

Draft Recommendation for  
Space Data System Standards

|  |
| --- |
| Mission Operations—Mission Planning and Scheduling Services |

AUTHORITY

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | Issue: | , |  |
|  | Date: |  |  |
|  | Location: | Not Applicable |  |
|  |  |  |  |

**(WHEN THIS RECOMMENDED STANDARD IS FINALIZED, IT WILL CONTAIN THE FOLLOWING STATEMENT OF AUTHORITY:)**

This document has been approved for publication by the Management Council of the Consultative Committee for Space Data Systems (CCSDS) and represents the consensus technical agreement of the participating CCSDS Member Agencies. The procedure for review and authorization of CCSDS documents is detailed in *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-4), and the record of Agency participation in the authorization of this document can be obtained from the CCSDS Secretariat at the email address below.

This document is published and maintained by:

CCSDS Secretariat

National Aeronautics and Space Administration

Washington, DC, USA

Email: secretariat@mailman.ccsds.org

STATEMENT OF INTENT

**(WHEN THIS RECOMMENDED STANDARD IS FINALIZED, IT WILL CONTAIN THE FOLLOWING STATEMENT OF INTENT:)**

The Consultative Committee for Space Data Systems (CCSDS) is an organization officially established by the management of its members. The Committee meets periodically to address data systems problems that are common to all participants, and to formulate sound technical solutions to these problems. Inasmuch as participation in the CCSDS is completely voluntary, the results of Committee actions are termed **Recommended Standards** and are not considered binding on any Agency.

This **Recommended Standard** is issued by, and represents the consensus of, the CCSDS members. Endorsement of this **Recommendation** is entirely voluntary. Endorsement, however, indicates the following understandings:

o Whenever a member establishes a CCSDS-related **standard**, this **standard** will be in accord with the relevant **Recommended Standard**. Establishing such a **standard** does not preclude other provisions which a member may develop.

o Whenever a member establishes a CCSDS-related **standard**, that member will provide other CCSDS members with the following information:

-- The **standard** itself.

-- The anticipated date of initial operational capability.

-- The anticipated duration of operational service.

o Specific service arrangements shall be made via memoranda of agreement. Neither this **Recommended Standard** nor any ensuing **standard** is a substitute for a memorandum of agreement.

No later than five years from its date of issuance, this **Recommended Standard** will be reviewed by the CCSDS to determine whether it should: (1) remain in effect without change; (2) be changed to reflect the impact of new technologies, new requirements, or new directions; or (3) be retired or canceled.

In those instances when a new version of a **Recommended Standard** is issued, existing CCSDS-related member standards and implementations are not negated or deemed to be non-CCSDS compatible. It is the responsibility of each member to determine when such standards or implementations are to be modified. Each member is, however, strongly encouraged to direct planning for its new standards and implementations towards the later version of the Recommended Standard.

FOREWORD

Through the process of normal evolution, it is expected that expansion, deletion, or modification of this document may occur. This Recommended Standard is therefore subject to CCSDS document management and change control procedures, which are defined in the *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-4). Current versions of CCSDS documents are maintained at the CCSDS Web site:

http://www.ccsds.org/

Questions relating to the contents or status of this document should be sent to the CCSDS Secretariat at the email address indicated on page i.

At time of publication, the active Member and Observer Agencies of the CCSDS were:

Member Agencies

* Agenzia Spaziale Italiana (ASI)/Italy.
* Canadian Space Agency (CSA)/Canada.
* Centre National d’Etudes Spatiales (CNES)/France.
* China National Space Administration (CNSA)/People’s Republic of China.
* Deutsches Zentrum für Luft- und Raumfahrt (DLR)/Germany.
* European Space Agency (ESA)/Europe.
* Federal Space Agency (FSA)/Russian Federation.
* Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
* Japan Aerospace Exploration Agency (JAXA)/Japan.
* National Aeronautics and Space Administration (NASA)/USA.
* UK Space Agency/United Kingdom.

Observer Agencies

* Austrian Space Agency (ASA)/Austria.
* Belgian Science Policy Office (BELSPO)/Belgium.
* Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
* China Satellite Launch and Tracking Control General, Beijing Institute of Tracking and Telecommunications Technology (CLTC/BITTT)/China.
* Chinese Academy of Sciences (CAS)/China.
* China Academy of Space Technology (CAST)/China.
* Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
* Danish National Space Center (DNSC)/Denmark.
* Departamento de Ciência e Tecnologia Aeroespacial (DCTA)/Brazil.
* Electronics and Telecommunications Research Institute (ETRI)/Korea.
* Egyptian Space Agency (EgSA)/Egypt.
* European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
* European Telecommunications Satellite Organization (EUTELSAT)/Europe.
* Geo-Informatics and Space Technology Development Agency (GISTDA)/Thailand.
* Hellenic National Space Committee (HNSC)/Greece.
* Hellenic Space Agency (HSA)/Greece.
* Indian Space Research Organization (ISRO)/India.
* Institute of Space Research (IKI)/Russian Federation.
* Korea Aerospace Research Institute (KARI)/Korea.
* Ministry of Communications (MOC)/Israel.
* Mohammed Bin Rashid Space Centre (MBRSC)/United Arab Emirates.
* National Institute of Information and Communications Technology (NICT)/Japan.
* National Oceanic and Atmospheric Administration (NOAA)/USA.
* National Space Agency of the Republic of Kazakhstan (NSARK)/Kazakhstan.
* National Space Organization (NSPO)/Chinese Taipei.
* Naval Center for Space Technology (NCST)/USA.
* Netherlands Space Office (NSO)/The Netherlands.
* Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
* Scientific and Technological Research Council of Turkey (TUBITAK)/Turkey.
* South African National Space Agency (SANSA)/Republic of South Africa.
* Space and Upper Atmosphere Research Commission (SUPARCO)/Pakistan.
* Swedish Space Corporation (SSC)/Sweden.
* Swiss Space Office (SSO)/Switzerland.
* United States Geological Survey (USGS)/USA.

PREFACE

This document is a draft CCSDS Recommended Standard. Its ‘Red Book’ status indicates that the CCSDS believes the document to be technically mature and has released it for formal review by appropriate technical organizations. As such, its technical contents are not stable, and several iterations of it may occur in response to comments received during the review process.

Implementers are cautioned **not** to fabricate any final equipment in accordance with this document’s technical content.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

DOCUMENT CONTROL

|  |  |  |  |
| --- | --- | --- | --- |
| **Document** | **Title** | **Date** | **Status** |
|  | , , |  | Current draft |
|  |  |  |  |
|  |  |  |  |

CONTENTS

Section Page

[1 Introduction 1-1](#_Toc191212990)

[1.1 General 1-1](#_Toc191212991)

[1.2 Purpose and Scope 1-2](#_Toc191212992)

[1.3 Applicability 1-2](#_Toc191212993)

[1.4 Rationale 1-3](#_Toc191212994)

[1.5 Document Structure 1-3](#_Toc191212995)

[1.6 Definitions 1-4](#_Toc191212996)

[1.7 Nomenclature 1-8](#_Toc191212997)

[1.8 Conventions 1-9](#_Toc191212998)

[1.9 References 1-9](#_Toc191212999)

[2 Overview 2-1](#_Toc191213000)

[2.1 General 2-1](#_Toc191213001)

[2.2 Mission Planning Concept 2-1](#_Toc191213002)

[2.3 Relationship To Mission Operations Services 2-4](#_Toc191213003)

[2.4 MPS Information Model Overview 2-6](#_Toc191213004)

[2.5 MPS Services Overview 2-13](#_Toc191213005)

[2.6 Optional Elements of the Recommended Standard 2-15](#_Toc191213006)

[3 MPS Service Specifications 3-16](#_Toc191213007)

[3.1 Overview 3-16](#_Toc191213008)

[3.2 Optional Elements 3-16](#_Toc191213009)

[3.3 Conventions 3-18](#_Toc191213010)

[3.4 External Definitions 3-20](#_Toc191213011)

[3.5 Service: Planning Request Service 3-20](#_Toc191213012)

[3.6 Service: Plan Distribution Service 3-30](#_Toc191213013)

[3.7 Service: Plan Execution Control Service 3-38](#_Toc191213014)

[3.8 Service: Plan Information Management Service 3-58](#_Toc191213015)

[3.9 Service: Plan Edit Service 3-66](#_Toc191213016)

[4 MPS Information Model 4-1](#_Toc191213017)

[4.1 Overview 4-1](#_Toc191213018)

[4.2 Optional Elements 4-1](#_Toc191213019)

[4.3 Conventions 4-3](#_Toc191213021)

[4.4 External Definitions 4-5](#_Toc191213022)

[4.5 MPS Service Objects 4-7](#_Toc191213023)

[4.6 MPS Data Types 4-47](#_Toc191213024)

[5 Error Codes 5-1](#_Toc191213025)

[5.1 Overview 5-1](#_Toc191213026)

[5.2 MPS-Specific MO Errors 5-1](#_Toc191213027)

[5.3 Data Type: SecondaryErrorCodeEnum 5-3](#_Toc191213028)

[6 Service Specification XML 6-1](#_Toc191213029)

[6.1 Overview 6-1](#_Toc191213030)

[6.2 XML Schema Definition (XSD) for MO Services 6-1](#_Toc191213031)

[6.3 MAL XML 6-2](#_Toc191213032)

[6.4 MPS XML 6-2](#_Toc191213033)

[7 XML File Formats 7-1](#_Toc191213034)

[7.1 Overview 7-1](#_Toc191213035)

[7.2 XML Schema Namespace 7-1](#_Toc191213036)

[7.3 XML Schema Encoding 7-1](#_Toc191213037)

[7.4 XML Schema Structure 7-3](#_Toc191213038)

[7.5 XML File Structure 7-5](#_Toc191213039)

[7.6 Planning Request XML File Formats 7-7](#_Toc191213040)

[7.7 Plan XML File Format 7-9](#_Toc191213041)

[7.8 MPS XML File Format Schema Location 7-10](#_Toc191213042)

[ANNEX A Protocol Implementation Conformance Statement (PICS) Proforma   (Normative) A-1](#_Toc186645501)

[ANNEX B Security, SANA, and Patent Considerations   (Informative) B-1](#_Toc186645502)

[ANNEX C Definition of Acronyms   (Informative) C-1](#_Toc186645503)

[ANNEX D Informative References   (Informative) D-1](#_Toc186645504)

Figure

[2-1 Functions involved in Mission Planning 2-1](#_Toc191144796)

[2-2 Entities and Functions Involved in Mission Planning 2-3](#_Toc191144797)

[2-3 MO MPS Services Generic Protocol Stack 2-5](#_Toc191144798)

[2-4 MPS Data Items 2-7](#_Toc191144799)

[2-5 MPS Data Items and their Constituent MO Objects 2-9](#_Toc191144800)

[7-1 XML Schema Hierarchy 7-4](#_Toc191144801)

[7-2 MPS XML File Structure 7-5](#_Toc191144802)

Table

[3-1 Example Service Table 3-18](#_Toc191144803)

[3-2 Example Operation Template 3-19](#_Toc191144804)

[3-3 Example Error References Table 3-19](#_Toc191144805)

[4-1 Mandatory and Optional Elements of the Information Model 4-2](#_Toc191144806)

[4-2 Example Data Structure Table 4-3](#_Toc191144807)

[4-3 Example Enumeration Table 4-4](#_Toc191144808)

[4-4 MO Object Numbers 4-4](#_Toc191144809)

[4-5 MO Object Numbers 4-70](#_Toc191144810)

[4-6 MO Object Numbers 4-75](#_Toc191144811)

[7-1 Mapping of MAL Attribute Types to XSD Types 7-2](#_Toc191144812)

# Introduction

## General

Mission planning is an activity that often requires interaction between multiple entities. This may be to support distributed planning, where the responsibility for different aspects of mission operations planning is spread over multiple entities, including the space segment. It may also be to facilitate collaboration between missions, or to allow the planning of payloads by multiple end-users or the planning of multiple payloads from different agencies hosted on the same spacecraft. Other missions, such as observatories, may make payloads available to a wider user community. Some planning responsibility may be delegated to the spacecraft itself and the corresponding capabilities hosted on board. Historically, these interoperable interfaces have typically been defined on a per-mission or per-agency basis.

This Recommended Standard has the objective of specifying generic, interoperable mission planning and scheduling interfaces, for all typical space mission use cases, including the ones identified above. These use cases are elaborated in the associated Informational Report (Green Book) (reference [D2]) *Mission Planning and Scheduling*. This Recommended Standard focuses on the Mission Planning and Scheduling (MPS) Services identified for supporting interoperability and defines an information model that defines the data structures required by the operations of these services.

Mission planning and scheduling are integral parts of Mission Operations (MO) and closely related to the other aspects of the overall monitoring and control of space missions. This close relationship is recognized in the context of the CCSDS Mission Operations and Information Management (MOIMS) Area by the fact that the MPS services have been identified and included from the start among the envisaged MO services described in reference [D1], *Mission Operations Services Concept*. This Recommended Standard defines the MO MPS services in conformance with the CCSDS Mission Operations service framework described therein.

The MPS services are a set of services that support:

1. interaction with a mission planning system and its users at the level of planning requests;
2. distribution of the plans generated;
3. control of the **execution** of those plans.

It is expected, but not required, that these are used in conjunction with other mission operations services, such as Monitoring & Control (reference [D4]) and Automation, as identified in reference [D1].

The MPS services are defined in terms of the Message Abstraction Layer (MAL) (see reference [2], *Mission Operations Message Abstraction Layer*), that is the core of the MO service framework.

## Purpose and Scope

This Recommended Standard defines, in an abstract manner, the MPS services in terms of:

1. service specifications that define the operations necessary to provide the services, together with their parameter data and required behaviour;
2. an information model that describes the structure of MPS data, including planning requests, plans, and supporting information objects that are referenced by the MPS services;
3. file-based message formats for the exchange of planning requests and plans, for use in mission deployments that opt not to make use of service-based interfaces.

Some parts of this Recommended Standard are optional; not all of its aspects need to be applied in the context of a specific MPS system in order to support a conformant interface. Each service is optional and may include optional capability sets within it. Some aspects of the MPS information model are optional. A summary of the optional elements of the standard is provided in 2.6.

This Recommended Standard does not specify:

1. individual implementations or products;
2. the internal implementation of planning systems;
3. the methods or technologies required for communications;
4. how required MPS service configuration data is made available to deployed MPS functions;
5. the expression language used for representation of conditions and calculations embedded within MPS data.

## Applicability

This specification is applicable to any mission operations component that provides mission planning functionality or executes mission plans and exposes mission planning interfaces. This includes interfaces between:

* Mission users and the Mission Planning system;
* Hierarchical or distributed components of a Mission Planning system;
* Mission Planning and Plan Execution components;
* Plan Execution and Mission Control.

Further detail is given in the associated Informational Report (Green Book) (reference [D2]).

This Recommended Standard is intended to apply to interfaces wherever they may occur in a space system:

* between ground-based components across a terrestrial link;
* between ground-based and space-based components across a space link;
* between space-based components across a space link;
* and potentially between on-board components across an on-board interface.

## Rationale

This standard was developed to fulfil the need for mission planning interoperability among agencies and space system users at the level of exchanged planning requests and plans. Additionally, it provides service specifications that facilitate interoperability between missions and systems within an agency and that promote the development of re-usable infrastructure for space systems.

## Document Structure

This Recommended Standard is organized in the following sections:

* **Section 1—Introduction**: provides purpose and scope, applicability, and rationale of this Recommended Standard and lists the definitions, conventions, and references used throughout the document;
* **Section 2—Overview**: describes the mission planning concept and how this relates to MO services, as well as giving a high-level overview of both the MPS information model and the set of MPS services specified.
* **Section 3—MPS Service Specifications**: provides the formal specification of the MPS services.
* **Section 4—MPS Information Model**: provides the formal specification of MPS service objects, including MO objects, and other data structures that are contained in or referenced by MPS service messages.
* **Section 5—Error Codes**: provides the formal specification of MPS error codes.
* **Section 6—Service Specification XML**: specifies the internet location of the formal service specification eXtensible Markup Language (XML).
* **Section 7—XML File Formats**: specifies the internet location of the formal specification of XML file formats for the exchange of planning requests and plans.

Sections 1–7 contain the normative specification of the Recommended Standard; section 2 is purely informative; section 1 (this section) contains normative definitions and normative references, as well as some informative material related to the document itself.

## Definitions

For the purposes of this document, the following definitions apply.

NOTES

1. Abbreviations are to be found in annex C.
2. The terms plan and planning are used throughout this specification, but these terms are intended to also encompass schedule and scheduling, respectively.
3. The prefix ‘planning’ is used to disambiguate terms used in this specification from other more general uses of a term. This applies to planning activities, constraints, events, requests, and resources.

|  |
| --- |
|  |
| action: A single executable task of an MO M&C service provider. A telecommand is an example of an action. |
| activity definition: The definition element of a planning activity. It forms part of the planning configuration data. |
| activity details: The information required to create an activity instance from an activity definition. It may be contained within a planning request to request inclusion of a planning activity, or within an activity definition (to specify child activities). |
| activity instance: The instance element of a planning activity. Activity instances are contained within plans. |
| area: A group of related MO services with an associated identifier and number. This Recommended Standard forms part of the MPS area, with the area number 5. |
| argument: A run-time parameter provided to various control items on invocation. For example, arguments apply to planning requests, planning activities and planning events. |
| constraint: (See planning constraint.) |
| custom function: An ancillary MPS service object that allows access to built-in Boolean functions of a planning system, for example in the context of planning constraints (specifically a function constraint). The custom function must be pre-defined to be referenceable and the MPS custom function definition holds the declaration of an available function. |
| definition: The statically declared information associated with an information object. This may, for example, include a description, set of defined arguments or any other information that applies to all occurrences of the information object. There may be multiple definitions [versions] over the mission lifetime associated with the same identity [definitionID]. |
| details: A data structure used to specify the information needed to create an instance from a definition for an information object that has multiple occurrences. |
| direction: An MPS data type that is used to represent a pointing direction or attitude of a spacecraft, payload instrument, or other object. |
| domain: A namespace that partitions separately addressable entities (e.g., planning activities, planning events, or planning resources) in the mission. The mission is decomposed into a hierarchy of domains within which entity identifiers are unique. |
| effect: A type of planning constraint that is used in the context of modelling planning resources. It specifies the impact that executing a planning activity will have on a planning resource. |
| event definition: The definition element of a planning event. It forms part of the planning configuration data. |
| event instance: The instance element of a planning event. Event instances are contained within plans. |
| expression: A calculation to be performed at run time that supplies a value of a defined data type. Expressions are specified as text strings, together with the identification of the expression language used. No standard expression language is specified in this document. |
| full plan: A plan that contains the full contents of a plan. Used to distinguish from a patch plan. |
| identity: A unique identity associated with an MO object, which comprises:   * The domain of the object; * The area of the object; * The **type** of the object; * A key (identifier) for the object, unique within domain, area and type; * The version of the object [optional]. |
| information object: The set of information about a real-world entity that is exchanged across an interface. This may include static definitions, dynamic status and metadata.  NOTE – Mission planning information objects include: planning requests, plans, planning activities, planning events and planning resources. |
| instance: A dynamically created object representing each new occurrence of an information object. This includes a unique instanceID of the occurrence and any unchanging data associated with it as a set of static fields. It also includes the current status of the object as a set of dynamic fields. An instance has a reference to its definition. |
| key: Part of the identity of an MO object, the key is a unique identifier for the object within the scope of the domain, area and object type. |
| **partial plan:** A view into a selected subset of a **plan**. It contains only the **activity instances** of a **plan** that match a given set of criteria. | |
| patch plan: A plan that only contains the delta (changes) from a precursor plan. A patch plan must be merged with its precursor plan to generate the target plan. |
| plan: The output of the planning process. It contains a set of selected planning activities associated with time, position, or planning event. A plan may contain additional related information.  NOTE – In the context of the mission planning and scheduling standardization activity, there is no distinction between the terms ‘plan’ and ‘schedule’ and only the term plan is used. |
| plan execution: The process of executing plans on board or on the ground. |
| plan revision: The set of changes that describe the difference between two plans. These changes can include creation, removal, or modification of activity and event instances. | |
| planning: The process of creating one or more plans (output) from planning requests (input). |
| planning activity: A meaningful unit of what can be planned. The granularity of a planning activity depends on the use case; it may be hierarchical. In other words, planning activities are the building blocks for planning. |
| planning configuration data: The set of configuration data required by MPS service providers and consumers. It includes activity definitions, event definitions, resource definitions, request definitions, and MPS system configuration parameters. |
| planning constraint: Something that limits or restricts the planning of planning activities. Different types of constraint exist, including: temporal constraints, sequential constraints between planning activities and/or planning events, resource constraints, and geometric constraints (position and pointing). |
| planning event: The meeting of a condition that is external to planning. **Planning events** may be linked to related **planning activities** by means of relative timing or by passing **arguments** of the **planning event** on to **planning activities**. |
| planning request: An input to the planning process, which requests one or more planning activities. |
| planning resource: An abstraction of a real-world resource, physical or virtual, that is represented as a quantity. A planning resource may constrain or trigger the execution of planned activities, which may in turn have an effect on the value of the resource. |
| planning user: Any entity that is responsible for submitting planning requests to a planning function and potentially receiving feedback on the status of planning requests and generated plans. For example, this could be an external Principal Investigator (PI) or a mission operations system or role. |
| position: An MPS data type that is used to represent the physical location of a spacecraft, or other object. |
| potential event: A type of planning event that is not predictable, but may still have a defined response within a plan. |
| precursor plan: A plan from which the current plan represents an evolution through replanning or an iterative planning cycle. The current plan contains the specification of the changes from the precursor plan as a set of plan revisions. |
| predicted event: A type of planning event that is expected to occur at a particular time or position that can be predicted as an input to planning and contained within a plan. Orbital events are an example of predicted events. |
| procedure: In the context of MPS, an executable process that is invoked to fulfil the execution of a planned activity. Automated operations procedures, on-board control procedures, and procedures supported by an MO Automation service provider (see reference [D1]) are examples of a procedure. |
| repetition: A data structure used in the context of a planning request to request the repeated execution of planning activities. Various sub-types of repetition are defined to support the specification of repeat cycles by different criteria, such as time, position, or pointing. |
| request definition: The optional definition element of a planning request that contains the specification of a re-usable planning request template. It forms part of the planning configuration data. |
| request instance: The instance element of a planning request. This may change over time if the request is updated by the user, each comprising a separate version of the request. |
| resource definition: The **definition** element of a **planning resource** that specifies its static fields. It may omit dynamic fields of the resource (its value) and forms part of the **planning configuration data**. |
| resource profile: Provision for the evolution of the value of a planning resource over time. |
| slider: A relative position with respect to an MPS object, such as a planning activity, where 0 represents the start and 1 the end of the activity. The slider is a real number that can represent a specific point between these two extremes. |
| sub-plan: A unique subset of activity instances, identified by a single sub-plan identifier.  NOTE – This can be used to allow execution control over multiple distinct **sub-plans** within **plans**, as needed to support separate payloads, or individual spacecraft within a constellation. | |
| target plan: A full plan that is the result of applying a patch plan to its precursor plan. |
| trigger: A construct that allows specification of the specific condition that marks the start or end of something. It is used in the context of both planning activities and plans to specify when an activity should start or end. Triggers may be defined in terms of time, position, pointing, or a planning event. |
| update: A data structure used to report the changing value of dynamic fields and arguments of an MO object (including its status) at a specific point in time. |
| version: Part of the identity of an MO object, typically of a definition object, that represents a defined set of values for its static fields. When a definition is updated, the version is incremented, but other elements of the object’s identity, including its key remain unchanged. |

## Nomenclature

### Normative Text

The following conventions apply for the normative specifications in this Recommended Standard:

1. the words ‘shall’ and ‘must’ imply a binding and verifiable specification;
2. the word ‘should’ implies an optional, but desirable, specification;
3. the word ‘may’ implies an optional specification;
4. the words ‘is’, ‘are’, and ‘will’ imply statements of fact.

NOTE – These conventions do not imply constraints on diction in text that is clearly informative in nature.

### Informative Text

In the normative sections of this document, informative text is set off from the normative specifications either in notes or under one of the following subsection headings:

* Overview;
* Background;
* Rationale;
* Discussion.

## Conventions

In this standard,

1. the notation <area>::<type name> is used to reference data types that are defined by other MO specifications. For example, ‘MAL::Boolean’ refers to the Boolean type defined within the MAL area.

## References

The following publications contain provisions which, through reference in this text, constitute provisions of this document. At the time of publication, the editions indicated were valid. All publications are subject to revision, and users of this Recommended Standard are encouraged to investigate the possibility of applying the most recent editions of the publications indicated below. The CCSDS secretariat maintains a register of currently valid CCSDS publications.

[1] *Mission Operations Reference Model*. Issue 1. Recommendation for Space Data System Practices (Magenta Book), CCSDS 520.1-M-1. Washington, D.C.: CCSDS, July 2010.

[2] *Mission Operations Message Abstraction Layer*. Issue 3. Recommendation for Space Data System Standards (Blue Book), CCSDS 521.0-B-3. Washington, D.C.: CCSDS, March 2024.

[3] *OMG® Unified Modeling Language (OMG® UML)*. Version 2.5.1. formal/2017-12-05. Needham, Massachusetts: Object Management Group, December 2017.

[4] T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. STD 66. Reston, Virginia: ISOC, January 2005.

[5] H. Thompson and C. Lilley. *XML Media Types*. RFC 7303. Reston, Virginia: ISOC, July 2014.

[6] *Space Assigned Numbers Authority (SANA)—Role, Responsibilities, Policies, and Procedures*. Issue 3. CCSDS Record (Yellow Book), CCSDS 313.0-Y-3. Washington D.C.: CCSDS, October 2020.

[7] *Reference Architecture for Space Data Systems*. Issue 2. Recommendation for Space Data System Practices (Magenta Book), CCSDS 311.0-M-2. Washington, D.C.: CCSDS, December 2024.

[8] *IEEE Standard for Floating-Point Arithmetic*. 3rd ed. IEEE Std 754-2019. New York: IEEE, 2019.

[9] *XML Path Language (XPath)*. Version 3.1. <https://www.w3.org/TR/xpath-31/>. W3C, 21 March 2017.

[10] *Pointing Request Message*. Issue 1, Technical Corrigendum 2. Recommendation for Space Data System Standards (Blue Book), CCSDS 509.0-B-1. Washington, D.C.: CCSDS, October 2023.

NOTE – Informative references are listed in annex D.

# 

# Overview

## General

This section introduces the concepts behind the MPS services. It has the following main sections:

* Mission Planning Concept;
* Relationship to Mission Operations Services;
* MPS Information Model Overview;
* MPS Services Overview;
* Optional Elements of the Recommended Standard.

## Mission Planning Concept

A diagram of a plan

Description automatically generated

Figure 2‑1 : Interfaces involved in Mission Planning

Mission Planning encompasses application level functions of a space mission system that may be distributed across multiple organizations and physical nodes, both in the space and ground segments. Standardization in this area concerns only the interaction between functions at the application level, and not the mission planning functions themselves.

The scope of standardization includes both the format/model of data exchanged, as well as the semantics of the interactions for their exchange, captured by the associated service level interfaces.

A generalized view of the functions involved in mission planning and their interactions with other functions is given in figure 2‑12‑1‑2‑1, which is based on the Reference Architecture for Space Data Systems (RASDS) notation (see reference [7]).

The entities shown in blue are in the functional area of mission planning. The entities shown in different colours belong to other functional areas of mission operations, such as monitoring and control, navigation, and the ground station and communication network.

The following mission planning functions are identified:

* **Planning User**: a generic function that is responsible for submitting requests to the planning function. It may also receive feedback on the status of planning requests and the generated plans. It is not a planning function itself, but is a user of planning data and services. A deployment in an actual space mission may contain multiple types of planning user function, some of which correspond to other mission operations functions within the space mission system.
* **Planning**: the function responsible for performing mission planning. Internally it may be hierarchically organized and/or distributed. Planning requests are received from multiple Planning users (or other mission planning functions) and feedback on their status is provided. The output of the planning function is plans, which may be retrieved by planning users and submitted to plan execution functions or follow-up planning functions or entitities (via planning requests). Planning may also control the execution of plans via the plan execution functions. Planning is itself a user of the navigation function and may receive predicted planning events, as a future standard Navigation Event Message (NEM) (reference [D5]) or in a custom format, that are related to orbital information, attitude, or slew times; and negotiates the scheduling of ground station support via Cross Support Services (CSS) services (reference [D6]).
* **Plan Execution**: the function responsible for executing a plan (or part of it). There may be multiple plan execution functions distributed between space and ground segments. The plan execution function may represent actual or simulated execution of plans to support the planning function on-board or on the ground. It is not a planning function itself, but it does support a common model of the plan in its interface with planning. It receives or retrieves distributed plans, allows external control of the plan execution process, and provides feedback on execution status of the plan. Plan execution may use underlying mission control services to effect the execution of planned activities. Mission controllers may interact with plan execution functions to control the plan execution process and to edit plans. External functions may also edit plans, for example to update planning events or resources.

It should be noted that in an actual deployment, there may be multiple copies of all the functions identified in figure 2‑1‑2‑1.

Figure 2‑1‑2‑1 shows MPS services as blue lines. Interactions supported by other CCSDS Recommended Standards are shown in other colours. The circle at one end indicates which function is the service provider: Planning is the service provider for Planning Request and Plan Distribution services. Plan execution is the service provider for Plan Execution Control and Plan Edit services. Both functions can provide the Plan Information Management service. These services are introduced in section 2.5 below and formally defined in section 3.

The identified functions may be distributed over a number of distinct entities (organizations and systems) within a given space mission system. There is not a fixed set of such entities, but typical examples include:

* User Community / PIs;
* Science/Payload Operations Centre;
* Payload Processing Centre;
* Mission Operations Centre;
* Flight Dynamics / Navigation;
* Ground Tracking Network;
* Uncrewed Spacecraft;
* Surface Lander / Rover;
* Human Space Vehicle.

A diagram of a mission control

Description automatically generated

Figure 2‑2 : Entities and Functions Involved in Mission Planning

As an example, figure 2‑22‑22‑22‑22‑2‑2‑2 illustrates potential deployment of each of the functions identified in figure 2‑1‑2‑1 to the entities listed above. The circles indicate where each of the functions are typically deployed in existing systems, or where they could potentially be deployed in the future. The arrows indicate the interactions in a typical current deployment, but the potential distribution of functions indicated by the circles shows that all the functional interfaces shown in figure 2‑1‑2‑1 can be exposed to the boundaries between entities. It is where the interactions between the functions are exposed across one or more boundaries between entities that there is a need for standardization within CCSDS as a potentially interoperable interface between agencies.

The interactions within the scope of mission planning and scheduling standardization can be grouped into five services:

* **Planning Request**: submission of planning requests to a planning function, associated responses and their subsequent management and status feedback;
* **Plan Distribution**: distribution and access to plans generated by the planning function;
* **Plan Execution Control**: submission of plans to a plan execution function, management of the execution process, and status feedback;
* **Plan Information Management**: access to planning data definitions;
* **Plan Edit**: direct manipulation of plans outside the planning process, either to update planning events and resources with the latest information or for emergency intervention.

A common MPS information model applies to the planning requests and plans transferred or Referenced by these services and also to the common configuration data required by service providers and consumers to interpret the planning requests and plans. This information model is introduced in section 2.4 below and the associated data structures are formally defined in section 4.

For those organizations that do not wish to standardize the service level interaction, but only to standardize the data format used for the exchange of planning requests and plans, standard XML-based file formats are defined in section 7.

## Relationship To Mission Operations Services

The MO Services Concept provides a standard framework for the specification of end-to-end services between mission operations applications (reference [D1]). MO services are defined in terms of a MAL (reference [2]), which provides a means of specifying data and service interfaces in an implementation, encoding, and communication agnostic manner.

Figure 2‑3‑2‑3 is based on the CCSDS Application and Support Layer Architecture (reference [D9]) and shows the generic protocol stack for the MPS services.

A screenshot of a computer game

Description automatically generated

Figure 2‑3 : MO MPS Services Generic Protocol Stack

The MO MAL defines:

* A set of MAL data types that can be used to represent the individual data fields of message structures;
* A set of MAL Interaction patterns that correspond to the message exchange behaviour of individual service operations.

The abstract specification of the service interfaces and data can be mapped to a concrete implementation through:

1. a technology binding that defines how the abstract messages (composed of a sequence of MAL Attributes) are encoded in a concrete format (e.g., binary, XML, or ASCII);
2. a technology binding that defines how the resulting messages are carried over a concrete message transport protocol by mapping the standard MAL interaction patterns to that protocol;
3. a language binding that transforms the abstract service interface into a concrete API for a given programming language (e.g., Java, C++ or Python).

Figure 2‑3‑2‑3 illustrates a generic deployment of MPS services using the MO service framework with service consumer and provider functions hosted on different deployment nodes. MPS specific functions and protocol layers are shown in blue; elements of the ‘vanilla’ MO framework in yellow; and underlying communications infrastructure layers in tan (light orange). The application level MPS service interaction is shown by the direct interface between service provider and consumer functions, carrying MPS service messages defined in terms of data structures specified in the MPS information model.

Adopting a single MAL technology binding in any specific deployment ensures on-the-wire interoperability. Transfer protocol equates to the messaging or file transfer service used over the underlying Transport and physical Data Link Layers. The diagram illustrates how different language bindings can be used by provider and consumer for the service API, as this does not affect the wire level protocol.

It is noted that while the MAL may be implemented as a specific software layer for reasons of maintainability and reusability, it is not a requirement to do so. The MAL may be used as an abstract specification that enables transformation of the service specification by the applied technology bindings into a concrete implementation of that service with no distinct MAL layer. The MO MPS, MO MAL, and MAL technology binding layers in the diagram are effectively combined into a single software component. This is an important distinction for deployment contexts where the implementation is required to be both compact and efficient, such as on board a spacecraft.

An MO object is an entity defined within the information model of an MO compliant service specification that has a unique identity enabling it to be referenced by other MO objects and in the body of MO service messages. The **identity** of an MO object is defined by its type and unique **key**, scoped by its **area**, **domain** and optionally by a **version**. The specification of a service-specific MO object class includes a custom set of references to other MO objects that capture the relationships between those objects. In the context of the MPS services, MO objects are defined to represent planning requests, plans and the planning activities, planning events, and resources that they reference.

An MO application-level service specification comprises a set of operations that the service consumer may invoke on the service provider. Each operation is mapped to a standard interaction pattern defined by the MAL and provides the service-specific body of the constituent messages.

## MPS Information Model Overview

### General

The information exchanged across the interfaces supported by MPS services is complex. Requests for activities to be planned require the specification of the activities to be performed and any constraints on their execution. Plans express not only the planned activities they contain, but also what triggers their execution relative to time, position or planning events.

The MPS information model describes the information objects that are transferred or operated on by service operations and the relationships between them. It addresses both the content of the messages that constitute service operations and also the configuration data that both consumer and provider must have access to to interpret those messages.

This document contains the specification of data structures derived from the MPS information model that are used in the body of service messages or referenced by them. These data structure definitions are a normative part of the Recommended Standard and are expressed in the tabular format described in 4.3.1.

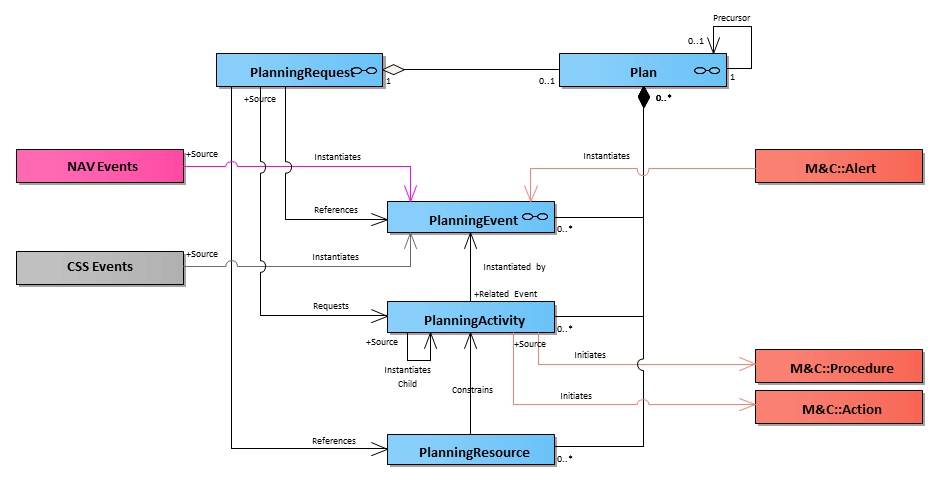


Figure 2‑4 : MPS Service Objects

A high-level overview of the MPS information model is given in figure 2‑4‑2‑4. This shows the principal MPS service objects and their interrelationships using standard UML notation (see reference [3]).

The rectangles in the diagram correspond to standard service objects. The lines between them define the relationships between those service objects. Service objects are color-coded by functional area:

◼ Mission Planning service objects defined in this document are shown in blue.

◼ Mission Control service objects (red) correspond to CCSDS MO Monitoring & Control and the proposed MO Automation standards.

◼ Navigation service object (magenta) correspond to the CCSDS Navigation Event Message Recommended Standard.

◼ Cross Support Services service objects (grey) correspond to CCSDS Cross Support Service Management Recommended Standards.

Relationships shown in black are within the scope of mission planning standardization, others are color-coded by their respective area.

The following principal MPS service objects are shown in the diagram:

* **Planning Requests;**
* **Plans;**
* **Planning Activities;**
* **Planning Events;**
* **Planning Resources.**

These are introduced in the subsections below. Each of these service objects comprises a set of **MO objects** with its own object **identity**.

The status of planning requests, plans, planning activities, and planning events can evolve over time and is reported through the defined MPS services. State models are associated with each of these service objects in the full information model, but only the minimum set of states has been defined to work with the defined services: missions may effectively extend these with additional states using additional information fields.

**Planning requests** and **plans** are both container objects, whose content relates to a set of planning service objects: **planning events**, **planning activities**, and **planning resources**. For each of these three types (or classes) of planning data, there is a defined set of items that can be referenced or instantiated within planning requests and plans. Together these definitions comprise the **planning configuration data**.

**Planning constraints** are not self-standing service objects, but can be attached to planning requests and planning activities. They are defined in 4.6.6. Subsection 4.6 also includes the definition of other MPS supporting data types, including:

* MPS **Position** and **Direction** data types;
* **Expressions;**
* **Arguments;**
* **Triggers;**
* **Repetitions.**

Some aspects of the MPS information model are optional. These aspects are not required to be supported by a compliant MO MPS service provider, although this may limit the set of service capabilities and associated operations that can be supported. Optional aspects of the model include:

* Planning Resources;
* Functions;
* Planning Constraints, other than representation as a text expression using any defined expression syntax supported by the service provider;
* Position and Direction data types;
* Repetitions (the representation of repetitive occurrences of **planning activities**).

### MO Objects

Figure 2‑5 shows how each of the MPS Service Objects comprises multiple MO objects following a two- or three-element MO object pattern. MO objects are shown with a bold purple border.

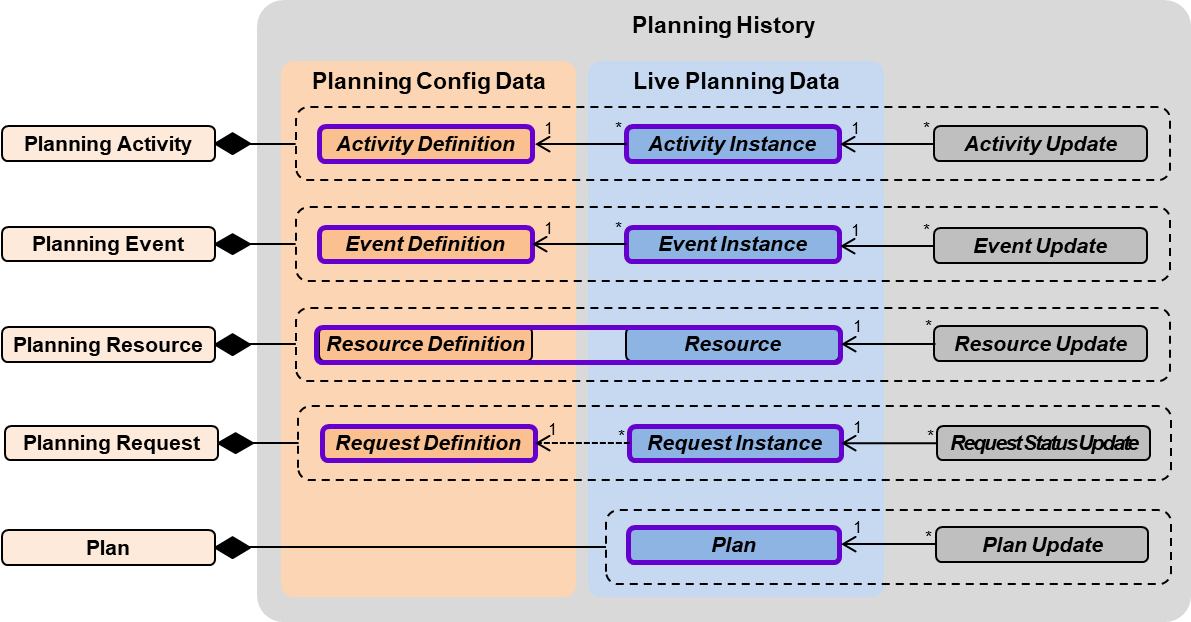


Figure 2‑5 : MPS Service Objects and their Constituent MO Objects

Each MO object has a unique object **identity**, which includes an unchanging key (Identifier) and optionally a **version**. When a **definition** object is updated, it retains the same key, but its **version** is updated to uniquely identify a specific version of the definition. Planning request instances and plans also have an associated version.

The **definition** objects associated with **planning activities**, **planning events**, **planning resources** and **planning requests** all form part of the planning configuration data that must be available to both communicating parties that exchange **planning requests** and **plans**. Definition objects comprise only static fields. The MPS services defined in this Recommended Standard do not address the bulk transfer of MPS configuration data between communicating parties, but individual definitions can be accessed using the Plan Information Management Service.

The **instance** objects associated with all MPS service objects are the live planning data created by and exchanged between communicating parties during planning and plan execution. Instance objects may comprise both static and dynamic fields. For **planning resources**, the live instance has the same object **identity** as the associated **definition**,the only distinction being that the live instance may include dynamically updating fields, such as the value of a **planning resource.**

Changes in state of the dynamic fields of live planning data **instance** objects may be notified through an **update** structure exchanged through MPS service messages and optionally stored in planning history. These updates are not themselves MO objects, but reference the corresponding instance and the timestamp of the update as well as the value of dynamic fields.

**Planning requests** may reference **planning activities**, **planning events**,and **planning resources** but do not contain instances of these items. Instead they specify the **activity details** for the requested planning activities.

**Plans** contain instances of planned activities, and optionally of planning events. They may also optionally contain planning resource profiles that express their planned evolution over time.

### Planning Request

**Planning requests** are the main input to the planning function. A planning request is a container for the information needed to be exchanged between the requester and the planner. It supports the specification of a request to plan one or more **planning activities**. Alternatively, it can support a request to use an existing **plan** (already containing a number of **planning activities**)as an input to the planning process. It can constitute a one-off planning request, or request the repetitive planning of activities as a ‘standing order’.

The main characteristic of the planning request is that, being a container, it needs to hold references to, or instances of, the constituent information items that are required by the planner and agreed by the interacting parties for exchange at interface level. It has one or more **planning activities** as the basis of the request. In addition, the request may optionally reference **planning events**. Information about **planning constraints** on when a requested activity can or shall be planned may be exchanged as part of the planning request, by referencing constraints on the timing or position of planning activities, both absolute and relative to planning events or other planning activities, and on the state of **planning resources**.

### Plan

The **plan** is the output of a planning process. The plan is basically a container of one or more selected **planning activities**, optionally associated with **planning events**. In addition, the usage of **planning resources** may be contained in the plan. The plan may contain specific information from the planning process, which applies to the plan as a whole. In the hierarchical and distributed planning concepts, the output of one planning function could be the input of another one. As such, a **planning request** could refer to an entire plan.

Plans may be iterative, and therefore overlap with the previous plan. This introduces the notion that a plan may have an identified predecessor or **precursor plan**, and also that if a planning service object is contained in multiple iterations of the plan, then it should have the same unique **identity** (except for an updated **version**) in each successive iteration of the plan to avoid ambiguity and duplication.

Plans comprise the following main elements:

* Plan Information: header data relating to the **plan** as a whole;
* Planned Items: the set of contained **planning activities** and **planning events**;
* Plan Revisions [optional]: summaries of the changes between this version of the plan and another specified version of a plan, usually its precursor plan;
* Plan Resources [optional]: value profiles covering the period of the plan for a set of **planning resources**.

### Planning Activity

A **planning activity** is the basic building block for the planning: a meaningful unit of what can be planned. As such, it has to be understood by the planning function. It could eventually be translated to something that can be executed by a plan execution function; this includes CCSDS MO M&C **actions** (reference [D4]) (that may represent telecommands) and CCSDS MO Automation **procedures** (reference [D1])(that may represent any automated telecommand sequence, operational procedure, on-board control procedure, or function).

Planning activities support hierarchy: a planning activity may be composed of one or more subordinate planning activities. A planning activity may define **arguments** (parameters), which could be used to instantiate a specific planning activity in a plan, based on its generic definitions. Arguments of a planning request or planning event can be passed through to the arguments of planning activities resulting from these. Arguments can then similarly cascade down through a hierarchy of planning activities. A plan execution function may then flow down these arguments to any **action** or automated **procedure** initiated.

**Planning constraints** can also be associated with a planning activity, either generic constraints applicable to all occurrences (or instances) of the planning activity that are contained within its definition, or specific constraints associated with a particular instance that are defined in the context of the **planning request**. These planning constraints can be expressed in terms of the timing or position of a planning activity, both absolute and relative to planning events or other planning activities, and on the state of **planning resources**.

### Planning Event

A **planning event** marks when a condition is being met, expressed in terms of time or position. They may be used to represent predicted or planned events, such as predicted orbital events or planned periods of contact with a spacecraft, that are typically received as an input by the mission planning function, from an external function, such as navigation.

Planning events may be grouped hierarchically to represent a compound event, such as the start and end of a satellite pass over a ground station (AOS/LOS), or a satellite passing through eclipse (penumbra entry, umbra entry, umbra exit, penumbra exit). A planning event may define **arguments** (parameters) to convey additional information relevant to the planning process.

**Planning activities** may be linked to a related planning event. The start or end of the planning activity can be relative to the planning event, and the **arguments** of the event can be flowed down to the planning activity. **Planning requests** may also reference planning events, associating them with requested planning activities.

Planning events may be classified as **predicted events** or **potential events**:

* Predicted events are those that are expected to occur at a particular time or position that is known at the time of planning and can be contained within a **plan**. Uncertainty in the timing of predicted events may be refined closer to their time of occurrence. This can either be handled by re-planning, or by updating the events within an executing plan.
* Potential events are not predictable, but may still have a defined response within a plan: virtual observatory Target Of Opportunity (TOO) events are an example. Such events can be inserted into an executing plan.

### Planning Resource

A **planning resource** is an abstract status modelling the state of the system being planned. It may be necessary to model some aspects of system state in order to:

* trigger the execution of a planning activity;
* constrain the execution of a planning activity;
* define the effect that the execution of a planning activity has on the planning resource.

A planning resource is in effect a value of defined type that can evolve over time. A resource profile can be used to capture and communicate that evolution over time in the context of a **plan**.

If an event or constraint on a planning activity needs to be expressed in terms of the state of the system (rather than just time or position) then this corresponds to the state of planning resources. This is considered not internal to the planning function, if it forms part of the **planning request** or **plan**.

A planning resource could in principle be considered as information that is internal to the planning system. However, some resources may be shared across multiple planning entities. As such, information regarding a resource may need to be communicated between entities, and therefore has to be referenced as part of a planning request or of the plan, in terms of requested or consumed resources respectively. This may include the initialization or synchronization of planning resource values at specific points in the plan.

Planning resources are an optional element of the MPS information model. There is no requirement for a compliant MPS system to support them.

## MPS Services Overview

### General

This document specifies standards for the following MPS services, which are introduced in the following subsections:

* Planning Request Service (PRS);
* Plan Distribution Service (PDS);
* Plan Execution Control Service (PECS);
* Plan Information Management Service (PIMS);
* Plan Edit Service (PES).

A compliant MPS system may support only a subset of these services. None of the services is mandatory; any subset can be supported.

Each service comprises a set of service operations that the service consumer can invoke on the service provider. Service operations reference the MO objects defined in the MPS information model. Each service operation follows one of the standard MO MAL interaction patterns that may comprise multiple messages flowing in both directions between service consumer and provider. Where not fully defined within the MAL, the body of these messages forms part of the service specification and are defined in terms of MAL Attribute data types and/or data structures defined in the MPS information model.

Service operations are grouped into capability sets. A compliant MPS system may only support a subset of capability sets for each supported service.

### Planning Request Service

The Planning Request Service is offered by the planning function of an MPS system to enable its users to submit, cancel, and modify planning requests, as well as to receive feedback on their status. The service may be used by another planning function in a hierarchical or distributed MPS system, or by an MPS system user.

Planning requests are defined in 4.5.5 and may include a set of requested planning activities or an existing plan (the output of a planning function in a hierarchical or distributed MPS system).

If a Plan is used as the body of the request, this can either be embedded within the request, or passed by reference. If passed by reference, the Plan Distribution Service can be used to retrieve the Plan. In a hierarchical or distributed planning system, the domain can be used to identify where to retrieve it from.

### Plan Distribution Service

The Plan Distribution Service is offered by the planning function of an MPS system to enable its users to obtain the plans output by it, as well as to receive feedback on their status. The service may be used by another planning function in a hierarchical or distributed MPS system, or by an MPS system user. Plans are defined in 4.5.6.

The service does not provide the capability to control the planning function itself or to generate plans. This capability can be supported if the planning function exposes a standard set of MO Monitoring & Control services. Submission of plans to a plan execution function is supported by the Plan Execution Control Service (see 3.7).

### Plan Execution Control Service

The Plan Execution Control Service is offered by an MPS system’s plan execution function to enable its users to submit (and revoke) plans for execution; to control their execution at plan, sub-plan, and activity levels; and to receive feedback on their execution status.

The Plan Execution Control Service may be used by a planning function, or by an MPS system user responsible for mission operations.

Plans are defined in 4.5.6, including the state model for plans to which the operations of the service correlate.

Sub-plans are not defined as an MO object in the MPS information model, but are specified by an Identifier associated with the constituent ActivityInstances contained in a Plan. This can be used to sub-divide a Plan based on domain (spacecraft or subsystem), operational responsibility, or another criterion. Each ActivityInstance can only be associated with a single sub-plan. Control may be exercised via the service at the level of sub-plans.

### Plan Information Management Service

The Plan Information Management Service is offered by the planning function of an MPS system to enable its users to list and retrieve available definitions for MPS service objects, including: planning requests, planning events, planning activities, planning resources, and MPS system configuration data. The service may also be offered by a plan execution function.

The service does not support the transfer of planning configuration data to planning or plan execution functions, which is outside the scope of the current MPS services. Nor does it support the insertion or modification of MPS service object definitions.

The structure of the various MPS service object definitions is given in 4.5.

### Plan Edit Service

The Plan Edit Service is offered by an MPS system’s plan execution function to enable its users to modify plans that have already been submitted for execution. It allows an external user or function to update the status of the plan; insert, modify, or delete planning activity and event instances; update the value of resources; and apply a time shift to a plan.

This may be used by expert mission operations users in a non-nominal operational scenario to modify a plan that is executing or about to execute in order to avert or recover from a failure. Where there is sufficient time, it is recommended to re-plan using the nominal planning process rather than to edit the plan directly, as this circumvents any constraint checking performed by the planning function.

Another use of the service is for a third party functions to update elements of the plan to reflect information available in near-real-time. Examples are:

* Update of the input arguments of a planned activity instance to fine tune its behaviour in response to currently observed status.
* Update of a predicted planning event instance that is already contained within a plan, with refined timing or other details.
* The injection of instances of planning events. These may be detected in real-time and correlate to potential (rather than predicted) events for which a planned response is available. An example is a ‘Target of Opportunity’ event that may be notified to an astronomical observatory mission.
* Update of a resource value by an external modelling function to more accurately reflect currently observed status.

Plans are defined in 4.5.6, while their constituent planned items are defined in 4.5.2 (planning activities), 4.5.3 (planning events) and 4.5.4 (planning resources).

## Optional Elements of the Recommended Standard

The MPS services specification defines a substantial MPS information model and five services. Compliance of an individual mission planning system deployment to the MPS service specification does not imply that either the complete set of services specified here or the full MPS information model has to be supported.

The level of compliance of a specific deployment to the Recommended Standard can be selected as follows:

1. the set of MPS services supported;
2. the capability sets supported within each MPS service;
3. the optional elements of the MPS information model supported.

# **MPS Service Specifications**

## Overview

This section contains the service specifications for the MPS services. An overview of the services has been given previously in 2.5 above. Each of the following services is defined in a separate section below:

* Planning Request Service;
* Plan Distribution Service;
* Plan Execution Control Service;
* Plan Information Management Service;
* Plan Edit Service.

There is no requirement to implement all services. Any subset of the services may be supported by an MPS deployment.

Each service specification comprises the following parts:

1. Overview: a brief overview of the service.
2. Definition: a table listing the service operations and grouping them into capability sets.
3. Discussion: a more detailed description of the service and its operations, including the set of MPS service objects (defined in section 4) that are operated on by the service or referenced by service operations.
4. High-Level Requirements: such as the set of configuration data required by the service.
5. Functional Requirements: specific requirements on the behaviour of the service provider and consumer.
6. Operations: specification of each service operation.

All services defined in this document are part of the MPS Area, which has the Area number 5.

## Optional Elements

Table 3-1 defines the optional capability sets and information model elements for all MPS services. Optional capabilities and elements are shown with a grey background. The set of operations of which each capability set is comprised is found in section 3.

For each service capability set, applicability of information model element sets are shown as follows:

✓ Required; if a service capability set is supported, then the information model element sets indicated by this symbol must also be supported.

O Optional; this symbol is used to show that one or more service capability set operations include messages that contain data structures from the indicated optional element sets.

- Not Applicable; indicates that none of the service capability set operations contain data structures from optional element sets.

An indication is also given of which service specific data structures are relevant to the operations of each service capability set. It should be noted this is shown at the level of the MPS service objects defined in 4.5: only the specific data structures used in the messages of the associated service operations are required.

Table 3‑1: Optional Service Capabilities

| Services | | Information Model | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Service | Capability Set | Core Features | Basic Constraints | Plan Revisions | Resources | Position & Direction | Functions | MPS Service Objects for which MO Objects or Service Data Structures are used in Service Operations |
| PRS | 1: Submit Request / Status | ✓ | O | - | O | O | O | Planning Request |
| 2: Cancel Request | ✓ | - | - | - | - | - | Planning Request |
| 3: Update Request | ✓ | O | - | O | O | O | Planning Request |
| 4: Subscribe to Status | ✓ | - | - | - | - | - | Planning Request |
| 5: Retrieve Request | ✓ | O | - | O | O | O | Planning Request |
| PDS | 1: List & Obtain Plans | ✓ | O | O | O | O | O | Plan |
| 2: Subscribe to Status | ✓ | - | - | - | - | - | Plan |
| 3: Subscribe to Plans | ✓ | O | O | O | O | O | Plan |
| 4: Query Plans | ✓ | O | O | O | O | O | Plan |
| 5: Retrieve Partial Plan | ✓ | O | O | O | O | O | Plan |
| PECS | 1: Submit Plan / Status | ✓ | O | O | O | O | O | Plan |
| 2: Plan Activation | ✓ | - | O | - | - | - | Plan |
| 3: Subscribe to Status | ✓ | - | - | - | - | - | Plan |
| 4: Subscribe to Detail | ✓ | - | - | O | O | - | Planning Activity/Event/Resource |
| 5: SubPlan Activation | ✓ | - | - | - | - | - | Plan (SubPlan) |
| 6: Subscribe to SubPlans | ✓ | - | - | - | - | - | Plan (SubPlan) |
| 7: Activity Suspension | ✓ | - | - | - | - | - | Planning Activity |
| 8: Activity Status | ✓ | - | - | - | - | - | Planning Activity |
| PIMS | 1: Request Definitions | ✓ | O | - | O | O | O | Planning Request |
| 2: Event Definitions | ✓ | - | - | - | - | - | Planning Event |
| 3: Activity Definitions | ✓ | O | - | O | O | O | Planning Activity |
| 4: Resource Definitions | ✓ | - | - | ✓ | - | - | Planning Resource |
| PES | 1: Update Plan Status | ✓ | - | - | - | - | - | Plan |
| 2: Insert/Delete Activity/Event | ✓ | O | - | O | O | O | Planning Activity/Event |
| 3: Update Activity/Event | ✓ | - | - | - | O | - | Planning Activity/Event |
| 4: Update Resource | ✓ | - | - | ✓ | - | - | Planning Resource |
| 5: Update Resource Profile | ✓ | - | - | ✓ | - | - | Planning Resource |
| 6: Apply Time Shift | ✓ | - | - | - | - | - | Plan |

## Conventions

### Services

For each service specification, a summary table is provided. This defines the area, service and version numbers used in MAL message headers, and lists the service operations together with their MAL interaction patterns, operation numbers, and the capability set to which they belong.

Table 3‑2 : Example Service Table

| Area Identifier | Service Identifier | Area Number | Service Number | Area Version |
| --- | --- | --- | --- | --- |
| MPS | <Service name> | 5 | <#> | <#> |

| Interaction Pattern | Operation Identifier | Operation Number | Capability Set |
| --- | --- | --- | --- |
| REQUEST | submitRequest | <#> | <#> |
| … | … | … | … |

### Service Operations

The definition of each MPS service operation in this section contains a table based on the appropriate interaction pattern template for the applicable MAL interaction pattern (see reference [2]).

Table 3‑3 : Example Operation Template

| Operation Identifier | <Operation name> |
| --- | --- |
| Interaction Pattern | PUBLISH-SUBSCRIBE |
| Subscription Keys | planID : (MAL::Identifier)  precursor : (MAL::Identifier)  status : (MAL::UShort)  originator : (MAL::Identifier) |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| OUT | PUBLISH/NOTIFY | Yes¦No | Field name : (data type) |
| … | … | … | … |

Light blue cells (grey when printed on a monochrome printer) contain table headings, light grey cells contain fields that are fixed for a pattern, and white cells contain values that   
are specific to the operation.

The subscription keys row is only included in the case of a PubSub operation and lists the subscription key fields (name and type) specified for the operation.

The message direction denotes the direction of the message relative to the provider of the  
service and is either IN or OUT. So all messages directed towards the provider are IN  
messages, and all messages directed away from the provider are OUT messages.

The type signature contains a list of message fields, often a single field referencing an MPS data structure, but in some cases there may be a series of fields each identified by a field name and data type. The nullable column indicates whether each of the listed message fields is nullable or not.

### Errors

Errors that may be returned by the operation are listed in a simple table, which references the error by name and the area in which it is defined. MPS errors are defined in section 5, together with the corresponding error number and the type and description of any ExtraInfo field. Standard MAL errors (not listed) may also be returned.

Table 3‑4 : Example Error References Table

| Error | Area |
| --- | --- |
| <Error name> | <Area> |
| … | … |

## External Definitions

### Interaction Patterns

The specifications of the operations supported by each service are given in terms of interaction patterns. These interaction patterns describe a standard message exchange pattern, such that these details need not be given for each operation individually. The standard MAL interaction patterns and their associated operation templates are used, both of which are defined in reference [2].

### MO Errors

Operations that did not finish successfully may, depending on the failure mode, return an MO Error as defined in section 5 of reference [2]. All operations may return one of the MAL-specific MO Errors from reference [2]. In addition, some operations may return an MPS-specific MO Error; where this is applicable, this is indicated in the error table for that operation. The different types of MPS-specific MO Errors and the cases in which they shall be returned are specified in section 5 of this standard.

### Publish-Subscribe Operations

The Publish-Subscribe operations defined in this standard follow the interaction pattern described in section 3.6.6 of reference [2]. Due to the use of a general-purpose MAL broker in its implementation, this interaction pattern does not support MPS Enum types as subscription filter keys. Instead, where a defined subscription key corresponds to an MPS enumerated value, the subscription key shall be defined with the type MAL::UShort, where the integer corresponds to the enumerated value.

NOTES

1. The broker filters updates by applying all the specified criteria (the **domain** and specified subscription key filters are ANDed by the broker). When multiple values are passed for a single subscription key filter, its option are ORed by the broker.
2. Where most MPS operations may return an INVALID error when invoked, that is explicitly not the case for any Publish-Subscribe operations; those operations interact with the MAL broker and are hence limited to only MAL-compatible errors (such as INVALID).

## Service: Planning Request Service

### Overview

The Planning Request Service, introduced in 2.5.2, is provided by a planning function and enables its consumers to manage the submission of planning requests and to receive feedback on their status. It comprises the operations defined below, of which only those in capability set 1 are mandatory.

In the context of a hierarchical or federated planning system, the Planning Request Service submitRequest operation can be used to submit a Plan (4.5.6) to a planning function, either embedding the Plan in the request itself or passing it by reference. If passed by reference, the Plan can be retrieved using the Plan Distribution Service (3.6). Patch plans are not permitted in the context of a planning request.

### Definition

| Area Identifier | Service Identifier | Area Number | Service Number | Area Version |
| --- | --- | --- | --- | --- |
| MPS | PlanningRequest | 5 | 1 | 1 |

| Interaction Pattern | Operation Identifier | Operation Number | Capability Set |
| --- | --- | --- | --- |
| REQUEST | submitRequest | 1 | 1 |
| REQUEST | getRequestSummaries | 2 |
| PROGRESS | getRequestStatus | 3 |
| SUBMIT | cancelRequest | 4 | 2 |
| REQUEST | updateRequest | 5 | 3 |
| PUBSUB | monitorRequestStatus | 6 | 4 |
| PROGRESS | getRequest | 7 | 5 |

### Discussion

#### General

Three of the operations are concerned with the submission and subsequent management of planning requests, enabling a service consumer to submit a planning request, and (where supported) to update or cancel that request.

The submitRequest operation results in the creation of a new RequestInstance, returning its identity in the response message. The updateRequest operation results in a new version of an existing RequestInstance, returning its identity (key and version) in the response message. The cancelRequest operation stops the referenced RequestInstance being considered in the generation of future Plans, but does not imply that it is immediately deleted by the provider.

The remaining operations are concerned with obtaining feedback on the status or content of planning requests.

The getRequestSummaries operation requests a list of available planning requests based on a supplied filter. It returns a list of RequestSummaryStatuses that contain the identity, header information and status for each available RequestInstance that matches the filter.

A consumer may already hold the identity (and static content) of RequestInstances, either because it has submitted planning requests, or previously performed a getRequestSummaries operation. It can then request an update of their dynamic status using the getRequestStatus operation. The request lists references to RequestInstances and the response provides a corresponding list of RequestStatusUpdates.

Where supported, consumers may also subscribe to receive RequestStatusUpdates as they become available via the monitorRequestStatus. This uses the standard MAL publish-subscribe pattern and notifies the consumer with RequestStatusUpdates corresponding to the specified subscription keys as they are published by the provider.

The getRequest operation, where supported, enables a consumer to retrieve the full content of one or more planning requests. This is similar to the getRequestStatus operation, but instead of returning RequestStatusSummaries it returns RequestInstances. It should be noted that this is not an archive service, only planning requests currently being managed by the provider can be returned.

#### MPS Service Objects

MPS service objects relevant to the planning request service and their relationships are defined within the MPS information model in 4.5.

The following MO objects are directly applicable to the service:

* RequestDefinition;
* RequestInstance.

RequestDefinitions are configuration data that provide re-usable templates for planning requests, but are not required for the submission of an ad-hoc planning request. Their identity comprises both key and version.

RequestInstances are created in response to the submission of a planning request and contain both static data and dynamic status information. As planning requests can be updated, their identity comprises both key and version assigned by the service provider. The planning request submitted by the service consumer contains only the static data needed to specify the planning request in the form of a PlanningRequestDetails structure. If the planning request is derived from a RequestDefinition, this is referenced in the PlanningRequestDetails and the RequestInstance created from it.

As the RequestInstance is unknown at the time of submitting a planning request, the consumer may provide a user reference. This is returned by the provider together with the identity of the RequestInstance in the response to the planning request submission.

Updates to the dynamic status information associated with a RequestInstance are reported by the service provider to the service consumer using the RequestStatusUpdate structure. The RequestSummaryStatus structure is used to provide the identity, descriptive header fields, and current status of available RequestInstances.

The following MO objects may be referenced in the context of the planning request service:

* **PlanningUser:** the user responsible for submitting the planning request;
* **ActivityDefinition:** the activities specified to be planned in the form of ActivityDetails structures in the body of the planning request;
* **EventDefinition and EventInstance:**  references to planning events can be contained within ActivityDetails and Constraints contained in the body of the planning request;
* **Resource [Optional]:** references to planning resources can be contained within the Constraints contained in the body of the planning request;
* **Plan:** a Plan or reference to a Plan can be used as the body of a planning request. Once the planning request has been planned, a reference to the resulting output Plan may also be included in the RequestInstance.

### High-Level Requirements

The following set of mission planning configuration data shall be available to both provider and consumers in any deployment of the planning request service:

1. Planning Request Definitions (as RequestDefinition objects [4.5.5.1]);
2. Planning Activity Definitions (as ActivityDefinition objects [4.5.2.1]);
3. Planning Event Definitions (as EventDefinition objects [4.5.3.1]);
4. Planning Resource Definitions (as Resource objects [4.5.4.2]) [Optional];

### Functional Requirements

In response to a submitRequest operation, the service provider shall create a corresponding RequestInstance object, set its creationTime and return its identity (key and version) to the consumer.

In response to an updateRequest operation, if supported, the service provider shall create a new version of the referenced RequestInstance object, set its creationTime and return its identity (key and version) to the consumer.

The service provider shall only consider the latest version of a RequestInstance in the generation of future Plans, following a successful updateRequest operation.

In response to a cancelRequest operation, if supported, the service provider shall set the requestStatus of the referenced RequestInstance object to ‘CANCELLED’ and stops it being considered in the generation of future Plans.

In response to internal planning and feedback from external plan execution processes, the service provider shall model the status of RequestInstance objects in accordance with the planning request state model (4.5.5.2).

NOTE – Following a successful updateRequest operation, it is implementation dependent what the status of the previous version of the RequestInstance is set to. Previous versions may already have been incorporated into Plans.

If a service consumer generates a planning request based on an existing planning request definition, then the PlanningRequestDetails submitted shall:

1. Reference the source RequestDefinition (key and version) in the definition field;
2. Contain a matching set of arguments (name and type) to the RequestDefinition in the arguments field;
3. Have a matching value to the RequestDefinition in the standingOrder field.

NOTE – While typically the PlanningRequestDetails will also have matching content to the RequestDefinition in the activities and constraints fields it is allowed to add and remove individual activities and constraints.

In the context of a planning request that contains or references an existing Plan, the Plan shall be a full plan, patch plans are not permitted.

Following successful inclusion of a planning request in a generated Plan, the service provider shall update the outputPlanRefs field of the RequestInstance with the identity of the Plan (key and version).

NOTE – The consumer may then use the reference to the output Plan to retrieve the Plan or its status using the PlanDistributionService. Where the planning request has been incorporated into multiple alternate plans, these will be listed in the outputPlanRefs field.

### Operation: submitRequest

#### Overview

The submitRequest operation sends a planning request to the provider, which then creates a corresponding RequestInstance object and returns its identity to the consumer.

#### Definition

| Operation Identifier | submitRequest |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No | requestDetails : (PlanningRequestDetails) |
| OUT | RESPONSE | No | requestResponse : (PlanningRequestResponse) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| UNSUPPORTED | MPS |

### Operation: getRequestSummaries

#### Overview

The getRequestSummaries operation allows consumers to obtain a filtered list of currently available RequestInstances. The request uses the RequestFilter structure to select the set of planning requests of interest, using the following keys:

* Domain of the RequestInstance;
* Instance ID (key and version) of the RequestInstance;
* Creation date and time of the RequestInstance version (as a time range);
* Definition ID (key and version) of the RequestInstance (the RequestDefinition from which it was created);
* User ID of the PlanningUser who initiated the RequestInstance;
* User Reference supplied by the User when submitting the RequestInstance.

The response returns a list of RequestSummaryStatus structures containing the identity (key and version), descriptive header fields, and status of the RequestInstances that match the filter.

#### Definition

| Operation Identifier | getRequestSummaries |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No | requestFilter : (RequestFilter) |
| OUT | RESPONSE | No | requestSummaries : (List <RequestSummaryStatus>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

### Operation: getRequestStatus

#### Overview

The getRequestStatus operation is used to obtain the current status of one or more known RequestInstances. The operation uses the Progress interaction pattern, to allow the response to be spread across multiple messages.

#### Definition

| Operation Identifier | getRequestStatus |
| --- | --- |
| Interaction Pattern | PROGRESS |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | PROGRESS | No | requestRefs : (List <MAL::ObjectRef <RequestInstance>>) |
| OUT | ACK | No | Empty |
| OUT | UPDATE | No | requestStatuses : (List <RequestStatusUpdate>) |
| OUT | RESPONSE | No | Empty |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

### Operation: cancelRequest

#### Overview

The cancelRequest operation is used by a consumer to cancel a previously submitted planning request. The service provider acknowledges the cancellation of the RequestInstance or returns an error.

#### Definition

| Operation Identifier | cancelRequest |
| --- | --- |
| Interaction Pattern | SUBMIT |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | SUBMIT | No | requestRef : (MAL::ObjectRef <RequestInstance>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| CANCEL\_FAILED | MPS |

If it was not possible to cancel the RequestInstance, for example because the resultant activities have already been executed or activated within a plan execution function, then the CANCEL\_FAILED error shall be returned.

### Operation: updateRequest

#### Overview

The updateRequest operation may be used to modify the PlanningRequestDetails associated with a previously submitted planning request. This results in the creation of a new version of the RequestInstance (with the same key) by the service provider, which returns the identity (key and version) of the new version to the consumer.

#### Definition

| Operation Identifier | updateRequest |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No  No | requestRef : (MAL::ObjectRef <RequestInstance>)  requestDetails : (PlanningRequestDetails) |
| OUT | RESPONSE | No | requestResponse : (PlanningRequestResponse) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| UNSUPPORTED | MPS |
| UPDATE\_FAILED | MPS |

If it was not possible to update the RequestInstance, for example because the resultant activities have already been executed or activated within a plan execution function, then the UPDATE\_FAILED error shall be returned.

### Operation: monitorRequestStatus

#### Overview

The monitorRequestStatus operation is used to subscribe to status updates for a filtered set of planning RequestInstances. The operation uses the Publish-Subscribe interaction pattern, with the body of the notification message comprising a RequestStatusUpdate for a subscribed RequestInstance.

#### Definition

| Operation Identifier | monitorRequestStatus |
| --- | --- |
| Interaction Pattern | PUBLISH-SUBSCRIBE |
| Subscription Keys | instanceID : (MAL::Identifier)  definitionID : (MAL::Identifier)  userID : (MAL::Identifier)  userReference : (MAL::Identifier)  status : (MAL::UShort)  outputPlanID : (MAL::Identifier) |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| OUT | PUBLISH/NOTIFY | No | requestStatusUpdate : (RequestStatusUpdate) |

#### Requirements

The monitorRequestStatus subscription shall be based on the provision of the following keys in addition to the domain of the required RequestInstances in the Register message, all of which are nullable:

1. InstanceID: instance key as MAL::Identifier;
2. DefinitionID: definition key as MAL::Identifier;
3. UserID: user key as MAL::Identifier;
4. UserReference: userReference of subscribed RequestInstances as MAL::Identifier;
5. Status: status as MAL::UShort [RequestStatusEnum];
6. OutputPlanID: outputPlanRef key as MAL::Identifier.

NOTE – For Status, the enumerated value associated with the RequestStatusEnum must be used, and not the associated string.

#### Errors

This operation returns no errors in addition to the standard MAL errors.

### Operation: getRequest

#### Overview

The getRequest operation is used to obtain the full content of one or more known RequestInstances. The operation uses the Progress interaction pattern, to allow the response to be spread across multiple messages.

#### Definition

| Operation Identifier | getRequest |
| --- | --- |
| Interaction Pattern | PROGRESS |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | PROGRESS | No | requestRefs : (List <MAL::ObjectRef <RequestInstance>>) |
| OUT | ACK | No | Empty |
| OUT | UPDATE | No | requestInstances : (List <RequestInstance>) |
| OUT | RESPONSE | No | Empty |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

## Service: Plan Distribution Service

### Overview

The Plan Distribution Service, introduced in 2.5.3, is provided by a planning function and enables its consumers to access generated plans and to receive updates on their status. It comprises the operations defined below, of which only those in capability set 1 are mandatory.

### Definition

| Area Identifier | Service Identifier | Area Number | Service Number | Area Version |
| --- | --- | --- | --- | --- |
| MPS | PlanDistribution | 5 | 2 | 1 |

| Interaction Pattern | Operation Identifier | Operation Number | Capability Set |
| --- | --- | --- | --- |
| REQUEST | getPlanSummaries | 1 | 1 |
| PROGRESS | getPlan | 2 |
| REQUEST | getPlanStatus | 3 |
| PUBSUB | monitorPlanStatus | 4 | 2 |
| PUBSUB | monitorPlan | 5 | 3 |
| PROGRESS | queryPlan | 6 | 4 |
| REQUEST | getPartialPlan | 7 | 5 |

### Discussion

#### General

NOTE – The getPlanStatus, with the exception of the service number, is identical to the operation of the same name in the Plan Execution Control Service.

The mandatory operations of capability set 1 allow a service consumer to access plans, or the status of plans, generated by a planning function.

To retrieve a Plan, the consumer needs to have the identity of that Plan (key and optionally version). A reference to the output Plan associated with a planning request may have been received through planning request service feedback. Otherwise, the consumer must use the getPlanSummaries operation to discover available plans.

The getPlanSummaries operation returns PlanSummaries for the set of currently available plans that meet the specified filter criteria. These give the identity of the plans, the associated plan information (context and descriptive fields) and the current status of the Plan, but do not include the full content of the plans.

The getPlan operation returns the full content of the referenced Plan(s).

The getPlanStatus operation returns only the current status of the referenced Plan(s) as PlanUpdates.

Where supported, the monitorPlanStatus operation enables the consumer to subscribe to automatic notification of changes in status of plans as PlanUpdates. Similarly the monitorPlan operation forwards new Plans matching the subscription to the consumer.

Two further non-mandatory operations allow more complex selection of plans.

The queryPlan operation allows the consumer to request plans based on multiple query criteria, including various header fields and on whether the Plan includes specific planning activities or events.

The getPartialPlan operation allows the consumer to request a subset of a Plan, based on the domain, subPlan allocation, or tags associated with the contained planning activities, as well as a period of the plan based on time, position, or events.

#### MPS Service Objects

MPS service objects relevant to the plan distribution service and their relationships are defined within the MPS information model in 4.5.

The following MO objects are directly applicable to the service:

* Plan.

The identity of Plans comprises both key and version. Plans are created by a planning function as an output that may be passed to a plan execution function, or to another planning function in a distributed or hierarchical planning system. Where a Plan is updated due to replanning, a new version of the Plan with the same key may be created.

Plans may be stand-alone or reference a defined precursor with which they may overlap. In the latter case, a patch plan may be created which only contains the differences from the precursor plan, and includes a reference to the target Plan which results from applying these changes to the precursor Plan.

A selected subset of a Plan can be extracted as a PartialPlan. This is specified as a shorter period covered by the plan, or by the domain, SubPlan, or tags associated with its contained planning activities.

The following MO objects may be referenced in the context of the planning request service:

* ActivityDefinition and ActivityInstance: the ActivityInstances contained within a plan may be referenced in service operations by their associated ActivityDefinition;
* EventDefinition and EventInstance: the EventInstances contained within a plan may be referenced in service operations by their associated EventDefinition;
* Resources may be contained within a Plan;
* RequestInstance: the ActivityInstances contained within a Plan reference their source planning request.

### High-Level Requirements

The following set of mission planning configuration data shall be available to both provider and consumers in a any deployment of the plan distribution service:

1. Planning Activity Definitions (as ActivityDefinition objects [4.5.2.1]);
2. Planning Event Definitions (as EventDefinition objects [4.5.3.1]);
3. Planning Resource Definitions (as Resource objects [4.5.4.2]) [Optional];

### Functional Requirements

If Plan version 0 is specified in the getPlan and getPlanStatus operations, then the service provider shall assume the latest version of the Plan is required.

In response to internal planning and feedback from external plan execution processes, the service provider shall model the status of Plan objects in accordance with the plan state model (see 4.5.6.2).

NOTE – A planning function that is a service provider is not required to support real-time provision of Plan status changes. Feedback from plan execution functions may only be processed periodically by the planning function as part of a planning cycle.

### Operation: getPlanSummaries

#### Overview

The getPlanSummaries operation allows consumers to obtain a filtered list of currently available Plans. The request uses the PlanFilter structure to select the set of plans of interest, using the following keys:

* Domain of the Plan;
* planID (key and version) of the Plan;
* precursorPlan of the Plan;
* status of the Plan;
* originator of the Plan;
* validity period of the Plan as a time window.

The response returns a list of PlanSummaryStatus structures containing the identity (key and version), descriptive header fields, and status of the Plans that match the filter.

#### Definition

| Operation Identifier | getPlanSummaries |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No | planFilter : (PlanFilter) |
| OUT | RESPONSE | No | planSummaries : (List <PlanSummaryStatus>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

### Operation: getPlan

#### Overview

The getPlan operation is used to obtain the full content of one or more known Plans. The operation uses the Progress interaction pattern, to allow the response to be spread across multiple messages.

#### Definition

| Operation Identifier | getPlan |
| --- | --- |
| Interaction Pattern | PROGRESS |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | PROGRESS | No | planRefs : (List <MAL::ObjectRef <Plan>>) |
| OUT | ACK | No | Empty |
| OUT | UPDATE | No | retrievedPlan : (Plan) |
| OUT | RESPONSE | No | Empty |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

### Operation: getPlanStatus

#### Overview

The getPlanStatus operation is used to obtain the current status of one or more known Plans. The operation uses the Request interaction pattern.

#### Definition

| Operation Identifier | getPlanStatus |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No | planRefs : (List <MAL::ObjectRef <Plan>>) |
| OUT | RESPONSE | No | responsePlans : (List <PlanUpdate>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

### Operation: monitorPlanStatus

#### Overview

The monitorPlanStatus operation is used to subscribe to status updates for a filtered set of Plans. The operation uses the Publish-Subscribe interaction pattern, with the body of the notification message comprising a PlanUpdate for a subscribed Plan.

#### Definition

| Operation Identifier | monitorPlanStatus |
| --- | --- |
| Interaction Pattern | PUBLISH-SUBSCRIBE |
| Subscription Keys | planID : (MAL::Identifier)  precursor : (MAL::Identifier)  status : (MAL::UShort)  originator : (MAL::Identifier) |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| OUT | PUBLISH/NOTIFY | No | planUpdate : (PlanUpdate) |

#### Requirements

The monitorPlanStatus subscription shall be based on the provision of the following keys in addition to the *domain* of the required Plans in the Register message, all of which are nullable:

1. PlanID: key of subscribed Plan as MAL::Identifier;
2. Precursor: key of precursorPlan as MAL::Identifier;
3. Status: status as MAL::UShort [PlanStatusEnum];
4. Originator: originator of subscribed Plan as MAL::Identifier.

NOTE – For Status, the enumerated value associated with the PlanStatusEnum must be used, and not the associated string.

#### Errors

This operation returns no errors in addition to the standard MAL errors.

### Operation: monitorPlan

#### Overview

The monitorPlan operation is used by a consumer to subscribe to receive new Plans, or new versions of Plans, as they published. The operation uses the Publish-Subscribe interaction pattern, with the body of the notification message comprising a Plan.

#### Definition

| Operation Identifier | monitorPlan |
| --- | --- |
| Interaction Pattern | PUBLISH-SUBSCRIBE |
| Subscription Keys | planID : (MAL::Identifier)  precursor : (MAL::Identifier)  status : (MAL::UShort)  originator : (MAL::Identifier) |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| OUT | PUBLISH/NOTIFY | No | plan : (Plan) |

#### Requirements

The monitorPlan subscription shall be based on the provision of the following keys in addition to the domain of the required Plans in the Register message, all of which are nullable:

1. PlanID: key of subscribed Plan as MAL::Identifier;
2. Precursor: key of precursorPlan as MAL::Identifier;
3. Status: status as MAL::UShort [PlanStatusEnum];
4. Originator: originator of subscribed Plan as MAL::Identifier.

NOTE – For Status, the enumerated value associated with the PlanStatusEnum must be used, and not the associated string.

#### Errors

This operation returns no errors in addition to the standard MAL errors.

### Operation: queryPlan

#### Overview

The queryPlan operation enables a consumer to retrieve a filtered set of plans, based on an extended set of filter criteria, including relevant fields of the plan information sections of the plan, as well as the type of planning activities and planning events contained within the plan.

#### Definition

| Operation Identifier | queryPlan |
| --- | --- |
| Interaction Pattern | PROGRESS |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | PROGRESS | No | query : (PlanQuery) |
| OUT | ACK | No | Empty |
| OUT | UPDATE | No | queriedPlan : (Plan) |
| OUT | RESPONSE | No | Empty |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

### Operation: getPartialPlan

#### Overview

The getPartialPlan operation enables a consumer to extract a subset of a Plan that meets the supplied partialPlanFilter. The filter can select the partial plan content based on:

1. a shorter period than that covered by the plan, specified by time, position, or events;
2. a subset of contained ActivityInstances, based on their domain, associated SubPlan or tags.

The PartialPlan returned includes the filter criteria and a version of the plan containing only the ActivityInstances that match those criteria. It is implementation dependent what is returned in terms of events and resources, but it may be assumed that any related events and resources would be included in the returned partial plan.

#### Definition

| Operation Identifier | getPartialPlan |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No | partialPlanFilter : (PartialPlanFilter) |
| OUT | RESPONSE | No | partialPlan : (PartialPlan) |

#### Requirements

The PartialPlan returned shall include all ActivityInstances contained in the source Plan that pass the specified PartialPlanFilter criteria.

The PartialPlan returned should include all EventInstances and ResourceProfiles in the source Plan that lie within the specified period for the partial plan and which are related to the included ActivityInstances.

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

## Service: Plan Execution Control Service

### Overview

The Plan Execution Control Service, introduced in 2.5.4, is provided by a plan execution function and enables its consumers to submit (and revoke) Plans for execution; to control their execution at plan, sub-plan, and activity levels; and to receive feedback on their execution status. It comprises the operations defined below, of which only those in capability set 1 are mandatory.

### Definition

| Area Identifier | Service Identifier | Area Number | Service Number | Area Version |
| --- | --- | --- | --- | --- |
| MPS | PlanExecutionControl | 5 | 3 | 1 |

| Interaction Pattern | Operation Identifier | Operation Number | Capability Set |
| --- | --- | --- | --- |
| SUBMIT | submitPlan | 1 | 1 |
| SUBMIT | revokePlan | 2 |
| REQUEST | getPlanStatus | 3 |
| REQUEST | activatePlan | 4 | 2 |
| REQUEST | deactivatePlan | 5 |
| PUBSUB | monitorPlanExecution | 6 | 3 |
| PUBSUB | monitorPlanExecutionDetail | 7 | 4 |
| REQUEST | activateSubPlan | 8 | 5 |
| REQUEST | deactivateSubPlan | 9 |
| REQUEST | getSubPlanStatus | 10 |
| PUBSUB | monitorSubPlanExecution | 11 | 6 |
| REQUEST | suspendActivity | 12 | 7 |
| REQUEST | resumeActivity | 13 |
| REQUEST | getActivityStatus | 14 | 8 |

NOTE – The getPlanStatus, with the exception of the service number, is identical to the operation of the same name in the Plan Distribution Service.

### Discussion

#### General

##### Plan Level Execution Control Operations

Four of the service operations are concerned with submission and subsequent management of plans, submitPlan, and revokePlan being mandatory operations in capability set 1.

The submitPlan operation delivers a Plan to the service provider, making it available for execution. The revokePlan operation makes a previously submitted Plan unavailable for execution. Typically the service provider deletes any local copy of the Plan.

Execution of a Plan is normally a two-stage operation: first the Plan is submitted, then once it has been received by the service provider it can be activated. If the service implementation is restricted to capability set 1, then there is effectively no activation step before a plan is executed: either the plan is activated locally or is immediately activated on submission by the service provider. If an activation step is required, then capability set 2 should also be implemented.

The activatePlan operation enables execution of one or more Plans by the service provider, subject to the triggering constraints (time, position, or event-based) associated with contained planning activities in each plan.

It is only possible to activate a Plan within its stated validity period and providing the specified start of the Plan is in the future.

The deactivatePlan operation disables the execution of the specified Plans, specifying a deactivation mode for the case where execution has already started.

If a Plan is deactivated before the start of the Plan, then it remains available to the service provider, its status returning to ‘submitted’. It can be re-activated using the activatePlan operation. Any ActivityInstances and EventInstances belonging to the deactivated Plan are removed from the active plan.

If a Plan is deactivated after the start of the Plan, then its status and that of all contained ActivityInstances and EventInstances that have not completed transition to the ‘terminated’ state with the status information ‘Cancelled’.

Three service operations provide feedback on the execution of Plans, of these only the getPlanStatus operation is mandatory in capability set 1. In order to ensure provision of feedback on the execution of a plan at a sufficiently detailed level to enable feedback for planning requests, it is recommended that capability sets 2 and 3 are implemented.

The getPlanStatus operation returns the current status of the referenced plan(s) as PlanUpdates. It is equivalent to the service operation of the same name in the Plan Distribution Service, but only reports those plan statuses relevant to the plan execution function.

Where supported, the monitorPlanExecution operation enables the consumer to subscribe to automatic notification of changes in status of plans as PlanUpdates. It is equivalent to the monitorPlanStatus operation of the Plan Distribution Service.

The monitorPlanExecutionDetail operation enables the consumer to subscribe to automatic notification of changes in status of the ActivityInstances, EventInstances, and Resources contained within Plans, provided as ActivityUpdates, EventUpdates, and ResourceUpdates respectively. It should be noted that as resource profiles are an optional element of a Plan, ResourceUpdates are only provided where they are supported.

##### SubPlan Level Execution Control Operations

Support for SubPlans is optional, but where supported an additional 4 service operations enable control at the level of SubPlans.

Planning ActivityInstances contained within a Plan may be associated with a single SubPlan ID. This can be used as a mechanism to identify multiple sub-plans within plans, for example to support individual spacecraft within a constellation, or separate payloads. SubPlan IDs are not unique to an individual Plan, but apply globally to a mission planning system. SubPlan operations apply to all activated Plans managed by the service provider. Individual ActivityInstances are only executed if both their container Plan and associated SubPlan are in the ACTIVATED state.

The activateSubPlan operation enables execution of one or more SubPlans by the service provider. It is implementation dependent whether it is initially assumed that all SubPlans are active, in which case the operation would only be needed to re-enable execution following a deactivateSubPlan operation.

The deactivateSubPlan operation disables execution of one or more SubPlans by the service provider.

The getSubPlanStatus operation returns the current status of the referenced SubPlans as SubPlanUpdates.

Where supported, the monitorSubPlanExecution operation enables the consumer to subscribe to automatic notification of changes in status of SubPlans as SubPlanUpdates.

##### Activity Level Execution Control Operations

Optional support for execution control at the level of ActivityInstances is provided by an additional 3 service operations.

Selected ActivityInstances are identified by Plan, instance ID, and/or tag. The use of tags[[1]](#footnote-2) allows arbitrary (and overlapping) groups of ActivityInstances to be managed together.

The suspendActivity operation requests suspension of the execution of selected activities, in one or more plans, without changing the state of those plans. The response gives the resulting ActivitySuspensionStatus of each referenced ActivityInstance.

The resumeActivity operation requests the resumption of previously suspended activities in one or more plans without changing the state of those plans. The response gives the resulting ActivitySuspensionStatus of each referenced ActivityInstance.

The getActivityStatus operation returns the status of ActivityInstances, as ActivityUpdates, at activity, sub-plan, or tag level.

#### MPS Service Objects

MPS service objects relevant to the plan execution control service and their relationships are defined within the MPS information model in 4.5.

The following MO objects are directly applicable to the service:

* Plan;
* ActivityInstance;
* EventInstance;
* Resource.

The identity of Plans comprises both key and version. Plans are submitted to a plan execution function for execution.

Plans contain ActivityInstances and EventInstances, the same instance of which may occur in multiple overlapping Plans or versions of Plans. Plans may also optionally contain ResourceProfiles referencing Resources.

The plan execution control service does not result in the creation of new MO objects.

The following MO objects may be referenced by the ActivityInstances and EventInstances contained within a Plan:

* ActivityDefinition;
* EventDefinition;
* RequestInstance.

### High-Level Requirements

The following set of mission planning configuration data shall be available to both provider and consumers in a any deployment of the plan execution control service:

1. Planning Activity Definitions (as ActivityDefinition objects [4.5.2.1]);
2. Planning Event Definitions (as EventDefinition objects [4.5.3.1]);
3. Planning Resource Definitions (as Resource objects [4.5.4.2]) [Optional];

### Functional Requirements

In response to a submitPlan operation, the service provider shall make the submitted Plan available for execution.

If the service does not support plan activation (capability set 2), the service provider shall provide an internal mechanism to activate submitted Plans.

NOTE – This may be as simple as to automatically activate Plans on submission.

In response to a revokePlan operation, the service provider shall make the submitted Plan unavailable for execution. If this is not possible (for example, if the plan has already executed), then a REVOKE\_FAILED error shall be returned.

In response to internal plan execution processes, the service provider shall model the status of Plans in accordance with the plan state model (see 4.5.6.2).

In response to an activatePlan operation, or internal plan activation where not supported, the service provider shall enable the execution of the referenced Plans and the ActivityInstances contained within them, subject to the triggering constraints specified within the Plans.

In response to an activatePlan operation, if a referenced Plan overlaps with its already activated precursor Plan, then the service provider should merge plan revisions into the currently active Plan.

NOTE – It is implementation dependent whether overlapping Plans are supported and how this is managed internally by the plan execution function. However, it is common for successive iterations of a Plan to overlap, for example with a Plan being generated daily for the coming week. In this case, the same ActivityInstance may appear in the overlap period of both Plans, and may be unchanged, modified or deleted in the successor Plan. New ActivityInstances may also be inserted in the overlap period or in the extended period covered by the successor Plan. It is anticipated that when the successor Plan is activated, then its predecessor is superseded from the start point of the successor Plan with any revisions applicable to the overlap period being applied. Only the latest revision of the ActivityInstance is executed.

In response to an activatePlan operation, if a referenced Plan is a target or Patch Plan, then the service provider shall also activate the associated precursor Plan.

NOTES

1. It is implementation dependent whether the target Plan is reconstituted prior to activation, or the precursor is activated and the Patch Plan merged into this.
2. If the precursor Plan is already activated, then the Patch Plan can be merged into the currently active Plan. If the precursor Plan is not available, then the operation cannot be performed.

It shall not be possible to activate a Plan if it is outside the validity period of the Plan, or if the start of the Plan period has already passed. In this case, the activatePlan operation shall fail by returning error ACTIVATE\_FAILED.

In response to a deactivatePlan operation, if supported, the service provider shall disable the execution of the referenced Plans and the ActivityInstances contained within them, subject to the specified deactivationMode.

NOTE – Supported deactivationModes are specific to the implementation of the service provider.

If Plan version 0 is specified in the getPlanStatus, monitorPlanExecution, and monitorPlanExecutionDetail operations, then the service provider shall assume the latest version of the Plan is required.

If the monitorPlanExecutionDetail operation (or activity level operations) are supported, then in response to internal plan execution processes, the service provider shall model the status of ActivityInstances and EventInstances in accordance with their respective state models (4.5.2.2 and 4.5.3.2).

If the monitorPlanExecutionDetail operation is supported, the service provider may model the evolving value of planning Resources in accordance with the ResourceProfiles (4.5.4.4) defined within Plans and any Effects (4.6.6.4.7.3) associated with ActivityInstances.

If SubPlans are supported, the service provider shall maintain the current status of SubPlans as ACTIVATED or DEACTIVATED.

NOTE – The status of SubPlans is independent to that of Plans and a change in their activation status has no impact on the status of the Plan. It may impact the status of individual ActivityInstances, which can be observed using the monitorPlanExecutionDetail operation.

In response to an activateSubPlan operation, if supported, the service provider shall activate the referenced SubPlans and enable the execution of ActivityInstances that are contained in activated Plans and allocated to activated SubPlans.

NOTE – It is implementation dependent whether SubPlans are initially ACTIVATED and therefore do not require activation unless previously deactivated.

In response to a deactivateSubPlan operation, if supported, the service provider shall deactivate the referenced SubPlans and disable the execution of ActivityInstances that are contained in activated Plans and allocated to deactivated SubPlans, subject to the specified deactivationMode.

NOTE – Supported deactivationModes are specific to the implementation of the service provider.

In response to a suspendActivity operation, if supported, the service provider shall suspend the execution of referenced ActivityInstances, subject to the specified suspensionMode.

NOTE – Supported suspensionModes are specific to the implementation of the service provider.

In response to a resumeActivity operation, if supported, the service provider shall resume the execution of the referenced ActivityInstances where it is safe to do so.

### Operation: submitPlan

#### Overview

The submitPlan operation is used to send a plan to a plan execution function (the service provider), making it available for execution. The service provider acknowledges the reception of the plan or returns an error.

NOTE – The submitted plan may be a full plan or a patch plan.

#### Definition

| Operation Identifier | submitPlan |
| --- | --- |
| Interaction Pattern | SUBMIT |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | SUBMIT | No | plan : (Plan) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| SUBMIT\_FAILED | MPS |
| UNSUPPORTED | MPS |

### Operation: revokePlan

#### Overview

The revokePlan operation is used to request a plan execution function to revoke a previously submitted Plan, making it unavailable for execution. The service provider acknowledges the revocation of the Plan or returns an error.

#### Definition

| Operation Identifier | revokePlan |
| --- | --- |
| Interaction Pattern | SUBMIT |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | SUBMIT | No | planRef : (MAL::ObjectRef <Plan>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| REVOKE\_FAILED | MPS |

### Operation: getPlanStatus

#### Overview

The getPlanStatus operation is used to obtain the current status of one or more known Plans that have been previously submitted to a plan execution function.

#### Definition

| Operation Identifier | getPlanStatus |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No | planRefs : (List <MAL::ObjectRef <Plan>>) |
| OUT | RESPONSE | No | planStatus : (List <PlanUpdate>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

### Operation: activatePlan

#### Overview

The activatePlan operation is used to request the execution of specified Plans that have previously been submitted to a plan execution function. The service provider enables the execution of the referenced Plans and the ActivityInstances contained within them, subject to the triggering constraints specified within the Plans. It is not possible to activate a Plan outside its validity period, or after the start of the Plan period. In this case, the operation will return an ACTIVATE\_FAILED error.

NOTES

1. Multiple plans with a common precursor may have been submitted to a plan execution function. Usually only one of these is considered the nominal plan, the other alternative or contingency plans having the isAlternate flag set. It is implementation dependent whether the service provider will allow activation of Plans that have the isAlternate flag set, but this may be blocked for operational safety. Where this is the case, the plan edit service can be used to change the state of the isAlternate flag prior to activation (see 3.9.5).
2. In order to activate a patch Plan, the precursor Plan on which it is based must also be activated. It is recommended that the activatePlan operation references the target Plan (the result of merging the patch Plan with its precursor), rather than the patch Plan itself (although this is allowed). It is implementation dependent how it is achieved (merge patch with precursor prior to activation, or activate precursor and then merge patch), but if the precursor Plan is not already activated, then activating a target or patch Plan implies that the precursor is also activated. If the precursor plan has not previously been submitted to the service provider (or has been revoked), then it is not possible to activate the target or patch Plan.

#### Definition

| Operation Identifier | activatePlan |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No | planRefs : (List <MAL::ObjectRef <Plan>>) |
| OUT | RESPONSE | No | activationStatus : (List <PlanActivationStatus>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| ACTIVATE\_FAILED | MPS |

### Operation: deactivatePlan

#### Overview

The deactivatePlan operation is used to request deactivation of specified Plans that have previously been activated. The service provider disables the execution of the referenced Plans and the ActivityInstances contained within them, where it is possible to do so.

The deactivationMode argument allows selection of the deactivation behaviour. For example:

* Orderly (ceases execution of any new activities, but allows those already initiated to complete);
* Rapid (ceases execution of the Plan, but allows activities already initiated to continue until their next defined breakpoint);
* Immediate (ceases execution of the Plan and all activities currently in progress).

It should be noted that it is dependent on the service provider implementation which deactivationModes are supported, and that the above list is not exhaustive.

The service provider returns a list of PlanActivationStatus data structures comprising Plan status and activationInfo as a String for each Plan in the deactivation list. The activationInfo allows the return of deployment specific details on the deactivation, such as the deactivation mode applied or reasons for a failure to deactivate.

If a Plan is deactivated prior to any of its constituent ActivityInstances being executed (or before the specified planPeriodStart), then all new ActivityInstances and EventInstances contained in the Plan are unloaded or removed, and the status of the Plan reverts to SUBMITTED.

If a Plan is deactivated after any of its constituent ActivityInstances have been executed (or after the specified planPeriodStart), then the status of the Plan and the status of all contained ActivityInstances and EventInstances that will not be executed are set to TERMINATED with the additional statusInfo ‘CANCELLED’.

#### Definition

| Operation Identifier | deactivatePlan |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No  No | planRefs : (List <MAL::ObjectRef <Plan>>)  deactivationMode : (MAL::Identifier) |
| OUT | RESPONSE | No | activationStatus : (List <PlanActivationStatus>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| DEACTIVATE\_FAILED | MPS |

### Operation: monitorPlanExecution

#### Overview

The monitorPlanExecution operation is used to subscribe to status updates for a filtered set of Plans that have been submitted to a plan execution function. The operation uses the Publish-Subscribe interaction pattern, with the body of the notification message comprising a PlanUpdate for a subscribed Plan.

The operation is equivalent to the monitorPlanStatus operation of the Plan Distribution Service, but only reports the status of plans currently being managed by a plan execution function.

#### Definition

| Operation Identifier | monitorPlanExecution |
| --- | --- |
| Interaction Pattern | PUBLISH-SUBSCRIBE |
| Subscription Keys | planID : (MAL::Identifier)  precursor : (MAL::Identifier)  status : (MAL::UShort)  originator : (MAL::Identifier) |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| OUT | PUBLISH/NOTIFY | No | planUpdate : (PlanUpdate) |

#### Requirements

The monitorPlanExecution subscription shall be based on the provision of the following keys in addition to the *domain* of the required Plans in the Register message, all of which are nullable:

1. PlanID: key of subscribed Plan as MAL::Identifier;
2. Precursor: key of precursorPlan as MAL::Identifier;
3. Status: status as MAL::UShort [PlanStatusEnum];
4. Originator: originator of subscribed Plan as MAL::Identifier.

NOTE – For Status, the enumerated value associated with the PlanStatusEnum must be used, and not the associated string.

#### Errors

This operation returns no errors in addition to the standard MAL errors.

### Operation: monitorPlanExecutionDetail

#### Overview

The monitorPlanExecutionDetail operation is used to subscribe to updates that report changes in the detailed execution status for a filtered set of Plan contents at the level of planning activities, events and resources. A planning function requires feedback at the level of planning activities and events to be able to reconstitute the status of planning requests, as well to support re-planning. The operation uses the Publish-Subscribe interaction pattern.

It is implementation dependent which details are reported on, but this may be any combination of planning activities, events, and resources. The notification message body comprises a single structure of the abstract class *PlanDetailUpdate*, which corresponds to one of the concrete classes ActivityUpdate, EventUpdate, or ResourceUpdate.

#### Definition

| Operation Identifier | monitorPlanExecutionDetail |
| --- | --- |
| Interaction Pattern | PUBLISH-SUBSCRIBE |
| Subscription Keys | planID : (MAL::Identifier)  subPlan : (MAL::Identifier)  tag : (MAL::String)  type : (MAL::Identifier) |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| OUT | PUBLISH/NOTIFY | No | detailUpdate : (PlanDetailUpdate) |

NOTE – Due to limitations of the MAL broker (see reference [2]), a *PlanDetailUpdate* can only be published for one tag at a time. If it is desirable for a *PlanDetailUpdate* to match multiple tags, the provider implementation must publish it multiple times, once under each tag.

#### Requirements

The monitorPlanExecutionDetail subscription shall be based on the provision of the following keys in addition to the *domain* of the required Plans in the Register message, all of which are nullable:

1. PlanID: key of subscribed Plan as MAL::Identifier;
2. SubPlan: subPlan of subscribed ActivityInstances as MAL::Identifier;
3. Tag: tag of subscribed ActivityInstances as MAL::String;
4. Type: type of the MO object to which the *PlanDetailUpdate* relates as MAL::Identifier. This must be one of ActivityInstance, EventInstance, or Resource.

The service provider shall publish a combination of ActivityUpdates, EventUpdates, and ResourceUpdates.

NOTE – It is implementation dependent which types of update are supported by the service provider.

If the subscription includes subPlan or tag keys, then the consumer shall only be notified of updates to ActivityInstances associated with the specified subPlan or tag. If the service provider supports EventUpdates and/or ResourceUpdates then these should be notified irrespective of subscription by subPlan or tag.

If the subscription includes Type keys, then the consumer shall only be notified with updates to the specified MO object types (ActivityInstance, EventInstance, or Resource).

#### Errors

This operation returns no errors in addition to the standard MAL errors.

### Operation: activateSubPlan

#### Overview

The activateSubPlan operation is used to request that the service provider activates the referenced SubPlans and enables the execution of ActivityInstances that are contained in activated Plans and allocated to activated SubPlans.

NOTE

1. It is implementation dependent whether SubPlans are initially ACTIVATED and therefore do not require activation unless previously deactivated.
2. Where the operation is directly supported by the service provider there is little reason for the activation to fail, but if the operation is delegated, for example to an on-board planning function, there is the potential for the operation to fail.

#### Definition

| Operation Identifier | activateSubPlan |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No | subPlanIDs : (List <MAL::Identifier>) |
| OUT | RESPONSE | No | activationStatus : (List <SubPlanActivationStatus>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| ACTIVATE\_SUBPLAN\_FAILED | MPS |

### Operation: deactivateSubPlan

#### Overview

The deactivateSubPlan operation is used to request that the service provider deactivates the referenced SubPlans and disables the execution of ActivityInstances that are contained in activated Plans and allocated to the deactivated SubPlans, where it is possible to do so.

The deactivationMode argument allows selection of the deactivation behaviour. For example:

* Orderly (ceases execution of any new activities, but allows those already initiated to complete);
* Rapid (ceases execution of the Sub-plan, but allows activities already initiated to continue until their next defined breakpoint);
* Immediate (ceases execution of the Sub-plan and all activities currently in progress).

It should be noted that it is dependent on the service provider implementation which deactivationModes are supported, and that the above list is not exhaustive.

The service provider returns a list of SubPlanActivationStatus data structures comprising sub-plan status and activationInfo as a String for each sub-plan in the deactivation list. The activationInfo allows the return of deployment specific details on the deactivation, such as the deactivation mode applied or reasons for a failure to deactivate.

NOTE – Where the operation is directly supported by the service provider there is little reason for the deactivation to fail, but if the operation is delegated, for example to an on-board planning function, there is the potential for the operation to fail.

#### Definition

| Operation Identifier | deactivateSubPlan |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No  No | subPlanIDs : (List <MAL::Identifier>)  deactivationMode : (MAL::String) |
| OUT | RESPONSE | No | activationStatus : (List <SubPlanActivationStatus>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| DEACTIVATE\_SUBPLAN\_FAILED | MPS |

### Operation: getSubPlanStatus

#### Overview

The getSubPlanStatus operation is used to obtain the current status of one or more SubPlans.

#### Definition

| Operation Identifier | getSubPlanStatus |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No | subPlanIDs : (List <MAL::Identifier>) |
| OUT | RESPONSE | No | subPlanStatus : (List <SubPlanUpdate>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

### Operation: monitorSubPlanExecution

#### Overview

The monitorSubPlanExecution operation is used to subscribe to status updates for a filtered set of SubPlans. The operation uses the Publish-Subscribe interaction pattern, with the body of the notification message comprising a SubPlanUpdate for a subscribed sub-plan.

#### Definition

| Operation Identifier | monitorSubPlanExecution |
| --- | --- |
| Interaction Pattern | PUBLISH-SUBSCRIBE |
| Subscription Keys | subPlan : (MAL::Identifier) |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| OUT | PUBLISH/NOTIFY | No | subPlanUpdate : (SubPlanUpdate) |

#### Requirements

The monitorSubPlanExecution subscription shall be based on the provision of the following keys in addition to the *domain* of the required SubPlans in the Register message, all of which are nullable:

SubPlan: subPlan ID as MAL::Identifier

#### Errors

This operation returns no errors in addition to the standard MAL errors.

### Operation: suspendActivity

#### Overview

The suspendActivity operation is used to request suspension of the execution of selected activities in one or more plans, without changing the state of the plan(s).

The suspensionMode argument allows selection of the suspension behaviour. For example:

* Orderly (suspends execution of any new activities, but allows those already initiated to complete);
* Rapid (suspends execution of any new activities, but allows any activities and their sub-activities already initiated to continue until their next defined breakpoint);
* Immediate (suspends execution of all activities, including those currently in progress).

It should be noted that it is dependent on the service provider implementation which deactivationModes are supported, and that the above list is not exhaustive.

The service provider responds with a list of ActivitySuspensionStatus data structures comprising activity status and suspensionInfo (as a String) for each activity subject to the suspension request.

The suspensionInfo allows the return of deployment specific details on the suspension, such as the suspension mode applied or reasons for a failure to suspend.

#### Definition

| Operation Identifier | suspendActivity |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | Yes  Yes  Yes  No | planRefs : (List <MAL::ObjectRef <Plan>>)  activityRefs : (List <MAL::ObjectRef <ActivityInstance>>)  tags : (List <MAL::String>)  suspensionMode : (MAL::String) |
| OUT | RESPONSE | No | suspensionStatus : (List <ActivitySuspensionStatus>) |

#### Requirements

The ActivityInstances to be suspended shall be specified in the request message of the suspendActivity operation, by the following filter fields, each of which is nullable:

1. planRefs;
2. activityRefs;
3. tags (associated with ActivityInstances to be suspended).

NOTE – As for other filters, the planRefs, activityRefs, and tags filters are ANDed together, but multiple items within each filter are ORed. This also applies to the tag filter, where if an ActivityInstance has any one of the listed tag values, it passes the filter. activityRefs and tags filters would not normally be used in conjunction.

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

### Operation: resumeActivity

#### Overview

The resumeActivity operation is used to request resumption of the execution of selected activities in one or more plans, without changing the state of the plan(s).

The service provider responds with a list of ActivitySuspensionStatus data structures comprising activity status and suspensionInfo (as a String) for each activity subject to the resumption request.

The suspensionInfo allows the return of deployment specific details on the resumption, such as the reasons for a failure to resume.

#### Definition

| Operation Identifier | resumeActivity |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | Yes  Yes  Yes | planRefs : (List <MAL::ObjectRef <Plan>>)  activityRefs : (List <MAL::ObjectRef <ActivityInstance>>)  tags : (List <MAL::String>) |
| OUT | RESPONSE | No | suspensionStatus : (List <ActivitySuspensionStatus>) |

#### Requirements

The ActivityInstances to be resumed shall be specified in the request message of the resumeActivity operation, by the following filter fields, each of which is nullable:

1. planRefs;
2. activityRefs;
3. tags (associated with ActivityInstances to be resumed).

NOTE – As for other filters, the planRefs, activityRefs, and tags filters are ANDed together, but multiple items within each filter are ORed. This also applies to the tag filter, where if an ActivityInstance has any one of the listed tag values, it passes the filter. activityRefs and tags filters would not normally be used in conjunction.

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

### Operation: getActivityStatus

#### Overview

The getActivityStatus operation is used to request a detailed report from the service provider on the current status of ActivityInstances, selected at activity, sub-plan, or tag levels.

#### Definition

| Operation Identifier | getActivityStatus |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | Yes  Yes  Yes  Yes | planRefs : (List <MAL::ObjectRef <Plan>>)  activityRefs : (List <MAL::ObjectRef <ActivityInstance>>)  subPlans : (List <MAL::Identifier>)  tags : (List <MAL::String>) |
| OUT | RESPONSE | No | activityStatus : (List <ActivityUpdate>) |

#### Requirements

The ActivityInstances whose status is to be reported shall be specified in the request message of the getActivityStatus operation, by the following filter fields, each of which is nullable:

1. planRefs;
2. activityRefs;
3. subPlans (associated with ActivityInstances to be reported);
4. tags (associated with ActivityInstances to be reported).

NOTES

1. It is implementation dependent what is reported if none of the above are provided (all ActivityInstances or none).
2. As for other filters, the planRefs, activityRefs, subPlans, and tags filters are ANDed together, but multiple items within each filter are ORed. This also applies to the tag filter, where if an ActivityInstance has any one of the listed tag values, it passes the filter. activityRefs would not normally be used in conjunction with subPlans or tags filters.

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

## Service: Plan Information Management Service

### Overview

The Plan Information Management Service, introduced in 2.5.5, is provided by a planning function and enables its consumers to list and retrieve available MPS configuration data. This includes definitions for planning requests, planning activities, planning events and planning resources, and also MPS system configuration data. The service may also be provided by a plan execution function (excluding planning request definitions). It comprises the operations defined below, none of which are mandatory.

### Definition

| Area Identifier | Service Identifier | Area Number | Service Number | Area Version |
| --- | --- | --- | --- | --- |
| MPS | PlanInformationManagement | 5 | 4 | 1 |

| Interaction Pattern | Operation Identifier | Operation Number | Capability Set |
| --- | --- | --- | --- |
| PROGRESS | listRequestDefs | 1 | 1 |
| REQUEST | getRequestDefs | 2 |
| PROGRESS | listEventDefs | 3 | 2 |
| REQUEST | getEventDefs | 4 |
| PROGRESS | listActivityDefs | 5 | 3 |
| REQUEST | getActivityDefs | 6 |
| PROGRESS | listResourceDefs | 7 | 4 |
| REQUEST | getResourceDefs | 8 |

### Discussion

#### General

Each of the first four capability sets corresponds to a particular type of MPS service object and comprises two operations:

1. listDefs: request a list of available definitions, filtered by domain and/or key;
2. getDef: obtain the full definitions for a specified set of definition IDs.

The first enables a consumer to discover which definitions are available, and the second enables their retrieval.

#### MPS Service Objects

MPS service objects relevant to the plan information management service and their relationships are defined within the MPS information model in section 4, and specifically identified as planning configuration data in 4.5.7.

The following MO objects are applicable to the service:

* RequestDefinition;
* ActivityDefinition;
* EventDefinition;
* Resource [Definition];

Each definition has its own identity (key and version).

In the case of planning Resources, the definition can omit the value field, although it can also be used to provide a default or initial value.

### High-Level Requirements

The following set of mission planning configuration data shall be available to the provider in any deployment of the plan information management service:

1. Planning Request Definitions (as RequestDefinition objects [4.5.5.1]);
2. Planning Activity Definitions (as ActivityDefinition objects [4.5.2.1]);
3. Planning Event Definitions (as EventDefinition objects [4.5.3.1]);
4. Planning Resource Definitions (as Resource objects [4.5.4.2]) [Optional];

### Functional Requirements

If version 0 is specified for the definition to be retrieved in a list[Item]Defs operation then the service provider shall list all available versions of the definition.

If version 0 is specified for the definition to be retrieved in a get[Item]Defs operation then the service provider shall return the latest available version of the definition.

### Operation: listRequestDefs

#### Overview

The listRequestDefs operation is used to obtain a list of available RequestDefinitions (key and version) together with their descriptions. The list can be filtered by domain or restricted to specified definition IDs. All available versions are listed.

The domain field is an ordered list of identifiers representing a domain hierarchy, any node of which can use ‘\*’ as a wildcard (meaning any domain identifier at that level of the hierarchy). If a set of domains is required that cannot be represented through the use of wildcards, then the operation will need to be repeated using different domain filters.

#### Definition

| Operation Identifier | listRequestDefs |
| --- | --- |
| Interaction Pattern | PROGRESS |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | PROGRESS | Yes  Yes | domain : (List <MAL::Identifier>)  requestDefs : (List <MAL::ObjectRef <RequestDefinition>>) |
| OUT | ACK | No | Empty |
| OUT | UPDATE | No | requestDefs : (List <DefListEntry>) |
| OUT | RESPONSE | No | Empty |

#### Requirements

The RequestDefinitions to be listed shall be specified in the request message of the listRequestDefs operation, by the following fields, each of which is nullable:

1. domain;
2. requestDefs.

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

### Operation: getRequestDefs

#### Overview

The getRequestDefs operation is used to retrieve one or more available RequestDefinitions, whose identity is known to the consumer.

#### Definition

| Operation Identifier | getRequestDefs |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No | requestDefs : (List <MAL::ObjectRef <RequestDefinition>>) |
| OUT | RESPONSE | No | definitions : (List <RequestDefinition>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

### Operation: listEventDefs

#### Overview

The listEventDefs operation is used to obtain a list of available EventDefinitions (key and version) together with their descriptions. The list can be filtered by domain or restricted to specified definition IDs. All available versions are listed.

The domain field is an ordered list of identifiers representing a domain hierarchy, any node of which can use ‘\*’ as a wildcard (meaning any domain identifier at that level of the hierarchy). If a set of domains is required that cannot be represented through the use of wildcards, then the operation will need to be repeated using different domain filters.

#### Definition

| Operation Identifier | listEventDefs |
| --- | --- |
| Interaction Pattern | PROGRESS |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | PROGRESS | Yes  Yes | domain : (List <MAL::Identifier>)  eventDefs : (List <MAL::ObjectRef <EventDefinition>>) |
| OUT | ACK | No | Empty |
| OUT | UPDATE | No | eventDefs : (List <DefListEntry>) |
| OUT | RESPONSE | No | Empty |

#### Requirements

The EventDefinitions to be listed shall be specified in the request message of the listEventDefs operation, by the following fields, each of which is nullable:

1. domain;
2. eventDefs.

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

### Operation: getEventDefs

#### Overview

The getEventDefs operation is used to retrieve one or more available EventDefinitions, whose identity is known to the consumer.

#### Definition

| Operation Identifier | getEventDefs |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No | eventDefs : (List <MAL::ObjectRef <EventDefinition>>) |
| OUT | RESPONSE | No | definitions : (List <EventDefinition>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

### Operation: listActivityDefs

#### Overview

The listActivityDefs operation is used to obtain a list of available ActivityDefinitions (key and version) together with their descriptions. The list can be filtered by domain or restricted to specified definition IDs. All available versions are listed.

The domain field is an ordered list of identifiers representing a domain hierarchy, any node of which can use ‘\*’ as a wildcard (meaning any domain identifier at that level of the hierarchy). If a set of domains is required that cannot be represented through the use of wildcards, then the operation will need to be repeated using different domain filters.

#### Definition

| Operation Identifier | listActivityDefs |
| --- | --- |
| Interaction Pattern | PROGRESS |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | PROGRESS | Yes  Yes  Yes | domain : (List <MAL::Identifier>)  activityDefs : (List <MAL::ObjectRef <ActivityDefinition>>)  defaultTags : (List <MAL::String>) |
| OUT | ACK | No | Empty |
| OUT | UPDATE | No | activitytDefs : (List <DefListEntry>) |
| OUT | RESPONSE | No | Empty |

#### Requirements

The ActivityDefinitions to be listed shall be specified in the request message of the listActivityDefs operation, by the following fields, each of which is nullable:

1. domain;
2. activityDefs;
3. defaultTags (associated with ActivityDefinitions to be included).

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

### Operation: getActivityDefs

#### Overview

The getActivityDefs operation is used to retrieve one or more available ActivityDefinitions, whose identity is known to the consumer.

#### Definition

| Operation Identifier | getActivityDefs |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No | activityDefs : (List <MAL::ObjectRef <ActivityDefinition>>) |
| OUT | RESPONSE | No | definitions : (List <ActivityDefinition>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

### Operation: listResourceDefs

#### Overview

The listResourceDefs operation is used to obtain a list of available Resources (key and version) together with their descriptions. The list can be filtered by domain or restricted to data types. All available versions are listed.

The domain field is an ordered list of identifiers representing a domain hierarchy, any node of which can use ‘\*’ as a wildcard (meaning any domain identifier at that level of the hierarchy). If a set of domains is required that cannot be represented through the use of wildcards, then the operation will need to be repeated using different domain filters.

#### Definition

| Operation Identifier | listResourceDefs |
| --- | --- |
| Interaction Pattern | PROGRESS |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | PROGRESS | Yes  Yes | domain : (List <MAL::Identifier>)  dataType : (List <MAL::AttributeType>) |
| OUT | ACK | No | Empty |
| OUT | UPDATE | No | resourceDefs : (List <DefListEntry>) |
| OUT | RESPONSE | No | Empty |

#### Requirements

The Resources to be listed shall be specified in the request message of the listResourceDefs operation, by the following fields, each of which is nullable:

1. domain;
2. dataType: list of Resource data types to be included.

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

### Operation: getResourceDefs

#### Overview

The getResourceDefs operation is used to retrieve the definition of one or more available Resources, whose identity is known to the consumer.

It should be noted that this operation is designed to retrieve the resource definition and not the current value of the resource (the value field may contain a default value for the resource).

#### Definition

| Operation Identifier | getResourceDefs |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No | resources : (List <MAL::ObjectRef <Resource>>) |
| OUT | RESPONSE | No | definitions : (List <Resource>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |

## Service: Plan Edit Service

### Overview

The Plan Edit Service, introduced in 2.5.6, is provided by a plan execution function and enables its consumers to modify Plans that have already been submitted for execution. It allows an external user or function to update the status of the Plan; insert, modify, or delete its constituent ActivityInstances and EventInstances; update the value of Resources; and apply a time shift to a Plan. It comprises the operations defined below, of which only those in capability set 1 are mandatory.

In some deployments, the Plan Edit Service could also be provided by a planning function to enable users to make adjustments to their planned activities prior to submission of the plan for execution.

### Definition

| Area Identifier | Service Identifier | Area Number | Service Number | Area Version |
| --- | --- | --- | --- | --- |
| MPS | PlanEdit | 5 | 5 | 1 |

| Interaction Pattern | Operation Identifier | Operation Number | Capability Set |
| --- | --- | --- | --- |
| SUBMIT | updatePlanStatus | 1 | 1 |
| REQUEST | insertActivity | 2 | 2 |
| REQUEST | insertEvent | 3 |
| SUBMIT | deleteActivity | 4 |
| SUBMIT | deleteEvent | 5 |
| SUBMIT | updateActivity | 6 | 3 |
| SUBMIT | updateEvent | 7 |
| SUBMIT | updateResourceValue | 8 | 4 |
| SUBMIT | updateResourceProfile | 9 | 5 |
| SUBMIT | applyTimeShift | 10 | 6 |

### Discussion

#### General

The updatePlanStatus operation allows the service consumer to modify the status of a previously submitted Plan, including the isAlternate flag.

For planning activities and events, three operations are defined for each item type to insert, optionally to update, and to delete an ActivityInstance or EventInstance in a referenced Plan that has previously been submitted to a plan execution function (the service provider). The insert operation results in the creation by the service provider of a new ActivityInstance or EventInstance in the specified Plan. A reference to the new instance (key and version) is returned in the response message.

For planning resources, if supported, it is only possible to update the value of the Resource. This is supported in two ways, as two distinct operations:

* updateResourceValue: a discrete update to a specified value at a specified instant of time;
* updateResourceProfile: a revised profile covering a period of time.

The applyTimeShift operation is a special operation that allows the timings contained in an entire Plan, or selected SubPlans to be shifted by a specified offset.

#### MPS Service Objects

MPS service objects relevant to the plan edit service and their relationships are defined within the MPS information model in 4.5.

The following MO objects are directly applicable to the service:

* Plan;
* ActivityInstance;
* EventInstance;
* Resource.

The identity of Plans comprises both key and version. Plans must have been previously submitted to a plan execution function for execution (using the plan execution control service).

Plans contain ActivityInstances and EventInstances, the same instance of which may occur in multiple overlapping Plans or versions of Plans. Plans can also optionally contain ResourceProfiles referencing Resources.

The insertActivity and insertEvent operations of the plan edit service result in creation of new ActivityInstances and EventInstances respectively. Deletion of ActivityInstances and EventInstances does not necessarily result in their removal from the plan execution function, but rather their transition to a terminated state.

The following MO objects can be referenced by the ActivityInstances and EventInstances contained within a Plan and plan edit service operations:

* ActivityDefinition;
* EventDefinition.

### High-Level Requirements

The following set of mission planning configuration data shall be available to both provider and consumers in a any deployment of the plan edit service:

1. Planning Activity Definitions (as ActivityDefinition objects [4.5.2.1]);
2. Planning Event Definitions (as EventDefinition objects [4.5.3.1]);
3. Planning Resource Definitions (as Resource objects [4.5.4.2]) [Optional];
4. Functional Requirements.

In response to an updatePlanStatus operation, the service provider shall update the isAlternate flag of the Plan accordingly and apply any change to Plan status consistently with the plan state model (see 4.5.6.2).

NOTE – It is implementation dependent what the service provider does in the event of a Plan status change when the Plan has executing ActivityInstances. It is recommended that the plan execution control service is used to manage Plan deactivation in an orderly manner.

In response to an insertActivity operation, the service provider shall create a new ActivityInstance in the referenced Plan and return its identity to the consumer.

In response to an insertEvent operation, the service provider shall create a new EventInstance in the referenced Plan and return its identity to the consumer.

In response to a deleteActivity operation, the service provider shall transition the referenced ActivityInstance to the TERMINATED state.

NOTE – If the ActivityInstance is in the EXECUTING state, then it is implementation dependent what action is taken by the service provider.

In response to a deleteEvent operation, the service provider shall transition the referenced EventInstance to the TERMINATED state.

NOTE – This may also affect any ActivityInstances whose start or end trigger is linked to the deleted EventInstance as a terminated event will not result in the activity being triggered.

In response to an updateActivity operation, the service provider shall apply the changes contained in the ActivityUpdate to the referenced ActivityInstance.

In response to an updateEvent operation, the service provider shall apply the changes contained in the EventUpdate to the referenced EventInstance.

In response to an updateResourceValue operation, the service provider shall apply the changes contained in the ResourceUpdate to the referenced Resource.

In response to an updateResourceProfile operation, the service provider shall apply the supplied ResourceProfile to the referenced Resource.

In response to an applyTimeShift operation, the service provider shall apply the specified time shift to the unexpired (future) content, or the specified time period of the referenced Plan or its SubPlan(s), including temporal start and end triggers on ActivityInstances.

NOTE – It is implementation dependent whether any other elements of the Plan are time shifted, particularly with regard to the eventTime of EventInstances and, where supported, the start and end times on resource ProfileSegments and the times on Profile Entries.

### Operation: updatePlanStatus

#### Overview

The updatePlanStatus operation may be used to modify the status of a previously submitted Plan. Directly modifying the status field of a Plan may be used by a third party function to autonomously terminate (or activate) a Plan, but the operation also allows the isAlternate flag to be set or cleared.

It is implementation dependent what action the service provider takes in response to a change of Plan status. The service provider may not permit certain state changes (for example to modify the status of a TERMINATED plan, which is inconsistent with the plan status model), in which case an UPDATE\_FAILED error shall be returned.

A set of Plans with a common precursor may be submitted to a plan execution function to cater for alternative or contingency scenarios. All but one of these Plans should have the isAlternate flag set, to inform the plan execution function (and the mission operations team) which is the nominal Plan. It is implementation dependent whether a plan execution control service provider will allow a Plan to be activated with the isAlternate flag set, but for operational safety reasons this may be blocked. In a contingency scenario, the updatePlanStatus operation can be used to set the flag on the nominal Plan, and reset the flag on the required contingency Plan, making it operational.

#### Definition

| Operation Identifier | updatePlanStatus |
| --- | --- |
| Interaction Pattern | SUBMIT |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | SUBMIT | No  No  No | planRef : (MAL::ObjectRef <Plan>)  status : (PlanStatusEnum)  isAlternate : (MAL::Boolean) |

#### Requirements

The status and isAlternate fields of the updatePlanStatus submit message shall both be nullable, but one must be present.

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| UPDATE\_FAILED | MPS |

### Operation: insertActivity

#### Overview

The insertActivity operation sends an InsertedActivityDetails structure (an ActivityDetails structure with Plan reference and start/end triggers) to the provider, which then creates a corresponding ActivityInstance object in the referenced Plan and returns its identity to the consumer. It is up to the planning system, how to manage concurrent access to the plan.

Insertion may fail if the Plan is already in the TERMINATED state, in which case an INSERT\_FAILED error shall be returned.

#### Definition

| Operation Identifier | insertActivity |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No | activityDetails : (InsertedActivityDetails) |
| OUT | RESPONSE | No | activityRef : (MAL::ObjectRef <ActivityInstance>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| UNSUPPORTED | MPS |
| INSERT\_FAILED | MPS |

### Operation: insertEvent

#### Overview

The insertEvent operation sends an InsertedEventDetails structure, which includes a Plan reference, to the provider, which then creates a corresponding EventInstance object in the referenced Plan and returns its identity to the consumer. It is up to the planning system, how to manage concurrent access to the plan.

Insertion may fail if the Plan is already in the TERMINATED state, in which case an INSERT\_FAILED error shall be returned.

#### Definition

| Operation Identifier | insertEvent |
| --- | --- |
| Interaction Pattern | REQUEST |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | REQUEST | No | eventDetails : (InsertedEventDetails) |
| OUT | RESPONSE | No | eventRef : (MAL::ObjectRef <EventInstance>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| INSERT\_FAILED | MPS |

### Operation: deleteActivity

#### Overview

The deleteActivity operation requests that a specified ActivityInstance within a Plan is deleted by the service provider. In practice, the activity is not removed, but transitioned to the TERMINATED state with deletion indicated in the statusInfo field. The ActivityInstance is not subsequently executed by the service provider, but it is implementation dependent what action is taken by the service provider if the ActivityInstance is in the EXECUTING state. It is up to the planning system, how to manage concurrent access to the plan.

Deletion may fail if the referenced Plan or ActivityInstance is already in the TERMINATED state, in which case the DELETE\_FAILED error shall be returned.

#### Definition

| Operation Identifier | deleteActivity |
| --- | --- |
| Interaction Pattern | SUBMIT |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | SUBMIT | No  No | planRef : (MAL::ObjectRef <Plan>)  activityRef : (MAL::ObjectRef <ActivityInstance>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| DELETE\_FAILED | MPS |

### Operation: deleteEvent

#### Overview

The deleteEvent operation requests that a specified EventInstance within a Plan is deleted by the service provider. In practice, the event is not removed, but transitioned to the TERMINATED state with deletion indicated in the statusInfo field. The EventInstance is not subsequently triggered by the service provider. It is up to the planning system, how to manage concurrent access to the plan.

Deletion may fail if the referenced Plan or EventInstance is already in the TERMINATED state, in which case the DELETE\_FAILED error shall be returned.

#### Definition

| Operation Identifier | deleteEvent |
| --- | --- |
| Interaction Pattern | SUBMIT |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | SUBMIT | No  No | planRef : (MAL::ObjectRef <Plan>)  eventRef : (MAL::ObjectRef <EventInstance>) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| DELETE\_FAILED | MPS |

### Operation: updateActivity

#### Overview

The updateActivity operation may be used to modify an ActivityInstance in a Plan that has already been submitted to the service provider. The consumer submits an ActivityUpdate structure which is applied by the service provider to the referenced ActivityInstance. It is up to the planning system, how to manage concurrent access to the plan.

Update may fail if the referenced Plan or ActivityInstance is already in the TERMINATED state, in which case the UPDATE\_FAILED error shall be returned.

#### Definition

| Operation Identifier | updateActivity |
| --- | --- |
| Interaction Pattern | SUBMIT |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | SUBMIT | No  No | planRef : (MAL::ObjectRef <Plan>)  activityUpdate : (ActivityUpdate) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| UPDATE\_FAILED | MPS |

### Operation: updateEvent

#### Overview

The updateEvent operation may be used to modify an EventInstance in a Plan that has already been submitted to the service provider. The consumer submits an EventUpdate structure which is applied by the service provider to the referenced EventInstance. It is up to the planning system, how to manage concurrent access to the plan.

Update may fail if the referenced Plan or EventInstance is already in the TERMINATED state, in which case the UPDATE\_FAILED error shall be returned.

#### Definition

| Operation Identifier | updateEvent |
| --- | --- |
| Interaction Pattern | SUBMIT |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | SUBMIT | No  No | planRef : (MAL::ObjectRef <Plan>)  eventUpdate : (EventUpdate) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| UPDATE\_FAILED | MPS |

### Operation: updateResourceValue

#### Overview

The updateResourceValue operation may be used to modify the value of a Resource at the specified point in time, in a Plan that has already been submitted to the service provider. The consumer submits a ResourceUpdate structure which is applied by the service provider to the referenced Resource. It is up to the planning system, how to manage concurrent access to the plan.

Update may fail if the referenced Plan is already in the TERMINATED state, in which case the UPDATE\_FAILED error shall be returned.

#### Definition

| Operation Identifier | updateResourceValue |
| --- | --- |
| Interaction Pattern | SUBMIT |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | SUBMIT | No  No | planRef : (MAL::ObjectRef <Plan>)  resourceUpdate : (ResourceUpdate) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| UNSUPPORTED | MPS |
| UPDATE\_FAILED | MPS |

### Operation: updateResourceProfile

#### Overview

The updateResourceProfile operation may be used to modify the value of a Resource over a period of time, in a Plan that has already been submitted to the service provider. The consumer submits an ResourceProfile structure which is applied by the service provider to the referenced Resource. It is up to the planning system, how to manage concurrent access to the plan.

Update may fail if the referenced Plan is already in the TERMINATED state, in which case the UPDATE\_FAILED error shall be returned.

#### Definition

| Operation Identifier | updateResourceProfile |
| --- | --- |
| Interaction Pattern | SUBMIT |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | SUBMIT | No  No | planRef : (MAL::ObjectRef <Plan>)  resourceProfile : (ResourceProfile) |

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| UNSUPPORTED | MPS |
| UPDATE\_FAILED | MPS |

### Operation: applyTimeShift

#### Overview

The applyTimeShift operation may be used to request a shift in the timing by a fixed offset of the ActivityInstances, EventInstances, and ResourceProfiles contained within a Plan that has previously been submitted to a plan execution function. The operation may also be restricted to one or more SubPlans within the referenced Plan and/or to a specified time period within the Plan. The service provider applies the time shift to the timing of ActivityInstances, EventInstances, and ResourceProfiles contained within the Plan or SubPlan(s).

The time shift may fail if the referenced Plan is already in the TERMINATED state, in which case the UPDATE\_FAILED error shall be returned.

The operation is designed to support backward compatibility[[2]](#footnote-3) with simple time-based on-board schedules, and may not be appropriate for use with plans that include event or position-based triggers and resource profiles. What is shifted within the Plan is implementation dependent, but shall include time-based start and end triggers on ActivityInstances. EventInstances may also be shifted, but it is noted that some EventInstances correspond to predicted orbital events that cannot meaningfully be shifted. Similarly, where supported, resource profiles may reflect the ActivityInstances contained within the Plan and if those are shifted, the corresponding changes in Resource value should also be shifted.

NOTE – ActivityInstances have duration which means they may overlap the start or end of the specified TimeWindow for the applicability of the time shift. It is implementation dependent how this is managed, but a reasonable assumption is that the start time of the ActivityInstances must be within the specified TimeWindow. Given the potential to introduce inconsistencies into a Plan, it must be assumed that users of this service operation understand both its operational implications and its specific implementation.

#### Definition

| Operation Identifier | applyTimeShift |
| --- | --- |
| Interaction Pattern | SUBMIT |

| Pattern Sequence | Message | Nullable | Type Signature |
| --- | --- | --- | --- |
| IN | SUBMIT | No  Yes  No  No | planRef : (MAL::ObjectRef <Plan>)  subPlans : (List <MAL::Identifier>)  timePeriod : (TimeWindow)  offset : (MAL::Duration) |

#### Requirements

The subPlans and timePeriod fields of the applyTimeShift submit message shall both be nullable.

#### Errors

In addition to the standard MAL errors, the operation may return the MPS-specific MO Errors defined below. The cases in which these errors are returned are defined in section 5.

| Error | Area |
| --- | --- |
| INVALID | MPS |
| UPDATE\_FAILED | MPS |

# MPS Information Model

## Overview

This section defines the data types (MO objects and other data classes) applicable to the Mission Planning and Scheduling Recommended Standard. An overview of the MPS information model has been given in 2.4 above.

The MPS information model has been defined in terms of the CCSDS Mission Operations (MO) framework, specifically the MO Message Abstraction Layer (MAL) (see reference [2]) and the associated set of MAL data types. This is to enable the specification of MO compliant data formats and services that reference elements of the information model.

In the case of the MPS File Formats defined in section 7, an explicit XML encoding of MAL Attribute types is used, which means the implementation dependency on the MAL is removed, although some data structures defined in the MAL are reused.

This section describes both the data actively exchanged by MPS Services as well as the data that are either referenced by or required as common configuration data by service providers and users. Mission specific configuration data (see A3.1) is excluded here.

As previously introduced in 2.4, some elements of the MPS information model are optional. This is the case for individual data elements where:

* they may not be required by the supported capability sets of supported services;
* they form an optional element of a required data structure.

Where this is the case, this is indicated in the description of each element in the body of this section.

The section is organized into the following main sections:

* Overview (this subsection);
* Optional Elements;
* Conventions;
* External Definitions;
* MPS Service Objects;
* MPS Data Types.

## Optional Elements

The elements of the MPS information model are grouped into element sets as detailed in table 4‑1 below. The Core Features are composed of the non-optional elements of the MPS information model and as such are mandatory, subject to the further caveat that only those MPS information model elements exposed by supported capability sets of MPS services need to be implemented. All other element sets are optional, shown with a grey background.

At the interface level, a deployment must support all data structures that may appear within the messages of supported service operations. However, the deployment is not required to generate any data structure from an element set it does not support. If the deployment receives a message containing a data structure from an optional element set it does not support, the service operation will return an UNSUPPORTED error.

Some optional element sets will rely on other optional element sets, specifically:

* support for Resource Constraints is dependent on support for Resources;
* support for Geometric Constraints is dependent on support for Position and Direction types.

The support for Effects is strongly linked to support for Resources and Resource Constraints, and as such Effects are included in the latter element set.

Table 4‑1 : Mandatory and Optional Elements of the Information Model

| # | Information Model Element Set | MO Objects | MPS Data Types | Constraints |
| --- | --- | --- | --- | --- |
| 1 | Core Features (mandatory) | Planning Requests Plans Planning Activities Planning Events | Base Data Types Expressions Arguments Time & Event Triggers Temporal & Event Repetitions | Constraint Expression |
| 2 | Basic Constraints |  |  | Temporal Constraints Sequential Constraint Separation Constraint |
| 3 | Plan Revisions | Patch Plans | Plan Revisions |  |
| 4 | Resources | Resources | Resource Profiles Plan Resources |  |
| 5 | Resource Constraints (requires Resources) | Effects | Resource and Argument Constraints |
| 6 | Position & Direction |  | Position & Direction Types Position, Direction and Angle Triggers Location, Pointing and Angle Repetitions |  |
| 7 | Geometric Constraints (requires Pos. & Dir.) |  | Geometric Constraints |
| 8 | Functions | Functions |  | Function Constraint |

## Conventions

### Data Structures

Each data structure (or type) definition contained in section 4 contains a table following the standard structure outlined below.

Table 4‑2 : Example Data Structure Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **<Data Structure name>** | Extends | <Parent name> | SFP | <#> |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| ­<name> | <data type> | Yes¦No | <Description> |
| ­<name> | <data type> | Yes¦No | <Description> |
| … | … | … | … |

The first row of the table specifies the name of the MPS data structure (in bold), and that of the structure it extends, which may either be a MAL data type (typically MAL::Composite) or another MPS data type. The Short Form Part (SFP) gives the number used by the MAL to identify this structure within the area.

This is followed by a list of fields that constitute the data structure. According to the MAL specification (reference [2] section 4.2.4.2), inherited fields shall be included here and shall be shown with a grey background. For brevity reasons, in this document this is only done for data structures of type MO Object, to highlight the inherited identity field.

Field data types may either be a MAL Attribute type, or another MPS data structure.

The nullable column indicates whether the field is allowed to contain a null value. A nullable field does not need to be provided by the consumer, but must be supported by the provider unless it is an optional element of the standard.

A default value may be specified in the description for a non-nullable field. This means that a value must be supplied in any service message ‘on the wire’, to avoid the need for a provider implementation to have knowledge of the default, but that in the context of a user (or Web-based) interface, the default value may be initially populated to avoid the need for the user to specify anything in the general case.

By convention data structure names start with an upper case letter. If the data structure is abstract (only used to define an inheritance hierarchy) then its name is italicized and the word ‘abstract’ is substituted in the SFP. Field names start with a lower case letter. In the context of MPS MO objects (definitions and instances), static and dynamic fields are differentiated by underlining the name of the static fields.

### Enumerations

Enumerations are also contained in section 4 and defined using tables of the following format.

Table 4‑3 : Example Enumeration Table

|  |  |  |  |
| --- | --- | --- | --- |
| Name | **<Enumeration name>** | SFP | <#> |

| Status | Value | Description |
| --- | --- | --- |
| <STATE NAME> | <#> | <Description> |
| … | … | … |

The set of allowed statuses/enumerations is listed together with their corresponding integer values and a description.

By convention the name of an enumeration ends with ‘Enum’, and the names of the statuses/enumerations are all in upper case.

NOTE – Following from their definition in reference [2], enumerations may be extended by implementations to support custom values. It is recommended to use the upper range of the supported set of enumeration values for this, to prevent conflicts with future updates to the extended enumeration.

### MO Object Numbers

The identity of MO objects requires the specification of the area and object type. In an efficient encoding this may be represented by a number, which is the same as the Short Form Part for the corresponding data structure. The following table specifies the number codes assigned for efficient encoding of the defined MO object types:

Table 4‑4 : MO Object Numbers

|  |  |  |  |
| --- | --- | --- | --- |
| Area | Area # | MO Object Type | Type # |
| MPS | 5 | ActivityDefinition | 101 |
| ActivityInstance | 102 |
| EventDefinition | 201 |
| EventInstance | 202 |
| Resource | 301 |
| RequestDefinition | 401 |
| RequestInstance | 402 |
| Plan | 501 |
| PlanningUser | 601 |

## External Definitions

### Time Systems

Time system references allow specification of the time system used for time fields within an MPS system. This may be specified in the context of a planning request or a plan or as a system-wide default.

The set of allowed time system values is specified in reference [10] annex B1 and in addition specified in the SANA registry for time systems:

<https://sanaregistry.org/r/time_systems/>

To allow for evolution, both of the set of time systems defined within this registry and through mission specific extension, time systems are not defined as an enumeration but represented as a MAL::Identifier.

Each implementation shall define a default time system (see annex A). This time system shall be used as default value whenever a nullable field in a message represents a time system.

### Reference Frames

Some sub-types of MPS Position and Direction require the specification of the coordinate system reference frame used. The set of allowed reference frame values is specified in reference [10] annex B2 and in addition specified in the SANA registries for reference frames (see also reference [10] annex E2):

<https://sanaregistry.org/r/celestial_body_reference_frames/>

<https://sanaregistry.org/r/orbit_relative_reference_frames/>

<https://sanaregistry.org/r/spacecraft_body_reference_frames/>

To allow for evolution, both of the set of standard coordinate systems defined within this registry and through mission specific extension, reference frames are not defined as an enumeration but represented as a MAL::Identifier.

### Celestial Bodies

Some sub-types of MPS Position and Direction require the specification of a celestial body.

The set of allowed celestial body values is specified with the Navigation and Ancillary Information Facility (NAIF) Integer ID Codes and in addition specified in the SANA registry for orbit centers (see also reference [10] annex E2):

<https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/FORTRAN/req/naif_ids.html> <https://sanaregistry.org/r/orbit_centers/>

To allow for evolution, both of the set of standard celestial bodies defined within these registries and through mission specific extension, celestial bodies are not defined as an enumeration but represented as a MAL::Identifier.

### Pointing Templates

The NAV Pointing Request Message (PRM) standard defines a number of pointing templates, which can be used in the specification of pointing constraints.

The set of allowed pointing templates is specified in reference [10] section 4 and in addition specified in the SANA registry for the PRM:

<https://sanaregistry.org/r/pointing_request_message/>

To allow for evolution, both of the set of pointing templates within this registry and through mission specific extension, pointing templates are not defined as an enumeration but represented as a MAL::Identifier.

### Units

Typical units of measure for commonly used unit types are specified in reference [10] annex D. In addition, mission specific unit type and units may be defined.

NOTE – Although the names of the ‘units’ fields are defined in the plural form, only the units of measure of a **single** quantity are expected here.

### Nullability

Certain message fields may be NULL. The concept of NULL and the resulting concept of nullability are defined in sections 4.2.8 and 4.2.9 of reference [2].

In the basis, the presence of NULL for a field solely indicates its absence. Alternatively, nullable fields may be interpreted as a default value. In such cases in this standard, this is indicated explicitly.

### Lists

Lists follow the definition given in section 4.2.6 of reference [2]. Lists may be of arbitrary length, including a length of zero elements, unless specified otherwise.

NOTE – The concepts of nullability and emptiness (of Lists) are orthogonal. A NULL List represents the absence of a List, whereas an empty List represents a List with no entries in the encoded message. These two representations are distinct, though in some scenarios their interpretation may be identical.

NOTE – While Lists may be nullable, their elements shall always be non-nullable.

### Object References

References to objects are expressed using the MAL::ObjectRef type, defined in section 4.5.19 of reference [2]. The convention given in reference [2] is followed that a field with type MAL::ObjectRef <T> may refer only to objects of type T.

## MPS Service Objects

### General

#### Overview

The principal MPS Service Objects defined are those introduced in 2.4.2 that correspond to a set of MO objects that can be directly referenced in the context of the MPS services:

* Planning Activities;
* Planning Events;
* Planning Resources [Optional];
* Planning Requests;
* Plans;
* Planning Users;
* Custom Functions [Optional].

The following subsections contain the definition of each MPS Service Object and are structured as follows:

* MO Objects defined for the Service Object;
* Status enumeration for the Service Object (if relevant)
* Subordinate Data Types defined for the Service Object (if relevant);
* Service-specific Data Types defined for the Service Object (these are used in the context of MPS service messages).

#### Updates to multiple Object Types

##### Data Type: PlanDetailUpdate

###### Overview

Specifically in the case of reporting the detailed execution status of a plan, updates may be reported for multiple object types: planning activities, planning events, and planning resources. To support this an abstract type of PlanDetailUpdate is defined as follows.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PlanDetailUpdate** | Extends | MAL::Composite | SFP | Abstract |

### Planning Activities

#### Activity MO Objects

##### Data Type: ActivityDefinition

###### Overview

An ActivityDefinition is an MO object that contains static configuration data relating to multiple occurrences of a planning activity. Its identity is defined by a definitionID, which includes a constant key and an evolving version that is updated each time the definition is revised. ActivityDefinitions form part of the planning configuration data.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ActivityDefinition** | Extends | MAL::Object | SFP | 101 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| identity | MAL::ObjectIdentity | No | Identity of the ActivityDefinition, including version. |
| description | MAL::String | No | Description of the Activity. |
| argDefs | List <ArgDef> | Yes | List of Argument Definitions. |
| constraints | Constraint | Yes | A single constraint or a constraint node that may contain multiple constraints, applicable to all instances of the Activity. |
| effects | List <Effect> | Yes | Set of Effects applicable to all instances of the Activity. |
| executionDefinition | MAL::Identifier | Yes | Reference to the definition of an executable body for the Activity (procedure, action sequence, etc.). The manner in which this reference is interpreted is implementation-specific. |
| durationSpec | Expression <MAL::Duration> | Yes | Supports calculation of an estimated duration of an Activity Instance. |
| children | List <ActivityDetails> | Yes | Set of activity details specifying child activities. |
| activityType | MAL::String | Yes | Free-text field that can be used to categorize an activity into one of several arbitrary categories. Enables a planning system to customize behavior for activities, such as their presentation in displays, based on the specified value. |
| defaultTags | List <MAL::String> | Yes | Default set of Tags that may be used to associate the Activity with others, grouping activities by operational responsibility (controller/group/system) or other criteria. |

##### Data Type: ActivityInstance

###### Overview

An ActivityInstance is an MO object that contains the identity of a specific occurrence of a planning activity, together with both static and dynamic information associated with that occurrence. It supports relationships to its definition, source, a related planning event and any child activities.

ActivityInstances may be contained within a Plan.

NOTE – The start and end fields specify the trigger conditions (including time) that specify when the ActivityInstance starts and/or ends in the context of a Plan. The duration is an estimate of the time taken to execute the ActivityInstance rather than an offset, which may for example be used in the visualization of a Plan. Duration may be used in conjunction with a specified end trigger to determine the planned start time of an ActivityInstance.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ActivityInstance** | Extends | MAL::Object | SFP | 102 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| identity | MAL::ObjectIdentity | No | Identity of the ActivityInstance |
| definition | MAL::ObjectRef <ActivityDefinition> | No | Reference to the ActivityDefinition. |
| source | MAL::ObjectRef <MAL::Element> | No | Object Type: RequestInstance | ActivityInstance | PlanningUser  Reference to the source of the ActivityInstance, which is either its parent ActivityInstance, a RequestInstance if it is a root Activity, or a PlanningUser if directly inserted. |
| relatedEvent | MAL::ObjectRef <EventInstance> | Yes | Optional reference to an EventInstance that is specifically associated with this instance of the Activity. Typically the Activity is placed in response to the Event. |
| children | List <MAL::ObjectRef <ActivityInstance>> | Yes | References to any child ActivityInstances. |
| comments | MAL::String | Yes | Any notes associated with this instance of the Activity. |
| constraints | Constraint | Yes | A single constraint or a constraint node that may contain multiple constraints, applicable to this instance of the Activity. |
| effects | List <Effect> | Yes | Set of Effects applicable to this instance of the Activity. |
| arguments | List <Argument> | Yes | Argument values for each Argument defined in the Activity Definition. |
| start | Trigger | Yes | Optionally specifies the trigger that initiates the Activity: may be Time, Position or Event based. |
| end | Trigger | Yes | Optionally specifies the trigger that ends the Activity. |
| duration | MAL::Duration | Yes | Optional duration of the Activity (estimated until execution, actual post execution). |
| subPlan | MAL::Identifier | Yes | Optional association of the Activity with a defined sub-plan. |
| tags | List <MAL::String> | Yes | Set of Tags that may be used to associate the Activity with others, grouping activities by operational responsibility (controller/group/system) or other criteria. |
| status | ActivityStatusEnum | No | Current Status of the Activity Instance (see Activity State Model in 4.5.2.2). |
| executionInstance | MAL::Identifier | Yes | Reference to the instance of an executable body for the Activity (procedure, action sequence, etc.). The manner in which this reference is interpreted is implementation specific. |
| returnData | List <MAL::NamedValue> | Yes | Optional return data from the planning process, provided as a list of ID-Value pairs. This can be used to provide additional information required by the User to interpret the planned activity. |
| statusInfo | MAL::String | Yes | StatusInfo provides the reason for entering the Terminated State and is customizable, but if the following conditions exist then the specified text shall be used:  - Completed (nominal);  - Expired (prior to Activation or during plan Suspension);  - Deleted;  - Failed (see ErrorCode/ErrorInfo). |
| errorCode | MAL::Integer | Yes | Error Code optional in the case of a failure status for the planning activity (for example Terminated state with statusInfo Failed). The codes are implementation specific. |
| errorInfo | MAL::String | Yes | Supplementary Error Information. |

#### Activity Status

##### Data Type: ActivityStatusEnum

###### Overview

An **ActivityStatusEnum** represents the set of states possible for an ActivityInstance.

###### Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Name | **ActivityStatusEnum** | SFP | 103 |

| Status | Value | Description |
| --- | --- | --- |
| PLANNED | 1 | The Activity Instance has been included in the Plan. |
| ACTIVATED | 2 | The Plan including the Activity Instance has been Activated within the plan execution function. |
| EXECUTING | 3 | Execution of the Activity Instance has been initiated. |
| SUSPENDED | 4 | Execution of the Activity Instance has been suspended. |
| TERMINATED | 5 | Execution of the Activity Instance has been terminated (further information is provided in statusInfo). |

#### Activities: Subordinate Data Types

##### Data Type: ActivityDetails

###### Overview

Contains the information required to create one or more ActivityInstances, including the specification of argument values and constraints.

It should be noted that the activityRef and activityOffset fields are only relevant in the case that a Repetition has been specified in a parent ActivityNode. Temporal and sequential constraints associated with the ActivityInstance can be specified as constraints attached to a concrete SimpleActivityDetails structure.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ActivityDetails** | Extends | MAL::Composite | SFP | Abstract |

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Type | Nullable | Description |
| activityRef | Slider | Yes | Specifies how the ActivityInstance is placed with respect to any defined Repetition (0=Start; 1=End).  Default is Start. |
| activityOffset | Expression <MAL::Duration> | Yes | Specifies an offset in time for the ActivityInstance from any defined Repetition.  Default is no offset. |
| relatedEvent | Expression  <MAL::ObjectRef <MAL::Element>> | Yes | Object Type: EventInstance.  Specifies a related Event (or Event Group) for the ActivityInstance. Argument specifications and constraints may reference arguments and fields of the RelatedEvent. |
| comments | MAL::String | Yes | Any notes associated with the ActivityDetails. |

##### Data Type: ActivityNode

###### Overview

A concrete sub-type of ActivityDetails, an ActivityNode is a container node for a set of ActivityDetails together with an optional Repetition specification.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ActivityNode** | Extends | ActivityDetails | SFP | 104 |

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Type | Nullable | Description |
| repetition | Repetition | Yes | Optional Repetition specification. |
| activities | List <ActivityDetails> | Yes | Set of ActivityDetails. |

##### Data Type: SimpleActivityDetails

###### Overview

A concrete sub-type of ActivityDetails, a SimpleActivityDetails provides the information required to instantiate a single ActivityInstance.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **SimpleActivityDetails** | Extends | ActivityDetails | SFP | 105 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| activityDefinition | MAL::ObjectRef <ActivityDefinition> | No | Reference to the ActivityDefinition. |
| argSpecs | List <ArgSpec> | Yes | Set of argument specifications for each argument definition contained in the referenced activity definition. These supply a value for each argument, or an expression to enable the value to be derived. |
| constraints | Constraint | Yes | A single constraint or a constraint node that may contain multiple constraints, specific to the ActivityInstance to be created. |
| effects | List <Effect> | Yes | Set of Effects specific to the ActivityInstance to be created. |
| subPlan | MAL::Identifier | Yes | Optional association of the ActivityInstance with a defined sub-plan. |
| tags | List <MAL::String> | Yes | Set of tags that may be used to associate the Activity with an identified subset of the Plan, grouping activities by operational responsibility (controller/group/system) or other criteria. |

#### Activities: Service Support Data Types

##### Data Type: ActivityUpdate

###### Overview

ActivityUpdate is a data structure that is used to report the dynamic status of an ActivityInstance in the context of the MPS Plan Execution Control service monitorPlanExecutionDetail and getActivityStatus operations.

ActivityUpdates may be distributed to subscribing applications, including status displays, to inform them of the latest status of the activity. This may be particularly relevant in conjunction with a plan execution function. ActivityUpdates may be stored in activity history to provide a complete record of evolving status over time.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ActivityUpdate** | Extends | PlanDetailUpdate | SFP | 106 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| activityInstance | MAL::ObjectRef <ActivityInstance> | No | Reference to the ActivityInstance to which the status update relates. |
| timestamp | MAL::Time | Yes | Time of status update.  Only nullable in the context of an updateActivity operation: the timestamp must be provided when reporting ActivityInstance status. |
| plan | MAL::ObjectRef <Plan> | Yes | Optional reference to the Plan containing the ActivityInstance to which this update pertains. |
| arguments | List <Argument> | Yes | Argument values. |
| start | Trigger | Yes | Optionally specifies the trigger that initiates the ActivityInstance: may be time, position, or event based. |
| end | Trigger | Yes | Optionally specifies the trigger that ends the ActivityInstance. |
| duration | MAL::Duration | Yes | Optional duration of the ActivityInstance (estimated until execution, actual post execution). |
| subPlan | MAL::Identifier | Yes | Optional association of the ActivityInstance with a defined sub-plan. |
| tags | List <MAL::String> | Yes | Set of tags that may be used to associate the ActivityInstance with an identified subset of the Plan, grouping activities by operational responsibility (controller/group/system) or other criteria. |
| status | ActivityStatusEnum | No | Current status of the ActivityInstance. |
| executionInstance | MAL::Identifier | Yes | Reference to the instance of an executable body for the ActivityInstance (procedure, action sequence, etc.). The manner in which this reference is interpreted is implementation-specific. |
| returnData | List <MAL::NamedValue> | Yes | Optional return data from the planning process, provided as a list of ID-Value pairs. This can be used to provide additional information required by the User to interpret the planned activity. |
| statusInfo | MAL::String | Yes | StatusInfo provides the reason for entering the Terminated State and is customizable, but if the following conditions exist then the specified text shall be used:  - Completed (nominal);  - Expired (prior to Activation or during plan Suspension);  - Deleted;  - Failed (see ErrorCode/ErrorInfo). |
| errorCode | MAL::Integer | Yes | Error Code optional in the case of a failure status for the planning activity (for example Terminated state with statusInfo Failed). The codes are implementation specific. |
| errorInfo | MAL::String | Yes | Supplementary error information. |

##### Data Type: InsertedActivityDetails

###### Overview

A concrete sub-type of ActivityDetails (see 4.5.2.3) that is a variation of SimpleActivityDetails providing additional details for a single ActivityInstance to be inserted into a Plan using the MPS Plan Edit service.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **InsertedActivityDetails** | Extends | ActivityDetails | SFP | 107 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| plan | MAL::ObjectRef <Plan> | No | Reference to the Plan into which the ActivityInstance is to be inserted. |
| start | Trigger | Yes | Optionally specifies the trigger that initiates the ActivityInstance: may be time, position, or event based. |
| end | Trigger | Yes | Optionally specifies the trigger that ends the ActivityInstance. |
| activityDefinition | MAL::ObjectRef <ActivityDefinition> | No | Reference to the ActivityDefinition. |
| argSpecs | List <ArgSpec> | Yes | Set of argument specifications for each argument definition contained in the referenced activity definition. These supply a value for each argument, or an expression to enable the value to be derived. |
| constraints | Constraint | Yes | A single constraint or a constraint node that may contain multiple constraints, specific to the ActivityInstance to be created. |
| effects | List <Effect> | Yes | Set of Effects specific to the ActivityInstance to be created. |
| subPlan | MAL::Identifier | Yes | Optional association of the ActivityInstance with a defined sub-plan. |
| tags | List <MAL::String> | Yes | Set of tags that may be used to associate the Activity with a subset of the Plan, grouping activities by operational responsibility (controller/group/system) or other criteria. |

##### Data Type: ActivitySuspensionStatus

###### Overview

A data structure that returns the status and supplementary suspension information for an ActivityInstance affected by an MPS Plan Execution Control service suspendActivity or resumeActivity operation.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ActivitySuspensionStatus** | Extends | MAL::Composite | SFP | 108 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| activityInstance | MAL::ObjectRef <ActivityInstance> | No | Reference to an ActivityInstance. |
| plan | MAL::ObjectRef <Plan> | Yes | Optional reference to the Plan containing the ActivityInstance. |
| status | ActivityStatusEnum | No | Current Status of the ActivityInstance. |
| suspensionInfo | MAL::String | Yes | Supplementary information on the suspension/resumption status of the ActivityInstance.  This may detail the point of suspension, which may be specific to the suspension mode; or a reason why resumption was not possible. |

### Planning Events

#### Event MO Objects

##### Data Type: EventDefinition

###### Overview

An EventDefinition is an MO object that contains static configuration data relating to multiple occurrences of a planning event. Its identity is defined by a definitionID, which includes a constant key and an evolving version, which is updated each time the definition is revised. Event definitions form part of the planning configuration data.

Events may be either Predicted or Potential. Events that are predictable either by time or position can have specific instances included in a Plan. Potential events are those that may occur during the execution of a Plan, but the specific time or position is not predicted.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **EventDefinition** | Extends | MAL::Object | SFP | 201 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| identity | MAL::ObjectIdentity | No | Identity of the EventDefinition, including version. |
| description | MAL::String | No | Description of the event. |
| predictability | PredictabilityEnum | No | Enumeration: one of {Predicted, Potential} indicating whether the event occurrence is known in advance or can occur at any time. |
| eventType | MAL::String | Yes | Free-text field that can be used to categorize an event into one of several arbitrary categories. Enables a planning system to customize behaviour for events, such as their presentation in displays, based on the specified value. |
| argDefs | List <ArgDef> | Yes | List of argument definitions. |
| eventDefinitions | List <MAL::ObjectRef <EventDefinition>> | Yes | List of child event definitions. For a single event, this list shall be empty; for a group event, the list shall be populated. |

NOTE – In practice, some planning systems support planning events that are not necessarily instantaneous. Rather, they may have a non-zero duration. This is not the case for planning events as defined in this standard; yet, such behaviour can be supported by providing a ‘duration’ argument for an EventDefinition or EventInstance.

##### Data Type: EventInstance

###### Overview

An EventInstance is an MO object that contains the identity of a specific occurrence of a planning event, together with both static and dynamic information associated with that occurrence. It supports relationships to its definition and source.

The source of an EventInstance may be an external event, corresponding to a NAV Predicted Event or a CSS Contact Event.

EventInstances may be contained within a Plan.

EventInstances may be referenced as a related event by an ActivityInstance, so that the ActivityInstance can reference the timing and arguments of the related EventInstance.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **EventInstance** | Extends | MAL::Object | SFP | 202 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| identity | MAL::ObjectIdentity | No | Identity of the EventInstance |
| definition | MAL::ObjectRef <EventDefinition> | No | Reference to the EventDefinition. |
| sourceEvent | MAL::Identifier | Yes | Reference to an external source event (e.g., NAV predicted event, or CSS contact event). |
| events | List <MAL::ObjectRef <EventInstance>> | Yes | List of references to child EventInstances. For a single event, this list is empty; for a group event, the list will be populated. |
| eventTime | MAL::FineTime | Yes | Predicted or actual time of the event. EventTime is nullable: it can be predicted without an eventTime (e.g., if position based). |
| arguments | List <Argument> | Yes | Argument values for each argument defined in the EventDefinition. |
| eventStatus | EventStatusEnum | No | Current status of the event instance (see event state model in 4.5.3.2). |
| statusInfo | MAL::String | Yes | StatusInfo provides the reason for entering the terminated state and is customizable, but if the following conditions exist then the specified text shall be used:  - Occurred (Event has been triggered);  - Did Not Occur (Event expired or did not occur within validity period);  - Deleted (Event was deleted). |

#### Event Status

##### Data Type: EventStatusEnum

###### Overview

The EventStatusEnum represents the status of a given EventInstance.

###### Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Name | **EventStatusEnum** | SFP | 203 |

| Status | Value | Description |
| --- | --- | --- |
| GROUP | 1 | The EventInstance is a group event. |
| PLANNED | 2 | The EventInstance has been included in the Plan . |
| ACTIVATED | 3 | The Plan including the EventInstance has been Activated within the plan execution function. |
| TERMINATED | 4 | The EventInstance has reached a terminal status (further information is provided in statusInfo). |

#### Events: Subordinate Data Types

##### Data Type: PredictabilityEnum

###### Overview

The PredictabilityEnum enumeration is used to indicate whether a given Event is predictable or can occur at any time.

###### Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Name | **PredictabilityEnum** | SFP | 204 |

| Enumeration | Value | Description |
| --- | --- | --- |
| PREDICTED | 1 | Events that are predictable either by time or position can have specific instances included in a Plan. |
| POTENTIAL | 2 | Potential events are those that may occur during the execution of a Plan, but the specific time or position is not predicted. |

#### Events: Service Support Data Types

##### Data Type: EventUpdate

###### Overview

EventUpdate is a data structure that is used to report the dynamic status of an EventInstance in the context of the MPS Plan Execution Control service monitorPlanExecutionDetail operation.

EventUpdates may be distributed to subscribing applications, including status displays, to inform them of the latest status of the event. This may be particularly relevant in conjunction with a plan execution function. EventUpdates may be stored in event history to provide a complete record of evolving status over time.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **EventUpdate** | Extends | PlanDetailUpdate | SFP | 205 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| eventInstance | MAL::ObjectRef <EventInstance> | No | Reference to the EventInstance to which the status update relates. |
| timestamp | MAL::Time | No | Time of status update. |
| eventTime | MAL::FineTime | No | Predicted or actual time of the event. EventTime is nullable: it can be predicted without an EventTime (e.g., if position based). |
| arguments | List <Argument> | Yes | Argument values. |
| eventStatus | EventStatusEnum | No | Current status of the EventInstance. |
| statusInfo | MAL::String | Yes | StatusInfo provides the reason for entering the Terminated state and is customizable, but if the following conditions exist then the specified text shall be used:  - Occurred (Event has been triggered);  - Did Not Occur (Event expired or did not occur within validity period);  - Deleted (Event was deleted). |

##### Data Type: InsertedEventDetails

###### Overview

A data structure that provides the information required to create the EventInstance to be inserted into a Plan using the MPS Plan Edit service.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **InsertedEventDetails** | Extends | MAL::Composite | SFP | 206 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| plan | MAL::ObjectRef <Plan> | No | Reference to the Plan into which the Event is to be inserted. |
| eventDefinition | MAL::ObjectRef <EventDefinition> | No | Reference to the EventDefinition. |
| eventTime | MAL::FineTime | No | Specifies the predicted or actual time of the event. For an inserted event this must be present. |
| arguments | List <Argument> | Yes | Argument values. |

### Planning Resources [Optional]

#### General

Planning resources are an optional element of the MPS information model. Support for planning resources is not a requirement for compliance of an MPS system with the MPS service interfaces.

#### Resource MO Objects

##### Data Type: Resource

###### Overview

A resource is an MO object that contains both the static fields that define a planning resource and a dynamic field that holds its current value. Its identity is defined by a constant key and evolving version, which is updated each time the definition is revised. Resource definitions form part of the planning configuration data and in practice the value field may be omitted in this context, although it may also be used to provide an initial or default value.

Depending on the resource data type, the resource definition may require additional type-specific fields to support data validation. Sub-types are defined for Numeric, String, and enumerated Status type resources. The base Resource MO object type can be used where no data validation is applicable. The following fields are applicable to the base type and all sub-types.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **Resource** | Extends | MAL::Object | SFP | 301 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| identity | MAL::ObjectIdentity | No | Identity of the Resource, including version of the Resource definition. |
| description | MAL::String | No | Description of the Resource. |
| dataType | MAL::AttributeType | No | Specifies the data type of the Resource, which must be a supported MAL Attribute type. |
| units | MAL::String | Yes | Optional. Specifies the units of a single quantity, in which the value of the Resource is expressed in. |
| validationData | ValidationDetails | Yes | Optional. Specifies the allowed range of values for the Resource, with concrete subtypes specific to the data type of the Resource. |
| value | MAL::Attribute | Yes | Value of the resource. MAL Attribute type must match the dataType of the Resource definition.  The value is only nullable in the context of a Resource definition (planning configuration data). |

#### Resources: Subordinate Data Types

##### Data Type: NumericResource

###### Overview

An additional concrete sub-type of ValidationDetails applicable only to Resources of any numeric type, including Duration, that provides additional fields for the specification of numeric data validation.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **NumericResource** | Extends | ValidationDetails | SFP | 302 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| minimum | ResourceProfile | No | Defines the permitted minimum value over time. |
| maximum | ResourceProfile | No | Defines the permitted maximum value over time. |

#### Resource Profile Data Types

##### Data Type: ResourceProfile

###### Overview

A ResourceProfile provides the evolution of a value for a single planning resource over time as a set of ProfileSegments.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ResourceProfile** | Extends | MAL::Composite | SFP | 303 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| resource | MAL::ObjectRef <Resource> | No | Reference to a Resource. |
| profileSegments | List <ProfileSegment> | No | Set of Profile Segments; if these segments are not contiguous, the value of the profile in those places is undefined. The resulting behaviour may be defined by the planning system. |

##### Data Type: ProfileSegment

###### Overview

A ProfileSegment defines the time range and interpolation method for a set of ProfileEntries.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ProfileSegment** | Extends | MAL::Composite | SFP | 304 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| interpolation | InterpolationTypeEnum | Yes | Interpolation method to be applied for values lying between points defined in the profile segment.  Default = Step. |
| start | Expression <MAL::Time> | No | Start of time range covered by the profile segment. |
| end | Expression <MAL::Time> | No | End of time range covered by the profile segment. |
| startIncluded | MAL::Boolean | Yes | Indicates whether the start time is included in the profile segment.  Default = True. |
| endIncluded | MAL::Boolean | Yes | Indicates whether the end time is included in the profile segment. This allows the same time to be used as the end of one segment and the start of another.  Default = False. |
| profileEntries | List <ProfileEntry> | No | Set of profile entries (resource value points). |

##### Data Type: InterpolationTypeEnum

###### Overview

The InterpolationTypeEnum describes the set of supported interpolation types for a given operation.

###### Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Name | **InterpolationTypeEnum** | SFP | 305 |

| Enumeration | Value | Description |
| --- | --- | --- |
| STEP | 1 | No interpolation: resource values remain unchanged between defined points. |
| LINEAR | 2 | Linear interpolation: resource values follow a straight line between defined points. |
| POLYNOMIAL | 3 | Polynomial interpolation: resource values follow a curve interpolating the defined points. |

##### Data Type: ProfileEntry

###### Overview

Defines the value (or minimum/maximum value) of a resource at a particular point in time.

###### Definition

The data type of the value can be any valid MAL Attribute type, but must match the defined dataType in the corresponding Resource definition.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ProfileEntry** | Extends | MAL::Composite | SFP | 306 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| time | Expression <MAL::Time> | No | Time of resource data point. |
| value | MAL::Attribute | No | Value of resource data point. |

##### Data Type: RelativeResourceProfile

###### Overview

A variation on ResourceProfile, the RelativeResourceProfile uses relative timestamps of type Duration (indicating an offset from a reference time, such as the start time of an Activity). RelativeResourceProfiles are used in the context of a complex resource constraint.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **RelativeResourceProfile** | Extends | MAL::Composite | SFP | 307 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| resource | MAL::ObjectRef <Resource> | No | Reference to a Resource. |
| profileSegments | List <RelativeProfileSegment> | No | Set of RelativeProfileSegments. |

##### Data Type: RelativeProfileSegment

###### Overview

A RelativeResourceSegment defines the time range and interpolation method for a set of RelativeProfileEntries.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **RelativeProfileSegment** | Extends | MAL::Composite | SFP | 308 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| interpolation | InterpolationTypeEnum | Yes | Interpolation method to be applied for values lying between points defined in the relative profile segment.  Default = Step. |
| start | Expression <MAL::Duration> | No | Relative start of time range covered by the relative profile segment. |
| end | Expression <MAL::Duration> | No | Relative end of time range covered by the relative profile segment. |
| startIncluded | MAL::Boolean | Yes | Indicates whether the start time is included in the relative profile segment.  Default = True. |
| endIncluded | MAL::Boolean | Yes | Indicates whether the end time is included in the relative profile segment. This allows the same time to be used as the end of one segment and the start of another.  Default = False. |
| profileEntries | List <RelativeProfileEntry> | No | Set of relative profile entries (resource value points). |

##### Data Type: RelativeProfileEntry

###### Overview

Defines the value (or minimum/maximum value) of a resource at a relative point in time.

Specific sub-types exist for each allowed data type for a Resource. These replace the variant type value field with one of concrete data type.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **RelativeProfileEntry** | Extends | MAL::Composite | SFP | 309 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| time | Expression <MAL::Duration> | No | Relative time of resource data point. |
| value | MAL::Attribute | No | Value of resource data point. |

#### Resources: Service Support Data Types

##### Data Type: ResourceUpdate

###### Overview

ResourceUpdate is a data structure that is used to report the value of a Resource at a given point in time in the context of the MPS Plan Execution Control service monitorPlanExecutionDetail operation, or to supply an updated value for a Resource in the context of the MPS Plan Edit service.

Resource updates may be distributed to subscribing applications, including status displays, to inform them of the latest value of the Resource. This may be particularly relevant in conjunction with a plan execution function. Resource updates may be stored in resource history to provide a complete record of evolving value over time.

Resource updates are also effectively contained within a Plan to describe the predicted evolution of Resources over the duration of that Plan. However, in this context the ResourceProfile construct is used (see 4.5.4.4 above).

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ResourceUpdate** | Extends | PlanDetailUpdate | SFP | 310 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| resource | MAL::ObjectRef <Resource> | No | Reference to the Resource to which the value update relates. |
| timestamp | MAL::Time | No | Time of Resource value update. |
| value | MAL::Attribute | No | Value of the resource. MAL Attribute type must match the dataType of the resource definition. |

### Planning Requests

#### Planning Request MO Objects

##### Data Type: RequestDefinition

###### Overview

A RequestDefinition is an MO object that contains the specification of a re-usable planning request template.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **RequestDefinition** | Extends | MAL::Object | SFP | 401 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| identity | MAL::ObjectIdentity | No | Identity of the RequestDefinition, including version. |
| description | MAL::String | No | Description of the re-usable RequestDefinition. |
| argDefs | List <ArgDef> | Yes | List of argument definitions. Arguments may be referenced in ActivityDetails and constraints. |
| standingOrder | MAL::Boolean | No | A flag that indicates whether the planning request is for a repetitive standing order (unbounded other than by the validity period), or is a one-off request. If it is a standing order, then the supplied **activity details** must be an ActivityNode with specification of the repetition criteria. It should be noted that a one-off request can still include repetition. |
| activities | List <ActivityDetails> | No | Set of activity details specifying requested activities. |

NOTE – Some planning systems support some notion of prioritization. That is not natively supported by the planning request concept presented in this standard, due to the multitude of ways in which prioritization can be interpreted and implemented. Instead, attention is drawn to the fact that equivalent functionality can be implemented in individual planning systems by means of arguments.

##### Data Type: RequestInstance

###### Overview

A RequestInstance is an MO object that contains the specification of a planning request. This may change over time if the request is updated by the user, each comprising a separate version of the request with the same object key.

In the context of a hierarchical or federated planning system, a RequestInstance can be used to submit a Plan (4.5.6) to a planning function, either embedding the Plan in the RequestInstance or passing it by reference. If passed by reference, the Plan can be retrieved using the Plan Distribution Service (3.6). Patch plans are not permitted in this context.

NOTE – RequestInstances may be created from a RequestDefinition that has defined arguments (as ArgDefs) and will in this case have the associated Arguments. An ad-hoc RequestInstance is not anticipated to hold any Arguments. The values that can be parameterized through the arguments of a re-usable RequestDefinition can be directly entered in a RequestInstance, and there would be no corresponding ArgDef associated with any Arguments supplied.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **RequestInstance** | Extends | MAL::Object | SFP | 402 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| identity | MAL::ObjectIdentity | No | Identity of the RequestInstance, including version. |
| requestDetails | PlanningRequestDetails | No | The contents of the planning request. |
| creationTime | MAL::Time | No | Creation date and time of the RequestInstance version. |
| status | RequestStatusEnum | No | Current status of the ActivityInstance (see planning request state model in 4.5.5.2). |
| outputPlanRefs | List <MAL::ObjectRef <Plan>> | Yes | Reference to the output Plan(s) that contains the activities resulting from the planning request. Where multiple alternate plans have been generated, these may be listed here. |
| returnData | List <MAL::NamedValue> | Yes | Optional return data from the planning process, provided as a list of ID-Value pairs. This can be used to provide additional information required by the User to interpret the planned operations. |
| statusInfo | MAL::String | Yes | StatusInfo provides the reason for termination and is customizable, but if the following conditions exist then the specified text shall be used:  - Completed (all constituent activities completed successfully);  - Expired (constituent activities expired prior to execution);  - Failed (constituent activities failed during execution);  - Deleted (constituent activities were deleted);  - Partially Completed.  It may also be used to provide the reason for rejection. |
| errorCode | MAL::Integer | Yes | Error Code optional in the case of a failure status for the planning request (for example Terminated state with statusInfo Failed). The codes are implementation specific. |
| errorInfo | MAL::String | Yes | Supplementary error information. |

#### Planning Request Status

##### Data Type: RequestStatusEnum

###### Overview

The RequestStatusEnum enumeration represents the different statuses in which a planning request may be found.

###### Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Name | **RequestStatusEnum** | SFP | 403 |

| Status | Value | Description |
| --- | --- | --- |
| REQUESTED | 1 | The planning request has been submitted to the planning function. |
| ACCEPTED | 2 | The planning request has been accepted by the planning function. |
| REJECTED | 3 | The planning request has been rejected by the planning function. |
| CANCELLED | 4 | The planning request has been cancelled by the user. |
| PLANNED | 5 | The planning request has been incorporated into a Plan (see outputPlanRefs). |
| PROCESSING | 6 | The corresponding Plan has been activated within plan execution. |
| PROCESSED | 7 | Execution of the all constituent activities of the planning request have terminated. Awaiting confirmation of the status of the planning request. |
| TERMINATED | 8 | The planning request has completed, either successfully or with a failure condition (further information is provided in statusInfo). |

#### Planning Requests: Service Support Data Types

##### Data Type: RequestStatusUpdate

###### Overview

RequestStatusUpdate is a data structure that is used to report changes in status of the RequestInstance as it proceeds through both planning and plan execution functions. Reporting is the responsibility of the planning function.

Planning request status updates may be distributed to subscribing applications, including both Users and status displays, to inform them of the latest status of the planning request. This may be particularly relevant in conjunction with a plan execution function. Status updates may be stored in planning request history to provide a complete record of evolving status over time.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **RequestStatusUpdate** | Extends | MAL::Composite | SFP | 404 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| requestInstance | MAL::ObjectRef <RequestInstance> | No | Reference to the planning request instance to which the status update relates. |
| timestamp | MAL::Time | No | Time of status update. |
| status | RequestStatusEnum | No | Current status of the planning request. |
| outputPlanRefs | List <MAL::ObjectRef <Plan>> | Yes | Reference to the output Plan(s) that contains the activities resulting from the planning request. Where multiple alternate plans have been generated, these may be listed here. It should be noted that this is only available once the planning request has been processed and successfully planned. The outputPlanRefs may be updated following iterative planning cycles or re-planning. |
| returnData | List <MAL::NamedValue> | Yes | Optional return data from the planning process, provided as a list of ID-Value pairs. This can be used to provide additional information required by the User to interpret the planned operations. |
| statusInfo | MAL::String | Yes | StatusInfo provides the reason for termination and is customizable, but if the following conditions exist then the specified text shall be used:  - Completed (all constituent activities completed successfully);  - Expired (constituent activities expired prior to execution);  - Failed (constituent activities failed during execution);  - Deleted (constituent activities were deleted);  - PartiallyCompleted.  It may also be used to provide the reason for rejection. |
| errorCode | MAL::Integer | Yes | Error Code optional in the case of a failure status for the planning request (for example Terminated state with statusInfo Failed). The codes are implementation specific. |
| errorInfo | MAL::String | Yes | Supplementary error information. |

##### Data Type: PlanningRequestDetails

###### Overview

PlanningRequestDetails is a data structure used in the context of the MPS Planning Request service SubmitRequest and UpdateRequest operations, where the RequestInstance MO object cannot be used as the full identity of the resulting RequestInstance (key and version) is not yet known.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PlanningRequestDetails** | Extends | MAL::Composite | SFP | 405 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| userReference | MAL::Identifier | No | User supplied reference for the planning request. This is distinct from the identity of the RequestInstance that is assigned by the planning function. No guarantees are made by the planning system about the contents of this identifier; that is entirely up to the user who supplies the reference. |
| definition | MAL::ObjectRef <RequestDefinition> | Yes | Reference to the RequestDefinition from which the RequestInstance was created, if a planning request template was used. |
| planningPeriod | MAL::Identifier | No | Specifies which planning period the planning request applies to. Planning period IDs are mission specific, but can be used to indicate mission phase; planning cycle; or ‘semester’ in observatory missions. |
| validityTimes | List <TimeWindow> | Yes | Validity period for the planning request, expressed as one or more time windows. The planning request must be satisfied within this period.  When multiple TimeWindows are provided, the planning request may be satisfied within any individual TimeWindow.  If this field is null, no restriction is placed on the times between which this request must be planned. |
| validityEvents | List <EventWindow> | Yes | Validity period for the planning request, expressed as one or more event windows. The planning request must be satisfied within this period.  When multiple EventWindows are provided, the planning request may be satisfied within any individual EventWindow.  If this field is null, no restriction is placed on any events between which this request must be planned. |
| timeSystem | MAL::Identifier | Yes | Specifies the time system used for all time fields within the planning request (see 4.4.1).  If null, the default time system is used. |
| user | MAL::ObjectRef <PlanningUser> | No | The User ID for the person or organization raising the planning request. |
| description | MAL::String | No | Description of the request. |
| arguments | List <Argument> | Yes | List of named argument values. If created from a template planning request, this will include the arguments defined in the RequestDefinition. |
| standingOrder | MAL::Boolean | No | A flag that indicates whether the planning request is for a repetitive standing order (unbounded other than by the validity period), or is a one-off request. If it is a standing order, then the supplied **activity details** must be an ActivityNode with specification of the repetition criteria. It should be noted that a one-off request can still include repetition. |
| activities | List <ActivityDetails> | No | Set of activity details specifying requested activities. |
| inputPlanRef | MAL::ObjectRef <Plan> | Yes | Reference to an existing Plan (output of one planning function) submitted as a planning request to another planning function in the context of a distributed or hierarchical planning system.  Only one of inputPlanRef and inputPlan should be present within the planning request. |
| inputPlan | Plan | Yes | An existing Plan (output of one planning function) submitted as a planning request to another planning function in the context of a distributed or hierarchical planning system. The Plan is embedded within the planning request.  Only one of inputPlanRef and inputPlan should be present within the planning request. |
| comments | MAL::String | Yes | Free text for any additional user comments about the request. |

##### Data Type: PlanningRequestResponse

###### Overview

PlanningRequestResponse is a data structure used in the context of the MPS Planning Request service SubmitRequest and UpdateRequest operations, in response to the submitted PlanningRequestDetails defined above. It contains a reference to the created RequestInstance and the supplied userReference to allow the user to correlate the two.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PlanningRequestResponse** | Extends | MAL::Composite | SFP | 406 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| instance | MAL::ObjectRef <RequestInstance> | No | Reference to the RequestInstance created in response to a SubmitRequest operation, or the updated version of the RequestInstance following an UpdateRequest operation. |
| userReference | MAL::Identifier | No | User supplied reference for the planning request. This is distinct from the identity of the RequestInstance that is assigned by the planning function. |

##### Data Type: RequestSummaryStatus

###### Overview

RequestSummaryStatus is a data structure used in the context of the MPS Planning Request service getRequestSummaries operation, where a list of these structures is returned. It contains header fields of the planning request and its status, but not the request content (arguments, activities and constraints).

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **RequestSummaryStatus** | Extends | MAL::Composite | SFP | 407 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| requestInstance | MAL::ObjectRef <RequestInstance> | No | Reference to the RequestInstance (key and version). |
| userReference | MAL::Identifier | Yes | Optional user supplied reference for the planning request. This is distinct from the identity of the RequestInstance that is assigned by the planning function. |
| creationTime | MAL::Time | No | Creation date and time of the RequestInstance version. |
| definition | MAL::ObjectRef <RequestDefinition> | Yes | Reference to the RequestDefinition from which the RequestInstance was created, if a planning request template was used. |
| planningPeriod | MAL::Identifier | No | Specifies which planning period the planning request applies to. Planning period IDs are mission specific, but can be used to indicate mission phase; planning cycle; or ‘semester’ in observatory missions. |
| validityTimes | List <TimeWindow> | Yes | Validity period for the planning request, expressed as one or more time windows. The planning request must be satisfied within this period.  Only one of validityTime or validityEvent should be present in a planning request. |
| validityEvents | List <EventWindow> | Yes | Validity period for the planning request, expressed as one or more event windows. The planning request must be satisfied within this period.  Only one of validityTime or validityEvent should be present in a planning request. |
| user | MAL::ObjectRef <PlanningUser> | No | The User ID for the person or organization raising the planning request. |
| description | MAL::String | No | Description of the request. |
| standingOrder | MAL::Boolean | No | A flag that indicates whether the planning request is for a repetitive standing order (unbounded other than by the validity period), or is a one-off request. |
| comments | MAL::String | Yes | Free text for any additional user comments about the request. |
| status | RequestStatusEnum | No | Current status of the ActivityInstance (see planning request state model in 4.5.5.2). |
| outputPlanRefs | List <MAL::ObjectRef <Plan>> | Yes | References to output Plan(s) that contains the activities resulting from the planning request. |
| statusInfo | MAL::String | Yes | StatusInfo provides the reason for termination and is customizable, but includes:  - Completed (all constituent activities completed successfully);  - Expired (constituent activities expired prior to execution);  - Failed (constituent activities failed during execution);  - Deleted (constituent activities were deleted);  - PartiallyCompleted.  It may also be used to provide the reason for rejection. |

##### Data Type: RequestFilter

###### Overview

RequestFilter is a data structure used in the context of MPS Planning Request Service operations to specify a filtered set of planning requests.

NOTE – All fields are nullable and it is valid to specify a RequestFilter with no filter criteria; this corresponds to an open filter in which all available planning requests are returned.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **RequestFilter** | Extends | MAL::Composite | SFP | 408 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| domain | List <MAL::Identifier> | Yes | Domain of the RequestInstance. An ordered list representing a domain hierarchy, ‘\*’ can be used to represent a wildcard at that level. |
| instanceID | MAL::ObjectRef <RequestInstance> | Yes | Identity (key and version) of the RequestInstance. |
| creationTime | MAL::Time | Yes | Creation date and time of the RequestInstance version. |
| definitionID | MAL::ObjectRef <RequestDefinition> | Yes | Identity (key and version) of the RequestDefinition from which the RequestInstance was created. |
| userID | MAL::ObjectRef <PlanningUser> | Yes | userID of the User who initiated the RequestInstance. |
| userReference | MAL::Identifier | Yes | Reference supplied by User when submitting the RequestInstance. |
| status | RequestStatusEnum | Yes | Current status (enum) of the RequestInstance. |
| outputPlanRefs | List <MAL::ObjectRef <Plan>> | Yes | Reference to the output Plan(s) generated in response to the RequestInstance. |

###### Requirements

All filter criteria specified shall be applied using a logical AND (not OR).

### Plans

#### Plan MO Objects

##### Data Type: Plan

###### Overview

A Plan is an MO object that contains both the static fields that define a version of a plan and dynamic fields that hold its current state. Its identity is defined by a constant key and an evolving version, which is updated each time the Plan is revised.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **Plan** | Extends | MAL::Object | SFP | 501 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| identity | MAL::ObjectIdentity | No | Identity of the Plan, including version. |
| isPatchPlan | MAL::Boolean | No | Flag indicating if the Plan is a patch plan that only contains details of the changes from the precursor Plan. A patch plan must have a precursor. It must also include a single PlanRevision relative to the precursor Plan. |
| precursorPlan | MAL::ObjectRef <Plan> | Yes | Reference to a precursor (or predecessor) Plan from which the changes are detailed in the Plan. This may be used if there is an iterative re-planning cycle in which successive plans overlap, or where a previous Plan has been updated through re-planning. If there is no precursor, then the Plan must be a self-standing full plan.  If the Plan is a Patch Plan, then a precursor plan must be specified. |
| targetPlan | MAL::ObjectRef  <Plan> | Yes | Applicable only for patch plans, this is a reference to the target Plan. This target Plan is the result of applying the patch plan to the precursor Plan and is distinct from the identity of the patch plan itself. Patch plans are not permitted in the context of a planning request. |
| information | PlanInformation | No | Contains header information relating to the Plan, including its originator and validity period. |
| items | PlannedItems | No | Contains the planned activities and events that constitute the Plan. |
| revisions | List <PlanRevision> | Yes | Details the changes between this Plan and other Plans (or other versions of the same Plan), usually the precursor Plan. Optional, but must contain at least one element in a patch plan.  Multiple revisions may be included, documenting the differences with any other version of a Plan. This can be used to provide a change history for successive versions of the same Plan, or to document the differences between alternate Plans. |
| resources | List <ResourceProfile> | Yes | Optional. If present, must contain one ResourceProfile per planning resource. These profiles shall provide the projected evolution of the value of a planning resource, or its initial value at the start of the Plan. Which approach is used is a deployment choice. |
| isAlternate | MAL::Boolean | No | Flag indicating if the Plan has currently been released as an Operational or Alternate plan. |
| status | PlanStatusEnum | No | Current status of the Plan. |
| statusInfo | MAL::String | Yes | Supplementary information for a Plan in the Terminated state. This is customizable, but if the following conditions exist then the specified text shall be used:  - Completed (nominal);  - Superseded (by a successor Plan);  - Revoked;  - Cancelled (deactivated after start of execution);  - Expired. |

#### Plan Status

##### Data Type: PlanStatusEnum

###### Overview

PlanStatusEnum represents the status of a given Plan object.

###### Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Name | **PlanStatusEnum** | SFP | 502 |

| Status | Value | Description |
| --- | --- | --- |
| DRAFT | 1 | The Plan has been saved by the planning function. |
| RELEASED | 2 | The Plan has been released for operational use by the planning function. |
| SUBMITTED | 3 | The Plan has been submitted to the plan execution function and is available for use, but will not execute until activated. |
| ACTIVATED | 4 | The Plan has been activated by the plan execution function. |
| TERMINATED | 5 | The Plan has reached a terminal state, as detailed in the statusInfo. This includes the following cases:  - Completed (nominal);  - Superseded by a successor Plan;  - Revoked by a User;  - Cancelled (deactivated after start of execution);  - Expired (reached the end of its validity period without being activated). |

#### Plans: Subordinate Data Types

##### Data Type: PlanInformation

###### Overview

The PlanInformation section of a plan contains administrative and validity details associated with the plan as a whole.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PlanInformation** | Extends | MAL::Composite | SFP | 503 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| originator | MAL::Identifier | No | Identity of the entity or system responsible for the production of the plan. The implementing planning system is responsible for defining the value to be provided for this field. |
| productionTime | MAL::Time | No | Date and time of production of the plan. |
| description | MAL::String | No | Description of the plan. |
| comments | MAL::String | Yes | Field for additional comments or notes to the operations team regarding the plan. |
| validityStart | MAL::Time | No | Start of validity period for the plan.  The validity period defines when the plan is available for operational use. It cannot be used outside its validity period. |
| validityEnd | MAL::Time | No | End of validity period for the plan. |
| planPeriodStart | Trigger | No | Start of the plan period.  The plan period defines the start and end points of the plan. Planned items (planning activities and events) contained within the plan must at least partially overlap the plan period. The use of the trigger structure allows this to be specified in terms of time, position, pointing, or planning events. Examples are:  - a specified period of time;  - an orbital repeat cycle;  - a period between two events. |
| planPeriodEnd | Trigger | No | End of the plan period. |
| timeSystem | MAL::Identifier | Yes | Specifies the time system used for all time fields within the Plan (see 4.4.1).  If Null, the default time system is used. |

##### Data Type: PlannedItems

###### Overview

The PlannedItems section of the Plan specifies the set of planning activities and planning events contained within the Plan. It comprises two lists of contained MO objects: one of EventInstances and one of ActivityInstances. Both lists can be empty.

###### Definition

If the Plan is a full plan, then there must be an entry in the list for all planned items contained within the plan, whether changed or not. If the plan is a patch plan, then there must only be an entry in the list for those planned items that have changed (new or modified).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PlannedItems** | Extends | MAL::Composite | SFP | 504 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| plannedEvents | List <EventInstance> | Yes | List of planned events contained within the Plan. |
| plannedActivities | List <ActivityInstance> | Yes | List of planned activities contained within the Plan. |

#### Plan Revision Data Types [Optional]

##### Data Type: PlanRevision

###### Overview

Each PlanRevision comprises an ordered set of ItemRevisions that document the change to individual planned items (planning events and activities). Each ItemRevision references an individual EventInstance or ActivityInstance and indicates whether the planned item is new, modified or deleted in the current Plan.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PlanRevision** | Extends | MAL::Composite | SFP | 505 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| revisedPlan | MAL::ObjectRef <Plan> | No | Reference to the Plan (key and version) with respect to which the plan revisions are detailed. Typically this is the precursor Plan, but any other Plan can be used. |
| revisionStart | MAL::Time | No | Start time of the earliest revision. |
| revisionEnd | MAL::Time | No | End time of the latest revision. |
| itemRevisions | List <ItemRevision> | Yes | Ordered list (earliest to latest) of revisions to planned items (activity and event instances). |

NOTE – Unmodified items do not appear in a PlanRevision.

###### Requirements

The order of the ItemRevisions shall be from earliest to latest modification within the plan period, to allow for the earliest and most critical changes to be applied first to a currently executing plan.

New or modified items shall also exist in the set of planned items in a Plan, but deleted items shall not be contained within the current Plan.

##### Data Type: ItemRevision

###### Overview

An ItemRevision represents the changes that were made to a single planned item inside a revision.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ItemRevision** | Extends | MAL::Composite | SFP | 506 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| itemRef | MAL::ObjectRef <MAL::Element> | No | Object Type: ActivityInstance | EventInstance.  Reference to a planned ActivityInstance or EventInstance that is new or modified in the current Plan, or has been deleted with respect to the referenced revisedPlan. |
| revisionStatus | RevisionStatusEnum | No | Revision status of the referenced item. May be one of New, Modified, or Deleted.  Default = Undefined. |

##### Data Type: RevisionStatusEnum

###### Overview

The RevisionStatusEnum represents the type of changes that were made to an item in a given revision.

###### Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Name | **RevisionStatusEnum** | SFP | 507 |

| Status | Value | Description |
| --- | --- | --- |
| NEW | 1 | The item is new in this revision of the Plan. |
| MODIFIED | 2 | The item has been modified in this revision of the Plan. |
| DELETED | 3 | The item has been deleted in this revision of the Plan. |
| UNDEFINED | 4 | The item is unchanged in this revision of the Plan, or its revision status is undefined. |

#### Plans: Service Support Data Types

##### Data Type: PlanUpdate

###### Overview

PlanUpdate is a data structure that is used to report changes in status of the Plan as it proceeds through both planning and plan execution functions. It is returned in the context of the MPS Plan Distribution service getPlanStatus and monitorPlanStatus operations, and also the MPS Plan Execution Control service monitorPlanExecution and getPlanStatus operations.

PlanUpdates may be distributed to subscribing applications, including status displays, to inform them of the latest status of a Plan. PlanUpdates may be stored in plan history to provide a complete record of evolving status over time.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PlanUpdate** | Extends | MAL::Composite | SFP | 508 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| plan | MAL::ObjectRef <Plan> | No | Reference to the Plan (key and version) to which the status update relates. |
| timestamp | MAL::Time | No | Time of status update. |
| isAlternate | MAL::Boolean | No | Flag indicating if the Plan has currently been released as an Operational or Alternate plan. |
| status | PlanStatusEnum | No | Current status of the Plan. |
| statusInfo | MAL::String | Yes | Supplementary information for a Plan in the Terminated state. This is customizable, but if the following conditions exist then the specified text shall be used:  - Completed (nominal);  - Superseded by a successor Plan;  - Revoked by a User;  - Cancelled (deactivated after start of execution);  - Expired (reached the end of its validity period without being activated). |

##### Data Type: PlanSummaryStatus

###### Overview

PlanSummaryStatus is a data structure that provides a summary view of a Plan that includes the PlanInformation section and current status, but not the full details of the Plan. It is returned in the context of the MPS Plan Distribution service GetPlanSummaries operation.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PlanSummaryStatus** | Extends | MAL::Composite | SFP | 509 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| plan | MAL::ObjectRef <Plan> | No | Reference to the Plan (key and version) to which the summary status relates. |
| isPatchPlan | MAL::Boolean | No | Flag indicating if the Plan is a patch plan that only contains details of the changes from the precursor Plan. A patch plan must have a precursor. It must also include a single PlanRevision relative to the precursor Plan. |
| precursorPlan | MAL::ObjectRef <Plan> | Yes | Reference to a precursor (or predecessor) Plan from which the changes are detailed in the Plan. This may be used if there is an iterative re-planning cycle in which successive plans overlap, or where a previous Plan has been updated through re-planning. If there is no precursor, then the Plan must be a self-standing full plan.  If the Plan is a Patch Plan, then a precursor plan must be specified. |
| targetPlan | MAL::ObjectRef <Plan> | Yes | Applicable only for patch plans, this is a reference to the target Plan. This target Plan is the result of applying the patch plan to the precursor Plan and is distinct from the identity of the patch plan itself. Patch plans are not permitted in the context of a planning request. |
| information | PlanInformation | No | Contains header information relating to the Plan, including its originator and validity period. |
| isAlternate | MAL::Boolean | No | Flag indicating if the Plan has currently been released as an Operational or Alternate plan. |
| status | PlanStatusEnum | No | Current status of the Plan. |
| statusInfo | MAL::String | Yes | Supplementary information for a Plan in the Terminated state. |

##### Data Type: PlanActivationStatus

###### Overview

PlanActivationStatus is a data structure that returns the activation status of a Plan in the context of the MPS Plan Execution Control service ActivatePlan and DeactivatePlan operations.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PlanActivationStatus** | Extends | MAL::Composite | SFP | 510 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| plan | MAL::ObjectRef <Plan> | No | Reference to the Plan (key and version) to which the status relates. |
| status | PlanStatusEnum | No | Current status of the Plan. |
| activationInfo | MAL::String | No | ActivationInfo provides customizable detailed information on the result of the activation/deactivation request for the referenced Plan. |

##### Data Type: SubPlanUpdate

###### Overview

SubPlanUpdate is a data structure that is used to report changes in status of a sub-plan during plan execution. It is returned in the context of the MPS Plan Execution Control service MonitorSubPlanExecution and GetSubPlanStatus operations.

Sub-plans are not defined as objects within the MPS model. Individual activities within a Plan may be associated with a single sub-plan via its Identifier. The plan execution function is responsible for managing and reporting sub-plan status associated with relevant Plan Execution Control service operations, if supported.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **SubPlanUpdate** | Extends | MAL::Composite | SFP | 511 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| subPlan | MAL::Identifier | No | Identifier of the sub-plan to which the update relates. |
| timestamp | MAL::Time | No | Time of status update. |
| status | SubPlanStatusEnum | No | Current status of the sub-plan, which may be Activated or Deactivated. |

##### Data Type: SubPlanStatusEnum

###### Overview

This enumeration may be used to indicate whether or not a given subplan is active.

###### Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Name | **SubPlanStatusEnum** | SFP | 512 |

| Status | Value | Description |
| --- | --- | --- |
| ACTIVATED | 1 | The sub-plan is active. |
| DEACTIVATED | 2 | The sub-plan is not active. |

##### Data Type: SubPlanActivationStatus

###### Overview

SubPlanActivationStatus is a data structure that returns the activation status of a sub-plan in the context of the MPS Plan Execution Control service ActivateSubPlan and DeactivateSubPlan operations.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **SubPlanActivationStatus** | Extends | MAL::Composite | SFP | 513 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| plan | MAL::Identifier | No | Identifier of the sub-plan to which the status relates. |
| status | SubPlanStatusEnum | No | Current status of the sub-plan, which may be Activated or Deactivated. |
| activationInfo | MAL::String | No | ActivationInfo provides customizable detailed information on the result of the activation/deactivation request for the referenced sub-plan. |

##### Data Type: PlanQuery

###### Overview

PlanQuery is a data structure used in the context of queryPlan operation of the MPS Plan Distribution Service. It is used to specify search criteria for querying the available set of Plans.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PlanQuery** | Extends | MAL::Composite | SFP | 514 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| planID | MAL::ObjectRef <Plan> | Yes | Query for Plans with the specified PlanID. |
| hasPrecursor | MAL::Boolean | Yes | Query for Plans with or without a precursor. |
| isPatchPlan | MAL::Boolean | Yes | Query for Plans that are or are not patch plans. |
| precursorPlan | MAL::ObjectRef <Plan> | Yes | Query for Plans with the specified precursor Plan. |
| targetPlan | MAL::ObjectRef <Plan> | Yes | Applicable only for patch plans. Query for patch plans that have the specified target Plan. |
| originator | MAL::Identifier | Yes | Query for Plans with the specified originator. |
| productionTime | TimeWindow | Yes | Query for Plans with a production date in the specified range. |
| validityPeriod | TimeWindow | Yes | Query for Plans with a validity period within (overlapping with) the specified range. |
| isAlternate | MAL::Boolean | Yes | Query for Plans that are or are not Alternate plans. |
| status | List <PlanStatusEnum> | Yes | Query for Plans that have a current status matching one of the specified list of Plan statuses. |
| plannedEvents | List <MAL::ObjectRef <EventDefinition>> | Yes | Query for Plans that contain EventInstances inside plannedItems whose definition matches one of the specified list of EventDefinitions. |
| plannedActivities | List <MAL::ObjectRef <ActivityDefinition>> | Yes | Query for Plans that contain ActivityInstances inside plannedItems whose definition matches one of the specified list of ActivityDefinitions. |
| revisedEvents | List <MAL::ObjectRef <EventDefinition>> | Yes | Query for patch plans that contain EventInstances inside their revisions whose definition matches one of the specified list of EventDefinitions. |
| revisedActivities | List <MAL::ObjectRef <ActivityDefinition>> | Yes | Query for patch plans that contain ActivityInstances inside their revisions whose definition matches one of the specified list of ActivityDefinitions. |

###### Requirements

All fields are nullable, in which case they shall not apply as a search criteria.

##### Data Type: PartialPlan

###### Overview

A PartialPlan is a data structure returned from the getPartialPlan operation of the Plan Distribution Service that contains a reference to the source Plan, the criteria used to select the partial plan, and the partial plan itself. The partial plan uses the same structure as a normal Plan, with header fields matching those of the source Plan, but only containing the subset of ActivityInstances that matches the selection criteria. Whether EventInstances and Resources are included is implementation specific, but it might be assumed that any events and resources related to the selected ActivityInstances would be included in the returned partial plan.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PartialPlan** | Extends | MAL::Composite | SFP | 515 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| sourcePlan | MAL::ObjectRef <Plan> | No | Reference to the Plan of which the partial plan is a selected subset. |
| domain | List <MAL::Identifier> | Yes | Selection criterion based on the domain of contained ActivityInstances.  An ordered list representing a domain hierarchy, ‘\*’ can be used to represent a wildcard at that level. |
| subPlan | MAL::Identifier | Yes | Selection criterion based on the subPlan of contained ActivityInstances. |
| tags | List <MAL::String> | Yes | Selection criterion based on tags associated with contained ActivityInstances. |
| partialPlanStart | Trigger | Yes | Selection criterion indicating the start of a range of time, position, or events associated with contained ActivityInstances. |
| partialPlanEnd | Trigger | Yes | Selection criterion indicating the end of a range of time, position, or events associated with contained ActivityInstances. |
| partialPlan | Plan | No | The returned partial plan. |

##### Data Type: PlanFilter

###### Overview

PlanFilter is a data structure used in the context of MPS Plan Distribution Service operations to specify a filtered set of Plans.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PlanFilter** | Extends | MAL::Composite | SFP | 516 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| domain | List <MAL::Identifier> | Yes | Domain of the Plan.  An ordered list representing a domain hierarchy, ‘\*’ can be used to represent a wildcard at that level. |
| planID | MAL::ObjectRef <Plan> | Yes | Identity (key and version) of the Plan. |
| precursorPlan | MAL::ObjectRef <Plan> | Yes | Identity (key and version) of the precursor Plan. |
| status | PlanStatusEnum | Yes | Current status (enum) of the Plan. |
| originator | MAL::Identifier | Yes | Originator of the Plan. |
| validityPeriod | TimeWindow | Yes | Period of time with which the validity period of the Plan overlaps. |

###### Requirements

All filter criteria specified shall be applied using a logical AND (not OR).

##### Data Type: PartialPlanFilter

###### Overview

PartialPlanFilter is a data structure input to the getPartialPlan operation of the Plan Distribution Service that contains a reference to the source Plan, and specifies the criteria used to select the partial plan.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PartialPlanFilter** | Extends | MAL::Composite | SFP | 517 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| sourcePlan | MAL::ObjectRef <Plan> | No | Reference to the Plan of which the partial plan is a selected subset. |
| domain | List <MAL::Identifier> | Yes | Selection criterion based on the domain of contained ActivityInstances.  An ordered list representing a domain hierarchy, ‘\*’ can be used to represent a wildcard at that level. |
| subPlan | MAL::Identifier | Yes | Selection criterion based on the subPlan of contained ActivityInstances. |
| tags | List <MAL::String> | Yes | Selection criterion based on tags associated with contained ActivityInstances |
| partialPlanStart | Trigger | Yes | Selection criterion indicating the start of a range of time, position, or events associated with contained ActivityInstances. If no actual time is known for a Trigger, its predicted time may be used instead to derive the relevant range. |
| partialPlanEnd | Trigger | Yes | Selection criterion indicating the end of a range of time, position, or events associated with contained ActivityInstances. If no actual time is known for a Trigger, its predicted time may be used instead to derive the relevant range. |

### Planning Users

#### Data Type: PlanningUser

##### Overview

The source of a planning request is the user that raises it, and this is identified in the user field of a RequestInstance as a reference to a PlanningUser object.

##### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PlanningUser** | Extends | MAL::Object | SFP | 601 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| identity | MAL::ObjectIdentity | No | Identity of the PlanningUser, including version. |

##### Discussion

The information held on planning users is outside the scope of this Recommended Standard. The only requirement on the PlanningUser object is that it is formulated as an MO object, with an associated object identity. Any additional content [fields] of the PlanningUser object are system specific. As the PlanningUser object is not transferred in any message of the MPS services, there is no requirement to fully define the data structure within this Recommended Standard, but it is referenced in other MPS objects and data structures using a field of type MAL::ObjectRef.

### Custom Functions [Optional]

#### Data Type: FunctionDefinitionDetails

##### Overview

FunctionDefinitionDetails is a data structure that contains static configuration data relating to custom functions: built-in Boolean functions of an MPS system, each of which has a specified Identifier and optional set of argument definitions. This may change over time, each comprising a separate version of the definition. FunctionDefinitions form part of the planning configuration data.

##### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **FunctionDefinitionDetails** | Extends | MAL::Composite | SFP | 701 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| functionID | MAL::Identifier | No | ID of the custom function. |
| version | MAL::UInteger | No | Version of the FunctionDefinition. |
| description | MAL::String | No | Description of the custom function. |
| argDefs | List <ArgDef> | Yes | List of argument definitions. |

#### Data Type: FunctionDetails

##### Overview

Contains the information required to invoke a defined function, including the specification of argument values.

##### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **FunctionDetails** | Extends | MAL::Composite | SFP | 702 |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| function | MAL::Identifier | No | ID of a specific FunctionDefinition. |
| argSpecs | List <ArgSpec> | Yes | Set of argument specifications for each argument definition contained in the referenced function definition. These supply a value for each argument, or an expression to enable the value to be derived. |

## MPS Data Types

### Overview

MPS Data Types are supporting data structures and enumerations used in the context of MPS Service Objects and MPS service messages:

* Base Data Types;
* Position and Direction Data Types [Optional];
* Expressions;
* Arguments;
* Constraints;
* Effects [Optional];
* Triggers;
* Repetitions.

#### MAL Data Types

The MPS information model is built on existing MAL data types (see reference [2]). The following MAL data types are used in the current MPS information model (the section numbers are in scope of reference [2] here):

1. MAL fundamental types (section 4.3), with the abstract types Element, Attribute, and Composite;
2. MO Objects (section 4.4), based on the composite type ObjectIdentity;
3. MAL attributes (section 4.5), with the types Blob, Boolean, Duration, Float, Double, Identifier, Octet, UOctet, Short, UShort, Integer, UInteger, Long, ULong, String, Time, FineTime, URI, and ObjectRef;
4. Enumeration type AttributeType (section 4.6.4);
5. Composites types Pair (section 4.6.10), NamedValue (section 4.6.11), File (section 4.6.12), and ObjectIdentity (section 4.6.13).

#### Expressions

Expressions are denoted in the MPS information model as Expression <T> for any type T. An expression is not a data type itself, but will rather evaluate to a specific data type. At the MAL layer, an expression translates to a MAL::Element, which is an abstract type that can represent any of the supported types T. The expression types are fully specified in 4.6.4.

### Base Data Types

#### Data Type: ArgTypeEnum

##### Overview

ArgTypeEnum is an MPS extension of the MAL::AttributeType enumeration (see reference [2] section 4.6.4) that also allows specification of the data types Position or Direction.

##### Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Name | **ArgTypeEnum** | SFP |  |

| Status | Value | Description |
| --- | --- | --- |
| BLOB | 1 | Binary object. |
| BOOLEAN | 2 | Boolean value (True or False). |
| DURATION | 3 | Length of time in nanosecond resolution. |
| FLOAT | 4 | Floating point number (32 bits). |
| DOUBLE | 5 | Double precision floating point number (64 bits). |
| IDENTIFIER | 6 | The Identifier structure is used to store an identifier and can be used for indexing. It is a variable-length, unbounded, Unicode string. |
| OCTET | 7 | Signed 8 bit Integer. |
| UOCTET | 8 | Unsigned 8 bit Integer. |
| SHORT | 9 | Signed 16 bit Integer. |
| USHORT | 10 | Unsigned 16 bit Integer. |
| INTEGER | 11 | Signed 32 bit Integer. |
| UINTEGER | 12 | Unsigned 32 bit Integer. |
| LONG | 13 | Signed 64 bit Integer. |
| ULONG | 14 | Unsigned 64 bit Integer. |
| STRING | 15 | Text. It is a variable length, unbounded, Unicode string. |
| TIME | 16 | Absolute date-time to millisecond resolution. |
| FINETIME | 17 | Absolute date-time to nanosecond resolution. |
| URI | 18 | Uniform Resource Identifier (address). It is a variable-length, unbounded Unicode string (see reference [4]). |
| OBJECTREF | 19 | Object Reference. |
| POSITION | 129 | MPS Position. |
| DIRECTION | 130 | MPS Direction. |
| ANY | 131 | Argument may be of any type. It will map to a MAL::Element. It can be used to specify an argument list of heterogeneous types. |

NOTE – To support future extension of the set of MAL Attributes, enumeration values up to and including 128 are reserved for MAL Attributes. MPS-specific argument types are assigned enumeration values from 129 upwards.

#### Data Type: NamedElement

##### Overview

The NamedElement composite represents a pair of a MAL::Identifier and an abstract MAL::Element. It is an extension of the MAL::NamedValue composite that adds support for non-MAL::Attribute values.

##### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **NamedElement** | Extends | MAL::Composite | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| name | MAL::Identifier | No | Name identifying the element. |
| value | MAL::Element | Yes | Corresponding MAL::Element value. |

#### 

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | 3 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | 4 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | 5 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | 6 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | 7 |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

### Position and Direction Data Types [Optional]

#### Overview

This subsection defines MPS Position and Direction data types and support types required in the definition of pointing constraints. These are consistent with those used within CCSDS Navigation data format Recommended Standards, and specifically the Pointing Request Message (PRM) (reference [10]), but for MPS, to enable use of the MO Framework, they are defined explicitly in terms of MAL Attributes.

MPS Position and Direction data types are only required in the context of the following MPS data structures:

* Geometric Constraints (see 4.6.6.4);
* Triggers of type PositionTrigger and DirectionTrigger (see 4.6.8);
* Repetitions of type PositionRepetition and DirectionRepetition (see 4.6.9).

As all of these are considered optional elements of the MPS information model, MPS Position and Direction data types are themselves optional.

#### Position Data Types

##### Data Type: Position

###### Overview

Abstract type that represents a unique position in three-dimensional space. Depending on the concrete subtype used, the actual position may be derived in different manners.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **Position** | Extends | MAL::Composite | SFP | Abstract |

##### Data Type: CartesianPosition

###### Overview

Concrete type representing a Position in Cartesian coordinates.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **CartesianPosition** | Extends | Position | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| x | MAL::Double | No | Cartesian x coordinate defined in the given frame and with values of the given unit. |
| y | MAL::Double | No | Cartesian y coordinate defined in the given frame and with values of the given unit. |
| z | MAL::Double | No | Cartesian z coordinate defined in the given frame and with values of the given unit. |
| frame | MAL::Identifier | No | Reference frame within which the position is expressed (see 4.4.2). |
| units | MAL::String | Yes | The units for the quantity of distance.  Default = ‘km’. |

##### Data Type: SurfacePosition

###### Overview

A SurfacePosition is typically used to specify a coordinate on the surface of a celestial body. Optionally, an altitude above the surface may also be specified. The reference ellipsoid used to define the surface may be mission-specific.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **SurfacePosition** | Extends | Position | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| longitude | MAL::Double | No | Angular coordinate. May also represent azimuth. |
| latitude | MAL::Double | No | Angular coordinate. May also represent elevation. |
| frame | MAL::Identifier | No | Reference frame used to determine the origin and orientation of the reference ellipsoid. Must be a celestial body reference frame (see 4.4.2). |
| units | MAL::String | Yes | The units for the quantity of angle, in which to express the longitude and latitude.  Default = ‘deg’. |
| altitude | MAL::Double | Yes | Altitude above a reference ellipsoid (negative values allowed).  Default = 0. |
| altitudeUnits | MAL::String | Yes | The units for the quantity of altitude.  Default = ‘m’. |

##### Data Type: OrbitFilePosition

###### Overview

An OrbitFilePosition represents a Position that is defined with respect to some Orbit Data Message (ODM) file (reference [D10]).

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **OrbitFilePosition** | Extends | Position | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| orbitFile | MAL::File | No | Name of or reference to a file containing an ODM. |

###### Requirements

The planning system shall evaluate the resulting orbit when needed to obtain the position at a given time.

##### Data Type: OrbitalPosition

###### Overview

An OrbitalPosition represents a Position that is defined with respect to some mission-specific orbit. The conventions used to derive the orbitNumber and orbitAngle follow from a mission-specific definition.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **OrbitalPosition** | Extends | Position | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| orbitNumber | MAL::Integer | No | Orbit number. Depending on the relativeOrbit flag, the orbit number may be absolute (since start of mission) or relative (to the orbital repeat cycle). |
| relativeOrbit | MAL::Boolean | No | Flag indicating if the orbit number is absolute or relative to the orbital repeat cycle. |
| orbitAngle | MAL::Double | No | Angle within orbit. Whether this angle is the mean or true anomaly and from which datum it is measured are mission specific. |
| units | MAL::String | Yes | The units for the quantity of angle, in which the orbitAngle is expressed in.  Default = ‘deg’. |

##### Data Type: ObjectPosition

###### Overview

An ObjectPosition is a Position that coincides with the position of an existing object. The manner in which the planning system derives the value of this Position from the name of the referenced object is implementation-defined.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ObjectPosition** | Extends | Position | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| object | MAL::Identifier | No | Name or identifier of a catalogued celestial object or a mission specific object. |

##### Data Type: PositionReference

###### Overview

A PositionReference is a Position that is evaluated based on a given reference position. The manner in which the Position is computed by the planning system based on this reference may be mission specific.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PositionReference** | Extends | Position | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| reference | MAL::Identifier | No | Name of a mission specific position definition. |

#### Direction Data Types

##### Data Type: Direction

###### Overview

Abstract type that represents a unique direction in three-dimensional space. The actual manner in which this direction is evaluated depends on the concrete subtype used.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **Direction** | Extends | MAL::Composite | SFP | Abstract |

##### Data Type: CartesianDirection

###### Overview

Dimensionless unit vector. Either a direction in the base frame or in a secondary frame may be defined.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **CartesianDirection** | Extends | Direction | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| x | MAL::Double | No | Cartesian x coordinate defined in the given frame. |
| y | MAL::Double | No | Cartesian y coordinate defined in the given frame. |
| z | MAL::Double | No | Cartesian z coordinate defined in the given frame. |
| frame | MAL::Identifier | No | Reference frame within which the direction is expressed (see 4.4.2). |

##### Data Type: SphericalDirection

###### Overview

Typically used to define a direction in a secondary frame. When used to specify a surface coordinate, this actually represents a {longitude, latitude} pair.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **SphericalDirection** | Extends | Direction | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| azimuth | MAL::Double | No | Angular coordinate. May also represent longitude. |
| elevation | MAL::Double | No | Angular coordinate. May also represent latitude. |
| frame | MAL::Identifier | No | Reference frame within which the direction is expressed. Must be a celestial body or spacecraft reference frame (see 4.4.2). |
| units | MAL::String | Yes | The units for the quantity of angle, in which the azimuth and elevation are expressed in.  Default = ‘deg’. |

##### Data Type: RADecDirection

###### Overview

Represents a Direction based on celestial angular coordinates of right ascension and declination.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **RADecDirection** | Extends | Direction | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| ra | MAL::Double | No | Right Ascension: Celestial angular coordinate, measured eastward along the celestial equator. |
| dec | MAL::Double | No | Declination: Celestial angular coordinate, north or south of the celestial equator. |
| frame | MAL::Identifier | No | Reference frame within which the direction is expressed. Must be a quasi-inertial celestial body or orbit-related frame (see 4.4.2). |
| units | MAL::String | Yes | The units for the quantity of angle, in which the ra and dec are expressed in.  Default = ‘deg’. |

##### Data Type: NamedTargetDirection

###### Overview

A NamedTargetDirection is a Direction that points to an existing object. The manner in which the planning system derives the value of this Direction from the name of the referenced object is implementation-defined.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **NamedTargetDirection** | Extends | Direction | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| namedTarget | MAL::Identifier | No | Name or identifier of a catalogued celestial object or a mission specific object (see 4.4.3). |

##### Data Type: DirectionReference

###### Overview

A DirectionReference is a Direction that may be computed following some mission specific definition.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **DirectionReference** | Extends | Direction | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| reference | MAL::Identifier | No | Name of a mission specific direction definition. |

###### Requirements

The reference field shall be used to distinguish between different possible mission specific directions.

#### Physical Value Data Types

##### Data Type: PhysicalValue

###### Overview

PhysicalValue is an abstract base type for the specific value types defined below. Only specific value types are used in the pointing constraint definitions below.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PhysicalValue** | Extends | MAL::Composite | SFP | Abstract |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| value | MAL::Double | No | Physical value. |
| units | MAL::String | Yes | Optional units. The units for a single quantity. The unit type depends on the specific value type. |

##### Data Type: Angle

###### Overview

Physical value with units of type Angle.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **Angle** | Extends | PhysicalValue | SFP |  |

##### Data Type: AngularVelocity

###### Overview

Physical value with units of type AngularVelocity.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **AngularVelocity** | Extends | PhysicalValue | SFP |  |

##### Data Type: Distance

###### Overview

Physical value with units of type Distance.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **Distance** | Extends | PhysicalValue | SFP |  |

#### 

### Expressions

#### Data Type: Expression

##### Overview

When entering MPS data, it is often not possible to provide a concrete value for a required field of some type T. Instead, a field may be defined as type Expression <T>. This denotes that the field can have not just a value of type T, but may also be of some other type that can be evaluated by the planning system to a value of type T.

##### Requirements

Expressions of a given type T shall be denoted using the notation Expression <T>, where the data type T is restricted to be any defined ArgType (see 4.6.1.2), which may be any MAL Attribute type, Position, or Direction.

In such cases, the underlying type shall be a MAL::Element that must evaluate to type T.

NOTE – The manner in which this evaluation occurs is implementation defined.

Where an expression of type Expression <T> is expected, implementations shall at minimum support values of type T and of type ExternalExpression.

When an operation receives a value for an Expression <T> that does not evaluate to the expected type T, an INVALID error shall be returned. This error shall have INCONSISTENT as secondary error code.

#### Data Type: ExternalExpression

##### Overview

When the MPS data types are not sufficiently expressive, it is possible to provide an external expression that evaluates into a given data type, using the ExternalExpression data type. These external expressions are themselves text strings in some external language.

The manner in which this expression is evaluated is implementation specific.

##### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ExternalExpression** | Extends | MAL::Composite | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| type | ArgTypeEnum | No | Enumeration specifying the data type of the result of the expression. |
| expressionLanguage | MAL::String | No | Defines the expression language used to specify the expression. |
| expression | MAL::String | No | The text of the expression. |

### Arguments

#### Overview

The instance objects of several MPS service objects have associated arguments that can be used to parameterize the object. The set of arguments is defined in the associated definition object, with the argument values forming part of the instance object. Arguments apply to ActivityInstance, EventInstance, RequestInstance and Functions.

#### General

##### Data Type: ArgDef

###### Overview

The definition of an argument is an ArgDef, a set of which may be contained within the definition MO object of a planning event, planning activity, or planning request. This defines the name and data type of the argument. Depending on the data type, the ArgDef may require additional type specific fields to support data validation.  SubTypes are identified for Numeric, String, and Status arguments.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ArgDef** | Extends | MAL::Composite | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| argName | MAL::Identifier | No | Name of the argument. |
| description | MAL::String | No | Extended description of the argument. |
| argType | ArgTypeEnum | No | Enumeration specifying the data type of the argument. |
| argUnits | MAL::String | Yes | The units of a single quantity, in which the argument value is expressed in. |
| isArray | MAL::Boolean | No | If True, indicates that the argument is an array of values of type ArgTypeEnum. |
| validationData | ValidationDetails | Yes | Optional. Specifies the allowed range of values for the Argument, with concrete subtypes specific to the data type of the Argument. |

###### Requirements

If the argument is an array, then all values of the array shall be of the same type, as defined in argType.

##### Data Type: Argument

###### Overview

The instance of an argument is an Argument, a set of which may be contained within the instance MO object of a planning event or planning activity or within a planning request. This comprises the name and value of the argument, corresponding to the set of arguments defined in the ArgDef. Argument values are represented as a MAL Element of appropriate data type.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **Argument** | Extends | MAL::Composite | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| argName | MAL::Identifier | No | Name of the argument. |
| argValues | List <MAL::Element> | No | Argument value (or values if it is an array). The MAL Element subtype(s) must match the argument type supplied in the corresponding ArgDef. |

###### Requirements

If the argument is an array (count > 1) then all values of the array shall be of the same type, as defined by the argType of the associated ArgDef.

##### Data Type: ArgSpec

###### Overview

In the case of the planning activity, there is also an ArgSpec, a set of which may be contained within the ActivityDetails structure embedded within a planning request or parent planning activity definition. The ArgSpec defines how to derive the value of an Argument when instantiating it at run-time.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ArgSpec** | Extends | MAL::Composite | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| argName | MAL::Identifier | No | Name of the argument. |
| argSpecs | List <Expression <MAL::Element>> | No | Expression that can be evaluated at run-time to provide argument value(s) of appropriate data type. |

###### Requirements

The argSpec field is an Expression, the result of which shall be a value matching the defined ArgType.

#### Validation Details

##### Data Type: ValidationDetails

###### Overview

Abstract type that is used to represent an allowed range of values for a given Argument or Resource.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ValidationDetails** | Extends | MAL::Composite | SFP | Abstract |

##### Data Type: NumericRange

###### Overview

Concrete sub-type of *ValidationDetails* that provides additional fields to support data validation for numeric data types.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **NumericRange** | Extends | ValidationDetails | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| min | MAL::Double | Yes | Minimum value of the argument; if omitted, no minimum value is considered. |
| max | MAL::Double | Yes | Maximum value of the argument; if omitted, no maximum value is considered. |

##### Data Type: StringPattern

###### Overview

Concrete sub-type of *ValidationDetails* that provides additional fields to support data validation for the string data type.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **StringPattern** | Extends | ValidationDetails | SFP |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Type | Nullable | Description |
| maxLength | MAL::UInteger | Yes | Maximum length of the string (characters). If omitted, no maximum length is enforced. |
| regex | MAL::String | Yes | A ‘regular expression’ or sequence of characters defining a character pattern that the string value must match. If omitted, all character sequences are permitted.  The choice of ‘regular expression’ specification to follow is implementation-specific. |

##### Data Type: StatusValues

###### Overview

Concrete sub-type of *ValidationDetails* that provides additional fields to support data validation and interpretation for integer type arguments that are effectively enumerated Statuses.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **StatusValues** | Extends | ValidationDetails | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| allowedValues | List <StateDef> | No | Set of allowed State definitions (see 0), comprising the enumerated value and an associated text label. |

### Constraints

#### General

##### Data Type: Constraint

###### Overview

Abstract type representing a **planning constraint**, a Boolean condition which restricts the planning of **planning activities**.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **Constraint** | **Extends** | MAL::Composite | **SFP** | Abstract |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| negate | MAL::Boolean | Yes | Specifies whether the result of combining the Constraints is to be inverted (NOT function).  Default = False. |

##### Data Type: ConstraintNode

###### Overview

Multiple planning constraints can be combined using a ConstraintNode. The ConstraintNode specifies the logical operation (AND or OR) to be used when combining a set of constraints together. As the ConstraintNode is itself defined as a sub-type of Constraint, it is possible to construct a tree of ConstraintNodes using different logical operators.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ConstraintNode** | Extends | Constraint | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| operator | LogicOpEnum | Yes | Enumeration specifying the logic for combining multiple Boolean conditions together. One of {AND, OR}.  Default = AND. |
| constraints | List <Constraint> | No | The set of Constraints to be combined. Must contain at least one element. |

##### Data Type: LogicOpEnum

###### Overview

A LogicOpEnum represents the type of logic used to combine two Boolean conditions.

###### Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Name | **LogicOpEnum** | SFP |  |

|  |  |  |
| --- | --- | --- |
| Enumeration | Value | Description |
| AND | 1 | Logical AND |
| OR | 2 | Logical OR |

##### Data Type: Constraint Expression

###### Overview

All types of constraint can be considered conditions that are either met or not met when a planning activity is placed in a Plan. They can therefore be specified as a potentially complex Boolean expression that combines references to the arguments and fields of objects in the MPS information model using operators of various types (arithmetic, comparative, logical, string, temporal, and geometric). The expression must evaluate to TRUE for the constraint to be met.

As introduced in 1.2, this Recommended Standard does not define a full expression language capable of supporting such complex Boolean expressions. It does, however, support the use of externally defined expression languages. The ConstraintExpression type allows for the use of such an expression language to define any type of constraint, providing communicating entities all have the capability to evaluate that expression language.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ConstraintExpression** | Extends | Constraint | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| constraint | Expression <MAL::Boolean> | No | Potentially complex conditional expression that must evaluate to TRUE for the constraint to be met. |

#### Temporal Constraints [Optional]



##### Data Type: TimeConstraint

###### Overview

The time at which a **planning activity** must be planned.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **TimeConstraint** | Extends | Constraint | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| time | Expression <MAL::Time> | No | The time at which the planning activity must be planned. |
| timeRef | Slider | No | The point in the duration of the planning activity that is time constrained.  0: the start of the planning activity.  1: the end of the planning activity. |

##### Data Type: TimeWindowConstraint

###### Overview

A time window within which the **planning activity** is to be planned.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **TimeWindowConstraint** | Extends | Constraint | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| startRef | Slider | Yes | The point in the duration of the activity that is constrained to be after the start time of the time window. Although typically the start of the activity (0), this can be any point up to the end of the activity (1).  Default is the start of the planning activity. |
| endRef | Slider | Yes | The point in the duration of the activity that is constrained to be before the end time of the time window. Although typically the end of the activity (1), this can be any point up to the start of the activity (0).  Default is the end of the planning activity. |
| timeWindows | List <TimeWindow> | No | The [set of] TimeWindows within which the activity must be placed on the Plan. |

##### Data Type: DurationConstraint

###### Overview

A DurationConstraint restricts the duration of a **planning activity** within the plan.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **DurationConstraint** | Extends | Constraint | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| minDuration | Expression <MAL::Duration> | Yes | Specifies the minimum duration of the planning activity.  If omitted, a value of 0 is used. |
| maxDuration | Expression <MAL::Duration> | Yes | Specifies the maximum duration of the planning activity.  If omitted, the maximum representable MAL::Duration value is assumed. |

#### Resource and Argument Constraints [Optional]

##### Data Type: ArgumentConstraint

###### Overview

An argument constraint may be associated with a planning activity to restrict when it can be planned, based on the value of an argument of the planning activity itself or a related planning activity or event.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ArgumentConstraint** | Extends | Constraint | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| objectRef | Expression <MAL::ObjectRef <MAL::Element>> | Yes | Object Type: ActivityDefinition | EventDefinition  Specifies the definition (class) of the planning activity or planning event whose argument is to be referenced. If omitted the activity containing the constraint is assumed. |
| argName | MAL::Identifier | No | Identifies the specific argument of the referenced Object whose value is to be compared |
| comparator | ExpressionOperatorEnum | No | Comparison operator, which may be one of:  =, !=, >, >=, <, <=, contains, icontains  The contains operator only applies to strings and may be case sensitive or insensitive. |
| value | MAL::Element | No | Value (of same type as the referenced Argument) to be compared against. |

##### Data Type: ExpressionOperatorEnum

###### Overview

Whenever a value comparison is needed in this standard, multiple Boolean operations may be chosen from. These operations are described by each of the possible enumeration values of ExpressionOperatorEnum.

###### Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Name | **ExpressionOperatorEnum** | SFP |  |

|  |  |  |
| --- | --- | --- |
| Enumeration | Value | Description |
| EQUAL | 1 | = |
| DIFFER | 2 | != |
| GREATER | 3 | > |
| GREATER\_OR\_EQUAL | 4 | >= |
| LESS | 5 | < |
| LESS\_OR\_EQUAL | 6 | <= |
| CONTAINS | 7 | Case sensitive containment (Strings only). |
| ICONTAINS | 8 | Case insensitive containment (Strings only). |

###### Requirements

The following sets of data types shall support one or more operations:

* **Binary values:** MAL::Blob and MAL::Boolean.
* **Integers:** MAL::Octet, MAL::UOctet, MAL::Short, MAL::UShort, MAL::Integer, MAL::UInteger, MAL::Long and MAL::ULong.
* **Floating-point numbers:** MAL::Float and MAL::Double.
* **Unicode strings:** MAL::String, MAL::Identifier, and MAL::URI.
* **Time representations:** MAL::Time and MAL::FineTime.
* **Geometric types:** *Position* and *Direction*.

The following classes of operations shall be distinguished:

* **Equality:** EQUAL and DIFFER.
* **Ordering:** GREATER, GREATER\_OR\_EQUAL, LESS, and LESS\_OR\_EQUAL.
* **Containment:** CONTAINS and ICONTAINS.

The precise semantics for each supported combination of data types and operations are given in the table below.

Table 4‑5 : Supported Expression Operations per Data Type

| Data Types | Supported operations | Semantics |
| --- | --- | --- |
| Binary values | Equality | Bitwise equality. |
| Integers | Equality | Integer equality. |
| Ordering | Integer ordering. |
| Floating-point numbers | Equality | Follows IEEE 754 comparison predicates (reference [8]). Implementations may opt for either signaling or quiet comparison. |
| Ordering | Follows IEEE 754 comparison predicates (reference [8]). Implementations may opt for either signaling or quiet comparison. |
| Unicode strings | Equality | Case-sensitive equivalence. Implementations may opt to apply Unicode normalization (reference [9]) when checking for equivalence. |
| Ordering | Respects lexical ordering based on Unicode collation (see reference [9]). The locale used to determine this ordering is implementation specific. |
| Containment | Whether the given string is equivalent to any of the substrings of the value being compared to. Implementations may opt to apply Unicode normalization (reference [9]) when checking for equivalence. |
| Time representations | Equality | Two times are equal when they encode the same time point. |
| Ordering | Greater-than implies encoding a time point later in time, and vice versa. |
| Geometric types | Equality | Equal when all individual coordinates are equal following floating-point comparison, given the same coordinate frame and units.  Implementations may additionally support equivalence across difference frames and units in an implementation-specific manner. |

##### Data Type: ResourceConstraint

###### Overview

ResourceConstraint is an abstract type that represents a constraint expressed in terms of the value of a given Resource.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ResourceConstraint** | Extends | Constraint | SFP | Abstract |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| resourceRef | MAL::ObjectRef <Resource> | No | Identifies the planning resource that is constrained for the duration of the planning activity. |
| comparator | ExpressionOperatorEnum | No | Comparison operator, which may be one of:  =, !=, >, >=, <, <=, contains, icontains.  The contains operator only applies to strings and may be case sensitive or insensitive. |

##### Data Type: SimpleResourceConstraint

###### Overview

The simple resource constraint must be satisfied for the duration of the referenced planning activity.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **SimpleResourceConstraint** | Extends | ResourceConstraint | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| value | MAL::Attribute | No | Value (of same type as the referenced Resource) to be compared against. |

##### Data Type: ComplexResourceConstraint

###### Overview

In the [simple] resource constraint, the value of the referenced planning resource is constrained against a single value for the entire duration of the referenced planning activity.

With the complex resource constraint, the period over which the constraint applies can be customized relative to the referenced planning activity; and the value against which the referenced planning resource is constrained can be specified as a relative resource profile which evolves over time.

The fields of the complex resource constraint extend or modify those of the [simple] resource constraint as given below.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ComplexResourceConstraint** | Extends | ResourceConstraint | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| startRef | Slider | Yes | Identifies the point in the duration of the referenced planning activity to which the start of the constraint period relates.  Default is the start of the **planning activity**. |
| endRef | Slider | Yes | Identifies the point in the duration of the referenced planning activity to which the end of the constraint period relates.  Default is the end of the **planning activity**. |
| startOffset | Expression <MAL::Duration> | Yes | Offset from startRef that specifies the start of the constraint period. A positive offset implies a shift later in time.  Default is no offset. |
| endOffset | Expression <MAL::Duration> | Yes | Offset from endRef that specifies the end of the constraint period. A positive offset implies a shift later in time.  Default is no offset. |
| valueProfile | RelativeResourceProfile | No | ResourceProfile specifying an evolving value over time against which the value of the planning resource is to be compared (see 4.5.4.4). |

#### Geometric Constraints [Optional]

##### Data Type: GeometricConstraint

###### Overview

Geometric constraints restrict the planning of the **planning activity** by imposing a geometric condition that must be valid during some constraint period.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **GeometricConstraint** | Extends | Constraint | SFP | Abstract |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| startRef | Slider | Yes | Identifies the point in the duration of the referenced planning activity to which the start of the constraint period relates.  Default is the start of the **planning activity**. |
| endRef | Slider | Yes | Identifies the point in the duration of the referenced planning activity to which the end of the constraint period relates.  Default is the end of the **planning activity**. |
| startOffset | Expression <MAL::Duration> | Yes | Offset from startRef that specifies the start of the constraint period. A positive offset implies a shift later in time.  Default is no offset. |
| endOffset | Expression <MAL::Duration> | Yes | Offset from endRef that specifies the end of the constraint period. A positive offset implies a shift later in time.  Default is no offset. |

##### Data Type: PositionConstraint

###### Overview

Sub-type of geometric constraint expressed in terms of a specified Position and a tolerance. The tolerance is defined as a sphere around the specified position, expressed as a distance or angle. It should be noted that the position itself can be expressed using any of the concrete position sub-types, including orbital and surface positions. The use of a constraint expressed by OrbitalPosition is particularly relevant for Earth observation satellites with a repetitive ground track and on-board position-based planning function. The position can also specified as an expression.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PositionConstraint** | Extends | GeometricConstraint | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| position | Expression <Position> | No | Specifies the required position expressed using any concrete position type. |
| tolerance | Expression <MAL::Double> | No | Specifies the maximum distance or angle from the required position that satisfies the constraint, effectively defining a sphere around the required position. |
| units | MAL::String | Yes | Optional. The tolerance units name, either for the quantity of distance or the quantity of angle.  Default = ‘km’, but ‘deg’ is more relevant for an OrbitalPosition. |

##### Data Type: EllipsoidalPositionConstraint

###### Overview

Geometric constraint that specifies a Position with a tolerance given in terms of a frame-aligned ellipsoid. The frame within which this tolerance ellipsoid is expressed may be different from the frame in which the required position is expressed.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **EllipsoidalPositionConstraint** | Extends | GeometricConstraint | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| position | Expression <Position> | No | Specifies the required position expressed using any concrete position type. |
| x | Expression <MAL::Double> | No | Length of the ellipsoid axis that is aligned with the x axis of the specified frame. |
| y | Expression <MAL::Double> | No | Length of the ellipsoid axis that is aligned with the y axis of the specified frame. |
| z | Expression <MAL::Double> | No | Length of the ellipsoid axis that is aligned with the z axis of the specified frame. |
| frame | MAL::Identifier | No | Reference frame with which the axes of the tolerance ellipsoid are aligned (see 4.4.2). |
| units | MAL::String | Yes | Optional. The tolerance units name, either for the quantity of distance or the quantity of angle.  Default = ‘km’, but ‘deg’ is more relevant for an OrbitalPosition. |

###### Requirements

This constraint shall be satisfied when the position falls within the tolerance ellipsoid that is centered around the required position.

##### Data Type: PointingConstraint

###### Overview

Pointing constraints impose a restriction on a planning activity appearing in a Plan, based on the pointing direction of a physical object, such as a spacecraft or instrument.

As with the Direction data types (see 4.6.3.3), pointing constraints are consistent with the pointing templates defined for use within CCSDS Navigation data format Recommended Standards, and specifically the Pointing Request Message (PRM) (reference [10]). PointingConstraint is a concrete sub-type of *GeometricConstraint* that includes fields common to all pointing templates. The pointing template itself is then identified as a field and any additional arguments applicable to the template are provided as a list of name-value pairs.

NOTE – Pointing templates are common, generic templates that describe pointing modes that may be followed by spacecraft.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PointingConstraint** | Extends | GeometricConstraint | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| pointingFrame | MAL::Identifier | Yes | Optional. Reference frame to which the pointing constraint applies (see 4.4.2).  Default frame is the spacecraft frame or any other mission specific default frame. |
| boresight | Direction | No | Direction in any spacecraft frame. |
| boresightMargin | Angle | Yes | Defines an optional cone region around the boresight, allowing a margin for application of the pointing constraint.  Default = 0.0. |
| phaseAngleMargin | Angle | Yes | Defines an optional rotation around the boresight, w.r.t. the default phase angle, allowing a margin for application of the pointing constraint.  Default = 0.0. |
| unconstrainedPhaseAngle | MAL::Boolean | Yes | If TRUE no constraint will apply to the phaseAngle. The phaseAngleMargin field will be ignored in this case.  Default = FALSE. |
| pointingTemplate | MAL::Identifier | No | One of the pointing templates defined in the PRM or a mission specific pointing template (see 4.4.4). |
| pointingArguments | List <NamedElement> | Yes | The argument list is consistent with the referenced template by name. |

###### Requirements

The following table prescribes the currently defined pointing templates and their additional arguments (where applicable).

Table 4‑6 : Predefined Pointing Templates

| Pointing Template | Arguments | Type |
| --- | --- | --- |
| Inertial Pointing | target  phaseAngle  offsetAngle  angularRate | Direction  Angle  Angle  AngularVelocity |
| Sun Pointing | phaseAngle  offsetAngle  angularRate | Angle  Angle  AngularVelocity |
| Track with Inertial Direction Yaw Steering | targetBody  phaseAngle | Position  Angle |
| Track with Power Optimized Yaw Steering | targetBody | Position |
| Nadir with Power Optimized Yaw Steering |  |  |
| Nadir with Ground Track Aligned Yaw Steering |  |  |
| Nadir with Orbital Pole Aligned Yaw Steering |  |  |
| Limb Pointing with Power Optimized Yaw Steering | surface  dirVector  height | MAL::String  Direction  Distance |
| Limb Pointing with Inertial Direction Yaw Steering | surface  dirVector  height  phaseAngle | MAL::String  Direction  Distance  Angle |
| Velocity Pointing with Orbital Pole Yaw Steering | phaseAngle | Angle |

NOTES

1. Following reference [10], some pointing template arguments are optional; in those cases, the argument may be omitted from the passed pointing arguments list.
2. What are referred to as ‘arguments’ in the table below correspond one-to-one with the elements that are indicated as ‘parameters’ in reference [10].

##### Data Type: RevolutionConstraint

###### Overview

Specifies a range of revolution angles for a rotating spacecraft.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **RevolutionConstraint** | Extends | GeometricConstraint | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| revolutionAngle | Expression <MAL::Double> | No | Angle of revolution. |
| tolerance | Expression <MAL::Double> | No | Tolerance in the angle of revolution. |
| units | MAL::String | Yes | Optional. The units for the quantity of angle, in which the revolutionAngle and tolerance are expressed in.  Default = ‘deg’. |

##### Data Type: DistanceConstraint

###### Overview

Specifies a range of distances between two physical objects (the observer and the target).

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **DistanceConstraint** | Extends | GeometricConstraint | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| observer | Expression <Position> | No | Position of the observer [Object1]. |
| target | Expression <Position> | No | Position of the target [Object2]. |
| minDistance | Expression <MAL::Double> | No | Minimum distance between observer and target. |
| maxDistance | Expression <MAL::Double> | No | Maximum distance between observer and target. |
| units | MAL::String | Yes | Optional. The units for the quantity of distance.  Default = ‘km’. |

##### Data Type: AngleConstraint

###### Overview

Specifies a range of values for the angle subtended between three physical objects.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **AngleConstraint** | Extends | GeometricConstraint | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| centreObject | Expression <Position> | No | Position of the centre object. |
| targetObject1 | Expression <Position> | No | Position of target object 1. |
| targetObject2 | Expression <Position> | No | Position of target object 2. |
| minAngle | Expression <MAL::Double> | No | Minimum angle subtended at the centre object by target objects 1 and 2. |
| maxAngle | Expression <MAL::Double> | No | Maximum angle subtended at the centre object by target objects 1 and 2. |
| units | MAL::String | Yes | Optional. The units for the quantity of angle, in which the minAngle and maxAngle are expressed in.  Default = ‘deg’. |

###### Requirements

The constrained angle shall be that subtended at the centre object by target objects 1 and 2.

#### Other Constraints [Optional]

##### Data Type: SequentialConstraint

###### Overview

Sequential constraints impose a restriction on the order of planning activities in a Plan with respect to other planning activities and planning events.

Two objects are identified: the parent activity and its opponent. The parent activity is the activity for which the constraint is defined. The opponent may be either a **planning activity** or a **planning event** and must be placed in the Plan relative to the parent activity.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **SequentialConstraint** | Extends | Constraint | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| opponent | Expression <MAL::ObjectRef <MAL::Element>> | No | Object Type: ActivityDefinition | EventDefinition  Specifies the definition (class) of the opponent planning activity or planning event. |
| parentRef | Slider | Yes | Point on the parent activity that must be followed by the opponent.  Default = 1. |
| opponentRef | Slider | Yes | Point on the opponent that must follow the parent activity.  Default = 0. |
| minOffset | Expression <MAL::Duration> | Yes | Minimum offset between the specified points on the parent activity and the opponent.  Default is no offset. |
| maxOffset | Expression <MAL::Duration> | Yes | Maximum offset between the specified points on the parent activity and the opponent.  Default is no offset. |

##### Data Type: SeparationConstraint

###### Overview

A separation constraint specifies that the parent **planning activity** (the activity for which the constraint is defined) must be separate in time from another **planning activity** or **planning event**, the opponent.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **SeparationConstraint** | Extends | Constraint | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| opponent | Expression  <MAL::ObjectRef <MAL::Element>> | No | Object Type: ActivityDefinition | EventDefinition  Specifies the definition (class) of the opponent planning activity or planning event. |
| startRef | Slider | Yes | Identifies the point in the duration of the parent activity with respect to which to place the start of its constraint window.  Default = 0. |
| startOffset | Expression <MAL::Duration> | Yes | Offset with respect to startRef of the start of the parent activity constraint window.  Default is no offset. |
| endRef | Slider | Yes | Identifies the point in the duration of the parent activity with respect to which to place the end of its constraint window.  Default = 1. |
| endOffset | Expression <MAL::Duration> | Yes | Offset with respect to endRef of the end of the parent activity constraint window.  Default is no offset. |
| opponentStartRef | Slider | Yes | Identifies the point in the duration of the opponent with respect to which to place the start of its constraint window.  Default = 0. |
| opponentStartOffset | Expression <MAL::Duration> | Yes | Offset with respect to opponentStartRef of the start of the opponent constraint window.  Default is no offset. |
| opponentEndRef | Slider | Yes | Identifies the point in the duration of the opponent with respect to which to place the end of its constraint window.  Default = 1. |
| opponentEndOffset | Expression <MAL::Duration> | Yes | Offset with respect to opponentEndRef of the end of the opponent constraint window.  Default is no offset. |

###### Requirements

This constraint shall be fulfilled if two constraint windows, placed relative in time to the parent activity and to its opponent, do not overlap.

Objects are specified by definition [class] rather than instance. The separation condition shall apply to all instances of the class.

##### Data Type: FunctionConstraint

###### Overview

Function constraints make use of an external custom function to determine whether or not a constraint is satisfied.

As for complex resource constraints, the period over which the function constraint applies is specified relative to the referenced planning activity.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **FunctionConstraint** | Extends | Constraint | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| startRef | Slider | Yes | Identifies the point in the duration of the referenced planning activity to which the start of the constraint period relates.  Default is the start of the **planning activity**. |
| endRef | Slider | Yes | Identifies the point in the duration of the referenced planning activity to which the end of the constraint period relates.  Default is the end of the planning activity. |
| startOffset | Expression <MAL::Duration> | Yes | Offset from startRef that specifies the start of the constraint period. A positive offset implies a shift later in time.  Default is no offset. |
| endOffset | Expression <MAL::Duration> | Yes | Offset from endRef that specifies the end of the constraint period. A positive offset implies a shift later in time.  Default is no offset. |
| function | FunctionDetails | No | Specifies the Function to be applied and its set of input arguments. |

###### Requirements

Available functions shall be pre-defined (see 4.5.7.1.3) to allow these to be referenced in a function constraint.

### Effects [Optional]

#### Data Type: Effect

##### Overview

An Effect is an abstract type that may be used to represent the impact that executing a **planning activity** will have on a **planning resource**.

##### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **Effect** | Extends | MAL::Composite | SFP | Abstract |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| resourceRef | MAL::ObjectRef <Resource> | No | Identifies the planning resource that is constrained for the duration of the planning activity. |

#### Data Type: SimpleEffect

##### Overview

A simple effect applies the defined operation on the specified planning resource at the time relative to the planning activity defined by timeRef+timeOffset.

##### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **SimpleEffect** | **Extends** | Effect | SFP |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Type | Nullable | Description |
| timeRef | Slider | No | The point in the duration of the planning activity to which the time of the Effect is relative.  0: the start of the planning activity  1: the end of the planning activity |
| timeOffset | Expression <MAL::Duration> | Yes | Offset from timeRef that specifies the time at which the Effect is to be applied.  Default is no offset. |
| operator | EffectOperationEnum | No | Operation to be performed on the planning resource. One of: SET, INCREASE, DECREASE.  Increase and decrease are only applicable to numeric data types. |
| value | MAL::Attribute | No | The value that the planning resource is to be set to if the Effect operator is SET; or to be increased/decreased by if it is INCREASE or DECREASE. |

#### Data Type: EffectOperationEnum

##### Overview

An EffectOperationEnum is used to denote the specific type of change made to a **planning resource** for a given Effect.

##### Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Name | **EffectOperationEnum** | SFP |  |

| Status | Value | Description |
| --- | --- | --- |
| SET | 1 | Set to specified value |
| INCREASE | 2 | Increase by specified value |
| DECREASE | 3 | Decrease by specified value |

#### Data Type: ComplexEffect

##### Overview

In the simple effect, the value of the impacted planning resource is set to the specified value at a single point in time.

With the complex effect, the value of the impacted planning resource can be evolved over a specified time period in accordance with a defined RelativeResourceProfile.

##### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **ComplexEffect** | Extends | Effect | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| startRef | Slider | Yes | Identifies the point in the duration of the referenced planning activity to which the start of the effect period relates.  Default is the start of the **planning activity**. |
| endRef | Slider | Yes | Identifies the point in the duration of the referenced planning activity to which the end of the effect period relates.  Default is the end of the **planning activity**. |
| startOffset | Expression <MAL::Duration> | Yes | Offset from startRef that specifies the start of the effect period. A positive offset implies a shift later in time.  Default is no offset. |
| endOffset | Expression <MAL::Duration> | Yes | Offset from endRef that specifies the end of the effect period. A positive offset implies a shift later in time.  Default is no offset. |
| operator | EffectOperationEnum | No | Operation to be performed on the planning resource. One of: SET, INCREASE, DECREASE.  Increase and decrease are only applicable to numeric data types. |
| value | RelativeResourceProfile | No | Resource profile specifying an evolving value to which the value of the planning resource is to be set if the Effect operator is SET; or to be increased/decreased by if it is INCREASE or DECREASE (see 4.5.4.4). |

### Triggers

#### Data Type: Trigger

##### Overview

All sub-classes of Trigger include the time at which they are predicted to occur (in advance of execution); and, where applicable, the time at which they actually occurred (post execution).

##### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **Trigger** | Extends | MAL::Composite | SFP | Abstract |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| time | MAL::Time | No | Predicted or actual time of Trigger. The predicted time may evolve during the planning process up to the time of execution. The actual time is only available post execution, and hence can only be provided by a plan execution function. |

#### Data Type: TimeTrigger

##### Overview

Sub-type of Trigger based on time. The trigger time is the specified constraint, and will usually match the predicted time on the base class during the planning process, but the actual time could still be slightly different post-execution.

##### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **TimeTrigger** | Extends | Trigger | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| triggerTime | MAL::Time | No | Planned time of Trigger. |

#### Data Type: PositionTrigger [Optional]

##### Overview

Sub-type of Trigger based on position. Depending on the coordinate type of position used, a margin may be specified in terms of distance from the specified position.

##### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PositionTrigger** | Extends | Trigger | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| triggerPosition | Position | No | Planned position of Trigger. |
| distanceMargin | Distance | Yes | Defines a sphere around the trigger position within which a position is considered to meet the trigger condition. |

##### Requirements

No margin shall be present for an OrbitalPosition, as an orbiting spacecraft would pass through the specified angle.

#### Data Type: DirectionTrigger [Optional]

##### Overview

Sub-type of Trigger based on pointing. Depending on the coordinate type of direction used, a margin may be specified in terms of angle from the specified direction.

##### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **DirectionTrigger** | Extends | Trigger | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| triggerDirection | Direction | No | Planned direction of Trigger. |
| angleMargin | Angle | Yes | Defines a circle around the trigger direction within which a direction is considered to meet the trigger condition. |

##### Requirements

No margin shall be present for a RevolutionDirection, as a rotating spacecraft or instrument would pass through the specified angle.

#### Data Type: AngleTrigger [Optional]

##### Overview

Sub-type of Trigger based on the angle subtended between three physical objects.

##### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **AngleTrigger** | Extends | Trigger | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| centreObject | Position | No | Position of the centre object. The trigger angle is that subtended at the centre object by target objects 1 and 2. |
| targetObject1 | Position | No | Position of target object 1. |
| targetObject2 | Position | No | Position of target object 2. |
| minAngle | Angle | No | Minimum angle subtended at the centre object by target objects 1 and 2. |
| maxAngle | Angle | No | Maximum angle subtended at the centre object by target objects 1 and 2. |

##### Requirements

The trigger angle shall be that subtended at the centre object by target objects 1 and 2.

The trigger shall occur when the trigger angle is between the minimum and maximum angle.

#### Data Type: EventTrigger

##### Overview

Sub-type of Trigger based on planning event.

##### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **EventTrigger** | Extends | Trigger | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| triggerEvent | MAL::ObjectRef <EventInstance> | No | Reference to an EventInstance |
| timeOffset | MAL::Duration | No | Time offset from the EventInstance |

### Repetitions

#### General

##### Data Type: Repetition

###### Overview

A repetition is used to specify the repeated instantiation of a [set of] planning activities. Multiple sub-types of Repetition are defined to support the specification of repeat cycles by different criteria. It can be used in the context of a planning request to specify a standing order for repeated execution of the [set of] planning activities.

In the context of an ActivityNode embedded within a planning request (see 4.5.2.3), it is possible to nest one Repetition inside another, enabling the specification of complex repetitive sequences of activities.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **Repetition** | Extends | MAL::Composite | SFP | Abstract |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| count | MAL::Integer | Yes | Maximum number of repeat cycles/instances [optional].  If not specified there is no limit to the number of repetitions. |
| timeWindow | TimeWindow | Yes | Time period over which the repetition is applicable [optional].  If not specified repetition continues indefinitely. |
| separationType | SeparationTypeEnum | No | Specifies whether the repetition interval is **Relative** to the previous occurrence, or **Absolute** for all occurrences. |

##### Data Type: SeparationTypeEnum

###### Overview

A SeparationTypeEnum is used to define whether the separation between repetitions is relative or absolute.

###### Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Name | **SeparationTypeEnum** | SFP |  |

| Enumeration | Value | Description |
| --- | --- | --- |
| RELATIVE | 1 | Tolerance on separation is only considered between any two occurrences. |
| ABSOLUTE | 2 | Tolerance on separation applies to a multiple of the separation from the initial occurrence. |

#### Location Repetitions [Optional]



##### Data Type: PositionRepetition

###### Overview

A sub-type of Repetition that starts at a given Position and repeats based on separation from each subsequent occurrence.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **PositionRepetition** | Extends | Repetition | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| initialPosition | Expression <Position> | No | Nominal position of first occurrence. |
| repetitionDirection | Expression <Direction> | No | Direction of repetition. |
| separation | Expression <MAL::Double> | No | The required Distance between occurrences. |
| tolerance | Expression <MAL::Double> | No | The allowed tolerance (+/-) in the required distance between occurrences, the interpretation of which is dependent on the separationType. |
| units | MAL::String | Yes | Optional. The units for the quantity of distance, in which the separation and tolerance are expressed in.  Default = ‘km’. |

##### Data Type: OrbitRepetition

###### Overview

A sub-type of Repetition based on the orbital cycle.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **OrbitRepetition** | Extends | Repetition | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| orbitNumber | Expression <MAL::Long> | No | Orbit number for the first occurrence. Depending on the relativeOrbit flag, the orbit number may be absolute (since start of mission) or relative (to the orbital repeat cycle).  The datum with respect to which the orbit number is counted is mission-specific. |
| relativeOrbit | MAL::Boolean | No | Flag indicating if the orbit number is absolute or relative to the orbital repeat cycle. |
| orbitSeparation | Expression <MAL::Long> | No | The required number of orbits separation between occurrences. If orbitNumber is Relative and the required repetition is once per repeat cycle, this is the number of orbits in the repeat cycle, but the value 0 may also be used. |
| angleSeparation | Expression <MAL::Double> | No | The required angular separation between occurrences. This allows for multiple repetitions within an orbit. The value 0 indicates only one occurrence within the orbit. |
| orbitAngle | Expression <MAL::Double> | No | The required position of the first occurrence within the orbit expressed as an angle. |
| tolerance | Expression <MAL::Double> | No | The allowed tolerance (+/-) in the required orbital angle. |
| units | MAL::String | Yes | Optional. The units for the quantity of angle, in which the orbitAngle, angularSeparation, and tolerance are expressed in.  Default = ‘deg’. |

#### Pointing Repetitions [Optional]



##### Data Type: DirectionRepetition

###### Overview

A sub-type of Repetition based on direction, which supports the specification of astronomical surveys.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **DirectionRepetition** | Extends | Repetition | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| initialDirection | Expression <Direction> | No | Nominal direction of first occurrence. |
| targetDirection | Expression <Direction> | No | Specifies the direction of repetition as line connecting the initial and target directions. |
| separation | Expression <MAL::Double> | No | The required angle between occurrences. |
| tolerance | Expression <MAL::Double> | No | The allowed tolerance (+/-) in the required angle between occurrences, the interpretation of which is dependent on the separationType. |
| units | MAL::String | Yes | Optional. The units for the quantity of angle, in which the separation and tolerance are expressed in.  Default = ‘deg’. |

##### Data Type: RevolutionRepetition

###### Overview

A sub-type of Repetition based on the revolutions of a rotating spacecraft or instrument.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **RevolutionRepetition** | Extends | Repetition | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| revsSeparation | Expression <MAL::Long> | No | The required number of revolutions between occurrences. |
| revsTolerance | Expression <MAL::Long> | No | The allowed tolerance (+/-) in the required number of revolutions between occurrences, the interpretation of which is dependent on the separationType. |
| revAngle | Expression <MAL::Double> | No | Specifies the angle within a revolution. |
| units | MAL::String | Yes | Optional. The units for the quantity of angle, in which the revAngle is expressed in.  Default = ‘deg’. |

#### Other Repetitions

##### Data Type: TemporalRepetition

###### Overview

A sub-type of Repetition based on time.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **TemporalRepetition** | Extends | Repetition | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| initialTime | Expression <MAL::Time> | No | Nominal time of first occurrence. |
| separation | Expression <MAL::Duration> | No | The required time interval between occurrences. |
| tolerance | Expression <MAL::Duration> | No | The allowed tolerance (+/-) in the required time between occurrences, the interpretation of which is dependent on the separationType. |

##### Data Type: AngleRepetition [Optional]

###### Overview

A sub-type of Repetition based on the angle subtended between three physical objects.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **AngleRepetition** | Extends | Repetition | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| centreObject | Expression <Position> | No | Position of the centre object. |
| targetObject1 | Expression <Position> | No | Position of target object 1. |
| targetObject2 | Expression <Position> | No | Position of target object 2. |
| initialAngle | Expression <MAL::Double> | No | Initial angle subtended at the centre object by target objects 1 and 2. |
| separation | Expression <MAL::Double> | No | The required angle between occurrences.  If this is zero, this implies that repetition is between multiple occurrences of the initialAngle. |
| tolerance | Expression <MAL::Double> | No | The allowed tolerance (+/-) in the required angle between occurrences, the interpretation of which is dependent on the separationType. |
| units | MAL::String | Yes | Optional. The units for the quantity of angle, in which the separation and tolerance are expressed in.  Default = ‘deg’. |

###### Requirements

The repetition angle shall be that subtended at the centre object by target objects 1 and 2.

##### Data Type: EventRepetition

###### Overview

A sub-type of Repetition based on planning events.

###### Definition

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | **EventRepetition** | Extends | Repetition | SFP |  |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| eventRef | Expression <MAL::ObjectRef <EventDefinition>> | No | Reference to an EventDefinition (type of event). |
| separation | Expression <MAL::Long> | No | Number of occurrences of the planning event required between occurrences of the planning activity. |
| tolerance | Expression <MAL::Long> | No | The allowed tolerance (+/-) in the number of occurrences of the planning event between occurrences of the planning activity, the interpretation of which is dependent on the separationType. |

# Error Codes

## Overview

Standard error codes defined by the MAL are applicable to the MPS service operations. In particular it is noted that this includes errors associated with delivery issues and authorization failure.

Not all failure modes for the operations defined in this standard are adequately described using only the MAL error codes. For this purpose, a set of MPS-specific MO Errors is defined in this section. Where applicable, the operations in this standard may return one or more of these errors.

## MPS-Specific MO Errors

Each MPS-defined MO Error carries an additional ‘ExtraInfo’ field that can be used to convey additional information about the returned error. The type of this field may differ between different error codes; the type corresponding with each error code is given in its specification.

### Definition

| Error | # | Description | ExtraInfo Type | ExtraInfo Description |
| --- | --- | --- | --- | --- |
| INVALID | 1 | One or more fields in the message contain invalid values. | List <MAL::Pair> | ExtraInfo comprises a list of structures, each identifying an invalid field, comprising:   1. A MAL::String giving a dot-separated nested index for the invalid field(s). This allows for fields that are themselves a structure or list element to be denoted. For example, the form ‘3.2.4’ could mean the 4th element of the 2nd field of the composite structure that is the 3rd field of the message. 2. A MAL::UShort giving a secondary error code that details the reason for invalidity. |
| CANCEL\_FAILED | 2 | The cancelRequest operation failed to cancel the referenced RequestInstance. | MAL::String | ExtraInfo provides additional information on the reason for failure as a free format string. |
| UPDATE\_FAILED | 3 | The update operation (to Request, PlanStatus, Activity, Event or Resource) failed to update the referenced object. | MAL::String | ExtraInfo provides additional information on the reason for failure as a free format string. |
| REVOKE\_FAILED | 4 | The revokePlan operation failed to revoke the referenced Plan, for example because it has already started executing. | MAL::String | ExtraInfo provides additional information on the reason for failure as a free format string. |
| INSERT\_FAILED | 5 | The insertActivity or insertEvent operation failed to insert the requested object. | MAL::String | ExtraInfo provides additional information on the reason for failure as a free format string. |
| DELETE\_FAILED | 6 | The deleteActivity or deleteEvent operation failed to delete the requested object. | MAL::String | ExtraInfo provides additional information on the reason for failure as a free format string. |
| ACTIVATE\_FAILED | 7 | The activatePlan operation failed as the activation was outside the validity period of the Plan, or the start of the planPeriod had already passed. | MAL::String | ExtraInfo indicates ‘Validity’ or ‘Expired’ as appropriate. |
| DEACTIVATE\_FAILED | 8 | The deactivatePlan operation failed. | MAL::String | ExtraInfo provides additional information on the reason for failure as a free format string. |
| SUBMIT\_FAILED | 9 | The submitPlan operation failed as the submitted plan was already terminated. | MAL::String | ExtraInfo provides additional information on the reason for failure as a free format string. |
| UNSUPPORTED | 10 | An optional data structure used in the message is not supported by the service provider. | List <MAL::String> | ExtraInfo comprises a list of MAL::Strings giving a dot-separated nested index for the unsupported field(s). This allows for fields that are themselves a structure or list element to be denoted. For example, the form ‘3.2.4’ could mean the 4th element of the 2nd field of the composite structure that is the 3rd field of the message. |
| ACTIVATE\_SUBPLAN\_FAILED | 11 | The activateSubPlan operation failed. | MAL::String | ExtraInfo provides additional information on the reason for failure as a free format string. |
| DEACTIVATE\_SUBPLAN\_FAILED | 12 | The deactivateSubPlan operation failed. | MAL::String | ExtraInfo provides additional information on the reason for failure as a free format string. |

## Data Type: SecondaryErrorCodeEnum

### Overview

For the INVALID error, the secondary error code is a MAL::UShort that allows for deployment specific extensibility.

### Definition

|  |  |  |  |
| --- | --- | --- | --- |
| Name | **SecondaryErrorCodeEnum** | SFP |  |

| Error | Error # | Description |
| --- | --- | --- |
| UNKNOWN | 0 | Referenced MO object is not available to the service provider. |
| UNDEFINED | 1 | Undefined value for enumeration field. |
| OUT\_OF\_RANGE | 2 | A numeric value is outside the supported range. |
| UNRECOGNIZED | 3 | Value of type MAL::Identifier or MAL::String (referencing a named item) does not correspond to a known item. |
| BAD\_TIME | 4 | A date-time value is outside the supported time period. |
| BAD\_POSITION | 5 | A position value is outside the supported position range. |
| BAD\_DIRECTION | 6 | A direction value is outside the supported direction range. |
| INCONSISTENT | 7 | A value is inconsistent with that of another field within the message. This indicates violation of a constraint rule. |

# Service Specification XML

## Overview

The MO MAL specification (reference [2]) defines a normative XML Schema Definition (XSD) for validating MO service specifications and the MAL XML specification. The use of XML for service specification provides a machine-readable format rather than the text-based document format (reference [5]).

The MPS service specification defined in this document is also represented as an XML specification that follows the MAL defined schema.

The published XML Schema Definition (XSD) and the service specifications are held in an online SANA registry (reference [6]) located at:

<https://sanaregistry.org/r/moschemas/>

While these schemas encode the specifications prescribed by the standard in XML, that should not be confused as a prescription that any conforming implementation shall be encoded in XML. Implementations in any language can conform to the MPS service specification, or to the MO service specifications in general.

## XML Schema Definition (XSD) for MO Services

The XML Schema Definition (XSD) that is used to validate the actual XML service specifications has a filename with the structure ‘ServiceSchema-vBBB.xsd’, where ‘BBB’ is replaced with the issue number of the corresponding document.

The normative XML for an MO service specification has a filename with the structure ‘areaAAA-vBBB-AREA’ where ‘AAA’ is replaced with the area number, ‘BBB’ is replaced with the area version which shall match the issue number of the corresponding document and ‘AREA’ is replaced by the area name.

For this specification the following version of the XML Schema Definition (XSD) is applicable:

<https://sanaregistry.org/files/moschemas/ServiceSchema-v003.xsd>

The latest version of the XML Schema Definition is directly available from the address:

<https://sanaregistry.org/files/moschemas/ServiceSchema.xsd>

## MAL XML

The normative XML for the MAL specification, validated against the XML Schema Definition (XSD), is located at:

<https://sanaregistry.org/files/moschemas/area001-v003-MAL.xml>

where 001 corresponds to the area number for MAL and 003 corresponds to the area version on which this specification is based.

The latest version of the MAL specification is directly available from the address:

<https://sanaregistry.org/files/moschemas/ServiceDefMAL.xml>

## MPS XML

The normative XML for the MPS specification, validated against the XML Schema Definition (XSD), is located at[[3]](#footnote-4):

<https://beta.sanaregistry.org/files/moschemas/area005-v001-MPS.xml>

where 005 corresponds to the area number for MPS and 001 corresponds to the area version which shall match the issue number of this document.

The latest version of the MPS specification is directly available from the address:

<https://beta.sanaregistry.org/files/moschemas/ServiceDefMPS.xml>

# XML File Formats

## Overview

Some MPS deployments, including those based on legacy systems, may not be designed to use service-based interaction, using instead the transfer of files with a defined data structure. In order to support the exchange of information using the MPS Information Model structures, the following standardized file formats are defined:

* Planning Request;
* Planning Response (to a Planning Request);
* Plan.

These are based on the MPS information model defined in section 4, expressed as XML schemas, restricted to the data structures required to support Planning Requests and Plans respectively.

This section describes the approach taken to define the normative XML schemas and provides the location of the schemas in an online SANA registry. Detailed definition of the data structures referenced and the interpretation of their constituent fields can be found in section 4, except for additional or modified types specific to the file formats that are defined in this section.

## XML Schema Namespace

The XML schemas for the MPS File Formats are defined within a single namespace with the following URN:

urn:ccsds:schema:mo:mps

The XML schemas for the MALDataTypes is defined within a separate namespace for the MAL with the following URN:

urn:ccsds:schema:mo:mal

## XML Schema Encoding

### Overview

Data structures of the MPS information model are ultimately composed of fields defined as one of the MAL Attribute types. The encoding of these data types is fixed in the context of the XML file structures and associated XML schema. It follows the approach already defined for XML encoding of the MAL in section 5 of reference [D8], but is not directly compatible as it is adapted for file-based usage.

### Mapping

The following table defines the mapping of MAL Attribute types to XSD schema types used in the context of the MPS XML file structures.

Table 7‑1 : Mapping of MAL Attribute Types to XSD Types

| MAL Type | XSD Type | Comment |
| --- | --- | --- |
| MAL::Boolean | boolean |  |
| MAL::Octet | byte |  |
| MAL::UOctet | unsignedByte |  |
| MAL::Short | short |  |
| MAL::UShort | unsignedShort |  |
| MAL::Integer | int |  |
| MAL::UInteger | unsignedInt |  |
| MAL::Long | long |  |
| MAL::ULong | unsignedLong |  |
| MAL::Float | float |  |
| MAL::Double | double |  |
| MAL::Duration | duration |  |
| MAL::Time | mal:Time | Corresponds to XSD dateTime but with format constrained to CCYY-MM-DDThh:mm:ss.sss. |
| MAL::FineTime | mal:FineTime | Corresponds to XSD dateTime but with format constrained to CCYY-MM-DDThh:mm:ss.sssssssss. |
| MAL::String | string |  |
| MAL::Blob | hexbinary |  |
| MAL::Identifier | mal:Identifier |  |
| MAL::URI | anyURI |  |
| MAL::ObjectRef | mal:ObjectRef | Corresponds to an XSD complex type. |
| MAL::Element | anyType |  |

NOTES

1. The majority of MAL Attribute types correspond to the specified XSD simple types.
2. The remainder are defined within the MALDataTypes schema that is imported into the MPS schema. These are shown in the table above with the ‘mal:’ namespace prefix and either imply constraints to be imposed on their encoding as XSD simple types or define MAL specific XSD complex types.
3. MAL::Time and MAL::FineTime correspond to the XSD dateTime simple type but with constraints on the time format to be used.
4. In the specific case of an Expression of type Time defined in the MPSDataTypes schema, the value field is encoded as dateTime and can be used to represent either MAL type, according to the rules below.

Fields of type MAL::Time shall be encoded using the XSD dateTime simple type with the format CCYY-MM-DDThh:mm:ss.sss.

Fields of type MAL::FineTime shall be encoded using the XSD dateTime simple type with the format CCYY-MM-DDThh:mm:ss.sssssssss.

Fields of type MAL::Identifier shall be encoded using the XSD string simple type with any format constraint imposed by the MAL (reference [2]).

NOTE – MAL::Identifier corresponds to the XSD string simple type, but the string must follow the format defined for MAL Identifiers.

Fields of type MAL::ObjectRef shall be encoded using the XSD MAL::ObjectRef complex type. If a field is defined in the information model as being of type MAL::ObjectRef <T>, with a specified concrete object type T, then the *area* and type fields of the reference shall be omitted, with the correct values substituted in by the implementation.

NOTE – MAL::ObjectRef corresponds to an XSD complex types defined as part of the MPS XML schema hierarchy. The MALDataTypes schema also includes an XSD complex type for the MAL::ObjectIdentity composite.

Fields defined as a list of items of defined type shall be encoded as an XSD element of defined type with minimum and maximum occurrences.

Attributes of abstract types MAL::Element or MAL::Attribute shall be represented using the XSD anyType type.

## XML Schema Structure

### Overview

The MPS XML file structures are defined as a hierarchy of XML schemas, as illustrated in figure 7‑1 below. The names of the schemas correspond to the XSD files (appending the file extension ‘.xsd’), see also 7.8 for the list of schemas in the SANA registry.

Each MPS XML file structure has a root XSD schema that includes:

* the top level element for the file;
* XSD representations of each MPS Service Object that may be included as part of the file.

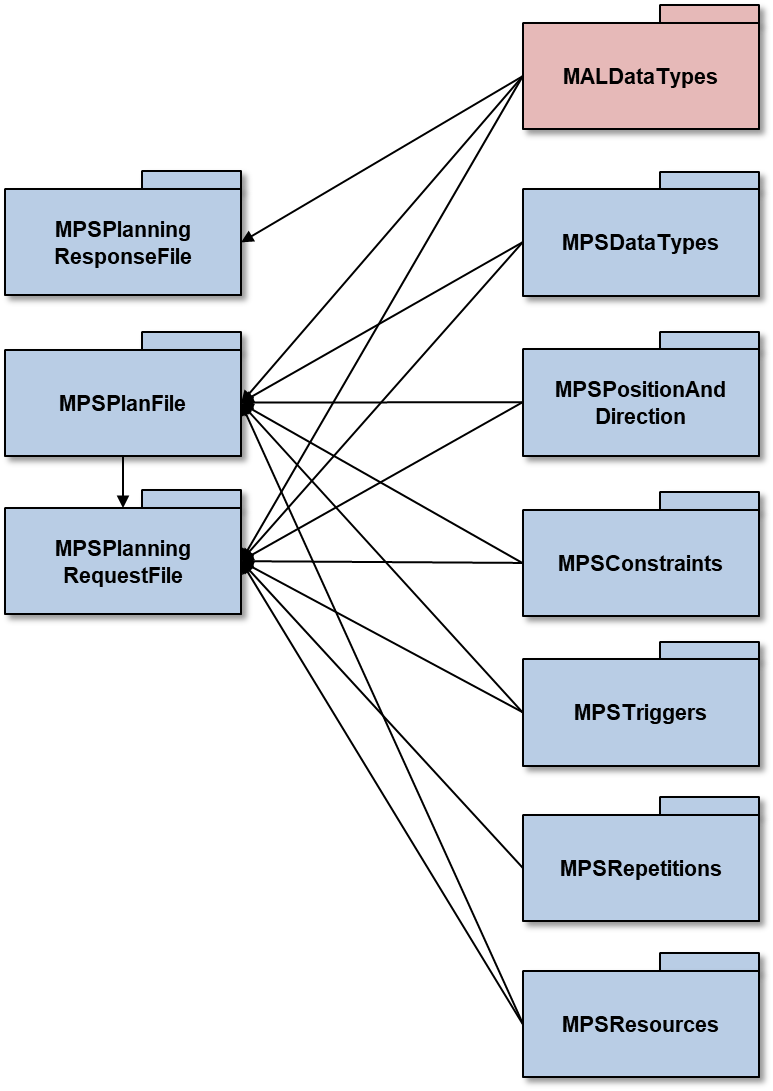


Figure 7‑1 : XML Schema Hierarchy

It should be noted that these root XSD schemas do not include all defined data structures relating to an MPS service object, but only those required for the specific file structure. Those not required, such as Definition data structures and service specific data structures, are omitted.

A set of XSD schemas are also defined that are included by the above MPSPlanFile and MPSPlanningRequestFile to define referenced data types within the MPS information model. It should be noted that these are not required by the MPSPlanningResponseFile schema. Separate schema are defined to cover:

* MAL Data Types: MAL data types that do not map directly to XSD simple types:
* Time, FineTime, and Identifier,
* ObjectRef and ObjectIdentity,
* NamedValue and File;
* MPS Data Types covering:
* Base Data Types (see 4.6.2),
* Expressions (see 4.6.4),
* Arguments (see 4.6.5);
* Position and Direction (see 4.6.3) [Optional];
* Constraints (see 4.6.6);
* Triggers (see 4.6.8);
* Repetitions (see 4.6.9);
* Resources (see 4.5.4);

The MPSRepetitions schema is only included by the MPSPlanningRequestFile schema.

The MPSPlanningRequestFile schema also includes the MPSPlanFile schema to support the direct embedding of a plan within a planning request.

The MPSResources schema defines resource profiles used within the MPSPlanFile and MPSConstraints schema.

## XML File Structure

### Overview

The XML file structure is illustrated in figure 7‑2 below. There is an XSD top level element for each MPS file.

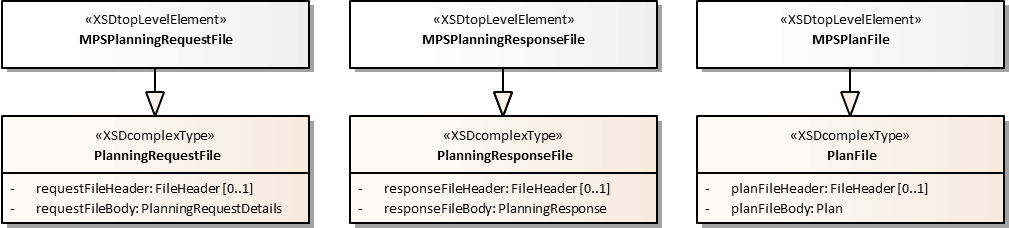


Figure 7‑2 : MPS XML File Structure

The general structure of the MPS files comprises an optional common FileHeader, followed by a file body that corresponds to:

* PlanningRequestDetails (see 4.5.5.3) in the case of a PlanningRequestFile;
* A dedicated structure, defined below, in the case of a PlanningResponseFile that is a combination of a PlanningRequestResponse and a RequestStatusUpdate (see 4.5.5.3);
* Plan (see 4.5.6) in the case of the PlanFile.

The FileHeader is an additional concrete data structure not defined in the context of the MPS Information Model, but defined below. This is included in the MPSDataTypes schema.

### PlanningRequestFile

#### Definition

|  |  |
| --- | --- |
| Name | **PlanningRequestFile** |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| requestFileHeader | FileHeader | Yes | File header giving details of the source and scope of the MPS file. |
| requestFileBody | PlanningRequestDetails | No | File body corresponding to a the PlanningRequestDetails structure. |

### PlanningResponseFile

#### Definition

|  |  |
| --- | --- |
| Name | **PlanningResponseFile** |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| responseFileHeader | FileHeader | Yes | File header giving details of the source and scope of the MPS file. |
| responseFileBody | PlanningResponse | No | File body corresponding to a combination of PlanningRequestResponse and RequestStatusUpdate (defined in 7.6 below). |

### PlanFile

#### Definition

|  |  |
| --- | --- |
| Name | **PlanFile** |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| planFileHeader | FileHeader | Yes | File header giving details of the source and scope of the MPS file. |
| planFileBody | Plan | No | File body corresponding to a Plan structure. |

### FileHeader

#### Overview

In the absence of any standard MO message headers, the MPSFileHeader provides optional information about the origin, creation time, schema version, and scope of the file.

#### Definition

|  |  |
| --- | --- |
| Name | **FileHeader** |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| originator | MAL::Identifier | Yes | Identity of the entity responsible for generation of the file. |
| creationTime | MAL::Time | Yes | File creation date and time. |
| schemaVersion | MAL::Identifier | Yes | Version of the XSD schemas used to encode the current XML document, in the format ‘v.v’ (see 7.8). |
| mission | MAL::Identifier | Yes | The space mission to which the file relates. |
| domain | List <MAL::Identifier> | Yes | Mission domain to which the file relates. |
| topic | MAL::String | Yes | Mission specific information that further defines the scope of the file. |
| comments | MAL::String | Yes | Field for any additional information or comments. |

## Planning Request XML File Formats

### General

Two XML file formats are defined for use in conjunction with MPS planning requests:

* MPSPlanningRequestFile;
* MPSPlanningResponseFile.

The body of the MPSPlanningRequestFile corresponds to a PlanningRequestDetails structure from the MPS Information Model (see 4.5.5.3).

The MPSPlanningRequestFile schema includes the definition of XSD complex types for the following data structures:

* PlanningRequestDetails;
* ActivityDetails (ActivityNode and SimpleActivityDetails).

The XSD representation of these data structures omits some fields that are only meaningful where they are dynamically updatable. Similarly the set of allowed values for status fields may be restricted to those relevant to the context of the message, omitting those relevant only during or post execution of the planned activities.

In the specific case of the Planning Request File, the option to include an embedded requested Plan is supported by inclusion of the MPSPlanFile schema.

### PlanningResponse

#### Definition

|  |  |
| --- | --- |
| Name | **PlanningResponse** |

| Field | Type | Nullable | Description |
| --- | --- | --- | --- |
| instance | MAL::ObjectRef <RequestInstance> | No | Reference to the RequestInstance created in response to a submitted MPSPlanningRequestFile. |
| userReference | MAL::Identifier | No | User supplied reference for the planning request. This is distinct from the identity of the RequestInstance that is assigned by the planning function. |
| status | RequestStatusEnum | Yes | Current status of the planning request. |
| outputPlanRefs | List <MAL::ObjectRef <Plan>> | Yes | Reference to the output Plan(s) that contains the activities resulting from the planning request. Where multiple alternate plans have been generated, these may be listed here. It should be noted that this is only available once the planning request has been processed and successfully planned. The outputPlanRefs may be updated following iterative planning cycles or re-planning. |
| returnData | List <MAL::NamedValue> | Yes | Optional return data from the planning process, provided as a list of ID-Value pairs. This can be used to provide additional information required by the User to interpret the planned operations. |
| statusInfo | MAL::String | Yes | StatusInfo provides the reason for termination and is customizable, but includes:  - Completed (all constituent activities completed successfully)  - Expired (constituent activities expired prior to execution)  - Failed (constituent activities failed during execution)  - Deleted (constituent activities were deleted) |
| errorCode | MAL::Integer | Yes | Error Code optional in the case of a failure status for the planning request (for example Terminated state with statusInfo Failed). The codes are implementation specific. |
| errorInfo | MAL::String | Yes | Supplementary error information. |

#### Discussion

In order to be able to interpret a plan and correlate a planning request with the resultant planning activities contained in the plan, the identity of the planning request is required. This identity is assigned by the planning function, not the originator of the planning request. This may be returned through ad-hoc mechanisms; however, the MPSPlanningResponseFile is provided as a standard file format to supply this feedback. The file may also optionally include the result of the planning request. The body of the MPSPlanningResponseFile is based on the PlanningRequestResponse and RequestStatusUpdate structures from the MPS information model, but uses the dedicated hybrid structure detailed below.

Although the PlanningResponse message combines the information contained in both the PlanningRequestResponse and the RequestStatusUpdate, this does not imply that the corresponding operations within the planning system have to be combined. It is implementation-dependent when the file is generated and whether the status information is included. There is no definition of the service interface for deployments using the file formats.

This allows the return of both the reference to the created RequestInstance and current status of the request in a single PlanningResponseFile. No timestamp is included within the structure above, but it is recommended that the creation time field of the file header is used, particularly if the status field is provided.

The point at which any PlanningResponseFile is generated by a planning function is deployment-specific. This may limit the set of possible status values, and whether or not the other nullable fields are meaningful in context. The outputPlanRef is only relevant if the planning process has generated an output Plan addressing the planning request. The last three fields may be used to supply a reason for rejection, but are most relevant post execution of the Plan.

NOTE – In the equivalent Planning Request Service operation, an error may be returned if the planning request is INVALID, and no RequestInstance is created. In the case of an invalid PlanningRequestFile, this error reporting mechanism is not available. Providing the PlanningRequestFile is not so badly formed that its identity cannot be interpreted, it would be possible to provide a PlanningResponseFile in return. In this case, a RequestInstance would need to be created and assigned the status REJECTED, the statusInfo field set to ‘INVALID’ and the errorCode and errorInfo fields set consistently with the error code defined in section 4 above.

## Plan XML File Format

A single XML file format is defined for distribution of MPS plans:

* MPSPlanFile.

The body of the MPSPlanFile corresponds to a Plan structure from the MPS information model (see 4.5.6).

The MPSPlanFile schema includes definition of XSD complex types for the following data structures:

* Plan (together with subsidiary data structures for PlanInformation, PlannedItems, PlanRevision);
* EventInstance;
* ActivityInstance;
* ResourceProfile.

The XSD representation of these data structures omits some fields that are only meaningful where they are dynamically updatable, such as statusInfo. Similarly the set of allowed values for status fields may be restricted to those relevant to the context of the message, omitting those relevant only during or post execution of the planned activities. The following details the specific omissions for each data structure:

Plan: PlanStatusEnum restricted to Draft and Released, statusInfo omitted.

EventInstance: eventStatus and statusInfo omitted.

ActivityInstance: status, executionInstance, returnData, statusInfo, errorCode, and errorInfo omitted.

## MPS XML File Format Schema Location

The published XML schemas for MPS File Formats are held in an online SANA registry, located at the following URL:

<https://sanaregistry.org/r/mpsfileschemas/>

The following schemas are located within this registry:

MPSPlanningRequestFile.xsd

MPSPlanningResponseFile.xsd

MPSPlanFile.xsd

MPSDataTypes.xsd

MPSPositionAndDirection.xsd

MPSConstraints.xsd

MPSTriggers.xsd

MPSRepetitions.xsd

MPSResources.xsd

MALDataTypes.xsd

It is not possible to add versions to schema files because this would invalidate the file references within the schemas. As such, all the above files are contained within a single .zip file with the following naming convention: ‘MPSFileFormats-v.v.zip’, where the ‘v.v’ part is replaced with the issue number of the corresponding document and an incremental patch number counting from zero.

The file schemas defined by this document can be found at[[4]](#footnote-5):

<https://beta.sanaregistry.org/files/mpsfileschemas/MPSFileFormats-1.0.zip>

The latest version of any specification may always be directly addressed by removing the ‘-v.v’ part from the URL; for example:

<https://beta.sanaregistry.org/files/mpsfileschemas/MPSFileFormats.zip>

1. Protocol Implementation Conformance   
   Statement (PICS) Proforma  
      
   (Normative)
   1. INTRODUCTION
      1. OVERVIEW

This annex provides the Protocol Implementation Conformance Statement (PICS) Requirements List (RL) for an implementation of the Mission Operations MPS Services Recommended Standard. The PICS for an implementation is generated by completing the RL in accordance with the instructions below. An implementation claiming conformance must satisfy the mandatory requirements referenced in the RL.

The MO MPS Services Recommended Standard includes optional elements, as outlined in 2.6. These comprise:

* Optional services and service capability sets as summarized in table 3‑1. A compliant deployment can support any combination of MO MPS services. If a service is supported, then it must support any mandatory service capability sets; other capability sets are optional.
* Optional data structures corresponding to optional elements of the MPS Information Model, as summarized in table 4‑1.

An implementation’s completed RL is called the PICS. The PICS states which protocol features have been implemented. The following entities can use the PICS:

* the protocol implementer, as a checklist to reduce the risk of failure to conform to the Recommended Standard through oversight;
* the supplier and acquirer or potential acquirer of the implementation, as a detailed indication of the capabilities of the implementation, stated relative to the common basis for understanding provided by the standard PICS proforma;
* the user or potential user of the implementation, as a basis for initially checking the possibility of interworking with another implementation (while interworking can never be guaranteed, failure to interwork can often be predicted from incompatible PICSes);
* a protocol tester, as the basis for selecting appropriate tests against which to assess the claim for conformance of the implementation.
  + 1. Notation
       1. Status Column Symbols

The following are used in the RL to indicate the status of features:

|  |  |
| --- | --- |
| **Symbol** | **Meaning** |
| M | Mandatory |
| O | Optional |
| C | Mandatory if its optional parent feature is implemented |

* + - 1. Support Column Symbols

The support of every item as claimed by the implementer is stated by entering the appropriate answer (Y, N, or N/A) in the support column.

|  |  |
| --- | --- |
| **Symbol** | **Meaning** |
| Y | Yes, supported by the implementation |
| N | No, not supported by the implementation |
| N/A | Not applicable |

* 1. General Information
     1. Identification of PICS

|  |  |  |
| --- | --- | --- |
| **Ref** | **Question** | **Response** |
| 1 | Date of Statement (DD/MM/YYYY) |  |
| 2 | CCSDS document number containing the PICS |  |
| 3 | Date of CCSDS document containing the PICS |  |

* + 1. Identification of implementation under test (IUT)

|  |  |  |
| --- | --- | --- |
| **Ref** | **Question** | **Response** |
| 1 | Implementation name |  |
| 2 | Implementation version |  |
| 3 | Machine name |  |
| 4 | Machine version |  |
| 5 | Operating System name |  |
| 6 | Operating System version |  |
| 7 | Special Configuration |  |
| 8 | Other Information |  |

* + 1. User Identification

|  |  |
| --- | --- |
| Supplier |  |
| Contact Point for Queries |  |
| Implementation name(s) and Versions |  |
| Other Information Necessary for full identification, for example, name(s) and version(s) for machines and/or operating systems;  System Name(s) |  |

* + 1. Instructions for Completing the RL

An implementer shows the extent of compliance to the protocol by completing the RL; the resulting completed RL is called a PICS.

* 1. MPS Services PICS

The MPS RL has an entry for each configuration parameter, service, service capability set, file format, and information model element set.

* + 1. Configuration Parameters PICS

The configuration parameters that may be used to configure a deployment are shown below. Each configuration parameter is shown as a separate RL item using the format C.*n*.

| **Configuration Parameters** | | | | |
| --- | --- | --- | --- | --- |
| **Item** | **Protocol Feature: (Data Type)** | **Reference** | **Status** | **Support** |
| C.1 | additionalTimeSystems: (List <MAL::Identifier>) | 4.4.1 | O |  |
| C.2 | defaultTimeSystem: (MAL::Identifier) | 4.4.1 | M |  |
| C.3 | additionalReferenceFrames: (List <MAL::Identifier>) | 4.4.2 | O |  |
| C.4 | defaultReferenceFrame: (MAL::Identifier) | 4.4.2 | C |  |
| C.5 | additionalCelestialBodies: (List <MAL::Identifier>) | 4.4.3 | O |  |
| C.6 | additionalPointingTemplates: (List <MAL::Identifier>) | 4.4.4 | O |  |
| C.7 | additionalUnits: (List <MAL::String>) | 4.4.5 | O |  |
| C.8 | planningPeriods: (List <MAL::Identifier>) | 4.5.5.3.2.2, 4.5.5.3.4.2 | O |  |
| C.9 | orbitAnomalyType: (enum {TRUE = 1, MEAN = 2}) | 4.6.3.2.5.2 | C |  |
| C.10 | orbitDatum: (MAL::Double) | 4.6.3.2.5.2 | C |  |
| C.11 | positionReferences: (List <MAL::Identifier>) | 4.6.3.2.7.2 | O |  |
| C.12 | directionReferences: (List <MAL::Identifier>) | 4.6.3.3.6.2 | O |  |

NOTE – This standard is neither specifying the internal definitions of these configuration parameters, nor the distribution of these parameters across a mission ground segment.

* + 1. Services PICS

There are 5 separate MPS services defined, each of which has multiple capability sets. The service is shown as a top level item, with subsidiary items for each capability set, using the format S.*s*.*c* in the Item column, where *s* is the service number, and *c* is the capability set number. All services are optional, but if implemented, some capability sets are mandatory. Compliance with a service implies compliance with the service operations and message structures defined in 3, together with applicable high-level, functional, and structural requirements defined therein. For capability sets, the service operations it comprises are listed in the Protocol Feature column.

| **Service Capability Sets** | | | | |
| --- | --- | --- | --- | --- |
| **Item** | **Protocol Feature** | **Reference** | **Status** | **Support** |
| S.1 | Planning Request Service | 2.5.2 and 3.5 | O |  |
| S.1.1 | submitRequest  getRequestSummaries  getRequestStatus | 2.5.2 and 3.5 | C |  |
| S.1.2 | cancelRequest | 2.5.2 and 3.5 | O |  |
| S.1.3 | updateRequest | 2.5.2 and 3.5 | O |  |
| S.1.4 | monitorRequestStatus | 2.5.2 and 3.5 | O |  |
| S.1.5 | getRequest | 2.5.2 and 3.5 | O |  |
| S.2 | Plan Distribution Service | 2.5.3 and 3.6 | O |  |
| S.2.1 | getPlanSummaries  getPlan  getPlanStatus | 2.5.3 and 3.6 | C |  |
| S.2.2 | monitorPlanStatus | 2.5.3 and 3.6 | O |  |
| S.2.3 | monitorPlan | 2.5.3 and 3.6 | O |  |
| S.2.4 | queryPlan | 2.5.3 and 3.6 | O |  |
| S.2.5 | getPartialPlan | 2.5.3 and 3.6 | O |  |
| S.3 | Plan Execution Control Service | 2.5.4 and 3.7 | O |  |
| S.3.1 | submitPlan  revokePlan  getPlanStatus | 2.5.4 and 3.7 | C |  |
| S.3.2 | activatePlan  deactivatePlan | 2.5.4 and 3.7 | O |  |
| S.3.3 | monitorPlanExecution | 2.5.4 and 3.7 | O |  |
| S.3.4 | monitorPlanExecutionDetail | 2.5.4 and 3.7 | O |  |
| S.3.5 | activateSubPlan  deactivateSubPlan  getSubPlanStatus | 2.5.4 and 3.7 | O |  |
| S.3.6 | monitorSubPlanExecution | 2.5.4 and 3.7 | O |  |
| S.3.7 | suspendActivity  resumeActivity | 2.5.4 and 3.7 | O |  |
| S.3.8 | getActivityStatus | 2.5.4 and 3.7 | O |  |
| S.4 | Plan Information Management Service | 2.5.5 and 3.8 | O |  |
| S.4.1 | listRequestDefs  getRequestDefs | 2.5.5 and 3.8 | O |  |
| S.4.2 | listEventDefs  getEventDefs | 2.5.5 and 3.8 | O |  |
| S.4.3 | listActivityDefs  getActivityDefs | 2.5.5 and 3.8 | O |  |
| S.4.4 | listResourceDefs  getResourceDefs | 2.5.5 and 3.8 | O |  |
| S.5 | Plan Edit Service | 2.5.6 and 3.9 | O |  |
| S.5.1 | updatePlanStatus | 2.5.6 and 3.9 | C |  |
| S.5.2 | insertActivity  insertEvent  deleteActivity  deleteEvent | 2.5.6 and 3.9 | O |  |
| S.5.3 | updateActivity  updateEvent | 2.5.6 and 3.9 | O |  |
| S.5.4 | updateResourceValue | 2.5.6 and 3.9 | O |  |
| S.5.5 | updateResourceProfile | 2.5.6 and 3.9 | O |  |
| S.5.6 | applyTimeShift | 2.5.6 and 3.9 | O |  |

* + 1. File Formats PICS

Additional items are shown for each of the standard file formats that may be supported without services. These are indicated in the Item column by F.*n*. Compliance with a file format implies compliance with the file structures defined in 7 and formally expressed in the referenced XML schemas for MPS file formats.

| **File Formats** | | | | |
| --- | --- | --- | --- | --- |
| **Item** | **Protocol Feature** | **Reference** | **Status** | **Support** |
| F | MPS File Formats | 7 | O |  |
| F.1 | Planning Request File Format | 7.6.1 | O |  |
| F.2 | Planning Response File Format | 7.6.2 | O |  |
| F.3 | Plan File Format | 7.7 | O |  |

* + 1. Information Model PICS

The MPS services use an information model that itself has optional elements, which implies some features may not be supported within service messages.

While a compliant deployment shall support the full structure of messages exchanged at the service interface for supported services and capability sets, it is not required to support optional data structures within those messages at application level.

Support for optional elements of the information model are indicated in the Item column by I.*n*, with subsidiary items for each optional element set, corresponding to those identified in 4.2. This section should be completed in conjunction with both the supported services and file formats items in the PICS tables above.

| **Information Model Element Sets** | | | | |
| --- | --- | --- | --- | --- |
| **Item** | **Protocol Feature** | **Reference** | **Status** | **Support** |
| I.1 | Core Features | 4.5.1, 4.5.2, 4.5.3, 4.5.5, 4.5.6 (not 4.5.6.4), 4.5.7, 4.6.2, 4.6.4, 4.6.5, 4.6.6.1, 4.6.8.1, 4.6.8.2, 4.6.8.6, 4.6.9.1, 4.6.9.4.1, 4.6.9.4.3 | M |  |
| I.2 | Basic Constraints | 4.6.6.2, 4.6.6.5.1, 4.6.6.5.2 | O |  |
| I.3 | Plan Revisions | 4.5.6.4 | O |  |
| I.4 | Resources | 4.5.4 | O |  |
| I.5 | Resource Constraints (requires I.4) | 4.6.6.3, 4.6.7 | O |  |
| I.6 | Position & Direction | 4.6.3, 4.6.8.3, 4.6.8.4, 4.6.8.5, 4.6.9.2, 4.6.9.3, 4.6.9.4.2 | O |  |
| I.7 | Geometric Constraints (requires I.6) | 4.6.6.4 | O |  |
| I.8 | Functions | 4.5.8, 4.6.6.5.3 | O |  |

1. Security, SANA, and Patent Considerations  
      
   (Informative)
   1. Security Considerations
      1. System Security Requirements

Security requirements are specific to the deployed mission system and can vary significantly between different mission systems. The MPS services or file formats support a limited subset of the interactions supported by a typical mission system, and as such must be capable of deployment in the context of a mission or organization specific security architecture that supports multiple services.

The mission security architecture shall address the following:

* Protection of the communications link between MPS service consumer and provider to ensure data integrity and confidentiality. This may or may not include the encryption of service messages, depending on mission specific requirements.
* Control of access to specific MPS services, service operations and service data through the management of access rights associated with registered service users.
* Authentication to ensure only genuine registered service users have access to MPS services and to ascertain their level of access rights.

For the MPS services, the security considerations of this specification are the same as those of the MAL in reference [2]. Specifically, authentication and authorization of a participating consumer or provider is provided by the MAL access control concept and is covered in subsections 3.6, 5.2, and 5.3 of the Reference Model (reference [1]).

Security of the communications link carrying the MPS services is delegated to the implementation of the underlying Transport Layer.

For MPS file formats, this specification only addresses the format of the files, not the method of transfer. All security considerations relating to the transfer of these files must therefore be addressed by the actual file transfer service and security architecture of the deployed mission system.

* + 1. Potential Threats

In many mission systems, mission planning is the principal or nominal way of controlling the mission. Unauthorized access to the MPS services can therefore be a means of sabotaging the mission:

* the Planning Request Service could be used to inject dangerous planning requests into the system. This is mitigated by the Planning function itself, the implementation of which can be used to detect and flag potentially dangerous requests either for automatic rejection or to obtain authorization from the mission planner.
* the Planning Request and Plan Distribution Services could be used to access confidential information about submitted planning requests or planned activities.
* The Plan Execution Control Service could be used to submit unauthorized plans for automated execution, or to stop execution of the currently authorized plans.
* The Plan Edit Service could be used to make unauthorized changes to the currently authorized plans, by deleting or modifying planned activities and events, or by inserting new activities or events into the currently authorized plans.
  + 1. Access Control

The MPS services are closely tied to the Access Control aspect of the MAL where returned authentication identifiers are used in the MAL message header to authenticate and authorize messages via Access Control.

Registered users are assigned roles (access rights) that may limit their access to MPS services. The set of access control roles is specific to the service deployment. An implementation of the MPS service provider can then restrict access to services, service capability sets, and individual service operations based on the assigned roles. Similarly roles can be used to restrict access to a subset of service data, either by data class or domain.

Which access control roles are supported is specific to the mission deployment and depends on the access control requirements for the mission. Typical MPS roles include access to:

* Individual MPS Services: access to Plan Execution Control and Plan Edit Services is likely to be more restricted than to the Planning Request and Plan Distribution Services.
* Restricted Capability Sets or individual Operations of an MPS Service: more sensitive operations may require a special access control role.
* Data class or domain of contained information (for example to restrict access to a specific spacecraft, subsystem or payload in terms of available planning requests and planning activities)
* Information pertaining to other users: access for some users may be restricted to their own planning requests and planned activities.
* Submission of custom planning requests: access for some users may be restricted to predefined planning requests.

It is the responsibility of the implementation of the MPS service provider to enforce access control based on the assigned user roles.

* + 1. Data Integrity

As stated previously, the confidentiality and integrity of MPS service messages is delegated to the implementation of the underlying transport layer.

This is dependent on the technology used for the implementation of the transport layer and the corresponding MAL technology binding. It may include the encryption of the service messages.

* + 1. Authentication

Authentication for the MPS Services, as for all MO Services, may be supported through the MO Common Login Service (reference [D3]).

The Login service allows a service user to provide authentication information to the system. It takes the user’s credentials and uses a deployment-specific mechanism to authenticate the user; the result of this is used by the MAL during access control.

The Login service and the access control provided by the MAL are fully dependent on a deployment-specific security architecture (for example, the authentication protocol Kerberos). Both layers (Common and MAL) provide access to, and use of, this security service; they do not implement it themselves.

* + 1. Confidentiality

For some missions, there may be commercial or security considerations that result in a need for confidentiality of planning requests and resultant planned activities.

Where this is the case, the MPS service provider may be required to implement an access control filter on the return of information to users. In particular:

* Visibility of planning requests may be restricted (based on a specific access control role) to those raised by the user currently accessing the service.
* Distributed plans may similarly be filtered to restrict visibility of the planned activities resulting from restricted planning requests. A partial plan would then be returned in place of the full unrestricted plan.

It is mission and deployment specific how this is implemented, but it is expected that this would make use of special access control roles assigned to users. This may typically restrict access to information based on a level of access rights:

1. All Information;
2. User’s Own + Unrestricted Information;
3. Only User’s Own Information.
   * 1. Auditing

The MPS Services include the potential to access the evolving state of planning requests and plans (together with their contained planning activities, planning events and planning resources) through the delivery of status updates pertaining to the corresponding MPS service objects.

The full set of updates provides a comprehensive audit trail on the execution of MPS service operations at the level of planning requests and plans.

While storage of and access to historical updates is not directly supported by the MPS Service specification, this provides the potential for a service provider to implement such an audit trail.

* + 1. Availability

Availability requirements are mission specific. The required availability of an MPS service provider implementation will impact its design, both in terms of the physical deployment architecture and its software implementation.

* 1. SANA Considerations

The recommendations of this document request SANA populate the registry specified in reference [2] with the schema and XML detailed in section 6 of this document.

As stated in reference [2], the registration rule for change to this registry requires an engineering review by a designated expert. The expert shall be assigned by the MPS WG Chair, or in absence, MOIMS Area Director.

Specifically, this applies to the following registries:

<https://sanaregistry.org/r/moschemas/ServiceSchema-v003.xsd> for entries relating to the MPS Service Specifications.

<https://sanaregistry.org/r/mpsfileschemas/> for entries relating to MPS File Format Schemas.

* 1. Patent Considerations

The recommendations of this document have no patent issues.

1. Definition of Acronyms  
      
   (Informative)

| Acronym | Definition |
| --- | --- |
| AOS | Acquisition Of Signal |
| CSS | Cross Support Services |
| LOS | Loss of Signal |
| MAL | Message Abstraction Layer |
| MO | Mission Operations |
| MOIMS | Mission Operations and Information Management Systems [CCSDS Area] |
| MPS | Mission Planning and Scheduling |
| NAV | CCSDS Navigation Working Group |
| NEM | Navigation Event Message |
| ODM | Orbit Data Message |
| PDS | Plan Distribution Service |
| PECS | Plan Execution Control Service |
| PES | Plan Edit Service |
| PI | Principal Investigator |
| PICS | Protocol Implementation Conformance Statement |
| PIMS | Plan Information Management Service |
| PRM | Pointing Request Message |
| PRS | Planning Request Service |
| RASDS | Reference Architecture for Space Data Systems |
| RL | Requirements List |
| SANA | CCSDS Space Assigned Numbers Authority |
| SFN | Short Form Number |
| SFP | Short Form Part |
| TOO | Target Of Opportunity |
| UML | Unified Modeling Language |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| URN | Uniform Resource Name |
| XML | eXtensible Markup Language |

1. Informative References  
      
   (Informative)

[D1] *Mission Operations Services Concept*. Issue 3. Report Concerning Space Data System Standards (Green Book), CCSDS 520.0-G-3. Washington, D.C.: CCSDS, December 2010.

[D2] *Mission Planning and Scheduling*. Issue 1. Report Concerning Space Data System Standards (Green Book), CCSDS 529.0-G-1. Washington, D.C.: CCSDS, June 2018.

[D3] *Mission Operations—Common Services*. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 522.0-B-1. Washington, D.C.: CCSDS, May 2020.

[D4] *Mission Operations Monitor & Control Services*. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 522.1-B-1. Washington, D.C.: CCSDS, October 2017.

[D5] *Navigation Data Messages Overview*. Issue 3. Report Concerning Space Data System Standards (Green Book), CCSDS 500.2-G-3. Washington, D.C.: CCSDS, March 2023.

[D6] *Cross Support Service Management—Simple Schedule Format Specification*. Issue 1, Technical Corrigendum 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 902.1-B-1. Washington, D.C.: CCSDS, December 2021.

[D7] *Space Engineering—Telemetry and Telecommand Packet Utilization*. ECSS-E-ST-70-41C. Noordwijk, The Netherlands: ECSS Secretariat, 15 April 2016.

[D8] *Mission Operations—Message Abstraction Layer Binding to HTTP Transport and XML Encoding*. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 524.3-B-1. Washington, D.C.: CCSDS, June 2018.

[D9] *Application and Support Layer Architecture*. Issue 1. Report Concerning Space Data System Standards (Green Book), CCSDS 371.0-G-1. Washington, D.C.: CCSDS, November 2020.

[D10] *Orbit Data Messages*. Issue 3. Recommendation for Space Data System Standards (Blue Book), CCSDS 502.0-B-3. Washington, D.C.: CCSDS, April 2023.

1. Tags are used in the MPS information model to capture the equivalent concept to subschedules of PUS service 11 in the ECSS Packet Utilization Standard (reference [D7]). [↑](#footnote-ref-2)
2. For example, the operation is equivalent to the ECSS PUS Service 11 Time Shift operation (15). Service 11 is for the management of on-board time-based schedules (also known as on-board queues or mission timelines). [↑](#footnote-ref-3)
3. Note that this Red book contains references to the beta SANA registry and will be changed to the formal SANA registry on publication as a Blue Book. [↑](#footnote-ref-4)
4. Note that this Red book contains references to the beta SANA registry and will be changed to the formal SANA registry on publication as a Blue Book [↑](#footnote-ref-5)