CMR-CHECKS USER Manual

|  |  |
| --- | --- |
| **Organisational unit:** | [Level] |
| **Owner:** | Framework Team |
| **Version:** | **1.0** |
| **Effective from:** | 01.09.2015 |

|  |  |  |  |
| --- | --- | --- | --- |
| Document Details | | | |
| Name | CMR CHECKS | | |
| (Lead) Author | Andrei Popovici | | |
| Document Owner / OU | Framework Team | | |
| Content Summary |  | | |
| Classification level | [Level] | Superior group document |  |
| Impacted Division | ... | | |
| Application | Legal entities it applies to | | |
| Disclosure | Public | | |
| Publication | Erste Open Network | | |
| Next review date | 1 year after approval/review | | |
| Valid until: | 31.12.15 | | |
| Superior working instruction | Title | | |

|  |  |  |  |
| --- | --- | --- | --- |
| History | Changes | Approval Date | Approved by |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |
| --- | --- |
| Approvals | |
| Approval level required |  |
| Reason for approval request | e.g. change of process |
| Date of final approval | 01.09.2015 |

|  |
| --- |
| Distribution list |
| Unit / Function |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

# Table of Contents

[1. Table of Contents 3](#_Toc427836891)

[2. Summary 4](#_Toc427836892)

[3. Installation Prerequisites 5](#_Toc427836893)

[3.1 Database 5](#_Toc427836894)

[3.2 JVM 5](#_Toc427836895)

[3.3 Physical Location 5](#_Toc427836896)

[4. Installation Steps 6](#_Toc427836897)

[4.1 Database 6](#_Toc427836898)

[4.2 Application 6](#_Toc427836899)

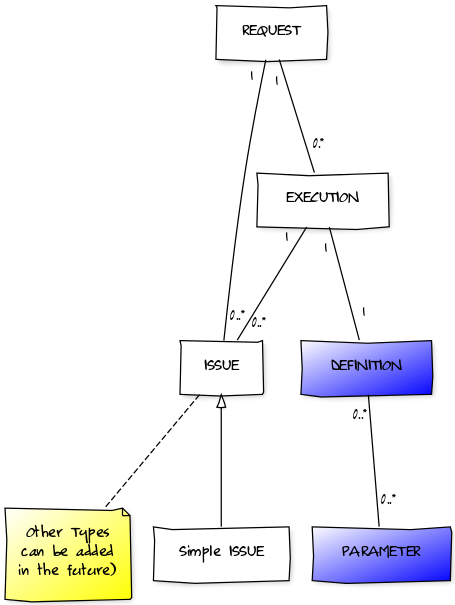
[5. Installation Verification 8](#_Toc427836900)

# Summary

This document will describe the high level concepts and entities used by the CMR-CHECKS. It will also provide for the reader some real-life examples of using this application in day to day activity. The solution design and information about the concepts behind the application can be found in the chapter Solution Presentation. The real-life examples are described in the chapter Use-Cases.

# Solution Presentation

## Logical Model



**Legend:** *White entities are created at runtime, blue entities are configuration entities and they are specified by the user as part of the application’s configuration.*

### PARAMETER

In the application you can define <<PARAMETERS>> and then use the same <<PARAMETER>> in multiple <<DEFINTIONS>>. They can be very useful if you have in your <<DEFINITIONS>> conditions or grouping criteria that are changing from time to time. A good candidate for a parameter is the release number for the framework.

A <<PARAMETER>> has a name and a value, which is represented as string and it is fixed, static. By fixed/static we mean that it cannot be calculate dynamically at runtime.



|  |  |
| --- | --- |
| Attribute Name | Explanation |
| Name | The name of the parameter. Using this name you can refer the <<PARAMETER>> in the <<DEFINITION>> |
| Value | The value that will be used instead of the parameter when the <<DEFINITION>> is executed. |

Example:

Name: RELEASE\_NUMBER

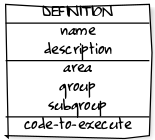
Value: "3.2"

Usage in definition: ${RELEASE\_NUMBER}

The name can contain only alphanumeric characters and the signs "-" and "\_".

### DEFINITION

<<DEFINITION>> is the most important entity from the application. By creating such an entity you define a check, a validation that can be executed at runtime. A diagram with the most important attributes of a definition is presented below.



|  |  |
| --- | --- |
| Attribute Name | Explanation |
| Name | The name given to the <<DEFINITION>>. Used to quickly identify the definition. For this purpose, there is also a Code attribute that can store a short alias, more appropriate for report or usage in application integration scenarios. |
| Description | A short summary of what the <<DEFINTION>> should check for. |
| Area | A way to classify <<DEFINTION>>. You can create a hierarchy of <<DEFINITIONS>> using <Area>, <Group>, and <Subgroup>. This classification can be used to filter the <<DEFINITIONS>> you run at a given moment in time. When you trigger the check after an ODI import of metadata you would want to run a different set of <<DEFINITIONS>> than in the case when you trigger a check before sending the entire release to FAT. An <Area> can have multiple <Groups> which in turn can have multiple <Subgroups> |
| Group | See the explanation for <Area> |
| Subgroup | See the explanation for <Area> |
| Code-to-Execute | An SQL code that will be execute at runtime. The SQL code can contain references to parameters. If the SQL query that results after the parameters are substituted return one or more rows it means that the check has failed, that there is something wrong and an user should investigate and fix whatever is broken. In that case a good name and precise description can come in handy. |

Examples:

Name: First check

Description: A check/validation used for example purposes.

Area: EXAMPLE

Group: **WITHOUT**\_PARAMETER

Subgroup: N/A

Code-To-Execute: select 1 from dual where 1>0

This definition will return a row, so the application will consider that the check has failed.

Name: Second check

Description: A check/validation used for example purposes.

Area: EXAMPLE

Group: **WITHOUT**\_PARAMETER

Subgroup: N/A

Code-To-Execute: select 1 from dual where 1<0

This definition will return no rows, so the application will consider that the check has passed.

Name: Third check

Description: A check/validation used for example purposes.

Area: EXAMPLE

Group: **WITH**\_PARAMETER

Subgroup: N/A

Code-To-Execute: select ${PI} from dual where 1>0

This definition will return one row when is executed, with the value 3.14 (assuming the parameter PI is defined and has the value 3.14 assigned to it).

### REQUEST

An entity created at runtime by the application. It holds some information to identify the requester, the purpose/context of the request, and the method through the request was issued. The purpose of this entity is to create a context, to group/bring together all the <<EXECUTIONS>> of specific <<DEFFINITIONS>> that were requested to run at a specific moment in time.



|  |  |
| --- | --- |
| Attribute Name | Explanation |
| Description | A short description about the purpose of running the requested <<DEFINITIONS>> |
| Requester-code | A code which identify the requester |
| Request-method | A code corresponding to the method through which the <<REQUEST>> was executed. It Is populated by the application and can be used in reporting |

Example:

Description: An example request used to better explain the <<REQUEST>> entity.

Requester-Code: HMN

Requester-Method: CMDL

**Legend:**

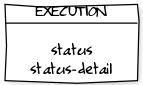
*HMN stands for Human and it is the default value used by the application. If you execute the REQUEST by calling the application from ODI than a more appropriate code will be ODI. It is a good practice to identify with different codes different applications/jobs/code that call the methods exposed by CMR CHECK.*

*CMDL stands for Command Line.*

### EXECUTION

<<EXECUTION>> holds the details about how and when a <<DEFINITION>> was run. It represents the instantiation of a <<DEFINITION>> and the result of the execution for that <<DEFINTION>> at the moment when a <<REQUEST>> was issued. The <<EXECUTION>> entity holds a copy of the <<DEFINITION>> as it was at the moment of the execution, just in case the <<DEFINITION>> changes over time. The status of the <<EXECUTION>> is retained in the attribute <Status> and can have the following values:

|  |  |
| --- | --- |
| Status | Explanation |
| SUBMITED | The <<EXECUTION>> was created and put in the internal queue of the application for processing. |
| IN PROGRESS | The <<EXECUTION>> was picked up from the queue and its processing has started. |
| ERROR | The <<EXECUTION>> processing was finalized and the check has failed (one or more rows were returned by the SQL code) |
| SUCCESS | The <<EXECUTION>> processing was finalized and the check was successful (zero rows were returned by the SQL code) |
| EXCEPTION | The <<EXECUTION>> processing could not be finalized. Something unexpected has happened and the details about what happened will be found in the attribute <Status-Details> |



### ISSUE & SIMPLE ISSUE

<<ISSUE> is an entity created at runtime if the <<EXECUTION>> of a <<DEFINTION>> has the status of *ERROR*. This means that every time a check fails, an <<ISSUE>> is created. <<ISSUE>> is an abstract entity which can have more implementations and each one can hold specific details about the results of the check that was performed. At the moment there is only one such implementation, the <<SIMPLE ISSUE>> which will provide the minimum details about the check that has failed. It does not take into account that the code executed for the <<DEFINITION>> returned 1 or 1000 records, it does not provide additional details. It just says that the check has failed. It is the responsibility of the user to investigate, collect additional details (using the information stored by <<DEFINITION>>) and remediate the problem.

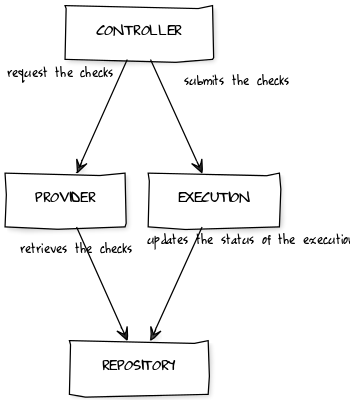
**Note**: *For the sake of brevity only the most important attributes are described. For a detailed list of attributes please look at the physical model and read the comments associated with the columns in the database.*

## Physical Model Considerations

This chapter contains some observations that are applied to this particular implementation of the logical model:

* Every record from every table has a generated primary key, a number generated from the sequence SEQ\_CHECKS.
* Every table has technical columns that provide information about the time when a record was created (INSERT\_TIME), when it was last updated (UPDATE\_TIME) and what user "touched" this record the last time (LAST\_USER). Please bear in mind that the user JUNIT\_DATA\_PREP is reserved and should not be used when creating/updating records.
* The foreign keys are declared in such a way that when you delete a <<REQUEST>> you delete also all the <<EXECUTIONS>> and the <<ISSUES>> associated with that <<REQUEST>>.
* The foreign keys are declared in such a way that when you delete a <<DEFINITION>> the foreign key at the <<EXECUTION>> level is replaced with the value *NULL*.

## High Level Logical Components



|  |  |
| --- | --- |
| Component | Explanation |
| CONTROLLER | A Java module responsible with orchestrating the entire process. It receives the request, then it calls on the << PROVIDER>> to retrieve the <<DEFINTIONS>> that must be executed and it gives those <<DEFINTIONS>> to the execution component <<EXECUTOR>>. The role of the <<CONTROLLER> is to implement the communication between the <<PROVIDER>> and the <<EXECUTOR>>, to monitor that communication and to make sure that they are not tightly coupled so one can be easily replaced without affecting the other. |
| PROVIDER | A Java module that is responsible with identifying the <<DEFINITIONS>> that needs to be executed based on the <<REQUEST>> parameters. |
| EXECUTOR | A Java module that receives the <<DEFINITIONS>> and executes them. In the process it creates and maintains all the runtime entities: <<REQUEST>>, <<EXECUTION>> and <<ISSUE>>. |
| REPOSITORY | The Oracle database where all the configuration and the runtime information is persisted. |

# Use Cases

## Command Line Request

In order to be able to call the application from command line the requester must provide some parameters. There are multiple combinations of parameters that can be used, depending on how much you want to customize your request. You can find in the table below a list with all the parameters that the application accepts. Also the description of all combinations possible is provided later on in the chapter.

All Parameters

|  |  |
| --- | --- |
| Parameter Name | Explanation |
| P\_AREA | The <Area> by which to filter the <<DEFINITIONS>> that must be run under this request. No wildcards are accepted. Mandatory parameter. The search is case insensitive. Because those 3 attributes form a hierarchy you cannot provide a lower level value without providing all values from the upper levels.  For an explanation about how <Area>, <Group>, <Subgroup> are used please see the logical entity <<DEFINITION>> explanation. |
| P\_GROUP | The <Group> by which to filter the <<DEFINITIONS>> that must be run under this request. No wildcards are accepted. Mandatory parameter. The search is case insensitive. Because those 3 attributes form a hierarchy you cannot provide a lower level value without providing all values from the upper levels. If you provide the empty string or null value the parameter will be ignored, meaning filter will not apply.  For an explanation about how <Area>, <Group>, <Subgroup> are used please see the logical entity <<DEFINITION>> explanation. |
| P\_SUBGROUP | The <Subgroup> by which to filter the <<DEFINITIONS>> that must be run under this request, No wildcards are accepted. Mandatory parameter. The search is case insensitive. Because those 3 attributes form a hierarchy you cannot provide a lower level value without providing all values from the upper levels. If you provide the empty string or null value the parameter will be ignored, meaning filter will not apply.  For an explanation about how <Area>, <Group>, <Subgroup> are used please see the logical entity <<DEFINITION>> explanation. |
| RQ\_CODE | The code that identifies the requester. The application will use as default value *HMN* (Human) |
| RQ\_DESC | A description about the purpose of this request. |
| DB\_URL | The JDBC Thin Oracle URL Driver for the database where the application should contain. It can be useful if the application is called only from an external application and the database parameters are kept there. |
| DB\_USER | The username under which the application should connect to the database. It can be useful if the application is called only from an external application and the database parameters are kept there. |
| DB\_PASS | The password of the user under which the application should connect to the database. It can be useful if the application is called only from an external application and the database parameters are kept there. |

Several combinations of parameters are available for launching a request from the command line.

### Combination 1

P\_AREA, P\_GROUP, P\_SUBGROUP

**EXAMPLES**

A request in which the <<DEFINITIONS>> are filtered by <Area> (mandatory parameter)

%installation-location%/CMR-DQI-v1.1/CMR-DQI/bin/CMR-DQI.bat "AREA1" "" ""

A request in which the <<DEFINITIONS>> are filtered by <Area> and <Group>

%installation-location%/CMR-DQI-v1.1/CMR-DQI/bin/CMR-DQI.bat "AREA1" "GROUP" ""

A request in which the <<DEFINITIONS>> are filtered by <Area>, <Group> and <Subgroup>

%installation-location%/CMR-DQI-v1.1/CMR-DQI/bin/CMR-DQI.bat "AREA1" "GROUP" "SUBGROUP"

An invalid request, because you cannot specify a <Subgroup> without specifying a <Group>

%installation-location%/CMR-DQI-v1.1/CMR-DQI/bin/CMR-DQI.bat "AREA1" "" "SUBGROUP"

### Combination 2

P\_AREA, P\_GROUP, P\_SUBGROUP, RQ\_CODE, RQ\_DESC

EXAMPLES:

%installation-location%/CMR-DQI-v1.1/CMR-DQI/bin/CMR-DQI.bat "AREA1" "" "" "ODI" "A description for this example"

### Combination 3

P\_AREA, P\_GROUP, P\_SUBGROUP, RQ\_CODE, RQ\_DESC,DB\_URL, DB\_USER, DB\_PASS

EXAMPLES:

%installation-location%/CMR-DQI-v1.1/CMR-DQI/bin/CMR-DQI.bat "AREA1" "" "" "ODI" "A description for this example" "jdbc:oracle:thin:@localhost:1521:XE" "CMR\_CHECKS" "pass123"

## Create Parameter

If you want to create a <<PARAMETER>> you need to insert a new record in the table CMR\_CHECK\_PARAMETER. The following rules must be taken into consideration:

* A <<PARAMETER>> must have both a name and a value.
* A <<PARAMETER>> name can contain only letter, number *dash* or *underscore* characters. This constraint expressed as a regular expression will look like this [a-zA-Z0-9,-,\_]+

The following script can be used to create a new parameter. You must connect with the user that owns the database objects created for the application. The values that you must replace are marked as parameters in the SQL script.

**insert** **into** cmt\_check\_parameter  
(PRM\_ID,PRM\_TYPE,PRM\_KEY,PRM\_VALUE,PRM\_REFRESH\_FLAG,LAST\_USER)  
**values** (seq\_checks.**nextval**, 'FIXED', '&PARAM\_NAME', '&PARAM\_VALUE', 'N','&USER');

## Create Definition

If you want to create a <<DEFINITION>> you can use the following script. Be sure to connect to the database with the user that owns the database objects created for the application. The values that you must replace are marked as parameters in the SQL script. For details about the significance of each parameter/field please read the in this manual the documentation about the <<DEFINITION>> entity.

**insert** **into** CMT\_CHECK\_DEFINITION  
(CHK\_ID, CHK\_NAME, CHK\_CODE, CHK\_DESCRIPTION, CHK\_AREA,CHK\_GROUP,CHK\_SUBGROUP, CHK\_EXECUTION\_FREQUENCY, CHK\_SQL, LAST\_USER )  
**values**   
(seq\_checks.**nextval**, '&CHK\_NAME', '&CHK\_NAME', '&CHK\_DESCRIPTION', '&CHK\_AREA','&CHK\_GROUP','&CHK\_SUBGROUP', 'DAILY', '&SQL\_TO\_EXECUTE', '&USER');

## Delete Old Data

Because of the foreign constraints definition, if you want to delete old data, all you have to do is delete old <<REQUESTS>>. The database engine will also delete <<EXECUTIONS>> and <<ISSUES>>. You can use the following syntax to achieve that. The values that you must replace are marked as parameters in the SQL script.

**DELETE** **FROM** CMT\_CHECK\_REQUEST **where** trunc(UPDATE\_TIME)<trunc(to\_date('&date','YYYY-MM-DD'))