ETSI GS NFV-SOL 022 V5.3.1 (2025-09)

GROUP SPECIFICATION

Network Functions Virtualisation (NFV) Release 5; Protocols and Data Models; Policy descriptor

Disclaimer

The present document has been produced and approved by the Network Functions Virtualisation (NFV) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
DGS/NFV-SOL022ed531

Keywords
data models, MANO, NFV, policy management

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights 5
Foreword 5
Modal verbs terminology 5
1 Scope 6
2 References 6
2.1 Normative references 6
2.2 Informative references 6
3 Definition of terms, symbols and abbreviations 7
3.1 Terms 7
3.2 Symbols 7
3.3 Abbreviations 7
4 General Aspects 7
4.1 Overview 7
4.2 Common Data Types 8
4.2.1 Introduction 8
4.2.2 Simple data types and enumerations 8
4.2.2.1 Introduction 8
4.2.2.2 Simple data types 8
4.2.2.3 Enumerations 8
5 Analysis of Existing Data Models 9
5.1 Criteria for selecting the optimal policy data model 9
5.2 Comparison of the existing data models against the NFV-MANO policy model requirements 9
6 Data Model Specifications 10
6.1 Policy 10
6.1.1 Introduction 10
6.1.2 Properties 10
6.1.3 Definition 10
6.2 BasicInformation 11
6.2.1 Introduction 11
6.2.2 Properties 11
6.2.3 Definition 11
6.3 InstructionElementInformation 12
6.3.1 Introduction 12
6.3.2 Properties 12
6.3.3 Definition 12
6.4 TargetScopeInfomation 12
6.4.1 Introduction 12
6.4.2 Properties 12
6.4.3 Definition 13
6.5 ExpectedEffectivenessInformation 14
6.5.1 Introduction 14
6.5.2 Properties 14
6.5.3 Definition 14
6.6 State 14
6.6.1 Introduction 14
6.6.2 Properties 14
6.6.3 Definition 15
6.7 Task 15
6.7.1 Introduction 15
6.7.2 Properties 15
6.7.3 Definition 16
6.8 ContextMap 16
6.8.1 Introduction 16

6.8.2 Properties 16
6.8.3 Definition 17
6.9 Context 17
6.9.1 Introduction 17
6.9.2 Properties 17
6.9.3 Definition 17
6.10 Logic 18
6.10.1 Introduction 18
6.10.2 Properties 18
6.10.3 Definition 18

Annex A (informative): PlantUML source code 19

A.1 Information model definition for policy model 19
A.1.1 Relationship UML diagram for policy model (see figure 4.1-1) 19

Annex B (informative): Analysis on the data model solutions based on the NFV-MANO policy data model requirements 20

B.1 TOSCA 20
B.1.1 Overview 20
B.1.2 Comparison of NFV-MANO policy data model requirements with TOSCA 20

B.2 YANG 21
B.2.1 Overview 21
B.2.2 Comparison of the basic and optional capabilities of the policy data model with YANG 22

B.3 JSON 23
B.3.1 Overview 23
B.3.2 Comparison of the basic and optional capabilities of the policy data model with JSON 23

Annex C (informative): Examples of how to use Logic to describe ExpectedEffectivenessInformation 24

C.1 Overview 24

C.2 Using Logic to Describe ExpectedEffectivenessInformation 24

Annex D (informative): Examples 25

D.1 Policy descriptors design example by using JSON 25

Annex E (informative): Change history 27

History 28

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI IPR online database.

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (ISG) Network Functions Virtualisation (NFV).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document specifies the data model of policy descriptors to fulfill the functional requirements specified in ETSI GS NFV-IFA 010 [1] and ETSI GS NFV-IFA 048 [2] by providing the solutions based on the analysis results of various existing data model solutions (e.g. TOSCA, YANG).

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found in the ETSI docbox.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are necessary for the application of the present document.

[1] ETSI GS NFV-IFA 010: "Network Functions Virtualisation (NFV) Release 5; Management and Orchestration; Functional requirements specification".

[2] ETSI GS NFV-IFA 048: "Network Functions Virtualisation (NFV) Release 5; Management and Orchestration; Policy Information Model Specification".

[3] ETSI GS NFV-SOL 013: "Network Functions Virtualisation (NFV) Release 5; Protocols and Data Models; Specification of common aspects for RESTful NFV MANO APIs".

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents may be useful in implementing an ETSI deliverable or add to the reader's understanding, but are not required for conformance to the present document.

[i.1] ETSI GR NFV 003: "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV".

[i.2] ETSI GR NFV-IFA 042: "Network Functions Virtualisation (NFV) Release 4 Management and Orchestration; Report on policy information and data models for NFV-MANO".

[i.3] ETSI GS NFV-SOL 012: "Network Functions Virtualisation (NFV) Release 5; Protocols and Data Models; RESTful protocols specification for the Policy Management Interface".

[i.4] TOSCA: "OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA) TC".

[i.5] IETF RFC 7950: "The YANG 1.1 Data Modeling Language".

[i.6] JSON: "ECMA-404 The JSON Data Interchange Standard".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms given in ETSI GR NFV 003 [i.1] and the following apply:

expected effectiveness information: one or multiple metrics describing the expected system behavior or feature after the execution of the policy

instruction element information: states or tasks involved in the execution of the policy

target scope information: identify the set of managed objects or manage domains that would be affected by the execution of the policy

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the abbreviations given in ETSI GR NFV 003 [i.1] apply.

4 General Aspects

4.1 Overview

The present document defines the data model based on the information model and requirements defined in ETSI GS NFV-IFA 048 [2] and extended with the following principles:

- The data model shall support the representation of the policy descriptors which includes a generic information model and domain-specific information models for policy management. The generic information model is used to describe the common characteristics of policies across different domains and/or levels, while the domain-specific information models are used to describe information related to the environment in which policies are executed and/or interactions of policies within each respective domain.
- The generic information model includes at least one of the following: basic information, instruction element information, target scope information, and expected effectiveness information. Among the elements, expected effectiveness information is not mandatory.
- The data model shall support the representation of instruction element information. For example, instruction element information can include trigger events, evaluation conditions, and action decisions to describe the execution of a policy.
- The data model shall support the representation of target scope information. For example, target scope information can include the management domain of policy application and the managed objects of policy application to describe the scope of policy application.
- The data model shall support the representation of expected effectiveness information. For example, expected effectiveness information can include functional indicators, performance indicators, and usability indicators that describe the effects of policy implementation.
- The domain-specific information model includes at least one of the following: a domain-specific information model corresponding to instruction element information for each domain, a domain-specific information model associated with target scope information for each domain, and a domain-specific information model related to expected effectiveness information for each domain.

Figure 4.1-1 shows the overview of information elements for the policy model.

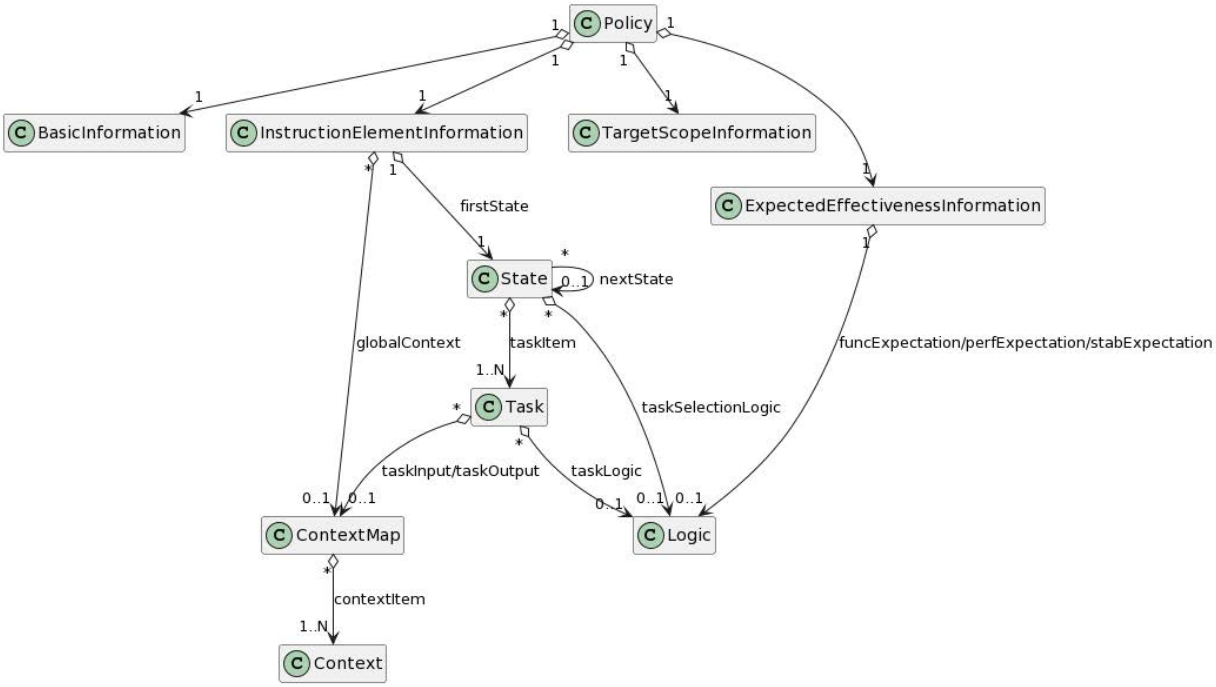

Figure 4.1-1: Overview of information elements for Policy Modelling

In clause 4, general aspects are specified that apply to multiple data model for policy descriptors. In the subsequent clauses, the different data models of policy descriptors will be evaluated, and a summarized analysis report as well as normative data model specifications for policy descriptors will be provided.

NOTE: In the present document, the term 'domain' refers to management functional domains such as NFVO, VNFM, VIM, and PIM.

4.2 Common Data Types

4.2.1 Introduction

Clause 4.3 specifies the common data types that are used for declaring the parameters and grammar elements throughout the present document.

4.2.2 Simple data types and enumerations

4.2.2.1 Introduction

This clause defines simple data types and enumerations that can be referenced from data structures defined in multiple interfaces.

4.2.2.2 Simple data types

The simple data type definitions in clause 7.2.2 of ETSI GS NFV-SOL 013 [3] shall apply.

4.2.2.3 Enumerations

The enumerations defined in clause 7.2.3 of ETSI GS NFV-SOL 013 [3] shall apply to be available for referencing from data type definitions in the present document.

5 Analysis of Existing Data Models

5.1 Criteria for selecting the optimal policy data model

Based on the recommendations for NFV-MANO policy models in ETSI GR NFV-IFA 042 [i.2], the policy information model defined in ETSI GS NFV-IFA 048 [2], and the policy management interface defined in ETSI GS NFV-SOL 012 [i.3], the data model shall have the following capabilities to support the policy descriptor:

- Semantic expression capability: The data model shall have rich semantic expression capability, accurately describing various attributes of the policy descriptor, including basic information, instruction element information, target scope information, and expected effectiveness information, while accurately describing the relationships between these different attributes.
- Data integration through predefined data types: The data model shall support the capability to reference predefined data types, enabling the integration of data and information from different domains and levels.
- Definition of policy types: The data model shall support the definition of policy types, ensuring accurate representation and management of different policy categories.
- Policy combination: The data model shall support the formation of complex policies through the combination of simple policies.
- Support inheritance of policy: The data model shall support defining extended policy types through inheritance of basic policy types, in order to formulate domain-specific policy types.
- Support policy management interface: The data model shall support the information elements that are transmitted via the policy management interface.

In the subsequent clauses, the support for the above capabilities will be analysed for different data models.

5.2 Comparison of the existing data models against the NFV-MANO policy model requirements

Based on previous analysis on policy descriptor solutions from Annex B, table 5.2-1 shows comparison of these solutions (TOSCA, YANG , JSON) against the NFV-MANO policy model requirements. Refer to "Capability" column from clause 5.1. The legend of "Support by policy descriptor solution" are following:

- "Yes": fully support the policy model requirements.
- "No": not support the policy model requirements.
- "Partial": partial support the policy model requirements.

Table 5.2-1: Comparison of the policy descriptor solutions against the NFV-MANO policy model requirements

Capability	Support by TOSCA	Support by YANG	Support by JSON
Semantic expression capability	Yes	Yes	Yes
Data integration through predefined data types	Yes	Yes	Yes
Definition of policy types	Yes	Partial	Yes
Policy combination	Partial	Yes	Yes
Support inheritance of policy	Yes	Partial	Partial
Support policy management interface	No	No	Yes

Based on this comparison, among policy descriptor solution candidates TOSCA, YANG and JSON, JSON is the most suitable one to meet the NFV-MANO policy model requirements. The JSON policy descriptor solution can be found in the attachment that accompanies the present document.

6 Data Model Specifications

6.1 Policy

6.1.1 Introduction

The policy information model, defined in ETSI GS NFV-IFA 048 [2] and extended in clause 4.1, is mapped to the JSON concepts. Policy occurrences are represented as JSON format, to be used by the NFVO for policy management.

6.1.2 Properties

The properties of the policy shall comply with the provisions set out in table 6.1.2-1.

Table 6.1.2-1: Properties

Name	Required	Type	Constraints	Description
basic_information	yes	BasicInformation		Used to describe basic information about a policy, including its name, type, source, version, etc.
instruction_element_information	yes	InstructionElementInformation		Used to describe information related to policy execution.
target_scope_information	yes	TargetScopeInformation		Used to describe information related to the scope of application of the policy.
expected_effectiveness_information	yes	ExpectedEffectivenessInformation		Used to describe information related to evaluating the execution effectiveness of the policy.

6.1.3 Definition

The syntax of the policy shall comply with the following definition:

```
{
  "Policy": {
    "basic_information": {
      "required": true,
      "type": "BasicInformation",
      "description": "Used to describe basic information about a policy, including its name,
type, source, version, etc."
    },
    "instruction_element_information": {
      "required": true,
      "type": "InstructionElementInformation",
      "description": "Used to describe information related to policy execution."
    },
    "target_scope_information": {
      "required": true,
      "type": "TargetScopeInformation",
      "description": "Used to describe information related to the scope of application of the
policy."
    },
    "expected_effectiveness_information": {
      "required": true,
      "type": "ExpectedEffectivenessInformation",
      "description": "Used to describe information related to evaluating the execution
effectiveness of the policy."
    }
  }
}
```

6.2 BasicInformation

6.2.1 Introduction

The BasicInformation data type describes the basic information about a policy, including its name, type, source, version, etc.

6.2.2 Properties

The properties of the BasicInformation data types shall comply with the provisions set out in table 6.2.2-1.

Table 6.2.2-1: Properties

Name	Required	Type	Constraints	Description
policy_function_name	yes	string		The function name of the policy, such as auto_scaling, self_healing, virtual resource optimization, etc.
policy_flavour	yes	Enum		The type of the policy, such as ECA, OODA, etc. The range of values: • ECA：Respectively representing Event, Condition, and Action. • OODA：Respectively representing Observation, Orient, Decision, and Action.
policy_source	yes	string	default: "NULL"	A string that identifies the source of the policy. For example, "Cloud" represents telco cloud professional operation and maintenance personnel, while "RAN" represents wireless professional operation and maintenance personnel.
policy_version	yes	string		A string that identifies the version information of the policy.

6.2.3 Definition

The syntax of the BasicInformation shall comply with the following definition:

```
{
  "BasicInformation": {
    "policy_function_name": {
      "required": true,
      "type": "string",
      "description": "The function name of the policy, such as auto_scaling, self_healing,
virtual resource optimization, etc."
    },
    "policy_flavour": {
      "required": true,
      "type": "Enum",
      "enum": ["ECA", "OODA"],
      "description": "The type of the policy, such as ECA, OODA, etc."
    },
    "policy_source": {
      "required": true,
      "type": "string",
      "default": "NULL",
      "description": "A string that identifies the source of the policy. For example,
\"Cloud\" represents telco cloud professional operation and maintenance personnel, while
\"RAN\" represents wireless professional operation and maintenance personnel."
    },
    "policy_version": {
      "required": true,
      "type": "string",
      "description": "A string that identifies the version information of the policy."
    }
  }
}
```

6.3 InstructionElementInformation

6.3.1 Introduction

The InstructionElementInformation data type describes the information related to policy execution.

6.3.2 Properties

The properties of the InstructionElementInformation data types shall comply with the provisions set out in table 6.3.2-1.

Table 6.3.2-1: Properties

Name	Required	Type	Constraints	Description
global_context	yes	ContextMap		Declarations of policy's system global context to be used during the policy execution. Context-awareness of the policy can relate to the actual execution of the policy. For instance, it can include system variables/service interface related to event/condition/action access interfaces/parameters/permissions that are used when the ECA policy is executed. See note.
first_state	yes	State		The first state for policy execution. For instance, in the case of an ECA policy, the first state is the "event" state.
NOTE: The specific modelling of the global context (i.e. the events triggering and actions as a result of the policy execution) is out of scope of the present document.				

6.3.3 Definition

The syntax of the InstructionElementInformation shall comply with the following definition:

```
{
  "InstructionElementInformation": {
    "global_context": {
      "required": true,
      "type": "ContextMap",
      "description": "Declarations of policy's system global context to be used during the policy
execution. Context-awareness of the policy can relate to the actual execution of the policy. For
instance, it can include system variables/service interface related to event/condition/action access
interfaces/parameters/permissions that are used when the ECA policy is executed."
    },
    "first_state": {
      "required": true,
      "type": "State",
      "description": "The first state for policy execution. For instance, in the case of an ECA
policy, the first state is the \"event\" state."
    }
  }
}
```

6.4 TargetScopeInfomation

6.4.1 Introduction

The TargetScopeInformation data type describes the information related to the target scope parameters required for policy execution.

6.4.2 Properties

The properties of the TargetScopeInfomation data types shall comply with the provisions set out in table 6.4.2-1.

Table 6.4.2-1: Properties

Name	Required	Type	Constraints	Description
target_domain_type	yes	Enum		Represents the domain of policy application management domain, and the names and meanings of all domains should align with the NFV-MANO framework. Possible values: • NFVO • VNFM • VIM • CISM • WIM • CCM
target_domain_id	yes	String		Identifier of the policy application management domain.
target_object_type	yes	Enum		Represents the type of the managed object in the policy application. The possible values of the managed object type depend on the selected policy application management domain. For example, when the policy application management domain is set to NFVO, possible values: • VNF • NS • VL
target_object_id	yes	String		Represents the identifier of the managed object in the policy application, which could include resources such as VNFs, NSs, or virtualized network infrastructure.

The target_domain_type, target_object_type, and target_object_id properties map to targetType, targetObjectType, and targetObjectId respectively in ETSI GS NFV-IFA 048 [2], clause 5.2.2.

6.4.3 Definition

The syntax of the TargetScopeInfomation shall comply with the following definition:

```
{
  "TargetScopeInformation": {
    "target_domain_type": {
      "required": true,
      "type": "Enum",
      "description": "Represents the domain of the policy application management, and the names and
meanings of all domains should align with the NFV-MANO framework. Possible values: NFVO, VNFM, VIM,
CISM, WIM, CCM."
    },
    "target_domain_id": {
      "required": true,
      "type": "String",
      "description": "Identifier of the policy application management domain."
    },
    "target_object_type": {
      "required": true,
      "type": "Enum",
      "description": "Represents the type of the managed object in the policy application. The
possible values of the managed object type depend on the selected policy application management
domain. For example, when the policy application management domain is set to NFVO, possible values:
VNF, NS, VL."
    },
    "target_object_id": {
      "required": true,
      "type": "String",
      "description": "Represents the identifier of the managed object in the policy application,
which could include resources such as VNFs, NSs, or virtualized network infrastructure."
    }
  }
}
```

6.5 ExpectedEffectivenessInformation

6.5.1 Introduction

The ExpectedEffectivenessInformation data type describes the information related to evaluating the effectiveness of policy execution.

6.5.2 Properties

The properties of the ExpectedEffectivenessInformation data types shall comply with the provisions set out in table 6.5.2-1.

Table 6.5.2-1: Properties

Name	Required	Type	Constraints	Description
func_expectation	yes	Logic		Expected effectiveness metrics for policy application.
perf_expectation	yes	Logic		Expected performance metrics for policy application.
stab_expectation	yes	Logic		Expected stability metrics for policy application

Annex C provides specific examples of how to use Logic to describe ExpectedEffectivenessInformation.

6.5.3 Definition

The syntax of the ExpectedEffectivenessInformation shall comply with the following definition:

```
{
  "ExpectedEffectivenessInformation": {
    "func_expectation": {
      "type": "Logic",
      "required": true,
      "description": "Expected effectiveness metrics for policy application."
    },
    "perf_expectation": {
      "type": "Logic",
      "required": true,
      "description": "Expected performance metrics for policy application."
    },
    "stab_expectation": {
      "type": "Logic",
      "required": true,
      "description": "Expected stability metrics for policy application."
    }
  }
}
```

6.6 State

6.6.1 Introduction

The State data type describes a phase in the policy execution process that includes operational tasks and state transition logic. This data type maps to the State information element defined in ETSI GS NFV IFA 048 [2].

6.6.2 Properties

The properties of the State data types shall comply with the provisions set out in table 6.6.2-1.

Table 6.6.2-1: Properties

Name	Required	Type	Constraints	Description
state_type	yes	Enum		Represents the type of policy state, which should be selected based on the policy_flavour property in the BasicInformation data type.For example, when policy_flavour is ECA, possible value: • Event • Condition • Action
task_item	yes	Task		Represents the information of the tasks included in the current state. Each state shall contain at least one task.
task_selection_logic	yes	Logic		When a state contains more than one task, it represents the logic for selecting a task based on the current context.
next_state	yes	String		Identifier of the next state.

6.6.3 Definition

The syntax of the State shall comply with the following definition:

```
{
  "State": {
    "state_type": {
      "required": true,
      "type": "Enum",
      "description": "Represents the type of policy state, which should be selected based on the
policy_flavour property in the BasicInformation data type. For example, when policy_flavour is ECA,
possible value: Event, Condition, Action."
    },
    "task_item": {
      "required": true,
      "type": "Task",
      "description": "Represents the information of the tasks included in the current state. Each
state shall contain at least one task."
    },
    "task_selection_logic": {
      "required": true,
      "type": "Logic",
      "description": "When a state contains more than one task, it represents the logic for
selecting a task based on the current context."
    },
    "next_state": {
      "required": true,
      "type": "String",
      "description": "Identifier of the next state."
    }
  }
}
```

6.7 Task

6.7.1 Introduction

The Task data type describes details of a task item included in a state. This data type maps to the Task information element defined in ETSI GS NFV IFA 048 [2].

6.7.2 Properties

The properties of the Task data types shall comply with the provisions set out in table 6.7.2-1.

Table 6.7.2-1: Properties

Name	Required	Type	Constraints	Description
task_type	yes	Enum		The type of task. Each state contains a task which correspond to event triggering, condition evaluation, and action decision & execution, respectively. Possible value: • EVENT-TASK • CONDITION-TASK • ACTION-TASK
task_input	yes	ContextMap		The input for the task execution.
task_logic	yes	Logic		The logic for the task.
task_output	yes	ContextMap		The output of the task execution.

6.7.3 Definition

The syntax of the Task shall comply with the following definition:

```
{
  "Task": {
"task_type": {
      "required": true,
      "type": "Enum",
      "description": "The type of task. Each state contains a task which correspond to event
triggering, condition evaluation, and action decision & execution, respectively. Possible value:
EVENT-TASK, CONDITION-TASK, ACTION-TASK"
    },
"task_input": {
      "required": true,
      "type": "ContextMap",
      "description": "The input for the task execution."
    },
    "task_logic": {
      "required": true,
      "type": "Logic",
      "description": "The logic for the task."
    },
"task_output": {
      "required": true,
      "type": "ContextMap",
      "description": "The output of the task execution."
    }
  }
}
```

6.8 ContextMap

6.8.1 Introduction

The ContextMap data type describes a list of Contexts which are grouped into a ContextMap. This data type maps to the ContextMap information element defined in ETSI GS NFV IFA 048 [2].

6.8.2 Properties

The properties of the ContextMap data types shall comply with the provisions set out in table 6.8.2-1.

Table 6.8.2-1: Properties

Name	Required	Type	Constraints	Description
map_name	yes	String		Name or identification of the ContextMap.
context_item	yes	array of Context		List of context that are grouped in the ContextMap.

6.8.3 Definition

The syntax of the ContextMap shall comply with the following definition:

```
{
  "ContextMap": {
    "map_name": {
      "required": true,
      "type": "String",
      "description": "Name or identification of the ContextMap."
    },
    "context_item": {
      "required": true,
      "type": "array",
      "description": "List of context that are grouped in the ContextMap.",
      "items": {
        "type": "Context"
      }
    }
  }
}
```

6.9 Context

6.9.1 Introduction

The Context data type maps to the ContextItem information element defined in ETSI GS NFV IFA 048 [2].

6.9.2 Properties

The properties of the Context data types shall comply with the provisions set out in table 6.3.2-1.

Table 6.9.2-1: Properties

Name	Required	Type	Constraints	Description
context_name	yes	String		Name or identification of the Context.
context_description	yes	String		Description about the Context, e.g. to provide information about the purpose or use of the context.
context_access	yes	String		It provides information about how to access or send the data or related interface operations contained in the Context.
context_parameter	yes	String		Parameters and values for the Context.

6.9.3 Definition

The syntax of the Context shall comply with the following definition:

```
{
  "Context": {
    "context_name": {
      "required": true,
      "type": "String",
      "description": "Name or identification of the Context."
    },
    "context_description": {
      "required": true,
      "type": "String",
      "description": "Description about the Context, e.g. to provide information about the purpose
or use of the context."
    },
    "context_access": {
      "required": true,
      "type": "String",
      "description": "It provides information about how to access or send the data or related
interface operations contained in the Context."
    },
    "context_parameter": {
      "required": true,
```

```
      "type": "String",
      "description": "Parameters and values for the Context."
    }
  }
}
```

6.10 Logic

6.10.1 Introduction

The Logic data type maps to the Logic information element defined in ETSI GS NFV IFA 048 [2].

6.10.2 Properties

The properties of the Logic data types shall comply with the provisions set out in table 6.10.2-1.

Table 6.10.2-1: Properties

Name	Required	Type	Constraints	Description
logic_name	yes	String		Name or identification of the Logic.
logic_dsl	yes	Enum		Defines the Domain Specific Language (DSL) of logic that is provided. Possible value: • Bash • Python
logic_code_type	yes	Enum		Defines the type of logic_code, indicating the source of the logic implementation. Possible values: • logicDsl • File
logic_code	yes	String		Includes the implementation of logic (written in a DSL as specified by logicDsl) or a reference to a file or artifact providing the implementation of the logic.

6.10.3 Definition

The syntax of the Logic shall comply with the following definition:

```
"Logic": {
  "logic_name": {
    "required": true,
    "type": "String",
    "description": "Name or identification of the Logic."
  },
  "logic_dsl": {
    "required": true,
    "type": "Enum",
    "description": "Defines the Domain Specific Language (DSL) of logic that is provided. Possible
values: Bash, Python."
  },
  "logic_code_type": {
    "required": true,
    "type": "Enum",
    "description": "Defines the type of logic_code, indicating the source of the logic
implementation. Possible values: logicDsl, File."
    },
  "logic_code": {
    "required": true,
    "type": "String",
    "description": "Includes the implementation of logic (written in a DSL as specified by logicDsl)
or a reference to a file or artifact providing the implementation of the logic."
  }
}
```

Annex A (informative): PlantUML source code

A.1 Information model definition for policy model

A.1.1 Relationship UML diagram for policy model (see figure 4.1-1)

```
@startuml
hide empty members
hide fields

InstructionElementInformation "1" o--> "1" State:firstState
InstructionElementInformation "*" o--> "0..1" ContextMap:globalContext
ExpectedEffectivenessInformation "1" o--> "0..1"
Logic:funcExpectation/perfExpectation/stabExpectation
State "*" o--> "1..N" Task:taskItem
State "*" o--> "0..1" Logic:taskSelectionLogic
State "*" --> "0..1" State:nextState
Task "*" o--> "0..1" ContextMap:taskInput/taskOutput
Task "*" o--> "0..1" Logic:taskLogic
ContextMap "*" o--> "1..N" Context:contextItem

Policy "1" o--> "1" BasicInformation
Policy "1" o--> "1" InstructionElementInformation
Policy "1" o--> "1" TargetScopeInformation
Policy "1" o--> "1" ExpectedEffectivenessInformation
@enduml
```

Annex B (informative): Analysis on the data model solutions based on the NFV-MANO policy data model requirements

B.1 TOSCA

B.1.1 Overview

This clause analyses the comparison of the NFV-MANO policy data model requirements defined in clause 5.1 with TOSCA.

TOSCA [i.4] is an OASIS open standard that defines the interoperable description of services and applications hosted on the cloud and elsewhere; including their components, relationships, dependencies, requirements, and capabilities, thereby enabling portability and automated management across cloud providers regardless of underlying platform or infrastructure.

B.1.2 Comparison of NFV-MANO policy data model requirements with TOSCA

This clause shows comparison of the basic and optional capabilities of the policy data model described in clause 5.1 with TOSCA.The legend of "Support by TOSCA " are following:

- "Yes": fully support the basic and optional capabilities of the policy data model.
- "No": not support the basic and optional capabilities of the policy data model.
- "Partial": partial support the basic and optional capabilities of the policy data model.

Table B.1.2-1: Comparison of NFV-MANO policy data model requirements with TOSCA

Capability	Requirement	Support by TOSCA	Related capability of TOSCA
Semantic expression capability	The data model have rich semantic expression capability, accurately describing various attributes of the policy descriptor	Yes	By defining nodes, relationships, attributes, interfaces, operations, and policies, TOSCA provides a detailed description of the various components of the application and their relationships, thus providing rich semantic expression capabilities to accurately describe the various attributes and expected effects of the policy descriptors
Data integration through predefined data types	The data model support the capability to reference predefined data types	Yes	Attributes, inputs, and outputs in the TOSCA template can refer to predefined data types, which allows these definitions to be shared and reused among different components.
Definition of policy types	The data model support the definition of policy types, ensuring accurate representation and management of different policy categories.	Yes	TOSCA supports the definition of policy types, ensuring the accurate representation and management of various policy categories.
Policy combination	The data model support the formation of complex policies through the combination of simple policies.	Partial	TOSCA does not directly support combining simple policies to form complex ones, but this can be indirectly achieved through custom node types, attributes, relationships, interfaces, and orchestration logic.
Support inheritance of policy	The data model support defining extended policy types through inheritance of basic policy types, in order to formulate domain-specific policy types.	Yes	TOSCA supports extension and customization on the basis of existing types through inheritance mechanism.
Support policy management interface	The data model support the information elements that are transmitted via the policy management interface.	No	The data model defined by TOSCA does not directly support the information elements transmitted via the policy management interface.

B.2 YANG

B.2.1 Overview

This clause analyses the comparison of the NFV-MANO policy data model requirements defined in clause 5.1 with YANG.

Yet Another Next Generation (YANG) [i.5] is a data modeling language used to model configuration data, state data, Remote Procedure Calls, and notifications for network management protocols.

B.2.2 Comparison of the basic and optional capabilities of the policy data model with YANG

This clause show comparison of the basic and optional capabilities of the policy data model described in clause 5.1 with YANG.The legend of "Support by YANG" are following:

- "Yes": fully support the basic and optional capabilities of the policy data model.
- "No": not support the basic and optional capabilities of the policy data model.
- "Partial": partial support the basic and optional capabilities of the policy data model.

Table B.2.2-1: Comparison of NFV-MANO policy data model requirements with YANG

Capability	Requirement	Support by YANG	Related capability of YANG
Semantic expression capability	The data model have rich semantic expression capability, accurately describing various attributes of the policy descriptor	Yes	YANG supports rich semantic expression capabilities through standardized syntax definitions, tree-structured data models, support for multiple data types, modular design, clear definition statements, and features such as scalability and flexibility.
Data integration through predefined data types	The data model support the capability to reference predefined data types	Yes	Supports creating custom data types through typedef statements and referencing predefined data types
Definition of policy types	The data model support the definition of policy types, ensuring accurate representation and management of different policy categories.	Partial	YANG does not have a built-in "policy type" but supports the definition of various data structures and nodes, which can be used to represent different policy types.
Policy combination	The data model support the formation of complex policies through the combination of simple policies.	Yes	YANG supports the definition of modules and submodules, which can be reused and combined to build more complex policies and data structures.
Support inheritance of policy	The data model support defining extended policy types through inheritance of basic policy types, in order to formulate domain-specific policy types.	Partial	YANG does not directly support inheritance, but can achieve similar functionality through the use of 'grouping' and 'extension'.
Support policy management interface	The data model support the information elements that are transmitted via the policy management interface.	No	The data model defined by YANG does not directly support the information elements transmitted via the policy management interface.

B.3 JSON

B.3.1 Overview

This clause analyses the comparison of the NFV-MANO policy data model requirements defined in clause 5.1 with JSON.

JavaScript Object Notation (JSON) [i.6] is a lightweight data-interchange format that is easy for humans to read and write, and for machines to parse and generate.

B.3.2 Comparison of the basic and optional capabilities of the policy data model with JSON

This clause show comparison of the basic and optional capabilities of the policy data model described in clause 5.1 with JSON.The legend of "Support by JSON" are following:

- "Yes": fully support the basic and optional capabilities of the policy data model.
- "No": not support the basic and optional capabilities of the policy data model.
- "Partial": partial support the basic and optional capabilities of the policy data model.

Table B.3.2-1: Comparison of NFV-MANO policy data model requirements with JSON

Capability	Requirement	Support by JSON	Related capability of JSON
Semantic expression capability	The data model have rich semantic expression capability, accurately describing various attributes of the policy descriptor.	Yes	JSON's rich data types and extensible structure are sufficient to accurately describe various attributes of policy descriptors.
Data integration through predefined data types	The data model support the capability to reference predefined data types.	Yes	JSON allows referencing predefined values or objects through key names, which can represent specific data types.
Definition of policy types	The data model support the definition of policy types, ensuring accurate representation and management of different policy categories.	Yes	JSON's key-value pair structure conveniently allows for the definition and distinction of various policy types.
Policy combination	The data model support the formation of complex policies through the combination of simple policies.	Yes	JSON's nested object and array data structures allow for the combination of multiple simple policies into hierarchical and structured complex policy representations.
Support inheritance of policy	The data model support defining extended policy types through inheritance of basic policy types, in order to formulate domain-specific policy types.	Partial	JSON does not directly support inheritance, but JSON's object structure allows nesting and referencing, it can simulate inheritance relationships and extend attributes..
Support policy management interface	The data model support the information elements that are transmitted via the policy management interface.	Yes	JSON is a lightweight data-interchange format, particularly suitable for the information elements transmitted via the policy management interface.

Annex C (informative): Examples of how to use Logic to describe ExpectedEffectivenessInformation

C.1 Overview

This clause provides an example of how to describe ExpectedEffectivenessInformation in a policy. The example is based on a VNF management policy and defines expectations regarding whether the VNF starts successfully, whether its throughput meets the required level, and whether it can run without failure for a defined period.

C.2 Using Logic to Describe ExpectedEffectivenessInformation

The following JSON example illustrates how to represent ExpectedEffectivenessInformation in a policy. It includes three properties:

- func_expectation: specifies the expected success rate of the VNF startup.
- perf_expectation: defines the expected performance level, such as throughput.
- stab_expectation: indicates the expected stability, including continuous uptime over a period.

Each expectation is described using a simple logic expression, as shown below:

```
{
  "ExpectedEffectivenessInformation": {
    "func_expectation": {
      "logic_name": "VNF Functionality Expectation",
      "logic_dsl": "Python",
      "logic_code_type": "logicDsl",
      "logic_code": "expected_vnf_startup_success_rate = 1.0  # Ensure VNF starts successfully"
    },
    "perf_expectation": {
      "logic_name": "VNF Performance Expectation",
      "logic_dsl": "Python",
      "logic_code_type": "logicDsl",
      "logic_code": "expected_throughput = 1000000000  # Expected throughput of 1Gbps"
    },
    "stab_expectation": {
      "logic_name": "VNF Stability Expectation",
      "logic_dsl": "Python",
      "logic_code_type": "logicDsl",
      "logic_code": "expected_vnf_uptime = 259200  # Ensure VNF runs without failure for 72 hours"
    }
  }
}
```

Annex D (informative): Examples

D.1 Policy descriptors design example by using JSON

In Annex A of ETSI GS NFV-IFA048 [2], an example of a policy model is provided. Based on this example, and the definition of using JSON as a data model given in the present document, a concrete example of the usage of the JSON data model is provided.

```
{
  "Policy": {
    "basic_information": {
      "policy_function_name": "auto_scale",
      "policy_flavour": "ECA",
      "policy_source": "Cloud",
      "policy_version": "1.0.0"
    },
    "instruction_element_information": {
      "global_context": {
        "map_name": "vnf utilization",
        "context_item": {
          "context_name": "vnf utilization",
          "context_description": "The utilization of VNF",
          "context_access": "vnfIndicator.utilization",
          "context_parameter": null
        }
      },
      "first_state": {
        "EventState": {
          "state_type": "Event",
          "task_item": {
            "Task": {
              "task_type": "EVENT-TASK",
              "task_input": {
                "ContextMap": {
                  "map_name": "eventContextMap",
                  "context_item": {
                    "context_name": "eventContext",
                    "context_description": "The context of event",
                    "context_access": "vnfIndicator.utilization",
                    "context_parameter": null
                  }
                }
              },
              "task_output": {
                "ContextMap": {
                  "map_name": null,
                  "context_item": {
                    "context_name": "event task output context",
                    "context_description": "Context of the output of event task",
                    "context_access": null,
                    "context_parameter": "NextState"
                  }
                }
              }
            }
          },
          "next_state": "ConditionState"
        }
      },
      "ConditionState": {
        "state_type": "Condition",
        "task_item": {
          "Task": {
            "task_type": "CONDITION-TASK",
            "task_input": {
              "ContextMap": {
                "map_name": "conditionContextMap",
                "context_item": {
                  "context_name": "conditionContext",
                  "context_description": "The context of condition",
                  "context_access": "vnfIndicator.utilization",
```

```
                  "context_parameter": null
                }
              }
            },
            "task_logic": {
              "Logic": {
                "logic_name": "ConditionLogic",
                "logic_code_type": "logicDsl",
                "logic_dsl": "Python",
                "logic_code": "(utilization_vnf_indicator>=60) and (call_proc_scale_level<3)"
              }
            },
            "task_output": {
              "ContextMap": {
                "map_name": null,
                "context_item": {
                  "context_name": "condition task output context",
                  "context_description": "Context of the output of Condition task",
                  "context_access": null,
                  "context_parameter": "NextState"
                }
              }
            }
          }
        },
        "next_state": "ActionState"
      },
      "ActionState": {
        "state_type": "Action",
        "task_item": {
          "Task": {
            "task_type": "ACTION-TASK",
            "task_input": {
              "ContextMap": {
                "map_name": "actionContextMap",
                "context_item": {
                  "context_name": "actionContext",
                  "context_description": "The context of action",
                  "context_access": "vnfIndicator.utilization",
                  "context_parameter": null
                }
              }
            },
            "task_output": {
              "ContextMap": {
                "map_name": "ScaleContextMap",
                "context_item": {
                  "context_name": "scale",
                  "context_description": "The scale of context",
                  "context_access": "Vnflcm.scale",
                  "context_parameter": "type:scale_out;aspect:call_proc;number_of_steps:1"
                }
              }
            }
          }
        },
        "next_state": null
      }
    },
    "target_scope_information": {
      "target_domain_type": "NFVO",
      "target_domain_id": "CMCC-Beijing-Fengtai-DC1",
      "target_object_type": "VNF",
      "target_object_id": null
    },
    "expected_effectiveness_information": {
      "perf_expectation": {
        "Logic": {
          "logic_name": "UtilityExpectationLogic",
          "logic_dsl": "Python",
          "logic_code_type": "logicDsl",
          "logic_code": "vnfIndicator.utilization<=60%"
        }
      }
    }
  }
}
```

Annex E (informative): Change history

Date	Version	Information about changes
January 2024	V0.0.1	Early draft including the following contributions until SOL#270 meeting: NFVSOL(23)000374, NFVSOL(23)000375r2, NFVSOL(23)000377r1, NFVSOL(23)000379, NFVSOL(23)000380r2
August 2024	V0.0.2	Early draft including the following contributions until SOL#289 meeting: NFVSOL(24)000078, NFVSOL(24)000079r2, NFVSOL(24)000080, NFVSOL(24)000166r1, NFVSOL(24)000177r2, NFVSOL(24)000178r2
February 2025	V0.0.3	Early draft including the following contributions until SOL#316 meeting: NFVSOL(24)000266r1, NFVSOL(24)000267r1, NFVSOL(24)000268r1, NFVSOL(24)000269r2, NFVSOL(24)000467r2, NFVSOL(24)000468r3, NFVSOL(24)000469r3
April 2025	V0.0.4	Early draft including the following contributions until SOL#319 meeting: NFVSOL(25)000050r1, NFVSOL(25)000051r1, NFVSOL(25)000052r1, NFVSOL(25)000057r1, NFVSOL(25)000058, NFVSOL(25)000059r1
June 2025	V0.0.5	Implemented approved contributions: NFVSOL(25)000102r1, NFVSOL(25)000103r1, NFVSOL(25)000104r1, NFVSOL(25)000105, NFVSOL(25)000217

History

Version	Date	Status
V5.3.1	September 2025	Publication