CHANGE LOG

Revision	Submission Date	Affected Sections or Pages	Change Summary
Initial	10/19/2023	All	Initial issue of document for ANMS v1.0
A		3.1	Added table of routes and example usage
B	05/29/2025	3.4	Added SQL database section

TABLE OF CONTENTS

TABLE OF TABLES

1 DOCUMENT OVERVIEW

1.1 IDENTIFICATION

Property	Value
Configuration ID (CI)	631.17
Element	Mission Control System (MCS)
Program Set	Asynchronous Network Management System (ANMS)
Version	1.0

1.2 PURPOSE

This document will outline the interfaces the Asynchronous Network Management System (ANMS) defines and uses to communicate with external systems. In Section 3 the 3 main user interfaces are looked at in depth with generated documentation. In Section 4 the external agent interfaces are discussed.

The supplied interfaces are intended to ensure that the systems various components can send messages and receive all necessary information to accomplish its functionality. In nominal use a user will only have to interact with the user facing web browser (HTTP user-agent) interface to accomplish the general use case of monitoring the network and commanding agents. But the system is designed with customization in mind. To that end this document can be used as a development tool for developers to create new components or replacement components for the system.

1.3 TERMINOLOGY AND NOTATION

ANMS-core	the asynchronous management system core backend server
Asynchronous Management Protocol (AMP)	The application protocol used to communicate between ANMS and its managed agents.
AMM Resource Identifier (ARI)	Identifies an AMM object
NM	Network manager, system manager using AMP

1.4 REFERENCES

Table 1: Applicable JPL Rules Documents

Title	DocID
Software Development	57653 rev 10

Table 2: Applicable MGSS Documents

Title	Document Number
MGSS Implementation and Maintenance Task Requirements	DOC-001455 rev F
ANMS Product Guide	DOC-005444
ANMS User Guide	DOC-005443

Table 3: Applicable Other Documents

Title	Document Number
Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content	IETF RFC 7231
Datagram Convergence Layers for the Delay- and Disruption-Tolerant Networking (DTN) Bundle Protocol and Licklider Transmission Protocol (LTP)	IETF RFC 7122
Bundle Protocol Version 7	IETF RFC 9171
Asynchronous Management Protocol	draft-birrane-dtn-amp-08 (ietf.org)

2 ENVIRONMENT

2.1 HARDWARE CHARACTERISTICS AND LIMITATIONS

The system run on RHEL-8 with network interfaces configured, and IP addressing, and DNS configured along with a running local firewall. The system has default ports for each system if non default ports are to be used make sure each component is updated corresponding components to ensure they can communicate, see ANMS Product Guide.

2.2 INTERFACE MEDIUM AND CHARACTERISTICS

The ANMS is deployed as a docker compose configuration so uses the container IP network(s) to manage external interfaces. The user interface (GUI and API) is not coupled to the agent messaging network in any way.

The ANMS-UI and ANMS-core are designed as pull systems whereas the transcoder and Agent interfaces are designed as push systems.

2.3 STANDARDS AND PROTOCOLS

The ANMS uses RESTful API and ASYNC API standards for messaging between external user applications and various ANMS components. The APIs have been checked with tools like swagger to ensure it conforms to the documented standards.

The ANMS also uses a draft version of the Asynchronous Management Protocol (AMP) for encoding messages and communicating with external AMP Agents.

2.4 SOFTWARE INITIALIZATION

The interfaces are initialized when the ANMS system is started.

3 USER APPLICATION PROGRAMMING INTERFACES

This section covers Application Programming Interfaces (APIs) which are ANMS user-facing, accessed either through a web browser or other HTTP user-agent. The higher-level use of the browser-based interface is described in more detail, including workflows, in the ANMS User Guide.

3.1 ANMS-CORE

The ANMS-core uses a REST interface to send and receive messages with outside components. An interactive swagger document can be accessed at {*hostname}:{anms_core_port}/docs/* at system startup. That interactive document is useful for testing functionality and validity of requests. The NM section is for interacting with the network mangers' API. A list of the REST interface is supplied below.

Alongside the RESTFUL interface, ANMS-core also uses the Transcoder's async API defined in Section 3.3. Specifically, it publishes to `*transcode/CoreFacing/Outgoing*` and subscribes to `*transcode/CoreFacing/Incoming'* to send and receive translation request.

The ANMS-core also communicates with the database using the Python SQLAlchemy engine. When a user or other component interacts with this interface the database is automatically updated so no further interaction to manually send SQL requests is necessary.

REQUEST TYPE	ROUTE	DESCRIPTION
GET	/hello	Hello World Test route should return helloworld
ARIs		
GET	/agents	Paged Registered Agents
GET	/agents/all	All Registered Agents
GET	/agents/id/{registered_agents_id}	Registered Agent By Id
GET	/agents/name/{agent_id_string}	Registered Agent By Name
GET	/agents/search/{query}	Paged Registered Agents
GET	/ari	Paged Ari
GET	/ari/all	All Ari
GET	/ari/all/display	All Ari Display
GET	/ari/id/display/{obj_metadata_id}/{obj_id}	Ari Display By Id
GET	/ari/id/{obj_metadata_id}/{obj_id}	Ari By Id
GET	/ari/name/display/{obj_name}	Ari Display By Name
GET	/ari/name/{obj_name}	Ari By Name

GET	/actual_objects	Paged Actual Object
GET	/actual_objects/all	All Actual Object
GET	/actual_objects/id/{obj_metadata_id}	Actual Object By Id
GET	/actual_objects/name/{obj_name}	Actual Object By Name
GET	/formal_objects	Paged Formal Object
GET	/formal_objects/all	All Formal Object
GET	/formal_objects/id/{obj_metadata_id}	Formal Object By Id
GET	/formal_objects/name/{obj_name}	Formal Object By Name
GET	/formal_objects/edd/all	All Edd Formal
GET	/formal_objects/edd/id/{obj_metadata_id}	Edd Formal By Id
GET	/formal_objects/mac/all	Mac Formal
GET	/formal_objects/mac/id/{obj_metadata_id}	Mac Formal By Id
GET	/formal_objects/rptt/all	All Rpt Formal
GET	/formal_objects/rptt/id/{obj_metadata_id}	Rpt Formal By Id
GET	/formal_objects/ctrl/all	All Control
GET	/formal_objects/ctrl/id/{obj_metadata_id}	Control By Id
GET	/formal_objects/ctrl/name/id/{obj_metadata_id}	Name Id Control
GET	/actual_parameter	Paged Actualparameter
GET	/actual_parameter/all	All Actualparameter
GET	/actual_parameter/id/{ap_spec_id}	Actual Parameter By Id
GET	/formal_parameter	Paged Formal Parameter
GET	/formal_parameter/all	All Formal Parameter
GET	/formal_parameter/id/{fp_spec_id}	Formal Parameter By Id
GET	/literal_object	Paged Literal Object
GET	/literal_object/all	All Literal Object
GET	/literal_object/id/{obj_actual_definition_id}	Literal Object By Id

Agent_Parameters		
GET	/agents/parameter/definition/page	Paged Definition
GET	/agents/parameter/definition/all	All Definition
POST	/agents/parameter/definition/send/ Send	Parameter
GET	/agents/parameter/received/page	All Received
GET	/agents/parameter/received/agent/{agent_id}	All Ari
PUT	/agents/parameter/send/{agent_id}/{agent_parameter_id}	Send Parameter
Logging		
POST	/logging	Log
POST	/logging/query	Get Logs
NM		
GET	/nm/version	Nm Get Version
GET	/nm/agents	Nm Get Agents
POST	/nm/agents	Nm Register Agent
PUT	/nm/agents/idx/{idx}/hex	Nm Put Hex Idx
PUT	/nm/agents/eid/{eid}/hex	Nm Put Hex Eid
PUT	/nm/agents/eid/{addr}/clear_reports	Nm Clear Reports
PUT	/nm/agents/eid/{addr}/clear_tables	Nm Clear Tables
GET	/nm/agents/eid/{addr}/reports/hex	Nm Get Reports Hex
GET	/nm/agents/eid/{addr}/reports	Nm Get Reports
GET	/nm/agents/eid/{addr}/reports/text	Nm Get Reports Text
GET	/nm/agents/eid/{addr}/reports/json	Nm Get Reports Json
GET	/nm/agents/eid/{addr}/reports/debug	Nm Get Reports Debug
GET	/nm/agents/{addr}	Nm Get Agents Info
PUT	/nm/agents/eid/{addr}/reports/clear	Nm Put Clear Reports
SYS_STATUS		

GET	/sys_status/version	Sys Status Get Version
GET	/sys_status/services	Sys Status Get Services Status
POST	/sys_status/agents	Nm Register Agent
PUT	/sys_status/agents/eid/{addr}/clear_tables	Nm Clear Tables
USER		
GET	/users/{username}	Get User
PUT	/users/{username}	Update User
POST	/users	Create User
FORMAL		
GET	/formal_objects	Paged Formal Object
GET	/formal_objects/all	All Formal Object
GET	/formal_objects/id/{obj_metadata_id}	Formal Object By Id
GET	/formal_objects/name/{obj_name}	Formal Object By Name
GET	/formal_objects/edd/all	All Edd Formal
GET	/formal_objects/edd/id/{obj_metadata_id}	Edd Formal By Id
GET	/formal_objects/mac/all	Mac Formal
GET	/formal_objects/mac/id/{obj_metadata_id}	Mac Formal By Id
GET	/formal_objects/rptt/all	All Rpt Formal
GET	/formal_objects/rptt/id/{obj_metadata_id}	Rpt Formal By Id
GET	/formal_objects/ctrl/all	All Control
GET	/formal_objects/ctrl/id/{obj_metadata_id}	Control By Id
GET	/formal_objects/ctrl/name/id/{obj_metadata_id}	Name Id Control
REPORTS		
GET	/report/all	Paged Report
GET	/report/name/all	All Report Name stored
GET	/report/entry/name/{agent_id}	Get all submitted reports from agent *agent_id*

GET	/report/entries/table/{agent_id}/{adm}/{report_name}	Get report entries for report *adm.report_name* for *agent_id*
Transcoder		
GET	/transcoder/db/all	Get all transcoder log entries
GET	/transcoder/db/search/{query}	Paged Transcoder Log
GET	/transcoder/db/id/{id}	Get transcoder log entry for transaction with id *id*
PUT	/transcoder/ui/incoming/{cbor}/hex	Submit CBOR to transcoder to be translated
PUT	/transcoder/ui/incoming/str	Submit string to transcoder to be translated
ADM		
GET	/adms/ Getall	List all stored ADMs
POST	/adms/ Update	Uplaode new/ update old Adm
GET	/adms/{adm_enum}	Get info of ADM with enum *adm_enum*
DELETE	/adms/{adm_enum}	Remove Adm with enum *adm_enum*
alerts		
GET	/alerts/incoming	List of alerts for changes to registered agents
PUT	/alerts/acknowledge/{index}	Remove alert from list

3.1.1 USING THE CORE INTERFACE

Below is an example of using the Core's endpoints to send a command to a manger and receive back a report. The ANMS UI stream lines this procedure by combining calls and simplifying the creation of ARI strings. The following curl commands assume the core is accessible at the {*hostname}:{anms_core_port }*

The first step is to translate a gen_rpts ARI string,

ari:/IANA:amp_agent/CTRL.gen_rpts([ari:/IANA:bpsec/RPTT.full_report],["ipn:1.7"]), to CBOR to send to a manager. ARI string can be generated using the ANMS UI build tab.

```
curl -X 'PUT' \
  'http://
{hostname}:{anms_core_port}/transcoder/ui/incoming/str?ari=%22ari%3A%2FIANA%3
Aamp_agent%2FCTRL.gen_rpts%28%5Bari%3A%2FIANA%3Abpsec%2FRPTT.full_report%5D%2
C%5B%22ipn%3A1.7%22%5D%29%22' \
  -H 'accept: application/json'
```

This command sends the string to the transcoder to be encoded into cbor. This process is asynchronous so this PUT command returns an ID that can be used to retrieve the result once the translation is done. In this example this command return *527* as the id of this transaction

Then the transition id *527* can be used with the *transcoder/db/id* GET command to receive a transcoder response object.

```
curl -X 'GET' \
  'http://{hostname}:{anms_core_port}/transcoder/db/id/527' \
  -H 'accept: application/json'
```

The transcoder response object will have a CBOR field that stores the CBOR translation of the ARI command.

```
{
  "transcoder_log_id": 527,
  "input_string":
"ari:/IANA:amp_agent/CTRL.gen_rpts([ari:/IANA:bpsec/RPTT.full_report],[\"ipn:
1.7\"])",
  "parsed_as": "URI",
  "ari": "{}",
  "uri":
"ari:/IANA:amp_agent/CTRL.gen_rpts([ari:/IANA:bpsec/RPTT.full_report],[\"ipn:
1.7\"])",
  "cbor": "0xC11541050502252381871 8CD41000501126769706E3A312E37"
}
```

This CBOR can then be sent to the agent using *nm/agents/eid/{}/hex*. In this example we are sending the CBOR to the endpoint *ipn:2.6.*

```
curl -X 'PUT' \
  'http://{hostname}:{anms_core_port}/nm/agents/eid/ipn%3A2.6/hex' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "data": "0xC11541050502252381871 8CD41000501126769706E3A312E37"
}'
```

Once commands are sent to agents and reports are generated the following commands can be used to retrieve the reported information.

The command below will list all report templates agent ipn:2.6 has generated.

```
curl -X 'GET' \
  'http://{hostname}:{anms_core_port}/report/entry/name/ipn%3A2.6' \
  -H 'accept: application/json'
```

The command */report/entries/table/{agent_id}/{adm}/{report_name}* can be used to return a JSON object (an example of the output shown below) containing all reports received for a given *adm.report_name* template, *amp_agent.full_report* in this example. Note that the first entry is the list name of each report values

```
curl -X 'GET' \
  'http://
{hostname}:{anms_core_port}/report/entries/table/ipn%3A2.6/amp_agent/full_rep
ort' \
  -H 'accept: application/json'
```


```
[
  [
    "time",
    "name",
    "version",
    "num_rpt_tpls",
    "num_tbl_tpls",
    "sent_reports",
    "num_tbr",
    "run_tbr",
    "num_sbr",
    "run_sbr",
    "num_const",
    "num_var",
    "num_macros",
    "run_macros",
    "num_controls",
    "run_controls",
    "num_rules"
  ],
  [
    "2024-11-26 14:16:14",
    "amp_agent",
    "v3.1",
    "4",
    "20",
    "30",
    "0",
    "30",
    "0",
    "0",
    "34",
    "1",
    "0",
    "0",
```

```
    "0",
    "33",
    "0"
  ],
  [
    "2025-02-17 18:50:59",
    "amp_agent",
    "v3.1",
...
```


Above is an example JSON object storing received reports.

To recap the routes used for sending a control to generate a report then display the received report is as follows:

transcoder/ui/incoming/str?ari=%22ari%3A%2FIANA%3Aamp_agent%2FCTRL.gen_rpts%28%5Bari%3A%2FIANA%3Abpsec%2FRPTT.full_report%5D%2C%5B%22ipn%3A1.7%22%5D%29%22'

returned the ID 527.

transcoder/db/id/527

returned an object which CBOR field contained

"cbor": "0xC1154105050225238187 18CD41000501126769706E3A312E37"

nm/agents/eid/ipn%3A2.6/hex with request body

{"data": "0xC1154105050225238187 18CD41000501126769706E3A312E37"}

report/entry/name/ipn%3A2.6

used to view aviable reports, would show amp_agent.full_report is available after above commands are executed

report/entries/table/ipn%3A2.6/amp_agent/full_report

would return a JSON object containing all received *amp_agent.full_report*(s) sent by ipn:2.6

3.2 ANMS-UI

The UI uses a restful API to communicate between the user facing front end and a server backend. The backend then communicates with the core using its API as defined in Section 3.1 as such the below table references the ANMS-core's API. These routes are designed to be automatically called by the system when needed and a standard user is not expected to interact with them. They are included for completeness and for a guide for future development.

Table 4: ANMS-UI RESTful API

URI Path	Method	Description
/hello	get	Path for testing API
/core/service_status	get	Get status of the system services
/users/:userName	get	Get the user profile for a specific user :userName = string username
/users	post	Create a new user profile. Body of the request is a `{ "first_name": "John", "last_name": "Doe", "email": "john.doe@example.com", "username": "johndoe" }`
/users/:userName	put	Update :userName Profile Body of the request is `{ "first_name": "John", "last_name": "Doe", "email": "john.doe@example.com", "username": "johndoe" }`
/core/adms	get	Get all known ADM information
/core/adms	post	Upload a new ADM body is the new file to upload
/agents	get	Get all know agents
/agents/id/:id	get	Get information about a specific agent :id = the interger id of the agent
/agents/search/:query	get	Get information about agents that match search criteria :query
/agents/parameter/name/:id	get	Get list of agent paraments that can be controlled for agent :id = the integer id of the agent

URI Path	Method	Description
/agents/parameter/send/:id/:optId	put	Send new parameters to a agent :id = the integer id of the agent :optId = the integer id of the parameter updating Body of message `{ "data": [ "string" ] }`
/build/ari/all	get	Calls core's /ari/all/display
/build/ari/id/:meta_id/:obj_id	get	Calls core's /ari/id/display/{obj_metada_id}/{obj_id}
/transcoder/ui/incoming/:cbor/hex	put	Call core's /transcoder/ui/incoming/{cbor}/hex
/transcoder/ui/incoming/str	put	Calls core's /transcoder/ui/incoming/str
/transcoder/ui/log	get	Calls core's /transcoder/db/all
/transcoder/ui/log/:query	get	Call core's /transcoder/db/search/{query}
/nm/version	get	call core's /nm/agents
/nm/agents	post	call core's /nm/agents/idx/{idx}/hex
/nm/agents/idx/:idx/hex	put	call core's /nm/agents/eid/{eid}/hex
/nm/agents/eid/:eid/hex	put	call core's /nm/agents/eid/{addr}/clear_reports
/nm/agents/eid/:addr/clear_reports	put	call core's /nm/agents/eid/{addr}/clear_tables
/nm/agents/eid/:addr/clear_tables	put	call core's /nm/agents/eid/{addr}/reports/hex
/nm/agents/eid/:addr/reports/hex	get	call core's /nm/agents/eid/{addr}/reports
/nm/agents/eid/:addr/reports	get	call core's /nm/agents/eid/{addr}/reports/text
/nm/agents/eid/:addr/reports/text	get	call core's /nm/agents/eid/{addr}/reports/json

URI Path	Method	Description
/nm/agents/eid/:addr/reports/json	get	call core's /nm/agents/eid/{addr}/reports/debug
/nm/agents/eid/:addr/reports/debug	get	call core's /nm/agents/eid/{addr}
/nm/agents/eid/:addr	get	call core's /nm/agents/eid/{addr}/reports/clear
/nm/agents/eid/:addr/reports/clear	put	call core's /report/entry/name/{obj_agent_id}
/report/entry/name/:obj_agent_id	get	call core's /report/entries/table/{obj_agent_id}/{adm}/{report_name}
/report/entries/table/:obj_agent_id/:adm/:report_name	get	call core's /report/entries/table/{obj_agent_id}/{adm}/{report_name}

3.3 TRANSCODER

The transcoder interface is an AsyncAPI that allows the ANMS-core to continuously send translation request and listen for results back from a codec that is setup to process and translate AMP URIs. The interface has three main components the ANMS-core translation client that receives and tracks translation request via the ANMS-core's RESTAPI. The Transcoder that preforms further error handling and message forwarding and the ARICodec, the system that handles translation. ARICodec use the ACE tool to encode ARIs from/to string representation to/from CBOR representation. An MQTT client is also used to handle AsycnAPI message transport.

The system is designed in this way to decentralize processing, allowing for a straightforward way of swapping out the translation codec used. A developer would need to write a ASYNC wrapper for the new translation tool that subscribes to '*transcode/CodexFacing/Incoming'* to receive translation request messages and publish result messages to *'transcode/CodexFacing/Outgoing'.* The translation request messages are simply the ARI to be translated. Then the translation tool would process the ARIs, returning a translated message which contains the information generated from the translation process including the original input string what it was translated as and the final translated output. The full schema for the incoming and outgoing messages used are defined at *ANMS/transcoder/asyncapi.yaml.*

3.4 REF-DB

The refdb-sql is the postgres implementation of the dtnma-tools database that stores the objects defined by ADMs and the table and reports generated by agents and managers.

Each ARI collection has its own table that uses binary object to stores its entries. Each ARI object type has its own table for its actual definition (parametrized values if it contains parameters) and if its an object that can be parametrized it will have a formal definition entry as well to store the formal parameters.

The database also stores agent andmessaging information generated by the dtnma-tools system.

The database can be quickly interacted with by using that *anms-core* routines described above. For a user wishing to write their own API to interact with the database predefined database views and stored procedures have been created to give users quicker access into the database.

3.4.1 VIEWS

The below table describes the views stored in the database. Each view combines foreign key relationships of tables to give a concise table to view all the data used.

View name	Description
vw_actual_parameters	Combines the actual parameters with formal parameters definition
vw_ari	Combines the tables
vw_ari_tblt	Combines table template table with ARI collection table
vw_ari_union	Combines the all formal and actual ARI tables to give complete view of all stored ADMs
vw_const_actual	Combines metadata and actual definition table with constant actual definition table
vw_ctrl_actual	Combines metadata and actual definition table with the control actual definition table
vw_ctrl_definition	Combines metadata and formal definition table with control formal definition table
vw_edd_actual	Combines metadata and actual definition table with EDD actual definition table
vw_edd_formal	Combines metadata and formal definition table with EDD formal definition table
vw_formal_parameters	View into the formal_parmspec table
vw_ident_actual	Combines metadata and actual definition table with ident actual definition table
vw_ident_formal	Combines metadata and formal definition table with ident formal table
vw_mac_actual	Combines metadata view and actual definition table with macro actual definition table

vw_mac_definition	Combines metadata view and formal definition table with macro definition table
vw_obj_actual_def	Combines the metadata view and the actual object tables
vw_obj_formal_def	Combines metadata view and the formal object tables
vw_obj_metadata	Combines the metadata table with the data model table
vw_oper_actual	Combines metadata and formal definition table with
vw_oper_formal	Combines metadata and formal definition table with
vw_rpt_set	Combines report set table with report template table
vw_rptt	Combines report table with ari collection table
vw_sbr_actual	Combines metadata and actual definition table with
vw_tbr_actual	Combines metadata and actual definition table with time based rules actual table
vw_typedef_actual	Combines metadata and actual definition table with type def actual table
vw_var_actual	Combines metadata and actual definition table with variable actual table
vw_var_formal	Combines metadata and formal definition table with

3.4.2 STORED PROCEDURES

The table below list the stored procedures that combine several important actions into one. The main features are for inserting and deleting entries into the database.

Procedure name	Arguments	Description
sp__add_agent_parameter	p_command_name, p_command_parameters	Add a new predefined agent parameter control
sp__add_agent_parameter_received	p_manager_id, p_registered_agents_id, p_agent_parameter_id, p_command_parameters	Log that agent control was received
sp__delete_const_actual_definition	p_obj_id, p_obj_name	Given an object id or name, delete its entry from the labeled table
sp__delete_control_actual_definition	p_obj_id, p_obj_name	
sp__delete_control_formal_definition	p_obj_id, p_obj_name	
sp__delete_edd_actual_definition	p_actual_definition_id, p_obj_name	
sp__delete_edd_formal_defintion	p_obj_id, p_obj_name	
sp__delete_expression	p_expr_id	
sp__delete_mac_actual_definition	p_inst_id, p_obj_name	
sp__delete_macro_formal_defintion	p_obj_id, p_obj_name	
sp__delete_obj_actual_definition	p_act_id, p_obj_name	
sp__delete_obj_formal_definition	p_obj_id, p_obj_name	
sp__delete_obj_metadata	p_obj_id	
sp__delete_operator_formal_defintion	p_obj_id, p_obj_name	
sp__delete_sbr_actual_definition	p_obj_id, p_obj_name	
sp__delete_table_template_actual_definition	p_obj_id	
sp__delete_tbr_actual_definition	p_obj_id, p_obj_name	
sp__delete_variable_definition	p_definition_id, p_obj_name	
sp__get_delimiter_position	p_str, p_delimiter, p_start_position	
sp__insert_ac_id	p_num_entries, p_entries, p_use_desc	With the supplied data metadata

sp__insert_actual_parmspec	p_fp_spec_id, p_value_set, p_use_desc	values insert a new entry into the respective table
sp__insert_adm_defined_date_model	p_issuing_org, p_date_model_string, p_version, p_adm_name_string, p_enumeration, p_enumeration_label, p_use_desc	
sp__insert_agent	p_agent_id_string	
sp__insert_ari_rpt_set	p_correlator_nonce, p_reference_time, p_report_list, p_agent_id	
sp__insert_const_actual_definition	p_obj_id, p_use_desc, p_data_type, p_data_value_string	
sp__insert_control_actual_definition	p_obj_definition_id, p_ap_spec_id, p_use_desc	
sp__insert_control_formal_definition	p_obj_id, p_use_desc, p_fp_spec_id, p_result	
sp__insert_data_model	p_namespace_type, p_name, p_enumeration, p_version_name, p_use_desc	
sp__insert_data_model	p_namespace_type, p_name, p_enumeration, p_version_name, p_use_desc	
sp__insert_date_model	p_date_model_type, p_issuing_org, p_name_string, p_version	
sp__insert_edd_actual_definition	p_obj_definition_id, p_use_desc, p_ap_spec_id	
sp__insert_edd_formal_definition	p_obj_id, p_use_desc, p_fp_spec_id, p_external_data_type	
sp__insert_expression	p_out_type, p_postfix_operations	
sp__insert_formal_parmspec	p_num_parms, p_use_desc, p_parameters	

sp__insert_ident_actual_definition	p_obj_id, p_use_desc, p_ap_spec_id	
sp__insert_ident_formal_definition	p_obj_id, p_use_desc, p_fp_spec_id	
sp__insert_incoming_message_entry	p_set_id, p_message_order, p_start_ts, p_ac_id	
sp__insert_incoming_message_set	p_created_ts, p_modified_ts, p_state, p_agent_id	
sp__insert_macro_actual_definition	p_obj_definition_id, p_ap_spec_id, p_actual_ac, p_use_desc	
sp__insert_macro_formal_definition	p_obj_id, p_use_desc, p_fp_spec_id, p_max_call_depth, p_definition_ac	
sp__insert_message_entry_agent	p_mid, p_agent	
sp__insert_message_group_agent_id	p_group_id, p_agent	
sp__insert_message_group_agent_name	p_group_id, p_agent	
sp__insert_message_report_entry	p_msg_id, p_order_num, p_ari_id, p_tnvc_id, p_ts	
sp__insert_network_defined_date_model	p_issuing_org, p_name_string, p_version, p_issuer_binary_string, p_tag	
sp__insert_obj_actual_definition	p_obj_metadata_id, p_use_desc	
sp__insert_obj_formal_definition	p_obj_metadata_id, p_use_desc	
sp__insert_obj_metadata	p_obj_type_id, p_obj_name, p_data_model_id, p_object_enumeration, p_status, p_reference, p_description	
sp__insert_operator_actual_definition	p_obj_definition_id, p_use_desc, p_ap_spec_id	
sp__insert_operator_formal_definition	p_obj_id, p_use_desc, fp_spec_id,	

	p_num_operands, p_operands, p_result_name, p_result_type	
sp__insert_sbr_actual_definition	p_obj_id, p_use_desc, p_condition, p_action, p_min_interval, p_max_count	
sp__insert_table_template_actual_definition	p_obj_id, p_use_desc, p_columns_id	
sp__insert_tbr_actual_definition	p_obj_id, p_use_desc, p_wait_per, p_run_count, p_start_time, p_action	
sp__insert_typdef_actual_definition	p_obj_id, p_use_desc, p_data_type_id	
sp__insert_variable_actual_definition	p_obj_id, p_use_desc, p_ap_spec_id	
sp__insert_variable_formal_definition	p_obj_id, p_use_desc, p_fp_spec_id, p_out_type, p_expression, p_init_value	
sp__update_incoming_message_set	p_set_id, p_state	Update the message set with set id p_set_id state to p_state
sp__update_outgoing_message_set	p_set_id, p_state	

4 MANAGED AGENT INTERFACES

This section covers the AMP interface between the ANMS and its external managed AMP Agents.

Messaging between the ANMS and managed agents operates over the following layered protocols:

1. AMP messages following the *draft-birrane-dtn-amp-08* document
2. Bundle Protocol (BP) version 7 bundles following IETF RFC 9171
3. UDP/IP datagram transport following RFC 7122 and its dependencies

The factory configuration of the ANMS uses a single BP convergence layer (CL) which embeds bundles as individual UDP datagrams (the UDPCL). The UDPCL in the ANMS uses default UDP port 4556, and the ANMS can be configured to use other port numbers for UDPCL running on AMP Agents.