GridOps Management Suite 3.10

Manual Points Integration

Functional Specification

Document Version: 1.0

Updated: June, 2024

Table of Contents

Table of Figures

Table of Tables

1. REFERENCES

#	Title	Description
1.	EcoStruxure GridOps Management Suite 3.10 Enterprise Integration Platform - Functional Specification	The document represents a set of common integration principles applied to all baseline integration adapters.

2. ABSTRACT

EcoStruxure GridOps Management Suite is a family of solutions designed to help electric utilities in the operations and management of their grid. It is offered as EcoStruxure ADMS, EcoStruxure Grid Operation, EcoStruxure DERMS or EcoStruxure Energy Transmission Operation solutions, which share the same technology platform.

NOTE: The functionality described in this document applies to all solutions.

NOTE: Most images presented in this document are related to the EcoStruxure ADMS solution and should be used as an example. The images for other solutions may differ slightly.

The purpose of this document is to describe the exchange of the manual points values between the source and the destination system.

This integration is needed for the offers such as Standalone DERMS or Grid Operation, where the dynamic data comes from ADMS system.

3. REQUIREMENTS

ReqId	Comment
MPI-001	The integration will encompass the statuses of manual points between Source and Destination system. Manual point is the point that does not have a remote point assigned to it on the Destination system.
MPI-002	Source system will have a subset of points that it will send to the Destination system. It will be possible to define this subset in multiple ways: through configuration or by sending the list of points through exposed interface. Changes of points outside of that subset will not be processed.
MPI-003	The mapping of the signals between Source and Destination systems will be done by custom IDs of the signals, meaning that the matching signals in the two systems must have identical customIDs (case sensitive).
MPI-004	Communication between the Source adapter and Destination adapter will be secure, impemented as REST APIs.
MPI-005	Communication will always be initiated from the Source side. Destination side will never initiate communication, but will rather only provide endpoints to be invoked.
MPI-006	Integrity update mechanism will be implemented between Source and Destination adapter
MPI-007	Source adapter will only send the data to the Destination side if both adapters are on Hot and Active sites. Source adapter will only attempt to send the data if both Source and Destination side are available. In case the Destination adapter is not available, Source adapter will not attempt to send the data.
MPI-008	Destination side will perform the timestamp validation, in the sense that it will not apply the value that is older than the value already applied to the signal.
MPI-009	If the same value is sent from the Source to the Destination side, the value and timestamp of the signal on the Destination side will not be changed.
MPI-010	Single source of truth for the signal values will be the Source side. The value of a signal on the Destination side can be changed locally, but when the new value comes from the Source side, it will overwrite the value on the Destination side (if the timestamp of the new value is more recent).
MPI-011	The sequence of events during the time Source or Destination side adapters were down, will not be preserved, in that only the actual signal values at the time of integrity update will be sent.

4. OVERVIEW

4.1. Functional Design

The manual point integration is designed as a collaboration between the source and the destination adapter.

The destination adapter is a stateless adapter, which receives the payload containing the signal values, validates it, and applies the valid changes to the NDS of the hosting system. The destination adapter also provides the list of the ids for the required signals to the source side when queried.

The source adapter consists of several components, described in more detail in the Component Design chapter. The source adapter actions are synchronized through internal state machine, described in more detail in the Source Adapter State Machine chapter. The data is sent using a producer-consumer pattern. The producer (which can be NDS subscriber described in the MPI Source Internal Model chapter, or NDS reader for the integrity update, described in the MPI Source Integity Update Reader chapter) stores the data to be sent into the queue which is emptied by the consumer thread, which sends the data to the destination adapter. The source adapter also filters the data to be sent.

4.1.1. Data Exchange

The sequence diagram describing the data exchange in the normal working mode is shown in Figure 4.1.

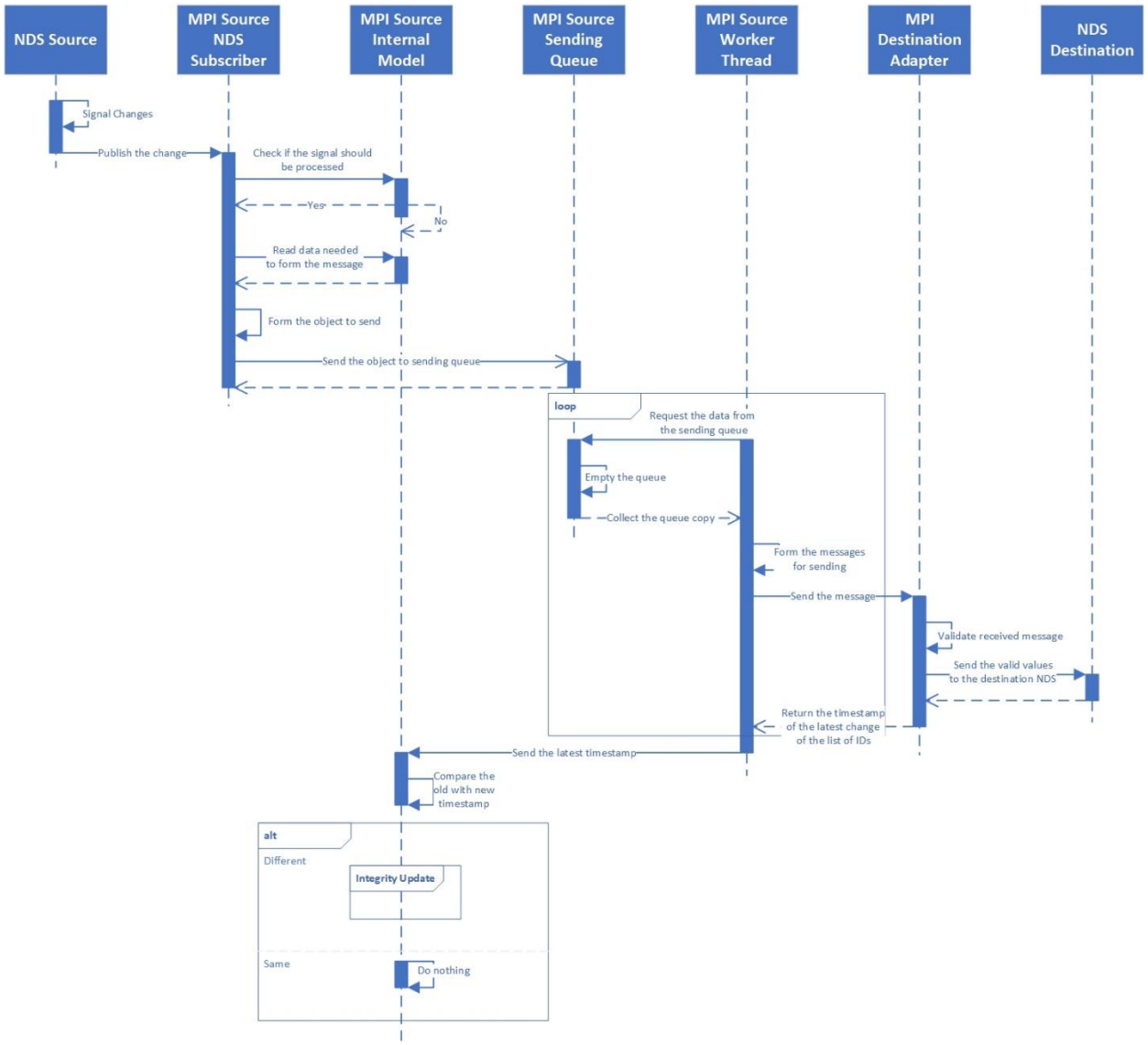

Figure 4.1 – Data Exchange in the normal working mode

4.1.2. Integrity Update

The integrity update process is described in the sequence diagram shown in Figure 4.2.

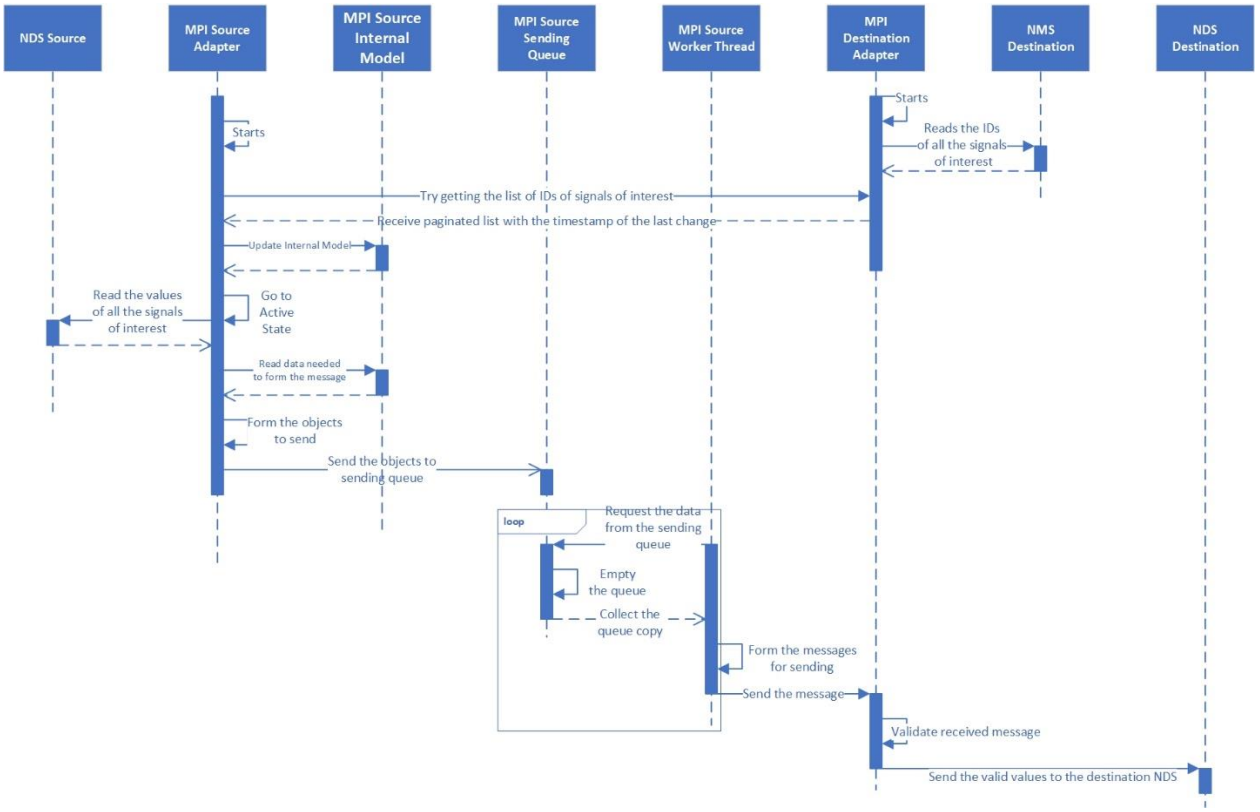

Figure 4.2 – Integrity Update

Integrity update process means that all the latest values of the up-to-date list of signals (kept in MPI Source Points Cache Filter) are sent to the sending queue and then from there to the destination side. The list of signals can be initially emptied and then updated through GetSignalIDs operation (see Get Signal Ids Operation), or can simply be used as is, in case of the communication exception with the source NDS.

4.1.3. Update of the List of Signal Ids to Send

The destination side dictates the values of which signals are to be sent by the source side. The destination side keeps the list of the signals it expects and detects if it changed by keeping the timestamp of the last change. The process of notifying the source side of the change is shown in Figure 4.3.

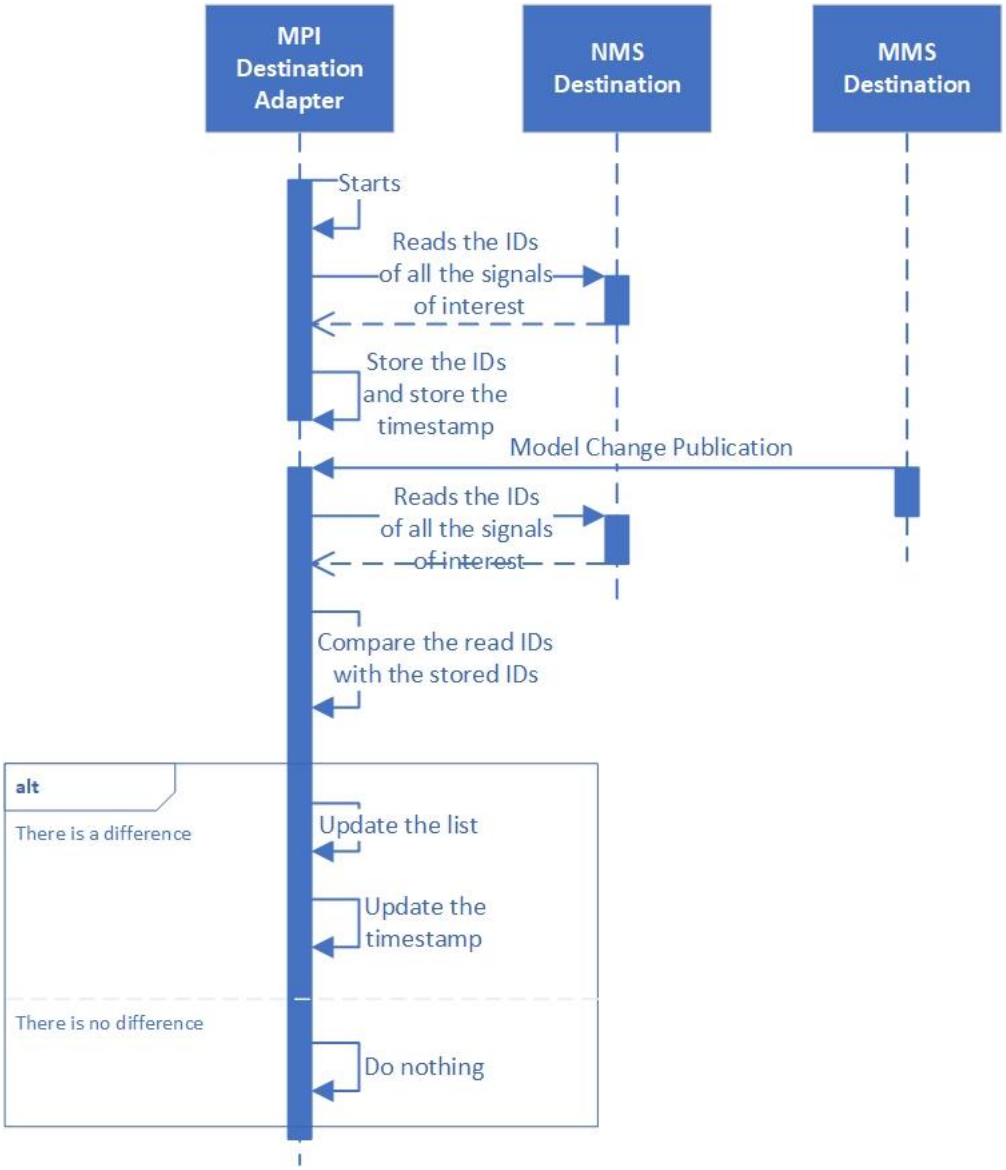

Figure 4.3 – Update the list of Signal IDs

4.1.4. Source Adapter State Machine

The source adapter will work based on the state machine.

The state machine consists of the following states:

- Standby state – the state machine is in this state when the source adapter is on the standby node. The adapter is not subscribed to the changes of the signals and does not attempt to communicate with the destination adapter.
- Inactive state – when the state machine is in this state, the source adapter is on the hot node. The adapter is not subscribed to the changes of the signals, but attempts to communicate with the destination adapter, trying to get all the signal ids from the destination adapter.
- Active state – when the state machine is in this state, the source adapter is on the hot node. The adapter is subscribed to the changes of the signals. The sender queue thread works and the signal

values obtained either through NDS IU reader or through NDS subscriber are sent to the destination side.

The events that are used to trigger the state machine are:

- StartCollectingIDsEvent – this event is fired when the node the adapter is on switches from standby to hot state.
- GetSignalsIdsOKEvent – this event is fired when all the signals IDs are collected from the destination side. Upon receiving all the signals IDs from the destination adapter, the source adapter can transit to the active state and can start IU, as well as activating the NDS subscription, sending the signals values as they change on the source side.
- SignalChangeReceivedEvent – this event is fired when the node the adapter is on is in the Hot state, so the NDSSubscriber is active. Whenever the signal changes, the NDSSubscriber fires this event, thus ensuring that the signal will only be processed in the active state of the state machine
- NDSSubscriptionExceptionEvent – this event is fired when NDSSubscriber is disconnected from the NDS, so the IU needs to be initiated. This is done through this event when the state machine is in the active state.
- FailoverEvent – this event is fired when the node the adapter is on transitions to the standby state
- ModelUpdateEvent – this event is fired when model update happens on the source side.

The state machine is described in Table 4.1.

Table 4.1 – Source Adapter State Machine

	State		
Event	**Inactive**	**Active**	**Standby**
Start Collecting IDs Event	Next State: **Inactive** Actions: • Collect the signal Ids from the destination side • Fire GetSignalsIds OK event	Next State: **Inactive** Actions: • Collect the signal Ids from the destination side • Fire GetSignalsIds OK event	Next State: **Inactive** Actions: • Collect the signal Ids from the destination side • Fire GetSignalsIds OK event
Get Signals Ids OK Event	Next State: **Active** Actions: • Start worker thread • Perform IU read • Add read values to the sender queue	Next State: **Active** Actions: N/A	Next State: **Standby** Actions: N/A
NDS Subscription Exception Event	Next State: **Inactive** Actions: N/A	Next State: **Active** • Perform IU read • Add read values to the sender queue	Next State: **Standby** Actions: N/A

	State		
Event	**Inactive**	**Active**	**Standby**
Signal Change Received Event	Next State: **Inactive** Actions: N/A	Next State: **Active** Actions: • Add the changed value to the senders queue • In case of an exception, transit to the InactiveState, collect the signal Ids from the destination side and fire Get Signals Ids OK upon finishing	Next State: **Standby** Actions: N/A
Failover Event	Next State: **Standby** Actions: • Stop sender thread • Clear sending queue	Next State: **Standby** Actions: • Stop sender thread • Clear sending queue	Next State: **Standby** Actions: N/A
Model Update Event	Next State: **Inactive** Actions: N/A	Next State: **Active** Actions: • Check and cleanup the signal Ids filter	Next State: **Standby** Actions: N/A

4.2. Architectural Design

This integration is designed as a pair of the adapters communicating with each other. The communication is always initiated from the source side, since the source side is the side with the higher security level. Both adapters are in their respective DMZ zones and are deployed in the hot-standby failover cluster configurations. The architecture of the solution is shown in Figure 4.4.

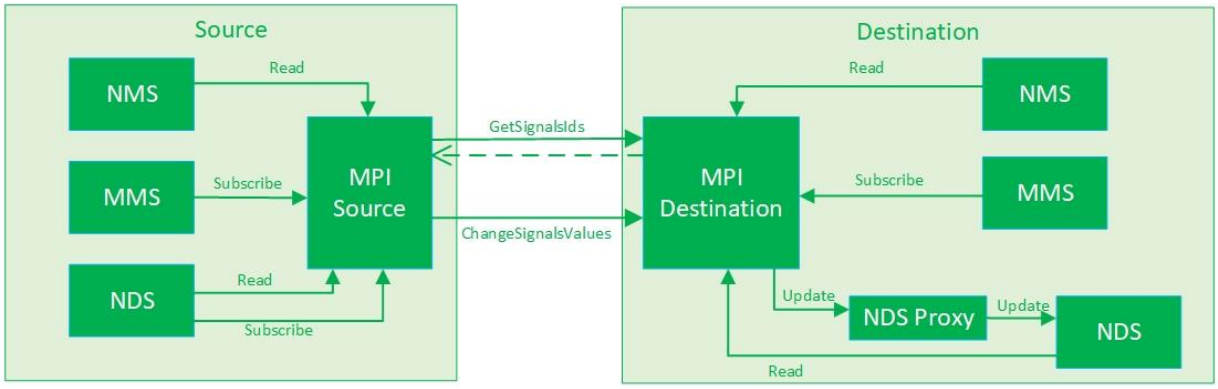

Figure 4.4 – MPI Architecture

The source adapter is subscribed to the NDS for the signal changes.

It is also subscribed to the model changes, so the signal filter can be checked for inconsistencies when the model is changed. Specifically, this is a check whether a signal that was received from the destination side has changed its customId, in which case it would be removed from the filter.

The source adapter also reads signal customIds and gids from NMS when forming the filter.

The source adapter reads signal values and timestamps from NDS during integrity update.

The destination adapter exposes two REST endpoints, as described in more details in the Change Signals Values Operation and Get Signal Ids Operation chapters. These endpoints are used by the source adapter.

The destination adapter reads customIds of the signals of interest from NMS when the adapter starts and when model changes.

The destination adapter is subscribed to the model changes, so it would refresh the internal model of the custom Ids of the signals of interest.

It also reads signal values and timestamps from NDS, so it would filter out outdated signals or signal changes that did not change signal value.

The destination adapter sends the valid signals changes to NDS via NDS Proxy Adapter in DMZ zone.

5. DETAILED DESIGN

5.1. Component Design

The components of the source and destination adapters are described in the following chapters.

5.1.1. MPI Source Internal Model

MPI Source Internal Model is a component that contains the ids of the signals that the destination side expects to be sent. This model is populated via GetSignalsIds operation (see Get Signal Ids Operation) and MPI Source NMS Filtered Query (see MPI Source NMS Queries).

5.1.2. MPI Source NMS Queries

MPI Source NMS Queries are component responsible for reading the data from NMS. There are two queries that are used on the source side. The Filtered query gets the gids of the signals for the customIds obtained via GetSignalsIds operation (Get Signal Ids Operation) and is used to populate the internal model (MPI Source Internal Model). The CustomIds query is used on model change to verify that the customIds of the signals in the internal model have not changed.

5.1.3. MPI Source NDS Subscriber

MPI Source Subscriber is the component which subscribes to the changes in the source NDS and receives the changes of the signals it is subscribed to. It then invokes MPI Source Internal model (MPI Source Internal Model) to filter the signals and form the outgoing message for the valid changes and puts it on the sender queue (MPI Source Sending Queue).

5.1.4. MPI Source Data Filter

MPI Source Data Filter is the component containing the GID to CustomId map of the signals for which the values are to be sent to the destination side.

5.1.5. MPI Source Integrity Update Reader

MPI Source Integrity Update Reader is the component responsible for reading the data from NDS. This component is used during integrity update to read the values and timestamps of the signals of interest which it obtains from the internal model (MPI Source Internal Model). It then forms the message payloads which it then returns to the caller.

5.1.6. MPI Source Sending Queue

MPI Source Sending Queue is the component which stores the changes to be sent to the destination side. This component ensures the thread safety for this data storage. It also hosts and starts the sender thread which collects the data from the MPI Source Sending Queue, transforms it in the appropriate form and sends it to the destination side using the MPI Sender Communication Component.

5.1.7. MPI Sender Communication Component

MPI Sender Communication Component is the component responsible for REST communication with the Destination adapter. This component hosts the source side APIs and encapsulates the REST clients which invoke the APIs on the destination side. The main two communication components are SignalValuesSender and SignalIdsGetter.

SignalValuesSender sends the signal values changes to the destination side. This component is also used to send the signal values to the destination side during integrity update, as the payloads are the same.

SignalIdsGetter retrieves the customIds of the signals of interest from the destination side and maps them to the gids of the corresponding signals on the source side. This map is then used to populate the internal model (MPI Source Internal Model). The important thing to mention here is that, due to the possibly high number of the signal ids, the payload is paginated, as described in Data Format. SignalIdsGetter receives the complete list, which is obtained by SignalIdsCollector. SignalIdsCollector invokes GetSignalsIds operation as long as there are nextUrls in the payload and provides the complete customIds list to SignalIdsGetter.

5.1.8. MPI Destination Internal Model

MPI Destination Internal Model is a component that contains the set of ids of signals that it the destination side will expect to be sent. It also contains the timestamp of the last time this set was updated.

5.1.9. MPI Destination NMS Reader

The NMS reader on the destination side reads all manual discrete points from NMS and returns gids and customIds to populate the internal model (MPI Destination Internal Model).

5.1.10. MPI Destination NDS Reader

The NDS reader on the destination side reads values and timestamps of the signal values received from the source side, in order to filter outdated or unchanged values.

5.1.11. MPI Destination Model Update Subscriber

The Model Update Subscriber on the destination side reacts to NMS model changes. On each change, it invokes NMS Reader (MPI Destination NMS Reader) to collect all the signals of interest and then compares the results to those in the Internal Model (MPI Destination Internal Model). If the results are the same, no changes are made. However, if the results differ from the Internal Model, then the Internal Model is updated and the Last Model Update Timestamp is updated accordingly, to reflect the change.

6. INTERFACES

The REST APIs used in this integration are described in the following chapters.

6.1. Change Signals Values Operation

6.1.1. Overview

The changesignalsvalues operation will be used to send the signal values from the source side to the destination side. This will be invoked in two use cases:

- Regular data exchange (see Data Exchange)
- Integrity update (see Integrity Update)

6.1.2. API Operation Specification

The operation specification is provided in the following table:

Table 6.1 – Change signals values – API Operation specification

Operation Specification	
Operation id	changesignalsvalues
URL	/changesignalsvalues
URL parameters	-
HTTP verb	PUT
Description	Method can be used to change values of the signals in ADMS. If successful, returns 200 OK and will include the header containing the timestamp of the last signal IDs list update, as described in the Physical Format chapter.

6.1.3. API Operation Usage

Table 6.2 – Change signals values – API operation usage

API usage details	
Host system	DERMS
Consuming system	ADMS
Retry number	Needs to be configurable in adapter configuration, 3 by default.
Retry condition	ADMS cannot connect to DERMS API.
Time between retries	Needs to be configurable in adapter configuration, 5 seconds by default.

6.1.4. Data Format

6.1.4.1. Header Format

Following attributes are used in the integration interface as part of the HTTP header:

- X-Source-ID – unique ID of the system which sent the HTTP request, supported values from the manual points integration standpoint are “ADMS” and “DERMS”. Source ID received in the HTTP request is preserved in HTTP response.
- X-Transaction-ID – needs to be populated in a form of Global Unique Identifier (GUID), unique ID of the transaction, HTTP request and HTTP response need to contain the same transaction ID.
- X-Timestamp – timestamp when request or response was created and sent.

6.1.4.2. Logical Format

The data specification of the payload in the logical format is provided in the following table:

Table 6.3 – Change signals values – logical data format

Device Status		Description
Property Name	Data Type	
id	String	Unique id of the signal. This maps to custom ID of the signal.
timestamp	DateTime	Timestamp when signal value change was done. This is in UTC.
value	Number	Value of the signal.

The response contains no payload. However, the response will contain the header with the timestamp of the last model update. This information will be in the header provided in the following table:

Table 6.4 – Change signals values response – logical data format

Device Status		Description
Property Name	Data Type	
lastmodelupdatetimestamp	DateTime	Timestamp of the last signals IDs list update. This is in UTC. This will only be a part of the header of the response with the response status 200 OK.

6.1.4.3. Physical Format

The JSON schema for change signals values payload is shown below:

```
{
  "type": "object",
  "properties": {
    "signalvalues": {
      "type": "array",
      "maxItems": 10000,
```

```
      "items": {
        "type": "object",
        "properties": {
          "id": {
            "type": "string"
          },
          "timestamp": {
            "type": "string",
            "format": "date-time"
          },
          "value": {
            "type": "number"
          }
        },
        "required": [
          "id",
          "timestamp",
          "value"
        ]
      }
    }
  },
  "required": [
    "signals"
  ]
}
```

Table 6.5 – Change Signal Values – Data specification in the physical format

Data Specification	
Name	Signals
Data content	Signal attributes as defined in Table 6.3.
Expected data size	No more than 10.000 elements in the array
Data format	JSON
Compress format	No
Encoding	UTF-8 https://en.wikipedia.org/wiki/Character_encoding
Date time format	UTC in the ISO-8601 format

6.2. Get Signal Ids Operation

6.2.1. Overview

The getsignalids operation will be used to send the ids of the required signals from the destination side to the source side. This will be invoked during integrity update (see Integrity Update). The source side will use those ids to filter the signals for which the values will be sent to the destination side.

6.2.2. API Operation Specification

The operation specification is provided in the following table:

Table 6.6 – Provide Signal IDs – API Operation specification

Operation Specification	
Operation id	getsignalids
URL	/getsignalids
URL parameters	start – index of first the element in the list to be returned (initially 0) end – index of the last element in the list to be returned (initially 10.000)
HTTP verb	GET
Description	Method can be used to set the filter for the signals on the source ADMS side. Returns paginated list of the signal ids as per the provided parameters. If there are less ids than parameters indicate, all the ids within that range will be returned.

6.2.3. API Operation Usage

Table 6.7 – Provide Signal IDs – API operation usage

API usage details	
Host system	ADMS
Consuming system	DERMS
Retry number	needs to be configurable in adapter configuration, 3 by default
Retry condition	DERMS cannot connect to ADMS API
Time between retries	needs to be configurable in adapter configuration, 5 seconds by default

6.2.4. Data Format

6.2.4.1. Header Format

Following attributes are used in the integration interface as part of the HTTP header:

- X-Source-ID – unique ID of the system which sent the HTTP request, supported values from the manual points integration standpoint are “ADMS” and “DERMS”. Source ID received in the HTTP request is preserved in HTTP response.
- X-Transaction-ID – needs to be populated in a form of Global Unique Identifier (GUID), unique ID of the transaction, HTTP request and HTTP response need to contain the same transaction ID.
- X-Timestamp – timestamp when request or response was created and sent.

6.2.4.2. Logical Format

The data specification in the logical format is provided in the following table:

Table 6.8 – Provide Signal IDs – logical data format

Device Status		Description
Property Name	**Data Type**	
id	String	Unique id of the signal. This maps to custom ID of the signal.
nextUrl	String	Provides the parameters for the next URL to be invoked. Source adapter forms the next URL by appending this value to the configured destination URL. If null, no action will be taken.

Table 6.9 – Provide Signal IDs response – logical data format

Device Status		Description
Property Name	**Data Type**	
LastModelUpdateTimestamp	DateTime	Timestamp of the last signals IDs list update. This is in UTC. This will only be a part of the header of the response with the response status 200 OK.

6.2.4.3. Physical Format

The JSON schema for get signal ids payload is shown below:

```
{
  "type": "object",
  "properties": {
    "signalids": {
      "type": "array",
      "maxItems": 10000,
      "items": {
        "type": "object",
        "properties": {
          "id": {
            "type": "string"
          }
            },
        "required": [
          "id"
        ]
      }
    },
    "nextUrl" : {
    "type": "string"
    }
  },
  "required": [
    "signalids"
  ]
}
```

Table 6.10 – Provide Signal IDs – Data specification in the physical format

Data Specification	
Name	Signals
Data content	Signal ids as defined in Table 6.8.
Expected data size	No more than 10.000 elements in the array.
Data format	JSON
Compress format	No
Encoding	UTF-8 https://en.wikipedia.org/wiki/Character_encoding
Date time format	UTC in the ISO-8601 format for the timestamp in the header.

7. ADDITIONAL CONSIDERATIONS

7.1. Quality and Testing

The testing of this integration can prove to be challenging, because two ADMS systems are needed to perform end to end testing properly. Each adapter can be tested separately.

Destination adapter can be tested with any REST test client.

For the source adapter, there should exist a test service that will mimic the destination side.

This test tool should:

- Read a subset of the signals from the source side and store that set in the internal cache.
- Expand and contract that subset in the internal cache, mimicking the model update.
- Write the received payload on the console or in some report for the examination of the test results.

The test cases hereafter will assume the two systems are set up, but the same can be achieved with test clients / test services.

7.1.1. Test GetSignalIds Interface

The purpose of this test is to demonstrate the blue sky scenario for the GetSignalIds interface.

7.1.1.1. Preconditions

DMSRealTime service must be started and in Hot State on the destination side.

7.1.1.2. Execution

1. Start the Integration service on the destination side.
2. MPIDestinationAdapter should start and switch to Hot state together with the Integration Service. This can also be verified in the adapter logs.
3. The SensitiveLog of the MPIDestinationAdapter it should contain the list of all relevant signals and the timestamp whe it was loaded.
4. Invoke GET operation on getsignalids resource, as defined in Get Signal Ids Operation. (example: *http://localhost:8082/MpiDestinationService/getsignalids?start=0&end=10000*).
5. The payload of the response should contain the list of ids of all manual discrete signals on the system. The payload should conform to the format described in Data Format. The response status should be 200 OK. The response should contain a header LastModelUpdateTimestamp with the timestamp of when the adapter was started (can be found in sensitive log). If there are more than 10000 signals on the system, the response should also contain the nextUrl field, looking like this:
   a. ?start=20000&end=30000
6. If there were less than 10000 signalids in the payload, the nextUrl field should be emtpy. Repeat the previous step using the parameters from the nextUrl field until the nextUrl field is empty.
7. The adapter log and the adapter sensitive log should follow all the actions.

7.1.2. Test ChangeSignalsValues Interface

The purpose of this test is to demonstrate the ChangeSignalValues interface.

7.1.2.1. Preconditions

DMSRealTime service must be started and in Hot State on the destination side.

7.1.2.2. Execution

1. Start the Integration service on the destination side.
2. MPIDestinationAdapter should start and switch to Hot state together with the Integration Service. This can also be verified in the adapter logs.
3. The SensitiveLog of the MPIDestinationAdapter it should contain the list of all relevant signals and the timestamp whe it was loaded.
4. Invoke PUT operation on changesignalsvalues resource, as defined in Change Signals Values Operation. (example: *http://localhost:8082/MpiDestinationService/changesignalsvalues*)
5. The payload of the request should contain the following list of signals:
    a. Valid signal:
        i. customId of a local discrete signal from the system.
        ii. value that is different from the previous value of the signal.
        iii. timestamp in the past, but newer than the previous timestamp.
    b. Non-existing signal:
        i. customId that does not exist on the destination system.
        ii. some valid value.
        iii. timestamp in the past, but newer than the previous timestamp.
    c. Outdated signal:
        i. customId of a local discrete signal from the system.
        ii. value that is different from the previous value of the signal.
        iii. timestamp in the past, and older than the previous timestamp.
    d. Signal from the future:
        i. customId of a local discrete signal from the system.
        ii. value that is different from the previous value of the signal.
        iii. timestamp in the future.
    e. Unchanged signal:
        i. customId of a local discrete signal from the system.
        ii. value that is the same as the previous value of the signal.
        iii. timestamp in the past, but newer than the previous timestamp.
6. The response status should be 200 OK. The response should contain a header LastModelUpdateTimestamp with the timestamp of when the adapter was started (can be found in sensitive log).
7. Only the first signal from step 5 should be changed accordingly on the destination side. For all other signals, the appropriate error message should be logged in the regular and sensitive log of the destination adapter.
8. The adapter log and the adapter sensitive log should follow all the actions.

7.1.3. Test Model Update on Destination Side

The purpose of the test is to demonstrate the response of the adapter to the model update on the system.

7.1.3.1. Preconditions

DMSRealTime service must be started and in Hot State on the destination side.

7.1.3.2. Execution

1. Start the Integration service on the destination side.
2. MPIDestinationAdapter should start and switch to Hot state together with the Integration Service. This can also be verified in the adapter logs.
3. The SensitiveLog of the MPIDestinationAdapter it should contain the list of all relevant signals and the timestamp whe it was loaded.
4. Invoke PUT operation on changesignalsvalues resource, or get operation on getsignalids resource, as described in the Test GetSignalIds Interface or Test ChangeSignalsValues Interface chapters.
5. The response status should be 200 OK. The response should contain a header LastModelUpdateTimestamp with the timestamp of when the adapter was started (can be found in sensitive log).
6. Perform a model update on the system which does not include adding or removing a local discrete signal.
7. The sensitive log of the adapter should not indicate that there was a change in the internal model.
8. Repeat step 4.
9. The result should be the same as in step 5.
10. Perform a model update on the system which includes adding or removing a local discrete signal.
11. The sensitive log of the adapter should indicate that there was a change in the internal model. The LastModelUpdateTimestamp in the sensitive log should reflect the time when model update was performed.
12. Repeat step 4.
13. The response status should be 200 OK. The response should contain a header LastModelUpdateTimestamp with the timestamp which is newer than the previous time. The timestamp should be the one of the model update. It can be deduced from the sensitive log of the adapter.

7.1.4. Test Start of MPISource Adapter Without Destination Side

The purpose of this test is to demonstrate the initialization of the MPISourceAdapter.

7.1.4.1. Preconditions

DMSRealTime service must be started and in Hot State on the source side. The destination adapter should not be available for this test.

7.1.4.2. Execution

1. Start the Integration service on the source side.

2. MPISourceAdapter should start and switch to Hot state together with the Integration Service. This can also be verified in the adapter logs.
3. The logs should indicate that the MPIAdapterManager is initializing and that the State Machine is in Inactive state after StartCollectingIdsEvent is fired.
4. The logs should then indicate failure to get signal ids repetitively.

7.1.5. Test Start of MPISource Adapter With Destination Side

The purpose of this test is to demonstrate the initialization of the MPISourceAdapter and its connection to the destination side.

7.1.5.1. Preconditions

DMSRealTime service must be started and in Hot State on the source side. The destination adapter should exist and be available for this test.

7.1.5.2. Execution

1. Start the Integration service on the source side. The destination adapter should not be running at this time.
2. MPISourceAdapter should start and switch to Hot state together with the Integration Service. This can also be verified in the adapter logs.
3. The logs should indicate that the MPIAdapterManager is initializing and that the State Machine is in Inactive state after StartCollectingIdsEvent is fired.
4. The logs should then indicate failure to get signal ids repetitively.
5. Start the destination adapter.
6. The logs on the source side should indicate that the integrity update took place and that the State Machine switched to the Active state after GetSignalsIdsOKEvent was fired.
7. On the destination side, the logs should indicate that the integrity update took place. All the values of the signals of interest on both sides should now be aligned, except the values on the destination side that were changed manually while the connection with source side was broken, as per requirements MPI-008 and MPI-010.

7.1.6. Test the Regular Signal Transfer

The purpose of this test is to demonstrate the blue sky scenario of the Manual Points Integration.

7.1.6.1. Preconditions

DMSRealTime service must be started and in Hot State on the source side. The destination adapter should exist and be available for this test.

7.1.6.2. Execution

1. Repeat all the steps from the Test Start of MPISource Adapter With Destination Side chapter.
2. Change the value of one of the affected signals on the source side.
3. The value of the corresponding signal on the destination side should change accordingly.
4. Change the value of one of the unaffected.

5. There should be no indication in the logs on the destination side that the change on the source side took place.

7.1.7. Reconnect the Source Side After the Failure of the Destination Side

The purpose of this test is to demonstrate how the MPISourceAdapter reconnects to the Destination side after the Destination side fails and restarts.

7.1.7.1. Preconditions

DMSRealTime service must be started and in Hot State on the source side. The destination adapter should exist and be available for this test.

7.1.7.2. Execution

1. Repeat all the steps from the Test Start of MPISource Adapter With Destination Side chapter.
2. Shut down the destination side.
3. There should be no changes in the logs or behavior of the source side.
4. Start the destination side.
5. Repeat steps 2 through 5 from the Test the Regular Signal Transfer chapter.
6. The results should be the same as in the Test the Regular Signal Transfer chapter.
7. Shut down the destination side.
8. Repeat step 2 from the Test the Regular Signal Transfer chapter.
9. In the logs there should be an indication that the destination side is unavailable, indicating also that the state machine has transited to Inactive state.
10. Start the destination side
11. The logs on the source side should indicate that the integrity update took place and that the State Machine switched to the Active state after GetSignalsIdsOKEvent was fired.
12. On the destination side, the logs should indicate that the integrity update took place. All the values of the signals of interest on both sides should now be aligned, except the values on the destination side that were changed manually while the connection with source side was broken, as per requirements MPI-008 and MPI-010.

7.1.8. Reaction of MPISourceAdapter to Model Update on the Destination Side

This test case demonstrates how the MPISourceAdapter reacts to model update on the source side that doesn't affect the signals of interest, and how it reacts to model update on the source side that affects the signals of interest.

7.1.8.1. Preconditions

DMSRealTime service must be started and in Hot State on the source side. The destination adapter should exist and be available for this test.

7.1.8.2. Execution

1. Repeat all the steps from the Test Start of MPISource Adapter With Destination Side chapter.

2. Perform a model update on the system which does not include adding or removing a local discrete signal.
3. There should be no evidence of model update in the sensitive log of the destination adapter.
4. There should be no changes in the logs or behavior of the source side.
5. Repeat step 2 from the Test the Regular Signal Transfer chapter.
6. The corresponding signal on the destination side should be updated accordingly. Nothing else should be indicated in the logs of the source or destination adapters.
7. Perform a model update on the system which includes adding or removing a local discrete signal.
8. The sensitive log of the adapter should indicate that there was a change in the internal model. The LastModelUpdateTimestamp in the sensitive log should reflect the time when model update was performed.
9. Repeat step 2 from the Test the Regular Signal Transfer chapter.
10. The logs on the source side should indicate that the integrity update took place and that the State Machine switched to the Active state after GetSignalsIdsOKEvent was fired.
11. On the destination side, the logs should indicate that the integrity update took place. All the values of the signals of interest on both sides should now be aligned.

7.2. Deployment and Configuration

7.2.1. Deployment

The adapters are installed as part of the Integration service. Each adapter is installed as part of its own add-on. These add-ons need to be selected on the corresponding system: MPISource add-on should be selected on the source system, whereas MPIDestination should be selected on the destination system.

This will deploy the binaries and configuration files to the server hosting the Integration service.

After the add-on installation, Config Tool needs to be run.

7.2.2. Configuration

MPIDestinationAdapter and MPISourceAdapter have two configuration files each:

- Registry configuration file
- WebServiceConfig configuration file

The parameters of interest are described in the tables below. The parameters that are not described are standard parameters used in all adapters, as described in *EcoStruxure GridOps Management Suite 3.10 Enterprise Integration Platform - Functional Specification* document [1].

Table 7.1 – MPI Source Registry Configuration

Parameter	Description
MaxNumberOfPages	Maximum number of pages to be retrieved from the destination side. Defaulted to 10000 if not present in the config file. Generally, not needed to be changed.

Table 7.2 – MPI Source WebService ChangeSignalsValues Configuration

Parameter	Description
ServiceUrl	URL of the ChangeSignalsValue endpoint.
NumOfRetries	Number of retries if the call fails. After this number is exceeded, communication exception is thrown which will initiate the transition of MPISourceAdapter state machine to the inactive state. Default value is 5. Has to be positive value.
RetryTimeout	Timeout between retries. Default value: 00:00:05.
AuthenticationType	Possible values for AuthenticationType are: None, OneWaySslAuthentication, TwoWaySslAuthentication, TransportWithMessageCredentialAuthentication, MessageSecurityCertificate.
ClientCertificateName	Name of the certificate to be used. Mandatory if AuthenticationType is anything other than None.

Table 7.3 – MPI Source WebService GetSignalIds Configuration

Parameter	Description
ServiceUrl	Base URL of the GetSignalIds endpoint. This value will be used to form the URL together with parameters returned in the NextURL field (see Logical Format).
NumOfRetries	Number of retries if the call fails. For this parameter, this value must be -1, as this call needs to be repeated until it succeeds.
RetryTimeout	Timeout between retries. Default value: 00:00:05.
AuthenticationType	Possible values for AuthenticationType are: None, OneWaySslAuthentication, TwoWaySslAuthentication, TransportWithMessageCredentialAuthentication, MessageSecurityCertificate.
ClientCertificateName	Name of the certificate to be used. Mandatory if AuthenticationType is anything other than None.

Table 7.4 – MPI Destination WebService Configuration

Parameter	Description
ServiceUrl	Base URL of the destination endpoints. This value will be used to form the URLs for the changesignalvalues and getsignalids endpoints.
AuthenticationType	Possible values for AuthenticationType are: None, OneWaySslAuthentication, TwoWaySslAuthentication,

Parameter	Description
	TransportWithMessageCredentialAuthentication, MessageSecurityCertificate.
ServiceCertificateName	Name of the certificate to be used. Mandatory if AuthenticationType is anything other than None.

7.3. Upgradability

The upgrade of the EcoStruxure ADMS version should have minimal effect on this integration. The interfaces used by these adapters are:

- GDA Queries
- NDSProxyContract
- ChangeSignalsValues (see Change Signals Values Operation)
- GetSignalIds (see Get Signal Ids Operation)

It is unlikely that these interfaces will change. Moreover, if they do not change, the adapters can be used to integrate two ADMS systems on different versions.

7.4. Extensibility

No extensibility is planned or intended for this integration.

7.5. Security

Security aspects of this integration are in accordance with Enterprise Integration Platform, as described in *EcoStruxure GridOps Management Suite 3.10 Enterprise Integration Platform - Functional Specification* document [1].

8. DEFINITIONS AND ABBREVIATIONS

Definition/Abbreviation	Description
ADMS	Advanced Distribution Management System
API	Application Programming Interface
DERMS	Distributed Energy Resources Management Systems
DMZ	Demilitarized Zone
GDA	Generic Data Access
GID	Global Identifier in DMS model
GUID	Global Unique Identifier
HTTP	Hyper-Text Transfer Protocol
ID	Identification
IU	Integrity Update
JSON	JavaScript Object Notation
MPI	Manual Points Integration
NDS	Network Dynamics Service
NMS	Network Model Service
REST	Representational State Transfer
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
UTF	Unicode Transformation Format