

Report Concerning Space Data System Standards

|  |
| --- |
| Producer-Archive Interface Specification (PAIS)  A Tutorial |

Draft Informational Report

CCSDS 651.2-G-0

Draft Green Book

August 2015

AUTHORITY

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | Issue: | Draft Green Book, Issue 0 |  |
|  | Date: | August 2015 |  |
|  | Location: | Not Applicable |  |
|  | | | |

**(WHEN THIS INFORMATIONAL REPORT 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 reflects the consensus of technical experts from 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-3).

This document is published and maintained by:

CCSDS Secretariat

Space Communications and Navigation Office, 7L70

Space Operations Mission Directorate

NASA Headquarters

Washington, DC 20546-0001, USA

FOREWORD

Through the process of normal evolution, it is expected that expansion, deletion, or modification of this document may occur. This document is therefore subject to CCSDS document management and change control procedures which are defined in *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-3). 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 addressed to the CCSDS Secretariat at the 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 e.V. (DLR)/Germany.
* European Space Agency (ESA)/Europe.
* Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
* Japan Aerospace Exploration Agency (JAXA)/Japan.
* National Aeronautics and Space Administration (NASA)/USA.
* Federal Space Agency (FSA)/Russian Federation.
* UK Space Agency/United Kingdom.

Observer Agencies

* Austrian Space Agency (ASA)/Austria.
* Belgian Federal Science Policy Office (BFSPO)/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.
* Chinese 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.
* 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.
* Indian Space Research Organization (ISRO)/India.
* Institute of Space Research (IKI)/Russian Federation.
* KFKI Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
* Korea Aerospace Research Institute (KARI)/Korea.
* Ministry of Communications (MOC)/Israel.
* 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.
* 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.
* United States Geological Survey (USGS)/USA.

DOCUMENT CONTROL

|  |  |  |  |
| --- | --- | --- | --- |
| **Document** | **Title and Issue** | **Date** | **Status** |
| CCSDS 651.2-G-0 | Producer-Archive Interface Specification (PAIS) – A Tutorial, Draft Informational Report, Issue 0 | August 2015 | Current draft |
|  |  |  |  |

CONTENTS

Section Page

[1 Introduction 1-1](#_Toc427311407)

[1.1 Purpose And Scope 1-1](#_Toc427311408)

[1.2 Document Structure 1-1](#_Toc427311411)

[1.3 Definitions 1-1](#_Toc427311412)

[1.3.1 Acronyms and Abbreviations 1-1](#_Toc427311413)

[1.3.2 Glossary of Terms 1-3](#_Toc427311414)

[1.4 Conventions 1-5](#_Toc427311415)

[1.5 References 1-7](#_Toc427311416)

[2 PAIS at a Glance 2-1](#_Toc427311417)

[3 Modeling Transfers 3-4](#_Toc427311418)

[3.1 Model of Objects for Transfer 3-4](#_Toc427311419)

[3.1.1 Transfer Object Type Descriptor 3-5](#_Toc427311420)

[3.1.2 Collection Descriptor 3-6](#_Toc427311421)

[3.2 Submission Information Package (SIP) 3-7](#_Toc427311422)

[3.2.1 SIP Constraints 3-7](#_Toc427311423)

[3.2.2 SIP Model 3-7](#_Toc427311424)

[3.2.3 relationships between sip constraints and mot 3-8](#_Toc427311425)

[3.3 A Methodology for Modeling a Transfer 3-9](#_Toc427311426)

[4 Writing XML Descriptors And SIP Constraints 4-12](#_Toc427311427)

[4.1 Structures and Construction Rules 4-12](#_Toc427311428)

[4.1.1 POLDER – a high level description 4-12](#_Toc427311429)

[4.1.2 Organization Of XML Documents 4-13](#_Toc427311430)

[4.1.3 XML Namespace 4-14](#_Toc427311431)

[4.1.4 CollectionS 4-15](#_Toc427311432)

[4.1.5 Transfer Objects 4-17](#_Toc427311433)

[4.1.6 Transfer Objects – Group 4-19](#_Toc427311434)

[4.1.7 Transfer Objects – Data Objects 4-23](#_Toc427311435)

[4.2 Management of MOT Identifiers 4-25](#_Toc427311436)

[4.3 Objects Occurrences and Sizes 4-25](#_Toc427311437)

[4.3.1 Occurrence Type 4-26](#_Toc427311438)

[4.3.2 Transfer Object Sizes 4-29](#_Toc427311439)

[4.4 Objects Encodings 4-30](#_Toc427311440)

[4.4.1 encoded Type 4-31](#_Toc427311441)

[4.5 Objects Relations 4-33](#_Toc427311442)

[4.5.1 parent collection relation 4-34](#_Toc427311443)

[4.5.2 association TYPE relation 4-34](#_Toc427311444)

[4.6 SIP Constraints 4-39](#_Toc427311445)

[4.7 Customization – Extensions and Specializations 4-43](#_Toc427311446)

[4.7.1 PAIS XML Schemas 4-43](#_Toc427311447)

[4.7.2 Extensions 4-44](#_Toc427311448)

[4.7.3 Restrictions 4-46](#_Toc427311449)

[5 Building and Manipulating SIPs 5-51](#_Toc427311450)

[5.1 Understanding the PAIS Abstract SIP 5-51](#_Toc427311451)

[5.1.1 SIP container 5-53](#_Toc427311452)

[5.1.2 Transfer Object Container 5-55](#_Toc427311453)

[5.1.3 Transfer Object Group container 5-56](#_Toc427311454)

[5.1.4 Data Object container 5-58](#_Toc427311455)

[5.2 XFDU SIPs 5-60](#_Toc427311456)

[5.2.1 Linkage between descriptor IDs and SIP 5-63](#_Toc427311457)

[5.3 Non-XFDU SIPs 5-63](#_Toc427311458)

[6 Use Cases 6-1](#_Toc427311459)

[6.1 ISEE – A Typical Use Case 6-1](#_Toc427311460)

[6.2 ESA-SAFE – Transfer of SAFE products 6-1](#_Toc427311461)

[6.3 BnF and METS – A non-XFDU SIP Implementation 6-1](#_Toc427311462)

[6.4 CoRoT – End of Mission Bulk Transfer 6-1](#_Toc427311463)

[7 Software Tools 7-2](#_Toc427311464)

[7.1 CNES Prototype 7-2](#_Toc427311465)

[7.2 ESA SIP Builder 7-3](#_Toc427311466)

[ANNEX A Associated Descriptor Data Identifiers A-1](#_Toc427311467)

[ANNEX B ISEE Use Case Descriptors B-1](#_Toc427311468)

[ANNEX C ESA-SAFE use Case Descriptors C-1](#_Toc427311469)

[ANNEX D ESA-SAFE use Case Descriptors D-2](#_Toc427311470)

[ANNEX E BnF use Case Descriptors E-3](#_Toc427311471)

[ANNEX F COROT use Case Descriptors F-4](#_Toc427311472)

[INDEX F-1](#_Toc427311473)

Figure Page

[Figure 1‑1: Key to UML Relationships 1-6](#_Toc427310284)

[Figure 2‑1: Example of Transfer 2-1](#_Toc427310285)

[Figure 2‑2: Typical steps driving a PAIS Producer-Archive Project Definition 2-2](#_Toc427310286)

[Figure 3‑1: Model of Objects for Transfer (MOT) 3-4](#_Toc427310287)

[Figure 3‑2: Transfer Object Descriptor 3-5](#_Toc427310288)

[Figure 3‑3: Collection Descriptor 3-6](#_Toc427310289)

[Figure 3‑4: SIP Constraints 3-7](#_Toc427310290)

[Figure 3‑5: SIP Model 3-8](#_Toc427310291)

[Figure 3‑6: Relationships Between SIP Constraints and MOT 3-9](#_Toc427310292)

[Figure 3‑7: A Methodology for Modeling a Transfer 3-10](#_Toc427310293)

[Figure 4‑1: POLDER MOT 4-13](#_Toc427310294)

[Figure 4‑2: Example of PAIS XML documents in a directory 4-14](#_Toc427310295)

[Figure 4‑3: Snippet of MOT with Parent Collection Relations Identified 4-34](#_Toc427310296)

[Figure 4‑4: Snippet of MOT Showing Parent and Association Relationships 4-38](#_Toc427310297)

[Figure 5‑1: Abstract View of SIP, Transfer Object, Transfer Object Group, and Data Object 5-53](#_Toc427310298)

[Figure 5‑2: Combination of XFDU Schema and PAIS Schema to Form XFDU SIP Schema 5-61](#_Toc427310299)

[Figure 5‑3: XFDU PAIS SIP 5-62](#_Toc427310300)

[Figure 7‑1: Identification of PAIS elements used by the ESA SIP Builder 7-3](#_Toc427310301)

[Figure 7‑2: File and Folder Collectors 7-3](#_Toc427310302)

[Figure 7‑3: Example of Collectors supplying TC1 Groups and Data Objects 7-6](#_Toc427310303)

[Figure 7‑4: Collectors mapping from test data to SIP Groups and Data Objects 7-7](#_Toc427310304)

Table

[Table 4‑1: Example of a Descriptor for a Root Collection 4-15](#_Toc427309836)

[Table 4‑2: Example of Child Collection 4-17](#_Toc427309837)

[Table 4‑3: Example of Transfer Object Type for a series of L0 Products 4-17](#_Toc427309838)

[Table 4‑4: Example of Group Type 4-20](#_Toc427309839)

[Table 4‑5: Example of Data Object Type 4-23](#_Toc427309840)

[Table 4‑6: Definition of Occurrence Type 4-26](#_Toc427309841)

[Table 4‑7: A Bounded Number of Transfer Objects 4-27](#_Toc427309842)

[Table 4‑8: An Unlimited Number of Groups 4-27](#_Toc427309843)

[Table 4‑9: A Fixed Number of Data Objects 4-27](#_Toc427309844)

[Table 4‑10: An Optional Transfer Object 4-28](#_Toc427309845)

[Table 4‑11: A Denied Data Object 4-28](#_Toc427309846)

[Table 4‑12: Invalid Occurrence – Attempted Negative Bound 4-29](#_Toc427309847)

[Table 4‑13: Invalid Occurrence – Attempted Maximum Value Less Than Minimum Value 4-29](#_Toc427309848)

[Table 4‑14: Definition of Transfer Object Type Size 4-29](#_Toc427309849)

[Table 4‑15: Definition of Encoded Type 4-31](#_Toc427309850)

[Table 4‑16: Example of Group Type with Encoding 4-31](#_Toc427309851)

[Table 4‑17: Example of Data Object Type with Encoding 4-32](#_Toc427309852)

[Table 4‑18: Definition of Relation 4-33](#_Toc427309853)

[Table 4‑19: Definition of associationType Relation 4-35](#_Toc427309854)

[Table 4‑20: Example of Transfer Object Type with Associations 4-36](#_Toc427309855)

[Table 4‑21: SIP Constraints 4-39](#_Toc427309856)

[Table 4‑22: Example of SIP Constraints Content 4-40](#_Toc427309857)

[Table 4‑23: Example of SIP Constraints Content 4-41](#_Toc427309858)

[Table 4‑24: Example of Extended Collection 4-46](#_Toc427309859)

[Table 4‑25: Example of Restrictions 4-47](#_Toc427309860)

[Table 4‑26: Example of Restricted Type – Simple Type – String Patterns 4-48](#_Toc427309861)

[Table 4‑27: Example of Restricted Type – Simple Type – Enumeration Type 4-48](#_Toc427309862)

[Table 4‑28: Example of ?? 4-49](#_Toc427309863)

# Introduction

## Purpose And Scope

The purpose of this CCSDS report is to provide a tutorial for the Producer-Archive Interface Specification (PAIS) standard [1].

## Document Structure

This document is broken down as follows:

Section 1 defines the purpose, scope, structure, definitions for terminology and references to standards and external documents used in this CCSDS report;

Section 2 introduces the concept of transfer, the terminology inherited from PAIS and OAIS, and provides an overview and guidelines for building a Model of Objects for Transfer (MOT) and associated SIPs;

Section 3 introduces the PAIS XML descriptors and provides concrete examples covering all XML elements and provides best practices;

Section 4 describes the PAIS implementation of SIPs their generation, ingestion, validation, either as XFDU or non-XFDU packages;

Section 5 provides a series of use cases that enforce the understanding of the PAIS standard through concrete and complete examples

Section 6 introduces a series of existing software tools that may help implement the PAIS standard.

Annexes:

* Annex A addresses the Associated Descriptor Data Identifier. .
* Annexes B to E contain the descriptors, SIP constraints and extract of a SIP manifest from the 4 practical use cases NASA ISEE, ESA SAFE, CNES COROT, BnF METS.

## Definitions

### Acronyms and Abbreviations

For the purposes of this document, the following acronyms and abbreviations apply.

**AIP** Archival Information Package

**ASCII** American Standard Code for Information Interchange

**BnF** Bibliothèque nationale de France (French);  
French National Library (English)

**CCSDS** Consultative Committee for Space Data Systems

**CMC** CCSDS Management Council

**CNES** Centre National d'Études Spatiales

**CoRoT** COnvection ROtation et Transits planétaires (French);  
COnvection ROtation and planetary Transits (English)

**DFDL** DFDL

**DIP** Dissemination Information Package

**EO** Earth Observation

**ERS** European Remote Sensing Satellite

**ESA** European Space Agency

**FITS** Flexible Image Transport System

**GUI** Graphical User Interface

**ISEE** International Sun-Earth Explorer ~~(international cooperative program)~~

**MB** Mega Bytes

**METS** Metadata Encoding and Transmission Standard

**MOT** Model of Objects for Transfer

**NASA** National Aeronautics and Space Administration

**NSSDCA** National Space Flight Data Coordinating Archive

**OAIS** Open Archival Information System

**PAIS** Producer Archive Interface Specification

**PAIMAS** Producer-Archive Interface Methodology Abstract Standard

**PREMIS** Preservation Metadata Implementation Strategies

**SAFE** Standard Archive Format for Europe

**SAR** Synthetic Aperture Radar

**SIP** Submission Information Package

**SLA** Service Level Agreement

**SPAR** Scalable Preservation and Archiving Repository

**TIFF** Tagged Image File Format

**UML** Unified Modeling Language

**XFDU** XML Formatted Data Unit

**XML** Extensible Markup Language

**ZIP** [TBD]

### Glossary of Terms

PAIS terminology, as defined in reference [1] and reference [4], is used throughout this CCSDS Report. Only brief definitions are provided here.

**Archive**: An organization that intends to preserve information for access and use by a Designated Community.

**Content Information**: The set of information that is the primary target for preservation. It is an Information Object comprised of its Content Data Object and its Representation Information. An example of Content Information could be a single table of numbers representing, and understandable as, temperatures, but excluding the documentation that would explain its history and origin, how it relates to other observations, etc.

**Collection Descriptor**: A set of attributes that describes a view of a single collection of data and that identifies the parent collection of which it is a part.

**Data Object**: Either a Physical Object or a Digital Object.

**Data Object Type**: A set of characteristics describing a Data Object (such as the size of this object and the description of its content). Typically there will be multiple Data Objects conforming to the same Data Object Type.

**Descriptor**: Either a Collection Descriptor or a Transfer Object Type Descriptor.

**Descriptor Model**: A model that defines the mandatory and optional attributes needed for a Collection Descriptor or a Transfer Object Type Descriptor.

**Fixity Information**: The information which documents the authentication mechanisms and provides authentication keys to ensure that the Content Information Object has not been altered in an undocumented manner.

**Formal Definition Phase**: The Formal Definition Phase includes completing the SIP design with precise definitions of the digital objects to be delivered, completing the Submission Agreement with precise contractual transfer conditions such as restrictions on access and establishing the delivery schedule.

**Information**: Any type of knowledge that can be exchanged. In an exchange, it is represented by data. An example is a string of bits (the data) accompanied by a description of how to interpret a string of bits as numbers representing temperature observations measured in degrees Celsius (the Representation Information).

**Information Package**: A conceptual container composed of optional Content Information and optional associated Preservation Description Information. Associated with this Information Package is Packaging Information used to delimit and identify the Content Information and Package Description Information used to facilitate searches for the Content Information.

**Model**: A data entity described independently from any instance in a data product, and corresponding to a re-usable data entity definition, from which other data entities may inherit the attributes and apply some specialization rules.

**Model of Objects for Transfer (MOT)**: The set of all Descriptors for a given Producer-Archive Project. It is used jointly by the Producer and the Archive to provide a common and understandable hierarchical view of the Producer’s Data Objects to be transferred and their organization into collections, and it supports possible additional relationships among them. The hierarchy may be viewed as a tree having leaf and non-leaf nodes. The Data Objects to be transferred, organized as ‘Transfer Objects,’ are represented by the leaves of the MOT. Thus the nodes of the MOT have a different meaning depending on whether they are leaves or not:

* A leaf node corresponds to a single Transfer Object Type and therefore one exists for each Transfer Object Type Descriptor.
* A non-leaf node corresponds to a collection view of Transfer Object Types, or of a collection of collections. A non-leaf node exists for each Collection Descriptor.

**Producer**: The role played by those persons or client systems who provide the information to be preserved. This can include other OAISes or internal OAIS persons or systems.

**Producer-Archive Project**: A Producer-Archive Project is a set of activities and the means used by the information Producer as well as the Archive to ingest a given set of information into the Archive.

**Submission Information Package (SIP)**: An Information Package that is delivered by the Producer to the OAIS for use in the construction or update of one or more AIPs and/or the associated Descriptive Information.

**Transfer Object**: A set of one or more Transfer Object Groups containing at least one Data Object that are to be transferred to the Archive.

**Transfer Object Group**: A set of zero or more Data Objects and zero or more Transfer Object Groups.

**Transfer Object Type**: A set of characteristics describing a Transfer Object (such as the size of this object, the description of its content, and its makeup in terms of one or more Data Object Types). Typically there can be multiple Transfer Objects conforming to the same Transfer Object Type.

**Transfer Object Type Descriptor**: A set of attributes that describes a Transfer Object Type and that identifies the parent collection of which it is a part.

**Transfer Phase**: The Transfer Phase performs the actual transfer of the SIP from the Producer to the Archive and the preliminary processing of the SIP by the Archive, as it is defined in the agreement.

**Validation Phase**: The Validation Phase includes the actual validation processing of the SIP by the Archive and any required follow-up action with the Producer. Different systematic or in-depth levels of validation may be defined. Validations may be performed after each delivery, or later, depending on the validation constraints.

## Conventions

While the document is written in “Times New Roman” 12 points, the XML elements are written in “Courier New” 12 points.

Text in tables and figures may be in a different size font.

This document uses the UML convention for the figures. The object relationships are described in the UML diagrams below. This is extracted from Annex C of reference [4].



Figure 1‑1: Key to UML Relationships

A Class is indicated by a rectangle containing the Class name. The UML representation of a class is a three-compartment rectangle with name in the top compartment attributes in the second compartment and methods in the lowest compartment. In this document the attributes and operations compartments are always empty and UML states empty compartments can be suppressed.

Classes of objects are related to one another through Associations, and there are various multiplicities that may be attached to these associations as shown. The multiplicity refers to the number of instances, or objects, of that class that are involved in the relationship.

A solid line connecting two classes indicates the general association, among two classes. The line is labeled with an association name, indicating the nature of the association, and a solid arrowhead indicating the direction that the relationship should be read. The multiplicity of each class is shown next to the class near the association line. If the association forms a class that may have its own attributes or methods, that association class is shown as a rectangle connected to the solid line by a dashed line. The multiplicity may be omitted if the association is one to one.

There are two particular associations that are commonly used, aggregation and specialization, and these have particular symbols to indicate them.

An Aggregation association is one where a class is considered to be a part of another class. In UML, a diamond connecting the aggregation association to the aggregated class shows association. There are two types of aggregation defined by UML. Composition (sometimes referred to as strong aggregation), where the part classes are physically stored as part of the aggregated class, is shown with a solid diamond. In a Composition, if the aggregated class is destroyed, the child classes are also destroyed. Weak aggregation, where the part classes are referred to by the aggregated class, is shown with an empty diamond. In a weak aggregation, if the aggregated class is destroyed, the part classes are not destroyed and may be aggregated into other new classes. Composition can be thought of as aggregation by value, while weak aggregation can be thought of as aggregation by reference. In figure C‑1, the aggregation association says that the Assembly class contains exactly one Part-1 class instance and zero or more Part-2 class instances. Also if an instance Assembly is destroyed the Part-1 instance will continue to exist but all the Part-2 instances will be destroyed.

A Specialization association is one where a child class inherits attributes and methods from the parent class. In UML, a broad triangle connecting the aggregation association to the parent class shows specialization. An instance of a child class contains all the attributes and methods contained by its parent class, so an instance of the child class can be used in any operation where an instance of the parent class would be valid. However, the child class may add any number of new attributes or methods so an instance of a parent class is not necessarily a valid replacement for the child class. In the figure, the specialization association says that the Parent class attributes and methods are inherited by the Child-1 class and the Child-2 class.

## References

The following documents are referenced in this Report. At the time of publication, the editions indicated were valid. All documents are subject to revision, and users of this Report are encouraged to investigate the possibility of applying the most recent editions of the documents indicated below. The CCSDS Secretariat maintains a register of currently valid CCSDS documents.

[1] *Producer Archive Interface Specification (PAIS)*. Draft Recommendation for Space Data System Standards, CCSDS 651.1-R-1. Red Book. Issue 1. Washington, D.C.: CCSDS, October 2013.

[2] *XML Formatted Data Unit (XFDU) – Structure and Construction Rules*. Recommendation for Space Data System Standards, CCSDS 661.0-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, September 2008. [Equivalent to ISO 13527:2010.]

[3] *Producer-Archive Interface Methodology Abstract Standard (PAIMAS)*. Recommendation for Space Data System Standards, CCSDS 651.0-M-1. Magenta Book. Issue 1. Washington, D.C.: CCSDS, May 2004. [Equivalent to ISO 20652:2006.]

[4] *Reference Model for an Open Archival Information System (OAIS)*. Recommendation for Space Data System Standards, CCSDS 650.0-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, January 2002. [Equivalent to ISO 14721:2003.]

[5] *Organization and Processes for the Consultative Committee for Space Data Systems*. CCSDS Record, CCSDS A02.1-Y-3. Yellow Book. Issue 3.

[6] *CCSDS Publications Manual*. CCSDS Record, CCSDS A20.0-Y-3. Yellow Book. Issue 3. Washington, D.C.: CCSDS, December 2011.

# PAIS at a Glance

PAIS will enable the Producer to share with the Archive a precise and unambiguous formal definition of the Digital Objects to be produced and transferred. It does this by means of a model and its instantiation as XML files. It will also enable a precise definition of the packaging of these objects in the form of Submission Information Packages (SIPs), including the ability to specify the order in which they should be transferred.

A transfer, as seen by the PAIS standard, is the movement of Data Objects from a Producer to an Archive. The Data Objects are not transferred as independent plain items but rather grouped and encapsulated in higher level objects known as Submission Information Packages (SIPs), thereby providing better control in term of content types, fixity information, inter-relationships and sequencing as outlined in the following figure 2-1.

651x2g0-figure-2-1

Figure ‑: Example of Transfer

The Producer is responsible for the creation of SIPs according to content types agreed with the Archive and for their submission in a sequencing order that may also have been negotiated with the receiving Archive. In the example above, the Producer has generated and submitted four SIPs, one of Content Type A, the second of Content Type B and the remainders of Content Type C. As suggested by their names, the Content Types govern the actual content allowed for a SIP in term of structure and data format.

According to the PAIS standard the contents of the SIPs are decomposed into Transfer Objects (i.e. depicted as colored boxes in the figure 2-1 above) holding one or more trees of Groups (i.e. usually denoting folders) organizing the Data Objects that are the subject of the transfer (i.e. usually a single file or a set of files). A typical example of a Transfer Object could be an Earth Observation product composed of various metadata and data files (i.e. the Data Objects) organized in a tree of folders (i.e. the Groups). The PAIS standard supports the control of these objects through the description of their types, namely the Transfer Object Types, Group Types and Data Object Types.

According to the PAIS, the definition of these SIP Content Types is given by a “SIP Constraints” XML document that specifies the Transfer Object Types and their frequency of occurrence in each SIP Content Type. It can also specify the order in which SIP Content Types are transferred to the Archive. Referring again to Figure 2-1, SIP Content Type A may have specified that only ‘blue type’ Transfer Objects may be present. Content Type B may have specified that both ‘blue and red type’ Transfer Objects may be present. And finally the SIP Content Type C may require an “orange type” Transfer Object and one or more “green type” Transfer Objects. The SIP Constraints document may also have specified that SIPs of type A and type B must be transferred prior to all type C SIPs. Alternatively, there may be no constraints on the order of transfer.

PAIS provides a mechanism, called Transfer Object Descriptors, to formally define Transfer Object Types. These Descriptors are instantiated as XML documents. A Descriptor specifies a content tree comprised of Group Types possibly holding other Group Types and/or Data Object Types. It also supports the specification of occurrences, sizes and association between types.

In addition, the PAIS standard specifies the attributes that must be included in a SIP for the complete typing of all the objects it contains. These attributes link the objects (Transfer Objects, Group Objects, Data Objects) to their specifications within the PAIS descriptors. The PAIS standard also defines a default SIP format based on the CCSDS XFDU recommended standard (see reference [2]). According to the XFDU standard, the SIPs are containers of any type (e.g., usually a ZIP archive or a root folder), that hold the Data Object files organized as an arbitrary number of nested folders. This structure is accompanied by an XFDU Manifest XML document that registers all the Data Objects and, when specialized as defined by the PAIS, uniquely identifies their types within a PAIS Producer-Archive Project (i.e. identifies the PAIS Data Object Types, Group Types, Transfer Object Types, SIP Content Type, etc.)

The list of methods for writing PAIS descriptors is endless and it is likely that none is suitable for all project contexts. Nevertheless, the following workflow gives an overview of the major steps that are usually to be addressed during a Producer-Archive Project definition:

651x2g0-figure-xxx-method-overview

Figure 2‑2: Typical steps driving a PAIS Producer-Archive Project Definition

In summary, Producer-Archive Projects can increase Archive ingest efficiency and quality control by adopting the PAIS standard. The Producer Archive Project will need to write a set of XML documents according to a formal XML language.to model the transfer. This facilitates validating these descriptors against XML Schema documents provided in the annex of the standard. It also enhances the understanding of the transfer project by both the Producer and the Archive.

If the Producer-Archive Project uses that model and adopts the PAIS standard for packaging the data into SIPs for transfer, they will gain benefits when developing or reusing tools for building, transferring, receiving and validating SIPs.

# Modeling Transfers

The PAIS standard in reference [1] specifies material for the modeling and control of the transfer of Digital Objects from a Producer to an Archive. This material consists of a set of XML language specifications and construction rules supporting the concrete implementation of the Producer-Archive Interface Methodology Abstract Standard (PAIMAS), itself deriving from the Reference Model for an Open Archival Information System (OAIS) (see references [3] and [4]). The XML schemas defined by the PAIS do not intend to implement all aspects of the broad PAIMAS abstract standard but are focused on the specification of the formal model and the associated SIPs during the Formal Definition Phase, the validation and follow-up of the SIPs during the Transfer Phase and the Validation Phase as defined in PAIMAS.

Note: The figures in this section employ UML modeling to give an overview of the PAIS modeling using XML based Descriptors.

## Model of Objects for Transfer

The Model of Objects for Transfer (MOT) is a set of PAIS XML descriptors controlling a Producer-Archive Project and is an instantiation of the PAIMAS formal model. The UML diagram in Figure 3-1 shows that the MOT must be composed of at least one Collection Descriptor and one Transfer Object Descriptor.

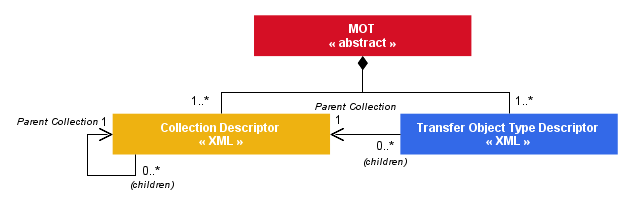


Figure ‑: Model of Objects for Transfer (MOT)

Each Transfer Object Type Descriptor models a unit of transfer broken down in a tree of Group Types and sub-Group Types of Data Objects. The Transfer Object Type Descriptor is further described in section 3.1.1.

The Collection Descriptors organize the Transfer Object Types in a logical tree with unlimited levels of parent Collections. The Collection Descriptor is further described in section 3.1.2 below.

According to this definition, the minimal MOT is composed of two XML files and defines one Transfer Object Type as a part of one Collection.

### Transfer Object Type Descriptor

A Transfer Object Type Descriptor defines a Transfer Object Type. Objects conforming to the Descriptor can be transferred from the Producer to the Archive. Each Descriptor defines one and only one Transfer Object Type and it has to be part of a Collection.

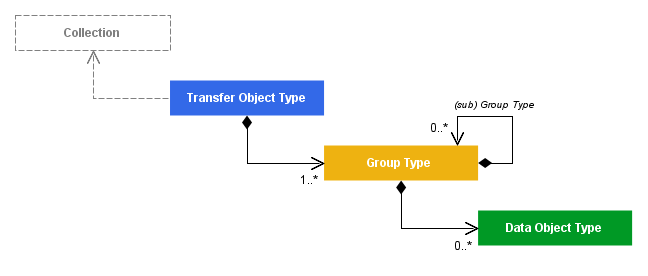


Figure ‑: Transfer Object Descriptor

A Transfer Object Type is uniquely identified across the overall Producer-Archive Project and decomposes the object into one or more trees of Group Type nodes and Data Object Type leaves. A Transfer Object Type defines the minimal and maximal number of objects, or instances, of this type that could occur in the overall project. It may optionally define the minimal and maximal size of each individual object of this type. The complete specification of the Transfer Object Type is provided in section 3.2 of PAIS standard (see reference [1]) and illustrated throughout the section 4 of this report.

A Transfer Object Type must contain at least one Group Type, all accepting as many sub-group types as required up to the Data Object Type leaves. The Group Type definition specifies the minimal and maximum number of occurrences expected within its parent Group or its parent Transfer Object. The Group Types may correspond to directories or ZIP entries, or they may simply represent the concept of a set or sequence. It is important to note that when there are nested groups with multiple occurrences, the order in which the group instances should be instantiated is ambiguous. This should be addressed by specifying the order semantically in the text description that is a part of each group specification. It could also be addressed by incorporating one or more user defined attributes into the Transfer Object Descriptor. This issue is addressed further in section x and the ISEE use case of section 6.1.

The Data Object Type is the lowest level of description of the MOT and usually corresponds to a single file type. It may also represent multiple files if this set of files can be considered as a single Data Object at the transfer level. This feature enables the avoidance of overly detailed Descriptor specifications. For example, it may be convenient to consider an Earth Observation product as a single Data Object although it is composed of multiple files. The modeling of the header files, the image bands and other auxiliary files composing this product may not be of interest if they are never disjoint and never referenced individually during the transfer. The Data Object Type specifies the minimal and maximal number of occurrences expected within its instance of parent Group Type. It is also possible to control the minimal and maximal number of ccfiles composing a Data Object.

Any of the Transfer Object Types, Group Types and Data Object Types can be interrelated and/or associated to a Collection. The semantics of an associations are unconstrained by PAIS and are to be agreed between the Producer and the Archive during the Project definition. Typical semantics include the data/metadata relationship or the association with a Representation Information that may support the cataloguing and the validation activity on the Archive side. These relations are further discussed in section 4.5 of this report.

### Collection Descriptor

Each Collection Descriptor defines a single Collection. Each Collection Descriptor references a single parent Collection. The parent Collection of the top level Collection has the value ‘none’. Any Collection can be referenced as a parent by zero or more other Collections and any number of Transfer Object Types. Thus the set of Collections defined in a MOT forms a tree.

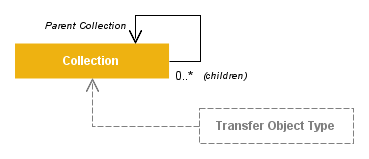


Figure ‑: Collection Descriptor

A Collection is a logical grouping concept whose semantics are to be determined by the Producer and the Archive. For example, the Collection may support Archive behavior in terms of cataloguing, validation means or storage. Typical Collections could distinguish project documentation from data production, could group products according to their processing levels, the production phases, etc. The complete specification of Collection is provided in section 3.3 of PAIS standard (see reference [1]) and illustrated throughout the section 4 of this report.

## Submission Information Package (SIP)

The Submission Information Package (SIP) is the actual physical unit of transfer that can hold one or more Transfer Objects. The PAIS standard defines a SIP Constraints file and various rules for specifying and controlling the SIP contents based on the MOT elements introduced in the sections above.

### SIP Constraints

The SIP Constraints file is an XML document defining all the SIP Content Types that may occur in a Producer-Archive Project. At least one SIP Content Type must be defined for the project. If more than one SIP Content Type is defined, then the sequencing of transfers to the Archive for those SIP Content Types can be specified.

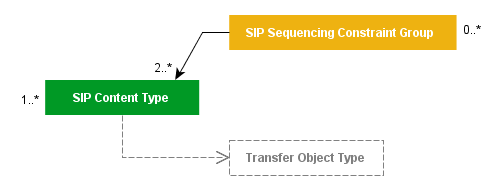


Figure ‑: SIP Constraints

Each SIP Content Type specifies the Transfer Object Types authorized for this type of SIP and, optionally, their minimal and maximal numbers of occurrences allowed for each SIP instance.

The SIP Constraints file may also define one or more SIP Sequencing Constraint Groups that impose a sequencing order between two or more SIP Content Types. This feature may be convenient to secure, for example, the prior transfer of Representation Information required for the validation of the successive packages.

### SIP Model

The PAIS standard defines an abstract SIP Model that has to be followed by any PAIS compliant SIP. Unless specified otherwise, any reference to SIP in this report is supposed compliant to the PAIS SIP Model.

According to this SIP Model, a SIP is composed of one mandatory SIP Global Information section and optional Transfer Objects to Delete and Transfer Object sections as outlined in the figure 3-5 below. At least one of the two sections is mandatory.

Description: 651x2g0-figure-2-6.emf

Figure ‑: SIP Model

The SIP Global Information includes the identification of the SIP with respect to the Producer-Archive Project, the actual source that has generated this SIP and, in particular, a reference to the SIP Content Type it instantiates. The SIP Global Information may also include a sequence number necessary for disambiguation when the MOT descriptors do not provide fixed occurrence values.

The Transfer Objects are instances of Transfer Object Types of the Producer-Archive Project’s MOT. A Transfer Object in a SIP must be of a Transfer Object Type authorized by the SIP Content Type referenced by the SIP Global Information. Similarly, the number of Transfer Objects of an authorized type must be within the range defined in the SIP Content Type.

Following the type hierarchy, the Transfer Objects are composed of Transfer Object Groups, that are instances of Group Types, and Data Objects, that are instances of Data Object Types. The actual instances must explicitly reference the corresponding MOT types (i.e., Descriptor specifications) to allow the formal validation of the SIP structure at ingestion.

Finally, the Data Objects reference one or more Byte Streams typically instantiated as physical files.

### relationships between sip constraints and mot

The figure below is an overview of the MOT, SIP Constraints and links between them defined during the Formal Definition Phase, as detailed in this section “Modeling Transfers”.

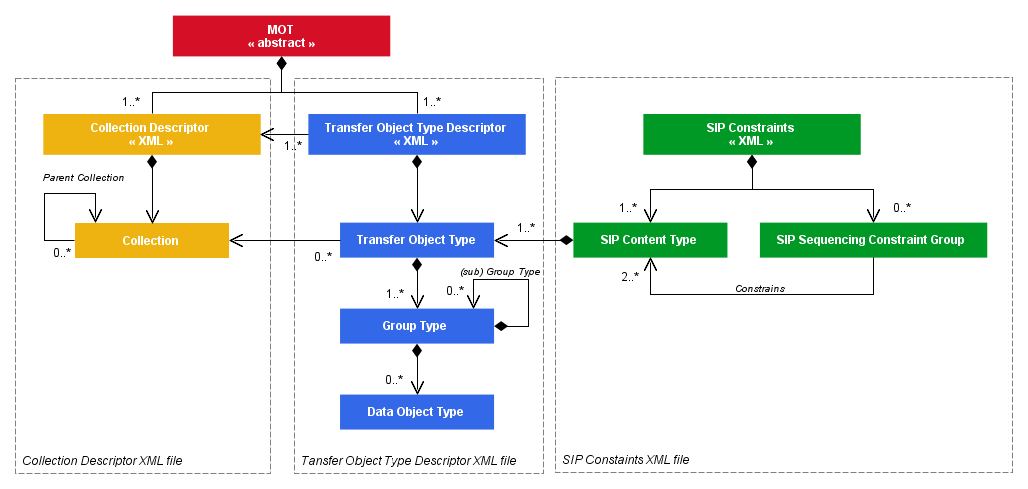
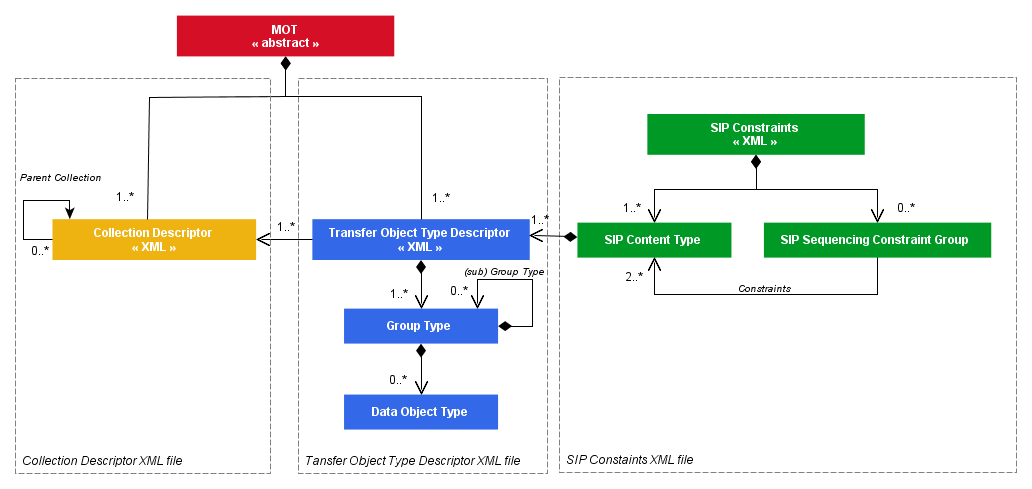


Figure 3‑6: Relationships Between SIP Constraints and MOT

## A Methodology for Modeling a Transfer

As introduced in the section 2 above, there are probably countless methods that could apply for defining the PAIS descriptors, but it is likely that none is suitable for all project contexts. However, the following workflow diagram introduces a typical methodology illustrating major steps that most implementers could follow.

651x2g0-figure-3-6

Figure ‑: A Methodology for Modeling a Transfer

The workflow steps can be summarized as follows:

* **Define Project Context**: according to PAIMAS standard (see reference [3]), this step represents a possibly long preliminary phase during which the Producer and the Archive converge towards a common understanding of the transfer. At the PAIS level, the critical outputs are the Producer-Archive Project identifier and the potential Producer Source identifiers that will be necessary in the header of all SIPs;
* **Define Content Information**: at this step the Producer and the Archive agree on what kind of Digital Objects are to be transferred. From this step, the workflow forks in two branches involving the Producer and the Archive separately;
* **Estimate the Data Volumes**: at this step the Producer determines the volumes of each type of Digital Objects. This may depend on the variety of formats, of data sources, etc.
* **Assess AIPs/DIPs**: at this step the Archive has to consider how and where it will host/store the Digital Objects in term of Archival Information Packages (AIPs). A preliminary analysis of the Dissemination Information Packages (DIPs) is usually a good practice for the proper design of AIPs. This step may also identify the validation procedures that will be required at ingestion to populate the archive and ensure adequate quality;
* **Identify Project Constraints**: this steps consists in merging the information collected during the two previous parallel steps and ensuring that all parties can handle the planned transfer e.g. network resources, hardware, manpower. The objective is to identify the project constraints that can influence the modeling of the objects to be transferred in terms of grouping, sizing and sequencing;
* **MOT Design**: at this step, all the project information (e.g. data objects, formats, collections etc.) and constraints are known and it becomes possible to model the transfer and write the PAIS descriptors for the transfer project i.e. the Collection descriptors and the Transfer Object Type descriptors. It is recommended at this step to model only the elements that are actually required or justified for the transfer. For example, the PAIS does not require Descriptors that may serve as complete Representation Information outside the context of a transfer. Thus, modeling all files directories because they are there may be more cumbersome than useful and should not be considered as a good practice;
* **Model SIP Constraints**: this step consists in writing the SIP Constraints XML document that defines the SIP Content Types allowed in the transfer and potentially the sequencing constraints e.g. SIPs holding data specifications or auxiliary data may be required first to allow the validation or the reprocessing of the primary data;
* **Validate MOT and Constraints**: at this decision step the PAIS descriptors and the SIP Constraints XML documents are validated. They have to be well-formed according to the XML recommendation and validated against the PAIS XML Schemas provided in the annex of the standard. It is recommended to complete this automatic validation with reviews involving both the Producer and the Archive and to simulate the transfer of all potential SIP Content Types in an environment as close as possible to the target operational one. In case of failure, the workflow must be reiterated from the “MOT Design” step.

# Writing XML Descriptors And SIP Constraints

This section describes the XML descriptors composing the MOT (Collections and Transfer Object Descriptors). A basic understanding of the MOT concepts introduced in the section 3.1 above is required. This section includes examples based on the POLDER Data Set, described section 4.1.1.

## Structures and Construction Rules

### POLDER – a high level description

The POLDER data example is used to highlight typical points being described in the tutorial. This subsection presents a high level description of the POLDER data without further details ( as this amount of understanding should be sufficient for the readers’ understanding of the examples.

POLDER (POLarization and Directionality of the Earth's Reflectances) was developed by the French space agency, CNES.

POLDER is a wide field of view imaging radiometer that has provided the first global, systematic measurements of spectral, directional and polarized characteristics of the solar radiation reflected by the Earth/atmosphere system. Its original observation capabilities have opened up new perspectives for discriminating the radiation scattered in the atmosphere from the radiation actually reflected by the surface.

The schematic of the MOT for the POLDER use case, as shown in Figure 4-1, involves two Collection Descriptors –one for the raw data called “L0”, one for the upper level product called “L1”, and one Transfer Object Descriptor for the documentation.

The L0 collection contains a Transfer Object Descriptor for the raw data “L0DATA”, the associated Representation Information “L0REPINFO” and the auxiliary data “AUXDATA”, used to build the upper level products “L1”. For the example only 3 products are modelled.

The L1 collection contains three Transfer Object Descriptors, one for the L1 products “LIG1DATA”, one for the associated browses “L1GB1DATA”, one for the L1 products Representation Information “L1REPINFO”.

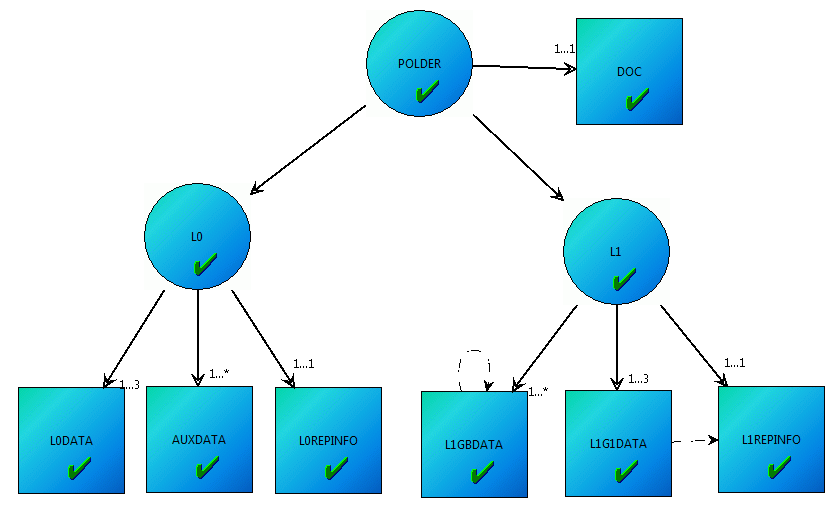


Figure 4‑1: POLDER MOT

### Organization Of XML Documents

The physical organization of XML documents is basically free e.g. as files or database entry, in the same directory or in separate ones.

Figure 4-2 shows an example of a set of XML files grouped in a directory. Files are named using the naming convention of

<project-id>-pais-<type>[-<id>].xml

Where:

<type> can either be collection, transfer-object or sip-constraints;

[-<id>] is either the descriptorId of the Collection or Transfer Object Descriptors or nothing for SIP Constraints files that is unique.

├── polder-pais-collection-l0.xml

├── polder-pais-collection-l1.xml

├── polder-pais-collection-polder.xml

├── polder-pais-sip-constraints.xml

├── polder-pais-transfer-object-auxdata.xml

├── polder-pais-transfer-object-l0data.xml

├── polder-pais-transfer-object-l0repinfo.xml

├── polder-pais-transfer-object-l1g1data.xml

├── polder-pais-transfer-object-l1gbdata.xml

├── polder-pais-transfer-object-l1repinfo.xml

└── polder-pais-transfer-object-doc.xml

Figure ‑: Example of PAIS XML documents in a directory

### XML Namespace

PAIS descriptors are standard XML 1.0 documents that should begin with a standard XML prolog:

<?xml version="1.0" encoding="UTF-8"?>

XML Namespaces provide a method to avoid element name conflicts. A so-called **namespace** must be defined. The namespace is defined by the **xmlns attribute** in the start tag of an element. Any sub-element inherits the xmlns attribute – and so the namespace). The namespace declaration has the following syntax. xmlns[:prefix]="URI" (prefix is optional).

The current version of the PAIS produces elements in the “urn:ccsds:schema:pais:1” namespace:

Xmlns[:pais]="urn:ccsds:schema:pais:1"

#### Default namespace

The "urn:ccsds:schema:pais:1"namespace could be used as the **default namespace** i.e. non-prefixed PAIS elements:

<?xml version="1.0" encoding="UTF-8"?>

<collectionDescriptor **xmlns**="urn:ccsds:schema:pais:1">

<identification>

<descriptorModelID>CCSD0015</descriptorModelID>

[…]

</collectionDescriptor>

In this example, the collectionDescriptor and all sub-elements *implicitely* belongs to the urn:ccsds:schema:pais:1 namespace - even if not prefixed.

#### Namespace prefix

The “urn:ccsds:schema:pais:1” namespace also could be used with explicit **namespace prefix** i.e. preference to “pais” prefix but any other token would work (this may be useful to distinguish PAIS elements from other from different namespaces - only valid for extended PAIS descriptors):

<?xml version="1.0" encoding="UTF-8"?>

<**pais**:collectionDescriptor **xmlns:pais**="urn:ccsds:schema:pais:1">

<**pais**:identification>

<**pais**:descriptorModelID>CCSD0015</**pais**:descriptorModelID>

[…]

</**pais**:collectionDescriptor>

In this example, the collectionDescriptor explicitly belongs to the urn:ccsds:schema:pais:1 namespace.

#### Default namespace VS Namespace prefix

As much as possible, it is recommended to use the **default namespace** (especially because this form is easier to read by humans).

The prolog and namespace declarations will not be repeated along the examples of this section. Tabular representation is preferred.

### CollectionS

One XML document per Collection Descriptor must be created.

Any Collection must have a parent Collection, so each Collection Descriptor will include a single parentCollection element.

Table 4-1 shows a snippet of the contents of an example of a root collection descriptor: POLDER Collection. You can identify that this is the root collection since the value of the parentCollection element is “NONE” as is shown at **➋** in the table.

Table ‑: Example of a Descriptor for a Root Collection

|  |  |
| --- | --- |
| **Element** | **Sample Value** |
| **collectionDescriptor ➊** |  |
| identification |  |
| descriptorModelID | CCSD0015 |
| descriptorModelVersion | V1.0 |
| descriptorID | POLDER |
| description |  |
| collectionTitle | POLDER 1 and 2 Products |
| collectionDescription | POLarization and Directionnality of the Earth's Reflectance |
| Relation |  |
| **parentCollection ➋** | **NONE** |

All elements in Table 4-1 are mandatory. This example corresponds to the minimal set of elements required for a collection. Collection Descriptors accept a few more optional elements but this example would fit most use cases.

The descriptorModelID and descriptorModelVersion uniquely identify the model on which this Descriptor is based. It may be the Descriptor model as given in the PAIS standard, in which case the values are CCSD0015 and V1.0 respectively as shown in Table 4-1. However the Descriptor model can be specialized, either by CCSDS within a revised standard in which case one or both of the elements will be given new values, or by the Archive in which case the values and specifications need to be managed by the Archive and agreed by the Producer.

The CCSDS will update only the descriptorModelVersion when the updated schema is backward compatible with the original schema (i.e., changes are simply restrictions of semantics or syntax). Otherwise CCSDS will update the descriptorModelID. In order to maintain unambiguous responsibility for model ownership, only CCSDS is allowed to update the desciptorModelVersion value in association with the CCSDS defined descriptorModelD values of the form CCSDxxxx.

Should an Archive decide to make any changes to the schema as given by the PAIS standard, it must use a new value for the descriptorModelID that is not of the form CCSDxxxx. It is recommended that the Archive use the same rule of backward compatibility in the management of their schema’s model identifier and version values.

Note: All identifiers of the form CCSDxxxx, along with their specification and meaning, can be found in CCSDS standards.

The descriptorModelID and descriptorModelVersion are defined in the PAIS XML Schema as unrestricted xsd:string allowing empty strings, tabs etc. A good practice would be to restrict these elements to the values expected for the Producer-Archive project as described in section 4.7 below.

The descriptorID uniquely identifies the Collection within the Producer-Archive project.

The descriptorID is defined in the XML Schema as an xsd:string which authorizes empty strings or whatever content of any length. As for descriptorModelID, a recommended practice is to restrict the string to a controlled space, for example through a pattern or an enumeration valid for the overall project. It should reject empty strings. See section 4.7.

The collectionTitle and collectionDescription can greatly help the understanding of the purpose and scope of the Collection and should be written with care. Empty strings or the duplication of the Collection identifier is to be considered as a poor practice. Formal projects could allow restriction of the PAIS XML Schemas to prevent those situations.

Table 4-2 presents an example of a child collection. Its descriptorID value is L0 and it can be referred to as the L0 collection. As shown at **➊,** the relation/parentCollection element identifies the parent Collection through its descriptorID - POLDER.

Table ‑: Example of Child Collection

|  |  |
| --- | --- |
| **Element** | **Sample Value** |
| **collectionDescriptor** |  |
| **Identification** |  |
| **descriptorID** | L0 |
| *… removed for brevity …* |  |
| Relation |  |
| **parentCollection ➊** | **POLDER** |

Only a single parentCollection can be defined. However other associations can also be defined. A collection can be associated with other collections or other PAIS entities. Additional discussion of object relationships can be found in section 4.5.

Collections are logical entities that by default do not have a physical representation in the actual transfer. They are simply referenced by Transfer Objects that are part of the collection in order for the receiving archive to understand which piece of the transfer they are receiving.

### Transfer Objects

One XML document per Transfer Object Type Descriptor must be created.

Any Transfer Object must have a parent Collection, so each Transfer Object Type Descriptor will include a single parentCollection element.

Unlike with Collection Descriptors, Transfer Object Type Descriptors also serve as type definitions for the instantiation of one or more Transfer Objects that are to be incorporated into SIPs and sent to an Archive. Each such Transfer Object within a SIP will include a pointer whose value is the descriptorID of the corresponding Transfer Object Type Descriptor, thus linking each Transfer Object in the SIP to its type definition and allowing the recipient of the SIP to understand the Transfer Objects. This is discussed in more detail in section 5.1.1.

Table 4-3 is an example of a Transfer Object Type Descriptor for POLDER Level 0 products consisting of 1 to 3 Transfer Objects.

**Table 4‑3: Example of Transfer Object Type for a series of L0 Products**

|  |  |
| --- | --- |
| **Element** | **Sample Value** |
| **transferObjectTypeDescriptor ➊** |  |
| identification |  |
| descriptorModelID | CCSD0014 |
| descriptorModelVersion | V1.0 |
| producerSourceID | CNES |
| descriptorID | L0DATA |
| description |  |
| transferObjectTypeTitle | POLDER Level 0 Transfer Object |
| transferObjectTypeDescription | A single POLDER Level 0 product (TAR) |
| transferObjectTypeOccurrence |  |
| minOccurrence | 1 |
| maxOccurrence | 3 |
| *namePreservationRule* | *Producer to use the source names* |
| relation |  |
| parentCollection | L0 |
| groupType |  |
| groupTypeID | L0GROUP |
| groupTypeStructureName | directory |
| groupTypeOccurrence |  |
| minOccurrence | 1 |
| maxOccurrence | 1 |
| dataObjectType |  |
| dataObjectTypeID | L0DATAOBJECT |
| dataObjectTypeOccurrence |  |
| minOccurrence | 1 |
| maxOccurrence | 1 |

The identification section is the same as for the Collection identification section described previously except for the additional optional element producerSourceID.

The producerSourceID uniquely identifies an entity of the Producer that has provided the Transfer Object.. When used, an XML restriction to an enumeration may be defined to ensure that only valid values are provided. This identification section allows the Archive to uniquely identify the sending Producer entity should the need for communication arise.

The description section is similar to the corresponding section in the Collection for the collectionTitle and collectionDescription. However because the Transfer Object Type Descriptor is a type definition, the additional transferObjectTypeOccurrence element controls the number of occurrences of the Transfer Objects of this type **within the overall Producer-Archive project**. Refer to section 4.3 for further explanations about occurrences.

The optional namePreservationRule is provided to allow a statement as to how the Producer and the Archive should treat the names used for the Data Objects within the Transfer Objects derived from a given Transfer Object Descriptor, including how the names should be preserved by the Archive. In the example of Table 4-3, the element’s value is the statement ‘Producer to use the Source Names’. The Producer understands that ‘Source Names’ refers to the names used in the Producer’s environment. Since each Data Object will exist as one or more files in the Producer’s environment, this rule instructs the Producer to use each file’s existing name when instantiating each file of the Transfer Objects corresponding to the subject Transfer Object Descriptor. The existence of the rule means the receiving Archive must preserve each file name in the received Transfer Objects, although how it does this is at the discretion of the Archive. Most likely it would store the files with their received names. If this element were not included in the Descriptor, technically the Archive could preserve the files using some local naming convention unless instructed otherwise via an external agreement.

As another example of the namePreservationRule, its value may be the statement ‘Archive is required to maintain the file name extensions as received’. This may be appropriate for multi-file Data Objects where the file extensions are used to distinguish the file types of the Data Object. This does not constrain the Producer, as in the previous example, but instead the Archive is constrained to preserve the file extensions but not necessarily the file names apart from their extensions. How it does this is at the discretion of the Archive.

As a third example, the Archive may be asked to alter the file names following an algorithm given in the statement of the rule. This may occur when the Producer and the Archive agree that the names used in the Producer’s environment are not user friendly for the general user of the Archive. This could cause problems with documentation so this would have to be carefully addressed.

The namePreservationRule is defined in the XML schema as xsd:string.

The relation/parentCollection element identifies the parent Collection through its descriptorID. The example in Table 4-3 states that there is a parent Collection Descriptor whose descriptorID is L0, which is the example child collection of Table 4-2.

The parentCollection is defined in the XML schema as xsd:string.

The relation section can also contain other elements that allow this Transfer Object Type to be associated with other Transfer Object Types and with other Collections. Refer to section 4.5 for further explanations about relations.

As noted in section 3.1.1, a Transfer Object Type is broken down into a tree of groupTypes and dataObjectTypes. It contains at least one groupType.

The goupType element and its children are described in the following section 4.1.7.

The dataType element and its children are described in the following section 4.1.7.

### Transfer Objects – Group

Every Transfer Object Type Descriptor must have at least one groupType element. Each groupType element may contain any number of additional groupType elements and an unlimited number of dataObjectType elements. These elements are convenient for modeling hierarchies such as directory structures.

The groupType is specified in the XML schema and is of type transferObjectGroupType.

Table ‑: Example of Group Type

|  |  |
| --- | --- |
| **Element** | **Sample Value** |
| transferObjectTypeDescriptor |  |
| *… removed for brevity …* |  |
| **groupType ➊** |  |
| groupTypeID | L0GROUP |
| *groupTypeDescription* | *Level 0 Group Type* |
| groupTypeStructureName ➋ | directory |
| groupTypeOccurrence |  |
| minOccurrence | 1 |
| maxOccurrence | 1 |
| ***groupType*** ➌ |  |
| *… removed for brevity …* |  |
| ***dataObjectType*** ➍ |  |
| *… removed for brevity …* |  |
| ***dataObjectType*** ➎ |  |
| *… removed for brevity …* |  |

Table 4-4 provides an example of groupType for the POLDER example.

The groupTypeID is required and uniquely identifies the group, within the scope of the Producer-Archive Project, for reference purposes and is given as L0GROUP in this example.

The groupTypeID is defined in the XML schema as xsd:string.

The optional groupTypeDescription provides the ability to have a meaningful description as to what the group means or how it is being used.

The groupTypeDescription is defined in the XML schema as xsd:string.

The groupTypeStructureName is required and provides a semantic identification as to how the structure of the group is organized.

Although groupTypeStructureName is defined in the XML schema as xsd:string, it is recommended that it be restricted to a limited enumeration. The standard has predefined the four values of ‘directory’, ‘set’, ‘sequence’, and ‘undefined’, and it is recommended that they be lowercase for interoperability purposes. Their meanings are as follows:

Directory: A groupTypeStructureName with the value ‘directory’ is understood to be a directory structure with a name, corresponding to a physical file system folder or equivalent as folders in a ZIP or Tar. All instances of dataObjectTypes (i.e, Data Objects) and other groups at the same level within this groupType, are understood to be within this directory. Table 4-4 shows that the POLDER Transfer Object is viewed as a directory structure.

Set: A groupTypeStructureName with a value of ‘set’ is understood to state that the groupType holds an unordered grouping of all the instances of dataObjectTypes (i.e., Data Objects) and all the instances of groups at the same level, and nested immediately within, this groupType. A ‘set’ may or may not have a name. For example, the Data Objects could be several files related to observations of an event and there could be an included groupType, also specified as a ‘set’, holding additional files related to calibration of the observing instrument.

Sequence: A groupTypeStructureName with a value of ‘sequence’ is understood to state that the groupType holds an ordered grouping of either all the instances of dataObjectTypes (i.e., Data Objects) or all the instances of groups at the same level, and nested immediately within, this groupType. It is not permitted to mix groups and Data Objects under the same sequence and therefore, for example, a groupType with a groupTypeStructureName of ‘set’ and containing a dataObjectType can not also contain a groupType. As an explicit example, the Data Objects could be a images of solar eruptions taken over 20 years and ordered by the size of the eruption from smallest to largest. It is recommended that the optional groupTypeDescription be used to specify how the ordering is to be recognized. As an example for a sequence of Data Objects, they may be named alphabetically in the defined order.

Undescribed: A groupTypeStructureName with a value of ‘undescribed’ is understood to state that the detailed modeling of the groupType and all data structures instantiated under this groupType in the SIP have intentionally been left undescribed. Such a groupType is not allowed to contain any dataObjectTypes or other groupTypes, although all other elements of groupType may be present. This allows the modeling of Transfer Objects of complex structure to be terminated when the PAIS formal modeling effort become impractical or other modeling specifications already exist. For example, if the Transfer Object is a complex directory structure, it may be sufficient to model only the first directory level and to treat nested groupTypes as ‘undescribed’ because specifications for the directory structure exist elsewhere. This could give an Archive sufficient information to recognize and partially validate the receipt of an expected Transfer Object without requiring an onerous modeling effort by the Producer. The Producer must select, when creating the Transfer Object for incorporation into a SIP, the data that is to be included in the ‘undescribed’ groupType. Upon receipt of the SIP and encountering a Transfer Object with an ‘undescribed’ groupType, the Archive is expected to preserved all data found to be associated with this groupType, and to maintain its relationship with the rest of the Transfer Object.

The PAIS puts only two constraints on the mixing of groupTypeStructureName values that may be incorporated into a hierarchy of groupTypes within a Transfer Object. One exception is the use of ‘sequence’ for Data Objects that stops any further hierarchy under this groupType and therefore any further groupTypeStructureName values. The other exception is the use of ‘undescribed’ in a groupType that stops any further modeling of possible hierarchy under this groupType and therefore any further groupTypeStructureName values. This modeling flexibility is available to describe the Transfer Object as it will appear in a SIP. It should be noted that it may or may not correspond to the organization of the data in the Producer’s environment and it does not require that the described representation be maintained as the actual organization in the Archive environment. In other words, the structures described are the structures to be used for the SIP transfers and need not be the same as the physical implementations at either the Producer or the Archive site. For example, the tree of groups may be interleaved with groups not present in the Producer environment’s physical structure. The tree of groups may also skip some levels of the Producer environment’s physical structure. This was discussed briefly in section 3.2.3 and is present in the use case example of section 6.1. However for convenience, the structures used for transfer may, if desired, correspond to the physical structure at the Producer’s site, or may, if desired, match the expected organization at the Archive.

As noted in section 3.1, there is ambiguity in the order of instantiation when a tree of groups includes multiple instances of the groups. This can be resolved by incorporating the needed semantics within the groupTypeDescription. See also section 6.1 for a use case example.

The groupTypeOccurrence is required and specifies the number of instances of this groupType that should be present in the Transfer Object. This may be expressed as one, or more, or as a range of values. This number may not be known at the time of specification. In the case of a unique value, the minimal occurrence and the maximal occurrence values will be the same.

The groupTypeOccurrence is defined in the XML schema as type occurrenceType and is a complexType with the three elements minOccurrence of type xsd:nonnegativeInteger, maxOccurrence of type xsd:nonNegativeInteger, and maxUnknown of type anySimpleType.

Table 4-4 shows an occurrence of the groupType of exactly one. A more complete discussion is presented in section 4.3

Table 4-4 does not show all the PAIS defined elements that may be present in a groupType. The Transfer Object data associated with the groupType may be encoded, such as via a compression algorithm. It may be used iteratively to describe possible nested encodings, in which case they are described in the order of application. The result of encoding is a single Data Object instantiated as a single file in the Transfer Object. The possible presence of Data Object (dataObjectType) and any additional groupType specifications under this groupType specification is understood to comprise a detailed model of this group after the encoding has been reversed. With the use of this encoding element, validation of the encoded content may not be possible because the resulting data structure will not carry any of the identifiers associated with the detailed modeling of this group. Note that a groupType that is specified to be encoded may also be specified to be ‘undescribed’. In this case there is no modeling of the result of reversing the encoding. Encoding of groups is supported using the complex element groupTypeEncoded of type pais:encodingType as defined in the XML schema. Details are provided in section 4.4 below.

A groupType may also establish a relation with other entities within the Producer-Archive project. For example, it could be of interest to establish the relation ‘described by’ between the top level POLDER directory and a document describing the directory structure, provided in a different POLDER Collection. (Note: not actually true for the POLDER use case) This is supported using the complex element groupTypeAssociation of type associationType as defined in the XML schema. Details are provided in section 4.5 below.

A groupType may include any number of other groupTypes. Table 4-4 shows that the POLDER Transfer Object has nested groupTypes consistent with a directory structure.

A groupType may also have any number of dataObjectTypes. A dataObjectType is used to describe a single file, or a few files that are typically used together such as in a standard format. Table 4-4 shows that each groupType includes at least one dataObjectType. The dataObjectType is described in detail in section below.

### Transfer Objects – Data Objects

A Data Object is the lowest level of modeling in the Transfer Object Descriptor and is typically instantiated as a file or groups of files.

Data Objects may be physically transferred, or may be referenced by URLs.

Each Data Object must have a parent groupType .

The Data Object is specified in the XML schema and is of type dataObjectType.

Table ‑: Example of Data Object Type

|  |  |
| --- | --- |
| **Element** | **Content** |
| transferObjectTypeDescriptor |  |
| *… removed for brevity …* |  |
| **groupType ➋** |  |
| *… removed for brevity …* |  |
| **dataObjectType ➊** |  |
| dataObjectTypeID | L0DATAOBJECT |
| *dataObjectDescription* | *Level 0 Data Object* |
| dataObjectTypeOccurrence |  |
| minOccurrence | 1 |
| maxOccurrence | 1 |
| *dataObjectTypeFileOccurrence* **➍** |  |
| minOccurrence | 2 |
| maxOccurrence | 2 |
| *dataObjectTypeFormat* **➌** |  |
| *mimeType* | application/binary |

Table 4-5 provides an example of dataObjectType for the POLDER example.

The dataObjectTypeID and the dataObjectTypeOccurrence are the two required elements for the Data Object Type description:

* The dataObjectTypeID uniquely identifies the group, within the scope of the Producer-Archive Project, for reference purposes and is given as L0DATAOBJECT in this example. It is defined in the XML schema as xsd:string.
* The dataObjectTypeOccurrence is required and specifies the number of instances of this DataObjectType that should be present in the Group (TBC). This may be expressed as one, or more, or as a range of values.  This number may not be known at the time of specification.  In the case of a unique value, the minimal occurrence and the maximal occurrence values will be the same.

As for the groupTypeOccurrence, it is defined in the XML schema as type occurrenceType (see section 4.1.5 before).

Table 4-5 shows an occurrence of the groupType of exactly one. A more complete discussion is presented in section 4.3

The optional dataObjectTypeDescription provides the ability to have a meaningful description as to what the Data Object means. It is defined in the XML schema as xsd:string.

Table 4-5 does not show all the PAIS defined elements that may be present in a dataObjectType.

The Data Objects may be encoded (the description is the same as for the group encoding, see section 4.1.5), and may be related to other elements in the MOT (see section 4.5.2 for Association Type description).

If the Data Object is composed of different types of files (for example a header file along with a set of measurement files), then the dataObjectTypeFileOccurrence **➍** should be used to give the number of files expected. The example Table 4-5 expressed that each L0 Data Object is made up of two related and different files, a header file and a data file. Note that if the Transfer Object Descriptor included the namePreservationRule element, then the names of each file belonging to a Data Object of a Transfer Object corresponding to the Descriptor need to be preserved by the Archive.

The optional dataObjectTypeFormat **➌** may also be present (The example Table 4-5 expressed that the header and data file are in binary). This information could be used on the Archive side to perform validation, for example.

In the case where a Data Object is composed of different types of files, it is recommended to use the dataObjectType Format as follows:

* If the different files have the same format, then the dataObjectTypeFormat with this common value.
* If not, then the format information should be included in a companion, related, Transfer Object.

Note: it is also possible to model the group of files as a Group with a Set of Data Objects, each having its own format information.

## Management of MOT Identifiers

MOT identifiers are all elements whose name ends with “ID” (e.g. descriptorModelID, descriptorID, dataObjectTypeID…) The only exception is producerSourceID which is not a MOT identifier – but a reference to an external producer identifier.

**All MOT identifiers have to be unique across the overall Producer-Archive project and therefore across all XML descriptors composing a MOT**.

This is a mandatory constraint because the IDs shall be referenced **from** **outside** **the MOT** and especially from the SIPs. The ID references from outside the MOT shall uniquely identify the target resource independently from its semantic.

MOT IDs are typed xs:string. There is no recommendation about ID names. User is free to create any ID, as far as he keeps uniqueness of IDs in the MOT. User should use any methodology for verifying the uniqueness of these ID in the MOT. All platforms and free solution should be preferred.

## Objects Occurrences and Sizes

This section describes how to control the occurrences of the Transfer Objects, Transfer Object Groups, Data Objects and Files of Data Objects. It also covers the control of the Transfer Objects size.

### Occurrence Type

The occurrence type is a common type that is used by Transfer Objects, Transfer Object Groups, Data Objects and the Files of Data Objects. Its use is the same regardless of the object type. It allows the user to describe the expected number of objects allowed within the immediate parent grouping. It does NOT define the total number of objects allowed within the SIP. The transferObjectOccurrence, transferObjectGroupOccurrence, and dataObjectOccurrence are required elements and dataObjectFileOccurrence is an optional element.

If an occurrence element is supplied for an object, it will include

* a minOccurrence and a maxOccurrence or
* a minOccurrence and a maxUnknown

Table ‑: Definition of Occurrence Type

|  |  |
| --- | --- |
| **Element** | **Type** |
| occurrenceType |  |
| **minOccurrence** | **xsd:nonNegativeInteger** ➊ |
| ➌ **maxOccurrence** | **xsd:nonNegativeInteger** |
| ***maxUnknow*** | *- empty when used -* ➋ |

A minOccurrence shall have a value of zero or a positive integer defined by xsd:nonNegativeInteger (as shown at ➊ in Table 4.6 above). The value represents the minimum number of occurrences allowed for the objects. A zero value for minOccurrence indicates that this is an optional object.

**Implementation Note:** Within the users computing environment it may be desirable to restrict nonNegativeInteger to a maximum suitable for computer. Basically the same as xsd:int or xsd:integer . See section 4.7.3 for more information about how to restrict types.

The maxOccurrence shall have a value of zero or positive integer. However that value must be greater than or equal to the minOccurrence value.

A zero value may be confusing and could correspond to a situation where the objects of this type are unexpected, disabled or denied. This may help during the development of the PAIS descriptors or during some ad hoc situations.

**Implementation Note:** Within the users computing environment it may be desirable to restrict nonNegativeInteger to a maximum suitable for computer. Basically the same as xsd:int or xsd:integer. This also applies if the local community decides to restrict the use of zero as a maxOccurrence value.

If it is used, maxUnknown is an empty element as shown at ➋ in the table above.. It expresses that number of occurrences is variable and that there is no upper limit on the number of objects or that the limit is unknown.

The maxOccurrence and maxUnknown are mutually exclusive (as indicated at ➌ in the table above) so only one should be specified in the same parent element.

#### Occurrence Control Use Cases

A number of occurrence examples will be presented in the following tables.

Table ‑: A Bounded Number of Transfer Objects

|  |  |
| --- | --- |
| **Element** | **Value** |
| transferObjectTypeOccurrence |  |
| **minOccurrence** | **26** ➊ |
| **maxOccurrence** | **53** ➋ |

For the example in Table 4-7, a variable number of objects between 26 (as shown at ➊) and 53 (as shown at ➋) objects inclusive are required to appear in the parent group. If you were taking measurement every week or two for a year and you knew that you would take a measurement at least every 2 weeks, you would expect between 26 and 53 sets of measurements over the year.

Table ‑: An Unlimited Number of Groups

|  |  |
| --- | --- |
| **Element** | **Value** |
| groupTypeOccurrence |  |
| minOccurrence | 444 ➊ |
| **maxUnkown** | *- none -* ➋ |

For the example in Table 4-8, a minimum of 444 (as shown at ➊) objects inclusive are required to appear in the parent group and the maximum number of object is unknown (as shown at ➋).

Table 4‑9: A Fixed Number of Data Objects

|  |  |
| --- | --- |
| **Element** | **Value** |
| dataObjectTypeFileOccurrence |  |
| **minOccurrence** | **2** ➊ |
| **maxOccurrence** | **2** ➋ |

Table 4-9 provides an example where a fixed number of objects appear in the parent group. This is indicated by a minimum value (as shown at ➊) being equal to the maximum value (as shown at ➋). A fixed number of Data Objects might be used if you were receiving readings from 2 spacecraft. For this particular mission and the particular science needs, readings from both spacecraft are needed and readings from a single spacecraft would not be useful. If you only received information from a single spacecraft, it should be discarded. So you want to require that you received two data objects.

Table 4‑10: An Optional Transfer Object

|  |  |
| --- | --- |
| **Element** | **Value** |
| transferObjectTypeOccurrence |  |
| **minOccurrence** | **0** ➊ |
| **maxOccurrence** | **1** ➋ |

The example in Table 4-10 represents a single optional transfer object appearing in the parent group. This is defined by a minimum value of 0 (as shown at ➊) and a maximum value of 1 (as shown at ➋).

Table 4‑11: A Denied Data Object

|  |  |
| --- | --- |
| **Element** | **Value** |
| **dataObjectTypeOccurrence** |  |
| **minOccurrence** | 0 ➊ |
| **maxOccurrence** | **0** ➋ |

The example in Table 4-11 is a special case. Although a denied object is allowed by the standard, we expect that it will rarely be used. This case will likely be used primarily to document that an object exists and to document that it is not intended for transfer to the Archive. A denied object is identified by having a zero value for both the minimum value (as shown at ➊) and the maximum value (as shown at ➋). A possible use for this construct would be to model data on the Producer’s site where a file exists that records user access to the set of data and where the Producer wants to ensure that the file is not transferred in order to protect the privacy of the users.

Table ‑: Invalid Occurrence – Attempted Negative Bound

|  |  |
| --- | --- |
| **Element** | **Value** |
| dataObjectTypeFileOccurrence |  |
| **minOccurrence** | **-1** ➊ |
| **maxOccurrence** | **2** |

The example in Table 4-12 shows a case that should never exist. Providing a negative value (as shown at ➊) for either the minimum value or the maximum value is an error.

Table 4‑13: Invalid Occurrence – Attempted Maximum Value Less Than Minimum Value

|  |  |
| --- | --- |
| **Element** | **Value** |
| dataObjectTypeFileOccurrence |  |
| **minOccurrence** | **5** ➊ |
| **maxOccurrence** | **2** ➋ |

The example in Table 4-13 shows another case that should never exist. The maximum value should never be less than the minimum value. In this case an attempt is made to define the minimum value as 5 (as shown at ➊) while defining the maximum value as 2 (as shown at ➋) which is less than the minimum value and therefore is an error.

### Transfer Object Sizes

The Transfer Object Type Size is a type that provides an optional element that provides an indication of the total size range for Transfer Objects. The size range may not be known at the time of descriptor creation and therefore the corresponding element may be omitted.

Table 4‑14: Definition of Transfer Object Type Size

|  |  |
| --- | --- |
| **Element** | **Type** |
| transferObjectTypeSize |  |
| **minSize** | **xsd:float** ➊ |
| **maxSize** | **xsd:float** ➋ |
| **unitsType** | **xsd:String** ➌  xsd:enumeration value="**KB**"  xsd:enumeration value="**MB**"  xsd:enumeration value="**GB**"  xsd:enumeration value="**TB**"  xsd:enumeration value="**PB**" |

A unitsType shall have a value that, as shown at ➌ in Table 4.14, is one of the strings “KB”, “MB”, “GB”, “TB”, or “PB” which indicates that the size values provided are in kilobytes, megabytes, gigabytes, terabytes, or petabytes respectively.

**Implementation Note:** The PAIS Standard does not indicate whether the prefixes represent multiples of 1000 bytes or 1024 bytes. The Producer and Archive should come to an agreement on the meaning of this for the specific Producer-Archive Project.

A minSize shall have a value defined by xsd:float (as shown at ➊ in Table 4.14 above). A maxSize shall have a value defined by xsd:float (as shown at ➋). However the minSize value must be less than or equal to the maxSize value. If the minSize value equals the maxSize value, then the size of the Transfer Object must be the given size. In both cases the value represents the size in units defined by unitsType.

**Implementation Note:** The xsd:float Type allows both positive and negative numbers. Since the PAIS Standard does not define the meaning of negative sizes, it may be desirable to restrict xsd:float to a non-negative values. See section 4.7.3 for more information about how to restrict types.

## Objects Encodings

This section addresses how to describe the encoding of Transfer Object Groups and Data Objects.

When applied to a Transfer Object Group, using the element groupTypeEncoded of type encodedType, it is applied to all the data associated with all contained Data Objects and all contained Transfer Object Groups. It results in a single file in the Transfer Object.

When applied to a Data Object, using the element dataObjectTypeEncoded of type encodedType, it is applied to all the files associated with the Data Object and the result is a single file in the Transfer Object.

### encoded Type

The encodedType is specified in the XML schema. It is a complex type consisting of two elements, encodingName of type xsd:string and encodingDescription of type xsd:string, as shown in Table 4-15.

Table 4‑15: Definition of Encoded Type

|  |  |
| --- | --- |
| **Element** | **Type** |
| encodedType |  |
| **encodingName** | **xsd:string** |
| **encodingDescription** | **xsd:string** |

#### Encoded Type with Groups

As an example, the top level groupType of Table 4-4 has been altered to include two groupTypeEncoded elements. When there are multiple encodings such as this, they are to be performed in the order given. Therefore the directory structure associated with the various groupTypes will be maintained by the tar function, and the data files associated with the various dataObjectTypes will be incorporated accordingly. Once the tar file has been generated, it will be gzip encoded into another file for inclusion in the Transfer Object in a SIP. While it is not a requirement, for this example the names and descriptions follow the mime type expressions.

Table 4‑16: Example of Group Type with Encoding

|  |  |
| --- | --- |
| **Element** | **Sample Value** |
| transferObjectTypeDescriptor |  |
| identification |  |
| descriptorID | L0DATA |
| *… removed for brevity …* |  |
| **groupType ➊** |  |
| groupTypeID | L0GROUP |
| *groupTypeDescription* | *Level 0 Group Type* |
| groupTypeStructureName ➋ | directory |
| groupTypeOccurrence |  |
| minOccurrence | 1 |
| maxOccurrence | 1 |
| groupTypeEncoded |  |
| encodingName | tar |
| encodingDescription | application/x-tar |
| groupTypeEncoded |  |
| encodingName | gzip |
| encodingDescription | application/x-gzip |
| ***groupType*** ➌ |  |
| *… removed for brevity …* |  |
| ***dataObjectType*** ➍ |  |
| *… removed for brevity …* |  |
| ***dataObjectType*** ➎ |  |
| *… removed for brevity …* |  |

Note that upon receipt by an Archive, the associated Transfer Object will include an identifier pointing to LODATA and therefore the Archive can associate this Descriptor with the encoded data file. However upon unpacking the encoded data file, which is to be performed in reverse order to that given in the groupType, the resulting directory structure will not include any identifiers of the nested groupTypes or dataObjectTypes that have been modeled. Therefore it may not be possible to make much use of the modeling that was done for groupTypes under the groupType having the encoding elements. Section 5 discusses the instantiation of Transfer Objects within a SIP in more detail.

#### Encoded Type with Data Objects

As an example, the transferObjectTypeDescriptor of Table 4-4 has been altered to include a dataObjectTypeEncoded element under the dataObjectType element of the first nested groupType.

Table 4‑17: Example of Data Object Type with Encoding

|  |  |
| --- | --- |
| **Element** | **Sample Value** |
| transferObjectTypeDescriptor |  |
| identification |  |
| descriptorID | L0DATA |
| *… removed for brevity …* |  |
| **groupType ➊** |  |
| groupTypeID | L0GROUP |
| *groupTypeDescription* | *Level 0 Group Type* |
| groupTypeStructureName ➋ | directory |
| groupTypeOccurrence |  |
| minOccurrence | 1 |
| maxOccurrence | 1 |
| ***groupType*** ➌ |  |
| groupTypeStructureName | directory |
| *… removed for brevity …* |  |
| ***dataObjectType*** ➍ |  |
| dataObjectTypeID | L0DATAOBJECT | |
| *dataObjectDescription* | *Level 0 Data Object* | |
| dataObjectTypeOccurrence |  | |
| minOccurrence | 1 | |
| maxOccurrence | 1 | |
| dataObjectTypeEncoded |  |
| encodingName | compress |
| encodingDescription | LZW compression |
| dataObjectTypeFormat |  |
| mimeType | PDF |
| registrationInformation | Application/pdf |
| ***dataObjectType*** ➎ |  |
| *… removed for brevity …* |  |

It states that the data of this dataObjectType, shown to be a single PDF file, is to be encoded using the LZW compression algorithm before being instantiated into the Transfer Object within a SIP.The instantiated Transfer Object will reflect the directory structure of this transferObjectTypeDescriptor, having a top level directory with a single dataObjectType (details not provided for brevity) and a sub-directory with a compressed PDF data file. Section 5 discusses the instantiation of Transfer Objects within a SIP in more detail.

## Objects Relations

The PAIS standard supports directed, binary, relationships expressed between entities in the MOT using the complex element relation, and it sub-elements, as defined in the XML schema. This element is mandatory in both Transfer Object Descriptors and Collection Descriptors. It contains the mandatory element parentCollection and the optional elements association and any as shown in Table 4-18. The any element allows users of the PAIS to define new relation elements. Details on the use of any are discussed in section 4.7.

Table 4‑18: Definition of Relation

|  |  |
| --- | --- |
| **Element** | **Type** |
| relation |  |
| **parentCollection** | **xsd:string** |
| ***association*** | **pais:associationType** |
| ***any*** | **pais:extensionType** |

### parent collection relation

A Descriptor’s mandatory parentCollection element supports an aggregate view of the Collection to which this Descriptor belongs. It is of type xsd:string as defined in the XML schema.

The value of parentCollection must be the identifier of the single Collection to which the Descriptor (be it Transfer Object Descriptor or Collection Descriptor) belongs. In other words, the value must be the descriptorID value of the Parent Collection. The top level Collection, of which there is only one, must have a parentCollection whose value is none.

Figure 4-3 provides a schematic view of a MOT, consisting of stylized XML Descriptors, with the parentCollection relations identified.

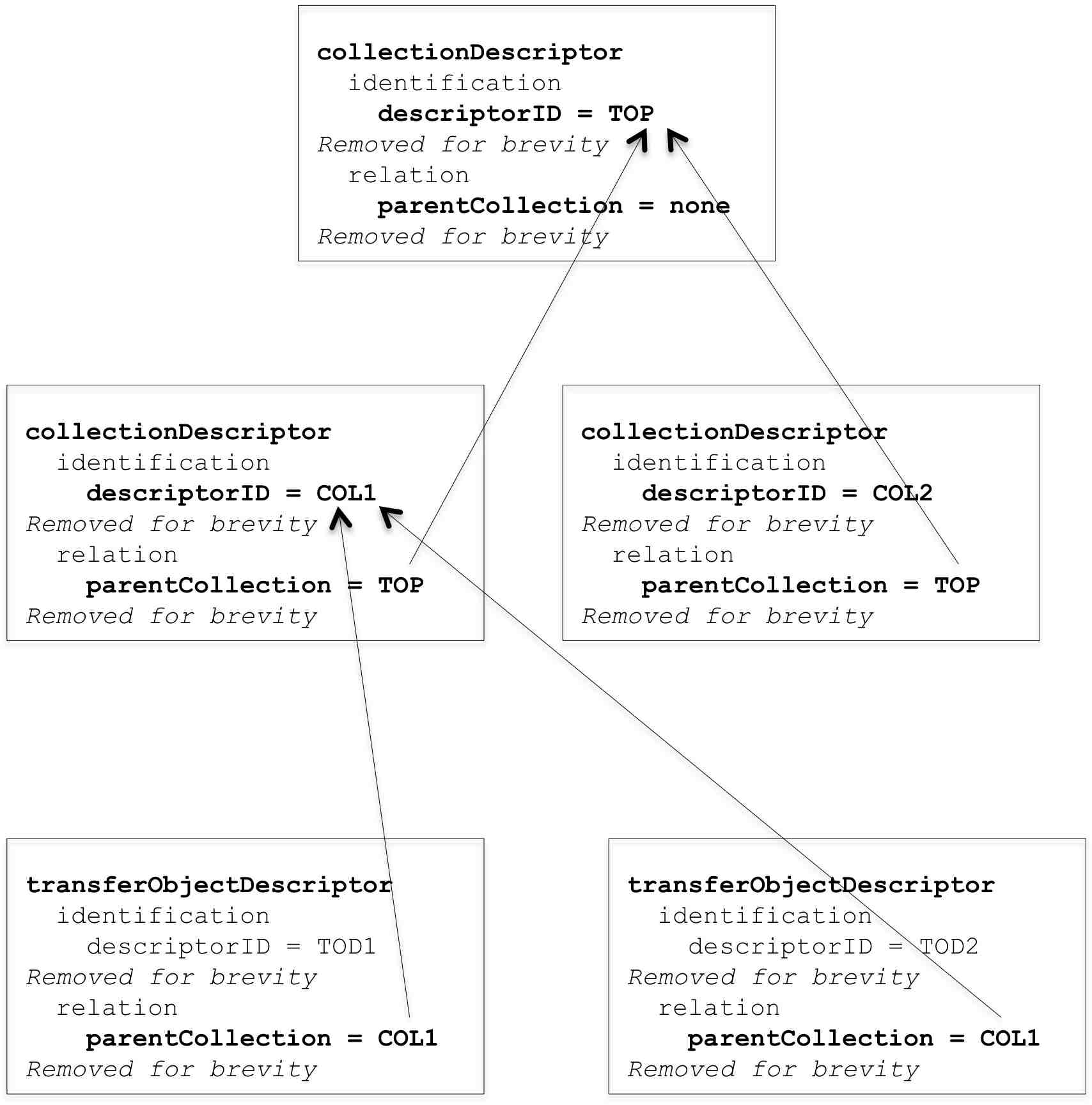


Figure 4‑3: Snippet of MOT with Parent Collection Relations Identified

### association TYPE relation

The pais:associationType is a complex common type as defined in the XML schema and shown in Table 4-19. It is the type assigned to three elements: association, groupTypeAssociation, and dataObjectTypeAssociation. While each is optional, each may be used as often as needed to establish all the relationships that are to be modeled.

Table 4‑19: Definition of associationType Relation

|  |  |
| --- | --- |
| **Element** | **Type** |
| associationType |  |
| **targetID** | **xsd:string** |
| **relationDescription** |  |
| **relationType** | **xsd:string** |
| ***relationTextualDescription*** | **xsd:string** |

The element association establishes a relationship from the Descriptor (i.e., Transfer Object Descriptor (Transfer Object Type) or Collection Descriptor) in which it is found **toward** a target Descriptor or target entity within a Descriptor. The target entity within a Descriptor is either a Transfer Object Group Type or a Data Object Type.

The element groupTypeAssociation establishes a relationship from the Transfer Object Descriptor’s Group Type in which it is found toward a target Descriptor or target entity within a Descriptor. The target entity within a Descriptor is either a Transfer Object Group Type or a Data Object Type.

The element dataObjectTypeAssociation establishes a relationship from the Transfer Object Descriptor’s Data Object Type in which it is found toward a target Descriptor or target entity within a Descriptor. The target entity within a Descriptor is either a Transfer Object Group Type or a Data Object Type.

As the Transfer Object Descriptor is also a type definition, the relationships also apply to the instantiated Transfer Objects, Transfer Object Groups, and Data Objects as applicable.

The targetID element has the type xsd:string as defined in the XML schema. The value of targetID must be a valid identifier of one of the allowed targets, either descriptorID, groupTypeID, or dataObjectTypeID.

The complex relationDescription element is defined in the XML schema. It is composed of the mandatory relationType element and the optional relationTextualDescription element.

Table 4-20 shows a partial expansion of the Transfer Object Descriptor TOD1 from the MOT of Figure 4-1 and it highlights the use of the association type of relationships for the three elements association, groupTypeAssociation and dataObjectTypeAssociation.

Table 4‑20: Example of Transfer Object Type with Associations

|  |  |
| --- | --- |
| **Element** | **Sample Value** |
| transferObjectTypeDescriptor |  |
| identification |  |
| **descriptorID** | **TOD1** |
| *… removed for brevity …* |  |
| relation |  |
| parentCollection | COL1 |
| **association** |  |
| **targetID** | **COL2** |
| **relationDescription** |  |
| **relationType** | **Data described by** |
| **relationTextualDescription** | **Data described by targeted collection of documentation** |
| *… removed for brevity …* |  |
| groupType |  |
| groupTypeID | L0GROUP |
| **groupTypeAssociation** |  |
| **targetID** | **COL2TO-1** |
| **relationDescription** |  |
| **relationType** | **Data related to** |
| **relationTextualDescription** | **Data from instrument has targeted instrument description** |
| **groupTypeAssociation** |  |
| **targetID** | **COL2TO-2** |
| **relationDescription** |  |
| **relationType** | **Data related to** |
| **relationTextualDescription** | **Data from mission as targeted mission description** |
| *… removed for brevity …* |  |
| *dataObjectType* |  |
| dataObjectTypeID | L0DATAOBJECT | |
| **dataObjectTypeAssociation** |  |
| **targetID** | **COL2TO-3** |
| **relationDescription** |  |
| **relationType** | **Data formatted as** |
| **relationTextualDescription** | **Data whose format is targeted format description** |
| *… removed for brevity …* |  |

Under the relation element, the complex association element shows a targetID value of COL2. This states that the relationship is being established from this Transfer Object Descriptor to the Collection Descriptor whose collectionID value is COL2. This Collection Descriptor is shown in Figure 4-1. The relationship direction is always from the entity holding the described relationship toward the targeted entity. The relationType is stated to be Data described by and the optional relationTextualDescription is given as Data described by targeted collection of documentation. Although not shown in Figure 4-1, Table 4-20 assumes that the Collection COL2 has three Transfer Object Descriptors as children and that they are describing different types of documentation related to the data described by Transfer Object Descriptor TOD1.

Under groupType, there are two different groupTypeAssociation elements. The first states that this groupType is related via the targetID to an entity whose ID is COL2TO-1. Although not shown, this is the descriptorID of a Transfer Object Descriptor under the COL2 collection. This Descriptor describes text documentation about the instrument used to generate the data associated with the TOD1 Descriptor. The relationType is given as Data related to and the optional relationTextualDescription is given as Data from instrument has targeted instrument description. More precisely it is the modeling of the data that is being related to the modeling of the instrument documentation by the use of the Descriptor, but the relationships carries through to their respective Transfer Objects.

The second groupTypeAssociation element is much like the first, however its targetID refers to a different Transfer Object Descriptor whose descriptorID is COL2TO-2. This also is not shown in Figure 4-1 due to space limitations. This Descriptor describes text documentation about the mission under which the instrument was used to generate the data associated with Transfer Object Desccriptor TOD1.

Under dataObjectType, there is one complex dataObjectTypeAssociation element. Using the targetID element, it establshes a realtionship from the Descriptor’s dataObjectType specification to an entity with the identifier COL2TO-3. This is the descriptorID value of a Transfer Object Descriptor (not shown) that is also a child under the COL2 Collection. This Descriptor describes a text document specifying the format of the data for the Data Object Type shown in Table 4-20. The relationType is given as Data formatted as and the optional relationTextualDescription is given as Data whose format is targeted format description.

Figure 4-4 is a partial expansion of the MOT snippet of Figure 4-1. It shows some of the parentCollection relationships (solid arrows), an association relationship (dotted arrow), and a groupTypeAssociation relationship (dotted arrow).

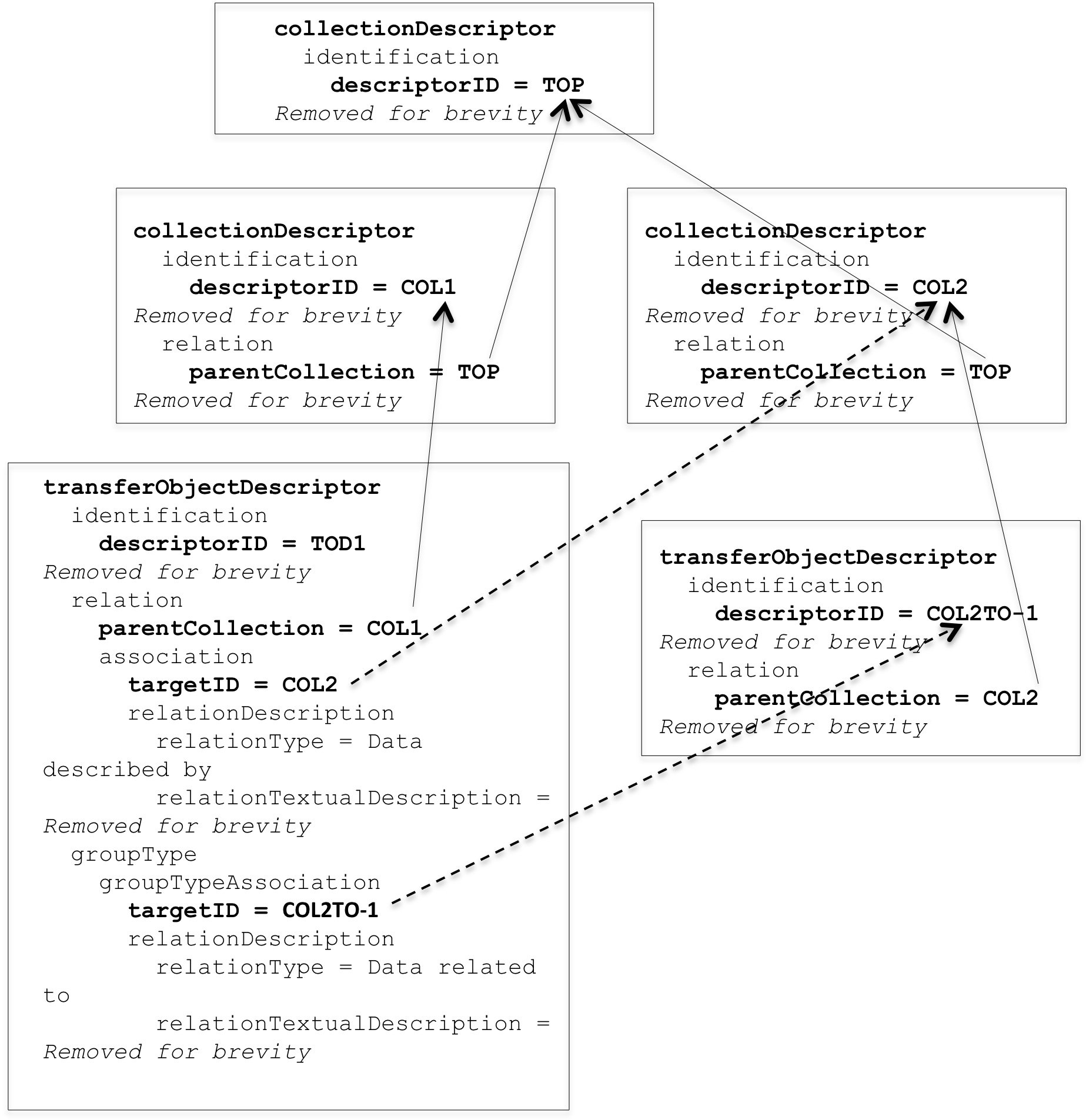


Figure 4‑4: Snippet of MOT Showing Parent and Association Relationships

In summary, the association type relation can be used between any two pairs of the following four Descriptor entities: Transfer Object Descriptor, Collection Descriptor, Group Type, and Data Object Type. Pairings involving Group Type and Data Object Type may even be within the same Transfer Object Descriptor. When the relation involves a Transfer Object Descriptor or one of its sub-entities, the relation extends to the Descriptor’s instantiated Transfer Objects.

## SIP Constraints

There a two types of constraints that apply to the actual SIPs.

1. Defines SIP types and defines what data, i.e. Transfer Object Types, appears in each type of SIP.
2. Defines the order in which SIPs should arrive at the Archive.

An example of when these constraints would be used is a case where the Producer and Archive have agreed that the Producer will send descriptions of the content in SIPs before the Producer sends the primary content in following SIPs.

Table 4‑21: SIP Constraints

|  |  |
| --- | --- |
| **Element** | **Type** |
| sipConstraints |  |
| **producerArchiveProjectID** | **xsd:string** |
| **sipContentType** |  |
| **sipContentTypeID** | **xsd:string** |
| **authorizedDescriptor** |  |
| **descriptorID** | **xsd:string** |
| **occurrence** | **pais:occurrenceType** |
| **minOccurrence** | **xsd:integer** |
| **maxOccurrence** | **xsd:integer** |
| **maxUnknown** | **xsd:string** |
| **sipSequencingConstraintGroup** |  |
| **groupName** | **xsd:string** |
| **constraintItem** |  |
| **sipContentTypeID** | **xsd:string** |
| **constraintSerialNumber** | **xsd:integer** |
|  |  |
|  |  |

An example of a minimal SIP Constraints file, for which there is only one Content Type and no sequencing constrains, is as follows:

<sipConstraints xmlns="urn:ccsds:schema:pais:1">

<producerArchiveProjectID>MyProject</producerArchiveProjectID>

<sipContentType>

<sipContentTypeID>Content Type A➊</sipContentTypeID>

<authorizedDescriptor>

<descriptorID>Blue Descriptor ID➋</descriptorID>

<occurrence>➌

<minOccurrence>2</minOccurrence>

<maxOccurrence>2</maxOccurrence>

</occurrence>

</authorizedDescriptor>

</sipContentType>

</sipConstraints>

Table 4‑22: Example of SIP Constraints Content

|  |  |
| --- | --- |
| **Node** | **Content** |
| **sipConstraints** |  |
| @xmlns | urn:ccsds:schema:pais:1 |
| producerArchiveProjectID | MyProject |
| **sipContentType** |  |
| sipContentTypeID | Content Type A➊ |
| **authorizedDescriptor** |  |
| descriptorID | Blue Descriptor ID➋ |
| **occurrence➌** |  |
| minOccurrence | 2 |
| maxOccurrence | 2 |

The Content Type A accepts only one Transfer Object Type identified as “Blue Descriptor ID” ➋. The example also defines that two and only two objects of this type are expected per SIP of this Content Type ➌.

As a second example of a SIP Constraints file, we present the case were Representation Information is sent to the Archive prior to sending the primary data. The example file is as follows:

<sipConstraints xmlns="urn:ccsds:schema:pais:1">

<producerArchiveProjectID>MyProject2</producerArchiveProjectID>

<sipContentType>

<sipContentTypeID>RepInfo Content Type➊</sipContentTypeID>

<authorizedDescriptor>

<descriptorID>IDRepInfo➋</descriptorID>

<occurrence>➌

<minOccurrence>1</minOccurrence>

<maxOccurrence>1</maxOccurrence>

</occurrence>

</authorizedDescriptor>

</sipContentType>

<sipContentType>

<sipContentTypeID>Raw Data Content Type➊</sipContentTypeID>

<authorizedDescriptor>

<descriptorID>IDRawData➋</descriptorID>

<occurrence>➌

<minOccurrence>12</minOccurrence>

<maxOccurrence>366</maxOccurrence>

</occurrence>

</authorizedDescriptor>

</sipContentType>

<sipSequencingConstraintGroup>

<groupName>My Single Restraint - RepInfo Before Data</groupName>

<constraintItem>

<sipContentTypeID>IDRepInfo</sipContentTypeID>

<constraintSerialNumber>1</constraintSerialNumber>

</constraintItem>

<constraintItem>

<sipContentTypeID>IDRawData</sipContentTypeID>

<constraintSerialNumber>2</constraintSerialNumber>

</constraintItem>

</sipSequencingConstraintGroup

</sipConstraints>

Table 4‑23: Example of SIP Constraints Content

|  |  |
| --- | --- |
| **Node** | **Content** |
| **sipConstraints** |  |
| @xmlns | urn:ccsds:schema:pais:1 |
| producerArchiveProjectID | MyProject2 |
| **sipContentType** |  |
| sipContentTypeID | RepInfo Content Type➊ |
| **authorizedDescriptor** |  |
| descriptorID | IDRepInfo➋ |
| **occurrence➌** |  |
| minOccurrence | 1 |
| maxOccurrence | 1 |
| **sipContentType** |  |
| sipContentTypeID | Raw Data Content Type➊ |
| **authorizedDescriptor** |  |
| descriptorID | IDRawData➋ |
| **occurrence➌** |  |
| minOccurrence | 12 |
| maxOccurrence | 366 |
| **sipSequencingConstraintGroup** |  |
| groupName | My Single Constraint – RepInfo Before Raw Data |
| **constraintItem** |  |
| sipContentID | IDRepInfo |
| sequenceNumber | 1 |
| **constraintItem** |  |
| sipContentID | IDRawData |
| sequenceNumber | 2 |
|  |  |

The RepInfo Content Type ➊ accepts only one Transfer Object Type identified as “IDRepInfo” ➋. The example defines that exactly one object of this type is expected in this type of SIP ➌.

The Raw Data Content Type ➊ also accepts only one Transfer Object Type identified as “IDRawData” ➋. Since the Raw Data for the example is a year’s worth of data and it may be collected daily, but it is required that at least one measurement be made every month. The example defines between twelve and three hundred sixty-six objects of this type are expected per SIP of this Content Type ➌.

The single sequencing constraints group specifies that SIPs identified by the “IDRepInfo” ID be delivered before any SIPs identified by the “IDRawData” ID.

## Customization – Extensions and Specializations

XML Schema is a formal language to describe XML files.

PAIS XML Schemas constitute the generic and formal definition of any Digital Objects to be transferred.

PAIS XML Schemas define the minimum required information that must be shared by the Producer and the Archive, to :

* Produce and transfer Digital Objects
* Receive and interpret in an unambiguous manner the transferred Digital Objects.

PAIS XML Schemas may be used as is by projects. Projects may also specialize the PAIS XML Schemas in order to provide a more precise specification of the model. There are many ways of specializing the PAIS XML Schemas (see following sections), each of them following a single simple rule: the model defined by the specialized schemas must be fully compatible with the generic XML Schema defined by the PAIS standard; in other words XML files produced using the specialized model (i.e. valid against the specialized schemas) must be interpretable using the generic model (i.e. valid against the generic schemas).When the PAIS XML Schema is specialized, it shall also be renamed (e.g. <project>-pais-descriptor-collection.xsd is a specialization of ccsds-pais-descriptor-collection.xsd).

The targetNamespace of the specialized schema remains identical to the targetNamespace of the generic PAIS XML Schema (for compatibility).

The descriptorModelID is no longer CCSD0015 since the schema used is a specialized schema, and the specialized schema fixes the value fot that element (e.g. CNES0023).

We recommend to add an annotation in the specialized schema in the form of:

<xsd:annotation>

<xsd:documentation>

CollectionDescriptor for <project>, based on ccsds-pais-descriptor-collection schema.

(descriptorModelID, descriptorModelVersion) = (CNES0023 , 2.1) based on (CCSD0015 , 1.0)

</xsd:documentation>

</xsd:annotation>

### PAIS XML Schemas

PAIS XML Schemas are mainly made up of three XML Schemas: ccsds-pais-descriptor-collection.xsd which specifies a collection and ccsds-pais-descriptor-transfer-object.xsd which specifies a Transfer Object. Both schemas use some common definitions, gathered in ccsds-pais-common-types.xsd.

The PAIS standard [1] provides the complete view and description of these schemas.

PAIS XML Schemas can be found on the official CCSDS SANA XML Schemas repository: <http://sanaregistry.org/r/daixml>

### Extensions

PAIS XML Schemas contain special elements, named ‘any’; these elements are extension points, which means that any additional element or attribute may be added at these points, the XML file remaining valid against the generic Schema.

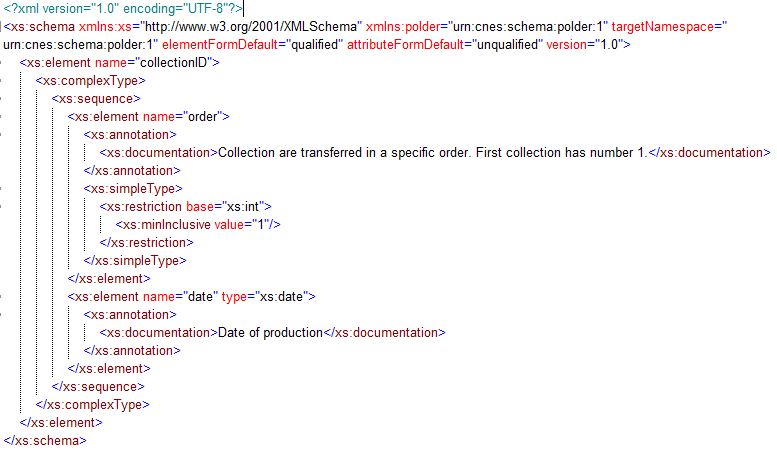
Extension points are the opportunity for a project to specify additional information for the transfer of Digital Objects.

The additional part (or fragment) has its own XML Namespace, since it has not been defined by the CCSDS. The XML Namespace and associated prefix are free, nevertheless we recommend to use similar rule to build the namespace of a project specific part.

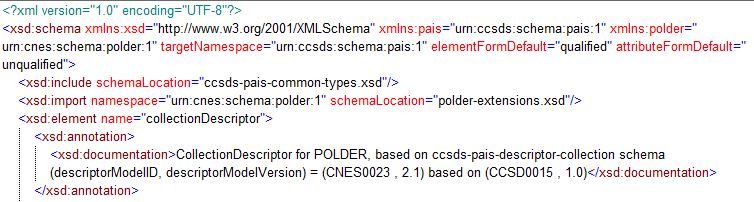
The following text presents a concrete example of extended schema.

For example, the project POLDER may need to order and date the transferred collections. For that purpose, the following XML Namespace is defined : **urn:cnes:schema:polder:1**

The additional XML Schema part, **polder-extensions.xsd** is defined: this schema contains the definition of one (or more) element(s), designed to complete the generic schema. In this example, the element “collectionID” is defined to complete the identification part: it is a sequence of a number and a date.

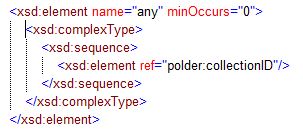


The PAIS XML Schema is renamed to polder-pais-descriptor-collection.xsd. The schema contains an import link to polder-extensions.xsd.



Note that the targetNamespace remains identical to the targetNamespace of the PAIS schema.

The any element of the identification structure is redefined. It contains now a reference to the element “collectionID”.



The following table presents an example of an extended collection.

Table 4‑24: Example of Extended Collection

|  |  |
| --- | --- |
| **Element** | **Content** |
| collectionDescriptor |  |
| identification |  |
| descriptorModelID➍ | CNES0023 |
| descriptorModelVersion | 2.1 |
| descriptorID | POLDER |
| any ➊ |  |
| polder:collectionID ➋ |  |
| @xmlns ➌ | urn:cnes:schema:polder:1 |
| polder:order | 1 |
| polder:date | 2015-06-18 |

NOTE: ➊ Allowed only once and accepts only **one** child ➋ with a namespace ➌ different than the one of the PAIS.

NOTE: ➍descriptorModelID is no longer CCSD0015 since the schema used is a specialized schema.

### Restrictions

In XML Schema the restriction is the standard mechanism for sub-typing, or controlling built-in type constraints e.g. pattern, ranges, etc. Restrictions are very common for XML Schemas that are not based only on direct reference to the built-in types.

Another way of specializing the generic PAIS XML Schema is therefore to restrict definitions:

* Restrict the number of allowed values in an enumeration
* Restrict the range of allowed values for an integer or for a real
* Limit the size of a character string
* Specialize a character string using pattern (for an unlimited number of allowed values)
* Specialize a character string using an enumeration type (for a limited number of allowed values)
* Specify the maximum occurrence number of some elements (e.g. pais:association may occur 0, 1 or more times, up to ∞ (often set to ‘unbounded’ in the generic schema)
* Delete optional elements
* Specify that an element is required while it is optional in the generic schema
* Change a choice between multiple elements into a sequence of a single element

Or any other restriction that makes the new schema compatible with the generic schema.

The following table suggests restrictions for some elements of the PAIS XML Schema ccsds-pais-descriptor-collection:

Table 4‑25: How to specialize a Collection Descriptor

|  |  |
| --- | --- |
| **Element** | **Restrictions** |
| **Collection Descriptor** | |
| descriptorModelID | fixed➊ |
| descriptorModelVersion | fixed➊ |
| descriptorID | pattern recommended, length > 1 |
| collectionTitle | length > 5 |
| collectionDescription | length > 5 |
| minSize | Should be positive or null |
| maxSize | May be restricted to the actual/total Archive capabilities or any intermediaet system that could not handle the size e.g. ZIP. |
| unitsType | restrict enum to minimal set e.g. MB or GB |
| parentCollection |  |
| targetID | pattern recommended, length > 1➋ |
| relationType | restrict to actually used e.g. representationInformation, dependency, use, etc. ➌ |
| relationTextualDescription |  |

NOTE: ➊ fixed value may be defined using the “fixed” attribute or using a single-value enumeration type.

The following table illustrates ➋:

Table 4‑26: Example of Restricted Type – Simple Type – String Patterns

|  |  |
| --- | --- |
| **Element** | **Content** |
| xs:element |  |
| @name | targetID |
| xs:simpleType |  |
| xs:restriction |  |
| @base | xs:string |
| xs:pattern |  |
| @value | POLDER\_.\* |
| xs:minLength |  |
| @value | 10 |
| xs:whiteSpace |  |
| @value | preserve |

The following table illustrates ➌:

Table 4‑27: Example of Restricted Type – Simple Type – Enumeration Type

|  |  |
| --- | --- |
| **Element** | **Content** |
| xs:element |  |
| @name | relationType |
| xs:simpleType |  |
| xs:restriction |  |
| @base | xs:string |
| xs:enumeration |  |
| @value | representationInformation |
| @value | dependency |

The following table suggests restrictions for some elements of the PAIS XML Schema ccsds-pais-descriptor-transfer-object:

Table 4‑28: How to specialize a Transfer Object Descriptor

|  |  |  |
| --- | --- | --- |
| **Element** | **Restrictions** | |
| **Transfer Object Type Descriptor** | | |
| descriptorModelID | | fixed➊ |
| descriptorModelVersion | | fixed➊ |
| descriptorID | | pattern recommended, length > 1 |
| producerSourceID | | pattern recommended, length > 1 |
| transferObjectTypeTitle | | length > 5 |
| transferObjectTypeDescription | | length > 5 |
| minOccurrence | | restrict xs:nonNegativeInteger to a type applicable for the project and for the target hardware e.g. xs:short, xs:int, etc. May constrain to an actual minimum of minimum. |
| maxOccurrence | | restrict xs:nonNegativeInteger to a type applicable for the project and for the target hardware e.g. xs:short, xs:int, etc. May constrain to an actual maximum of maximum |
| maxUnknown | | deny if not applicable |
| minSize | | Should be positive or null |
| maxSize | | May be restricted to the actual/total Archive capabilities or any intermediaet system that could not handle the size e.g. ZIP. |
| unitsType | | restrict to an agreed enumeration (MB, GB…) |
| namePreservationRule | | deny if not used or restrict to an enumeration |
| parentCollection | | pattern recommended, length > 1 |
| targetID | | pattern recommended, length > 1 |
| relationType | | restrict to actually used e.g. representationInformation, dependency, use, etc.➋ |
| relationTextualDescription | | free texte (no restriction) |
| groupTypeID | | pattern recommended, length > 1 |
| groupTypeDescription | | free text (no restriction) |
| groupTypeStructureName | | restrict to an agreed enumeration (directory, set, sequence…) |
| encodingName | | restrict to an agreed enumeration (zip, tar…) |
| encodingDescription | | free text (no restriction) |
| dataObjectTypeID | | pattern recommended, length > 1 |
| mimeType | | restrict to an agreed enumeration (text/xml, image/jpeg…). |
| registrationAuthority | | deny if not accepted or restrict to an agreed enumeration |
| registeredID | | deny if not accepted or restrict to an agreed enumeration or at a minimum, a pattern (not recommended) |

Each project should adopt a common policy for restricting all ID values e.g. min one char, printable characters, pattern.

# Building and Manipulating SIPs

The generation of the MOT, consisting of all the XML Descriptors for a given Archive Project, together with the XML constraints document, establishes the basis for the generation of the SIPs. These XML documents are used by the Producer, in conjunction with the actual data files in the Producer’s environment, as input to the SIP building process.

The SIP building process needs to take the XML modeling information and create a conforming mapping from the Producer’s data to actual SIPs. This mapping is instantiated, conceptually, by a combination of SIP building software and the information supplied by the Producer’s agent who is using the software. This mapping is not explicitly addressed by the PAIS. However the PAIS provides an abstract view of resulting SIPs that incorporate the required Transfer Objects along with other required information, such as a set of global SIP attributes and whether any previously sent Transfer Objects are to be deleted.

The following sub-sections address the abstract SIP (section 5.1), the PAIS specified SIP instantiated using the XFDU packaging mechanism (section 5.2), and the abstract SIP instantiated by a non-PAIS specified packaging mechanism (section 5.3).

## Understanding the PAIS Abstract SIP

The PAIS abstract SIP is a specification that has been registered with the CCSDS and has been given the identifier ‘CCSD0017’. The fact that the SIP specification is registered with CCSDS simply means that a CCSDS document exists that defines this specification. Definition of an abstract SIP is significant because the PAIS allows for the instantiation of SIPs using different physical SIP packaging mechanisms. While the PAIS recommendation specifies the XFDU packaging mechanism and details how this should be adapted, a user-defined specification using some other packaging mechanism is also possible. The user-defined specification should reference the PAIS abstract SIP specification CCSD0017 as its foundation.

As the PAIS standard notes, “SIPs carry the data, or pointers to the data, being transferred to an Archive. The abstract SIP, or SIP Model, is an abstraction that puts constraints on all possible SIPs. It conceptually conveys one or more complete Transfer Objects. It also conceptually conveys a number of attributes about the SIP. The framework for this SIP model is based on the concept of containers. The SIP Model is a container that holds any number of internal containers which themselves may have containers, and so on, thus supporting multiple hierarchies of containers. A container may also hold attributes about itself.”

The PAIS provides several diagrams with text detailing the SIP container and the allowed containers within a SIP. These containers are identified as follows:

* A SIP container holds:
  + one SIP Global Information container;
  + any number of Transfer Object containers; and
  + any number of Transfer Object To Delete containers.
* A Transfer Object container holds:
  + one Transfer Object Identification and Status container; and
  + one or more Transfer Object Group containers.
* A Transfer Object Group container holds:
  + Transfer Object Group Identification container;
  + any number of other Transfer Object Group containers; and
  + any number of Data Object containers.
* A Data Object container holds:
  + Data Object Identification container; and
  + One or more Byte Stream containers.

A summary diagram from PAIS, presented in Figure 5-1 below, shows the containers and their relationships. The containers and their attributes are discussed below.

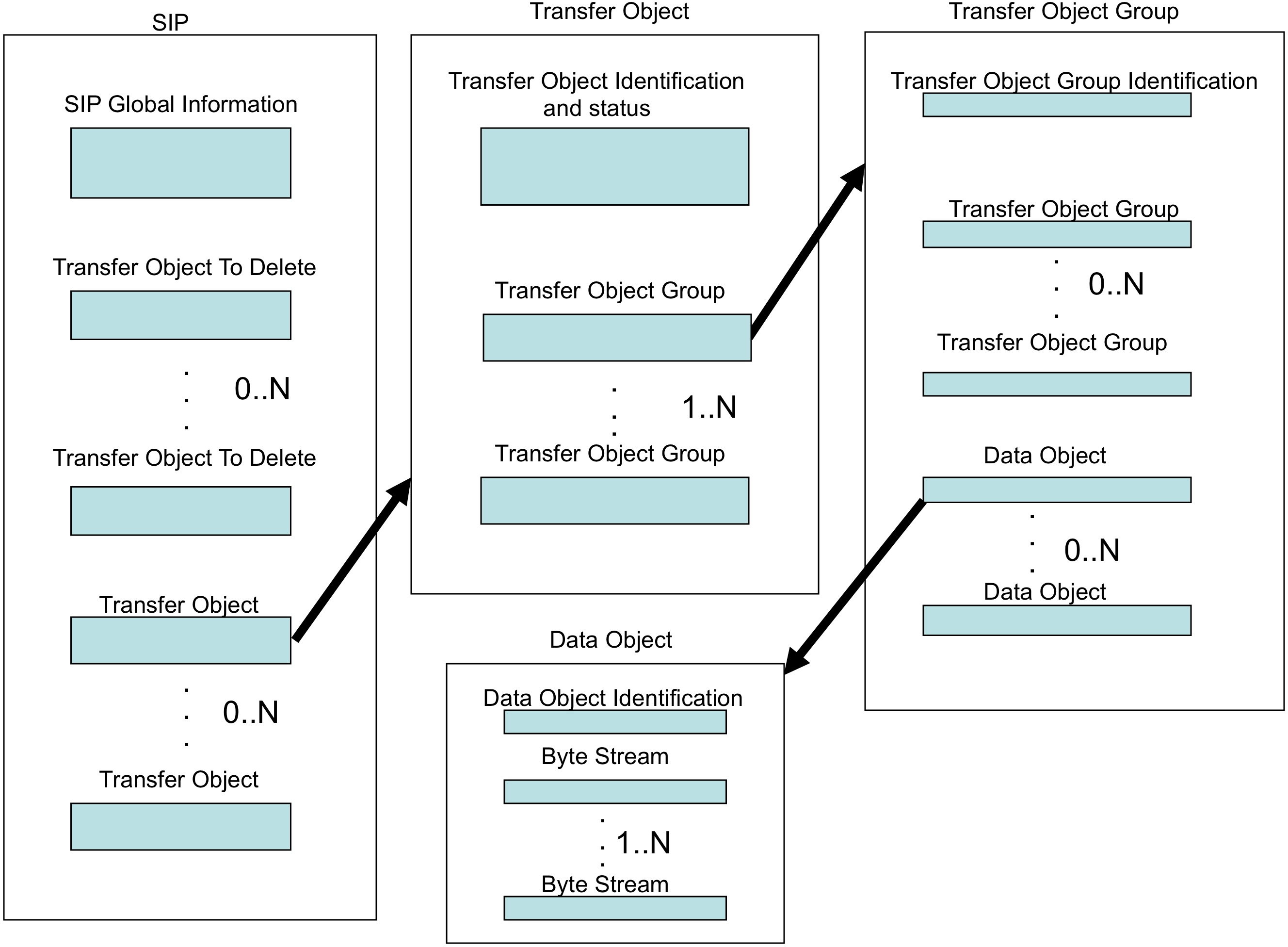


Figure 5‑1: Abstract View of SIP, Transfer Object, Transfer Object Group, and Data Object

### SIP container

This container holds any number of Transfer Object containers, any number of Transfer Object to Delete containers and one SIP Global Information container.

#### SIP Global Information container

This container holds a number of attributes as follows:

**SIP ID**

The SIP ID is a mandatory attribute that must have a unique value across all SIPs within a given Producer-Archive Project. If there is a single entity within the Producer’s environment that is creating and sending SIPs, then this identifier could be simply a sequence number. However if there are multiple entities within the Producer’s environment creating and sending SIPs, then this identifier could be made unique by pre-pending a unique identifier of each sending entity to a sequence number generated by each sending entity. Each sending entity is referred to as a ‘Producer Source.’ Both the Producer and Archive need to agree on the form of this identifier. A unique SIP ID allows checking for duplicate submissions and provides a common identifier for the Producer and Archive if they need to communicate about this submission.

**Producer Source ID**

The Producer Source ID is a mandatory attribute that uniquely identifies the sending entity within a given Producer-Archive Project. Should the Archive need to contact the sending entity of a given SIP, for example to resolve a problem, this identifier makes clear which of several possible sending entities originated that SIP. Both the Producer and the Archive need to agree on the form of this identifier.

**Producer-Archive Project ID**

The Producer-Archive Project ID is a mandatory attribute assigned by the Archive to ensure uniqueness across all such Producer-Archive Projects involving the Archive. Its presence within a SIP enables the Archive to uniquely identify the project to which a SIP belongs and thus to identify the MOT and constraints document applicable to that SIP.

**SIP Content Type ID**

The SIP Content Type ID is a mandatory attribute that enables the identification, within the constraints document, of the specification as to the types of Transfer Objects allowed in the SIP and their frequency of occurrence. Its value should be checked by the Archive upon receipt of the SIP. It also allows the Archive to check as to whether receipt of the SIP has violated any constraints on the order in which the SIPs are to be received.

**SIP Sequence Number**

The SIP Sequence Number is an optional attribute that indicates the order in which the SIPs have been sent. It must be unique within the context of a give Producer Source. When a given Producer Source is sending Transfer Objects whose Descriptor does not specify the number of Transfer Objects to be sent, this attribute becomes mandatory for all its SIPs to enable the Archive to check that it has not missed any Transfer Objects.

**Any other attributes**

The abstract SIP allows the user to define additional attributes that may be included as Global Information. There are many possible reasons to define additional attributes. Some examples might be for a Producer to track the individual who generated this SIP or to identify an agreement or contract between the Producer and the Archive under which this SIP falls.

#### Transfer Object to Delete container

This optional container holds one or more attributes as follows:

**Transfer Object to Delete ID**

The Transfer Object to Delete ID is a mandatory attribute giving the value of the Transfer Object ID for a Transfer Object that should be deleted by the Archive. Any number of such attributes may be present.

**Any other attributes**

The abstract SIP allows the user to define additional attributes that may be included in the Transfer Object to Delete container. This may include attributes such as the reason for deletion and authorization details regarding the deletion.

### Transfer Object Container

This container holds one Transfer Object Identification and Status container and one or more of Transfer Object Group containers.

#### Transfer Object Identification and Status container

This container holds a number of attributes supporting the unique identification of this Transfer Object, the identification of the associated Descriptor, and whether this Transfer Object is to replace a previously sent Transfer Object, and whether this is the last Transfer Object of this type that the Producer Source will be transferring as follows:

**Descriptor ID**

The Descriptor ID is a mandatory attribute that identifies the Transfer Object Type Descriptor that describes this Transfer Object. It is obtained from the MOT. Its presence allows the Archive to compare the received Transfer Object with its specification (type definition) as agreed between the Producer and the Archive.

**Transfer Object ID**

The Transfer Object ID is a mandatory attribute that uniquely identifies this Transfer Object within the Producer-Archive Project. The form of the identifier needs to be agreed between the Producer and the Archive. For example, it could be constructed by pre-pending the unique SIP ID to a sequence number generated by each Producer Source entity. It could also be constructed by concatenating the Producer Source ID, the Descriptor ID, and a sequence number generated by each Producer Source entity.

**Last Transfer Object Flag**

The Last Transfer Object Flag is an optional attribute indicating that this is the last Transfer Object of this type (i.e. of the associated Descriptor) that this Producer Source entity expects to send. It is particularly useful when the total number of Transfer Objects of a given type, to be sent, has not been specified in the associated Descriptor. In this case there must be a SIP Sequence Number and if there is a single Producer Source entity sending Transfer Objects, this identifier allows the Archive to determine when all of the Transfer Objects of this type have been received. However if there are multiple Producer Source entities sending SIPs, it only indicates that this Producer Source sending entity does not expect so send any additional Transfer Objects of this type. There may be another Producer Source entity that may, or may not, be sending additional Transfer Objects of this type and thus the Archive may not know when it has received all such Transfer Objects. In this case additional Producer-Archive communications or agreements will be needed.

**Replacement Transfer Object ID**

The Replacement Transfer Object ID is an optional attribute stating that the value of this attribute is the Transfer Object ID of a previously sent Transfer Object that the Archive is to replace with this Transfer Object.

**Any other attributes**

The abstract SIP allows the user to define additional attributes that may be included in the Transfer Object Identification and Status container. Additional attributes that could be added would be things like who approved the Transfer Object as being ready for archiving or information on the date the Transfer Object was last updated.

### Transfer Object Group container

This container holds any number of additional Transfer Object Group containers, any number of Data Object containers and one Transfer Object Group Identification container.

For each Transfer Object Group Type described by a Transfer Object Group Type specification within a Descriptor, there must be one or more Transfer Object Group Type containers in the SIP. Generally there will be multiple containers when there are multiple instances of the group type. However there are two special cases that are exceptions, as follows:

Case 1:

When a Transfer Object Group Type specification includes the Transfer Object Group Type Encoded attribute, and regardless of any other attributes, the abstract SIP will have a Data Object container instead of a Transfer Object Group Type container. The instantiation of the Data Object container in the actual SIP will be a single file.

Case 2:

When a Transfer Object Group Type specification does NOT include the Transfer Object Group Type Encoded attribute, but its Transfer Object Group Type Structure Name has the value ‘undescribed’, then this Transfer Object Group Type specification results in the abstract SIP having some combination of Transfer Object Group containers and Data Object containers that was not described when the Transfer Object Group was modelled. A data structure in the Producer’s environment that has been modeled in the Descriptor as a group type that is ‘undescribed’ with no encoding is to be fully instantiated in the SIP. The Transfer Object Group containers are used for directories or for holding groupings of files and the Data Object containers are used for files. The Producer must ensure that the correct data are included in the SIP for each ‘undescribed’ group. The Archive cannot use the associated Descriptor to completely verify that data as it has not been fully modeled.

#### Transfer Object Group Identification container

This container holds a number of attributes identifying the type of the group and optionally naming the group instance.

**Associated Descriptor Group Type ID**

The Associated Descriptor Group Type ID is a mandatory attribute that identifies the associated group type description within the associated Descriptor.

* When the group type description is not specifying that it is ‘undescribed’, then this identifier’s value is the Transfer Object Group Type ID of the subject Descriptor’s group type specification.
* When the group type description is specifying that it is ’undescribed’, then this same identifier value is used for the Associated Descriptor Group Type ID not only for this group container, but for all nested group containers and nested data object containers. An example of this situation is shown in Annex A titled ‘Associated Descriptor Data Identifiers.’

There is a choice of which of the next two attributes – Transfer Object Group Instance Name or Transfer Object Group Preservation Name – is used. The primary difference between the two is that if Transfer Object Group Preservation Name is used, then the Archive should maintain that name with the group.

**Transfer Object Group Instance Name**

The Transfer Object Group Instance Name is an optional attribute that may be inserted by the Producer to name the group, such as to provide a directory name or to name a set of Data Objects or other groups. If this attribute is used, the Transfer Object Group Preservation Name may not be used.

**Transfer Object Group Preservation Name**

The Transfer Object Group Preservation Name is an optional attribute that may be inserted by the Producer to name the group and to indicate to the Archive that this name needs to be preserved with the group. If this attribute is used, the Transfer Object Group Instance Name may not be used.

**Any other attributes**

The abstract SIP allows the user to define additional attributes that may be included in the Transfer Object Group Identification container.

### Data Object container

This container holds one or more Byte Stream containers and a single Data Object Identification container.

#### Data Object Identification container

This container holds one or more attributes identifying the type of Data Object and optionally supplies a name that is to be preserved in association with the byte stream or streams.

**Associated Descriptor Data ID**

The Associated Descriptor Data ID is a mandatory attribute that identifies a part of a Descriptor that is to be associated with this Data Object. The part of a Descriptor that is to be used depends on several factors as follows:

* If this Data Object is an instance of a Data Object Type defined in a Descriptor, then the Associated Descriptor Data ID should be the value of the Data Object Type ID of that Data Object Type specification.
* If this Data Object is an instance of a Transfer Object Group Type defined in the Descriptor to be encoded and thus the group becomes a single Data Object, then the Associated Descriptor Data ID should be the value of the Transfer Object Group Type ID of that Transfer Object Group Type specification.
* If this Data Object is an instance that is transferred within the context of a Descriptor-defined Transfer Object Group Type whose Transfer Object Group Type Structure Name has the value ‘undescribed’, then the Associated Descriptor Data ID should be the value of the Transfer Object Group Type ID of that Transfer Object Group Type specification.

An example is given in Annex A titled ‘Associated Descriptor Data Identifiers.’

**Data Object Preservation Name**

The Data Object Preservation Name is an optional attribute that may be inserted into the SIP by the Producer to tell the Archive that this name is to be preserved in association with this Data Object. When the Data Object is a single file, this name is in addition to the name of the file in the Transfer Object. When the Data Object is composed of multiple byte streams, or files, the name is associated with all of them. Note that the use of Data Object Preservation Name provides a second name apart from the names of the files in the Transfer Object. Therefore there is no conflict if the associated Transfer Object Descriptor has included the optional namePreservationRule element giving a rule for how the Data Object names (i.e.,file names) are to be constructed by the Producer or possibly altered and subsequently preserved by the Archive.

For example, assume each Data Object consists of a single file composed of a day’s observations of magnetic field values taken at 60 second averaged intervals and has been given a file name of the form spacecraft\_mag\_julianday.asc. The Producer may decide that it is desirable to associate a more user friendly name, such as spacecraft\_mag\_year\_day.asc, with each Data Object by inserting the Data Object Preservation Name attribute, with this value, into the SIP. The Archive would be required to preserve this name in association with the Data Object.

As another example, assume each Data Object consists of two files with different file extensions taking the forms spacecraft\_mag\_julianday.asc and spacecraft\_mag\_julianday.bin. These files contain data averaged over 60 seconds during each day. However due to gaps in the data the mid-point of a day does not always conform to the mid-point of the observations. The Producer may decide that it is desirable not only to have a single preserved name for the pair of files but to also indicate a weighted mid-point of the observations within each day by constructing a name of the form spacecraft\_mag\_year\_day\_hour. The Producer would insert this value, for the Data Object Preservation Name attribute, into the SIP. Again the Archive would be required to preserve this name in association with multi-file Data Object. Note that there is no requirement in the PAIS standard that the Data Object Preservation Name value be unique, but it is recommended that it be unique at least within the context of each Transfer Object.

**Any other attributes**

The abstract SIP allows the user to define additional attributes that may be included in the Data Object Identification container.

#### Byte stream container

This container provides a number of attributes to provide a single byte stream within the SIP, or to provide a pointer to a byte stream outside the SIP, or both. If both mechanisms are used, the Producer and Archive need to agree on the relationship between the two.

**Byte Stream**

This optional attribute is the byte stream that will be instantiated as a single file within the SIP.

**Byte Stream Checksum**

This optional attribute provides a checksum value covering the stream of bytes. Its instantiation within the SIP will also require identification of the checksum algorithm, such as MD5.

**Pointer to Byte Stream**

This optional attribute provides a mechanism to point to a byte stream outside of the SIP. For example, it may be instantiated in the SIP as a URL.

**Any other attributes**

The abstract SIP allows the user to define additional attributes that may be included in the Byte Stream container.

## XFDU SIPs

The previous sections described an abstract view of the SIP. Any SIP implementation should include an implementation of all the attributes and requirements discussed in that section. The PAIS standard proposes one implementation of the recommended SIP Model based on the CCSDS XML Formatted Data Units (XFDU) packaging standard (see reference [2]). The XFDU implementation adopts the use of standard XFDU mechanisms to convey some of the attributes and uses XFDU’s extensibility to include the remaining attributes.

As a brief introduction to the most relevant features of this standard, an XFDU package is a container, usually a ZIP archive file, composed of one Manifest XML file referencing all the other files of the package. The Manifest file consists of a series of sections among which are a Package Header containing general information, an Information Package Map providing a logical break-down of the package content in a tree of nested Content Units, and an XFDU Data Object Section referencing all the packaged files.

The XFDU PAIS SIP definition makes use of the XFDU extendibility mechanisms to implement the SIP Model introduced in the previous section. This implementation is depicted in Figure 5-2 and is completely defined in the PAIS standard (see section 6 of reference [1]). The XFDU standard (see Reference[2]) defines the base XFDU structures. The PAIS standard provides a number of rules for how and where the abstract SIP items appear in an XFDU instance (see Section 6 in Reference[6]) and it defines several XML schema snippets (see Annex E in Reference[6]) that fully define the additional XML elements that allow for the inclusion of PAIS information within the XFDU document.



Figure 5‑2: Combination of XFDU Schema and PAIS Schema to Form XFDU SIP Schema

This resulting extended XFDU implementation is depicted in Figure 5-3 below.

Description: 651x2g0-figure-2-7.emf

Figure ‑: XFDU PAIS SIP

The SIP Global Information is implemented as an extension of the Package Header section of the XFDU Manifest.

The Transfer Objects to Delete are implemented as an extension of a Content Unit of the Information Package Map Section of the XFDU Manifest.

The Transfer Objects and their Transfer Object Groups and Data Objects children are also implemented as extensions of Content Units of the Information Package Map Section. However, because the XFDU does not authorize the direct references to the packaged files, a Data Object Content Unit makes use of XFDU defined Data Object Pointers to reference entries into the Data Object Section of the XFDU Manifest. The Byte Streams are then referenced from this latter Data Object Section. When a PAIS SIP Data Object is composed of multiple Byte Streams, the XFDU Content Unit implementing that Data Object will have multiple XFDU Data Object Pointers referencing multiple XFDU Data Objects in the XFDU Data Object Section. Thus there is not always a one-to-one mapping between PAIS defined Data Objects and XFDU defined Data Objects. The term ‘Data Object’ normally refers to PAIS defined Data Objects unless the context makes clear that the XFDU Data Object is intended.

A SIP that did not transfer any information would not be very useful, so the PAIS standard requires that at least one Data Object is required within a SIP.

When the Producer creates the SIP instance, it includes a number of items that allow the Archive to verify the SIP constructs against the MOT, i.e. the agreement between the Producer and the Archive about what is being transferred. The major items that allows these checks to be made are the inclusion of IDs in the SIP that point back to definitions of those constructs in the MOT. For example the SIP TranferObject implementation includes the descriptorID that links back to the definition of that Transfer Object Type. Other examples are Transfer Object Groups containing the associatedDescriptorGroupTypeID element and Data Objects containing the associatedDescriptorDataObjectTypeID element.

As a reminder, the Producer is able to specify instance names for Transfer Object Groups (directories) (the transferObjectGroupInstanceName or the transferObjectGroupPreservationName) and Data Objects (files) (the dataObjectInstanceName) as part of the SIP. The names used may be the actual names of the directories and files in the Producer’s data environment but other names may be provided if desired.

The XFDU standard includes many features that have not been adopted for the SIP/XFDU implementation. The primary role for the XFDU is an implementation of the abstract SIP to allow the various PAIS defined constructs to be found and compared with their definitions in the MOT. The PAIS puts few explicit restrictions on use of the XFDU features. One of these restrictions is the limit of the XFDU byteStream element to a single fileLocation element.

The Producer and Archive may make a separate agreement to make use of other XFDU features to include information not required by the implementation of the abstract SIP. While this is primarily outside the scope of this Green Book, one feature of note is the ability of the XFDU to include a mime type value in association with an XFDU Data Object and/or in association with an XFDU byteStream element. Such a mime type value must not be in conflict with the associated Descriptor defined format information or mime type value.

Practical examples of XFDU PAIS SIP implementations are provided and discussed in section 6 below.

### Linkage between descriptor IDs and SIP

## Non-XFDU SIPs

Within the PAIS standard two levels of PAIS conformance are identified. One level of conformance makes use of the PAIS XFDU implementation defined in that standard. Another level of conformance, the abstract level of PAIS conformance, allows for development of non-XFDU implementations of PAIS that still fulfill all the requirements for information sharing and for automation of the Producer to Archive information transfer.

In the previous subsections of this document the XFDU implementation has been discussed. While the XFDU implementation is useful in many domains, there are also many other domains and communities that primarily make use of general packaging mechanisms other than XFDU. Within those domains and communities it may be worthwhile considering developing or making use of a PAIS implementation in the communities dominant packaging mechanism that conforms to the abstract PAIS.

For communities with an existing non-XFDU dominant packaging mechanism, it may be better to use that mechanism for a PAIS implementation to gain greater PAIS adoption. It is also possible that this non-XFDU implementation of PAIS could be standardized within that community. The CCSDS participants who developed the PAIS standard may be able to support those efforts or they may be able to review what that community develops.

If a decision is reached to develop a non-XFDU implementation of PAIS, we recommend that the community should try make use of the native features of their chosen packaging mechanism to represent the required PAIS information rather than just tacking on a separate PAIS module.

Another point should be made that once a non-XFDU implementation of PAIS is developed, it may be possible to develop associated tools that would support conversions between the XFDU PAIS implementation and the non-XFDU implementation. This is particularly true if the non-XFDU PAIS implementation is an XML-based implementation. In that case, it is likely that an XSL transformation between the two implementations could quickly be developed.

An example of a non-XFDU implementation is provided in Section 6 where a Metadata and Encoding Transition Standard (METS) implementation of PAIS is presented. The METS implementation was developed by the French National library (BnF) in consultation with the CCSDS PAIS development team. METS is a widely used standard for transmission of metadata and packaging information within the cultural heritage and library communities.

The BnF developed this METS implementation to study the applicability of PAIS to their environment. BnF is not currently using PAIS in its Scalable Preservation and Archiving Repository (SPAR). Nonetheless, PAIS could be profitably implemented in the case of complex and predictable transfer objects in the future.

# Use Cases

This section registers a series of use cases that were generally elaborated during the development and validation of the PAIS standard. These examples do not claim to provide turnkey solutions for operations. They may, however, improve the understanding of the PAIS standard through concrete cases built from various aspects. They may also help implementers start up their projects with patterns and snippets they can arrange at their discretion.

## ISEE – A Typical Use Case

**[NOTE – Section temporarily moved to 651x2g0-[6.1]-isee.docx document]**

## ESA-SAFE – Transfer of SAFE products

**[NOTE – Section temporarily moved to 651x2g0-[6.2]-polder.docx document]**

## CoRoT – End of Mission Bulk Transfer

**[NOTE – Section temporarily moved to 651x2g0-[6.4]-corot.docx document]**

## BnF and METS – A non-XFDU SIP Implementation

**[NOTE – Section temporarily moved to 651x2g0-[6.3]-pds4.docx document]**

# Software Tools

This section describes the two prototypes developed as part of the CCSDS PAIS standard [1] and may not be considered as operational tools. That’s why each software is delivered as is, without maintenance. The Descriptors, data, SIPs, SIP constraints, are embedded in the CNES prototype.

## CNES Prototype

The CNES prototype is available at the following address: https://logiciels.cnes.fr.

It is a java implementation that conforms to the CCSDS Producer Archive Specification (PAIS) recommended practice based on the PAIS XML schemas, and uses other components such as a Derby database for the management of the Data Objects received by the Archive.

This software is governed by the CeCILL-C license under French law and abiding by the rules of distribution of free software. It uses the dataChecker library under the term of EPL license, and the jgraph library under the term of LGPL license.

The CNES implementation was developed to cover the following functions during the following main phases of a Producer-Archive Project:

* During the Formal Definition Phase:
* MOT creation and visualization:
* MOT structure design;
* Descriptors instantiation and validation with XSD Models, or direct import of Descriptors received from the Producer;
* MOT visualization (see figure 4-1, and MOT figures provided with each practical cases section 6), using an easy GUI;
* SIP specification: SIP content definition, and sequencing constraints between SIPs;
* During the Transfer and Validation Phases (on the Archive side):
* State of the delivered Transfer Objects in the MOT and detailed follow-up (the number of delivered objects can be seen on the same window, instead of the occurrence);
* Using the same graphical visualization;
* Validations performed on the delivered SIPs (integrity, Data Objects occurrences, SIP contents, SIP sequencing constraints). They are applied on the XFDU manifest (not on the attached data).

It is supposed to be installed on the Archive side.

It is implemented in the JAVA programming language. It includes the Open Sources Xample (XML form) and JGraph (graphical design of the MOT).

## ESA SIP Builder

The SIP Builder is available at the following address: <https://github.com/gael-systems/ccsds-sip-builder>. It is under LGPL v3 license, described in the README.txt file, available along with the tool.

The ESA prototype, also called “ESA SIP Builder”, is a Java command line software dedicated to the generation of SIPs according to the PAIS XFDU SIP Model specified in section 6 of the PAIS Draft Recommended Standard (see reference [1]). The following diagram identifies the main PAIS elements that are critical for the ESA SIP Builder software.

Description: C:\Users\Mbaye\Documents\Professionnel\P288-CNES-SIP-Builder-ADD-SUM\working-document\GAEL-P288-SUM-001-01-Static.emf

*SIP Constraints*

*Transfer Object Descriptors*

Figure 7‑1: Identification of PAIS elements used by the ESA SIP Builder

On top of these standard PAIS definitions, the ESA SIP Builder introduces the concept of so called Collectors responsible for the supply of the actual data files and folders to be conveyed by the output SIPs.

Description: C:\Users\Mbaye\Documents\Professionnel\P288-CNES-SIP-Builder-ADD-SUM\working-document\GAEL-P288-SUM-001-02-Static-Collectors.emf

*Model of Objects for Transfer*

*SIP Builder Project*

*File System Repository*

Figure 7‑2: File and Folder Collectors

The ESA SIP Builder Collectors are attached to a target Group Type or a Data Object Type and hold one or more inclusion or exclusion patterns (file naming rules) that select files and folders in a given file system repository.

The ESA SIP Builder processes the input descriptors and collectors in an internal in-memory tree of objects before any conversion from-to the disk as XFDU packages. For disambiguation with PAIS and XFDU elements or objects, the internal in-memory objects are further prefixed with the term "Logical" e.g. Logical SIP composed of Logical Groups themselves composed of Logical Data Object.

The main steps of the ESA SIP Builder processing are the following:

– **Read and validate** the Collection Descriptors and Transfer Object Descriptors

– **Attach Collectors** to the descriptors Group Types and Data Objects Types

– **Select the SIP Types** to be produced from the SIP Constraints

– **Sort the SIP Types** from the SIP Constraints, if any

– **For each SIP Type**:

**For each authorized descriptor** of the SIP Type:

**For each Group Type** and sub-Group Type of the descriptor:

**Run the attached Collector**, if any

For each collected folder or once if none:

**For each Data Object Type**:

**Run the attached Collector** considering the current Group Type instance name as context location

For each collected file:

**Create a Logical Data Object** of the current Data Object Type

**Create as many Logical Groups as necessary** to hold the created Logical Data Objects or intermediate Logical Groups

**Create as many Logical Transfer Objects as necessary** to hold the created Logical Groups

**Create as many Logical SIPs as necessary** to hold the created **Logical Transfer Objects**

– **Validate the Logical SIPs** against descriptor constraints and limits e.g. occurrences, sizes, etc.

– **Convert the Logical SIPs** to definitive **XFDU** packages

A special attention has to be paid for the three "**as many as**" used in the above processing tree. The general strategy is a top-down analysis of the PAIS elements from the SIP Types to the Data Object Type leaves. However, the ESA SIP Builder Collectors may select more folders or files than allowed for a given Data Object Type or a Group Type cf. maxOccurrence usage in PAIS descriptors and constraints file. The ESA SIP Builder considers that extra instances of the parent Group Types, Transfer Object Types or even SIP Types need to be created. The use of maxOccurrence's is even the only means for building multiple instances of Group Types or Data Object Types in the current implementation. It is not said that this is the best strategy but it seemed reasonable so far, limiting the complexity of configuration to simple patterns. Future improvements could be considered by implementing less "passive" Collectors capable of, for example, expressing grouping rules or more complex mapping.

The following practical example is probably more efficient than any further theory. It derives from a test case elaborated by NASA from data acquired by the International Sun-Earth Explorer (ISEE) first two spacecrafts.

The input data set of about 2 Gb has the following structure with two main directories isee1 and isee2 that share the same layout. The data are broken down in individual directories matching the year of acquisition. Each data file (.asc-gz) corresponds to a day of acquisition and has an associated metadata file (.asc-gz\_att) in an attrib sub directory.

.

├── isee1

│   └── mag

│   └── 60\_sec\_ascii\_gz

│   ├── 1977

│   │   ├── attrib

│   │   │   ├── isee1\_mag\_60s\_0001\_1977\_295.asc-gz\_att

│   │   │   ├── isee1\_mag\_60s\_0002\_1977\_298.asc-gz\_att

│   │   │   ├── isee1\_mag\_60s\_0003\_1977\_300.asc-gz\_att

│   │   │   ├── isee1\_mag\_60s\_0004\_1977\_302.asc-gz\_att

│   │   │   └── ...

│   │   ├── isee1\_mag\_60s\_0001\_1977\_295.asc-gz

│   │   ├── isee1\_mag\_60s\_0002\_1977\_298.asc-gz

│   │   ├── isee1\_mag\_60s\_0003\_1977\_300.asc-gz

│   │   ├── isee1\_mag\_60s\_0004\_1977\_302.asc-gz

│   │   └── ...

│   ├── 1978

│   │   └── ...

│   ├── ...

│   └── 1987

│      └── ...

└── isee2

   └── mag

   └── 60\_sec\_ascii\_gz

   ├── 1979

   │   ├── attrib

   │   │   ├── isee2\_mag\_60s\_0001\_1977\_295.asc-gz\_att

   │   │   └── ...

   │   ├── isee2\_mag\_60s\_0001\_1977\_295.asc-gz

   │   └── ...

   ├── ...

   └── 1987

      └── ...

In order to simulate an archiving project of this data set, NASA has designed a series of PAIS descriptors and SIP constraints files. The detail about these files is out of the scope of the present document but the main elements relevant for the present illustration are summarized in the following diagram.

Description: C:\Users\Mbaye\Documents\Professionnel\P288-CNES-SIP-Builder-ADD-SUM\working-document\GAEL-P288-SUM-001-03-TC1-Static.emf

Figure 7‑3: Example of Collectors supplying TC1 Groups and Data Objects

The descriptors define, among others, a Transfer Object Type identified by ISEE\_...\_Metadata\_TC1. This latter is described has composed of a Satellite\_Group Group Type, itself composed of a Yearly\_Group Group Type that accepts a NSSDC\_Metadata\_ISEE\_Mag\_Data\_File Data Object Type.

This tree of types matches quite well the layout of the data set described earlier where the Satelite\_Group would correspond to the isee1 and isee2 directories, the Yearly\_Group to the 1977, 1978, ... and 1987 directories and finally the NSSDC\_...\_Mag\_Data\_File would represent the metadata files in the attrib sub-directories. Three collectors are necessary to express this mapping to the ESA SIP Builder. They are represented in Yellow in the above diagram.

You may notice that different patterns could have produced the same results. For example, the pattern "isee[12]" could have been replaced by "isee." (any trailing character in regular expression) because only isee1 and isee2 are present in the input data set. Similarly, the "mag" prefix for the Yearly\_Group filters nothing as far as there is no other directory than the mag one for any spacecraft.

More specifically, you may notice that some collecting patterns includes some constraints that limit the file selection to a subset of what is actually available in the input data set. For example the collector attached to the Yearly\_Group type selects only the files under the 1977 directory. Similarly, the ".\*(29[5-9]|300).asc-gz\_att" of the Data Object Type collector filters only those files acquired between the 295th and the 300th days of the year, included. These constraints are derived from the descriptors documentation e.g. a transferObjectTypeDescription element was containing "Selected ISEE 1,2 Magnetic\_Field Metadata grouped by Spacecraft and then by Yearly Directories (1977 only) for days 295-300 inclusive." and thus required specific collector patterns for building SIPs with the right content.

The ESA SIP Builder interpretation of the input descriptors and collection patterns for this example can be summarized by the following diagram where a Transfer Object, on the left, will be generated with two sub-groups, one for each satellite, each containing a single year group containing three files. The tree on the right correspond to the input data set.

Description: C:\Users\Mbaye\Documents\Professionnel\P288-CNES-SIP-Builder-ADD-SUM\working-document\GAEL-P288-SUM-001-04-TC1-Dynamic.emf

*Type Instances*

*File System Repository*

*Collectors*

Figure 7‑4: Collectors mapping from test data to SIP Groups and Data Objects

The configuration of the ESA SIP Builder is provided through a so called project file which is an XML document.

1. Associated Descriptor Data Identifiers

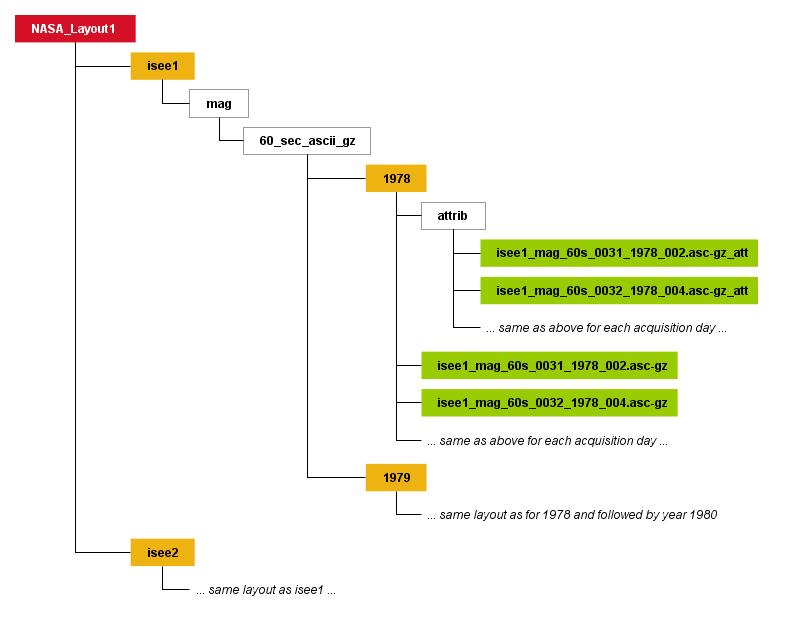
Under section 5.1.4.1 (Data Object Identification container), one of the required attributes is Associated Descriptor Data ID. This attribute is used to identify that part of a Descriptor that is to be associated with a given Data Object. It takes on three different types of identification values depending on several factors as follows:

* Type A: If this Data Object is an instance of a Data Object Type defined in a Descriptor, then its Associated Descriptor Data ID should be the value of the Data Object Type ID of that Data Object Type specification.
* Type B: If this Data Object is an instance of a Transfer Object Group Type defined in the Descriptor to be encoded and thus the group becomes a single Data Object, then its Associated Descriptor Data ID should be the value of the Transfer Object Group Type ID of that Transfer Object Group Type specification. This will be the case even when the Group Type is also defined to be undescribed.
* Type C: If this Data Object is an instance that is transferred within the context of a Descriptor-defined Transfer Object Group Type whose Transfer Object Group Type Structure Name has the value undescribed, then its Associated Descriptor Data ID should be the value of the Transfer Object Group Type ID of that Transfer Object Group Type specification.

This annex provides examples of these three types of Associated Descriptor Data ID values. The first example is for Type A defined above.

**Type A**

The data structure for this example is taken from the ISEE use case of section 6.1 and its data layout is reproduced here as Figure A-1.

Figure A-1: ISEE1/ISEE 2 Data Repository Layout

For the purpose of highlighting the use of the Type A Associated Descriptor Data ID in the abstract SIP and in a resulting XFDU SIP, a simple Transfer Object Descriptor is sufficient. It is assumed that the Producer will be submitting two ISEE 1 files from 1978 to an Archive. A possible resulting Transfer Object Descriptor, in table form, is specified as follows:

Table A-1: Example of a Transfer Object Type Descriptor for Single File

|  |  |
| --- | --- |
| **Element** | **Sample Value** |
| **transferObjectTypeDescriptor ➊** |  |
| identification |  |
| descriptorModelID | CCSD0014 |
| descriptorModelVersion | V1.0 |
| descriptorID | ISEE\_1978\_mag\_data |
| description |  |
| transferObjectTypeTitle | ISEE 1978 Demo Example 1 |
| transferObjectTypeDescription | Demonstrate Associated Descriptor Data ID for Type A situation |
| transferObjectTypeOccurrence |  |
| minOccurrence | 1 |
| maxOccurrence | 2 |
| relation |  |
| parentCollection | Demo 1 Parent Collection |
| groupType |  |
| groupTypeID | Yearly\_Group |
| groupTypeStructureName | directory |
| groupTypeOccurrence |  |
| minOccurrence | 1 |
| maxOccurrence | 1 |
| dataObjectType |  |
| dataObjectTypeID | ISEE\_1978\_Mag\_60s |
| dataObjectTypeOccurrence |  |
| minOccurrence | 1 |
| maxOccurrence | 1 |

This specifies that there can be at most 2 Transfer Objects and each consists of a directory with a single file (see dataObjectTypeOccurrence above).

Assuming that the SIP Constraints specify that only one Transfer Object of the above type may be included in a SIP, a possible resulting abstract SIP is as follows:

Table A-2: Example of an Abstract SIP for Table A-1 Descriptor

|  |  |
| --- | --- |
| **Containers and Attributes** | **Example Attribute Values** |
| **SIP containter➊** |  |
| SIP Global Information container |  |
| *SIP ID* | SIP-Demo-1\_01 |
| *Producer Source ID* | Demo\_1\_Producer |
| *Producer-Archive Project ID* | Demo\_1\_Project |
| *SIP Content Type ID* | SIP\_Demo\_1\_Single |
| Transfer Object Container |  |
| Transfer Object Identification and Status container |  |
| *Descriptor ID* | **ISEE\_1978\_mag\_data** |
| *Transfer Object ID* | Demo\_1\_TO\_01 |
| Transfer Object Group container |  |
| Transfer Object Group Identification container |  |
| *Associated Descriptor Group Type ID* | **Yearly\_Group** |
| *Transfer Object Group Instance Name* | 1978 |
| Data Object container |  |
| Data Object Identification container |  |
| *Associated Descriptor Data ID* | **ISEE\_1978\_Mag\_60s** |
| Byte Stream ontainer |  |
| *Byte Stream* | PGTesv^&895…etc. |

The attribute values in a bold font are those that reference values from the associated Descriptor. The SIP begins with a container for the SIP Global Information and follows with containers for Transfer Objects of which there is only one because in this scenario it has been assumed that the SIP constraint require only one Transfer Object per SIP. It references the associated Descriptor through the Descriptor ID with a value of **ISEE\_1978\_mag\_data.** There must be at least one Transfer Object Group container and the Descriptor specifies it to be a directory. The abstract SIP references this Group container through the Associated Group Type ID value of **Yearly\_Group**. The Transfer Group Instance Name is given as 1978 and this is one of the directory names shown in Figure A-1. There is one Data Object container as there is only one file being transferred. It references the relevant Descriptor information by the attribute Associated Descriptor Data ID with a value of **ISEE\_1978\_Mag\_60s.**

This abstract SIP can be implemented using the PAIS specified XFDU packaging mechanism. Most of the containers are implemented as XFDU content units using its extension capability. The exception is the Byte Stream container because the XFDU has its own Byte Stream mechanism. The resulting SIP will not be further detailed as it is not releant to the objective of this Annex. However see section 6.1 and its relevant annex for a more complete example of ISEE Descriptors, SIP Constraints, and an XFDU SIP.

**Type B**

The Type B case occurs when a Descriptor has a group type that is encoded. The example Descriptor is given in Table A-3 below. It is identical to the Descriptor in Table A-1 except many elements have been removed for brevity and the element groupTypeEncoded has been inserted. It states that the group will be incoded into a single tar file. Thus the file will appear as a Data Object in the SIP.

Table A-3: Example of a Transfer Object Type Descriptor for Encoded Group

|  |  |
| --- | --- |
| **Element** | **Sample Value** |
| **transferObjectTypeDescriptor ➊** |  |
| …removed for brevity… |  |
| groupType |  |
| groupTypeID | Yearly\_Group |
| groupTypeStructureName | directory |
| groupTypeEncoded |  |
| encodingName | tar |
| encodingDescription | application/x-tar |
| groupTypeOccurrence |  |
| minOccurrence | 1 |
| maxOccurrence | 1 |
| dataObjectType |  |
| dataObjectTypeID | ISEE\_1978\_Mag\_60s |
| dataObjectTypeOccurrence |  |
| minOccurrence | 1 |
| maxOccurrence | 1 |

The resulting abstract SIP would look as follows:

Table A-4: Example of an Abstract SIP for Table A-3 Descriptor

|  |  |
| --- | --- |
| **Containers and Attributes** | **Example Attribute Values** |
| **SIP containter➊** |  |
| SIP Global Information container |  |
| *…removed for brevity* |  |
| Transfer Object Container |  |
| Transfer Object Identification and Status container |  |
| *Descriptor ID* | **ISEE\_1978\_mag\_data** |
| *Transfer Object ID* | Demo\_1\_TO\_01 |
| Data Object container |  |
| Data Object Identification container |  |
| *Associated Descriptor Data ID* | **Yearly\_Group** |
| Byte Stream ontainer |  |
| *Byte Stream* | Rtvexig\*345… etc. |

Note that the abstract SIP’s Transfer Object container no longer has a group container as this has been replaced by a Data Object container. The section of the Descriptor that is referenced by its Associated Descriptor Data ID is Yearly\_Group, which the specification for the Transfer Object Group Type. This links the tar file with its specification in the Descriptor. The associated byte stream of the Data Object container will be the tar file.

**Type C**

The Type C case occurs when a Descriptor has a group type that is undescribed. The example Descriptor is given in Table A-5 below.

Table A-5: Example of a Transfer Object Type Descriptor for undescribed Group

|  |  |
| --- | --- |
| **Element** | **Sample Value** |
| **transferObjectTypeDescriptor ➊** |  |
| …removed for brevity… |  |
| groupType |  |
| groupTypeID | Yearly\_Group |
| groupTypeStructureName | undescribed |

It is identical to the Descriptor in Table A-1 up to the Transfer Object Group Type, except many elements have been removed for brevity. The groupTypeStructueName is given as undescribed. This means that the Group Type and all data under that Group Type, regardless of complexity, has not been modeled by this Descriptor. Other elements, except for dataObjectType and groupType, may be in the undescribed groupType. Nevertheless, the Producer must still package directories as groups and data files as Data Objects. For this example, the Producer must package the directory ‘1978’ as a Tranfer Object Group container and a single file as a Data Object container. The resulting abstract SIP would look as follows:

Table A-6: Example of an Abstract SIP for Table A-5 Descriptor

|  |  |
| --- | --- |
| **Containers and Attributes** | **Example Attribute Values** |
| **SIP containter➊** |  |
| SIP Global Information container |  |
| *… removed for brevity…* |  |
| Transfer Object Container |  |
| Transfer Object Identification and Status container |  |
| *Descriptor ID* | **ISEE\_1978\_mag\_data** |
| *Transfer Object ID* | Demo\_1\_TO\_01 |
| Transfer Object Group container |  |
| Transfer Object Group Identification container |  |
| *Associated Descriptor Group Type ID* | **Yearly\_Group** |
| *Transfer Object Group Instance Name* | 1978 |
| Data Object container |  |
| Data Object Identification container |  |
| *Associated Descriptor Data ID* | **Yearly\_Group** |
| Byte Stream ontainer |  |
| *Byte Stream* | PGTesv^&895…etc. |

This abstract SIP looks like the abstract SIP of Table A-2 with the exception that the Associated Descriptor Data ID points to the same section of the Descriptor as does the Associated Descriptor Group Type ID. All that is known, from this modeling information, is that the Data Object container is associated with the Yearly\_Group part of the Descriptor. When a Descriptor employs a groupTypeStructureName of undescribed, it is good practice for the details of the undescribed data to be provided to the Archive in other documentation.

1. ISEE Use Case Descriptors

**[NOTE – Annex temporarily moved to 651x2g0-[6.1]-isee.docx document]**

1. ESA-SAFE use Case Descriptors

**[NOTE – Annex temporarily moved to 651x2g0-[6.1]-isee.docx document]**

1. ESA-SAFE use Case Descriptors
2. BnF use Case Descriptors
3. COROT use Case Descriptors

[SCOPE – at least all PAIS XML element shall be indexed]