.

To select **Assessment Token**s that are to serve as **Authentication Expression**, edit an **Assessment** (double click on the corresponding row), use check boxes to choose valuable tokens.

4.3 User Group

User Group is an associative collection of **User**s, aimed for management purposes and further usages.

To create a **User Group**, use context menu in **Editor** for **User Origin**, call *New Child/User Group* action. Fill properties for newly created **User Group** in **Property Sheet**, pay attention to *Contact section*.

To assign **User** to a group, use *Users section/Link User* button. One **User** can participate in several groups simultaneously.

Chapter 5

How to issue a license

License is a reasonably loaded entity as it is obliged to embrace lots of precise information regarding licensed product functionality and the licensees. It is proved useful to keep most of this information, that keeps repeating itself from one issued license to another, separately, as a dedicated entity: **License Plan**.

5.1 License Plan

License Plan is designed as license template and keeps issuing similar licenses from boilerplate actions. From this perspective **License Plan** is defined by the set of **Feature Version**s.

It is also intended to be associated with product management information, such as, for instance, billing strategy. Besides all that **License Plan** also deposits information about each license ever issued under its guidance.

To create a **License Plan** use, for example, popup menu in **License Projects Navigator** and invoke *New/License Plan...* action.

Name the plan according to licensing aims at hand. It can be leasing commercial licenses for particular product version release, or for defined customer, or for education purposes, or short-term evaluation licensing.

Place corresponding **.licenses_xmi* file into *licenses* folder of *Operator Workspace -data* project, like *mycompany-data/licenses*. Open the file in **Editor**, fill its properties in **Property Sheet**.

Pick *Identifier* and *Name* according to the purpose of this **License Plan**. *Identifier* is to be unique in the global scope.

Fill your **License Plan** with the information regarding product features that are going to be covered by all the licenses, built over this plan. Use context menu for **License Plan** entity in **Editor**, then invoke *New Child/License Plan Feature* action.

License Plan Feature is a special entity intended to describe principles of applying a **Feature Version** to a license. Fill its properties in **Property Sheet**.

Feature Identifier is to be linked to a **Feature Version**, select one from the available set.

Feature Version here is used as a beacon point, against which licensing engine is to check an actually used functionality at runtime through the *Version Match Rule*.

- **perfect** rule tells the licensing engine inside the product that if there is a license grant for this functionality for version *a.b.c* then at runtime this functionality has to have precisely the same version *a.b.c* to be covered by this particular license grant;
- **equivalent** rule allows licensing engine to approve also runtime version *a.b.c1* - *patch* segment of version can differ. But if *major* or *minor* version segments do not match - this license grant cannot cover usage of this functionality;
- **compatible**, which is default, covers feature of runtime version of *a.b1.c1* version;
- **greaterOrEqual** lets licensing engine lease a permission to use any runtime feature of *a1.b1.c1* where *major* segment is the same or greater than the one of licensed version (a1 >= a).

Keep adding **Feature Version**s to the **License Plan** until all features are covered which are to be protected by a license, issued under this plan.

5.2 Licensing Schemes

5.2.1 Personal licensing

Personal licensing grants a certain single **User** access to features of a product that is installed on their workstation(s). All parties of this licensing process

physically reside on the same host. Licensing protection, developed inside the product, conducts **User** authentication and makes decision whether or not certain functionality is authorized for this **User** basing on information, supplied to the process by **Personal License**.

5.2.2 Floating licensing [5]

Floating licensing limits the number of **User**s who can simultaneously access specific features of the product. This scheme provides granular control over feature accessibility within an organization, allowing to set caps on the concurrent usage of particular features across multiple instances of the product.

Access is controlled by a dedicated License Server, to which instances of the product appeal through network in attempt to acquire a grant for certain product functionality.

Floating License is deployed under this *License Server*, supplying it with information as to

- who can access the feature - how to authenticate a **User**, who's instance of product is asking for a grant;
- which features can be accessed and under which conditions;
- how many concurrent grants can be leased over each particular feature.

5.3 Issue License

License Plan can be used to issue both

- **Personal License** for traditional licensing scheme;
- **Floating License** to be deployed under Floating License Server.

5.3.1 Personal License

To issue a **Personal License** use context menu in **License Projects Navigator** or **Editor** and invoke Issue Personal License... action.

On the *License Request* page of the corresponding wizard select

- **License Plan**, which defines features to be covered by this license and restrictions of licensing protection for them;

- **User** to represent the licensee for this license;
- **Product Version** to define the product this license is intended to run with;
- time period during which the license is valid.

User Authentication page enlists all the *Workstation*s of the selected **User** that are configured sufficiently to host a license. Select *Workstation*s on which this **User** supposed to run the product.

Next two pages *License Package* and *License Details* show all gathered information in structural and textual forms, close to actual license, for *Licensing Operator* to review carefully.

Press *Finish* button to lease the license. Subsequent dialog notifies *Licensing Operator* as to where exactly just issued **License Package** is placed.

5.3.2 Floating License

To issue a **Floating License** use context menu in **License Projects Navigator** or **Editor** and invoke Issue Floating License... action.

On the *License Request* page of the corresponding wizard define

- **License Plan**, which defines feature set to be covered by this license and restrictions of licensing protection for them;
- all the **User**s eligible for this license;
- **Product Version** to define the product for which this license is issued;
- time period during which the license is valid;
- *Grant Capacity* - amount of grants available for simultaneous acquisition from *License Server* by different **User**s

User Authentication page shows all the *Workstation*s of all the selected **User**s that are configured sufficiently to be authenticated against a license. Choose at least one *Workstation* for each **User**, on which this **User** supposed to run the product.

License Package page shows all gathered information in structural form for *Licensing Operator* to review.

On *License Server* page define whether or not it is necessary for each enlisted **User** to have personal access-file for the License Server. Can be omitted, if these **User**s previously were licensed by floating scheme to the same Server. To generate the access files, describe License Server network coordinates as they will work for users workstations.

Press *Finish* to lease the license. Subsequent dialog notifies *Licensing Operator* as to where exactly just issued **License Package**is stored.

Chapter 6

What is in a Personal License Package

Product user sees **License** as a small encrypted text file, which is to be available at the product runtime to grant them access to certain features of the product.

For *Licensing Operator* this file is a critical part of **License Package**, when the latter also contains additional information vital for license data further support and management.

Issued **Personal License Package** content is to be found under *Operator Workspace -data* project in directory *issued/product-id/product-version*, under dedicated folder named after the **Personal License Package** *identifier*.

The folder contains

- *identifier*.licen - encrypted license file, intended for end user
- *identifier*.lic - not encrypted license source file

Chapter 7

What is in a Floating License Package

7.1 Package

When operator issues a floating license pack, it resides in a folder named after the *pack identifier* like *d2b83215-b65d-4031-a8c8-a10421d56260*. Name of each file in the folder starts with this identifier as well. The folder contains all necessary data for all parties involved in floating licensing.

7.2 Files

There are in the folder

- **license files**, which contain list of all users eligible for the license and any feature can be used by it; they should be passed under control of **Floating License Server**

- optionally, **floating access files**, one per user, which keep connection and authentication credentials of the Server; these files are to be passed to each user mentioned in the floating license pack.

Table 7.1 shows a typical file structure of a floating license pack, delivered to a client.

Table 7.1: Sample floating license pack content

File	Role	Action
pack-id.flicen	Encrypted license	Pass to agent to be deployed under Floating License Server.
*product-id*_*product-version*.pub	Product's public key	Pass to agent to be deployed under Floating License Server, together with the license file.
*pack-id*_*user-id*.flaen	Encrypted floating access file	Pass to the mentioned user to be stored close to their installation of the product.

7.3 Content

7.3.1 Floating license file

Floating license File **.flicen* accumulates

- **Floating License Server authentication** information, as each floating license is to be released for a particular residence. A license is not intended to be functional in case it is deployed on a server, that cannot be authenticated with these credentials;
- enlistment of all **users allowed to exploit the product** - their identifiers, authentication conditions, other info;
- list of product features that can be used in the scope of this license pack, associated with the amount of available grants (simultaneous usages).

A product, running on a *user*'s installation, *acquires a grant* for a *feature*, which is valid for short period (*vivid* for 60 minutes by default). Since a grant is acquired, it's not accessible for another user until it is released.

After the *feature*'s work is over, the product *releases the grant* and thus it becomes available for acquisition again.

Feature grant capacity

Several product installations (users) can acquire different grants for the same feature for overlapping time slots. Feature grant's *capacity* field defines this

amount. If there are no more available grants, the acquisition fails, and the product blocks the feature utilization.

Feature grant vivid

A feature grant, been leased for a user, should not last long. Thus, a feature grant information contain instruction on how *vivid* is the grant. Measured in minutes, it defines how long will the grant be valid after leased (limited by the whole license pack validity period). After this relatively short vivid period the grant is not active, but still acquired until is explicitly released.

Feature grant validity period

A feature grant can define its own validity period. Even if it is declared to be wider than the owning license pack validity period, at runtime the former is still limited by the latter. This capability is used by licensing party to narrow validity period for a particular feature in the pack.

7.3.2 Floating access file

A product installation, been located on a user workstation, needs to be told how to access a Floating License Server.

Floating access file delivers this information. It contains:

- user identifier,
- the Server connection url components: ip address and port,
- the Server authentication condition.

Each user enlisted in the license file should possess such an access file and host it where the product is configured to look for it. Usually it is the product's folder under default *.passage* data directory: ~/.passage/*product-id*/*product-version*

7.4 Destination

Floating license files are to be put under control of the Floating License Server for which this floating license pack has been issued. See fls:upload command description.

Floating access file is to be stored in ~/.passage/*product-id*/*product-version* on a workstation where the product under licensing is installed.

Chapter 8

Best Practices

Through the years of applying *ArSysOp Cordon LCO* to different areas on variety of infrastructures several template solutions have been invented that come in handy. Most of them are centered around the idea that there is nothing more precious than your license management data.

8.1 Keep data safe

Both *Meta* and *Data* workspace files keep information, that can be crucial for the business.

We recommend to repose these two projects under a Version Control system [7] and constantly synchronize theirs contents with the operative state.

This could also help several *Licensing Operator*s to work on license issuing simultaneously in the same *global licensing scope*.

We also recommend to use different repositories with separately managed access rights for *Meta* and *Data* projects, and in some cases even for *issued licenses* and *keys* storage folders.

8.2 Updating to newer LCO version

As *ArSysOp Cordon LCO* tool evolves, it grows new functionality which in most cases requires new data. That, in turn, means changes to the format your existing data is stored. So it is highly recommended to

- try newer *ArSysOp Cordon LCO* version before your license for the one you have ends;
- install newer *ArSysOp Cordon LCO* in separate location;
- backup your data before running it under newer version of *ArSysOp Cordon LCO*;
- install fresh *Operator Workspace* in firstly run newer *ArSysOp Cordon LCO*: import all the projects from scratch, never reuse *Operator Workspace* left from an older versions, never use newer *Operator Workspace* in older *ArSysOp Cordon LCO* versions.

8.3 Operator Workspace

Strictly follow folder naming convention, described in this guide. Otherwise several subsystems can grow dysfunctional.

8.4 Information management

8.4.1 General

Give it a though upfront and invent a consistent naming (identifying) strategy, that would serve your particular needs, for all the artifacts in the field:

- file storage units for **Product Line**s, **Feature Set**s, **User Origin**s, etc;
- each object in each domain tree.

The strategy must guarantee uniqueness in the global scope.

This will save resources in the future and render maintenance easier.

Nesting namespaces earned themselves a good reputation as a tool for this task.

8.4.2 Product management

When it comes to a product release, simultaneous versioning proved itself useful in many cases. According to this practice, all features, including main product-launch feature, are versioned synchronously with the product, then new **Product Version** gets filled with these newly created **Feature Version**s. New **License Plan** is added to issue licenses for this particular release.

For example, it is planned to launch new release for a product *A* named *2.0.0* in half a year. Then, following the practice, *Licensing Operator* should

- find **Product** *A* under *-meta* project, *products* folder;
- create new **Product Version** for it with *2.0.0* version;
- generate new pair of cryptography keys for this **Product Version**;
- follow the keys to theirs storage: *-data* project, *keys* folder, *<product-name>/<version>* sub-folder;
- send public key (file with .pub extension) to development; this file must be included into the product release distributive for licensing engine to read licenses, issued for this version of the product;
- fill just created **Product Version** with all the **Product Version Feature** for each feature that is planned for this release; if a feature evolved significantly, new version for it must be created as well to be used in this release;
- create new **License Plan** that will serve as ground base for issuing licenses this release of the product: under *-data* project, in *licenses* folder; identify the plan accordingly - include both product name and version in the file name and in **License Plan**'s *Identifier*, describe the **License Plan**;
- add **License Plan Feature** to just created **License Plan** for each **Product Version Feature** that compose the **Product Version** under development;
- synchronize formal values of product version the and versions of all the features in it with development department.

Now everything is settled and ready to handle license requests for this product release.

8.4.3 User management

Grouping

Design clustering criteria of **User**s upfront to ease future navigation.

Workstations

It is highly recommended to keep all the records of any **User** workstation assessment and closely monitor workstations, declared for licensing. That could help with the disputes. In typical case a **User** asks to re-issue a license under the claim that only the part of hardware that was used for authentication is replaced when actually new license is going to run on entirely new workstation; same data can prove the **User**'s right and settle the ground for free license re-issuing.

8.4.4 License management

License Plan is a potent notion and can be associated with several kinds of management information. For instance, it appears useful to build a scheme of **License Plan**s with the billing strategy in mind.

References

[1] *ArSysOp*. URL: https://arsysop.ru.

[2] *ArSysOp Cordon LCO*. URL: https://arsysop.ru/cordon.

[3] *ArSysOp Cordon Support Repository*. URL: https://github.com/arsysop/cordon-support.

[4] *Eclipse Passage Project*. URL: https://projects.eclipse.org/projects/technology.passage.

[5] *Floating Licensing*. URL: https://en.wikipedia.org/wiki/Floating_licensing.

[6] *Semantic Versioning*. URL: https://semver.org/.

[7] *Version Control System*. URL: https://en.wikipedia.org/wiki/Version_control.