REPSTANCE

Prepare to be Amazed

Repstance Advanced Edition

User Guide

2020

Collabcloud Limited
47 St Pauls Road, Staines-upon-Thames, Surrey, TW18 3HQ, England

TABLE OF CONTENTS

1. ABOUT REPSTANCE

1.1 What it is

Repstance is an entirely new real-time data propagation product for homogeneous and heterogeneous environments that supports various replication topologies.

Repstance Advanced Edition supports Oracle, SQL Server (with the ability to enable CDC) as a Source and Oracle, SQL Server, PostgreSQL, MySQL, Aurora (PostgreSQL, MySQL) as a Target.

Repstance includes the functionality to support DML and DDL operations, both of which can be automatically included in the replication stream and if desired uses of sophisticated transformation abilities. Fast initial data loading and the ability to synchronize data from any desired timestamp. Flexible third-party product integration (APIs, CLI, custom integration). Ability to easily configure and monitor replication using either GUI or CLI tool.

1.2 How it works

The Service is delivered via an "Amazon Machine Image" which runs within its own separate Amazon Instance, this is a Linux based instance and runs as a "Daemon" within this environment. Repstance has to have access to both Source and Target Databases, in addition it needs a set of minimal Database Objects to function. This functionality is added during the "Prepare Database" stage (see chapter 7.1 Prepare Source and Target Databases).

Repstance uses the running Daemon process to "Multi-task" – that is to say, by using "threads" within the Daemon it is capable of running Multiple Capture and Apply Processes concurrently.

Both of the Processes, while they depend on each other to supply or insert the necessary data they, nonetheless, run as independent threads.

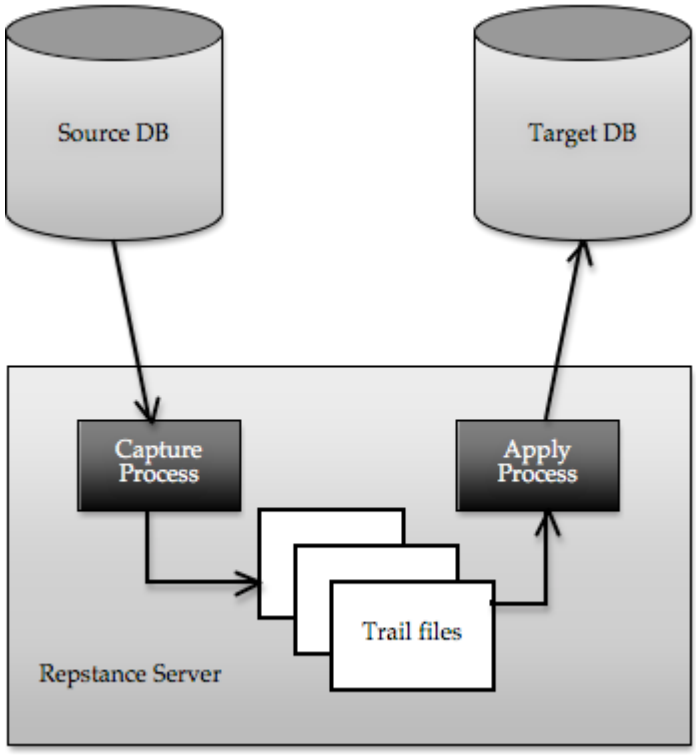
The Capture Process is the means by which the data to be transferred is extracted from the Source Database. It puts this information into locally stored Trail Files, which the Apply Processes can use. The data from a single Capture Process can therefore be used by many Apply Processes and, as a result, this data can therefore be propagated into multiple Target Databases.

The data extracted by the Capture Process is written to these Trail Files in the same sequential order as the transactions occurred in the Source Database, which in turn allows the Apply Process to insert this data into the Target Database in the same order they were executed in the Source Database. This means that both Source and Target Databases will be synchronized, that is to say, in a "Consistent State".

The Capture Process does not need to have a running or configured Apply Process as it will quite simply continue extracting data using the criteria supplied, conversely the Apply Process only needs valid Trail Files from a Capture Process to consume.

Both Capture and Apply Processes insert Checkpoints in the form of a LSN/SCN – this in turn means that there is the possibility for Repstance to be restarted from any specified point in the circumstances where any unexpected event occurs that may lead to data loss. This provides a robust form of data security and ensures that the data is, again, always "Consistent".

The diagram below demonstrates the flow of data from Source to Target Databases.



2. GETTING STARTED WITH REPSTANCE IN AWS

The Repstance Server is delivered as an AMI paid instance and is accessible from the AWS Marketplace, using the "1 Click " download process here:

https://aws.amazon.com/marketplace/pp/B086GPVHTF

It requires you as a user to have your own AWS account and be subscribed to the product, and select AWS instance of sufficient size and processing power to manage your database requirements (see vendor recommended instance size).

During the AWS installation process you must ensure that TCP port "22" is opened for access, as a minimum, and in the case that you want remote access, that port "8797" is also opened. In order to use GUI port "3000" should be opened as well.

To connect to the operating system, use SSH protocol. The only username that can be used is "**ec2-user**" along with the key that you were provided with during the "launch" of the installation process. See the following link for help:

https://docs.aws.amazon.com/en_us/AWSEC2/latest/UserGuide/AccessingInstancesLinux.html

Once it is installed in your environment, it is ready to run. The product can be managed using the **repcli** interface from within this environment.

Now **repcli** can be invoked directly, and the user can start to configure Processes without the need for any further configuration. Run **repcli** directly on the server:

```
$ repcli
$ repcli>help

Commands:
  alter         This command is used to change both the Capture and Apply
process parameters

  clear         Clear The Screen

  exit          Exit The Program

  help          Display Help
```

etc.

It can also be accessed remotely using the **repcli** client versions, which are found in the following directories, once the product is installed:

- `/opt/repstance/cli/macos` – For MacOS Operating System
- `/opt/repstance/cli/win` – For Windows
- `/opt/repstance/cli/linux` – For Linux

Or they can be downloaded from here:

- https://repstance.com/download/cli/macos/repcli.zip
- https://repstance.com/download/cli/win/repcli.zip
- https://repstance.com/download/cli/linux/repcli.zip

These remote clients are only able to connect using the HTTPS protocol and are supplied with the following default configuration:

- HTTPS port is 8797
- For HTTPS initial use a Self signed certificate is provided
- HTTPS authorisation "Token" in the form of the Repstance AWS Instance ID

An alternative to either of the above communication methods is to call the REST API detailed in this document.

2.1 Configure repcli for Remote Access

In order to use the **repcli** utility to communicate with the remote instance it has to be configured using the **repcli** command "configure".

The following example demonstrates the command syntax:

```
$ repcli configure url=https://hostname:8797/ token=aws-instance-id
verifyhttps=0
```

Note – The **`verifyhttps`** parameter should be set to **0** to avoid the Self-signed certificate error.

2.2 Configure Repstance Server to Accept Remote Connection

As the product is initially supplied with a Self-Signed Certificate it is highly recommended that the user replaces this with a security certificate of the required Security level themselves. The new certificate and key needs to be supplied to the Repstance Server and the appropriate entries in the `/opt/repstance/conf/repstance.conf` file need to be edited to reflect these changes:

- `key` – Path to the key
- `crt` – Path to the certificate

Using this method the default HTTPS port can also be changed by modifying the **port** parameter.

The default token used to access the server can also be changed by modifying the **token** parameter in the same file.

If the user wishes to prevent any external access, the **port** parameter should be set to value **0**.

3. INTERACTING WITH A REPSTANCE SERVER

There are currently only two possible ways to engage with a running Repsance Server Instance, they are, either via the Rest API or via the Command Line Interface (CLI).

3.1 REST API

This is the detailed command structure for communicating with Repstance Services, it sends an HTTP/HTTPS request in the "JSON Format" to the services and adheres to the established protocols for the JSON format. List of the available Repstance commands and their specification is provided in chapter 7. COMMANDS TO BE USED.

The following example demonstrates how to use REST API to call "**status"** command:

```
$curl --header "Content-Type: application/json" \
 --request POST \
 --data '{"command": "status"}' http://localhost:8796/control/process
```

3.2 Command Line Interface (CLI)

The Repstance Command Line Interface utility (**repcli**) is the interface with the Repstance environment and it is used to configure and manage the environment as well as providing a reporting mechanism, and once connected to a Repstance server it can be used with a predefined set of "Command" parameters, which are parsed by Repstance to ensure validity. Their specific use is detailed below.

It is delivered together with a Repstance Server Instance and available in the "**bin"** folder under Repstance home directory. By default it establishes network layer connection to the Repstance Server using IP address 127.0.0.1 (localhost).

The Repstance Server can be configured to accept an HTTPS connection by external means. The **repcli** utility can be configured to communicate with remote Repstance Servers also using the HTTPS protocol (see chapter 2.1 Configure **repcli** for Remote Access).

The **repcli** utility can operate in two modes either "Command Mode" or "Fully Interactive Mode". The two different ways of using Repstance have been developed to provide alternate methods of controlling the product.

If **repcli** is executed without any parameters it starts functioning in "Fully Interactive Mode", by default, which means that it will wait for an input command, until user issues the **"exit"** command at which point it returns to the console.

The "Fully Interactive Mode" is useful for a user to construct and/or configure Repstance, as it provides, at each stage of the process construction, a list of the possible commands and their applicable parameters, see the "Visuals" below.

"Command mode" is more likely to be used when there is a need to "Embed" Repstance functions into a deployment Script for Automation processes within the user's own environment.

3.2.1 Fully Interactive Mode

This is the utility's default behavior. As the user starts to provide a command for any Process and then "presses" the "Tab" key, **repcli** utility will supply the user with the expected possible parameters.

The following example shows the **repcli** interface displaying the list of all possible commands:

```
repcli>
status    exit   help    prepare   remove    alter    reset    clear     validate
show    run    stop
```

This example shows all commands starting with "s":

```
repcli>s
status   show   stop
```

And the next one shows all the possible parameters for the "**alter**" command:

```
repcli>alter process=capture id=1
server=         port=       user=          password=       dbname=          name=
autostart=   debuglevel=   ddlinclude=   ddlexclude=   dmlinclude= dmlexclude=
map=
```

The **repcli** has built-in help facility, which can be invoked to provide detailed "Help" for each of the commands, and is shown below:

```
repcli>alter help

This command is used to change the Capture or Apply process parameters.

The Syntax is:

      alter process=capture|apply id=processID [parameter=value, ...]

Note: The 'id' parameter can not be changed.
      The 'name' parameter for a Capture process as well as the 'capturename'
parameter for an Apply process can be changed only in the case that it has
not been provided before - eg. it must be "Unique".
      The process has to exist otherwise the command will fail.

      To see available possible parameters, start to use the command.
```

3.2.2 Command Mode

This is the basic interface format, when used in this mode it will simply supply the result of the process defined and return to the console prompt:

```
$repcli status process=capture

Capture Process 1, Name: czdt, DB: zdt, Server: source.repstance.com
  RUN (In Process), Waiting for Transactions
  Last Source DB Change : 2019/02/15 15:47:00.410 UTC (0x000006830000319C0004)
  Last Captured DB Change  :  2019/02/15 15:47:00.410 UTC (0x000006830000319C0004)
Total since  2019/02/25 13:30:00.973 UTC (0x00006830000025750004):
  Processing the Transactions :  00:00:58.505
  Waiting for Transactions    :  00:45:22.859
  Average Speed (ops)     :  4.683
  Transactions: 274 (DDL: 0,  Delete: 137,  Insert: 137,  Update: 0)

Capture Process 2, Name: newcap, DB: mydb, Server: local
  RUN (Failed), Waiting for the Next Command, [lookup local: no such host]

$
```

4. SUPPORTED DATABASE REQUIREMENTS

4.1 SQL Server Database

Currently the product supports the following database as a Source:

SQL Server – either Enterprise Editions starting 2008 or Standard Editions from 2016 onwards.

Note – As not ALL versions of SQL Server allow **C**hange **D**ata **C**apture functionality, only those which **DO allow CDC functionality** are **currently supported**.

Only user databases are supported.

The SQL Server Agent must be running.

Restrictions:

- A system database is not supported
- No Contained databases are supported
- No databases enabled with In-Memory Optimization (2014/2016 feature) are currently supported
- Database Compatibility level must be 100 or higher
- Asynchronous AlwaysOn databases are not supported

Both "EC2" and "RDS" usage for Source and/or Target Databases are supported.

Any SQL Server databases starting 2008, including Azure SQL Server instances are supported as a Target.

4.2 Oracle Database

Currently Repstance supports the following:

Oracle – versions 10g through 19c.

All of the above versions must be configured to run in ARCHIVELOG mode prior to use.

It is recommended practice to ensure that the archive logs are not "cleaned-up" until they are processed by the Repstance server.

Both "EC2" and "RDS" usage for Source and/or Target Databases are supported.

4.3 MySQL Database

Currently the product supports MySQL as a Target database only. The following versions are supported:

- MySQL versions 5.5-5.7
- MySQL 8.0
- Aurora MySQL
- MariaDB versions 10.0 – 10.3

4.4 PostgreSQL Database

Currently the product supports PostgreSQL as a Target database only. The following versions are supported:

- PostgreSQL versions 10.1-11.6
- Aurora PostgreSQL compatible with PostgreSQL 10.x.

5. HOW TO USE REPSTANCE

Before running the replication both Source and Target Databases need to be prepared. It is necessary to enable CDC for SQL Server database and enable "supplemental logging" for Oracle. The Repstance's objects to support the replication need to be installed as well.

For the Target Database only the Repstance's objects have to be installed. Repstance has built-in functionality that allows it to prepare databases by simply running the CLI or REST API command (see chapter 7.1 Prepare Source and Target Databases).

Once it is done we recommend to validate both databases by running the "**validate**" command (see chapter 7.3 Validate Source and Target Database) to make sure that the necessary functionality is installed.

The next step is to configure the replication processes. This is done in two parts, the first is to configure the Capture Process (see chapter 7.4 Prepare Capture Process), which is extracting the data from the specified Source Database, and the second is to configure the corresponding Apply Process (see chapter 7.9 Prepare Apply Process), which will insert the "captured" data into the specified Target Database.

The data to be replicated will be defined during the Capture Process configuration. Validation of a correctly constructed Capture or Apply Processes can be checked using the "**validate**" command (see chapters 7.7 Validate Capture Process and 7.12 Validate Apply Process).

For SQL Server the Capture Process will enable CDC on the specified tables only the first time it is ran. If the Capture Process is "altered" at any point, during the next "run" it will disable CDC on the unwanted tables and initialise CDC on any new tables that are specified.

The first time the Capture Process is run, is the first point and the only point at which data will start to be captured by this process.

Assuming that the Capture Process has been properly configured and run, only at this point will it start to produce Trail Files, which can be used by Apply Process.

In order for an Apply Process to succeed, of necessity the Trail Files of the corresponding Capture Process must exist.

In order to ensure that an Apply Process will start from the specified LSN/SCN, this LSN/SCN must exist in the Trail Files and the "**reset"** command (see chapter 7.13 Reset Apply Process) can be used to point this Apply Process at the required LSN/SCN. If the Apply Process has not been reset, it will start processing data from the first available transaction found in the Trail File.

Once the Apply Process has been configured to use and the proper LSN/SCN (if needed) has been specified, the Apply Process can be run (see chapter 7.15.1 Run any Capture or Apply Processes).

At the point when both the Capture and Apply Processes are running, the "**status**" command (see chapter 7.15.3 Status of any Capture or Apply Processes) can be used to monitor them.

In the event it is necessary to "stop" any currently running processes use the "**stop**" command (see chapter 7.15.2 Stop any Capture or Apply Processes).

The following example demonstrates how to setup simple DML and DDL replications between two SQL Server databases for all objects located in the **dbo** schema:

```
repcli>prepare database=source dbtype=mssql server=source.repstance.com
port=1433 user=ds password=secret dbname=sdemo
[Completed]: Database [source.repstance.com:1433,sdemo] has been prepared

repcli>prepare database=target dbtype=mssql dbtype=mssql
server=target.repstance.com port=1433 user=ds password=secret dbname=tdemo
[Completed]: Database [target.repstance.com:1433,tdemo] has been prepared

repcli>validate database=source dbtype=mssql server=source.repstance.com
port=1433 user=ds password=secret dbname=sdemo
[Valid]: Database SDEMO is valid

repcli>validate database=target dbtype=mssql server=target.repstance.com
port=1433 user=ds password=secret dbname=tdemo
[Valid]: Database TDEMO is valid

repcli>prepare process=capture dbtype=mssql id=1 name=rep1
server=source.repstance.com port=1433 user=ds password=secret dbname=sdemo
dmlinclude=dbo.% ddlinclude=dbo.%
[Completed]: Capture 1 has been added

repcli>validate process=capture id=1
[Valid]: Capture process 1 is valid

repcli>run process=capture id=1
[In Process]: Command [RUN] has been sent. Use [status] command to monitor
the Capture 1 process

repcli>prepare process=apply id=1 dbtype=mssql capturename=rep1
server=target.repstance.com port=1433 user=ds password=secret dbname=tdemo
repuser=1
[Completed]: Apply 1 has been added

repcli>validate process=apply id=1
[Valid]: Apply process 1 is valid

repcli>run process=apply id=1
[In Process]: Command [RUN] has been sent. Use [status] command to monitor
the Apply 1 process

repcli>status
Capture Process 1, Name: rep1, DB: sdemo, Server: source.repstance.com
```

```
  RUN (In Process), Waiting for Transactions
    Last Source DB Change : 2019/02/25 21:13:00.173 UTC (0x00000044000002AE0004)
    Last Captured DB Change : 2019/02/25 21:13:00.173 UTC (0x00000044000002AE0004)
  Total since  2019/02/25 21:10:00.983 UTC (0x000000440000020A0004):
    Processing the Transactions :  00:00:05.289
    Waiting for Transactions    :  00:02:44.396
    Average Speed (ops)  :  1.134
    Transactions: 6 (DDL: 0,  Delete: 3,  Insert: 3,  Update: 0)

Apply Process 1, Transactions Provider (Capture): rep1, DB: tdemo, Server:
target.repstance.com
  RUN (In Process), Waiting for Transactions
     Last Source DB Change  : 2019/02/25 21:13:00.173 UTC (0x00000044000002AE0004)
     Last Applied DB Change : 2019/02/25 21:13:00.173 UTC (0x00000044000002AE0004)
     Lag                    :  00:00:00.000
  Total since  2019/02/25 21:13:00.150 UTC (0x00000000000000000001):
    Processing the Transactions :  00:00:02.065
    Waiting for Transactions    :  00:00:08.284
    Average Speed (ops)  :  2.905
    Transactions: 6 (DDL: 0,  Delete: 3,  Insert: 3,  Update: 0)
```

6. HOW TO REMOVE REPSTANCE

The steps involved in removing Repstance are detailed below.

6.1 How to remove Apply Process

To remove an Apply Process use the "**remove**" command (see chapter 7.14 Remove Apply Process).

After using this command it will remove **all** information about the Apply Process, namely configuration files, and the checkpoint details, together with historical information. If the Target Database **is** reachable the "**remove**" command will clean-up all the checkpoint information on the Target Database.

As the "**remove"** command can only accept a single "Apply parameter", this action can only be used for one Apply Process at a time.

6.2 How to remove Capture Process

To remove a Capture Process use the "**remove**" command (see chapter 7.8 Remove Capture Process).

After using this command it will remove **all** information about the Capture Process, namely configuration files, checkpoint details, and the Trail Files. If the Source Database is SQL Server and it **is** reachable the "**remove**" command will disable CDC on the tables that were used by this Capture Process only.

As the "**remove**" command can only accept a single "Capture parameter", this action can only be used for one Capture Process at a time.

Note – If there are still any Apply Processes using Trail Files from this Capture Process, they will fail.

The steps involved in removing Repstance Server Database Objects are detailed below.

6.3 Remove Target Database Objects

To remove Target Database Objects use the "**remove**" command (see chapter 7.2 Remove Repstance Database Objects).

After using this command it will remove all the Data Objects that were required by the Apply Process. If there are still Apply Processes inserting data, using these objects they will immediately fail.

Note – It is strongly recommended that before running this command you run the "**status**" command and make a record the LSN/SCN in the "Last Applied DB Change"

section for each Apply Process running on this database, in this way it will be possible to reverse this action for this database at a later point in time, assuming that the correct Trail Files still exist and can be accessed.

6.4 Remove Source Database Objects

To remove Source Database Objects use the "**remove**" command (see chapter 7.2 Remove Repstance Database Objects).

Note – When using this command it MUST be understood that there will be no way to "Recover" from its effects.

After using this command it will remove all the Data Objects that were required by the Capture Processes. If there are still Capture Processes extracting data, using these objects they will immediately fail. For SQL Server it will disable CDC on this database. At this point Repstance will no longer be able to record or extract data from this database.

7. COMMANDS TO BE USED

7.1 Prepare Source and Target Databases

Both Source and Target Databases must be prepared for replication before running any replication processes. The "Prepare" database command is used to configure databases. The command must be executed on both Source and Target Databases.

7.1.1 Prepare SQL Server Database as a Source Database

The "**prepare**" database command is used to configure SQL Server database in order to use it as a Source. It enables CDC and creates the Database Objects necessary for any Capture Processes.

Note – In order to use the "**prepare**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://*repstance_url*/configure/database
- **Method:** POST
- **Header:**
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
   "command":"prepare",
   "parameters":[
      ["database","source "],
      ["dbtype","mssql"],
      ["server","host_name"],
      ["port","port_number"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"],
      ["housekeeping","0|1|2"]
   ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Completed|Failed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare database=source dbtype=mssql \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName housekeeping={0|1|2}
```

The input parameters are:

- **database** – Database role. The only possible value is *source* – to insert Capture Process objects and enable CDC
- **dbtype** – Type of the Source database. The value is *mssql* – SQL Server database
- **server** – Host name or IP address of the database server
- **port** – Database port number
- **dbname** – Database name to be connected to
- **user** – Database user name
- **password** – Database user password
- **housekeeping** – This parameter determines how the CDC data will be cleaned up. The possible values are:
  - *0* – Data will be cleaned up by an SQL Server job only
  - *1* – Install Repstance job to clean up data only after extraction and keep SQL Server cleanup job, which is a time based CDC setting
  - *2* – Install Repstance job and remove SQL Server cleanup job. Data will be cleaned up only by the Repstance job.

7.1.2 Prepare Oracle Database as a Source Database

The "**prepare**" database command is used to configure Oracle database in order to use it as a Source. It enables the necessary level of supplemental logging and creates the database objects necessary for any Capture Processes.

Note – In order to use the "Prepare" database command the database user must have sufficient privileges.

There are two possible connection types - EZCONNECT and TNS. Depending on which one is chosen, there will be a different set of possible parameters.

REST API:

- **Endpoint:** https://*repstance_url*/configure/database
- **Method:** POST
- **Header:**
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
   "command":"prepare",
   "parameters":[
      ["database","source"],
```

```
        ["dbtype","oracle"],
        ["connectiontype","tns|ezconnect"],
        ["tnsname","tns_alias"],
        ["server","host_name"],
        ["port","port_number"],
        ["servicename","service_or_SID"],
        ["dbname","database_name"],
        ["user","db_user_name"],
        ["password","db_user_password"],
        ["tablespace","user_tablespace"]
    ]
}
```

Server response:

- **HTTP Status** - the status of command. The possible codes are:
  - **200** – if no error occurs
  - **422** – if error occurs
- **Body**:

```
{
  "Status":"{Completed|Failed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare database=source \
      dbtype=oracle \
      connectiontype=tns|ezconnect \
      tnsname=tns_alias \
      server=databaseHost port=databasePort \
      servicename=service_or_SID
      user=username password=password \
      dbname=databaseName \
      tablespace=user_tablespace
```

The input parameters are:

- **database** – Database role. The possible value is *source* – to insert Capture Process objects
- **dbtype** – Type of RDBMS. The only possible value is *oracle*
- **connectiontype** – Specifies the Oracle connection method. The possible values are:
  - *tns* – Local Naming Method to be used
  - *ezconnect* – EZCONNECT to be used
- **tnsname** – Name of the TNS alias. Valid **only** if connectiontype=*tns*
- **server** – Host name or IP address of the database server. Valid **only** if connectiontype=*ezconnect*
- **port** – Database port number. Valid **only** if connectiontype=*ezconnect*
- **servicename** – Database service name or SID. Valid **only** if connectiontype=*ezconnect*

- **`dbname`** – Name of either container or pluggable database. Valid **only** if `database=source` and for any Oracle versions 12c-19c but excluding RDS instances
- **`user`** – Database user name
- **`password`** – Database user password
- **`tablespace`** – Name of the tablespace that the Repstance's objects are to be installed in. The default is the USERS tablespace if no alternative has been specified

7.1.3 Prepare SQL Server Database as a Target

The "Prepare" database command is used to configure SQL Server database as a Target. The command must be executed before running Apply process. It creates the necessary database objects for any Apply Processes.

Note – In order to use the "**prepare**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** `https://repstance_url/configure/database`
- **Method:** `POST`
- **Header:**
  - `Content-Type: application/json`
  - `X-Token: token`
- **Body**:

```
{
   "command":"prepare",
   "parameters":[
      ["database","target"],
      ["dbtype","mssql"],
      ["server","host_name"],
      ["port","port_number"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
   ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **`200`** – If no error occurs
  - **`422`** – If error occurs
- **Body**:

```
{
  "Status":"{Completed|Failed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare database=target dbtype=mssql \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role. The only possible value is *target* – to insert Apply Process objects
- **dbtype** – Type of RDBMS. The only appropriate value is *mssql*
- **server** – Host name or IP address of the database server
- **port** – Database port number
- **dbname** – Database name to be connected to
- **user** – Database user name
- **password** – Database user password

7.1.4 Prepare Oracle Database as a Target

The "Prepare" database command is used to configure Oracle database as a Target. The command must be executed before running Apply process. It creates the necessary database objects for any Apply Processes.

Note – In order to use the "Prepare" database command the database user must have sufficient privileges.

There are two possible connection types - EZCONNECT and TNS. Depending on which one is chosen, there will be a different set of possible parameters.

REST API:

- **Endpoint:** https://*repstance_url*/configure/database
- **Method:** POST
- **Header:**
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
   "command":"prepare",
   "parameters":[
      ["database"," target"],
      ["dbtype","oracle"],
      ["connectiontype","tns|ezconnect"],
      ["tnsname","tns_alias"],
      ["server","host_name"],
      ["port","port_number"],
      ["servicename","service_or_SID"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"],
      ["tablespace","user_tablespace"]
```

```
    ]
}
```

Server response:

- **HTTP Status** - the status of command. The possible codes are:
  - **200** – if no error occurs
  - **422** – if error occurs
- **Body**:

```
{
  "Status":"{Completed|Failed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare database=target \
      dbtype=oracle \
      connectiontype=tns|ezconnect \
      tnsname=tns_alias \
      server=databaseHost port=databasePort \
      servicename=service_or_SID \
      user=username password=password \
      dbname=databaseName \
      tablespace=user_tablespace
```

The input parameters are:

- **database** – Database role. The possible values are:
  - *source* – to insert Capture Process objects
  - *target* – to insert Apply Process objects
- **dbtype** – Type of RDBMS. The only possible value is *oracle*
- **connectiontype** – Specifies the Oracle connection method. The possible values are:
  - *tns* – Local Naming Method to be used
  - *ezconnect* – EZCONNECT to be used
- **tnsname** – Name of the TNS alias. Valid **only** if connectiontype=*tns*
- **server** – Host name or IP address of the database server. Valid **only** if connectiontype=*ezconnect*
- **port** – Database port number. Valid **only** if connectiontype=*ezconnect*
- **servicename** – Database service name or SID. Valid **only** if connectiontype=*ezconnect*
- **dbname** – Name of either container or pluggable database. Valid **only** if database=*source* and for any Oracle versions 12c-19c but excluding RDS instances
- **user** – Database user name
- **password** – Database user password
- **tablespace** – Name of the tablespace that the Repstance's objects are to be installed in. The default is the USERS tablespace if no alternative has been specified.

7.1.5 Prepare PostgreSQL and Aurora PostgreSQL Databases as a Target

The "Prepare" database command is used to configure PostgreSQL and Aurora PostgreSQL databases as a Target. The command must be executed before running Apply Process. It creates the necessary database objects for any Apply Processes.

Note – In order to use the "**prepare**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://*repstance_url*/configure/database
- **Method:** POST
- **Header:**
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
   "command":"prepare",
   "parameters":[
      ["database","target"],
      ["dbtype","postgresql"],
      ["server","host_name"],
      ["port","port_number"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
   ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Completed|Failed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare database=target dbtype=postgresql \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role. The only possible value is *target* – to insert Apply Process objects
- **dbtype** – Type of RDBMS. The only value is *postgresql*

- **server** – Host name or IP address of the database server
- **port** – Database port number
- **dbname** – Database name to be connected to
- **user** – Database user name
- **password** – Database user password

7.1.6 Prepare MySQL and Aurora MySQL Databases as a Target

The "Prepare" database command is used to configure MySQL, MariaDB and Aurora MySQL databases as a Target. The command must be executed before running Apply process. It creates the necessary database objects for any Apply Processes.

Note – In order to use the "**prepare**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://*repstance_url*/configure/database
- **Method:** POST
- **Header:**
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
   "command":"prepare",
   "parameters":[
      ["database","target"],
      ["dbtype","mysql"],
      ["server","host_name"],
      ["port","port_number"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
   ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Completed|Failed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare database=target dbtype=mysql \
     server=databaseHost port=databasePort \
```

```
user=username password=password \
dbname=databaseName
```

The input parameters are:

- **database** – Database role. The possible value is *target* – to insert Apply Process objects
- **dbtype** – Type of RDBMS. The possible value is *mysql*
- **server** – Host name or IP address of the database server
- **port** – Database port number
- **dbname** – Database name to be connected to
- **user** – Database user name
- **password** – Database user password

7.2 Remove Repstance Database Objects

The "**remove**" database objects command is used to remove Repstance database replication objects.

The successful result of using this command will be:

- For the Source and Target Database, it will remove the Repstance Database Objects that were created by the "**prepare**" command;
- For SQL Server Database configured as a Source, it will also disable CDC on the tables used by Repstance.

After successful completion of the command any remaining running Processes will fail.

7.2.1 Remove Repstance Database Objects in SQL Server

Note – In order to use the "**remove**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://*repstance_url*/configure/database
- **Method**: POST
- **Header**:
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
   "command":"remove",
   "parameters":[
      ["database","source|target"],
      ["dbtype","mssql"],
      ["server","host_name"],
      ["port","port_number"],
      ["dbname","database_name"],
```

```
        ["user","db_user_name"],
        ["password","db_user_password"]
    ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
   "Status":"{Completed|Failed}",
   "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli remove database=source|target dbtype=mssql \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role. The possible values are:
  - *source* – to remove Capture Process objects and disable CDC
  - *target* – to remove Apply Process objects
- **dbtype** – Type of RDBMS. The possible value is *mssql*
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database user name
- **password** – Database user password

7.2.2 Remove Repstance Database Objects in Oracle

Note – In order to use the "**remove**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://*repstance_url*/configure/database
- **Method**: POST
- **Header**:
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
   "command":"remove",
   "parameters":[
```

```
        ["database","source|target"],
        ["dbtype","oracle"],
        ["connectiontype","tns|ezconnect"],
        ["tnsname","tns_alias"],
        ["server","host_name"],
        ["port","port_number"],
        ["servicename","service_or_SID"],
        ["dbname","database_name"],
        ["user","db_user_name"],
        ["password","db_user_password"]
       ]
}
```

Server response:

- **HTTP Status** - The status of command. The possible codes are:
  - **200** - if no error occurs
  - **422** - if error occurs
- **Body**:

```
{
   "Status":"{Completed|Failed}",
   "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli remove database=source|target \
      connectiontype=tns|ezconnect \
      tnsname=tns_alias \
      server=databaseHost port=databasePort \
      servicename=service_or_SID
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role. The possible values are:
  - *source* – to remove Capture Process objects
  - *target* – to remove Apply Process objects
- **dbtype** – Type of RDBMS. The possible value is *oracle*
- **connectiontype** – Specifies the Oracle connection method. The possible values are:
  - *tns* – Local Naming Method to be used
  - *ezconnect* – EZCONNECT to be used
- **tnsname** – Name of the TNS alias. Valid **only** if `connectiontype=tns`
- **server** – Host name or IP address of the database server. Valid **only** if `connectiontype=ezconnect`
- **port** – Database port number. Valid **only** if `connectiontype=ezconnect`
- **servicename** – Database service name or SID. Valid **only** if `connectiontype=ezconnect`
- **dbname** – Name of either container or pluggable database. Valid **only** if `database=source` and for any Oracle versions 12c-19c but excluding RDS instances

- **user** – Database user name
- **password** – Database user password

7.2.3 Remove Repstance Database Objects in PostgreSQL and Aurora PostgreSQL Databases

---

Note – In order to use the "**remove**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://*repstance_url*/configure/database
- **Method**: POST
- **Header**:
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
   "command":"remove",
   "parameters":[
      ["database","target"],
      ["dbtype","postgresql"],
      ["server","host_name"],
      ["port","port_number"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
    ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
   "Status":"{Completed|Failed}",
   "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli remove database=target dbtype=postgresql \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role. The possible value is *target* – to remove Apply Process objects

- **dbtype** – Type of RDBMS. The possible value is *postgresql*
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database user name
- **password** – Database user password

7.2.4 Remove Repstance Database Objects in MySQL and Aurora MySQL Databases

---

Note – In order to use the "**remove**" database command the database user must have sufficient privileges.

REST API:

- **Endpoint:** https://*repstance_url*/configure/database
- **Method**: POST
- **Header**:
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
   "command":"remove",
   "parameters":[
      ["database","target"],
      ["dbtype","mysql"],
      ["server","host_name"],
      ["port","port_number"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
    ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
   "Status":"{Completed|Failed}",
   "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli remove database=target dbtype=mysql \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **`database`** – Database role. The possible value is *`target`* – to remove Apply Process objects
- **`dbtype`** – Type of RDBMS. The possible value is *`mysql`*
- **`server`** – Host name or IP address of the database server
- **`port`** – Database port
- **`dbname`** – Database name
- **`user`** – Database user name
- **`password`** – Database user password

7.3 Validate Source and Target Database

The "**validate**" database objects command is used to validate that the correctly initialized Database Objects exist, and Database is ready for replication.

It is considered to be "Good Practice" to run this command after either the initial configuration has been completed or if changes have been made to either a Capture or Apply Process.

7.3.1 Validate SQL Server Database

REST API:

- **Endpoint:** `https://repstance_url/configure/database`
- **Method:** `POST`
- **Header:**
  - `Content-Type: application/json`
  - `X-Token: token`
- **Body**:

```
{
   "command":"validate",
   "parameters":[
      ["database","source|target"],
      ["dbtype","mssql"],
      ["server","host_name"],
      ["port","port_number"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
   ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **`200`** – If no error occurs
  - **`422`** – If error occurs
- **Body**:

```
{
```

```
  "Status":"{Valid|Invalid}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli validate database=source|target dbtype=mssql \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role. The possible values are:
  - *source* – to install Capture Process objects
  - *target* – to install Apply Process objects
- **dbtype** – Type of RDBMS. The possible value is *mssql*
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database user name
- **password** – Database user password

7.3.2 Validate Oracle Database

REST API:

- **Endpoint:** https://*repstance_url*/configure/database
- **Method:** POST
- **Header:**
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
   "command":"validate",
   "parameters":[
      ["database","source|target"],
      ["dbtype","oracle"],
      ["connectiontype","tns|ezconnect"],
      ["tnsname","tns_alias"],
      ["server","host_name"],
      ["port","port_number"],
      ["servicename","service_or_SID"],
      ["dbname","database_name"],
      ["user","db_user_name"],
      ["password","db_user_password"]
   ]
}
```

Server response:

- **HTTP Status** – the status of command. The possible codes are:
  - **200** – if no error occurs

- **422** – if error occurs

- **Body**:

```
{
  "Status":"{Valid|Invalid}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli validate database=source|target \
      dbtype=oracle \
      connectiontype=tns|ezconnect \
      tnsname=tns_alias \
      server=databaseHost port=databasePort \
      servicename=service_or_SID
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role. The possible values are:
  - *source* – to remove Capture Process objects
  - *target* – to remove Apply Process objects
- **dbtype** – Type of RDBMS. The possible value is *oracle*
- **connectiontype** – Specifies the Oracle connection method. The possible values are:
  - *tns* – Local Naming Method to be used
  - *ezconnect* – EZCONNECT to be used
- **tnsname** – Name of the TNS alias. Valid **only** if `connectiontype=tns`
- **server** – Host name or IP address of the database server. Valid **only** if `connectiontype=ezconnect`
- **port** – Database port number. Valid **only** if `connectiontype=ezconnect`
- **servicename** – Database service name or SID. Valid **only** if `connectiontype=ezconnect`
- **dbname** – Name of either container or pluggable database. Valid **only** if `database=source` and for any Oracle versions 12c-19c but excluding RDS instances
- **user** – Database user name
- **password** – Database user password

7.3.3 Validate PostgreSQL and Aurora PostgreSQL Databases

REST API:

- **Endpoint:** https://*repstance_url*/configure/database
- **Method:** POST
- **Header:**
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
```

```
    "command":"validate",
    "parameters":[
       ["database","target"],
       ["dbtype","postgresql"],
       ["server","host_name"],
       ["port","port_number"],
       ["dbname","database_name"],
       ["user","db_user_name"],
       ["password","db_user_password"]
    ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Valid|Invalid}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli validate database=target dbtype=postgresql \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role. The possible value is *target* – to install Apply Process objects
- **dbtype** – Type of RDBMS. The value is *postgresql*
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database user name
- **password** – Database user password

7.3.4 Validate MySQL Database and Aurora MySQL Databases

REST API:

- **Endpoint:** https://*repstance_url*/configure/database
- **Method:** POST
- **Header:**
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
   "command":"validate",
```

```
    "parameters":[
       ["database","target"],
       ["dbtype","mysql"],
       ["server","host_name"],
       ["port","port_number"],
       ["dbname","database_name"],
       ["user","db_user_name"],
       ["password","db_user_password"]
    ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Valid|Invalid}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli validate database=target dbtype=mysql \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName
```

The input parameters are:

- **database** – Database role. The possible value is *target* – to install Apply Process objects
- **dbtype** – Type of RDBMS. The value is *mysql*
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database user name
- **password** – Database user password

7.4 Prepare Capture Process

The "**prepare**" Capture Process command is used to add new Capture Process.

The "**prepare**" command cannot be used to validate the database connection, or the presence of the necessary data of captured objects. In order to validate that the process is configured properly and it is able to run on the Source Database, the "**validate**" command must be used (see chapter 7.7 Validate Capture Process).

The "**prepare**" command does not enable the Database to start capturing data on the configured objects, until the specified Capture Process runs (see chapter 7.15.1 Run any Capture or Apply Processes for the details).

Parameter **id** is mandatory. All others parameters are optional. All parameters except **id** can be changed using "**alter**" Capture Process command (see chapter 7.5 Alter Capture Process). The **name** parameter can be altered only in the case if it has not been provided earlier. If there is another Capture Process, which has the same **id** or the same **name** then the "**prepare**" command will fail.

7.4.1 Prepare Capture Process for SQL Server Database

REST API:

- **Endpoint:** `https://repstance_url/configure/process`
- **Method:** `POST`
- **Header**:
  - `Content-Type: application/json`
  - `X-Token: token`
- **Body**:

```
{
   "command":"prepare",
   "parameters":[
      ["process","capture"],
      ["id","captureID"],
      ["dbtype","mssql"],
      ["name","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["ddlinclude","objectsMask"],
      ["ddlexclude","objectsMask"],
      ["dmlinclude","objectsMask"],
      ["dmlexclude","objectsMask"],
      ["loadinclude","objectsMask"],
      ["loadexclude","objectsMask"],
      ["map=mapID","mappingClause"],
      ["skipapply","skipapplyMask"]
   ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare process=capture id=captureID \
      dbtype=mssql name=captureName \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName autostart={0|1} \
      debuglevel={0-15} \
      ddlinclude=objectsMask ddlexclude=objectsMask \
      dmlinclude=objectsMask dmlexclude=objectsMask \
      loadinclude=objectsMask loadexclude=objectsMask \
      map=mapID,mappingClause \
      skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only appropriate value is *capture*
- **id** – The Capture Process id
- **dbtype** – The appropriate value is *mssql*
- **name** – Name of the Capture Process
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database user name
- **password** – Database user password
- **autostart** – Determines if the Process must be run automatically. The possible values are:
  - *0* – Do not run the Process automatically (default value)
  - *1* – To run the Process automatically
- **debuglevel** – The level of debugging. Possible values are *0-15*, the default value is *0*
- **ddlinclude** – Mask of the DDL objects to be captured (see chapter 7.4.3 Capture Objects Specification for the details)
- **ddlexclude** – Mask of the DDL objects to be skipped by the Capture Process (see chapter 7.4.3 Capture Objects Specification for the details)
- **dmlinclude** – Mask of the DML objects to be captured (see chapter 7.4.3 Capture Objects Specification for the details)
- **dmlexclude** – Mask of the DML objects to be skipped by the Capture Process (see chapter 7.4.3 Capture Objects Specification for the details)
- **loadinclude** – Mask of the objects to be included into the Initial Load (see chapter 7.4.4 Initial Load for the details)
- **loadexclude** – Mask of the objects to be skipped during Initial Load (see chapter 7.4.4 Initial Load for the details)
- **map** – The set of the parameters to determine objects' transformation (see chapter 7.4.5 Objects Mapping and Possible Transforms for the details)
- **skipapply** – Used to define the behavior of a Capture Process when extracting data, which was produced by Apply Processes as part of the "loop-back" control function, in that, this Capture Process will extract or ignore the data. The Apply Processes to be skipped are specified by id*s* separated by comma. In order to

specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*all*".

7.4.2 Prepare Capture Process for Oracle Database

REST API:

- **Endpoint:** https://*repstance_url*/configure/process
- **Method:** POST
- **Header**:
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
   "command": "prepare",
   "parameters": [
      ["process","capture"],
      ["id","captureID"],
      ["dbtype","oracle"],
      ["name","captureName"],
      ["connectiontype","tns|ezconnect"],
      ["tnsname","tns_alias"],
      ["server","host_name"],
      ["port","port_number"],
      ["servicename","service_or_SID"],
      ["dbname","databaseName"],
      ["user","username"],
      ["password","password"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["ddlinclude","objectsMask"],
      ["ddlexclude","objectsMask"],
      ["dmlinclude","objectsMask"],
      ["dmlexclude","objectsMask"],
      ["loadinclude","objectsMask"],
      ["loadexclude","objectsMask"],
      ["map=mapID","mappingClause"],
      ["skipapply","skipapplyMask"]
   ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare process=capture id=captureID \
```

```
dbtype=oracle name=captureName \
connectiontype=tns|ezconnect \
tnsname=tns_alias \
server=databaseHost port=databasePort \
servicename=service_or_SID
user=username password=password \
dbname=databaseName autostart={0|1} debuglevel={0-15} \
ddlinclude=objectsMask ddlexclude=objectsMask \
dmlinclude=objectsMask dmlexclude=objectsMask \
loadinclude=objectsMask loadexclude=objectsMask \
map=mapID,mappingClause \
skipapply=skipapplyMask
```

The input parameters are:

- **`process`** – The only appropriate value is *`capture`*
- **`id`** – Capture Process id
- **`dbtype`** – Type of RDBMS. The possible value is *`oracle`*
- **`connectiontype`** – Specifies the Oracle connection method. The possible values are:
  - *`tns`* – Local Naming Method to be used
  - *`ezconnect`* – EZCONNECT to be used
- **`tnsname`** – Name of the TNS alias. Valid **only** if `connectiontype=`*`tns`*
- **`server`** – Host name or IP address of the database server. Valid **only** if `connectiontype=`*`ezconnect`*
- **`port`** – Database port number. Valid **only** if `connectiontype=`*`ezconnect`*
- **`servicename`** – Database service name or SID. Valid **only** if `connectiontype=`*`ezconnect`*
- **`dbname`** – Name of either container or pluggable database. Valid **only** if `database=`*`source`* and for any Oracle versions 12c-19c but excluding RDS instances
- **`name`** – Name of the Capture Process
- **`user`** – Database user name
- **`password`** – Database user password
- **`autostart`** – Determines if the Process must be run automatically. The possible values are:
  - *`0`* – Do not run the Process automatically (default value)
  - *`1`* – To run the Process automatically
- **`debuglevel`** – The level of debugging. Possible values are *`0-15`*, the default value is *`0`*
- **`ddlinclude`** – Mask of the DDL objects to be captured (see the 7.4.3 Capture Objects Specification chapter for the details)
- **`ddlexclude`** – Mask of the DDL objects to be skipped by the Capture Process (see the 7.4.3 Capture Objects Specification chapter for the details)
- **`dmlinclude`** – Mask of the DML objects to be captured (see the 7.4.3 Capture Objects Specification chapter for the details)
- **`dmlexclude`** – Mask of the DML objects to be skipped by the Capture Process (see the 7.4.3 Capture Objects Specification chapter for the details)

- **`loadinclude`** – Mask of the objects to be included into the Initial Load (see chapter 7.4.4 Initial Load for the details)
- **`loadexclude`** – Mask of the objects to be skipped during Initial Load (see chapter 7.4.4 Initial Load for the details)
- **`map`** – The set of the parameters to determine objects' transformation (see 7.4.5 Objects Mapping and Possible Transforms chapter for the details)
- **`skipapply`** – Used to define the behavior of a Capture Process when extracting data, which was produced by Apply Processes as part of the "loop-back" control function, in that, this Capture Process will extract or ignore the data. The Apply Processes to be skipped are specified by id*s* separated by comma. In order to specify all Processes the "*`all`*" value to be used. In order to avoid skipping any Apply Processes the "*`none`*" value to be used. The default value is "*`all`*".

7.4.3 Capture Objects Specification

Repstance allows flexibility to determine a set of tables, which are to be captured. The tables for DML and DDL operations are specified by the different parameters.

The following parameters are used for DML replication:

- `dmlinclude` – List of the tables to be included into the DML replication
- `dmlexclude` – List of the tables to be excluded from the DML replication

The following parameters are used for DDL replication:

- `ddlinclude` – List of the tables to be included into the DDL replication
- `ddlexclude` – List of the tables to be excluded from the DDL replication

Note – These criteria are mutually exclusive. The table will only be replicated in the case that the name matches the "include" criteria and does not match the "exclude" criteria.

The table is specified in *`schema_name.table_name`* format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It may also be used in both *`schema_name`* and *`table_name`* parts.

The following examples show various ways of defining tables:

- **Customers** and **employees** tables, which are in the **dbo** schema:

```
dbo.customers,dbo.employees
```

- Any tables in the **dbo** schema:

```
dbo.%
```

- Any tables having name started from **rep** or **tmp**:

```
%.rep%,%.tmp%
```

- Any table in report schema ends with **#:**

```
report.%#
```

The following examples show how to use the **DML/DDL** parameters:

- To capture DML changes for all tables in **dbo** schema except where tables start with **rep**:

```
dmlinclude=dbo.%
dmlexclude=dbo.rep%
```

- To capture DML changes for all tables in the **cap1**, **cap2** and **cap3** schemas except the **report1** table from the **cap1** schema:

```
dmlinclude=cap1.%,cap2.%,cap3.%
dmlexclude=cap1.report1
```

- To capture DML and DDL operations for **any** tables in the **dbo** schema:

```
dmlinclude=dbo.%
ddlinclude=dbo.%
```

- To capture only DDL operations for objects in the **report** schema:

```
ddlinclude=report.%
```

These parameters influence the newly created tables as well. The description below shows the capture behavior of the tables created:

Tables that match DML criteria	Tables that match DDL criteria	Capture Behavior
true	true	This create statement is captured. Any further DML statements will be captured as well
true	false	This create statement is not captured. No DML statements will be captured
false	true	This create statement is captured. DML statements are not captured. However any further DDL statements are captured
false	false	This create statement is not captured. Neither DML or DDL statements will be captured

7.4.4 Initial Load

Repstance has built-in functionality to load objects' data, and start the replication process from the timestamp at which the data has been created. Initial Loading is performing by the Capture Process at the "Run" stage (see chapter 7.15.1 Run any Capture or Apply Processes for the details).

This functionality is used when it is necessary to synchronize data between the Source and Target Databases before starting the replication process. The Initial Load parameters can be configured to either clean up, or preserve data in the Target Database before loading from the Source Database. Two further options are to create table in the Target Database if it doesn't exist and recreate this table in the Target Database if it does exist.

The following parameters are used to configure the objects for Initial Loading:

- `loadinclude` – List of the tables to be included into the Initial Load
- `loadexclude` – List of the tables to be excluded from the Initial Load

---

Note – These parameters MUST be used in conjunction with the *`dmlinclude`*/ *`dmlexclude`* object definitions as well, otherwise the Capture Process will skip them.

The format to be specified takes the following form:

```
schema_name1.table_name1:[loadOption1],
schema_name2.table_name2:[loadOption2], ...
```

In order to define the list of the tables to be part of this process, and where there are more than one tables mask to be defined each of these must be separated by comma. The `%` symbol may be used in order to match any number of characters. It may also be used in both *`schema_name`* and *`table_name`* parts.

The *`loadOption`* is an optional parameter, and it used to perform the action detailed below on the data before insertion into the Target Database, and can have the following values:

- `A` – Preserve the data which is the **DEFAULT** value
- `T` – Use **truncate** statement to clean up the data
- `D` – Use **delete** statement to clean up the data
- `C` – Create table if it doesn't exist
- `R` – Recreate table if it is already exist

Note – Once the Initial Load has been completed by Capture Process the values of *`loadinclude`* and *`loadexclude`* parameters will be removed automatically. The details of Initial Load criteria can be found in the Capture log files.

The following examples show various ways of defining the Initial Loading parameters:

- To reload data for the **emp** and **emp_audit_trail** tables, which are in the **dbo** schema and **truncate** these tables before insertion:

  ```
  loadinclude=dbo.emp:T,dbo.emp_audit_trail:T
  ```

- Any data from tables starting with the rep_ prefix in the dbo schema which need to be **deleted** before insertion:

```
loadinclude=dbo.rep_%:D
```

- To reload all tables started with emp prefix except the employee_archive tables:

```
loadinclude=dbo.emp%:T
loadexclude=dbo.employee_archive
```

- To reload **customer** table and create it if it doesn't exist in the Target Database:

```
loadinclude=dbo.customer:C
```

7.4.5 Objects Mapping and Possible Transforms

Transforms are primarily used where there is a need to reformat any statements. The transformation can only be configured for a Capture Process, as the Apply function is inherent in the use of transformation. Therefore it is not possible nor is it necessary to construct an Apply Process for a Transform.

Repstance supports transformation rules, which can be applied to the following objects:

- schema name
- table name
- column name
- "identity" attribute (valid only for SQL Server DB)
- "primary key" attribute
- data type
- data type length
- data

Transformation rules can be triggered based on:

- schema name mask
- table name mask
- column name mask
- "identity" attribute (valid only for SQL Server DB)
- reference "identity" column (valid only for SQL Server DB)
- "primary key" attribute
- data type mask
- data type length

Transformation rules can be used to exclude the following objects:

- table
- column
- "identity" attribute (valid only for SQL Server DB)
- reference "identity" column (valid only for SQL Server DB)
- "primary key" attribute

Transformation rules can be used to add the following objects:

- column
- "identity" attribute (valid only for SQL Server DB)

- “primary key” attribute

Transformation rules can be used to change the following objects:

- schema name
- table name
- column name
- “identity” attribute (valid only for SQL Server DB)
- “primary key” attribute
- data type
- data type length
- data

Note – Data can be changed to the :

- constant value
- original value of another column from **this** table
- result of SQL Server function execution. The original values of any columns from this table can be passed to the Function as parameters. **The Function must exist in the Target Database, where the Apply Process is to be run.**

There are normally three separate parts that comprise a Transformation. They are:

- **`id`** – The id, which is unique to each Transform has the purpose of determining the order in which the Transform is executed by the Process
- **`rule`** – Specifies the object to be transformed and specifies how it is to be transformed
- **`description`** – A meaningful description of this Transformation

The syntax used is:

```
map=id,rule=(CaptureCriteria:TransformCriteria),description="desc"
```

where:

```
CaptureCriteria=schema_mask.table_mask.column_mask.[attr_spec]
```

```
attr_spec=PK=PK;TYPE=type;ID=id;LEN=len
```

The `CaptureCriteria` consists of:

- `schema_mask` – the mask is used to specify schema name
- `table_mask` – the mask is used to specify table name
- `column_mask` – the mask is used to specify column name

The `attr_spec` is:

- `PK` – The specification of PK attribute. It accepts the following values:
  - *`N`* – to determine column without PK
  - *`U`* – to include PK column
- `ID` – Determine if the column should or should not be handled as an “identity_insert”. The possible values are:
  - *`0`* – to process as “non identity_insert” column

- *1* – to process as “identity_insert” column
- *2* – to process as reference “identity” column

- TYPE – The name of the data type
- LEN – Length of data type if applicable

TransformCriteria=*schema_mask.table_mask.column_mask.*[*attr_spec*]

attr_spec=PK=*PK*;TYPE=*type*;ID=*id*;LEN=*len*;DATA="*data_specification*"

data_specification=DATA="*SQL_function_or_operation*"

The TransformCriteria consists of:

- schema_mask – the mask is used to specify schema name
- table_mask – the mask is used to specify table name
- column_mask – the mask is used to specify column name

The attr_spec is:

- PK – The specification of PK attribute. It accepts the following values:
  - *N* – To determine column without PK
  - *U* – To include PK column
- ID – Determine if the column should or should not be handled as an “identity_insert”. The possible values are:
  - *0* – To process as “non identity_insert” column
  - *1* – To process as “identity_insert” column
  - *2* – To process as reference “identity” column
- TYPE – The name of the data type
- LEN – Length of data type if applicable
- DATA – This is the data that will result from the transformation specified on the captured data. This can be:
  - Predefined value. The format is "*predefined_value*"
  - Any operations based on the values of this column. In order to use the values found in the column the % needs to be used
  - Any operations based on the values of any columns in this table. In order to use the values of any column the following format is used "C*:number_of_column*". Table name can be passed as parameter as well. The format is "*T:0*".

To **exclude columns** the following syntax is used:

*schema_mask.table_mask.*NULL

where:

- schema_mask – the mask is used to specify schema name
- table_mask – the mask is used to specify table name

To **include any columns that were not represented in the source table** the following syntax is used:

schema_mask.table_mask.column_name.[TYPE=*type*;LEN=*len*;

```
PK=pk;ID=id;DATA="data_specification"]
```

where:

- `schema_mask` – The mask is used to specify schema name
- `table_mask` – The mask is used to specify table name
- `column_name` – Column name (this must be included and can not use the % symbol)
- `TYPE` – Data type definition of the column to be created (this must be included and can not use the % symbol)
- `LEN` – Length of data type, where necessary must be specified here
- `PK` – The specification of PK attribute. It accepts the following values:
  - *`N`* – to determine column without PK
  - *`U`* – to include PK column
- `ID` – Determine if the column should or should not be handled as an "identity_insert". The possible values are:
  - *`0`* – to process as "non identity_insert" column
  - *`1`* – to process as "identity_insert" column
- `DATA` – This is the data that will result from the transformation specified on the captured data. This can be:
  - Predefined value. The format is "*`predefined_value`*"
  - Any operations based on the values of this column. In order to use the values found in the column the % needs to be used
  - Any operations based on the values of any columns in this table. In order to use the values of any column the following format is used "`C:`*`number_of_column`*". Table name can be passed as parameter as well. The format is "*`T:0`*".

The following examples show how to use the Transformation Rules:

- For all tables located in the **dbo** schema change the schema name to **report**:

```
map=1,rule=(dbo.%:report.%),description="sample 1"
```

- Rename **dbo.emp** table to **report.employees** table:

```
map=2,rule=(dbo.emp:report.employees),description="sample 2"
```

- Exclude **info** column in the **dbo.customer** table from the replication:

```
map=3,rule=(dbo.customer.info:dbo.customer.NULL),description="sample 3"
```

- For any tables in the **dbo** schema having name starting with **rep_** replace the prefix to the **report_**:

```
map=4,rule=(dbo.rep_%:dbo.report_%),description="sample 4"
```

- For all tables in the **dbo** schema having **ident** column change the column name to **id**:

```
map=5,rule=(dbo.%.ident:dbo.%.id),description="sample 5"
```

- Exclude from the replication any columns having name **id** and defined as primary key:

```
map=6,rule=(dbo.%.id.[PK=U]:dbo.%.id.NULL),description="sample 6"
```

- For any tables having column **id** of **int** type handle this column as primary key. Any update and delete statements will be built to use **id** column as primary key or the part of composite primary key:

```
map=7,rule=(dbo.%.id.[TYPE=int]:dbo.%.id.[PK=P]),description="sample 7"
```

- Put the value of the **desc** column in the **branch** table into the uppercase:

```
map=8,rule=(dbo.branch.desc:dbo.branch.desc.[DATA=\"upper(%)\"])
```

- Add new column named **tag** of the **nvarchar(100)** type into the **customer** table contains predefined value **rep**:

```
map=9,rule=(dbo.customer.NULL:dbo.customer.tag.[TYPE=nvarchar;
                    LEN=100;DATA=\"'rep'\"])
```

- Add new column named **name** of the **nvarchar(255)** type to the **person** table contains concatenation of the **firstname** and **lastname** columns (in the table firstname has the position 2 and lastname has the position 3):

```
map=10,rule=(dbo.person.NULL:dbo.person.name.[TYPE=nvarchar;
                    LEN=255;DATA=\"C:2+C:3\"])
```

- For any tables in the **dbo** schema having name starting with **rep_** and column **id**, which is defined as primary key and **identity_insert**, rename the table into the **report** schema and without **rep_** prefix and use custom function **report.generateID** to reproduce the value of this column. This column will still be used as a primary key, but not as **identity_insert**:

```
map=11,rule=(dbo.rep_%.id.[PK=U;ID=1]:report.%.id.[PK=U;ID=0;
                    DATA=\"report.generateID()\"])
```

7.4.6 Transformation Rules and Triggering Order

It is imperative that you understand that the order in which the rules are applied can significantly impact on the expected results. The following logic is used to handle rule order processing. The specified Capture Process will go and look for any possible data that meets the first of the configured rules, to apply. In the case where there is more than one rule to be processed, the rule **which will** "take precedence" will be the rule which best fulfills the data found. If the objects found match the criteria given in the **first rule** to be processed any subsequent rules will be ignored because the previous criteria have been met.

The logic used in transformation does not necessarily follow the rules for human logic, in that, the least significant amount of data to be processed should be the **first rule**, and the rule which affects the largest amount of data should be the **last rule**. In this way the user will be able to apply the requirements of the transform in the desired way.

The following example demonstrates this program logic. Assume that you need to rename all tables in the **dbo** schema having a name starting with **emp** into the schema **hr** and keep the original table name. Another requirement is to rename the **ident** columns to be the **id** for any tables in the **dbo** schema, in the case where we have an **employee** table that contains the **ident** column, this matches both rules. The examples demonstrate the replication behavior depending on the order that the rules are presented:

- In this case you can't expect the **ident** column to be renamed into **id** column together with renaming **dbo** schema in **hr** schema. The **dbo** schema for **employee** table will be changed to **hr** schema but the **ident** column will not be changed:

```
map=1,rule=(dbo.emp%:hr.emp%),description="sample 1"
map=2,rule=(dbo.%.ident:dbo.%.id),description="sample 2"
```

- In this case the **dbo** schema will not be changed into the **hr** schema with renaming the **ident** column to the **id** column. The **dbo** schema for **employee** table will not be changed but the **ident** column will be changed to **id**:

```
map=1,rule=(dbo.%.ident:dbo.%.id),description="sample 3"
map=2,rule=(dbo.emp%:hr.emp%),description="sample 4"
```

- In this case the **dbo** schema for **employee** table will be changed and the **ident** column will be changed to **id**:

```
map=1,rule=(dbo.emp%.ident:hr.emp%.id)
map=2,rule=(dbo.emp%:hr.emp%)
map=3,rule=(dbo.%.ident:dbo.%.id)
```

7.5 Alter Capture Process

The "**alter**" Capture Process command is used to change the Capture Process parameters. The **`id`** parameter can not be changed. The **`name`** parameter can be changed only in the case that it has not been provided before - eg. it must be "Unique".

The Capture Process has to exist otherwise the command will fail.

Note – You can alter the Capture Process even if it is running, however these changes will only be implemented after the Capture Process has been stopped and rerun.

7.5.1 Alter Capture Process for SQL Server Database

REST API:

- **Endpoint:** `https://repstance_url/configure/process`
- **Method:** `POST`
- **Header:**
  - `Content-Type: application/json`

    - X-Token: *token*
- **Body:**

```
{
   "command": "alter",
   "parameters": [
      ["process","capture"],
      ["id","captureID"],
      ["dbtype","mssql"],
      ["name","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["ddlinclude","objectsMask"],
      ["ddlexclude","objectsMask"],
      ["dmlinclude","objectsMask"],
      ["dmlexclude","objectsMask"],
      ["loadinclude","objectsMask"],
      ["loadexclude","objectsMask"],
      ["map=mapID","mappingClause"],
      ["skipapply","skipapplyMask"]
   ]
}
```

Server response:

- **HTTP Status** – The status of the command. The possible codes are:
    - **200** – If no error occurs
    - **422** – If error occurs
- **Body:**

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli alter process=capture id=captureID dbtype=mssql \
      name=captureName \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName autostart={0|1} \
      debuglevel={0-15} \
      ddlinclude=objectsMask ddlexclude=objectsMask \
      dmlinclude=objectsMask dmlexclude=objectsMask \
      map=mapID,mappingClause skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only appropriate value is *capture*
- **id** – The Capture Process identifier
- **dbtype** – Type of RDBMS. The possible value is *mssql*
- **name** – Name of the Capture Process

- **`server`** – Host name or IP address of the database server
- **`port`** – Database port number
- **`dbname`** – Database name
- **`user`** – Database user name
- **`password`** – Database user password
- **`autostart`** – Determines if the Capture Process must be run automatically. Possible values are:
  - *`0`* – Do not run the Capture Process automatically (default value)
  - *`1`* – To run the Capture Process automatically
- **`debuglevel`** – The level of debugging. Possible values are: *`0-15`*
- **`ddlinclude`** – Mask of the DDL objects to be captured (see chapter 7.4.3 Capture Objects Specification for the details)
- **`ddlexclude`** – Mask of the DDL objects to be skipped by the Capture Process (see chapter 7.4.3 Capture Objects Specification for the details)
- **`dmlinclude`** – Mask of the DML objects to be captured (see chapter 7.4.3 Capture Objects Specification for the details)
- **`dmlexclude`** – Mask of the DML objects to be skipped by the Capture Process (see chapter 7.4.3 Capture Objects Specification for the details)
- **`loadinclude`** – Mask of the objects to be included into the Initial Load (see chapter 7.4.4 Initial Load for the details)
- **`loadexclude`** – Mask of the objects to be skipped during Initial Load (see chapter 7.4.4 Initial Load for the details)
- **`map`** – The set of the parameters to determine objects' transformation (see chapter 7.4.5 Objects Mapping and Possible Transforms for the details)
- **`skipapply`** – Used to define the behavior of a Capture Process when extracting data, which was produced by Apply Processes as part of the "loop-back" control function, in that, this Capture Process will extract or ignore the data. The Apply Processes to be skipped are specified by ids separated by comma. In order to specify all Processes the "*`all`*" value to be used. In order to avoid skipping any Apply Processes the "*`none`*" value to be used. The default value is "*`all`*".

7.5.2 Alter Capture Process for Oracle Database

REST API:

- **Endpoint:** `https://repstance_url/configure/process`
- **Method:** `POST`
- **Header:**
  - `Content-Type: application/json`
  - `X-Token: token`
- **Body:**

```
{
   "command": "alter",
   "parameters": [
      ["process","capture"],
      ["id","captureID"],
      ["dbtype","oracle"],
```

```
        ["name","captureName"],
        ["connectiontype","tns|ezconnect"],
        ["tnsname","tns_alias"],
        ["server","host_name"],
        ["port","port_number"],
        ["servicename","service_or_SID"],
        ["dbname","databaseName"],
        ["autostart","{0|1}"],
        ["debuglevel","{0-15}"],
        ["ddlinclude","objectsMask"],
        ["ddlexclude","objectsMask"],
        ["dmlinclude","objectsMask"],
        ["dmlexclude","objectsMask"],
        ["loadinclude","objectsMask"],
        ["loadexclude","objectsMask"],
        ["map=mapID","mappingClause"],
        ["skipapply","skipapplyMask"]
    ]
}
```

Server response:

- **HTTP Status** – The status of the command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body:**

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli alter process=capture id=captureID \
      dbtype=oracle name=captureName \
      connectiontype=tns|ezconnect \
      tnsname=tns_alias \
      server=databaseHost port=databasePort \
      servicename=service_or_SID
      user=username password=password \
      dbname=databaseName autostart={0|1} debuglevel={0-15} \
      ddlinclude=objectsMask ddlexclude=objectsMask \
      dmlinclude=objectsMask dmlexclude=objectsMask \
      loadinclude=objectsMask loadexclude=objectsMask \
      map=mapID,mappingClause skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only possible value is *capture*
- **id** – Capture Process id
- **dbtype** – Type of RDBMS. The possible value is *oracle*
- **connectiontype** – Specifies the Oracle connection method. The possible values are:
  - *tns* – Local Naming Method to be used
  - *ezconnect* – EZCONNECT to be used

- **`tnsname`** – Name of the TNS alias. Valid **only** if `connectiontype=tns`
- **`server`** – Host name or IP address of the database server. Valid **only** if `connectiontype=ezconnect`
- **`port`** – Database port number. Valid **only** if `connectiontype=ezconnect`
- **`servicename`** – Database service name or SID. Valid **only** if `connectiontype=ezconnect`
- **`dbname`** – Name of either container or pluggable database. Valid **only** if `database=source` and for any Oracle versions 12c-19c but excluding RDS instances
- **`name`** – Name of the Capture Process
- **`user`** – Database user name
- **`password`** – Database user password
- **`autostart`** – Determines if the Capture Process must be run automatically. The possible values are:
  - *0* – Do not run the Capture Process automatically (default value)
  - *1* – To run the Capture Process automatically
- **`debuglevel`** – The level of debugging. Possible values are *0-15*
- **`ddlinclude`** – Mask of the DDL objects to be captured (see 7.4.3 Capture Objects Specification chapter for the details)
- **`ddlexclude`** – Mask of the DDL objects to be skipped by the Capture Process (see 7.4.3 Capture Objects Specification chapter for the details)
- **`dmlinclude`** – Mask of the DML objects to be captured (see 7.4.3 Capture Objects Specification chapter for the details)
- **`dmlexclude`** – Mask of the DML objects to be skipped by the Capture Process (see 7.4.3 Capture Objects Specification chapter for the details)
- **`loadinclude`** – Mask of the objects to be included into the Initial Load (see chapter 7.4.4 Initial Load for the details)
- **`loadexclude`** – Mask of the objects to be skipped during Initial Load (see chapter 7.4.4 Initial Load for the details)
- **`map`** – The set of the parameters to determine objects' transformation (see chapter 7.4.5 Objects Mapping and Possible Transforms for the details)
- **`skipapply`** – Used to define the behavior of a Capture Process when extracting data, which was produced by Apply Processes as part of the "loop-back" control function, in that, this Capture Process will extract or ignore the data. The Apply Processes to be skipped are specified by ids separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*all*".

7.6 Show Capture Process

The "**show**" Capture command is used to show a Capture Process configuration. This command will show the latest "Set" of configured processes, regardless of whether or not the Capture Process has been "stopped and re-applied", this will not necessarily be the configuration of the currently running Capture Process.

Note – If the configuration has been changed when the Capture Process is running, the "**show**" command will still display the currently configured process, not the "running" one.

7.6.1 Show Capture Process for SQL Server Database

REST API:

- **Endpoint**: `https://repstance_url/configure/process`
- **Method**: `POST`
- **Header**:
  - `Content-Type: application/json`
  - `X-Token: token`
- **Body**:

```
{
   "command": "show",
   "parameters": [
      ["process","capture"],
      ["id","captureID"]
   ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details",
  "parameters":[
      ["dbtype","mssql"],
      ["name","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["ddlinclude","objectsMask"],
      ["ddlexclude","objectsMask"],
      ["loadinclude","objectsMask"],
      ["loadexclude","objectsMask"],
      ["dmlinclude","objectsMask"],
      ["dmlexclude","objectsMask"],
      ["map=mapID","mappingClause"]
   ]
}
```

CLI Syntax:

```
repcli show process=capture id=captureID
```

The input parameters are:

- **`process`** – The only possible value is *`capture`*
- **`id`** – The Capture Process identifier

Note – Only if the command completes successfully will the response contain all the "parameter" values and at the same time this information will be displayed in Plain Text in the "Message" display.

7.6.2 Show Capture Process for Oracle Database

Note – If the configuration has been changed when the Capture Process is running, the "Show" command will still display the currently configured process, not the "running" one.

REST API:

- **Endpoint**: `https://repstance_url/configure/process`
- **Method**: `POST`
- **Header**:
  - `Content-Type: application/json`
  - `X-Token: token`
- **Body**:

```
{
   "command": "show",
   "parameters": [
      ["process","capture"],
      ["id","captureID"]
   ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **`200`** – If no error occurs
  - **`422`** – If error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details",
  "parameters":[
      ["id","captureID"],
      ["dbtype","oracle"],
      ["name","captureName"],
      ["connectiontype","tns|ezconnect"],
      ["tnsname","tns_alias"],
      ["server","host_name"],
      ["port","port_number"],
      ["servicename","service_or_SID"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
```

```
        ["ddlinclude","objectsMask"],
        ["ddlexclude","objectsMask"],
        ["dmlinclude","objectsMask"],
        ["dmlexclude","objectsMask"],
        ["loadinclude","objectsMask"],
        ["loadexclude","objectsMask"],
        ["map=mapID","mappingClause"]
    ]
}
```

CLI Syntax:

```
repcli show process=capture id=captureID
```

The input parameters are:

- **process** – The value is *capture*. This is a "constant" value and **can only be** *capture*
- **id** – The Capture Process identifier

Note – Only if the command completes successfully will the response contain all the "parameter" values and at the same time this information will be displayed in Plain Text in the "Message" display.

7.7 Validate Capture Process

The "**validate**" Capture Process command is used to perform a set background checks to ensure that all the necessary parameters have been supplied to enable the functionality of a Capture configuration.

REST API:

- **Endpoint:** https://*repstance_url*/configure/process
- **Method:** POST
- **Header:**
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
    "command": "validate",
    "parameters": [
        ["process","capture"],
        ["id","captureID"]
    ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
```

```
  "Status":"{Valid|Invalid}",
  "Message":"The command execution details",
}
```

CLI Syntax:

```
repcli validate process=capture id=captureID
```

The input parameters are:

- **process** – The only possible value is *capture*
- **id** – The Capture Process identifier

In case the **Status** is *Invalid* the **Message** will contain detailed information about the validation error.

7.8 Remove Capture Process

The "**remove**" Capture Process command is used to remove **all** of the Capture Process configuration files, Trail Files and the checkpoint information.

For SQL Server Database if the Source Database **is** reachable the "**remove**" command will disable CDC on the tables, which are configured to be captured by this Process only.

The command can only be executed if the specified Capture Process is not running.

It is strongly recommended that you backup all trail and configuration files before running this command (see chapter 8.4 Backup Repstance Files).

Note – After removing the Capture Process it is not possible to reverse this action.

Any Apply Processes using data from this Capture will cease to work.

This command cannot be applied to a group of "Configures" at the same time, it must be used individually for each of the configured Capture Processes.

REST API:

- **Endpoint:** https://*repstance_url*/configure/process
- **Method:** POST
- **Header:**
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
   "command": "remove",
   "parameters": [
      ["process","capture"],
      ["id","captureID"]
   ]
```

```
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Completed|Error|Warning}",
  "Message":"The command execution details",
}
```

CLI Syntax:

```
repcli remove process=capture id=captureID
```

The input parameters are:

- **process** – The only possible value is *capture*
- **id** – The identifier of the Capture Process

The output parameters are:

- **Status** – Status of the command. The possible values are:
  - *Error* – The command has failed to run
  - *Warning* – The process has failed to remove all the Capture objects
  - *Completed* – The command has completed successfully
- **Message** – The details of command execution

The following example demonstrates how to remove Capture 2:

```
repcli remove process=capture id=2
```

7.9 Prepare Apply Process

The "**prepare**" Apply command is used to add a new Apply Process. The "**prepare**" command can not be used to validate the database connection, and availability of the Trail Files or the presence of the necessary database objects to be used by the Apply Process.

In order to validate that the Apply Process is configured properly and it is able to run on the Target Database, the "**validate**" command must be used (see chapter 7.12. Validate Apply Process).

The "**prepare**" command does not enable the Apply Process to write data into the Target Database until the specified Apply Process is run (see chapter 7.15.1 Run any Capture or Apply Processes).

Parameter **id** is mandatory. All others parameters are optional. All parameters except **id** can be changed using "**alter**" Apply Process command (see chapter 7.10 Alter Apply Process for the details).

7.9.1 Prepare Apply Process for SQL Server Database

REST API:

- **Endpoint:** https://*repstance_url*/configure/process
- **Method:** POST
- **Header:**
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
   "command": "prepare",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","mssql"],
      ["capturename","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["repuser","0|1"],
      ["ddlcreate","objectMask:options"],
      ["ddldrop","objectMask:options"],
      ["ddlprocessing","native|dictionary"],
      ["skipapply","skipapplyMask"]
   ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare process=apply id=applyID dbtype=mssql \
      capturename=captureName \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName autostart={0|1} \
      debuglevel={0-15} repuser={0|1} \
      ddlcreate=objectMask:options \
      ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
```

```
skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process id
- **dbtype** – Type of RDBMS. The possible value is *mssql*
- **capturename** – The name of the Capture Process which is providing the data to be used
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database user name
- **password** – Database user password
- **autostart** – Determines if the Apply Process must be run automatically. Possible values are:
  - *0* – Do not run the Apply Process automatically (default value)
  - *1* – To run the Apply Process automatically
- **debuglevel** – The level of debugging. Possible values are *0-15*
- **repuser** – Determines if the Apply Process must represent itself as SQL Server replication process. The possible values are:
  - *0* – Do not represent itself as SQL Server replication process (default value)
  - *1* – To represent itself as SQL Server replication process
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The % symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – Do not run DDL create command if the table already exists
  - *recreate* – To drop the existing table and recreate it
- **ddldrop** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The % symbol may be used in order to match any number of characters. It has the following option:
  - *skip* – Do not run DDL drop command if the table doesn't exist
- **ddlprocessing** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - *native* – To process DDL based on the user statement (default value)
  - *dictionary* – To generate DDL based on the dictionary changes
- **skipapply** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by ids separated by comma. In order to specify all Processes the "*all*" value to be used.

In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*none*".

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

You should be aware that if the **repuser** parameter is set to "1" and is invoked, then when delivering data to a SQL Server Target Database, any changes of objects marked as 'Not for Replication' (Identity columns, table triggers, foreign keys and check constraints) will be handled in exactly the same way, as they would be produced by an SQL Server replication agent. As an example, any triggers which are created with the "Not for Replication" option will never be fired.

7.9.2 Prepare Apply Process for Oracle Database

REST API:

- **Endpoint:** https://*repstance_url*/configure/process
- **Method:** POST
- **Header:**
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
   "command": "prepare",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","oracle"],
      ["capturename","captureName"],
      ["connectiontype","tns|ezconnect"],
      ["tnsname","tns_alias"],
      ["server","host_name"],
      ["port","port_number"],
      ["servicename","service_or_SID"],
      ["dbname","databaseName"],
      ["user","username"],
      ["password","password"],
      ["autostart","{0|1}"],
      ["ddlcreate","objectMask:options"],
      ["ddldrop","objectMask:options"],
      ["ddlprocessing","native|dictionary"],
      ["debuglevel","{0-15}"],
      ["skipapply","skipapplyMask"]
   ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs

- **422** – If error occurs

- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare process=apply id=applyID dbtype=oracle \
      capturename=captureName \
      server=databaseHost port=databasePort \
      user=username password=password \
      autostart={0|1} \
      ddlcreate=objectMask:options ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
      debuglevel={0-15} skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only appropriate value is *apply*
- **id** – The Apply Process id
- **dbtype** – Type of RDBMS. The possible value is *oracle*
- **connectiontype** – Specifies the Oracle connection method. The possible values are:
  - *tns* – Local Naming Method to be used
  - *ezconnect* – EZCONNECT to be used
- **tnsname** – Name of the TNS alias. Valid **only** if connectiontype=*tns*
- **server** – Host name or IP address of the database server. Valid **only** if connectiontype=*ezconnect*
- **port** – Database port number. Valid **only** if connectiontype=*ezconnect*
- **servicename** – Database service name or SID. Valid **only** if connectiontype=*ezconnect*
- **capturename** – The name of the Capture Process which is providing the data to be used
- **user** – Database user name
- **password** – Database user password
- **autostart** – Determines if the Apply Process must be run automatically. Possible values are:
  - *0* – Do not run the Apply Process automatically
  - *1* – To run the Apply Process automatically
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The % symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – Do not run DDL create command if the table already exists
  - *recreate* – Drops the existing table and recreates it
- **ddldrop** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in

schema_name.table_name format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:

- *skip* – Do not run DDL drop command if the table doesn't exist

- **debuglevel** – The level of debugging. Possible values are *0-15*
- **ddlprocessing** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - *native* – To process DDL based on the user statement (default value)
  - *dictionary* – To generate DDL based on the dictionary changes
- **skipapply** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by ids separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*none*".

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.9.3 Prepare Apply Process for PostgreSQL and Aurora PostgreSQL Databases

REST API:

- **Endpoint:** https://*repstance_url*/configure/process
- **Method:** POST
- **Header:**
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
   "command": "prepare",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","postgresql"],
      ["capturename","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["ddlcreate","objectMask:options"],
      ["ddldrop","objectMask:options"],
      ["ddlprocessing","native|dictionary"],
      ["skipapply","skipapplyMask"]
   ]
```

```
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare process=apply id=applyID dbtype=postresql \
      capturename=captureName \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName autostart={0|1} \
      debuglevel={0-15} \
      ddlcreate=objectMask:options \
      ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
      skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process id
- **dbtype** – Type of RDBMS. The possible value is *postgresql*
- **capturename** – The name of the Capture Process which is providing the data to be used
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database user name
- **password** – Database user password
- **autostart** – Determines if the Apply Process must be run automatically. Possible values are:
  - *0* – Do not run the Apply Process automatically (default value)
  - *1* – To run the Apply Process automatically
- **debuglevel** – The level of debugging. Possible values are *0-15*
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The **%** symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – Do not run DDL create command if the table already exists
  - *recreate* – Drops the existing table and recreates it

- **`ddldrop`** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in *`schema_name.table_name`* format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:
  - *`skip`* – Do not run DDL drop command if the table doesn't exist
- **`ddlprocessing`** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - *`native`* – To process DDL based on the user statement (default value)
  - *`dictionary`* – To generate DDL based on the dictionary changes
- **`skipapply`** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by ids separated by comma. In order to specify all Processes the "*`all`*" value to be used. In order to avoid skipping any Apply Processes the "*`none`*" value to be used. The default value is "*`none`*".

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.9.4 Prepare Apply Process for MySQL and Aurora MySQL Databases

REST API:

- **Endpoint:** https://*repstance_url*/configure/process
- **Method:** `POST`
- **Header:**
  - `Content-Type: application/json`
  - `X-Token: token`
- **Body**:

```
{
   "command": "prepare",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","mysql"],
      ["capturename","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["ddlcreate","objectMask:options"],
      ["ddldrop","objectMask:options"],
      ["ddlprocessing","native|dictionary"],
      ["skipapply","skipapplyMask"]
```

```
    ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli prepare process=apply id=applyID dbtype=mysql \
      capturename=captureName \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName autostart={0|1} \
      debuglevel={0-15} \
      ddlcreate=objectMask:options \
      ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
      skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process identifier
- **dbtype** – Type of RDBMS. The possible value is *mysql*
- **capturename** – The name of the Capture Process which is providing the data to be used
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database user name
- **password** – Database user password
- **autostart** – Determines if the Apply Process must be run automatically. Possible values are:
  - *0* – Do not run the Apply Process automatically (default value)
  - *1* – To run the Apply Process automatically
- **debuglevel** – The level of debugging. Possible values are *0-15*
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The **%** symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – Do not run DDL create command if the table already exists
  - *recreate* – Drops the existing table and recreates it

- **`ddldrop`** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in *`schema_name.table_name`* format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:
  - *`skip`* – Do not run DDL drop command if the table doesn't exist
- **`ddlprocessing`** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - *`native`* – To process DDL based on the user statement (default value)
  - *`dictionary`* – To generate DDL based on the dictionary changes
- **`skipapply`** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by ids separated by comma. In order to specify all Processes the "*`all`*" value to be used. In order to avoid skipping any Apply Processes the "*`none`*" value to be used. The default value is "*`none`*".

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.10 Alter Apply Process

The "**alter**" Apply command is used to change the Apply Process parameters. The **`id`** parameter can not be changed. The Apply Process has to exist otherwise the command will fail. The **`capturename`** parameter can be altered only in the case that it has not been used earlier.

Note – You can alter the Apply Process even if it is running, however **these changes will only be implemented after this Apply Process has been stopped and rerun**.

7.10.1 Alter Apply Process for SQL Server Database

REST API:

- **Endpoint:** `https://repstance_url/configure/process`
- **Method:** `POST`
- **Header:**
  - `Content-Type: application/json`
  - `X-Token: token`
- **Body:**

```
{
   "command": "alter",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
```

```
        ["dbtype","mssql"],
        ["capturename","captureName"],
        ["server","databaseHost"],
        ["port","databasePort"],
        ["user","username"],
        ["password","password"],
        ["dbname","databaseName"],
        ["autostart","{0|1}"],
        ["debuglevel","{0-15}"],
        ["repuser","0|1"],
        ["ddlcreate","objectMask:options"],
        ["ddldrop","objectMask:options"],
        ["ddlprocessing","native|dictionary"],
        ["skipapply","skipapplyMask"]
    ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli alter process=apply id=applyID \
      dbtype=mssql \
      capturename=captureName \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName autostart={0|1} \
      debuglevel={0-15} repuser={0|1} \
      ddlcreate=objectMask:options \
      ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
      skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process identifier
- **dbtype** – Type of RDBMS. The possible value is *mssql*
- **capturename** – The name of the Capture Process which is providing the data to be used
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database user name
- **password** – Database user password

- **autostart** – Determines if the Apply Process must be run automatically. Possible values are:
  - *0* – Do not run the Apply Process automatically (default value)
  - *1* – To run the Apply Process automatically
- **debuglevel** – The level of debugging. Possible values are *0-15*
- **repuser** – Determines if the Apply Process must represent itself as SQL Server replication process. The possible values are:
  - *0* – Do not represent itself as SQL Server replication process (default value)
  - *1* – To represent itself as SQL Server replication process
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – Do not run DDL create command if the table already exists
  - *recreate* – Drops the existing table and recreates it
- **ddldrop** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – Do not run DDL drop command if the table doesn't exist
- **ddlprocessing** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - *native* – To process DDL based on the user statement (default value)
  - *dictionary* – To generate DDL based on the dictionary changes
- **skipapply** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by ids separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*none*".

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.10.2 Alter Apply Process for Oracle Database

REST API:

- **Endpoint:** https://*repstance_url*/configure/process
- **Method:** POST
- **Header:**

    - Content-Type: application/json
    - X-Token: *token*
- **Body:**

```
{
   "command": "alter",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","oracle"],
      ["capturename","captureName"],
      ["connectiontype","tns|ezconnect"],
      ["tnsname","tns_alias"],
      ["server","host_name"],
      ["port","port_number"],
      ["servicename","service_or_SID"],
      ["dbname","databaseName"],
      ["user","username"],
      ["password","password"],
      ["autostart","{0|1}"],
      ["ddlcreate","objectMask:options"],
      ["ddldrop","objectMask:options"],
      ["debuglevel", "{0-15}"],
      ["ddlprocessing","native|dictionary"],
      ["skipapply","skipapplyMask"]
   ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
    - **200** – If no error occurs
    - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli alter process=apply id=applyID dbtype=oracle \
      capturename=captureName \
      server=databaseHost port=databasePort \
      user=username password=password \
      autostart={0|1} \
      ddlcreate=objectMask:options ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
      debuglevel={0-15} skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process identifier
- **dbtype** – Type of RDBMS. The possible value is *oracle*
- **connectiontype** – Specifies the Oracle connection method. The possible values are:

- *tns* – Local Naming Method to be used
- *ezconnect* – EZCONNECT to be used

- **tnsname** – Name of the TNS alias. Valid **only** if `connectiontype=tns`
- **server** – Host name or IP address of the database server. Valid **only** if `connectiontype=ezconnect`
- **port** – Database port number. Valid **only** if `connectiontype=ezconnect`
- **servicename** – Database service name or SID. Valid **only** if `connectiontype=ezconnect`
- **capturename** – The name of the Capture Process which is providing the data to be used
- **user** – Database user name
- **password** – Database user password
- **autostart** – Determines if the Apply Process must be run automatically. Possible values are:
  - *0* – Do not run the Apply Process automatically (default value)
  - *1* – To run the Apply Process automatically
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – Do not run DDL create command if the table already exists
  - *recreate* – Drops the existing table and recreates it
- **ddldrop** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – Do not run DDL drop command if the table doesn't exist
- **debuglevel** – The level of debugging. Possible values are *0-15*
- **ddlprocessing** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - *native* – To process DDL based on the user statement (default value)
  - *dictionary* – To generate DDL based on the dictionary changes
- **skipapply** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by ids separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*none*".

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.10.3 Alter Apply Process for PostgreSQL and Aurora PostgreSQL Databases

REST API:

- **Endpoint:** https://*repstance_url*/configure/process
- **Method:** POST
- **Header:**
  - Content-Type: application/json
  - X-Token: *token*
- **Body:**

```
{
   "command": "alter",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","postgresql"],
      ["capturename","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["ddlcreate","objectMask:options"],
      ["ddldrop","objectMask:options"],
      ["ddlprocessing","native|dictionary"],
      ["skipapply","skipapplyMask"]
   ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli alter process=apply id=applyID \
      dbtype=postgresql \
      capturename=captureName \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName autostart={0|1} \
      debuglevel={0-15} \
      ddlcreate=objectMask:options \
      ddldrop=objectMask:options \
      ddlprocessing=native|dictionary \
```

```
skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process identifier
- **dbtype** – Type of RDBMS. The possible value is *postgresql*
- **capturename** – The name of the Capture Process which is providing the data to be used
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database user name
- **password** – Database user password
- **autostart** – Determines if the Apply Process must be run automatically. Possible values are:
  - *0* – Do not run the Apply Process automatically (default value)
  - *1* – To run the Apply Process automatically
- **debuglevel** – The level of debugging. Possible values are *0-15*
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The **%** symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – Do not run DDL create command if the table already exists
  - *recreate* – Drops the existing table and recreates it
- **ddldrop** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The **%** symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – Do not run DDL drop command if the table doesn't exist
- **ddlprocessing** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - *native* – To process DDL based on the user statement (default value)
  - *dictionary* – To generate DDL based on the dictionary changes
- **skipapply** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by ids separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*none*".

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.10.4 Alter Apply Process for MySQL and Aurora MySQL Databases

REST API:

- **Endpoint:** https://*repstance_url*/configure/process
- **Method:** POST
- **Header:**
  - Content-Type: application/json
  - X-Token: *token*
- **Body:**

```
{
   "command": "alter",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["dbtype","mysql"],
      ["capturename","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["ddlcreate","objectMask:options"],
      ["ddldrop","objectMask:options"],
      ["ddlprocessing","native|dictionary"],
      ["skipapply","skipapplyMask"]
   ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli alter process=apply id=applyID \
      dbtype=mysql capturename=captureName \
      server=databaseHost port=databasePort \
      user=username password=password \
      dbname=databaseName autostart={0|1} \
      debuglevel={0-15} \
```

```
ddlcreate=objectMask:options \
ddldrop=objectMask:options \
ddlprocessing=native|dictionary \
skipapply=skipapplyMask
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process identifier
- **dbtype** – Type of RDBMS. The possible value is *mysql*
- **capturename** – The name of the Capture Process which is providing the data to be used
- **server** – Host name or IP address of the database server
- **port** – Database port
- **dbname** – Database name
- **user** – Database user name
- **password** – Database user password
- **autostart** – Determines if the Apply Process must be run automatically. Possible values are:
  - *0* – Do not run the Apply Process automatically (default value)
  - *1* – To run the Apply Process automatically
- **debuglevel** – The level of debugging. Possible values are *0-15*
- **ddlcreate** – Used to define the Apply Process behavior when creating a table, if a table with the same name is found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – Do not run DDL create command if the table already exists
  - *recreate* – Drops the existing table and recreates it
- **ddldrop** – Used to define the Apply Process behavior when dropping a table, if a table with the same name is not found. The table is specified in *schema_name.table_name* format. In order to specify the list of the tables, each table must be separated by comma. The `%` symbol may be used in order to match any number of characters. It has the following options:
  - *skip* – Do not run DDL drop command if the table doesn't exist
- **ddlprocessing** – Determines if DDL must be processed based on the user statement or generated based on the dictionary changes. The possible values are:
  - *native* – To process DDL based on the user statement (default value)
  - *dictionary* – To generate DDL based on the dictionary changes
- **skipapply** – Used to define the behavior of the Apply Process when inserting data, which was originally produced by Apply Processes in the Source Database as part of the "loop-back" control function, in that, this Apply Process will insert, or ignore the data. The Apply Processes to be skipped are specified by ids separated by comma. In order to specify all Processes the "*all*" value to be used. In order to avoid skipping any Apply Processes the "*none*" value to be used. The default value is "*none*".

Note – "dictionary" value is commonly used in the case of Heterogeneous replication. For Homogeneous type of replication it is recommended to use "native" method of DDL processing.

7.11 Show Apply Process

The "**show**" Apply Process command is used to show an Apply configuration. This command will show the latest "Set" of configured Processes, regardless of whether or not the Apply Process has been "stopped and re-applied", this will not necessarily be the configuration of the currently running Apply Process.

Note – If the configuration has been changed when the Apply Process is running, the "**show**" command will still display the currently configured Apply Process, and not the "running" one.

REST API:

- **Endpoint:** `https://repstance_url/configure/process`
- **Method:** `POST`
- **Header:**
  - `Content-Type: application/json`
  - `X-Token: token`
- **Body:**

```
{
   "command": "show",
   "parameters": [
      ["process","apply"],
      ["id","applyID"]
   ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body:**

```
{
  "Status":"{Failed|Completed}",
  "Message":"The command execution details",
  "parameters":[
      ["process","apply"],
      ["id","applyID"],
      ["capturename","captureName"],
      ["server","databaseHost"],
      ["port","databasePort"],
      ["user","username"],
      ["password","password"],
      ["dbname","databaseName"],
      ["autostart","{0|1}"],
      ["debuglevel","{0-15}"],
      ["repuser","0|1"]
```

```
    ]
}
```

CLI Syntax:

```
repcli show process=apply id=applyID
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process identifier

Note – Only in the case that the command completes successfully will the response contain all the "parameter" values and at the same time this information will be displayed in Plain Text in the "Message" display.

7.12 Validate Apply Process

The "**validate**" Apply Process command is used to perform a set of background checks to ensure that all the necessary parameters have been supplied to enable the functionality of an Apply configuration on the Target Database. This command will also provide the "last" successfully applied transaction's LSN/SCN but **only** if the Apply Process has previously been run or reset.

REST API:

- **Endpoint:** https://*repstance_url*/configure/process
- **Method**: POST
- **Header**:
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
   "command": "validate",
   "parameters": [
      ["process","apply"],
      ["id","applyID"]
   ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body:**

```
{
  "Status":"{Valid|Invalid}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli validate process=apply id=applyID
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The Apply Process identifier

In case the **Status** is `Invalid` the **Message** will contain detailed information about the validation error.

7.13 Reset Apply Process

The "**reset**" command can **only** be used for an Apply Process – it is used where there is a need to force the Apply Process to start from specific LSN/SCN or after some LSN/SCN.

It is primarily used to "Set" or "Change" the "Startpoint" for an Apply Process.

REST API:

- **Endpoint**: https://*repstance_url*/reset/process
- **Method**: `POST`
- **Header**:
  - `Content-Type: application/json`
  - `X-Token: token`
- **Body**:

```
{
   "command": "reset",
   "parameters": [
      ["process","apply"],
      ["id","applyID"],
      ["LSN","LSN"],
      ["skip","{0|1}"],
   ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Completed|Failed}",
  "Message":"The command execution details",
}
```

CLI Syntax:

```
repcli reset process=apply id=applyID LSN=LSN skip={0|1}
```

The input parameters are:

- **`process`** – The only possible value is *`apply`*
- **`id`** – The Apply Process identifier
- **`LSN`** – The point at which we need to start reprocessing data
- **`skip`** – Determines if we need to jump over or not, a specific LSN

The following example demonstrates how to reset an Apply Process to start from the first transaction found, after **LSN** = **0x000000450000007B0004**:

```
repcli reset process=apply id=1 lsn=0x000000450000007B0004 skip=1
```

The following example demonstrates how to reset an Apply Process to start using the transactions from the **LSN** = **0x000000450000007F0004:**

```
repcli reset process=apply id=1 lsn=0x000000450000007F0004 skip=0
```

7.14 Remove Apply Process

The "**remove**" Apply Process command is used to remove **all** of the Apply Process – configuration files and the checkpoint details, together with historical information. If the Target Database **is reachable** the "**remove**" command will cleanup all the checkpoint information on the Target Database.

The command can only be executed if the specified Apply Process is not running.

It is strongly recommended that you backup configuration files and last successfully processed LSN before running this command (see chapter 8.4 Backup Repstance Files).

This command cannot be applied to a group of Apply Processes at the same time, it must be used individually for each of the configured Apply Processes.

REST API:

- **Endpoint:** `https://repstance_url/configure/process`
- **Method:** `POST`
- **Header:**
  - `Content-Type: application/json`
  - `X-Token: token`
- **Body:**

```
{
  "command": "remove",
  "parameters": [
    ["process","apply"],
    ["id","applyID"]
  ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **`200`** – If no error occurs
  - **`422`** – If error occurs

- **Body**:

```
{
  "Status":"{Completed|Error|Warning}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli remove process=apply id=applyID
```

The input parameters are:

- **process** – The only possible value is *apply*
- **id** – The identifier of the Apply Process

The output parameters are:

- **Status** – Status of the command. The possible values are:
  - *Error* – The command has failed to run
  - *Warning* – The process has failed to remove all the Apply Objects
  - *Completed* – The command has completed successfully
- **Message** – The details of command execution

The following example demonstrates how to remove Apply Process having id=2:

```
repcli remove process=apply id=2
```

7.15 Control Repstance Processes

There are three commands that are used to control the Capture and Apply Processes. They are :

- run
- stop
- status

Each of these commands can be used to control a single Process, a group of Processes or all Processes. **The default is to run "All" the Processes that are available to be run, stopped or get the status of, if no additional "Parameters" are specified**.

The syntax used for these commands is:

```
repcli run|stop|status [process=(capture|apply) [id=processID]]
```

Here are some examples of using **run** command.

This command will "run" everything (all existing Capture and Apply Processes):

```
repcli run
```

This command will "run" ALL Capture Processes:

```
repcli run process=capture
```

This command will "run" a single specified Capture Process:

```
repcli run process=capture id=1
```

The command format is identical in use for both the "stop" and "status" commands.

7.15.1 Run any Capture or Apply Processes

This command is used to "Run" either a single Capture or Apply Process or a group of either Capture, or Apply Processes or all configured Processes. **The default is to run all Processes that are available to be run, if no additional "Parameters" are specified**.

As part of the "**run**" command, Repstance carries out background checks to ensure that Source and/or Target Databases are properly configured with the necessary functionality to enable the Capture and/or Apply Processes to run. In the event that either of the Databases are not properly configured, it will display an error message listing the inconsistencies.

If this is the "First time" a Capture Process has been run, it is **ONLY** at this point that CDC will be implemented on the Source Database, and the first LSN will be written to the Trail File generated by this Capture Process.

If the previously defined Objects (see chapter 7.4.3 Capture Objects Specification) are changed or the Transformation Rules (see chapter 7.4.6 Transformation Rules and Triggering Order) are altered, at this point all the DDL and DML changes will be re-implemented using the new criteria.

If any Objects are no longer configured for Capture Process the "**run**" command will "Turn Off" CDC on these Objects and they will no longer be written to the Trail Files – the reverse is true i.e. if new Objects are specified then the "run" command will enable CDC for them and they will now be written to the Trial Files.

If this is the "First time" an Apply Process has been run, it is **ONLY** at this point that data from the specified Capture Process will be written to the Target Database using the first available LSN in the Trail File generated by this Capture Process.

The Apply Process can be configured to use a specific LSN from a Capture Process as a start point, i.e it can be configured to use any available LSN from a Trail File, but this is done by using the "**reset**" command (see chapter 7.13 Reset Apply Process).

If the specified Capture Process either is not configured or has never been run, the Apply Process will fail.

REST API:

- **Endpoint:** `https://repstance_url/control/process`
- **Method:** `POST`
- **Header:**
  - `Content-Type: application/json`
  - `X-Token: token`

- **Body:**

```
{
  "command": "run",
  "parameters": [
    ["process","apply|capture"],
    ["id","processID"]
  ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Completed|Error}",
  "Message":"The command execution details"
}
```

CLI Syntax:

```
repcli run process=apply|capture id=processID
```

The input parameters are:

- **process** – Constant value *apply* for Apply Processes or constant value *capture* for Capture Processes
- **id** – The identifier of the Process

The output parameters are:

- **Status** – Status of the command. The possible values are:
  - *Error* – The command has failed to run
  - *Completed* – The command has completed successfully
- **Message** – Details of command execution

7.15.2 Stop any Capture or Apply Processes

This command is used to stop either a single Capture or an Apply Process or a group of either Capture, or Apply Processes or all configured Processes. **The default is to stop everything that is running, if no additional parameters are specified**.

REST API:

- **Endpoint:** https://*repstance_url*/control/process
- **Method:** POST
- **Header:**
  - Content-Type: application/json
  - X-Token: *token*
- **Body:**

```
{
  "command": "stop",
  "parameters": [
```

```
      ["process","apply|capture"],
      ["id","processID"]
    ]
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
{
  "Status":"{Completed|Error}",
  "Message":"The command execution details",
}
```

CLI Syntax:

```
repcli stop process=apply|capture id=processID
```

The input parameters are:

- **process** – Constant value *apply* for Apply Processes or constant value *capture* for Capture Processes
- **id** – The identifier of the Process

The output parameters are:

- **Status** – Status of the command. The possible values are:
  - *Error* – The command has failed to run
  - *Completed* – The command has completed successfully
- **Message** – The details of command execution

7.15.3 Status of any Capture or Apply Processes

This command is used to show the status of either a single Capture or an Apply Process or a group of either Capture, or Apply Processes or all configured Processes. **The default is to show the status of every Process that is running, if no additional parameters are specified**.

REST API:

- **Endpoint:** https://*repstance_url*/control/process
- **Method**: POST
- **Header**:
  - Content-Type: application/json
  - X-Token: *token*
- **Body**:

```
{
  "command": "status",
  "parameters": [
    ["process","apply|capture"],
    ["id","processID"]
  ]
```

```
}
```

Server response:

- **HTTP Status** – The status of command. The possible codes are:
  - **200** – If no error occurs
  - **422** – If error occurs
- **Body**:

```
[
   {
      "Process":"capture|apply",
      "ID":processID,
      "Parameters":{
        "Name|CaptureName":"captureName",
        "Server":"databaseHost",
        "Port":"databasePort",
        "DBname":"databaseName"},
      "Command":"INITIALISED|RUN|STOP|RESET",
      "CommandStatus":"Completed|In Process|Failed",
      "StatusDetails":"Waiting for the Next Command|
                       Processing the Transactions|
                       Waiting for Transactions",
      "CommandStartTime":"Time_in_UTC",
      "LastActivityTime":"Time_in_UTC",
      "StartLSN":"LSN",
      "StartLSNTime":"Time_in_UTC",
      "CheckpointLSN":"LSN",
      "CheckpointLSNTime":"Time_in_UTC",
      "LastDBLSN":"LSN",
      "LastDBLSNTime":"Time_in_UTC",
      "WAITING":"time_in_nanoseconds",
      "PROCESSING":"time_in_nanoseconds",
      "LOADING":"time_in_nanoseconds",
      "Message":"The command execution details",
      "CNTD":"Number_of_deletes",
      "CNTI":"Number_of_inserts",
      "CNTU":"Number_of_updates",
      "CNTDDL":"Number_of_ddls",
      "CNTLOBJ":"Number_of_loaded_objects",
      "CNTLOBJREC":"Number_of_loaded_records",
      "CNTTX":"Number_of_transactions"
   },
   {
    Next_Process_description
   }, ...
]
```

CLI Syntax:

```
repcli status process=apply|capture id=processID
```

The input parameters are:

- **process** – The *apply* value for Apply Processes or the *capture* value for Capture Processes
- **id** – The identifier of the Process

The output parameters are:

- **`Process`** – Name of the configured Process. The possible values are:
  - *`capture`*
  - *`apply`*
- **`ID`** – The identifier of the Process
- **`Parameters`** – The list of the currently configured parameters:
  - *`Name`* – The name of a Capture Process (only for Capture Processes)
  - *`CaptureName`* – The name of Capture Process being used by the Apply Process (only for Apply Processes)
  - *`Server`* – Host name or IP address of the database server
  - *`Port`* – Database port
  - *`DBname`* – Database name
- **`Command`** – The last executed command on the listed Process. The possible values are:
  - *`INITIALISED`* – Means that the Process exists but currently has no command executed
  - *`RUN`* – Means that the Process is running
  - *`STOP`* – Means that the Process is stopping
  - *`RESET`* – Means that the Apply Process is resetting to the specified LSN
- **`CommandStatus`** – Status of the command. The possible values are:
  - *`Waiting for the Next Command`*
  - *`Processing the Transactions`*
  - *`Waiting for Transactions`*
  - *`Loading the Data`*
- **`CommandStartTime`** – Time in UTC format when the command was initialized (the parameter is available only in JSON response)
- **`LastActivityTime`** – Time in UTC format when the Process activity changed (the parameter is available only in JSON response)
- **`StartLSN`** – The first LSN (in hexadecimal format) when a transaction was processed (in **repcli** the parameter is displayed in the "Total since" section)
- **`StartLSNTime`** – Time of the transaction in UTC format the Process has started with (in **repcli** the parameter is displayed in the "Total since" section)
- **`CheckpointLSN`** – The LSN (in hexadecimal format) of the last transaction that has been successfully processed (in **repcli** the parameter is displayed in the "Last Captured/Applied DB Change" section)
- **`CheckpointLSNTime`** – Time in UTC format of the last transaction that has been successfully processed (in **repcli** the parameter is displayed in the "Last Captured/Applied DB Change" section)
- **`LastDBLSN`** – The last known transaction LSN (in hexadecimal format) in the Source Database. The parameter is provided by Capture Process and valid only in the case the Capture Process is running (in **repcli** the parameter is displayed in the "Last Sourced DB Change" section)
- **`LastDBLSNTime`** – Time in UTC format of the last known transaction LSN in the Source Database. The parameter is provided by Capture Process and valid only in the case the Capture Process is running (in **repcli** the parameter is displayed in the "Last Sourced DB Change" section)

- **WAITING** – Number of nanoseconds that the Process has been waiting for data to process (in **repcli** the parameter is displayed in the "Waiting for Transactions" section in HH:MM:SS.FFF format)
- **PROCESSING** – Number of nanoseconds that the Process has been processing the data (in **repcli** the parameter is displayed in the "Processing the Transactions" section in HH:MM:SS.FFF format)
- **LOADING** – Number of nanoseconds that the Process has been loading the data (in **repcli** the parameter is displayed in the "Loading the Data" section in HH:MM:SS.FFF format)
- **Message** – Details of the command execution
- **CNTD** – Number of Delete Operations currently performed by the Process (in **repcli** the parameter is displayed in the "Transactions" section on "Delete" position)
- **CNTI** – Number of Insert Operations currently performed by the Process (in **repcli** the parameter is displayed in the "Transactions" section on "Insert" position)
- **CNTU** – Number of Update Operations currently performed by the Process (in **repcli** the parameter is displayed in the "Transactions" section on "Update" position)
- **CNTDDL** – Number of DDL statements currently processed by the Process (in **repcli** the parameter is displayed in the "Transactions" section on "DDL" position)
- **CNTLOBJ** – Number of Objects currently processed by Initial Loading (in **repcli** the parameter is displayed in the "Loaded Objects" section)
- **CNTLOBJREC** – Number of records currently processed by Initial Loading (in **repcli** the parameter is displayed in the "Loaded Objects" section on "Rows" position)
- **CNTTX** – Number of transactions currently processed by the Process (in **repcli** the parameter is displayed in the "Transactions" section)

The **Average Speed** parameter is average number of records processed per second (calculated value).

The **Lag** parameter is the latency (delay) between the last record processed and the timestamp of the last transaction in Source Database (calculated value). The **Lag** is valid only in the case that the corresponding Capture Process is running. In **repcli** it is displayed in HH:MM:SS.FFF format.

The following example shows statuses of all existing Processes:

```
[http://localhost:8796/] repcli>status

Capture Process 1, Name: repv1, DB: mcdb, Server: source.repstance.com
  RUN (In Process), Waiting for Transactions
    Last Source DB Change   :  2019/03/10 21:47:00.610 UTC (0x000000E300001A750004)
    Last Captured DB Change :  2019/03/10 21:47:00.610 UTC (0x000000E300001A750004)
  Total since  2019/03/10 21:43:00.367 UTC (0x000000D400000A850004):
    Processing the Transactions :  00:00:02.214
    Waiting for Transactions    :  00:04:00.565
    Average Speed (ops)         :  45168.973
    Transactions   : 12 (DDL: 2,  Delete: 4,  Insert: 100016,  Update: 0)
```

```
Apply Process 1, Transactions Provider (Capture): repv1, DB: mcdb, Server:
target.repstance.com
  RUN (In Process), Waiting for Transactions
    Last Source DB Change   :  2019/03/10 21:47:00.610 UTC (0x000000E300001A750004)
    Last Applied DB Change  :  2019/03/10 21:47:00.610 UTC (0x000000E300001A750004)
    Lag                     :  00:00:00.000
  Total since  2019/03/10 21:44:00.696 UTC (0x00000000000000000001):
    Processing the Transactions :  00:00:09.230
    Waiting for Transactions    :  00:03:23.067
    Average Speed (ops)         :  10836.252
    Transactions   : 12 (DDL: 2,  Delete: 4,  Insert: 100016,  Update: 0)
```

The following example shows status of Capture Processes:

```
[http://localhost:8796/] repcli>status process=capture

Capture Process 1, Name: cap1, DB: prod1, Server: source.repstance.com
  RUN (In Process), Waiting for Transactions
    Last Source DB Change   :  2019/03/05 19:58:00.283 UTC (0x0000006F00004C900004)
    Last Captured DB Change :  2019/03/05 19:58:31.103 UTC (0x0000006F00005C340003)
  Total since  2019/03/05 19:58:00.283 UTC (0x0000006F00004C900004):
    Processing the Transactions :  00:00:00.588
    Waiting for Transactions    :  00:00:01.165
    Loading the Data            :  00:00:04.926
    Average Load Speed (ops)    :  190907.891
    Average Speed (ops)    :  0.000
    Transactions   : 0 (DDL: 0,  Delete: 0,  Insert: 0,  Update: 0)
    Loaded Objects : 2 (Rows: 940416)
```

8. REPSTANCE SERVER MAINTENANCE

As Repstance is a pre-configured Linux service it starts/stops together with the Server and MUST be maintained using the Linux Service Command.

8.1 How to Stop Repstance Services

In order to stop the Repstance Server the "service repstance stop" must be used:

```
# service repstance stop
Stopping repstance:          [ OK ]
```

8.2 How to Start Repstance Services

In order to start Repstance Server the "service repstance start" must be used:

```
# service repstance start
Starting repstance:          [ OK ]
```

8.3 How to Check Repstance Service Status

In order to check Repstance Server status the "service repstance status" must be used:

```
# service repstance status
repstance (pid 3033) is running...
```

8.4 Backup Repstance Files

Regularly backup the Trail Files which are located in `/opt/repstance/trail` directory along with the Configuration Files, which are located in `/opt/repstance/conf` and Capture Checkpoint Files in `/opt/repstance/captureckpt.*` and lastly make sure that you record the last applied checkpoint in a file of you choosing.

8.5 Housekeeping

Schedule regular jobs which are responsible for cleaning up "old" Trail and Log Files which are found under these directories:

- `/opt/repstance/trail` – Trail Files
- `/opt/repstance/log` – Log Files

GLOSSARY

Source Database – Is the Database from which you wish to extract/copy Data.

Target Database – Is the Database into which you wish to insert/copy Data.

Capture Process – Is the "Process" that extracts Data from the specified Source Database.

Apply Process – Is the "Process" that inserts the extracted Data into the specified Target Database.

Process – This can be either Capture or Apply Process.

Trail Files – These are the "Source Files" – containing Data extracted by a Capture Process and used by an Apply Process.

LSN – Every record in Trail Files is identified by a Log Sequence Number generated by SQL Server CDC functionality.

SCN – every record in the Trail Files is identified by a System Change Number generated by Oracle.

Capture Checkpoint – The last LSN identifier for any data change captured.

Apply Checkpoint – The last LSN identifier for any data change applied.

DML – Type of the Database Statement, that is used to modify Data in the tables.

DDL – Type of the Database Statement, that is used to provide or modify a Dictionary Definition of a database object.

Transformation Rules – these are the rules provided by user to change/control transactions processing behavior.