Oracle® Communications

Diameter Signaling Router

DCA Programmer's Guide

Release 8.6.0.0.0

F56203-01

April 2022

Table of Contents

List of Figures

List of Tables

1 Introduction

Diameter Custom Applications (DCA) is a framework that enables a significant reduction of the coding – testing – deployment – maintenance cycle in the development of Diameter applications.

The present document is intended to developers of DCA Apps. It describes how DCA Apps can be created, how their business logic and configuration data can be provisioned, how their lifecycle from development to production can be managed, as well as the various APIs available.

Following the DCA Framework and DCA Apps activation (chapter 2), the document is organized around three DCA Apps examples: "Blacklist" (chapter 3), "CountULR" (chapter 6) and "Rate" (chapter 8), which demonstrate the basic features of the DCA Framework. A number of additional chapters, interleaved with the chapters describing the three DCA Apps provide a gradual insight into essential capabilities of the DCA framework, like the DCA App lifecycle management (chapter 4), statefull DCA Apps development mechanisms (chapter 5) and tools for monitoring the execution of DCA Apps (chapter 7).

Chapter 9 provides a complete GUI reference.

The various APIs available are described in chapter 10.

1.1 References

[1] DCA Framework and Application Activation and Deactivation

[2] DCA Development Environment

[3] DSR Software Installation and Configuration Procedure

1.2 Glossary

This section lists terms and acronyms specific to this document.

Acronym	Description
API	Application Programming Interface
ART	Application Routing Table
AVP	Attribute Value Pair (in context of Diameter protocol)
ComAgent	Communication Agent
DA-MP	Diameter Agent Message Processor
DAI	DSR Application Infrastructure
DAL	Diameter Application Layer
DBCA	Database Change Agent
DCA	Diameter Custom Applications (framework)
DRL	Diameter Routing Layer
DSR	Diameter Signaling Router
EDL	Encode-Decode Library
UDR	Unified Data Repository
JSON	Java Script Object Notation
MEAL	Measurement, Event and Alarm
MO	Managed Object
NOAM	Network Operations Administration and Maintenance
OAM	Operations, Administration & Maintenance
OID	Object Identifier (SNMP)
Perl	“Practical Extraction and Reporting Language” – a scripting language
PRT	Peer Routing Table
SNMP	Simple Network Management Protocol
SOAM	Site Operations Administration and Maintenance

TTR	Trace Transaction Record (in context of IDIH)

1.3 Terminology

Acronym	Description
A-Level	NOAM –level
Asynchronous Call Symbol	Symbol in the Development Environment that represents a code statement that calls an asynchronous function provided by the DCA Perl API. The code statement occurs within a preceding Execution Block. The symbol displays the name of an asynchronous function that is invoked.
B-Level	SOAM- level
DCE Development Environment	Web application where a custom Diameter application developer can edit, save, check syntax, compile the application code for a Diameter Custom Application and generate an Interactive Flow Control Chart from the application code.
Execution Block Symbol	Symbol in the Development Environment that corresponds to an application subroutine where the name of the symbol is also the name of the subroutine.
Internal Variable	A storage mechanism that allows persistence during a Diameter transaction lifetime.
Start Symbol	Symbol in the Development Environment that marks the start of execution for the application.
Termination Symbol	Symbol in the Development Environment that represents an end point of the application execution.

2 DCA Activation and Deactivation

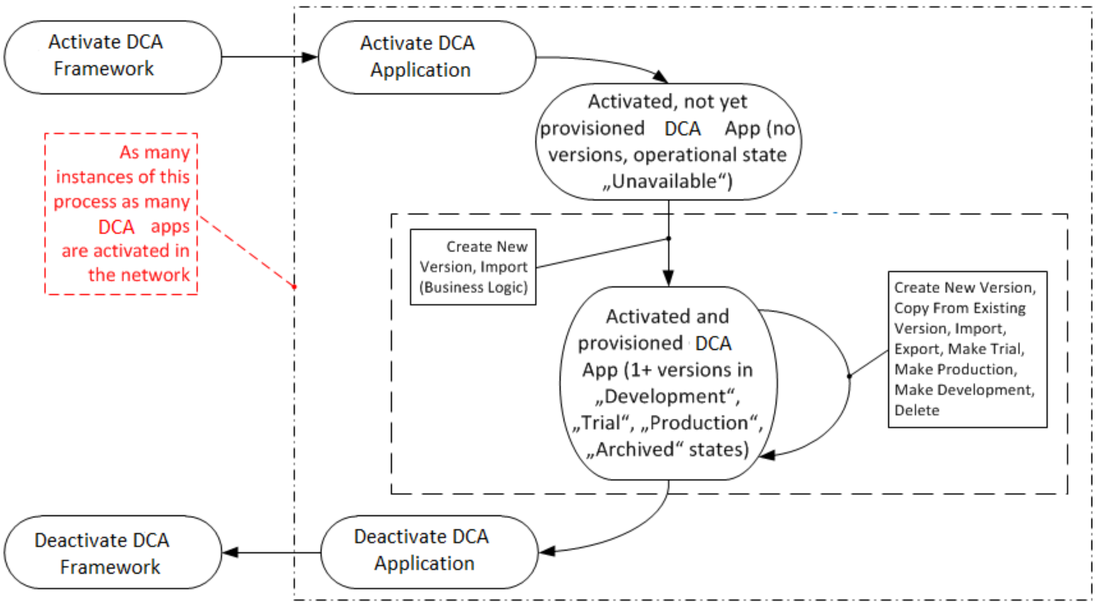
Activation and deactivation are standard procedures that enable DSR applications in general and DCA Apps in particular to be “installed” and “uninstalled” on a network.

2.1 DCA Activation

In order to start developing a new DCA App, the following two steps need to be performed:

- Activation of the DCA framework on the NO. See Procedure 5 in [1] for the instructions. This step needs to be performed only once for a given network.
- Activation of the new DCA App on the NO. See Procedure 6 in [1] for the instructions. This step has to be performed once per DCA App (similar to native DSR applications). Note however that only a limited number of DCA Apps (currently 5) can be simultaneously activated. Therefore, old DCA Apps may need to be deactivated in order to make room to new DCA Apps.

Figure 1 provides an overview of the activation-deactivation lifecycle.



Activate DCA Framework
Activate DCA Application
Activated, not yet provisioned DCA App (no versions, operational state „Unavailable“)
As many instances of this process as many DCA apps are activated in the network
Create New Version, Import (Business Logic)
Activated and provisioned DCA App (1+ versions in „Development“, „Trial“, „Production“, „Archived“ states)
Create New Version, Copy From Existing Version, Import, Export, Make Trial, Make Production, Make Development, Delete
Deactivate DCA Framework
Deactivate DCA Application

Figure 1: DCA Activation- Deactivation Lifecycle

2.1.1 DCA Framework Activation

When the DCA framework is initialized, the DCA Framework folder with the Configuration file becomes visible in the left side menu (Figure 2).

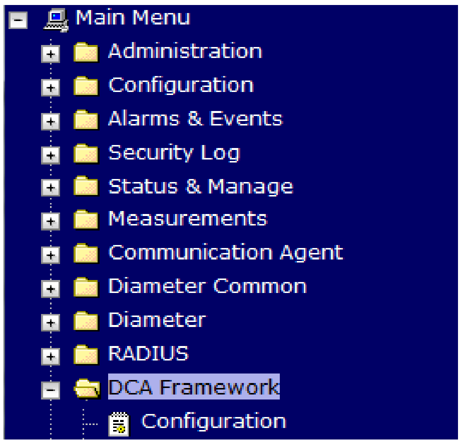
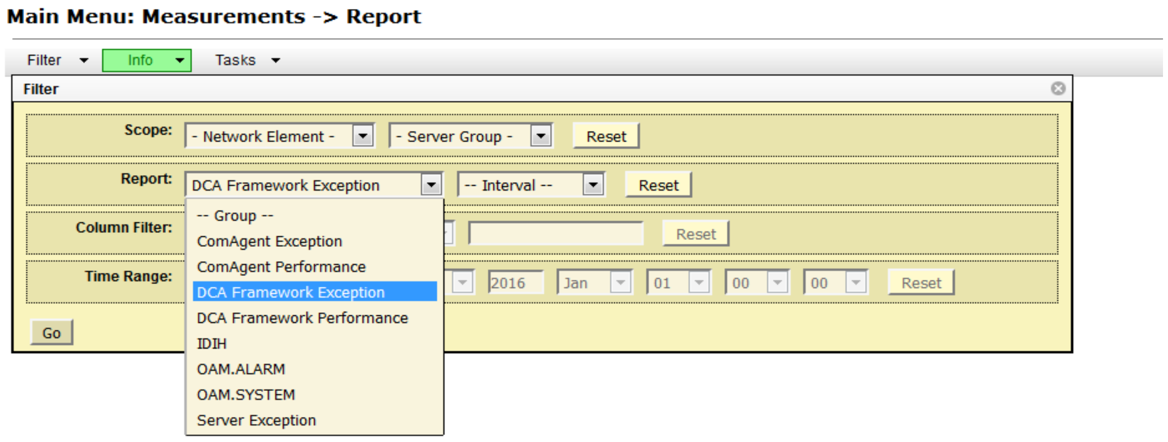


Figure 2 DCA Framework Menu

All the measurements (Figure 3) and KPIs (Figure 4) associated with the DCA Framework become visible as well.


Main Menu: Measurements -> Report

Filter | Info | Tasks

Filter

Scope: - Network Element - | - Server Group - | Reset

Report: DCA Framework Exception | -- Interval -- | Reset

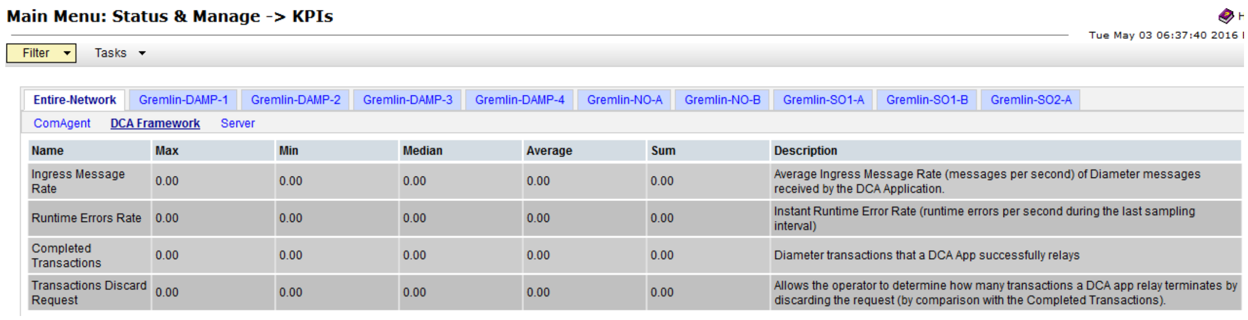
Column Filter: | Reset

Time Range: 2016 | Jan | 01 | 00 | 00 | Reset

Go

-- Group --
ComAgent Exception
ComAgent Performance
DCA Framework Exception
DCA Framework Performance
IDIH
OAM.ALARM
OAM.SYSTEM
Server Exception

Figure 3 DCA Measurements


Main Menu: Status & Manage -> KPIs

Tue May 03 06:37:40 2016

Filter | Tasks

Entire-Network | Gremlin-DAMP-1 | Gremlin-DAMP-2 | Gremlin-DAMP-3 | Gremlin-DAMP-4 | Gremlin-NO-A | Gremlin-NO-B | Gremlin-SO1-A | Gremlin-SO1-B | Gremlin-SO2-A

ComAgent | DCA Framework | Server

Name	Max	Min	Median	Average	Sum	Description
Ingress Message Rate	0.00	0.00	0.00	0.00	0.00	Average Ingress Message Rate (messages per second) of Diameter messages received by the DCA Application.
Runtime Errors Rate	0.00	0.00	0.00	0.00	0.00	Instant Runtime Error Rate (runtime errors per second during the last sampling interval)
Completed Transactions	0.00	0.00	0.00	0.00	0.00	Diameter transactions that a DCA App successfully relays
Transactions Discard Request	0.00	0.00	0.00	0.00	0.00	Allows the operator to determine how many transactions a DCA app relay terminates by discarding the request (by comparison with the Completed Transactions).

Figure 4: DCA KPIs

2.1.2 DCA App Activation

When the new DCA App is activated, the DCA App subfolder with the name provided by the user during the activation procedure becomes visible in the left side menu (Figure 5). The DCA App subfolder includes the screens for enabling the business logic and provisioning configuration data. The DCA App becomes visible across DSR (ART, maintenance screen, etc.).

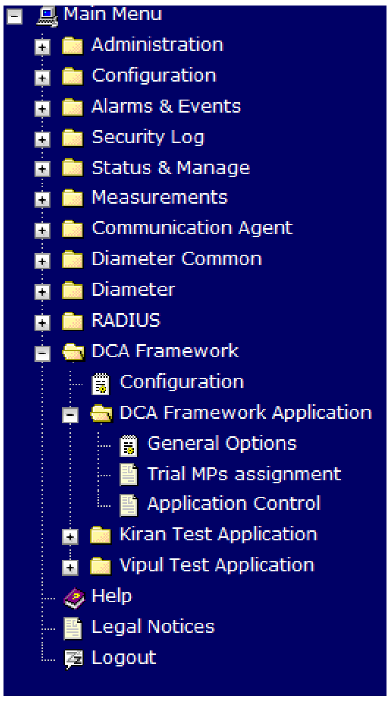


Figure 5: DCA Application Menu

2.1.3 Post-Activation DCA App State

Following the activation procedure, the DCA App is in the disabled state. While in disabled state, Diameter traffic will not be delivered to the DCA App. First, the DCA App must be enabled from the **SO Main Menu: Diameter→Maintenance→Applications**. Note that on this screen the DCA App is identified by the "short name" configured by the user during the DCA App activation procedure.

Independently from the enabled/disabled state of the DCA App, at this stage no version of the DCA App has been provisioned yet. As a result, there is no version in “Production” and “Trial” state. As long as no “Production” or “Trial” version is available for a DA-MP to run, the DCA App‘s operational status will be “unavailable”(see **Main Menu: Diameter→Maintenance→Applications,** on the SO).

The behavior of a DCA App while in operational state “unavailable” (provided that the DCA App has been enabled) is configurable on the SO from the **Main Menu:DCA Framework→<DCA App Name>→System Options** (see section 9.4); possible options are dropping the Diameter request, forwarding the Diameter request or returning a Diameter answer with a configurable error code.

From this point on the user can provision the configuration and business logic for the DCA App.

2.2 DCA Deactivation

The deactivation procedures enable a DCA App and respectively the DCA framework to be removed from a given network.

2.2.1 DCA Application De-Activation

The deactivation of a DCA App will not be allowed as long as versions of the respective DCA App are still in “Production” and/or a “Trial” state (see chapter 4).

Following deactivation, the DCA app‘s GUI folder under „DCA Framework“ menu item will disappear. The DCA App will be deregistered from the ART, its KPIs and measurements will not be displayed and respectively reported any longer.

2.2.2 DCA Framework De-Activation

DCA framework deactivation will not be allowed as long as at least one DCA App is activated in the network.

Following deactivation, the DCA framework GUI folder will disappear from the left-hand GUI menu.

3 DCA App Provisioning – The “Blacklist” DCA App

This section is a learning by doing guide to provisioning the configuration data and business logic for a simple DCA App.

3.1 The „Blacklist“ DCA App

The “Blacklist” DCA App checks the Origin-Host AVP of incoming Diameter requests and verifies whether it is blacklisted or not. In case the Origin-Host is blacklisted, the Diameter request will be dropped, otherwise the Diameter request will be forwarded unchanged.

3.2 Prerequisites

The DCA Framework must have been previously activated as described in [1]. Also, a DCA App with the name “Blacklist” shall be activated as described in [1].

The “Blacklist” DCA App has to be enabled on all the DA-MPs in the network from the SO **Main Menu: Diameter→Maintenance→Applications**.

An ART rule shall be added that enables Diameter messages to be delivered to the “Blacklist” DCA App.

3.3 The Process

The following step must be followed in order to provision the “Blacklist” DCA App:

Step 1: Configure the general options and behavior of the "Blacklist" DCA App;

Step 2: Create a new development version of the “Blacklist” DCA App;

Step 3: Define the structure of tables to store the “Blacklist” configuration data;

Step 4: Provision the “Blacklist” configuration data;

Step 5: Provision the “Blacklist” business logic – essentially a Perl script;

Step 6: Render the Flow Control Chart based on the Perl script. Save and perform syntax checks;

Step 7: Test the “Blacklist” DCA App: configure the Trial DA-MPs and promote “Blacklist” to Trial state;

Step 8: Compile “Blacklist”, promote “Blacklist” to Production state.

3.3.1 Step 1: Configure the DCA App's General Options and Behavior

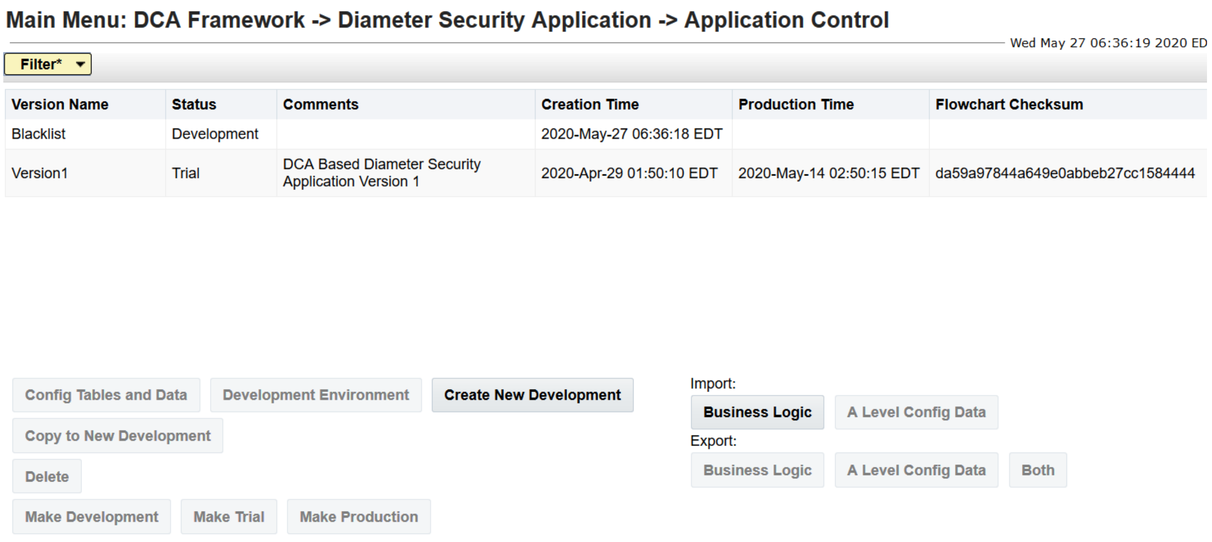
At this stage there is no version available for the “Blacklist” DCA App. As a result, the DCA App will be in the operational state “Unavailable”. No traffic is forwarded to the “Blacklist” DCA App and for outside observers the DCA App behaves as specified in the SO screen **Main Menu: DCA Framework→<DCA App Name>→System Options**, **Application unavailable configuration** section (see also 9.4).

The **Run-time error configuration** section of the same screen defines the behavior of the DCA App in case a runtime error occurs during the execution of the event handlers.

Finally, the DCA App programmer must ensure that the names specified on the NO screen **Main Menu: DCA Framework→<DCA App Name>→General Options** (see section 9.2.3) for the Diameter request and answer event handlers (Perl subroutines) are consistently used in the Perl script. For "Blacklist" in particular, “Perl Subroutine for Diameter Answer” shall be left empty because there is no event handler defined to process the Diameter answers.

3.3.2 Step 2: Create New Development Application Version

Go to the **Main Menu: DCA Framework→<DCA App Name>→Application Control** screen on the **NO** and click **"Create New Development"** (see Figure 6). The "Create New Development" screen will be displayed. Specify a name for the newly created "Blacklist" version and optionally provide comments (e.g. author name, brief description of the business logic, etc.). Figure 7 shows the newly created version.

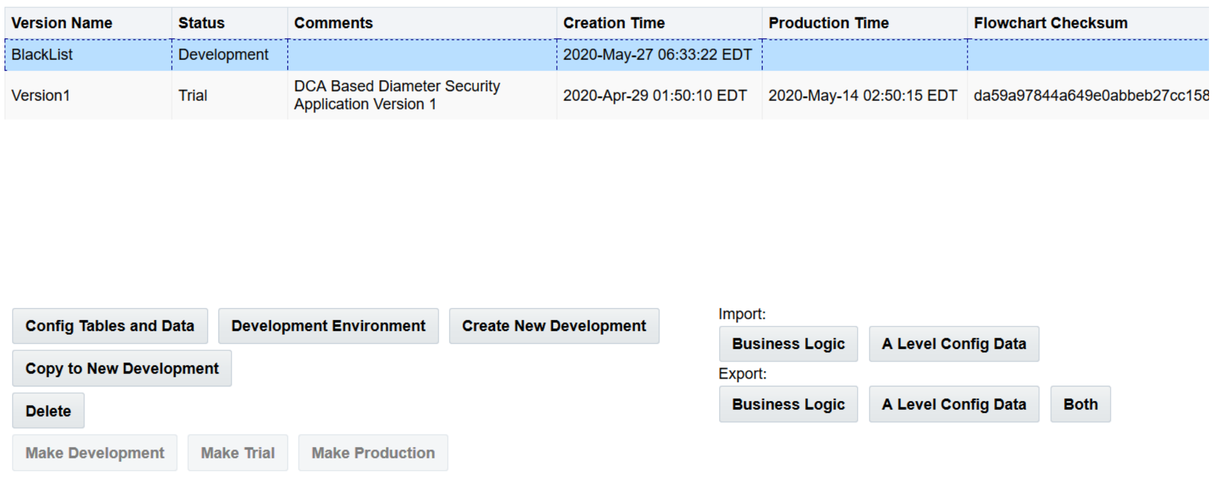


Main Menu: DCA Framework -> Diameter Security Application -> Application Control

Wed May 27 06:36:19 2020 ED

Version Name	Status	Comments	Creation Time	Production Time	Flowchart Checksum
Blacklist	Development		2020-May-27 06:36:18 EDT		
Version1	Trial	DCA Based Diameter Security Application Version 1	2020-Apr-29 01:50:10 EDT	2020-May-14 02:50:15 EDT	da59a97844a649e0abbeb27cc1584444

Figure 6: Create a New Application Version

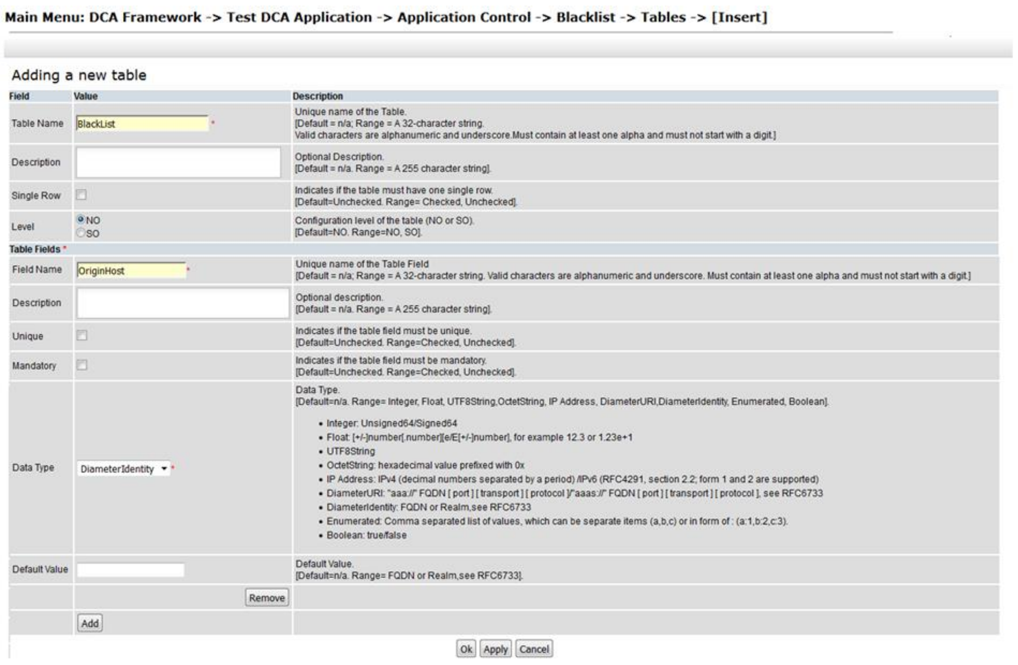


Version Name	Status	Comments	Creation Time	Production Time	Flowchart Checksum
BlackList	Development		2020-May-27 06:33:22 EDT		
Version1	Trial	DCA Based Diameter Security Application Version 1	2020-Apr-29 01:50:10 EDT	2020-May-14 02:50:15 EDT	da59a97844a649e0abbeb27cc158

Figure 7: New Application Version Created

3.3.3 Step 3: Define the configuration data structure

Select the newly created development application version on the "Application Control" screen and click **"Config Tables and Data"**. The "Tables" screen (Figure 8) will open. Click the **"Insert"** button on the "Tables" screen and create a new configuration table for provisioning the blacklist. The "Blacklist" DCA App configuration table contains only one field: OriginHost, which is of type DiameterIdentity, see Figure 8).



Main Menu: DCA Framework -> Test DCA Application -> Application Control -> Blacklist -> Tables -> [Insert]

Adding a new table

Field	Value	Description
Table Name	BlackList *	Unique name of the Table. [Default = n/a; Range = A 32-character string. Valid characters are alphanumeric and underscore.Must contain at least one alpha and must not start with a digit.]
Description		Optional Description. [Default = n/a. Range = A 255 character string].
Single Row	☐	Indicates if the table must have one single row. [Default=Unchecked. Range= Checked, Unchecked].
Level	● NO ○ SO	Configuration level of the table (NO or SO). [Default=NO. Range=NO, SO].
Table Fields *		
Field Name	OriginHost *	Unique name of the Table Field [Default = n/a; Range = A 32-character string. Valid characters are alphanumeric and underscore. Must contain at least one alpha and must not start with a digit.]
Description		Optional description. [Default = n/a. Range = A 255 character string].
Unique	☐	Indicates if the table field must be unique. [Default=Unchecked. Range=Checked, Unchecked].
Mandatory	☐	Indicates if the table field must be mandatory. [Default=Unchecked. Range=Checked, Unchecked].
Data Type	DiameterIdentity *	Data Type. [Default=n/a. Range= Integer, Float, UTF8String,OctetString, IP Address, DiameterURI,DiameterIdentity, Enumerated, Boolean]. • Integer: Unsigned64/Signed64 • Float: [+/-]number[.number][e/E[+/-]number], for example 12.3 or 1.23e+1 • UTF8String • OctetString: hexadecimal value prefixed with 0x • IP Address: IPv4 (decimal numbers separated by a period) /IPv6 (RFC4291, section 2.2; form 1 and 2 are supported) • DiameterURI: "aaa://" FQDN [port] [transport] [protocol]/"aaas://" FQDN [port] [transport] [protocol], see RFC6733 • DiameterIdentity: FQDN or Realm,see RFC6733 • Enumerated: Comma separated list of values, which can be separate items (a,b,c) or in form of : (a:1,b:2,c:3). • Boolean: true/false
Default Value		Default Value. [Default=n/a. Range= FQDN or Realm,see RFC6733].
	Remove	
	Add	

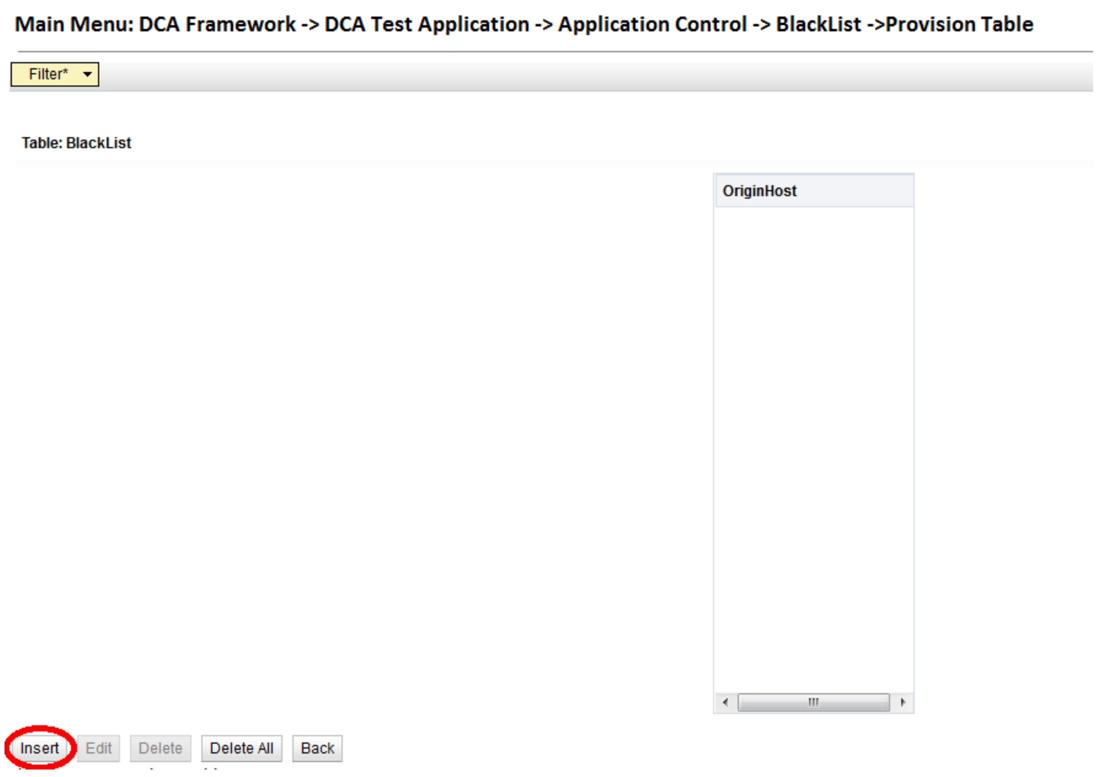
Figure 8: Create a New Database

Note: In this example the configuration table is defined at the NO level. That means the configuration table will be replicated to all the DA-MPs in the network.

Alternatively, a configuration table may be defined at the SO level. That means, while its structure is defined across the entire NO, its content will be replicated only to the DA-MPs in each individual SO. In this way distinct SOs may use different configuration data (see 9.3.5).

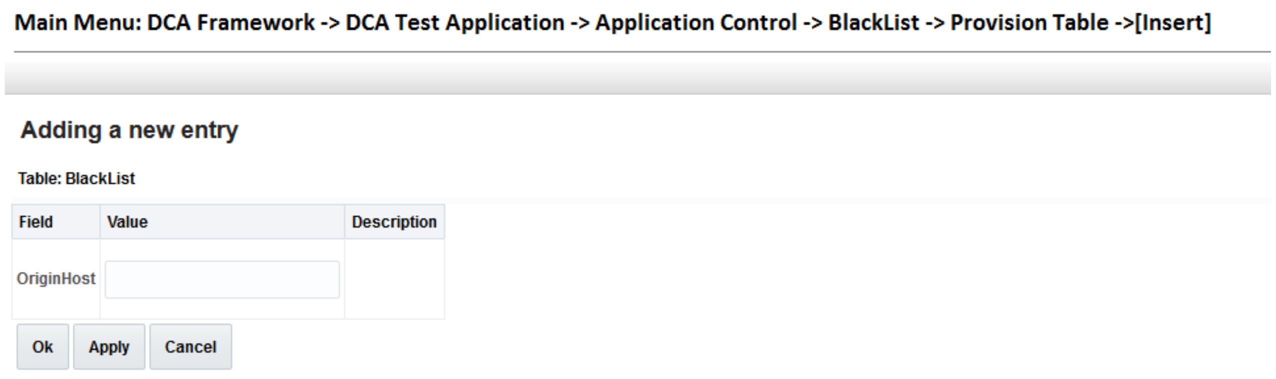
3.3.4 Step 4: Provision the Configuration Data

Once the structure of the “Blacklist” table is defined, the table will show up in the “Tables” screen. Select it and click **“Provision Table”** button. The “Provision Table View” screen will open (Figure 9). Click the **“Insert”** on the “Provision Table” View screen and insert all the blacklisted Origin-Hosts to the table one by one (Figure 10).



Main Menu: DCA Framework -> DCA Test Application -> Application Control -> BlackList ->Provision Table

Figure 9: Provision Table "BlackList"



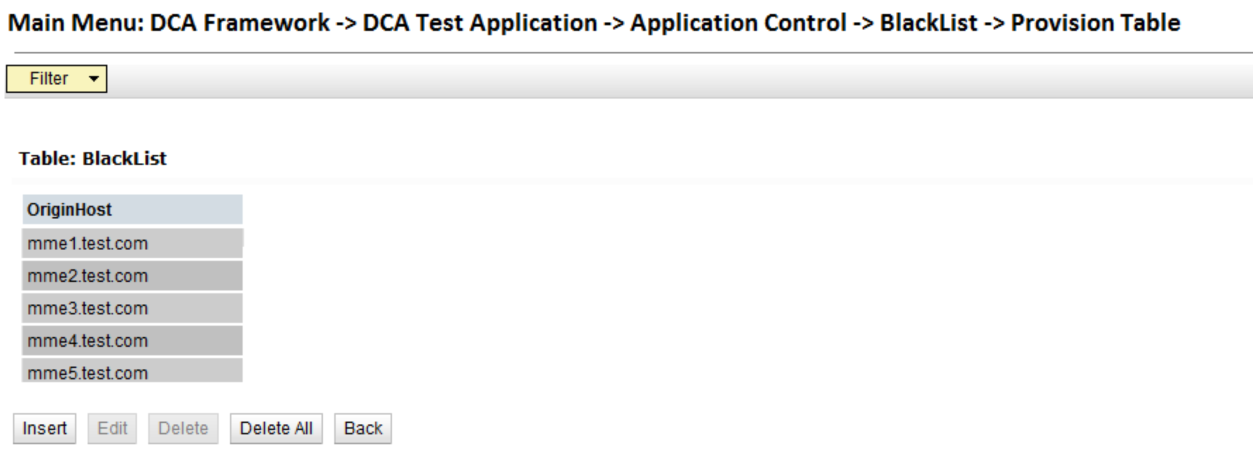
Main Menu: DCA Framework -> DCA Test Application -> Application Control -> BlackList -> Provision Table ->[Insert]

Adding a new entry

Table: BlackList

Field	Value	Description
OriginHost		

Figure 10: Insert a new data row to the "BlackList" table



Main Menu: DCA Framework -> DCA Test Application -> Application Control -> BlackList -> Provision Table

Table: BlackList

OriginHost
mme1.test.com
mme2.test.com
mme3.test.com
mme4.test.com
mme5.test.com

Figure 11: Provision DCA DB Tables

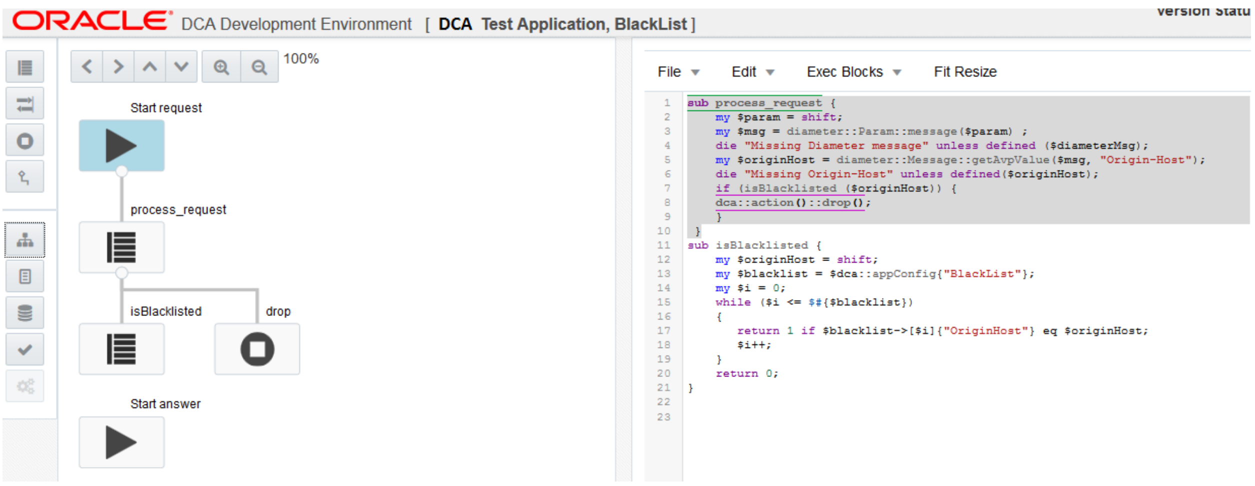
3.3.5 Step 5: Provision the Business Logic

Go back to the “Application Control” screen, select the application version and click the **“Development Environment”** button.

In the development environment the user can edit, save, check syntax and compile the DCA App's Perl code, which defines the business logic that the DCA App implements. Additionally, an interactive Flow Control Chart is rendered based the DCA App's Perl script. The Flow Control Chart provides an overview of the control flow within the DCA App and is particularly useful in following the asynchronous calls and indicating the terminating actions (forward, drop or return answer).

See [2] for more details on Development Environment.

The development environment of the “Blacklist” DCA App is illustrated in Figure 12.



```
sub process_request {
    my $param = shift;
    my $msg = diameter::Param::message($param) ;
    die "Missing Diameter message" unless defined ($diameterMsg);
    my $originHost = diameter::Message::getAvpValue($msg, "Origin-Host");
    die "Missing Origin-Host" unless defined($originHost);
    if (isBlacklisted ($originHost)) {
    dca::action()::drop();
    }
 }
sub isBlacklisted {
    my $originHost = shift;
    my $blacklist = $dca::appConfig{"BlackList"};
    my $i = 0;
    while ($i <= $#{$blacklist})
    {
       return 1 if $blacklist->[$i]{"OriginHost"} eq $originHost;
       $i++;
    }
    return 0;
}
```

Figure 12 The "Blacklist" DCA App Development Environment

First, the DCA App programmer has to write in the right-hand panel the Perl code illustrated in Figure 13. The left-hand panel containing the flowchart will be empty until the flowchart will be rendered in Step 6.

```
sub process_request {
    my $param = shift;
    my $msg = diameter::Param::message($param) ;
    die "Missing Diameter message" unless defined ($msg);
    my $originHost = diameter::Message::getAvpValue($msg, "Origin-Host");
    die "Missing Origin-Host" unless defined($originHost);
    if (isBlacklisted ($originHost)) {
          dca::action::drop();
    } else {
          dca::action::forward();
    }
}
sub isBlacklisted {
    my $originHost = shift;
    my $blacklist = $dca::appConfig{"BlackList"};
    my $i = 0;
    while ($i <= $#{$blacklist}) {
          return 1 if $blacklist->[$i]{"OriginHost"} eq $originHost;
```

```
            $i++;
        }
        return 0;
}
```

Figure 13 "Blacklist" Perl Code

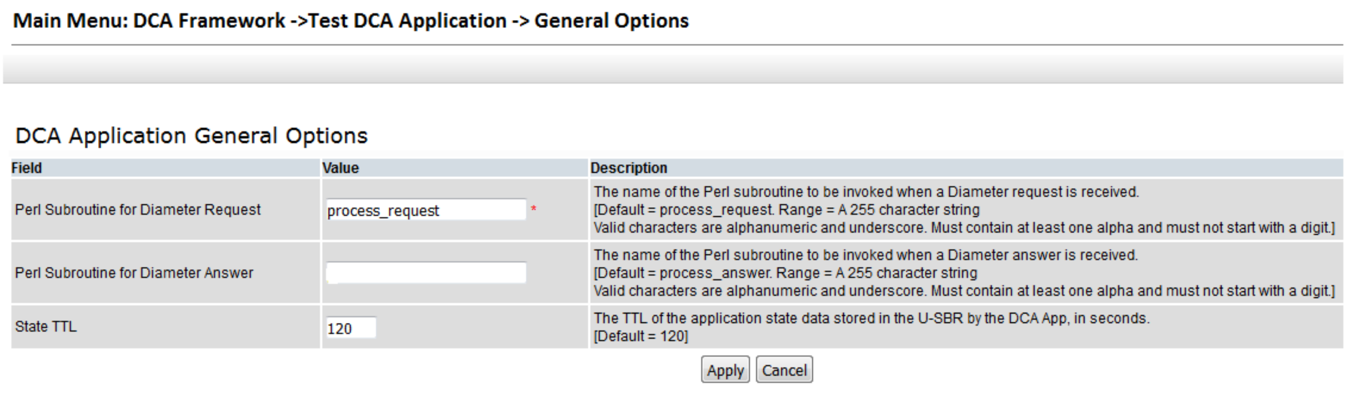
The Perl script (see Figure 13) makes use of the `getAvpValue` function to read the value of an AVP. The `getAvpValue` function is part of the EDL API, which is described in section 10.1.2. It also uses the `drop` and `forward` functions to discard and respectively forward the Diameter request. The drop function is part of the basic routing API, which is described in section 10.4.

3.3.5.1 Where is the Perl script being executed?

First, let's eliminate any possible confusion: even though the Perl script is edited via the NO GUI, the Perl script is replicated to and eventually executed on the DA-MPs. There is no possibility to make the Perl script process traffic other than running it on the DA-MPs.

3.3.5.2 How do the Event Handlers get invoked?

Let's observe that the business logic of a DCA App consists of a collection of event handlers, which are invoked when a Diameter message is delivered to the respective DCA App. A DCA App may therefore define one event handler for Diameter requests and one event handler for Diameter answers. Subsequent sections will introduce another category of event handlers, related to asynchronous database queries, but let's stick to the "Blacklist" DCA App for now. "Blacklist" defines only one event handler: `process_request`. Unlike `isBlacklisted`, which is a standard Perl subroutine invoked from `process_request`, `process_request` itself is not explicitly invoked from anywhere in the Perl script. The event handlers are explicitly invoked by the Perl running environment of the DCA framework. Their names are configured in the **NO Main Menu→DCA Framework→< Application Name>→General Options** screen and by default these names are `process_request` and `process_answer`. These names may be changed, but one needs to make sure that the configured event handler names are consistent with the names used in the Perl script. Also, the event handler names shall be left empty if there is no corresponding event handler defined in the Perl script (see Figure 14).



Main Menu: DCA Framework ->Test DCA Application -> General Options

DCA Application General Options

Field	Value	Description
Perl Subroutine for Diameter Request	process_request *	The name of the Perl subroutine to be invoked when a Diameter request is received. [Default = process_request. Range = A 255 character string Valid characters are alphanumeric and underscore. Must contain at least one alpha and must not start with a digit.]
Perl Subroutine for Diameter Answer		The name of the Perl subroutine to be invoked when a Diameter answer is received. [Default = process_answer. Range = A 255 character string Valid characters are alphanumeric and underscore. Must contain at least one alpha and must not start with a digit.]
State TTL	120	The TTL of the application state data stored in the U-SBR by the DCA App, in seconds. [Default = 120]

Figure 14 Event Handler Subroutine Name Configuration

3.3.5.3 How does the DCA App configuration data get accessed?

The configuration data of a DCA App is accessible to the Perl script through the `$dca::appConfig` variable, which is a complex variable representing a hash of arrays of hashes. One has to dereference it with exactly the same table names and field names specified when the structure of the configuration tables has been defined in step 3.3.3:

```
$dca::appConfig{"<table_name>"}->[<record_number>]{"<field_name>"}
```

in our case:

```
$dca::appConfig{"BlackList"}->[<record_number>]{"OriginHost"}
```

3.3.5.4 What is the „main part“ good for?

“Blacklist” has an empty “main part”. The “main part” of a Perl script is where the Perl interpreter starts executing instructions. In DCA the main part is executed **only once** following the successfully compilation of the script.

The “main part” is typically used to perform whatever initializations are necessary (like for instance Custom MEAL objects, as we will describe later on).

Another task that fits into the “main part” is DCA App configuration data post-processing. We have seen in section 3.3.5.3 that the “Blacklist” configuration data is accessible to the business logic (Perl script) as an array. “Blacklist” simply loops through the array when looking for each Origin-Host, but a more performance–aware version would certainly convert the array into a more performant data structure, like for instance a hash table keyed by the Origin-Host values.

Other DCA apps may even need to use multiple keys (hence multiple hash tables) or compound keys; the “main part” is the right place to perform this kind of structural optimizations on the DCA App configuration data.

3.3.6 Step 6: Render Flow Control Chart, Save Script, Check Syntax

After editing the script, while in the Development state, the following actions are possible (see Figure 15):

- **Render Chart** (to generate the flowchart from the Perl code);
- **Render Code** (to generate a Perl code skeleton from the flowchart);
- **Save** (to save the Perl code and the flowchart);
- **Check Syntax** (to check syntax of Perl script).

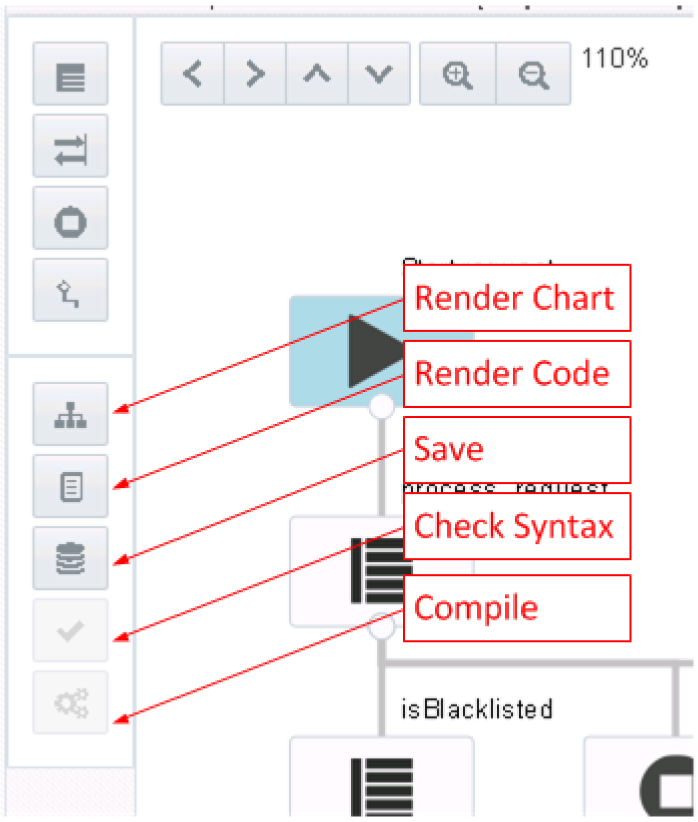


Figure 15 Development Environment Buttons

The "Render Chart" action generates a flowchart based on the Perl code. Note that the flowchart has a Perl *subroutine granularity* and not a Perl *instruction granularity*. The flowchart's main purposes are: (i) to describe how the callback subroutines are linked to the event handlers (Diameter message handlers or other callback subroutines) that registers them and (ii) to indicate the terminating actions (drop, forward or return answer).

The flowchart will not illustrate on which condition a Perl subroutine is invoked (i.e. if conditions) or how many times a Perl subroutine is invoked (i.e. loop conditions). Also, the "Render Chart" action shall be explicitly triggered by clicking the corresponding button after each modification of the Perl script.

The approach pursued by this ("Blacklist") and subsequent DCA App examples in this document ("CountULR" and "Rate") is based on the idea that a DCA App programmer will first provision the Perl code and then render the flowchart. The "Render Code" action allows a somewhat opposite approach, by first drawing a flowchart and then generating a Perl script *skeleton* based on it.

The “Save” button allows the flowchart and Perl code to be saved, while the DCA App version is in Development or Trial state.

The “Check Syntax” button becomes enabled once the "Save" action has been completed, while the DCA App version is in Development or Trial state. It performs a syntax check on the Perl code and displays the errors if the syntax check fails.

3.3.7 Step 7: Test the DCA App Version

Having the configuration data and business logic provisioned, it is now time to test the “Blacklist” DCA App.

A DCA App version is tested by promoting it to the Trial state, which will automatically result in running it on the dedicated Trial DA-MPs.

The first step is therefore to configure the Trial DA-MPs, which can be done from the “Trial MPs Assignment” screen (see Figure 16 and section 9.2.4).

The Trial DA-MPs assignment is configured per DCA App, that is, it needs not be repeated for each DCA App version.

Note also that our network contains only one DA-MP, which will be also a Trial DA-MP. However, in a real life deployment there would typically be a few Trial DA-MPs and a number of non-Trial DA-MPs.

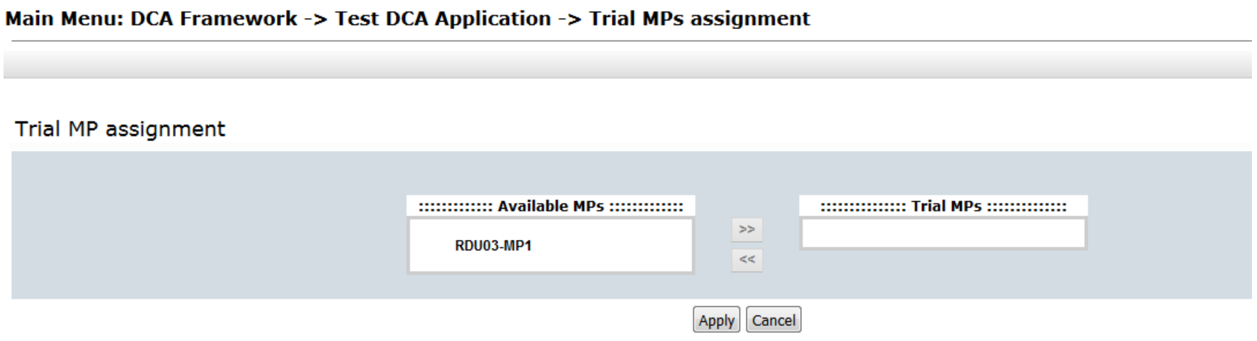

Main Menu: DCA Framework -> Test DCA Application -> Trial MPs assignment
Trial MP assignment
:::::::::::: Available MPs ::::::::::::
RDU03-MP1
>>
<<
:::::::::::::: Trial MPs ::::::::::::
Apply
Cancel

Figure 16: Trial MP Assignment

Next, on the “Application Control” screen, promote the DCA App version from Development to Trial state by selecting it and clicking on the **“Make Trial”** button.

While in Trial state the DCA App version can be: modified, saved, have the syntax checked and, in addition to the Development state, it can also be compiled (by clicking the "Compile" button, see Figure 15), as further described in chapter 4. During each new cycle starting with the first Perl code modification and lasting until the next successful compilation (with an arbitrarily number of modifications, save and syntax check actions taking place during this time), the Trial DA-MPs will execute the previously successfully compiled Perl script of the respective DCA App version.

If successfully compiled, the “Blacklist” DCA App on the Trial DA-MP will switch into the operational state Available (see the **SO Main Menu: Diameter→Maintenance→Applications** screen). On the non-Trial DA-MPs the DCA App operational state will remain Unavailable because there is no DCA App version in Production state at this moment.

3.3.8 Step 8: Promote the DCA App Version to Production State

A successfully compiled Trial DCA App version can be promoted to the Production state. For this purpose, on the “Application Control” screen, the DCA App version shall be selected and the **“Make Production”** button clicked.

At this stage the only DCA App version available so far is in Production state. All non-Trial DA-MPs will start running it and on these DA-MPs the DCA App operational state will become Available. Because there is no DCA App version in the Trial state, the Trial DA-MPs will run the Production version as well.

Please note that our network is a very particular case that contains one single DA-MP, which is configured as a Trial DA-MP. This means that the Production version will be executed on this only DA-MP if and only if no Trial version exists. As soon as a (new) Development version will be promoted to the Trial state, the Trial DA-MP will stop executing the Production version and will start executing the (new) Trial version.

While in Production state, the business logic of the DCA App version cannot be changed anymore. It’s only the configuration data that can be updated.

We have achieved our initial objective of running the “Blacklist” DCA App in our network. From this point on a number of alternatives are possible:

- Demote the DCA App version from Production state back to Development to fix bugs, re- test and promote back to Production state;
- Copy the DCA App version into a new version with the purpose to improve its business logic (in terms of efficiency, functionality or both) and eventually promote the newer version to Production state;
- Export the DCA App version from the current network and import it onto another network;

We are touching on the DCA App lifecycle management topic, which will be described in more detail in the next chapter.

4 DCA Application Lifecycle

The DCA Application Lifecycle enables the DCA App programmer to manage the lifecycle of a DCA App.

So far we have developed one single DCA App version, we tested it and promoted to the Production state. The state transitions are illustrated in Figure 17.

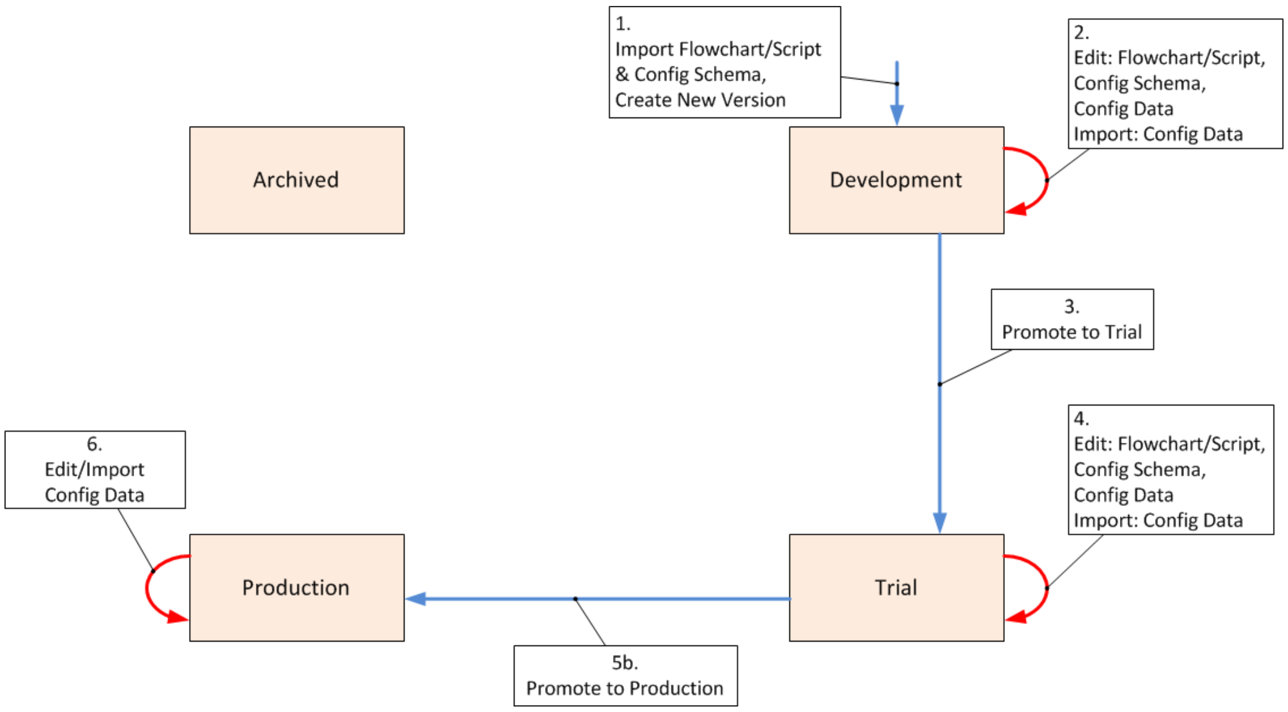

1.
Import Flowchart/Script & Config Schema, Create New Version
2.
Edit: Flowchart/Script, Config Schema, Config Data
Import: Config Data
Archived
Development
3.
Promote to Trial
4.
Edit: Flowchart/Script, Config Schema, Config Data
Import: Config Data
6.
Edit/Import Config Data
Production
Trial
5b.
Promote to Production

Figure 17 Transitions from Development to Production State

In a real life deployment a DCA App may need to be continuously enhanced both in terms of efficiency as well as features. A typical approach would be to “clone” the DCA App version currently in Production state to a new version in Development state, work on the new version (while the old version is processing the Diameter traffic), test the new version and eventually replace the older version in Production state with the newer one. This process is illustrated by the transition path 7 → 3 → 5b → 9 in Figure 18.

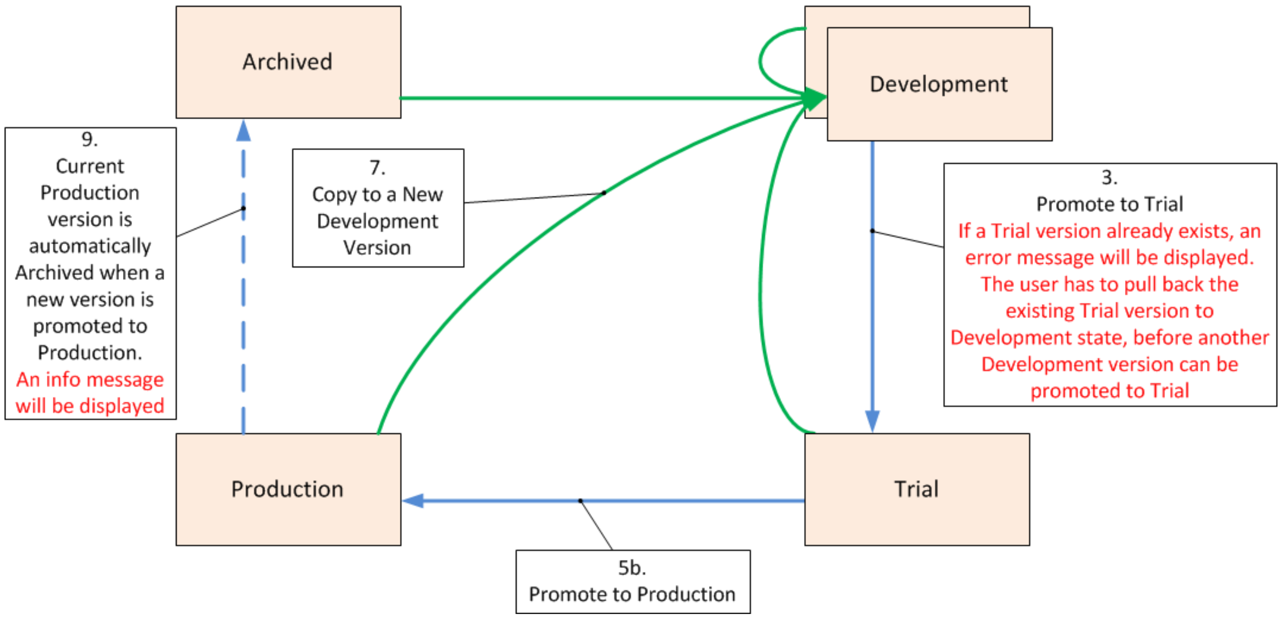

Archived
Development
9.
Current Production version is automatically Archived when a new version is promoted to Production.
An info message will be displayed
7.
Copy to a New Development Version
3.
Promote to Trial
If a Trial version already exists, an error message will be displayed. The user has to pull back the existing Trial version to Development state, before another Development version can be promoted to Trial
Production
Trial
5b.
Promote to Production

Figure 18 Creating a New DCA App Version

The DCA App Lifecycle management is done via the **Main Menu: DCA Framework→<DCA App Name>→Application Control** screen.

Each DCA app version can be in one of the following states:

- **Development** (initial state)
  - There are zero or more Development versions in the system.
  - Development version is not executed on any MP.
  - Configuration schema (databases), configuration data, flowchart may be updated.
  - A new version in Development state is created in the system when:
    - A "Create New Development" button is clicked, see 9.2.5. In this case, the version will have an empty flowchart, empty configuration schema and empty configuration data.
    - Importing the business logic (w/ or w/o configuration data), see 0. In this case the flowchart and the configuration schema (databases) will be copied from the imported version. Optionally, configuration data may be imported along with the business logic as well.
    - Copying a new Development version from an existing version in the system, see 0. In this case the business logic as well as the configuration data of the selected version will be copied into the new version.
- **Trial**
  - There are zero or one Trial versions in the system.
  - Trial version is executed on the DA-MPs assigned to run the Trial version
  - If no Trial version exists, then the Trial MPs will run the Production version (see Figure 19).
  - Configuration schema (databases), configuration data, flowchart may be updated.
- **Production**
  - There are zero or one Production version in the system.
  - When no Production version exists in the system, the operational state of the DCA application on MPs supposed to run the Production version will be set to "unavailable"(**Main Menu: Diameter→Maintenance→Applications)**. This may happen if the Production version is rolled back to the Development state or deleted.
  - Is executed:
    - On all the DA-MPs, if no Trial version exists, or
    - On all the DA-MPs except the DA-MPs assigned to run the Trial version, if a Trial version exists (see Figure 19).
  - Configuration schema (databases) & Flowchart are read-only.
  - Configuration data may be updated.
- **Archived**
  - There are zero or more Archived versions in the system.

– Archived versions are the application versions that have previously been in the Production state. They serve as backups for the purpose of bringing the system back to a previous known state with minimum service interruption.
– Archived version is not executed on any MP.
– Configuration schema (databases), Configuration Data and Flowchart are read-only, but can be exported and copied into a new version.

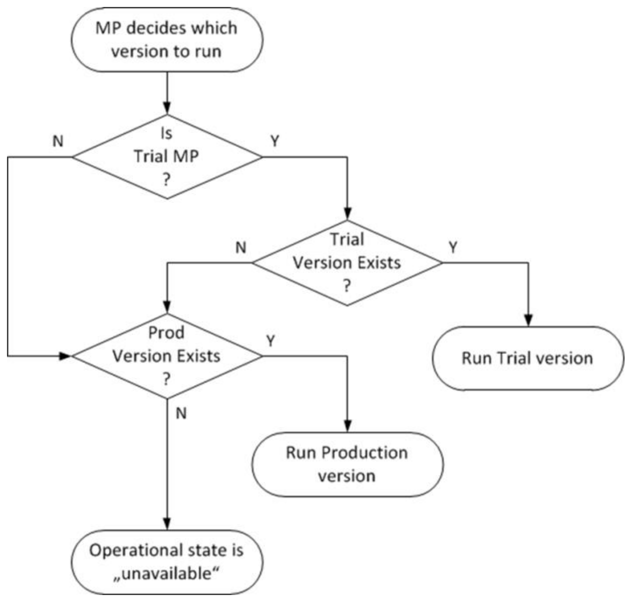


Figure 19: Assignment of the Version to a DA-MP

The following transitions are possible for a given DCA App version:

- Development → Trial (only if syntax was successfully checked and no other version is in Trial state)
- Trial → Production (only if the code/flow control chart was successfully compiled and no other version is in Production state)
- Production → Archived (automatic transition when a new version is promoted to Production)
- Trial → Development
- Production → Development (the operational state of the DCA App becomes Unavailable)
- Archived → Development
- Archived → Trial
- Archived → Production

5 Developing Statefull DCA Apps

The “Blacklist” DCA App introduced in chapter 3 was a stateless Diameter application because it was processing each Diameter message individually without maintaining any state between a Diameter request and its corresponding answer (Diameter transaction state) or across Diameter transactions (e.g Diameter session state) or across Diameter sessions (e.g. user state).

DCA Apps may however need to store state:

- Diameter transaction state – for instance collect some information from the Diameter request and use that information when processing the Diameter answer.
  This task can be addressed in two ways:
  1. Using the Diameter transaction context variables API documented in section 10.2.2.
  2. Developers familiar with the Internal Variables from the Mediation feature may use Internal Variables for this purpose, as described in section 10.2.1. However, Internal Variables involve a configuration overhead and therefore unless there is a strong argument in favor of using them (e.g. they need to be set or read from Mediation rules) the Diameter transaction context variables, being a purely programming interface, are preferable
- Diameter session or user state – for instance collect information across multiple Diameter transactions in the same session or user information across multiple Diameter sessions.
  This task can be addressed using the Universal Session Binding Repository (UDR) and is described in section 10.7.

6 A Statefull DCA App Using the UDR DB

In chapter 3 we have developed a stateless DCA App. Chapter 5 introduces the mechanisms available in DCA to develop statefull DCA Apps.

This chapter describes the additional configuration steps that need to be performed and introduces the API available to develop a statefull DCA App that uses the UDR (Unified Data Repository). The UDR provides a generic interface to the DSR, which implements a scalable, distributed and persistent database infrastructure, which DCA Apps as well as other Oracle applications may use.

6.1 The „CountULR“ DCA App

The “CountULR” DCA App maintains a per-user count of ULR messages and deletes it when a CLR message from the respective user is received. The user is identified based on the content of the User-Name AVP in the incoming Diameter requests.

6.2 Prerequisites

The DCA Framework must have been previously activated as described in [1]. Also, a DCA App with the name “CountULR” shall be activated as described in [1].

The “CountULR” DCA App has to be enabled on all the DA-MPs in the network from the SO **Main Menu: Diameter→Maintenance→Applications**.

An ART rule shall be added that enables ULR and CLR Diameter requests to be delivered to the “CountULR” DCA App.

6.3 The Process

The following steps must be followed in order to provision the “CountULR” DCA App:

Business Logic and Configuration Data Provisioning	**UDR DB Configuration**
Step 1: Configure the general options and behavior of the "CountULR" DCA App **Step 2:** Create a new development version of the “CountULR” DCA App;	**Step A:** Configure UDR DBs (as required by the DCA App business logic);

Step 3: Define the structure of tables to store the “CountULR” configuration data;
 	
Step 4: Provision the “CountULR” configuration data;
 	
Step 5: Provision the “CountULR” business logic – essentially a Perl script;
 	
Step 6: Render the Flow Control Chart based on the Perl script. Save and perform syntax checks;	

Step 7: Test the “CountULR” DCA App: configure the Trial DA-MPs and promote “CountULR” to Trial state;
 	
Step 8: Compile “CountULR”, promote “CountULR” to Production state;	

Steps 1 to 8 are similar to those described in chapter 3.

Steps A and B are required in order to create UDR DB and allow the “CountULR” DCA App to interact with it. UDR DB configuration is independent from the DCA App configuration, except that a relative ordering must be followed:

- Step A may be executed in any order relative to steps 1 and 8.

6.3.1 Step 1: Configure the DCA App's Global Options and Behavior

In addition to the considerations discussed in section 3.3.1, for DCA Apps that use UDR, the following configuration options may need to be adjusted:

- On the NO screen **Main Menu: DCA Framework→<DCA App Name>→General Options** (see section 9.2.3):
  - ❑ "Read-Only UDR Access as Guest", which may be used to control the access of the DCA App to UDR DBs owned by other DCA Apps. This option is not relevant to "CountULR" because "CountULR" will exclusively use the UDR DB owned by itself (see section **Error! Reference source not found.**)
- It is recommended the state data size (consisting of the size of the lookup key and respectively the size of the state data itself) of any new DCA App to be kept below the default values configured on the **NO Main Menu: DCA Framework→Configuration** screen (see section 9.2.1). If, for good reasons, a DCA App requires a larger lookup key or more data to store, then these limits shall be increased.
  Note that these limits apply globally to all active DCA Apps. As a result, decreasing these value may result in existing DCA Apps having their UDR queries rejected with a `dca::udr::ResultCode::MaxStateSize` error, and is therefore not recommended.

6.3.2 Step 2: Create a New Development Version

See section 3.3.2.

6.3.3 Step A: Configure the UDR DBs

6.3.3.1 Configure UDR DB as Remote server

Note: Comagent Configuration with UDR DB will be NOAM Level Configuration.

6.3.3.1.1 *ComAgent Configuration on DSR*

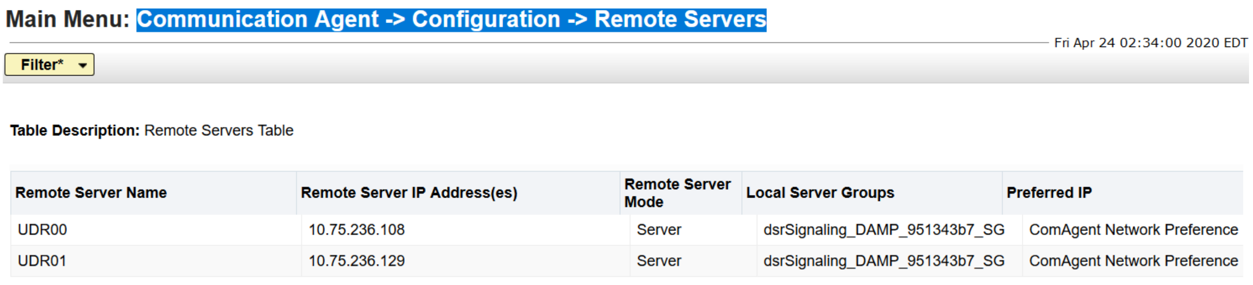
For Comagent configuration go to Communication agent TAB on Active DSR NO GUI and configure UDB DB Server IMI IP as remote server.

Note:

If DSR and UDR deployment are in same network use UDR IMI IP as Comagent Remote Server Configuration.
If DSR and UDR deployment are in different network use UDR XSI IP as Comagent Remote Server configuration.
For this, add new XSI Interface on both DSR and UDR side for Comagent Communication.
Make sure new added XSI interface are Desktop routable and accessible from both side.
Do not use DSR signaling Interface (XSI Interface) for comagent communication.

- **Remote Server Configuration :**

Configure UDR DB as Remote Server.


Main Menu: Communication Agent -> Configuration -> Remote Servers

Fri Apr 24 02:34:00 2020 EDT

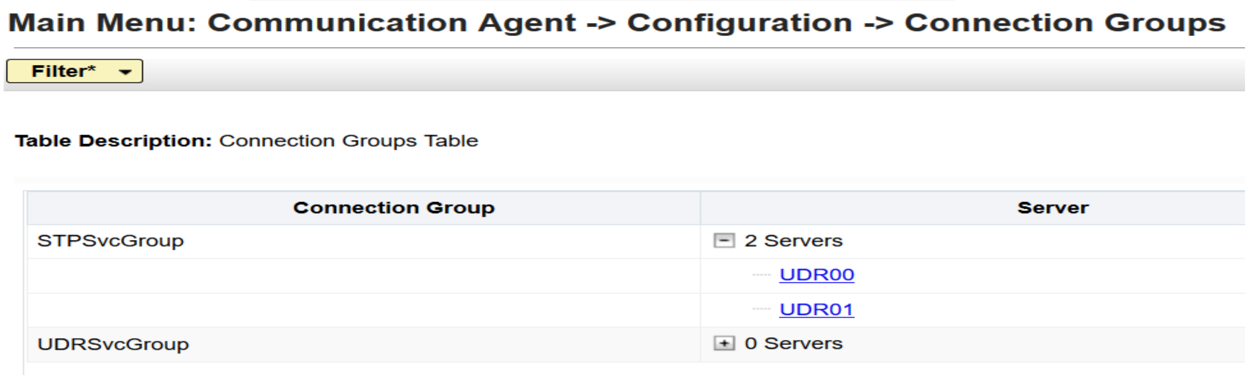
Filter*

Table Description: Remote Servers Table

Remote Server Name	Remote Server IP Address(es)	Remote Server Mode	Local Server Groups	Preferred IP
UDR00	10.75.236.108	Server	dsrSignaling_DAMP_951343b7_SG	ComAgent Network Preference
UDR01	10.75.236.129	Server	dsrSignaling_DAMP_951343b7_SG	ComAgent Network Preference

Figure Remote Server Configuration on DSR NO Server

- **Connection Group configuration :**
  Add previously configured Remote Server to **STBDbSvc** Connection Group.


Main Menu: Communication Agent -> Configuration -> Connection Groups

Filter*

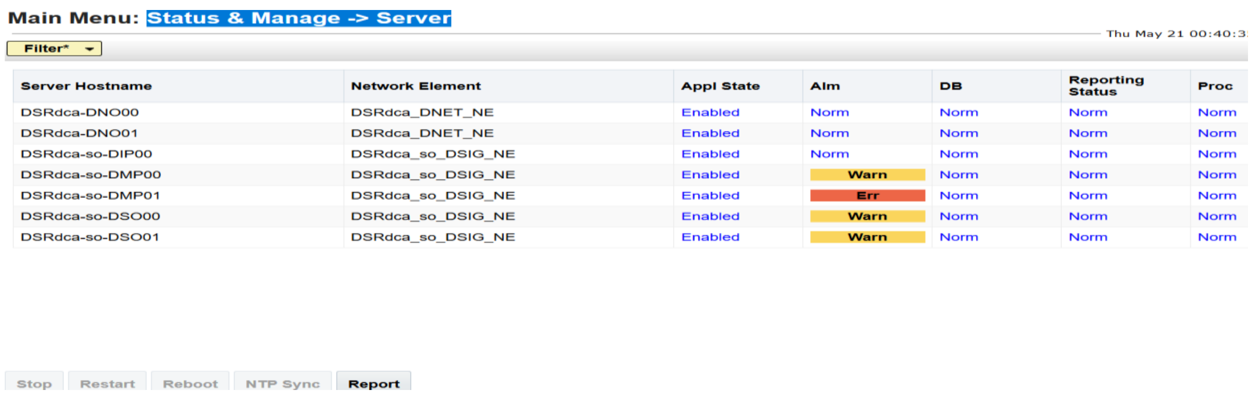
Table Description: Connection Groups Table

Connection Group	Server
STPSvcGroup	⊟ 2 Servers
	UDR00
	UDR01
UDRSvcGroup	⊞ 0 Servers

Figure Connection group Configuration on DSR NO Server

Note: Restart the MPs Server to make the Comagent service /connection up

- **Steps to Restart the MPs Server :**
  - Go to Active DSR NOAM status & Manage section, select the MP server and restart the MP server with click on restart button.


Main Menu: Status & Manage -> Server

Thu May 21 00:40:3

Filter*

Server Hostname	Network Element	Appl State	Alm	DB	Reporting Status	Proc
DSRdca-DNO00	DSRdca_DNET_NE	Enabled	Norm	Norm	Norm	Norm
DSRdca-DNO01	DSRdca_DNET_NE	Enabled	Norm	Norm	Norm	Norm
DSRdca-so-DIP00	DSRdca_so_DSIG_NE	Enabled	Norm	Norm	Norm	Norm
DSRdca-so-DMP00	DSRdca_so_DSIG_NE	Enabled	Warn	Norm	Norm	Norm
DSRdca-so-DMP01	DSRdca_so_DSIG_NE	Enabled	Err	Norm	Norm	Norm
DSRdca-so-DSO00	DSRdca_so_DSIG_NE	Enabled	Warn	Norm	Norm	Norm
DSRdca-so-DSO01	DSRdca_so_DSIG_NE	Enabled	Warn	Norm	Norm	Norm

Figure Active NOAM Status and Manage screen

6.3.3.1.2 *Comagent Configuration on UDR*

For Comagent configuration go to Communication agent TAB on UDR NO GUI and configure all the DSR MP IMI IP as client.
Note: Please refer section #ComAgent Configuration on DSR Note for configuring the Interface IP as client.

- **Remote Server Configuration :**

  Configure DSR MPs IMI IP as Client.

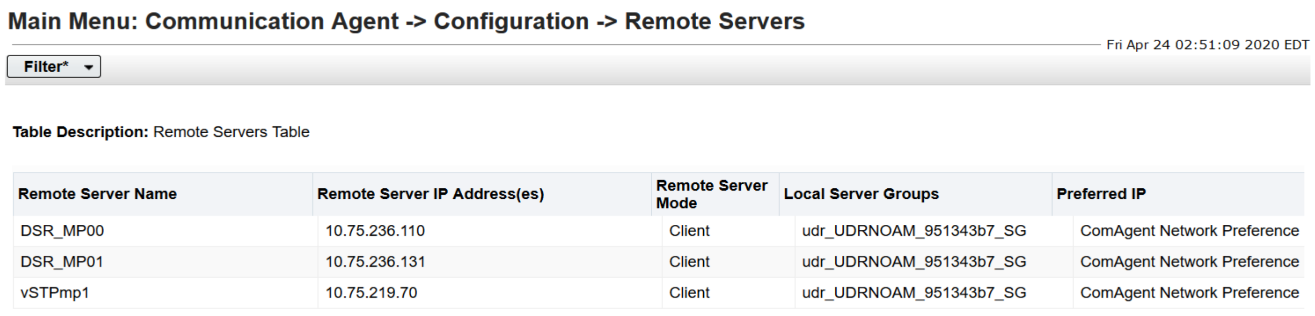
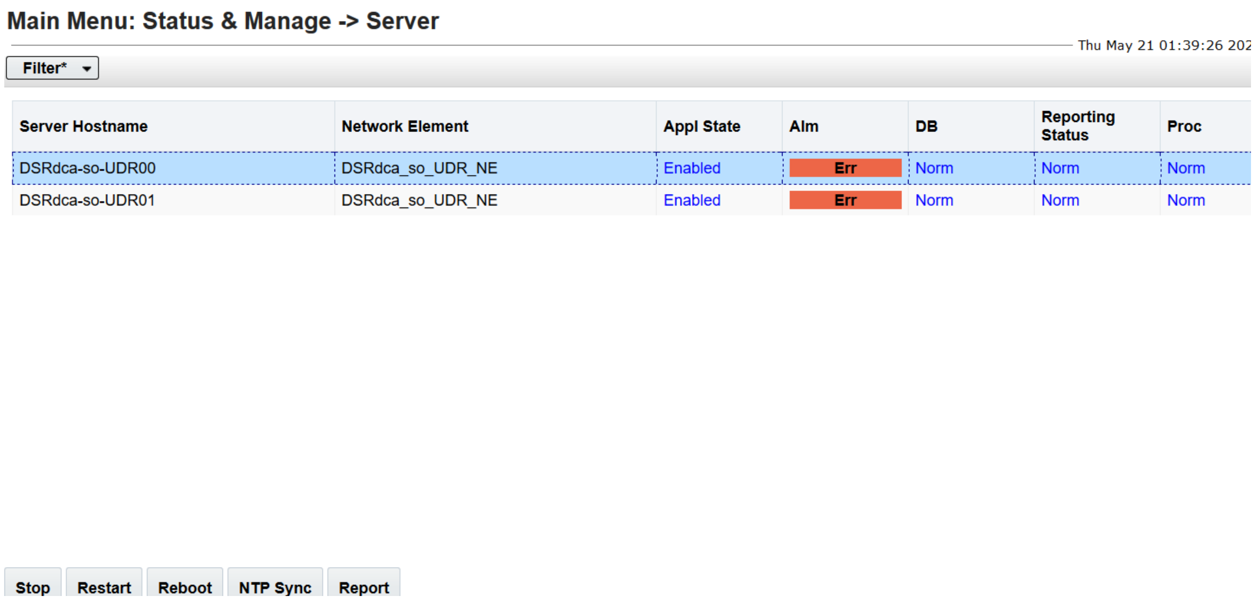

Main Menu: Communication Agent -> Configuration -> Remote Servers
Fri Apr 24 02:51:09 2020 EDT
Filter*

Table Description: Remote Servers Table

Remote Server Name	Remote Server IP Address(es)	Remote Server Mode	Local Server Groups	Preferred IP
DSR_MP00	10.75.236.110	Client	udr_UDRNOAM_951343b7_SG	ComAgent Network Preference
DSR_MP01	10.75.236.131	Client	udr_UDRNOAM_951343b7_SG	ComAgent Network Preference
vSTPmp1	10.75.219.70	Client	udr_UDRNOAM_951343b7_SG	ComAgent Network Preference

Note: Reboot the Active UDR NOAM Server to make the Comagent service /connection up.

- **Steps to Restart the MPs Server :**
  - Go to Active UDR NOAM status & Manage section, select the Active NOAM server and Reboot the Active NOAM server with click on reboot button.

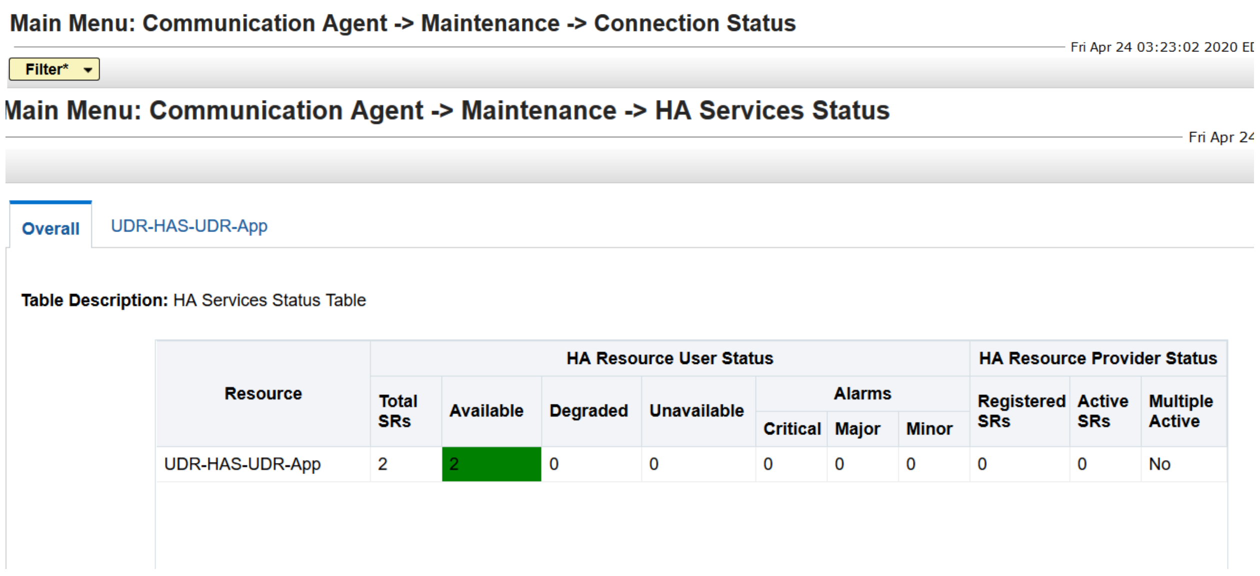


Server Hostname	Network Element	Appl State	Alm	DB	Reporting Status	Proc
DSRdca-so-UDR00	DSRdca_so_UDR_NE	Enabled	Err	Norm	Norm	Norm
DSRdca-so-UDR01	DSRdca_so_UDR_NE	Enabled	Err	Norm	Norm	Norm

6.3.3.1.3 *Comagent Connection Status Validation*

- **Comagent Connection status check on DSR NO Server :**

For Connection, status check go to Communication agent Maintenance TAB on DSR NO GUI.



Main Menu: Communication Agent -> Maintenance -> Connection Status

Fri Apr 24 03:23:02 2020 EI

Filter*

Main Menu: Communication Agent -> Maintenance -> HA Services Status

Fri Apr 24

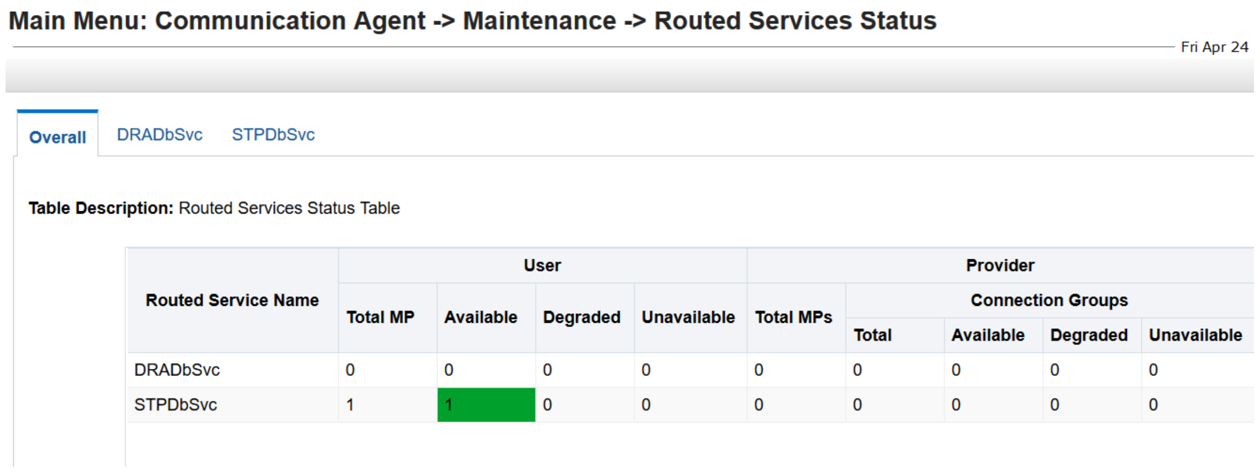
Overall | UDR-HAS-UDR-App

Table Description: HA Services Status Table

Resource	HA Resource User Status							HA Resource Provider Status		
	Total SRs	Available	Degraded	Unavailable	Alarms			Registered SRs	Active SRs	Multiple Active
					Critical	Major	Minor			
UDR-HAS-UDR-App	2	2	0	0	0	0	0	0	0	No

- **Routed Service status check on DSR NO Server :**

For routed service, status check go to Communication agent Maintenance TAB on DSR NO GUI.



Main Menu: Communication Agent -> Maintenance -> Routed Services Status

Fri Apr 24

Overall | DRADbSvc | STPDbSvc

Table Description: Routed Services Status Table

Routed Service Name	User				Provider				
	Total MP	Available	Degraded	Unavailable	Total MPs	Connection Groups			
						Total	Available	Degraded	Unavailable
DRADbSvc	0	0	0	0	0	0	0	0	0
STPDbSvc	1	1	0	0	0	0	0	0	0

Figure Routed Service status on DSR NOAM Server

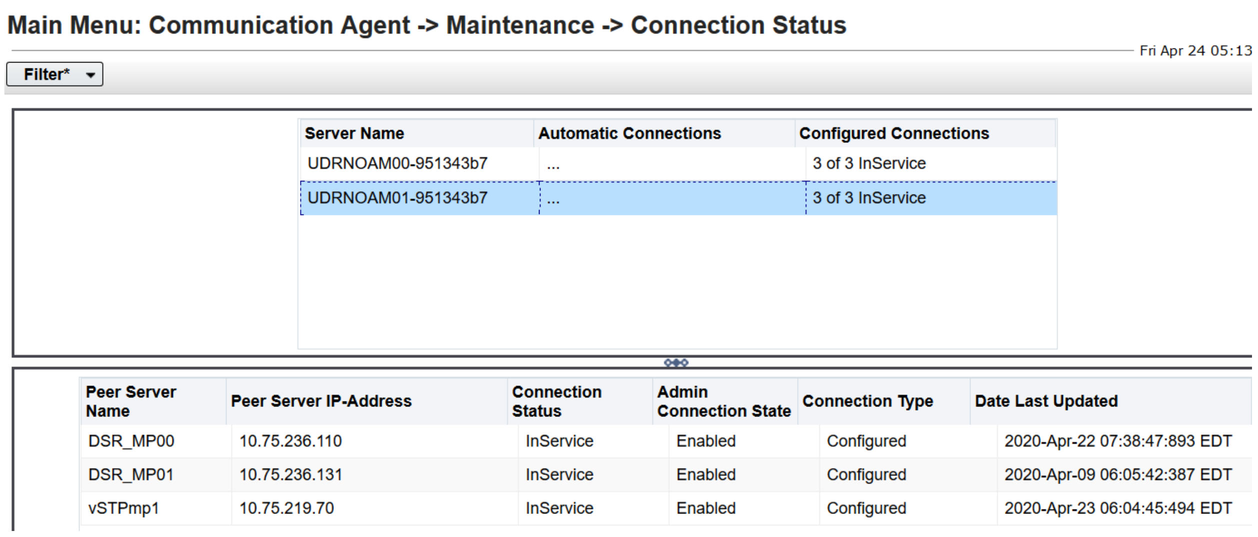
- **HA Service status check on DSR NO Server :**

For HA Service status check go to Communication agent Maintenance TAB on DSR NO GUI.

- **Comagent Connection status check on UDR NO Server :**

For Connection, status check go to Communication agent Maintenance TAB on UDR NO GUI.

Figure HA Service Status Check on NOAM Server



Main Menu: Communication Agent -> Maintenance -> Connection Status

Fri Apr 24 05:13

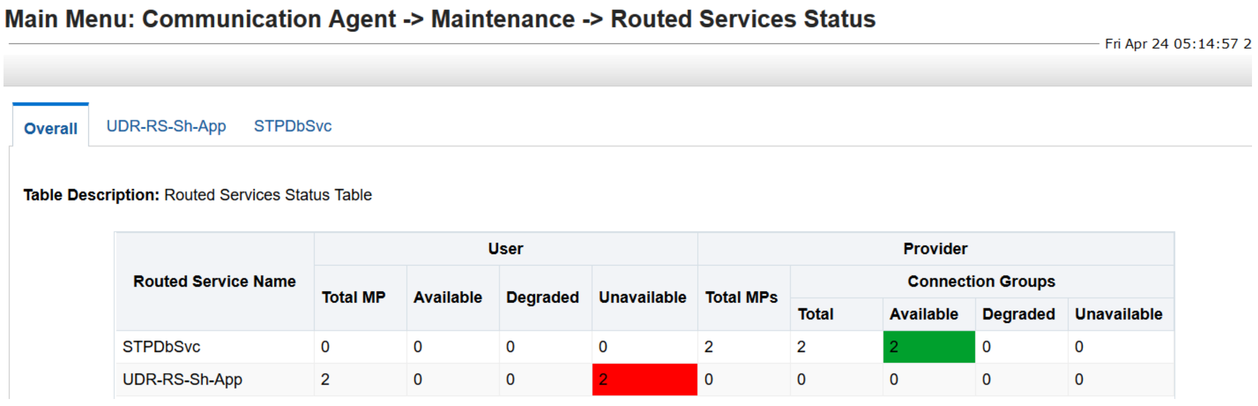
Filter*

Server Name	Automatic Connections	Configured Connections
UDRNOAM00-951343b7	...	3 of 3 InService
UDRNOAM01-951343b7	...	3 of 3 InService

Peer Server Name	Peer Server IP-Address	Connection Status	Admin Connection State	Connection Type	Date Last Updated
DSR_MP00	10.75.236.110	InService	Enabled	Configured	2020-Apr-22 07:38:47:893 EDT
DSR_MP01	10.75.236.131	InService	Enabled	Configured	2020-Apr-09 06:05:42:387 EDT
vSTPmp1	10.75.219.70	InService	Enabled	Configured	2020-Apr-23 06:04:45:494 EDT

- **Routed Service status check on UDR NO Server :**

For routed service, status check go to Communication agent Maintenance TAB on UDR NO GUI.



Main Menu: Communication Agent -> Maintenance -> Routed Services Status

Fri Apr 24 05:14:57 2

Overall | UDR-RS-Sh-App | STPDbSvc

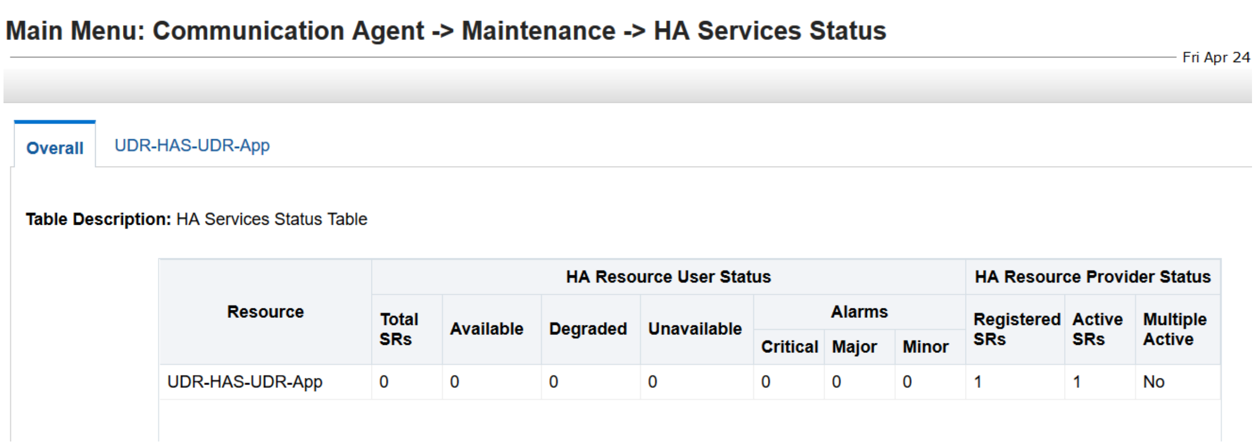
Table Description: Routed Services Status Table

Routed Service Name	User				Provider				
						Connection Groups			
	Total MP	Available	Degraded	Unavailable	Total MPs	Total	Available	Degraded	Unavailable
STPDbSvc	0	0	0	0	2	2	🟩 2	0	0
UDR-RS-Sh-App	2	0	0	🟥 2	0	0	0	0	0

Figure Routed Service Status check on UDR NO Server

- **HA Service status check on UDR NO Server :**

For HA Service status check go to Communication agent Maintenance TAB on UDR NO GUI.


Main Menu: Communication Agent -> Maintenance -> HA Services Status

Fri Apr 24

Table Description: HA Services Status Table

Resource	HA Resource User Status							HA Resource Provider Status		
	Total SRs	Available	Degraded	Unavailable	Alarms			Registered SRs	Active SRs	Multiple Active
					Critical	Major	Minor			
UDR-HAS-UDR-App	0	0	0	0	0	0	0	1	1	No

Figure Routed Service status check on UDR NO Server

6.3.3.2 Enable Security Profile on Active UDR NOAM for DSA Application

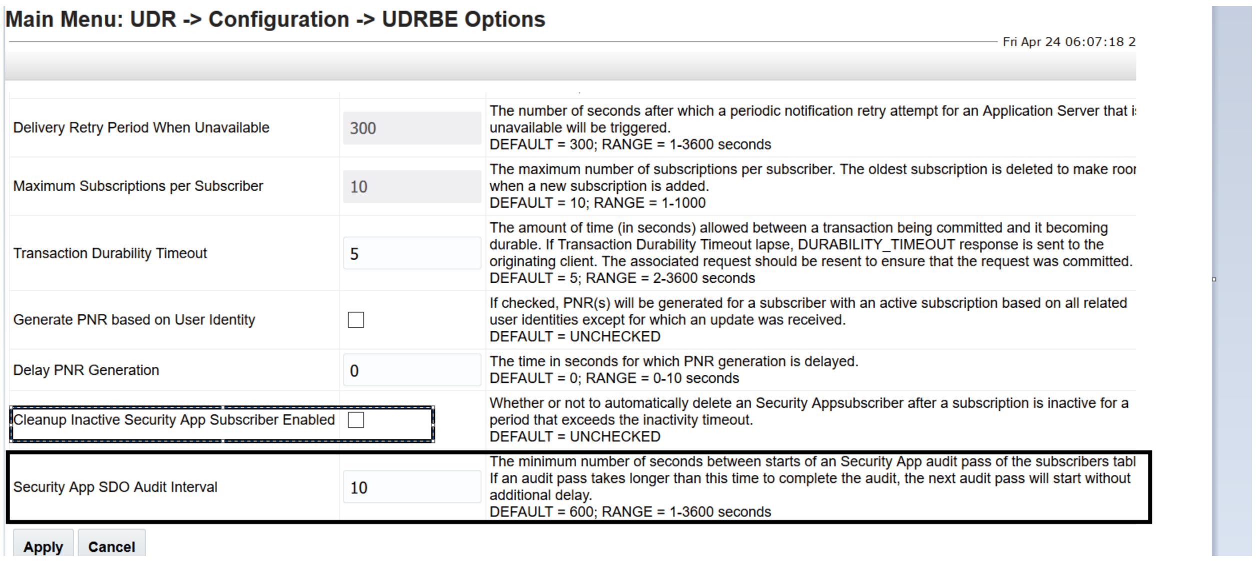
Login to Active NOAM Server through putty session and run this loader (***enableSecurityApp***) with following steps:

- Go to this path *"/usr/TKLC/udr/prod/maint/loaders/upgrade"*
- Execute the "enableSecurityApp" script.
- Reboot the both UDR NOAM server.

6.3.3.3 Audit Time Configuration on Active UDR NOAM

By Default this configuration will disable (unchecked) e.g. no record will be clean up on UDR server.

- if you want to clean old record on UDR ,we need to configure as
- "Cleanup Inactive Security App Subscriber Enabled" is checked (enable) and "Security App SDO Audit Interval" is set value as "10" => all records will be cleared after 10 sec.


Main Menu: UDR -> Configuration -> UDRBE Options

Fri Apr 24 06:07:18 2

Delivery Retry Period When Unavailable	300	The number of seconds after which a periodic notification retry attempt for an Application Server that i unavailable will be triggered. DEFAULT = 300; RANGE = 1-3600 seconds
Maximum Subscriptions per Subscriber	10	The maximum number of subscriptions per subscriber. The oldest subscription is deleted to make roo when a new subscription is added. DEFAULT = 10; RANGE = 1-1000
Transaction Durability Timeout	5	The amount of time (in seconds) allowed between a transaction being committed and it becoming durable. If Transaction Durability Timeout lapse, DURABILITY_TIMEOUT response is sent to the originating client. The associated request should be resent to ensure that the request was committed. DEFAULT = 5; RANGE = 2-3600 seconds
Generate PNR based on User Identity	☐	If checked, PNR(s) will be generated for a subscriber with an active subscription based on all related user identities except for which an update was received. DEFAULT = UNCHECKED
Delay PNR Generation	0	The time in seconds for which PNR generation is delayed. DEFAULT = 0; RANGE = 0-10 seconds
Cleanup Inactive Security App Subscriber Enabled	☐	Whether or not to automatically delete an Security Appsubscriber after a subscription is inactive for a period that exceeds the inactivity timeout. DEFAULT = UNCHECKED
Security App SDO Audit Interval	10	The minimum number of seconds between starts of an Security App audit pass of the subscribers tabl If an audit pass takes longer than this time to complete the audit, the next audit pass will start without additional delay. DEFAULT = 600; RANGE = 1-3600 seconds

Figure Configuration of Audit Time on UDR

6.3.4 Step 3: Define the Configuration Data Schema

“CountULR” does not use any DCA App configuration data.

6.3.5 Step 4: Provision the Configuration Data

“CountULR” does not use any DCA App configuration data.

6.3.6 Step 5: Provision the DCA App Business Logic

The “CountULR” DCA App implements the following business logic:

- When receiving a ULR message, extract the user name from the User-Name AVP and check if a state has been created for the respective user:
  - If the user name is not found, create a state data.
  - If the user name already exists, read the existing state.
- When receiving a CLR message, extract the user name from the User-Name AVP and delete the state corresponding to the respective user, if it exists.

Error! Reference source not found. illustrates a typical call flow. “CountULR” uses two UDR API c alls: createOrRead, BulkDelete . The UDR API is described in section 10.7.

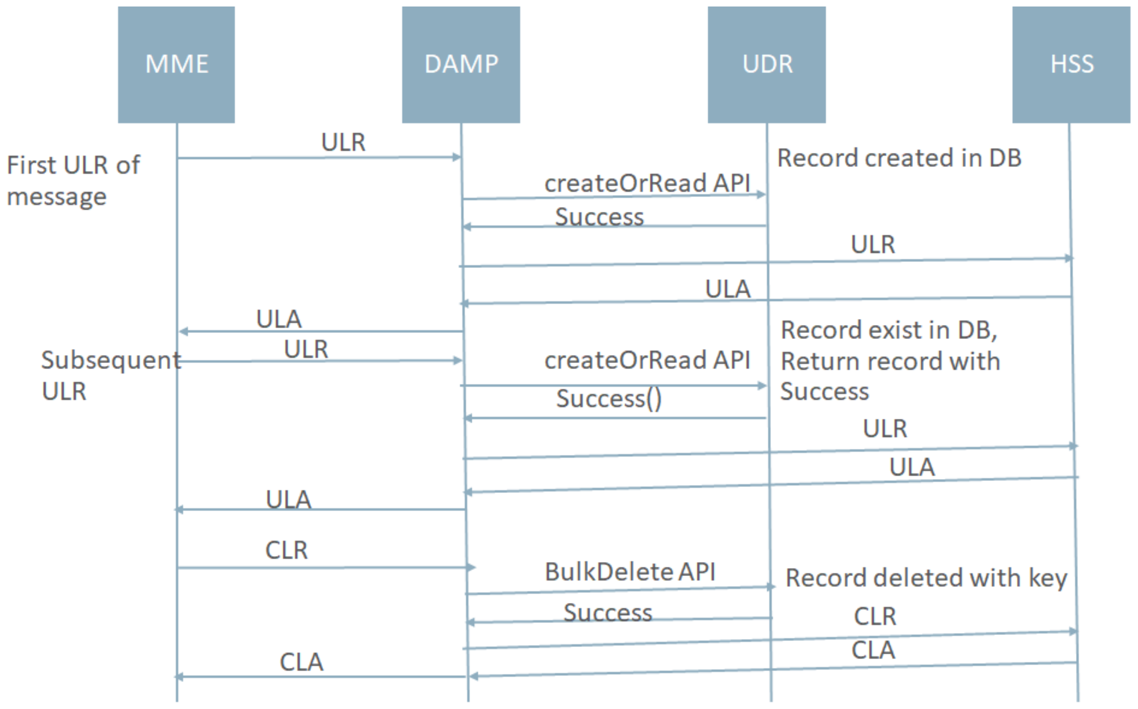


Figure 20: “CountULR” Call Flow

The Perl code is illustrated in **Error! Reference source not found.**.

```
use constant{
        # key types for our app - only NAI is currently used,
        # the others are for exemplification
        IMSI => 0,
        SESSION => 1,
        NAI => 2,
        IPv4 => 3,
        # command codes for S6 commands
        ULR_CMD => 316,
        CLR_CMD => 317,
};

# this function is called when receiving a diameter request
# message
sub process_request{
        # session state to be stored on the udr
        # the session state stores:
        #  - no of requests for this user-name
        #  - no of success replies for this user-name
        #  - no of error replies for this user-name
        my $udr_state =
        {
                requests => 1            # only requests are currently counted
                #ok_replies => 0,
                #err_replies => 0
        };

        # diameter message is the first parameter
        my $param = shift;
        # only one key type for this app: NAI
        my $key_type = NAI;

        # get the diameter message object
        my $msg = diameter::Param::message($param);
        if(!defined($msg)){
                die "Bad diameter message parameter.";
        }

        # try to get the the diameter command code from the diameter message
        my $cmd = diameter::Message::commandCode($msg);
        if(!defined($cmd)){
                die "No command code in diameter message.";
        }

        # get User-Name from the message
        my $user = diameter::Message::getAvpValue($msg,"User-Name");
        if(!defined($user)){
                # could not create $user
                die "Could not get the User-Name value from the message"
        }
         my $xmlData = create_xml_State_data(%udr_state);
        if(ULR_CMD == $cmd){
                # process Update-Location-Request
                # Instantiate and send the "CreateOrRead" UDR stack event
                my $result = dca::udr::udrInstance("GLOBAL_UDR")->createOrRead(
                        IMSI_KEY_TYPE,
                        dca::udr::KeyDataType::STRING, $imsi,
                        dca::udr::StateDataType::STRING, $xmlData,
                        "createOrReadCb");
                # check the "synchronous" error
                if(!defined($result)){
                        # could not create the udr request
                        die "could not create the UDR request";
                }
        }
        elsif(CLR_CMD == $cmd){
                # process Cancel-Location-Request
                # instantiate and send the "Delete" UDR stack event
my $result = dca::udr::udrInstance("GLOBAL_UDR ")-> bulkDelete(IMSI_KEY_TYPE,
                        \@userIds ,
                        "deleteCb");
                # check the "synchronous" error
                if(!defined($result)){
                        # could not create request
```

```
                          die "could not create the UDR request";
                 }
        }
        else{
                 die "unknown diameter command received";
        }
}

# this function is called when receiving a diameter answer
# message
sub process_answer{

}

# this function is called when receiving an DeleteStateResult
# answer from the UDR
sub deleteCb{
        my $udr_code = dca::udr::result()->code();

        if(!defined($udr_code)){
                 # could not get the result code of the UDR answer
                 die "did not get the result code of UDR answer";
        }

        if(dca::udr::ResultCode:: SubscriberNotFound== $udr_code){
                 die "could not find a record with the given key on the UDR";
        }
        elsif( dca::udr::ResultCode::Success != $udr_code){
                 die "UDR error: $udr_code";
        }
}

# this function is called when receiving an CreateOrReadStateResult
# answer from the UDR
sub createOrReadCb
{

         my $udr_code = dca::udr::result()->code();
        my $udr_state = dca::udr::result()->data();

        # diameter message is the first parameter
        my $param = shift;
        # only one key type for this app: NAI
        my $key_type = NAI;

        # get the diameter message object
        my $msg = diameter::Param::message($param);

        if(!defined($msg)){
                 die "Bad diameter message parameter.";
        }

        # get User-Name from the message
        my $user = diameter::Message::getAvpValue($msg,"User-Name");
        if(!defined($user)){
                 # could not create $user
                 die "Could not get the User-Name value from the message"
        }

        if(!defined($udr_code)){
                 # Raise critical alarm ExecutionFailed with error details, Mark a
message as not vulnerable
                 raise_alarm($cmType, REQUEST_MSG, dca::meal::Major,
&UDR_INVALID_RESULT,$transData->[IMSI]);
perform_error_action($cmType, UDR_QUERY_ERROR, $transData->[CMD_CODE]);
        }

        if(!defined($udr_state)){
                # could not get the udr state
                die "did not get the udr state in the UDR answer";
        }

        exec_nxt_CM_for_req_and_exit($msg, ++$priority, $transData);
}
```

Figure 21: "CountULR" Perl Code

6.3.6.1 What does a "state" consist of?

A state is essentially a mapping between a Key and a Value. What exactly the Key and Value are is completely under the DCA App's control. The UDR does not attach any semantics to a DCA App state. In "CountULR" the Key is the user name extracted from the User-Name AVP and the Value is basically a counter that counts the total number of ULR messages.

Even though "CountULR" uses a single Key (of type NAI), DCA Apps may, in general, use multiple Keys (IMSI, MSISDN, IP addresses, Diameter Session-Id, etc.).

A DCA App may distinguish between the different Keys by declaring their Key Types. The Key Type helps avoid collisions like for instance between NAI key "fred" and IPv4 address key 66.72.65.64, or between IP source address key 1.2.3.4 and destination IP address key 1.2.3.4.

The Value associated to a Key is the value of a Perl variable. For "CountULR" the Value is a Perl hash table containing one key "requests" that stores an integer representing the ULR counter. Perl complex data structures like hash tables and arrays are converted to JSON and stored in the UDR DB as strings. When retrieved from the UDR they are converted back to the original data structure. Scalar Perl variables, on the other hand, need not undergo a JSON conversion.

6.3.6.2 What are Asynchronous API Calls and Callbacks?

The `dca::udr::udrInstance("GLOBAL_UDR")→createOrRead`, `dca::udr::udrInstance("GLOBAL_UDR ")→update` and `dca::udr::udrInstance("GLOBAL_UDR ")→bulkDelete` API functions initiate, each of them, an UDR query. They are **asynchronous** functions, in the sense that they do not wait until a response from the UDR is received. They construct the UDR DB query and return immediately, to allow the other Diameter messages to be processed. The query itself is sent after the event handler execution completes.

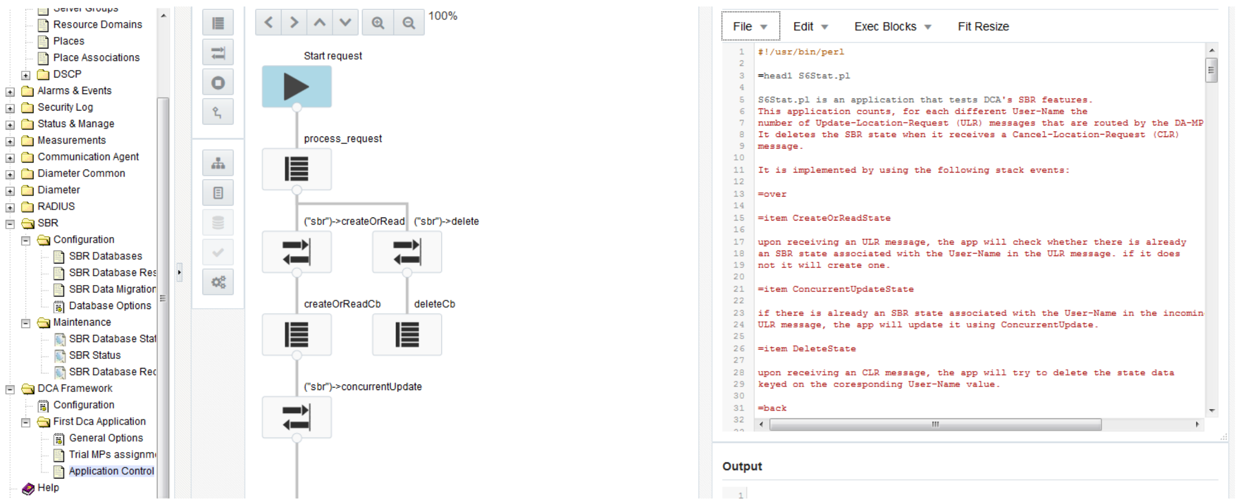
How can then the DCA App learn about the outcome of the UDR DB query it just sent? It may be observed that all the UDR API functions can register, as the last parameter, the name of a **callback** subroutine. The callback subroutine is invoked by the DCA framework when the outcome of the corresponding UDR query is known. The outcome may be: (i) an error condition that prevented the UDR query to even be sent, (ii) the UDR DB response itself or (iii) an error condition indicating that no response has been received within a certain timeout interval.

6.3.6.3 How is the UDR state returned to the Perl script?

In the callback subroutine the DCA App programmer can use the `dca::udr::result()` class to retrieve the error code and, if the query was successful, the result.

6.3.7 Step 6: Render the Flow Control Chart

Render the Flow Control Chart based on the Perl script. Save the code and check the syntax.



```
#!/usr/bin/perl

=head1 S6Stat.pl

S6Stat.pl is an application that tests DCA's SBR features.
This application counts, for each different User-Name the
number of Update-Location-Request (ULR) messages that are routed by the DA-MP
It deletes the SBR state when it receives a Cancel-Location-Request (CLR)
message.

It is implemented by using the following stack events:

=over

=item CreateOrReadState

upon receiving an ULR message, the app will check whether there is already
an SBR state associated with the User-Name in the ULR message. if it does
not it will create one.

=item ConcurrentUpdateState

if there is already an SBR state associated with the User-Name in the incomin
ULR message, the app will update it using ConcurrentUpdate.

=item DeleteState

upon receiving an CLR message, the app will try to delete the state data
keyed on the coresponding User-Name value.

=back
```

Figure 22: Flow Control Chart

6.3.8 Step 7: Test the DCA App Version

See 3.3.7.

6.3.9 Step 8: Promote the DCA App Version to Production

See 3.3.8.

7 Monitoring a DCA App

Custom MEAL general description – templates and their purpose

The monitoring of the execution of a DCA App is possible by means of the Custom MEAL feature.

The Custom MEAL feature enables a DCA App programmer to define and use measurements, KPIs and events, on demand:

- Measurements are used to count specific events or amounts, as required by the DCA App's business logic. Their historical values measured during specific time intervals and/or on specific hosts are available via reports;
- KPIs display real-time statistics of the measured events or amounts, like for instance average values;
- Events may be triggered automatically when the currently measured values exceed the configured thresholds.
  Alternatively, events may be triggered explicitly from the DCA App code.

The Custom MEAL feature hides most of the complexity of the underlying DSR objects that implement the measurements, KPIs and events by defining a number of four templates, which are designed to implement specific tasks:

- The Counter template – is used to count events. The counter values are available only off-line through the Measurement Reports;
- The Rate template – is most typically used to calculate message rates. It generates KPIs, Measurement Reports and may be used to automatically raise alarms if the configured threshold values are exceeded;
- The Basic template – is used to measure averages or number of elements in a set (e.g. to calculate average size of AVPs, messages or number of users registering/deregistering). It generates KPIs, Measurement Reports and may be used to automatically raise alarms if the configured threshold values are exceeded;
- The Event template – is used to explicitly raise/clear alarms or generate events from the Perl script when specific business logic conditions are detected.

Each of the templates is available in scalar and arrayed format.

We denote by "differentiation" the process of assigning a C-MEAL template instance to a DCA App. We denote by "un-differentiation" the reverse process of removing a C-MEAL from a DCA App and basically returning it to the pool of un-differentiated C-MEAL, from where it can be re-assigned to another (or even the same) DCA App.

8 A DCA App Using Custom MEALs

Chapter 7 introduced the Custom MEAL (C-MEAL) templates and their applicability.

This chapter describes a simple DCA App that uses a Rate C-MEAL to monitor the rate of the incoming Diameter requests with just two lines of Perl code.

8.1 The „Rate“ DCA App

The “Rate” DCA App differentiates a Rate C-MEAL, initializes it and pegs it every time a Diameter request is received. The operator can monitor the incoming message rate in real time (KPI), check the history of the measured value (measurement report) and get notified when the configured thresholds are exceeded (alarm).

8.2 Prerequisites

The DCA Framework must have been previously activated as described in [1]. Also, a DCA App with the name “Rate” shall be activated as described in [1].

The “Rate” DCA App has to be enabled on all the DA-MPs in the network from the SO **Main Menu: Diameter→Maintenance→Applications**.

An ART rule shall be added that enables Diameter requests to be delivered to the “Rate” DCA App.

8.3 The Process

The following steps must be followed in order to provision the “Rate” DCA App:

Business Logic and Configuration Data Provisioning	**Custom MEAL Configuration**
Step 1: Configure the general options and behavior of the " Rate " DCA App **Step 2:** Create a new development version of the “Rate” DCA App; **Step 3:** Define the structure of tables to store the “Rate” configuration data; **Step 4:** Provision the “Rate” configuration data; **Step 5:** Provision the “Rate” business logic – essentially a Perl script; **Step 6:** Render the Flow Control Chart based on the Perl script. Save and perform syntax checks;	**Step I:** Differentiate a scalar Rate C-MEAL

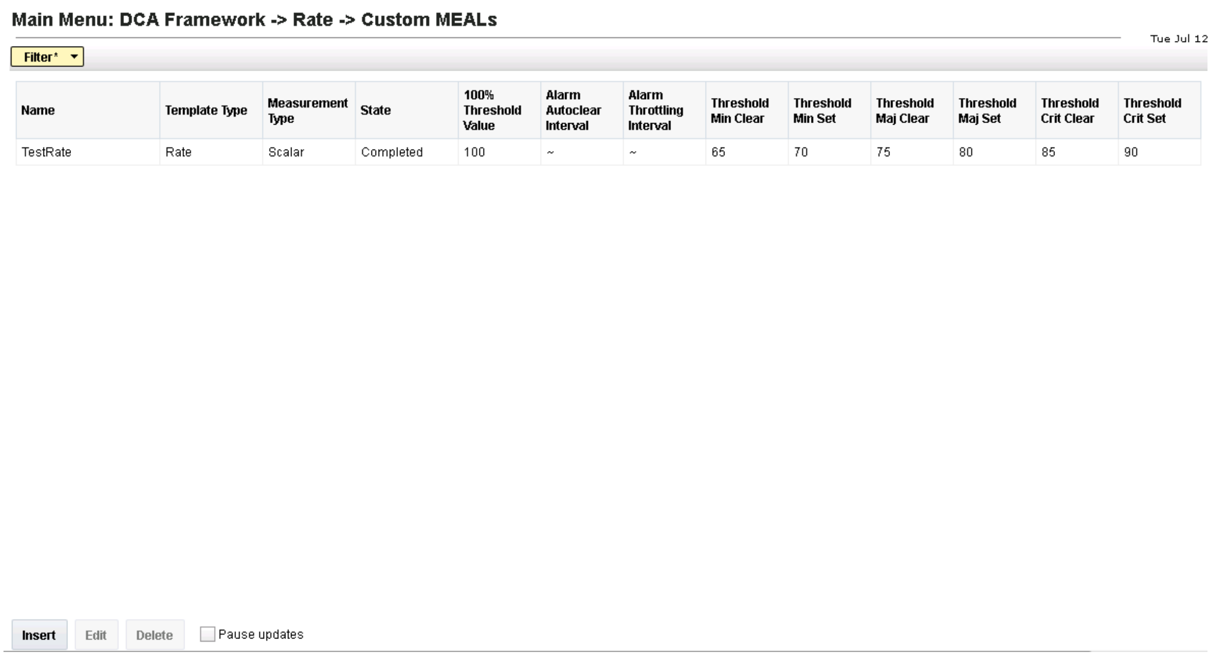
Step 7: Test the “Rate” DCA App: configure the Trial DA-MPs and promote “Rate” to Trial state;
Step 8: Compile “Rate”, promote “Rate” to Production state;	

Steps 1 to 8 are similar to those described in chapter 3. Step I is required in order to assign a C-MEAL to the “Rate” DCA App, which can be then be used via the C-MEAL API, which is described in section 10.6.

Step I may be executed in any order relative to steps 1 to 5.

8.3.1 Step I: Differentiate a C-MEAL

C-MEALs are differentiated from the **Main Menu→DCA Framework→Rate→Custom MEALs** screen, by clicking on the **Insert** button. For the “Rate” DCA App in particular, "TestRate", a scalar rate C-MEAL, will be differentiated (see Figure 23). "TestRate" will raise an alarm when the configured thresholds are exceeded. The threshold values represent percentages from the 100% Threshold Value, which in our example is exactly 100.



Main Menu: DCA Framework -> Rate -> Custom MEALs

Tue Jul 12

Name	Template Type	Measurement Type	State	100% Threshold Value	Alarm Autoclear Interval	Alarm Throttling Interval	Threshold Min Clear	Threshold Min Set	Threshold Maj Clear	Threshold Maj Set	Threshold Crit Clear	Threshold Crit Set
TestRate	Rate	Scalar	Completed	100	~	~	65	70	75	80	85	90

Figure 23 TestRate Differentiation

8.3.2 Step 1: Configure the DCA App's General Options and Behavior

See section 3.3.1.

8.3.3 Step 2: Create a New Development Version

See section 3.3.2

8.3.4 Step 3: Define the Configuration Data Schema

“Rate” does not need any DCA App configuration data.

8.3.5 Step 4: Provision the Configuration Data

“Rate” does not need any DCA App configuration data.

8.3.6 Step 5: Provision the DCA App Business Logic

The “Rate” DCA App implements a simple business logic that consists of pegging the Rate C-MEAL each time a Diameter request is received.

The Perl code is illustrated in Figure 24. Note that the C-MEAL name used to initialize the Perl object must be the same as the one configured for the C-MEAL during differentiation (“TestRate”).

```
my $rateObject = new dca::meal::rate("TestRate");
die "Failed to bind to the rate template" unless $rateObject;
                        # force compilation error if
                        # rateObject initialization fails

# my $eventObject = new dca::meal::event("TestEvent");
# die "Failed to bind to the event template" unless $eventObject

sub process_request{
    $rateObject->peg(); # or `die unless $rateObject->peg();´ to force
                        # a runtime error if pegging fails
}

# And that's it! Alarms will be automatically raised when the configured
# thresholds are exceeded

#
# Alternative version to log an event when pegging fails - un-comment
# eventObject initialization
#
# sub process_request{
#   my $err = $rateObject->peg();
#
#   if (! $err){
#          if (! $eventObject->isThrottled(dca::meal::Minor)){
#                 $err = $eventObject->log(dca::meal::Minor,
#                                                        "Pegging failed");
#          }
#   }
# }
```

Figure 24 The "Rate" DCA App Code

8.3.7 Step 6: Render the Flow Control Chart

The same process described in section 3.3.6 shall be followed.

8.3.8 Step 7: Test the DCA App Version

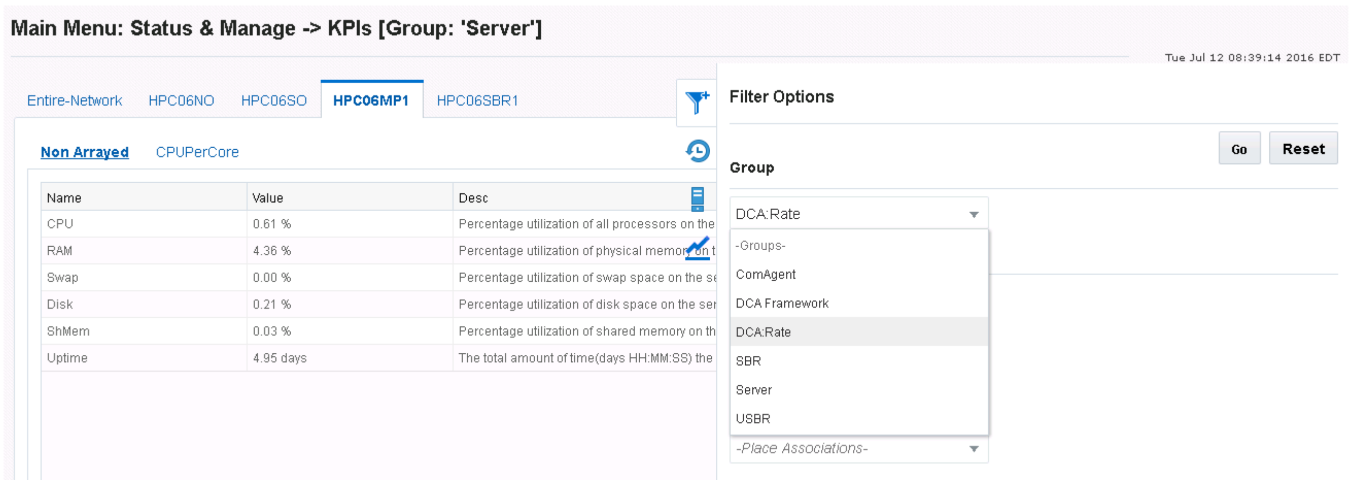
The same process described in 3.3.7 shall be followed.

At this stage, we can finally monitor the "Rate" DCA App in the following ways:

1. The "DCA:Rate" KPI group includes all the KPIs that belong to the "Rate" DCA App. In the **Main Menu→Status&Manage→KPIs** the "DCA:Rate" group shall be included in the KPI filter criteria (see Figure 25). As a result, the exponentially smoothened average of the ingress rate (TestRate) is displayed in real time (see Figure 26).
   The history of the measured values can be accessed from the Main **Menu→Measurements→Report** screen. The "DCA:Rate" measurements group includes all the measurements that belong to the "Rate" DCA App and shall be included in the filtering criteria (see Figure 27). As a result, the history of the TestRate measurements is displayed (see Figure 28).

An alarm with the corresponding severity is raised when the respective threshold values are exceeded. This can be seen for instance in Figure 26. The alarm details can be accessed from **Main Menu→Alarms&Events**.

2. Figure 29 illustrates the alarm history, obtained by progressively increasing the message rate above the critical set threshold and then progressively reducing it below the minor clear threshold.

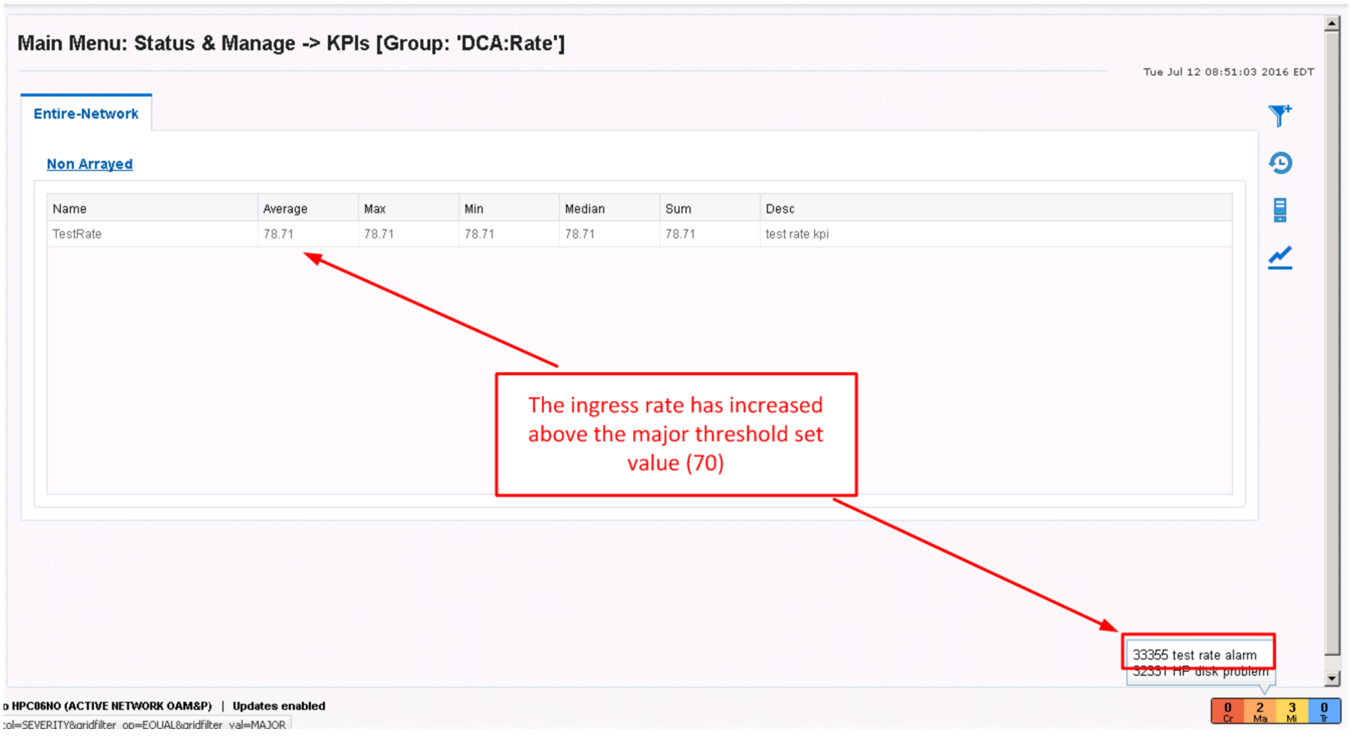


Main Menu: Status & Manage -> KPIs [Group: 'Server']

Tue Jul 12 08:39:14 2016 EDT

Name	Value	Desc
CPU	0.61 %	Percentage utilization of all processors on the
RAM	4.36 %	Percentage utilization of physical memory on t
Swap	0.00 %	Percentage utilization of swap space on the se
Disk	0.21 %	Percentage utilization of disk space on the ser
ShMem	0.03 %	Percentage utilization of shared memory on th
Uptime	4.95 days	The total amount of time(days HH:MM:SS) the

Figure 25 Filter the DCA:Rate KPIs



Main Menu: Status & Manage -> KPIs [Group: 'DCA:Rate']

Tue Jul 12 08:51:03 2016 EDT

Entire-Network
Non Arrayed

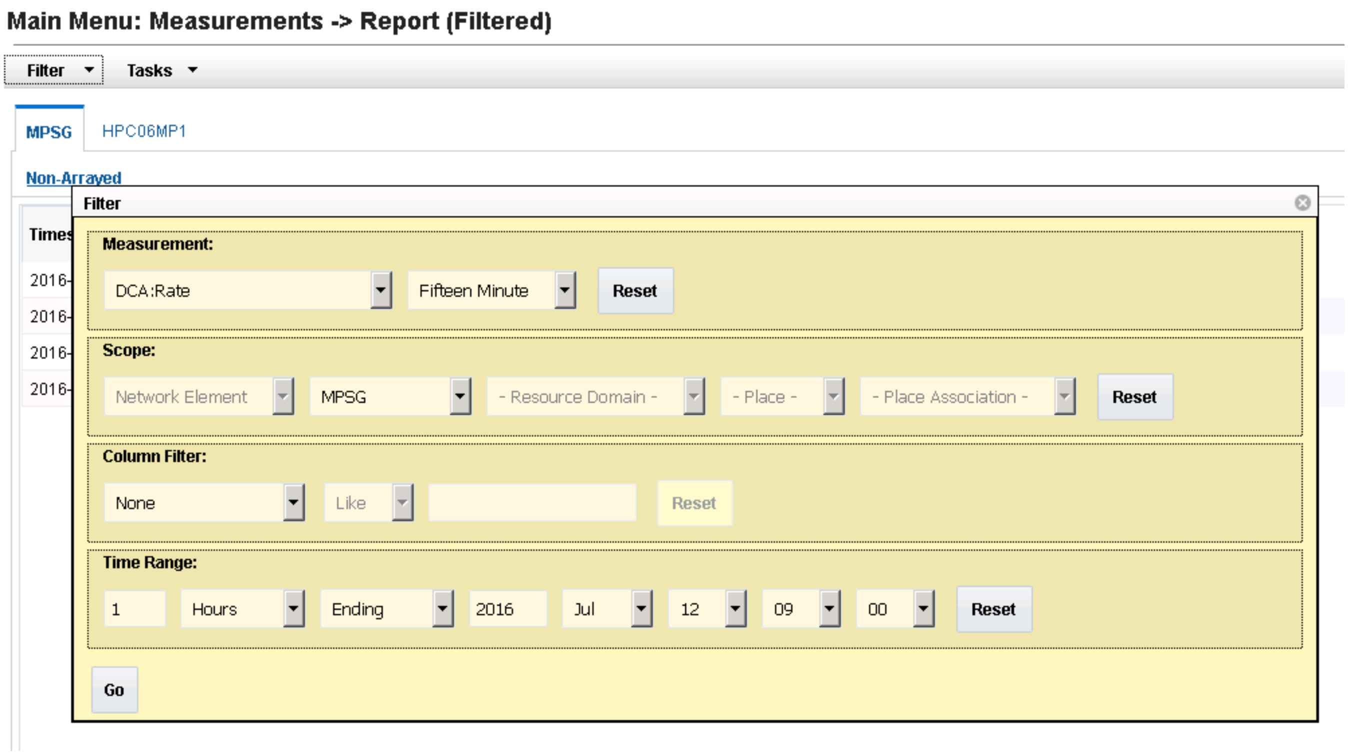
Name	Average	Max	Min	Median	Sum	Desc
TestRate	78.71	78.71	78.71	78.71	78.71	test rate kpi

The ingress rate has increased above the major threshold set value (70)

33355 test rate alarm
32331 HP disk problem

o HPC06NO (ACTIVE NETWORK OAM&P) | Updates enabled
:ol=SEVERITY&gridfilter_op=EQUAL&gridfilter_val=MAJOR
0 Cr
2 Ma
3 Mi
0 Tr

Figure 26 Display TestRate KPI


Main Menu: Measurements -> Report (Filtered)

Filter | Tasks

MPSG | HPC06MP1

Non-Arrayed

Filter

Measurement: DCA:Rate | Fifteen Minute | Reset

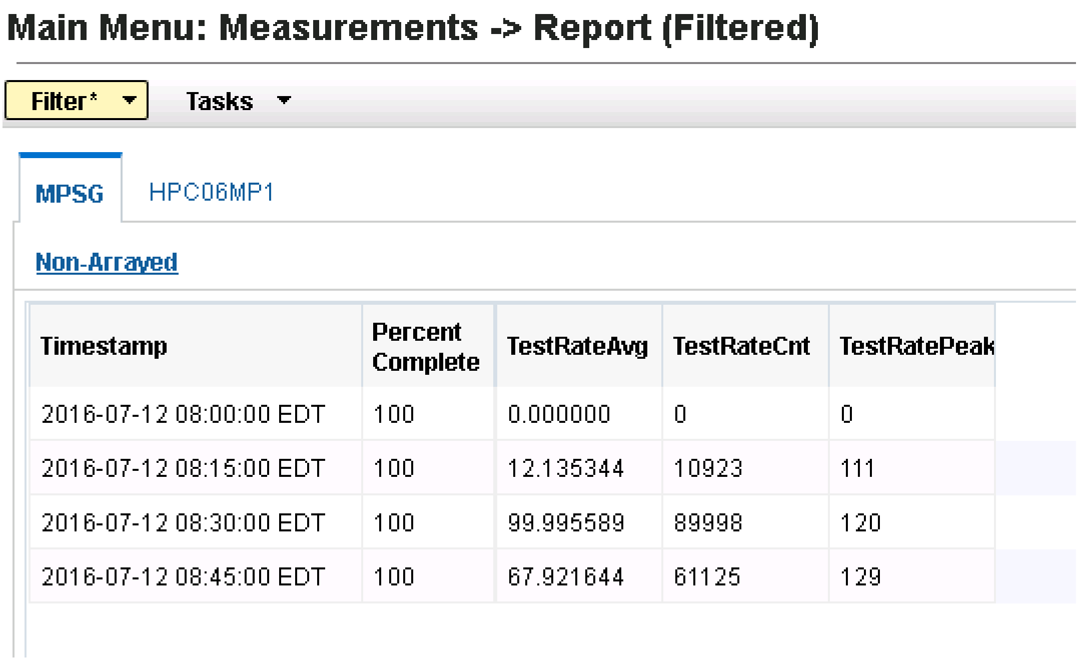
Scope: Network Element | MPSG | - Resource Domain - | - Place - | - Place Association - | Reset

Column Filter: None | Like | | Reset

Time Range: 1 | Hours | Ending | 2016 | Jul | 12 | 09 | 00 | Reset

Go

Figure 27 Filter the DCA:Rate Measurements


Main Menu: Measurements -> Report (Filtered)

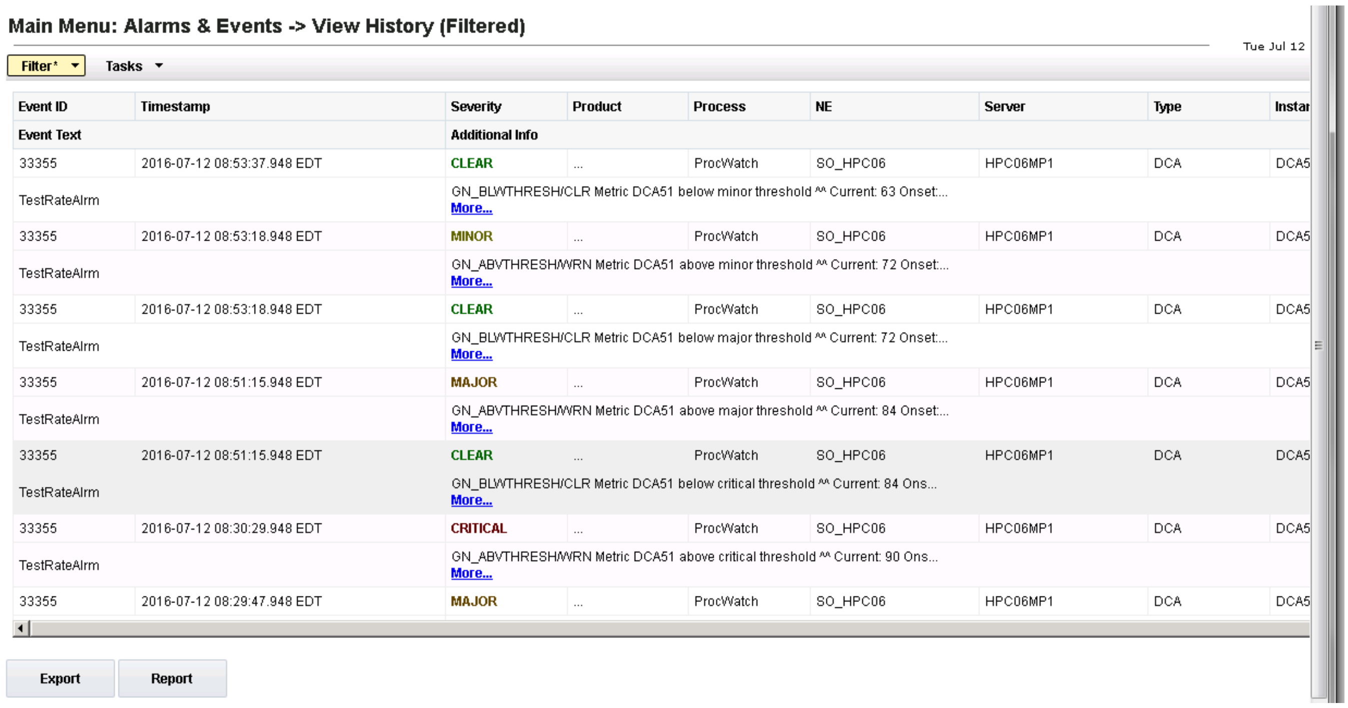
Filter* | Tasks

MPSG | HPC06MP1

Non-Arrayed

Timestamp	Percent Complete	TestRateAvg	TestRateCnt	TestRatePeak
2016-07-12 08:00:00 EDT	100	0.000000	0	0
2016-07-12 08:15:00 EDT	100	12.135344	10923	111
2016-07-12 08:30:00 EDT	100	99.995589	89998	120
2016-07-12 08:45:00 EDT	100	67.921644	61125	129

Figure 28 Display the TestRate measurements



Main Menu: Alarms & Events -> View History (Filtered)

Tue Jul 12

Filter* Tasks

Event ID	Timestamp	Severity	Product	Process	NE	Server	Type	Instar
Event Text		Additional Info						
33355	2016-07-12 08:53:37.948 EDT	CLEAR	...	ProcWatch	SO_HPC06	HPC06MP1	DCA	DCA5
TestRateAlrm		GN_BLWTHRESH/CLR Metric DCA51 below minor threshold ^^ Current: 63 Onset:... More...						
33355	2016-07-12 08:53:18.948 EDT	MINOR	...	ProcWatch	SO_HPC06	HPC06MP1	DCA	DCA5
TestRateAlrm		GN_ABVTHRESH/WRN Metric DCA51 above minor threshold ^^ Current: 72 Onset:... More...						
33355	2016-07-12 08:53:18.948 EDT	CLEAR	...	ProcWatch	SO_HPC06	HPC06MP1	DCA	DCA5
TestRateAlrm		GN_BLWTHRESH/CLR Metric DCA51 below major threshold ^^ Current: 72 Onset:... More...						
33355	2016-07-12 08:51:15.948 EDT	MAJOR	...	ProcWatch	SO_HPC06	HPC06MP1	DCA	DCA5
TestRateAlrm		GN_ABVTHRESH/WRN Metric DCA51 above major threshold ^^ Current: 84 Onset:... More...						
33355	2016-07-12 08:51:15.948 EDT	CLEAR	...	ProcWatch	SO_HPC06	HPC06MP1	DCA	DCA5
TestRateAlrm		GN_BLWTHRESH/CLR Metric DCA51 below critical threshold ^^ Current: 84 Ons... More...						
33355	2016-07-12 08:30:29.948 EDT	CRITICAL	...	ProcWatch	SO_HPC06	HPC06MP1	DCA	DCA5
TestRateAlrm		GN_ABVTHRESH/WRN Metric DCA51 above critical threshold ^^ Current: 90 Ons... More...						
33355	2016-07-12 08:29:47.948 EDT	MAJOR	...	ProcWatch	SO_HPC06	HPC06MP1	DCA	DCA5

Export Report

Figure 29 TestRate Alarm History

8.3.9 Step 8: Promote the DCA App Version to Production

The same process described in 3.3.8 shall be followed.

9 GUI Overview

9.1 NO/SO differences

Table 1: NO/SO GUI differences

NO	SO
Framework Configuration	Read-only
General Options	Read-only
Custom MEALs	Read-only
Trial MP Assignment	Read-only
New application versions are created	-
Existing application versions are copied	-
Business Logic and/or NO Config data imported/exported	SO Config data imported/exported
Flowchart and Script Development	Read-only
Application version state transitions	Read-only
Defining the configuration tables (schema)	Read-only
Provisioning NO Configuration Data (table content)	Provisioning SO Configuration Data (table content) NO configuration read-only.
-	System Options

9.2 NO Screens

The DCA Framework left hand menu on the NO includes the following screens:

- Configuration Screen

Each activated application is represented by the separate menu folder with the given application name. The application folder on the NO includes the following screens ("Application Control" screen contains the buttons that lead to other DCA screens):

- Custom Meals
- General Options Screen
- Trial MPs Assignment Screen
- Application Control Screen
    - ❑ Create New Development Screen
    - ❑ Copy to New Development Screen
    - ❑ Import Pop-Up Window
    - ❑ Export Pop-Up Window
    - ❑ Development Environment
    - ❑ Tables Screen

- Provision Tables Screen

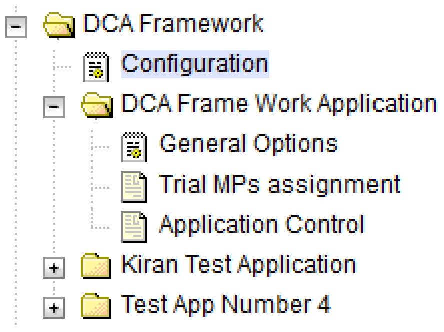
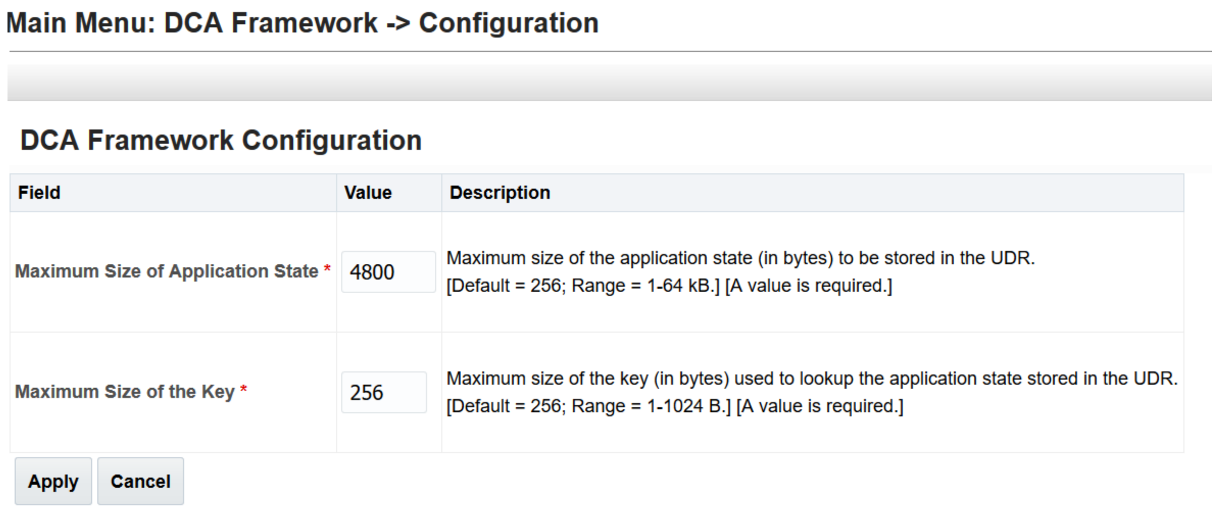


Figure 30: NO Screens

9.2.1 Configuration Screen

The NO **Main Menu→ DCA Framework→ Configuration** screen allows configuring DCA Framework parameters: “Maximum Size of Application State” and “Maximum Size of the Key”. See Figure 31.



Main Menu: DCA Framework -> Configuration

DCA Framework Configuration

Field	Value	Description
Maximum Size of Application State *	4800	Maximum size of the application state (in bytes) to be stored in the UDR. [Default = 256; Range = 1-64 kB.] [A value is required.]
Maximum Size of the Key *	256	Maximum size of the key (in bytes) used to lookup the application state stored in the UDR. [Default = 256; Range = 1-1024 B.] [A value is required.]

Figure 31: NO Configuration Screen

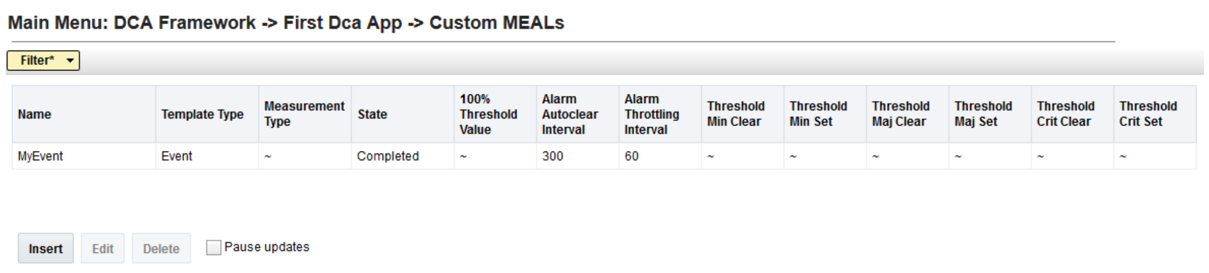
9.2.2 Custom MEALs

9.2.2.1 View Custom MEALs

The NO **Main Menu: DCA Framework→<DCA App Name>→Custom MEALs** screen (illustrated in Figure 32) lists the Custom MEAL templates differentiated for the current DCA App. It also enables new Custom MEAL templates to be differentiated and differentiated Custom MEAL templates to be modified.

There are a limited number of Custom MEAL templates of each type for all the DCA Apps activated in a network. An error will be displayed if the DCA App programmer attempts to exceed these limits.

It is not possible to modify the counter/basic/rate/event and scalar/arrayed type of a differentiated Custom MEAL template. If the type needs to be modified, then a new Custom MEAL template shall be created (provided the limits haven't been exceeded yet) and the old one shall be deleted.

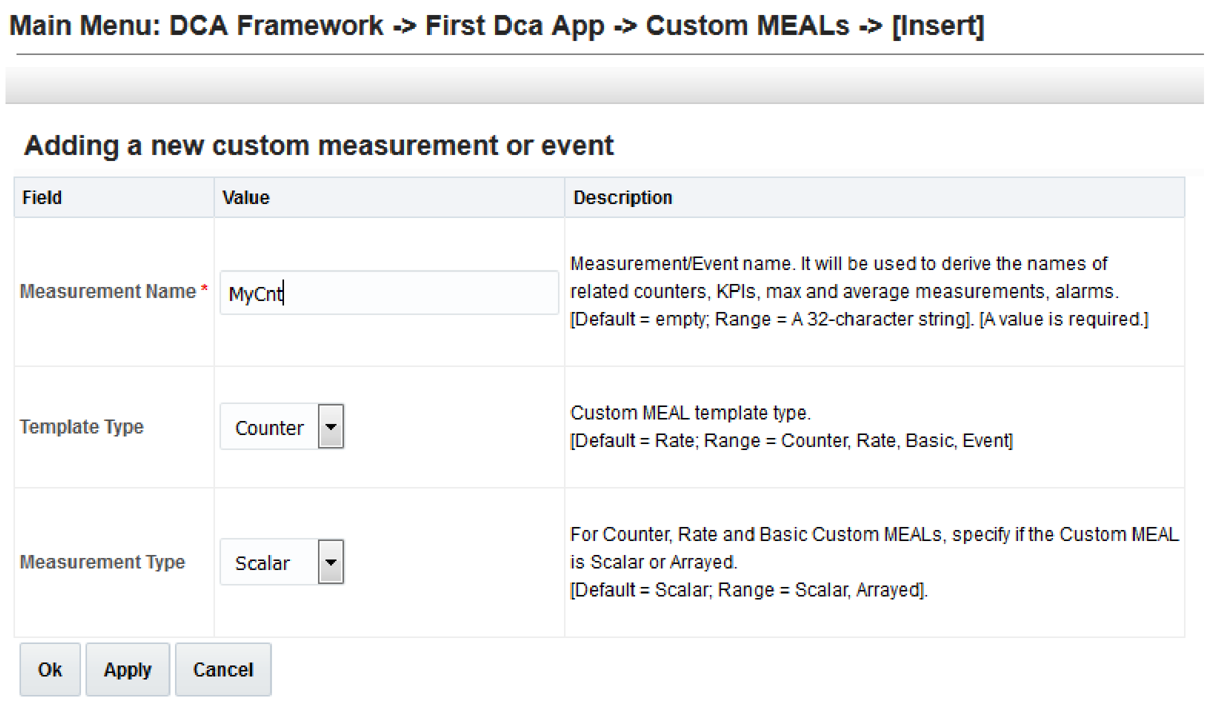

Main Menu: DCA Framework -> First Dca App -> Custom MEALs

Name	Template Type	Measurement Type	State	100% Threshold Value	Alarm Autoclear Interval	Alarm Throttling Interval	Threshold Min Clear	Threshold Min Set	Threshold Maj Clear	Threshold Maj Set	Threshold Crit Clear	Threshold Crit Set
MyEvent	Event	~	Completed	~	300	60	~	~	~	~	~	~

Figure 32 The Custom MEAL View Screen

9.2.2.2 Configure the Counter Custom MEAL Template

Figure 33 illustrates the configuration options for inserting a Counter template.


Main Menu: DCA Framework -> First Dca App -> Custom MEALs -> [Insert]

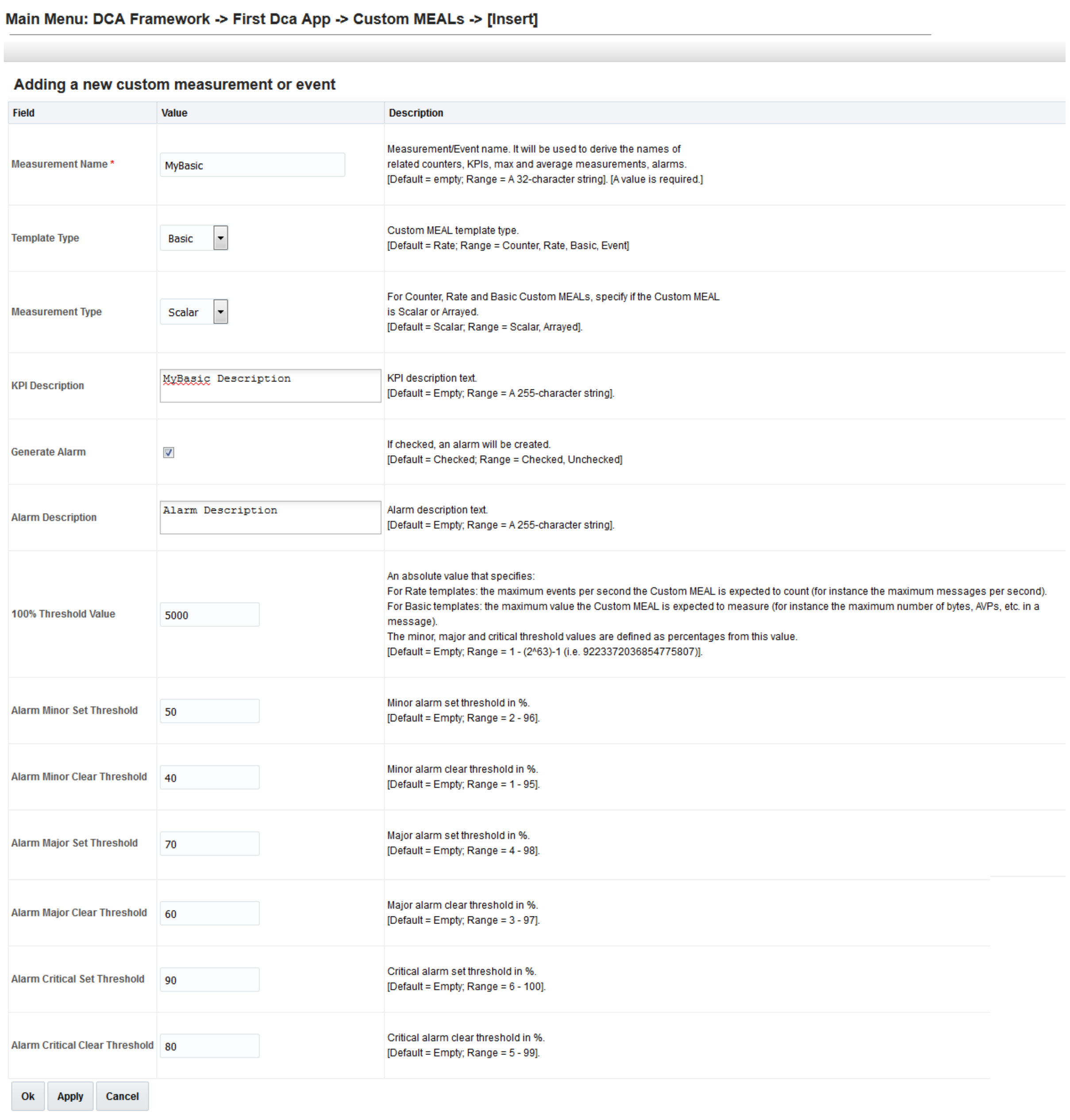
Adding a new custom measurement or event

Field	Value	Description
Measurement Name *	MyCnt	Measurement/Event name. It will be used to derive the names of related counters, KPIs, max and average measurements, alarms. [Default = empty; Range = A 32-character string]. [A value is required.]
Template Type	Counter	Custom MEAL template type. [Default = Rate; Range = Counter, Rate, Basic, Event]
Measurement Type	Scalar	For Counter, Rate and Basic Custom MEALs, specify if the Custom MEAL is Scalar or Arrayed. [Default = Scalar; Range = Scalar, Arrayed].

Figure 33 The Counter Template Configuration Screen

9.2.2.3 Configure the Basic Custom MEAL Template

Figure 34 illustrates the configuration options for inserting a Basic template. The Basic template is optionally associated with an alarm which will be automatically raised if the configured thresholds are exceeded.



Main Menu: DCA Framework -> First Dca App -> Custom MEALs -> [Insert]

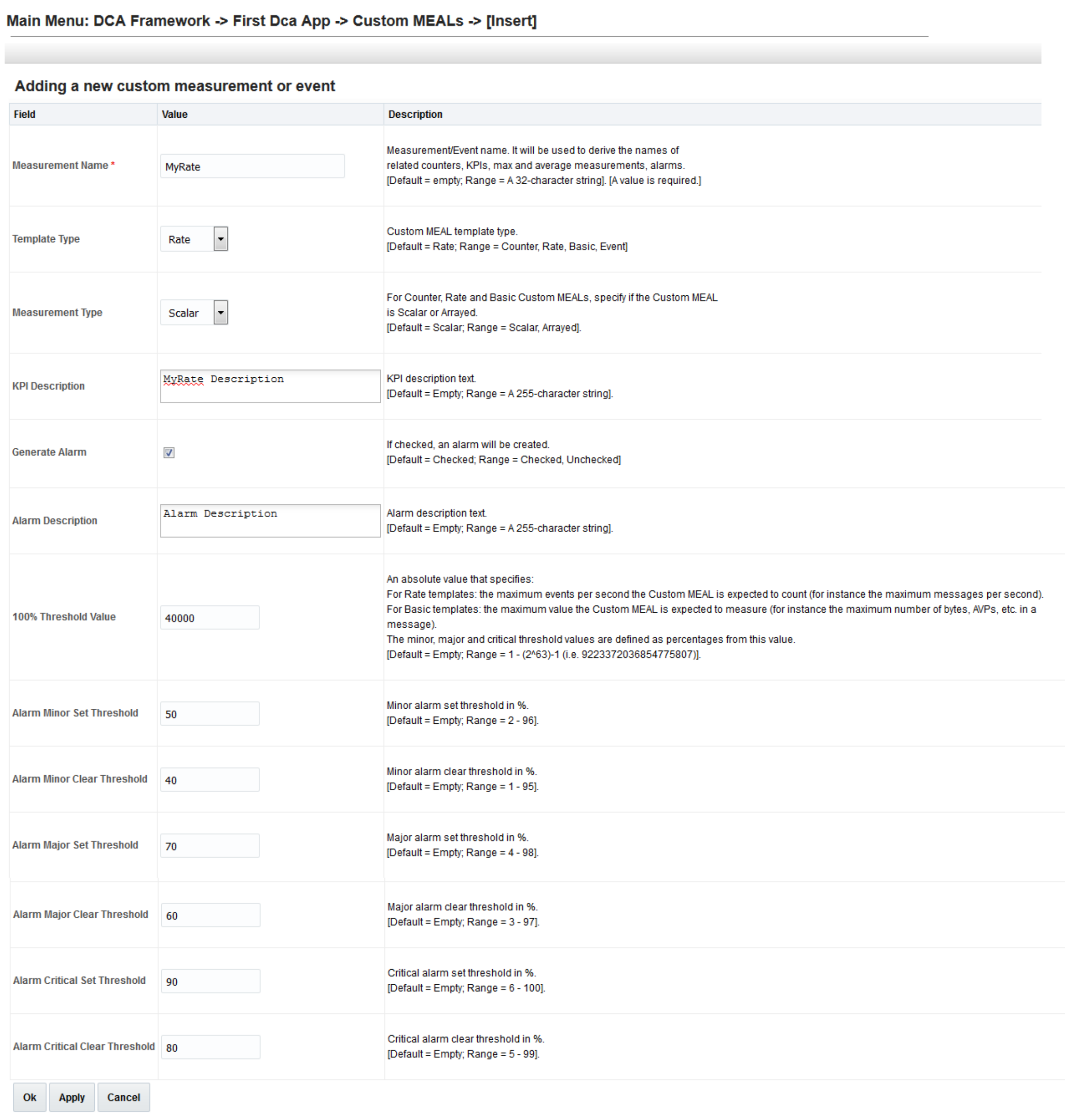
Adding a new custom measurement or event

Field	Value	Description
Measurement Name *	MyBasic	Measurement/Event name. It will be used to derive the names of related counters, KPIs, max and average measurements, alarms. [Default = empty; Range = A 32-character string]. [A value is required.]
Template Type	Basic	Custom MEAL template type. [Default = Rate; Range = Counter, Rate, Basic, Event]
Measurement Type	Scalar	For Counter, Rate and Basic Custom MEALs, specify if the Custom MEAL is Scalar or Arrayed. [Default = Scalar; Range = Scalar, Arrayed].
KPI Description	MyBasic Description	KPI description text. [Default = Empty; Range = A 255-character string].
Generate Alarm	✓	If checked, an alarm will be created. [Default = Checked; Range = Checked, Unchecked]
Alarm Description	Alarm Description	Alarm description text. [Default = Empty; Range = A 255-character string].
100% Threshold Value	5000	An absolute value that specifies: For Rate templates: the maximum events per second the Custom MEAL is expected to count (for instance the maximum messages per second). For Basic templates: the maximum value the Custom MEAL is expected to measure (for instance the maximum number of bytes, AVPs, etc. in a message). The minor, major and critical threshold values are defined as percentages from this value. [Default = Empty; Range = 1 - (2^63)-1 (i.e. 9223372036854775807)].
Alarm Minor Set Threshold	50	Minor alarm set threshold in %. [Default = Empty; Range = 2 - 96].
Alarm Minor Clear Threshold	40	Minor alarm clear threshold in %. [Default = Empty; Range = 1 - 95].
Alarm Major Set Threshold	70	Major alarm set threshold in %. [Default = Empty; Range = 4 - 98].
Alarm Major Clear Threshold	60	Major alarm clear threshold in %. [Default = Empty; Range = 3 - 97].
Alarm Critical Set Threshold	90	Critical alarm set threshold in %. [Default = Empty; Range = 6 - 100].
Alarm Critical Clear Threshold	80	Critical alarm clear threshold in %. [Default = Empty; Range = 5 - 99].

Figure 34 The Basic Template Configuration Screen

9.2.2.4 Configure the Rate Custom MEAL Template

Figure 35 illustrates the configuration options for inserting a Rate template. The Rate template is optionally associated with an alarm which will be automatically raised if the configured thresholds are exceeded.



Main Menu: DCA Framework -> First Dca App -> Custom MEALs -> [Insert]

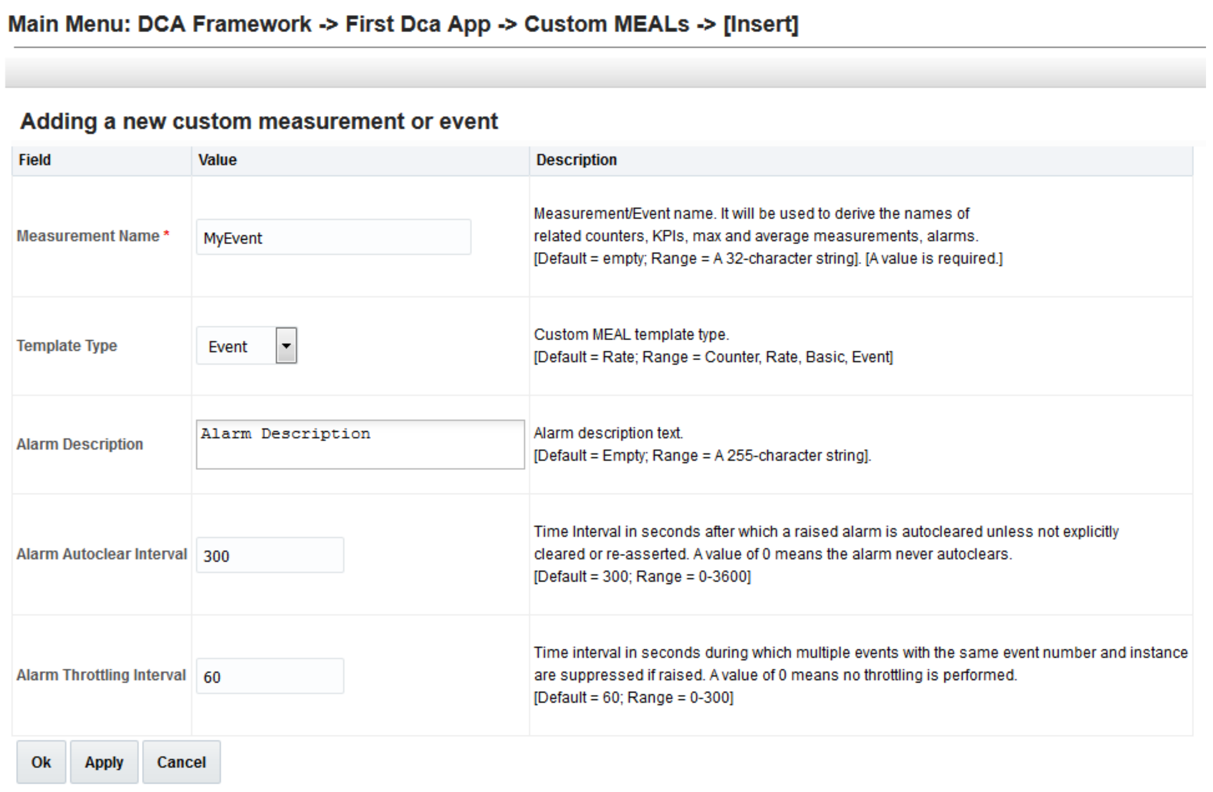
Adding a new custom measurement or event

Field	Value	Description
Measurement Name *	MyRate	Measurement/Event name. It will be used to derive the names of related counters, KPIs, max and average measurements, alarms. [Default = empty; Range = A 32-character string]. [A value is required.]
Template Type	Rate	Custom MEAL template type. [Default = Rate; Range = Counter, Rate, Basic, Event]
Measurement Type	Scalar	For Counter, Rate and Basic Custom MEALs, specify if the Custom MEAL is Scalar or Arrayed. [Default = Scalar; Range = Scalar, Arrayed].
KPI Description	MyRate Description	KPI description text. [Default = Empty; Range = A 255-character string].
Generate Alarm	✓	If checked, an alarm will be created. [Default = Checked; Range = Checked, Unchecked]
Alarm Description	Alarm Description	Alarm description text. [Default = Empty; Range = A 255-character string].
100% Threshold Value	40000	An absolute value that specifies: For Rate templates: the maximum events per second the Custom MEAL is expected to count (for instance the maximum messages per second). For Basic templates: the maximum value the Custom MEAL is expected to measure (for instance the maximum number of bytes, AVPs, etc. in a message). The minor, major and critical threshold values are defined as percentages from this value. [Default = Empty; Range = 1 - (2^63)-1 (i.e. 9223372036854775807)].
Alarm Minor Set Threshold	50	Minor alarm set threshold in %. [Default = Empty; Range = 2 - 96].
Alarm Minor Clear Threshold	40	Minor alarm clear threshold in %. [Default = Empty; Range = 1 - 95].
Alarm Major Set Threshold	70	Major alarm set threshold in %. [Default = Empty; Range = 4 - 98].
Alarm Major Clear Threshold	60	Major alarm clear threshold in %. [Default = Empty; Range = 3 - 97].
Alarm Critical Set Threshold	90	Critical alarm set threshold in %. [Default = Empty; Range = 6 - 100].
Alarm Critical Clear Threshold	80	Critical alarm clear threshold in %. [Default = Empty; Range = 5 - 99].

Figure 35 The Rate Template Configuration Screen

9.2.2.5 Configure the Event Custom MEAL Template

Figure 36 illustrates the configuration options for inserting an Event template.



Main Menu: DCA Framework -> First Dca App -> Custom MEALs -> [Insert]

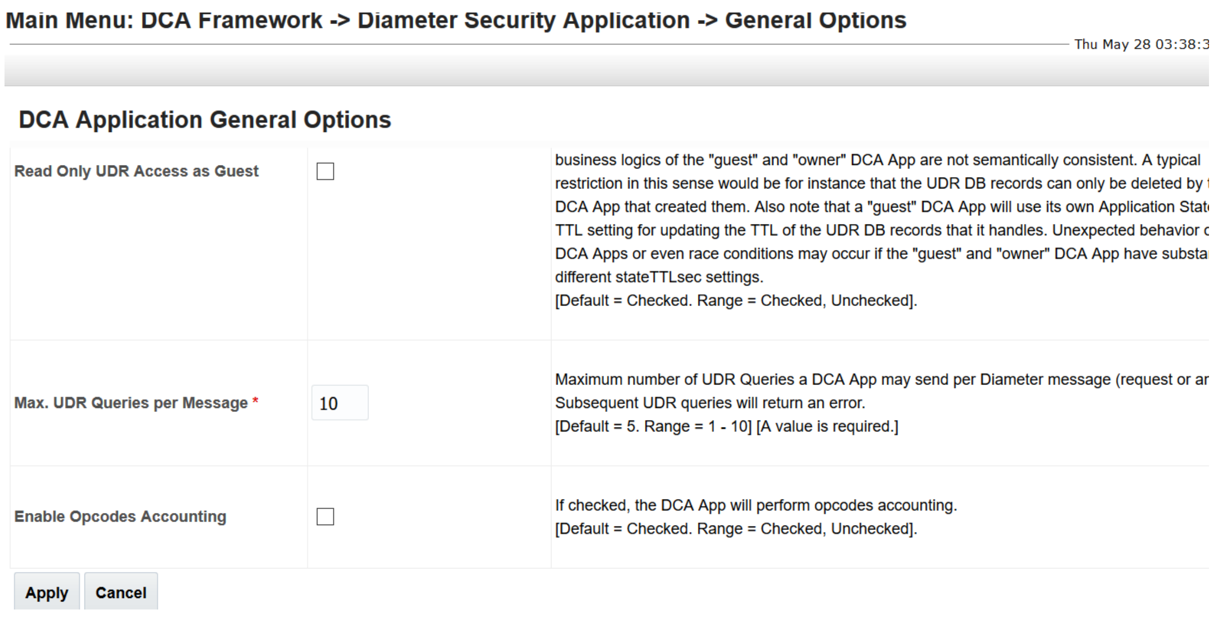
Adding a new custom measurement or event

Field	Value	Description
Measurement Name *	MyEvent	Measurement/Event name. It will be used to derive the names of related counters, KPIs, max and average measurements, alarms. [Default = empty; Range = A 32-character string]. [A value is required.]
Template Type	Event	Custom MEAL template type. [Default = Rate; Range = Counter, Rate, Basic, Event]
Alarm Description	Alarm Description	Alarm description text. [Default = Empty; Range = A 255-character string].
Alarm Autoclear Interval	300	Time Interval in seconds after which a raised alarm is autocleared unless not explicitly cleared or re-asserted. A value of 0 means the alarm never autoclears. [Default = 300; Range = 0-3600]
Alarm Throttling Interval	60	Time interval in seconds during which multiple events with the same event number and instance are suppressed if raised. A value of 0 means no throttling is performed. [Default = 60; Range = 0-300]

Figure 36 The Event Template Configuration Screen

9.2.3 General Options Screen

The NO **Main Menu→DCA Framework→** < **Application Name>→ General Options** screen enables specifying the Perl Subroutines for Diameter Request and Answer, “Application State Data TTL”, “Read Only UDR Access as Guest” and “Max. UDR Queries per Message”. See Figure 37.



Main Menu: DCA Framework -> Diameter Security Application -> General Options

Thu May 28 03:38:3

DCA Application General Options

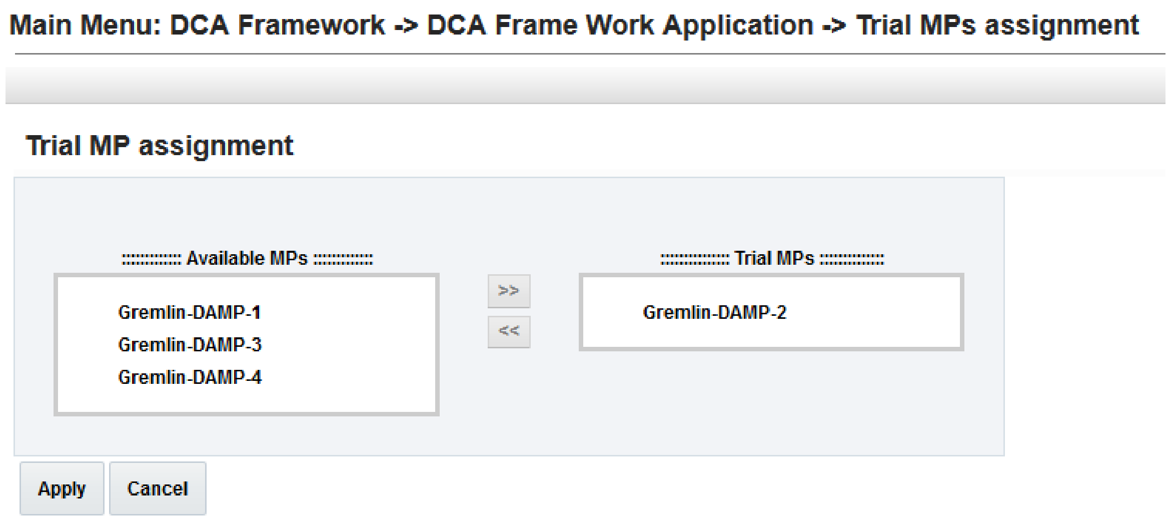
Read Only UDR Access as Guest	☐	business logics of the "guest" and "owner" DCA App are not semantically consistent. A typical restriction in this sense would be for instance that the UDR DB records can only be deleted by t DCA App that created them. Also note that a "guest" DCA App will use its own Application Stat TTL setting for updating the TTL of the UDR DB records that it handles. Unexpected behavior c DCA Apps or even race conditions may occur if the "guest" and "owner" DCA App have substa different stateTTLsec settings. [Default = Checked. Range = Checked, Unchecked].
Max. UDR Queries per Message *	10	Maximum number of UDR Queries a DCA App may send per Diameter message (request or ar Subsequent UDR queries will return an error. [Default = 5. Range = 1 - 10] [A value is required.]
Enable Opcodes Accounting	☐	If checked, the DCA App will perform opcodes accounting. [Default = Checked. Range = Checked, Unchecked].

Figure 37: NO General Options

9.2.4 Trial MPs Assignment Screen

The **NO Main Menu→ DCA Framework→ <DCA App Name>→ Trial MPs Assignment** screen allows specifying which DA-MPs shall run the Trial version of the DCA App (see Figure 38). If there is no Trial version available, the Trial DA-MPs will run the Production version, if there is any available.

If a DCA App version is promoted to the Trial state but no Trial DA-MPs are currently configured assigned, a warning message will be displayed.


Main Menu: DCA Framework -> DCA Frame Work Application -> Trial MPs assignment

Trial MP assignment

:::::::::::: Available MPs ::::::::::::

Gremlin-DAMP-1
Gremlin-DAMP-3
Gremlin-DAMP-4

:::::::::::: Trial MPs ::::::::::::

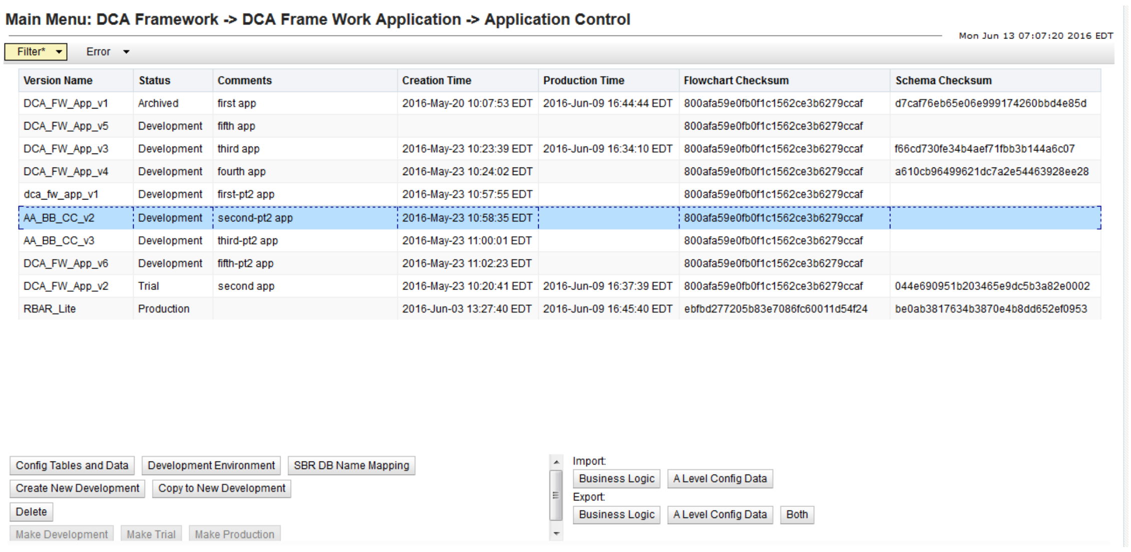
Gremlin-DAMP-2

Figure 38: NO Trial MPs Assignment

9.2.5 Application Control Screen

The NO **Main Menu→DCA Framework→ < Application Name>→Application Control** screen (see Figure 39) allows:

- Listing all application versions configured in the system
- Inserting a new application version (via NO New Development Insert Screen)
- Copying and modifying an existing application version (via NO New Development Copy Screen)
- Exporting an application version entirely (business logic + provisioned data from the NO)
- Exporting only the NO provisioned data of an application version
- Importing a previously exported application version (business logic + NO provisioned data) (via NO Import Pop-Up Window).
- Importing only the NO provisioned data to an existing application version (via NO Import Pop-Up Window)
- Accessing the application version configuration tables (via NO Tables View Screen)
- Accessing business logic and flowchart of an application version (via NO Development Environment Screen)
- Deleting an existing application version
- Changing the status of an application version (Development, Trial, Production, Archived)


Main Menu: DCA Framework -> DCA Frame Work Application -> Application Control

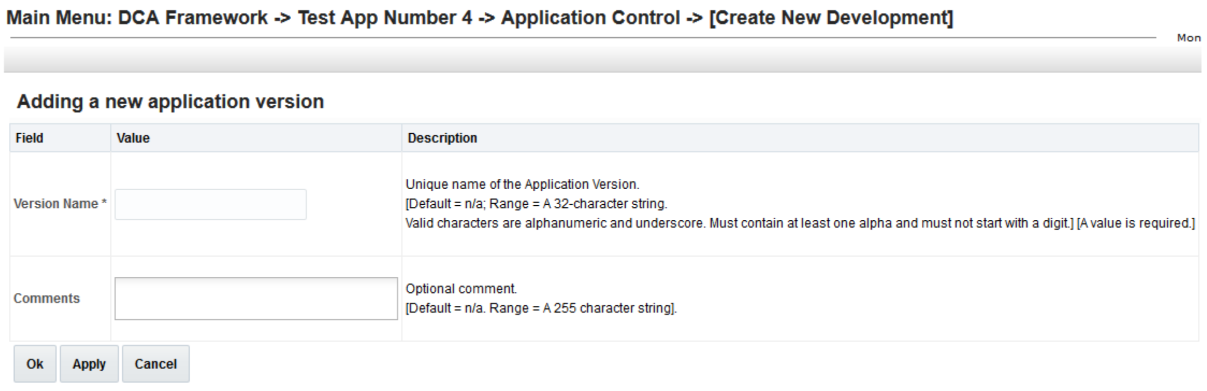
Mon Jun 13 07:07:20 2016 EDT

Version Name	Status	Comments	Creation Time	Production Time	Flowchart Checksum	Schema Checksum
DCA_FW_App_v1	Archived	first app	2016-May-20 10:07:53 EDT	2016-Jun-09 16:44:44 EDT	800afa59e0fb0f1c1562ce3b6279ccaf	d7caf76eb65e06e999174260bbd4e85d
DCA_FW_App_v5	Development	fifth app			800afa59e0fb0f1c1562ce3b6279ccaf	
DCA_FW_App_v3	Development	third app	2016-May-23 10:23:39 EDT	2016-Jun-09 16:34:10 EDT	800afa59e0fb0f1c1562ce3b6279ccaf	f66cd730fe34b4aef71fbb3b144a6c07
DCA_FW_App_v4	Development	fourth app	2016-May-23 10:24:02 EDT		800afa59e0fb0f1c1562ce3b6279ccaf	a610cb96499621dc7a2e54463928ee28
dca_fw_app_v1	Development	first-pt2 app	2016-May-23 10:57:55 EDT		800afa59e0fb0f1c1562ce3b6279ccaf	
AA_BB_CC_v2	Development	second-pt2 app	2016-May-23 10:58:35 EDT		800afa59e0fb0f1c1562ce3b6279ccaf	
AA_BB_CC_v3	Development	third-pt2 app	2016-May-23 11:00:01 EDT		800afa59e0fb0f1c1562ce3b6279ccaf	
DCA_FW_App_v6	Development	fifth-pt2 app	2016-May-23 11:02:23 EDT		800afa59e0fb0f1c1562ce3b6279ccaf	
DCA_FW_App_v2	Trial	second app	2016-May-23 10:20:41 EDT	2016-Jun-09 16:37:39 EDT	800afa59e0fb0f1c1562ce3b6279ccaf	044e690951b203465e9dc5b3a82e0002
RBAR_Lite	Production		2016-Jun-03 13:27:40 EDT	2016-Jun-09 16:45:40 EDT	ebfbd277205b83e7086fc60011d54f24	be0ab3817634b3870e4b8dd652ef0953

Figure 39: NO Application Control

9.2.6 Create New Development Screen

The NO **Main Menu→DCA Framework→<Application Name>→Application Control→ Create New Development** screen allows creating a new DCA App version with a given name and comments. It is accessed by clicking **"Create New Development"** button on the "Application Control" screen, see Figure 40.


Main Menu: DCA Framework -> Test App Number 4 -> Application Control -> [Create New Development]

Mon

Adding a new application version

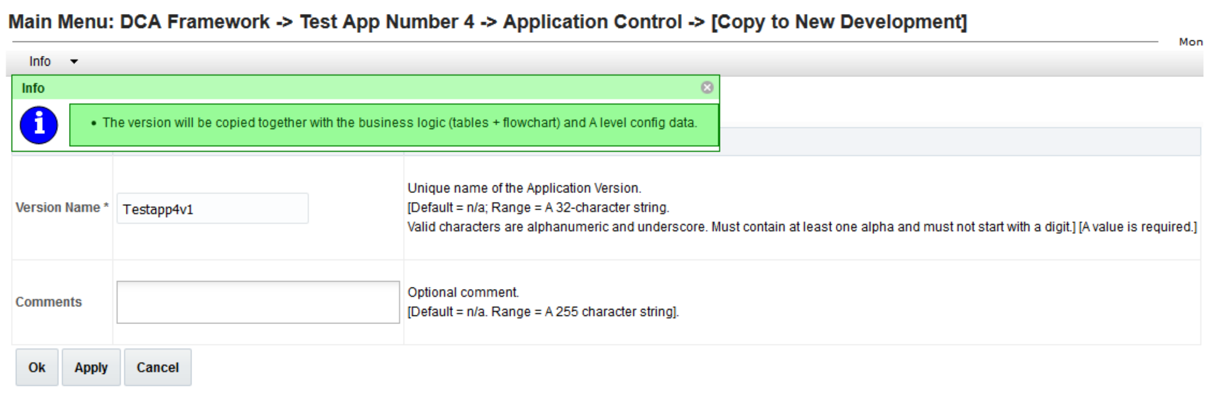
Field	Value	Description
Version Name *		Unique name of the Application Version. [Default = n/a; Range = A 32-character string. Valid characters are alphanumeric and underscore. Must contain at least one alpha and must not start with a digit.] [A value is required.]
Comments		Optional comment. [Default = n/a. Range = A 255 character string].

Figure 40: NO Create New Development Screen

Currently, there might be up to 10 application versions at a time.

9.2.7 Copy to New Development Screen

The NO **Main Menu-→ DCA Framework→< Application Name>→ Application Control→ Copy to New Development** screen allows copying an entire DCA App version, consisting of business logic (Perl script, flowchart and configuration table schemas) and the NO provisioned configuration data, into a new version. It is accessed by selecting the application version and clicking **"Copy to New Development"** button on the "Application Control" screen, see Figure 41.


Main Menu: DCA Framework -> Test App Number 4 -> Application Control -> [Copy to New Development]

Info

- The version will be copied together with the business logic (tables + flowchart) and A level config data.

Version Name *	Testapp4v1	Unique name of the Application Version. [Default = n/a; Range = A 32-character string. Valid characters are alphanumeric and underscore. Must contain at least one alpha and must not start with a digit.] [A value is required.]
Comments		Optional comment. [Default = n/a. Range = A 255 character string].

Figure 41: NO Copy to New Development

When the new Application Version is copied it will become visible on the Application Control screen displaying the user provisioned name in the "Version Name" column and comments in the "Comments" column.

The copied Application will also include the business logic (DB tables + Perl script) and the A level (NO level) configuration data (if any was specified).

9.2.8 Export Pop-Up Window

The exported application version is stored in the form of a JSON file.

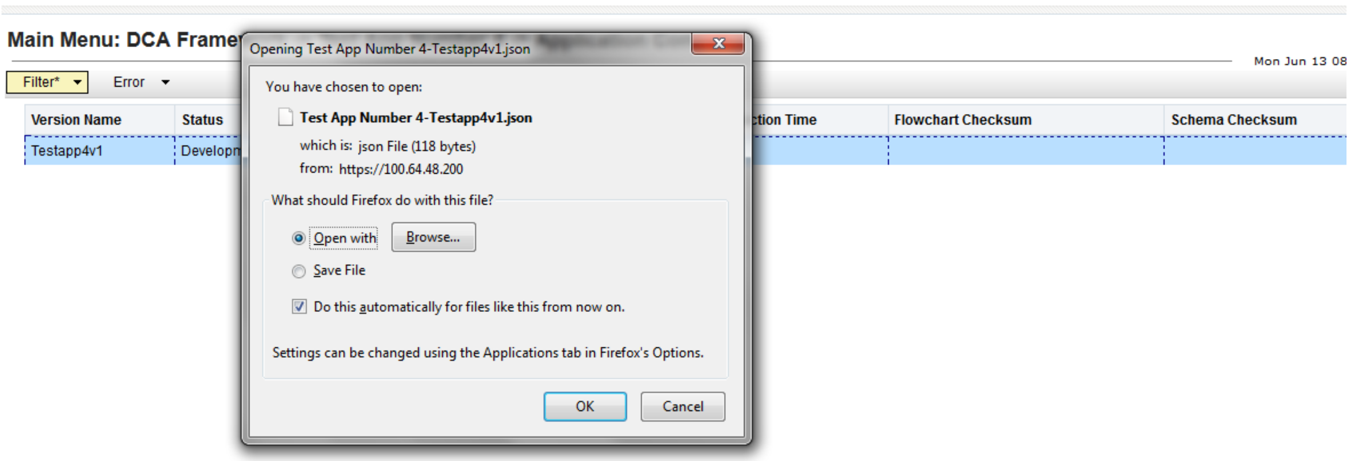
DCA Framework GUI offers three export options:

1. Export the business logic only (that includes the defined tables, flow control chart, the script, custom Meals, KPIs, Events associated with the application version. It does not include the provisioned data)
2. Export the business logic and the configuration data (in addition to the business logic the provisioned data for the tables is also exported)
3. Export the configuration data only

For the first option, select the application version and click "**Export Business Logic**" (becomes enabled when the row is selected).

For the second option, select the application version and click **"Export Both"** (becomes enabled when the row is selected).

For the third option, select the application version and click **"Export A Level Config Data"** (becomes enabled when the row is selected). The export popup window is illustrated in Figure 42.


Main Menu: DCA Frame

Opening Test App Number 4-Testapp4v1.json

You have chosen to open:

Test App Number 4-Testapp4v1.json

which is: json File (118 bytes)

from: https://100.64.48.200

What should Firefox do with this file?

[✓ Open with] Browse...

Save File

[✓] Do this automatically for files like this from now on.

Settings can be changed using the Applications tab in Firefox's Options.

Version Name	Status	ction Time	Flowchart Checksum	Schema Checksum
Testapp4v1	Developn			

Figure 42: NO Export

When the user tries to export the business logic, there is a validation to check whether the flowchart/script has been compiled. If not, the export will be aborted and the error will be given.

The A level (NO level) configuration data can be exported from the NO machine, but not from the SO.

9.2.9 Import Pop-Up Window

The NO Import Pop-Up window allows specifying a JSON file from which the business logic (if required) and the NO provisioned data shall be imported.

Note: The provisioned data imported to the existing business logic shall be appended to the existing data rows.

If the user wants to overwrite the configuration data, it is recommended to first delete all provisioned rows on the “Provision Table” screen and then import the new configuration data.

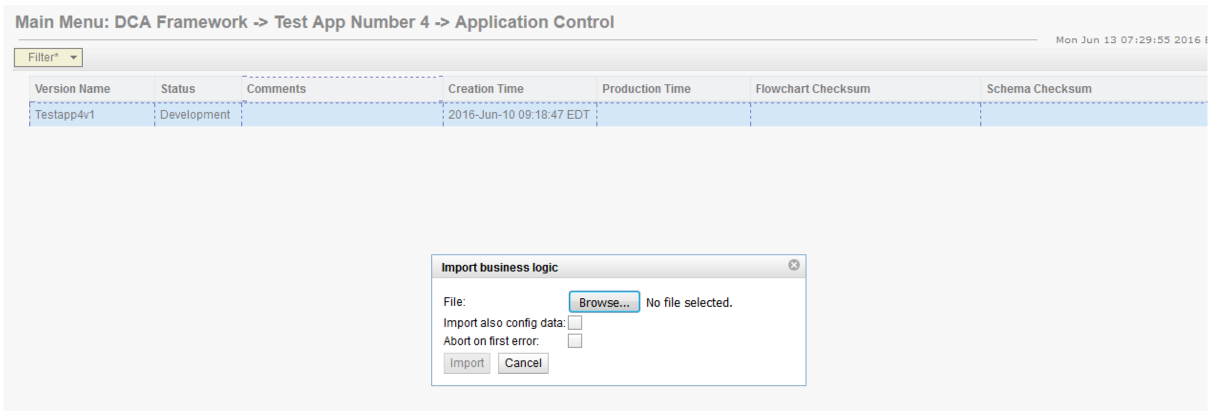
DCA Framework GUI offers three import options:

1. Import the business logic only (that includes the defined tables, flow control chart, the script, custom Meals, KPIs, Events associated with the application version. It does not include the provisioned data import, hence the defined tables are empty after the import)
2. Import the business logic and the configuration data (in addition to the business logic the provisioned data for the tables is also imported)
3. Import the configuration data only

For the first option, click **“Import Business Logic”** (always enabled) on the NO “Application Control” screen. Leave the checkbox “Import also Config data” unchecked, see Figure 43. Select the file.

For the second option, click **“Import Business Logic”** (always enabled) on the NO “Application Control” screen. Check the checkbox “Import also Config data”. Select the file.

For the third option, select the application version and click **“Import A Level Config Data”** (becomes enabled when the row is selected), see Figure 44. Select the file.



Main Menu: DCA Framework -> Test App Number 4 -> Application Control

Mon Jun 13 07:29:55 2016

Version Name	Status	Comments	Creation Time	Production Time	Flowchart Checksum	Schema Checksum
Testapp4v1	Development		2016-Jun-10 09:18:47 EDT			

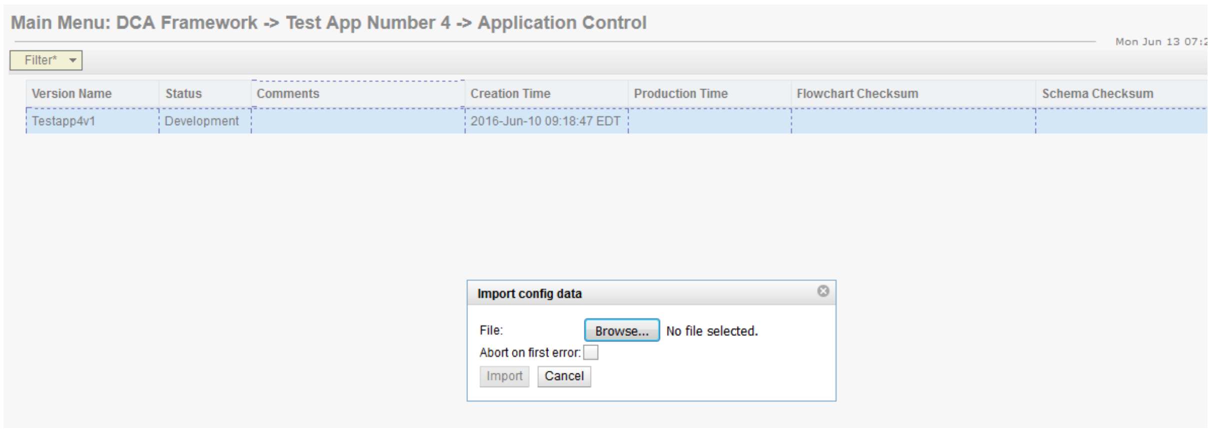
Import business logic

File: Browse... No file selected.

Import also config data: ☐

Abort on first error: ☐

Figure 43: NO Import Business Logic



Main Menu: DCA Framework -> Test App Number 4 -> Application Control

Mon Jun 13 07:

Filter*

Version Name	Status	Comments	Creation Time	Production Time	Flowchart Checksum	Schema Checksum
Testapp4v1	Development		2016-Jun-10 09:18:47 EDT			

Import config data

File: Browse... No file selected.

Abort on first error:

Import Cancel

Figure 44: NO Import Configuration Data

During the import, validations are performed particularly in order to ensure that the format of the DCA App configuration data to be imported is compatible with that of the target DCA App version.

As a result, a number of fatal errors may occur during the import, which will force the import to be aborted regardless of “Abort on first error” checkbox. Such fatal errors are:

- File larger than 25MB;
- File has wrong structure or missing data;
- All the errors that happen during the business logic import;
- If the user tries to import the config data to an existing application version, but none of the table names from the imported file matches the table names of the selected application;
- If the user tries to import the config data to an existing application version, but none of the field names in the tables from the imported file matches the field names in the tables of the selected application;
- Level mismatch. A -level DCA Application configuration data can be imported only on the A level. The same applies to the B level data.

Non-fatal errors, on the other hand, let the user decide whether to abort the import or not (depending on the value of “Abort on first error” checkbox).

9.2.10 Development Environment

Development Environment is accessed by selecting the application version and clicking the “Development Environment” button on the Application Control screen. The DCA Development Environment (DCA-DE) is where a custom Diameter application developer can edit, save, check syntax, and compile the application code for a Diameter Custom Application.

See [1] for more details.

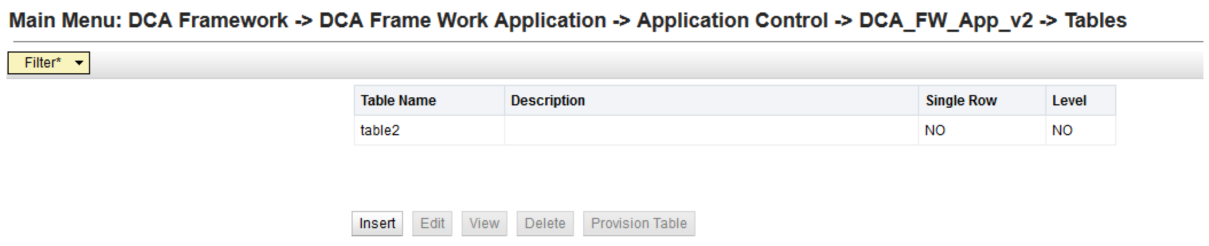
9.2.11 Tables Screen

The NO **Main Menu→ DCA Framework→ < Application Name>→ Application Control→<Version Name>→ Tables View** screen (see Figure 45) allows

- Listing all the config tables (NO+SO) defined for an application version
- Inserting/editing a new config table (NO or SO) for the development or trial application version (via NO Table Insert/Edit Screen).
- Deleting an existing config table (NO or SO) for the development or trial application version

- Viewing an existing config table of an archived or production application version (via NO Table View Screen).
- Accessing the Provision Table View and Insert/Edit screens (via NO Provision Table View Screen, NO Provision Table Insert Screen and NO Provision Table Edit Screen).

"Tables View" screen is accessed by selecting the application version and clicking the **"Config Tables and Data"** button on the "Application Control" screen.



Main Menu: DCA Framework -> DCA Frame Work Application -> Application Control -> DCA_FW_App_v2 -> Tables

Table Name	Description	Single Row	Level
table2		NO	NO

Figure 45: NO Tables View Screen

The "Insert", "Edit" and "Delete" buttons are disabled on the "Tables View" screen for the archived and production application versions.

The "View" button is enabled for the archived and production application versions if the table is selected.

The "View" button is disabled for the development and trial application version.

The "Provision Table" button is always enabled if the NO table is selected (it is disabled for the SO tables from the NO GUI).

Table 2 illustrates the access rights for the DCA App configuration schema and data provisioning tables. The NO/SO DCA database tables (schema) can be created, deleted and modified from the NO GUI for the development and trial application versions; they can be only viewed for the archived and production application version. The NO DCA database tables can be provisioned anytime from the NO GUI. The SO tables cannot be provisioned from the NO GUI.

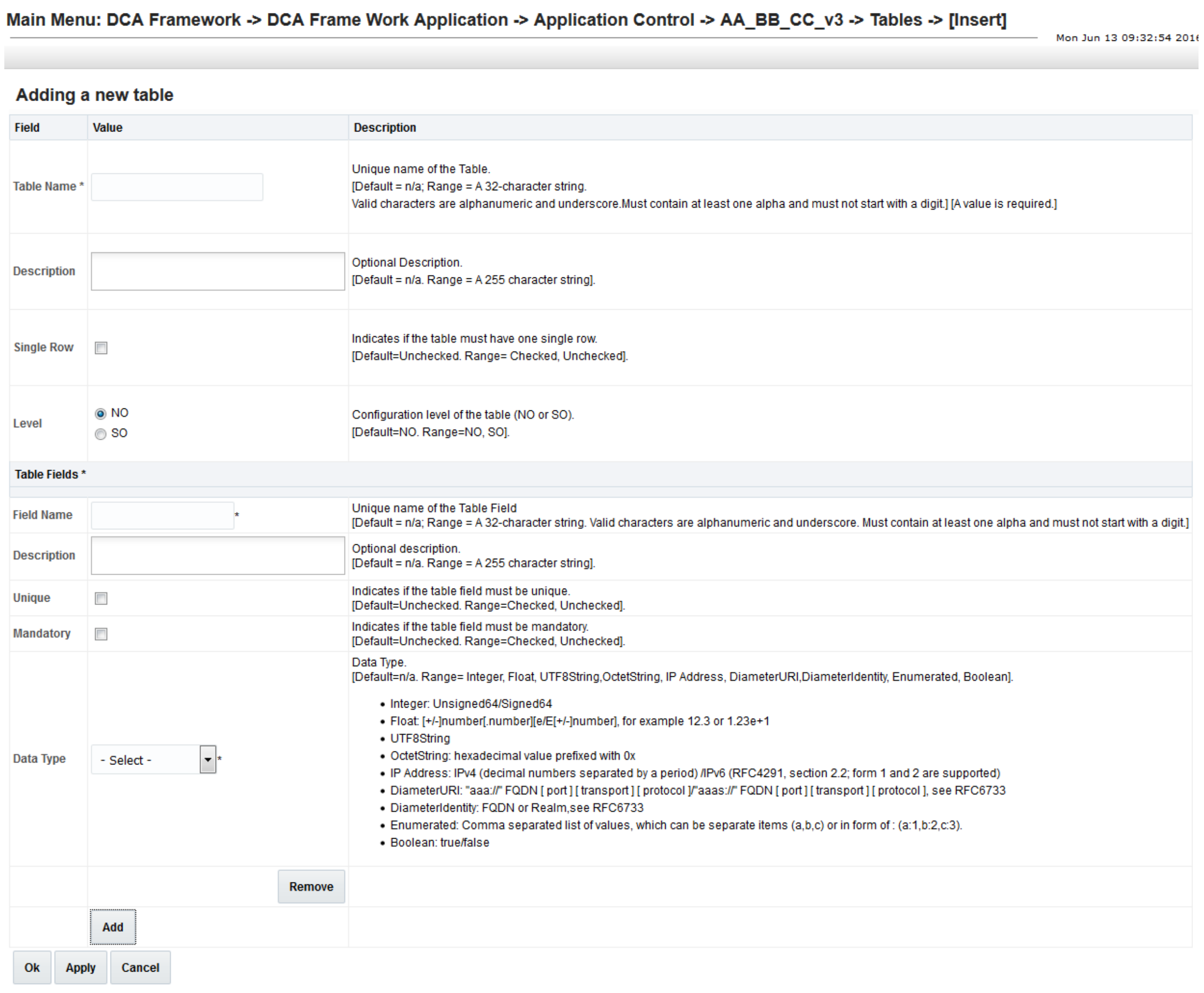
Table 2: NO GUI tables and configuration data accessibility

The accessibility of level A and level B table schema and content from the NO GUI:

	NO GUI		
	Archived	**Production**	**Development, Trial**
NO tables schema (level A)	ro	ro	rw
NO tables content (level A)	rw	rw	rw
SO tables schema (level A - shares same field as NO tables schema)	ro	ro	rw
SO tables content (level B)	n/a	n/a	n/a

ro: read-only
rw: read-write
n/a: not available

The NO **Main Menu→ DCA Framework→ < Application Name>→ Application Control→<Version Name>→ Table Insert** screen (see Figure 46) allows defining a new configuration table (NO or SO). It is accessed by clicking the **"Insert"** button on the "Tables View" screen for the development and trial application versions.



Main Menu: DCA Framework -> DCA Frame Work Application -> Application Control -> AA_BB_CC_v3 -> Tables -> [Insert]

Mon Jun 13 09:32:54 201

Adding a new table

Field	Value	Description
Table Name *		Unique name of the Table. [Default = n/a; Range = A 32-character string. Valid characters are alphanumeric and underscore.Must contain at least one alpha and must not start with a digit.] [A value is required.]
Description		Optional Description. [Default = n/a. Range = A 255 character string].
Single Row	☐	Indicates if the table must have one single row. [Default=Unchecked. Range= Checked, Unchecked].
Level	[✓ NO] SO	Configuration level of the table (NO or SO). [Default=NO. Range=NO, SO].
Table Fields *		
Field Name	*	Unique name of the Table Field [Default = n/a; Range = A 32-character string. Valid characters are alphanumeric and underscore. Must contain at least one alpha and must not start with a digit.]
Description		Optional description. [Default = n/a. Range = A 255 character string].
Unique	☐	Indicates if the table field must be unique. [Default=Unchecked. Range=Checked, Unchecked].
Mandatory	☐	Indicates if the table field must be mandatory. [Default=Unchecked. Range=Checked, Unchecked].
Data Type	- Select - *	Data Type. [Default=n/a. Range= Integer, Float, UTF8String,OctetString, IP Address, DiameterURI,DiameterIdentity, Enumerated, Boolean]. • Integer: Unsigned64/Signed64 • Float: [+/-]number[.number][e/E[+/-]number], for example 12.3 or 1.23e+1 • UTF8String • OctetString: hexadecimal value prefixed with 0x • IP Address: IPv4 (decimal numbers separated by a period) /IPv6 (RFC4291, section 2.2; form 1 and 2 are supported) • DiameterURI: "aaa://" FQDN [port] [transport] [protocol]/"aaas://" FQDN [port] [transport] [protocol], see RFC6733 • DiameterIdentity: FQDN or Realm,see RFC6733 • Enumerated: Comma separated list of values, which can be separate items (a,b,c) or in form of : (a:1,b:2,c:3). • Boolean: true/false
	Remove	
	Add	

Figure 46: NO Tables Insert Screen

Currently, there might be up to 10 configuration tables per application version (NO+SO).

The configuration table definition includes:

- Table Name and Description
- Number of table rows (single vs multiple up to 2000 rows)
- Table level (whether the table resides on the NO or the SO)
- Table Fields (up to 20 now)
  - ❑ Field Name and Description
  - ❑ Whether the field is unique
  - ❑ Whether the field is mandatory
  - ❑ Field Data Type
  - ❑ Field Default value

The table fields can be of the following types (depending on the selected data type, ranges must be also defined):

- Integer (Range: Min. and Max. values)

- Float (Range: Min. and Max. values)
- UTF8String (Range: Max. length)
- OctetString (Range: Max. length)
- IPaddress
- DiameterURI
- DiameterIdentity
- Enumerated (The values)
- Boolean

The NO **Main Menu→ DCA Framework→ < Application Name>→ Application Control→<Version Name>→ Table Edit** screen allows editing the schema of an existing DCA App configuration table (NO or SO).

The NO **Main Menu→ DCA Framework→ < Application Name>→ Application Control→<Version Name>→ Table View (Read-only Insert/Edit)** screen allows viewing a DCA App configuration table in read-only mode. It is accessed when the table is selected and the "View" button is clicked on the NO "Tables View" screen for the archived and production application version.

9.2.12 Provision Tables Screen

The NO **Main Menu→ DCA Framework→ < Application Name>→ Application Control→<Version Name>→ Provision Table View** screen (Figure 48) allows:

- Listing all the data rows provisioned for the NO configuration table
- Inserting a new data row to the NO configuration table (via NO Provision Table Insert Screen)
- Editing a data row of the NO configuration table (via NO Provision Table Edit Screen)
- Deleting a data row from the NO configuration table
- Deleting all provisioned rows at once

It is accessed by selecting the table and clicking **"Provision Table"** button on the "Tables View" screen, see Figure 47.

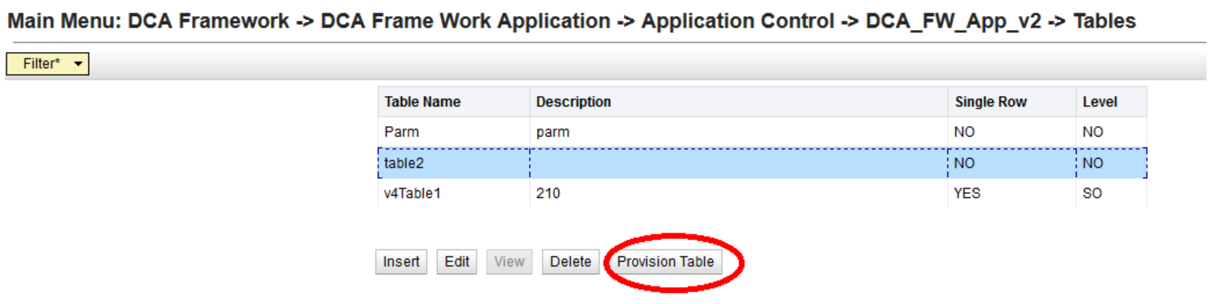

Main Menu: DCA Framework -> DCA Frame Work Application -> Application Control -> DCA_FW_App_v2 -> Tables

Table Name	Description	Single Row	Level
Parm	parm	NO	NO
table2		NO	NO
v4Table1	210	YES	SO

Figure 47: Provision Table button

The "Provision Table" button is disabled for the SO tables from the NO GUI, see Table 2.

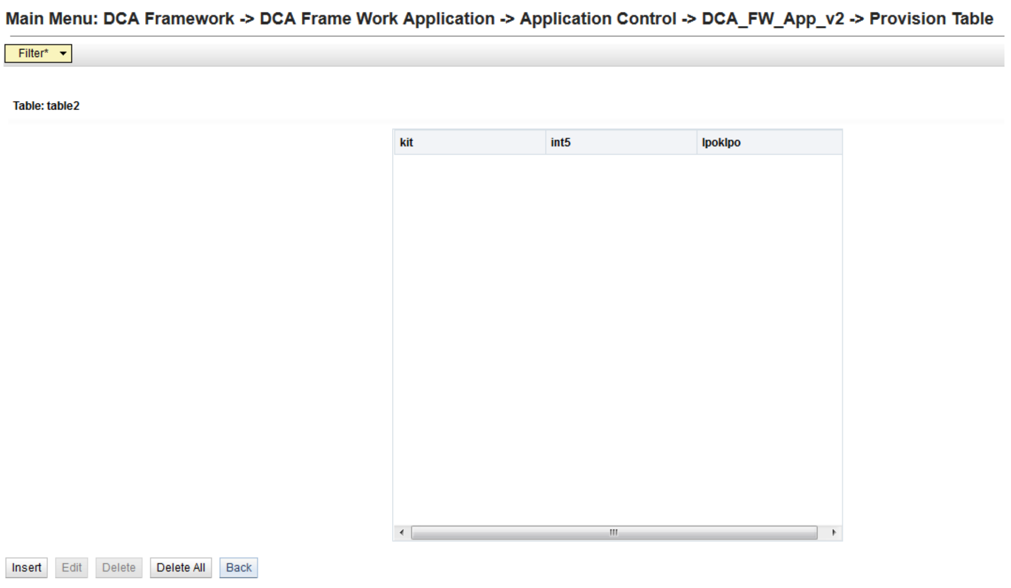

Main Menu: DCA Framework -> DCA Frame Work Application -> Application Control -> DCA_FW_App_v2 -> Provision Table

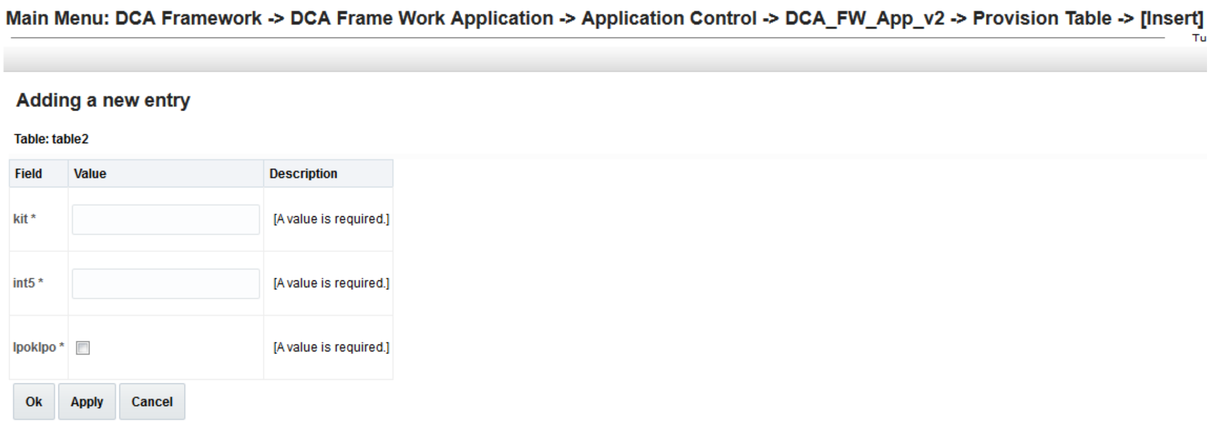
Table: table2

kit	int5	lpoklpo

Figure 48: NO Provision Table View Screen

Up to 2000 rows of data can be provisioned per table unless the table has only single row (the "Single row" checkbox has been checked on the "Table Insert" screen).

The NO **Main Menu→ DCA Framework→ < Application Name>→ Application Control→<Version Name>→ Provision Table Insert** screen (see Figure 49) allows inserting a new data row to the NO configuration table.


Main Menu: DCA Framework -> DCA Frame Work Application -> Application Control -> DCA_FW_App_v2 -> Provision Table -> [Insert]

Adding a new entry

Table: table2

Field	Value	Description
kit *		[A value is required.]
int5 *		[A value is required.]
lpoklpo *	☐	[A value is required.]

Figure 49: NO Provision Table Insert Screen

During the data insert, the GUI performs the following validations:

- Whether the mandatory value is present
- Whether the unique value is unique
- Whether the maximum of data rows is reached
- Whether the data inserted corresponds to the specified field data type
- Whether the data inserted is between the specified min-max range for the field
- Whether the entered sting value is no longer than the allowed maximum for the field
- Whether the entered enumerated value is within the allowed range of enumerated values for the field
- Etc.

The NO **Main Menu→ DCA Framework→ < Application Name>→ Application Control→<Version Name>→ Provision Table Edit** screen allows editing a data row of the NO configuration table.

9.3 SO Screens

The DCA Framework left hand menu on the SO includes the following screens:

- Configuration Screen (NO screen, read-only on the SO)

Each activated application is represented by the separate menu folder with the given application name. The application folder on the NO includes the following screens ("Application Control" screen contains the buttons that lead to other DCA screens):

- Custom Meals (NO screen, read-only on the SO)
- General Options Screen (NO screen, read-only on the SO)
- Trial MPs Assignment Screen (NO screen, read-only on the SO)
- Application Control Screen
  - ❑ Import Pop-Up Window
  - ❑ Export Pop-Up Window
  - ❑ Development Environment (NO screen, read-only on the SO)
  - ❑ Tables Screen (NO screen, read-only on the SO, except for "View" and "Provision Table" button)
    - o Provision Tables Screen
- System Options Screen

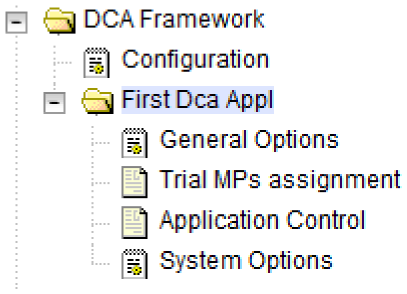


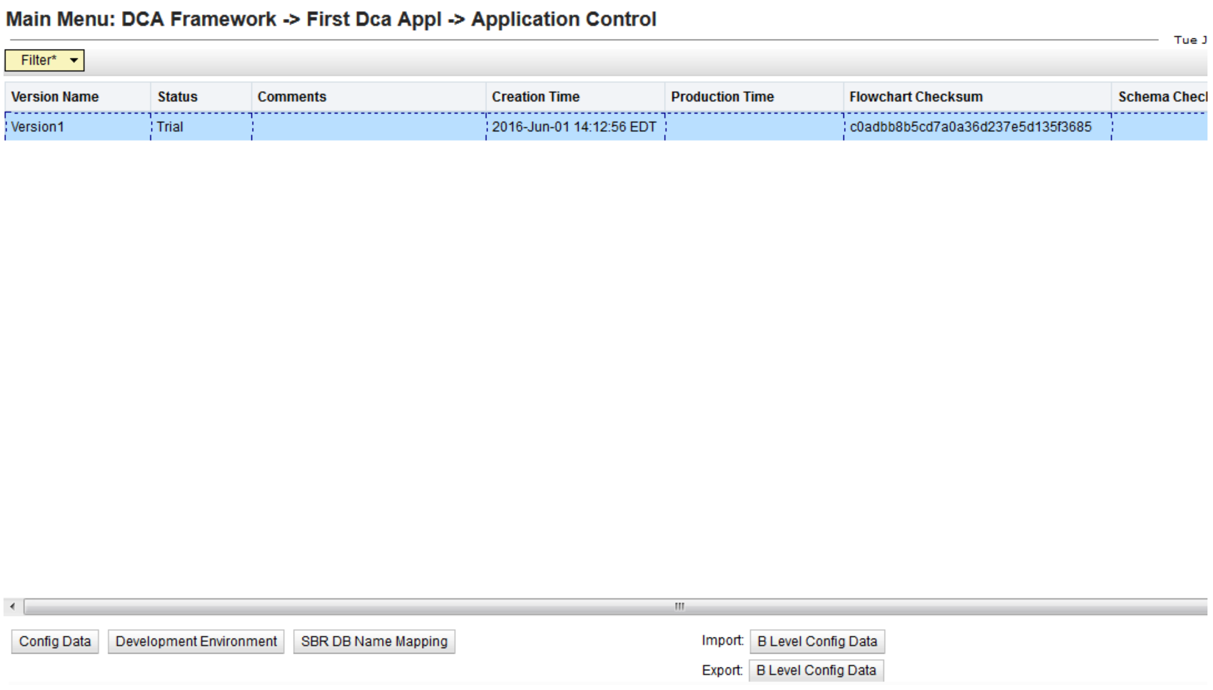
Figure 50: SO Screens

9.3.1 Application Control Screen

The SO **Main Menu→ DCA Framework→ < Application Name>→ Application Control** screen (see Figure 51) allows:

- Listing all application versions configured in the system
- Exporting only the SO provisioned data of an application version (via SO Export Pop-Up Window)
- Importing only the SO provisioned data to an existing application version (via SO Import Pop-Up Window).
- Accessing the application version configuration tables (via SO Tables View Screen)

- Accessing the flowchart and business logic of an application version (via development environment ,read-only)


Main Menu: DCA Framework -> First Dca Appl -> Application Control

Version Name	Status	Comments	Creation Time	Production Time	Flowchart Checksum	Schema Checl
Version1	Trial		2016-Jun-01 14:12:56 EDT		c0adbb8b5cd7a0a36d237e5d135f3685	

Figure 51: SO Application Control Screen

9.3.2 Export Pop-Up Window

The B level (SO level) configuration data can be exported from the SO machine, but not from the NO.

To export the configuration data to a JSON file, select the application version and click **"Export B Level Config Data"** (becomes enabled when the row is selected).

9.3.3 Import Pop-Up Window

The SO Import Pop-Up window allows specifying a JSON file from which the SO provisioned data shall be imported.

Note: The provisioned data imported to the existing business logic shall be appended to the existing data rows.

If the user wants to overwrite the configuration data, it is recommended to first delete all provisioned rows on the "Provision Table" screen and then import the new configuration data.

The B level (SO level) configuration data can be imported only to the SO machine.

To import the configuration data from the JSON file, select the application version and click **"Import B Level Config Data"** (becomes enabled when the row is selected). Select the file.

9.3.4 Tables Screen

The SO **Main Menu→ DCA Framework→ < Application Name>→ Application Control→<Version Name>→ Tables View** screen (see Figure 52) allows:

- Listing all the config tables (NO+SO) defined for an application version
- Viewing an existing config table (via NO/SO Table View Screen)
- Accessing the Provision Table View and Insert/Edit screens (via SO Provision Table View Screen, SO Provision Table Insert Screen and SO Provision Table Edit Screen).

The SO “Tables View” screen is accessed by selecting the application version and clicking **“Config Data”** button on the SO “Application Control” screen.

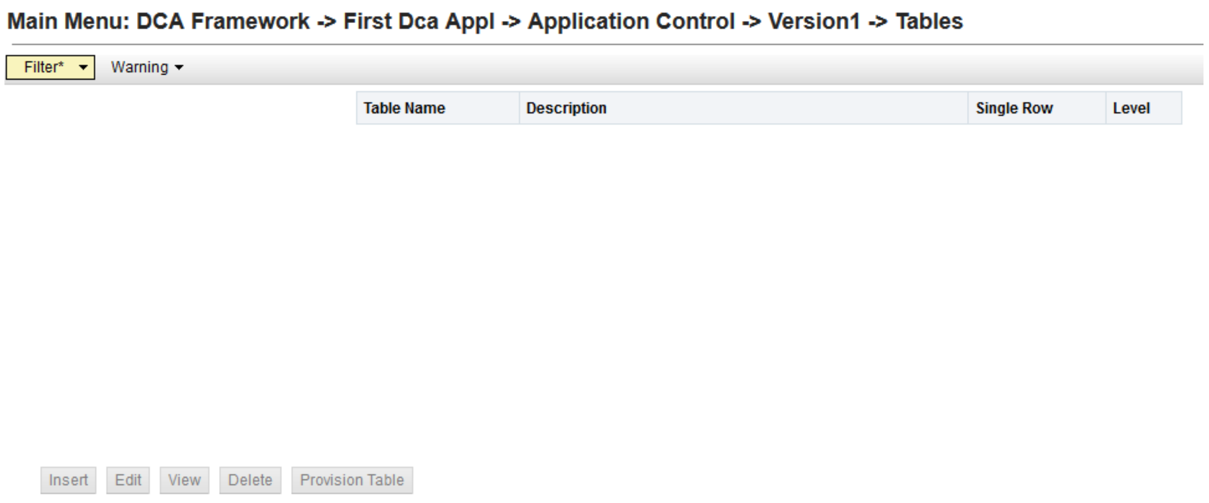

Main Menu: DCA Framework -> First Dca Appl -> Application Control -> Version1 -> Tables
Filter*
Warning
Table Name
Description
Single Row
Level
Insert
Edit
View
Delete
Provision Table

Figure 52: SO Tables View Screen (empty)

The “Insert”, “Edit” and “Delete” buttons are disabled on the SO “Tables View” screen.

The “View” button is enabled if the table is selected.

The “Provision Table” button is always enabled if the NO/SO table is selected.

Table 3 illustrates the access rights for the DCA App configuration schema and data provisioning tables on the SO. The NO/SO DCA App table schemas can only be viewed. The level A DCA App configuration tables content can only be view from the SO GUI. The level B DCA App configuration tables can be provisioned.

Table 3: SO GUI tables and Configuration data accessibility

The accessibility of level A and level B table schema and content from the SO GUI:

	SO GUI		
	Archived	**Production**	**Development, Trial**
NO tables schema (level A)	ro (replicated)	ro (replicated)	ro (replicated)
NO tables content (level A)	ro (replicated)	ro (replicated)	ro (replicated)
SO tables schema (level A - shares same field as NO tables schema)	ro (replicated)	ro (replicated)	ro (replicated)
SO tables content (level B)	rw	rw	rw

ro: read-only
rw: read-write
n/a: not available

The SO **Main Menu→ DCA Framework→ < Application Name>→ Application Control→<Version Name>→ Table View (Read-only Insert/Edit)** screen allows viewing a configuration table in read-only mode. It is accessed when the table is selected and the “View” button is clicked on the SO “Tables View” screen.

9.3.5 Provision Tables Screen

The SO **Main Menu→ DCA Framework→ < Application Name>→ Application Control→<Version Name>→ Provision Table**, View screen allows:

- Listing all the data rows provisioned for the SO-rooted DCA App configuration table

- Inserting a new data row to the SO-rooted DCA App configuration table (via SO Provision Table Insert Screen)
- Editing a data row of the SO-rooted DCA App configuration table (via SO Provision table Edit Screen)
- Deleting a data row from the SO-rooted DCA App configuration table.
- Deleting all provisioned rows at once

Note: The NO-rooted DCA App configuration tables, as well as the schema definitions of both the NO-rooted and SO-rooted DCA App configuration tables are accessible on the SO only in read-only mode.

The SO “Provision Table View” screen is accessed by selecting the table and clicking **“Provision Table”** on the SO “Tables View” screen.

The SO **Main Menu→ DCA Framework→ < Application Name>→ Application Control→<Version Name>→ Provision Table**, Insert screen allows inserting a new data row to the SO-rooted DCA App configuration table.

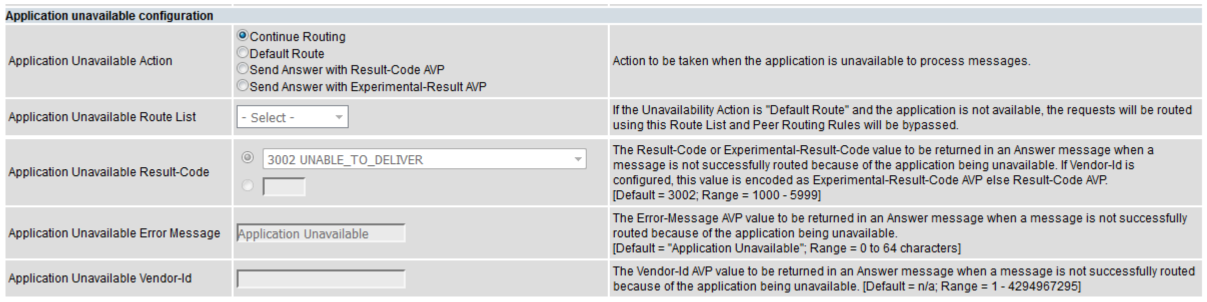
The SO **Main Menu→ DCA Framework→ < Application Name>→ Application Control→<Version Name>→ Provision Table**, Edit screen allows editing a data row of the SO-rooted DCA App configuration table.

9.4 System Options

“System Options” screen is present on the SO only. See Figure 53- Figure 57.

“System Options” screen enables the configuration of the DSR application parameters that are:

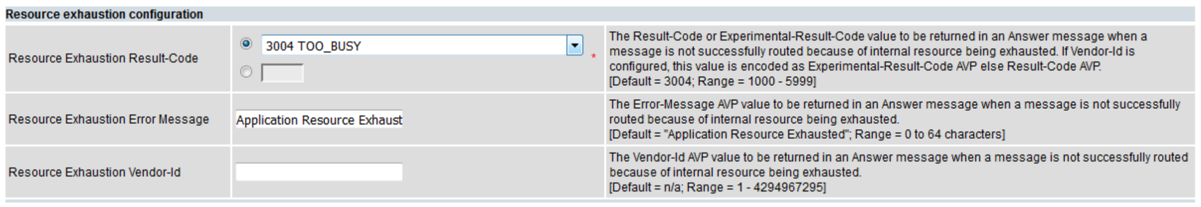
- Relevant to the operational status “unavailable”. These options allow to specify the behavior in case when the application state is “unavailable” (**Main Menu: Diameter→Maintenance→Applications).** The possible behavior is:
    - Continue Routing
    - Use default route + specify application unavailable route list
    - Send Answer with Result-Code AVP + specify Result-Code and Error Message
    - Send Answer with Experimental-Result AVO + specify Result-Code, Error Message and Vendor-Id.



Application unavailable configuration		
Application Unavailable Action	[✓ Continue Routing] Default Route Send Answer with Result-Code AVP Send Answer with Experimental-Result AVP	Action to be taken when the application is unavailable to process messages.
Application Unavailable Route List	- Select -	If the Unavailability Action is "Default Route" and the application is not available, the requests will be routed using this Route List and Peer Routing Rules will be bypassed.
Application Unavailable Result-Code	[✓ 3002 UNABLE_TO_DELIVER]	The Result-Code or Experimental-Result-Code value to be returned in an Answer message when a message is not successfully routed because of the application being unavailable. If Vendor-Id is configured, this value is encoded as Experimental-Result-Code AVP else Result-Code AVP. [Default = 3002; Range = 1000 - 5999]
Application Unavailable Error Message	Application Unavailable	The Error-Message AVP value to be returned in an Answer message when a message is not successfully routed because of the application being unavailable. [Default = "Application Unavailable"; Range = 0 to 64 characters]
Application Unavailable Vendor-Id		The Vendor-Id AVP value to be returned in an Answer message when a message is not successfully routed because of the application being unavailable. [Default = n/a; Range = 1 - 4294967295]

Figure 53: System Options for the "Unavailable" Operation Status

- Relevant to the case when the DRL resources are exhausted. The behavior is to send an error message with the specified Result-Code, Error Message and Vendor-Id.

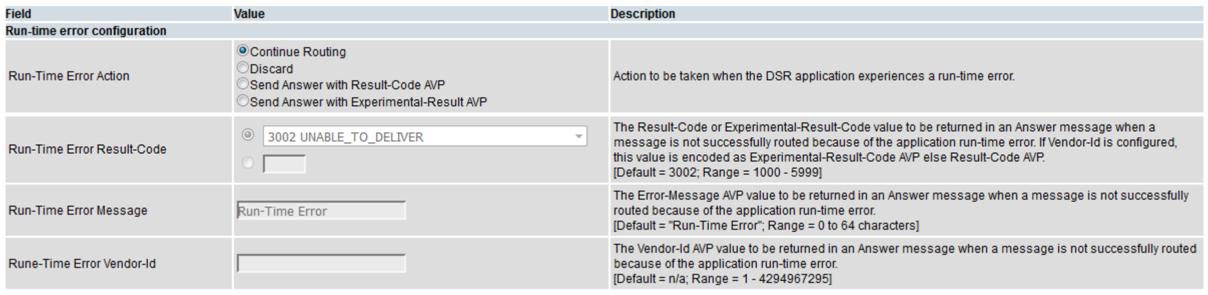


Resource exhaustion configuration		
Resource Exhaustion Result-Code	(● 3004 TOO_BUSY) ○ [ ] *	The Result-Code or Experimental-Result-Code value to be returned in an Answer message when a message is not successfully routed because of internal resource being exhausted. If Vendor-Id is configured, this value is encoded as Experimental-Result-Code AVP else Result-Code AVP. [Default = 3004; Range = 1000 - 5999]
Resource Exhaustion Error Message	Application Resource Exhaust	The Error-Message AVP value to be returned in an Answer message when a message is not successfully routed because of internal resource being exhausted. [Default = "Application Resource Exhausted"; Range = 0 to 64 characters]
Resource Exhaustion Vendor-Id		The Vendor-Id AVP value to be returned in an Answer message when a message is not successfully routed because of internal resource being exhausted. [Default = n/a; Range = 1 - 4294967295]

Figure 54: System Options for the Exhausted DRL Resources

- Relevant to the run-time error. These options allow to specify the behavior in case of a run-time error. Runtime errors fall into two categories:
  - Perl specific runtime errors – e.g. division by zero, a "die" statement, calling an undefined (utility, not event handler) subroutine etc.,
  - Runtime errors triggered by the DCA framework – e.g. invoking an event handler that does not exist or exceeding the maximum configured number of executed opcodes.

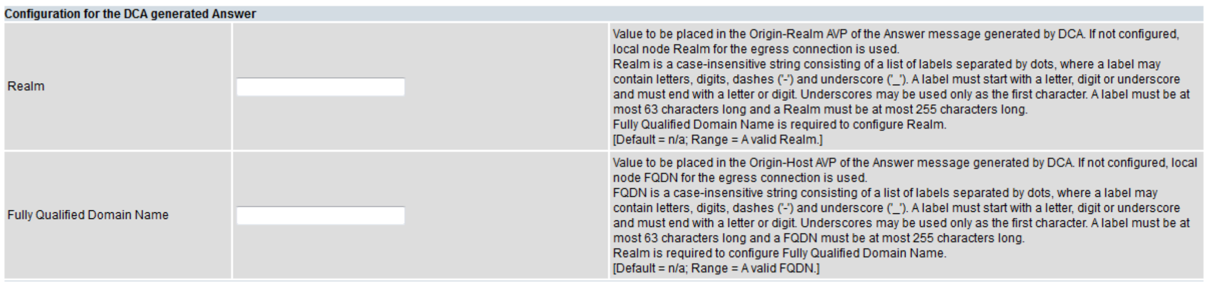
  The possible behavior is:
  - Continue Routing
  - Discard
  - Send Answer with Result-Code AVP + specify Result-Code and Error Message
  - Send Answer with Experimental-Result AVO + specify Result-Code, Error Message and Vendor-Id.



Field	Value	Description
Run-time error configuration		
Run-Time Error Action	[✓ Continue Routing] Discard Send Answer with Result-Code AVP Send Answer with Experimental-Result AVP	Action to be taken when the DSR application experiences a run-time error.
Run-Time Error Result-Code	(● 3002 UNABLE_TO_DELIVER) ○ [ ]	The Result-Code or Experimental-Result-Code value to be returned in an Answer message when a message is not successfully routed because of the application run-time error. If Vendor-Id is configured, this value is encoded as Experimental-Result-Code AVP else Result-Code AVP. [Default = 3002; Range = 1000 - 5999]
Run-Time Error Message	Run-Time Error	The Error-Message AVP value to be returned in an Answer message when a message is not successfully routed because of the application run-time error. [Default = "Run-Time Error"; Range = 0 to 64 characters]
Rune-Time Error Vendor-Id		The Vendor-Id AVP value to be returned in an Answer message when a message is not successfully routed because of the application run-time error. [Default = n/a; Range = 1 - 4294967295]

Figure 55: System Options for the Run-Time Error

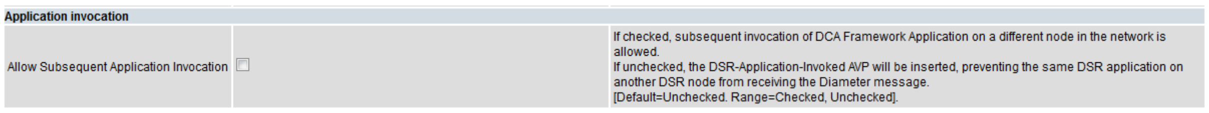
- Realm and FQDN that are placed in Answer message generated by the DCA. These are the values that will be placed in the Origin-Realm and Origin-Host AVPs of the Answer message generated by DCA. If they are not configured, local node Realm and FQDN for the egress connection will be used.



Configuration for the DCA generated Answer		
Realm		Value to be placed in the Origin-Realm AVP of the Answer message generated by DCA. If not configured, local node Realm for the egress connection is used. Realm is a case-insensitive string consisting of a list of labels separated by dots, where a label may contain letters, digits, dashes ('-') and underscore ('_'). A label must start with a letter, digit or underscore and must end with a letter or digit. Underscores may be used only as the first character. A label must be at most 63 characters long and a Realm must be at most 255 characters long. Fully Qualified Domain Name is required to configure Realm. [Default = n/a; Range = A valid Realm.]
Fully Qualified Domain Name		Value to be placed in the Origin-Host AVP of the Answer message generated by DCA. If not configured, local node FQDN for the egress connection is used. FQDN is a case-insensitive string consisting of a list of labels separated by dots, where a label may contain letters, digits, dashes ('-') and underscore ('_'). A label must start with a letter, digit or underscore and must end with a letter or digit. Underscores may be used only as the first character. A label must be at most 63 characters long and a FQDN must be at most 255 characters long. Realm is required to configure Fully Qualified Domain Name. [Default = n/a; Range = A valid FQDN.]

Figure 56: System Options for the Realm and FQDN

- Application invocation. This option is needed to indicate if the subsequent invocation of application on a different node in the network is allowed or not.
  If unchecked, the DSR-Application-Invoked AVP will be inserted, preventing the same DSR application on another DSR node from receiving the Diameter message.



Application invocation		
Allow Subsequent Application Invocation	☐	If checked, subsequent invocation of DCA Framework Application on a different node in the network is allowed. If unchecked, the DSR-Application-Invoked AVP will be inserted, preventing the same DSR application on another DSR node from receiving the Diameter message. [Default=Unchecked. Range=Checked, Unchecked].

Figure 57: System Options for the Application Invocation

10 APIs

This chapter documents the various APIs available to a DCA App programmer.

10.1 The EDL API

10.1.1 API to Manipulate the Diameter Header

Purpose: Retrieve the Diameter message object needed for subsequent operations on the Diameter message header and body

Prototype:

```
my $msg = diameter::Param::message($param);
```

where `$param` is a default parameter provided by all the event handlers and may be retrieved with

```
my $param = shift;
```

Purpose: Read the Diameter version number in the Diameter header

Prototype:

```
my $ver = diameter::Message::version($msg);
```

where `$ver` is `undef` in case of failure (e.g. wrong object passed in `$msg`) or the Diameter version number if success

Purpose: Set the Diameter version number in the Diameter header

Prototype:

```
$err = diameter::Message::setVersion($msg, $ver);
```

where `$err` is `undef` in case of failure (e.g. wrong object passed in `$msg`) or a non-zero value in case of success

Purpose: Return the length (as number of bytes) of the Diameter message

Prototype:

```
my $len = diameter::Message::messageLength($msg);
```

where `$len` is `undef` in case of failure (e.g. wrong object passed in `$msg`) or the length of the Diameter message if success

Purpose: Read the Command Flags of the Diameter message.

Prototype:

```
my $cmdFlags = diameter::Message::commandFlags($msg);
```

where `$cmdFlags` is `undef` in case of failure (e.g. wrong object passed in `$msg`) or the Command Flags if success.

Purpose: Read the Request flag of the Diameter message.

Prototype:

```
my $r = diameter::Message::isRequest($msg);
```

where `$r` is 1 if the Request flag is set, 0 if the Request flag is not set or `undef` if error (e.g wrong object passed in `$msg`).

Purpose: Read the Diameter Proxiable flag in the Diameter header.

Prototype:

```
my $p = diameter::Message::isProxiable($msg);
```

where `$p` is 1 if the Proxiable flag is set, 0 if the Proxiable flag is not set or `undef` if error (e.g wrong object passed in `$msg`).

Purpose: Set (set to 1) the Diameter Proxiable flag in the Diameter header.

Prototype:

```
$err = diameter::Message::setProxiable($msg);
```

where `$err` is `undef` if error (e.g wrong object passed in `$msg`) or a non-zero value if success.

Purpose: Clear (set to 0) the Diameter Proxiable flag in the Diameter header.

Prototype:

```
$err = diameter::Message::clearProxiable($msg);
```

where `$err` is `undef` if error (e.g wrong object passed in `$msg`) or a non-zero value if success.

Purpose: Read the Diameter Error flag in the Diameter header.

Prototype:

```
my $e = diameter::Message::isError($msg)
```

where `$e` is 1 if the Error flag is set, 0 if the Error flag is not set or `undef` if error (e.g wrong object passed in `$msg`).

Purpose: Set (set to 1) the Diameter Error flag in the Diameter header.

Prototype:

```
$err = diameter::Message::setError($msg);
```

where `$err` is `undef` if error (e.g wrong object passed in `$msg`) or a non-zero value if success.

Purpose: Clear (set to 0) the Diameter Error flag in the Diameter header.

Prototype:

```
$err = diameter::Message::clearError($msg);
```

where `$err` is `undef` if error (e.g wrong object passed in `$msg`) or a non-zero value if success.

Purpose: Read the Diameter Retransmission flag in the Diameter header.

Prototype:

```
my $t = diameter::Message::isRetransmission($msg);
```

where `$t` is 1 if the Retransmission flag is set, 0 if the Retransmission flag is not set or `undef` if error (e.g wrong object passed in `$msg`).

Purpose: Set (set to 1) the Diameter Retransmission flag in the Diameter header.

Prototype:

```
$err = diameter::Message::setRetransmission($msg);
```

where `$err` is `undef` if error (e.g wrong object passed in `$msg`) or a non-zero value if success.

Purpose: Clear (set to 0) the Diameter Retransmission flag in the Diameter header.

Prototype:

```
$err = diameter::Message::clearRetransmission($msg);
```

where `$err` is `undef` if error (e.g wrong object passed in `$msg`) or a non-zero value if success.

Purpose: Read the Diameter 4th reserved bit of the Command Flags in the Diameter header.

Prototype:

```
my $r4 = diameter::Message::isReservedBit4($msg);
```

where `$t` is 1 if the 4th bit in the Command Flags flag is set, 0 if the bit is not set or `undef` if error (e.g wrong object passed in `$msg`).

Purpose: Set (set to 1) the Diameter 4th reserved bit of the Command Flags in the Diameter header.

Prototype:

```
$err = diameter::Message::setReservedBit4($msg);
```

where `$err` is `undef` if error (e.g wrong object passed in `$msg`) or a non-zero value if success.

Purpose: Clear (set to 0) the Diameter 4th reserved bit of the Command Flags in the Diameter header.

Prototype:

```
$err = diameter::Message::clearReservedBit4($msg);
```

where `$err` is `undef` if error (e.g wrong object passed in `$msg`) or a non-zero value if success.

Purpose: Read/Set/Clear the Diameter 5th, 6th and 7th reserved bit in the Command Flags in the Diameter header.

Prototype:

See three examples above where the “Bit4” suffix in the function names is accordingly replaced by “Bit5”, “Bit6” and respectively “Bit7”.

Purpose: Read the Diameter Command Code in the Diameter header

Prototype:

```
my $cmd = diameter::Message::commandCode($msg);
```

where `$cmd` is `undef` if error (e.g wrong object passed in `$msg`) or contains the Command Code if success.

Purpose: Set the Diameter Command Code in the Diameter header

Prototype:

```
$err = diameter::Message::setCommandCode($msg, $cmd);
```

where `$err` is `undef` if error (e.g wrong object passed in `$msg`) or a non-zero value if success.

Purpose: Read the Diameter Application-ID in the Diameter header

Prototype:

```
my $appId = diameter::Message::applicationId($msg);
```

where `$appId` is `undef` if error (e.g wrong object passed in `$msg`) or contains the Application-ID if success

Purpose: Set the Diameter Application-ID in the Diameter header

Prototype:

```
$err = diameter::Message::setApplicationId($msg, $appId);
```

where `$err` is `undef` if error (e.g wrong object passed in `$msg`) or a non-zero value if success

Purpose: Read the Diameter Hop-by-Hop Identifier in the Diameter header

Prototype:

```
my $hbh = diameter::Message::hopByHopId($msg);
```

where `$hbh` is `undef` if error (e.g wrong object passed in `$msg`) or contains the Hop-by-Hop Identifier if success

Purpose: Set the Diameter Hop-by-Hop Identifier in the Diameter header

Prototype:

```
$err = diameter::Message::setHopByHopId($msg, $hbh);
```

where `$err` is `undef` if error (e.g wrong object passed in `$msg`) or a non-zero value if success

Purpose: Read the Diameter End-to-End Identifier in the Diameter header

Prototype:

```
my $err = diameter::Message::endToEndId($msg);
```

where `$err` is `undef` if error (e.g wrong object passed in `$msg`) or contains the End-to-End Identifier if success

Purpose: Set the Diameter End-to-End Identifier in the Diameter header

Prototype:

```
$err = diameter::Message::setEndToEndId($msg, $e2e);
```

where `$err` is `undef` if error (e.g wrong object passed in `$msg`) or a non-zero value if success

10.1.2 API to Manipulate the Diameter AVPs

Purpose: Read from a Diameter message the value of an AVP identified by name and instance number

Prototype:

```
my $val = diameter::Message::getAvpValue($msg, $avp_name [,
$instance]);
```

The return values shall be:

- `undef` if `$instance` is 0,
- `undef` if there are less instances of the AVPin the Diameter message than the `$instance` value or an AVP with the specified name does not exist in the Diameter message or the AVP name is not specified in the AVP Dictionary,
- The value of the `$instance`-th instance of the AVP (starting from 1),
- The value of the first instance of the AVP if `$instance` has been omitted,
- `undef` if `$msg` does not contain a `diameter::Message` or `diameter::GroupedAvp` object or the other parameters (if any) are `undef`.

Purpose: Add at the end of the Diameter message an AVP identified by name and value

Prototype:

```
my $err = diameter::Message::addAvpValue($msg, $avp_name, $avp_val);
```

The return code shall be:

- Non-zero in case of success,
- `undef` if the AVP name does not exist in the AVP Dictionary,
- `undef` if the AVP name exists in the AVP Dictionary,
- `undef` if the AVP value cannot be converted to the AVP data type specified in the AVP Dictionary,
- `undef` if `$msg` does not contain a `diameter::Message` or `diameter::GroupedAvp` object or the other parameters (if any) are `undef`.

Purpose: Set the value of an AVP in a Diameter message

Prototype:

```
my $err = diameter::Message::setAvpValue($msg, $avp_name, $avp_val
[, $instance]);
```

If `$instance` has been omitted, the first instance of the AVP will be set. The return code shall be:

- Non-zero in case of success,
- `undef` if the AVP name does not exist in the AVP Dictionary,
- `undef` if the AVP name exists in the AVP Dictionary,
- `undef` if the AVP name is valid but no such AVP exists in the Diameter message,
- `undef` if `$instance` is 0,
- `undef` if the AVP exists in the Diameter message but `$instance` value is greater than the number of AVP instances in the Diameter message,
- `undef` if the AVP value cannot be converted to the AVP data type specified in the AVP Dictionary,
- `undef` if `$msg` does not contain a `diameter::Message` or `diameter::GroupedAvp` object or the other parameters (if any) are `undef`.

Purpose: Set the value of an existing AVP in a Diameter message or add that AVP at the end of the Diameter message if the message already contains exactly `$instance` – 1 AVPs:

Prototype:

```
my $err = diameter::Message::setAddAvpValue($msg, $avp_name,
$avp_val [, $instance]);
```

If `$instance` has been omitted, it defaults to 1. The return code shall be:

- 1 in case an AVP with the specified instance number exists and its value has been successfully set,
- 2 if the Diameter messages contains exactly `$instance` – 1 AVPs of the specified type, in which case the `$instance's` AVP will be added to the end of the message,
- `undef` if the Diameter messages contains strictly less than `$instance` – 1 AVPs of the specified type,
- `undef` if the AVP name does not exist in the AVP Dictionary,
- `undef` if the AVP name exists in the AVP Dictionary,
- `undef` if the AVP name is valid but the Diameter messages already contains `$instance` or more AVPs of the specified type,
- `undef` if `$instance` is 0,
- `undef` if the AVP value cannot be converted to the AVP data type specified in the AVP Dictionary,
- `undef` if `$msg` does not contain a `diameter::Message` or `diameter::GroupedAvp` object or the other parameters (if any) are `undef`.

Purpose: Read the value of an AVP's flag octet

Prototype:

```
my $flags = diameter::Message::getAvpFlags($msg, $avp_name [,
$instance]);
```

The return value shall be:

- The value of flags octet of the `$instance`-th instance of the AVP (starting from 1),
- The value of the first instance of the AVP if `$instance` has been omitted,
- `undef` if there are less instances of the AVP in the Diameter message than the `$instance` value

- `undef` if `$instance` is 0
- `undef` if an AVP with the specified name does not exist in the Diameter message
- `undef` if the AVP name is not specified in the AVP Dictionary
- `undef` if `$msg` does not contain a `diameter::Message` or `diameter::GroupedAvp` object or the other parameters (if any) are `undef`.

Purpose: Set the value of an AVP's flag octet

Prototype:

```
my $err = diameter::Message::setAvpFlags($msg, $avp_name, $mask [,
$instance]);
```

A 1 bit in `$mask` indicates a bit to set, while a 0 bit in `$mask` preserves the original bit value.

If `$instance` has been omitted, the flags of the first instance of the AVP will be set. The return code shall be:

- Non-zero in case of success,
- `undef` if the AVP name does not exist in the AVP Dictionary,
- `undef` if the AVP name is valid but no such AVP exists in the Diameter message,
- `undef` if the AVP exists in the Diameter message but `$instance` value is greater than the number of AVP instances in the Diameter message,
- `undef` if `$instance` is 0,
- `undef` if `$msg` does not contain a `diameter::Message` or `diameter::GroupedAvp` object or the other parameters (if any) are `undef`.

Note: The "V" bit preserves the original value regardless the `$mask` value.

Purpose: Clear specific bits in an AVP's flag

Prototype:

```
my $err = diameter::Message::clearAvpFlags($msg, $avp_name, $mask [,
$instance]);
```

A 1 bit in `$mask` indicates a bit to clear, while a 0 bit in `$mask` preserves the original bit value.

If `$instance` has been omitted, the flags first instance of the AVP will be cleared. The return code shall be:

- Non-zero in case of success,
- `undef` if the AVP name does not exist in the AVP Dictionary,
- `undef` if the AVP name is valid but no such AVP exists in the Diameter message,
- `undef` if the AVP exists in the Diameter message but `$instance` value is greater than the number of AVP instances in the Diameter message,
- `undef` if `$instance` is 0,
- `undef` if `$msg` does not contain a `diameter::Message` or `diameter::GroupedAvp` object or the other parameters (if any) are `undef`.

Note: The "V" bit preserves the original value regardless the `$mask` value.

Purpose: Delete an AVP identified by name, from a Diameter message

Prototype:

```
my $err = diameter::Message::delAvp($msg, $avp_name [, $instance]);
```

If `$instance` has been omitted, the first instance of the AVP will be deleted. The return code shall be:

- 1 in case AVP is deleted,
- 0 if AVP does not exist in message,
- `undef` if the AVP name does not exist in the AVP Dictionary,
- `undef` if the AVP exists in the Diameter message but `$instance` value is greater than the number of AVP instances in the Diameter message,
- `undef` if `$instance` is 0,
- `undef` if `$msg` does not contain a `diameter::Message` or `diameter::GroupedAvp` object or the other parameters (if any) are `undef`.

Purpose: Delete all the instances of an AVP from a Diameter message

Prototype:

```
my $err = diameter::Message::delAvpAll($msg, $avp_name);
```

The return code shall be:

- 1 in case AVP is deleted,
- 0 if AVP does not exist in message,
- `undef` if the AVP name does not exist in the AVP Dictionary,
- `undef` if `$msg` does not contain a `diameter::Message` or `diameter::GroupedAvp` object or the other parameters (if any) are `undef`.

Note: The AVPs on the same nesting level are deleted, i.e. the un-grouped AVPs in a Diameter message, if the function is called with a Diameter message parameter or the AVPs in a specific grouped AVP that are not deeper nested in a further grouped AVP, if the function is called with a Grouped AVP parameter.

Purpose: Return the number of instances of an AVP from a Diameter message

Prototype:

```
my $cnt = diameter::Message::countAvp($msg, $avp_name);
```

The return value shall be:

- 0 if the AVP does not exist in the Diameter message,
- A strictly positive number indicating the number of occurrences of the respective AVP in the Diameter message,
- `undef` if the AVP name does not exist in the AVP Dictionary,
- `undef` if `$msg` does not contain a `diameter::Message` or `diameter::GroupedAvp` object or the other parameters (if any) are `undef`.

Note: The AVPs on the same nesting level are counted, i.e. the un-grouped AVPs in a Diameter message, if the function is called with a Diameter message parameter or the AVPs in a specific grouped AVP that are not deeper nested in a further grouped AVP, if the function is called with a Grouped AVP parameter.

Purpose: Check whether a specific AVP (instance) exists in a Diameter message

Prototype:

```
my $exists = diameter::Message::avpExists($msg, $avp_name [,
$instance]);
```

The return value shall be:

- True if `$instance` is omitted and at least one AVP with the specified name exists,
- True if `$instance` is specified and an AVP with the specified name and instance number exists,
- False if no AVP with the specified name exists in the Diameter message,
- False if `$instance` is specified, at least one AVP with the specified name exists, but the number of instances of the respective AVP is strictly less than the specified `$instance`,
- `undef` if the AVP name does not exist in the AVP Dictionary,
- `undef` if `$msg` does not contain a `diameter::Message` or `diameter::GroupedAvp` object or the other parameters (if any) are `undef`.

Note: The AVPs on the same nesting level are checked, i.e. the un-grouped AVPs in a Diameter message, if the function is called with a Diameter message parameter or the AVPs in a specific grouped AVP that are not deeper nested in a further grouped AVP, if the function is called with a Grouped AVP parameter.

Purpose: Return the length of the payload of an AVP from a Diameter message

Prototype:

```
my $len = diameter::Message::avpDataLength($msg, $avp_name [,
$instance]);
```

If `$instance` has been omitted, the length of the first instance of the AVP will be returned. The return value shall be:

- `undef` if no AVP with that name exists in the Diameter message,
- `undef` if `$instance` is specified but less than `$instance` AVPs exists in the Diameter message,
- A strictly positive number or 0, indicating the length of the payload of the indicated AVP instance.
- `undef` if the AVP name does not exist in the AVP Dictionary,
- `undef` if `$msg` does not contain a `diameter::Message` or `diameter::GroupedAvp` object or the other parameters (if any) are `undef`.

10.1.3 API to Manipulate the Diameter Grouped AVPs

All the API functions introduced in the previous section, work on grouped AVPs as well. For instance, the value of the Subscription-Id grouped AVP may be read with:

```
my $gVal = diameter::Message::getAvpValue($msg, "Subscription-Id");
```

and the Subscription-Id grouped AVP may be added to a Diameter message with:

```
my $err = diameter::Message::addAvpValue($msg, "Subscription-Id",
$gVal);
```

It shall be noted, however, that in this case `$gVal` is an OctetString that contains both the “Subscription-Id-Type” and the “Subscription-Id-Data” AVPs.

This approach is particularly handy when the “Subscriber-Id” grouped AVP needs to be copied from one Diameter message to another, without having to look into the individual AVPs included in it.

However, if accessing the individual AVPs included into a grouped AVP is desired, then the `getGroupedAvp` and `addGroupedAvp` API calls provide the necessary support:

Purpose: Access a Grouped AVP in a Diameter message

Prototype:

```
my $gAvp = diameter::Message::getGroupedAvp($msg, $avp_name [,
$instance]);
```

The return value shall be:

- `undef` if the AVP name does not exist in the AVP dictionary,
- `undef` if AVP name exists in the AVP dictionary but it is not defined as a Grouped AVP,
- `undef` if the AVP name is valid but the Diameter message does not contain a Grouped AVP with that name,
- `undef` if the AVP name is valid but the Diameter message contains less Grouped AVPs with that name than specified in `$instance`,
- A `diameter::GroupedAvp` Grouped AVP object that corresponds to the respective instance of the Grouped AVP (or to the first instance if `$instance` is omitted).

The `$gAvp` `diameter::Grouped` AVP object can be used to manipulate the AVPs that it contains using any of the API functions introduced so far:

```
$result = diameter::GroupedAvp::<API_function>($gAVP,
<API_function_params>);
```

where the `$gAVP` object of type `diameter::GroupedAvp` replaces the `$msg` object of type `$diameter`::`Message` and `$result` represents the return parameter of the respective API function..

Note: `getGroupedAvp` works recursively to get a grouped AVP (`$nested`_gAVP) contained in another grouped AVP (`$gAvp`):

```
my $nested_gAvp = diameter::Message::getGroupedAvp($gAvp,
$avp_name);
```

where `$gAvp` is a `dimeter::GroupedAvp` object

Purpose: Add a Grouped AVP to the end of a Diameter message

Prototype:

```
my $gAvp = diameter::Message::addGroupedAvp($msg, $avp_name);
```

where `$gAvp` is a `diameter::GroupedAvp` object.

The return value shall be:

- `undef` if the AVP name does not exist in the AVP dictionary,
- `undef` if AVP name exists in the AVP dictionary but it is not defined as a Grouped AVP.

A `diameter::GroupedAvp` Grouped AVP object can be further used to manipulate the AVPs that it contains:

```
my $subscription_id = diameter::Message::addGroupedAvp($msg,
"Subscription-Id");

diameter:GroupedAvp::addAvpValue($subscription_id, "Subscription-Id-
Type", $avp_val);

diameter::GroupedAvp::addAvpValue($subscription_id, "Subscription-
Id-Data", $avp_val);
```

Note: `addGroupedAvp` works recursively to add a grouped AVP (`$nested`_gAVP) within another grouped AVP (`$gAvp`):

```
my $nested_gAvp = diameter::Message::addGroupedAvp($gAvp,
$avp_name);
```

where `$gAvp` is a `diameter::GroupedAvp` object

10.2 Diameter Transaction Stateful APIs

10.2.1 Internal Variables

This API is primary intended to enable a DCA App to interact with Mediation Rules through Internal Variables. Internal Variables have been introduced by the Mediation feature and can be configured from Main Menu: Diameter → Mediation → Internal Variables. Internal Variables are persistent throughout the lifetime of a Diameter transaction.

Purpose: Access Internal Variables

Prototype:

```
my $iv_ref = new diameter::InternalVarDef("<IV_Name>");

my $internalVarMap = diameter::Param::internalVarMap($param);
```

where `$param` is the opaque parameter passed to every event handler and `<IV_Name>` is the name assigned to the Internal Variable in Main Menu: Diameter → Mediation → Internal Variables.

Note: The Internal Variables are configurable at the B level, therefore the `<IV_Name>` must be configured on all the sites. Otherwise, the initialization will fail when invoked on those DA-MP located in sites where `<IV_Name>` does not exist.

Purpose: Set and Get Internal Variables

Prototype:

```
diameter::InternalVarMap::set($internalVarMap, $iv_ref, $val);

$val = diameter::InternalVarMap::get($internalVarMap, $iv_ref);
```

Enables setting values to and retrieving values from an internal variable, where `$iv_ref` and `$internalVarMap` are initialized as shown before.

10.2.2 Diameter Transaction Context Variables

The Diameter transaction context variables offer Diameter transaction persistent storage, similar to Internal Variables. Unlike Internal Variables, Diameter transaction context variables are not configurable via the GUI (which provides for a much simpler API) and cannot be shared with other features.

Purpose: Store Diameter transaction context variables

Prototype:

```
$err = dca::transctx::store("<var_id>", $var)
```

The function shall return `undef` if `$var` is `undef` or any error occurs (e.g. `$var` is a Perl hash or array that cannot be successfully encoded into JSON or DSR cannot allocate more memory space for the

Diameter context variable) and 1 if the operation is successful.

Purpose: Retrieve Diameter transaction context variables

Prototype:

```
$var = dca::transctx::fetch("<var_id>");
```

`undef` will be returned in case of failure (e.g. `<var_id>` is not found because no variable with that name has been previously stored).

10.3 Read DCA App Configuration Data

This API enables a DCA App to access its configuration data which was specified and provisioned as described in sections 3.3.3 and 3.3.4.

When the Perl script is generated, the DCA App configuration data is converted into a Perl variable. The Perl variable name is `%dca::appConfig` and is a hash (one key for each table) of arrays (one index for each record) of hashes (one key for each field in the table).

Read-only access on the DCA App configuration data is enforced using the Const::Fast CPAN module and applies to the data included in the `%dca::appConfig` definition (which is automatically generated from the DCA App configuration data).

Note that there are semantical differences from one Const::Fast version to another, which affect the way `%dca::appConfig` can be subsequently manipulated in the Perl script with regard to adding new records to `%dca::appConfig` or accessing records that are not defined in `%dca::appConfig`.

For instance, in version 0.006, which is the one currently used, an attempt to read or assign a value to an inexistent table (outermost hash key) `%dca::appConfig` will result in a runtime error.

On the other hand, assigning values to inexistent indexes (table records) and/or inexistent fields (innermost hash key) will succeed and can be subsequently successful read, while reading from inexistent indexes and/or inexistent fields will return `undef`. These indexes and fields will not be written back to the DCA App configuration data.

Purpose: Read the DCA App configuration data

Prototype:

```
$dca::appConfig{"<config_table_name>"}[<row_index>]{"<field_name>"}
```

for non-"single row" configuration tables,

```
$dca::appConfig{"<config_table_name>"}{"<field_name>"}
```

for "single row" configuration tables.

Example: Assuming a DCA App defines a configuration table called "MyTable" with two fields "FieldA" and "FieldB" and provisions a few rows, it shall be possible to retrieve the NOAM and SOAM provisioned data from the DCA app in the following way:

```
for $i (0 .. $#dca::appConfig{"MyTable"} ) {
    dca::application::logInfo($dca::appConfig{"MyTable"}[$i]{"Field1"});
    dca::application::logInfo($dca::appConfig{"MyTable"}[$i]{"Field2"});
}
```

10.4 Routing API

The routing API enables a DCA App to perform some basic routing functions.

The `dca::action::forward()`, `dca::action::answer($ans)` and `dca::action::drop()` API functions terminate the execution of the event handler. This means that the statements that follow them in the Perl code are not executed. This also has a side effect on the UDR queries initiated before invoking any of `dca::action::forward()`, `dca::action::answer($ans)` and `dca::action::drop()` because, as mentioned in **Error! Reference source not found.**, the UDR q ueries are actually sent after the execution of the event handler completes: the side effect is therefore that the UDR queries will be also not executed (i.e. sent to the UDR).

Besides `dca::action::forward()`, `dca::action::answer($ans)` and `dca::action::drop()`, an event handler's execution flow also terminates (as any other Perl subroutine) when a `return` statement is encountered or when the enclosing curly bracket is reached. In this case the implicit routing decision that the DCA framework takes is the one configured for the runtime error behavior (section 3.3.1) even though this situation need not necessary be an error condition. However, if the value returned from the event handler is negative, the "DCA Runtime Errors" alarm (Alarm ID 33304) will also be raised.

Purpose: Complete the processing and drop the message

Prototype:

```
dca::action::drop();
```

Note: Invoking `dca::action::drop()` causes the event handler to immediately terminate execution.

Purpose: Built a Diameter Answer

Prototype:

```
$ans = new dca::application::answer(<error_code>, <error_text>,
<vendor_id>);
```

The function shall return `undef` in case of failure, or a `diameter::Message` object.

When receiving a Diameter request or answer this API function enables a DCA App to construct a Diameter answer and either return it to the originator of the corresponding Diameter request or, respectively, substitute the original Diameter answer message.

The EDL API (see section 10.1) may be used to further process the `$ans` Diameter answer (e.g. add more AVPs).

Purpose: Send a Diameter Answer Created by the DCA App

Prototype:

```
dca::action::answer($ans);
```

Note: Invoking `dca::action::answer($ans)` causes the event handler to immediately terminate execution.

Purpose: Complete the processing and pass the message

Prototype:

```
dca::action::forward();
```

Enables a DCA App to pass a Diameter message to the Diameter Routing Layer for routing.

Note: Invoking `dca::action::forward()` causes the event handler to immediately terminate execution.

Purpose: Specify an ART based on which a Diameter request shall be routed

Prototype:

```
$err = dca::route::setART(<ART_table_name>);
```

The function shall return `undef` if the name of the ART does not exist (failure) or 1 if success.

Before invoking `dca::action::forward()` on a Diameter request, this routing API function enables a DCA App to specify which ART to be used for routing the respective Diameter request.

Note: The ART is configurable at the B level, therefore the `<ART_table_name>` must be configured on all the sites. Otherwise, the API function will fail when invoked on those DA-MP located in sites where `<ART_table_name>` does not exist.

Purpose: Specify a PRT based on which a Diameter request shall be routed

Prototype:

```
$err = dca::route::setPRT(<PRT_table_name>);
```

The function shall return `undef` if the name of the PRT does not exist (failure) or 1 if success.

Before invoking `dca::action::forward()` on a Diameter request, this routing API function enables a DCA App to specify which PRT to be used for routing the respective Diameter request.

Note: The PRT is configurable at the B level, therefore the `<PRT_table_name>` must be configured on all the sites. Otherwise, the API function will fail when invoked on those DA-MP located in sites where `<PRT_table_name>` does not exist.

10.5 Debugging API

The Debugging API allows tracking the execution of the event handlers by supporting the equivalent of "printf", "log", "echo", etc. functions in other programming/scripting languages.

The messages are logged in the **dsr.DCA** trace file (use tr.tail dsr.DCA). The following masks may be applied using the **tr.set** command to filter the ERROR, INFO and WARNING error messages: 0x00000001 (error), 0x00000002 (info) and respectively 0x00000004 (warning).

All the traces generated by a DCA app using the API calls will be prefixed with the DCA application name (in order to allow for further filtering e.g. using the grep utility).

Note however that in a production network DSR logs only the vital traces are therefore the main debugging tool for DCA Apps in production networks is the IDIH feature.

Purpose: Retrieve the application name

Prototype:

```
$appname = dca::application::getAppName();
```

Purpose: Retrieve the version name

Prototype:

```
$vername = dca::application::getVersionName();
```

Note: Besides debugging, another possible use case for reading the version name is including it in the DCA app state stored on the UDR. This will support backward compatibility in case the DCA app frequently changes the format of the DCA app across DCA app versions.

Purpose: Retrieve the current state

Prototype:

```
$verstate = dca::application::getState();
```

Note: The states returned can be either Trial or Production, since these are the only states when the DCA App is executed.

Purpose: Generate a trace containing user-defined messages and having a severity of INFO

Prototype:

```
dca::application::logInfo(<message>);
```

The user-defined messages shall be logged into dsr.DCA (tr.tail dsr.DCA).

Purpose: Generate a trace containing user-defined messages and having a severity of WARNING

Prototype:

```
dca::application::logWarn(<message>);
```

Purpose: Generate a trace containing user-defined messages and having a severity of ERROR

Prototype:

```
dca::application::logError(<message>);
```

10.6 Custom MEAL API

Once the Custom MEAL objects are differentiated from the **Main Menu: DCA Framework→<DCA App Name>→Custom MEALs** screen (see section 9.2.2), they can be initialized and used from DCA Apps.

10.6.1 Counter Template API

Purpose: A DCA App shall be able to bind to a Scalar Counter Custom MEAL by referring to it by the Custom MEAL configured name:

Prototype:

```
my $all_Cnt = new dca::meal::counter("MyCnt");
```

where `"MyCnt"` is the name specified when differentiating a Custom MEAL template of type "Counter" and measurement type "scalar".

The API call shall return a valid Custom MEAL object in case of success. The Custom MEAL object may be used in subsequent API calls to perform specific operations on the Scalar Counter.

In case of failure `undef` shall be returned. Possible failure cases are:

- No Custom MEAL with the specified name is currently defined;
- A Custom MEAL with that name exists, but either the differentiation process is not yet completed, or the un-differentiation process was initiated;
- A Custom MEAL with that name exists, but it is not a Scalar Counter.

Note: As a matter of best practice, the initialization of the Custom MEAL objects shall be performed in the main body of the Perl script, which is executed once right after a successful compilation (rather than in an event handler):

```
die „Custom MEAL differentiation failure"
                  unless $obj = new dca::meal::<TemplateType>("MyCustomMeal");
```

This ensures that a compilation error will be triggered if the binding process has failed, for instance because there is a name mismatch between the Perl script and the differentiation GUI screen. Using an undefined `$obj` later in the event handlers will trigger run-time errors.

Purpose: A DCA App shall be able to peg a Scalar Counter Custom MEAL:

Prototype:

```
$err = $all_Cnt->peg();
```

where `$all_Cnt` shall be a valid Scalar Counter Custom MEAL object.

The API call shall return 1 if success and `undef` if the operation on the underlying Comcol object has failed.

Purpose: A DCA App shall be able to bind to an Arrayed Counter Custom MEAL by referring to it by the Custom MEAL configured name:

Prototype:

```
my $per_Cnt = new dca::meal::arrayedCounter("MyArrayedCnt");
```

where `"MyArrayedCnt"` is the name specified when differentiating a Custom MEAL template of type "Counter" and measurement type "arrayed".

The API call shall return a valid Custom MEAL object in case of success. The Custom MEAL object may be used in subsequent API calls to perform specific operations on the Arrayed Counter.

In case of failure `undef` shall be returned. Possible failure cases are:

- No Custom MEAL with the specified name is currently defined;
- A Custom MEAL with that name exists, but either the differentiation process is not yet completed, or the un-differentiation process was initiated;
- A Custom MEAL with that name exists, but it is not an Arrayed Counter.

Purpose: A DCA App shall be able to peg a specific index of an Arrayed Counter Custom MEAL:

Prototype:

```
$err = $per_Cnt->peg($index);
```

where `$per_Cnt` shall be a valid Arrayed Counter Custom MEAL object and `$index` is the index to be pegged.

The API call shall return 1 if success and `undef` if the either operation on the underlying Comcol object has failed or the index value is negative.

10.6.2 Rate Template

Purpose: A DCA App shall be able to bind to a Scalar Rate Custom MEAL by referring to it by the Custom MEAL configured name:

Prototype:

```
my $all_Rate = new dca::meal::rate("MyRate");
```

where `"MyRate"` is the name specified when differentiating a Custom MEAL template of type “Rate” and measurement type “scalar”.

The API call shall return a valid Custom MEAL object in case of success. The Custom MEAL object may be used in subsequent API calls to perform specific operations on the Scalar Rate.

In case of failure `undef` shall be returned. Possible failure cases are:

- No Custom MEAL with the specified name is currently defined;
- A Custom MEAL with that name exists, but either the differentiation process is not yet completed, or the un-differentiation process was initiated;
- A Custom MEAL with that name exists, but it is not a Scalar Rate.

Purpose: A DCA App shall be able to peg a Scalar Rate Custom MEAL:

Prototype:

```
$err = $all_Rate->peg();
```

where `$all_Rate` shall be a valid Scalar Rate Custom MEAL object.

The API call shall return 1 if success and `undef` if the operation on the underlying Comcol object has failed.

Purpose: A DCA App shall be able to read the current value of a Scalar Rate Custom MEAL:

Prototype:

```
$val = $all_Rate->readRate();
```

where `$all_Rate` shall be a valid Scalar Rate Custom MEAL object.

The API call shall return an integer representing the current value in case of success and `undef` if the operation on the underlying Comcol object has failed.

Purpose: A DCA App shall be able to read the average value of a Scalar Rate Custom MEAL:

Prototype:

```
$val = $all_Rate->readAvgRate();
```

where `$all_Rate` shall be a valid Scalar Rate Custom MEAL object.

The API call shall return an integer representing the average value in case of success and `undef` if the operation on the underlying Comcol object has failed.

Purpose: A DCA App shall be able to determine the current severity of the alarm associated to an Scalar Rate template:

Prototype:

```
$err = $all_Rate->getSeverity();
```

where `$all_Rate` shall be a valid Scalar Rate Custom MEAL object.

The API call shall return:

- `dca::meal::Critical`, `dca::meal::Major`, `dca::meal::Minor`, `dca::meal::Cleared`
- `undef` if the operation on the underlying Comcol object has failed.

Note: The severity values are defined as:

```
use constant {
    Cleared     => 0,
    Info  => 1,
    Minor => 2,
    Major => 3,
    Critical    => 4,
};
```

which enables comparing them. For instance:

```
if ($all_Rate->getSeverity() >= dca::meal::Major)
```

will be true if the severity is Major or Critical and will be false if the severity if Minor. This also applies to Basic as well as arrayed templates.

Purpose: A DCA App shall be able to bind to an Arrayed Rate Custom MEAL by referring to it by the Custom MEAL configured name:

Prototype:

```
my $per_Rate = new dca::meal::arrayedRate("MyArrayedRate");
```

where `"MyArrayedRate"` is the name specified when differentiating a Custom MEAL template of type "Rate" and measurement type "arrayed".

The API call shall return a valid Custom MEAL object in case of success. The Custom MEAL object may be used in subsequent API calls to perform specific operations on the Arrayed Rate.

In case of failure `undef` shall be returned. Possible failure cases are:

- No Custom MEAL with the specified name is currently defined;
- A Custom MEAL with that name exists, but either the differentiation process is not yet completed, or the un-differentiation process was initiated;
- A Custom MEAL with that name exists, but it is not an Arrayed Rate.

Purpose: A DCA App shall be able to peg a specific index of an Arrayed Rate Custom MEAL:

Prototype:

```
$err = $per_Rate->peg($index);
```

where `$per_Rate` shall be a valid Arrayed Rate Custom MEAL object and `$index` is the index to be pegged.

The API call shall return 1 if success and `undef` if either the operation on the underlying Comcol object has failed or the index value is negative.

Purpose: A DCA App shall be able to read the current value of a specific index of an Arrayed Rate Custom MEAL:

Prototype:

```
$val = $per_Rate->readRate($index);
```

where `$per_Rate` shall be a valid Arrayed Rate Custom MEAL object and `$index` is the index the current value of which shall be read.

The API call shall return an integer representing the current value of the specified index in case of success and `undef` if either the operation on the underlying Comcol object has failed or the index value is negative.

Purpose: A DCA App shall be able to read the average value of a specific index of an Arrayed Rate Custom MEAL:

Prototype:

```
$val = $per_Rate->readAvgRate($index);
```

where `$per_Rate` shall be a valid Arrayed Rate Custom MEAL object and `$index` is the index the average value of which shall be pegged.

The API call shall return an integer representing the average value of the specified index in case of success and `undef` if either the operation on the underlying Comcol object has failed or the index value is negative.

Purpose: A DCA App shall be able to determine the current severity of the alarm associated to an Arrayed Rate template:

Prototype:

```
$err = $per_Rate->getSeverity($index);
```

where `$per_Rate` shall be a valid Arrayed Rate Custom MEAL object and `$index` identifies the particular index the alarm status of which shall be read.

The API call shall return:

- `dca::meal::Critical`, `dca::meal::Major`, `dca::meal::Minor`, `dca::meal::Cleared`
- `undef` if either the operation on the underlying Comcol object has failed or the index value is negative.

10.6.3 Basic Template

Purpose: A DCA App shall be able to bind to a Scalar Basic Custom MEAL by referring to it by the Custom MEAL configured name:

Prototype:

```
my $all_Size = new dca::meal::basic(“MyBasic");
```

where `“MyBasic”` is the name specified when differentiating a Custom MEAL template of type “Basic” and measurement type “scalar”.

The API call shall return a valid Custom MEAL object in case of success. The Custom MEAL object may be used in subsequent API calls to perform specific operations on the Scalar Basic template.

In case of failure `undef` shall be returned. Possible failure cases are:

- No Custom MEAL with the specified name is currently defined;
- A Custom MEAL with that name exists, but either the differentiation process is not yet completed, or the un-differentiation process was initiated;
- A Custom MEAL with that name exists, but it is not a Scalar Basic.

Purpose: A DCA App shall be able to set the value of a Scalar Basic Custom MEAL:

Prototype:

```
$err = $all_Size->setValue($value);
```

where `$all_Size` shall be a valid Scalar Basic Custom MEAL object and `$value` is the value the Scalar Basic Custom MEAL shall be set to.

The API call shall return 1 if success and `undef` if the operation on the underlying Comcol object has failed.

Purpose: A DCA App shall be able to increment the value of a Scalar Basic Custom MEAL:

Prototype:

```
$err = $all_Size->increment($count);
```

where `$all_Size` shall be a valid Scalar Basic Custom MEAL object and `$count` is the value the Scalar Basic Custom MEAL shall be incremented with.

The API call shall return 1 if success and `undef` if the operation on the underlying Comcol object has failed.

Purpose: A DCA App shall be able to decrement the value of a Scalar Basic Custom MEAL:

Prototype:

```
$err = $all_Size->decrement($count);
```

where `$all_Size` shall be a valid Scalar Basic Custom MEAL object and `$count` is the value the Scalar Basic Custom MEAL shall be decremented with.

The API call shall return 1 if success and `undef` if the operation on the underlying Comcol object has failed.

Purpose: A DCA App shall be able to read the current value of a Scalar Basic Custom MEAL:

Prototype:

```
$val = $all_Size->getValue();
```

where `$all_Size` shall be a valid Scalar Basic Custom MEAL object.

The API call shall return an integer representing the current value in case of success and `undef` if the operation on the underlying Comcol object has failed.

Purpose: A DCA App shall be able to read the average value of a Scalar Basic Custom MEAL:

Prototype:

```
$val = $all_Size->getAvgValue();
```

where `$all_Size` shall be a valid Scalar Basic Custom MEAL object.

The API call shall return an integer representing the average value in case of success and `undef` if the operation on the underlying Comcol object has failed.

Purpose: A DCA App shall be able to determine the current severity of the alarm associated to an Scalar Basic template:

Prototype:

```
$err = $all_Size->getSeverity();
```

where `$all_Size` shall be a valid Scalar Basic Custom MEAL object.

The API call shall return:

- `dca::meal::Critical`, `dca::meal::Major`, `dca::meal::Minor`, `dca::meal::Cleared`
- `undef` if the operation on the underlying Comcol object has failed.

Purpose: A DCA App shall be able to bind to an Arrayed Basic Custom MEAL by referring to it by the Custom MEAL configured name:

Prototype:

```
my $per_Size = new dca::meal::arrayedBasic(“MyArrayedBasic");
```

where `“MyArrayedBasic”` is the name specified when differentiating a Custom MEAL template of type “Basic” and measurement type “arrayed”.

The API call shall return a valid Custom MEAL object in case of success. The Custom MEAL object may be used in subsequent API calls to perform specific operations on the Arrayed Basic template.

In case of failure `undef` shall be returned. Possible failure cases are:

- No Custom MEAL with the specified name is currently defined;
- A Custom MEAL with that name exists, but either the differentiation process is not yet completed, or the un-differentiation process was initiated;
- A Custom MEAL with that name exists, but it is not an Arrayed Basic.

Purpose: A DCA App shall be able to set the value of an Arrayed Basic Custom MEAL:

Prototype:

```
$err = $per_Size->setValue($value, $index);
```

where `$per_Size` shall be a valid Arrayed Basic Custom MEAL object, `$index` is the index the value of which shall be set and `$value` is the value it shall be set to.

The API call shall return 1 if success and `undef` if either the operation on the underlying Comcol object has failed or the index value is negative.

Purpose: A DCA App shall be able to increment the value of an Arrayed Basic Custom MEAL:

Prototype:

```
$err = $per_Size->increment($count, $index);
```

where `$per_Size` shall be a valid Arrayed Basic Custom MEAL object, `$index` is the index the value of which shall be incremented and `$count` is the value it shall be incremented with.

The API call shall return 1 if success and `undef` if either the operation on the underlying Comcol object has failed or the index value is negative.

Purpose: A DCA App shall be able to decrement the value of an Arrayed Basic Custom MEAL:

Prototype:

```
$err = $per_Size->decrement($count, $index);
```

where `$per_Size` shall be a valid Arrayed Basic Custom MEAL object, `$index` is the index the value of which shall be decremented and `$count` is the value it shall be decremented with.

The API call shall return 1 if success and `undef` if either the operation on the underlying Comcol object has failed or the index value is negative.

Purpose: A DCA App shall be able to read the current value of an Arrayed Basic Custom MEAL:

Prototype:

```
$val = $per_Size->getValue($index);
```

where `$per_Size` shall be a valid Arrayed Basic Custom MEAL object and `$index` is the index the value of which shall be read.

The API call shall return an integer representing the current value of the specified index in case of success and `undef` if either the operation on the underlying Comcol object has failed or the index value is negative.

Purpose: A DCA App shall be able to read the average value of an Arrayed Basic Custom MEAL:

Prototype:

```
$val = $per_Size->getAvgValue($index);
```

where `$per_Size` shall be a valid Arrayed Basic Custom MEAL object and `$index` is the index the average value of which shall be read.

The API call shall return an integer representing the average value of the specified index in case of success and `undef` if either the operation on the underlying Comcol object has failed or the index value is negative.

Purpose: A DCA App shall be able to determine the current severity of the alarm associated to an Arrayed Basic template:

Prototype:

```
$err = $per_Size->getSeverity($index);
```

where `$per_Size` shall be a valid Arrayed Basic Custom MEAL object and `$index` identifies the particular index the alarm status of which shall be read.

The API call shall return:

- `dca::meal::Critical`, `dca::meal::Major`, `dca::meal::Minor`, `dca::meal::Cleared`
- `undef` if either the operation on the underlying Comcol object has failed or the index value is negative.

10.6.4 Event Template

Purpose: DCA App shall be able to bind to an Event Custom MEAL by referring to it by the Custom MEAL configured name:

Prototype:

```
my $errorEvent = new dca::meal::event("MyEvent");
```

where `"MyEvent"` is the name specified when differentiating a Custom MEAL template of type "Event".

The API call shall return a valid Custom MEAL object in case of success. The Custom MEAL object may be used in subsequent API calls to perform specific operations on the Event.

In case of failure `undef` shall be returned. Possible failure cases are:

- No Custom MEAL with the specified name is currently defined;
- A Custom MEAL with that name exists, but either the differentiation process is not yet completed, or the un-differentiation process was initiated;
- A Custom MEAL with that name exists, but it is not a Event.

Purpose: A DCA App shall be able to generate an event (Info severity), raise an alarm (Minor, Major, Critical severity) and clear an alarm (Clear severity):

Prototype:

```
$err = $errorEvent->log($severity, $addInfoText);
```

where `$errorEvent` shall be a valid Event Custom MEAL object, `$severity` is one of the possible values (`dca::meal::Critical`, `dca::meal::Major`, `dca::meal::Minor`, `dca::meal::Cleared`) and `$addInfoText` is the text that should be included in the alarm's additional information field.

The API call shall return 1 if success and `undef` if the operation on the underlying Comcol object has failed.

Purpose: A DCA App shall be able to determine whether an event or alarm is throttled:

Prototype:

```
$err = $errorEvent->isThrottled($severity);
```

where `$errorEvent` shall be a valid Event Custom MEAL object, `$severity` is one of the possible values (`dca::meal::Critical`, `dca::meal::Major`, `dca::meal::Minor`, `dca::meal::Info`).

The API call shall return:

- 1 if the event/alarm is throttled;
- 0 if the event/alarm is not throttled;
- `undef` if the operation on the underlying Comcol object has failed.

Purpose: A DCA App shall be able to determine the current severity of an event or alarm:

Prototype:

```
$err = $errorEvent->getSeverity();
```

where `$errorEvent` shall be a valid Event Custom MEAL object.

The API call shall return:

- `dca::meal::Critical`, `dca::meal::Major`, `dca::meal::Minor`, `dca::meal::Info`, `dca::meal::Cleared`
- `undef` if the operation on the underlying Comcol object has failed.

10.7 UDR API

The UDR API enables a DCA App to create, read, update and delete data in a UDR DB. As described in section **Error! Reference source not found.** the UDR API calls work asynchronously and a c allback subroutine is necessary in order to fetch the result of the query.

10.7.1 The Prototype of Queries and Query Results

This section describes the common structure of the UDR API functions and how the results of a UDR query can be retrieved in the Perl script.

Section 10.7.2 further describes the particularities of each individual UDR API function.

10.7.1.1 Specifying the Query

All the UDR API functions share a common prototype:

```
$err = dca::udr::udrInstance(<GLOBAL_UDR>)-><API_function>(
                    <key_type>, <key_data_type>, $key,
                    <value_data_type>, $value,
                    <callback_subroutine>,
                    );
```

where:

- `<GLOBAL_UDR>` is a string (a constant value or a scalar variable) containing the global name of the UDR DB the query shall be sent to.
- `<API_function>` is one of: createOrRead, update, bulkDelete ..
- `<key_type>` is typically a constant value defined by the DCA App. It distinguishes between different key types that a DCA App may use (e.g. IMSI, NAI, IP, IP_SRC, etc.). For example, the key value "fred" of type "NAI" is a different key from 66.72.65.64 of type "IP", even though they have the same binary representation;
- `<key_data_type>` is pre-defined constant that describes the data type of the key and must be one of:
  - `dca::udr::KeyDataType::BCD` – the key shall be a scalar,
  - `dca::udr::KeyDataType::UINT32` – the key shall be a scalar,
  - `dca::udr::KeyDataType::UINT64` – the key shall be a scalar,
  - `dca::udr::KeyDataType::STRING` – the key shall be a scalar,
  - `dca::udr::KeyDataType::IPv4` – the key shall be a NetAddr::IP object,
  - `dca::udr::KeyDataType::IPv6` – the key shall be a NetAddr::IP object.

  Note: There is no explicit data type for float numbers, float numbers will be converted to strings.
- `$key` is a Perl variable that holds the key part of the key-value pair to be created, read, updated or deleted.
- `<value_data_type>` is pre-defined constant that describes the data type of the key and must be one of:
  - `dca::udr::StateDataType::BCD` – the key shall be a scalar,
  - `dca::udr::StateDataType::UINT32` – the key shall be a scalar,
  - `dca::udr::StateDataType::UINT64` – the key shall be a scalar,
  - `dca::udr::StateDataType::STRING` – the key shall be a scalar, an array reference or a hash reference.,

  Note: Arrays and hashes are encoded into JSON and stored in the UDR DB in string format.
  - `dca::udr::StateDataType::IPv4` – the key shall be a NetAddr::IP object,
  - `dca::idr::StateDataType::IPv6` – the key shall be a NetAddr::IP object.

  Note: There is no explicit data type for float numbers, float numbers will be converted to strings.
- `$value` is a Perl variable that holds the value part of the key-value pair to be written into the UDR (via create or update operations). Note therefore that read and delete do not specify a `$value` parameter and as a result also no `<value_data_type>` parameter;
- `<callback_subroutine>` is a string representing the name of the Perl subroutine that will be invoked by the DCA framework to deliver the query result;

The API call shall return:

- 1 if the parameters are successfully parsed and encoding into a Stack Event.
  Note that, because the API call works asynchronously, at this stage the query hasn't been sent yet, its outcome cannot be known, `$err` merely tells whether a query could be successfully built.
- `undef` if parsing or encoding the parameters fails.

10.7.1.2 Retrieving the Query Result

The result of a UDR query can be retrieved in the callback function by using the `dca::udr::result()` class. An error code will always be returned and some queries also return data (consisting of the data type along with the data itself):

1. `$err_code = dca::udr::result()->code();`
Retrieves the error code. If the error codes indicates success (`dca::udr::ResultCode::Success`) then <u>some</u> API functions also return data, which can be retrieved using the `dataType()` and `data()` methods described below.

A number of error codes are common to all UDR API functions:

- `dca::udr::ResultCode::Success` – indicates the query has successfully executed the intended operation;
- `dca::udr::ResultCode::AccessError` – an error occurred on the UDR side that prevented the query to be executed;
- `dca::udr::ResultCode::SendError` – an error occurred when attempting to send the query, typically because of ComAgent overload (ComAgent related alarms will be raised in this case);
- `dca::udr::ResultCode::MaxStateSize` – the size of either the key or the data, the DCA App attempts to look up or respectively store in the UDR DB, exceeds the configured maximum sizes (**Main Menu: DCA Framework→Configuration**, "Maximum Size of Application State" and respectively "Maximum Size of the Key" options)
- `dca::udr::ResultCode::MaxEventReached` – the maximum number of UDR queries that a Diameter message event handler is allowed to send has been exceeded (see **Main Menu: DCA Framework→<DCA App Name>→General Options**, "Max. UDR Queries per Message" option).

2. `$data_type = dca::udr::result()->dataType();`
If the result contains data, then `datatype()` will return the data type of the stored data, i.e. one of: `dca::udr::StateDataType::BCD`, `dca::udr::StateDataType::UINT32`, `dca::udr::StateDataType::UINT64`, `dca::udr::StateDataType::STRING`, `dca::udr::StateDataType::IPv4`, `dca::udr::StateDataType::IPv6`;
If the result contains <u>no data</u>, then `datatype()` will return `undef`.

3. `$data = dca::udr::result()->data();`
If the result contains data, then `data()` will return the stored data.
If the result contains <u>no data</u>, then `data()` will return `undef`.

10.7.2 The UDR API Functions

Purpose: Attempts to create a key-value record in a UDR DB or fails if a record with the same key already exists.

Prototype: (see also section 10.7.1.1)

```
$err = dca::udr::udrInstance(<GLOBAL_UDR>)->create(
                     <key_type>, <key_data_type>, $key,
                     <value_data_type>, $value,
                     <callback_subroutine>);
```

Query Results: The possible result of the create API function are described in the table below (see also section 10.7.1.2):

dca::udr::result()->code()	**dca::udr::result() ->dataType()**	**dca::udr::result() ->data()**

`dca::udr::ResultCode::Success` (The record does not exist and was created)	`undef`	`undef`

`dca::udr::ResultCode:: AccessError,`
`dca::udr::ResultCode::SendError`
`dca::udr::ResultCode::MaxStateSize`
`dca::udr::ResultCode::MaxEventReached`
	`undef`	`undef`

	`undef`	`undef`

Purpose: Creates a key-value record in a UDR DB or returns the record, if a record with the same key already exists.

Prototype: (see also section 10.7.1.1)

```
        $err = dca::udr::udrInstance(<GLOBAL_UDR>)->createOrRead(
                           <key_type>, <key_data_type>, $key,
                           <value_data_type>, $value,
                           <callback_subroutine>,
                           );
```

Query Results: The possible result of the create API function are described in the table below (see also section 10.7.1.2):

`dca::udr::result()->code()`	**`dca::udr::result()->dataType()`**	**`dca::udr::result()->data()`**
`dca::udr::ResultCode::Success` (The record does not exist and was created)	`undef`	`undef`

`dca::udr::ResultCode::DBError,`
`dca::udr::ResultCode::SendError`
`dca::udr::ResultCode::AccessError`
`dca::udr::ResultCode::MaxStateSize`
`dca::udr::ResultCode::MaxEventReached`
	`undef`	`undef`

Purpose: Reads the value associated to a key from the UDR DB, or fails if the key is not found.

Prototype: (see also section 10.7.1.1)

```
        $err = dca::udr::udrInstance(<GLOBAL_UDR>)->read(
                           <key_type>, <key_data_type>, $key,
                           <callback_subroutine>,
                           );
```

Note that no `$value` parameter is present since no value is supposed to be written into the UDR DB.

Query Results: The possible result of the create API function are described in the table below (see also section 10.7.1.2):

`dca::udr::result()->code()`	**`dca::udr::result()->dataType()`**	**`dca::udr::result()->data()`**
`dca::udr::ResultCode::Success` (The record exists and was read)	The data type of the existing record	The existing record

`dca::udr::ResultCode::DBError,`
`dca::udr::ResultCode::SendError`
`dca::udr::ResultCode::AccessError`
`dca::udr::ResultCode::MaxStateSize`
`dca::udr::ResultCode::MaxEventReached`
	`undef`	`undef`

Purpose: Attempts to update the value associated with a key in the UDR DB or fails if a record with the key could not be found.

Prototype: (see also section 10.7.1.1)

```
$err = dca::udr::udrInstance(<GLOBAL_UDR>)->update(
                     <key_type>, <key_data_type>, $key,
                     <value_data_type>, $value,
                     <callback_subroutine>);
```

Query Results: The possible result of the create API function are described in the table below (see also section 10.7.1.2):

dca::udr::result()->code()	**dca::udr::result() ->dataType()**	**dca::udr::result() ->data()**
dca::udr::ResultCode::Success (The record exists and was updated)	undef	undef
dca::udr::ResultCode::DBError, dca::udr::ResultCode::SendError dca::udr::ResultCode::AccessError dca::udr::ResultCode::MaxStateSize dca::udr::ResultCode::MaxEventReached	undef	undef

Purpose: Attempts to create or update the value associated with a key

Prototype: (see also section 10.7.1.1)

```
$err = dca::udr::udrInstance(<GLOBAL_UDR>)->createOrUpdateRequest(
                     <key_type>, <key_data_type>, $key,
                     <value_data_type>, $value,
                     <callback_subroutine>);
```

Query Results: The possible result of the create API function are described in the table below (see also section 10.7.1.2):

dca::udr::result()->code()	**dca::udr::result() ->dataType()**	**dca::udr::result() ->data()**
dca::udr::ResultCode::Success (The record exists and was successfully updated)	undef	undef
dca::udr::ResultCode::DBError, dca::udr::ResultCode::SendError dca::udr::ResultCode::AccessError dca::udr::ResultCode::MaxStateSize dca::udr::ResultCode::MaxEventReached	undef	undef

Purpose: Deletes a key-value record from the UDR DB, or fails if the key is not found.

Prototype: (see also section 10.7.1.1)

```
$err = dca::udr::udrInstance(<GLOBAL_UDR>)->bulkDelete(
                     <key_type>, <key_data_type>, $key,
                     <callback_subroutine>);
```

Note that no `$value` parameter is present since no value is supposed to be written into the UDR DB.

Query Results: The possible result of the create API function are described in the table below (see also section 10.7.1.2):

dca::udr::result()->code()	**dca::udr::result() ->dataType()**	**dca::udr::result() ->data()**
dca::udr::ResultCode::Success (The record exists and was deleted)	undef	undef
dca::udr::ResultCode::DBError,	undef	undef

`dca::udr::ResultCode::SendError`
`dca::udr::ResultCode::AccessError`
`dca::udr::ResultCode::MaxStateSize`
`dca::udr::ResultCode::MaxEventReached`		

A.1 Notes

1) Reserved keywords: initDcaVars