

Murray Peters

August 2018

FNMEA 3rd Party License Server Mgt

& CRL support

Contents

[Overview 3](#_Toc524025097)

[Disclaimer/Support 3](#_Toc524025098)

[Components 4](#_Toc524025099)

[Plugin architecture 4](#_Toc524025100)

[Installing the plugins 4](#_Toc524025101)

[Admin server plugin 4](#_Toc524025102)

[FNMEA agent plugin 5](#_Toc524025103)

[Setting up a 3rd party License server 6](#_Toc524025104)

[Enabling 3rd Party License Server management 6](#_Toc524025105)

[Defining Common Report Log (CRL) definitions 8](#_Toc524025106)

[Defining Third-Party Licensing Templates 12](#_Toc524025107)

[Adding a Third-Party License server 13](#_Toc524025108)

[License server commands and what they (must) do 14](#_Toc524025109)

[Testing the end to end setup 16](#_Toc524025110)

[CRL generator 19](#_Toc524025111)

[Overview 19](#_Toc524025112)

[CRL generator design principles/how it works 19](#_Toc524025113)

[Anatomy of a Properties file 20](#_Toc524025114)

[Global CRL settings 20](#_Toc524025115)

[Event based entries 21](#_Toc524025116)

[Working examples 25](#_Toc524025117)

[How videos/demo’s 25](#_Toc524025118)

# Overview

FNMEA natively supports all FlexLM based license servers using lmgrd whereby it can monitor their health, perform administration functions such as start/stop license servers and rotate/collect report logs that are accumulating on the license server where an FNMEA agent is installed.

For 3rd party license servers (non-FlexLM based) FNMEA offers the Common Report Log (CRL) interface to allow usage data/report logs to be imported into FNMEA so it can be reported on along with all other standard FlexLM usage data. This requires the 3rd party data to be transformed to format that FNMEA can understand/import and then imported into FNMEA through the web interface (one by one) or via a command line bulk import on the admin server.

When the version 5.x agents were developed, functionality was added to the agent and the administration server to allow (java based) plugins to be developed to cater for 3rd party extensions. The capability being made available here and explained in this document is a generic plugin (both for the agent and the admin server) that allows any 3rd party license server to be managed by FNMEA. This includes the capability of Starting and Stopping the license server, generating health/status and automating report log rotation and upload to FNMEA. The plugin also includes a rule based CRL generation engine which allows new vendor log formats to be supported by defining the matching rules in a config file read at runtime.

The plugins have been tested with the following vendors/products:

|  |  |
| --- | --- |
| * Reprise * Sentinel * Fekete * LM-X (version 301) * LM-X (version 363) * SpaceGass | * Bentley * MATLAB (based on FlexLM debug logs) * Tasking * Reprise – Log Format 0 * Reprise – Log Format 2 |

# Disclaimer/Support

The capabilities provided by these plugins/extensions are provided on an as-is basis without support or warranty of any kind.

# Components

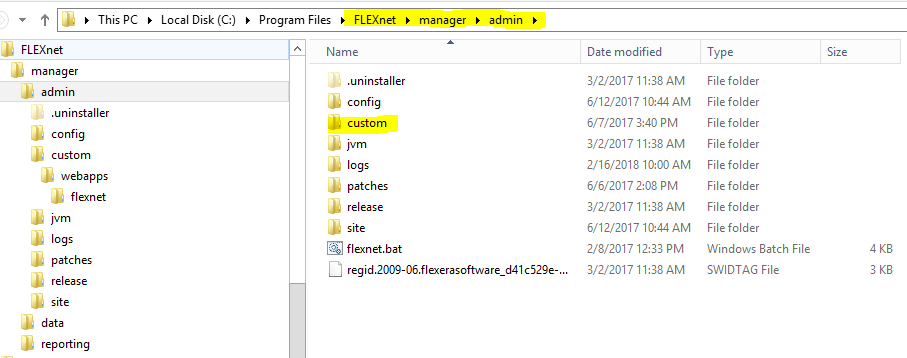
## Plugin architecture

|  |  |
| --- | --- |
|  | This solution is comprised of two separate plugin – one registered on the FNMEA admin server and one registered with the FNMEA agent.  The admin server plugin initiates requests to the agent to Start/Stop/Status check and Rotate logs from the server and processes the messages sent back from the agent.  The agent plugin has two purposes: one to respond to admin requests and action the Start/Stop/Status and Rotate commands. The other feature of the plugin is a general purpose CRL generator engine (rule based). |

## Installing the plugins

### Admin server plugin

1. Stop the FNMEA admin service
2. On the FNMEA admin server, copy the “custom” folder containing the FNMEA admin server plugin to the FLEXnet/manager/admin. It should look like this:

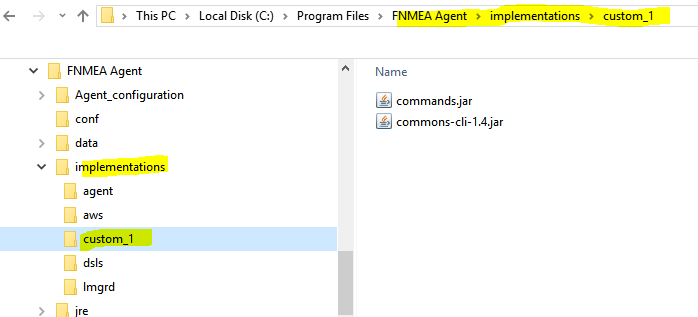


1. Execute the following command to build the site:

flexnet site make

1. Restart the FNMEA admin service

### FNMEA agent plugin

1. Stop the FNMEA agent service
2. On the FNMEA agent server, copy the “custom\_1” folder containing the FNMEA admin server plugin to the “FNMEA Agent/implementations”. It should look like this:  
   
3. Restart the FNMEA agent service

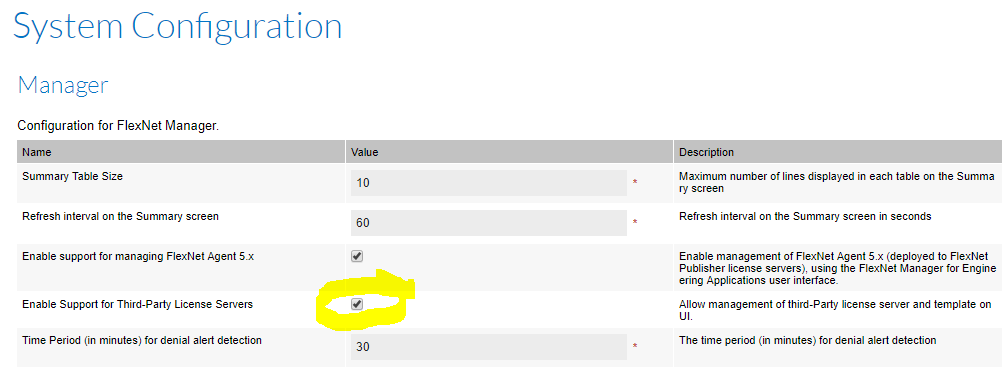
# Setting up a 3rd party License server

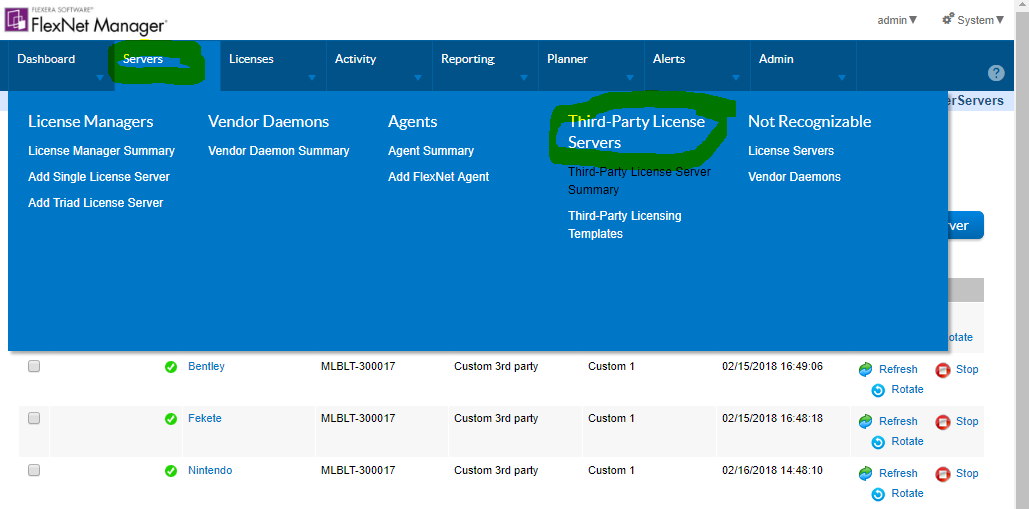
It is assumed here that FNMEA is set up with FlexNet agents version 5.x or later already registered (windows or \*nix).

The process for adding 3rd party license servers involves a number of preparatory steps:

1. Enabling 3rd party license server management in FNMEA
2. Defining Common Report Log (CRL) definitions for use by the plugins
3. Defining a Third Party Licensing template use as a basis for each 3rd party server
4. Finally, adding the 3rd party licensing server based on the chosen template.

## Enabling 3rd Party License Server management

1. A default install of FNMEA does not automatically enable 3rd party license server management. To enable it, select the checkbox shown below (under System Configuration/Manager):  
   

* You can then access the web pages for managing 3rd party license servers from the following menu:  
  

## Defining Common Report Log (CRL) definitions

Note: If using the CRL generation engine in this plugin to transform 3rd party usage data to files that can be imported into FNMEA, the engine generates output based on a predetermined CRL structure (i.e. the order of the fields being Feature, Version, DateTime etc). Hence it is important to follow the steps in this guide for setting up the CRL definitions and template.

Alternatively, if you are using external scripts to generate CRL files you set up your CRL definitions based on your own preference.

We will create two CRL definitions: one to support imports with data that is being matched by TransactionID’s and the other to support imports with data that is being matched by Duration’s.

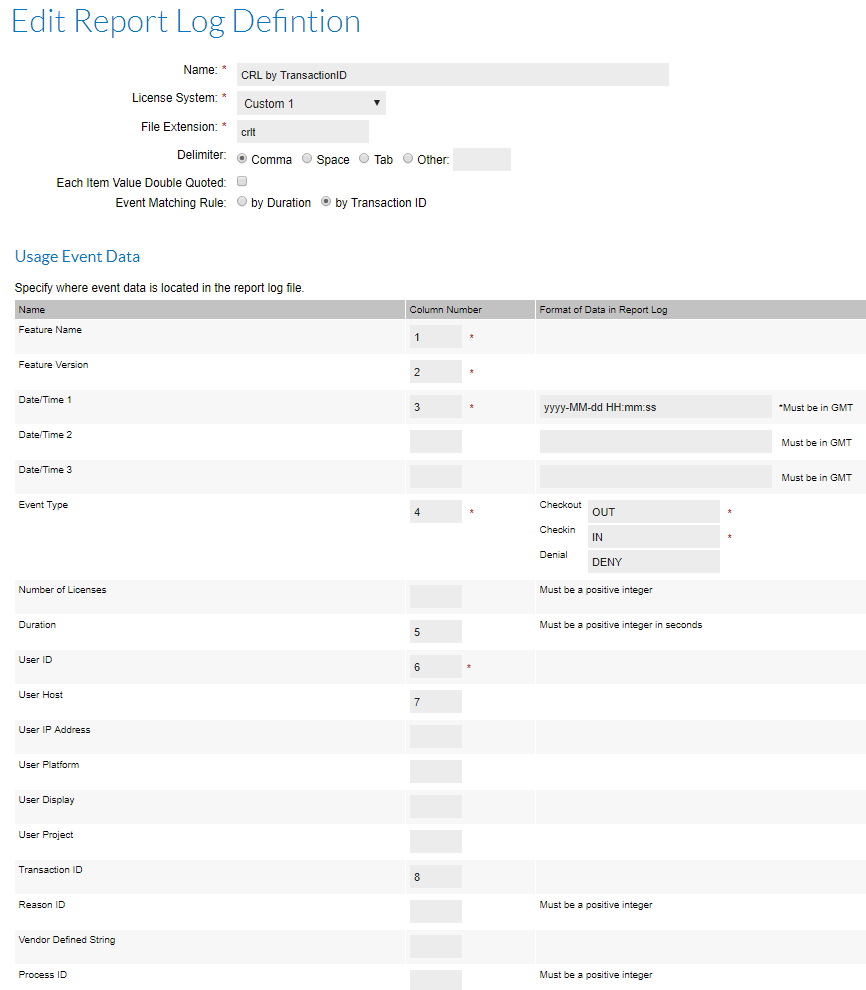
1. **CRL Definition by TransactionID**
   1. Create a new report log definition by selecting the menu item: Reporting/Create Report Log Definition. Complete by entering values in to the following fields:
   2. Name: CRL by TransactionID
   3. License System: Custom\_1
   4. File extension: crlt  
      Note: this extension is somewhat arbitrary, but it must match the extensions generated by the CRL engine. The engine generates CRL based on the file extension specified in the properties definition for the respective vendor. See section XXX for details about setting the value for the (crl.fileextension) properties definition.
   5. Delimiter: Comma
   6. Event Matching Rule: by Transaction ID
   7. Usage Event Data:
      1. Feature Name: 1
      2. Feature Version 2
      3. Date/Time 1 3 yyyy-MM-dd HH:mm:ss  
         Note: this datetime format is somewhat arbitrary, but it must match the format generated by the CRL engine. The engine generates dates based on the date time format specified in the properties definition for the respective vendor. See section XXX for details about setting the value for the (crl.datetimeformat) properties definition.
      4. Event Type 4 Checkout: OUT

Checkin: IN

Denial DENY

* + 1. Duration: 5
    2. User ID: 6
    3. User Host: 7
    4. Transaction ID: 8

1. Hit Save - here is what the completed definition should look like:



1. **CRL Definition by Duration**

This definition is essentially identical to the previous one except for specifying that this is matching by Duration and hence the file extension is also different so FNMEA knows which CRL definition to use. The highlighted fields below have values different from the previous definition.

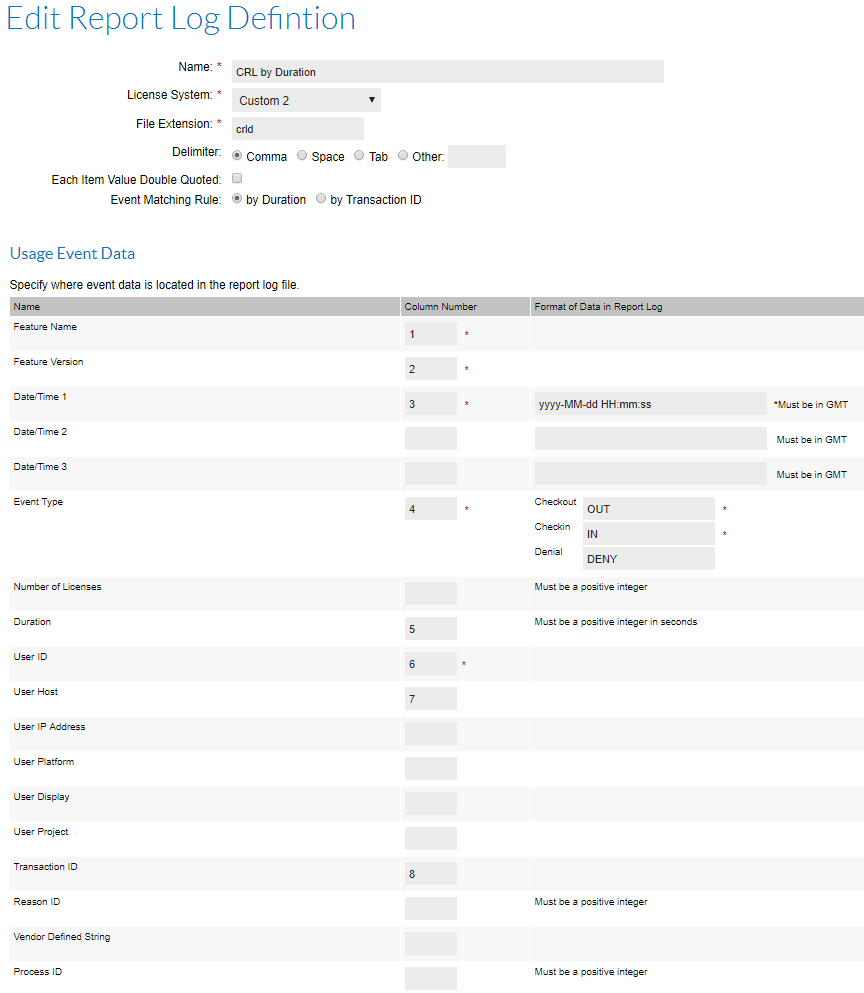
* 1. Create a new report log definition by selecting the menu item: Reporting/Create Report Log Definition. Complete by entering values in to the following fields:
  2. Name: CRL by Duration
  3. License System: Custom\_2  
     Note: FNMEA doesn’t allow the same License System to be chosen for two different CRL definitions (unfortunately) – hence any other License System could be chosen here (the value for the License System doesn’t impact the functionality of the plugins).
  4. File extension: crld  
     Note: this extension is somewhat arbitrary, but it must match the extensions generated by the CRL engine. The engine generates CRL based on the file extension specified in the properties definition for the respective vendor. See section XXX for details about setting the value for the (crl.fileextension) properties definition.
  5. Delimiter: Comma
  6. Event Matching Rule: by Duration
  7. Usage Event Data:
     1. Feature Name: 1
     2. Feature Version 2
     3. Date/Time 1 3 yyyy-MM-dd HH:mm:ss  
        Note: this datetime format is somewhat arbitrary, but it must match the format generated by the CRL engine. The engine generates dates based on the date time format specified in the properties definition for the respective vendor. See section XXX for details about setting the value for the (crl.datetimeformat) properties definition.
     4. Event Type 4 Checkout: OUT

Checkin: IN

Denial DENY

* + 1. Duration: 5
    2. User ID: 6
    3. User Host: 7
    4. Transaction ID: 8

1. Hit Save - here is what the completed definition should look like:

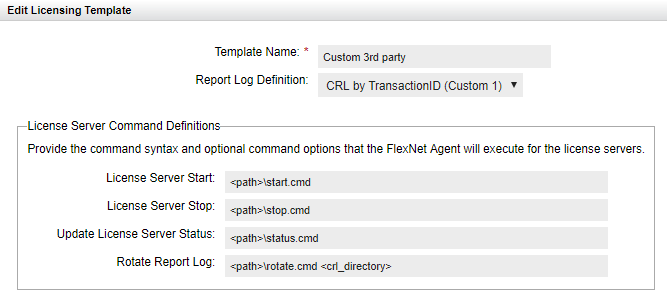


## Defining Third-Party Licensing Templates

Third-Party licensing templates are used to default the settings for managing a single server, this includes the set of commands used to Start/Stop/Status and Rotate report logs as well as pairing it to the CRL definition that should be used.

We will only need one single template definition which can then be used generically for any 3rd party technology.

1. Create a new Third-Party licensing template by selecting the menu item: Servers/Third Party Licensing Templates and then selecting the button: “New Third-Party Licensing Template”. Complete by entering values in to the following fields:
2. Template Name: Custom 3rd Party
3. Report Log Definition: CRL by TransactionID (Custom 1)  
   Note: You should see both CRL definitions in the drop down list, but just choose the first one (Custom 1). It doesn’t functionality matter as it doesn’t impact the plugins, so is really only seen as a column in the FNMEA web page views.
4. License Server Start: <path>\start.cmd  
   License Server Stop: <path>\stop.cmd  
   License Server Status: <path>\status.cmd  
   License Server Rotate: <path>\rotate.cmd <crl\_directory>  
     
   Note: these are simply template commands – when adding each 3rd party licensing server we will end up replacing the template commands with the appropriate commands for the target (windows or \*nix) platform. So here, these template commands simply server as a reminder of typical values to enter.
5. Hit Save - here is what the completed definition should look like:

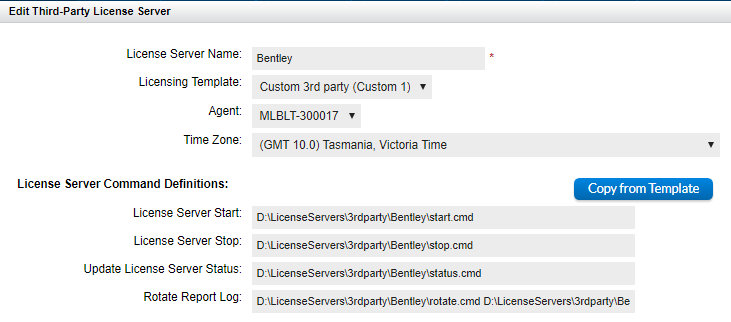


## Adding a Third-Party License server

Third-Party licensing templates are used to default the settings for managing a single server, this includes the set of commands used to Start/Stop/Status and Rotate report logs as well as pairing it to the CRL definition that should be used.

Individual 3rd party license servers can now be added based on the defined template and adjusting any commands related to the target.

1. Create a new Third-Party license server by selecting the menu item: Servers/Third Party License Server Summary and then selecting the button: “New Third-Party License Server”.   
   Complete by entering values in to the following fields (we will use Bentley as an example):
   1. License Server Name: Bentley
   2. Licensing Template: Custom 3rd party (Custom 1)
   3. Agent: MLBLT-300017  
      Note: you should choose whichever machine your agent is installed on.
   4. License Server Start: D:\LicenseServers\3rdparty\Bentley\start.cmd  
      License Server Stop: D:\LicenseServers\3rdparty\Bentley\stop.cmd  
      Update License Server Status: D:\LicenseServers\3rdparty\Bentley\status.cmd  
      Rotate Report Log: D:\LicenseServers\3rdparty\Bentley\rotate.cmd   
       D:\LicenseServers\3rdparty\Bentley\Outgoing   
        
      Note: the paths and commands above are arbitrary, but examples of working files are available from the following location that use these values and command files.  
        
      The one exception is the Rotate Report Log command which MUST specify a command and a second value denoting the directory the agent can collect generated CRL files from. The agent will then enumerate all files in that directory and upload them to the FNMEA server (hence it is assumed that the file extensions of generated files will match the CRL definitions defined earlier).
2. Hit Save - here is what the completed definition should look like:



## License server commands and what they (must) do

The four commands (Start/Stop/Status and Rotate) that are sent to the agent and executed must adhere to the following:

* All:
  + The command supplied must be executable on the target system as specified, i.e. must be a valid batch file, script, executable accessible by the account the FNMEA agent is running as (typically Local System on windows, root or named account on \*nix).
* Start/Stop
  + No other constraints – the commands are executed by the agent, but no return codes or output is expected.
* Status
  + This command is expected to at least return (to stdout) the status of the license server (UP or DOWN), but could also optionally provide real-time details of licenses in use. The output of this command (i.e. to stdout) should conform to the following syntax.  
    Any lines in stdout not starting with any of the following keywords will be ignored:  
      
    LICENSESERVERHOST:, VENDORDAEMON:, LICENSESERVERSTATUS:, DATETIMEFORMAT:, USAGEEVENT:
  + Notes:
    - The LICENSESERVERHOST, VENDORDAEMON values are just text.
    - The LICENSESERVERSTATUS value should be either UP or DOWN.
    - The DATETIMEFORMAT is your way of describing how to interpret the dates in the subsequent usage events. It must be a value datetimeformat string (see https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html)
    - The USAGEEVENT entries/values represent a license in use – one per line. This is a comma separated line comprised of exactly 9 tokens in the following order:
      * DateTime  
        Required: Date/time the license was checked out.
      * FeatureName  
        Required: Name of feature in use
      * FeatureVersion  
        Required: Version of feature in use
      * User  
        Required: Username who has the license checked out
      * Host  
        Required: Hostname of machine license was checked out from
      * LicensesInUse  
        Optional: Number of licenses the checkout is using  
        Default is 1.
      * LicensesAvailable  
        Optional: Total number of licenses in the pool from this server (independent of how many are being used right now).  
        Default is 999999
      * DisplayName  
        Optional: For X-Windows systems, the name of the display the user is interacting with
      * Handle  
        Optional: the process id/handle of the program that checked out the license.
  + Some examples of stdout reflecting the status of license servers are:
    - License server is down:

LICENSESERVERNAME: Bentley  
LICENSESERVERHOST: MELLT-300017  
VENDORDAEMON: Bentley  
LICENSESERVERSTATUS: DOWN

* + - License server is up but no usage:

LICENSESERVERNAME: Bentley  
LICENSESERVERHOST: MELLT-300017  
VENDORDAEMON: Bentley  
LICENSESERVERSTATUS: UP

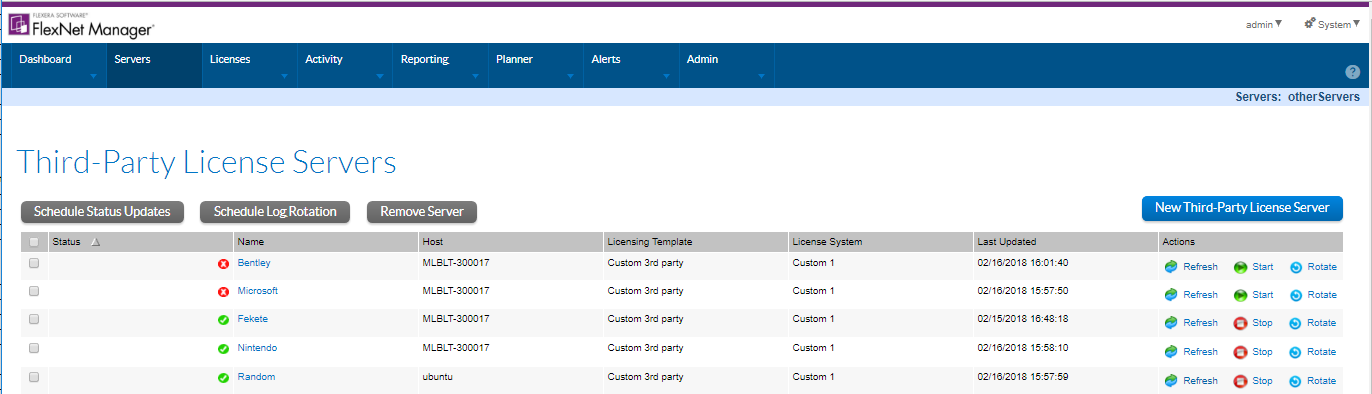
* + - License server is up and licenses in use:

LICENSESERVERNAME: Bentley  
LICENSESERVERHOST: MELLT-300017  
VENDORDAEMON: Bentley  
LICENSESERVERSTATUS: UP  
DATETIMEFORMAT: yyyyMMddHHmmss  
USAGEEVENT: 20170606130000,FeatureA,2013,newton,applemac,1,50,,  
USAGEEVENT: 20170609042358,FeatureA,2013,darwin,darwinspc,1,60,,  
USAGEEVENT: 20170609042358,FeatureB,2013,plato,platospc,10,60,,  
USAGEEVENT: 20170609042358,FeatureD,2013,aristotle,aristotlepc,5,60,,  
USAGEEVENT: 20170609120000,FeatureA,2016,galileo,galileospc,1,50,,

* Rotate (example: <path>\rotate.cmd <crloutputdirectory>)
  + This command has a mandatory parameter which qualifies the directory that any CRL files will be placed in by the command and therefore the agent plugin can find them to carry on with the upload process. The command can execute whatever it needs to to rotate logs, transform data formats to CRL format etc, but the generated files must be placed in the directory specified in the parameter so that the agent plugin can collect/upload the generated CRLs. The following describes possible scenarios:
    - The command is executed but no CRL files are generated. Subsequently, the agent/plugin has no further work to do.
    - The command is executed and one or more CRL files are generated and put in the <crloutputdirectory> by the rotate command. The agent/plugin continues after the command finished, iterates through each file in the <crloutputdirectory> and uploads each on to the FNMEA server. Standard FNMEA agent handling of uploads ensures that processed files are archived in the standard agent way.

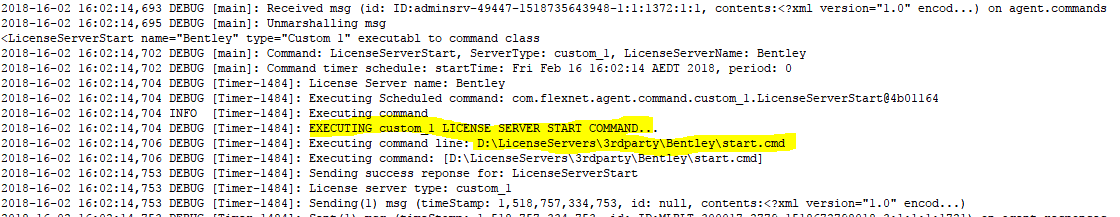
## Testing the end to end setup

After completing the preceding steps you should now be able to see the list of all registered 3rd party license servers from the menu: Servers/Third Party License Server Summary.

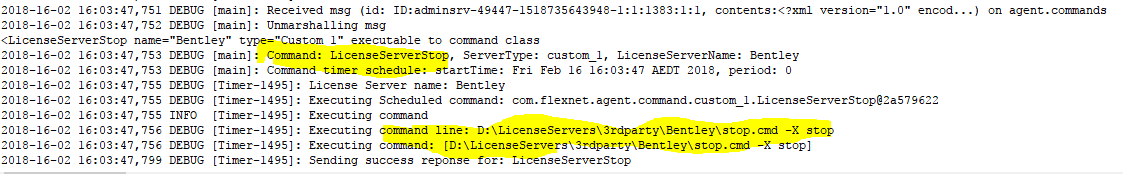


You can now initiate the Start/Stop/Status and Rotate commands manually by selecting the button, or you can schedule updates by entering the appropriate settings in the report log definition. To test that the agent is receiving the instructions from the admin server and is executing the commands specified you can monitor the agents debug.log where you should see entries like:

**Start**:

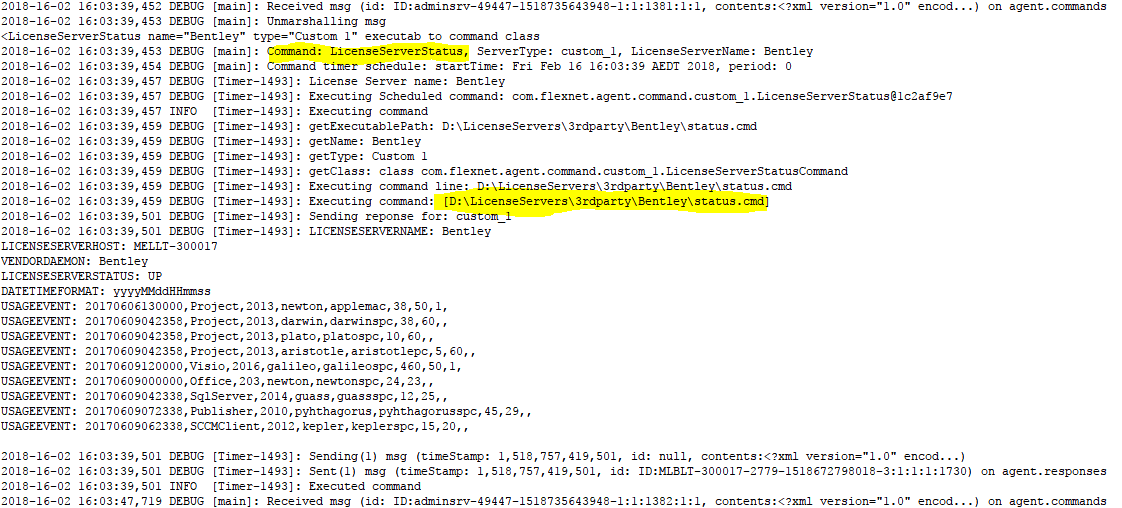


**Stop**:



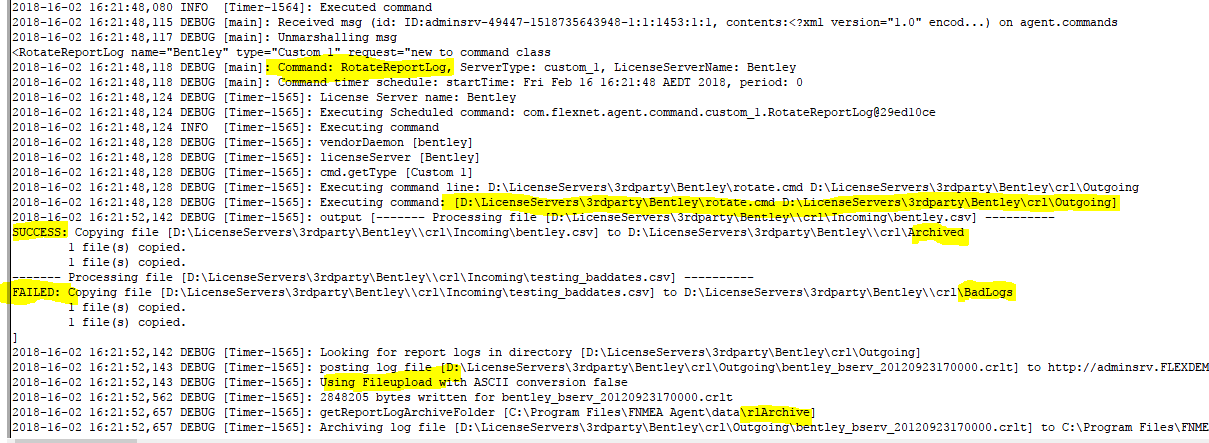
**Status**:

In the example below the status.cmd file is simulating actual usage and returning data representing the server status (UP/DOWN) as well as licenses in use.



**Rotate**:

In the example below the rotate.cmd file is invoking the agent plugin capability to generate CRL files from raw usage. The status below shows 2 files were waiting to be processed, one “Bentley.csv” which is processed successfully and for testing purposes the other file had deliberately corrupted dates to demonstrate how invalid files get handled. In both cases the files processed are either archived (on SUCCESS) or moved to a BadLogs folder (FAILED).



# CRL generator

## Overview

The CRL generator functionality included in this plugin is available for use when generating CRL files, but is not required to be able to use the 3rd Party License Server features. Ultimately, the 3rd party commands (start/stop/status/rotate) can be fully implemented using your own scripts – the one caveat being that the value for the rotate command must have a single parameter specifying the directory/folder containing generated CRL files to be uploaded to FNMEA.

Similarly, the CRL generator can be used in isolation (outside of the 3rd party license server capabilities) purely to process raw usage data to generate CRL files.

As the plugin is java based it works successfully on both windows and \*nix platforms. Some features of the CRL generator are:

* Timezone support for specifying the timezone to be applied
* Custom date formats can be specified for use in generated CRL files
* Feature mapping to support cases where feature codes are used in usage and you need the CRL data to supply readable feature names.
* Supports Single event rows (where corresponding checkins and checkouts are in separate rows of data) as well as Complete event rows (where the checkout and checkin is in the same row of data).

## CRL generator design principles/how it works

The goal of the CRL generator was to provide a single framework/engine that removes the need to develop scripts or programs from scratch to cater for separate 3rd party usage logs. Ultimately, all (useful) logs will contain the same kind of information such as Server start/stop/reread, Usage events (checkins/checkouts/denials) etc, but will always be in different formats.

If you physically look at each row of data you intuitively know what it represents, so if you can describe a pattern in the row to filter out rows of interest and then also specify how to parse/tokenise the row of data you can then automate the processing of that file. So, the generator takes a config file as input which has rules specifying how to interpret the data as well as being given a file to process. The output is one or more generated CRL files whereby each generated file is bookended by each server Start/Stop found in the original file.

The way the engine works is:

1. Accept parameters specifying the input usage data (file) and the associated config file (specifying the rules for that log format).
2. Each row is processed in turn whereby:
   1. Each rule from the config is evaluated to see if it matches the row
   2. If it matches, the rule is applied to extract elements that make up a usage event. The usage event could be a server start/stop/reread,checkout,checkin,denial etc.
   3. The generated Usage event is stored in a list
   4. When all data is read, the list is sorted chronologically (catering for the possibility of out-of-sequence data being fed in). If two events happen to have the exact same date/time, then they are ordered by event type in the order start,deny, checkout, checkin, end.
   5. The list is then traversed and CRL files generated:
      1. A new file is generated when a server start is encountered (in the case where no server start is found and a real usage event is encountered, the engine will fabricate a server start based on the date/time of the first usage event).
      2. Usage data is written when standard events are encountered (in case there was no preceding Server start encountered the engine will fabricate a server start).
      3. The CRL file is completed when a Server stop is encountered (in the case where no server stop is found, the engine will fabricate a server stop at the end of the data).

## Anatomy of a Properties file

Each properties file is used to specify all rules for processing a license server log format. Whilst the contents of the Properties file (i.e. fields and their values) can be written in any order it is most logical to retain a consistent structure as detailed in the following sections.

Note: comments are entered by using a #

### Global CRL settings

These settings specify overall CRL info such as the vendor name/server name and CRL generation formats. An example is shown below:

crl.vendor=spacegass  
crl.servername=TITANSERV  
crl.datetimeformat=yyyy-MM-dd HH:mm:ss  
crl.fileextension=crlt  
crl.features=

Items of note here are the crl.fileextension field which must match the file extension used in the CRL Definition (entered into the FNMEA web UI when setting up the third party license servers).

The crl.datetimeformat is used to control the format of all dates written out to the generated CRL file.

The purpose of the crl.features option is to optionally specify a CSV file that contains a list of features served by the license server and the quantities of available licenses for each feature. This information is the basis of the Feature List section in the generated CRL file: It does the following:

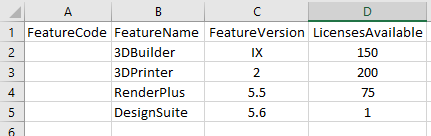
* allows you to specify a list of features including their name, version and #licenses.
* allows you to map a feature code that appears in the input data to a friendly feature name that should be used to be written out in the generated CRL file.

The format of the CSV file must have the following four columns:

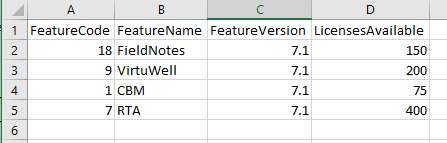
* FeatureCode
* FeatureName
* FeatureVersion
* LicensesAvailable

Some examples of files are:

1. A typical/simple case with no feature mapping:



1. Including feature mapping where the raw data only uses the FeatureCode, but we want the FeatureName used in the generated CRL file (i.e. 18 in the raw data would be written in the CRL as FieldNotes) :



### Event based entries

The remainder of the Properties file contains sections representing the rule sets for individual event types such as Checkout, Checkin, Denial, Server Start, Server stop etc.

Each rule set has a rule specifying how it is Matched against of row of incoming data, as well as a secondary rule for Discarding it if needed. Here are some examples:

#Checkout  
#1,11:42:48 06/09/2011,mary.poppins,HOSTNAME123,Licence Create,0,864958248,SPACE GASS 10,,1,SUCCESS

checkout.match=^.\*,Licence Create,.\*$  
checkout.discard=^.\*,Licence Create,.\*,FAILED$  
checkout.eventtype=OUT

Here, the use of the word “checkout” as the initial qualifier is arbitrary. It can be any text and should be named to help provide context for the rule it is defining. A properties file can contain any number of these rule sets which allows it to be possible to cater for data with multiple formats for a single event as you could for example define two different rule sets for a checkout.

The green text is pasted from the actual usage file to help document the properties file. The checkout.match denotes the rule for matching a Checkout and the value is a regular expression which summarizes as any line that contains the exact string “,Licence Create,”. The checkout.discard value is optional but in this case it is being used to Discard FAILED checkouts whereby the very end of the row has to end with the word “FAILED”.

The rest of the rules related to processing the related checkout are below:

checkout.feature.type=token  
checkout.feature.tokensdelim=-  
checkout.feature.tokens=,-7  
checkout.feature.stripchars=

checkout.version.type=fixedvalue   
checkout.version.fixedvalue=1.0

checkout.user.type=token  
checkout.user.tokensdelim=-  
checkout.user.tokens=,-2  
checkout.user.stripchars=

checkout.host.type=token  
checkout.host.tokensdelim=-  
checkout.host.tokens=,-3  
checkout.host.stripchars=:

checkout.transactionid.type=linenumber  
  
checkout.datetime1.type=token  
checkout.datetime1.tokensdelim=-  
checkout.datetime1.tokens=,-1  
checkout.datetime1.stripchars=  
checkout.datetime1.format=HH\:mm:\ss dd/MM/yyyy

Here, you can see that related to a Checkout event, there is data representing the Feature, Version, User, Host, TransactionID, Datetime etc. Hence, there are descriptors for each of these and subsequent attributes for each describe how to tokenise/parse the row of data to extract the value needed. Often, the parsing is simply a matter of tokenising the string by specifying the delimiter and then the index or position of the related data amongst those tokens.

Let’s take a look at one specific example in here where the Feature name is extracted from the row:

out.feature.type=token  
out.feature.tokensdelim=-  
out.feature.tokens=,-7  
out.feature.stripchars=

1. The “out.feature.type” has the value “token” which means that the row is parsed by tokenizing it. Other values this could take would be “fixedwidth” which would be used when there are no delimiters but the data can be extracted by specifying the start/end position in the string.
2. The “out.feature.tokensdelim” is used by the engine to know how to interpret the next field “out.feature.tokens”. In this example it’s value of “-“ means that the value of “out.feature.tokens” can be split by using “-“ as a delimiter. See the next description for why that is important.
3. The “out.feature.tokens” is where most of the dissecting of the data will happen. The value of this field represents a map for getting to the actual feature name and works by specifying one or more “pairs” of data representing a token/position. The value here of “,-7” means the following:
   1. First, since the “-“ was nominated to split this string we end up with a comma “,” and a seven “7”. This instructs the engine to tokenise the row of data by commas, then extract the string in position 7. From the Green commented line representing the same data you will see that this extracts the value of “SPACEGASS 10”.
   2. As a different example, a value of “,-7@-3\*-5” would be split into pairs of
      1. , 7
      2. @ 3
      3. \* 5

This would tell the engine to first tokenise by “,” and extract the string in position 7, then tokenise the extracted string by “@” and extract from it the string in position 3, then tokenise that extracted string by “\*” and extract the string in position 5.   
Whilst it is unlikely that usage data will be encountered that need this many levels of tokenizing/extracting it is comforting to know that you can supply as many levels of token/position pairs as needed.

1. The “out.feature.stripchars” is an optional way to finally remove characters from the string. This is useful for example if the Feature name is contained in quotes – here you could specify the string character as the value here and the engine will remove that character. Other examples might be “[]” or “()” to remove brackets etc.

# Working examples

The plugins have been tested with the following vendors/products:

|  |  |
| --- | --- |
| * Reprise * Sentinel * Fekete * LM-X (version 301) * LM-X (version 363) * SpaceGass | * Bentley * MATLAB (based on FlexLM debug logs) * Tasking * Reprise – Log Format 0 * Reprise – Log Format 2 |

These examples are available along with the plugins and documentation from Flexera Labs.

# How videos/demo’s

Videos describing the capabilities and use of the plugin are also available from the Flexera Learning Center. The learning center is accessible to Flexera employees, customers and partners via the Flexera community portals:

* Partner Portal : <https://flexeracommunity.force.com/partner/>
* Customer Portal : <https://flexeracommunity.force.com/customer>

After accessing the learning center, search for the following topics:

1. FNMEA – 3rd party license license server management
2. FNMEA – Rules based CRL generator

# Attachments



About Flexera

Flexera is reimagining the way software is bought, sold, managed and secured. We view the software industry as a supply chain, and make the business of buying and selling software more transparent, secure, and effective.  Our Monetization and Security solutions help software sellers transform their business models, grow recurring revenues and minimize open source risk. Our Vulnerability and Software Asset Management (SAM) solutions strip waste and unpredictability out of buying applications, helping companies purchase only the software and cloud services they need, manage what they have, and reduce license compliance and security risk. In business for 30+ years, our 1000+ employees are passionate about helping our 80,000+ customers generate millions in ROI every year.  Visit us at: **[www.flexera.com](http://www.flexera.com/)**



**Flexera**

300 Park Blvd., Suite 500

Itasca, IL 60143

USA

Itasca (Global Headquarters):

+1 800-374-4353

United Kingdom (Europe, Middle East Headquarters)

+44 370-871-1111

+44 870-873-6300

Japan (Asia, Pacific Headquarters)

+81 3-4360-8291

Australia

+61 3 9895 2000

[www.flexera.com](http://www.flexera.com)

©2017 Flexera.  All rights reserved.   
 All other brand and product names are trademarks, registered trademarks, or service marks of their respective owners.

©2017 Flexera All rights reserved. All other brand and product names are trademarks, registered trademarks, or service marks of their respective owners.